Algoritmos e Estruturas de Dados
Trabalho 2 Professor: Tadeu Zubaran

.

1 Introdução

Este trabalho consiste em projetar e implementar uma *Árvore Binária de Pesquisa* (ABP) com alocação dinâmica de memória e com encadeamento també para os pais dos nodos.

2 Cliente

Os *Clientes* devem conter as informações.

- Chave
- Nome

O valor 0 é reservado para a chave do *Cliente* "dummy". Um *Cliente* dummy é usado para designar um *Cliente* que não existe.

Você pode assumir que a chave é um identificador único para o *Cliente* (como seria um CPF por exemplo).

3 Árvore

Neste trabalho vamos complementar a implementação de ABP vista em aula **incluindo um ponteiro em cada nodo para o seu pai**. Desta forma podemos fazer o percorrimento das folhas para a raiz de forma eficiente. Note que a raiz tem ponteiro para o pai definido com NULL (uma vez que ela não tem pai). Também implementaremos **rotações**, que são usadas em ABPs balanceadas.

As operações que esta ABP deve suportar são:

1. Criar uma ABP vazia. Assinatura: Abp() (pode ser idêntica a da aula)
2. Apagar uma *ABP*. Assinatura: ∼Abp() (pode ser idêntica a da aula)

3. Lista os clientes em pré-ordem. Assinatura: preOrdem() (pode ser idêntica a da aula)

4. Inserir um novo *Cliente*. Assinatura: void insere(Cliente cliente)

5. Retornar um **ponteiro para um Nodo** que tem o cliente com a chave dada. Assinatura: Nodo * buscaNodo(int chave)

6. Percorrer listando as chaves dos clientes do nodo apontado pelo ponteiro que é passado como parâmetro dele até a raiz (de baixo para cima). Assinatura: void imprimeAncestrais(Nodo * nodo)

7. Rotacionar a árvore para esquerda. Assinatura: void rotacionaEsquerda()

8. Rotacionar a árvore para direita. Assinatura: void rotacionaDireita()

A operação 5, caso a chave não esteja na árvore, deve retornar NULL.

No exemplo da Figura 1, se a operação 6 receber um ponteiro para o nodo que contém a chave 10, deve retornar $10, 11, 9, 5$

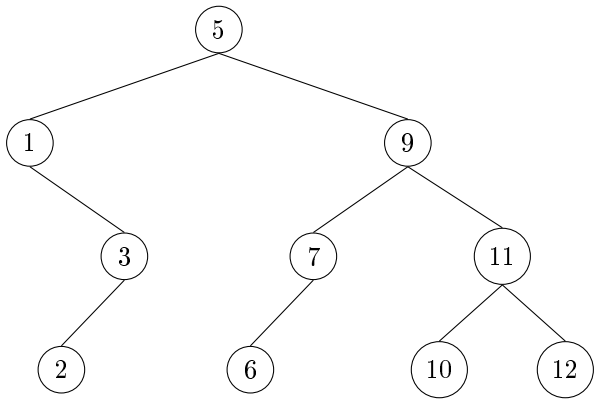
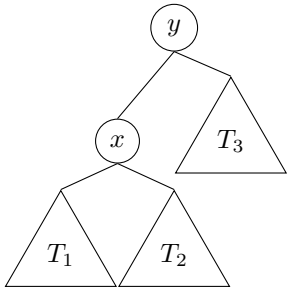


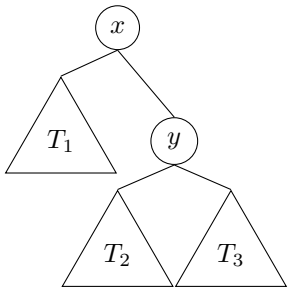
Figura 1: Árvore binária de pesquisa T_{abp}.

A rotações para à direita e esquerda são usadas na árvore de pesquisa binária balanceadas. É fortemente recomendado ao aluno pesquisar sobre a árvore AVL em livros ou na internet, pois não teremos tempo para implementá-la por completo na disciplina.

