Instituto Federal de Minas Gerais

Campus Formiga

João Paulo de Souza

Trabalho de Implementação de Tipos Estruturados de Dados de Árvores Binárias

Formiga/MG

2018

Sumário

Introdução 3
Desenvolvimento 4
Problema Inicial 4
Estruturas/TADs 5
Funções do programa 7
Biblioteca “file.h” 7
Biblioteca “tree.h” 8
Exemplo de funcionamento do programa 12
Conclusão 14
Referências 15

Lista de Imagens

Figura 1 - Imagem exemplo de uma árvore binária 4
Figura 2 - Imagem da estrutura de um ramo da árvore binária 5
Figura 3 - Exemplo de árvore desbalanceada 6
Figura 4 - Exemplo de funcionamento das rotações da árvore AVL 6
Figura 5 - Estrutura para leitura do arquivo de entrada 7
Figura 6 - Imagem exemplo de rotação para direita 9
Figura 7 - Imagem exemplo de rotação para esquerda 9
Figura 8 - Imagem exemplo de duas rotações, uma para esquerda e outra para direita 10
Figura 9 - Exemplo de Arquivo de Entrada 12
Figura 10 - Exemplo de arquivo de saída 13
Figura 11 - Exemplo de saída de console 13
Figura 12 - Exemplo de comando de execução 14

Introdução

Este trabalha tem como objetivo relatar o desenvolvimento de um software em linguagem C utilizando estruturas de árvore binária de busca com balanceamento de árvore quando necessário. Neste software são implementas funções de inserção, remoção, busca e exibição de dados.

O software trabalha com um arquivo de entrada que passa comandos para serem executados pelo próprio software, comandos que seguem, os comandos citados acima. O arquivo é lido, interpretado e os comandos são executados e alguns deles como busca e impressão são salvos em um arquivo de saída para servir como relatório dos acontecimentos na estrutura da árvore.

No fim a árvore deve ser destruída, ou seja, os nós, devem ser removidos e o espaço de memória liberado.

Realizando todas essas operações o programa conclui todas as suas tarefas básicas de funcionamento.

Desenvolvimento

O programa requerido no trabalho como solicitado em aula, foi desenvolvido em linguagem de programação C, utilizando apenas bibliotecas próprias da linguagem ou desenvolvidas pelo desenvolvedor(aluno).

Assim, a seguir serão apresentadas, a estruturação das TADs(Tipo Abstrato de Dados), o problema inicial, as funções do programa e por fim demonstrações de funcionamento básico.

Problema Inicial

O solicitado na documentação do trabalho, é realizar o desenvolvimento de um programa que gerencie todo o ciclo de vida de dados em uma árvore binária, inserção de dados, remoção de dados, exibição dos dados na árvore em um dos três percursos (pre-ordem, em-ordem e pós-ordem) busca e destruição da árvore.

O programa deve manter a regra básica de uma árvore binária, o valor da raiz é sempre maior que seu valor associado à esquerda, e ao mesmo tempo menor que seu valor associado a direita, para que a árvore se mantenha concisa.

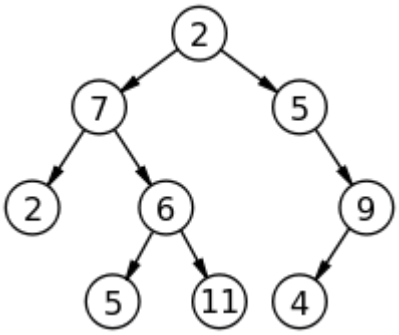

Figura 1 - Imagem exemplo de uma árvore binária

O programa deverá receber um arquivo texto com comandos a serem seguidos pelo programa. Os comandos são:

- “INCLUI ‘número’”, comando que insere um valor na árvore, valor que está logo em seguida na mesma linha do comando.
- “EXCLUI ‘número’”, comando que exclui um valor na árvore (se o valor não estiver presente nada acontecerá), valor que está logo em seguida na mesma linha do comando.

