Árvores

Estrutura de Dados

Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação

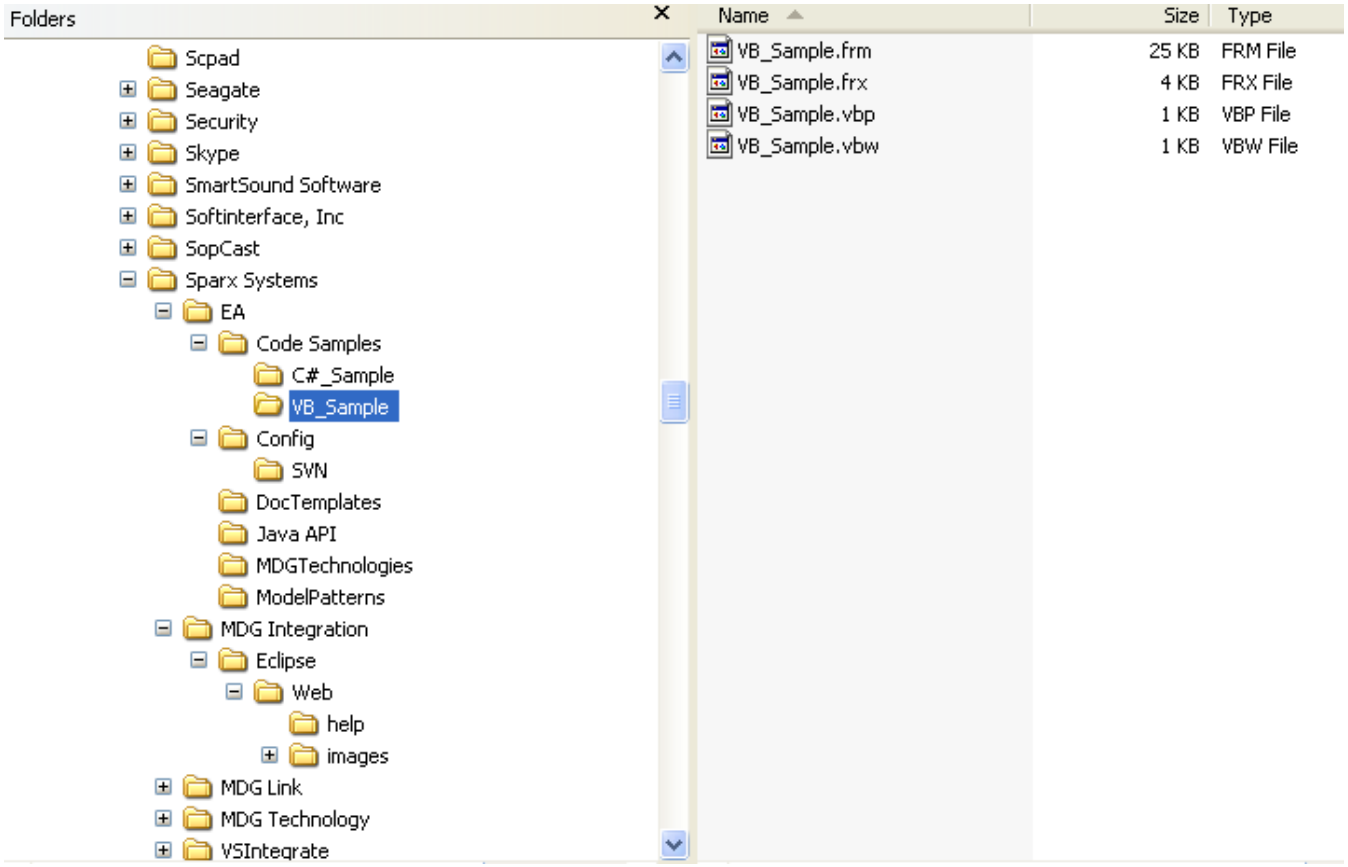
Conteúdo

- Árvores
  - Definições
  - Conceitos
  - Algoritmos
- Árvore Binária
  - Implementação
  - Algoritmos
- Árvore Binária de Busca
  - Implementação
  - Algoritmos
- Exercícios

Árvores

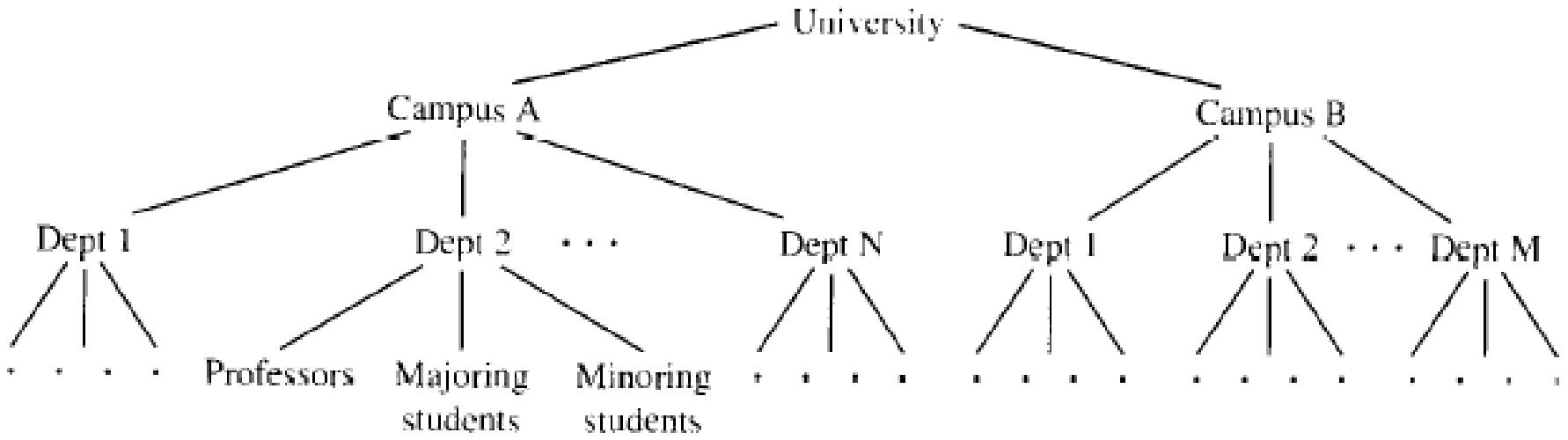
Árvores: Motivação

- Qual estrutura de dados um gerenciador de arquivos deve utilizar?



Árvores: Motivação

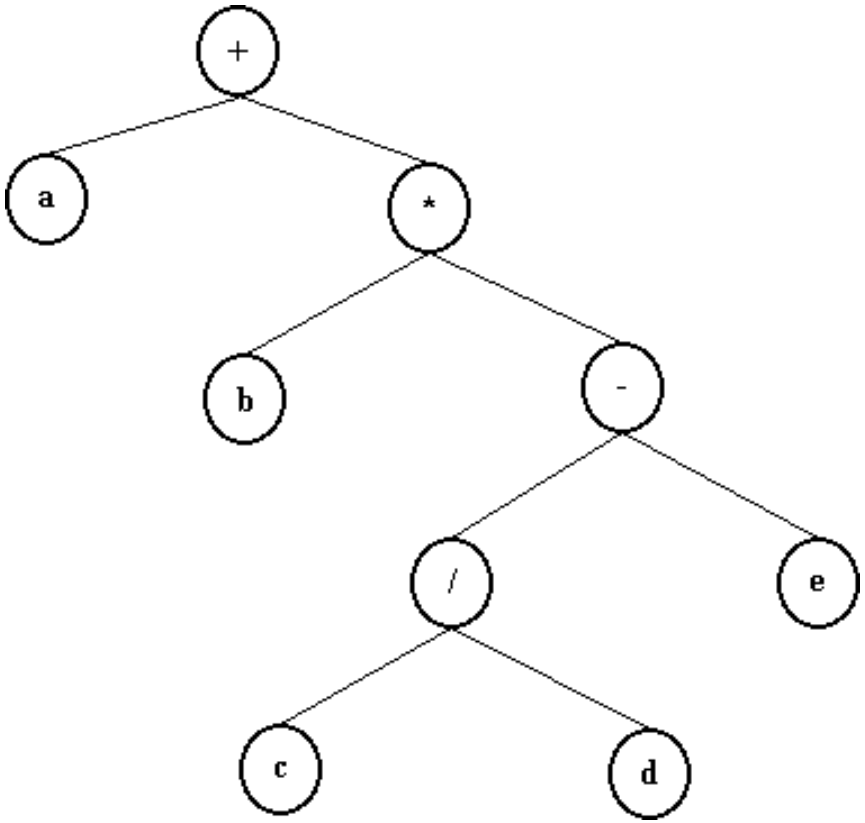
- Ex: Hierarquia Universitária



Árvores: Motivação

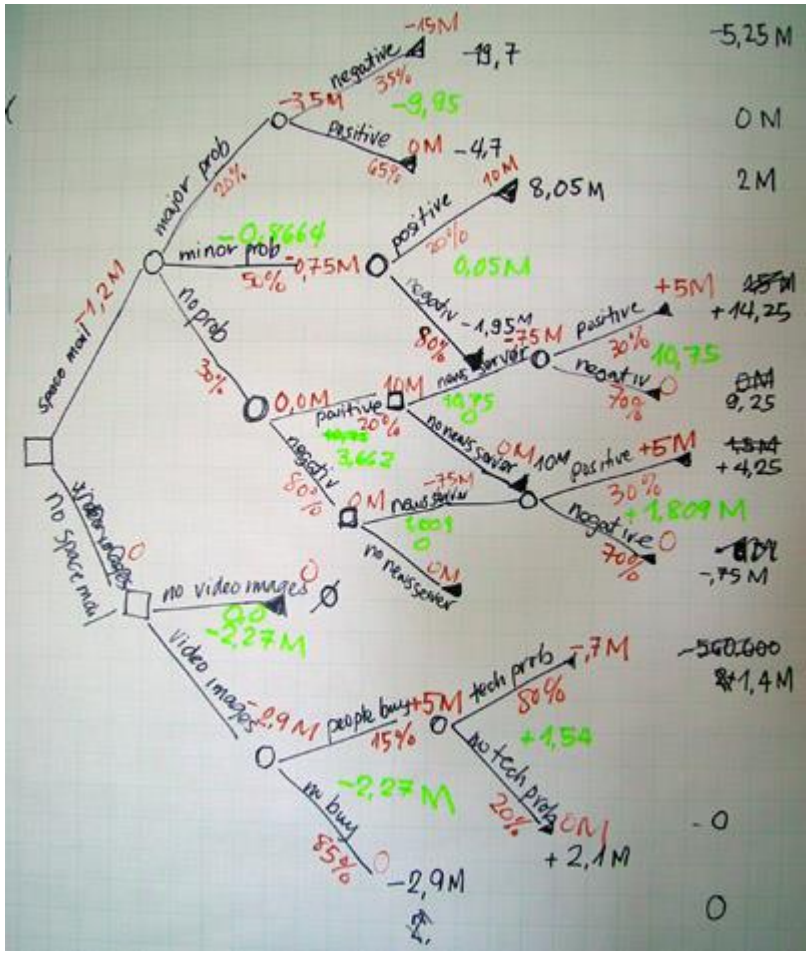
- Representação da expressão aritmética:

