Aula 02 - Árvores Binárias

Estrutura de dados e seus algoritmos

Profa: Taiane C. Ramos

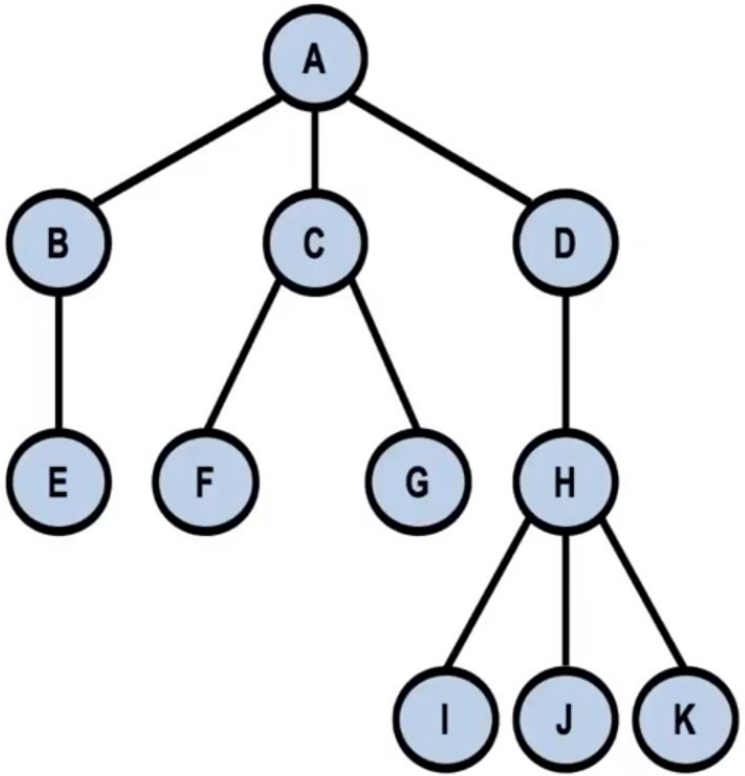
Árvore

É uma estrutura com hierarquia.

Nó raiz: A

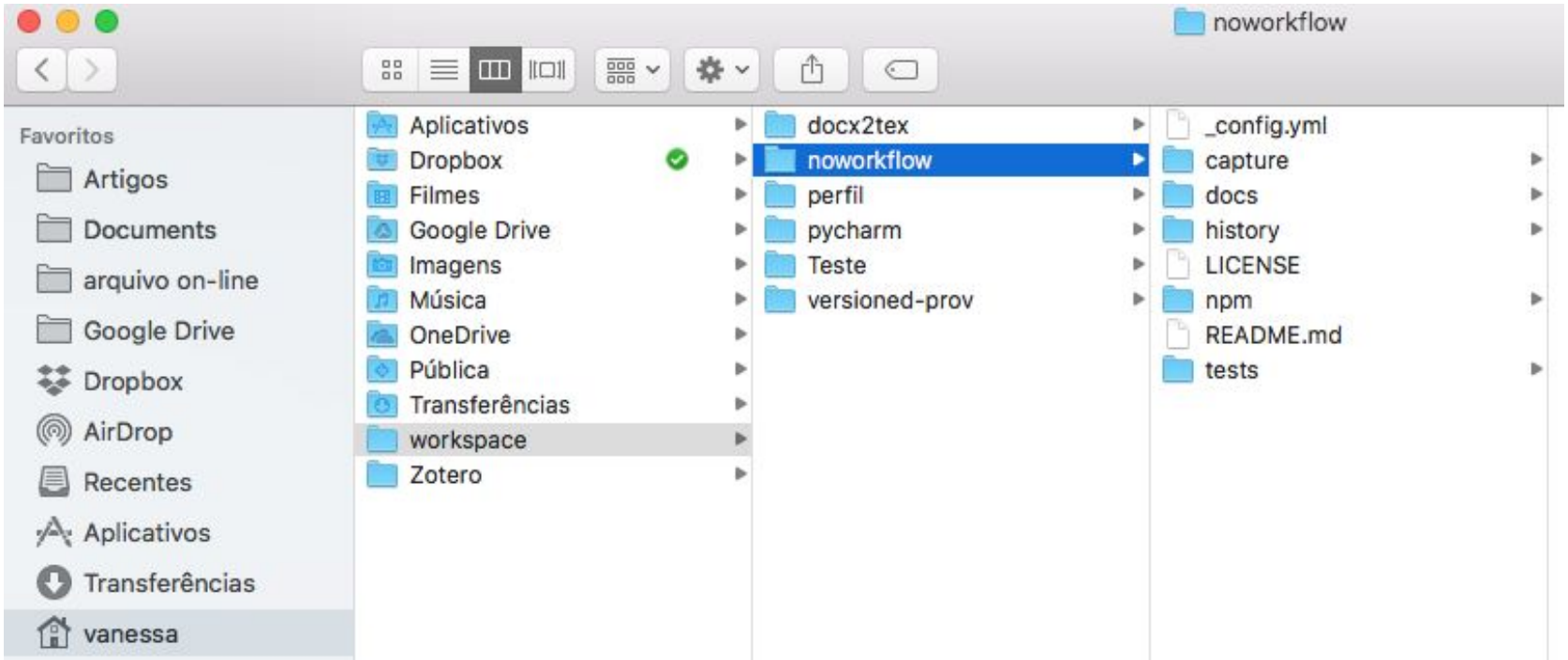
Filhos de A: B, C, D

Nós folha: E, F, G, I, J, K



Exemplo de aplicação

Sistema de arquivos do computador

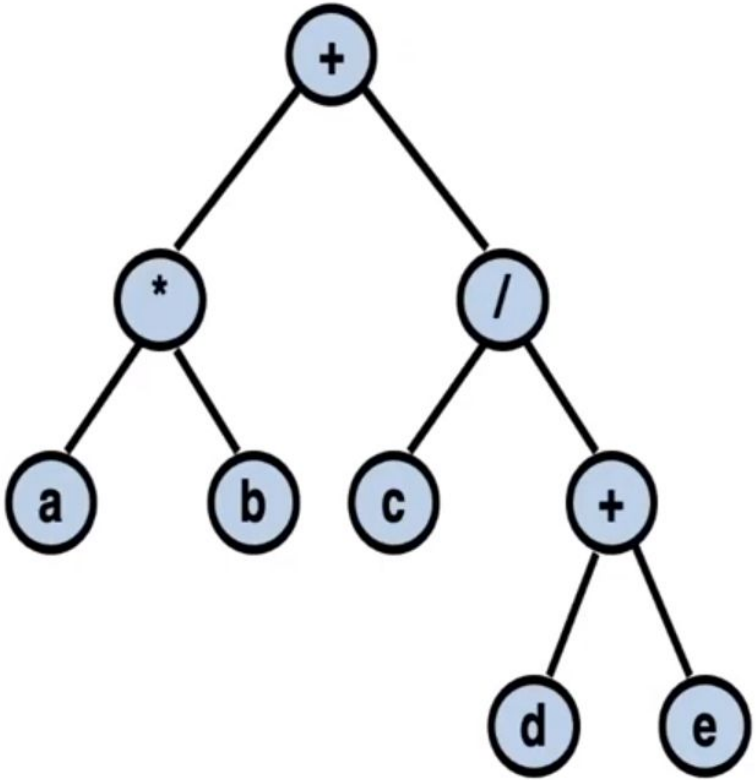


Exemplo de aplicação

Árvore de decisão

Exemplo de aplicação

Árvore de derivação



Expressão aritmética: (a * b) + (c / (d + e))

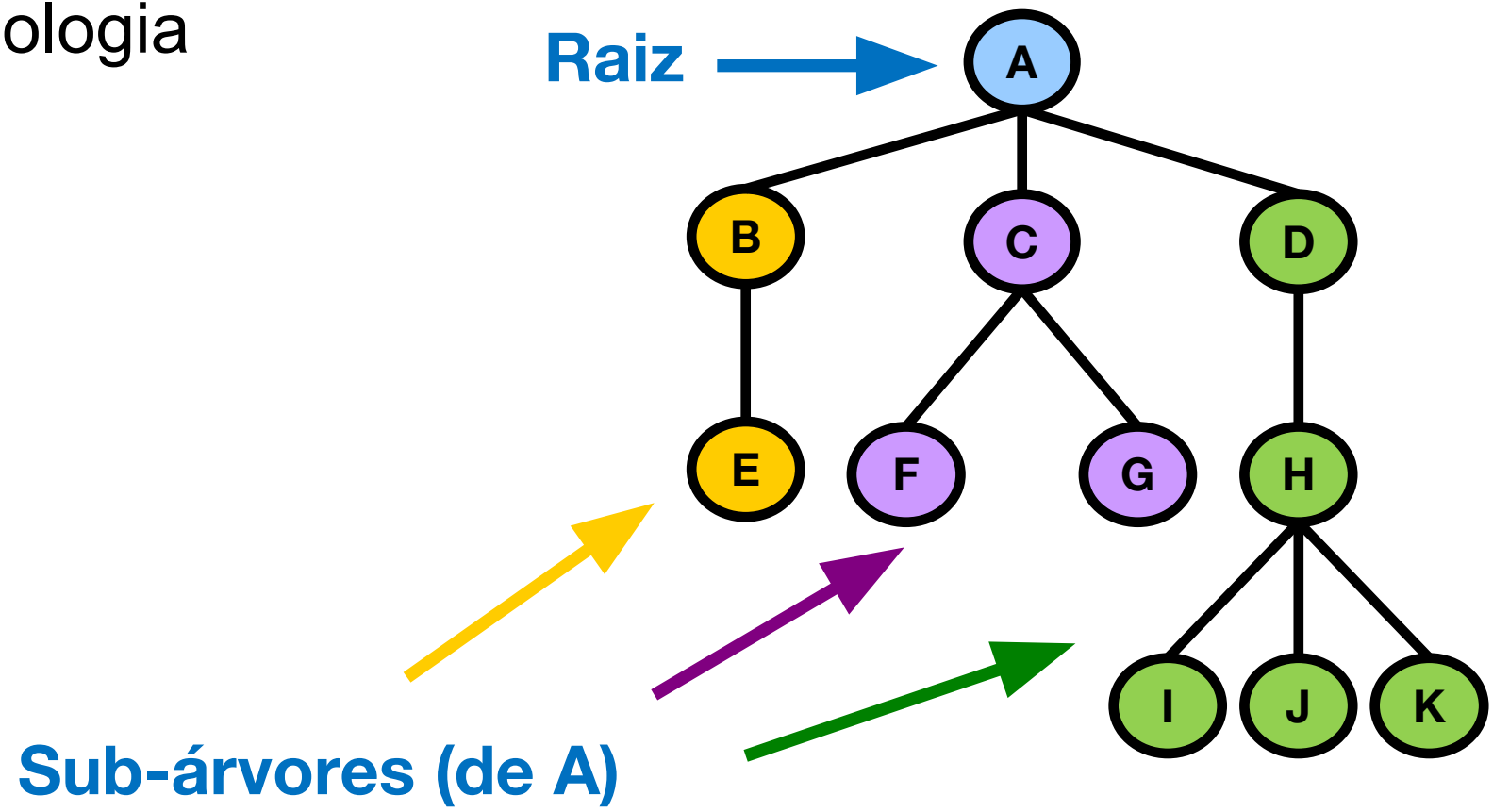
Definições

Toda árvore tem uma única raiz.

