Se um café pela manhã não te acordar, execute o seguinte em um servidor em produção: `rm -rf --no-preserve-root /`

Busca em Largura e Profundidade

Paulo Ricardo Lisboa de Almeida

DSBD

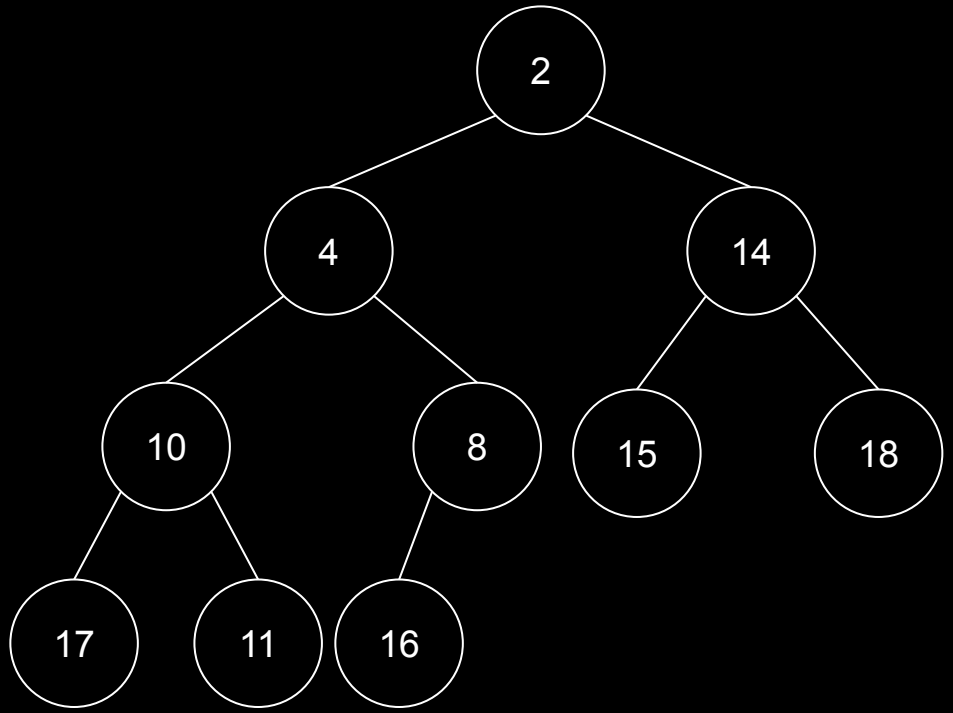
UFPR

Relembrando ...

Travessia em **Pré-Ordem**

- Visitar a raiz.
- Aplicar pré-ordem na subárvore esquerda.
- Aplicar pré-ordem na subárvore direita.

Como fica a travessia da árvore do exemplo?

