Árvore

Conceitos Gerais e Árvore Binária

Maria Adriana Vidigal de Lima

FACOM - UFU

Sumário

Árvore: Definição e Terminologia

Uma **árvore** é uma coleção finita de *n* nós.

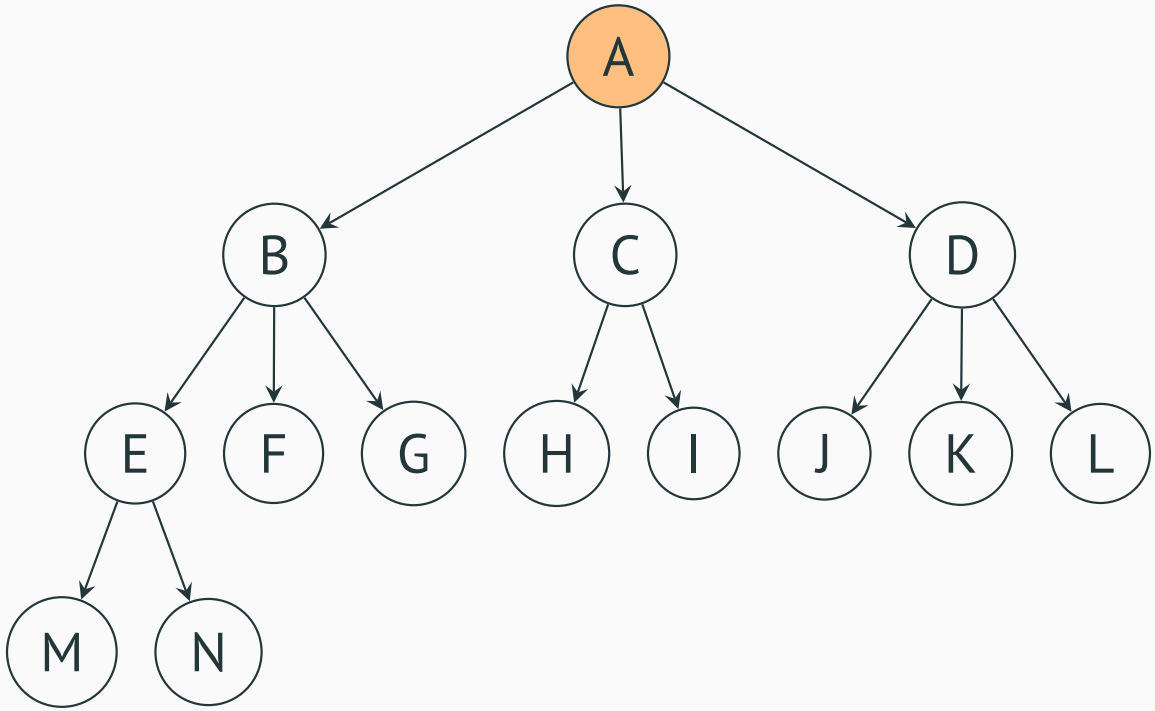
- Se n = 0, a árvore é nula.
- Se n > 0, então a árvore apresenta as seguintes características:
  - existe um nó especial denominado **raiz**;
  - os demais são particionados em $T_1, T_2, \ldots, T_k$ que representam estruturas disjuntas de árvores;
  - as estruturas $T_1, T_2, \ldots, T_k$ são chamadas de subárvores, o que caracteriza uma definição recursiva.

A exigência que $T_1, T_2, \ldots, T_k$ sejam coleções disjuntas, garante que um mesmo nó não aparecerá em mais de uma subárvore ao mesmo tempo.

Árvore: Definição e Representação

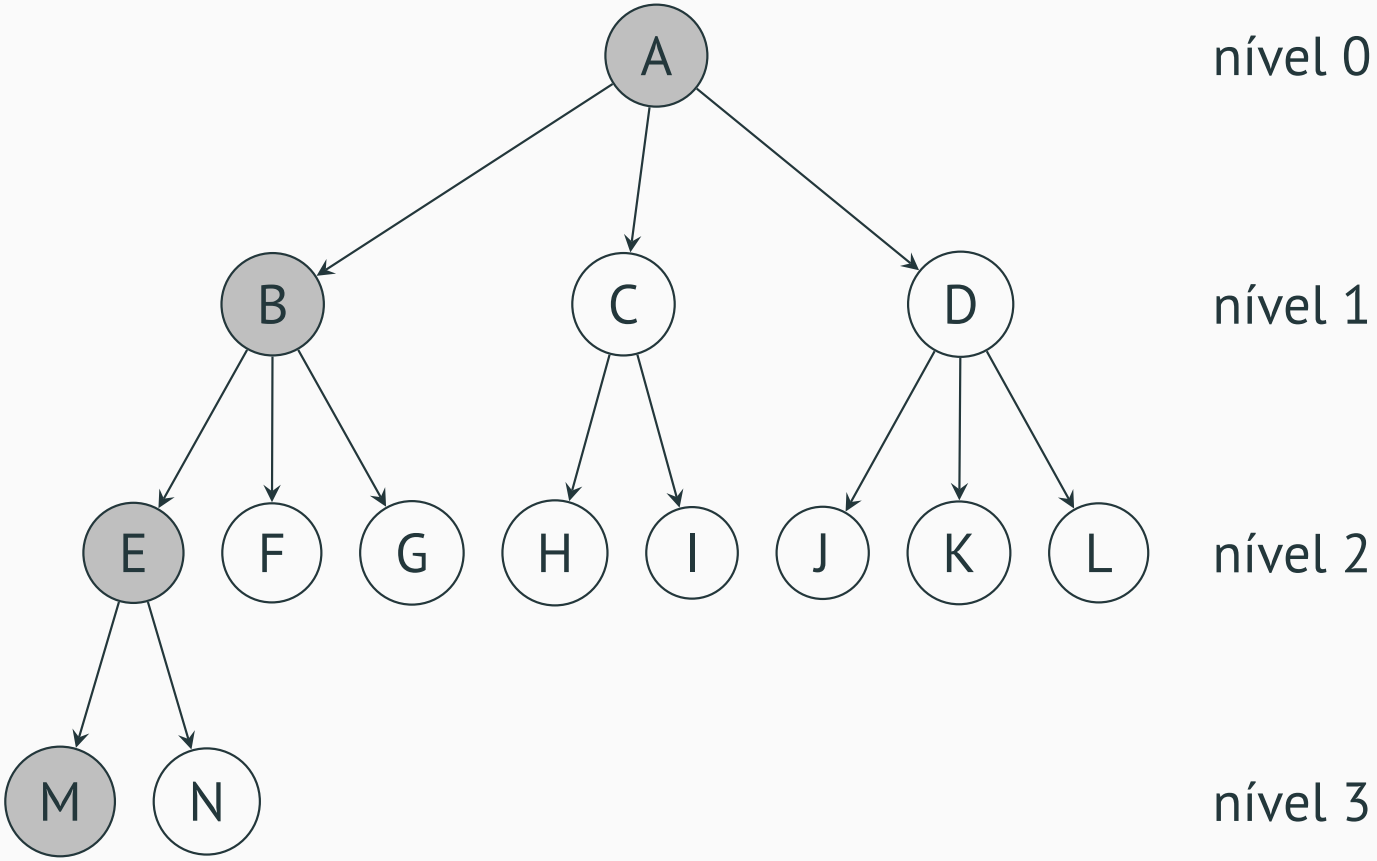
Uma árvore é um grafo sem ciclos.

- A raiz da árvore é o nó de início **A**.
- O nó **A** possui três subárvores cujas raízes são **B**, **C** e **D**.
- Os **nós internos** da árvore são os nós com filhos.
- As **folhas** ou nós externos da árvore são os nós sem filhos.



