Geração e Enumeração de Extensões Lineares de Conjuntos Parcialmente Ordenados

Andréa Werneck Richa

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovado por :

Jayme Luiz Szwarcfiter, Ph.D.
Presidente

Arnaldo Mandel, Ph.D.

Lilian Markenzon, D.Sc.

Valmir Carneiro Barbosa, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 1992

RICHA, ANDRÉA WERNECK

Geração e Enumeração de Extensões Lineares de Conjuntos Parcialmente Ordenados. Rio de Janeiro, 1992.

viii, 166 p., 29,7cm (UFRJ/COPPE, M.Sc., Engenharia de Sistemas e Computação, 1992)

Tese – Universidade Federal do Rio de Janeiro, COPPE.

1 – Conjuntos Parcialmente Ordenados 2 – Algoritmos/Combinatória

I. COPPE/UFRJ II. Título (série)

Aos meus pais
e a Guilherme, na esperança de que um dia
entenda minha realização por meio deste trabalho,

Agradecimentos

Ao professor (e amigo) Jayme Luiz Szwarcfiter, por sua valiosa orientação e pelo apoio e incetivo recebidos não só durante a elaboração desta tese.

A professora Lilian Markenzon, quem primeiro me "iniciou nas artes" dos algoritmos e da combinatória.

Ao professor Arnaldo Mandel, pelo interesse demonstrado pelo meu trabalho.

Aos meus pais, pelo incentivo, apoio e correta orientação que me permitiram atingir o estágio atual. Em particular à minha mãe, a quem pude sempre tomar como exemplo.

A todos aqueles, em especial aos colegas de trabalho e da COPPE, que de alguma forma contribuiram para a elaboração desta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Geração e Enumeração de Extensões Lineares de Conjuntos Parcialmente Ordenados

Andréa Werneck Richa

Fevereiro, 1992

Orientador : Jayme Luiz Szwarcfiter
Programa : Engenharia de Sistemas e Computação

Resumo

Este trabalho se concentra na análise dos problemas de geração e enumeração de extensões lineares de conjuntos parcialmente ordenados (posets), ambos fortemente relacionados entre si. O primeiro trata da questão de se gerar (listas) efetivamente todas as extensões lineares de um poset. Este problema foi resolvido de maneira ótima (em tempo médio constante) para o caso geral, mas continua em aberto quando exigimos que cada duas extensões lineares listadas sucessivamente difiram entre si por uma transposição, ou quando exigimos que esta geração seja feita em ordem lexicográfica. Algumas subclasses de posets que admitem uma geração por transposição são caracterizadas e um algoritmo de tempo médio constante é descrito para a classe dos posets graduados. O problema da geração em ordem lexicográfica para posets série-paralelos é resolvido de maneira ótima. Um teorema original mostra que todo poset admite uma geração por inserção para a esquerda e uma por inserção para a direita. O problema de enumeração, recentemente provado sei #P-completo no caso geral, simplesmente conta as extensões lineares de um dado poset. Esta tese discute ainda alguns algoritmos (não-polinomiais) para o caso geral, assim como algumas subclasses de posets onde o problema de enumeração foi resolvido em tempo polinomial.

Abstract of the Thesis presented to COPPE/UFRJ as partia1 fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Generation and Enurneration of Linear Extensions of Partially Ordered Sets

Andréa Werneck Richa

February, 1992

Thesis Supervisor : Jayrne Luiz Szwarcfiter
Department : Computing and Systerns Engineering

Abstract

This work is mainly concerned with the problems of generating and enumerating linear extensions of partially ordered sets (posets), which are strongly related to each other. The former deals with the question of effectively generating (listing) all the Linear extensions of a poset. An optimum solution was found for the general case (in constant average time). However it remains an open problem if it is demanded that each two successive linear extensions in the generation differ by a single transposition, or if the generation is demanded to be in lexicographical order. Some subclasses of posets that have a generation by transposition are characterized. A constant average time algorithm for the ranked posets is then given. The lexicographical order generation is solved in an optimum way for sesies-parallel posets. A new theorem is presented and it states that every poset has at least one generation by left insertion and at least one by right insertion. The enumeration problem, which has been recently proved to be #P-complete in general, counts all the linear extensions of a certain poset. This thesis also discusses some (non-polynomial) algorithms for the general case, as well as some subclasses for which this problem was solved in polynomial time.

Índice

Capitulo I

Introdução

O interesse por conjuntos parcialmente ordenados finitos (ou posets, para abreviar) e suas propriedades combinatórias tem crescido nos últimos anos. Outras estruturas matemáticas, como grafos, grupos e reticulados, têm sido utilizadas como ferramentas para o estudo destes conjuntos, invertendo-se os papéis. Os resultados obtidos justificam esta abordagem, uma vez que o desenvolvimento da teoria de conjuntos parcialmente ordenados ajudou a esclarecer certos aspectos de vários problemas combinatórios.

Muitos objetos combinatórios podem ser representados como permutações que obedecem a certas restrições. O conjunto de extensões lineares de um poset pode ser interpretado como um conjunto de permutações de seus elementos que respeitam as relações de precedência inerentes ao poset. Este conjunto é de grande interesse combinatório uma vez que, dependendo do poset, o conjunto de suas extensões pode ser identificado com permutações (irrestritas), permutações alternadas, tableaux de Young, permutações em multiconjuntos, etc. Informação geral sobre posets pode ser encontrada em Aigner [Aig79], Knuth [Knu68], Stanley [Sta86] ou Trotter [Tro83].

Este trabalho se propõe a abordar dois problemas envolvendo extensões lineares de posets. O número de extensões lineares de um poset pode varias em um intervalo que vai de 1 (todo poset admite pelo menos uma extensão linear) a $n!$, se n é o número de elementos deste poset. Dado um poset $\mathcal{P}$ duas questões surgem naturalmente. A *questão da geração* pergunta se as extensões lineares de $\mathcal{P}$ podem ser otimamente geradas ou ainda se podem ser geradas de tal forma segundo algum critério de sistematização definido. A *questão da enumeração* pergunta se o número destas extensões

pode ser eficientemente determinado.

O primeiro problema considerado é o problema da geração. O principal desafio para tal geração é conseguir executá-la de modo ótimo (a menos de constantes), o que foi provado ser possível no resultado recente de Pruesse & Ruskey [PR91]. Em contrapartida, o outro problema abordado, o da enumeração, foi provado também recentemente ser #P-completo, ou seja, provavelmente intratável. É interessante notar que estes dois resultados foram obtidos quase simultaneamente, o relativo ao problema da enumeração precedendo o outro. Estes problemas se encontravam em aberto há algumas décadas. Este é o primeiro caso onde se conhece um algoritmo de tempo médio constante para a geração de uma classe de objetos combinatórios cujo problema de enumeracão correspondente foi provado ser #P-completo.

Existe na literatura um bom número de algoritmos para se gerar as extensões de um poset arbitrário anteriores ao de Pruesse & Ruskey. Wells [Wel71], Kalvin e Varol [KV83] e Knuth e Çzwarcfiter [KS74] apresentam algoritmos que utilizam *backtracking*, enquanto Varol e Rottem [VR81] apresentam uma versão não-recursiva do mesmo. Discutiremos estes algoritmos no capítulo III. Em Knuth [Knu79], ainda encontramos um algoritmo que gera a função inversa das extensões lineares (permutações admissíveis) de um poset do tipo floresta em ordem lexicográfica. Com relação ao problema de enumeração, podemos citar os algoritmos de Wells [Wel71] e Atkinson [Atk85] como soluções ineficientes para o problema, enquanto que uma pequena adaptação do algoritmo de geração de Pruesse & Ruskey nos dá o algoritmo mais eficiente, apesar de exponencial, conhecido para este problema.

O teorema 5 mostra um resultado original acerca da geração por inserção das extensões de um poset. O algoritmo de tempo médio constante para a geração das extensões de um poset série-paralelo em ordem lexicográfica, a ser apresentado no capítulo V, também constitui material original.

No capítulo II, introduzimos toda a notação e conceitos a serem utilizados no decorrer dos outros capítulos. Nos capítulos III, IV, V e VI trataremos do problema de geração das extensões lineares de posets. No capítulo III, abordaremos o problema de geração de extensões em geral, dando um breve histórico do mesmo e considerando o caso particular da geração de permutações (irrestritas). O problema de geração de todas as permutações de um conjunto de n elementos é naturalmente resolvido por meio de transposições adjacentes, como mostra o clássico algoritmo de Steinhaus

[Ste64], Johnson [Joh63] e Trotter [Tro62]. Ainda neste capítulo apresentaremos um resultado novo sobre a geração por inserção. No capítulo IV, consideraremos as gerações por transposição e transposição adjacente, caracterizando os posets que as admitem (por exemplo, subclasses de posets floresta, de multiconjuntos, ...) e descrevendo um algoritmo de tempo médio constante para a geração por transposição de uma subclasse dos posets graduados. O capítulo V apresenta um algoritmo de tempo médio constante que resolve a geração das extensões de um poset série-paralelo em ordem lexicográfica. Não se sabe de outro algoritmo na literatura sobre posets, a não ser no caso de permutações irrestritas, que considere a ordem lexicográfica para a geração das extensões. O último capítulo sobre o problema de geração descreve o recente algoritmo desenvolvido por Pruesse & Ruskey [PR91], que resolve de maneira ótima, porém não tanto sistemática, este problema.

O capítulo VII apresenta o problema de enumeração de extensões lineares de um poset. Este capítulo se encontra dividido em várias seções. A primeira delas se preocupa em apresentar o resultado obtido por Brightwell & Winkler [BW90], que mostra a iinprobabilidade de que se consiga contar o número de extensões de um poset arbitrário de maneira eficiente; a segunda mostra algoritmos (não polinomiais) para este problema; enquanto a terceira e última seção mostra algumas classes de posets onde o problema da enumeração está resolvido, como é o caso das florestas (e multiconjuntos), dos posets série-paralelos e dos posets de largura k.

Capítulo II

Preliminares

II.1 Posets

Um conjunto parcialmente ordenado, ou *poset* para abreviar, $\mathcal{P}(S, \mathrm{R})$ é um conjunto não-vazio S ao qual é associada a relação binária parcial R, denominada *ordem parcial* de S. Podemos usar indistintamente x $\preceq_{\mathcal{P}}$ y (ou simplesmente x $\preceq$ y) para denotar xRy. Uma relação binária parcial possui as seguintes propriedades :

(i)	se $x \preceq y$ e $y \preceq x \Rightarrow x = y$	(antissimétrica)
(ii)	se $x \preceq y$ e $y \preceq z \Rightarrow x \preceq z$	(transitiva)
(iii)	$x \preceq x$	(reflexiva)

(A relação de igualdade é interpretada da maneira usual). Como a propriedade de reflexividade não tem significado para o problema de ordenação topológica, usamos x $\prec$ y toda vez que x $\preceq y$ e x $\neq$ y (e dizemos que x *precede* y na ordenação parcial). Seguem as propriedades abaixo :

(i)	se $x \prec y \Rightarrow y \not\prec x$	(assimétrica)
(ii)	se $x \prec y$ e $y \prec z \Rightarrow x \prec z$	(transitiva)
(iii)	x $\not\prec$ x	(irreflexiva)

Podemos então formular o problema da ordenação topológica nos seguintes termos : dada uma ordem parcial R sobre um conjunto não-vazio S de n elementos, queremos achar uma permutação $\mathrm{q} = q_1 q_2 \ldots q_n$ dos elementos de S de tal forma que x aparece à esquerda de y em q sempre que x $\prec$ y. Ou seja, se $q_i \prec q_j$ então

$i < j$. q será dita uma extensão linear (ou ordenação topológica) do poset. Denotaremos por $E(\mathcal{P})$ e $e(\mathcal{P})$ o conjunto e o número *de* extensões lineares de um poset $\mathcal{P}$, respectivamente. As seguintes afirmações são equivalentes :

(a) Dada uma ordem parcial R em um conjunto finito não-vazio S, ache uma ordem linear (total) na qual a ordem parcial R possa ser embutida.

(b) Dada uma ordem parcial R em um conjunto finito não-vazio, ache uma maneira de se permutar as linhas e colunas da matriz de adjacências M de R de tal modo que M se torne uma matriz triangular superior.

(c) Dado um digrafo acíclico D com n vértices, ache a rotulação dos vértices de D utilizando os inteiros $\{1, 2, \ldots, n)$ tal que para cada aresta (x, y) em D, o rótulo de x seja menor que o rótulo de y.

Podemos citar como exemplos práticos do problema da ordenação topológica : construção de um glossário de termos, escalonamento de tarefas (de um projeto, em uma linha de produção, de uma frota de veículos), definição de um currículo universitário, etc. As permutações dos elementos de um conjunto constituem um caso particular deste problema, onde o poset em questão tem um conjunto de relações R vazio [KV83, Sed77, Joh63, PR88]. O conjunto de extensões lineares de um poset desperta interesse na combinatória, uma vez que muitos objetos coinbinatórios podem ser representados através de um conjunto parcialmente ordenado. As extensões de um poset, dependendo deste, podem ser identificadas com permutações, permutações alternadas [Rusdo, BR90], permutações em tableaux de Young [Knu68] ou em multiconjuntos [Rus88a, Knu68], etc. Uma visão geral sobre o conceito de posets e extensões lineares pode ser encontrada em Aigner [Aig79], Knuth [Knu68] ou Stanley [Sta86].

Dois elementos distintos x, y em S são ditos comparáveis se ou $x \prec y$ ou $y \prec x$, sendo ditos incomparáveis caso contrário, quando denotamos por $x \| y$. Se $x \prec y$ e $\nexists z \in S$ tal que $x \prec z \prec y$, dizemos que y cobre x. Para posets finitos, todo o conjunto R fica bem determinado por esta relação de cobertura, sendo por isso freqüentemente usada para representar o poset diagramaticamente. Modelando $\mathcal{P}(S, R)$ como um digrafo $D(S, E)$ onde existe aresta $(x, y) \in E$ se e somente se $x \prec y$, temos que a redução transitiva [Szw83] de D corresponde exatamente ao conjunto de arestas

$\mathcal{P}(S, R)$ onde $R = \{(a,a),(a,b),(a,c),(a,d),(b,b),(b,c),(b,d),(c,c),(d,d)\}$

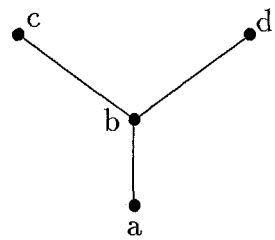
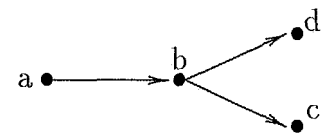


diagrama de Hasse



redução transitiva

Figura 11.1:

(x, y) tais que y cobre x, $\forall x, y \in S$. Uma outra forma de representar essa relação de cobertura seria por meio de um diagrama de Hasse [Tro83, Rus88a] : se y cobre x em $\mathcal{P}$, então y é posicionado acima de x e uma aresta (não-direcionada) liga x a y.

A relação binária contendo todos os pares comparáveis de $\mathcal{P}$ é chamada de relação de comparabilidade do poset. Esta relação é claramente simétrica e define o conjunto de arestas de um grafo cujo conjunto de vértices é dado pelos elementos de S, o qual chamamos de *grafo* de *comparabilidade* de $\mathcal{P}$ e denotamos por $G_C(\mathcal{P})$. Se pensamos nos pares incomparáveis de $\mathcal{P}$ como definindo um outro grafo, o grafo de *incomparabilidade* de $\mathcal{P}$, vemos que este é exatamente o grafo complemento de $G_C(\mathcal{P})$. Dado um grafo não orientado $G(V, E)$, G é o grafo de comparabilidade de algum poset $\mathcal{Q}$ se e somente se G possui uma orientação $\vec{G}(V, \vec{E})$ de suas arestas tal que para qualquer par (v, w) e $(w, z) \in \vec{E}$ então $(v, z) \in \vec{E}$. Ou seja, G admite uma orientação transitiva.

Um poset é dito uma cadeia (ou ordem total ou ordem linear) [Tro83, Rus88a] quando quaisquer dois de seus elementos são comparáveis entre si. A denominação acima se dá ao fato da redução transitiva do poset em questão ser uma cadeia, como vemos na figura 11.2. Denotamos uma cadeia com k elementos por $\mathcal{C}_k$.

Seja o poset $\mathcal{P}(S, R)$. Um subposet de $\mathcal{P}$ é um poset $\mathcal{P}'(S', R')$, onde $S' \subseteq S$ e R' é a restrição $R_{S'}$ de R em $S' \times S'$. Segundo esta definição, um subposet é caracterizado pelo seu subconjunto de elementos, sendo por isto denotado também por $\mathcal{P}_{S'}$. Uma

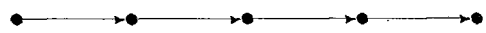


Figura 11.2: Cadeia de tamanho 5.

extensão linear de um subposet de um poset $\mathcal{P}$ é dita uma *extensão parcial* de $\mathcal{P}'$.

Seja $S' \subseteq S$. Por $\mathcal{P}\backslash S'$ (ou, indistintamente, $\mathcal{P} - S'$), denotamos o poset $\mathcal{Q}(S\backslash S', R\backslash R')$, onde $R' = \{(x,y) \in R \mid x \in S' \text{ ou } y \in S'\}$. Se $R' \subseteq S \times S$, denotamos por PUR' o poset com conjunto de elementos S e ordem parcial dada pelo fecho transitivo de $R \cup R'$. Uma *extensão* de um poset $\mathcal{P}(S,R)$ é um poset $\mathcal{Q}(S',R')$ tal que $S = S'$ e $R \subseteq R'$. Uma ordem total que seja uma extensão de $\mathcal{P}$ nada mais é do que uma extensão linear de $\mathcal{P}$.

Se $\mathcal{P}(S,\mathrm{R})$ e $\mathcal{Q}(S',R')$ são posets tais que o fecho transitivo de $R \cup R'$ é antissiinétrica, denotemos por $\mathcal{P} + \mathcal{Q}$ o poset sobre o conjunto $S \cup S'$ com ordem parcial dada pelo fecho transitivo de RUR'. Se $\mathcal{P}+\mathcal{Q} = \mathcal{P}$, então dizemos que $\mathcal{P}$ *induz* $\mathcal{Q}$. Por $a_1a_2 \ldots a_n$ denotamos o poset definido pela ordem total $a_i \prec a_j, \forall 1 \leq i < j \leq \mathrm{p}$, sobre o conjunto de elementos $\{a_1, a_2, \ldots, a_n\}$. Para orclens totais disjuntas em elementos $\alpha, \beta, \gamma, \delta$, denotemos por $\alpha(\beta \cup \gamma)\delta$ o poset $\alpha\beta\delta + \alpha\gamma\delta$.

O dual R' de uma relação binária R é o conjunto de pares (x,y) para os quais $(y,x) \in R$. Quando R é uma ordem parcial em S, R' também o é, e se torna natural se referir a $\mathcal{P}'(S,R')$ como o *dual* de $\mathcal{P}(S,R)$. Se o diagrama de Hasse de $\mathcal{P}$ é invertido, ele se torna o diagrama de seu dual.

Vejamos agora um teorema formulado por Szpilrajn [Szp30], básico para nosso estudo :

Teorema 1

(i) *Toda ordenação parcial R de um poset $\mathcal{P}(S,R)$ possui uma extensão linear (ordenação topológica);*

(ii) *A interseção das relações definidas por cada extensão linear de $\mathcal{P}$ (cada extensão $q_1q_2 \ldots q_n$ define uma relação $\preceq'$ tal que se $i \leq j \Rightarrow q_i \preceq' q_j$) define a própria*

relação R,

Consideramos o poset vazio $(S = \emptyset, R = \emptyset)$ como contendo uma única extensão linear (vazia). O poset trivial é aquele constituído por um Único elemento. O tamanho de um poset é dado pelo seu número de elementos.

Pelo teorema 1, temos que qualquer ordenação parcial pode ser determinada pela interseção de suas extensões lineares, ou seja, para quaisquer dois elementos incomparáveis x e y em um poset P, existem pelo menos uma extensão de P na qual $x \prec y$ e outra na qual $y \prec x$. Entretanto, geralmente não precisamos de todas as extensões de um poset para determinar sua ordem parcial. Definimos a dimensão de um poset $\mathcal{P}(S, R)$ - $dim(\mathcal{P})$ - como sendo a cardinalidade do menor subconjunto de extensões lineares de P cuja interseção determina R. Existe na literatura toda uma teoria que estuda e classifica os posets de acordo com suas dimensões (ver Kelly & Trotter [KT82] ou Trotter [Tro83]). Temos que, para qualquer poset $\mathcal{P}$ com $n \geq 4$ elementos, a seguinte desigualdade (desigualdade de Hiraguchi, 1951) é válida :

$$dim(\mathcal{P}) \leq \frac{n}{2}$$

As cadeias constituem os posets de dimensão 1; já os posets série-paralelos, os definidos por tableaux de Young, as anticadeias (são um caso particular de posets série-paralelos), assim como os reticulados planares (reticulados que podem ser representados no plano sem nenhum cruzamento entre arestas) têm dimensão no máximo dois [VTL79, KT82, Tro83]. Estas classes de posets serão definidas na seção 11.2.

Um elemento $x \in S$ é dito minimal quando não cobre nenhum outro elemento do poset; é dito maximal quando $\nexists y \in S$ que o cubra. Denotamos o conjunto de todos os elementos minimais do poset por $Min(\mathcal{P})$, enquanto usamos $Max(P)$ para denotar o conjunto dos elementos maximais. Um elemento que preceda (seja precedido por) todos os outros elementos do poset é dito *mínimo* (máximo).

Seja $\mathcal{P}$ um poset, x um elemento de $\mathcal{P}$ e l uma extensão linear de P. A altura $h(x, l)$ (ou $h(x)$, quando isto não causar ambigüidade) de x em l é dada pelo número de elementos que precedem x nesta extensão mais um. Por $E(P|h(x) = i)$ e $e(P|h(x) = i)$ denotaremos, respectivamente, o conjunto e o número de extensões de $\mathcal{P}$ onde x ocupa a i-ésima posição.

O comprimento de um poset será dado pelo tamanho de sua maior cadeia. A largura de um poset será dada pelo tamanho de sua maior anticadeia (subconjunto de

elementos de S onde quaisquer dois de seus elementos não são comparáveis entre si) [Tro83]. O teorema a seguir, conhecido como teorema de decomposição de Dilworth [Dil51], diz respeito ao particionamento de um poset P em um conjunto de cadeias.

Teorema 2 (Teorema de Dilworth) Se $\mathcal{P}(S, R)$ é um poset de largura L, então existe uma partição de $S = C_1 \cup C_2 \cup \ldots \cup C_k$ onde cada C; induz uma cadeia C; em $\mathcal{P}$.

Prova : Procedemos por indução em $n = |S|$. O caso $n = 1$ é trivial. Assumimos válido o caso de posets com menos que n elementos e consideramos o poset $\mathcal{P}$ com n elementos. Podemos assumir, sem perda de generalidade, que a largura de $\mathcal{P}$ é maior que 1 (senão $\mathcal{P}$ seria uma cadeia).

Seja x E $Max(\mathcal{P})$ e y $\in Min(\mathcal{P})$, com y $\preceq$ x. Seja $\mathcal{Q} = \mathcal{P} - \{x, y\}$. Se a largura de $\mathcal{Q}$ é menor que L, então particionamos $\mathcal{Q}$ em menos que k cadeias, as quais junto com a cadeia yx forma uma partição de $\mathcal{P}$ em no máximo L cadeias. Se $\mathcal{Q}$ tem largura L, então y $\prec$ x em $\mathcal{P}$ e escolhemos uma anticadeia $A = \{a_1, \ldots, a_k\}$ em $\mathcal{Q}$.

Seja agora $U = \{u \in S \mid a_i \preceq u$ para algum a; $\in A\}$ e $D = \{d \in S \mid d \preceq a_j$ para algum $a_j \in A\}$. Evidentemente, x E $U \cap D$ e $y \in D \backslash U$. Então existem partições $U = C_1' \cup \ldots \cup C_k'$ e $D = C_1'' \cup \ldots \cup C_k''$, onde cada C_i' induz uma cadeia C; e cada C_j'' induz uma cadeia C''_j em P. Podemos rotular estas cadeias de tal modo que a_i E $C_i' \cap C_i''$, $1 \leq i \leq L$. Então, $C_i = C_i' \cup C_i''$ induz uma cadeia C_i em P, para cada i, e uma partição de S é $C_1 \cup \ldots \cup C_k$. □

O teorema de Dilworth tem uma versão trivial para o particionainento de um poset em anticadeias.

Teorema 3 Se $\mathcal{P}(S, R)$ é um poset de comprimento L, então existe uma partição de $S = A_1 \cup A_2 \cup \ldots \cup A_k$ onde cada A; induz uma anticadeia em $\mathcal{P}$.

11.2 Algumas Classes de Posets

Quando o poset é constituído por um conjunto de cadeias disjuntas é denominado um *multiconjunto*. Se o multiconjunto éformado por t cadeias de tamanhos $n_1, n_2, \ldots, n_t$, denota-10-emos pelo vetor $n = (n_1, \ldots, n_t)$.

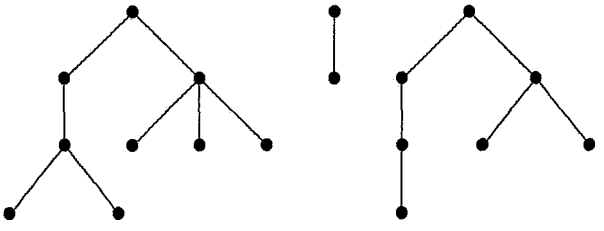


Figura 11.3: Diagrama de Hasse de um poset floresta com três árvores.

Um poset *floresta* é um poset cuja redução transitiva é uma floresta de in-árvores (grafo direcionado onde todo vértice tem grau de saíi da ≤ 1). Um exemplo é dado na figura 11.3. Os multiconjuntos são um caso especial de florestas.

Um poset $\mathcal{P}(S, \mathrm{R})$ será dito graduado quando pudermos organizar seus elementos em "níveis", de tal forma que cada elemento de S só cubra elementos que se encontrem em um nível imediatamente superior ao seu. Esta caracterização fica mais clara quando pensamos no diagrama de Hasse do poset $\mathcal{P}$. Vejamos os exemplos da figura II.4.

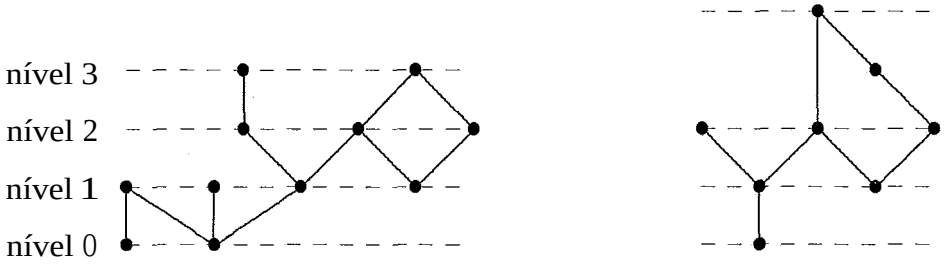
Mais formalmente, Pruesse & Ruskey [PR88] definem um poset graduado como sendo aquele sobre o qual podemos definir uma função $\rho : S \to \mathrm{N}$ tal que $\rho(a) = 0$, se a é minimal, e $\rho(b) = \rho(a) + 1$, se b cobre a. A classe dos posets florestas está propriamente incluída na classe dos posets graduados.

Seja $\mathcal{Q}$ um poset e $a_1, \ldots, a_h \in \mathcal{Q}$. Sejam $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h$ posets mutuamente disjuntos. Então, denotamos por

$$\mathcal{P} = \mathcal{Q}_{a_1,\ldots,a_h}^{\mathcal{P}_1,\ldots,\mathcal{P}_h}$$

o poset resultante da substituição de cada elemento a; em $\mathcal{Q}$ pelo poset P; correspondente $(1 \leq \mathrm{i} \leq h)$. Mais formalmente,

$$a \prec_{\mathcal{P}} \mathrm{b} \;\Leftrightarrow\; \exists i \text{ tal que a,b} \in \mathcal{P}_i \text{ e } a \preceq_{\mathcal{P}_i} \mathrm{b} \text{ ou}$$
$$\exists \mathrm{i} \neq j \text{ tal que a E } \mathcal{P}_i, \mathrm{b\ E}\ \mathcal{P}_j \text{ e a; } \prec_{\mathcal{Q}} a_j.$$



$\mathcal{P}$ graduado $\mathcal{P}$ não graduado

Figura 11.4: Poset graduado e poset não graduado.

A substituição de $\mathcal{P}_i$ é dita *própria* se $1 < |\mathcal{P}_i| < |\mathcal{P}|$. Um poset é dito *decomponível* se pode ser obtido por substituição própria. De outro modo, o poset é dito *indecomponível* ou *primo*. Veja [MR84, Ste83, HM79, BM83] para uma descrição mais detalhada sobre a operação de substituição ou sua decoinposição associada.

Cada poset decomponível $\mathcal{P}$ é obtido através de uma seqüência :

$$\mathcal{P}_1 = \mathcal{Q}_1,\ \mathcal{P}_2 = \mathcal{P}_1{}_{a_2}^{\mathcal{Q}_2},\ \ldots,\ \mathcal{P} = \mathcal{P}_{m-1}{}_{a_m}^{\mathcal{Q}_m}$$

de substituições "elementares", nas quais cada $\mathcal{Q}_j$ é primo. Os posets $\mathcal{Q}_j$ são únicos (a menos de isomorfismo e reordenação) e são chamados de *fatores* de $\mathbf{P}$. *O* tamanho máximo destes fatores é chamado de *diâmetro de decomposição* de $\mathcal{P}$.

Teorema 4 (Teoreina de Decoinposição) *Para cada poset decomponível* $\mathcal{P}$, *um dos três casos abaixo se aplica* :

(1) $\mathcal{P} = \mathcal{Q}_{a_1,\ldots,a_h}^{\mathcal{P}_1,\ldots,\mathcal{P}_h}$, *onde* $\mathcal{Q}$ *é uma anticadeia. Então,* $\mathcal{P}$ *é dito obtido por uma* composição em paralelo *de* $\mathcal{P}_1,\ldots,\mathcal{P}_h$;

(2) $\mathcal{P} = \mathcal{Q}_{a_1,\ldots,a_h}^{\mathcal{P}_1,\ldots,\mathcal{P}_h}$, *onde* $\mathcal{Q}$ *é uma cadeia. Então,* P *é dito obtido por uma* composição em série *de* $\mathcal{P}_1,\ldots,\mathcal{P}_h$;

(3) $\mathcal{P} = \mathcal{Q}_{a_1,\ldots,a_h}^{\mathcal{P}_1,\ldots,\mathcal{P}_h}$, *onde* $\mathcal{Q}$ *é um poset primo (unicamente determinado). Então,* $\mathcal{P}$ *é dito do* tipo primo *e* $\mathcal{Q}$ *é chamado de* poset quociente primo associado.

Um caso especial de posets decomponíveis são os posets *série-paralelos* [VTL79, HM87, Val78], os quais representam a menor classe de posets que contém o poset trivial e é fechada sob composições em série e em paralelo (no sentido dos casos (1) e (2) do teorema de decomposição). Os posets floresta também se encontram propriamente incluídos nesta classe.

Definição 1

(i) O *digrafo trivial* é série-paralelo minimal (SPM);

(ii) *Se* $D'(V', E')$ *e* $D''(V'', E'')$ *são dois digrafos SPM, assim o é o digrafo construido por uma das seguintes operações* :

(a) Composição em paralelo : $D(V' \cup V'', E' \cup E'')$,

(b) Composição em série : $D(V' \cup V'', E' \cup E'' \cup [Max(D') \times Min(D'')])$, *onde* $Max(D')$ *é o conjunto de sumidouros de* D' *e* $Min(D'')$ *é o conjunto de fontes de* D''.

Definição 2 *Um digrafo* é série-paralelo *se e somente se sua redução transitiva* é *série-paralelo minimal (SPM).*

Um digrafo SPM pode ser representado através de sua *árvore binária de decomposição* : uma árvore binária onde cada folha representa um vértice do digrafo e onde cada nó interno representa uma composição em série $(\otimes)$ ou em paralelo $(\oplus)$ dos digrafos SPM representados pelas subárvores enraizadas nos filhos deste nó. Árvores binárias de decomposição provêem uma descrição concisa da estrutura de um digrafo SPM. Entretanto, várias árvores binárias de decomposição não-isomorfas podem representar o mesmo digrafo SPM, devido à simetria da composição em paralelo e à associatividade de coinposições em série ou em paralelo consecutivas.

Definimos um poset série-paralelo $\mathcal{P}(S, R)$ como sendo aquele cuja redução transitiva é um digrafo SPM (ou como aquele que pode ser representado por um digrafo série-paralelo). É interessante notar que qualquer poset série-paralelo tem dimensão menor ou igual a 2 [VTL79]. Em Valdes et al [VTL79], podemos encontrar um algoritmo linear em $n = |S|$ e $m = |R|$ para o reconhecimento desta classe, que também determina uma árvore de decomposição binária para o poset, caso este seja de fato série-paralelo. Um exemplo destes posets pode ser encontrado na figura 11.5.

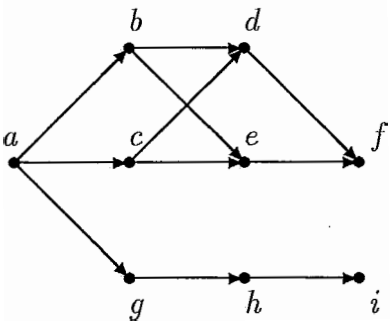
Um *tableau* de Young de formato $(n_1, \ldots, n_m)$, onde $n_1 \geq n_2 \geq \ldots \geq n_m \geq 0$, é uma disposição de $n_1 + \ldots + n_m$ inteiros distintos em uma tabela de linhas justificadas à esquerda, com n_i elementos na linha i, tal que as entradas de cada linha estão em ordem crescente da esquerda para a direita, enquanto que as de cada coluna estão em ordem crescente de cima para baixo. O exemplo da figura II.6 é um tableau de Young de formato (6,4,4,1). Um tableau pode ser representado através de um digrafo acíclico onde existe aresta (x, y) se e somente se o rótulo do vértice x é menor que o do vértice y. Na figura 11.7, aparece o diagrama de Hasse relativo ao tableau da figura 11.6.

Um poset $\mathcal{P}(S, R)$ é dito um reticulado, quando para cada par de elementos $x, y \in S$, existem um único a e um único b tais que : $x \preceq b$ e $y \preceq b$ e não existe $z \prec b$ tal que $x \preceq z$ e $y \preceq z$; $a \preceq x$ e $a \preceq y$ e não existe $a \prec z$ tal que $x \preceq x$ e $z \preceq y$. A relação (usual) $\subseteq$ sobre o conjunto de partes de um conjunto A, define o que chamamos de um reticulado de partes. Já o reticulado definido por expressões de até k variáveis na álgebra booleana é chamado de reticulado de álgebra booleana. Estas ordens são discutidas em Aigner [Aig79].

Na seção 11.3.1, caracterizamos ainda os B-posets, assim como no capítulo VII várias outras classes de posets são definidas – a saber, posets de largura k, posets bipartite, posets floresta não enraizada, poset zigzag, etc.

11.3 Considerando as Extensões Lineares de um Poset

A transposição de dois elementos em uma permutação de n elementos consiste simplesmente na troca das posições destes dois elementos, resultando assim em uma nova



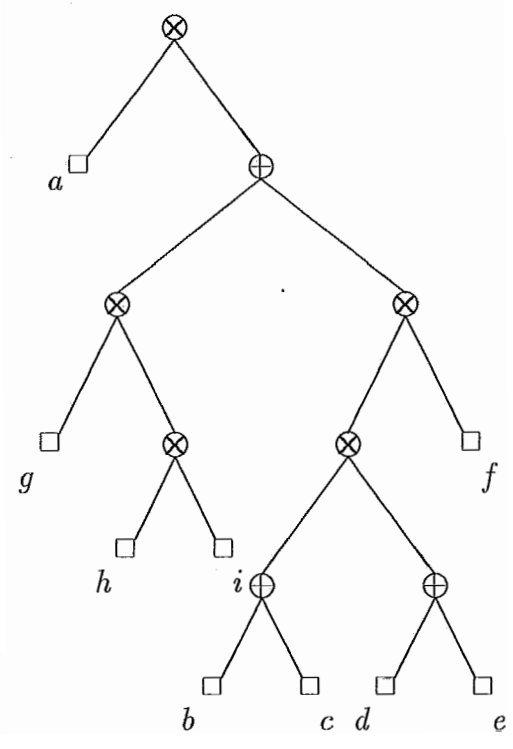


Figura 11.5: Digrafo SPM e sua árvore de decomposição.

1	2	5	9	10	15
3	6	7	13		
4	8	12	14		
11					

Figura 11.6: Tableau de formato (6,4,4,1).

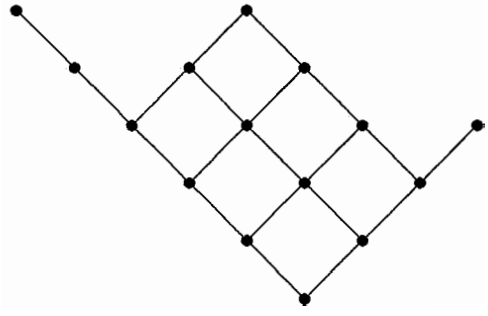


Figura II.7: Diagrama de Hasse de um tableau de formato (6,4,4,1).

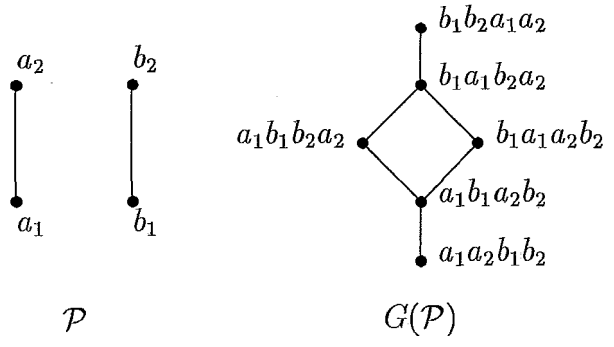


Figura 11.8: Um poset e seu grafo de transposição.

permutação dos n elementos. Quando pensamos em gerar extensões lineares por meio de transposições, naturalmente surge a idéia de um *grafo de transposição* $G(\mathcal{P})$ associado ao poset $\mathcal{P}$, como definido por Rusliey [Rus88a]. Os vértices deste grafo são as permutações que correspondem a extensões do poset, existindo aresta entre um par de vértices de $G(\mathcal{P})$ se e somente se as extensões correspondentes diferirem por apenas uma transposição. Na figura 11.8, temos um exemplo de um poset e seu grafo de transposição. Note que o grafo $G(\mathcal{P})$ é sempre um subgrafo induzido do grafo de transposição para permutações irrestritas do conjunto de elementos de $\mathcal{P}$. O problema de se gerar todas as extensões lineares de um poset por meio de transposições fica então reduzido ao problema de se achar um *caminho hamiltoniano em* $G(\mathcal{P})$. Rusliey [Rus88a, Rus88b, Rusdo], Pruesse & Ruskey [PR91] e Eades, Hickey & Read [EHR84] apresentam trabalhos onde o problema é tratado sob este enfoque. Mais adiante, no capítulo IV, abordaremos esta questão.

Podemos definir ainda um *grafo de transposição adjacente* $G'(\mathcal{P})$, onde o conjunto de vértices é o mesmo de $G(\mathcal{P})$, mas onde existe aresta entre um par de extensões se e somente se estas diferirem por uma transposição de dois elementos adjacentes entre si. $G'(\mathcal{P})$ nem sempre é um subgrafo induzido do grafo de transposição para permutações irrestritas, mas note-se que ele é um subgrafo convexo de $G(\mathcal{P})$ (se H é um subgrafo convexo de G, então todo vértice de G em um caminho mais curto entre qualquer par de vértices de H também está em H [Rus88a]). No caso de qualquer poset de largura dois (veja figura II.9), ambos $G(\mathcal{P})$ e $G'(\mathcal{P})$ coincidem, pois uma transposição que não

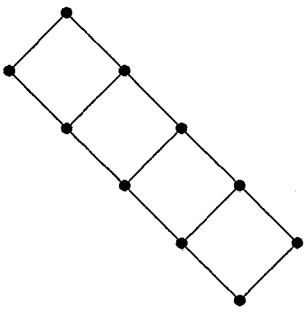


Figura 11.9: Tableau de Young de largura 2.

fosse adjacente implicaria na existência de pelo menos três elementos incomparáveis entre si em P .

Propriedades do grafo de transposição adjacente $G'(\mathcal{P})$ refletem propriedades importantes do poset P . O *número de salto* de uma extensão linear $q_1 q_2 \ldots q_n$ é o número de índices $1 \leq i \leq n-1$ tais que q_i e q_{i+1} são incomparáveis. Ou seja, o número de salto de uma extensão linear corresponde ao grau deste vértice em $G'(\mathcal{P})$. O número de salto de um poset é o menor número de salto de suas extensões lineares, ou ainda, é o menor grau de um vértice em $G'(\mathcal{P})$. Números de salto são discutidos em Chein & Habib [Che80], Rival [Riv83] e Syslo [Sys85].

O número de choque de uma extensão linear $q_1 q_2 \ldots q_n$ é o número de índices $1 \leq i \leq n-1$ tais que $q_i \prec q_{i+1}$. Analogamente ao número de salto, definimos o número de choque de um poset como sendo o maior dentre os números de choque de suas extensões lineares. O número de choque de um poset será dado pela diferença entre $(n-1)$ e o menor grau de um vértice em $G'(\mathcal{P})$ (número de salto de $\mathcal{P}$). Números de choque são discutidos em Fishburn & Gehrlein [FG86] e Zaguia [Zag88].

Seja l uma extensão linear de um poset $\mathcal{P}(S, \mathrm{R})$. Uma *inserção* de um elemento i E S é a operação de se remover i de sua posição em uma extensão linear l de P, inserindo-o antes de/após um outro elemento j E S nesta extensão. As posições relativas entre todos os elementos do poset com exceção de i, permanecem inalteradas após uma inserção. Uma *k-inserção* é uma inserção de um elemento i tal que i

é sobreposto a k outros elementos, i. e., [(posição destino de i) – (posição origem de i)| $= k$. Transposições adjacentes são equivalentes a 1-inserções. Assim, todos os resultados apresentados no capítulo IV para gerações por transposição adjacente também valem para gerações por inserção, mais ainda por 1-inserção.

