Resumo 2 - EDII

Aluno: *Nathann Zini dos Reis*

Matrícula: *19.2.4007*

Aula 8 - Intercalação balanceada de vários caminhos

A maior diferença entre a ordenação externa e a ordenação interna é em relação ao custo operacional referente ao custo de acesso à memória secundaria ser maior ao custo de acesso à memória primária. E, para realização da ordenação externa, além de acessá-la para obter o dado/informação, ela também é acessada para gravar esse dado de volta na memória secundaria de maneira ordenada.

O método mais importante na ordenação externa é o método de intercalação: consiste em combinar alguns blocos diferentes ordenados em um bloco só, para que diminua a quantidade de custo.

>ESTRATÉGIA GERAL<

dividir o arquivo em blocos de tamanhos que caibam em memória interna, ordenar na memória interna e depois fazer a intercalação dos blocos ordenados da memória externa, até que todos os blocos se tornem um bloco só ordenado;

Intercalação Balanceada de Vários Caminhos

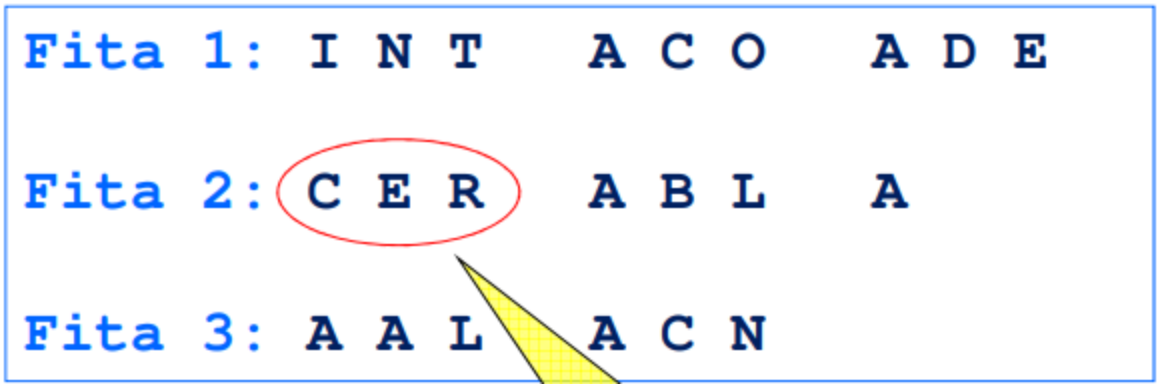
Exemplificando a funcionalidade:

INTERCALACAOBALANCEADA

- Considerações:
  - memória interna com capacidade para três itens;
  - seis unidades disponíveis de fita magnética.

1º passo: é quebrar o arquivo em blocos de tamanho disponível de memória, no caso, quebrar a informação em vetores de 3 posições (letras)

Para cada bloco, é transferido para memória interna, feito a ordenação em memória interna de cada um dos blocos e cada bloco é salvo em uma fita (nesse caso, temos 3 fitas de entrada e 3 fitas de saída). Logo, as fitas de entradas ficariam, após essa primeira fase, desse modo:



OBS: Consideraremos cada fita como sendo um arquivo externo auxiliar que será utilizado

como fita;

2º passo: é a fase de intercalação que consistem em, utilizando a memória interna disponível, vai fazer a intercalação do primeiro bloco de cada fita. Funciona da seguinte maneira:

- Primeiro a memória é preenchida com o primeiro item de cada bloco das fitas disponíveis;

- Depois, é percorrida a memória para achar o elemento com a menor chave, esse elemento então é passado para a primeira fita de saída disponível.
- Após a transferência, a posição do vetor em memória principal é preenchida com o próximo item da fita de onde o valor transferido por último pertence; Quando não há mais valores no bloco de alguma fita, o espaço referente à fita cujo o bloco já foi inteiro transferido, é "inutilizado"
- Após a finalização desse processo, a primeira fita de saída é preenchida, de maneira ordenada, com os valores dos primeiros blocos de cada fita de entrada;
- e repete o mesmo processo para os demais blocos das fitas de entrada, armazenando nas outras fitas de saída;
- Após finalizado a intercalação com todos os blocos das fitas de entradas, obtém-se fitas de saídas ordenadas;

- Resultado da 1ª passada sobre o arquivo na fase de intercalação:

```
Fita 4: A A C E I L N R T
Fita 5: A A A B C C L N O
Fita 6: A A D E
```

- Observa-se que cada fita agora possui apenas 1 bloco (nesse exemplo) de dados ordenados;
- Agora, repete-se o mesmo processo para essas fitas (que passam a ser fitas de entradas) e salva o bloco intercalado na fita que anteriormente era de entrada e que agora passa a ser de saída;

- Repetindo o processo de intercalação nos blocos formados após a 1ª passada sobre o arquivo, tem-se o arquivo ordenado (2ª passada sobre o mesmo):

```
Fita 1: A A A A A A A B C C C D E E I L L N N O R T
Fita 2:
Fita 3:
```

OBS: Enquanto tiver mais de um bloco de itens por fita, repete-se o processo de intercalação (alternando as fitas de entrada e saída) até obter apenas um bloco com todos os itens ordenados;

A complexidade é dada por:

$$P(n) = \log_f (n/m)$$

Em que:

- Seja **n** o número de registros do arquivo.
- Seja **m** o número de palavras possíveis na memória interna, sendo uma palavra igual a um registro do arquivo.

