Grafos

Marcelo K. Albertini

1 de agosto de 2023

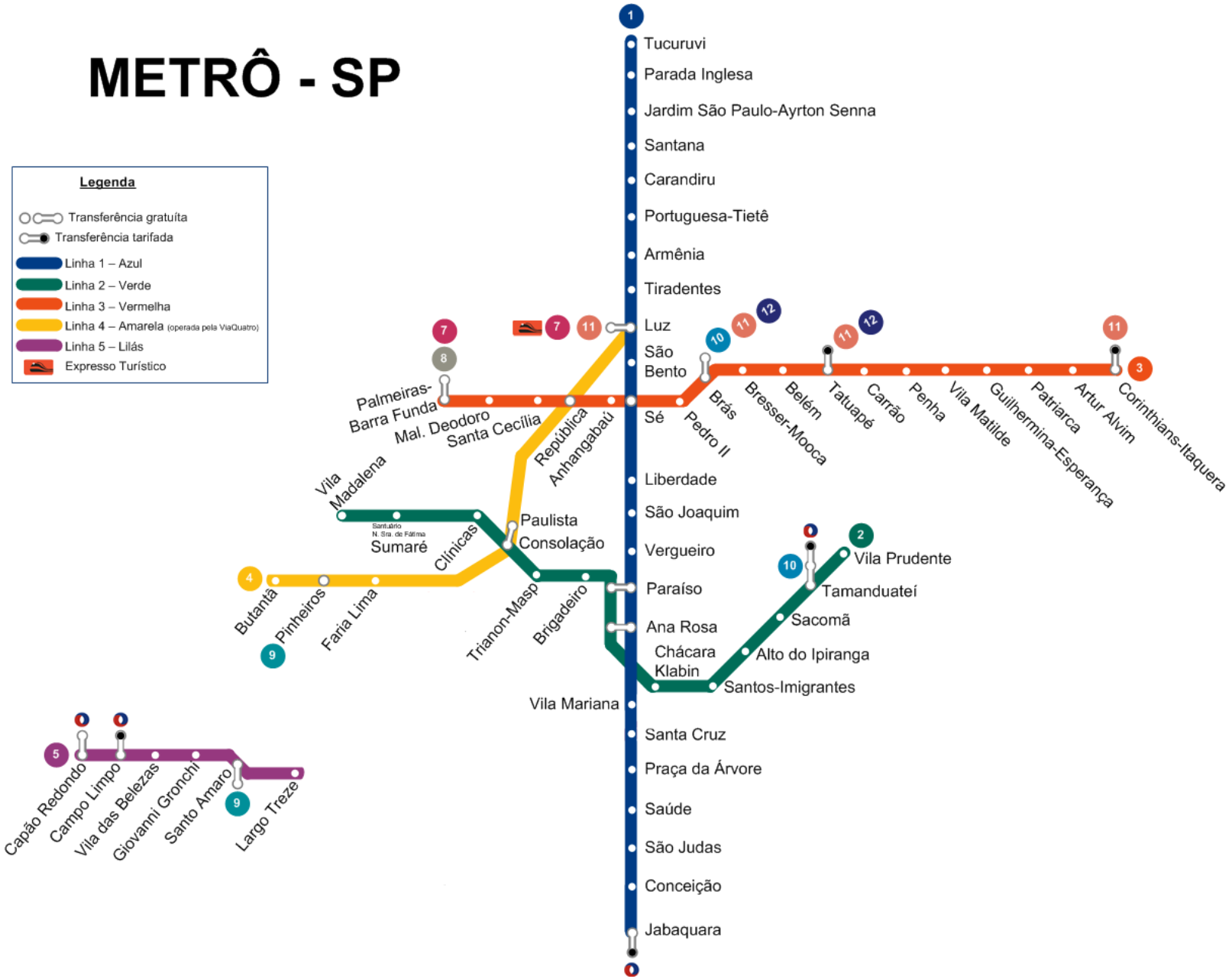
Grafo

Um conjunto de vértices conectados em pares por arestas.

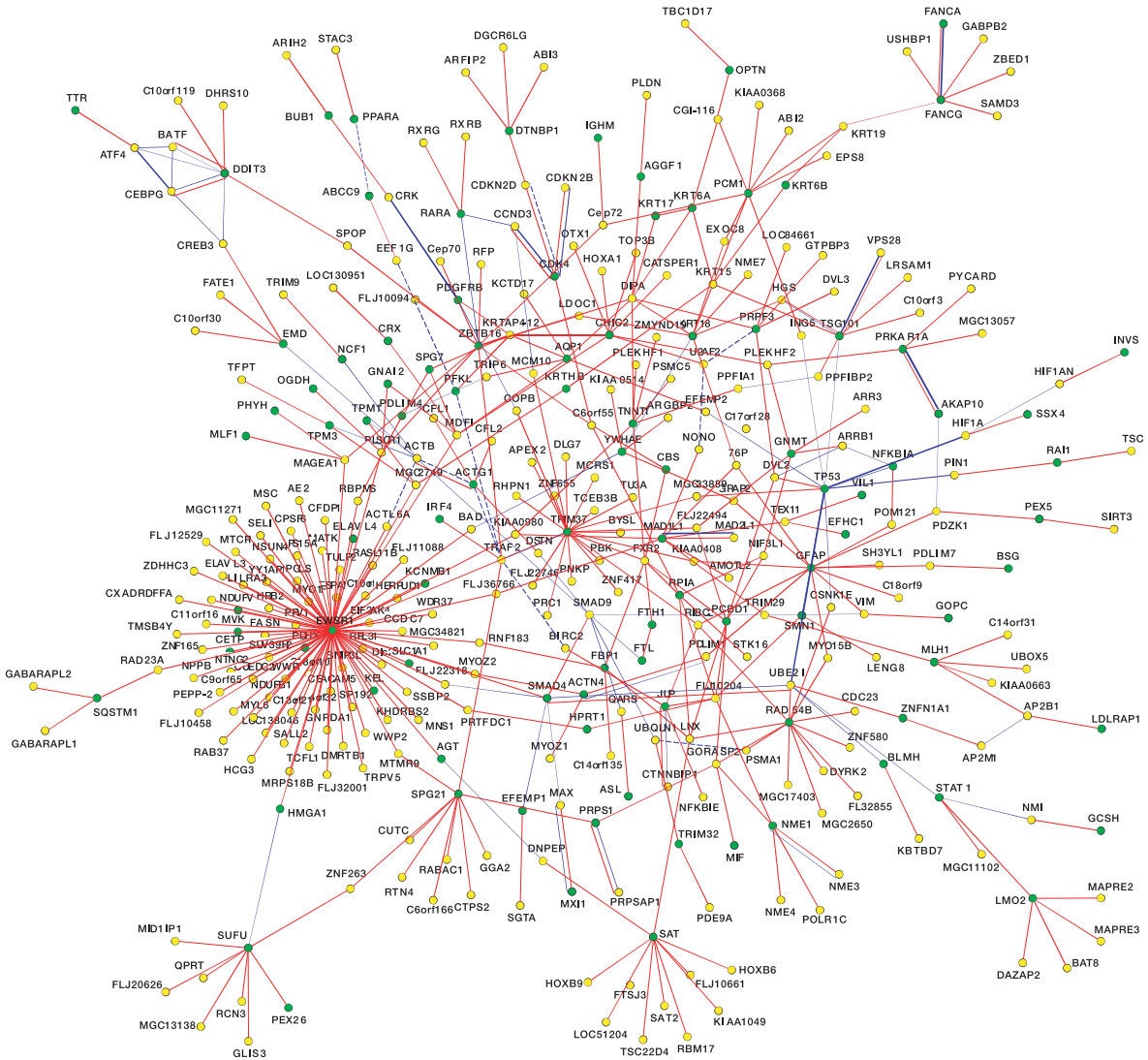
Porque estudar grafos?

- Muitas aplicações
- Abstração útil e interessante
- Muitos algoritmos para processar grafos existentes
- Campo de conhecimento em constante evolução