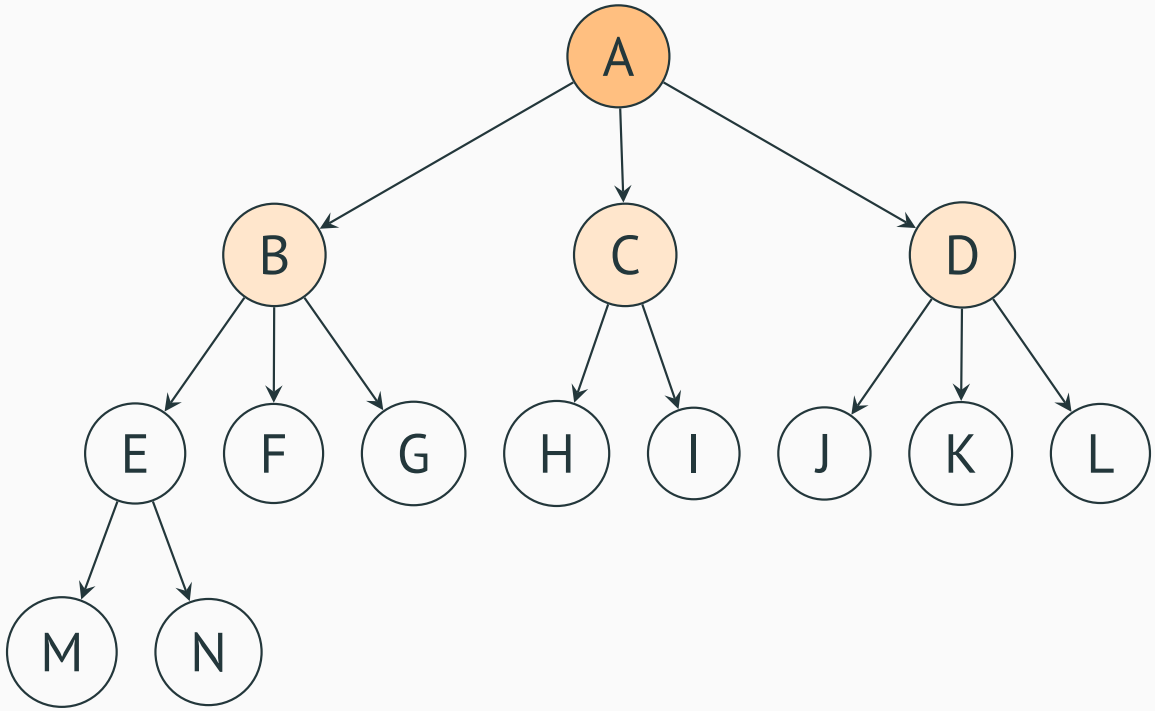
Árvore: Definição e Representação

- Todos os nós são acessíveis a partir da raiz.
- O nó raiz de uma árvore encontra-se no nível 0 (alternativamente, pode-se considerar nível 1)



Árvore: Definição e Representação

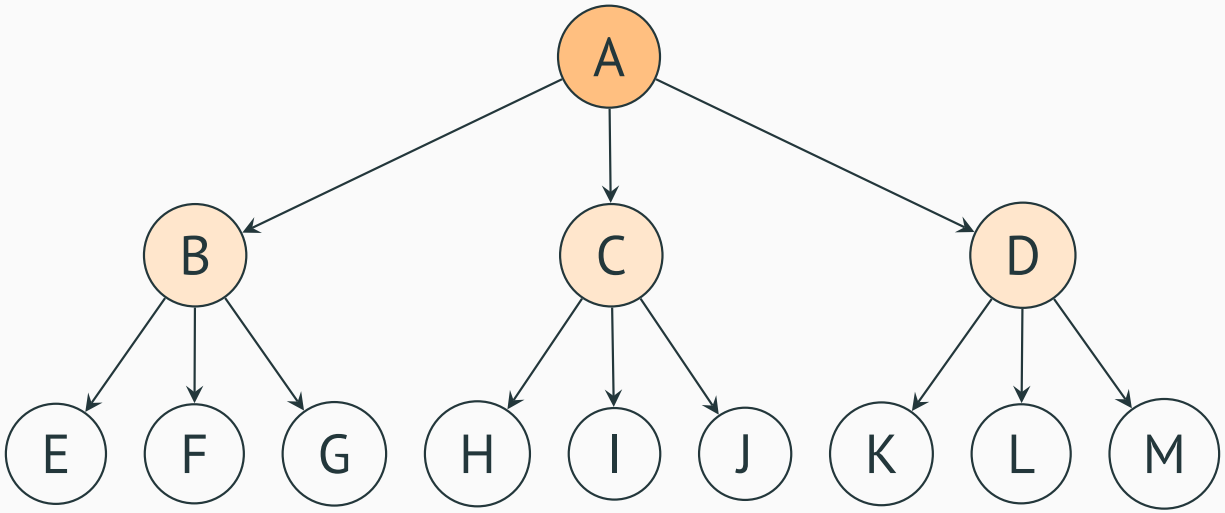
- O número de subárvores de um nó denomina-se **grau de um nó**.
- O nó A tem grau 3, o nó C tem grau 2 e o nó F tem grau 0 (zero), ou seja, não possui filhos.
- O **grau de uma árvore** (aridade), é definido como sendo igual ao máximo grau entre todos os seus nós. Neste caso, o grau da árvore é 3, pois nenhum nó tem mais de 3 filhos (subárvores).



Árvore Completa

Uma árvore de grau g é uma **árvore completa** (cheia) se:

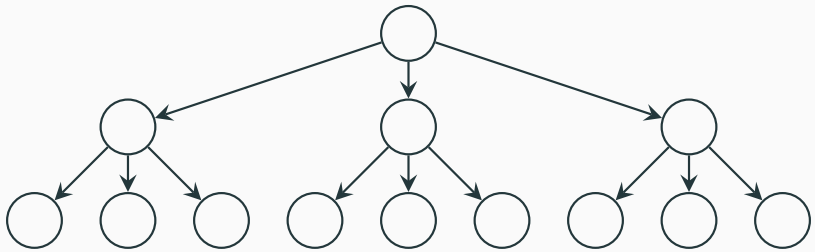
- Todos os nós tem exatamente g filhos, exceto as folhas;
- Todas as folhas estão na mesma altura.



A árvore de grau g=3 acima é completa.

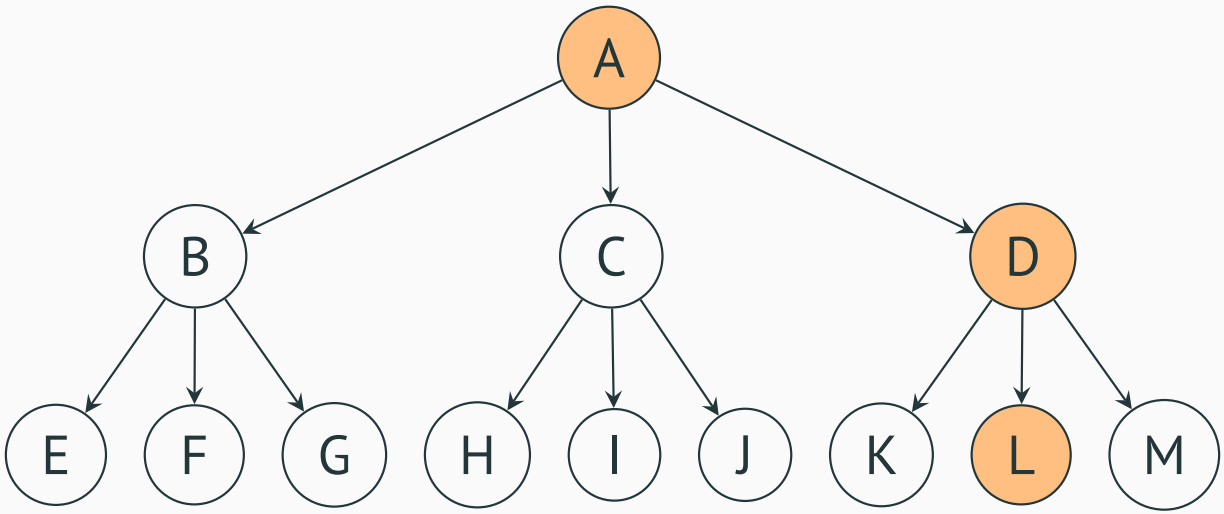
Árvore: Número máximo de nós

- O número máximo de nós em árvore de altura h é atingido quando a árvore está completa.
- Para uma árvore de grau g:
  - Nível 0 contém g^0 nós (um nó raiz)
  - Nível 1 contém g^1 nós (descendentes da raiz)
  - Nível 2 contém g^2 nós descendentes
  - ...
  - Nível k contém g^k nós descendentes



