DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Aula 7 – Árvores

Professores: Luiz Chaimowicz e Raquel Prates

Conceitos básicos

- Organização NÃO LINEAR de dados
  - Diferentemente de vetores, listas, pilhas e filas
- Organiza um conjunto em uma estrutura hierárquica (muito utilizada em ciência da computação).
- Cada elemento é chamado de nó
- Relação: pai – filhos
  - Raiz: primeiro da hierarquia (“pai de todos”)
  - Folhas: nós que não têm filhos
  - Nós internos: nós que não são raiz e tem pelo menos um filho
- Recursividade: o filho de um nó é a raiz de uma subárvore

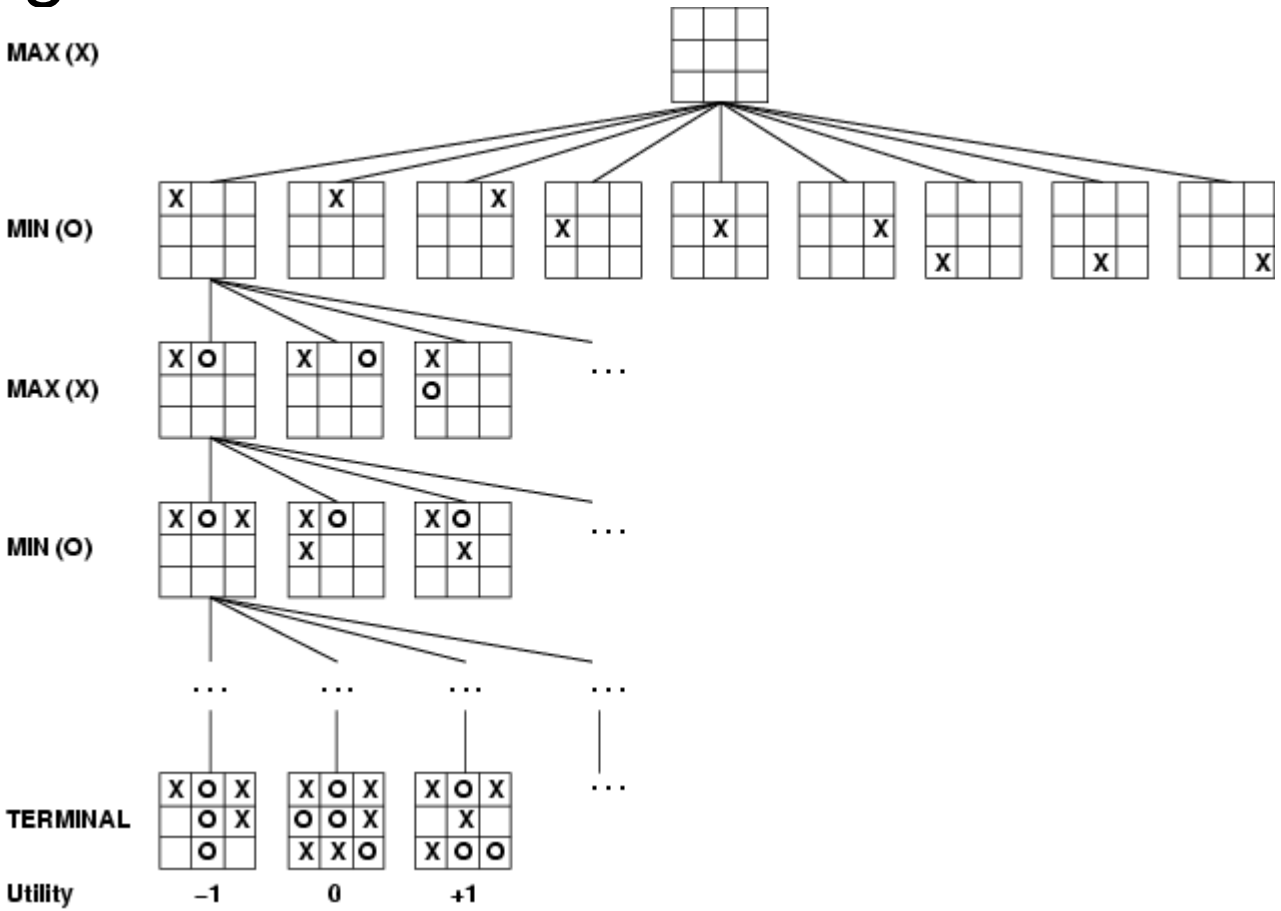
Definição

- Um único nó é uma árvore. Este nó é raiz da árvore.
- Suponha que n é um nó e T1, T2, ..., Tk sejam árvores com raízes n1, n2, ... , nk, respectivamente. Podemos construir uma nova árvore tornando n a raiz e T1, T2, ..., Tk sejam subárvores da raiz. Nós n1, n2, ..., nk são chamados filhos do nó n.

(Aho, Hopcroft e Ullman - 1983)

