SISTEMA DE ORDENAÇÃO
PARA O
TERMINAL INTELIGENTE

IZIO AJDELSZTAJN

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 CIÊNCIAS (M.Sc.)

Aprovada por :

Presidente

RIO DE JANEIRO ,RJ - BRASIL

JULHO DE 1979

AJDELSZTAJN, IZIO

Sistema de Ordenação Para o Terminal Inteligente (Rio de Janeiro) 1979.

XII, 85p. 29,7 cm (COPPE-UFRJ, M.Sc, Sistemas, 1979).

Tese-Univ.Fed.Rio de Janeiro Fac.Engenharia.

1. Pesquisa e implementação de um sistema de ordenação para um micro - computador.I.COPPE/URFJ.II.Título(série).

D E D I C A T Ó R I A

À minha esposa,

Doris.

A G R A D E C I M E N T O S

A GUILHERME CHAGAS RODRIGUES pela orientação de todo o trabalho, sem a qual seria impossível realizá-lo.

A JOSÉ ANTONIO DOS SANTOS BORGES e a PAULO CESAR MO - RAES MELLO; pelo apoio ao uso do Terminal Inteligente e ao desenvolvimento e a implantação do sistema.

A IVAN DA COSTA MARQUES e SUELI MENDES DOS SANTOS, pela participação na banca examinadora.

Muito Obrigado

RESUMO

Este trabalho descreve um sistema genérico de ordenação para o Terminal Inteligente, microcomputador projetado e construído no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro.

São apresentadas definições, algorítimos, forma de utilização, características do desenvolvimento e,finalmente, uma análise de desempenho do sistema e respectivas conclusões.

O produto final obtido é um sistema de ordenação cuja dependência do tempo de execução com o número de registros a ordenar é aproximadamente linear no intervalo de interesse.

A B S T R A C T

This work describes a sort generalized system for the "Intelligent Terminal", a microcomputer designed and built in the Electronic Computing Nucleus of the Rio de Janeiro Federal University.

The following subjects are presented: definitions, algorithms, utilization format, development characteristics and a performance analysis with related conclusions.

The final product obtained is a sort system in which the execution time is approximately linear with respect to the number of records to be sorted in the interest interval.

Í N D I C E

CAPÍTULO I

INTRODUÇÃO

I. INTRODUÇÃO

1. Objetivo

Sendo a ordenação uma técnica bastante utilizada quando se lida com qualquer quantidade considerável de dados, um sistema de computação não pode prescindir de um software de ordenação, pois este é indispensável em uma larga gama de aplicações.

Esta real necessidade inspirou o presente trabalho : realizar um Sistema de ORdenação para o Terminal Inteligente:

SORTI

2. O Terminal Inteligente :

O Terminal Inteligente-TI- é um microcomputador projetado e construído nos laboratórios do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro- NCE / UFRJ, baseado no microprocessador INTEL-8008.

O TI possui até 16K bytes de memória, um teclado e um vídeo e pode funcionar acoplado a um processador central.

Dentre os periféricos que podem ser conectados ao TI, temos: discos magnéticos, impressora, leitora de cartões , unidades cassete e diskette e terminais.

O software do TI consiste basicamente no Sistema Operacional em disCO (SOCO) e um compilador/interpretador de linguagem de alto nível (PLTI).

Por ser um sistema operacional para um terminal inteligente, o SOCO é por definição interativo. Todo o diálogo com o operador se faz através de questões propostas no vídeo do TI.

A linguagem PLTI é baseada na linguagem PLM originalmente criada para utilização com o microprocessador INTEL 8008. Possui as estruturas normais das atuais linguagens de alto nível e é na realidade bastante semelhante do PL/I.

3. Restrições para o Desenvolvimento :

O SORTI deve ser, tanto quanto possível, independente do hardware do TI, de forma a ser compatível com futuras versões do equipamento.Não há vantagem em construir um software de máxima eficiência sujeito a tornar-se inútil como conseqüência de modificações efetuadas no hardware do TI.

Desta forma, o SORTI é escrito em linguagem de alto nível — PLTI e os acessos a periféricos são regidos pelas regras da linguagem e do sistema operacional SOCO.

O fornecimento de parâmetros para o SORTI deve ser feito de forma conversacional, mantendo a filosofia do SOCO . Existe também uma opção não-conversacional.

Sendo o TI uma máquina de pequeno porte, não deve ser utilizada para controlar arquivos de tamanho exagerado .

O SORTI é feito então para funcionar de forma ideal com arquivos de tamanho médio, digamos até cinco mil regis - tros.

4. Características Básicas:

O SORTI é um utilitário de ordenação. Como tal, será capaz de ordenar qualquer arquivo do TI.

Para cada caso específico, o usuário deverá informar os parâmetros necessários à ordenação do arquivo, tais como tamanho do registro, características da chave de ordenação, etc.

Considerando a importância do disco magnético como principal unidade de armazenagem de dados do TI, o SORTI é basicamente orientado para a ordenação de arquivos em disco. Isto não impede porém o uso do SORTI para ordenar arquivos contidos em outros periféricos, uma vez que tais arquivos podem ser movidos para disco com o uso de um utilitário simples.

Por motivos de eficiência, então, os arquivos de entrada, saída e as áreas de trabalho estarão obrigatoriamente em disco.

Desta forma, a configuração mínima para o uso do SORTI é o TI com vídeo, teclado e uma unidade de disco.

5. Um Programa de Ordenação:

Ao analisarmos o problema da construção de um programa de ordenação, a primeira dificuldade que encontramos está no número de registros do arquivo a ordenar.

Quando todos esses registros podem ser lidos e colocados na memória do computador simultaneamente, basta gravar tais registros num arquivo de saída de forma ordenada. Este processo de colocar em ordem um grupo de registros que se encontra na memória, é denominado ordenação interna.

Na maior parte dos casos que ocorrem na prática porém,não é possível colocar de uma só vez todo o arquivo na memória.Considerando o pequeno tamanho da memória do TI, podemos até afirmar que raramente será possível ordenar um arquivo inteiro com uma única ordenação interna.

Para ordenar arquivo de tamanho maior do que a área de memória disponível, colocamos o maior número de registros simultaneamente na memória, ordenando-os e gravando-os em disco. Repetindo tal procedimento até o fim do arquivo,teremos em disco várias cadeias de registros ordenados.

Esta fase de geração de cadeias de registros ordenadas é chamada fase de ordenação interna ("sort interno").

O problema de reunir as cadeias em uma única que será o arquivo ordenado será discutido mais adiante, na fase de intercalação ("merge").

Ao conjunto de operações formado pelas fases de ordenação interna e de intercalação, denominamos ordenação externa ("sort externo").

6. Ordenação por Chaves:

Uma técnica utilizada quando é grande a restrição de memória — precisamente o nosso caso — é a ordenação por chaves.

A ordenação por chaves é implementada criando-se, a partir do arquivo de entrada, um arquivo "reduzido". Tal arquivo contém o mesmo número de registros do arquivo original, mas cada registro é composto apenas da informação de controle para ordenação — chave — e de um apontador contendo o endereço em disco de seu registro correspondente no arquivo original.

Aplica-se então a ordenação ao arquivo reduzido, obtendo-se o arquivo reduzido ordenado.

Utilizando-se os apontadores contidos nos registros, associa-se a cada registro do arquivo reduzido ordenado o seu correspondente no arquivo de entrada, gerando desta forma o arquivo de saída ordenado.

Como na prática o registro reduzido é bem menor do que o original — aproximadamente 20% do tamanho — será possível colocar um número maior de registros simultaneamente na memória. Isto permite gerar cadeias maiores, diminuindo o tempo total de ordenação.

Por outro lado, esta técnica introduz o problema de criação do arquivo de saída com base no arquivo reduzido ordenado, o que consume um tempo considerável, dado o elevado número de acesso a disco necessários para realizar a operação. Existem, no entanto, métodos para otimizar esta última fase, o que será visto adiante.

De uma forma geral, podemos afirmar que a ordenação por chaves apresenta vantagens sobre a ordenação direta dos registros em máquinas com pouca memória disponível para aumentar o fator de bloco dos arquivos, precisamente o caso do TI. (SCHICK[1]).

7. Algorítimo de Ordenação:

O algorítimo de ordenação usado pelo SORTI é o "replacement selection".(KNUTH[4]). Sua característica básica consiste em, uma vez determinado um registro da memória e gravado na saída, o espaço que ocupava anteriormente é imediatamente preenchido com um novo registro da entrada.

Desta forma,supondo um arquivo de entrada em ordem aleatória, o tamanho médio das cadeias ordenadas formadas será o dobro do número de registros que cabem simultaneamente na memória. (FRIEND[2]) (GASSNER[3]) .

8. Compactação dos Registros:

Ainda com o objetivo de aumentar o tamanho das cadeias , o SORTI compacta os registros reduzidos, permitindo a inclusão de um número maior de registros na memória disponível.

Para isto o usuário informará os tipos de campos que compõe a chave de ordenação:

Se o campo pode conter qualquer dos 64 caracteres ASCII, é compactado em 6 bits por carater, passando a ocupar 6/8≐ 75% do tamanho original, no melhor caso.

Se o campo pode conter apenas as 26 letras e o branco, é compactado em 5 bits por carater, passando a ocupar 5/8≅ 62% do tamanho original, no melhor caso.

Se o campo pode conter apenas a representação ASCII dos dígitos de Ø a 9, é compactado em 4 bits por dígito, passando a ocupar 4/8=50% do tamanho original no melhor caso.

9. Intercalação ("merge"):

Na fase de intercalação, as cadeias de registro ordenados geradas pela fase de ordenação são reunidas em uma única.

Chama-se ordem de intercalação ao número de cadeias intercaladas simultaneamente para gerar uma única : por exemplo, 4 cadeias podem ser reunidas por uma intercalação de ordem 4 ou por três de ordem 2.

A primeira vista, pode parecer ideal utilizarmos sempre a maior ordem de intercalação possível. Mas quando podemos variar o tamanho do bloco dos arquivos das cadeias , isto não é necessariamente verdade.(KNUTH[4]) (FERGUSON[5]).

No caso do TI, no entanto, a restrição de memória nos obriga a fixar o tamanho do bloco na fase de ordenação . Por este motivo, a ordem ótima de intercalação passa a ser a maior permitida pelo espaço de memória disponível para carga de blocos das cadeias.

A técnica de ordenação por chaves utilizada pelo SORTI , acarreta a geração de um número bem pequeno de cadeias , conforme será visto adiante. Isto torna pouco importante o fato de não podermos otimizar ainda mais a ordem de intercalação com a variação do tamanho do bloco das cadeias.

10. Arquivo Reduzido:

Como já vimos, o SORTI não ordena diretamente o arquivo de entrada. A partir deste é criado um outro, o arquivo reduzido.

O arquivo reduzido recebe este nome porque seus regis - tros contém apenas a chave de ordenação e um apontador para o registro correspondente no arquivo original.

A parte da chave pode ser compactada de forma a diminuir o tamanho do registro reduzido. Os critérios de compacta ção são 3 : caracteres ASCII, letras e dígitos. Esta com pactação é feita de forma a manter a ordem da chave ori- ginal.