Logo, a primeira fase produz **n/m** blocos ordenados.

- Seja P(n) o número de passadas na fase de intercalação.
- Seja f o número de fitas utilizadas em cada passada.

Alternativa para q quantidade de fitas na intercalação:

Uma outra maneira de realizar a intercalação é com o método de f + 1 fitas em que temos f fitas de entrada e 1 fitas de saída:

> Nesse método, diferentemente do anterior, as fitas de entrada e saída não se alternam; Os blocos ordenados gerados a partir das fitas de entradas são armazenados na única fita de saída, depois cada um desses blocos é remanejados para cada uma das fitas de entradas e realiza o processo novamente até obter-se apenas um bloco com todos itens de modo alternado;

Implementação por meio de Substituição por Seleção

Analogamente à maneira anterior, em sua essência, este método consiste em fazer o mesmo processo de intercalação, usando a memória interna disponível, para armazenar, de forma ordenada, nas fitas disponíveis, os itens do arquivo externo;

Uma maneira mais eficiente de fazê-lo é utilizando as filas de prioridades. Dessas, uma que se encaixa perfeitamente é a prioridade Heap

> A prioridade Heap consiste em guardar em um vetor pais e filhos em que as chaves dos pais sejam SEMPRE menores que as chaves dos filhos. A relação dos pais e filhos no vetor se comporta da seguinte maneira:
>
> - o filho esquerdo é v[2*i+1] e o filho direito é v[2*i+2]
> - o pai é v[(i-1)/2]

Uma vantagem de usar o heap durante a intercalação, não terá que fazer uma busca linear para achar o menor elemento, mas reconstituir o heap e pegar o primeiro elemento. A complexidade deixa de ser O(n) e passa a ser Log n;

Uma desvantagem é que, ao montar e reconstituir o heap, não pode-se mais basear a posição do item no vetor da memória interna para saber a qual fita aquele item em questão é referente. A partir disso, torna-se necessário guardar além do item na posição do vetor, guardar também a qual fita ele pertence para conseguir pegar o próximo valor mantendo a ordenação;

Aula 9 - Substituição por seleção e intercalação polifásica

Continuando o último assunto introduzindo no resumo da aula anterior, foi mencionado um outro método de intercalação balanceada que utiliza filas de prioridades. Esse método é a substituição por seleção. Especificadamente, foi mencionado a utilização da fila de prioridade Heap como uma alternativa mais eficaz para o processo de intercalação;

1º Passo: Geração dos blocos iniciais ordenados

Nesse método, será utilizado a prioridade heap para ordenar os elementos que estiverem na memória principal. A definição de quantidade de itens por blocos vai depender de como vai se dar o processo de geração para cada conjunto de valor.

Inicialmente, os valores são ordenados (montando o heap). Depois de montado, o primeiro valor do vetor da memória interna, que será o menor dentre os valores, será transferido para a primeira fita e será substituído na memória interna pelo próximo elemento do arquivo externo. Nesse momento é feito uma análise:

- O valor que está entrando é maior que o valor que ele está substituindo? Se sim, apenas substitui os valores e reconstitui o heap e faz o mesmo processo jogando o primeiro valor da memória interna para a primeira fita.

  Mas, caso esse valor que estiver entrando seja MENOR que o valor que ele está substituindo, o valor que está entrando deverá ser marcado como MAIOR (exemplo na aula dado como *) que os demais e então é refeito o heap e repetido o processo.

  No momento em que, na memória interna, tem-se TODOS os valores do vetor marcados como maiores, então é finalizado aquele bloco (que estava sendo colocado na fita 1), os valores são desmarcados, e o processo se inicia do começo, porém agora salvando na próxima fita e assim sucessivamente até que todos os valores tenham sido inseridos em blocos iniciais ordenados;

Esse método é interessante pois gera blocos iniciais com, possivelmente, mais valores e menor quantidade de blocos em si, que facilita e acelera o processo de intercalação.

