MC-202 — Aula 24
Representação de grafos em memória e percursos[1]

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

[1] Esses slides foram adaptados do material didático gentilmente cedido pelo Prof. **Guilherme P. Telles**. Todos os erros são meus.

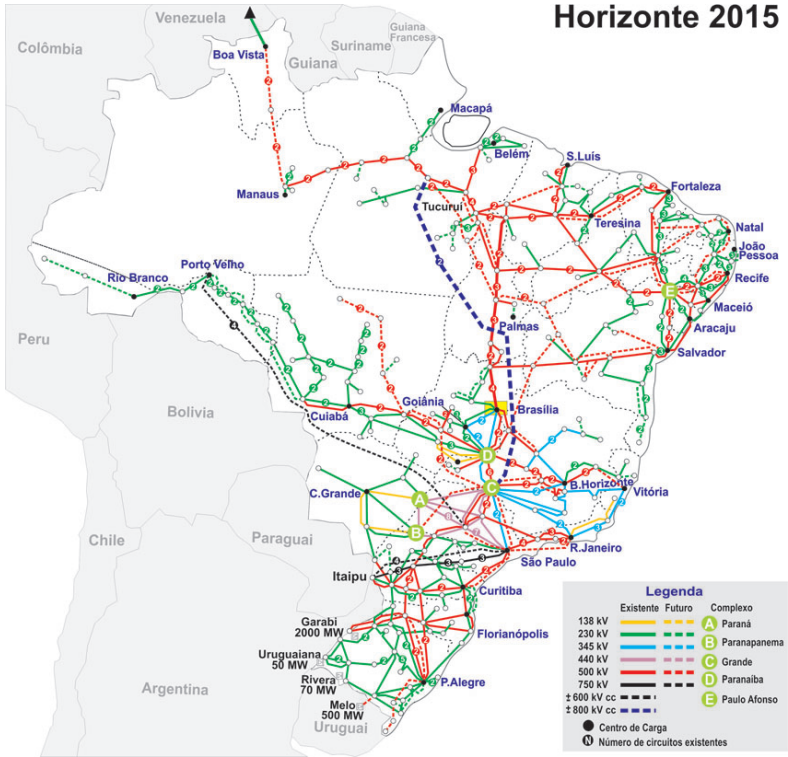
Roteiro

Dever de casa

Discutam, pesquisem e tragam escrito para a próxima aula:

1. Em uma fábrica automotiva, há diversos produtos: parafusos, roda, motor, injetor, volante, transmissão etc. Para fabricar um produto, pode ser necessário montar outros produtos anteriormente. Por exemplo, para montar um carro, primeiro é preciso ter montado o capô; para montar o capô, primeiro é necessário ter colocado o motor e assim por diante. Pense em algum algoritmo que, dada a lista de produtos e dependência, indique qual deve ser a ordem de montagem.
2. Você tem 3 baldes que cheios comportam 3, 5 e 8 litros. Os dois primeiros estão vazios e o último está cheio de água. Seu objetivo é dividir toda água em partes iguais (sem desperdiçar!). Infelizmente, as marcas dos baldes estão apagadas e você não tem nada para medir o volume além dos próprios baldes. Despejar cada litro de água gasta um minuto.
   - Como dividir a água com o menor tempo possível?
   - (desafio) Escreva um algoritmo que resolva o problema para o caso geral com n baldes, cada um com capacidade para l_i litros. Dica: podemos ver esse problema como um problema de grafo?

Sistema de Transmissão de Energia

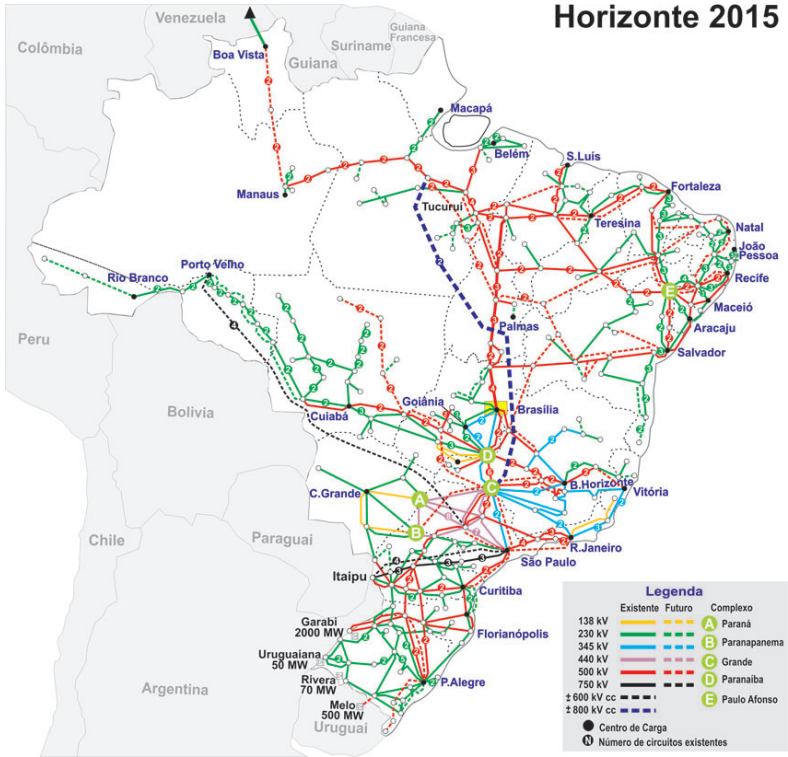


Fonte: SIN

Problema

O sistema de transmissão de energia pode ser modelado por um grafo conexo!
Vez ou outra, podem ocorrer falhas de estações ou de linhas de transmissão.

Sistema de Transmissão de Energia



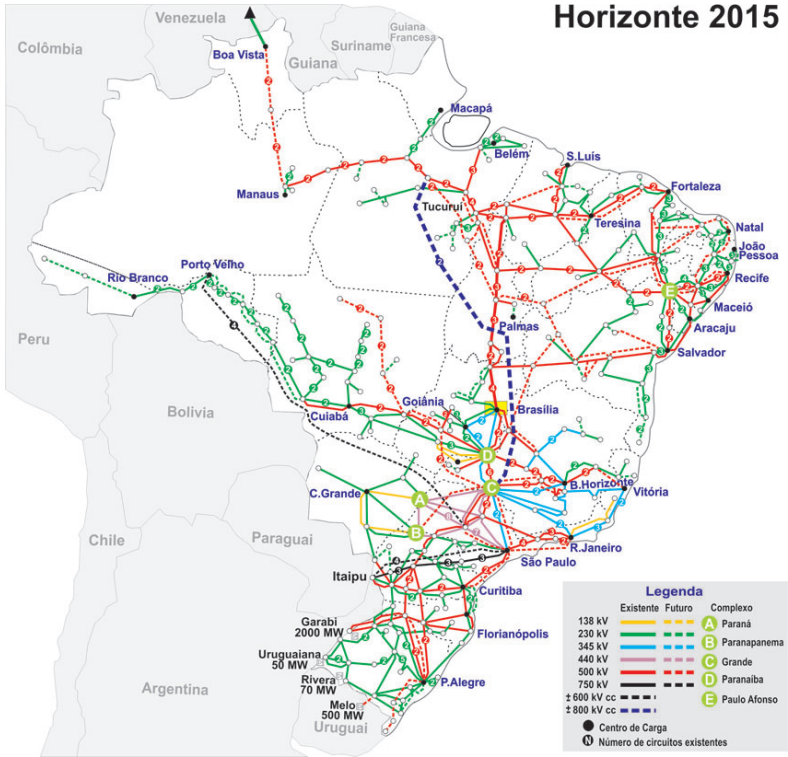
Fonte: SIN

Problema

O sistema de transmissão de energia pode ser modelado por um grafo conexo!
Vez ou outra, podem ocorrer falhas de estações ou de linhas de transmissão.

1. Como representar na memória?

Sistema de Transmissão de Energia



Fonte: SIN

Problema

O sistema de transmissão de energia pode ser modelado por um grafo conexo!
Vez ou outra, podem ocorrer falhas de estações ou de linhas de transmissão.

1. Como representar na memória?
2. Como verificar se um grafo é conexo?

Representação de grafos em memória

Objetivo: guardar grafo na memória

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar
- tempo de operação:

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar
- tempo de operação:
  - verificar se existe aresta

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar
- tempo de operação:
  - verificar se existe aresta
  - descobrir vizinhos

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar
- tempo de operação:
  - verificar se existe aresta
  - descobrir vizinhos, etc.

Representação de grafos em memória

Objetivo: guardar grafo na memória

Dificuldades

- conjunto dinâmico de elementos:
  - inserir e remover vértices
  - inserir e remover arestas
- memória utilizada para armazenar
- tempo de operação:
  - verificar se existe aresta
  - descobrir vizinhos, etc.

Ideias?

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

- string

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

- string
- endereço de memória do nó associado

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

- string
- endereço de memória do nó associado
- um número $\{0, 1, ..., n-1\}$ para um grafo com n vértices

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

- string
- endereço de memória do nó associado
- um número $\{0, 1, ..., n-1\}$ para um grafo com n vértices

Lista de adjacência

A **lista de adjacências** é definida por:

- lista de rótulos $Adj(u)$ dos vizinhos para cada vértice u;
- conjunto de vértices indexados por seus rótulos

Lista de adjacências

Rótulo

Um rótulo (ou *chave*) é algum **identificador** do vértice:

- string
- endereço de memória do nó associado
- um número $\{0, 1, ..., n-1\}$ para um grafo com n vértices

Lista de adjacência

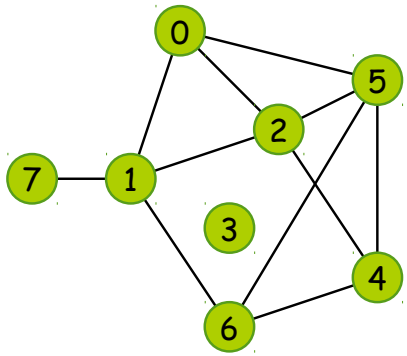
A **lista de adjacências** é definida por:

- lista de rótulos $Adj(u)$ dos vizinhos para cada vértice u;
- conjunto de vértices indexados por seus rótulos

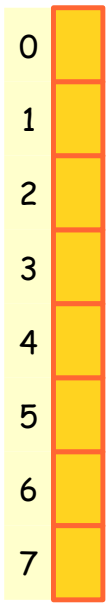
Exemplo:

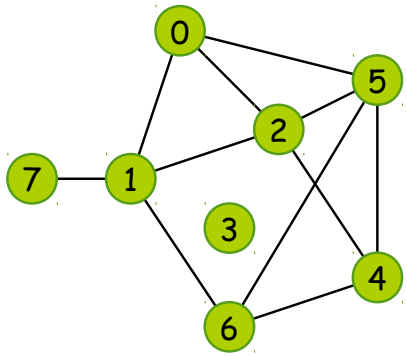
- **lista de rótulos:** lista de números de $\{0, ..., n-1\}$
- **conjunto de vértices:** vetor de ponteiros de tamanho n

Listas de adjacência: vetor de ponteiros



Listas de adjacência: vetor de ponteiros

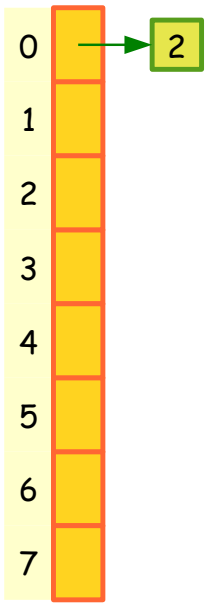


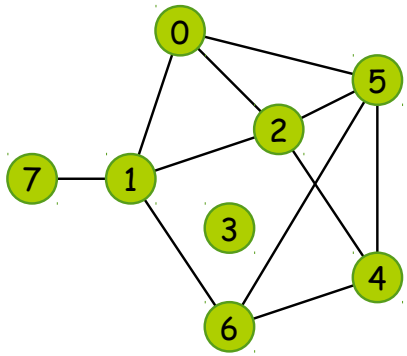


Vetor de ponteiros

1. criamos um vetor de ponteiros

Listas de adjacência: vetor de ponteiros

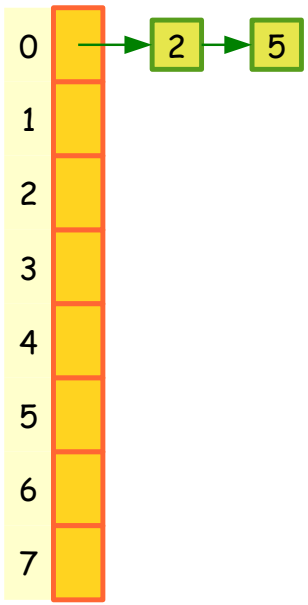


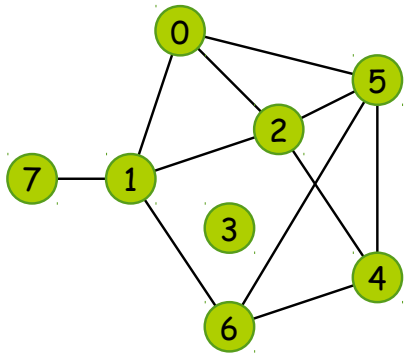


Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos

Listas de adjacência: vetor de ponteiros

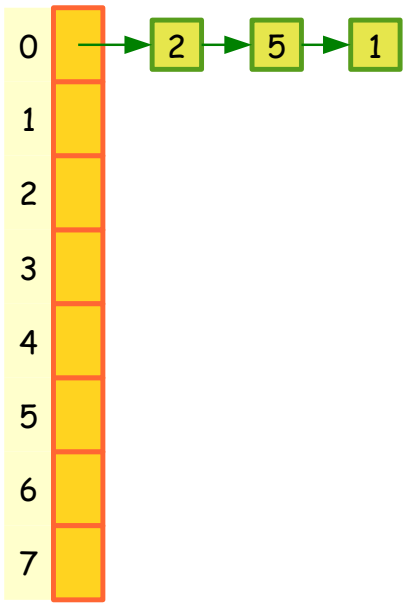


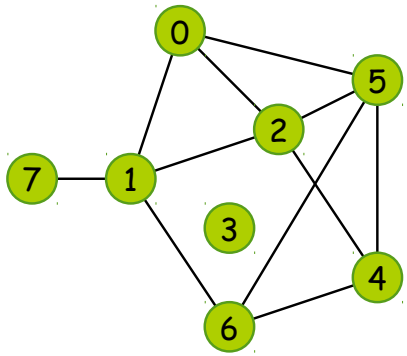


Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos

Listas de adjacência: vetor de ponteiros

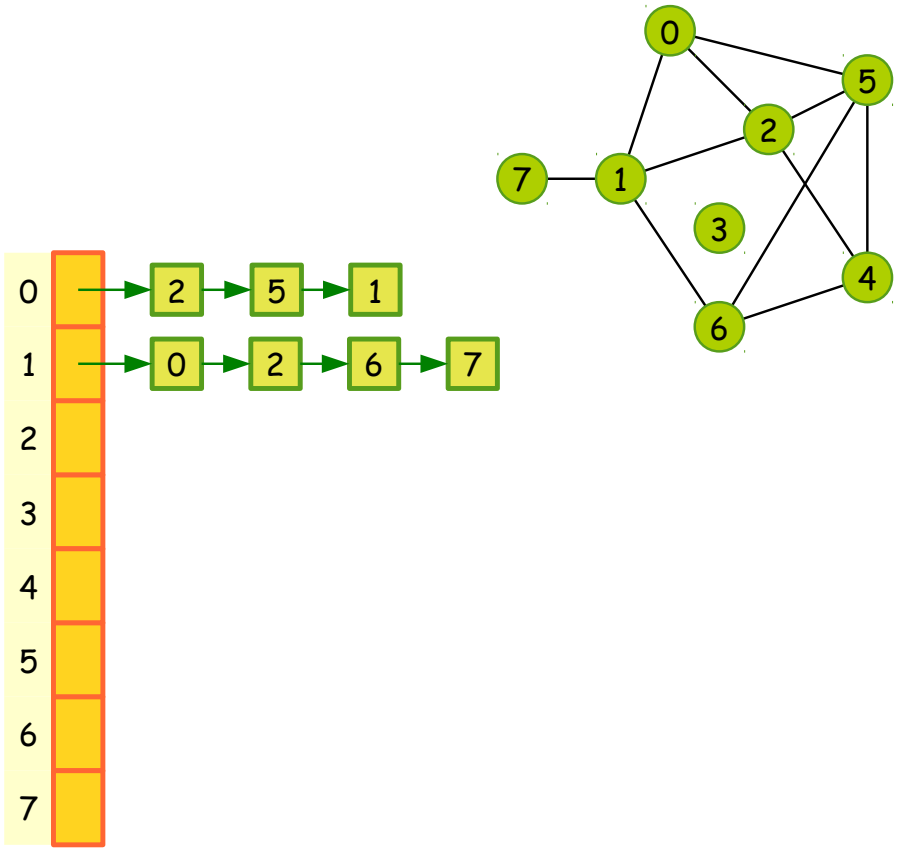




Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos

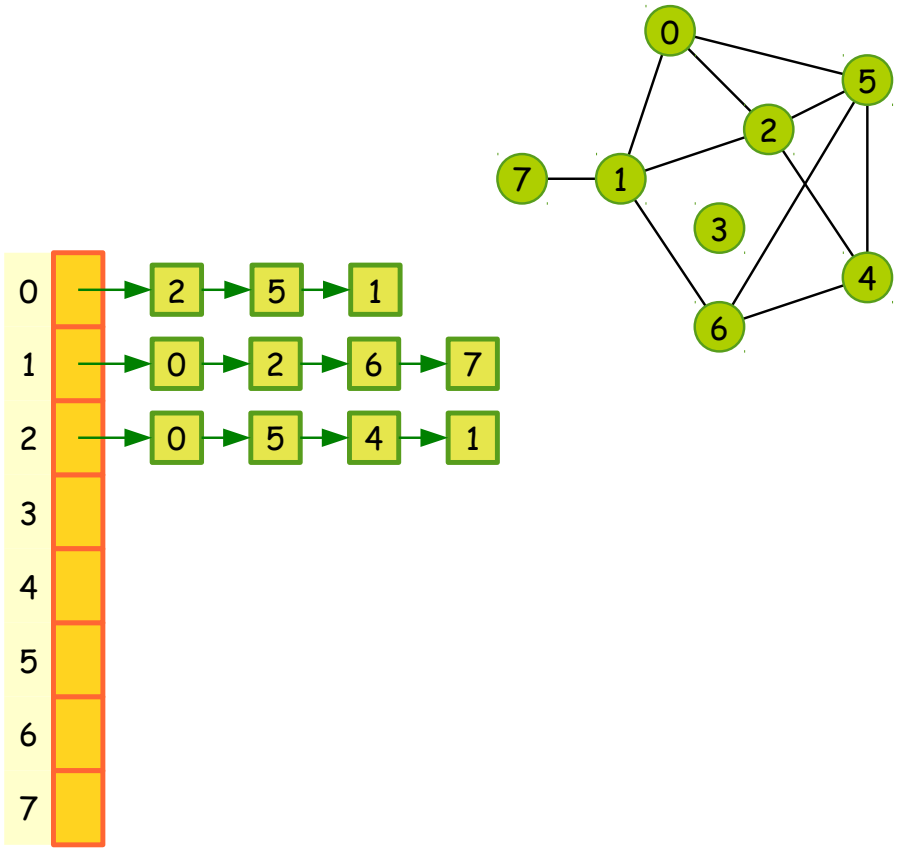
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

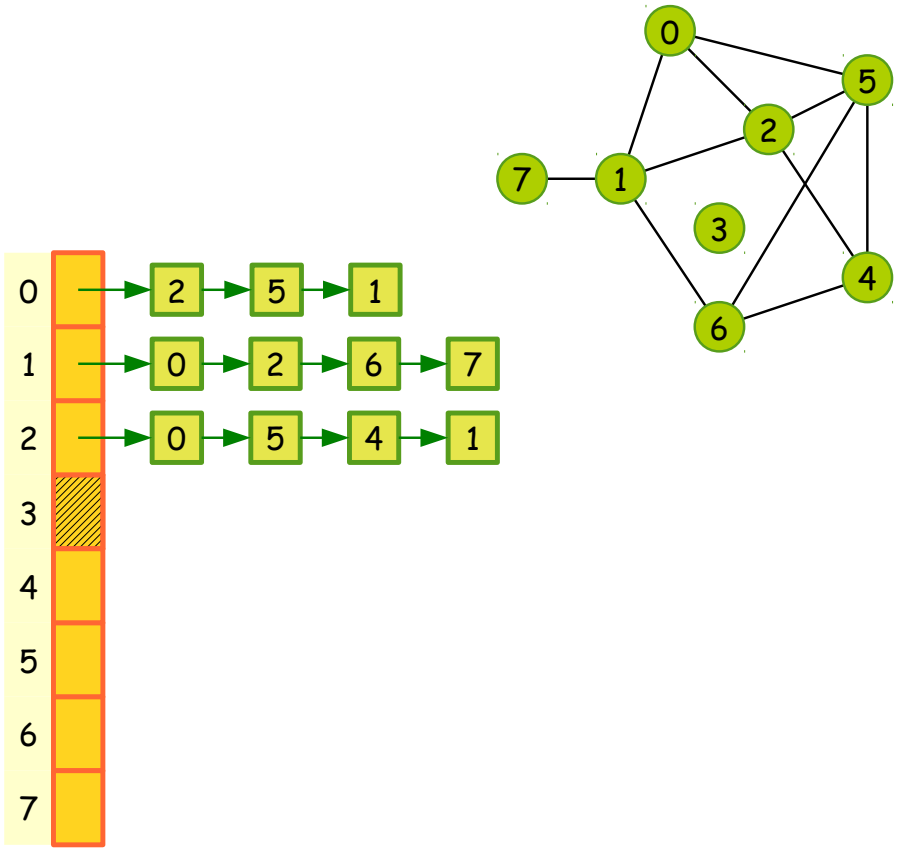
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

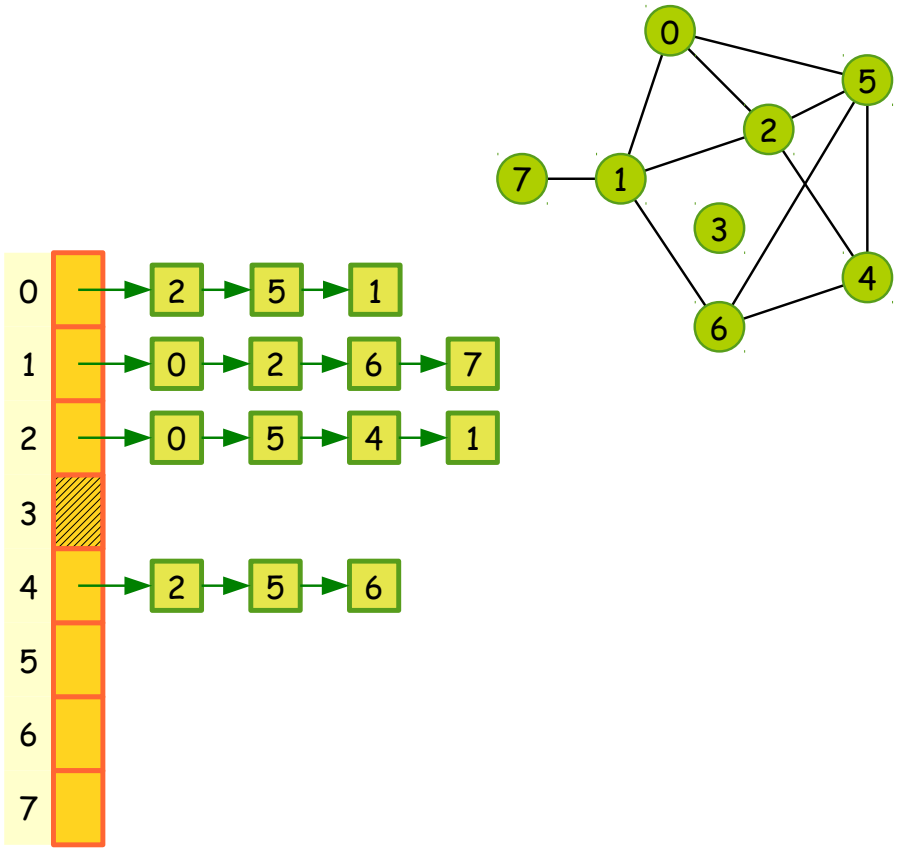
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

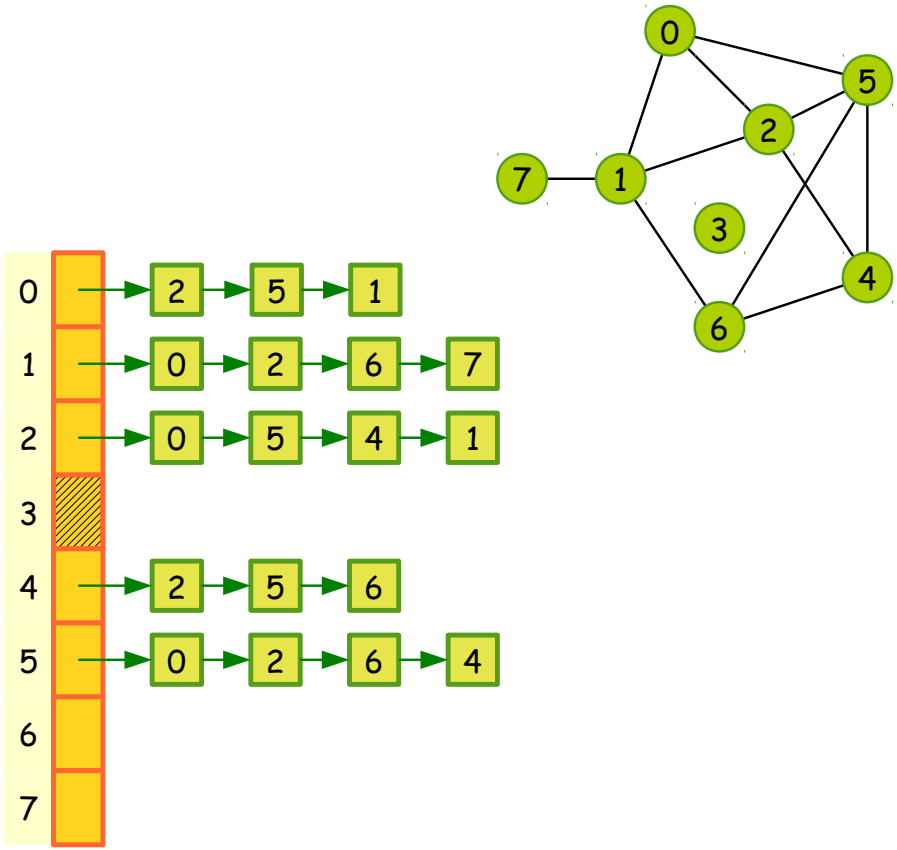
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

