ANÁLISE DE ALGORITMOS PARA DETERMINAÇÃO DE CAMINHOS MÍNIMOS ENTRE TODOS OS PARES DE NÓS DE UM GRAFO ORIENTADO

Carlos Alberto da Silva Franco

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA (M.Sc.)

Aprovada por:

Presidente

RIO DE JANEIRO

ESTADO DO RIO DE JANEIRO - BRASIL

JUNHO DE 1975

AGRADECIMENTOS

De forma especial a Nelson Maculan Filho pela confiança em que minha atuação seja útil ao Programa de Sistemas e também por seu constante incentivo.

A João Lizardo, não só por admitir ser o responsável pelo desenvolvimento deste trabalho menor, mas também por ser o modêlo de capacidade que todos nós, seus alunos, procuramos alcançar.

A Paulo Boaventura por ter concordado, em circunstâncias especiais, em fazer parte da banca examinadora.

SUMÁRIO

A presente exposição trata do problema de determinação do menor caminho entre cada par de vértices de um grafo orientado.

Primeiramente são discutidos e comparados os tratamentos que diversos autores dão aos procedimentos considerados mais eficientes para a solução daquele problema.

Em seguida dois processos são analizados com maior detalhe e é proposta em um deles, uma modificação que resulta em uma diminuição significativa do esforço computacional.

São apresentadas listagens dos programas escritos para cada processo, bem como para as modificações discutidas, anexando-se ainda os resultados da verificação experimental.

A B S T R A C T

The present work is concerned with the problem of determining the shortest path between each pair of nodes in a directed graph. At first we discuss and compare the approaches followed by several authors to the procedures considered to be the most efficient for the solution of that problem.

We then proceed to analyse two procedures in greater detail; for one of them we propose a modification that significantly decreases computational effort. Program listings are offered for the procedures and modifications discussed, as well as the results of experimental verification.

Í N D I C E

CAPÍTULO I

1. Introdução

O problema de determinação de caminhos mínimos em grafos, tem sido extensamente abordado em livros e artigos sobre teoria de grafos, aplicações de grafos, pesquisa operacional, etc. Nas referências consultadas para execução deste trabalho, duas linhas de procedimento foram claramente observadas: uma é a que se propõe ao estabelecimento de provas formais para os diversos métodos de busca e apresentam, eventualmente, indicações mais ou menos precisas sobre a forma computacional dos métodos. A outra, mais objetiva, apresenta (algumas vezes em virtude de observação das provas formais)discussões informais sobre os métodos e descrição, na maior parte das vezes completa mas nem sempre correta, dos passos dos algoritmos projetados para resolução daquele problema e, muitas vezes, o algoritmo completo já programado em uma linguagem de alto nível.

A idéia original que ocasionou a presente apresentação, surgiu da observação de um trabalho de Minieka | [12] |, que se enquadra no segundo grupo descrito acima.

Assim, todo o desenvolvimento que se segue, tem aspecto eminentemente prático quando examina métodos de busca projetados para problemas particulares, métodos esses com provas formais em uma ou mais das referências apresentadas. O trabalho de Minieka | [12] |, apresenta dois algoritmos para determinação dos K menores caminhos entre cada par de vértices de um grafo orientado, com arcos de comprimento positivo e/ou negativo. Os dois processos propostos no referido trabalho são generalizações de dois algoritmos

projetados para determinação do menor caminho entre cada par de vértices de um grafo orientado com arcos de custo positivo e/ou negativo.

Como os procedimentos propostos naquele trabalho são generalizações de outros, dos quais diz o autor serem correntemente considerados como os mais eficientes para resolução do problema de determinação do menor caminho entre todos os pares de vértices, procurou-se (com base nessa asservativa) determinar se era possível, para este problema:

a) acelerar os referido métodos, ou

b) projetar um algoritmo mais eficiente.

Assim, o desenvolvimento deste trabalho tem como segunda característica, o tratamento do problema de determinação do menor caminho entre todos os pares de vértices de um grafo orientado. No estudo das referências disponíveis, tratando desse problema algumas conclusões interessantes são obtidas e relatadas na seção seguinte.

2. Preliminares

Para resolução do problema de determinação do caminho mínimo entre cada par de vértices de um grafo orientado, dois tipos de processos podem ser identificados:

I - os projetados especificamente para este tipo de problema e

II- os projetados para determinar o menor caminho entre um par de vértices e que aplicados uma vez à cada par do grafo, resolvem o mesmo problema.

Além disso, esses processos podem envolver o tratamento de estrutura de árvore, métodos iterativos ou matriciais (produto de matrizes).

Os processos mais eficientes do tipo I são o de Floyd | 6 |, o de Dantzig (ambos discutidos em Dreyfus | 2 | e nos quais se baseia o trabalho de Minieka | 12 |), o de Bilde and Krarup | 1 | (desenvolvido como aperfeiçoamento do processo de Farbey, Landand Murchland | 5 |) e o chamado "Revised Cascade Method" (RCM) apresentado por Elmaghraby | 4 |. Destes, o de Dantzig e os das referências | 6 |, | 1 | e | 4 | são métodos iterativos e o de | 5 | é apresentado como resultado de operações de produto matricial, feito segundo regras particulares.

Do tipo II os processos mais eficientes são o de Whiting and Hillier | 17 | e o de Dijkstra (como analisado nas referências | 2 |, | 4 | e | 10 |), ambos apresentados originalmente como processos de busca em árvore.

Do exame dessas referências, observa-se primeiramente que todas apresentam algoritmos projetados para determinar o comprimento (ou custo) dos menores caminhos e não os caminhos em si (isto é: uma sequência ordenada de nós intermediários entre um nó origem e um nó terminal); embora os caminhos possam ser obtidos trivialmente em todos os processos, isto exige sempre pequenas modificações nos mesmos. Assim, mantendo-se ao lado desta posição geral, o desenvolvimento a seguir analisa processos (e finalmente propõe uma modificação) que deveriam ser com mais rigor chamados de processos de determinação de custo mínimo, não se preocupando em discutir o custo computacional de modificações introduzidas para obtenção do caminho. Isto finalmente se revelará de nenhuma importância, uma vez que os dois processos que serão comparados, permitem a mesma modificação (ocasionando exatamente o mesmo custo computacional) para obtenção dos caminhos.

O segundo ponto observado, relaciona-se com as referências | 1 |, | 4 |, | 5 | e | 6 |. Floyd | 6 | apresentou em 1962, um processo para determinação do custo mínimo, em forma de um algoritmo de nove linhas escrito em linguagem Algol. O desconhecimento deste trabalho (que se mostrará experimentalmente no Capítulo II ser o mais eficiente) levou Farbey, Land e Murchland | 5 | e publicar em 1967, um procedimento que tem alguma semelhança com o de Floyd, porém bem ineficiente se comparado com o mesmo, uma vez que exige maior número de comparações e adições, mais memória, além de exigir uma programação mais complexa. Este procedimento recebeu dos autores o nome de "Cascade Algorithm".

Tendo por base o trabalho de | 5 |, Bilde e Krarup | 1 |, reivindicando uma prova mais simples para a mesma idéia, introduzem uma modificação que reduz em 50% o número de "operações" requeridas para determinar uma matriz de custo D, onde cada d_{ij} é o custo do menor caminho do nó i ao nó j em um grafo orientado. Uma "operação" para Bilde e Krarup é uma aplicação da forma

$$d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \quad (1.2.1),$$

que envolve uma soma e uma comparação.

A utilização do termo "operação" no sentido acima, também é utilizada em | 2 | e | 4 | e conduz, senão à incorreções, pelo menos à possibilidade de uma interpretação imprecisa quando da comparação entre algoritmos, tomando por base o número de operações. Com efeito, Elmaghraby | 4 | na página 4 de seu trabalho, refere-se ao método de Floyd, dizendo que o mesmo requer $n(n - 1)(n - 2)$ operações elementares; mais adiante na mesma página refere-se ao método de Dijkstra dizendo que o mesmo requer $3n^3$ (o que em grandeza já é incorreto) operações elementares, onde uma operação elementar é uma soma ou uma comparação, consideração esta última que torna incorreta a afirmação

sobre o número de operações requeridas pelo método de Floyd.

Dreyfus | [2] |, utiliza por vezes uma forma imprecisa quando fala do número de operações. Na página 403 daquele trabalho por exemplo, é afirmado que o método de Floyd requer n(n-1)(n-2) adições e comparações; isto é verdade uma vez que se considere a aplicação de 1.2.1, mas é inconsistente se comparado com o restante do trabalho, quando adições e comparações são consideradas sempre isoladamente,

O método de Bilde e Krarup | [1] | é essencialmente igual ao método de Floyd e recebeu dos autores a denominação de "Modified Cascade Algorithm".

