Documentação do Trabalho Prático III da disciplina de Estruturas de Dados

Lucca Silva Medeiros - 2019054773

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

luccasilva@dcc.ufmg.br

1. Introdução

Esta documentação lida com o problema da exploração do Composto Z, material essencial para desenvolver as tecnologias capazes de restaurar as condições de vida em uma Terra pós-apocalíptica. Após o sucesso da fase exploração, e o sucesso na fase de organização dos dados tem-se início o processo de retorno do composto Z coletado para a Terra. Vê-se necessário criar um método de codificação das mensagens para evitar possíveis pilhagens de mercenários espaciais, logo foi desenvolvido um programa capaz de codificar e decodificar as mensagens envolvidas no processo. Esta documentação tem como objetivo detalhar, analisar e discutir o algoritmo de cifragem implementado.

Esta documentação está separada em seções. Na segunda é apresentado a implementação da solução do problema. Na terceira as instruções de compilação e execução do programa. Na quarta a Análise de complexidade de tempo e espaço de cada função e por último a conclusão do Trabalho.

2. Implementação

O código possui o objeto BinaryTree, este é uma estrutura de dados do tipo arvore binária. No programa ela é a responsável por dispor os caracteres recebidos na entrada seguindo a seguinte lógica: As letras são dispostas na arvore na sequência em que aparecem e de modo que, em cada nó, uma letra menor esteja sempre à esquerda e uma letra maior, sempre à direita. Essa estrutura possibilita codificar e decodificar as mensagens. Esse objeto possui os seguintes métodos:

- BinaryTree: Construtor da árvore.

- ~BinaryTree: Destrutor da árvore.

- Insert: responsável por inserir um nó na árvore, respeitando as regras de inserção descritas acima.

- Remove: responsável por remover um nó na árvore.

- dCommand: responsável por decodificar um comando de entrada através do caminhamento pela árvore.

- cCommand: responsável por codificar um comando de entrada através do caminhamento pela árvore.

- recursiveInsertion: método de inserção recursiva.

- recursiveRemotion: método de remoção recursiva.

- Previous: método auxiliar de remoção.

- Search: responsável por procurar um nó na árvore.

- recursiveSearch: método de procura recursiva, utilizado no programa para realizar a codificação letra por letra da mensagem de entrada.

O código também conta com o objeto Node, responsável por armazenar os valores do nó da árvore: o Item, ou seja o char do nó, e os apontadores para os nós da esquerda e da direita.

A função main está no arquivo Main.cc, presente na pasta src. Ela é responsável por ler os comandos do arquivo de entrada e executar cada um individualmente. Na função readFile, identifica-se a ordem de comando através dos dois primeiros caracteres da linha:

- Caso “A:”, o restante da string é passada para a função Acommand que insere letra por letra na árvore binária.

- Caso “D:”, o restante da string é passada para o método dCommand que decodifica a mensagem e a imprime no terminal. Durante o processo de decodificação é necessário identificar se o número da mensagem é par ou ímpar, essa tarefa é realizada pela função isEven.

- Caso “C:”, o restante da string é passada para o método cCommand que codifica a mensagem e a imprime no terminal. O processo de codificação é feito através dos métodos Search e recursiveSearch, os quais procuram na árvore binária o caractere em questão e o codifica em uma sequência de números impares e/ou pares dependendo da sua posição na árvore. As funções randOddNumber e randEvenNumber são utilizadas para se imprimir valores aleatórios impares ou pares respectivamente.

A configuração utilizada para testar o programa está especificada abaixo:

– **Sistema Operacional**: Windows 10 Home

– **Linguagem de programação implementada:** C++

– **Compilador utilizado:** mingw / G++

– **Processador**: Inter(R) Core(TM) i7-8750H CPU @2.20GHz 2.21GHz

– **Quantidade de memória RAM:** 16,0 GB

3. Instruções de compilação e execução

1- Baixe o arquivo.zip e extraia a pasta tp3;

2- A pasta deverá conter os seguintes arquivos;

/ bin – run.out e entrada.txt

/ include – BinaryTree.h

/ obj

/ src –Main.cc e BinaryTree.cc

/ Makefile

3- Utilizando um terminal, abra o diretório da pasta tp3 e execute o arquivo [Makefile] utilizando o seguinte comando: < make >;

4- Com esse comando o Makefile irá compilar o programa e deverá criar com sucesso os objetos: ./obj/Main.o ./obj/BinaryTree.o.

5- Utilizando um terminal, abra o diretório da pasta bin e execute o arquivo [run.out] utilizando o seguinte comando: < run.out argumento1.txt >. O argumento 1 deverá ser o arquivo.txt que contém os dados de entrada.

6- Com isso o programa irá executar e a saída esperada aparecerá no terminal.

4. Análise de Complexidade

Objeto Node:

Função Node – complexidade de tempo – essa função realiza operações de atribuição em tempo O(1).

Função Node – complexidade de espaço – essa função possui complexidade de espaço O(1).

Objeto binaryTree:

Função binaryTree – complexidade de tempo – essa função realiza operações de atribuição em tempo O(1).

Função binaryTree – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função Insertion – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de chamadas realizadas recursivamente da função recursiveInsertion.

Função Insertion – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função recursiveInsertion – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de níveis caminhados pela árvore para realizar a inserção em questão.

Função recursiveInsertion – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função isEven – complexidade de tempo – essa função realiza operações de atribuição em tempo O(1).

Função isEven – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função dCommand – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de caracteres da string de entrada.

Função dCommand – complexidade de espaço – essa função possui complexidade de espaço O(1).

Funções randEvenNumber e randOddNumber – complexidade de tempo – essas funções possuem complexidade de tempo O(n), sendo n o número de tentativas de se gerar o número aleatório em questão.

Funções randEvenNumber e randOddNumber –essas funções possuem complexidade de espaço O(1).

Função Search – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de chamadas realizadas recursivamente da função recursiveSearch.

Função Search – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função recursiveSearch – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de níveis caminhados pela árvore para realizar a busca em questão.

Função recursiveSearch – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função cCommand – complexidade de tempo – essa função possui complexidade de tempo O(n*m), sendo n o número de caracteres da string de entrada e m o número de chamadas realizadas recursivamente da função recursiveSearch.

Função cCommand – complexidade de espaço – essa função possui complexidade de espaço O(1).

Main:

Função Acommand – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de caracteres da string de entrada.

Função Acommand – complexidade de espaço – essa função possui complexidade de espaço O(n), sendo n o número de caracteres da string de entrada.

Função readFile – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de linhas do arquivo de entrada.

Função readFile – complexidade de espaço – essa função possui complexidade de espaço O(1).

5. Conclusão

Este trabalho lidou com o problema de codificação e decodificação de mensagens da missão de extração do Composto Z para evitar possíveis pilhagens de mercenários espaciais, o qual a abordagem utilizada para resolução foi a implementação de um programa em C++ capaz de codificar e decodificar as mensagens. Por meio da resolução desse trabalho, foi possível praticar os conceitos relacionadas à disciplina Estrutura de Dados como a implementação de uma árvore binária, e ver na prática sua aplicação e execução para a solução de um problema palpável. Além de relembrar conceitos relacionados à disciplina de Programação e Desenvolvimento de Software II como programação orientada à Objetos, boas práticas de programação e modularização de código.

6. Referências

- Chaimowicz, L. and Prates, R. (2021). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.