IMPLEMENTAÇÃO DE ÁRVORE AVL

Implementação de árvore binária de busca AVL para armazenamento de produtos de um mercado.

Eduardo de Figueiredo Costa

17.10.2022

Universidade Federal de Pelotas

Centro de Desenvolvimento Tecnológico
Algoritmos e Estrutura de Dados

Introdução:

Programa de uma ABB AVL de um mercado, com a finalidade de armazenar o código do produto, validade e valor.

TAD:

1. main.c
2. AVL.c
3. AVL.h (protótipos:)
    a. typedef struct date Date;
    b. typedef struct marketplace Marketplace;
    c. int menu();
    d. int maxx(int x, int y);
    e. int h(Marketplace *no);
    f. Marketplace *rotation_left(Marketplace *root);
    g. Marketplace *rotation_right(Marketplace *root);
    h. Marketplace *left_right(Marketplace *root);
    i. Marketplace *right_left(Marketplace *no);
    j. Marketplace *create_nodo(long int key);
    k. char *read_string(char *product);
    l. Marketplace *balance(Marketplace *root, long int key,

        char*product, int month, int year, float value);

    m. Marketplace *insert_tree(Marketplace *root, long int key,

        char *product, int month, int year, float value);

    n. Marketplace *removee(Marketplace *root, long int key);
    o. int search_menu();
    p. Marketplace *search_nodo (Marketplace *root, long int code);
    q. void *edit_element(Marketplace *root);
    r. void printer(Marketplace *root);
    s. int size_tree(Marketplace *no);
    t. void tree_startup(Marketplace *root);
    u. void tree(Marketplace *root, int depth, char *way, int direita);
    v. Marketplace *destroy(Marketplace *root);
    w. void free_root(Marketplace *root);
    x. void free_tree (Marketplace *root);

Estilos de escrita de substituição de espaços:

1. Funções: Snake case.
2. Variáveis: Camel case.

Estruturas:

Foram usadas duas structs para o programa, a “date” é uma struct utilizada dentro da struct “marketplace”, ela utiliza duas variáveis int para armazenar mês e ano da validade do produto.

```
struct date{
   int month;
   int year;
};
typedef struct date Date;
```

A struct denominada “Marketplace” é a estrutura de cada um dos nós pertencentes à árvore binária de busca (ABB, contendo os seguintes campos:

```
struct marketplace{
   long int code;                  code será o código do produto;
   char *product;                  irá apontar para o nome do produto;
   Date validity;                  var date para a validade;
   float value;                    valor do produto
   int h;                          altura do nó na árvore
   struct marketplace *left;       filho esquerdo
   struct marketplace *right;      filho direito
};
typedef struct marketplace Marketplace;
```

Funções:

1. int menu();
   1.1. Menu para as funções do programa (insere, busca, edita, remove, imprime crescente, número de elementos, altura da árvore) que retorná um inteiro para o switch da main, que por sua vez fará a operação solicitada.

2. int maxx(int x, int y);
   2.1. Retorna o maior valor entre dois inteiros.

3. int h(Marketplace *no);
   3.1. Retorna a altura do nó passado à ela.

4. Marketplace *rotation_left(Marketplace *root);
   4.1. Rotação à esquerda, cria um no aux = root-> direita;
   4.2. root->direita=aux->esquerda (ou root->direita->esquerda);
   4.3. aux->esquerda =root;
   4.4. recalcula a altura de ambos;
   4.5. retorna aux para a raiz (primeiro nó, pai de todos).

5. Marketplace *rotation_right(Marketplace *root);
   5.1. Rotação à direita, cria um nó aux=root->esquerda;
   5.2. Para então, mudar root->esquerda para aux->direita;
   5.3. aux=direita->root;
   5.4. recalcula a altura de ambos;
   5.5. retorna aux para a raiz (primeiro nó, pai de todos).