Elmaghraby | [4] | apresenta em 1970 o método de Floyd, com a denominação de "Revised Cascade Method". Neste trabalho, como em | [1] | e | [2] |, aquele método é apresentado como requerendo n(n-1)(n-2) operações.

Essas observações sobre inconsistencia ou incorreções menores não são feitas como tentativa de invalidar os trabalhos em que elas ocorrem, mas sim com o objetivo de estabelecer com maior exatidão o número de operações requeridas por procedimentos diferentes (visando um mesmo resultado)que é em realidade o que o usuário utiliza para uma tomada de decisão.

No capítulo seguinte, o método de Floyd, doravante chamado de "procedimento de Floyd" será apresentado por completo, determinando-se o número de adições e comparações requeridas, juntamente com resultados experimentais da modificação do processo segundo | [1] | e | [4] |, que mostrarão ser as referidas apresentações ineficientes quando comparadas com o trabalho original | [6] |. Deve-se observar que o processo de Dantzig como apresentado em | [2] | e | [12] | requer o mesmo número de adições e comparações que o processo de Floyd, exceto para casos particulares como examinados por Grassin

e Minoux | 7 | por exemplo (grafos não orientados e de arcos com custos positivos). Porém por exigir programação mais complexa, este processo praticamente não é utilizado.

A generalização do método de Dijkstra, doravante chamado "procedimento de Dijkstra", será apresentado no Capítulo III, mostrando-se o número de adições e comparações por ele requerido e uma modificação, que toma por base a apresentação feita por Dreyfus | 2 |, será discutida com seus resultados computacionais. Na apresentação desses dois procedimentos, empregar-se-á o termo "operação" como uma adição ou uma comparação. O motivo de se fazer a análise tomando por base esses dois tipos de operação é por serem elas realmente dominantes em termos de tempo de máquina, quando comparadas com outras (assumindo-se uma equivalência entre o número de operações de indexação nos processos comparados). O fator considerado para comparação entre procedimentos será o tempo, embora alguma observação sobre memória seja feita eventualmente. O resultado finalmente obtido, diz respeito à aceleração de um procedimento que já existe. Superar a simplicidade e eficiência do procedimento de Floyd para o problema em questão, pelo desenvolvimento de um novo algoritmo, revelou-se uma tarefa complexa o suficiente para ser abandonada em um primeiro ataque, embora continue sendo uma possibilidade de algum interesse.

C A P Í T U L O II

1. Introdução

O procedimento de Floyd, resolve o problema de determinação do menor custo entre cada par de nós de um grafo orientado, com arcos de custo positivo e/ou negativo. Como apresentado por | 1 |, | 2 |, | 4 | e pelo próprio Floyd | 6 |, circuitos de custo negativo não são permitidos, porém uma pequena modificação no procedimento conduz a resultados corretos em tal caso. A afirmação contida em | 2 |, de que o número de operações requeridas pelo procedimento é n(n-1)(n-2) para o cado $d_{ij} \geq 0$ não é precisa, uma vez que são obtidos resultados corretos com um ou mais dos d_{ij} tendo valor negativo, existindo apenas a restrição da não existência de circuitos.

Para um grafo orientado com n nós, constrói-se uma matriz de distâncias D(n x n) com as seguintes características:

a) $d_{ii} = 0 \quad i$

b) d_{ij} = c, onde c é o custo associado ao arco que une os nós i e j($i \neq j$)

c) $d_{ij} = \infty$ se não existe arco unindo os nós i e j.

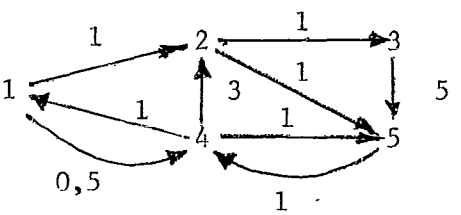
O procedimento determina o custo do caminho mínimo inserindo nós, quando apropriado, em caminhos menores. Começando com a matriz D de distâncias diretas, n matrizes são construidas sequencialmente (em realidade as modificações serão efetuadas sôbre a matriz original). A matriz de ordem K é interpretada como contendo os custos dos menores caminhos entre os pares (i, j), onde apenas caminhos com nós intermediários pertencendo ao conjunto

de nós de 1 à K são permitidos. A matriz de ordem K é construida da de ordem (K - 1) pela utilização de:

$$d_{ij}^{K} = \min(d_{ij}^{K-1}, d_{i,K}^{K-1} + d_{K,j}^{K-1}) \qquad (2.1.1)$$

$d_{ij}^{0} = d_{ij}$ são os elementos da matriz de distâncias diretas. K é inicialmente 1 e é incrementado de 1 depois de i e j (nesta ordem) terem assumido valores de 1 à N, até K = n no final.

Como exemplo, é apresentado um grafo com arcos de custo positivo, com cinco nós; para cada matriz de ordem K os elementos d_{ij} modificados são indicados em separado. Um ponto interessante é que o procedimento independe da numeração dos nós.



$$D^{0} = \begin{matrix} 0 & 1 & \infty & 0.5 & \infty \\ \infty & 0 & 1 & \infty & 1 \\ \infty & \infty & 0 & \infty & 5 \\ 1 & 3 & \infty & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{matrix}$$

K = 1

$$D^1 = \begin{matrix} 0 & 1 & \infty & 0.5 & \infty \\ \infty & 0 & 1 & \infty & 1 \\ \infty & \infty & 0 & \infty & 5 \\ 1 & 2 & \infty & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{matrix}$$

, aqui $d_{42} = d_{41} + d_{14}$

K = 2

$$D^2 = \begin{matrix} 0 & 1 & 2 & 0.5 & 2 \\ \infty & 0 & 1 & \infty & 1 \\ \infty & \infty & 0 & \infty & 5 \\ 1 & 2 & 3 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{matrix}$$

, aqui $d_{13} = d_{12} + d_{23}$

$d_{15} = d_{12} + d_{25}$

$d_{43} = d_{42} + d_{23}$

K = 3

$$D^3 = \begin{matrix} 0 & 1 & 2 & 0.5 & 2 \\ \infty & 0 & 1 & \infty & 1 \\ \infty & \infty & 0 & \infty & 5 \\ 1 & 2 & 3 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{matrix}$$

aqui não houve modificação de nenhm elemento, donde se pode concluir que o

nó de número 3 não é intermediário de nenhum caminho.

K = 4

$$D^4 = \begin{matrix} 0 & 1 & 2 & 0.5 & 1.5 \\ \infty & 0 & 1 & \infty & 1 \\ \infty & \infty & 0 & \infty & 5 \\ 1 & 2 & 3 & 0 & 1 \\ 2 & 3 & 4 & 1 & 0 \end{matrix}$$

aqui : $d_{15} = d_{14} + d_{45}$

$d_{51} = d_{54} + d_{41}$

$d_{52} = d_{54} + d_{42}$

$d_{53} = d_{54} + d_{43}$

K = 5

$$D^5 = \begin{matrix} 0 & 1 & 2 & 0.5 & 1.5 \\ 3 & 0 & 1 & 2 & 1 \\ 7 & 8 & 0 & 6 & 5 \\ 1 & 2 & 3 & 0 & 1 \\ 2 & 3 & 4 & 1 & 0 \end{matrix}$$

aqui : $d_{21} = d_{25} + d_{51}$

$d_{24} = d_{25} + d_{54}$

$d_{31} = d_{35} + d_{51}$

$d_{32} = d_{35} + d_{52}$

$d_{34} = d_{35} + d_{54}$

Como K = n, interrompe-se o processo.

Os caminhos (P) obtidos:

a) a partir do nó 1

P_{12} = 1,2 (arco direto)

P_{13} = 1,2,3

b) a partir do nó 2

P_{21} = 2,5,4,1

P_{23} = 2,3

$P_{14} = 1,4$

$P_{15} = 1,4,5$

$P_{24} = 2,5,4$

$P_{25} = 2,5$

c) a partir do nó 3

$P_{31} = 3,5,4,1$

$P_{32} = 3,5,4,1,2$

$P_{34} = 3,5,4$

$P_{35} = 3,5$

d) a partir do nó 4

$P_{41} = 4,1$

$P_{42} = 4,1,2$

$P_{43} = 4,1,2,3$

$P_{45} = 4,5$

e) a partir do nó 5

$P_{51} = 5,4,1$

$P_{52} = 5,4,1,2$

$P_{53} = 5,4,1,2,3$

$P_{54} = 5,4$

2. Número de operações do procedimento de Floyd

A aplicação da forma 2.1.1 é feita n^2 vezes para cada K. Tem-se então, no total, n^3 aplicações que correspondem à $2n^3$ operações realizadas sobre a matriz de custos diretos para que se obtenha a matriz de custos mínimos.