- “BUSCA ‘número’”, comando que busca um determinado valor na árvore, se o valor existir, mostra o mesmo, senão será mostrada a frase: “Elemento não encontrado”, o valor está logo em seguida do comando, como nos outros comandos acima. Este comando é refletido no arquivo de saída.
- “IMPRIME ‘percurso’”, comando que salva no arquivo e mostra em console os valores presentes na árvore em um determinado percurso, que está após o comando. Os percursos são pré-ordem, em ordem e pós-ordem.

O arquivo de entrada tem vários comandos, que são iguais a estes apresentados acima, quando estiver no final do arquivo, um comando “FIM” determinará o final das instruções.

Já o arquivo de saída, deve conter os percursos a serem impressos e os valores buscados (encontrados e não encontrados).

Estruturas/TADs

A estrutura básica do programa é a ABB (árvore binária de busca), que consistem em elementos organizados de forma que haja uma raiz e valores menores que a raiz, fiquem a esquerda e valores maiores a direita, seguindo essa ordem para os outros ramos da árvore inclusive.

No programa a TAD árvore foi definida seguindo o conceito padrão com uma peculiaridade, tem-se uma variável inteira para guardar o valor, dois ponteiros da estrutura para os ramos esquerda e direita do ramo estrutural, e pôr fim a peculiaridade que é uma variável inteira para guardar a altura do ramo.

```
struct BRANCH{
    int value;
    branch right;
    branch left;
    int height;
};
```

Figura 2 - Imagem da estrutura de um ramo da árvore binária

Junto com a forma ABB, deve-se haver um balanceamento para que a árvore não tenha um dos lados muito maior que o outro e perca justamente o que a deixa eficiente, uma pesquisa rápida.

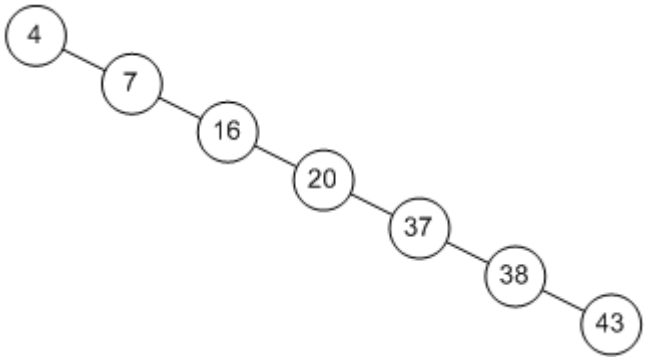

Figura 3 - Exemplo de árvore desbalanceada

Assim utiliza-se a completação com técnica de AVL, que consiste em balancear a árvore para que não haja esses problemas. Então as inserções e remoções devem sempre verificar a necessidade de balanceamento da árvore.

A técnica para a árvore AVL é a verificação de um fator, chamado fator de balanceamento, onde conta-se a partir do nó base como 0 e vai contando os nós até que chegue no nó folha(nó não aponta para nenhum outro mais), conta-se assim as duas pontas(esquerda e direita)a partir do nó base, esquerda conta-se positivamente e na direita conta-se negativamente, se a diferença do fator de balanceamento entre as pontas do nó base for igual a 2 ou mais em módulo, essa árvore precisa de balanceamento.

