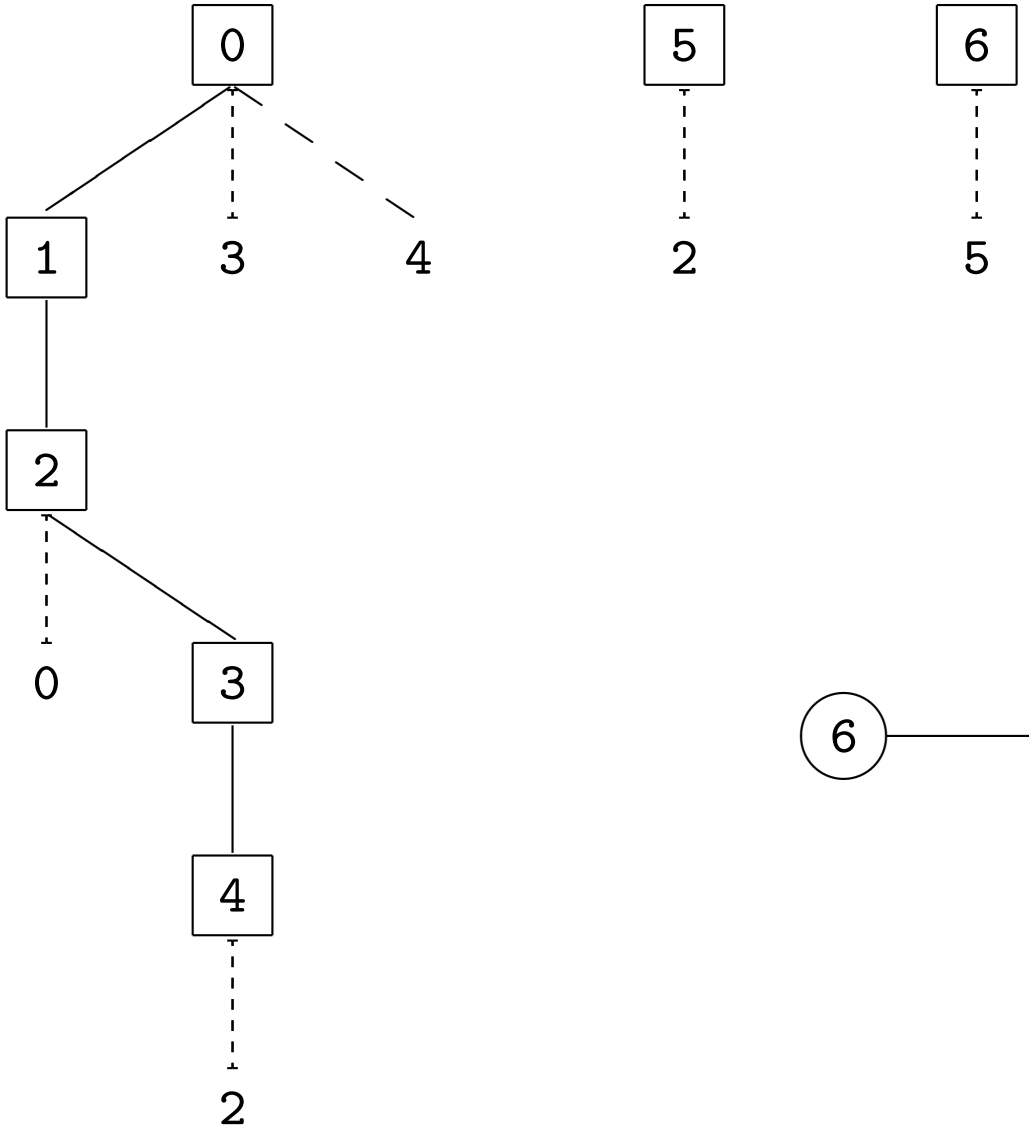
# Capítulo III

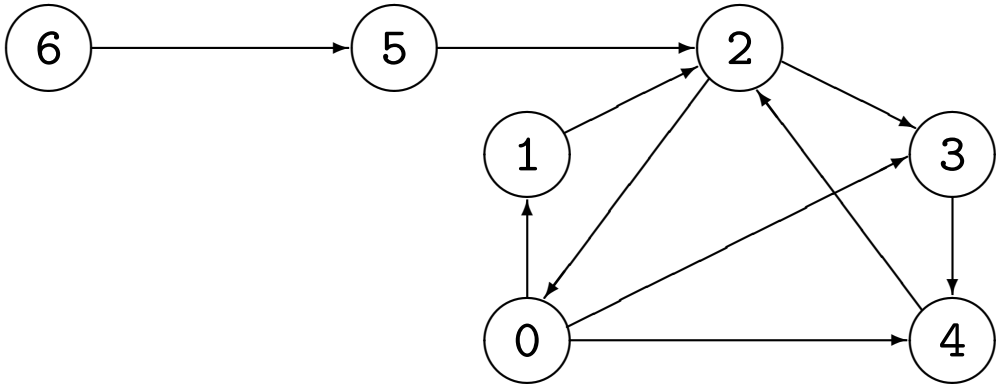
# Percursos em Profundidade e em Largura (num grafo orientado ou não orientado)