Este resultado pode ser obtido por inspeção em Floyd | 6 | e é apresentado também por Minieka | 12 |. Nas referências | 1 |, | 2 | e | 4 | fala-se em n(n-1)(n-2) operações, o que é menos preciso. Mais correto seria dizer 2n(n-1)(n-2) operações, quando algumas aplicações de 2.1.1 são evitadas, em virtude das considerações seguintes.

Para cada K, os elementos da K-ésima linha e da K-ésima coluna não precisam ser examinados, uma vez que não podem sofrer alteração dado o fato de ser o nó K, um nó terminal em d_{ik}(i) e o nó inicial em d_{Kj}(j); assim é desnecessário verificar se existe redução do custo pela introdução de um nó em um caminho onde ele é o nó inicial ou terminal. Tal fato só ocorreria caso fosse permitido circuito negativo.

Para cada K tem-se então $(n-1)^2$ aplicações de 2.1.1. Também em virtude de se ter considerado $d_{ii} = 0$ i e não se admitindo circuitos negativos, os elementos da diagonal principal não sofrerão, em todo o processo,qualquer tipo de alteração. Portanto, em cada K, onde já se está fazendo apenas $(n-1)^2$ aplicações de 2.1.1, mais (n-1) podem ser evitadas.

Assim, em termos de operações, o processo inteiro requer:

$$2n((n-1)^2 - (n-1)) = 2n(n-1)(n-2) =$$

$$2n^3 - 6n^2 + 4n \text{ operações} \quad (2.2.1)$$

O que não é observado em | 1 |, | 2 | e | 4 | é que embora 2.2.1 seja teoricamente correto, é necessário introduzir um certo número de operações do tipo comparação e/ou indexação para que se possa evitar algumas aplicações de 2.1.1; o número dessas operações extraordinárias, quando consideradas no processo, alteram 2.2.1. Na seção seguinte, são apresentadas duas formas de introdução dessas operações e seus resultados práticos são apresentados.

3. Tratamento computacional do procedimento

A versão original do procedimento de Floyd, requerendo $2n^3$ o-

perações, foi programada em linguagem FORTRAN. Com base nessa programação, tres modificações foram tentadas, visando obter o resultado 2.2.1:

a) introdução de comparações para evitar aplicações de 2.1.1 aos elementos de linhas e colunas.

b) introdução de comparações para evitar aplicações de 2.1.1 aos elementos de linhas, colunas e diagonal.

c) criação de um vetor auxiliar para evitar aplicações de 2.1.1 aos elementos de linhas e colunas.

As modificações do primeiro tipo descrito acima, requerem n^3 comparações adicionais; as do segundo tipo requerem $2n^3 - 2n^2 + n$ comparações adicionais(isto é trivialmente obtido por inspeção das listagens apresentadas no Apendice I, partes C e D - para o item a - e partes E e F - para o item b).

Temos então para o procedimento de Floyd:

I - original (sem eliminações) = $2n^3$ operações

II - evitando linhas e colunas(por comparação) = $3n^3 - 6n^2 + 4n$ operações

III- evitando linhas, colunas e diagonal (por comparação) = $4n^3 - 8n^2 + 5n$ operações

O tratamento do último item, não requer operações adicionais de comparação. A idéia é criar um vetor de n posições (seja IND o nome do vetor) e inicializá-lo com IND(N) = N. Para cada K no procedimento,

IND (K) = IND (1)

IND (1) = K.

Para variação dos i e j em 2.1.1, consideram-se agora os elementos (a partir do segundo) do vetor IND. Assim, todas as linhas e colu-

nas estão sendo consideradas, exceto as de ordem K para cada K (as listagens correspondentes à esta modificação estão no Apêndice I, partes G e H). O número de operações de indexação introduzidas, é de $n^3 - n^2$.

São apresentadas a seguir, duas tabelas que indicam o tempo gasto em processamento somente para alteração da matriz de distâncias diretas até se obter a matriz de custos mínimos. Como se pode notar nos programas apresentados no Apêndice I, este tempo não envolve leitura ou impressão de dados, ou chamada de qualquer tipo de sub-programa ou sub-rotina. O processo original e as modificações aplicam-se sempre ao mesmo grafo, que é gerado pelos próprios programas com características que serão vistas no Capítulo IV.

Nas tabelas 1 e 2 dadas a seguir empregou-se:

FLOYD - procedimento de Floyd sem modificações.

F.E.L.C.C.- procedimento de Floyd evitando linhas e colunas por comparações.

F.E.L.C.D.- procedimento de Floyd evitando linhas, colunas e diagonal.

F.E.L.C.V.- procedimento de Floyd evitando linhas e colunas por vetor.

A tabela 1 é o resultado de processamentos no sistema IBM-1130 e a tabela 2, no sistema B-6700. Em todos os casos a unidade de tempo é o segundo.

T A B E L A 1

Número de nós	FLOYD	F.E.L.C.C.	F.E.L.C.D.	F.E.L.C.V.
3	0.044	0.022	0.011	0.022
6	0.302	0.223	0.190	0.268
9	1.007	0.839	0.750	0.996
12	2.385	2.083	1.959	2.486
15	4.670	4.199	4.009	5.006
18	8.007	7.358	7.100	8.747
21	12.655	11.827	11.491	14.055
24	18.883	17.852	17.483	21.212
27	26.767	25.524	25.087	30.329
30	36.702	35.257	34.787	41.865
33	48.854	47.185	46.681	56.033
36	63.246	61.375	60.871	72.878
39	81.972	78.276	77.806	92.926
42	100.195	97.988	97.574	116.311
45	124.062	120.556	120.254	143.180
48	148.526	146.507	146.361	174.003
51	178.561	176.019	176.019	209.103
54	211.859	209.305	209.529	248.606
57	248.919	246.355	246.836	292.700
60	290.147	287.548	288.366	341.723
63	335.619	333.401	334.364	395.807
66	385.548	383.476	384.787	455.324
69	440.081	438.356	439.857	543.547

72	499.531	498.422	500.583	591.595
75	564.289	563.494	566.451	668.998
78	634.547	634.547	637.918	752.897
81	710.382	710.852	714.873	
84	792.097	793.183	797.439	

T A B E L A 2

Número de nós	FLOYD	F.E.L.C.C.	F.E.L.C.D.	F.E.L.C.V.
3	0.002	0.001	0.001	0.001
6	0.017	0.013	0.013	0.017
9	0.058	0.050	0.049	0.054
12	0.144	0.122	0.133	0.136
15	0.272	0.251	0.257	0.270
18	0.454	0.433	0.455	0.466
21	0.730	0.688	0.732	0.726
24	1.055	1.039	1.107	1.154
27	1.541	1.51	1.584	1.601
30	2.089	2.149	2.266	2.193
33	2.805	2.883	2.888	3.086
36	3.601	3.598	3.880	3.761
39	4.474	4.483	4.993	4.706
42	5.786	5.777	6.522	5.859

45	6.974	7.152	7.694	7.294
48	8.333	8.470	9.062	8.730
51	10.234	10.104	10.820	10.876
54	11.677	11.961	12.854	13.288
57	13.520	14.702	15.540	15.712
60	15.695	16.903	18.435	18.212
63	19.041	19.524	20.853	19.928
66	22.452	22.076	24.317	24.227
69	24.769	24.948	28.590	27.912
72	26.940	28.570	32.128	31.221
75	30.085	33.123	35.451	34.087
78	34.038	37.546	39.408	38.618
81	38.068	40.918	44.980	44.104
84	42.391	45.980	50.053	47.580
87	47.858	52.412	57.610	56.539
90	54.121	57.665	62.768	61.164
93	60.997	64.834	67.278	68.226

4 . Conclusões

É interessante observar que a tentativa da não aplicação de 2.1.1 à linhas e colunas, pela criação de um vetor auxiliar, envolve $n^3 - n^2$ operações do tipo

II = IND(I),

que se apresenta(para o sistema IBM-1130)bem pior que o esperado. Aqui a expectativa era de que o resultado obtido, estivesse próximo da forma II, isto porque as operações de indexação simples são, com frequência, consideradas como requerendo o mesmo tempo que as operações de comparação; entretanto, esta modificação se revela pior que a da forma III.

Deve-se observar ainda, que as modificações sobre o processo original, podem ter alguma vantagem para grafos de até um porte determinado. Assim o F.E.L.C.C. (modificação mais vantajosa), consome menos tempo para grafos com até 75 nós, para a máquina utilizada na determinação da tabela 1, e grafos com aproximadamente 40 nós,para a máquina utilizada na determinação da tabela 2.