Árvore: Relação pai-filho e Caminho

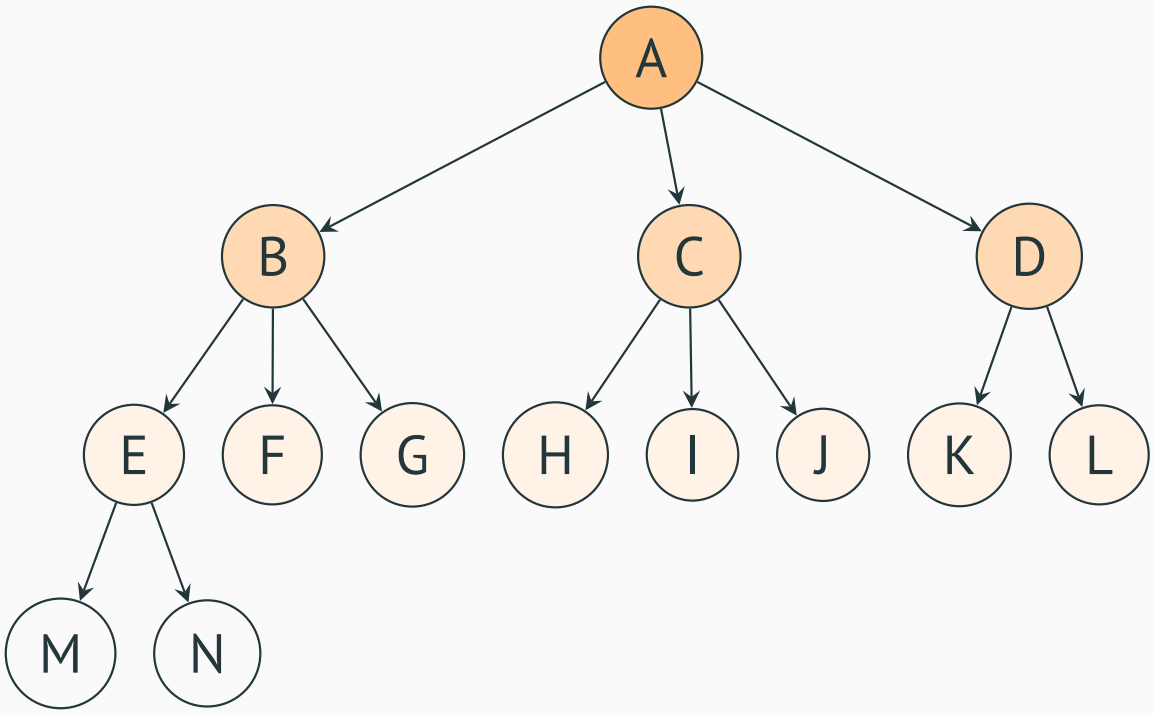
- As raízes das subárvores de um nó X (nó **pai**) são os **filhos** de X;
- Os filhos (descendentes) de um mesmo nó pai (antecessor) são denominados **irmãos**;
- Uma sequência de nós distintos $v_1, v_2, \ldots, v_k$ tal que sempre existe a relação v_i *é pai de* v_{i+1}, para $1 \leq i < k$ é denominada **caminho** entre v_1 e v_k.



Exemplo: A, D, L é um caminho entre A e L, formado pela sequência (A,D), (D,L). O comprimento do caminho entre A e L é 2.

Árvore: Altura de um nó

- A **altura de um nó** é o número de arcos (ligações) no maior caminho desde o nó até um de seus descendentes;
- Os nós folhas tem altura 0 (zero);
- A **altura de uma árvore** corresponde à altura do nó raiz.



Exemplo:

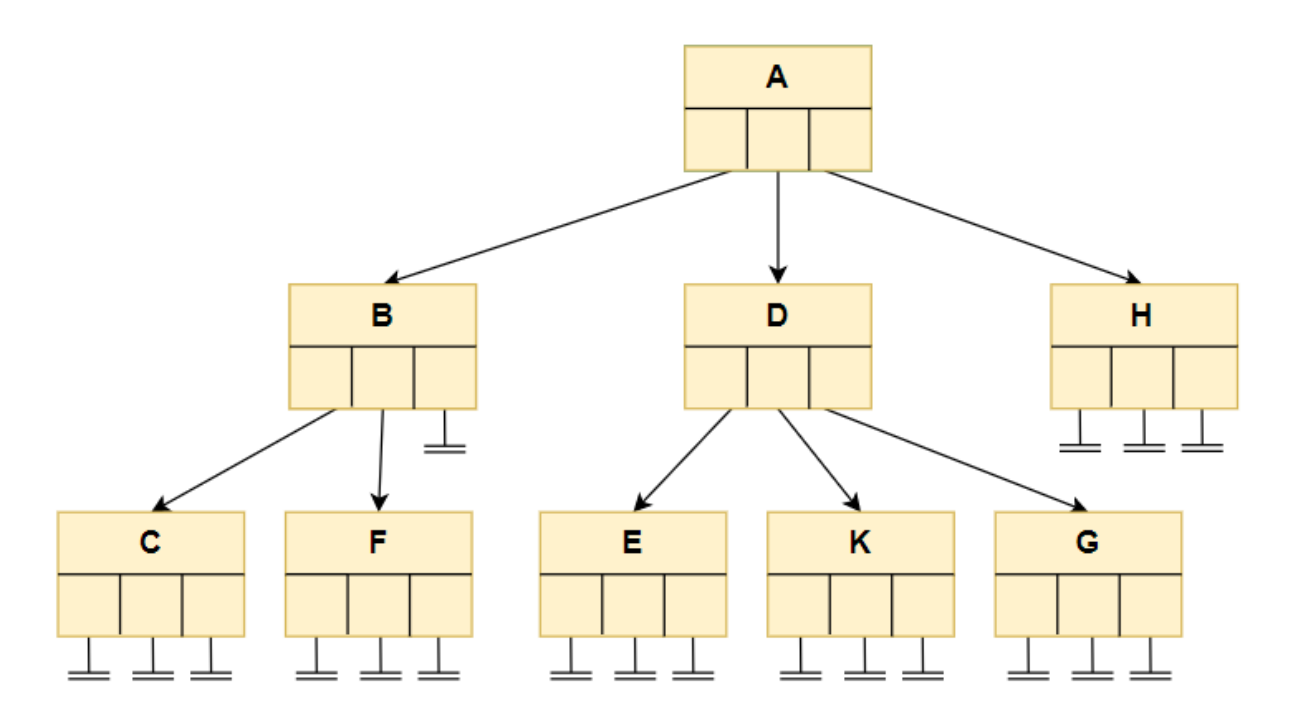
- F, G, H, I, J, K, L, M, N têm altura 0
- E, C, D têm altura 1
- B tem altura 2
- A tem altura 3

Implementação de Árvores

Árvores podem ser implementadas utilizando-se listas encadeadas em que:

- Cada nó possui um campo para informação e uma lista de ponteiros para os seus nós filhos, de acordo com a quantidade de filhos.