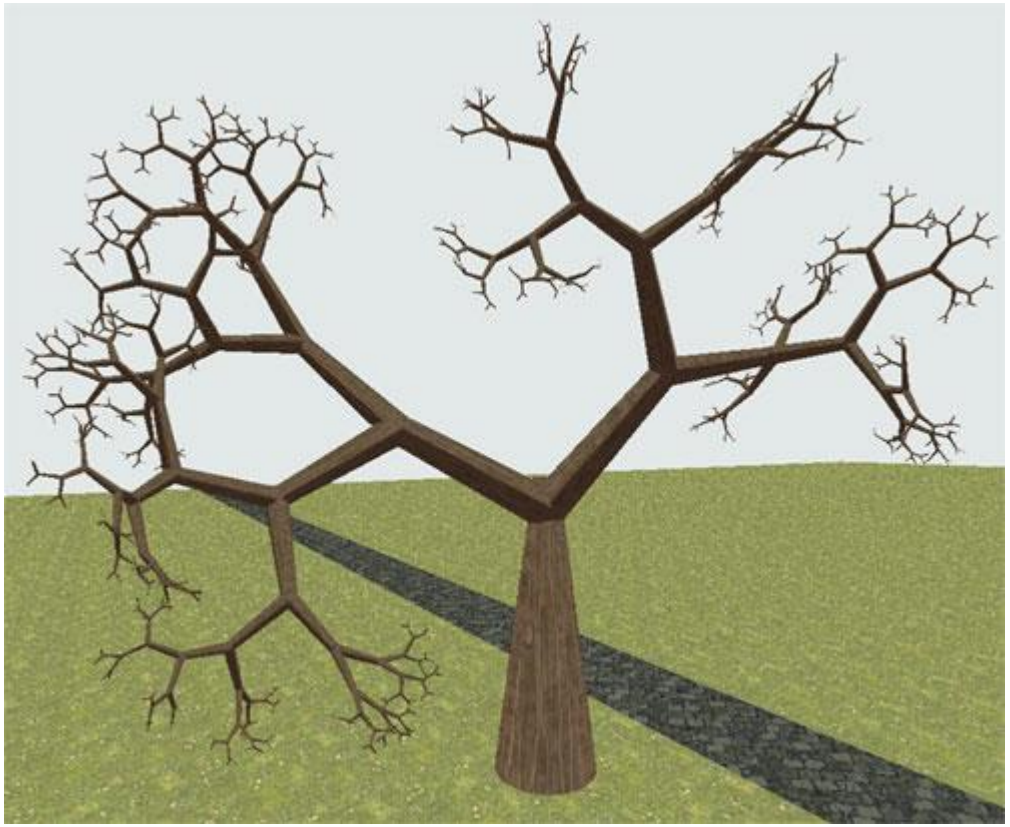
  (a + (b *((c / d) - e)))



Árvores: Motivação

- E muitas outras aplicações...





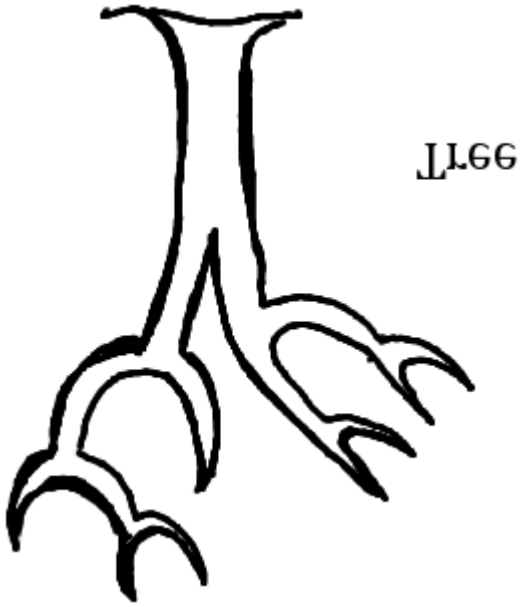
Árvores: Motivação

- Listas ligadas
  - São mais flexíveis do que vetores;
  - Possuem acesso sequencial;
  - São estruturas lineares sendo difícil utilizá-las para organizar representação hierárquica de objetos.
- Pilhas e filas
  - Refletem alguma hierarquia;
  - Mas são limitadas somente a uma dimensão.

Árvores: Motivação

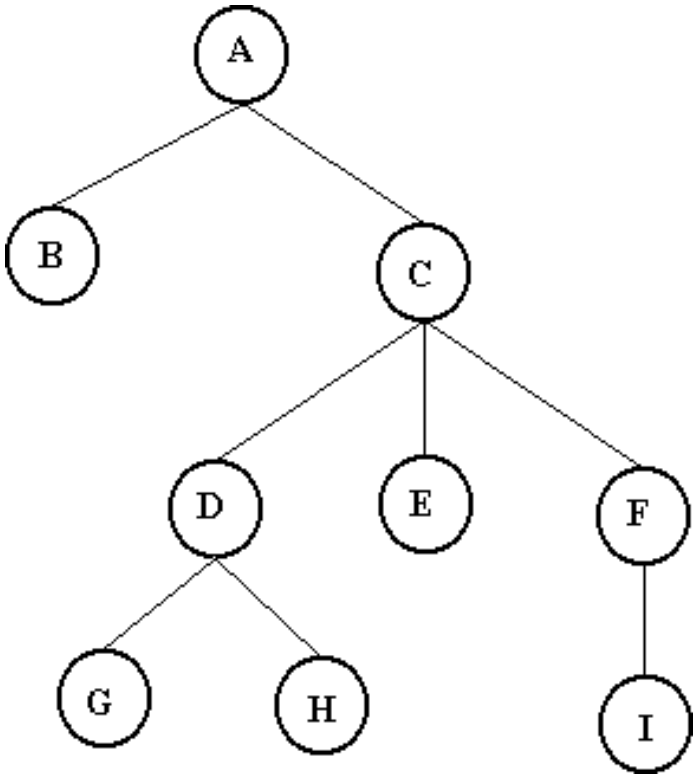
- Árvore
  - Estrutura criada para superar limitações de listas ligadas, pilhas e filas;
  - Consiste de nós e de arcos;
  - São representadas com a raiz no topo e as folhas na base (*diferente de árvore natural*).

Tree



Árvores: Representações Gráficas

- Hierárquica:



alinhamento dos nós:

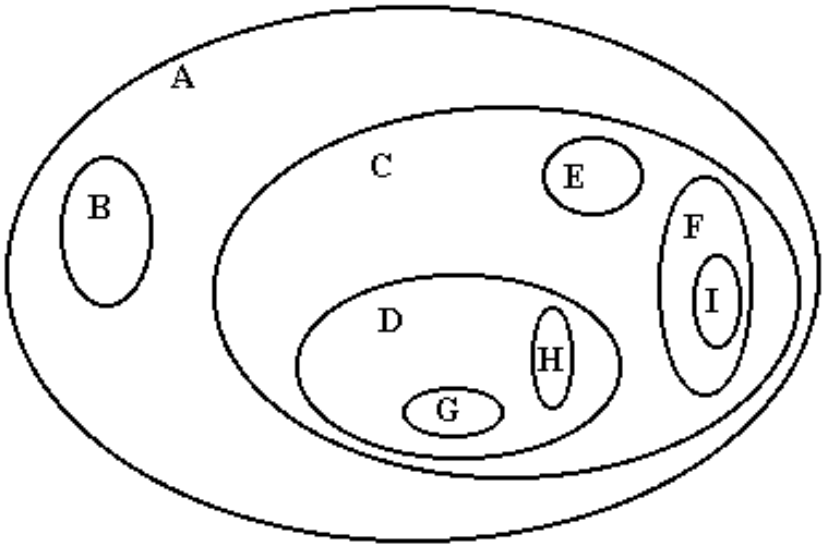
```
A
    B
    C
        D
                G
                H
        E
        F
                I
```

Árvores: Representações Gráficas

- Parênteses aninhados

  (A (B) (C (D (G) (H)) (E) (F (I))))

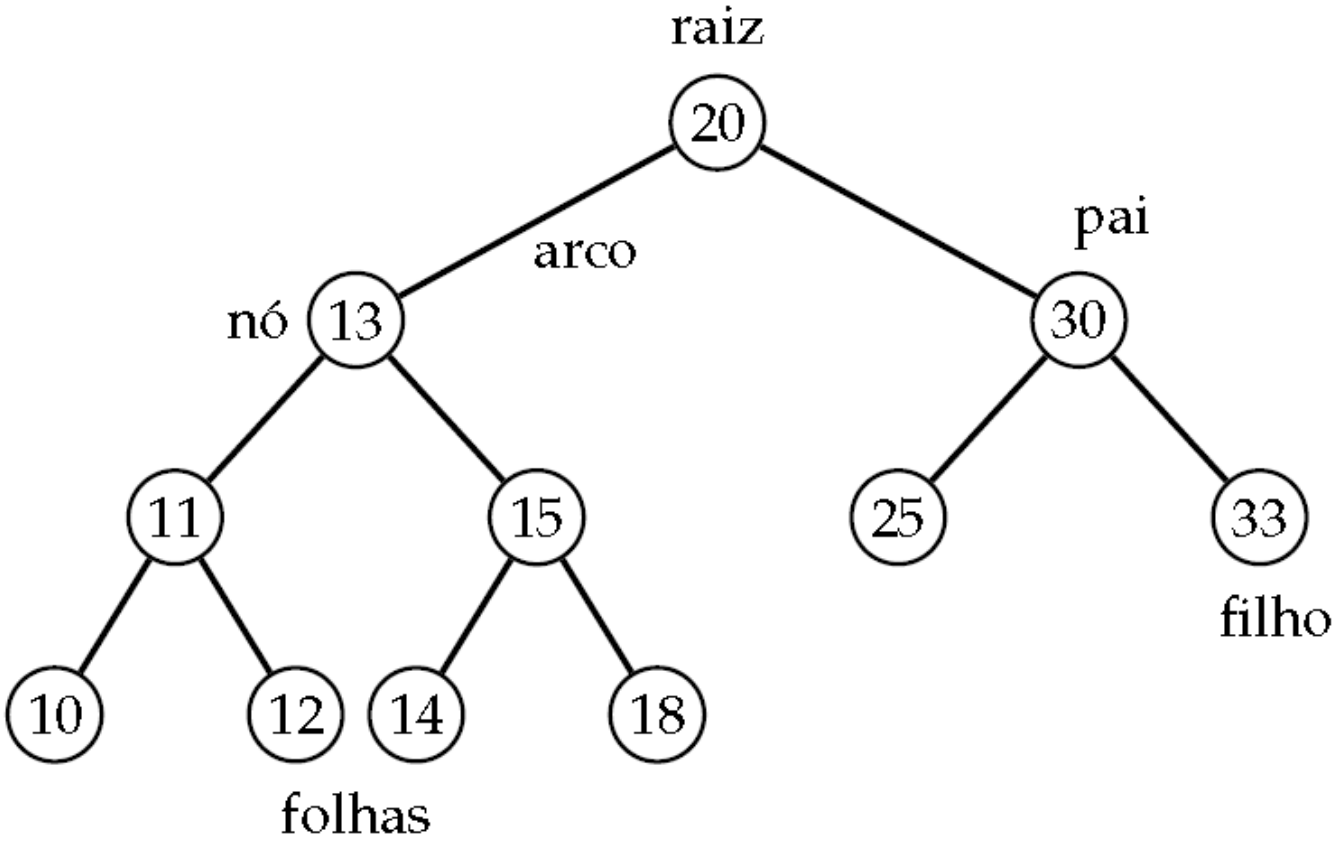
- Diagramas de inclusão



Árvores: Definições

- Nó: Elemento que contém a informação
- Arco: Liga dois nós
- Pai: nó superior de um arco
- Filho: nó inferior de um arco
- Raiz (nó topo)
  - Não possui ancestrais (não tem nó pai)
  - Só pode ter filhos
- Folhas
  - nós das extremidades inferiores
  - Não têm nós filhos (ou melhor, seus filhos são estruturas vazias)