Uma fórmula de recorrência que calcula o número de extensões lineares de um poset $\mathcal{P}$, $e(\mathcal{P})$, é dada abaixo [Rus88a] :

$$e(\mathcal{P}) = \begin{cases} 1, & \text{se } |S| = 1; \\ \sum_{x \in Min(\mathcal{P})} \text{e(? } - \{\text{x}\})\text{ ,} & \text{caso contrário.} \end{cases} \qquad \text{(II.1)}$$

A fórmula **(11.1)**classifica as extensões segundo seus primeiros elementos. Ela não é eficiente, uma vez que envolve essencialmente a geração de todas as extensões lineares de $\mathcal{P}$. Aliás, a enumeração das extensões de um poset não constitui de modo algum um problema trivial : foi provado ser #P-completo (ver seção 11.4) no caso geral, só sendo conhecidas fórmulas diretas que nos retornem este valor para alguns poucos casos especiais de posets, como por exemplo quando a redução transitiva de $\mathcal{P}$ é uma floresta [SW78] ou é um tableau de Young [Knu68] (veja capítulo VII).

Se rotulamos os n elementos de um poset $\mathbf{P}$ usando os inteiros $1, 2, \ldots, n$ de tal forma que a permutação correspondente a 1 2 ... n seja uma extensão linear do poset, então definimos a *permutação* canônica de $\mathcal{P}$ como sendo exatamente esta permutação. Seja $d(\mathcal{P})$ o número de extensões que se encontram a uma distância par da permutação canônica no grafo de transposição menos o número de extensões que se encontram a uma distância ímpar (em outras palavras $d(\mathcal{P})$ é o número de permutações pares menos o número de permutações ímpares). Este número será conhecido como a diferença de paridade do poset $\mathcal{P}$, podendo ser calculado através da fórmula de recorrência a seguir. Vale ressaltar que $d(\mathcal{P})$ não é uma função que dependa somente do poset em consideração, uma vez que seu sinal varia de acordo com a rotulação do poset (escolha da extensão linear a ser rotulada como permutação canônica), mas isto não nos trará nenhum problema.

$$d(\mathcal{P}) = \begin{cases} 1, & \text{se } |S| = 1; \\ \sum_{x \in Min(\mathcal{P})} (-1)^{x-1} d(\mathcal{P} - \{\text{x}\}), & \text{caso contrário.} \end{cases} \qquad (11.2)$$

Assumimos que a mesma ordenação dos elementos em $\mathcal{P}$ é usada para definir a permutação canônica de $\mathbf{P} - \{\text{x}\}$. Vemos que a fórmula (11.2) é válida pois a diferença

de paridade para permutações que tenham como elemento inicial o elemento x é $d(P - \{x\})$ e temos que fazer ainda $x - 1$ transposições adjacentes para voltar com x ao seu lugar próprio na permutação canônica. E sempre válido lembrar que todo caminho entre um par de vértices qualquer em $G(\mathcal{P})$ tem paridade única ($G(\mathcal{P})$ é bipastite, como provamos no capítulo IV), portanto podemos escolher um caminho arbitrário entre a permutação em questão e a canônica. Neste caso, escolhemos o que considera somente transposições adjacentes.

Assim como em (11.1)) a fórmula de recorrência para diferenças de paridade só foi resolvida para alguns casos especiais de posets. E claro ainda que poderíamos ter usado a recorrência sobre o conjunto $Max(\mathcal{P})$ ao invés de $Min(\mathcal{P})$, tomando o cuidado de usar o fator $(-1)^{n-x}$ para multiplicar $d(P - \{x\})$, x E $Max(\mathcal{P})$, pois agora as transposições adjacentes seriam feitas partindo-se da última posição na permutação.

Um poset será dito *balanceado* quando possuir o mesmo número de permutações ímpares e pares como extensões, ou seja, $d(\mathcal{P}) = 0$. Existe uma conjectura estabelecida por Ruskey [Rus88b], onde temos que se o poset é balanceado então podemos gerar todas as suas extensões lineares por meio de transposições.

Se G é um grafo, seja $G \times K_2$ o grafo correspondente à duas cópias de G ligadas por arestas que traduzem um isomorfismo entre estas cópias. Para diferenciar entre as duas cópias de G, prefixaremos os vértices de uma com o sinal "+" e os da outra com o sinal "−". Para o poset da figura II.8, a figura II.10 mostra o grafo $G(\mathcal{P})$ x K_2.

II.3.1 B-posets

Definiremos aqui a classe dos B-posets e enunciaremos um lema provado em Ruskey [Rus88a] que diz respeito a esta classe. Os B-posets representam um papel de vital importância para as provas de geração por transposição para certos posets graduados (seção IV.3) e de geração para o caso geral em tempo médio constante (capítulo VI).

Definição 3 Um B-poset é um poset $\mathcal{P}$ *tal* que seus elementos podem ser particionados em duas cadeias disjuntas $x_1 \prec x_2 \prec \cdots \prec x_r$ e $y_1 \prec y_2 \prec \cdots \prec y_s$, onde $y_j \not\preceq x_i$, para todo $1 \leq j \leq s$ e $1 \leq i \leq r$.

Ilustramos um B-poset na figura 11.11.

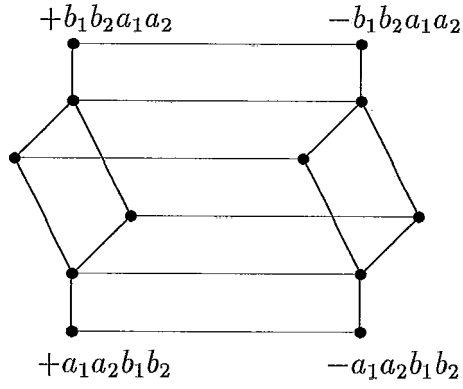


Figura 11.10: O grafo $G(\mathcal{P})$ x K_2.

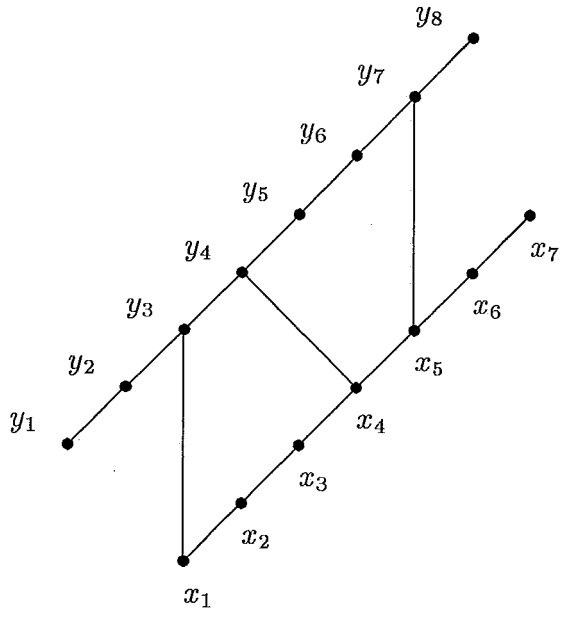


Figura 11.11: Uin B-poset.

Chamamos a r e s de parâmetros de um B-poset. Um B-poset é caracterizado pela seqiiência s $\geq l_1 \geq l_2 \geq \ldots \geq 1, \geq 0$, onde $l_i = |\{y_j \mid x_i \prec y_j\}|$. Assim, o número de B-posets com parâinetros r e s é (r + s)!/(r!s!). Um poset é um B-poset se e somente se tem número de salto menor ou igual a 1.

Notemos que $\kappa = x_1 x_2 \ldots x_r y_1 y_2 \ldots y_s$ é sempre uma extensão linear de um B-poset, que chamaremos de extensão *canônica* do B-poset. Para o propósito do lema 1, um grafo conexo com uma única aresta é considerado como contendo um ciclo hamiltoniano.

Lema 1 *Seja* $\mathcal{P}$ um B-poset. Então $G(\mathcal{P}) \times K_2$ admite um ciclo hamiltoniano contendo a aresta $(+\kappa, -\kappa)$.

Como qualquer B-poset $\mathcal{P}$ é um poset de largura 2 (para uma caracterização completa dos posets de largura 2, ver seção VII.3.6),temos que $G(\mathcal{P}) = G'(\mathcal{P})$. Assim, o lema acima também vale para $G'(\mathcal{P})$, ou seja, só utilizamos transposições adjacentes na construção do ciclo deste lema.

Agora que já introduziinos os conceitos básicos relativos a conjuntos parcialmente ordenados, vamos analisar como resolver o problema preliminar de se achar uma extensão linear qualquer de um dado poset. Existe basicamente um único método para se resolver este problema, variações do mesmo diferindo apenas na estrutura de dados utilizada para manipulação do poset. Este método é ilustrado na figura 11.12.

A fórmula de solução presente no algoritmo da figura 11.12 pode ser encontrada em Kahn [Kah62], Knuth & Szwarcfiter [KS74], e Wirth [Wir76]. Já Aho, Hopcsoft & Ullman [AHU74], Horowitz & Sahni [HS76] e Szwarcfiter [Szw83] apresentam o problema usando um digrafo D para modelar o poset, enquanto Tarjan [Tar74] e Reignold, Nievergelt & Deo [RND77] usam uma busca em profundidade para obter uma floresta de espalhamento de D. Ainda podemos citar uma outra variante do problema presente em Nijenhius & Wilf [NW75] que envolve a triangulação de matrizes. Mantendo-se uma fila de vértices candidatos a j-ésima posição da extensão pode-se implementar o algoritmo apresentado em tempo O(n + m), n = $|S|$ e m = $|R|$, como pode ser visto em Szwarcfiter [Szw83].

dado poset $\mathcal{P}(S, \mathrm{R})$
para $j = 1, \ldots,$ n **efetuar**
Escolher elemento x E S tal que $\nexists$ relação do tipo y $\prec$ x em R;
Atribuir a x a j-*ésima* posição da extensão linear;
S := S – {x};
Retirar de R todas as relações do tipo x $\preceq$ y, y E S;
fim algoritino

Figura 11.12: Algoritmo para achar uma extensão linear.

Já o problema de se gerar todas as extensões lineares de um poset $\mathcal{P}(S, \mathrm{R})$ não é tão trivial. Como o número de extensões pode no pior caso ser $O(n!)$, vamos aqui definir o que chamamos um *algoritmo de tempo médio constante* para a geração de extensões lineares : o algoritmo gera todas as extensões em tempo $O(e(\mathcal{P}))$, independente de $|S|$ ou de $|R|$. Ou seja, cada extensão é gerada em tempo médio constante. Nos capítulos IV, V e VI, consideraremos algoritmos de tempo médio constante para geração de extensões lineares, seja em ordem lexicográfica, por transposição ou com atraso dois, seja para o caso geral ou para alguma classe particular de posets.

II.4 A Classe $\#P$

Como já mencionado, o problema de enumeração de extensões lineares de um poset foi provado ser #P-completo. Sendo este problema um dos dois tópicos principais deste trabalho (o outro é o problema de geração das extensões), parece adequado definir aqui a *classe de complexidade* $\#P$.

Dado um problema Π, temos associado a cada instância **I** deste problema um conjunto de soluções $S_\Pi(I)$. Dado um *problema de busca* Π somos requisitados, para cada instância **I** de 11, a encontrar *um* elemento de $S_\Pi(I)$ (o problema de decisão correspondente questiona se $S_\Pi(I)$ é vazio ou não). O *problema de enumeração* baseado no problema de busca Π poderia ser enunciado como "Dada uma instância **I**, qual é a cardinalidade de $S_\Pi(I)$, ou seja, quantas soluções existem para Π ?". Se Π é um problema de busca tal que, para cada instância **I** de Π e cada solução o E $S_\Pi(I)$, o

tamanho de a é um polinômio no tamanho de $\mathbf{I}$ e para o qual podemos determinar em tempo polinomial, dados $\mathbf{I}$ e o, se o E $S_{\Pi}(I)$, então o problema de enumeração correspondente se encontra em **#P**. Ou ainda, a classe $\#P$ consiste de todos os problemas de enumeração cujas soluções são o número de estados de aceitação de alguma máquina de Turing não-deterministica de tempo polinomial. Vemos, portanto, que $\#P$ contém ao menos uma grande paste dos problemas de-enumeracão que se possa querer considerar e, certamente, muitos deles parecem ser bastante difíceis, como é o caso do nosso problema de enumeração de extensões.

O conceito de completude para $\#P$ é mais uma vez usado para capturar a noção de problema "mais difícil" nesta classe. Um problema de enumeração Π será dito **#P**-completo se Π E $\#P$ e, para todo Π' E **#P**, Π' pode ser polinomialmente reduzido a Π. Veja [GJ79] para uma descrição mais completa das classes de complexidade.

Capítulo III

Geração de Extensões Lineares

O conjunto das extensões lineares de um poset é de grande interesse combinatório por sua forte relação com problemas de enumeração. As extensões lineares de um poset, dependendo deste, podem ser identificaclas com permutações, permutações alternadas, tableaux de Young, reticulados, permutações em multiconjuntos, combinações ou ainda ordenações possíveis para um dado conjunto. Aliás, esta estreita ligação entre extensões lineares de posets e problemas de ordenação baseados em comparação e de escalonamento explica o grande interesse de cientistas da computação (onde as extensões lineares são comumente chamadas de ordenações topológicas) por este problema. Por exemplo, existem vários problemas de escalonamento com processador único e restrições de precedência que foram provados ser NP-completos [LRKB77]. Uma maneira óbvia de se resolver estes problemas seria gerar todas as extensões compatíveis com as restrições de precedência e escolher a ótima. O problema de geração de todas as extensões lineares de um poset $\mathcal{P}(S, \mathrm{R})$ de forma ótima não é tão trivial quanto possa parecer à primeira vista. Este problema permaneceu em aberto (para o caso geral) até meados deste ano, quando Pruesse & Ruskey [PR91] apresentaram um algoritino de tempo médio constante (portanto, ótimo) que o resolve (ver capítulo VI). Este é o primeiro caso conhecido onde o problema de geração foi resolvido, mas onde o problema de enumeração correspondente foi provado ser #P-completo. Apresentaremos nesta seção um histórico do problema nos últimos anos, comparando os vários algoritmos entre si, e consideraremos alguns casos particulares da geração de extensões.

Uma estratégia bastante usada para se gerar um conjunto de objetos combinatórios

é a de se insistir que objetos sucessivos nesta geração difiram entre si de alguma maneira discreta e bem definida. Listas de objetos combinatórios com esta propriedade são chamadas códigos de Gray (gerais ou combinatórios). Por exemplo, o código binário refletido é um código de Gray que nos dá um método de geração de todas as cadeias de n-bits tal que cada cadeia difere de sua antecessora por um bit somente. Códigos de Gray foram encontrados para várias classes de objetos combinatórios, muitos destes descritos em [Wil89]. Outro exemplo de código de Gray pode ser tomado da geração de permutações irrestritas segundo o algoritmo de Steinhaus-Johnson-Trotter (seção III.1), onde cada permutação gerada difere de sua antecessora por uma transposição adjacente. As gerações por transposição, por transposição adjacente, por inserção ou em ordem lexicográfica, discutidas durante este capítulo e nos capítulos IV e V, também definem códigos de Gray para algumas classes especiais de posets. No capítulo VI, veremos como adaptar o algoritmo de geração para o caso geral de modo que duas extensões difiram entre si por no máximo duas transposições, incluindo assim o conjunto de extensões lineares de um poset qualquer no conjunto de objetos combinatórios que definem códigos de Gray.

O número de extensões lineares de um poset $\mathcal{P}(S, \mathrm{R})$ será frequentemente bem menor que n!, o número total de permutações de um conjunto S com n elementos. Poder-se-ia gerar todas as n! permutações e depois testar a consistência de cada uma delas com a ordenação parcial R, mas isto seria absurdamente ineficiente. Estendendo o algoritmo para obtenção de uma extensão linear, chegamos naturalmente a um algoritmo que utiliza a técnica de *backtracking* para gerar todas as extensõesde $\mathcal{P}$: dada uma permutação parcial $p_1 p_2 \ldots p_i$, com i variando entre 1 e n, ache todas as maneiras de se completar $p_1 \ldots p_i$ de modo a se obter uma extensão linear $p_1 \ldots \mathrm{p}_{\cdot}$. O método utilizado ramifica as soluções para todas as escolhas possíveis de p_{i+1}, como vemos na figura 111.1.

O procedimento $ExtLin(i)$ vai gerar todas as permutações de S consistentes com a ordem parcial R U $\{(p_i, p_{i+1}) \mid 1 \leq \mathrm{i} < \mathrm{n} - 1)$. Na entrada (e saída) de $ExtLin(i)$, $S - \{p_1, \ldots, p_{i-1}\}$ é o conjunto dos elementos ainda não considerados neste nível da recursão. Repare-se que o conjunto dos elementos minimais $Min(\mathcal{P})$ varia durante o processo, uma vez que retiramos e adicionamos relações ao poset P .

Um algoritmo apresentado por Wells [Wel71], também discutido em Kalvin e Varol [KV83], é uma implementação correta, mas bastante ineficiente, que também

procedimento $ExtLin(i)$
  se i < n + 1 **então**
    para $p_i \in Min(\mathcal{P})$ **faça**
      $S := S - \{p_i\}$
      Retirar de R todas as relações do tipo $p_i \preceq j$, j E S.
      $ExtLin(i + 1)$
      $S := S \cup \{p_i\}$
      Recuperar todas as relações do tipo $p_i \preceq j$, j E S.
  senão retornar $p_1, \ldots, p_n$
fim **procedimento**

dado poset $\mathcal{P}(S, \mathbf{R})$, onde n = $|S|$
$ExtLin(1)$

Figura 111.1: Algoritmo básico utilizando backtracking.

se utiliza de *backtracking*. Várias simplificações foram introduzidas e redundâncias retiradas para que Kalvin e Varol [KV83] apresentassem uma versão melhorada do algoritmo de Wells.

Um outro algoritmo de backtracking, este mais eficiente, é devido a Knuth & Szwarcfiter [KS74]. Este algoritmo tem complexidade de espaço O(m + n) e leva no máximo tempo O(m + n) por extensão gerada, uma vez que uma lista com todos os candidatos a serem escolhidos para a próxima posição da extensão (posição i + 1) é mantida. Assim, evitamos ter que fazer uma busca por forca bruta para escolha do (i + 1)-ésimo elemento, o que acarreta um ganho da ordem de n no cálculo da complexidade. A seguir, o algoritmo de Knuth-Szwarcfiter em linhas bem gerais :

procedimento $ExtLin(i)$
var j, base
se D não vazia **então**
  base := elemento mais à direita de D
  repita
    j := elemento mais à direita de D

Retire de D seu elemento mais à direita.
Retire todas as relações do tipo $j \prec k$.
Retorne j para a posição i$1 da extensão.
se i = n − 1 **então retornar** mais uma extensão linear
$ExtLin(i + 1)$
Recupere todas as relações do tipo $j \prec$ L.
Insira j à esquerda de D.
até que elemento mais à direita de D = base
fim procedimento

Algoritmo de *Knuth-Szwarcfiter.*

O problema central na aplicação da técnica de *backtracking* é o de se encontrar uma maneira conveniente para sequenciar sobre as possíveis escolhas do (i+1)-ésimo elemento. Para evitar uma busca por força bruta sobre o conjunto de elementos ainda não selecionados ($\neq \{p_1, \ldots, p_i\}$), descobrindo aqueles que não têm predecessores outros que $p_1, \ldots,$ p;, manteremos um vetor *count* onde o valor corrente de $count[j]$ é o número de relações do tipo k $\prec j$, L E $S - \{p_1, \ldots,$ p;), além de rima lista linear duplamente encadeada D com restrições de saída (só podemos retirar elementos mais à direita de D, podendo inserir elementos em ambos os extremos da lista). A cada instante, D vai conter precisamente os elementos j E $S - \{p_1, \ldots, p_i\}$ que têm $count[j] = 0$. A execução do procedimento $ExtLin$ pode causas modificações temporárias nos conteúdos de D e do vetor count, mas ambos serão restaurados na saída do procedimento. As operações de retirada e recuperação de relações vão respectivamente decrementar e incrementar as entradas apropriadas em count, além de inserir e remover elementos mais à direita de D.

Citaremos ainda um algoritmo apresentado por Varol & Rottem [KV83, VR81] que gera as extensões iterativamente. Ele gera as soluções a partir de uma permutação canônica 1 2 ... n (rotulada de modo a ser uma extensão linear do poset) pela aplicação de uma série de transposições adjacentes e rotações cíclicas para a direita, usando uma técnica que em muito se assemelha à utilizada no algoritmo clássico de Steinhaus-Johnson-Trotter.

Os dois últimos algoritmos mencionados possuem complexidade $O((n+m)e(\mathcal{P}))$ e $O(n^2 e(\mathcal{P}))$, respectivamente. Dependendo do poset fornecido como entrada para

o algoritmo, os desempenhos desses algoritmos podem se tornar sensivelmente mais eficientes. Por exemplo, se temos a priori uma extensão linear do poset que sabemos ser uma extensão inicial apropriada para o algoritmo de Varol & Rottem, este gera as extensões em tempo O(n e(?)) (linear por extensão). O algoritmo de Knuth & Szwarcfiter também tem sua complexidade reduzida à linear em vários casos.

III.1 Geração de Permutações Irrestritas

Existem na literatura vários algoritmos (algumas dezenas) para a geração das $n!$ permutações de n elementos, o que corresponde a se gerar todas as extensões lineares de um poset $\mathcal{P}(S, \mathrm{R})$ com n elementos e conjunto de relações $\mathrm{R} = \emptyset$. Este problema é um exemplo não trivial da necessidade da interação entre computação e matemática combinatória, suscitando interesse devido ao grande número de diferentes abordagens do problema que podem ser comparadas entre si. O mais clássico de todos estes algoritmos é o algoritmo formulado por Steinhaus [Ste64], Johnson [Joh63] e Trotter [Tro62] em 1962, que se utiliza somente de transposições adjacentes. Outros algoritmos geram as permutações de forma recursiva, utilizando *backtraking*, em ordem lexicográfica, etc (ver Sedgewick [Sed77], para uma coletânea destes algoritmos).

Vamos assumir, sem perda de generalidade, que os n elementos a serem permutados são os inteiros $1, 2, \ldots,$ n. A idéia central do algoritmo de Steinhaus-Johnson-Trotter consiste em sempre se mover o maior inteiro k possível, segundo uma dada direção e por meio de transposições adjacentes sobre outros inteiros menores que k. A cada inteiro associaremos uma direção (para esquerda ou para direita), denotando-a por uma seta e que representa a direção para onde o elemento "tende a ir". Começamos com todos os elementos apontando para à esquerda. Então, se temos 4 elementos a serem permutados (n = 4) :

$$\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{3}\,\overleftarrow{4}$$

Um inteiro k é **móvel** se existe um inteiro menor que e adjacente a k na direção para a qual k aponta. Por exemplo, na permutação acima, os inteiros 2, 3 e 4 são móveis, enquanto que na permutação

$$\overleftarrow{2}\,\overrightarrow{3}\,\overleftarrow{4}\,\overrightarrow{1}$$

somente o inteiro 4 é móvel.

(passo 1) Se não existem inteiros móveis, então **pare.**

(passo 2) Ache o maior inteiro móvel m.

(passo **3)** Troque *m* com o inteiro adjacente para o qual m aponta.

(passo 4) Reverta a direção de todos os inteiros k tais que **7 >** m. Volte ao *passo* **1.**

Algoritmo de Steinhaus-Johnson-Trotter.

Vejamos um exemplo de como funciona o algoritmo para n = 4

$\overset{\leftarrow}{1}\overset{\leftarrow}{2}\overset{\leftarrow}{3}\overset{\leftarrow}{4}$	$\overset{\leftarrow}{3}\overset{\leftarrow}{1}\overset{\leftarrow}{2}\overset{\leftarrow}{4}$	$\overset{\rightarrow}{2}\overset{\leftarrow}{3}\overset{\rightarrow}{1}\overset{\leftarrow}{4}$
$\overset{\leftarrow}{1}\overset{\leftarrow}{2}\overset{\leftarrow}{4}\overset{\leftarrow}{3}$	$\overset{\leftarrow}{3}\overset{\leftarrow}{1}\overset{\leftarrow}{4}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{2}\overset{\leftarrow}{3}\overset{\rightarrow}{4}\overset{\leftarrow}{1}$
$\overset{\leftarrow}{1}\overset{\leftarrow}{4}\overset{\leftarrow}{2}\overset{\leftarrow}{3}$	$\overset{\leftarrow}{3}\overset{\leftarrow}{4}\overset{\leftarrow}{1}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{2}\overset{\leftarrow}{4}\overset{\rightarrow}{3}\overset{\leftarrow}{1}$
$\overset{\leftarrow}{4}\overset{\leftarrow}{1}\overset{\leftarrow}{2}\overset{\leftarrow}{3}$	$\overset{\leftarrow}{4}\overset{\leftarrow}{3}\overset{\leftarrow}{1}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{4}\overset{\leftarrow}{2}\overset{\rightarrow}{3}\overset{\leftarrow}{1}$
$\overset{\rightarrow}{4}\overset{\leftarrow}{1}\overset{\leftarrow}{3}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{4}\overset{\rightarrow}{3}\overset{\leftarrow}{2}\overset{\leftarrow}{1}$	$\overset{\rightarrow}{4}\overset{\leftarrow}{2}\overset{\leftarrow}{1}\overset{\rightarrow}{3}$
$\overset{\leftarrow}{1}\overset{\rightarrow}{4}\overset{\leftarrow}{3}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{3}\overset{\rightarrow}{4}\overset{\leftarrow}{2}\overset{\leftarrow}{1}$	$\overset{\leftarrow}{2}\overset{\rightarrow}{4}\overset{\leftarrow}{1}\overset{\rightarrow}{3}$
$\overset{\leftarrow}{1}\overset{\leftarrow}{3}\overset{\rightarrow}{4}\overset{\leftarrow}{2}$	$\overset{\rightarrow}{3}\overset{\leftarrow}{2}\overset{\rightarrow}{4}\overset{\leftarrow}{1}$	$\overset{\leftarrow}{2}\overset{\leftarrow}{1}\overset{\rightarrow}{4}\overset{\rightarrow}{3}$
$\overset{\leftarrow}{1}\overset{\leftarrow}{3}\overset{\leftarrow}{2}\overset{\rightarrow}{4}$	$\overset{\rightarrow}{3}\overset{\leftarrow}{2}\overset{\leftarrow}{1}\overset{\rightarrow}{4}$	$\overset{\leftarrow}{2}\overset{\leftarrow}{1}\overset{\rightarrow}{3}\overset{\rightarrow}{4}$

Como em $\overset{\leftarrow}{2}\overset{\leftarrow}{1}\overset{\rightarrow}{3}\overset{\rightarrow}{4}$ não há inteiro móvel, o algoritmo termina.

Provamos a validade deste algoritmo facilmente por indução. Quando n = 2, começamos com a permutação $\overset{\leftarrow}{1}\overset{\leftarrow}{2}$ e chegamos por uma transposição adjacente à permutação $\overset{\leftarrow}{2}\overset{\leftarrow}{1}$, onde não existe inteiro móvel, e o algoritmo termina. Assumindo que o algoritmo funciona para n, vamos provar para n **+ 1.** É claro que, a princípio, n **$1** aponta para a esquerda e se moverá nesta direção passo a passo até que cheguemos a

$$(n\overset{\leftarrow}{+}1)\overset{\leftarrow}{1}\overset{\leftarrow}{2}\ldots\overset{\leftarrow}{n}.$$

Até então, os inteiros $1, 2, \ldots,$ n guardaram suas posições relativas e direções originais. Todas as permutações onde $n+1$ é inserido dentro desta ordem foram geradas. Agora, n **$1** está imóvel. Então, o inteiro m(= n) *a* ser selecionado é o mesmo que seria determinado se o algoritmo fosse aplicado a $1, 2, \ldots, n$ somente e, além disso, m troca de lugar com o mesmo inteiro adjacente que no caso de $n+1$ não estar presente.

O próximo passo será reverter a direção de $n+1$, para depois movê-lo sobre todos os outros elementos para a direita. Novamente, ocorre a mesma mudança que ocorreria se $n+1$ não estivesse presente, e mais uma vez $n+1$ "desliza" para o outro lado. Fica claro então que, para cada permutação de $1, 2, \ldots, n$, o inteiro $n+1$ é posicionado em todos possíveis intervalos. Ao chegarmos à Última permutação de $1, \ldots, n$ e $n+1$ deslizar para a outra extremidade, não teremos mais inteiro móvel e o algoritmo termina.

III.2 Geração por Inserção

A seguir, provaremos que qualquer poset admite uma geração de suas extensões lineares onde duas extensões sucessivas diferem somente por uma operação de inserção, ou seja, admite uma *geração por inserção. O sentido* de uma inserção de um elemento x é dado pelo sentido (para a esquerda ou para a direita) para onde movemos x partindo de sua posição corrente. Em outras palavras, se (posição origem de x) − (posição destino de x) < 0, então temos uma *inserção para a esquerda,* caso contrário temos uma *inserção para* a *direita.*

Teorema 5 *Todo poset $\mathcal{P}(S, R)$ admite uma geração por inserção de suas extensões lineares onde as inserções envolvidas possuem todas sentido de direção para a esquerda, e uma onde todas possuem sentido de direção para a direita.*

É válido ressaltar que os dois casos de geração do teorema não são necessariamente distintos (transposições adjacentes podem ser consideradas 1-inserções para a esquerda ou para a direita).

Prova : Provaremos o teorema por indução no número de elementos do poset. Primeiramente, analisaremos o caso de inserções somente para a esquerda.

Seja $\mathcal{P}(S, \mathrm{R})$ um poset com n elementos. Seja M o conjunto dos elementos minimais de $\mathcal{P}$. É fácil ver que o subposet $\mathcal{P}_M$ admite uma geração por inserção para a esquerda : basta usar o algoritmo de Steinhaus-Johnson-Trotter para permutações irrestritas, onde cada transposição adjacente pode ser interpretada como uma 1-inserção para a esquerda.

$$a_1 \ldots a_t \ldots a_{n-2} a_{n-1} x$$
$$a_1 \ldots a_t \ldots a_{n-2} x a_{n-1}$$
$$\ddots$$
$$a_1 \ldots a_t a_{t+1} x \ldots a_{n-2} a_{n-1}$$
$$a_1 \ldots a_t x a_{t+1} \ldots a_{n-2} a_{n-1}$$

Figura 111.2: Inserção do elemento maximal x em uma extensão $a_1 \ldots a_{n-1}$, onde a_t é o elemento mais à direita desta extensão tal que $a_t \prec$ x

Vamos supor agora que todo poset $\mathcal{Q}$ com $n' <$ n elementos admite uma geração por inserção para a esquerda. Seja x um elemento maximal $\in S \setminus M$ e $\mathcal{Q}$ o subposet $\mathcal{P} -$ {x}. Seja $\Gamma = \mathbf{e}_1, \ldots, \mathbf{e}_{e(\mathcal{Q})}$ uma geração por inserção para a esquerda de $E(\mathcal{Q})$. Começamos com $\mathbf{e}_1 = p_1 p_2 \ldots p_{n-1}$ e inserimos x nesta extensão. Seja p_s o elemento mais à direita em $\mathbf{e}_1$ tal que $p_s \prec_{\mathcal{P}}$ x, $1 \leq$ s $\leq$ n $-$ 1. Uma observação : o elemento p_s sempre existe, caso contrário x seria também minimal em $\mathcal{P}$ e já teria sido considerado em M. Inserimos x em todas as posições possíveis nesta extensão através de 1-inserções de x para a esquerda. "Deslizamos" x da posição mais à direita de $\mathbf{e}_1$ até a posição s + 1, como ilustrado na figura 111.2, onde $t =$ s e $a_1 \ldots a_{n-1} = p_1 \ldots p_{n-1}$.

Agora, tomemos a extensão e, $= q_1 q_2 \ldots q_{n-1}$, $2 \leq$ r $\leq e(\mathcal{Q})$, e suponhamos que as extensões e;, $1 \leq$ i $<$ r já tenham sido consideradas. Seja q_s, $1 \leq$ s $\leq$ n $-$ 1, o primeiro elemento $\prec_{\mathcal{P}}$ x mais à direita em e,. Como observado para $\mathbf{e}_1$, q_s tem que existir. Temos dois casos a analisas :

(a) Se a última extensão gerada a partir de $\mathbf{e}_{r-1} = p_1 p_2 \ldots p_{n-1}$ foi $p_1 \ldots p_{n-1} x$, então a geração de e, fica como ilustrado na figura 111.2 $(t =$ s e $a_1 \ldots a_{n-1} = q_1 \ldots q_{n-1})$, por meio de 1-inserções para a esquerda de x. Como $\mathbf{e}_{r-1}$ e e, diferiam por uma inserção para a esquerda em Γ, então $p_1 \ldots p_{n-1} x$ e $q_1 \ldots q_{n-1} x$, também o diferem na geração de E($\mathcal{P}$).

(b) Se a última extensão gerada a partir de $\mathbf{e}_{r-1} = p_1 \ldots p_{n-1}$ foi $p_1 \ldots p_l x \ldots p_{n-1}$, $1 \leq l <$ n $-$ 1[1], então reescrevamos $\mathbf{e}_{r-1}$ como $p_1 \ldots p_j \ldots p_{i-1} p_i p_{i+1} \ldots p_{n-1}$ e e, como $p_1 \ldots p_{j-1} p_i p_j \ldots p_{i-1} p_{i+1} \ldots p_{n-1}$, onde a inserção de p_i à frente de p_j $(1 \leq j < i \leq$ n $-$ 1) em $\mathbf{e}_{r-1}$ é a inserção para a esquerda que leva de $\mathbf{e}_{r-1}$ a e, em Γ. Seja $p_k = q_s$

[1]Se $l = n - 1$, cairíamos no caso (a).

o primeiro elemento $\prec$ x mais à direita em e,, $1 \leq k \leq$ l. Devemos analisar como proceder de acordo com o valor de l.

- Se $l =$ i, então temos que a última extensão gerada a partir de $\mathbf{e}_{r-1}$ foi $p_1 \ldots p_j \ldots p_{i-1} p_i x p_{i+1} \ldots p_{n-1}$, $1 \leq j <$ i < n – 1. Temos também que p_k ou é o próprio p; ou se encontra à esquerda deste e à direita de p_{j-1} em $\mathbf{e}_{r-1}$, i. e., $\mathbf{j} \leq k \leq$ i. A geração das extensões inserindo x em ***e,*** pode ser feita do seguinte modo : deslizar x para a direita por meio de 1-inserções para a esquerda de um elemento imediatamente à direita de x; inserir x à frente de p_{i-1}; deslizar x para a direita por meio de 1-inserções do mesmo até encontrar p_k. Esta geração é ilustrada na figura 111.3. Reparemos que L pode coincidir com i ou j sem problemas. Se $p_{i-1} \prec$ x, então L = i – 1 e a geração para em †.

- Se $l \neq$ i, isto implica necessariamente em k = l. A última extensão gerada a partir de $\mathbf{e}_{r-1}$ pode ser representada como $p_1 \ldots p_j \ldots p_l x \ldots p_i \ldots p_{n-1}$, $1 \leq l <$ n – 1 e $1 \leq \mathbf{j} < i \leq n - 1$. Como indicam os índices, p_l pode aparecer em qualquer posição relativa a p; ou p_j, e pode coincidir com p_j (mas não com p_i, por suposição). Simplesmente inserimos p_i à frente de p_j e deslizamos x para a direita, até quando possível, por meio de 1-inserções para a esquerda, como ilustra a figura 111.4, onde l foi suposto ser maior ou igual a j sem perda de generalidade. Uma observação relevante é a de que a inserção de p_i à frente de p_j não viola a consistência com as relações de R (com relação ao elemento inserido x), pois x é por suposição inaximal e nunca poderiamos ter x $\prec$ p;. E se p; $\prec$ x, p_i já se encontrava à esquerda de x em $\mathbf{e}_{r-1}$ e foi deslocado mais ainda para a esquerda em e,.

A prova de existência de uma geração somente com inserções para a direita de um poset $\mathcal{P}$ pode ser facilmente obtida a partir de uma geração por inserção para esquerda (cuja existência acabamos de provar) de seu dual $\mathcal{P}'$. Podemos estabelecer uma correspondência biunívoca entre as extensões ***e)*** E $E(\mathcal{P}')$ e as extensões $\mathbf{e}_k \in E(\mathcal{P})$, tal que se $\mathbf{e}'_k = p_1 p_2 \ldots p_n$ então $\mathbf{e}_k = p_n p_{n-1} \ldots p_1$, $\forall\, 1 \leq k \leq e(\mathcal{P}')$. Cada inserção para a esquerda de um elemento p; á frente de outro elemento p_j em uma extensão $\mathbf{e}'_k$ corresponcle a uma inserção para direita de p_i após p_j em $\mathbf{e}_k$, $1 \leq j <$ i $\leq$ n e $1 \leq$ L $\leq e(\mathcal{P}')$. Assim, uma geração para a esquerda em $\mathcal{P}'$ corresponde a uma geração para a direita em $\mathcal{P}$, se tomamos as extensões correspondentes. O

- Se $j \neq 1$ e $k \neq i$:

$$p_1 \cdots p_i p_j \cdots p_k \cdots p_{i-1} x p_{i+1} \cdots p_{n-1}$$
$$p_1 \cdots p_i p_j \cdots p_k \cdots p_{i-1} p_{i+1} x \cdots p_{n-1}$$
$$\vdots$$
$$p_1 \cdots p_i p_j \cdots p_k \cdots p_{i-1} p_{i+1} \cdots p_{n-1} x \quad \dagger$$
$$p_1 \cdots p_i p_j \cdots p_k \cdots x p_{i-1} p_{i+1} \cdots p_{n-1}$$
$$p_1 \cdots p_i p_j \cdots p_k \cdots x p_{i-2} p_{i-1} p_{i+1} \cdots p_{n-1}$$
$$\vdots$$
$$p_1 \cdots p_i p_j \cdots p_k x \cdots p_{i-1} p_{i+1} \cdots p_{n-1}$$

- Se $j = 1$ e $k \neq i$:

$$p_i p_1 \cdots p_k \cdots p_{i-1} x p_{i+1} \cdots p_{n-1}$$
$$p_i p_1 \cdots p_k \cdots p_{i-1} p_{i+1} x \cdots p_{n-1}$$
$$\vdots$$
$$p_i p_1 \cdots p_k \cdots p_{i-1} p_{i+1} \cdots p_{n-1} x \quad \dagger$$
$$p_i p_1 \cdots p_k \cdots x p_{i-1} p_{i+1} \cdots p_{n-1}$$
$$\vdots$$
$$p_i p_1 \cdots p_k x \cdots p_{i-1} p_{i+1} \cdots p_{n-1}$$

- Se $j \neq 1$ e $k = i$:

$$p_1 \cdots p_i p_j \cdots p_{i-1} x p_{i+1} \cdots p_{n-1}$$
$$p_1 \cdots p_i p_j \cdots p_{i-1} p_{i+1} x \cdots p_{n-1}$$
$$\vdots$$
$$p_1 \cdots p_i p_j \cdots p_{i-1} p_{i+1} \cdots p_{n-1} x \quad \dagger$$
$$p_1 \cdots p_i p_j \cdots x p_{i-1} p_{i+1} \cdots p_{n-1}$$
$$\vdots$$
$$p_1 \cdots p_i x p_j \cdots p_{i-1} p_{i+1} \cdots p_{n-1}$$

- Se $j = 1$ e $k = i$:

$$p_i p_1 \cdots p_{i-1} x p_{i+1} \cdots p_{n-1}$$
$$p_i p_1 \cdots p_{i-1} p_{i+1} x \cdots p_{n-1}$$
$$\vdots$$
$$p_i p_1 \cdots p_{i-1} p_{i+1} \cdots p_{n-1} x \quad \dagger$$
$$p_i p_1 \cdots x p_{i-1} p_{i+1} \cdots p_{n-1}$$
$$\vdots$$
$$p_i x p_1 \cdots p_{i-1} p_{i+1} \cdots p_{n-1}$$

Figura 111.3: Se $l =$ i.

- Se $i \neq n-1$:

$$p_1 \ldots p_i p_j \ldots p_l x \ldots p_{n-1}$$
$$p_1 \ldots p_i p_j \ldots p_l p_{l+1} x \ldots p_{n-1}$$

$$p_1 \ldots p_i p_j \ldots p_l \ldots x p_{n-1}$$
$$p_1 \ldots p_i p_j \ldots p_l \ldots p_{n-1} x$$

- Se $i = n-1$:

$$p_1 \ldots p_{n-1} p_j \ldots p_l x \ldots p_{n-2}$$
$$p_1 \ldots p_{n-1} p_j \ldots p_l p_{l+1} x \ldots p_{n-2}$$
$$\ddots$$
$$p_1 \ldots p_{n-1} p_j \ldots p_l \ldots x p_{n-2}$$
$$p_1 \ldots p_{n-1} p_j \ldots p_l \ldots p_{n-2} x$$

Figura 111.4: Se $l \neq$ i. p_l é suposto, sem perda de generalidade, não à esquerda de p_j.

Capitulo IV

Geração de Extensões Lineares por Transposição

Neste capítulo continuaremos a abordar o problema da geração de todas as extensões lineares compatíveis com dado poset, mas levando em consideração a seguinte restrição : duas permutações geradas sucessivamente diferem apenas pela transposição de dois de seus elementos. Se conseguimos listar (gerar) as extensões desta maneira, dizemos que estas foram *geradas por transposição.* Se, além de geradas por transposição, as extensões forem geradas somente por transposições adjacentes (ou seja, as transposições ocorrem sempre entre dois elementos consecutivos de uma extensão) teremos um caso ainda mais particular de *geração* por *transposição adjacente.* Vale notar que esta geração por transposição não é tão trivial , só sendo conhecida para algumas subclasses bem restritas de posets. Por exemplo, no caso em que o poset é constituído por duas cadeias disjuntas, só obteremos êxito nesta geração se o comprimento de ambas as cadeias for ímpar, com exceção dos casos triviais, enumerados no teorema 7 (veja Buck & Wiedemann [BW84], Eades, Hicky & Read [EHR84], ou Ruskey [Rus88a]). O algoritmo clássico de Steinhaus-Johnson-Trotter se utiliza de transposições adjacentes para gerar todas as permutações de um conjunto de n elementos, como visto na seção 111.1.

