ESTRUTURA DE DADOS II

Ordenação Externa

Alunos: Samuel Gonçalves
Wagner da Silva Almeida
Professor: Wender Magno Cota

Barbacena – MG
16 de Junho de 2015

Sumário

1. Introdução

O objetivo deste trabalho prático é realizar a implementação de um algoritmo que permita a ordenação externa utilizando o método de intercalação por vários caminhos. O algoritmo consiste em dividir um arquivo ao qual se queira ordenar em blocos na memória principal, ordenar os dados de cada bloco e inserir e arquivos temporários (runs). Depois abrir estes arquivos temporários já ordenados previamente, e utilizando a técnica de intercalação, adicionar uma quantidade em um buffer de entrada e deste buffer para uma fila de prioridades, está ira sempre ter o menor elemento removido e este será encaminhado para um buffer de saída. Quando o buffer de saída estiver cheio, este será gravado em disco. Estas fases se repetem até que todo o arquivo esteja ordenado pelo critério escolhido. A fim de testar o desempenho, foram implementadas duas filas de prioridade, uma utilizando heap e outra utilizando lista encadeada ordenada. Além disso, foi utilizado um uma fila, com tamanho definido que funciona como buffer de entrada e saída.

2. Implementação

Estruturas auxiliares e métodos de ordenação utilizados

Como já citado na parte introdutória do presente documento, foi necessário a utilização de três estruturas auxiliares para a elaboração deste. Foram utilizadas uma fila de prioridades com implementada através de heap, uma fila de prioridades com implementação por lista encadeada ordenada e uma fila encadeada para ser utilizada como buffer de entrada e saída, está com a peculiaridade de possuir um valor máximo de elementos definidos pelo usuário. Estas funções não serão detalhadas neste documento porque não fazem parte do escopo deste trabalho.

Arquivos

registro.h

O arquivo registro.h possui a definição da estrutura que armazena os dados utilizada do trabalho e da estrutura que armazena estes dados e as informações referentes ao arquivo de onde o elemento foi retirado para ser inserido nas filas de prioridades.

OrdenacaoExterna.h

Possui as definições das funções utilizadas para implementação do algoritmo de ordenação externa.

OrdenacaoExterna.c

Possui a implementações das funções definidas no arquivo de cabeçalho OrdenacaoExterna.h. É onde a maioria das funcionalidades são implementadas. As funções serão descritas a seguir.

Funções

void salvaArquivo(char *nome, registro *V, int tam)

A função salvarArquivo() recebe como parâmetro o nome do arquivo, um array com os elementos a serem salvos e o tamanho deste. Salva o arquivo com o nome recebido por parâmetro em disco. Foi criada para auxiliar na criação dos arquivos temporários criados para a etapa de merge.

void salvaArquivoOrdenado(char *nome, Buffer b)

A função salvaArquivoOrdenado() recebe como parâmetro os nome do arquivo e o buffer de saída que contém os arquivos. Ela salva o conteúdo do buffer no fim do arquivo aberto garantindo que nenhum dado já salvo, (já ordenados) seja perdido na operação.

int criaArquivosOrdenados(char *nome, int tamanhoBloco){

A função criarArquivosOrdenados() recebe como parâmetros o nome do arquivo a ser ordenado e o tamanho do bloco. Abre o arquivo para leitura e vai lendo cada registro, quando o total de registros que o bloco estiver no máximo, utiliza a função qsort para ordenar os arquivos no bloco (array) e salva o arquivo temporário utilizando a função salvaArquivo(). Realiza esta operação até que todo o arquivo seja lido, dividido em runs pré-ordenadas e gravadas em arquivos temporários. A função retorna a quantidade de arquivos temporários gerados.

int compara(const void* a, const void* b)

A função compara recebe dois ponteiros genéricos como parâmetros. Ela serve como critério de comparação para a função qsort da linguagem c. Esta implementação é necessária para que a função possa ordenar o array de forma correta.

void preencheBuffer(struct arquivo* arq, int tamBuffer)

A função preencheBuffer recebe como parâmetros a struct arquivo e o tamanho do buffer que foi definido pelo usuário na inicialização do programa. Ela abre o arquivo temporário, e carrega

os registros para o buffer até que este esteja completo, e incrementa a variável máximo para sinalizar cada inserção no buffer. Caso o arquivo esteja vazio, fecha-o.

```
int procuraMenor(struct arquivo* arq, int numArqs, void* fila, int numFila,int tamBufferEntrada, tipoInfo *menor)
```

A função procuraMenor() recebe como parâmetros o arquivo temporário, o número de arquivos gerados, um ponteiro genérico, o tipo de fila de prioridade utilizada, o tamanho do buffer de entrada e um ponteiro para retornar o menor valor achado. A função seleciona o menor valor na fila de prioridade e atribui ao ponteiro passado por parâmetro o valor lido. Verifica se o buffer de entrada de onde o arquivo retirado está vazio, caso não esteja retira um novo elemento do buffer e insere na fila de prioridade, caso esteja vazio, preenche o buffer de entrada com mais valores do arquivo de origem, caso este não esteja vazio, e insere um valor na fila de prioridade. Retorna 1 caso a operação seja bem sucedida, e 0 caso não seja (caso não houver mais elementos na fila e nos arquivos).

```
void merge(char *nome, int numArqs, int tamBufferEntrada, int tamBufferSaida, void* fila,int numFila)
```