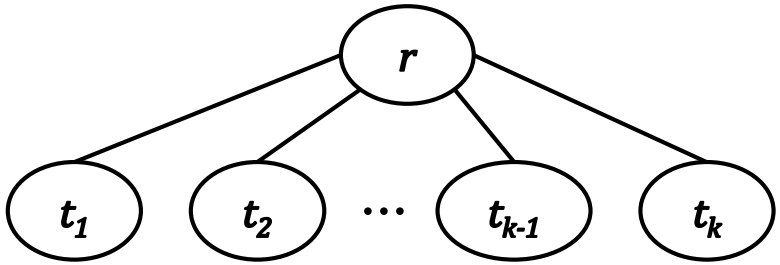
Árvores: Definições



Árvores: Definições

- Definição recursiva de árvore

1. Uma estrutura vazia é uma árvore vazia.
2. Se t_1, ..., t_k são raízes de árvores disjuntas, então a estrutura cuja raiz r tem como suas filhas as raízes t_1, ..., t_k também é uma árvore.

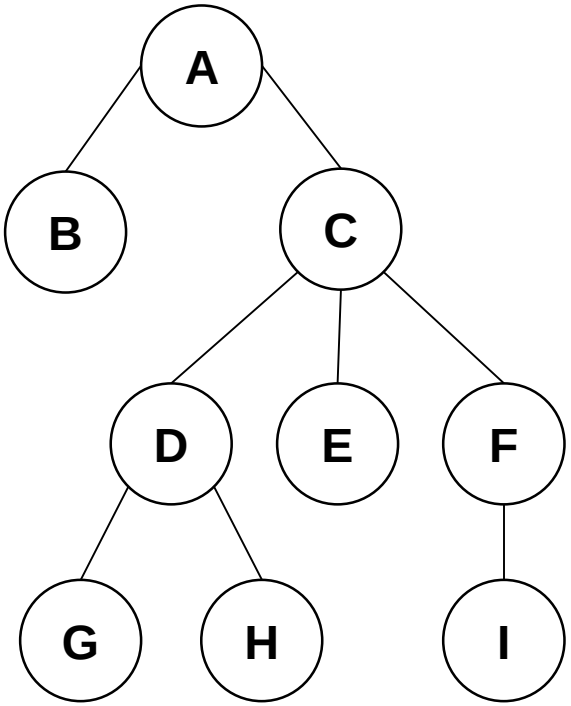


3. Somente estruturas geradas pelas regras 1 e 2 são árvores.

Árvores: Definições

- **Grau de um nó** é o número de subávores de um nó. (Exemplo no próximo slide)
- **Grau de uma árvore** (aridade): é definido como sendo igual ao grau máximo entre todos os nós da árvore.
- Cada nó tem que ser atingível a partir da raiz, através de uma sequência única de arcos chamada de **caminho**.
- O **comprimento do caminho** é o número de arcos do caminho
- **Nível de um nó** é a distância (em arcos) entre o nó e a raiz da árvore. A raiz tem nível igual a 0.

Árvores: Definições



- Graus dos nós
  - G(A)=2
  - G(B)=0
  - G(C)=3
  - G(D)=2
  - G(E)=0
  - G(F)=1
  - G(G)=0
  - G(H)=0
  - G(I)=0

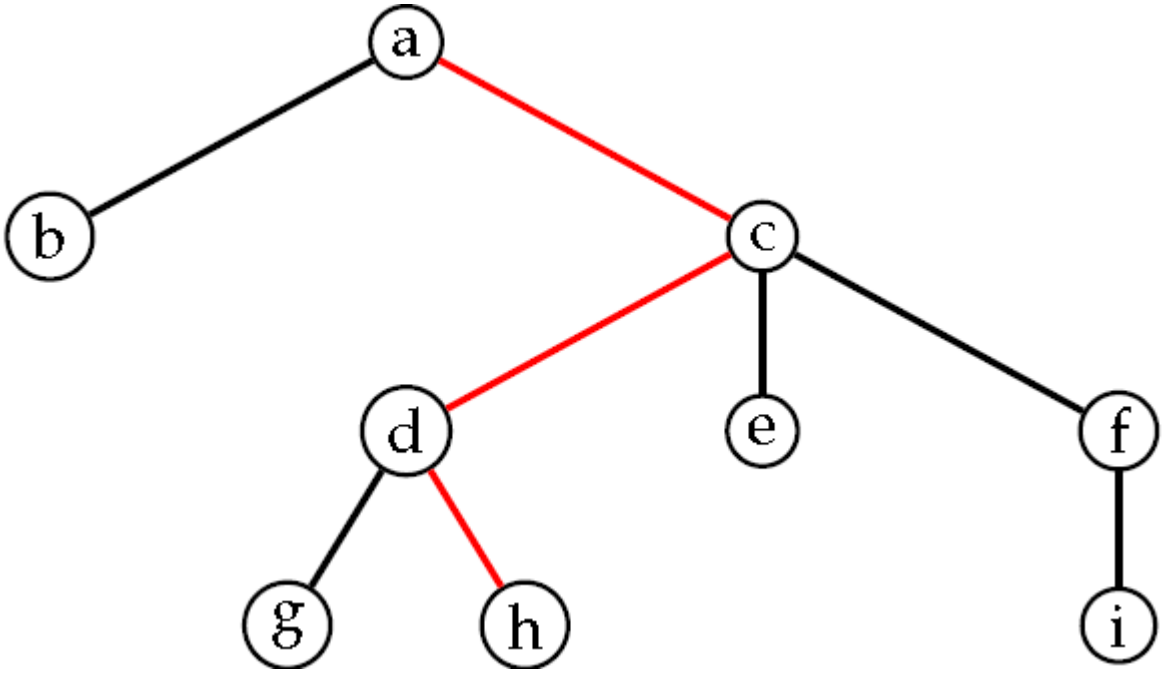
Nós Internos

Folhas

Grau(T) = 3

Árvores: Exemplos

- Exemplo de caminho que vai do nó *a* até o nó *h*.
- O comprimento desse caminho é 3.
- Nesse exemplo, o nó *a* tem nível 0; os nós *b* e *c* têm nível 1; o nó *h* tem nível 3.

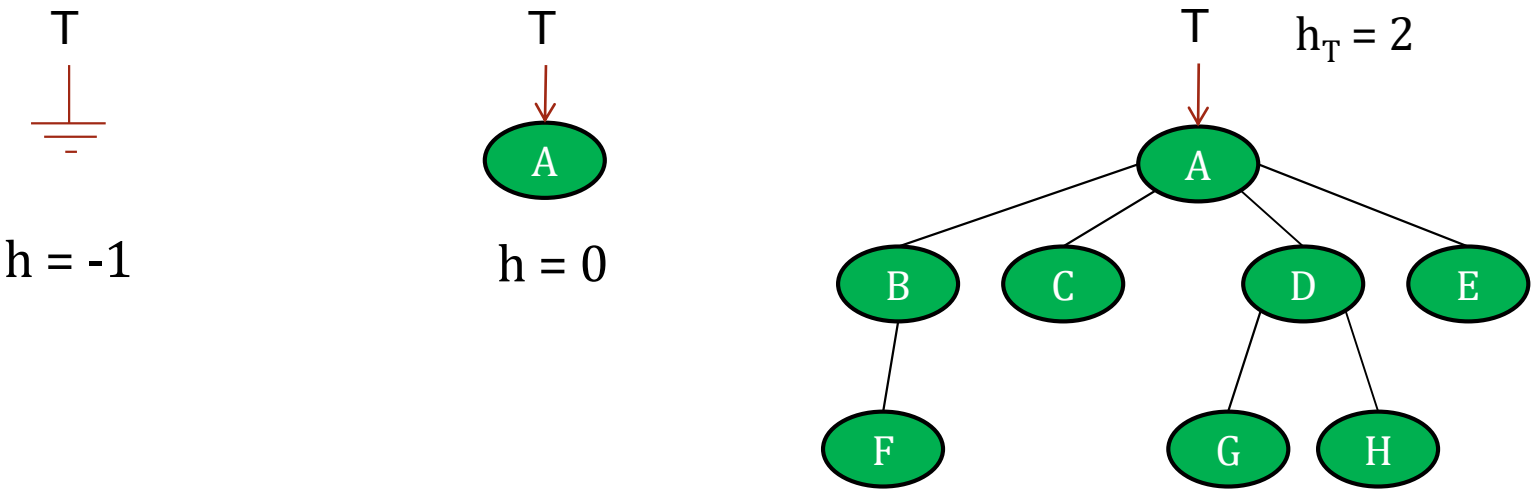


Árvores: Definições

- A **altura de um nó** *x* é a distância entre *x* e o seu descendente mais afastado (folha). Mais precisamente, a altura de *x* é o número de arcos do mais longo caminho que vai de *x* até uma folha.
- **Altura** (ou **profundidade**) da árvore é o nível do nó folha que tem o mais longo caminho até a raiz.
  - A árvore vazia é uma árvore de altura -1, por definição.
  - Uma árvore com um único nó tem altura 0.
  - O nó é raiz e folha ao mesmo tempo.
- Toda árvore com n>1 nós possui:
  - no mínimo 1 folha; e
  - no máximo *n*-1 folhas.