Mais uma vez buscando uma analogia com algoritmos de ordenação, transposições são nestes frequentemente chamadas de "trocas" (Knuth [Knu68]) e desempenham papel vital em algoritmos de ordenação por comparação. Em computação paralela, tais algoritmos são de grande interesse, uma vez que podemos efetuar até $n/2$

transposições simultaneamente para um conjunto qualquer com n elementos.

A idéia de geração por transposição está intimamente ligada ao conceito do grafo de transposição $G(\mathcal{P})$ (no caso de transposições adjacentes, $G'(\mathcal{P})$). $G(\mathcal{P})$ é sempre um subgrafo induzido do grafo de transposição que considera todas as permutacões dos elementos de S, enquanto $G'(\mathcal{P})$ é um subgrafo (não necessariamente indúzido) de $G(\mathcal{P})$. Traduzimos agora o problema como sendo o de se achar um caminho hamiltoniano em $G(\mathcal{P})$, ou seja, encontrar um modo de percorrer todas as extensões lineares do poset (vértices de $G(\mathcal{P})$) onde a próxima extensão a ser percorrida difere da anterior por apenas uma transposição (existe aresta entre as duas extensões em $G(\mathcal{P})$). Não é difícil concluir que $G(\mathcal{P})$ é um grafo bipartite, uma vez que uma transposição sempre reverte a paridade de uma permutação. Se o número de vértices nas duas partições diferir por mais de urna unidade ($d(\mathcal{P}) > 1$), então necessariamente não existe caminho hamiltoniano em $G(\mathcal{P})$. Em [Rus88b], Ruskey propõe a seguinte questão :

Coiijectura *1 Seja* $\mathcal{P}$ um *poset balanceado* ($d(\mathcal{P}) = 0$). *Então* $\mathcal{P}$ *admite uma geração por transposição.*

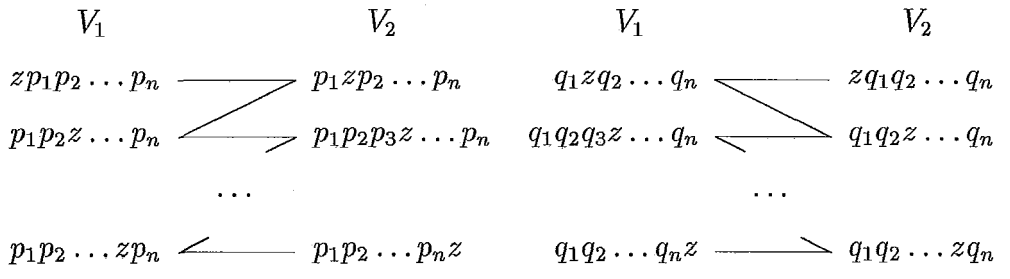
Mesmo sendo válida esta conjectura, ainda não teremos meios para caracterizar completamente os posets que admitem geração por transposição, uma vez que teríamos que considerar sua volta : esta é obviamente falsa para os casos (triviais) quando o poset é constituído por uma única cadeia, ou é constituído por duas cadeias, sendo uma de tamanho par e a outra o poset trivial. A única caracterização efetiva conhecida quanto à admissão desta geração se dá na classe dos multiconjuntos, como mostra o teorema 7.

Para provar que G(P(S,R)) é de fato bipartite, provaremos que o grafo de transposição das permutações irrestritas de n = $|S|$ elementos o é, uma vez que $G(\mathcal{P})$ é um subgrafo induzido deste. Provaremos por indução em n. Denotemos por $\mathcal{P}_n$ o poset com n elementos e conjunto de relações vazio. Se n = 2, $G(\mathcal{P}_2)$ é bipartite. Supondo $G(\mathcal{P}_n)$ bipartite, onde V_1 é o conjunto das permutações pares e V_2 é o conjunto das permutações ímpares segundo uma permutação de $\mathcal{P}_n$ escolhida como canônica, vamos provar para $\mathcal{P}_{n+1}$.

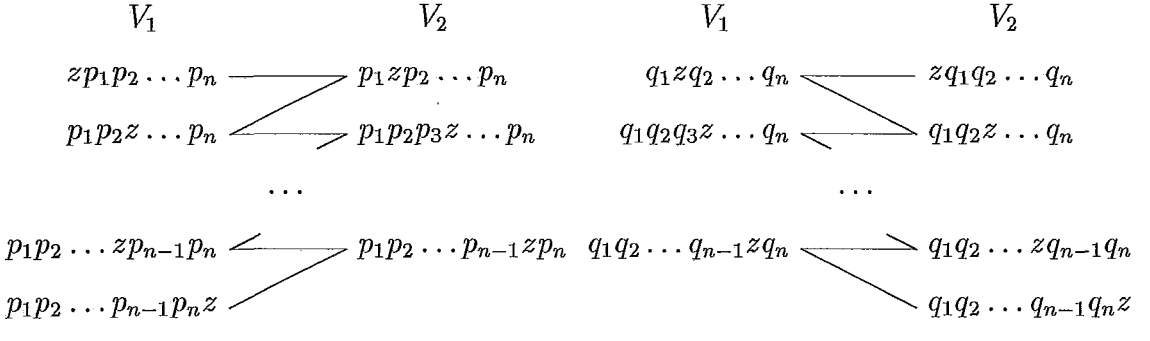
Seja z o elemento de $\mathcal{P}_{n+1}$ que não pertence a $\mathcal{P}_n$. Movendo-se z da esquerda para a direita sobre uma permutação par em $G(\mathcal{P}_n)$ ($p_1 p_2 \ldots p_n \in V_1$) e da direita para a

esquerda sobre uma permutação ímpar ($q_1 q_2 \ldots q_n \in V_2$), teríamos a seguinte divisão dos conjuntos V_1 e V_2 em $G(\mathcal{P}_{n+1})$:

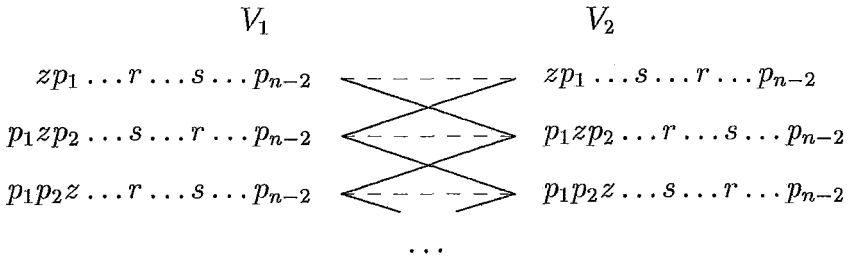
e se n é ímpar :



e se n é par :



Considerando agora que $p = p_1 \ldots r \ldots s \ldots p_{n-2}$ e $q = p_1 \ldots s \ldots r \ldots p_{n-2}$ diferiam por uma transposição em $G(\mathcal{P}_n)$, a saber entre os elementos r e s, temos que nossa nova divisão dos conjuntos de bipartição ainda preserva a condição de bipartidade de $G(\mathcal{P}_{n+1})$ com relação às transposições já existentes em $G(\mathcal{P}_n)$. Vejamos isto por meio do esquema abaixo, onde uma linha tracejada denota uma transposição entre os elementos r e s, enquanto cada linha contínua denota uma transposição adjacente entre z e um outro elemento do poset :



Acabamos de mostras que, segundo esta divisão dos conjuntos V_1 e V_2 para $G(\mathcal{P}_{n+1})$, as arestas que representavam transposições adjacentes entre z e um outro elemento pertencente a $\mathcal{P}_n$ ou as transposições (não necessariamente adjacentes) que envolviam somente elementos de $\mathcal{P}_n$ respeitam a condição de bipartidade em $G(\mathcal{P}_{n+1})$. Resta agora analisar como as transposições não adjacentes entre z e um outro elemento se comportam em relação à divisão dos conjuntos de bipartição.

Seja r o elemento a ser transposto com z, sejam i e j, i $< j$, as posições onde ocorre a transposição; $\mathbf{p} = p_1 \ldots r \ldots z \ldots p_{n-2}$ e $\mathbf{q} = p_1 \ldots z \ldots \mathrm{r} \ldots p_{n-2}$ as respectivas permutações. Denotemos ainda por $\mathbf{p}'$ e $\mathbf{q}'$ as permutações $\mathbf{p}$ e $\mathbf{q}$ sem o elemento z (ou seja, são as permutações correspondentes em $\mathcal{P}_n$).

(i) Se $(j-1)-i$ é par então $\mathbf{p}'$ e $\mathbf{q}'$ pertenciam ambas ao mesmo conjunto de bipartição em $G(\mathcal{P}_n)$, pois para levar r de i a $\mathbf{j-1}$ em $G(\mathcal{P}_n)$ (bipartite por suposição) necessitamos de um número par de transposições adjacentes $(= (j-1) - \mathrm{i})$. Então $\mathbf{j}-\mathrm{i}$ é ímpar, o que coloca $\mathbf{p}$ e $\mathbf{q}$ em conjuntos de bipartição para $G(\mathcal{P}_{n+1})$ distintos, pois as posições relativas de z nessas duas permutações têm paridades diferentes e $\mathbf{p}'$ e $\mathbf{q}'$ pertenciam ambas ao mesmo conjunto de bipartição em $G(\mathcal{P}_n)$ (ver divisão dos conjuntos de bipartição para $G(\mathcal{P}_{n+1})$ feita acima).

(ii) Se $(j-1)-\mathrm{i}$ é ímpar, $\mathbf{p}'$ e $\mathbf{q}'$ pertenciam a conjuntos distintos em $G(\mathcal{P}_n)$. Como $j-\mathrm{i}$ é par, p e q são também colocadas em conjuntos diferentes em $G(\mathcal{P}_{n+1})$, o que era de se esperar.

Por indução acabamos de provar que o grafo de transposição (ou transposição adjacente) de qualquer poset é bipartite. Reparem que os conjuntos V_1 e V_2 também representam os conjuntos das permutações pares e ímpares em $G(\mathcal{P}_{n+1})$.

Lema **2** *Seja* $\mathcal{P}(S, \mathrm{R})$ *um poset e* z *um elemento distinto de qualquer elemento de* S. *Se* $G(\mathcal{P})$ *tem um caminho (ciclo) hamiltoniano, então* $G(\mathcal{P} \cup \{\mathrm{z}\})$ *também o tem.*

Prova : Sejam $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N$ as extensões lineares sucessivas ao longo do caminho (ciclo) hamiltoniano em $G(\mathcal{P})$. Suponha que $|S| = \mathrm{n}$. Então $\mathbf{p}_1 z$ se torna $z\mathbf{p}_1$ após n transposições adjacentes, onde cada transposição move z uma posição para a esquerda. Dizemos que z foi "deslizado da direita para a esquerda" sobre p;. Agora substitua $z\mathbf{p}_1$ por $z\mathbf{p}_2$ e desloque z da esquerda para a direita sobre $\mathbf{p}_2$. Continuamos desta maneira : deslizando z sobre p; da direita para a esquerda quando i é ímpar, e da esquerda para a direita quando i é par. Sendo $G(\mathcal{P})$ bipartite, se o mesmo admite um ciclo hamiltoniano então N tem que necessariamente ser par. Portanto as extensões inicial e final do caminho hamiltoniano em G(P∪{z}) são $\mathbf{p}_1 z$ e $\mathbf{p}_N z$, as quais diferem por uma transposição (a saber a que levava de $\mathbf{p}_1$ a $\mathbf{p}_N$ no ciclo em $G(\mathcal{P})$) e existe um ciclo hamiltoniano em G(P ∪ {z}). o

O seguinte lema pode ser provado analogarnente ao lema 2, bastando que $G(\mathcal{P})$ seja substituído por $G'(\mathcal{P})$.

Lema 3 Seja $\mathcal{P}(S, \mathrm{R})$ *um poset e* z *um elemento distinto de qualquer elemento de* S. *Se* $G'(\mathcal{P})$ *tem um caminho (ciclo) hamiltoniano, então* $G'(\mathcal{P} \cup \{\mathrm{z}))$ *também o tem.*

Aplicando-se repetidamente o processo de construção do caminho hamiltoniano induzido pelo lema acima ao poset com n elementos e conjunto de relações vazio, obtemos o algoritmo de Steinhaus-Johnson-Trotter para gerar as n! permutações de n elementos.

A seguir caracterizaremos subclasses de posets para os quais é possível gerar todas suas extensões lineares por transposição, ou até mesmo por transposição adjacente. As provas dos teoremas e lemas que levam a esta geração são todas construtivas, ou seja, provamos a existência de uma geração por transposição mostrando como achar um caminho hamiltoniano no grafo de transposição (transposição adjacente). Para a última subclasse a ser apresentada (seção IV.3), Pruesse & Ruskey [PR88] mostram um algoritmo (e sua implementação) que gera as extensões por transposição e em tempo médio constante.

1 Florestas

Se o diagrama de Hasse de um poset é uma floresta enraizada, então as relações de recorrência (11.1) e (11.2) podem ser resolvidas. As raízes das árvores que constituem a floresta são obviamente os elementos maximais do poset. Uma floresta com t árvores é denotada por :

$$\mathcal{F} = \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t$$

onde cada $\mathcal{T}_i$ é uma árvore da floresta. Ao remover a raiz de uma árvore $\mathcal{T}_i$ obtemos uma floresta que denotaremos por $s(\mathcal{T}_i)$, introduzindo também a seguinte notação :

$$\mathcal{F}_i = \mathcal{T}_1, \ldots, s(\mathcal{T}_i), \ldots, \mathcal{T}_t$$

A raiz de uma árvore sempre aparece à direita de seus descendentes em uma extensão linear. Se a floresta é constituída apenas por árvores-cadeia, então teremos permutações em multiconjuntos, como veremos a seguir. Reescrevendo as fórmulas (11.1) e (11.2) para florestas :

$$e(\mathcal{F}) = \begin{cases} 1, & \text{se } t = 1 \text{ e } |\mathcal{T}_1| = 1; \\ \sum_{i=1}^{t} e(\mathcal{F}_i), & \text{caso contrário.} \end{cases} \qquad \text{(IV.1)}$$

$$d(\mathcal{F}) = \begin{cases} 1, & \text{se } t = 1 \text{ e } |\mathcal{T}_1| = 1; \\ \sum_{i=1}^{t} (-1)^{|\mathcal{T}_1|+\ldots+|\mathcal{T}_{i-1}|} d(\mathcal{F}_i), & \text{caso contrário.} \end{cases} \qquad \text{(IV.2)}$$

Suponhamos que existam n nós na floresta e que o número de descendentes do nó i é dado por d_i (o nó i é descendente dele próprio). Veremos no capítulo VII que o número de extensões de um poset floresta é dado por :

$$e(\mathcal{F}) = \frac{n!}{d_1 \ldots d_n} \qquad \text{(IV.3)}$$

Um *emparelhamento perfeito* (ou *1-fator)* de um grafo $G(V, E)$ com n vértices é uma coleção de arestas $\{(v_i, w_i) \mid 1 \leq i \leq n/2\} \subseteq E$, tal que todo vértice $v \in V$ ocorre *exatamente uma vez* como extremo de alguma aresta nesta coleção.

Definição *4 Uma floresta $\mathcal{F}$ admite um* pareamento *se existe um emparelhamento perfeito em $\mathcal{F}$ ou $\mathcal{F}_i$, para algum i.*

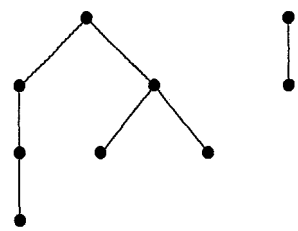


Figura IV.1: Não existe pareamento possível; $d(\mathcal{P}) = 0$.

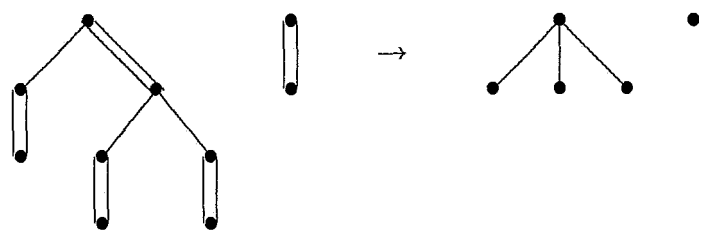


Figura IV.2: Pareamento e floresta colapsada; $d(\mathcal{P}) = 30$.

O pareamento, se existir, é único e pode ser construído de "cima para baixo", partindo-se das folhas para as raízes.

Definição 5 *A* floresta aglutinada $\overline{\mathcal{F}}$ *de uma floresta* $\mathcal{F}$ *que admite um pareamento é a floresta obtida ao se contrair as arestas emparelhadas de* $\mathcal{F}$ *e remover a raiz não emparelhada (se existir uma).*

Lema 4 Para uma floresta $\mathcal{F}$*, a diferença de paridade pode ser expressa como*

$$d(\mathcal{F}) = \begin{cases} e(\overline{\mathcal{F}}), & \text{se } \mathcal{F} \text{ tem um pareamento;} \\ 0, & \text{caso contrário.} \end{cases} \tag{IV.4}$$

A prova do lema acima pode ser encontrada em [Rus88a].

Acredita-se que sempre que $d(\mathcal{F}) = 0$, existe um caminho hamiltoniano em $G(\mathcal{F})$, reforçando a conjectura 1 para posets balanceados. Todo poset floresta onde não existe vértice com exatamente um filho admite uma geração por transposição (garante um caminho hamiltoniano em $G(\mathcal{F})$). Este resultado é uma conseqüência direta do resultado de Pruesse & Ruskey [PR88] (ver seção IV.3) para uma certa subclasse dos posets graduados, onde as florestas se encontram propriamente incluídas. Pruesse & Ruskey definiram um algoritmo de geração de tempo médio constante para esta subclasse dos posets graduados.

O teorema a seguir caracteriza casos de posets floresta que admitem um ciclo hamiltoniano. Seriam estes os casos onde não existe vértice com exatamente um filho e o poset floresta não se encaixa em nenhuma das exceções citadas no teorema. O teorema 6 é uma generalização da caracterização de ciclos em florestas apresentada por Ruskey [Rus88a]. Este teorema poderia, de outro modo, ter sido provado com base nos resultados de Pruesse & Ruskey [PR88].

Teorema 6 *Seja $\mathcal{F}$ um poset do tipo floresta tal que $\mathcal{F}$ não é o poset trivial ou o poset constituido por dois elementos isolados, nem uma árvore única constituída exatamente por duas folhas filhas de um pai comum. Se em $\mathcal{F}$ não existe vértice com exatamente um filho, então $G(\mathcal{F})$ admite ciclo hamiltoniano.*

Prova : Se $\mathcal{F}$ é uma floresta constituída por apenas uma árvore, então as permutações dos filhos da raiz podem ser geradas por transposições adjacentes usando o algoritmo de Steinhaus-Johnson-Trotter; a raiz da árvore deve ser inserida à direita de todos seus filhos em cada extensão. Se $\mathcal{F}$ é constituída por mais de uma árvore, as permutações das raízes destas árvores também podem ser geradas através do algoritmo de Steinhaus-Johnson-Trotter. Se $\mathcal{F}$ satisfaz as condições do teorema, então teremos neste estágio um número par de extensões.

Expanda a floresta de cima para baixo, adicionando filhos a um vértice já considerado x. Sejam $z_1, z_2, \ldots, z_m$, m > I, os filhos de x. Considere a permutação genérica $\alpha x \beta$ antes da adição dos filhos de x ao poset. Desejamos inserir os filhos de x de todas as maneiras possíveis em a. Em primeiro lugar inserimos dois filhos z_1 e z_2, gerando as novas extensões sucessivamente por meio de transposições, de modo que a extensão inicial seja $z_1 z_2 \alpha x \beta$ e a extensão final $z_2 z_1 \alpha x \beta$. Tal caminho é possível e pode ser

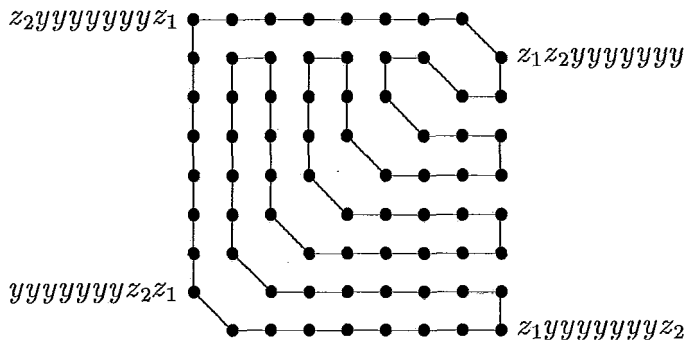


Figura IV.3: Ciclo hamiltoniano para n ímpar (n = 7).

visto nas figuras IV.3 e IV.4, onde n = $|\alpha|$. Os elementos de a são representados por n y's; os vértices representam todas as maneiras de se inserir z_1 e z_2 em a. Na parte triangular inferior z_1 encontra-se à esquerda de z_2, enquanto que na superior z_2 se encontra sempre à esquerda de z_1. Arestas horizontais representam transposições entre z_1 e um y; arestas verticais transposições entre z_2 e um y, e arestas diagonais transposições entre z_1 e z_2.

Os filhos remanescentes $z_3, \ldots, z_m$ são incluídos no poset, "deslizando" um filho de cada vez sobre os elementos que se encontram à esquerda de x nas extensões que vão sendo geradas, usando o princípio de Steinhaus-Johnson-Trotter. A permutação inicial após a adição de todos os filhos de x é $z_m \ldots z_3 z_1 z_2 \alpha x \beta$ e a permutação final, $z_m \ldots z_3 z_2 z_1 \alpha x \beta$. É fácil verificar que com a inserção de z_1 e z_2 obtivemos um número par de extensões, que se manterá par após a inserção dos outros filhos $z_3, \ldots, z_m$ de x. Se $\alpha' x \beta'$ é a permutação que segue $\alpha x \beta$, então procedemos como acima, mas com os papéis de z_1 e z_2 trocados. Ou seja, as permutações inicial e final após a inserção dos filhos de x seriam respectivamente $z_m \ldots z_3 z_2 z_1 \alpha' x \beta'$ e $z_m \ldots z_3 z_1 z_2 \alpha' x \beta'$. Como tínhamos antes da inclusão dos filhos de x um número par de extensões, após a geração por transposição das novas extensões que incluem os mesmos teremos extensões inicial e final diferindo também por uma transposição e o ciclo hamiltoniano poderá ser fechado. o

Note-se que a prova do teorema acima não provê um ciclo hamiltoniano em $G'(\mathcal{F})$

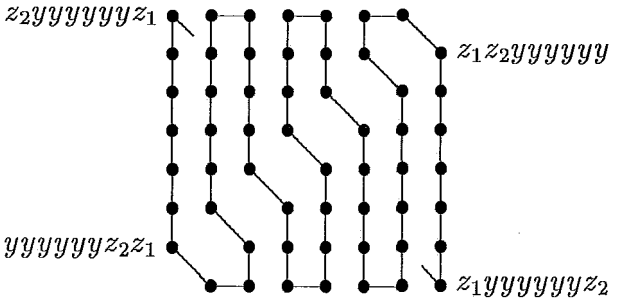


Figura IV.4: Ciclo hamiltoniano para n par (n = 6).

simplesmente por causa da transposição que leva de $z_1\alpha z_2$ a $z_2\alpha z_1$ usada na construção dos ciclos como o da figura IV.4, que não é uma transposição adjacente. Aliás, nada ainda se pode afirmar sobre a existência ou não de caminho em $G'(\mathcal{F})$, mesmo se $\mathcal{F}$ respeita as condições do teorema acima.

IV.2 Multiconjuntos

Nesta seção consideraremos posets constituídos apenas por cadeias disjuntas. $\mathcal{C}_k$ denota uma cadeia de k elementos. Tal poset fica completamente determinado pelo tamanho de cada uma de suas cadeias, portanto usamos a notação $\mathbf{n} = n_1, n_2, \ldots, n_t$ para um poset constituído de t cadeias de tamanhos $n_1, n_2, \ldots, n_t$. É óbvio (segue direto da fórmula (IV.3)) que $e(\mathbf{n}) = \frac{n!}{n_1!n_2!\ldots n_t!}$, onde $n = n_1 + \ldots + n_t$. Aqui também conseguimos resolver a equação de recorrência (11.2). Isto não é nenhuma surpresa, uma vez que multiconjuntos nada mais são do que uma subclasse com propriedades muito particulares de florestas. A equação de recorrência para a diferença de paridade em multiconjuntos foi resolvida em [KR88] e [Rus87], mas pode também ser derivada diretamente do lema 4.

Lema 5 *A diferença de paridade para multiconjuntos é dada por*

$$d(\mathbf{n}) = \begin{cases} e(\mathbf{m}), & \textit{se o número de } n_i's \textit{ ímpares} \leq 1; \\ 0, & \textit{caso contrário.} \end{cases} \qquad \text{(IV.5)}$$

onde $m_i = \lfloor n_i/2 \rfloor$, $1 \leq i \leq t$, *e* $\mathbf{m} = m_1, m_2, \ldots, m_t$.

O lema 5 estabelece a condição de necessidade do teorema 7, ou seja, a diferença de paridade vai exceder um se n tem menos que duas cadeias de comprimento ímpar, a não ser nos casos triviais.

Teorema 7 *Permutações em multiconjuntos podem ser geradas por transposição se e somente se pelo menos dois tipos de elementos têm multiplicidade ímpar, com exceção dos casos triviais onde o poset ou é constituído por um único tipo de elemento, ou existem dois tipos de elementos e um deles ocorre com multiplicidade* **1.**

Pelo teorema enunciado acima, mostramos que $G(\mathbf{n})$ tem um caminho hamiltoniano se n é balanceado. Podemos prová-lo indutivamente no número de cadeias e no comprimento destas, tomando como base da indução o poset constituído de duas cadeias de um elemento (ou seja, dois elementos isolados). O teorema é obviamente válido no caso de multiconjuntos com única cadeia (único tipo de elemento). A prova deste teorema é muito extensa e portanto não nos cabe aqui apresentá-la; entretanto, é uma prova interessante, por ser construtiva, e pode ser encontrada na íntegra em [Rus88a]. Um caminho hamiltoniano em $G(\mathbf{n})$ será obtido se o modificamos de modo a construir um caminho hamiltoniano em $G(\mathbf{n}')$, onde $\mathbf{n}$ é o poset obtido ao se aumentar o comprimento de alguma cadeia de $\mathbf{n}'$ de duas unidades ou ao se adicionar a $\mathbf{n}'$ uma nova cadeia de comprimento menor ou igual a dois.

Os caminhos hamiltonianos construídos desta forma em [Rus88a] são chamados de *sequencialmente adjacentes*. Ou seja, qualquer transposição entre dois elementos x e y das cadeias i e j respectivamente, onde $i < j$, satisfaz à restrição de que qualquer elemento z entre x e y na extensão pertence a uma cadeia de índice maior que j. Em particular, os elementos da última cadeia (de tamanho n_t) só participam de transposições adjacentes.

Continua, entretanto, em aberto o problema da existência de um caminho hamiltoniano em G^1(n) toda vez que n for balanceado. Ruskey [Rus88a] suporta esta conjectura. Conjecturamos também a existência de um ciclo hamiltoniano em $G(\mathbf{n})$ se $\mathbf{n}$ for balanceado e contiver pelo menos 3 cadeias ($t > 2$) [Rus88a]. De fato, não podemos ter tal ciclo em $G(\mathbf{n})$ quando $t = 2$, uma vez que teríamos dois vértices "pendentes" (adjacentes a somente uma aresta) neste grafo. Notemos que G ser

balanceado (e conseqüentemente G') já é de fato uma conclição necessária para a existência de um ciclo harniltoniano no poset, uma vez que G é bipartite.

IV.2.1 Ciclos e Caminhos Hamiltonianos em $G'(\mathbf{n})$

Existem ainda muitas questões em aberto acerca da geração por transposição adjacente para multiconjuntos. Conjecturamos acima a existência de um caminho em G (n) toda vez que $\mathbf{n}$ for balanceado. O lema a seguir mostra um caso onde não existem vértices "pendentes" e G admite ciclo hamiltoniano, mas onde, mesmo assim, G' não comporta este ciclo.

Lema 6 O *grafo* $G'(1,1,\mathrm{n})$ *possui um ciclo hamiltoniano se e somente se* n é *ímpar.*

Os grafos $G'(1,1,\mathrm{n})$ com n par, e $G'(n,\mathrm{m}) = G(n,m)$ (é um poset de largura 2) com n e m ímpares, são os únicos exemplos conhecidos onde existe um caminho hamiltoniano em $G'(\mathbf{n})$ mas não um ciclo. Acredita-se que estes grafos sejam de fato as únicas exceções. Os resultados abaixo foram todos apresentados por Ruskey [Rus88a], com exceção dos corolários 2 e **3.** Todos estes resultados tem como base o lema 7, onde uma cadeia de dois elementos é adicionada a um poset balanceado que possua caminho (ciclo) hamiltoniano. Esta restrição quanto à validade do lema 7 somente para posets balanceados reduz de maneira bastante significativa a aplicabilidade deste lema.

Lema 7 *Seja* $\mathcal{P}$ *um poset balanceado. Se* $G'(\mathcal{P})$ *possui um caminho hamiltoniano, então* $G'(\mathcal{P} \cup \mathcal{C}_2)$ *também o possui. Se* $|\mathcal{P}| \geq \mathbf{3}$ *e* $G'(\mathcal{P})$ *possui um ciclo hamiltoniano, então* $G'(\mathcal{P} \cup \mathcal{C}_2)$ *também o possui.*

Prova : Assumamos que $\mathcal{P}(S,\mathrm{R})$ é um poset que satisfaz as condições do lema. Agrupamos as extensões par a par segundo um caminho hamiltoniano em $\mathcal{P}$ (a partir de uma de suas extremidades). Seja (α,β) um desses pares de extensões tal que $\mathbf{a} = \gamma xy\delta$ e $\beta = \gamma yx\delta$, onde γ, δ são extensões parciais de $\mathcal{P}$ e x, y E S. Ao inserir os elementos de $\mathcal{C}_2$, os quais denotaremos por z's, produzimos um grafo de transposição adjacente $H_{n,k}$, onde n = $|\mathcal{P}|$ e $k = |\gamma x|$, isomorfo ao grafo G[1](a(x $\cup$ y)δ + $\mathcal{C}_2$).

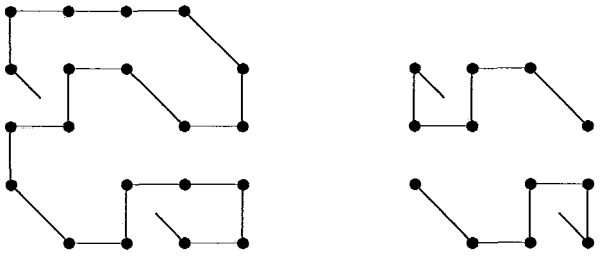


Figura IV.5: (a) Ciclo hamiltoniano em $H_{3,1}$, (b) Caminho hamiltoniano em $H_{2,1}$.

Agora mostramos que $H_{n,k}$ possui um ciclo hamiltoniano se n $\geq$ **3.** O argumento é indutivo, mas foi apresentado informalmente na figura IV.6. A figura IV.5(a) mostra que $H_{3,1} \approx H_{3,2}$ possui um ciclo hamiltoniano. A idéia básica é que o ciclo construído em $H_{n,k}$ pode ser modificado de modo a se obter ciclos em $H_{n+1,k}$ e $H_{n+1,k+1}$, como mostrado na figura IV.6.

Se $|\mathcal{P}| = 2$, então $\mathcal{P}$ consiste de dois elementos não relacionados e o caminho hamiltoniano correspondente é o ilustrado na figura IV.5(b). Se $|\mathcal{P}| \geq$ **3,** então usamos os ciclos harniltonianos construídos segundo as figuras IV.5(a) e IV.6 para cada par de vértices do ciclo. Não há problemas nas interfaces entre os pares selecionados, pois os z's se encontram na extremidade direita de uma extensão neste momento. A afirmação anterior é válida, uma vez que os ciclos em $H_{n,k}$ sempre incluem a aresta que liga αzz a βzz. Esta aresta é então removida de modo a se obter um caminho hamiltoniano que leva de αzz a βzz. O

Corolário 1 *O grafo* $G'(n, \mathrm{m}, 2)$ *possui caminho hamiltoniano se e somente se* n *e* m *são ímpares.*

Prova : Se n e m não são ambos ímpares, temos pelo teorema 7 que não existe caminho harniltoniano em $G(n, \mathrm{m}, 2)$ e, portanto, também não em $G'(n, \mathrm{m}, 2)$. Se n e m são ambos ímpares, $G'(n, \mathrm{m})$ satisfaz as condições do lema 7 (pois $G'(n, \mathrm{m}) = G(n, m)$). O

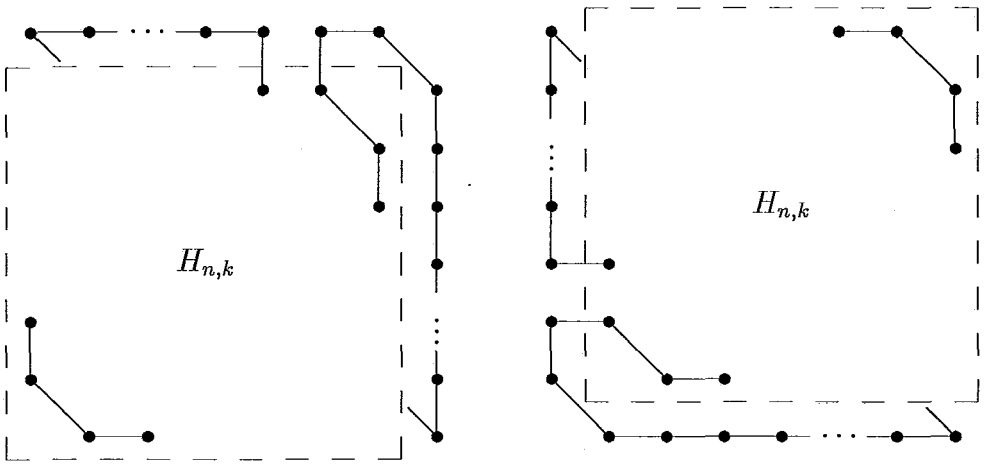


Figura IV.6: Estendendo ciclos hainiltonianos em $H_{n,k}$ para $H_{n+1,k}$ e $H_{n+1,k+1}$.

A seguir mostramos duas novas extensões de resultados para subclasses de multiconjuntos que admitem geração por transposição adjacente.

Corolário 2 *O grafo* $G'(1,2,\mathrm{m},\mathrm{n})$ *possui caminho hamiltoniano se* m, n *são Zmpares.*

Este corolário segue trivialmente do corolário 1 e do lema **3.** O interessante é que a volta neste caso é facilmente provada falsa : basta o contra-exemplo $G'(1,1,2,2)$ que possui, inclusive, ciclo hamiltoniano.

Corolário 3 *O grafo* $G'(m,\mathrm{n},2,2)$ *possui caminho hamiltoniano se e somente se* m, n *são Zmpares.*

OBS. : Se $\mathrm{m},\mathrm{n}=1$, caímos no caso do teorema 8.

Prova : $G'(m,\mathrm{n},2)$ possui caminho hamiltoniano e é balanceado. Então, pelo lema 7, $G'(m,\mathrm{n},2,2)$ possui caminho hamiltoniano.

Se m ou n é par então, pelo teorema 7, $G(m,\mathrm{n},2,2)$ não admite caminho hamiltoniano, assim como não o pode admitir $G'(m,\mathrm{n},2,2) \subseteq G(m,\mathrm{n},2,2)$. □

Teorema 8 *O grafo $G'(1,1,m,n)$ possui ciclo hamiltoniano para quaisquer $m,n \geq 1$.*

IV.3 Posets Graduados

Mais uma evidência que suporta a conjectura de que todo grafo de transposição de um poset balanceado possui caminho hamiltoniano será apresentada nesta seção, que trata de posets graduados (como já definidos). Esta classe de posets engloba os posets floresta (seção IV.1), que por sua vez englobam os multiconjuntos (seção IV.2).

Teorema 9 *As extensões lineares de qualquer poset graduado onde todo elemento não-maximaí tem ao menos dois elementos que o cubram podem ser geradas por transposição.*

O teorema acima representa o resultado principal do trabalho de Pruesse & Ruskey em [PR88]. Usando o lema 9, conseguimos uma prova bem sucinta para este teorema. O lema 8 [PR88] será usado na prova do lema 9.

Lema 8 *Existe uma lista $\Gamma_n = \mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_{n!}$ das permutações de $1,2,\ldots,\mathrm{n}$, $\mathrm{n} \geq 2$, tal que :*

(a) se i é ímpar então p_i e p_{i+1} diferem pela transposição de seus dois primeiros elementos;

(b) se i é par então p_i e p_{i+1} diferem por uma transposição, e

(c) p_1 e $\mathrm{p}_{n!}$ diferem por uma transposição.

Lema 9 *Seja $\mathcal{P}$ um poset e M um subconjunto de seus elementos minimais tal que $G(\mathcal{P} - M)$ admite um caminho (ciclo) hamiltoniano. Se nenhum elemento de $\mathcal{P} - M$ tem exatamente um descendente em M, então $G(\mathcal{P})$ admite um caminho (ciclo) hamiltoniano.*

Antes de provar este lema, sejam feitas algumas observações com relação às condições impostas por ele. Antes de tudo, a condição que diz que nenhum elemento de

$\mathcal{P} - M$ pode ter exatamente um descendente em M não é equivalente à condição de que nenhum elemento de $P - M$ cubra exatamente um outro elemento em M. Um elemento pode cobrir exatamente um elemento em M e, ainda assim, ter mais de um descendente em M. Em segundo lugar, notemos que se $|M| > 1$ e $\mathcal{P}$ satisfaz às condições do lema, então $G(\mathcal{P})$ tem que ser balanceado : um emparelhamento perfeito em $G(\mathcal{P})$ pode ser definido pela transposição dos elementos de M mais à esquerda em uma extensão linear.

Agora apresentemos a prova do lema 9.

Prova : Sejam P e M satisfazendo as condições do lema, onde M tem m elementos e $\mathcal{P} - M$ possui n elementos. Se $|M| = 1$, o elemento único a de M não cobre nenhum outro elemento de P. Então, podemos usar a idéia de Steinhaus-Johnson-Trotter de se deslizar *a* para a esquerda e para a direita sobre as permutações segundo o caminho (ciclo) hamiltoniano em G(P – M) gerando um caminho (ciclo) hamiltoniano em $G(\mathcal{P})$. Assumamos n > 1.

Construiremos o caminho hamiltoniano em três etapas. Cada etapa produz uma lista de extensões lineares, que será expandida na etapa seguinte. A primeira etapa consiste em se listas os vértices segundo um caminho (ciclo) hamiltoniano em G(P – M). Denotemos esta lista por

$$q_1, q_2, \ldots, q_s \text{ onde } s = e(P - M).$$

Na segunda etapa, substituimos cada q_j por uma lista de todas as extensões lineares de $\mathcal{P}$ da forma pq_j, onde p é uma permutação dos elementos de M. Seja $p_1, p_2, \ldots, p_r$, onde r = m!, a lista produzida pelo lema 8. Então, esta etapa produz iterativamente, para $i = 1, 2 \ldots, s$, as extensões lineares

$$p_1q_i, p_2q_i, \ldots, p_rq_i, \text{ se i é ímpar, e}$$
$$p_rq_i, \ldots, p_2q_i, p_1q_i, \text{ se } i \text{ é par.}$$

Cada par de extensões lineares sucessivas difere por uma tranposição. Ao final da etapa dois, temos m! e(P – M) extensões lineares na lista. A primeira delas é p_1q_1. Se $e(P - M)$ é ímpar, então a extensão final na lista é p_rq_s. Se é par, então a extensão final é p_1q_s. Portanto, se tinhamos um ciclo hamiltoniano em G(P – M), a primeira e a Última extensão da lista no final desta etapa ainda diferem por uma transposição.

Na terceira etapa, intercalamos os elementos de M e $\mathcal{P} - \mathrm{M}$. Seja $\mathrm{p} = p_1 p_2 \ldots p_m$ $q_1 q_2 \ldots q_n$ uma extensão linear da lista e seja

$$\mathcal{Q} = \mathcal{P}[\mathbf{p}] = \mathcal{P}[p_1 \ldots p_m;\ q_1 \ldots q_n]$$

a extensão do poset $\mathcal{P}(S, \mathrm{R})$ com mesmo conjunto de elementos S e com ordem parcial dada pelo fecho transitivo de

$$R \cup \{(p_i, p_{i+1}) \mid i = 1, 2, \ldots, m-1\} \cup \{(q_i, q_{i+1}) \mid i = 1, 2, \ldots, n-1\}$$

O poset $\mathcal{Q}$ é um B-poset (ver seção II.3.1). Consideremos duas extensões sucessivas ao final da etapa dois que difiram pela transposição de seus dois primeiros elementos. Denotemo-as por $\mathrm{p} = p_1 p_2 \ldots p_m q_1 q_2 \ldots q_n$ e $\mathbf{p}' = p_2 p_1 \ldots p_m q_1 q_2 \ldots q_n$. Sejam

$$\begin{aligned} \mathcal{P}_1 &= \mathcal{P}[p_1 p_2 \ldots p_m;\ q_1 \ldots q_n] = \mathcal{P}[\mathbf{p}] \text{ e} \\ \mathcal{P}_2 &= \mathcal{P}[p_2 p_1 \ldots p_m;\ q_1 \ldots q_n] = \mathcal{P}[\mathbf{p}']. \end{aligned}$$

Usaremos $\prec_1$ ($\prec_2$) para denotar $\prec_{\mathcal{P}_1}$ ($\prec_{\mathcal{P}_2}$), por simplicidade. Afirmamos que $\mathcal{P}_1$ e $\mathcal{P}_2$ são B-posets isomorfos, e que este isomorfismo é dado por uma função de transposição. Somente nos preocuparemos com relações envolvendo p_1 ou p_2 e um elemento de $\mathcal{P} - M$. Se $p_1 \prec_1 q_j$, então $\exists\, i > 1$ tal que $p_i \prec_1 q_j$, caso contrário q_j teria um único descendente em M. Sendo i = 2 ou, senão, por transitividade, temos $p_2 \prec_1 q_j$. Se $p_2 \prec_1 q_j$ então obviamente $p_1 \prec_1 q_j$. Analogamente, $p_1 \prec_2 q_j$ se e somente se $p_2 \prec_2 q_j$. Notemos que p_1 não é coberto por nenhum elemento de $\mathcal{P} - \mathrm{M}$ em $\mathcal{P}_1$, assim como p_2 não o é em $\mathcal{P}_2$. Os elementos p_1 e p_2 podem ser transpostos em qualquer extensão linear de $\mathcal{P}_1$ de modo a se obter uma extensão de $\mathcal{P}_2$, e vice-versa. Portanto, o subgrafo de $G(\mathcal{P})$ induzido pelo conjunto de vértices $E(\mathcal{P}_1) \cup E(\mathcal{P}_2)$ é isomorfo ao grafo $G(\mathcal{P}_1) \times K_2$ (onde $\mathcal{P}_1$ é um B-poset).