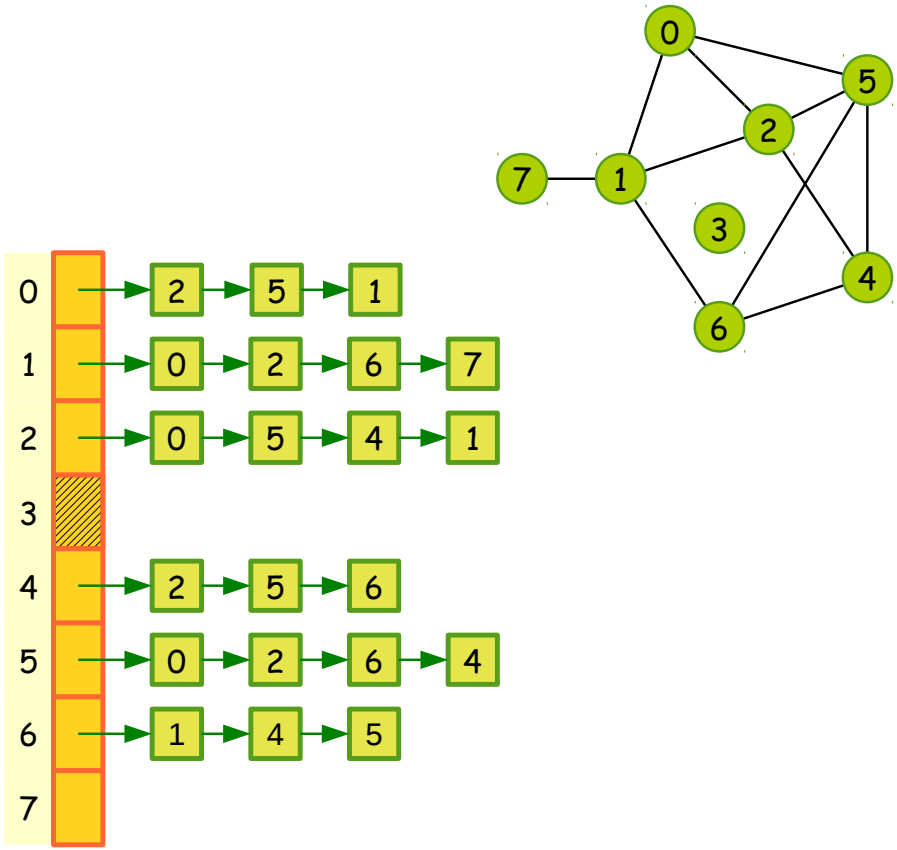
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

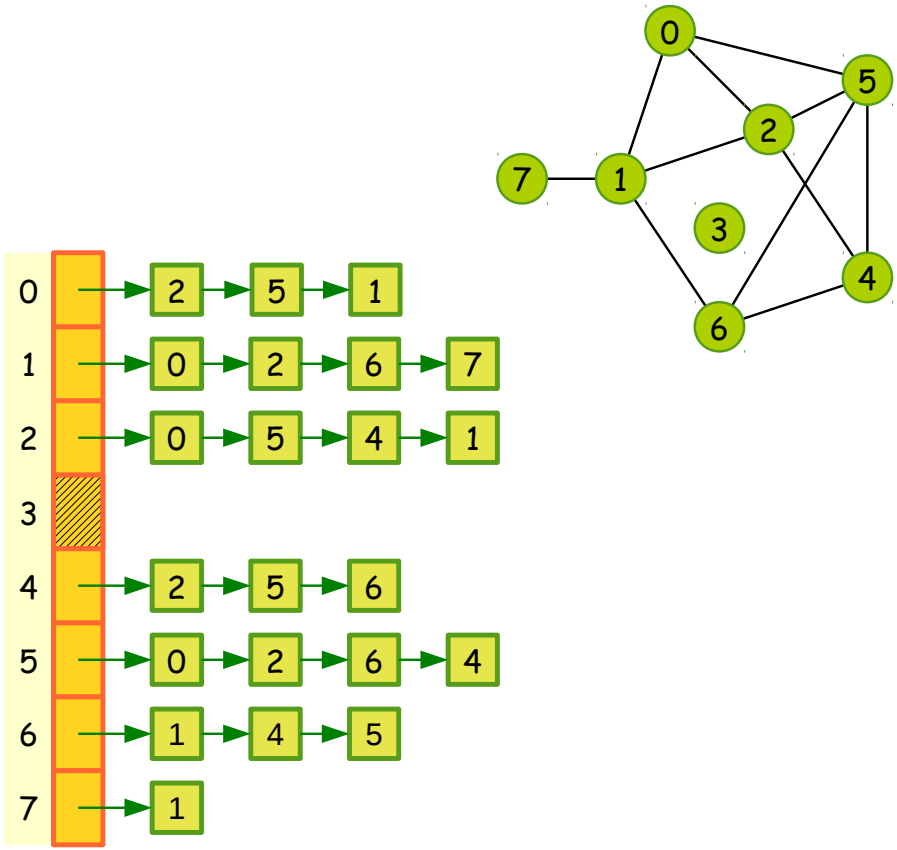
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

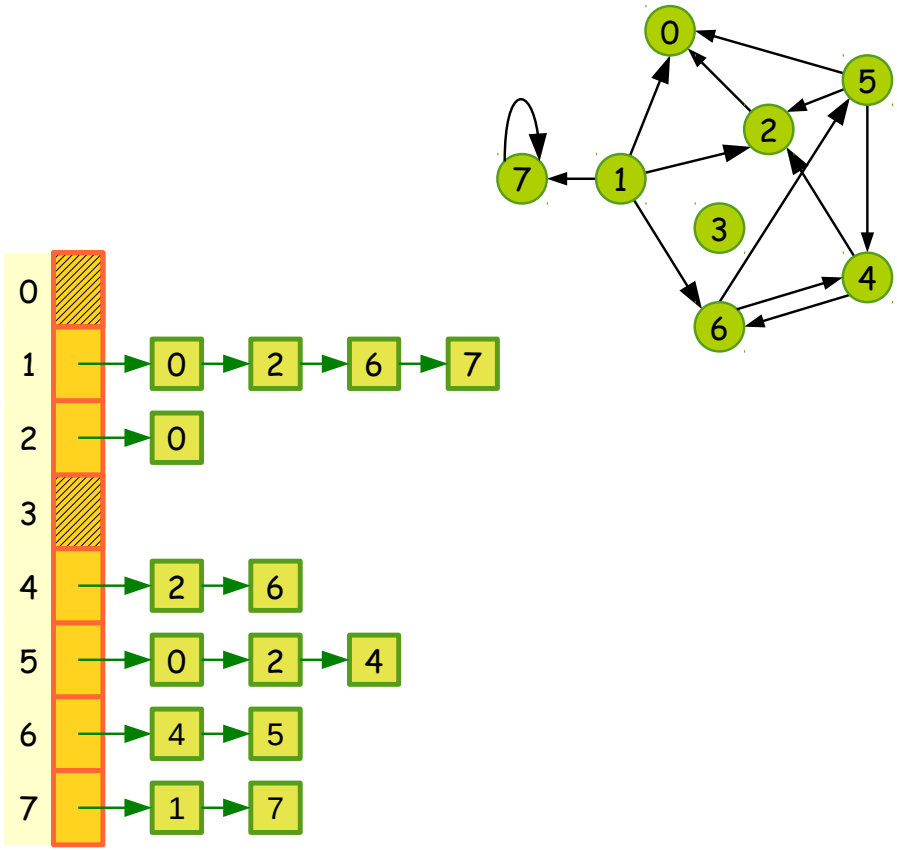
Listas de adjacência: vetor de ponteiros



Vetor de ponteiros

1. criamos um vetor de ponteiros
2. criamos uma lista de índices de vizinhos para cada vértice

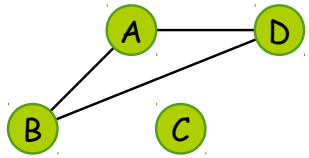
Listas de adjacência: grafo direcionado



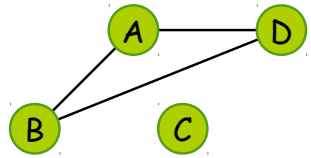
Vetor de ponteiros

- as arestas só aparecem em uma lista!

Listas de adjacência: lista de listas



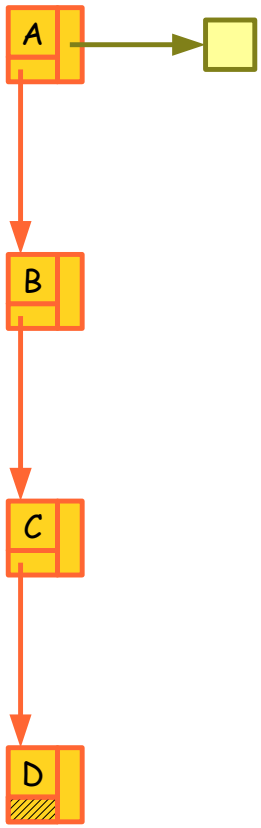
Listas de adjacência: lista de listas

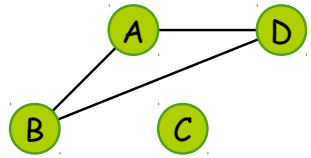


Lista de listas

1. criamos um lista de listas

Listas de adjacência: lista de listas

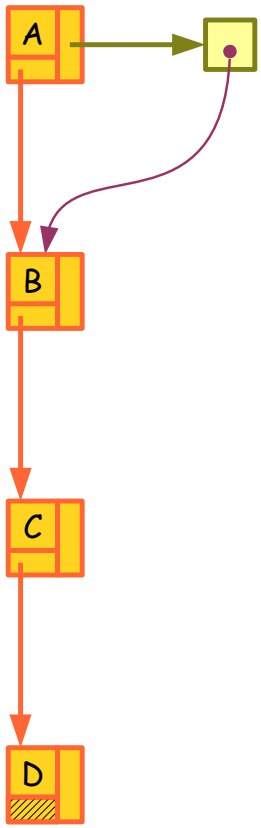


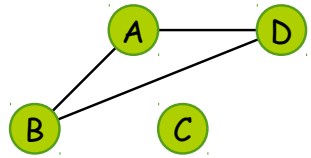


Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos

Listas de adjacência: lista de listas

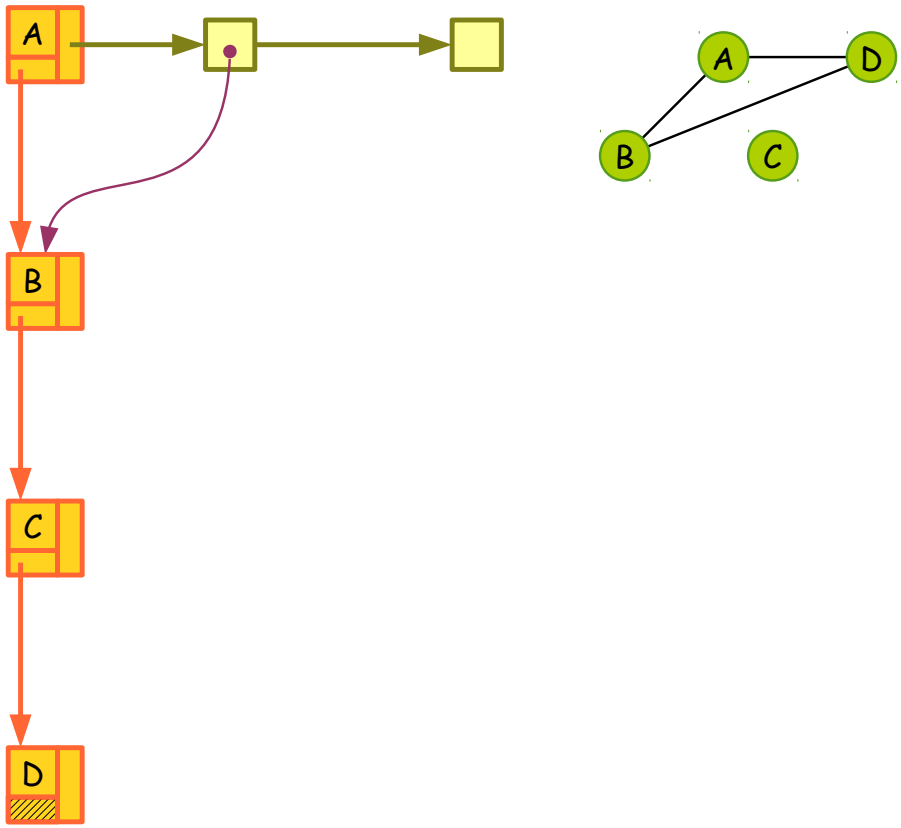




Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos

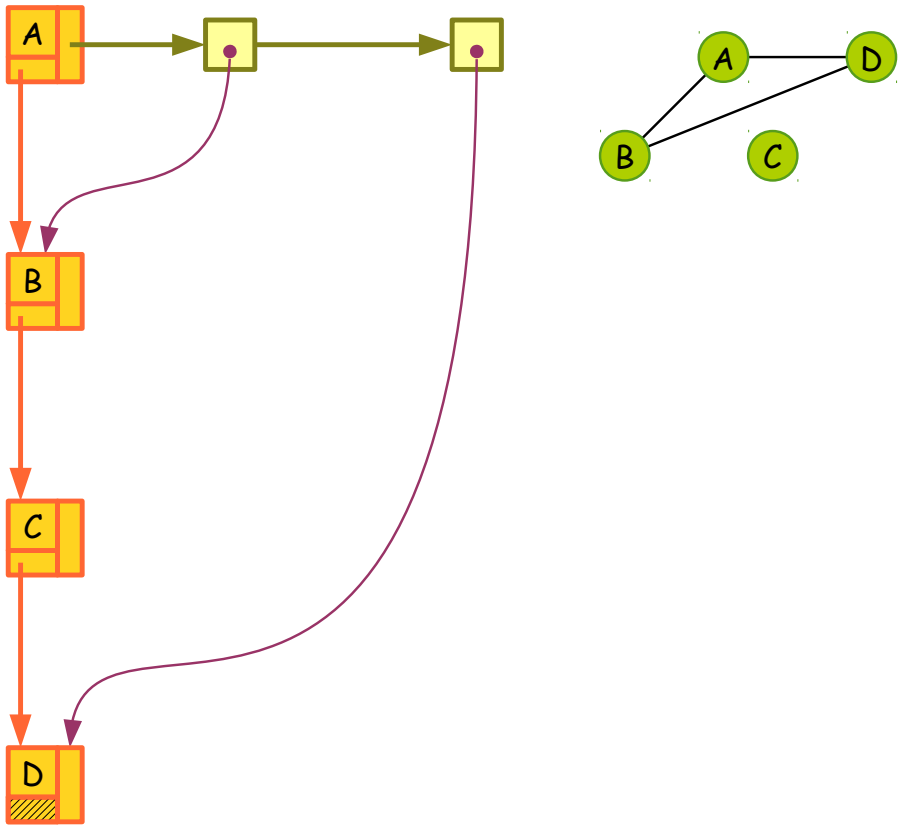
Listas de adjacência: lista de listas



Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos

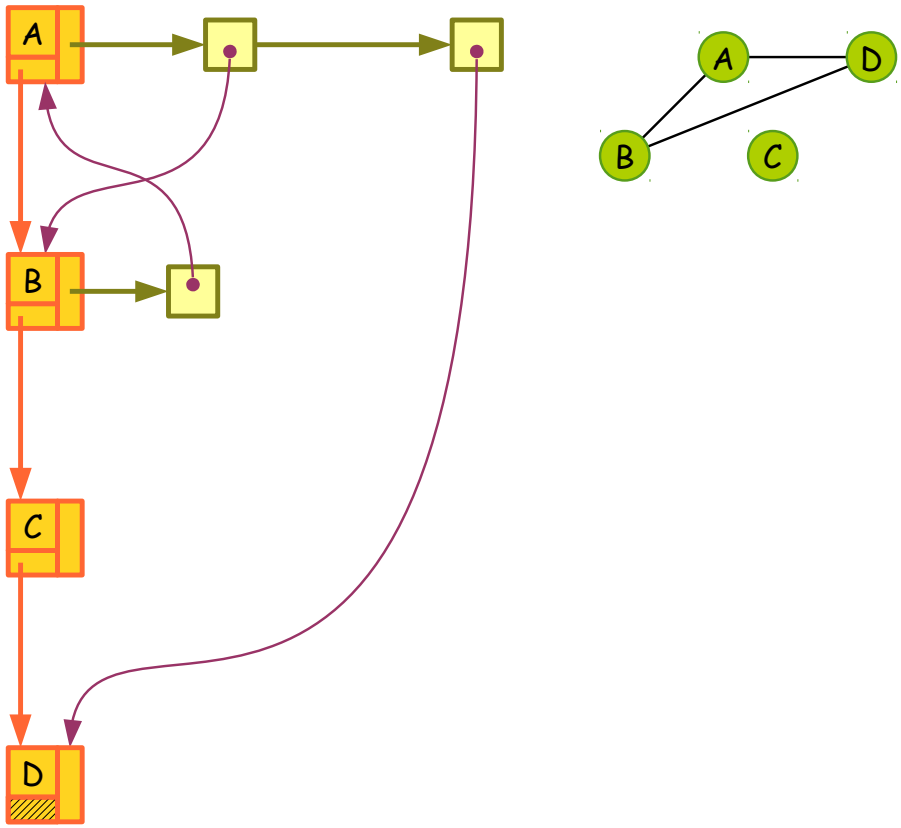
Listas de adjacência: lista de listas



Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos para cada vértice

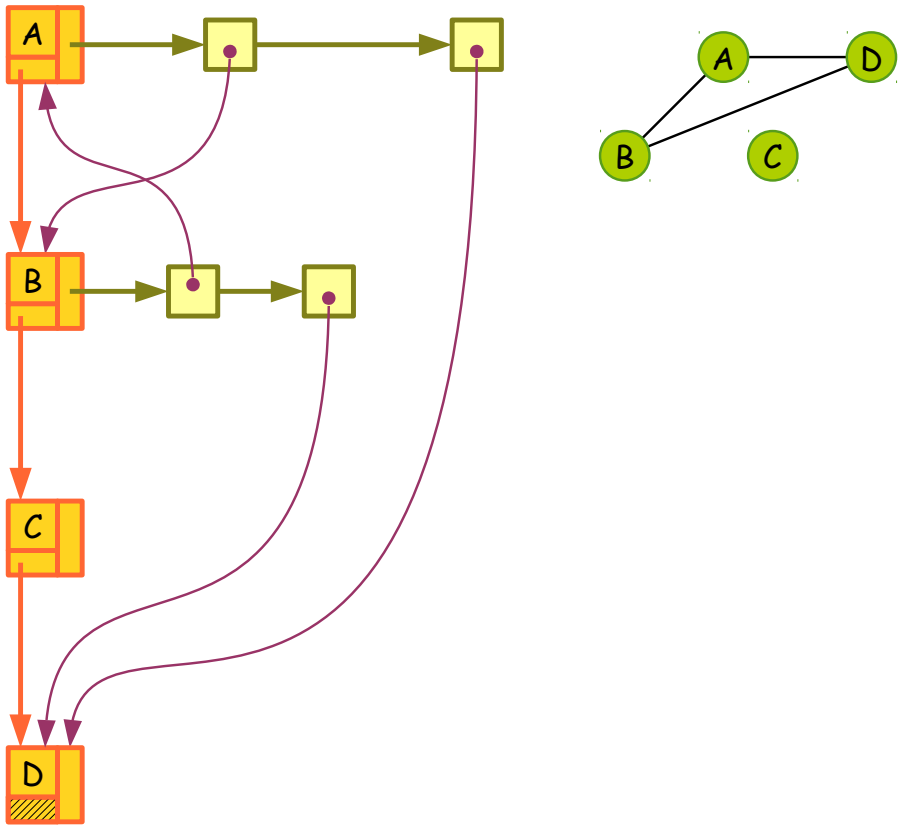
Listas de adjacência: lista de listas



Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos para cada vértice

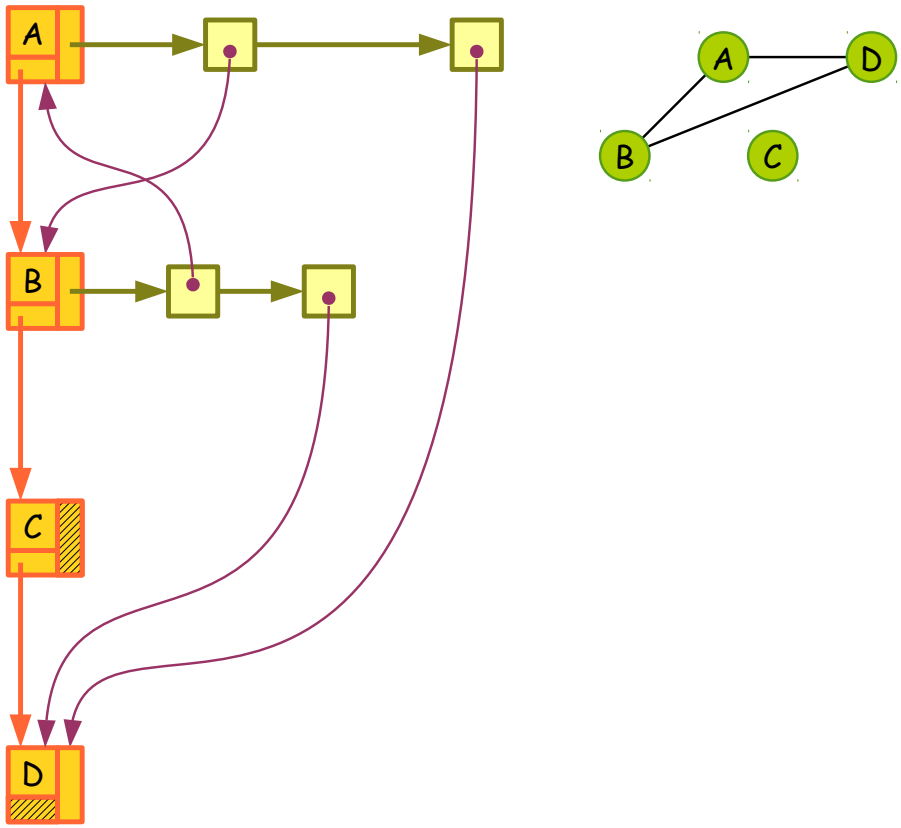
Listas de adjacência: lista de listas



Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos para cada vértice

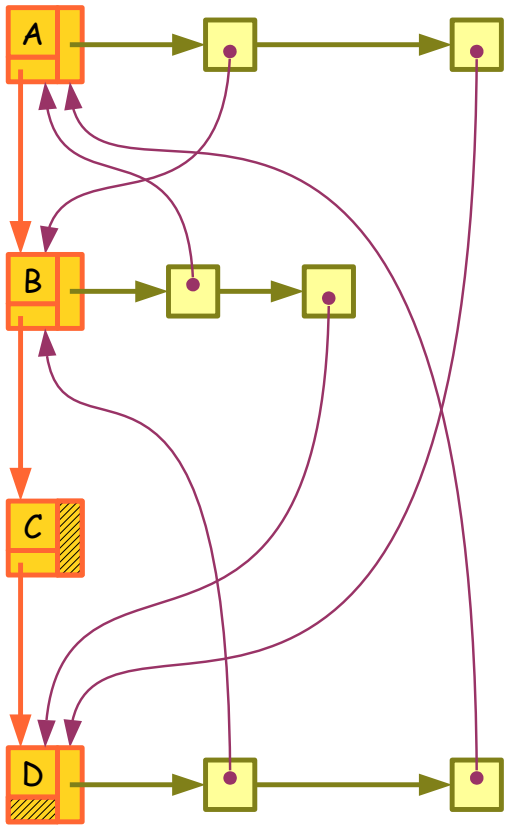
Listas de adjacência: lista de listas

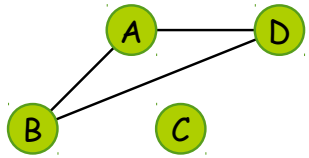


Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos para cada vértice

Listas de adjacência: lista de listas

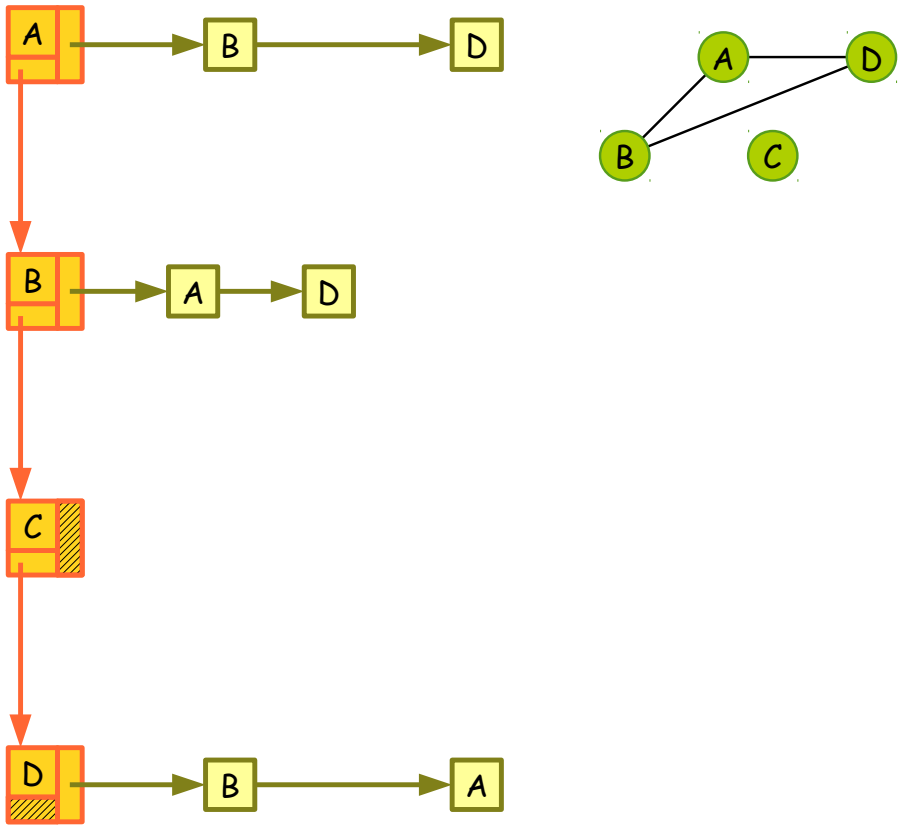




Lista de listas

1. criamos um lista de listas
2. criamos uma lista de ponteiros de vizinhos para cada vértice

Listas de adjacência: lista de listas (alternativa)



Lista de listas com Tabela Hash

1. criamos um lista de listas (guardados em uma Tabela Hash)
2. criamos uma lista de rótulos de vizinhos para cada vértice