O ponto para que se deve atentar é que essa "vantagem" para grafos de pequeno porte (com limite em função do sistema computacional empregado) não é significativa a não ser em casos extremos. Com efeito veja-se que no caso do sistema IBM-1130, para o trabalho em 1000 grafos de 42 nós, teriamos um ganho aproximado de 35 minutos em 27 horas se fosse utilizado o F.E.L.C.C. em lugar de FLOYD.

O que se tentou mostrar então, é que em geral a forma mais eficiente (que consome menos tempo, uma vez que a memória para as modificações dos primeiro e segundo tipos é a mesma e para o terceiro tipo apenas é ne-

cessário mais um vetor de n posições) do procedimento de FLOYD é a que aplica 2.1.1 à todos os elementos da matriz de distâncias original. Assim a comparação com outros procedimentos deve tomar por base o fato de que o procedimento de FLOYD requer $2n^3$ operações e não $2n^3 - 6n^2 + 4n$ como é usualmente apresentado.

C A P Í T U L O III

1. Introdução

O procedimento de Dijkstra foi originalmente proposto pelo mesmo como um método de busca que permite determinar uma solução (um caminho) para um problema em que se define um estado inicial (um nó raiz) e um estado objetivo (com um nó objetivo ou mais). O mesmo processo foi proposto por Whiting and Hillier | 17 | sem referência ao trabalho de Dijkstra e também por Nilsson | 13 | com o nome de "Uniform-Cost Method".

A aplicação do procedimento, origina uma "árvore de busca" onde são utilizados apontadores em cada nó, para que o caminho-solução seja reconstruido até o nó origem quando se atinge um nó objetivo. Uma proposição sobre a forma mais eficiente de se trabalhar com as estruturas provenientes da aplicação deste procedimento é feita por Johnson | 10 |. Uma pequena modificação no método original permite que se trabalhe também em grafos orientados ou não. O custo de transição entre estados (valores associados aos arcos que ligam os nós) pode ser positivo ou negativo (aqui outra modificação deve ser feita), porém após a publicação de um trabalho de Johnson | 9 |, observa-se que apenas com custos positivos são obtidos resultados superiores à outros processos. Johnson | 9 |, questiona a afirmativa de Elmaghraby | 4 | de que o procedimento de Dijkstra requer $3n^3$ operações para resolver o problema de determinar os menores caminhos entre um nó e todos os outros em um grafo orientado, com arcos de custo positivo e/ou negativo; é proposta uma estrutura com arcos de custo negativo e se prova que são requeridas $n2^n$ operações. As considerações feitas a seguir, estarão, por este motivo restritas ao caso de arcos

de custo positivo em grafos orientados.

Fixando um nó com raiz e estabelecendo-se como nó objetivo cada um dos n-1 restantes de cada vez, a aplicação do procedimento de Dijkstra n-1 vezes resulta na determinação dos menores caminhos entre um nó e todos os outros. Neste nível de generalização do procedimento observa-se que um tratamento iterativo é possível e vantajoso. A forma proposta por Elmaghraby | 4 | é interessante, mas como apresentada por Dreyfus | 2 | é mais clara e bastante simples:

a) atribuir ao nó raiz o custo zero e aos n-1 restantes, infinito. O custo do nó raiz é considerado permanente .

b) comparar o custo de cada nó (exceto o do raiz) com a soma do custo do nó raiz com o custo do arco direto do nó raiz ao nó em questão. Se a soma for menor, substituir o custo presente pelo valor da soma.

c) determinar qual o menor dos custos (excetuando-se os permanentes) e tomá-lo como permanente.

d) comparar o custo de cada nó restante com a soma do custo do nó feito permanente no passo anterior e o custo do arco direto do nó permanente ao nó em questão. Se a soma for menor, substituir o custo presente pelo valor da soma.

e) voltar ao passo c e repetir os passos c e d.

Com n-1 execuções do passo c, interrompe-se o processo.

Para exemplificar o procedimento, tome-se o grafo apresentado no Capítulo II onde se quer determinar os menores caminhos entre o nó 1 e os demais. Na representação a seguir, um nó de custo permanente, será representado com um traço sob o custo.

nós (i)	1	2	3	4	5	
custo (d_{1i}^k)	$\underline{0}$	∞	∞	∞	∞	K = 1
		1	∞	$\underline{0.5}$	∞	K = 2, $d_{1i}^k = \min(d_{1i}^{k-1}, d_{11} + d_{1i})$, $i \neq 1$
		$\underline{1}$	∞		1.5	K = 3, $d_{1i}^k = \min(d_{1i}^{k-1}, d_{14} + d_{4i})$, $i \neq 1$ e $i \neq 4$
			2		$\underline{1.5}$	K = 4, $d_{1i}^k = \min(d_{1i}^{k-1}, d_{12} + d_{2i})$, $i \neq 1$, $i \neq 4$ e $i \neq 2$
			$\underline{2}$			K = 5, $d_{1i}^k = \min(d_{1i}^{k-1}, d_{15} + d_{5i})$, $i \neq 1$, $i \neq 4$, $i \neq 2$ e $i \neq 5$.

Assim: $d_{12} = 1$, $d_{13} = 2$, $d_{14} = 0.5$ e $d_{15} = 1.5$ são os custos dos caminhos mínimos. Os caminhos são obtidos na seguinte ordem:

$P_{14} = 1,4$ $\qquad$ $P_{12} = 1,2$ $\qquad$ $P_{15} = 1,4,5$

$P_{13} = 1,2,3$

2. Número de operações no procedimento de Dijkstra

$\frac{n(n-1)}{2}$ adições e $\frac{n(n-1)}{2}$ comparações para tentativa de modificação dos custos.

$\frac{n(n-1)}{2}$ comparações para determinar qual custo será permanente.

Então: $\frac{3}{2}\, n(n-1)$ operações para determinar os menores caminhos entre um nó e todos os outros. Se o processo é aplicado aos n nós, tem-se $\frac{3}{2}\, n^2(n-1)$ operações. Além disso é necessário indicar os nós que vão tendo os custos considerados como permanentes. A forma sugerida como eficiente por Dreyfus | [2] |, é associar à cada nó, um índice que varia de zero à um quando o custo é tomado como permanente. Nos passos b e d anteriormente descritos, inclui-se um teste de verificação do índice associado ao nó em exame: se o índice é zero, o processo é aplicado normalmente; se o índice é um (custo permanente), examina-se o nó seguinte. Desta forma, $n(n-1)$ comparações adicionais são necessárias na aplicação do processo à cada nó; para determinar os menores caminhos entre cada par de nós, $n^2(n-1)$ comparações de índices são portanto necessárias.

O processo todo envolve então: $\frac{3}{2}\, n^2(n-1) + n^2(n-1) =$
$2.5\, n^3 - 2.5\, n^2$ operações (3.2.1)

Também $n^2(n-1)$ operações de indexação são efetuadas.

Se comparado com o procedimento de Floyd, a apresentação do procedimento de Dijkstra nessa forma é realmente menos eficiente, justificando uma indicação negativa para este último feita por Dreyfus | [2] |. A memória requerida pelo procedimento de Dijkstra é também maior que a requerida por Floyd. Mesmo que as modificações de custo sejam efetuadas diretamente sobre a matriz de custos diretos é necessário um conjunto de n elementos para o trabalho com os índices que indicam se o custo é ou não permanente em um dado instante.

3. Tratamento computacional do procedimento

Uma análise no procedimento de Dijkstra como apresentado por Dreyfus | [2] | revela dois pontos que se convenientemente tratados devem melhorar a sua eficiência.

O primeiro ponto diz respeito aos passos a e b descritos na seção 2. Resulta da aplicação do passo b, que a atribuição temporária feita no passo a, irá se reduzir aos custos apresentados pela matriz de custos diretos. Assim é desnecessário trabalhar com um conjunto auxiliar de custos, onde são efetuadas as modificações; a utilização desse comjunto de n elementos, juntamente com as n posições adicionais para os índices que indicam se o custo é ou não permanente, resulta em 2n posições de memória a mais em Dijkstra com relação a Floyd. As modificações podem ser feitas então diretamente sobre a matriz de distâncias, considerando-se inicialmente o custo do nó raiz como permanente e eliminando-se os passos a e b da descrição feita na seção 1 deste Capítulo.

O segundo ponto se relaciona com a forma de indicar qual nó tem custo permanente pela utilização de índices zero ou um como sugerido por Dreyfus | [2] |. A forma aqui proposta se revela mais eficiente, uma vez que não são necessários $n^2(n - 1)$ comparações de índices, alterando 3.2.1 de maneira significante.

