NP10 – Simulação da Prova 2

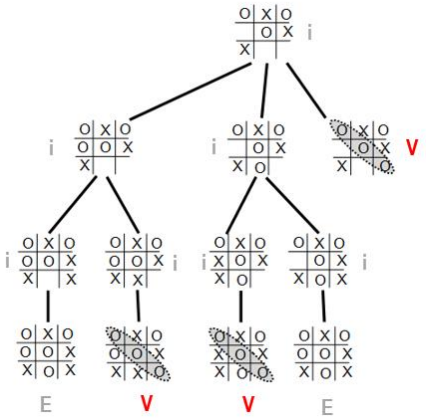
Atividade avaliativa para Nota de Participação
Algoritmos e Estruturas de Dados 1 (1001502)

Orientações Gerais

- **Tempo para elaboração: 2h**.

Orientações Quanto a Notação, Nomes das Variáveis, e Estruturas

- Use os **mesmos nomes fornecidos no enunciado** (L, F, X, etc.). Utilize variáveis auxiliares temporárias, o tanto quanto for necessário. É só declarar e usar. Mas **não considere a existência de nenhuma outra variável permanente**, além das definidas no enunciado. Não considere prontas para uso nenhuma operação, salvo se explicitamente indicado no enunciado da questão.
- Considere as **estruturas exatamente conforme definido no enunciado**, seja no texto da questão, seja nos diagramas.
- Para o desenvolvimento de algoritmos, use preferencialmente a notação adotada nas aulas: **p = NewNode; Deletenode(P), P->Info e P->Next,** sendo P uma variável do tipo **NodePtr** (ponteiro para nó). Quando a estrutura for duplamente encadeada, ao invés de P->Next considere que a notação contenha **P->Dir e P->Esq.** Também é possível implementar em C ou C++.



Tema 1 – Árvores em Geral

Questão 1 (4 pontos) Considere uma **Árvore de Previsão de Jogadas** na qual cada nó possui a situação do tabuleiro em uma possível jogada, o resultado do jogo para aquela jogada, e um ponteiro para cada um dos filhos daquele nó. O resultado para uma jogada pode ser 'V' (Vitória), 'D' (Derrota), 'E' (Empate) ou '?' (indefinido). A árvore é implementada com alocação encadeada e dinâmica de memória, conforme a declaração abaixo. A raiz (R) e os filhos (Sons) são do tipo ponteiro para nó (nodeptr). Implemente uma função que conta e retorna quantos resultados do tipo 'V' (Vitória) existem em uma árvore R.

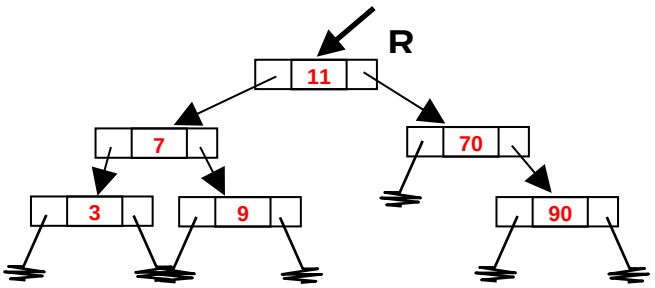
```
const int max_sons = 9; // número máximo de filhos

struct _node {
        Tabuleiro T; // situação do tabuleiro em uma possível jogada
        Char Resultado; // 'V', 'E', 'D', ou '?'
        struct _node *Sons[max_sons]; // ponteiros para os filhos do nó
        } ;

typedef struct _node node;
typedef struct _node* nodeptr;

nodeptr R = nullptr; // raiz da árvore

Int NumeroDeVitorias (nodeptr R) {
// Conta e retorna o número de jogadas com resultado 'V' na Árvore de Previsão de Jogadas R.

} // NumeroDeVitorias
```

Tema 2 – Árvores Binárias de Busca

Questão 2 (4 pontos) Considere uma ***Árvore Binária de Busca*** (ABB), de raiz R, implementada com alocação encadeada e dinâmica de memória, conforme os diagramas abaixo. A Árvore não contém elementos repetidos. Implemente a operação:

void Remove (variável por referência R do tipo ABB, Variável X do tipo int, variável por referência Ok do tipo bool);