O apontador contém o endereço de disco do registro cor - respondente no arquivo de entrada.

11. Arquivo de Saída:

Uma vez criado o arquivo reduzido ordenado, parte-se para a criação do arquivo de saída.

Tal arquivo conterá os registros do arquivo de entrada , dispostos na ordem indicada pelo arquivo reduzido ordenado. Isto é conseguido lendo-se cada registro do arquivo reduzido ordenado e utilizando-se seu apontador para obter o registro correspondente no arquivo de entrada, gravando-o na saída.

CAPÍTULO II

UTILIZAÇÃO

II . UTILIZAÇÃO

Este capítulo se destina ao usuário e contém todas as informações necessárias para a utilização do SORTI. É na realidade, o "Manual do Usuário".

Para utilizar o SORTI, o usuário precisa preparar arquivos em disco e fornecer dados sobre o tamanho do registro do arquivo a ordenar e sobre os campos da chave de ordenação.

Os arquivos em disco são quatro: o de entrada, o de saída e dois de trabalho.

Os parâmetros podem ser fornecidos de forma conversacional ou via parâmetro da linguagem de controle, quando se tratar de uma aplicação que utilize o SORTI de forma não interativa.

1. Arquivos :

Para a utilização do SORTI são necessários os seguintes arquivos: ENTRA, SAIDA, CADE1 e CADE2, descritos adiante.

De forma a obter uma boa performance, tais arquivos deverão ser colocados em unidades de disco diferentes.Caso seja necessário juntar alguns (ou todos)no mesmo disco, devem ser alocados em áreas contíguas para minimizar o tempo de execução do programa.

Vejamos a descrição dos arquivos:

a) ENTRA:

É o arquivo de entrada.

Segundo as convenções normais do TI, deverá ter um bloco de 512 bytes e tantos registros por bloco quanto possível. Deverá também ter um indicados de fim de arquivo, ou seja, um registro extra no final totalmente preenchido com o valor hexadecimal AA. Não é permitida a concatenação de arquivos.

b) SAIDA:

É o arquivo de saída gravado pelo SORTI. Terá como conteúdo os registros do arquivo ENTRA colocados de forma ordenada.

Deverá ter espaço suficiente para conter todos os registros do arquivo ENTRA; normalmente deverá ser alocado com o mesmo tamanho deste.

Devido à utilização da técnica de ordenação por chaves, o arquivo SAIDA não poderá ocupar no disco o mesmo lugar do arquivo ENTRA nem das áreas de trabalho.

lho CADE1 e CADE2.

c) CADE1 e CADE2:

São arquivos de trabalho para gravação de cadeias ordenadas de registros.

O tamanho do registro lógico é de um setor (512 bytes).

Para determinar o tamanho de cada arquivo, proceder da seguinte forma :

Determinar :

FB = fator de bloco do arquivo de chaves
MC = tamanho mínimo das cadeias
TA = tamanho de cada arquivo em setores

A partir dos valores de :(ver capítulo IV,item 3)

TC = tamanho da chave em bytes
MD = memória disponível para chaves em bytes
NR = número de registros a ordenar

Calcular, ignorando casas decimais após cada divisão :

$$FB = \frac{512}{TC + 3}$$

$$MC = \frac{MD}{TC + 10}$$

$$TA = \frac{NR}{FB} + \frac{NR}{MC} + 2$$

Nesta equação, o termo NR/FB representa espaço para

chaves, NR/MC representa o número de cadeias e Z é um arredondamento devido ao abandono das frações nas divisões. O termo NR/MC é devido ao fato de só iniciarmos uma cadeia em um setor novo, por motivos de simplicidade da lógica.

Se o tamanho mínimo das cadeias MC for menor do que o número de registros a ordenar NR (MC $\leq$ NR) então CADE2 não será usado e pode ser definido com 1 setor apenas, ou como "DUMMY".

Caso haja problema com espaço em disco para estas áreas, recalcular o tamanho da chave considerando a compactação, ou seja,

campos tipo C- 6 bits para cada carater
campos tipo L- 5 bits para cada letra ou branco
campos tipo N- 4 bits para cada dígito
campos tipo X- 8 bits (não há compactação)

É importante notar que cada campo da chave compactada ocupa um número inteiro de bytes.

Calculando o tamanho dos arquivos de trabalho com base no tamanho da chave compactada precisaremos de uma área menor.

Caso espaço em disco não seja problema, não há necessidade de calcular o tamanho da chave compactada, ficando simplificada a determinação do tamanho das áreas de trabalho. Além disso, pode-se também superdimensionar esses arquivos e mantê-los alocados para serem usados por todas as aplicações do SORTI.

2. Parâmetros:

Ao iniciar o programa, certos parâmetros devem ser fornecidos para definir corretamente o arquivo de entrada e a ordenação desejada. Para isto, o SORTI fará perguntas no vídeo, que deverão ser respondidas pelo utilizador.

Tais dados podem também ser fornecidos via passagem de parâmetro no comando de execução. Esta forma é a mais complicada, uma vez que o SORTI é orientado para processamento conversacional : as perguntas não são colocadas no vídeo quando o usuário fornece como parâmetro de execução as respostas correspondentes.

Neste caso, recomenda-se utilizar primeiramente a forma conversacional, anotando-se as respostas fornecidas.Para utilizar então a forma não-conversacional, basta chamar o SORTI passando como parâmetro as respostas anotadas, todas separadas por vírgula ",".

a) PROCEDIMENTO PADRÃO : (S,N)

significado: Indica se o SORTI deverá seguir seu procedimento padrão, não propondo as questões (b) MENSAGENS, (c) TRACE e (d) COMPACTAÇÃO.

respostas aceitas: "S", "N", "Ø".

"S"- sim, assumir procedimento padrão. respostas a (b),(c) e (d) assumidas como "Ø".

"N"- não
questões (b),(c) e (d) serão propostas.

"Ø"- aceito como "S"

b) MENSAGENS : (V,I)

significado: Indica para que periférico devem ser enviadas as mensagens do SORTI

respostas aceitas: "V", "I", "Ø"

"V"- no vídeo
"I"- na impressora
"Ø"- aceito como "V".

c) TRACE : (S,N)

significado: Indica se o SORTI deverá fornecer mensagens indicando os principais fatores calculados pelo programa.

respostas aceitas: "S", "N", "Ø"

"S"- sim, serão fornecidas tais mensagens
"N"- não
"Ø"- aceito como "N".

d) COMPACTAÇÃO:(S,N)

significado: Indica se o SORTI deverá compactar os campos da chave de acordo com o TIPO (h)

respostas aceitas: "S", "N", "Ø"

"S"-sim, os campos serão compactados
"N"-não
"Ø"-aceito como "S".

e) TAMANHO DO REGISTRO : (NNN)

significado: Indica o tamanho do registro lógico do arquivo a ordenar

resposta aceita : Um número entre 1 e 512 inclusive

f) POSIÇÃO INICIAL: (NNN)

significado: Indica a posição inicial de um campo da chave

resposta aceita: Um número entre 1 e o TAMANHO DO REGISTRO (e)

g) TAMANHO (NN)

significado: Indica o tamanho do campo da chave cuja POSIÇÃO INICIAL (f) foi fornecida.

resposta aceita: Um número maior do que zero de forma que o campo não ultrapasse o final do registro.

h) TIPO (C,L,N,X)

significado: Indica o tipo de informação contida no campo, para efeito de compactação.

C- caracteres ASCII
L- letras ASCII de A a Z e o branco
N- número ASCII de Ø a 9
X- qualquer configuração.

Esta questão será proposta somente se (d) COMPACTAÇÃO for "S"

respostas aceitas : "C","L","N","X","Ø"
"Ø"-aceito como "C"

i) ORDEM: (A,D)

significado: Indica se o campo deverá ser ordenado de forma ascendente ou descendente.

respostas aceitas: "A","D", ou "Ø".

"A"- ascendente
"D"- descendente
"Ø"- aceito como "A"

j) MAIS CAMPOS : (S,N)

significado: Indica se ainda existem mais campos na chave de ordenação

respostas aceitas: "S", "N", "Ø"

"S"- sim,há mais campos
"N"- não,todos os campos já foram definidos
"Ø"- aceito como "N".

3. MENSAGENS:

Durante a execução do SORTI, aparece no vídeo uma mensagem indicando o prosseguimento do programa, informando qual a fase que está em execução:

FASE 1- fornecimento de parâmetros

FASE 2- criação do arquivo de chaves

FASE 3- ordenação-geração de cadeias ordenadas

FASE 4- intercalação-união das cadeias em uma única

FASE 5- criação do arquivo de saída.

A mensagem informa também o tamanho em bytes da área de memória disponível para chaves (MD). Este valor é utilizado para determinar o tamanho dos arquivos de trabalho.

A partir da fase 3, a mensagem de execução informa o número total de registros a processar e o número aproximado de registros já processados pela fase corrente.

Além da mensagem de execução, apenas duas outras mensagens podem ser fornecidas:

- Falta de espaço em disco
- Erro de entrada e saída em disco

No primeiro caso, rever o espaço alocado para áreas de trabalho e de saída, corrigí-lo e recomeçar o SORTI.

No segundo caso, rodar o SORTI novamente e, se o problema persistir será devido à falha na unidade de disco.

4. EXEMPLOS:

A) Seja um arquivo ARQ, cujo registro de 100 bytes possui o seguinte layout:

CAMPO	POSIÇÃO INICIAL	TAMANHO	CONTEÚDO
código	1	10	dígitos
nome	11	25	letras e branco
endereço	36	25	caracteres
outros dados	61	40	vários

Sabendo-se que o arquivo possui quase 1000 registros, ordená-lo de forma ascendente por código:

Solução supondo MD = 5000 :

- alocar em disco o arquivo ARQORD para a saída do SORTI, tendo as mesmas características do arquivo ARQ

- alocar em disco os arquivos TRAB1 e TRAB2, tamanho de registro 512, com o seguinte número de registros :

$$FB = 512/(TC+3) = 512/(10+3) = 39$$
$$MC = MD/(TC+10) = 5000/(10+10) = 250$$
$$TA = \frac{NR}{FB} + \frac{NR}{MC} + 3 = \frac{1000}{39} + \frac{1000}{250} + 3 = 32$$

32 registros cada um, número que poderia ser diminuído se fosse considerado o tamanho da chave compactada. Ver também o exemplo B .

- executar SORTI(ENTRA=ARQ),(SAIDA=ARQORD),
(CADE1=TRAB1),(CADE2=TRAB2).

- responder as questões do SORTI da seguinte forma:

SORTI	UTILIZADOR
PROCEDIMENTO PADRÃO : (S,N)	S
TAMANHO DO REGISTRO : (NNN)	100
POSIÇÃO INICIAL : (NNN)	1
TAMANHO : (NN)	10
TIPO : (C,L,N,X)	N
ORDEM : (A,D)	A
MAIS CAMPOS: (S,N)	N

- como alternativa, o SORTI poderia ser executado em forma não interativa:

```
SORTI 'S,100,1,10,N,A,N'. (ENTRA = ARQ),(SAIDA=ARQORD)
                           (CADE1=TRAB1),(CADE2=TRAB2).
```

B) Ordenar o mesmo arquivo do exemplo anterior por ordem alfabética de nome e endereço, usando o mínimo de espaço em disco possível.