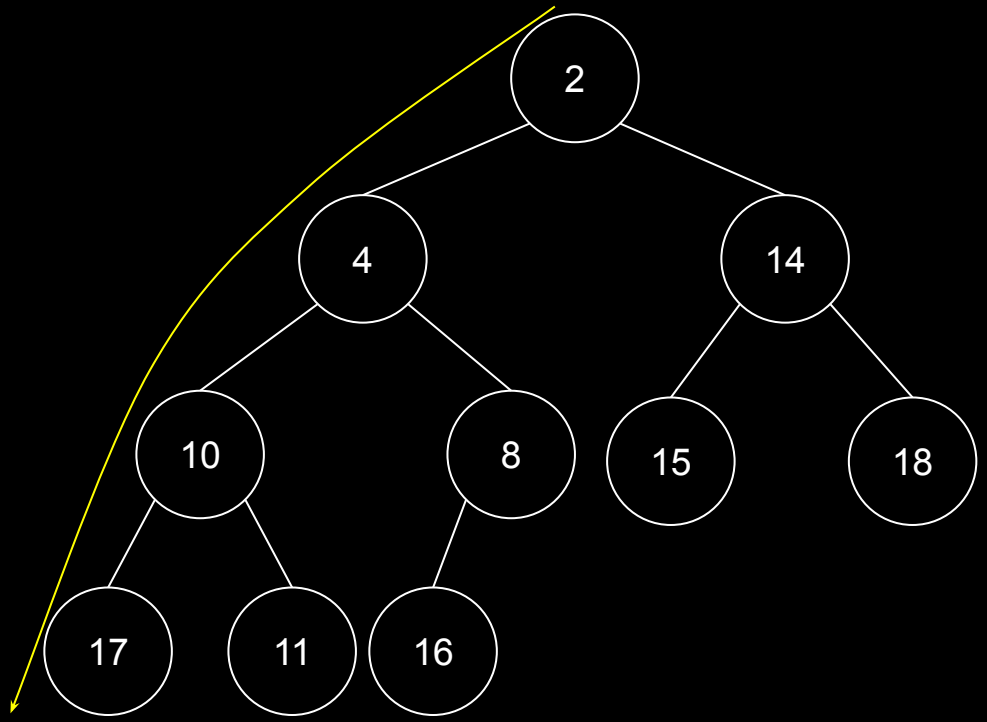


Pré-Ordem

Em pré-ordem buscamos os ramos mais profundos da esquerda para a direita.

Essa técnica para percorrer a árvore é conhecida por **percurso em profundidade** ou ***depth-first***.

```
void preordem(struct nodo* no){
   if (no !=NULL){
       printf(no->valor);
       preordem(no->esq);
       preordem(no->dir);
   }
}
```



Visita em Pré-ordem: 2, 4, 10, 17, 11, 8, 16, 14, 15, 18.

Transformando em uma busca

Suponha que desejamos **retornar o nodo que contém determinado valor**.

Como fica o protótipo e a implementação da função?

```
void preordem(struct nodo* no){
   if (no !=NULL){
       printf(no->valor);
       preordem(no->esq);
       preordem(no->dir);
   }
}
```