Árvores: Definições

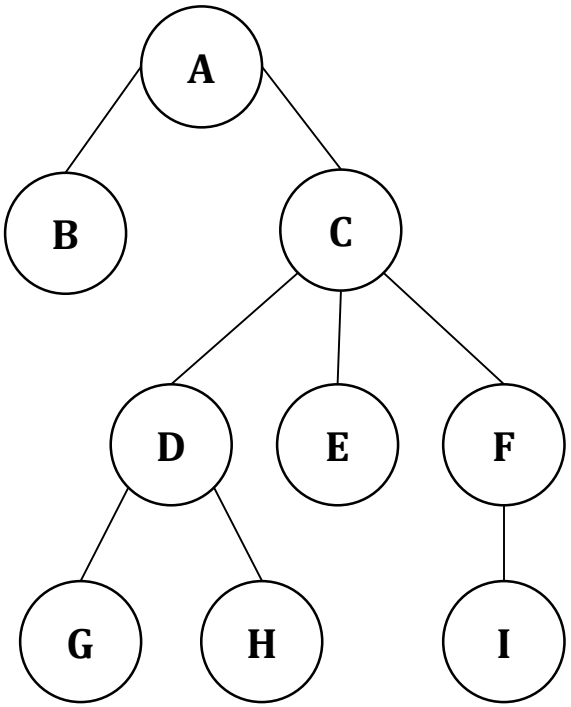
- Exemplos de alturas da árvore T:



- Pela definição, a altura de uma subárvore de uma folha é -1. Portanto, a altura de qualquer folha é 0 (zero).

Árvores: Definições

- Exemplo de níveis e altura da árvore

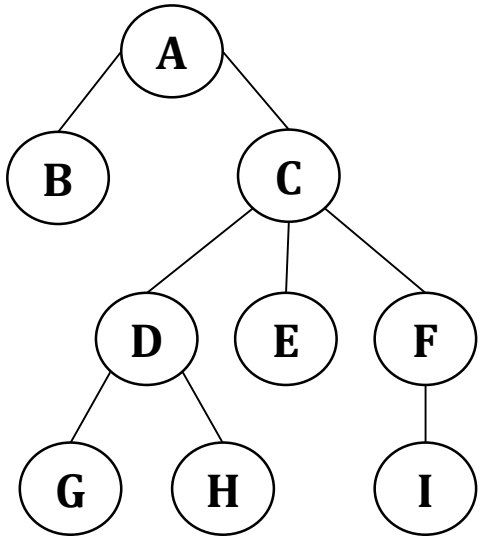


NÍVEIS	
A	0
B, C	1
D, E, F	2
G, H, I	3

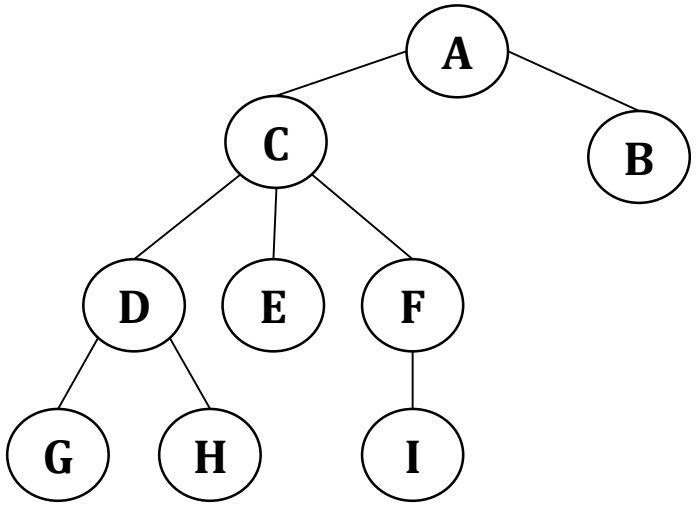
h(T) = 3

Árvores: Definições

- Árvore Ordenada: os filhos de cada nó estão ordenados (assume-se ordenação da esquerda para a direita) .
- Exemplos:



Ordenada



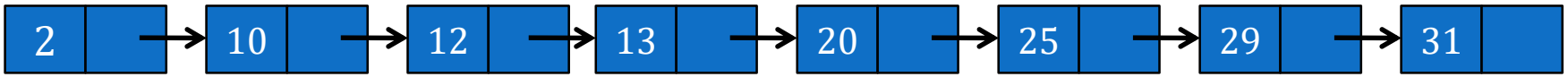
Não-ordenada

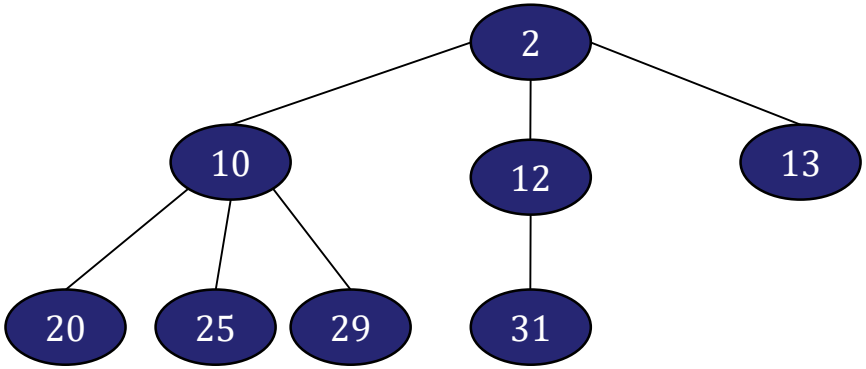
Árvores: Definições

- A definição de árvore não impõe qualquer condição sobre o número de filhos de um nó:
  - Pode variar de 0 a qualquer inteiro;
- Árvores são muito utilizadas em sistemas gerenciadores de banco de dados.

Árvore

- Considerando a lista encadeada e a árvore abaixo, qual pesquisa é mais rápida para achar um valor (chave)?

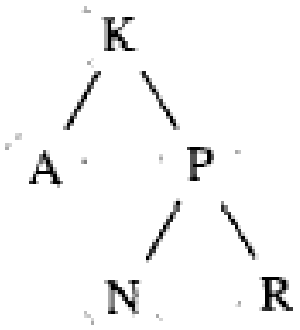


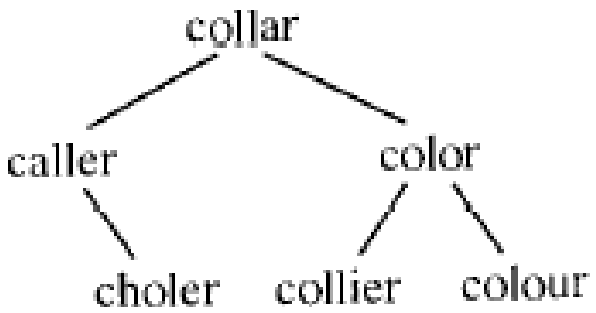


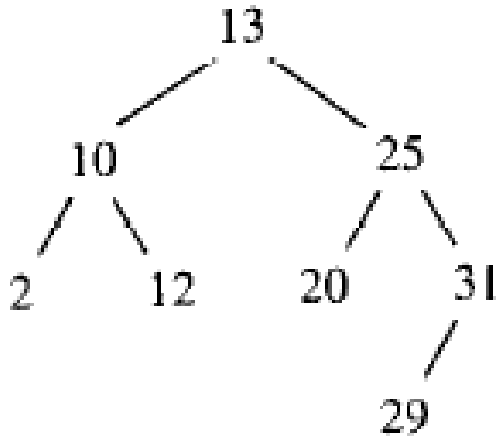
Árvore Binária

Árvores Binárias

- Uma árvore binária é uma árvore cujos nós têm no máximo, dois filhos, portanto, a árvore binária tem **grau 2**.
- Em uma árvore binária cada filho é designado como filho à esquerda ou filho à direita (ou Nó filho ESQUERDO e Nó filho DIREITO).
- Exemplos:





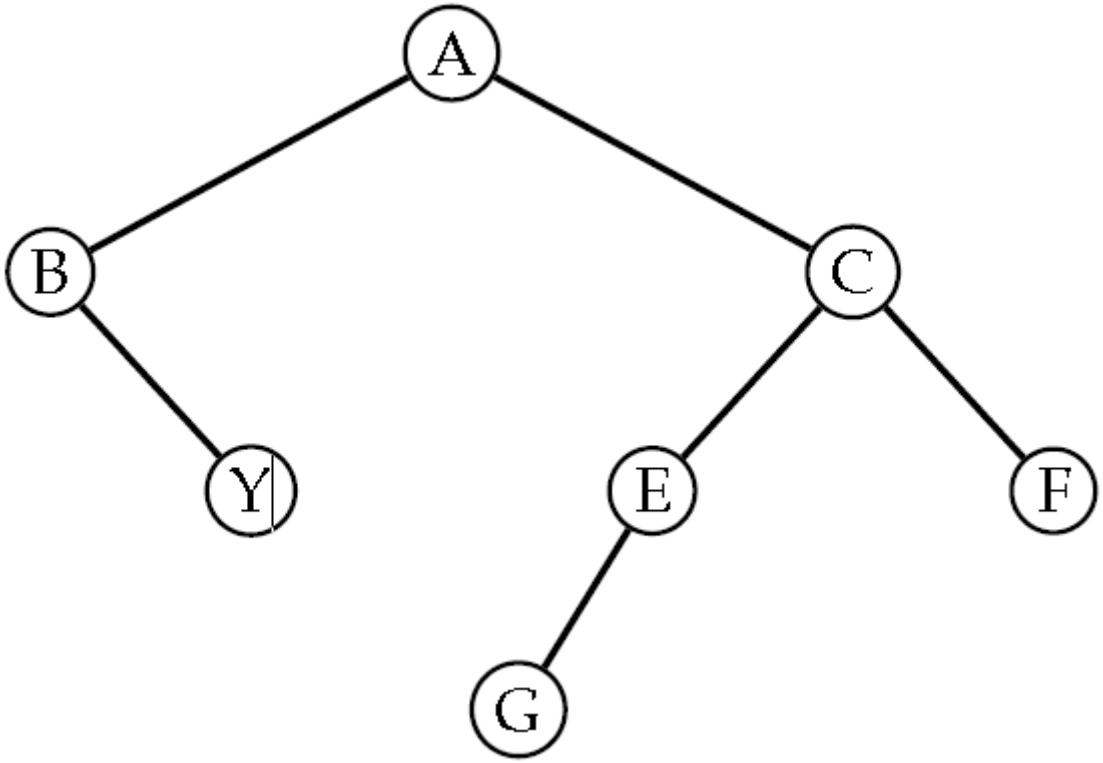


Árvores Binárias

- Definição: Uma árvore binária T é um conjunto finito de elementos denominados nós ou vértices, tal que:
  - Se $T = \emptyset$, a árvore é dita vazia, ou
  - Existe um nó especial r, chamado raiz de T, e os nós restantes podem ser divididos em dois subconjuntos disjuntos, T_r^L e T_r^R, correspondentes às subárvores a esquerda e a direita de r, respectivamente, que também são árvores binárias.