Solução:

- definição : $\lceil x \rceil$ é o menor inteiro que contém x
- alocar o arquivo ARQORD com espaço para 1000 registros de 100 bytes
- alocar os arquivos TRAB1 e TRAB2 com registros de 512 bytes e o seguinte número de registros :

  chave : 25 bytes tipo L e 25 do tipo C

Tamanho da chave compactada : $25x\frac{5}{8} + 25x\frac{3}{4} =$

$=16+19=35$

FB= 512/(TC+3)=512/(35+3)= 13
MC= MD/(TC+10)=5000/(35+10)=111
TA=$\frac{NR}{FB}+\frac{NR}{MC}+3 = \frac{1000}{13}+\frac{1000}{111}+3 = 88$

88 registros cada arquivo.

- executar SORTI (ENTRA=ARQ),(SAIDA=ARQORD),
  (CADE1=TRAB1),(CADE2=TRAB2).

- responder as questões do SORTI da seguinte forma:

SORTI	UTILIZADOR
PROCEDIMENTO PADRAO : (S,N)	S
TAMANHO DO REGISTRO : (NNN)	100
POSICAO INICIAL : (NNN)	11
TAMANHO : (NN)	25
TIPO : (C,L,N,X)	L
ORDEM : (A,D)	A
MAIS CAMPOS : (S,N)	S
POSICAO INICIAL : (NNN)	36
TAMANHO: (NN)	25
TIPO : (C,L,N,X)	C
ORDEM : (A,D)	A
MAIS CAMPOS : (S,N)	N

- como alternativa, o SORTI poderia ser executado de forma não interativa :

```
SORTI 'S,100,11,25,L,A,S,36,25,C,A,N.'
(ENTRA=ARQ),(SAIDA=ARQORD),(CADE1=TRAB1),(CADE2=TRAB2).
```

CAPÍTULO III

ESPECIFICAÇÃO

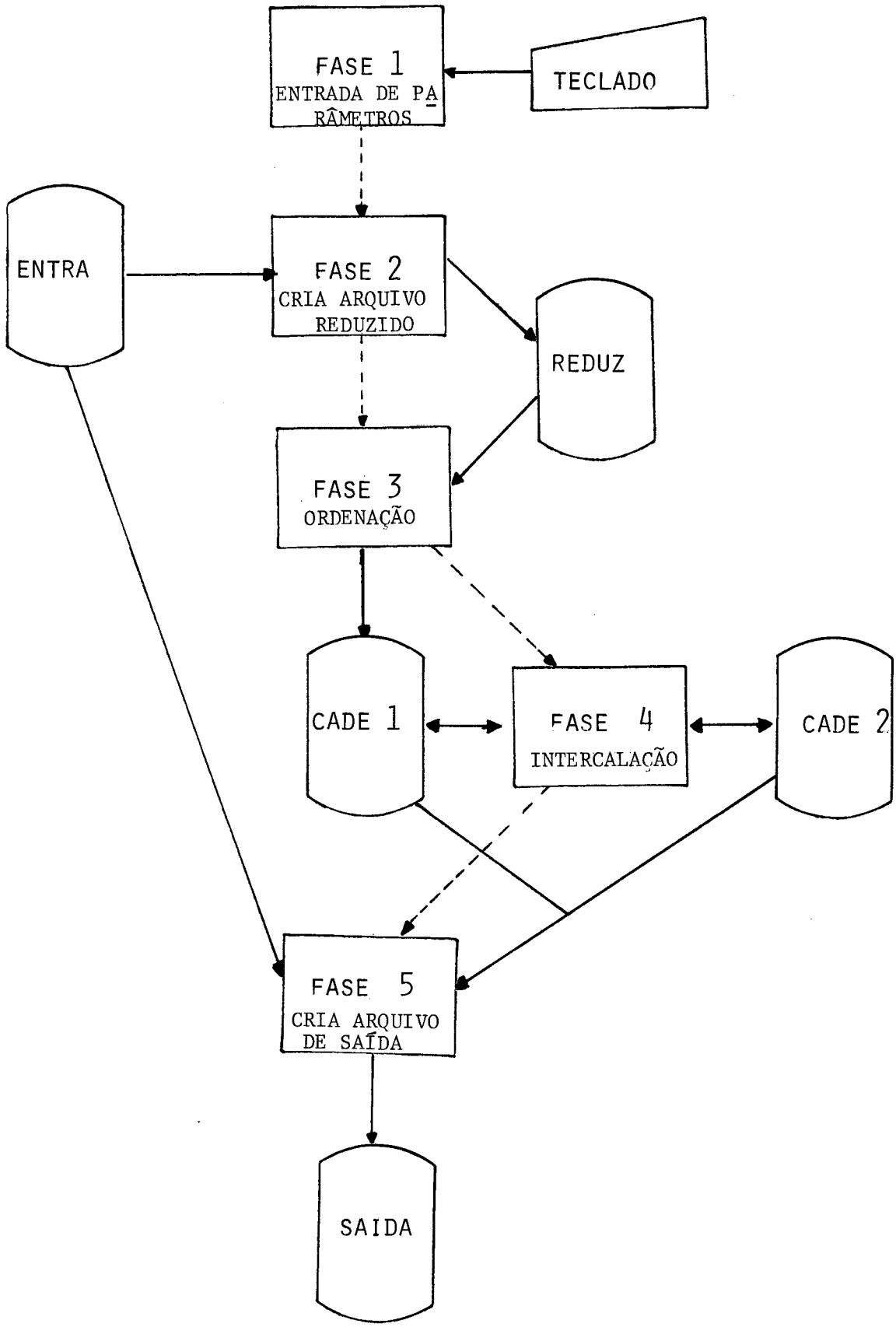
III. ESPECIFICAÇÃO

1 - Introdução

Este capítulo trata das características técnicas do SORTI, através da descrição de sua estrutura, seus módulos, arquivos e algorítimos utilizados.

A seguir apresentamos o fluxograma geral do SORTI , com uma descrição sucinta de suas fases.

FLUXÓGRAMA GERAL DO SORTI



Fase 1 : Entrada de Parâmetros

Controla a entrada de parâmetros necessários para a execução do SORTI. Tais dados, após analisados segundo critérios de validação, são passados às outras fases.

Fase 2 : Criação do Arquivo Reduzido

A partir do arquivo de entrada, a fase 2 cria o arquivo reduzido, onde cada registro contém apenas a chave de ordenação e um apontador para o registro correspondente no arquivo de entrada.

Fase 3 : Ordenação

A fase 3 tem como entrada o arquivo reduzido , e gera cadeias ordenadas de registros no arquivo CADE1.

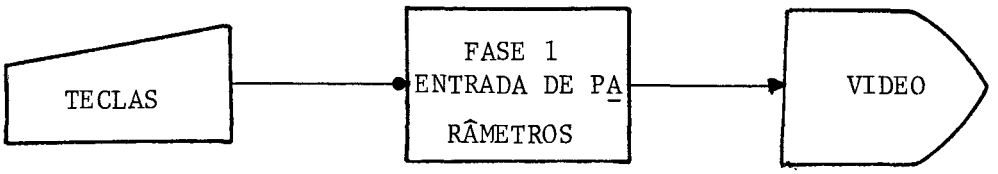
Fase 4 : Intercalação

A fase 4 tem por objetivo intercalar as cadeias de registros geradas pela fase 3 do arquivo CADE1, de forma a gerar uma única cadeia. Caso a fase 3 gere apenas uma cadeia, então a fase 4 não é executada. Dependendo do número de cadeias a intercalar, a cadeia única criada pela fase 4 poderá estar no arquivo CADE1 ou no CADE2.

Fase 5 : Criação do Arquivo de Saída

A fase 5 obtém os registros do arquivo de entrada a partir da ordem definida pela cadeia final ordenada, que pode estar em CADE1 ou CADE2, gerando assim o arquivo de SAÍDA.

2- FASE 1- ENTRADA DE PARAMETROS



Na fase 1 é feita toda a interação com o usuário para que sejam fornecidos os parâmetros necessários à execução do SORTI. Tais dados serão informados no modo conversacional ou no modo não-conversacional.

Modo conversacional

A fase 1 coloca no vídeo diversas perguntas, com o objetivo de identificar as características do arquivo a ordenar bem como a chave de ordenação. Cada resposta fornecida pelo usuário é verificada pelo programa e, caso não seja aceita pelos critérios de validação, a pergunta correspondente é repetida.

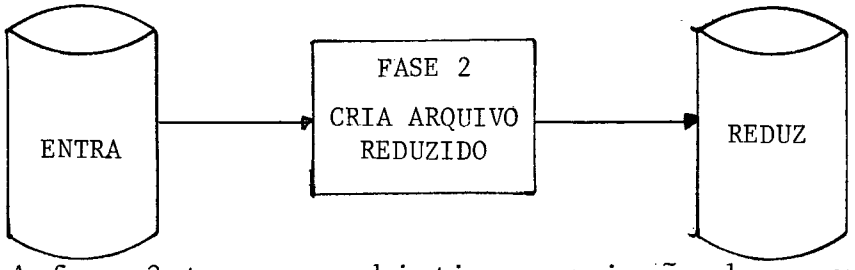
As perguntas e os critérios de validação estão definidas no capítulo 2-Utilização, Item 2.2-Parâmetros.

Modo não-conversacional

Os dados sobre o arquivo e a chave de ordenação são fornecidos via parâmetro de comando de execução do SORTI.

Tais informações são verificadas tal como no modo conversacional mas, em caso de informação inconsistente, o programa passa automaticamente para o modo conversacional.

3- FASE 2- Criação do Arquivo Reduzido



A fase 2 tem como objetivo a criação do arquivo reduzido, a partir do arquivo de entrada.

3.1- Arquivo ENTRA

É o arquivo a ordenar. Possui um bloco de tamanho 512 bytes e é sempre blocado com o maior fator de bloco possível.

Seu conteúdo não sofre alteração alguma com a execução do programa.

3.2- Arquivo REDUZ

Possui também um bloco de 512 bytes contendo tantos registros quanto possível.

Seu registro possui 3 campos :

NUMERO DO BLOCO (2 bytes): Contém o número de ordem do bloco onde se encontra o registro correspondente no arquivo ENTRA , considerando-se o valor zero para o primeiro bloco do arquivo.

NUMERO DO REGISTRO (1 byte) : Contém o número de ordem dentro de um bloco do registro correspondente no arquivo ENTRA, considerando-se o valor zero para o primeiro registro de cada bloco.

CHAVE : Contém a versão compactada da chave de ordenação do registro correspondente no arquivo ENTRA. O tamanho deste campo

depende do tamanho da chave de ordenação e dos critérios de compactação utiliza - dos, mas é invariável durante uma execução do SORTI.

3.3- Procedimento geral :

Para cada registro lido do arquivo ENTRA é gravado um registro no arquivo REDUZ.

Os campos NUMERO DO BLOCO e NUMERO DO REGISTRO de cada registro do arquivo REDUZ são preenchidos com o auxílio de contadores apropriados colocados junto à rotina de leitura do arquivo ENTRA.