6. Marketplace *left_right(Marketplace *root);
   6.1. Rotação esquerda;
   6.2. retorna direita.

7. Marketplace *right_left(Marketplace *no);
    7.1. Rotação direita
    7.2. retorna rotação esquerda.

8. Marketplace *create_nodo(long int key);
    8.1. Cria um nó=NULL;
    8.2. usa malloc pra criar ele com o campo da estrutura Marketplace dinamicamente;
    8.3. se não há memória encerra retornando 1 ao programa.
    8.4. inicializa os campos do nó (code, validity,..) e só insere a chave (que foi passada por parâmetro na função) no campo code da estrutura ;
    8.5. retorna o nó;

9. char *read_string(char *product);
    9.1. Função para leitura de string;
    9.2. Aloca dinamicamento no ponteiro product 25 espaços de char (product[0-24]);
    9.3. Lê a palavra até um enter;
    9.4. realoca tamanho do ponteiro para não haver espaço não usado que o ponteiro aponta(tamanho da palavra possuirá um espaço a mais no final com \0 que representa o enter do teclado);
    9.5. retorna o ponteiro product.

10. Marketplace *balance(Marketplace *root, long int key, char*product, int month, int year, float value);
    10.1. Função para balancear a árvore;
    10.2. OBS: pelo insere nó em árvore (insert_tree) precisa passar por parâmetro os campos do novo nó que será balanceado;
    10.3. se (key < root->code)
        10.3.1. root->esquerda =insere nó em árvore root->esquerda
        10.3.2. se (FB = 2)
            10.3.2.1. se (key < root->esquerda->code)
                10.3.2.1.1. root=rotation_right(root);
            10.3.2.2. senão
                10.3.2.2.1. root=left_right(root);
        10.3.3. senão
    10.4. se (key > root->code)
        10.4.1. root->direita = insere no em árvore root->direita;

10.4.2. se (FB = 2)

10.4.2.1. se (key > root->direita->code)

10.4.2.1.1. root=rotation_left(root);

10.4.2.2. senão

root=right_left(root);

10.5. restabelece altura do root;

10.6. retorna root;

11. Marketplace *insert_tree(Marketplace *root, long int key, char *product, int month, int year, float value);

11.1. se nó null;

11.1.1. cria nó

11.1.2. insere os elementos product, validity e value;

11.2. passa o nó para a função de balanceamento;

11.3. retorna nó;

12. Marketplace *removee(Marketplace *root, long int key);

12.1. se null retorna null;

12.2. senão se root->code > key;

12.2.1.1. retorna recursivo o nó filho esquerda;

12.3. senão se root->code < key

12.3.1.1. retorna recursivo o nó filho direita;

12.4. senão se os nós filhos forem nulos

12.4.1. remove nó;

12.5. senao se root->direita == NULL

12.5.1. root=root->esquerda;

12.6. senão se root->esquerda ==NULL

12.6.1. root=root->direita;

12.7. senão

12.7.1. remove último nó esquerdo da direita

12.8. retorna root;

13. int search_menu();
    13.1. menu utilizado para descobrir qual elemento do nó deseja alterar;
    13.2. OBS: optei por não mudar o “code”, pois pensando em um banco de dados, se tirarmos a chave primária o dado fica inconsistente, então caso queira mudar um item deverá excluir o anterior e inserir um novo nó com os dados atualizados;

14. Marketplace *search_nodo (Marketplace *root, long int code);
    14.1. recebe o nó pai de todos e a chave desejada;
    14.2. O programa irá procurar de forma recursiva o nó desejado e retornar ele.

15. void *edit_element(Marketplace *root);
    15.1. utiliza a search_menu();
    15.2. Edita o elemento desejado;

16. void printer(Marketplace *root);
    16.1. Imprime de forma ordenada;

17. int size_tree(Marketplace *no);
18. retorna a altura da árvore (root->h).

19. void tree_startup(Marketplace *root);
    19.1. inicializa um ponteiro de controle para imprimir estrutura da árvore.

20. void tree(Marketplace *root, int depth, char *way, int direita);
    20.1. Imprime em estrutura de árvore horizontal.

21. void free_root(Marketplace *root);
    21.1. libera no;
22. void free_tree (Marketplace *root);
    22.1. libera nó raiz (pai de todos)

menu:

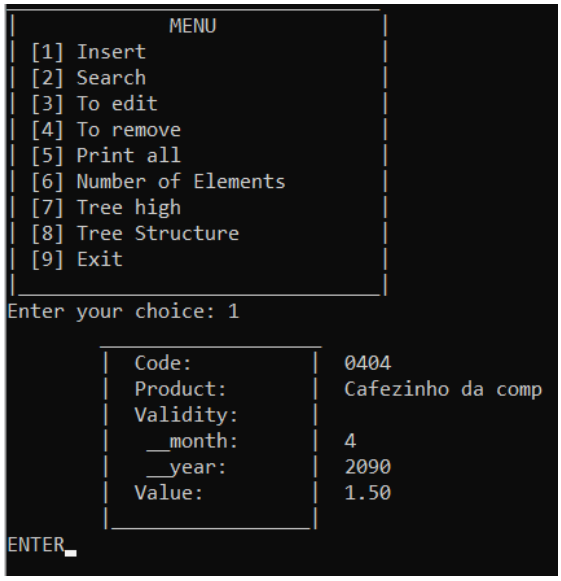
Pressione Enter para iniciar

O programa solicitará:

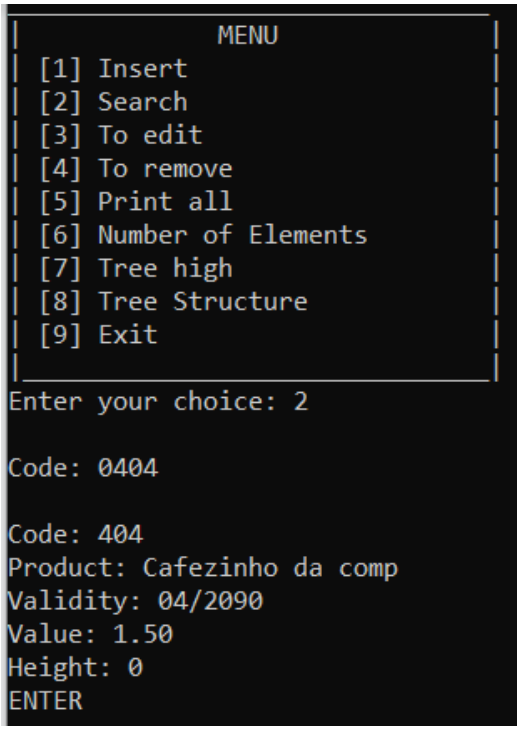
1. Insere nó;
2. Procura no e entrega os dados;
3. Edita um elemento;
    a. editar:
        i. Nome:
        ii. Validade
        iii. Valor
4. Remove elemento
5. Edita elemento
6. Número de elementos
7. Altura da árvore
8. Imprime em estrutura de árvore

Inserções de exemplo:

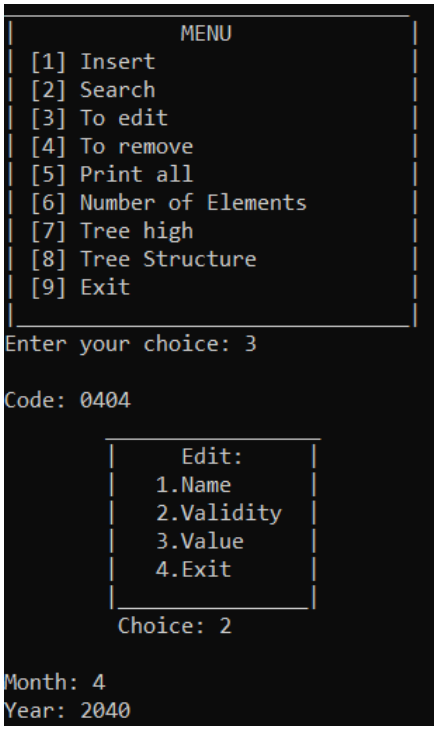
Insert: Todo elemento inserido pelo comando ”1” deverá preencher os campos da segunda tabela (code, product, validity, month, year, value):



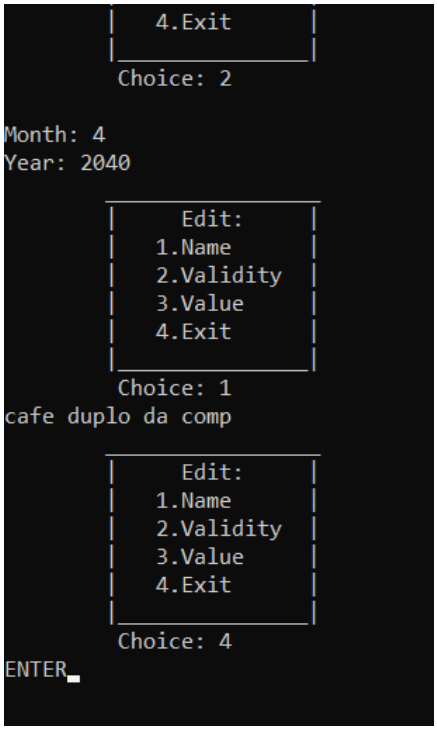
Search: solicita o código do produto que deseja buscar e imprime:



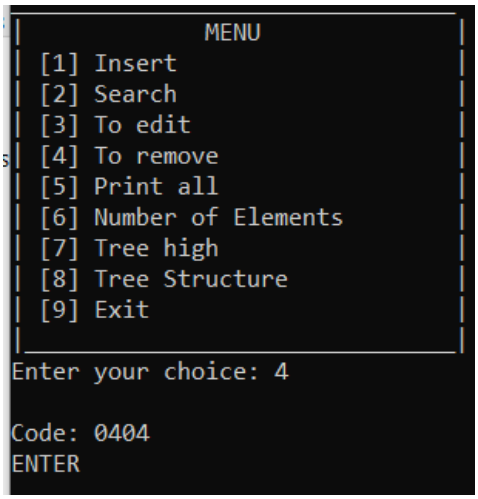
To edit: solicita o código do produto que deseja modificar e abre o menu que solicita o campo a ser modificado:



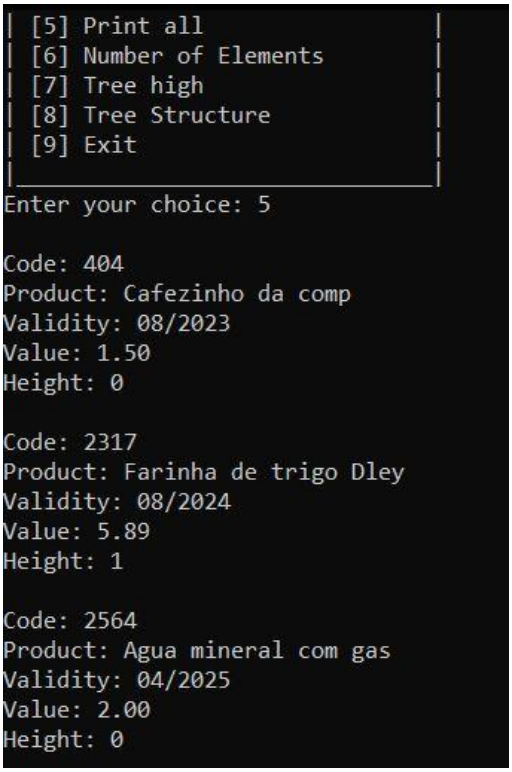
+

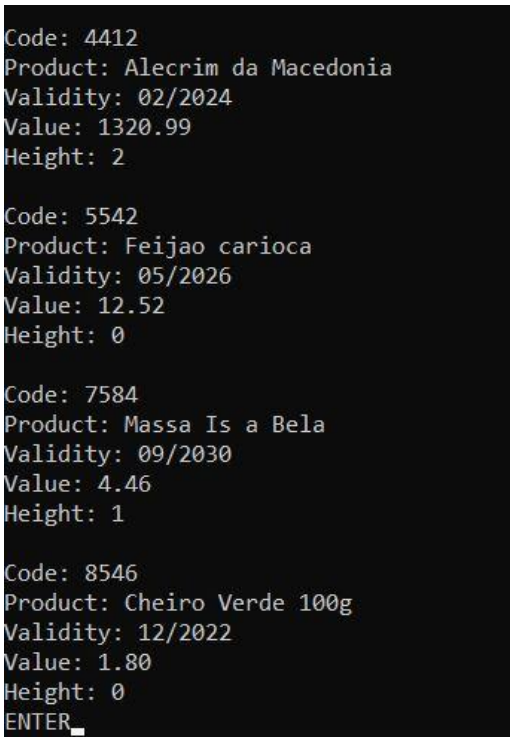


To remove: solicita o código do item e o remove:

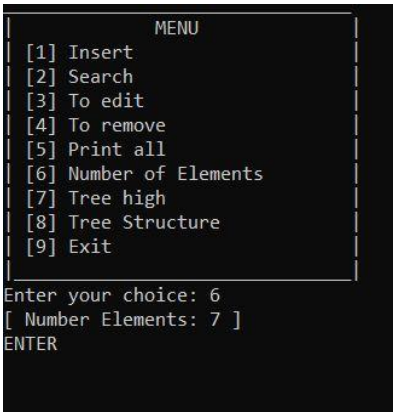


Print ALL: imprime em ordem crescente pelo número do código (code):

