Estruturas de Dados

Percurso em Grafos

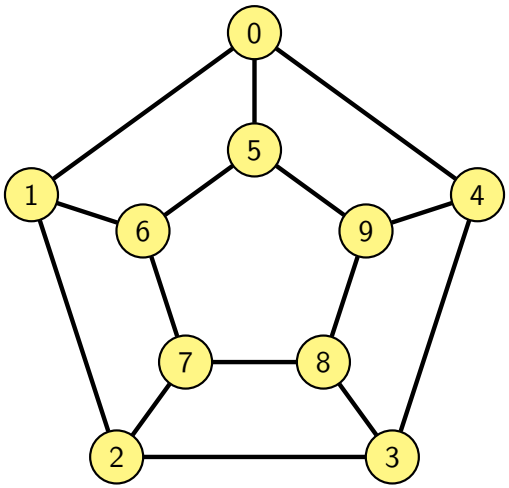
Aula 12

Prof. Felipe A. Louza

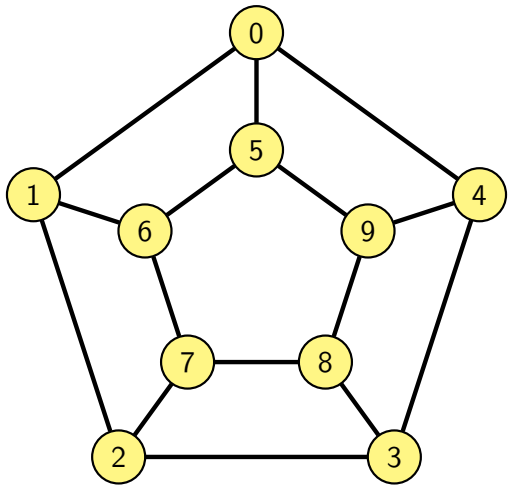
Roteiro

Roteiro

Caminhos em Grafos



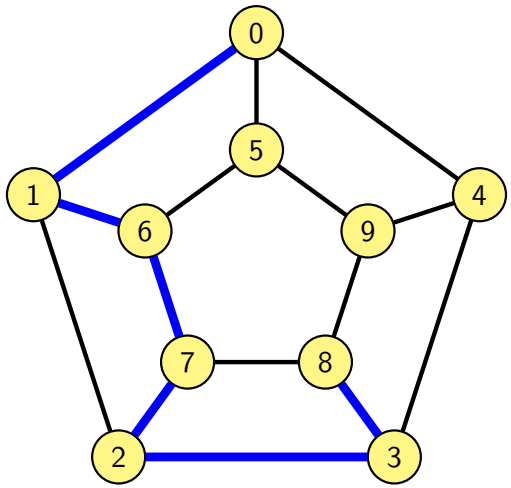
Caminhos em Grafos



Um caminho em um grafo de u para v é:

- Uma sequência sem repetição de vértices vizinhos
- Começando em u e terminado em v

Caminhos em Grafos



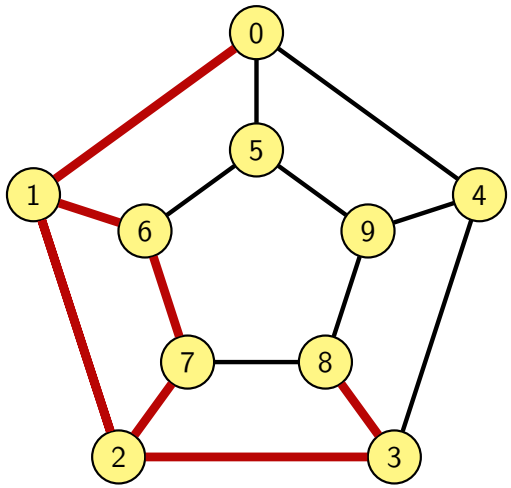
Um caminho em um grafo de u para v é:

- Uma sequência sem repetição de vértices vizinhos
- Começando em u e terminado em v

Por exemplo:

- $0, 1, 6, 7, 2, 3, 8$ é um caminho de 0 para 8

Caminhos em Grafos



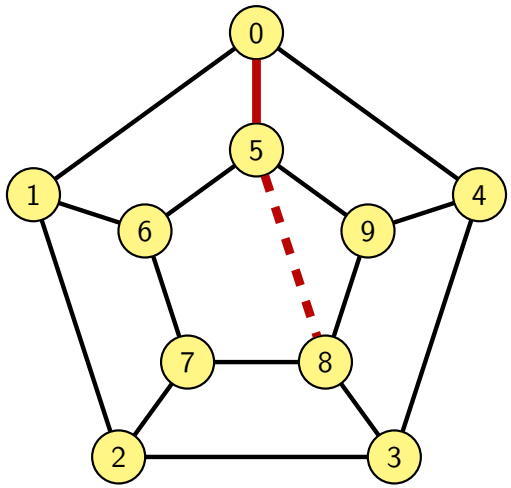
Um caminho em um grafo de u para v é:

- Uma sequência sem repetição de vértices vizinhos
- Começando em u e terminado em v

Por exemplo:

- $0, 1, 6, 7, 2, 3, 8$ é um caminho de 0 para 8
- $0, 1, 2, 7, 6, 1, 2, 3, 8$ **não** é um caminho de 0 para 8

Caminhos em Grafos



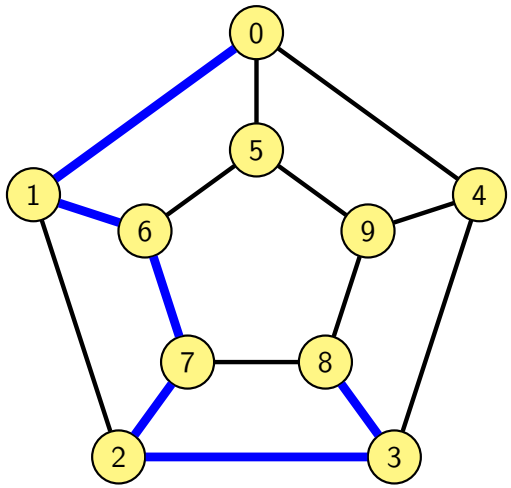
Um caminho em um grafo de u para v é:

- Uma sequência sem repetição de vértices vizinhos
- Começando em u e terminado em v

Por exemplo:

- $0, 1, 6, 7, 2, 3, 8$ é um caminho de 0 para 8
- $0, 1, 2, 7, 6, 1, 2, 3, 8$ **não** é um caminho de 0 para 8
- $0, 5, 8$ **não** é um caminho de 0 para 8

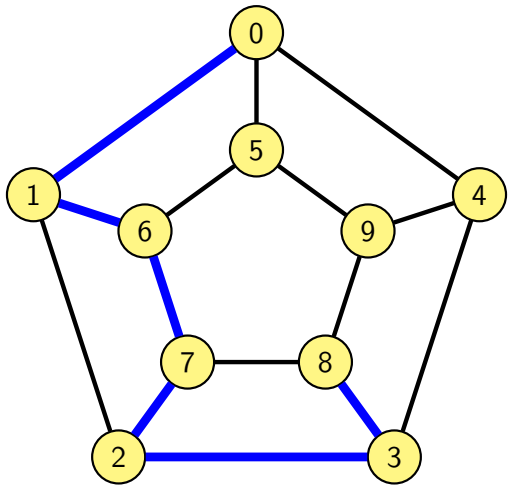
Caminhos em Grafos



Um caminho em um grafo de u para v é:

- Uma sequência de vértice $v_0, v_1, \ldots, v_k$ onde
- $v_0 = u$ e $v_k = v$
- $\{v_i, v_{i+1}\}$ é uma aresta para todo $0 \leq i \leq k-1$
- $v_i \neq v_j$ para todo $0 \leq i < j \leq k$ ← sem repetição

Caminhos em Grafos



Um caminho em um grafo de u para v é:

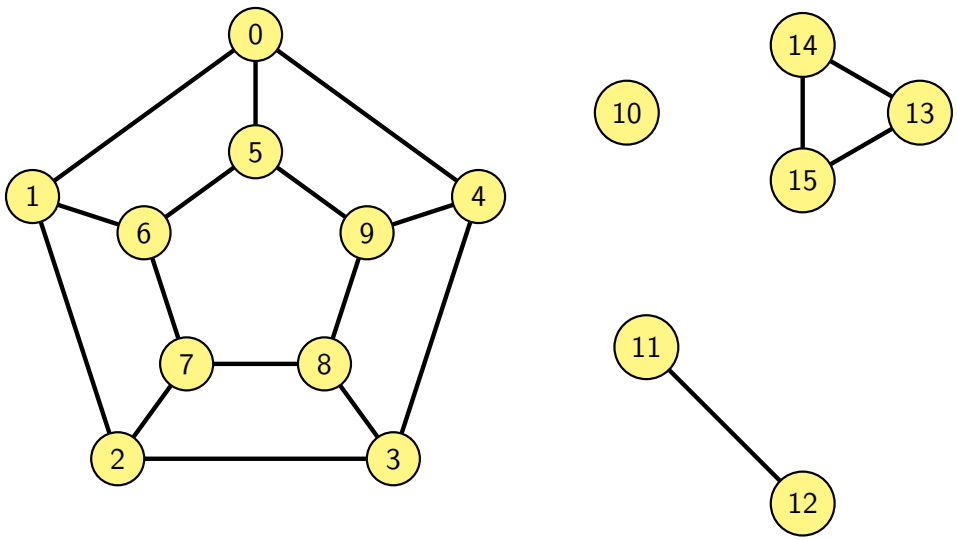
- Uma sequência de vértice $v_0, v_1, \ldots, v_k$ onde
- $v_0 = u$ e $v_k = v$
- $\{v_i, v_{i+1}\}$ é uma aresta para todo $0 \leq i \leq k-1$
- $v_i \neq v_j$ para todo $0 \leq i < j \leq k$ ← sem repetição

k é o comprimento do caminho

- $k = 0$ se e somente se $u = v$

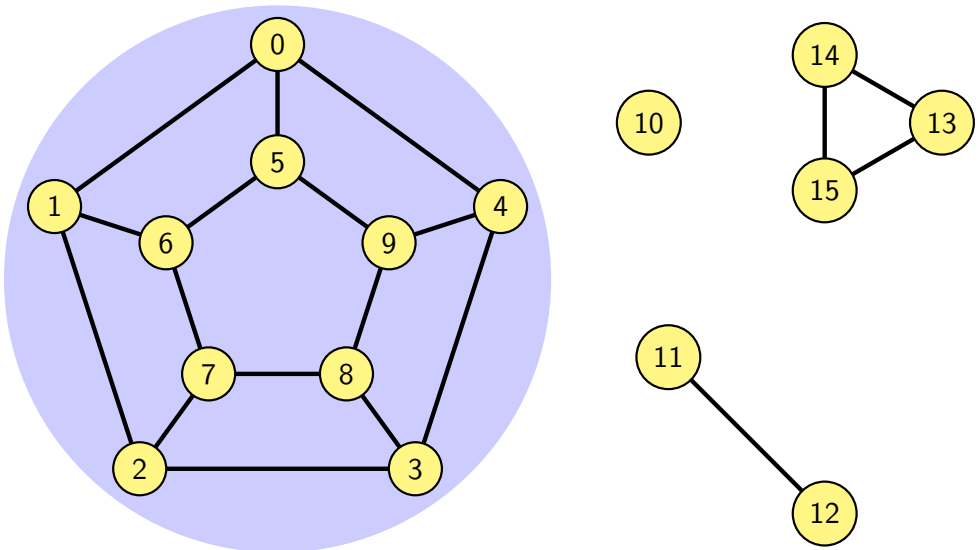
Componentes Conexas

Um grafo pode ter várias "partes"



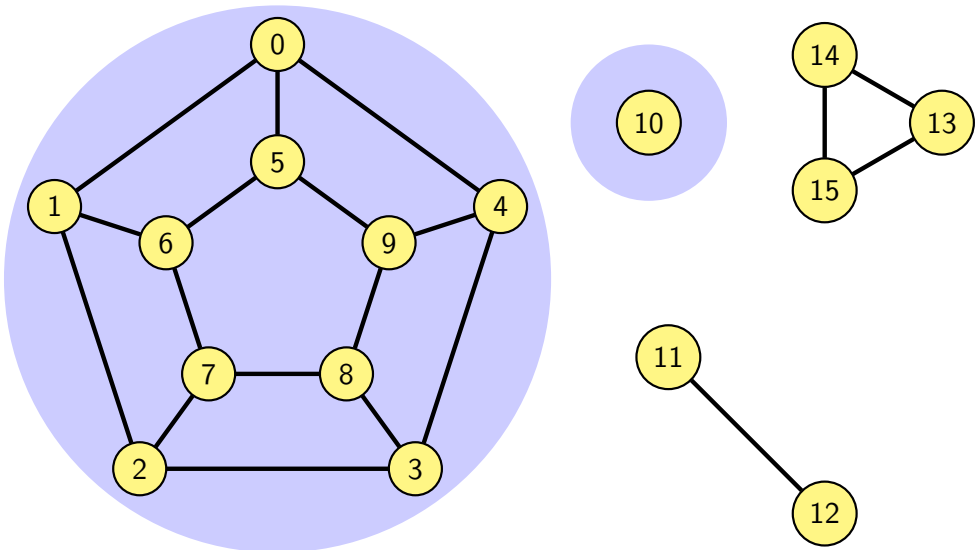
Componentes Conexas

Um grafo pode ter várias "partes"



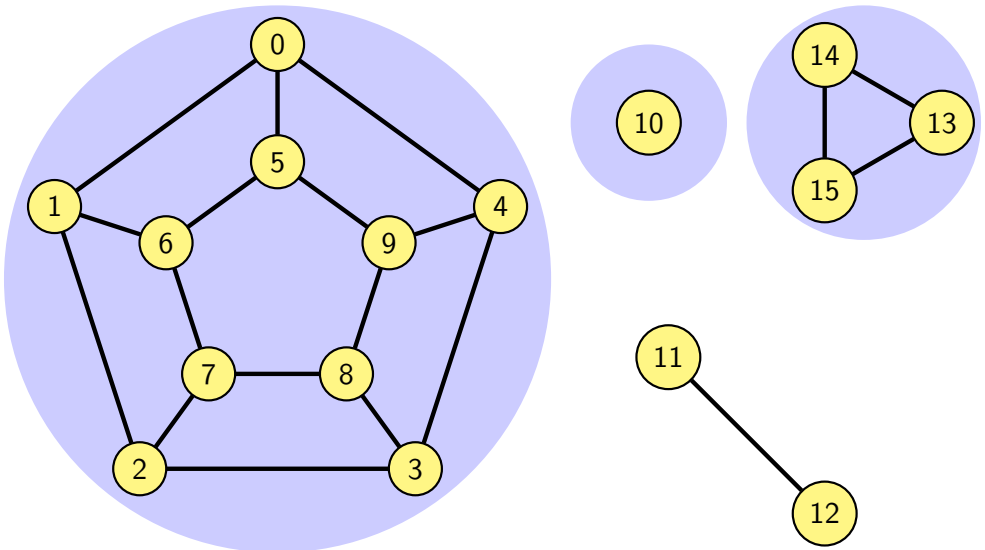
Componentes Conexas

Um grafo pode ter várias "partes"



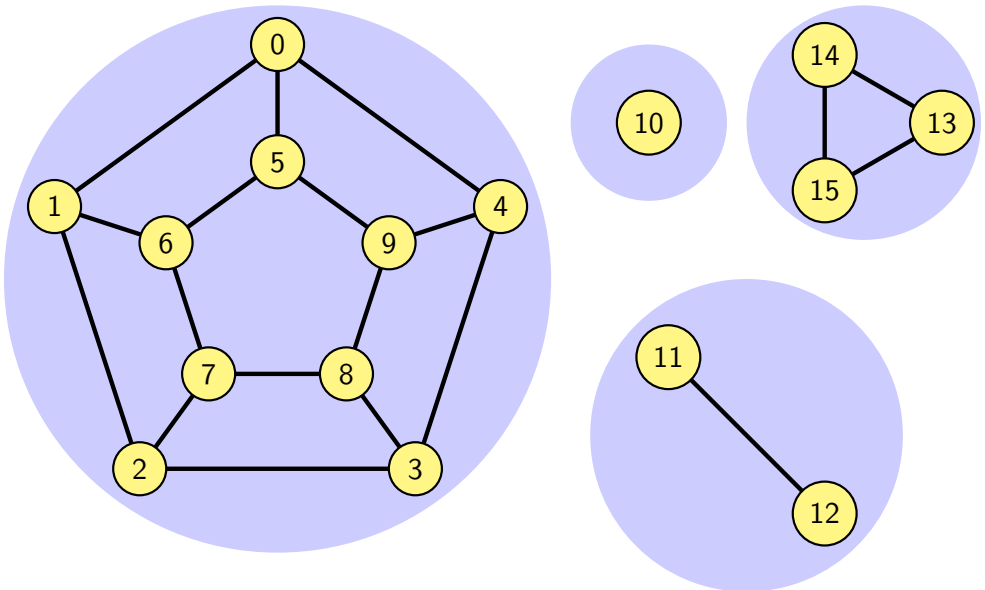
Componentes Conexas

Um grafo pode ter várias "partes"



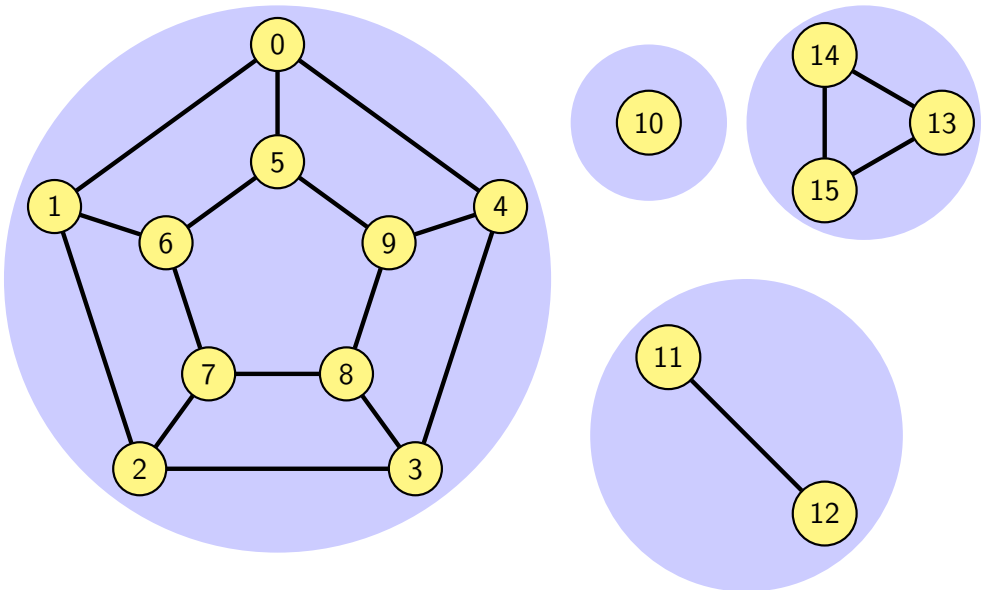
Componentes Conexas

Um grafo pode ter várias "partes"



Componentes Conexas

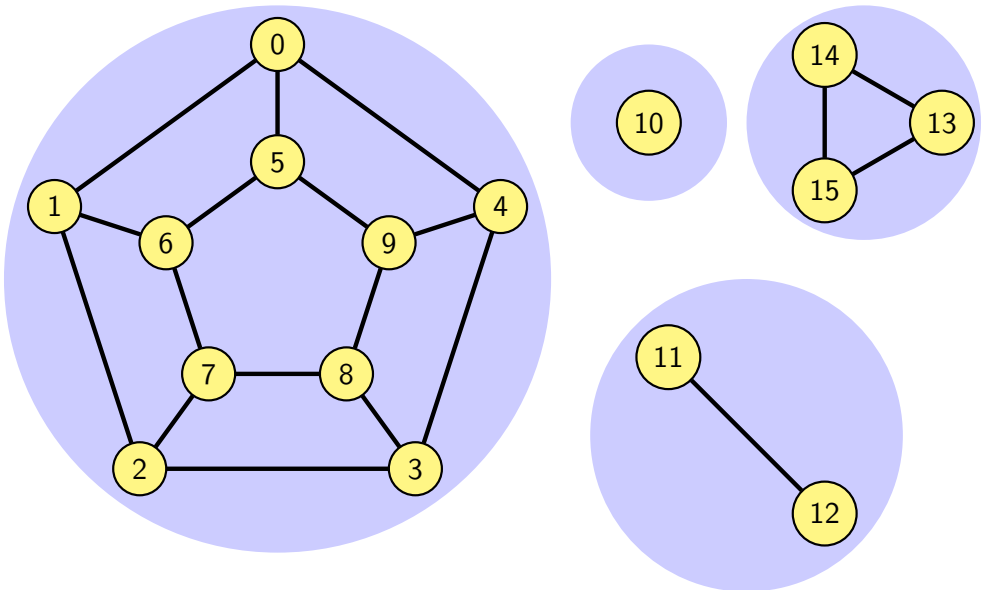
Um grafo pode ter várias "partes"



Chamamos essas partes de Componentes Conexas

Componentes Conexas

Um grafo pode ter várias "partes"

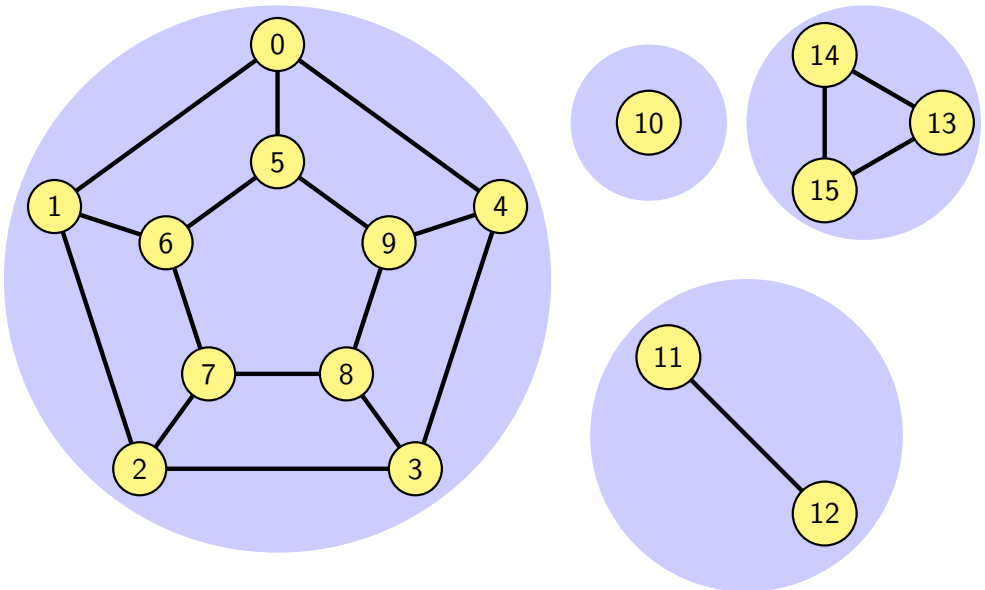


Chamamos essas partes de Componentes Conexas

- Um par de vértices está na **mesma componente** se e somente se existe caminho entre eles

Componentes Conexas

Um grafo pode ter várias "partes"

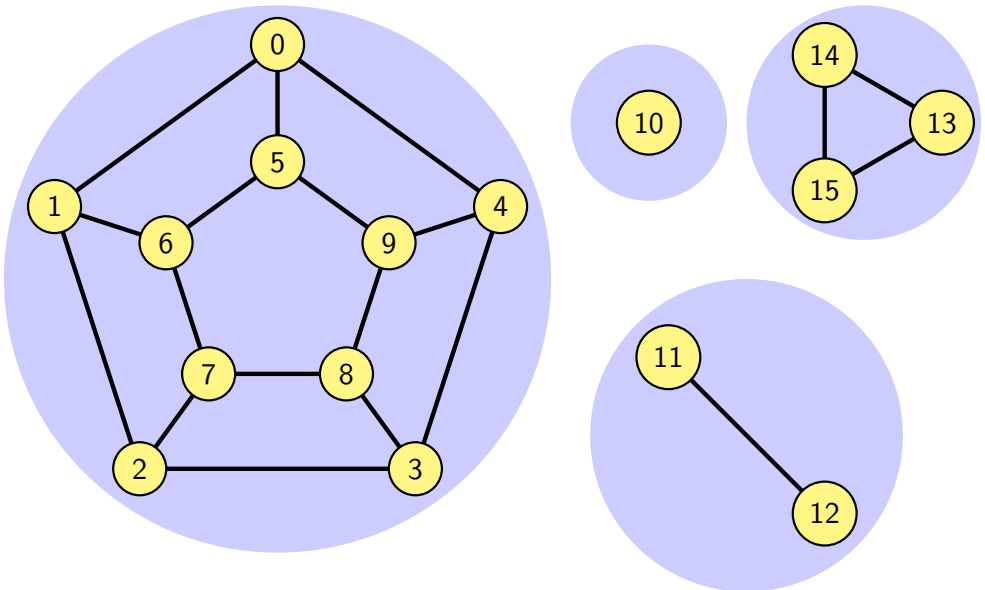


Chamamos essas partes de Componentes Conexas

- Um par de vértices está na **mesma componente** se e somente se existe caminho entre eles
  - Não há caminho entre vértices de componentes distintas

Componentes Conexas

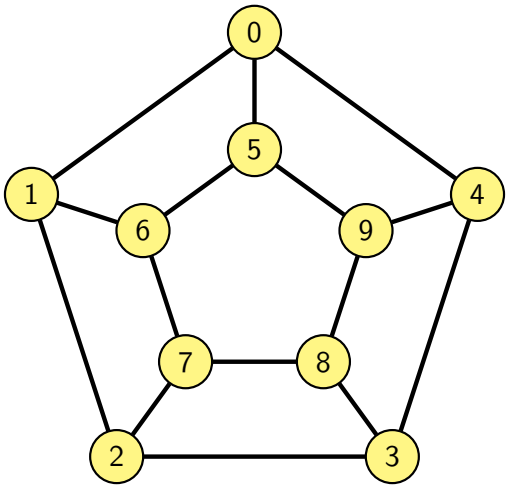
Um grafo pode ter várias "partes"



Chamamos essas partes de Componentes Conexas

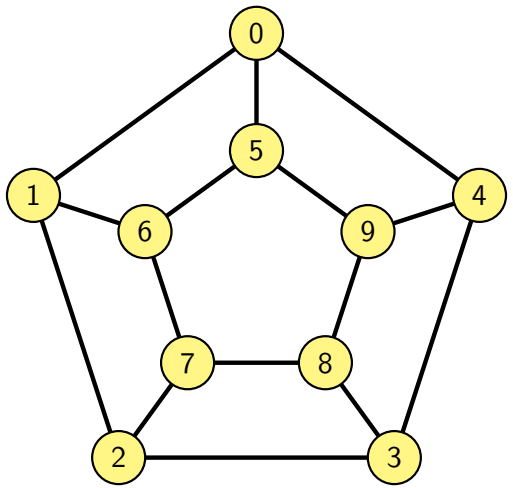
- Um par de vértices está na **mesma componente** se e somente se existe caminho entre eles
  - Não há caminho entre vértices de componentes distintas
- Um grafo conexo tem apenas **uma** componente conexa

Ciclos em Grafos



Um ciclo em um grafo é:

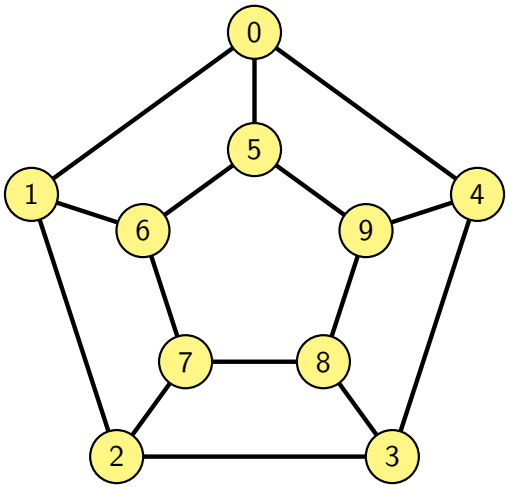
Ciclos em Grafos



Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Ciclos em Grafos

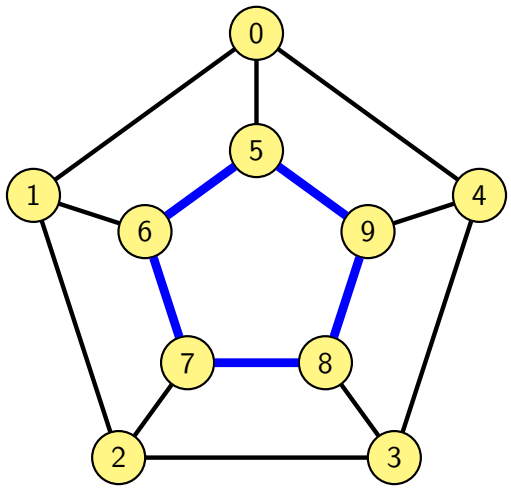


Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

Ciclos em Grafos



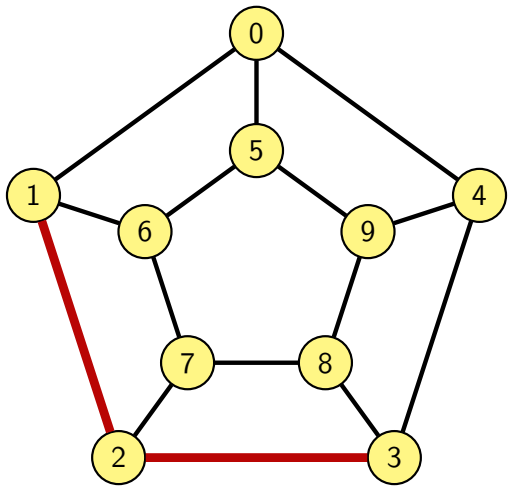
Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

- $5, 6, 7, 8, 9, 5$ é um ciclo

Ciclos em Grafos



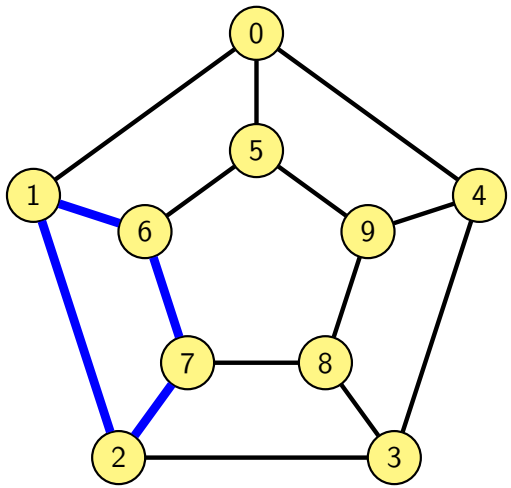
Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

- $5, 6, 7, 8, 9, 5$ é um ciclo
- $1, 2, 3$ **não** é um ciclo

Ciclos em Grafos



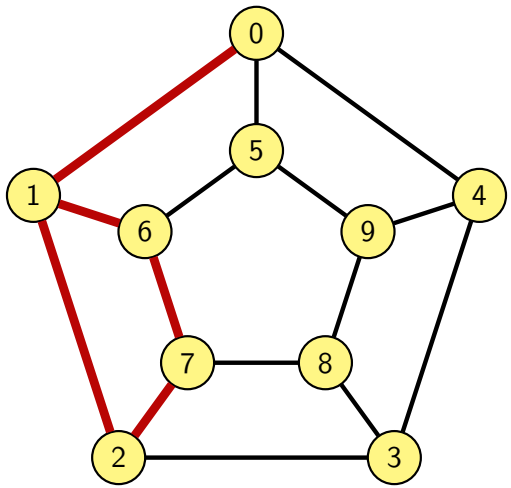
Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

- $5, 6, 7, 8, 9, 5$ é um ciclo
- $1, 2, 3$ **não** é um ciclo
- $1, 2, 7, 6, 1$ é um ciclo

Ciclos em Grafos



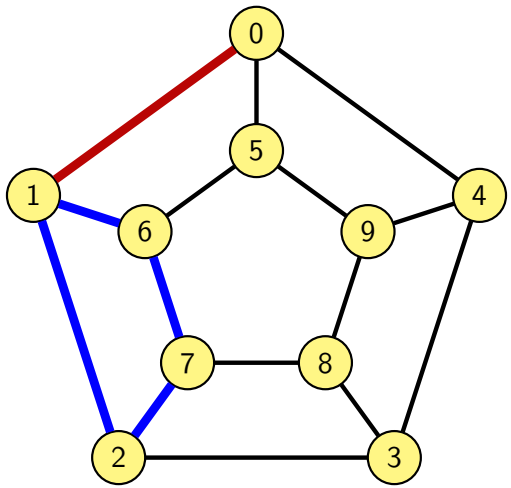
Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

- $5, 6, 7, 8, 9, 5$ é um ciclo
- $1, 2, 3$ **não** é um ciclo
- $1, 2, 7, 6, 1$ é um ciclo
- $1, 2, 7, 6, 1, 0$ **não** é um ciclo

Ciclos em Grafos



Um ciclo em um grafo é:

- Uma sequência de **vértices vizinhos** sem repetição exceto pelo **primeiro** e o **último** vértice (que são os mesmos)

Por exemplo:

- $5, 6, 7, 8, 9, 5$ é um ciclo
- $1, 2, 3$ **não** é um ciclo
- $1, 2, 7, 6, 1$ é um ciclo
- $1, 2, 7, 6, 1, 0$ **não** é um ciclo (mas contém um ciclo)

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

- Em uma árvore "enraizada" definimos uma raiz

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Um subgrafo é um **grafo obtido** a partir da remoção de vértices e arestas de um dado grafo

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

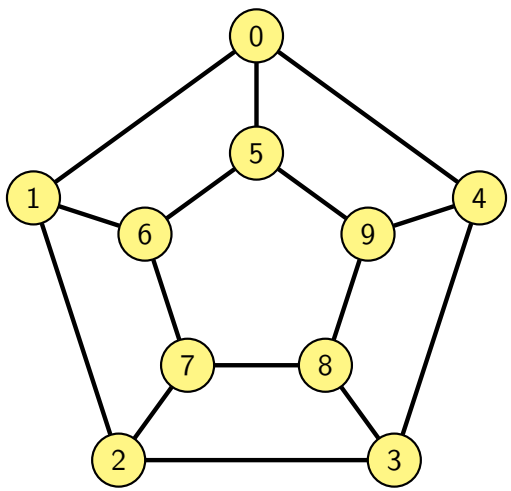
- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Um subgrafo é um **grafo obtido** a partir da remoção de vértices e arestas de um dado grafo

- Podemos considerar também árvores/florestas como subgrafos



Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

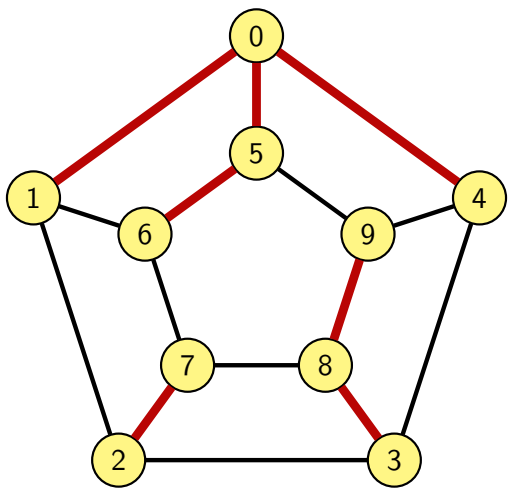
- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Um subgrafo é um **grafo obtido** a partir da remoção de vértices e arestas de um dado grafo

- Podemos considerar também árvores/florestas como subgrafos



Um subgrafo que é uma floresta

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

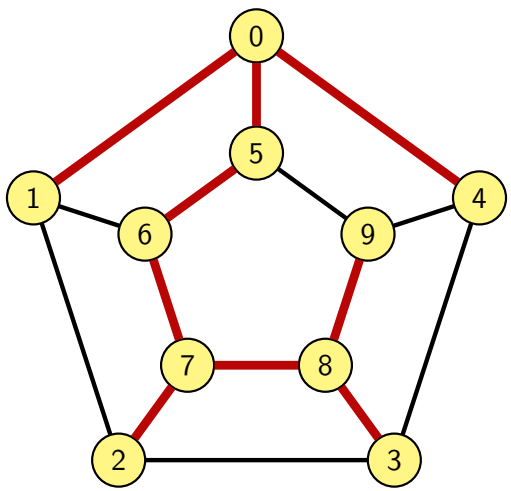
- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Um subgrafo é um **grafo obtido** a partir da remoção de vértices e arestas de um dado grafo

- Podemos considerar também árvores/florestas como subgrafos



Um subgrafo que é uma árvore

Árvores, Florestas e Subgrafos

Uma árvore é um grafo conexo acíclico

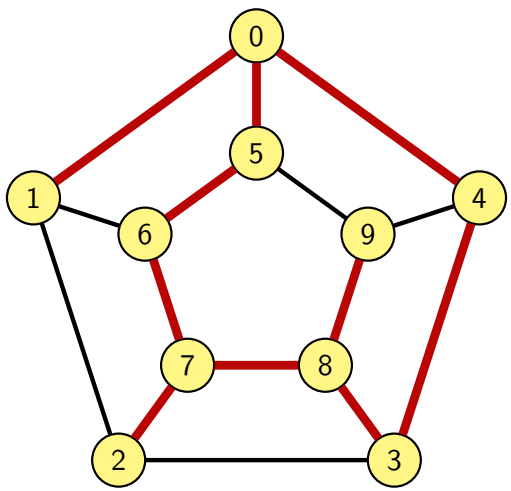
- Em uma árvore "enraizada" definimos uma raiz

Uma floresta (conjunto de árvores) é um grafo acíclico

- Suas componentes conexas são árvores

Um subgrafo é um **grafo obtido** a partir da remoção de vértices e arestas de um dado grafo

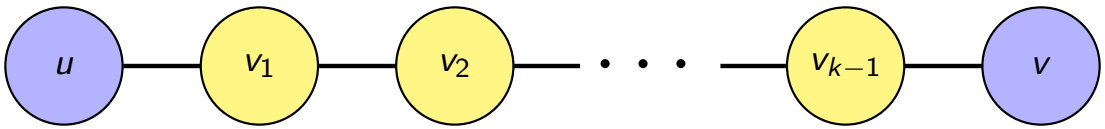
- Podemos considerar também árvores/florestas como subgrafos



Um subgrafo com ciclo

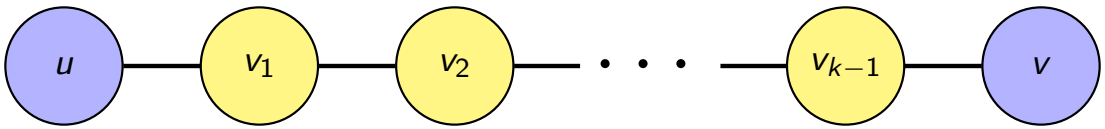
Nosso primeiro problema

Queremos saber se existe um caminho entre u e v



Nosso primeiro problema

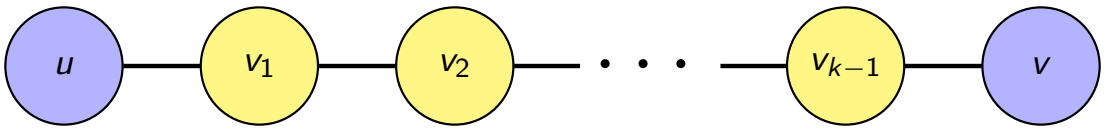
Queremos saber se existe um caminho entre u e v



Se existe caminho e $u \neq v$, existe um segundo vértice v_1

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v

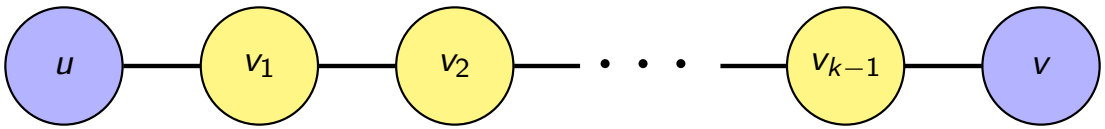


Se existe caminho e $u \neq v$, existe um segundo vértice v_1

- E v_1 é vizinho de u

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v

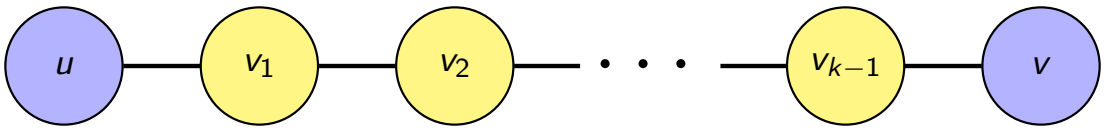


Se existe caminho e $u \neq v$, existe um segundo vértice v_1

- E v_1 é vizinho de u
- Então, ou $v_1 = v$, ou existe um terceiro vértice v_2

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v

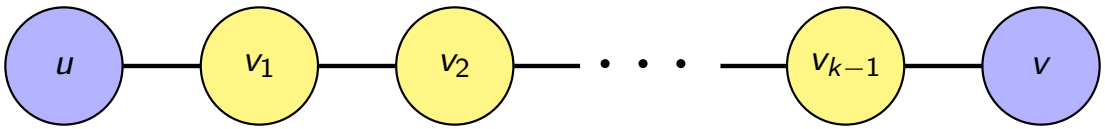


Se existe caminho e $u \neq v$, existe um segundo vértice v_1

- E v_1 é vizinho de u
- Então, ou $v_1 = v$, ou existe um terceiro vértice v_2
  - E v_2 é vizinho de v_1

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v

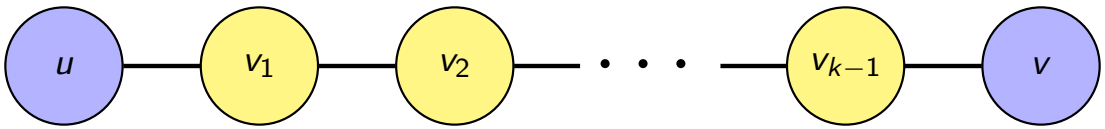


Se existe caminho e $u \neq v$, existe um segundo vértice v_1

- E v_1 é vizinho de u
- Então, ou $v_1 = v$, ou existe um terceiro vértice v_2
  - E v_2 é vizinho de v_1
- E assim por diante...

