Trabalho Prático 2

Antony Guilherme Costa do Amaral

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

antonyguilherme@ufmg.br

Matrícula: 2020026796

Introdução

Esta documentação tem como objetivo explicitar o funcionamento do programa que realiza a ordenação de sites e seus respectivos números de acesso. Isso é realizado por meio do uso simulado de fitas, afim de limitar de forma proposital o poder de armazenamento do programa. Dito isso, o programa possui duas ações principais: **Intercalar** e **GerarRodadas.**

Estruturas de Dados

Ordenador

O Ordenador é responsável por ordenar um vetor de url's utilizando o algoritmo QuickSort. Este possui uma peculiaridade na escolha de seu pivô, no caso ele é a mediana das url's da posição inicial, mediana e final do intervalo particionado do vetor. Isso se deve ao fato de que o desempenho de um pivô mal escolhido é bem inferior ao comum, assim faz-se necessário tomar essa precaução na sua escolha.

Escritor de Arquivo

O Escritor de Arquivo é responsável por realizar a escrita de dados em arquivos, no formato de texto. Ademais, essa estrutura também é responsável por fechar o arquivo de maneira adequada.

Fita

A Fita é responsável por realizar o controle de acesso as url's que estão armazenadas no seu respectivo arquivo. Isso é realizado utilizando o Escritor de Arquivos para acessar o arquivo que é identificado pelo número da Fita, ou seja, se temos uma Fita de número **n** a identificação do arquivo é rodada-**n**.txt.

Heap

O Heap é responsável por realizar o armazenamento das url's oriundas das fitas armazenadas na memória física do computador, disco local. Além disso, o Heap também realiza a retirada do seu maior elemento utilizando sua estrutura de árvore em formato vetorial. Para manter sua estrutura (todos os pais devem ser maiores que seus respectivos filhos), a cada elemento adicionado ele é reconstruído.

Leitor de Arquivo

O Leitor de Arquivo é responsável pela leitura dos dados advindos de arquivos. Contudo, essa leitura possui uma peculiaridade, a cada espaço em branco ou quebra de linha encontrada a leitura é parada e o valor obtido, até aquele momento, é devolvido ao utilizador. Ademais, essa estrutura também é responsável por controlar a abertura e o fechamento dos arquivos lidos.

Lista

A Lista é responsável por armazenar as url's advindas do arquivo fornecido pelo usuário. Além disso, essa estrutura é utilizada para simular a limitação de memória citada na introdução deste documento.

URL

A URL é responsável por armazenar o endereço e o número de acessos informados, através do arquivo, pelo usuário. Além disso, essa estrutura é a encarregada de comparar o seu "valor" (número de acesso e/ou ordem alfabética) com outra URL. Isso é fundamental para a ordenação citada anteriormente, pois para ordenarmos algo é necessário estipular uma ideia de melhor, igual ou pior.

GerarRodadas

A função Gerar Rodadas é de extrema importância para o programa, pois utilizando os recursos acima ela não só realiza a leitura das url's do arquivo informado pelo usuário, mas também simula um limite (também informado pelo usuário) de memória. Ou seja, ela é responsável por ler as url's e separar em n arquivos diferentes de acordo com o limite estipulado pelo usuário.

Intercalar

A função Intercalar é responsável por agrupar as url's, separadas anteriormente pela função Gerar Rodadas. Em conjunto a esse processo em que as url's são agrupadas, a ordenação também é realizada. Isso ocorre com utilização da estrutura de dados Heap, citada anteriormente.

Aplicação Exception

A estrutura de dados Aplicação Exception é responsável por realizar o controle das exceções geradas pela aplicação. Para isso, existem classes que herdam os atributos e comportamentos dessa classe por meio de Herança, afim de que a aplicação mantenha um padrão no retorno de erros ao usuário. Além disso, vale ressaltar que as classes em questão são: **Heap Exception, Arquivo Exception, Lista Exception e Argumento Exception**.

Análise de Complexidade

A análise de complexidade do programa irá considerar as duas funções principais do programa, Intercalar e GerarRodadas.

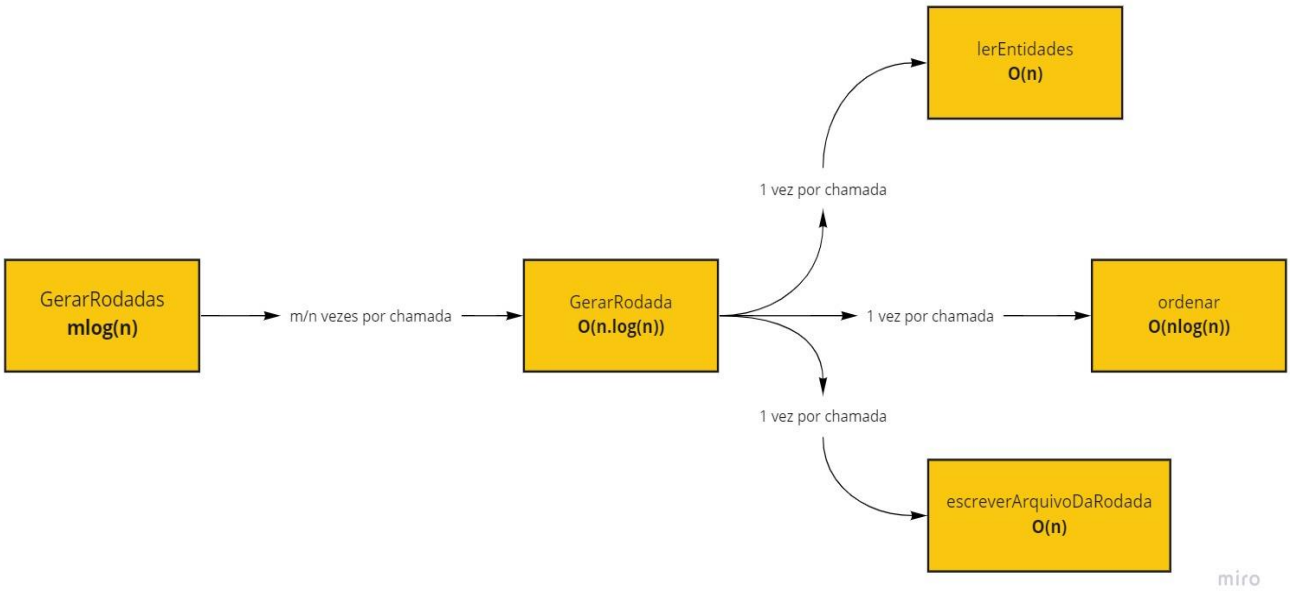
M: Número de url's fornecidas pelo usuário