Transporte público

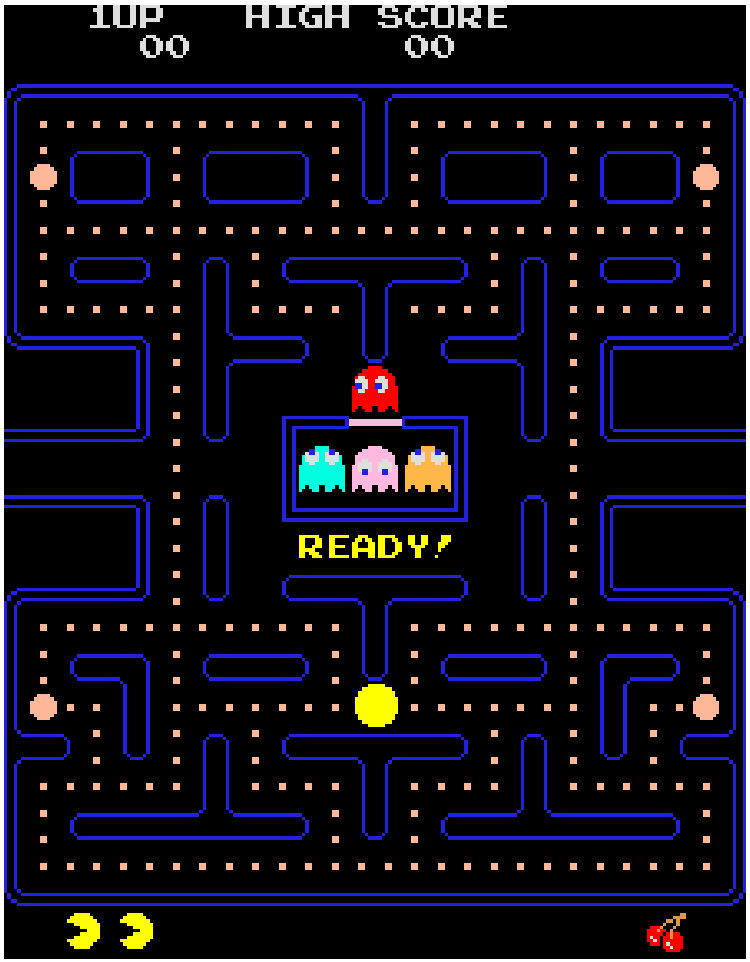


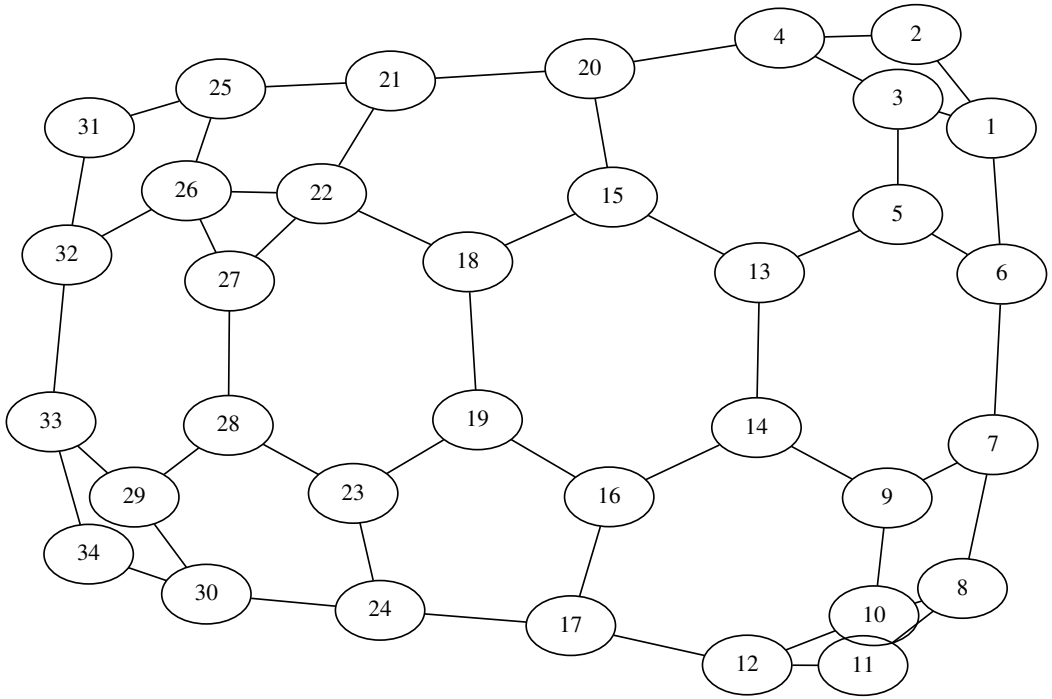
Grafo da interação de proteínas humanas com doenças



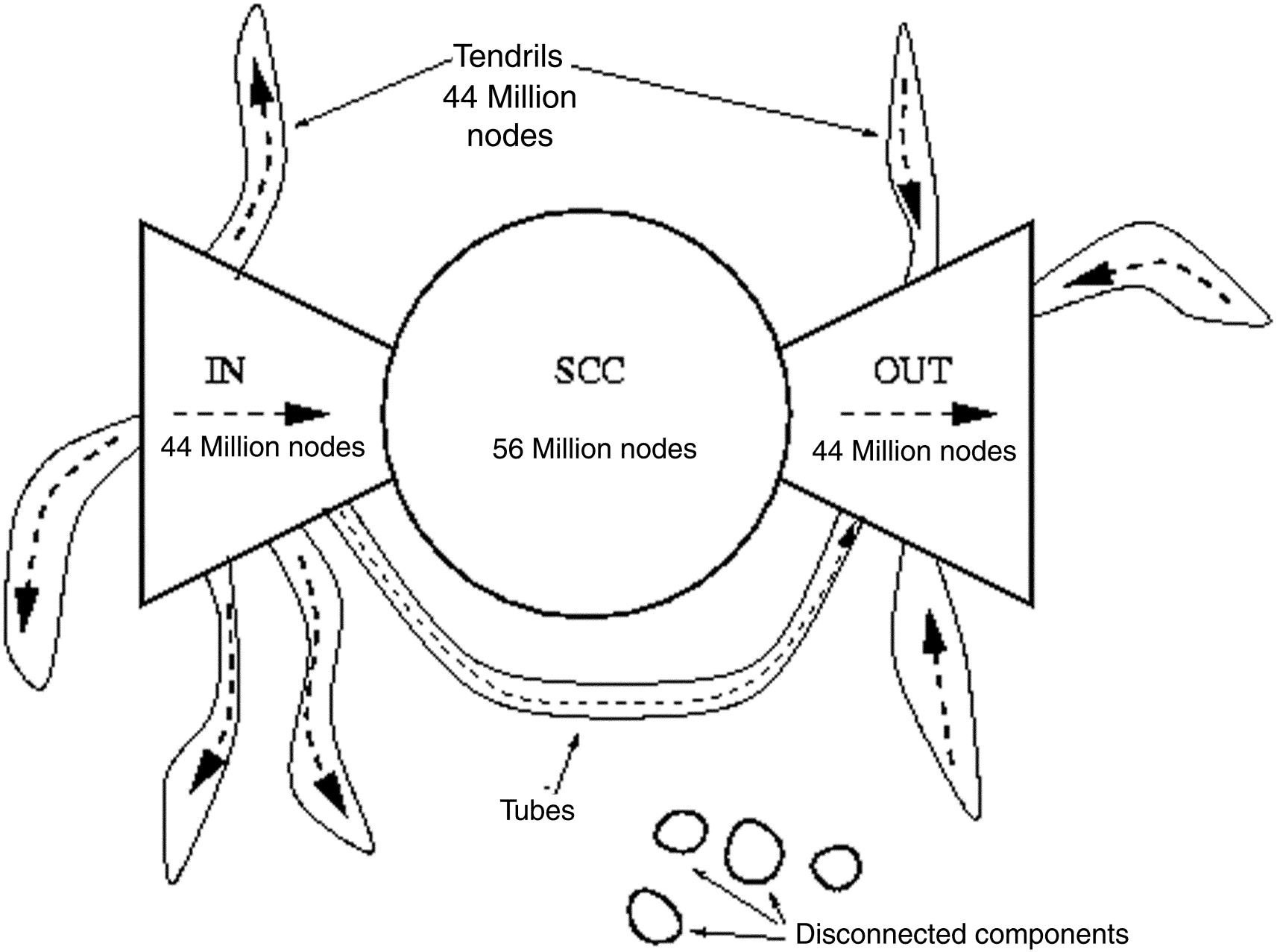
- Interaction network of disease-associated CCSB-HI1 proteins. *Towards a proteome-scale map of the human protein-protein interaction network. Rual et al., Nature 2005.*

Jogos


1UP HIGH SCORE
00 00
READY!


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Graph structure in the web, Broder et al. (2000)