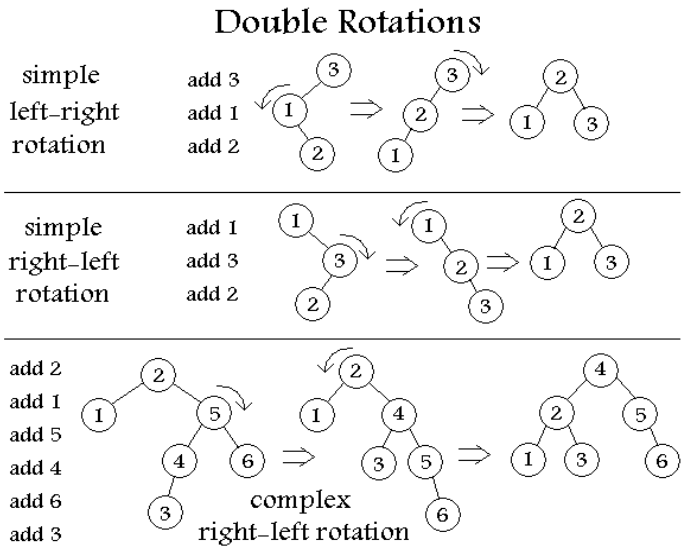

Figura 4 - Exemplo de funcionamento das rotações da árvore AVL

Como na figura 4, após a inserção do valor 2, a ponta esquerda a partir do nó base (nesse caso o nó raiz) ficou com um fator de balanceamento de 2, e assim precisou ser rotacionado e o nó base foi substituído por um outro valor melhor. A imagem mostra também uma rotação para direita e uma com duas rotações, uma para direita e outra para esquerda.

Funções do programa

Biblioteca “file.h”

A biblioteca file tem uma estrutura para recebimento dos dados chamada ‘read’, que consiste em uma variável do tipo inteiro para recebimento do valor e uma variável do tipo ‘string’ para o recebimento das instruções. Segue abaixo imagem demostrando a estrutura.

```
typedef struct READ{
    int value;
    char command[25];
}read;
```

Figura 5 - Estrutura para leitura do arquivo de entrada

A biblioteca ‘file’ além da estrutura possui as seguintes funções:

- read* read_file(char path[]) – Função que a partir do caminho fornecido por ‘string’, abre o arquivo e realiza a leitura utilizando a estrutura acima, por um ponteiro dessa estrutura que no fim trabalha como vetor a partir da função malloc para alocação de memória, para que isso ocorra a necessário a utilização de uma função disponível apenas no arquivo.c dessa biblioteca que verifica a quantidade de instruções no arquivo, assim a quantidade de alocação de memória é feita com base no tamanho fornecido por essa função. Então o arquivo é aberto, e as funções lidas, e cada função é inserida no vetor alocado da estrutura para ser retornado a partir de seu ponteiro e utilizado na função principal.
- void save_preorder(branch root, FILE* file) – Função para salvar os dados de forma recursiva no arquivo, utilizando o percurso pré-ordem, então salva o valor no arquivo, chama recursivo a esquerda e chama recursivo a direita. Recebe por parâmetro o nó raiz e o ponteiro do arquivo.

- void save_inorder(branch node, FILE* file) – Função para salvar os dados de forma recursiva no arquivo, utilizando o percurso em ordem, então chama recursivo para esquerda, salva o valor no arquivo, e chama recursivo na direita. Recebe por parâmetro o nó raiz e o ponteiro do arquivo.
- void save_postorder(branch node, FILE* file) – Função para salvar os dados de forma recursiva no arquivo, utilizando o percurso pós-ordem, então chama recursivo para esquerda, depois chama recursivo para direita, e por fim salva o valor no arquivo. Recebe por parâmetro o nó raiz e o ponteiro do arquivo.

Biblioteca “tree.h”

A biblioteca “tree” utiliza a estrutura demonstrada na imagem 2 para criação dos nós da árvore binária, sendo uma estrutura chamada “branch”, que possui dois ponteiros do próprio tipo da estrutura, sendo um para direita e outro para esquerda, esses guardam o endereço de memória dos nós que são menor que o nó base(esquerda) e o que é maior que o nó base(direita). Além de possui uma variável inteira para informação e uma variável inteira para altura do nó.