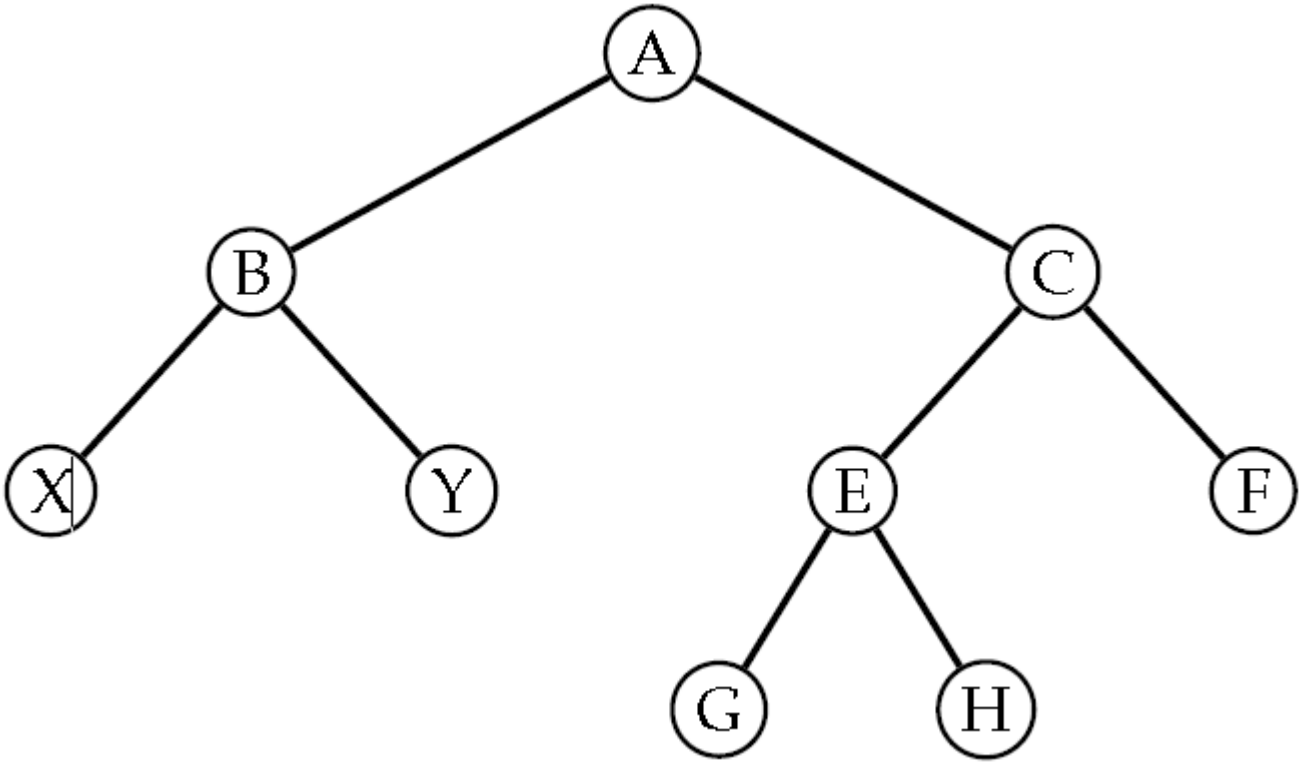
Árvores Binárias

- Exemplo:



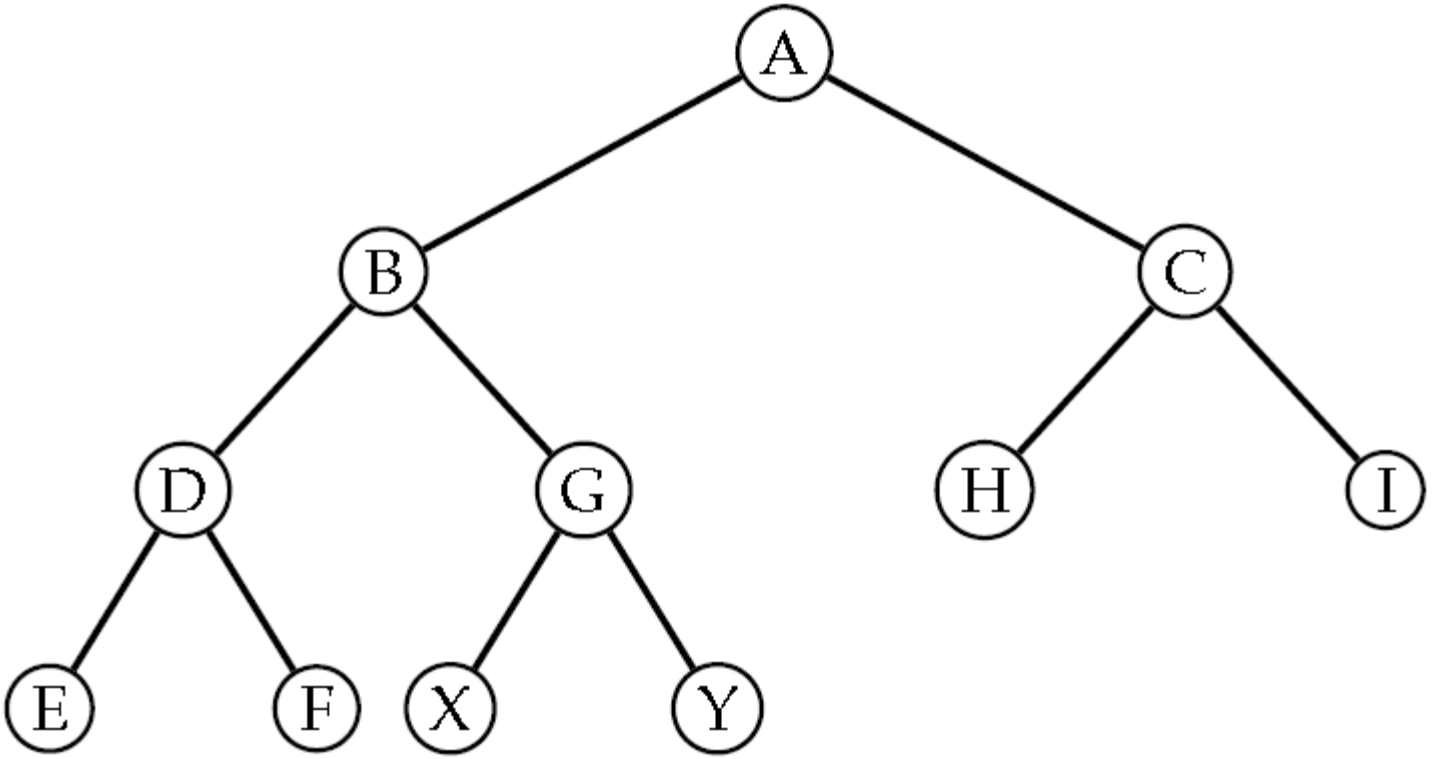
Árvores Binárias

- **Árvore estritamente binária**: cada nó possui 0 ou 2 filhos.



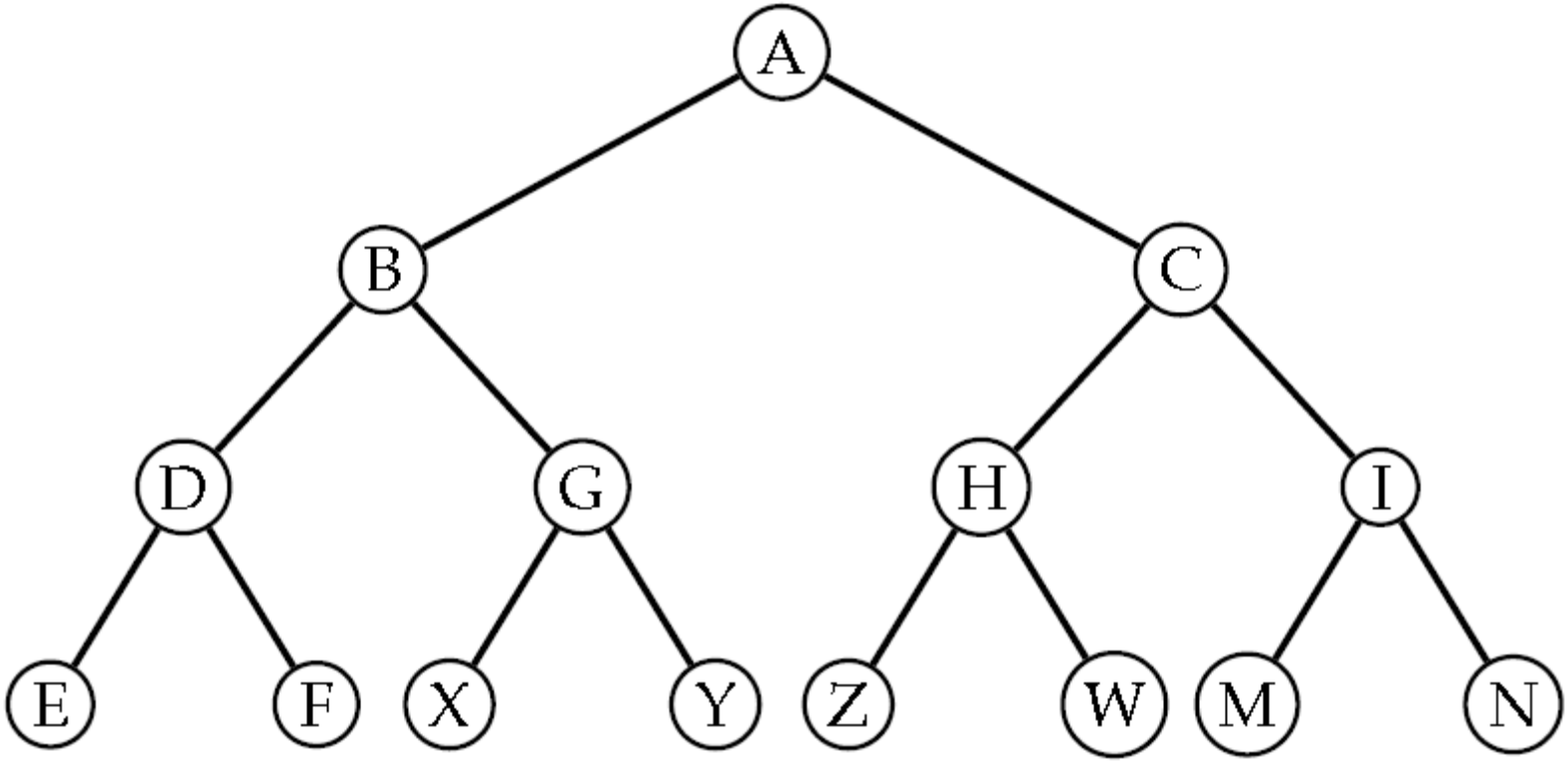
Árvores Binárias

- **Árvore binária completa**: se *v* é um nó tal que alguma subárvore de *v* é vazia, então *v* se localiza ou no último (maior) ou no penúltimo nível da árvore.