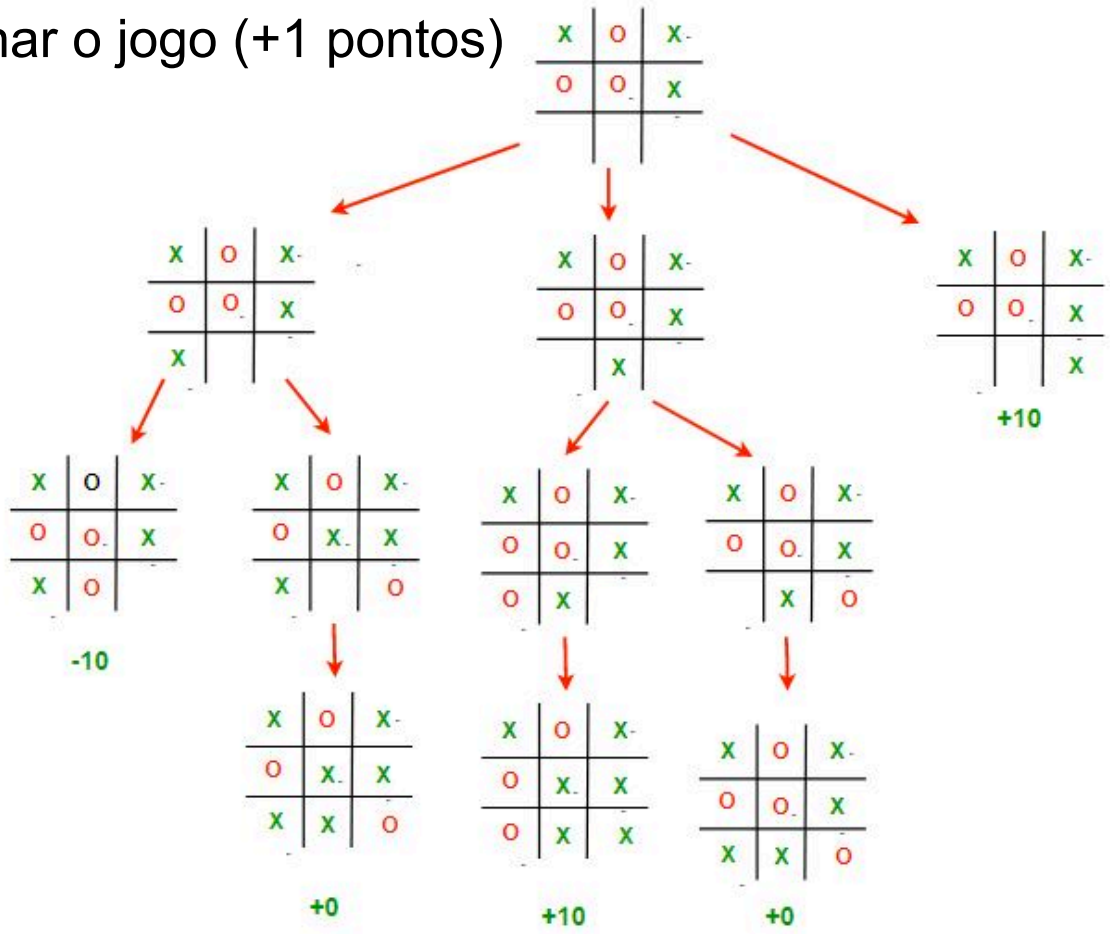
Exemplo

- IA, Jogos: Minimax



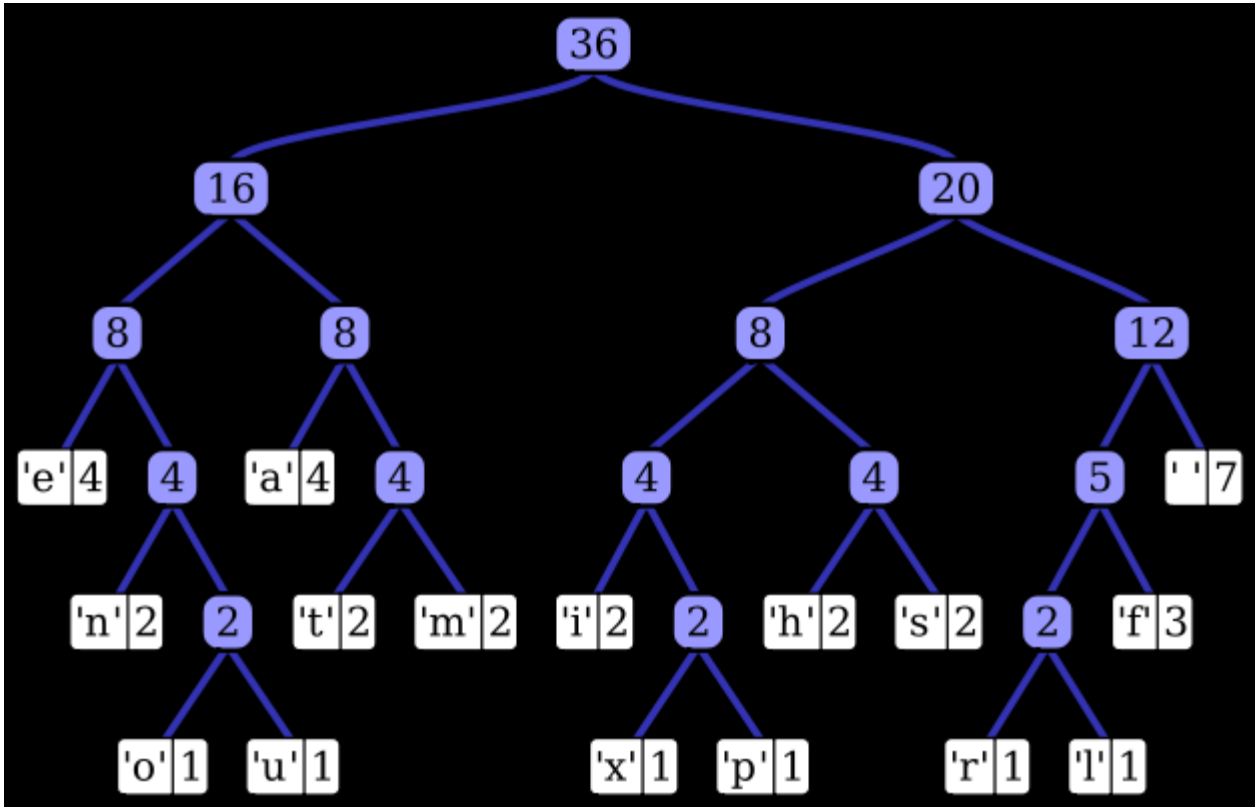
Algoritmo MiniMax

- X tem 3 opcoes:
  - [2,0]: 0 pode entao jogar [2,1] e ganhar (-1 pontos)
  - [2,1]: 0 pode entao jogar [2,2] e empatar (0 pontos)
  - [2,2]: e ganhar o jogo (+1 pontos)



Exemplo

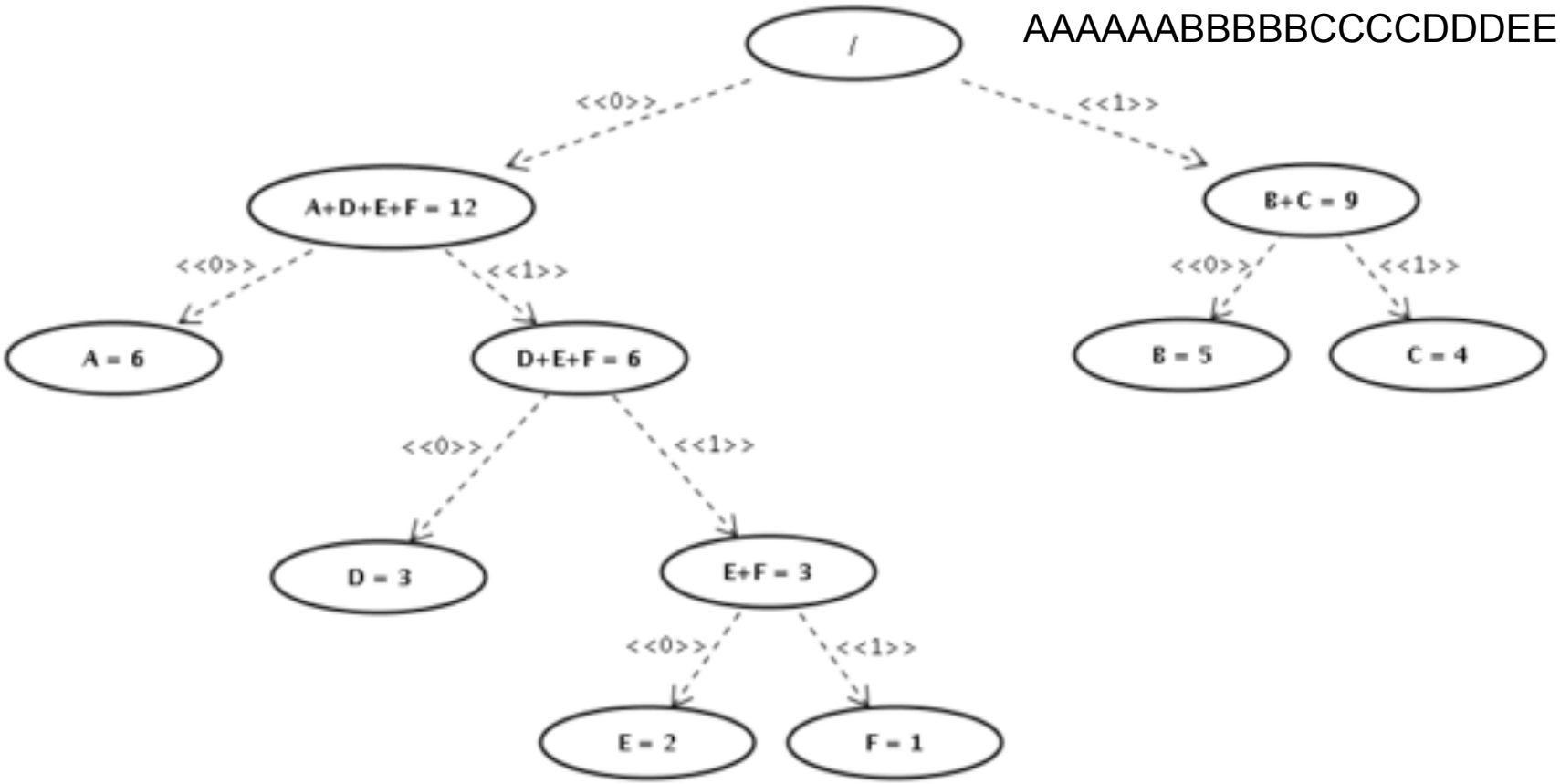
- Compressão de dados: Árvore de Huffman