A biblioteca “tree” possui as seguintes funções:

- branch insert_on_tree(branch root, int value) – Função que insere um valor na árvore binária que recebe por parâmetro o nó raiz, e o valor a ser inserido na árvore. A função realiza a ligação com o nó raiz, ou caso for o primeiro nó somente aloca memória para ele e esse nó fica sendo o nó raiz, e sua altura será 0. Caso não seja o nó raiz, verifica se o valor passado por parâmetro é maior ou menor que a raiz, e assim vai indo recursivamente para esquerda se for menor ou direita se for maior, verificando sua posição de acordo com o nó base atual, iniciando pela raiz. Nos dois casos, é retornado na função a referência para raiz e quando termina de verificar qual a posição adequada na árvore, atualiza a altura do novo nó. A função de inserir, verifica a necessidade de rotacionar a árvore para evitar desbalanceamento, verificando os 4 possíveis casos que necessitam de rotação.
    1. Caso de rotação para direita: Se o fator de balanceamento (advindo da função que retorna a altura da esquerda subtraída da altura da direita) do novo nó for maior que 1 e o fator de balanceamento da sua esquerda

for maior ou igual a 0, é necessário realizar a rotação para direita. Segue imagem demostrando abaixo:

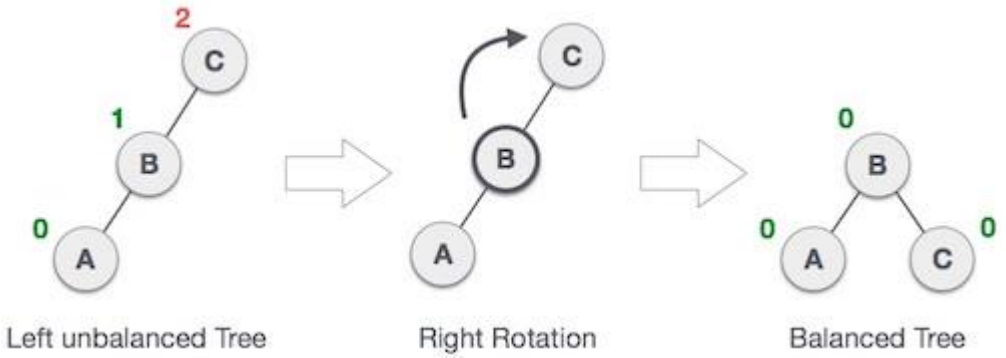

Figura 6 - Imagem exemplo de rotação para direita

Como dito acima, o fator de balanceamento do nó base é maior que 1 em módulo, assim se faz necessário a troca do nó base por um nó mais adequado que deixa a árvore melhor balanceada.

2. Caso de rotação para esquerda: Se o fator de balanceamento (advinda da mesma função utilizada acima) do novo nó for menor que -1 e o fator de balanceamento do nó a direita dele for menor igual a 0, é necessário rotacionar a esquerda.

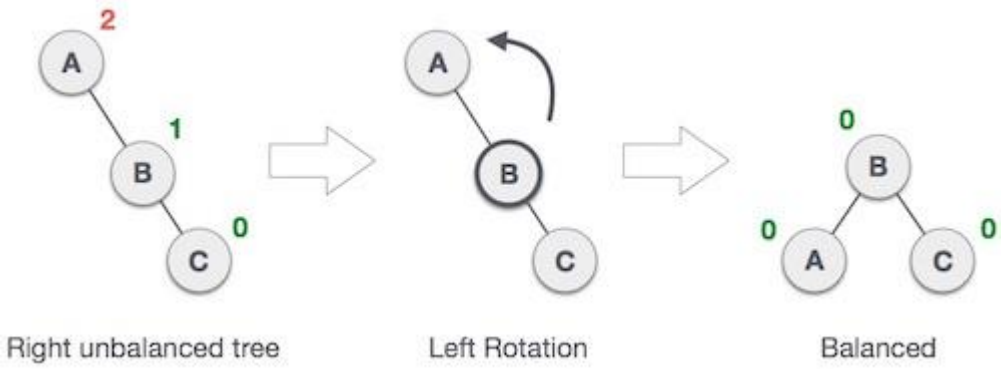

Figura 7 - Imagem exemplo de rotação para esquerda

Como dito acima, o fator de balanceamento do nó base era maior que 1 em módulo, assim se fez necessário rotacionar para esquerda, encontrando um nó base mais apropriado para manter a árvore balanceada.

