Fabio Henrique Alves Fernandes - 19.1.4128

Ordenação Externa

A ordenação externa se torna interessante quando temos arquivos que seu tamanho são maiores que os suportados pela memória principal. Os métodos de ordenação externa diferem muito dos vistos em ordenação em memória interna, pois temos que diminuir o número de acessos às unidades externas. Como os arquivos são armazenados como um arquivo sequencial, apenas um registro pode ser acessado a cada momento, o que difere muito da ordenação em um vetor, tornando métodos de ordenação em memória principal inviáveis para o uso em memória secundária. Um dos métodos mais importantes na ordenação externa é o de intercalação, porém é um processo muito custoso. Com isso, devemos então reduzir o número de intercalações durante a ordenação.

A melhor estratégia para se ordenar um arquivo é quebrá-lo em blocos que caibam na memória principal; ordenar todos esses blocos um por um dentro da memória principal; e intercalar os blocos já ordenados.

Intercalação Balanceada de Vários Caminhos

De acordo com o tamanho do arquivo, pegamos as chaves dos registros presentes, dividimos ele em blocos, de acordo com o tamanho disponível na memória interna, e usamos unidades de memória disponíveis: fitas, para guardar temporariamente tais blocos.

Dividimos o processo em duas fases. A primeira fase de uma ordenação é onde eu crio os blocos e os ordeno, já a segunda é onde eu intercalo os blocos já ordenados.

Durante a primeira fase, quando colocamos em memória principal os registros necessários, usamos métodos de ordenação em memória principal para ordenar cada bloco.

Na segunda fase: a fase de intercalação, pegamos o primeiro de cada fita e guardamos na memória principal. Logo após, faço uma análise do menor elemento presente entre esses e o coloco nas fitas de saída, sempre substituindo o elemento retirado pelo seu sucessor da sua fita de entrada, até o final do bloco, fazendo o mesmo com os blocos subsequentes. Quando todas as fitas de saídas estiverem ordenadas, faremos mais uma passada, para ordenar novamente as fitas, até que o arquivo fique completo.

Exemplo:
Levaremos em consideração I N T E R C A L A C A O B A L A N C E A D A

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

➔ Primeira passada nos blocos

Fita 4: A A C E I L N R T (Primeira parte de cada fita)
Fita 5: A A A B C C L N O (Segunda parte de cada fita)
Fita 6: A A D E (Terceira parte de cada fita)

Segunda passada nos blocos

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:

Nesse exemplo foram utilizadas 2f fitas, mas é possível usar apenas f+1 fitas, sendo que nesse método são feitas mais passagens e existe uma redistribuição entre os blocos, isso causa uma passada a mais para cada intercalação.

Intercalação por meio de Substituição por Seleção

A partir do conceito de heap, podemos alterar a primeira fase de intercalação a deixando mais eficiente, agora substituindo o menor item existente na memória interna pelo próximo item da fita de entrada.

De início, monta-se uma fila de prioridades com ‘m’ itens, onde o menor item vai sendo substituído pelo próximo item da fila até o último item. Se o próximo item for menor que o de saída, ele será marcado para que depois ele possa ser tratado como maior do que os subsequentes do próximo bloco.

Como exemplo, vamos usar os mesmos registros do método anterior:

I N T E R C A L A C A O B A L A N C E A D A

Entrada	Fila 1	Fila 2	Fila 3
E	I	N	T

Bloco 1: I

R	N	E*	T

Bloco 1: I N

C	R	E*	T

Bloco 1: I N R

A	R	E*	C*

Bloco 1: I N R T

L	A*	E*	C*

Como o bloco está praticamente todo marcado, criamos um novo

Bloco 2: A

A	C	E	L

Bloco 2: A C

C	E	A*	L

Bloco 2: A C E

A	L	A*	C*

Bloco 2: A C E L

O	A*	A*	C*

Bloco 3: A

B	A	O	C

Bloco 3: A A

A	B	O	C

Bloco 3: A A B

L	C	O	A*

Bloco 3: A A B C

A	L	O	A*

Bloco 3: A A B C L

N	O	A*	A*

Bloco 3: A A B C L O

C	A*	N*	A*

Bloco 4: A

E	A	N	C

Bloco 4: A A

A	C	N	E

Bloco 4: A A C

D	E	N	A*

Bloco 4: A A C E

A	N	D*	A*

Bloco 4: A A C E N

	A*	D*	A*

Bloco 5: A

	A	D	

Bloco 5: A A

	D		

Bloco 5: A A D

_____

Bloco 1: I N R T
Bloco 2: A C E L
Bloco 3: A A B C L O
Bloco 4: A A C E N
Bloco 5: A A D

Para intercalar, monta-se uma fila de prioridade de tamanho ‘f’ a partir dos primeiros itens dos blocos. Depois ir substituindo o item do topo da fila, colocando na fita de saída e substituindo pelo próximo item na fila.

