Árvore Binária (AB)

1. Considerar os tipos abstratos de dados definidos à seguir.

   Para o nó de uma árvore binária de números inteiros:

```
class No {
 private:
   No *esq;  // ponteiro para o filho à esquerda
   int info; // informação (valor) do nó (inteiro)
   No *dir;  // ponteiro para o filho à direita

 public:
   No()                  { };
   ~No()                 { };
   void setEsq(No *p)    { esq = p;     };
   void setInfo(int val) { info = val;  };
   void setDir(No *p)    { dir = p;     };
   No* getEsq()          { return esq;  };
   int getInfo{}         { return info; };
   No* getDir()          { return dir;  };
};
```

   Para a árvore binária (AB):

```
class ArvBin{
  private:
    No *Raiz;
  public:
    ArvBin();
    ~ArvBin();
};
```

   Cada operação a seguir deve ser declarada no bloco privado da classe ArvBin e o seu chamador, com os parâmetros necessários - sem o ponteiro para No - no bloco público.

   Desenvolver as seguintes operações no MI para o TAD ArvBin:

   (a) int ArvBin::impares(). Dada uma árvore Arv, calcular e retornar o número de nós que armazena valores ímpares fazendo um percurso pós-ordem.

   (b) int ArvBin::soma(). Dada uma árvore binária Arv, calcular e retornar a soma dos valores armazenados nos nós de Arv.

   (c) float ArvBin::media(). Dada uma árvore binária Arv, calcular e retornar a média dos valores armazenados nos nós de Arv.

   (d) float ArvBin::mediaPares(). Dada uma árvore binária a, calcular e retornar a média dos valores pares armazenados nos nós de a.

   (e) int ArvBin::maiorVal(). Dada uma árvore binária a, determinar e retornar o maior valor armazenado nos nós de a.

   (f) int ArvBin::menorVal(). Dada uma árvore binária a, determinar e retornar o menor valor armazenado nos nós de a.

(g) int ArvBin::maiores(int x). Dada uma árvore binária a, calcular e retornar a quantidade de nós que armazena chaves com valores maiores que x fazendo um percurso pós-ordem.

(h) float ArvBin::mediaNivel(int nivel). Dada uma árvore binária a, calcular e retornar a média dos valores armazenados num dado nível. Uma vez que o nó do nível é visitado não é necessário percorrer os níveis abaixo dele.

(i) void ArvBin::imprimirNivel(int nivel). Dada uma árvore binária a, imprimir o valor (campo info) de todos os nós que têm níveis menores ou iguais a um dado nível, ou seja, imprimir os valores de todos os nós de nível i tal que i $\leq$ nivel. Uma vez que o valor de um nível é impresso não é necessário percorrer os níveis abaixo dele. Fazer a operação imprimirNivel() realizando o percurso em:

- pré-ordem;
- ordem;
- pós-ordem.

(j) int ArvBin::Sucessor(int valor). Dada uma árvore binária a, determinar e retornar o valor do nó da AB a que representa o sucessor de valor. O sucessor de valor é o menor valor encontrado na AB a maior que valor. Se não existir sucessor, retornar o próprio valor.

(k) Nos exercícios acima, de que forma o tipo de percurso pode influenciar o resultado da operação sobre a árvore binária?

2. Implementar uma função (operação no MI) para determinar se uma árvore binária é ou não uma árvore cheia. Para uma árvore cheia tem-se: $n = 2^{h+1} - 1$, onde n é o número de nós e h a altura da AB T. Percorrer a AB apenas uma vez. Protótipo: bool ArvBin::eCheia()

3. Implementar uma função (operação no MI) para determinar se uma árvore binária é ou não uma árvore completa. Para saber se uma árvore é completa, basta desconsiderar o nível h e verificar se a árvore resultante é cheia. Onde h é a altura da árvore. Protótipo:

   bool ArvBin::eCompleta()