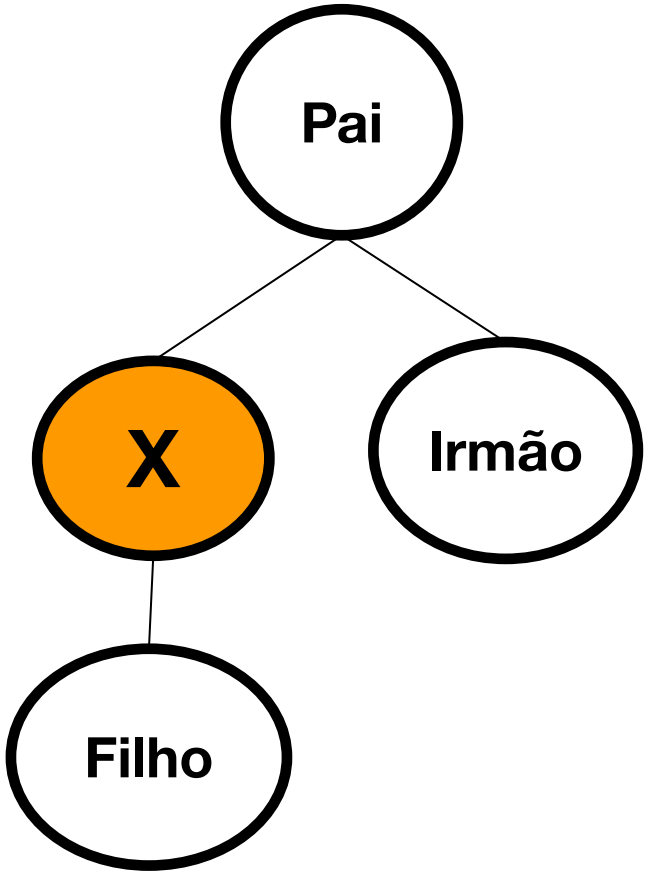
Uma árvore com zero nós tem a raiz vazia.

Uma árvore contém sub-árvores e cada subárvore tem uma raiz.

Terminologia



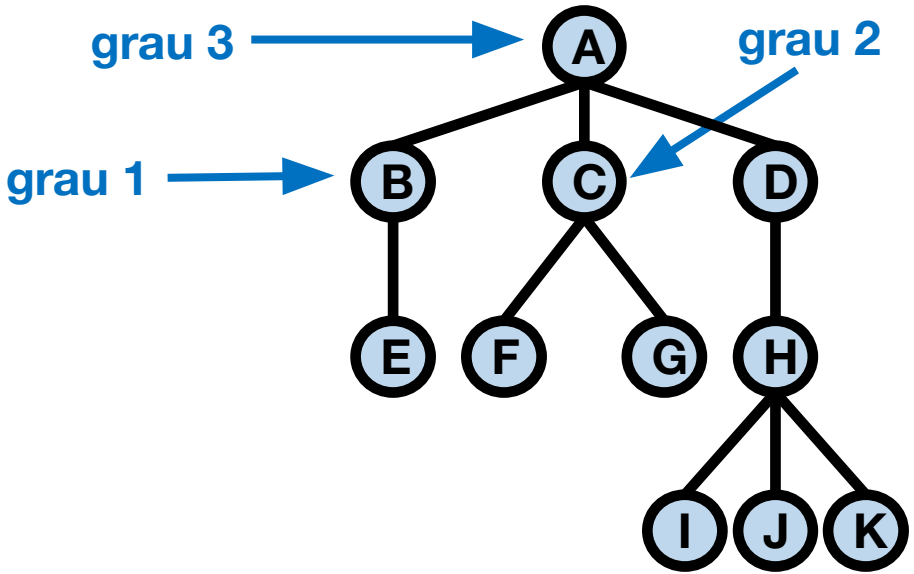
Terminologia



Terminologia

Grau do nó

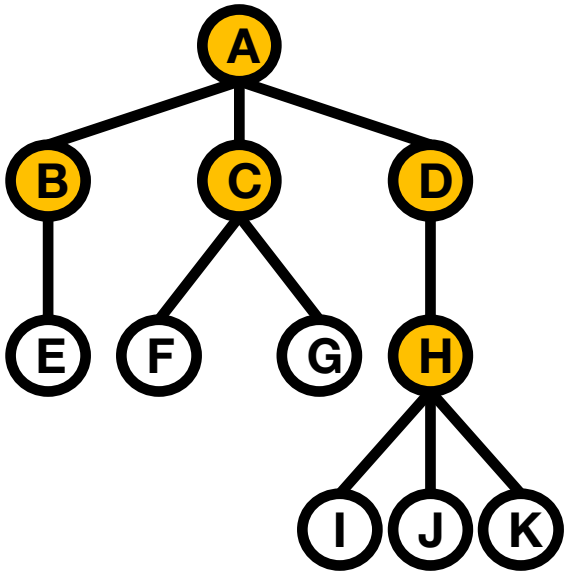
- número de nós filhos



Terminologia

Nó interno (ou nó de derivação)

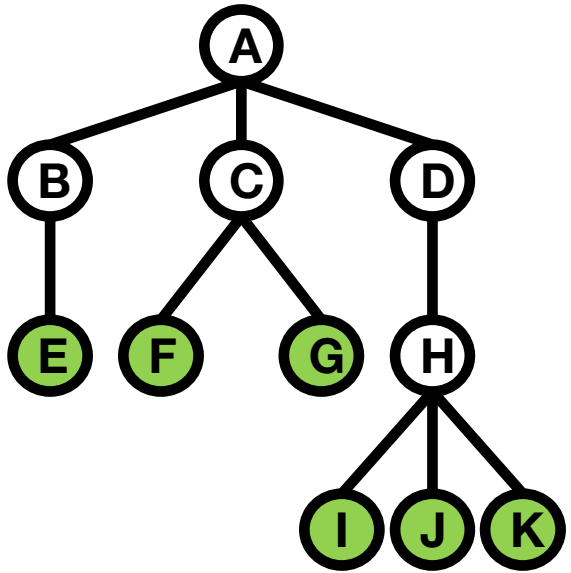
- nó com pelo menos 1 filho (grau > 0)



Terminologia

Nó folha (nó terminal ou nó externo)

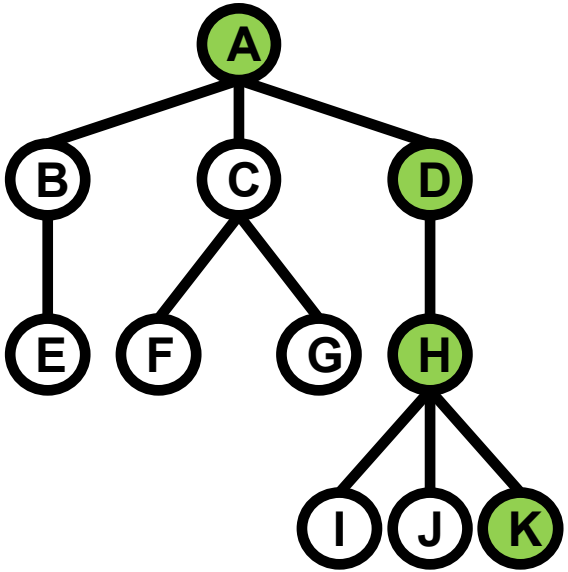
- nó sem filho (grau = 0)



Terminologia

Um caminho é uma sequência de nós que leva de um nó a outro.

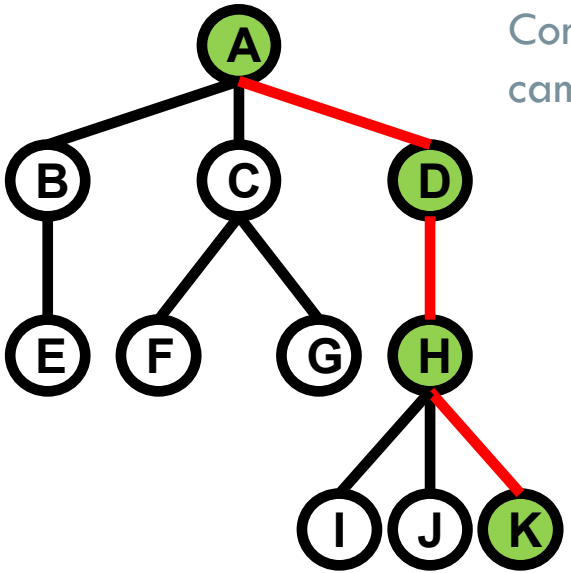
Caminho de A a K: A, D, H, K



Terminologia

Comprimento do caminho

- Número de ligações entre os nós do caminho

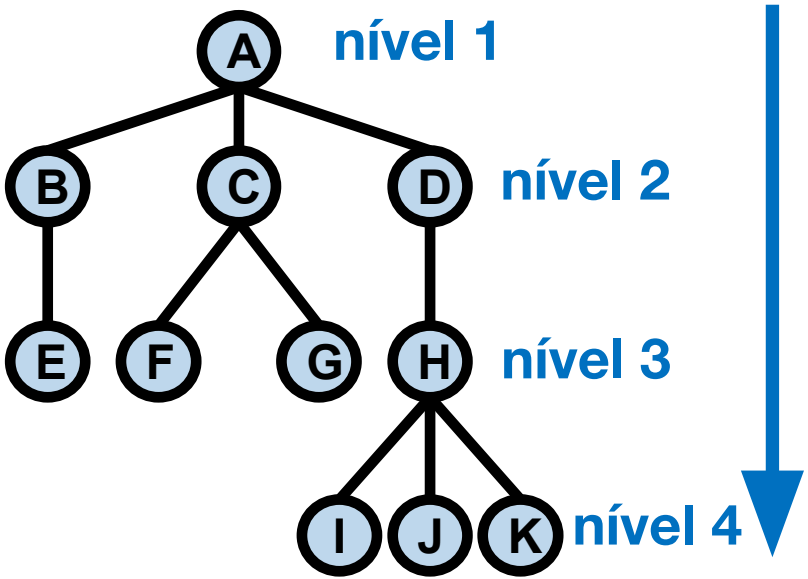


Comprimento do caminho= 3

Terminologia

Nível:

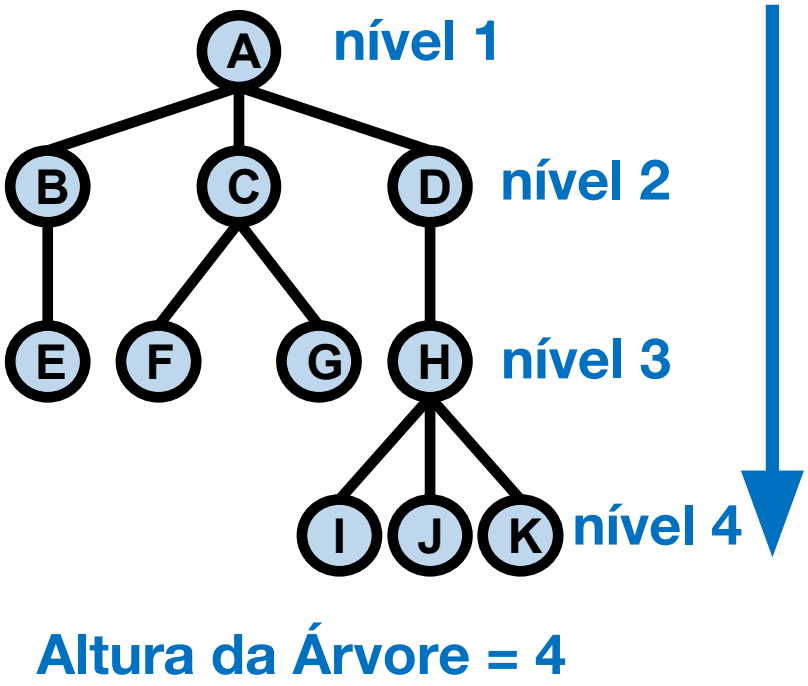
- Raiz é sempre nível 1
- A cada “passo” pra chegar em um nó, aumenta um nível.



Terminologia

Altura da árvore

- maior nível



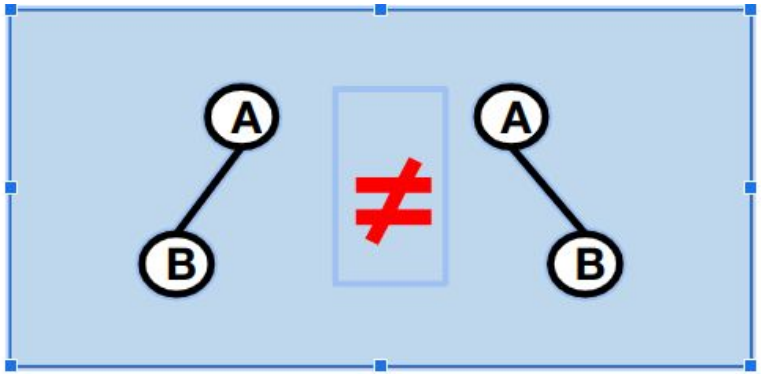
Terminologia

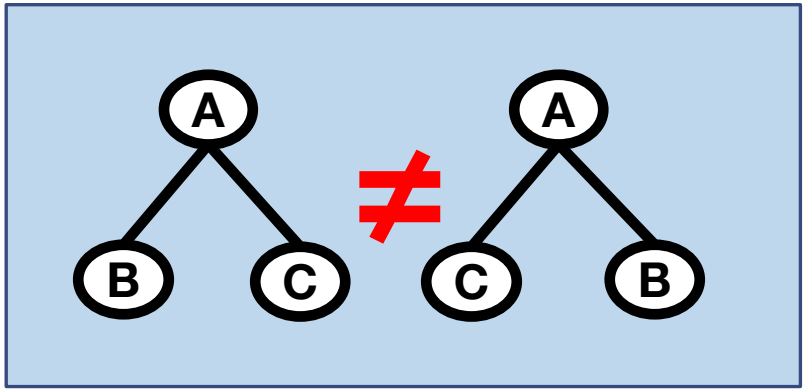
Árvore ordenada

- Ordem das sub-árvores é relevante

Uma árvore ordenada tem os filhos em ordem da esquerda pra direita.

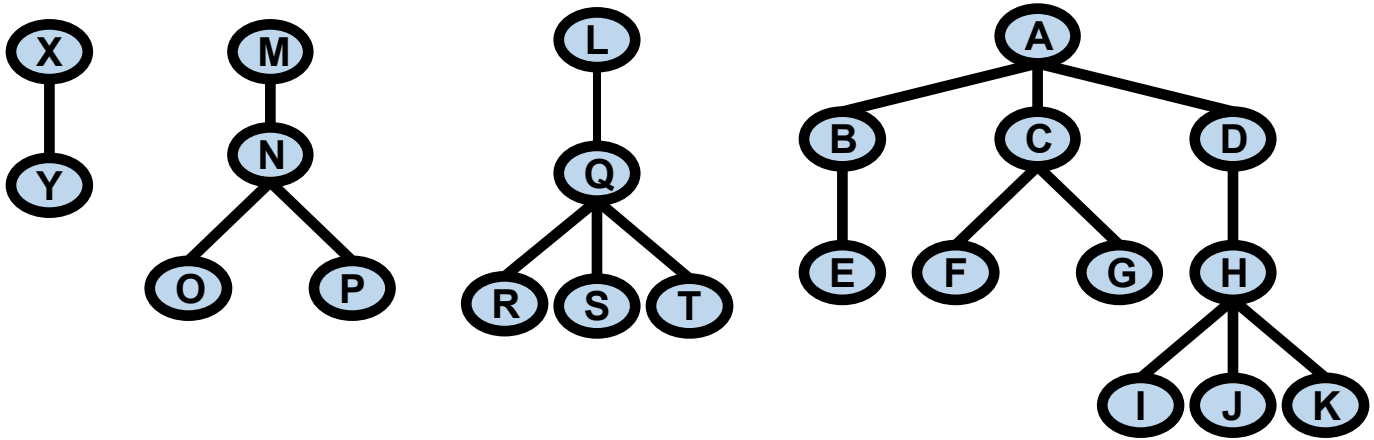
Duas árvores são isomorfas quando diferem apenas na ordem das sub-árvores.





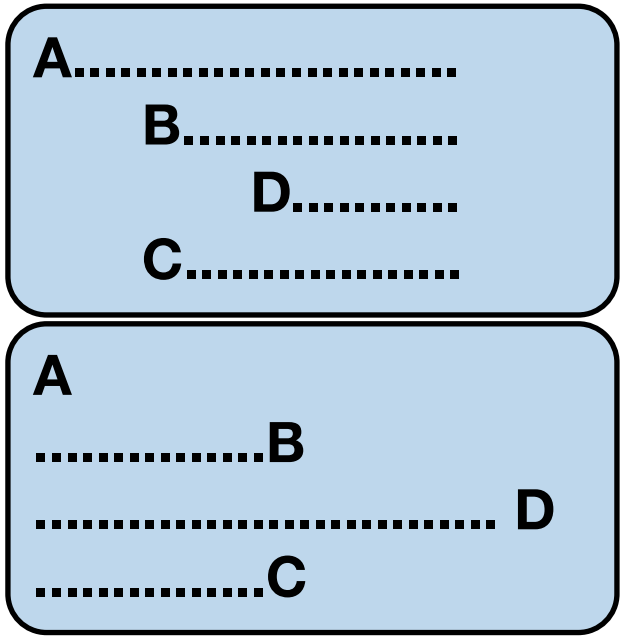
Terminologia

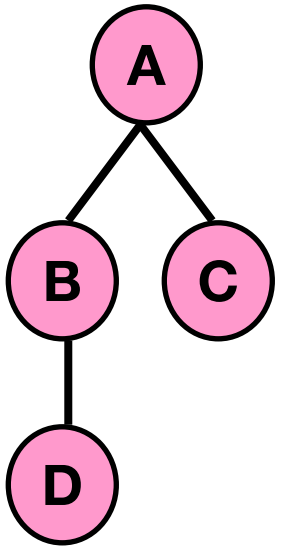
Floresta é um conjunto de árvores



Representação em modo texto

Diagrama de barras

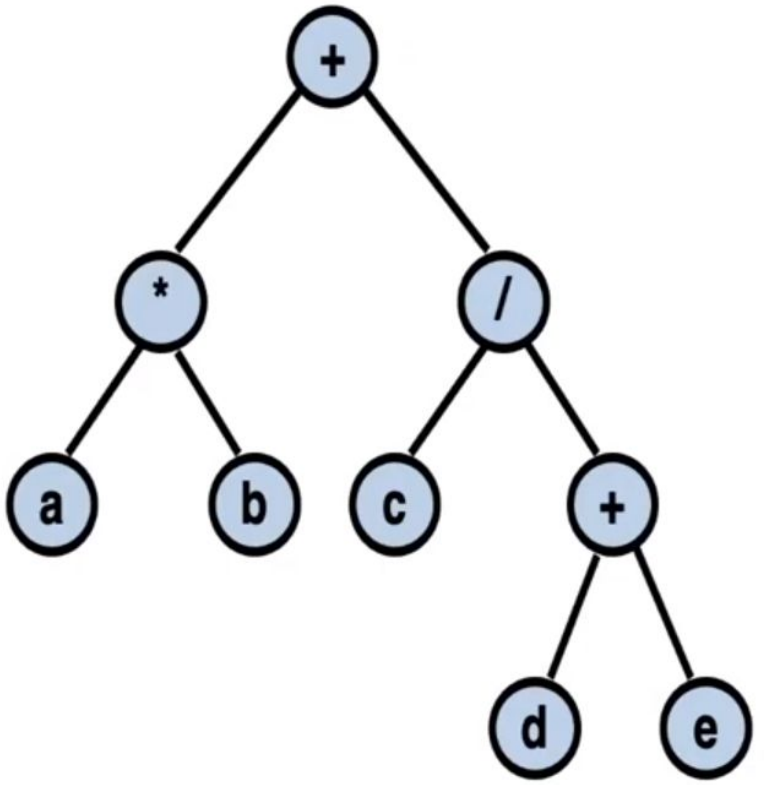




Árvores Binárias

Árvores Binárias

Cada nó aponta 2 filhos.