Aplicamos agora o lema 1 para substituir o par de extensões sucessivas p e $\mathbf{p}'$ por uma lista com as extensões lineares de $\mathcal{P}_1$ e $\mathcal{P}_2$, onde extensões sucessivas nesta lista diferem por uma transposição. A lista começa em p e termina em p'.

Nesta última etapa, fazemos a substituição acima para cada par de extensões lineares sucessivas na lista resultante da etapa dois que difiram entre si pela transposição de seus dois elementos mais à esquerda. A prova está completa. O

A prova do teorema segue como conseqüência direta do lema 9.

Prova (Teorema 9) : Começando com os elementos minimais do poset de "baixo para cima", um nível de cada vez, podemos gerar as extensões lineares dos elementos minimais por transposição usando o algoritmo de Steinhaus-Johnson-Trotter para permutações irrestritas. Seja $\mathcal{P}$ o subposet constituído pelos elementos que se encontram em um nível $\leq r$ e seja M o conjunto de todos os elementos no nível r. Assumimos que todas as extensões dos elementos em um nível menor que r já foram geradas (ou seja, existe um caminho hamiltoniano em $G(P - M)$). Pela versão dual do lema 9, as extensões lineares de P podem ser geradas por transposição. □

Seguem dois corolários imediatos do teorema 9.

Corolário 4 *Para* **n** *ímpar, permutações alternadas podem ser geradas por transposição.*

Uma permutação $q_1, q_2, \ldots q_n$ é dita *alternada* quando $q_1 < q_2 > q_3 < q_4 \ldots$; estas permutações são contadas por números de Euler (veja Stanley [Sta86]). A inversa de uma permutação alternada é uma extensão linear de um poset cerca, os quais serão considerados na seção VII.3.5 (veja figura VII.1). O corolário 4 foi provado por Ruskey [Rusdo].

Corolário 5 *As extensões lineares de um poset do tipo coroa podem ser geradas por transposição.*

Uma *coroa* é um poset cujo diagrama de Hasse (como grafo subjacente) é um ciclo de comprimento par e onde todos os elementos do poset se encontram em um dentre dois níveis.

Corolário 6 *As extensões lineares de um reticulado de álgebra booleana ou de um reticulado de partes podem ser geradas por transposição.*

E interessante notar que sabemos que um reticulado de álgebra booleana possui um ciclo hamiltoniano, mas quantos vértices ele possui nos é desconhecido (Sha & Kleitman [SK87]).

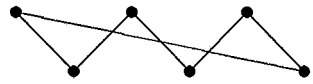


Figura IV.7: Poset coroa.

Se existe um caminho (ciclo) hamiltoniano em $G'(\mathcal{P}-M)$, onde M é o conjunto dos elementos maximais do poset, então o único problema que teríamos para obter um caminho (ciclo) em $G'(\mathcal{P})$ seria na prova do lema 8 [PR88], onde algumas transposições não-adjacentes podem ser efetuadas. Geras as permutações como no lema 8, mas usando somente transposições adjacentes, constitui um problema ainda em aberto.

Capítulo V

Geração de Extensões Lineares para Posets Série-Paralelos

Neste capítulo, apresentaremos um algoritmo de tempo médio constante que gera as extensões de um poset série-paralelo (como definido no capítulo II) em ordem lexicográfica. Este é um algoritmo recursivo, que toma como base os nós internos de uma árvore de decomposição aglutinada do poset, a ser definida. A ordem lexicográfica é garantida por um algoritmo de geração das r-combinações de n elementos nesta ordem, verificando-se as analogias necessárias. É válido ressaltar que apesar do problema de geração de extensões lineares para o caso geral estar resolvido [PR91], não é conhecido nenhum outro algoritmo de tempo médio constante que resolva este problema lexicograficamente para alguma classe de posets (a não ser no caso de permutações irrestritas).

Grafos série-paralelos foram primeiramente propostos por Duffin em 1964 [Duf65], fazendo uma analogia a redes elétricas, como sendo a menor classe de grafos fechada sob operações em série e em paralelo e que contém o poset trivial. Estes grafos foram mais tarde estudados por vários outros autores, devido a suas propriedades algorítmicas favoráveis. Veja [TNS82] para uma coletânea de resultados sobre estes grafos.

Lawler em [Law78], enquanto estudando um certo problema de escalonamento, propôs a definição de posets série-paralelos como sendo a menor classe de posets que contém o poset trivial e é fechada sob composições em série e em paralelo - casos (1) e (2) do teorema de decomposição (teorema 4). Ou ainda, os posets série-paralelos

Figura V.1: Poset N e "torre". A torre é o grafo de comparabilidade de um poset N.

são aqueles que possuem diâmetro de decomposição igual a 2, o que significa que esta é a menor classe de posets que contém o poset trivial e é fechada sob a substituição de um elemento por uma cadeia ou anticadeia de tamanho 2. Valdes [Val78] propôs o seguinte teorema :

Teorema *10 Para u m poset* $\mathcal{P}$, *as seguintes afirmações são equivalentes* :

(i) $\mathcal{P}$ *é u m poset série-paralelo;*

(ii) $\mathcal{P}$ *não contém "*$\mathbb{N}$*" (figura V.1) como subposet induzido;*

(iii) $G_C(\mathcal{P})$*, o grafo de comparabibidade de* $\mathcal{P}$*, não contém nenhum subgrafo isomorfo à "torre" (figura V.I).*

A propriedade "ser série-paralelo" é uma propriedade hereditária de posets, como podemos concluir trivialmente de (ii).

V.1 Notação *e* Definições

Seja S um conjunto e $\prec'$ uma ordem total de S. Sejam $S_1 = x_1, x_2, \ldots, x_p$ e $S_2 = y_1, y_2, \ldots, y_q$ duas seqüências de elementos de S. Diz-se que S_1 é *lexicograficamente maior* do que S_2 em $\prec'$, denotando por $S_1 \prec' S_2$, quando :

1. Existe algum índice j, $1 \leq j \leq p, q$ tal que $x_j \prec' y_j$, e para todo L, $1 \leq \mathrm{L} < j$, $x_k = y_k$. Ou então :

2. p > q e para todo h, $1 \leq k \leq q$, $x_k = y_k$.

(A operação = é interpretada da maneira usual.) As seqüências $S_1, S_2, \ldots, S_r$ estão ordenadas lexicograficamente quando $S_i \prec' S_j \Rightarrow i < j$. Por exemplo, as palavras de um dicionário estão ordenadas lexicograficamente segundo a $\prec'$ b $\prec' \ldots \prec' z$.

Seja T(V,E) uma árvore ordenada e x,y E V tais que x é pai de y em T. Uma operação de aglutinação de x e y em T é aquela onde x e y são identificados no vértice x; os filhos de y passam agora a ser filhos de x e a ordenação entre os filhos de x é dada pela ordem em que os filhos de x e de y apareciam em T, da esquerda para a direita. Ou seja, uma operação de aglutinação de x,y em T, tal que x é pai de y, resulta na árvore $T'(V', E')$ tal que $V' = V \backslash \{y\}$, $E' = E \backslash \{(y,z)$ E $E\} \cup \{(x,z) | (y,z)$ E $E)$ e a ordenação dos filhos de x em T' é dada pela ordem em que estes apareciam em T. (A ordenação dos filhos dos vértices de T' é preservada de T.)

Seja T(F) uma árvore de decomposição de um poset $\mathcal{P}(S, R)$ e sejam $1, \ldots, n-1$ os nós internos de T ($n = |S|$). Definimos a árvore de decomposição aglutinada $T_A(\mathcal{P})$ obtida a partir de T como sendo aquela obtida a partir de T mediante sucessivas aglutinações de nós série adjacentes, até que isto não seja mais possível. $T_A(\mathcal{P})$ não é uma árvore necessariamente binária.

Por n_A denotaremos o número de nós internos de $T_A(\mathcal{P})$, onde $n_A \leq n - 1$. Omitiremos o parâmetro $\mathcal{P}$ quando isto não causar ambigüidade. É importante notar que a estrutura dos nós paralelos em T_A não foi alterada, continuando estes a ser estritamente binários (por nó estritamente binário entenda-se aquele que tem exatamente dois filhos). Uma conseqüência das aglutinações efetuadas é que em T_A não mais existem nós série adjacentes entre si. Ou seja, o pai de um nó série, se este existir, é sempre um nó paralelo, e os filhos de um nó série ou são folhas (elementos de $\mathcal{P}$) ou são nós paralelos. Cada nó série em T_A tem k, $2 \leq k \leq n$, filhos. Um nó série em T_A é interpretado como uma seqüência de composições em série binárias entre cada par de subárvores consecutivas definidas nos filhos deste nó. Um aspecto principal a ser ressaltado é que o resultado final de um conjunto de composições em série em T é preservado em T_A, fazendo-se a correspondência exata entre os nós destas árvores. A ambigüidade existente na representação de um poset por várias árvores de decomposição não isomorfas devido à associatividade de composições em série consecutivas fica resolvida em T_A (a associatividade e simetria de composições

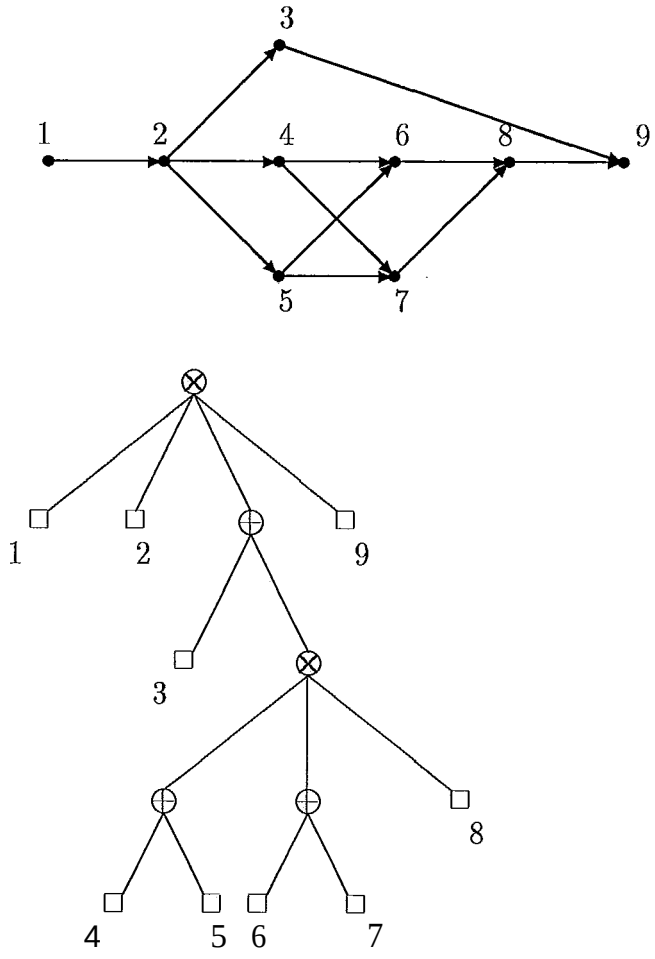


Figura V.2: Exemplo de poset série-paralelo e sua árvore de decomposição aglutinada

em paralelo consecutivas continuam no entanto não resolvidas. Veremos mais adiante como modificar a árvore aglutinada de modo a resolver o problema da simetria.)

Denotaremos por $T_A{}^i$ a subárvore de raiz i, i E T_A, em T_A. Falaremos indistintamente em subposet com subárvore aglutinada $T_A{}^k$ e subposet de $\mathcal{P}$ de raiz k. Por percorrer uma árvore em pós-ordem entendemos : "Visitar o filho mais à esquerda da raiz em pós-ordem; Visitar o 2^o filho mais à esquerda da raiz em pós-ordem; ...; Visitar o filho mais à direita da raiz em pós-ordem; Visitar a raiz".

V.2 O Algoritmo

A idéia central do algoritmo consiste em se usar a árvore de decomposição aglutinada T_A de um poset série-paralelo para sistematizar a geração de suas extensões. Os nós internos de T_A contêm toda informação sobre a estrutura de $\mathcal{P}$ (seqüência de composições em série e em paralelo). Cada subárvore $T_A{}^i$ corresponde também a um subposet série-paralelo.

Se percorrermos os nós de T_A em pós-ordem podemos, construtivamente, analisar a estrutura de **P.** Por construtivamente entendemos que, ao analisar a composição efetuada em um nó interno de T_A sobre os subposets enraizaclos nos seus filhos, estes já tenham sido anteriormente analisados. Assim, durante todo o processo de geração estaremos efetuando chamadas recursivas, onde cada nó interno só chama recursivamente seu sucessor em pós-ordem. Seja 1,..., n a numeração dos elementos do poset (folhas de T_A) na ordem em que aparecem da esquerda para a direita em T_A. Cada chamada recursiva do nó L sucede sempre à geração de uma nova extensão parcial do subposet enraizado no nó que efetuou a chamada, i. e., no nó $k-1$, anterior a k em pós-ordem. Esta extensão parcial deverá ser "completada" de todas as maneiras possíveis com os outros elementos do poset. Isto será feito mediante chamadas aos nós posteriores a $k-1$ em pós-ordem. Quando visitando um nó L (executando chamada recursiva relativa ao mesmo), olhamos para as extensões parciais já geradas nos seus filhos e as combinamos segundo a composição de k. Ou seja, se k é um nó em série, simplesmente concatenamos as extensões parciais dos seus filhos, da esquerda para a direita. Se a composição de k é em paralelo, devemos "intercalar" as extensões parciais dos filhos esquerdo e direito deste nó. Sejam e e d estes filhos e n;, i nó interno de T_A, o número de elementos do subposet de raiz i. A intercalação feita

no nó paralelo k de duas extensões de tamanhos n_e e n_d, $n_e + n_d = n_k$, corresponde à geração das n_e(ou n_d)-combinações de n_k. Ao usar um algoritmo de geração de combinações em ordem lexicográfica [NW75, RND77] para efetuar as composições em paralelo, assim estaremos gerando as extensões do poset, uma vez que as composições em série não influem nesta ordem. O retorno de uma chamada recursiva do nó k se dá quando esgotamos as combinações possíveis neste nó.

Para que a complexidade seja de fato linear em e(?), uma estrutura de dados muito conveniente deverá ser utilizada. Em linhas bem gerais, apresentamos uma primeira versão do algoritmo. Seja $ext(k)$ a última extensão parcial relativa ao sub-poset de raiz k gerada. Na descrição do algoritmo, por "||" entendemos a operação de concatenação de duas seqüências de elementos.

algoritmo GeraExtSérieParalelo
dados poset $\mathcal{P}(S, \mathrm{R})$, n $= |S|$
Gerar árvore de decomposição $T(\mathcal{P})$, reconhecendo se $\mathcal{P}$ é de fato série-paralelo;
Gerar árvore de decomposição aglutinada $T_A(\mathcal{P})$, aglutinando nós série adjacentes em T, onde $n_A =$ nº de nós internos em T_A $(1 \leq n_A \leq \mathrm{n} - 1)$;
Numerar os nós internos de T_A em pós-ordem e as folhas de T_A de 1 a n da esquerda para a direita na árvore;
Geração(1)
fim algoritmo

procediniento *Geração*(k)
    se k é um nó série então
        GeraExtSérie(k)
    senão
        GeraExtParal(k)
fim procedimento

procedimento *GeraExtSérie*(k)
    $ext(k) := ext$(filho mais à esq. de k)|| ... ||ext(filho mais à dir. de k)
    se $k =$ raiz de T_A então

Mais uma extensão linear, $ext(k)$, foi gerada.

senão

$Geração(k\$1)$

fim procedimento

procedimeiito $GeraExtParal(k)$

{ Sejam e e d as numerações em pós-ordem dos filhos esquerdo e direito de k respectivamente. Vou intercalar os elementos em $T_A{}^e$ com os em $T_A{}^d$ em ordem lexicográfica, seguindo as n_e-combinações dos inteiros $1,\ldots,n_k$ }

Gerar 1ª combinação em ordem lexicográfica (1... n,);

$ext(k) := \text{ext(e)}||ext(d)$

se L = raiz de T_A **então**

Mais uma extensão linear, $ext(k)$, foi gerada.

senão

$Geração(k+1)$

enquanto não gerei todas combinações **efetuar**

Gerar próxima combinação em ordem lexicográfica,

achando extensão linear $ext(k)$ correspondente;

se L = raiz de T_A **então**

Mais uma extensão linear, $ext(k)$, foi gerada.

senão

$Geração(k+1)$

fim **procedimeiito**

Algoritmo de geração de extensões lineares de posets série-paralelo

Ao chegar a uma chamada recursiva do nó n_A, raiz de T_A, estaremos gerando uma ou mais extensões completas do poset, pois a subárvore enraizada em n_A caracteriza o próprio poset $\mathcal{P}$. Chamaremos de extensão canônica àquela que for a primeira extensão linear completa gerada. Se analisarmos a seqüência de chamadas do algoritmo que vai combinando as extensões parciais até chegar à extensão canônica com a pri-

algoritmo GeraCombLex
dados r, t
para $j = 1, \ldots,$ r efetuar
    $a_j := j$
repita
    h $:= min\{j \mid a_{r+1-j} \neq t + 1 - j\}$
    m $:= a_{r+1-h}$
    para $j := 1, \ldots,$ h efetuar
        $a_{r+1-j} := m + j$
até que $a_1 = t + 1 - r$
fim algoritmo

Figura V.3: Algoritmo de geração de combinações em ordem lexicográfica

meira chamada do nó raiz veremos, sem dificuldade, que esta extensão é a extensão $12 \ldots$ n. Uma vez que esta deve ser a menor, lexicograficamente, dentre todas as extensões de E(?), é segundo a ordem total da esquerda para a direita entre as folhas de T_A que o algoritmo gera lexicograficamente as e(?) extensões.

Vamos agora analisar um algoritmo para geração das r-combinações de t elementos, que aparece em Nijenhuis & Wilf [NW75] (a estrutura básica do algoritmo tem este item como referência, cabendo somente uma pequena correção no cálculo da quantidade média de trabalho por combinação) ou em Reingold, Nievergelt & Deo [RND77]. O algoritmo aparece na figura V.3.

O conjunto $\{a_1, a_2, \ldots,$ a,) representa uma r-combinação. Seja $\mathcal{P}$ um poset com t elementos cuja redução transitiva consiste de duas cadeias disjuntas $\mathcal{C}_1$ e $\mathcal{C}_2$, de tamanhos r e $t -$ r respectivamente. A relação existente entre as r-combinações de t inteiros e o conjunto E($\mathcal{P}$) é a de que existe uma função biunívoca que leva cada combinação $a_1, \ldots,$ a,, $a_i < a_j$ se i $<$ j, a uma extensão linear $\in$ E($\mathcal{P}$) tal que a;, $1 \leq$ i $\leq$ r, fornece a posição do i-ésimo elemento de $\mathcal{C}_1$ nesta extensão.

É interessante medir o gasto médio de tempo envolvido na geração das r-combinações. O índice h mede a quantidade de trabalho para gerar cada combinação. Para 1 fixo, o número de r-combinações com h $= 1$\$1, i. e., com

$a_r = t,\ a_{r-1} = t-1,\ \ldots,\ a_{r-l+1} = t-l+1$ e $a_{r-l} < t-l$

é exatamente

$$\binom{t-l-1}{r-l}$$

uma vez que $a_1, \ldots, a_{r-l}$ pode ser qualquer (r − 1)-combinação de $\{1, 2, \ldots, t-l-1\}$. Segue que

$$\sum_{l=0}^{r} \binom{t-l-1}{r-l} = \binom{t}{r}$$

pois cada r-combinação contribui uma única vez para o lado esquerdo da igualdade.

Vamos supor que chegamos a calcular h quando a_1 assume o valor $t+1-r$. Sem perda de generalidade, arbitramos o valor de h neste caso como sendo $r+1$. Vale ressaltar que estamos *aumentando* a quantidade de trabalho implícita no algoritmo e, portanto, esta pequena modificação não invalida a análise de complexidade a ser efetuada. Calculando o valor médio de h :

$$\begin{aligned}
\overline{h} &= \binom{t}{r}^{-1} \sum_{l=0}^{r} \binom{t-l-1}{r-l}(l+1) = \\
&= \binom{t}{r}^{-1} \left\{ \binom{t-r-1}{0} + \left[\binom{t-r}{1} + \binom{t-r-1}{0} \right] + \cdots + \right. \\
&\quad \left. \left[\binom{t-1}{r} + \cdots + \binom{t-r-1}{0} \right] \right\} = \\
&= \binom{t}{r}^{-1} \left\{ \binom{t-r}{0} + \binom{t-r+1}{1} + \cdots + \binom{t-1}{r-1} + \binom{t}{r} \right\} = \\
&= \binom{t}{r}^{-1} \left[\binom{t}{r-1} + \binom{t}{r} \right] = \\
&= 1 + \frac{r}{t-r+1}
\end{aligned}$$

Assim, se $r < (t/2)$ necessitaremos, em média, de menos que duas unidades de trabalho por r-combinação gerada.

A seguir, uma versão mais detalhada do algoritmo, que considera aspectos relevantes para o cálculo da complexidade, como eliminar as concatenações feitas em

um nó série ou a reordenação dos filhos esquerdo e direito de um nó paralelo. Esta reordenação é feita de modo que o número de elementos relativo ao filho esquerdo de qualquer nó paralelo seja sempre menor ou igual ao número de elementos relativo a seu filho direito, assegurando assim um tempo médio constante na geração das combinações neste nó. O algoritmo foi implementado em Pascal (Turbo Pascal 5.0) e rodado eficientemente mesmo em um PC-XT (ver apêndice).

algoritmo GeraExtSérieParalelo

dados poset $\mathcal{P}(S, \mathbf{R})$, n $= |S|$

Gerar árvore de decomposição T**(P),** reconhecendo se $\mathcal{P}$ é de fato série-paralelo;

Gerar árvore de decomposição aglutinada $T_A(\mathcal{P})$, aglutinando nós série adjacentes em T, onde $n_A =$ nº de nós internos de T_A $(1 \leq n_A \leq n - 1)$;

Trocar, em T_A, o filho esquerdo com o filho direito de um nó paralelo toda vez que for o caso de à subárvore enraizada no filho esquerdo corresponderem mais elementos do poset que à do filho direito;

Numerar os nós internos de T_A em pós-ordem e as folhas de T_A de 1 a n, da esquerda para a direita na árvore. Durante este passo, calcular também $n[k]$ $(1 \leq k \leq n_A)$, o número de elementos E $T_A{}^k$. $n[k] = n[k_1] + n[k_2] + \ldots + n[k_r]$, onde $k_1, \ldots, k_r$ são os filhos de k;

Inicializar vetor Elmto e inicializar lista duplamente encadeada com extensão canônica

$Elmto[i] \uparrow .elmto := i$

$Elmto[i-1] \uparrow .prox := Elmto[i]$

$Elmto[i] \uparrow .ant := Elmto[i-1], \quad i = 1, \ldots, n$

$Elmto[1] \uparrow .ant :=$ A

$Elmto[n] \uparrow .prox :=$ A;

$Geração(1)$

fim algoritmo

procedimento $Geração(k)$

se k é um nó série **então**

$GeraExtSérie(k)$

senão

$GeraExtParal(k)$

fim procedimento

procedimento $GeraExtSérie(k)$

$Prim[k] := Prim[$filho mais à esquerda de $k]$

$Ult[k] := Ult[$filho mais à direita de $k]$

se $k = n_A$ **então**

Mais uma extensão linear, $ext(k)$, foi gerada.

senão

$Geração(k + 1)$

fim procedimento

procedimento $GeraExtParal(k)$

Sejam e e d as numerações em pós-ordem dos filhos esquerdo e direito de k, respectivamente;

$ext(k) := ext(e)||ext(d)$

$Prim[k] := Prim[e]$

$Ult[k] := Ult[d]$

$r := n[e]$

$A[1] := Prim[k]$

$a[1] := 1$

para $j := 2, \ldots, r$ **faça**

$a[j] := j$

$A[j] :=$ índice do elemento correspondente a $Elmto[A[j-1]] \uparrow .prox$

$(= Elmto[A[j-1]] \uparrow .prox \uparrow .elmto)$

se $k = n_A$ **então**

Mais uma extensão linear, $ext(k)$, foi gerada.

senão

$Geração(k \$ 1)$

enquanto $a[1] \neq n[k] - r + 1$ **faça**

$h := min\{j \mid a[r + 1 - j] \# n[k] + 1 - j\}$

$m := a[r + 1 - h]$
$a[r \$ 1 - h] := m + 1$
Inserir o elemento $A[r + 1 - h]$ uma posição mais à direita que a sua na última extensão gerada;
para $j := 2, \ldots, h$ **faça**
$a[r + j - h] := m + j$
Inserir o elemento $A[r + j -$ h] logo atrás de $A[r\$ j - h - 1]$;
se $E = n_A$ **então**
Mais uma extensão linear, $ext(k)$, foi gerada.
senão
$Geração(k + 1)$
fim procedimento

Versão mais detalhada do algoritmo de geração de extensões lineares de posets séie-paralelo em ordem lexicográfica

A estrutura principal de dados utilizada globalmente pelo algoritmo consiste de quatro vetores de n_A posições, cada uma correspondendo a um nó interno de T_A :

Prim[k] **:** elemento mais à esquerda da última extensão de $T_A{}^k$ gerada;

Ult[k] : elemento mais à direita da última extensão de $T_A{}^k$ gerada;

n[k] : número de elementos do subposet considerado por $T_A{}^k$;

PosOrdem[k] : ponteiro para o E-ésimo nó interno de T_A em pós-ordem;

uma lista duplamente encadeada com os n elementos de $\mathcal{P}$, que contém todas as extensões parciais geradas mais recentemente, e um vetor de n entradas para os n elementos do poset :

Elmto[i] : ponteiro para a posição do elemento i dentro da lista duplamente encadeada.

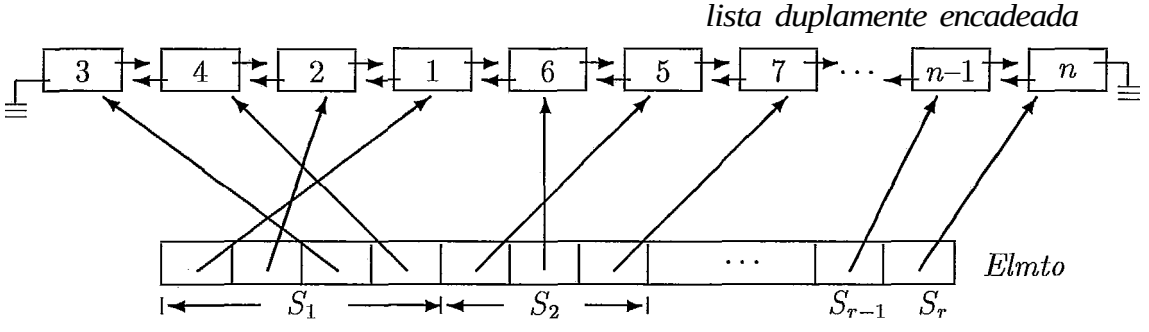


Figura V.4: Vetor *Elmto* e lista duplamente encadeada em um dado instante da execução do algoritmo para um certo poset $\mathcal{P}$ com n elementos. $\mathcal{P}_{S_1}, \ldots, \mathcal{P}_{S_r}$ são os subposets maximais sendo analisados neste instante.

Não se esquecendo da árvore aglutinada T_A, com n_A nós internos (n_A < n) e n folhas.

Consideremos, todos os subposets maximais $\mathcal{P}_{S_1}, \mathcal{P}_{S_2}, \ldots, \mathcal{P}_{S_r}$ sendo analisados em T_A em um dado instante, onde $S_i \subseteq$ S. Então, $\forall(S_i, S_j)$ {i < $j \Leftrightarrow \forall(x \in S_i$, y E $S_j)$ x < y}. Ou seja, S_i se encontra totalmente à esquerda de S_j em 12...n, se i < j (ver exemplo na figura V.4). Assim, as extensões parciais relativas a estes subposets se encontram dispostas na lista duplamente encadeada de maneira natural para a concatenação em um nó série ou mesmo para a geração da primeira extensão de uma composição em paralelo. Se mantivermos atualizados os ponteiros que apontam do último elemento de uma extensão parcial maximal gerada para o primeiro elemento da extensão parcial que a segue imediatamente, dispensamos as operações de concatenação em nós série, o que será crucial na análise de complexidade. A operação de concatenação que ainda aparece nos nós paralelos é na verdade uma "restauração" das posições originais das extensões parciais de seus filhos após uma geração das combinações.

Além da estrutura de dados global, a cada chamada recursiva a um nó paralelo k, dois vetores de r posições, r $\leq (n/2)$, são empilhados :

a[i] : contém o i-ésimo menor elemento de uma r-combinação;

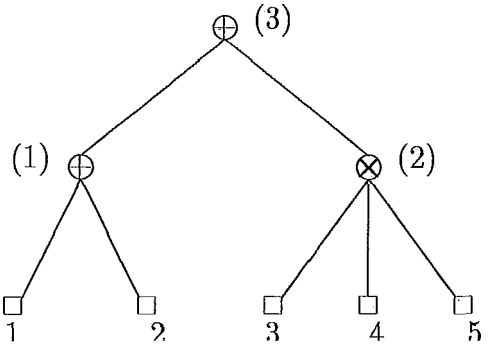


Figura V.5:

A[i] : contém a posição do i-ésimo elemento de ext(filho esquerdo de k) dentro da extensão parcial do nó paralelo k sendo gerada.

Na figura V.5 temos urna árvore aglutinada de um poset série-paralelo, que já tem os filhos de seus nós paralelos ordenados (segundo número de elementos de suas subárvores), seus nós internos numerados em pós-ordem e as folhas numeradas da esquerda para a direita. A seqüência de chamadas recursivas na geração das extensões do poset definido por esta árvore aparece na figura V.6. As chamadas ao procedimento Geração foram omitidas.

Como temos no máximo uma cadeia de $n_A <$ n chamadas recursivas, o espaço necessário para os vetores A e a empilhados é O(n^2). Já para a estrutura global apresentada, necessitamos somente de espaço $O(n)$. Portanto, o algoritmo necessita de espaço O(n^2).

V.3 Análise de Complexidade

Devemos, primeiramente, lembrar que cada operação de inserção ou de concatenação em um nó paralelo pode ser feita em tempo constante e que a cada geração de uma r-combinação de t elementos, onde $r \leq t/2$, correspondem em média menos que duas

	Extensões Parciais Maximais Geradas
	1 2 3 4 5
$GeraExtParal(1)$	12 3 4 5
$GeraExtSérie(2)$	12 345
$GeraExtParal(3)$	12345
"gerar próxima combinação em (3)"	13245
"gerar próxima combinação em (3)"	13425
"gerar próxima combinação em (3)"	13452
"gerar próxima combinação em (3)"	31245
"gerar próxima combinação em (3)"	31425
"gerar próxima combinação em (3)"	31452
"gerar próxima combinação em (3)"	34125
"gerar próxima combinação em (3)"	34152
"gerar próxima combinação em (3)"	34512
"gerar próxima combinação em (1)"	21 3 4 5
$GeraExtSérie(2)$	21 345
$GeraExtParal(3)$	21345
"gerar próxima combinação em (3)"	23145

Figura V.6: Seqüência de chamadas recursivas para o poset da figura anterior

inserções.

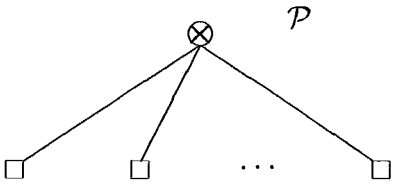
O reconhecimento de um poset série-paralelo é feito em tempo O(m + n), onde m = |R| [VTL79]. A aglutinação da árvore de decomposição, assim como a numeração em pós-ordem de $T_A(\mathcal{P})$ e a inicialização da extensão canônica consomem tempo $O(n)$. O mesmo acontece com o cálculo do número de elementos em cada subárvore de T_A : cada $n[i]$, $1 \leq i \leq n_A$, só entra no cálculo de $n[k]$, onde k é pai de i em T_A, e cada $n[k]$, $1 \leq k \leq n_A$, só é calculado uma Única vez durante todo o processo. Assim, a complexidade deste passo também é $O(n)$. Vamos finalmente analisar a contribuição da chamada $Geração(1)$ para a complexidade do algoritmo.

O tempo total gasto no processo real de geração das extensões (sem contar com a parte de inicialização) é dado por :

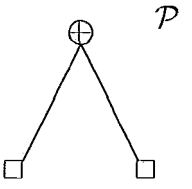
$$tempo\ geração\ (\mathcal{P}) \leq c_1\ NCR(T_A(\mathcal{P})) + c_2\ e(\mathcal{P}) \qquad (V.1)$$

onde $NCR(T_A(\mathcal{Q}))$ é o número de chamadas recursivas $Geração(k)$ efetuadas pelo algoritmo no cálculo das extensões de um poset série-paralelo $\mathcal{Q}$ com árvore aglutinada $T_A(\mathcal{Q})$; c_1 e c_2 são constantes. O termo $c_2\ e(\mathcal{P})$ se refere à combinação das extensões parciais dos filhos da raiz dentro da chamada recursiva referente a este nó. Ou seja, se o nó raiz é um nó série, $e(\mathcal{P})$ é igual ao número de chamadas deste nó, pois para cada chamada existe uma única maneira de concatenar as extensões parciais de seus filhos mais recentemente geradas. Se a raiz é um nó paralelo, então, se e_1, n_1 e e_2, n_2 são o número de extensões e o número de elementos de seus filhos direito e esquerdo respectivamente, geraremos $e_1 e_2 \begin{pmatrix} n_1 + n_2 \\ n_1 \end{pmatrix} = e(\mathcal{P})$ extensões completas por meio de n_1-combinações para $e_1 e_2$ chamadas do nó raiz. Como $T_A(\mathcal{P})$ tem os filhos de seus nós paralelos ordenados, temos que $n_1 \leq n_2$ e cada n_1-combinação efetuada no nó raiz consome no máximo duas unidades de tempo. O tempo gasto na combinação de extensões parciais na raiz é, portanto, $\leq 2e(\mathcal{P})$ unidades de tempo.

O tempo gasto na geração das extensões estritamente parciais contribui com $c_1\ NCR(T_A(\mathcal{P}))$, pois a cada nova extensão estritamente parcial gerada no nó k corresponde uma chamada $Geração(k+1)$, $1 \leq L < n_A$. Outra vez, como $T_A(\mathcal{P})$ tem os filhos de um nó paralelo ordenados com relação aos números de elementos relativos aos subposets enraizados nos mesmos, gastaremos em média menos que duas unidades de tempo por extensão parcial gerada, sendo $c_1 = 2$ um valor conveniente para esta



$$\begin{cases} e(\mathcal{P}) = 1 \\ NCR(T_A(\mathcal{P})) = 1 \end{cases}$$



$$\begin{cases} e(\mathcal{P}) = 2 \\ NCR(T_A(\mathcal{P})) = 1 \end{cases}$$

Figura V.7: Dois únicos casos de posets com um único nó interno.

constante.

Devemos então provar que $NCR(T_A(\mathcal{P}))$ é $O(e(\mathcal{P}))$, ou não atingiremos a complexidade almejada. A prova segue por indução no número de nós de T_A, sendo estes considerados em ordem inversa à pós-ordem (ou in-ordem). Para o processo de indução não nos deteremos a árvores aglutinadas ordenadas, como é o caso da árvore aglutinada construída pelo algoritmo. Tomemos como hipótese de indução que :

$$NCR(T_A(\mathcal{P}')) \leq 2e(\mathcal{P}') - 1, \tag{V.2}$$

onde $\mathcal{P}'$ é um poset série-paralelo tal que $T_A(\mathcal{P}')$ possui $n_A < n_A$ nós internos. O poset $\mathcal{P}$ tal que $T_A(\mathcal{P})$ possui um Único nó interno satisfaz a (V.2), como vemos nos dois únicos casos da figura V.7.

Seja $\mathcal{P}'$ um poset série-paralelo cuja árvore de decomposição aglutinada $T_A(P^1)$ possui $n_A{}'$ nós internos. Seja $\mathcal{P}$ um poset tal que $T_A(\mathcal{P})$ é obtida adicionando-se um novo nó interno α em in-ordem a $T_A(\mathcal{P}')$. $T_A(\mathcal{P})$ possui $n_A = n_A{}' + 1$ nós internos. Reparem que a posição onde a pode ser inserido em $T_A(\mathcal{P}')$ não é necessariamente única; basta que se garanta que a será o primeiro nó de $T_A(\mathcal{P})$ visitado em pós-ordem (ver figura V.8).

Juntamente com a adição de um nó interno se dá a adição dos elementos de $\mathcal{P} - \mathcal{P}'$ (folhas de $T_A(\mathcal{P})$) correspondentes a este nó. No caso de α ser um nó série, o número de elementos adicionados pode ser maior que 1. Seja x o elemento/folha que ocupa a

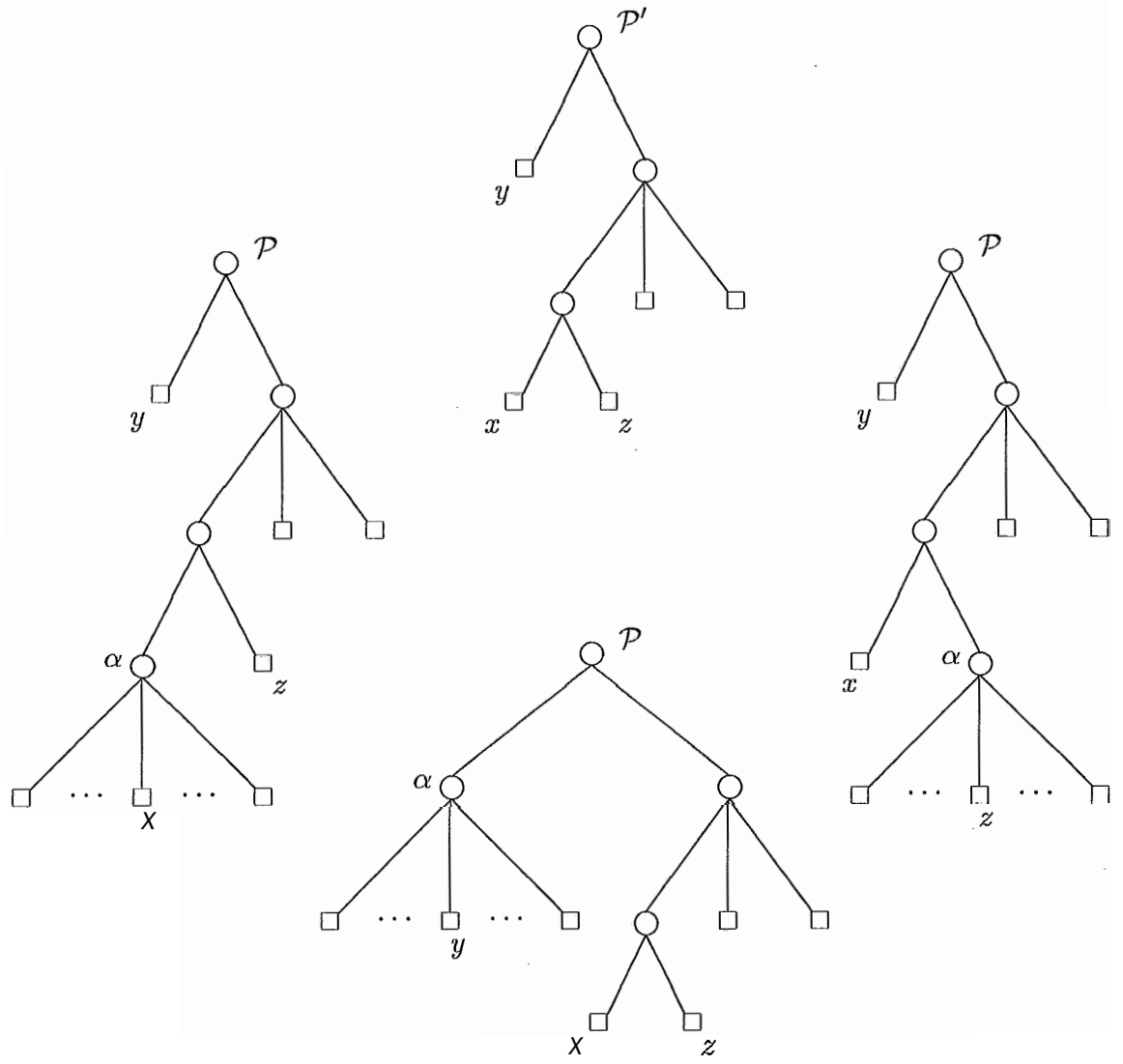


Figura V.8: *a* pode ser inserido em **3** *(e* somente **3)** diferentes posições em $T_A(\mathcal{P}')$.