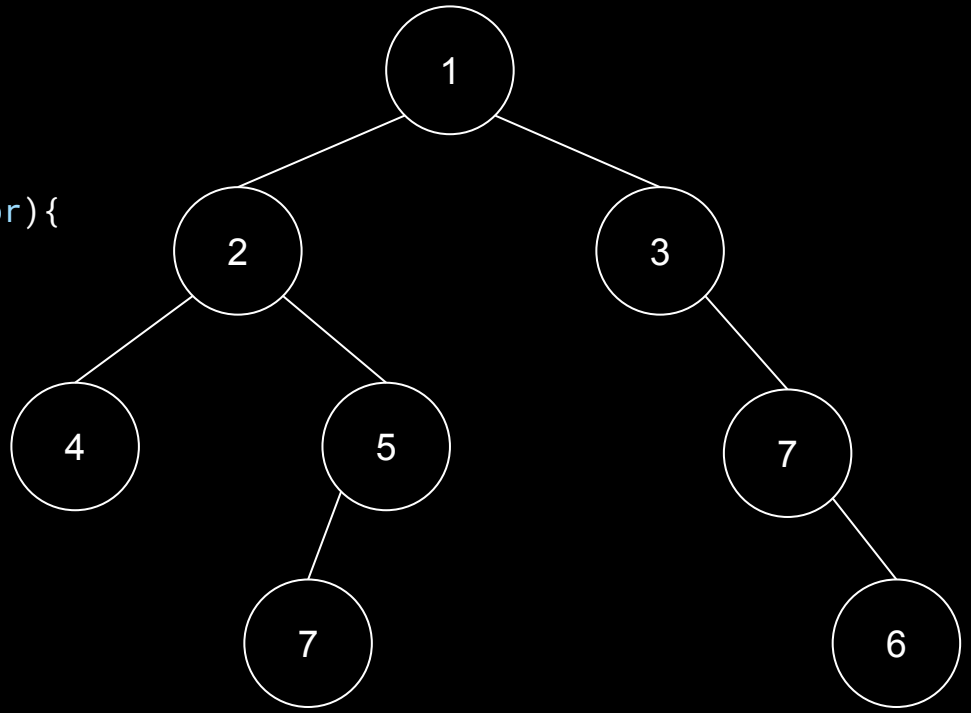
Busca em Profundidade

Exemplo de implementação de uma Busca em Profundidade - Depth-First Search (**DFS**).

```
struct nodo* dfs(struct nodo* nodo, int valor){
   if(nodo == NULL)
       return NULL;

   if(nodo->valor == valor)
       return nodo;

   struct nodo* retorno;
   retorno = dfs(nodo->fe, valor);
   if(retorno)
       return retorno;
   retorno = dfs(nodo->fd, valor);

   return retorno;
}
```

Busca em Profundidade

```c
struct nodo* dfs(struct nodo* nodo, int valor){
    if(nodo == NULL)
        return NULL;

    if(nodo->valor == valor)
        return nodo;

    struct nodo* retorno;
    retorno = dfs(nodo->fe, valor);
    if(retorno)
        return retorno;
    retorno = dfs(nodo->fd, valor);

    return retorno;
}
```



Quantas comparações são feitas para as chamadas a seguir?

```c
dfs(raiz, 5);
dfs(raiz, 7);
dfs(raiz, 18);
```

Removendo a recursão

Como remover a recursão?

```
struct nodo* dfs(struct nodo* nodo, int valor){
    if(nodo == NULL)
        return NULL;

    if(nodo->valor == valor)
        return nodo;

    struct nodo* retorno;
    retorno = dfs(nodo->fe, valor);
    if(retorno)
        return retorno;
    retorno = dfs(nodo->fd, valor);

    return retorno;
}
```

DFS Iterativo

```
função dfsIterativo(r,v)
entrada: nodo raiz da árvore binária r, e um valor a ser encontrado v.
saída: o primeiro nodo encontrado que contém v como chave, ou não caso tal nodo não
exista
empilhar(r)
enquanto pilha não vazia
     n = desempilhar()
     se n.valor == v
          retorne n

     empilhar(???)
retorne não
```

DFS Iterativo

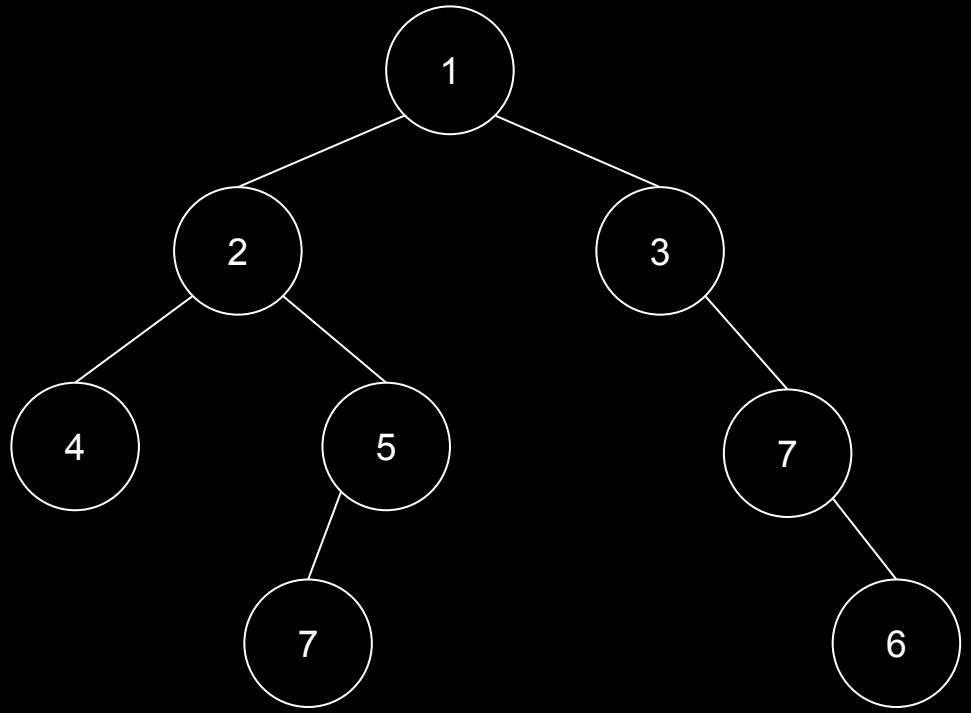
```
função dfsIterativo(r,v)
entrada: nodo raiz da árvore binária r, e um valor a ser encontrado v.
saída: o primeiro nodo encontrado que contém v como chave, ou não caso tal nodo não
exista
empilhar(r)
enquanto pilha não vazia
     n = desempilhar()
     se n.valor == v
          retorne n
     se n possui filho direito
          empilhar(n.filhoDireito)
     se n possui filho esquerdo
          empilhar(n.filhoEsquerdo)
retorne não
```