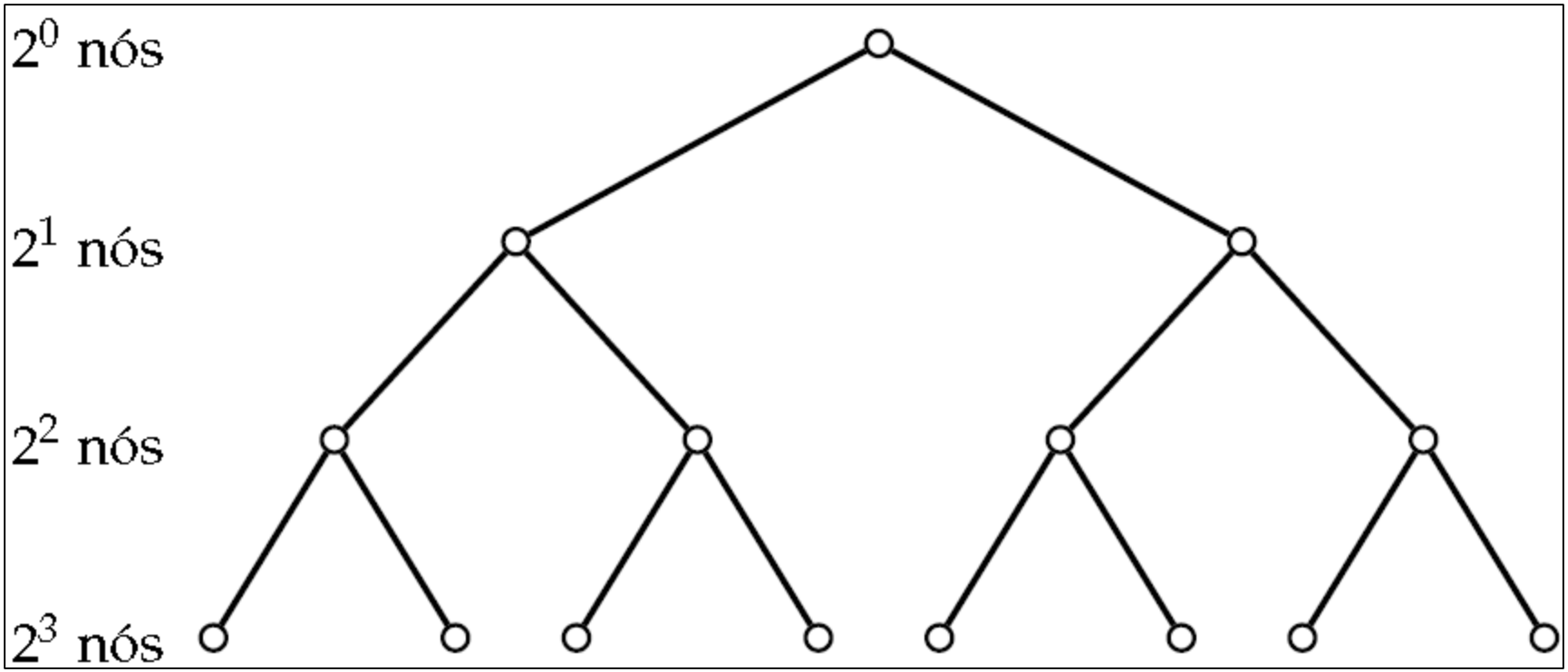
Árvores Binárias

- **Árvore binária cheia**: se *v* é um nó tal que alguma subárvore de *v* é vazia, então *v* se localiza no último (maior) nível da árvore. Neste caso, *v* é um nó folha.



Árvores Binárias

- Propriedade: Em uma árvore **binária cheia**, o número de nós do nível *i* é igual a 2^i. Consequentemente, em qualquer árvore binária existe no máximo 2^i nós no nível *i*.



Representações de Árvores Binárias

- Representação sequencial como matriz

Índice	Info	Esquerda	Direita
0	13	4	2
1	31	6	-1
2	25	7	1
3	12	-1	-1
4	10	5	3
5	2	-1	-1
6	29	-1	-1
7	20	-1	-1

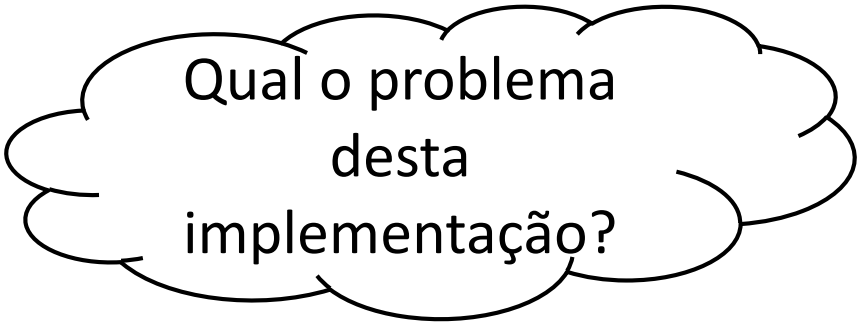
- Nó: estrutura com um campo de informação e dois indicadores para os filhos a esquerda e a direita.
- A raiz é alocada na primeira célula (de índice 0).
- Indicador igual a -1 corresponde a inexistência de filho naquela direção.

- Aplicação: desenhar a árvore representada acima.

Representações de Árvores Binárias

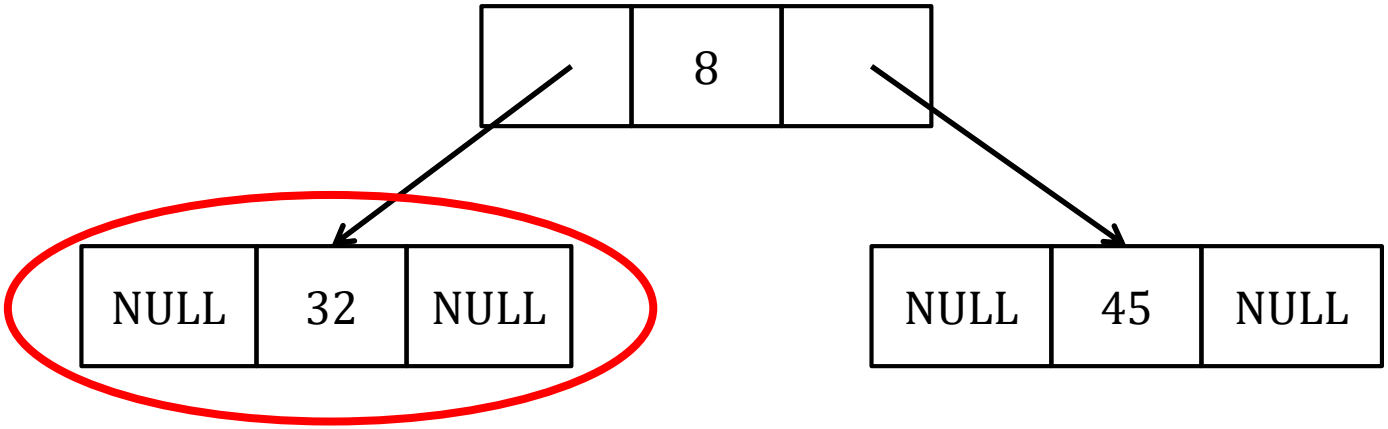
- Representação sequencial como matriz

Índice	Info	Esquerda	Direita
0	13	4	2
1	31	6	-1
2	25	7	1
3	12	-1	-1
4	10	5	3
5	2	-1	-1
6	29	-1	-1
7	20	-1	-1



Representações de Árvores Binárias

- Representação encadeada:
- Neste caso, um nó será constituído por três campos:
  - Campo de informação.
  - Ponteiro para nó filho a esquerda.
  - Ponteiro para nó filho a direita.
- Esquematicamente:



Nó da árvore

Árvores Binárias

- TAD `NoArv` para uma árvore binária de inteiros

```
class NoArv
{
  private:
    NoArv *esq; // ponteiro para o filho à esquerda
    int info;   // informação do nó (int)
    NoArv *dir; // ponteiro para o filho à direita
  public:
    NoArv(); //construtor e destrutor
    ~NoArv();
    void setEsq(NoArv *p); //operações setter
    void setInfo(int val);
    void setDir(NoArv *p);
    NoArv* getEsq(); //operações getter
    int getInfo();
    NoArv* getDir();
};
```

Árvores Binárias

- MI do TAD `NoArv`

```
NoArv::NoArv() {}

NoArv::~NoArv() {}

void NoArv::setEsq(NoArv *p)   { esq = p;}

void NoArv::setInfo(int val)   { info = val;}

void NoArv::setDir(NoArv *p)   { dir = p;}

NoArv* NoArv::getEsq()        { return esq;}

int NoArv::getInfo()           { return info;}

NoArv* NoArv::getDir()        { return dir;}
```

Árvores Binárias

- Como representar a árvore binária? Basta ter um ponteiro para a raiz. TAD `ArvBin` (inteiros):

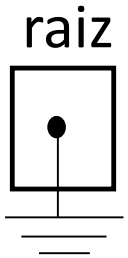
```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o nó raiz da árvore

   public:
      ArvBin();
      ~ArvBin();
      int getRaiz();
      // cria novo nó como raiz das sub-árvores à
      //esquerda (sae) e à direita (sad)
      void cria(int val, ArvBin *sae, ArvBin *sad);
      bool vazia(); // verifica se a árvore está vazia
      void imprime();
      bool busca(int val);
};
```

Árvores Binárias

- MI do TAD `ArvBin`
  - Construtor

```
ArvBin::ArvBin()
{
    raiz = NULL;
}
```



Cria árvore binária vazia

Árvores Binárias

- MI do TAD `ArvBin`
  - Operação getRaiz:

```
int ArvBin::getRaiz()
{
    if(raiz != NULL) //ou if(!vazia())
        return raiz->getInfo();
    else
    {
        cout << "Árvora vazia!“ << endl;
        exit(1);
    }
}
```

Árvores Binárias

- MI do TAD `ArvBin`
  - Operação vazia:

```
bool ArvBin::vazia()
{
   return raiz == NULL;
}
```

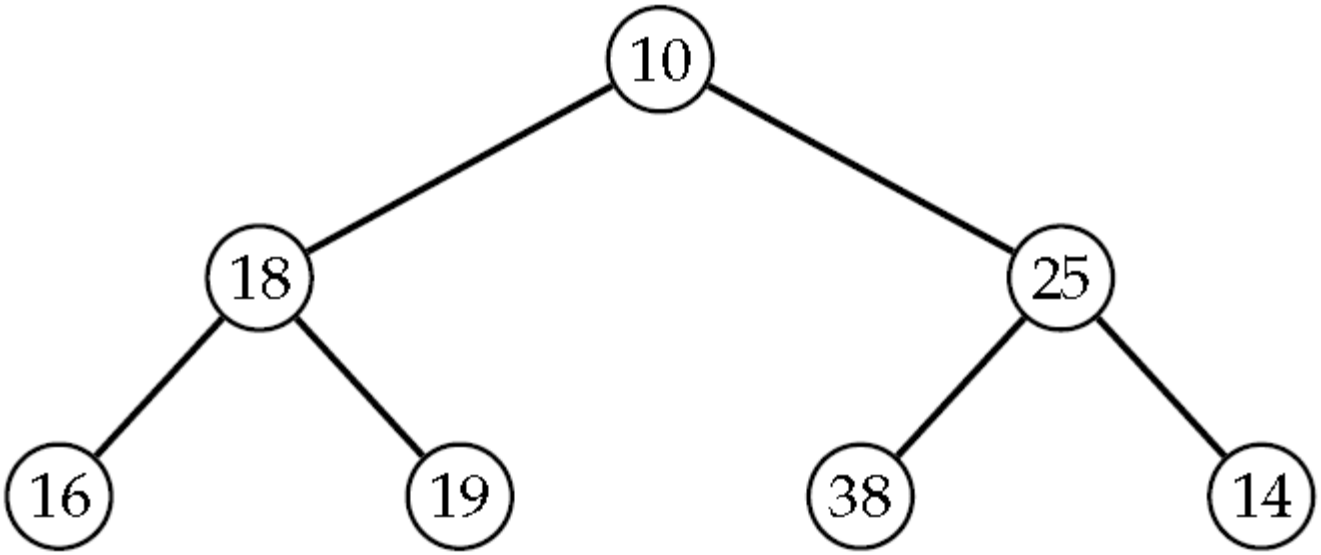
Árvores Binárias

- MI do TAD `ArvBin`
  - Operação cria: cria um novo nó como raiz das sub-árvores da esquerda (sae) e da direita (sad):

```
void ArvBin::cria(int val, ArvBin *sae, ArvBin *sad)
{
    NoArv *p = new NoArv();
    p->setInfo(val);
    p->setEsq(sae->raiz);
    p->setDir(sad->raiz);
    raiz = p;
}
```

Árvores Binárias

- Como usar a função cria() para gerar a seguinte árvore?



