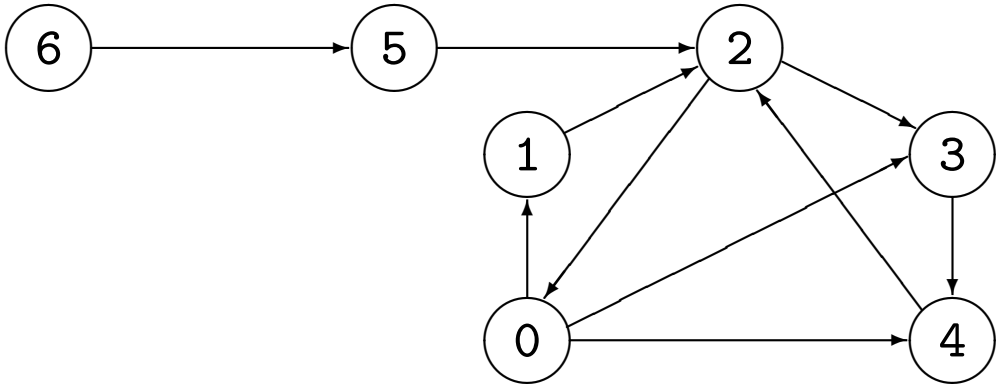
Capítulo III

Percurso em Profundidade
(num grafo orientado ou não orientado)

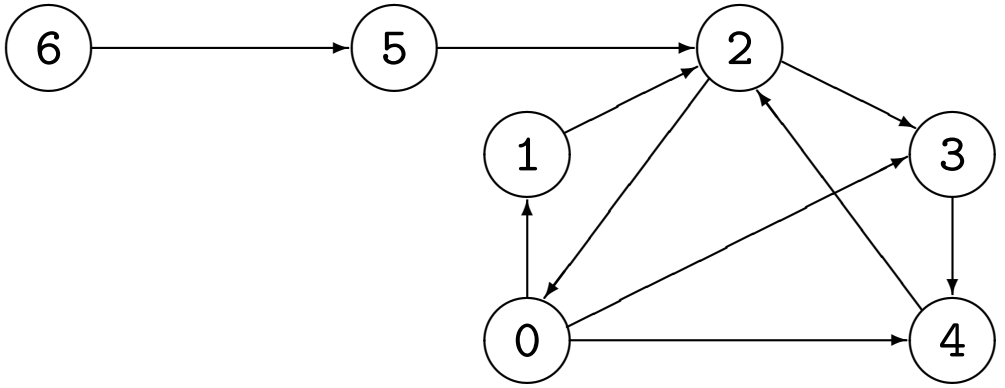
Percurso em Profundidade



Percurso em Profundidade

0

Ordem:

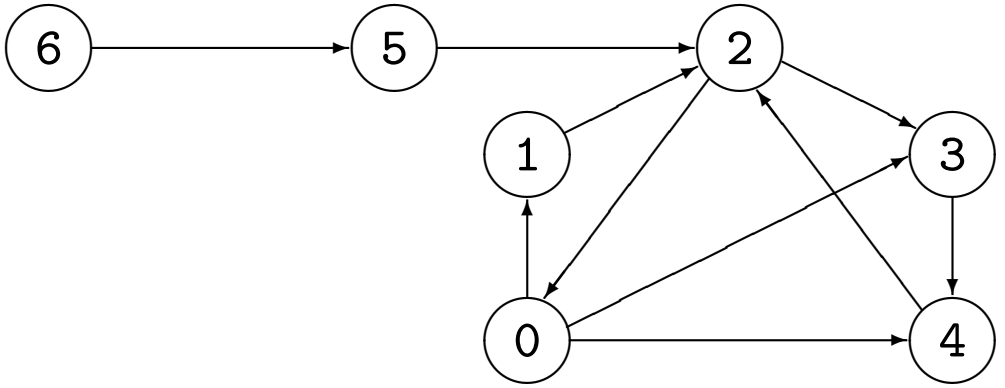


Percurso em Profundidade

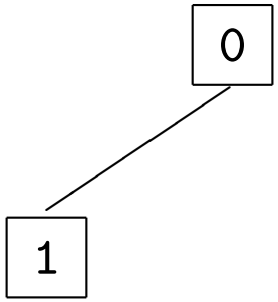
0

Ordem:

0

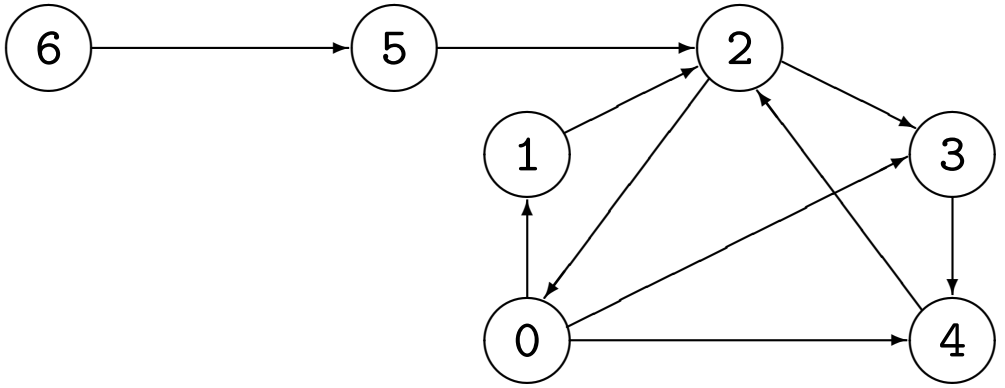


Percurso em Profundidade

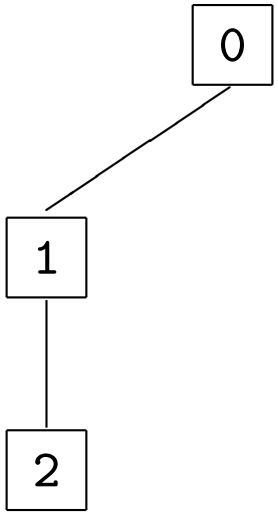


Ordem:

0 1

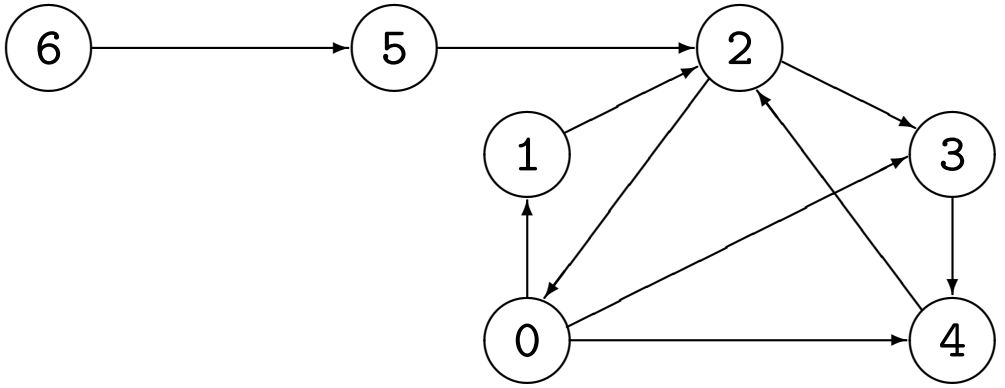


Percurso em Profundidade

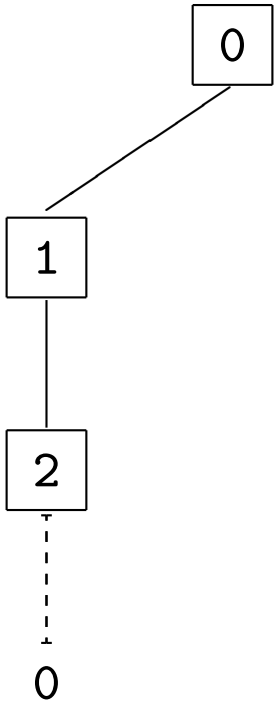


Ordem:

0 1 2

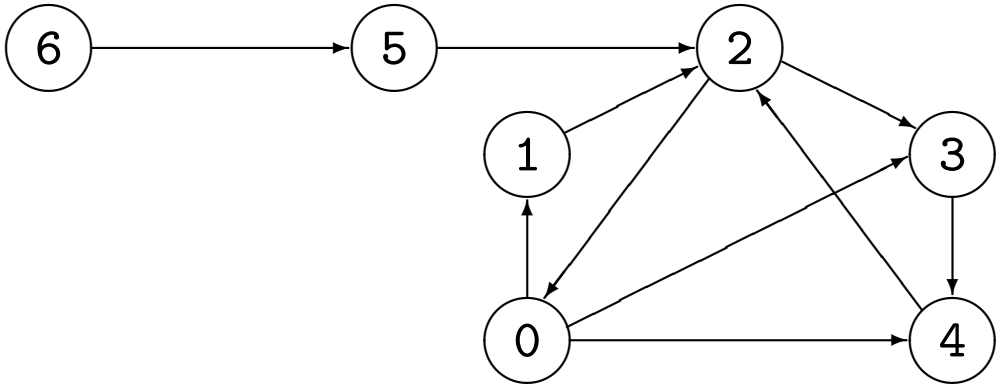


Percurso em Profundidade

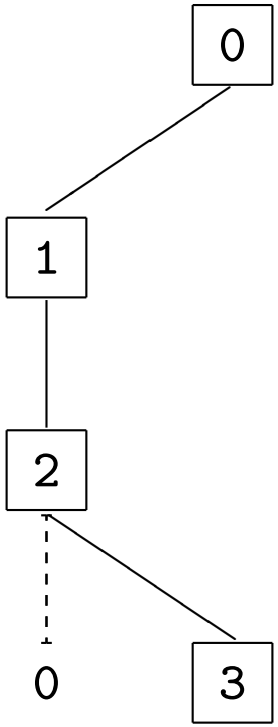


Ordem:

0 1 2

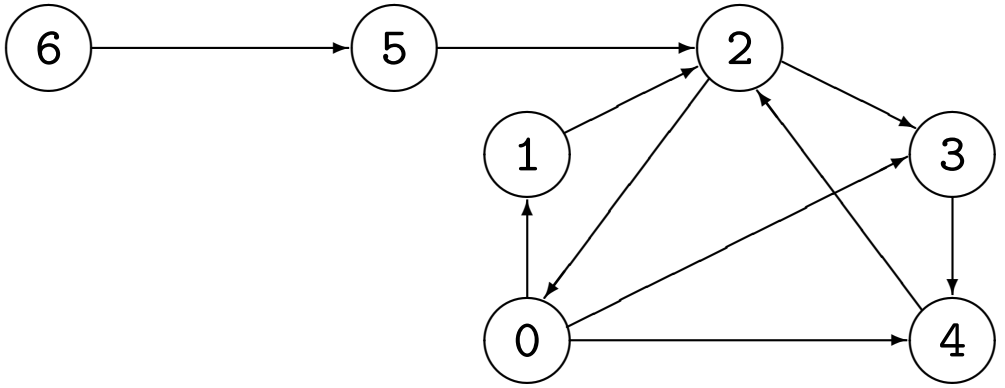


Percurso em Profundidade

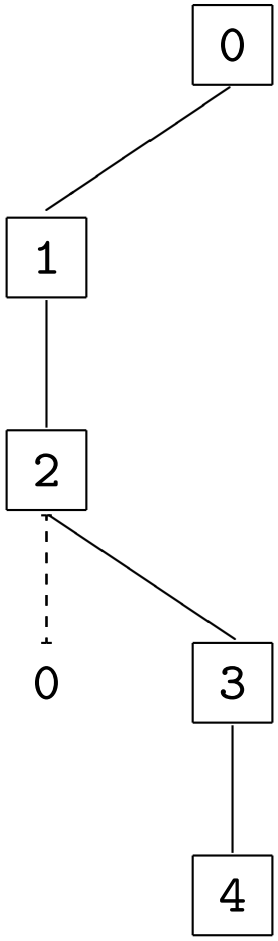


Ordem:

0 1 2 3

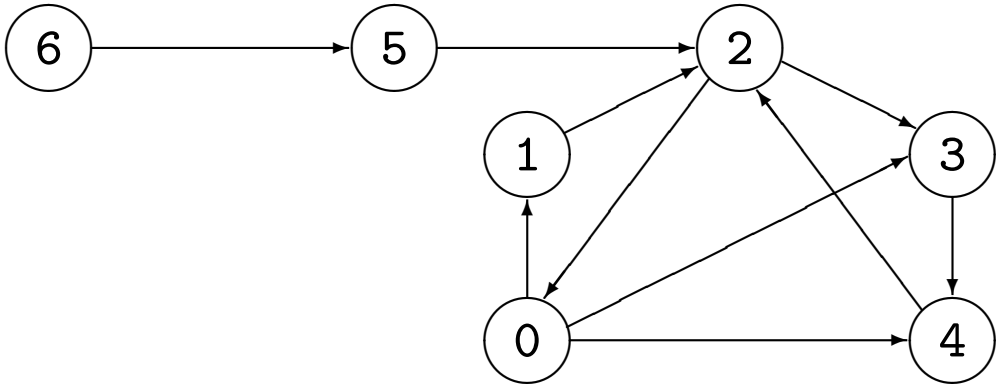


Percurso em Profundidade

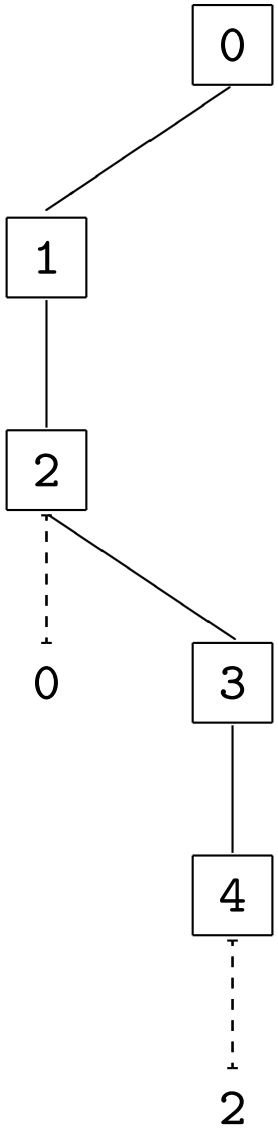


Ordem:

0 1 2 3 4

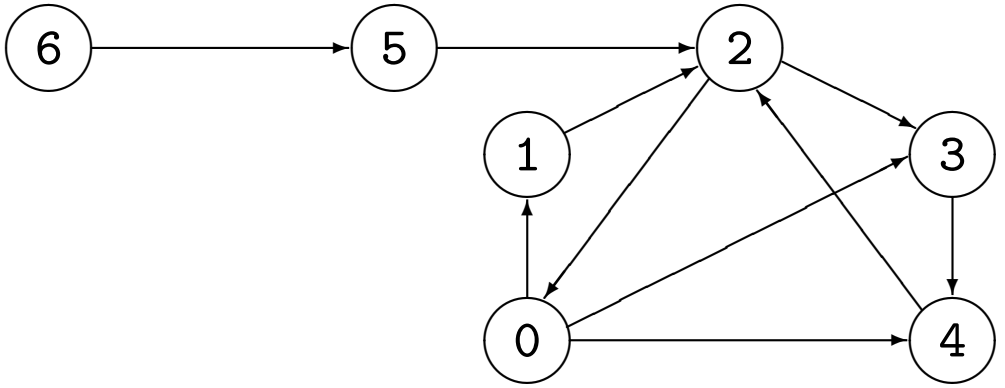


Percurso em Profundidade

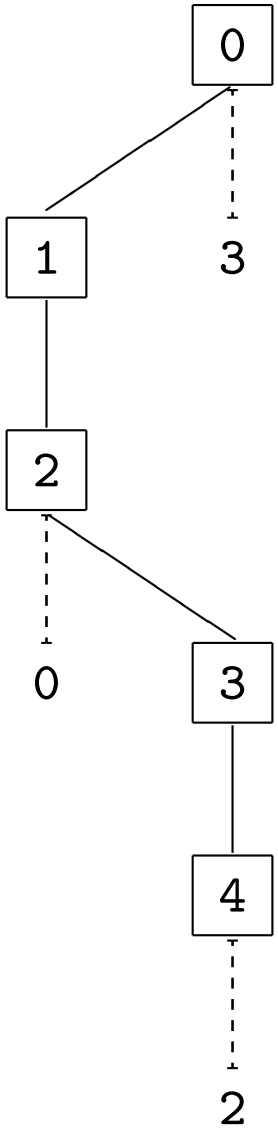


Ordem:

0 1 2 3 4

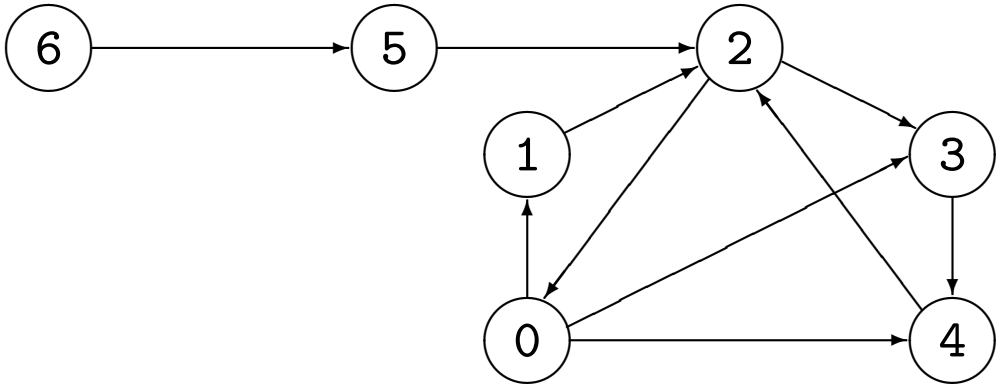


Percurso em Profundidade

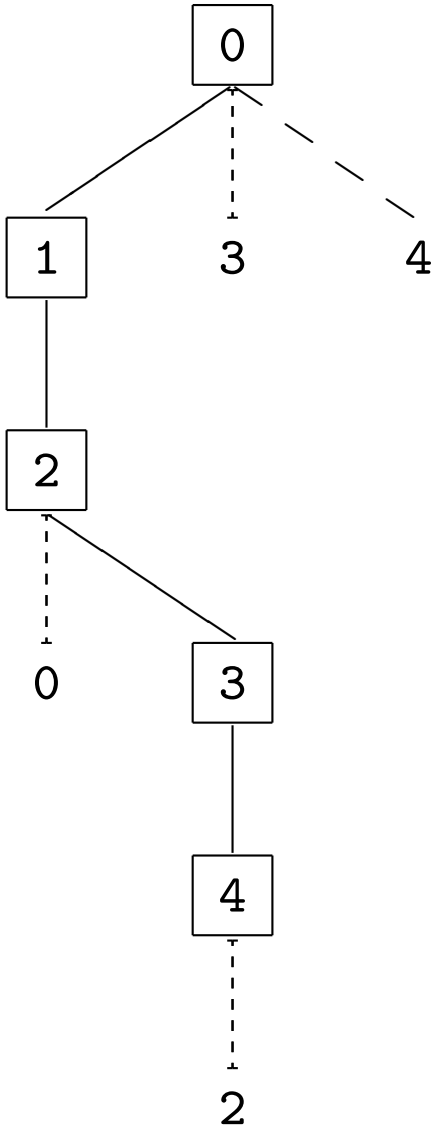


Ordem:

0 1 2 3 4

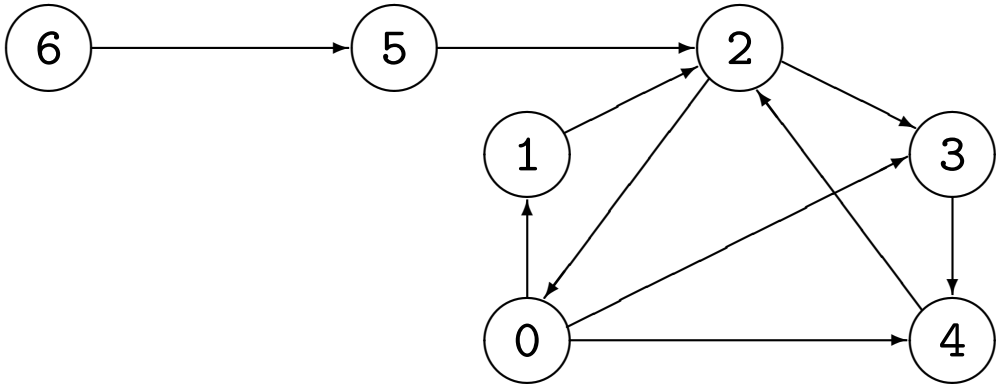


Percurso em Profundidade

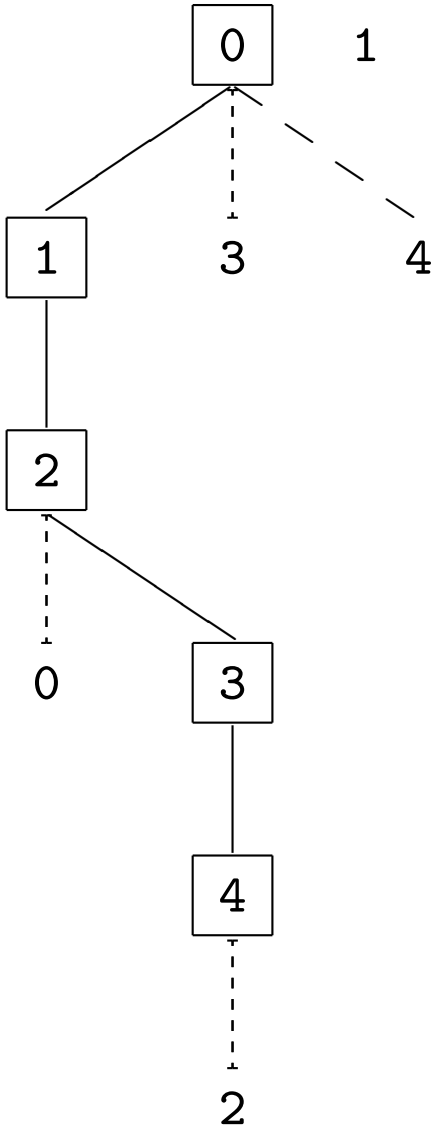


Ordem:

0 1 2 3 4

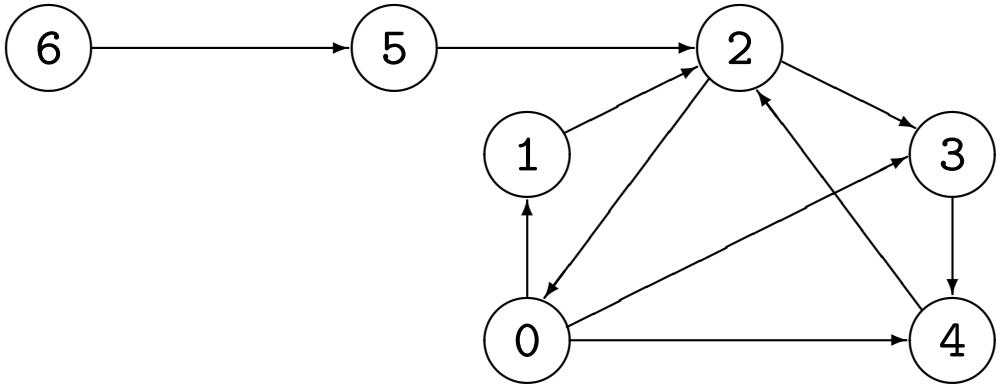


Percurso em Profundidade

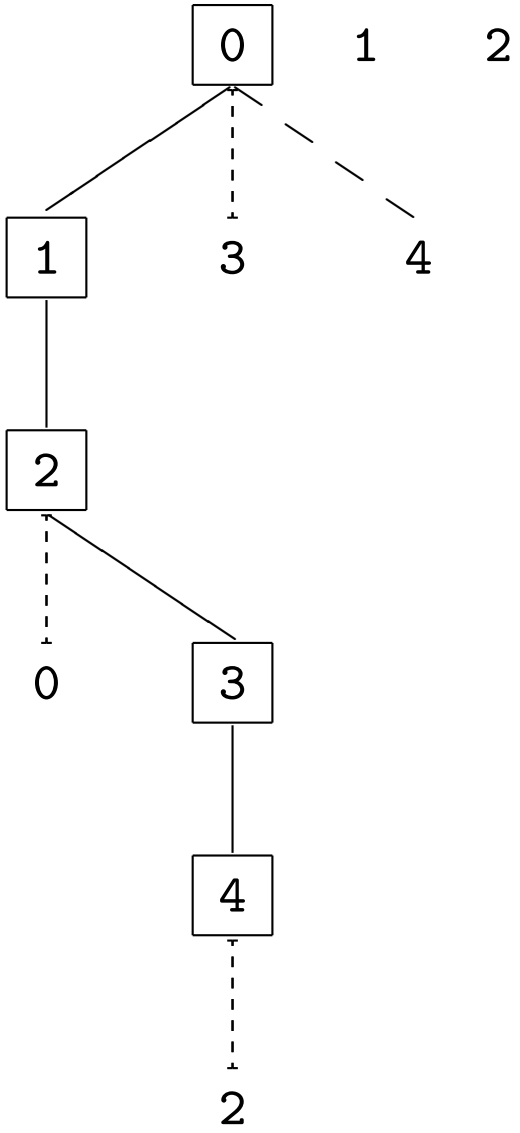


Ordem:

0 1 2 3 4

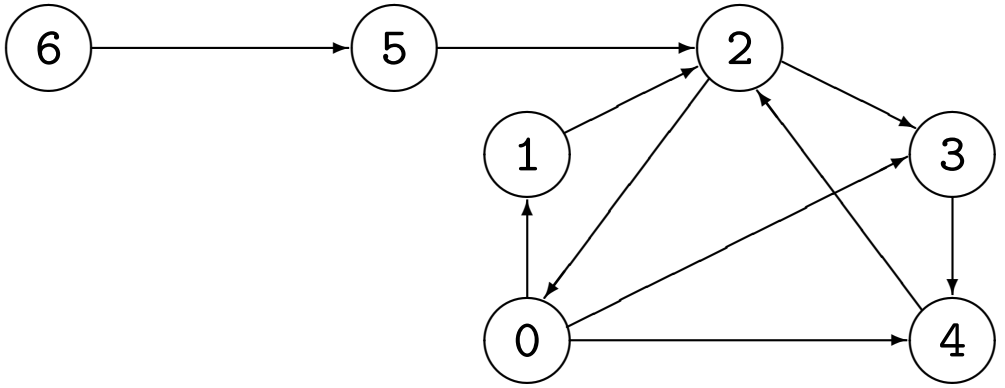


Percurso em Profundidade

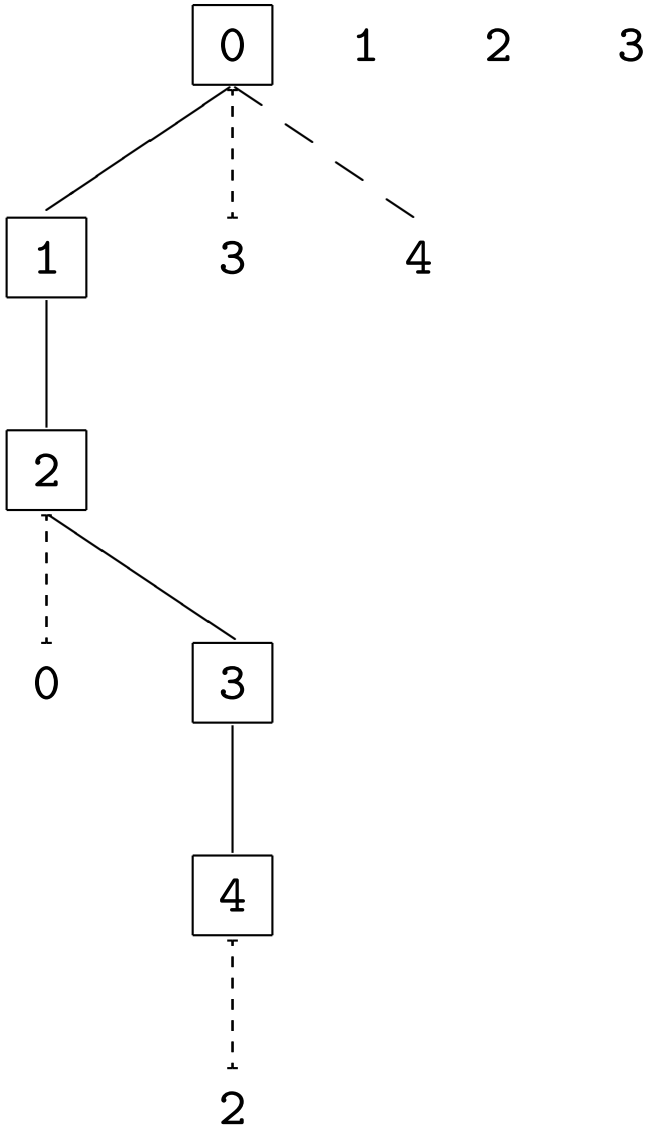


Ordem:

0 1 2 3 4

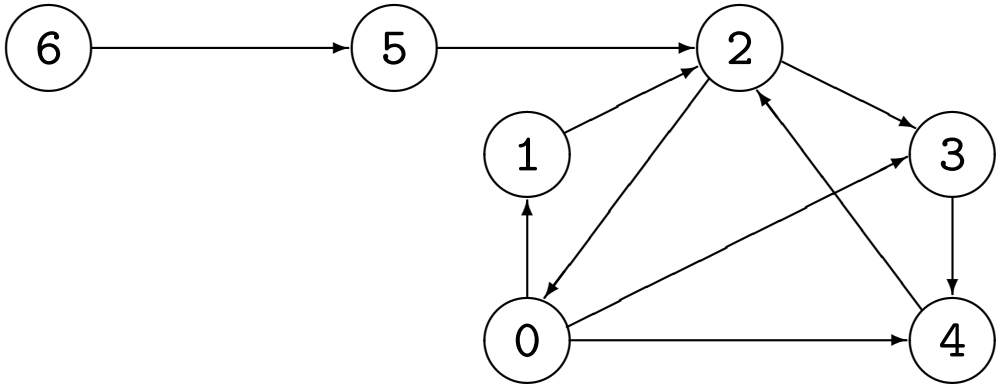


Percurso em Profundidade

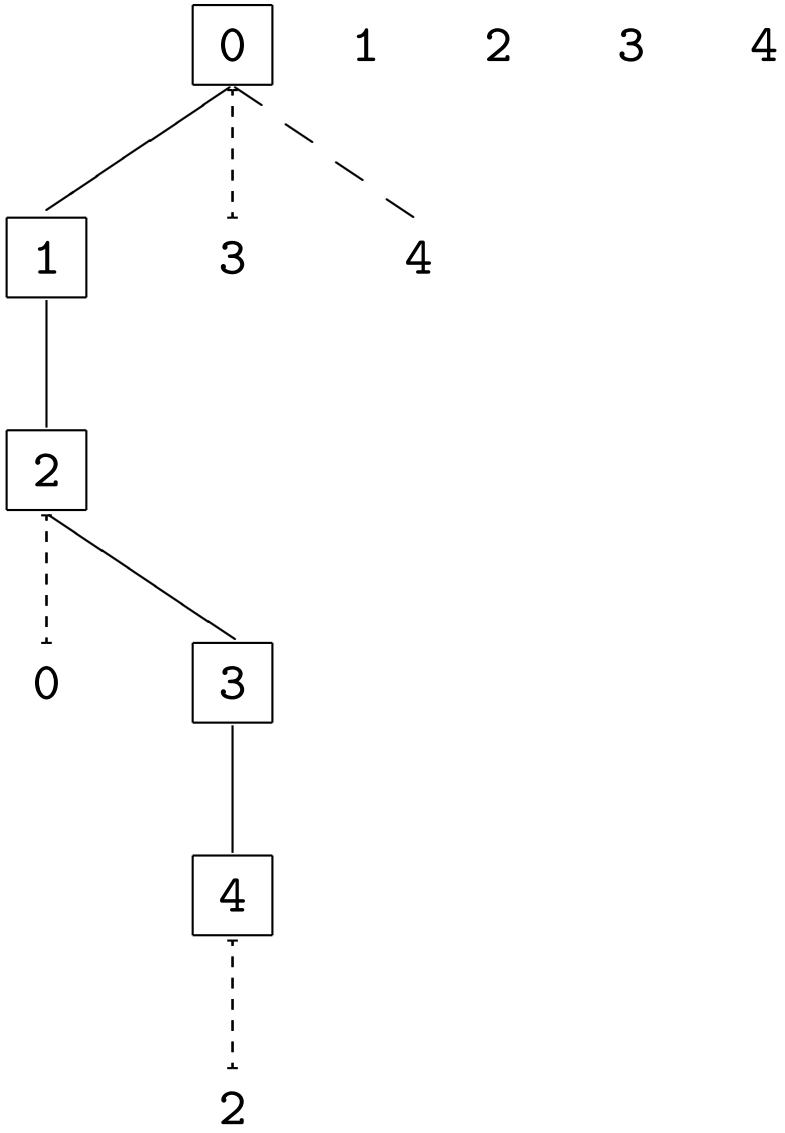


Ordem:

0 1 2 3 4

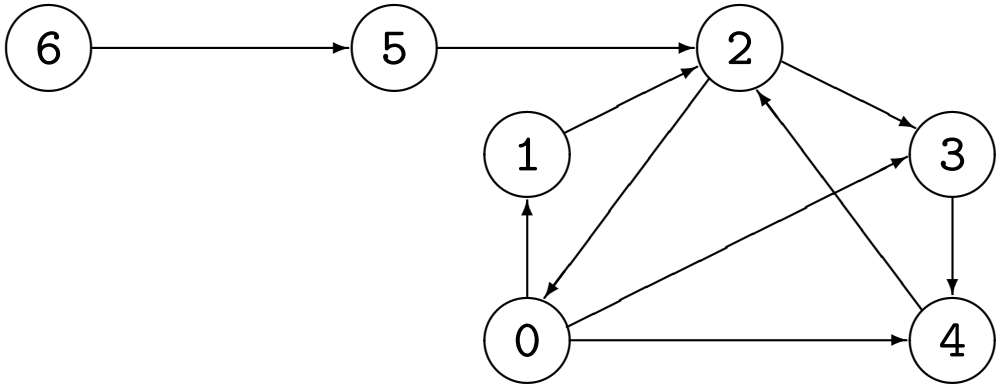


Percurso em Profundidade

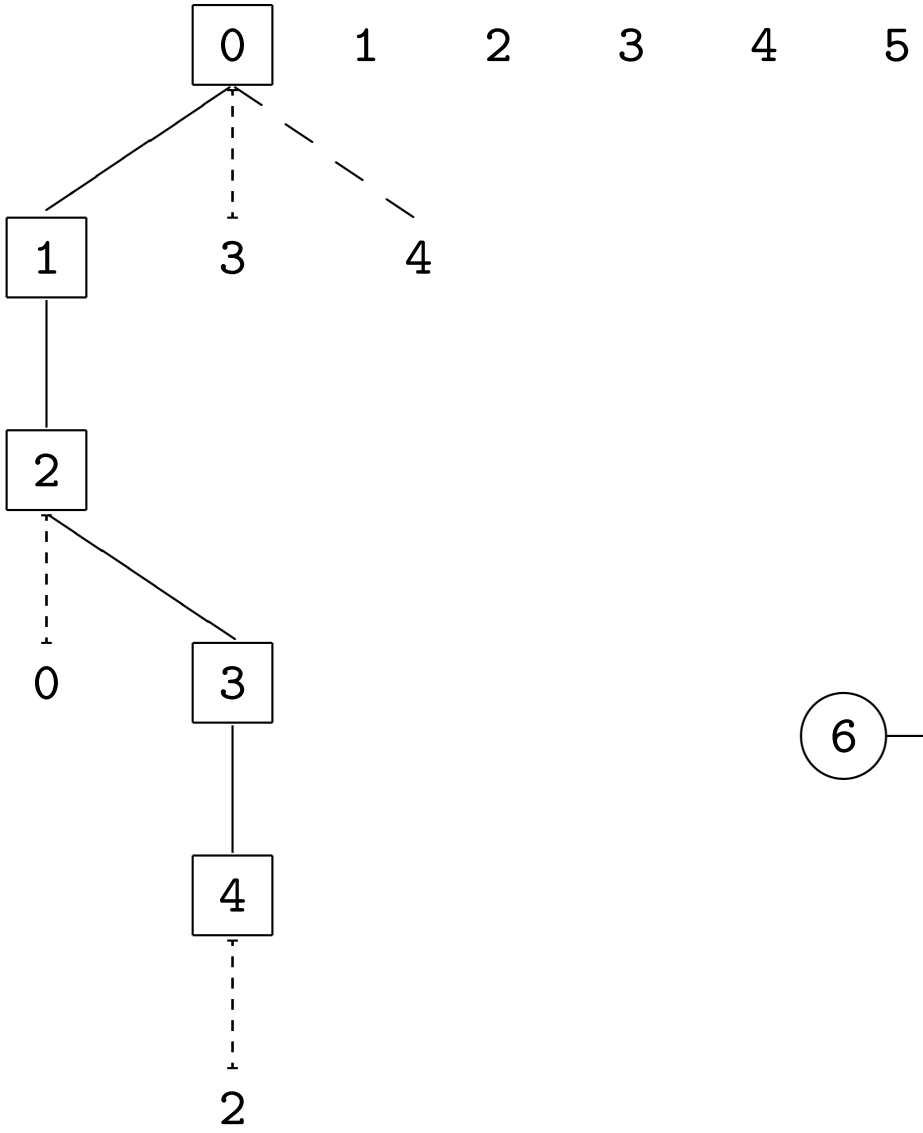


Ordem:

0 1 2 3 4

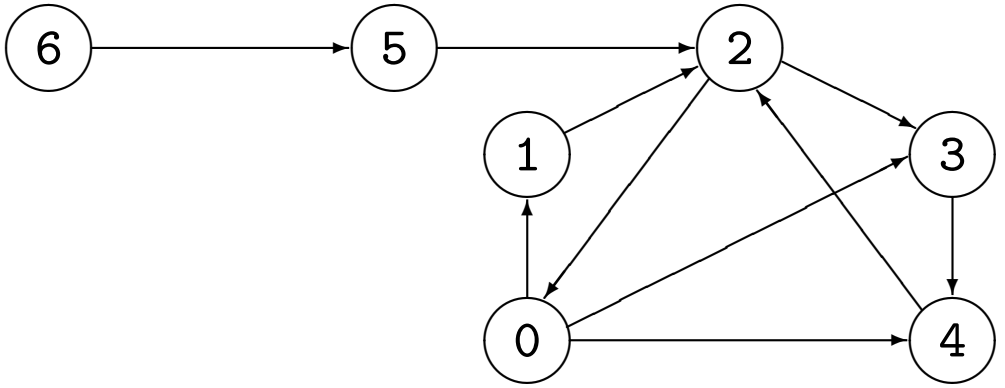


Percurso em Profundidade

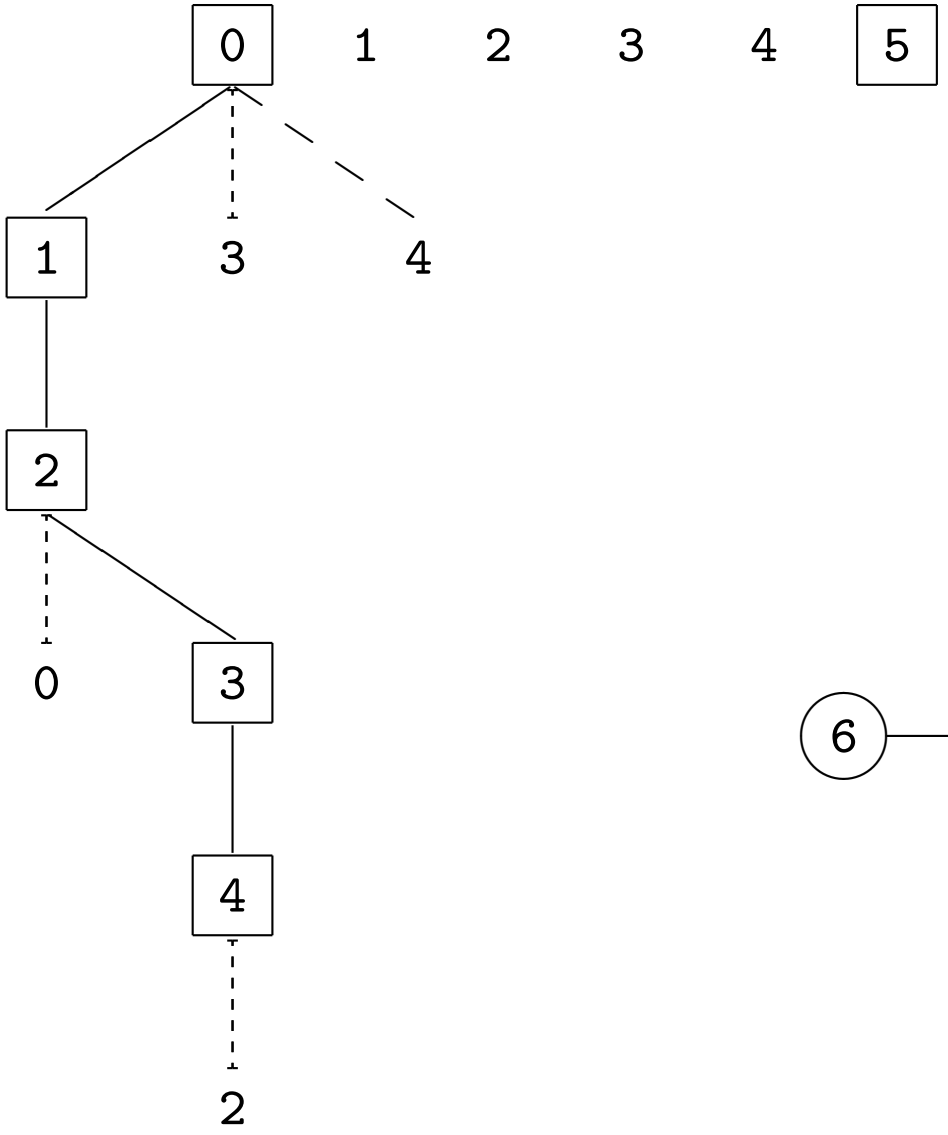


Ordem:

0 1 2 3 4

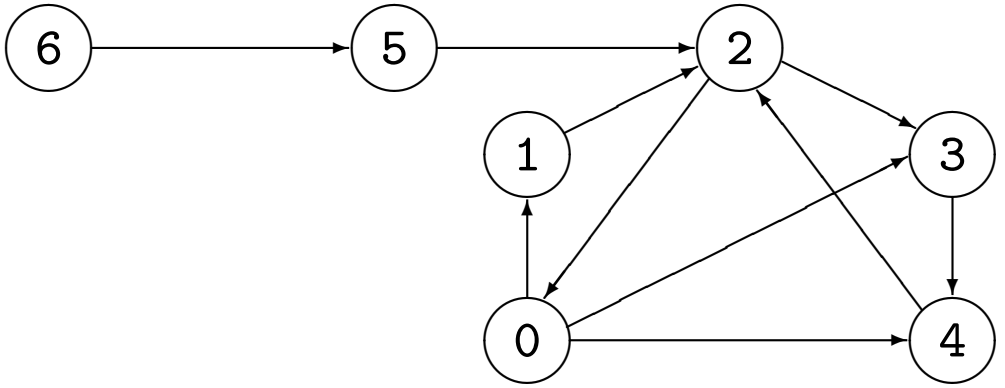


Percurso em Profundidade

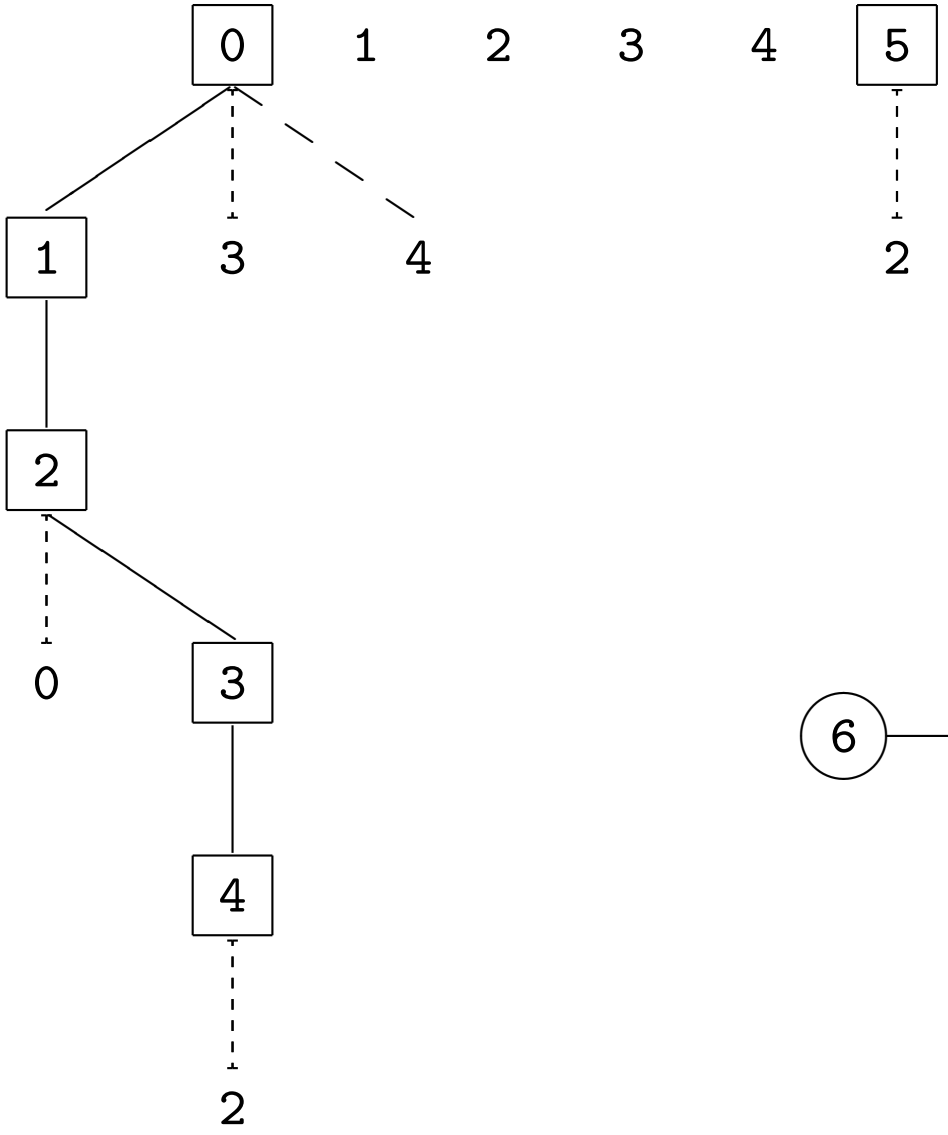


Ordem:

0 1 2 3 4 5

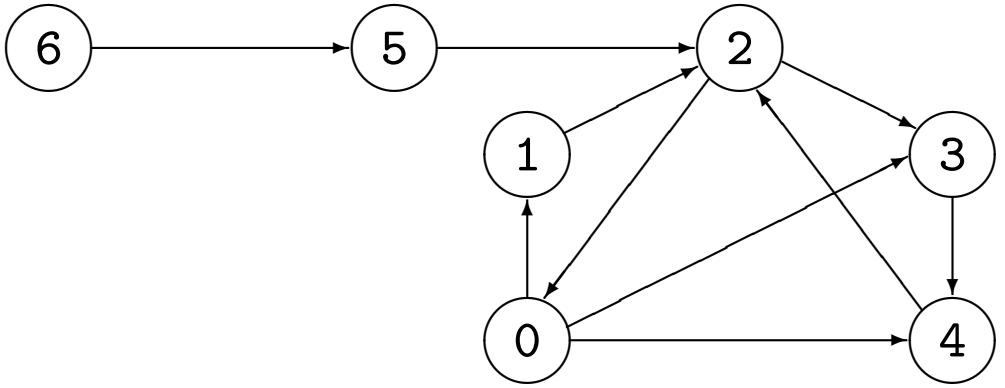


Percurso em Profundidade

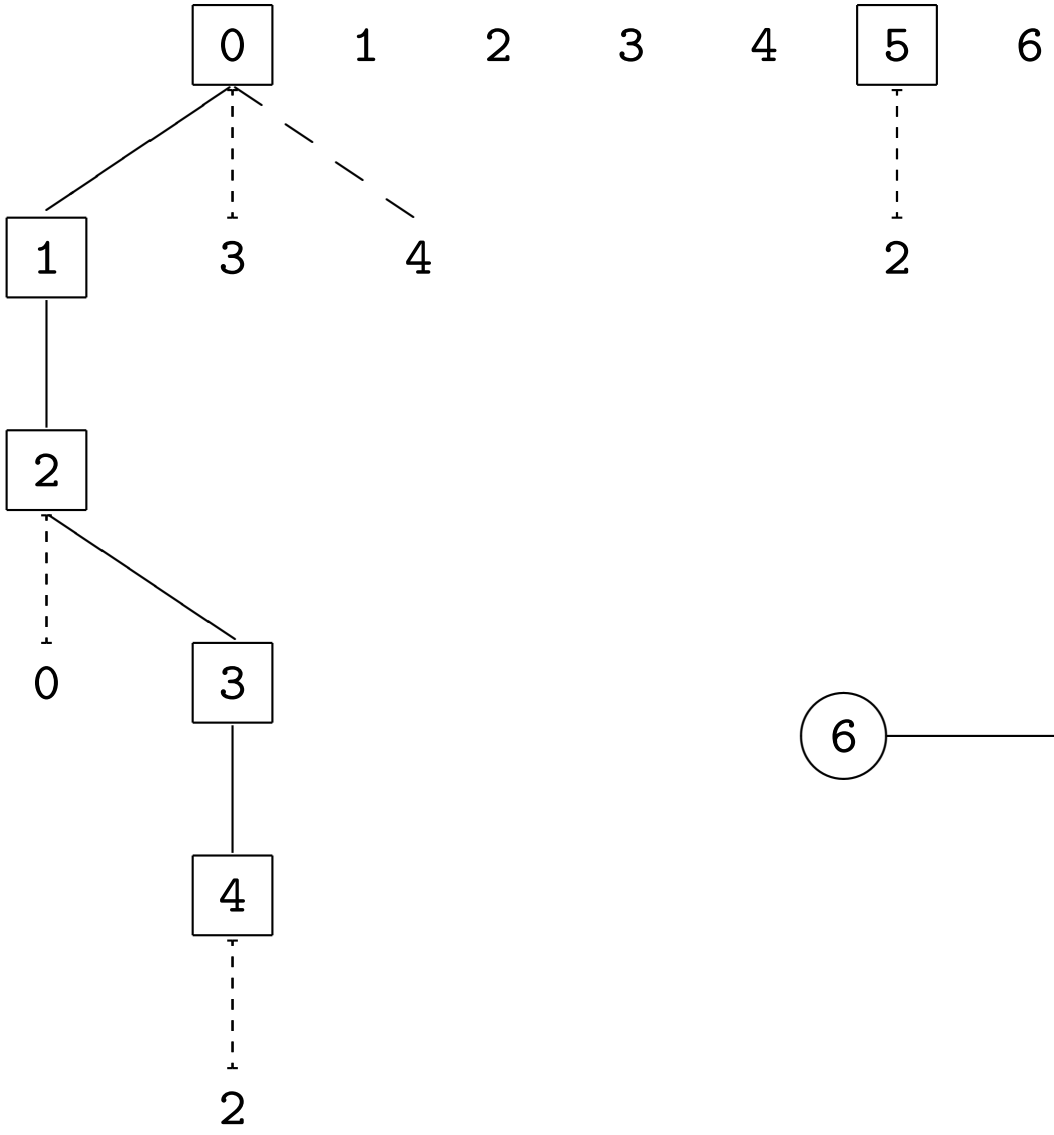


Ordem:

0 1 2 3 4 5

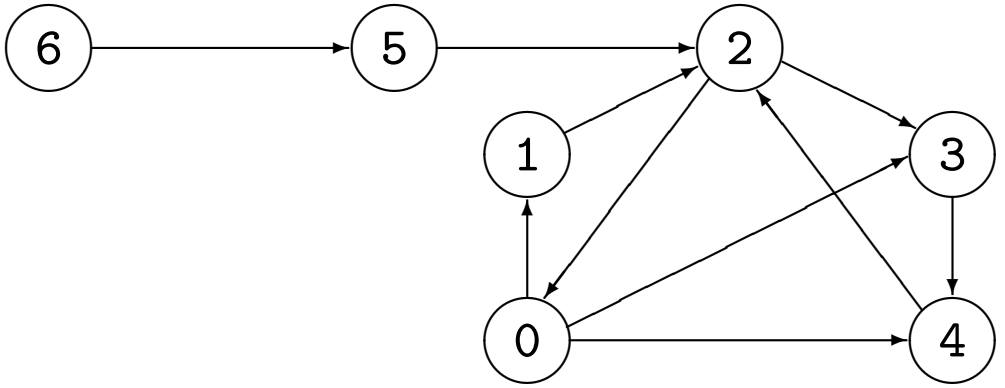


Percurso em Profundidade

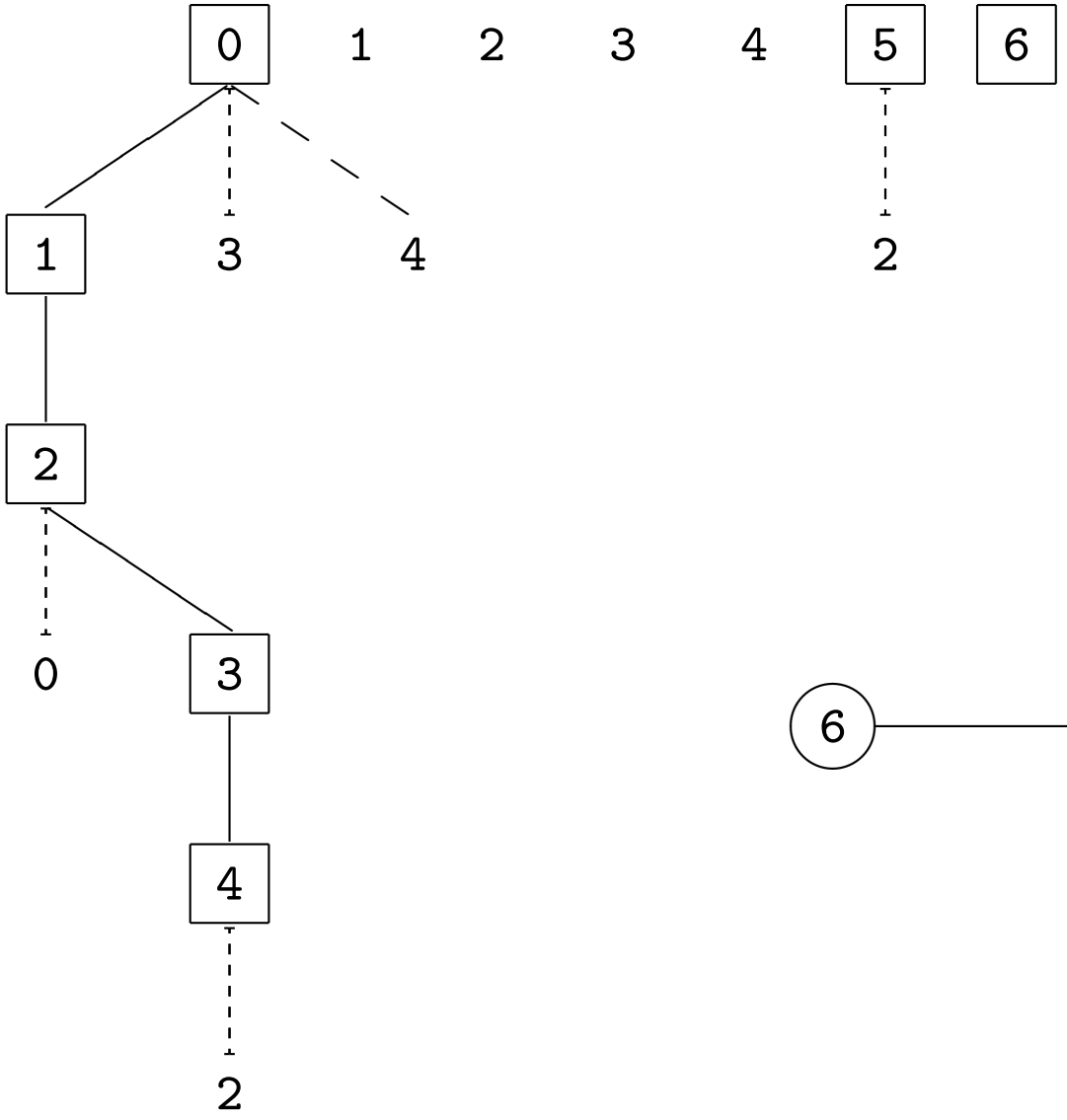


Ordem:

0 1 2 3 4 5

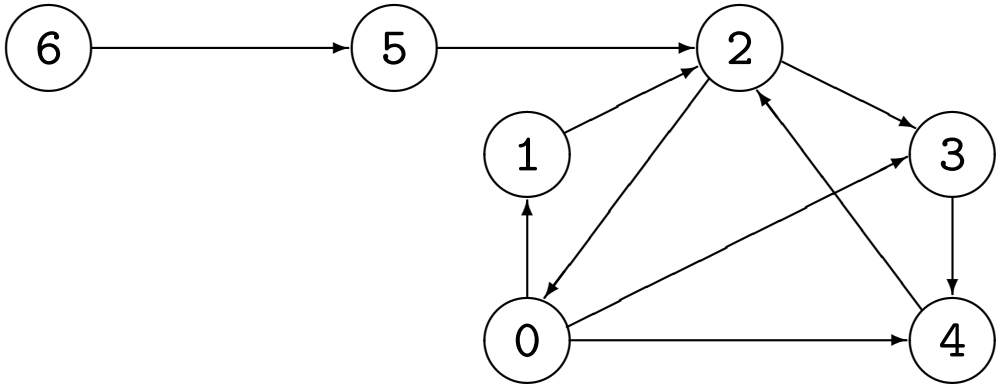


Percurso em Profundidade

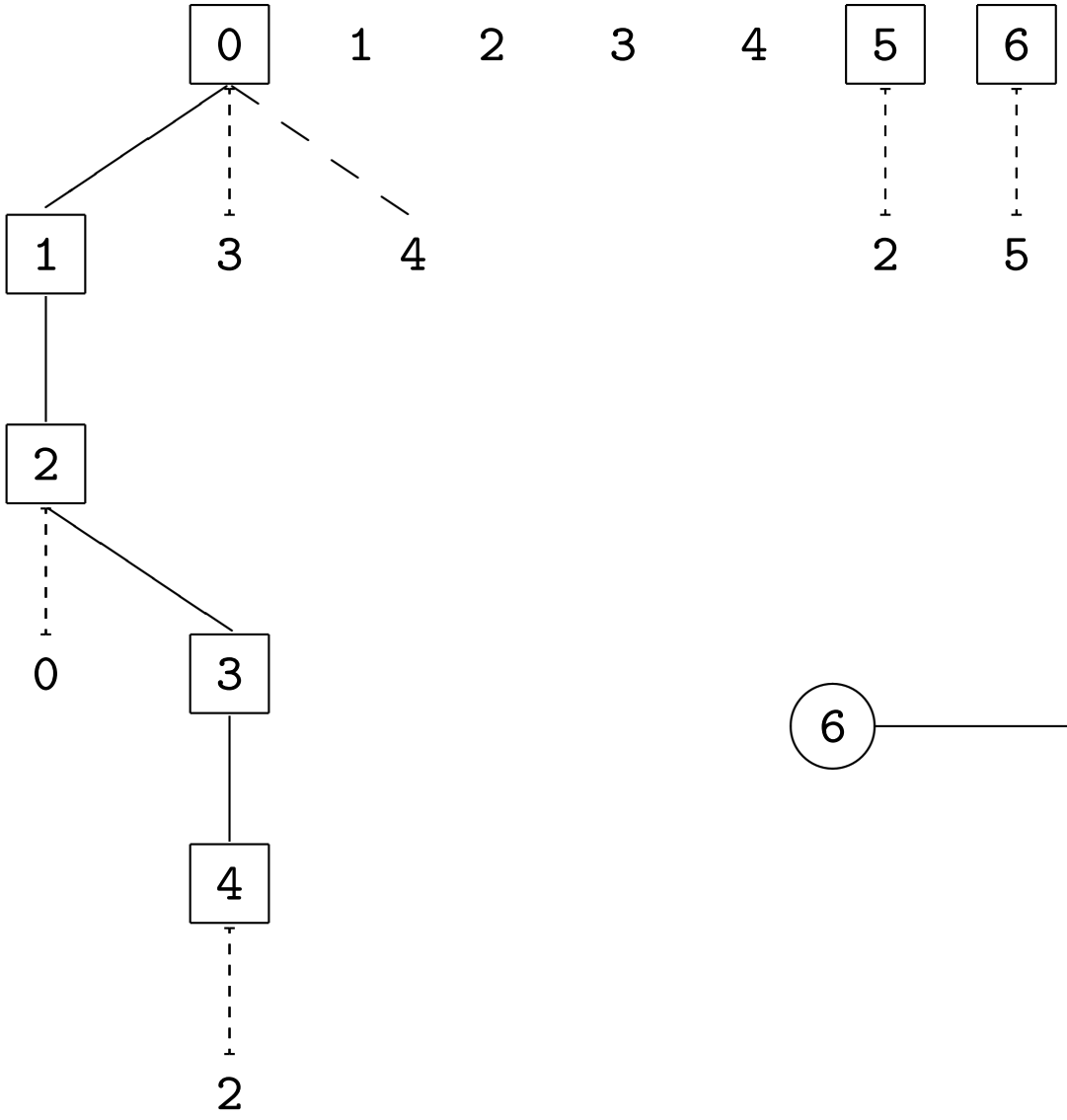


Ordem:

0 1 2 3 4 5 6

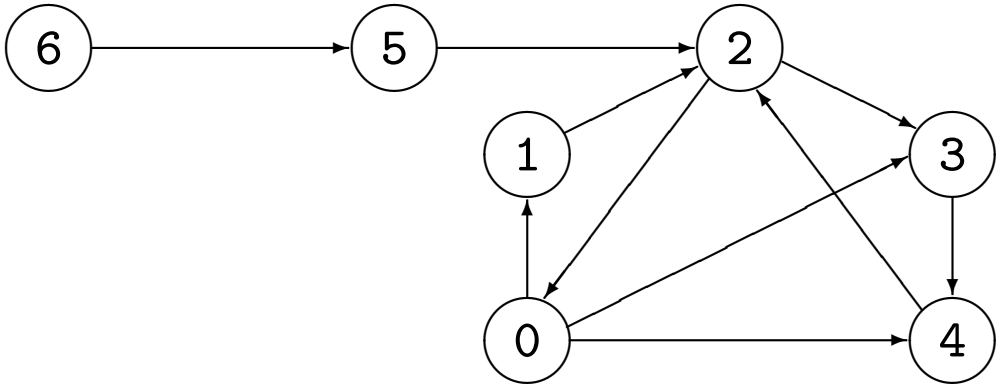


Percurso em Profundidade



Ordem:

0 1 2 3 4 5 6



Percurso em Profundidade

(Depth-First Search Traversal)

```
void dfsTraversal( Digraph graph ) {

    boolean[] processed = new boolean[ graph.numNodes() ];

    for every Node v in graph.nodes()
        processed[v] = false;

    for every Node v in graph.nodes()
        if ( !processed[v] )
            dfsExplore(graph, processed, v);
}
```

Árvore em Profundidade (recursivo)

```
void dfsExplore( Digraph graph,  boolean[] processed,  Node root ) {

    // PROCESS(root)
    processed[root] = true;

    for every Node v in graph.outAdjacentNodes(root)
        if ( !processed[v] )
            dfsExplore(graph, processed, v);
}
```

Complexidade de **dfsExplore** (recursivo)

- **Custo da uma chamada (vértice w)**
  - Se PROCESS(w) não for constante, considerar o seu custo
  - Altera-se o booleano $\Theta(1)$
  - Iteram-se os sucessores de w
    - Grafo em matriz de adjacências $\Theta(|V|)$
    - Grafo em vetor de listas de adjacências (suc.) $\Theta(|\text{Suc}(w)|)$
- **Custo de todas as chamadas (uma por cada vértice)**
  - Grafo em matriz de adjacências $\Theta(|V|^2)$
  - Grafo em vetor de listas de adjacências (suc.) $\Theta(|V| + |A|)$

    Num grafo orientado, $\sum_{w \in V} |\text{Suc}(w)| = |A|$.

    Num grafo não orientado, $\sum_{w \in V} |\text{Suc}(w)| = 2 \times |A|$.

Complexidades de **dfsTraversal** (recursivo)

Grafo em matriz de adjacências

Tempo

Criação do vetor processed	$\Theta(1)$
1º ciclo (inicialização do vetor processed)	$\Theta(\|V\|)$
2º ciclo (ignorando execuções de dfsExplore)	$\Theta(\|V\|)$
Execuções de dfsExplore	$\Theta(\|V\|^2)$
TOTAL	$\Theta(\|V\|^2)$

Espaço

Vetor processed	$\Theta(\|V\|)$
Pilha de chamadas recursivas	$O(\|V\|)$
TOTAL	$\Theta(\|V\|)$

Complexidades de **dfsTraversal** (recursivo)

Grafo em vetor de listas de adjacências (suc.)