É formado um vetor IND, de n posições, com IND(1) contendo o número do nó raiz; a cada vez que o custo de um nó for tomado como permanente, o seu número é deslocado para a posição seguinte ao último elemento em IND que foi tomado como permanente.

Desta posição de IND, até IND(N) estão os números dos nós de custo não permanente.

Como exemplo, seja a linha 1 da matriz D^0 do grafo apresentado no Capítulo II. O vetor associado aquela linha será:

	IND	(1)	(2)	(3)	(4)	(5)
custos da 1ª. linha		0	1	∞	0.5	∞

Com n-2 comparações determina-se que IND(4) (que contém o número do nó 4) é o que tem menor custo associado, então ele é tomado como permanente e fica-se com:

IND	(1)	(4)	(3)	(2)	(5)
	0	0.5	∞	1	∞

Aplica-se agora $d_{1i} = \min(d_{1i}, d_{ip} + d_{pi})$ para $3 \leq i \leq N$, onde p = IND(2). Fica-se com

IND	(1)	(4)	(3)	(2)	(5)
	0	0.5	∞	1	1.5

o menor é IND(4) (que contém o número de nó 2). Então:

IND	(1)	(4)	(2)	(3)	(5)
	0	0.5	1	∞	1.5

agora $d_{1i} = \min(d_{1i}, d_{1p} + d_{pi})$ para $4 \leq i \leq N$, onde p = IND(3).

Fica-se com

IND	(1)	(4)	(2)	(3)	(5)
	0	0.5	1	2	1.5

e o menor é IND(5) (que contém o número do nó 5). Então:

IND	(1)	(4)	(2)	(5)	(3)
	0	0.5	1	1.5	2

agora $d_{1i} = \min(d_{1i}, d_{1p} + d_{pi})$ para i=N e p=IND(4).

Neste ponto o processo pode ser interrompido.

O número de operações com estas duas modificações é:

$\frac{(n-1)(n-2)}{2}$ adições e $\frac{(n-1)(n-2)}{2}$ comparações para tentativa de modificação dos custos.

$\frac{(n-1)(n-2)}{2}$ comparações para determinar o menor custo.

Então: $\frac{3}{2}(n-1)(n-2)$ para determinar os menores caminhos entre um nó e todos os outros. A aplicação do procedimento aos n nós resulta em

$1.5\ n^3 - 4.5\ n^2 + 3n$ operações (3.3.1).

O número de operações de indexação está em torno de $\frac{n(n-1)(n-2)}{2}$ para todo o processo. Observe-se que este último número, mesmo se somado à 3.3.1, ainda resulta em um total que é inferior aos $2n^3$ do procedimento de Floyd.

Na tabela apresentada a seguir, indica-se o tempo gasto em processamento sòmente para alteração da matriz de distâncias diretas até se obter a matriz de custos mínimos. Êste tempo não envolve leitura ou impressão de dados, ou chamada de qualquer função ou sub-rotina; os procedimentos comparados aplicam-se sempre ao mesmo grafo, que é gerado pelos próprios programas com características que serão vistas no Capítulo IV. Aqui empregou-se:

FLOYD - procedimento de Floyd modificações.

DIJKSTRA - procedimento de Dijkstra com modificações.

Os sitemas utilizados são o IBM-1130 e o B-6700; a unidade de tempo é o segundo. Os programas estão listados no Apêndice I, partes A e B para o primeiro procedimento e no Apêndice II, para o segundo procedimento.

T A B E L A 3

Número de	IBM - 1130		B - 6700	
Nós	FLOYD	DIJKSTRA	FLOYD	DIJKSTRA
3	0.044	0.022	0.002	0.001
6	0.302	0.212	0.017	0.011
9	1.007	0.739	0.058	0.041
12	2.385	1.791	0.144	0.097
15	4.670	3.505	0.272	0.186
18	8.007	6.059	0.454	0.316
21	12.655	9.609	0.730	0.517
24	18.883	14.369	1.055	0.746
27	26.767	20.395	1.541	1.073
30	36.702	27.921	2.089	1.469
33	48.854	37.217	2.805	1.923
36	63.246	48.215	3.601	2.477
39	81.972	61.331	4.474	3.148
42	100.195	76.361	5.786	4.057
45	124.062	93.911	6.974	4.897

48	144.526	113.825	8.333	5.999
51	178.561	136.382	10.234	7.143
54	211.859	161.839	11.677	8.207
57	248.919	190.175	13.520	9.641
60	290.147	221.793	15.695	11.212
63	335.619	256.513	19.041	13.006
66	385.548	294.817	22.452	14.917
69	440.081	336.447	24.769	17.271
72	499.531	382.323	26.940	19.605
75	564.289	431.558	30.085	23.267
78	634.547	485.486	34.038	25.963
81	710.382	543.446	38.068	27.627
84	792.097	605.763	42.391	31.884
87	880.969	672.705	47.858	34.553
90			54.121	38.621
93			60.997	43.553
96			65.916	48.224

4. Conclusões

Na tabela 3, observa-se que a menos de uma flutuação nas medidas feitas em grafos de pequeno porte devido à imprecisão dos mecanismos de determinação do tempo, já para quinze nós tem-se um ganho superior à vinte por cento em Dijkstra. Para noventa nós, o ganho já está bem próximo à vinte e cinco por cento e quase estabilizado.

As modificações propostas no procedimento de Dijkstra, são realmente significantes, embora tenha-se reduzido o campo de aplicação aos grafos com arcos de custo positivo. Além disso a memória requerida é um pouco maior (mais um vetor de n posições) e a complexidade do programa também aumenta.

Essas "desvantagens" em Dijkstra são compensadas pelo fato de se ter um procedimento que se presta à solução de dois problemas:

a - determinação do menor caminho entre todos os pares

b - determinação do menor caminho entre um nó e todos os outros,

e também por se poder, em Dijkstra, trabalhar com a matriz original em arquivo e em cada passo operar com apenas uma linha na memória central. Assim o que se faz é resolver n problemas do tipo b acima. Embora êsse tratamento também possa ser dispensado ao procedimento de Floyd, ele é mais oneroso em virtude de ser necessário transferir n vezes cada linha da matriz original em arquivo para a memória central. Para grafos de médio e grande porte a "vantagem" do procedimento de Dijkstra deve se acentuar ao ganho de tempo na operação com arquivos.

C A P Í T U L O IV

1. Geração dos Grafos

Para comparação dos tempos gastos pelos procedimentos apresentados nos dois capítulos anteriores, o mesmo conjunto de grafos foi gerado em cada programa.

Tomou-se o valor 3 como razoável para o incremento na dimensão dos grafos. Assim o processo é gerar um grafo de n nós (inicialmente n toma o valor 3) e aplicar o procedimento; gerar novo grafo com n + 3 nós e aplicar novamente o procedimento continuando-se desta forma até se esgotar o tempo solicitado ao sistema computacional.

Como se estuda sempre o caso de grafos orientados, é necessário para cada grafo de n nós, gerar uma matriz D(n x n). Essa matriz tem tres tipos de elementos:

I - elementos da diagonal principal iguais à zero.

II- elementos com um valor igual ao custo do arco ligando dois nós.

III-elementos iguais à "infinito", o que significa não haver arco ligando dois nós.

Para os elementos do tipo II, optou-se (sem nenhum motivo particular) pela faixa dos reais de um à dez. Para os tipo III, é suficiente fazer para cada grafo, cada um destes elementos tomar o valor nx10; isto porque habendo no máximo n - 1 arcos ligando dois nós e o custo de cada arco sendo no máximo igual a dez, aquele valor é superior ao custo de qualquer caminho que se venha a obter.

Para atribuir um custo real ou infinito à cada arco, o processo é:

I - gerar um número aleatório entre zero e um.

II - se o número é inferior à 0.5, atribuir "infinito" (n x 10) ao elemento em questão, caso contrário

III- gerar números aleatórios até se obter um igual ou superior à 0.1. Quando isto acontecer, multiplica-lo por dez e atribuir este valor ao elemento em questão.

IV - repetir a partir de I, até que a matriz D(n x n) esteja completa.

A geração de aleatórios se fez no sistema B-6700 pela chamada da função RANDOM. No sistema IBM-1130 foi necessário introduzir uma rotina de geração (ver programas dos apêndices). Também para êsse sistema, embora fossem gerados números aleatórios entre zero e um, os valores atribuidos aos elementos da matriz de distâncias eram sempre inteiros; isto porque o processamento de quantidades reais no IBM-1130 é feito por software e este fato ocasiona diferenças significantes nos tempos obtidos em cada procedimento, além de aumentar em muito o tempo global de utilização da máquina.