   Nas questões 4 a 7, considerar o seguinte TAD No usado para representar os nós de AB:

```
class No {
  private:
    No *esq;    // ponteiro para o filho a esquerda
    int altura; // armazena altura do nó da AB
    int info;   // informação do nó (int)
    No *dir;    // ponteiro para o filho a direita
  public:
    No()                        { };
    ~No()                       { };
    void setEsq(No *p)          { esq = p;       };
    void setInfo(int val)       { info = val;    };
    void setAltura(int alt)     { altura = alt;  };
    void setDir(No *d)          { dir = d;       };
    No* getEsq()                { return esq;    };
    int getInfo()               { return info;   };
    int getAltura()             { return altura; };
```

```
    No* getDir()                { return dir;    };
};
```

A esse novo TAD No é adicionado o atributo privado altura. Essa variável membro do TAD No armazena o altura em que se encontra o nó na AB. O novo TAD ArvBinAlt permanece com a mesma definição de ArvBin do exercício 1 (não foi definido aqui. Basta trocar os nomes no TAD ArvBin).

4. void ArvBinAlt::cria(int valRaiz, ArvBinAlt* sae, ArvBinAlt *sad). Dadas as árvores binárias Sae e Sad, cria uma nova AB com raiz em Raiz e com subárvore à esquerda Sae e à direita Sad. Não se esquecer de calcular a altura da nova raiz.

5. void ArvBinAlt::alturaNos(). Dada uma árvore binária a, calcular a altura de cada nó da AB a e armazená-la no campo Altura em cada nó. Essa operação será útil se a operação cria() acima não for usada.

Solution:

```
//a função altura(No *p, int alt) é a mesma da questão anterior
void ArvBinAlt::alturaNos(){
    altura(raiz, 0);
    return;
}
```

6. No* ArvBinAlt::noAlt(int alt). Dada a AB T, determinar e retornar um ponteiro para o primeiro nó cuja altura é igual à altura alt dada.

Solution:

```
No* ArvBinAlt::noAlt(int alt) {
    return auxNoAlt(raiz, alt);
}

No* ArvBinAlt::auxNoAlt(No* T, int Alt) {
    if(T == NULL) return T;
    else {
        if(T->getAltura() < alt &&
           T->getEsq() == NULL && T->getDir() == NULL) return NULL;
        else if(T->getAltura() == alt) return T;
        else {
                        No *p;
            p = auxNoAlt(T->getEsq(), alt);
            if(p != NULL && p->getAltura() == alt) return p;
            else {
```

```
                p = auxNoAlt(T->getDir(), alt);
                if(p != NULL && p->getAltura() == alt) return p;
                else return NULL;
            }
        }
    }
}
```

7. Implementar uma função (operação no MI) para determinar se uma árvore binária segue as restrições de uma árvore AVL. Uma AB T é AVL se T é binária de busca (ver exercício para verificar se uma AB é uma ABB) e para todo nó V de T a diferença, em módulo, da altura da subárvore a esquerda do nó V pela altura da subárvore a direita de V é no máximo 1. Essa função deve obedecer ao protótipo: bool ArvBinAlt::eAVL().

Solution:

```
int ArvBinAlt::auxeAVL(No *p) {
    if(p == NULL) return 0;
    else {
        int alturaesq = auxeAVL(p->getEsq());
        if(alturaesq == -1) return -1;
        int alturadir = auxeAVL(p->getDir());
        if(alturadir == -1) return -1;

        if(alturadir > alturaesq) {
            if(alturadir - alturaesq > 1) return -1;
            else return (alturadir + 1);
        }
        else if(alturaesq > alturadir) {
            if(alturaesq - alturadir > 1) return -1;
            else return (alturaesq + 1);
        }
        else return (alturaesq + 1);
    }
}
```

Árvores Binárias de Busca (ABB)

1. Implementar uma função (operação no MI) para verificar se uma árvore binária é de busca. Esta função deverá ter o protótipo bool ArvBin::eABB(). Uma AB T é de busca se para todo nó não folha V de T, a raiz da subárvore a esquerda, caso exista, tem valor menor que o de V e da subárvore a direita, caso exista, tem valor maior que o de V. Ou

```
(V -> getEsq()) -> getInfo() <= V -> getInfo() &&
(V -> getDir()) -> getInfo() >  V -> getInfo()
```

Essa condição deve valer para qualquer nó V de T desde que existam uma ou as duas subárvores.