/* esta função deve procurar X na ABB R e, caso encontrar, deve remover da árvore e retornar Ok = true. Caso não encontrar, Ok deve retornar false. O tipo ABB é análogo ao tipo NodePtr, ou seja, ponteiro para o nó da árvore */



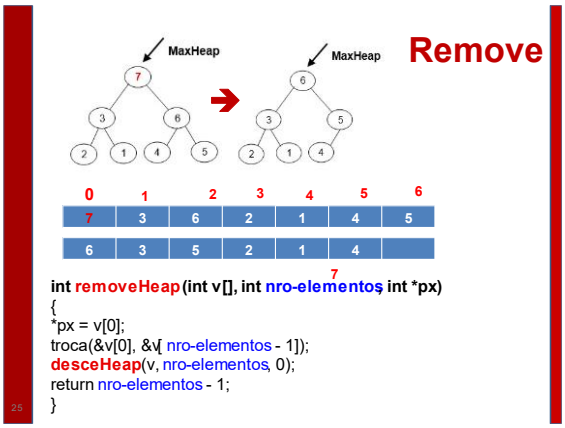
árvore vazia

Tema 3 – Alternativa Heap

Questão 3 – Alternativa Heap – Exemplo (2 pontos) Em um ***Heap-Binário-de-Máximo***, o elemento que está em um determinado ***nó*** de uma árvore binária tem valor maior ou igual do que o valor de seus ***filhos*** direito e esquerdo. Entre os nós ***irmãos***, não há necessariamente uma ordenação, como mostra o diagrama a seguir. Um ***Heap-Binário-de-Máximo*** pode ser implementado em um vetor, como mostra o diagrama. Nessa implementação, temos que:

- O ***pai*** de um elemento armazenado na posição i encontra-se armazenado na posição (i-1)/2;
- O ***filho-esquerdo*** de um elemento armazenado na posição i encontra-se armazenado na posição 2*i+1;
- O **filho-direito** de um elemento armazenado na posição i encontra-se armazenado na posição 2*i+2.

Na operação de **remoção de um elemento de um Heap-Binário-de-Máximo**, retiramos o elemento que está na raiz, colocamos na raiz o valor do último elemento do vetor, e então diminuímos o tamanho do vetor (veja algoritmo no diagrama). Precisamos então corrigir o Heap até que este recomponha sua propriedade (cada nó com informação maior do que a de seus filhos). Para restaurar a propriedade do heap, utilizamos o procedimento "desceheap", no qual trocamos o valor de um nó com o valor do maior elemento dentre seus filhos, até que a propriedade esteja totalmente restaurada e/ou até o final do vetor.



Implemente a operação:

void **desceHeap**(int v[], int nro-elementos, int pos)
// corrige o heap, descendo o elemento em v[pos] até restaurar a propriedade de heap

}

Tema 3 – Alternativa Análise de Algoritmos / Ordenação / Busca / Tópicos Complementares

Questão 3 – Alternativa Análise de Algoritmos – Exemplo (2 pontos) Na ordenação por inserção, consideramos um trecho do vetor já ordenado e percorremos o restante do vetor, pegando um elemento e inserindo este elemento na posição correta, na porção já ordenada do vetor. Fazemos assim com um a um dos elementos do vetor, e este estará ordenado (veja a ilustração). (a) Desenvolva um procedimento para ordenar um vetor pelo método de inserção. (b) Apresente a análise de complexidade de tempo do algoritmo desenvolvido, para o melhor caso e para o pior caso - utilize a Notação O (análise assintótica). (c) Apresente a análise da correção do algoritmo, usando invariantes.

void **InsertionSort** (int V[], int N)
// ordena o vetor V de tamanho N, por inserção: percorre o vetor e insere cada elemento na posição correta de um vetor ordenado.

Ordenação por Inserção

Percorre o vetor e **insere** o elemento na posição correta.