UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP

Ciência da Computação

ESTRUTURA DE DADOS II

RESUMO ORDENAÇÃO EXTERNA

Gabriel Mace do Santos Ferreira

Marcus Vinícius Souza Fernandes

Ouro Preto

2021

Ordenação Externa, o que é?

A ordenação externa tem como objetivo a implementação de técnicas de ordenação quando os dados a serem ordenados não cabem em memória principal.

Podemos ressaltar que os métodos de ordenação externa são diferentes dos de ordenação interna, uma vez que o número de acessos à memória externa tendem a ser reduzidos.

A eficiência de uma técnica de ordenação externa é medida por meio do número de comparações efetuadas, bem como o número de movimentações feitas, além disso ao se tratar de elementos na memória externa são consideradas o número de transferências entre a memória principal e secundária.

Em geral, as técnicas de ordenação externa são mais custosas se comparadas a ordenação interna, visto que o acesso à memória externa é mais custoso do que o acesso à memória interna. Os métodos de ordenação externa variam de acordo com o estado atual da tecnologia, visto que o acesso aos dados pode ser modificado de acordo com a tecnologia que se está lidando.

Intercalação balanceada

A intercalação balanceada de vários caminhos é o método mais importante da ordenação externa, usualmente utilizado como uma operação auxiliar. O método consiste em combinar diferentes blocos ordenados, de forma que ao final da operação, o arquivo se encontre completamente ordenado.

Funcionamento usual da intercalação:

1. Dividir o arquivo em blocos, de forma a caberem na memória principal.
2. Ordenar cada bloco na memória interna, utilizando métodos de ordenação para a memória principal.
3. Intercalar os blocos, fazendo várias passadas para os arquivos. Buscando assim, reduzir o número de passadas ao mínimo.

Processo da intercalação:

1. Lê-se o primeiro registro de cada fita.
2. Retira-se a menor chave do registro e a armazena em uma fita de saída.
3. Leitura de um novo registro do arquivo em que houve a remoção. Ao ler um terceiro registro de um dos blocos, a fita se torna inativa, se tornando ativa novamente quando os terceiros registros das demais fitas são lidos. Neste passo, na fita de saída se encontra um bloco de nove registros.
4. Repetição do processo para os demais blocos.

O cálculo para o número de passadas necessárias para ordenar um determinado arquivo de tamanho arbitrário pode ser dado:

- Seja **n** o número de registros no arquivo.
- Seja **m** o número de palavras possíveis na memória interna, sendo uma palavra igual à um registro do arquivo.
- A primeira etapa 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.

Logo, obtemos: $\mathbf{P(n) = \log_f (n/m)}$.

Substituição por seleção

A implementação do método de substituição por seleção pode ser realizada através de filas de prioridades, onde as duas etapas (divisão e intercalação) podem ser criadas de forma mais eficiente e o menor item existente será substituído pelo próximo item da fita de entrada.

Idealmente para a implementação utiliza-se a estrutura “heap”, no entanto os itens do heap deverão possuir campos referentes a seus valores, bem como uma variável para o arquivo de origem do mesmo.

O método é semelhante a intercalação balanceada normal, no entanto utiliza um heap para reduzir o número de comparações durante a fase de intercalação.

Durante a fase de intercalação os itens pegos são organizados em um heap, após a retirada, assim como descrito anteriormente, o menor item existente é substituído pelo próximo item da fila de entrada para ser comparado aos demais itens e reorganizar o heap criado.

Primeira fase:

São gerados os blocos de ordenação iniciais, por meio do método de ordenação interna ou substituição por seleção, devido a necessidade de ordenar e montar os blocos essa fase pode demorar um longo tempo dependendo da quantidade de elementos presentes.

Caso o elemento inserido no heap seja menor que o elemento que “saiu” do mesmo, ele deve ser marcado de forma a ser considerado maior que os elementos restantes neste heap.

Quando todos os elementos dentro do heap se encontram marcados, é necessário retirar as marcas e inseri-las ordenadamente em um novo bloco.

Segunda fase:

Intercalação dos blocos utilizando a intercalação balanceada de vários com **2f**, ou seja, os blocos são distribuídos entre várias fitas ou intercalação balanceada de vários caminhos com **f+1** onde os blocos são intercalados em uma única fita.

Esse método de implementação é vantajoso apenas quando **f** é maior ou igual a 8, caso contrário podem ser utilizados os métodos de ordenação em memória interna.

Intercalação polifásica

A intercalação polifásica foi implementada de forma a solucionar os problemas da intercalação balanceada, tais como: A necessidade um grande número de fitas realizando leituras e escrituras, sendo que uma intercalação de **f** caminhos são necessários **2f** fitas e mesmo utilizando **f+1** fitas ainda há o custo de uma cópia adicional do arquivo em questão.