Exemplos

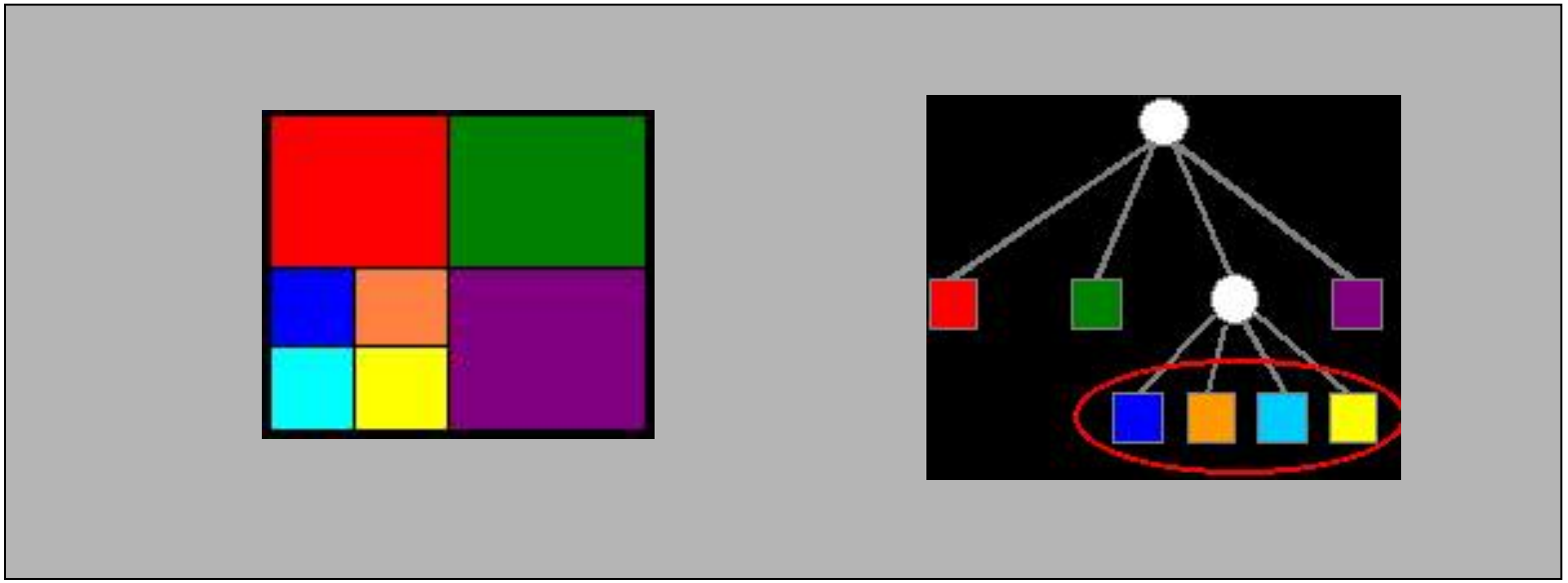
- Compressão de dados: Árvore de Huffman