Expressão aritmética: (a * b) + (c / (d + e))

Definição formal

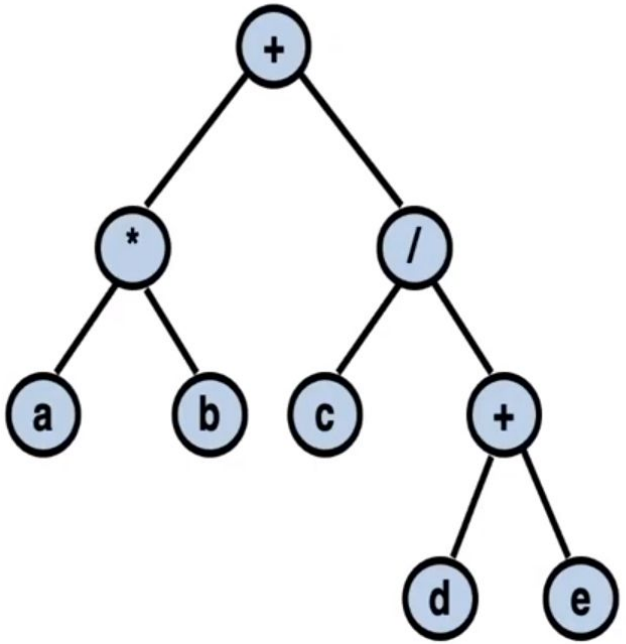
Conjunto finito T de zero ou mais nós (nodos), tal que:

Se número de nós é maior do que zero

- existe um nó denominado raiz da árvore
- os demais nós formam 2 conjuntos disjuntos S_1, S_2 onde cada um destes é uma árvore binária

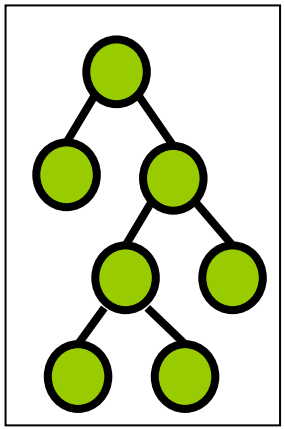
Se número de nós é igual a zero

- árvore vazia

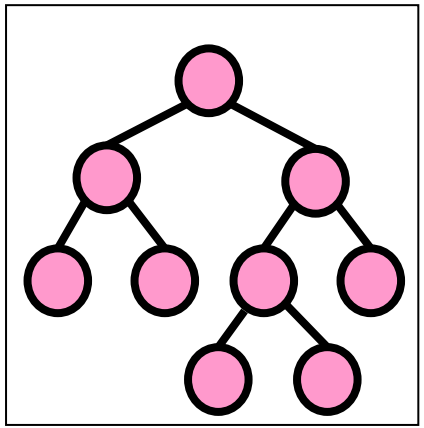


Expressão aritmética: (a * b) + (c / (d + e))

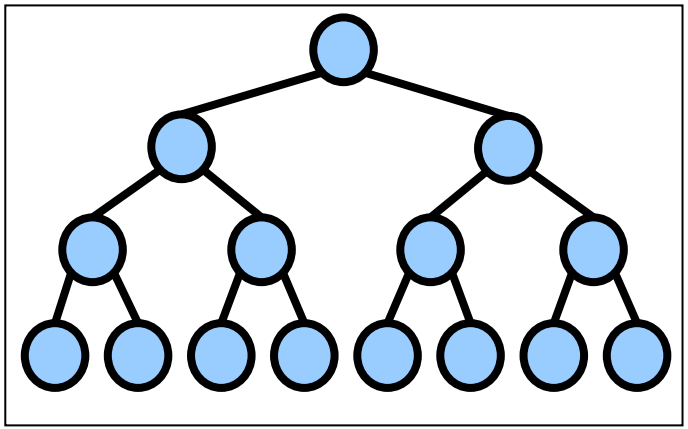
TIPOS DE ÁRVORES BINÁRIAS



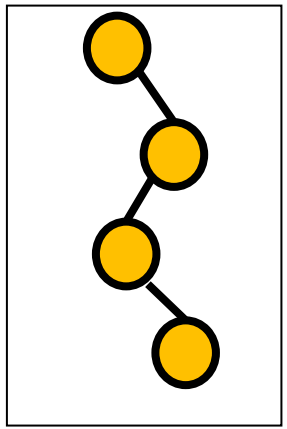
Estritamente Binária
0 ou 2 filhos



Binária Completa
Sub-árvores vazias apenas no último ou penúltimo nível



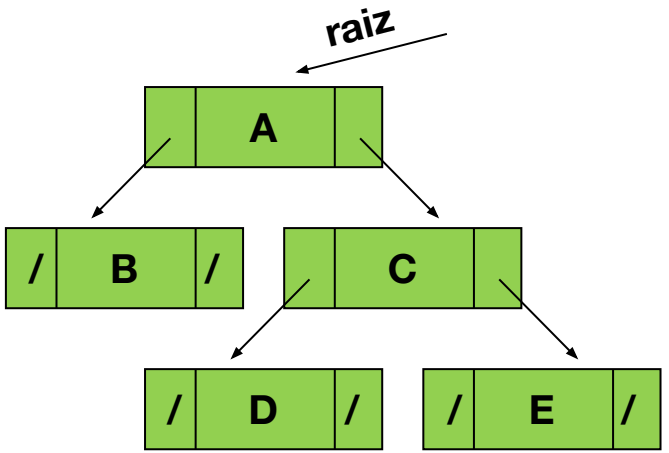
Binária Cheia
Sub-árvores vazias somente no último nível



Zigue Zague
Nós internos com 1 subárvore vazia

Código - Struct do nó

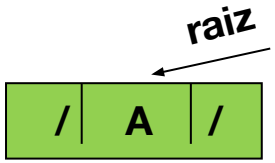
```
typedef struct noA{
    char info;
    struct noA *esq;
    struct noA *dir;
} TNoA;
```



Representação do nó:

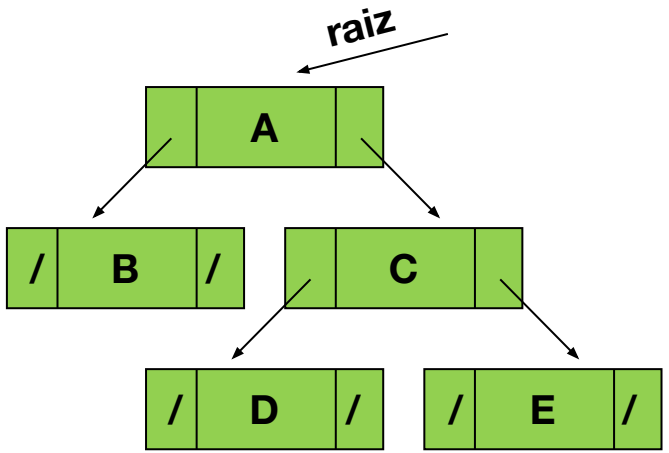
Código - Criar árvore (nó raiz)

```
TNoA *criaNo(char ch) {
    TNoA *novo;
    novo = (TNoA *) malloc(sizeof(TNoA));
    novo->info = ch;
    novo->esq = NULL;
    novo->dir = NULL;
    return novo;
}

int main(void) {
    TNoA *raiz;
    raiz = criaNo('A');

}
```



Código - Inserir filhos

```
int main(void) {
    TNoA *raiz;
    raiz = criaNo('A');
    raiz->esq = criaNo('B');
    raiz->dir = criaNo('C');
    raiz->dir->esq = criaNo('D');
    raiz->dir->dir = criaNo('E');
    imprime(raiz, 0);
};
```



Código - Imprimir árvore

```
void imprime(TNoA *nodo, int tab) {
    for (int i = 0; i < tab; i++) {
        printf("-");
    }
    if (nodo != NULL) {
        printf("%c\n", nodo->info);
        imprime(nodo->esq, tab + 2);
        printf("\n");
        imprime(nodo->dir, tab + 2);
    } else printf("vazio");
}
```

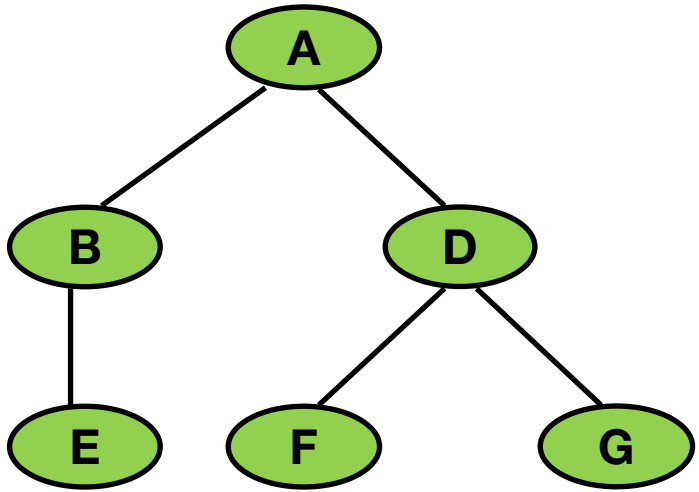
```
A
--B
----vazio
----vazio
--C
----D
------vazio
------vazio
----E
------vazio
------vazio
```

Percorrer a árvore

Queremos percorrer todos os nós da árvore para executar alguma operação.

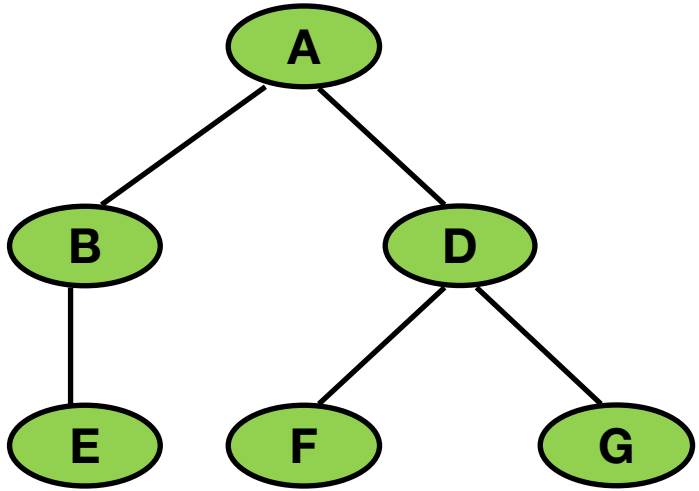
Exemplo:

- Imprimir todos os nós
- Buscar um valor específico



Percorrer a árvore

Exemplos de caminhos possíveis:



Caminho em largura:

A, B, D, E, F, G

Caminho em profundidade:

A, B, E, D, F, G

Percorrer a árvore

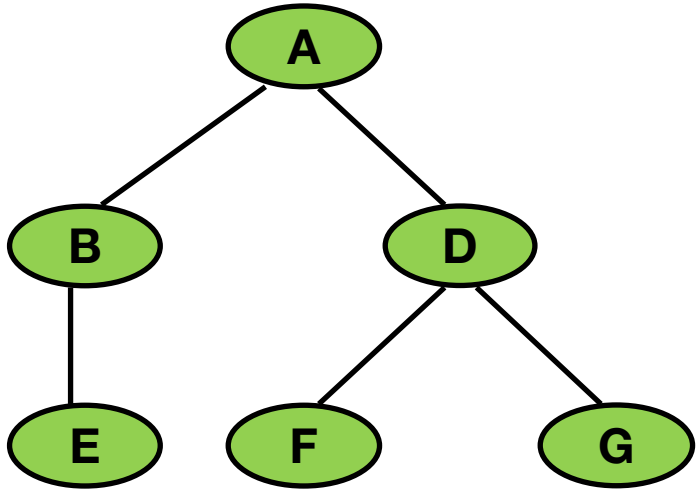
Busca em profundidade (pré-ordem):