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v



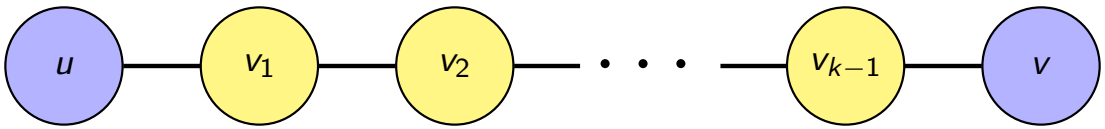
Se existe caminho e $u \neq v$, existe um segundo vértice v_1

- E v_1 é vizinho de u
- Então, ou $v_1 = v$, ou existe um terceiro vértice v_2
  - E v_2 é vizinho de v_1
- E assim por diante...

A dificuldade é acertar **qual vizinho** v_1 de u devemos usar...

Nosso primeiro problema

Queremos saber se existe um caminho entre u e v



Se existe caminho e $u \neq v$, existe um segundo vértice v_1

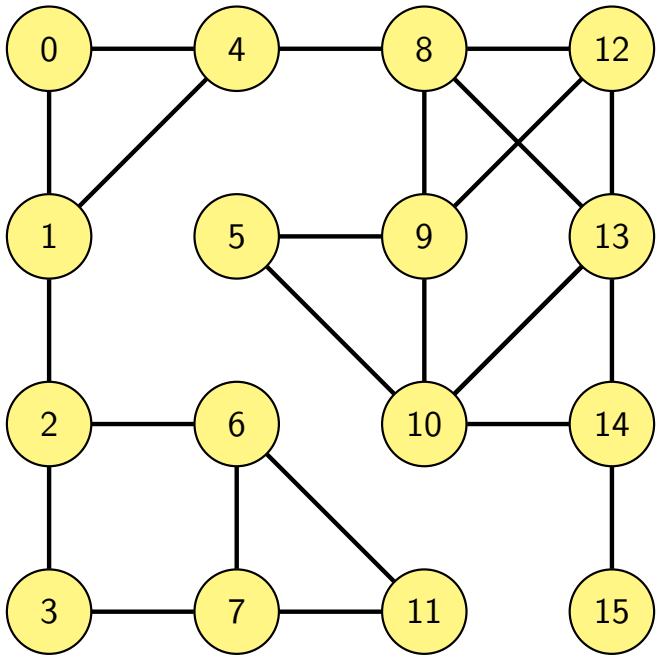
- E v_1 é vizinho de u
- Então, ou $v_1 = v$, ou existe um terceiro vértice v_2
  - E v_2 é vizinho de v_1
- E assim por diante...

A dificuldade é acertar **qual vizinho** v_1 de u devemos usar...

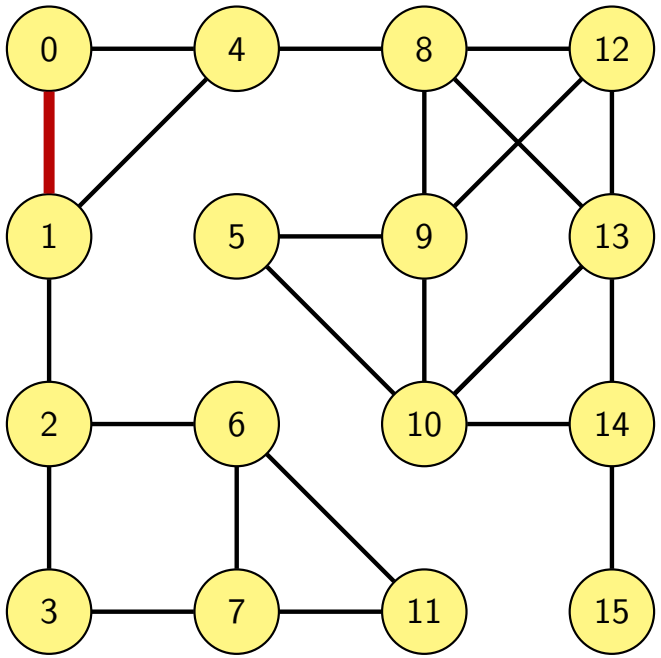
- Solução: testar todos!

Roteiro

Exemplo - Existe caminho de 0 até 15?

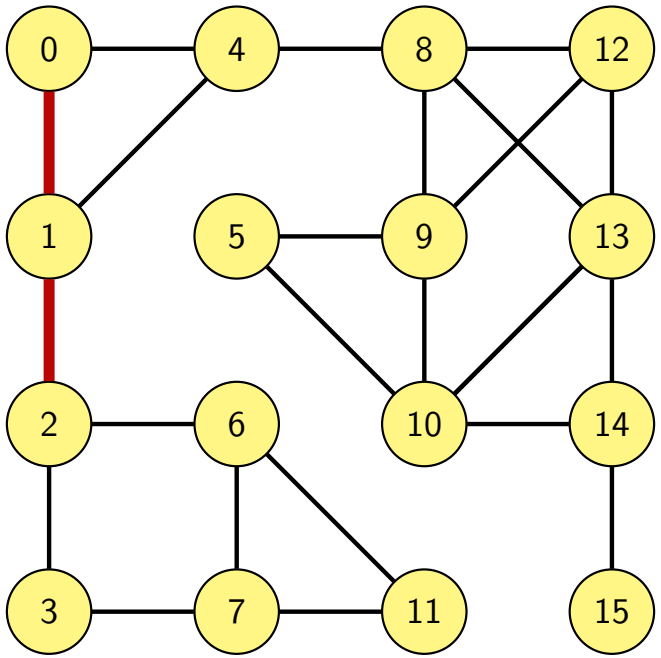


Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

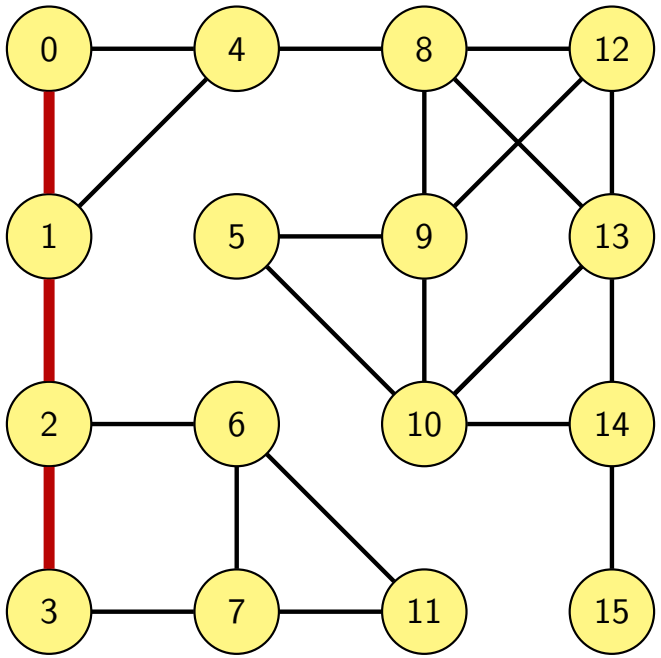
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção

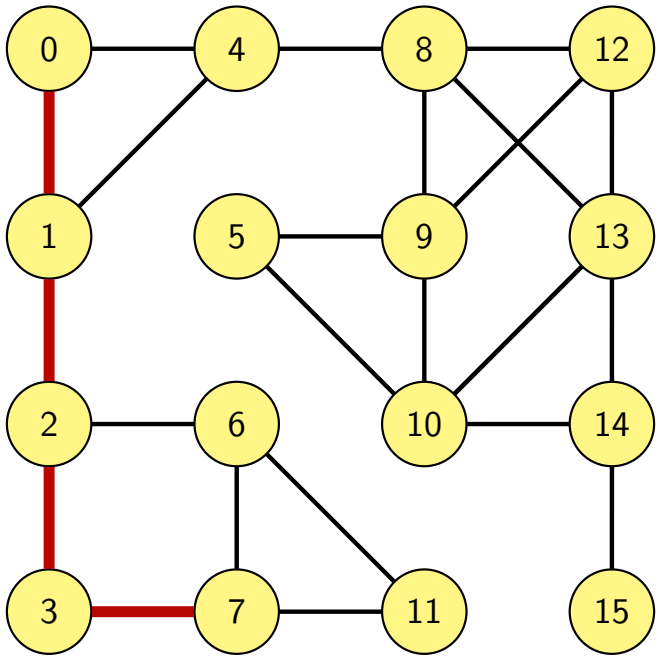
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção

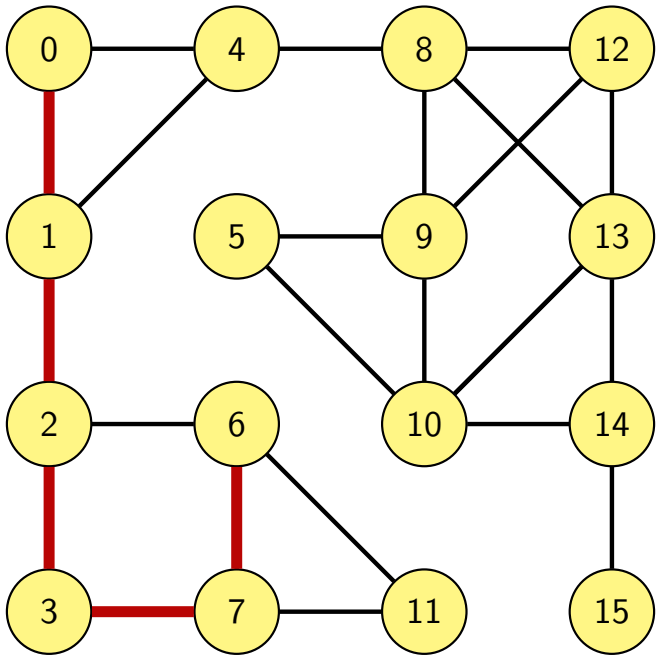
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção

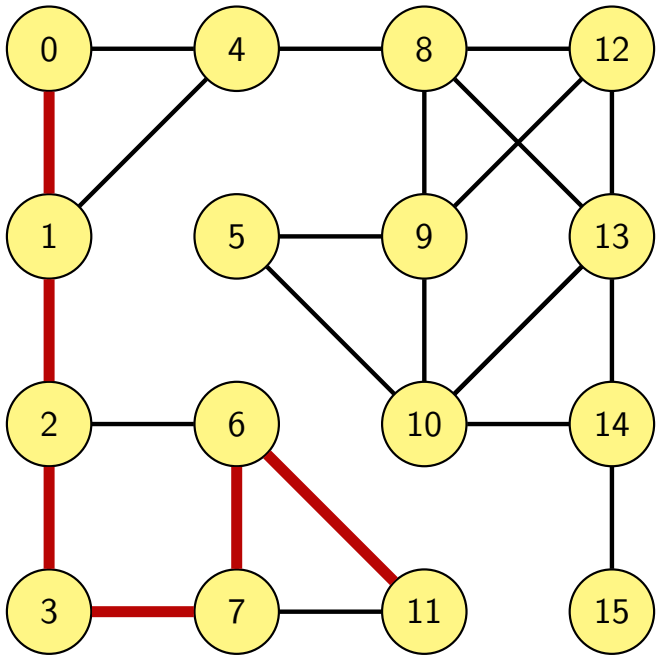
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção

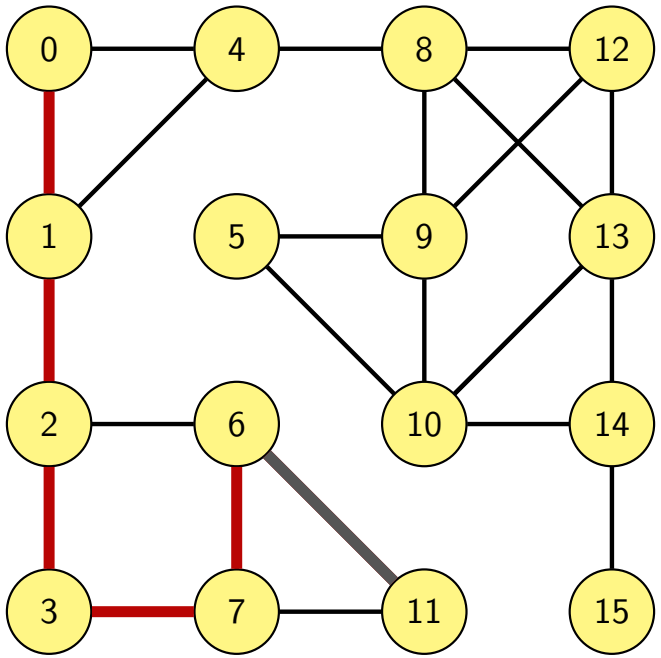
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

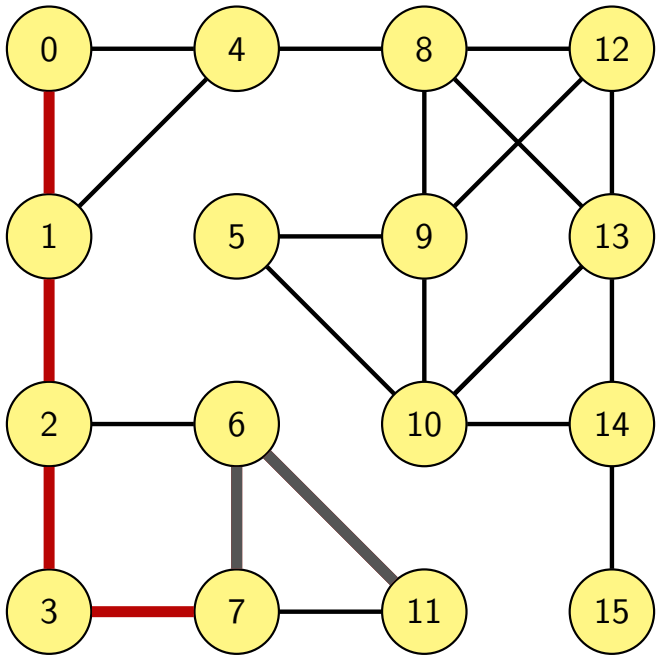
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

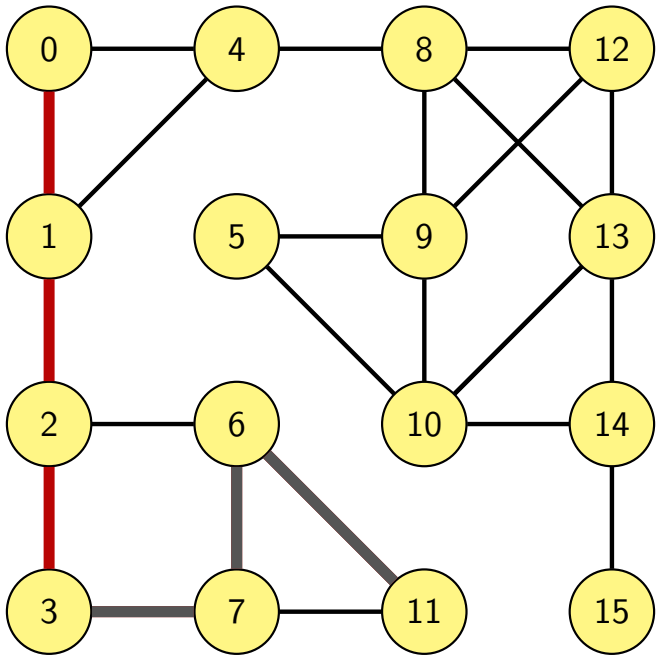
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

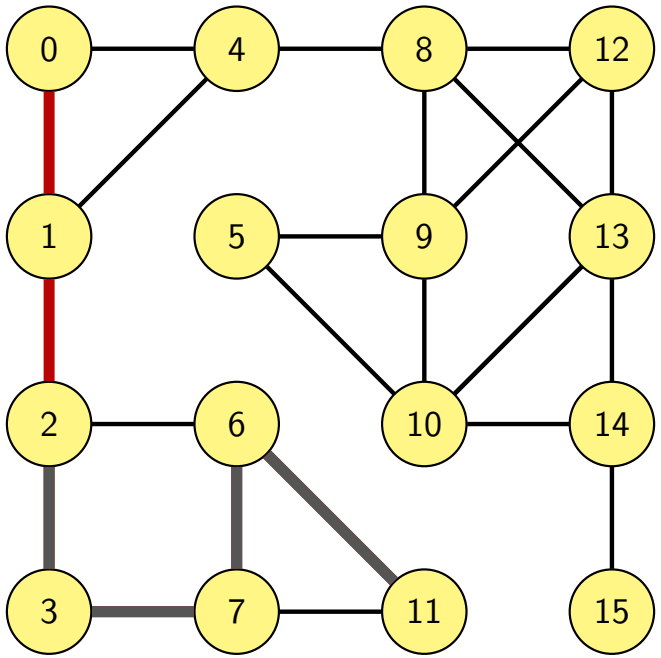
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

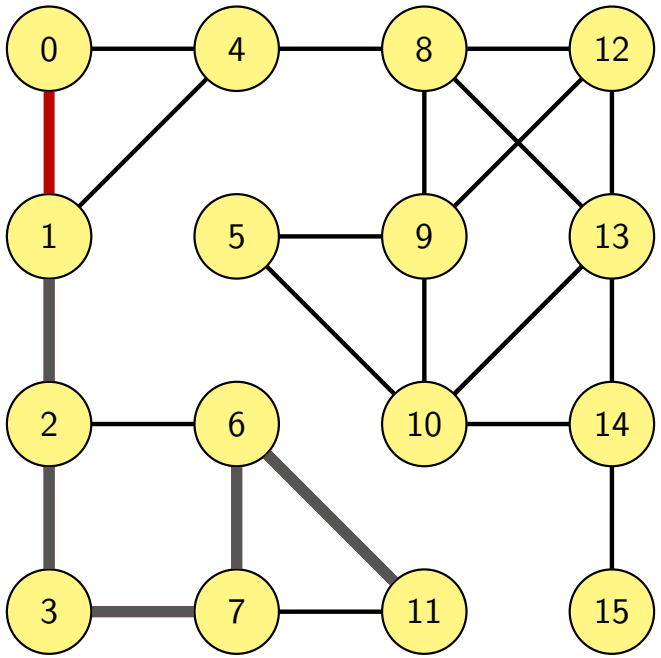
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

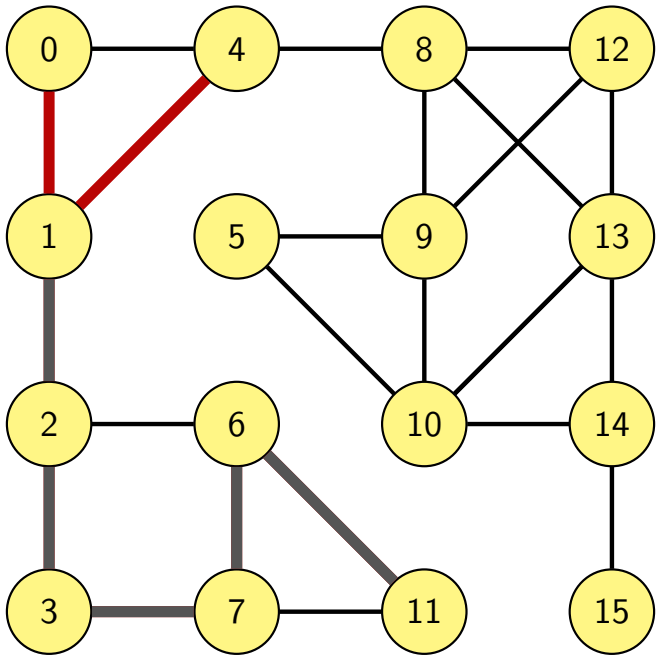
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

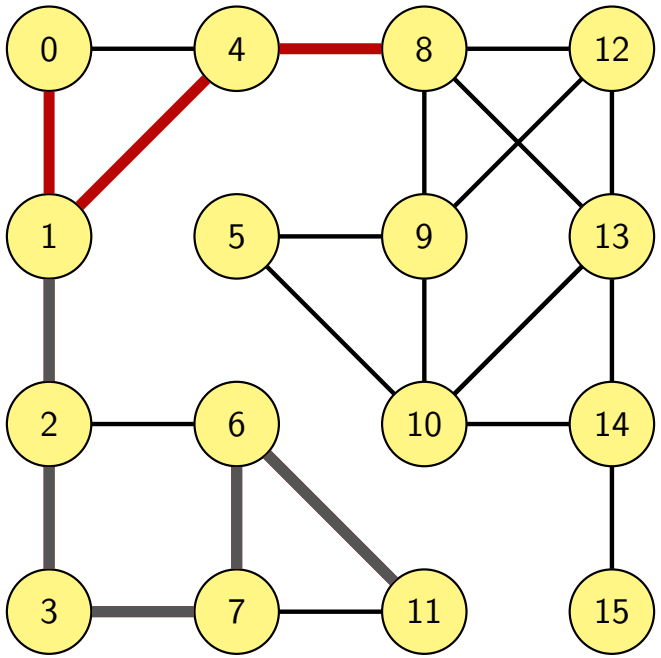
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

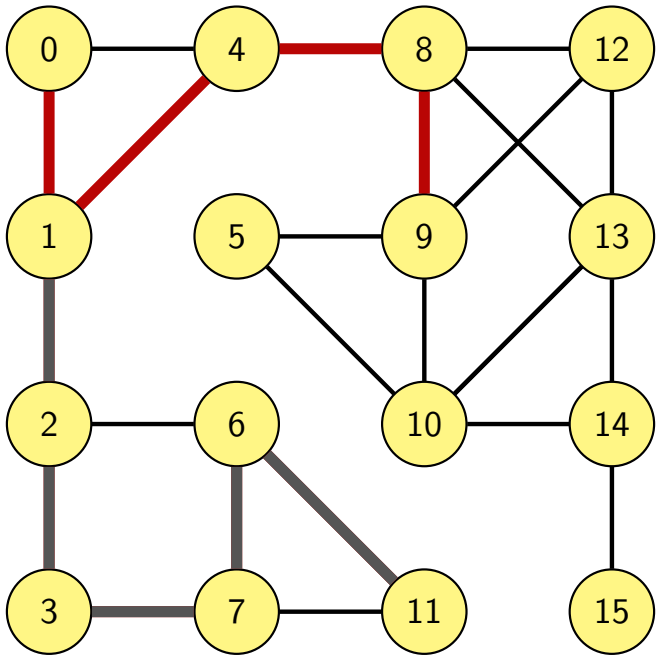
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

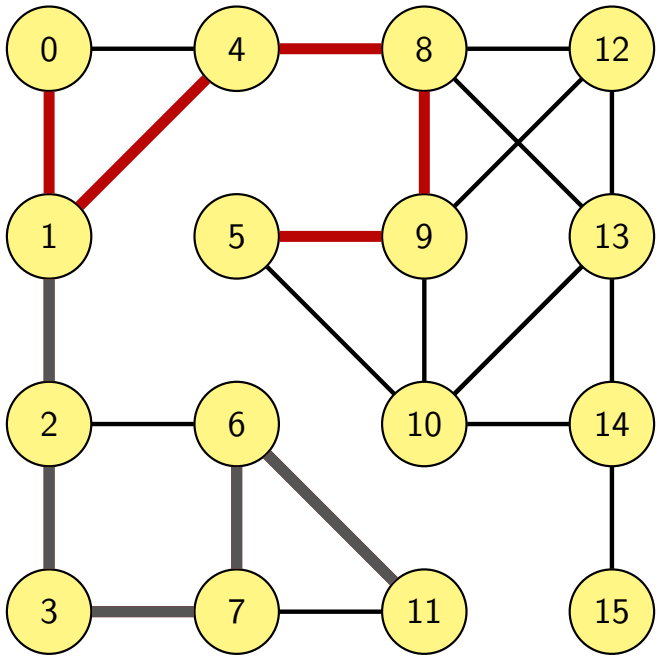
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

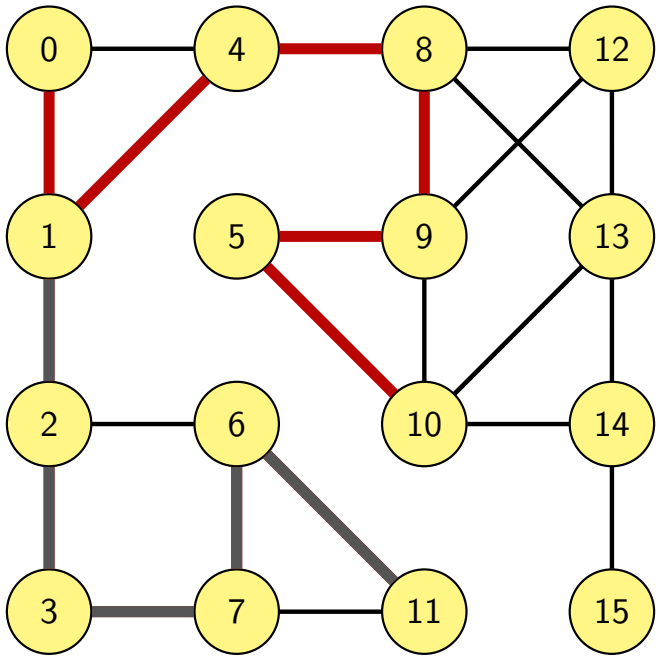
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

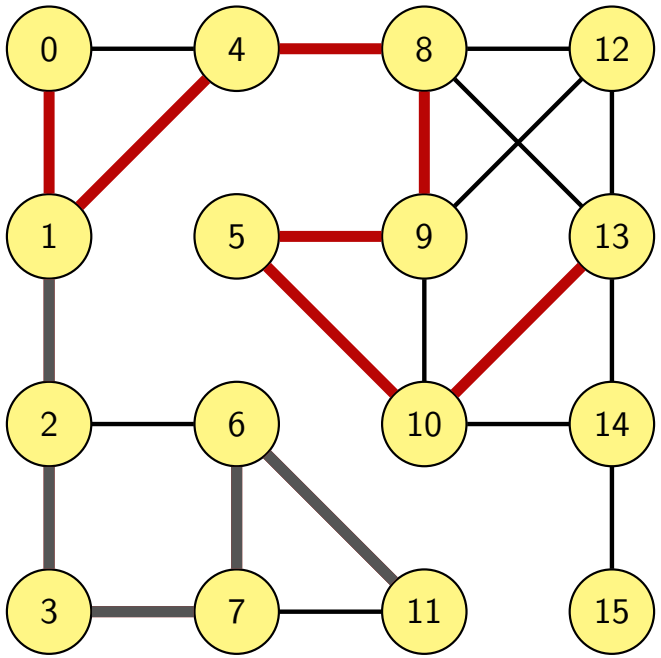
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

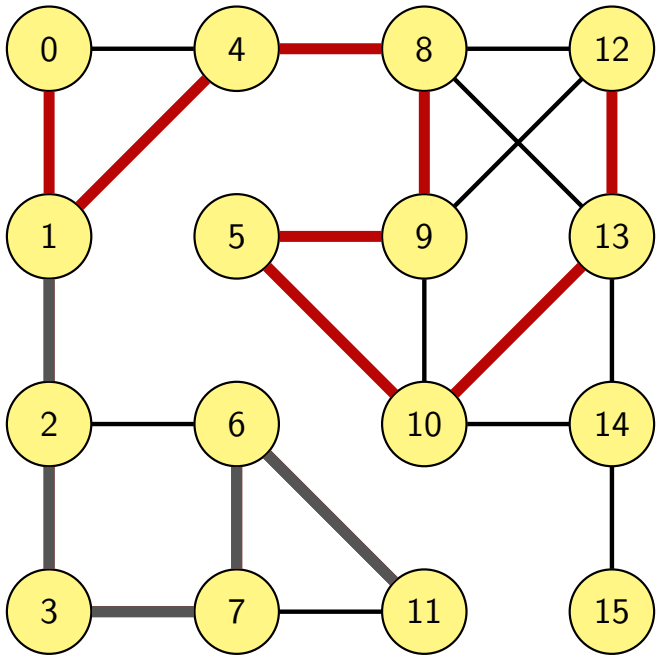
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

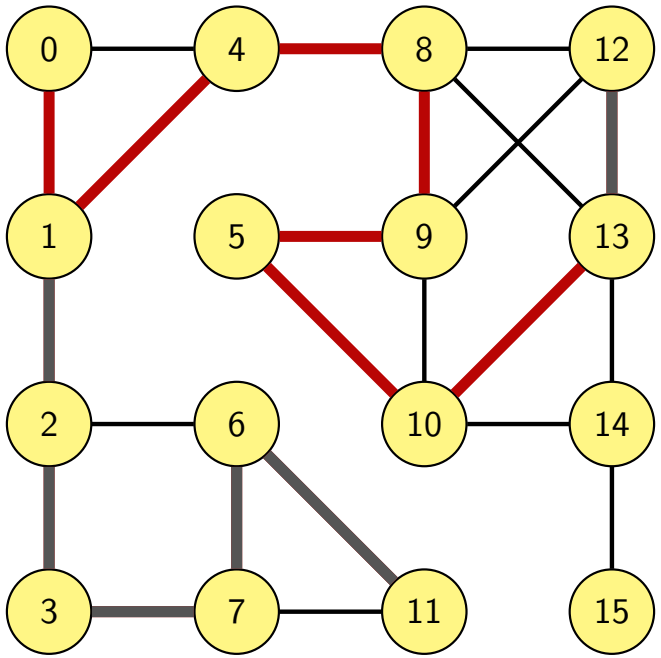
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

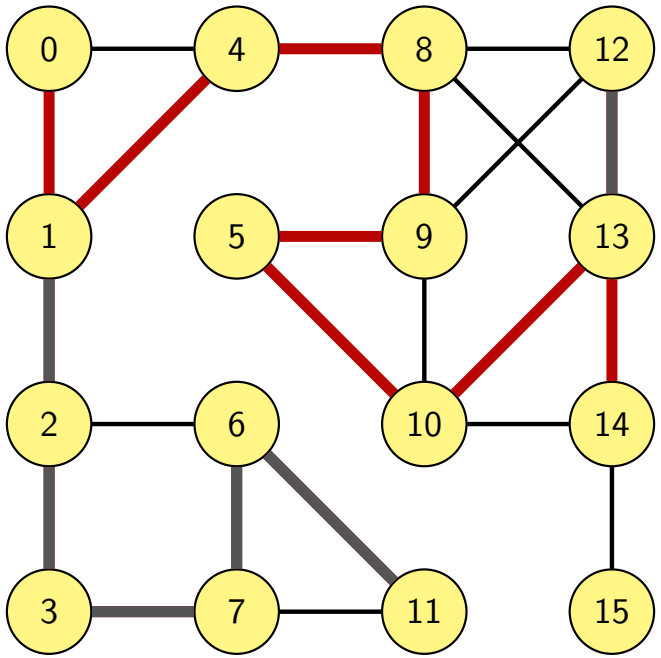
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

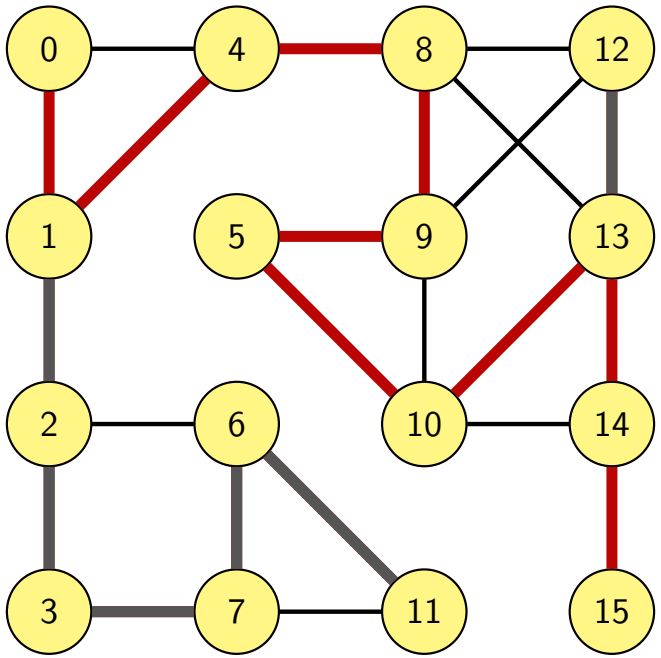
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

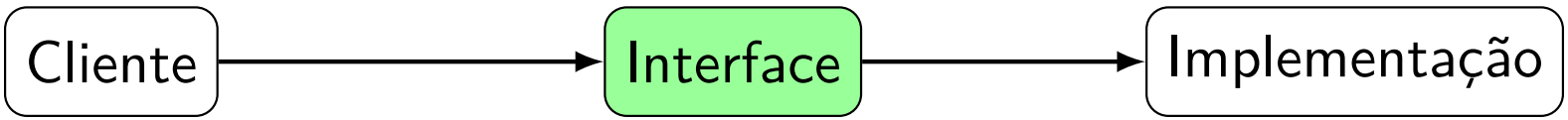
Exemplo - Existe caminho de 0 até 15?



Vamos fazer uma busca em profundidade:

- Vá o máximo possível em uma direção
- Se não encontrou o vértice, volte e tente outro caminho por um vértice **não visitado**

TAD - Interface



grafo.h

```
12 //Funções
13 Grafo* criar_grafo(int n);
14 void destruir_grafo(Grafo *p);
15
16 void inserir_aresta(Grafo *p, int u, int v);
17 void remover_aresta(Grafo *p, int u, int v);
18
19 int tem_aresta(Grafo *p, int u, int v);
20 void imprimir_arestas(Grafo *g);
21
22 int get_vertices(Grafo *p);
23
24 int existe_caminho(Grafo *p, int u, int v);//busca em profundidade
```

Grafo - Busca em profundidade

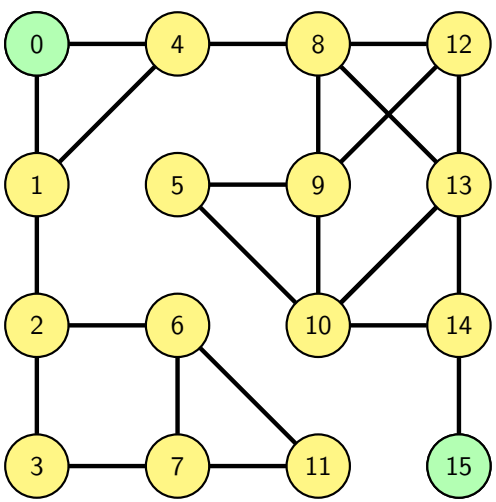
grafo.c

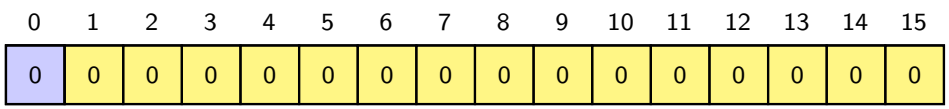
```
90 int existe_caminho(Grafo *p, int u, int v) {
91   int *visitado = (int*) malloc (p->n * sizeof(int));
92   int i;
93   for (i = 0; i < p->n; i++)
94     visitado[i] = 0;
95   int encontrou = busca_recursiva(p, visitado, u, v);
96   free(visitado);
97   return encontrou;
98 }
```

Grafo - Busca em profundidade

grafo.c

```
78 int busca_recursiva(Grafo *p, int *visitado, int u, int v) {//profundidade
79   int w;
80   if (u == v) return 1; //sempre existe caminho de u para v
81   visitado[u] = 1;
82   for (w = 0; w < p->n; w++){
83     if (tem_aresta(p, u, w) && !visitado[w])
84       if (busca_recursiva(p, visitado, w, v))
85         return 1;
86   }
87   return 0;
88 }
```





Cliente

exemplo1.c

```
5  int main() {
6    int n, m, i, u, v;
7    scanf("%d %d", &n, &m);
8    Grafo *G = criar_grafo(n);
9    for (i = 0; i < m; i++) {
10     scanf("%d %d", &u, &v);
11     inserir_aresta(G, u, v);
12   }
13   imprimir_arestas(G);
14   for (u = 0; u < n; u++)
15     for (v = u+1; v < n; v++)
16       if(existe_caminho(G, u, v))
17         printf("%d <-> %d\n", u, v);
18   return 0;
19 }
```

Makefile

Vamos usar o Makefile para compilar:

```
1 LIB = grafo_matriz.o
```

```
1 exemplo1: exemplo1.c $(LIB)
2   gcc $^ -o $@
```

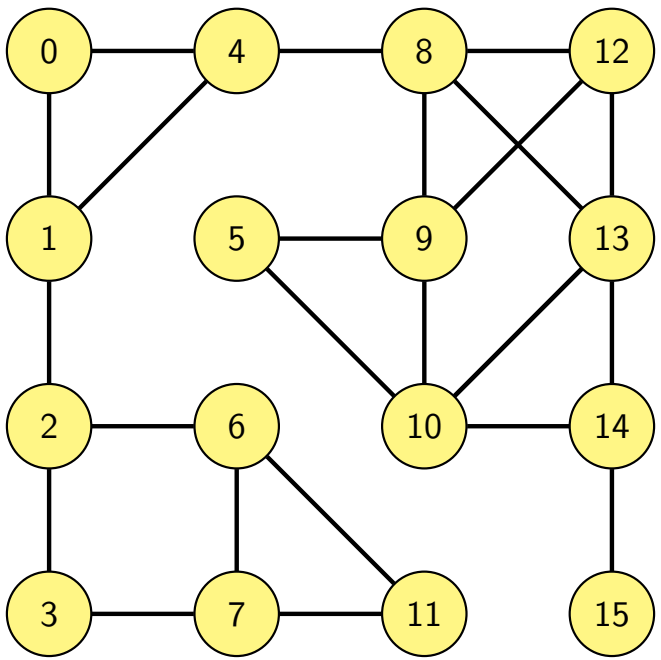
Vamos executar:

```
1  $ ./exemplo1 < teste1.in
2  {0,1}
3  {1,2}
4  {3,5}
5  {4,5}
6  0 <-> 1
7  0 <-> 2
8  1 <-> 2
9  3 <-> 4
10 3 <-> 5
11 4 <-> 5
```

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha

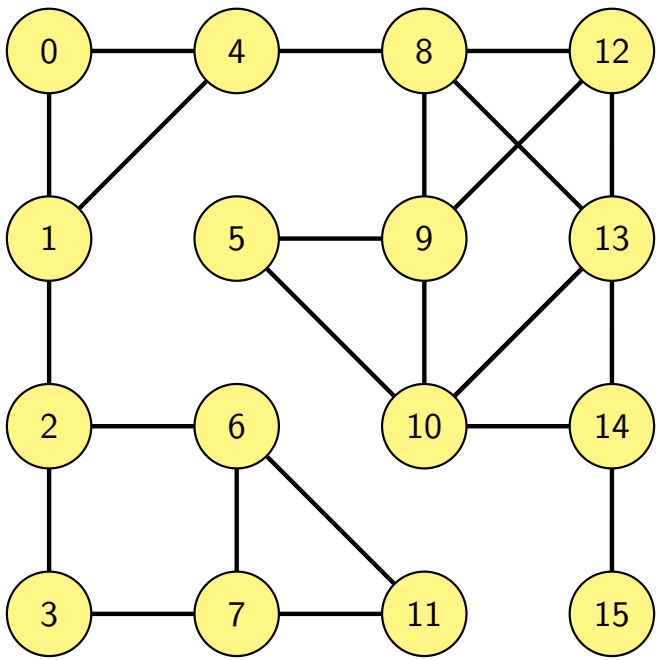


Pilha

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha

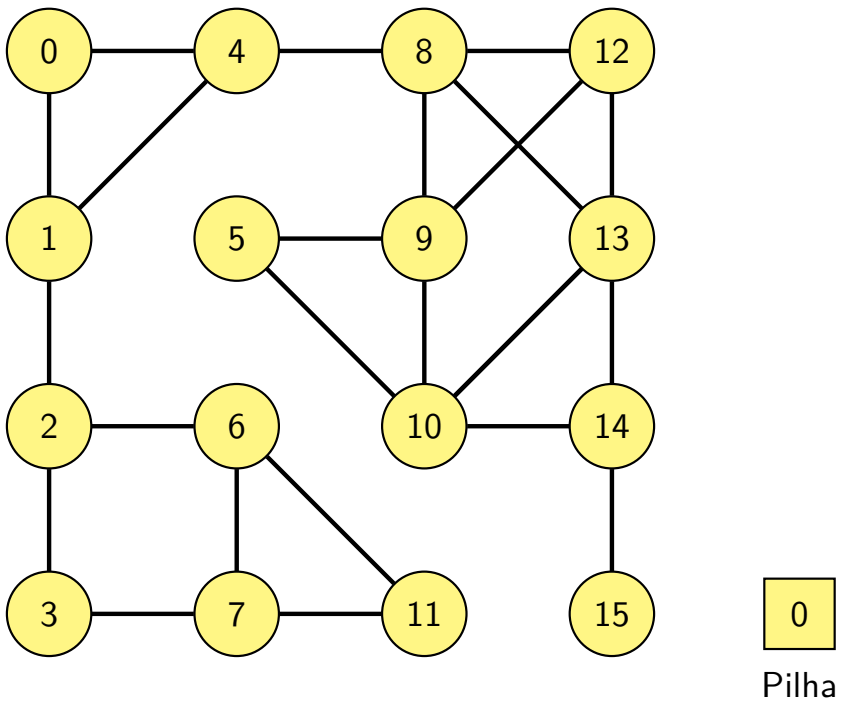


Pilha

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

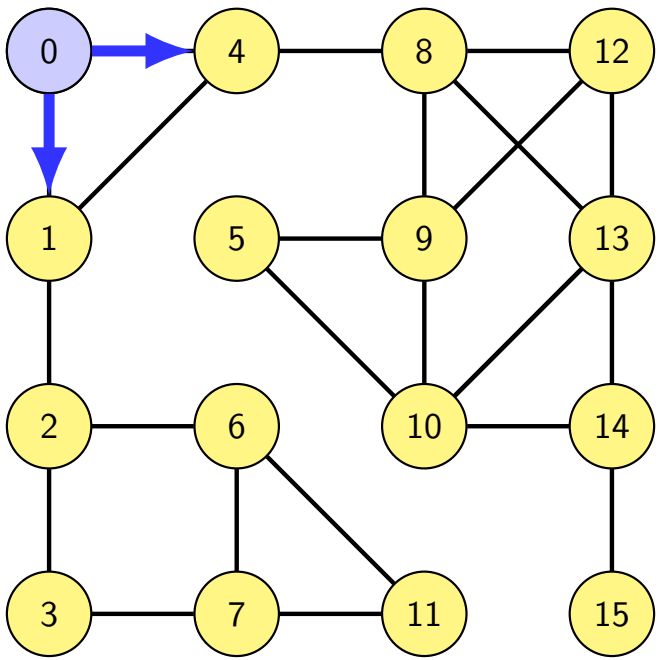
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha

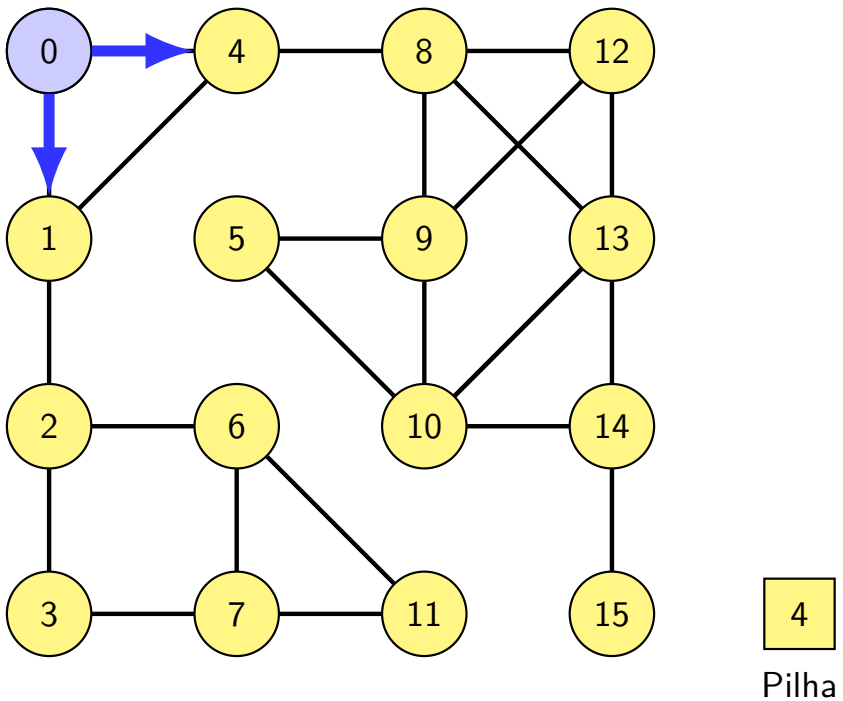


Pilha

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

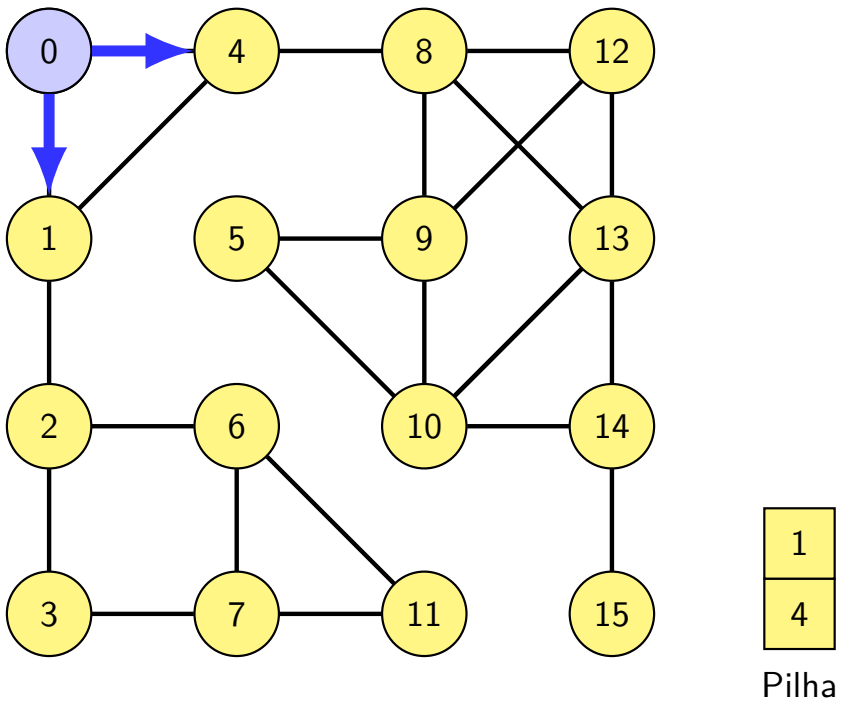
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

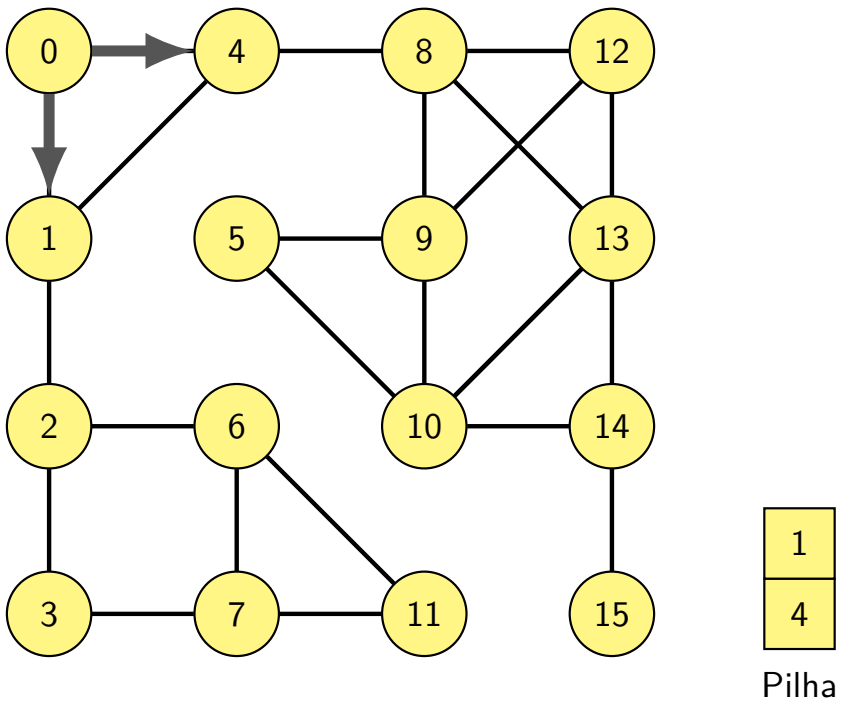
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

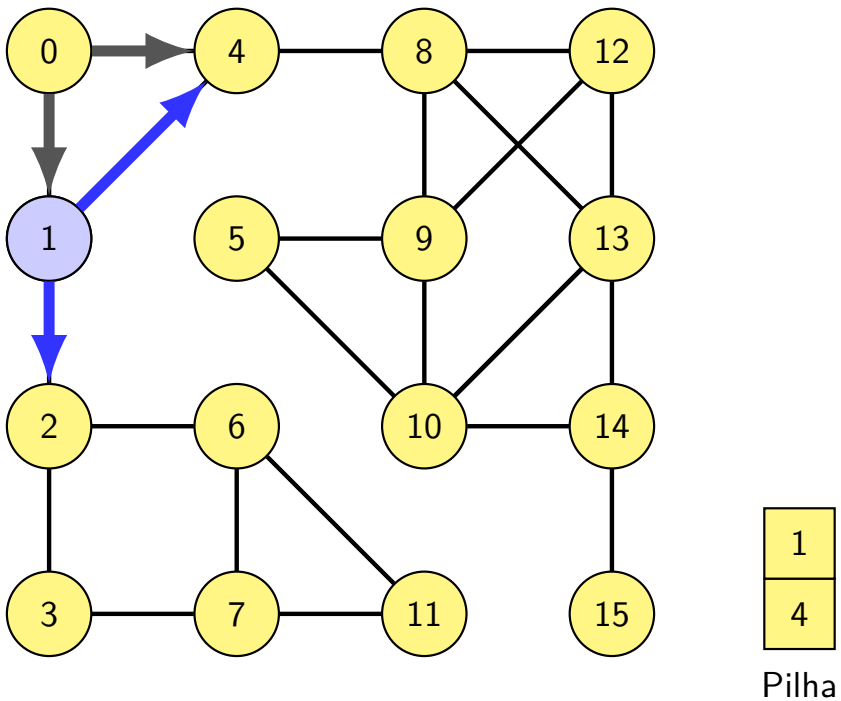
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

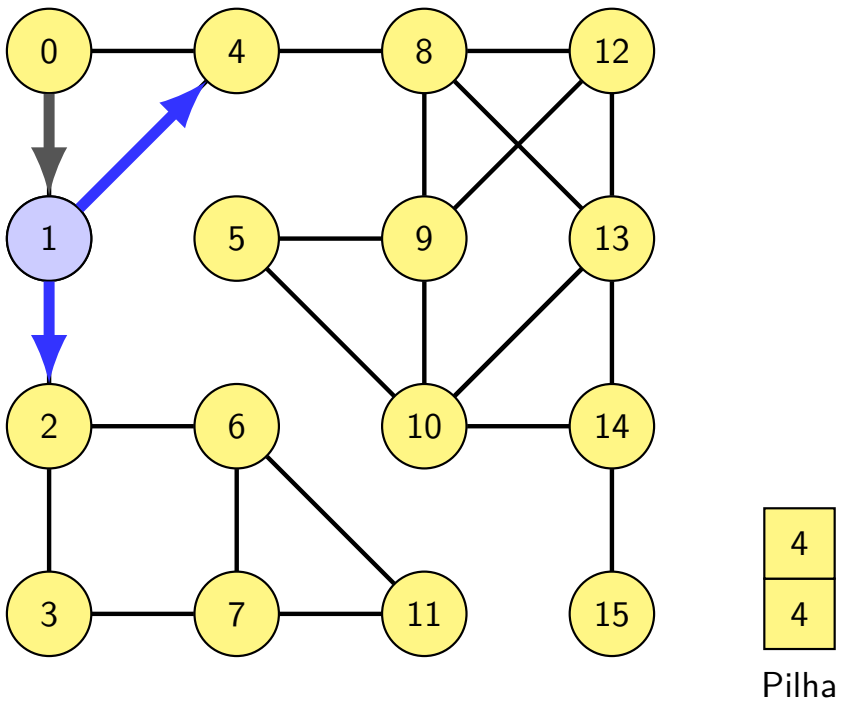
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

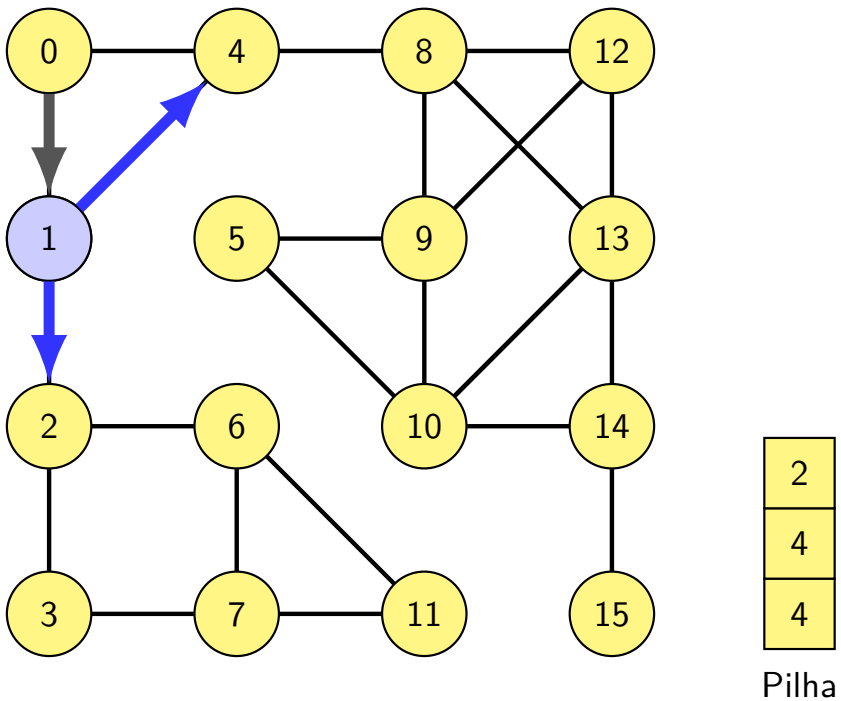
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

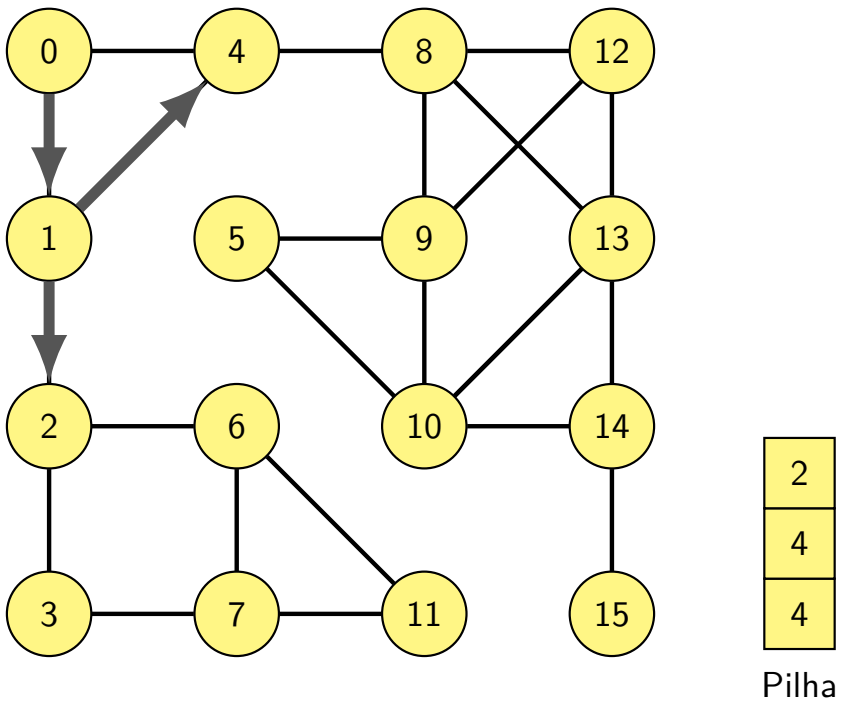
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

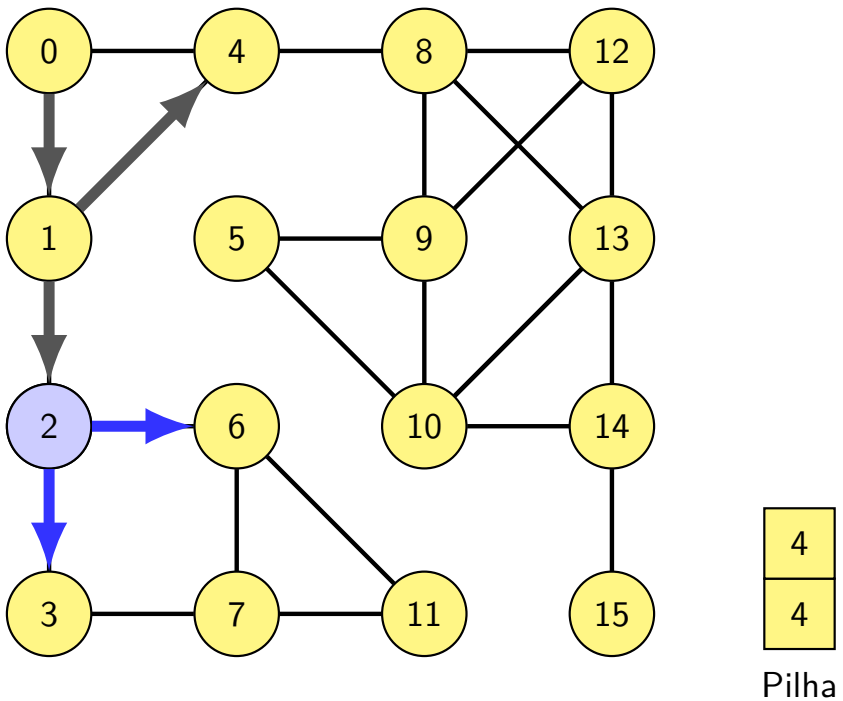
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

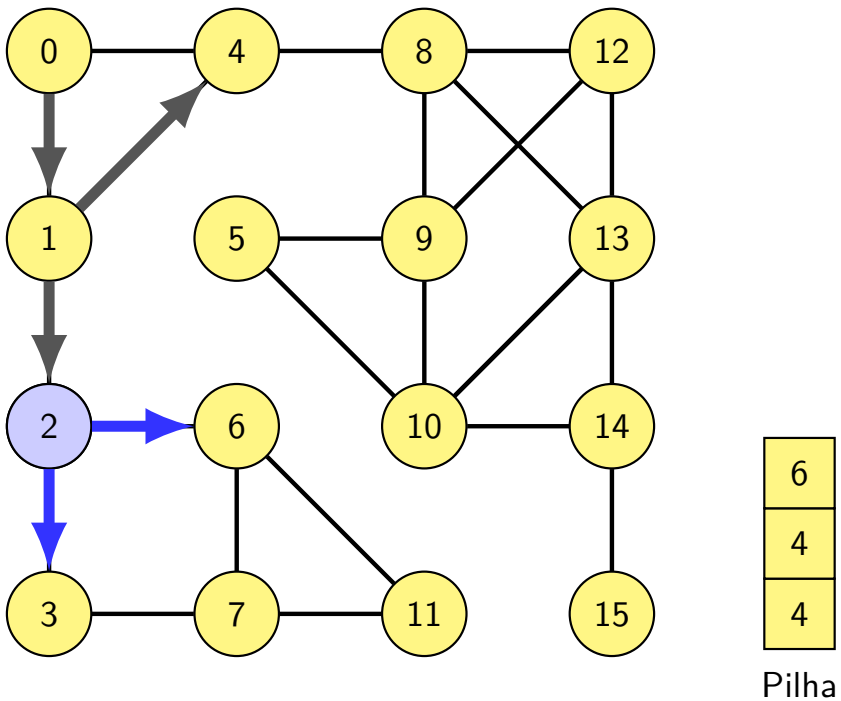
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

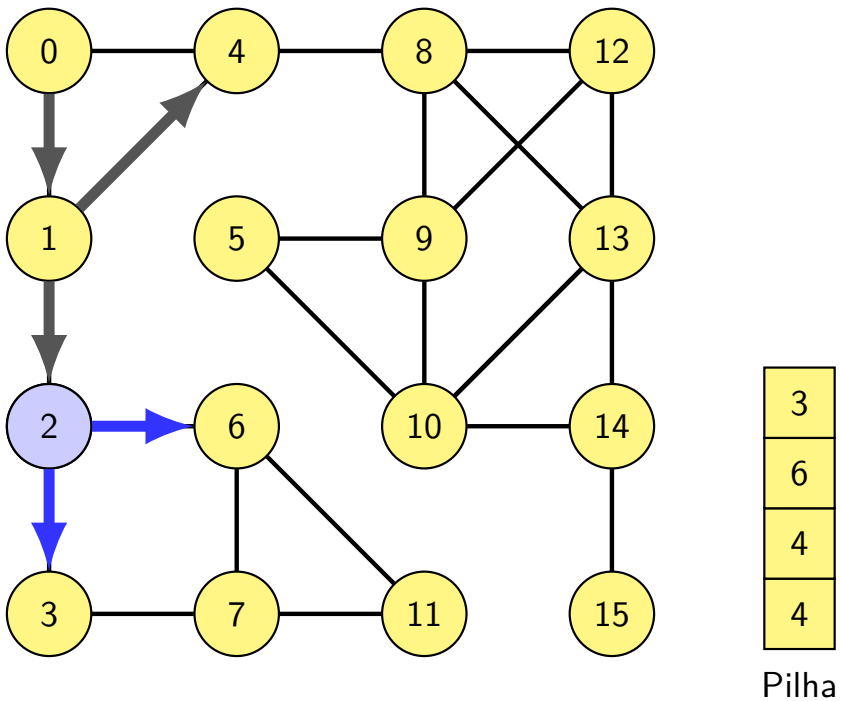
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

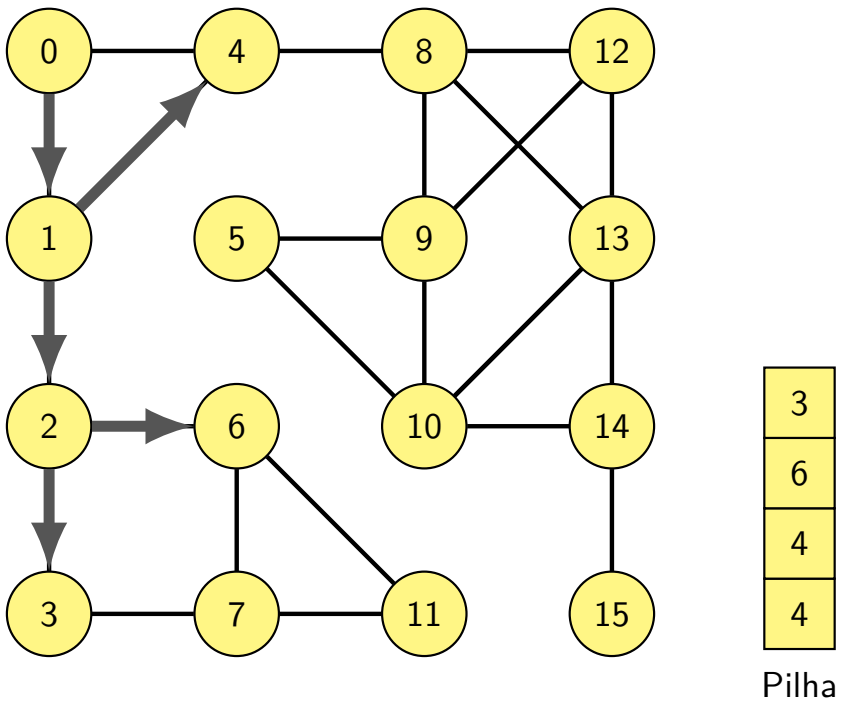
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

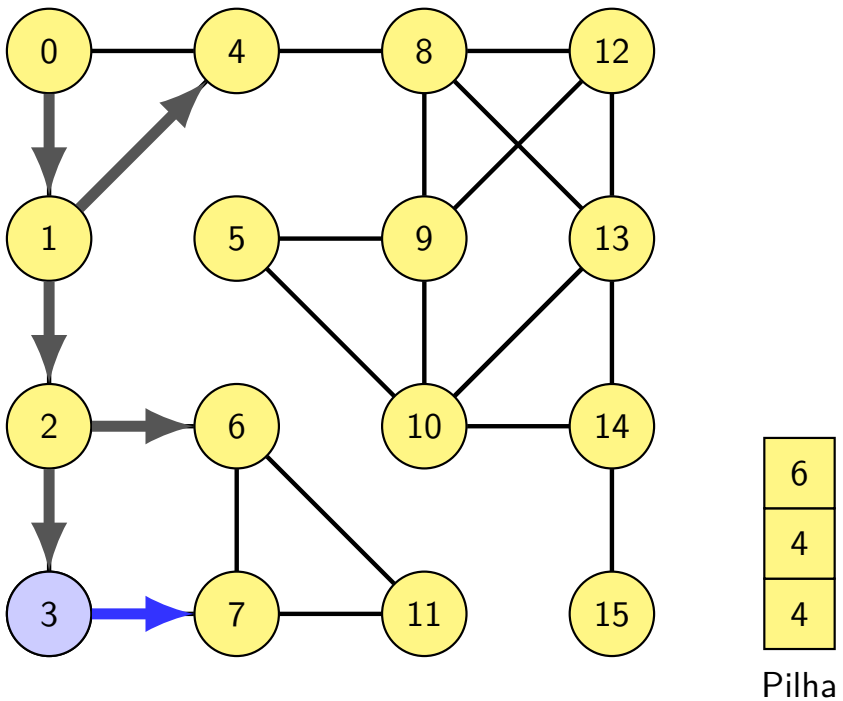
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

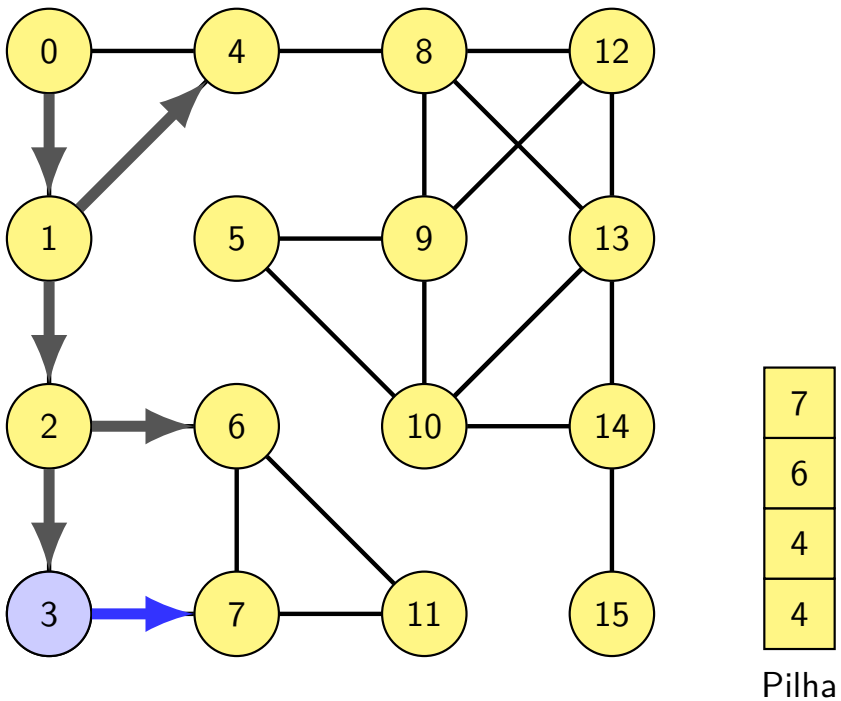
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

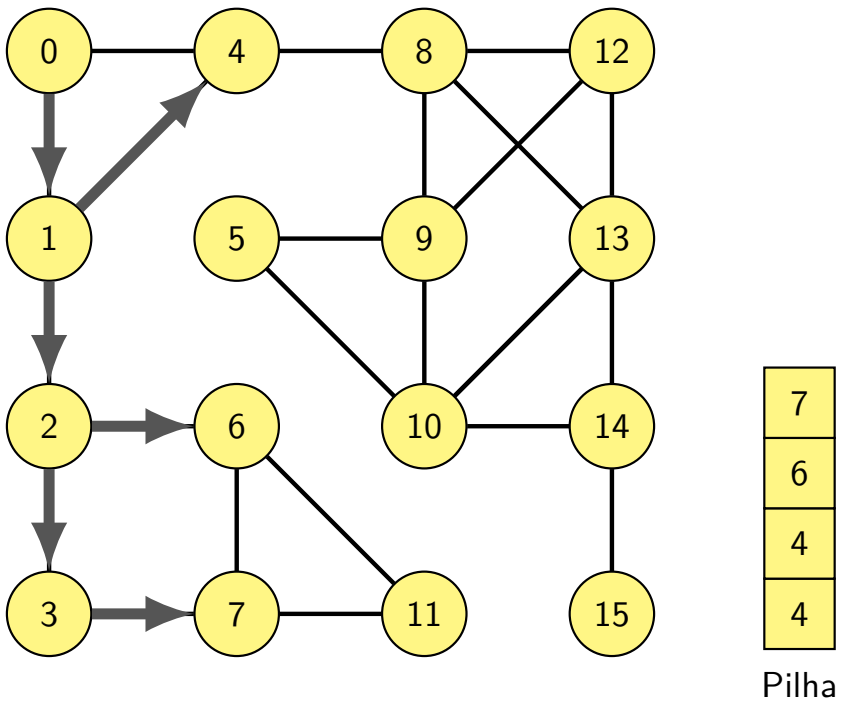
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

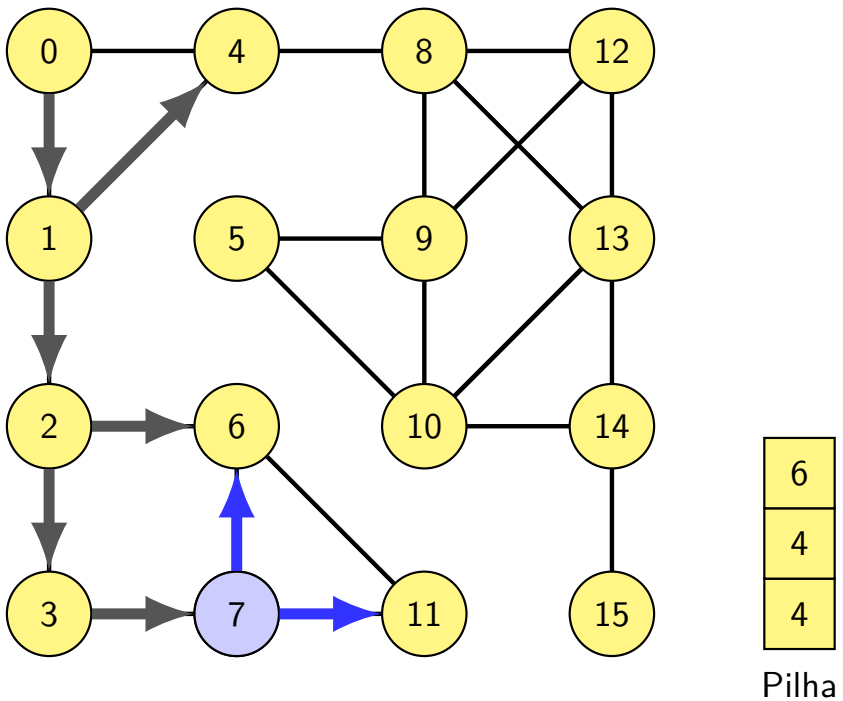
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

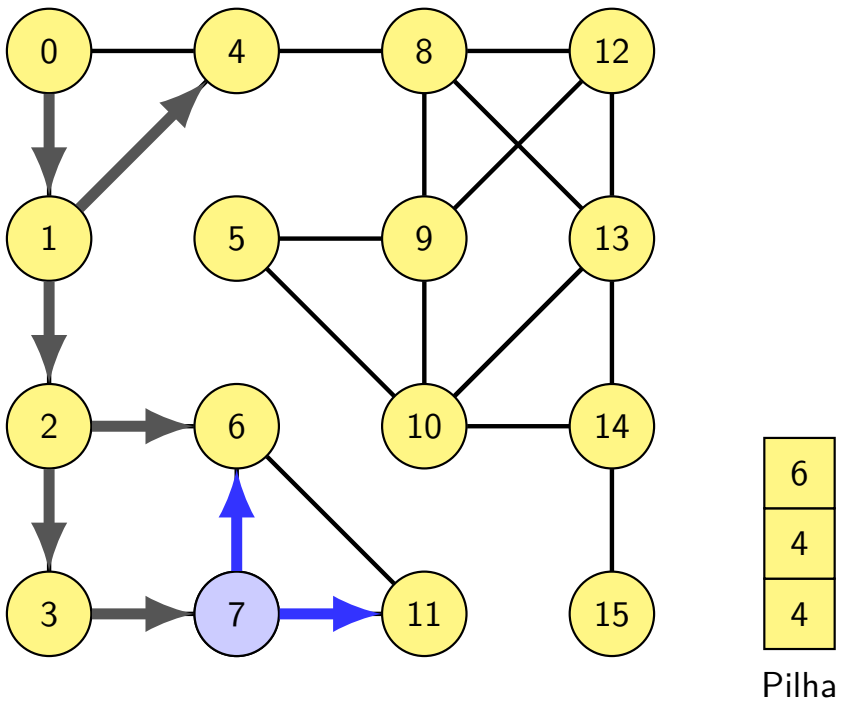
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

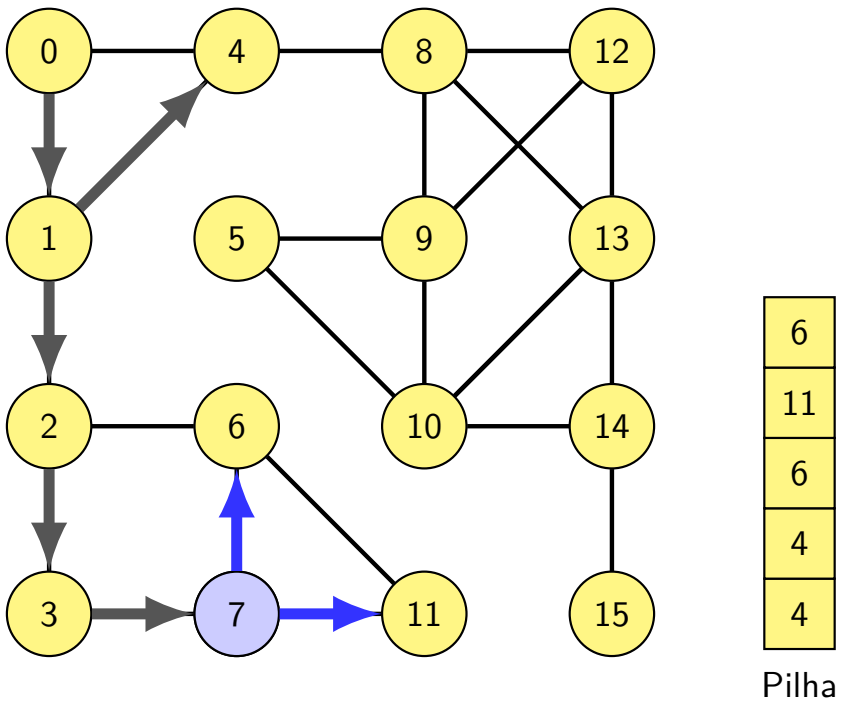
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

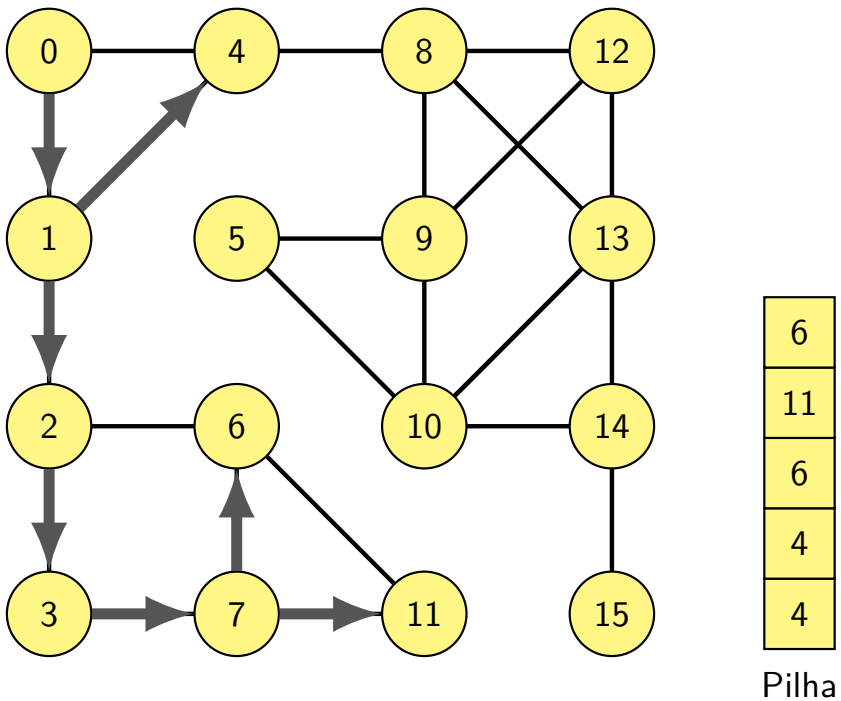
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

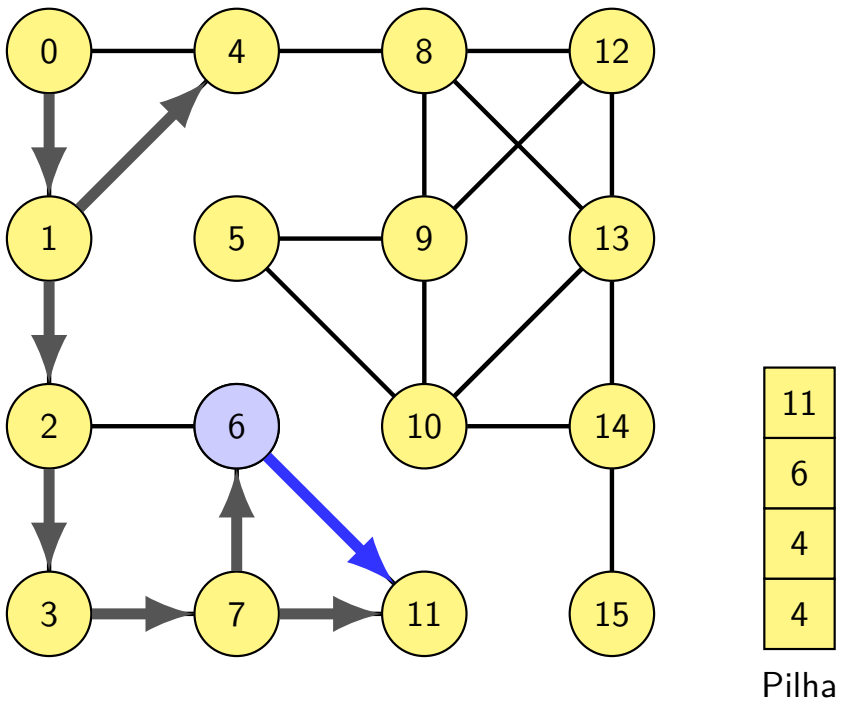
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

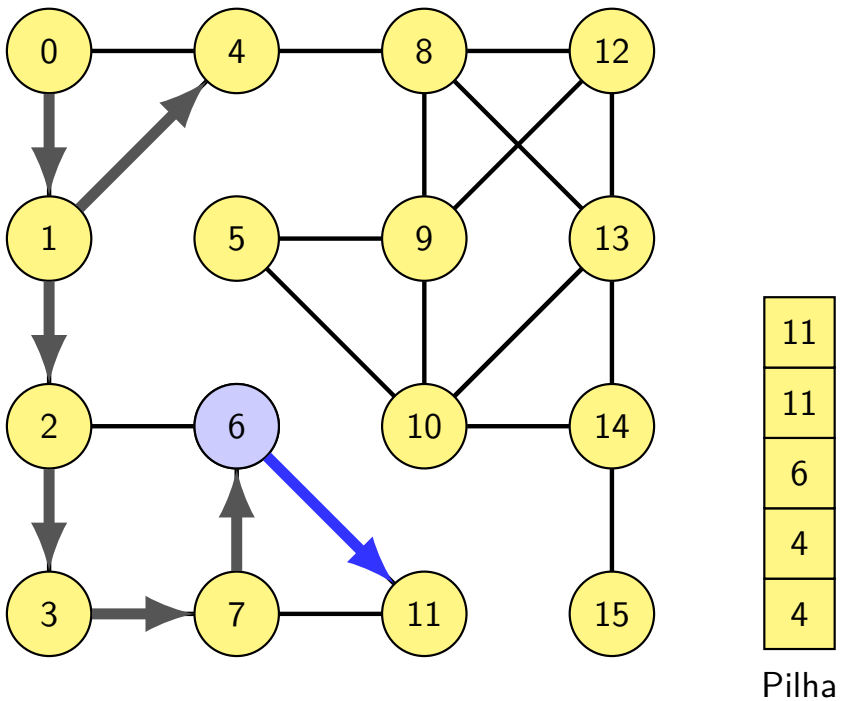
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

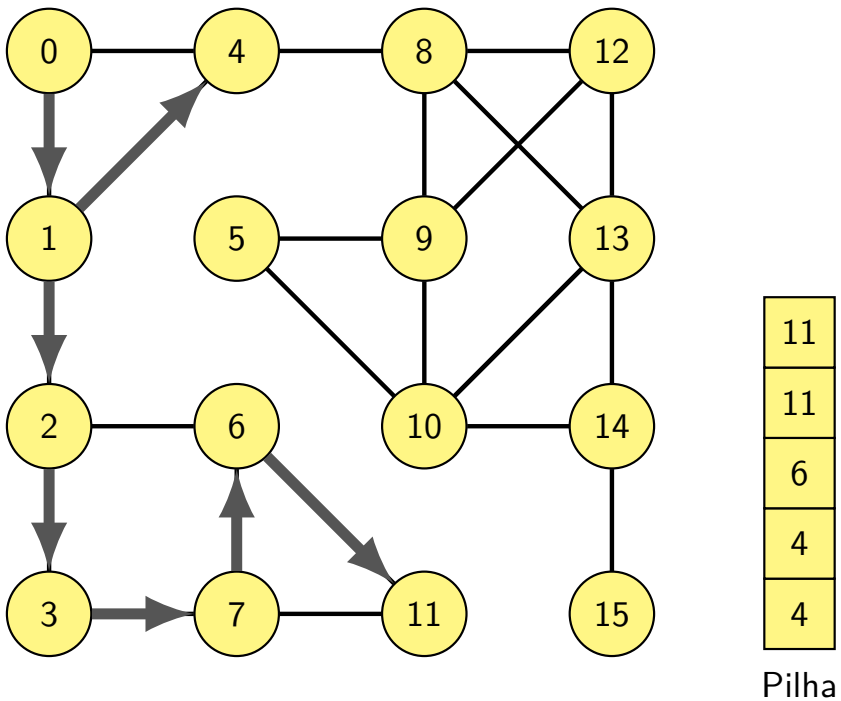
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

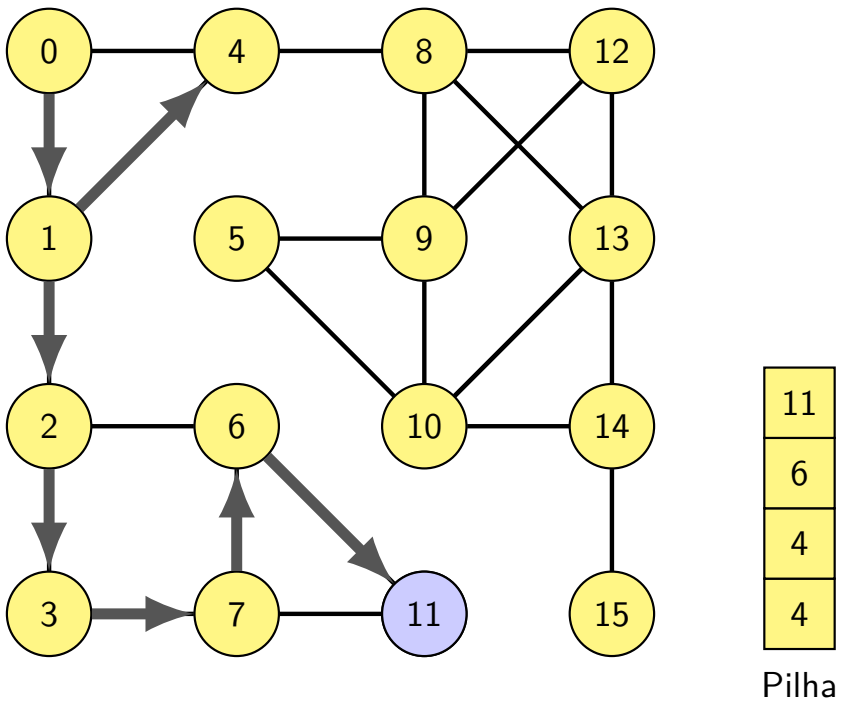
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