Obs: O pior caso é quando o arquivo a ser ordenado (ascendentemente) está ordenado descendentemente. Nesse caso, será gerado a mesma quantidade de blocos de mesmo tamanhos que a ordenação interna. Logo, é o pior caso;

Apesar das vantagens, esse método é indicado para quando se tem um número de fitas (f) maior ou igual a 8 (f ≥ 8), pois obtém-se uma complexidade Log 2 f comparações para obter o menor item;

- Para valores pequenos de **f**, não é vantajoso utilizar seleção por substituição para intercalar blocos, já que o menor item pode ser obtido por **f-1** comparações.
- Quando $f \geq 8$, o método é considerado adequado, realizando $\mathbf{\log_2 f}$ comparações para se obter o menor item.

Intercalação Polifásica

A intercalação polifásica surgiu a partir da necessidade de minimizar ou tratar os problemas gerados com os métodos de intercalação por vários caminhos.

Os problemas são:

- Problemas com a intercalação balanceada de vários caminhos:
  - Necessita de um grande número de fitas, fazendo várias leituras e escritas entre as fitas envolvidas.
    - Para uma intercalação balanceada de **f** caminhos são necessárias, geralmente, **2f** fitas.
    - Alternativamente, pode-se copiar o arquivo de uma única fita de saída para **f** fitas de entrada, reduzindo o número de fitas para **f + 1**.
  - Há um custo de uma cópia adicional do arquivo.

Principalmente esse ultimo item. Ao finalizar uma rodada de blocos de ordenação, eles são jogados para as fitas de entradas e depois para as fitas de saídas, nesse momentos, temos os mesmos valores em ambas as fitas de entrada e de saída, gerando essa mencionada cópia adicional do arquivo que, por sua vez, gera um custo operacional.

Processo de Intercalação Polifásica

Os blocos são substituídos de maneira desigual entra as fitas , exceto uma fita que sempre deverá estar vazia de inicio (qualquer que seja a fita, porém é obrigatório ter uma fita vazia que representará a fita de saída). Similar ao processo de f + 1 fitas, será feita o processo de intercalação entre as fitas com os blocos e os blocos intercalados serão salvos na fita que estava vazia inicialmente (fita de saída) até que uma das duas fitas que incialmente continha valores fique vazia. Essa fita que agora está vazia se torna a fita de saída e o processo é repetindo salvando os blocos nessa nova fita de saída. E isso é feito até que obtenha-se apenas um único bloco final ordenado;

Exemplo de execução:

- Blocos ordenados obtidos por meio de seleção por substituição:

```
Fita 1: I N R T     A C E L   A A B C L O
Fita 2: A A C E N   A A D
Fita 3:
```

- Intercalação-de-2-caminhos das fitas 1 e 2 para a fita 3:

```
Fita 1: A A B C L O
Fita 2:
Fita 3: A A C E I N N R T   A A A C D E L
```

- Intercalação-de-2-caminhos das fitas 1 e 3 para a fita 2:

```
Fita 1:
Fita 2: A A A A B C C E I L N N O R T
Fita 3: A A A C D E L
```

21

UFOP – BCC203 – Prof. Guilherme Tavares de Assis

Intercalação Polifásica

- Intercalação-de-2-caminhos das fitas 2 e 3 para a fita 1:

```
Fita 1: A A A A A A A B C C C D E E I L L N N O R T
Fita 2:
Fita 3:
```

Por ser feita em várias fases, o nome é dado por polifásica. As fases não envolvem todos os blocos e nenhuma cópia entra fitas é realizadas (criar fitas de saídas);

O pior caso acontece quando os blocos estão ordenados igualmente entre as fitas, pois em algum momento, terá mais de 1 bloco em uma única fita. Para tratar isso, terá que mover um bloco dessa fita (normalmente o primeiro bloco) para outra fita para continuar a fazer a intercalação.

Analisando a Intercalação Polifásica, é interessante utilizá-la para quantidades pequenas de fitas (f < 8), pois é ligeiramente melhor que a intercalação balanceada.

- Análise:
  - A análise da intercalação polifásica é complicada.
  - O que se sabe é que ela é ligeiramente melhor do que a intercalação balanceada para valores pequenos de **f**.
  - Para valores de **f** > 8, a intercalação balanceada de vários caminhos pode ser mais rápida.

Aula 10 - QuickSort Externo

Similarmente ao QuickSort Interno, utiliza o paradigma de divisão e conquista:

> Escolhendo um valor para ser o pivô e separa o vetor em duas partes: menores que o pivô e maiores que o pivô. E o processo é repetido para cada uma das novas partes geradas e faz isso recursivamente até que obtenha uma partição de tamanho que caiba em memória interna para que seja feita o processo de ordenação interno dela.

Diferentemente dos métodos de intercalação, o arquivo é ordenado dentro do próprio arquivo, não precisando de "fitas" para passar os dados do arquivo para memória interna e realizar o processo ali dentro.

