ordenação externa

Diego Dariva e Thiago Pavin

ddeltz@inf.ufsm.br e tbpavin@inf.ufsm.br

1. Introdução

A ordenação de dados, na ciência da computação, é um assunto muito importante e bastante discutido, que tem o objetivo de colocar os elementos de uma dada sequência em uma certa ordem. Atualmente a necessidade de algoritmos de ordenação externo mais eficientes tem aumentado, pois a quantidade de dados está ficando cada vez maior, sendo impossível de usar a ordenação interna em todos os casos. Neste trabalho será apresentado um algoritmo de ordenação externa, para organizar os dados em ordem alfabética.

2. Algoritmo

A ordenação externa consiste em ordenar arquivos de tamanho maior que a memória interna (principal) disponível. Nos algoritmos de ordenação externa deve-se reduzir o número de acessos ao disco, responsável por grande parte do custo do algoritmo. O método de ordenação externa apresentado neste trabalho utiliza as seguintes estratégias:

1. Quebrar o arquivo em blocos do tamanho da memória interna disponível.

2. Ordene cada bloco na memória interna (Quicksort).

3. Intercale os blocos ordenados, fazendo várias passadas sobre o arquivo.

4. Com cada passada é adicionado a um bloco maior os dados, até que todo o arquivo esteja ordenado.

Na ordenação de cada bloco do arquivo (segunda estratégia) é utilizado o método de ordenação interna Quicksort, que é um método de ordenação muito rápido e eficiente.

2.1 Função 'Reduz'

Função 'Reduz' pega uma palavra (String) passada por parâmetro e transforma todos os seus caracteres em caracteres minúsculos.

```
char* reduz(char p[PAL]){
    int n=strlen(p);
    for (int i = 0; i < n; i++){
        if (p[i]<'a'){
            p[i]+=('a'-'A');
        }
    }
    return p;
}
```

2.2 Função ‘QuickSort’ para vetor de strings

Função ‘Quick Sort’ ordena um vetor de palavras (Strings) em ordem alfabética, nesta função é escolhido um pivô (palavra no meio do vetor), com o pivô escolhido todos os dados menores vão para uma posição anterior ao pivô, e os dados maiores vão para uma posição posterior ao pivô, portanto no final deste processo o pivô estará em sua posição final e com duas sublistas de dados não ordenados e cada lado. Então em cada sublista é utilizada recursão para ordenar seus dados.

O processo é finito, pois a cada iteração pelo menos um elemento é posto em sua posição final e não será mais manipulado na iteração seguinte.

```
void quickSort(char **strings, int esq, int dir){

    int i = esq, j = dir;
    char *x;
    char temp[50];


     //pega a string do meio como o pivo
    if(j - i >= 1){
        x = strings[i];
        strcpy(temp, strings[esq]);
        strcpy(strings[esq], x);
        strcpy(x, temp);
        while(j > i){
            //  palavra[i] é menor que o pivô e i < direita
            while((strcmp(reduz(strings[i]),x) <= 0) && i < dir && j > i){
                i++;
            }
            //  palavra[j] é maior que o pivô e j > esquerda
            while((strcmp(reduz(strings[j]),x) >= 0) && j > esq && j >= i){
                j--;
            }
            // troca as palavras
            if(j > i){
                strcpy(temp, strings[i]);
                strcpy(strings[i], strings[j]);
                strcpy(strings[j], temp);
            }
        }

        strcpy(temp, strings[esq]);
        strcpy(strings[esq], strings[j]);
        strcpy(strings[j], temp);

        quickSort(strings, esq, j-1);

        quickSort(strings, j+1, dir);
    }
}
```

2.3 Função ‘NumPalavras’

A função ‘NumPalavras’ conta a quantidade de palavras existentes no arquivo passado por referência.

```
int NumPalavras(char *NomeArq){
    int comecouPalavra = 0, numPalavras = 0, numLinhas = 0;
    FILE *descritor;
    char *caracter;

    descritor = fopen(NomeArq, "r");
    while (!feof(descritor)) {
          fread(caracter, 1, 1, descritor);
          if  ((*caracter!=' ') && (*caracter!='\n') && (!comecouPalavra)) {
             comecouPalavra = 1;
          }
          if  (((*caracter==' ') || (*caracter == '\n')) && (comecouPalavra)) {
             comecouPalavra = 0;
             numPalavras++;
          }
          if (*caracter=='\n') {
             numLinhas++;
             }
   }
   return numPalavras + 1;
}
```

2.4 Função ‘LimpaArquivo’

A função ‘LimpaArquivo’ retira todos os caracteres especiais do arquivo.

```
int LimpaArquivo(char Arq[]){

    FILE *entrada, *saida;
    entrada=fopen(Arq,"r+");
    saida=fopen("limpo.txt", "w+");
    char c;
    while(1){
        c=fgetc(entrada);
        if (feof(entrada)){
            break;
        }
        if (c==' ' || (c>='0' && c<='9') || (c>='A' && c<='Z') || (c>='a' && c<='z') || c =='\n'){
            if(c == '\n'){
                c = ' ';
            }
            fprintf(saida, "%c", c);
        }
    }
    fclose(entrada);
    fclose(saida);

    return 0;
}
```

2.5 Função ‘separaArq’

A função ‘separaArq’ separa o arquivo original em quatro blocos com uma quantidade menor de dados para serem ordenados.

```
void separaArq(void){
    FILE * inicio=fopen("limpo.txt","r");
    FILE *parte;
    int n=NumPalavras("limpo.txt");
    int narq=0;
    int i=0;

    char name[15];
    char palavra[50];
    printf("%d\n", n);
    n=(n/4)+1;
    printf("%d\n", n);
    while(!feof(inicio)){
        geraNomeArq(narq,name);

        parte=fopen(name, "w+");
        while(i<n && !feof(inicio)){
            fscanf(inicio, "%s", palavra);
            fprintf(parte, "%s ", palavra);
            i++;
        }

        i=0;
        narq++;
        fclose(parte);
    }
    fclose(inicio);
    while(narq<4)
    {
        geraNomeArq(narq,name);
        parte=fopen(name, "w+");
        fclose(parte);
        narq++;
    }
}
```

2.6 Função ‘geraNomeArq’

A função ‘geraNomeArq’ cria os nomes de cada bloco (arquivo) novo criado.

```
void geraNomeArq(int n, char *retorno){
    char num[2];
    char part[5];
    char txt[5];
    char name[15];

    strcpy(part,"part");
    strcpy(txt,".txt");

    strcpy(name,part);
    num[0]=n+'0';
    num[1]='\0';
    strcat(name,num);
    strcat(name,txt);

    strcpy(retorno, name);
}
```

2.7 Função ‘ordenaBlocos’

A função ordena cada bloco menor do arquivo inicial separadamente com a função QuickSort.

2.8 Função ‘uneArq’

A função unifica todos os blocos menores em apenas um grande arquivo ordenado e apaga os arquivos temporários que continham os blocos.

2.9 Main

```
int main(){
    //cria uma string com o nome do arquivo
    char arquivo[25] = "aristotle-on-267.txt";

    //retira todos os caracteres especiais
    LimpaArquivo(arquivo);

    //separa os arquivo em blocos menores
    separaArq();

    //ordena cada bloco menor
    ordenaBlocos();

    //unifica os blocos menores em um maior ordenado
    uneArq();

    getchar();
    return 0;
}
```

3. Aplicações

A ordenação é muito importante para a computação, pois tem o objetivo de tornar mais simples, rápida e viável a localização ou a recuperação de uma determinada informação em um conjunto grande de informações, que é requisito comum em aplicações que precisam apresentar informações ordenadas como por exemplo:

1. Aplicações para apresentar uma listagem dos empregados por ordem alfabética.
2. Organizar nomes em ordem alfabética numa lista telefônica.
3. Aplicações de música, para o fácil acesso as músicas buscadas.
4. Dado um conjunto de um milhão de registros de dados, remova ou mescle as duplicatas.
5. Compare dois grandes conjuntos de itens e descubra onde eles se diferem.

Existem várias razões para se ordenar uma sequência. A principal delas é a possibilidade se acessar seus dados de modo mais eficiente.

Referências

https://pt.wikipedia.org/wiki/Quicksort

https://www.trabalhosgratuitos.com/Sociais-Aplicadas/Servi%C3%A7o-Social/Metodo-De-Ordena%C3%A7%C3%A3o-386783.html

https://en.wikipedia.org/wiki/External_sorting

João Vicente Ferreira Lima "Notas de aula - Pesquisa e Ordenação de Dados A"

https://www.quora.com/In-computer-programming-why-is-sorting-important-When-are-the-sorting-algorithms-used-in-real-coding