Faça você mesmo

Faça o teste de mesa para as chamadas

```
dfsIterativo(raiz,3)
dfsIterativo(raiz,14)
```

```
função dfsIterativo(r,v)
empilhar(r)
enquanto pilha não vazia
     n = desempilhar()
     se n.valor == v
          retorne n
     se n possui filho direito
          empilhar(n.filhoDireito)
     se n possui filho esquerdo
          empilhar(n.filhoEsquerdo)
retorne não
```



Percurso em Largura

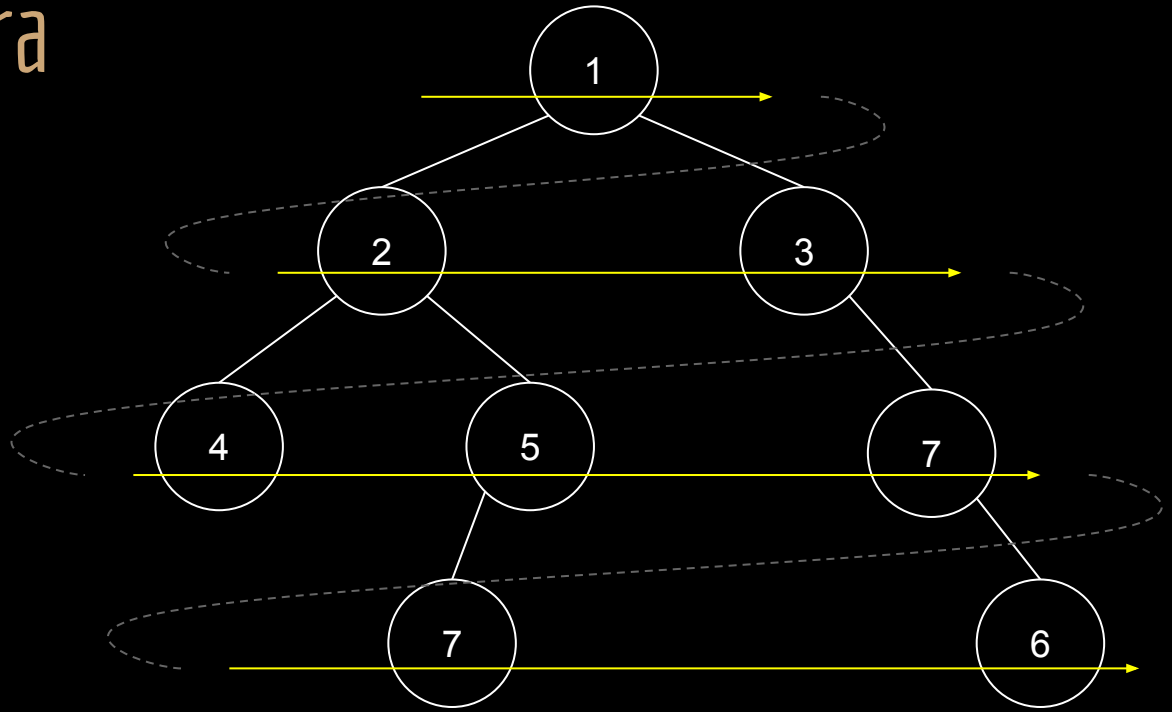
Percurso em Largura

Inicia na raiz;