Árvores Binárias

- Antes de ver como implementar as operações `imprime()`, `busca()` e o destrutor da classe `ArvBin`, serão apresentados os diferentes algoritmos de percurso em árvore.
- Percurso corresponde a uma visita sistemática a cada um dos nós da árvore.
- Esta é uma das operações básicas relativas à manipulação de árvores.
- Uma árvore é essencialmente uma estrutura não-sequencial.

Percurso Árvores Binárias

- Para percorrer a árvore deve-se, então, visitar cada um de seus nós.
- Visitar um nó significa **operar** com a informação do nó.
- Por exemplo: imprimir, atualizar informações etc.
- Percorrer uma árvore significa visitar os seus nós exatamente uma vez
- Contudo, durante um percurso pode-se passar várias vezes por alguns nós sem visitá-los.

Percurso Árvores Binárias

- Passos básicos do percurso em uma árvore binária T.
- Visitar a raiz de cada subárvore de T.
- Visitar as 2 subárvores de T.
- Percorrer as subárvores da esquerda e da direita da raiz.
- Logo, as três operações que compõem o algoritmo são:
  - **Visitar** a raiz, **percorrer** a subárvore da esquerda e **percorrer** a subárvore da direita.
- Questão: definir a ordem em que estas operações são realizadas de acordo com o problema.

Percurso Árvores Binárias

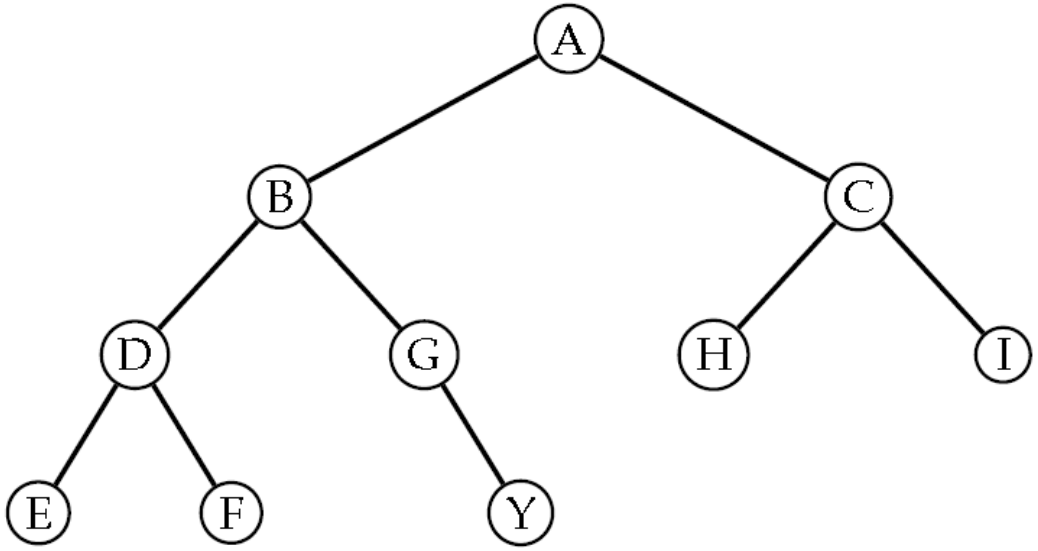
- Algoritmo percurso em profundidade.
  - Seguir tanto quanto possível à esquerda (ou direita);
  - Então mover para trás até a primeira encruzilhada;
  - Seguir um passo para direita (ou esquerda);
  - Novamente, seguir tanto quanto possível para a esquerda (ou direita);
  - Repetir o processo até que todos os nós tenham sido visitados.

Percurso Árvores Binárias

- Envolvem 3 tarefas:
  - V – Visitar um nó;
  - L – Percorrer subárvore esquerda (*left*);
  - R – Percorrer subárvore direita (*right*).
- 3! possibilidades: **VLR**, **LVR**, **LRV**, VRL, RVL, RLV.
  - percurso em pré-ordem (ou pré-fixado): VLR
  - percurso em ordem simétrica (central ou in-ordem): LVR
  - percurso em pós-ordem (ou pós-fixado): LRV

Percurso Árvores Binárias

- Percurso em pré-ordem (pré-fixado)
- Passos:
  - Visitar a raiz;
  - Percorrer sua subárvore esquerda, em pré-ordem;
  - Percorrer sua subárvore direita, em pré-ordem.



- Exemplo: A, B, D, E, F, G, Y, C, H, I

Percurso Árvores Binárias

- Percurso em ordem e pós-ordem
  - Percorrer sua subárvore esquerda, em in-ordem;
  - Visitar a raiz;
  - Percorrer sua subárvore direita, em in-ordem.
- Pós-ordem:
  - Percorrer sua subárvore esquerda, em in-ordem;
  - Percorrer sua subárvore direita, em in-ordem;
  - Visitar a raiz;
- Observe a natureza recursiva dos algoritmos de percurso.
- Exercício: execute os percursos em ordem e pós-ordem na árvore binária do exemplo anterior. Qual a sequência de visita aos nós?

Árvores Binárias

- Agora, a função imprime do TAD `ArvBin` pode ser implementada utilizando, por exemplo, um percurso pré-ordem.

```
void ArvBin::imprime()
{
   auxImprime(raiz);
}

void ArvBin:: auxImprime(NoArv *p)
{
   if(p != NULL)
   {
      cout << p->getInfo() << endl;
      auxImprime(p->getEsq());
      auxImprime(p->getDir());
   }
}
```

Árvores Binárias

- Mudanças na classe `ArvBin`:

```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o no raiz da árvore
      void auxImprime(NoArv *p);

      //etc ......

   public:

      //etc ......

};
```

- A função `auxImprime(NoArv*)` é que realiza toda a tarefa de imprimir o conteúdo da árvore binária.

Árvores Binárias

```
void ArvBin:: auxImprime(NoArv *p)
{
   if(p != NULL)
   {
      auxImprime(p->getEsq());
      cout << p->getInfo() << endl;
      auxImprime(p->getDir());
   }
}
```

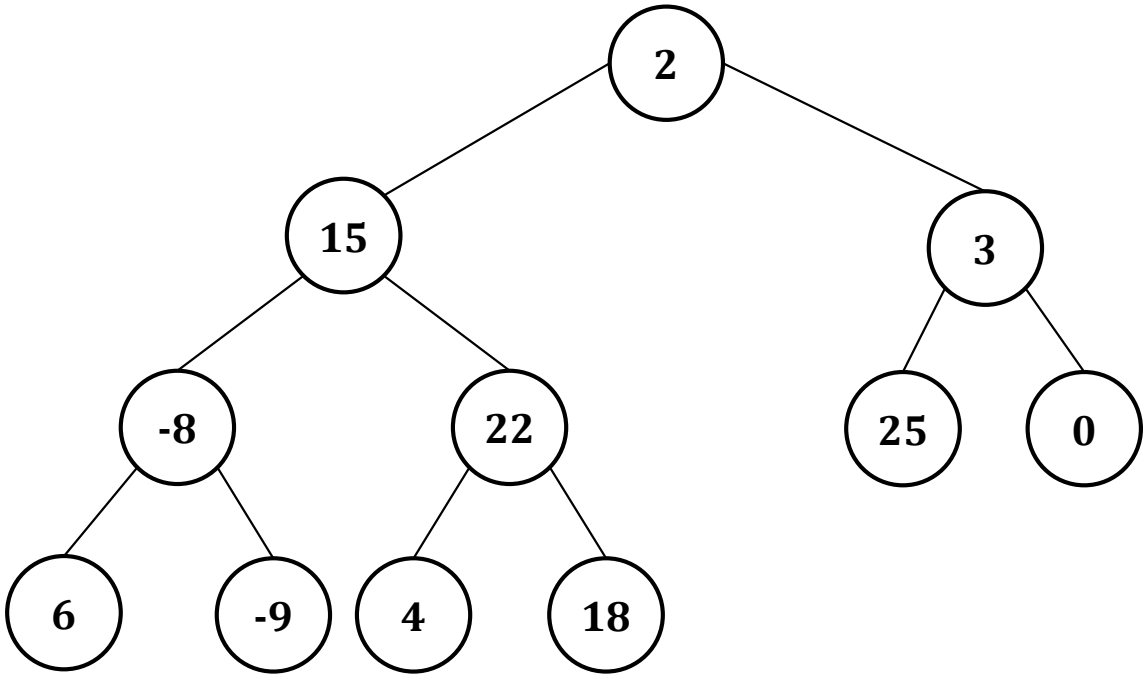
Percurso em ordem

```
void ArvBin:: auxImprime(NoArv *p)
{
   if(p != NULL)
   {
      auxImprime(p->getEsq());
      auxImprime(p->getDir());
      cout << p->getInfo() << endl;
   }
}
```

Percurso pós ordem

Árvores Binárias

- Exemplo – Apresentar a sequência de nós visitados ao percorrer em profundidade a árvore binária a seguir em pré-ordem, em pós-ordem e em ordem:



Árvores Binárias

- MI do TAD `ArvBin`
  - Operação busca:

```
bool ArvBin::busca(int val)
{
    return auxBusca(raiz, val);
}
```

- Observação: como o parâmetro raiz da operação `auxBusca` é um ponteiro, essa operação deve ser necessariamente privada. O TAD `ArvBin` fica:

Árvores Binárias

- TAD `ArvBin` (árvore binária de inteiros)

```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o nó raiz da árvore
      void auxImprime(NoArv *p);
      bool auxBusca(NoArv *p, int val);
   public:
      ArvBin();
      int getRaiz();
      // cria novo nó como raiz das sub-árvores à
      //esquerda (sae) e à direita (sad)
      void cria(int val, ArvBin *sae, ArvBin *sad);
      bool vazia(); // verifica se a árvore está vazia
      bool busca(int val);
      void imprime();
      ~ArvBin();
};
```