Na tabela apresentada a seguir, mostra-se o número de arcos que foram considerados como "infinito" (n x10), para cada grafo pelos sistemas B-6700 e IBM-1130.

T A B E L A 4

Nº de nós	B-6700	IBM-1130
3	2	3
6	12	16
9	33	36

12	64	63
15	112	108
18	161	146
21	219	205
24	274	263
27	337	340
30	428	420
33	502	514
36	602	607
39	720	743
42	842	869
45	974	1003
48	1106	1148
51	1256	1290
54	1408	1437
57	1596	1616
60	1769	1800
63	1979	1973
66	2168	2152
69	2374	2351
72	2592	2551
75	2827	2768
78	3071	2980
81	3296	3241
84	3545	3499
87	3799	3771

90	4061
93	4343
96	4630

2. Obtenção de tempos

Em todos os programas o tempo medido relaciona-se apenas com as modificações da matriz de distancias originais até ser obtida a matriz de custos mínimos, não estando envolvida nenhuma leitura ou impressão ou chamadas de funções ou subrotinas. Também em nenhum dos programas apresentados é feita obtenção dos caminhos mínimos mas tão sòmente do custo.

No sistema IBM-1130 a medição é feita pela chamada de rotinas que trabalham com pulsos da impressora 1132. A unidade é o milisegundo e em tempos superiores à dez segundos o êrro é inferior à 1%.

No sistema B-6700, utilizou-se uma rotina que fornece o tempo de processamento em unidades de 2.4 microsegundos. Este sistema trabalha com partições de memória e para se obter tempos com êrro inferior à 5% em diferentes tomadas é necessário a utilização de uma opção do compilador que aloca toda a matriz sem segmentação. Sem a utilização dessa opção, o tempo para diferentes tomadas é totalmente flutuante, não possibilitando nenhuma comparação.

3. Extensões

Uma complementação direta do presente trabalho, é a já citada na última seção do capítulo anterior. Sendo imprescindível o trabalho com arquivos para grafos de grande porte, a determinação da variação de tempo gas

to pelos dois processos com a inclusão das operações de transferência arquivo-memória central é de especial interêsse para o usuário.

Como os resultados aqui obtidos, dizem respeito à diferentes tratamentos em têrmos de programação para procedimentos já conhecidos, um trabalho interessante é atacar outros procedimentos para determinação de caminhos mínimos, visando torná-los mais eficientes apenas por um mais bem estudado tratamento computacional. É possível que procedimentos já abandonados sejam na realidade melhores que outros usualmente considerados como mais eficientes,bastando utilizar naqueles alguns detalhes de programação possíveis em modernos sistemas computacionais.

Finalmente, uma extensão natural deste estudo em que se mostra ser o procedimento de Dijkstra mais eficiente que o procedimento de Floyd, é a verificação do comportamento do procedimento A* como apresentado por Hart,Nilsson e Raphael | [8] | e por Nilsson | [13] |. Nestas referências prova-se que para o problema de determinação do menor caminho entre dois pontos, o procedimento A* é mais eficiente que o procedimento de Dijkstra; dispondo-se então de uma informação adicional sôbre o grafo, que permita o estabelecimento de uma "função de avaliação", é aparentemente possível estabelecer regras para a aplicação eficiente do procedimento A* a cada par do grafo(n^2 aplicações) e determinar então se para esse problema(de determinação do menor custo entre cada par de vértices de um grafo orientado com arcos de custo positivo) a eficiência ainda é superior à do procedimento de Dijkstra.

Observação especial - Este trabalho estava terminado, quando por uma referência de Grassim e Minoux | [7] |descobriu-se o trabalho de Yen | [15] | e posteriormente Williams e White | [18] | e ainda Yen | [16] |.

A idéia desenvolvida por Yen em | [15] | e | [16] | é a mesma aqui

apresentada, embora as modificações não sejam feitas sôbre a matriz original de custos e sim sôbre um vetor auxiliar. O procedimento de Yen utiliza portanto um vetor adicional (n posições a mais de memória) e diz o autor ter obtido resultados próximos ao teoricamente esperado.

REFERÊNCIAS

[1] Bilde, Ole and Krarup, Jakob, "A Modified Cascade Algorithm for Shorstest Paths (C)". METRA, vol. VIII, nº 2, p.231-241 (1969).

[2] Dreyfus, Stuart E., "An Appraisal of Some Shortest-Path Algorithms" . OPERATIONS RESEARCH, vol. 17, nº 3, p.395-412 (Maio-1969).

[3] Edmonds, Jack and Karp, Richard M., "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". JOURNAL OF THE ACM, vol. 19, nº 2, p.248-264 (abril - 1972).

[4] Elmaghraby, Salah E., "The Theory of Networks and Management Science". Part I . MANAGEMENT SCIENCE, vol. 17, nº 1, p.1-34 (Setembro - 1970).

[5] Farbey, B.A., Land, A.H. and Murchland, J.D., "The Cascade Algorithm For Finding All Shortest Distances In a Directed Graph". MANAGEMENT SCIENCE, vol. 14, nº 1, p.19-28 (Setembro - 1967).

[6] Floyd, Robert W., "Algorithm 97 - Shortest Path". COMMUNICATIONS OF THE ACM, vol. 5, nº 6, p. 345 (Junho - 1962).

[7] Grassin, J. et Minoux, M., "Variations Sur Un Algorithme De Dantzig - Application A La Recherche Des Plus Courts Chemins Dans Les Grands Reseaux". R.A.I.R.O., vol. 1, p. 53-61 (Março - 1973).

[8] Hart, Peter E., Nilsson, Nils J. and Raphael, Bertram, "A Formal Basis For The Heuristic Determination of Minimum Cost Paths". IEE TRANSACTIONS OF S.S.C., vol. SSC-4, nº 2, p. 100-107 (Julho-1968).

[9] Johnson, Donald B., "A Note on Dijkstra's Shortest Path Algorithm". JOURNAL OF THE ACM, vol. 20, nº 3, p. 385-388, (Julho - 1973).

[10] Johnson, Ellis L., "On Shortest Paths and Sorting". IBM - RC 3691 (16719) - MATHEMATICS (Janeiro - 1972).

[11] Knuth, Donald E., "The Art Of Computer Programming", vol. 1, "Fundamental Algorithms". ADDISON WESLEY P.C. (1968)

[12] Minieka, Edward, "On Computing Sets of Shortest Paths In a Graph". COMMUNICATIONS OF THE ACM, vol. 17, nº 6, p. 351-353 (Junho - 1974).

[13] Nilsson, Nils J., "Problem - Solving Methods In Artificial Intelligence". McGRAW-HILL, Inc. (1971).

[14] Roy, B., "Algébre Moderne et Théorie des Graphes" - Tome 2 - "Applications et Problèmes Specifiques". Fascicule 2. DUNOD (1970).

[15] Yen, Jin Y., "Finding The Lengths of All Shortest Paths In N-Node Nonnegative - Distance Complete Network Using $\frac{1}{2} N^3$ Additions and N^3 Comparisons". JOURNAL OF THE ACM, vol. 19, nº 3, p. 423-424, (Julho - 1972).

| 16 | _____ ,"Reply to Williams and White's Note". JOURNAL OF THE ACM., vol. 20, nº 3, p. 390 (Julho - 1973).

| 17 | Whiting, P.D. and Hillier, J.A., "A Method For Finding the Shortest Route Through A Road Network". OPERATIONAL RESEARCH QUARTERLY, vol. 11, nº 1/2, p. 37-40 (1960).

| 18 | Williams, Thomas A., and White , Gregory P., "A Note On Yen's Algorithm for Finding The Length of All Shortest Paths In N-Node Nonnegative - Distance Networks". JOURNAL OF THE ACM, vol. 20, nº 3, p. 389-390, (Julho - 1973)

OBS : Referências | 3 |, | 11 | e | 14 | não são citadas no texto.