Visita todos os nodos com distância 1;

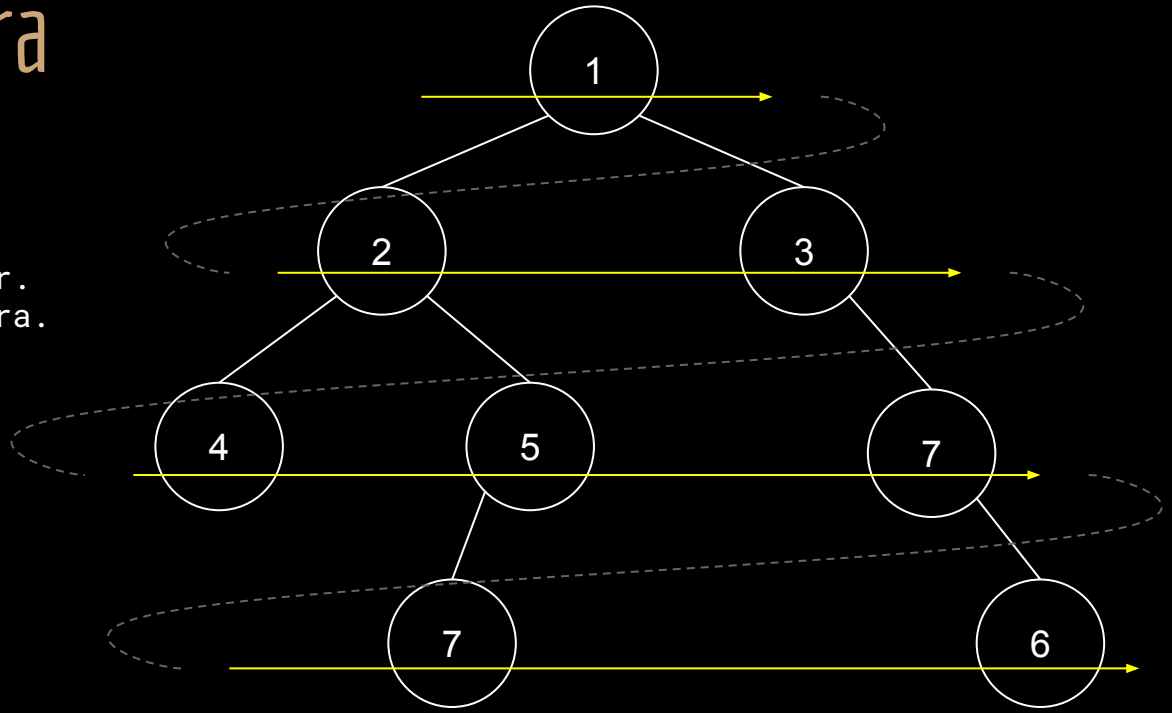
Visita todos os nodos com distância 2;

...