É extremamente difícil saber o número de passadas completas realizadas durante a intercalação polifásica, no entanto é possível inferir que ela é ligeiramente melhor que a intercalação balanceada para valores de **f** menores ou iguais à 8. Caso os valores sejam maiores que 8 é ideal utilizar a intercalação balanceada de vários caminhos.

As etapas para o funcionamento da intercalação polifásica:

1. Os blocos ordenados são distribuídos desigualmente(visando garantir que uma das fitas de entrada ainda contenha elementos, evitando que apenas uma das fitas contenha elementos) entre as fitas disponíveis, deixando uma fita vazia para funcionar como a fita de saída.
2. A intercalação entre os blocos ordenados é executada até que uma das fitas de entrada fique vazia, essa fita vazia será utilizada como a próxima fita de saída.

Dessa forma não há o custo de criar fitas adicionais para receberem a saída da intercalação. É importante ressaltar que a intercalação é realizada em múltiplas fases que não envolvem todos os blocos e durante o processo não há nenhuma cópia direta entre as fitas realizadas. Embora a implementação da intercalação polifásica seja simples, a distribuição inicial dos blocos ordenados entre as fitas é bem delicada.

Quicksort externo

O quicksort foi proposto por Monard no ano de 1980, o algoritmo se caracteriza por fazer uso do paradigma de divisão e conquista, ou seja, um pivô é escolhido e a memória secundária é dividida baseada na sua relação de grandeza com o pivô. Vale lembrar que a ordenação do arquivo ocorre in situ, portanto não é necessária a criação de uma memória secundária auxiliar.

Embora não seja necessário criar uma memória secundária auxiliar, são utilizados espaços da memória para armazenar o pivô que pode ser um conjunto de elementos ordenados. A partição da memória interna possui uma complexidade O(log n).

A partição da memória secundária (arquivo) ocorre de acordo com a relação com o pivô, dado que os elementos menores ficam em uma das divisões e os elementos maiores que o pivô ficam em outra divisão.

O armazenamento do pivô na memória interna ocorre em uma área de tamanho **j-i-1** maior ou igual à 3.

Funcionamento das chamadas recursivas:

- Inicialmente deve ser ordenado o subarquivo de menor tamanho.
- Subarquivos vazios ou com um único registro são ignorados.
- Caso o arquivo de entrada tenha no máximo **j-i-1** registros a ordenação ocorre em um único passo.

Variáveis presentes:

- **Esq:** Limite de posição.
- **Dir:** Limite de posição.
- **Li:** Apontador de leitura inferior.
- **Ei:** Apontador de escrita inferior.
- **Ls:** Apontador de leitura superior.
- **Es:** Apontador de escrita superior.
- **Linf:** Limite inferior ou **Ri**.
- **Lsup:** Limite superior ou **Rj**.
- ***i:** Valor retornado pela partição.
- ***j:** Valor retornado pela partição.

Inicialmente as leituras prévias a área da memória interna a ser preenchida (**TamArea** - 1) são efetuadas alternando entre as extremidades do arquivo A (memória secundária) e armazenados na memória interna, ficando uma única posição do pivô não preenchida.

Na próxima leitura, a chave do elemento lido será comparado com a chave presente no limite superior (**Lsup**), caso esse seja maior, o maior elemento do pivô

(**j**) recebe o valor de **Es**. Caso contrário, o elemento é comparado com a chave do limite superior (**Linf**) e analogamente se for menor que o limite inferior, o menor valor do pivô (**i**) recebe o valor de **Ei**. Curiosamente se o elemento lido estiver no intervalo **Linf** <= **Registro** <= **Lsup**, o registro simplesmente será inserido na área da memória interna.

Durante esse processo, se em algum momento a área da memória enche, é necessário a remoção de um registro, sendo considerado o tamanho atual dos subarquivos(**A_1 e A_2**). Contando que as variáveis **Esq** e **Dir** armazenem a primeira e última posição do arquivo **A** respectivamente, o tamanho de **A_1** é definido por Tamanho = **Ei** - **Esq** e o tamanho de **A_2** é definido por Tamanho = **Dir** - **Es**.

Caso o tamanho de **A_1** seja menor que o tamanho de **A_2**, o registro de menor chave é removido da área da memória e escrito em **Ei** (**A_1**), e **Linf** é atualizado com a chave do registro. Caso contrário, é escolhido o registro com maior chave é removido da área da memória e inserido em **Ei** (**A_2**), e **Lsup** é atualizado com a chave do registro.

No que tange a análise de casos, destaca-se como o melhor deles **O(n/b)**, onde **n** é o número de registros e **b** o tamanho do bloco de leitura ou gravação do SO. Este caso ocorre quando o arquivo de entrada já se apresenta ordenado. Um pior caso seria **O(n^2/TamArea)**, quando os tamanhos das partições são inadequados. Por fim, o caso médio **O(n/b * log(n/TamArea))**, o mais provável de ocorrer.