3. Caso duas rotações, uma para direita e outra para esquerda: Se o fator de balanceamento do novo nó for maior que um e o valor da sua

esquerda for menor que o valor do novo nó, então se faz necessário uma rotação a esquerda, e logo depois uma rotação logo acima a direita.

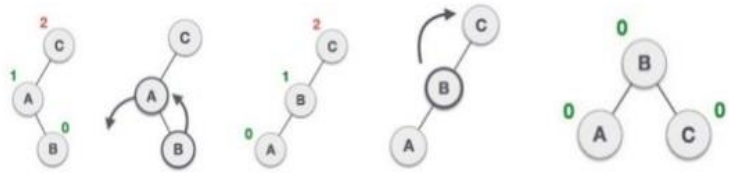

Figura 8 - Imagem exemplo de duas rotações, uma para esquerda e outra para direita

Como dito acima, primeiro realiza-se uma rotação a esquerda para deixar a parte esquerda da árvore ou sub árvore balanceada, e depois rotaciona para direita para terminar o balanceamento.

4. E por fim o último caso, quando rotaciona-se uma para direita e depois uma para esquerda: Se o fator de balanceamento do novo nó for menor que -1 e o valor do nó a direita for maior que o valor do novo nó, então rotaciona-se uma vez para direita, e logo acima rotaciona-se para esquerda.

- void show_tree_preorder(branch root) – Função para mostrar os elementos da árvore no console utilizando o percurso pré-ordem. Primeiro mostra o elemento, vai para esquerda recursivamente, empilhando os valores da esquerda, depois para direita também de forma recursiva, empilhando os valores da direta. Recebe por parâmetro o nó raiz.
- void show_tree_inorder(branch root) – Função para mostrar os elementos da árvore no console utilizando o percurso em ordem. Primeiro vai para esquerda de forma recursiva, empilhando os valores da esquerda, mostra o elemento, depois vai para direita de forma recursiva, empilhando os elementos da direita. Recebe por parâmetro o nó raiz.
- void show_tree_postorder(branch root) – Função para mostrar os elementos da árvore no console utilizando o percurso pós ordem. Primeiro vai para esquerda de forma recursiva, empilhando os valores da esquerda, depois vai para direita de forma recursiva, empilhando os valores da direita e por fim mostra o elemento. Recebe por parâmetro o nó raiz.

- int calculate_branch_degree(branch br) – Função para calcular o grau do nó passado por parâmetro. Se os nós, à direita e a esquerda do nó passado por parâmetro forem nulos, então o grau dele é 0. Se um deles não é nulo e outro é nulo, então seu grau é 1. E por fim se os dois forem diferentes de nulo, então o grau é 2.
- int calculate_level(branch root,branch br) – Função para calcular o nível de um nó. Recebe por parâmetro o nó raiz e um outro nó, então tenta chegar nesse outro nó caminhando da raiz até ele, para que isso seja feito, verifica se o valor do nó é maior ou menor que a raiz, e assim vai decidindo entre seguir para direita ou esquerda e para quando chegar nesse nó, retornando o nível dele.
- branch verify_value_on_tree(branch br, int value) – Função para verificar se um valor existe na árvore. Recebe por parâmetro o nó raiz e o valor a ser verificado. Então guarda o nó raiz em uma variável auxiliar, e vai verificando se o valor é menor ou maior que o nó base atual e assim vai caminhando entre direita e esquerda até chegar no valor ou retornar nulo, mostrando que não encontrou o valor na árvore.
- branch delete_value_on_tree(branch node, int value) – Função para remover um nó da árvore e retorna a referência do nó raiz, recebe por parâmetro o ponteiro do nó raiz e o valor a ser removido. Se o valor removido for o valor da raiz, ou um valor que tenha filhos, usa uma função para ir até o máximo a esquerda, e depois um a direita para substituir esse valor a ser removido, e depois de colocado esse valor em uma ponta, enfim remove o elemento e libera seu espaço de memória do nó. Caso não seja o nó raiz ou um elemento com filhos, só precisa dar free no espaço de memória alocado.