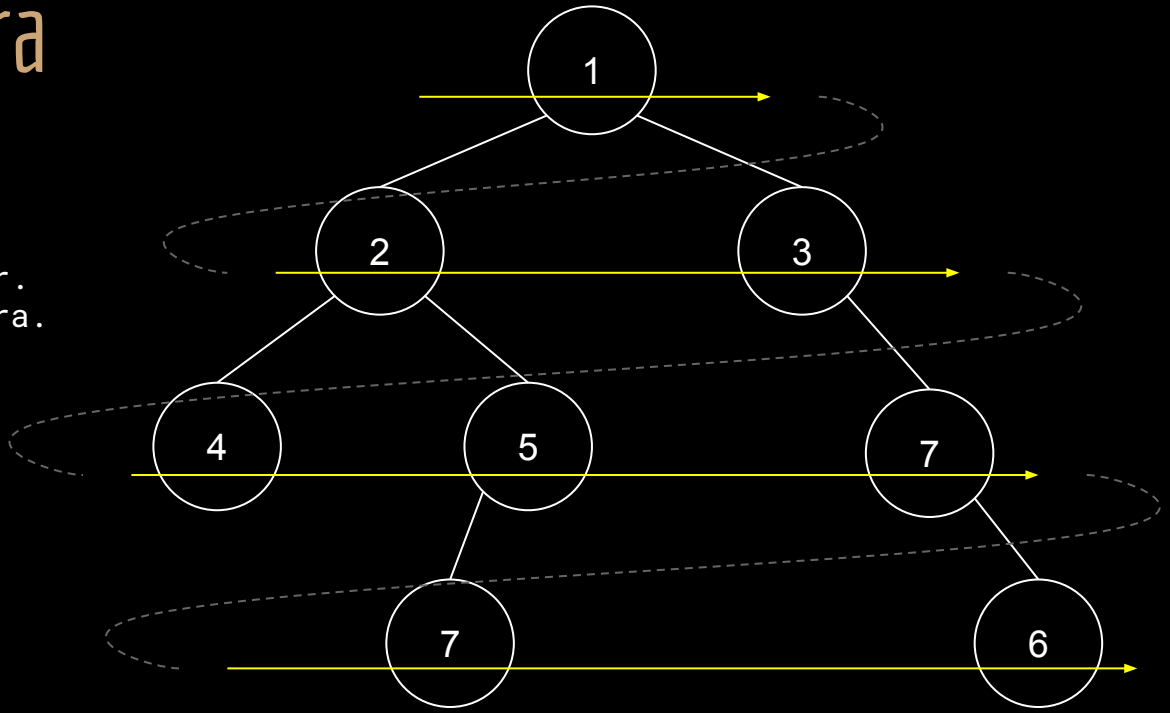
Percurso em Largura

```
função percursoLargura(r)
entrada: nodo raiz da árvore binária r.
saída: a árvore é percorrida em largura.
enfileirar(r)
enquanto fila não vazia
     n = removerCabeça()
     imprimir(n)

     enfileirar(???)

retorne
```