Intercalação Polifásica

Para resolver os problemas de alto número de fitas necessárias para o processo de intercalação e diminuir o alto custo, por ter basicamente uma cópia adicional do arquivo ordenado, usamos a Intercalação Polifásica. No momento da intercalação, para usarmos esse método os blocos já ordenados devem ser distribuídos desigualmente pelas fitas disponíveis, deixando somente uma fita livre. A partir disso, intercalamos os blocos até que uma das fitas de entrada fique vazia. Quando isso ocorre, essa fita torna-se a próxima fita de saída.

A intercalação é feita em várias fases, que não envolvem todos os blocos e nenhuma cópia direta entre fitas é feita.

Para exemplificar, usaremos os mesmos registros de exemplos anteriores:

I N T E R C A L A C A O B A L A N C E A D A

Já com os blocos já ordenados
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 entre os dois primeiros blocos da fita 1 e 2 na 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 entre o primeiro bloco da fita 1 e 3 na 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

Intercalação entre o primeiro bloco das fitas 2 e 3 na fita 1

Ficando:

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

A implementação da intercalação polifásica é simples, porém a parte mais delicada está na distribuição inicial dos blocos ordenados entre as fitas.

Quicksort Externo

Esse método é uma alteração do feito para ordenação interna, utilizando o paradigma da divisão e conquista, ordenando, dentro do próprio arquivo, ‘n’ registros armazenados em acesso randômico, não sendo necessário memória adicional no armazenamento externo, usando somente O(log n) unidades de memória.

A forma de ordenação se dá particionando o arquivo e depois chamando recursivamente o algoritmo em cada um dos subarquivos gerados. o Pivô do algoritmo serão os registros intermediários dos subarquivos, e ficam armazenados na memória interna durante a execução.

Para a partição do arquivo, é utilizada uma área de memória interna para o armazenamento do pivô, sendo o tamanho da área = j − i − 1, sendo necessariamente maior que 3. Durante as chamadas recursivas, a menor partição será a primeira a ser ordenada, partições vazias são ignoradas e se o arquivo de entrada tenha j-i-1 registros, ele é ordenado em apenas um passo.

Durante a partição, temos algumas considerações a serem feitas. Temos os limites inferiores (Linf) e superiores (Lsup) da partição. Também, durante a leitura e a escrita, temos apontadores para as partes inferiores (Li e Ei) e superiores (Ls e Es), que são incrementados ou decrementados a cada leitura/escrita feita na partição.

Os primeiros (TamArea) - 1 registros são lidos, alternativamente, dos extremos da partição e armazenados na área de memória interna, que será uma Estrutura de Dados a ser definida.

O registro (TamArea)ésimo lido, passará por algumas comparações:

Se for maior que o limite Lsup, j recebe Es e o registro é escrito na partição A2;

Se for menor que o limite Linf, i recebe Ei e o registro é escrito na partição A1;

Se estiver entre os dois limites, então é inserido na área de memória interna.

Para garantir que os apontadores de escrita estejam atrás dos apontadores de leitura, a ordem alternada de leitura é interrompida se (Li = Ei) ou (Ls = Es), assim, garantindo que nenhum registro seja perdido durante a ordenação in situ.

Quando a área de memória enche, temos que remover um registro da mesma, considerando os tamanhos atuais das partições A1 e A2. Para isso, definimos algumas condições para que as partições fiquem balanceadas:

Quando as variáveis Esq e Dir forem iguais ao primeiro e ao último registro respectivamente, o tamanho das partições A1 e A2 serão (T1 = Ei – Esq) e (T2 = Dir – Es);

Se (T1 < T2), o registro de menor chave é removido da memória, sendo escrito em Ei (A1), e Linf é atualizado com tal chave;

Se (T2 ≤ T1), o registro de maior chave é removido da memória, sendo escrito em Es (A2), e Lsup é atualizado com tal chave.

Dessa forma, o processo de partição continua até que Li e Ls se cruzem, ou seja, (Ls < Li).

Exemplo de funcionamento do QuickSort Externo: