UNIVERSIDADE FEDERAL DE MINAS GERAIS
ALGORITMOS E ESTRUTURAS DE DADOS III

TRABALHO PRÁTICO
04

ORDENAÇÃO EXTERNA

Aluna: Thalia de Almeida Magalhães Campolina
Matrícula: 2010054827
e-mail: thaliacampolina@gmail.com

1.INTRODUÇÃO

Esse trabalho visa a ordenação, em ordem alfabética, de contatos de pessoas (número e seu telefone) de um celular.
Uma memória restrita é utilizada para tal, portanto foi utilizada a ordenação externa. É definido um tamanho máximo de um buffer, que é a memória utilizada pra realizar a ordenação. Como forma de auxilio para essa ordenação são utilizados arquivos que recebem os nomes conforme esses são ordenados, mas a ordenação só pode ser realizada em etapas, ordenando o número de contatos conforme o tamanho dessa memória.

Foi utilizada a linguagem C e o compilador gcc. Compilado em máquina Linux.

2.SOLUÇÃO PROPOSTA

O arquivo que é utilizado como entrada do programa entrada informa, além dos contatos, o tamanho do buffer utilizado para ordenação, e o número de contatos a serem ordenados. O número de arquivos, que são blocos auxiliares para ordenar os nomes, é definido pela divisão dos números de contatos pelo tamanho do buffer.

Após ler as entradas dos nomes, é carregado no buffer os contatos que cabem no mesmo, e enviados para um arquivo output_i.txt. Conforme o buffer enche, é criado um novo arquivo, portanto em cada um desses arquivos possui no máximo o número de contatos que o buffer pode carregar. Depois dessa etapa, foi utilizado uma estrutura *heap* em forma de vetor, e os contatos de cada arquivo foi ordenado. O heap consiste em uma estrutura que ordena seu primeiro elemento(a posição *zero* do vetor) como o menor de todos.

2.1. Estrutura utilizada

Foi criado uma estrutura contact, presente no arquivo contact.h e contact.c.
Essa estrutura de contatos possui, dentro dela, apenas um char* name_, que guarda o nome do contato seguido de seu número.

3.ALGORITMOS

No arquivo contact.c estão presentes as funções:

IsMinorThen, que testa se uma string (a primeira passada como parâmetro) é menor do que a outra string(a segunda passada como parâmetro). São testadas duas strings apenas.

HeapSort, que constroi uma estrutura heap depois que um vetor é criado. Esse vetor,

passado como parâmetro, é um vetor vazio que será ordenado para ficar organizado como um heap. O segundo parâmetro é o tamanho desse vetor, que corresponde ao tamanho do buffer que irá organizar os contatos em ordem alfabética.

SortNames, que utiliza o método de ordenação *selection* sort(ordenação por seleção). Essa função será utilizada para que os nomes de cada arquivo gerado para a divisão em blocos da entrada, seja ordenado.

SortNameInFile, função que utiliza a de cima (SortNames) para ordenar os nomes do arquivo output_i.txt passado como parâmetro de entrada para a função. Para essa etapa é criado um vetor com os nomes dentro de cada um dos arquivos, portanto o tamanho do vetor é o tamanho do buffer. Após os nomes serem carregados na memória, eles são ordenados chamando a função SortNames e colocados de volta no mesmo arquivo.

TreatFiles, função que ordena os arquivos intermediários criados e ordenados nas funções acima. Nessa função, cada um dos arquivos gerados inserido em um vetor de arquivos e aberto. Então, de cada arquivo são lidos os primeiros elementos até que o heap esteja cheio. Quando estiver cheio, é aplicado o heapsort e o menor elemento é colocado no vetor de saída; então o próximo elemento do próximo arquivo é inserido no heap, que é novamente ordenado. Isso acontece repetidamente até que todos os nomes tenham sido lidos e ordenados.

3.1 Considerações

É necessário ressaltar que o programa está funcionando corretamente até a ordenação dos nomes de cada um dos arquivos de saída intermediários, ou seja, até a função SortNameInFile. A função TreatFiles não está funcionando corretamente, os nomes ordenados não estão sendo inseridos em totalidade no arquivo de saída geral, nomeado output.txt.

Também é necessário considerar que não está sendo tratado o caso em que o número de arquivos intermediários output_i.txt gerados é em maior do que o tamanho do buffer, portanto o que deveria ser feito é uma leitura e intercalação também em blocos, portanto seriam carregados apenas o número de arquivos igual ao tamanho do buffer e intecalados em outro arquivo intermediário output_i.txt, e isso seria feito repetidamente até o fim da leitura dos arquivos. Portanto, outros arquivos intermediários seriam gerados e cada vez com um maior número de contatos, e novamente esses arquivos maiores seriam intercalados.

4.COMPILAÇÃO

A compilação deve ser utilizada com o comando make, relativo ao Makefile. Então o comando a ser executado deverá ser:

./tp4 -i input.txt -o output.txt

4.1 Entrada

O arquivo de entrada deverá chamar input.txt e conter, na primeira linha, um inteiro representando o número de contatos, a segunda linha representando o tamanho do buffer, e as linhas subsequentes deverá conter os contatos em si.

4.2 Saída

A saída consiste em um arquivo output.txt e deverá conter apenas os contatos de telefone ordenados em ordem alfabética.

5.COMPLEXIDADE

5.1 Complexidade em espaço

A estrutura utilizada para o programa é um vetor, portanto tem complexidade de espaço n, relativa ao tamanho desse vetor.

5.2 Complexidade do algoritmo

A complexidade do algoritmo é O(n log n) a partir do momento que utiliza o heapsort para ordenar os contatos.

Porém, quando utiliza o Selection Sort sua complexidade torna-se quadrática $O(n^2)$, já que o selection sort utiliza um laço interno ao outro.