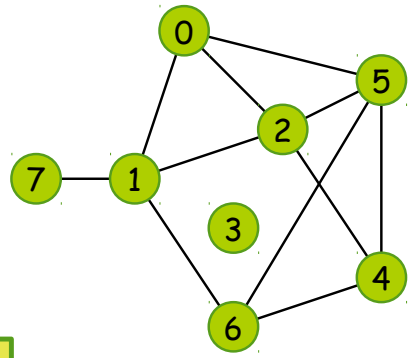
Operações com Listas de Adjacência

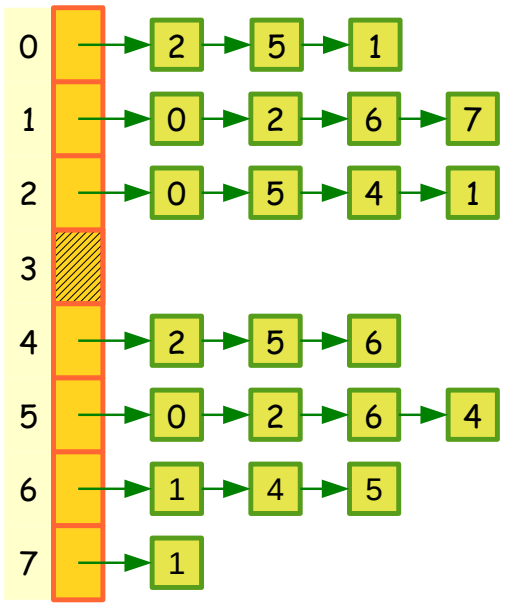
Operações

- verificar se há aresta $(u, v) \in E$
  - **procedimento:** procurar v na lista $Adj(u)$
  - **tempo:** $O(\deg(u))$
- percorrer vizinhos de u
  - **procedimento:** percorrer lista $Adj(u)$
  - **tempo:** $O(\deg(u))$
- inserir novo vértice u
  - **procedimento:** criar $Adj(u)$; insere u em $Adj(v)$ para cada $v \in Adj(u)$
  - **tempo:** $O(\deg(u))$
- inserir nova aresta (u, v)
  - **procedimento:** adicionar v em $Adj(u)$ (e adicionar u em $Adj(v)$)
  - **tempo:** $O(1)$
- remover aresta (u, v)
  - **procedimento:** remover v de $Adj(u)$ (e remover u em $Adj(v)$)
  - **tempo:** $O(\deg(u))$

Exemplo de remoção

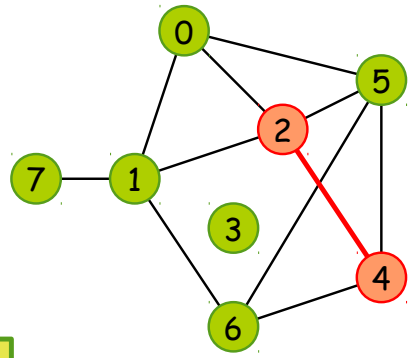
Remover aresta {2,4}

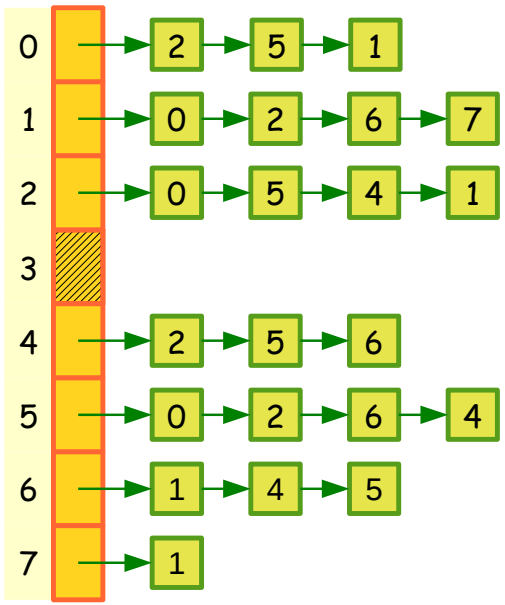




Exemplo de remoção

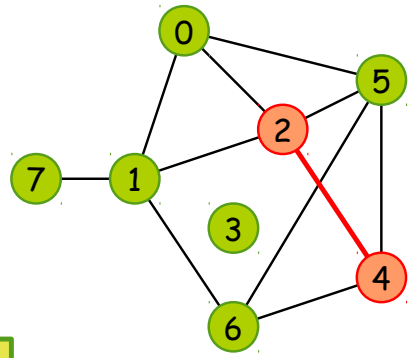
Remover aresta {2,4}

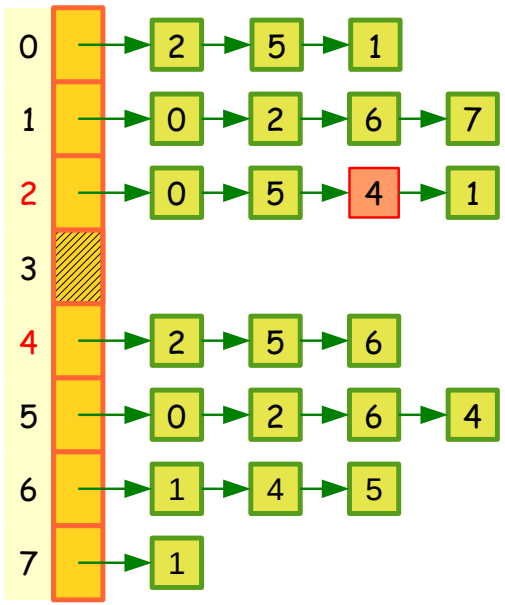




Exemplo de remoção

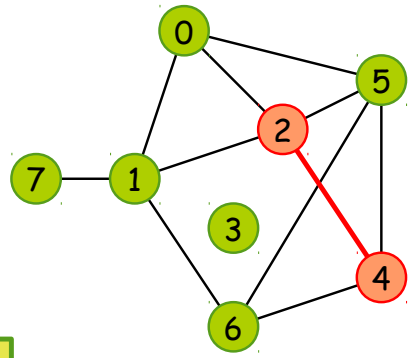
Remover aresta {2,4}

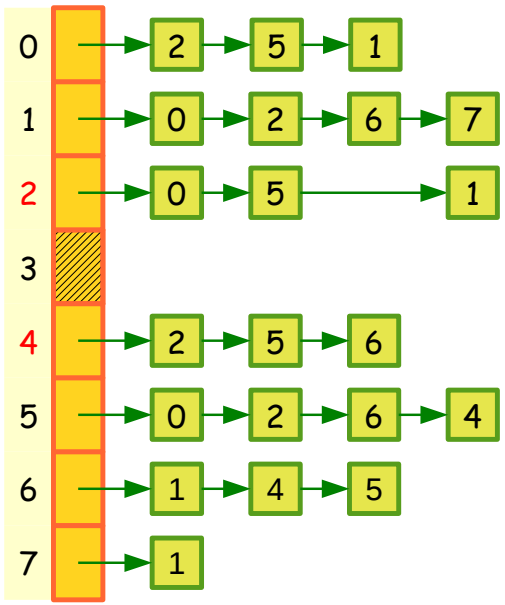




Exemplo de remoção

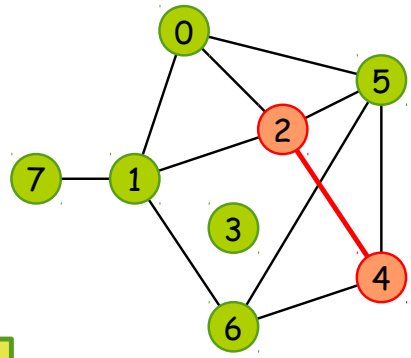
Remover aresta {2,4}

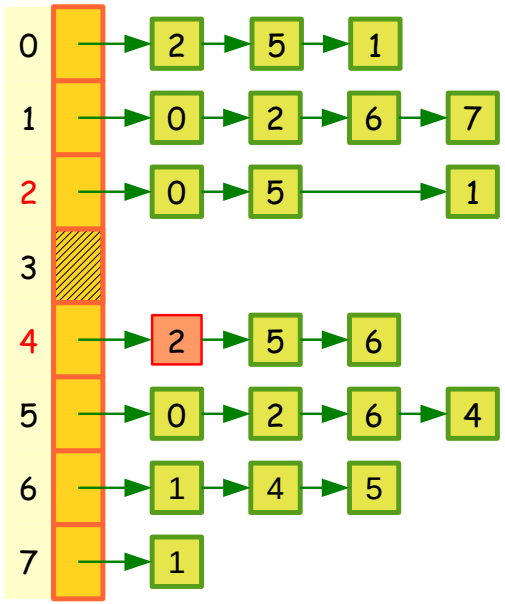




Exemplo de remoção

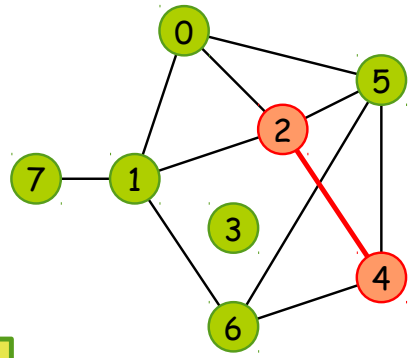
Remover aresta {2,4}

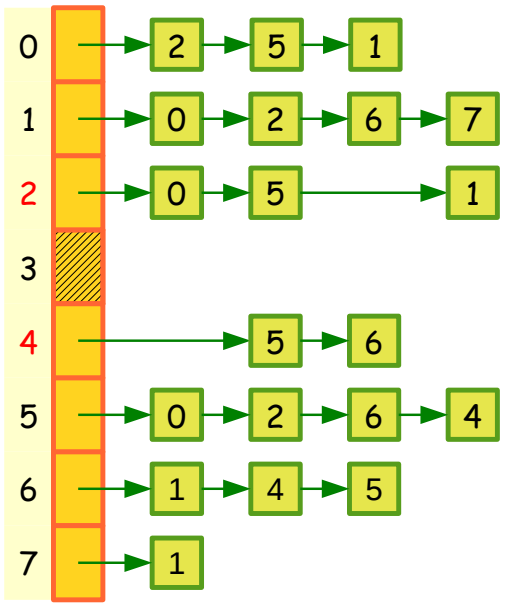




Exemplo de remoção

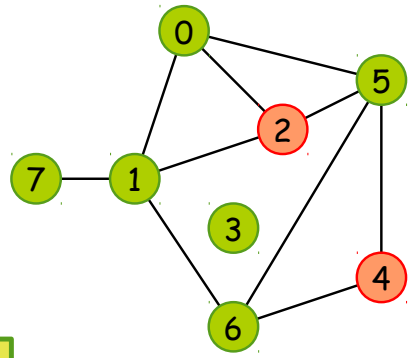
Remover aresta {2,4}

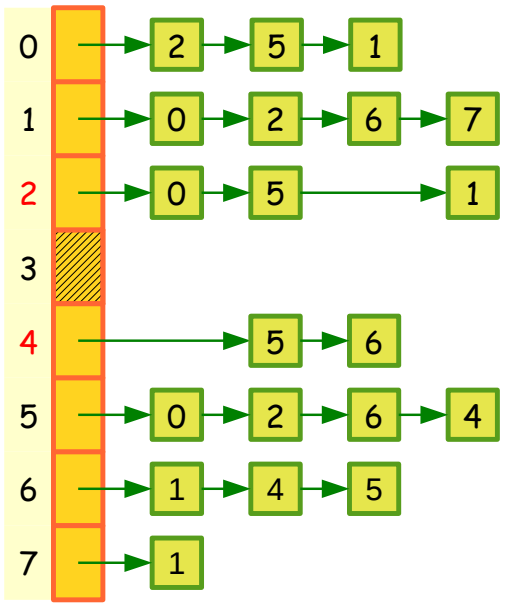




Exemplo de remoção

Remover aresta {2,4}





Outro problema

Livro das Caretas

Em uma nova rede social de compartilhamento de fotos um usuário só pode ver a foto do outro se eles forem amigos. Há milhares de acessos simultâneos e o acesso às fotos deve ser rápido.

Outro problema

Livro das Caretas

Em uma nova rede social de compartilhamento de fotos um usuário só pode ver a foto do outro se eles forem amigos. Há milhares de acessos simultâneos e o acesso às fotos deve ser rápido.

Problema: uma lista de adjacências não é rápido o suficiente

Matriz de adjacências

Matriz de adjacências

A **matriz de adjacências** de um gafo simples $G = (V, E)$ é uma matriz quadrada de ordem $|V|$, cujas linhas e colunas são indexadas pelos vértices em V e tal que:

$$A_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

Matriz de adjacências

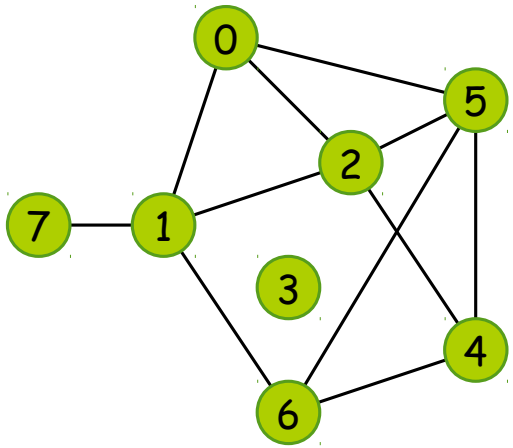
Matriz de adjacências

A **matriz de adjacências** de um gafo simples $G = (V, E)$ é uma matriz quadrada de ordem $|V|$, cujas linhas e colunas são indexadas pelos vértices em V e tal que:

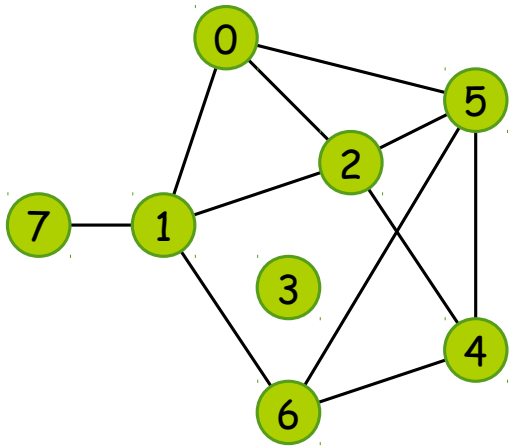
$$A_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

Observação: Se G é não orientado, então a matriz de adjacências é simétrica.

Exemplo de matriz de adjacências

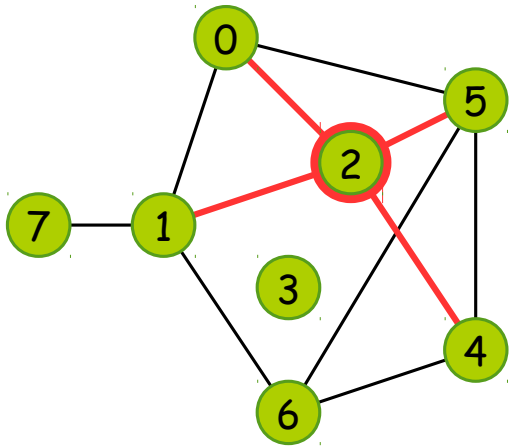


Exemplo de matriz de adjacências



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Exemplo de matriz de adjacências



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Operações com Matriz de Adjacência

Operações

- verificar se há aresta $(u, v) \in E$
  - **procedimento:** olhar posição A_{uj}
  - **tempo:** $O(1)$
- percorrer vizinhos de u
  - **procedimento:** percorrer linha $A_{u(\cdot)}$
  - **tempo:** $O(n)$
- inserir ou remover novo vértice u
  - **procedimento:** recriar matriz
  - **tempo:** $O(n^2)$
- inserir ou remover aresta (u, v)
  - **procedimento:** alterar A_{uv} (e A_{vu})
  - **tempo:** $O(1)$

Operações com Matriz de Adjacência

Operações

- verificar se há aresta $(u, v) \in E$
  - **procedimento:** olhar posição A_{uj}
  - **tempo:** $O(1)$
- percorrer vizinhos de u
  - **procedimento:** percorrer linha $A_{u(\cdot)}$
  - **tempo:** $O(n)$
- inserir ou remover novo vértice u
  - **procedimento:** recriar matriz
  - **tempo:** $O(n^2)$
- inserir ou remover aresta (u, v)
  - **procedimento:** alterar A_{uv} (e A_{vu})
  - **tempo:** $O(1)$

Observação: Pode-se utilizar implementações de matriz mais eficientes

Uso de memória

Listas de adjacência

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)
  - em grafos orientados: usamos $|E|$ rótulos

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)
  - em grafos orientados: usamos $|E|$ rótulos

TOTAL: $O(|V| + |E|)$

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)
  - em grafos orientados: usamos $|E|$ rótulos

TOTAL: $O(|V| + |E|)$

Matriz de adjacência

- uma matriz com n^2 inteiros

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)
  - em grafos orientados: usamos $|E|$ rótulos

TOTAL: $O(|V| + |E|)$

Matriz de adjacência

- uma matriz com n^2 inteiros

TOTAL: $O(|V|^2)$

Uso de memória

Listas de adjacência

- um vetor de apontadores de tamanho $|V|$
- para cada lista, um conjunto de rótulos
  - em grafos não-orientados: usamos $2|E|$ rótulos (por quê?)
  - em grafos orientados: usamos $|E|$ rótulos

TOTAL: $O(|V| + |E|)$

Matriz de adjacência

- uma matriz com n^2 inteiros

TOTAL: $O(|V|^2)$

Listas são mais baratas para grafos esparsos (com "poucas" arestas).

Outras formas de representação

Variantes de matriz de adjacência

- grafos não-simples (com arestas múltiplas)
- com pesos nas arestas, com cores nas arestas, etc.

Outras formas de representação

Variantes de matriz de adjacência

- grafos não-simples (com arestas múltiplas)
- com pesos nas arestas, com cores nas arestas, etc.

Outras formas

- **Vetor de arestas** para grafos que mudam pouco
  - um vetor de arestas E
  - vetor de vértices V
  - arestas incidentes em i são $E[V[i]....V[i+1]-1]$

Outras formas de representação

Variantes de matriz de adjacência

- grafos não-simples (com arestas múltiplas)
- com pesos nas arestas, com cores nas arestas, etc.

Outras formas

- **Vetor de arestas** para grafos que mudam pouco
  - um vetor de arestas E
  - vetor de vértices V
  - arestas incidentes em i são $E[V[i]....V[i+1]-1]$
- **Matriz de incidência** para problemas de fluxo e outros
  - uma matriz I de tamanho $m \times n$
  - cada célula $I_{ij} = 1$ se aresta i é incidente no vértice j

Outras formas de representação

Variantes de matriz de adjacência

- grafos não-simples (com arestas múltiplas)
- com pesos nas arestas, com cores nas arestas, etc.

Outras formas

- **Vetor de arestas** para grafos que mudam pouco
  - um vetor de arestas E
  - vetor de vértices V
  - arestas incidentes em i são $E[V[i]....V[i+1]-1]$
- **Matriz de incidência** para problemas de fluxo e outros
  - uma matriz I de tamanho $m \times n$
  - cada célula $I_{ij} = 1$ se aresta i é incidente no vértice j
- **Representação implícita**
  - não guardamos uma estrutura específica
  - as arestas e vértices são deduzidos das estruturas do problema

Outras formas de representação

Variantes de matriz de adjacência

- grafos não-simples (com arestas múltiplas)
- com pesos nas arestas, com cores nas arestas, etc.

Outras formas

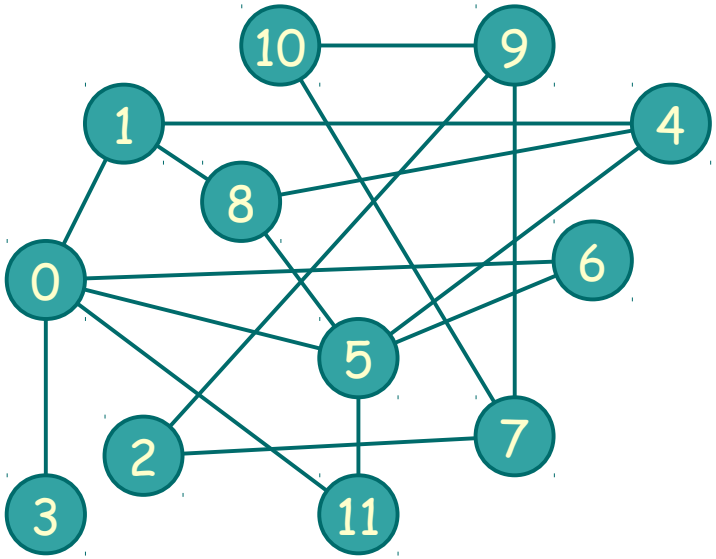
- **Vetor de arestas** para grafos que mudam pouco
  - um vetor de arestas E
  - vetor de vértices V
  - arestas incidentes em i são $E[V[i]....V[i+1]-1]$
- **Matriz de incidência** para problemas de fluxo e outros
  - uma matriz I de tamanho $m \times n$
  - cada célula $I_{ij} = 1$ se aresta i é incidente no vértice j
- **Representação implícita**
  - não guardamos uma estrutura específica
  - as arestas e vértices são deduzidos das estruturas do problema
  - exemplo: faça o exercício!

De volta ao nosso problema

Problema da transmissão

1. Como representar o grafo na memória?

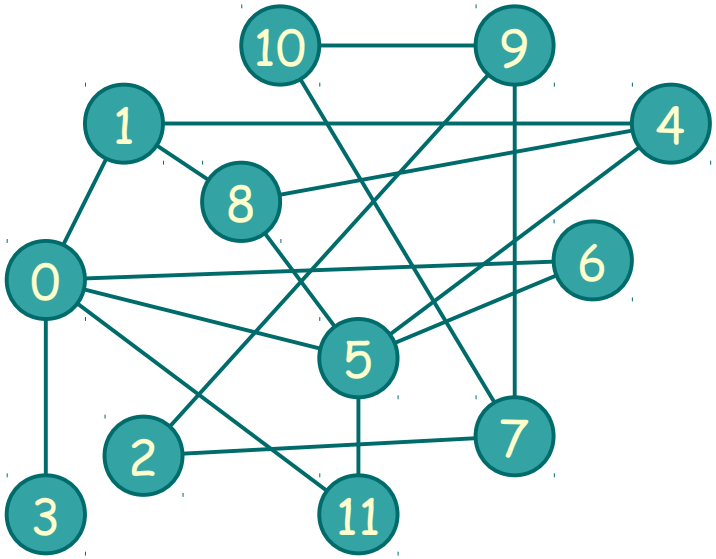
De volta ao nosso problema



Problema da transmissão

1. Como representar o grafo na memória? ✓

De volta ao nosso problema



Problema da transmissão

1. Como representar o grafo na memória? ✓
2. Como verificar se um grafo é conexo?

Buscas em grafos

Busca

Uma busca é uma forma sistemática de visitar/percorrer vértices e arestas.

Buscas em grafos

Busca

Uma busca é uma forma sistemática de visitar/percorrer vértices e arestas.

Aplicações

- encontrar componentes conexas
- encontrar menor caminho a partir de um vértice
- encontrar ciclos
- ordenar vértices, etc.

Buscas em grafos

Busca

Uma busca é uma forma sistemática de visitar/percorrer vértices e arestas.

Aplicações

- encontrar componentes conexas
- encontrar menor caminho a partir de um vértice
- encontrar ciclos
- ordenar vértices, etc.

Vamos estudar dois tipos interessantes de busca:

1. **busca em profundidade**
   - visita os caminhos mais profundos primeiro
2. **busca em largura**
   - visita vértices mais próximos primeiro

Especificação de exemplo

Nos algoritmos de busca:

- guardamos alguns atributos para cada vértice

Especificação de exemplo

Nos algoritmos de busca:

- guardamos alguns atributos para cada vértice

```
struct Vizinho {
    int rotulo;
    struct Vizinho *prox;
}
struct Vertice {
    int rotulo;           // rótulo
    struct Vizinho *adj; // lista de adjacências
    int visitado;         // indica se já foi visitado
    int distancia;        // distância do nó inicial
    int pai;              // nó por onde foi visitado
};
struct Grafo {
    int n;                 // número de vértices
    struct Vertice *vert; // vetor de ponteiros
};
```

Especificação de exemplo

Nos algoritmos de busca:

- guardamos alguns atributos para cada vértice

```
struct Vizinho {
    int rotulo;
    struct Vizinho *prox;
}
struct Vertice {
    int rotulo;           // rótulo
    struct Vizinho *adj; // lista de adjacências
    int visitado;         // indica se já foi visitado
    int distancia;        // distância do nó inicial
    int pai;              // nó por onde foi visitado
};
struct Grafo {
    int n;                 // número de vértices
    struct Vertice *vert; // vetor de ponteiros
};
```

Busca em profundidade

A busca em profundidade ou *depth-first search* (DFS) a partir de um certo vértice preferencialmente amplia o alcance da busca, afastando-se da origem.

Ideia geral

1. Cada vértice mantém um estado:
   - não visitado: não descoberto pela busca ainda
   - marcado: já descoberto pela busca
2. Visita cada vértice v:
   - marca v como visitado
   - visita recursivamente cada vizinho de v

Busca em profundidade

A busca em profundidade ou *depth-first search* (DFS) a partir de um certo vértice preferencialmente amplia o alcance da busca, afastando-se da origem.

Ideia geral

1. Cada vértice mantém um estado:
   - não visitado: não descoberto pela busca ainda
   - marcado: já descoberto pela busca
2. Visita cada vértice v:
   - marca v como visitado
   - visita recursivamente cada vizinho de v

Além de percorrer todos os vértices, a busca irá gerar para cada componente conexa a chamada **árvore de busca**.

Busca em profundidade: pseudocódigo

```
DFS(G)
1  for each vertex u ∈ V
2        u.marked = FALSE
3        u.π = NULL
4  for each vertex u ∈ V
5        if u.marked == FALSE
6              u.marked = TRUE
7              DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
1  for each v ∈ Adj(u)
2        if v.marked == FALSE
3              v.marked = TRUE
4              v.π = u
5              DFS-VISIT(G, v)
```

Busca em profundidade: implementando

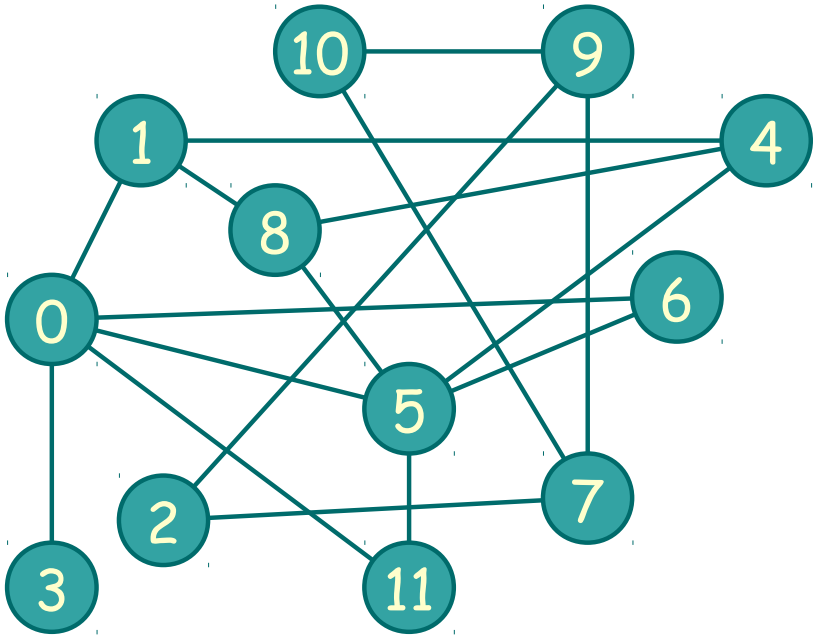
Buscar

```
void buscar_dfs(Grafo *g)
{
  int i;

  // desmarca cada vértice
  for (i = 0; i < g->n; i++) {
    g->vert[i].visitado = 0;
    g->vert[i].pai = -1;
  }
  // visita cada vértice
  for (i = 0; i < g->n; i++) {
    if (!g->vert[i].visitado) {
      g->vert[i].visitado = 1;
      visita_dfs(g, i);
    }
  }
}
```

Visitar

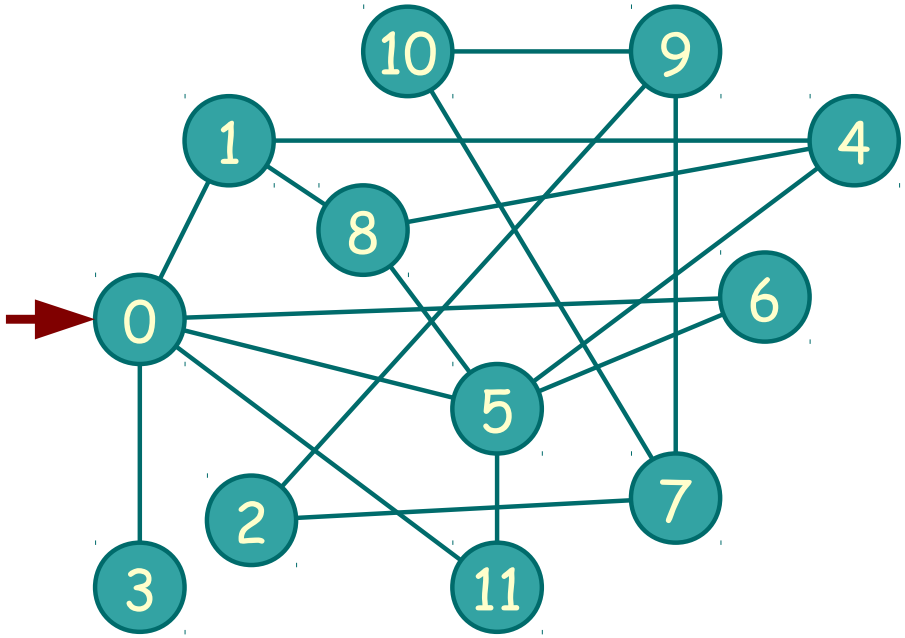
```
void visita_dfs(Grafo *g, int i)
{
  Vizinho *u;
  int j;

  // visita cada vizinho
  u = g->vert[i].adj;
  while (u != NULL) {
    j = u->rotulo;
    if (!g->vert[j].visitado) {
      g->vert[j].visitado = 1;
      g->vert[j].pai = i;
      visita_dfs(g, j);
    }
    u = u->prox
  }
}
```

