Análise sobre Algoritmos de Ordenação Externa

Rafael Campos Nunes
Mikael Messias

12 de junho de 2018

#+LANGUAGE: pt-br

Part I

Introdução

Atualmente algoritmos são utilizados em diversas atividades corriqueiras, mesmo que não conscientemente observadas são realizadas atividades como a rota para um local específico, buscar algum número de telefone no *smartphone* ou a busca por um amigo em alguma rede social. Todos esses algoritmos utilizam-se de algum método de ordenação e/ou busca, importante ressaltar ainda que estes ordenam sobre dados que por vezes não estão presentes em memória primária do dispositivo e sim localizados em memória de armazenamento em massa.

O presente trabalho realiza a análise dos algoritmos de ordenação em memória secundária, denominados na literatura de: algoritmos de ordenação externa. Esses algoritmos operam sobre dados não presentes na memória primária, isto porque na maioria das vezes os dados sobre quais esses algoritmos operam são muito grandes para a capacidade da memória primária e algumas estratégias são utilizadas para essa categoria de ordenação. Apesar dos referidos algoritmos trabalharem em memória secundária os dados são ordenados em memória primária, contudo, em menor quantidade.

Part II

Análise do Merge Sort

Chapter 1

Implementação do algoritmo

O *Merge Sort* externo é um algoritmo que rearranja os registros de um arquivo da memória secundária do computador utilizando uma estratégia de ordenação interna. Faz a leitura dos registros no arquivo especificado a ser ordenado e possui como principais etapas a classificação e a intercalação. Ao fim da execução, o algoritmo gera um arquivo de saída com os registros ordenados na memória secundária do computador.

1.1 Etapa de classificação

A ordenação externa torna-se necessária quando se tem um arquivo cujo número de registros é maior do que a memória primária pode armazenar. Esses registros precisam ser classificados em blocos menores, de forma que possam ser rearranjados utilizando alguma estratégia de ordenação interna.

Na implementação do algoritmo, para este trabalho, a memória primária é representada como um vetor de números inteiros, com tamanho máximo igual a 100. Portanto, independente da quantidade registros do arquivo original, o *Merge Sort* externo só será capaz de criar blocos de ordenação com 100 elementos.

Após a cópia dos registros para a memória interna, um algoritmo de ordenação interna é aplicado ao vetor que, logo em seguida, tem seus valores salvos em um arquivo temporário, tal arquivo é denominado como um bloco ordenado. O algoritmo *Heap Sort* foi utilizado para ordenação dos blocos na memória primária.

1.2 Etapa de intercalação

Após classificar os registros do arquivo original em blocos ordenados, é necessário combiná-los em um único bloco ordenado. Esse etapa é chamada intercalação.

No algoritmo, há uma estrutura cuja função é auxiliar a intercalação armazenando algumas informações relevantes durante o processo.

```
1 struct file {
2     FILE *f;
3     int pos;
4     int MAX;
5     int *buffer;
6 };
```

O primeiro passo é criar um vetor da estrutura `File` com tamanho igual ao número de blocos gerados pela etapa de classificação. O campo buffer, que referencia um valor ou conjunto de valores inteiros, será utilizado para armazenar os registros do arquivo referenciado por `f`.

1. Selecione o menor registro na posição atual de cada bloco.
2. Armazene o valor dentro do vetor na memória primária.
3. Quando o vetor atingir seu tamanho máximo, salve os registros dentro do arquivo de saída,
4. Repita os passos até que não haja mais valores para serem intercalados.

Chapter 2

Manipulação de arquivos

O *Merge Sort* externo requer leitura e escrita em arquivos na memória secundária do computador em alguns momentos da execução. O algoritmo implementado possui funções específicas para manipulação de arquivos, que permite realizar essas operações.

A função de inserção anexa ao fim de um arquivo os valores de um vetor de números inteiros. É utilizada tanto na classificação, quando são gerados blocos menores que serão posteriormente ordenados e escritos em arquivos temporários, quanto na intercalação, quando os blocos serão combinados novamente, gerando o arquivo de saída.

Durante a etapa de intercalação, como os blocos ordenados estão armazenados em arquivos na memória secundária do computador, é necessário utilizar vetores para recuperar os registros de cada bloco afim de intercalá-los e escrevê-los no arquivo de saída. A função de recuperação, a partir de um arquivo localizada na memória secundária do computador, preenche um vetor de números inteiros com os registros desse arquivo.

Chapter 3

Análise do algoritmo

Part III

Merge Sort utilizando Fila de Prioridades

A fila de prioridades é uma estrutura de dados em que os elementos desta obedecem uma propriedade, sendo esta a característica que confere à fila de prioridades, conhecida como heap, a denominação de *heap de máximo* ou *heap de mínimo*. A propriedade enunciada é similar nos dois casos de *heap* com a troca somente da condição da propriedade.

1. Todo *i-ésimo* elemento é maior que o elemento *2*i-ésimo+1*
2. Este *i-ésimo* elemento é também maior que o elemento *2*i-ésimo+2*

Tal estrutura é utilizada para na etapa de classificação e intercalação de elementos. Sua utilização traz alguns benefícios pois não necessariamente os blocos de arquivo serão divididos igualmente, por vezes sendo utilizados menos arquivos.