N: Número do limite url's que devem ser lidos por vez.

K: Número de Fitas geradas.

Gerar Rodadas

A função GerarRodadas depende diretamente e indiretamente de diversas funções. Contudo, darei foco apenas as mais importantes para o programa em questão. Em um primeiro plano, a função principal em análise depende da função GerarRodada **(O(nlog(n))**. Esta por sua vez depende das funções: lerEntidades**(O(n))**, ordenar**(O(nlog(n)))** e escreverArquivoDaRodada**(O(n))**. Então, assim temos a complexidade assintótica da função GerarRodas igual à **O(mlog(n))**, pois a função GerarRodada será executada **m/n vezes** e sendo a complexidade da última função citada igual à **O(n.log(n))**. Logo temos, **(m/n) * O(n.log(n))** que é equivalente à **O(m.log(n))**.



Intercalar

A função Intercalar depende diretamente e indiretamente de diversas funções. Entretanto, como feito anteriormente, a atenção será voltada para as funções mais importantes. Em consequência a isso, temos que a função Intercalar depende das funções: gerarFita(**O(k)**), Fita->read(**O(1)**), Heap->inserir(**O(log(k) * k)**), Heap->pop(**log(k) * k**), escreverLinha(**O(1)**). Assim temos, que a função intercalar possui como complexidade assintótica igual à **O(m^2log(m^2/k)),** pois pela dependência direta das funções citadas acima, a função de maior complexidade é a Heap -> pop(O**(log((m^2) /k) * m)**), no loop em específico, e como essa função é executa m vezes temos **O(m^2log(m^2/k))**. Ademais, vale ressaltar como foram levadas em conta somente as dependências diretas, não há a necessidade de um diagrama.

Análise de Espaço

A análise de espaço desse programa levará em conta pontos específicos da aplicação, pois nela há um gerenciamento inteligente do uso de memória. Logo, temos o construtor do Heap que aloca um espaço equivalente ao número de fitas geradas (número de fintas fornecidas / número de urls que devem ser lidas por vez) e esse espaço é desalocado (externamente à estrutura do Heap) a cada leitura de uma url. Ademais, outro ponto onde há alocação de memória é no uso da lista a qual possui tamanho equivalente ao número de urls que devem ser lidas por vez. Além disso, a Lista também segue a mesma estratégia do Heap para a limpeza da memória.

Orientações para a Compilação e Execução

Navegue até a raiz do projeto e execute o seguinte comando para executar a compilação: **make clean all.**

Para executar o programa basta executar a regra make compile, configurando os parâmetros exigidos pelo programa (nome do arquivo de entrada, nome do arquivo de saída e número de url's que devem ser lidas por vez) no arquivo makefile.

Exemplo:

```
compile:
    $(BIN_FOLDER)$(TARGET)  teste.txt saida1.txt 8
    $(BIN_FOLDER)$(TARGET)  teste.txt saida2.txt 11
    $(BIN_FOLDER)$(TARGET)  teste.txt saida3.txt 4
    $(BIN_FOLDER)$(TARGET)  teste.txt saida4.txt 3
    $(BIN_FOLDER)$(TARGET)  teste.txt saida5.txt 2
    $(BIN_FOLDER)$(TARGET)  teste.txt saida6.txt 1
    $(BIN_FOLDER)$(TARGET)  teste.txt saida6.txt 0
    $(BIN_FOLDER)$(TARGET)  teste teste.txt 1
    $(BIN_FOLDER)$(TARGET)  saida6.txt 1
```

Importante

A estratégia utilizada seguiu da orientação da monitora Raissa a qual orientou que: "O número de fitas serve mais para ajudar na hora de ler (dependendo da sua estratégia de leitura), e na manipulação da intercalação de fitas. Isso não te impede, porém, de usar outra estratégia que não precise do número de fitas.". No caso a estratégia usada não depende do número de fitas informadas, logo torna-se desnecessário informar esse parâmetro.

Conclusão

O trabalho teve como principal objetivo, na minha visão, avaliar como estruturas de dados como Heap e Lista se comportam com uma limitação de espaço e uma variação de entradas. Ademais, outro ponto que vale a pena ser ressaltado é a análise do comportamento de um algoritmo de ordenação eficiente com as mesmas limitações citadas anteriormente.

Bibliografia

Algoritmos - Teoria e Prática - 3º Edição Americana traduzida, por Thomas Cormen

C++ language documentation