Percurso em Largura

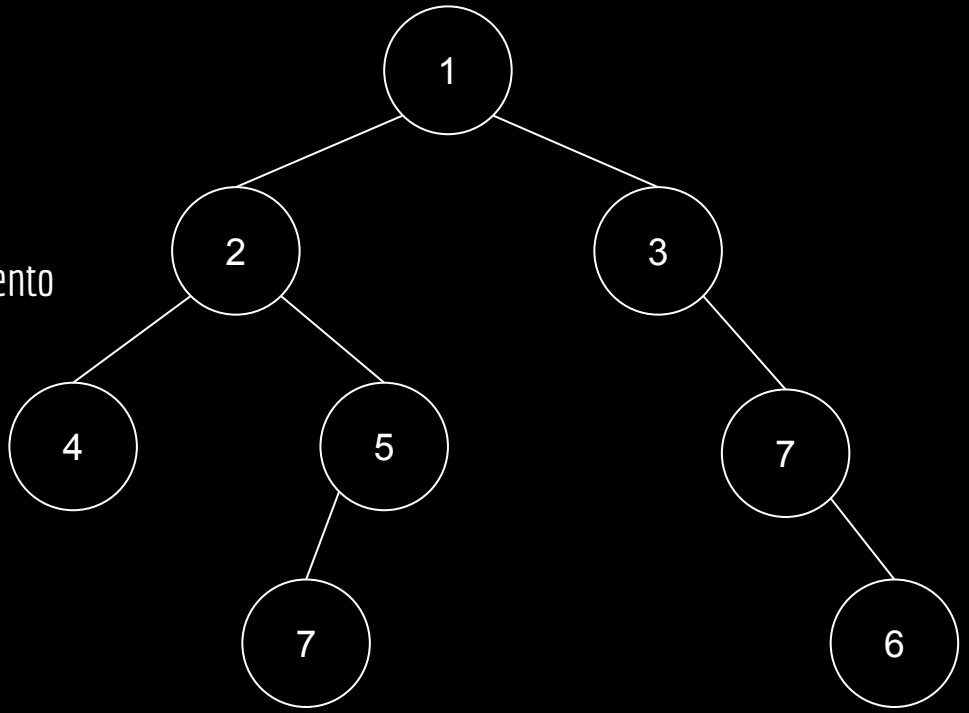
```
função percursoLargura(r)
entrada: nodo raiz da árvore binária r.
saída: a árvore é percorrida em largura.
enfileirar(r)
enquanto fila não vazia
     n = removerCabeça()
     imprimir(n)
     se n possui filho esquerdo
          enfileirar(n.filhoEsquerdo)
     se n possui filho direito
          enfileirar(n.filhoDireito)


retorne
```



Busca em Largura

Busca em Largura - Breadth-first search (BFS).

Percorremos a árvore em largura, até encontrar o elemento procurado.



Quantas comparações são feitas para as chamadas a seguir?

```
bfs(raiz, 5);
bfs(raiz, 7);
bfs(raiz, 18);
```

Busca em Largura

Como transformar o algoritmo do percurso em largura para realizar uma busca em largura?

```
função percursoLargura(r)
entrada: nodo raiz da árvore binária r.
saída: a árvore é percorrida em largura.
enfileirar(r)
enquanto fila não vazia
     n = removerCabeça()
     imprimir(n)
     se n possui filho esquerdo
          enfileirar(n.filhoEsquerdo)
     se n possui filho direito
          enfileirar(n.filhoDireito)


retorne
```

Busca em Largura

Como transformar o algoritmo do percurso em largura para realizar uma busca em largura?

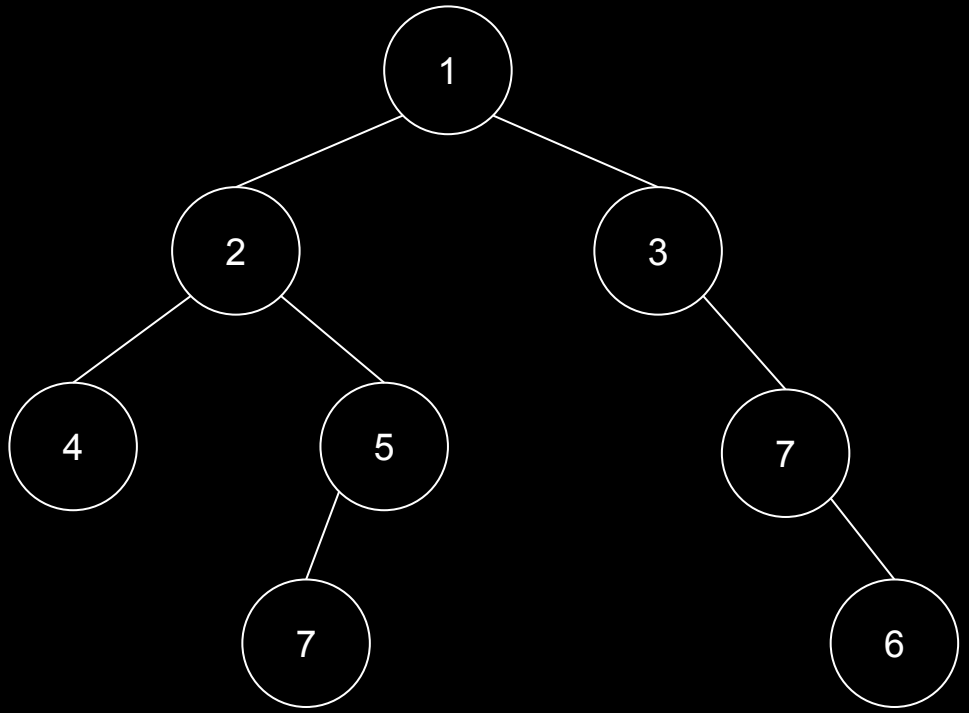
```
função bfs(r,v)
entrada: nodo raiz da árvore binária r, e um valor a ser encontrado v.
saída: o primeiro nodo encontrado que contém v como chave, ou não caso tal nodo não exista
enfileirar(r)
enquanto fila não vazia
     n = removerCabeça()
     se n.valor == v
          retorne n
     se n possui filho esquerdo
          enfileirar(n.filhoEsquerdo)
     se n possui filho direito
          enfileirar(n.filhoDireito)

retorne não
```