A P E N D I C E I

A

FLOYD - Procedimento de Floyd sem modificações (IBM-1130)

```
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE ALEA(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767
      IX=IY
      RETURN
      END
// DUP
*STORE      WS  UA  ALEA
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE RAND
      INTEGER D(90,90)
      COMMON N,D,NAM4
      NAM4=0
      IX=32767
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
    9 CALL ALEA(IX,IY,YFL)
      IF(YFL-.5)4,4,3
    4 IF(I-J)8,9,8
    8 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    3 CALL ALEA(IX,IY,YFL)
      IF(YFL-.1)3,3,11
   11 D(I,J)=YFL*10.
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
// DUP
*STORE      WS  UA  RAND
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(2501READER,1403PRINTER,DISK)
C PROGRAMA PRINCIPAL FLOYD SEM MODIFICACOES
      INTEGER D(90,90),DL
      COMMON N,D,NAM4
      N=3
   10 CALL RAND
      CALL START
      DO 1 K=1,N
      DO 1 I=1,N
      DO 1 J=1,N
      DL=D(I,K)+D(K,J)
```

```
      IF(D(I,J)-DL)1,1,2
    2 D(I,J)=DL
    1 CONTINUE
      CALL TIME(X)
      TEMPO=(X/1000.)
      WRITE(5,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.3,' SEGUNDOS',/)
      IF(N-90)80,90,90
   80 N=N+3
      GO TO 10
   90 CALL PARE
      CALL EXIT
      END
// XEQ
```


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

B

FLOYD - Procedimento de Floyd sem mòdificações (B-6700)

```
 CCMPILE FLOYD FORTRAN DATA
C SUBROTINA PARA GERAR ELEMENTOS DA MATRIZ DE DISTANCIAS
$SET LCNG
      SUBROUTINE RAND
      COMMCN N,D(99,99),NAM4
      NAM4=0
      IN=2**38
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
      XT=RANDOM(IN)
      IF(XT-.5)3,3,8
    3 IF(I-J)4,2,4
    4 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    8 XT=RANDOM(IN)*10.
      IF(XT-1.0)8,8,9
    9 D(I,J)=XT
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
C ESTE PROGRAMA TEM O LONG
$SET LCNG
C PROGRAMA PRINCIPAL FLOYD SEM MODIFICACOES
      REAL MENOR
      COMMCN N,D(99,99),NAM4
      N=3
   10 CALL RAND
      TEMPO=TIME(12)
      DO 1 K=1,N
      DO 1 I=1,N
      DO 1 J=1,N
      DU=D(I,K)+D(K,J)
      IF(D(I,J)-DU)1,1,2
    2 D(I,J)=DU
    1 CONTINUE
      TEMPO=(TIME(12)-TEMPO)*2.4/10**6
      WRITE(6,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.6,' SEGUNDOS',/)
      IF(N-99)80,90,90
   80 N=N+3
      GO TO 10
   90 STOP
      END
  END JOB.
```


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

C

F.E.L.C.C. - Procedimento de Floyd evitando linhas e colunas por comparações (IMB-1130)

```
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE ALEA(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767
      IX=IY
      RETURN
      END
// DUP
*STORE      WS  UA  ALEA
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE RAND
      INTEGER D(90,90)
      COMMON N,D,NAM4
      NAM4=0
      IX=32767
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
    9 CALL ALEA(IX,IY,YFL)
      IF(YFL-.5)4,4,3
    4 IF(I-J)8,9,8
    8 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    3 CALL ALEA(IX,IY,YFL)
      IF(YFL-.1)3,3,11
   11 D(I,J)=YFL*10.
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
// DUP
*STORE      WS  UA  RAND
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(2501READER,1403PRINTER,DISK)
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS E COLUNAS POR
C COMPARACOES
      INTEGER D(90,90),DL
      COMMON N,D,NAM4
      N=3.
   10 CALL RAND
      CALL START
      DO 1 K=1,N
      DO 2 I=1,N
      IF(I-K)11,2,11
```

```
      11 DO 12 J=1,N
         IF(J-K)13,12,13
      13 DU=D(I,K)+D(K,J)
         IF(D(I,J)-DU)12,12,14
      14 D(I,J)=DU
      12 CONTINUE
       2 CONTINUE
       1 CONTINUE
         CALL TIME(X)
         TEMPO=(X/1000.)
         WRITE(5,70)N,NAM4,TEMPO
      70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
        *=',I4,/,' TEMPO=',F10.3,' SEGUNDOS',/)
         IF(N-90)80,90,90
      80 N=N+3
         GO TO 10
      90 CALL PARE
         CALL EXIT
         END
// XEQ
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

D

F.E.L.C.C. - Procedimento de Floyd evitando linhas e colunas por comparações (B-6700)

```
 COMPILE FLOYD FORTRAN DATA
C SUBROTINA PARA GERAR ELEMENTOS DA MATRIZ DE DISTANCIAS
$SET LCNG
      SUBROUTINE RAND
      COMMON N,D(99,99),NAM4
      NAM4=0
      IN=2**38
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
      XT=RANDOM(IN)
      IF(XT-.5)3,3,8
    3 IF(I-J)4,2,4
    4 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    8 XT=RANDOM(IN)*10.
      IF(XT-1.0)8,8,9
    9 D(I,J)=XT
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
C ESTE PROGRAMA TEM O LONG
$SET LCNG
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS E COLUNAS POR
C COMPARACOES
      REAL MENOR
      COMMON N,D(99,99),NAM4
      N=3
   10 CALL RAND
      TEMPO=TIME(12)
      DO 1 K=1,N
      DO 2 I=1,N
      IF(I-K)11,2,11
   11 DO 12 J=1,N
      IF(J-K)13,12,13
   13 DU=D(I,K)+D(K,J)
      IF(D(I,J)-DU)12,12,14
   14 D(I,J)=DU
   12 CONTINUE
    2 CONTINUE
    1 CONTINUE
      TEMPO=(TIME(12)-TEMPO)*2.4/10**6
      WRITE(6,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.6,' SEGUNDOS',/)
      IF(N-99)80,90,90
   80 N=N+3
      GO TO 10
   90 STOP
      END
  END JOB.
```

E

F.E.L.C.D. - Procedimento de Floyd evitando linhas, colunas e diagonal (IBM-1130)

```
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE ALEA(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767
      IX=IY
      RETURN
      END
// DUP
*STORE      WS  UA  ALEA
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE RAND
      INTEGER D(90,90)
      COMMON N,D,NAM4
      NAM4=0
      IX=32767
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
    9 CALL ALEA(IX,IY,YFL)
      IF(YFL-.5)4,4,3
    4 IF(I-J)8,9,8
    8 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    3 CALL ALEA(IX,IY,YFL)
      IF(YFL-.1)3,3,11
   11 D(I,J)=YFL*10.
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
// DUP
*STORE      WS  UA  RAND
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(2501READER,1403PRINTER,DISK)
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS, COLUNAS E DIAGONAL
      INTEGER D(90,90),DL
      COMMON N,D,NAM4
      N=3
   10 CALL RAND
      CALL START
      DO 1 K=1,N
      DO 2 I=1,N
      IF(I-K)11,2,11
   11 DO 12 J=1,N
```

```
      IF(J-K)13,12,13
   13 IF(J-I)15,12,15
   15 CU=C(I,K)+C(K,J)
      IF(C(I,J)-CU)12,12,14
   14 C(I,J)=CU
   12 CONTINUE
    2 CONTINUE
    1 CONTINUE
      CALL TIME(X)
      TEMPC=(X/1CCC.)
      WRITE(5,7C)N,NAM4,TEMPC
   7C FORMAT(' NUMERC DE NOS =',I2,/,' NLMERO DE ARCCS IGUAIS A INFINITO
     *=',I4,/,' TEMPC=',F1C.3,' SEGLNDOS',/)
      IF(N-9C)8C,9C,9C
   8C N=N+3
      GC TC 1C
   9C CALL PARE
      CALL EXIT
      END
// XEG
```


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
UFRJ

F

F.E.L.C.D. - Procedimento de Floyd evitando linhas, colunas e diagonal (B-6700)

```
 COMPILE FLOYD FORTRAN DATA
C SUBROTINA PARA GERAR ELEMENTOS DA MATRIZ DE DISTANCIAS
$SET LONG
      SUBROUTINE RAND
      COMMON N,D(99,99),NAM4
      NAM4=0
      IN=2**38
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
      XT=RANDOM(IN)
      IF(XT-.5)3,3,8
    3 IF(I-J)4,2,4
    4 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    8 XT=RANDOM(IN)*10.
      IF(XT-1.0)8,8,9
    9 D(I,J)=XT
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
C ESTE PROGRAMA TEM O LONG
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS, COLUNAS E DIAGONAL
$SET LONG
      REAL MENOR
      COMMON N,D(99,99),NAM4
      N=3
   10 CALL RAND
      TEMPO=TIME(12)
      DO 1 K=1,N
      DO 2 I=1,N
      IF(I-K)11,2,11
   11 DO 12 J=1,N
      IF(J-K)13,12,13
   13 IF(J-I)15,12,15
   15 DU=D(I,K)+D(K,J)
      IF(D(I,J)-DU)12,12,14
   14 D(I,J)=DU
   12 CONTINUE
    2 CONTINUE
    1 CONTINUE
      TEMPO=(TIME(12)-TEMPO)*2.4/10**6
      WRITE(6,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.6,' SEGUNDOS',/)
      IF(N-99)80,90,90
   80 N=N+3
      GO TO 10
   90 STOP
      END
  END JOB.
```

G

F.E.L.C.V. - Procedimento de FLOYD evitando linhas e colunas por vetor (IBM-1130)