String:
AAAAAABBBBBCCCCDDDEEF



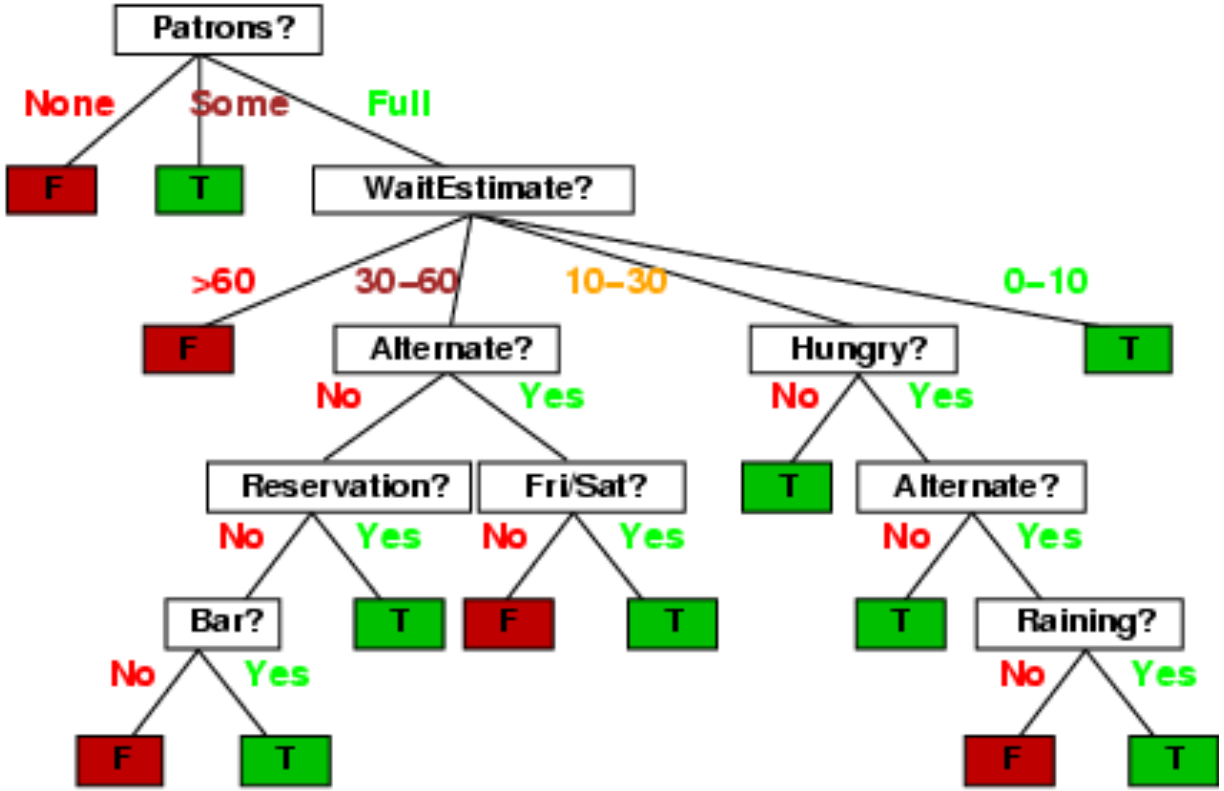
Exemplos

- PDI, CG, Visão: quadtree



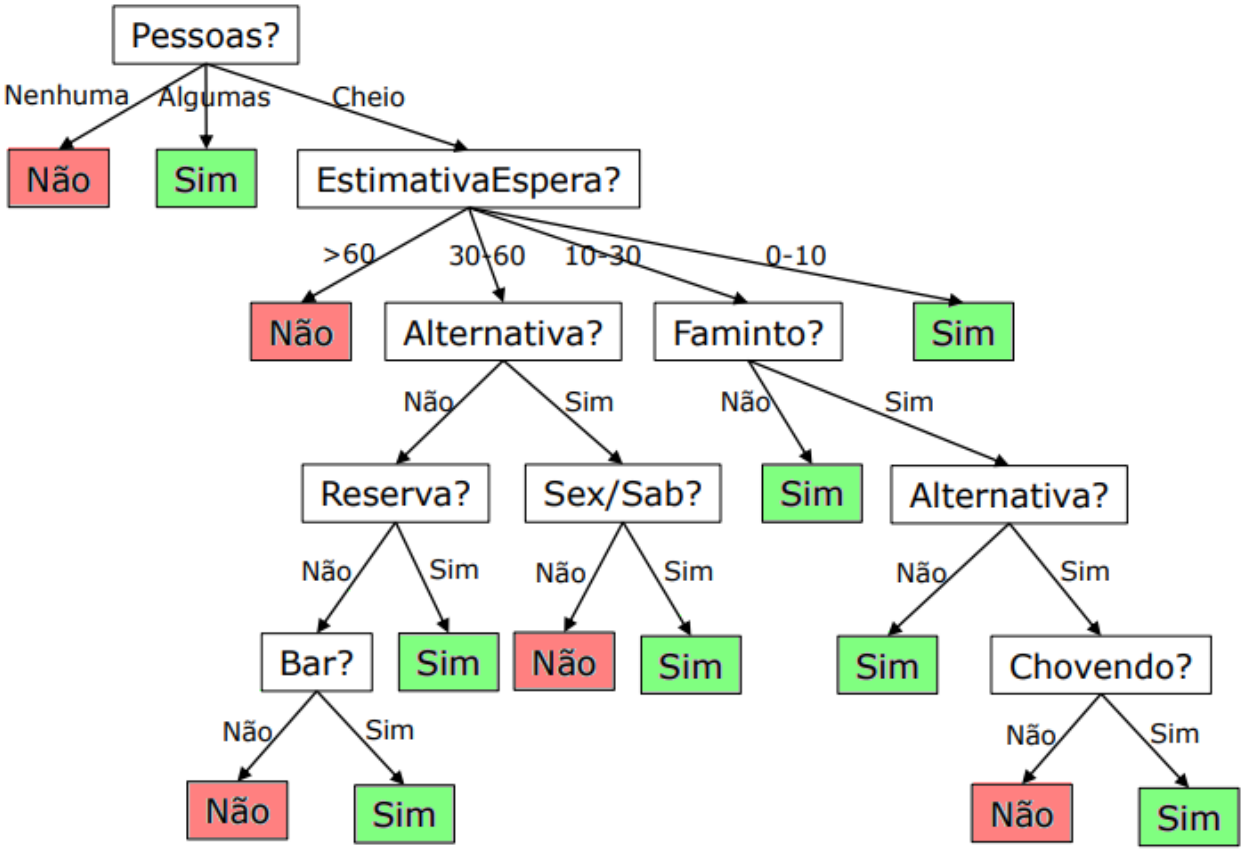
Exemplos

- IA: Árvores de Decisão



Exemplo

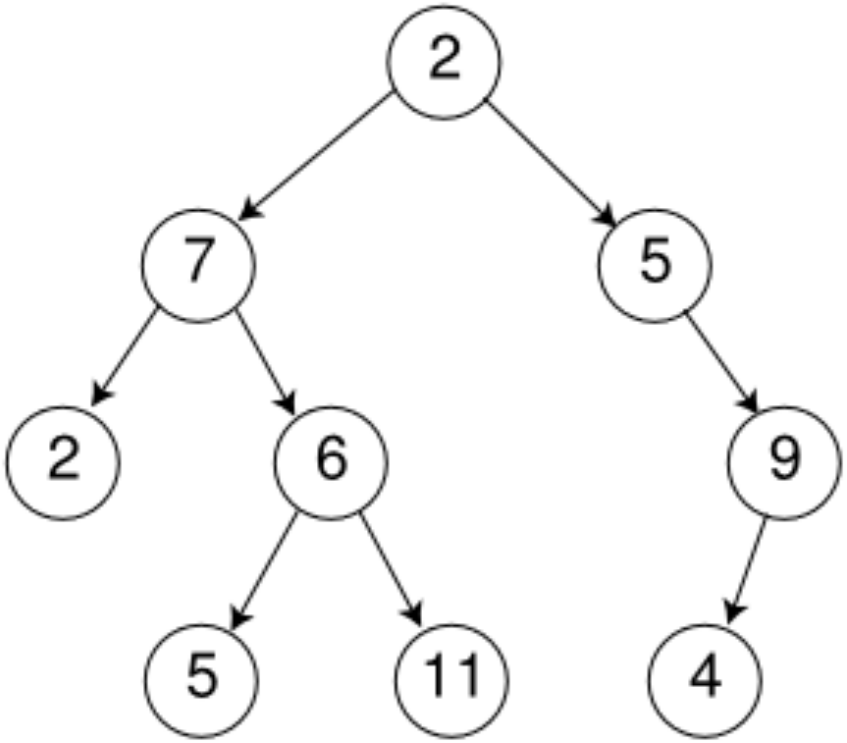
- IA: Árvore de Decisão



Exemplo

- Árvore Binária
  - Cada nó tem no máximo dois filhos

- Obs: a árvore ao lado não impõe nenhuma ordenação em seus nós



Ordem dos Filhos

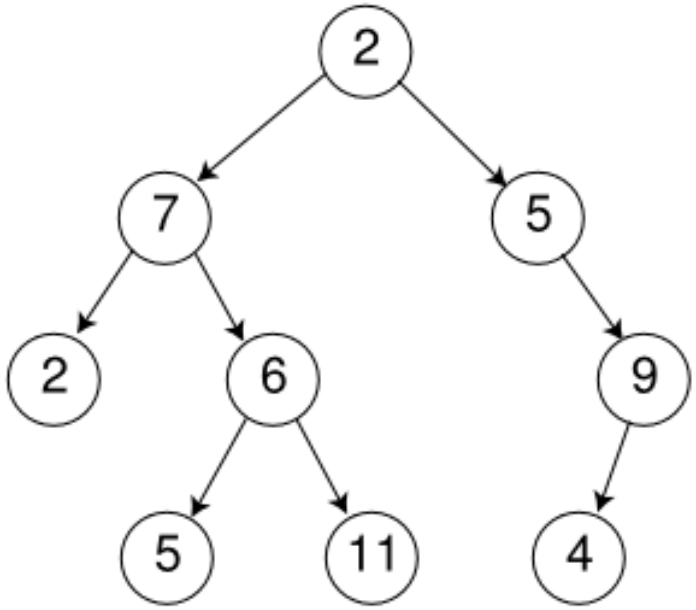
- A ordem dos filhos dos nós em uma árvore pode ser ou não significativa.
  - Exemplos, no heap, a ordem dos filhos não tem significado
  - Outros casos, pode se ter um significado (como veremos em pesquisa em árvores binárias)
- Considera-se que se a e b são nós irmãos, e a está à esquerda de b, então todos seus descendentes estão à esquerda de b e todos seus descendentes.

Conceitos sobre a estrutura

- O nível do nó raiz é 0.
- Se um nó está no nível *i* então a raiz de suas subárvores estão no nível *i* + *1*.
- Caminho: ligação entre quaisquer dois nós
  - Em uma árvore, só existe um caminho entre quaisquer dois nós: grafo sem ciclos!
- Ramo: caminho que termina numa folha
- Altura de um nó: comprimento do caminho mais longo deste nó até um nó folha.