2. Considerar que numa ABB cada nó armazena as seguintes informações sobre um aluno: nome, número de matrícula, nome da disciplina, turma e média. A chave da ABB é o número de matrícula do aluno.

   - Desenvolver os TADs para o nó e para a ABB que armazena os dados de vários alunos. Considerar, além do construtor e destrutor, as operações para: inserir, remover e imprimir (todos os dados de um aluno) dado seu número de matrícula.
   - Desenvolver uma operação para imprimir o nome do aluno que tem a maior média (não há notas com valores iguais).
   - Desenvolver uma operação para imprimir o nome do aluno que tem a menor média (não há notas com valores iguais).

3. Refazer o exercício (2), considerando como chave da ABB a média do aluno. Qual a complexidade para o melhor caso nos itens (ii) e (iii) dos exercícios (2) e (3).

4. Considerando ArvBinBusca, o TAD que implementa uma ABB de inteiros e No (ver exercício 1 sobre AB), o TAD dos nós da ArvBinBusca; desenvolver, usando a propriedade da ABB, as operações:

   (a) void ArvBinBusca::imprimeFilhos(int x). Dada uma árvore binária raiz e um inteiro x, imprimir os valores dos filhos, se existir, do nó que tem valor x.

   (b) void ArvBinBusca::imprimeIntervalo(int x, int y). Dada uma árvore binária raiz e dois números inteiros x e y, imprimir os valores dos nós que que estão no intervalo [x, y] e x ¡ y.

   (c) void ArvBinBusca::imprimeCrescente(). Dada uma árvore binária raiz, imprimir os valores dos nós da árvore em ordem crescente.

   (d) void ArvBinBusca::imprimeDecrescente(). Dada uma árvore binária raiz, imprimir os valores dos nós da árvore em ordem decrescente.

   (e) void ArvBinBusca::insereDoVet(int n, int *Vet). Dada uma árvore binária raiz e o vetor de inteiros Vet de tamanho n, inserir todos os valores de Vet na ABB raiz. Os valores do vetor Vet estão em ordem crescente. Desenvolver a operação de forma que:
      - a ABB torne-se degenerada (há mais de uma forma de fazer essa operação);
      - a ABB torne-se completa.

   (f) int* ArvBinBusca::insereVetCrescente(). Dada uma árvore binária raiz, preencher e retornar um vetor int *Vet com os valores armazenados na ABB Raiz. Considerar que há n nós na ABB. O vetor Vet deve ficar em ordem crescente.

   (g) int* ArvBinBusca::insereNoVetDecrescente(). Dada uma árvore binária raiz, preencher e retornar um vetor int *Vet com os valores armazenados na ABB Raiz. Considerar que há n nós na ABB. O vetor Vet deve ficar em ordem decrescente.

   (h) No* ArvBinBusca::buscaValor(int Chave). Dada uma árvore binária e um inteiro Chave, desenvolver a operação buscaValor não recursiva para achar e retornar o ponteiro para o nó da ABB cujo valor é Chave. Retornar NULL se Chave não for encontrada.

   (i) int ArvBinBusca::classificaChave(int x). Dada uma árvore binária e um inteiro x. Seja Nox, o ponteiro para o nó que tem valor x; retornar 2 se Nox tem 2 filhos, retornar 1 se Nox tem 1 filho e retornar 0 se Nox tem 0 filhos (folha).

(j) void ArvBinBusca::insereValor(int Val). Inserir um novo nó na ABB com valor inteiro Val. A operação InsereValor deve ser não recursiva.

(k) int ArvBinBusca::nos1Filho(). Operação para calcular e retornar o número de nós com um único filho.

(l) int ArvBinBusca::nos2Filho(). Operação para calcular e retornar o número de nós com 2 filhos.

(m) bool ArvBinBusca::estritamenteBin(). Operação para determinar se uma árvore binária é (true) ou não (false) uma ABB estritamente binária.

(n) void ArvBinBusca::transfABemABB(). Transformar a AB enraizada em Raiz em uma ABB completa. Retornar a ABB construída.