Trabalho Prático 3: Um resgate inesperado

Ricardo Dias Avelar
Matrícula: 2019054960

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

ricardodiasmode@hotmail.com

1. Introdução

Esta documentação tem como objetivo facilitar a compreensão do programa que lida com o problema de se comunicar através de mensagens codificadas com as naves que retornarão para a Terra. Essas mensagens precisam ser decodificadas e codificadas. Esse problema foi resolvido pelo programa utilizando um caminho numa árvore binária, que tem como chaves códigos alfanuméricos.

Para resolver o problema, foram utilizados conhecimentos adquiridos em aula para formular a lógica dos algoritmos de árvore binária e dos algoritmos de busca.

A seção 2 trata da implementação do programa, onde é explicado detalhadamente as classes e métodos utilizados no programa. Na seção 3, tratamos das instruções de compilação e execução do programa. Na seção 4, tratamos da complexidade de tempo e espaço dos métodos. E finalmente na seção 5, uma conclusão.

2. Implementação

O código desenvolvido se organiza em:

Main.cpp: Inicializador do programa. Aqui são chamadas as funções essenciais para o funcionamento do programa. Temos como parâmetro inicial argv[1], que é um arquivo .txt passado para a variável **InfoFile**, do tipo ifstream. Essa variável é utilizada para ler o arquivo durante toda a execução do programa. Além disso, temos um while loop que lê cada comando.

Arvore.h: Contém as classes “No” e “Arvore”.

Class No: Classe que tem uma chave chamada Item, um ponteiro do tipo No que aponta para a esquerda, e outro do mesmo tipo que aponta para a direita. É literalmente o nó da árvore.

Class Arvore: Classe da Árvore Binária que irá segurar os códigos alfanuméricos na disposição que queremos.

Arvore.cpp: Contém as funções relacionadas às classes do Arvore.h, sendo as principais:

void CarregarArvore: Cria a árvore de acordo com os dados recebidos.

void Decodificar: Transforma o código lido em uma string seguindo a árvore.

void Codificar: Transforma a string lida em um código que leva à sua montagem por um caminho da árvore.

void Insere: Chama o InsereRecursivo com a raiz.

void Limpa: Chama o ApagaRecursivo e limpa a raiz.

void InsereRecursivo: Insere um nó na árvore, colocando os maiores na direita e os menores na esquerda.

void ApagaRecursivo: Apaga todos os nós da árvore.

Configuração utilizada para testar o programa:

- O programa foi testado no SO Windows 10;
- Linguagem C++;
- Compilado por G++;
- Processador Ryzen 5 3600x e 16GB RAM.

3. Instruções de compilação e execução

Siga as instruções abaixo para executar o programa:

- Entre na pasta do programa e acesse o diretório projeto\bin;
- Cole o arquivo entrada.txt nessa pasta;
- Utilizando um terminal, entre na pasta do programa e execute o arquivo [Makefile] utilizando o comando < make all >;
- Esse comando criará os arquivos .o na pasta obj e o arquivo .out na pasta bin;
- Agora acesse o diretório bin e execute o arquivo [run.out] com o parâmetro [entrada.txt]. O comando será: < run.out entrada.txt >;
- Com esse comando, o programa será iniciado e irá printar no terminal a saída desejada, com o código passado na entrada decodificado ou codificado.

4. Análise de complexidade:

Serão analisadas aqui apenas as funções chamadas, e a análise de todas as outras funções estão incluídas nessas chamadas.

void main - complexidade de tempo: Essa função tem um “while” que é um laço que itera enquanto há linhas a serem lidas no arquivo da variável InfoFile. Dessa forma, a complexidade assintótica de tempo dessa função é $\Theta(n)$, sendo n o número de linhas.

void main - complexidade de espaço: Essa função tem um ponteiro do tipo Arvore que, sem nós, tem complexidade assintótica de espaço $\Theta(1)$. A cada inserção, é alocado um nó, o que faz com que sua complexidade assintótica de espaço seja $\Theta(1+n)$, com n sendo o número de nós e 1 para seu próprio ponteiro.

void CarregarArvore - complexidade de tempo: Essa função tem dois “while” separados, indo de 0 até o final do vetor StringArvore. Portanto, sua complexidade vai ser $\Theta(n)$, sendo n o maior valor que o tamanho de StringArvore irá ter.

void CarregarArvore - complexidade de espaço: Essa função aloca apenas o vetor StringArvore, com tamanho 1000. Portanto, sua complexidade de espaço é $\Theta(1000)$.

void Decodificar - complexidade de tempo: Essa função tem um “for”, de 1 até o final do vetor StringCodigo e outro “for” separado, de 0 até o final do vetor StringToPrint. Em todos os casos, o tamanho de StringCodigo será sempre maior que o tamanho de StringToPrint, portanto iremos ignorar esse último. Sua complexidade de tempo será Θ(n-1).

void Decodificar - complexidade de espaço: Essa função aloca o vetor StringCodigo, com tamanho 2000, e o vetor StringToPrint, com tamanho 1000. Portanto, sua complexidade de espaço é Θ(3000).

void Codificar - complexidade de tempo: Essa função tem um “for” inicial de 1 até o final do vetor StringLida. Dentro deste, há dois “while” separados por uma condição, e ambos caminham até o nó do número desejado. Dentro de cada “while” há outro “while” que randomiza um número. O “while” pode ser interrompido caso o nó em questão não exista. Portanto, sua complexidade de tempo é O((n-1)*m*z), sendo “n” o tamanho do vetor StringLida, “m” sendo o caminho até o nó desejado, e “z” o número de vezes que irá randomizar o número. Há um segundo “for”, fora desse “for” inicial, que estamos desconsiderando pois nunca terá a complexidade de tempo maior que o “for” inicial.

void Codificar - complexidade de espaço: Essa função aloca o vetor StringCodigo e o vetor StringLida, ambos com tamanho 1000. Portanto, sua complexidade de espaço é Θ(2000).

5. Conclusão

Esse trabalho lidou com o problema de se comunicar através de mensagens codificadas com as naves que retornarão para a Terra. Essas mensagens precisam ser decodificadas e codificadas. Esse problema foi resolvido pelo programa utilizando um caminho numa árvore binária, que tem como chaves códigos alfanuméricos. O programa lê um arquivo com comandos, e cada comando diz se o programa vai codificar ou decodificar a mensagem.

Com a solução adotada, pode-se verificar que a resolução do problema foi realizada com uma programação mais bruta, utilizando o mínimo possível de tipos e classes, para que o código ficasse simples e seguisse as instruções à risca.

Por meio da resolução desse trabalho consegui implementar e estudar os métodos da Árvore Binária, fazer buscas nela, e entender como funciona na prática.

Houveram desafios para a implementação da classe Arvore, onde tive que estudar bastante para conseguir implementar com sucesso seus métodos básicos. Além disso, Codificar e Decodificar os arquivos também tomou bastante tempo, haja vista que não era possível utilizar o tipo std::string e foquei em utilizar apenas métodos da biblioteca padrão.

References

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.