```
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE ALEA(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767
      IX=IY
      RETURN
      END
// DUP
*STORE      WS  UA  ALEA
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE RAND
      INTEGER D(90,90)
      COMMON N,D,NAM4
      NAM4=0
      IX=32767
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
    9 CALL ALEA(IX,IY,YFL)
      IF(YFL-.5)4,4,3
    4 IF(I-J)8,9,8
    8 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    3 CALL ALEA(IX,IY,YFL)
      IF(YFL-.1)3,3,11
   11 D(I,J)=YFL*10.
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
// DUP
*STORE      WS  UA  RAND
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(2501READER,1403PRINTER,DISK)
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS E COLUNAS POR VETOR
      INTEGER D(90,90),DU
      DIMENSION IND(90)
      COMMON N,D,NAM4
      N=3
   10 CALL RAND
      CALL START
      DO 11 I=1,N
   11 IND(I)=I
      DO 1 K=1,N
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
NCE UFRJ

```
      INC(K)=INC(1)
      INC(1)=K
      DO 1 I=2,N
      II=INC(I)
      DO 1 J=2,N
      JJ=INC(J)
      DU=D(II,K)+D(K,JJ)
      IF(D(II,JJ)-DU)1,1,2
    2 D(II,JJ)=DU
    1 CONTINUE
      CALL TIME(X)
      TEMPO=(X/1000.)
      WRITE(5,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.3,' SEGUNDOS',/)
      IF(N-90)80,90,90
   80 N=N+3
      GO TO 10
   90 CALL PARE
      CALL EXIT
      END
// XEQ
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

H

F.E.L.C.V. - Procedimento de FLOYD evitando linhas e colunas por vetor (B-6700)

```
 COMPILE FLOYD FORTRAN DATA
C SUBROTINA PARA GERAR ELEMENTOS DA MATRIZ DE DISTANCIAS
$SET LONG
      SUBROUTINE RAND
      COMMON N,D(99,99),NAM4
      NAM4=0
      IN=2**38
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
      XT=RANDOM(IN)
      IF(XT-.5)3,3,8
    3 IF(I-J)4,2,4
    4 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    8 XT=RANDOM(IN)*10.
      IF(XT-1.0)8,8,9
    9 D(I,J)=XT
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
C ESTE PROGRAMA TEM O LONG
$SET LONG
C PROGRAMA PRINCIPAL FLOYD EVITANDO LINHAS E COLUNAS POR VETOR
      REAL MENOR
      DIMENSION IND(99)
      COMMON N,D(99,99),NAM4
      N=3
   10 CALL RAND
      TEMPO=TIME(12)
      DO 11 I=1,N
   11 IND(I)=I
      DO 1 K=1,N
      IND(K)=IND(1)
      IND(1)=K
      DO 1 I=2,N
      II=IND(I)
      DO 1 J=2,N
      JJ=IND(J)
      DU=D(II,K)+D(K,JJ)
      IF(D(II,JJ)-DU)1,1,2
    2 D(II,JJ)=DU
    1 CONTINUE
      TEMPO=(TIME(12)-TEMPO)*2.4/10**6
      WRITE(6,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.6,' SEGUNDOS',/)
      IF(N-99)80,90,90
   80 N=N+3
      GO TO 10
   90 STOP
      END
   END JOB.
```

A P E N D I C E II

A

DIJKSTRA - Procedimento de Dijkstra com modificações (IBM-1130)

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

```
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE ALEA(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767
      IX=IY
      RETURN
      END
// DUP
*STORE      WS  UA  ALEA
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
      SUBROUTINE RAND
      INTEGER D(90,90)
      COMMON N,D,NAM4
      NAM4=0
      IX=32767
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
    9 CALL ALEA(IX,IY,YFL)
      IF(YFL-.5)4,4,3
    4 IF(I-J)8,9,8
    8 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    3 CALL ALEA(IX,IY,YFL)
      IF(YFL-.1)3,3,11
   11 D(I,J)=YFL*10.
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
// DUP
*STORE      WS  UA  RAND
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(2501READER,1403PRINTER,DISK)
C  METODO DIJKSTRA MODIFICADO
      INTEGER D(90,90),DL
      DIMENSION IND(90)
      COMMON N,D,NAM4
      N=3
   10 CALL RAND
      CALL START
      DO 1 I=1,N
      DO 2 M=2,N
    2 IND(M)=M
```

```
      INC(I)=1
      IJ=INC(2)
      MENCR=C(I,IJ)
      IK=2
      CC 3 J=3,N
      IJ=INC(J)
      IF(C(I,IJ)-MENCR)4,3,3
    4 MENCR=C(I,IJ)
      IK=J
    3 CONTINUE
      N1=N-1
      CC 1 JF=2,N1
      JJ=INC(IK)
      INC(IK)=INC(JF)
      IK=JF+1
      IJ=INC(IK)
      MENCR=C(I,IJ)
      CC 9 K=IK,N
      II=INC(K)
      CU=C(I,JJ)+C(JJ,II)
      IF(C(I,II)-CU)5,5,6
    6 C(I,II)=CU
    5 IF(C(I,II)-MENCR)7,9,9
    7 MENCR=C(I,II)
      IK=K
    9 CONTINUE
    1 CONTINUE
      CALL TIME(X)
      TEMPC=(X/1CCC.)
      WRITE(5,70)N,NAM4,TEMPC
   70 FORMAT(' NUMERC DE NOS =',I2,/,' NUMERC DE ARCCS IGUAIS A INFINITO
     *=',I4,/,' TEMPC=',F10.3,' SEGUNDCS',/)
      IF(N-9C)8C,9C,9C
   8C N=N+3
      GC TC 1C
   9C CALL PARE
      CALL EXIT
      END
// XEC
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

B

DIJKSTRA - Procedimento de Dijkstra com modificações(B-6700)

```
 COMPILE FLOYD FORTRAN DATA
C SUBRCTINA PARA GERAR ELEMENTOS DA MATRIZ DE DISTANCIAS
$SET LONG
      SUBRCUTINE RAND
      COMMON N,D(99,99),NAM4
      NAM4=0
      IN=2**38
      SUP=N*10.
      DO 1 I=1,N
      DO 2 J=1,N
      XT=RANDOM(IN)
      IF(XT-.5)3,3,8
    3 IF(I-J)4,2,4
    4 D(I,J)=SUP
      NAM4=NAM4+1
      GO TO 2
    8 XT=RANDOM(IN)*10.
      IF(XT-1.0)8,8,9
    9 D(I,J)=XT
    2 CONTINUE
    1 D(I,I)=0.0
      RETURN
      END
C ESTE PROGRAMA TEM O LONG
$SET LONG
C METODO DIJKSTRA MODIFICADO
      REAL MENOR
      DIMENSION IND(99)
      COMMON N,D(99,99),NAM4
      N=3
   10 CALL RAND
      TEMPO=TIME(12)
      DO 1 I=1,N
      DO 2 M=2,N
    2 IND(M)=M
      IND(I)=1
      IJ=IND(2)
      MENOR=D(I,IJ)
      IK=2
      DO 3 J=3,N
      IJ=IND(J)
      IF(D(I,IJ)-MENOR)4,3,3
    4 MENOR=D(I,IJ)
      IK=J
    3 CONTINUE
      N1=N-1
      DO 1 JF=2,N1
      JJ=IND(IK)
      IND(IK)=IND(JF)
      IK=JF+1
      IJ=IND(IK)
      MENOR=D(I,IJ)
      DO 9 K=IK,N
      II=IND(K)
```


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
UFRJ NCE

```
      DU=D(I,JJ)+D(JJ,II)
      IF(D(I,II)-DU)5,5,6
    6 D(I,II)=DU
    5 IF(D(I,II)-MENOR)7,9,9
    7 MENOR=D(I,II)
      IK=K
    9 CONTINUE
    1 CONTINUE
      TEMPO=(TIME(12)-TEMPO)*2.4/10**6
      WRITE(6,70)N,NAM4,TEMPO
   70 FORMAT(' NUMERO DE NOS =',I2,/,' NUMERO DE ARCOS IGUAIS A INFINITO
     *=',I4,/,' TEMPO=',F10.6,' SEGUNDOS',/)
      IF(N-99)80,90,90
   80 N=N+3
      GO TO 10
   90 STOP
      END
  END JCB.
```


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
UFRJ