1 - Visitar a raiz

2 - Descer pela esquerda até que seja vazio

3 - Voltar um nível e descer pela direita

Caminho: A, B, E, D, F, G



Percorrer a árvore

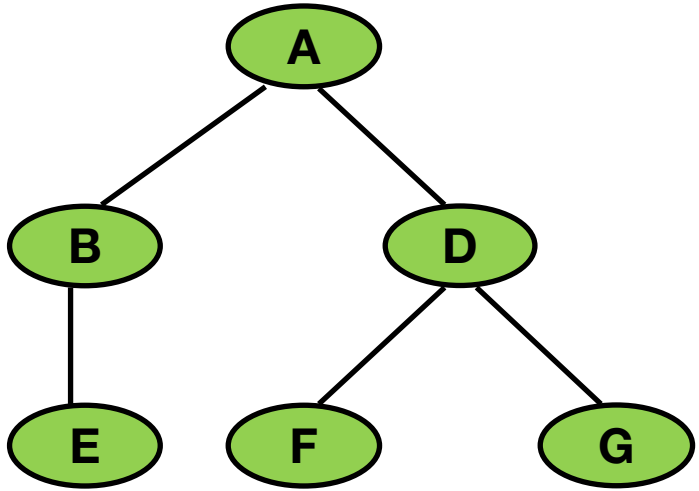
Busca em largura (por nível):

1 - Visitar a raiz

2 - Repetir com a árvore esquerda

3 - Repetir com a árvore direita

Caminho: A, B, D, E, F, G



Percorrer a árvore

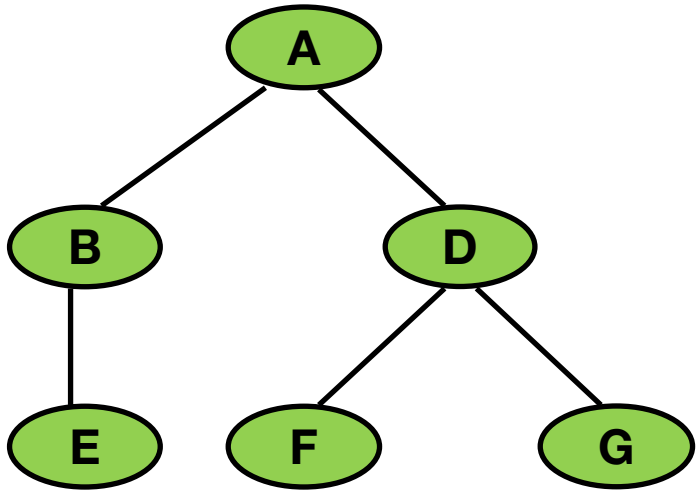
Ordem simétrica:

1 - Percorre sub-árvore esquerda

2 - Visitar a raiz

3 - Percorre sub-árvore direita

Caminho: E, B, A, F, D, G



Percorrer a árvore

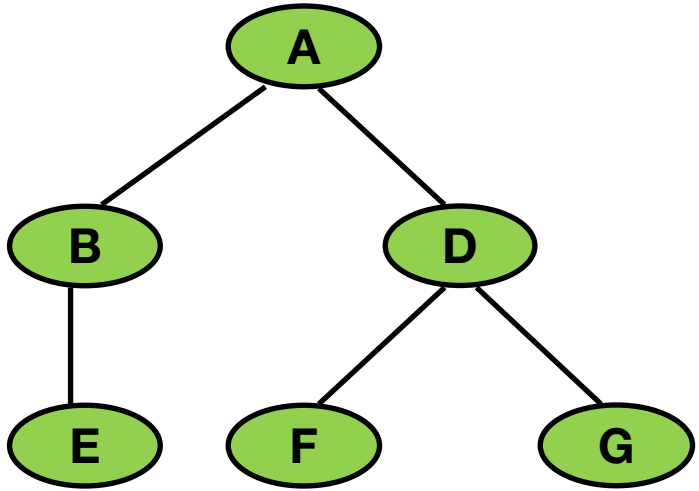
Pós-ordem:

1 - Percorre sub-árvore esquerda

2 - Percorre sub-árvore direita

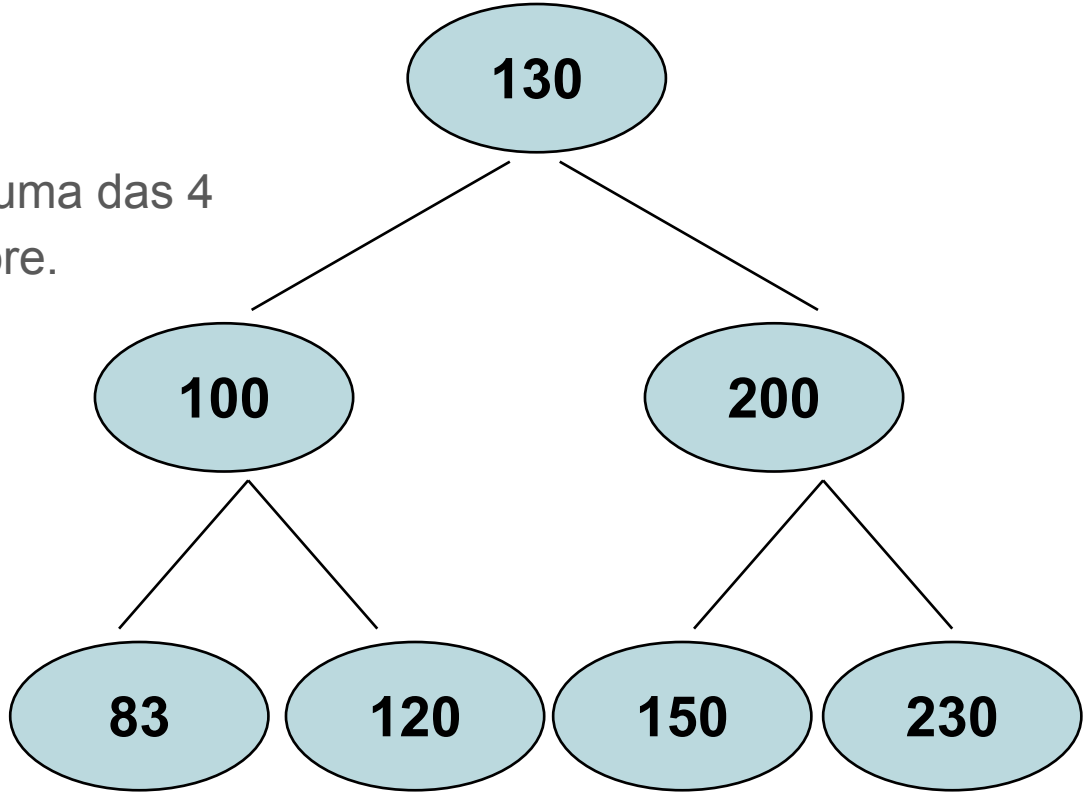
3 - Visitar a raiz

Caminho: E, B, F, G, D, A



Exercício

Escreva os caminhos em cada uma das 4 formas de visitar os nós da árvore.



Exercício

Profundidade:

130, 100, 83, 120, 200, 150, 230

Largura:

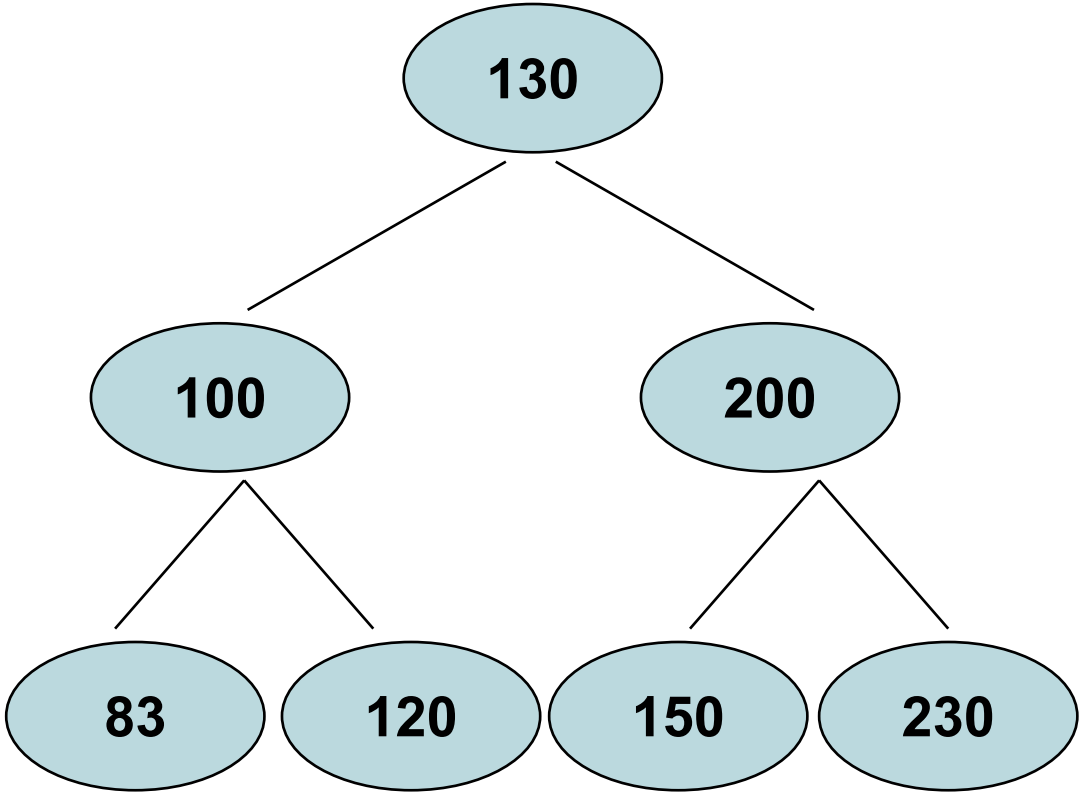
130, 100, 200, 83, 120, 150, 230

Simétrica:

83, 100, 120, 130, 150, 200, 230

Pós-ordem:

83, 120, 100, 150, 230, 200, 130



Percurso em profundidade

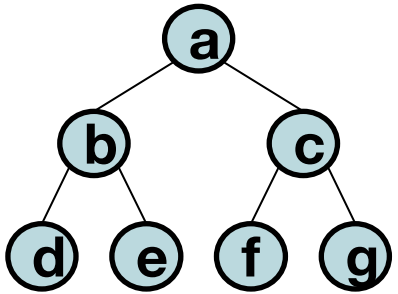
Implementação - Profundidade

Estrutura auxiliar necessária: pilha

Empilhar a raiz

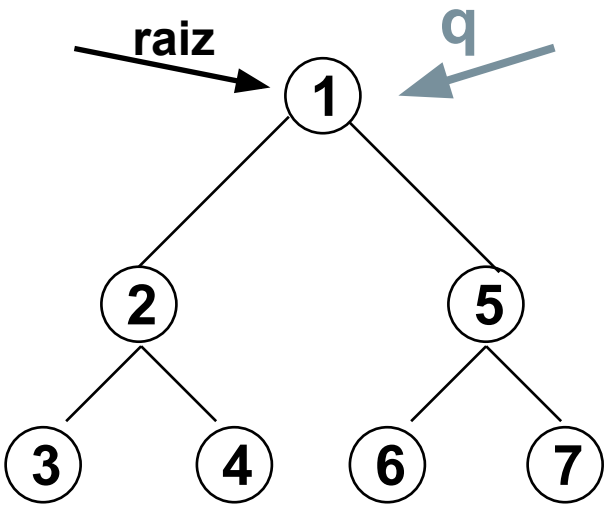
Repetir até que a pilha fique vazia

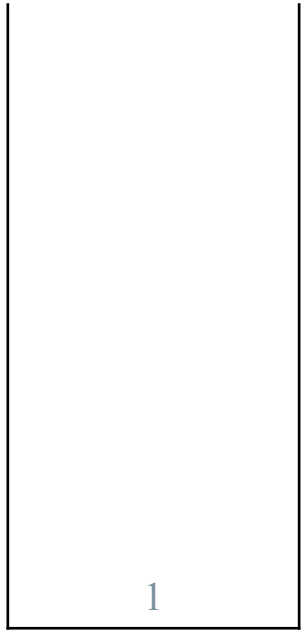
1. Desempilha topo da pilha (visita)
2. Empilha nó da direita (se diferente de NULL)
3. Empilha nó da esquerda (se diferente de NULL)



a - b - d - e - c - f - g

PERCORRER EM PROFUNDIDADE COM USO DE PILHA

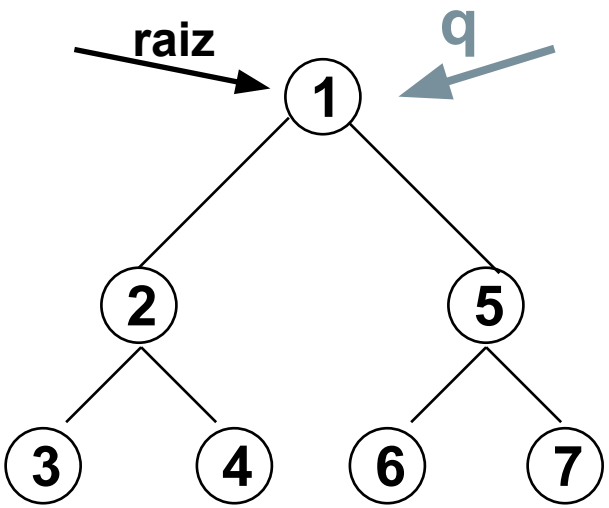


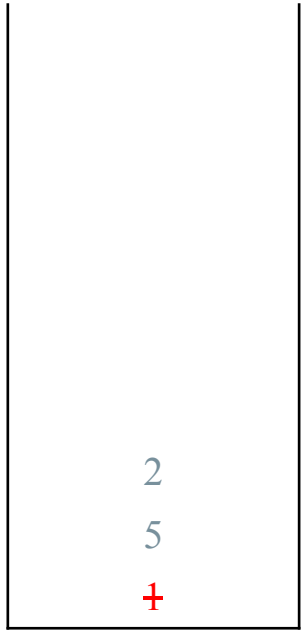


Pilha

Caminhamento:

PERCORRER EM PROFUNDIDADE COM USO DE PILHA

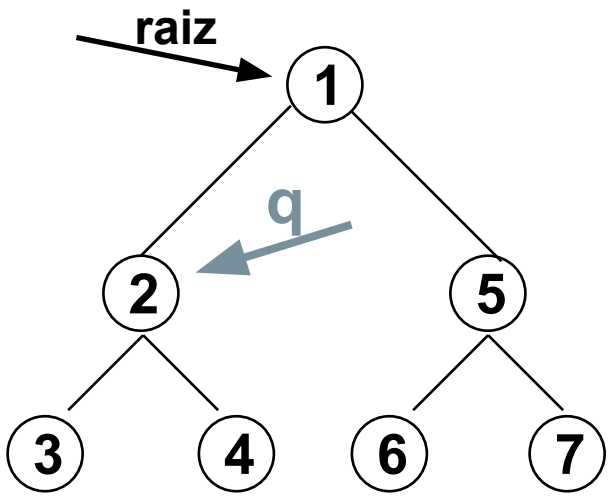


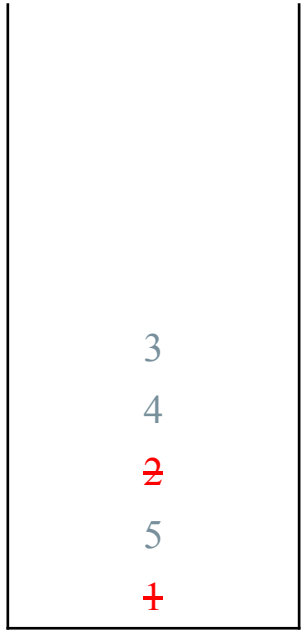


Pilha

Caminhamento: 1

PERCORRER EM PROFUNDIDADE COM USO DE PILHA

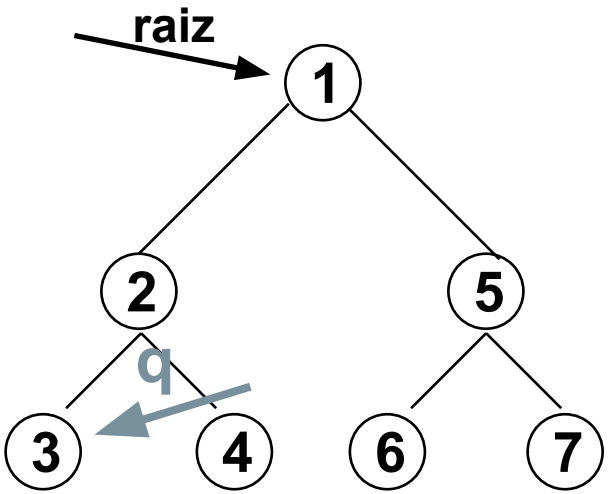


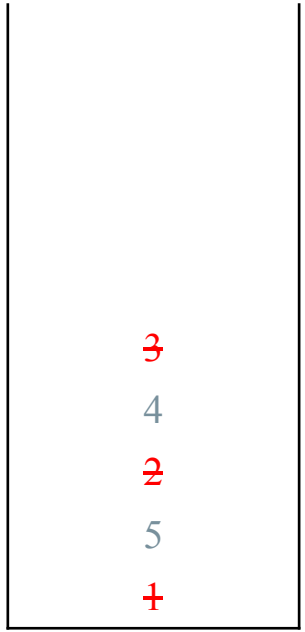


Pilha

Caminhamento: 1 – 2

PERCORRER EM PROFUNDIDADE COM USO DE PILHA

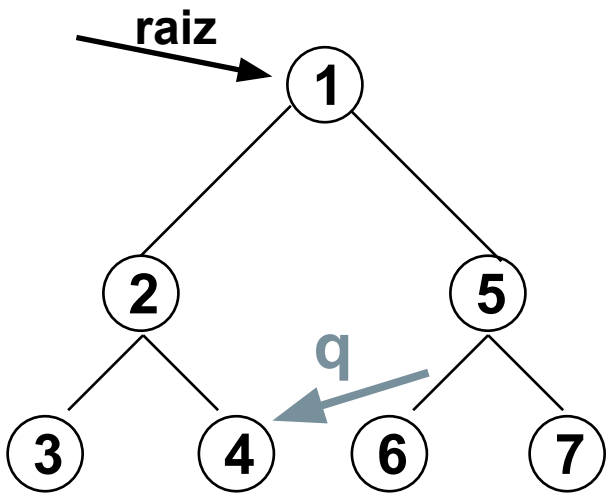


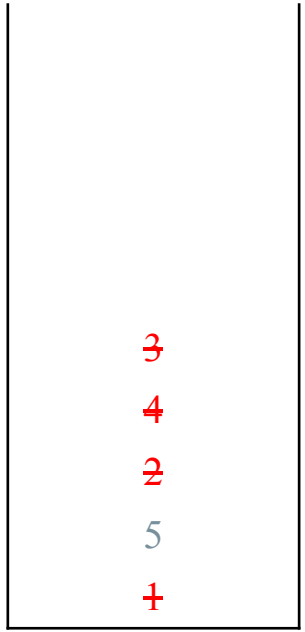


Pilha

Caminhamento: 1 – 2 – 3

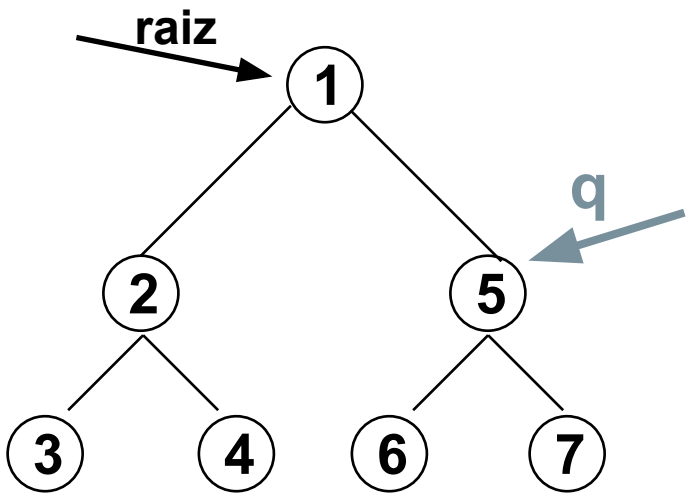
PERCORRER EM PROFUNDIDADE COM USO DE PILHA