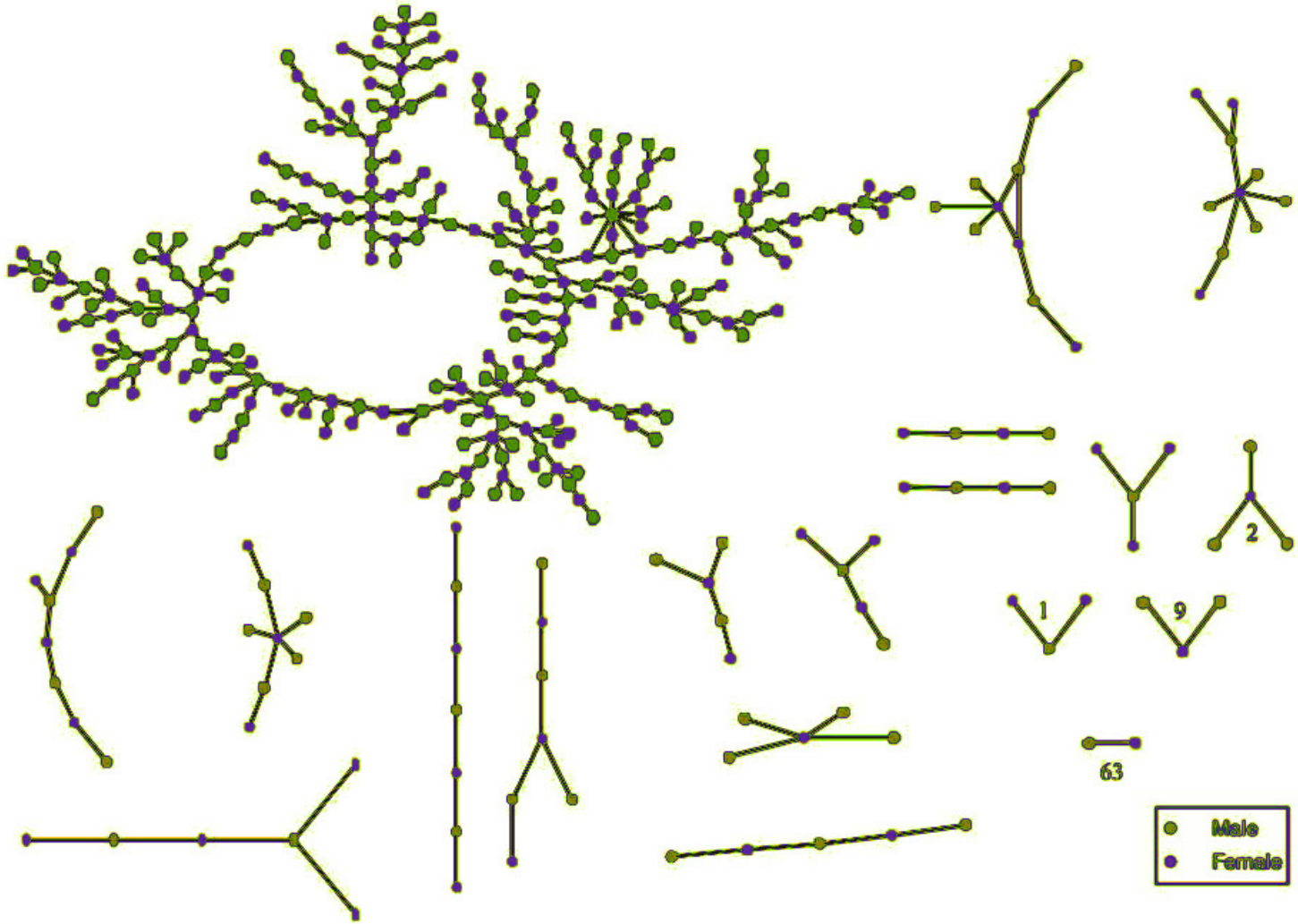
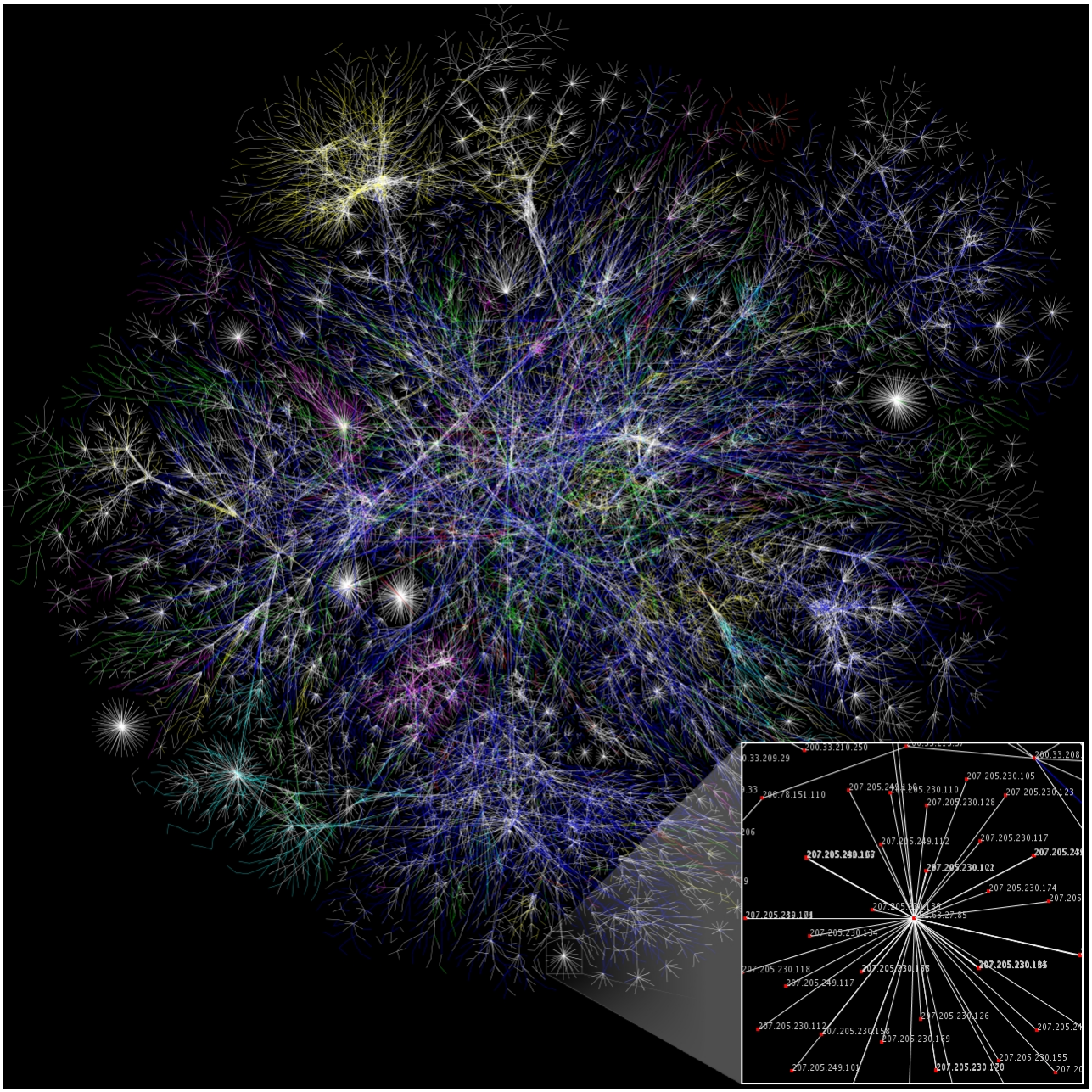


Figura: Peter Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. American Journal of Sociology, 110(1): 44-99, 2004.

Mapa da Internet



Rede elétrica

Kim and Obah: Vulnerability Assessment of Power Grid Using Topological Index

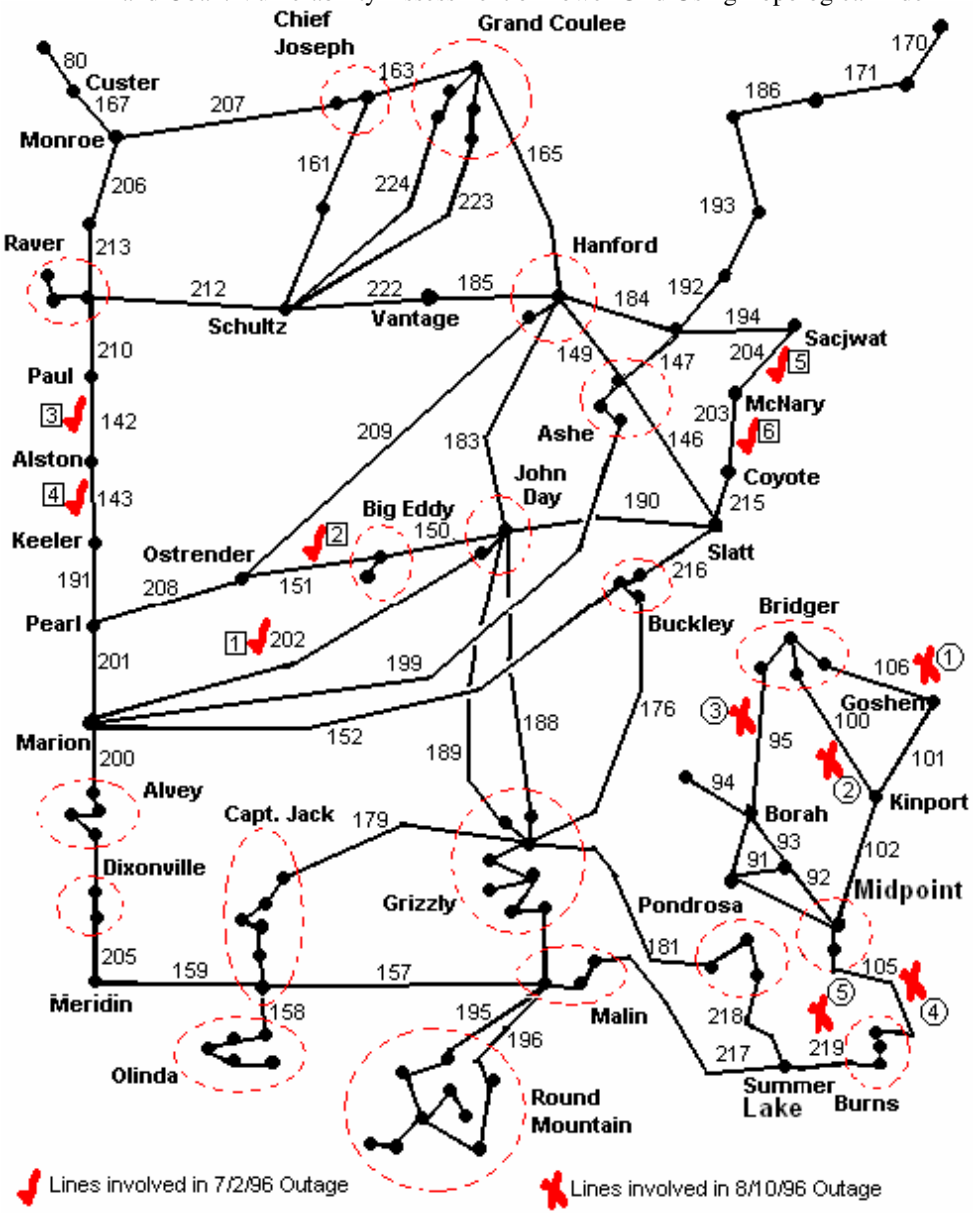
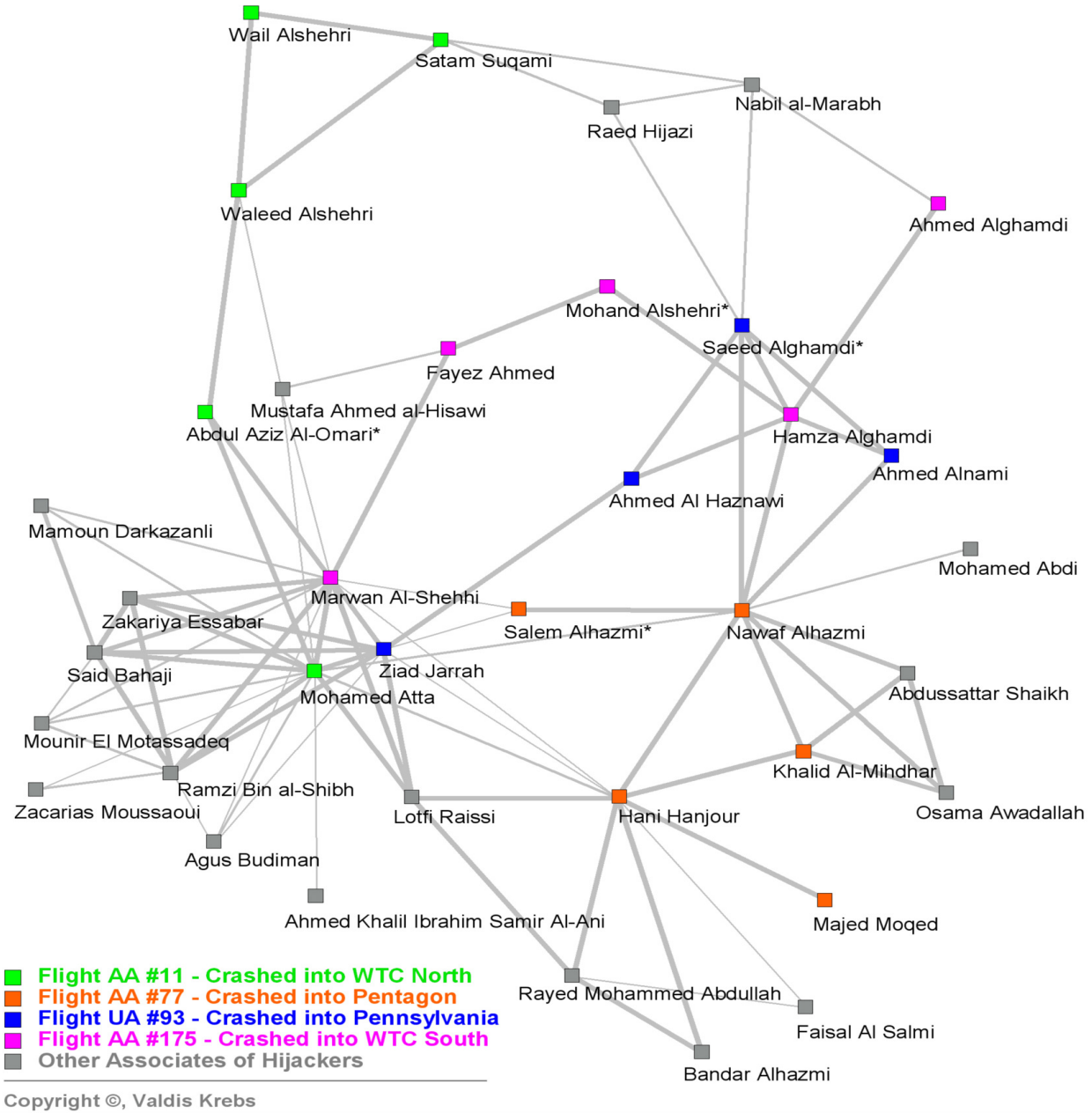