O campo CHAVE de cada registro do arquivo REDUZ é preechido compactando-se e concatenando-se os diversos campos da chave de ordenação, conforme a especificação feita pelo usuário durante a fase 1-entrada de parâmetros.

3.4- Compactação da chave

Para efeito de compactação, os campos da chave são classificados em 4 tipos distintos :

C : qualquer carater ASCII
L : representação ASCII das letras de A a Z e do branco
N : representação ASCII do dígito de Ø a 9
X : qualquer configuração.

A compactação é feita de forma a manter nas chaves compactadas a mesma relação de ordem das chaves não compactadas.

A seguir apresentamos a lógica geral das rotinas de compactação:

. C : qualquer caracter ASCII

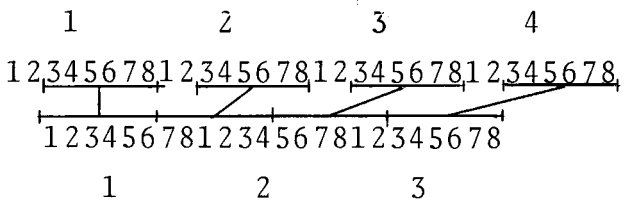
Os 64 caracteres ASCII podem ser representados em 6 bits, já que $2^6 = 64$. Sua configuração em hexadecimal é representada na forma 2x, 3x, 4x, 5x, ou seja no intervalo 20,5F .

Primeiramente subtrai-se #20 de cada byte , passando-os para o formato Øx,1x,2x,3x, ou seja no intervalo 0,3F .

Nesse momento, os dois primeiros bits estarão sempre zerados, e podem ser ignorados.

É feito então um reposicionamento, que efetivamente compacta a informação, segundo o esquema abaixo :

BYTES ORIGINAIS



BYTES COMPACTADOS

A forma compactada é, no melhor caso, de tamanho igual a 3/4=75% do original, representando um granho de até 25%.

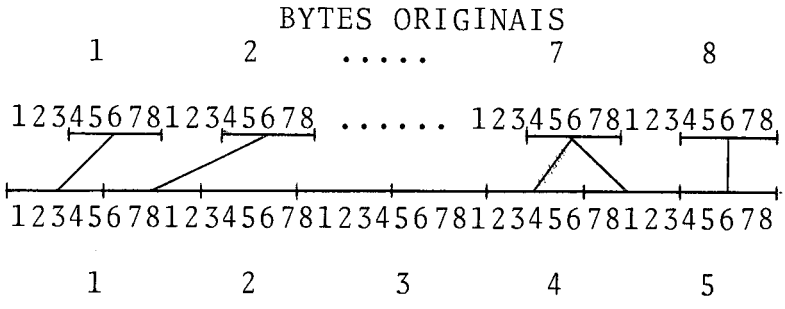
. L: representação ASCII das letras de A a Z e do branco

As 26 letras mais o branco podem ser representadas em 5 bits, já que $2^5 = 32$ 27. Sua

configuração em hexadecimal é representada na forma 4x, 5x e 20.

Ignorando-se os 3 primeiros bits, obtere - mos os formatos ØX e 1x. O branco 20 passa rá a 00.

Passamos então ao reposicionamento :



A forma compactada é, no melhor caso, de ta manho igual a 5/8=62,5% do original, repre- sentando um ganho de até 37,5%.

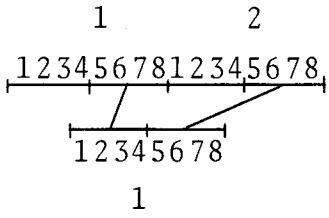
. N : representação ASCII dos dígitos de Ø a 9

Os 10 dígitos podem ser representados em 4 bits, já que 2^4 = 16 10. Sua configuração em hexadecimal é representada na forma 3x, de 3Ø a 39.

Ignorando os 4 primeiros bits, teremos os formatos ØØ a Ø9.

O reposicionamento será :

BYTES ORIGINAIS



BYTE COMPACTADO

A forma compactada é, no melhor caso, de tamanho igual a 1/2 =50% do original.

. X : qualquer configuração

Neste caso não é feita a compactação.

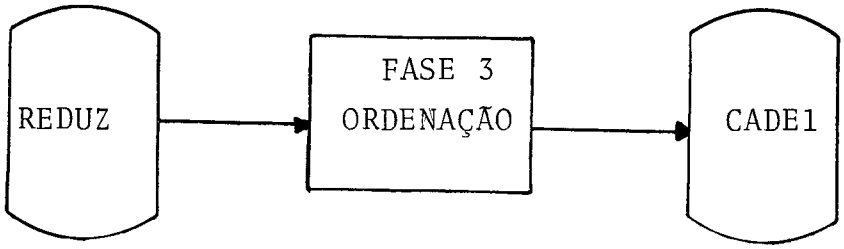
3.5- Preparação para ordenação descendente

Caso seja especificado pelo usuário ordenação descendente para um ou mais campos da chave, utiliza-se uma transformação que inverte a relação de ordem nas configurações, ordenando-se então ascendentemente o campo transformado.

A transformação utilizada é a operação "complemento a 1" feita em cada byte do campo a ordenar de forma descendente.

Note-se que tal transformação é válida mesmo para os campos compactados.

4 - FASE 3- Ordenação



A fase 3 processa o arquivo reduzido, ordenando seus registros, gerando cadeias ordenadas sobre o arquivo CADE1.

4.1- Arquivo REDUZ

Ver definição no item 3.3.2

4.2- Arquivo CADE1

Possui as mesmas características e descrição do arquivo REDUZ-ver item 3.3.2.

Existe porém um fato que o torna diferente deste último arquivo: CADE1 é na realidade um conjunto de sub-arquivos, cada qual contendo uma cadeia ordenada de registros.

O primeiro registro de cada cadeia é sempre o primeiro registro de um bloco — a recíproca não é necessariamente verdadeira — .

Após o último registro de cada cadeia existe um registro especial de fim de cadeia, identificado pelo valor hexadecimal ≠EE contido no campo NUMERO DO REGISTRO.

CADE1 contém também uma lista ligando cada final de cadeia ao final da cadeia anterior. Tal lista é implementada colocando-se no campo NUMERO DO BLOCO de cada registro de fim de cadeia o número de ordem do bloco onde se encontra o fim da cadeia anterior. O número

do registro dentro do bloco é colocado no espaço reservado à chave. Como a primeira cadeia não possui anterior, seu registro de fim contém o valor ≠EEEE no campo NUMERO DO BLOCO.

A seguir, um exemplo da lista do arquivo CADE1 , mostrando em especial os registros de fim de cadeia.

BLOCO	REGISTRO	VALOR DOS CAMPOS EM HEXADECIMAL			NUMERO DA CADEIA
		NUMERO DO BLOCO	NÚMERO DO REGISTRO	1a.POSIÇÃO DA CHAVE	
0	0	.	.	.	1
.	.				.
.	.				.
.	.				.
2	4	EEEE	EE	.	.
3	0	.	.	.	2
.	.				.
.	.				.
.	.				.
5	2	0002	EE	04	.
6	0	.	.	.	3
.	.				.
.	.				.
.	.				.
7	3	0005	EE	02	.
8	0	.	.	.	4
.	.				.
.	.				.
.	.				.
10	0	0007	EE	03	.

4.3-Procedimento geral

A seguir apresentamos uma forma simplificada do algoritimo utilizado na fase de ordenação interna :

PASSO A) Ler do arquivo de entrada para a memória o maior número possível de registros.

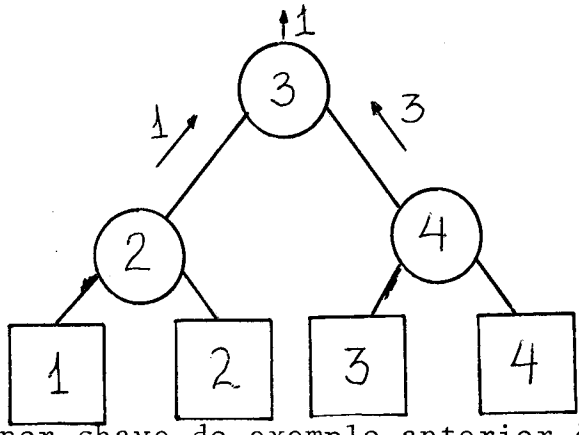
PASSO B) Selecionar da memória o registro de menor chave. Se este registro pertence à cadeia que está sendo gravada então é gravado na saída. Caso contrário prosseguir no passo D.

PASSO C) Ler um novo registro, colocando-o no espaço de memória liberado pelo registro recém-gravado. Se a chave do novo registro for menor do que a deste último, então o novo registro já não pertence a cadeia que está sendo gravada. Voltar ao passo B.

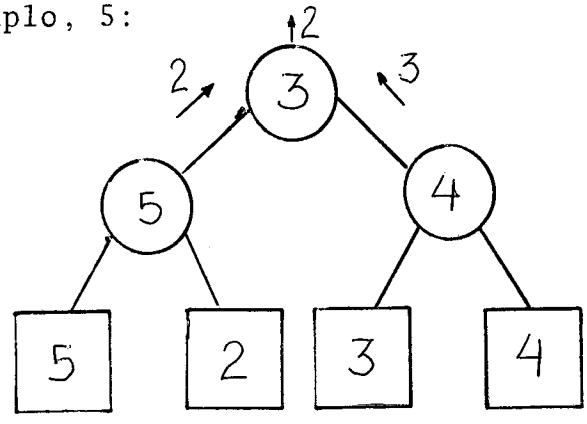
PASSO D) Neste ponto encerrou-se uma cadeia de registros ordenados e inicia-se uma nova cadeia, gravando-se na saída o registro recém-selecionado. Prosseguir no passo C.

O passo A será substituído pela criação, durante a inicialização, de uma cadeia de registros fictícios, que sofrerão o mesmo processo de seleção dos registros reais mas não serão gravados no arquivo de saída.

O passo B será realizado com o auxílio de uma árvore binária de comparações, cujo objetivo global é selecionar a menor chave. As folhas da árvore contém as chaves,e os nós internos indicam a maior chave quando se compara as menores chaves das sub-árvores à esquerda e à direita de cada nó, conforme o exemplo a seguir :



A menor chave do exemplo anterior é 1. Supondo que 1 será gravada no arquivo de saída, seu espaço origi - nal será preenchido por uma nova chave, digamos por exemplo, 5:



Desta vez a menor chave foi 2. Note-se, consultando o primeiro exemplo, que só foram realizadas comparações com as chaves indicadas nos nós encontrados nos caminhos da folha onde foi incluída a nova chave até a raiz.

Para a informação de cadeia necessária ao passo C , cada chave será associada a um campo "número da cadeia". O primeiro registro lido será sempre associado a uma nova cadeia, de forma a se destacar da cadeia inicial fictícia.

No passo D é gravado um registro especial na saída, indicando fim de cadeia.

4.4- Representação dos dados na memória

Usando o fato de que o número de nós externos de uma árvore binária é sempre uma unidade maior do que o número de nós internos, criamos um nó fictício extra, acima da raiz. Desta forma, a árvore modificada conterá um número de nós internos igual ao de externos.

Para efeito de simplificação, a representação da árvore será feita por uma seqüência de blocos de controle, cada um contendo um nó interno e um nó externo.