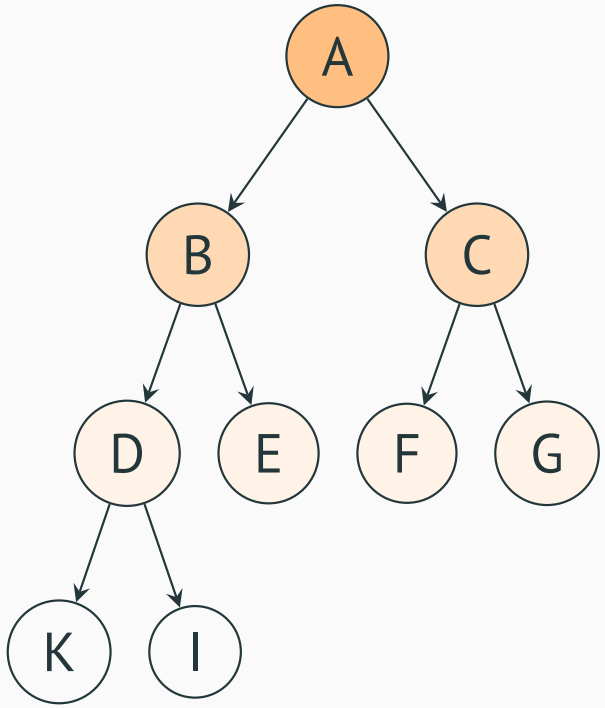
Para uma **árvore de grau 3**, tem-se:



Árvores Binárias

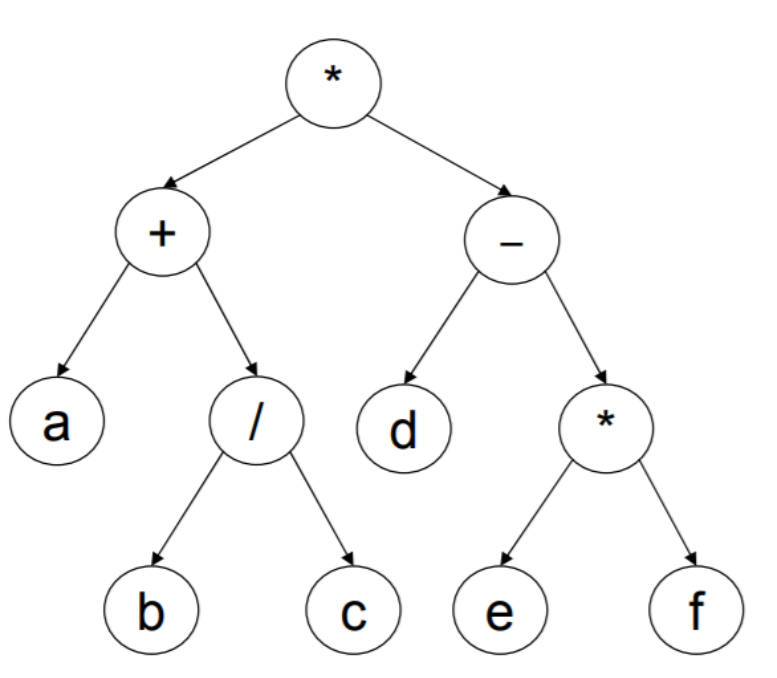
Uma **árvore binária** é uma árvore de grau 2 (nenhum nó tem mais que dois filhos) com as seguintes características:

- nó especial raiz;
- os demais nós são divididos entre T_1 e T_2, subárvores binárias disjuntas;
- T_1 é denominada subárvore esquerda e T_2 subárvore direita.



Árvores Binárias

Pode-se representar uma **expressão aritmética** (com operadores binários) por meio de uma árvore binária, em que cada operador é um nó da árvore e seus dois operandos são representados como subárvores.



A árvore representa a expressão: (a+b/c)*(d-e*f)

Representação: Árvores Binárias em Linguagem C

- Representação de uma árvore: através de um ponteiro para o nó raiz
- Representação de um nó da árvore:
  - a informação propriamente dita (exemplo: um caractere)
  - dois ponteiros para as sub-árvores, à esquerda e à direita

```
1 struct noArv {
2     char info;
3     struct noArv* esq;
4     struct noArv* dir;
5 };
```

Representação: Árvores Binárias em Linguagem C

Interface para o Tipo de Dados Abstrato: Árvore Binária

```
1  struct noArv {
2     char info;
3     struct noArv* esq;
4     struct noArv* dir;
5  };
6
7  typedef struct noArv NoArv;
8
9  NoArv* arv_cria_vazia (void);
10 NoArv* arv_cria (char c, NoArv* e, NoArv* d);
11 NoArv* arv_libera (NoArv* a);
12 int arv_vazia (NoArv* a);
13 int arv_pertence (NoArv* a, char c);
14 void arv_imprime_pre (Arv* a);
15 void arv_imprime_pos (Arv* a);
16 void arv_imprime_em (Arv* a);
17 int arv_altura (Arv * a);
```

Criação: Árvores Binárias em Linguagem C

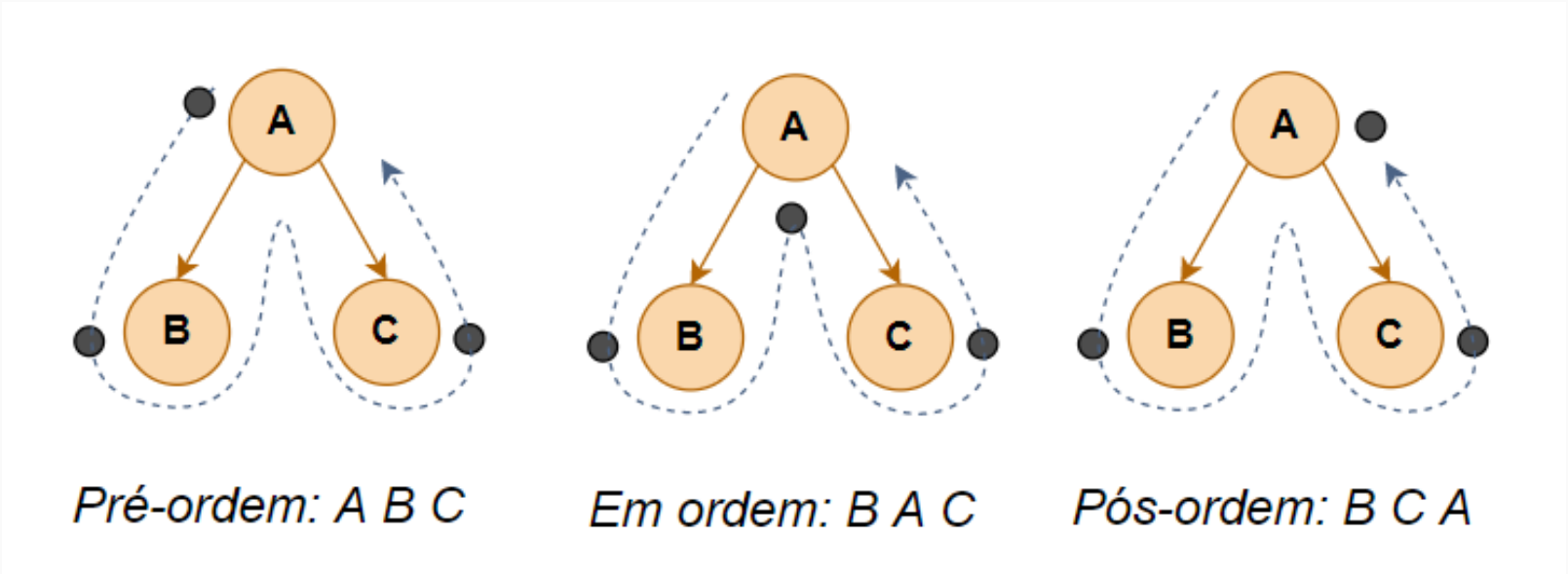
Funções para a **criação** de uma árvore binária:

```
1  Arv* arv_cria_vazia (void){
2     return NULL;
3  }
4
5  Arv* arv_cria (char c, Arv* subesq, Arv* subdir){
6     Arv* p=(Arv*) malloc (sizeof(Arv));
7     p->info = c;
8     p->esq = subesq;
9     p->dir = subdir;
10    return p;
11 }
```

Percursos: Árvores Binárias em Linguagem C

Percursos possíveis:

- **Pré-ordem**: visita o nó raiz, subárvore esquerda e subárvore direita
- **Em ordem**: visita a subárvore esquerda, o nó raiz e subárvore direita
- **Pós-ordem**: visita a subárvore esquerda, subárvore direita e o nó raiz.



