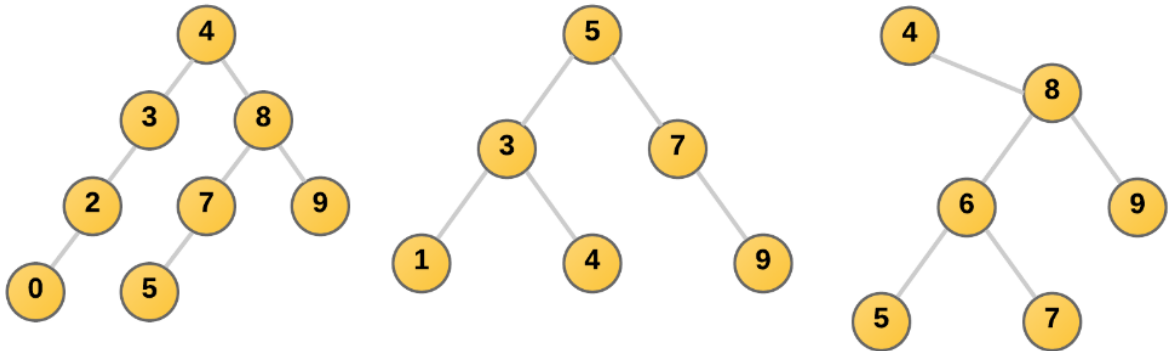


Lista de Exercícios – IBILCE UNESP

Estrutura de Dados I – Árvores binárias

1) Classifique as árvores binárias a seguir quanto a:

- a) Nós folhas;
- b) Nós internos;
- c) Nós pai com exatamente dois filhos.
- d) Altura;
- e) Grau;
- f) Nós de cada nível;
- g) Número de nós da árvore;



2) Responda às seguintes perguntas, justificando a resposta em cada caso:

- a) Qual a altura máxima e mínima de uma árvore binária com 31 nós?
- b) Qual é o número máximo de nós que podem ser encontrados nos níveis $h = 3, 4$ e 10 de uma AB?

3) A estrutura abaixo apresenta a definição de um tipo nó (de árvore binária - AB):

```
//-----
typedef struct no {
    char info;
    struct no *esq;
    struct no *dir;
} no;
//-----
```

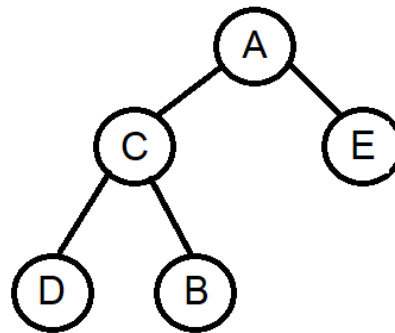
Com base na ED acima, implemente:

- a) Uma função que verifique se uma árvore binária está vazia.
Protótipo: `int vazia (no *raiz).`
- b) Uma função que faça a alocação dinâmica na memória de um nó de uma AB.
Protótipo: `no *cria_no (char c, no *esq, no *dir)`
- c) Uma função que libera na memória um nó da ABB.
Protótipo: `no *libera_no (no *t)`
- d) Uma função que insere um nó à esquerda.
Protótipo: `int insere_esq (no *pai, char elemento).`
- e) Uma função que percorre toda a árvore (sem repetir nós - percurso), imprimindo seus valores (campo `info`).
Protótipo: `void imprime_arvore (no *raiz).` Aqui, utilize a estratégia/percurso de **sua** preferência.

- 4) A função a seguir descreve um algoritmo recursivo para calcular a altura de uma árvore binária:

```
//-----  
int altura (no *t){  
    if (Vazia(t)) {    //Se nó t é vazio/nulo.  
        return 0;  
    }  
    int altE = altura (t->esq);  
    int altD = altura (t->dir);  
    if (altE > altD) {  
        return (altE + 1);  
    }  
    return (altD + 1);  
}  
//-----
```

Ilustre as etapas da pilha de execução (empilha, a cada chamada da função; e desempilha, quando a chamada é totalmente encerrada) após aplicar a função “altura” para os itens a), b) e c), considerando a seguinte AB:



- a) altura (t), t = A.
- b) altura (t), t = C.
- c) altura (t), t = E.

- 5) Determine o nó raiz de cada uma das AB a seguir:

- a) Árvore com percurso pós-ordem: 6-3-2-4-7.
- b) Árvore com percurso pré-ordem: 9-2-3-4-6-5-14.
- c) Árvore com percurso em ordem (assuma que a árvore é completa): 3-2-9-4-6-7-5.

6) Escreva as seguintes funções para manipular árvores binárias:

- a) Função que retorna o maior elemento da árvore (dado um campo chave inteiro).
- b) Função que retorne o total de nós folhas de uma AB.

- 7) Escreva uma função que verifique se uma árvore binária é **cheia**. Definição: uma AB é dita cheia se todos os seus nós têm exatamente 0 ou 2 filhos (estritamente binária), e todas as folhas estão no mesmo nível.

8) Considere a seguinte função, que calcula o percurso pós-ordem em uma AB:

```
//-----  
void posOrdem (no *t) {  
    if (t != NULL) {  
        posOrdem (t->esq);  
        posOrdem (t->dir);  
        printf ("Dado impresso: %c\n", t->info);  
    }  
}  
//-----
```

Escreva uma função não-recursiva para o percurso pós-ordem acima. Dica: utilizar pilha.

9) Escreva uma função que realize o percurso em largura (BFS) de uma AB. Dica: utilizar fila.

10) A partir do conjunto $C = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, construa duas propostas de árvores binárias de busca (ABB).

11) Implementa uma função para realizar buscas em uma ABB de números inteiros. Protótipo: `no *busca (no *raiz, int elemento)`.

12) Implemente uma função que realiza a Busca + inserção de um número inteiro na ABB. Protótipo: `no *busca_insere (no *raiz, int elemento)`.

13) Implemente uma função que encontra o menor elemento de uma ABB.

14) Escreva uma função que, dada uma ABB qualquer composta de inteiros, imprima todos os elementos menores que N.
Protótipo: `int *menores_valores (no *raiz, int N)`.

15) Sabe-se que uma ABB balanceada/perfeitamente balanceada possuem as estruturas ideais para buscas. No entanto, inserções e remoções podem desbalancear a árvore, tornando as buscas ineficientes. A fim de contornar esse problema, implemente o seguinte algoritmo de rebalanceamento:

A partir de um vetor ordenado de valores distintos, realizar os seguintes passos:

1. O registro do meio é inserido na ABB vazia (como raiz).
2. Tome a metade esquerda do *array*, e então repita o passo 1) para a sub-árvore esquerda.
3. Aplique a mesma ideia para a metade direita e sub-árvore direita.
4. Repita o processo até não poder dividir mais.

16) Insira os seguintes valores em uma ABB inicialmente vazia:

- a) 10, 20, 5, 2, 7, 25, 30, 22, 18
- b) 15, 10, 20, 8, 12, 17, 25, 16, 19, 30
- c) 50, 30, 70, 20, 40, 60, 80, 10, 25, 35, 45

- 17)** Dado a árvore a seguinte, executar o processo de renovação (cumulativo) dos seguintes nós: 100 – 150 – 80 – 270 - 400. Adote algum dos critérios de exclusão vistos em aula quando o nó tiver dois filhos.