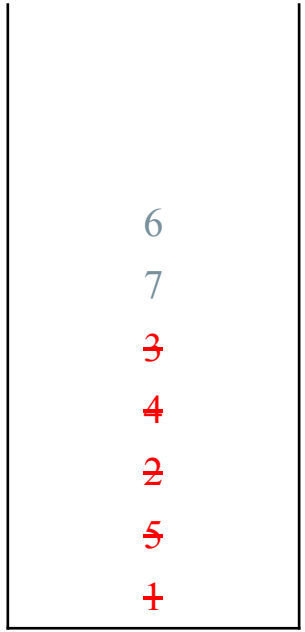




Caminhamento: 1 – 2 – 3 – 4

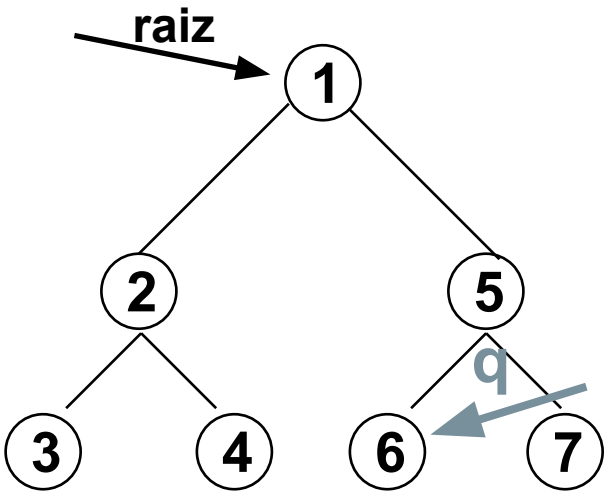
PERCORRER EM PROFUNDIDADE COM USO DE PILHA

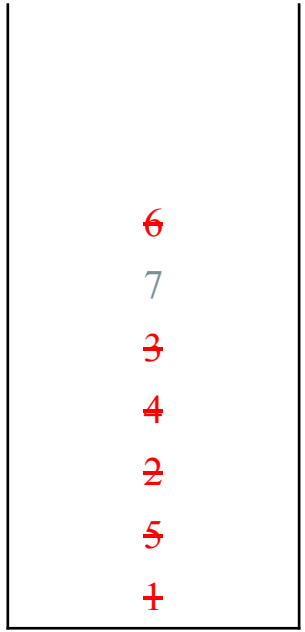




Caminhamento: 1 – 2 – 3 – 4 – 5

PERCORRER EM PROFUNDIDADE COM USO DE PILHA

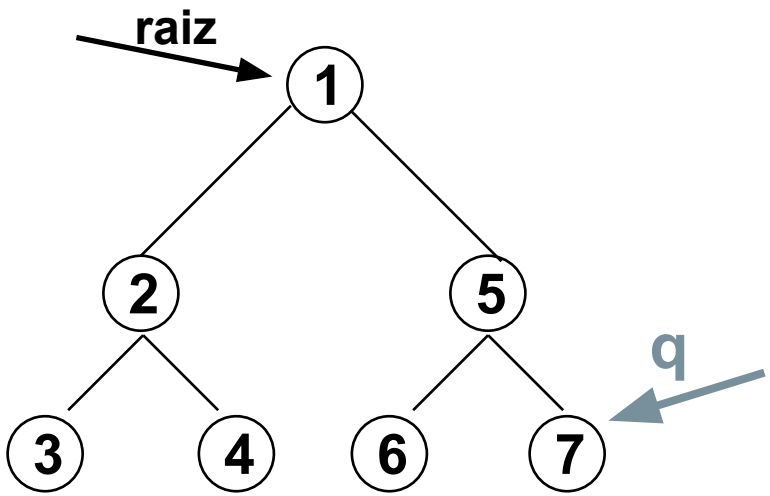


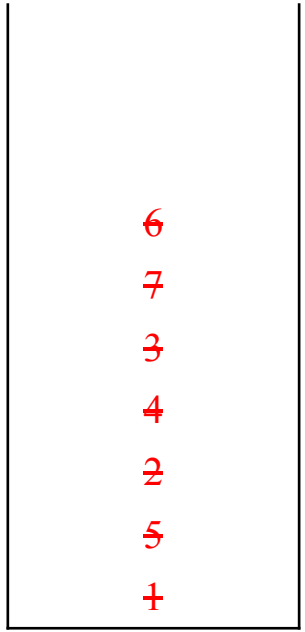


Pilha

Caminhamento: 1 – 2 – 3 – 4 – 5 – 6

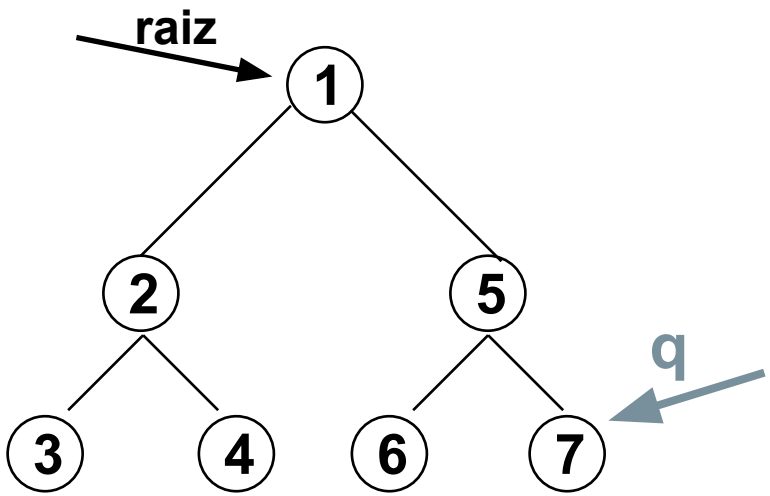
PERCORRER EM PROFUNDIDADE COM USO DE PILHA





Caminhamento: 1 – 2 – 3 – 4 – 5 – 6 – 7

PERCORRER EM PROFUNDIDADE COM USO DE PILHA



Pilha vazia: fim da execução

~~6~~
~~7~~
~~3~~
~~4~~
~~2~~
~~5~~
~~1~~

Pilha

Caminhamento: 1 – 2 – 3 – 4 – 5 – 6 – 7

Algoritmo para percorrer em profundidade

Profundidade

Estrutura auxiliar necessária: **pilha**

Empilhar a raiz

Repetir até que a pilha fique vazia

1. Desempilha topo da pilha (visita)
2. Empilha nó da direita (se diferente de NULL)
3. Empilha nó da esquerda (se diferente de NULL)

Percurso em Largura

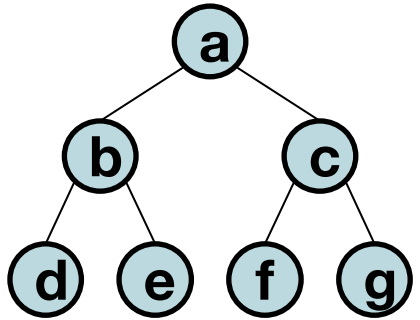
Implementação - Largura

Estrutura auxiliar necessária: fila

Adicionar a raiz na fila

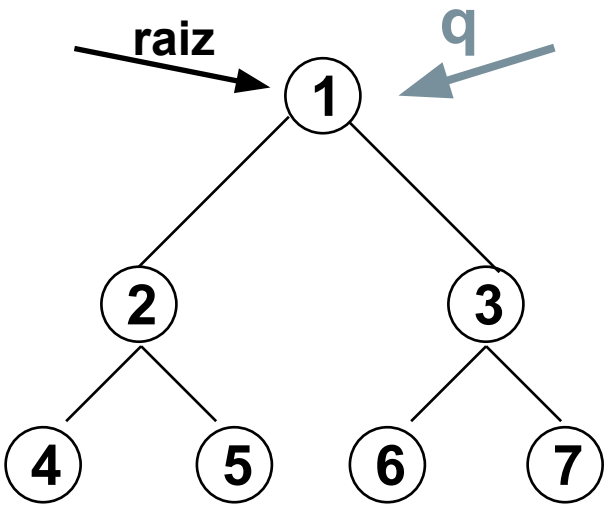
Repetir até que a fila fique vazia

1. Retirar primeiro da fila (visita)
2. Adicionar nó da esquerda na fila (se diferente de NULL)
3. Adicionar nó da direita na fila (se diferente de NULL)



a - b - c - d - e - f - g

PERCORRER EM LARGURA COM USO DE FILA

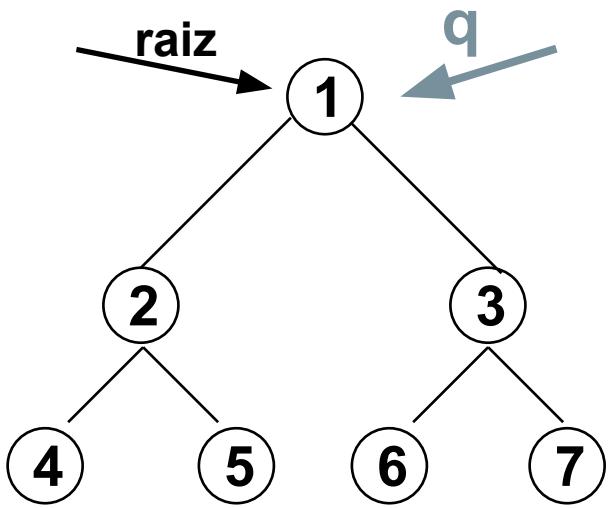


Fila

1

Caminhamento:

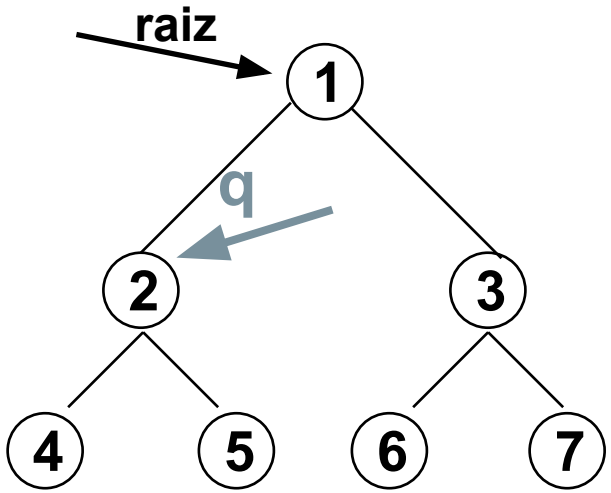
PERCORRER EM LARGURA COM USO DE FILA



Fila

Caminhamento: 1

PERCORRER EM LARGURA COM USO DE FILA

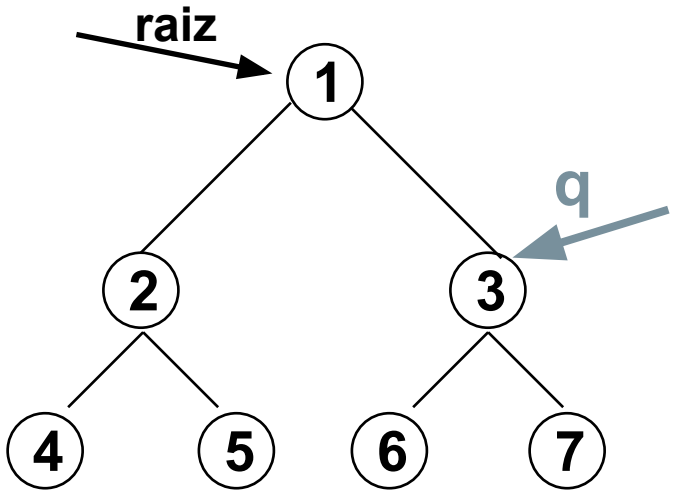


Fila

~~1~~	~~2~~	3	4	5

Caminhamento: 1 – 2

PERCORRER EM LARGURA COM USO DE FILA

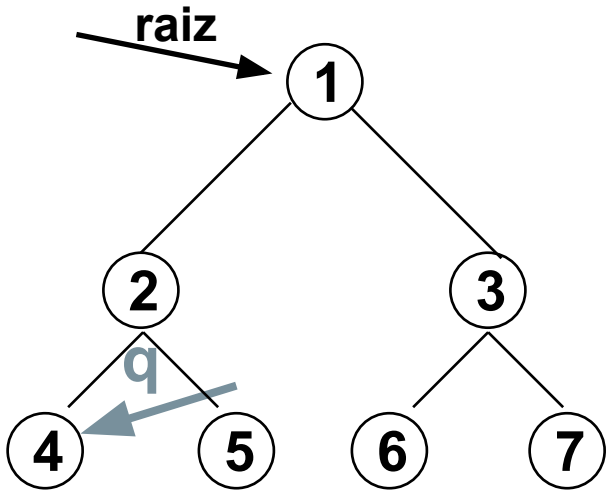


Fila

~~1~~	~~2~~	~~3~~	4	5	6	7

Caminhamento: 1 – 2 – 3

PERCORRER EM LARGURA COM USO DE FILA



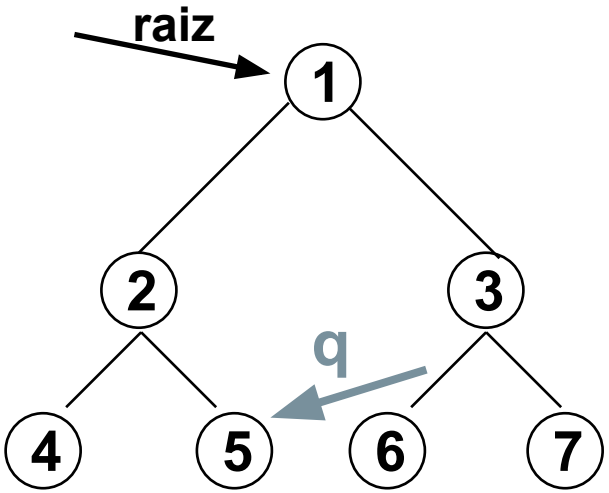
Fila

~~1~~	~~2~~	~~3~~	~~4~~	5	6	7

Filhos NULL não entram na fila

Caminhamento: 1 – 2 – 3 – 4

PERCORRER EM LARGURA COM USO DE FILA



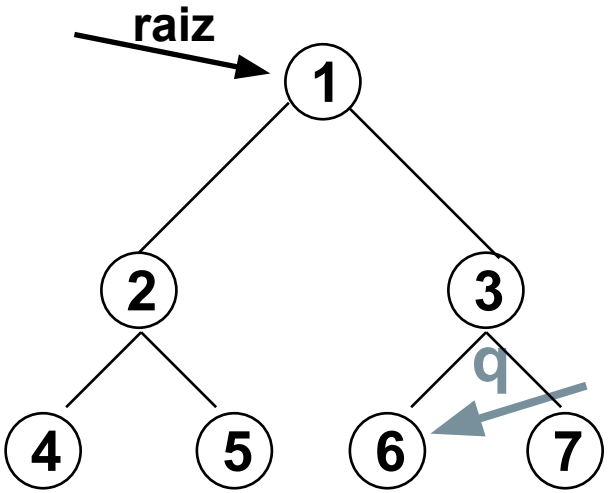
Fila

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	6	7

Filhos NULL não entram na fila

Caminhamento: 1 – 2 – 3 – 4 – 5

PERCORRER EM LARGURA COM USO DE FILA



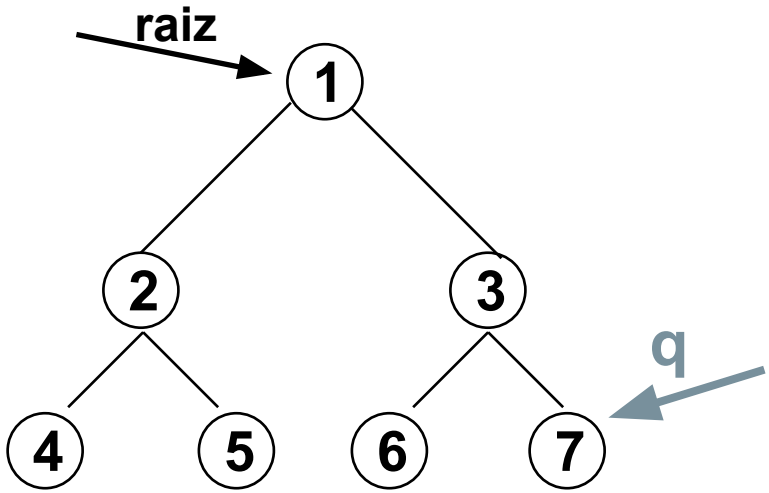
Fila

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	7

Filhos NULL não entram na fila

Caminhamento: 1 – 2 – 3 – 4 – 5 – 6

PERCORRER EM LARGURA COM USO DE FILA



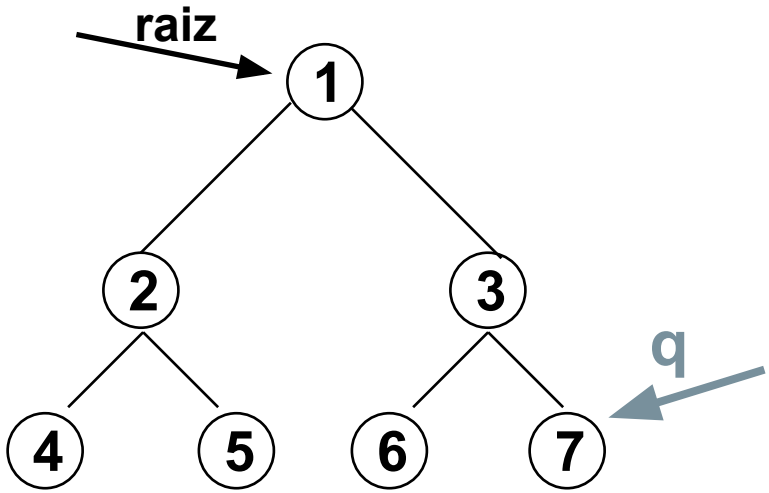
Fila

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~

Filhos NULL não entram na fila

Caminhamento: 1 – 2 – 3 – 4 – 5 – 6 – 7

PERCORRER EM LARGURA COM USO DE FILA



Fila

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~

Fila vazia: fim da execução

Caminhamento: 1 – 2 – 3 – 4 – 5 – 6 – 7

Algoritmo para percorrer em Largura

Largura

Estrutura auxiliar necessária: **fila**

Adicionar a raiz na fila

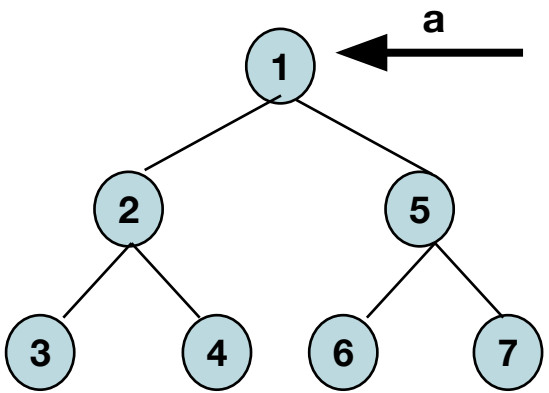
Repetir até que a fila fique vazia

1. Retirar primeiro da fila (visita)
2. Adicionar nó da esquerda na fila (se diferente de NULL)
3. Adicionar nó da direita na fila (se diferente de NULL)

Implementação recursiva

Implementação recursiva - Profundidade

```
void profundidade(TNoA* a)
{
      if (a!= NULL)
      {
            printf("%c\n",a->info);
            profundidade(a->esq);
            profundidade(a->dir);
       }
}
```



Profundidade: raiz, esquerda, direita

Intervalo

Provinha

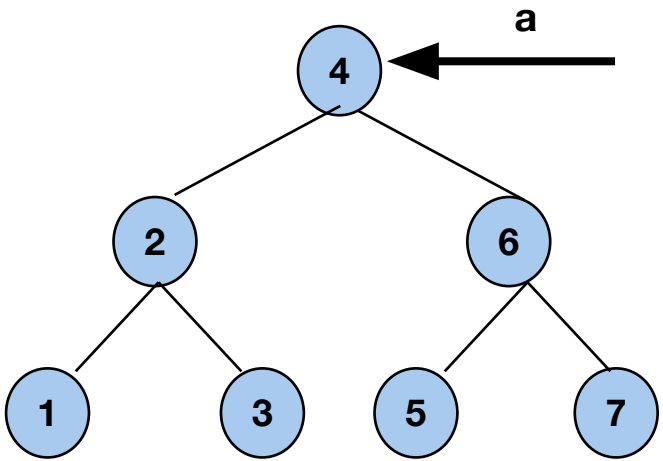
Provinha

Fazer as implementações recursivas para percorrer a árvore em ordem simétrica e pós-ordem

Fim

Implementação recursiva - Simétrica

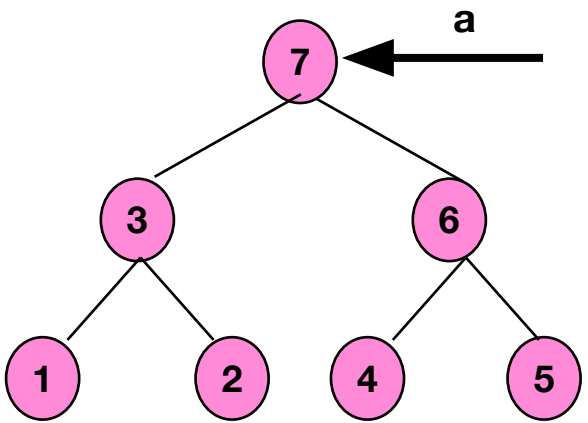
```
void simetrica(TNoA* a)
{
      if (a!= NULL)
      {
            simetrica(a->esq);
            printf("%c\n",a->info);
            simetrica(a->dir);
       }
}
```



Simétrica: esquerda, raiz, direita

Implementação recursiva - Pós-ordem

```
void posOrdem(TNoA* a)
{
      if (a!= NULL)
      {
            posOrdem(a->esq);
            posOrdem(a->dir);
            printf("%c\n",a->info);
       }
}
```



Pós-ordem: esquerda, direita, raiz