Campos do bloco de controle :

- relativos ao nó interno:
  PI- apontador para o pai do nó interno
  PERD- apontador para a maior chave dentre a menor da sub-árvore esquerda e a menor da sub-árvore direita
  NC- número da cadeia da chave apontada por PERD

- relativos ao nó externo:
  PE- apontador para o pai do nó externo
  REG-registro reduzido armazenado no nó externo; contém a chave e o endereço do registro correspondente no arquivo de entrada.

Outras variáveis utilizadas pelo algorítimo:

CMAX- contador que controla o fim do algorítimo
NCC - número da cadeia que está sendo gravada no momento corrente
ULC - última chave gravada na saída
PTR$V-apontador para a chave vencedora (a menor)

NCV- número da cadeia da chave vencedora
P - número de blocos de controle disponíveis

4.5- Algorítimo de Ordenação

PASSO 1: (Inicialização)

CMAX, NCC, NCV = Ø
ULC = 00
PTR$V= endereço do primeiro bloco de controle
P = número de blocos de controle disponíveis
Variando o índice J desde Ø a P-1, fazer:
PERD(J)=J
NC(J)= Ø
PE(J)= endereço do bloco de controle de ordem $\left\lfloor \frac{P+J}{2} \right\rfloor$

PI(J)= endereço do bloco de controle de ordem $\left\lfloor \frac{J}{2} \right\rfloor$

PASSO 2 : (Teste de fim de cadeia)

Se NCV = NCC prosseguir no passo 3
Caso contrário é fim de cadeia
Se NCV CMAX então fim do algorítimo
NCC=NCV

PASSO 3 : (Gravar o registro vencedor)

Se NCV≠Ø, gravar REG (PTR$V) e salvar a CHAVE (PTR$V) em ULC.

PASSO 4 : (Ler novo registro)

Se fim de arquivo então fazer NCV=CMAX+1 e prosseguir no passo 5.
Ler novo registro sobre REG (PTR$V)
Se CHAVE (PTR$V) < ULC então este registro já não pertence a esta cadeia, fazer NCV=NCV+ 1

e se NCV > CMAX fazer CMAX=NCV.

PASSO 5 : (Preparar para atualização dos perdedores)

T = PE (PTR$V)

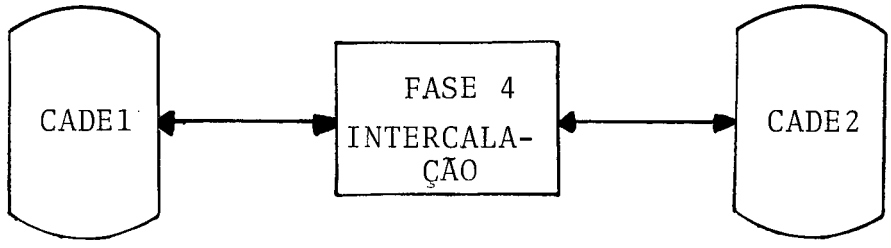
PASSO 6 : (Determinar novo perdedor)

Se NC(T) < NCV ou
NC(T)= NCV e CHAVE (PERD(T)) < CHAVE (PTR$V)
então trocar PERD(T) com PTR$V e NC(T) com NCV

PASSO 7 : (Subir a árvore)

Se T aponta para o bloco de controle de ordem 1 (a raiz) então prosseguir no passo 2.
Senão, fazer T=PI(T) e voltar ao passo 6.

5- FASE 4- Intercalação de Cadeias



A fase 4 tem como entrada o arquivo CADE1, que contêm as cadeias ordenadas de registro geradas pela fase 3 (ordenação). Sua saída será uma única cadeia ordenada, que poderá estar no arquivo CADE2 ou no próprio CADE1, dependendo do número de cadeias encontradas na entrada.

5.1- Arquivo CADE1
Ver definição no item 4.2.

5.2- Arquivo CADE2
Sua definição é idêntica à do arquivo CADE1- ver item 4.2.

5.3- Procedimento geral
O procedimento da fase de intercalação é constituído com base nos procedimentos da fase de ordenação. Convêm portanto consultar primeiramente o item 4.3— Procedimento geral da fase de ordenação, antes de examinar as seções seguintes.

A seguir apresentamos uma forma simplificada do algorítimo da fase de intercalação:

PASSO A) Preencher a memória com o maior número possível de registros, sendo cada um deles o primeiro de sua cadeia.

PASSO B) Selecionar o registro de menor chave. Se este registro pertence à cadeia que está sendo gravada então é gravado na saída. Caso contrário prosseguir no passo D.

PASSO C) Ler um novo registro da mesma cadeia de entrada que originou o registro recém-gravado. Se esta cadeia não possui mais registros,ler o primeiro registro de uma nova cadeia de entrada e marcá-lo como não pertencente a cadeia que está sendo gravada na saída. Voltar ao passo B.

PASSO D) Neste ponto encerrou-se uma cadeia de registros ordenados na saída e inicia-se uma nova cadeia, gravando-se o registro recém-selecionado. Prosseguir no passo C.

O passo A será substituído pela criação, durante a inicialização, de cadeias fictícias de tamanho zero .

O passo B será realizado com auxílio de uma árvore binária de comparações análoga à utilizada na fase de ordenação.

A informação de cadeia citada no passo C é inidicada pelo campo "número da cadeia" associado a cada registro. Para localizar uma nova cadeia, utiliza-se a lista que liga as cadeias, conforme definição dos arquivos CADE1 e CADE2.

No passo D é gravado um registro especial de fim de cadeia.

O final do algoritimo é detetado quando é gravada uma única cadeia na saída. Caso mais de uma cadeia tenha sido gerada, então volta-se ao passo A,utilizando-se como entrada o arquivo de saída e vice-versa.

5.4-Representação dos dados na memória

É análoga à do algorítimo de ordenação. O bloco de controle na memória, contém, além dos campos definidos no item 3.4.4, o campo NC$ENTRA contendo um apontador para o bloco de registros pertencentes a cadeia relativa a este nó externo.

5.5- Algorítimo de Intercalação

PASSO 1 : (Inicialização)

CMAX,NCC,NCV = Ø
PTR$V = endereço do primeiro bloco de controle
P = número de blocos de controle disponíveis
Variando o índice J desde Ø a P-1, fazer :
PERD (J) = J
NC (J) = Ø
NC$ENTRA(J)= J+1
PE(J)= endereço do bloco de controle de ordem $\left\lfloor \frac{P+J}{2} \right\rfloor$
PI(J)= endereço do bloco de controle de ordem $\left\lfloor \frac{J}{2} \right\rfloor$

PASSO 2 : (Teste de fim de cadeia)

Se NCV= NCC prosseguir no passo 3
Caso contrário é fim de cadeia
Se NCV > CMAX prosseguir no passo 8
NCC = NCV

PASSO 3 : (Gravar o registro vencedor)

Se NCV $\neq \emptyset$, gravar REG (PTR$V)

PASSO 4 : (Ler novo registro)

Ler um novo registro.
Se o registro lido pertence a uma nova cadeia da entrada então NCV=NCV+1 e se NCV > CMAX então fazer CMAX = NCV.
Se fim de arquivo então fazer NCV=CMAX+1.

PASSO 5 : (Preparar para atualização dos perdedores)

T = PE(PTR$V)

PASSO 6 : (Determinar novo perdedor)

Se NC(T) < NCV ou
NC(T)= NCV e CHAVE (PERD(T)) < CHAVE (PTR$V)
então trocar PERD(T) com PTR$V e NC(T) com NCV

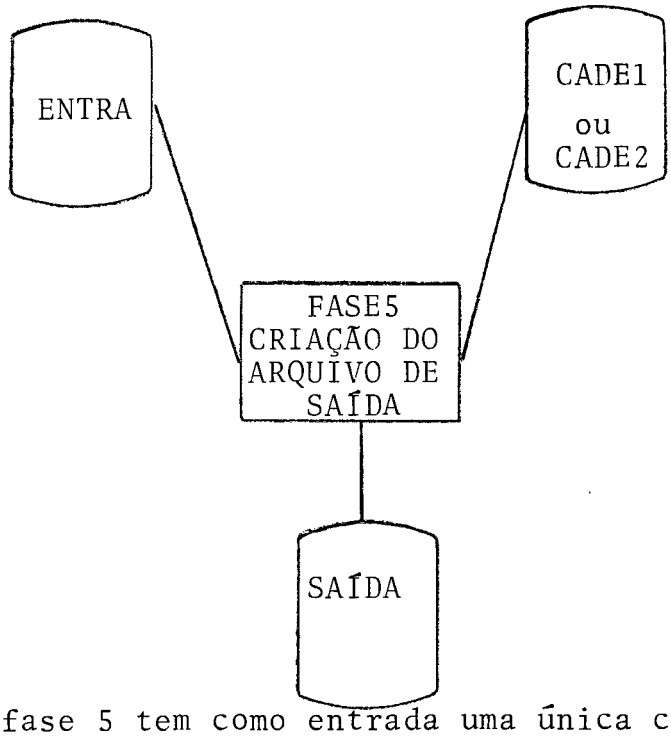
PASSO 7 : (Subir a árvore)

Se T aponta para o bloco de controle de ordem 1 (a raiz) então prosseguir no passo 2.
Senão, fazer T=PI(T) e voltar ao passo 6.

PASSO 8 : (Preparar para recomeçar)

Se NCV < =3 , então fim do algorítimo.
Trocar arquivo de entrada com o de saída e voltar ao passo 1.

6- FASE 5- Criação do arquivo de Saída



A fase 5 tem como entrada uma única cadeia de registros reduzidos ordenados que estará no arquivo CADE1 ou CADE2.Por intermédio dos apontadores contidos nesses registros consulta-se o arquivo ENTRA e gera- se o arquivo de SAIDA.

6.1- Arquivo ENTRA
Ver definição no item 3.1.

6.2- Arquivo CADE1/CADE2
Ver definição no item 4.2.

6.3- Arquivo SAIDA
Possui as mesmas características do arquivo ENTRA,ou seja, bloco de tamanho 512 bytes , e o maior fator de bloco permitido pelo tamanho do registro.

Contém os registros do arquivo ENTRA colocados na ordem especificada pelo usuário.

6.4- Procedimento geral:
Consiste em, para cada registro da cadeia de registros reduzidos ordenados, obter o seu correspondente no arquivo de entrada e gravá-lo no arquivo de saída.
Para efeito de otimização, este procedimento é feito por grupos de registros, realizando-se as leituras por ordem de endereço de disco. Desta forma, o braço do disco

será movimentado de fora para dentro, evitando um grande número de idas e vindas e diminuindo assim o tempo total de execução.

Podemos então estabelecer uma analogia completa entre o procedimento de criação do arquivo de saída e o da ordenação dos registros reduzidos: ao invés de utilizarmos a chave de ordenação nas comparações, usamos simplesmente o endereço do bloco do registro do arquivo de entrada. Cada cadeia de saída será formada por um grupo de registros do arquivo ENTRA lidos em conjunto para a memória.

Observadas essas modificações, passam a ser válidas para a criação do arquivo de saída todas as considerações analisadas nos itens 4.3—— procedimento geral da ordenação e 4.4 — representação dos dados na memória.