Busca em profundidade: exemplo



pular

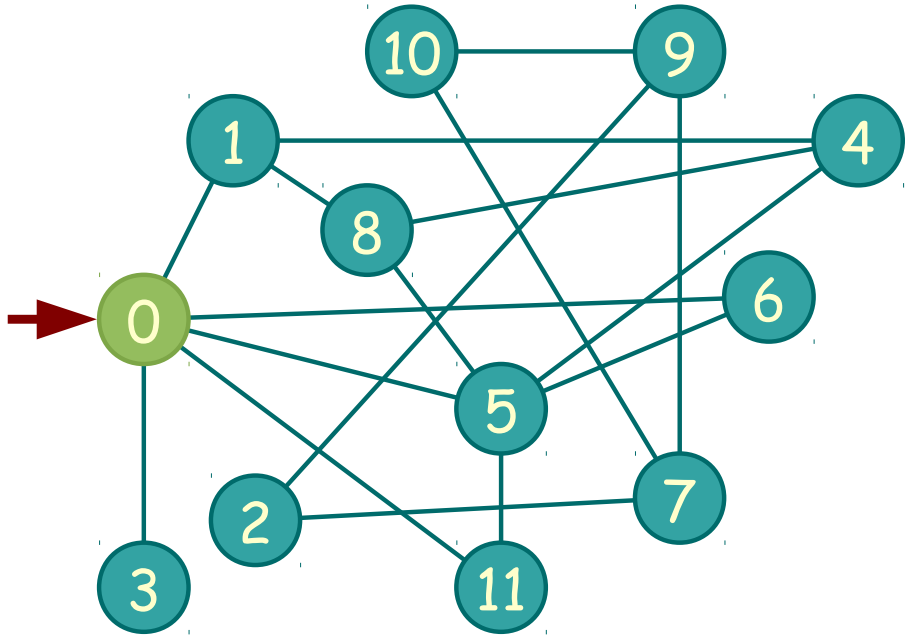
Busca em profundidade: exemplo



0

pular

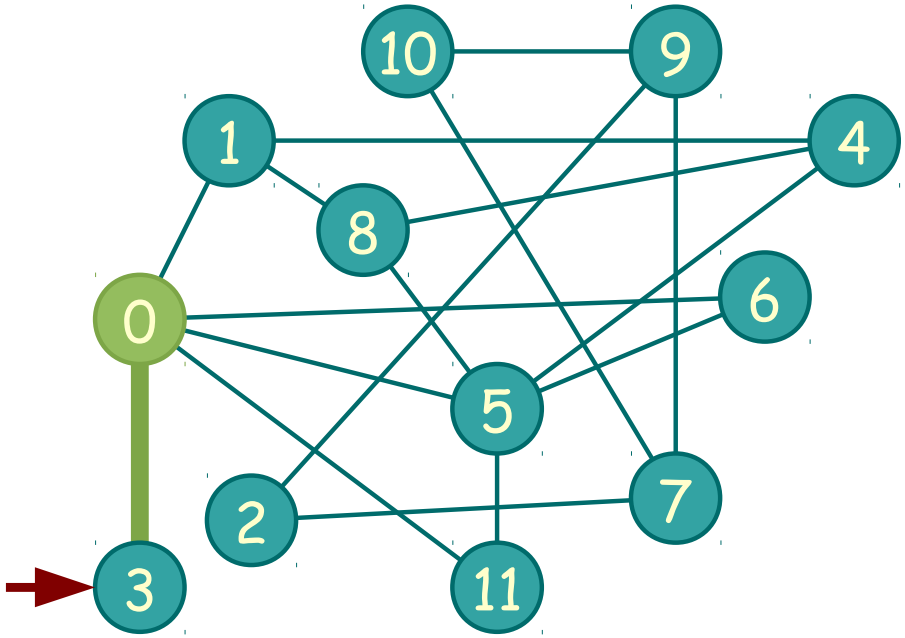
Busca em profundidade: exemplo

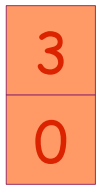


0

pular

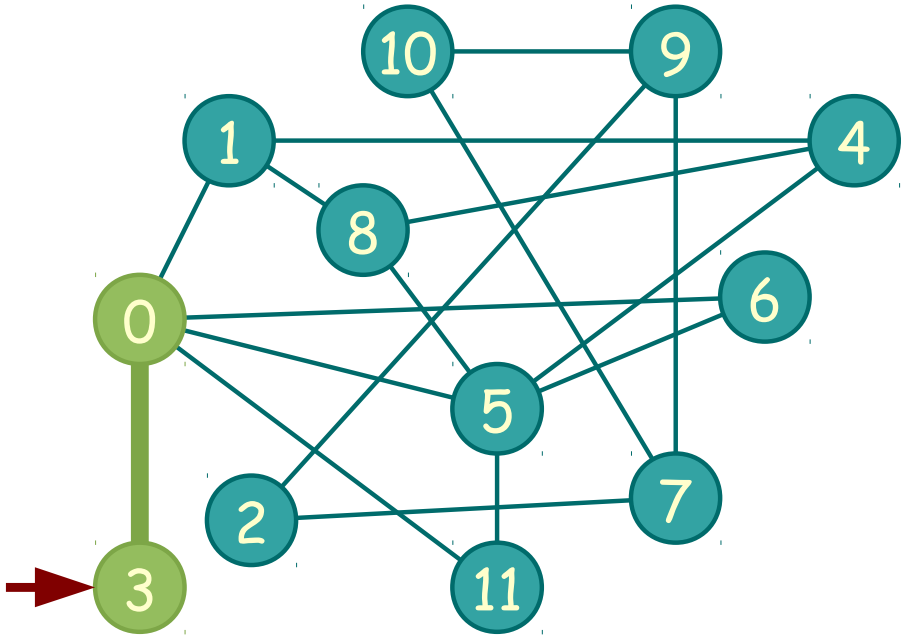
Busca em profundidade: exemplo

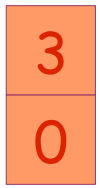




pular

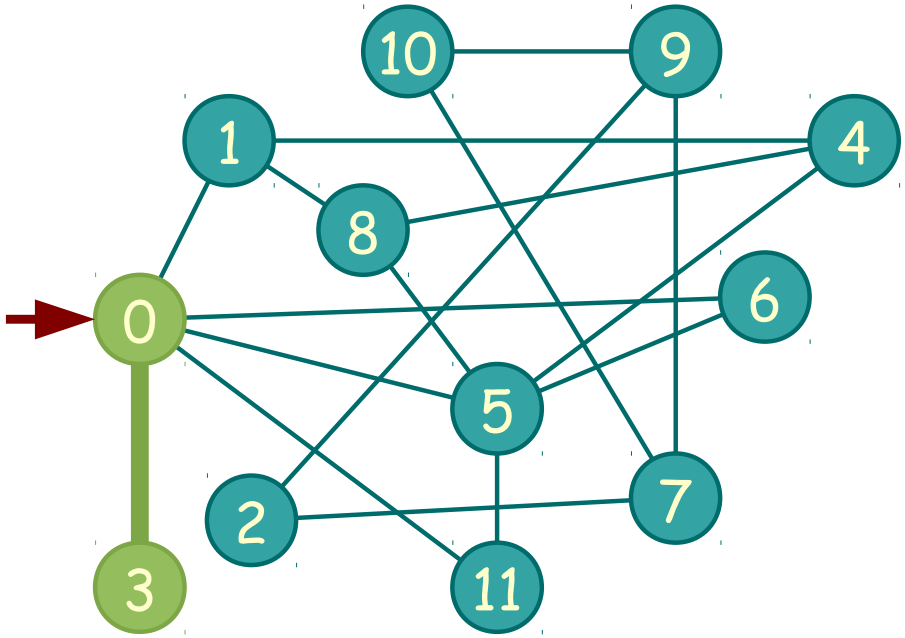
Busca em profundidade: exemplo





pular

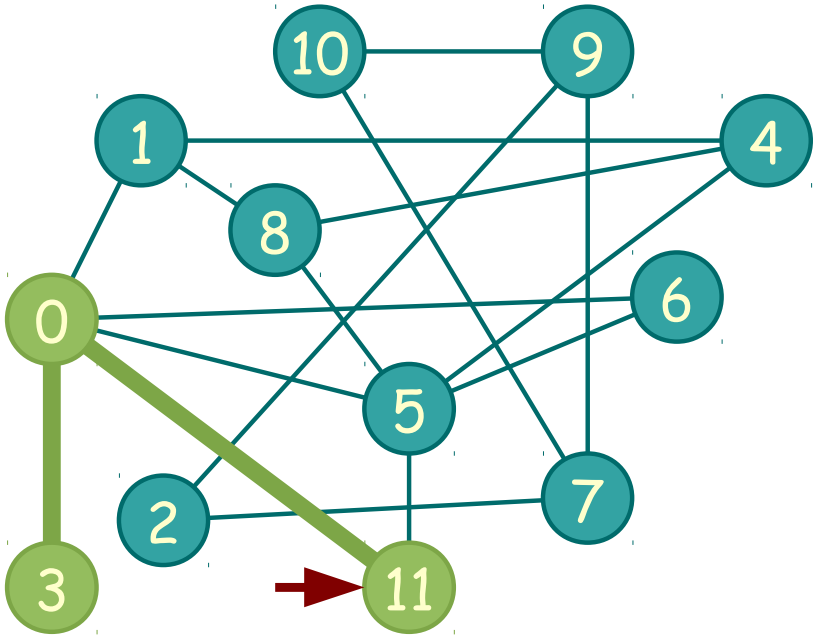
Busca em profundidade: exemplo



0

pular

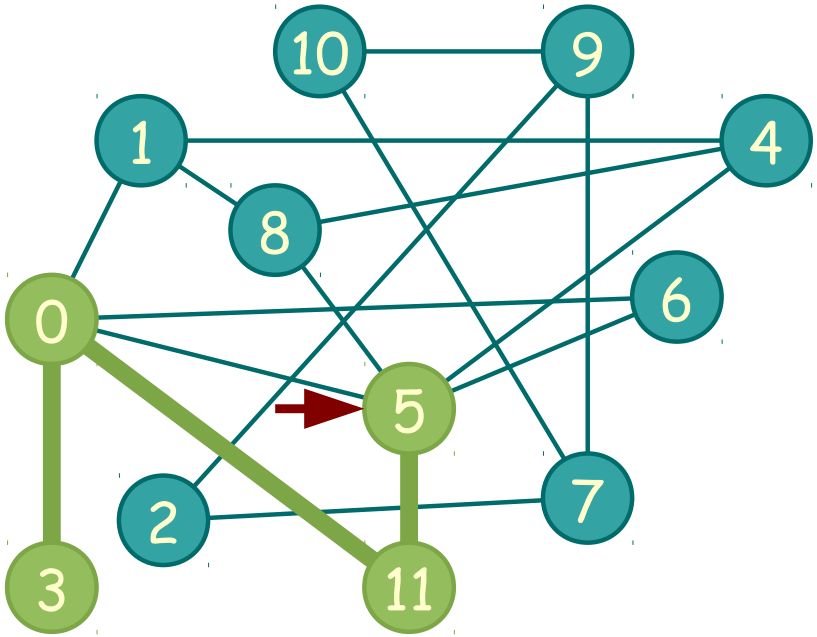
Busca em profundidade: exemplo



11
0

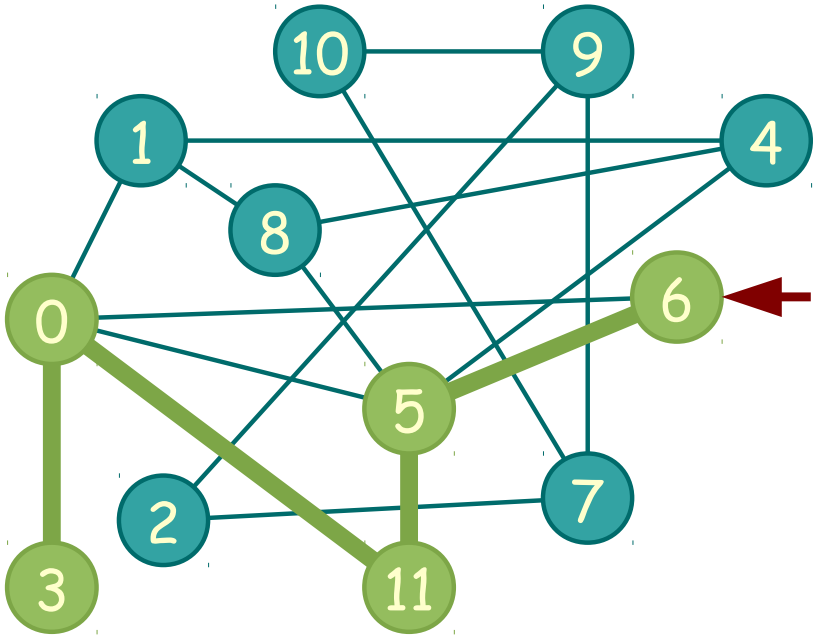
pular

Busca em profundidade: exemplo



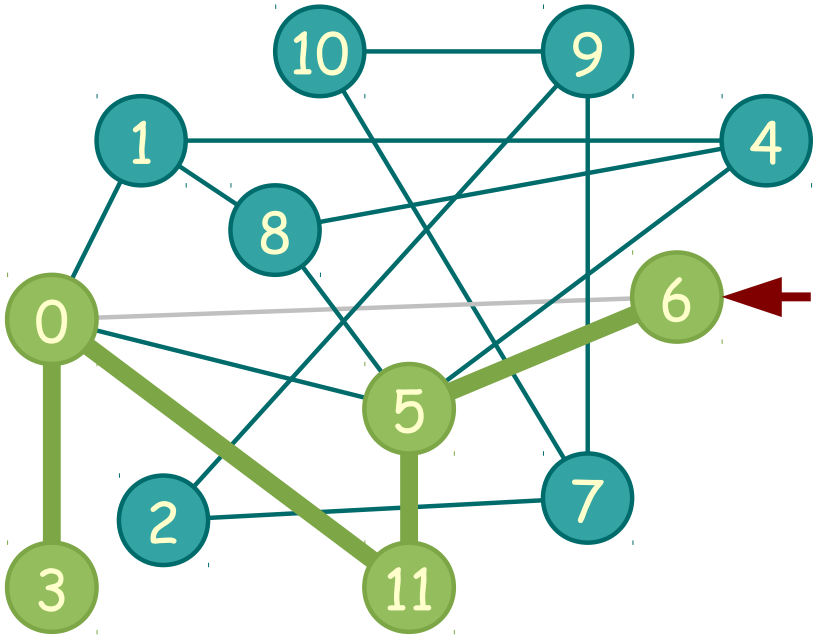
pular

Busca em profundidade: exemplo



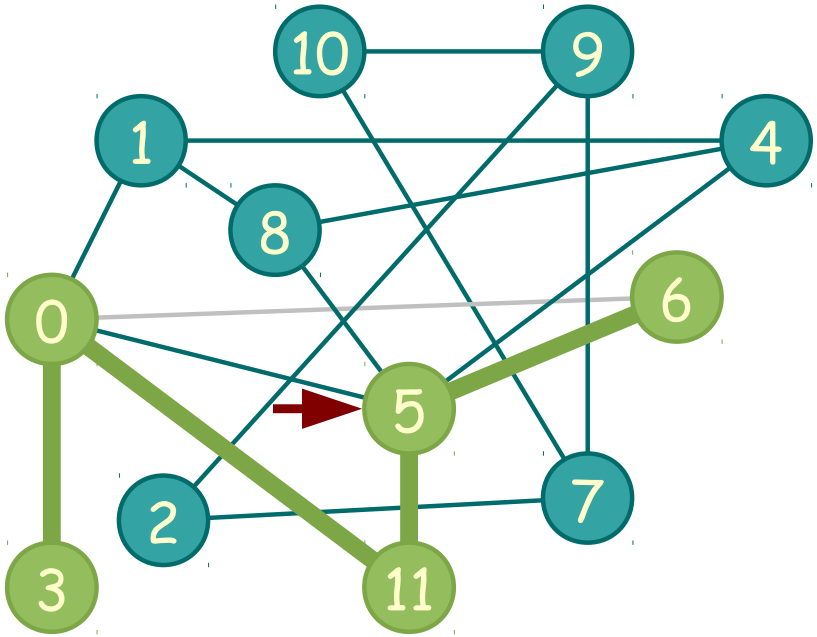
pular

Busca em profundidade: exemplo



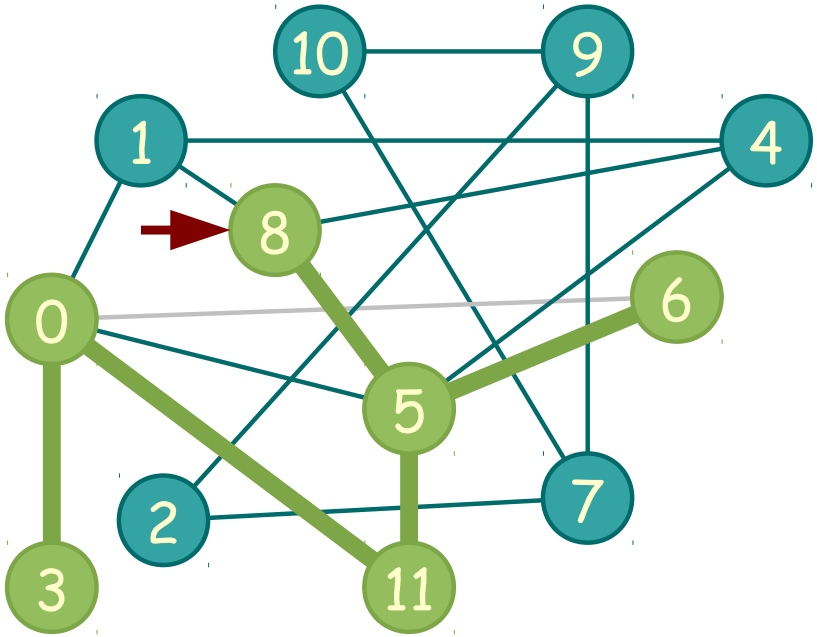
pular

Busca em profundidade: exemplo



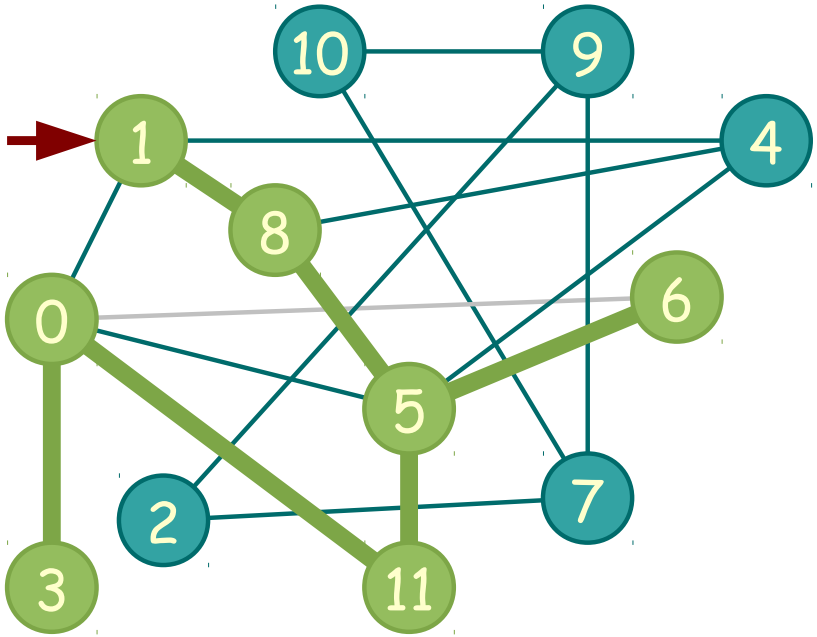
pular

Busca em profundidade: exemplo



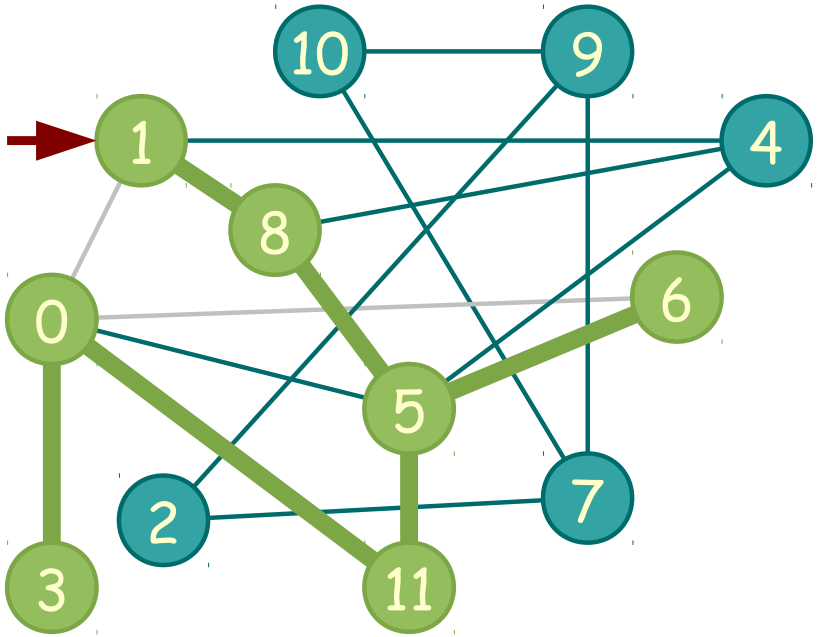
pular

Busca em profundidade: exemplo



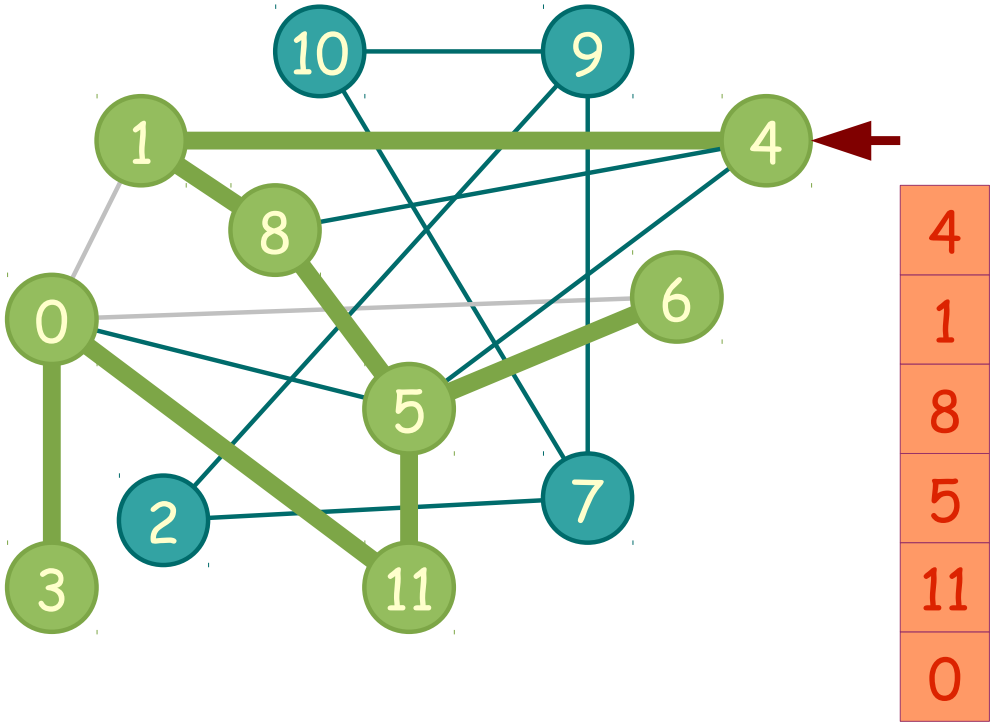
pular

Busca em profundidade: exemplo



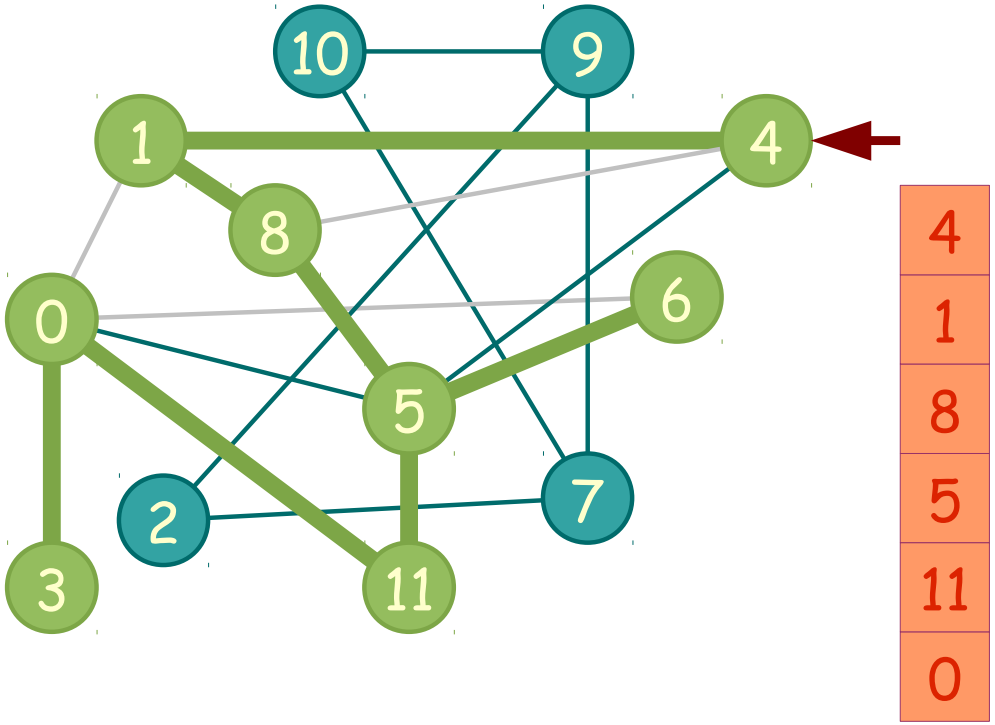
pular

Busca em profundidade: exemplo



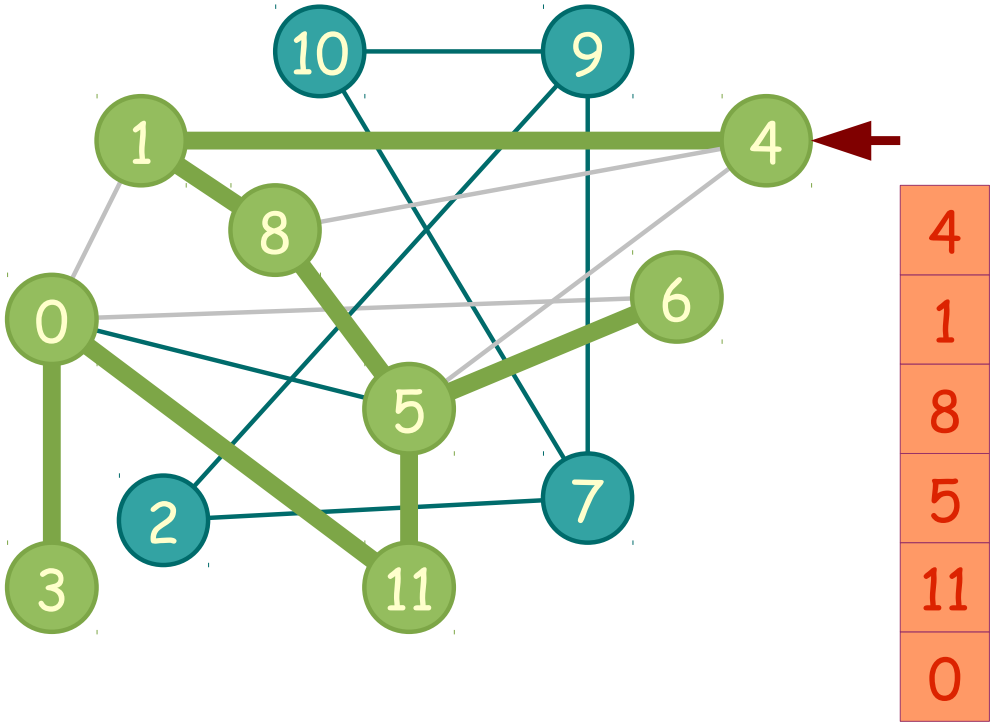
pular

Busca em profundidade: exemplo



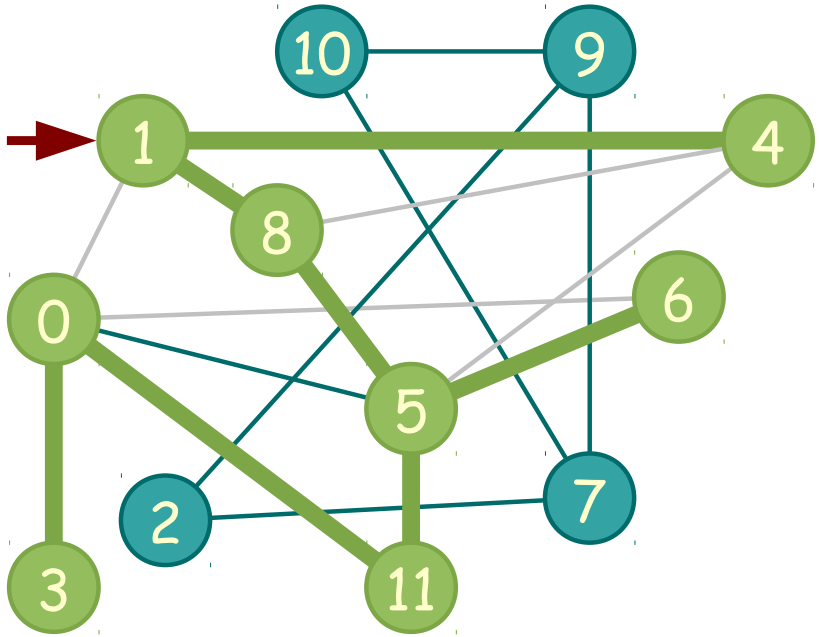
pular

Busca em profundidade: exemplo



pular

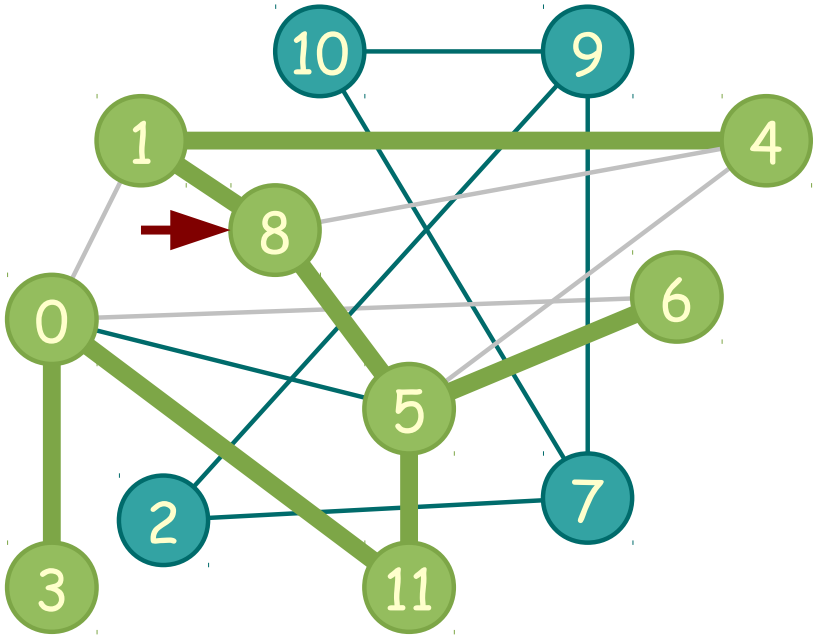
Busca em profundidade: exemplo



1
8
5
11
0

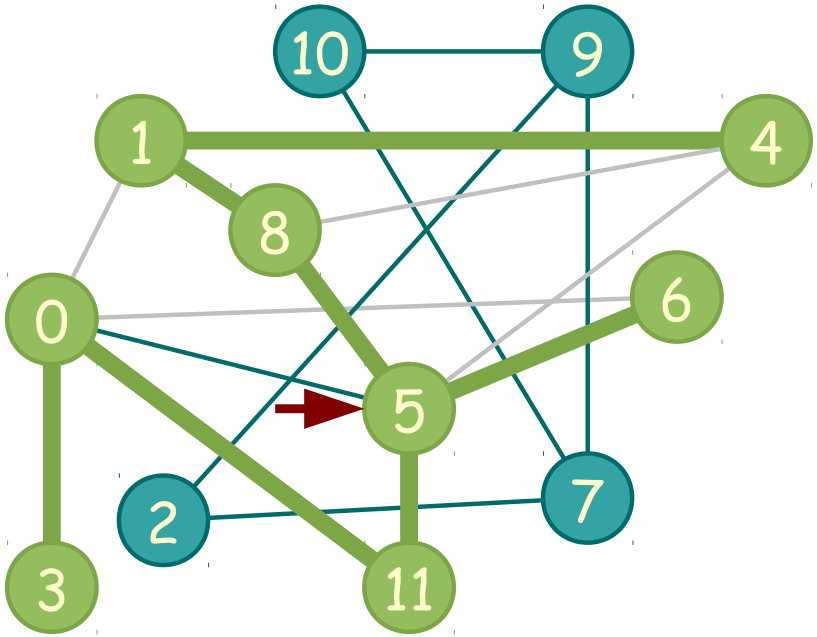
pular

Busca em profundidade: exemplo



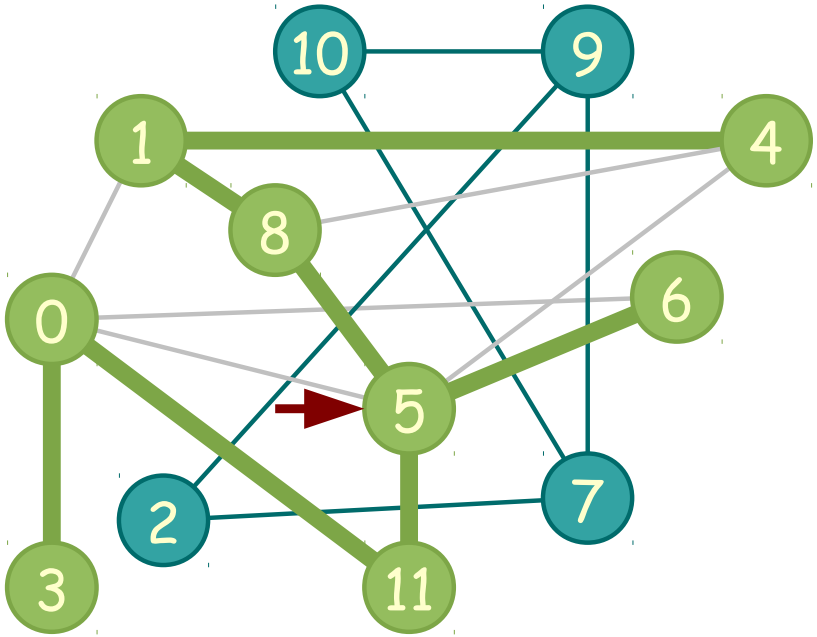
pular

Busca em profundidade: exemplo



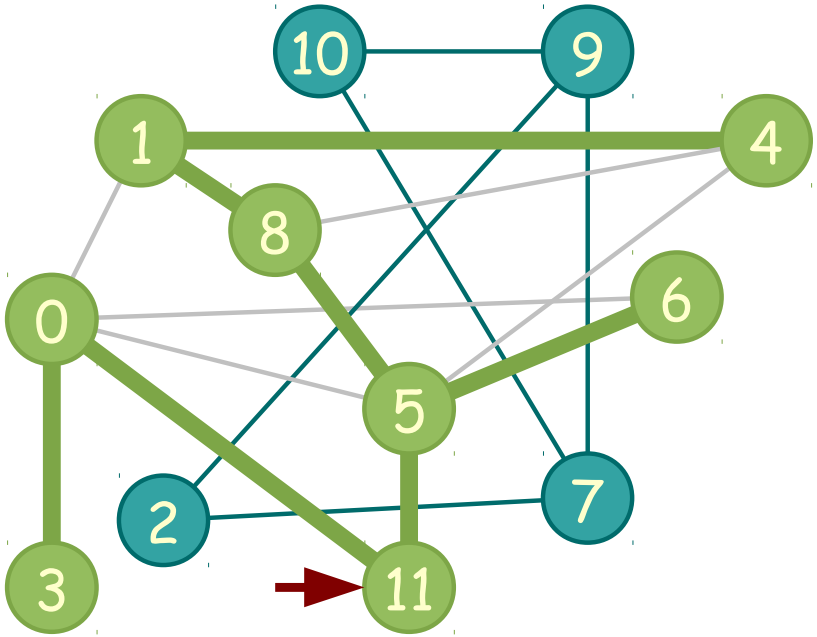
pular

Busca em profundidade: exemplo



pular

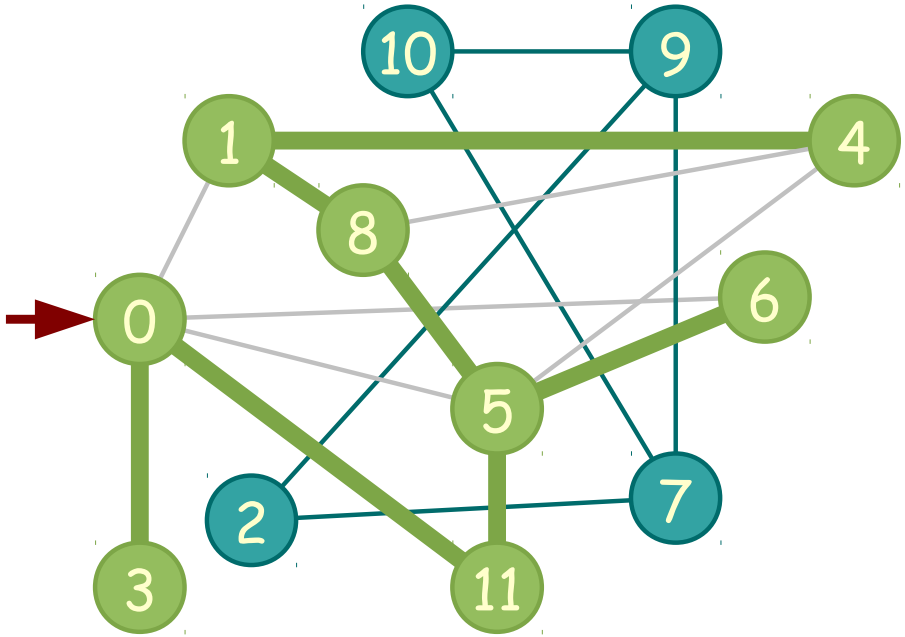
Busca em profundidade: exemplo



11
0

pular

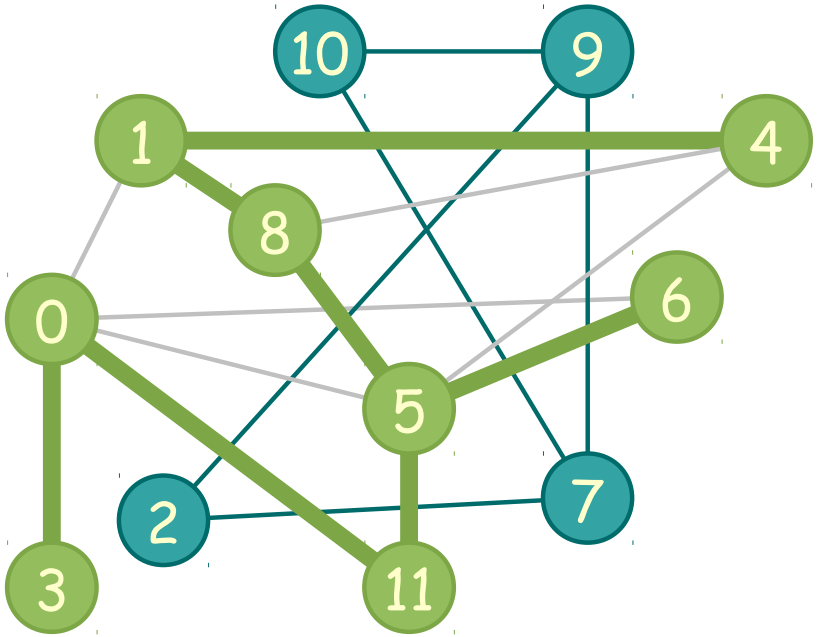
Busca em profundidade: exemplo



0

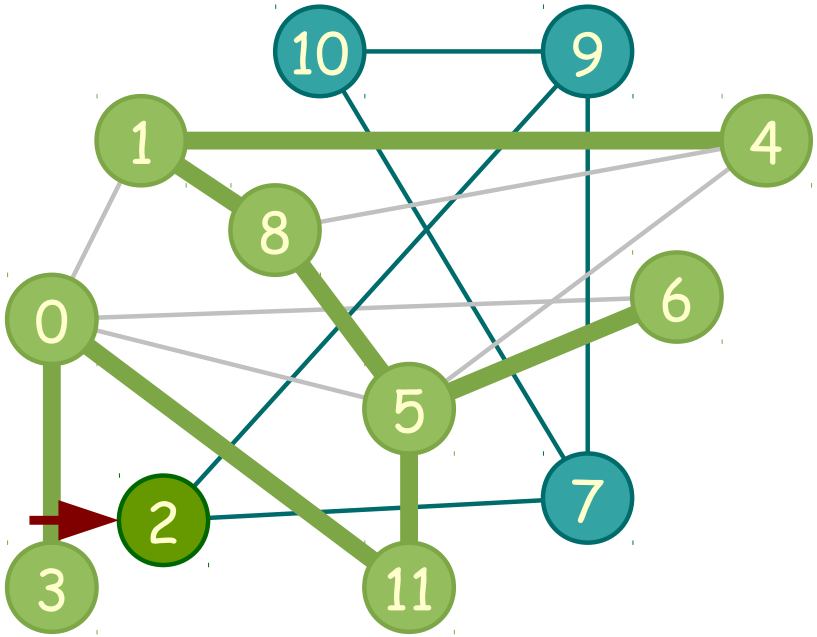
pular

Busca em profundidade: exemplo



pular

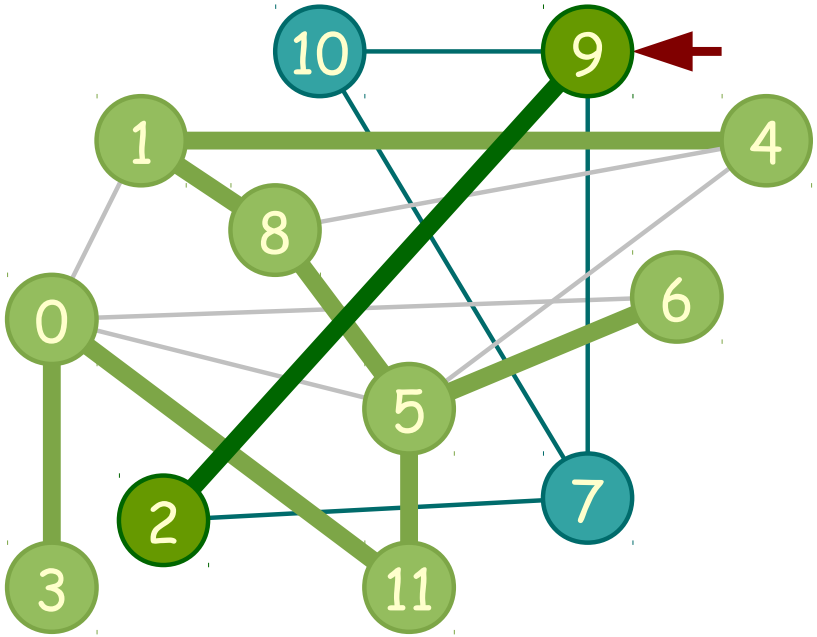
Busca em profundidade: exemplo

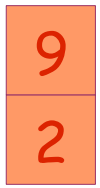


2

pular

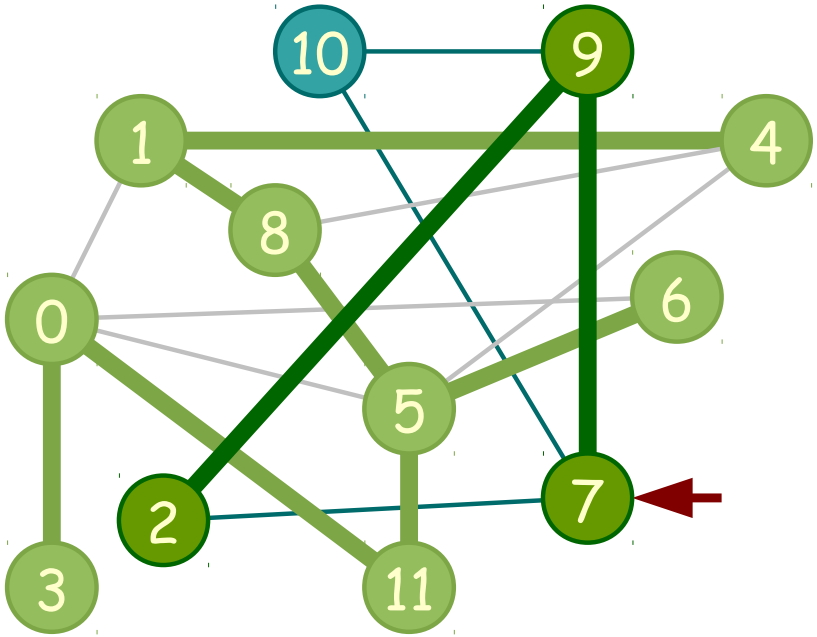
Busca em profundidade: exemplo





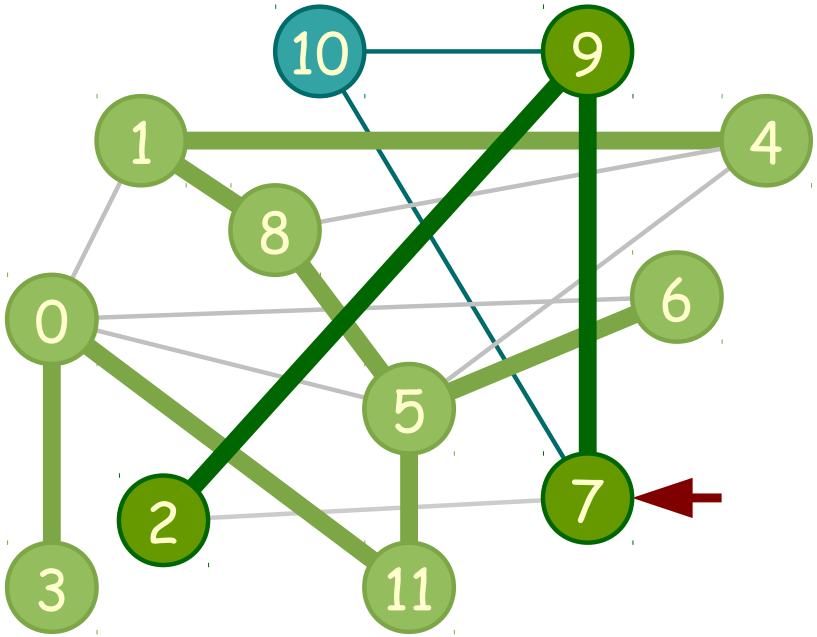
pular

Busca em profundidade: exemplo



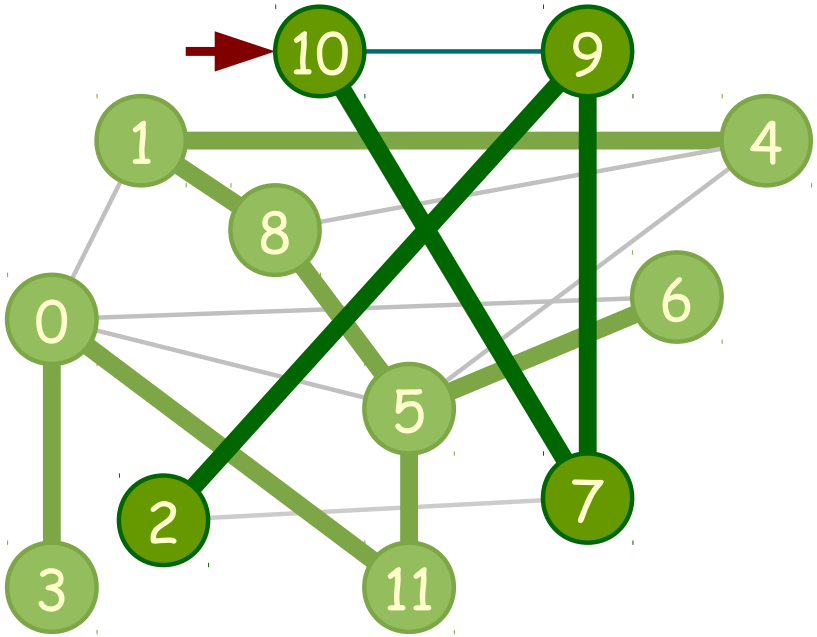
pular

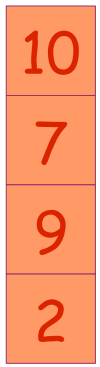
Busca em profundidade: exemplo



pular

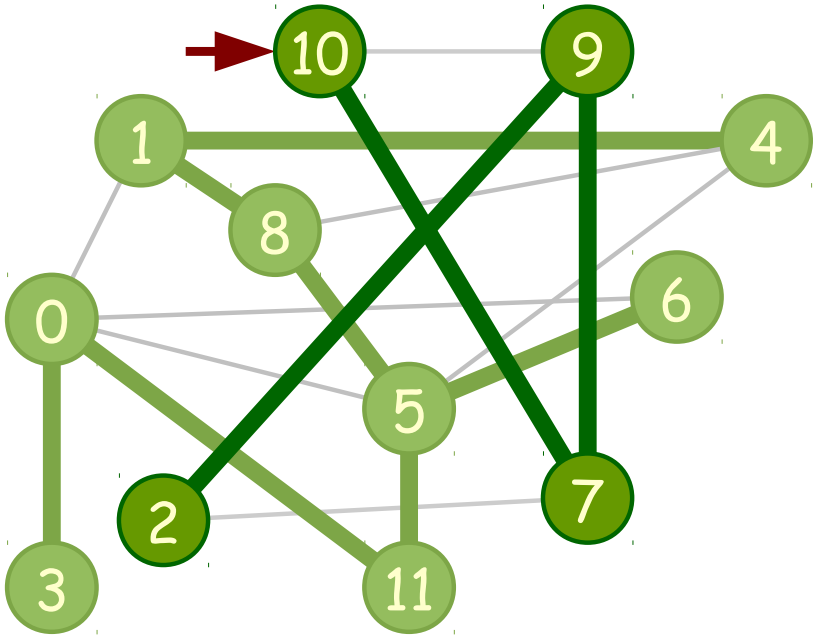
Busca em profundidade: exemplo

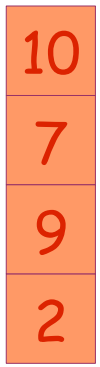




pular

