

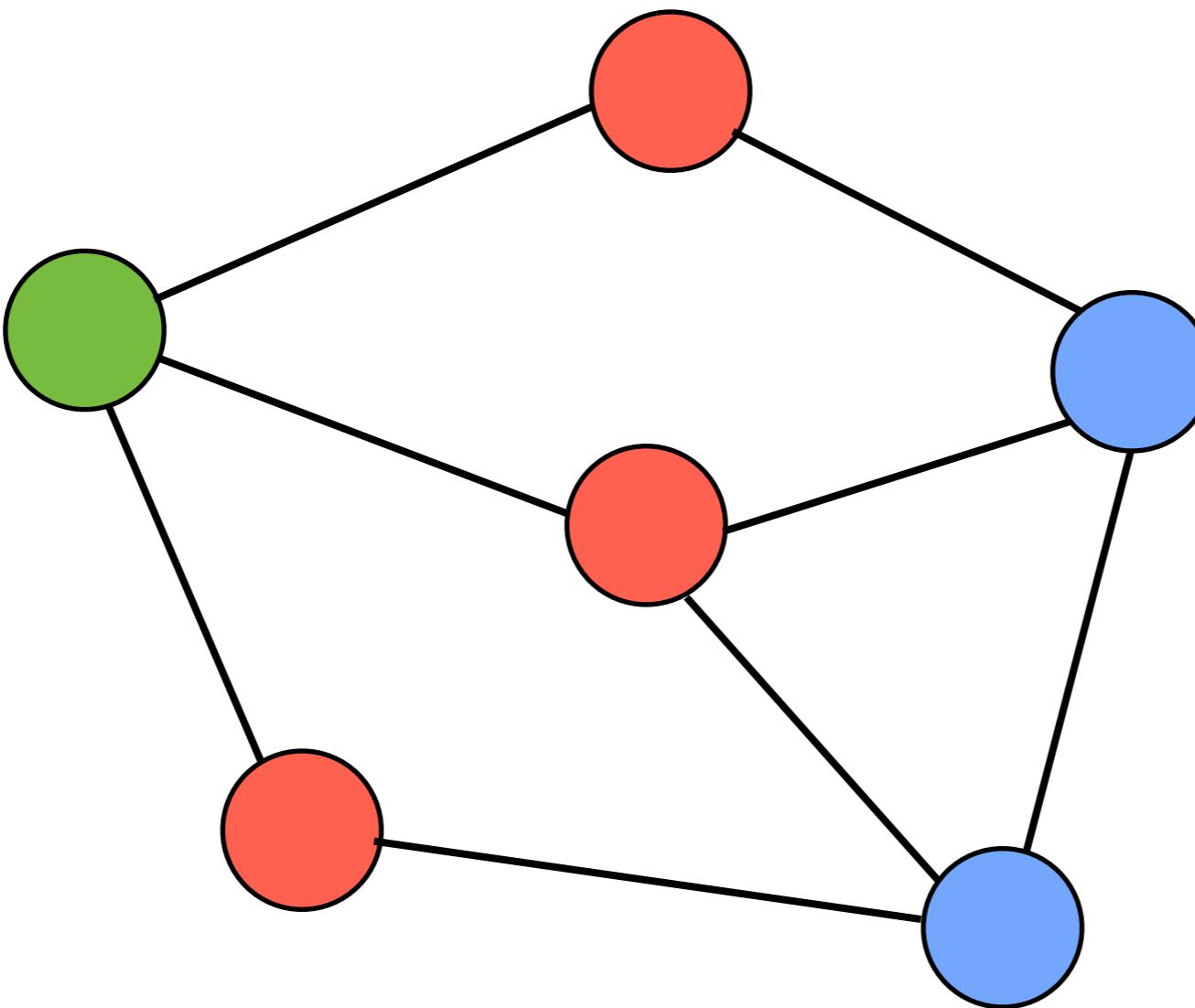
TREINAMENTO PARA COMPETIÇÕES DE PROGRAMAÇÃO

GRAFOS PARTE I

TAP
TURMA INICIANTE

Murilo Adriano Vasconcelos e Paulo Cezar P. Costa - INF/UFG
<http://murilo.wordpress.com>

Grafos: Introdução



Grafos

- O que é um grafo?
- Tipos de grafos
- Representações de grafos
- Algoritmos de busca
- Aplicações dos algoritmos de busca

O que é um grafo?

- Uma representação abstrata que pode ser usada para descrever a organização de sistemas de transportes, interações humanas, redes de telecomunicação entre outros.

O que é um grafo?

Matematicamente denotado como:

$G = (V, E)$, onde V é o conjunto de vértices e E o conjunto de arestas (edges) formado por pares ordenados ou não de vértices do conjunto V .

Tipos de grafos

- Direcionados / Não-direcionados
- Ponderados / Não-ponderados
- Cíclicos / Acíclicos
- Esparsos / Densos

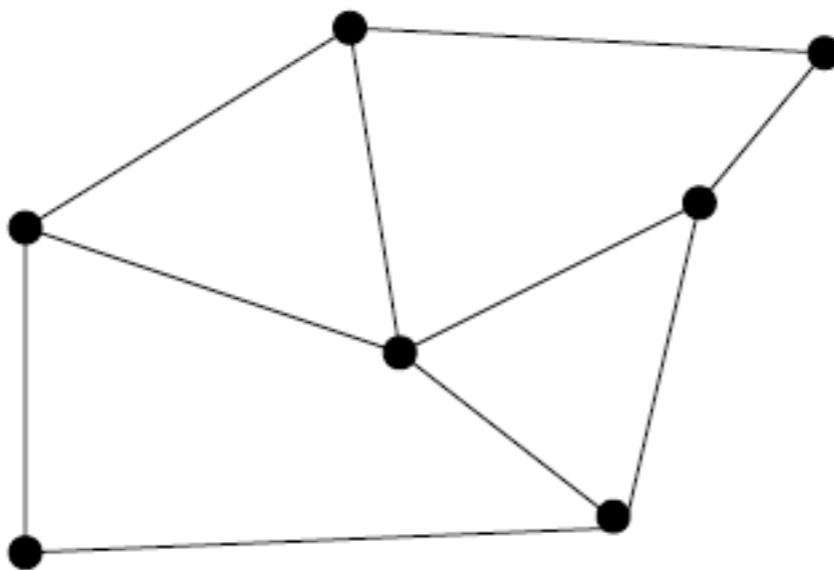
Grafos Não-Direcionados

- Um grafo $G(V, E)$ é dito **não-direcionado** se a existência de uma aresta $(u, v) \in E$ implica que a aresta (v, u) também pertence a E
- Estradas interligando cidades são exemplos de relações que podem ser representadas por grafos não-direcionados uma vez que as estradas possuem duplo sentido

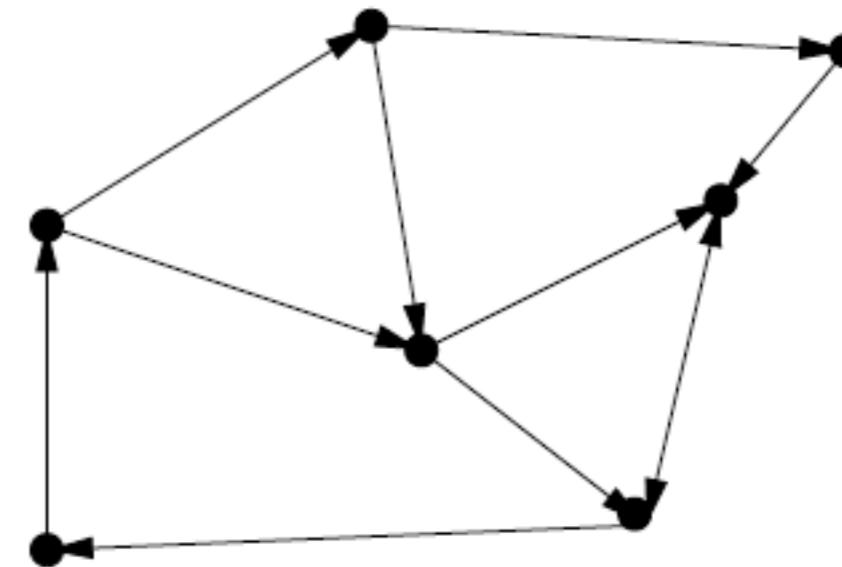
Grafos Direcionados

- Se a presença de uma aresta (u, v) não implica na existência da aresta (v, u) então dizemos que o grafo é **direcionado**
- Ruas de uma cidade podem ser vistas como um grafo direcionado pois nem todas as ruas possuem mão-dupla.

Grafos Não-Direcionados / Direcionados



Não-direcionado



Direcionado

*Figuras retiradas de [TADM] p.147

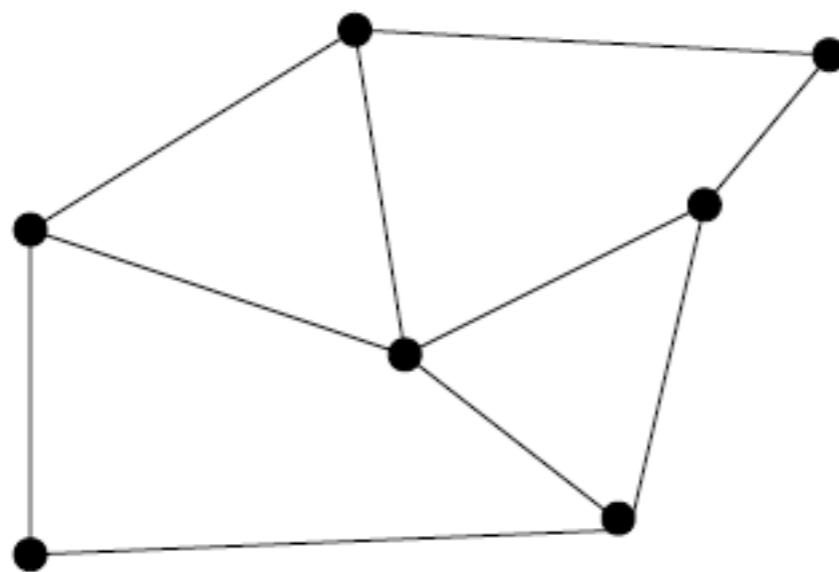
Grafos Ponderados

- Em um grafo **ponderado**, cada aresta (ou vértice) possui um valor que pode denotar o peso, custo, ganho etc.
- Em um grafo representando as cidades e as estradas que as interligam, os valores das arestas podem representar, por exemplo, o comprimento, o tempo de direção, o limite de velocidade ou o custo do pedágio de cada estrada.

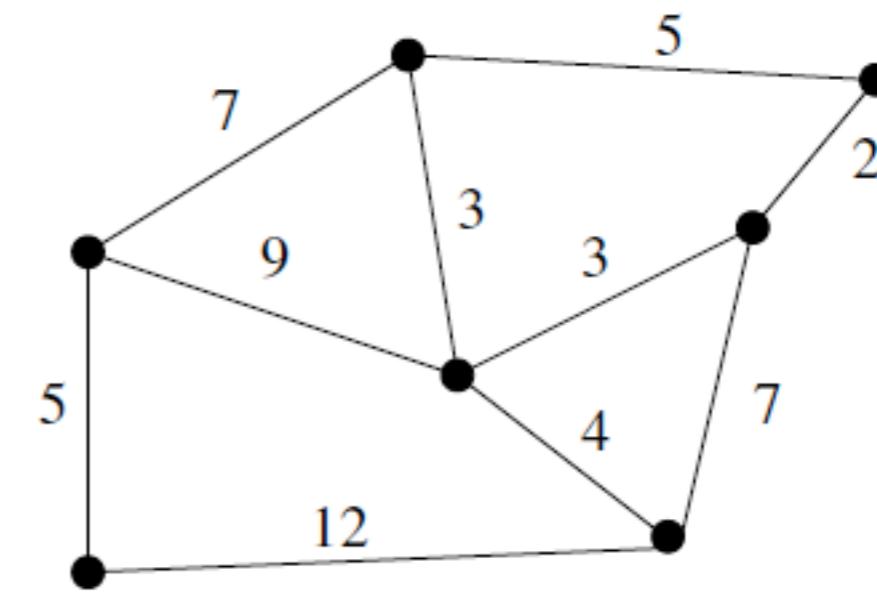
Grafos Não-ponderados

- Em um grafo **não-ponderado**, não há distinção de custo ou valor entre as arestas. Isto é, cada aresta possui o mesmo valor.
- Este tipo de grafo é utilizado quando o que importa são somente os relacionamentos.

Grafos Não-Ponderados / Ponderados



Não-ponderado



Ponderado

*Figuras retiradas de [TADM] p.147

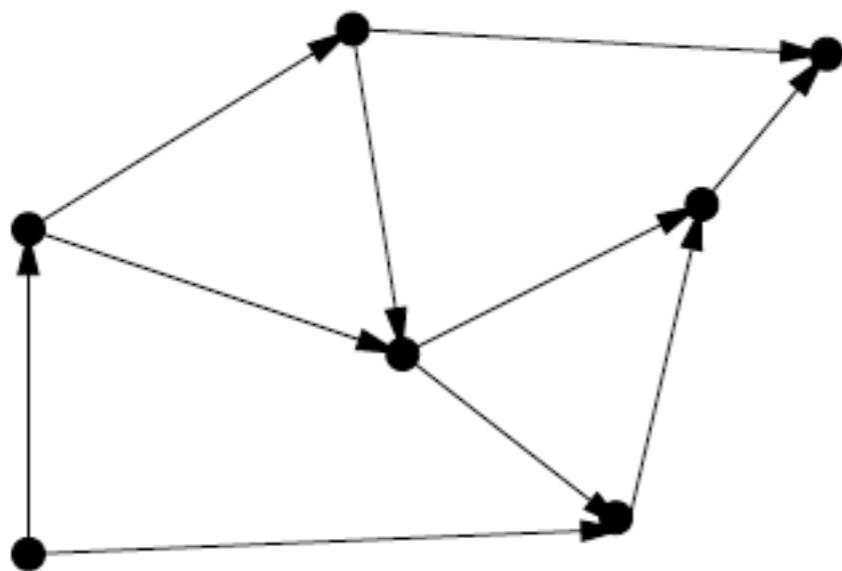
Grafos Cíclicos

- Um grafo **cíclico** é um grafo que não contém ciclos, ou seja, se partindo de um vértice v , conseguimos chegar a um vértice u , então a partir de u não conseguimos chegar novamente a v a não ser voltando pelo caminho usado de v até u

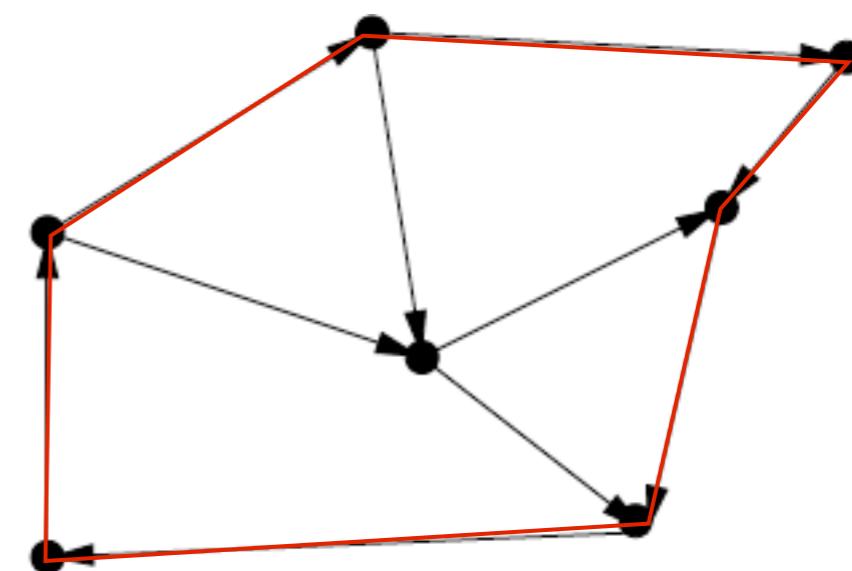
Grafos Acíclicos

- Um grafo é **acíclico** se não contém ciclos
- Grafos conexos não-direcionados acíclicos?
- **Árvores!!!**
- Grafos direcionados acíclicos (*DAGs*)

Grafos Acíclicos / Cíclicos



Acíclico



Cíclico

*Figuras retiradas de [TADM] p.147

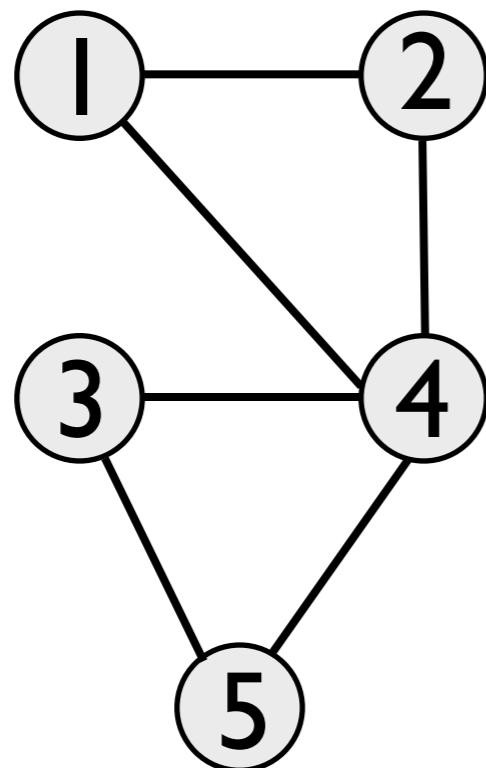
Grafos Esparsos / Densos

- Um grafo $G(V, E)$ é chamado **esparso** se $|E|$ for muito menor que $|V|^2$
- Se um grafo possui o número de arestas ($|E|$) próximo do número de vértices ao quadrado ($|V|^2$), então dizemos que este grafo é **denso**.

Representações de grafos

- Duas formas principais
 - Matrizes de adjacência
 - Lista de adjacência
- É de suma importância saber identificar a representação correta para a resolução de problemas

Representações de grafos: Matrizes de adjacência



-	1	2	3	4	5
1	1	1	0	1	0
2	1	1	0	1	0
3	0	0	1	1	1
4	1	1	0	1	1
5	0	0	1	1	1

Representações de grafos: Matrizes de adjacência

- Vantagens:
 - Facilidade e rapidez de implementação
 - Fácil checar se a aresta (i, j) existe ou não
- Desvantagens:
 - Alocar uma matriz n^2 pode não ser bom
 - Ex.: 3000 cidades, array de 9,000,000 posições de 4 bytes!

Representações de grafos: Listas de adjacência

- Podemos representar grafos esparsos de uma forma mais eficiente com listas de adjacência
- Cada vértice possui uma lista de vértices associada
- Se existe uma aresta (u, v) no grafo então a lista do vértice u contém o vértice v

Representações de grafos: Listas de adjacência

- Vantagens
 - Menor consumo de memória ($n + m$)
 - Não contém arestas inexistentes no grafo original, diminuindo assim a complexidade de buscas no grafo
- Desvantagens
 - Codificação mais demorada
 - Checar se a aresta (u, v) existe é mais demorado

Representações de grafos: Comparativo

Comparação	Vencedor
Checar se (u, v) pertence ao grafo	matrizes de adjacência $O(1)$
Checar o grau de um vértice	listas de adjacência
Memória em grafos pequenos	listas de adjacência $(n + m)$ vs n^2
Inserção ou remoção de arestas	matrizes de adjacência $O(1)$ vs $O(d)$
Busca em grafos	listas de adjacência $\Theta(m + n)$ vs $\Theta(n^2)$
Velocidade de implementação	matrizes de adjacência
Melhor para a maioria dos problemas	listas de adjacência

*Sugerido por [TADM] p.152

Implementação: Matrizes de adjacência

```
// Declarando
const int MAX_VERTICES = 100;
tipo grafo[MAX_VERTICES][MAX_VERTICES];

// Inicializando
for (int i = 0; i < num_arestas; ++i)
    for (int j = i; j < num_arestas; ++j)
        grafo[i][j] = grafo[j][i] = 0;

// Populando
int u, v;
for (int i = 0; i < num_arestas; ++i) {
    cin >> u >> v;
    grafo[u][v] = w;
    // grafo[v][u] = w; se não for direcionado
    // grafo[u][v] = true; se não for ponderado
}
```

Implementação: Listas de adjacência

```
#include <list> // std::list
#include <utility> // pair<> e make_pair()
using namespace std;

// Declarando
const int MAX_VERTICES = 100;
list< pair<tipo, int> > grafo[MAX_VERTICES];

// Inicializando
for (int i = 0; i < num_arestas; ++i)
    grafo[i].clear();

// Populando
int u, v, w;
for (int i = 0; i < num_arestas; ++i) {
    cin >> u >> v >> w;
    grafo[u].push_back(make_pair(v,w));
}
```

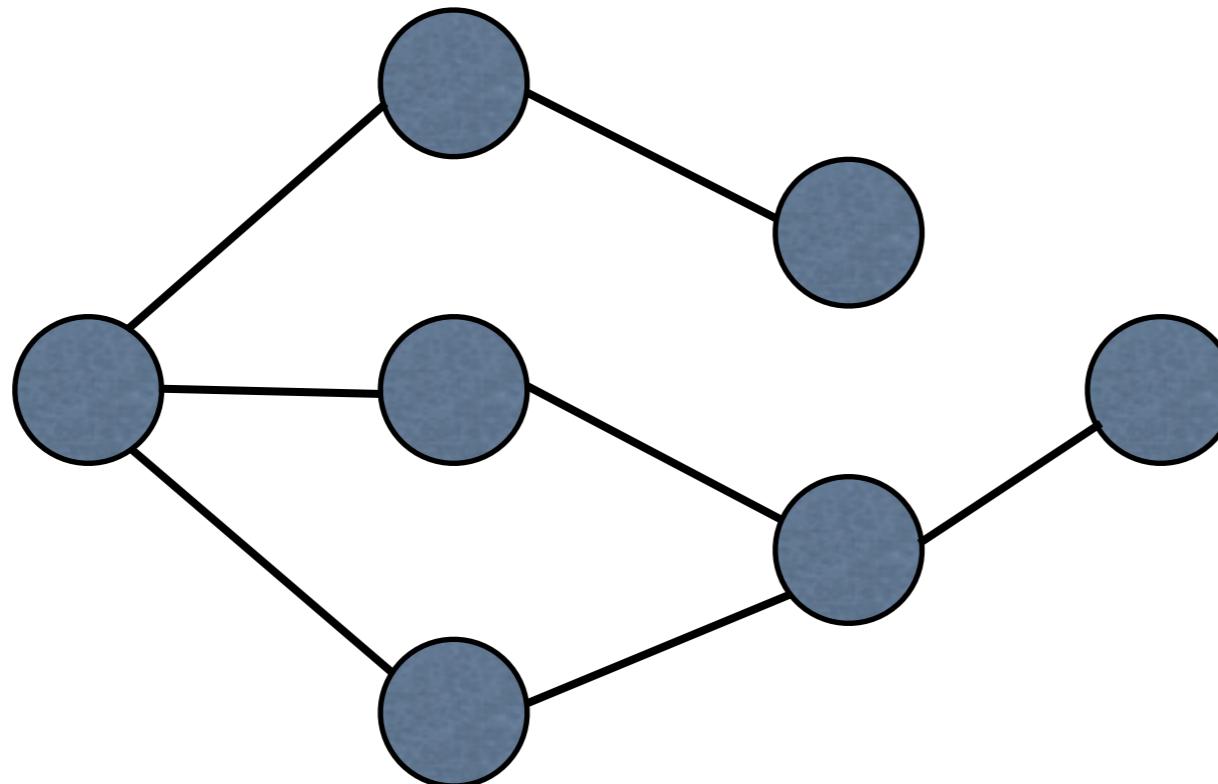
Nem tão mais complicado assim... :)

Algoritmos de busca em grafos

- Dois principais:
 - Busca em largura (ou *Breadth-First Search* - **BFS**)
 - Busca em profundidade (ou *Depth-First Search* - **DFS**)

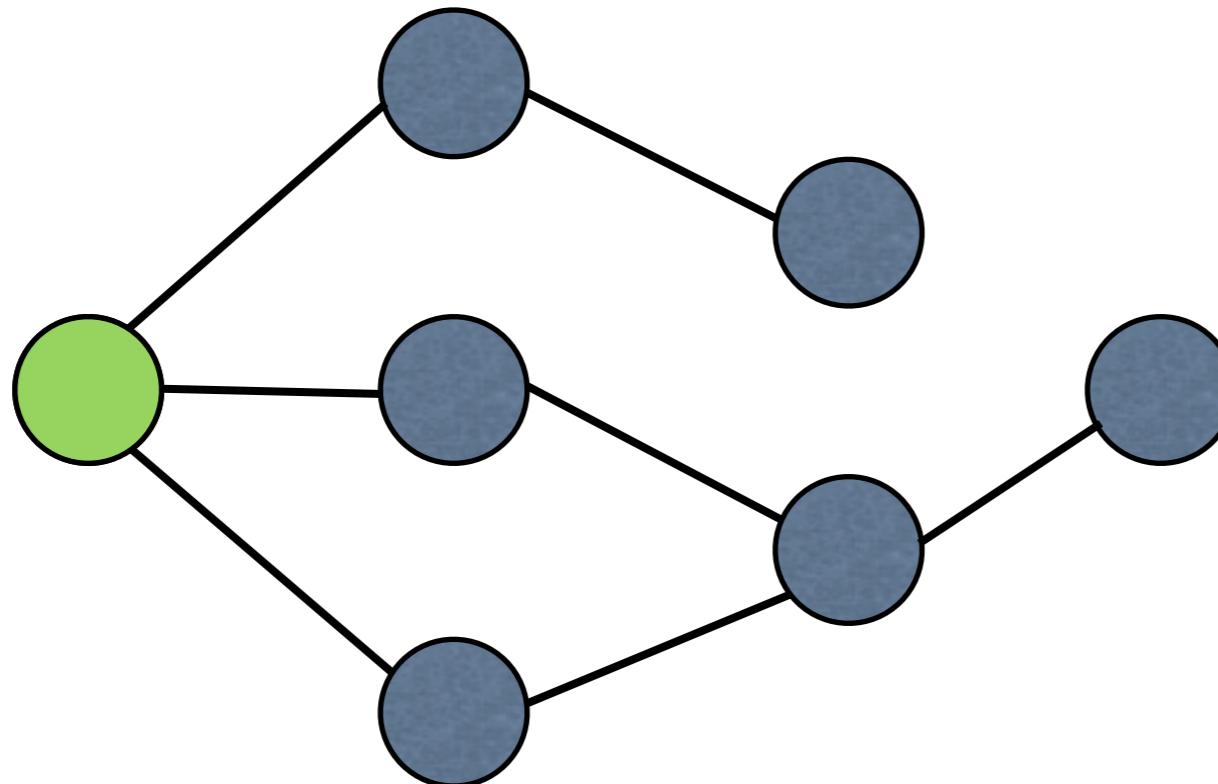
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



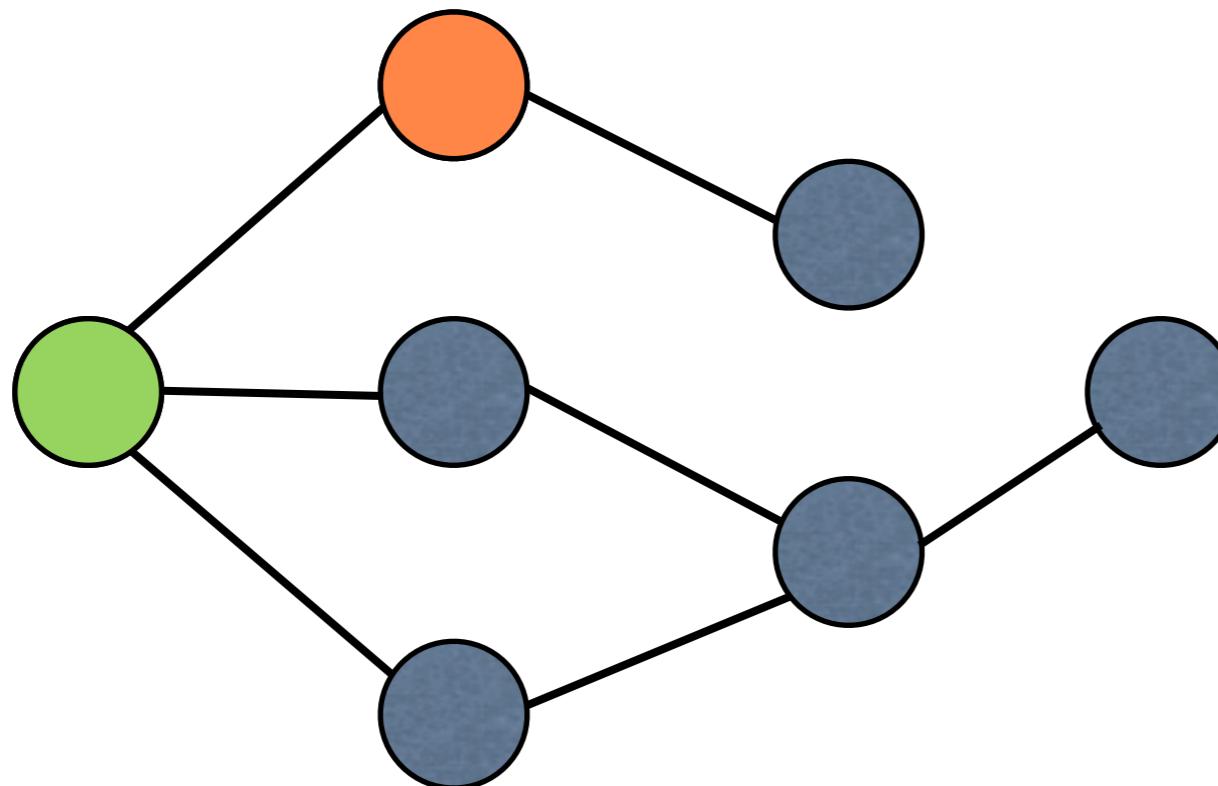
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



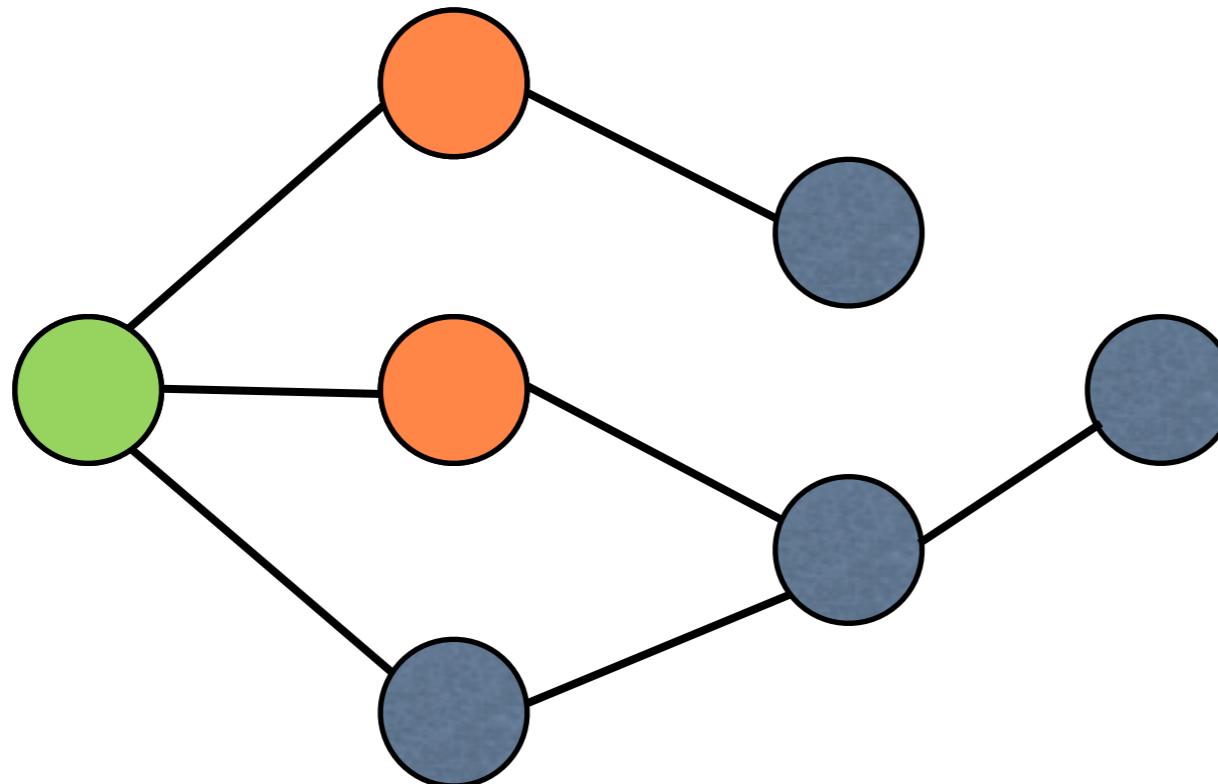
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



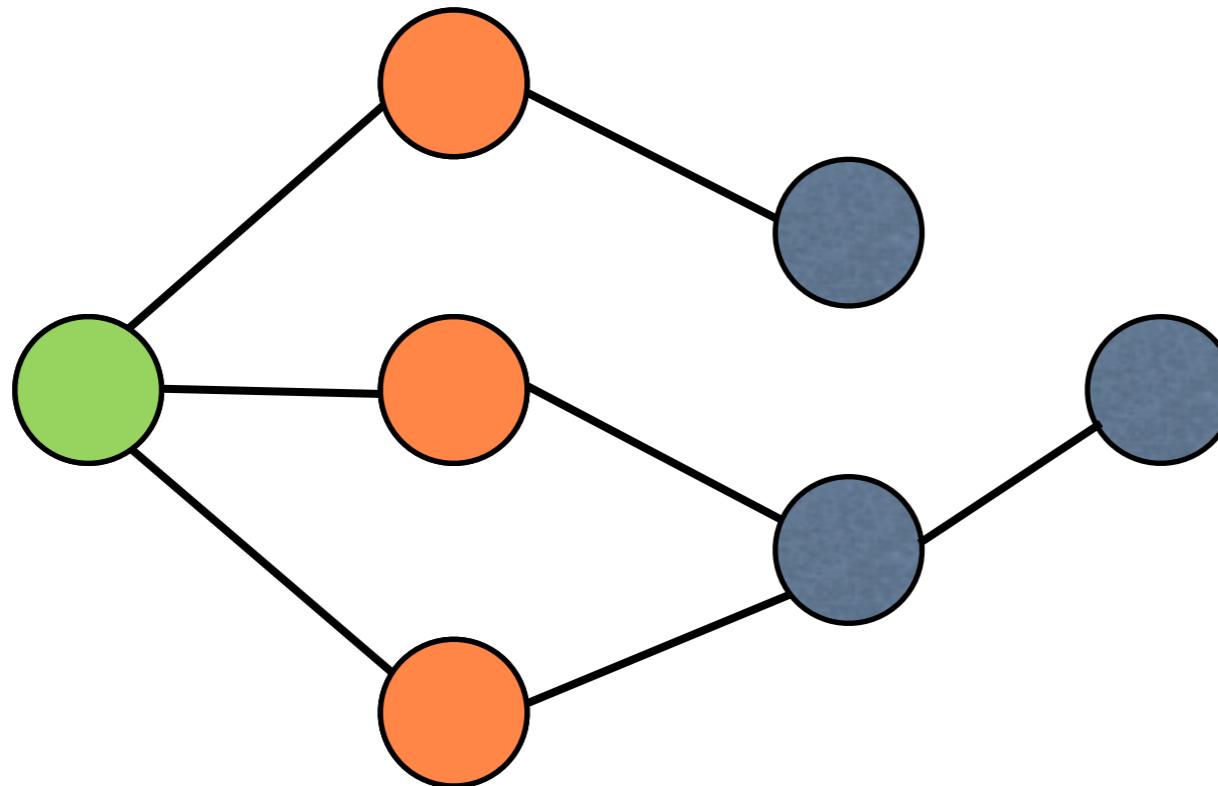
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



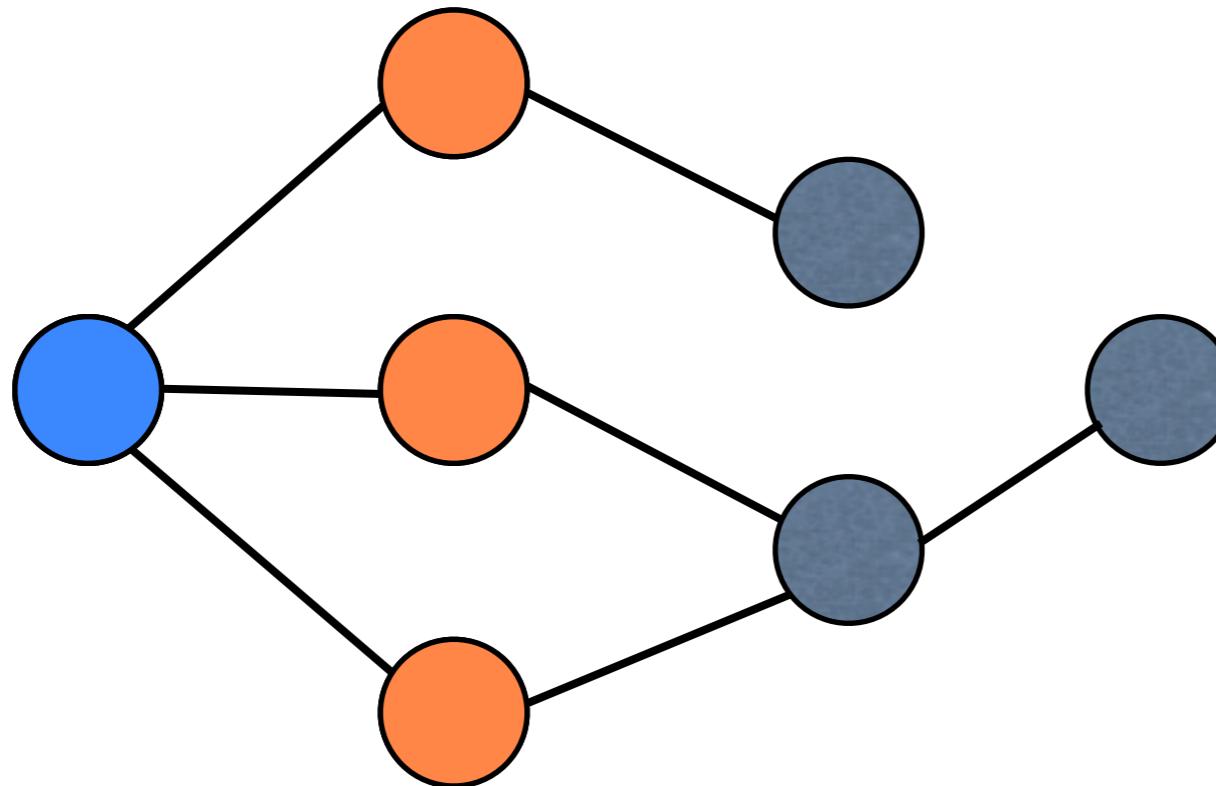
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



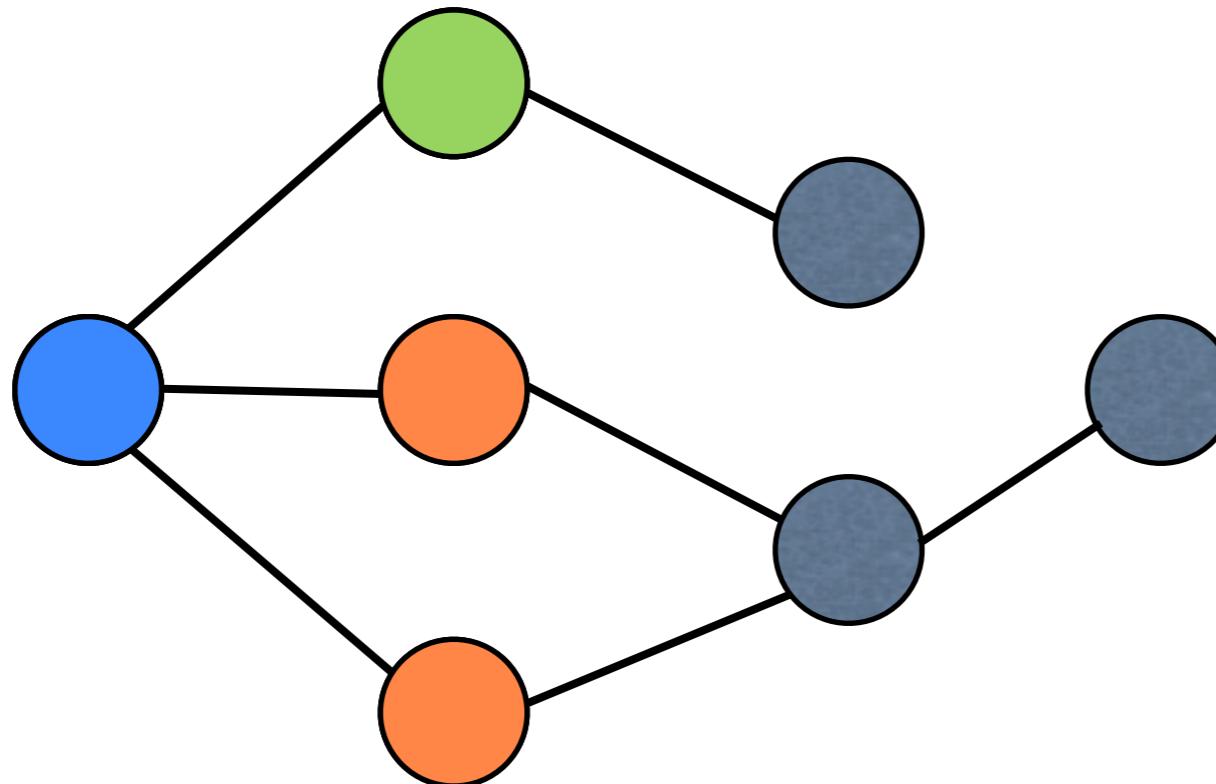
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



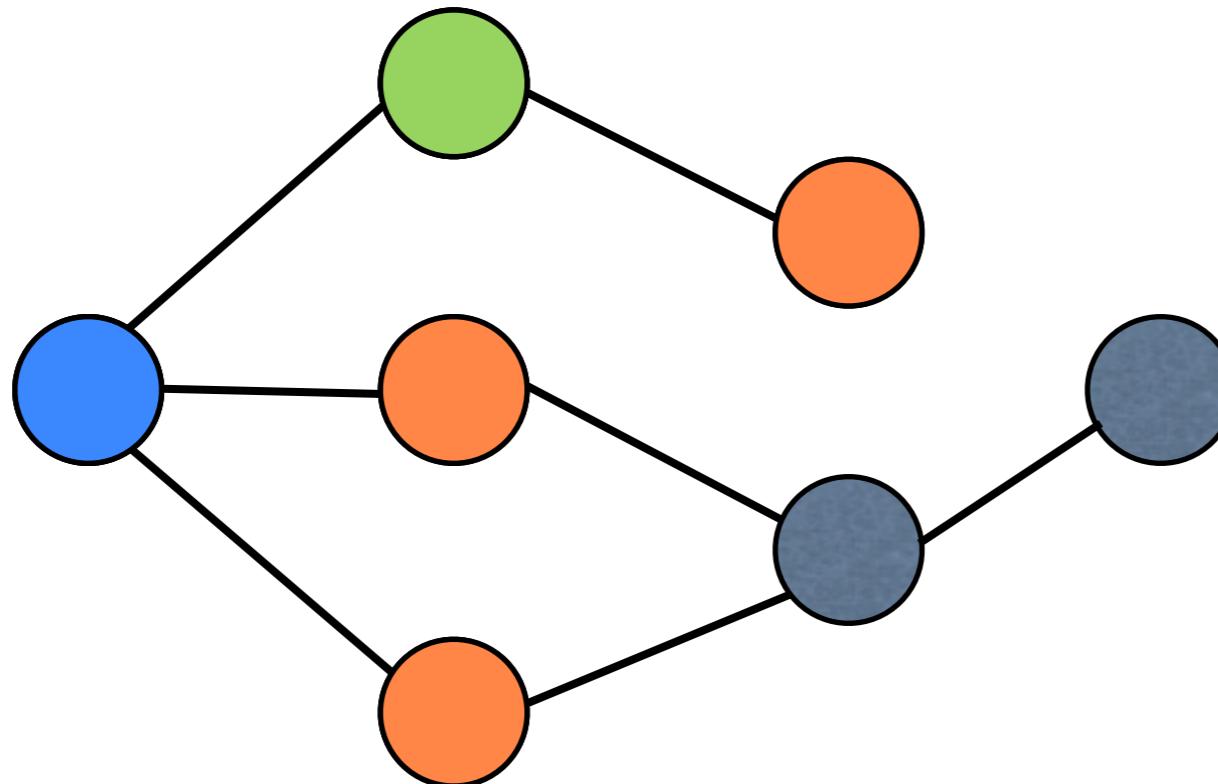
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



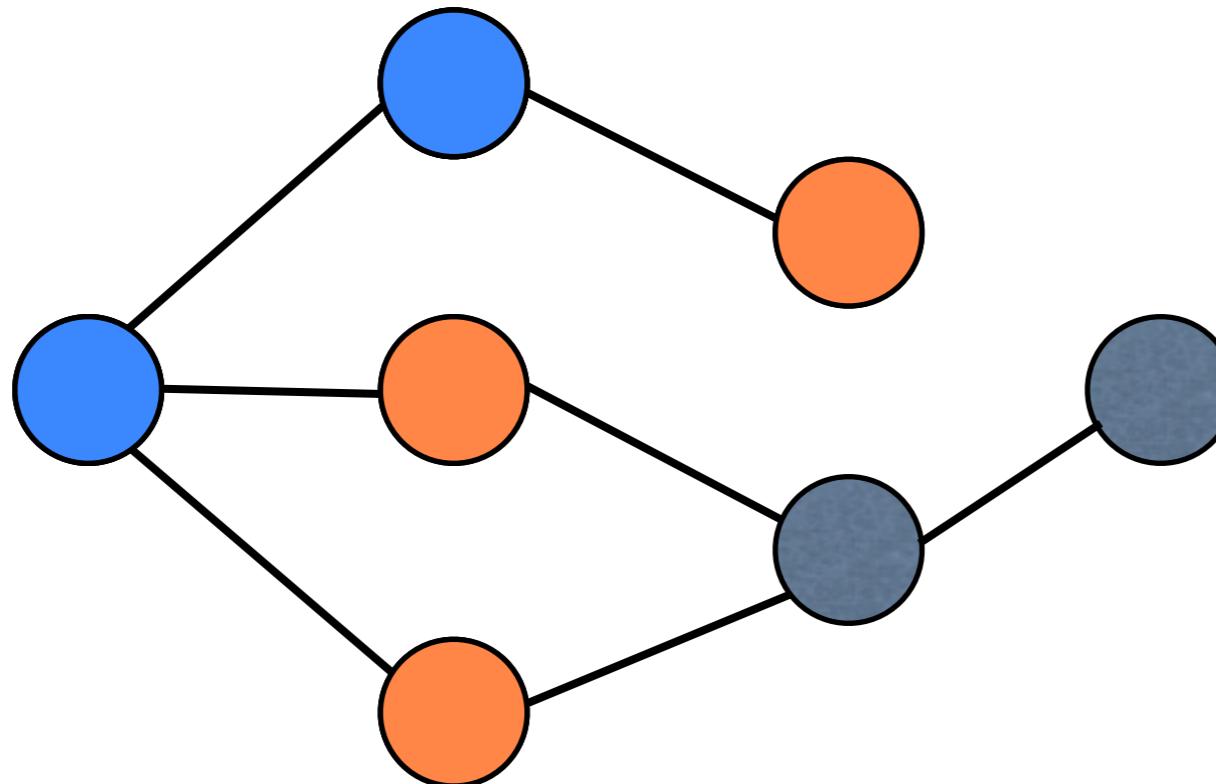
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



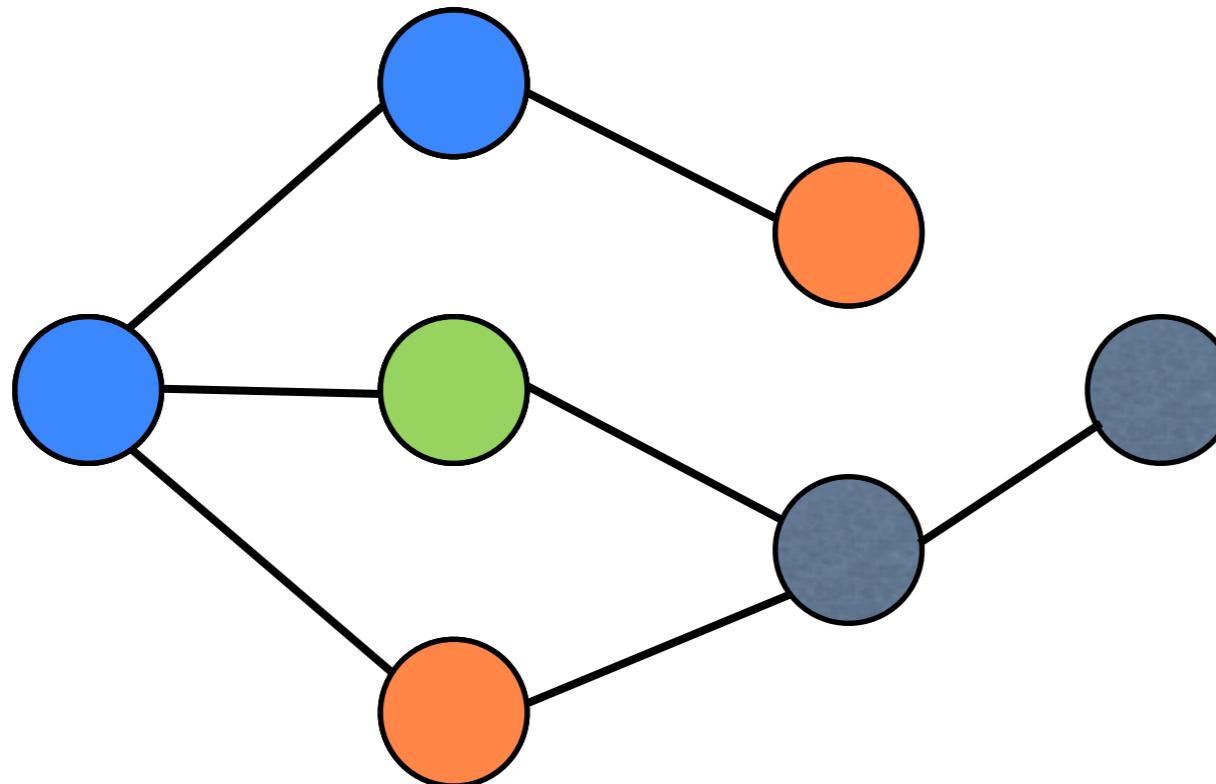
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



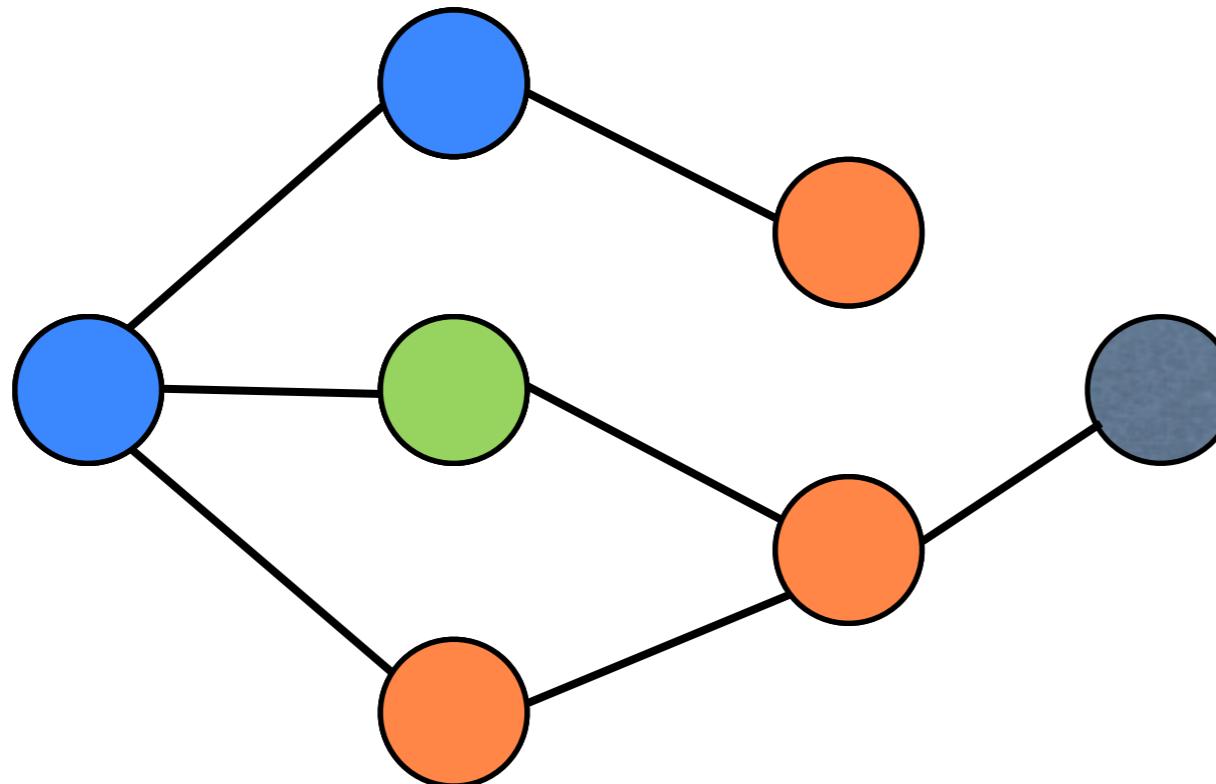
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



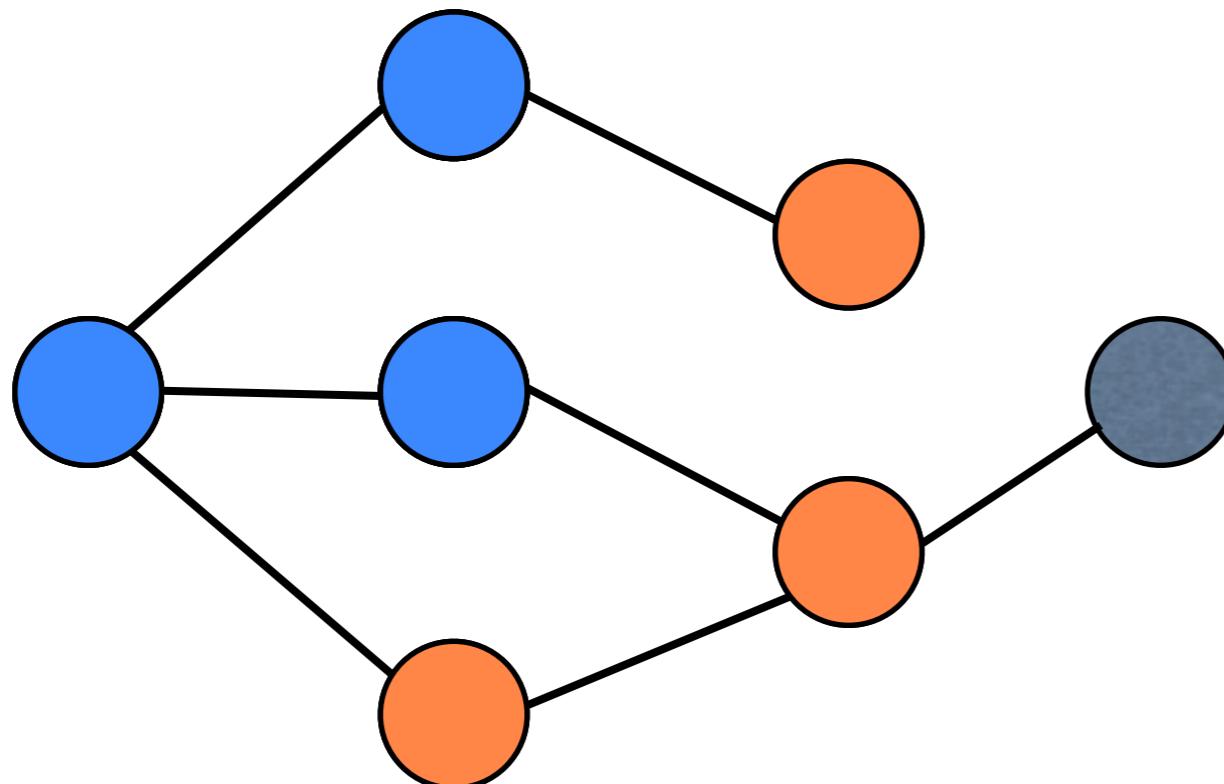
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



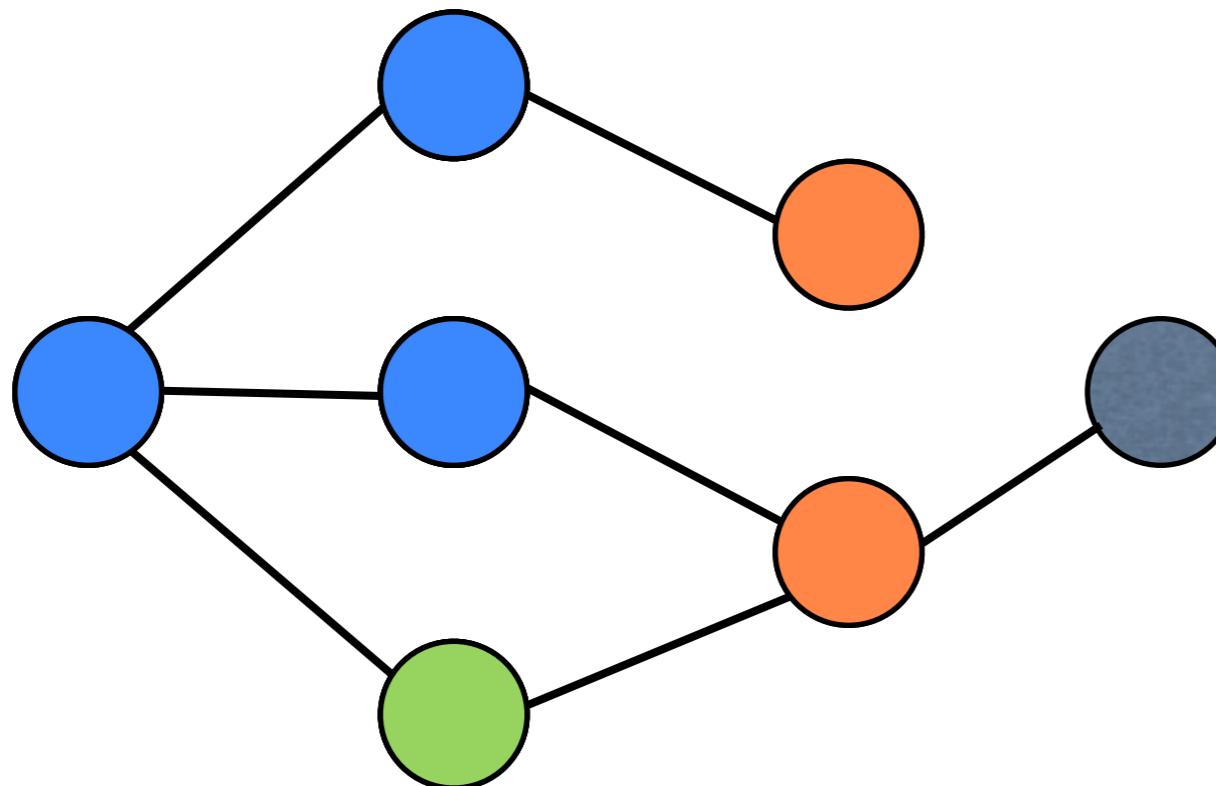
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



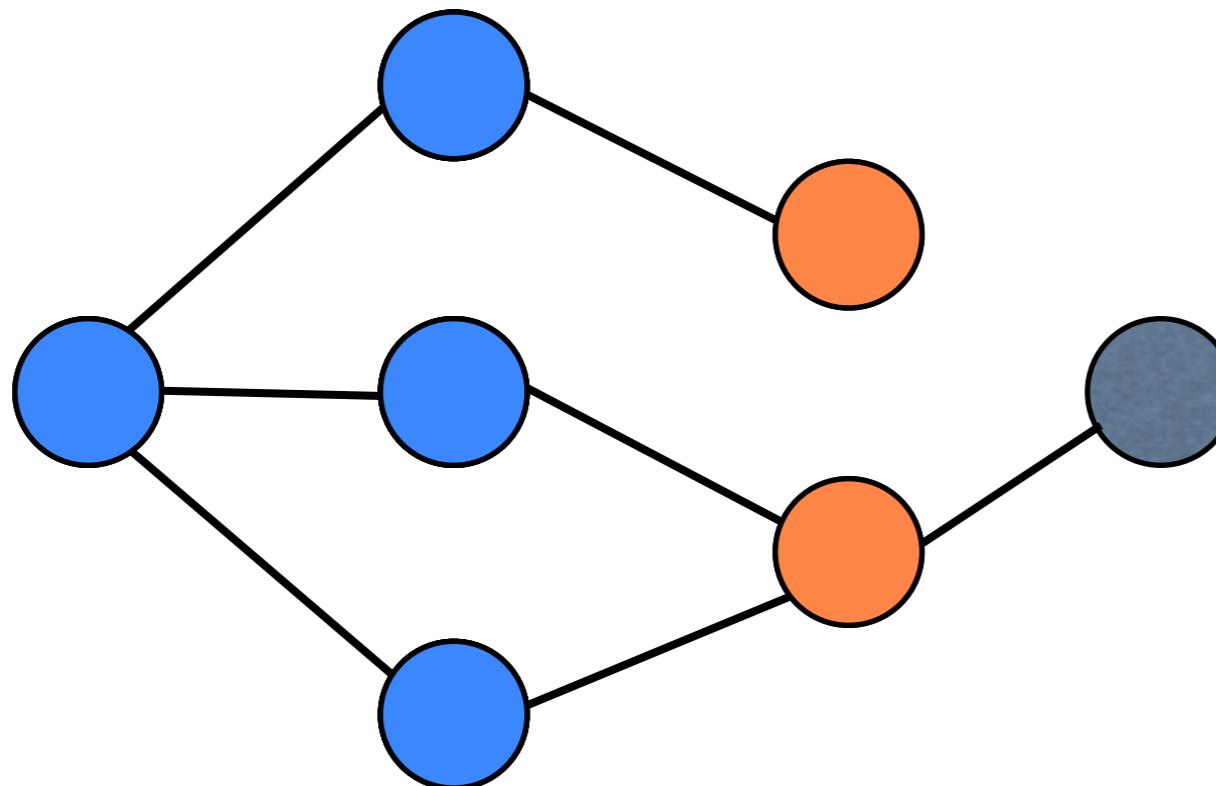
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



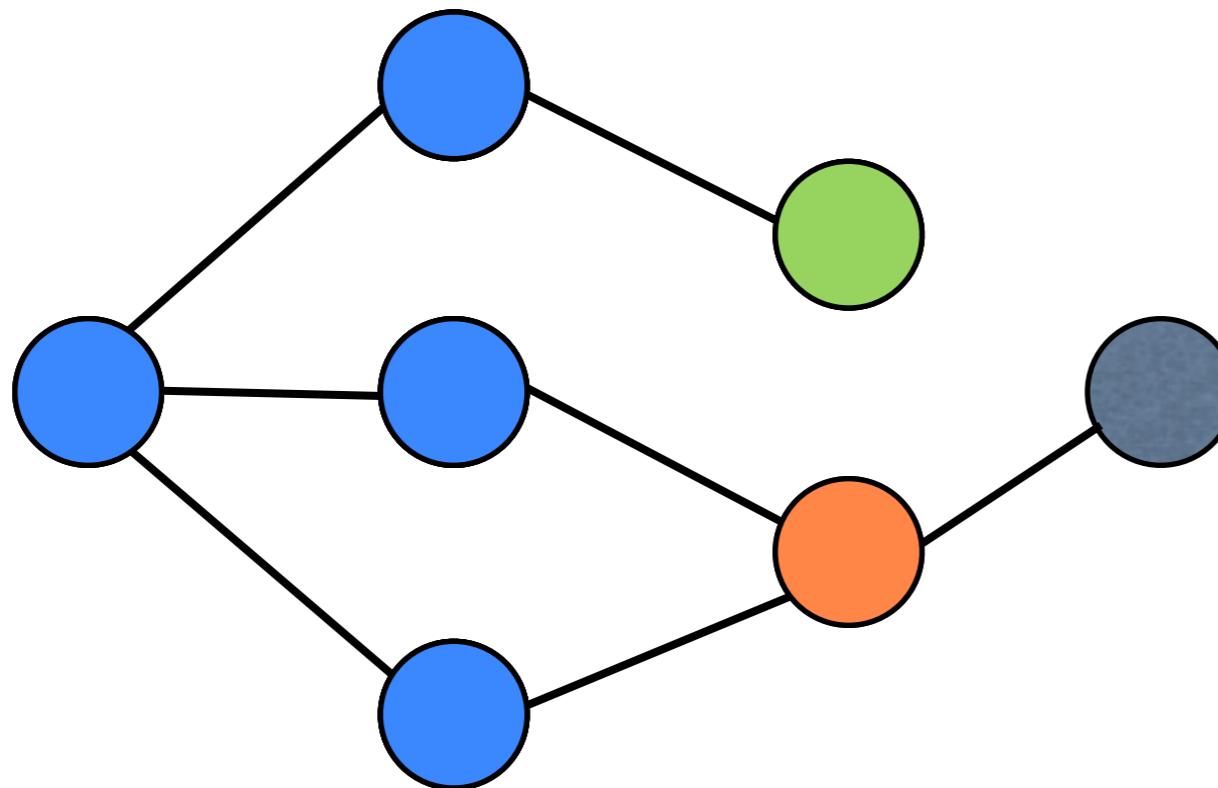
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



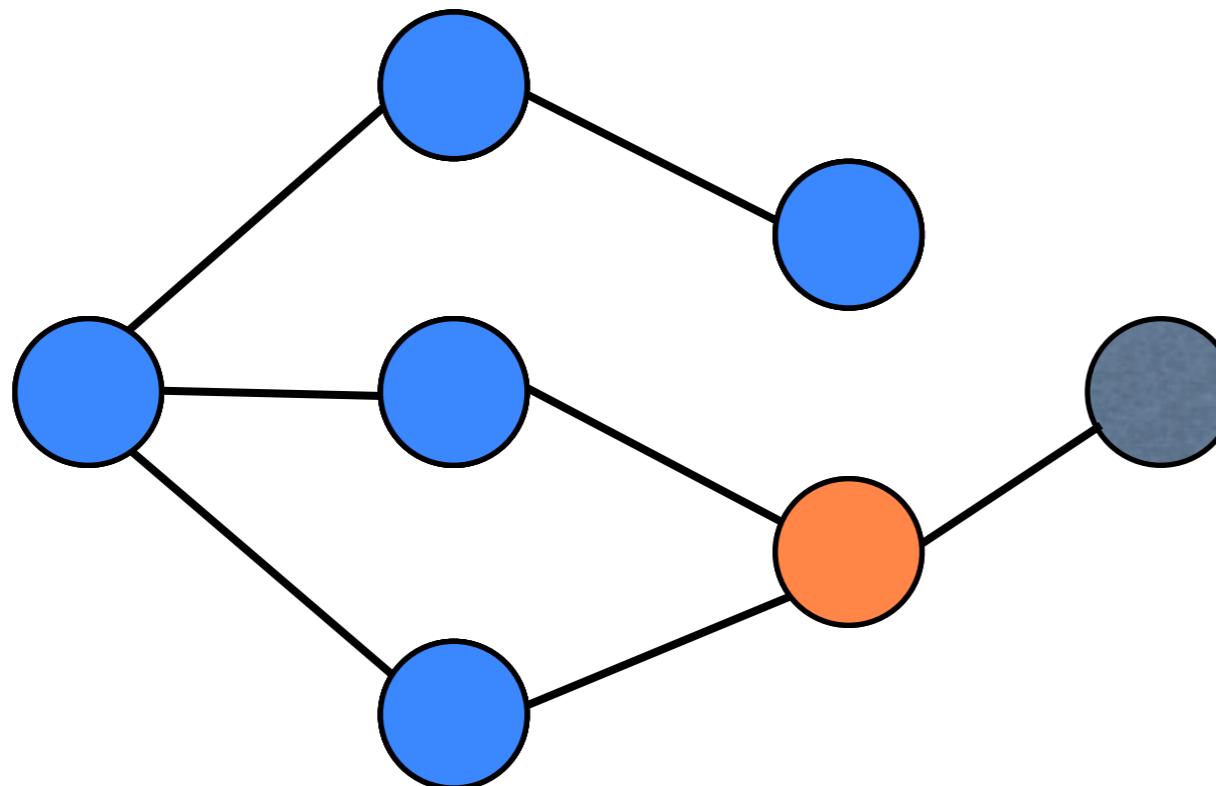
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



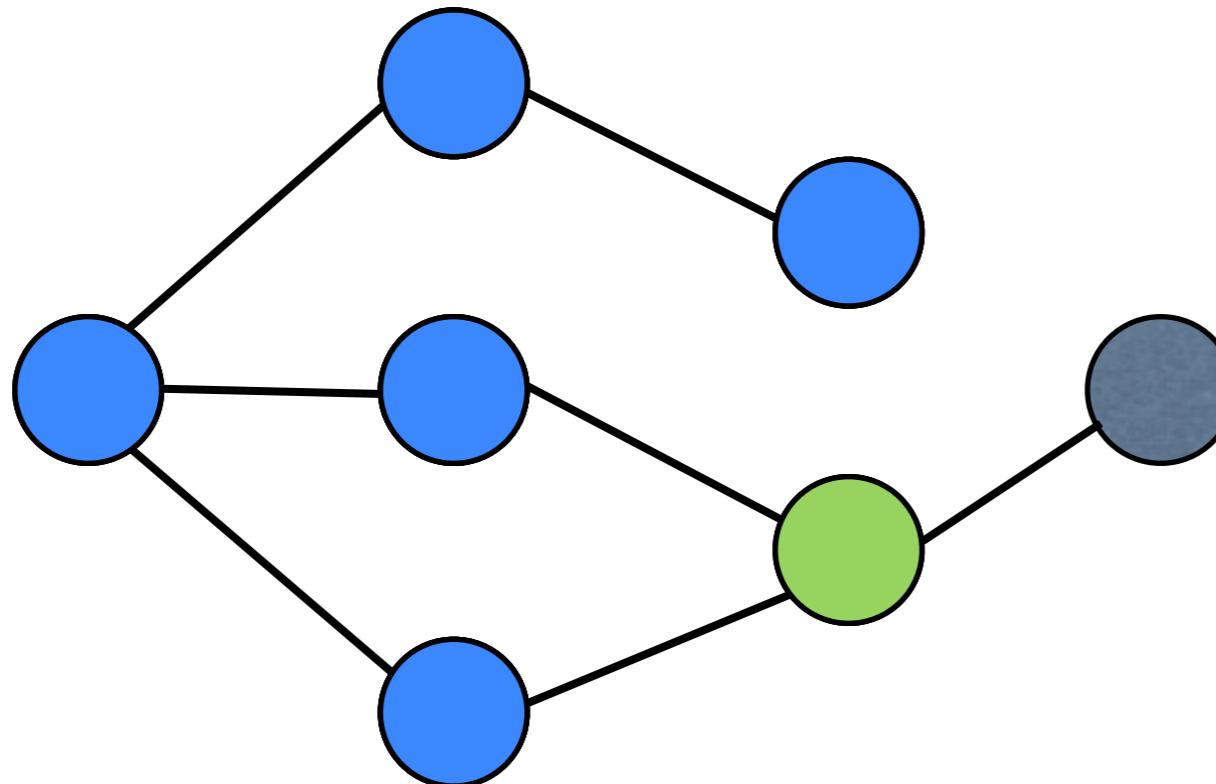
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



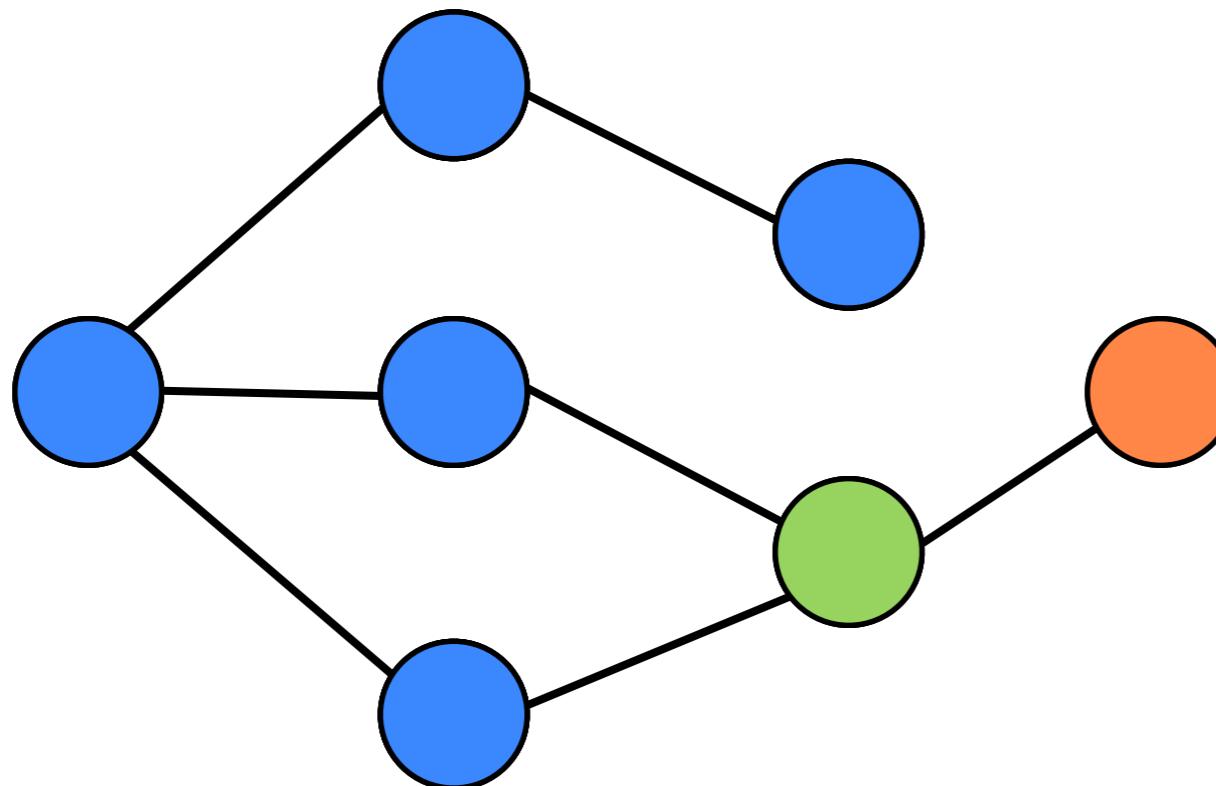
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



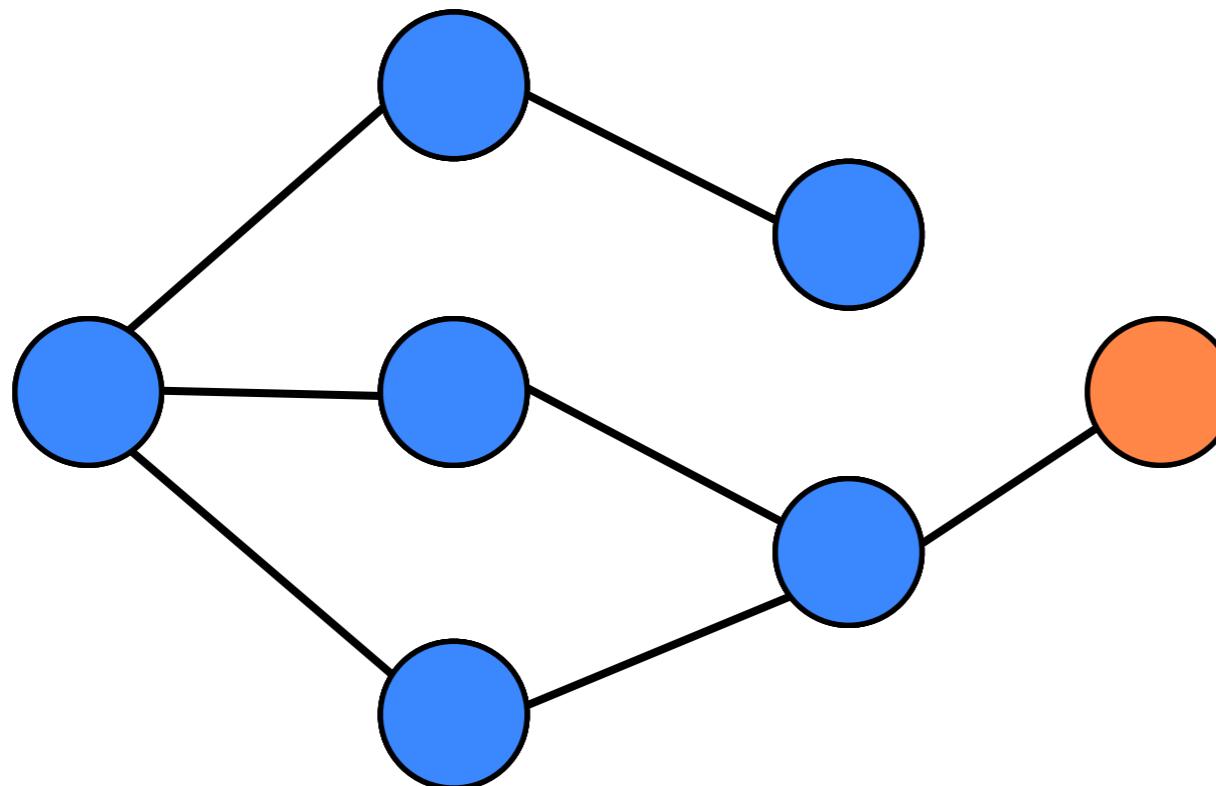
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



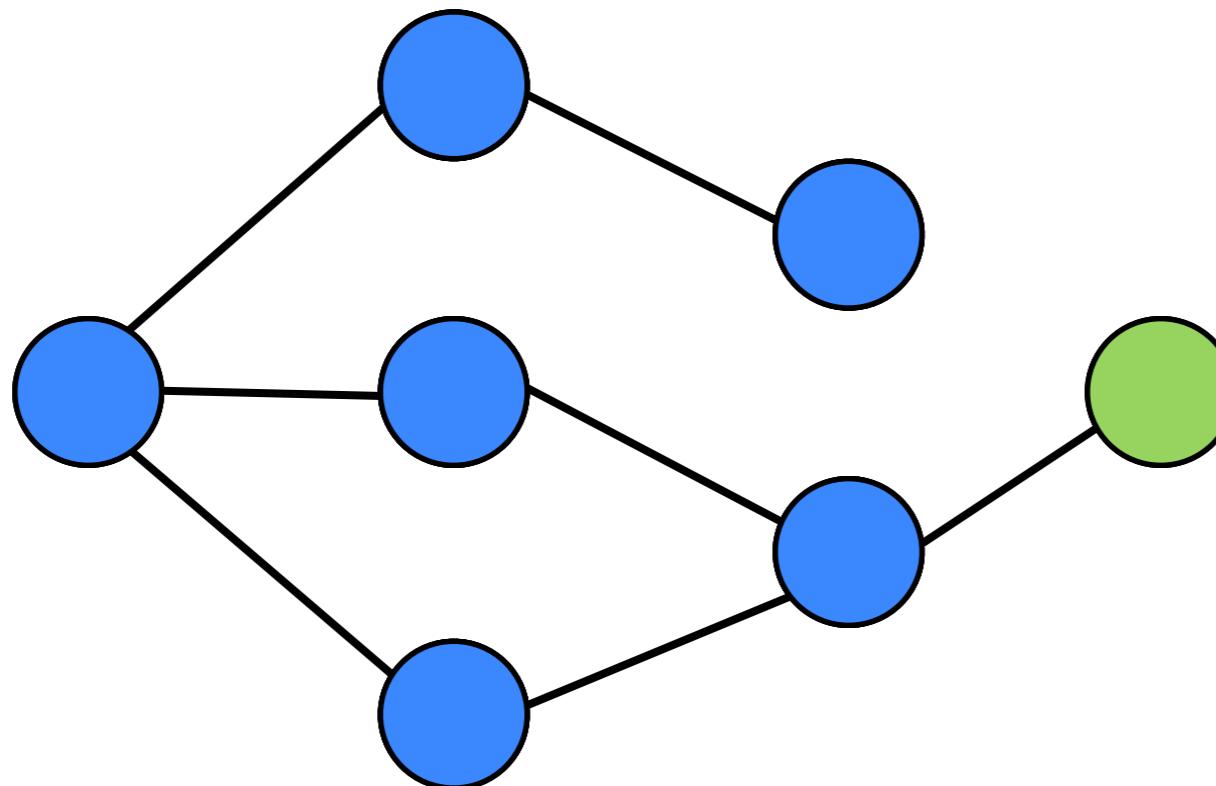
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



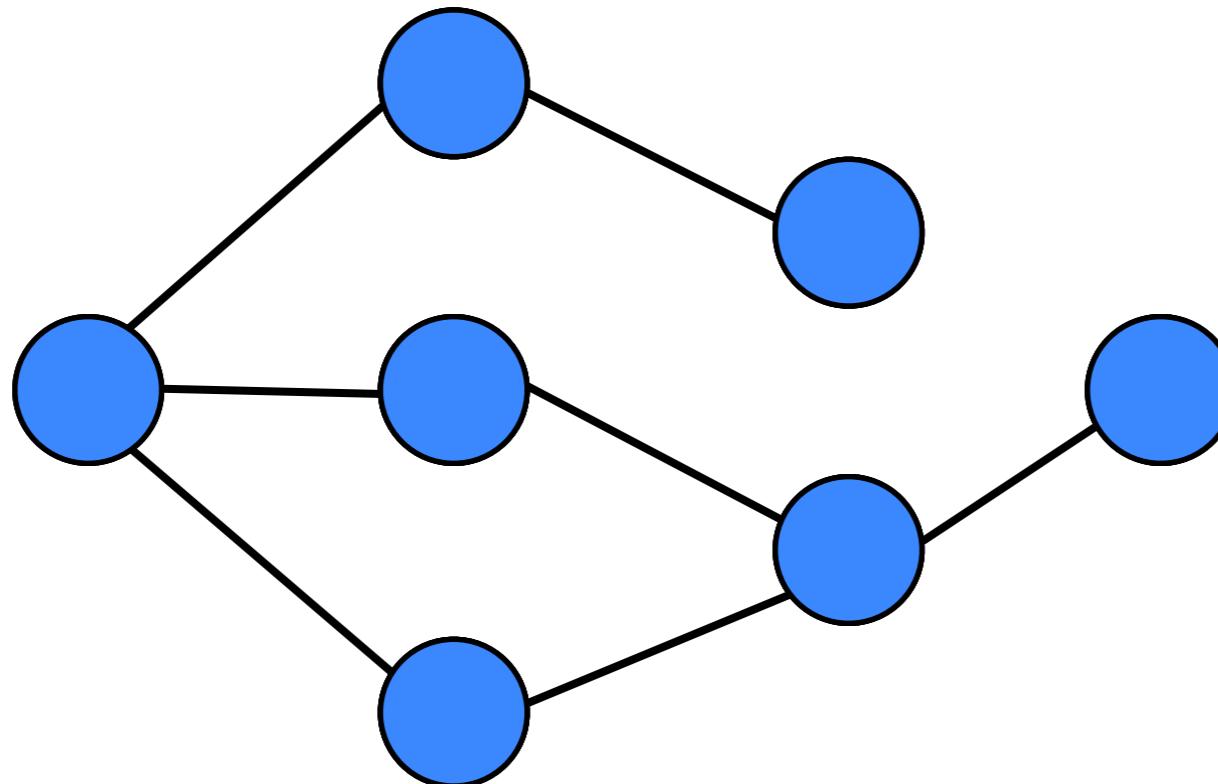
Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



Algoritmos de busca em grafos - Busca em largura

- Expande todos os vizinhos de um vértice primeiro:



Implementação: Busca em largura

```
#include <list> // std::list
#include <utility> // pair<> e make_pair()
using namespace std;

// Declarando
const int MAX_VERTICES = 100;
int grafo[MAX_VERTICES][MAX_VERTICES];
bool visitado[MAX_VERTICES];

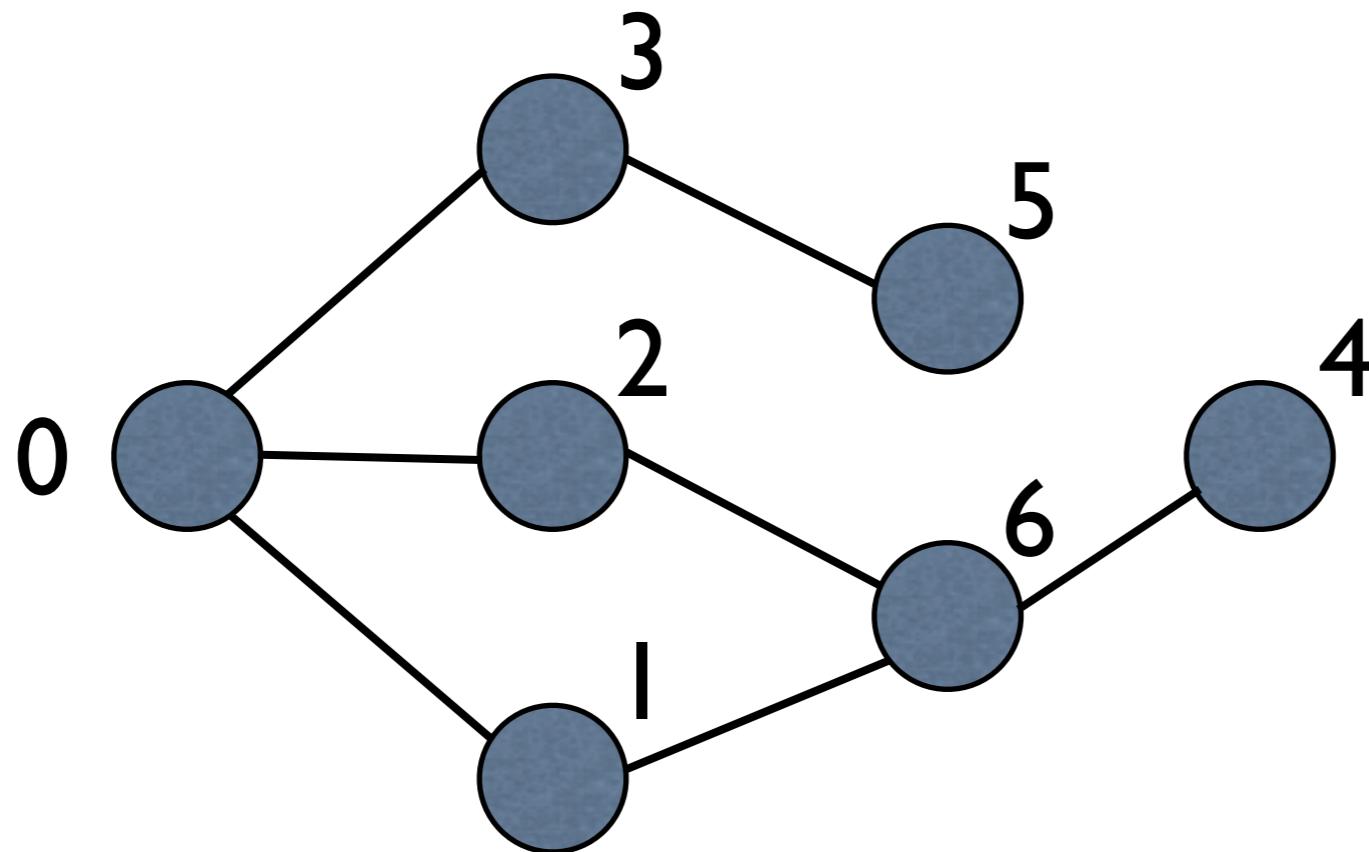
void bfs(int origem, int num_vertices) {
    for (int i = 0; i < num_vertices; ++i) {
        visitado[i] = false;
    }
    // Fila que guarda a ordem de visita dos vértices
    list<int> fila;
    /*...*/
```