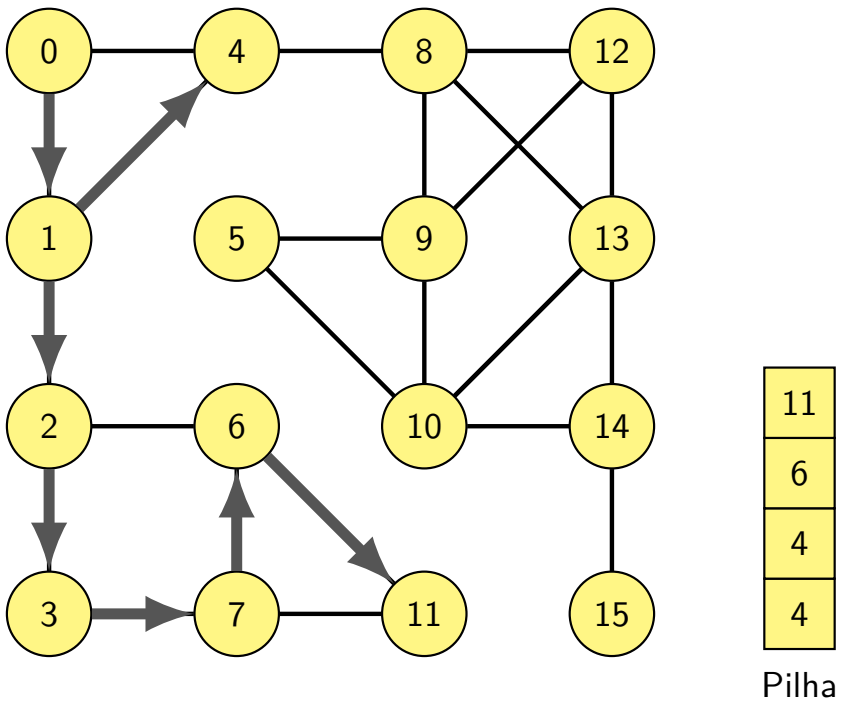
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

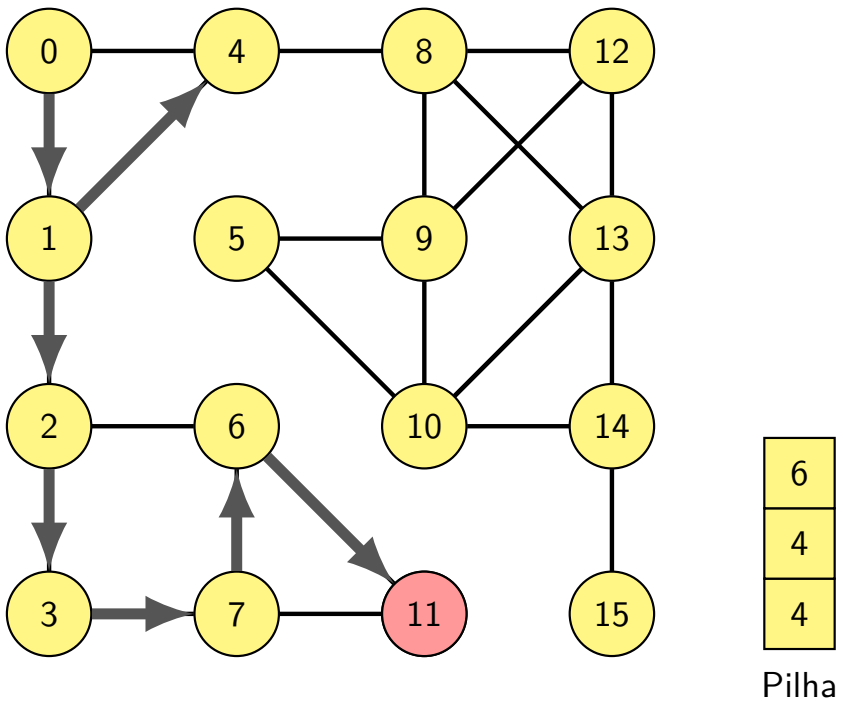
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

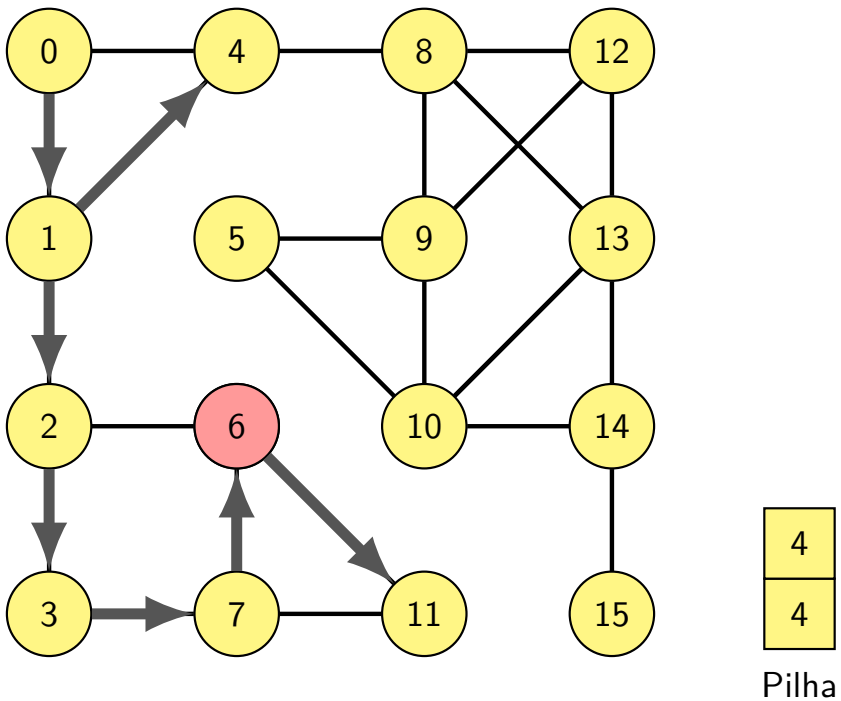
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

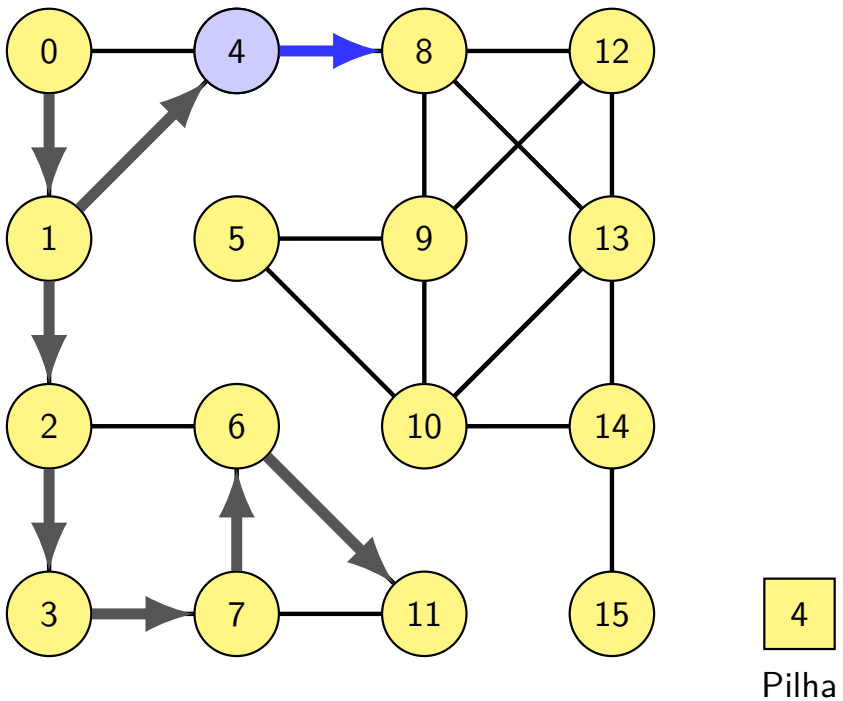
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

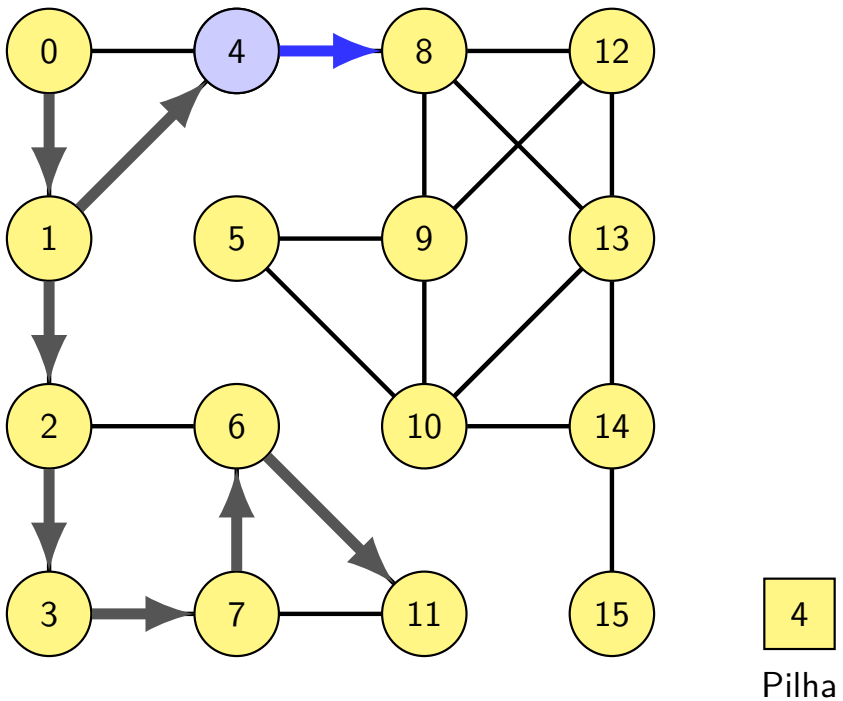
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

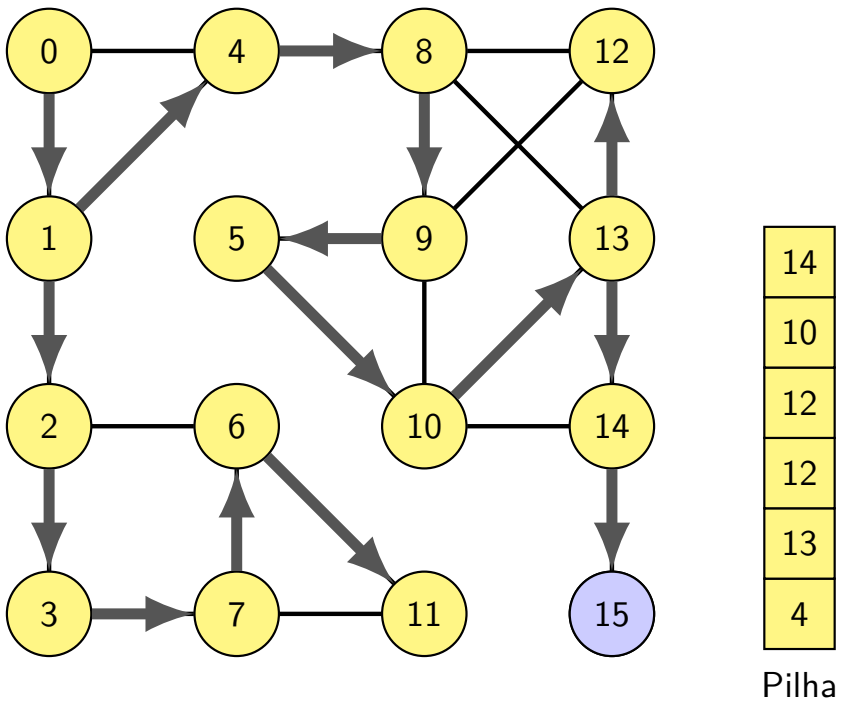
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

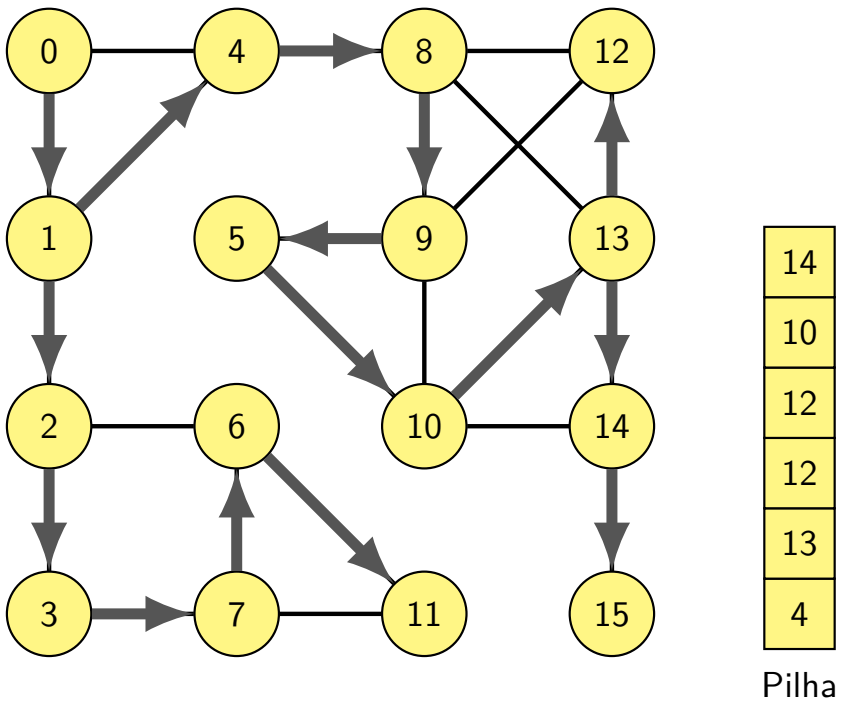
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

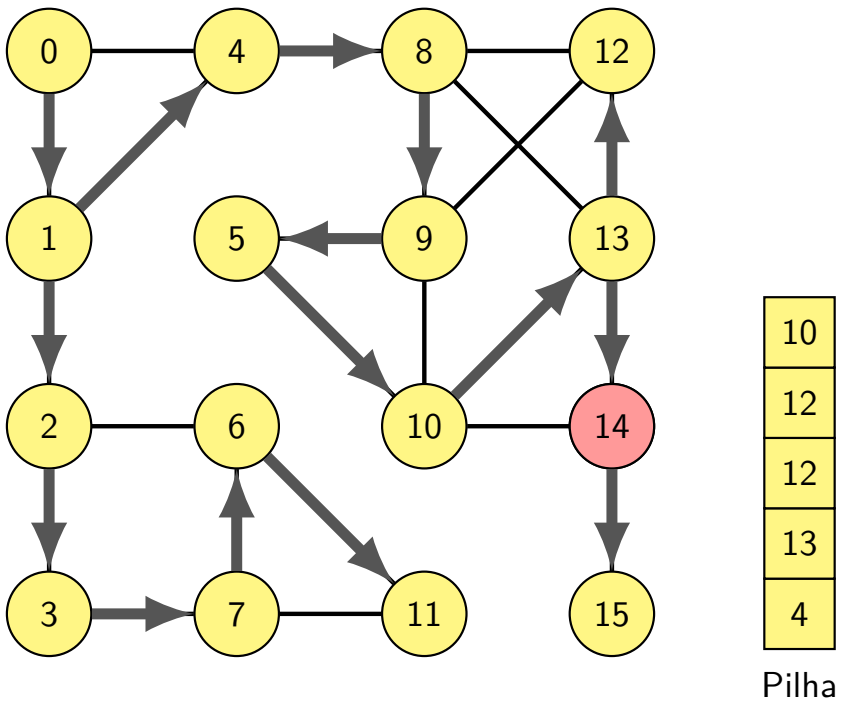
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

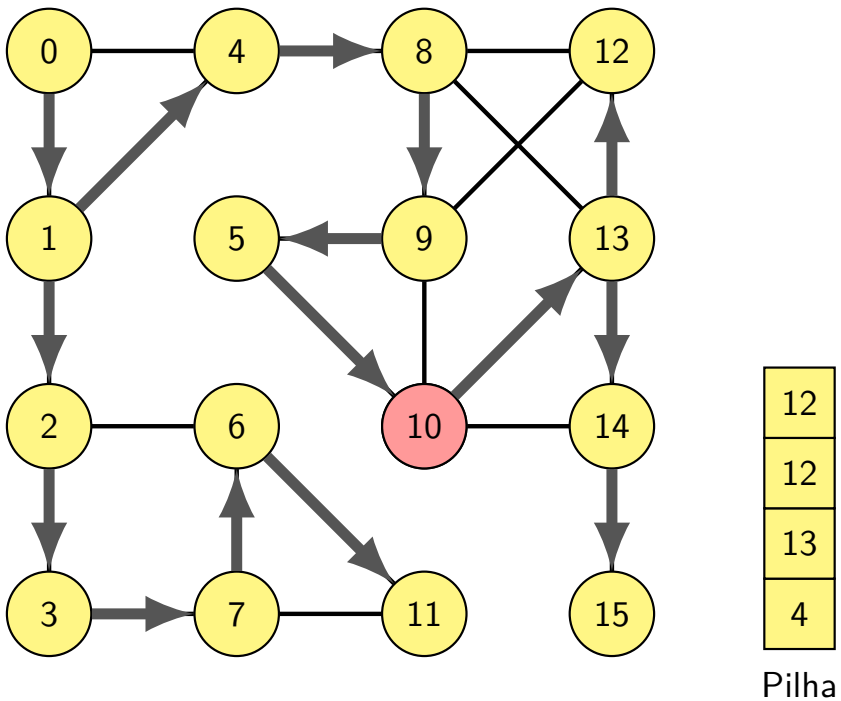
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

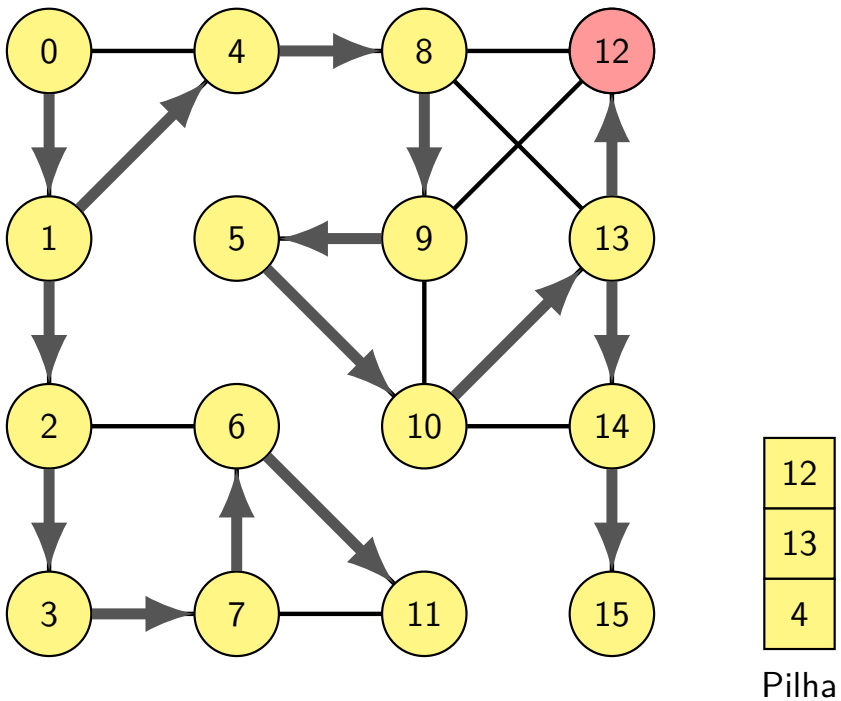
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

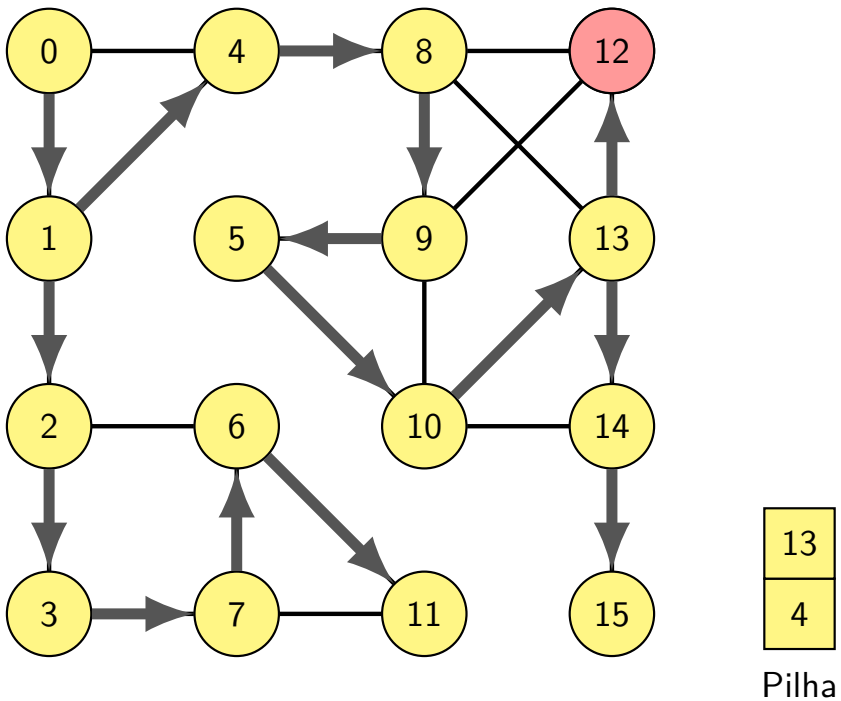
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

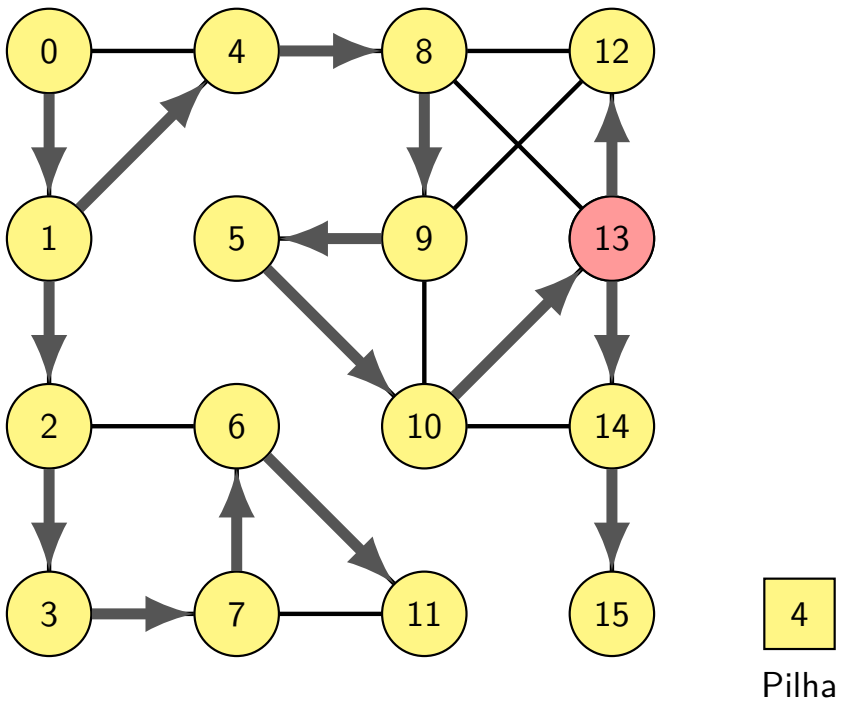
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

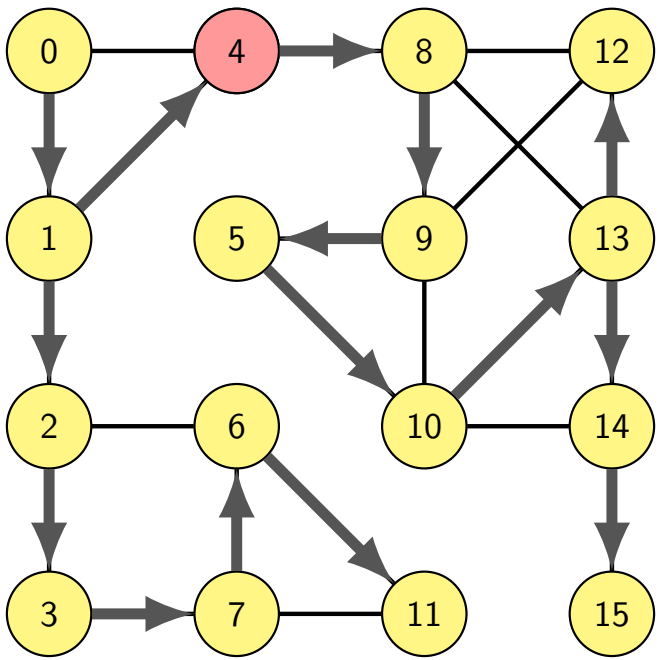
- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha

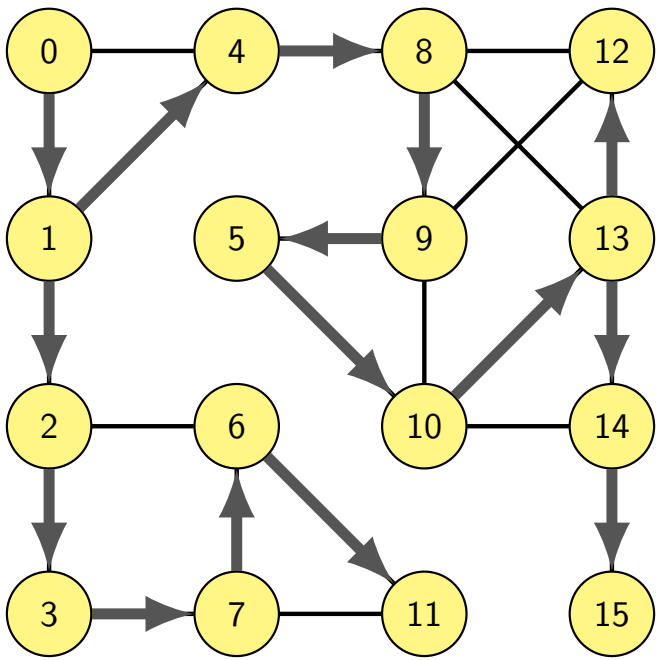


Pilha

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha



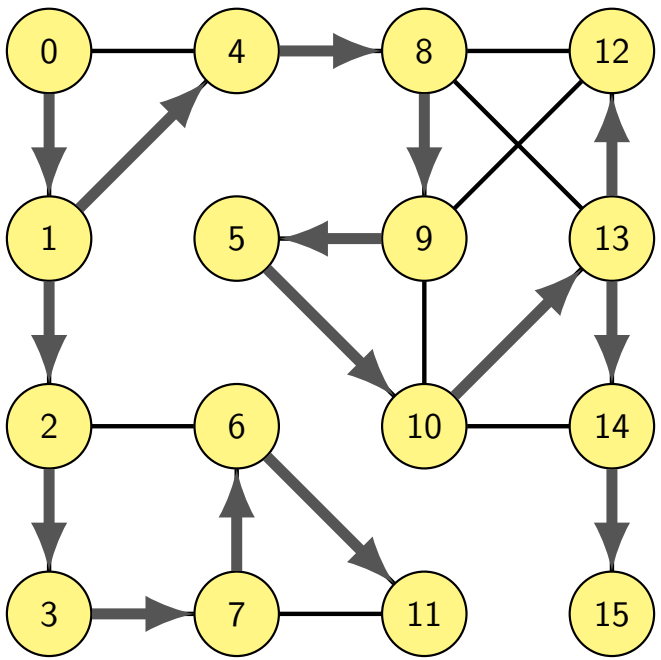
Pilha

Implementação na lista de exercícios.

Grafo - Busca em profundidade usando uma Pilha

Podemos fazer uma busca em profundidade usando **pilha**:

- A cada passo, desempilhamos um vértice **não visitado**
- E inserimos os **seus vizinhos** na pilha

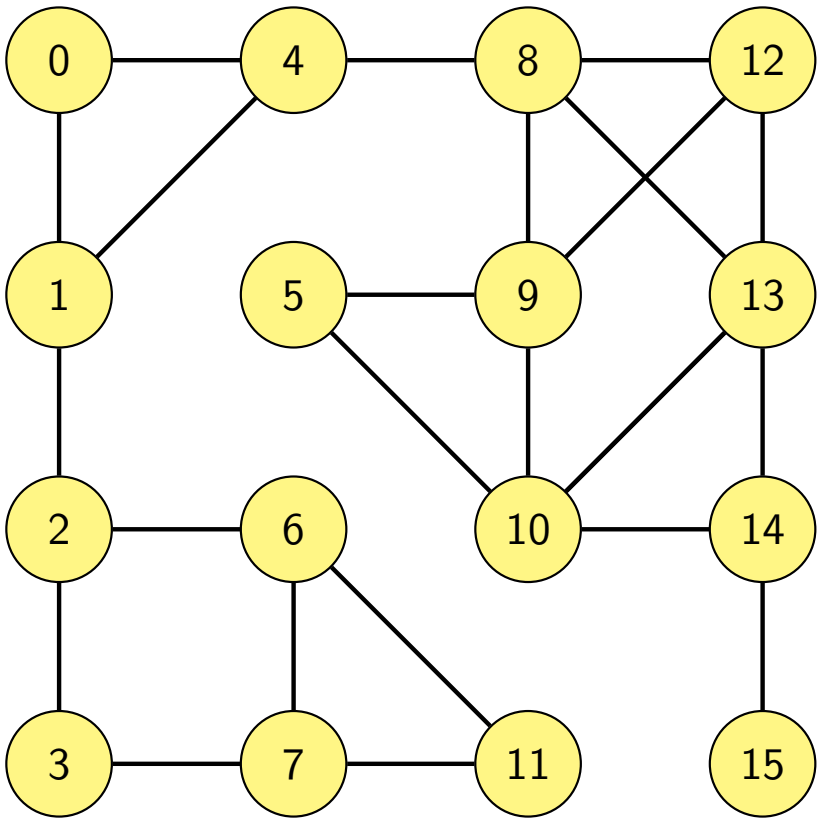


Pilha

E se usássemos uma Fila?