Percursos: Árvores Binárias em Linguagem C

Impressão **pré-ordem**:

```
1  void arv_imprime_pre (Arv* a){
2    if (!arv_vazia(a)){
3        printf(“%c”, a->info);          /* mostra raiz */
4        arv_imprime_pre(a->esq);     /* mostra subárvore esquerda */
5        arv_imprime_pre(a->dir);     /* mostra subárvore direita */
6    }
7  }
```

Impressão **em ordem**:

```
1  void arv_imprime_em (Arv* a){
2    if (!arv_vazia(a)){
3        arv_imprime_em(a->esq);     /* mostra subárvore esquerda */
4        printf(“%c”, a->info);          /* mostra raiz */
5        arv_imprime_em(a->dir);     /* mostra subárvore direita */
6    }
7  }
```

Pós-ordem: Árvores Binárias em Linguagem C

Impressão **pós-ordem**:

```
1  void arv_imprime_pos (Arv* a){
2    if (!arv_vazia(a)){
3        arv_imprime_pos(a->esq);       /* mostra subárvore esquerda */
4        arv_imprime_pos(a->dir);       /* mostra subárvore direita */
5        printf(‘‘%c’’, a->info);       /* mostra raiz */
6    }
7  }
```

Exemplos de percurso com números inteiros:

- Pré-ordem:

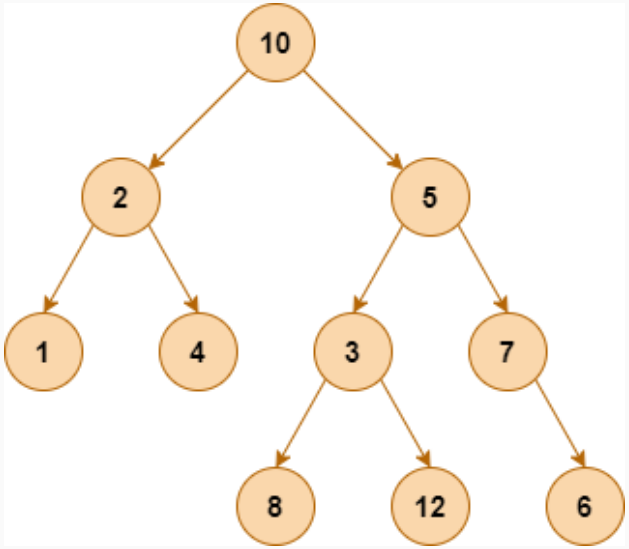
10 2 1 4 5 3 8 12 7 6

- Pós-ordem:

1 4 2 8 12 3 6 7 5 10

- Em ordem:

1 2 4 10 8 3 12 5 6 7



Altura: Árvores Binárias em Linguagem C

Função **altura**:

```
1  int max (int a, int b){
2    if (a>b) return a;
3    else return b;
4  }
5  int arv_altura (Arv * a){
6     if (arv_vazia(a))
7         return -1;        // raiz está no nível 0
8     else
9         return 1 + max(altura(a->esq), altura(a->dir));
10 }
```

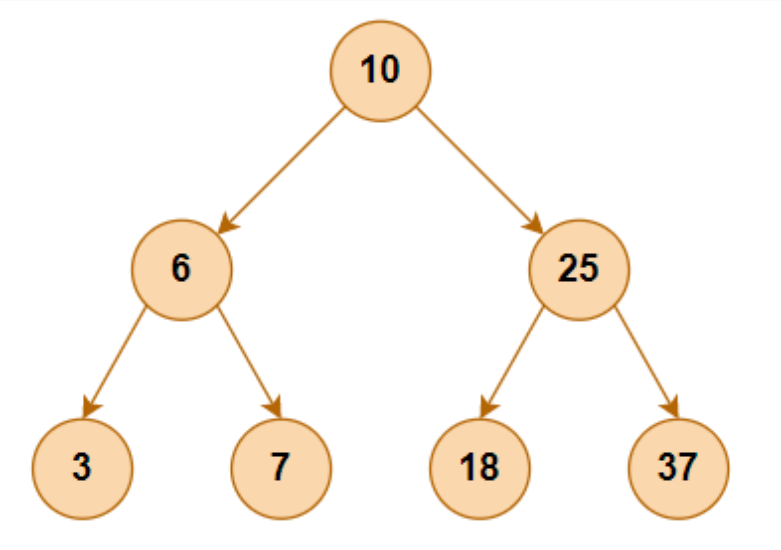
Função **pertence**:

```
1  int arv_pertence (Arv* a, char c){
2    if (arv_vazia(a))
3       return 0;          // árvore vazia: não encontrou
4    else
5       return (a->info==c || arv_pertence(a->esq,c) || arv_pertence(a->dir,c));
6  }
```

Árvore Binária de Busca

Uma árvore binária, cuja raiz armazena o elemento E, é denominada **árvore binária de busca** se:

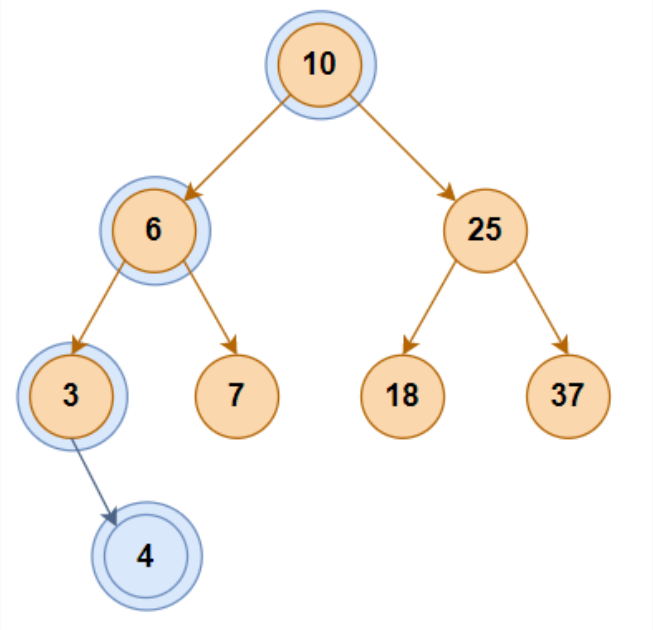
- os elementos da subárvore esquerda são **menores** que E;
- os elementos da subárvore direita são **maiores** que E;
- as subárvores esquerda e direita são, igualmente, árvores binárias de busca.



Inserção em Árvore Binária de Busca

A inserção numa árvore de busca binária é muito eficiente, pois **os novos elementos inseridos entram sempre na condição de folhas**:

- Para inserir o elemento 4, começa-se pelo nó raiz. Como 4 é menor que 10, toma-se a subárvore da esquerda.
- Comparando-se 4 com a nova raiz (6), tem-se que 4 é menor, então toma-se novamente a subárvore da esquerda, cuja raiz é o elemento 3.
- Neste ponto, tem-se que 4 é maior que 3, e então toma-se a subárvore da direita, que é nula.
- Ao encontrar uma subárvore nula, o novo nó deve ser alocado, entrando como raiz desta subárvore.



Inserção em Árvore Binária de Busca

Função **insere** para Árvore Binária de Busca, considerando a inserção de números inteiros:

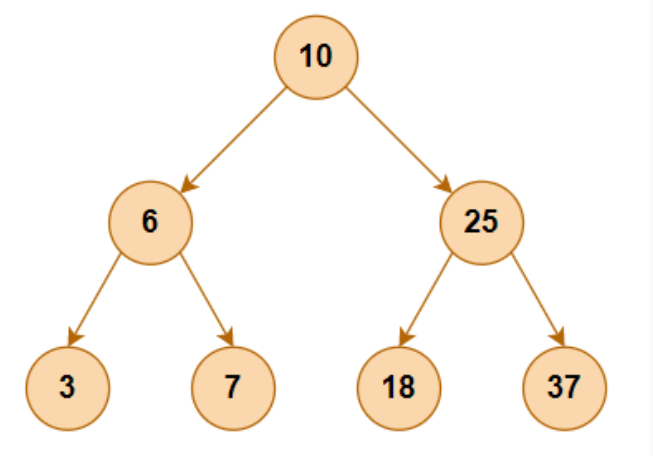
```
1  Arv* arv_insere (int c, Arv* raiz){
2     Arv* p;
3     if (arv_vazia(raiz)){
4        p = (Arv*) malloc (sizeof(Arv));
5        p->info = c;
6        p->esq = NULL;
7        p->dir = NULL;
8        return p;
9     }
10    else
11       if (c < raiz->info)
12          raiz->esq = arv_insere(c, raiz->esq);
13       else
14          raiz->dir = arv_insere(c, raiz->dir);
15    return raiz;
16 }
```