Na Figura 2 vemos como funcionam essas rotações. Na figura os triângulos T_1, T_2 e T_3 representam subárvores arbitrárias. Uma rotação para a direita de T_e gera T_d, e uma rotação para a esquerda de T_d gera T_e. Note que essas rotações (nesse trabalho) estão sempre sendo aplicadas à raiz da árvore. Note também que a rotação para a direita só está definida se o filho esquerdo da raiz



(a) Árvore binária de pesquisa T_e



(b) Árvore binária de pesquisa T_d

Figura 2: Uma rotação para a direita de T_e gera T_d, e uma rotação para a esquerda de T_d gera T_e

da árvore existe, e para a esquerda só está definida se o filho direito da raiz da árvore existe.

Você deve responder a seguinte pergunta, e argumentar sua resposta (no vídeo explicando o código): Uma rotação em uma ABP necessariamente gera uma ABP?

4 Avaliação

Não copie o código. Faça seu próprio código! Plágio receberá nota 0.

Critérios de avaliação:

- Clareza e corretude da explicação do código.
- Clareza e corretude do código.
- Resposta à pergunta sobre rotações.
- Eficiência do código.
- Código implementado com boas práticas de programação.

5 Entrega

A entrega consiste de duas coisas. Upload de um vídeo e upload do código no google class.

Use o seguinte código na main(), para fazer o teste de seu programa no vídeo (adapte aos nomes que você usou e a sua linguagem caso você use C).

```
#include <iostream>

#include "Aluno.h"
#include "Nodo.h"
#include "Arvore.h"

using namespace std;

int main()
{
    Aluno a1(1,"a1");
    Aluno a2(2,"a2");
    Aluno a3(3,"a3");
    Aluno a4(4,"a4");
    Aluno a5(5,"a5");
    Aluno a6(6,"a6");
    Aluno a7(7,"a7");
    Aluno a8(8,"a8");
    Aluno a9(9,"a9");
    Aluno a10(10,"a10");
    Aluno a11(11,"a11");
    Aluno a12(12,"a12");

    ABP abp;

    abp.insere(a8);
    abp.insere(a5);
    abp.insere(a3);
    abp.insere(a12);
    abp.insere(a9);
    abp.insere(a4);
    abp.insere(a11);
    abp.insere(a2);
    abp.insere(a6);
    abp.insere(a10);
    abp.insere(a1);
    abp.insere(a7);

    abp.preOrdem();
    abp.imprimeAncestrais(abp.buscaNodo(10));
    abp.imprimeAncestrais(abp.buscaNodo(1));
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    abp.rotacionaDir();
    abp.preOrdem();
    abp.imprimeAncestrais(abp.buscaNodo(10));
    abp.imprimeAncestrais(abp.buscaNodo(1));
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    abp.rotacionaEsq();
    abp.rotacionaEsq();
    abp.preOrdem();
    abp.imprimeAncestrais(abp.buscaNodo(10));
    abp.imprimeAncestrais(abp.buscaNodo(1));

    return 0;
}
```

O vídeo deverá poderá ser entregue pelo youtube (preferencial) ou por upload

no google class na tarefa junto com o código fonte. Caso você opte pelo upload no class **certifique-se que o vídeo funciona no programa VLC**. Outras maneiras de entregar o vídeo podem ser aceitas, mas o professor deve ser consultado com antecedência para estudar a viabilidade.

O vídeo deve **capturar sua tela e mostrar seu código e resultado de sua execução**. Este vídeo tem tempo mínimo de 5 minutos, recomendado de 10 e máximo de 20 minutos.

Seu vídeo deve conter a execução dos casos de teste onde você explica a saída apresentada no terminal. O tempo esperado desta etapa é de 1 minuto.

O resto do vídeo deve mostrar as partes mais importantes do código, onde o aluno explica em detalhes seu funcionamento. Em particular tenha certeza de mostrar:

- A estrutura de dados, suas variáveis e os detalhes seu funcionamento.
- Explicações detalhadas dos métodos do TAD e seu funcionamento
- Gerência de memória.

A escolha dos pedaços de código a serem explicados, a clareza e detalhes das explicações serão avaliadas.

Tanto o vídeo como o código devem ser enviados impreterivelmente até dia 12/11/2021 às 13:59.

TRABALHOS COM ATRASO, OU PLÁGIO RECEBEM NOTA 0