Figure 3. Reconstructed outages of 1996 WSCC blackout.

Grafo de terroristas



Outras aplicações de grafos

grafo	vértice	aresta
comunicações	telefone, computadores	fibra ótica
circuito	porta, registrador, processador	fio
finanças	ações, cotações	transações
transporte	cruzamento, aeroporto	rodovia, rota aéra
internet	rede classe C	conexão
jogos	posição no tabuleiro	movimento de peça
rede social	pessoa, ator	amizade, alenco
rede neural	neurônio	sinapse
rede proteica	proteína	interação de proteínas
composto químico	molécula	ligação

Terminologia: Graus de vértices

- Grau de um vértice
  - número de arestas que chegam e saem

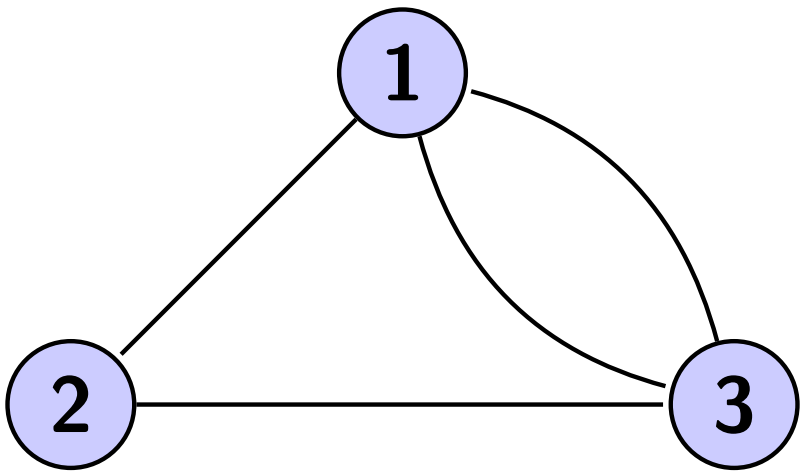


Figura: $grau(1) = 3$

Terminologia: Caminho em um grafo

Dois vértices estão conectados se existe um caminho entre eles.

- Caminho: sequência de vértices ligados por arestas
- Um vértice é alcançável se existe um caminho a ele.

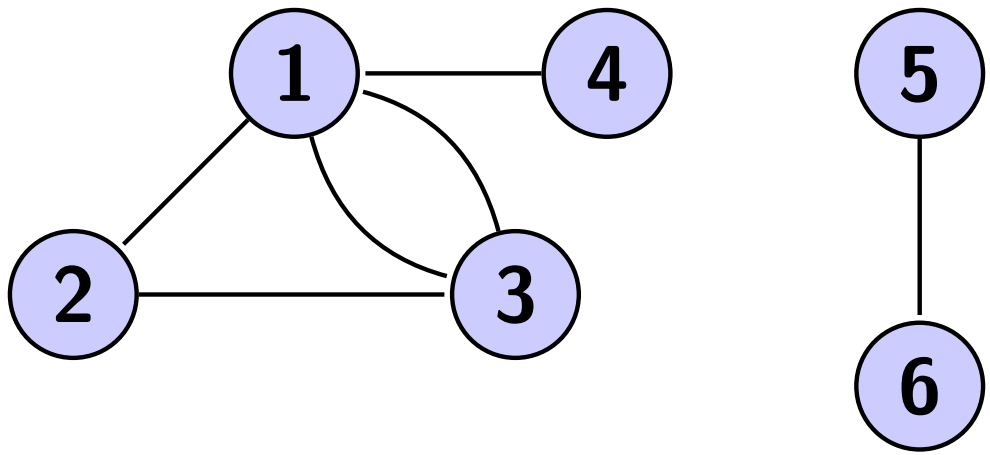
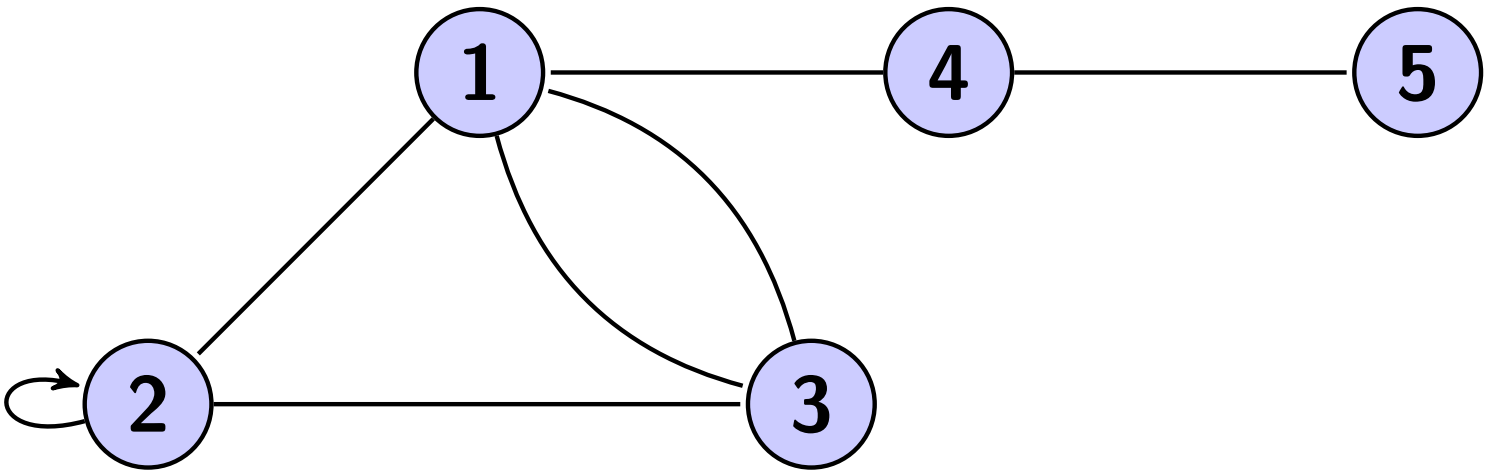


Figura: $(3, 1, 3, 1)$ é um caminho de tamanho 3. $(1, 2, 3)$ é um ciclo de tamanho 3. Vértices 5 e 6 não são alcançáveis a partir de nenhum outro vértice.

Terminologia: Caminho em um grafo

Definições: caminho

- Caminho é simples se todos os vértices são distintos

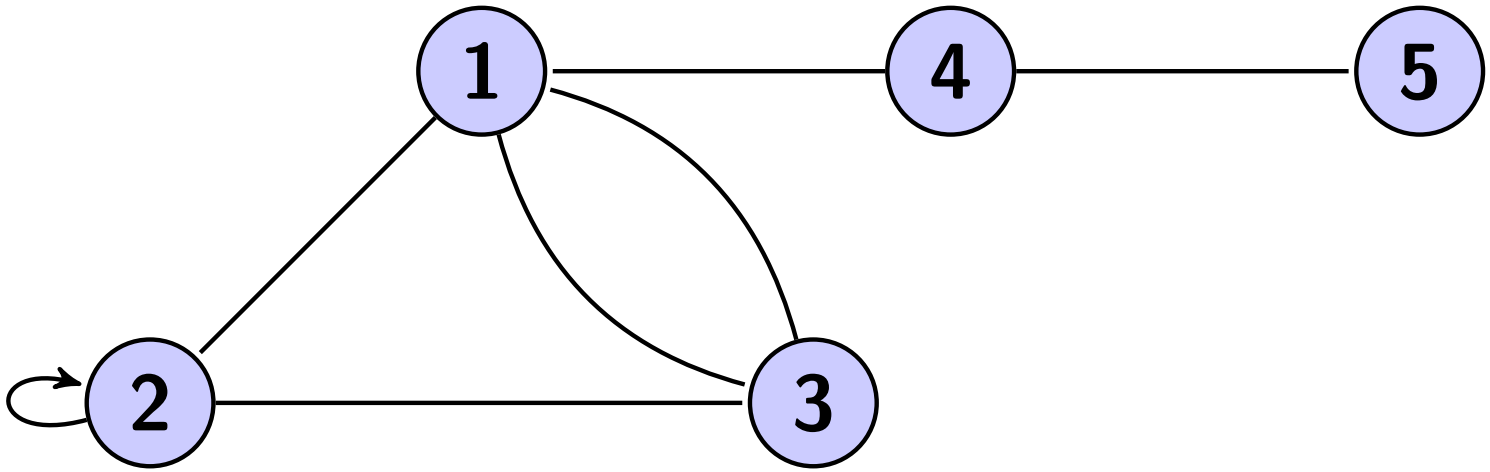


$(1, 2, 3)$ é um caminho simples.
$(1, 2, 3, 1)$ não é um caminho simples.