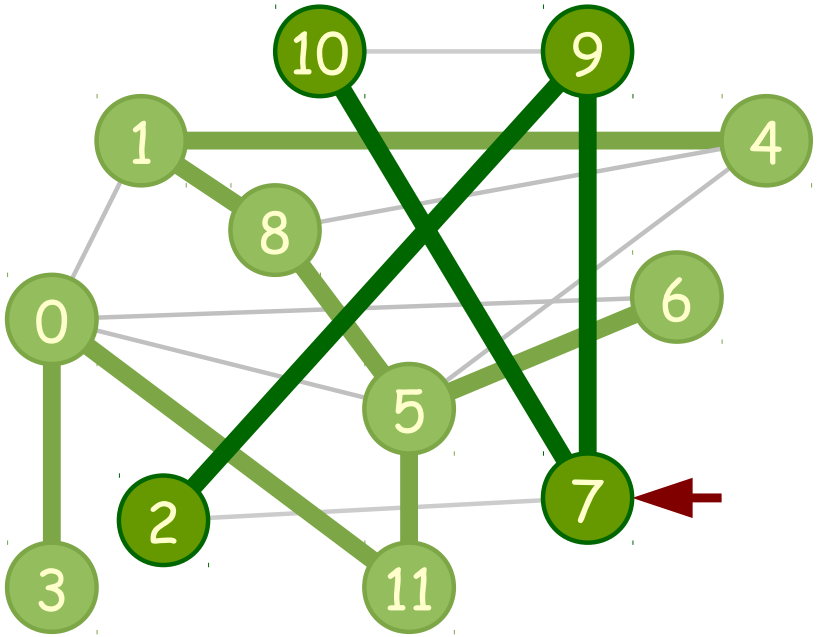
Busca em profundidade: exemplo





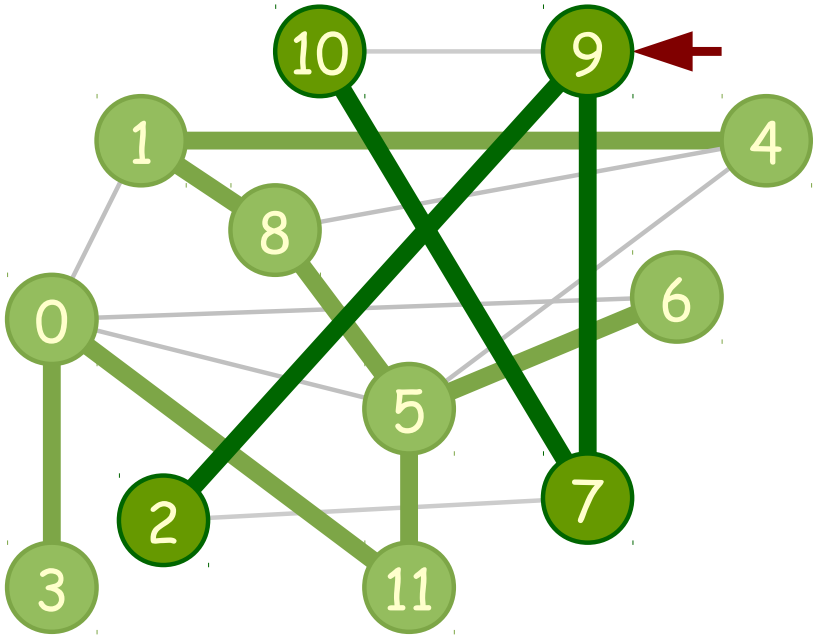
pular

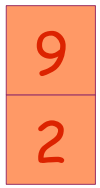
Busca em profundidade: exemplo



pular

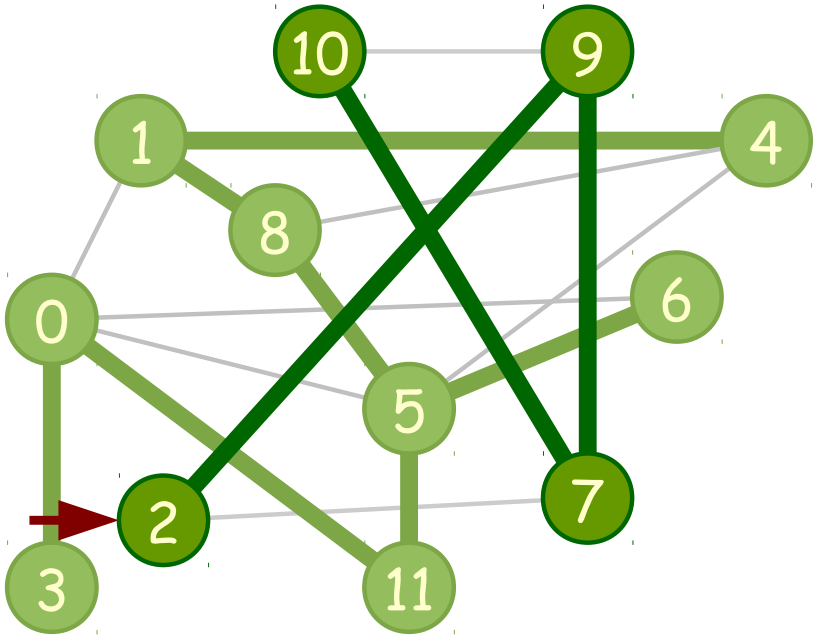
Busca em profundidade: exemplo





pular

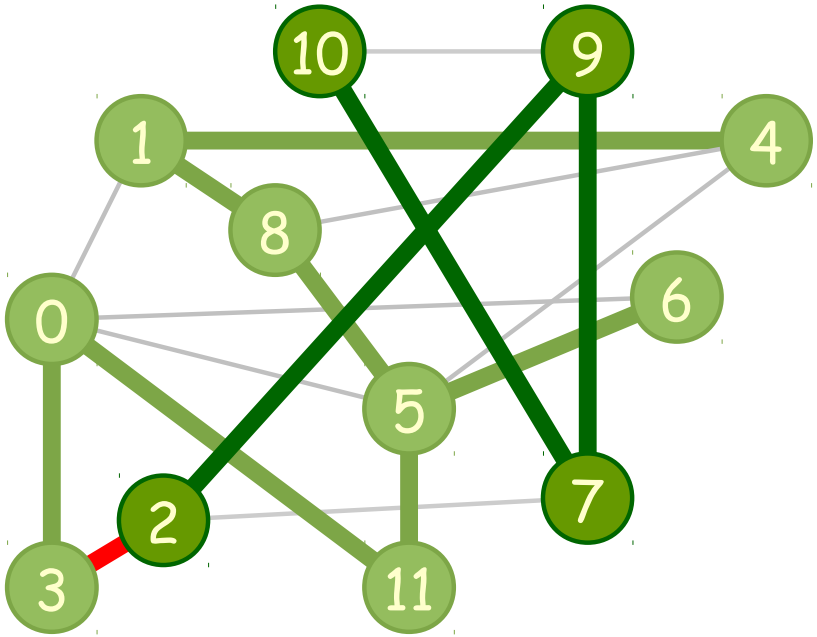
Busca em profundidade: exemplo



2

pular

Busca em profundidade: exemplo



pular

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$
  - percorremos cada vértice: $O(|V|)$

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$
  - percorremos cada vértice: $O(|V|)$
- O procedimento DFS-Visit:

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$
  - percorremos cada vértice: $O(|V|)$
- O procedimento DFS-VISIT:
  - uma chamada por vértice: $O(|V|)$

Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$
  - percorremos cada vértice: $O(|V|)$
- O procedimento DFS-Visit:
  - uma chamada por vértice: $O(|V|)$
  - em cada chamada olhamos as arestas do vértice: $O(|E|)$

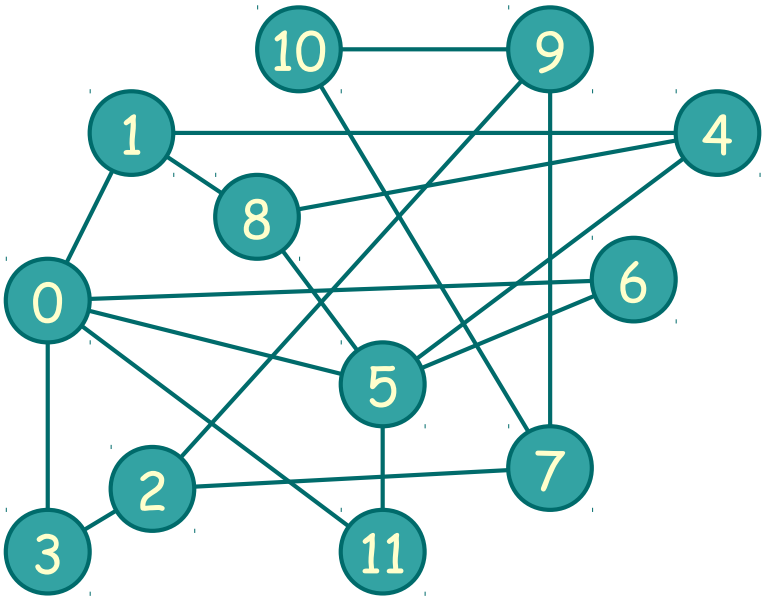
Busca em profundidade: tempo de execução

Complexidade da DFS

- O procedimento DFS:
  - inicializamos cada vértice: $O(|V|)$
  - percorremos cada vértice: $O(|V|)$
- O procedimento DFS-VISIT:
  - uma chamada por vértice: $O(|V|)$
  - em cada chamada olhamos as arestas do vértice: $O(|E|)$

TOTAL: $O(|V| + |E|)$

Um problema simples



Problema

Qual o menor caminho entre os vértices 0 e 9?

Busca em largura

A busca em largura ou breadth-first search (BFS) a partir de um certo vértice s visita os vértices do grafo em ordem de distância a s.

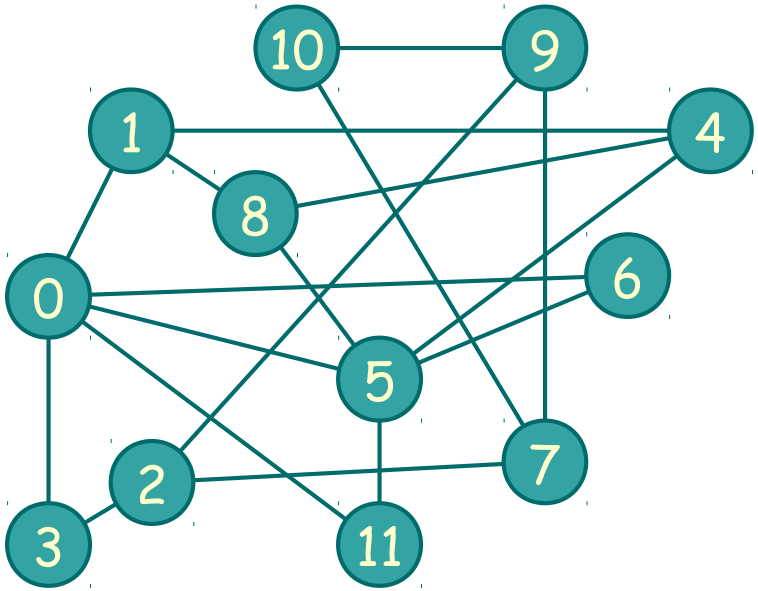
Ideia geral

1. Cada vértice mantém um estado:
   - não visitado: não descoberto pela busca ainda
   - marcado: já descoberto pela busca
2. Enfileira vértice s
3. Para cada vértice v da fila:
   - marca v como visitado
   - enfileira vizinhos de v não marcados

Busca em largura: pseudocódigo

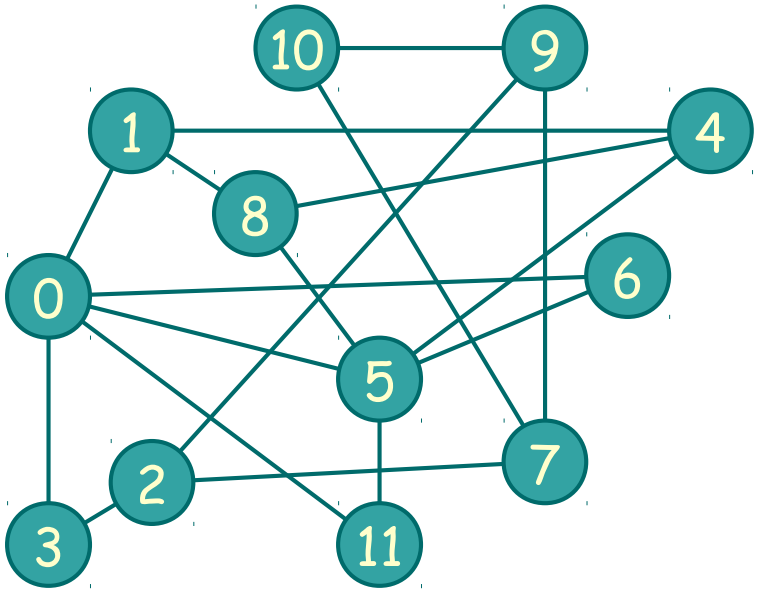
```
BFS(G, s)
 1  for each vertex u ∈ V
 2        u.d = ∞
 3        u.π = NULL
 4        u.marked = FALSE
 5  queue Q = ∅
 6  s.d = 0
 7  s.marked = TRUE
 8  ENQUEUE(Q, s)
 9  while Q ≠ ∅
10        u = DEQUEUE(Q)
11        for each v ∈ Adj(u)
12              if v.marked == FALSE
13                    v.d = u.d + 1
14                    v.π = u
15                    ENQUEUE(Q, v)
16                    v.marked = TRUE
```

Busca em largura: exemplo



pular

Busca em largura: exemplo

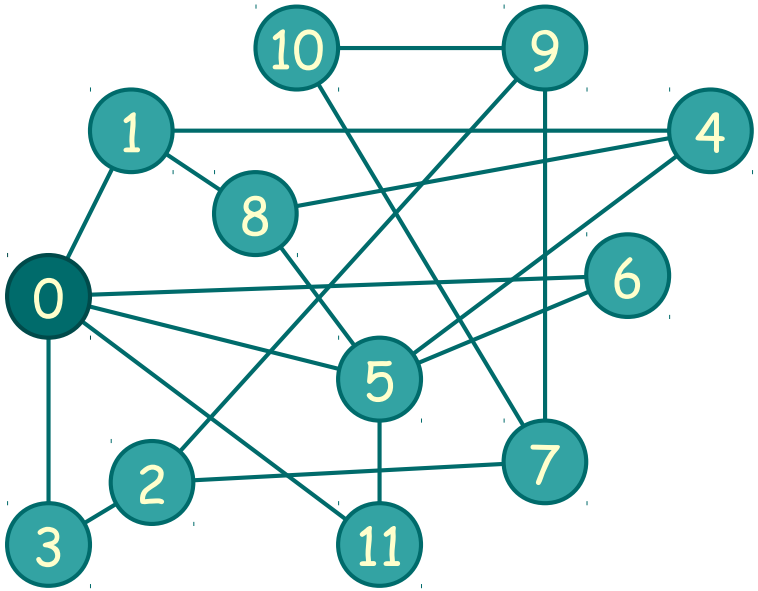


fila:

distância:

pular

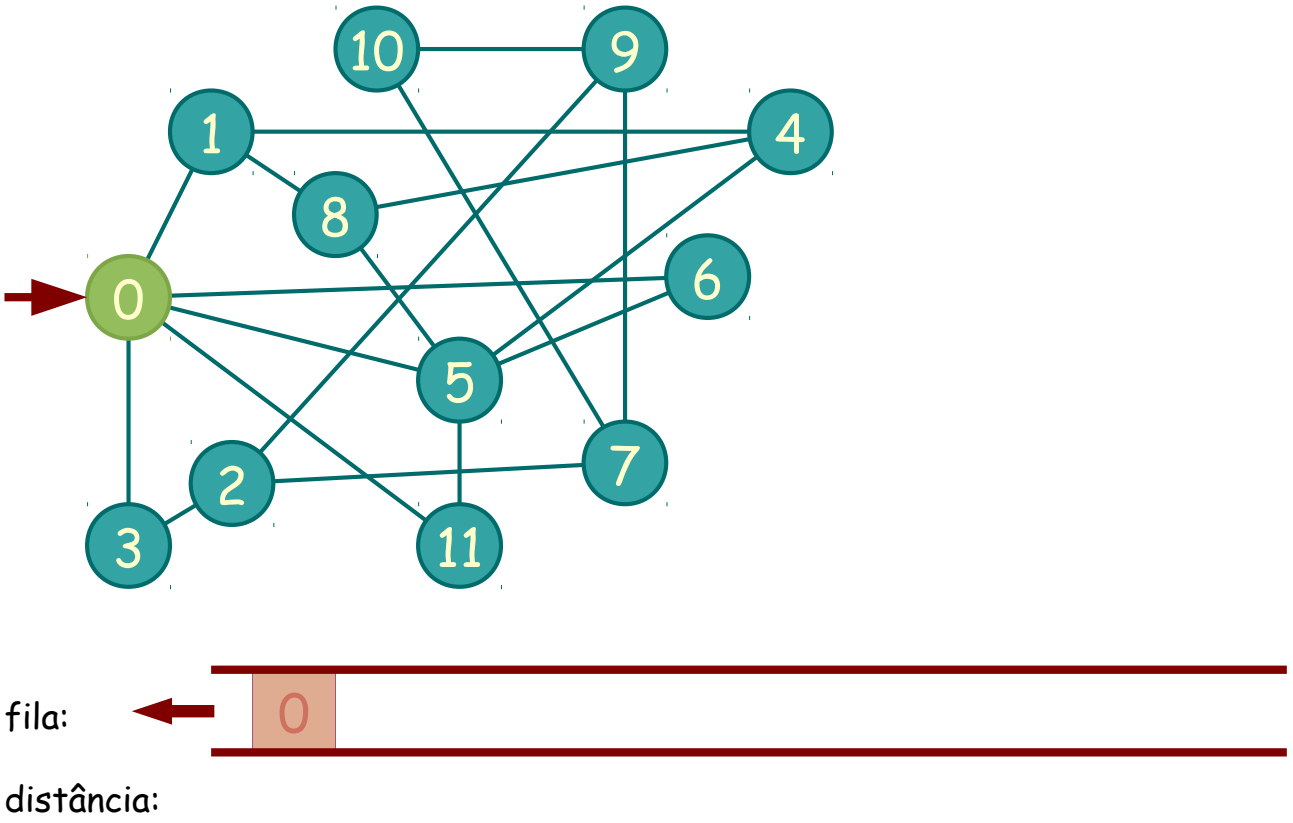
Busca em largura: exemplo



distância:

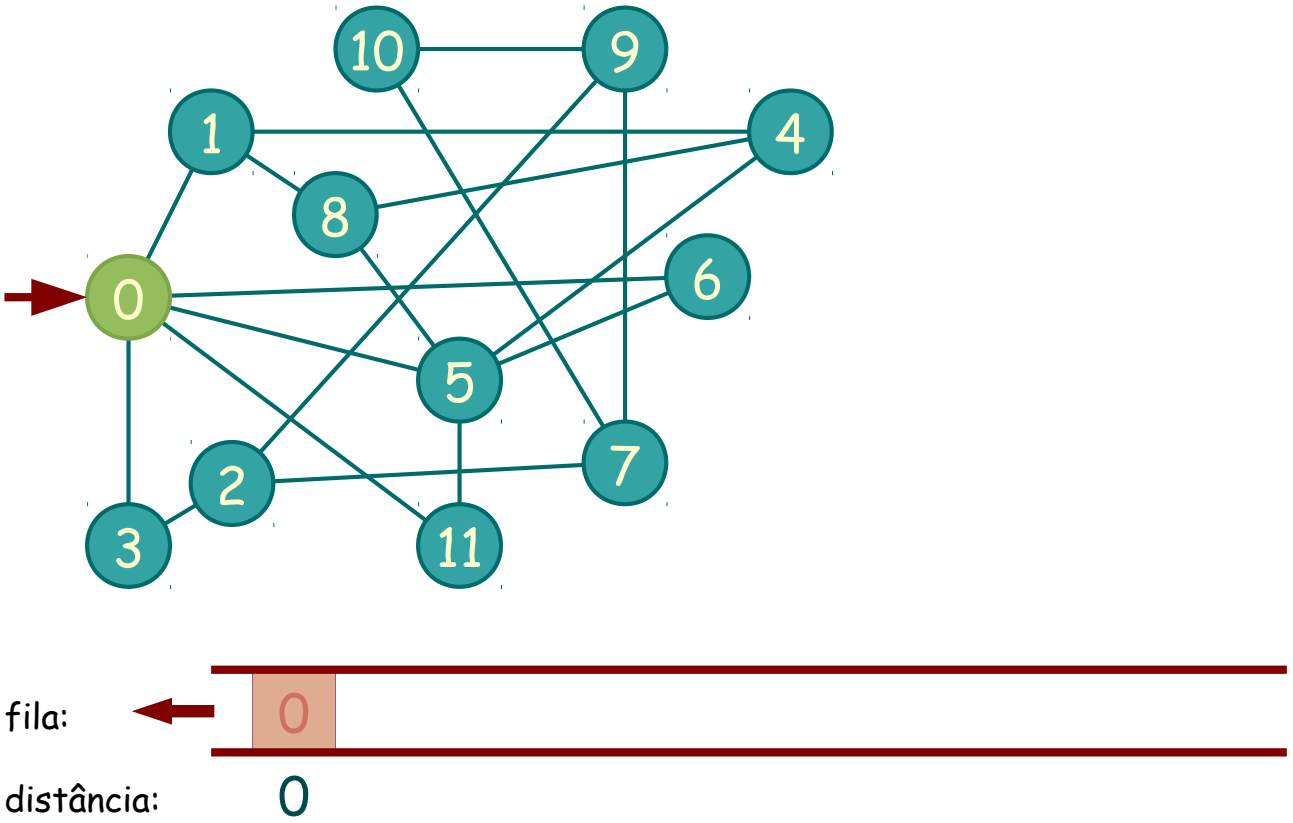
pular

Busca em largura: exemplo



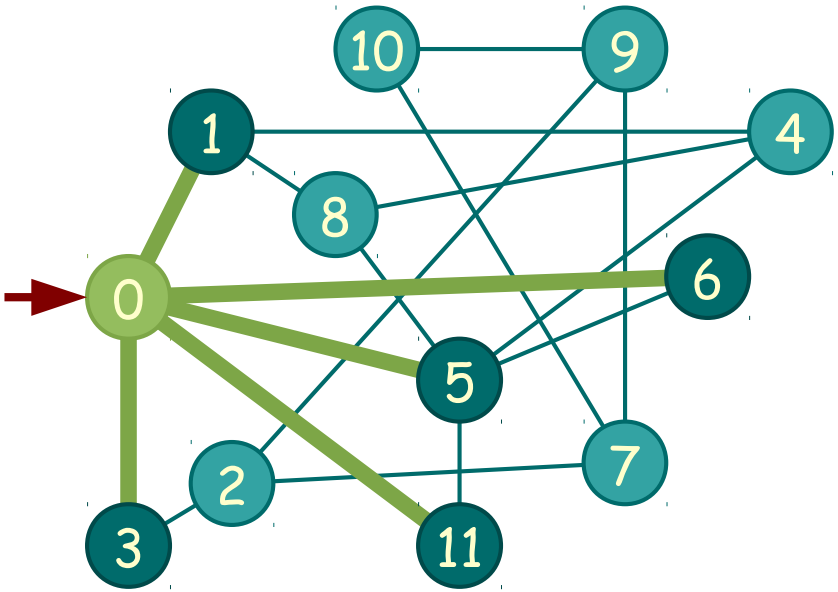
pular

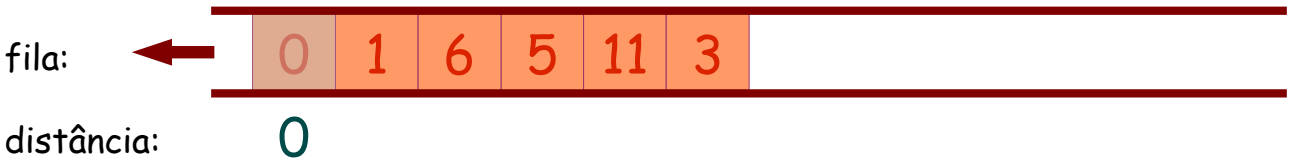
Busca em largura: exemplo



pular

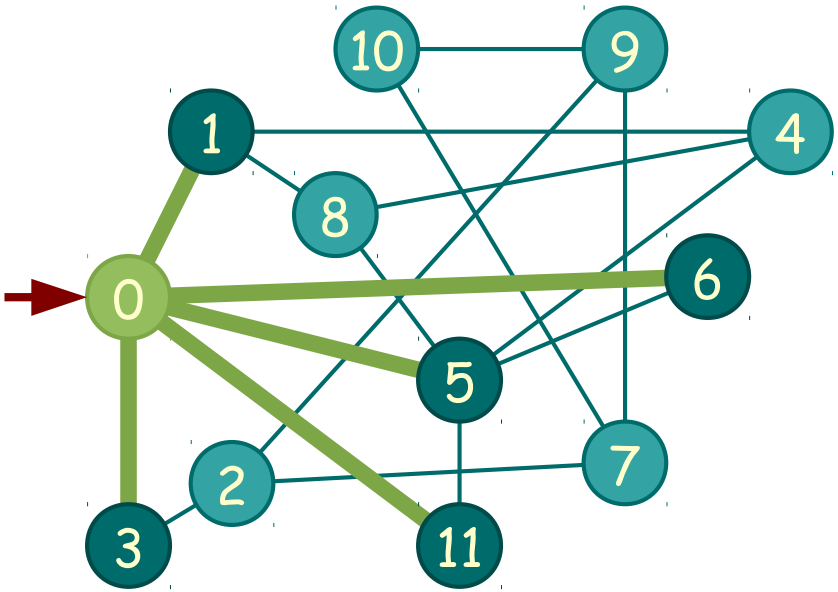
Busca em largura: exemplo

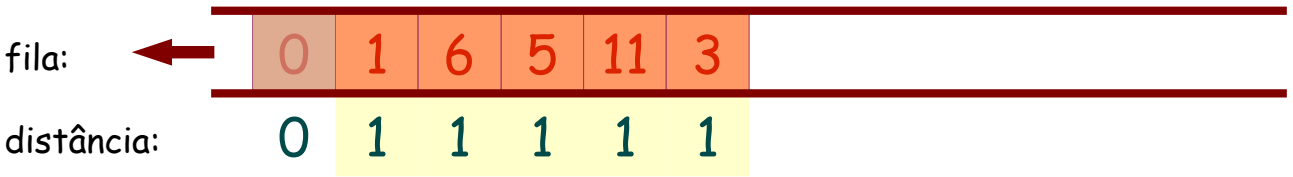




pular

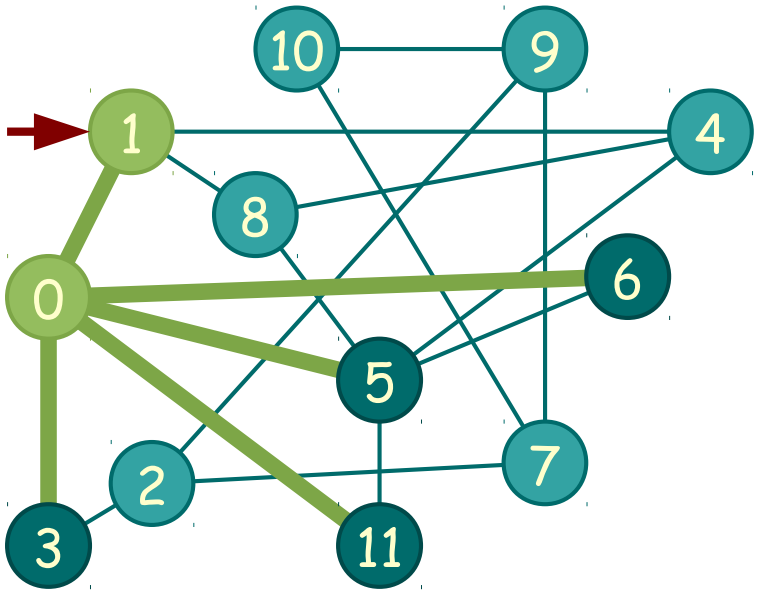
Busca em largura: exemplo

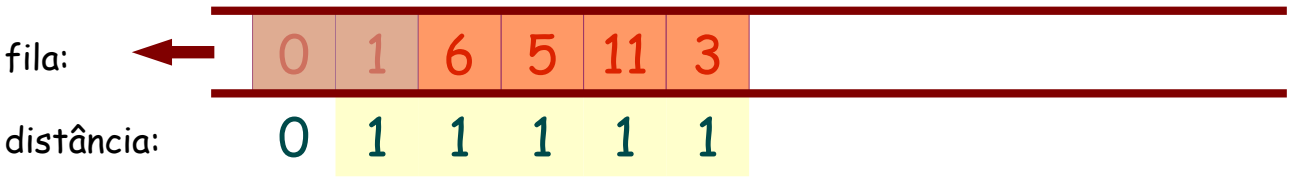




pular

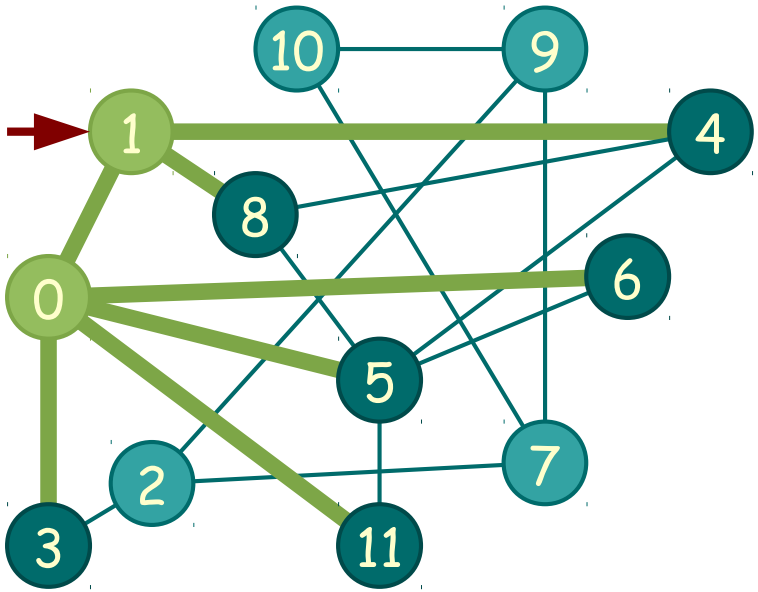
Busca em largura: exemplo

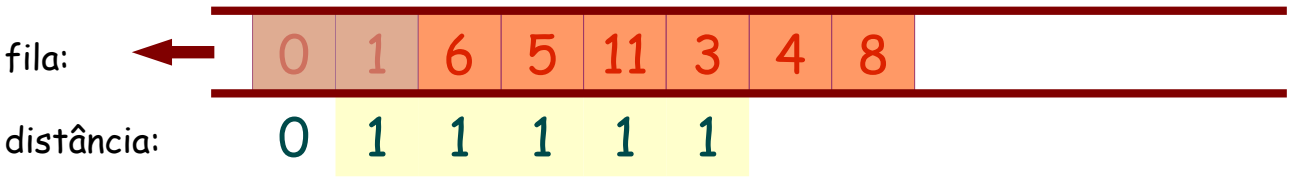




pular

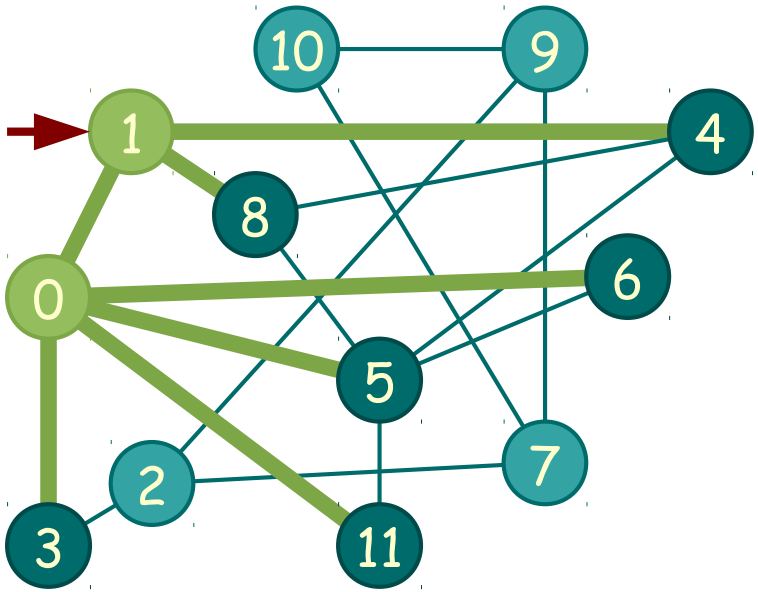
Busca em largura: exemplo





pular

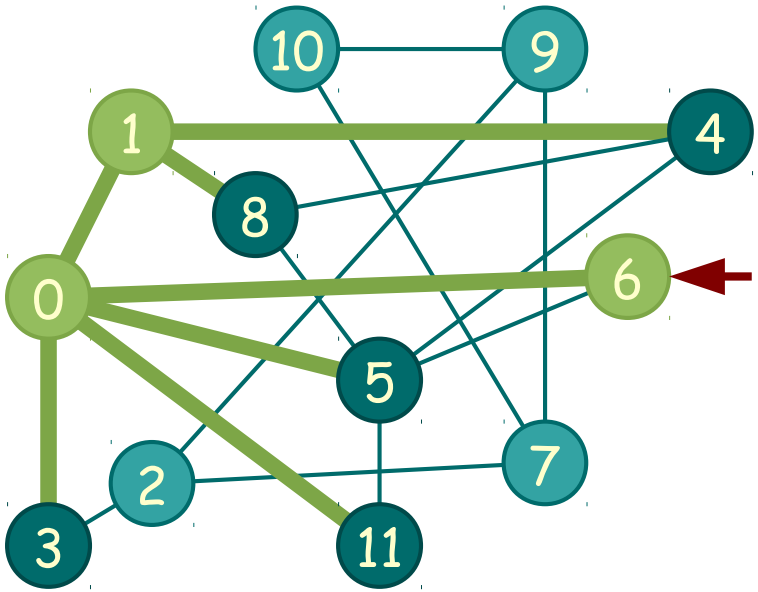
Busca em largura: exemplo

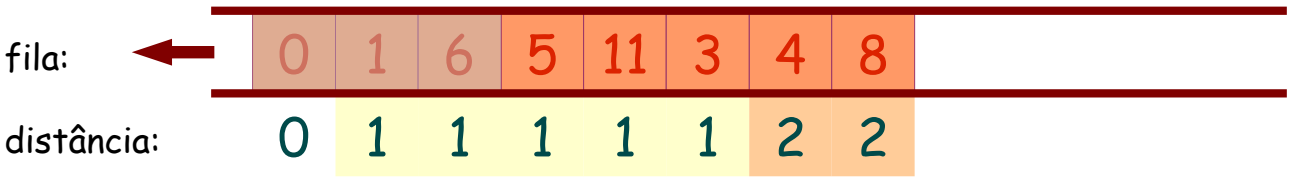


fila:	0	1	6	5	11	3	4	8
distância:	0	1	1	1	1	1	2	2

pular

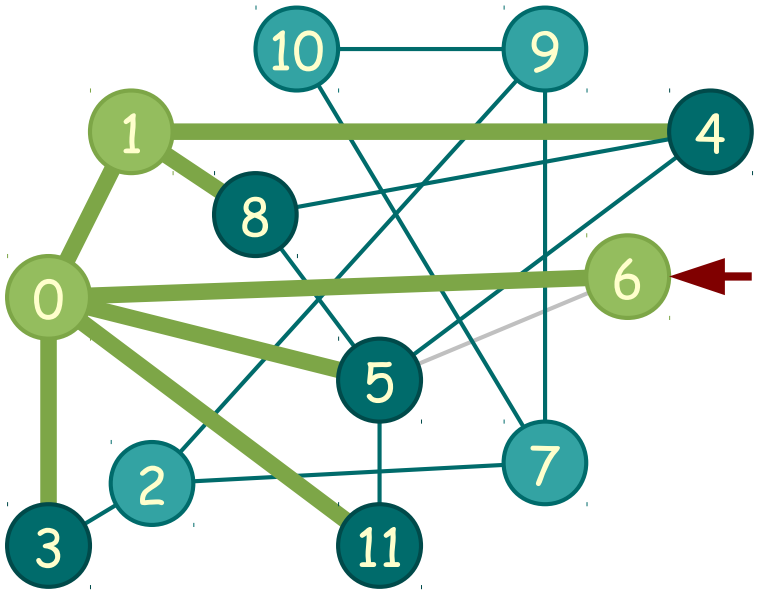
Busca em largura: exemplo

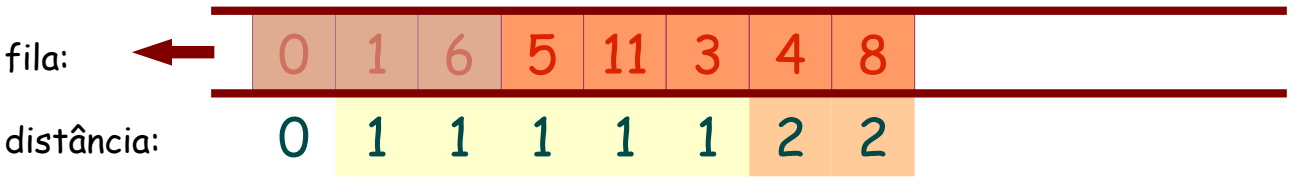




pular

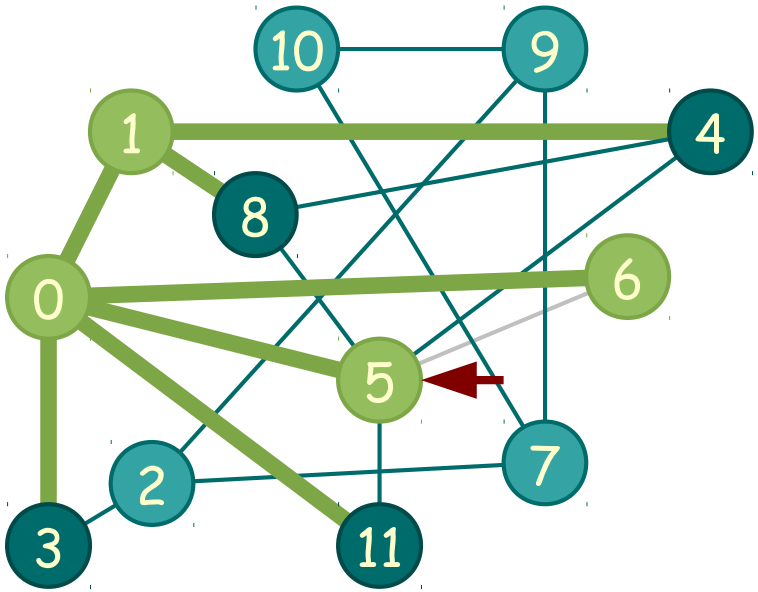
Busca em largura: exemplo

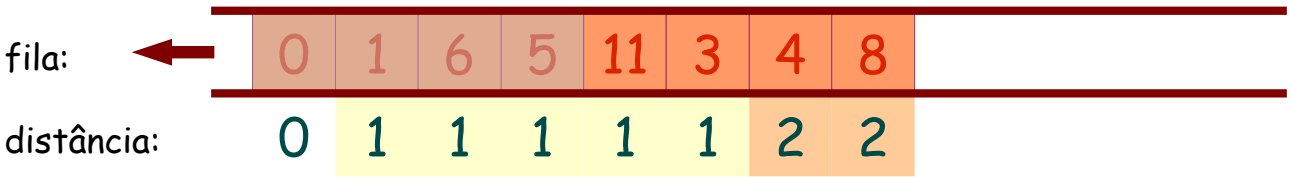




pular

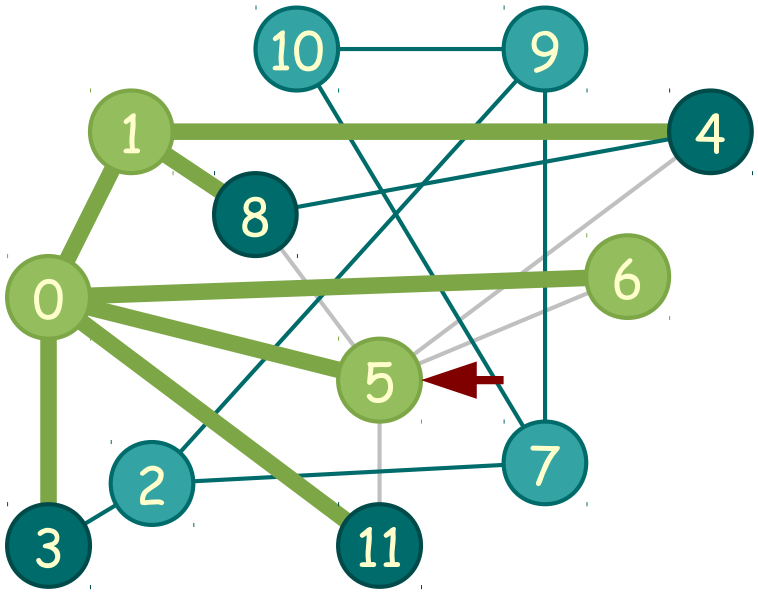
Busca em largura: exemplo

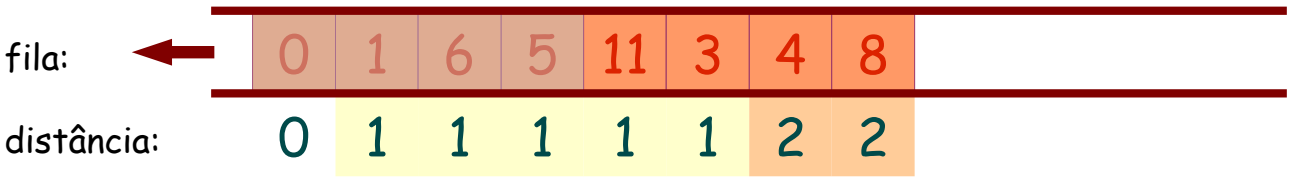




pular

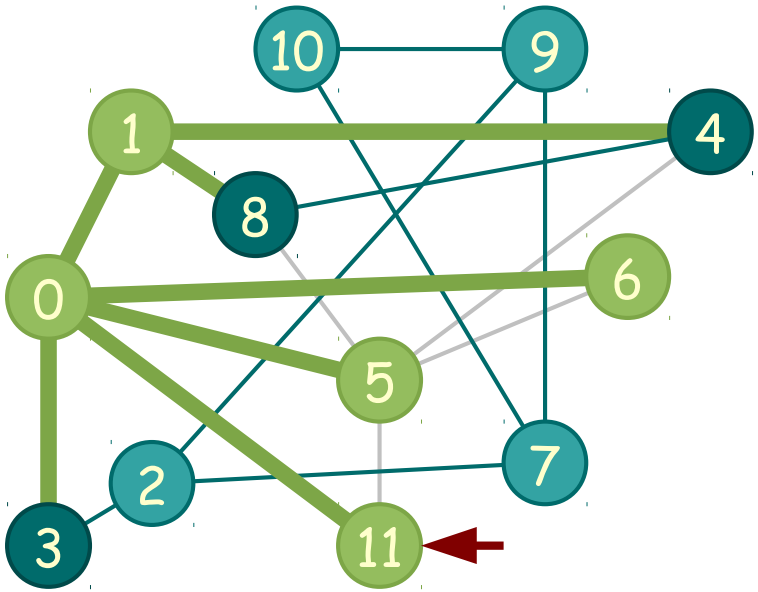
Busca em largura: exemplo

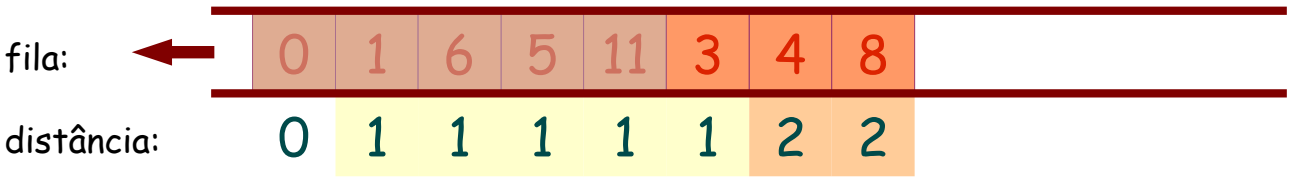




pular

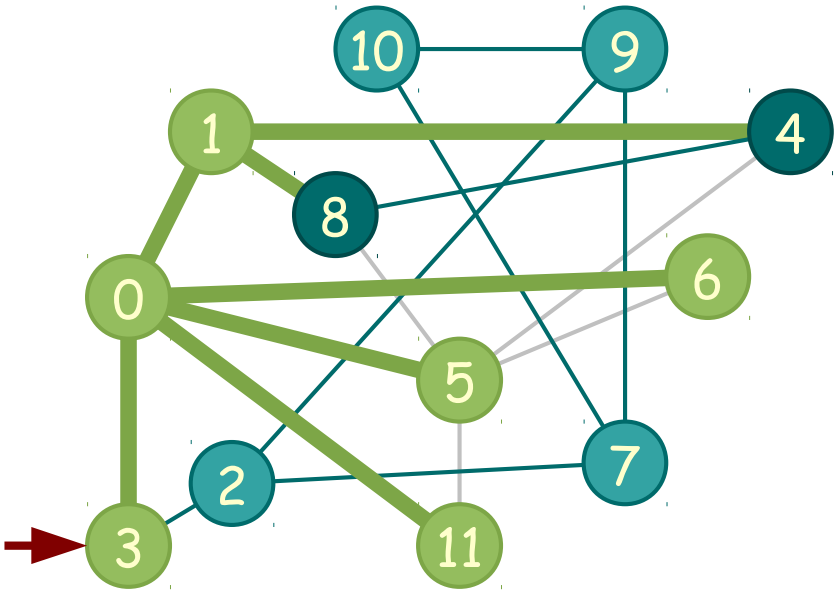
Busca em largura: exemplo

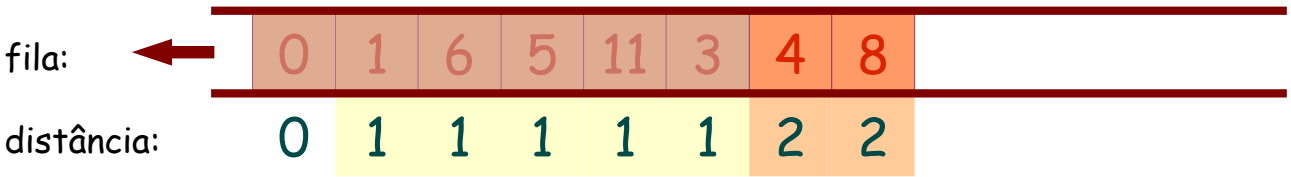




pular

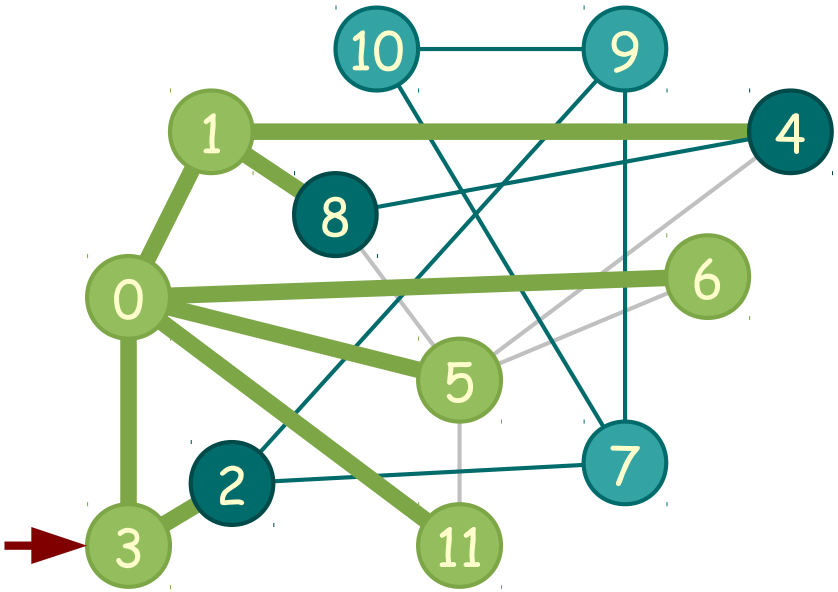
Busca em largura: exemplo

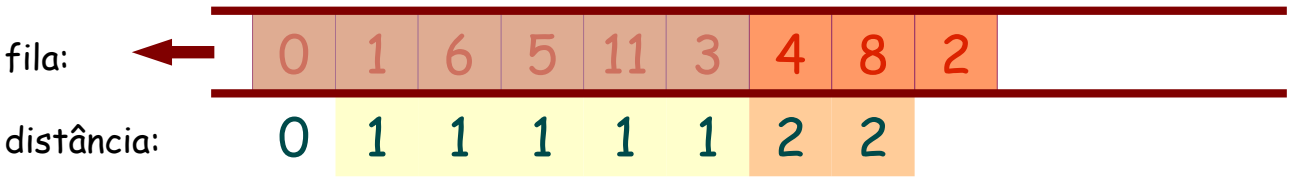




pular

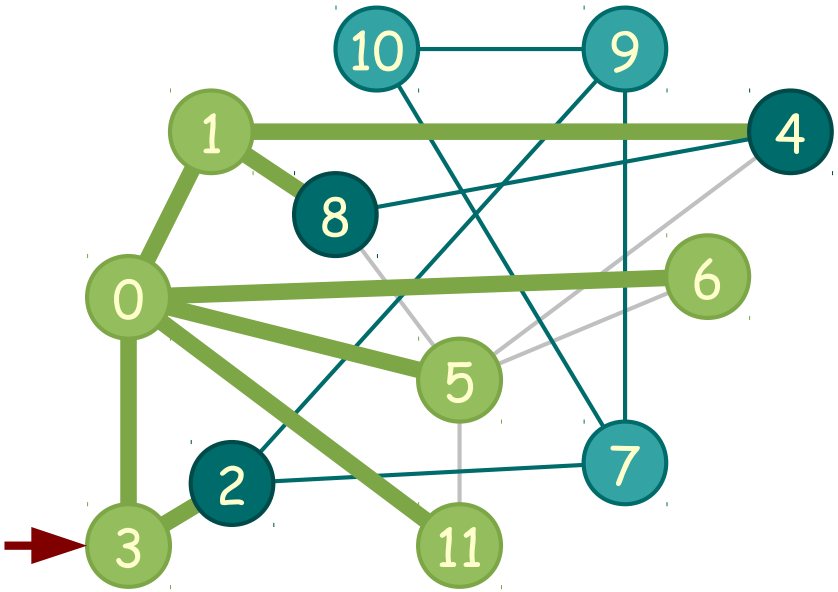
Busca em largura: exemplo

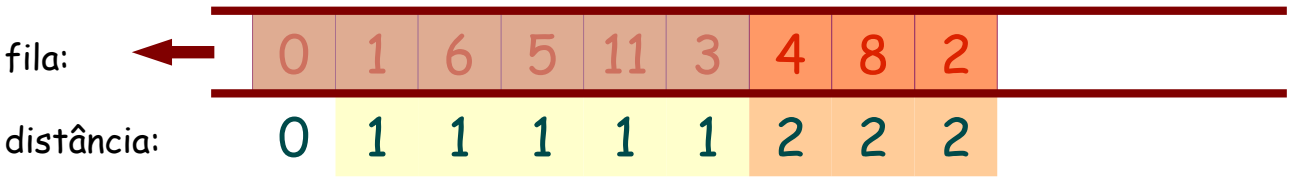




pular

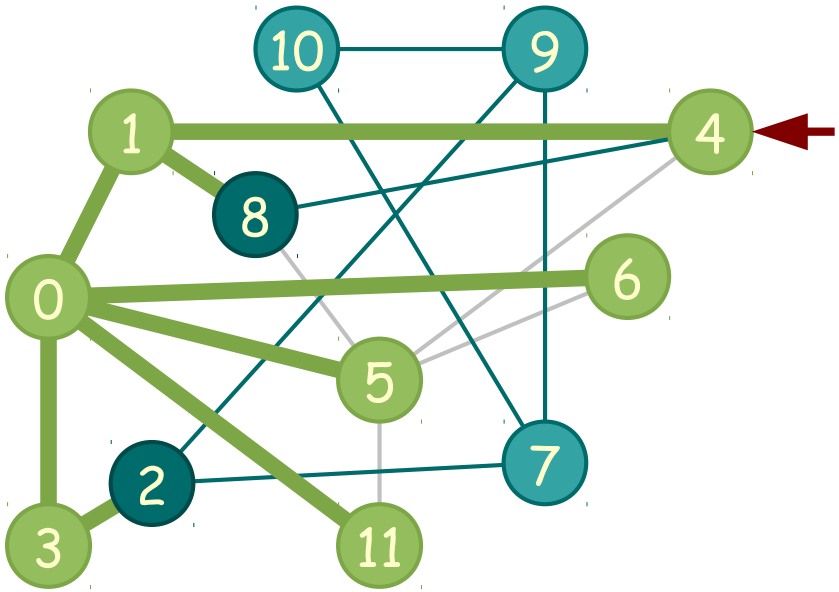
Busca em largura: exemplo

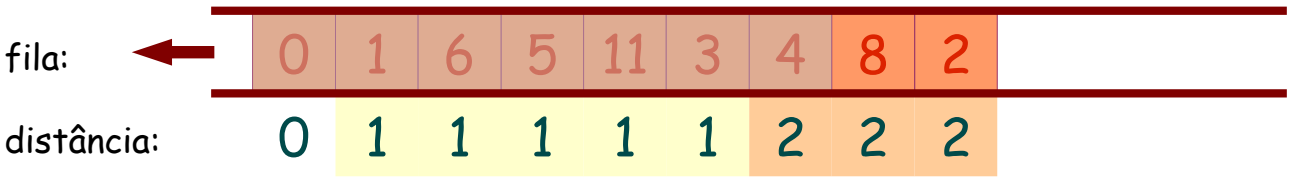




pular

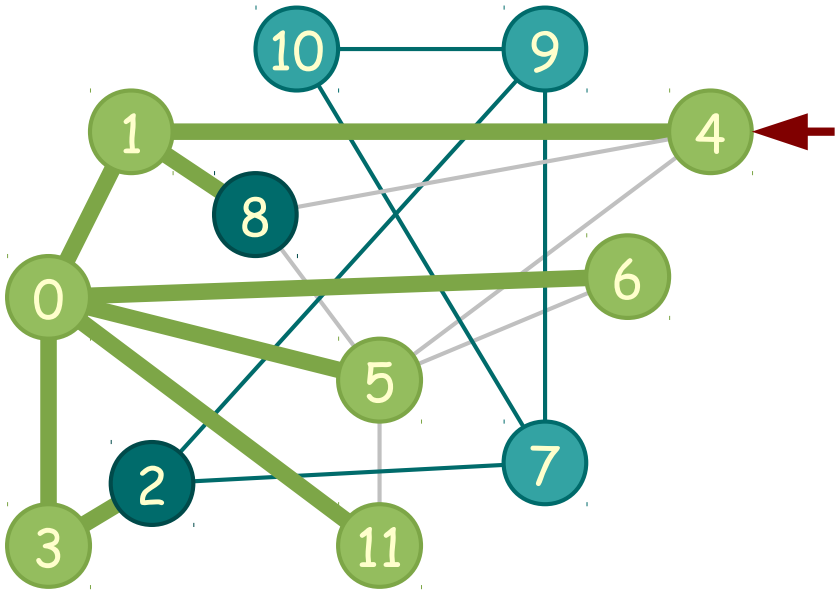
Busca em largura: exemplo

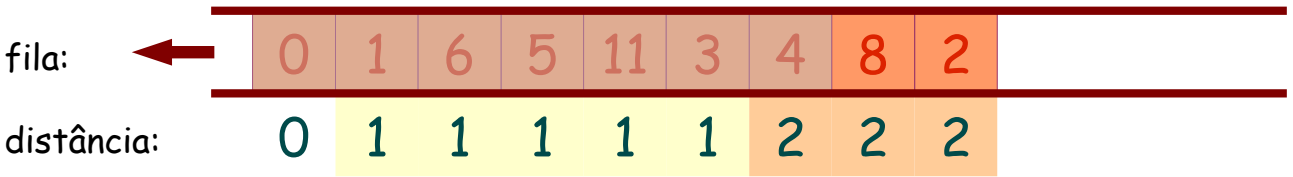




pular

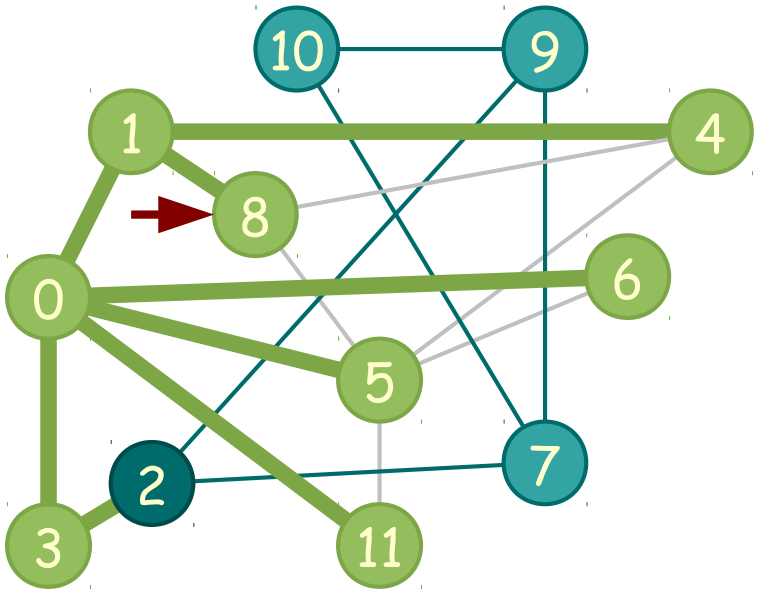
Busca em largura: exemplo

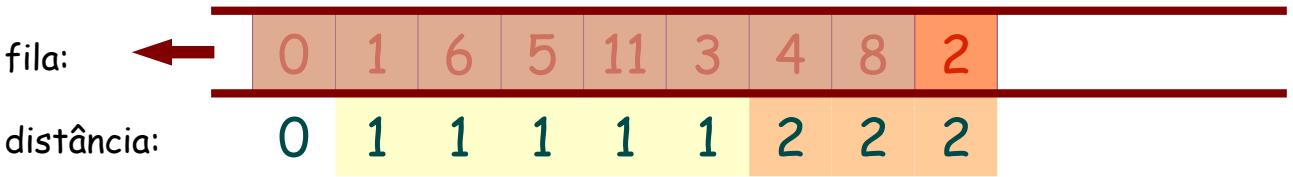




pular

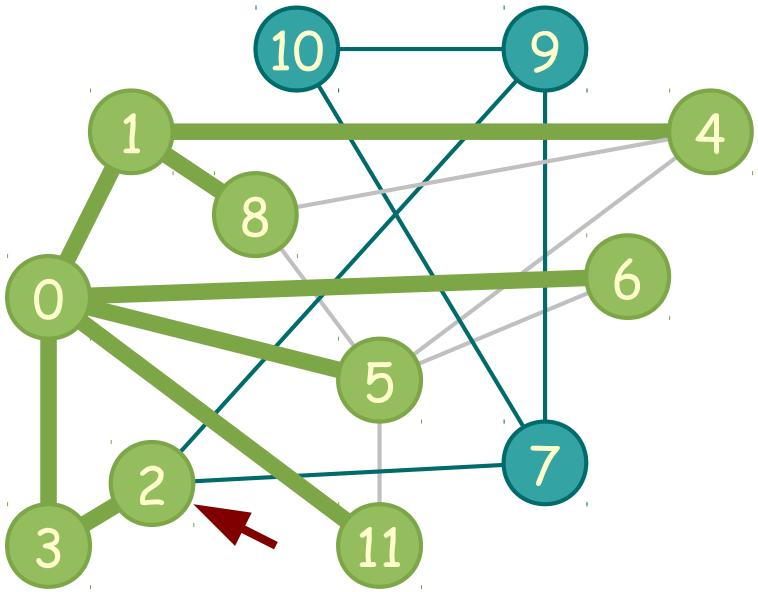
Busca em largura: exemplo

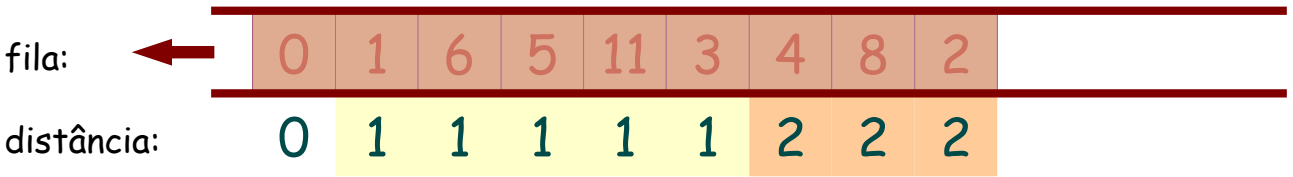




pular

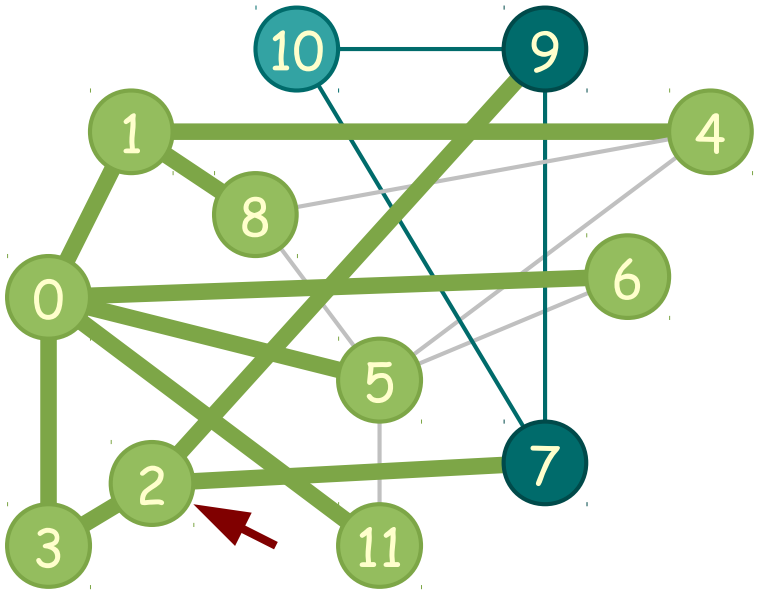
Busca em largura: exemplo

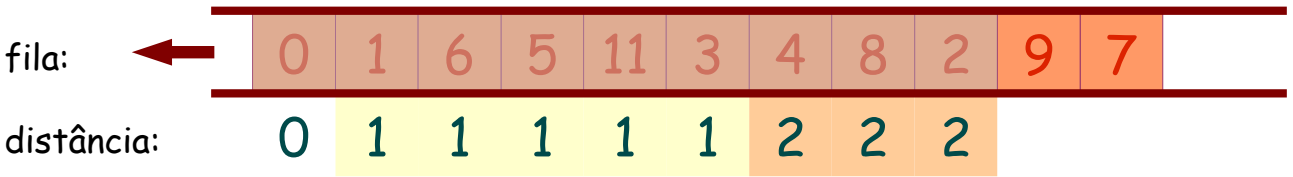




pular

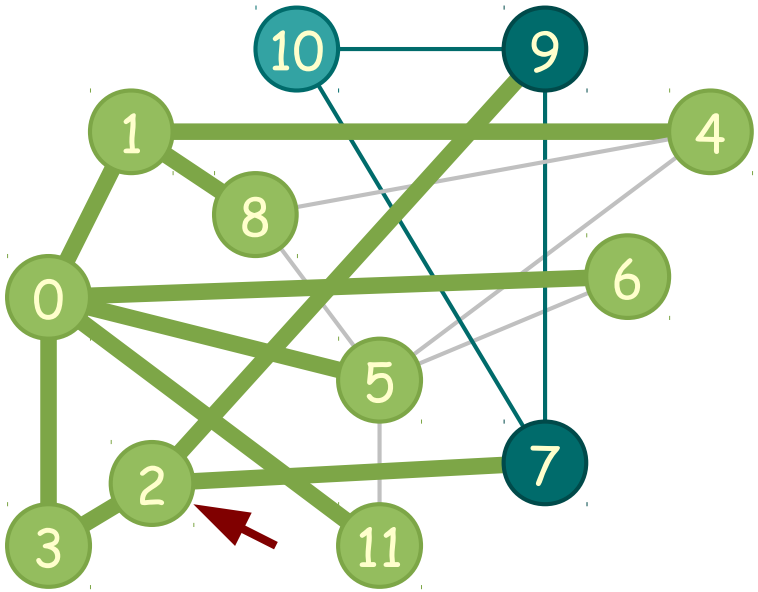
Busca em largura: exemplo

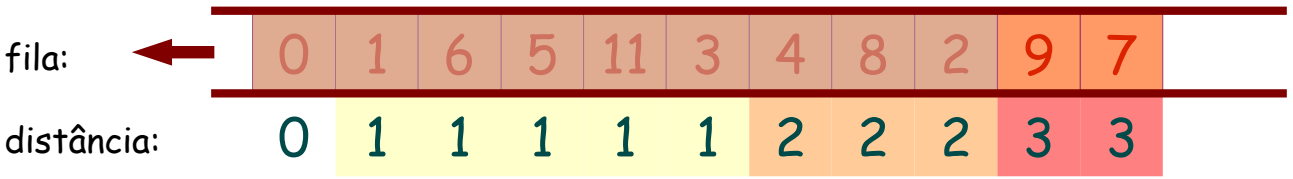




pular

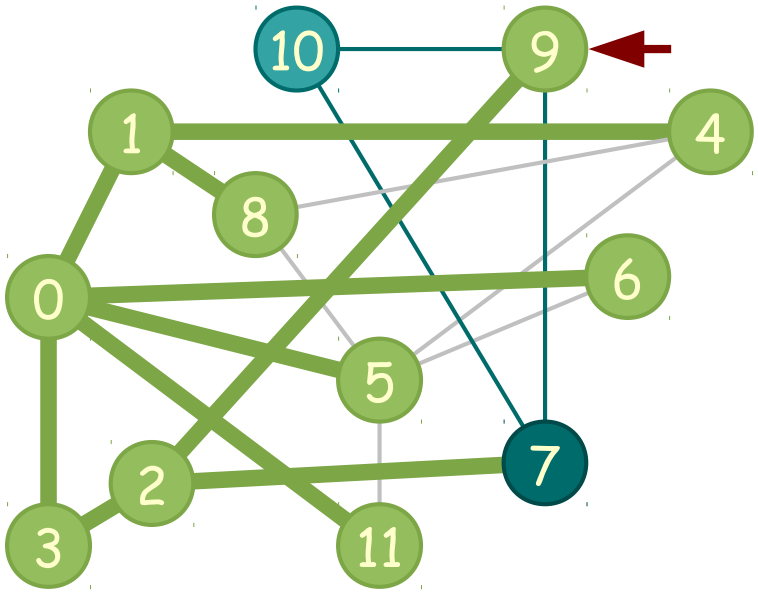
Busca em largura: exemplo





pular

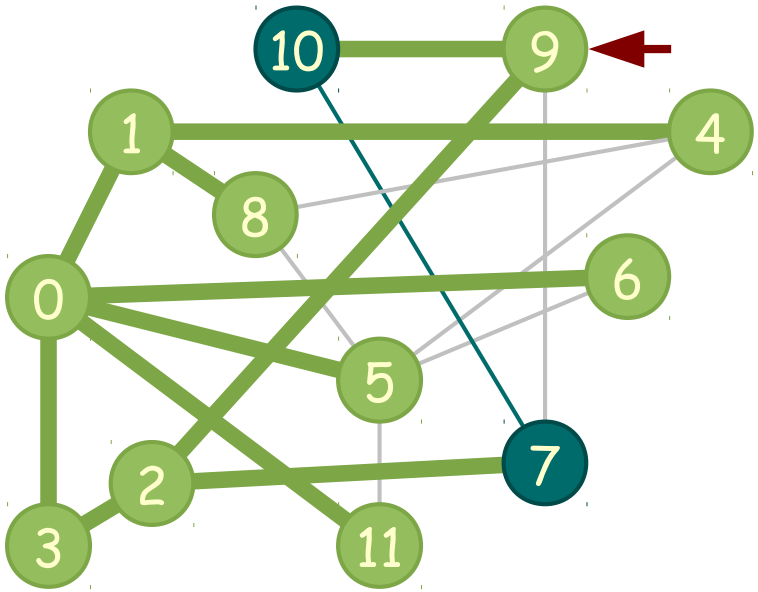