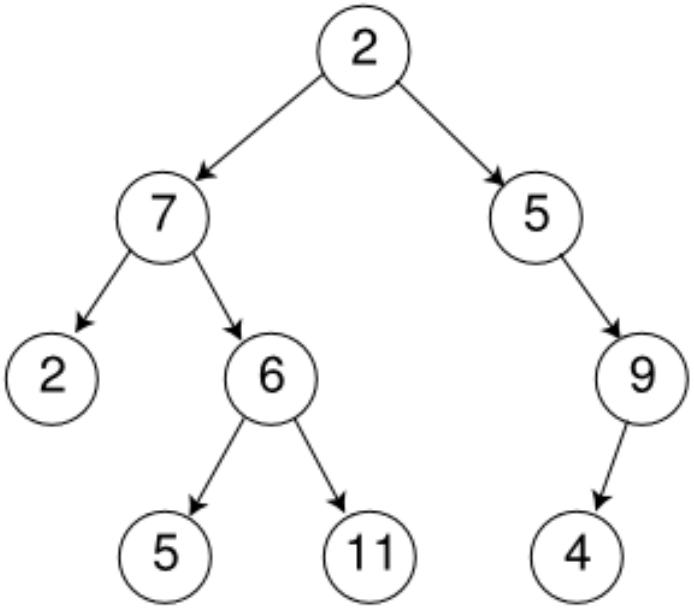
Caminho

- Um caminho entre dois nós n_i a n_k, onde n_i é antecedente a n_k, é a sequência de nós para se chegar de n_i a n_k.
- Se n_i é antecedente a n_k, n_k é descendente de n_i
- O comprimento do caminho é o número de nós do caminho – 1.



Altura / Profundidade

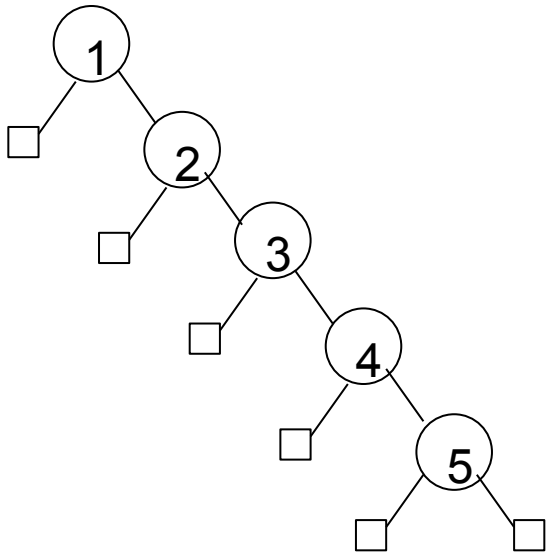
- A **profundidade** de um nó é o comprimento do caminho entre a raiz e aquele nó
- A **altura** de um nó é o número de arestas no caminho mais longo desse nó até uma folha
- **Altura da árvore** é igual a altura da raiz que é igual à sua maior profundidade



Profundidade do nó 7: **1**
Altura do nó 7: **2**
Altura da Árvore: **3**

Altura da Árvore

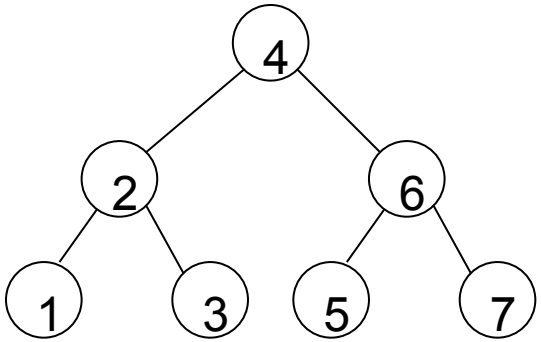
- Em que situação a árvore atinge sua altura máxima?
  - *Quando cada nó da árvore só tem um filho.*



- Qual a altura máxima de uma árvore binária?
  - *Altura máxima é n-1.*

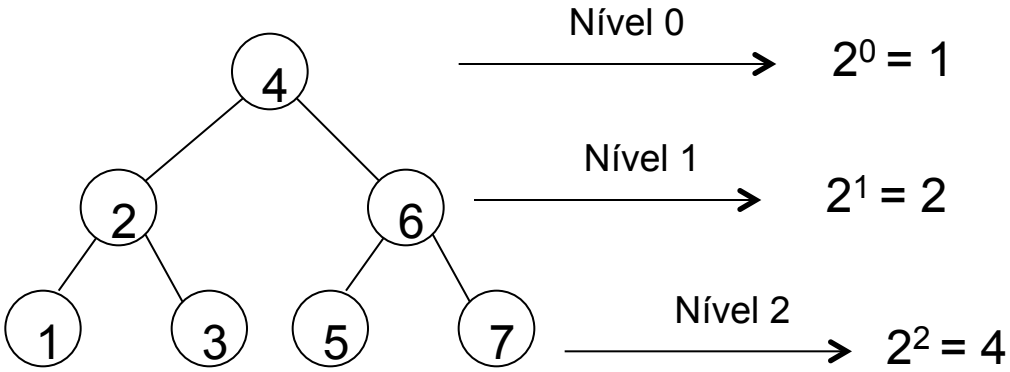
Altura da Árvore

- Em que situação a árvore atinge sua altura mínima?
  - *Quando a árvore está balanceada.*



Altura da Árvore

- Qual a altura mínima de uma árvore binária?



$$n = 2^0 + 2^1 + 2^2 + \ldots + 2^k$$

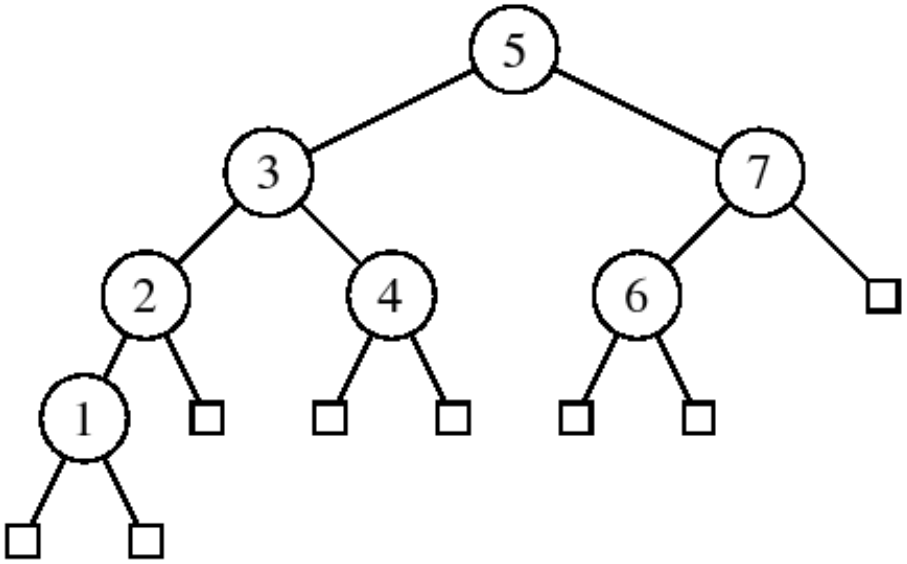
$$n = \sum 2^i \rightarrow n = 2^{k+1} - 1 \rightarrow n + \sum_{i=0}^{k} 1 = 2^{k+1} \rightarrow \log(n + 1) = k + 1$$

$$k = \log(n + 1) - 1$$

Caminhamento em Árvores

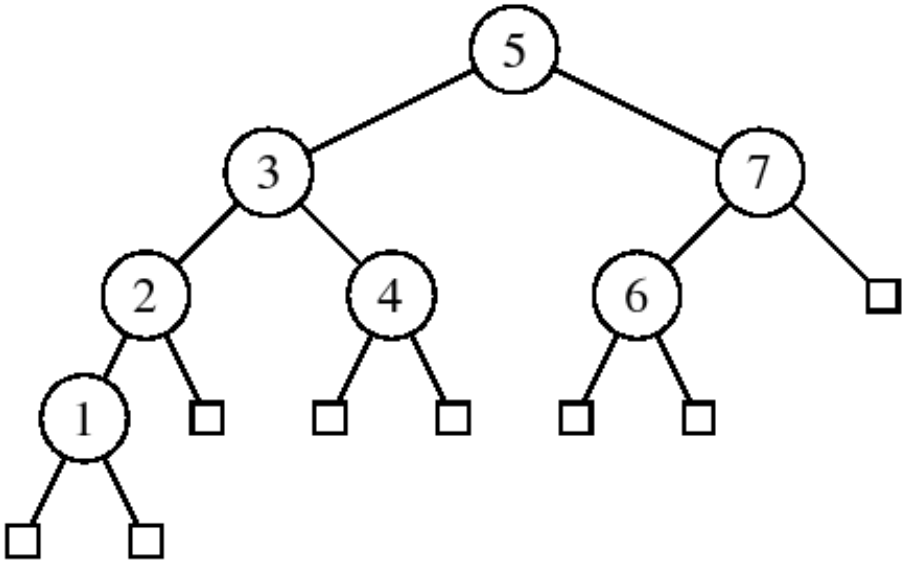
- Em alguns casos, pode ser necessário caminhar na árvores “visitando” (por exemplo imprimindo) todos os seus nodos
- Diversas formas de percorrer ou caminhar em uma árvore listando seus nós
- Tipos mais comuns:
  - Caminhamento em Profundidade
    - Pré-ordem (Pré-fixada)
    - In-ordem (Central ou Infixada)
    - Pós-ordem (Pós-fixada)
  - Caminhamento por nível