Terminologia: Caminhos cíclicos ou ciclos

Definições: ciclo

- Um ciclo é um caminho que começa e termina com o mesmo vértice
- Em um ciclo **simples**, apenas o primeiro e o último vértices são iguais
- Caso especial: um laço é ciclo com uma só aresta



$(1, 2, 3, 1)$ é um ciclo.
$(2, 2)$ é um ciclo.

Terminologia: Grafos conexos e desconexos

Definições: em um grafo não direcionado

- Um grafo é conexo se cada vértice tem pelo menos um caminho a qualquer outro vértice

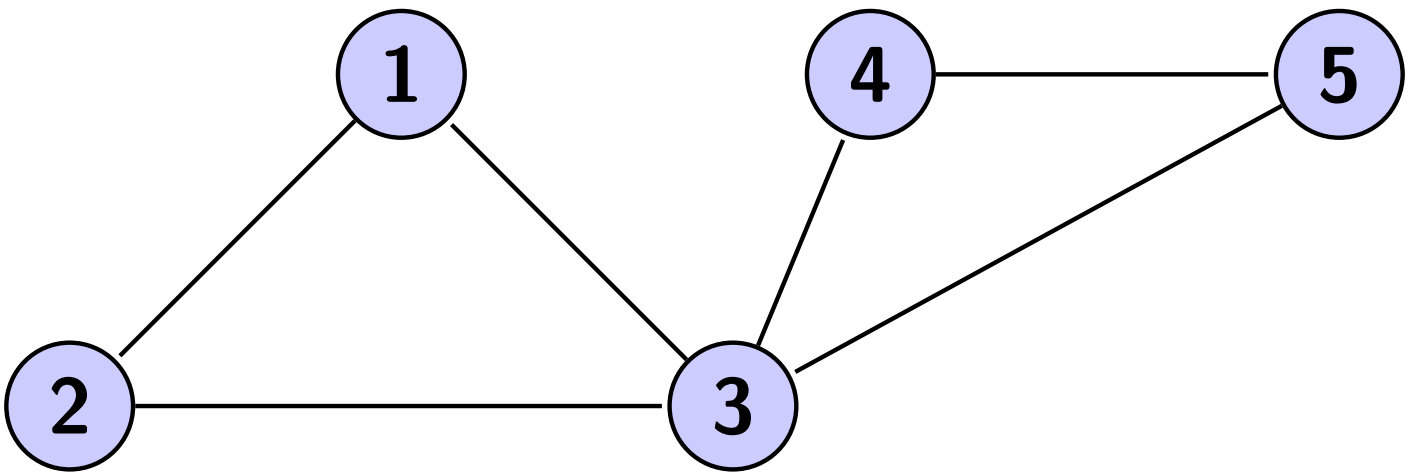


Figura: Grafo conexo

Terminologia: Grafos conexos e desconexos

Definições: em um grafo não direcionado

- Um grafo é conexo se cada vértice tem pelo menos um caminho a qualquer outro vértice

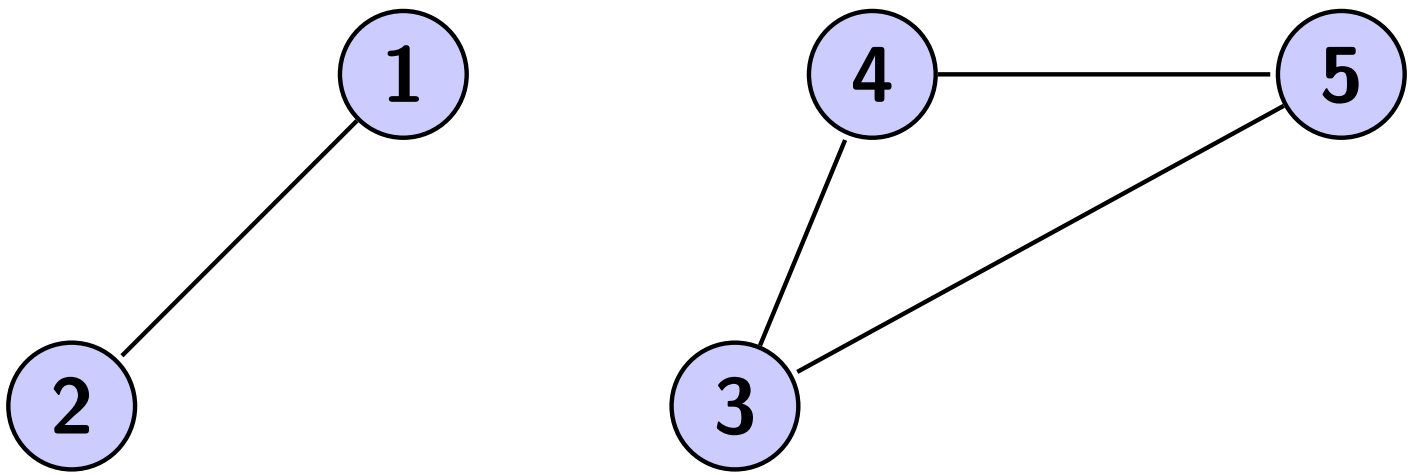


Figura: Grafo desconexo

Terminologia: Grafos completos

Definições: grafo completo

- Um grafo é completo se todos os vértices tiverem aresta ligando a todos os outros vértices
- Um grafo completo com n vértices é chamado K_n

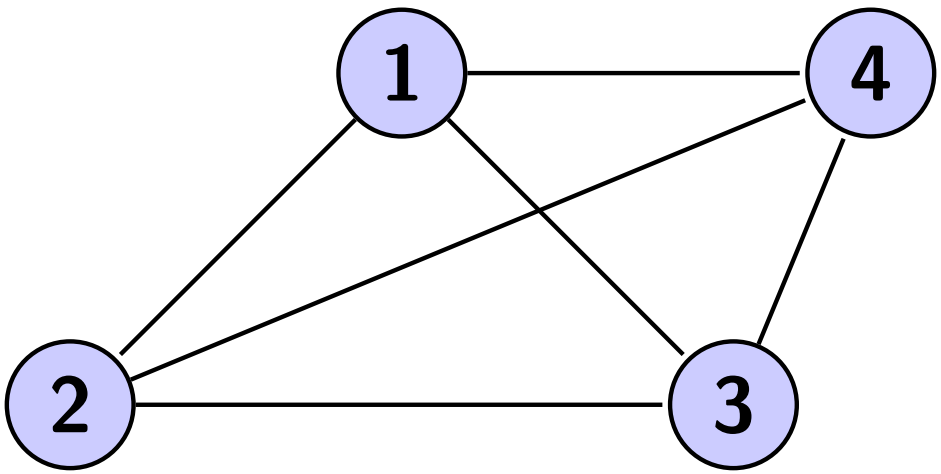


Figura: Grafo completo K_4

Problemas de processamento de grafos

Caminho. Existe um caminho entre vértices `s` e `t`?
Caminho mais curto. Qual é o caminho mais curto de `s` a `t`?

Ciclo. Existe um ciclo no grafo?
Caminho de Euler. Existe um caminho que usa cada aresta exatamente uma vez?
Ciclo de Hamilton. Existe um ciclo que usa cada vértice exatamente uma vez?

Conectividade. Todos os vértices estão conectados entre si?
Árvore Geradora Mínima. Qual é a melhor maneira de conectar todos os vértices?
Biconectividade. Existe um vértice que se removido disconecta o grafo?

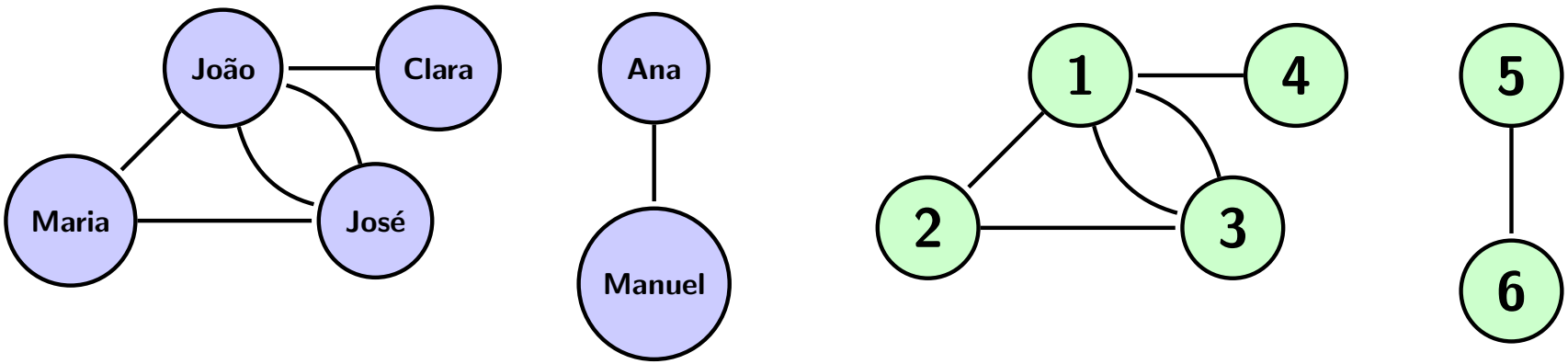
Planaridade. Podemos desenhar o grafo em um plano sem cruzar vértices?
Isomorfismo. Duas estruturas de dados representam o mesmo grafo?

Queremos saber quais desses problemas são fáceis, difíceis ou intratáveis.