Percurso em Ordem numa Árvore Binária de Busca

Função **imprime em ordem** para Árvore Binária de Busca

```
1  void arv_imprime_em (Arv* a){
2    if (!arv_vazia(a)){
3        arv_imprime_em(a->esq);       /* mostra subárvore esquerda */
4        printf("%d", a->info);        /* mostra raiz */
5        arv_imprime_em(a->dir);       /* mostra subárvore direita */
6    }
7  }
```

O percurso em ordem numa árvore binária de busca retorna os elementos em ordem crescente:
3 6 7 10 18 25 37

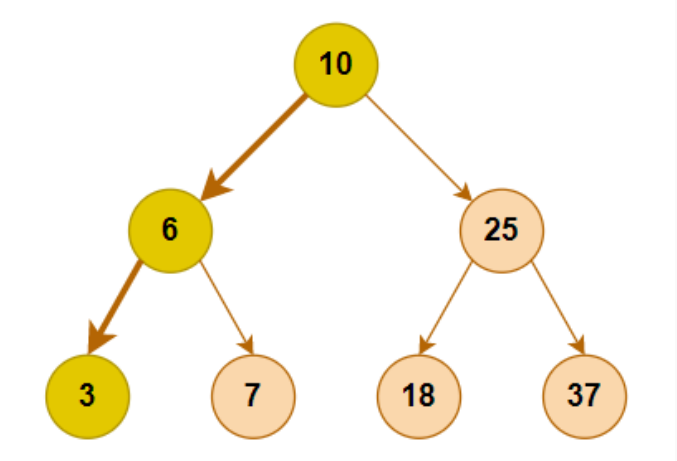


Menor elemento: Árvore Binária de Busca

Função para encontrar o **menor elemento** numa Árvore Binária de Busca:

```
1  Arv* arv_menor(Arv *a){
2     if (arv_vazia(a)) return NULL;
3     else
4        if (arv_vazia(a->esq))
5           return a;
6        else
7           return arv_menor(a->esq);
8  }
```

Para encontrar o menor elemento executa-se um percurso sempre à esquerda:
Menor elemento: 3

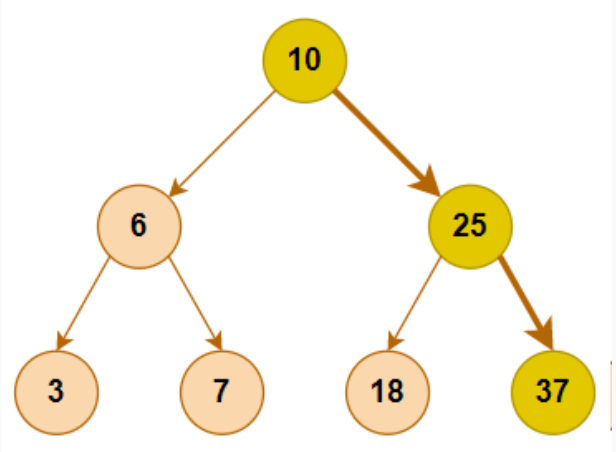


Maior elemento: Árvore Binária de Busca

Função (versão iterativa) para encontrar o **maior elemento** numa Árvore Binária de Busca:

```
1  Arv* arv_maior(Arv *a){
2  Arv* aux;
3     if (arv_vazia(a)) return NULL;
4     else{
5        aux = a;
6        while (!arv_vazia(aux->dir))
7            aux = aux->dir;
8     }
9     return aux;
10 }
```

Para encontrar o maior elemento executa-se um percurso sempre à direita:
Maior elemento: 37



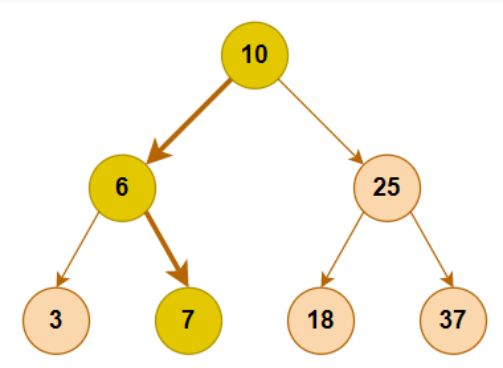
Busca de um elemento: Árvore Binária de Busca

Função para verificar a existência de um elemento numa Árvore Binária de Busca:

```
1  int arv_busca (Arv* a, int c){
2    if (arv_vazia(a)) return -1;
3    else
4        if (c == a->info)
5            return 1;
6        else
7            if (c < a->info)
8                return arv_busca(a->esq,c);
9            else
10               return arv_busca(a->dir,c);
11 }
```

Para encontrar um elemento qualquer, verifica-se se o mesmo é igual ao elemento raiz (melhor caso). Caso seja menor, busca-se na esquerda, caso contrário na direita.

Elemento 7: Encontrado



Remover elemento: Árvore Binária de Busca

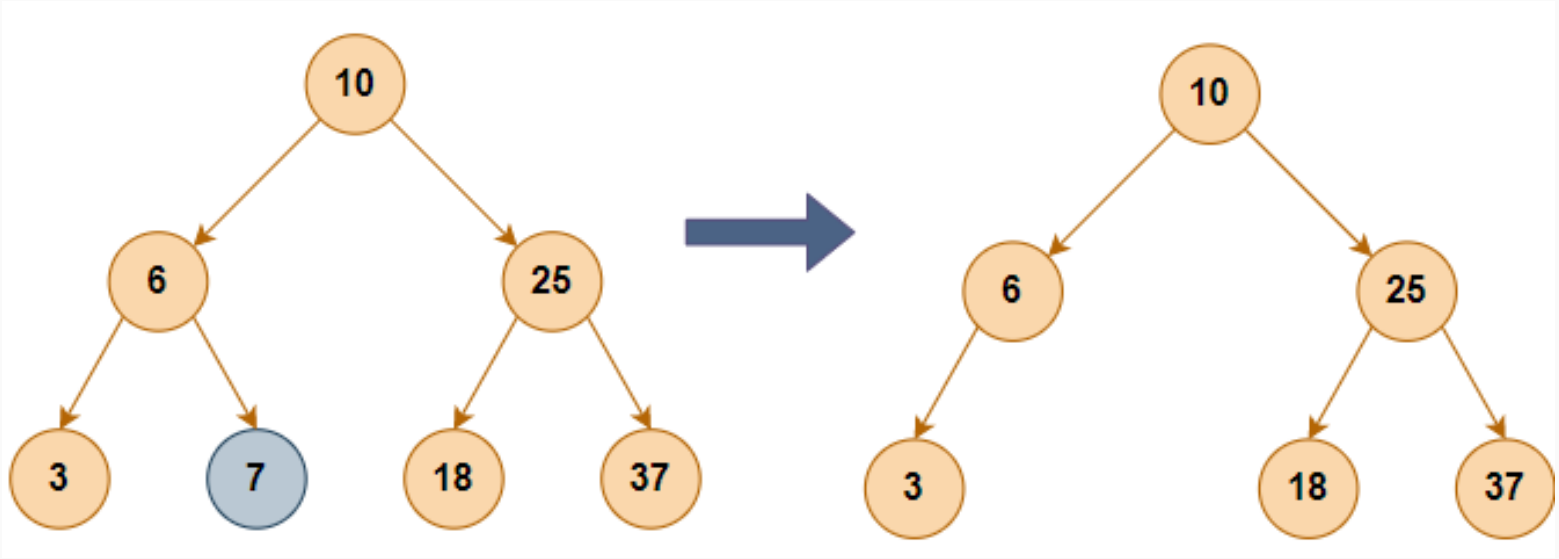
Algoritmo para **remover** um elemento de uma Árvore Binária de Busca:

- Encontrar o nó a ser removido.
- Casos:
  1. O nó é folha $\rightarrow$ eliminar o nó.
  2. O nó tem uma subárvore vazia $\rightarrow$ subir a subárvore não-vazia.
  3. O nó tem duas subárvores não-vazias:
     - Encontrar o maior valor na subárvore esquerda
     - Substituir, no nó a ser removido, o valor atual pelo valor maior da subárvore esquerda (recém encontrado)
     - Eliminar o maior valor da subárvore esquerda

Remover elemento: Árvore Binária de Busca

Caso 1 - Remoção de um elemento em nó folha

Remover o elemento 7:

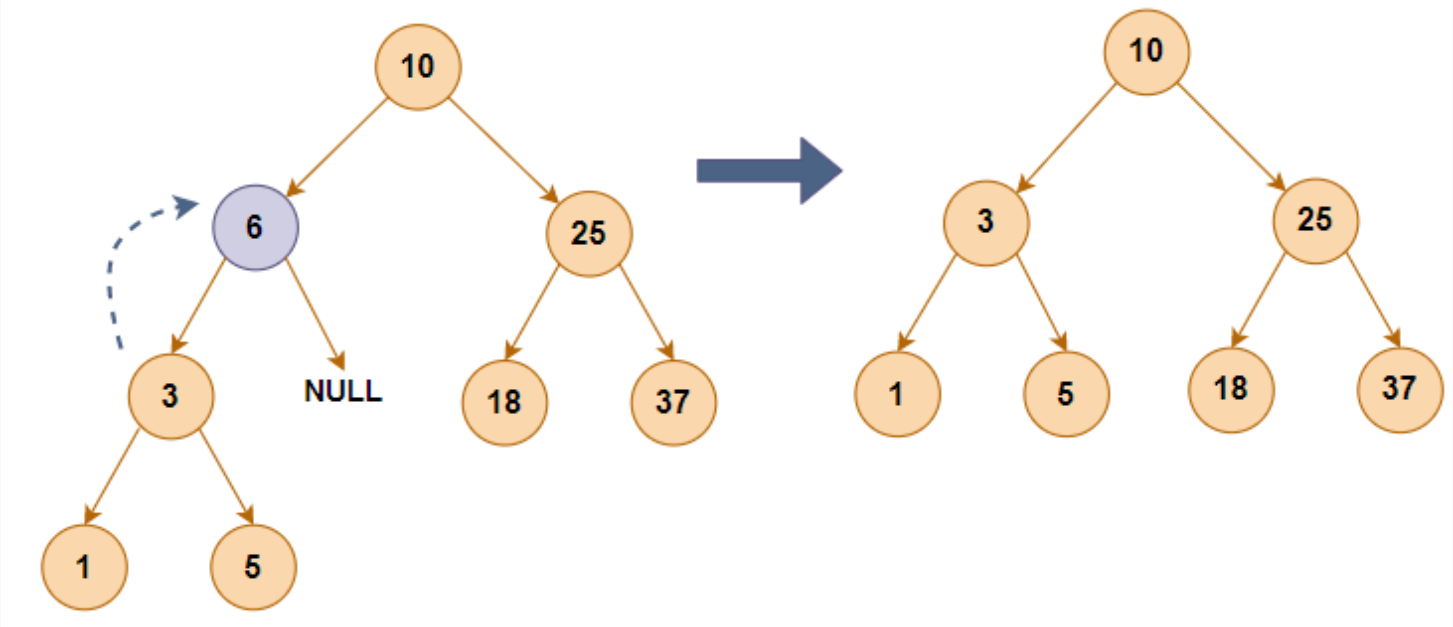


- encontrar o elemento a ser removido, raiz aponta para o nó 7;
- liberar a memória alocada para o nó raiz;
- retornar a raiz atualizada, que passa a ser NULL.

Remover elemento: Árvore Binária de Busca

Caso 2 - Remoção de um elemento com uma subárvore vazia

Remover o elemento 6:

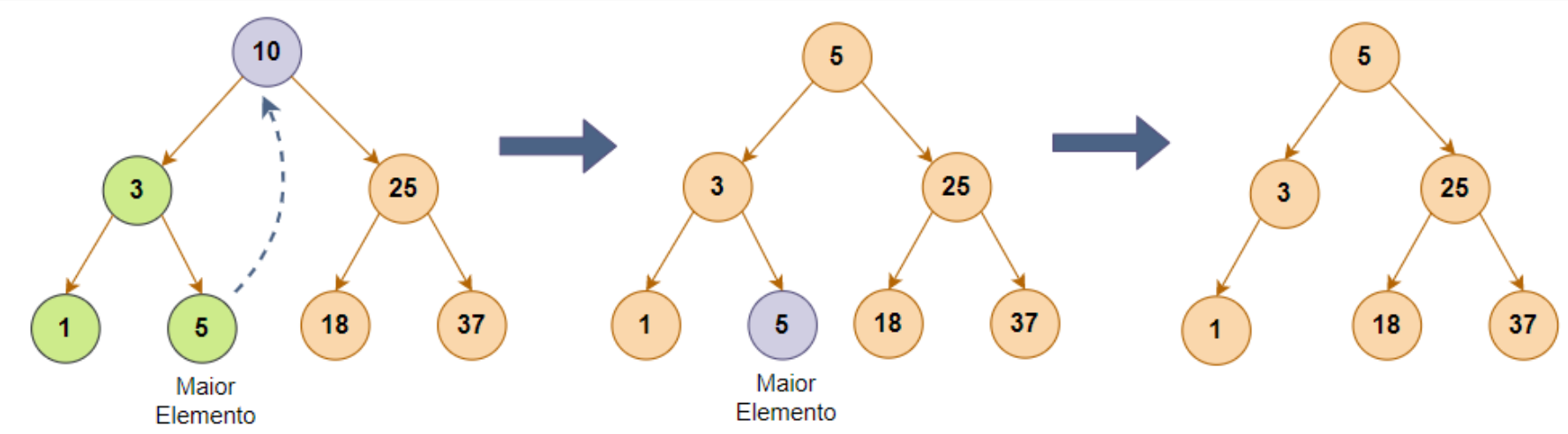


- encontrar o elemento a ser removido, raiz aponta para o nó 6;
- a raiz (nó 6) passa a ser o seu único filho;
- liberar a memória alocada para o nó 6.

Remover elemento: Árvore Binária de Busca

Caso 3 - Remoção de um elemento com duas subárvores não-vazias

Remover o elemento 10:



- encontrar o elemento a ser removido, raiz aponta para o nó 10;
- encontrar o maior elemento na subárvore da esquerda e usá-lo para substituir a raiz;
- remover o maior elemento da subárvore esquerda.

Remover elemento: Árvore Binária de Busca

Função para **remover** um elemento de uma Árvore Binária de Busca:

```
1  Arv* arv_remove (Arv* r, int c){
2     Arv *p;
3     if (arv_vazia(r)) return NULL;
4     else
5        if (c == r->info) {  // achou elemento a ser removido
6           if (arv_vazia(r->esq) && arv_vazia(r->dir)) {  // remoção em folha
7              free(r);  return NULL;  }
8           else
9              if (arv_vazia(r->esq)) { // remoção em nó interno (esquerda vazia)
10                p = r;  r = r->dir; free(p);  }
11             else
12                if (arv_vazia(r->dir)) { // remoção em nó interno (direita vazia)
13                   p = r; r = r->esq; free(p);  }
14                else {  // remoção em nó interno (dir/esq não vazias)
15                   p = arv_maior(r->esq);
16                   r->info = p->info;
17                   r->esq = arv_remove(r->esq, p->info);  }
18       else // continuar procurando o elemento
19          if (c < r->info)  r->esq = arv_remove(r->esq, c);
20          else  r->dir = arv_remove(r->dir, c);
21    return r; }
```