Exemplo de Caminhamentos



- Pré-Ordem: ?
- Central: ?
- Pós-Ordem: ?
- Por nível: ?

Exemplo de Caminhamentos



- Pré-Ordem: 5, 3, 2, 1, 4, 7, 6
- Central: 1, 2, 3, 4, 5, 6 , 7
- Pós-Ordem: 1, 2, 4, 3, 6, 7, 5
- Por nível: 5, 3, 7, 2, 4, 6, 1

Pré-Ordem

- Pré-ordem: lista o nó raiz, seguido de suas subárvores (da esquerda para a direita), cada uma em pré-ordem.

Procedimento PREORDEM (n: TipoNo);

```
Início
   Lista(n);
   Para cada filho f de n, da esquerda para direita faça
        PREORDEM(f);
Fim
```

Pós-Ordem

- Pós-ordem: Lista os nós das subárvores (da esquerda para a direita) cada uma em pós-ordem, lista o nó raiz.

Procedimento POSORDEM

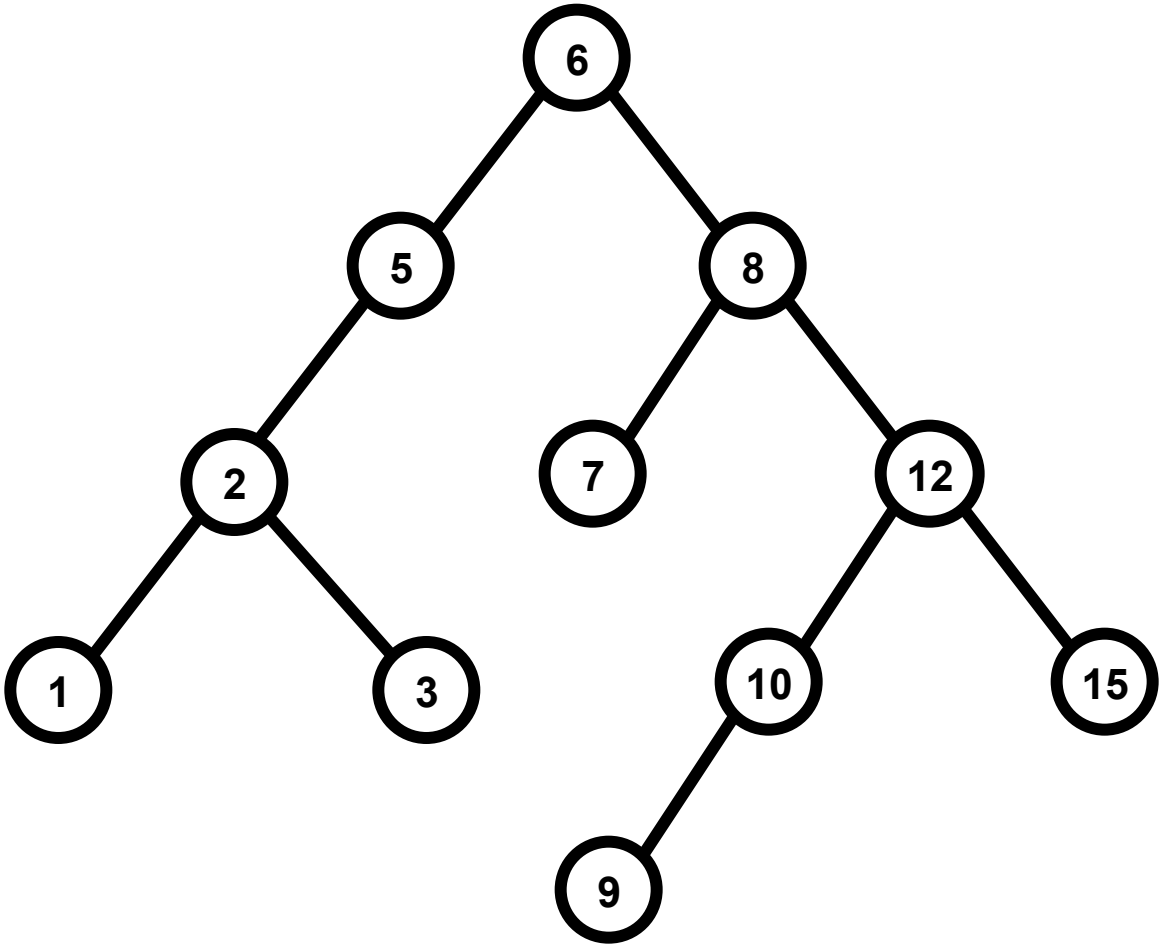
```
 Início
   Para cada filho f de n, da esquerda para direita faça
       POSORDEM(f);
  Lista(n);
Fim;
```

Central

- Central: lista os nós da 1ª. subárvore à esquerda usando o caminhamento central, lista o nó raiz n, lista as demais subárvores (a partir da 2ª.) em caminhamento central (da esquerda para a direita)

```
Procedimento CENTRAL (n: TipoNo);
Início
   /* Folha retorna se n é uma folha da árvore ou não
   Se Folha(n) então Lista(n);
   Senão
      CENTRAL (FilhoMaisEsquerda(n));
       Lista (n);
       Para cada filho f de n, exceto o mais à esquerda,
       da  esquerda para a direita faça
          CENTRAL (f);
Fim;
```

Exemplo 2 de Caminhamentos



- Pré-ordem:

6 5 2 1 3 8 7 12 10 9 15

- Central:

1 2 3 5 6 7 8 9 10 12 15

- Pós-ordem:

1 3 2 5 7 9 10 15 12 8 6

- Por nível

6 5 8 2 7 12 1 3 10 15 9

Outros conceitos

- Nó que não tem antecedente: raiz;
- Nós que não tem descendentes são chamados de folhas. (Os outros são os nós internos)
- A altura de um nó na árvore é o caminho de maior comprimento que se pode fazer deste nó a uma folha.
- A altura da árvore é a altura de sua raiz.
- A profundidade de um nó é o comprimento da raiz até o nó (só existe um caminho)

TAD de Árvore

- O que precisa ser representado?