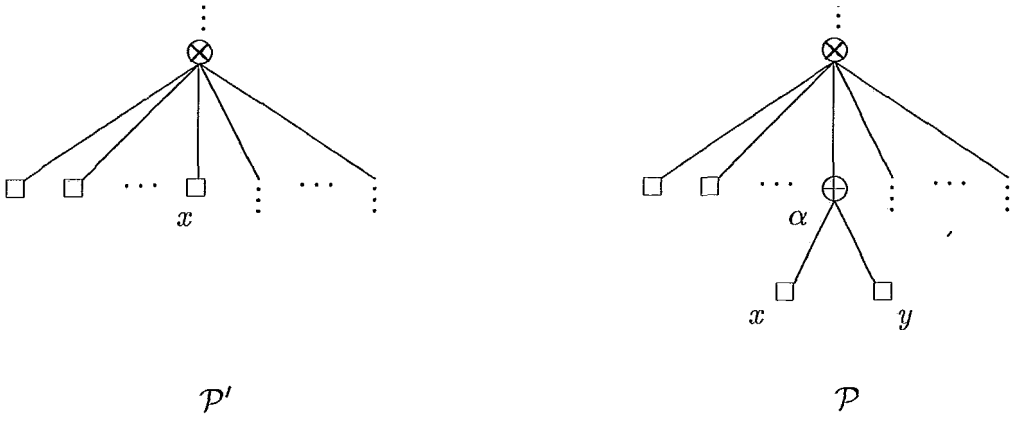


Figura V.9: a é um nó paralelo.

posição em $T_A(\mathcal{P}')$ onde α será inserido. Há dois casos a se considerar :

(a) a é um **nó paralelo.** Valem as seguintes afirmações :

$$e(\mathcal{P}) \geq 2e(\mathcal{P}') \tag{V.3}$$

$$NCR(T_A(\mathbf{P})) = 2\,NCR(T_A(\mathcal{P}')) + 1 \tag{V.4}$$

Não é difícil concluir que o caso onde $e(\mathcal{P})$ menos cresce em relação a $e(\mathcal{P}')$ ocorre quando os filhos do nó paralelo a se compõem em série com todos os outros elementos do poset. Ou seja, teríamos a seguinte configuração : o pai de α é um nó série, raiz de $T_A(\mathcal{P})$ (ou $T_A(\mathcal{P}')$), como ilustra a figura V.9. Pois, se o nó série pai de α não fosse raiz, o pai deste nó seria um nó paralelo (uma árvore aglutinada não tem nós série adjacentes, por definição) e os filhos de a não se comporiam em série com os demais elementos do poset. Neste caso, atingimos a igualdade em (V.3), uma vez que a cada $\beta x \gamma$ E E(?'), onde $\beta\gamma$ é uma extensão linear do poset $\mathcal{P}'\backslash\{x\}$, fazemos corresponder duas extensões $\beta xy\gamma$ e $\beta yx\gamma$ em E(?), onde y é o elemento $\notin \mathcal{P}'$ que foi adicionado junto com a a $T_A(\mathcal{P}')$. A igualdade em (V.4) é facilmente verificada. O nó a será o primeiro a ser chamado na seqüência de chamadas recursivas (é o primeiro em pós-ordem). Ele contribui, portanto, com somente uma chamada recursiva. Temos duas extensões parciais a serem geradas em a (xy e yx), que serão "completadas" com os outros elementos de $\mathcal{P}$ mediante chamadas recursivas aos outros nós internos

de $T_A(\mathcal{P})$. Isto corresponde a se gerar todas as extensões de $\mathcal{P}'$ para cada uma das duas extensões parciais geradas em a. Como para se gerar todas as extensões de $\mathcal{P}'$ necessitamos de $NCR(T_A(\mathcal{P}'))$ chamadas recursivas, chegamos a (V.4). O raciocínio construído só é válido uma vez que o nó inserido é o primeiro em pós-ordem na árvore, senão a seqüência de chamadas recursivas para se "completar" as extensões parciais com os elementos de $\mathcal{P}'$ seria alterada.

Substituindo $NCR(T_A(\mathcal{P}'))$ pela hipótese de indução em (V.4) :

$$NCR(T_A(\mathcal{P})) \leq 2[2e(\mathcal{P}') - 1] + 1 = 4e(\mathcal{P}') - 1$$

Substituindo agora (V.3) na desigualdade acima, chegamos a

$$NCR(T_A(\mathcal{P})) \leq 2e(\mathcal{P}) - 1$$

como queríamos demonstrar.

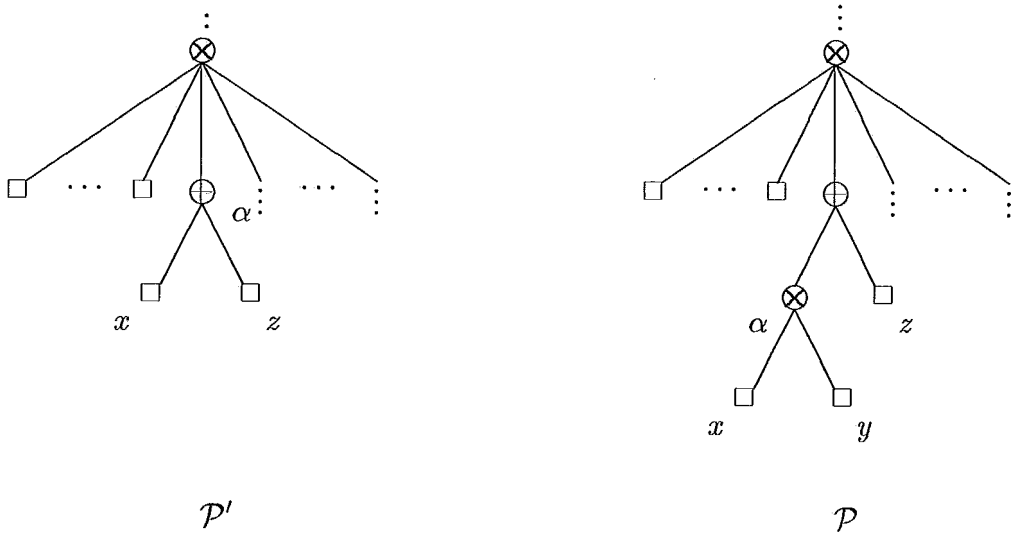
(b) α é um nó **série.** Valem as seguintes afirmações :

$$e(\mathcal{P}) \geq \frac{3}{2}\, e(\mathcal{P}') \qquad \text{(V.5)}$$

$$NCR(T_A(\mathcal{P})) = NCR(T_A(\mathcal{P}')) + 1 \qquad \text{(V.6)}$$

A equação (V.6) pode ser mostrada de maneira análoga à (V.4). Só que agora só temos uma única extensão parcial a ser gerada em a, uma vez que este é um nó série. Assim, as extensões de $\mathcal{P}'$ serão geradas uma única vez, mediante $NCR(T_A(P^1))$ chamadas recursivas. Somamos 1 pela própria chamada a este nó.

Já a desigualdade em (V.5) requer uma análise mais cuidadosa. Um pai de um nó série, se este existir, tem que necessariamente ser um nó paralelo, pela definição de árvore aglutinada. Ou seja, só podemos inserir o nó série a como filho de um nó paralelo de $\mathcal{P}'$. O caso onde $e(\mathcal{P})$ cresce menos em relação a $e(\mathcal{P}')$, ilustrado na figura V.lO, é aquele onde a é filho de um nó paralelo que tem como outro filho uma folha e que se compõe com todos os outros elementos do poset por meio de uma composição em série. Além disso, o nó α possui somente dois filhos (duas folhas). Por que esta configuração de $T_A(\mathcal{P})$ seria aquela mencionada ? Seja β o nó pai de a em $T_A(\mathcal{P})$. Se o outro filho de β não fosse uma folha (fosse a raiz de uma subárvore com mais de um elemento do poset), estaríamos aumentando o conjunto



$\mathcal{P}'$ $\mathcal{P}$

Figura V.lO: α é um nó série.

$e(\mathcal{P}') = 2$ $\qquad$ $e(\mathcal{P}) = 3$

$$\Rightarrow e(\mathcal{P}) = \tfrac{3}{2} e(\mathcal{P}')$$

Figura V.11:

de elementos sobre o qual tirar as n_α(ou np $- n_\alpha$)-combinações, aumentando assim o número de extensões parciais geradas em β em $T_A(\mathcal{P})$ com relação às geradas neste nó em $T_A(\mathcal{P}')$. Por outro lado, se fosse o caso de α ter mais que dois filhos, o número de n_a-combinações (= $(n_\beta - n_\alpha)$-combinações) também cresceria pois o número de elementos n_β de onde tiramos nossas combinações cresce linearmente com n_α. Pior ainda seria o caso de β não se compor em série com algum outro elemento não em $T_A{}^\beta(\mathcal{P}')$, pois o acréscimo de um elemento ao poset $\mathcal{P}'$ aumentaria de um fator maior que 1 o número de combinações em algum nó paralelo envolvendo os elementos do subposet enraizado em β.

Considerando então a configuração da figura V.lO, temos que para cada par de extensões $\gamma x z \sigma$ e $\gamma z x \sigma$ em $\mathcal{P}'$, fazemos corresponder três extensões $\gamma x y z \sigma$, $\gamma x z y \sigma$ e $\gamma z x y \sigma$ em $\mathcal{P}$ onde y é o elemento adicionado a $\mathcal{P}'$ junto com a como filho direito de β (ou $\gamma x z y \sigma$, $\gamma z x y \sigma$ e $\gamma z y x \sigma$, se a foi inserido como filho esquerdo de β) e $\gamma\sigma \in \mathrm{E}(\mathcal{P}' \backslash \{x, z\})$. Ou seja, $e(\mathcal{P})$ é $3/2$ vezes maior que $e(\mathcal{P}')$, atingindo a igualdade de (V.5). Um exemplo onde temos esta igualdade pode ser visto na figura V.11.

Finalmente, substituindo a hipótese de indução em (V.6), temos

$$NCR(T_A(\mathcal{P})) \leq 2e(\mathcal{P}') \qquad \text{(V.7)}$$

Mas, sabemos pelo teorema 1, pág. 8, que $e(\mathcal{Q}) \geq 1$, para qualquer poset $\mathcal{Q} \neq \emptyset$. Então, usando (V.5) e o fato que $e(\mathcal{P}') \geq 1$, obtemos

$$e(\mathcal{P}') \leq e(\mathcal{P}) - \frac{1}{2}$$

Substituindo em (V.7) :

$$NCR(T_A(\mathcal{P})) \leq 2e(\mathcal{P}) - 1$$

como queríamos demonstrar.

Em nenhum momento da prova acima nos preocupamos com a suposição de $T_A(\mathcal{P})$ ter seus filhos de nós paralelos ordenados segundo os números de elementos relativos às subárvores enraizadas nestes filhos, pois este caso está embutido no caso mais geral. No entanto, esta suposição é de vital importância para que fosse correta a afirmação de que o coeficiente de $NCR(T_A(\mathcal{P}))$ em (V.1) é uma constante. Portanto, supondo $T_A(\mathcal{P})$ ordenada desta maneira, combinando (V.1) com (V.2) e usando valores convenientes para c_1 e c_2 ($c_1 = c_2 = 2$), finalmente chegamos a :

$$\textit{tempo geração}\,(\mathbf{P}) \leq 6\,e(\mathcal{P}) - 2$$

unidades de tempo. Na prática, contudo, esta constante se mostrou bastante pessimista, como pode ser visto nos exemplos do apêndice.

Capítulo VI

Algoritmo de Tempo Médio Constante Para Geração de Extensões Lineares

Uma definição do advérbio "rapidamente" é "em acelerada sucessão". O propósito deste capítulo é mostrar que as extensões de um poset qualquer podem ser geradas rapidamente; mais ainda, tão rápido que nenhum outro algoritmo poderia gerá-las de maneira mais rápida, a menos de fatores constantes. Descreveremos, então, um algoritmo ótimo para geração das extensões lineares de um poset apresentado por Pruesse & Ruskey [PR91] em meados deste ano, resolvendo o já discutido problema. Sua complexidade é linear no número de extensões do poset, ou seja, é um algoritmo de tempo médio constante. Este capítulo tem como base o trabalho da dupla em [PR91] e, portanto, omitiremos referências ao mesmo.

É interessante ressaltar (mais uma vez) que este é o primeiro algoritmo de tempo médio constante para geração de objetos combinatórios cujo problema de enumeração correspondente foi provado ser #P-completo [BW90]. Este resultado indica que o problema de enumeração pode não ser mais fácil que o problema de geração correspondente. A solução apresentada para a questão da geração é ótima, no sentido que não podemos listar $e(\mathcal{P})$ elementos em tempo inferior a $O(e(\mathcal{P}))$. As constantes envolvidas no cálculo da complexidade são muito pequenas, estendendo, na prática, a classe de posets cujas extensões podem ser geradas e contadas. Não se conhece outro algoritmo $O(e(\mathcal{P}))$ que conte as extensões de um poset arbitrário; além disso,

algumas modificações podem ser introduzidas neste algoritmo de modo que não se necessite gerar, de fato, todas as extensões para poder contá-las.

A estratégia básica do algoritmo consiste em se gerar cada extensão linear duas vezes, cada cópia recebendo um sinal, positivo ("+") ou negativo ("−"). Somente as extensões positivas são listadas. Deste modo, o algoritmo cai na classe dos algoritmos de geração onde mais objetos do que necessário são gerados.

Introduziremos aqui alguma notação e conceitos relevantes à descrição do algoritmo. Seja $\mathcal{P}(S, \mathrm{R})$ um poset. Por $\pm E(\mathcal{P})$ denotaremos $\{+l, -l \mid l \in E(\mathcal{P})\}$. Suponhamos a e b dois elementos incomparáveis em $\mathcal{P}$, tais que a se relaciona da mesma forma que b com todos os outros elementos de S; mais precisamente, para todo $c \in S$, (c $\prec a$ **H** c $\prec$ b) e ($a \prec c$ **H** b $\prec$ c). Chamamos a e b de antigêmeos.

Provaremos agora um lema que será útil para desenvolvimentos posteriores.

Lema 10 Se a e b são *antigêmeos* em $\mathcal{P}$, então $G(\mathcal{P}) \cong G(\mathcal{P} + \mathrm{ab}) \dot{\times} K_2$.

Prova : Observemos que $E(\mathcal{P}) = E(\mathcal{P}+\mathrm{ab}) \cup E(\mathcal{P}+\mathrm{ba})$. Transpor a e b em qualquer extensão linear l de $\mathcal{P}$ nos dá outra extensão linear de $\mathcal{P}$. Deste modo, obtemos um isomorfismo entre $G(\mathcal{P} + \mathrm{ab})$ e $G(\mathcal{P} + \mathrm{ba})$ por meio da operação que transpõe a e b (o que garante a existência de arestas ligando vértices isomorfos nestes dois grafos de transposição). □

VI.1 $G(\mathcal{P}) \times K_2$ é Hamiltoniano

Para o propósito da indução, consideraremos o grafo com dois vértices ligados por uma aresta (o grafo trivial $\times K_2$) como contendo um ciclo hamiltoniano, uma vez que este possui um caminho hamiltoniano tal que seus extremos estão ligados por meio de uma aresta.

A prova de que $G(\mathcal{P}) \times K_2$ é hamiltoniano constitui a base do algoritmo a ser apresentado na próxima seção. Este resultado é válido (lema 1) para a classe dos B-posets, considerada na seção II.3.1. Um exemplo de B-poset, que mais tarde veremos ser um *2B-poset*, é dado na figura VI.l. Seja $mr(x_i)$, $1 \leq$ i $\leq$ r, o maior índice j, $1 \leq j \leq s$, tal que $x_i \| y_j$; se x; $\prec y_1$ então $mr(x_i) = 0$. Para o poset da figura VI.l, $mr(a) = 4$ e $mr(b) = 7$.

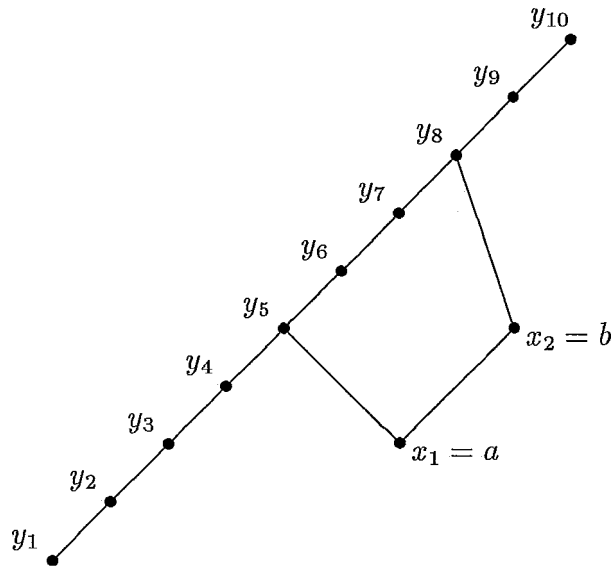


Figura *VI.1:* Um 2B-poset.

Na figura *VI.2,* vemos o grafo $G(\mathcal{P}) \times K_2$, onde $\mathcal{P}$ é o poset da figura VI.1. As arestas correspondentes ao isomorfismo entre as duas cópias de $G(\mathcal{P})$ foram omitidas. Estas representam uma simples troca de sinal entre as extensões. Uma aresta vertical corresponde a uma transposição adjacente de $b = x_2$ com um de seus vizinhos na extensão; já uma aresta horizontal corresponde a uma transposição adjacente de a $= x_1$. Um caminho hamiltoniano entre as duas cópias da extensão canônica do B-poset, $+\kappa$ e $-\kappa$, é mostrado na figura VI.3. Se adicionamos a aresta $(+\kappa, -\kappa)$ a este caminho, obtemos um ciclo hamiltoniano, como especificado pelo lema 1. Todos os B-posets a serem utilizados de agora em diante têm $r = 2$ e são, portanto, chamados de *2B-posets.* Entre os 2B-posets, existem dois casos distintos a se considerar : toda vez que $a \| y_1$, teremos um grafo "similar" ao apresentado na figura *VI.3,* que consideraremos como sendo o *caso típico;* se $a \prec y_1$, então $G(\mathcal{P})$ é um caminho simples e este será o *caso atípico.* Em outras palavras o caso típico ocorre quando $mr(a) > 0$ e o atípico quando $mr(a) = 0$. Se $mr(b)$ é par, o caminho da figura *VI.3* fica um pouco modificado, pois usaremos a aresta que leva de $+ay_1 \ldots y_{mr(b)} b \ldots y_s$ a $-ay_1 \ldots y_{mr(b)} b \ldots y_s$. O teorema a seguir é a base do algoritmo a ser descrito. Prova-10-emos por indução no número

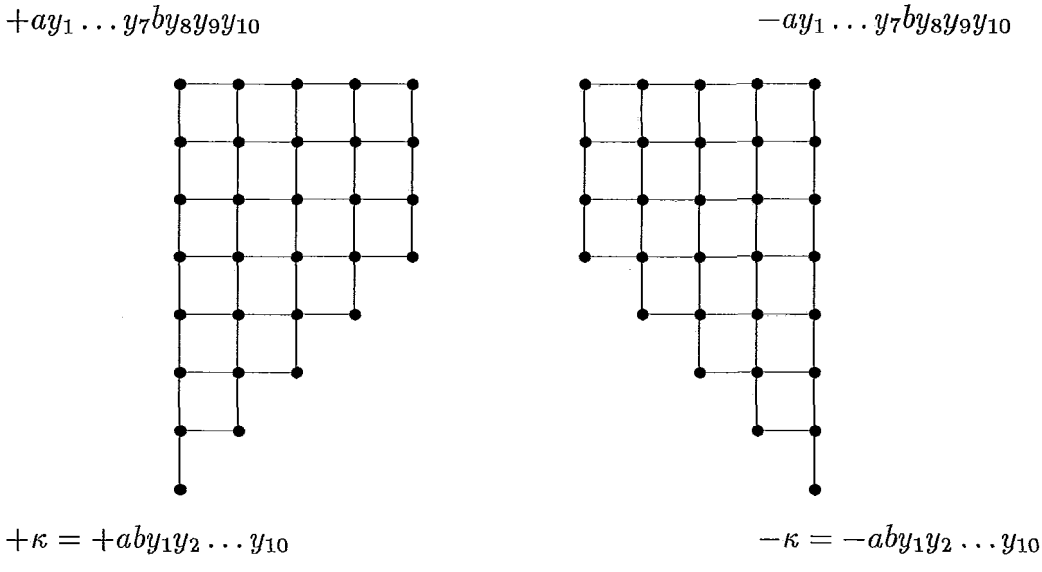


Figura VI.2: $G(\mathcal{P}) \times K_2$.

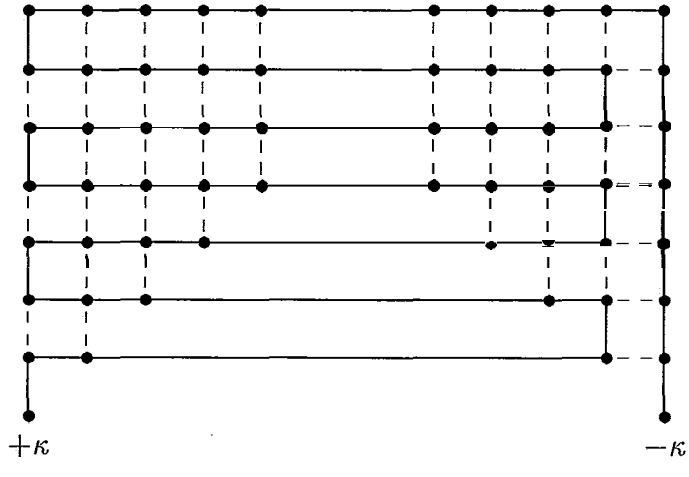


Figura V1.3: Um ciclo hamiltoniano em $G(\mathcal{P}) \times K_2$.

de elementos do poset.

Teorema 11 Para todo poset P, o *grafo* $G(\mathcal{P})$ x K_2 é humiltoniano.

Prova : A base da indução são os casos onde $\mathcal{P}$ possui um ou zero elementos e $G(\mathcal{P}) \times K_2$ é isomorfo a K_2.

Seja o poset $\mathcal{P}(S, \mathrm{R})$, com $|S| = \mathrm{n} > 1$ elementos. Consideremos o conjunto de elementos minimais de P . Se $\mathcal{P}$ possui um mínimo (elemento minimal único) a, então $G(\mathcal{P})$ r $G(\mathcal{P}\backslash\{a\})$ e, pela hipótese de indução, $G(\mathcal{P}\backslash\{a\})$ x K_2 é hamiltoniano.

De outro modo, suponhamos $\mathcal{P}$ com pelo menos dois elementos minimais a e b. Novamente pela hipótese de indução, $G(\mathcal{P}\backslash\{a, \mathrm{b})) \times K_2$ possui um ciclo hamiltoniano H'. Em H', substitua cada extensão positiva $+\alpha$ por $ab\alpha$ e cada extensão negativa $-\alpha$ por $ba\alpha$. Isto resulta em um ciclo $H' = \beta_1, \beta_2, \ldots, \beta_M$ em $G(\mathcal{P})$, onde $M = 2e(\mathcal{P}\backslash\{a, \mathrm{b}))$. Este ciclo visita exatamente aquelas extensões de $\mathcal{P}$ onde a e b precedem todos os outros elementos de S. H' só será um ciclo hamiltoniano em $\mathcal{P}$ se a e b precederem todos os elementos de $S\backslash\{a, \mathrm{b})$. Ou seja, H' percorre todas as extensões de

$$\mathcal{P}' = \mathcal{P} + \sum_{x \in S\backslash\{a,b\}} (ax + bx).$$

Para cada $\beta_i = x_i y_i \zeta_i$, onde $\{x_i, y_i\} = \{\mathrm{a}, \mathrm{b})$, o poset $\mathcal{P} + x_i y_i + \zeta_i$ é um B-poset (sendo mais específico, um 2B-poset). Pelo lema 1, existe um caminho hamiltoniano em G(P+$x_i y_i$+ζ_i) x K_2 que vai de $+\beta_i$ a $-\beta_i$. Mais precisamente, existe um caminho similar ao da figura VI.3 (ou um pouco modificado, se $mr(y_i)$ é par), se temos um caso típico (x_i não precede o elemento mínimo de ζ_i). Se temos o caso atípico, o caminho em G(P+ $x_i y_i$ + ζ_i) $\times K_2$ entre $+\beta_i$ e $-\beta_i$ é único e é dado pelas duas cópias de $G(\mathcal{P} + x_i y_i + \zeta_i)$ – neste caso um caminho simples – ligadas pela aresta que leva de $+x_i\zeta_{i_1} y_i \zeta_{i_2}$ a $-x_i\zeta_{i_1} y_i \zeta_{i_2}$, onde $\zeta_i = \zeta_{i_1}\zeta_{i_2}$ e y; precede todos os elementos de ζ_{i_2} no 2B-poset. Substituímos, então a ocorrência de β_i em H' pelo caminho que leva de $+\beta_i$ a $-\beta_i$, se i ímpar. Se i é par, então substituímos a ocorrência de β_i pelo reverso deste caminho, ou seja pelo caminho que leva de $-\beta_i$ a $+\beta_i$. Seja H o caminho resultante.

Para provar que H é um ciclo hamiltoniano em $G(\mathcal{P}) \times K_2$, é necessário que se mostre que cada vértice $+l$ ou $-l$ em H corresponde a uma extensão linear l de $\mathcal{P}$, e que toda extensão linear l de $\mathcal{P}$ aparece exatamente duas vezes, uma com sinal positivo e outra com sinal negativo, em H. É óbvio que o poset P + $x_i y_i$ + ζ_i é

uma extensão de $\mathcal{P}$ (só acrescentamos algumas relações compatíveis com a ordem parcial R) e, portanto, $\mathbf{E(P + } x_i y_i + \zeta_i) \subseteq E(\mathcal{P})$, para todo $\mathbf{1} \leq$ i $\leq$ M. Resta mostrar que para cada extensão l de $\mathcal{P}$, $+l$ e $-l$ ocorrem exatamente uma vez em H. Suponhamos, sem perda de generalidade, que l induz a ordem ab em $\{a, b)$ e a ordem ζ em $S\backslash\{a, b)$. Então l é uma extensão linear do 2B-poset $\mathcal{P}$ + ab + ζ e qualquer outro 2B-poset gerado ou não induz a ordem ab (induz ba) ou não induz ç, senão a extensão $ab\zeta$ de $\mathcal{P}'$ apareceria mais de uma vez em H'. Assim, $+l$ e $-l$ são consideradas somente quando buscamos um caminho hamiltoniano em G(P + $ab+\zeta$), i. e., são consideradas exatamente uma vez em H □

A prova introduzida acima foi direcionada para a construção do algoritmo, de modo que o mesmo possa ser traduzido como repetidas aplicações do processso de construção do caminho H indutivamente sobre o conjunto de elementos minimais do subposet de $\mathcal{P}$ sendo considerado. De outro modo, não precisaríamos ter especificado os caminhos usados na "expansão" de H' (por estarmos considerando sempre 2B-posets), uma vez que o resultado do lema 1 já seria suficiente para a prova de existência dos caminhos entre $\pm\beta_i$.

O corolário a seguir é imediato, se usamos o resultado do lema 10.

Corolário 7 Se $\mathcal{P}$ é um poset com um par de antigêmeos, então $G(\mathcal{P})$ é *hamiltoniano*.

O teorema 11 também vale para transposições adjacentes, como enunciado no teorema 12. Neste caso, a prova se torna bem mais complexa e foge ao escopo do nosso trabalho. A prova, por indução, deste teorema pode ser encontrada em [PR91].

Teorema 12 Para todo poset $\mathcal{P}$, o grafo $G'(\mathcal{P}) \times K_2$ é hamiltoniano.

VI.2 O Algoritmo

A prova do teorema 11 é construtiva, implicando diretamente em um algoritmo recursivo para a geração das extensões de um poset arbitrário, caso seja permitido se gerar duas vezes cada extensão do poset. Mais tarde, provaremos que tal algoritmo é

naturalmente de tempo médio constante, ou seja, ele gera todas as extensões do poset em tempo $O(e(\mathcal{P}))$.

Primeiro daremos uma idéia geral da estrutura do algoritmo, usando um pequeno exemplo para mostrar seu desempenho. Depois, consideraremos o mesmo em maiores detalhes e finalmente chegaremos às provas de suas correção e complexidade.

O algoritmo mantém um vetor Ext, que contém a extensão linear corrente (a última gerada), e uma variável booleana Posit, que indica se o sinal da extensão em *Ext* é positivo ("+") ou não ("–"). Passamos de uma extensão à outra na geração transpondo elementos em Ext ou trocando o sinal da extensão corrente (mudando o valor de Posit). O procedimento principal do algoritmo, *GerExt*, é recursivo e segue basicamente o caminho indicado na figura VI.3. A cada nível da recursão associa-se um par de elementos minimais do subposet sendo considerado. Por exemplo, no poset da figura II.8, a_1 e b_1 são um par de elementos minimais de $\mathcal{P}_1 = \mathcal{P}$, enquanto a_2 e b_2 são um par de elementos minimais de $\mathcal{P}_2 = \mathcal{P}_1 \backslash \{a_1, b_1\}$. Mais tarde, veremos como os pares de elementos minimais associados a cada nível são determinados a priori.

Os procedimentos Move e *TrocaContexto* são usados para se gerar uma nova extensão a partir da corrente. Eles operam em tempo $O(1)$.

TrocaContexto (i) : Se i $= 0$, então o procedimento troca o sinal de Posit; se i > 0, então a; e b_i, o par de elementos minimais em questão, são transpostos em Ext;

Move (x, d) : Se d $=\!\!\!=\!\!\!\blacktriangleleft$ então uma chamada a este procedimento transpõe x com o elemento à sua esquerda; se d $=\!\!\rightarrow$, transpõe x com o elemento à sua direita.

Cada vez que uma nova extensão linear l de $\mathcal{P}$ é gerada mediante uma chamada a $Move(i, +\text{– ou } +)$ ou $TrocaContexto(i)$, dentro de uma chamada $GerExt(i)$, $GerExt(i-1)$ é chamada recursivamente. Uma chamada a $GerExt(i-1)$ acarreta a geração das extensões lineares onde $a_1, b_1, \ldots, a_{i-1}, b_{i-1}$ são movidos de todas as maneiras possíveis sobre 1, preservando a ordem total definida em l pelos elementos $S \backslash \{a_1, b_1, \ldots, a_{i-1}, b_{i-1}\}$, além da ordem definida por a_{i-1} e b_{i-1} em l ($a_{i-1} \prec b_{i-1}$ ou $b_{i-1} \prec a_{i-1}$). Se i $= 1$, então $GerExt(i - 1 = 0)$ não faz nada.

A figura VI.4 mostra a seqüência de execução

$$GerExt(2); TrocaContexto(2); GerExt(2)$$

sobre o poset da figura 11.8, se partimos da extensão $+a_1 b_1 a_2 b_2$.

Chamadas de Procedimentos	**Extensões Lineares**
	$+a_1b_1a_2b_2$
$GerExt(2)$	
$\quad GerExt(1)$	
$\quad\quad Move(b_1, \rightarrow)$	$+a_1a_2b_1b_2$
$\quad\quad TrocaContexto(0)$	$-a_1a_2b_1b_2$
$\quad\quad Move(b_1, \leftarrow)$	$-a_1b_1a_2b_2$
$\quad TrocaContexto(1)$	$-b_1a_1a_2b_2$
$\quad GerExt(1)$	
$\quad\quad TrocaContexto(0)$	$+b_1a_1a_2b_2$
$TrocaContexto(2)$	$+b_1a_1b_2a_2$
$GerExt(2)$	
$\quad GerExt(1)$	
$\quad\quad Move(a_1, \rightarrow)$	$+b_1b_2a_1a_2$
$\quad\quad TrocaContexto(0)$	$-b_1b_2a_1a_2$
$\quad\quad Move(a_1, \leftarrow)$	$-b_1a_1b_2a_2$
$\quad TrocaContexto(1)$	$-a_1b_1b_2a_2$
$\quad GerExt(1)$	
$\quad\quad TrocaContexto(0)$	$+a_1b_1b_2a_2$

Figura VI.4: Exemplo de seqüência de chamadas dos procedimentos.

$i := j := 0$
$\mathcal{Q}(S', R') := \mathcal{P}(S, R)$
enquanto $S' \neq \emptyset$ **faça**
    se $\mathcal{Q}$ tem um mínimo a **então**
        $j := j + 1$
        $Ext[j] :=$ a
        $\mathcal{Q} := \mathcal{Q} \backslash \{a\}$
    senão
        Sejam a, b dois elementos minimais quaisquer de $\mathcal{Q}$.
        $i := i + 1$
        $j := j + 2$
        $A[i] :=$ a
        $B[i] := b$
        $Ext[j-1] :=$ a
        $Ext[j] := b$
        $\mathcal{Q} := \mathcal{Q} \backslash \{a, b\}$
$ParMax := i$

Figura VI.5: Rotina de inicialização.

Agora entremos em alguns detalhes de implementação. Na figura VI.6, temos o procedimento básico $GerExt$ em estrutura *Algol-like* e, na figura VI.5, a rotina de inicialização para aplicação de $GerExt$, que determina os pares de elementos minimais.

A estrutura de dados global do algoritino matém quatro vetores : um vetor Ext que contém a extensão linear corrente, um vetor que é a inversa de Ext (será importante para garantir a complexidade $O(1)$ das operações $TrocaContexto$ e Move) e os vetores A e B, que guardam os pares de elementos minimais a; e b_i. Determinados pela rotina de inicialização, a; e b„ $1 \leq$ i $\leq ParMax$, permanecem fixos durante toda a execução do algoritmo. Entretanto, $A[i]$ e $B[i]$ serão mantidos de tal forma que $A[i]$ sempre contenha o elemento mais à esquerda em ***Ext*** do i-ésimo par a;, b_i e $B[i]$ o elemento mais à direita. A função booleana Dir é usada para se determinar a possibilidade

```
procedimento GerExt(i)
se i > 0 então
    GerExt(i – 1)
    NMovBDir := 0
    Típico := FALSO
    enquanto Dir(B[i]) faça
        NMovBDir := NMovBDir + 1
        Move(B[i], →)
        GerExt(i – 1)
        NMovADir := 0
        se Dir(A[i]) então
            Típico := VERDADEIRO
            repita
                NMovADir := NMovADir + 1
                Move(A[i], →)
                GerExt(i – 1)
            até que not Dir(A[i])
        se Típico então
            TrocaContexto(i – 1)
            GerExt(i – 1)
            se NMovBDir é ímpar então
                NMovAEsq := NMovADir – 1
            senão NMovAEsq := NMovADir + 1
            para j := 1,..., NMovAEsq faça
                Move(A[i], ←)
                GerExt(i – 1)
    se Típico e NMovBDir é ímpar então
        Move(A[i], ←)
    senão TrocaContexto(i – 1)
    GerExt(i – 1)
    para j := 1,..., NMovBDir faça
        Move(B[i], ←)
        GerExt(i – 1)
fim procediniento
```

Figura VI.6: Procedimento *GerExt(i)*.

de um elemento x ser transposto com o elemento à sua direita. Esta função também opera em tempo $O(1)$:

Dir (B[i]) : Retorna VERDADEIRO se e somente se $B[i]$ é incomparável com o elemento à sua direita em Ext;

Dir (**A**[i]) : Retorna VERDADEIRO se e somente se $A[i]$ é incomparável com o elemento á sua direita em Ext, e este elemento não é $B[i]$.

Vejamos agora como funciona a rotina de inicializacão, que aparece na figura VI.5. Sucessivamente, buscamos os pares de elementos minimais a; e b_i e os retiramos do poset, até que o poset se torne o poset vazio. Se durante o processo um elemento mínimo é encontrado, simplesmente o removemos do poset (este elemento não fará parte de nenhum par). Seja ParMax o índice do último par de elementos minimais encontrado em $\mathcal{Q}$, o restante de $\mathcal{Q}$ sendo uma ordem total ou o poset vazio. ParMax não é unicamente determinado pelo poset $\mathcal{P}$ (assim como não o são os pares de elementos encontrados), mas depende da ordem como os elementos foram retirados do poset. Durante o restante deste capítulo, a;, b;, $1 \leq$ i $\leq$ ParMax, sempre denotarão os pares de elementos minimais como encontrados pela rotina de inicializacão.

Dizemos que uma extensão linear 1 está propriamente i-ordenada se, para todo $1 \leq j \leq$ i, os elementos a_j e b_j são adjacentes em 1 e 1 induz a ordem $a_1 a_2 \ldots a_i a_h$, para todo h tal que i $<$ h $\leq$ ParMax. Reparem que a extensão inicial obtida está propriamente $ParMax$-ordenada, o que é vital para o funcionamento correto do algoritmo.

Assumindo que $Dir(B[ParMax + 1])$ é falso, a chamada principal do algoritmo poderia ser simplesmente $GerExt(ParMax+1)$, correspondendo à seguinte seqüência de chamadas :

$$GerExt(\text{ParMax}); TrocaContexto(ParMax); GerExt(ParMax);$$

uma vez que $Posit$ tenha o valor VERDADEIRO, e Ext, A e B tenham sido propriamente inicializados como na figura VI.5.

Agora provaremos o seguinte teorema que diz respeito à correção do algoritmo :

Teorema 13 O algoritmo descrito gera as extensões lineares de acordo com um caminho hamiltoniano em $G(\mathcal{P}) \times K_2$.

Prova : Para provar este teorema, consideraremos primeiro a afirmativa abaixo.

Afirmativa : Seja $\xi = \delta x_i y_i \gamma$, {x;, y;) = {a;, b_i}, a extensão linear corrente em Ext, onde ξ está propriamente i-ordenada. Então, para cada extensão linear $l \in E(\mathcal{P} + x_i y_i + \gamma)$, $GerExt(i)$ gerará cada par de extensões $+l, -l$ exatamente uma vez. Além disso, se i = 1, então a última extensão gerada será $-\xi$ e, se i > 1, a última extensão gerada será $+\xi'$, onde ξ' difere de ξ por uma transposição de a_{i-1} e b_{i-1}.

Prova : A prova procede por indução em i. Se i = 1, a chamada recursiva $GerExt(0)$ não faz nada e δ é induzido por $\mathcal{P}$ (uma vez que a_1, b_1 é o primeiro par de elementos minimais encontrado; ou seja, os elementos que precedem a_1 e b_1 em uma extensão propriamente 1-ordenada de $\mathcal{P}$ induzem necessariamente uma ordem total em $\mathcal{P}$). É fácil verificar que o procedimento da figura V1.6, quando retiradas suas chamadas recursivas e quando $TrocaContexto$ simplesmente reverte o sinal da extensão corrente, segue exatamente o caminho mostrado na figura VI.3 (ou o caso um pouco modificado, quando $mr(y_1)$ é par). Neste caso, $GerExt(1)$ acha um caminho hamiltoniano que leva de $+\xi$ a $-\xi$ em $G(\mathcal{Q}) \times K_2$, onde $\mathcal{Q}$ é o 2B-poset $\mathcal{P} + x_1 y_1 + \gamma$.

Se i > 1 assumimos, sem perda de generalidade, que o sinal da extensão corrente quando na chamada de $GerExt(i)$ é "+". Então, existem a, @, tais que $+\xi = +\alpha x_{i-1} y_{i-1} \beta x_i y_i \gamma$, onde $\{x_{i-1}, y_{i-1}\} = \{a_{i-1}, b_{i-1}\}$, pois ξ está propriamente i-ordenada. Pelo modo como os pares de elementos minimais foram selecionados, podemos garantir que $\mathcal{P}$ induz o poset $\beta(a_i \cup b;)$, onde β pode ser vazio. Como já mencionado, a estrutura básica do algoritmo ao se retirar suas chamadas recursivas segue o caminho hamiltoniano de um 2B-poset indicado na figura VI.3, onde o procedimento $TrocaContexto(i)$ transpõe x_{i-1} e y_{i-1} (a_{i-1} e b_{i-1}). Deste modo, geramos todas as extensões em $+\mathrm{E}(\mathcal{P} + \alpha(a_{i-1} \cup b_{i-1})\beta(x_i y_i \cup \mathrm{y}))$, todas propriamente i-ordenadas.

Ao passo que cada extensão $+l = +\alpha x_{i-1} y_{i-1} \beta \zeta$ (ou $+l' = +\alpha y_{i-1} x_{i-1} \beta \zeta$), onde ζ E E(x;y; + y), é gerada, $GerExt(i-1)$ é chamada em $+l$ ($I[1]). Pela hipótese de indução, esta chamada gera $\pm E(\mathcal{P} + x_{i-1} y_{i-1} + \beta\zeta)$ (ou $\pm E(\mathcal{P} + y_{i-1} x_{i-1} + \beta\zeta)$, respectivamente), partindo de $+l$ $(+l')$ e terminando em $+l'$ $(+l)$, se i > 2, ou terminando em $-l$ $(-l')$, se i = 2. Como o número de vértices do produto de um grafo com uma aresta é sempre par, teremos um número par de chamadas a $GerExt(i-1)$. Assim, se i > 2, o sinal da permutação final gerada permanece inalterado, enquanto se i = 2, é

a ordem relativa entre a_{i-1} e b_{i-1} na permutação final que permanece inalterada. A união sobre todos os ζ nos dá $\pm E(\mathcal{P}+(a_{i-1}b_{i-1}\cup b_{i-1}a_{i-1})+x_iy_i+\gamma) = \pm E(\mathcal{P}+x_iy_i+\gamma)$. □

Sejam a e b, $A[ParMax]$ e $B[ParMax]$ respectivamente, e suponhainos que a rotina de inicialização seguida da seqüência de chamadas

$$GerExt(ParMax); TrocaContexto(ParMax); GerExt(ParMax)$$

foram aplicadas sobre $\mathcal{P}$. Pela afirmativa que acabamos de provar e pelo fato que a extensão inicial definida pelo algoritino estar propriamente $ParMax$-ordenada, a primeira chamada a GerExt gera as extensões de ±E (P + ab), achando um caminho harniltoniano em G(P + ab) x K_2; então a e b são transpostos e as extensões de $\pm E(\mathcal{P}$ + ba) são geradas segundo um caminho hamiltoniano em G(P + ba) × K_2 (é válido ressaltas que $G(\mathcal{P}+ab)$ e $G(\mathcal{P}+ba)$ não são necessariamente isornorfos). Assim, um caminho hamiltoniano é encontrado também em $G(\mathcal{P}) \times K_2$, cada extensão de $\pm E(\mathcal{P})$ é gerada uma única vez segundo este caminho e o teorema fica provado. □

Análise de Complexidade Se mantemos a inversa do vetor Ext, podemos trivialmente implementar os procedimentos Move e $TrocaContexto$, assim como a função Dir, em tempo constante. Cada chamada de Move e $TrocaCont$exto gera uma nova extensão linear de $\pm E(\mathcal{P})$. Cada iteração dos *loops* do procedimento GerExt gera, portanto, uma nova extensão. A cada extensão gerada no nível i, uma chamada a $GerExt(i-1)$ é efetuada. Quando i = 0, chamamos GerExt recursivamente sem que nenhuma nova extensão seja gerada dentro desta chamada. Mas isto ocorre somente uma vez por extensão, não contribuindo para um aumento da ordem de complexidade. Ainda temos uma chamada a $GerExt(i-1)$ para cada chamada $GerExt(i)$, i > 0, que também não gera nenhuma nova extensão. Veremos agora que mesmo estas chamadas ficam bem "distribuídas", quando consideramos o tempo médio de geração por extensão.