Faça você mesmo

Faça o teste de mesa para as chamadas

```
bfs(raiz,7)
bfs(raiz,14)
```

```
função bfs(r,v)
enfileirar(r)
enquanto fila não vazia
     n = removerCabeça()
     se n.valor == v
          retorne n
     se n possui filho esquerdo
          enfileirar(n.filhoEsquerdo)
     se n possui filho direito
          enfileirar(n.filhoDireito)

retorne não
```



Dica

Você pode usar uma lista encadeada como uma fila.

Existe uma implementação de lista encadeada pronta no kernel.

```
#include <sys/queue.h>
```

Veja detalhes no manual.

```
man queue
```

Comparando

Busca em Profundidade - DFS	Busca em Largura - BFS
Acha o item.	Acha o item com o melhor caminho (menor altura).
Busca descendo/subindo a árvore (pilha).	Busca enfileirando filhos (fila).
Problema com caminhos longos.	Problema se os graus dos nodos são altos.

Um adendo

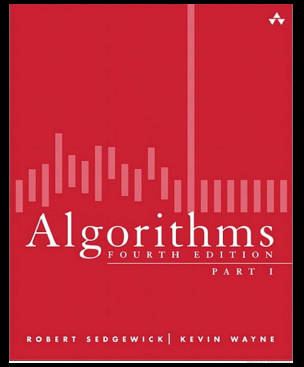
Os algoritmos apresentados podem ser facilmente adaptados para árvores que não sejam binárias.

Exercícios

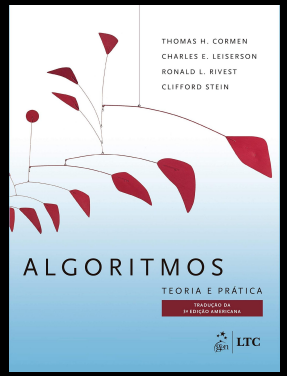
1. Implemente todos os algoritmos discutidos na aula em C.
2. Considere que o gasto máximo de memória do DFS e BFS acontece quando a pilha/fila possui a maior quantidade de elementos armazenados. Desenhe uma árvore e busque um elemento de forma que:
    a. O DFS gaste mais memória que o BFS
    b. O BFS gaste mais memória que o DFS

Referências

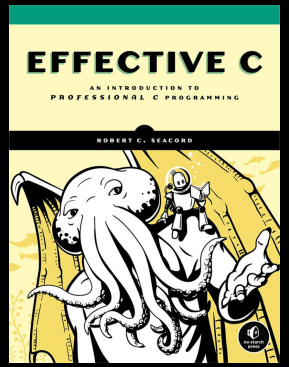
R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2011



T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 3a ed. 2012.



Seacord, R. C. Effective C: An ntroduction to Professional C Programming. 2020.



Licença