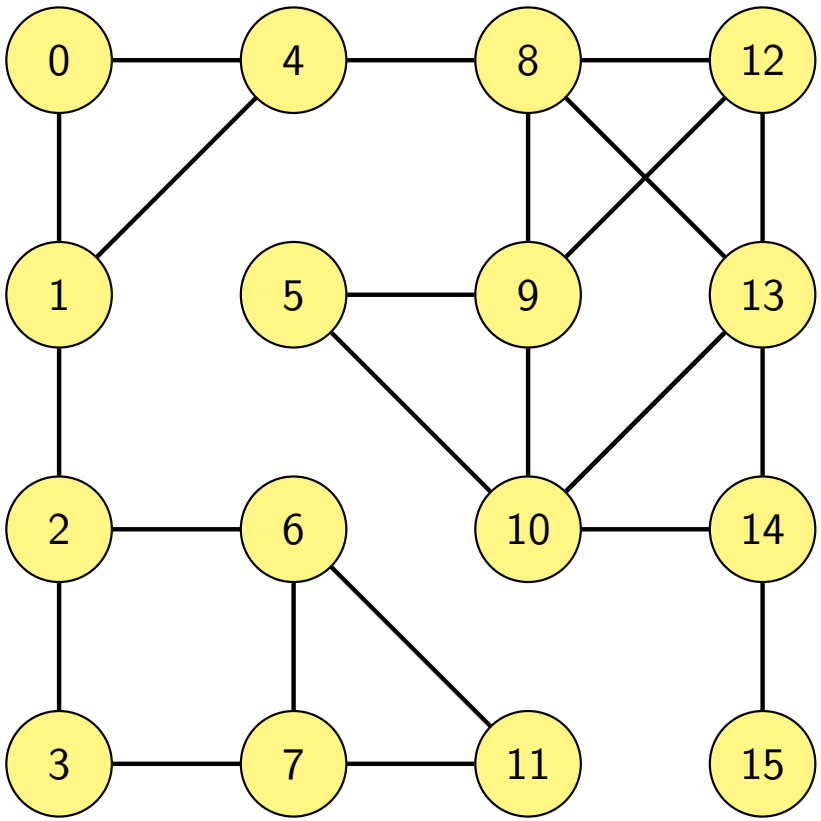
Roteiro

Busca em largura



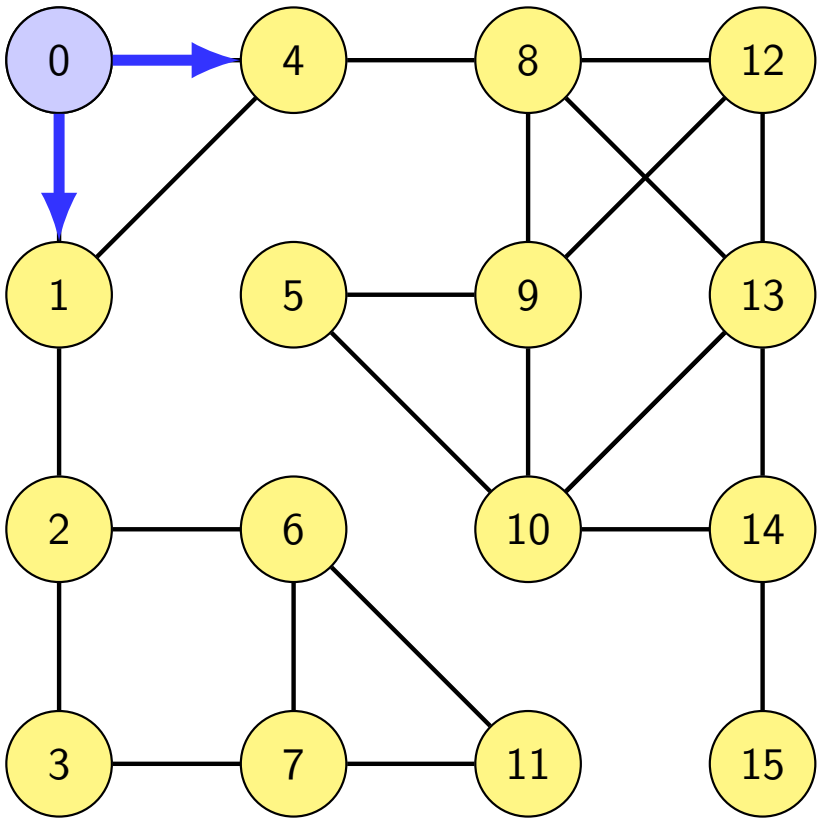
Fila

Busca em largura



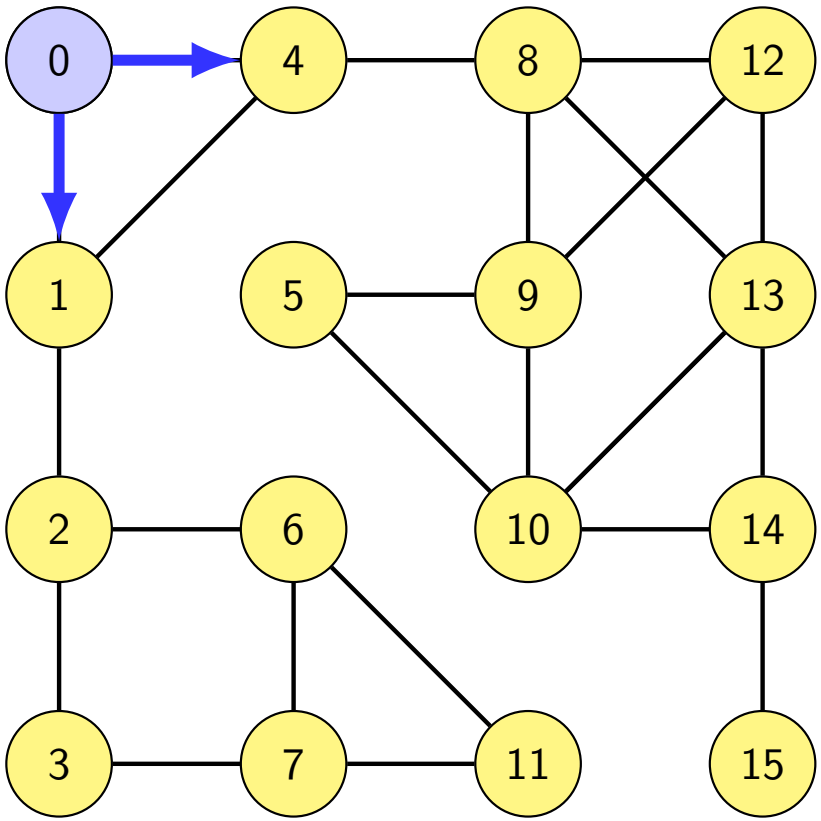
Fila | 0

Busca em largura



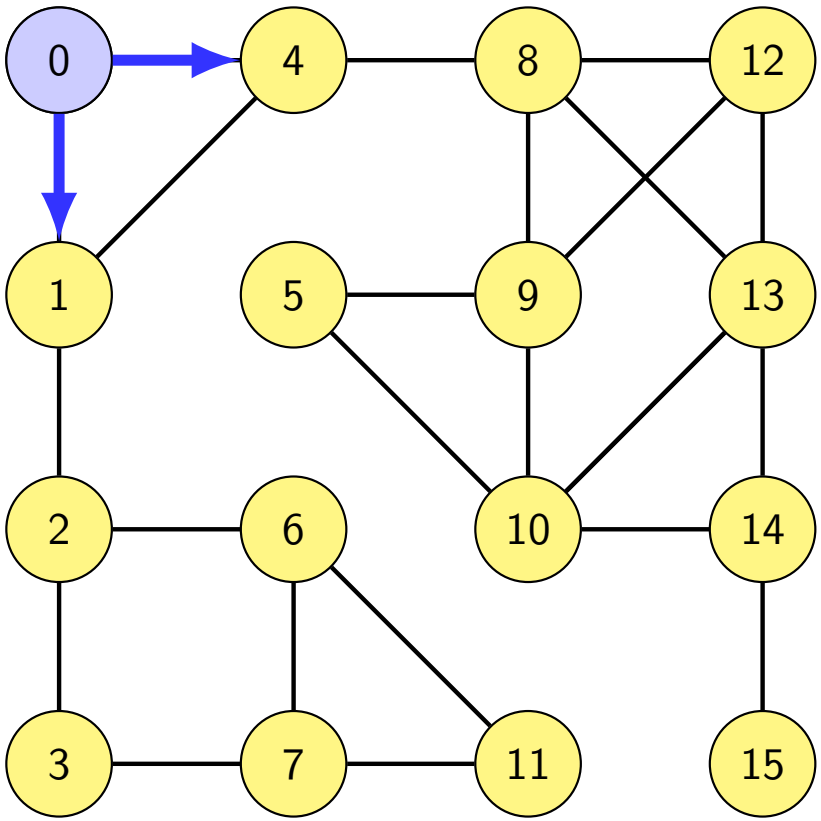
Fila

Busca em largura



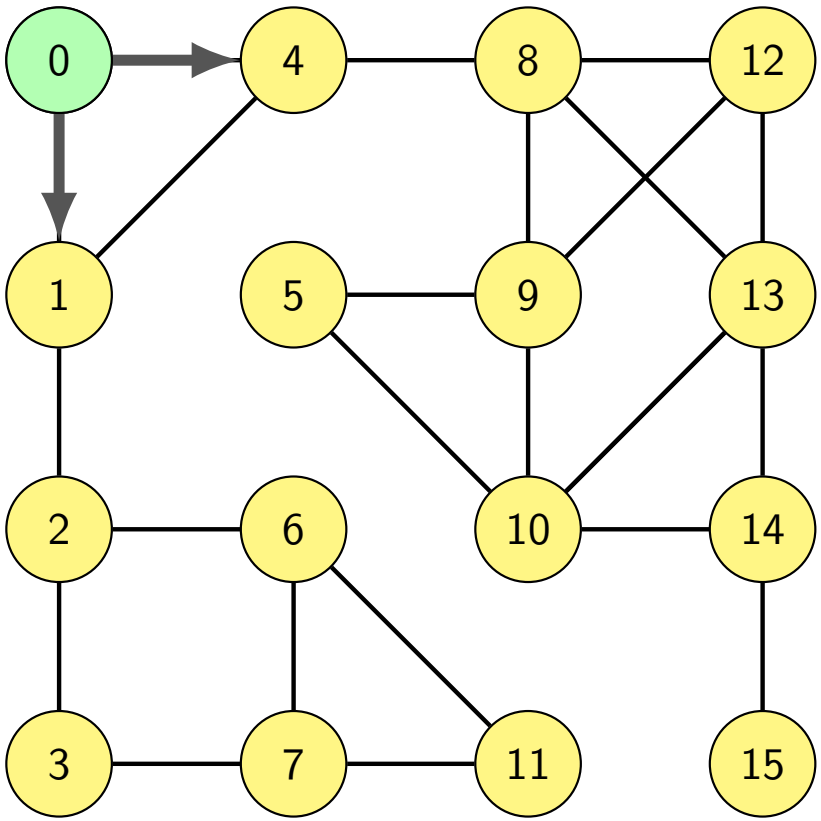
Fila | 1

Busca em largura



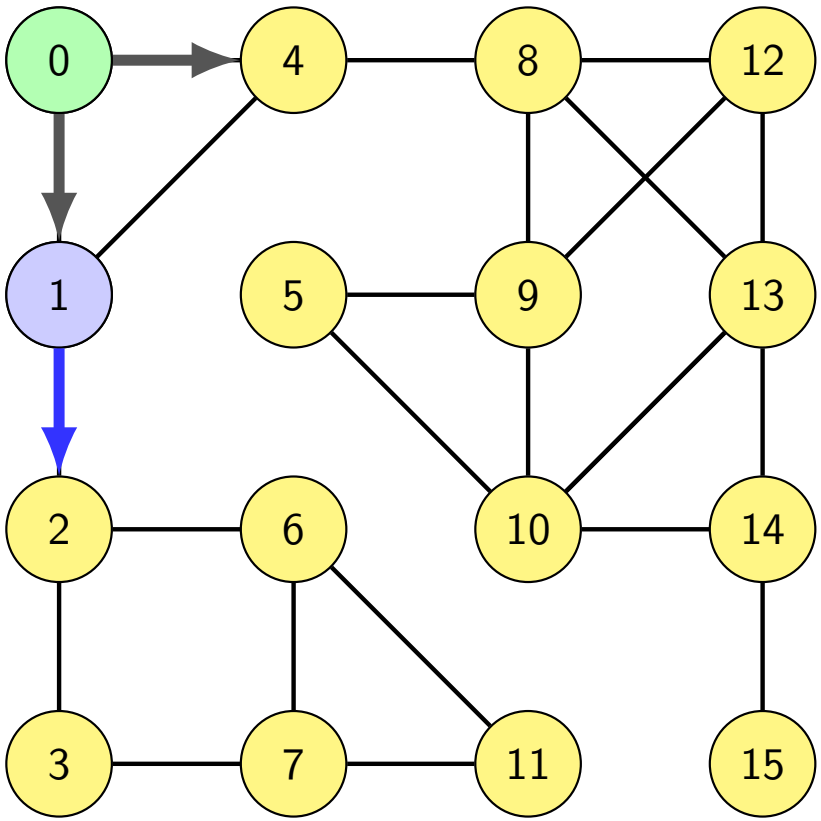
Fila | 1 | 4 |

Busca em largura



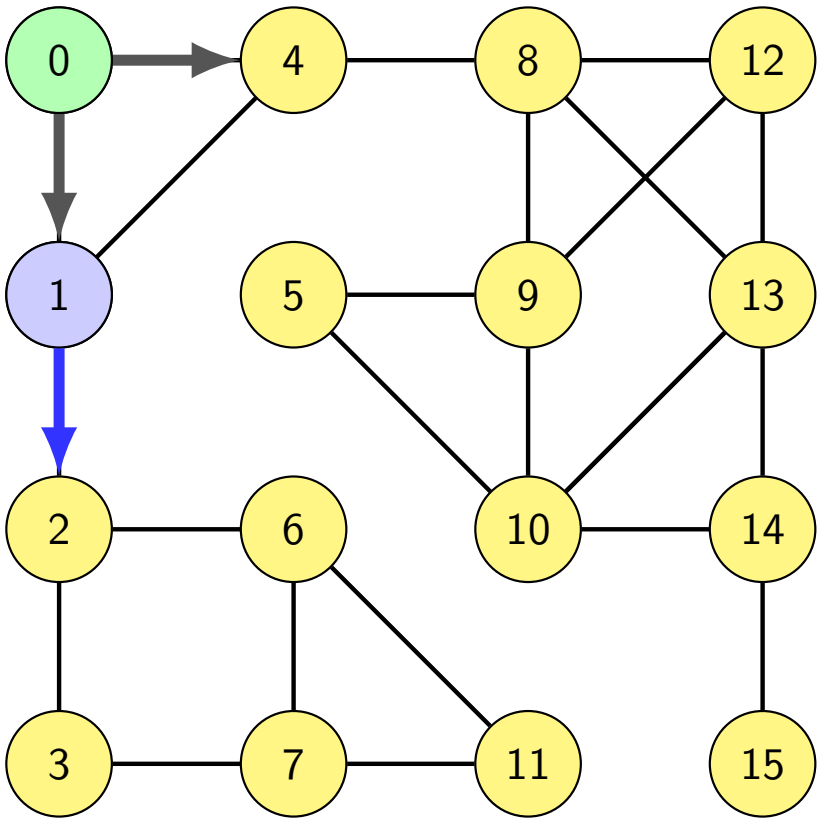
Fila | 1 | 4 |

Busca em largura



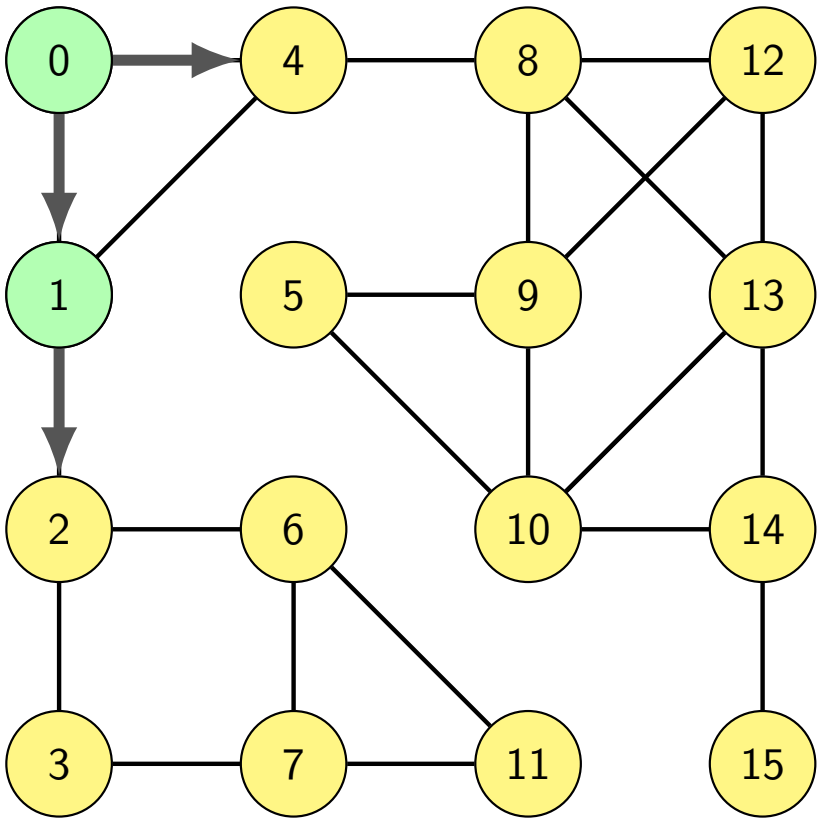
Fila | 4

Busca em largura



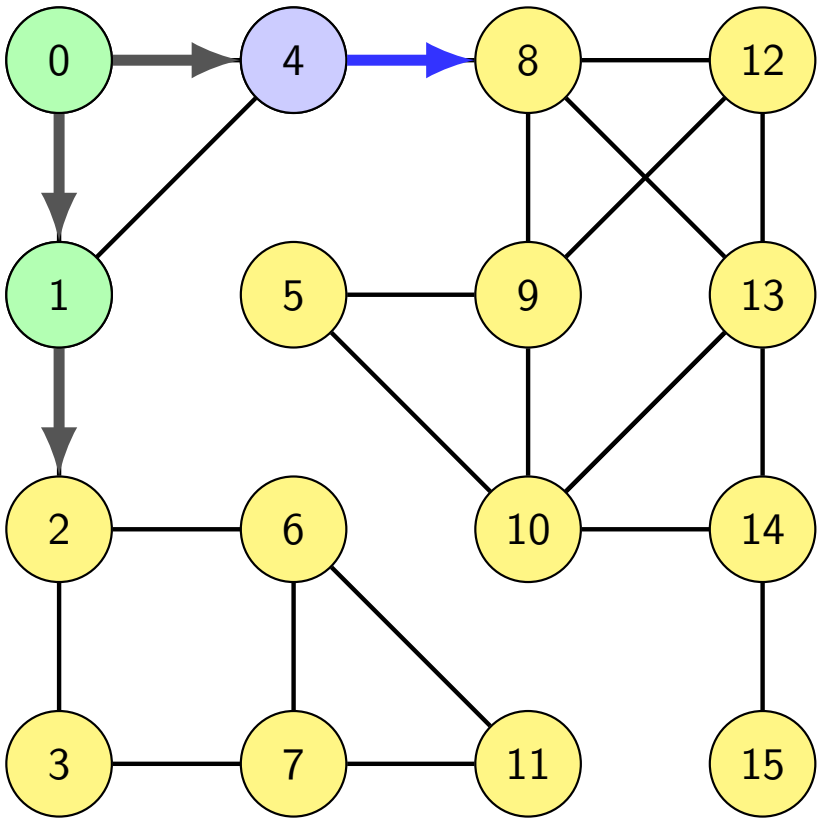
Fila | 4 | 2 |

Busca em largura



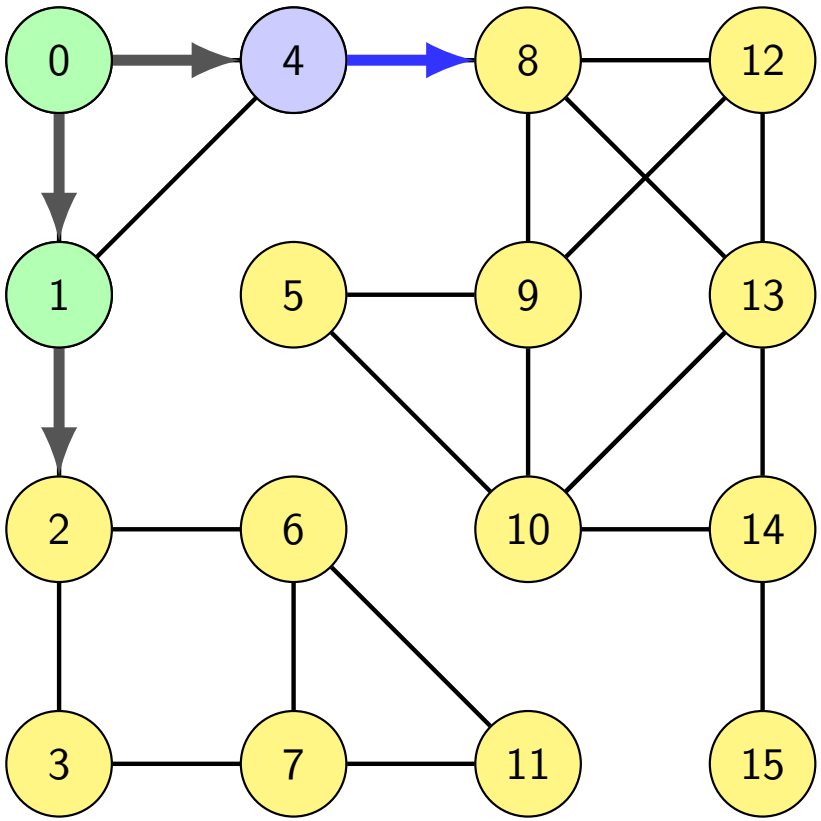
Fila | 4 | 2 |

Busca em largura



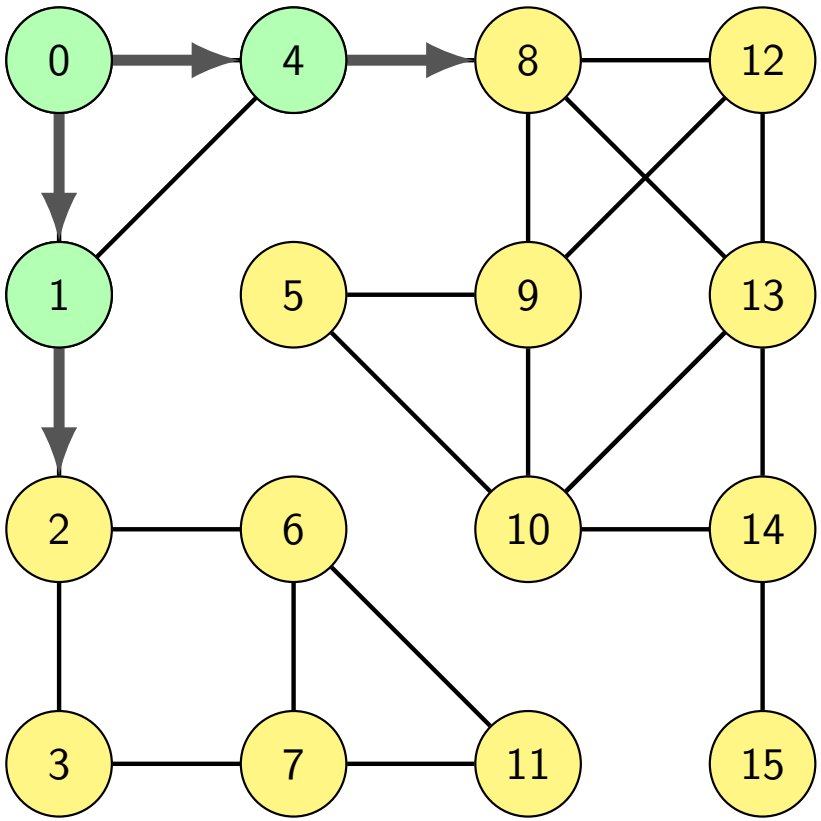
Fila | 2

Busca em largura



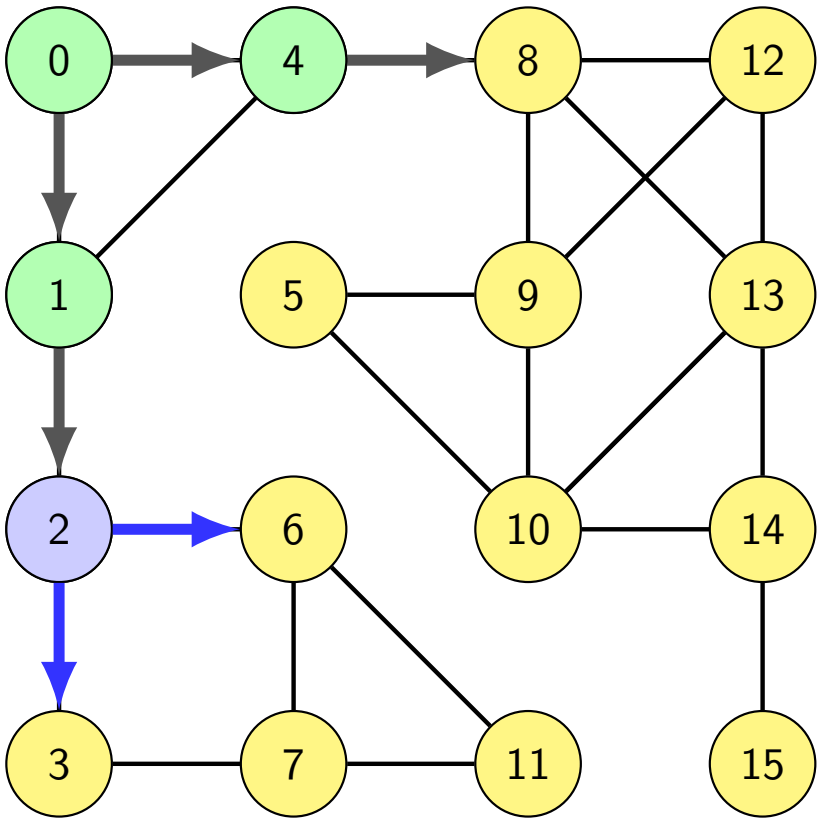
Fila | 2 | 8 |

Busca em largura



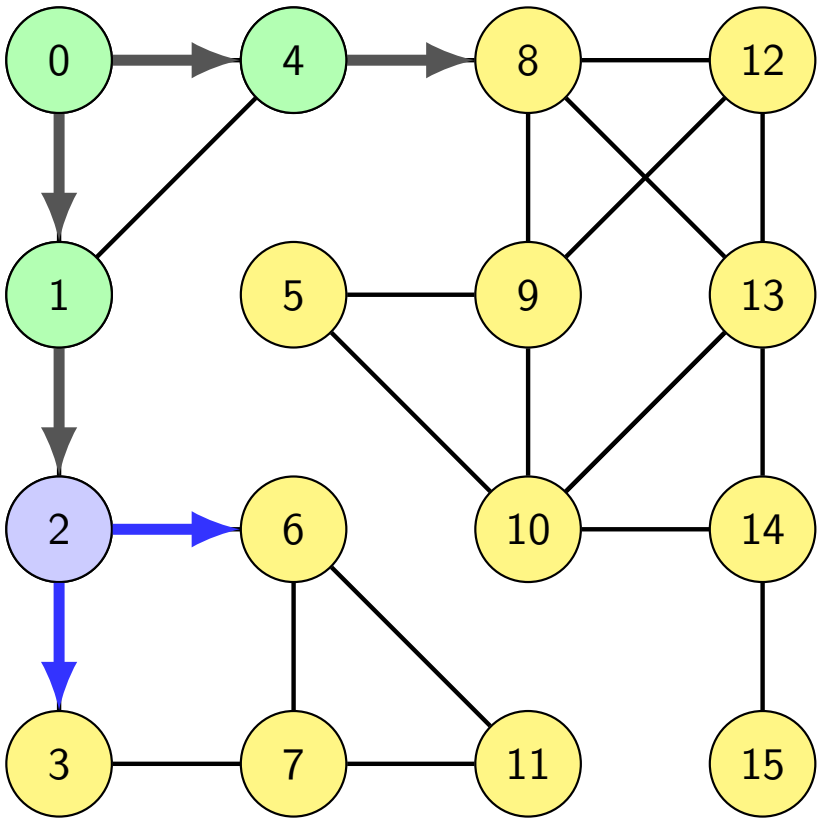
Fila | 2 | 8 |

Busca em largura



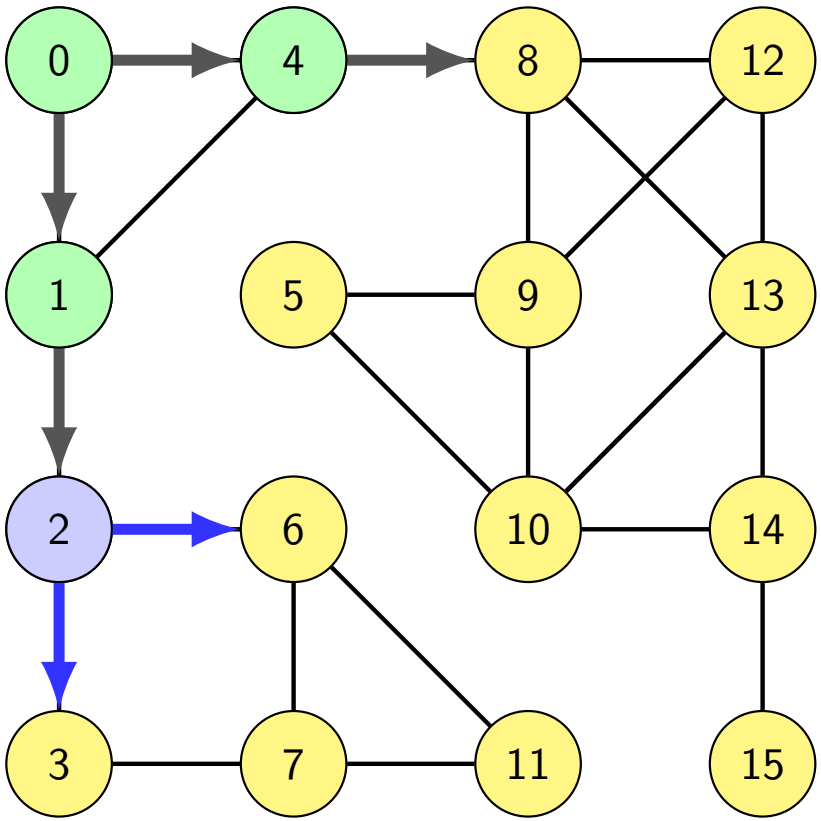
Fila | 8

Busca em largura



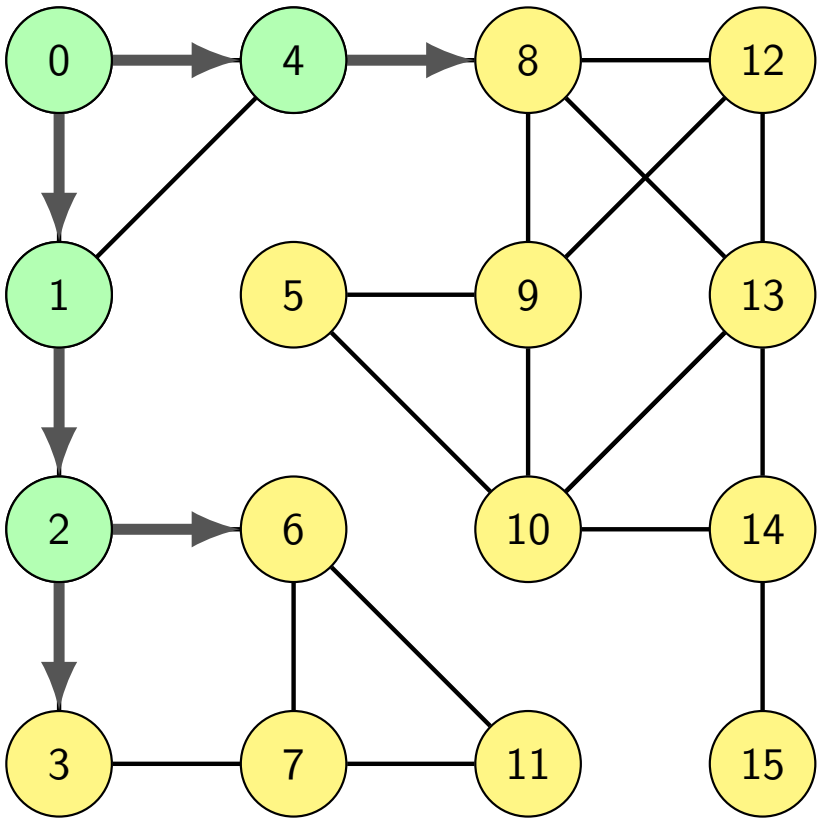
Fila | 8 | 3 |

Busca em largura



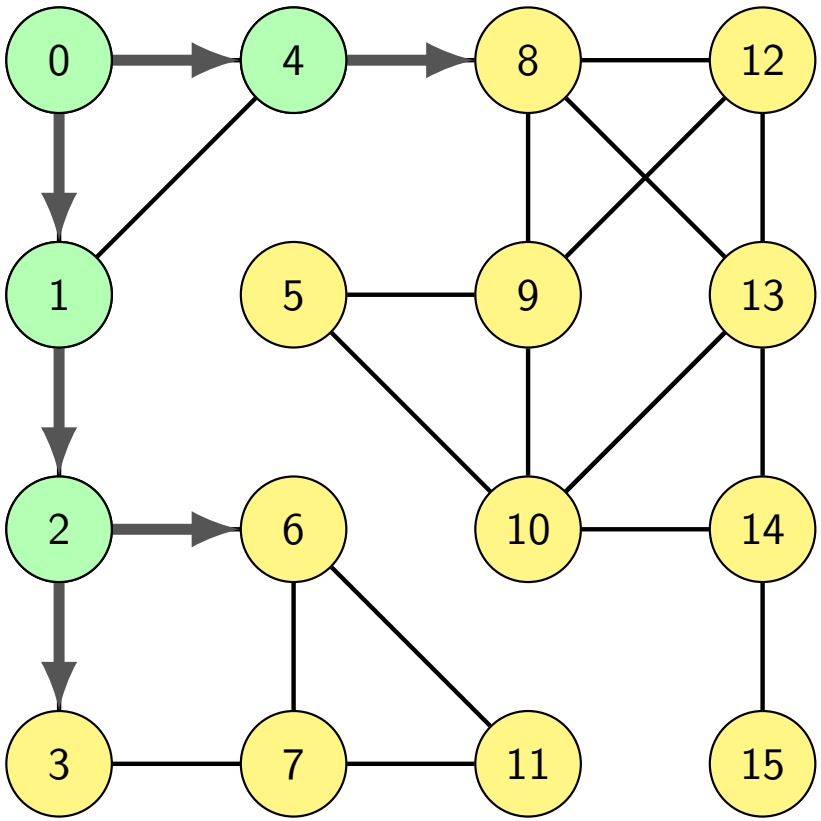
Fila | 8 | 3 | 6 |

Busca em largura



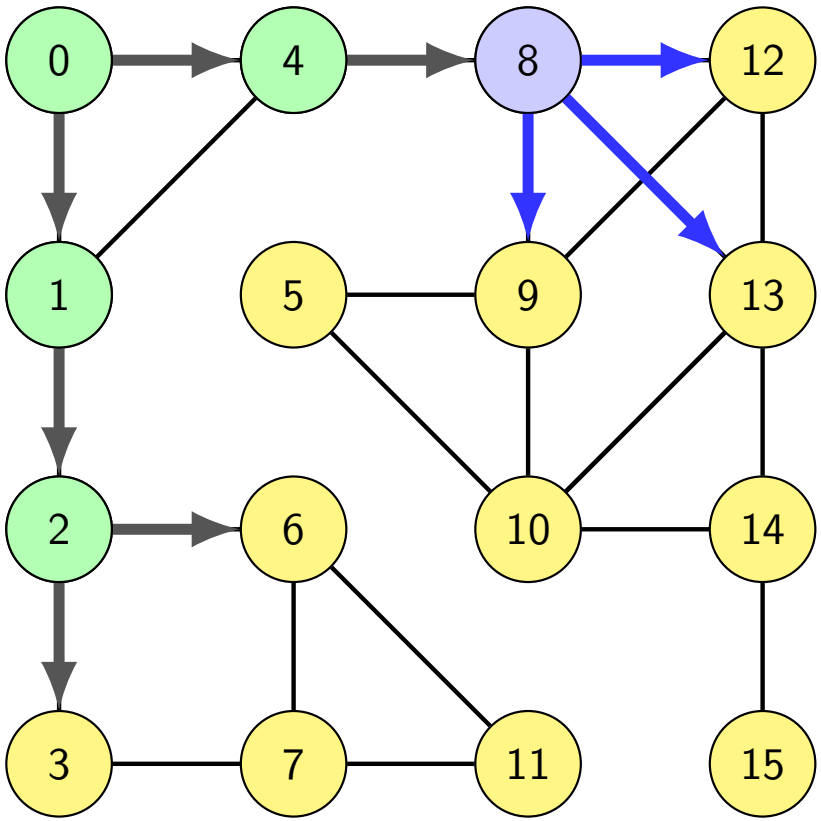
Fila | 8 | 3 | 6 |

Busca em largura



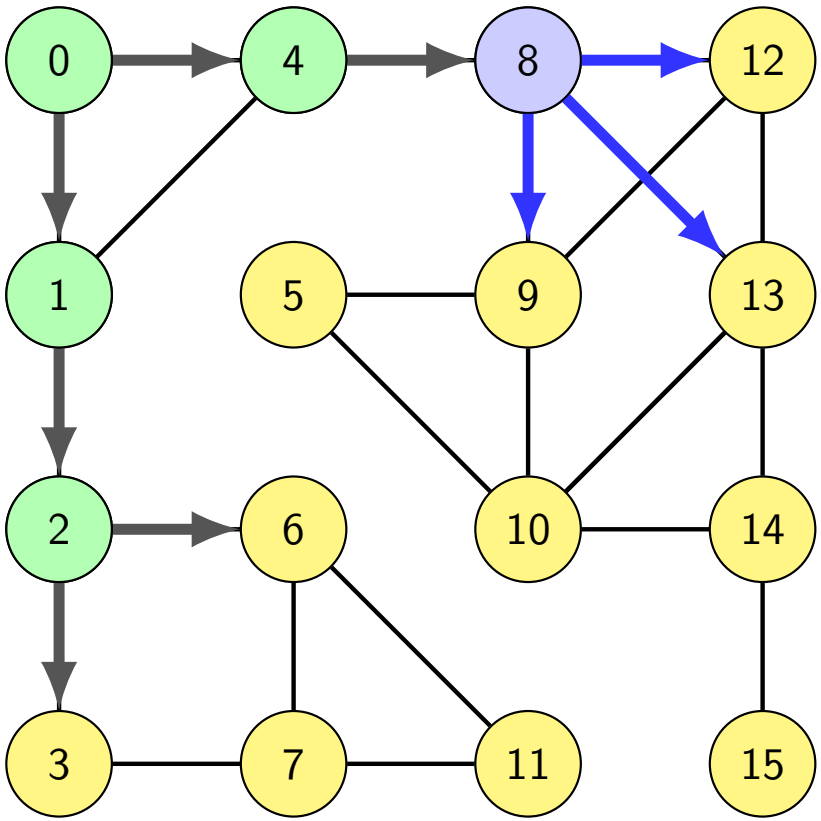
Fila | 8 | 3 | 6 |

Busca em largura



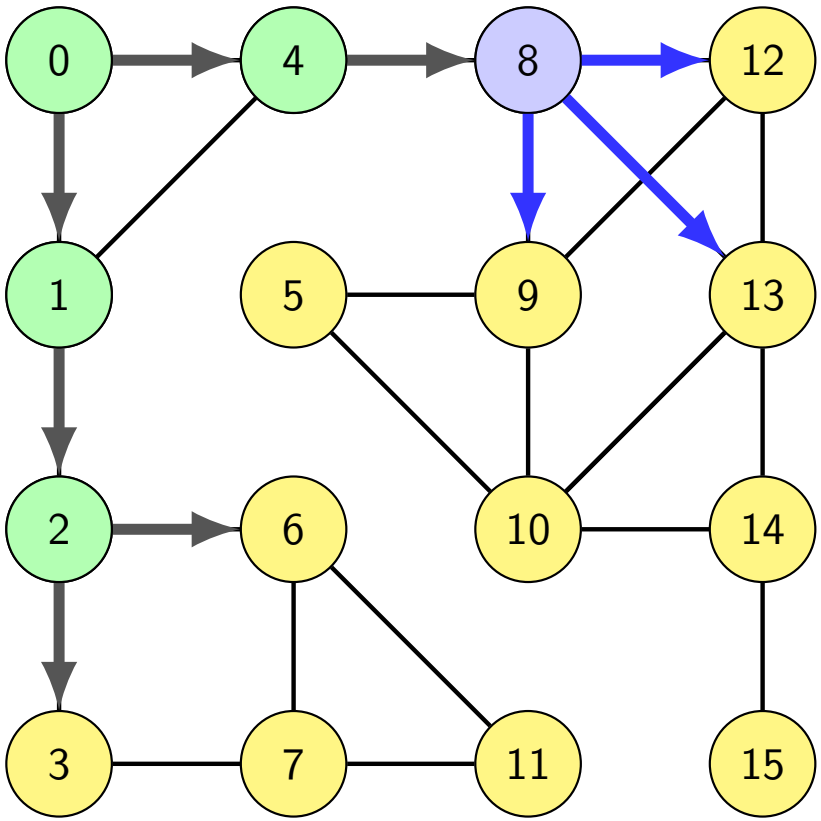
Fila | 3 | 6 |

Busca em largura



Fila | 3 | 6 | 9 |

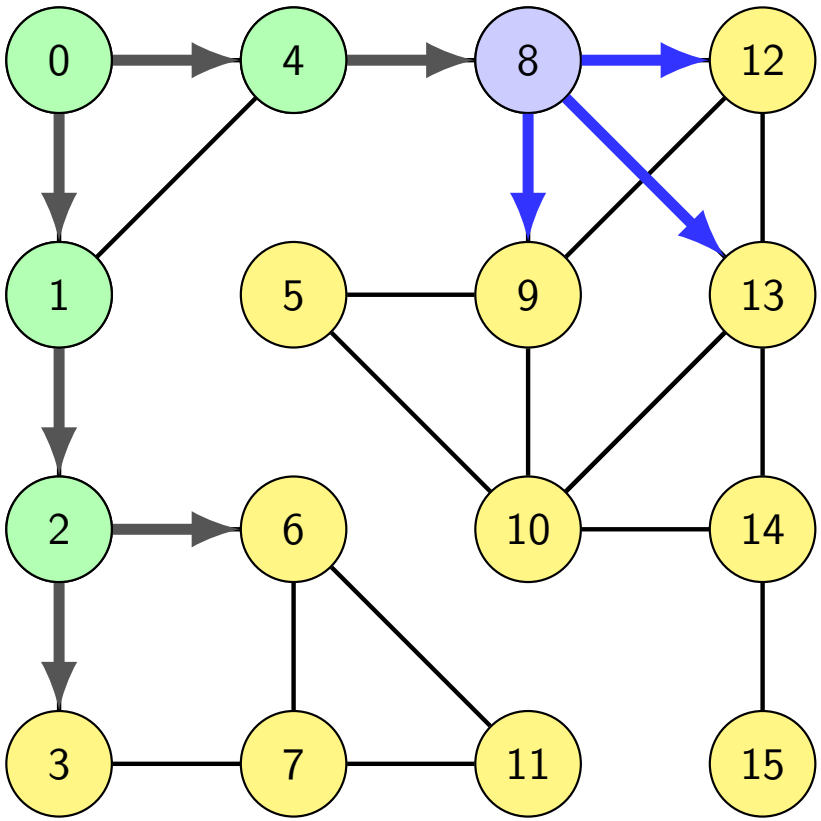
Busca em largura



Fila

3	6	9	12

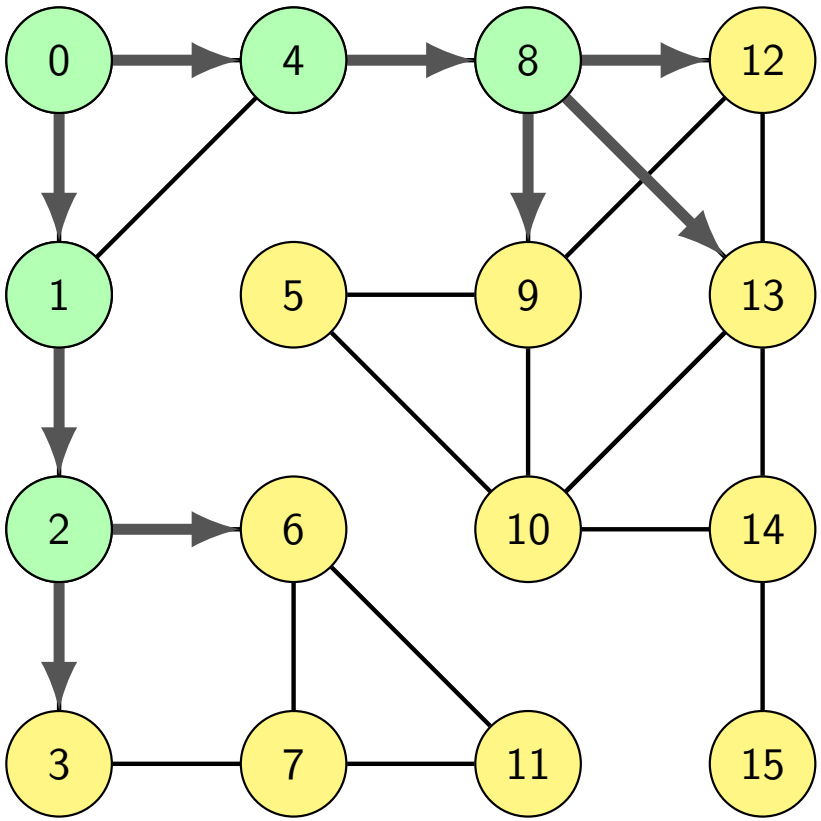
Busca em largura



Fila

3	6	9	12	13

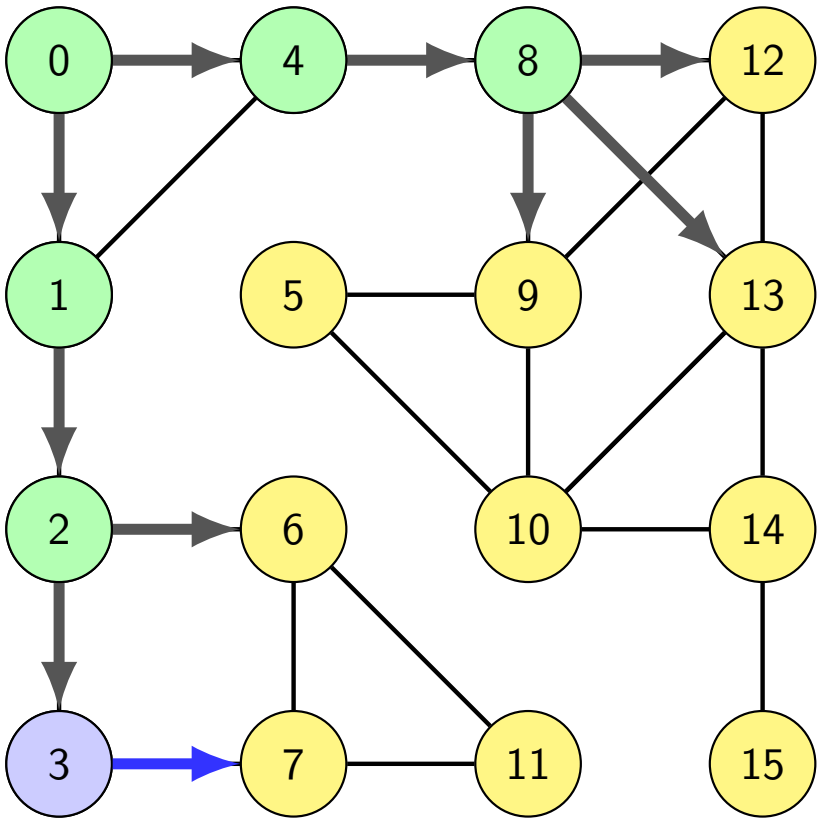
Busca em largura



Fila

3	6	9	12	13

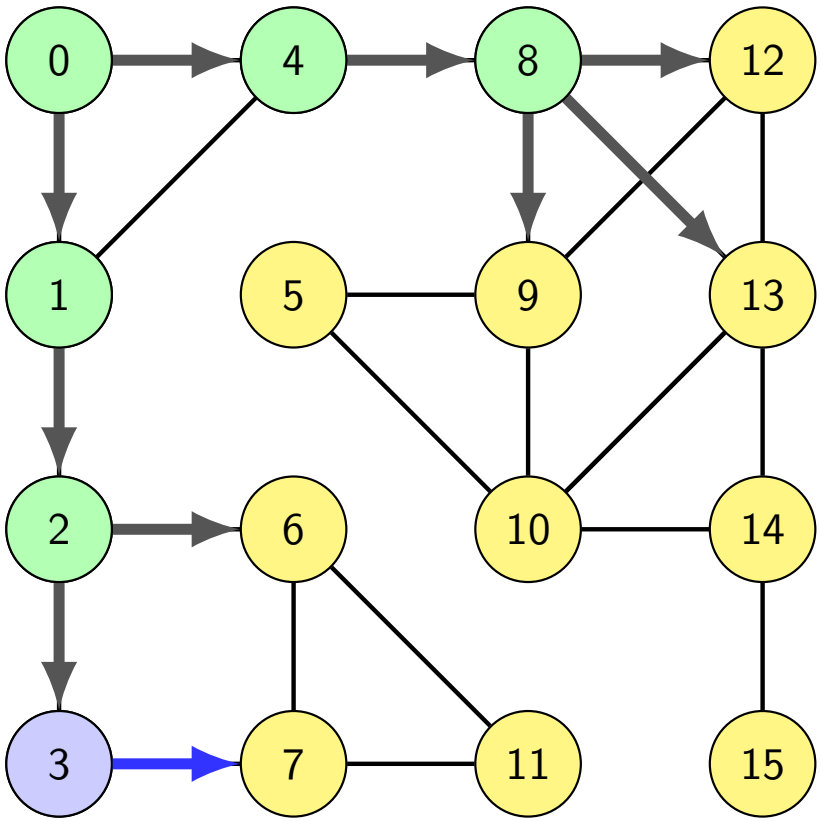
Busca em largura



Fila

6	9	12	13

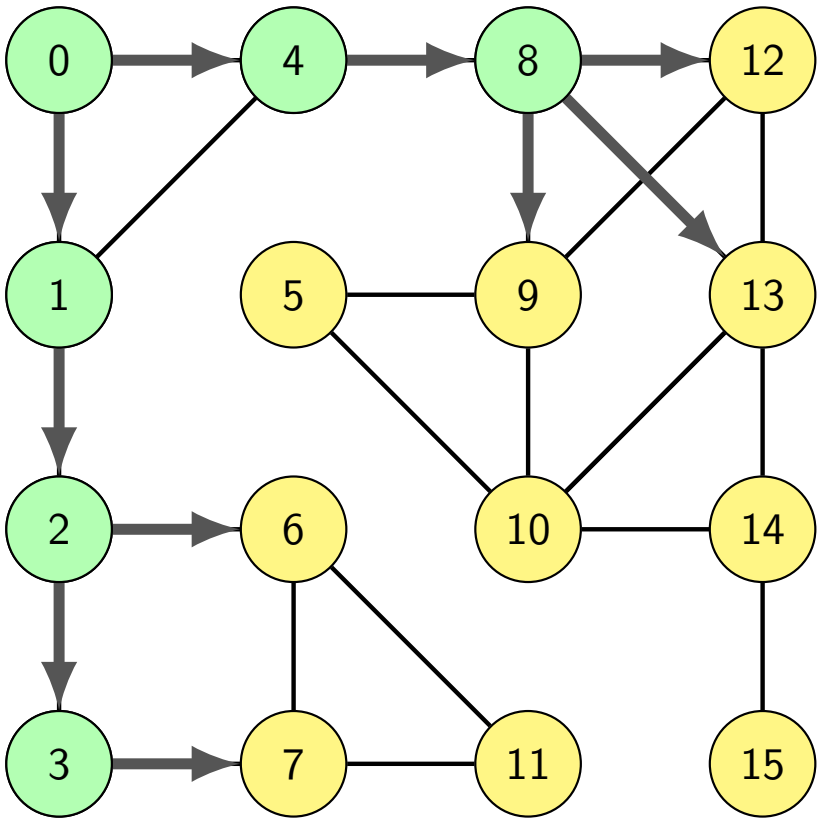
Busca em largura



Fila

6	9	12	13	7

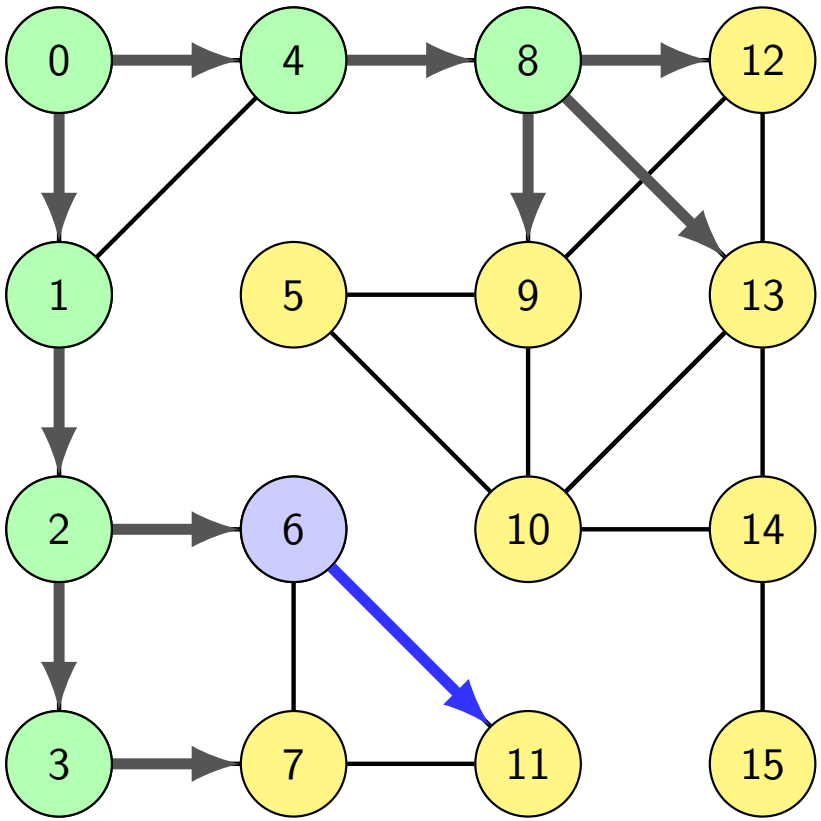
Busca em largura



Fila

6	9	12	13	7

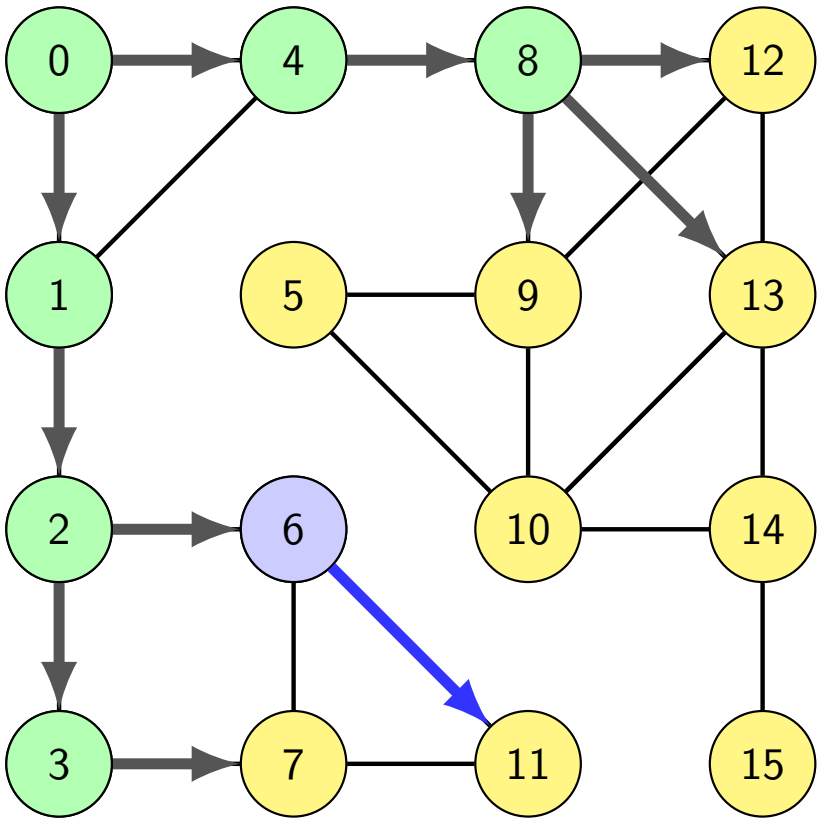
Busca em largura



Fila

9	12	13	7

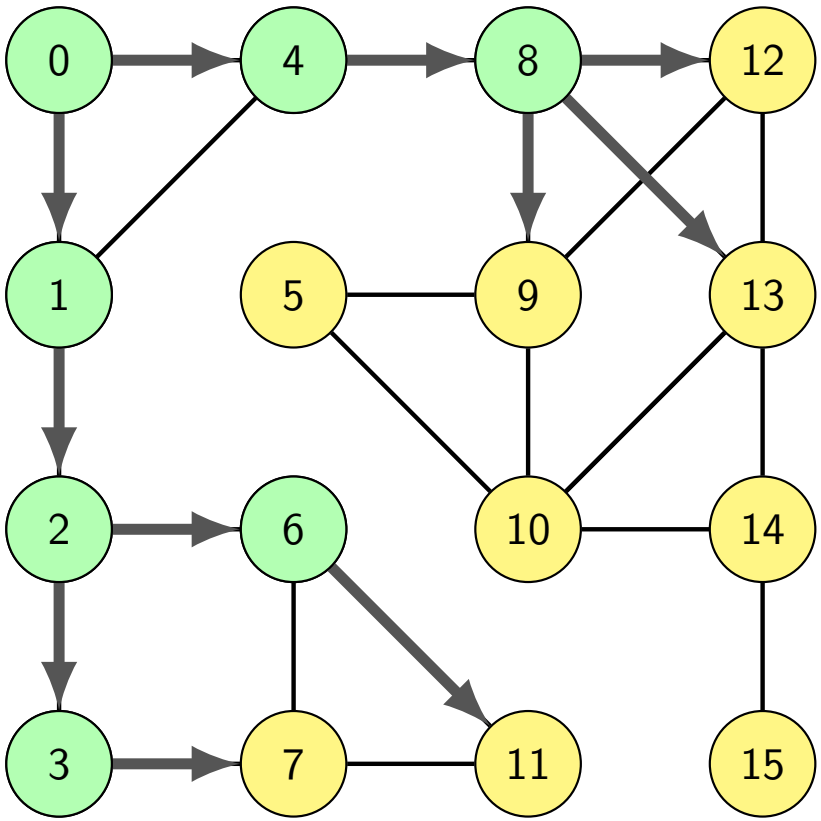
Busca em largura



Fila

9	12	13	7	11

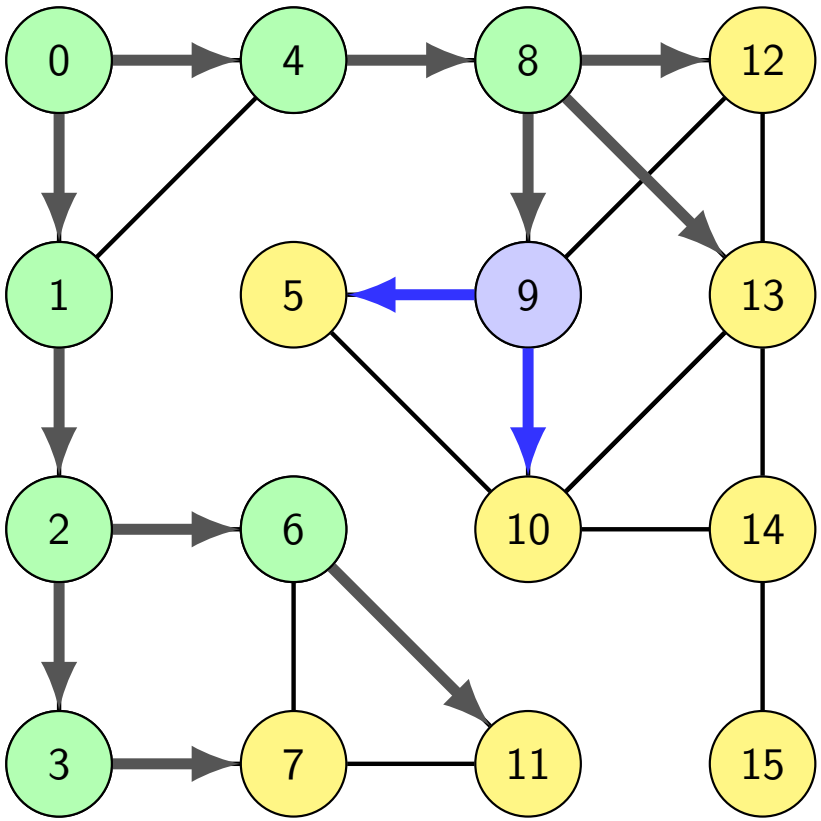
Busca em largura



Fila

9	12	13	7	11

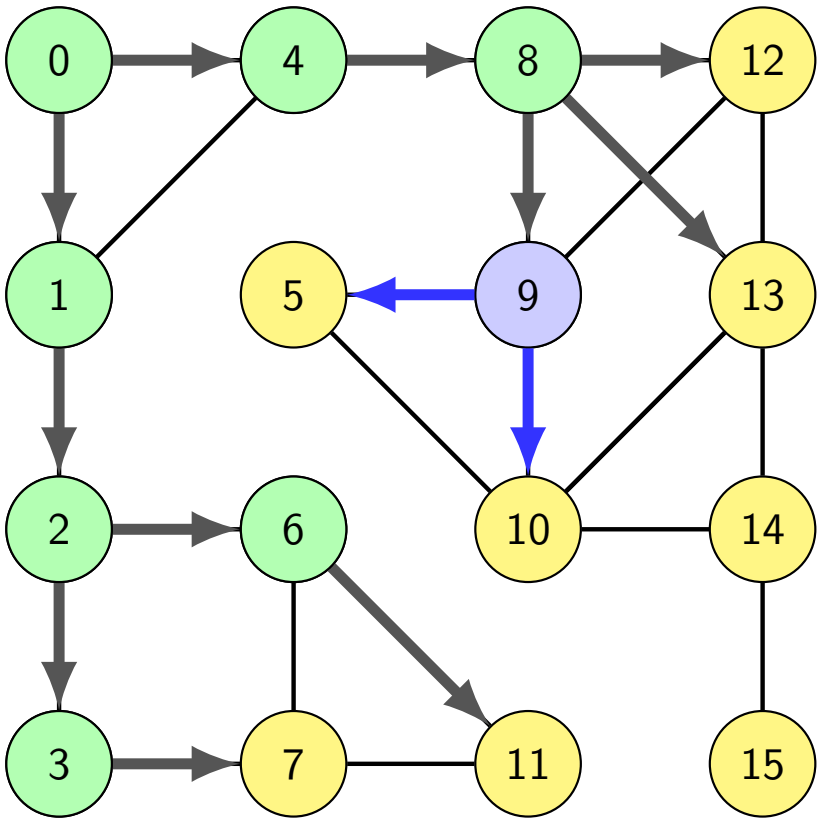
Busca em largura



Fila

12	13	7	11

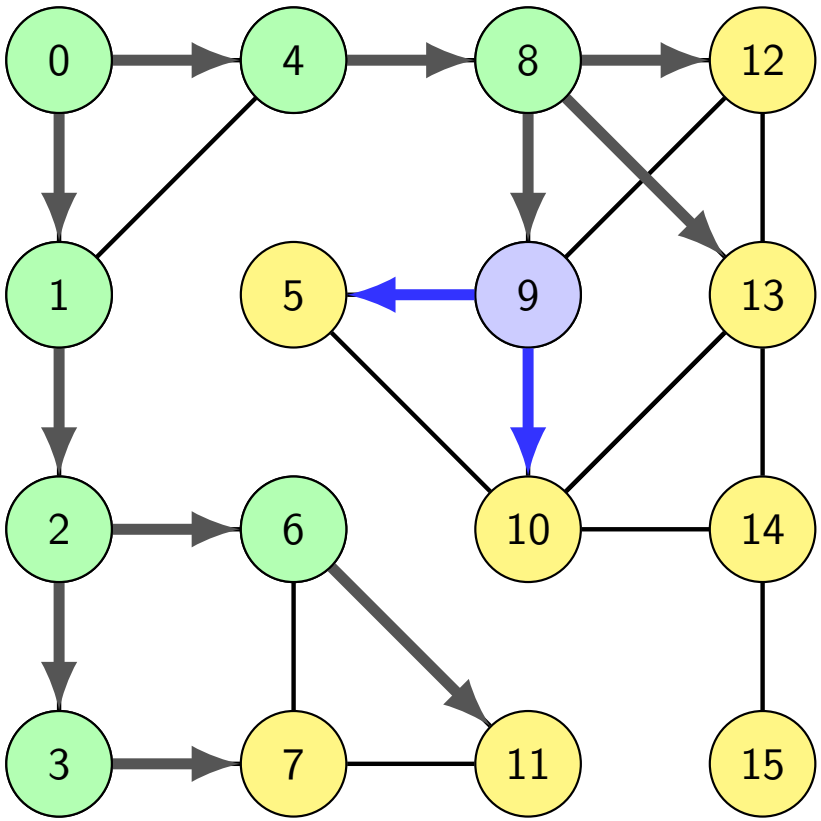
Busca em largura



Fila

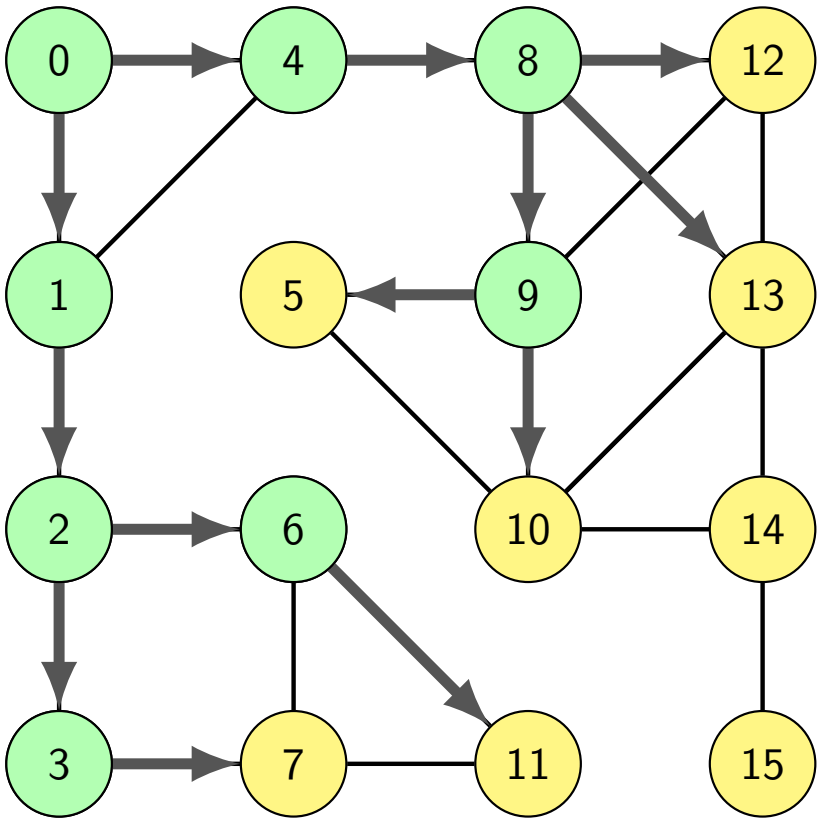
12	13	7	11	5

Busca em largura



Fila	12	13	7	11	5	10

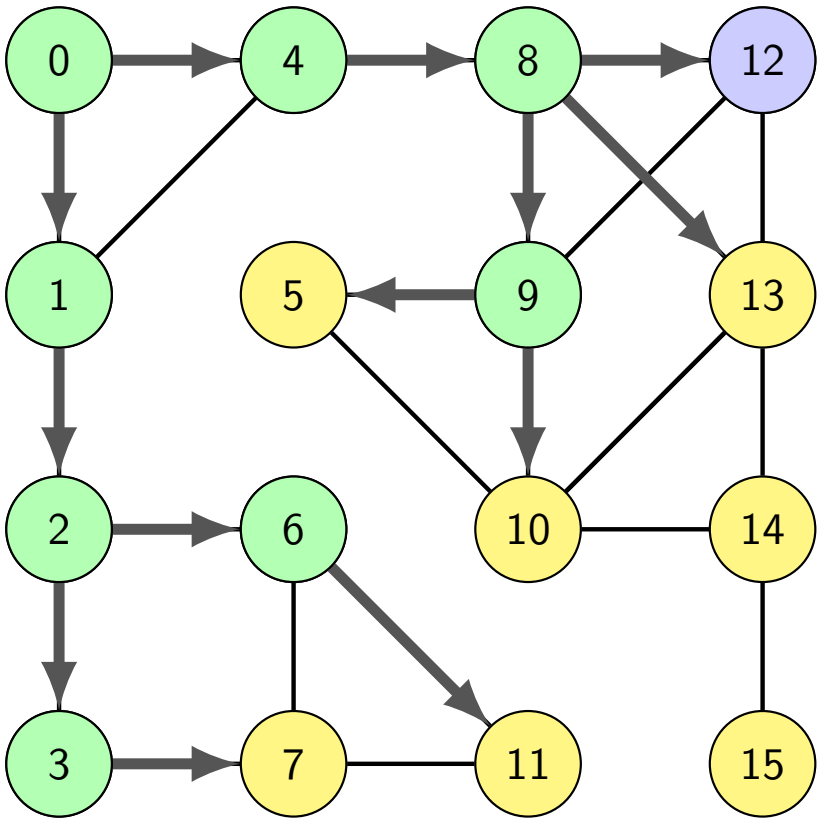
Busca em largura



Fila

12	13	7	11	5	10

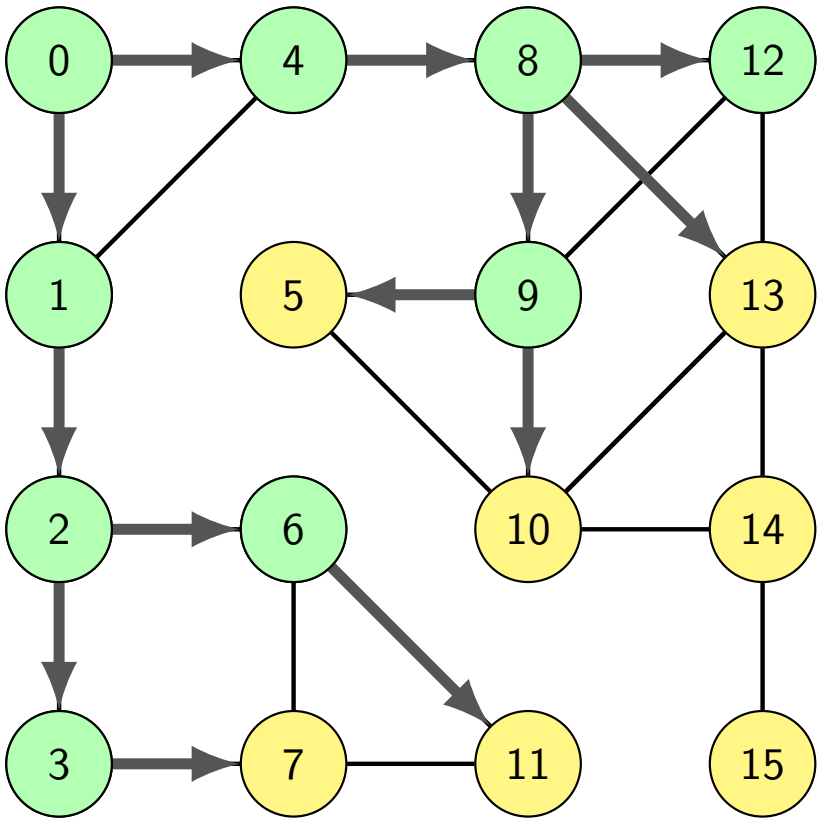
Busca em largura



Fila

13	7	11	5	10

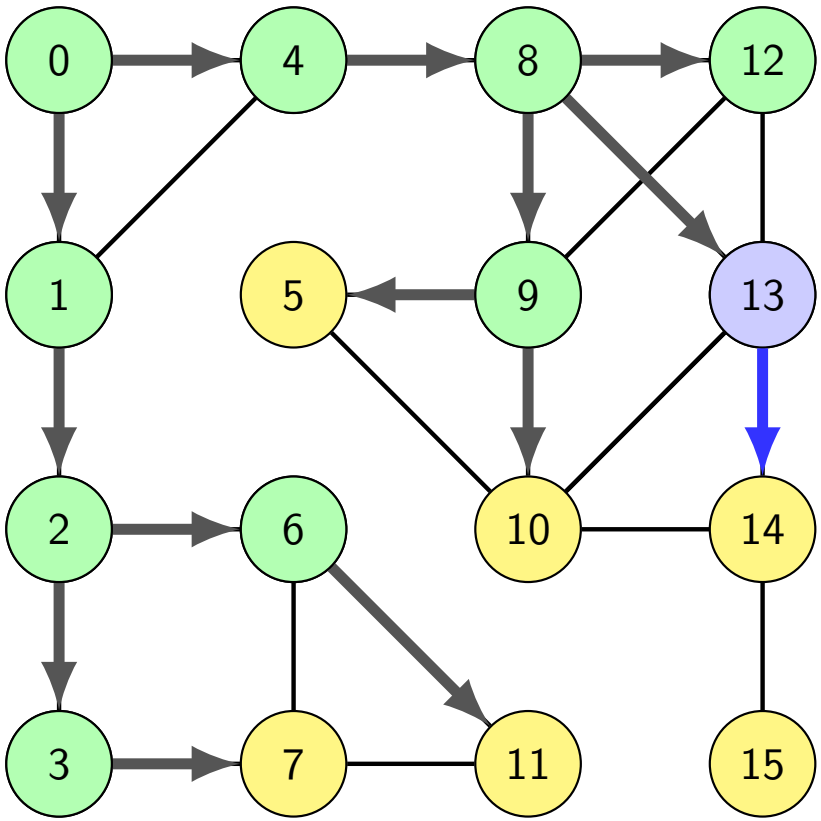
Busca em largura



Fila

13	7	11	5	10

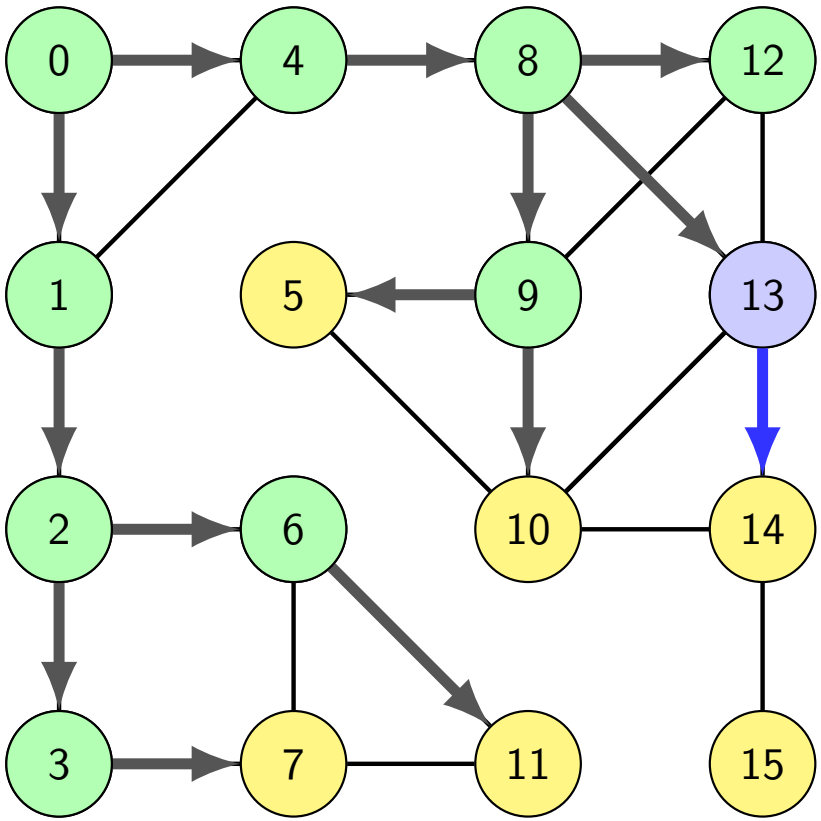
Busca em largura



Fila

7	11	5	10

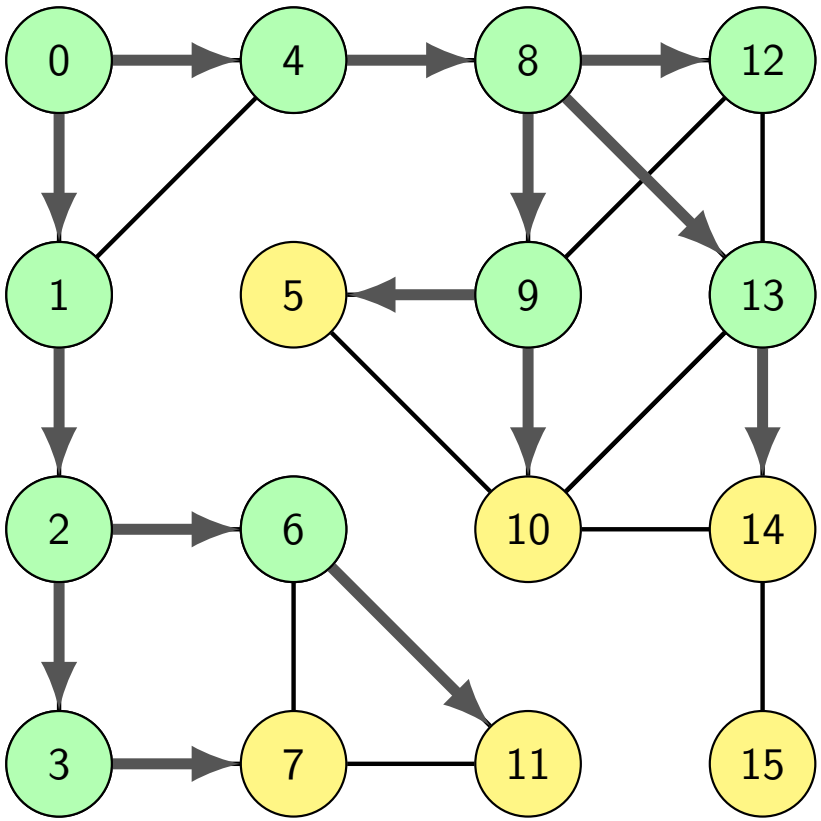
Busca em largura



Fila

7	11	5	10	14

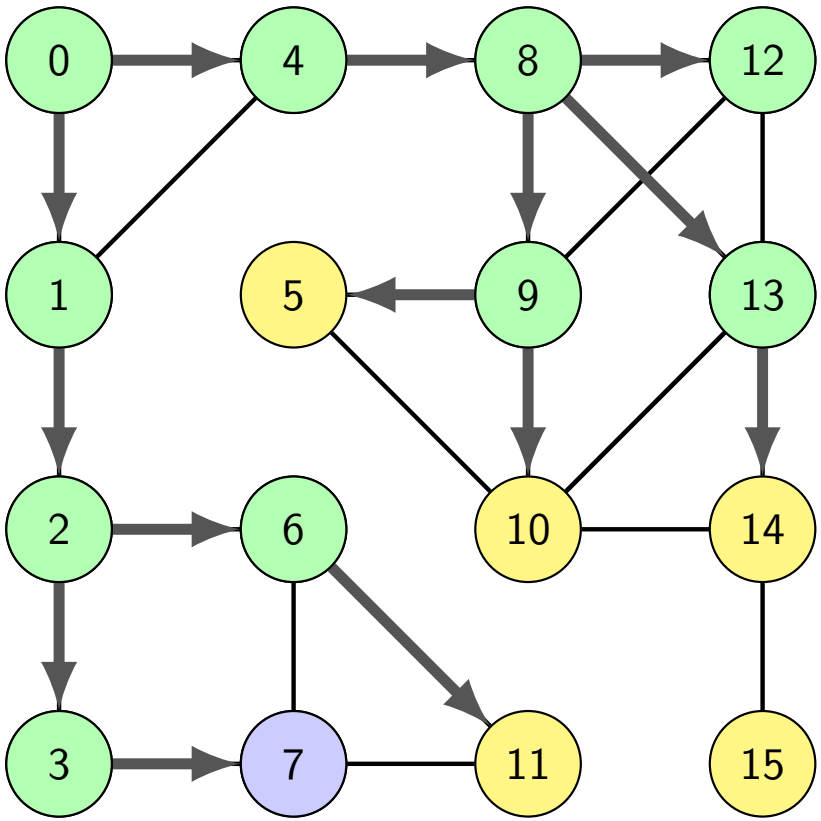
Busca em largura



Fila

7	11	5	10	14

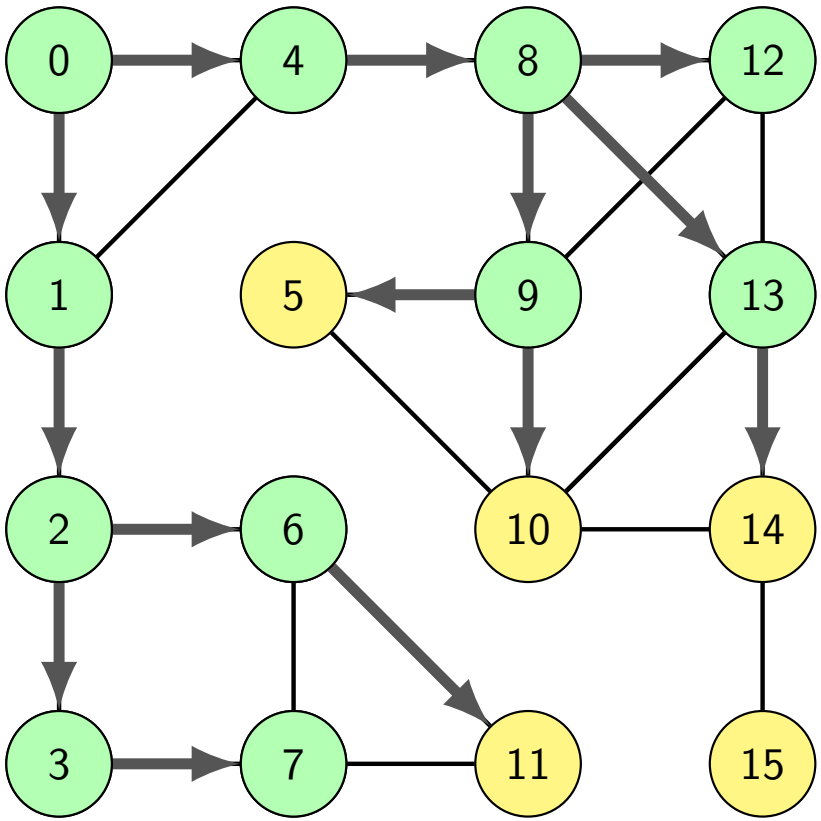
Busca em largura



Fila

11	5	10	14

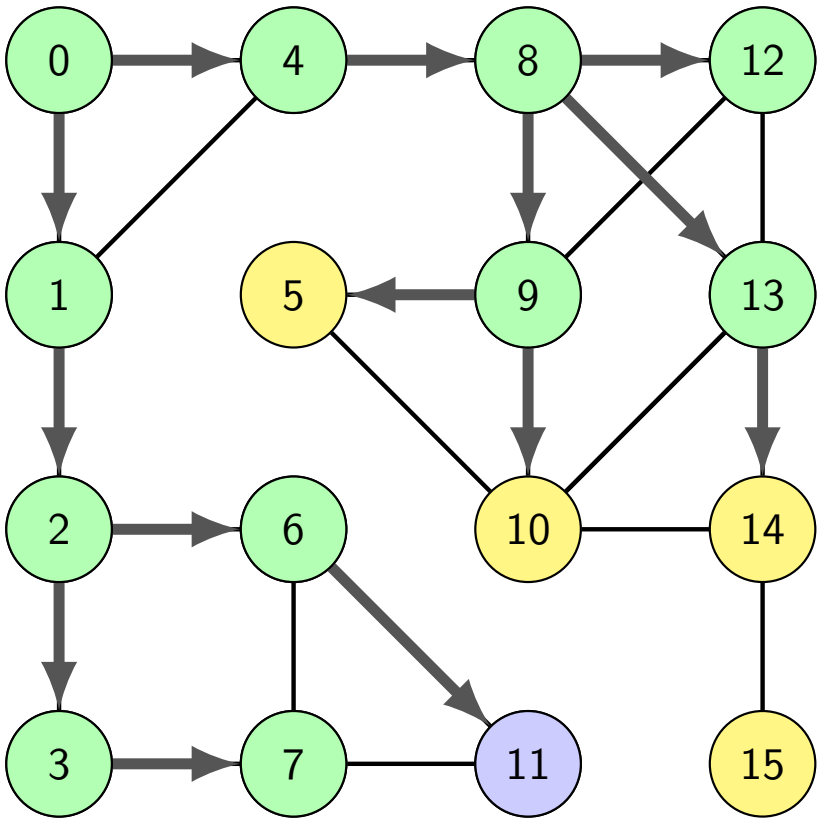
Busca em largura



Fila

11	5	10	14

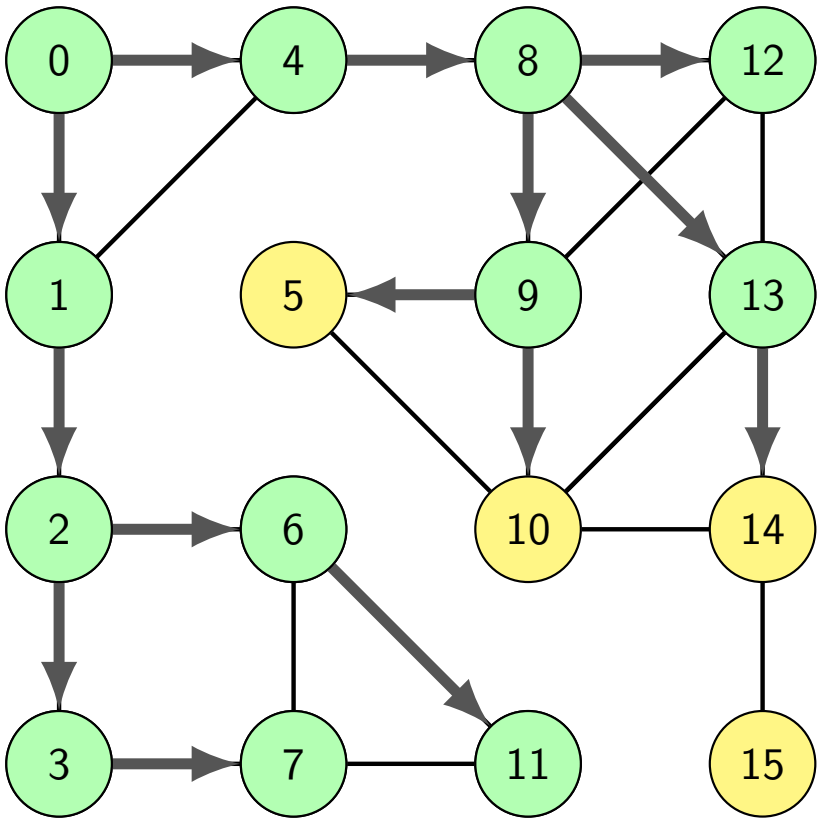
Busca em largura



Fila

5	10	14

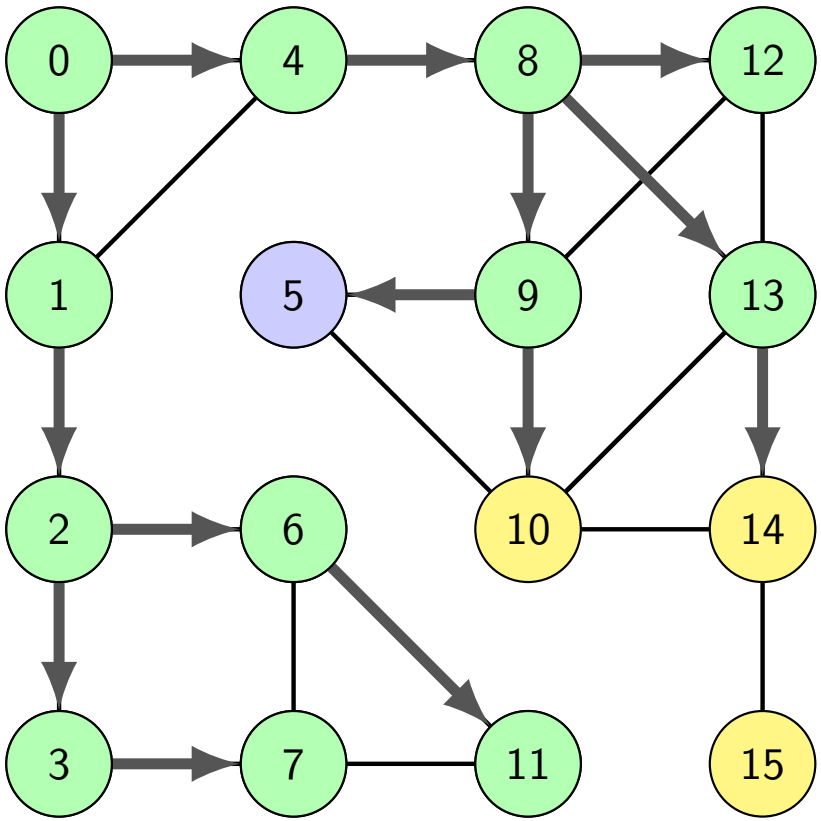
Busca em largura



Fila

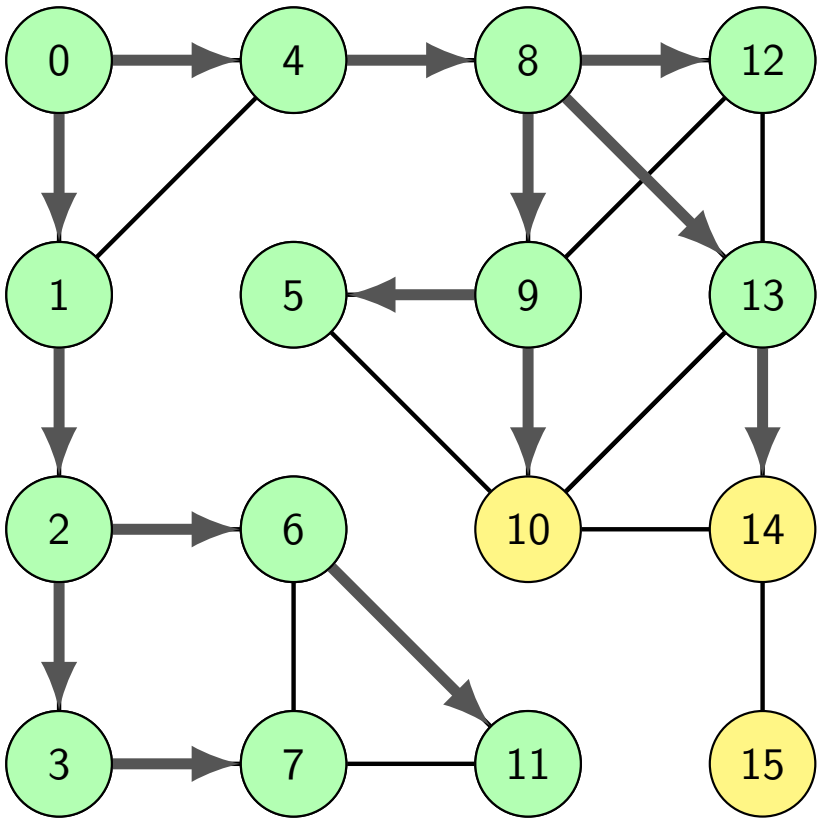
5	10	14

Busca em largura



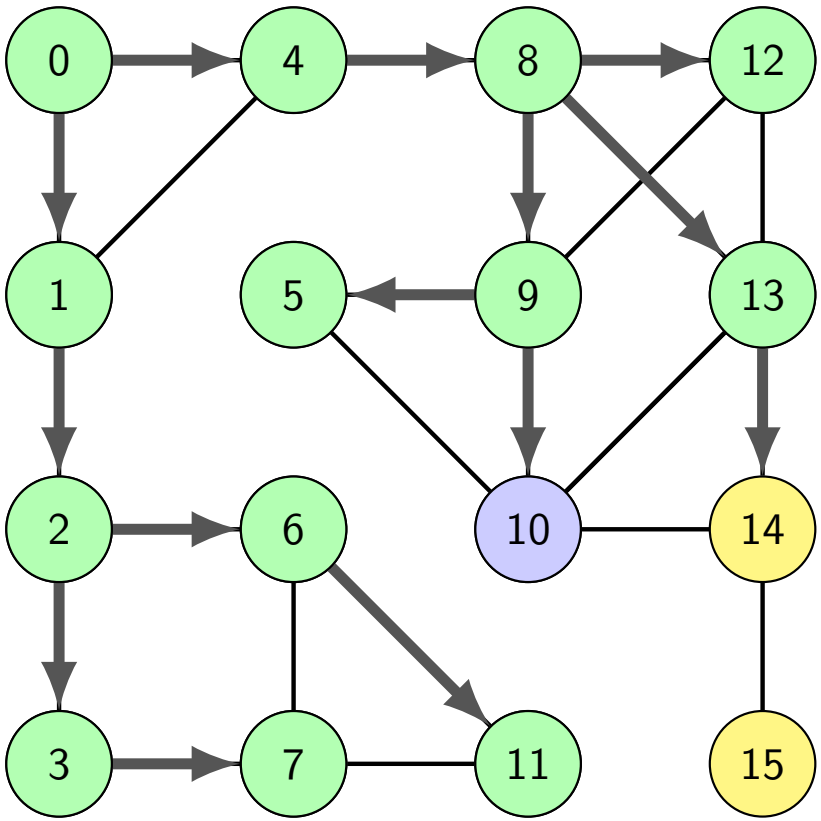
Fila | 10 | 14 |

Busca em largura



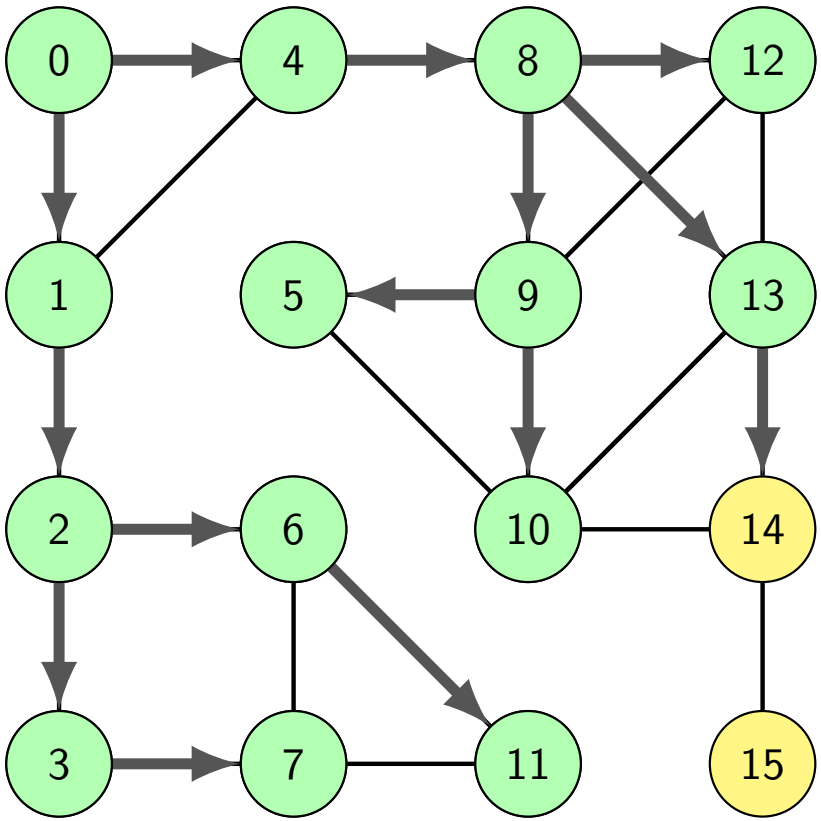
Fila | 10 | 14 |

Busca em largura



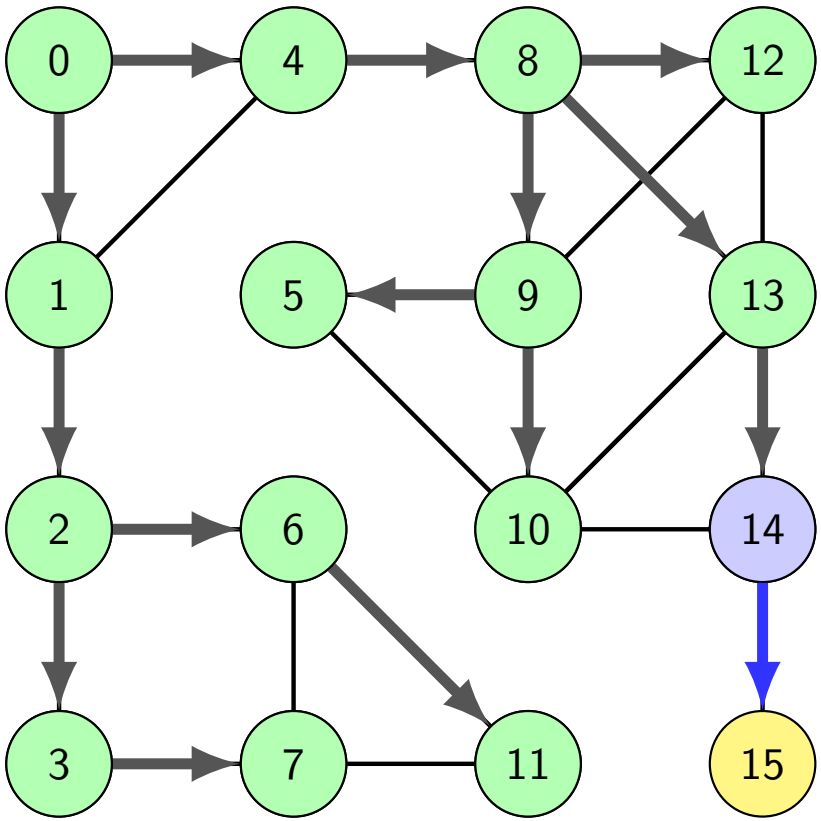
Fila | 14

Busca em largura



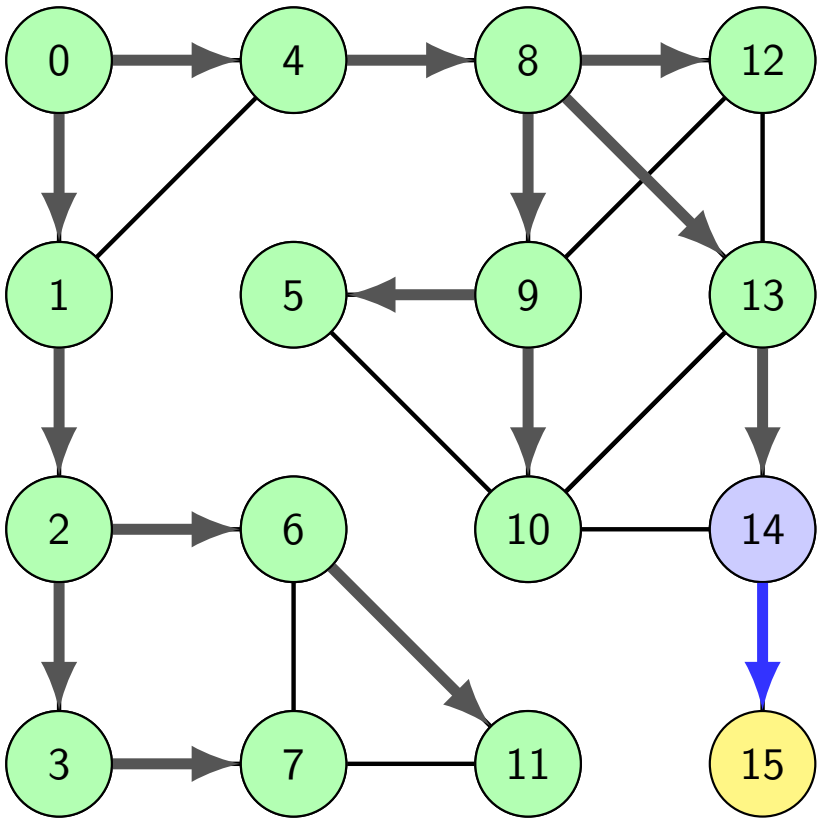
Fila | 14

Busca em largura



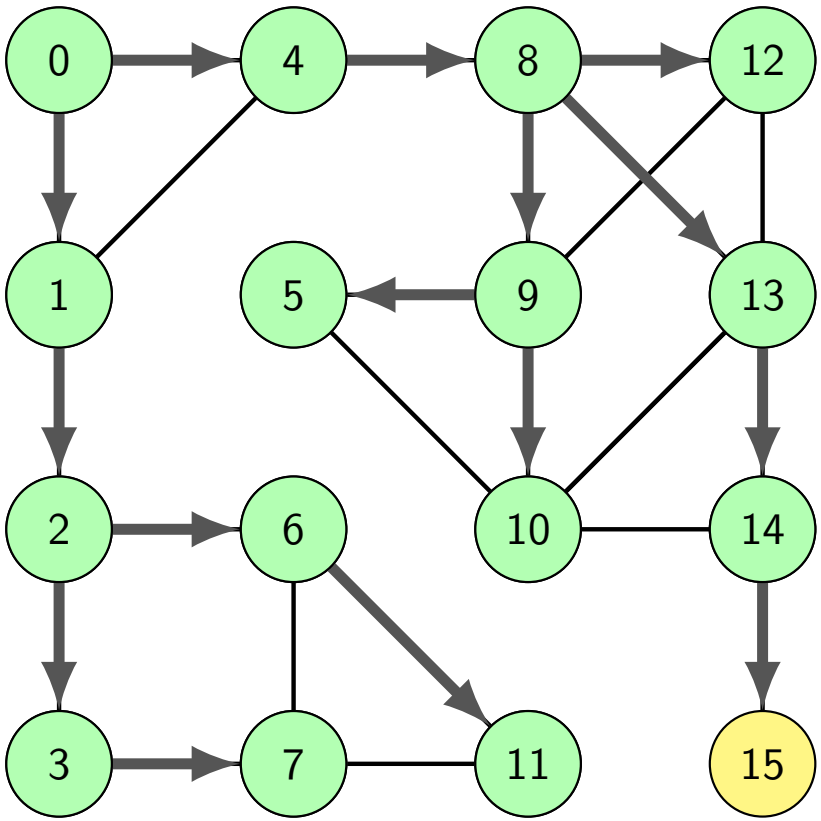
Fila

Busca em largura



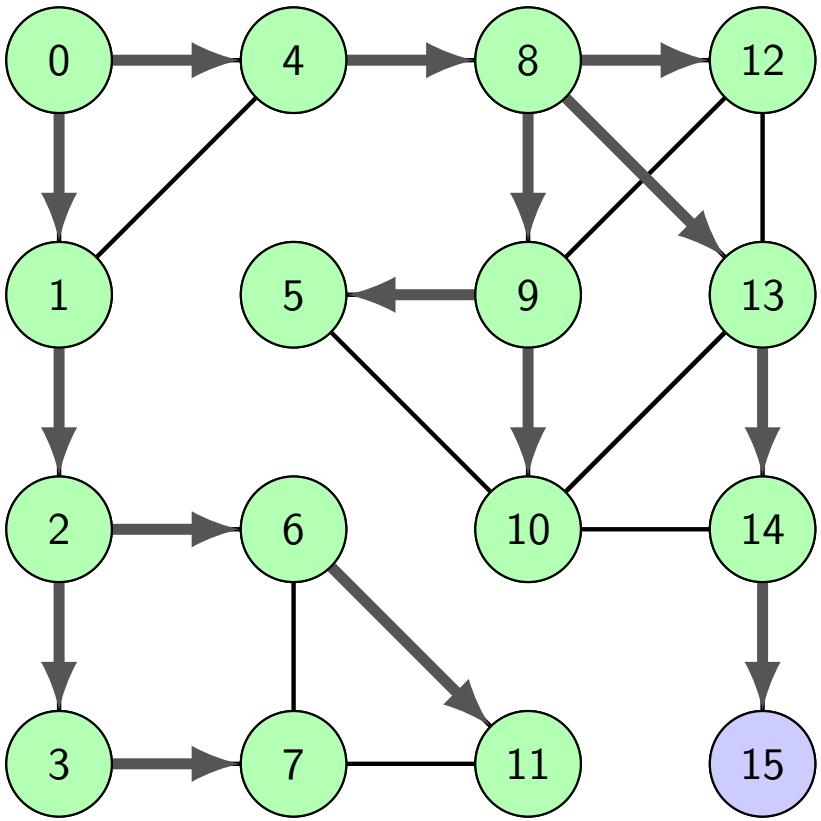
Fila | 15

Busca em largura



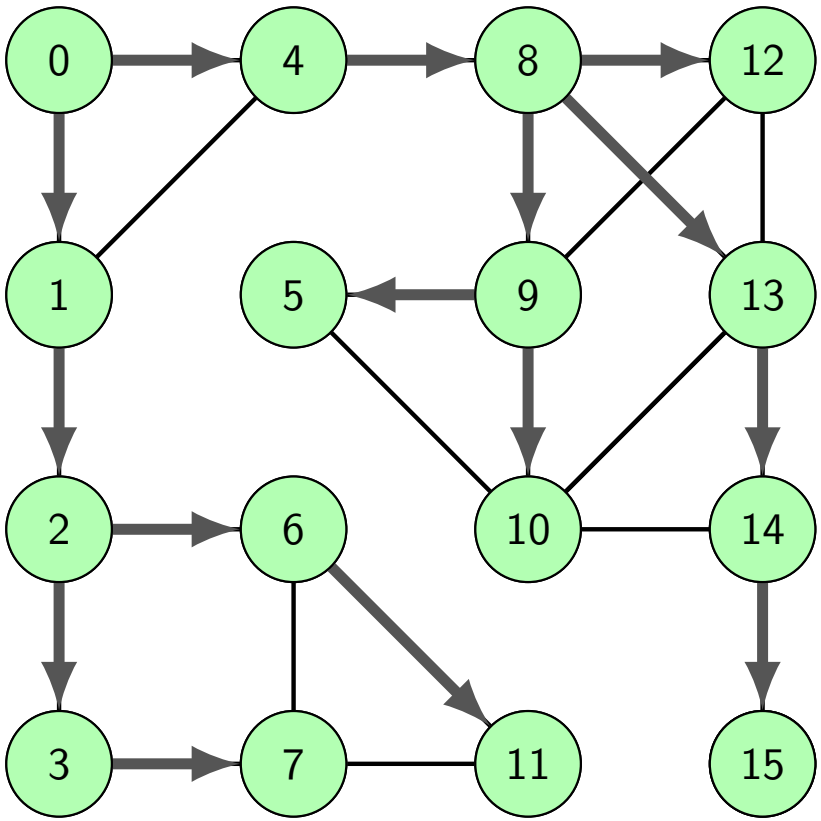
Fila | 15

Busca em largura



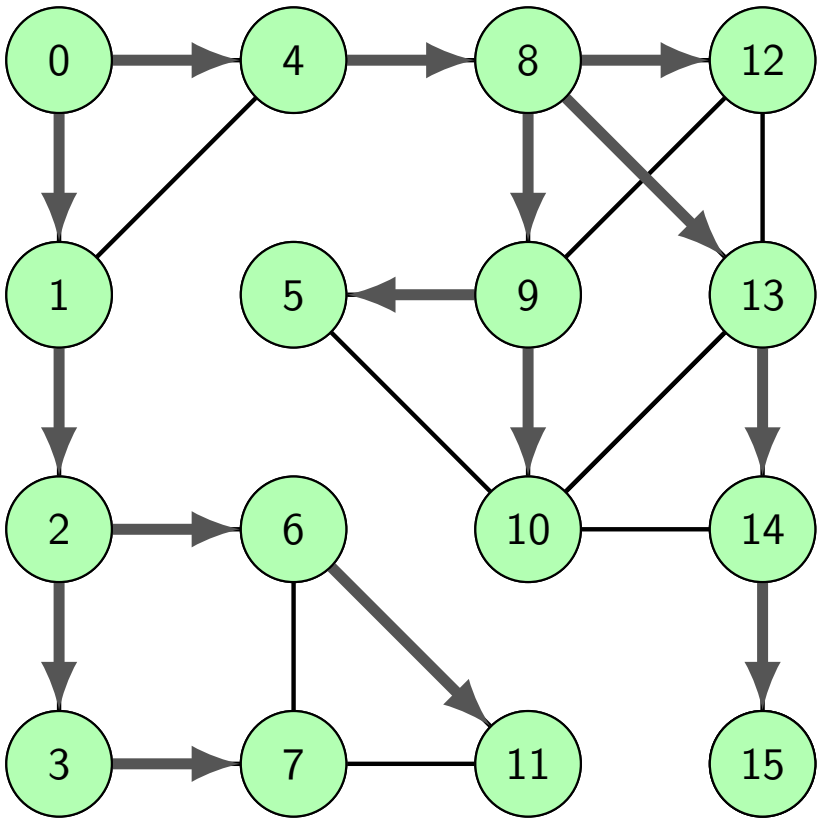
Fila

Busca em largura

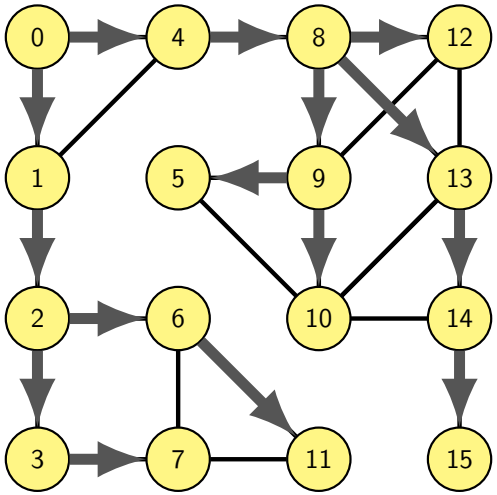


Fila

Busca em largura

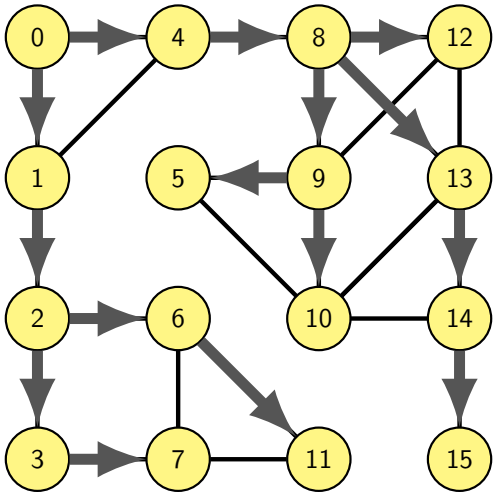


E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

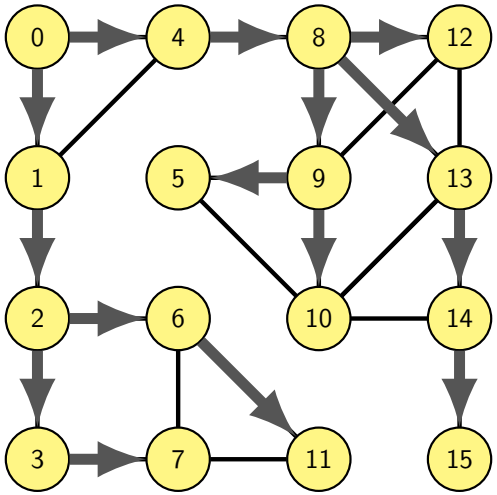
E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)

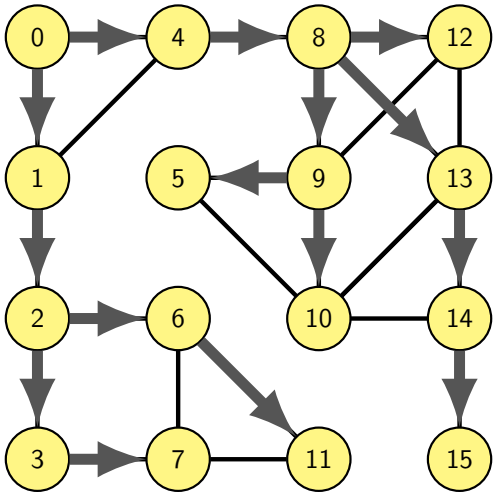
E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)
- Desenfileiramos um de seus vizinho

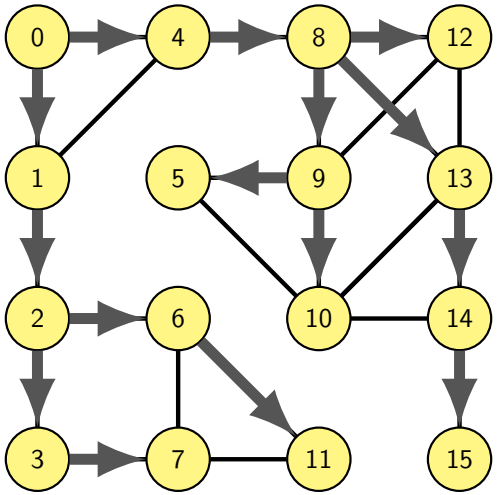