Busca em largura: exemplo

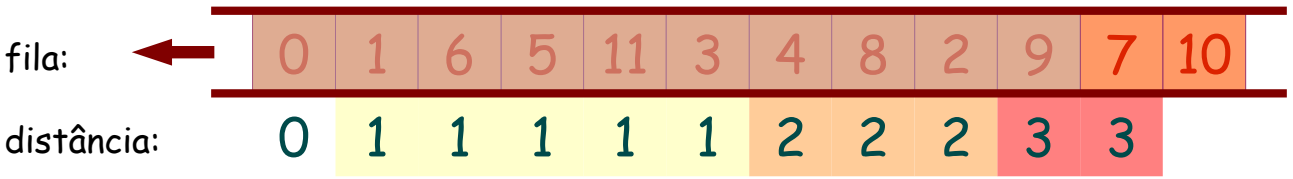


fila:	0	1	6	5	11	3	4	8	2	9	7
distância:	0	1	1	1	1	1	2	2	2	3	3

pular

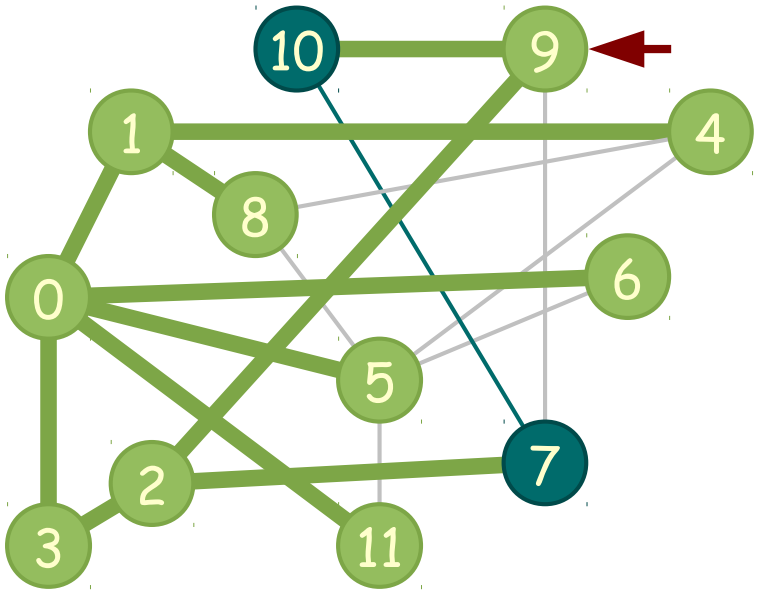
Busca em largura: exemplo

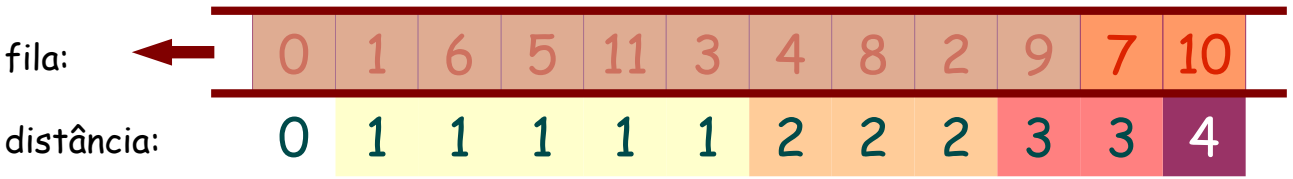




pular

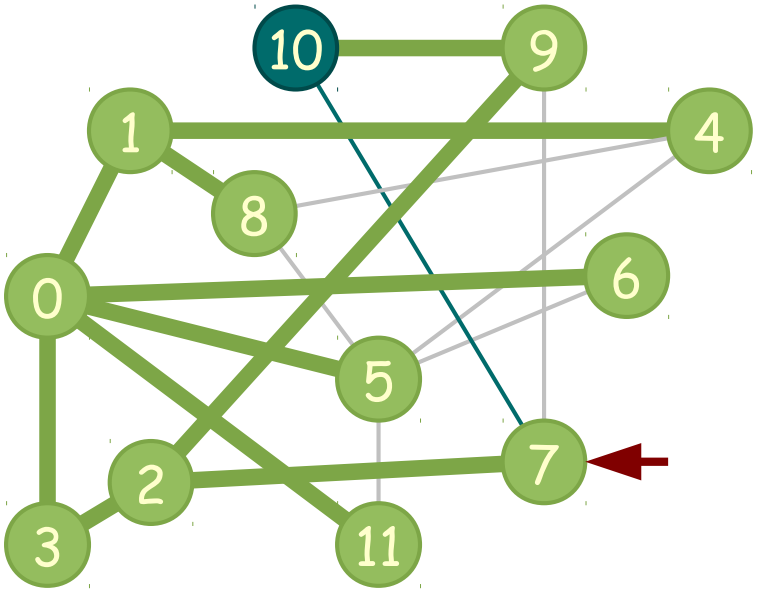
Busca em largura: exemplo

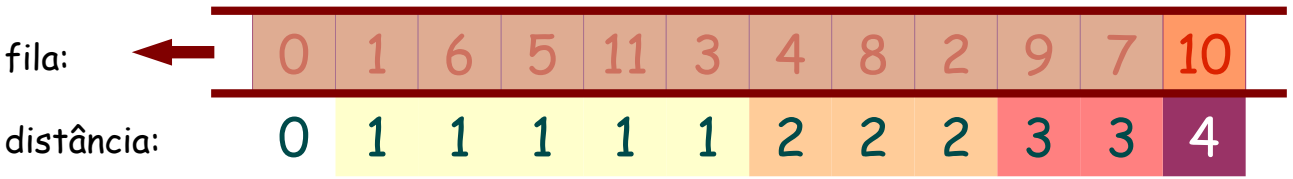




pular

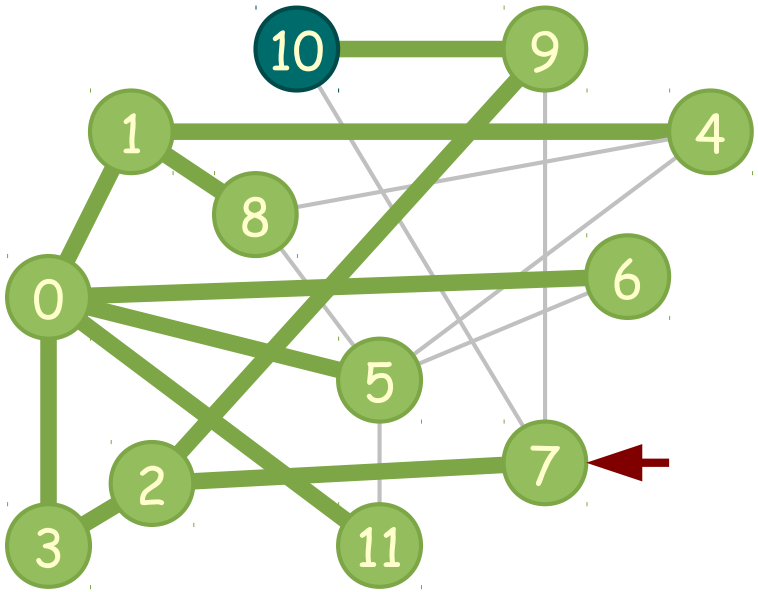
Busca em largura: exemplo





pular

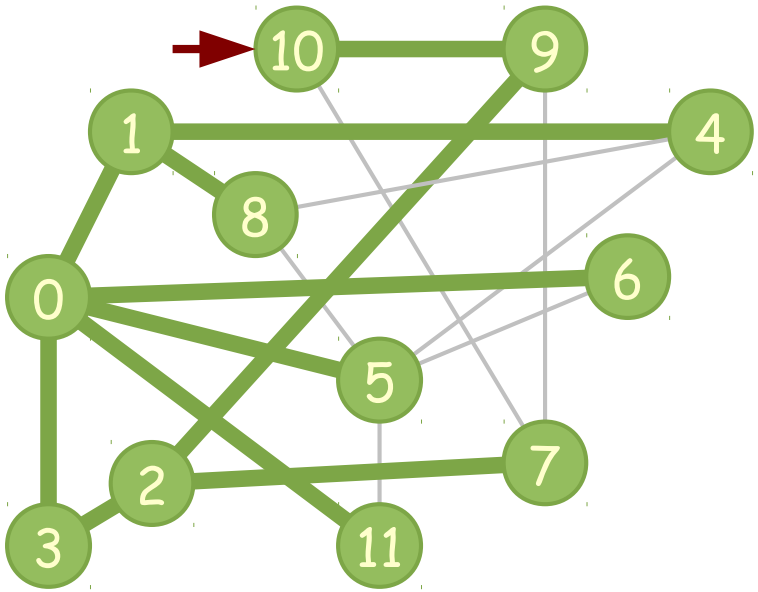
Busca em largura: exemplo



fila:	0	1	6	5	11	3	4	8	2	9	7	10
distância:	0	1	1	1	1	1	2	2	2	3	3	4

pular

Busca em largura: exemplo



fila:	0	1	6	5	11	3	4	8	2	9	7	10
distância:	0	1	1	1	1	1	2	2	2	3	3	4

pular

Impressão de caminhos em uma árvore de busca

Podemos obter o caminho na árvore de busca a partir do vértice inicial s até um vértice qualquer v:

PRINT-PATH(G, s, v)

```
1  if v == s
2        print s
3  elseif v.π == NULL
4        error "unreachable"
5  else
6        PRINT-PATH(G, s, v.π)
7        print v
```

Exercício

1. Escreva um algoritmo que, dado um grafo direcionado $D = (V, E)$, obtenha um novo grafo $D' = (V, E')$ em que a direção das arestas estão invertidas, isso é, se $(u, v) \in E$, então $(v, u) \in E'$ e vice-versa.
   1. usando listas de adjacência
   2. usando matriz de adjacência
2. Encontre uma árvore de busca em profundidade para o grafo do slide 25 a partir do vértice 6. Note que podem existir várias árvores dependendo da ordem em que os vizinhos sejam visitados. Para obter uma única árvore, sempre que houver mais de um vizinho não visitado, escolha o vértice de menor índice.
3. Qual a complexidade da busca em largura?

Exercício 2 - Representação implícita

Uma planta de uma casa é representada por uma matriz de caracteres onde:

- # representa uma parede;
- . representa um espaço vazio.

Qual o número de cômodos na casa? Considere que pode haver portas e corredores na casa e que cada cômodo tem dimensões pelo menos 2×2.
Na figura abaixo há 6 cômodos.

#	#	#	#	#	#	#	#	#	#	#
#	.	.	.	.	#	.	.	.	.	#
#	.	.	.	.	#	.	.	.	.	#
#	.	.	.	.	#	#	#	#	#	#
#	.	.	.	.	.	.	.	.	.	#
#	.	#	#	#	.	#	.	.	.	#
#	.	.	.	#	.	#	.	.	.	#
#	.	.	.	#	.	#	#	#	#	#
#	#	#	.	#	.	.	.	.	.	#
#	.	.	.	#	.	.	.	.	.	#
#	.	.	.	#	.	.	.	.	.	#
#	#	#	#	#	#	#	#	#	#	#

1. Formule o problema como um problema em grafo.
2. Se não houvesse corredores e portas, como você enunciaria o problema de grafo correspondente?
3. (desafio) Implemente um programa que conte o número de cômodos.