Modificaremos um pouco o algoritmo para que possamos provar sua complexidade linear. Ao invés de reconhecer a geração de uma nova extensão a cada chamada de Move ou $TrocaContexto$, façamos isto dentro de cada chamada $GerExt(0)$. Não é difícil concluir que a alteração feita não acarreta mudança real no algoritmo nem em sua complexidade, uma vez que a cada nova extensão linear gerada no nivel i, teremos

uma seqüência de chamadas de $GerExt(j)$, para j variando de i -1 a 0, que não gera nenhuma nova extensão. Agora, analisemos a árvore de computação referente à execução deste algoritmo, onde cada nó interno corresponde a uma chamada recursiva de $GerExt$(i), i $>$ 0, e cada folha corresponde a uma chamada de $GerExt(0)$ (ou a uma nova extensão linear gerada, se preferir). A quantidade total de computação envolvida pode ser dividida de modo que cada nó da árvore receba uma quantidade constante de computação. Analisemos esta última afirmação : dentro de cada chamada a $GerExt(i)$, i $>$ 0, temos uma quantidade fixa de computação associada mais uma quantidade que varia com o número de iterações de cada loop desta chamada. Mas, a cada iteração de um loop em $GerExt(i)$ corresponde uma nova chamada a $GerExt(i-1)$, ou seja, a mais um nó na árvore.

Observemos que cada chamada $GerExt(i)$, i $>$ 0, gera ao menos duas chamadas a $GerExt(i-1)$. Então, cada nó interno da árvore possui ao menos dois filhos e o número de folhas da árvore é maior que o número de nós internos. Seja f o número de folhas desta árvore, que sabemos ser igual ao número de extensões em $\pm E(\mathcal{P})$. Daí segue que a quantidade total de computação é $\leq kf$, onde k é uma constante. Se só listamos as extensões positivas de $\pm E(\mathcal{P})$, temos um algoritmo de tempo médio constante para gerar as extensões de E(P) que leva, em média por extensão, duas vezes o tempo necessário para se gerar uma extensão de $\pm$E(7').

VI.3 Gerando as Extensões com Atraso Dois

Nesta seção, mostraremos como modificar o algoritmo de modo que cada duas extensões listadas sucessivamente sempre difiram por uma ou duas transposições adjacentes. Antes, mostraremos a existência de uma seqüência das extensões tais que extensões sucessivas difiram por no máximo três transposições. Deste modo, estaremos contribuindo com mais um conjunto de objetos cuja geração define algum código de Gray (ver capítulo III).

Se a e β são extensões de $\mathcal{P}$, por $D(\alpha,\beta)$ denotaremos a distância (comprimento do menor caminho) entre a e β em $G(\mathcal{P})$; e por $D'(\alpha,\beta)$ a distância correspondente em $G'(\mathcal{P})$. Uma ordenação $\alpha_1,\alpha_2,\ldots,\alpha_{e(\mathcal{P})}$ das extensões de $\mathcal{P}$ tem atraso k se $D'(\alpha_i,\alpha_{i+1}) \leq k$, 0 $\leq$ i $<$ $e(\mathcal{P})$, onde $\alpha_0 = \alpha_{e(\mathcal{P})}$. Então, nosso objetivo é mostrar a existência uma ordenação com atraso 2 em $E(\mathcal{P})$. Ainda mais, que esta ordenação

pode ser encontrada em tempo médio constante. A existência de uma ordenação com atraso **3** não é difícil de se mostrar.

Se G é um grafo, por G^k denotaremos o grafo com mesmo conjunto de vértices que G, mas onde existe uma aresta para cada par de vértices entre os quais existe um caminho de comprimento $\leq k$ em G. Em outras palavras, se M é a matriz de incidência de G, então M^k é a matriz de incidência de G^k, onde as operações consideradas são binárias. G^3 é o cubo de G e G^2, o quadrado. Um resultado de Sekanina [Sek60] mostra que o cubo de todo grafo conexo é hamiltoniano. Como $G'(\mathcal{P})$ sempre é conexo, $G'(\mathcal{P})^3$ é hamiltoniano e uma ordenação das extensões de $\mathcal{P}$ com atraso **3** existe.

Infelizmente, o resultado que temos com relação ao quadrado de um grafo G não é tão forte quanto o citado para G^3, não sendo suficiente para provar a existência de uma ordenação com atraso 2. Fleischner [Fle74] obteve que o quadrado de todo grafo biconexo é hamiltoniano.

Um grafo está na classe $\mathcal{H}(s,t)$ se possui um ciclo (não simples) que visita cada vértice do grafo pelo menos s vezes e no máximo t vezes. (Veja [JW90, Pru90].) Assim, $\mathcal{H}(1,1)$ é a classe dos grafos hamiltonianos. Observemos que, se $G \times K_2$ é hamiltoniano, então G E $\mathcal{H}(1,2)$ – basta consideras o caminho resultante ao se identificar as duas cópias de G. Portanto, o teorema 12 diz que $G'(\mathcal{P})$ está em $\mathcal{H}(1,2)$. Em geral, estes grafos não são hamiltonianos.

Conjectura 2 *Se* $G \times K_2$ é hamiltoniano, então G^2 é harniltoniano.

O grafo $K_{2,6}$ mostra que a volta da conjectura 2 é falsa. Se a conjectura fosse verdadeira então, pelo teorema 12, existiria uma ordenação com atraso 2. Outra conjectura mais forte que a 2 é a de que G E $\mathcal{H}(1,2)$ implica em G^2 ser hamiltoniano.

Na definição do procedimento $GerExt$ na seção anterior, a geração das extensões negativas só ocorria no nível i = 1. Isto sugere que o caso i = **1** seja tratado como um caso especial. Poderíamos simplesmente omitir os vértices "–", passando para o próximo vértice positivo : teríamos algum ganho computacional, mas continuaríamos a ter a mesma ordenação das extensões que antes, e nada se poderia afirmar sobre o número de transposições que diferenciariam cada par de extensões sucessivas.

Para se garantir que extensões sucessivas difiram por uma ou duas transposições, é necessário se modificar os caminhos seguidos no caso i = 1. Um exemplo de modi-

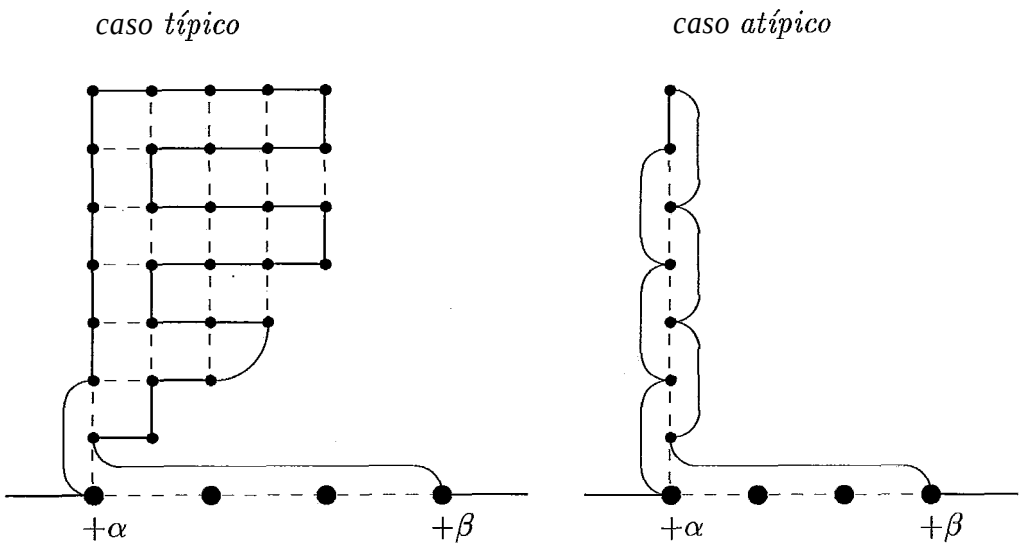


Figura VI.7: Caminhos a serem seguidos para código de Gray com atraso *2*.

ficação pode ser visto na figura VI.7, onde $+a$ e $+\beta$ são extensões sucessivas dentro de uma chamada $GerExt(2)$ (ou seja, não considerando as extensões geradas *efetivamente* dentro de uma chamada $GerExt(1)$). Esta modificação do procedimento $GerExt(i)$ quando $i = 1$, não implica em nenhuma alteração na complexidade do algoritmo e, portanto, podemos afirmar :

Teorema *14 As extensões lineares de qualquer poset podem ser geradas com atraso 2 em tempo médio constante.*

No capítulo VII, veremos como melhorar o algoritmo se quisermos simplesmente contar as extensões de um poset – o problema de enumeração de extensões admite, então, um algoritmo de tempo $O(\mathcal{P}\backslash\{a_1, b_1\})$. Este é o algoritmo mais eficiente conhecido para se contar as extensões de um poset no caso geral.

VI.4 Observações Finais

Segundo Pruesse & Ruskey [PR91], os algoritmos para gerar e contar as extensões de um poset foram totalmente implementados em C e rodados em uma estação de trabalho Sun SPARC SLC, se mostrando bastante eficientes na prática. Para certos exemplos de posets com mais de 2.000.000 de extensões, os algoritmos levavam cerca de 4 segundos para gerá-las e menos de 2 segundos para contá-las.

Capitulo VII

Enumeração de Extensões Lineares de Posets

Como medir a *quantidade de informação de ordem* (ou *grau de ordenação)* contida em um dado poset ? Sejam $\mathcal{P}$ e $\mathcal{Q}$ dois posets sobre um mesmo conjunto de elementos. Existem vários parâmetros de comparação segundo os quais pode se considerar $\mathcal{P}$ como contendo mais informação de ordem que $\mathcal{Q}$. Um parâmetro seria analisar se $\mathcal{Q}$ é induzido por $\mathcal{P}$ ou não, contendo mais informação de ordem no primeiro caso e nada se afirmando no segundo. Entretanto, este parâmetro de comparação é por demais forte, uma vez que em vários casos gostaríamos de poder dizer que $\mathcal{P}$ contém mais informação de ordem que $\mathcal{Q}$, sem que a definição acima nos permita isto.

Suponhamos que $\lambda(\mathcal{P})$ seja um parâmetro teórico de ordem de um poset $\mathcal{P}$. $\lambda(\mathcal{P})$ poderá ser adotado como medida do conteúdo de ordem de $\mathcal{P}$ se λ varia *monotonicamente* quando relações de ordem são adicionadas ao poset. Assim, parâinetros como a altura de um poset, sua largura, seu número de comparabilidades (pares de elementos comparáveis) ou seu número de extensões lineares podem ser adotados como tal. Medidas como altura e largura de um poset são pasâmetros pouco discriminantes, uma vez que assumem valores entre 0 e n, onde n é o número de elementos de $\mathcal{P}$, somente. O número de comparabilidades de um poset pode assumir valores entre 0 e $n(n-1)/2$ (é fácil mostrar que todos estes valores são, de fato, possíveis). Mas, dentre as medidas citadas, é o número de extensões lineares de um poset $\mathcal{P}$ a mais discriminante de todas, apesar de nem todo valor entre 1 e n! ser um valor possível para $e(\mathcal{P})$. Por exemplo, valores estritamente entre $n!/2$ e n! não podem ocorrer, uma

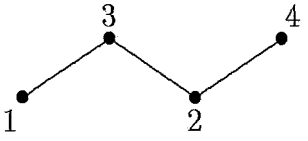


Figura VII.1: Um poset cerca.

vez que a adição de uma relação qualquer ao poset com ordem parcial vazia reduz o conjunto de extensões lineares do último à metade. Ou seja, enquanto o poset com número de comparabilidades igual a zero tem $e(\mathcal{P}) = n!$, o poset com um par de elementos comparáveis tem $e(\mathcal{P}) = n!/2$. Além disso (ou talvez em função disso), $e(\mathcal{P})$ é também a menos trivial de todas estas medidas : para o caso geral, o problema de enumeração *foi* provado ser #P-completo [BW90]. O número de extensões de um poset é uma medida decrescente da quantidade de informação presente em um poset, ou seja, quanto maior $e(\mathcal{P})$, menos informação de ordem contém o poset $\mathcal{P}$.

Como este número pode, no pior caso, ser $O(n!)$, a idéia de envolver a geração das extensões para poder contá-las não é um procedimento eficiente. É o que acontece na fórmula de recorrência 11.1. Esta fórmula classifica as extensões segundo seus primeiros elementos. Poderíamos ter estabelecido a recorrência sobre o último elemento de cada extensão, usando o conjunto $Max(\mathcal{P})$ ao invés de $Min(\mathcal{P})$. Referindo-se à dificuldade do problema de enumeração, Knuth [Knu68] mostra que o número de extensões lineares de um poset com n elementos pode nem sequer ser um divisor de n!, como acontece com o poset da figura VII.1, que admite cinco extensões lineares (1234, 1243, 2134, 2143 e 2413).

O número de extensões de um poset $\mathcal{P}$ é fundamental dentro da teoria de conjuntos ordenados, sendo também de grande interesse dentro da ciência da computação devido à sua forte conexão com problemas de ordenação. Por exemplo, em cada estágio de um algoritmo de ordenação baseado em comparação, a informação corrente pode ser expressa como uma ordem parcial do conjunto de dados; o conjunto de extensões lineares desta ordem restringe o conjunto de soluções possíveis para o problema de ordenação. Se fosse sempre fácil se calcular $e(\mathcal{P})$, poderia se determinar em uma ordenação seqüência1qual seria o par de elementos ótimos a ser comparado em seguida. (Kahn & Saks [KS84] provaram que sempre existe um par cuja comparação de seus

elementos divide o conjunto de extensões lineares em proporções 3/11 : 8/11, mas a prova apresentada não nos dá meios para determinação deste par. Há ainda a conjectura formulada por Linial [Lin84] que propõe a existência de um par de elementos x e y, tais que $1/3 \leq Pr(x \prec y) \leq 2/3$, onde $Pr(x \prec y)$ é a probabilidade de x preceder y em uma extensão linear do poset.)

Outra aplicação aparece em ciências sociais, quando uma lista de alternativas (por exemplo : produtos, candidatos a um emprego, atletas em uma competição) deve ser determinada a partir de uma ordem parcial (veja [FG86]). Uma maneira natural de se construir esta lista é segundo a "altura média" de cada elemento x (dada pela média das alturas de x em todas as extensões lineares). Aqui, novamente, o número de extensões deve ser calculado e, podendo este ser exponencial no número de elementos, não parece provável que métodos eficientes possam ser encontrados para o problema da altura média. O número de extensões lineares também aparece em limites inferiores para os problemas de se ordenar e produzir um poset $\mathcal{P}$ dentro da teoria da informação.

Infelizmente, de um ponto de vista teórico, determinar o número $e(\mathcal{P})$ tem, por si só, se mostrado frustantemente difícil, mesmo para subclasses de posets com propriedades bem restritas. Mais tarde veremos certas classes de posets para as quais existem algoritmos polinoiniais de enumeração. Entre estas, poderíamos citar os casos onde o poset é uma floresta, é série-paralelo, ou tem largura delimitada por uma constante L. O caso de altura limitada e, portanto, também o caso geral continuam sem solução.

O problema de se determinar exatamente $e(\mathcal{P})$ no caso geral há muito se suspeita ser intratável. O problema se encontra claramente na classe **#P**, introduzida por Valiant [Val79a] nos anos 70, uma vez que é fácil se checar se uma extensão linear é consistente com um poset dado. Desde então, conjecturava-se que o problema de enumeração de extensões fosse #P-completo, sendo assim provavelmente muito difícil (especialmente na visão dada pelo resultado de Toda [Tod89], o qual implica que uma chamada a um oráculo $\#P$ seja suficiente para resolver qualquer problema na hierarquia polinomial em tempo polinoinial determinístico). Até onde se sabe, o primeiro a propor tal conjectura foi Linial [Lin86], se referindo à questão como sendo "muito intrigante". Lovász [Lov86], pág. 61, menciona o problema, assim como muitos outros autores também já o consideraram.

Vários outros problemas de enumeração (ou contagem) precederam o da enumeração de extensões lineares de posets na sua caracterização como #P-completo. Entre estes estão a enumeração de anticadeias em uma ordem parcial [PB83], a enumeração de orientações acíclicas de um grafo [Lin86], calcular o número de extensões lineares de alguns tipos especiais [KTdo, Stedo] ou determinar o volume de um corpo convexo em um espaço Euclideano [DF].

Apresentaremos agora uma breve descrição da prova utilizada por Brightwell & Winkler em [BW90] para mostrar que o problema central deste capítulo é $\#P$-completo. Em seguida, descreveremos pequenas modificações a serem introduzidas no algoritmo visto no capítulo VI para geração das extensões no caso geral, que nos fornecerão o algoritmo mais eficiente conhecido para o problema de enumeração das mesmas, além de citarmos um resultado bem satisfatório obtido por meio de um algoritmo aproximativo. Na última seção, mostraremos algumas classes de posets para as quais existe uma fórmula ou, senão, um algoritmo de tempo polinomial que nos dê seu número de extensões. Dentre estas, primeiramente falaremos sobre as classes cujo problema de geração foi considerado em capítulos anteriores.

VII.1 *O* Problema é #P-completo

O método da prova é direto, mostrando que, com o auxílio de um oráculo que conte as extensões lineares, uma máquina de Turing pode contar o número de atribuições satisfatórias para uma instância de 3-SAT em tempo polinomial. Este contrasta com outros resultados #P-completos, como em [Lin86, PB83], que utilizam a sistemática desenvolvida em Valiant [Val79b]. De qualquer modo, ainda seguimos Valiant na abordagem do problema via enumeração módulo vários números primos distintos.

Enumeração de Extensões Lineares
Entrada. Um poset $\mathcal{P}$.
Saída. O número $e(\mathcal{P})$ de extensões lineares de $\mathcal{P}$.

Neste trabalho, vamos usar o fato básico, provado em [Val79a], de que o seguinte problema é #P-completo :

Enumeração de 3-SAT
Entrada. Uma fórmula proposicional **I** na forma normal 3-conjuntiva.
Saída. O número $s(I)$ de atribuições satisfatórias para **I.**

O resultado principal em [Val79a] mostra que calcular o permanente de uma matriz 0-1 (o que equivale a se contar o número de emparelhamentos completos em um grafo bipartite) é #P-completo. Este é um exemplo onde o problema de decisão "Existe um permanente para dada matriz 0-1 M ?" (ou "Existe um emparelhamento completo para dado grafo bipartite G ?") está em **P,** mas o problema de enumeração correspondente é #P-completo. O resultado que apresentamos é um exemplo ainda mais extremo deste fenômeno, uma vez que o problema de decisão associado "Existe uma extensão linear para dado poset $\mathcal{P}$?" é trivial : todo poset possui pelo menos uma extensão linear. Outro caso extremo ocorre com relação às cliques maximais de um grafo, uma vez que todo grafo admite pelo menos uma clique maximal por definição (encontrar uma clique maximal em dado grafo com n elementos é $O(n)$ e o problema de enumeração correspondente se encontra em $\#P$).

Teorema 15 Enumeração de Extensões Lineares é #P-completo.

Usaremos o seguinte fato à respeito a distribuição de números primos. A prova deste lema pode ser encontrada em [BW90].

Lema *11* Para todo $n \geq 4$, o produto do conjunto de primos estritamente entre n e n^2 é ao menos $n!\, 2^n$.

É evidente que o limite superior n^2 do lema 11 é por demais generoso; é possível se substituir este limite por $Kn \log n$, para alguma constante K suficientemente grande.

Não nos deteremos com a prova formal do teorema 15, encontrada em [BW90], mas apenas daremos as linhas gerais utilizadas por Brightwell & Winkler. Suponhamos que uma instância I de Enumeração de 3-SAT com m variávies e n cláusulas seja dada, e que tenhamos um oráculo que retorne o número de extensões lineares de um poset $\mathcal{P}$ qualquer de tamanho no máximo polinomial em n e m. O primeiro passo é construir, a partir da instância **I,** um poset auxiliar $\mathcal{P}_I$ de tamanho $(7n + m)$ e usar o oráculo para calcular o número de extensões lineares $e(\mathcal{P}_I)$. No próximo passo, acharemos

um conjunto X de primos entre (7n + m) e $(7n + m)^2$, cujo produto seja ao menos 2^m e tal que nenhum primo em X divide $e(\mathcal{P}_I)$, usando o lema 11 e a desigualdade $e(\mathcal{P}_I) \leq (7n + m)!$. Nosso objetivo é achar o número de atribuições satisfatórias de **I**, módulo p, para cada primo $p \in X$. Como o número de atribuições satisfatórias pode ser no máximo 2^m teremos, assim, determinado o número de atribuições satisfatórias de **I**.

Para cada primo p E X, formamos um poset $\mathcal{Q}_I(p)$ de tamanho aproximadamente $p(m + n)$, com a seguinte propriedade : o número de extensões de $\mathcal{Q}_I(p)$ pode ser escrito como $ap + s(I)$ bc $e(\mathcal{P}_I)$, onde a é um inteiro positivo, b e c são inteiros não divisíveis por p facilmente calculáveis (dependentes de p, n e m), e $s(I)$ é o número de atribuições satisfatórias de **I**. Então, achamos $e(\mathcal{Q}_I(p))$ com o auxílio do oráculo, o que corresponde a $s(I)$ bc $e(\mathcal{P}_I)$, módulo p. Agora, podemos achar $s(I)$ módulo p, como desejado.

VII.1.1 Alguns Problemas Relacionados

Discutiremos agora as implicações do teorema 15 para dois outros problemas fortemente relacionados com o problema de enumeração de extensões lineares [BW90].

Para dois elementos incomparáveis x e y de um poset $\mathcal{P}$, $Pr(x \prec y \mid \mathbf{P})$ (ou, abreviando, $Pr(x \prec y)$) denota a probabilidade de x preceder y em uma extensão linear de **P** escolhida arbitrariamente. Então, $Pr(x \prec y) = e(P \cup \{(x,y)\})/e(\mathcal{P})$.

Estritamente falando, o problema de se calcular $Pr(x \prec y)$ não pertence à classe **#P**, uma vez que este não é um problema de enumeração. Entretanto, do ponto de vista do teorema 16, ele pode ser considerado como um problema #P-completo.

Teorema 16 O problema de se calcular $Pr(x \prec y)$ em um poset $\mathcal{P}$ é polinomialmente equivalente a Enumeração de Extensões Lineares.

Prova : Dado um oráculo para Enumeração de Extensões Lineares, o aplicamos aos posets $\mathcal{P}$ e $\mathcal{P} \cup \{(x, y)\}$ e derivamos $Pr(x \prec y \mid P)$. Portanto, calcular $Pr(x \prec y)$ não é mais difícil que o problema de enumeração das extensões.

Para provar a volta, seja **P** uma instância de Enumeração de Extensões Lineares com n vértices, e suponhamos que tenhamos um oráculo que calcula $Pr(x \prec y \mid \mathcal{Q})$ em tempo constante, para qualquer par de elementos incomparáveis x e y em um poset $\mathcal{Q}$

com n elementos. Sejam (a;, b;), $1 \leq$ i $\leq M$, os pares de cobertura de $\mathcal{P}$, i. e., os pares (a, b) tais que a $\prec$ b, mas não existe c tal que $a \prec$ c $\prec$ b, onde a, b, c são elementos de $\mathcal{P}$. O número M desses pares pode ser no máximo $n^2/4$ (grafo bipartite com $n/2$ elementos em cada partição e número máximo de arestas). Definimos uma seqüência de posets com n vértices, como se segue : o poset $\mathcal{P}_0$ é uma anticadeia de tamanho n. Para $1 \leq$ i $\leq$ M, o poset $\mathcal{P}_i$ é definido recursivamente como $\mathcal{P}_{i-1}$ U {(a;, b;)). Deste modo $\mathcal{P}_M = \mathbf{P}$.

Temos que $e(\mathcal{P}_0) =$ n! e, para $1 \leq$ i $\leq M$, $e(\mathcal{P}_i)/e(\mathcal{P}_{i-1}) = Pr(a_i \prec b_i \mid \mathcal{P}_{i-1})$. Então, $e(\mathcal{P}) = e(\mathcal{P}_M)$ é dado por

$$n! \prod_{i=1}^{M} Pr(a_i \prec b_i \mid \mathcal{P}_{i-1}),$$

podendo ser rapidamente calculado em A4 $\leq$ $n^2/4$ chamadas ao oráculo. □

O cálculo de $e(\mathcal{P})$ na prova do teorema 16 pode ser feito em apenas $k \leq 2n \log n$ chamadas ao oráculo, se levarmos em consideração alguns resultados obtidos com algoritmos aproximativos para o problema (veja Brightwell & Winkler [BW90]) na construção da sequência de posets $\mathcal{P}_j$.

Um segundo problema relacionado diz respeito a se determinar a altura média de um elemento em um poset. Se x é um elemento de um poset $\mathcal{P}$ e l é uma extensão linear deste poset, então a altura média $H(x \mid \mathbf{P})$ de x em $\mathcal{P}$ é a média das alturas $h(x, 1)$ em todas as extensões 1 de P .

Teorema 17 O problema de se determinar a altura média de um elemento de um poset é polinomialmente equivalente ao problema de se determinar $Pr(x \prec$ y).

Prova : Temos as duas igualdades :

$$\begin{aligned} H(x \mid \mathcal{P}) &= \sum_{y \neq x} Pr(y \prec x \mid \mathcal{P}) + 1; \\ H(x \mid \mathcal{P}) &= Pr(y \prec x \mid \mathcal{P}) H(x \mid \mathcal{P} \cup \{(y,x)\}) + \\ &\quad Pr(x \prec y \mid \mathcal{P}) H(x \mid \mathcal{P} \cup \{(x,y)\}), \text{ para algum } y \neq x. \end{aligned}$$

A primeira igualdade nos permite calcular $H(x|\mathcal{P})$, dado um oráculo para $Pr(x \prec$ y); a segunda nos permite calcular $Pr(x \prec y)$ dado um oráculo para calcular a altura média, lembrando que $Pr(y \prec$ x) $= 1 - Pr(x \prec y)$. O

Assim, o problema de se determinar a altura média de um elemento em um poset é também polinomialmente equivalente a um problema #P-completo.

Além dos dois problemas mencionados, o problema de se calcular o volume de um politopo convexo de dimensão n foi mostrado ser fortemente #P-completo (veja [Khado]) devido ao resultado obtido por Brightwell & Winkler. Por ser fortemente #P-completo entendemos que o problema pode ser caracterizado como #P-completo mesmo se representamos os dados de entrada do problema em base unária.

VII.2 Considerando Algoritmos para o Caso Geral

O algoritmo $O(e(\mathcal{P}))$ apresentado no capítulo VI é ótimo com relação ao problema de geração das extensões lineares de um poset. Ele também é o algoritmo mais eficiente conhecido para se contar estas extensões. Algumas pequenas modificações podem ser introduzidas neste caso, de modo a se baixar ligeiramente sua complexidade. Um outro algoritmo para o problema de enumeração pode ser encontrado em Wells [Wel71], que segue a estrutura de "dividir para conquistar". Este algoritmo se baseia nas regras do produto e da soma de eventos mutualmente exclusivos, mas se mostra extremamente redundante na aplicação e combinação das mesmas. Em Atkinson [Atk85] também encontramos um algoritmo para a enumeração de extensões, que se mostra na prática ainda menos eficiente que o de Wells.

Assumimos que o leitor já esteja familiarizado com o algoritmo apresentado no capítulo VI e, portanto, não nos preocuparemos aqui em redefiní-lo ou em re-introduzir a notação necessária. Se queremos simplesmente contar o número de extensões de um poset $\mathbf{P}$ qualquer, algum ganho computacional pode ser obtido no nível $i = 1$ da recursão envolvida, não gerando as extensões neste nível explicitamente. Ou seja, nunca movendo a_1 e b_1, o par de elementos minimais de $\mathcal{P}$ primeiramente selecionado pela rotina de inicialização.

Sejam a o elemento mais à esquerda e b o mais à direita do par a_1, b_1 em uma extensão de $\mathcal{P}$. Usando a definição de $mr(x)$ presente naquele capítulo, o número de vértices do grafo $G(\mathcal{P})$ (extensões lineares de $\mathbf{P}$) pode ser determinado a partir dos valores $mr(a)$ e $mr(b)$ nas extensões de $\mathbf{P}$, que correspondem aos valores assumidos

pelas variáveis $NMovADir$ e $NMovBDir$, respectivamente, ao final de cada chamada $GerExt(1)$. É trivial verificar que o número de extensões que seriam geradas dentro de uma chamada $GerExt(1)$, para os valores de $mr(a)$ e $mr(b)$ correspondentes, é dado por $(mr(a) + 1)[mr(b) + 1 - mr(a)/2]$. Além disso, analisando somente a seqüência de extensões geradas nos níveis $i > 1$, temos que $mr(a)$ e $mr(b)$ variam de no máximo uma unidade entre duas extensões sucessivas nesta sequência, uma vez que em níveis mais altos $(i > 1)$ da recursão são usadas somente transposições adjacentes para a geração destas extensões. Assim, podemos calcular $mr(a)$ e $mr(b)$ em tempo constante dentro de cada chamada $GerExt(1)$, o que nos conduz a um algoritmo de complexidade $O(e(\mathcal{P}\backslash\{a_1, b_1\}))$ para contar as extensões de $\mathcal{P}$. Em geral, temos

$$2\, e(\mathcal{P}\backslash\{a_1, b_1\}) \leq e(\mathcal{P}) \leq n(n-1)\, e(\mathcal{P}\backslash\{a_1, b_1\}).$$

O limite inferior é alcançado quando $a_1 \prec x$ e $b_1 \prec x$, para todo x elemento de $\{a\, b_1\}$. O limite superior é alcançado quando a_1 e b_1 são também elementos maximais, assim como minimais, em $\mathcal{P}$.

Pelo teorema 15 sabemos que é muito improvável a existência de um algoritmo eficiente para o problema de enumeração de extensões de posets. Nestas circunstâncias, uma alternativa razoável seria poder ao menos aproximar um valor para este número. Isto é de fato possível, e gostaríamos de contrastar o teorema *15* com o resultado que se segue, atribuído a Dyer, Frieze & Kannan [DFR89].

Teorema 18 *Existe um algoritmo randômico A com as seguintes propriedades : os dados de entrada consistem de uma ordem parcial $\mathcal{P}$ com n elementos e números racionais positivos ε e β; a saída do algoritmo é um número L tal que*

$$Pr\left(\left|\frac{L}{E(\mathcal{P})} - 1\right| < \epsilon\right) > 1 - \beta.$$

Este algoritmo termina em tempo polinomial em n, $1/\epsilon$ e $\log(1/\beta)$.

Tal algoritmo é chamado de *esquema de aproximação randômica totalmente polinomial* para o problema Enumeração de Extensões Lineares. A interpretação adequada é a de que o algoritmo A acha com probabilidade arbitrariamente alta uma aproximação L para o número de extensões lineares, que se encontra distante do número correto dentro de um fator multiplicativo de $(1 + \varepsilon)$. Mais precisamente, a

estimativa de tempo deste algoritmo possui termo dominante n^9, o qual, apesar de longe de parecer ser ótimo, representa um refinamento para estimativas prévias sobre o problema.

Em [BW90], encontramos uma análise (bastante completa) do progresso mais recente na área de algoritmos aproximativos para o problema de enumeração de extensões de posets.

VII.3 Algumas Classes onde o Problema está Resolvido

Nesta seção apresentaremos as classes de posets onde o problema de enumeração foi resolvido. Primeiro, falaremos das classes para as quais analisamos a geração sistemática de suas extensões (florestas, multiconjuntos e posets série-paralelos). Depois, apresentaremos as outras classes, respeitando o que entendemos ser uma ordem de "simplicidade" de cálculo. Dois pontos a ressaltar : as classes de posets consideradas nesta seção foram todas aquelas onde se conseguiu levantar algum resultado na literatura para o problema da enumeração; em momento algum afirmamos que as classe apresentadas são totalmente disjuntas entre si. Por exemplo, os posets floresta se incluem claramente dentro daqueles cujo diagrama de Hasse é uma floresta não enraizada - se fizemos distinção na menção a estas duas classes foi por razões de clareza nos resultados obtidos. Antes de iniciar a apresentação das classes, vejamos o seguinte resultado, que diz que podemos considerar as componentes conexas de um poset independentemente.

Lema **12** *Sejam* $\mathcal{Q}$ *e* $\boldsymbol{R}$ *posets disjuntos em elementos e seja* $\mathcal{P} = \mathcal{Q} \cup \boldsymbol{R}$. *Então* :

$$e(\mathcal{P}) = e(\mathcal{Q})e(\mathcal{R}) \left(\begin{array}{c} |\mathcal{Q}| + |\mathcal{R}| \\ |\mathcal{Q}| \end{array} \right).$$

Este resultado será usado mais tarde no cálculo do número de extensões de um poset série-paralelo, e a prova do item (ii) do teorema 21 pode ser aplicada a este lema.

O teorema a seguir aparece em [Atk85, Atk89]. A prova do mesmo se encontra em [Atk85].

Teorema 19 *Sejam $\mathcal{Q}$ e $\boldsymbol{R}$ dois posets sobre conjuntos disjuntos de elementos X e Y, respectivamente. Seja $x \in X$ e $y \in Y$, Tomemos por P o poset obtido por $\mathcal{Q} \cup \boldsymbol{R}$ adicionado da relação de cobertura $x \prec y$. Então,*

$$e(\mathcal{P} \mid h(x) = r) = \sum e(\mathcal{Q} \mid h(x) = a)\, e(\mathcal{R} \mid h(y) = b) \binom{r-1}{a-1} \binom{s-r-1}{s-a-b} \binom{|X|+|Y|-s}{|X|-s+b},$$

onde o somatório se dá sobre todos os valores de $1 \leq a \leq r$, $1 \leq b \leq |Y|$ e $r < s \leq |X| + |Y|$.

É válido notar que, no teorema anterior, s representa a posição de y em uma extensão de P.

VII.3.1 Florestas e multiconjuntos

Teorema 20 *O número de extensões lineares de um poset floresta $\mathcal{F}$ com n elementos é dado por*

$$e(\mathcal{F}) = \frac{n!}{d_1 d_2 \ldots d_n},$$

onde d_i = número de descendentes do nó i (incluindo o mesmo) na floresta.

Prova : Provaremos o teorema acima por indução em n. O teorema é válido para $n = 1$. Supondo válido para o poset floresta com n elementos, vamos analisar o que acontece com a inserção do $(n+1)$-ésimo elemento z.

Inserindo z em uma posição qualquer (ignorando as relações de precedência que o envolvam) dentro das extensões que tínhamos para n elementos, teremos geradas $(n+1)!/d_1 \ldots d_n$ permutações. Separamos estas permutações em conjuntos, da seguinte maneira : agrupo todas as permutações onde o conjunto de posições ocupadas pelos d_{n+1} descendentes de z (incluindo o próprio) é o mesmo e onde, além disso, a ordem entre os descendentes próprios de z também é constante. Cada um desses conjuntos contém exatamente d_{n+1} permutações, que são caracterizadas a partir da posição de z em relação a seus descendentes próprios, como fazemos abaixo :

z aparece à frente de todos seus descendentes próprios;
z aparece atrás de 1 de seus descendentes próprios;
z aparece atrás de 2 de seus descendentes próprios;
...
z aparece atrás de todos seus d_{n+1} descendentes próprios.

Somente as permutações onde z aparece à frente de todos seus descendentes próprios são compatíveis com o poset. Portanto, somente $1/d_{n+1}$ permutações geradas com a inserção de z são extensões lineares do poset. Ou seja, o número de extensões lineares de um poset floresta com $n+1$ elementos é

$$\frac{(n+1)!}{d_1 \ldots d_n d_{n+1}}$$

□

A fórmula para florestas apresentada acima aparece como exercício em Knuth [Knu68], considerando somente o caso de árvores binárias. Segue um corolário imediato para multiconjuntos (subclasse de florestas)

Corolário 8 *O número de extensões lineares de um multiconjunto com t cadeias é dado por*

$$e(n) = \frac{n!}{n_1! n_2! \ldots n_t!}$$

onde n_i = tamanho da cadeia i

VII.3.2 Posets série-paralelos

A próxima classe para a qual consideraremos o problema da enumeração é a classe dos posets que são representados por digrafos série-paralelos [VTL79], também já definidos.

Teorema 21 *Sejam $\mathcal{P}'(S', R')$ e $\mathcal{P}''(S'', R'')$ dois posets série-paralelos e uma composição a ser aplicada sobre $\mathcal{P}'$ e $\mathcal{P}''$.*

(i) *Se a composição é em série, então*

$$e(\mathcal{P}' \otimes \mathcal{P}'') = e(\mathcal{P}')\, e(\mathcal{P}'')$$

(ii) Se a composição é em paralelo, então

$$e(\mathcal{P}' \oplus \mathcal{P}'') = e(\mathcal{P}')\, e(\mathcal{P}'') \begin{pmatrix} |S'| + |S''| \\ |S'| \end{pmatrix}$$

Prova :

(i) Se a composição efetuada foi em série, então todos os elementos de $\mathcal{P}'$ devem preceder todos os elementos de $\mathcal{P}''$. Ou seja, uma extensão de $\mathcal{P}' \cup \mathcal{P}''$ é a simples concatenação de uma extensão de $\mathcal{P}'$ seguida de uma extensão de $\mathcal{P}''$ e o número de modos distintos que podemos fazer esta concatenação é obviamente igual a $e(\mathcal{P}')\, e(\mathcal{P}'')$.

(ii) Se a composição efetuada foi em paralelo, nenhum dos elementos de $\mathcal{P}'$ se relaciona com algum elemento de $\mathcal{P}''$. Portanto, as extensões de $\mathcal{P}' \cup \mathcal{P}''$ seriam formadas pegando-se uma extensão de "?' e outra de "?" e intercalando-se seus elementos de todas as maneiras possíveis, respeitando a ordem original entre os elementos de cada extensão. Isto seria equivalente a se selecionar $|S'|$ posições, dentre as $|S'| + |S''|$ possíveis, para os elementos de $\mathcal{P}'$, como indica o coeficiente binomial da fórmula. É óbvio que teríamos $e(\mathcal{P}')\, e(\mathcal{P}'')$ opções para a escolha do par de extensões. o

O teorema 21 nos leva ao desenvolvimento de um algoritmo linear para calcular o número de extensões de um poset série-paralelo. Seja $\mathcal{P}(S, \mathrm{R})$ um poset série-paralelo e T sua árvore binária de decomposição. Se percorremos T sistematicamente, de baixo para cima e nível a nível, podemos calcular o número de extensões lineares dos subposets correspondentes às subárvores de T da seguinte forma : a cada novo nó percorrido examina-se a composição associada a este nó, calculando-se o número de extensões relativas à subárvore enraizada nele segundo o teorema 21, uma vez que $e(\mathcal{P}')$ *e* $e(\mathcal{P}'')$ - $\mathcal{P}'$ e $\mathcal{P}''$ sendo os subposets definidos pelas siibárvores à esquerda e à direita do nó, respectivamente - já foram calculados. Assim, ao percorrer a raiz da árvore de decomposição, estaremos calculando $e(\mathcal{P})$, como desejado. Podemos fazer isto em tempo linear ($O(m + \mathrm{n})$, se $\mathrm{n} = |S|$ e m = número de arestas na redução transitiva de R), uma vez que o número de nós internos de T é igual a n – 1. A construção da árvore binária de decomposição também pode ser feita em tempo linear [VTL79].

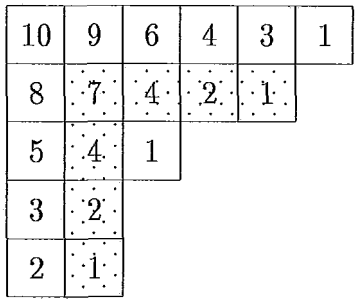


Figura VII.2: Torres e seus comprimentos - a área sombreada mostra a torre da célula (2,2), de comprimento 7.

VII.3.3 Tableaux de Young

Seja agora um tableau de Young de formato $(n_1, n_2, \ldots, n)$. O número de maneiras de se preencher este tableau com os inteiros $\{1, \ldots, n\}$, $n = n_1 + \ldots + n_m$, é obviamente igual ao número de extensões lineares compatíveis com o poset definido pelo mesmo. Definimos a *torre* de uma célula do tableau como sendo a própria célula mais todas as outras que se encontram abaixo ou à direita desta. O *comprimento* de uma torre é dado pelo número de células que a compõem. Veja o exemplo na figura VII.2, onde cada célula contém o comprimento de sua torre.

Seja o seguinte teorema :

Teorema 22 *O número de extensões lineares definidas por um tableau de Young de formato específico com n elementos é igual a n! dividido pelo produto dos comprimentos de suas torres.*

Prova : Este teorema é facilmente provado usando-se um outro teorema de Frank-Robinson-Thrall (enunciado em Knuth [Knu68], pág. 63), que nos dá o número de *tableaux* para um formato específico que, como visto, é igual ao número de extensões lineares do poset definido por este formato. Infelizmente, a prova do teorema de Frank-Robinson-Thrall não é tão trivial quanto pode parecer à primeira vista e não nos cabe aqui apresentá-la. O

VII.3.4 Árvores não-enraizadas

Vamos considerar agora o caso onde o diagrama de Hasse de um poset $\mathcal{P}$ é uma árvore não-enraizada. Para o caso de poset cujo diagrama é uma floresta não-enraizada (ou posets com diagramas sem ciclos), podemos aproveitar os resultados desta seção para cada árvore da floresta independentemente e depois combinar os valores obtidos usando o lema 12. Foi provado por Atkinson em 1985 [Atk85] que o número de extensões de tal poset pode ser calculado em tempo $O(n^5)$. Já em 1990 [Atk90], o próprio Atkinson apresenta um algoritmo $O(n^2)$ para o problema, que admite uma implementação bem simples. Comecemos a descrição deste algoritmo com algumas definições e resultados relevantes.

Para $a \in \mathbf{P}$, o *a-espectro* de $\mathbf{P}$ é a seqüência $(\lambda_1, \lambda_2, \ldots, \lambda_n)$, onde λ_i é o número de extensões lineares de $\mathcal{P}$ tais que $h(\alpha) = i$ (ou seja, a ocupa a i-ésima posição na ordem total). É óbvio que $e(\mathcal{P}) = \sum_i \lambda_i$.