E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)
- Desenfileiramos um de seus vizinho
- E enfileiramos os **vizinhos** deste vértice

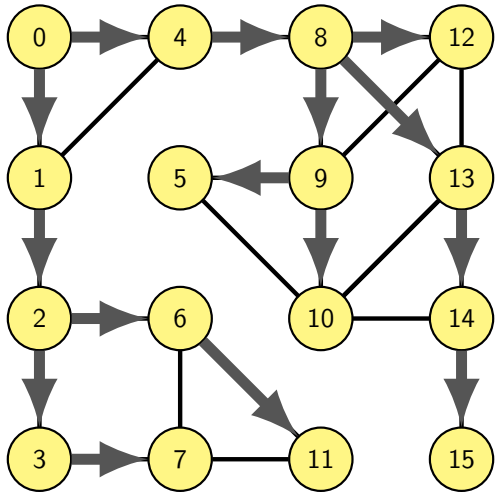
E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)
- Desenfileiramos um de seus vizinho
- E enfileiramos os **vizinhos** deste vértice
  - que estão a distância 2 de 0

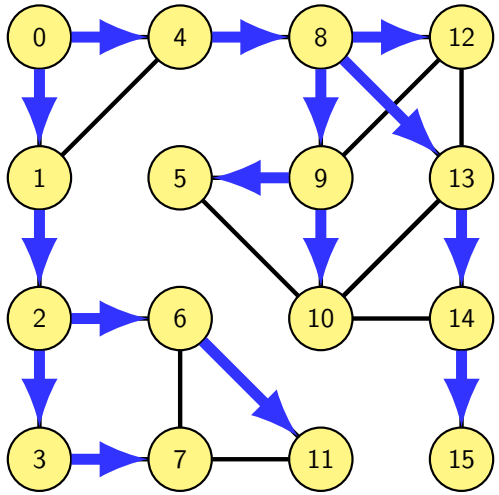
E se tivéssemos usando uma Fila?



Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)
- Desenfileiramos um de seus vizinho
- E enfileiramos os **vizinhos** deste vértice
  - que estão a distância 2 de 0
- Assim por diante...

E se tivéssemos usando uma Fila?

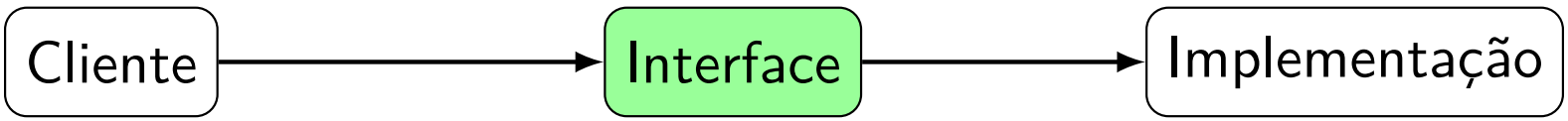


Usando uma fila, visitamos primeiro os vértices **mais próximos**

- Enfileiramos os **vizinhos** de 0 (que estão a distância 1)
- Desenfileiramos um de seus vizinho
- E enfileiramos os **vizinhos** deste vértice
  - que estão a distância 2 de 0
- Assim por diante...

A árvore nos dá um caminho mínimo entre raiz e vértice

Grafo - Busca em largura



grafo.h

```
4  #include "fila.h"
```

```
26 int existe_caminho_v2(Grafo *p, int u, int v);//busca em largura
```

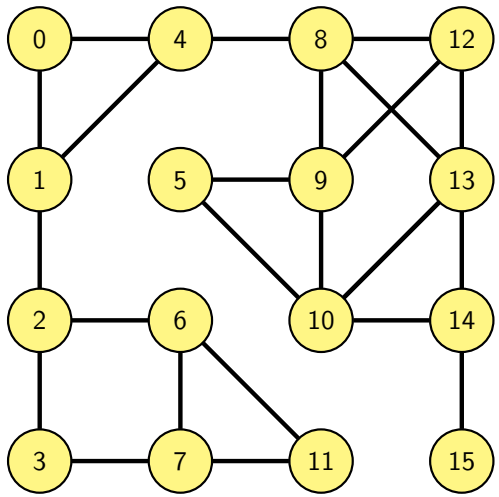
Grafo - Busca em largura

grafo.c

```
126 int existe_caminho_v2(Grafo* p, int u, int v) {
127   int *visitado = (int*) malloc (p->n * sizeof(int));
128   int i;
129   for (i = 0; i < p->n; i++)
130     visitado[i] = 0;
131   int encontrou = busca_em_largura(p, visitado, u, v);
132   free(visitado);
133   return encontrou;
134 }
```

Implementação da Busca em Largura

```
102 int busca_em_largura(Grafo *p, int *visitado, int u, int v) {//ex. u=0, v=9
103   int encontrou = 0;
104   Fila *F = fila_criar();
105   fila_adicionar(&F, u);
106   visitado[u] = 1;
107   while(fila_tamanho(F)>0) {
108     int i = fila_topo(F);
109     fila_remover(&F);
110     int w;
111     for (w = 0; w < p->n; w++){
112       if (tem_aresta(p, i, w) && !visitado[w]) {
113         if(w == v){
114           encontrou = 1;
115           break;
116         }
117         visitado[w] = 1; //evita repetição na fila
118         fila_adicionar(&F, w);
119       }
120     }
121   }
122   fila_destruir(&F);
123 return encontrou;
124 }
```



Cliente

exemplo2.c

```
5  int main() {
6    int n, m, i, u, v;
7    scanf("%d %d", &n, &m);
8    Grafo *G = criar_grafo(n);
9    for (i = 0; i < m; i++) {
10     scanf("%d %d", &u, &v);
11     inserir_aresta(G, u, v);
12   }
13   imprimir_arestas(G);
14   for (u = 0; u < n; u++)
15     for (v = u+1; v < n; v++)
16       if(existe_caminho_v2(G, u, v))
17         printf("%d <-> %d\n", u, v);
18   return 0;
19 }
```

Makefile

Vamos usar o Makefile para compilar:

```
LIB = grafo_matriz.o \
      lista_circular.o \
      fila.o
```

```
exemplo2: exemplo2.c $(LIB)
  gcc $^ -o $@
```

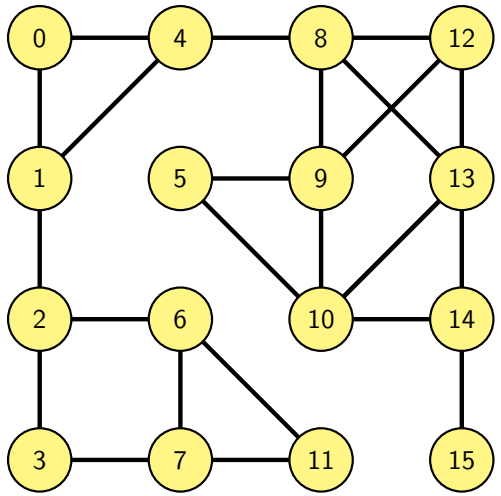
Vamos executar:

```
$ ./exemplo2 < teste2.in
{0,1}
{1,2}
{3,5}
{4,5}
0 <-> 1
0 <-> 2
1 <-> 2
3 <-> 4
3 <-> 5
4 <-> 5
```

Roteiro

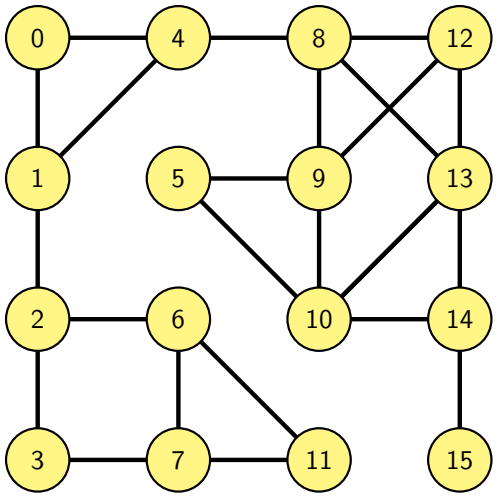
Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?

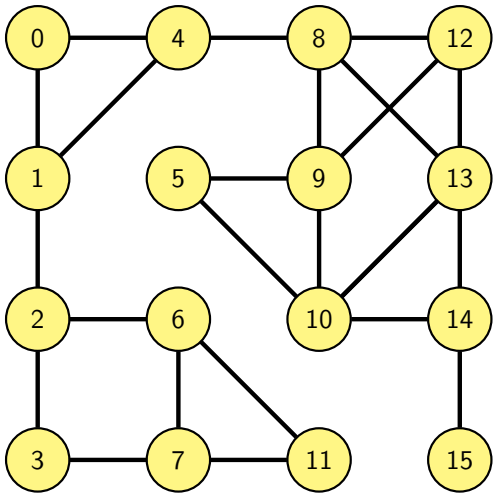


Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

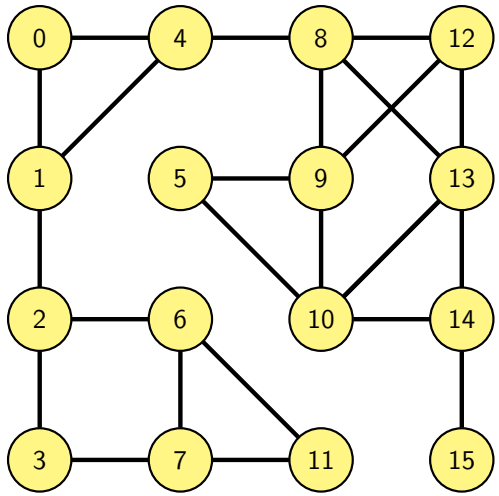
- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

- E empilha/enfileira seus vizinhos não visitados

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

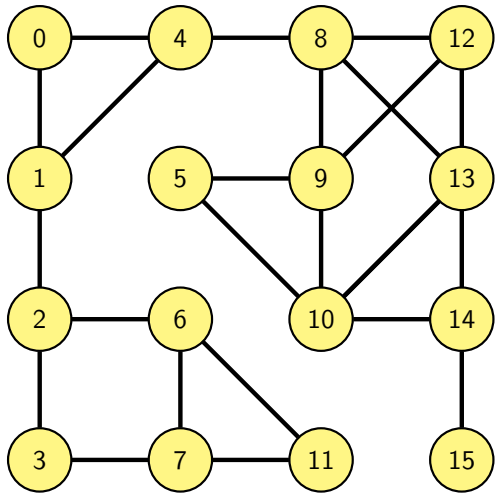
- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

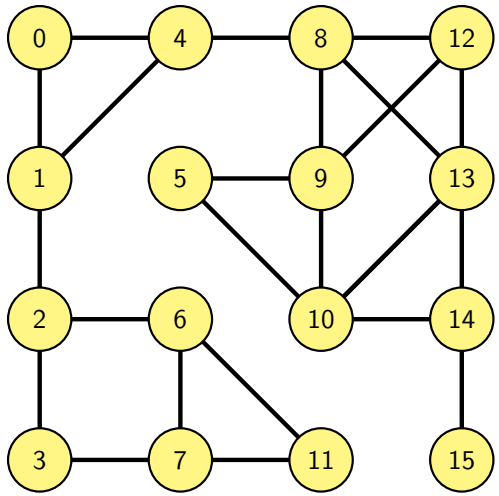
- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

E se usarmos Listas de Adjacência?

- Cada aresta é analisada **no máximo** duas vezes

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

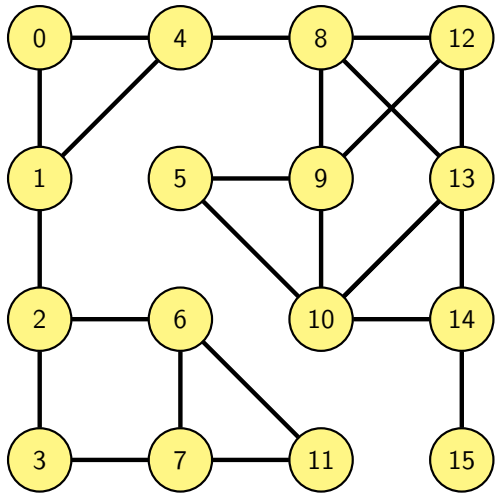
- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

E se usarmos Listas de Adjacência?

- Cada aresta é analisada **no máximo** duas vezes
- Gastamos tempo $O(\max\{|V|, |E|\}) = O(|V| + |E|)$

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

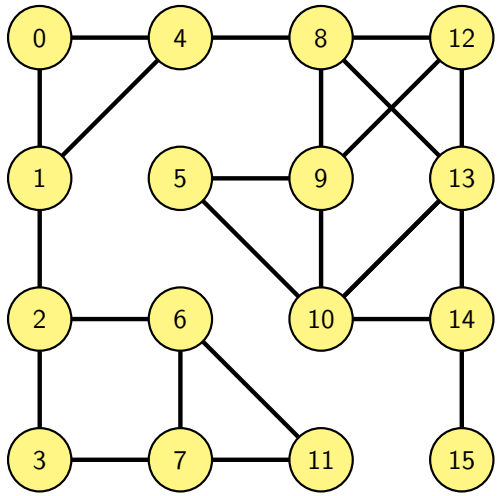
E se usarmos Listas de Adjacência?

- Cada aresta é analisada **no máximo** duas vezes
- Gastamos tempo $O(\max\{|V|, |E|\}) = O(|V| + |E|)$

Linear no **tamanho do grafo** (tamanho da representação)

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