Implementação: Busca em largura

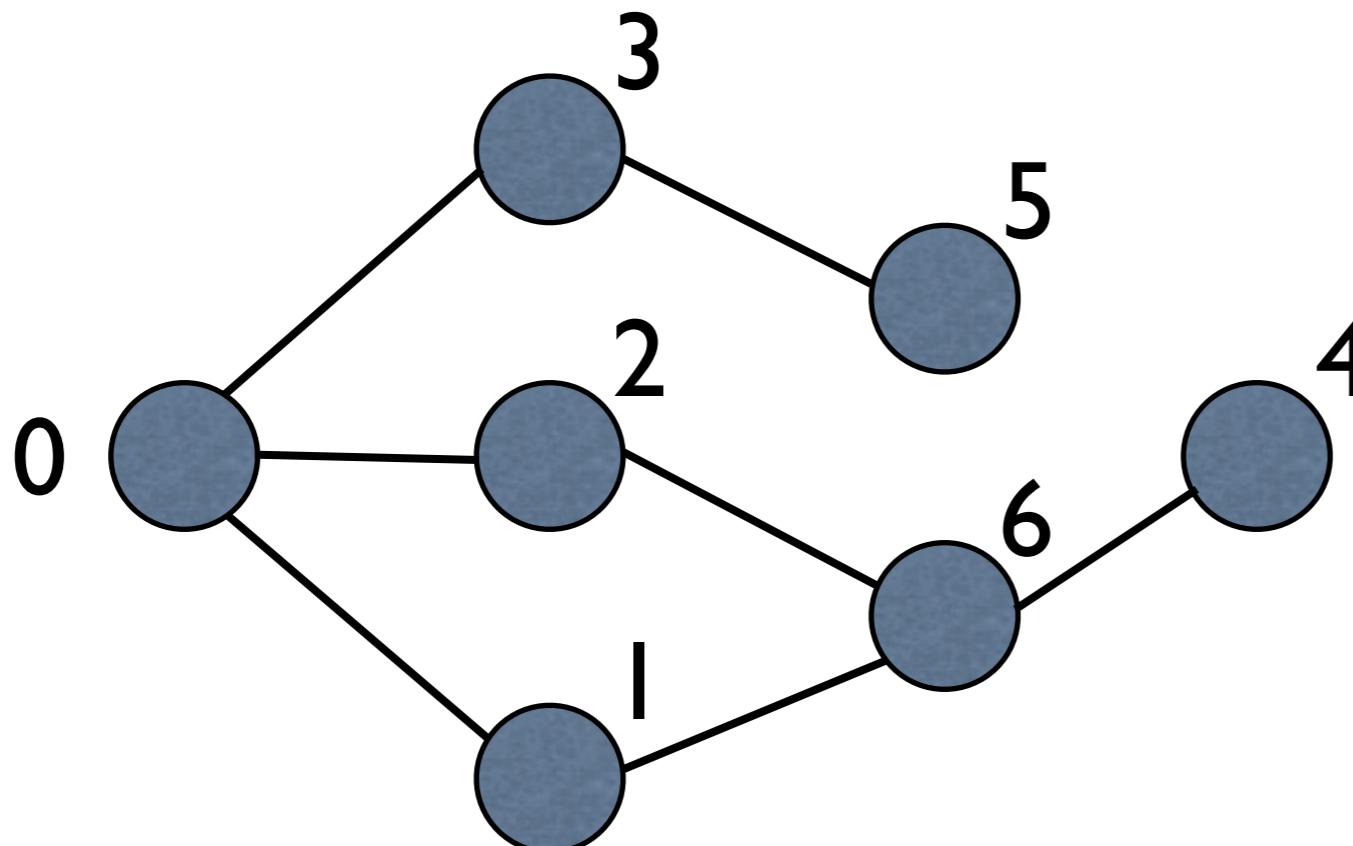
```
void bfs(int origem, int num_vertices) {
    for (int i = 0; i < num_vertices; ++i) {
        visitado[i] = false;
    }
    // Fila que guarda a ordem de visita dos vértices
    list<int> fila;
    // Adiciona a origem na fila
    fila.push_back(origem);
    visitado[origem] = true;

    int atual; // vértice atual que estamos visitando
    while (!fila.empty()) {
        atual = fila.front(); // pega o 1o vértice
        fila.pop_front(); // remove ele da fila
        for (int i = 0; i < num_vertices; ++i) {
            if (grafo[atual][i] && !visitado[i]) {
                lista.push_back(i);
                visited[i] = true;
            }
        }
    }
}
```

Algoritmos de busca em grafos - Busca em largura

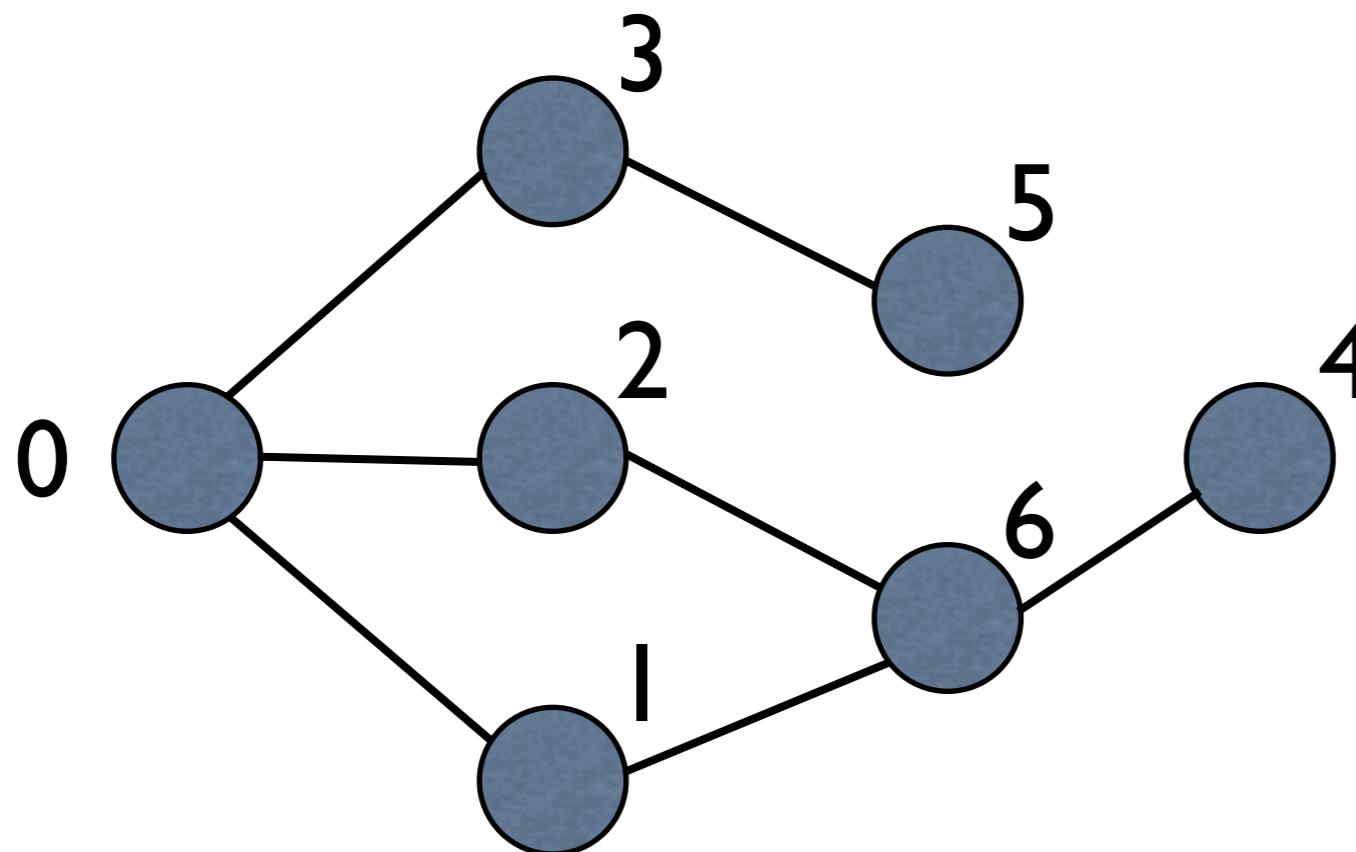


Algoritmos de busca em grafos - Busca em largura



	0	1	2	3	4	5	6
0	-	1	1	1	0	0	0
1	1	-	0	0	0	0	1
2	1	0	-	0	0	0	1
3	1	0	0	-	0	1	0
4	0	0	0	0	-	0	1
5	0	0	0	1	0	-	0
6	0	1	1	0	1	0	-

Algoritmos de busca em grafos - Busca em largura

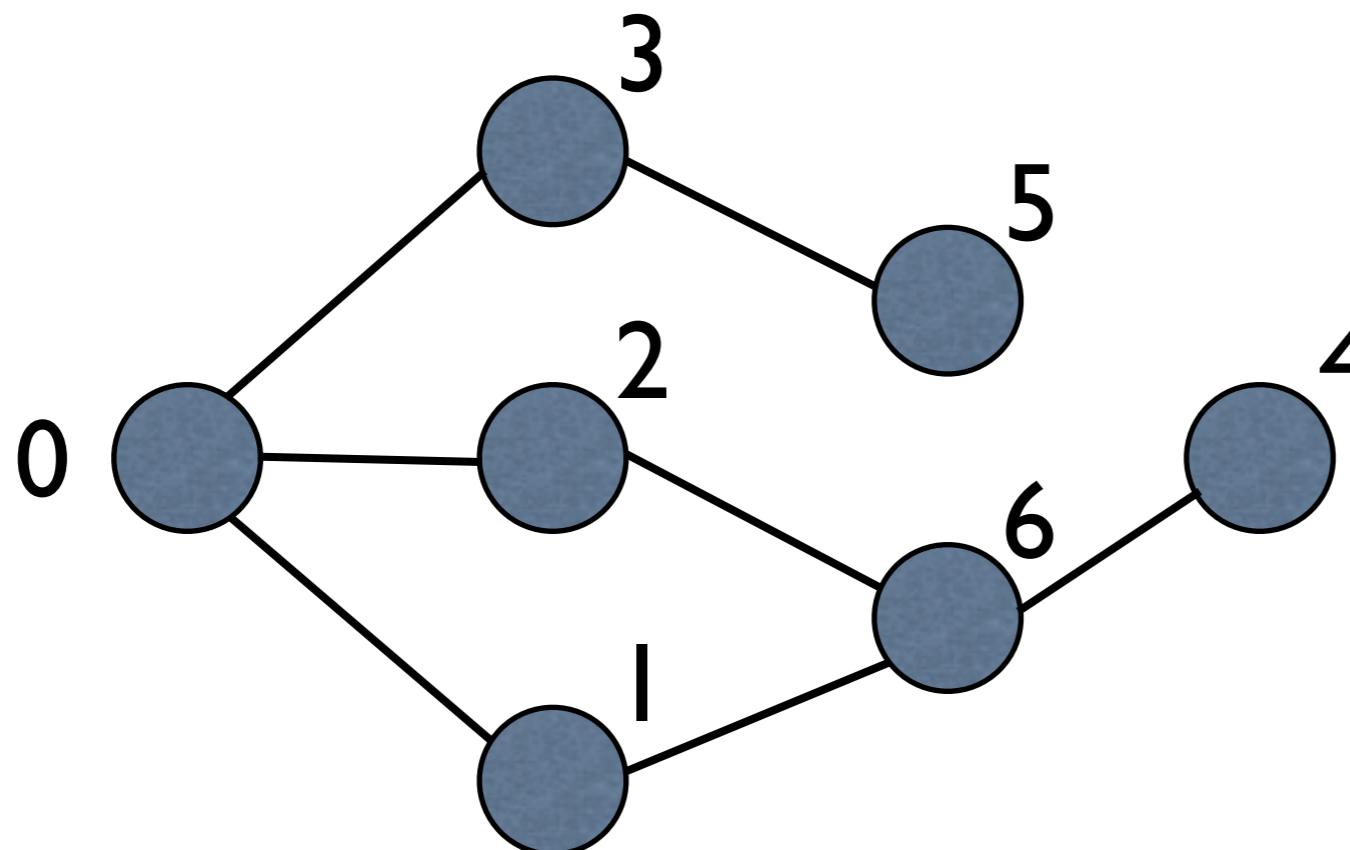


VISITADO							
0	1	2	3	4	5	6	
F	F	F	F	F	F	F	F

	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 0



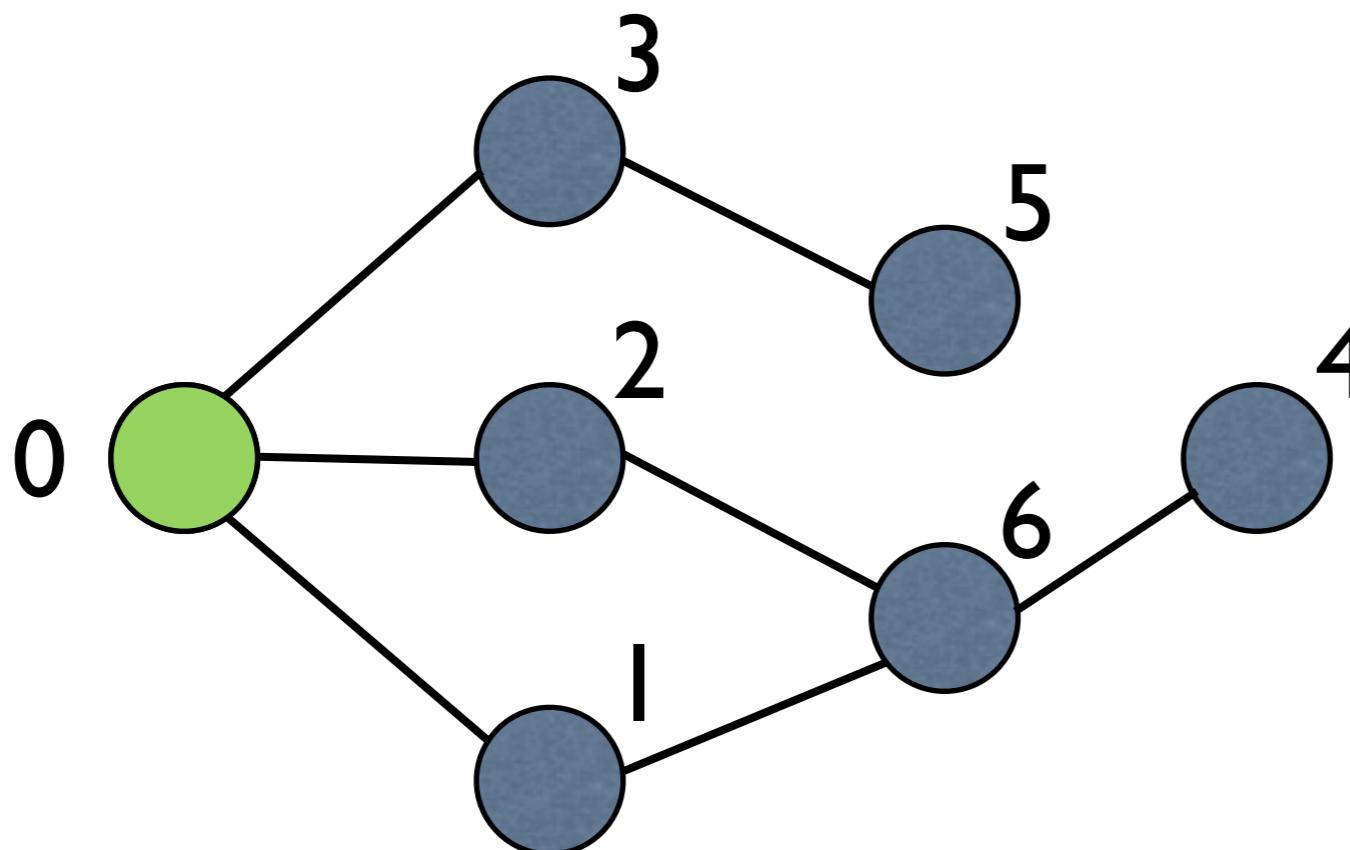
VISITADO							
0	1	2	3	4	5	6	
F	F	F	F	F	F	F	F

0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 0

VISITADO							
0	1	2	3	4	5	6	
F	F	F	F	F	F	F	F

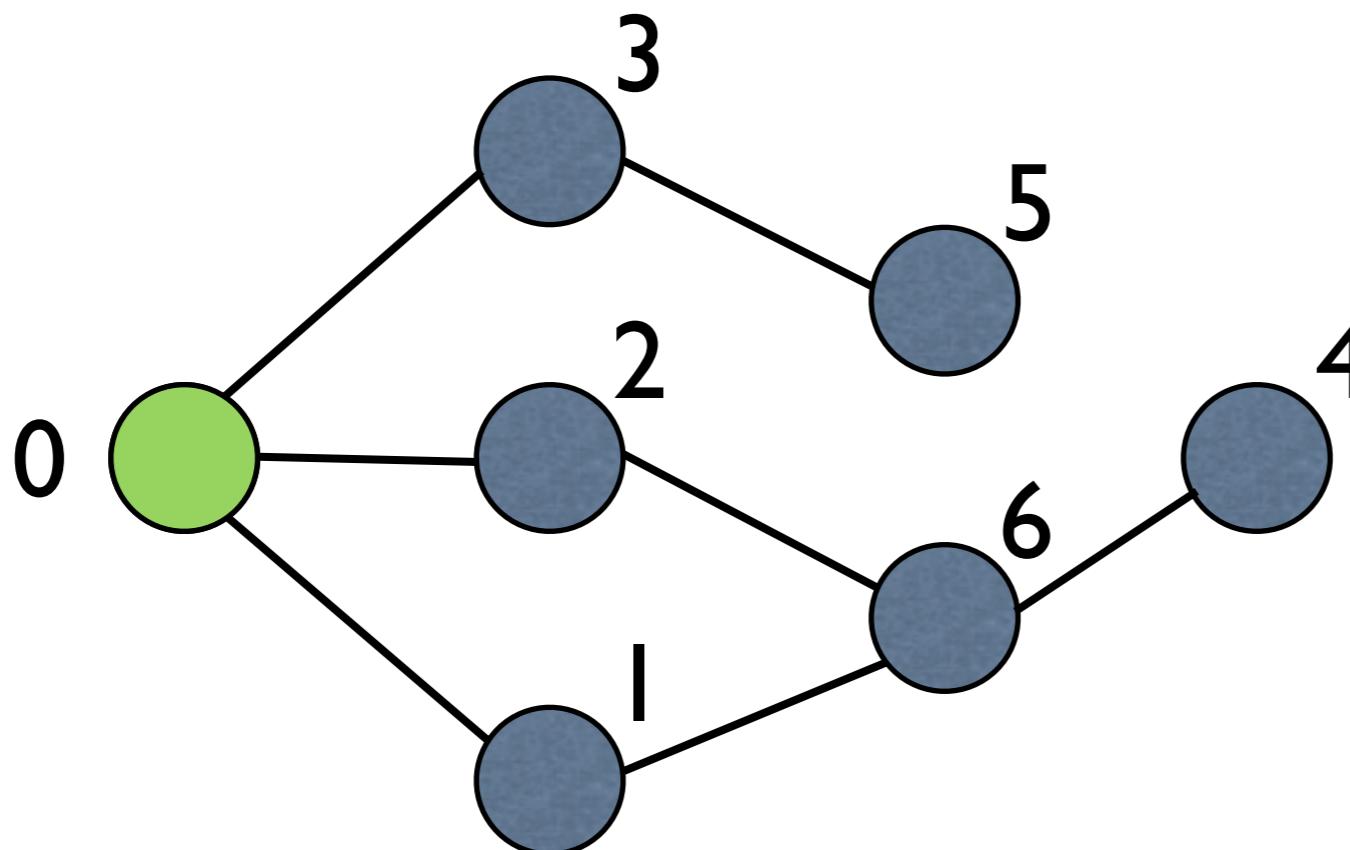


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 0

VISITADO							
0	1	2	3	4	5	6	
T	F	F	F	F	F	F	F

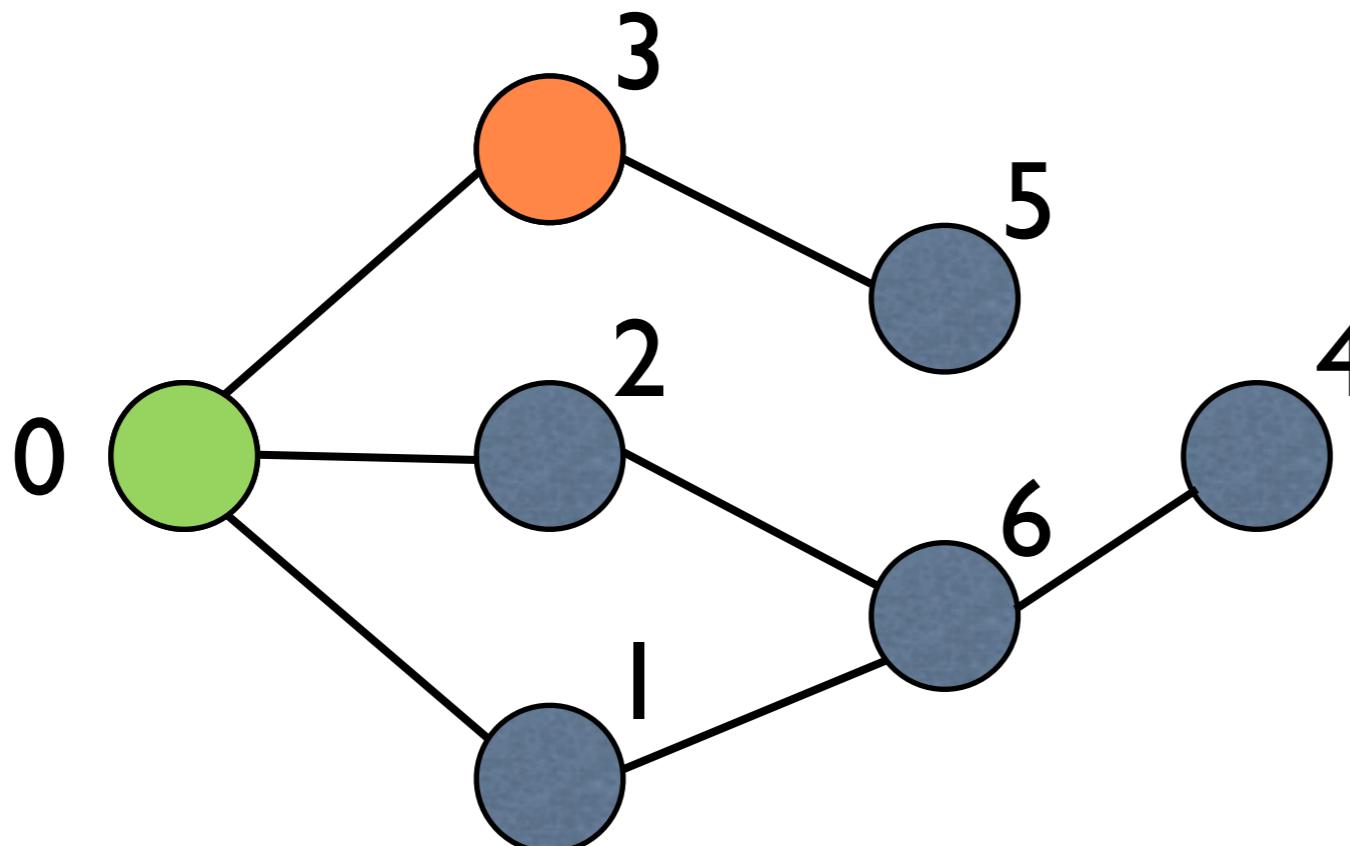


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 0

VISITADO							
0	1	2	3	4	5	6	
T	F	F	F	F	F	F	F

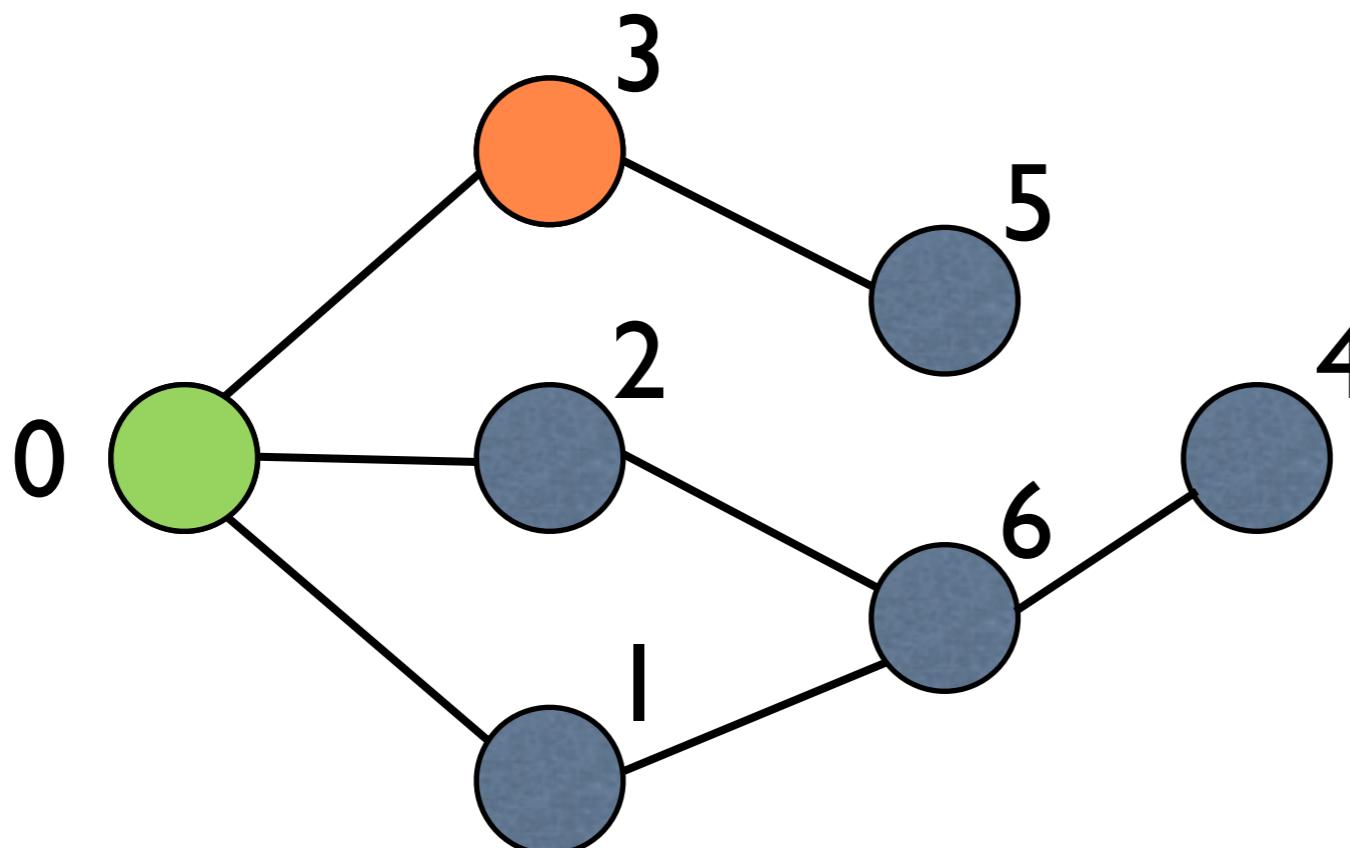


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3

VISITADO							
0	1	2	3	4	5	6	
T	F	F	F	F	F	F	F

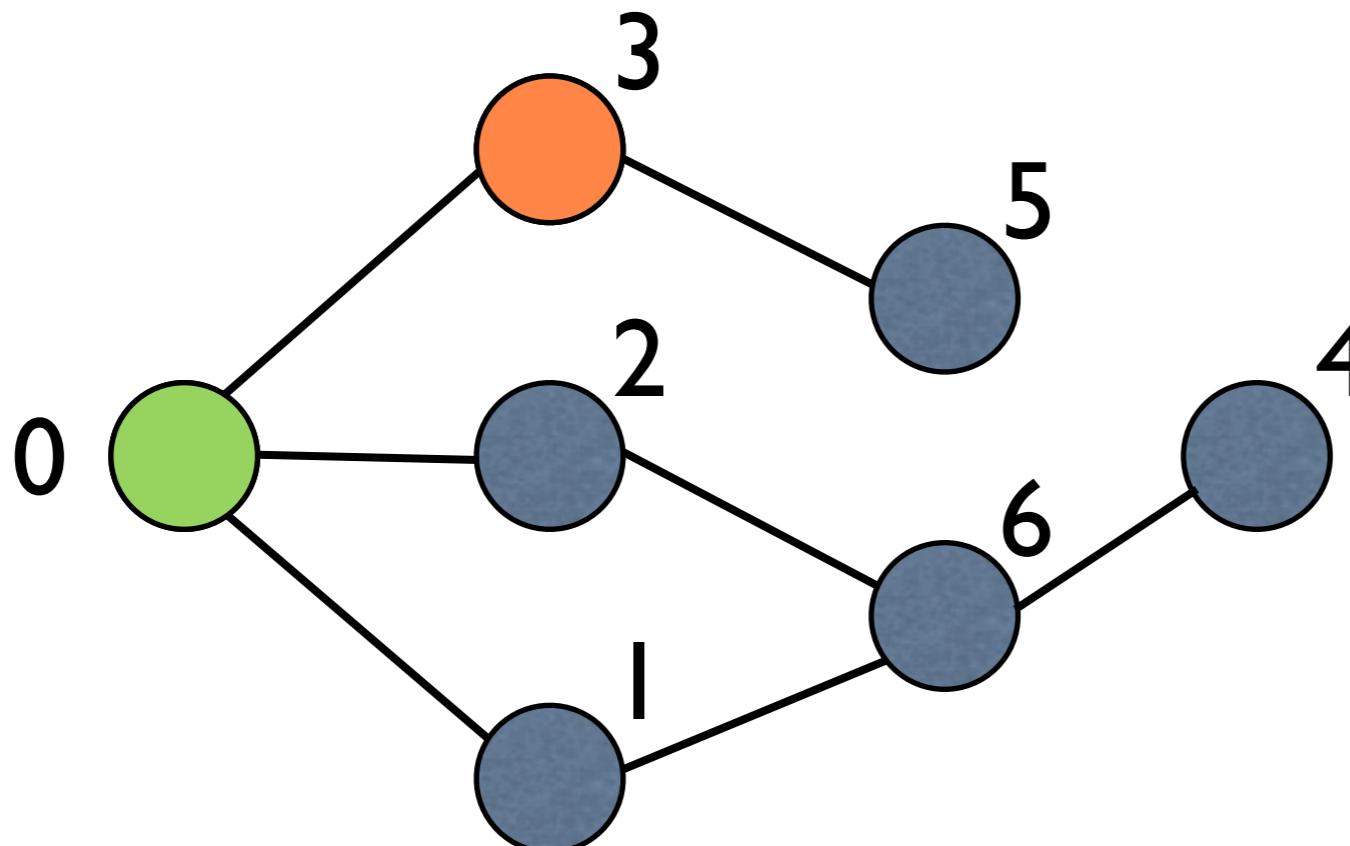


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3

VISITADO							
0	1	2	3	4	5	6	
T	F	F	T	F	F	F	F

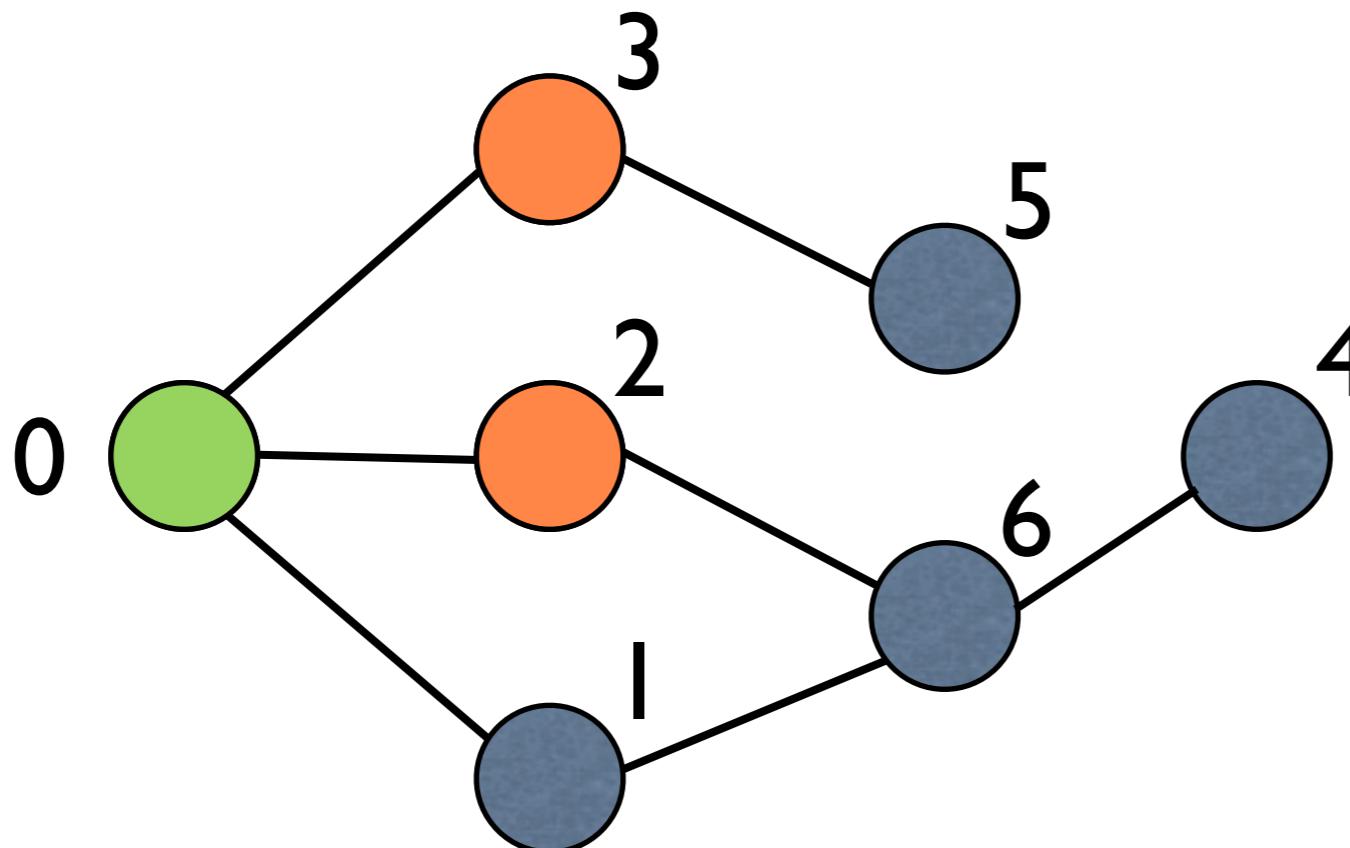


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3

VISITADO							
0	1	2	3	4	5	6	
T	F	F	T	F	F	F	F

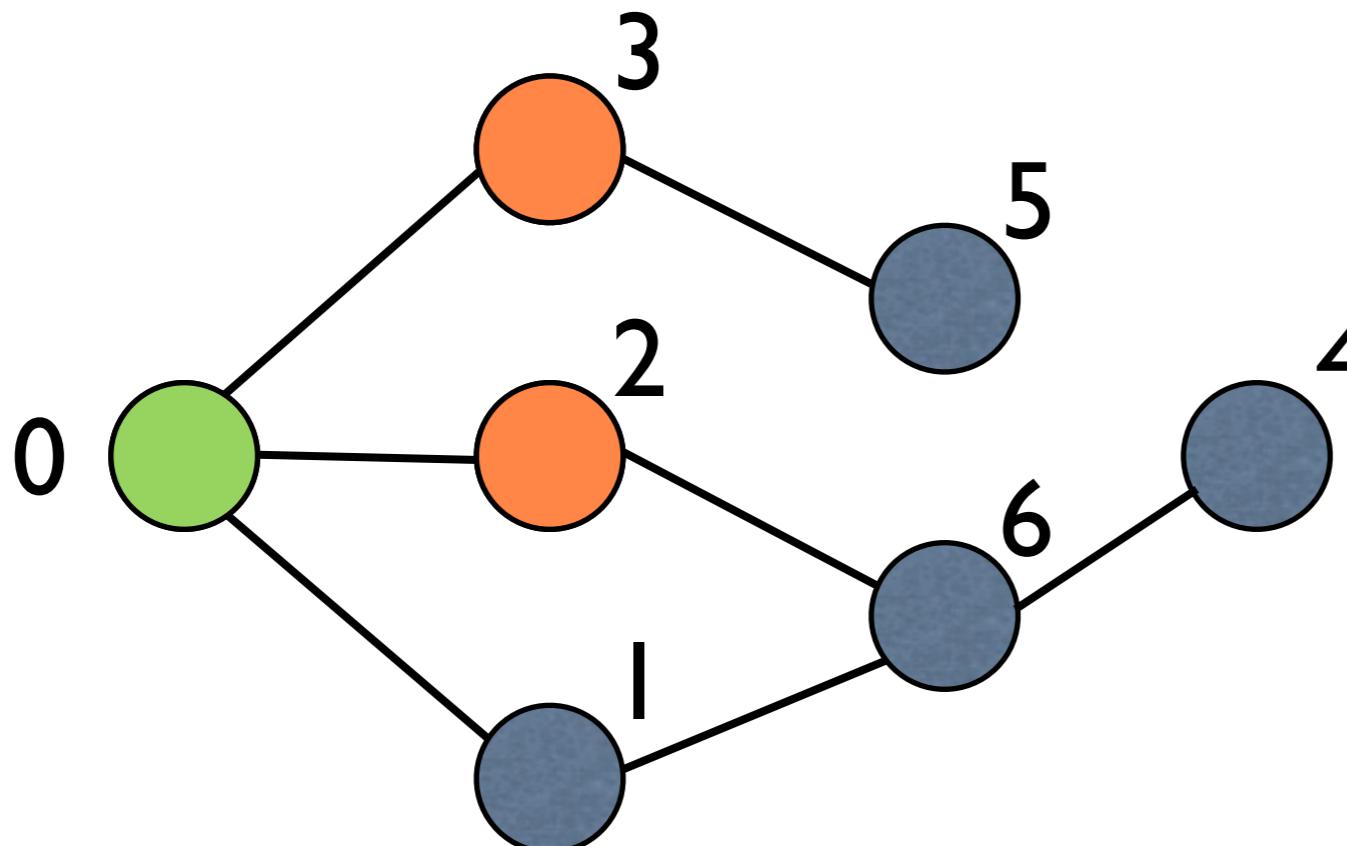


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2

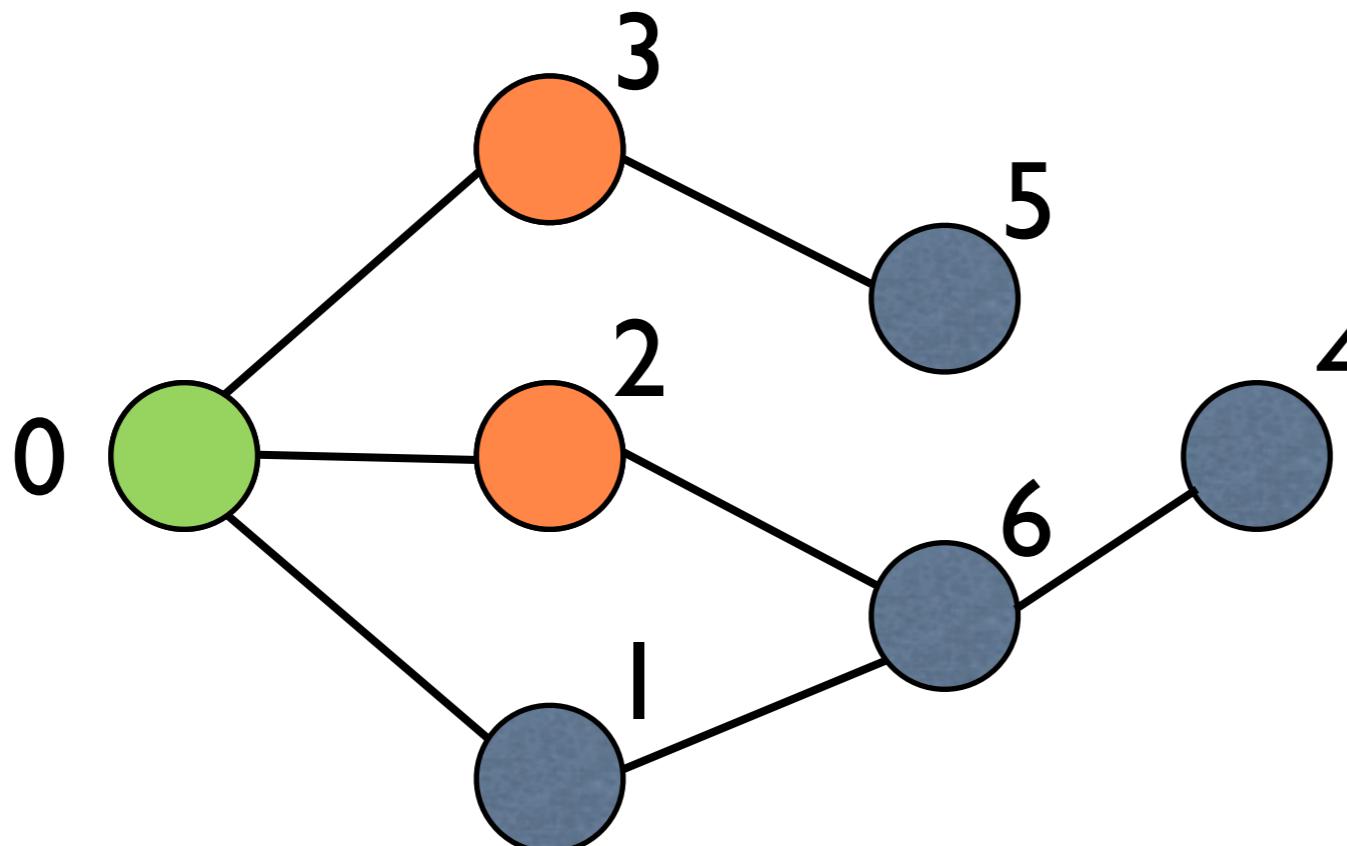
VISITADO							
0	1	2	3	4	5	6	
T	F	F	T	F	F	F	F



Algoritmos de busca em grafos - Busca em largura

FILA: 3 2

VISITADO							
0	1	2	3	4	5	6	
T	F	T	T	F	F	F	

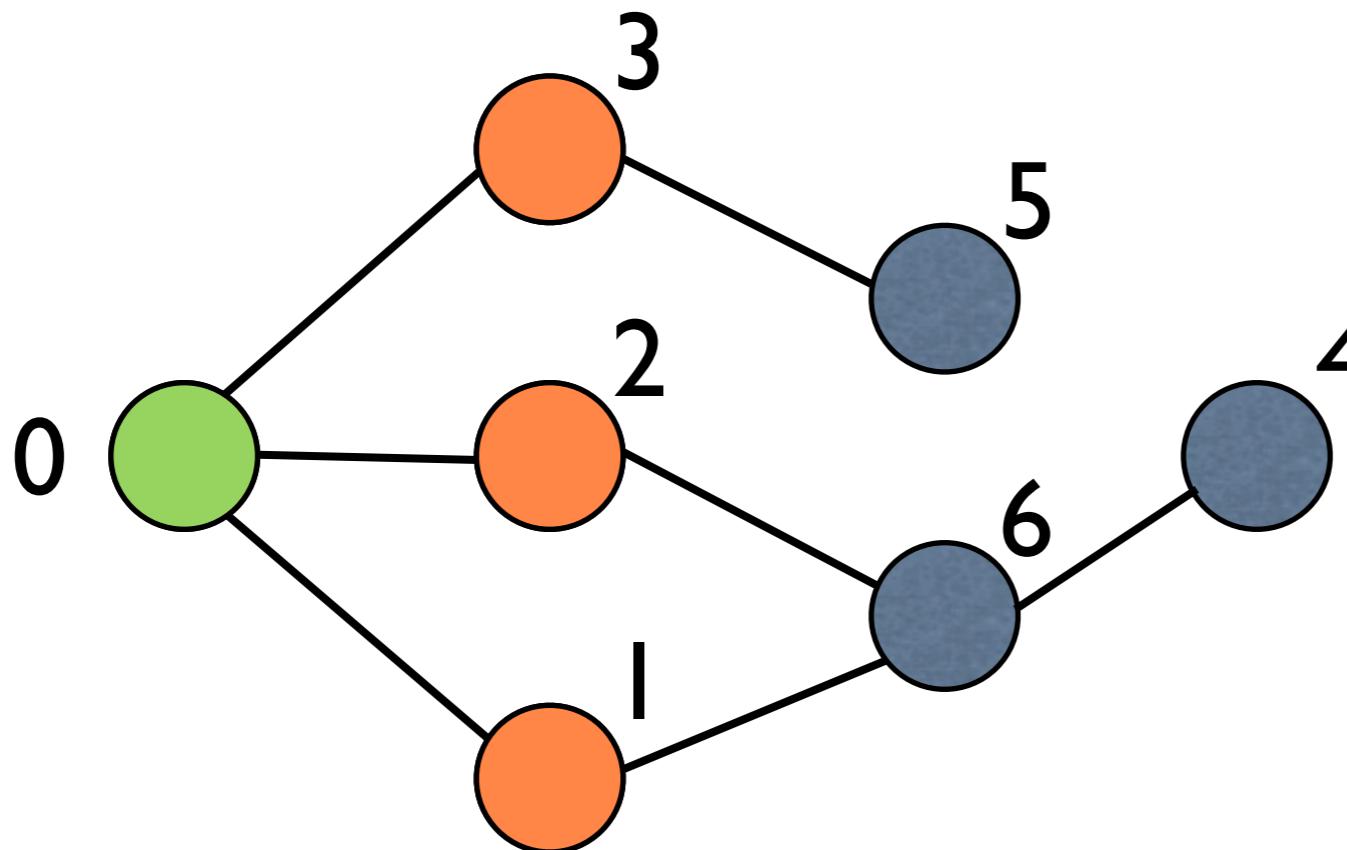


	0	1	2	3	4	5	6
0	-	1	1	1	0	0	0
1	1	-	0	0	0	0	1
2	1	0	-	0	0	0	1
3	1	0	0	-	0	1	0
4	0	0	0	0	-	0	1
5	0	0	0	1	0	-	0
6	0	1	1	0	1	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2

VISITADO							
0	1	2	3	4	5	6	
T	F	T	T	F	F	F	

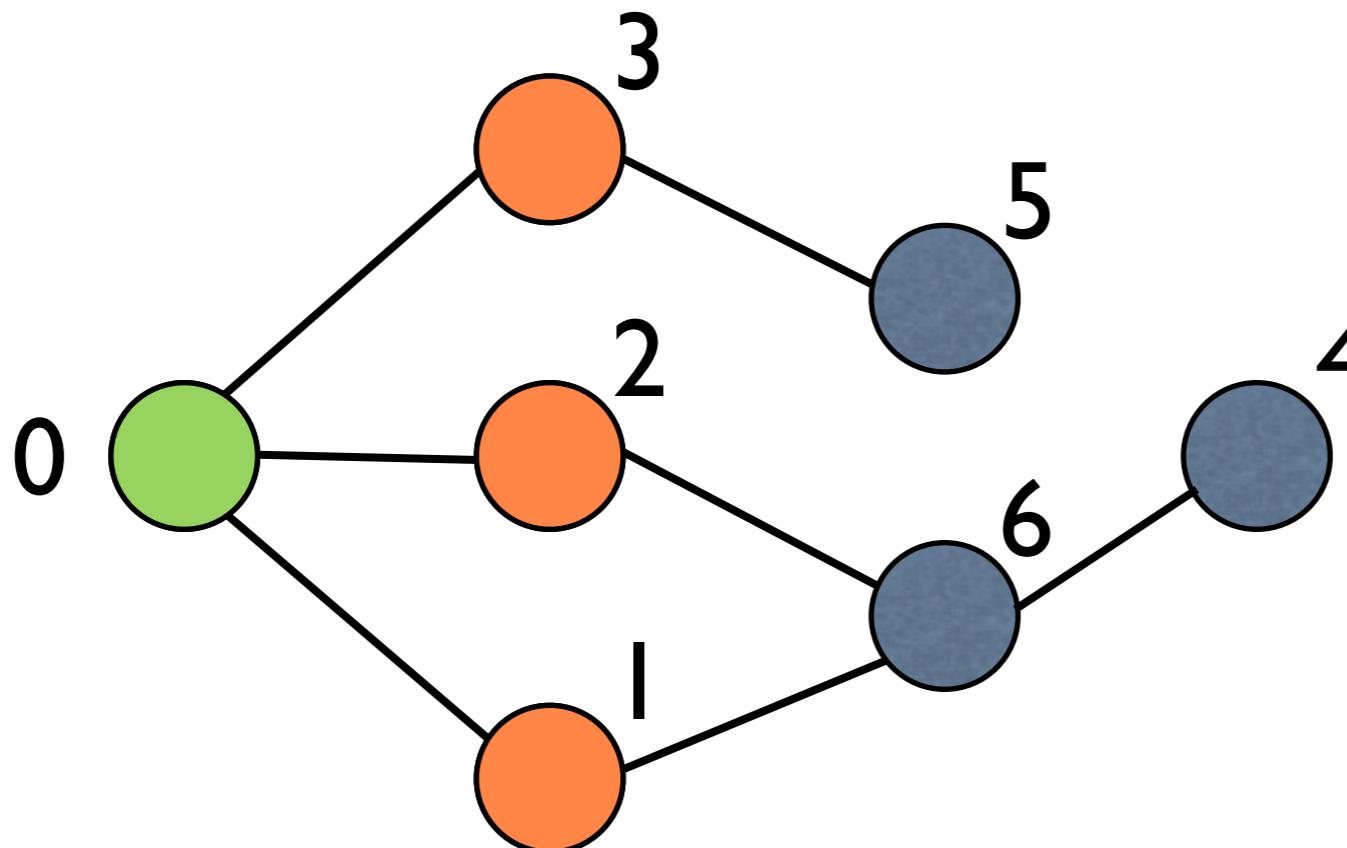


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2 1

VISITADO							
0	1	2	3	4	5	6	
T	F	T	T	F	F	F	

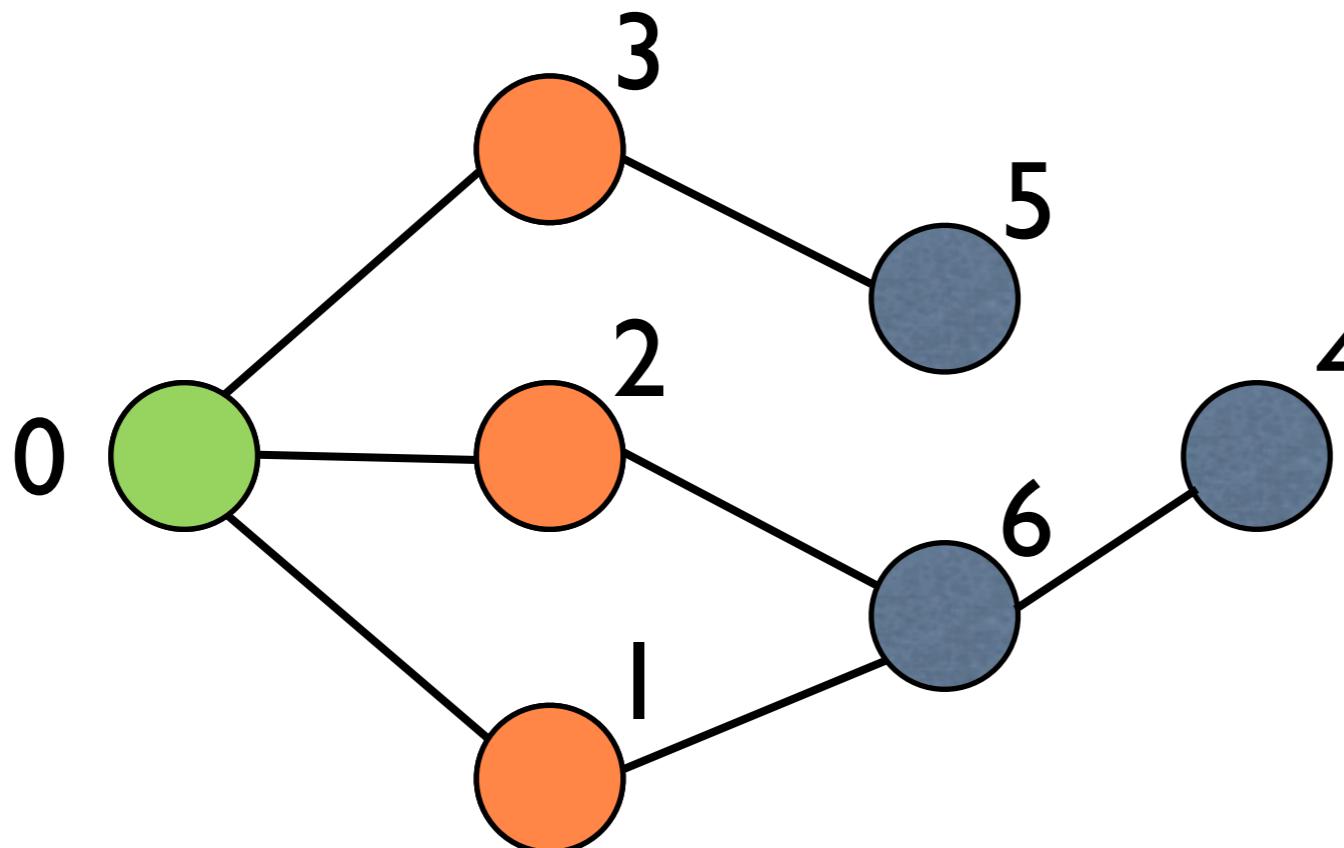


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2 1

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	

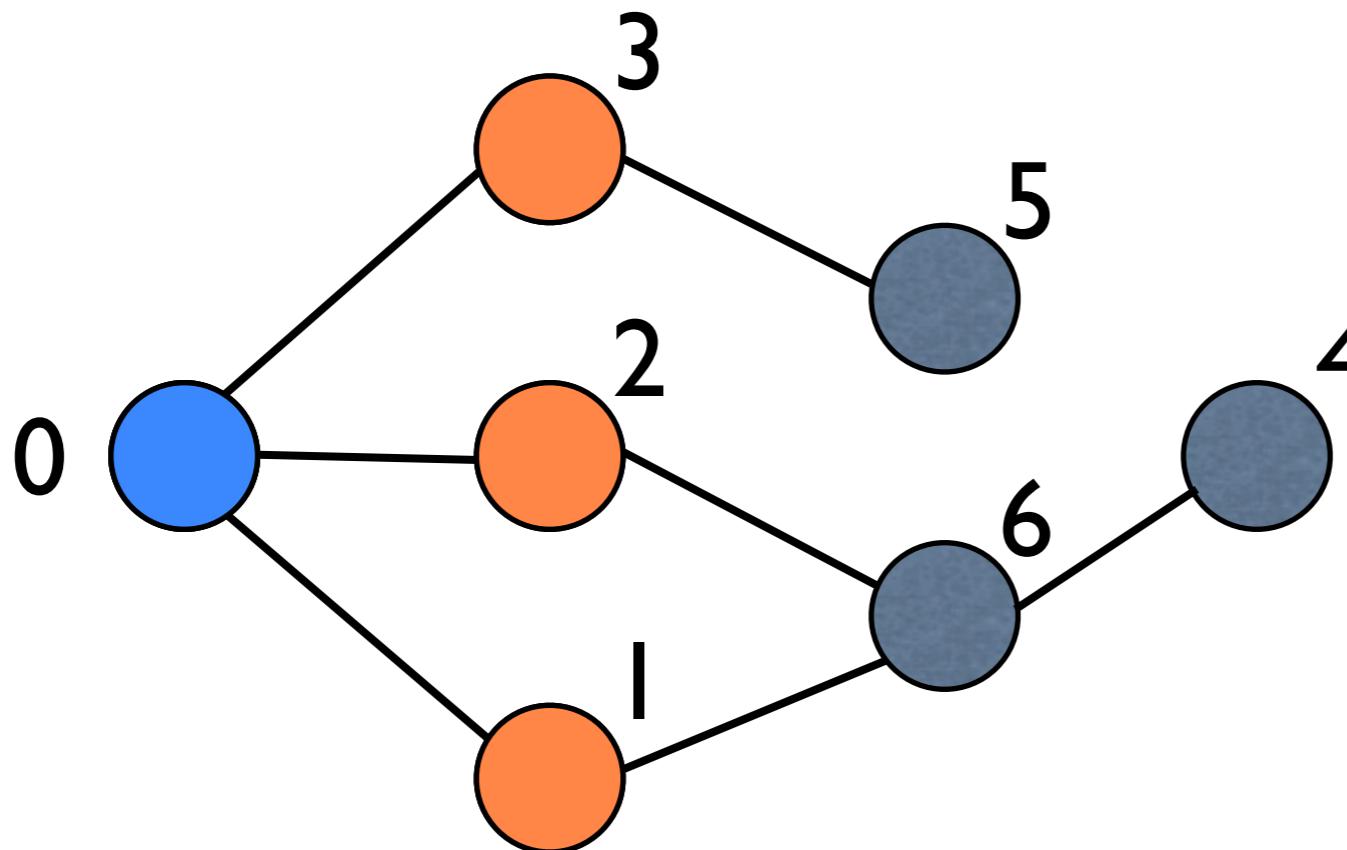


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2 1

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	

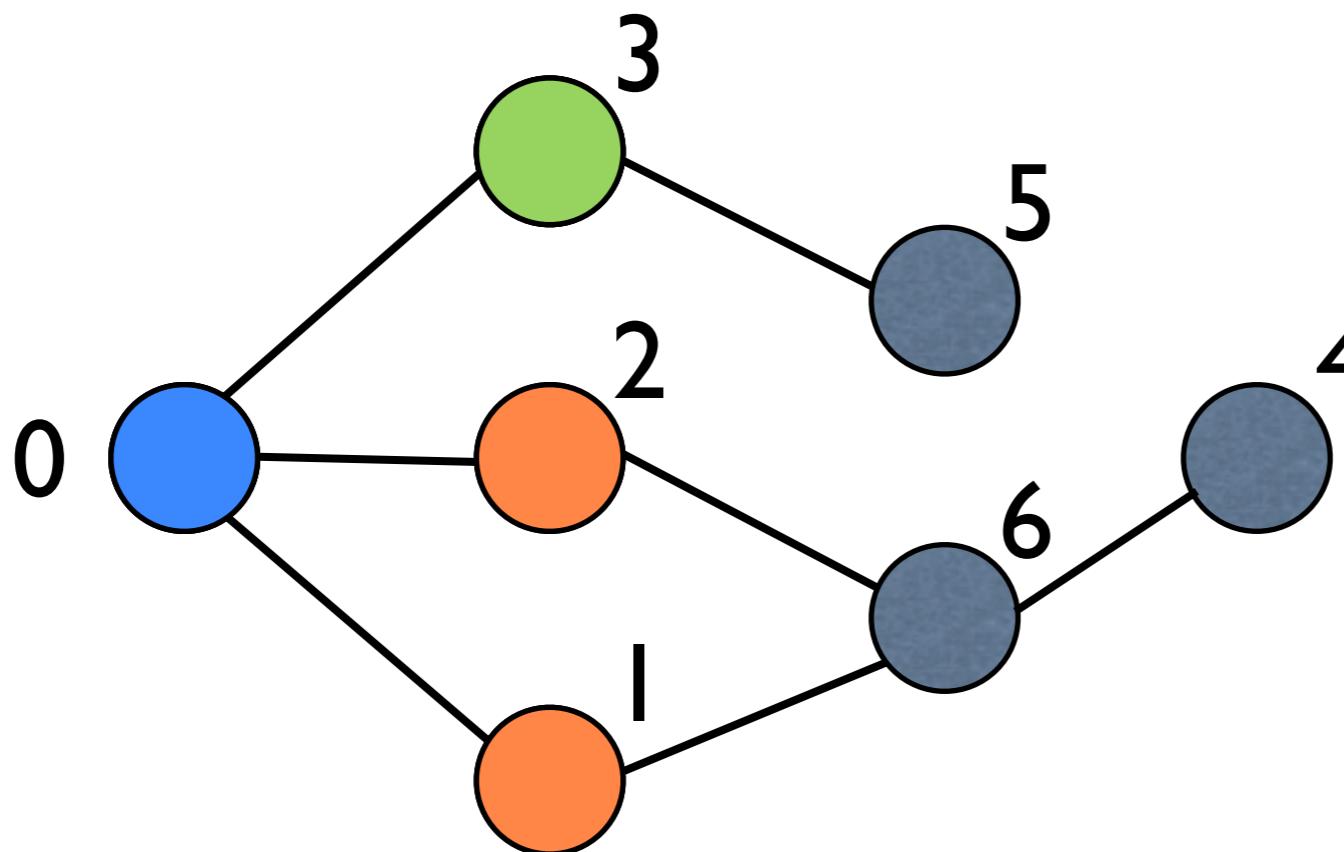


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 3 2 1

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	

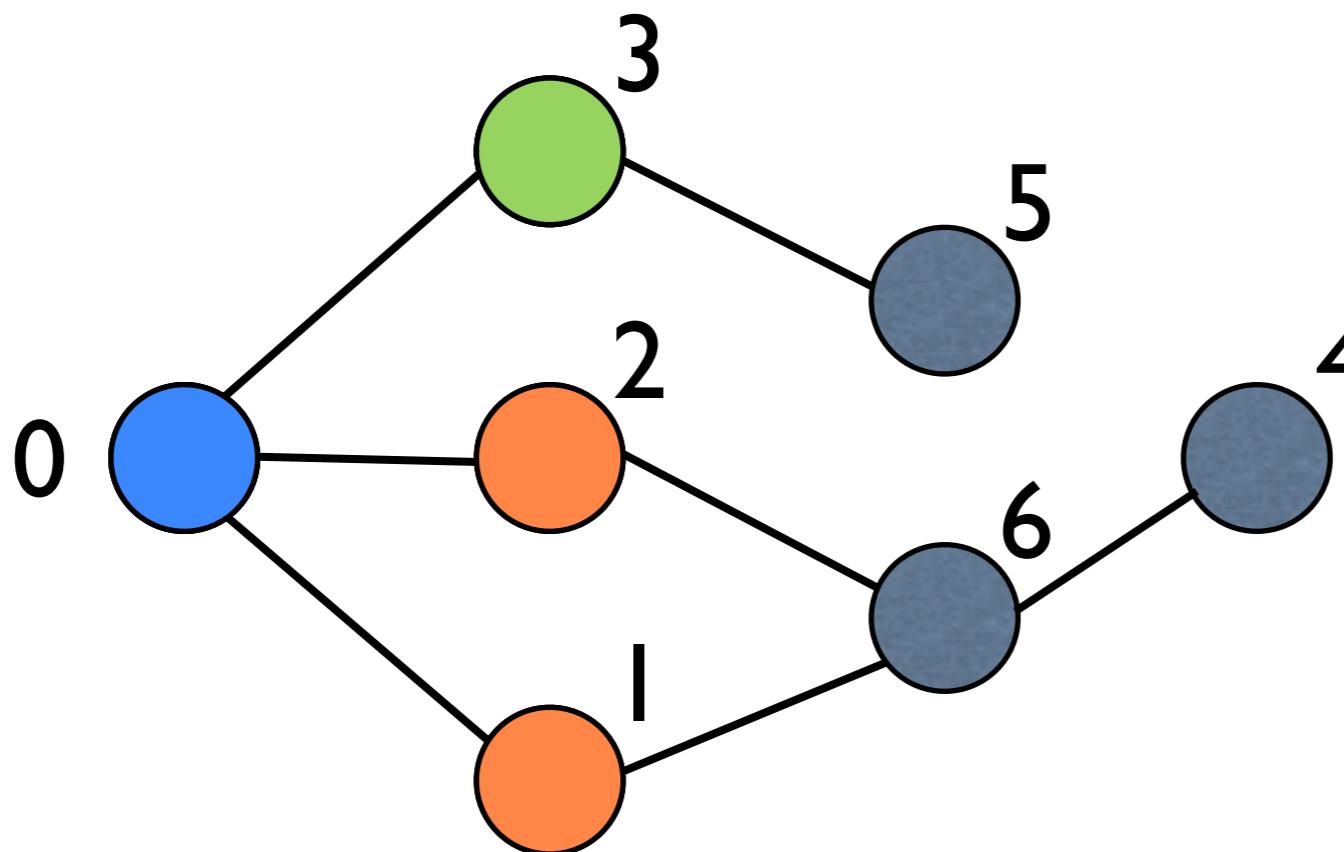


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 2 |

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	

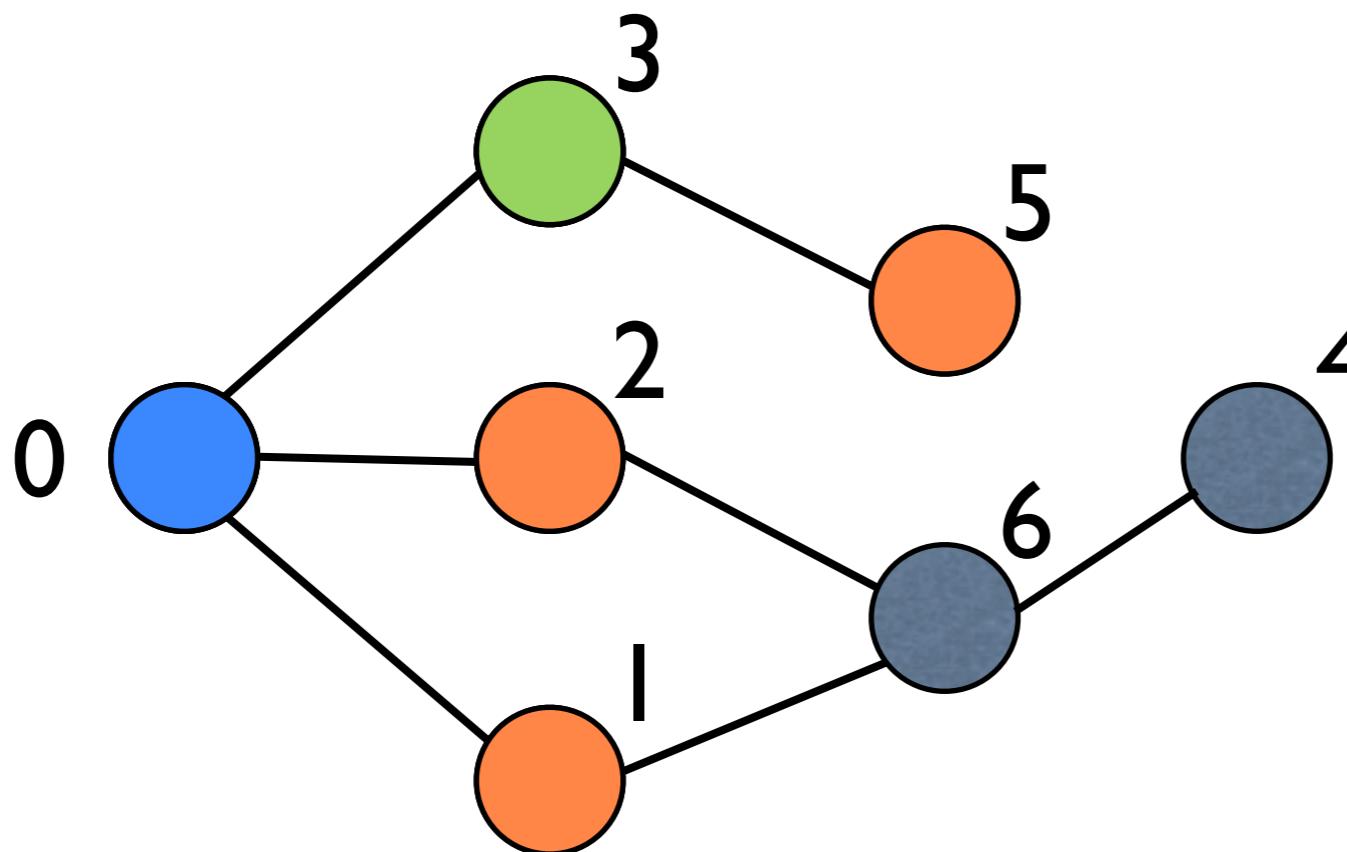


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 2 |

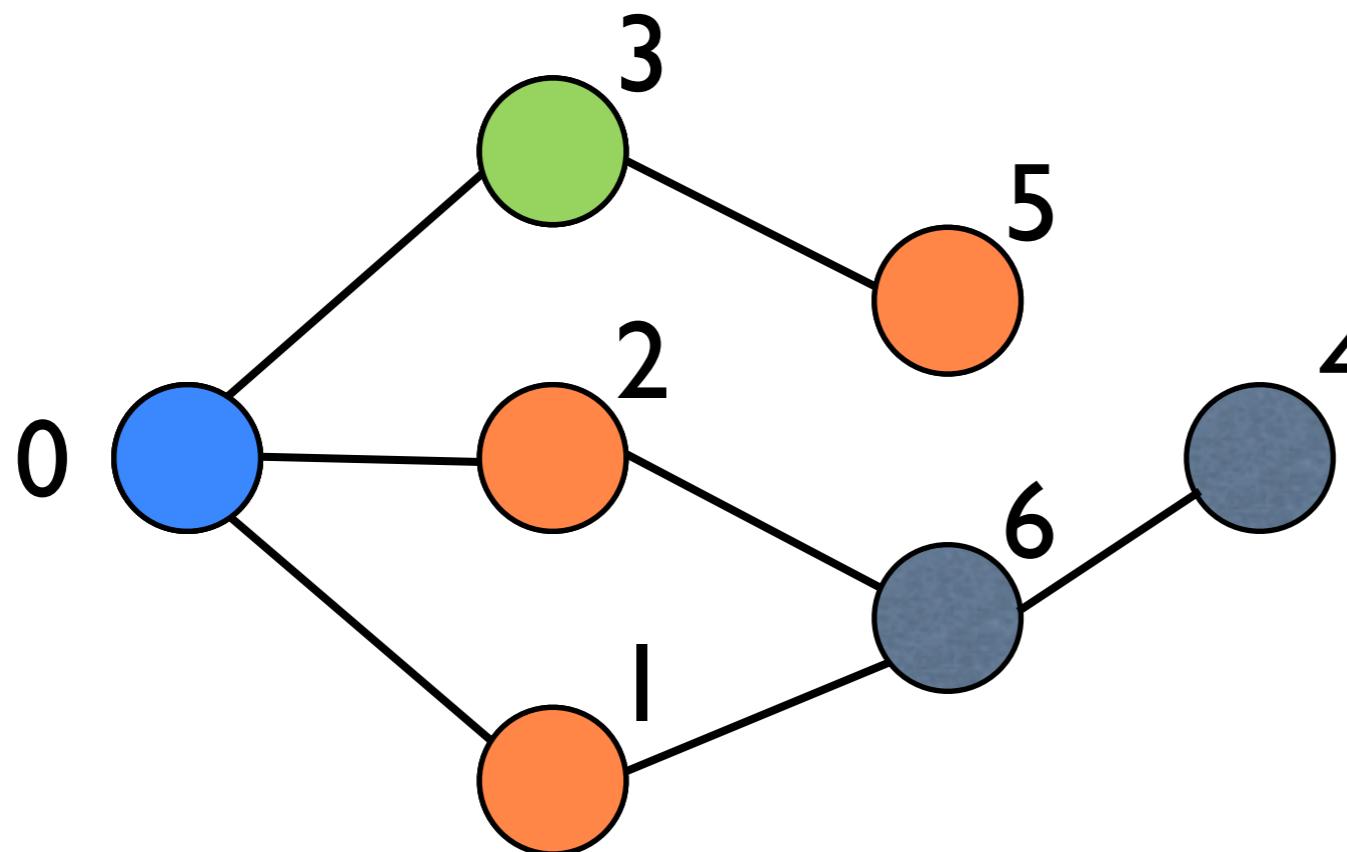
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	



Algoritmos de busca em grafos - Busca em largura

FILA: 2 | 5

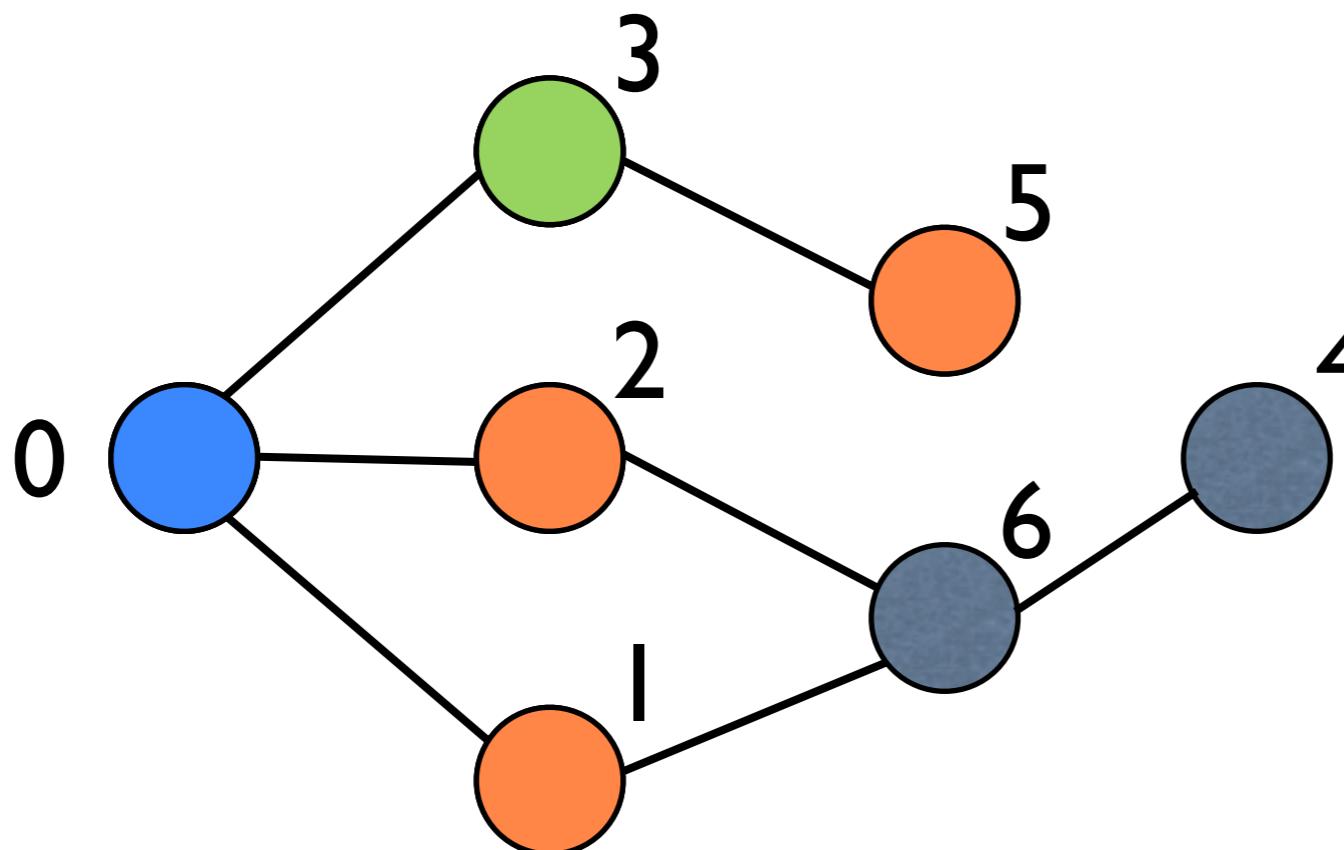
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	F	F	



Algoritmos de busca em grafos - Busca em largura

FILA: 2 | 5

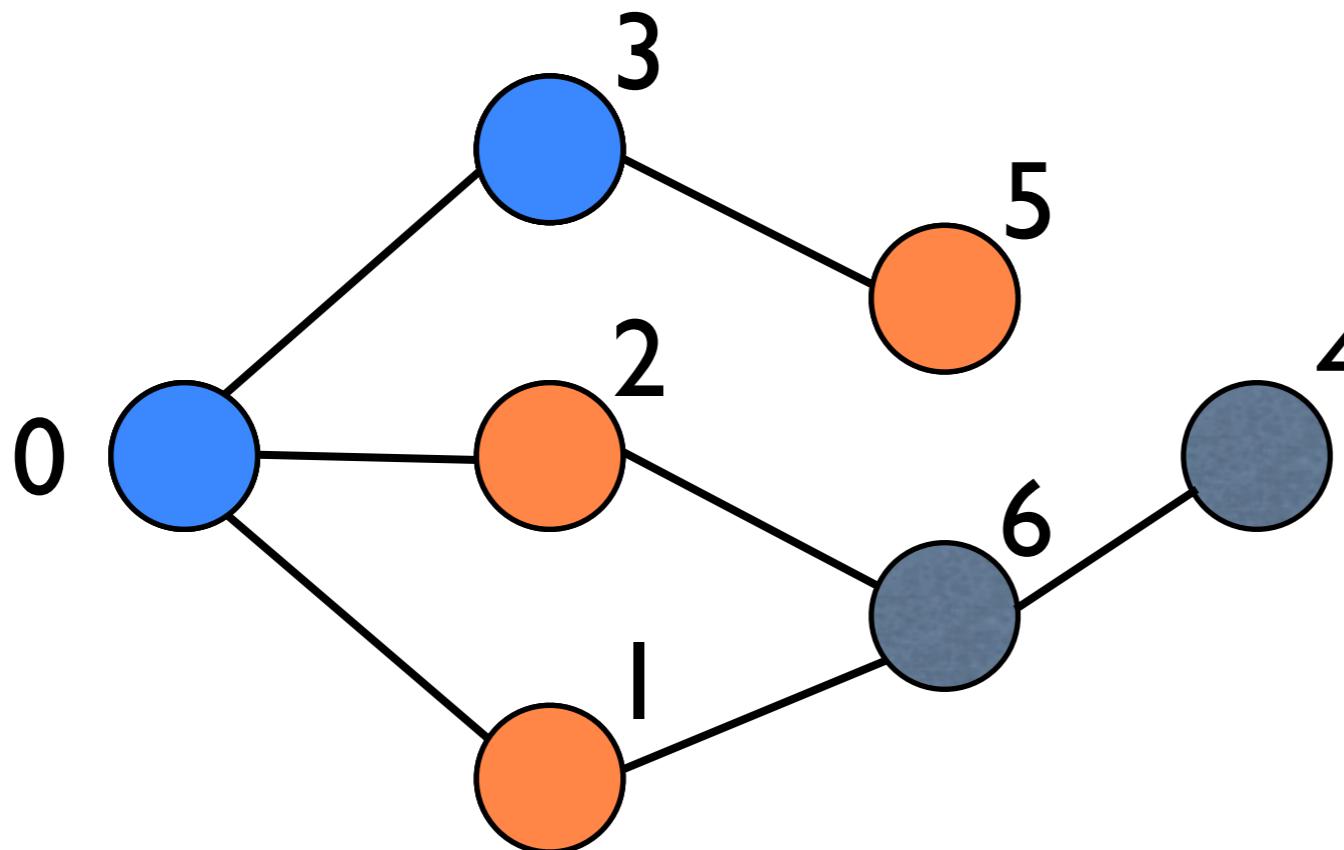
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	



Algoritmos de busca em grafos - Busca em largura

FILA: 2 | 5

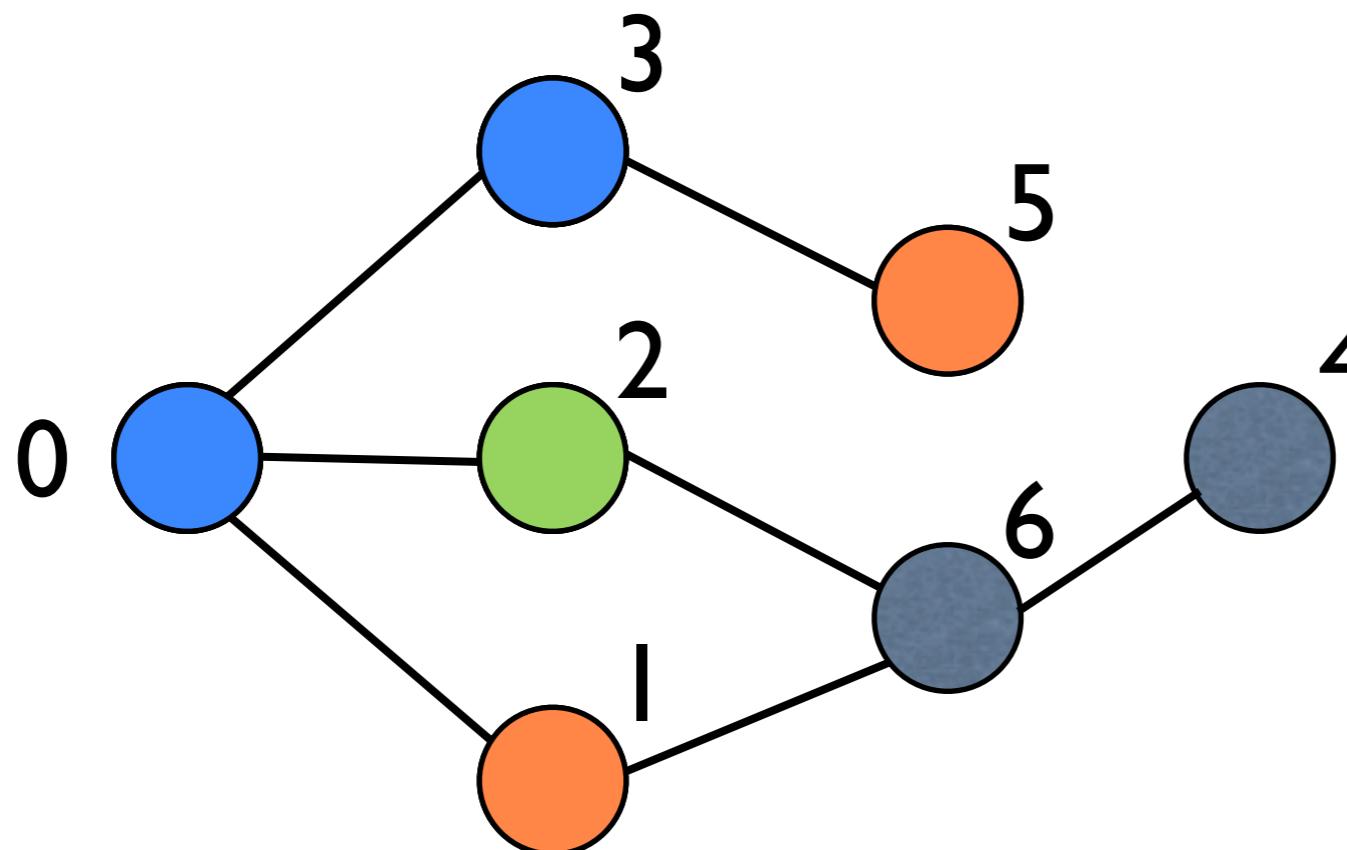
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	



Algoritmos de busca em grafos - Busca em largura

FILA: 2 | 5

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	

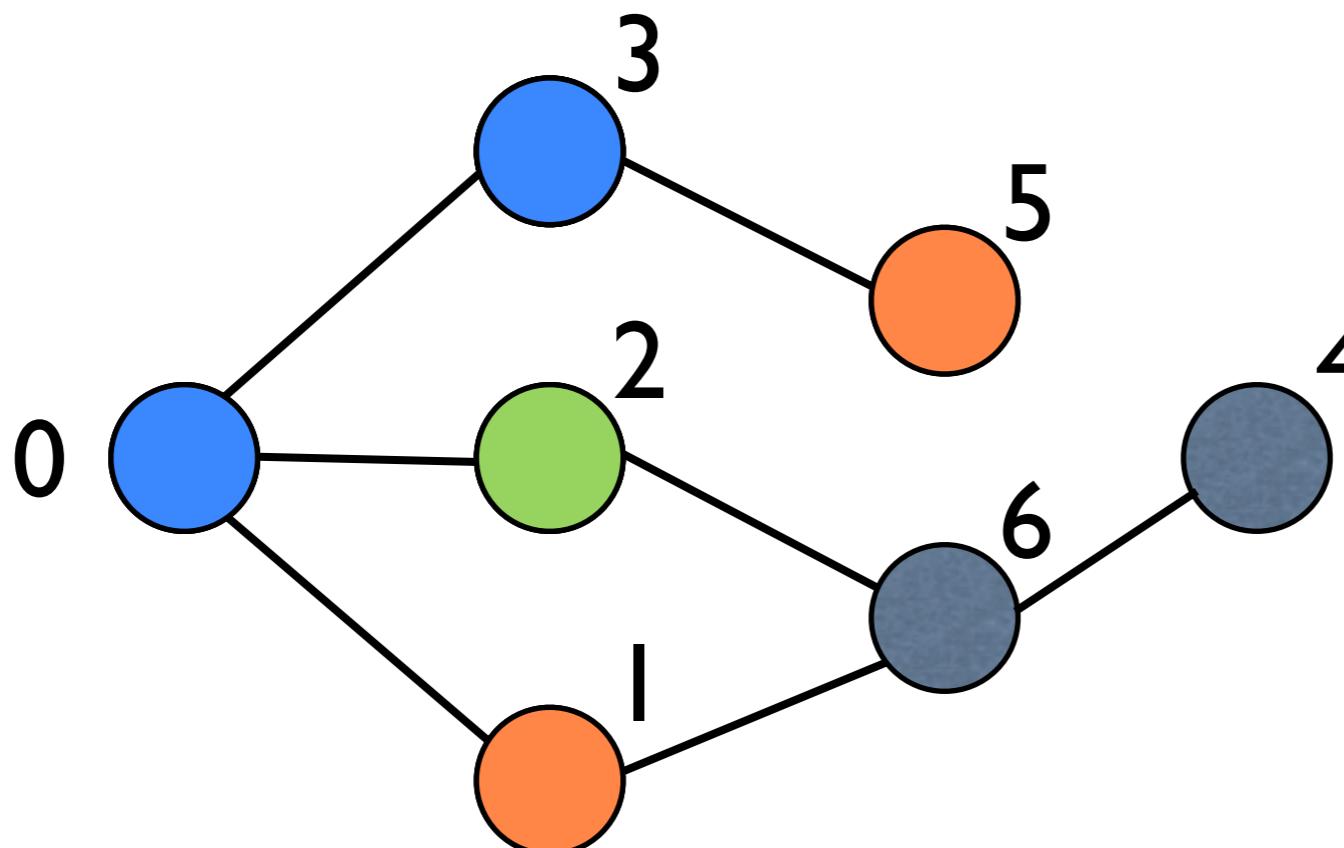


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: | 5

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	

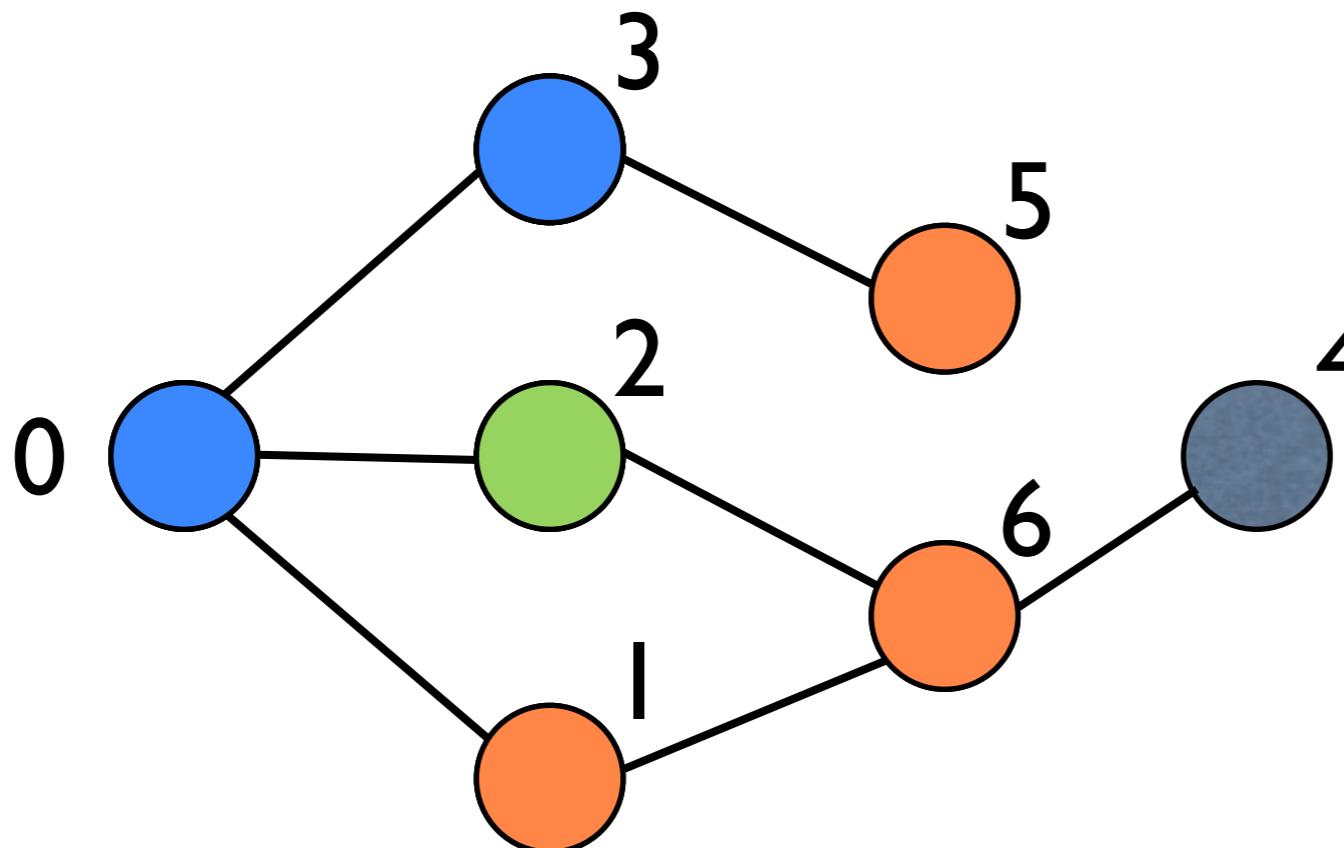


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: | 5

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	

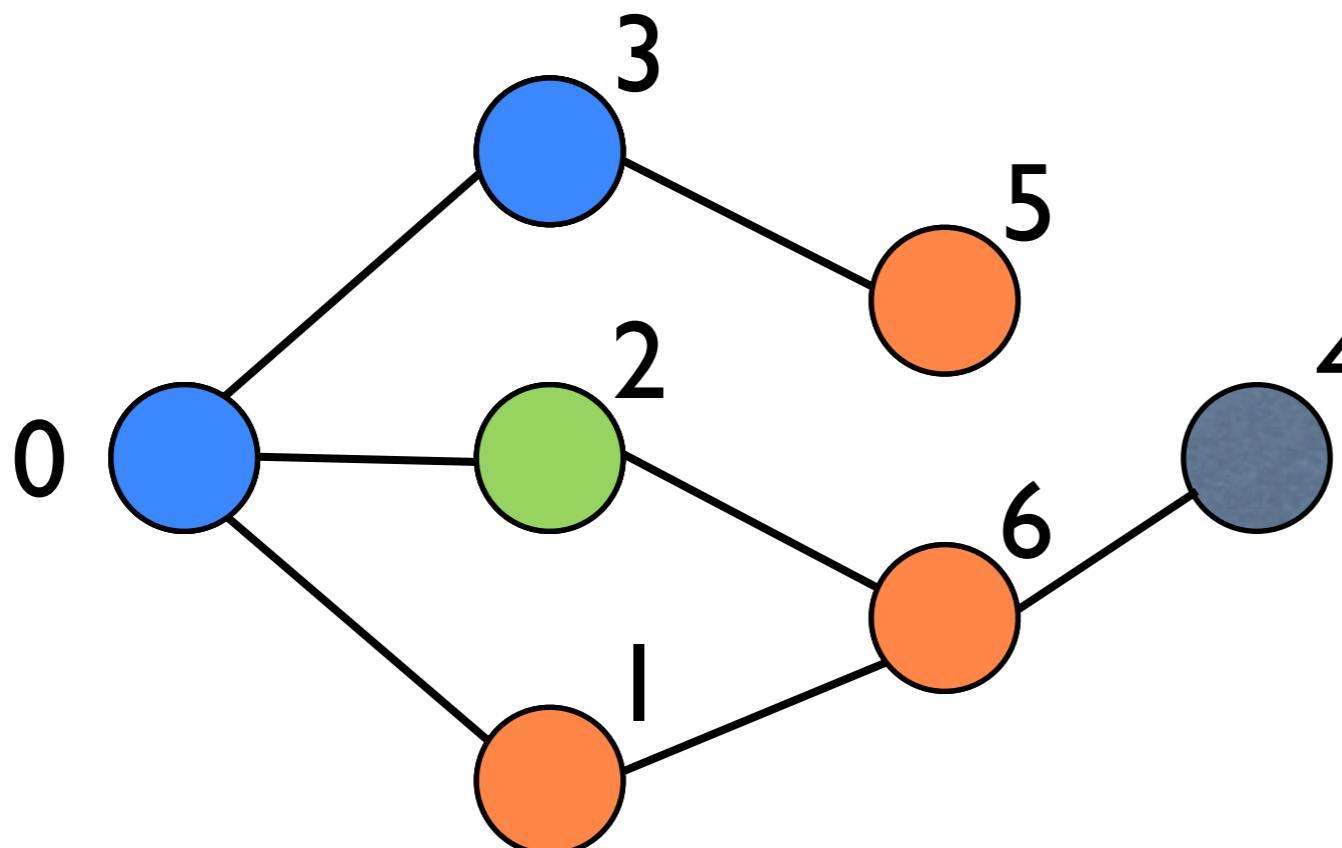


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: I 5 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	F	