Entretanto, os pivôres da ordenação são armazenados dentro da memória interna, gerando um espaço de Log n (n sendo numero total de elementos do arquivo) para salvar os pivores na memoria interna.

As partições serão geradas até que elas tenham tamanhos menores que o máximo para serem armazenadas em memória interna para aplicar o algorítimo de ordenação interna nela, ou então até que a partição tenha 1 valor. Partições vazias serão ignoradas.

Processo do quicksort externo:

O processo é controlado por 4 apontadores, 2 referentes à leitura e 2 à escrita, dois deles referentes aos valores iniciais do arquivo e os outros dois referentes as posições finais do arquivo (Li, Ei, Ls & Es) e também dois valores que indicam o tamanho da partição (Esq & Dir) e dois limites (Linf e Lsup) que controla a área da memória interna onde serão armazenados os valores para a ordenação externa e são inicialmente iniciados com -infinito e + infinito.

Basicamente funciona da seguinte maneira:

Os valores acima são iniciados nas extremidades da partição. A memória interna é preenchida a partir dos valores alternados das extremidades. Sempre que um valor da extremidade esquerda é lido, o Li é incrementado. Analogamente, quando um item da extremidade direita é lido, o Ls é decrementado.

Quando algum valor é escrito na extremidade esquerda, o Ei é incrementado. Analogamente, quando um valor é escrito na extremidade direita, o Es é decrementado. O valor de Ei nunca poderá ser maior que o valor de Li, o mesmo para Es e Ls. Caso o Ei e Li sejam igual e uma leitura, pela alternância, deverá ser feita na extremidade inicial da partição, a fim de manter a regra de que Ei não pode ser maior que Li, a alternância é quebrada e é feita a leitura na outra extremidade.

A memória interna representa um vetor de pivô. Este vetor deverá sempre estar ordenado.

Os valores serão levados da partição para memória interna até que os valores de Li e Ls se cruzem. Ao ser feito isso, os valores restante no pivô serão apenas escritos na partição e esses valores estarão ordenados entre si. Quando o processo termina, ele retorna dois valores: i* e j*; i* representa o final da primeira subPartição que deverá ser passada recursivamente para fazer o mesmo processo nela, e o j* representa o inicio da segunda subpartição.

Nota-se que os valores medianos entre essas duas subpartições, que sobraram anteriormente no vetor pivô, já se encontram ordenados e todos são maiores que a primeira subpartição e menores que a segunda subpartição.

O processo de leitura e escrita nas partições, tão como os valores de i* e j*, são feitos seguindo algumas regras:

- Primeiro é lido alternadamente os valores até que o vetor Pivô falte 1 valor para ficar completo. Esse valor será a chave C e ela será aplicada à algumas regras;

- Ao ler o TamArea-ésimo registro, cuja chave é **C**:
  - **C** é comparada com **Lsup** e, sendo maior, **j** recebe **Es** e o registro é escrito em $\mathbf{A_2}$;
  - caso contrário, **C** é comparada com **Linf** e, sendo menor, **i** recebe **Ei** e o registro é escrito em $\mathbf{A_1}$;
  - Caso contrário (**Linf** $\leq$ **C** $\leq$ **Lsup**), o registro é inserido na área de memória interna.

- Quando a área de memória enche, deve-se remover um registro da mesma, considerando os tamanhos atuais de $\mathbf{A_1}$ e $\mathbf{A_2}$.
  - Sendo **Esq** e **Dir** a 1ª e a última posição de **A**, os tamanhos de $\mathbf{A_1}$ e $\mathbf{A_2}$ são, respectivamente, ($\mathbf{T_1 = Ei - Esq}$) e ($\mathbf{T_2 = Dir - Es}$).
  - Se (**T1 < T2**), o registro de menor chave é removido da memória, sendo escrito em **Ei** ($\mathbf{A_1}$), e **Linf** é atualizado com tal chave.
  - Se (**T2** $\leq$ **T1**), o registro de maior chave é removido da memória, sendo escrito em **Es** ($\mathbf{A_2}$), e **Lsup** é atualizado com tal chave.

O melhor caso gera uma complexidade de O(n/b), quando todos os valores já se encontram ordenados no arquivo. Em que b equivale a quantidade de itens lidos de uma vez no arquivo, no caso desse exemplo é lido 1 unico item por vez. Logo, complexidade O(n);

O pior caso não acontecerá pois as subpartições, nesse código, são todas de tamanhos uniformes. E o pior caso é quando os tamanhos das sub partições são desiguais;

O caso que vai mais acontecer é o caso médio, que tem uma complexidade de O(n * Log(n/tamArea)). Logo, o tamanho da área

interna referente ao pivô influencia diretamente na complexidade do código.