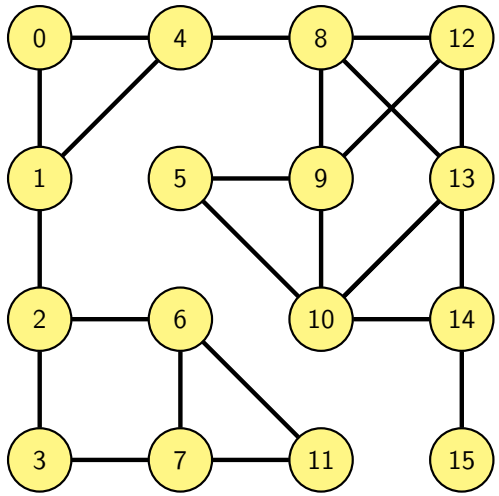
E se usarmos Listas de Adjacência?

- Cada aresta é analisada **no máximo** duas vezes
- Gastamos tempo $O(\max\{|V|, |E|\}) = O(|V| + |E|)$

Se G for denso, $O(|V| + |E|) = O(|V|^2)$

Tempo para fazer a busca

Custo computacional para fazer uma busca em profundidade ou em largura em um grafo com?



Suponha que **inserir** e **remover** na

- Pilha/Fila leva $O(1)$
- Podemos usar vetores ou listas ligadas

A busca percorre todos os vértices

- E empilha/enfileira seus vizinhos não visitados
- Se usarmos uma Matriz de Adjacências, leva $O(|V|^2)$

E se usarmos Listas de Adjacência?

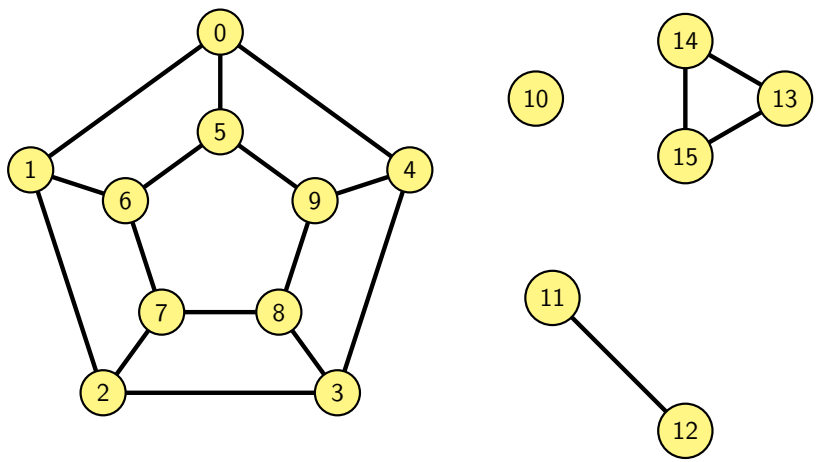
- Cada aresta é analisada **no máximo** duas vezes
- Gastamos tempo $O(\max\{|V|, |E|\}) = O(|V| + |E|)$

Se G for esparso, $O(|V| + k \cdot |V|) = O(|V|)$, k é o grau médio. (melhor **listas de ajd.**)

Roteiro

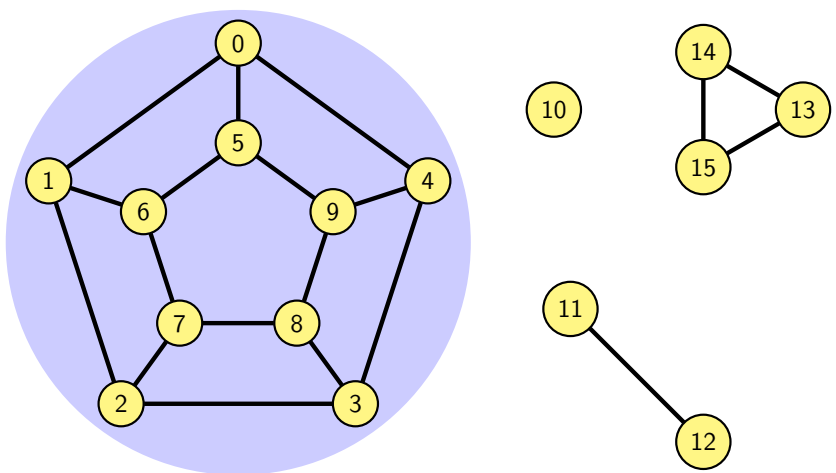
Como encontrar as componentes conexas?

```
138 void visita_recursiva(Grafo *p, int *C, int comp, int u) {
139   C[u] = comp;
140   int w;
141   for (w = 0; w < p->n; w++){
142     if (tem_aresta(p, u, w) && C[w] == -1)
143       visita_recursiva(p, C, comp, w);//busca em profundidade
144   }
145 }
146
147 int* encontra_componentes(Grafo *p) {
148   int *C = (int*) malloc(p->n * sizeof(int));
149   int u;
150   for (u = 0; u < p->n; u++)
151     C[u] = -1;
152   int comp = 0;
153   for (u = 0; u < p->n; u++){
154     if (C[u] == -1) {
155       visita_recursiva(p, C, comp, u);
156       comp++;
157     }
158   }
159 return C;
160 }
```



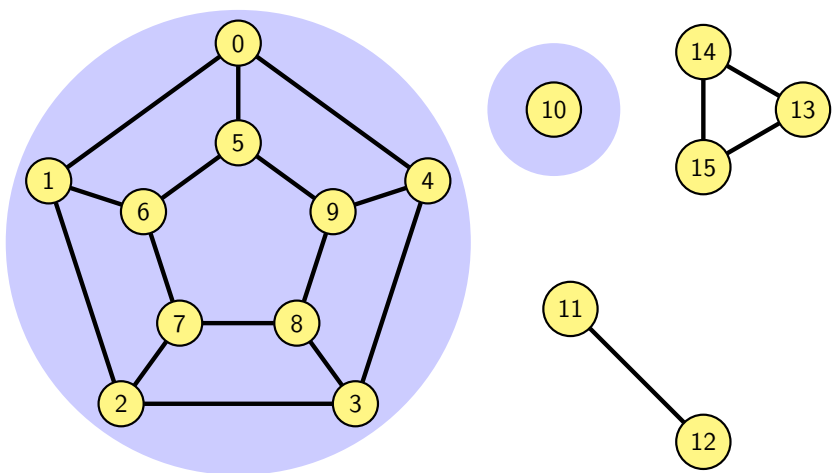
Como encontrar as componentes conexas?

```
138 void visita_recursiva(Grafo *p, int *C, int comp, int u) {
139   C[u] = comp;
140   int w;
141   for (w = 0; w < p->n; w++){
142     if (tem_aresta(p, u, w) && C[w] == -1)
143       visita_recursiva(p, C, comp, w);//busca em profundidade
144   }
145 }
146
147 int* encontra_componentes(Grafo *p) {
148   int *C = (int*) malloc(p->n * sizeof(int));
149   int u;
150   for (u = 0; u < p->n; u++)
151     C[u] = -1;
152   int comp = 0;
153   for (u = 0; u < p->n; u++){
154     if (C[u] == -1) {
155       visita_recursiva(p, C, comp, u);
156       comp++;
157     }
158   }
159 return C;
160 }
```



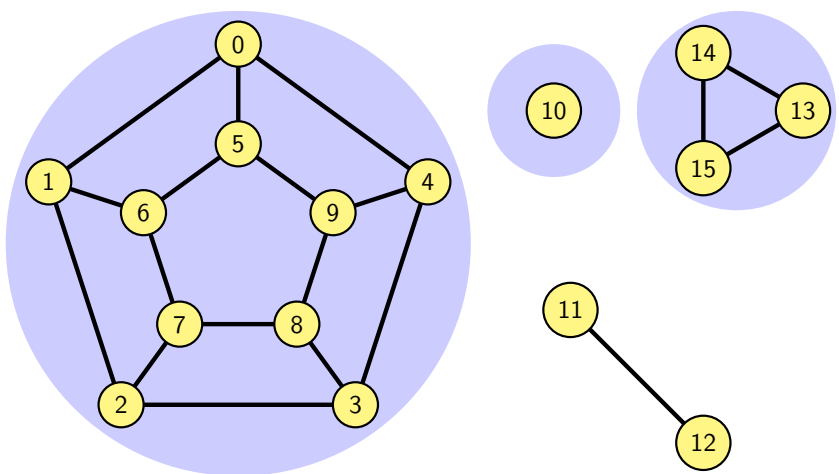
Como encontrar as componentes conexas?

```
138 void visita_recursiva(Grafo *p, int *C, int comp, int u) {
139   C[u] = comp;
140   int w;
141   for (w = 0; w < p->n; w++){
142     if (tem_aresta(p, u, w) && C[w] == -1)
143       visita_recursiva(p, C, comp, w);//busca em profundidade
144   }
145 }
146
147 int* encontra_componentes(Grafo *p) {
148   int *C = (int*) malloc(p->n * sizeof(int));
149   int u;
150   for (u = 0; u < p->n; u++)
151     C[u] = -1;
152   int comp = 0;
153   for (u = 0; u < p->n; u++){
154     if (C[u] == -1) {
155       visita_recursiva(p, C, comp, u);
156       comp++;
157     }
158   }
159 return C;
160 }
```



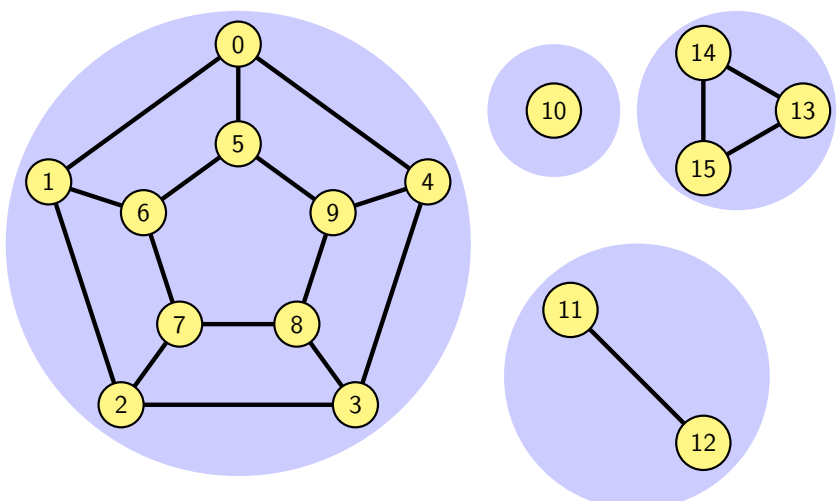
Como encontrar as componentes conexas?

```
138 void visita_recursiva(Grafo *p, int *C, int comp, int u) {
139   C[u] = comp;
140   int w;
141   for (w = 0; w < p->n; w++){
142     if (tem_aresta(p, u, w) && C[w] == -1)
143       visita_recursiva(p, C, comp, w);//busca em profundidade
144   }
145 }
146
147 int* encontra_componentes(Grafo *p) {
148   int *C = (int*) malloc(p->n * sizeof(int));
149   int u;
150   for (u = 0; u < p->n; u++)
151     C[u] = -1;
152   int comp = 0;
153   for (u = 0; u < p->n; u++){
154     if (C[u] == -1) {
155       visita_recursiva(p, C, comp, u);
156       comp++;
157     }
158   }
159 return C;
160 }
```



Como encontrar as componentes conexas?

```
138 void visita_recursiva(Grafo *p, int *C, int comp, int u) {
139   C[u] = comp;
140   int w;
141   for (w = 0; w < p->n; w++){
142     if (tem_aresta(p, u, w) && C[w] == -1)
143       visita_recursiva(p, C, comp, w);//busca em profundidade
144   }
145 }
146
147 int* encontra_componentes(Grafo *p) {
148   int *C = (int*) malloc(p->n * sizeof(int));
149   int u;
150   for (u = 0; u < p->n; u++)
151     C[u] = -1;
152   int comp = 0;
153   for (u = 0; u < p->n; u++){
154     if (C[u] == -1) {
155       visita_recursiva(p, C, comp, u);
156       comp++;
157     }
158   }
159 return C;
160 }
```