A função merge() é a função responsável por criar controlar a ordenação externa após a criação dos arquivos temporários. Ela recebe o nome do arquivo a ser criado, o número de arquivos temporários gerados, tamanho do buffer de entrada e de saída, um ponteiro genérico e o tipo de fila de prioridade que será utilizada. A função cria o buffer de saída utilizado e aloca espaço para estruturas arquivo criadas para manipulação dos arquivos temporários. Cada estrutura possui um buffer de entrada, um ponteiro para arquivo, um campo sinalizando a posição e um indicando a quantidade de elementos. Cada estrutura destas, guardará as informações de um arquivo temporário gerado. Preenche o buffer de entrada de cada arquivo e insere um arquivo de cada buffer de entrada na fila de prioridade. Depois do preenchimento inicial a função realiza chamadas sucessivas a função auxiliar procuraMenor() e vai inserindo os elementos enquanto houverem no buffer de saída, quando este estiver cheio, grava o conteúdo no arquivo. Quando a fila de prioridades estiver vazia, a função procuraMenor() retornará 0, sinalizando que não existem mais elementos. Se houverem elementos restantes no buffer, estes serão salvos no arquivo e as variáveis que foram utilizadas através de alocação de memória tem seu espaço liberado.

```
void mergeSortExterno(char *nome, int tamanhoBloco, int tamBufferEntrada, int tamBufferSaida,int numFila)
```

A função mergeSortExterno() recebe os parâmetros necessários para a execução do programa e controla o fluxo de ações da ordenação. Os parâmetros recebidos são o nome do arquivo, tamanho do bloco, tamanho dos buffers de entrada e saída e o tipo da fila q será utilizada.

3. Testes

Desempenho na execução

Foram executados alguns testes afim de verificar a velocidade de processamento dos dados na ordenação. Para os testes foi utilizado atráves da função fwrite a ordenação de 1.000.000 de registros com 512 bytes cada. Foram observados os seguintes valores:

Fila de prioridades utilizando heap

Tamanho Registro	Quantidade de Registros	Tamanho Bloco	Tamanho arquivo gerado	Tempo médio
512 bytes	1 milhão	10000	812 Mb	2931,2s ou 48min
4 bytes	1 milhão	10000	6,35 Mb	22.9s
512 bytes	1 milhão	5000	812 Mb	7833,6s ou 2h10min
4 bytes	1 milhão	5000	6,35 Mb	61,2s

Fila de prioridades utilizando lista encadeada

Tamanho Registro	Quantidade de Registros	Tamanho Bloco	Tamanho arquivo gerado	Tempo médio
512 bytes	1 milhão	10000	812 Mb	975,36s ou 16,05min
4 bytes	1 milhão	10000	6,35 Mb	7,62s
512 bytes	1 milhão	5000	812 Mb	591,36s ou 10,25min
4 bytes	1 milhão	5000	6,35 Mb	4,62s

Com base nos tempos apresentados podem ser constatadas as seguintes conclusões:

- Das duas formas de fila de prioridades utilizadas, a que apresenta melhores desempenhos é a estratégia que utiliza lista encadeada.

- Quanto maior for o tamanho do bloco, mais rápida será a ordenação utilizando fila de prioridade por heap.
- O tamanho do bloco impacta diretamente sobre a quantidade de arquivos temporários gerados, pois quanto maior o bloco, menos arquivos serão gerados.
- Quanto maior o bloco, mais rápido se torna o funcionamento da fila de prioridade por lista encadeada e a ordenação por este método. Isso demonstra que neste caso, o seu funcionamento é inversamente proporcional ao funcionamento da implementação por heap.

4. Conclusão

Em resumo o trabalho prático proporcionou uma visão da utilização de algoritmos de ordenação externa. Conclui-se que o acesso a disco tem um custo altamente impactante no desempenho de qualquer sistema, e que este deve ser gerenciado de forma que se evitem acessos recorrentes ao disco. Uma forma de contornar está situação, é carregando dados frequentemente utilizados para a memória principal, o que nem sempre é possível pois deve ser levar em consideração diferentes tipos de maquinas e sistemas, para que o programa seja o mais portável possível. A parte mais complexa no desenvolvimento deste trabalho prático, foi o controle dos arquivos para ordenação. Na solução desenvolvida, algumas vezes o sistema gerava runs de forma infinita, o que ocasionava no travamento do sistema por esgotamento de memória, ou deixava o sistema extremamente lento. No geral, a compreensão dos métodos de ordenação externa, em especial o utilizado neste trabalho, é de importância essencial para o desenvolvimento de sistemas, e deve ser feito da melhor forma possível, pois persistência dos dados é algo essencial nos dias atuais, e a utilização das técnicas aprendidas neste trabalho, possibilitará o desenvolvimento de sistemas melhores e estáveis para serem entregues como produto final.

5. Referências

TERADA, R. (1991) **Desenvolvimento de Algoritmos e Estruturas de Dados**. McGraw-Hill e Makron Books do Brasil, São Paulo, SP.

(Backes, 2013)
BACKES, André. Estrutura de Dados Descomplicado – Ordenação Externa. Disponível em: <https://programacaodescomplicada.wordpress.com/2014/09/23/ed1-aula-66-ordenacao-externa/ > Acesso: 01/06/2015.

ZIVIANI, Nívio. (2004) Projeto de algoritmos, Cap. 4.

ASSIS, Guilherme Tavares de. Ordenação Externa, DECOM, – Departamento de Computação - UFOP. <Disponível em: http://www.decom.ufop.br/guilherme/BCC203/geral/ed2_ordenacao-externa.pdf > Acesso: 04/06/2015.