Representação de grafos

Representação de vértices

- Usamos números inteiros entre 0 e $V - 1$, onde V é o número de vértices
- Podemos usar tabela de símbolos para converter nomes em inteiros



API de Grafos - Tipo Abstrato de Dados

public class	Grafo	
	Grafo(int V)	criar grafo vazio com V vértices
	Grafo(InputStream in)	criar grafo a partir de entradas
void	novaAresta(int v, int w)	criar aresta
Iterable<Integer>	adj(int v)	vértices adjacentes a *v*
int	V()	número de vértices
int	E()	número de arestas

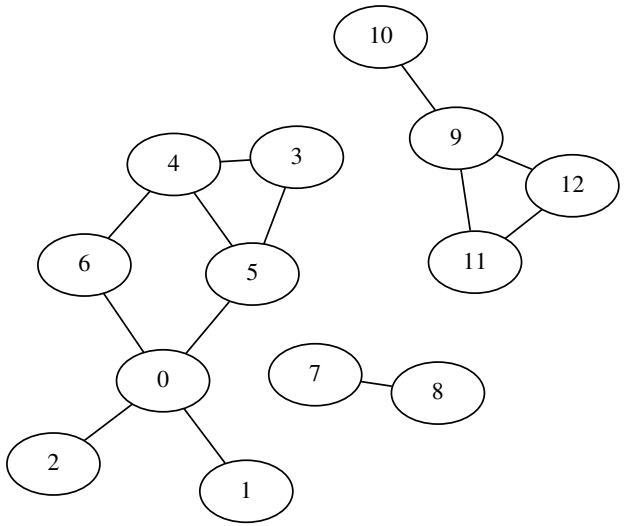
Exemplo de uso

```
1 public static void main(String args[]) {
2   Grafo G = new Grafo(System.in);
3
4   for (int v = 0; v < G.V(); v++)
5     for (int w: G.adj(v))
6       System.out.println(v+"--"+w);
7 }
```

Exemplo de uso: API de Grafo

Arquivo `grafo.in`

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



Comando no terminal:

```
1 % java Grafo < grafo.in
```

Saída:

```
1 0 — 5
2 0 — 1
3 0 — 2
4 0 — 6
5 1 — 0
6 2 — 0
7 3 — 4
8 ...
```

Operações em grafos

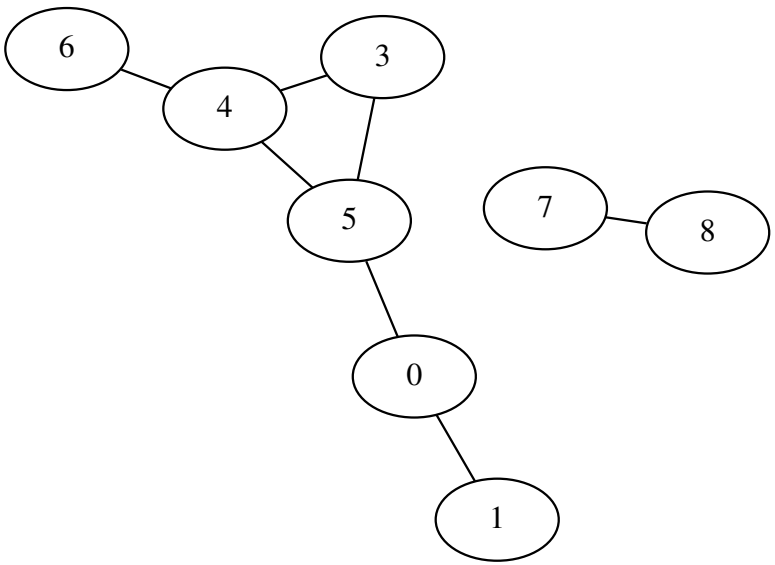
A partir da API básica é possível construir outras operações e aplicações completas:

```
1 /** Calcular o grau de um vértice */
2 public static int grau(Grafo G, int v) {
3   int grau = 0;
4   for (int w: G.adj(v))
5     grau++;
6   return grau;
7 }
```

```
1 /** Calcular o grau máximo de um grafo */
2 public static int grauMax(Grafo G) {
3   int max = 0;
4   for (int v = 0; v < G.V(); v++)
5     if (grau(G, v) > max)
6       max = grau(G, v);
7   return grau;
8 }
```

Implementação: representação por lista de arestas

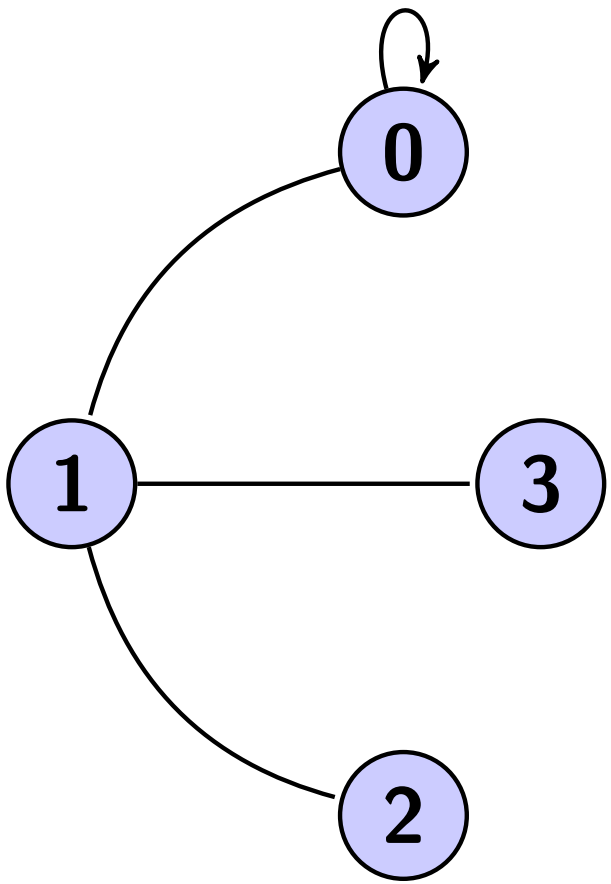
Manter lista de arestas (lista ligada ou vetor)



v	w
0	5
4	3
0	1
6	4
5	4
7	8
5	3

Implementação: representação por matriz de adjacências

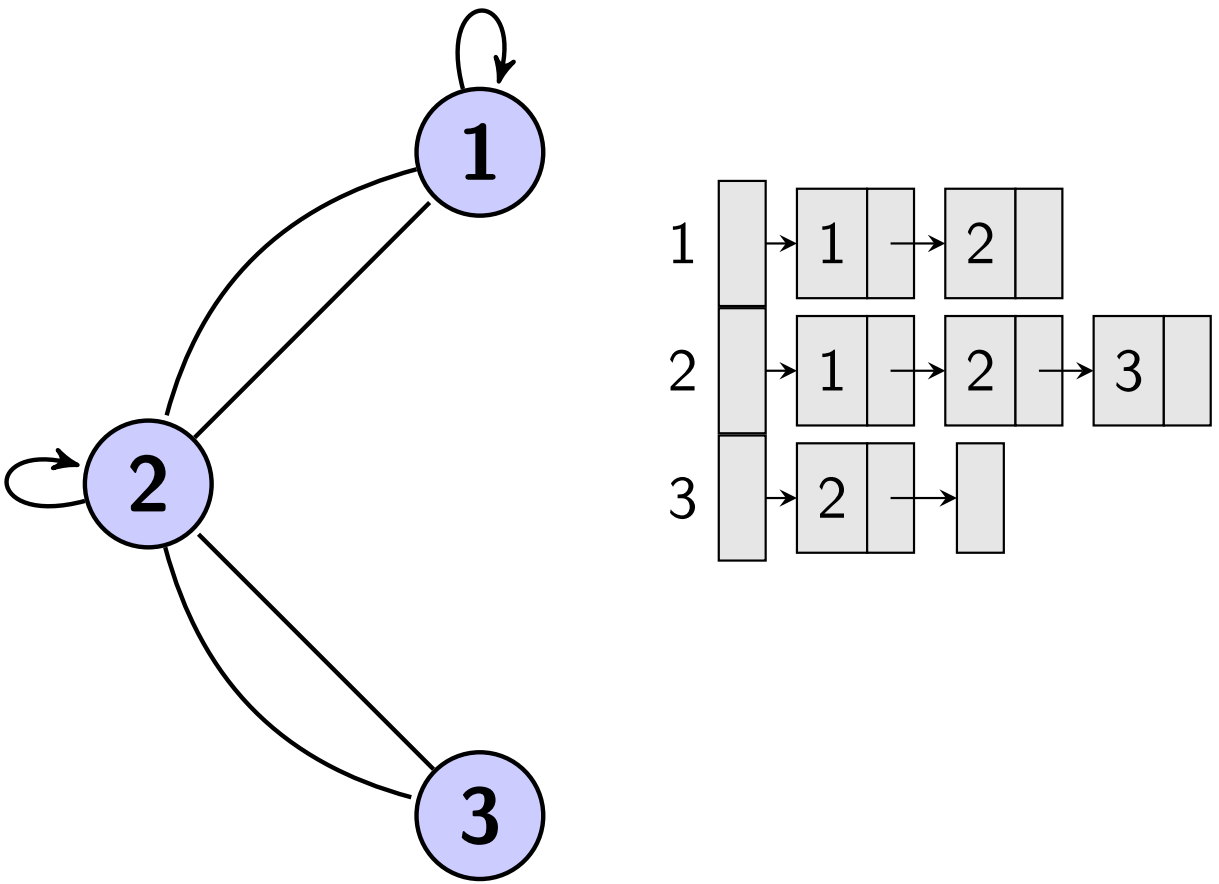
Manter matriz `adj[][]` com V linhas e V colunas. Se existe a aresta v-w, então a posição `adj[v][w] == adj[w][v] == 1`, senão é 0



	0	1	2	3
0	1	1	0	0
1	1	0	1	1
2	0	1	0	0
3	0	1	0	0

Implementação: representação por listas de adjacências

Manter um vetor de listas indexado por vértice.



```java
public class Grafo {
  int V;
  Lista<Integer>[] adj; // listas de adjacências

  public Grafo(int V) {
    this.V = V;
    adj = (Lista<Integer>[]) new Lista[V];
    for (int v = 0; v < V; v++)
        adj[v] = new Lista<Integer>();
  }

  public void novaAresta(int v, int w) {
      adj[v].add(w);
      adj[w].add(v);
  }

  public Iterable<Integer> adj(int v) {
      return adj[v];
  }
}
```

Estruturas de dados para grafos

Na prática: usar listas de adjacências

- Algoritmo em geral operam nos vértices
- Grafos reais tendem a ser esparsos (número grande de vértices, número pequeno de arestas por vértice)