- Operações:

TAD de Árvore

- O que precisa ser representado?
  - Nó, contendo informação e filhos

- Operações:
  - Cria árvore
  - Inserção
  - Retirada
  - Pesquisa
  - Caminhamento
  - Impressão

Exemplo de Estrutura de Dados para Representar uma Árvore Binária

```
typedef int TipoChave;

typedef struct _registro {
  TipoChave Chave;
   /* outros componentes */
} Registro;


 typedef Struct No {
   Registro Reg;
   Apontador Esq, Dir;
} No;

typedef struct No * Apontador;
typedef Apontador Árvore;
```

Operações de um TAD Árvore

- As operações de inserção, remoção, pesquisa, etc, dependem muito da aplicação na qual a árvore está sendo usada, pois normalmente devem manter uma certa ordenação / hierarquia

- Exemplos:
  - Heap
  - Árvore Binária de Pesquisa

- Essas operações serão estudadas nos próximos capítulos

Exercício

- Usando a estrutura de árvore binária criada no exemplo anterior, escreva funções que recebam um ponteiro para a raiz da árvore e façam:
  - o caminhamento pré-ordem
  - o caminhamento pós-ordem
  - o caminhamento central

Exercício – Caminhamento Pré-ordem

```
void preOrdem(Apontador no)
{
    if (no != NULL) {
        printf("%d ",no->item.Chave);
        preOrdem(no->Esq);
        preOrdem(no->Dir);
    }
}
```

Exercício – Caminhamento Pós-ordem

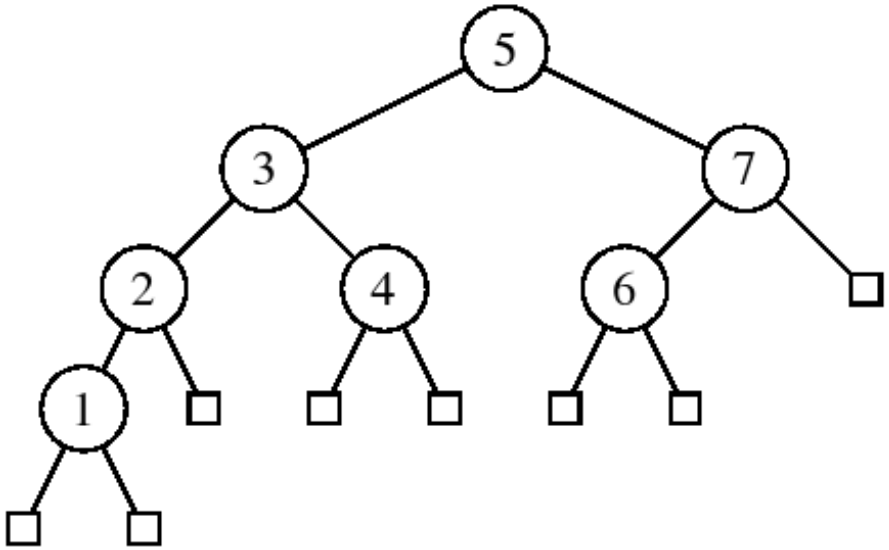
```
void posOrdem(Apontador no)
{
    if (no != NULL) {
        posOrdem(no->Esq);
        posOrdem(no->Dir);
        printf("%d ",no->item.Chave);
    }
}
```

Exercício – Caminhamento Central

```
void emOrdem(Apontador no)
{
    if(no != NULL) {
        emOrdem(no->Esq);
        printf("%d ",no->item.Chave);
        emOrdem(no->Dir);
    }
}
```

Exercício

Faça uma função que recebe um ponteiro para uma árvore e imprime o caminhamento por nível da árvore (*Dica: usar uma fila*)



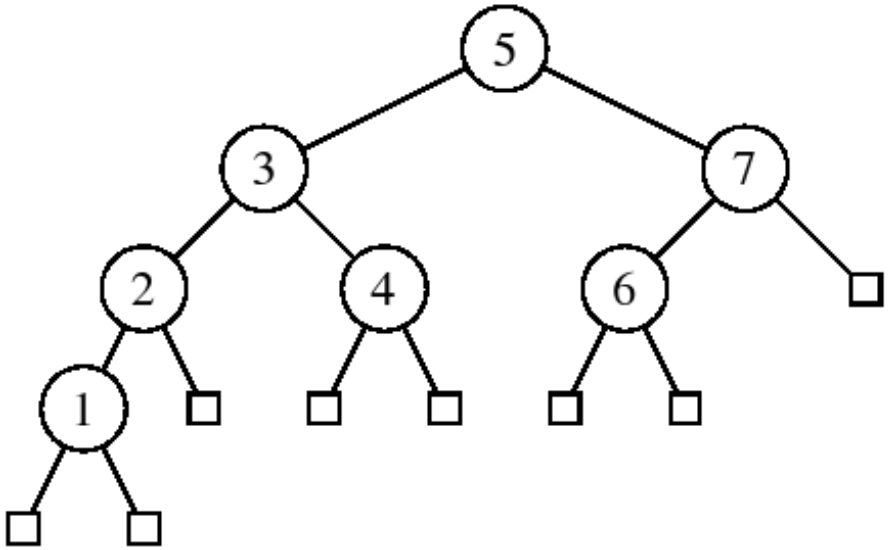
Caminhamento por nível: 5, 3, 7, 2, 4, 6, 1

Exercício – Caminhamento por Nível

```
void porNivel(Apontador no){
    TipoFila F;  // fila de apontadores para nos
     Apontador p;
     FFVazia(&F);
     Enfileira(&F,no);
     while(!FilaVazia(F)) {
          Desenfileira(&F, &p);
          if(p!=NULL) {
               printf("%d ", p->item.Chave);
               Enfileira(&F, p->Esq);
               Enfileira(&F, p->Dir);
          }
     }
}
```

Exercício

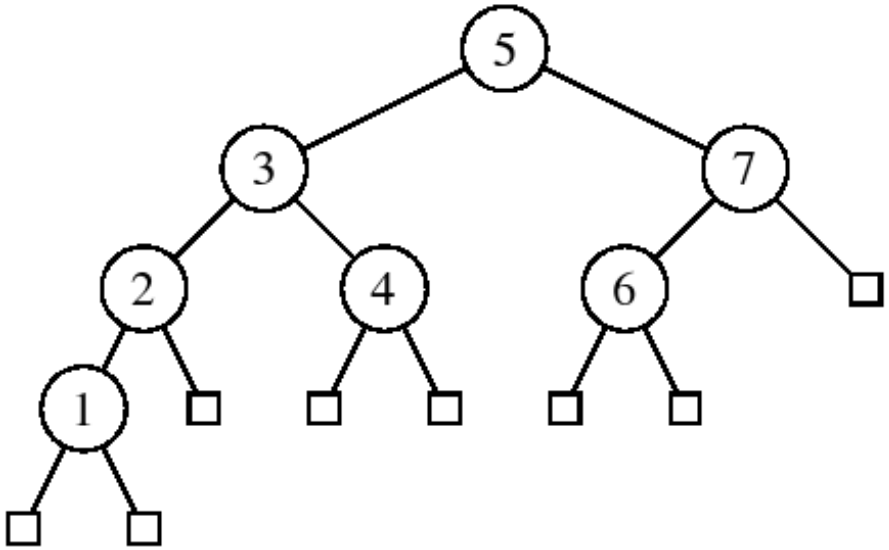
Faça uma função que recebe um ponteiro para uma árvore binária e imprime a sua visão pela direita (*Dica: usar uma fila*)



Visão pela direita: 5, 7, 6, 1

Exercício: Primos

Faça uma função que recebe um ponteiro para uma árvore binária, se dois nós são primos



(2,6): sim
(2,4): não
(1,4): não
(4,6): sim

Exercício: Primos

- Idéia:
  - Realizar um caminhamento central, guardando o nível e o pai dos nós X e Y
  - Uma vez encontrados os dois nós, se seus níveis são iguais e seus pais diferentes, então eles são primos

Exercícios (C++)

```
struct Node
{
    int data;
    Node *left, *right;

    Node(int data)
    {
        this->data = data;
        this->left = this->right = nullptr;
    }
};
```

```
struct NodeInfo
{
    int key;
    int level;
    Node* parent;
};
```