Função Principal: Árvore Binária de Busca

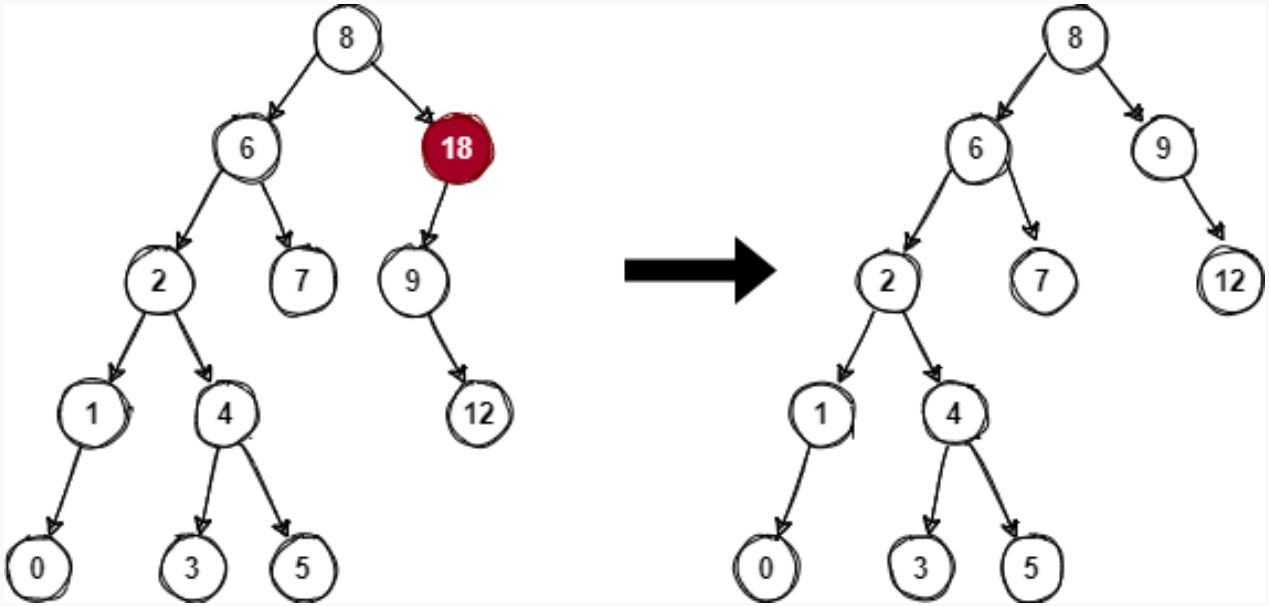
Função **principal** para a manipulação de uma Árvore Binária de Busca:

```
1  int main(){
2      Arv *a=NULL;
3      int v[] = {8,6,2,4,1,18,9,12,5,0,7,3};
4      int i, tam = 12, res, elem;
5
6      for(i=0;i<tam;i++) a = arv_insere(v[i],a);
7
8      printf("\nPercurso em Ordem: ");
9      arv_imprime_em(a);
10
11     elem = 18;
12     res = arv_busca(a,elem);
13     if (res==1)
14         printf(‘‘\nResultado da busca: Elemento %d encontrado’’,elem);
15     else printf(‘‘\nResultado da busca: Elemento %d não encontrado’’,elem);
16
17     a = arv_remove(a,elem);
18     printf(‘‘\nApós remoção do elemento %d:’’,elem);
19     printf(‘‘\nPercurso em Ordem: ’’);
20     arv_imprime_em(a);
21 }
```

Execução: Árvore Binária de Busca

Resultado da execução:

```
Percurso em Ordem: 0 1 2 3 4 5 6 7 8 9 12 18
Resultado da busca: Elemento 18 encontrado
Após remoção do elemento 18:
Percurso em Ordem: 0 1 2 3 4 5 6 7 8 9 12
```



Percurso em níveis: Árvore Binária de Busca

Função de **percurso com atravessamento em níveis** usando fila:

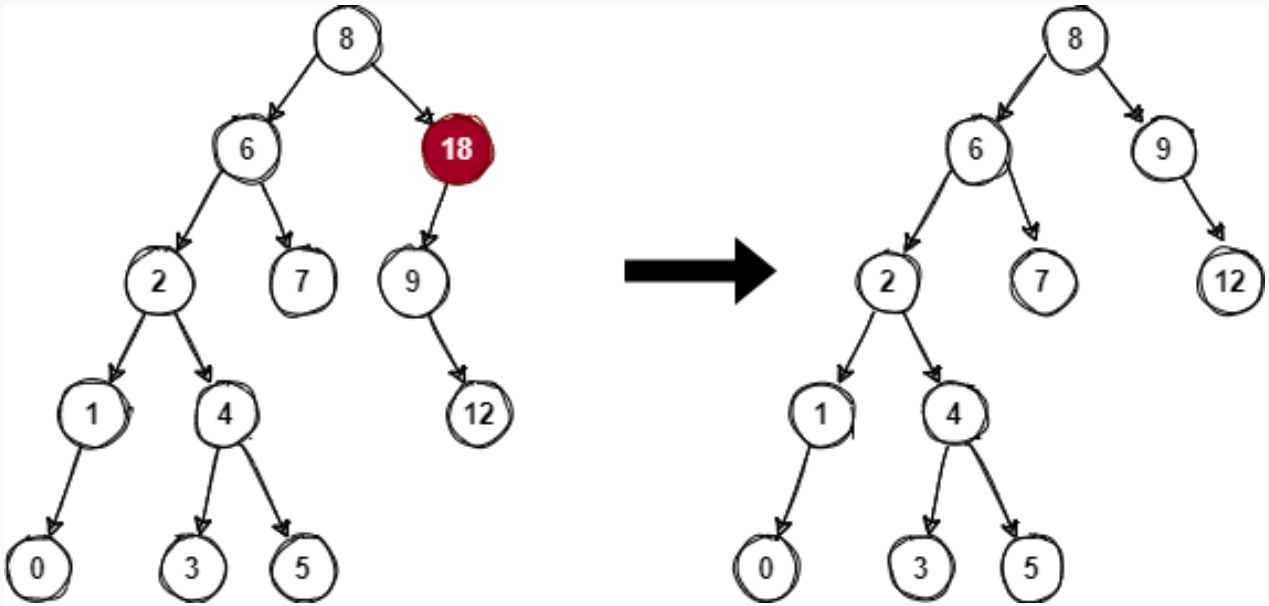
```
1  void arv_imprime_niveis (Arv* a){
2    TipoFila *f;
3    Arv *aux;
4    if (!arv_vazia(a)){
5        f = IniciaFila();
6        Enfileira(a,f);
7        while (!Vazia(f)){
8           aux = Desenfileira(f);
9           printf("%d ",aux->info);
10          if (!arv_vazia(aux->esq))
11                Enfileira(aux->esq,f);
12          if (!arv_vazia(aux->dir))
13                Enfileira(aux->dir,f);
14       }
15   }
16 }
```

```
1  void Enfileira(Arv *x, TipoFila *fila){
2     TipoNo *aux;
3     aux = (TipoNo *) malloc(sizeof(TipoNo));
4     aux->valor = x;
5     aux->prox = NULL;
6     if (Vazia(fila)){
7        fila->inicio=aux;
8        fila->fim=aux;    }
9     else {
10       fila->fim->prox = aux;
11       fila->fim = aux;  }
12    fila->tamanho++;
13 }
14
15 Arv* Desenfileira(TipoFila *fila){
16    TipoNo *q;   Arv *v;
17    if (Vazia(fila)) return 0;
18    q = fila->inicio;
19    v = fila->inicio->valor;
20    fila->inicio = fila->inicio->prox;
21    free(q);
22    fila->tamanho--;
23    return v;
24 }
```

Percurso em níveis: Árvore Binária de Busca

Resultado da execução:

```
Percurso em Ordem: 0 1 2 3 4 5 6 7 8 9 12 18
Percurso em Niveis: 8 6 18 2 7 9 1 4 12 0 3 5
Resultado da busca: Elemento 18 encontrado
Após remoção do elemento 18:
Percurso em Ordem: 0 1 2 3 4 5 6 7 8 9 12
Percurso em Niveis: 8 6 9 2 7 12 1 4 0 3 5
```



Referência e Material Extra

- Celes, W.; Cerqueira, R.; Rangel, J.L. *Introdução a Estruturas de Dados com Técnicas de Programação em C*. 2a. ed. Elsevier, 2016.

* Backes, A. *Programação Descomplicada - Estruturas de Dados*.
    - Vídeo-aulas 67 a 77: *https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/*