UFMG UNIVERSIDADE FEDERAL DE MINAS GERAIS

Arthur Siríaco Martins de Lacerda

TRABALHO PRÁTICO 2 - Estrutura de Dados
Documentação

Belo Horizonte
2022

Arthur Siríaco Martins de Lacerda

TRABALHO PRÁTICO 2
Estrutura de Dados

Este documento contém a documentação referente ao Segundo Trabalho Prático da disciplina Estrutura de Dados, do curso Sistemas de Informação - Universidade Federal de Minas Gerais.

Professores: Gisele Lobo Pappa/Wagner Meira Jr.

Belo Horizonte
2022

INTRODUÇÃO

Este trabalho tem como objetivo intercalar endereços de sites na web, comparando-os pelo número de acessos. Para tanto, chamaremos este conjunto contendo URL e número de acessos de Entidades e os agrupamos em rodadas. A motivação deste programa é simular uma tarefa corriqueira em plataformas de Web Analytics, que é avaliar os sites mais populares.

MÉTODO

Para a confecção deste trabalho, utilizei técnicas de programação em conjunto com algumas estruturas de dados intrínsecas à esta disciplina.
Conforme solicitado, para realizar uma ordenação inicial das rodadas, foi usado o algoritmo de ordenação QuickSort, que, nesse caso, é responsável por particionar um vetor de entidades e ordená-lo recursivamente em ordem crescente. A motivação para essa escolha será discutida a seguir. Dessa forma, cada rodada gerada já é escrita em seu respectivo arquivo “Rodada-n.txt” de maneira ordenada. Ademais, nesse caso, o vetor de entidades foi alocado de maneira dinâmica.

Feito isso, foi criada uma classe Pilha Arranjo, semelhante àquela contida no módulo de estrutura de dados desta disciplina. Escolhi a alocação estática, pois a implementação é consideravelmente menos sensível e o limite MAXTAM pode ser editado diretamente no código. Cada pilha deverá guardar as entidades de determinada rodada. Para que isso ocorra, implementei um vetor de “Pilhas Arranjo” cuja função é de armazenar as entidades lidas nas rodadas e indexá-las, de acordo com a rodada, visando facilitar a operação do heap.

Uma vez lidas as rodadas e devidamente armazenadas nas pilhas indexadas, construímos o primeiro heap. A classe Heap, além de construir e refazer um Heap, conforme ensinado nas aulas, possui a propriedade de intercalar, que é a parte mais importante dessa fase do algoritmo. Assim, construímos um Heap inicial coletando o maior valor de cada pilha. Isso explica a ordenação crescente, pois, ao desempilhar, certamente o maior valor da rodada será retornado. Uma vez feito o Heap, a maior entidade é escrita no arquivo de saída e inicia-se o processo de intercalação, até que o Heap se esvazie.

Para auxiliar na leitura e escrita de arquivos, utilizei, da biblioteca fstream, os tipos ifstream e ofstream.

ANÁLISE DE COMPLEXIDADE

Nessa fase, serão listados todos os procedimentos acompanhados de sua complexidade utilizando da notação O-Grande.

- A complexidade do QuickSort é O(nlogn) no melhor caso e no caso médio e $O(n^2)$ no pior caso.
- As funções LeRodada e EscreveRodada tem o pior caso como O(n)
- A função GeraRodada, como faz chamada para as três funções supracitadas, é dominada assintoticamente por todos os casos do algoritmo QuickSort.
- A função LeFitas tem complexidade $O(n^2)$, pois apresenta dois loops While aninhados, que dependem do número de fitas e do número de entidades, respectivamente.
- Todos os métodos da classe PilhaArranjo tem complexidade constante.
- O método HeapInicial da classe Heap tem complexidade O(n), pois é composto apenas de um loop For, que depende da variável size.
- Os métodos Constrói e Refaz da classe Heap, conforme passado em aula, tem complexidade O(nlogn) e O(logn), respectivamente.
- O método Intercala chama o método Constrói um número n de vezes, fazendo com que ele seja $O(n^2*logn)$
- O método HeapVazio possui complexidade constante.

CONCLUSÃO

Neste trabalho foi simulada a tarefa de analisar sites mais populares, por meio da intercalação de rodadas contendo várias URL's com seus respectivos números de acessos.

Durante a realização deste, foi possível aprender como interligar estruturas de dados distintas com algoritmos e técnicas de programação já conhecidas, demonstrando como são úteis e eficazes os tópicos aprendidos ao longo deste curso.

Foi possível, também, aprender mais sobre o comportamento da memória e do desempenho da máquina durante a compilação e execução do código.

BIBLIOGRAFIA

- Slides e videoaulas disponíveis no Moodle na Metaturma de Estrutura de Dados.
- Documentação de diversas bibliotecas e funções da Linguagem C++, que foram obtidas no site https://www.cplusplus.com/

INSTRUÇÕES PARA COMPILAÇÃO E EXECUÇÃO

COMPILAÇÃO: Para compilar o programa, basta ir ao diretório raiz (TP2), abrir o mesmo no terminal e rodar o comando “make”. Caso seja de interesse do usuário apagar os arquivos .o e executáveis gerados durante a compilação, existe o comando “make clean” que executa esta tarefa.

EXECUÇÃO: Para executar, existem 4 argumentos que devem ser passados obrigatoriamente (nessa ordem).

1. Inputfile: Isto é, o nome do arquivo de entrada, do qual deverão ser lidas as informações para a geração das rodadas ordenadas.
2. Outputfile: O nome do arquivo final, no qual serão escritas as entidades intercaladas e ordenadas.
3. O número de entidades em cada fita
4. O número de fitas a serem lidas

OBS: OS NOMES DOS ARQUIVOS SEMPRE DEVERÃO VIR ENTRE PARÊNTESES E ACOMPANHADOS DAS DEVIDAS EXTENSÕES.

Exemplo de comando para a execução: ./run.out "input.txt" "output.txt" 6 4
(No caso acima, o programa irá ler do arquivo “input.txt” e escrever no “output.txt”, considerando 6 entidades por fita e 4 fitas).