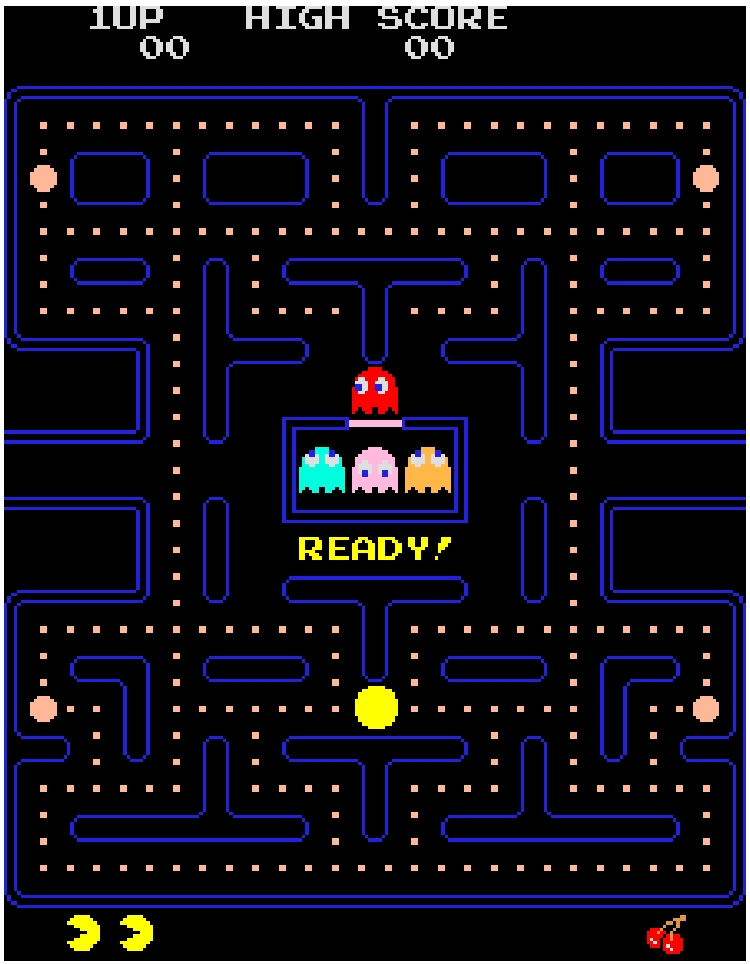
representação	espaço	criar aresta	existe v-w?	iterar adj(v)
lista de arestas	E	1	E	E
matriz de adjacências	V^2	1	1	V
listas de adjacências	E+V	1	grau(v)	grau(v)

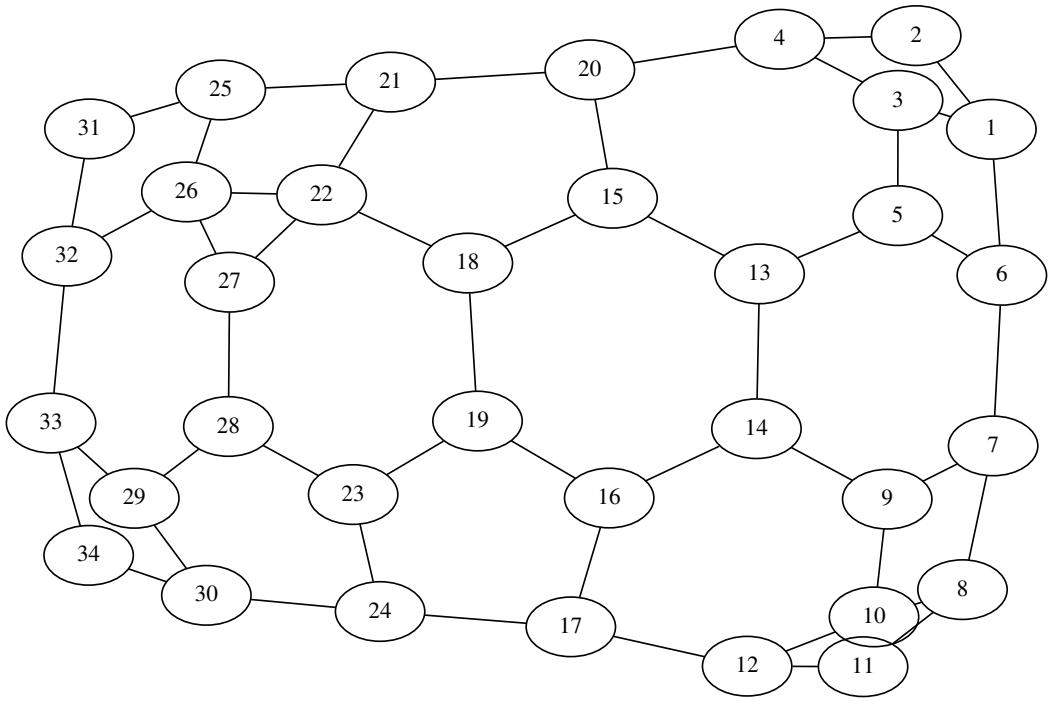
E = número de arestas
V = número de vértices

Exemplo: labirinto

Labirinto

- Vértice = intersecção
- Aresta = corredor





Algoritmo de Trémaux para exploração de labirintos

Algoritmo

- Desenrolar um novelo de fios por onde andar
- Marcar cada intersecção visitada e cada corredor passado
- Refazer passos quando estiver sem opções não visitadas

- Primeiro uso: talvez pelo Teseu para matar o Minotauro no labirinto, conforme sugerido pela donzela Ariadne.

Busca em profundidade

A busca em profundidade, (em inglês, *depth-first search*)

é um algoritmo para caminhar no grafo

Estratégia

- sempre buscar primeiro o vértice mais profundo no grafo
- após todas as arestas adjacentes forem exploradas, a busca retrocede no grafo para explorar vértices anteriores

Busca em profundidade

- O algoritmo é a base para tarefas importantes
  - verificação de grafos acíclicos
  - ordenação topológica
  - detecção de componentes fortemente conectados.

Busca em profundidade

Algoritmo: iniciar em um vértice arbitrário

1. Visita vértice
2. Se vértice for o buscado, retorna que encontrou
3. Senão, faz busca em profundidade para cada vértice adjacente ainda não visitado

Como implementar?

Desacoplamento

É possível fazer separação do Grafo da operação: desacoplamento.

Padrão de projeto para desacoplar o Grafo de suas operações

- Criar um objeto Grafo
- Passar o Grafo para uma rotina de processamento de uma operação
- Consultar a rotina por informações relacionadas à operação

public class	Caminhos	
	Caminhos(Grafo G, int v)	processar grafo a partir do vértice v
boolean	temLigacao(int w)	existe caminho entre v e w?
Iterable<Integer>	caminhoPara(int w)	obter caminho de v para w

API de caminhos

```
1  class Caminhos {
2    Grafo G; int origem;
3    boolean marcado[];//indica se foi visitado
4    int veioDe[];//indica por onde foi visitado
5
6    public Caminhos(Grafo G, int origem) {
7      this.origem = origem; this.G = G;
8      veioDe = new int[G.V()];
9      marcado = new boolean[G.V()];
10
11     dfs(G, origem);
12   }
13
14   //Obter o caminho pelo metodo de profundidade.
15   boolean dfs(Grafo G, int atual) {...}
16   //Vértice tem ligação à origem?
17   boolean temLigacao(int w) {...}
18   //qual é o caminho da origem para um vértice?
19   Iterable<Integer> caminhoPara(int w) {...}
20 }
```

Busca em profundidade

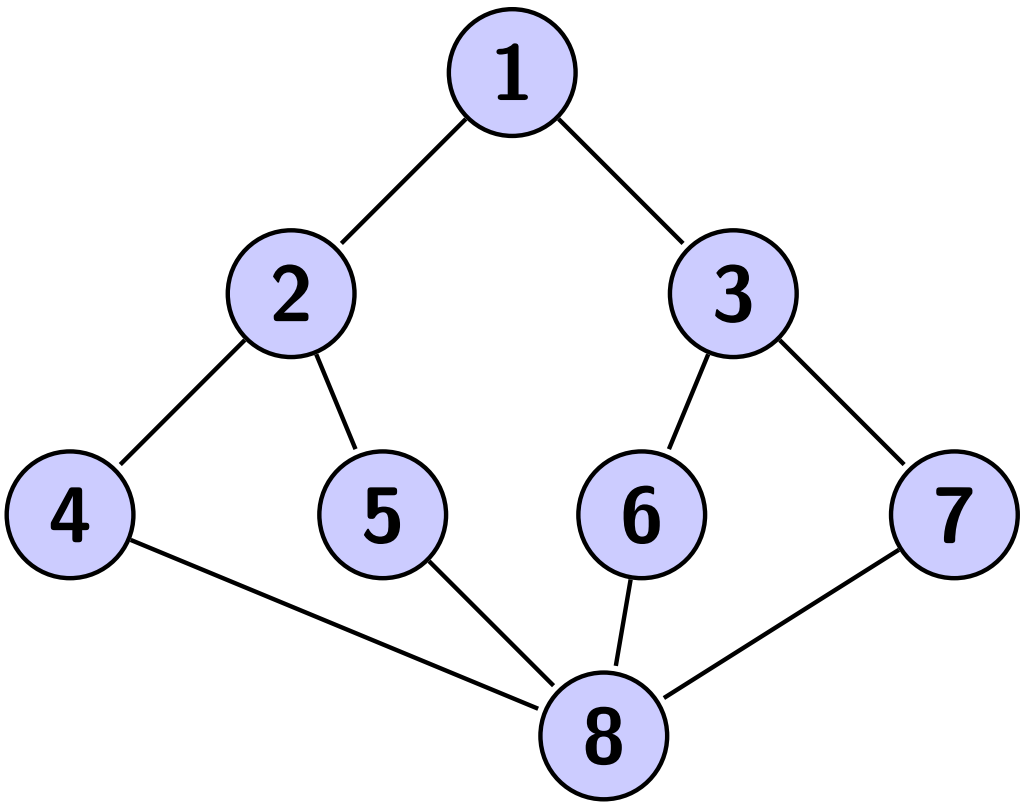
```
1  void dfs(Grafo G, int atual) {
2
3   // visita o vertice atual
4   marcado[atual] = true;
5   for (int adjacente: G.adj(atual)) {
6     if (marcado[adjacente] == false) {
7       veioDe[adjacente] = atual;
8       dfs(G, adjacente);
9     }
10  }
11 }
```

- Usar um array com a informação de "marcado"
- Problema: pode exceder o limite da pilha de recursão