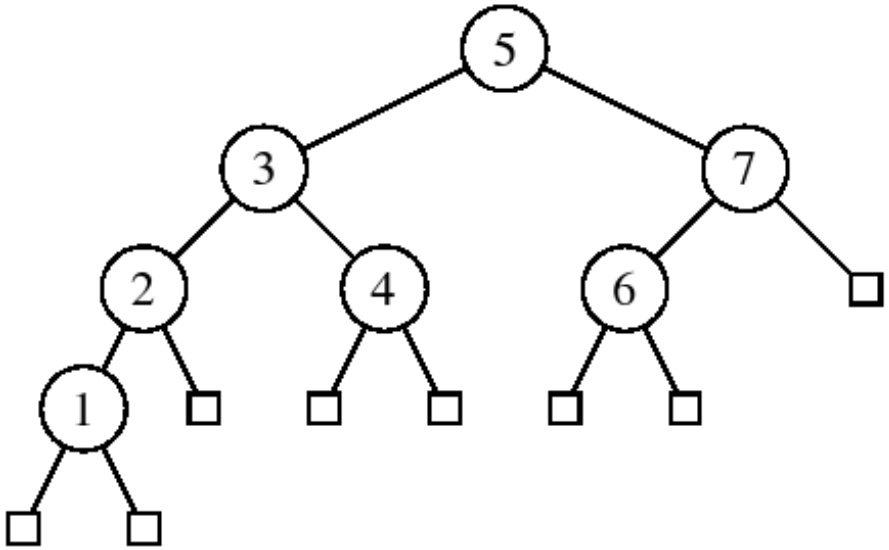
Exercício – Primos

```
bool primos(Node* root, int elem1, int elem2){
    // return if tree is empty
    if (root == nullptr)
        return false;

    int level = 1;                  // level of root is 1
    Node* parent = nullptr; // parent of root is null

    NodeInfo x = {elem1, level, parent};
    NodeInfo y = {elem2, level, parent};

    // perform in-order traversal of the array and
update x and y
    inorder(root, nullptr, 1, x, y);

    // return false if x and y have different level or
same parent
    if (x.level != y.level || x.parent == y.parent)
        return false;

    return true;
}
```

```
void inorder(Node* root, Node *parent, int level,
NodeInfo &x, NodeInfo &y){

    // base case: tree is empty
    if (root == nullptr)
        return;

    // traverse left subtree
    inorder(root->left, root, level + 1, x, y);

   if (root->key == x.key) {
        x.level = level;
        x.parent = parent;
    }

    if (root->key == y.key) {
        y.level = level;
        y.parent = parent;
    }

    // traverse right subtree
    inorder(root->right, root, level + 1, x, y);
}
```

Exercício: LCA

Faça uma função que recebe um ponteiro para uma árvore binária, dois nós x,y, e retorne o seu mínimo ancestral comum (LCA: Least Common Ancestor)



(1,7): 5
(1,4): 3
(6,7): 7

Exercício: LCA

- Idéia:
  - Verificar se X e Y estão presentes na árvore binaria. Se não, LCA=null;
  - Recursivamente encontrar um nó na árvore binária, tal que X esteja na sua subárvore direita e Y na subárvore esquerda, ou vice-versa;
  - Se tal nó não existir, então LCA=X se Y pertencer a uma subárvore de X ou LCA=Y se X pertencer a uma subárvore de Y.

Exercício – LCA

```
bool isNodePresent(Node* root, Node* node)
{
    // base case
    if (root == nullptr)
        return false;

    // if node is found, return true
    if (root == node)
        return true;

    // return true if node is found in the left
subtree  or right subtree
    return isNodePresent(root->left, node) ||
           isNodePresent(root->right, node);
}
```

```
Void printLCA(Node* root, Node* x, Node* y)
{
    Node *lca = nullptr;

    if (isNodePresent(root, y) &&
isNodePresent(root, x))
        findLCA(root, lca, x, y);

    // if LCA exists, print it
    if (lca != nullptr)
        cout << "LCA is " << lca->data << endl;
    else
        cout << "LCA do not exist\n";
}
```

Exercício – LCA

```
bool findLCA(Node* root, Node* &lca, Node* x, Node* y){

   // base case 1: return false if tree is empty
    if (root == nullptr)
        return false;

    // base case 2: return true if either x or y is found
    if (root == x || root == y){
        // set lca to current node
        lca = root;
        return true;
    }

    // recursively check if x or y exists in the left subtree
    bool left = findLCA(root->left, lca, x, y);

    // recursively check if x or y exists in the right subtree
    bool right = findLCA(root->right, lca, x, y);

    if (left && right)
          lca = root;

    // return true if x or y is found in either left or right subtree
    return left || right;
}
```

Exercícios

```
void Insere(Apontador *p, TipoItem x) {
    if(*p == NULL) {
        *p = (Apontador)malloc(sizeof(No));
        (*p)->item= x;
        (*p)->Esq = NULL;
        (*p)->Dir = NULL;
    }
    else if(x.Chave < (*p)->item.Chave)
        Insere(&(*p)->Esq, x);
    else if(x.Chave > (*p)->item.Chave)
        Insere(&(*p)->Dir, x);
    else
        printf("Erro: Registro ja existe na arvore\n");
```

Exercícios

1. Imprima a soma do valor n contido nos nós da árvore;
2. Imprima o valor do maior inteiro contido nos nós da árvore;
3. Imprima a soma do valor n contido nas folhas da árvore;
4. Conte o número de nós em que o valor n é par;
5. Imprima a altura da árvore;

Exercícios

1. Imprima a soma do valor n contido nos nós da árvore

```
int soma(Apontador no) {
    if(no != NULL) {
        return no->item.Chave + soma(no->Esq) + soma(no->Dir);
    }
    return 0;
}
```

Exercícios

2. Imprima o valor do maior inteiro contido nos nós de uma árvore (binária ordenada)

```
int maiorBinaria(Apontador no) {
    if(no->Dir != NULL)
        return maiorBinaria(no->Dir);
    return no->item.Chave;
}
```

Exercícios

2. Imprima o valor do maior inteiro contido nos nós da árvore (para árvores binárias sem ordem)

```
int maior(Apontador no) {
   if(no != NULL) {
      int minhaChave = no->item.Chave;
      int maiorEsq = maior(no->Esq);
      int maiorDir = maior(no->Dir);
      if( minhaChave > maiorEsq) { if(minhaChave > maiorDir)  return minhaChave;
                                          return maiorDir;
      } else{ if(maiorEsq > maiorDir) return maiorEsq;
                          return maiorDir;
      }
   }
   return 0;
}
```

Exercícios

3. Imprima a soma do valor n contido nas folhas da árvore

```
int somaFolhas(Apontador no) {
    if(no == NULL)
        return 0;
    if (no->Dir != NULL || no->Esq != NULL) {
        return somaFolhas(no->Esq) + somaFolhas(no->Dir);
    }
    return no->item.Chave;
}
```

Exercícios

4. Conte o número de nós em que o valor n é par;

```
int contaPar(Apontador no) {
    if(no == NULL)
        return 0;
    else {
        if(no->item.Chave % 2 == 0)
            return 1 + contaPar(no->Esq) + contaPar(no->Dir);
        return 0 + contaPar(no->Esq) + contaPar(no->Dir);
    }
}
```

Exercícios

5. Imprima a altura da árvore

```
int alturaArvore(Apontador no) {
    if(no == NULL)
        return -1;
    else {
        int alturaEsquerda = alturaArvore(no->Esq);
        int alturaDireita = alturaArvore(no->Dir);
        if(alturaEsquerda > alturaDireita)
            return 1 + alturaEsquerda;
        return 1+ alturaDireita;
    }
}
```