  A remoção também necessita de rotações caso esteja desbalanceada logo após uma remoção, então segue as regras de rotação impostas da função de inserção, seguindo os mesmos 4 casos. Enfim como já foi demostrada acima, não será novamente demostrada para a função de remoção, pois segue os casos de forma idêntica.
- branch destroy_tree(branch root) – Função para destruir a estrutura da árvore binária, usa a função de remover os nós da árvore, removendo sempre o nó da raiz, uma nova raiz sempre será escolhida e continua esse processo até que o último nó seja removido e a árvore esteja vazia.

Exemplo de funcionamento do programa

O funcionamento do programa se dá por o recebimento de um arquivo de entrada, com comandos para serem executados, inclui, exclui, imprime e busca. Esses comandos estão relacionados as estruturas de árvore já explicadas acima.

Para iniciar o programa é necessário executar o seguinte comando no console Linux: manager.exe ‘arquivo de entrada’ ‘arquivo de saída’. Como é possível observar, “manager” é o nome de execução da aplicação, que necessita de um arquivo de entrada como já dito e de um nome ou caminho para um arquivo de saída que será criado com dados das funções executadas, como um relatório. Abaixo segue um exemplo de um arquivo de entrada, de um arquivo de saída, exemplo de saída em console e exemplo de comando de execução.

```
INCLUI 81
IMPRIME INORDEM
IMPRIME INORDEM
IMPRIME INORDEM
IMPRIME INORDEM
INCLUI 108
IMPRIME POSORDEM
EXCLUI 81
INCLUI 957
BUSCA 880
BUSCA 105
EXCLUI 81
BUSCA 159
IMPRIME POSORDEM
IMPRIME PREORDEM
BUSCA 108
IMPRIME POSORDEM
INCLUI 297
INCLUI 894
IMPRIME POSORDEM
EXCLUI 894
IMPRIME PREORDEM
IMPRIME INORDEM
IMPRIME PREORDEM
BUSCA 703
BUSCA 957
BUSCA 423
IMPRIME INORDEM
EXCLUI 894
BUSCA 173
EXCLUI 297
INCLUI 996
INCLUI 319
INCLUI 555
BUSCA 81
BUSCA 574
IMPRIME INORDEM
INCLUI 878
BUSCA 776
IMPRIME INORDEM
BUSCA 315
IMPRIME INORDEM
IMPRIME PREORDEM
IMPRIME INORDEM
EXCLUI 297
IMPRIME POSORDEM
IMPRIME INORDEM
INCLUI 136
IMPRIME POSORDEM
IMPRIME INORDEM
IMPRIME PREORDEM
INCLUI 603
INCLUI 391
EXCLUI 996
BUSCA 957
BUSCA 32
BUSCA 996
BUSCA 941
FIM
```

Figura 9 - Exemplo de Arquivo de Entrada

```
<(81)>
<(81)>
<(81)>
<(81)>
<(108)(81)>
Elemento 880 não encontrado
Elemento 105 não encontrado
Elemento 159 não encontrado
<(957)(108)>
<(108)(957)>
108
<(957)(108)>
<(108)(957)(894)(297)>
<(297)(108)(957)>
<(108)(297)(957)>
<(297)(108)(957)>
Elemento 703 não encontrado
957
Elemento 423 não encontrado
<(108)(297)(957)>
Elemento 173 não encontrado
Elemento 81 não encontrado
Elemento 574 não encontrado
<(108)(319)(957)(555)(996)>
Elemento 776 não encontrado
<(108)(319)(957)(555)(878)(996)>
Elemento 315 não encontrado
<(108)(319)(957)(555)(878)(996)>
<(319)(108)(957)(555)(878)(996)>
<(108)(319)(957)(555)(878)(996)>
<(108)(957)(555)(878)(996)(319)>
<(108)(319)(957)(555)(878)(996)>
<(108)(136)(957)(555)(878)(996)(319)>
<(108)(136)(319)(957)(555)(878)(996)>
<(319)(108)(136)(957)(555)(878)(996)>
957
Elemento 32 não encontrado
Elemento 996 não encontrado
Elemento 941 não encontrado
```