Para evitar que os três arquivos CADE, ENTRA e SAÍDA sofram acessos simultâneos, determina-se o tamanho da cadeia, lê-se os registros reduzidos necessários do arquivo CADE e cria-se uma cadeia fictícia, com a finalidade de descarregar a cadeia lida de CADE. Neste ponto, os registros são lidos de ENTRA por ordem de endereço de disco e então gravados em SAIDA.

Cabe observar aqui que a ordenação utilizada , nesta fase, não precisaria utilizar um algoritimo sofisticado como o "replacement selection". Não havendo necessidade de gerar cadeias de tamanho maior do que o espaço de memória disponível para registros,poderia ser usado um algorítimo bem mais simples,cuja perda em performance no número de comparações efetuadas seria compensada pelo ganho de memória para registros acarretado pela não-utilização da árvore de

comparações.

Optou-se entretanto pela padronização, utilizando também nesta fase o "replacement selection", que tornou-se desta forma o único algorítimo utilizado pelo SORTI.

6.5- Algorítimo de criação do arquivo de saída

PASSO 1 : (Inicialização)

CMAX, NCC, NCV, NCFICT = Ø
PTR$V= endereço do primeiro bloco de controle
P = número de blocos de controle disponíveis
Variando o índice J de Ø a P-1 fazer :
PERD (J) = J
NC (J) = Ø
PE (J) = endereço do bloco de controle de ordem $\left\lfloor \frac{P+J}{2} \right\rfloor$
PI (J) = endereço do bloco de controle de ordem $\left\lfloor \frac{J}{2} \right\rfloor$

PASSO 2 : (Teste fim de cadeia)

Se NCV= NCC prosseguir no passo 3.
Caso contrário, se NCC≠NCFICT é fim de cadeia e descarrega-se a área de saída.
Se NCV-NCFICT > 1 então fazer NCFICT=NCV.
Se NCV > CMAX então fim do algorítimo NCC=NCV

PASSO 3 : (Gravar o registro vencedor)

Se NCV≠NCFICT então obter do arquivo ENTRA o registro correspondente ao vencedor e movê-lo para a área de SAÍDA.

PASSO 4 : (Ler novo registro reduzido)

Salvar NCV em NCV$OLD
Se o grupo de registros reduzidos não foi ainda completado, ler mais um.
Se fim do arquivo reduzido ordenado, então NCV= CMAX + 1
Caso contrário fazer NCV=NCV$OLD + 1 e se NCV > CMAX então fazer CMAX=NCV

PASSO 5 : (Preparar para atualização dos perdedores)

T = PE (PTR$V)

PASSO 6 : (Determinar novo perdedor)

Se NC(T) < NCV ou
NC(T)= NCV e endereço (PERD(T)) < endereço (PTR$V)
então trocar PERD(T) com PTR$V e NC(T) com NCV.

PASSO 7 : (Subir a árvore)

Se T aponta para o bloco de controle de ordem 1 (a raiz)então prosseguir no passo 2.
Senão fazer T=PI(T) e voltar ao passo 6.

CAPÍTULO IV

IMPLEMENTAÇÃO

IV. IMPLEMENTAÇÃO

1. Introdução

Este capítulo sumariza características da implementação do SORTI, desde a compilação dos módulos até a execução do Sistema.

2. Compilação

O SORTI foi totalmente programado em PLTI, a linguagem de alto nível própria ao Terminal Inteligente.

A maioria dos módulos foi compilada no próprio TI, sendo que uma restrição existente com relação ao número de variáveis permitidas obrigou que os módulos de ordenação, intercalação e saída fossem compilados no sistema Burroughs B-6700, utilizando-se um compilador cruzado PLTI.

Com o objetivo de liberar uma área de memória maior para carga de registros, a opção TRACE (ver capítulo II-Utilização, Item II-Parâmetros), foi transferida para o vídeo, deixando então de ser necessária a rotina de entrada e saída para a impressora.

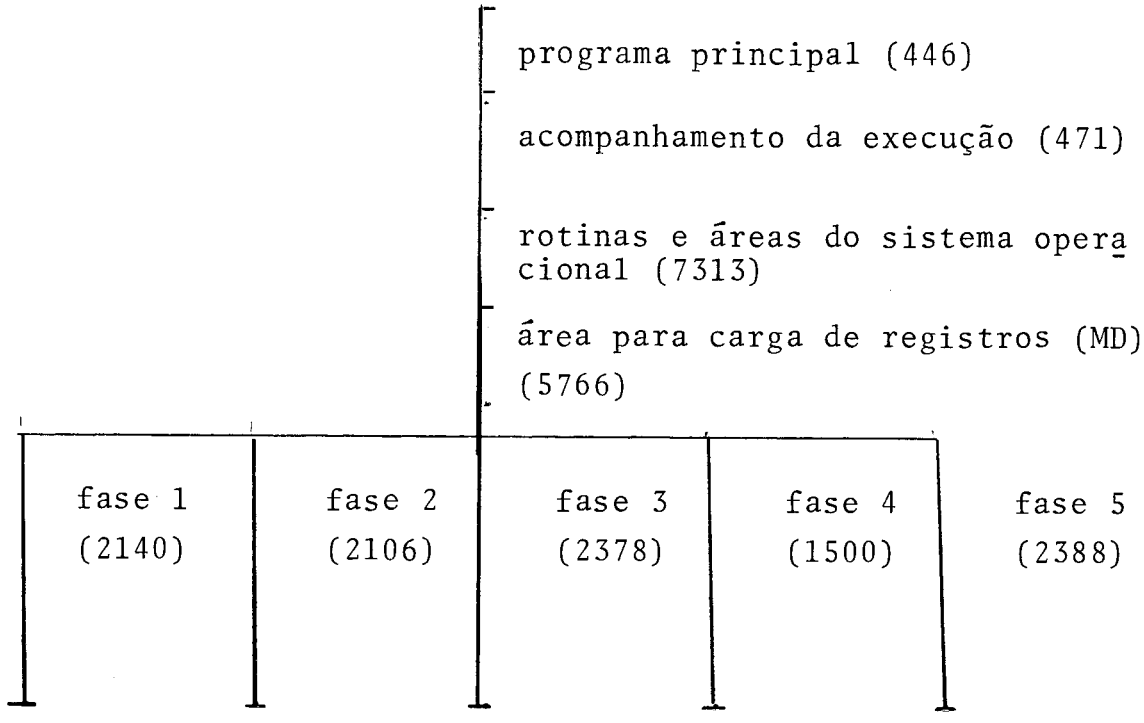
Para minimizar o tempo de entrada e saída em disco, o pessoal do Núcleo de Computação Eletrônica forneceu uma rotina especial sem controle de blocagem de registros, desnecessário para o SORTI.

No anexo pode ser encontrada a descrição dos módulos do SORTI.

3. Reunião dos Módulos Compilados

A reunião dos módulos compilados e respectiva resolução de referências externas foi feita no próprio TI, eliminando-se todas as facilidades de depuração e liberando assim maior área de memória para carga de registros.

Foi construída uma estrutura em "overlay", tendo na raiz o programa principal e cinco ramos mutuamente exclusivos, cada um responsável por uma fase do SORTI:



4. Execução

O SORTI foi executado um grande número de vezes, tanto na forma conversacional,quanto na não conversacional.

Foram efetuados diversos tipos de testes, sendo os resultados comentados no próximo capítulo.

CAPÍTULO V

CONCLUSÕES

1. Análise de Desempenho :

1.1 Introdução

De uma forma geral, a análise de desempenho do SORTI mostrou uma tarefa difícil particularmente devido ao grande número de variáveis que influem no estudo, como por exemplo, número de registros, tamanho dos registros, tamanho da chave, fator de bloco dos arquivos, tipo de compactação das chaves, memória disponível para o programa, distribuição estatística das chaves,etc.

Além do número de variáveis envolvido, houve também dificuldade em obter dados experimentais, uma vez que o tempo de cada medição foi longo, como será visto adiante (aproximadamente 35 minutos para 1000 registros e 3 horas para 5000 registros).

1.2 Fase 1- Entrada de Parâmetros

Na forma conversacional, o tempo de execução da fase 1 depende da velocidade e correção com que o operador fornece os parâmetros. Na forma não-conversacional, supondo que não hajam erros nos dados fornecidos, o tempo da fase 1 é constante aproximadamente e igual a 5 segundos.

1.3- Fase 2- Criação do Arquivo Reduzido

Os tempos em segundos observados para registros de tamanho 50 foram os seguintes :

TAMANHO DA CHAVE	NÚMERO DE REGISTROS 100	500	1000
2	25	91	174
5	25	93	180
20	26	100	192

Podemos notar que o tempo da fase 2 é linear com o número de registros e que há pouca variação com o tamanho da chave.

Além disso, se for usada compactação das chaves o tempo subirá linearmente com o número de registros.

1.4- Fase 3- Ordenação

Os tempos em segundos observados para registros de tamanho 50 foram os seguintes :

TAMANHO DA CHAVE	NÚMERO DE REGISTROS 100	500	1000
2	90	509	840
5	98	498	841
20	107	430	811

Podemos notar que o tempo da fase 3 é linear com o número de registros. Isto se deve ao fato que para um dado tamanho da chave, o tempo para gerar uma cadeia é razoavelmente constante e o tempo da fase é diretamente proporcional ao número de cadeias geradas.

Interessante o fato do tempo diminuir quando aumenta o tamanho da chave. O que ocorre é quando a chave aumenta diminui o número de chaves que podem ser colocadas simultaneamente na memória, diminuindo assim o número total de comparações nesta fase.

Isto não ocorre para 100 registros porque neste caso todas as chaves cabem na memória, havendo a geração de uma única cadeia.

Apesar do tempo variar com o tamanho da chave, esta variação não é grande.

1.5- Fase 4- Intercalação de Cadeias

Os tempos em segundos observados para registros de tamanho 50 foram os seguintes :

TAMANHO DA CHAVE	NÚMERO DE REGISTROS 100	500	1000
2	0	0	180
5	0	89	240
20	0	208	701

A análise desta fase foi a que apresentou maiores dificuldades por causa do longo tempo de duração do SORTI. Tal limitação obrigou que fossem feitos apenas testes de intercalação de 2 cadeias, com exceção do teste de 1000 chaves de 20 posições que foi de 3 cadeias.

Os pontos Ø indicados na tabela representam situações em que a ordenação gerou uma única cadeia,não havendo necessidade da fase de intercalação.

É difícil chegar a uma conclusão significativa sobre a duração desta fase por causa do consideravel tempo de movimentação do braço do disco, já que são percorridas no mínimo 3 áreas em disco (cadeias de entrada e a de saída).

1.6- Fase 5- Criação do Arquivo de Saída

Os tempos em segundos observados para registro de tamanho 50 foram os seguintes :

TAMANHO DA CHAVE	NÚMERO DE REGISTROS 100	500	1000
2	90	398	795
5	90	395	796
20	90	391	798

O quadro acima mostra que o tempo da fase 5 é linear com o número de registros e varia muito pouco com o tamanho da chave.