Lema **13** *Sejam $\mathcal{P}$ e $\mathcal{Q}$ dois posets sobre conjuntos disjuntos de tamanhos n e m, respectivamente. Seja $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ o a-espectro de $\mathcal{P}$ e $(\mu_1, \mu_2, \ldots, \mu_m)$ o β-espectro de $\mathcal{Q}$. Consideremos a ordem parcial definida por $\mathbf{R} = \mathcal{P} \cup \mathcal{Q} \cup \{(a, \beta)\}$. Então, o r-ésimo elemento do a-espectro de $\mathbf{R}$ é dado por*

$$\sum_{i=\max(1,r-m)}^{\min(n,r)} \lambda_i k_{ri} \sum_{j=r-i+1}^{m} \mu_j$$

onde

$$k_{ri} = \binom{r-1}{i-1} \binom{n+m-r}{n-i}.$$

A prova do lema anterior pode ser encontrada em [Atk90].

Nota : Existe uma fórmula similar, provada da mesma maneira, para o a-espectro de uma ordem parcial $\prec_{\mathcal{R}}$ cujas relações de cobertura são aquelas de $\mathcal{P}$ e $\mathcal{Q}$ mais a relação $\beta \prec_{\mathcal{R}} a$. Seu r-ésimo elemento é dado por

$$\sum_{i=\max(1,r-m)}^{\min(n,r)} \lambda_i k_{ri} \sum_{j=1}^{r-i} \mu_j.$$

Lema 14 O cálculo da fórmula no lema 13 (e daquela na nota que segue sua prova) pode ser feito em $O(nm)$ operações aritméticas, assumindo que todos os coeficientes *binomiais* a serem utilizados foram calculados anteriormente.

Prova : As quantidades $\omega_k = \sum_{j=k}^{m} \mu_j$, $k = m, m-1, \ldots, 1$, são calculadas a priori em $O(m)$ operações, usando as equações $\omega_m = \mu_m$ e $\omega_k = \omega_{k+1} + \mu_k$, $k < m$. Assumamos, sem perda de generalidade, que $n \leq m$ (o caso $n > m$ é considerado analogamente). Os membros $\sum_{i=\max(1,r-m)}^{\min(n,r)} \lambda_i k_{ri} \omega_{r-i+1}$ da seqüência espectral com $1 \leq r \leq n$ são calculados num total de $c(1 + 2 + \ldots + n) = cn^2/2 + O(n)$ operações, onde c é uma constante; os com $n < r \leq m$, requerem $cn(m - n)$ operações; enquanto que os com $m < r \leq n + m$ necessitam também de $c(1 + \cdots + n) = cn^2/2 + O(n)$ operações para seu cálculo. Somando todas estas operações, temos um total de $cnm + O(n)$ operações aritméticas envolvidas neste processo, como queríamos provar. O

Teorema 23 Seja $\mathcal{T}$ um poset com n elementos, cujo diagrama de Hasse é uma árvore não direcionada e seja $a \in \mathcal{T}$. Então, o a-espectro (e, por conseqüência, $e(\mathcal{P})$) pode ser calculado em $O(n^2)$ operações aritméticas.

Prova : Começamos calculando e guardando todos os coeficientes binomiais $\binom{a}{b}$, $0 \leq b \leq a \leq n$, usando um triângulo de Pascal : isto requer $O(n^2)$ operações. Escolhemos, agora, qualquer aresta do diagrama de Hasse incidente a a. Seja β o outro extremo desta aresta. Ao retirar esta aresta de $\mathcal{T}$, ficamos com duas componentes conexas $\mathcal{P}$ e $\mathcal{Q}$ (de tamanhos u e v) , que representam subposets de $\mathcal{T}$ contendo a e β, respectivamente. O α-espectro de $\mathcal{P}$ e o β-espectro de $\mathcal{Q}$ podem ser calculados recursivamente e, assim, o a-espectro de $\mathcal{T}$ pode ser encontrado, usando-se o lema 13.

Se $T(n)$ é o número de operações aritméticas requeridas pelo algoritmo temos que, pelo lema 14,

$$T(n) \leq T(u) + T(v) + cuv,$$

para alguma constante c. Agora, provamos que $T(n) \leq cn^2/2$ (ou seja, é $O(n^2)$) por indução em n. Como $u, v < n$, substituímos a hipótese de indução na equação anterior, obtendo :

$$T(n) \leq \frac{1}{2}cu^2 + \frac{1}{2}cv^2 + cuv = \frac{1}{2}c(u + v)^2 = \frac{1}{2}cn^2.$$

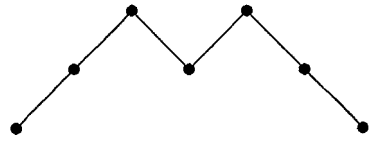


Figura VII.3: Um poset zigzag.

Pela base da indução, n = 1, podemos determinar um valor apropriado para a constante c. □

VII.3.5 Posets zigzag

Estes posets são um caso bem simples da classe de árvores não-enraizadas admitindo, como visto na seção anterior, um algoritmo $O(n^2)$ para o cálculo de seu número de extensões. No entanto, apresentaremos uma outra fórmula para o cálculo do número de extensões de um poset zigzag. Esta fórmula, apesar de possuir complexidade de tempo idêntica a do algoritmo da seção VII.3.4, é bem mais simples que este. Um poset ***zigzag*** é aquele cujo diagrama de Hasse como grafo subjacente é um caminho simples. Um exemplo deste poset é dado na figura VII.3.

O diagrama de Hasse de um poset zigzag pode ser construído adicionando-se, da esquerda para a direita, os vértices e arestas totalmente à direita dos vértices e arestas adicionados anteriormente. Seja $\mathcal{Z}$ um poset zigzag com n elementos $x_1, \ldots, x_n$, ordenados da esquerda para a direita segundo seu diagrama. Seja u_j o número de extensões lineares de um poset zigzag $\mathcal{P}$ com r elementos, nas quais a altura de x_r (elemento mais à direita de P) é dada por $h(x_r) = j$, $1 \leq j \leq r$. Se mantemos as quantidades $e(\mathcal{Q} \mid h(x_{n-1}) = j)$, $1 \leq j \leq n-1$, para o poset $\mathcal{Q} = \mathcal{Z} - \{x_n\}$ com $n-1$ elementos, podemos recalcular estas quantidades para $\mathcal{Z}$ como somas parciais das quantidades antigas, onde a soma é feita da esquerda para a direita ou da direita para a esquerda, dependendo se x_n foi inserido acima ou abaixo de x_{n-1}. Ou seja, o valor $e(\mathcal{Z} \mid h(x_n) = j) = \sum_{i=1}^{j-1} e(\mathcal{Q} \mid h(x_{n-1}) = i)$ ou $\sum_{i=j}^{n-1} e(\mathcal{Q} \mid h(x_{n-1}) = i)$. Para o poset zigzag da figura VII.3, obtemos a seguinte tabela, onde cada linha i corresponde

aos valores obtidos para o poset induzido pelos elementos $\{x_1, \ldots, x;)$.

$$
\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & & 0 & & 1 & & & & & \\
 & & & & 0 & & 0 & & 1 & & & & \\
 & & & 1 & & 1 & & 1 & & 0 & & & \\
 & & 0 & & 1 & & 2 & & 3 & & 3 & & \\
 & 9 & & 9 & & 8 & & 6 & & 3 & & 0 & \\
35 & & 26 & & 17 & & 9 & & 3 & & 0 & & 0
\end{array}
$$

A soma dos valores na última linha, igual a 90, nos dá o número de extensões do poset da figura VII.3.

Um poset cerca é um caso especial de um poset zigzag, onde $x_1 \prec x_2 \succ x_3 \prec x_4 \succ \ldots$. (Veja figura VII.l.) O número de extensões lineares de um poset cerca é contado através de números de Euler (veja Stanley [Sta86]), o que segue de uma observação trivial do método apresentado acima. Extensões de posets cerca correspondem à função inversa de permutações alternadas [Rusdo], consideradas no capítulo de geração por transposição (corolário 4).

VII.3.6 Posets de largura 2

Sejam $x_1, \ldots, x_p$ e $y_1, \ldots, y_q$ duas seqiiências de elementos distintos. Um *embaralhamento* destas duas seqüências é a seqüência $a_1, \ldots, a_{p+q}$, onde cada x_i e cada y_j ocorrem exatamente uma vez e as ordens relativas dos x; e dos y_j nas seqiiências originais são mantidas na seqüência final ($1 \leq$ i $\leq p$ e $1 \leq j \leq$ q). É sabido que o número desses embaralhamentos é dado por $\left(\begin{array}{c} p+q \\ p \end{array} \right)$.

Seja C uma coleção de pares (x_i, y_j) ou (y_r, x_s), $\mathbf{1} \leq i, s \leq p$ e $\mathbf{1} \leq j, r \leq \boldsymbol{q}$. Definimos um embaralhamento como C-restrito se u precede v neste embaralhamento toda vez que $(u, v) \in$ C. As restrições a serem satisfeitas podem ser representadas através de um conjunto parcialmente ordenado $\mathcal{P}$ sobre $\{x_1, \ldots, x_{,,} y_1, \ldots, y_q\}$, onde

1. $x_i \prec x_{i+1}$, $i = 1, \ldots, p-1$;
2. $y_j \prec y_{j+1}$, $j = 1, \ldots, q-1$;

3. $u \prec v$, se $(u, v) \in C$;

4. todas as conseqüências transitivas de **1,** 2 e **3,**

Assim, os embaralhameiitos C-restritos são exatamente as extensões lineares de $\mathcal{P}$. Além disso, a classe de posets definida acima é exatamente a classe dos posets de *largura* 2. Outros algoritmos polinomiais para enumeração desta classe aparecem na literatura : Linial [Lin84] apresenta um algoritmo O(n^3), assim como é a complexidade do algoritmo descrito na seção VII.3.7. Apresentaremos, agora um método O(n^2) (mais precisamente, $O(pq)$) para a enumeração de extensões de posets de largura 2, encontrado em [Atk89, AC87].

Um pré-processamento organiza os dados de entrada para o corpo principal do algoritmo. Neste pré-processamento, calculamos :

$$\begin{aligned} a_i &:= \min\{t \mid x_i \prec y_t\}, \text{ i} = 1, \ldots, p \\ b_i &:= \max\{t \mid y_t \prec x_i\}, \; i = 1, \ldots, p \end{aligned}$$

Por conveniência, y_0 e y_{q+1} são menor que, respectivamente, maior que qualquer outro elemento do poset. Portanto, a_i e b_i estão todos bem definidos. O tempo necessário ao pré-processamento depende de como os dados se apresentam. Uma matriz M, p x q, onde

$$t_{ij} = \begin{cases} 1 & \text{, se } x_i \prec y_j; \\ -1 & \text{, se } y_j \prec x_i; \\ 0 & \text{, caso contrário.} \end{cases}$$

é uma apresentação conveniente, pois possibilita se pré-processar os dados em tempo $O(p \log q)$ (as linhas da matriz são monotônicas e, portanto, poderíamos utilizar uma busca binária).

O corpo principal do algoritmo é como se segue :

procediineiito $Soma(u)$
  para $j := 1, \ldots,$ **q faça**
    $u_j := u_j + u_{j-1}$
fiin procedimento

$(u_0, \ldots, u_q) :=$ **(1**,0, O,. . . ,0)

para i $:= 1, \ldots, p$ **faça**
  $Soma(u)$ (1)
  $u_j := 0$, para todo $j < b_i$ e todo $j \geq$ a; (2)
retorne $\sum_i u_i$

Agora provamos a correção do algoritmo. Seja $\mathcal{P}_i$ o poset induzido em $\mathcal{P}$ pelos elementos $x_1, \ldots,$ x;, $y_1, \ldots, y_q$.

Proposição 1 Ao *final* da i-ésima iteração do loop principal do algoritmo, cada u_j nos dá o número de extensões lineares de $\mathcal{P}_i$ nas *quais* x_i se encontra entre y_j e y_{j+1}.

Prova : Usaremos indução em i, tomando como base o caso trivial quando i $= 1$. Suponhamos agora i > 1 e que, ao final da $(i-1)$-ésima iteração, cada u_j contém o número de extensões de $\mathcal{P}_{i-1}$ com x_{i-1} entre y_j e y_{j+1}. Construiremos $\mathcal{P}_i$, a partir de $\mathcal{P}_{i-1}$, em dois passos.

No primeiro passo, adicionamos a $\mathcal{P}_{i-1}$ o elemento x;, junto com a relação $x_{i-1} \prec x_i$ (bem como as demais relações implícitas por transitividade), para obter o poset $\mathcal{P}_i^*$. Para cada j, toda extensão de $\mathcal{P}_i^*$ na qual x; se encontra entre y_j e y_{j+1} é obtida inserindo-se x_i (nesta posição) em uma extensão de $\mathcal{P}_{i-1}$ onde x_{i-1} se encontra entre y_k e y_{k+1}, para algum $k \leq$ j (pois $x_{i-1} \prec x_i$). Assim, temos $u_j^* = \sum_{k \leqslant j} u_k$ extensões de $\mathcal{P}_i^*$, tais que x; se encontra entre y_j e y_{j+1}. Portanto, o passo **(1)** do algoritmo modifica o vetor u corretamente, de modo que ele contenha valores apropriados para $\mathcal{P}_i^*$.

O segundo passo corresponde ao passo (2) do algoritmo, que obviamente corresponde a se eliminar as extensões de $\mathcal{P}_i^*$ que não são compatíveis com as relações $y_{b_i} \prec$ x; e x; $\prec y_{a_i}$ e suas conseqüências transitivas. Concluído o passo (2) temos, portanto, os valores corretos de $u_0, \ldots, u_q$ para P;.

Ao fim do loop, $\mathcal{P}_p = \mathcal{P}$ e $u_0 + \ldots + u_q$ é o número de extensões lineares de $\mathcal{P}$. □

É bastante claro que o algoritmo tem complexidade $O(pq)$, uma vez que temos dois loops intercalados com p e q iterações respectivamente, e a rotina de inicialização pode ser feita em tempo O(plog q). O tempo de execução do algoritmo poderia ser consideravelmente reduzido em certos casos, pois o algoritmo gera vetores que podem

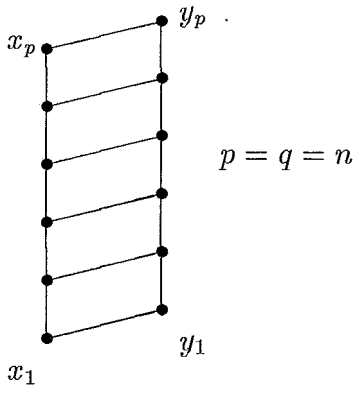


Figura VII.4:

começar e acabar com seqüências de zeros. Poderíamos, então restringir os somatórios aos segmentos não nulos desse vetores.

Alguns casos interessantes. O primeiro caso considerado é o poset cujo diagrama de Hasse é dado pela figura VII.4.

O número de extensões deste poset é dado pelo p-ésimo número de Catalão c_p. Os valores do vetor u são, por sua vez,

1
1 1
1 2 2
1 3 5 5
1 4 9 14 14
1 5 14 28 42 42
1 6 20 48 90 132 132
...

onde seqüências de zeros foram omitidas. O algoritmo produz cada linha desta tabela somando parcialmente a linha anterior e duplicando o último elemento. As somas das linhas (ou os elementos finais de cada linha) nos dão a seqüência dos números de Catalão. Uma prova informal deste fato pode ser encontrada em [AC87].

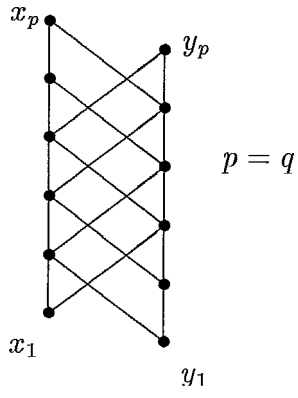


Figura VII.5:

Um outro caso interessante de posets de largura 2 pode ser visto na figura VII.5. O complemento deste poset é um poset cerca (veja figura VII.1), que é um caso especial de posets zigzag. Os valores do vetor u, como produzidos pelo algoritmo, são (omitindo-se zeros) :

1 1 1
2 3 3
5 8 8
13 21 21
...

Os valores não nulos r;, s_i, s_i na i-ésima coluna satisfazem

$$\begin{aligned} r_i &= r_{i-1} + s_{i-1}, \\ s_i &= r_{i-1} + 2s_{i-1} \end{aligned}$$

Segue, então, que $r_1, s_1, r_2, s_2, r_3, s_3, \ldots$ é a seqüência de Fibonacci $F_0, F_1, F_2, \ldots$ e que o número de extensões lineares do poset é dado por

$$r_p + s_p + s_p = F_{2p-2} + 2F_{2p-1} = F_{2p+1}.$$

Já o número de extensões lineares de um poset cerca é dado pelo número de Euler E_{2p}, como podemos ver na seção VII.3.5.

VII.3.7 Posets de largura k e posets com diâmetro de decomposição de largura k

Um primeiro algoritino polinomial para posets de largura k foi apresentado por Atkinson & Chang em [AC86]. Este algoritmo, de complexidade $O(n^{k^2})$, fazia uso de técnicas de programação dinâmica sobre uma decomposição em cadeias de um poset $\mathcal{P}$ de largura k. Mais recentemente, Steiner [cf. [Atk89]] desenvolveu um outro algoritmo de complexidade $O(n^{k+1})$ que também utiliza programação dinâmica, mas desta vez sobre o poset de ideais de ordem de $\mathcal{P}$.

Um *ideal de ordem* de um poset $\mathcal{P}(S, \mathrm{R})$ é um subconjunto $\mathbf{I} \subseteq S$ com a propriedade de que se p E **I** e q $\prec$ p, então q E **I.** Se $S' \subseteq \mathbf{S}$, então o ideal de ordem gerado por S' (denotamos $I(S')$) é o conjunto de todos os elementos y tais que y $\prec$ x, para algum x E S'. Um *filtro de ordem* de $\mathcal{P}$ é o complemento de um ideal.

A técnica de Steiner consiste em se usar a fórmula recursiva (II.1), válida para um poset qualquer. No caso geral, a aplicação desta fórmula é exponencial, dado que um número exponencial de subproblemas deve ser resolvido. A observação de que muitos destes subproblemas aparecem repetidas vezes durante o processo, nos leva a uma abordagem via programação dinâmica, onde estes subprobleinas são resolvidos uma primeira vez e depois têm sua solução guardada. Steiner usa um outro algoritmo, que também lhe é devido [Ste86], para gerar os ideais de ordem de um poset, levando $O(n)$ passos por ideal. Como em um poset de largura k o número de ideais de ordem é $O(n^k)$, estes podem ser gerados em tempo $O(n^{k+1})$. Deste modo, a aplicação repetida de **(11.1)**para se calcular $e(I)$, para todo ideal de ordem **I** de um poset $\mathcal{P}$ de largura k, também requer tempo $O(n^{k+1})$.

Seja uma decomposição por substituição (ver teorema 4) de um poset $\mathcal{P}$, tal que $\mathcal{P}$ é o poset obtido a partir da substituição dos vértices $a_1, \ldots,$ a, de um poset $\mathcal{Q}$ por posets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r$, respectivamente, onde $\mathcal{Q}, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r$ são posets disjuntos não vazios e menores em cardinalidade que $\mathcal{P}$. O poset $\mathcal{Q}$ é chamado de *fator externo* da decomposição. Se um poset pode ser construído por decomposição por substituições sucessivas na qual os fatores externos têm largura no máximo k, então o poset é dito ter *diâmetro de decomposição de largura* no máximo k. A abordagem de Steiner para posets de largura k, usando programação dinâmica, se estende à classe dos posets com diâmetro de decomposição de largura L, calculando $e(\mathcal{P})$ em tempo $O(n^{k+2})$ [cf.

[Atk89]].

VII.3.8 Posets com diâmetro de decomposição limitado

Esta classe de posets aparece em Habib & Mohring [HM87] como uma proposta de generalização da classe dos posets série-paralelos, que os autores consideram mais "coerente" (devido a uma série de propriedades características das duas classes) do que a generalização que toma a classe dos posets livres de **N** (veja [HM87]).

Seja $\wp_k$ a classe de posets iterativamente construída por substituição de posets primos de cardinalidade no máximo k. Por esta definição, é claro que $\wp_2 = \wp_3$ é a classe dos posets série-paralelos, uma vez que as cadeia e anticadeia de 2 elementos são os únicos posets primos de cardinalidade no máximo **3.** A partir de qualquer algoritmo de reconhecimento de posets decomponíveis, podemos construir um outro algoritmo que reconhece, com mesma complexiclade, os posets de $\wp_k$, para qualquer inteiro fixo k. Desta forma, se tomamos o melhor algoritmo conhecido, aquele de Muller & Spinrad [MS84], temos um algoritmo $O(n^2)$ para reconhecer esta classe. Para $k \leq 3$ (posets série-paralelos), existe um algoritmo linear de reconhecimento [VTL79]. Não se sabe se isto é válido também para qualquer inteiro fixo. É válido ressaltar que $\wp_k$ é uma classe hereditária [HM87].

Proposição 2 *Existe um algoritmo de tempo polinomial para calcular o número de extensões lineares de um poset $\mathcal{P}$ em $\wp_k$.*

Prova : Seja $\mathcal{P}(S, R)$ um poset de diâmetro de decomposição 5. A prova procede por indução em $|S| = n$. Se $\mathcal{P}$ é primo, então $|\mathcal{P}| \leq 5$ e podemos calcular *e(?)* usando um algoritmo de enumeração para o caso geral de posets, como por exemplo o algoritmo de Pruesse & Ruskey [PR91] descrito na seção VII.2. Caso contrário, usando o teorema de decomposição (teorema 4), sabemos que existe um único poset $\mathcal{Q}$ tal que

$$\mathcal{P} = \mathcal{Q}^{\mathcal{P}_1,\ldots,\mathcal{P}_h}_{a_1,\ldots,a_h}, \text{ onde } 1 < |\mathcal{P}_i| = n_i < |\mathcal{P}|.$$

(a) $\mathcal{Q}$ é uma ordem total (cadeia) Então, o problema está resolvido, com a hipótese de indução, uma vez que

$$e(\mathcal{P}) = \prod_{1 \leq i \leq h} e(\mathcal{P}_i) \; ;$$

(b) $\mathcal{Q}$ é uma anticadeia. Novamente, estamos resolvidos com a hipótese de indução, já que temos a fórmula

$$e(\mathcal{P}) = \prod_{1 \leq i \leq h} e(\mathcal{P}_i) \frac{(n_1 + \ldots + n_h)!}{n_1! \ldots n_h!};$$

(c) $\mathcal{Q}$ é primo. Então, necessariamente $|\mathcal{Q}| \leq$ L. Um argumento fácil (veja [HM87]) nos dá a seguinte fórmula

$$e(\mathcal{P}) = \left(\prod_{1 \leq i \leq h} e(\mathcal{P}_i) \right) e(\mathcal{Q}^{L_1,\ldots,L_h}_{a_1,\ldots,a_h}),$$

onde cada L; é uma extensão linear qualquer de $\mathcal{P}_i$.

Definamos $\mathcal{Q}' = \mathcal{Q}^{L_1,\ldots,L_h}_{a_1,\ldots,a_h}$. Só falta verificar que o cálculo do número de extensões de $\mathcal{Q}'$ pode ser feito em tempo polinomial. Para tal, vamos assumir que conhecemos $E(\mathcal{Q})$, dado que $|\mathcal{Q}| \leq$ L. Para cada $\mathbb{E} \in E(\mathcal{Q})$, associamos um subconjunto de $E(\mathcal{Q}')$, que denotaremos por $A(l)$, como se segue :

Para cada a;, $1 \leq$ i $\leq h$, seja $I_l(a_i)$ o intervalo $]a_i, \beta_i[$ de l tal que, para todo x $\in I_l(a_i)$, x $\|$ a; e I_l(a;) é maximal com respeito a esta propriedade. Por conveniência, denotaremos as extensões lineares L; por L; $= a_i(1), \ldots, a_i(n_i)$ (lembrando que $|L_i| = |\mathcal{P}_i| = n_i$) e definimos $X = \{a_1, \ldots, a_h\}$.

Para se obter uma extensão $\alpha \in A(l)$, substituímos cada vértice a; de $\mathbb{E}$ pelo primeiro elemento $a_i(1)$ de L_i, e inserimos os elementos restantes de L_i "entre" os elementos de $I_l(a_i)$ em l. Ou seja, fazemos um embaralhamento - ver seção VII.3.6 - destes dois conjuntos, inserindo os elementos restantes de L; de todas as maneiras possíveis na extensão α sendo construída, entre as posições ocupadas por $a_i(1)$ e β_i (ou o elemento pelo qual β_i foi trocado, se $\beta_i = a_j$, para algum **j).** Devemos agora provar que os conjuntos $A(l)$ formam uma partição de $E(\mathcal{Q}')$.

Suponhamos que $\alpha \in A(l) \cap A(l')$, onde l e l' são duas extensões lineares distintas de $\mathcal{Q}$. Então, existe ao menos um par a, b $\in \mathcal{Q}$ tal que a $\prec_l$ b, b $\prec_{l'}$ a e a $\|$ b em $\mathcal{Q}$.

- Quando a, b $\notin$ **X,** então temos necessariamente a $\prec_\alpha$ b e b $\prec_\alpha$ a, uma contradição;

  quando a $=$ a; e b $\notin$ **X,** temos que $a_i(1) \prec_\alpha$ b e b $\prec_\alpha a_i(1)$, também uma contradição;

- quando a, b E X, um argumento similar ao anterior leva mais uma vez à contradição.

Indo mais além, para qualquer extensão linear a E $E(\mathcal{Q}')$, podemos obter sua restrição I em $\mathcal{Q}$ removendo-se todos os elementos $a_i(j)$, $1 \leq$ i $\leq$ h e $1 <$ $j \leq$ n;, e trocando os elementos $a_i(1)$ por a;, $1 \leq$ i $\leq$ h. Assim, a E $A(l)$. Concluímos que os conjuntos $A(l)$ de fato particionam $E(\mathcal{Q}')$ e que $e(\mathcal{Q}') = \sum_{l \in E(\mathcal{Q})} |A(l)|$. Como $|\mathcal{Q}| \leq k$, existem no máximo $k!$ extensões lineares de $\mathcal{Q}$.

Agora só nos resta provas que, para cada extensão linear I, $|A(l)|$ é polinomialmente computável. Infelizmente esta prova não é em nada trivial, e achamos por bem não nos deter com ela. Esta parte da prova, assim como todos os resultados mencionados para esta classe de posets, se encontra na íntegra em [HM87] e nos diz que podemos obter $|A(l)|$ através da seguinte fórmula :

$$|A(l)| = \sum_{x^1} \cdots \sum_{x^h} \prod_{j=1}^{r} N(x_j^1, \ldots, x_j^h),$$

onde

$$N(x_j^1, \ldots, x_j^h) = \frac{(\sum_{i=1}^{h} x_j^i)!}{\sum_{i=1}^{h} (x_j^i)!},$$

para r $\leq k$ vetores de partição fixos e bem definidos. Segue também que existem no máximo $|L_i|^r \leq n^r \leq n^k$ vetores de partição x^i e, portanto, no máximo $(n^k)^h \leq$ n^{k_2} termos na soma. Cada termo involve no máximo $2k^2$ números $x_j^i \leq$ n. Deduzimos, finalmente, que o cálculo de $e(\mathcal{P})$ para posets $\mathcal{P}$ na classe $\wp_k$ necessita de no máximo $n^{k_2} \cdot 2k^2 = 0(n^{k_2})$ operações aritméticas. O

VII.3.9 Posets com ciclos disjuntos em arestas

As idéias contidas no trabalho de Atkinson & Chang em [Atk85], que incluem um algoritmo $O(n^5)$ para enumeração de extensões de árvores não-enraizadas, foram estendidas por Chang [Cha86] para uma classe mais abrangente de posets, que compreende todo poset tal que todos os ciclos no seu diagrama de Hasse (caminho não orientado no grafo subjacente que retorna a seu ponto de origem, sem passar duas vezes por um mesmo vértice) são *disjuntos* em arestas. Para posets desta classe, usamos a mesma abordagem que Atkinson & Chang para árvores não-enraizadas : as

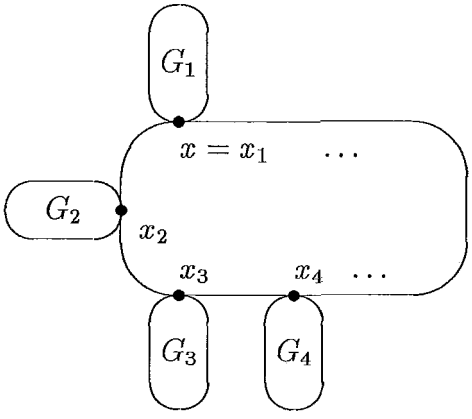


Figura VII.6:

quantidades e(P | $h(x) = i$) devem ser calculadas. A diferença é que a decomposição destes grafos é bem mais complicada, uma vez que duas componentes podem agora estar ligadas por mais de uma aresta e, desta forma, resultados mais gerais que aquele dado pelo teorema 19 se mostram necessários.

Para calcular e(P | $h(x) = i$), Chang observa que se x pertence a algum ciclo, o grafo formado somente pelas relações de cobertura de $\mathcal{P}$ (*grafo* de cobertura de **P**) tem a forma mostrada na figura VII.6. Se x não pertence a nenhum ciclo, então o grafo de cobertura tem a forma apresentada na figura VII.7.

Em qualquer dos casos, os grafos $G_1, G_2, \ldots$ também têm uma destas duas formas. O caso da figura VII.7 é mais fácil de se considerar. Podemos assumir que todos G_i já foram considerados e que as quantidades $e(G_i \mid h(y_i) = j)$ foram calculadas, para y; E G_i tal que x $\prec$ y_i ou $y_i \prec$ x. Os valores de e(P | $h(x) = i$) podem agora ser calculados aplicando-se o teorema 19 iterativamente.

No primeiro caso (figura VII.6), podemos novamente assumir que $e(G_i \mid h(y_i) = j)$ já foram calculadas. O cálculo de e(P | $h(x) = i$) requer neste caso um lema de Atkinson & Chang [Atk85], que mostra como contar as extensões lineares de um

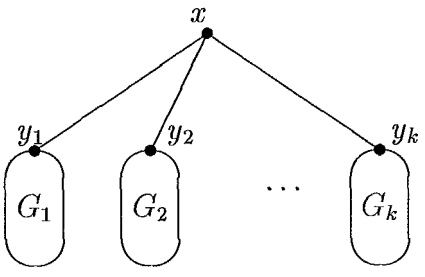


Figura VII.7:

poset obtido ao se unir dois posets disjuntos por meio de uma aresta, controlando-se as alturas de dois elementos em uma das componentes para tal. Na realidade, o teorema 19 é um caso particular deste lema. A aplicação do lema mencionado ao problema não é imediata e, portanto, não nos deteremos aqui com sua apresentação.

O algoritmo desenvolvido por Chang para posets com ciclos disjuntos em arestas calcula $e(\mathcal{P})$ em tempo $O(n^8)$.

VII.3.10 Orientações de grafos bipartite

Consideraremos uma variação do problema de enumeração analisado no decorrer deste capítulo. Ao invés de se tentar contar as extensões de uma dado poset $\mathcal{P}$, a questão a ser considerada agora consiste em, dado um grafo qualquer G, determinar uma orientação das arestas de G que nos dê o maior número de extensões lineares para o poset definido pelo fecho transitivo de G orientado.

Veremos que, no caso de grafos bipartite, esta orientação é a orientação natural, como havia sido conjecturado por Atkinson em [Atk89] e como foi provado por Stachowiak em [Sta88]. Ainda em [Atk89], este mesmo resultado foi provado para o caso do grafo ser uma árvore não orientada, mas esta classe se encontra contida na classe dos grafos bipartite.

Durante o decorrer desta seção, estaremos interessados apenas nas *orientações acíclicas* (não formam circuitos) de um grafo G, uma vez que uma orientação cíclica

em G não definiria um poset. Se $G(V, \mathrm{E})$ é um grafo bipartite, então existe uma partição $\mathrm{V} = V_1 \cup V_2$, tal que não existe aresta entre nenhum par de vértices em V_1 nem entre um par em V_2. Uma *orientação natural* de um grafo bipartite G é aquela onde toda aresta é direcionada de V_1 para V_2 (ou vice-versa). Todo grafo bipartite possui ao menos uma bipartição que respeita as condições acima, possuindo ao menos duas orientações naturais. Fica a cargo do leitor a verificação (trivial) de que todas as orientações naturais de um dado grafo bipartite geram posets com mesmo número de extensões lineares.

Teorema 24 *Qualquer orientação acíclica de u m grafo bipartite G possui u m número menor de extensões lineares que uma orientação natural.*

Antes de provar este teorema, mostramos o seguinte lema (lembrando que $Min(\mathcal{P})$ e $Max(\mathcal{P})$ denotam os conjuntos de elementos ininimais e maximais de um poset $\mathcal{P}$, respectivamente) :

Lema 15 *Se $\mathcal{P}$ não contém nenhum elemento incomparável a todos os outros elementos do poset, então*

$$2\,e(\mathcal{P}) = \sum_{x \in Min(\mathcal{P}) \cup Max(\mathcal{P})} e(\mathcal{P} - \{x\}).$$

Prova : Seja n o número de elementos de P. Temos que e(P|$h(x) = 1$) = e(P−{x}), para todo *x* E $Min(\mathcal{P})$. Portanto,

$$e(\mathcal{P}) = \sum_{x \in Min(\mathcal{P})} e(\mathcal{P} \mid h(x) = 1) = \sum_{x \in Min(\mathcal{P})} e(\mathcal{P} - \{x\}).$$

Analogamente,

$$e(\mathcal{P}) = \sum_{x \in Max(\mathcal{P})} e(\mathcal{P} \mid h(x) = n) = \sum_{x \in Max(\mathcal{P})} e(\mathcal{P} - \{x\})$$

Se $\mathcal{P}$ não contém elementos isolados, então os conjuntos $Min(\mathcal{P})$ e $Max(\mathcal{P})$ são disjuntos. Assim,

$$2\,e(\mathcal{P}) = \mathrm{C}_{x \in Min(\mathcal{P}) \cup Max(\mathcal{P})} e(\mathcal{P} \_ \{x\}).$$

□

Prova do teorema : Procedemos por indução em n = $|V|$. Se G é o grafo trivial (possui um único vértice), então o teorema é óbvio. Assumindo que a hipótese de indução é válida, vamos mostrar que continua válida para um grafo bipartite G com n vértices. Dois casos são possíveis :

(i) G possui um vértice isolado x. Temos então, para qualquer orientação acíclica $\mathcal{P}$ de G,

$$e(\mathcal{P}) = \sum_{i=1}^{n} \mathrm{e(P} \mid h(x) = \mathrm{i)} = \mathrm{n\, e(P} - \{\mathrm{x}\}).$$

É fácil ver que **P** é uma orientação natural de G se e somente se $\mathcal{P} - \{\mathrm{x}\}$ é uma orientação natural de G $-\{\mathrm{x}\}$, e o teorema segue pela hipótese de indução.

(ii) Todo vértice de G é adjacente a um outro vértice de G. Pelo lema 15, para uma orientação acíclica arbitrária **P** de G, temos que

$$2\, e(\mathcal{P}) = \sum_{x \in Min(\mathcal{P}) \cup Max(\mathcal{P})} e(\mathcal{P} - \{x\}).$$

Se $\mathcal{P}$ é uma orientação natural de G, então o somatório se dá sobre todos os elementos x de $\mathcal{P}$, e **P** $-\{\mathrm{x}\}$ é uma orientação natural de G $-\{\mathrm{x}\}$, para todo x. Neste caso, a soma é feita sobre todos x E $\mathcal{P}$ e cada termo e(P $-\{\mathrm{x}\}$) é máximo com relação às orientações de G $-\{\mathrm{x}\}$, pela hipótese de indução.

Por outro lado, se $\mathcal{P}$ não é uma orientação natural de G, então pelo menos um elemento y de $\mathcal{P}$ não é maximal nem minimal. Deste modo, o resultado da soma, sem o termo e(P $-\{\mathrm{y}\}$), é certamente estritamente menor que no caso de $\mathcal{P}$ ser uma orientação natural de G. o

Usando o teorema 24, podemos responder parcialmente a uma questão proposta por Atkinson em [Atk89]. Seja $\mathcal{P}(S, \mathrm{R})$ um poset e $G_C(\mathcal{P})$ seu grafo de comparabilidade. É bem sabido que o número de extensões de $\mathcal{P}$ é invariante dentre todas as possíveis orientações acíclicas transitivas de $G_C(\mathcal{P})$ (cf. [Sta86]).

Corolário 9 *Seja $\mathcal{P}$ um poset bipartite, i. e., $\mathcal{P}$ é uma orientação natural de $G_C(\mathcal{P})$. Seja $\mathcal{Q}$ um outro poset sobre o mesmo conjunto de elementos que* **P**. *Se* $G_C(\mathcal{P}) \subseteq$

$G_C(\mathcal{Q})$, então $e(\mathcal{P}) \geq$ e(&). Mais ainda, $e(\mathcal{P}) = e(\mathcal{Q})$ se e somente se $G_C(\mathcal{P}) = G_C(\mathcal{Q})$.

Prova: Suponhamos $G_C(\mathcal{P}) \subseteq G_C(\mathcal{Q})$. Orientemos as arestas de $G_C(\mathcal{P})$ como estas estão orientadas no fecho transitivo de $\mathcal{Q}$ (= $G_C(\mathcal{Q})$ orientado) e chamemos de R ao poset resultante. Notemos que $G_C(\mathcal{P}) \subseteq G_C(\mathcal{R}) \subseteq G_C(\mathcal{Q})$.

O poset $\mathcal{Q}$ tem necessariamente todas as relações de R. Portanto, $e(\mathcal{R}) \geq e(\mathcal{Q})$ e $e(\mathcal{R}) = e(\mathcal{Q})$ se e somente se $\mathrm{R} = \mathcal{Q}$. Os posets R e $\mathcal{P}$ são ambos orientações de $G_C(\mathcal{P})$, sendo $\mathcal{P}$ uma orientação natural. Pelo teorema 24, temos que (sendo $G_C(\mathcal{P})$ um grafo bipartite) $e(\mathcal{P}) \geq e(\mathcal{R})$ e $e(\mathcal{P}) = e(\mathcal{R})$ se e somente se $G_C(\mathcal{P}) = G_C(\mathcal{R})$. Então, $e(\mathcal{P}) \geq$ e(&), e $e(\mathcal{P}) = e(\mathcal{Q})$ se e somente se $G_C(\mathcal{P}) = G_C(\mathcal{Q})$. □

O problema quanto a $G_C(\mathrm{P}) \subseteq G_C(\mathcal{Q})$ implicar em $e(\mathcal{P}) \geq e(\mathcal{Q})$ para posets $\mathcal{P}$ e $\mathcal{Q}$ arbitrários continua em aberto.

Capítulo VIII

Conclusões

O problema de enumeração de extensões lineares foi recentemente provado sei #P-completo, como visto no capítulo VII. Portanto é aparentemente muito difícil, no sentido que provavelmente não conseguiremos contar as extensões de um poset arbitrário $\mathcal{P}$ em tempo polinomial no seu número de elementos. De fato, o algoritmo mais eficiente conhecido que conta as extensões de um poset $\mathcal{P}$ é aquele considerado na seção VII.2 de complexidade $O(e(\mathcal{P}\backslash\{a_1, b_1\}))$, onde a_1 e b_1 são dois elementos minimais de $\mathcal{P}$. Este algoritmo é uma adaptação trivial do algoritmo de Pruesse & Ruskey (capítulo VI) que resolve de maneira ótima o problema da geração de extensões lineares.

Para algumas classes particulares de posets conseguimos algoritmos polinomiais que contam o número de extensões de posets dentro destas classes, como visto no capítulo VII para posets floresta, multiconjuntos, série-paralelos, de largura k, etc. Certamente existem várias outras classes de posets para as quais podemos achar $e(\mathcal{P})$ em tempo polinomial. Uma destas classes parece ser a classe dos posets semi-ordem. Um poset semi-ordem $\mathcal{P}(S, \mathrm{R})$ pode ser sempre modelado da seguinte forma : dado um conjunto S de intervalos unitários em uma reta r, x $\prec_R$ y, x, y E S, se e somente se x se encontra totalmente à esquerda de y em r. O grafo de comparabilidade de um poset semi-ordem é o complemento de um grafo de indiferença (ver [Rob69, Tro85]). E os posets livres de **N** [HM87] ? Estes são uma relaxação dos posets série-paralelo quanto à composição em série admitida. Talvez seja possível se adaptar o algoritmo linear para enumeração de extensões de posets série-paralelos para os posets livres de N.

Outra questão a ser levantada é sobre a possibilidade de se construir um algoritmo de tempo médio constante para geração por transposição das extensões de um poset, toda vez que o mesmo admitisse tal geração. E o caso da geração por inserção ? Será que este tipo de geração também admite um algoritmo de tempo médio constante ?

O algoritmo de Pruesse & Ruskey constitui o primeiro algoritmo de geração de tempo médio constante conhecido para um problema combinatório #P-completo. Chegamos então a uma questão interessante acerca da complexidade de problemas de geração correspondentes a problemas de enumeração na classe $\#P$: será que todos os problemas #P-completos admitem uma geração de tempo médio constante ? Infelizmente isto parece pouco provável, mas não chegaria a implicar diretamente em $\mathbf{P} = \mathrm{NP}$.

Consideramos um algoritmo de tempo médio constante como sendo aquele que, dada uma solução inicial apropriada para um certo problema combinatório, gera todas as outras soluções para o problema em tempo médio constante. O pré-processamento que nos leva à solução inicial foi fácil no caso de extensões lineares ($O(n^2)$, chegando a ser $O(n)$ no caso de permutações irrestritas). Já no caso de ciclos hamiltonianos, o problema de enumeração dos quais também é #P-completo, este pré-processamento é claramente NP-completo, pois não se conhece um método eficiente (polinomial) que determine um ciclo hamiltoniano em um grafo (mesmo se sabemos a priori que o grafo é hamiltoniano). Será que a existência de um algoritmo de tempo médio constante para a geração de ciclos hamiltonianos implicaria em $\mathbf{P} = \mathrm{NP}$? A solução desta questão certamente ajudaria na determinação da hierarquia dos problemas de geração dentro da classe **#P.**

Serão os níveis da hierarquia dos problemas de geração dentro da classe dos problemas #P-completo delineados pelas complexidades dos problemas de decisão correspondentes ? Extensões lineares são soluções para problemas de decisão onde a resposta é sempre afirmativa (pelo teorema **1,** de Szpilrajn), admitindo algoritmo de geração de tempo médio constante. Admitiriam os ideais de ordem de um poset ou ainda as cliques maximais de um grafo, para os quais o problema de decisão também é trivial, uma geração de tempo médio constante ? Verificamos então a existência de vários problemas interessantes ainda em aberto envolvendo a hierarquia de complexidade de geração dentro da classe **#P,** que constituem um desafio estimulante.