Tempo

Criação do vetor processed	$\Theta(1)$
1º ciclo (inicialização do vetor processed)	$\Theta(\|V\|)$
2º ciclo (ignorando execuções de dfsExplore)	$\Theta(\|V\|)$
Execuções de dfsExplore	$\Theta(\|V\| + \|A\|)$
TOTAL	$\Theta(\|V\| + \|A\|)$

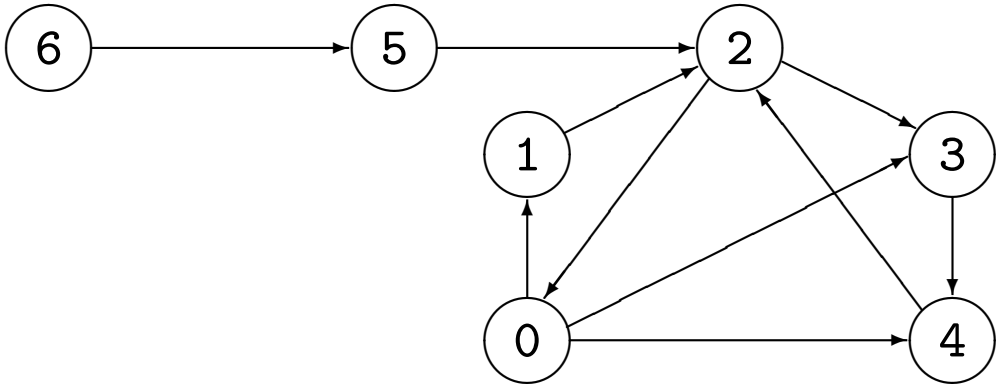
Espaço

Vetor processed	$\Theta(\|V\|)$
Pilha de chamadas recursivas	$O(\|V\|)$
TOTAL	$\Theta(\|V\|)$

Percurso em Profundidade

0

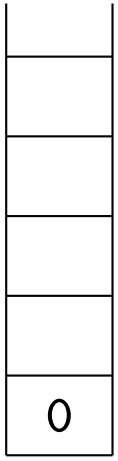
Ordem:

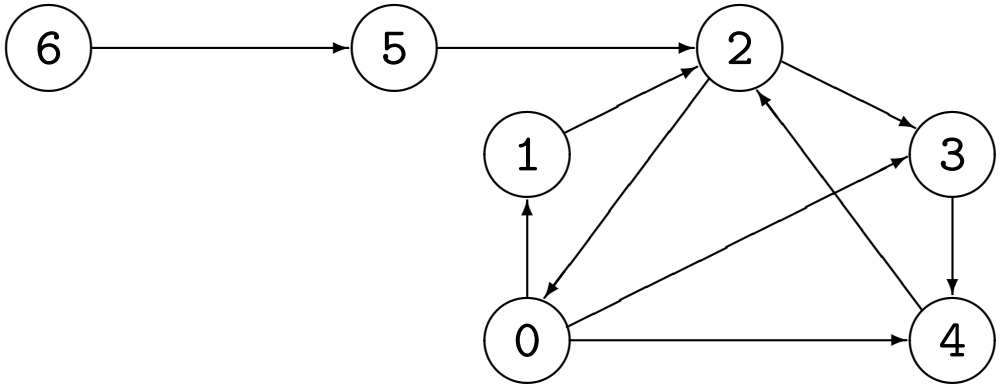


Percurso em Profundidade

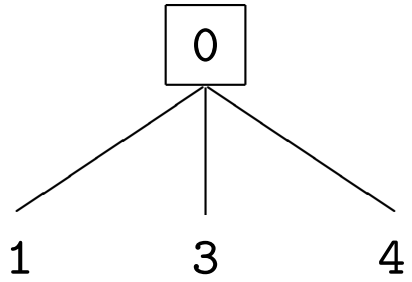
0

Ordem:



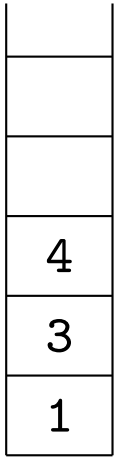


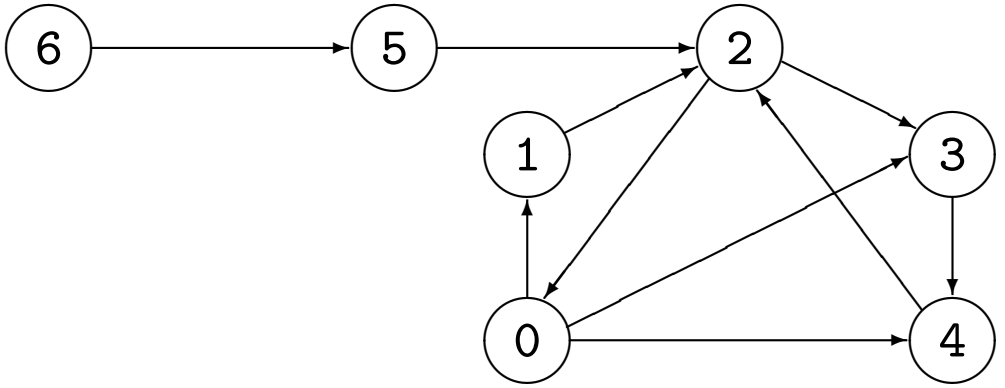
Percurso em Profundidade



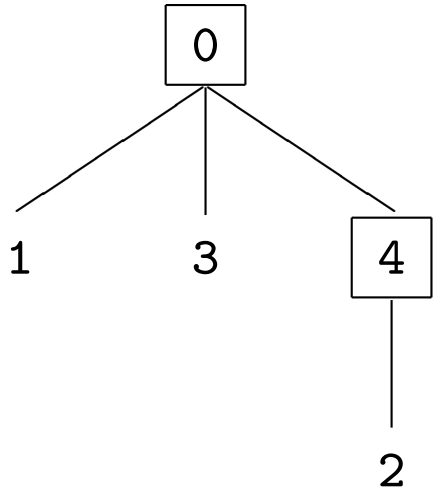
Ordem:

0



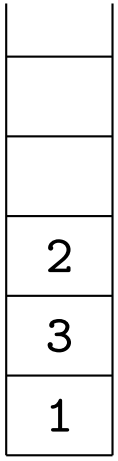


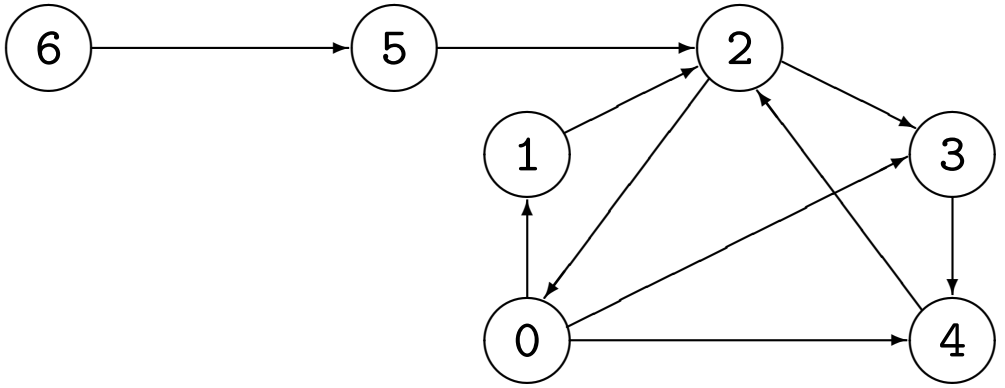
Percurso em Profundidade



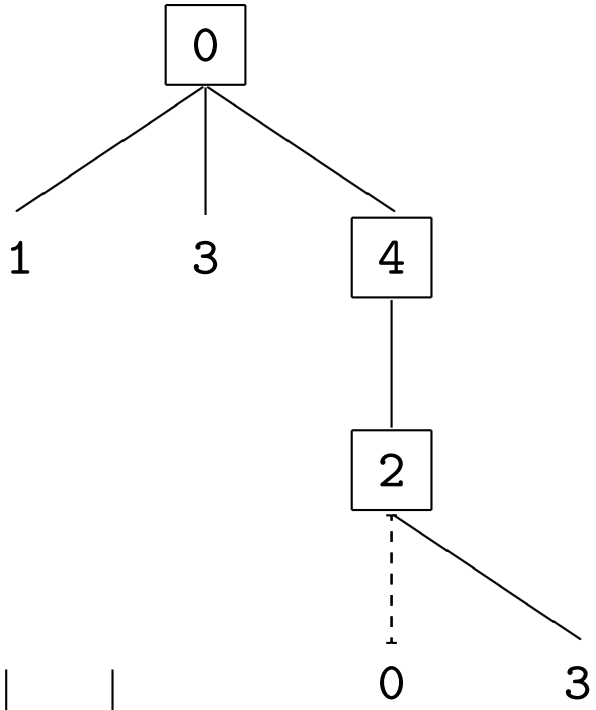
Ordem:

0 4



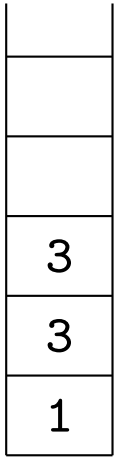


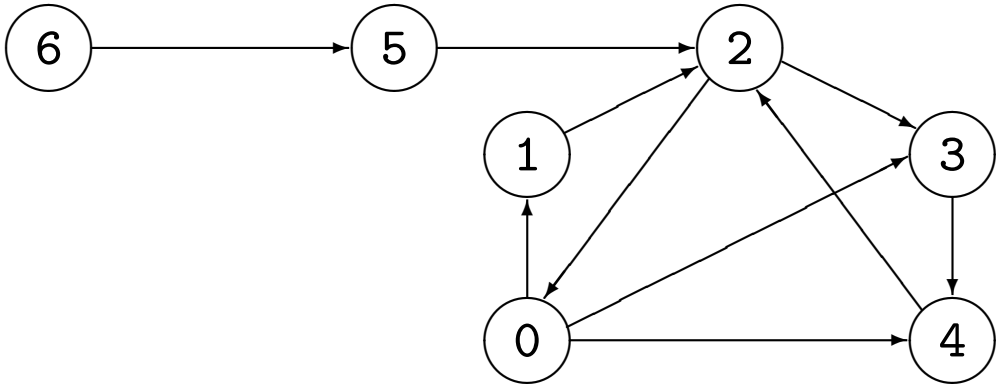
Percurso em Profundidade



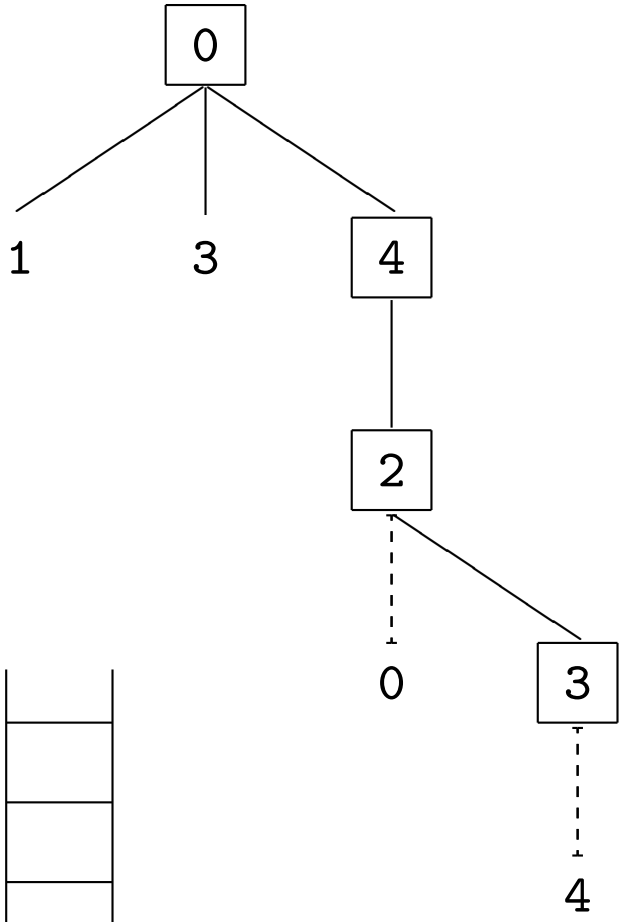
Ordem:

0 4 2



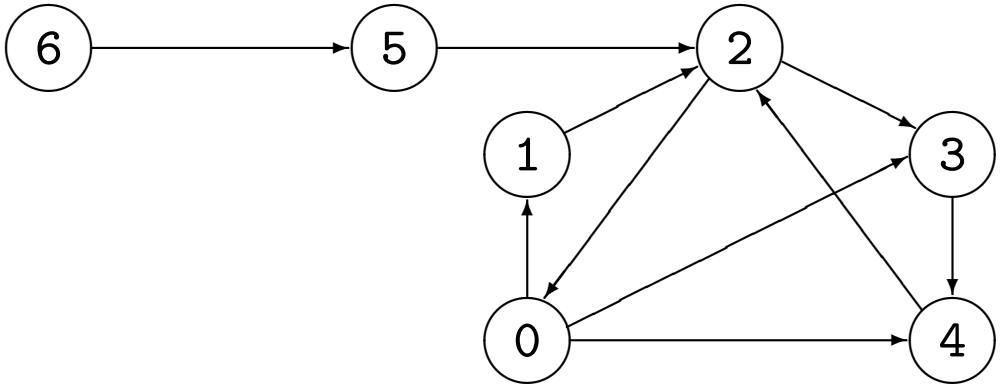


Percurso em Profundidade

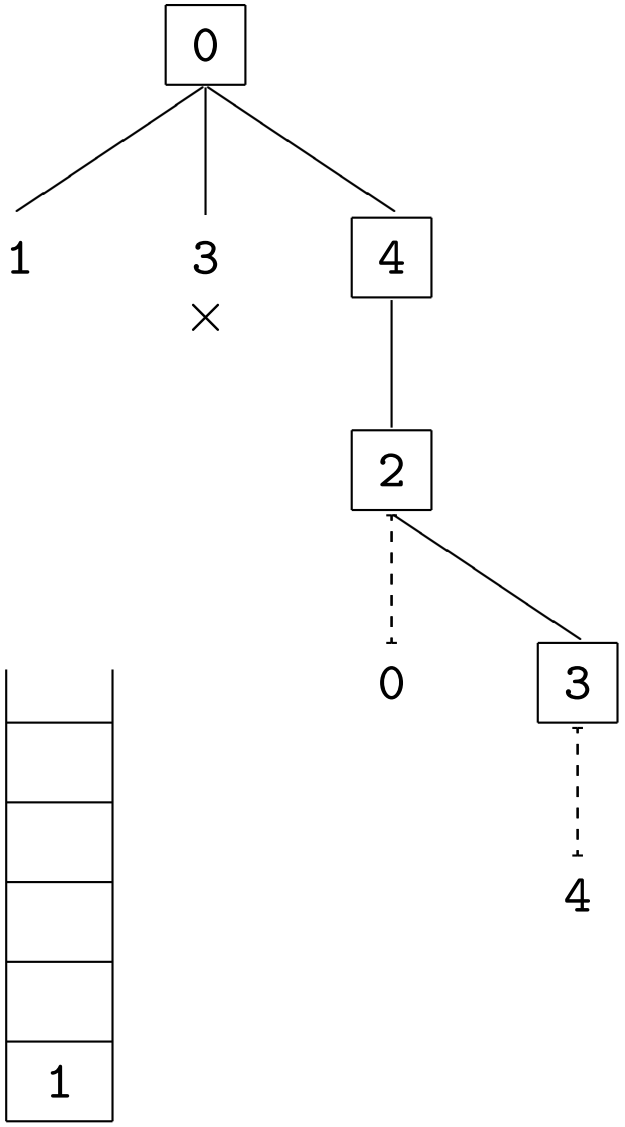


Ordem:

0 4 2 3

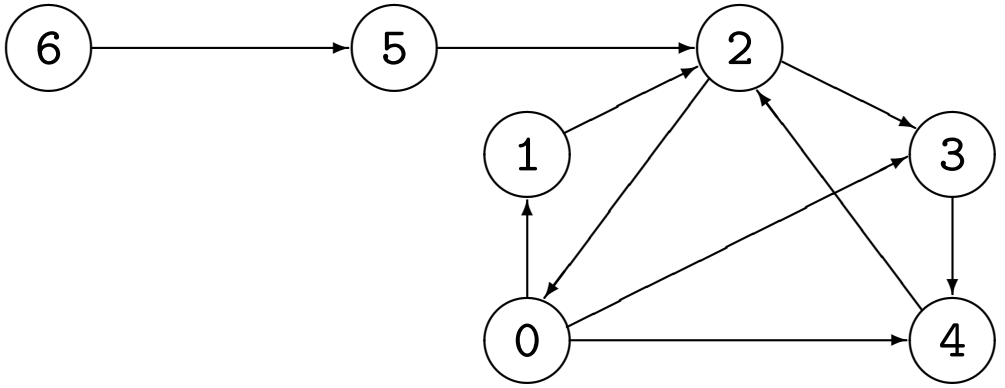


Percurso em Profundidade

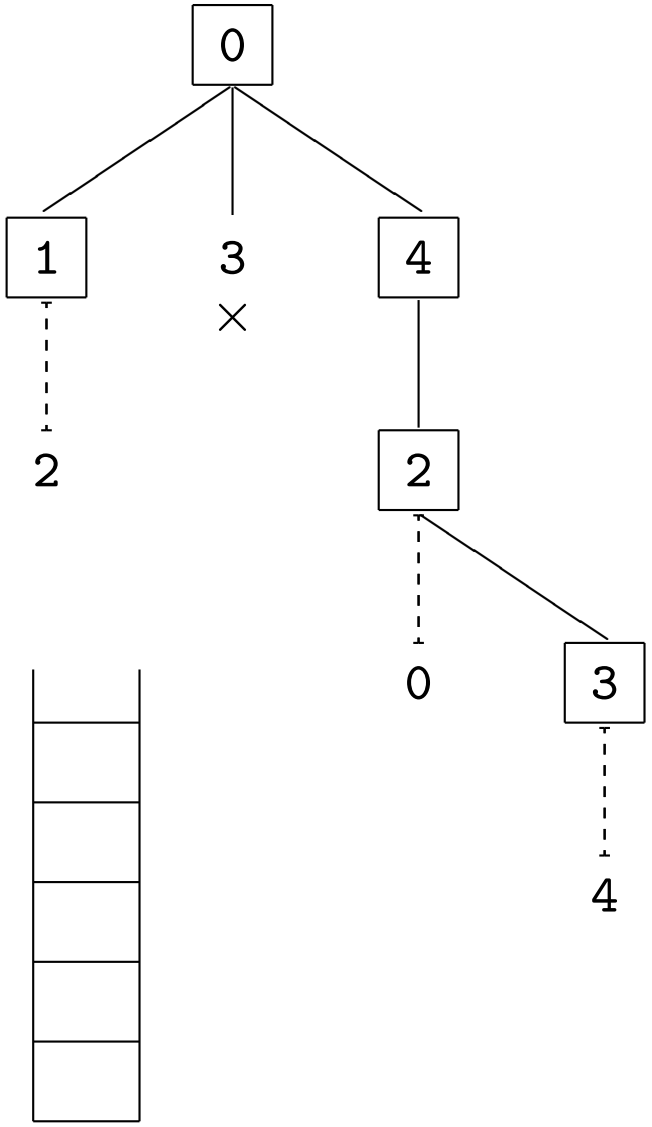


Ordem:

0 4 2 3

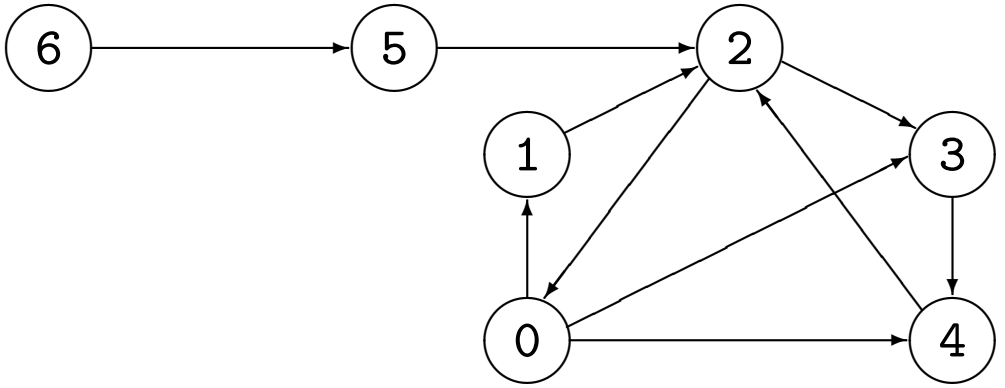


Percurso em Profundidade

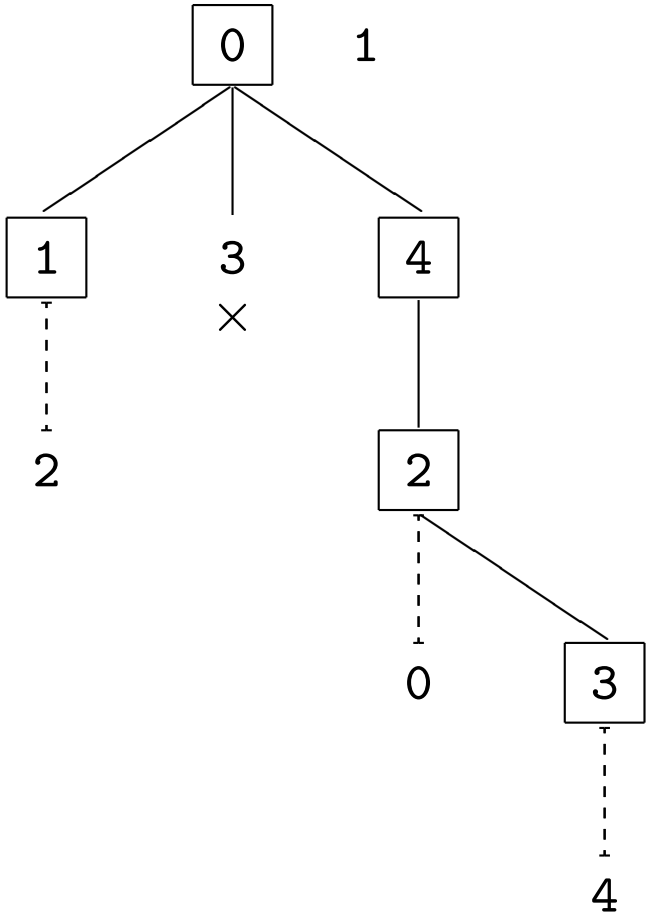


Ordem:

0 4 2 3 1

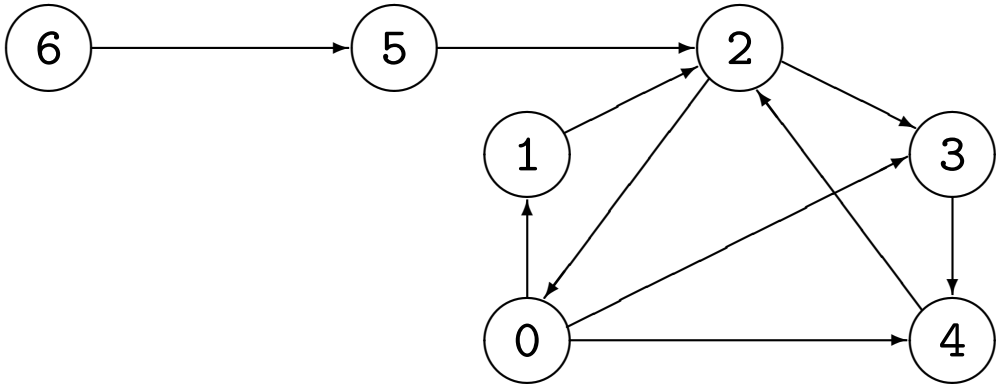


Percurso em Profundidade

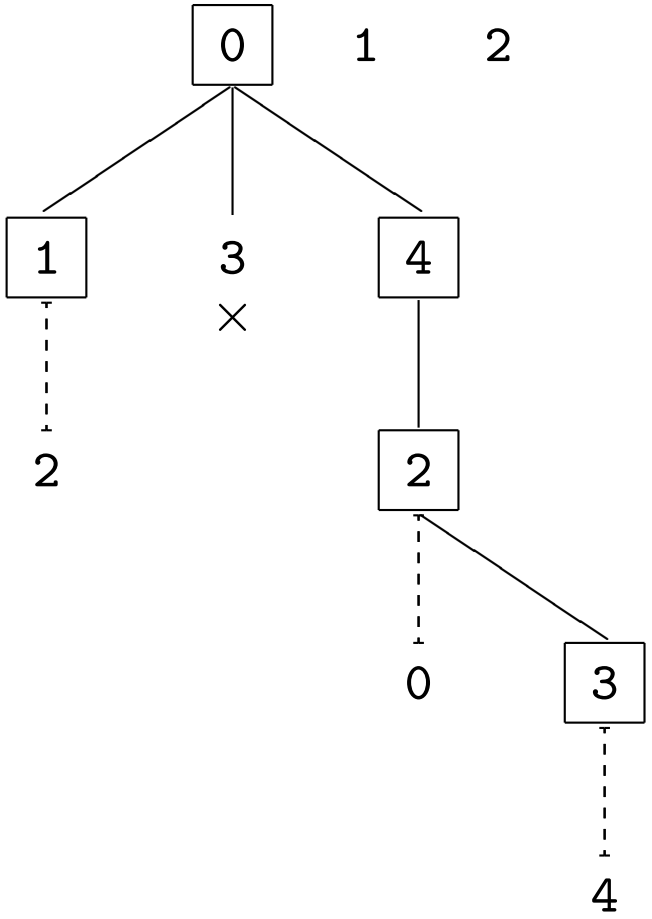


Ordem:

0 4 2 3 1

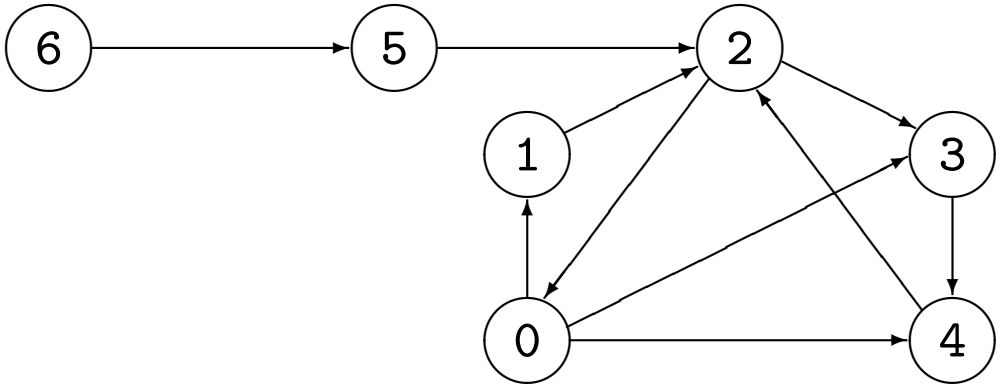


Percurso em Profundidade

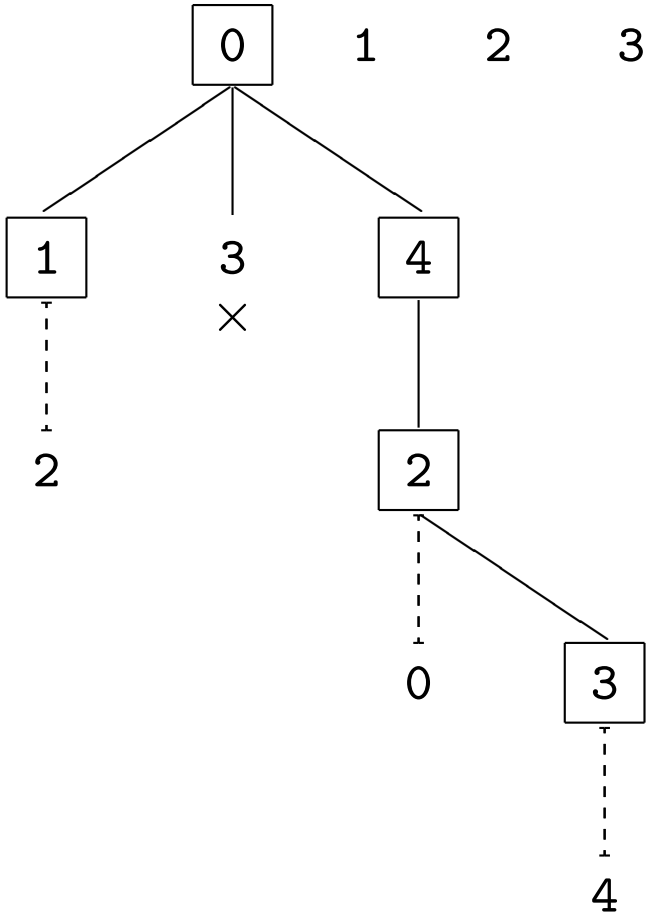


Ordem:

0 4 2 3 1

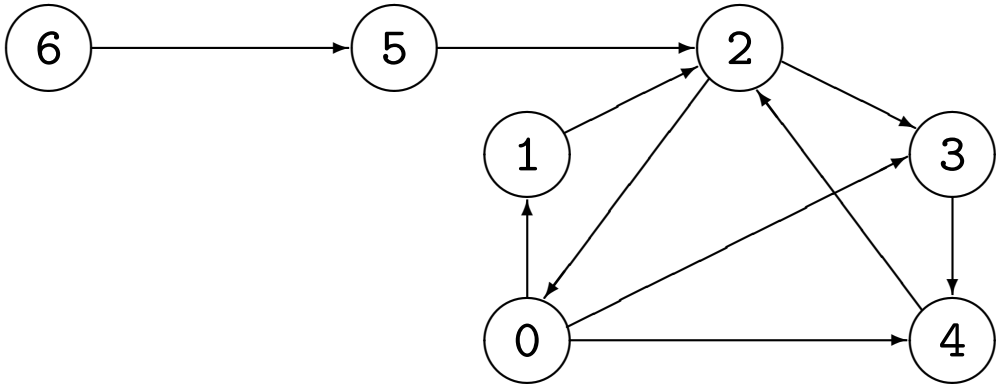


Percurso em Profundidade

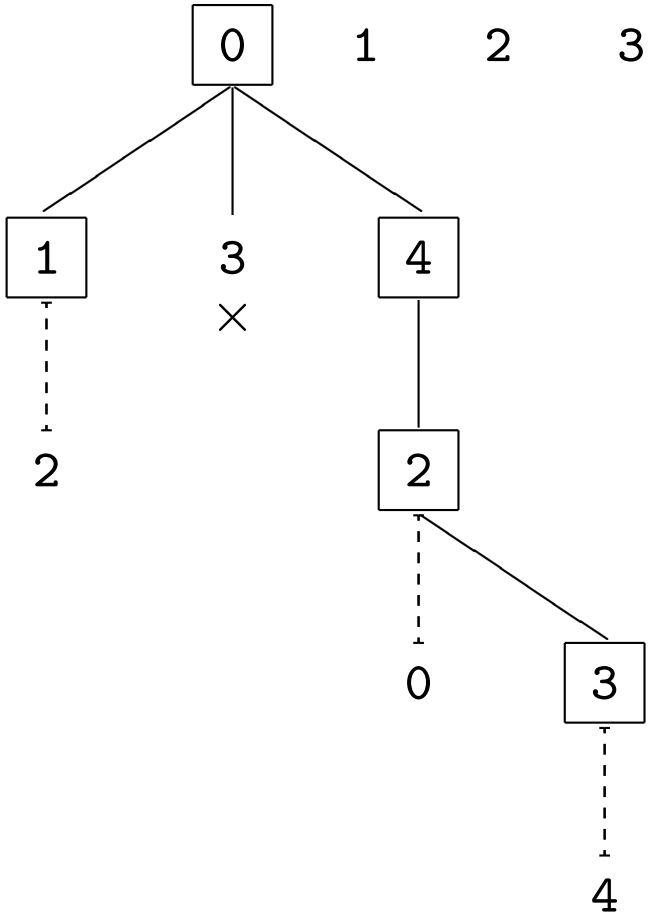


Ordem:

0 4 2 3 1

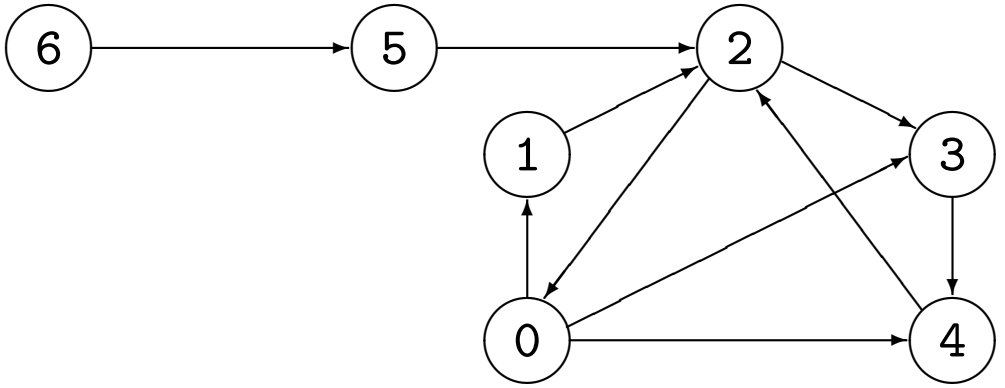


Percurso em Profundidade

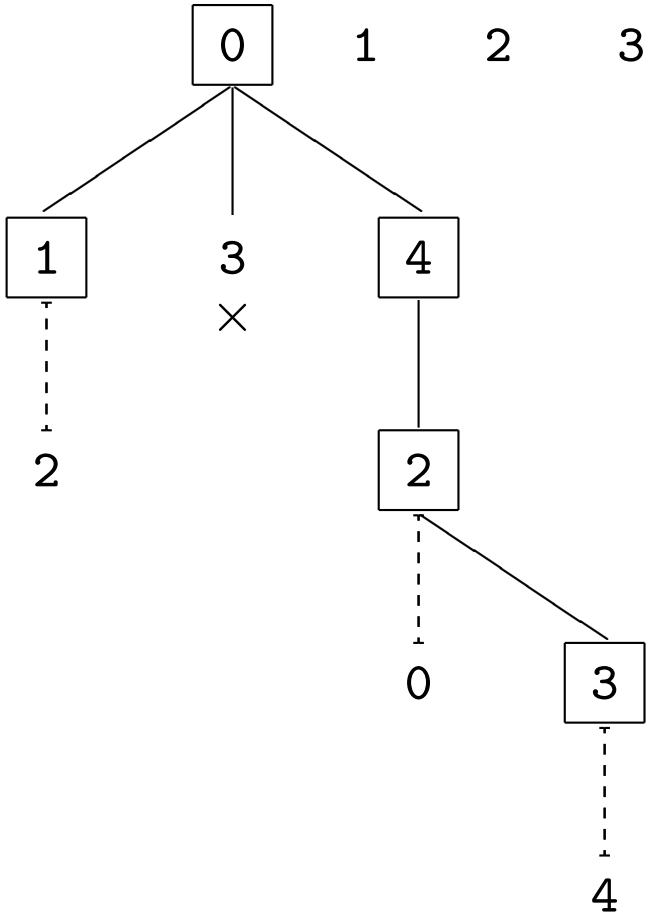


Ordem:

0 4 2 3 1

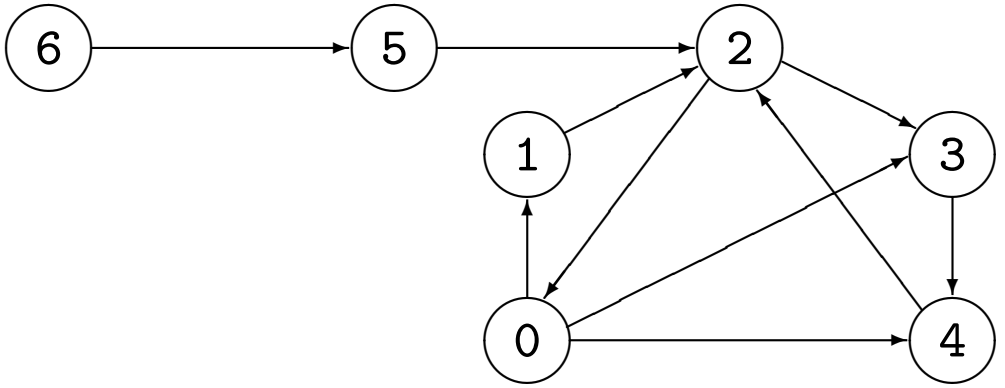


Percurso em Profundidade

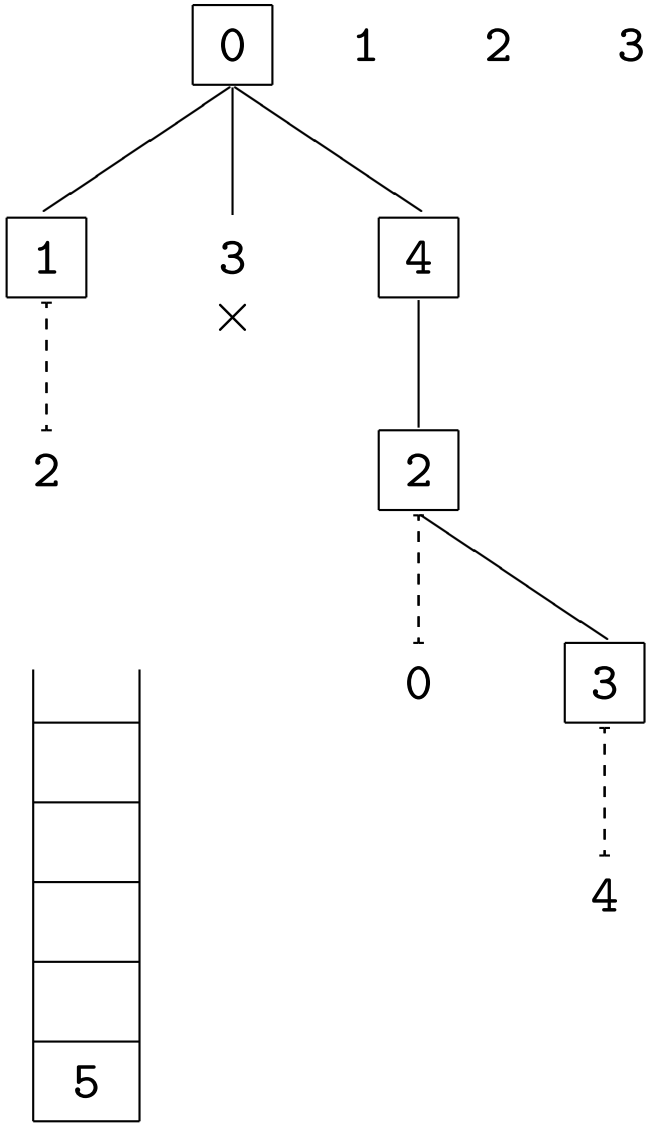


Ordem:

0 4 2 3 1

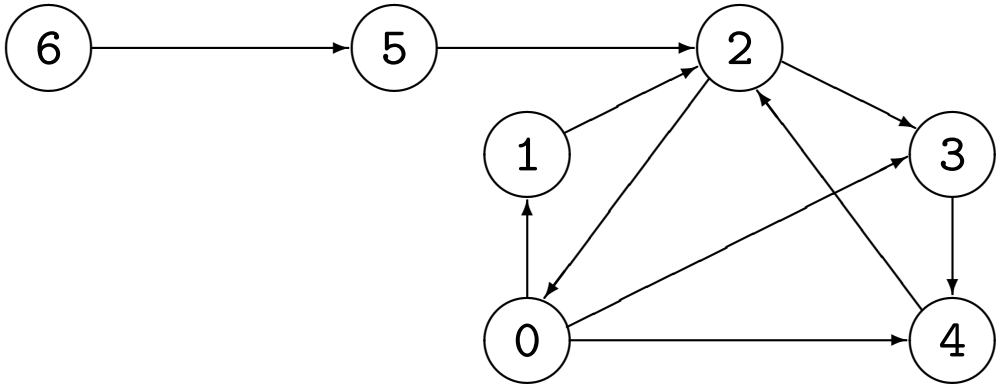


Percurso em Profundidade

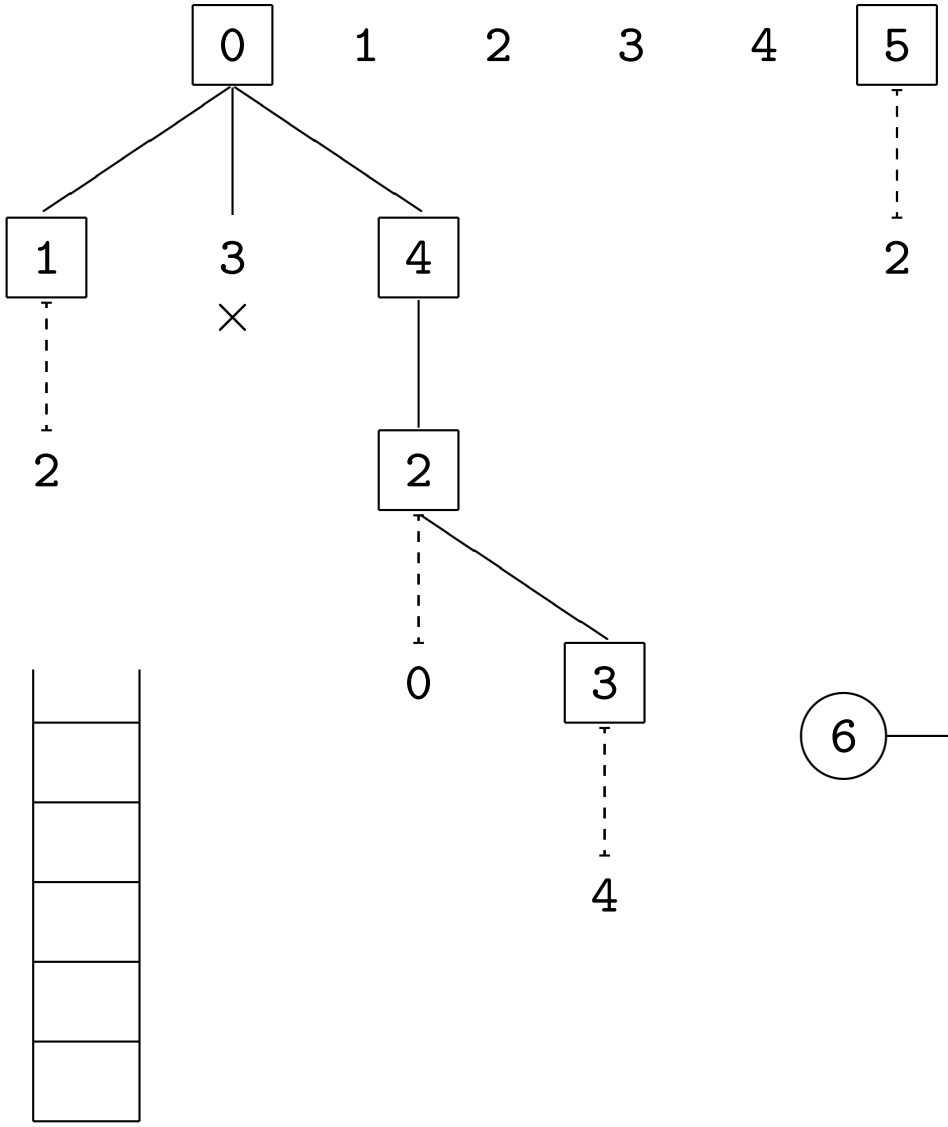


Ordem:

0 4 2 3 1

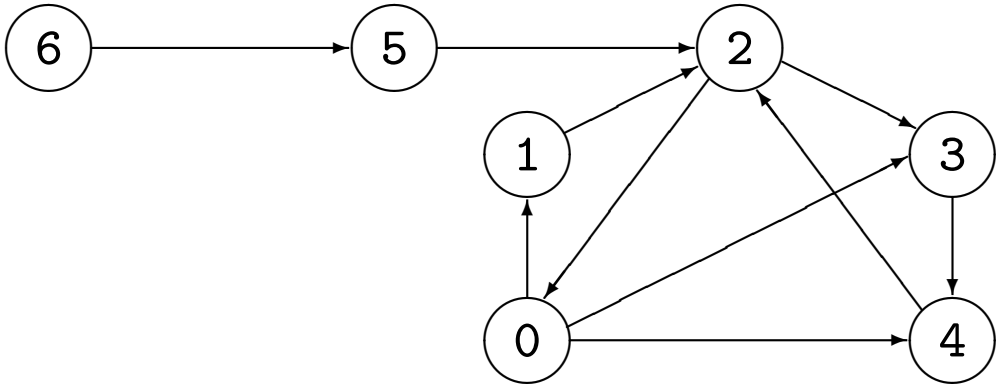


Percurso em Profundidade

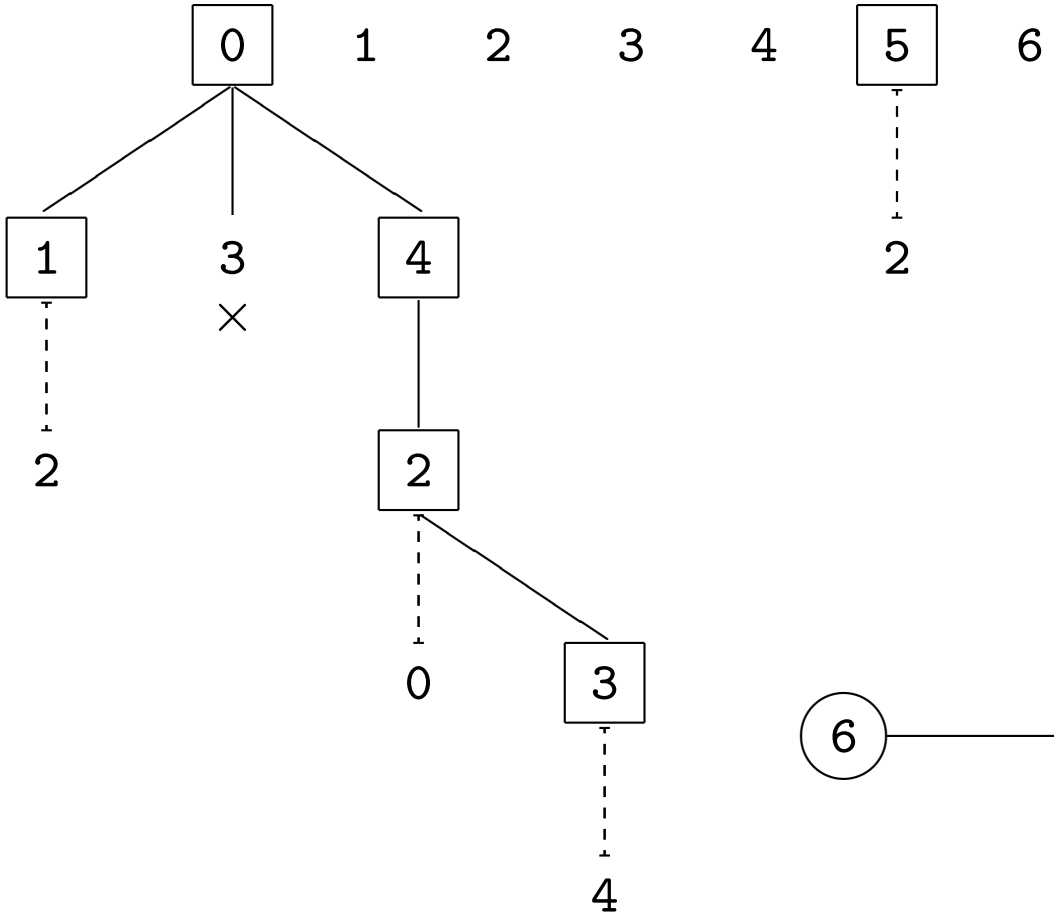


Ordem:

0 4 2 3 1 5

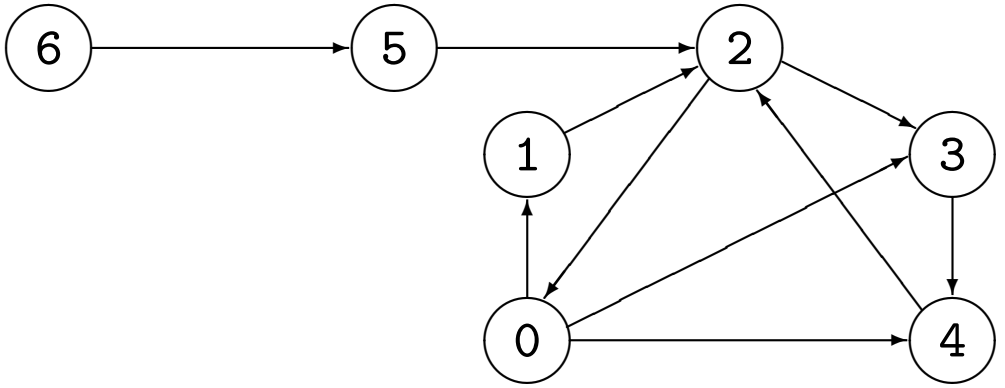


Percurso em Profundidade

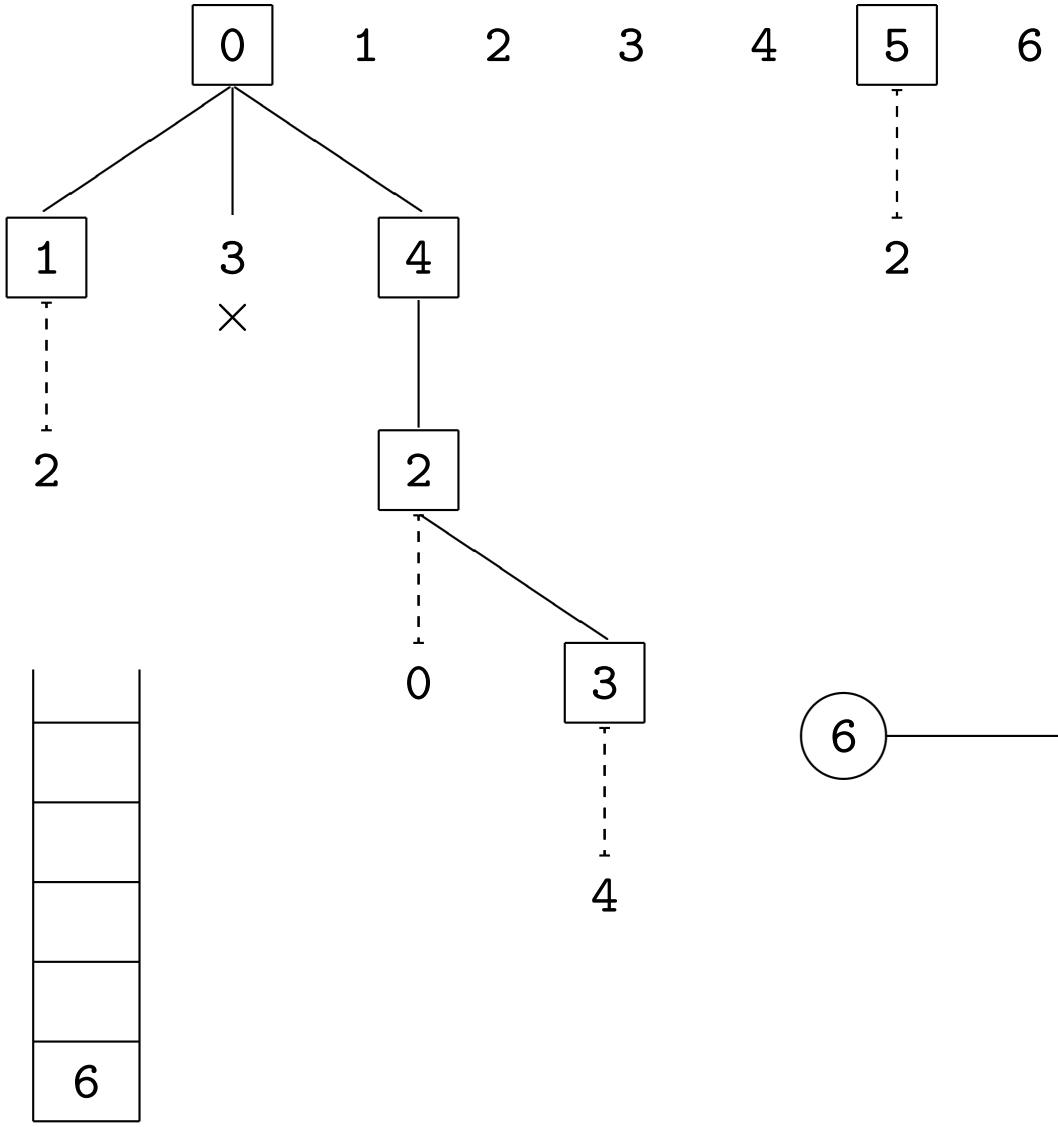


Ordem:

0 4 2 3 1 5

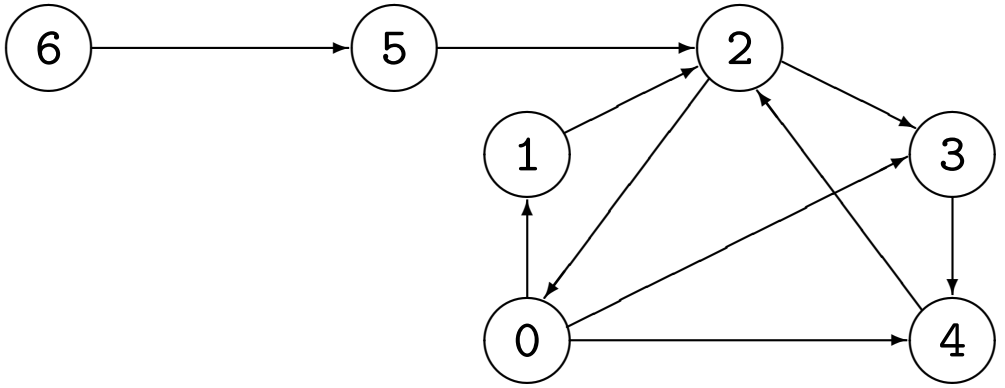


Percurso em Profundidade

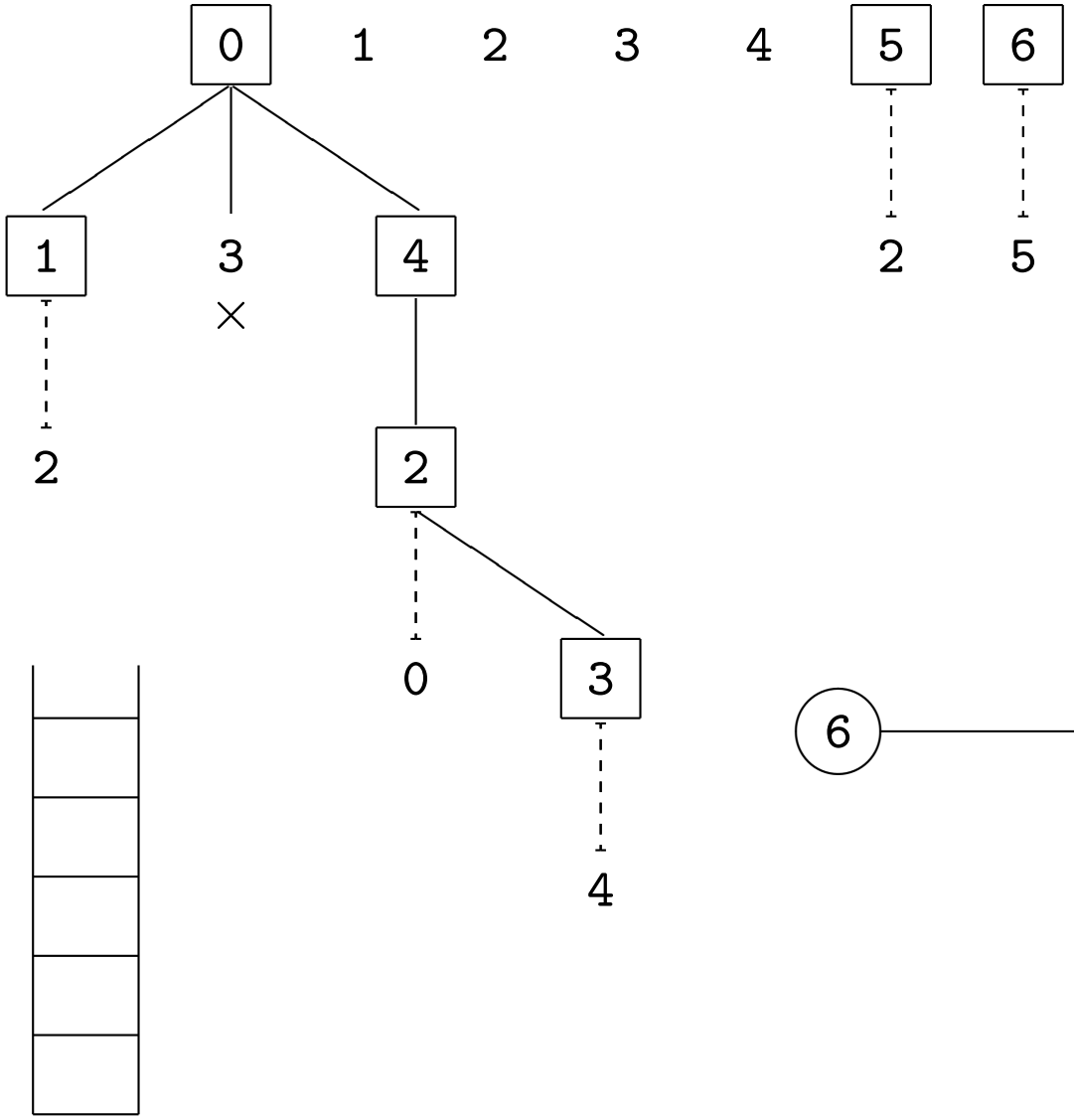


Ordem:

0 4 2 3 1 5

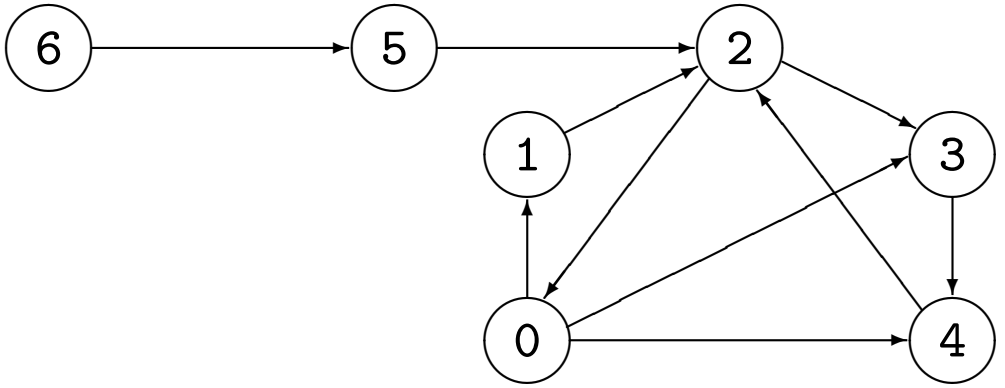


Percurso em Profundidade

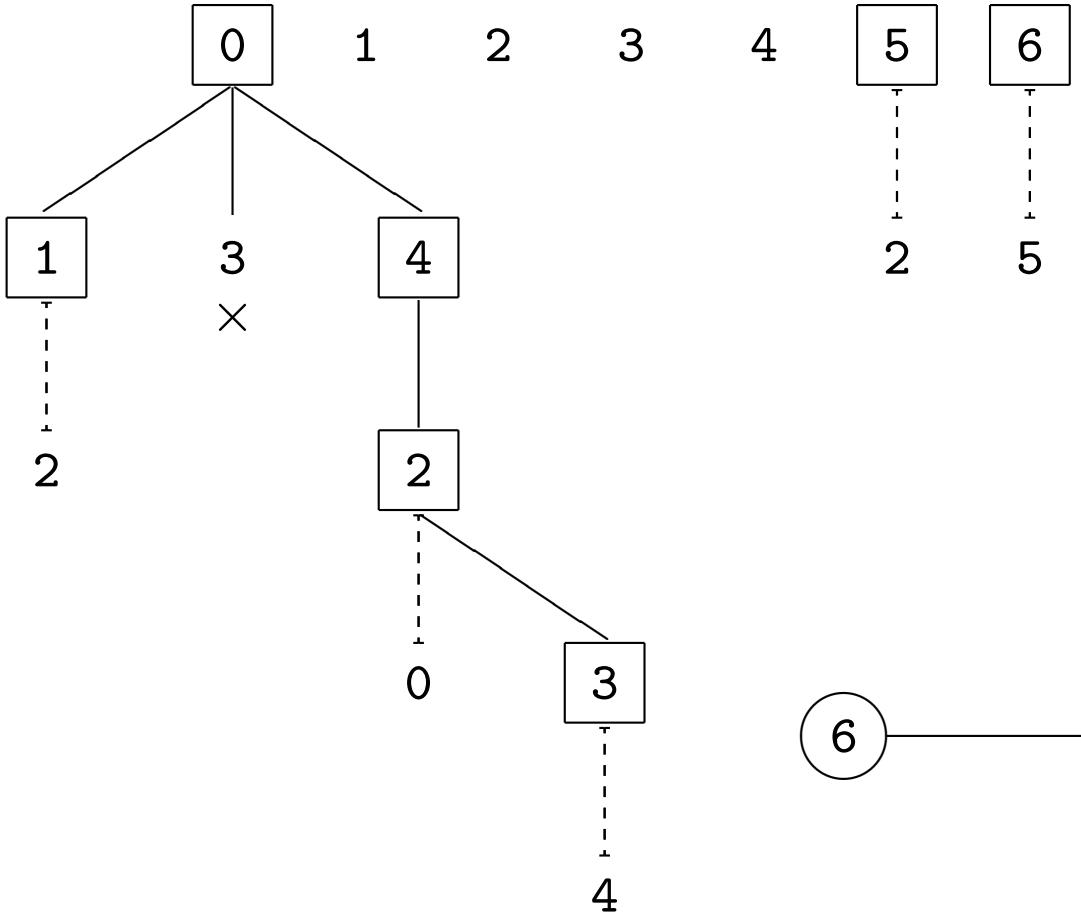


Ordem:

0 4 2 3 1 5 6

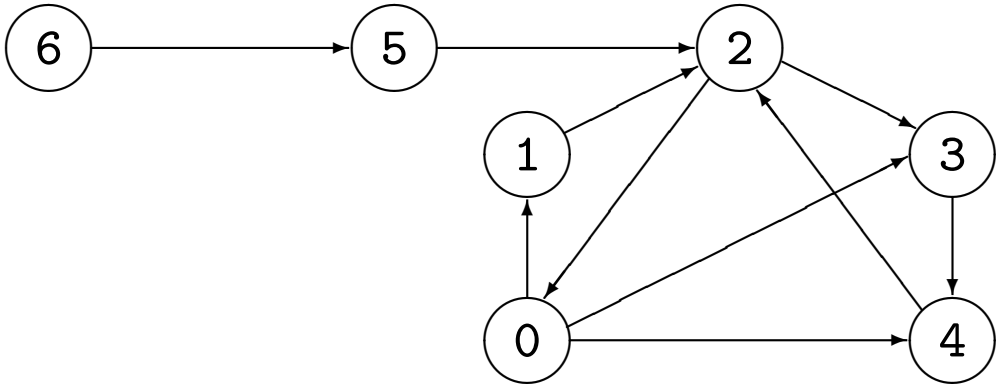


Percurso em Profundidade



Ordem:

0 4 2 3 1 5 6



Percurso em Profundidade

(Depth-First Search Traversal)

```
void dfsTraversal( Digraph graph ) {

    boolean[] processed = new boolean[ graph.numNodes() ];

    for every Node v in graph.nodes()
        processed[v] = false;

    for every Node v in graph.nodes()
        if ( !processed[v] )
            dfsExplore(graph, processed, v);
}
```

Árvore em Profundidade (iterativo)

```
void dfsExplore( Digraph graph,  boolean[] processed,  Node root ) {
    Stack<Node> foundUnprocessed = new StackIn...<>(?);
    foundUnprocessed.push(root);
    do {
        Node node = foundUnprocessed.pop();
        if ( !processed[node] ) {
            // PROCESS(node)
            processed[node] = true;
            for every Node v in graph.outAdjacentNodes(node)
                if ( !processed[v] )
                    foundUnprocessed.push(v);
        }
    }
    while ( !foundUnprocessed.isEmpty() );
}
```

Complexidade de **dfsExplore** (iterativo)

- **Por cada vértice (w)**
  - Se PROCESS(w) não for constante, considerar o seu custo
  - Altera-se o booleano $\Theta(1)$
  - Iteram-se os sucessores de w
    - Grafo em matriz de adjacências $\Theta(|V|)$
    - Grafo em vetor de listas de adjacências (suc.) $\Theta(|\text{Suc}(w)|)$
  - Empilham-se os sucessores não processados de w $O(|\text{Suc}(w)|)$
- **Custo de todas as execuções** (#POP = #PUSH)
  - Grafo em matriz de adjacências $\Theta(|V|^2)$
  - Grafo em vetor de listas de adjacências (suc.) $\Theta(|V| + |A|)$

    Num grafo orientado, $\sum_{w \in V} |\text{Suc}(w)| = |A|$.

    Num grafo não orientado, $\sum_{w \in V} |\text{Suc}(w)| = 2 \times |A|$.

Complexidades de **dfsTraversal** (iterativo)

Grafo em matriz de adjacências

(se push, pop e isEmpty de Stack forem $\Theta(1)$)

Tempo

Criação do vetor processed	$\Theta(1)$
1º ciclo (inicialização do vetor processed)	$\Theta(\|V\|)$
2º ciclo (ignorando execuções de dfsExplore)	$\Theta(\|V\|)$
Execuções de dfsExplore	$\Theta(\|V\|^2)$
TOTAL	$\Theta(\|V\|^2)$

Espaço

Vetor processed	$\Theta(\|V\|)$
Pilha foundUnprocessed	$O(\|A\|)$
TOTAL	$O(\|V\| + \|A\|)$

Complexidades de **dfsTraversal** (iterativo)

Grafo em vetor de listas de adjacências (suc.)

(se push, pop e isEmpty de Stack forem $\Theta(1)$)

Tempo

Criação do vetor processed	$\Theta(1)$
1^o ciclo (inicialização do vetor processed)	$\Theta(\|V\|)$
2^o ciclo (ignorando execuções de dfsExplore)	$\Theta(\|V\|)$
Execuções de dfsExplore	$\Theta(\|V\| + \|A\|)$
TOTAL	$\Theta(\|V\| + \|A\|)$

Espaço

Vetor processed	$\Theta(\|V\|)$
Pilha foundUnprocessed	$O(\|A\|)$
TOTAL	$O(\|V\| + \|A\|)$