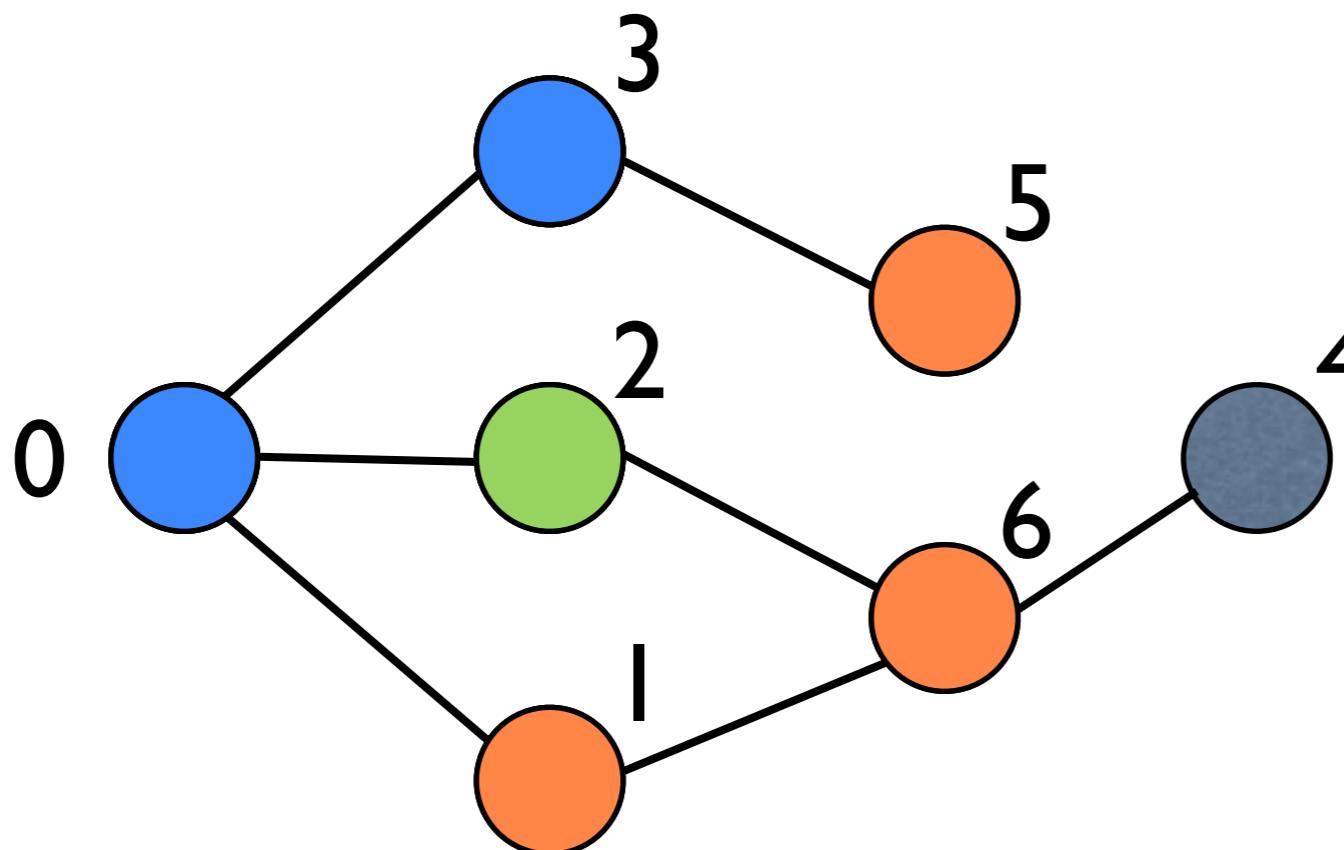


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: I 5 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

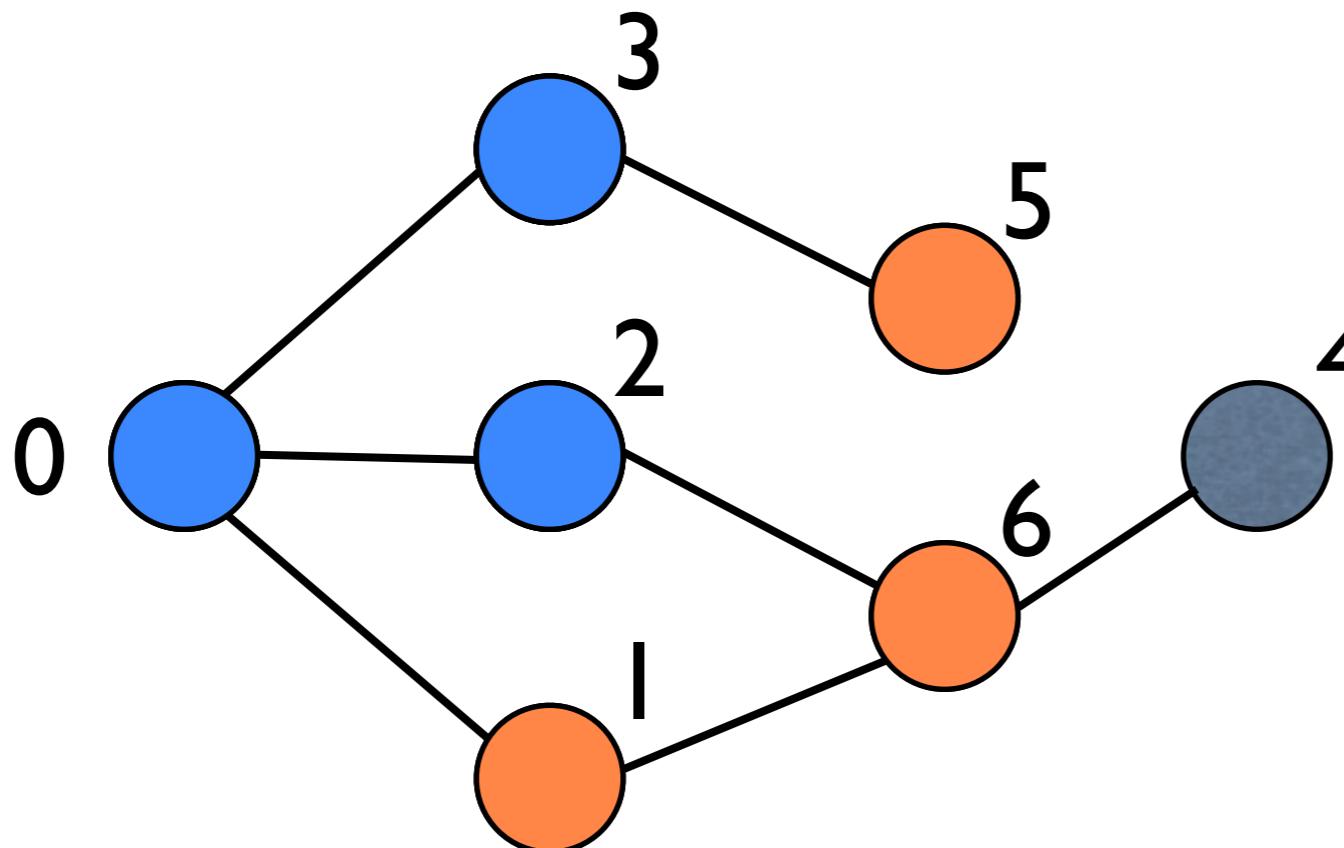


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: I 5 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

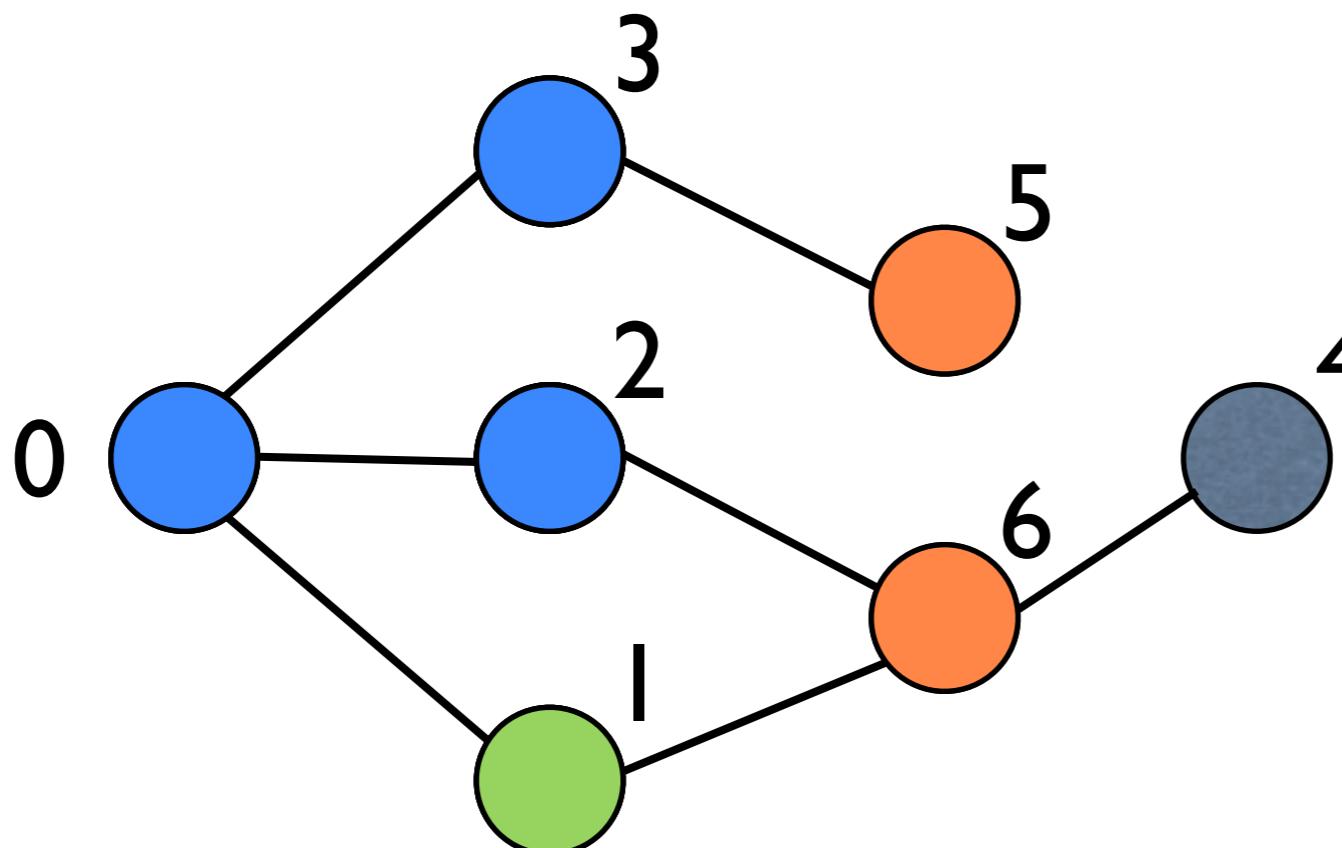


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: I 5 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

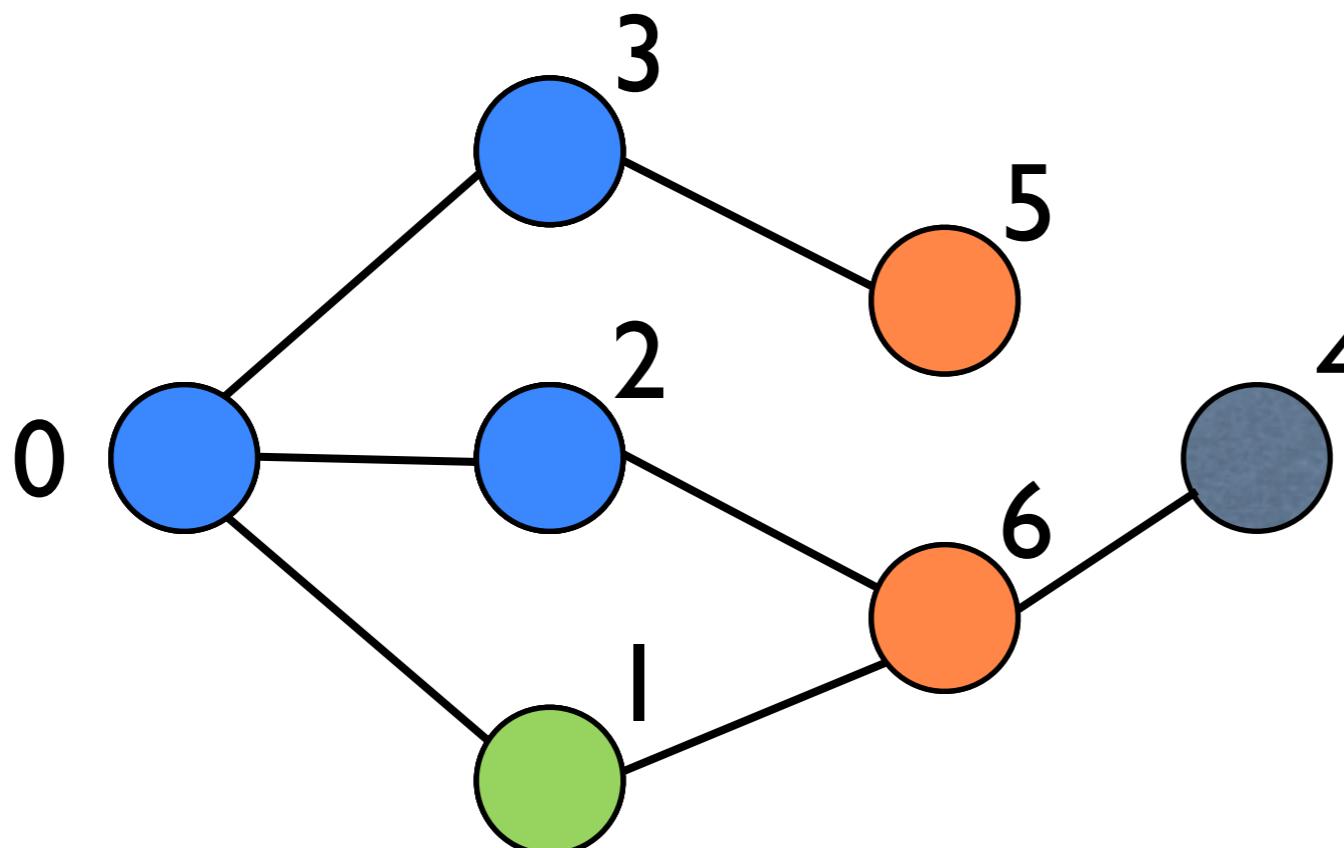


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 5 6

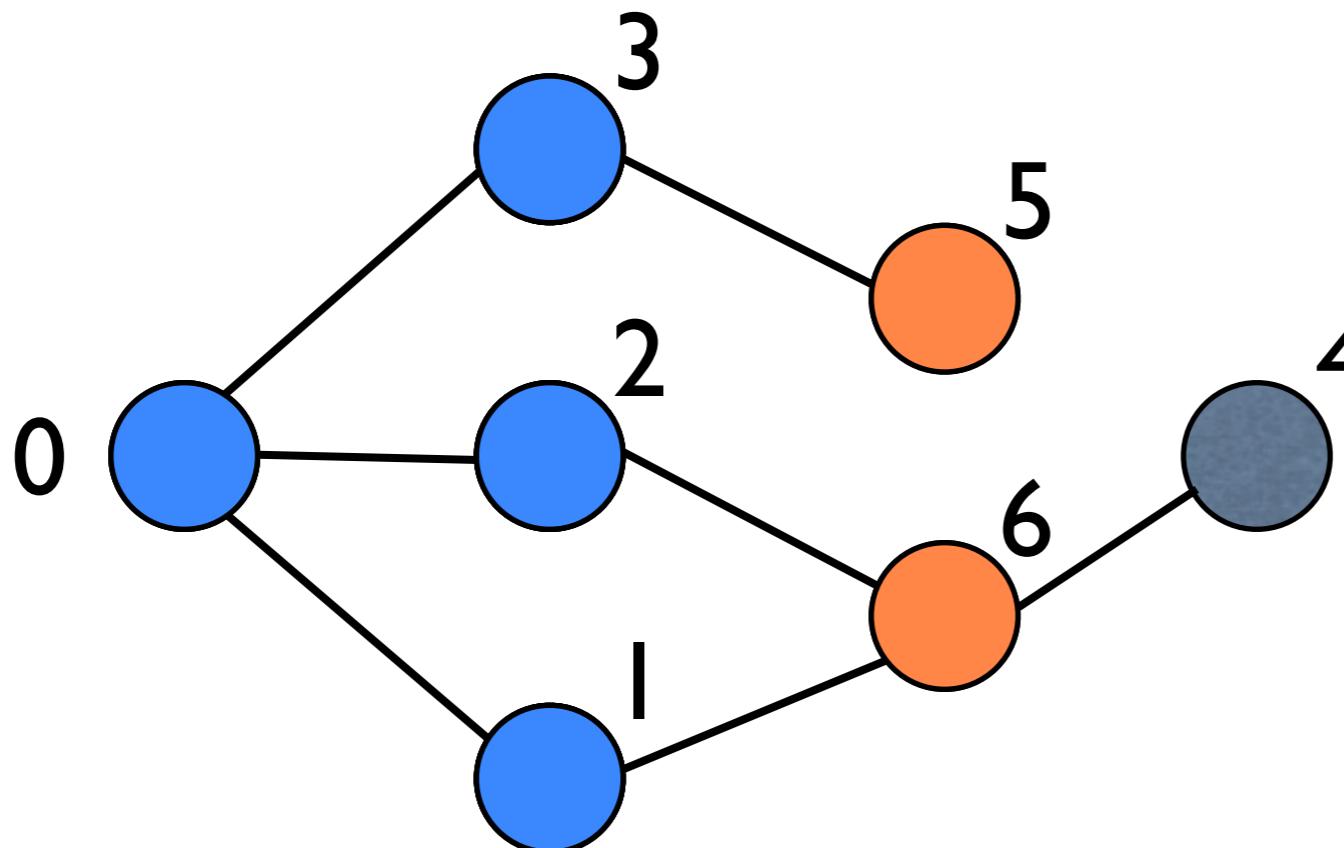
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	



Algoritmos de busca em grafos - Busca em largura

FILA: 5 6

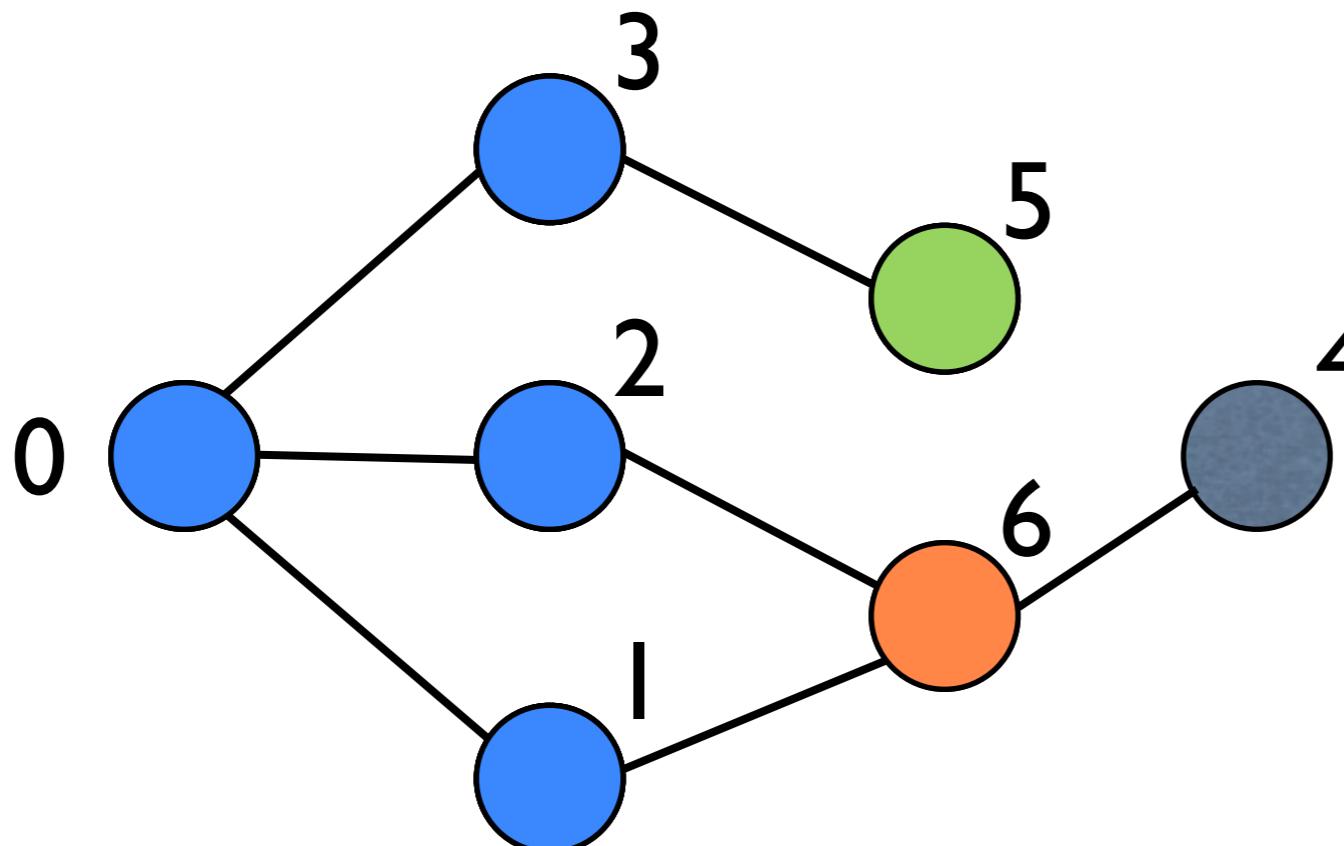
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	



Algoritmos de busca em grafos - Busca em largura

FILA: 5 6

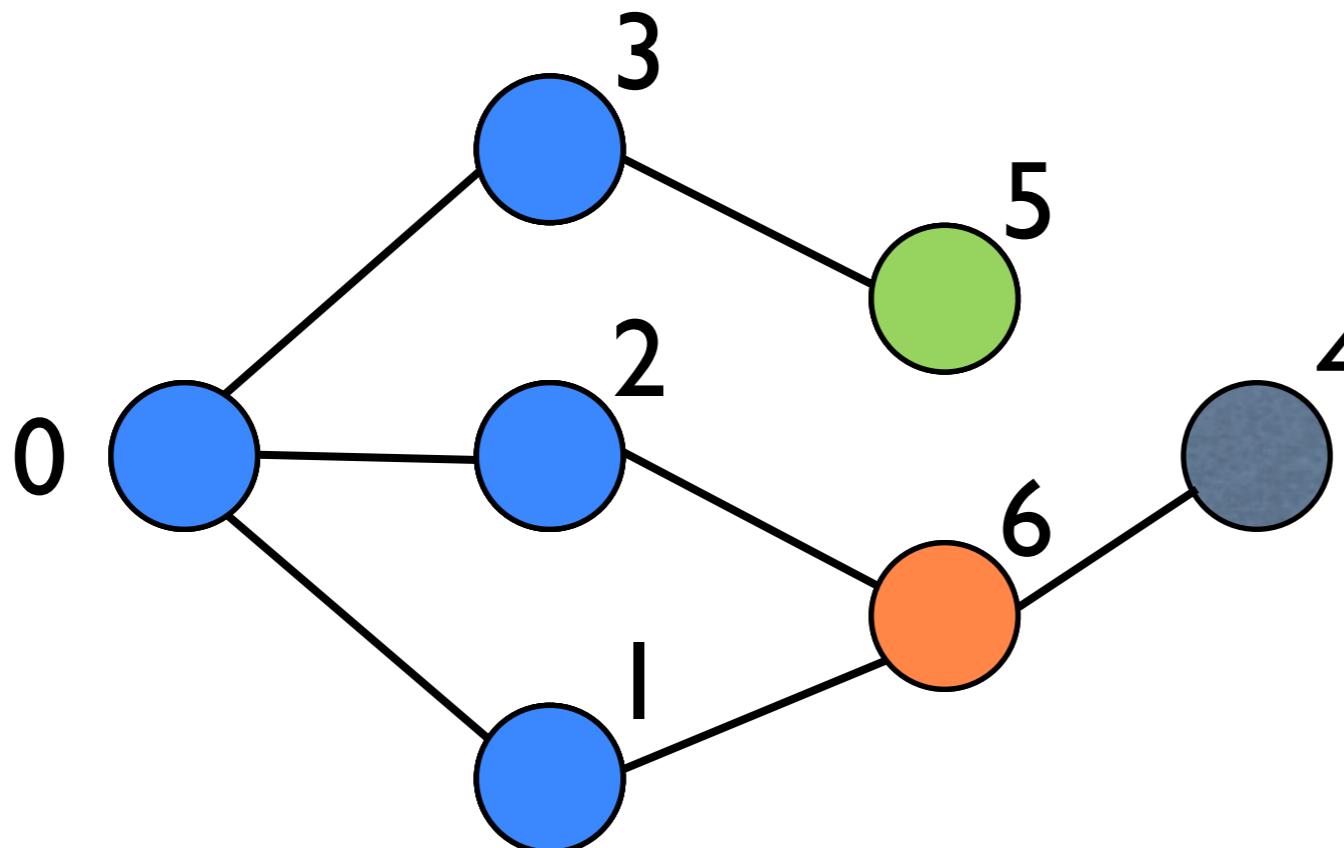
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	



Algoritmos de busca em grafos - Busca em largura

FILA: 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

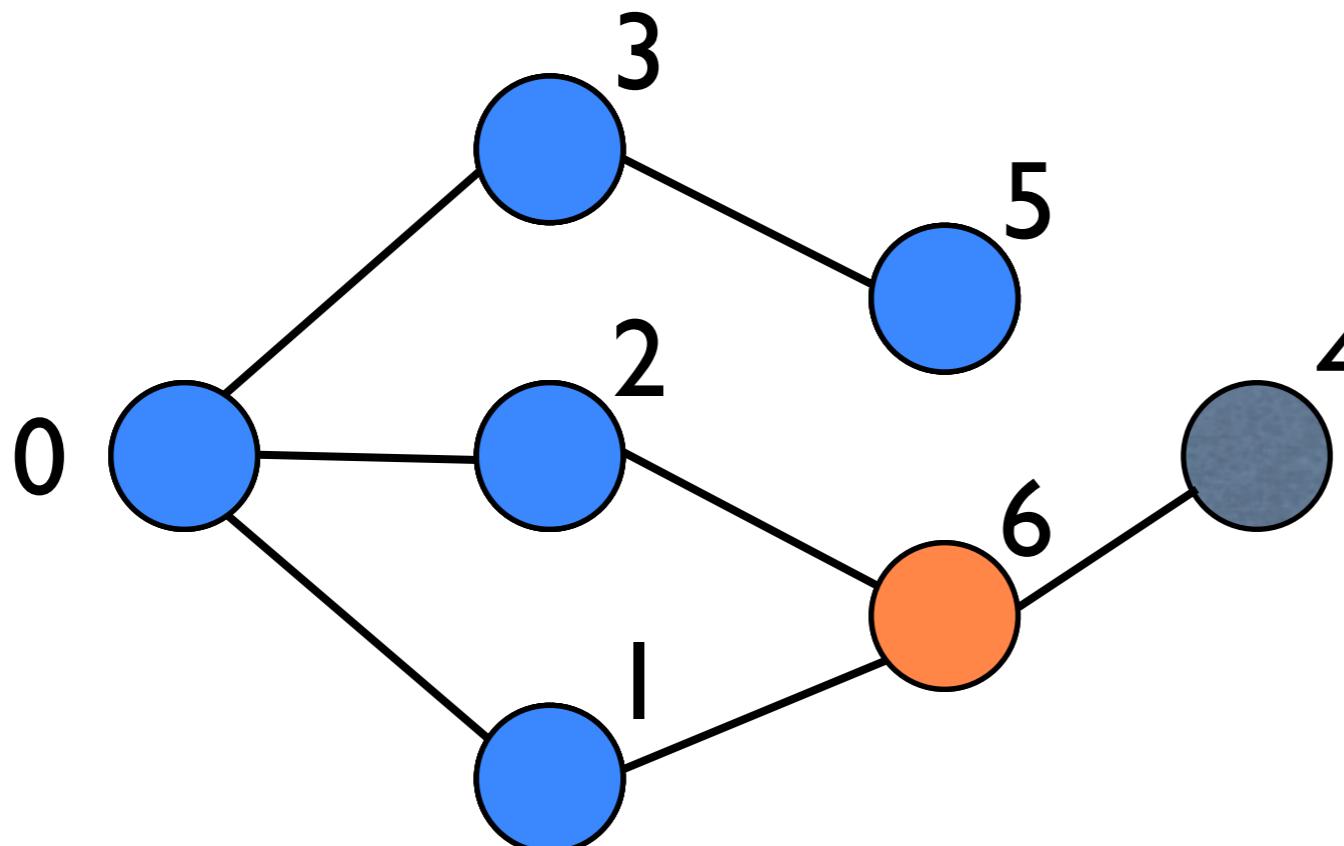


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 6

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

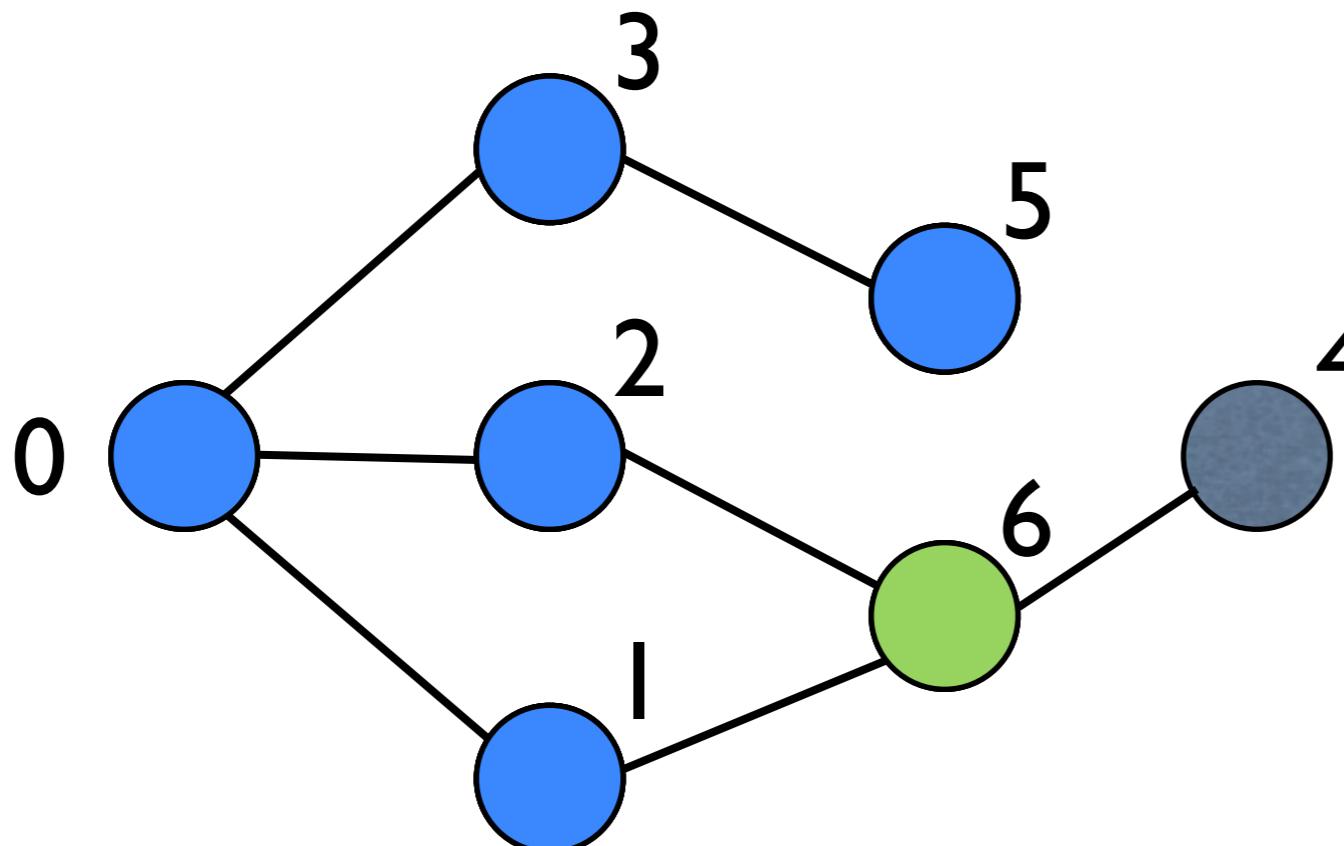


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 6

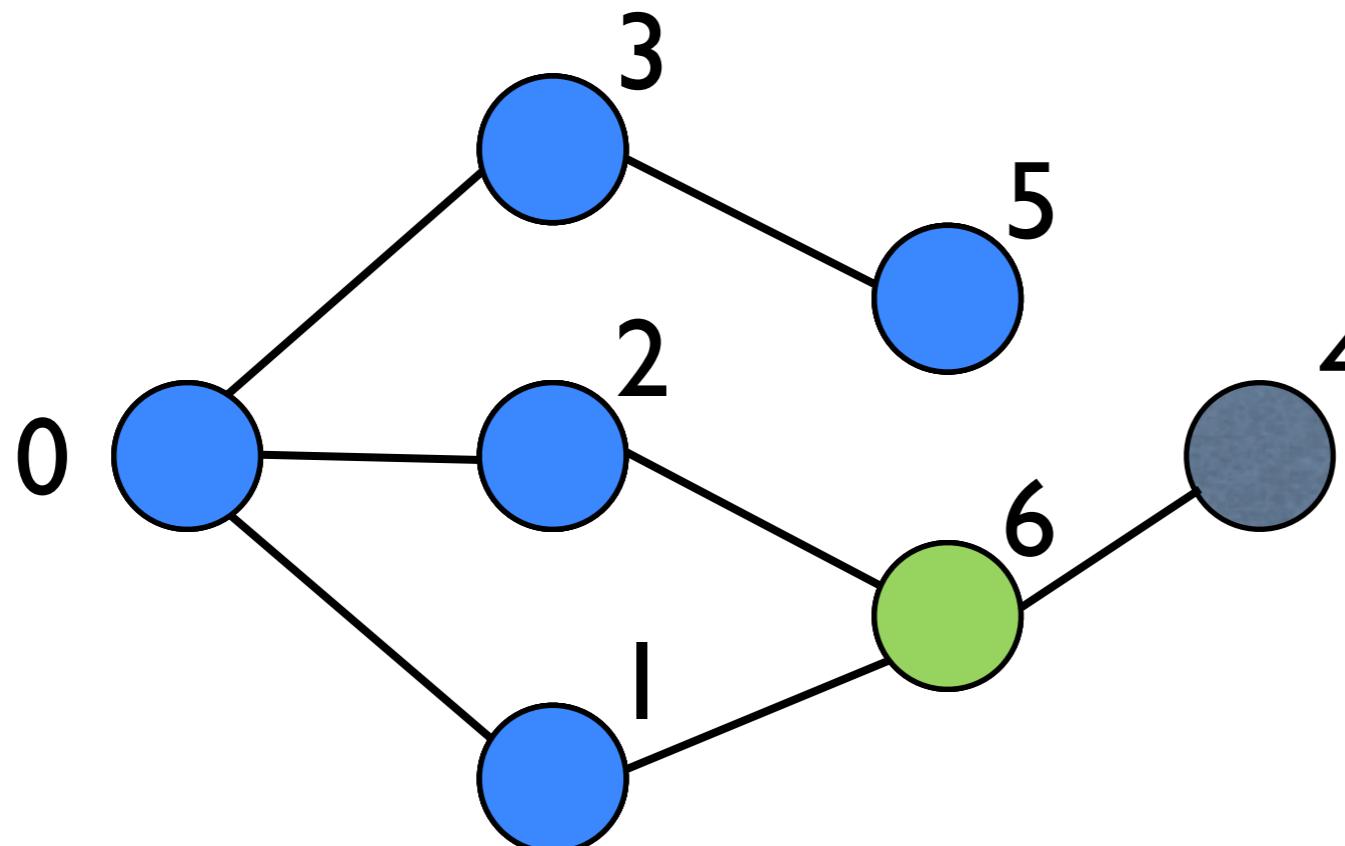
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	



	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA:

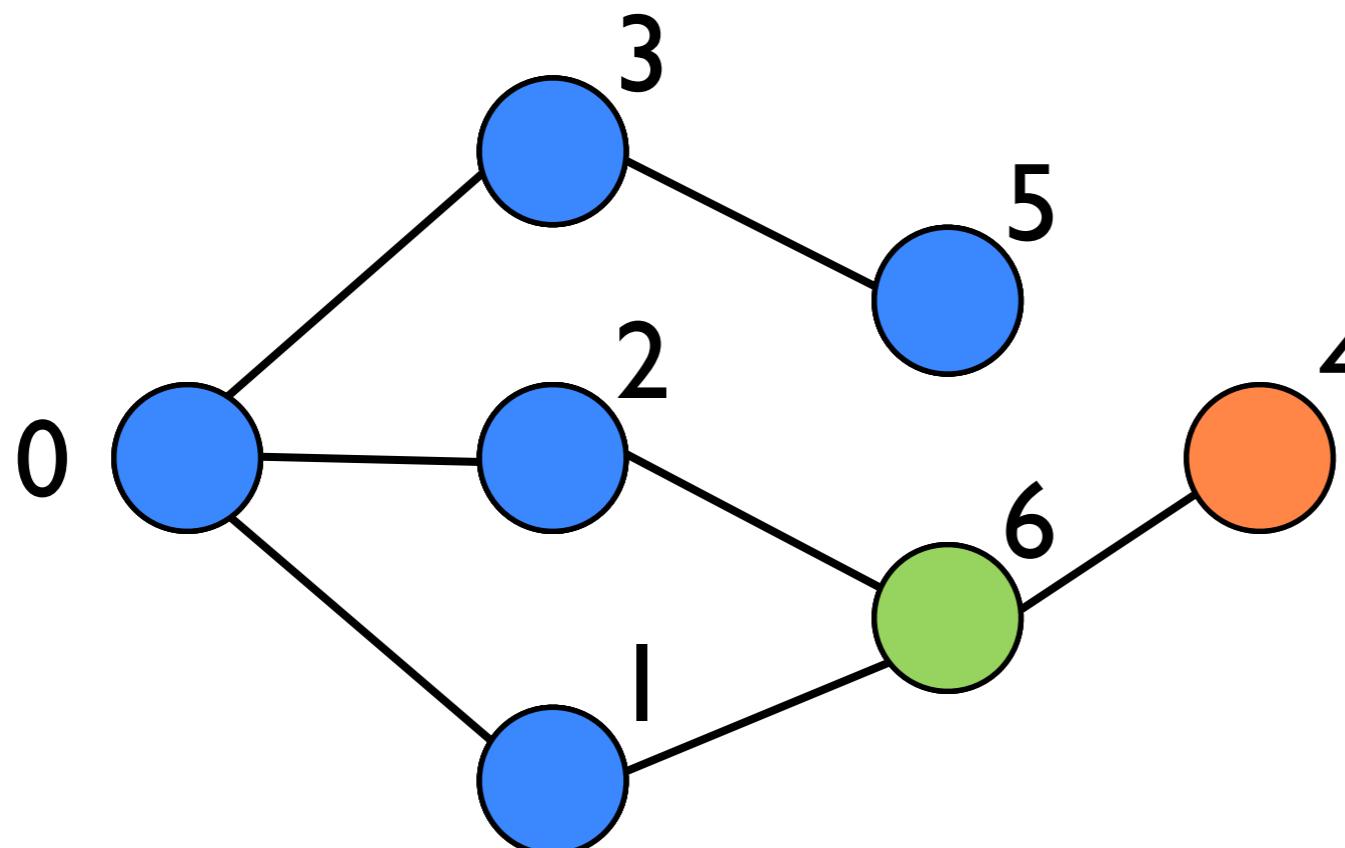


VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA:



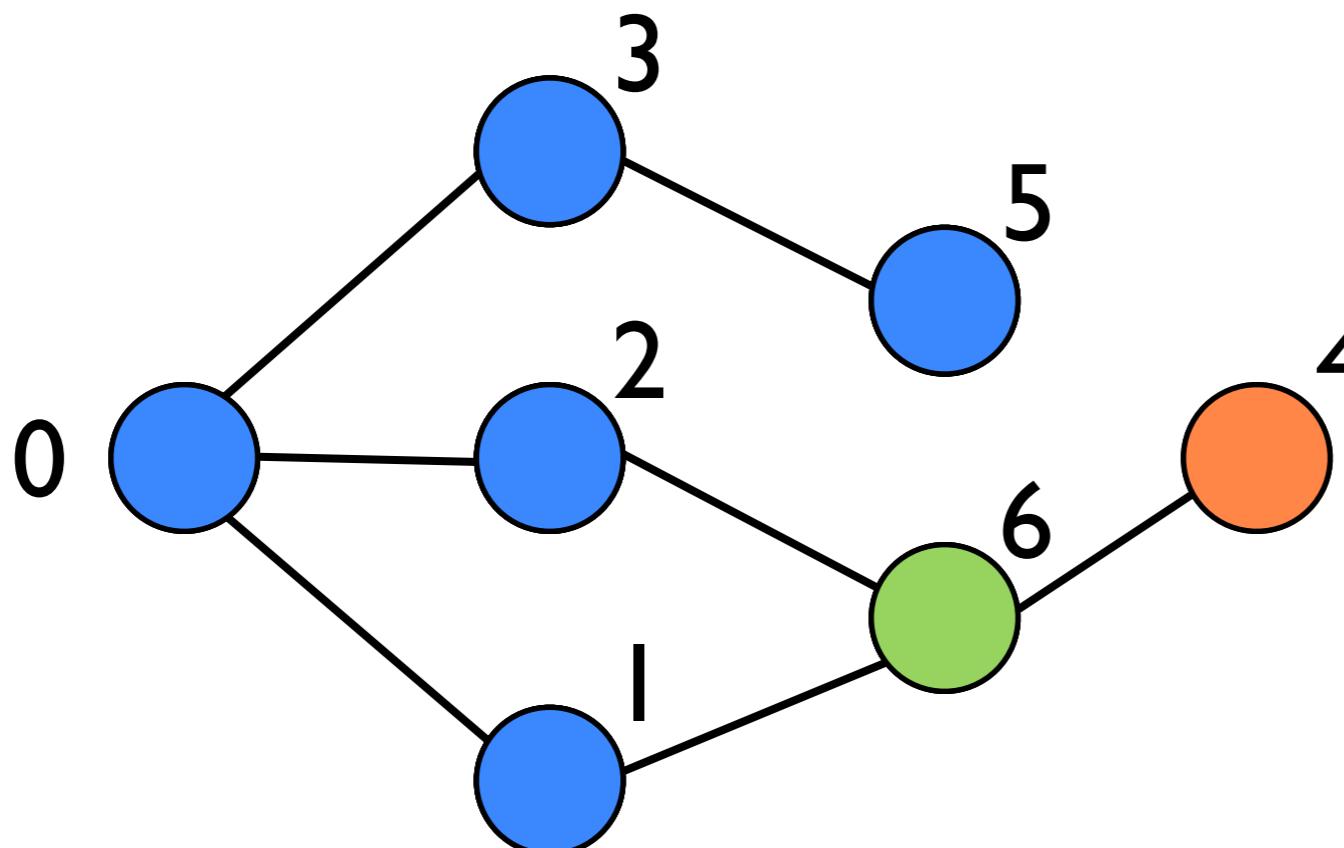
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 4

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	F	T	T	

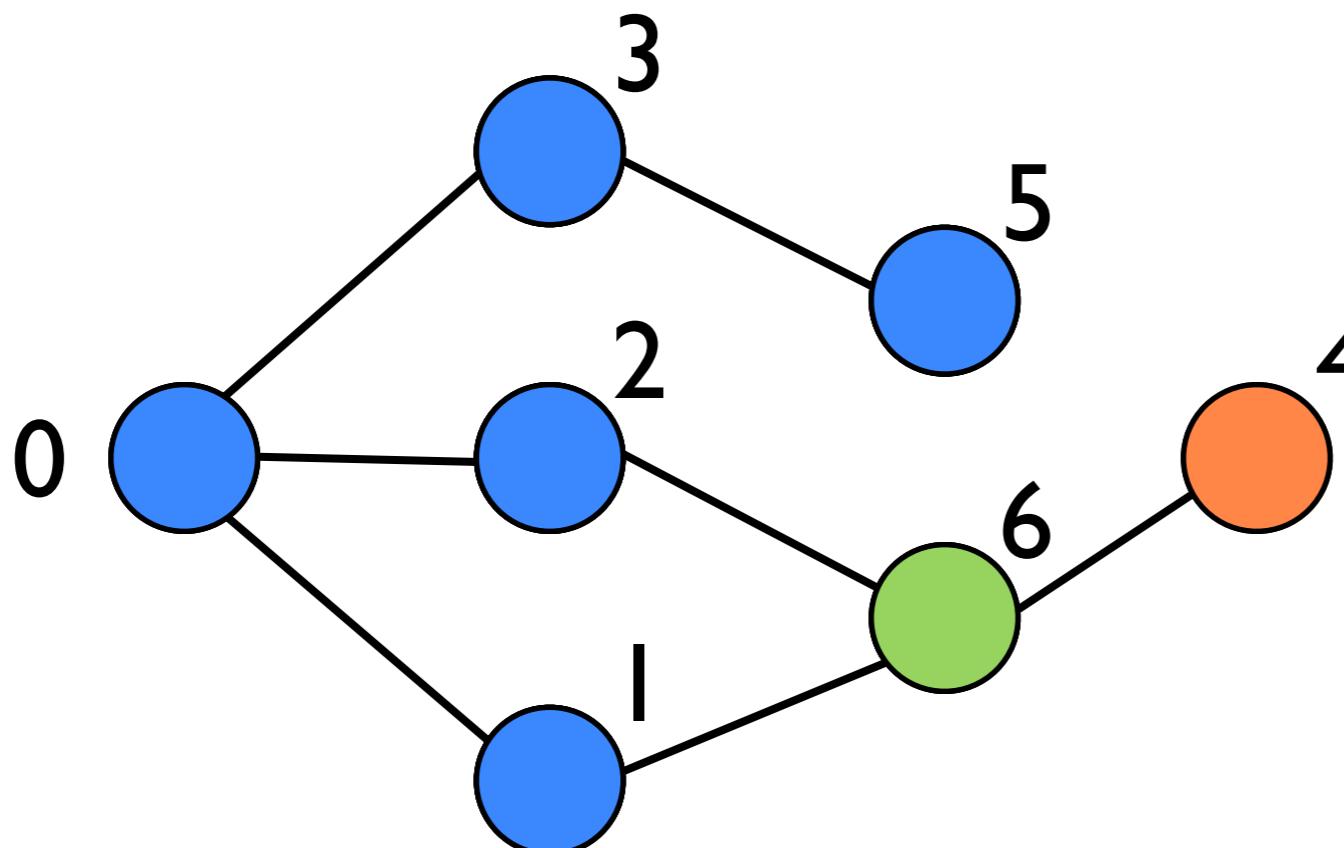


	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 4

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	T	T	T	T

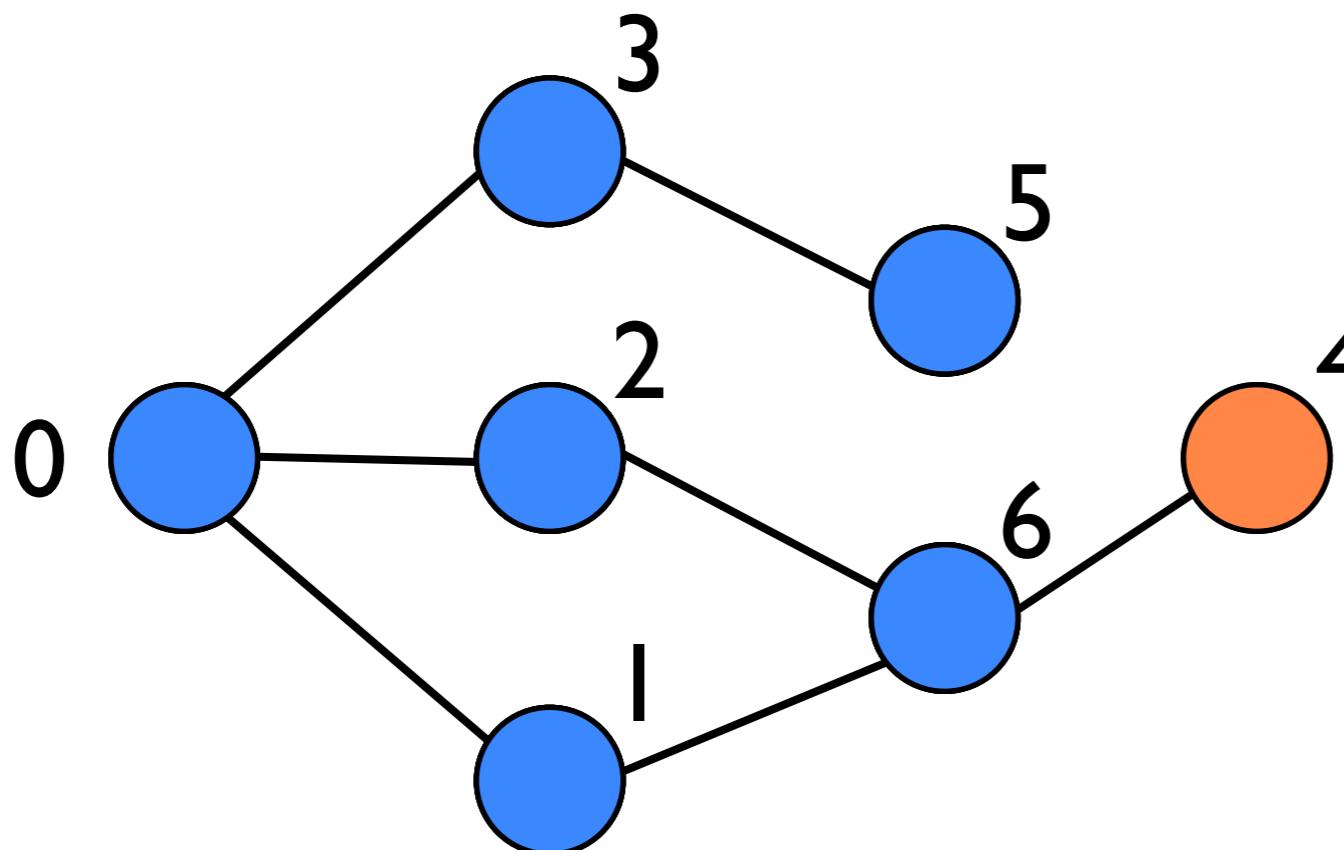


	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 4

VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	T	T	T	T

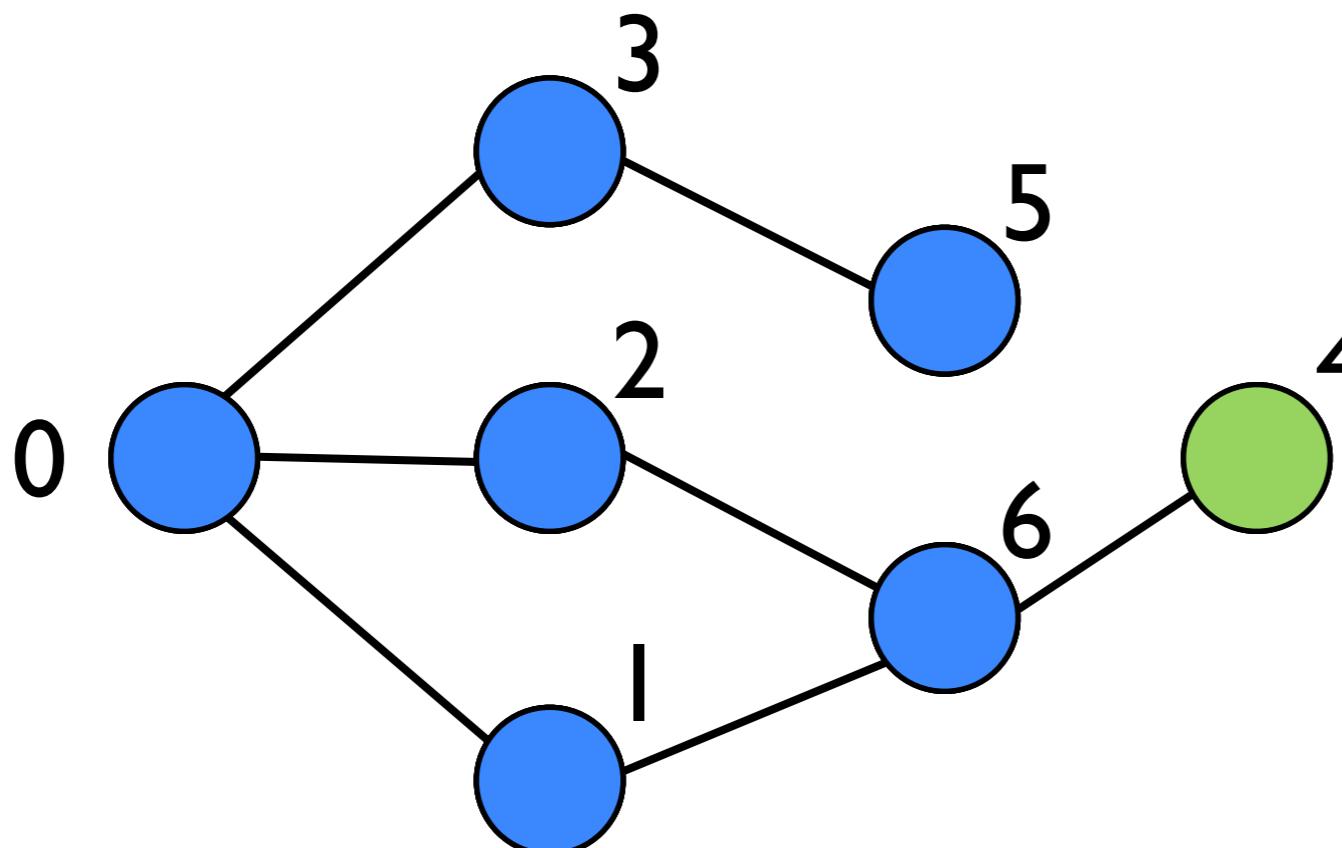


0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA: 4

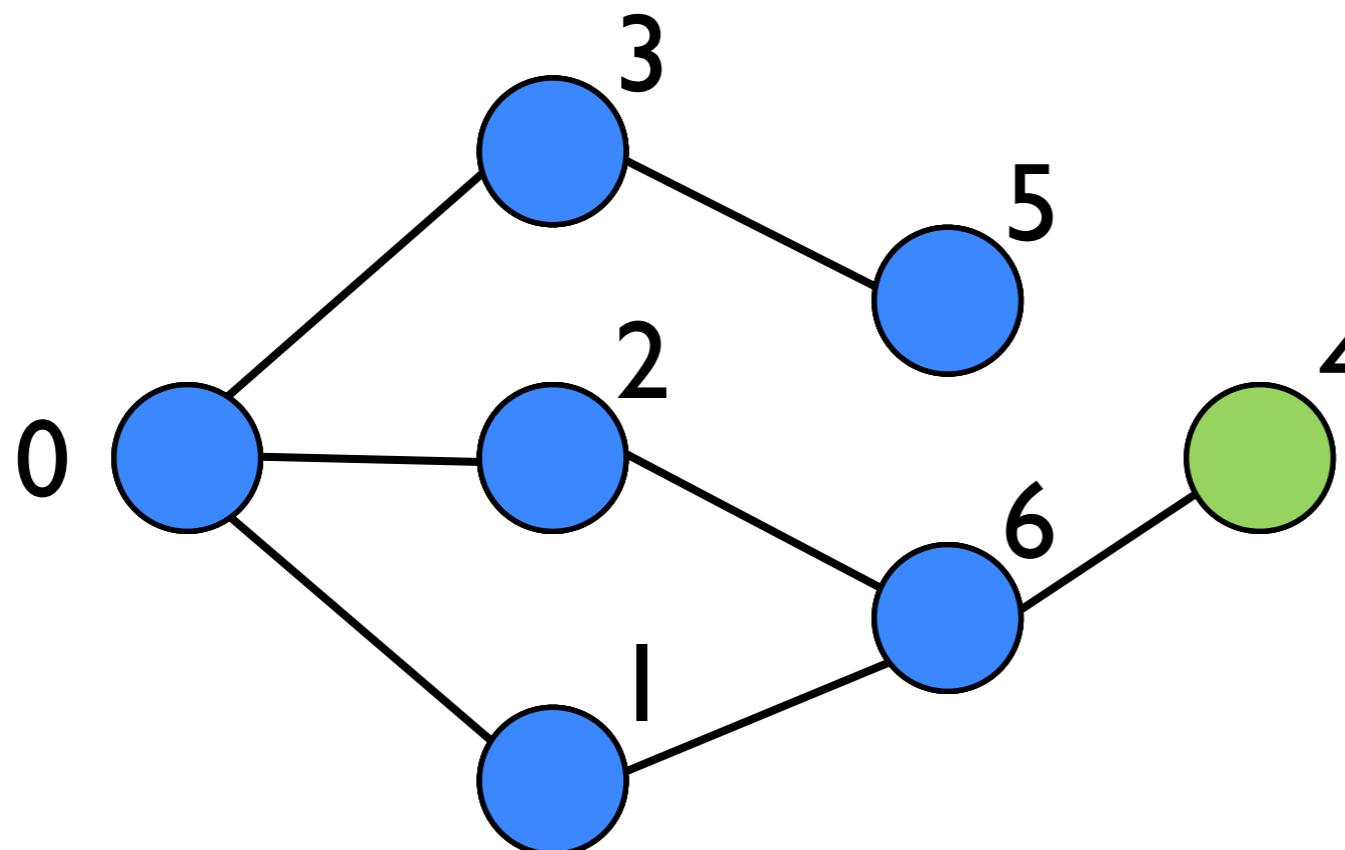
VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	T	T	T	T



0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA:

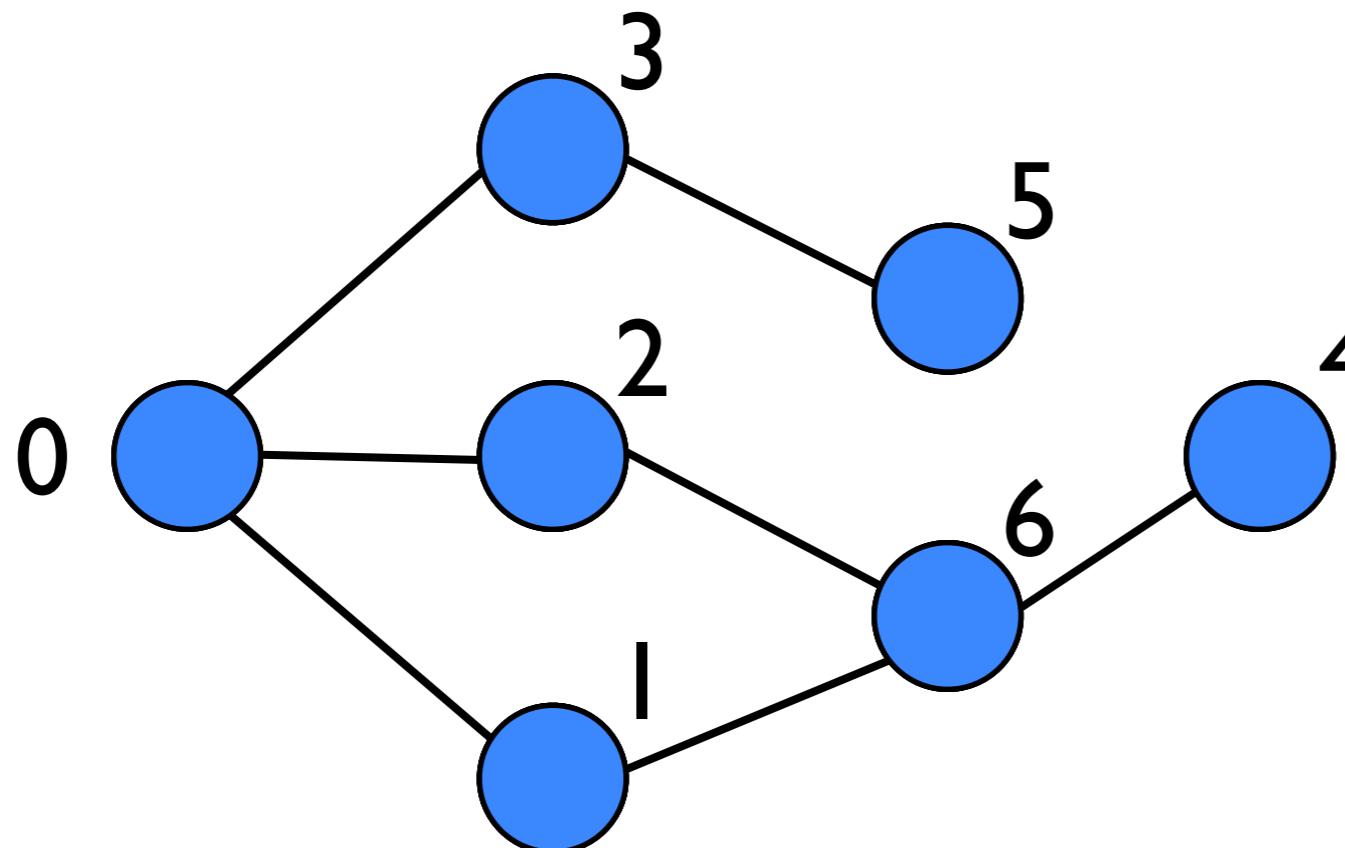


VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	T	T	T	T

0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

Algoritmos de busca em grafos - Busca em largura

FILA:

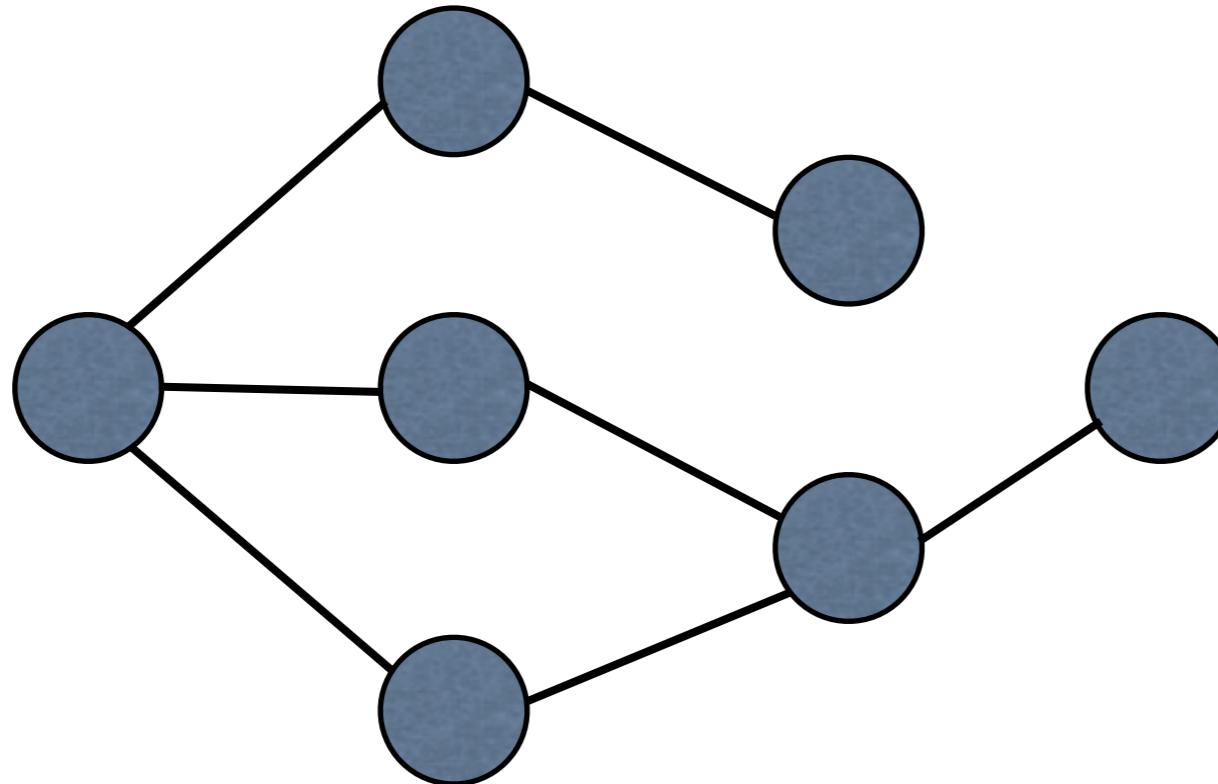


VISITADO							
0	1	2	3	4	5	6	
T	T	T	T	T	T	T	T

0	0	1	2	3	4	5	6
0	-	I	I	I	0	0	0
1	I	-	0	0	0	0	I
2	I	0	-	0	0	0	I
3	I	0	0	-	0	I	0
4	0	0	0	0	-	0	I
5	0	0	0	I	0	-	0
6	0	I	I	0	I	0	-

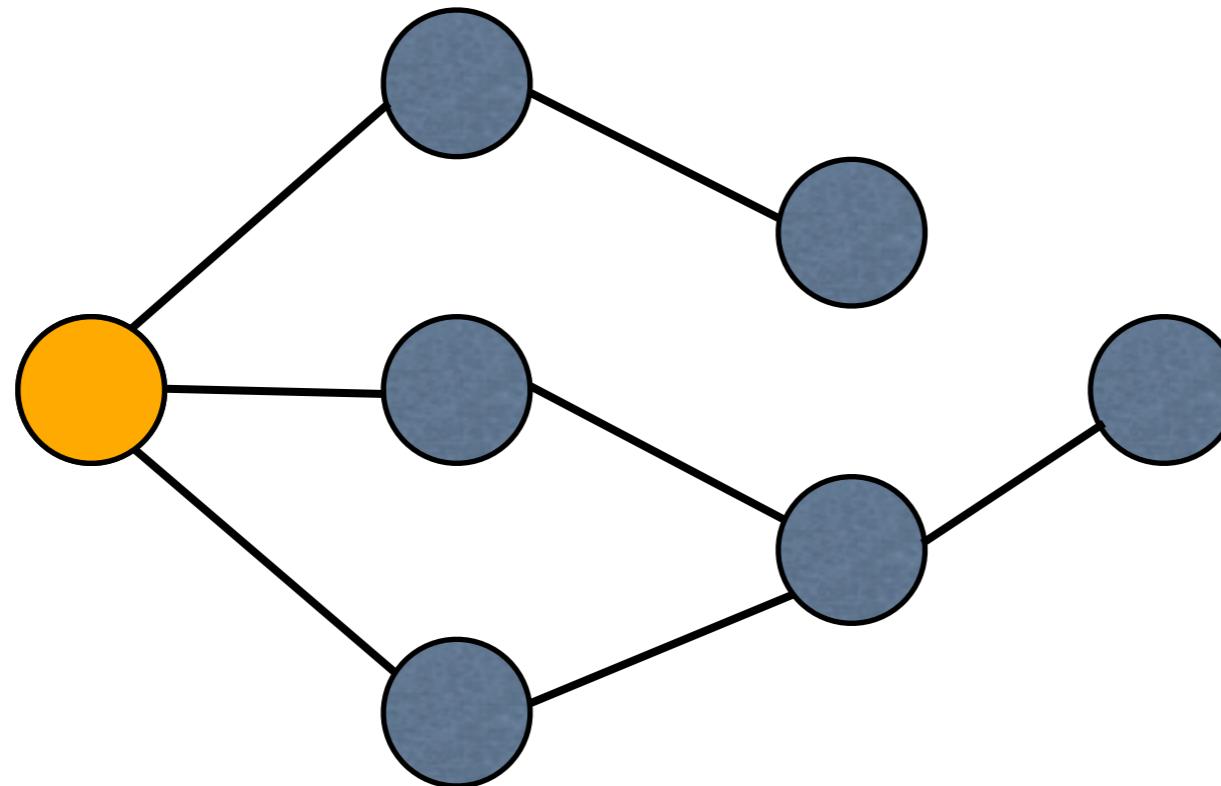
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



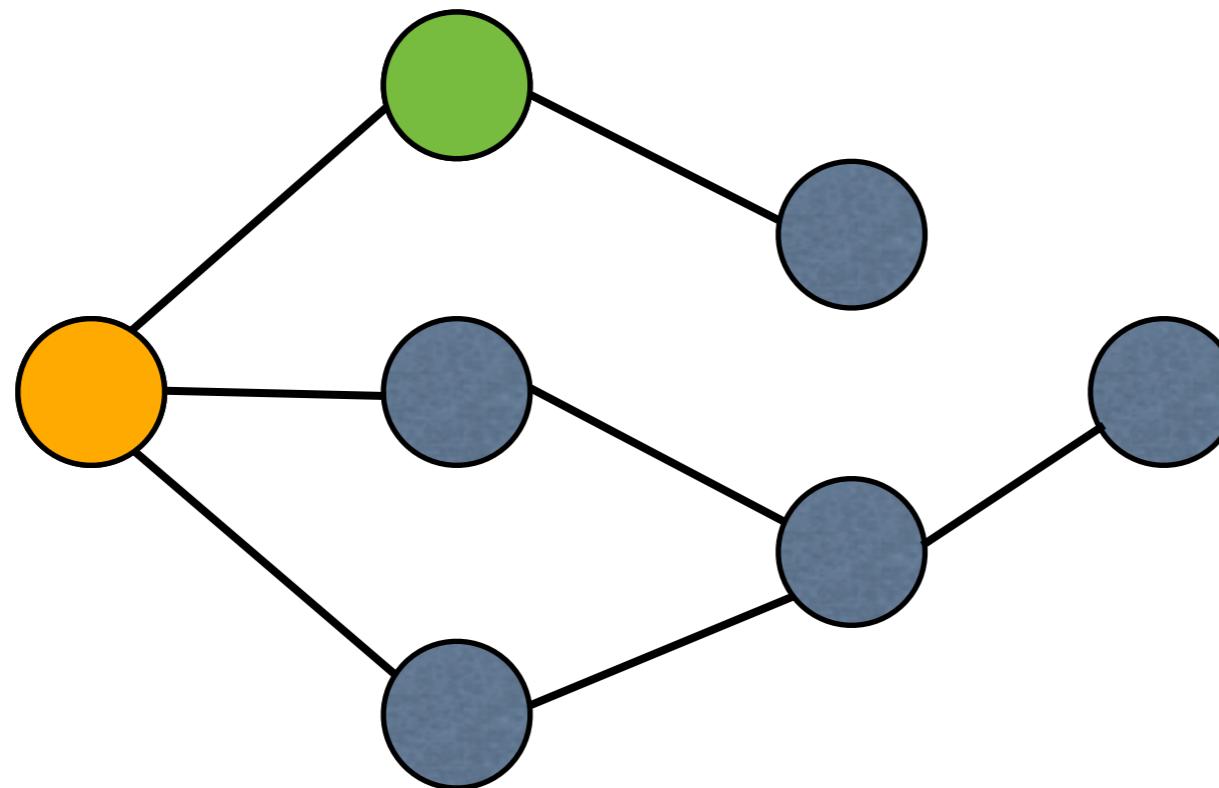
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



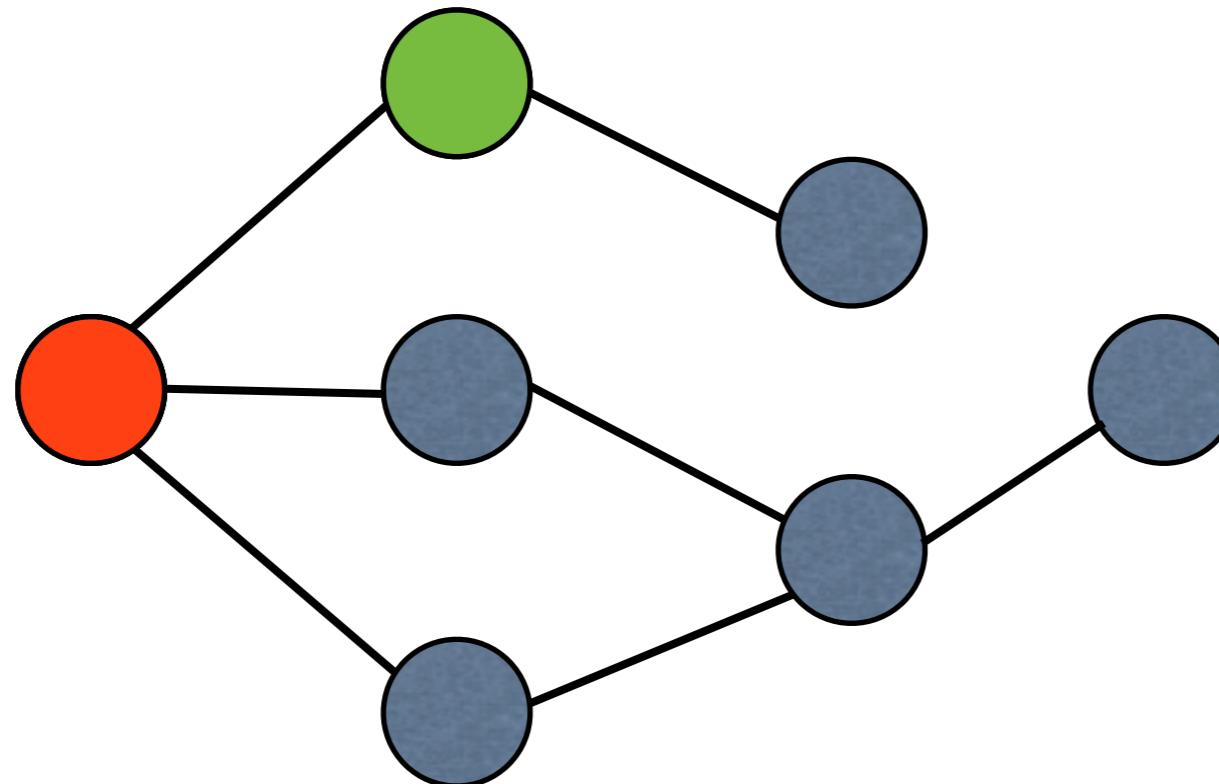
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



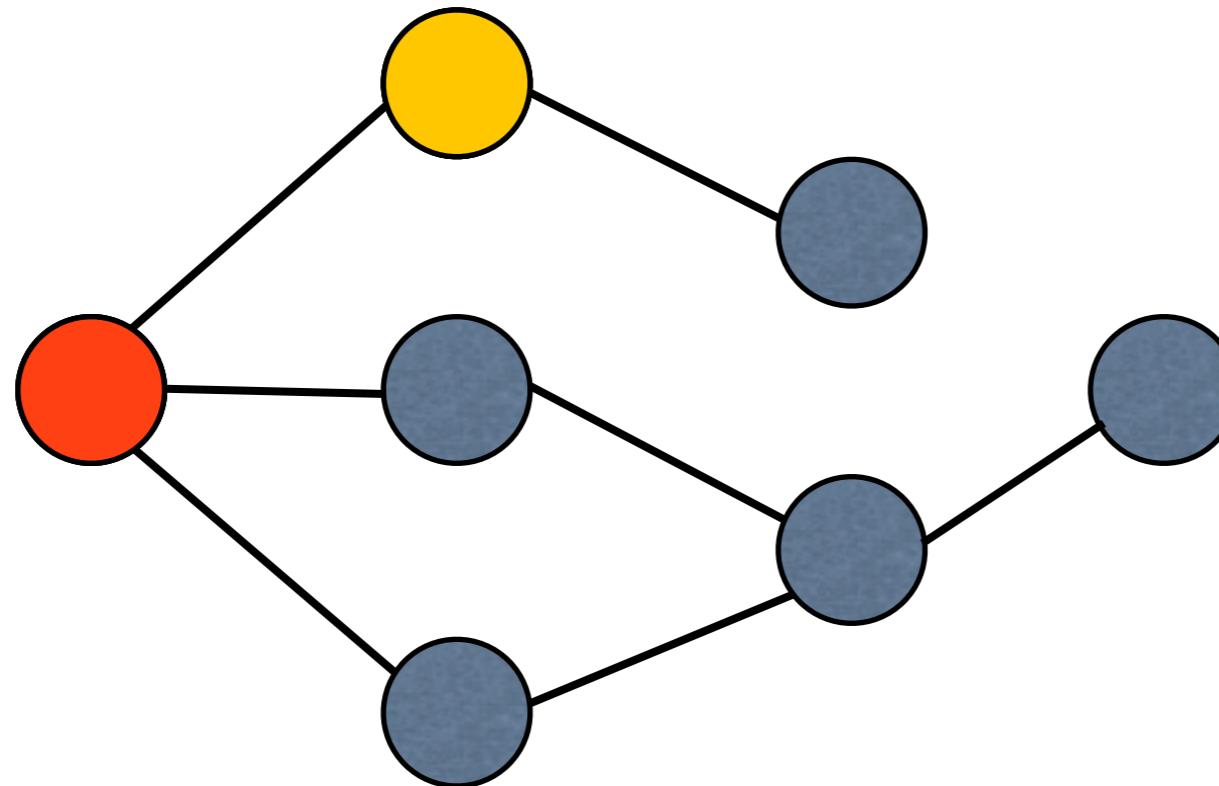
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



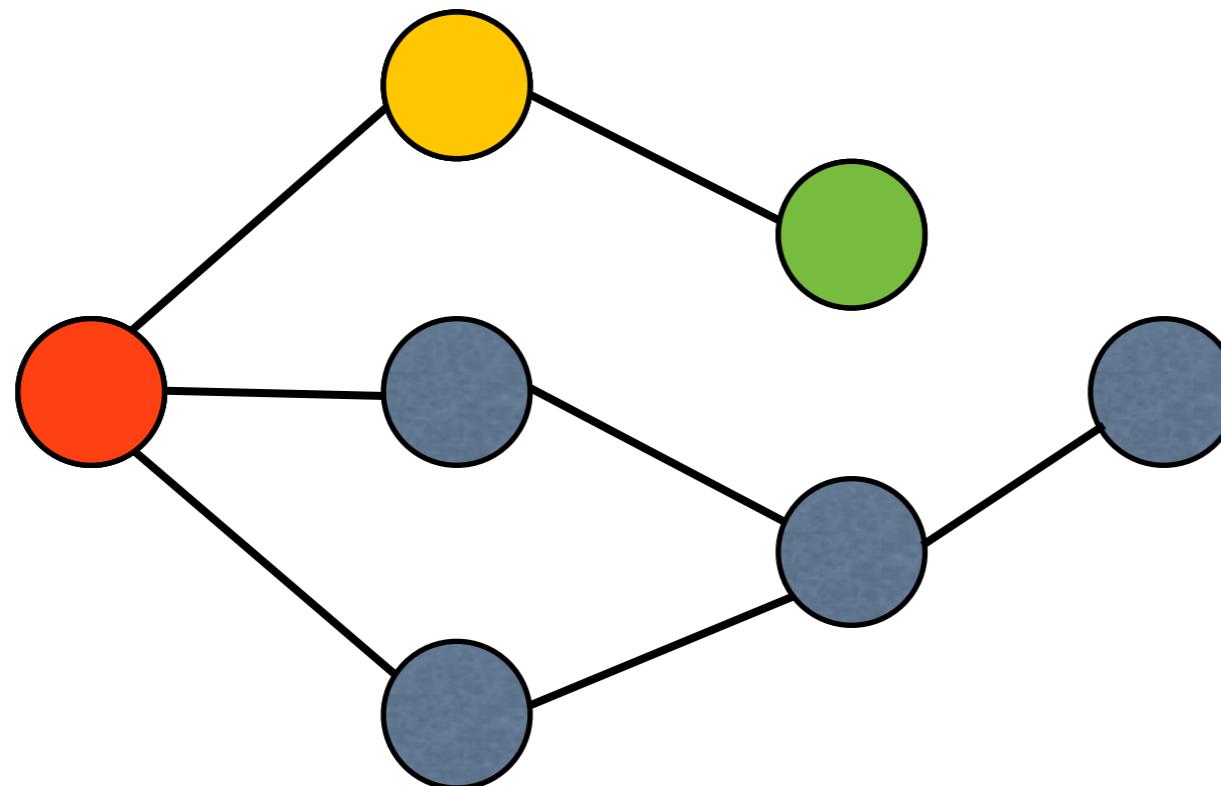
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



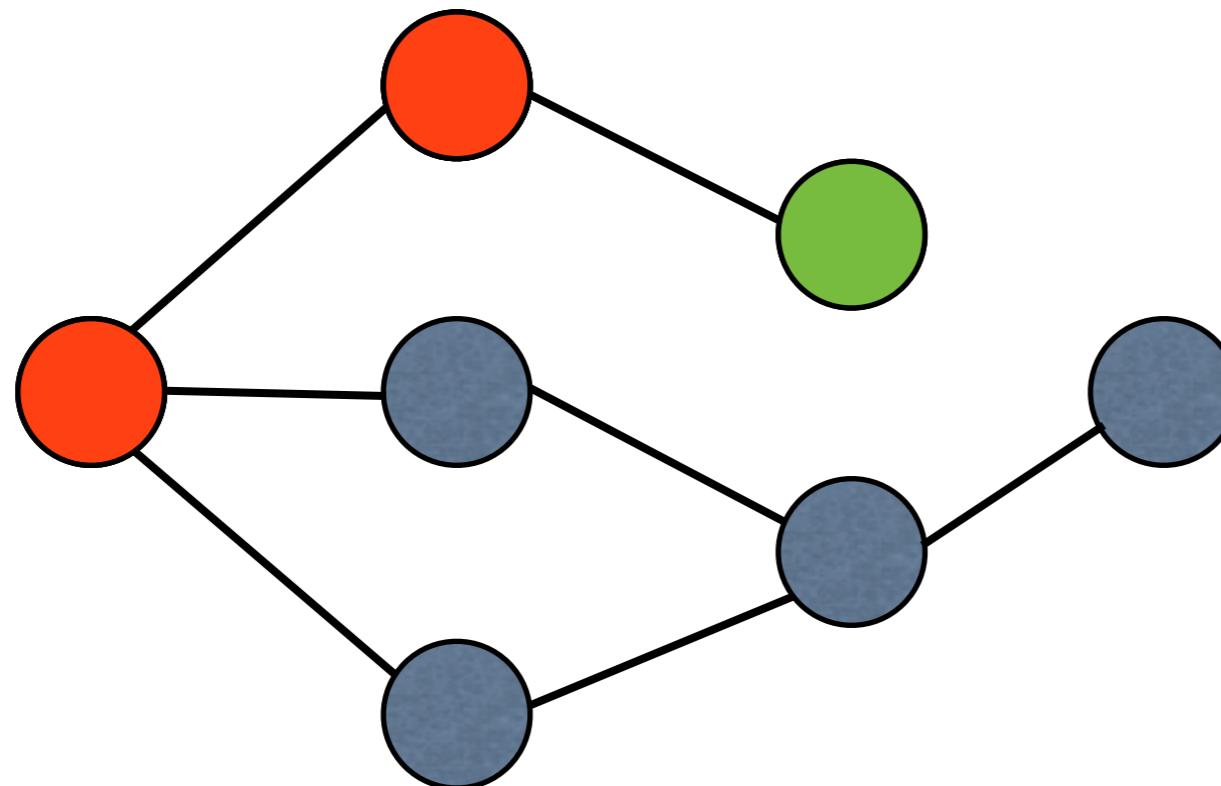
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



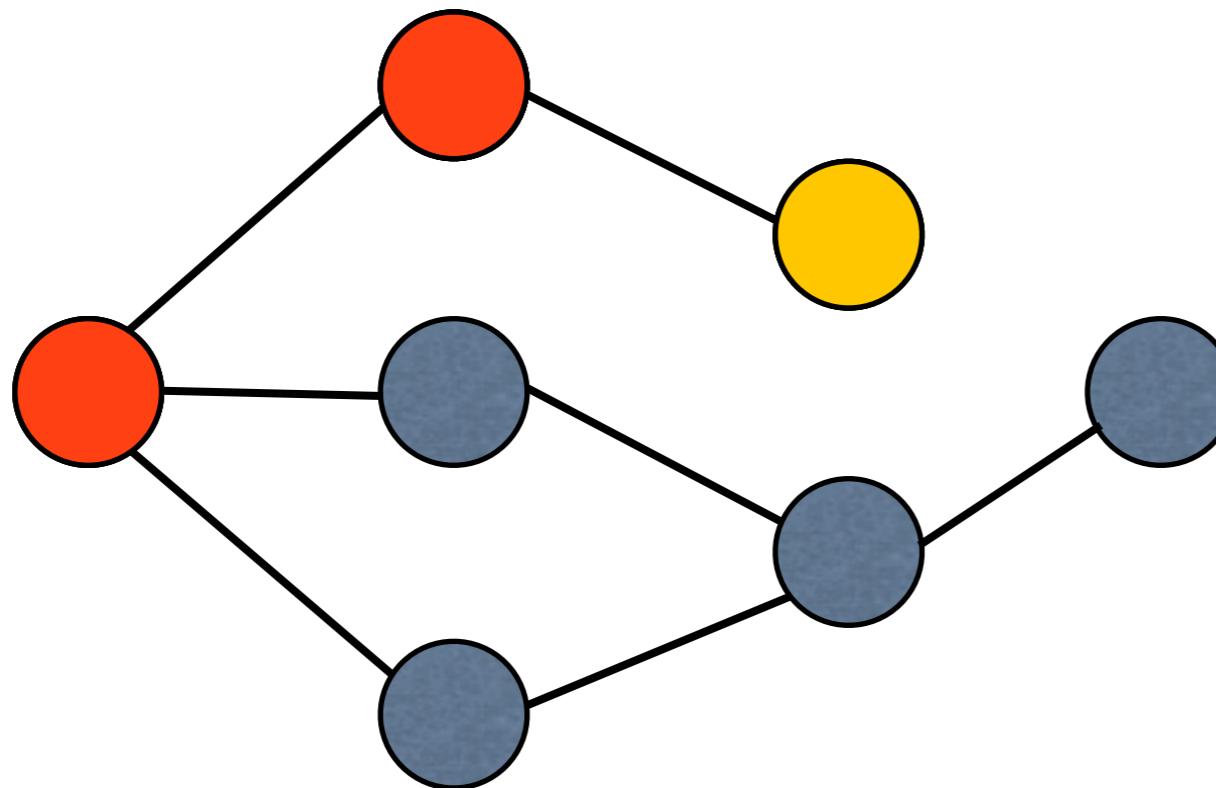
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