Bibliografia

[AC86] M.D. Atkinson and H.W. Chang. Extensions of partial orders of bounded width. Congressus Numerantium, 52:21–35, 1986.

[AC87] M.D. Atkinson and H.W. Chang. Computing the number of mergings with constraints. Inf. Proc. Letters, 24:289–292, 1987.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. The Design and Analysis of *Computer* Algorithms. Addison-Wesley, Reading, 1974.

[Aig79] M. Aigner. *Combinatorial* Theory. Springer Verlag, 1979.

[Atk85] M.D. Atkinson. Partial orders and comparison problems. *Congressus* Numerantium, 47:77–88, 1985.

[Atk89] M.D. Atkinson. The complexity of orders. Em I. Rival, editor, Algorithms and Order, págs. 195–230. Kluwer Academic, 1989.

[Atk90] M.D. Atkinson. Computing the number of linear extensions of a tree. Order, 7:23–25, 1990.

[BM83] H. Buer and R.H. Mohring. A fast algorithm for the decomposition of graphs and posets. Math. Oper. Res., 8:170–184, 1983.

[BR90] B. Bauslaugh and F. Ruskey. Generating alternating permutations lexicographically. Bit, 30:17–26, 1990.

[BW84] M. Buck and Wiedemann. Gray code with restricted density. *Discr. Math.*, 48:163–171, 1984.

[BW90] G. Brightwell and P. Winkler. Counting linear extensions is #P-complete. Relatório Técnico TR 90-49, DIMACS, 1990.

[Cha86] H.W. Chang. Linear extensions of partially ordered sets. Tese de mestrado, Carleton Un., Ottawa, Canada, 1986.

[Che80] M. Chein, M. Habib. The jump number of dags and posets : an introduction. Discr. Appl. Math., 9:189–194, 1980.

[DF] M. Dyer and A. Frieze. Computing the volume of convex bodies : a case where ranclomness provably helps. a ser publicado.

[DFR89] M. Dyer, A. Frieze, and Kannan R. A random polynomial time algorithm for estimating volumes of convex bodies. Em Proc. *21st* ACM Symposium on the Theory *of* Computing, págs. 375–381, 1989.

[Dil51] R.P. Dilworth. A deomposition theorem for partially ordered sets. Annals *of* Math., 51:161–165, 1951.

[Duf65] R.J. Duffin. Topology of series-parallel networks. Math. Annalysis Appl., 10:303–318, 1965.

[EHR84] P. Eades, M. Hickey, and R.C. Read. Some hamilton paths and a minimal change algorithm. **J.** ACM, 31:19–29, 1984.

[FG86] P.C. Fishburn and W.V. Gehrlein. Minimizing bumps in linear extensions of ordered sets. *Order*, 1:3–14, 1986.

[Fle74] H. Fleischner. The square of every two-connected graph is Hamiltonian. **J.** *Combin.* Theory *(B)*, 16:29–34, 1974.

[GJ79] M.R. Garey and D.S. Johnson. Computers and Intractability : A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, USA, 1979.

[HM79] M. Habib and M.C. Maurier. On the X-join decomposition for undirected graphs. Discrete Appl. Math., 1:201–206, 1979.

[HM87] M. Habib and R.H. Mohring. On some complexity properties of n-free posets and posets with bounded decomposition diameter. Discrete Math., 63:157–182, 1987.

[HS76] E. Horowitz and S. Sahni. Fundamentals *of* Data Structures. Comp. Science Press, Woodland Hills, 1976.

[Joh63] S.M. Johnson. Generation of permutations by adjacent transposition. Math. Comp., 17:282–285, 1963.

[JW90] B. Jackson and N.C. Wormald. k-walks of graphs. *Australasian* Journal *of Combinatorics*, 2:135–146, 1990.

[Kah62] A.B. Kahn. Topological sorting of large networlts. *Comm. ACM*, 5:558–562, 1962.

[Khado] L. Khachiyan. Complexity of polytope volume computation. Em J. Pach, editor, Recent Progress in Discrete *Computational* Geometry. Springer-Verlag, a ser publicado.

[Knu68] D.E. Knuth. The Art *of Computer* Programming - *vol.* ***3.*** Addison-Wesley, Reading, 1968.

[Knu79] D.E. Knuth. Lexicographic permutations with restrictions. Discr. Appl. *Math.*, 1:117–125, 1979.

[KR88] C.W. Ko and F. Ruskey. Solution of multidimensional lattice path parity difference recurrence relations. Discr. *Math.*, 71:47–56, 1988.

[KS74] D.E. Knuth and J.L. Szwarcfiter. A structured program to generate all topological sorting arrangements. *Inf.* Proc. Letters, 2:153–157, 1974.

[KS84] J. Kahn and M. Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984.

[KT82] D. Kelly and W.T. Trotter. Dimension for ordered sets. Em I. Rival, editor, *Ordered* Sets, págs. 171–211. Reidel, 1982.

[KTdo] H. Kierstead and W.T. Trotter. The number of depth-first searches of an ordered set. Order, a ser publicado.

[KV83] A.D. Kalvin and Y.L. Varol. On the generation of all topological sortings. *J.* Algorithms, 4:150–162, 1983.

[Law78] E. Lawler. Sequencing jobs to ininimize total weighted completion time subject to precedence constraints. Ann. *Discrete* Math., págs. 78–90, 1978.

[Lin84] N. Linial. The information theoretic bound is good for merging. SIAM J. Comp., 13:795–801, 1984.

[Lin86] N. Linial. Hard enumeration probleins in geometry and combinatorics. SIAM J. Alg. Discr. Math., 7(2):331–335, 1986.

[Lov86] L. Lovász. An Algorithmic Theory of Numbers, Graphs and *Convexity.* SIAM, Philadelphia, 1986.

[LRKB77] J.K. Lenstra, A.H.G. Rinooy Kan, and P. Brucker. Complexity of machine scheduling problems. Ann. of Discrete Math., 1:343–362, 1977.

[MR84] R.H. Mohring and F.J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. Ann. Discrete Math., 19:257–356, 1984.

[MS84] J.H. Muller and J. Spinrad. On-line modular decomposition. Relatório Técnico GIT-ICS-84/11, Georgia Inst. of Tech., Atlanta, USA, 1984.

[NW75] A. Nijenhius and S. Wilf. *Combinatorial* Algorithms. Academic Press, New York, 1975.

[PB83] S. Provan and M.O. Ball. On the complexity of counting cuts and computing the probability that a grph is connected. SIAM *J. Comp.*, 12:777–788, 1983.

[PR88] G. Pruesse and F. Ruskey. Transposition generation of the linear extensions of certain posets. Relatório Técnico DCS-102-IR, Un. of Victoria, Canada, 1988.

[PR91] G. Pruesse and F. Ruskey. Generating linear extensions fast. Relatório Técnico DCS-159-IR, Un. of Victoria, Canada, 1991.

[Pru90] G. Pruesse. A generalization of hamiltonicity. Relatório Técnico 236/90, U. Toronto, 1990.

[Riv83] I. Rival. Optimal linear extensions by interchanging chains. Proc. *American* Math. Society, 89:387–394, 1983.

[RND77] E.M. Reignold, J. Nievergelt, and N. Deo. *Combinatorial* Algorithms : Theory and Practice. Prentice-Hall, Englewood Cliffs, 1977.

[Rob69] F.S. Roberts. Indifference graphs. Em F. Harary, editor, *Proof* Techniques in Graph Theory, págs. 139–146. Academic Press, New York, 1969.

[Rus87] F. Rusltey. Solution of some lattice path parity difference recurrence relations using involutions. Congressus Numerantium, 59:257–266, 1987.

[Rus88a] F. Ruskey. Generating linear extensions of posets by transpositions. Technical report, Un. of Victoria, Canada, 1988.

[Rus88b] F. Rusltey. Research problem 90. *Discr.* Math., 70:111–112, 1988.

[Rusdo] F. Ruskey. Transposition generation of alternating permutations. Order, a ser publicado.

[Sed77] R. Sedgewick. Permutation generation methods. ACM *Comp. Surveys*, 9(2):137–164, 1977.

[Sek60] M. Sekanina. On an ordering of the set of vertices of a connected graph. *Publ.* Fac. *Sc.* Brno., 412:137–141, 1960.

[SK87] J. Sha and D.J. Kleitman. The number of linear extensions of subset ordering. Discrete Math., 63:271–278, 1987.

[Sta86] R.P. Stanley. *Enumerative Combinatorics*, volume I. Wadsworth, 1986.

[Sta88] G. Stachowiak. The number of linear extensions of bipartite graphs. *Order*, 5:257–259, 1988.

[Ste64] H. Steinhaus. One Hundred *Problems* in Elementary Mathmatics. Basic, 1964.

[Ste83] G. Steiner. On a decomposition algorithm for sequencing problems with precedence constraints. Relatório Técnico 213, Faculty of Business, McMaster Un. Hamilton, Canada, 1983.

[Ste86] G. Steiner. An algorithm to generate the ideals of a partia1 order. Oper. Res. Letters, 5:317–320, 1986.

[Stedo] G. Steiner. On counting constrained depth-first linear extensions of ordered sets. Congressus Numerantium, a ser publicado.

[SW78] J.L. Szwarcfiter and L.B. Wilson. Some properties of ternary trees. Comput. **J.**,21:66–72, 1978.

[Sys85] M.M. Syslo. A graph theoretic approach to the jump number problem. Em I. Rival, editor, Graphs and Order, págs. 185–215. Reidel, Dordrecht, 1985.

[Szp30] E. Szpilrajn. Sur l'extension de l'ordre partiel. *Fund.* Math., 16:386–389, 1930.

[Szw83] J.L. Szwarcfiter. Grafos e Algoritmos Computacionais. Campus, Rio de Janeiro, 1983.

[Tar74] R. Tarjan. Finding dominators in directed graphs. *SIAM* **J.** *Comp.*, 3:62–89, 1974.

[TNS82] K. Taltamiza, T. Nishiltei, and N. Saito. Linear time computability of combinatorial problems on series-parallel graphs. **J.** ACM, 29(3):623–641, 1982.

[Tod89] S. Toda. On the computational power of PP and +P. Em Proc. 30th *IEEE* Symposium on Foundations *of Computer* Science, págs. 514–519, 1989.

[Tro62] H.F. Trotter. Algorithm 115 : Perm. Cornrn. ACM, 5:434–435, 1962.

[Tro83] W.T. Trotter. Graphs and partially ordered sets. Em L.W. Beinelte, R.J. Wilson, and editors, editores, Selected Topics in Graph Theory, págs. 237–268. Academic Press, London, 1983.

[Tro85] W.T. Trotter. Interval graphs, interval orders and their generalizations. Relatório Técnico AZ 85287, Arizona State Un., Tempe, USA, 1985.

[Val78] J. Valdes. Parsing flowcharts and series-parallel graphs. Relatório Técnico STAN-CS-78-682, Computer Science Department, Stanford Un., Stanford, USA, 1978.

[Val79a] L.G. Valiant. The complexity of computing the permanent. Theoret. Comp. *Sci.*, 8:189–201, 1979.

[Val79b] L.G. Valiant. The complexity of enumeration and reliability problems. SIAM J. Comp., 8:410–421, 1979.

[VR81] Y.L. Varol and D. Rotem. An algorithm to generate all topological sorting arrangements. Comput. J., 24:83–84, 1981.

[VTL79] J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series parallel digraphs. ACM, 1979.

[Wel71] M.B. Wells. Elements of *Combinatorial* Computing. Pergamon Press, Elmsford, 1971.

[Wil89] H.S. Wilf. *Combinatorial* Algorithms, An Update. SIAM, 1989.

[Wir76] N. Wirth. Algorithms + Data Structures = Programs. Prentice-Hall, Englewood Cliffs, 1976.

[Zag88] N. Zaguia. Minimizing bumps for posets of width two. Discr. Appl. *Math.*, 21:157–162, 1988.

Anexo A

Neste apêndice mostramos uma implementação em Pascal (Turbo Pascal6.0, da Borland) do algoritmo de tempo médio constante para geração das extensões lineares de posets série-paralelos descrito no capítulo IV, seguida de um exemplo de sua execução para o poset da figura 1.5 e que aparece com seus elementos numerados segundo o algositmo na figura A.1. E válido notar que listamos cada extensão gerada, ao invés de simplesmente indicar as operações de inserção efetuadas para se geras esta extensão a partir da sua predecessora na geração. Fazemos isto por motivos de clareza quanto ao exemplo mostrado. Para manter a complexidade linear no número de extensões do algoritmo, deveríamos simplesmente especificar as operações de inserção e concatenação executadas pelo algoritmo.

O Programa :

```
Program GeraSP;
const
  NMax = 200;

type
  tipoelmto     = I..NMax;
  tipocomp      = char;
  noarvptr      = ^noarv;
  noarv         = record
                    tipono   : tipocomp;
                    po       : integer;
```

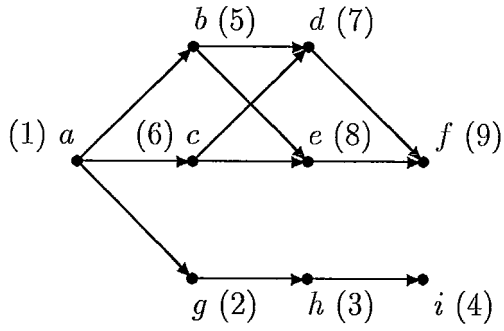


Figura **A.1:** Poset série-paralelo com elementos numerados da esquerda para a direita segundo árvore de decomposição aglutinada construída pelo algoritino.

```
                elmto     : tipoelmto;
                prim,ult,
                irmao     : noarvptr;
              end;
  elmtolistaptr = ^elmtolista;
  elmtolista    = record
                elmto      : tipoelmto;
                ant, prox : elmtolistaptr;
              end;

var
  arqent,arqsai   : text;
  T               : noarvptr;
  NCRS,NCRP,
  NExt,NPos,
  PO,n,nn,NFolhas : integer;
  Elmto           : array [tipoelmto] of elmtolistaptr;
  PosOrdem        : array [tipoelmto] of noarvptr;
  Prim, Ult,n_    : array [tipoelmto] of integer;
```

```
Procedure AbreArquivos;
var
  aux,
  nomearqent,nomearqsai  : string;

begin
  nomearqent := paramstr (1);
  if nomearqent = '' then
  begin
    writeln ('Arquivo que contem arvore de decomposicao : *.SP',
             ^H,^H,^H,^H);
    readln (nomearqent);
  end; { if }
  aux := nomearqent;
  while pos('\',aux) <> 0 do
    aux := copy (aux,pos('\',aux)+1,length(aux));
  if pos('.',aux) = 0 then
  begin
    nomearqsai := nomearqent + '.EXT';
    nomearqent := nomearqent + '.SP';
  end { if }
  else
    nomearqsai := copy(nomearqent,1,pos(nomearqent,'.')) + '.EXT';

  assign (arqsai,nomearqsai);
  assign (arqent,nomearqent);
  reset (arqent);
  rewrite (arqsai);
end; { procedure AbreArquivos }

Procedure FechaArquivos;
begin
```

```
  close(arqent) ;
  close(arqsai) ;
end; { procedure FechaArquivos }


Procedure Relatorio;
begin
  writeln(arqsai,^M,^M,^M);
  writeln(arqsai,' Numero de Extensoes Lineares do Poset          : ',
                  NExt);
  writeln(arqsai,' Numero de Chamadas Recursivas a Nos Serie      . ',
                  NCRS);
  writeln(arqsai,' Numero de Chamadas Recursivas a Nos Paralelos  : ',
                  NCRP);
  writeln(arqsai,' Numero de Chamadas Recursivas                  . ',
                  NCRS + NCRP);
  writeln(arqsai,' Numero de Alteracoes na Posicao de um Elemento');
  writeln(arqsai,'                                    na Extensao : ',
                  NPos);
end;{ procedure Relatorio }


Procedure LeArvore(var  T : noarvptr);
var
  arv             : array [tipoelmto] of noarvptr;
  strl            : string;
  i,j,k,indraiz,
  dummy           : integer;

begin
  readln (arqent,n);
  readln (arqent,indraiz);
  for i := 1 to (2*n - 1) do
```

```
    new (Arv[i]);
  for i := 1 to (2*n - 1) do
  begin
    readln (arqent,str1);
    strl := ' ' + strl + ' ';
    strl := copy (strl,pos(' ',str1)+1, length(str1));
    Arv[i]^.tipono := upcase (strl [1]);
    if Arv[i]^.tipono <> 'F' then
    begin
      strl := copy (str1,pos(' ',str1)+1, length(str1));
      val (copy(str1,1,pos(' ',str1)-1),j,dummy);
      Arv[i]^.prim := Arv[j];
      strl := copy (str1,pos(' ',str1)+1, length(str1));
      val (copy(str1,1,pos(' ',str1)-1),k,dummy);
      Arv[i]^.ult := Arv[k];
      Arv[j]^.irmao := Arv[k];
      Arv[k]^.irmao := nil;
    end { if }
    else { no e uma folha da arvore }
    begin
      Arv[i]^.prim := nil;
      Arv[i]^.ult := nil;
    end; { else }
  end; { for }
  T := Arv[indraiz];
end; { LeArvore }



Procedure AglutinaArvore (v : noarvptr);
{ Supoe-se que a arvore nao-aglutinada tenha a mesma estrutura
  da arvore aglutinada, sendo que aquela e estritamente binaria
  (filho direito = prim e filho esquerdo = ult).
```

Vou percorrer a arvore segundo uma busca em profundidade, aglutinando cada par de nos S consecutivos.

```
var
  w,ant : noarvptr;

begin
  w := v^.prim;
  ant := nil;
  while (w <> nil) do
  begin
    AglutinaArvore(w);
    if (w^.tipono = 'S') and (v^.tipono = 'S') then
    begin
      if (w = v^.prim) then
      begin
        v^.prim := w^.prim;
        w".ult^.irmao := w^.irmao;
      end { if }
      else
      begin
        if w = v^.ult then
        begin
          v^.ult := w^.ult;
          ant^.irmao := w^.prim;
        end { if }
        else { w <> v^.prim e v^.ult }
        begin
          ant^.irmao := w^.prim;
          w^.ult^.irmao := w^.irmao;
        end; { else }
      end; { else }
      ant := w;
```

```
      w := w^.ult;
      dispose(ant) ;
      Dec(nn) ;
    end; { if }
    ant := w;
    w := w^.irmao;
  end; { while }
end; { procedure AglutinaArvore }

Procedure OrdenaFilhosPar(v : noarvptr);
{ Aplica busca em profundidade sobre T, numerando nos(inclusive as
  folhas) e reordenando filhos de um no paralelo quando necessario. }
var
  nv   : integer;
  w    : noarvptr;

begin
  nv := ord(v^.tipono='F');
  w := v^.prim;
  while w <> nil do
  begin
    OrdenaFilhosPar (w);
    Inc(nv, n_[PO]);
    w := w^.irmao;
  end; { while }
  Inc(PO) ;
  v^.po := PO;
  n_[PO] := nv;
  if (v^.tipono = 'P') and (n_[v^.prim^.po] > (nv/2)) then
  begin
    w := v^.prim;
    v^.prim := v^.ult;
    v^.ult := w;
```

```
    v^.prim^.irmao := v^.ult;
    v^.ult^.irmao := nil;
  end; { if }
end; { procedure OrdenaFilhosPar }



Procedure NumeraPosOrdem(v  : noarvptr);
{ Numera somente os nos internos da arvore aglutinada em pos-ordem,
  enquanto calcula o numero de elementos que correspondem a cada um
  deles.                                                          }

var
  nv  : integer;
  w   : noarvptr;

begin
  nv := 0;
  w := v^.prim;
  while w <> nil do
  begin
    if w^.tipono = 'F' then
    begin
      Inc(nv) ;
      Inc(NFolhas); { incrementa numero de folhas da arvore, dando a
                      numeracao da esq. para dir. em T dos elementos
                      do poset                                      }
      w^.elmto := NFolhas;
    end
    else
    begin
      NumeraPosOrdem (w);
      Inc(nv,n_ [PO]);                  { PO = # em pos-ordem de w }
```

```
    end; { else }
    w := w^.irmao;
  end; { while }
  Inc (PO);                                  { PO = # em pos-ordem de v }
  PosOrdem [PO] := v;
  v^.po := PO;
  n_ [PO] := nv;
end; { procedimento NumeraPosOrdem }


Procedure ImprimeArv;
var
  i,j   : integer;
  aux   : noarvptr;

begin
   writeln(arqsai,'  * ARVORE AGLUTINADA DO POSET :');
   writeln (arqsai,'  (um  no entre parenteses representa uma ',
                  'folha/elemento ');
   writeln(arqsai, '    do poset, numerados da esquerda para a direita ',
                  'na arv.)',^M);
   writeln (arqsai,'         Tipo(Serie/','Filhos':14);
   writeln (arqsai,'          Paralelo)',^M);
   for i := 1 to nn do
   begin
    write(arqsai,'  No ',i,' : ',PosOrdem[i]^.tipono:6,' ':13);
    aux := PosOrdem[i] ^.prim;
    while aux <> nil do
    begin
      if aux^.tipono <> 'F' then
      begin
        j := I;
        while (PosOrdem[j] <> aux) do
```

```
          Inc (j);
        write(arqsai, ' ',j,'  ');
      end { if }
      else
        write (arqsai,'(',aux^.elmto,') ');
      aux := aux^.irmao;
    end; { while }
    writeln(arqsai) ;
  end; { for }
  writeln(arqsai,^M,^M,^M);
  writeln(arqsai,' * EXTENSOES LINEARES :',^M);
end; { procedure ImprimeArv }


Procedure ImprimeExt(i  : tipoelmto);
var
  auxlista  : elmtolistaptr;

begin
  auxlista := Elmto[i];
  Inc (NExt);
  write(arqsai,'  #',NExt,' : ');
  while auxlista <> nil do
  begin
    write (arqsai,auxlista^.elmto,'  ');
    auxlista := auxlista^.prox;
  end; { while }
  writeln(arqsai);
end; { procedure ImprimeExt }


Procedure CriaExtCanonica ;
var
```

```
  i : tipoelmto;

begin
  new(Elmt o [1]);
  Elmto[1]^.elmto := 1;
  Elmto[1]^.ant := nil;
  for i := 2 to n do
  begin
    new(Elmt o[i]);
    Elmto[i]^.elmto := i;
    Elmto[i-1]^.prox := Elmto[i];
    Elmto[i]^.ant := Elmto[i-1];
  end; { for }
  Elmto[n]^.prox := nil;
end; { procedure CriaExtCanonica }



Procedure Concatena(no  : integer; v,w : noarvptr);
var
  primv,ultv,
  primw,ultw   : tipoelmto;

begin
  if v^.tipono = 'F' then
  begin
    primv := v^.elmto;
    ultv := v^.elmto;
  end { if }
  else
  begin
    primv := Prim[v^.po];
    ultv := Ult[v- .po];
```

```
  end; { else }
  if w^.tipono = 'F' then
  begin
    primw := w^.elmto;
    ultw := w^.elmto;
  end { if }
  else
  begin
    primw := Prim[w^.po];
    ultw := Ult[w^.po];
  end; { else }

  Prim[no] := primv;
  Ult[no]  := ultw;
  if Elmtq[ultw]^.prox^.elmto = primv then
  { ext(v) e ext(w) estao em ordem trocada 'a desejada                    }
  begin
    Elmtq[ultw]^.prox := Elmtq[ultv]^.prox;
    if Elmto[primw]^.ant <> nilthen   { se primw nao era 1o. lista }
      Elmto[Elmto[primw]^.ant^.elmto]^.prox := Elmto[primv];
    if Elmtq[ultv]^.prox <> nil then   { se ultv nao era ult. lista }
      Elmto[Elmto[ultv]^.prox^.elmto]^.ant := Elmto[ultw];
    Elmtq[ultv]^.prox := Elmtq[primw];
    Elmto[primv]^.ant := Elmtq[primw]^.ant;
    Elmto[primw]^.ant := Elmtq[ultv];
  end; { if }
  { senao nao preciso fazer nada }
end; { procedure Concatena }


Procedure Insere(no  : integer; i,j : tipoelmto);
var
  auxl,aux2 : elmtolistaptr;
```

```
begin
  Inc (NPos);
  if Elmto[j]^.prox <> Elmto[i] then
  { se elementos ja nao estavam na posicao desejada }
  begin
    if i = Prim[no] then
      Prim[no] := Elmto[i]^.prox^.elmto
    else
       if i = Ult[no] then
          Ult[no] := Elmto[i]^.ant^.elmto;
    if j = Ult[no] then
      Ult[no] := i;

    aux1 := Elmto[i]^.prox;
    aux2 := Elmto[j]^.prox;
    if Elmto[i]^.ant <> nil then { se i nao era 1o. lista encadeada }
      Elmto[i]^.ant^.prox := Elmto[i]^.prox;
    Elmto[i]^.prox := Elmto[j]^.prox;
    Elmto[j]^.prox := Elmto[i];
    if auxl <> nil then          { se i nao era ult. lista encadeada }
      auxl^.ant := Elmto[i]^.ant;
    Elmto[i]^.ant := Elmto[j];
    if aux2 <> nil then          { se j nao era ult. lista encadeada }
      aux2^.ant := Elmto[i];
  end { if }
  { senao nao preciso fazer nada }
end; { procedure Insere }


Procedure Geracao (k : integer); forward;

Procedure GeraExtSerie (k : integer);
```

```
begin
  if PosOrdem[k]^.prim^.tipono = 'F' then
    Prim[k] := PosOrdem[k]^.prim^.elmto
  else
    Prim[k] := Prim [PosOrdem[k]^.prim^.po];
  if PosOrdem[k]^.ult^.tipono = 'F' then
    Ult[k] := PosOrdem[k]^.ult^.elmto
  else
    Ult[k] := Ult [PosOrdem[k]^.ult^.po];

  if k = nn then { se k e raiz de T }
    ImprimeExt (Prim [k])
  else
    Geracao (k+1);
end; { procedure GeraExtSerie }


Procedure GeraExtParal (k : integer);
var
  filhoesq,
  filhodir  : noarvptr;
  r,m,h,j   : integer;
  a         : array [tipoelmto] of integer;
  assoc     : array [tipoelmto] of tipoelmto;

Function AchaMinTroca : integer;
var i : integer;

begin
  i := 1;
  while a[r+1-i] = (n_[k] + 1 - i) do
    Inc(i);
  AchaMinTroca := i;
```

```
end; { function AchaMinTroca }


{ procedure GeraExtParal }
begin
{ Como ja ordenamos os filhos esq e dir de k segundo seus numeros
  de descendentes em T, sabemos que n_[filho esq] <= n_[k]/2 .      }

  filhoesq := PosOrdem[k]^.prim;
  filhodir := PosOrdem[k]^.ult;
  Concatena (k,filhoesq,filhodir);
  if filhoesq^.tipono = 'F' then
    r := 1
  else
    r := n_[filhoesq^.po];

  a[1] := 1;
  assoc[1] := Prim[k];
  Inc (NPos);
  for j := 2 to r do
  begin
    Inc (NPos);
    a[j] := j;
    assoc[j] := Elmto[assoc[j-1]]^.prox^.elmto;
  end; { for }
  if k = nn then { k e a raiz de T }
    ImprimeExt (Prim[k])
  else
    Geracao (k+1);
  repeat
    h := AchaMinTroca;
    m := a[r+1-h];
    a[r+1-h] := m + 1;
```

```
    Insere (k,assoc[r+1-h],Elmto[assoc[r+1-h]]^.prox^.elmto);
    for j := 2 to h do
    begin
      a[r+j-h] := m + j;
      Insere (k,assoc[r+j-h], assoc[r+j-h-1]);
    end; { for }
    if k = nn then { k e a raiz de T }
      ImprimeExt (Prim[k])
    else
      Geracao (k+1);
  until a[1] = n_[k] - r + 1;        { ja gerei todas as combinacoes }
end; { procedure GeraExtParal }


Procedure Geracao (k : integer);
begin
  if PosOrdem[k]^.tipono = 'S' then
  begin
    Inc(NCRS);
    GeraExtSerie (k)
  end { if }
  else
  begin
    Inc (NCRP);
    GeraExtParal (k);
  end; { else }
end; { procedure Geracao }


{ programa principal }
begin
  { Suponha-se dado um poset serie-paralelo por meio de sua arvore
    de decomposicao  estritamente binaria.                          }
```

```
  AbreArquivos;
  LeArvore (T);
  nn := n-1;
  AglutinaArvore (T);
  PO := 0;
  OrdenaFilhosPar(T);
  PO := 0; NFolhas := 0;
  NumeraPosOrdem (T);
  ImprimeArv;

  CriaExtCanonica;
  NExt := 0; NPos := 0;
  NCRS := 0; NCRP := 0;
  Geracao (1);
  Relatorio;
  FechaArquivos;
end. { programa principal }
```

O Exemplo

```
 * ARVORE AGLUTINADA DO POSET :
   (um  no entre parenteses representa uma folha/elemento
     do poset, numerados da esquerda para a direita na arv.)
          Tipo(Serie/        Filhos
           Paralelo)
 No 1 :       S             (2) (3) (4)
 No 2 :       P             (5) (6)
 No 3 :       P             (7) (8)
 No 4 :       S              2   3  (9)
 No 5 :       P              1   4
 No 6 :       S             (1)  5
```

* **EXTENSOES LINEARES** :

#1 : 1 2 3 4 5 6 7 8 9
#2 : 1 2 3 5 4 6 7 8 9
#3 : 1 2 3 5 6 4 7 8 9
#4 : i 2 3 5 6 7 4 8 9
#5 : i 2 3 5 6 7 8 4 9
#6 : 1 2 3 5 6 7 8 9 4
#7 : 1 2 5 3 4 6 7 8 9
#8 : 1 2 5 3 6 4 7 8 9
#9 : 1 2 5 3 6 7 4 8 9
#10 : 1 2 5 3 6 7 8 4 9
#11 : 1 2 5 3 6 7 8 9 4
#12 : 1 2 5 6 3 4 7 8 9
#13 : 1 2 5 6 3 7 4 8 9
#14 : 1 2 5 6 3 7 8 4 9
#15 : 1 2 5 6 3 7 8 9 4
#16 : 1 2 5 6 7 3 4 8 9
#17 : 1 2 5 6 7 3 8 4 9
#18 : 1 2 5 6 7 3 8 9 4
#19 : 1 2 5 6 7 8 3 4 9
#20 : 1 2 5 6 7 8 3 9 4
#21 : i 2 5 6 7 8 9 3 4
#22 : 1 5 2 3 4 6 7 8 9
#23 : 1 5 2 3 6 4 7 8 9
#24 : 1 5 2 3 6 7 4 8 9
#25 : 1 5 2 3 6 7 8 4 9
#26 : 1 5 2 3 6 7 8 9 4
#27 : 1 5 2 6 3 4 7 8 9
#28 : 1 5 2 6 3 7 4 8 9
#29 : 1 5 2 6 3 7 8 4 9
#30 : 1 5 2 6 3 7 8 9 4
#31 : 1 5 2 6 7 3 4 8 9

#32 : 1 5 2 6 7 3 8 4 9
#33 : 1 5 2 6 7 3 8 9 4
#34 : 1 5 2 6 7 8 3 4 9
#35 : 1 5 2 6 7 8 3 9 4
#36 : 1 5 2 6 7 8 9 3 4
#37 : 1 5 6 2 3 4 7 8 9
#38 : 1 5 6 2 3 7 4 8 9
#39 : 1 5 6 2 3 7 8 4 9
#40 : 1 5 6 2 3 7 8 9 4
#41 : 1 5 6 2 7 3 4 8 9
#42 : 1 5 6 2 7 3 8 4 9
#43 : 1 5 6 2 7 3 8 9 4
#44 : 1 5 6 2 7 8 3 4 9
#45 : 1 5 6 2 7 8 3 9 4
#46 : 1 5 6 2 7 8 9 3 4
#47 : 1 5 6 7 2 3 4 8 9
#48 : 1 5 6 7 2 3 8 4 9
#49 : 1 5 6 7 2 3 8 9 4
#50 : 1 5 6 7 2 8 3 4 9
#51 : 1 5 6 7 2 8 3 9 4
#52 : 1 5 6 7 2 8 9 3 4
#53 : 1 5 6 7 8 2 3 4 9
#54 : 1 5 6 7 8 2 3 9 4
#55 : 1 5 6 7 8 2 9 3 4
#56 : 1 5 6 7 8 9 2 3 4
#57 : 1 2 3 4 5 6 8 7 9
#58 : 1 2 3 5 4 6 8 7 9
#59 : 1 2 3 5 6 4 8 7 9
#60 : 1 2 3 5 6 8 4 7 9
#61 : 1 2 3 5 6 8 7 4 9
#62 : 1 2 3 5 6 8 7 9 4
#63 : 1 2 5 3 4 6 8 7 9
#64 : 1 2 5 3 6 4 8 7 9

#65 : 1 2 5 3 6 8 4 7 9
#66 : 1 2 5 3 6 8 7 4 9
#67 : 1 2 5 3 6 8 7 9 4
#68 : 1 2 5 6 3 4 8 7 9
#69 : 1 2 5 6 3 8 4 7 9
#70 : 1 2 5 6 3 8 7 4 9
#71 : 1 2 5 6 3 8 7 9 4
#72 : 1 2 5 6 8 3 4 7 9
#73 : 1 2 5 6 8 3 7 4 9
#74 : 1 2 5 6 8 3 7 9 4
#75 : 1 2 5 6 8 7 3 4 9
#76 : 1 2 5 6 8 7 3 9 4
#77 : 1 2 5 6 8 7 9 3 4
#78 : 1 5 2 3 4 6 8 7 9
#79 : 1 5 2 3 6 4 8 7 9
#80 : 1 5 2 3 6 8 4 7 9
#81 : 1 5 2 3 6 8 7 4 9
#82 : 1 5 2 3 6 8 7 9 4
#83 : 1 5 2 6 3 4 8 7 9
#84 : 1 5 2 6 3 8 4 7 9
#85 : 1 5 2 6 3 8 7 4 9
#86 : 1 5 2 6 3 8 7 9 4
#87 : 1 5 2 6 8 3 4 7 9
#88 : 1 5 2 6 8 3 7 4 9
#89 : 1 5 2 6 8 3 7 9 4
#90 : 1 5 2 6 8 7 3 4 9
#91 : 1 5 2 6 8 7 3 9 4
#92 : 1 5 2 6 8 7 9 3 4
#93 : 1 5 6 2 3 4 8 7 9
#94 : 1 5 6 2 3 8 4 7 9
#95 : 1 5 6 2 3 8 7 4 9
#96 : 1 5 6 2 3 8 7 9 4
#97 : 1 5 6 2 8 3 4 7 9

#98 : 1 5 6 2 8 3 7 4 9
#99 : 1 5 6 2 8 3 7 9 4
#100 : 1 5 6 2 8 7 3 4 9
#101 : 1 5 6 2 8 7 3 9 4
#102 : 1 5 6 2 8 7 9 3 4
#103 : 1 5 6 8 2 3 4 7 9
#104 : 1 5 6 8 2 3 7 4 9
#105 : 1 5 6 8 2 3 7 9 4
#106 : 1 5 6 8 2 7 3 4 9
#107 : 1 5 6 8 2 7 3 9 4
#108 : 1 5 6 8 2 7 9 3 4
#109 : 1 5 6 8 7 2 3 4 9
#110 : 1 5 6 8 7 2 3 9 4
#111 : 1 5 6 8 7 2 9 3 4
#112 : 1 5 6 8 7 9 2 3 4
#113 : 1 2 3 4 6 5 7 8 9
#114 : 1 2 3 6 4 5 7 8 9
#115 : 1 2 3 6 5 4 7 8 9
#116 : 1 2 3 6 5 7 4 8 9
#117 : 1 2 3 6 5 7 8 4 9
#118 : 1 2 3 6 5 7 8 9 4
#119 : 1 2 6 3 4 5 7 8 9
#120 : 1 2 6 3 5 4 7 8 9
#121 : 1 2 6 3 5 7 4 8 9
#122 : 1 2 6 3 5 7 8 4 9
#123 : 1 2 6 3 5 7 8 9 4
#124 : 1 2 6 5 3 4 7 8 9
#125 : 1 2 6 5 3 7 4 8 9
#126 : 1 2 6 5 3 7 8 4 9
#127 : 1 2 6 5 3 7 8 9 4
#128 : 1 2 6 5 7 3 4 8 9
#129 : 1 2 6 5 7 3 8 4 9
#130 : 1 2 6 5 7 3 8 9 4

#131 : 1 2 6 5 7 8 3 4 9
#132 : 1 2 6 5 7 8 3 9 4
#133 : 1 2 6 5 7 8 9 3 4
#134 : 1 6 2 3 4 5 7 8 9
#135 : 1 6 2 3 5 4 7 8 9
#136 : 1 6 2 3 5 7 4 8 9
#137 : 1 6 2 3 5 7 8 4 9
#138 : 1 6 2 3 5 7 8 9 4
#139 : 1 6 2 5 3 4 7 8 9
#140 : 1 6 2 5 3 7 4 8 9
#141 : 1 6 2 5 3 7 8 4 9
#142 : 1 6 2 5 3 7 8 9 4
#143 : 1 6 2 5 7 3 4 8 9
#144 : 1 6 2 5 7 3 8 4 9
#145 : 1 6 2 5 7 3 8 9 4
#146 : 1 6 2 5 7 8 3 4 9
#147 : 1 6 2 5 7 8 3 9 4
#148 : 1 6 2 5 7 8 9 3 4
#149 : 1 6 5 2 3 4 7 8 9
#150 : 1 6 5 2 3 7 4 8 9
#151 : 1 6 5 2 3 7 8 4 9
#152 : 1 6 5 2 3 7 8 9 4
#153 : 1 6 5 2 7 3 4 8 9
#154 : 1 6 5 2 7 3 8 4 9
#155 : 1 6 5 2 7 3 8 9 4
#156 : 1 6 5 2 7 8 3 4 9
#157 : 1 6 5 2 7 8 3 9 4
#158 : 1 6 5 2 7 8 9 3 4
#159 : 1 6 5 7 2 3 4 8 9
#160 : 1 6 5 7 2 3 8 4 9
#161 : 1 6 5 7 2 3 8 9 4
#162 : 1 6 5 7 2 8 3 4 9
#163 : 1 6 5 7 2 8 3 9 4

#164 : 1 6 5 7 2 8 9 3 4
#165 : 1 6 5 7 8 2 3 4 9
#166 : 1 6 5 7 8 2 3 9 4
#167 : 1 6 5 7 8 2 9 3 4
#168 : 1 6 5 7 8 9 2 3 4
#169 : 1 2 3 4 6 5 8 7 9
#170 : 1 2 3 6 4 5 8 7 9
#171 : 1 2 3 6 5 4 8 7 9
#172 : 1 2 3 6 5 8 4 7 9
#173 : 1 2 3 6 5 8 7 4 9
#174 : 1 2 3 6 5 8 7 9 4
#175 : 1 2 6 3 4 5 8 7 9
#176 : 1 2 6 3 5 4 8 7 9
#177 : 1 2 6 3 5 8 4 7 9
#178 : 1 2 6 3 5 8 7 4 9
#179 : 1 2 6 3 5 8 7 9 4
#180 : 1 2 6 5 3 4 8 7 9
#181 : 1 2 6 5 3 8 4 7 9
#182 : 1 2 6 5 3 8 7 4 9
#183 : 1 2 6 5 3 8 7 9 4
#184 : 1 2 6 5 8 3 4 7 9
#185 : 1 2 6 5 8 3 7 4 9
#186 : 1 2 6 5 8 3 7 9 4
#187 : 1 2 6 5 8 7 3 4 9
#188 : 1 2 6 5 8 7 3 9 4
#189 : 1 2 6 5 8 7 9 3 4
#190 : 1 6 2 3 4 5 8 7 9
#191 : 1 6 2 3 5 4 8 7 9
#192 : 1 6 2 3 5 8 4 7 9
#193 : 1 6 2 3 5 8 7 4 9
#194 : 1 6 2 3 5 8 7 9 4
#195 : 1 6 2 5 3 4 8 7 9
#196 : 1 6 2 5 3 8 4 7 9

```
#197 : 1  6  2  5  3  8  7  4  9
#198 : 1  6  2  5  3  8  7  9  4
#199 : 1  6  2  5  8  3  4  7  9
#200 : 1  6  2  5  8  3  7  4  9
#201 : 1  6  2  5  8  3  7  9  4
#202 : 1  6  2  5  8  7  3  4  9
#203 : 1  6  2  5  8  7  3  9  4
#204 : 1  6  2  5  8  7  9  3  4
#205 : 1  6  5  2  3  4  8  7  9
#206 : 1  6  5  2  3  8  4  7  9
#207 : 1  6  5  2  3  8  7  4  9
#208 : 1  6  5  2  3  8  7  9  4
#209 : 1  6  5  2  8  3  4  7  9
#210 : 1  6  5  2  8  3  7  4  9
#211 : 1  6  5  2  8  3  7  9  4
#212 : 1  6  5  2  8  7  3  4  9
#213 : 1  6  5  2  8  7  3  9  4
#214 : 1  6  5  2  8  7  9  3  4
#215 : 1  6  5  8  2  3  4  7  9
#216 : 1  6  5  8  2  3  7  4  9
#217 : 1  6  5  8  2  3  7  9  4
#218 : 1  6  5  8  2  7  3  4  9
#219 : 1  6  5  8  2  7  3  9  4
#220 : 1  6  5  8  2  7  9  3  4
#221 : 1  6  5  8  7  2  3  4  9
#222 : 1  6  5  8  7  2  3  9  4
#223 : 1  6  5  8  7  2  9  3  4
#224 : 1  6  5  8  7  9  2  3  4
```

Numero de Extensoes Lineares do Poset : 224
Numero de Chamadas Recursivas a Nos Serie : 229
Numero de Chamadas Recursivas a Nos Paralelos : 7
Numero de Chamadas Recursivas : 236

Numero de Alteracoes na Posicao de um Elemento
na Extensao : 338