1.7- Tempo Total do SORTI

Sumarizando, apresentamos o seguinte quadro de dependência entre o tempo de execução e os parâmetros de entrada:

FATORES QUE INFLUENCIAM O TEMPO DE EXECUÇÃO			
FASE	NÚMERO DE REGISTROS	TAMANHO DO REGISTRO	TAMANHO DA CHAVE
1. Parâmetros	não	não	não
2. Reduzido	sim	sim	pouco
3. Ordenação	sim	não	pouco
4. Intercalação	sim	não	sim
5. Saída	sim	sim	pouco

Pela análise anterior concluímos que o SORTI depende fortemente do número de registros a ordenar, do tamanho do registro e também tem alguma dependência do tamanho da chave. Outra conclusão é de que o tempo total é aproximadamente linear com relação a essas variáveis.

O quadro de dependência entre o tempo de execução e os parâmetros de entrada foi confirmado pelas diversas medições efetuadas.

Vamos analisar agora o tempo total de execução do SORTI:

TEMPO DE EXECUÇÃO DO SORTI REGISTROS DE 50 BYTES DISTRIBUIÇÃO DAS CHAVES ALEATÓRIA			
TAMANHO DA CHAVE	NÚMERO DE REGISTROS 100	500	1000
2	205	1010	1994
5	213	1075	2058
20	223	1129	2462

Examinando esses dados, observamos uma dependência razoavelmente linear do tempo total de execução do SORTI com o número de registros, uma vez fixados os tamanhos do registro e da chave de ordenação.

Foi observado apenas um desvio para mais no teste de 1000 registros com chave de 20 bytes, devido ao fato de ser este o único teste cuja fase de ordenação gerou 3 cadeias. Outros testes de 3 e mais cadeias não foram efetuados por causa da já citada dificuldade quanto ao tempo de execução.

O tempo total variou na média de 2 a 2,2 segundos por registro.

Numa tentativa de extensão desses resultados, foi medida a duração de 5000 registros com chaves de 5 bytes, encontrando se 3 horas, o que representa 2,16 segundos por registro e confirma o resultado do parágrafo anterior.

1.8- Influência da Distribuição das Chaves

Foram efetuados testes com chaves em distribuição aleatória, em ordem inversa e em ordem direta.

O tempo de ordenação do arquivo aleatório foi aproximadamente o mesmo do arquivo inverso, devido à predominância do tempo de entrada e saída.

Por outro lado, o tempo para ordenar um arquivo já ordenado foi menor no mínimo em 5%. Isto se deve a ausência da fase de intercalação, uma vez que a fase de ordenação gera uma única cadeia.

1.9- Influência da Compactação das Chaves

A vantagem de compactar a chave é diminuir o tamanho do registro do arquivo reduzido, aumentando o tama - nho das cadeias geradas pela fase de ordenação e diminuindo assim o número de cadeias. Isto deveria diminuir o tempo de entrada e saída de uma forma geral, além de otimizar a fase de intercalação.

Foi observado, no entanto, que nem sempre compensa compactar a chave, pois que o ganho obtido nas outras fases às vezes é menor do que o tempo gasto para compactar as chaves.

Existem, no entanto, dois casos importantes a considerar. Quando o arquivo pode ser diretamente ordenado , sem a fase de intercalação, então a compactação é prejudicial. Quando a compactação permite eliminar a fase de intercalação então há um ganho mínimo de 5% no tempo de execução e a compactação é benéfica.

Para examinarmos tais casos, devemos determinar :

TC = tamanho da chave em bytes
TCC= tamanho da chave compactada em bytes
MD = memória disponível para chaves em bytes
NR = número de registros a ordenar

Calcular o tamanho mínimo da cadeia descompactada :

$$MCD = \frac{MD}{TC+10}$$

e o tamanho mínimo da cadeia compactada :

$$MCC = \frac{MD}{TCC+10}$$

Se $NR < MCD$, ou seja $NR < \frac{MD}{TC + 10}$, então não se deve compactar a chave. Se esta primeira condição não for

atendida, então deve-se examinar se NR < MCC, ou seja NR < $\frac{MD}{TCC+10}$, neste caso devendo-se compactar.

Suponhamos, por exemplo, os seguintes valores :

MD = 5766
TC = 10
TCC= 5

Então calculamos:

$$MCD = \frac{MD}{TC+ 10} = \frac{5766}{10+10} = 288$$

$$MCC = \frac{MD}{TCC+ 10} = \frac{5766}{5+ 10} = 384$$

E concluímos que a chave deve ser compactada se o número de registros estiver entre 288 e 384.

2. Sugestões para Extensão do Trabalho

2.1- Compactação das Chaves

A compactação das chaves mostrou-se muito lenta, sendo inclusive desinteressante na maior parte dos casos.

Vale a pena investir nesta área para diminuir o tempo de execução.

Uma sugestão é a de converter as rotinas de compactação para Assembler.

2.2- Análise de Performance no TI-8080

Uma nova versão do Terminal Inteligente com a CPU INTEL 8080 está sendo desenvolvida. Seria interessante efetuar uma nova análise de performance do SORTI no novo TI.

2.3- Decisão quando a Compactação das Chaves

A análise de performance no novo TI com certeza irá modificar os critérios de compactação sugeridos no item 1.9-Influência da Compactação das Chaves.Esta decisão de compactar ou não a chave poderia passar para o próprio SORTI, liberando o utilizador de executar tal tarefa.

2.4- Saída com o Arquivo de Chaves

O SORTI pode receber uma nova opção que elimine a criação do arquivo de saída, colocando disponível após a ordenação um arquivo contendo apenas as posições das chaves em forma ordenada.

2.5- Continuação após Queda do Sistema

Devido aos grandes tempos de execução observados, seria interessante contar com a possibilidade de continuar uma execução após interrupção por falha de equipamento ou queda de força.

Esta característica pode ser incluída com a gravação em disco dos parâmetros de execução e correspondente adaptação dos módulos de entrada de dados e de controle.

3. Observações Finais :

O tempo de execução do SORTI resultou bastante longo. Isto se deve porém ao interpretador PLTI e ao hardware do Terminal Inteligente com a CPU INTEL-8008, particularmente no que diz respeito ao movimento de dados na memória e a quantidade de memória disponível para o programa. Acreditamos que os tempos cairão sensivelmente se o SORTI for implementado em uma nova versão do Terminal Inteligente com a CPU 8080.

ANEXO

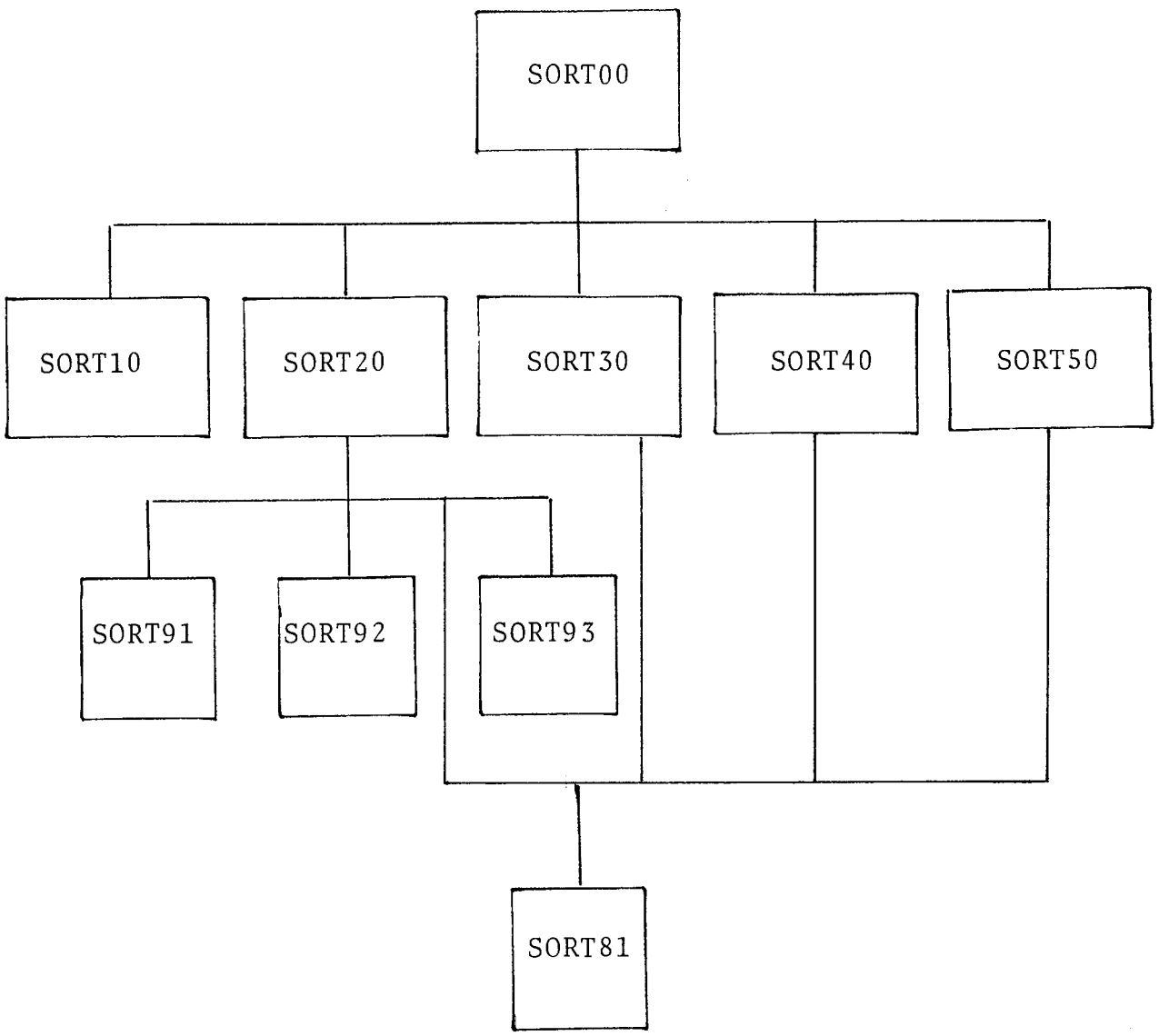
PROGRAMAÇÃO

1. Introdução

Este anexo contém informações a respeito da programação do SORTI. São descritas todas as rotinas, a nível de programa. É uma ferramenta de auxílio à manutenção do programa.

Segue-se a estrutura geral dos módulos do SORTI.

ESTRUTURA DO SORTI



SORT00 - módulo principal
SORT10 - entrada de parâmetros
SORT20 - criação arquivo reduzido
SORT30 - ordenação
SORT40 - intercalação
SORT50- criação arquivo saída
SORT81- acompanhamento de execução
SORT91- compactação de numeros
SORT92- compactação de letras
SORT93- compactação de caracteres

2. Módulo SORT00

2.1- Função

Controlar a execução de cada uma das fases do programa e a passagem de parâmetros entre as mesmas.

2.2- Módulos que o chamam:

Nenhum, pois este é o modulo principal do programa.

2.3- Módulos chamados:

SORT10, SORT20, SORT30, SORT40, SORT50, SORT81

2.4- Arquivos utilizados:

VIDEO.