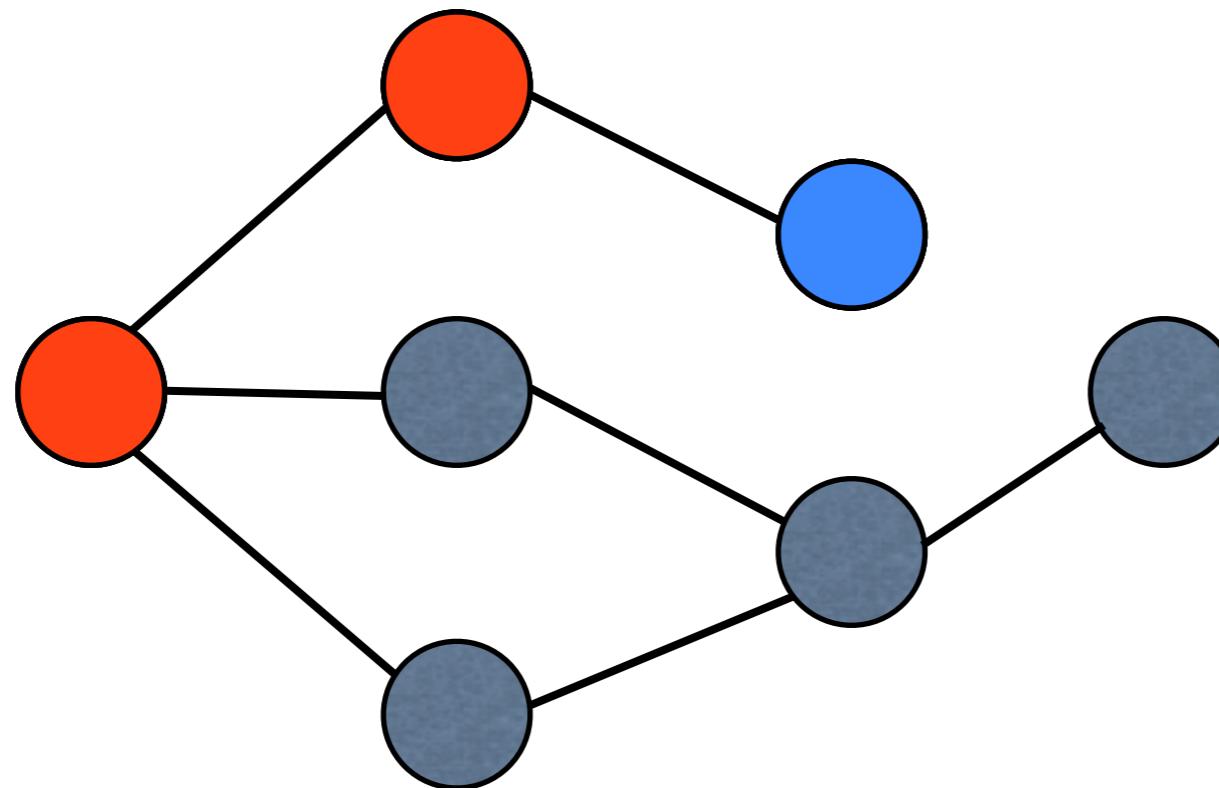
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



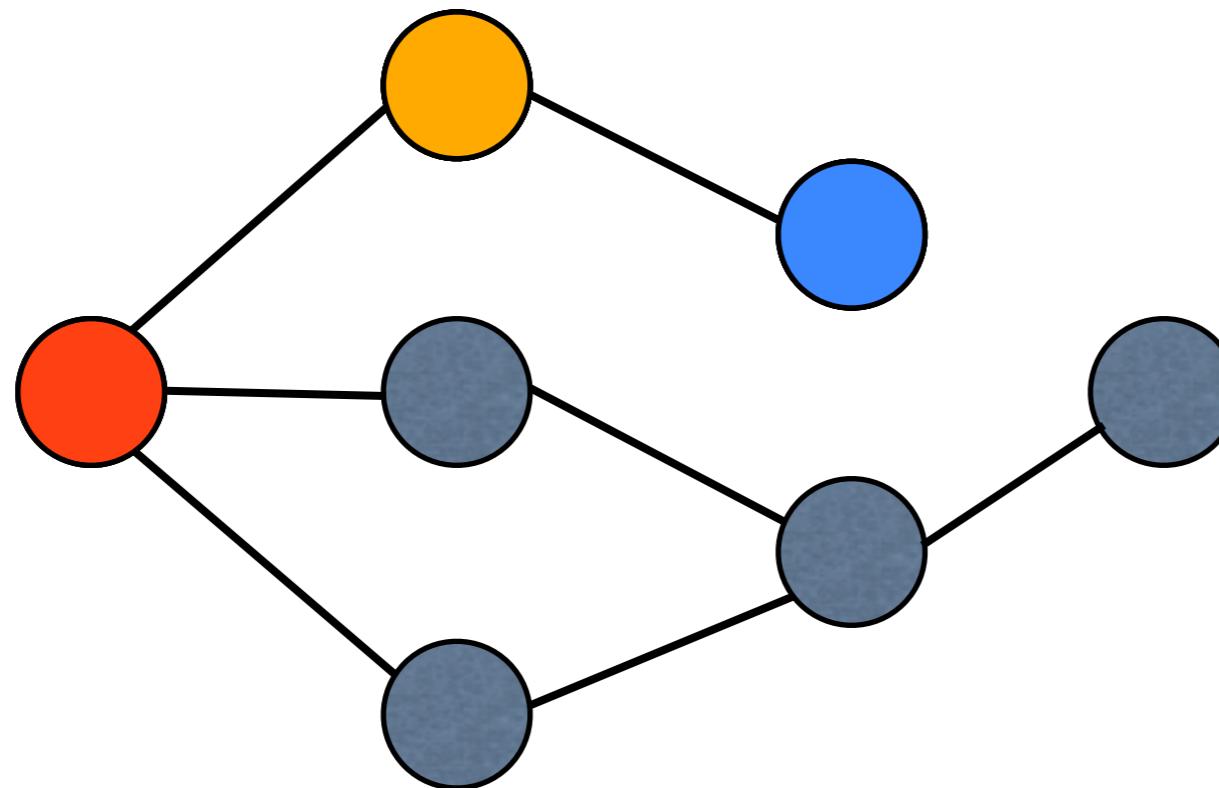
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



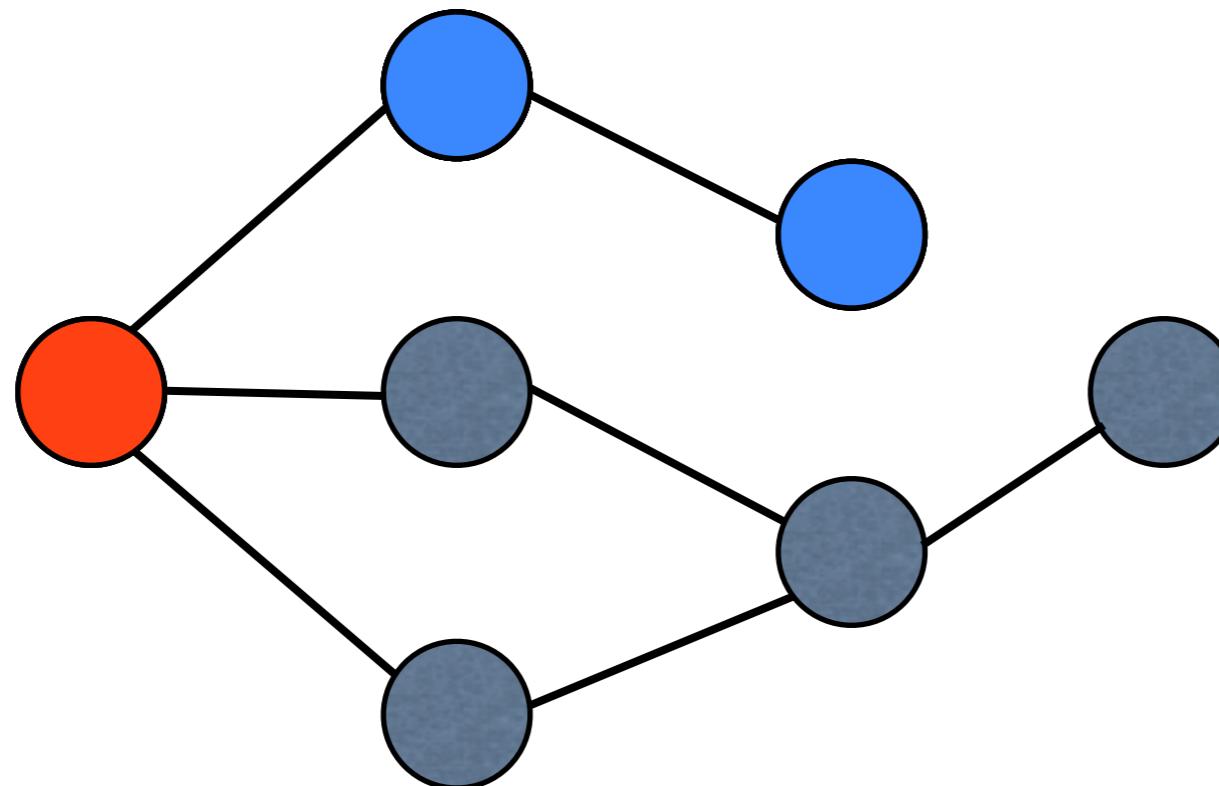
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



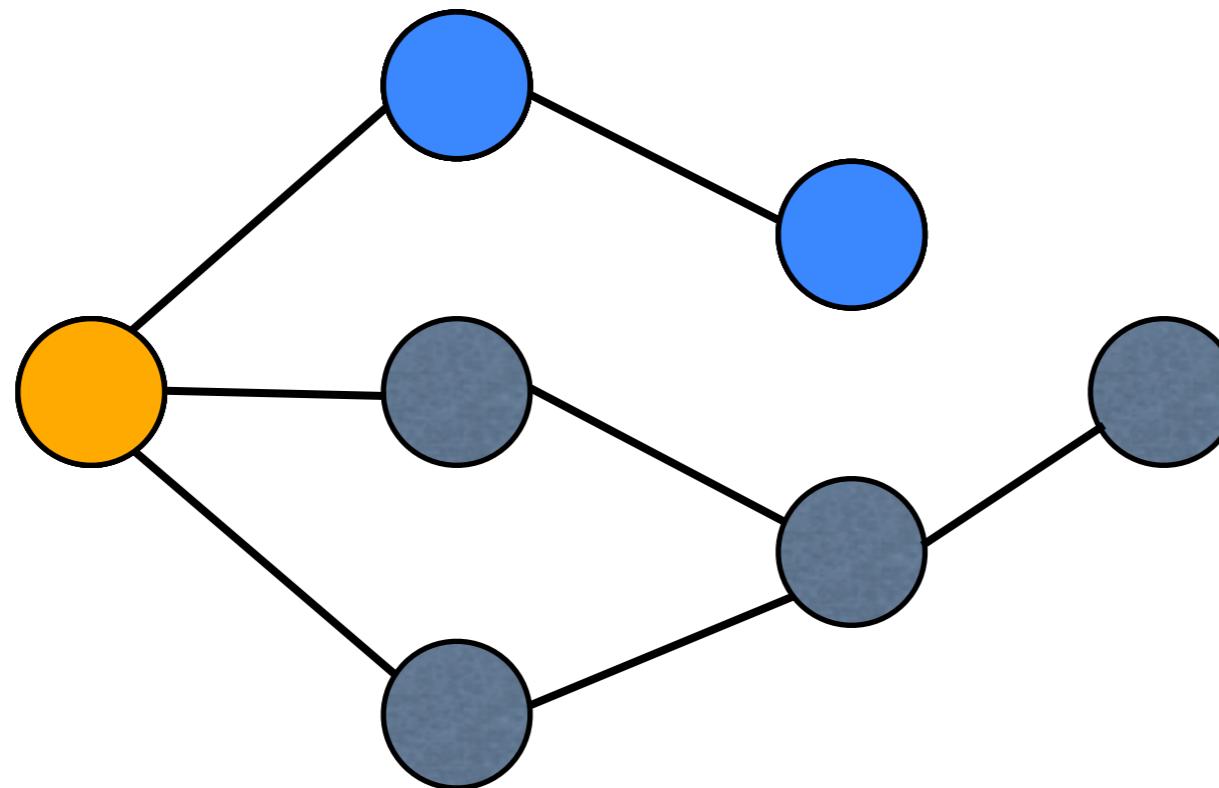
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



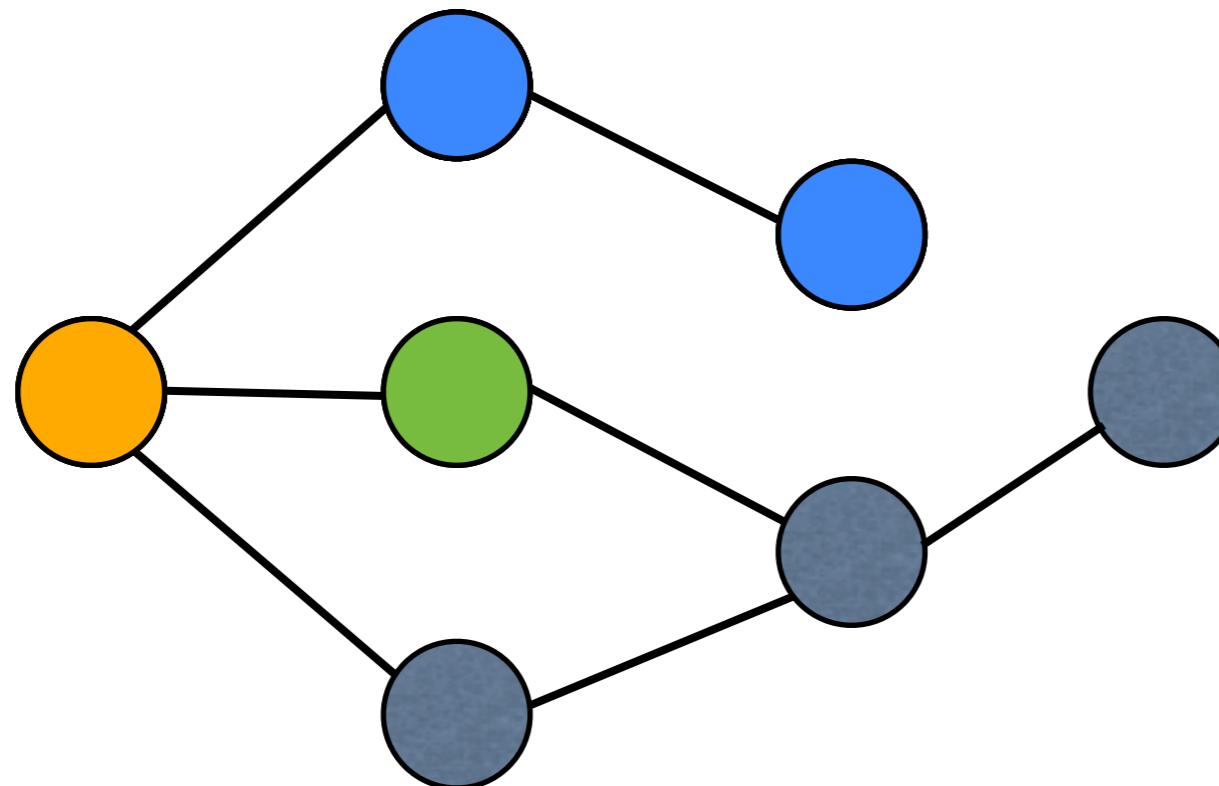
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



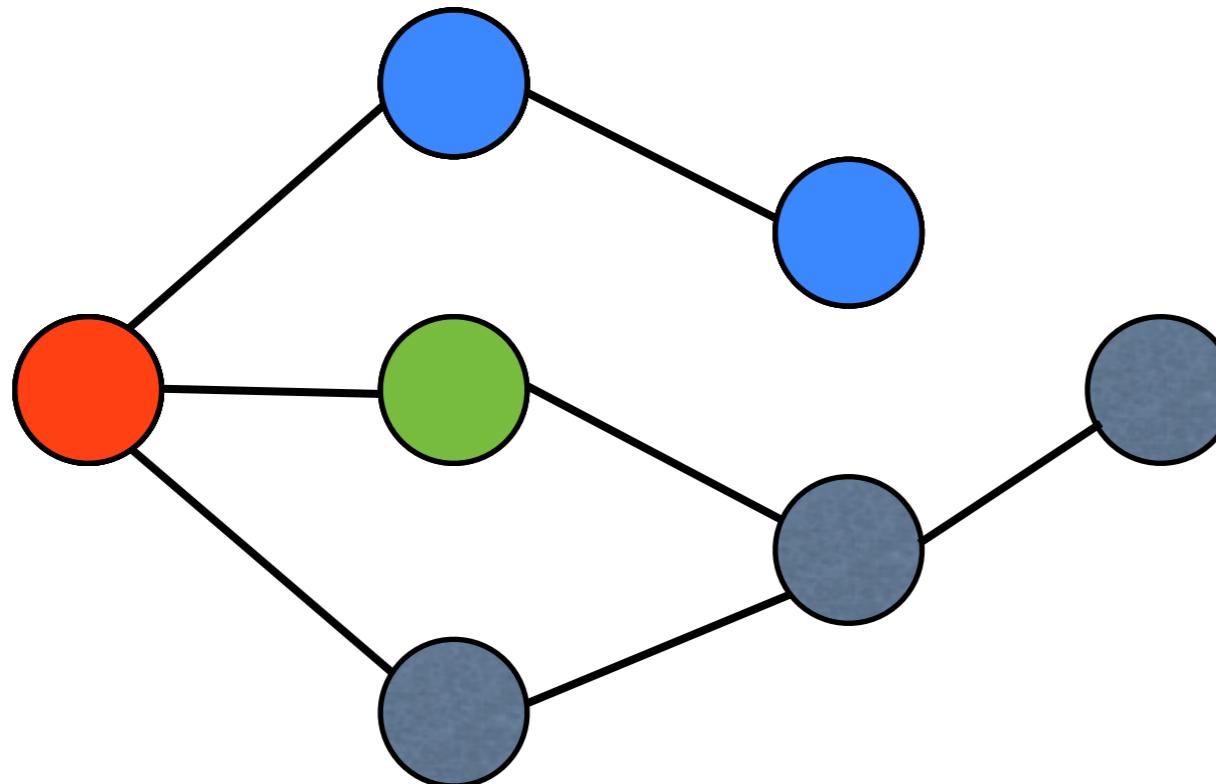
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



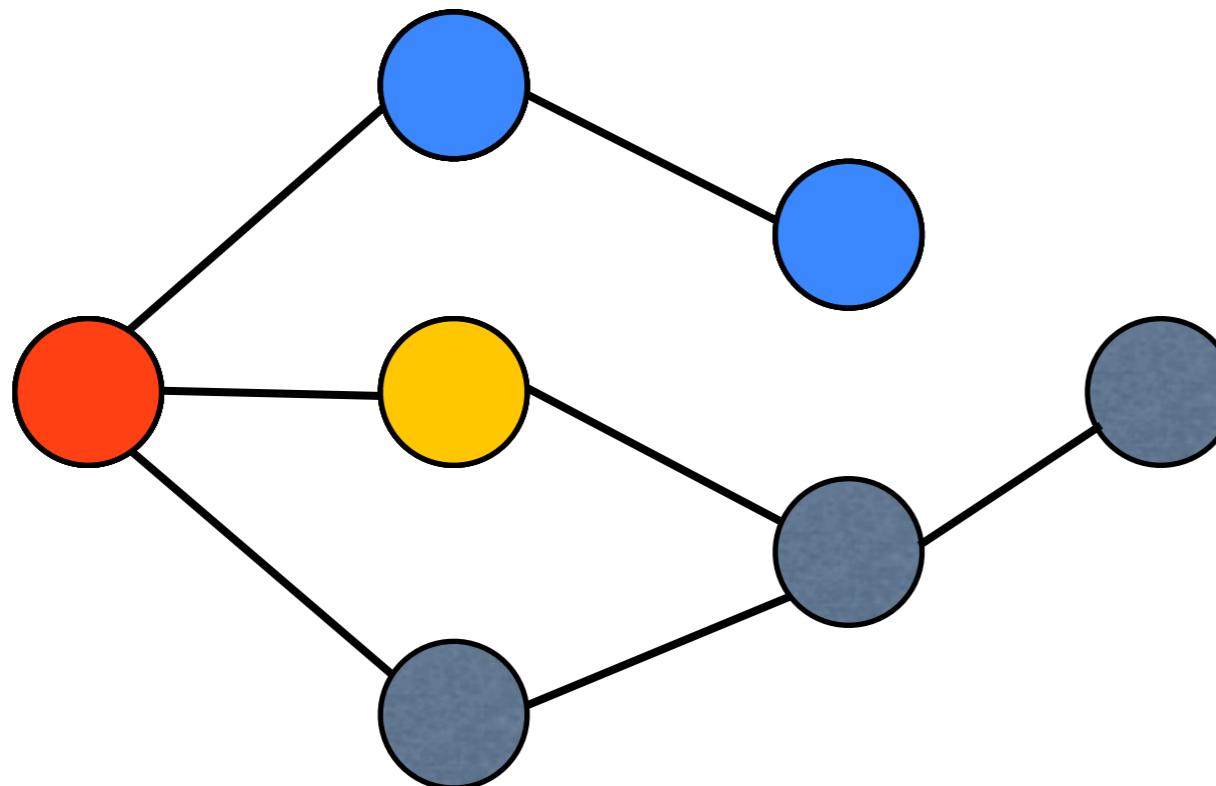
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



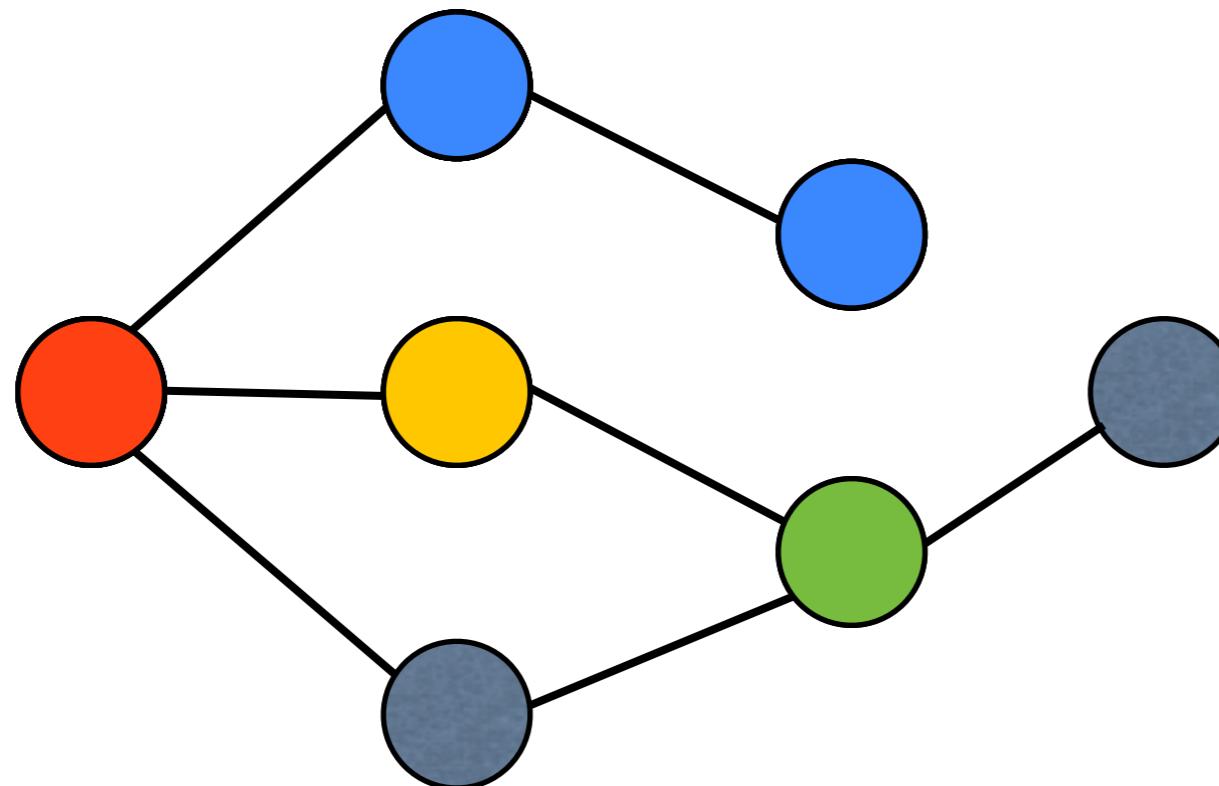
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



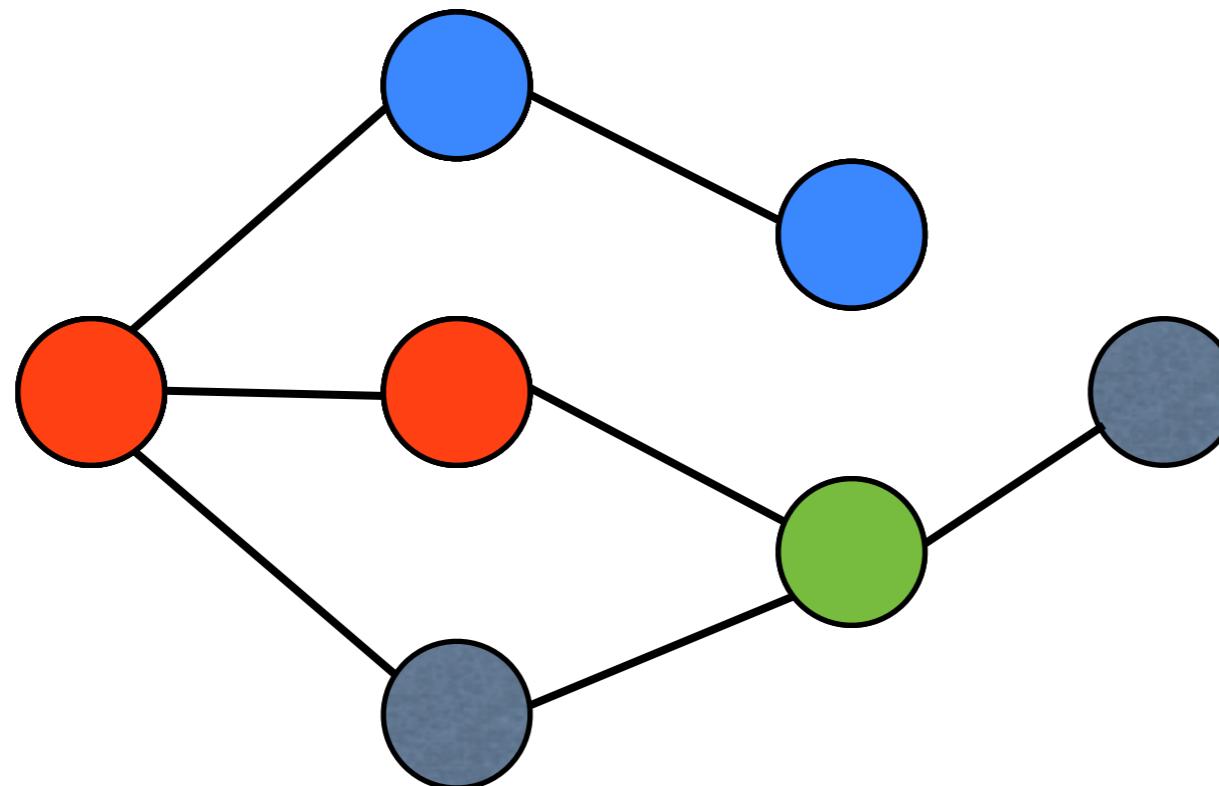
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



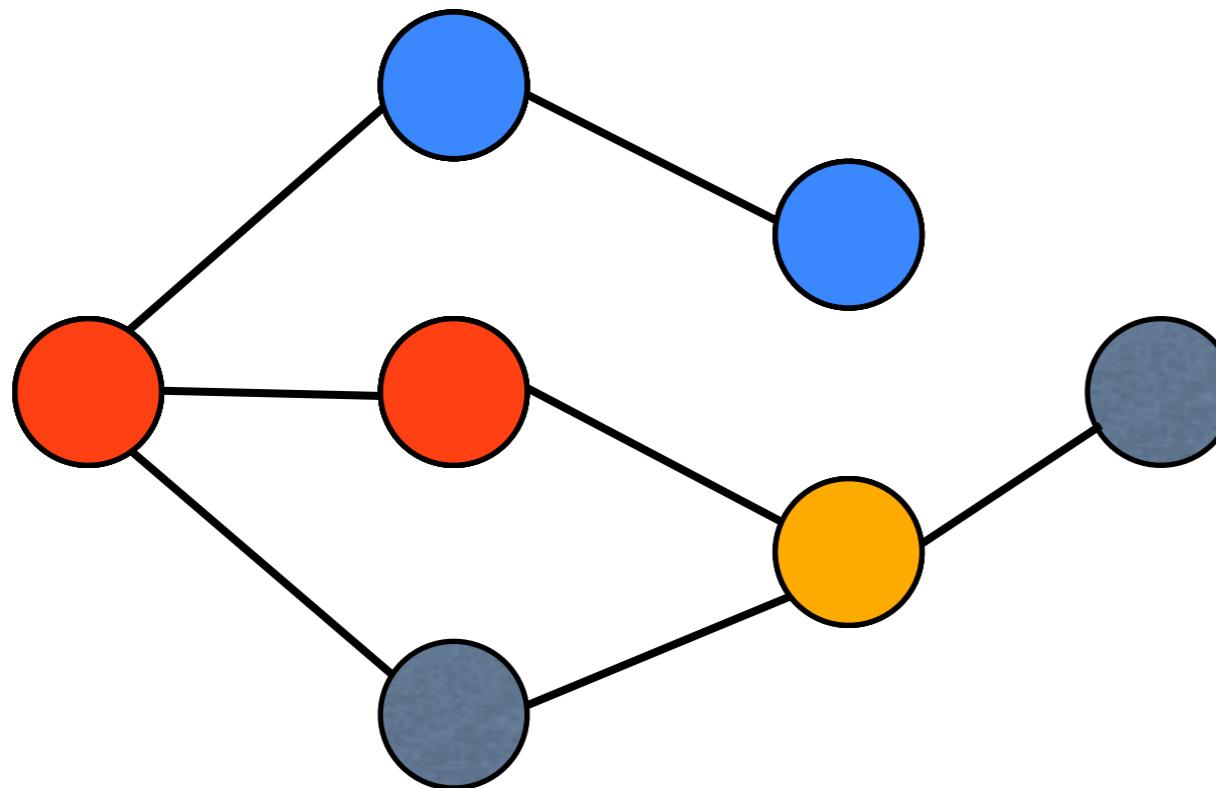
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



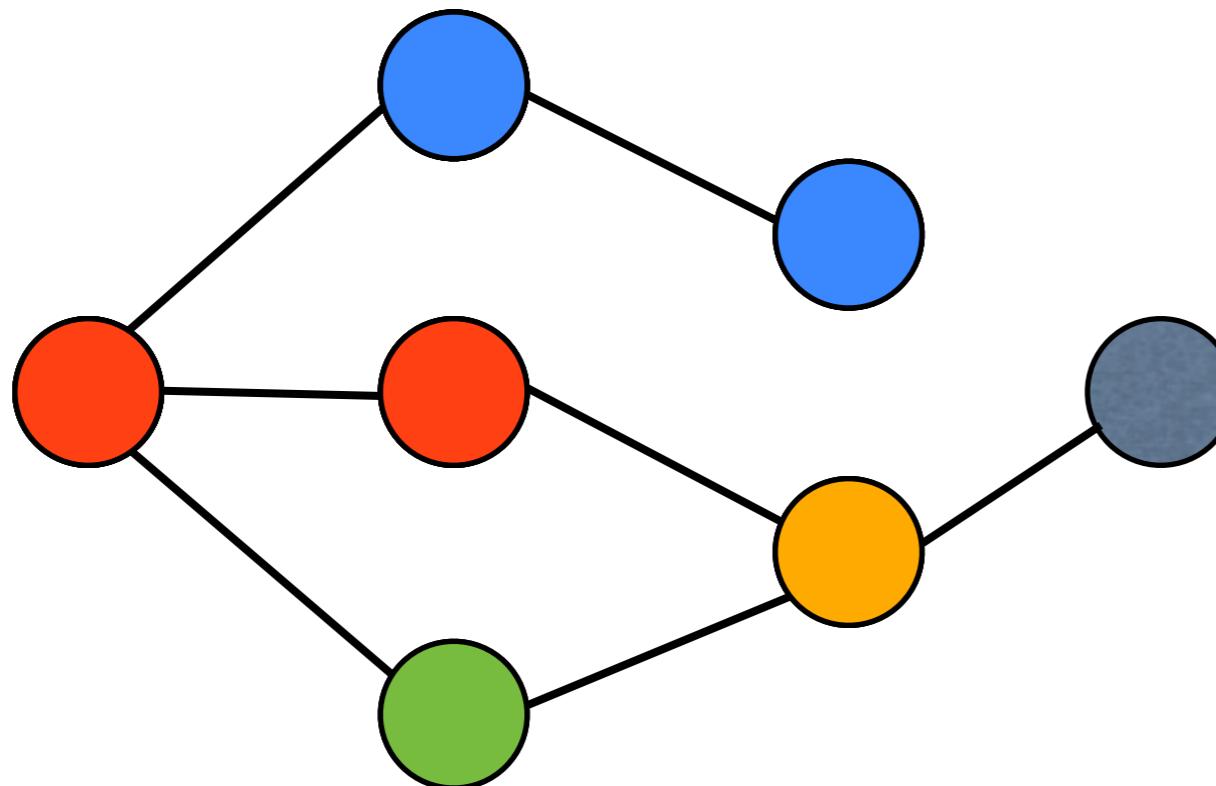
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



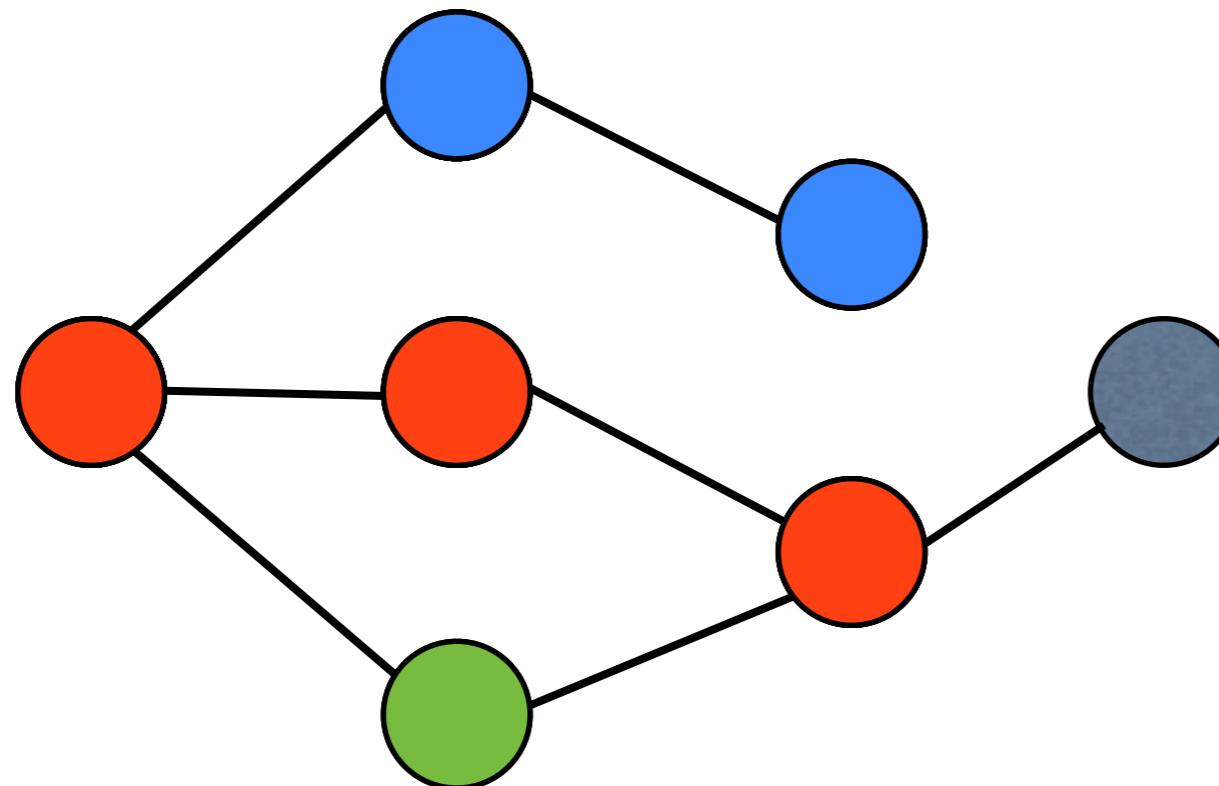
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



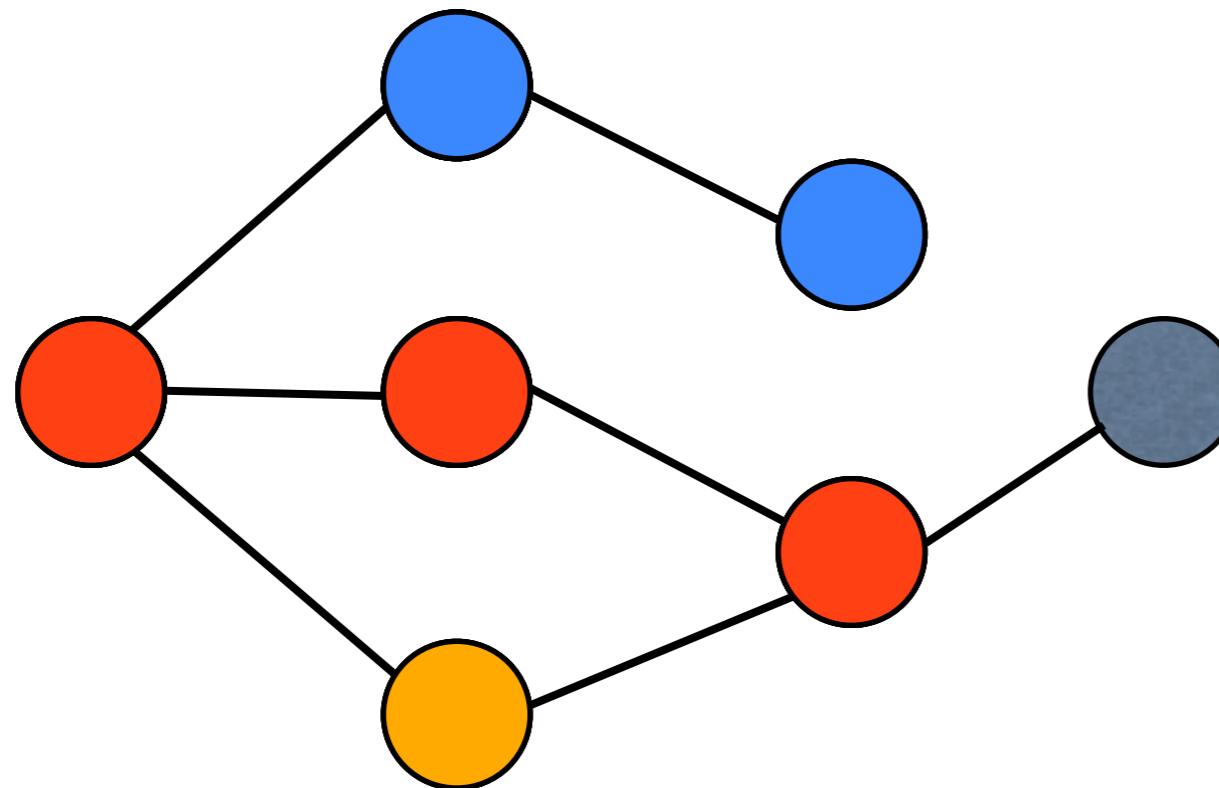
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



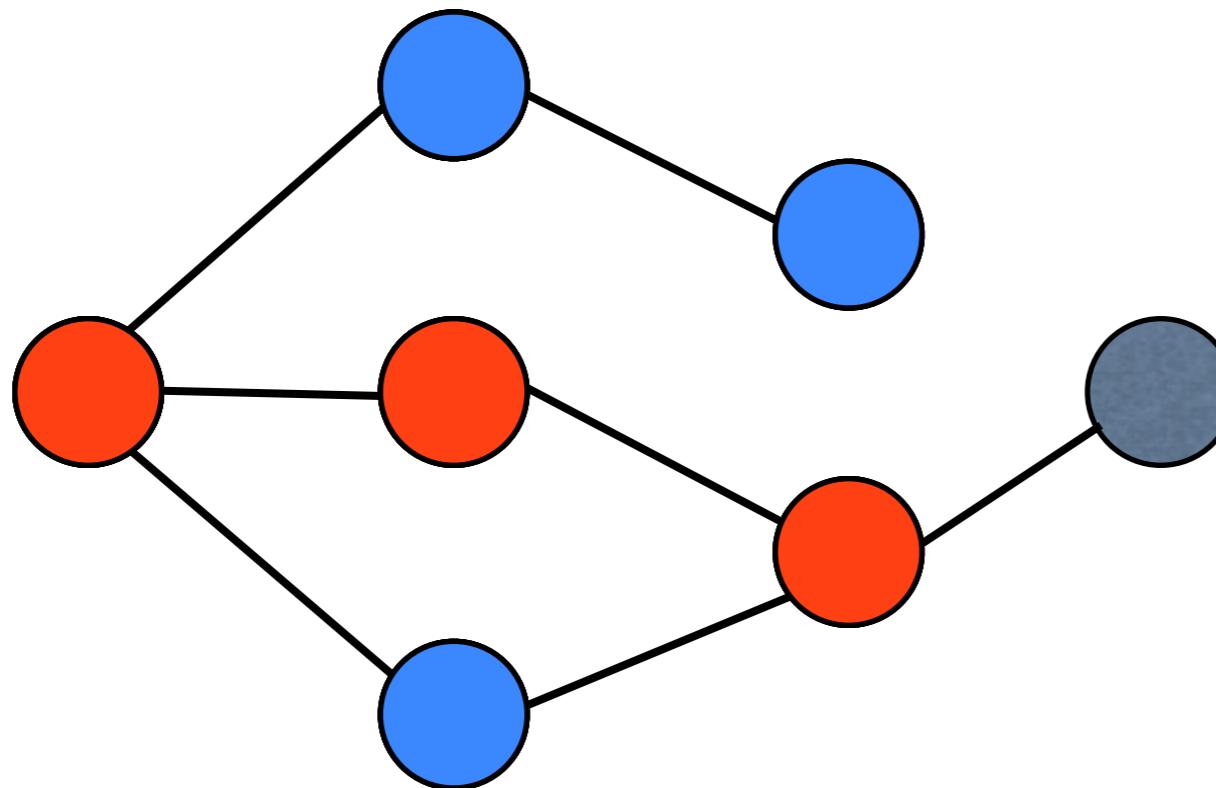
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