A implementação da fila de prioridades se concretizou por meio de duas estruturas, elas são mostradas no código abaixo.

```
1  struct heap_element {
2      int key;
3      int weight;
4  };
5
6  struct priority_queue {
7      // The vector that holds the heap inside the structure
8      struct heap_element *vector;
9
10     // The current index of the heap
11     uint16_t index;
12     // The current size of the heap
13     uint16_t size;
14 };
15
16 typedef struct priority_queue Heap;
17 typedef struct heap_element HeapElement;
```

Código 1: Representação da fila de prioridades em C

A primeira estrutura representa um elemento da fila de prioridades e a segunda representa a fila em si. As diversas operações realizadas sobre essa estrutura estão todas descritas no arquivo *heap.c*. A saber, as operações realizam as funções de inserção, inserção em um intervalo especificado, remoção do primeiro elemento e a ordenação dos elementos desta fila.

Tais operações são fundamentais nas etapas de classificação e intercalação de dados dado que a ordenação de seleção por meio de substituição utiliza a referida estrutura. Este algoritmo utiliza os seguintes passos para fazer a leitura e ordenação dos arquivos nas diversas fitas de dados:

1. Lê os números de um arquivo, preenchendo o espaço permitido na memória primária
2. Retira-se o primeiro elemento da fila, aqui chamado de v_0 para a leitura do próximo
3. Compara-se o v_0 com o elemento a ser inserido, se este for menor aumenta-se o peso
4. Insere-se o último número lido na primeira posição do *heap*
5. Escreve-se o elemento v_0 na fita correspondente ao seu peso

Chapter 4

Operações na Fila de Prioridades

Como enunciado anteriormente as operações na fila são quatro, nesta seção elas são trabalhadas com mais detalhes. A primeira delas é a inserção em um intervalo específico da fila, a configuração do algoritmo se dá como a seguir:

```
1  void sift_up_i(Heap *heap_t, int key, int weight, int index) {
2
3      if (heap_t == NULL) {
4          return;
5      }
6
7      HeapElement he;
8
9      he.key = key;
10     he.weight = weight;
11
12     heap_t->vector[index] = he;
13
14     heap_sort(heap_t);
15 }
```

Código 2: Inserção de elementos com pesos na fila de prioridades

O código faz uma verificação de segurança e após isso insere o valor criado na posição especificada pelo chamador da função. A operação posterior utiliza-se desta função para a inserção de novos elementos, dessa forma aumentando a coesão do código e reaproveitamento deste.

```
void sift_up(Heap *heap_t, int key, int weight)
    if (heap_t == NULL) {
        return;
    }

    sift_up_i(heap_t, key, weight, heap_t->index);

    heap_t->index = heap_t->index+1;
    heap_t->size = heap_t->index;
}
```

Código 3: Inserção de elementos com pesos na fila de prioridades

A remoção de um elemento na fila é realizada de maneira simples, só é necessário recuperar o primeiro dado dessa fila e retorná-lo. Observa-se, entretanto, que é necessário, logo após a remoção deste elemento a inserção de outro no mesmo lugar. A API da fila foi projetada dessa maneira para melhor atender às necessidades algoritmicas dos projetistas.

```
HeapElement sift_down(Heap* heap_t) {

    HeapElement e = heap_t->vector[0];

    heap_sort(heap_t);

    return e;
}
```

Código 4: Remoção de elementos na fila de prioridades

Chapter 5

Ordenação da Fila de Prioridades

A fila de prioridades, além de ter todas as operações já denotadas, contém um método de ordenação diferente pois necessita ordenar os elementos a partir de seus pesos e quando não se delinea diferença entre estes é utilizado o valor das chaves. O algoritmo criado para tal fim é mostrado abaixo:

```
void heapify(HeapElement *heap_elements, int n, int i) {

    int smallest = i;

    int l = 2*i+1;
    int r = 2*i+2;

    int smallest_weight = heap_elements[smallest].weight;

    if (l < n) {
        int l_weight = heap_elements[l].weight;

        if (smallest_weight < l_weight) {
            smallest = l;
        } else if (smallest_weight == l_weight) {
            if (heap_elements[smallest].key < heap_elements[l].key) {
                smallest = l;
            }
        }
    }

    if (r < n) {
        int r_weight = heap_elements[r].weight;

        if (smallest_weight < r_weight) {
            smallest = r;
        } else if (smallest_weight == r_weight) {
            if (heap_elements[smallest].key < heap_elements[r].key) {
                smallest = r;
            }
        }
    }

    if (i != smallest) {
        swap(&heap_elements[i], &heap_elements[smallest]);
        heapify(heap_elements, n, smallest);
    }
}
```

Código 5: Método de ordenação (construção do heap)

Ressalta-se, contudo, que este é uma das partes do método de ordenação pois ela é composta de outra parte tão igualmente importante a supracitada pois de outra forma não seria realizada ordenação nenhuma. Portanto, com essas operações elementares inicia-se o desenvolvimento do algoritmo para ordenação externa que se utiliza do algoritmo descrito na seção anterior para executar o método de ordenação externa de substituição por meio de seleção.

Part IV

Referências

CORMEN, Thomas et al. **Algoritmos**. 3. ed. Elsevier, 2012.
ZIVIANI, Nivio. **Projeto de algoritmos com implementações Pascal e C**. 4. ed. Sâo Paulo: Pioneira, 1999.