# Percurso em Profundidade




**Ordem:**

0 1 2 3 4 5 6



# Percurso em Profundidade

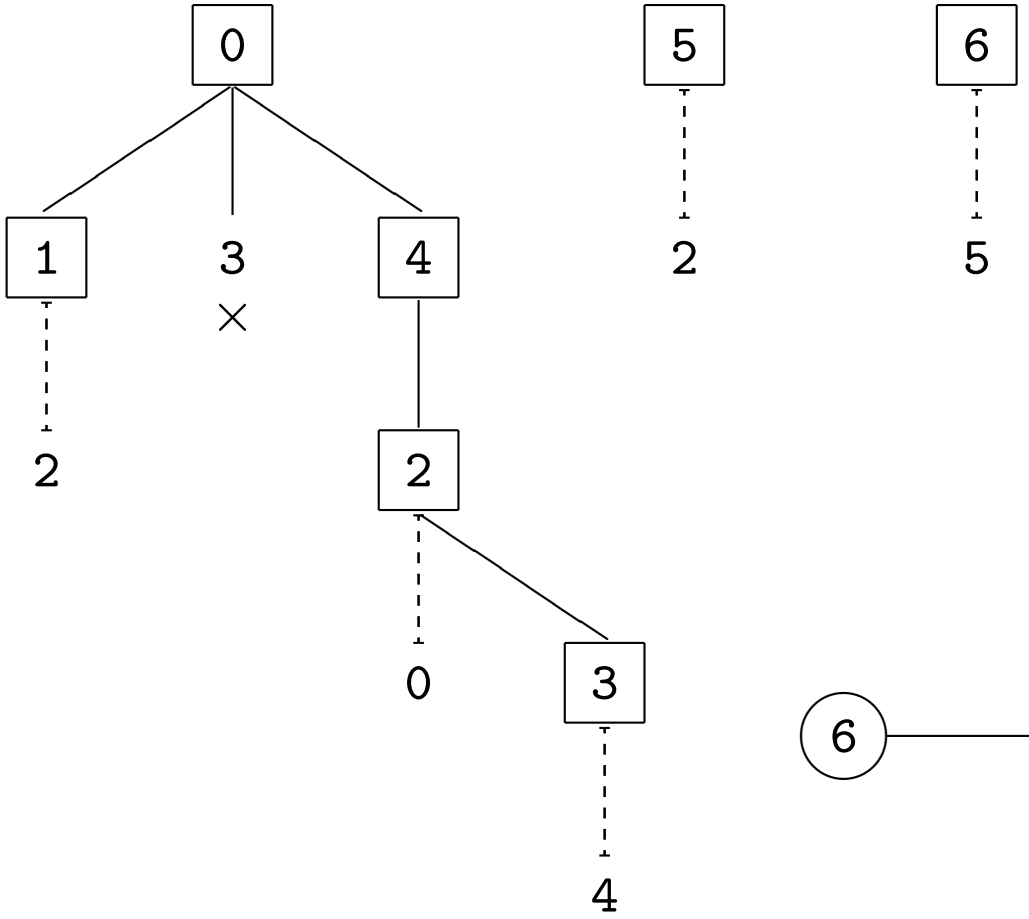
## *(Depth-First Search Traversal)*

```
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)

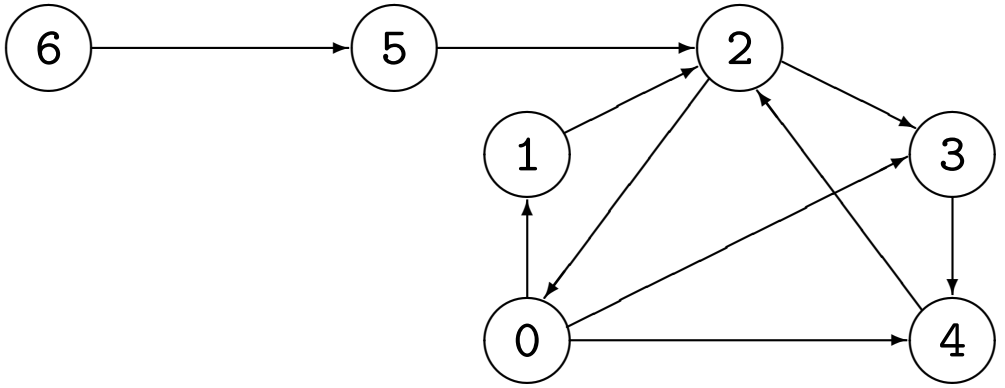
```
void dfsExplore( Digraph graph,  boolean[] processed,  Node root )
{
    // PROCESS(root)
    processed[root] = true;
    for every Node v in graph.outAdjacentNodes(root)
        if ( !processed[v] )
            dfsExplore(graph, processed, v);
