INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS SÃO JOÃO EVANGELISTA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ALGORITMOS E ESTRUTURAS DE DADOS II

ERICK SENA GODINHO
GUSTAVO BRENDON GOMES PIMENTA

TRABALHO PRÁTICO II

SÃO JOÃO EVANGELISTA
2023

ERICK SENA GODINHO
GUSTAVO BRENDON GOMES PIMENTA

Árvore Binária de Busca (BST)

SÃO JOÃO EVANGELISTA

2023

SUMÁRIO

1. INTRODUÇÃO

Para melhor funcionamento e praticidade, nas grandes e pequenas empresas, atualmente se é utilizado muitos sistemas e programas para melhor gerenciamento, lucratividade, velocidade e objetividade. Para que isso ocorra, por trás de tudo, foi utilizado a programação como meio para que tudo isso seja utilizado com excelência, também para facilitar e inovar o dia a dia de determinada empresa.

Ela (a programação), concede ao desenvolvedor a possibilidade de usufruir diversos meios para realizar esses processos. Neste caso, o TP, (Trabalho Prático), foi desenvolvido pensando a melhor maneira de utilizar as estruturas de algoritmos de dados para alcançar o resultado final.

1.1. Objetivo Geral

Este TP, (Trabalho Prático), tem como propósito compilar um sistema de RH, que é importante para gerenciar informações de funcionários de uma empresa, utilizando “Árvore Binária de Busca (BST)”.

A Árvore Binária de Busca é uma estrutura de dados em que cada nó da árvore possui um valor único e todos os valores à esquerda do nó são menores do que o valor do nó de valores à direita, que são maiores, sendo uma estrutura de dados que permite pesquisas eficientes em conjuntos de dados grandes e complexos, que começa pela raiz, depois os galhos e nós terminais. Primeiramente será feito uma interface com um menu de opções com as informações solicitadas, que no caso são: “Cadastrar funcionário”, “Buscar funcionário”, “Remover funcionário”, “Imprimir” e “Sair”. Cada seleção tem sua própria funcionalidade, como exemplo a “Buscar funcionário”, consulta os dados do funcionário pela “Árvore”, pelas informações cadastradas anteriormente, podendo utilizar para encontrar o cadastro tanto o seu “CPF” e o seu “Nome”.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Fixar conceitos sobre manipulação de árvores binárias;
- Realizar a implementação de árvore binária e suas TAD's;
- Compreender os métodos de inserção, busca e remoção em uma árvore.
- Estimular o raciocínio lógico para a resolução de problemas.
- Código de fácil compreensão;

1.3. Justificativa

É utilizado a Árvore Binária de Busca (BST), pois a sua eficiência depende da sua altura, que é o número máximo de nós entre a raiz e as folhas da árvore. Árvores com altura menor são mais eficientes, pois requerem menos operações para realizar uma busca, inserção ou remoção de um elemento.

Utilizando-se deste método, é feito um sistema objetivo, completo e de fácil entendimento, conseguindo alcançar todas metas e objetivos que foram designados para esse TP (Trabalho Prático).

2. DESENVOLVIMENTO

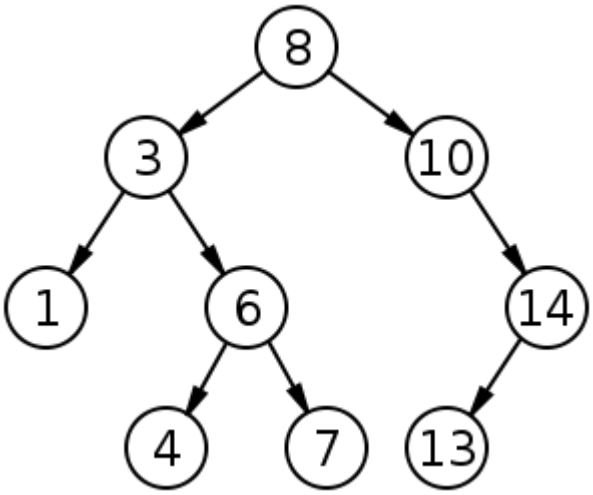
Como já dito anteriormente, para desenvolvimento desse TP (Trabalho Prático), foi se utilizado Árvore Binária de Busca (BST), mas o que é? Qual a sua finalidade? Melhor entendermos um pouco sobre o conceito de tal.

2.1. Árvore Binária de Busca (BST):

A Árvore Binária de Busca é uma estrutura de dados utilizada para armazenar um conjunto de valores hierarquicamente organizados em “nós”. Cada nó da árvore pode ter, no máximo, dois filhos: um filho esquerdo e um filho direito. Uma das propriedades fundamentais da Árvore Binária de Busca é a propriedade de busca, que estabelece que o valor de cada nó é maior do que todos os valores em sua sub-árvore esquerda e menor do que todos os valores em sua sub-árvore direita.

Essa propriedade permite que a Árvore Binária de Busca seja usada para pesquisas eficientes, inserção e remoção de elementos em um conjunto de dados organizado. Inserção e remoção de elementos em uma Árvore Binária de Busca são realizadas preservando a propriedade de busca e a ordem dos valores na árvore. Quando um novo valor é inserido, ele é comparado com o valor do nó atual e navegado para a sub-árvore esquerda ou direita, até que um nó vazio seja encontrado. Quando um valor é removido, a árvore é rearranjada de forma a manter a propriedade de busca.

Figura 1 – Árvore Binária de Busca (BST)



A Figura 1 apresenta como é a estrutura da árvore, como falado anteriormente, começando pela raiz, seus filhos divididos pela sub-árvore esquerda e pela sub-árvore da direita, até chegar a sua folha ou nó terminal.