Árvores Binárias

- MI do TAD `ArvBin`
  - A operação `auxBusca` procura a chave na árvore seguindo um percurso pré-ordem.

```
bool ArvBin::auxBusca(NoArv *p, int ch)
{
    if (p == NULL)
        return false;
    else if (p->getInfo() == ch)
        return true;
    else if (auxBusca(p->getEsq(), ch))
        return true;
    else
        return auxBusca(p->getDir(), ch)
}
```

Árvores Binárias

- Destrutor de objetos da classe `ArvBin`

```
ArvBin::~ArvBin(){
    raiz = libera(raiz);
}
```

- Obs: como o parâmetro raiz da função `libera` é um ponteiro, essa operação deve ser necessariamente privada.

```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o nó raiz da árvore
      bool auxBusca(NoArv *p, int val);
      void auxImprime(NoArv *p);
      NoArv* libera(NoArv *p);
   public:
      //etc .......}
;
```

Árvores Binárias

- TAD `ArvBin` (árvore binária de inteiros)

```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o nó raiz da árvore
      bool auxBusca(NoArv *p, int val);
      void auxImprime(NoArv *p);
      NoArv* libera(NoArv *p);
   public:
      //etc .......
};
```

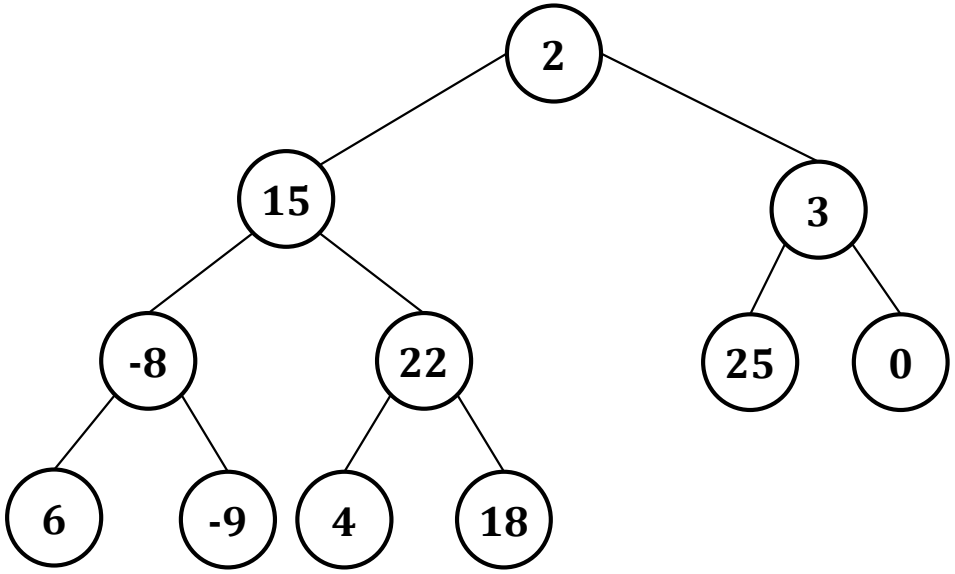
Árvores Binárias

- TAD `ArvBin` (árvore binária de inteiros)
  - Operação `libera`:

```
NoArv* ArvBin::libera(NoArv *p)
{
    if (p != NULL)
    {
        p->setEsq( libera(p->getEsq()));
        p->setDir( libera(p->getDir()));
        delete p;
        p = NULL;
    }
    return NULL;
}
```

Árvores Binárias

- **Percurso em largura (extensão).** Corresponde a visitar cada nó começando no nível mais alto (ou mais baixo) e movendo para baixo (ou para cima) nível a nível, visitando nós em cada nível da esquerda para a direita (ou da direita para a esquerda).



- Ex: 2, 15, 3, -8, 22, 25, 0, 6, -9, 4 e 18

Árvores Binárias

- Usa-se uma fila para a sua implementação. Por exemplo, para um percurso em largura de cima para baixo, da esquerda para a direita:
  - Entra com o nó raiz na fila;
  - Nó no início da fila é visitado e sai da fila;
  - Quando um nó é visitado, seus filhos (se houverem) são colocados no final da fila.
- Para um nó j no nível n, seus filhos estão no nível $n + 1$. Após a visita do nó j, seus filhos são colocados no final da fila. Portanto, eles só serão visitados depois que todos os nós do nível n forem visitados. Assim a restrição de que todos os nós no nível n precisam ser visitados antes de visitar quaisquer nós no nível $n + 1$ será satisfeita.

Árvores Binárias

- Algoritmo do percurso em largura:

```
Se a árvore não está vazia
{
   Coloca nó raiz na fila;
   Enquanto a fila não está vazia
   {
      Visita o nó do início da fila;
      Retira o nó da fila;
      Se tem filho à esquerda
         Coloca o nó na fila;
      Se tem filho à direita
         Coloca o nó na fila;
   }
}
```

Árvores Binárias

- **Exercício**: desenvolver uma operação do TAD `arvBin` para calcular e retornar a altura de uma árvore binária. Utilizar uma função auxiliar privada que seja recursiva.

```
class ArvBin
{
   private:
      NoArv *raiz; // ponteiro para o nó raiz da árvore
      int auxAltura(NoArv *p);
      //etc ...
   public:
      int altura() {auxAltura(raiz);};
      //etc .......
};
```

Árvores Binárias

- Operador `auxAltura`

```
int TArvBin::AuxAltura(No *p)
{
   //he e hd armazenam, respectivamente, as alturas
   //das sub-árvores da esquerda e da direita de p
   int he, hd;
   if (p == NULL)
      return -1;
   else
   {
      he = AuxAltura(p->getEsq());
      hd = AuxAltura(p->getDir());
      return 1 + (he > hd ? he : hd);
   }
}
```

Árvores Binárias

- Operador `auxAltura` (versão 2)

```
int TArvBin::AuxAltura(No *p)
{
   //he e hd armazenam, respectivamente, as alturas
   //das sub-árvores da esquerda e da direita de p
   int he, hd;
   if (p == NULL)
      return -1;
   else
   {
      he = AuxAltura(p->getEsq());
      hd = AuxAltura(p->getDir());
      return 1 + maior(he, hd); //maior de 2 inteiros
   }
}
```

Árvores Binárias

- **Exercício**: desenvolver uma operação para realizar o percurso em largura em uma árvore binária de números inteiros e imprimir as informações de todos os seus nós da **esquerda para a direita**. O algoritmo que realiza tal percurso já foi descrito anteriormente.
- Dica: será preciso utilizar uma fila (encadeada) cujos elementos são ponteiros para os nós da árvore binária. Ver definições das classes `FilaNoArv` e `No` a seguir.

Árvores Binárias

- Definição do nó usado na fila de nós de árvore binária.

```
class No
{
   private:
      NoArv *info; //ponteiro para um nó da árvore
      No *prox;//ponteiro para o próximo nó da fila
   public:
      No(){};
      ~No(){};
      void setInfo(NoArv *p)    {info = p;};
      void setProx(No *p)       {prox = p;};
      NoArv* getInfo()          {return info;};
      No* getProx()             {return prox;};
};
```

Árvores Binárias

- Definição da classe para a fila de nós de árvore binária.

```
class FilaEncad
{
 private:
   No *inicio, *fim;//ponteiros para os extremos
 public:
   FilaEncad();
   ~FilaEncad();
   NoArv* getInicio();
   void enfileira(NoArv* p);// insere no fim da fila
   NoArv* desenfileira(); //elimina do início da fila
   bool Vazia(); // verifica se fila está vazia
};
```

Árvores Binárias

- Operação pública percurso em largura para a AB.

```
void ArvBin::percursoLargura()
{
   NoArv *p = raiz, *q; //ponteiros para nó da árvore
   FilaEncad fila;
   if (p != NULL)//verifica se árvore está vazia
   {
      fila.Enfileira(p);//coloca nó raiz na fila
      while (!fila.Vazia()) //há nós na fila?
      {
         q = fila.desenfileira(); // tira nó da fila
         cout << q->getInfo()<< endl;
         if(q->getEsq() != NULL)
             fila.enfileira(q->getEsq());
         if(q->getDir() != NULL)
             fila.enfileira(q->getDir());
      }
   }
}
```

Exercícios - Árvores Binárias

- Nos exercícios a seguir, dada uma estrutura (um nome) **a** representando uma árvore binária, define-se:
  a) Sae(**a**): como uma árvore representando a subárvore à esquerda de **a**;
  b) Sad(**a**): como uma árvore representando a subárvore à direita de **a**;
  c) Val(**a**): como o valor da raiz da árvore **a**;
  d) Vazia(**a**): verifica se **a** está vazia. Verdadeiro se sim e falso, caso contrário.
  e) EhFolha(**a**): se o nó **a** é uma folha. Ou seja, **a** não tem filhos, retorna verdadeiro e falso caso contrário.

Exercícios - Árvores Binárias

1. Escrever uma operação para contar e retornar o número de nós de uma árvore binária.

$$nNos(a) = \begin{cases} 0, \text{ se } Vazia(a) \\ 1 + nNos(Sae(a)) + nNos(Sad(a)), \text{ caso contrário} \end{cases}$$

2. Escrever uma operação para contar e retornar o número de folhas de uma árvore binária.

$$nFolhas(a) = \begin{cases} 0, \text{ se } Vazia(a) \\ 1, \text{ se } EhFolha(a) \\ nFolhas(Sae(a)) + nFolhas(Sad(a)), \\ \qquad \text{caso contrário} \end{cases}$$

Exercícios - Árvores Binárias

3. Escrever uma operação para verificar se uma árvore binária (AB) é cheia (retornar **false**) ou não (retornar **true**). O número de nós n de uma AB cheia é dado por $n = 2^{h+1} - 1$, onde h é altura da AB. Também, pode-se considerar que uma AB é cheia quando o número de nós do último nível (número de folhas) da árvore binária é 2^h.
4. Escrever uma operação para excluir todas as folhas de uma árvore binária, deixando a raiz e os nós intermediários nos respectivos lugares. Dica: usar o percurso em pré-ordem.

Exercícios - Árvores Binárias

5. Reescrever a operação de percurso em pré-ordem usando uma pilha em vez da recursão.
6. Reescrever o procedimento que realiza o percurso em largura usando uma pilha em vez de uma fila. (observar que o resultado dos exercícios 5 e 6 deve ser o mesmo. Por quê?).
7. No exercício 3 (verificar se uma AB é cheia), a AB é percorrida 2 vezes: uma para calcular o número de folhas (ou nós) e outra para calcular a sua altura. Desenvolver a mesma operação porém percorrendo a árvore apenas uma única vez.

Exercícios - Árvores Binárias

8. Desenvolver uma operação para imprimir o nível seguido do valor do nó de uma AB.
9. Desenvolver uma operação para, dado um inteiro $k \geq 0$, calcular o número de nós no nível k de uma AB.
10. Escrever uma operação para verificar se uma árvore binária (AB) é completa. Retornar **true** (se sim) ou **false** (se não**)**. Dica: usar a operação do exercício anterior.
11. Escrever uma operação para verificar se uma árvore binária (AB) é estritamente binária. Retornar **true** (se sim) ou **false** (se não**)**.