}
```

# Percurso em Profundidade




**Ordem:**

0 4 2 3 1 5 6



# Árvore em Profundidade (iterativo)

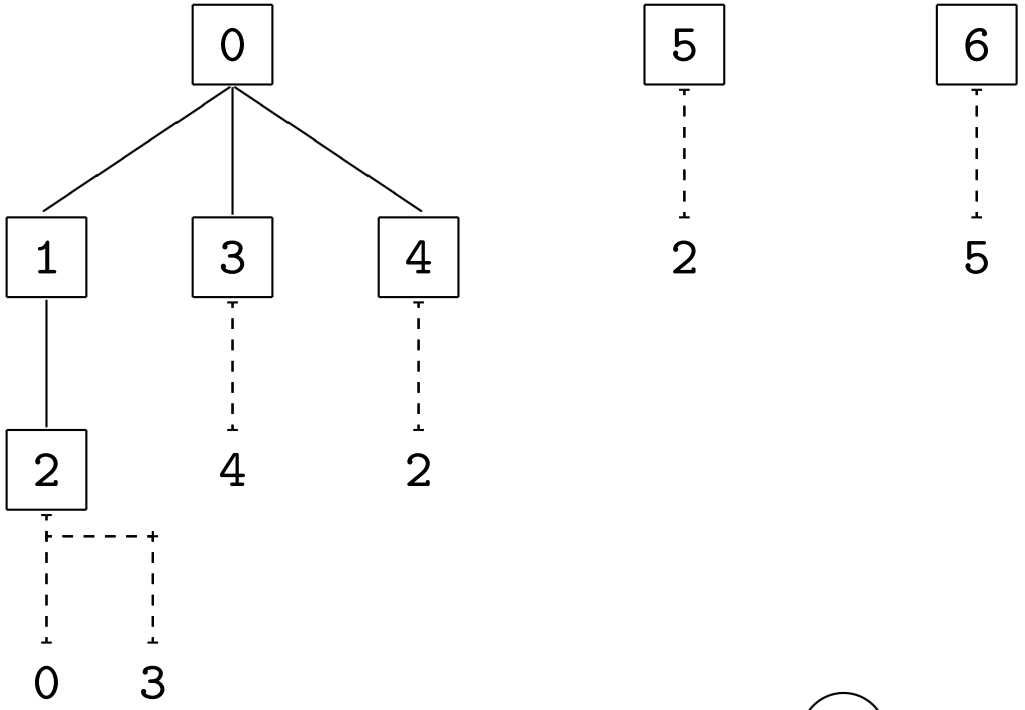
```
void dfsExplore( Digraph graph,  boolean[] processed,  Node root )
{
    Stack<Node> foundUnprocessed = new StackIn...<Node>(?);
    foundUnprocessed.push(root);
    do {
            Node node = foundUnprocessed.pop();
            if ( !processed[node] )
            {   // PROCESS(node)
        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