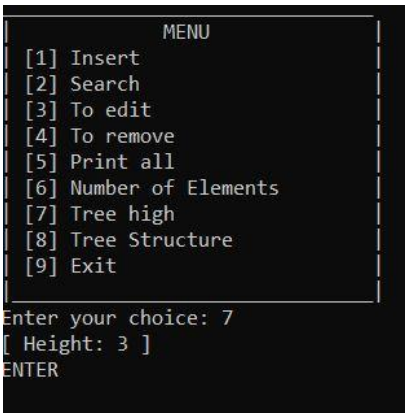




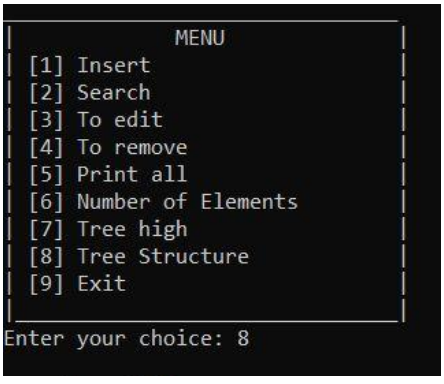
Number of elements: Quantidade de nós (quantidade de produtos) que a árvore possui.

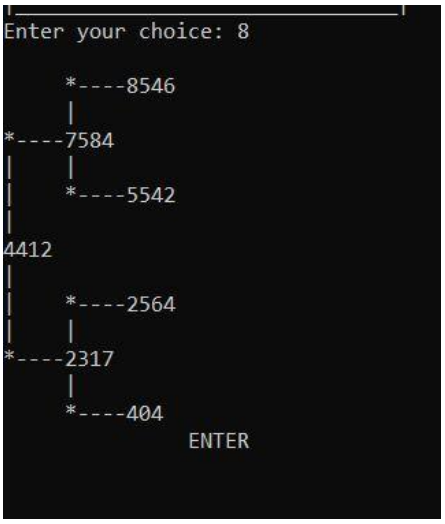


Tree high: altura da árvore:



Tree structure: imprime a na estrutura de árvore binária, com suas chaves:





Dificuldades:

As funções mais complexas para a implementação foram:

- A de balancear (balance) , demorou até entender bem como funciona a lógica de balancear durante a inserção, mais difícil que as funções das rotações.
- A função de imprimir em estrutura de árvore foi outra grande dificuldade, tentei de diversas formas e nunca funcionava, a forma que melhor funcionou foi apresentar uma árvore vertical equivalente.

CONCLUSÃO

Foi uma grande experiência a implementação desse trabalho, que trouxe um grande conhecimento da linguagem C,além de experiência no uso de ponteiros e alocação dinâmica, com o excesso de mudanças e correções no código, pude ver o quão fácil fica trabalhar nessa linguagem com o uso de ponteiros através das funções, a IDE utilizada para o desenvolvimento foi a CodeBlocks.

O tempo utilizado para a produção deste trabalho foram cerca de 32 horas em 4 dias, sendo no primeiro e segundo dia um tempo médio de 10 horas por dia, eu considero um bom tempo para programação pois o conteúdo todo do código continua fresco na memória e fica mais fácil de trabalhar, nos outros dias tive um custo maior de tempo em função de recapitulação de parâmetros de funções, seus funcionamentos, e afins.

Sugestões de trabalhos futuros:

Acredito que seria interessante pedir um documento solicitando ao aluno o mini mundo de seu projeto, como se o aluno fosse entregar para uma empresa específica o programa, a loja (por exemplo) descreveria ao programador quais dados ela precisa armazenar e o que ela precisa consultar, estas consultas seriam o menu e a árvore estaria implementada neste programa, se eu fizesse isso antes me daria mais clareza do que e o porquê eu estou criando o código, e com isso, por exemplo eu teria percebido antes da conclusão do código alguns campos adicionais que eu gostaria de implementar, como quantidade de produtos, quantidade vendida, entre outros detalhes, mas como diz o título, isso é uma sugestão, podendo ser utilizada inteiramente, parcialmente ou não

utilizada.