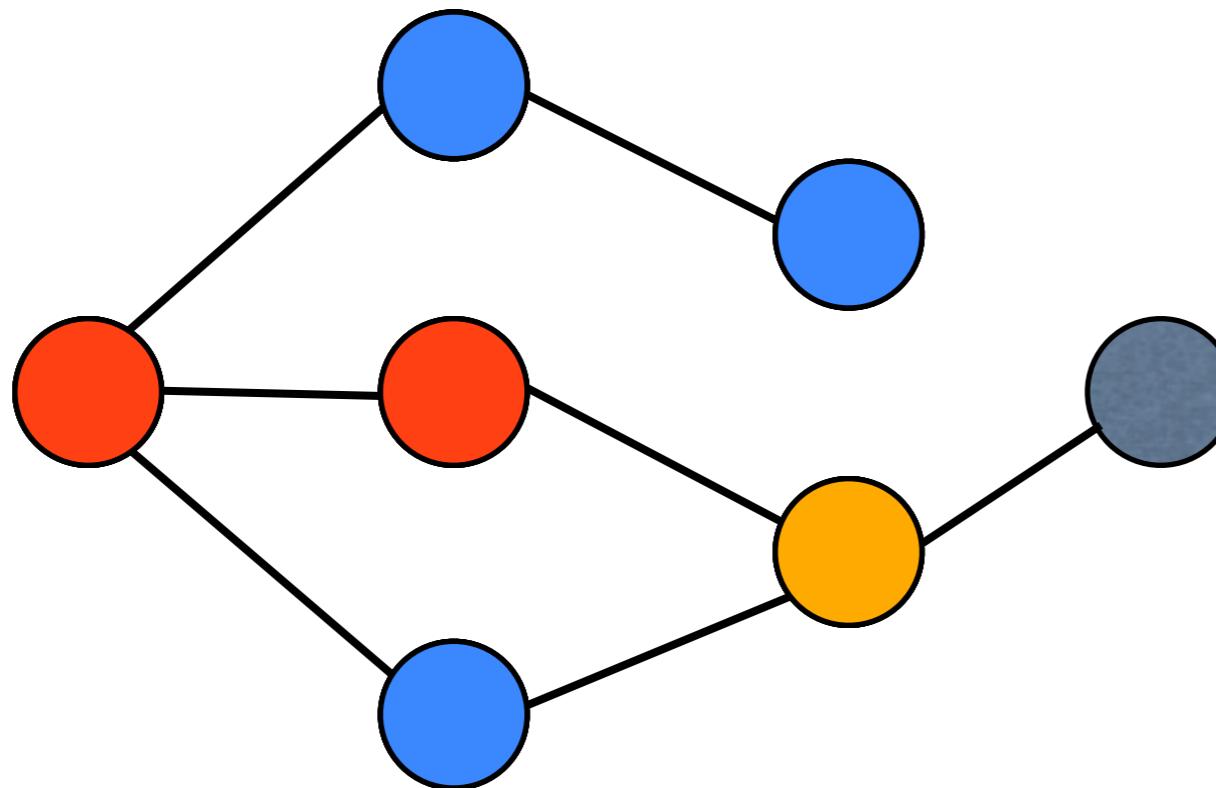
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



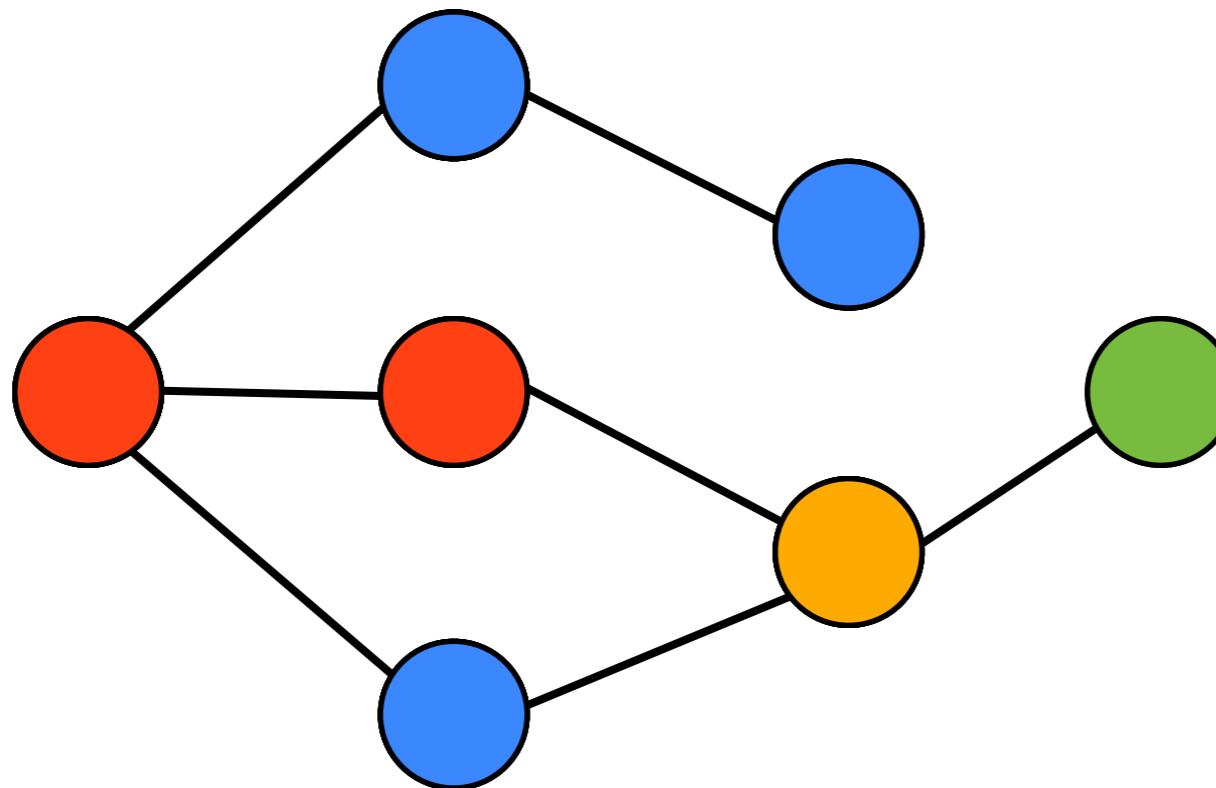
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



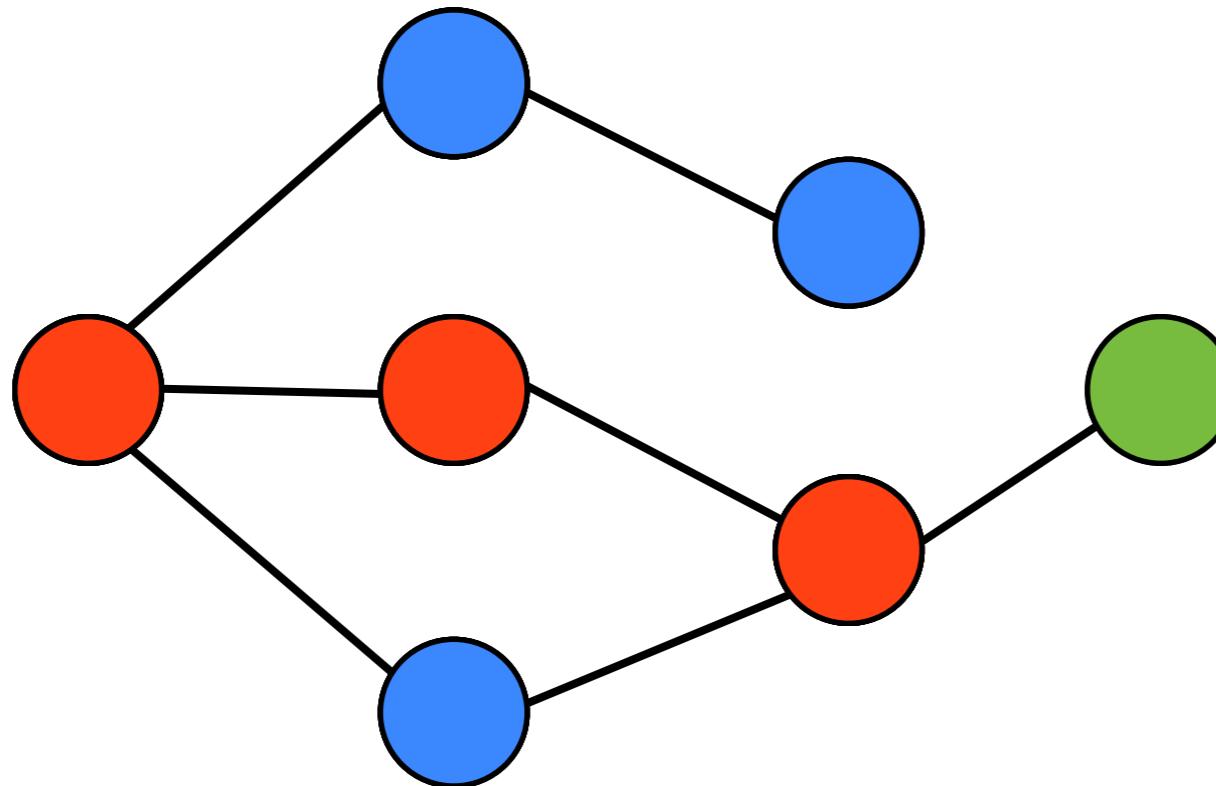
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



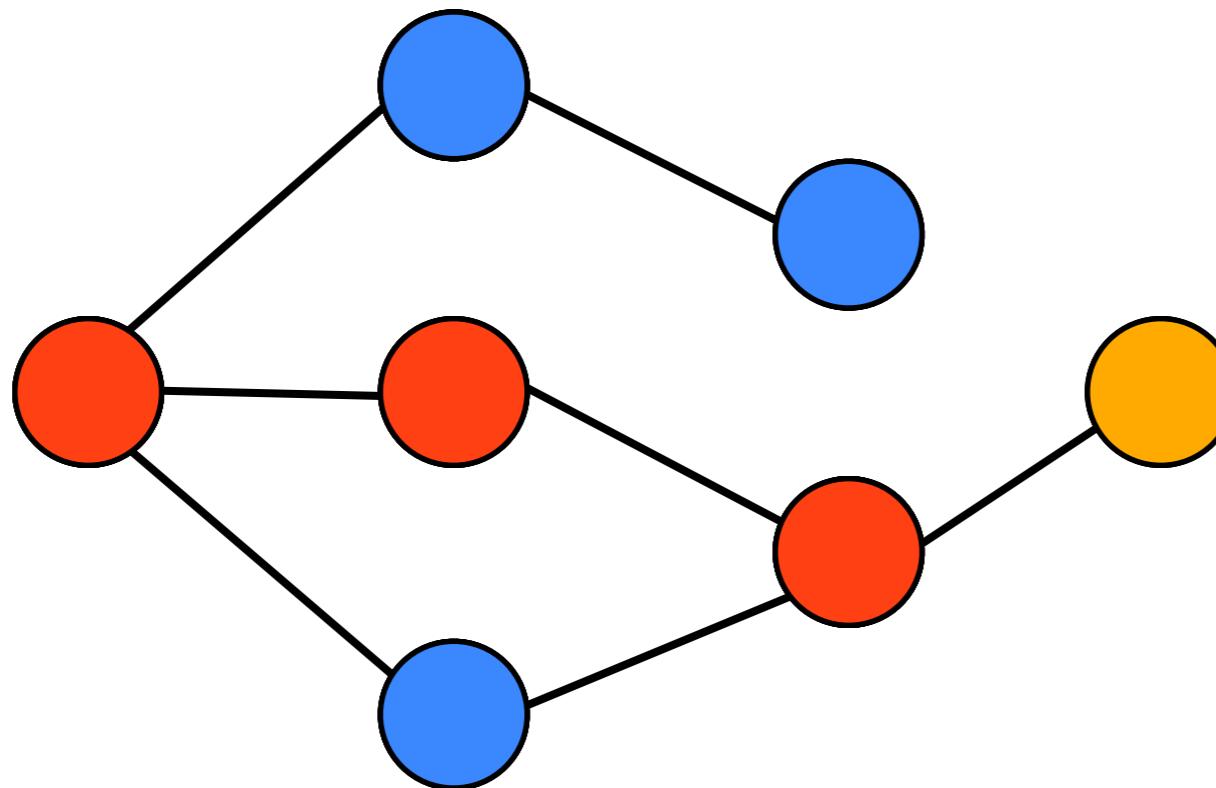
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



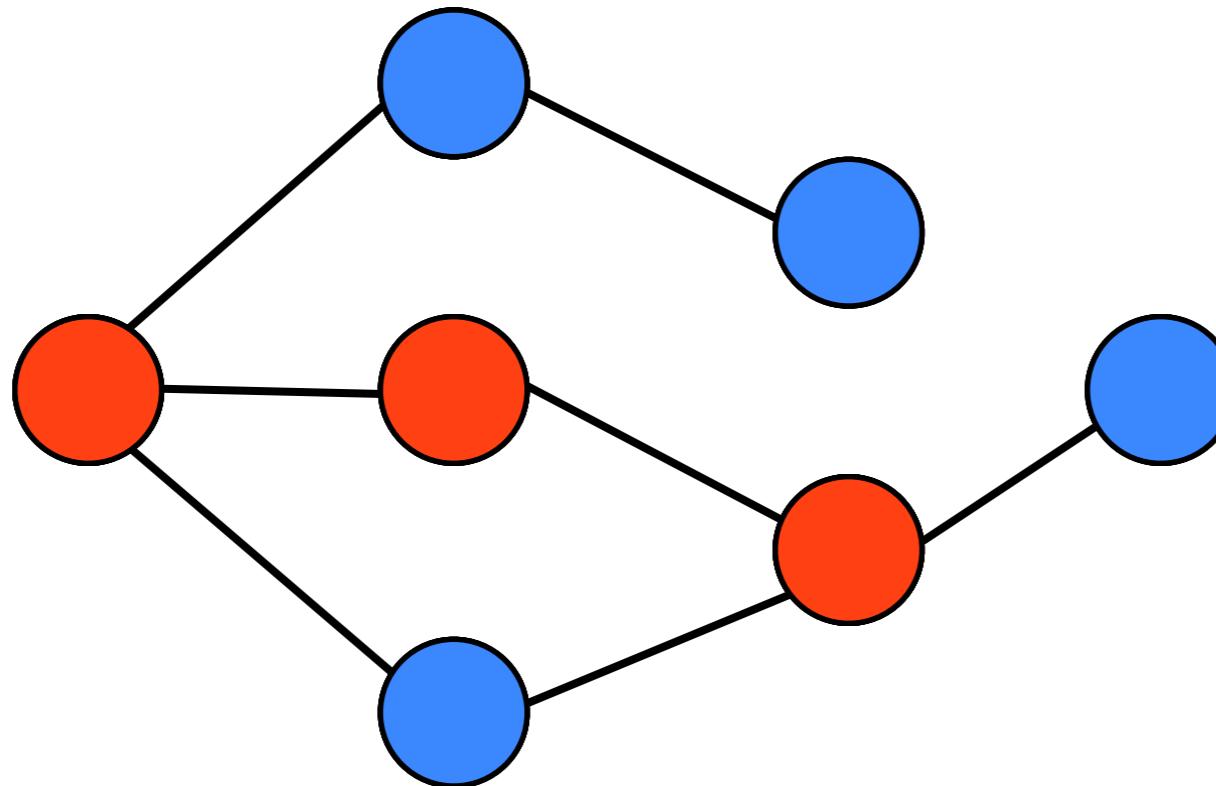
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



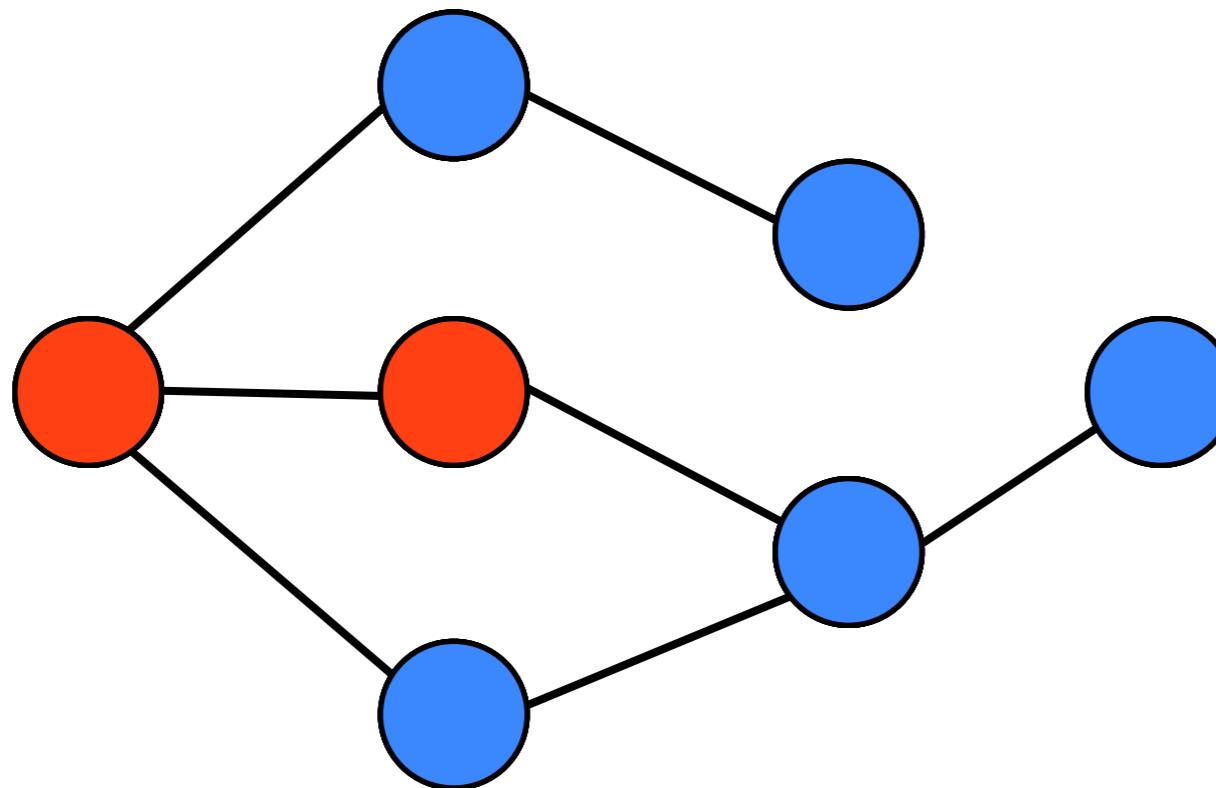
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



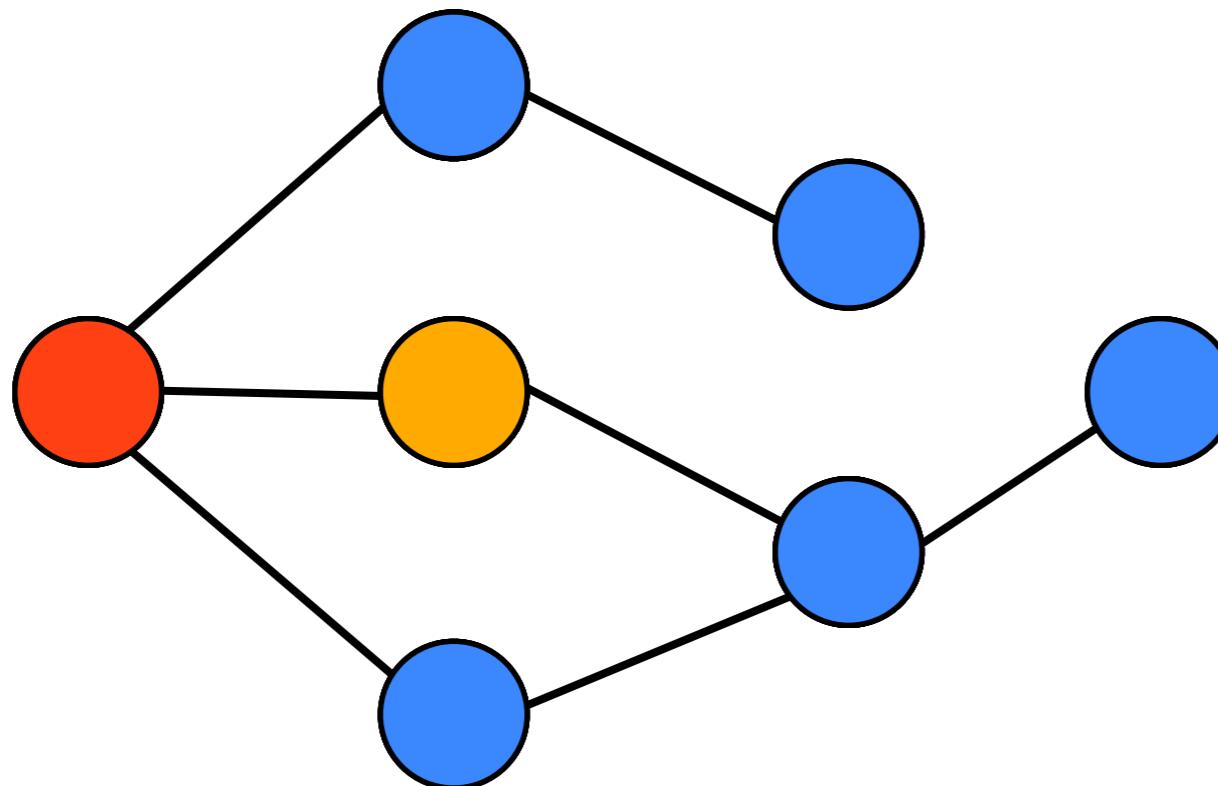
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



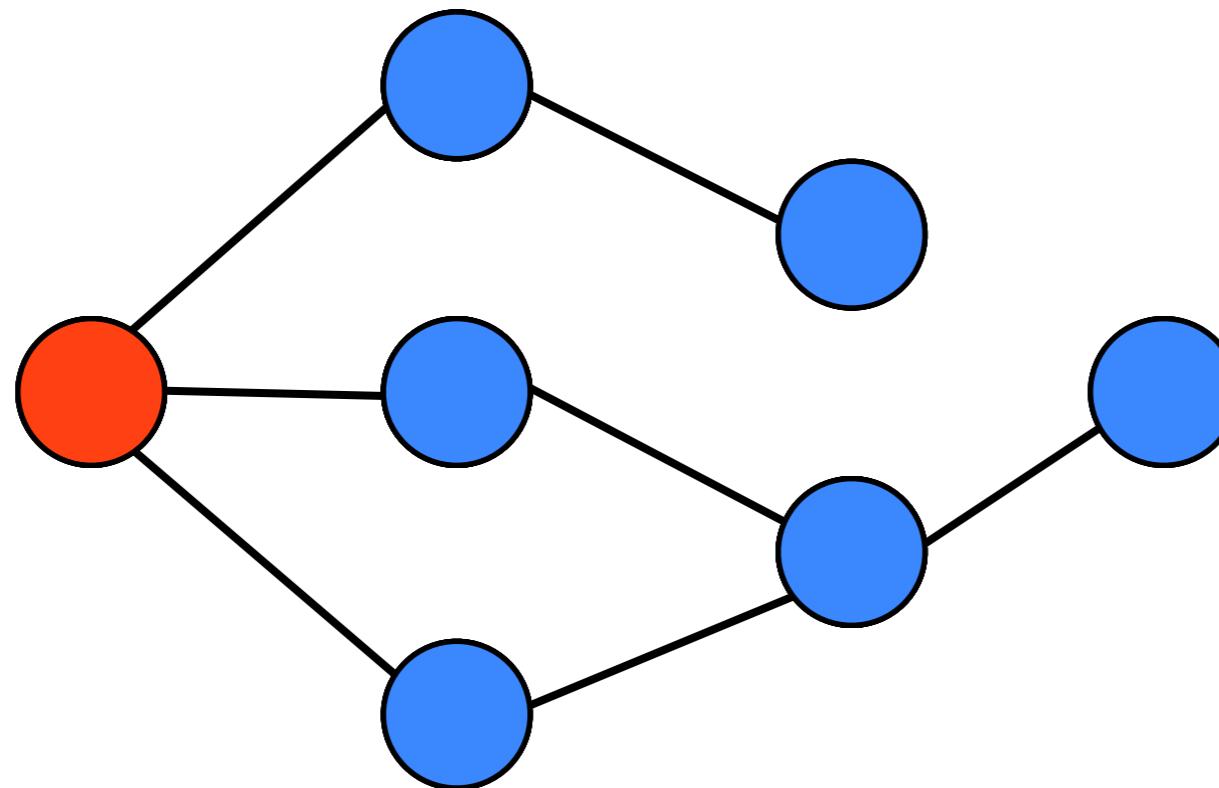
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



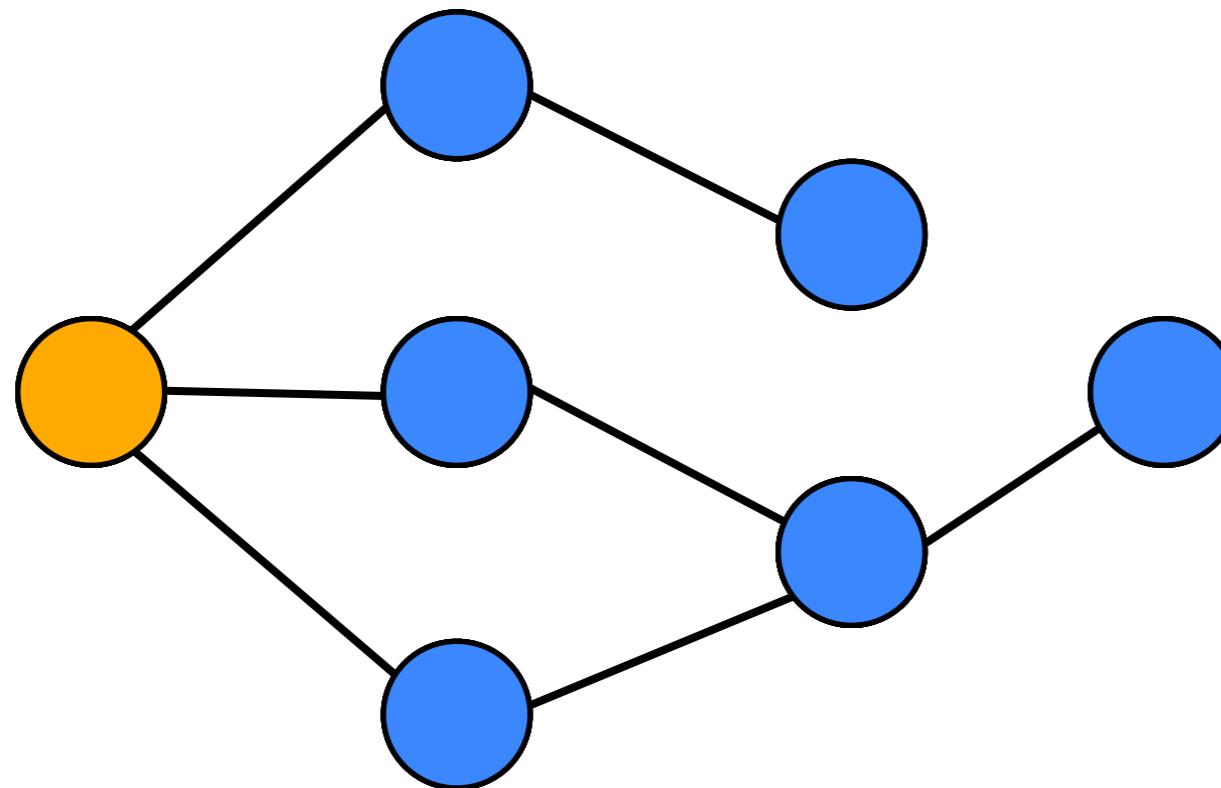
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



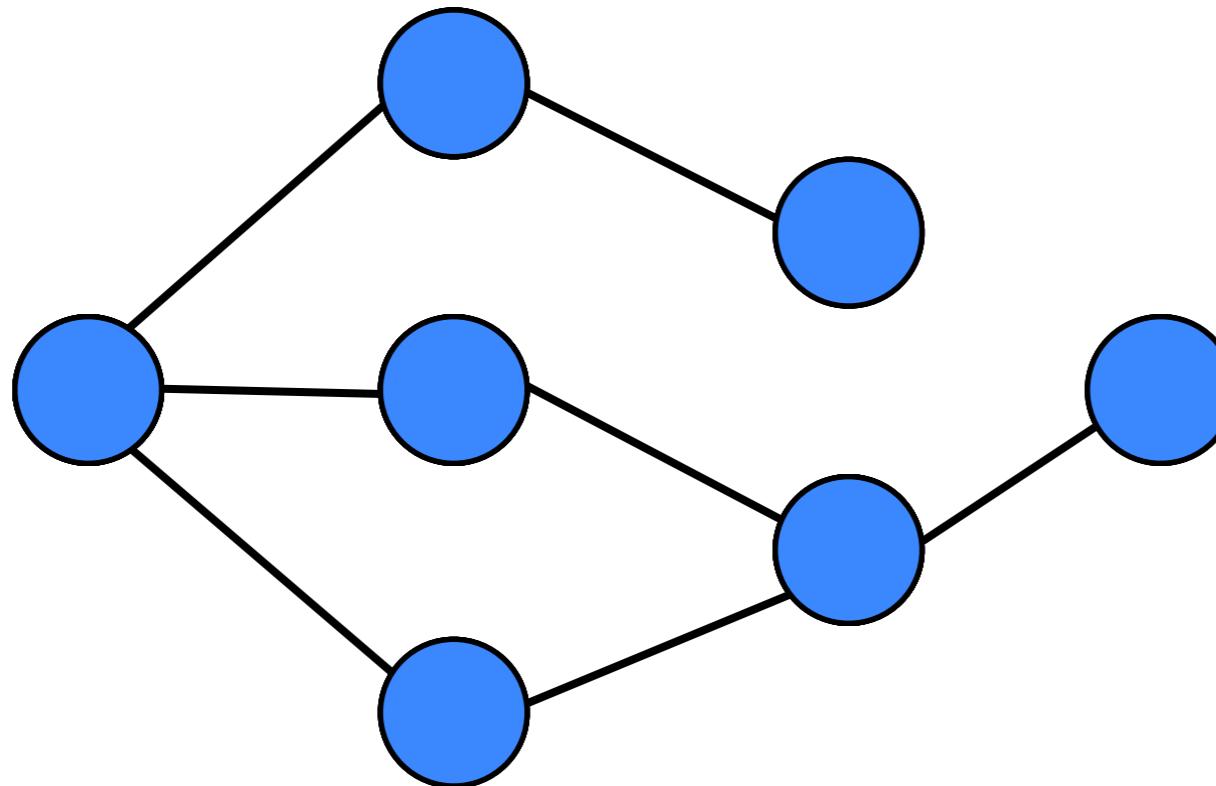
Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):



Algoritmos de busca em grafos - Busca em profundidade

- Explora primeiro o último vértice colocado na lista (funciona como uma pilha):

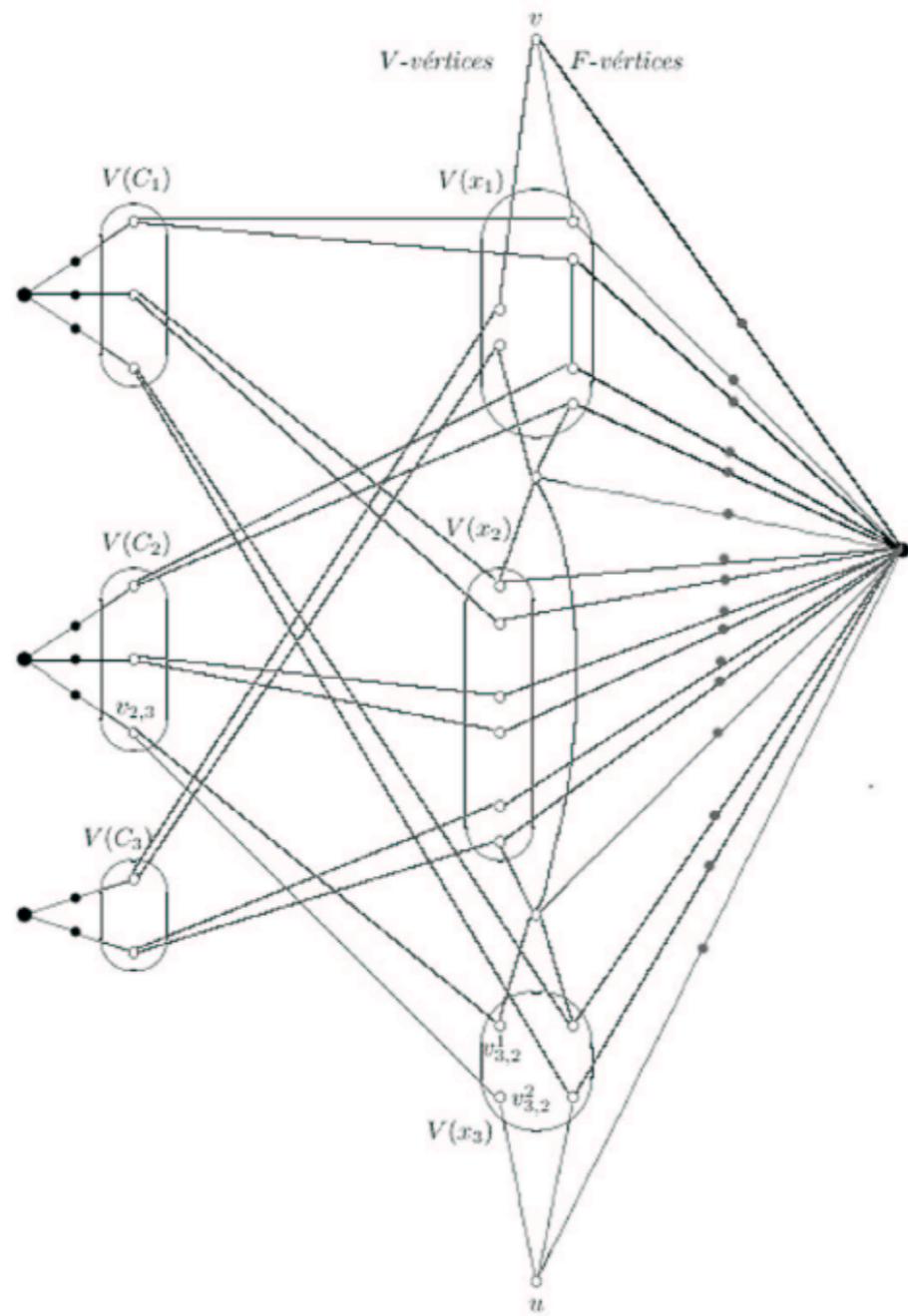


Implementação: Busca em profundidade

```
// Declarando
const int MAX_VERTICES = 100;
int grafo[MAX_VERTICES][MAX_VERTICES];
bool visitado[MAX_VERTICES];
int num_vertices; // qtd de vértices do grafo atual

// Recursivamente visita todos os vértices
// alcançáveis a partir de atual
void dfs(int atual) {
    for (int i = 0; i < num_vertices; ++i) {
        if (grafo[atual][i] && !visitado[i]) {
            visitado[i] = true;
            dfs(i);
        }
    }
}
```

Grafos: Aplicações



Aplicações

- Verificar se um grafo é bicolorível
- Detecção de Ciclos
- Ordenação Topológica
- Caminhos Mínimos (Grafos não ponderados)
- Conectividade

Aplicações: Bicoloração

- Em algumas espécies é difícil distinguir os machos das fêmeas, como é o caso de uma espécie recém descoberta, os happyRockers.
- Num laboratório são realizadas experiências para determinar potenciais machos e fêmeas em uma amostra.

Aplicações: Bicoloração

- São feitas anotações sempre que dois indivíduos se “relacionam”.
- A espécie não é monogâmica, portanto um indivíduo A pode tranquilamente ter relações com os indivíduo B e C.
- Porém, os sexos são bem definidos, machos apenas se relacionam com fêmeas e vice-versa. Então, se A se relaciona com B e C, estes são do mesmo sexo, o qual é diferente do sexo de A.

Aplicações: Bicoloração

- Dadas as anotações sobre as interações entre os indivíduos, o Dr. que coordena o laboratório quer saber se é possível distinguir quais indivíduos tem o mesmo sexo ou se existem indivíduos suspeitos no grupo.
- Como resolver ?

Aplicações: Bicoloração

- O problema pode ser modelado em um grafo não direcionado.
- Indivíduos são os vértices, relações são as arestas.
- Tá, e agora ?

Aplicações: Bicoloração

- Queremos saber se é possível separar os vértices em dois conjuntos, onde os vértices pertencentes a cada conjunto não se relacionaram durante o experimento.
- Ah, que legal, e como faço isso ?!?
- Basta verificar se é possível colorir os vértices do grafo usando apenas duas cores, ou seja, verificar se o grafo é bicolorível/bipartido.

Aplicações: Bicoloração

- E como fazer essa verificação ?
- Fácil ! Basta fazer uma busca em largura..
- Começamos um BFS de cada vértice ainda não colorido, pintando ele de branco e adicionando na fila;
- Quando tira-se um vértice da fila e vai processar seus vizinhos, tenta pintar eles da cor contrária a que o vértice atual está pintado.

Aplicações: Bicoloração

```
#include <iostream>
#include <list>

using namespace std;

int main(){

    int u, v, num_individuos, num_relacoes;
    list< int >::iterator w;

    cin >> num_individuos >> num_relacoes;
    list< int > grafo[num_individuos+1];
    int cor[num_individuos+1];

    for( u = 1; u <= num_individuos; u++ )
        cor[u] = -1;
```

Aplicações: Bicoloração

```
while( num_relacoes-- ){
    cin >> u >> v;
    grafo[u].push_back( v );
    grafo[v].push_back( u );
}

bool bipartido = true;
for( int i = 1; i <= num_individuos && bipartido; i++ ){
    if( cor[i] != -1 ) continue;
    cor[i] = 1;
    list< int > fila;
    fila.push_back( i );
```

Aplicações: Bicoloração

```
while( !fila.empty() && bipartido ){
    u = fila.front(); fila.pop_front();
    for( w = grafo[u].begin(); w != grafo[u].end(); w++ ){
        if( cor[*w] == -1 ){
            cor[*w] = 1-cor[u]; // pinta o vizinho da cor contraria
            fila.push_back( *w ); // e adiciona na fila
        } else if( cor[*w] != 1-cor[*w] ) {
            // O vizinho estava pintado da mesma cor, grafo nao e
            // bicolorivel.
            bipartido = false;
            break;
        }
    }
}
```

Aplicações: Bicoloração

```
}
```

```
if( bipartido ) cout << "Possivel distinguir os sexos\n";
else cout << "Existem individuos suspeitos no grupo\n";
```

```
return 0;
}
```

Aplicações: Detecção de Ciclos

- Sabemos que existem grafos cíclicos e acíclicos.
- Mas dado um grafo, como seria uma algoritmo que verifica se ele é cíclico ou acíclico ?

Aplicações: Detecção de Ciclos

- Durante um DFS, podemos classificar as arestas do grafo em 3 tipos, são eles:
 - **arestas descendentes** - ligam ancestrais a descendentes
 - **arestas de retorno** - ligam descendentes a ancestrais
 - **arestas cruzadas** - ligam vértices sem relação ancestral/descendente

Aplicações: Detecção de Ciclos

- Um grafo tem um ciclo, se e somente se ele possui uma aresta de retorno.
- Portanto, se durante uma busca em profundidade detectarmos tal tipo de aresta, podemos garantir que o grafo é cíclico.

Aplicações: Detecção de Ciclos

- A idéia apesar de muito parecida funciona de maneiras diferentes para grafos orientados e não orientados.
- No grafo não orientado, inicialmente todos estão pintados de branco, e a medida que são descobertos são pintados de preto.

Aplicações: Detecção de Ciclos

- Quando um vértice tem uma aresta para um vértice pintado de preto (que já foi descoberto), e esse não é seu pai, essa é uma aresta de retorno.
- Logo, o grafo tem um ciclo.

Aplicações: Detecção de Ciclos

- Nos grafos orientados, o fato de um vizinho já ter sido descoberto não significa que a aresta é de retorno;
- Isso só é verdade se o vértice ainda estiver sendo processado..
- Portanto, precisamos diferenciar entre ‘não-usado’, ‘descoberto’ e ‘processado’

Aplicações: Detecção de Ciclos

- Num grafo direcionado uma aresta de retorno é uma aresta que aponta para um vértice ‘descoberto’.
- Então.. vamos ao código !

Aplicações: Detecção de Ciclos

```
bool ciclico(){

    for( int u = 1; u <= num_vert; u++ )
        cor[u] = -1;
    for( int u = 1; u <= num_vert; u++ )
        if( cor[u] == -1 ){
            parent[u] = u; // necessario para grafos nao orientados
            if( dfsC( u ) ) return true;
        }
    return false;
}
```

Aplicações: Detecção de Ciclos

```
bool dfsC( int u ){
    // versao para grafos nao orientados
    cor[u] = 1;
    for( list<int>::iterator v = grafo[u].begin(); v != grafo[u].end(); v++ ){
        if( cor[*v] == -1 ){
            parent[*v] = u;
            if( dfsC( *v ) ) return true;
        }
        else if( *v != parent[u] ) return true;
    }
    return false;
}
```

Aplicações: Detecção de Ciclos

```
bool dfsC( int u ){
    // versao para grafos orientados
    cor[u] = 1; // vertice u foi descoberto
    for( list<int>::iterator v = grafo[u].begin(); v != grafo[u].end(); v++ ){
        if( cor[*v] == -1 ){
            if( dfsC( *v ) ) return true;
        }
        else if( cor[*v] == 1 ) return true;
    }
    cor[u] = 2; // vertice u foi processado
    return false;
}
```

Aplicações: Ordenação Topológica

- No curso de Ciência da Computação algumas disciplinas possuem pré-requisitos.
- Dadas as disciplinas que o aluno deve cursar e seus pré-requisitos como descobrir em qual ordem as matérias podem ser cursadas ?

Aplicações: Ordenação Topológica

- As disciplinas podem ser vistas como os vértices de um grafo;
- Existe uma aresta direcionada do vértice X para o vértice Y se a disciplina X é pré-requisito da disciplina Y.

Aplicações: Ordenação Topológica

- Uma ordenação topológica ordena os vértices em uma linha de modo que as arestas são todas direcionadas da esquerda pra direita.
- Logicamente, se o grafo é cíclico é impossível que tal ordenação exista.

Aplicações: Ordenação Topológica

- Mas como fazer uma ordenação topológica ?
 - Simples! Basta uma busca em profundidade para gerar uma ordenação topológica, ou retornar um erro caso o grafo seja cíclico.
- Como assim ?

Aplicações: Ordenação Topológica

- Na busca em profundidade depois que analisamos todas as arestas que saem de um vértice, podemos garantir que ele não tem nenhuma aresta de entrada vindo de um vértice que já foi processado.
- E se em algum momento detectamos a existência de um ciclo ?
 - Uma ordenação topológica não existe.

Aplicações: Ordenação Topológica

```
bool gera_ordenacao(){
    ordem.clear(); // pilha onde a ordenacao sera armazenada
    for( int u = 1; u <= num_vert; u++ )
        cor[u] = -1;
    for( int u = 1; u <= num_vert; u++ )
        if( cor[u] == -1 ){
            if( dfsT( u ) ) return false;
        }
    while( !ordem.empty() ){
        int u = ordem.top(); ordem.pop();
        cout << u << " ";
    }
    return true;
}
```

Aplicações: Ordenação Topológica

```
bool dfsT( int u ){

    cor[u] = 1; // vertice u foi descoberto
    for( list<int>::iterator v = grafo[u].begin(); v != grafo[u].end(); v++ ){
        if( cor[*v] == -1 ){
            if( dfsC( *v ) ) return true;
        }
        else if( cor[*v] == 1 ) return true;
    }
    cor[u] = 2; // vertice u foi processado
    pilha.push( u );
    return false;
}
```

Aplicações: Caminhos Mínimos

- Num grafo não ponderado, quando queremos encontrar o menor caminho do vértice **s** ao vértice **t**, precisamos apenas de uma busca em largura.

Aplicações: Caminhos Mínimos

- Por que ?
 - Se as arestas não tem pesos diferentes, o custo do menor caminho de **s** para **t** é na verdade, a menor profundidade, partindo de **s** com que conseguimos visitar **t**.

Aplicações: Caminhos Mínimos

- Dado o funcionamento da busca em largura, isso é exatamente o que ela calcula.
- Primeiro todos os vértices a profundidade 1 são visitados, em seguida todos que estão a profundidade 2 e assim por diante.

Aplicações: Caminhos Mínimos

```
int menorCaminho( int s, int t ){

    int dist[num_vert];
    memset( dist, -1, sizeof( dist ) );
    dist[s] = 0;

    ...

    return dist[t];
}
```

continua..

Referências e bibliografia

- [TADM] - Steven S. Skiena -*The Algorithm Design Manual*
- Thomas H. Cormen, Charles E. Leiserson - *Introduction to Algorithms, Second Edition*