2.2. Implementação

2.2.1 Funções

register_employee: Função que adiciona o funcionário tanto na árvore de CPF, quanto na árvore de Nome. Se o CPF ou o Nome não estiverem cadastrados. Caso tenha, não é cadastrado novamente.

search: Função que pesquisa o funcionário com base no CPF ou Nome.

remove_employee: Função que vai remover o funcionário com base no CPF ou Nome.

print_inorder: Função para imprimir os funcionários já cadastrados em ordem de CPF ou Nome.

print_preorder: Função para imprimir os funcionários já cadastrados em pré-ordem de CPF ou Nome.

print_postorder: Função para imprimir os funcionários já cadastrados em pós-ordem de CPF ou Nome.

3. CONCLUSÃO

Na parte final do TP (Trabalho Prático), é um resultado incrível, pois o projeto atende e supera todas as expectativas, na qual o projeto pode ser utilizado atualmente para aprendizado de qualquer usuário que queira ver na prática como funciona a Árvore Binária de Busca (BST). Concluindo o sistema, pode-se ter aumento e ganho de conhecimento, visto que novos métodos e meios podem ser utilizados em vários outros futuros sistemas, meios que são úteis e facilitam o código, deixando objetivo, limpo e uma facilidade maior de compreensão, deixando assim um grande aprendizado.

Também, fica-se realizado por conseguir desenvolver e aprender todas as matérias solicitadas pelo professor Eduardo Trindade. Uma vez que, se tem muitas dificuldades e com este TP (Trabalho Prático), é possível ter uma outra visão e grande avanço na matéria.

Por fim, se leva ganho também como programador, pelas experiências novas, métodos novos, conhecimentos e uma nova visão ao desenvolver um sistema em estrutura e análise de dados, consequentemente proveito de todo conhecimento para o que está porvir.

4. REFERÊNCIAS

AVA, Moodle IFMG SJE. Árvores, Eduardo Trindade, 2023. <https://ead.ifmg.edu.br/sje-presencial/pluginfile.php/158748/mod_resource/content/1/Aula%2012%20-%20%C3%81rvores.pdf>

AVA, Moodle IFMG SJE. Árvore Binária de Busca (BST), Eduardo Trindade, 2023. <https://ead.ifmg.edu.br/sje-presencial/pluginfile.php/158749/mod_resource/content/1/Aula%2013%20-%20%C3%81rvore%20Bin%C3%A1ria%20de%20Busca%20%28BST%29.pdf>

IME USP, Estrutura de Dados. Árvore Binária de Busca (BST), Sedgewick, 2019. <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-bst.html>

FAÇO PROGRAMA, Exercícios Árvore Binária, Tumblr, 2019. <https://facoprograma.tumblr.com/exe_arvore>

PAGINAS FEUP PT, Árvore Binária de Pesquisa, FEUP, 2018. <https://paginas.fe.up.pt/~rossetti/rrwiki/lib/exe/fetch.php?media=teaching:0910:aeda:aeda0910.14_arvores_3.pdf>

YOUTUBE, Mix Estrutura de Dados, ÁRVORE BINÁRIA de BUSCA, Programação Dinâmica, 2020. <https://www.youtube.com/watch?v=VmKkAQtnjsM&list=RDCMUC70mr11REaCqgKke7DPJoLg&start_radio=1&rv=VmKkAQtnjsM&t=0&ab_channel=Programa%C3%A7%C3%A3oDin%C3%A2mica>

REPOSITÓRIO GIT HUB <https://github.com/ErickSenaGodinho/TP2-Arvore-Binaria-de-Busca>

5. APÊNDICE A – (Funções)

Implementação, apêndice A (Funções), imagens que retrata o que foi relatado anteriormente na Implementação do TP (Trabalho Prático):

```
void register_employee(Tree *employee_tree_cpf, Tree *employee_tree_name)
{
    Employee employee = create_employee();
    Node *node_cpf = add_tree_cpf(employee_tree_cpf->root, employee);
    Node *node_name = add_tree_name(employee_tree_name->root, employee);
    if (node_cpf && node_name)
    {
        employee_tree_cpf->root = node_cpf;
        employee_tree_name->root = node_name;
        std::cout << "Funcionário adicionado com sucesso" << std::endl;
        Sleep(2000);
    }
    else
    {
        std::cout << "Funcionário já cadastrado" << std::endl;
        system("pause");
    }
}
```

```
void search(Tree *employee_tree_cpf, Tree *employee_tree_name)
{
    unsigned short option;
    do
    {
        show_search_menu();
        std::cin >> option;
        std::cin.ignore();
    } while (!option || option > 2);

    switch (option)
    {
    case 1:
    {
        std::cout << "Digite o nome:" << std::endl;
        char name[50];
        std::cin.getline(name, 50);
        search_employee_by_name(employee_tree_name->root, name);
    }
    break;

    case 2:
    {
        std::cout << "Digite o cpf:" << std::endl;
        char cpf[14];
        std::cin.getline(cpf, 50);
        search_employee_by_cpf(employee_tree_cpf->root, cpf);
    }
    }
}
```

```
void remove_employee(Tree *employee_tree_cpf, Tree *employee_tree_name, Employee *employee)
{
    if (!employee)
    {
        employee = new Employee();
        std::cout << "Digite o CPF do funcionário: " << std::endl;
        std::cin.getline(employee->cpf, 14);
        std::cout << "Digite o nome do funcionário: " << std::endl;
        std::cin.getline(employee->name, 50);
    }
    employee_tree_cpf->root = remove_tree_cpf(employee_tree_cpf->root, *employee);
    employee_tree_name->root = remove_tree_name(employee_tree_name->root, *employee);
    std::cout << "Removido com sucesso" << std::endl;
    system("pause");
}
```