Figura 10 - Exemplo de arquivo de saída

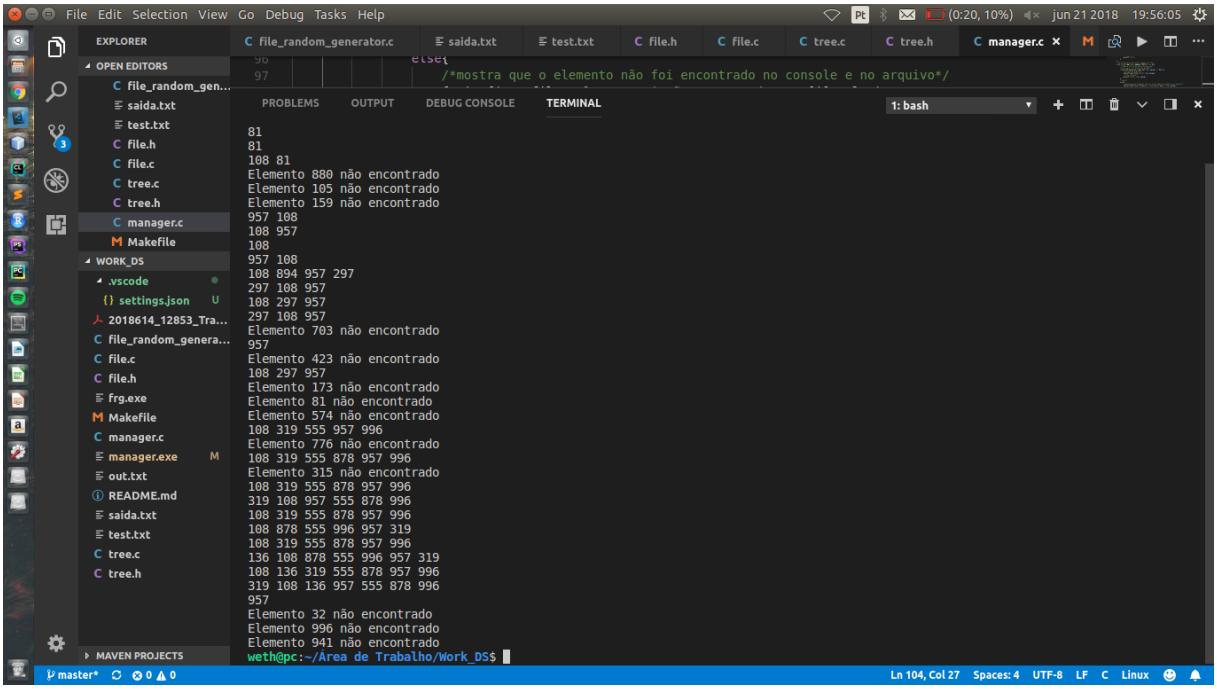

Figura 11 - Exemplo de saída de console

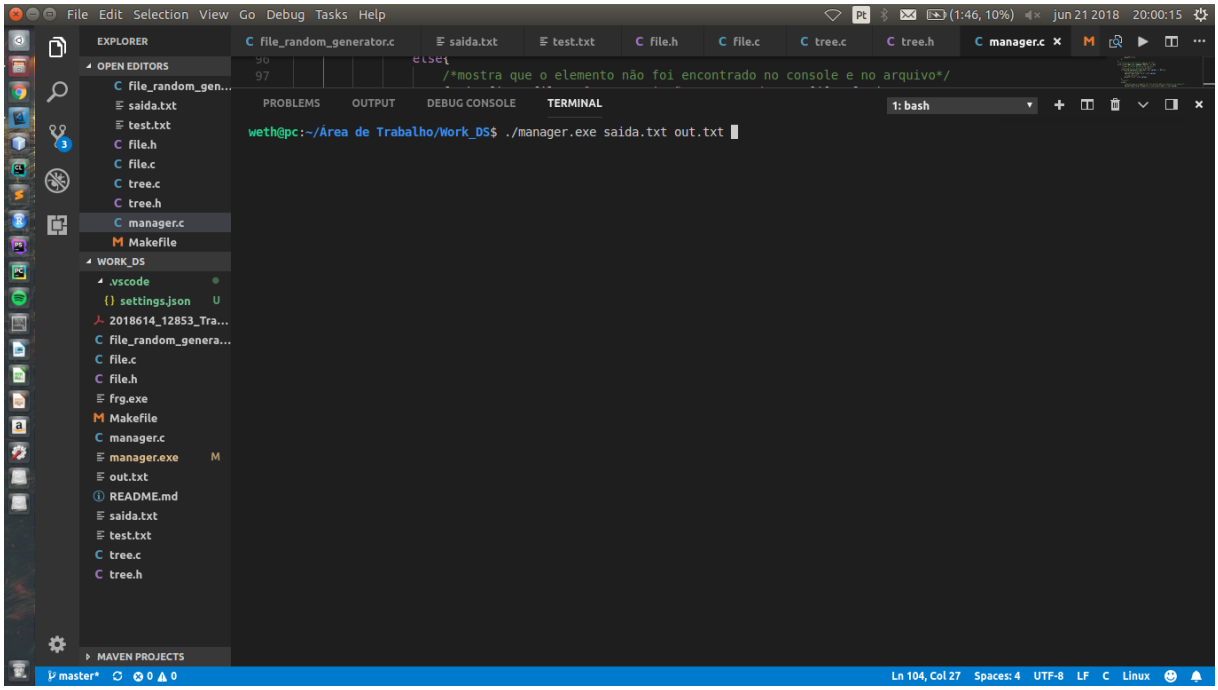

Figura 12 - Exemplo de comando de execução

Conclusão

As aplicações para estruturas de árvores, não somente binárias como as usadas neste trabalho, mas para ‘n’ nós, são inegáveis, quando o conjunto de dados é muito grande é necessita-se de uma pesquisa rápida, é um dos melhores métodos de organização de dados. Entretanto é uma estrutura que precisa de controles para gerenciar bem as rotações e casos necessários para elas, pois são movimentados ponteiros e por ser procedimento recursivo, a referência deve ser sempre guardada. Então desenvolver este trabalho realmente agregou em experiência, principalmente na abstração de organização de dados e em princípios de recursão.

Referências

Algoritmo de inserção de dados da árvore binária com balanceamento e rotações. Disponível em: <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>. Acesso em: 21 jun. 2018.

Algoritmo de remoção de dados da árvore binária com balanceamento e rotações. Disponível em: <https://www.geeksforgeeks.org/avl-tree-set-2-deletion/>. Acesso em: 21 jun, 2018.

Informações sobre árvores binárias e suas aplicações. Disponível em: https://technologyasilearn.wordpress.com/2015/03/09/avl-trees-and-its-applications/>. Acesso em: 21 jun. 2018.

Formatos de árvores e rotações para alguns casos. Disponível em: <https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm>. Acesso em: 21 jun. 2018.

Rotações simples para direita e esquerda. Disponível em: <https://www.slideshare.net/YakshJethva/data-structure-avl-tree-balanced-binary-search-tree>. Acesso em: 21 jun. 2018