Capítulo IV

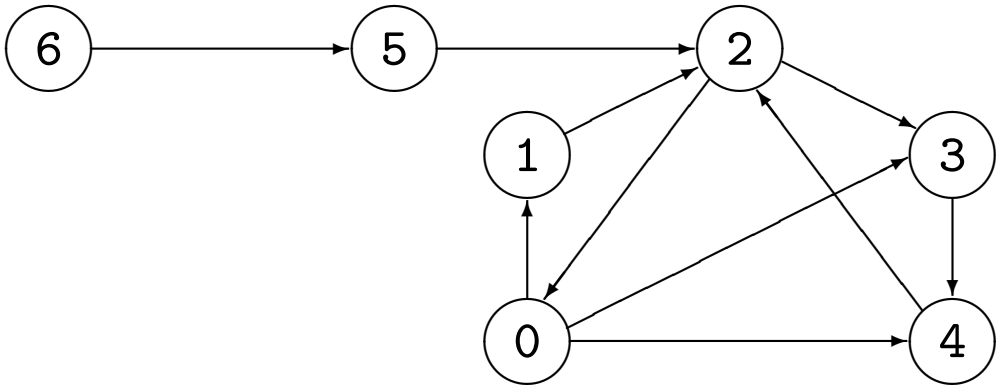
Percurso em Largura

(num grafo orientado ou não orientado)

Percurso em Largura

0

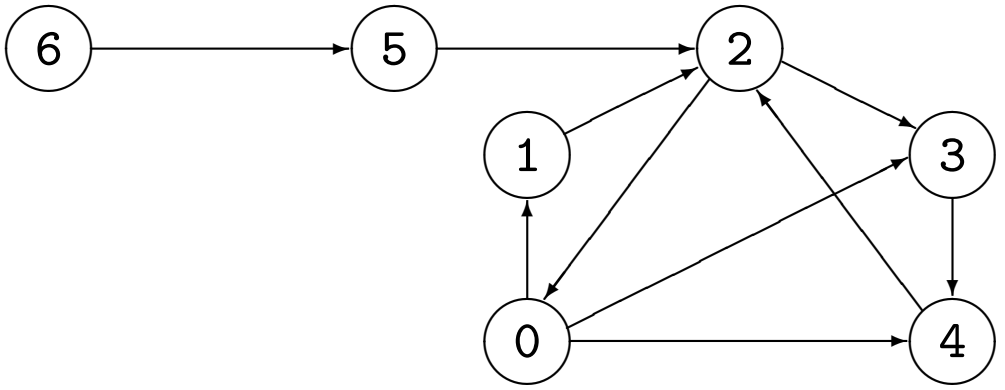
Ordem:



Percurso em Largura

0

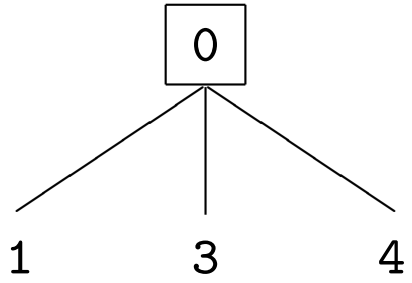
Ordem:



FIFO:

0

Percurso em Largura

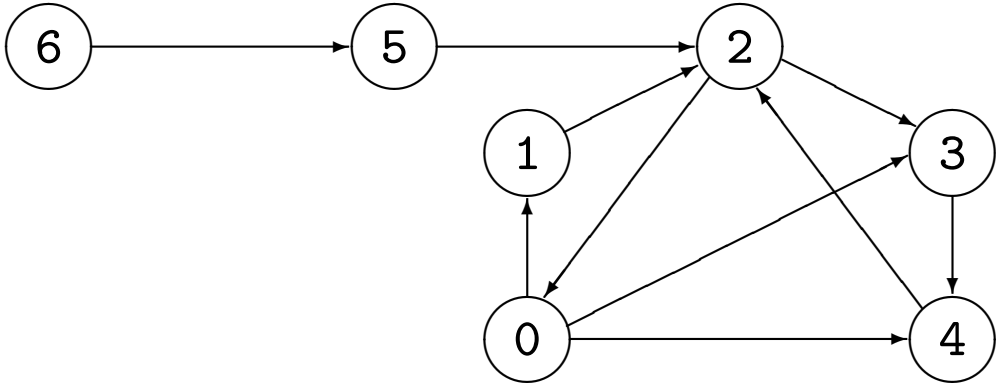


Ordem:

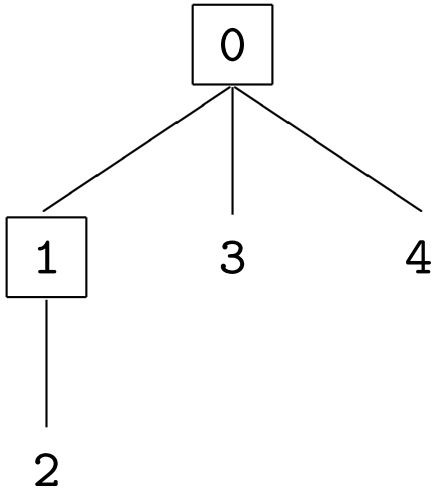
0

FIFO:

1 3 4

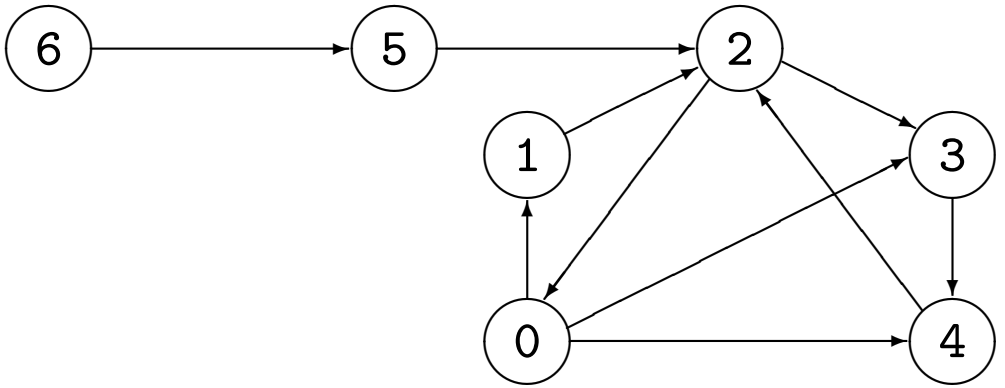


Percurso em Largura



Ordem:

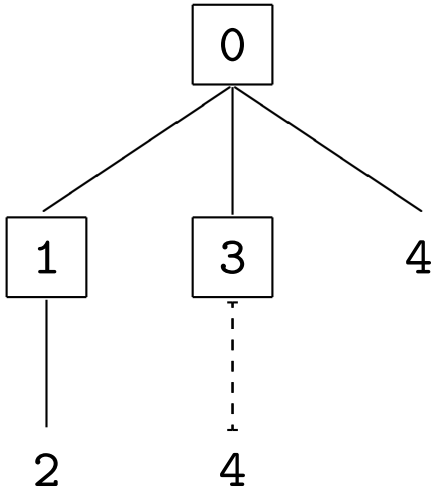
0 1



FIFO:

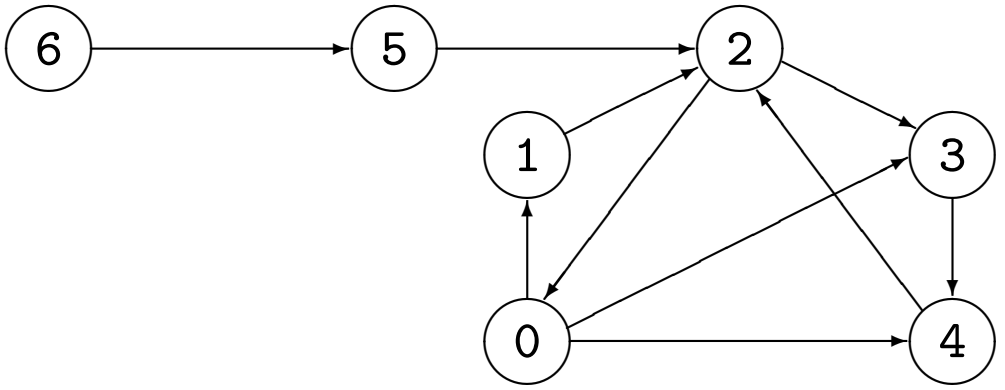
3 4 2

Percurso em Largura



Ordem:

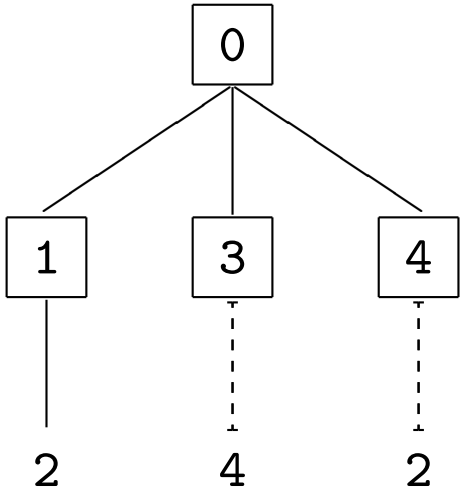
0 1 3



FIFO:

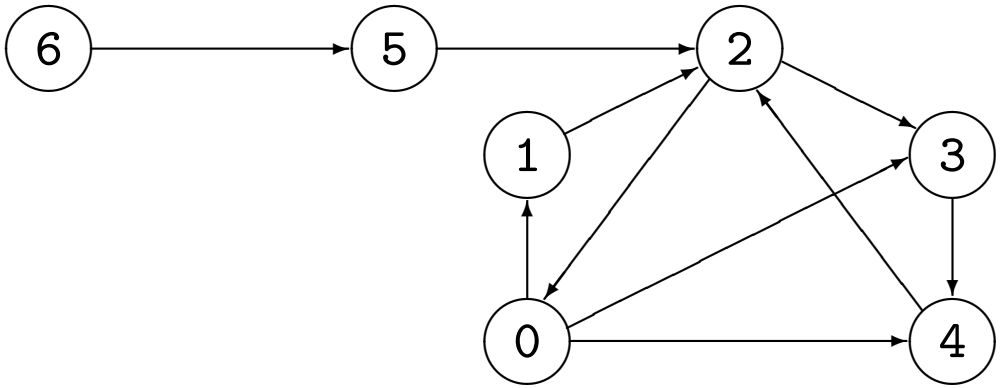
4 2

Percurso em Largura



Ordem:

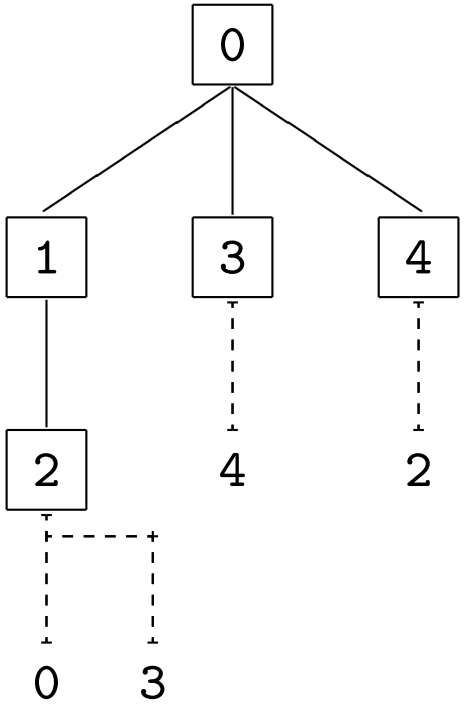
0 1 3 4



FIFO:

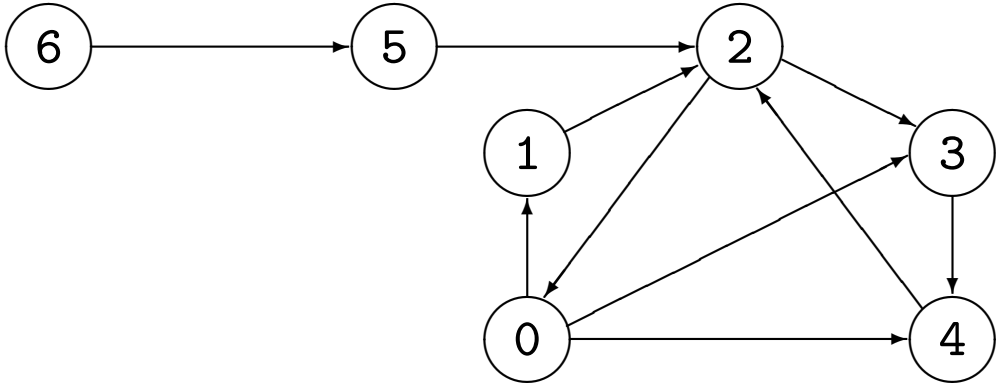
2

Percurso em Largura



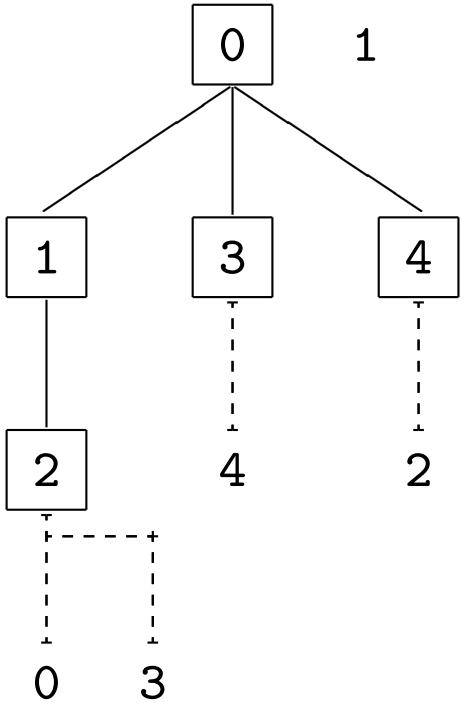
Ordem:

0 1 3 4 2



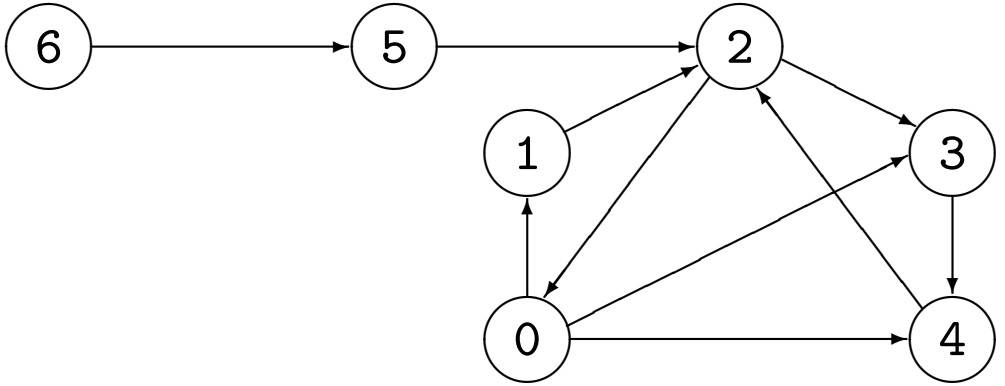
FIFO:

Percurso em Largura

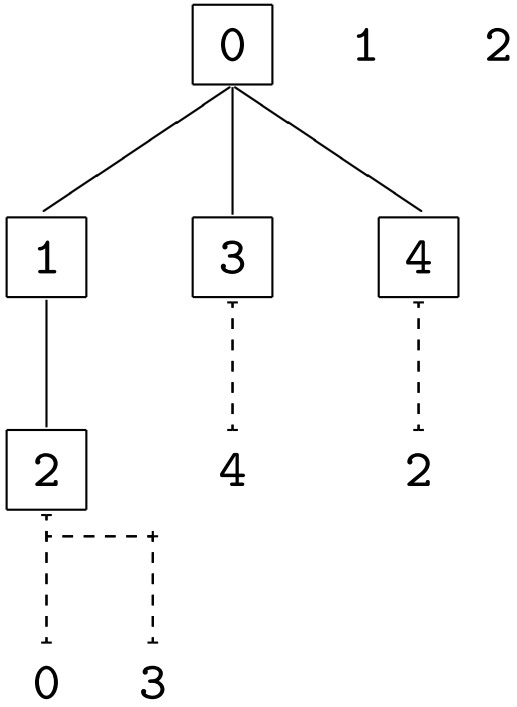


Ordem:

0 1 3 4 2

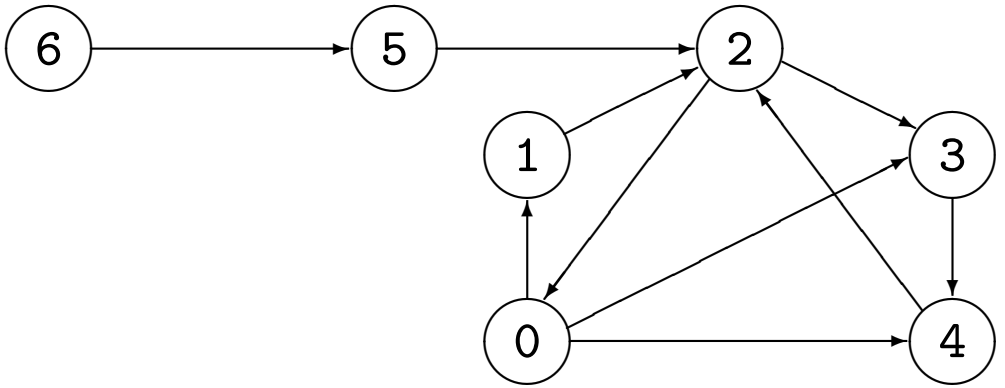


Percurso em Largura

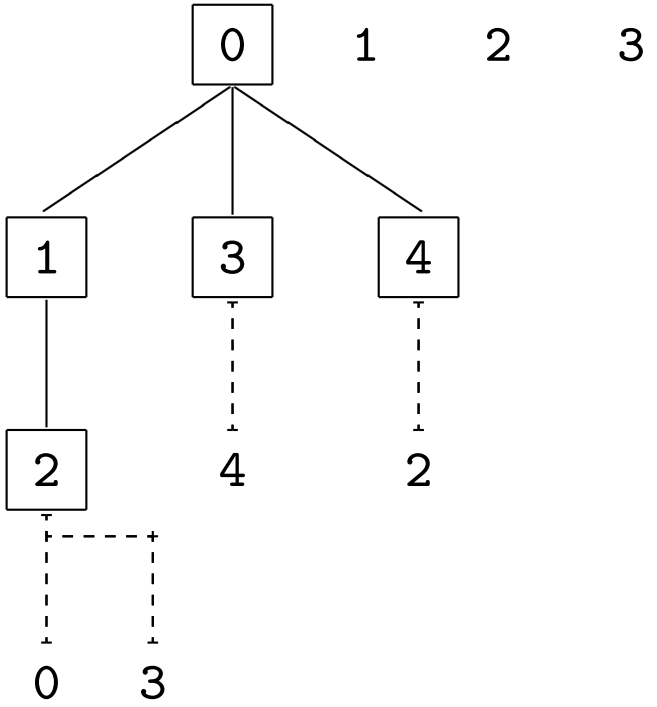


Ordem:

0 1 3 4 2

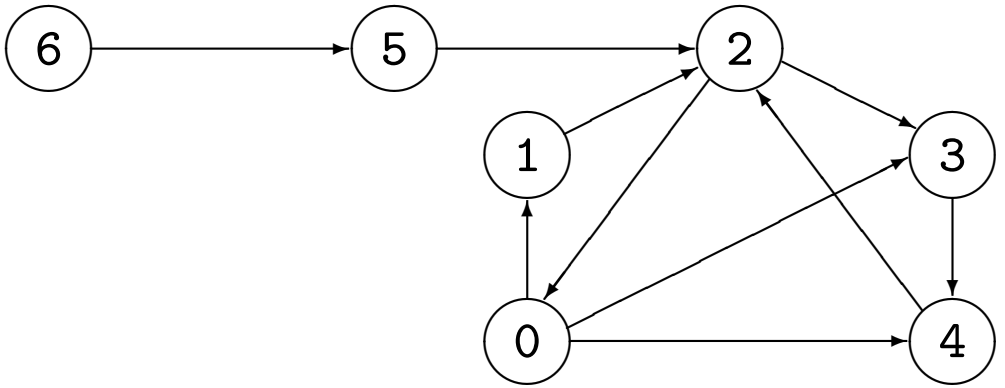


Percurso em Largura

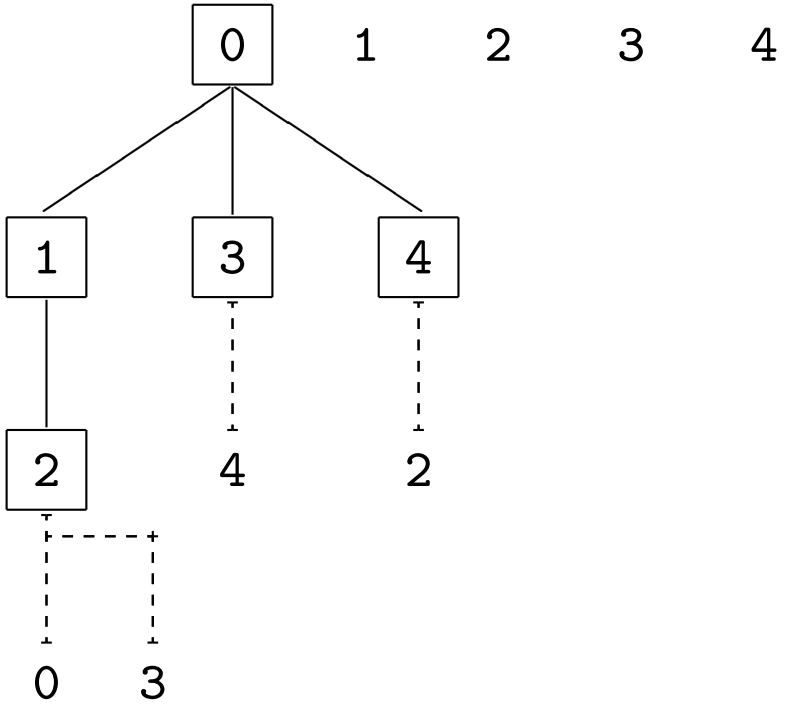


Ordem:

0 1 3 4 2

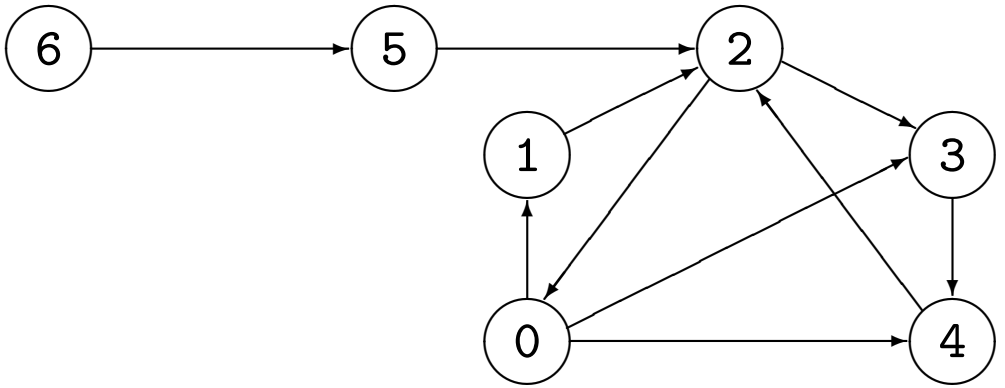


Percurso em Largura

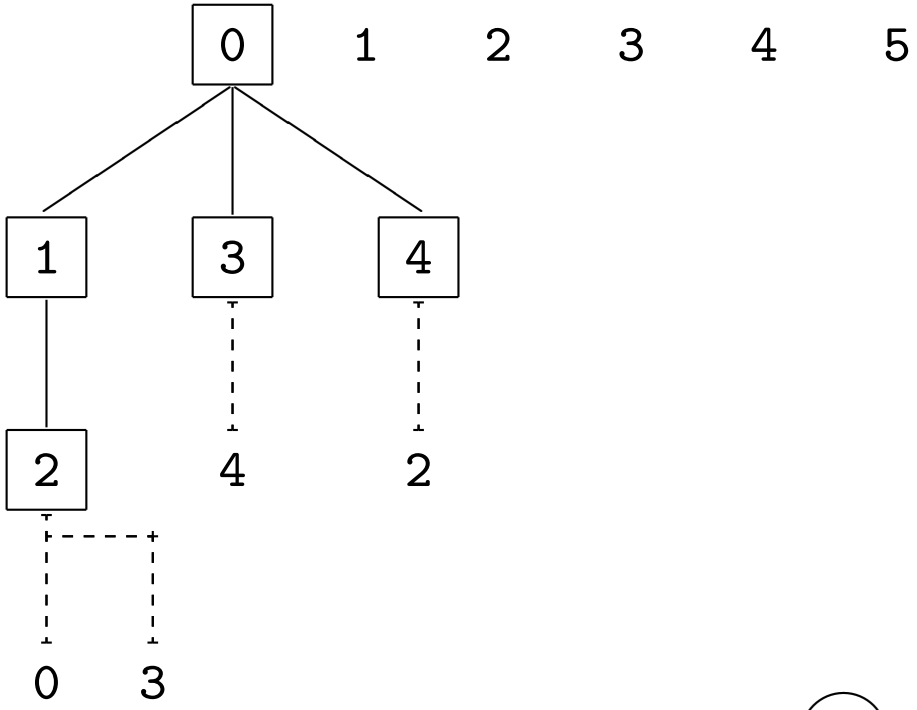


Ordem:

0 1 3 4 2

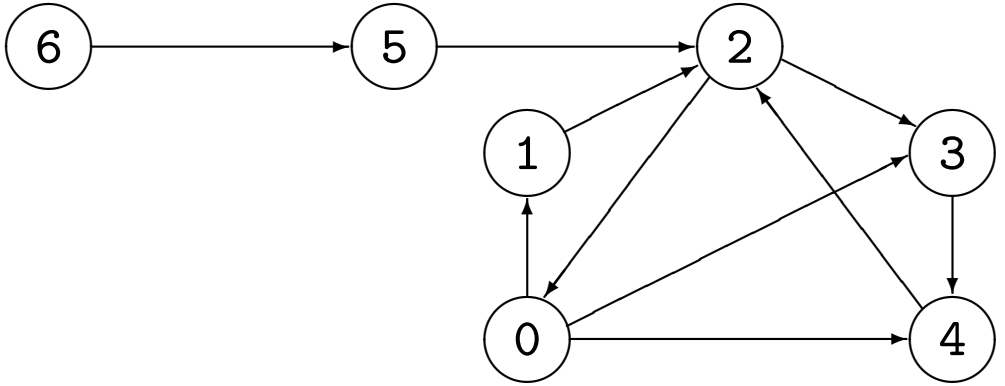


Percurso em Largura

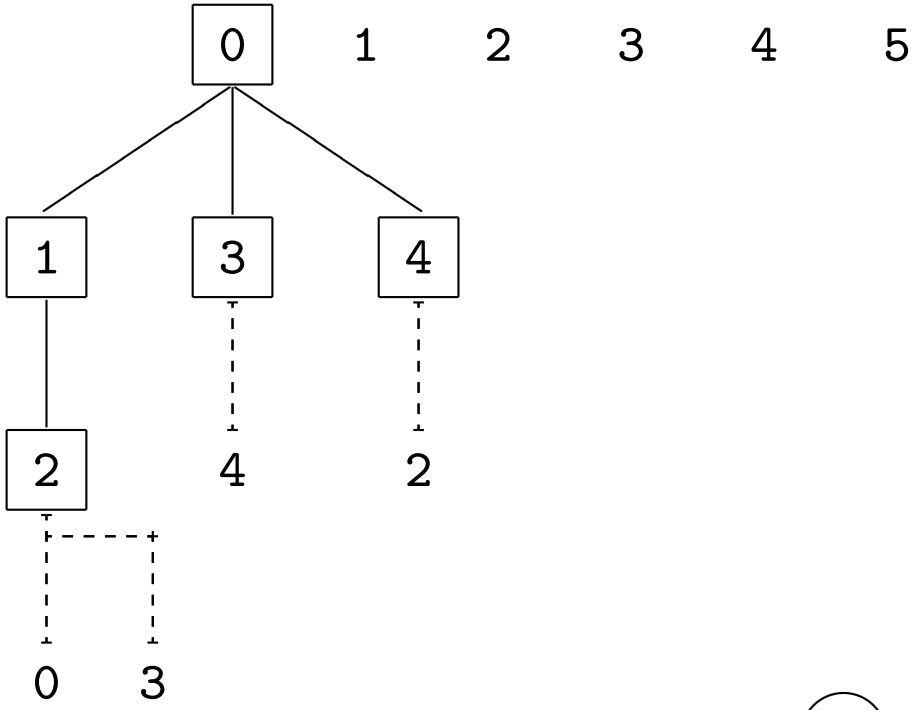


Ordem:

0 1 3 4 2

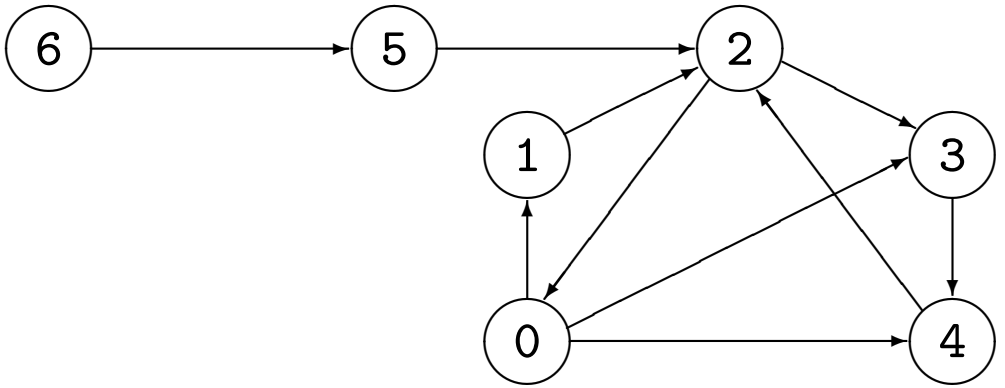


Percurso em Largura



Ordem:

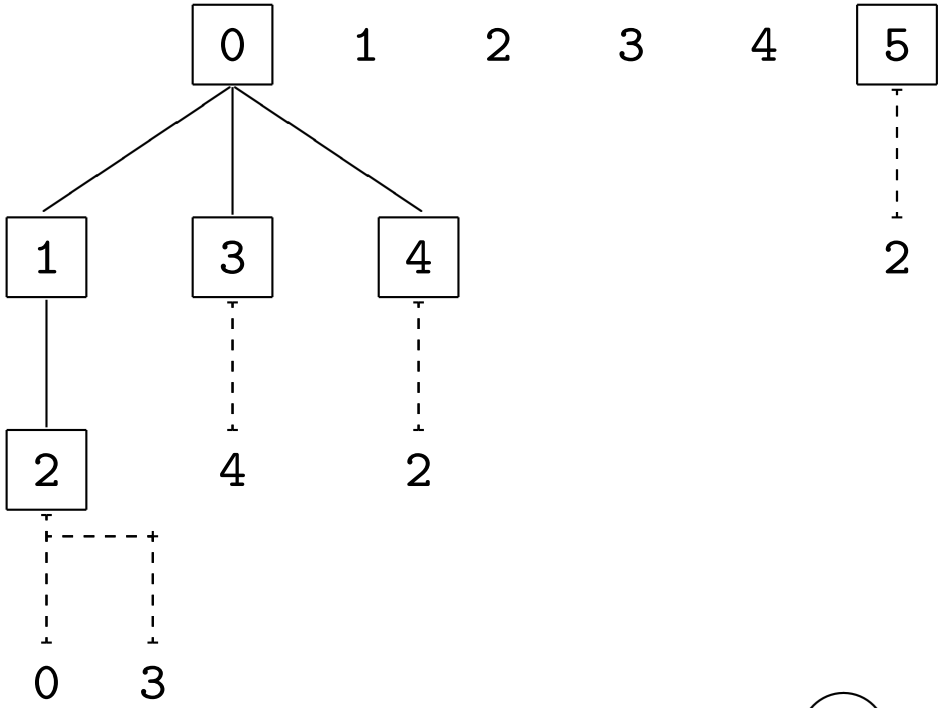
0 1 3 4 2



FIFO:

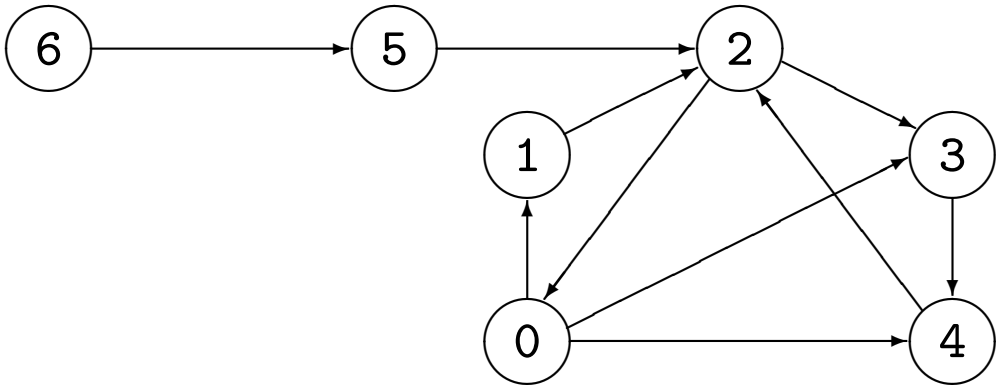
5

Percurso em Largura



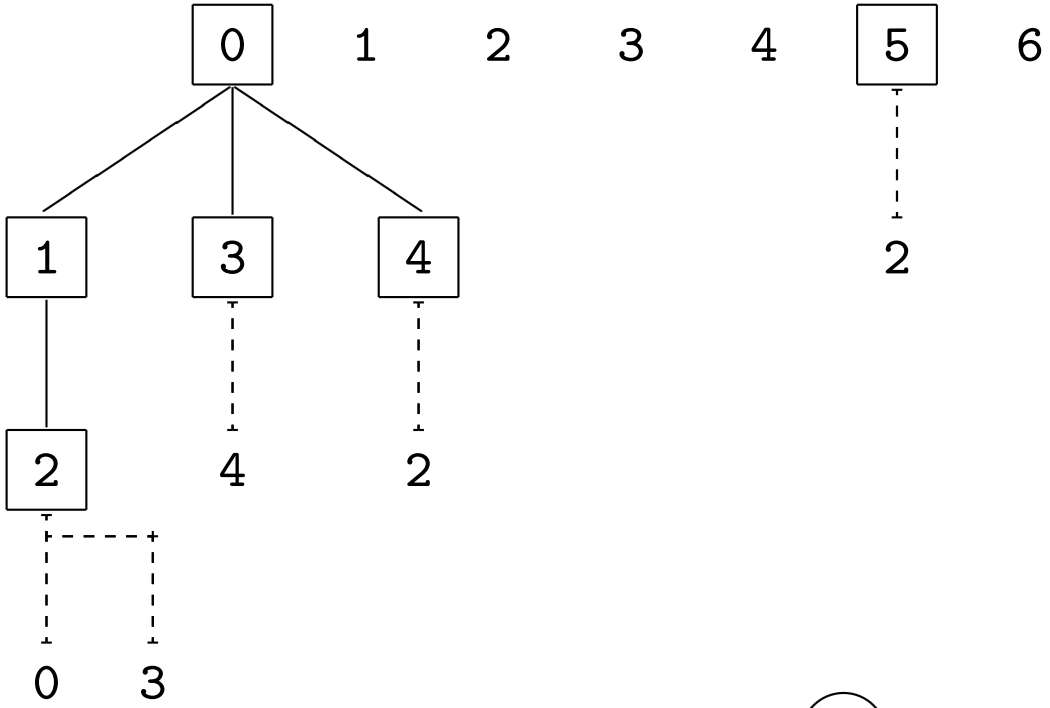
Ordem:

0 1 3 4 2 5



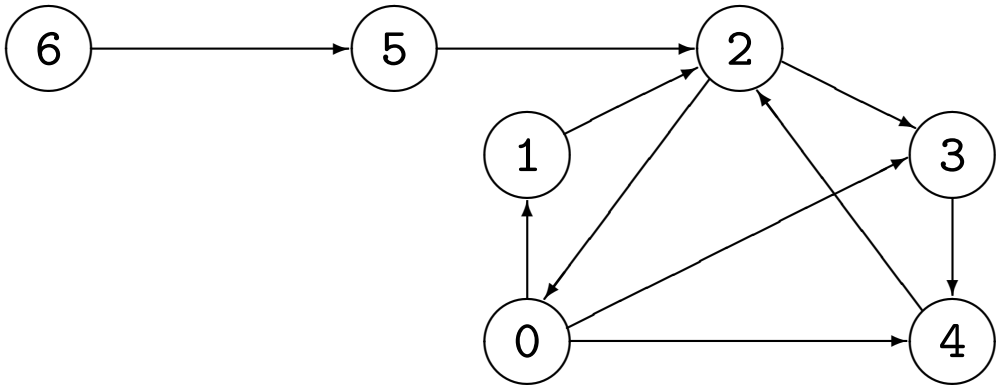
FIFO:

Percurso em Largura

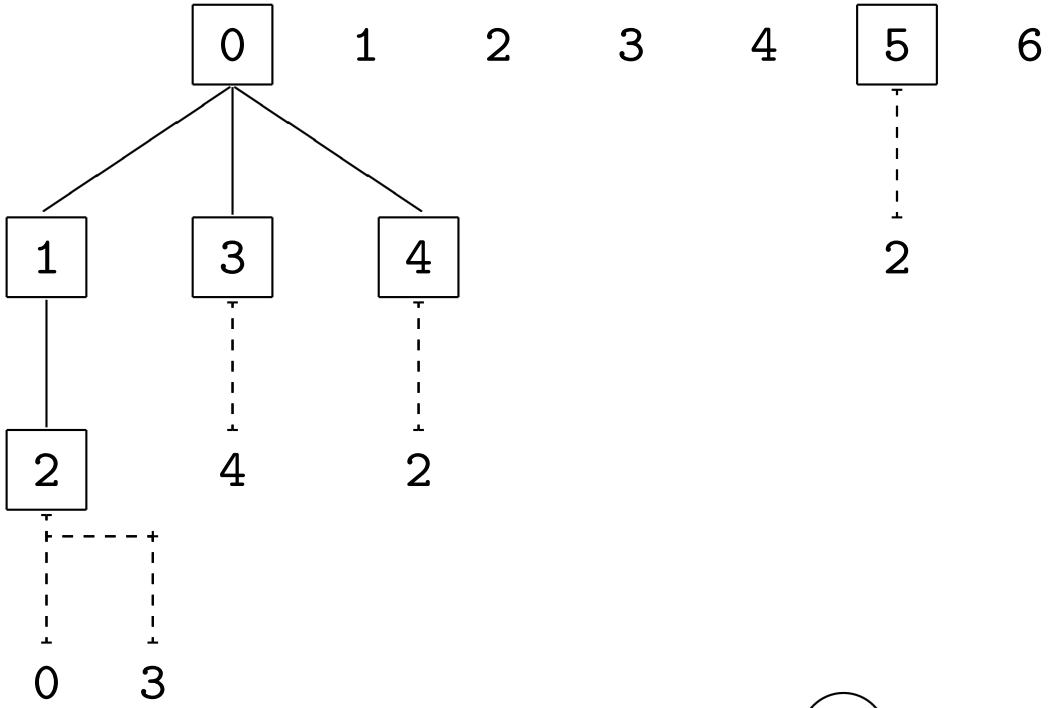


Ordem:

0 1 3 4 2 5

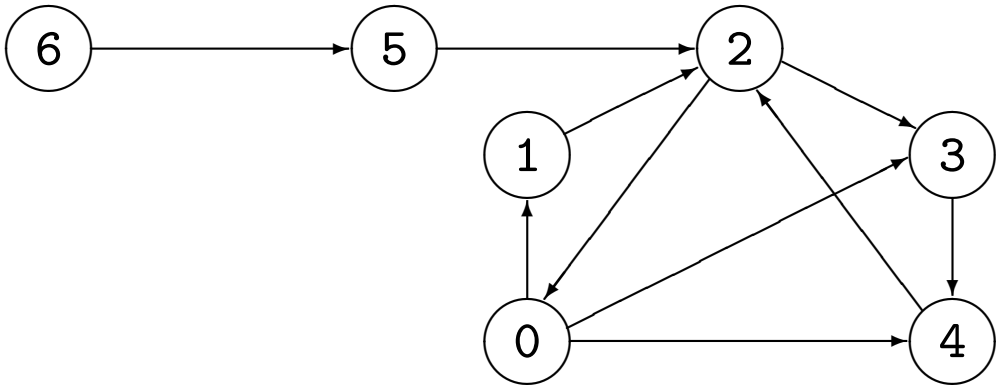


Percurso em Largura



Ordem:

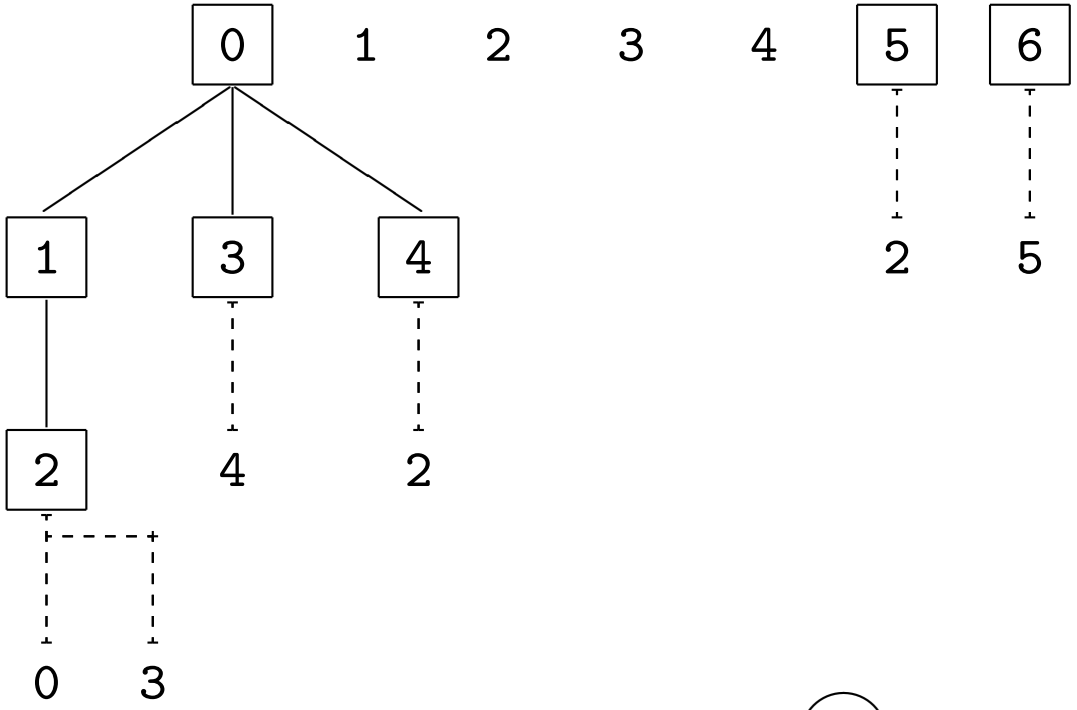
0 1 3 4 2 5



FIFO:

6

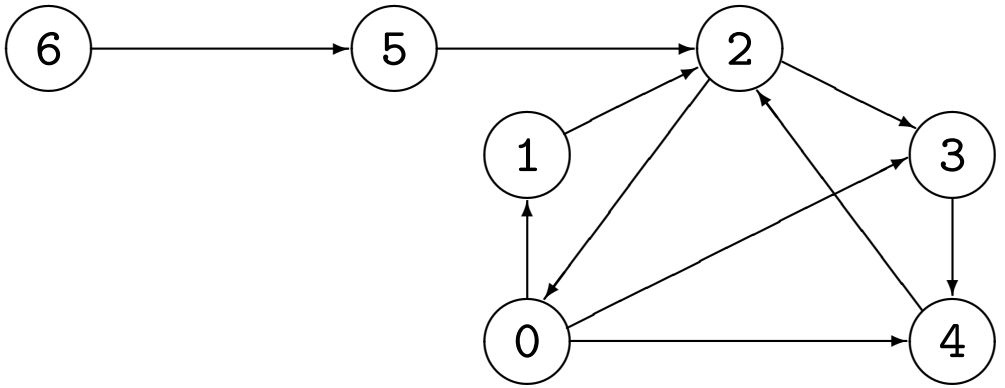
Percurso em Largura



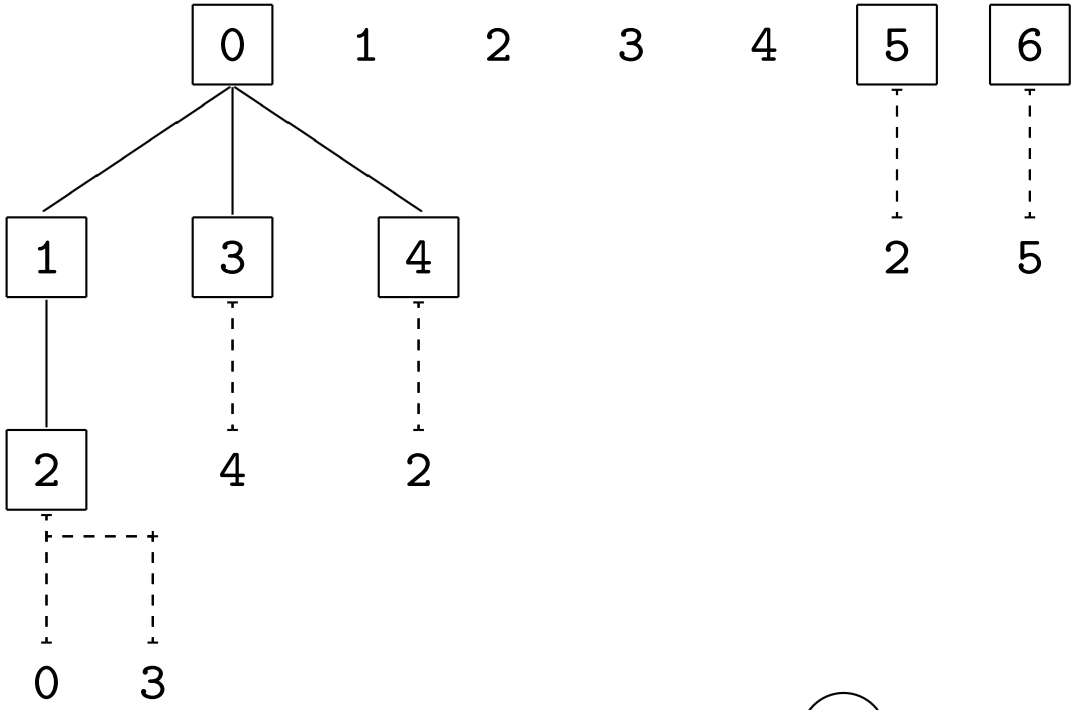
Ordem:

0 1 3 4 2 5 6

FIFO:

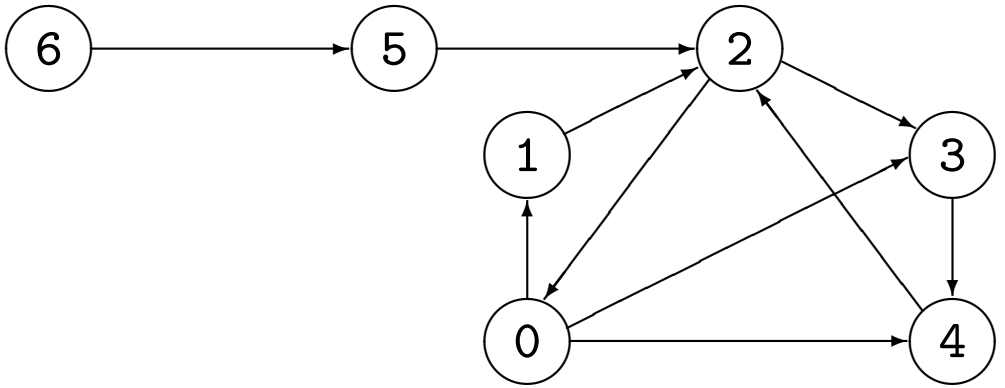


Percurso em Largura



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...<>(?);
    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() );
}
```

Complexidade de **bfsExplore**

- **Por cada vértice (w)**
  - Se PROCESS(w) não for constante, considerar o seu custo
  - Remove-se w da fila $\Theta(1)$
  - Iteram-se os sucessores de w
    - Grafo em matriz de adjacências $\Theta(|V|)$
    - Grafo em vetor de listas de adjacências (suc.) $\Theta(|\text{Suc}(w)|)$
  - Inserem-se os suc. nunca inseridos de w na fila $O(|\text{Suc}(w)|)$
- **Custo de todas as execuções** (#DEQUEUE = #ENQUEUE)
  - 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, $\sum_{w \in V} |\text{Suc}(w)| = 2 \times |A|$.

Complexidade Temporal de **bfsTraversal**
(se enqueue, dequeue e isEmpty de Queue forem $\Theta(1)$)

Grafo em matriz de adjacências

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

Grafo em vetor de listas de adjacências (suc.)

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

Complexidade Espacial de **bfsTraversal**

Vetor found	$\Theta(\|V\|)$
Fila waiting	$O(\|V\|)$
TOTAL	$\Theta(\|V\|)$