2.5- Descrição das rotinas internas

a) DSKERR:

Chamada em caso de erro, fornece uma mensagem indicativa no video.

b) DSKEOF:

Chamada em caso de falta de espaço em disco, fornece uma mensagem indicativa no video.

2.6- Descrição do módulo SORT00:

Chama SORT10 para a entrada de parâmetro
Chama SORT20 para a criação do arquivo reduzido
Chama SORT81 indicando o número de registros a ordenar
Chama SORT30 para a ordenação
Se SORT 30 gerar mais de uma cadeia,então chama SORT40.
Chama SORT50 e termina.

3. Módulo SORT10

3.1- Função:

Controlar a entrada, pelo usuário, de dados sobre o arquivo a ordenar e a chave de ordenação.

3.2- Módulos que o chamam:

SORT00.

3.3- Módulos chamados :

Nenhum.

3.4- Arquivos utilizados:

TECLADO, VIDEO, IMPRESSORA

3.5- Descrição das rotinas internas:

a) PERGUNTA

É chamada para cada dado a ser fornecido pelo usuário. No modo conversacional, a rotina indica no vídeo qual o dado a fornecer e aceita-o pelo teclado. No modo não-conversacional, a rotina extrai um campo da área de parâmetros da linguagem de controle. Opcionalmente, imprime todas estas informações.

b) CABEC

Coloca uma linha de cabeçalho no video ou na impressora.

c) ALFABE

Faz a crítica de um dado alfabético.

d) NUMERO

Faz a crítica de um dado numérico.

3.6- Descrição do Módulo SORT10:

Inicialmente chama a rotina CABEC para a geração da linha de cabeçalho.

Com o auxílio das rotinas PERGUNTA, ALFABE e NUMERO, obtém do usuário os dados relativos ao arquivo a ordenar e a chave de ordenação (ver capítulo II, item 3). Os dados são criticados e em caso de erro, cada dado é solicitado novamente.

Finalmente, verifica se o tamanho total da chave e menor ou igual a 64. Caso afirmativo, termina e caso negativo emite mensagem de erro e recomeça.

4. Módulo SORT20:

4.1- Função:

A partir do arquivo de entrada, criar o arquivo reduzido.

4.2- Módulos que o chamam:

SORT00.

4.3- Módulos chamados :

SORT81, SORT91, SORT92, SORT93.

4.4- Arquivos utilizados :

ENTRA - arquivo de entrada do SORTI
REDUZ - arquivo reduzido (saída)

4.5- Descrição das rotinas internas

a) READ$ENTRA:
Lê o arquivo ENTRA, deblocando seus registros e atualizando contadores de blocos e de registros dentro do bloco.

b) WRITE$REDUZ :
Grava o arquivo REDUZ, blocando seus registros e completando as informações de números de bloco e registro através dos contadores mantidos pela rotina READ$ENTRA.

4.6- Descrição do módulo SORT20:

Chama SORT81 para comunicar seu início ao usuário.
Processa cada registro do arquivo ENTRA, analisando cada um dos campos da chave de cada registro: se for especificado ordenação descendente, então cada byte do

campo é complementado e se for especificado compactação, então o campo é compactado com auxílio de uma das rotinas SORT91, SORT92, SORT93, dependendo se o campo contém números, letras ou caracteres, respectivamente.

Cada registro após processado é gravado no arquivo REDUZ e ao fim do arquivo de entrada grava-se um registro de fim de arquivo na saída e chama-se SORT81, indicando ao usuário o fim da fase 2.

5. Módulo SORT30:

5.1- Função:

Ordenar registros do arquivo REDUZ, gravando cadeias de registros ordenados no arquivo CADE1.

5.2- Módulos que o chamam:

SORT00.

5.3- Módulos chamados :

SORT81.

5.4- Arquivos utilizados :

REDUZ- arquivo reduzido (entrada)
CADE1- arquivo de cadeias(saída)

5.5- Descrição das rotinas internas:

a) INPUT :

Lê o arquivo REDUZ, deblocando seus registros. Chama SORT81 em intervalos fixos de número de registros lógicos obtidos, para que o usuário possa acompanhar o andamento da fase.

b) OUTPUT :

Grava o arquivo CADE1, blocando seus registros e mantendo contadores de blocos e de registros dentro do bloco.

c) FIMCADEIA :

Grava no arquivo CADE1 um registro de fim de cadeia, contendo o número do bloco e do registro onde se encontra o fim da cadeia anterior. Para que

isto possa ser feito, salva os números do bloco e do registro do fim de cadeia gravado.

5.6- Descrição do Módulo SORT30:

Ver algorítimo de ordenação (capítulo III, item 4.5).

6. Módulo SORT40:

6.1- Função :

Intercalar as cadeias geradas pelo módulo SORT30.

6.2- Módulos que o chamam:

SORT00.

6.3- Módulos chamados :

SORT81.

6.4- Arquivos utilizados :

CADE1- arquivo de trabalho de cadeias — usado ora como entrada, ora como saída.

CADE2- arquivo de trabalho de cadeias — usado ora como saída, ora como entrada.

6.5- Descrição das rotinas internas :

a) READ$INP:
Consulta a variável ARQ$INP, que indica dentre os arquivos CADE1 e CADE2 qual está sendo usada como entrada no momento, e faz um acesso de leitura.

b) WRITE$OUT:
Consulta a variável ARQ$INP e faz um acesso de gravação.

c) INPUT:
Lê o arquivo de entrada com auxílio da rotina READ $INP e debloca seus registros. Analisa o registro lógico obtido, verificando se é fim de cadeia; caso afirmativo, passa a ler a próxima cadeia (se esta existir). Chama SORT81 em intervalos fixos do número de registros lógicos obtidos, para que o

usuário possa acompanhar o andamento da fase.

d) OUTPUT :
Grava o arquivo de saída com auxílio da rotina WRITE$OUT, blocando seus registros. Mantém contadores de blocos e de registros dentro do bloco.

e) FIMCADEIA:
Grava no arquivo de saída, com auxílio da rotina WRITE$OUT, um registro de fim de cadeia contendo o número do bloco e do registro onde se encontra o fim da cadeia anterior. Para que isto possa ser feito, salva os números do bloco e do registro do fim de cadeia gravado.

6.6- Descrição do módulo SORT40:

Ver algorítimo de intercalação (capítulo III, item 5.5).

7. Módulo SORT50

7.1- Função:

Gravar o arquivo de saída do SORTI.

7.2- Módulos que o chamam :

SORTOO.

7.3- Módulos chamados:

SORT81.

7.4- Arquivos utilizados :

ENTRA- arquivo de entrada do SORTI
SAÍDA- arquivo de saída do SORTI
CADE1/CADE2- usado como entrada o que contiver uma única cadeia ordenada, gravada pelo SORT30 , ou pelo SORT40.

7.5- Descrição das rotinas internas:

a) INPUT:
Lê, dentre os arquivos CADE1 e CADE2, apenas o que contiver uma única cadeia ordenada. Debloca os registros. Mantém contadores de blocos e de registros para uso da rotina OUTPUT. Chama SORT81 em intervalos fixos de números de registros lógicos obtidos , para que o usuário possa acompanhar o andamento da fase.

b) OUTPUT:
A partir do apontador contido no registro vencedor da árvore de comparações, consulta diretamente o arquivo ENTRA, obtendo o registro original correspondente. A posição na memória que este registro ocupará é indicada pelos contadores mantidos pela rotina INPUT.

c) FIMCADEIA:
Grava no arquivo de SAIDA todos os registros colocados na memória pela rotina OUTPUT.

7.6- Descrição do módulo SORT50:

Ver algorítimo de criação do arquivo de saída (capítulo III, item 6.5).

8. Módulo SORT81

8.1- Função:

Enviar mensagens ao vídeo informando ao usuário o andamento do SORTI.

8.2- Módulos que o chamam:

SORT00, SORT20, SORT30, SORT40, SORT50.

8.3- Módulos chamados:

Nenhum.

8.4- Arquivos utilizados:

VIDEO.

8.5- Descrição das rotinas internas :

a) CONVERT:
Faz a conversão de números binários para dígitos ASCII.

8.6- Descrição do módulo SORT81:

O módulo realiza quatro funções, dependendo do valor do parâmetro enviado no momento da chamada.

Parâmetro = 1 : será enviada ao vídeo uma mensagem de início de fase.
Parâmetro = 2 : será incrementado o contados de registros processados e enviada ao vídeo uma mensagem informando o novo valor
Parâmetro = 3 : será enviada ao vídeo uma mensagem de fim de fase
Parâmetro = outro valor : indica que o módulo está sendo chamado pelo SORT00 para informar o número total de registros, a quantida

de de memória disponível e o intervalo em número de registros para chamada com parâmetro igual a 2.

9. Módulo SORT91

9.1- Função :

Compactar campos do tipo numérico.

9.2- Módulos que o chamam :

SORT2Ø.

9.3- Módulos chamados :

Nenhum.

9.4- Arquivos utilizados :

Nenhum.

9.5- Descrição das rotinas internas :

Não há.

9.6- Descrição do módulo :

Ver algorítimo no capítulo III, item 3.4.

10. Módulo SORT92

10.1- Função:

Compactar campos do tipo alfabético.

10.2- Módulos que o chamam :

SORT20.

10.3- Módulos chamados :

Nenhum.

10.4- Arquivos utilizados :

Nenhum.

10.5- Descrição das rotinas internas:

Não há.

10.6- Descrição do módulo :

Ver algorítimo no capítulo III, item 3.4.

11. Módulo SORT93

11.1- Função:

Compactar campos que contenham quaisquer caracteres ASCII.

11.2- Módulos que o chamam :

SORT20.

11.3- Módulos chamados:

Nenhum.

11.4- Arquivos utilizados :

Nenhum.

11.5- Descrição das rotinas internas:

Não há.

11.6- Descrição do módulo :

Ver algorítimo no capítulo III, item 3.4.

B I B L I O G R A F I A

1- SCHICK, Thomas- Disk File Sorting. Communications of the ACM , Volume 6, Number 6, 330-331 e 339, June 1963.

2- FRIEND, Edward Harry-Sorting on Electronic Compu ter Systems. Journal of the ACM, Number 3, 134 - 168, July 1956.

3- GASSNER,Betty Jane- Sorting by Replacement Selecting. Communications of the ACM,Volume 10, Number 2, 89-93, February, 1967.

4- KNUTH, Donald E.-Sorting and Searching. The Art of Computer Programming, Volume 3, Addison-Wesley, USA,1973.722p.

5- FERGUSON, David E.- Buffer Allocation in Merge-Sorting. Communications of the ACM, Volume 14, Number 7, 476-478, July 1971.

6- RICH, Robert P.-Internal Sorting Methods Ilustrated with PL/I Programs. Prentice-Hall, New Jersey, USA,1972.

7- LORIN, Harold. Sorting and Sort Systems.Addison -Wesley, USA,1975.

8- System 2200 Utilities Volume II. Wang Laboratories Inc., 1974.

9- BLACK, Neville A.- Optimum Merging focom Mass Storage. Communications of the ACM, Volume 13, Number 12,December 1970.

10- HUBBARD, George H.- Some Characteristics of Sorting in Computing Systems Using Random Access Storage Devices. Communications of the ACM,Volume 6,Number 5, May,1973.