Cliente

exemplo3.c

```c
int main() {
  int n, m, i, u, v;
  scanf("%d %d", &n, &m);
  Grafo *G = criar_grafo(n);
  for (i = 0; i < m; i++) {
    scanf("%d %d", &u, &v);
    inserir_aresta(G, u, v);
  }
  int *C = encontra_componentes(G);
  int c=0;
  for (i = 0; i < n; i++) //encontra o número de componentes
    c = max(C[i], c);
  for (i = 0; i < c; i++){ //para cada componente
    printf("Componente %d: ", i+1);
    for (u = 0; u < n; u++)
      if(C[u]==i) printf("%d ", u);
    printf("\n");
  }
  free(C);
return 0;
}
```

Makefile

Vamos usar o Makefile para compilar:

```
LIB = grafo_matriz.o \
      lista_circular.o \
      fila.o
```

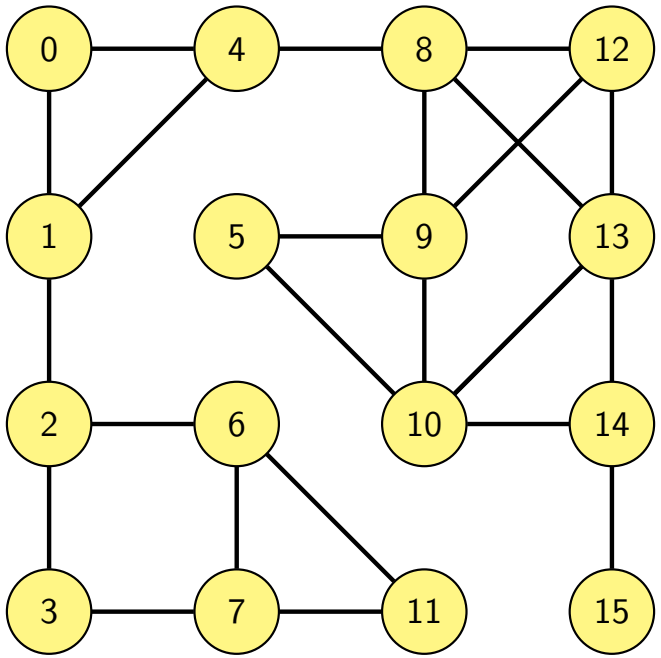
```
exemplo3: exemplo3.c $(LIB)
  gcc $^ -o $@
```

Vamos executar:

```
$ ./exemplo3 < teste3.in
Componente 1: 0 1 2 3 4 5 6 7 8 9
Componente 2: 10
Componente 3: 11 12
```

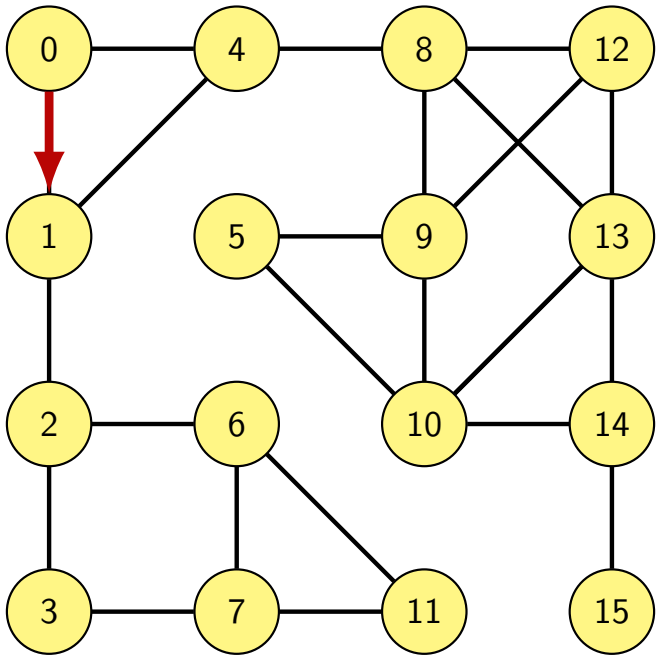
Roteiro

Caminhos de u para outros vértices da componente



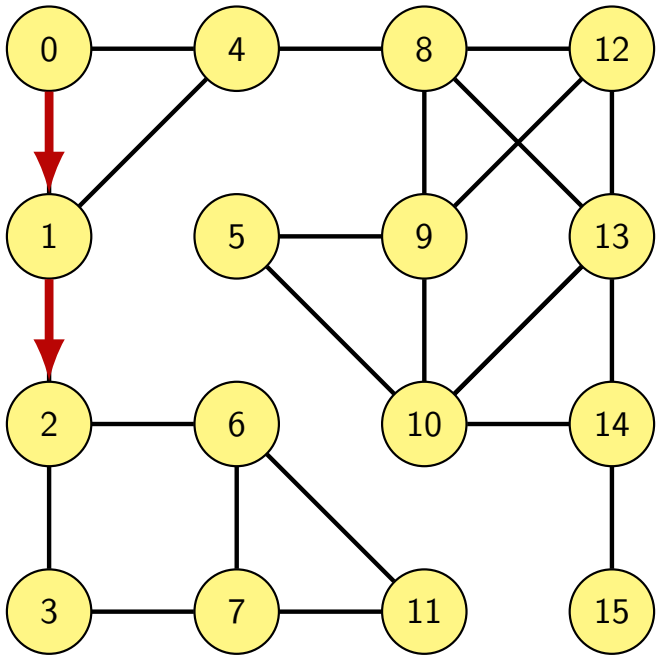
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



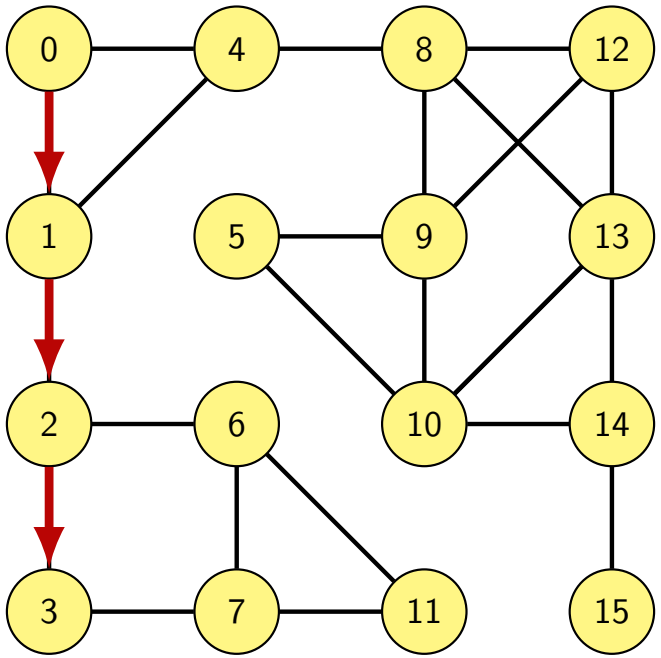
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



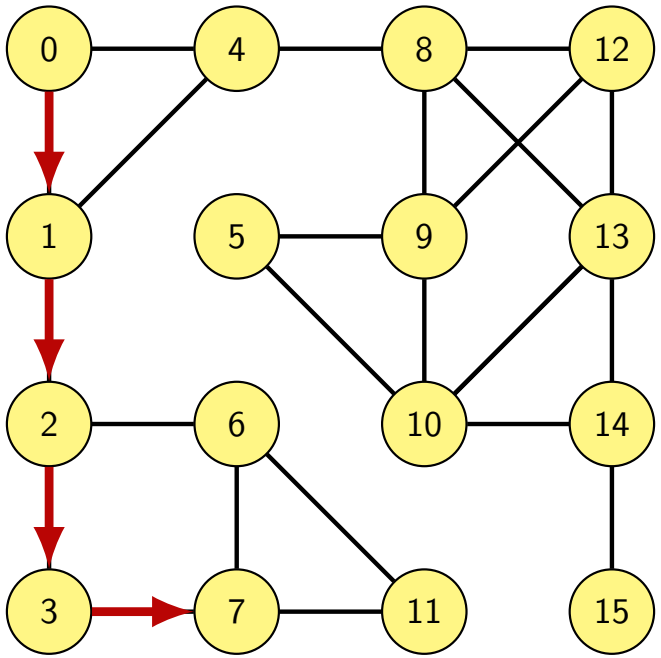
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



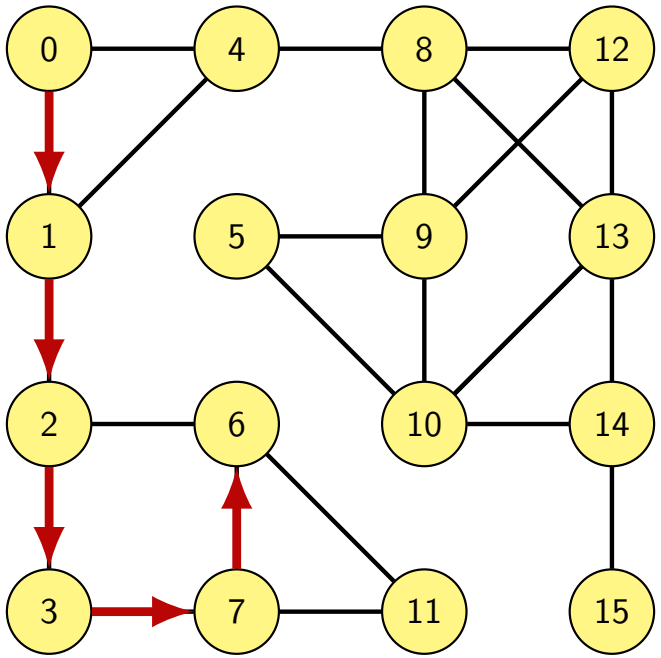
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



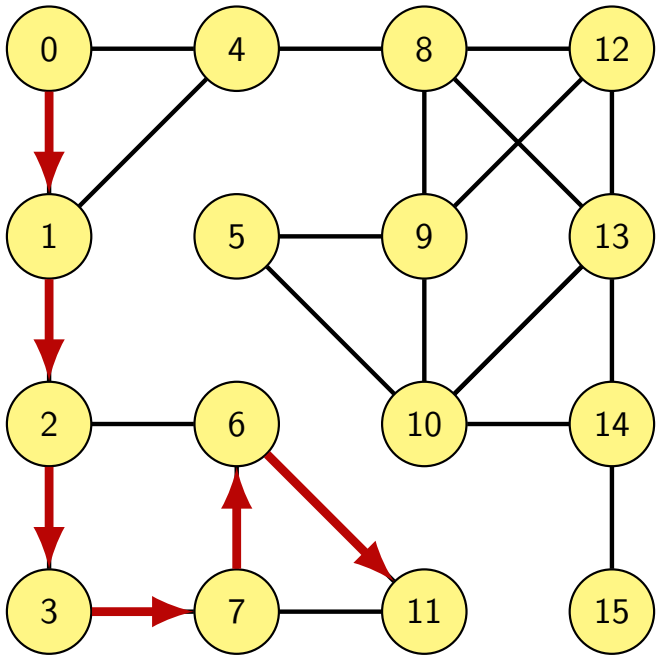
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



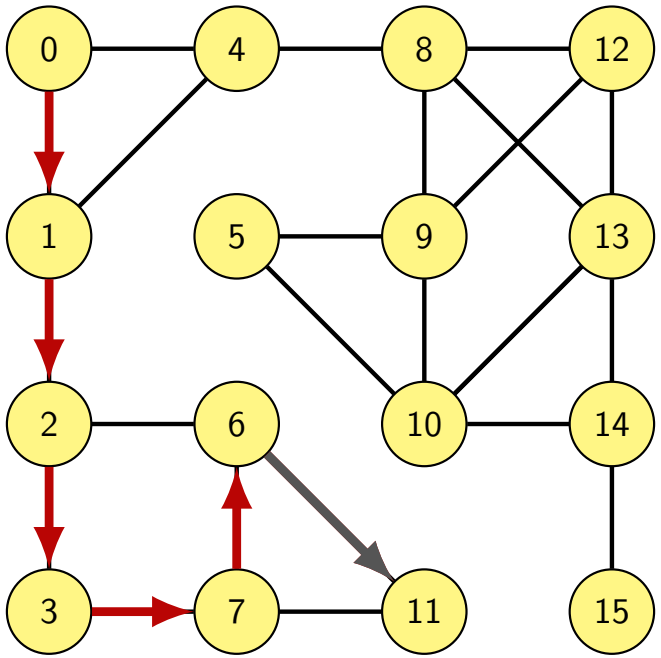
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



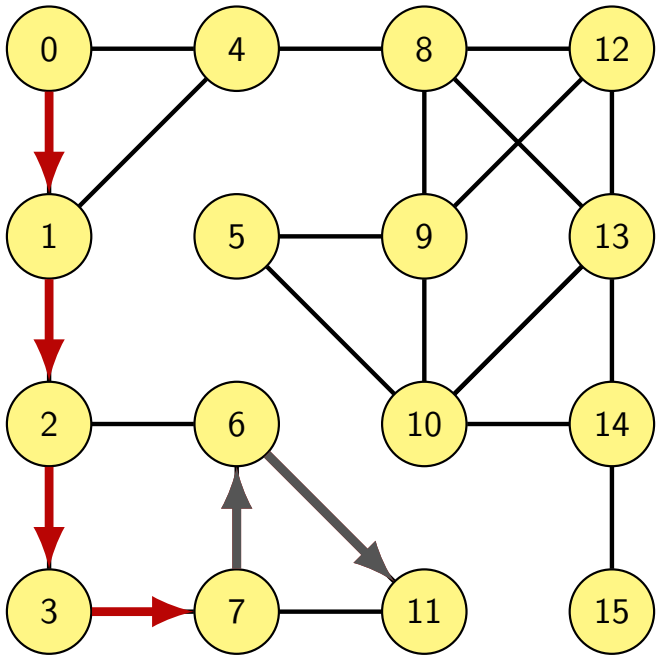
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



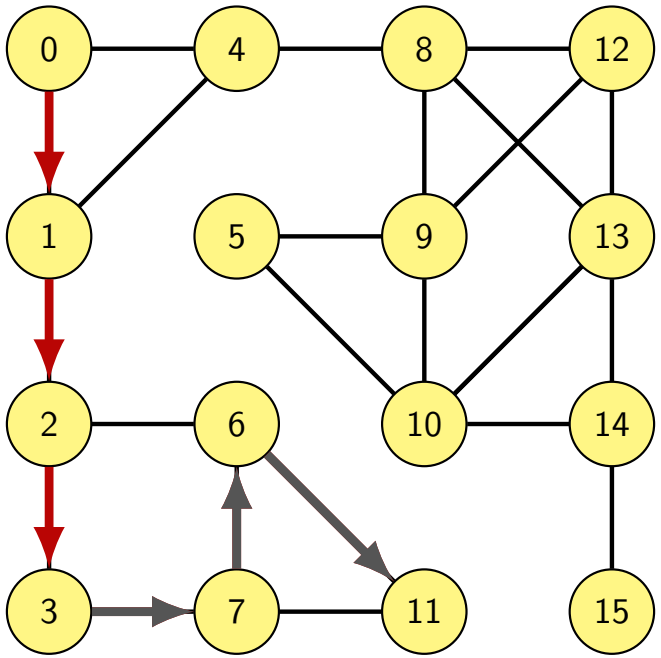
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



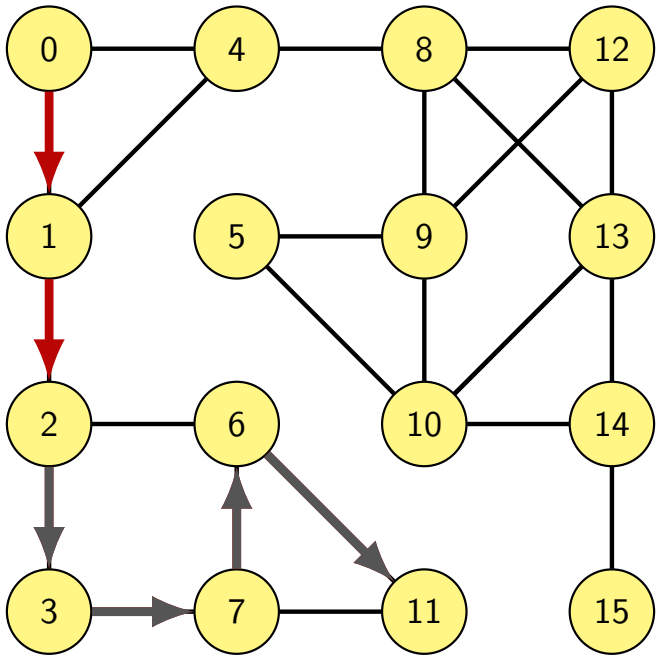
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



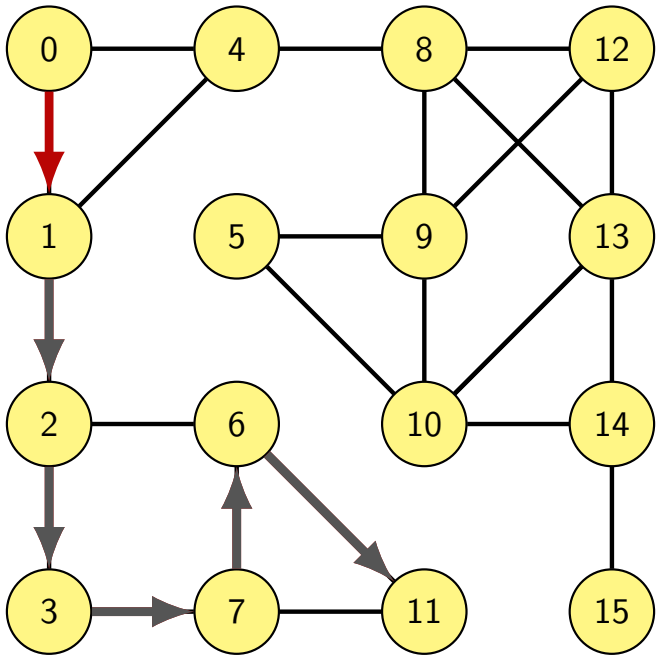
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



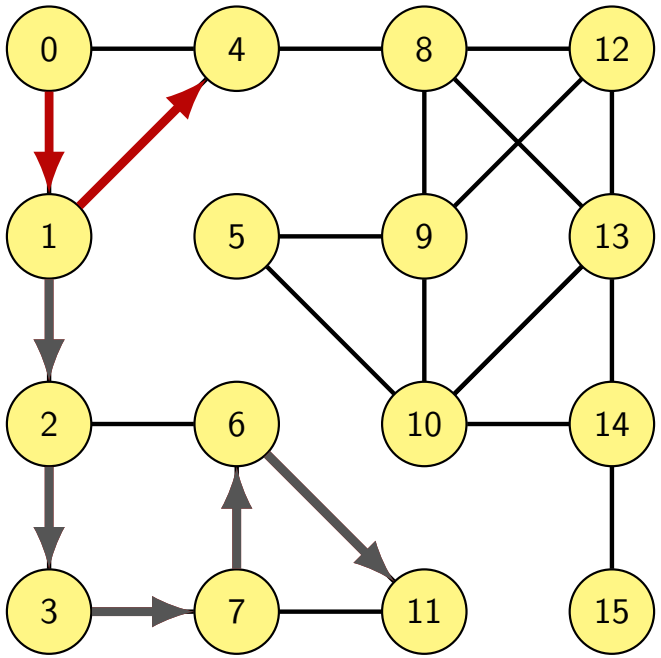
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



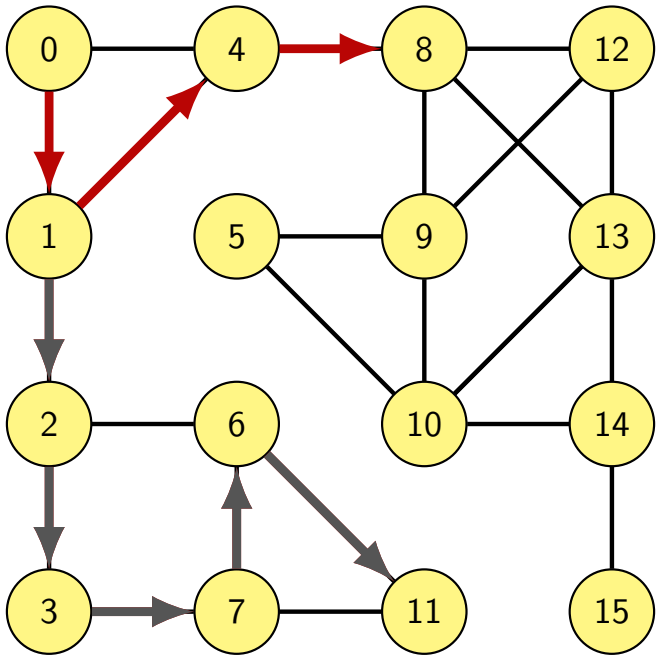
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



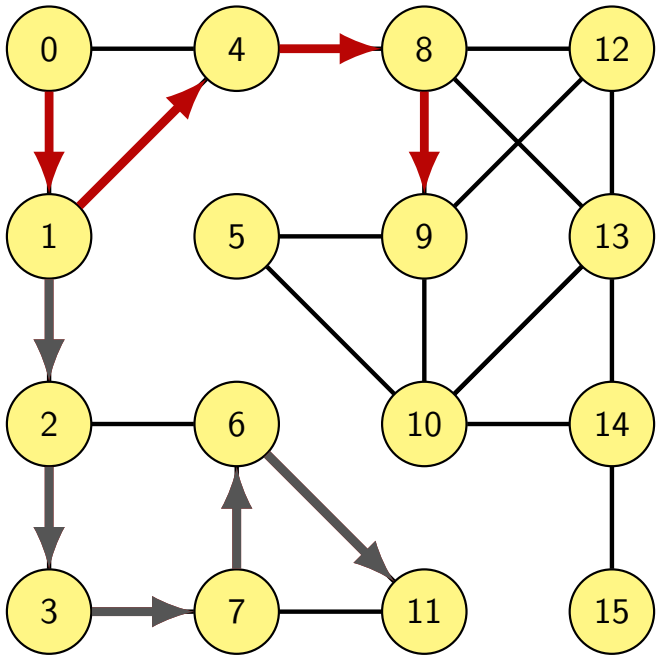
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



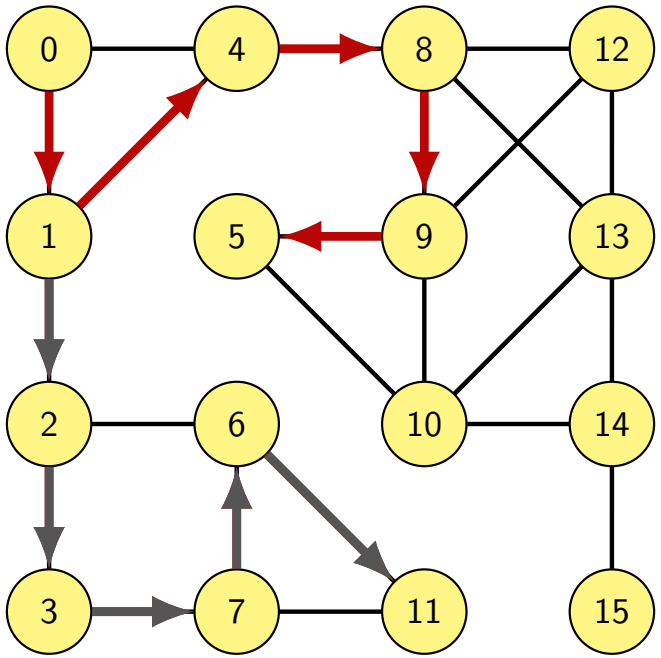
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



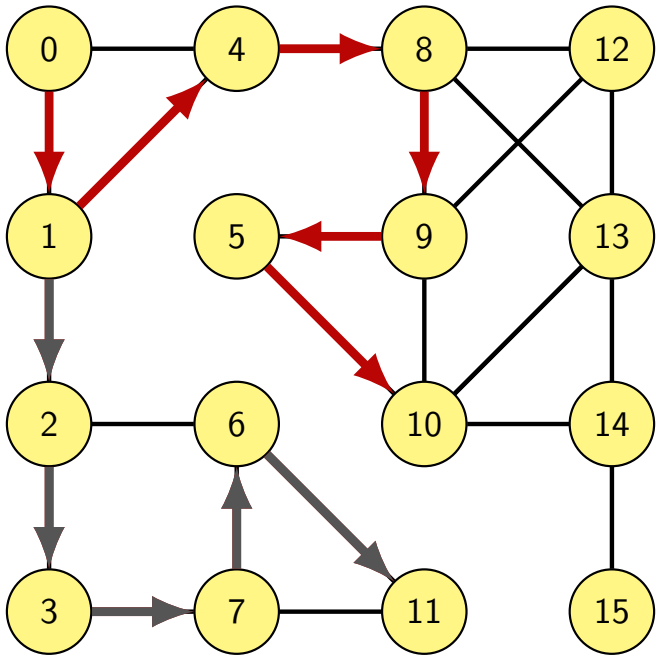
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



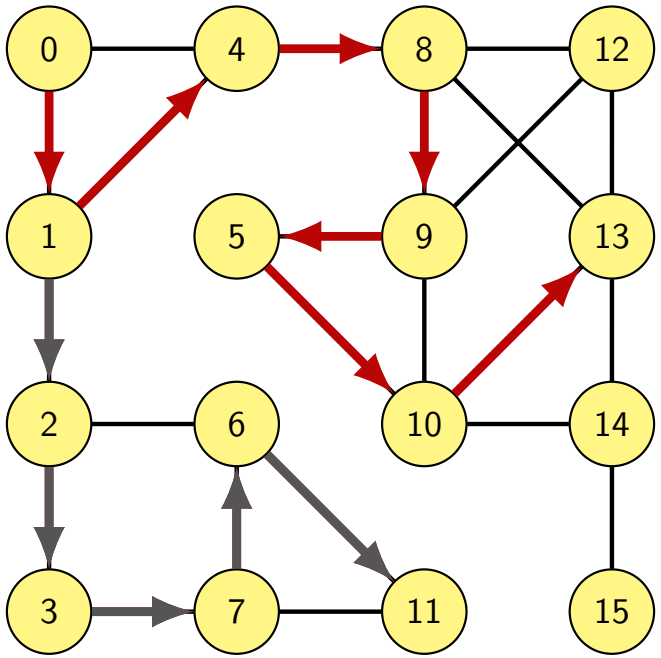
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



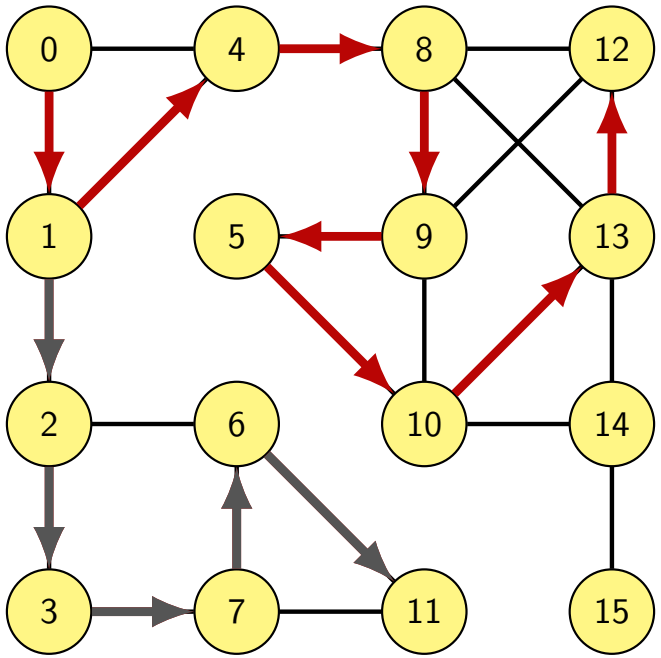
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



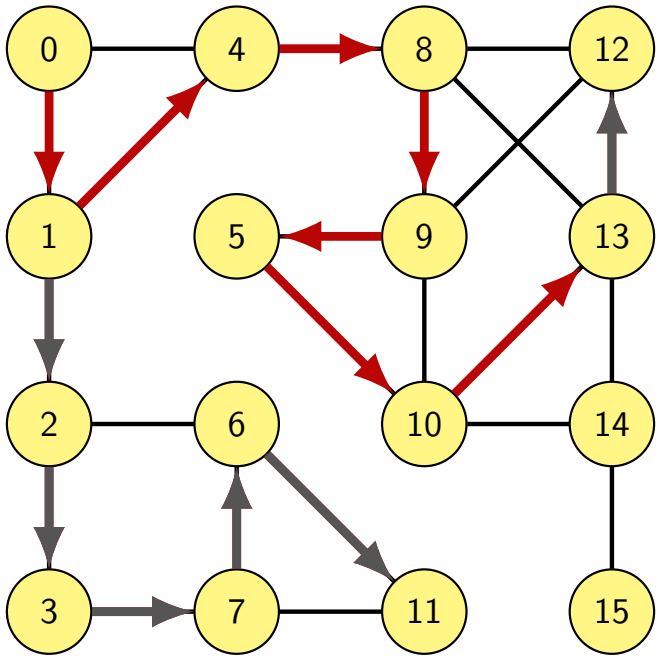
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



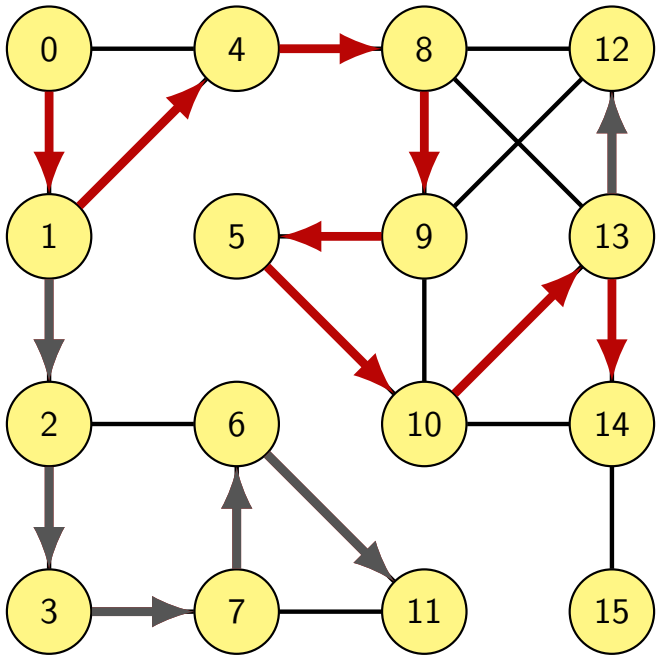
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



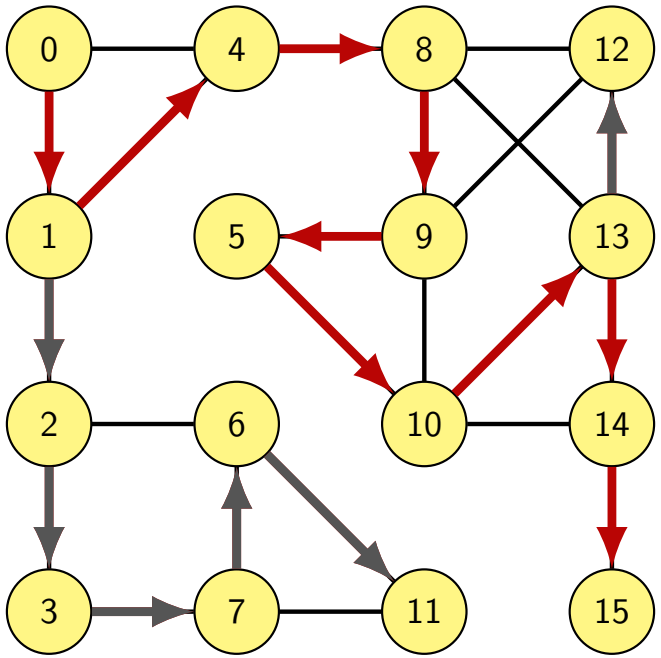
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



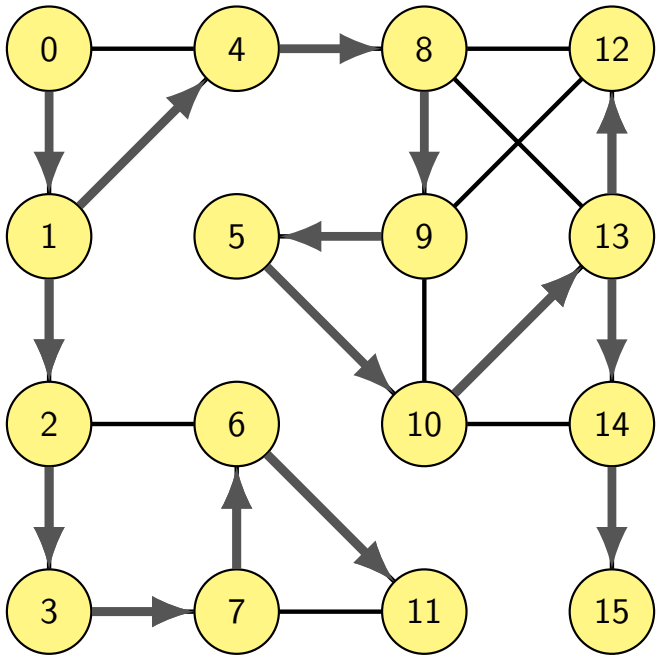
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



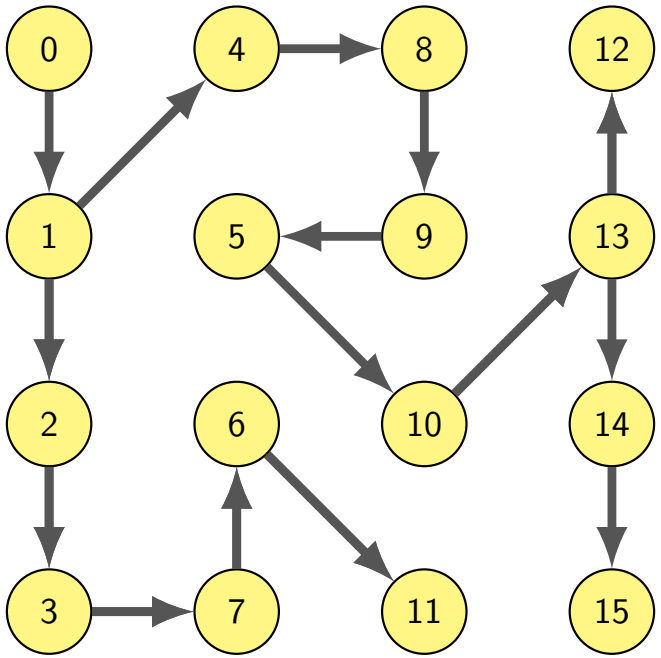
Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



Considere uma busca em profundidade no grafo G.

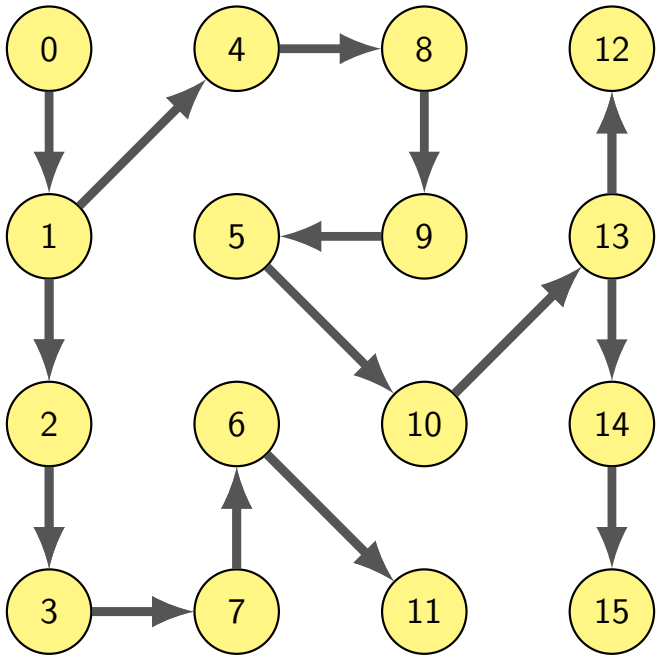
Caminhos de u para outros vértices da componente



As arestas usadas formam uma árvore!

Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente

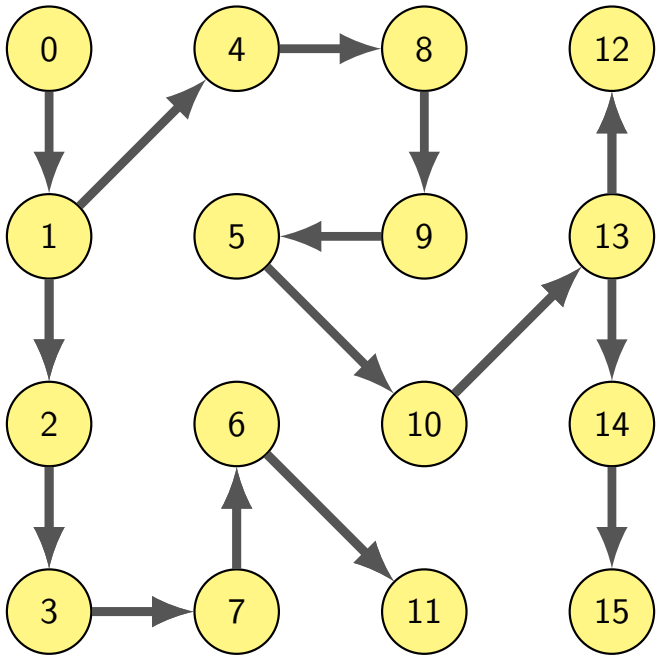


As arestas usadas formam uma árvore!

- Essa árvore dá **um** caminho de **qualquer vértice** até a raiz (origem)

Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente



As arestas usadas formam uma árvore!

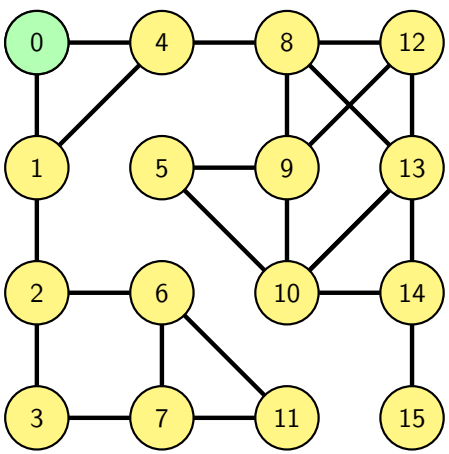
- Essa árvore dá **um** caminho de **qualquer vértice** até a raiz (origem)
- Para saber $u \rightsquigarrow v$: basta ir "subindo" na árvore

---

Considere uma busca em profundidade no grafo G.

Caminhos de u para outros vértices da componente

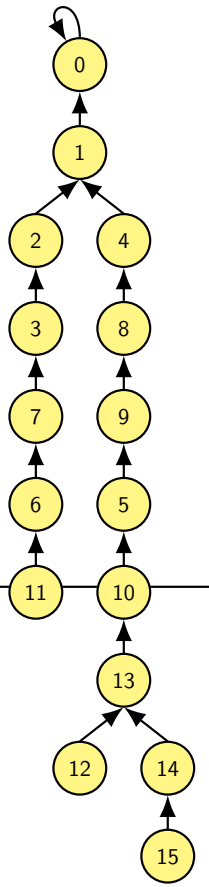
```
164 void visita_recursiva_v2(Grafo *p, int *pai, int u, int v) {
165   pai[v] = u;
166   int w;
167   for (w = 0; w < p->n; w++)
168     if (tem_aresta(p, v, w) && pai[w] == -1)
169       visita_recursiva_v2(p, pai, v, w);//busca em profundidade
170 }
171
172 int *encontra_caminhos(Grafo *p, int u) {
173   int i, *pai = (int*) malloc(p->n * sizeof(int));
174   for (i = 0; i < p->n; i++) pai[i] = -1;
175   visita_recursiva_v2(p, pai, u, u);
176 return pai;
177 }
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Imprimindo o caminho

```
179 void imprimir_caminho_reverso(int *pai, int v) {
180   printf("%d ", v);
181   if(pai[v] != v)
182     imprimir_caminho_reverso(pai, pai[v]);
183 }
184
185 void imprimir_caminho(int *pai, int v) {
186   if(pai[v] != v)
187     imprimir_caminho(pai, pai[v]);
188   printf("%d ", v);
189 }
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	0	9	7	3	4	8	5	6	13	10	13	14

Cliente

exemplo4.c

```
7  int main() {
8    int n, m, i, u, v;
9    scanf("%d %d", &n, &m);
10   Grafo *G = criar_grafo(n);
11   for (i = 0; i < m; i++) {
12     scanf("%d %d", &u, &v);
13     inserir_aresta(G, u, v);
14   }
15   int *pai = encontra_caminhos(G, 0);
16   for (i = 0; i < n; i++){
17     printf("%d: ", i);
18     imprimir_caminho(pai, i);
19     printf("\n");
20   }
21   free(pai);
22 return 0;
23 }
```

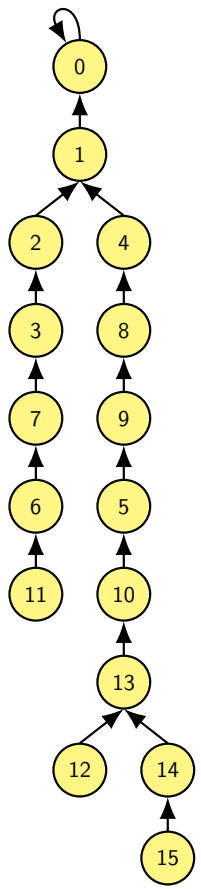
Makefile

Vamos usar o Makefile para compilar:

```
exemplo4: exemplo4.c $(LIB)
  gcc $^ -o $@
```

Vamos executar:

```
$ ./exemplo4 < teste4.in
0: 0
1: 0 1
2: 0 1 2
3: 0 1 2 3
4: 0 4
5: 0 4 8 9 5
6: 0 1 2 3 7 6
7: 0 1 2 3 7
8: 0 4 8
9: 0 4 8 9
10: 0 4 8 9 5 10
11: 0 1 2 3 7 6 11
12: 0 4 8 9 5 10 13 12
13: 0 4 8 9 5 10 13
14: 0 4 8 9 5 10 13 14
15: 0 4 8 9 5 10 13 14 15
```



Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Rafael C. S. Schouery, da Universidade Estadual de Campinas.
2. R. Sedgewick, "*Algorithms in C - Parts 5 - Third Edition*" (Capítulo 18)