# Complexidades

| **Implementação do Grafo** $(V, A)$ | **Percurso em Profundidade** (recursivo ou iterativo) (grafo orientado ou não orientado) |
|---|---|
| Matriz | $\Theta(\|V\|^2)$ |
| Vetor de Listas | $\Theta(\|V\| + \|A\|)$ |

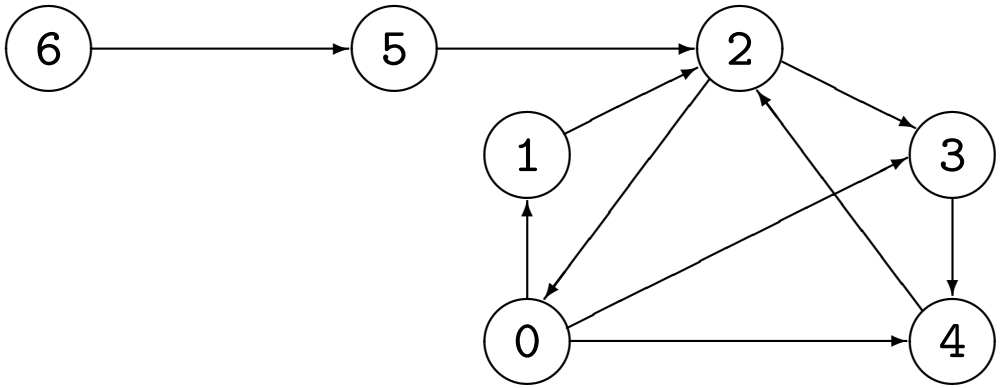
**foundUnprocessed**.size() $\leq$ $|A|$

# Percurso em Largura




**Ordem:**

0 1 3 4 2 5 6



# Percurso em Largura

## *(Breadth-First Search Traversal)*

```
void bfsTraversal( Digraph graph )
{
    boolean[] found = new boolean[ graph.numNodes() ];

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

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

# Árvore em Largura (iterativo)

```
void bfsExplore( Digraph graph,  boolean[] found,  Node root )
{
    Queue<Node> waiting = new QueueIn...<Node>(?);
    waiting.enqueue(root);
    found[root] = true;
    do {
            Node node = waiting.dequeue();
            // PROCESS(node)
            for every Node v in graph.outAdjacentNodes(node)
                if ( !found[v] )
                {   waiting.enqueue(v);
                    found[v] = true;
                }
        }
    while ( !waiting.isEmpty() );
}
```

# Complexidades

| **Implementação do Grafo** $(V, A)$ | **Percurso em Largura** (grafo orientado ou não orientado) |
|---|---|
| Matriz | $\Theta(\|V\|^2)$ |
| Vetor de Listas | $\Theta(\|V\| + \|A\|)$ |

**waiting**.size() $\leq |V| - 1$