Busca em profundidade: não recursivo

```
1  void dfs(Grafo G, int origem) {
2    Stack pilha = new Stack();// pilha substitui recursão
3    pilha.push(origem);
4
5    while (pilha.empty() == false) {
6      int atual = (int) pilha.pop();//pega mais recente
7      if (marcado[atual] == false) {
8          marcado[atual] = true;
9          for (int adjacente: G.adj(atual)) {
10             veioDe[adjacente] = atual; //marca caminho
11             pilha.push(adjacente);
12         }
13     }
14   }
15 }
```

Exemplo de caminho em profundidade

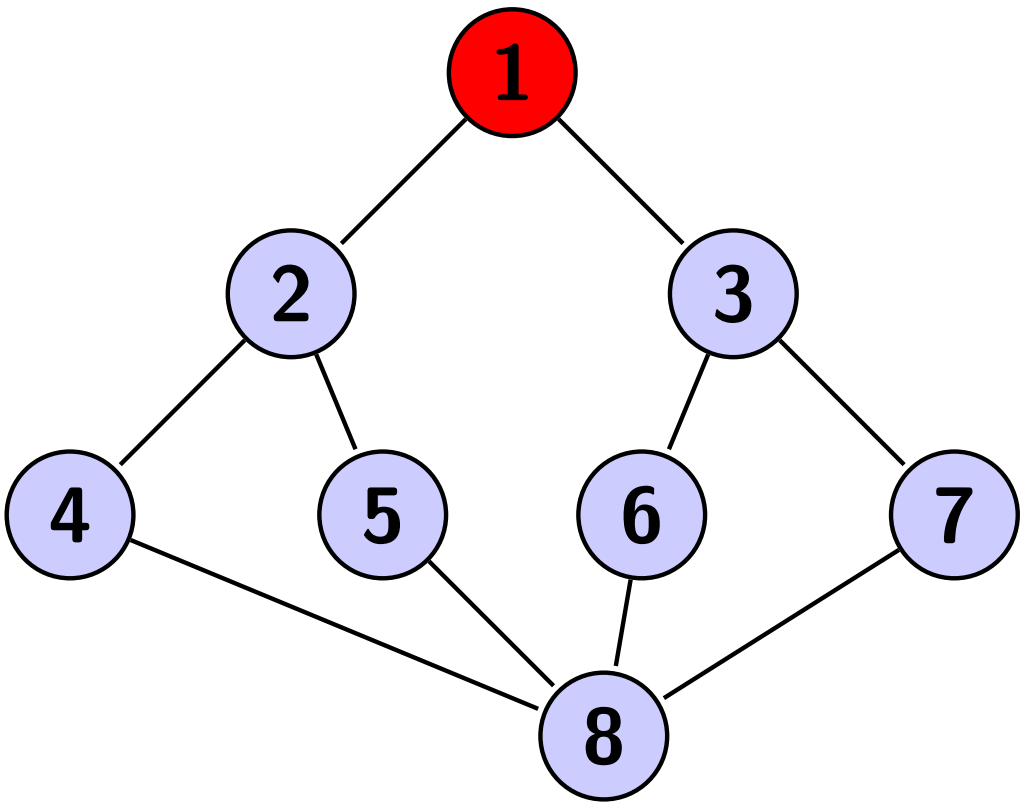


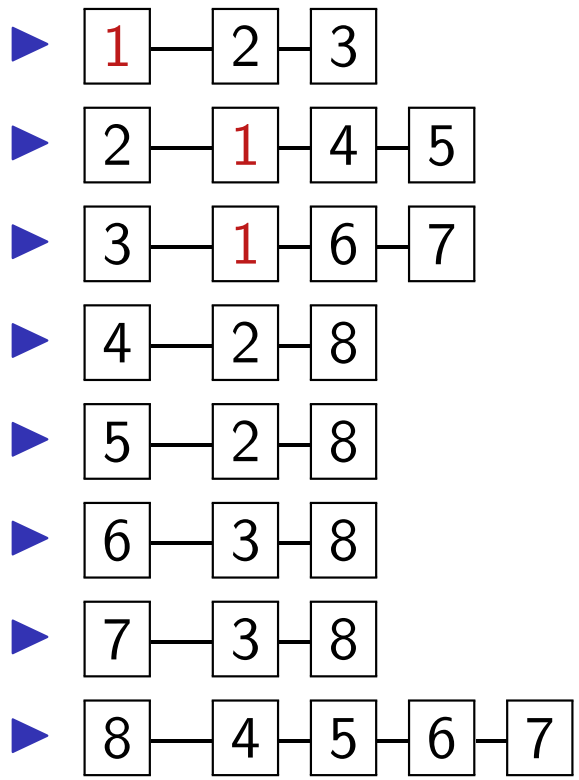
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Exemplo de caminho em profundidade

Início com 1.

A cor verde significa que vértice foi marcado.



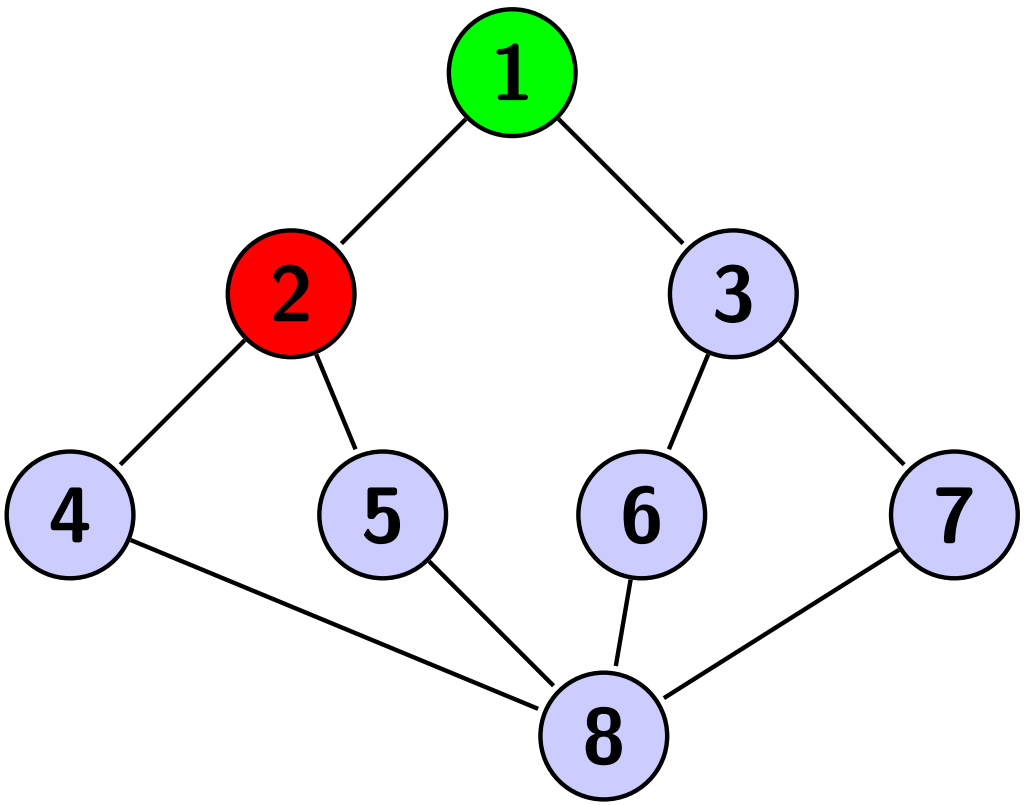


Qual é o próximo vértice a ser visitado?

Exemplo de caminho em profundidade

Visitando o 2.

A cor verde significa que vértice foi visitado.



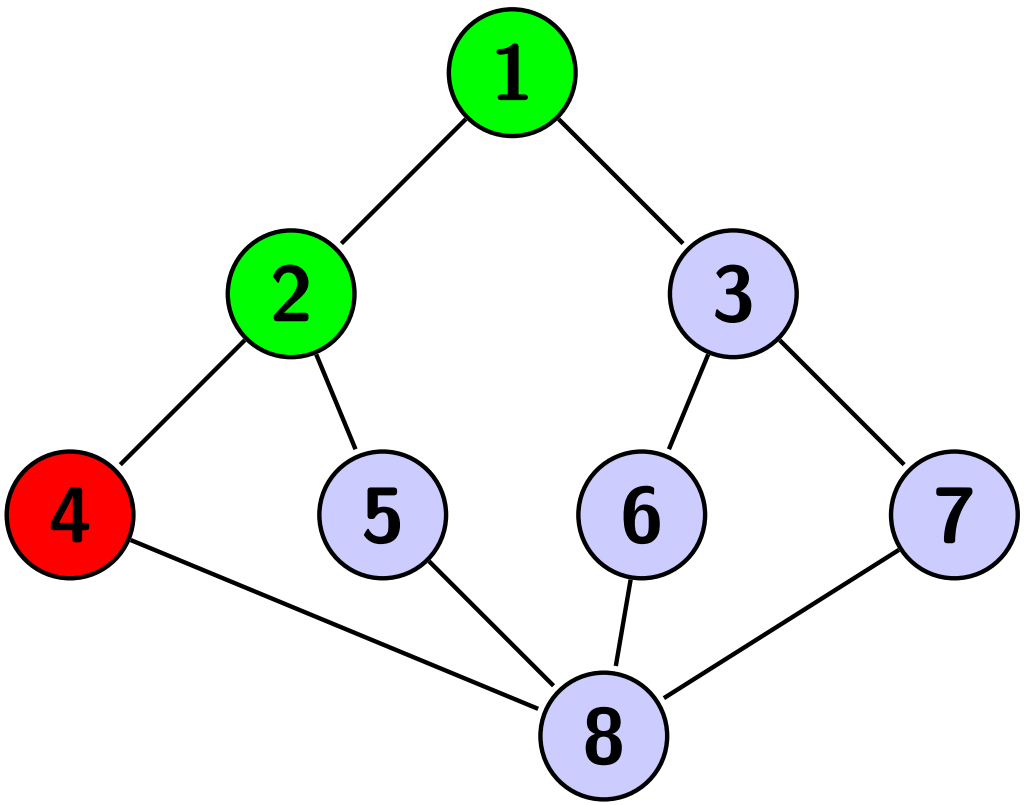
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 4.

A cor verde significa que vértice foi visitado.



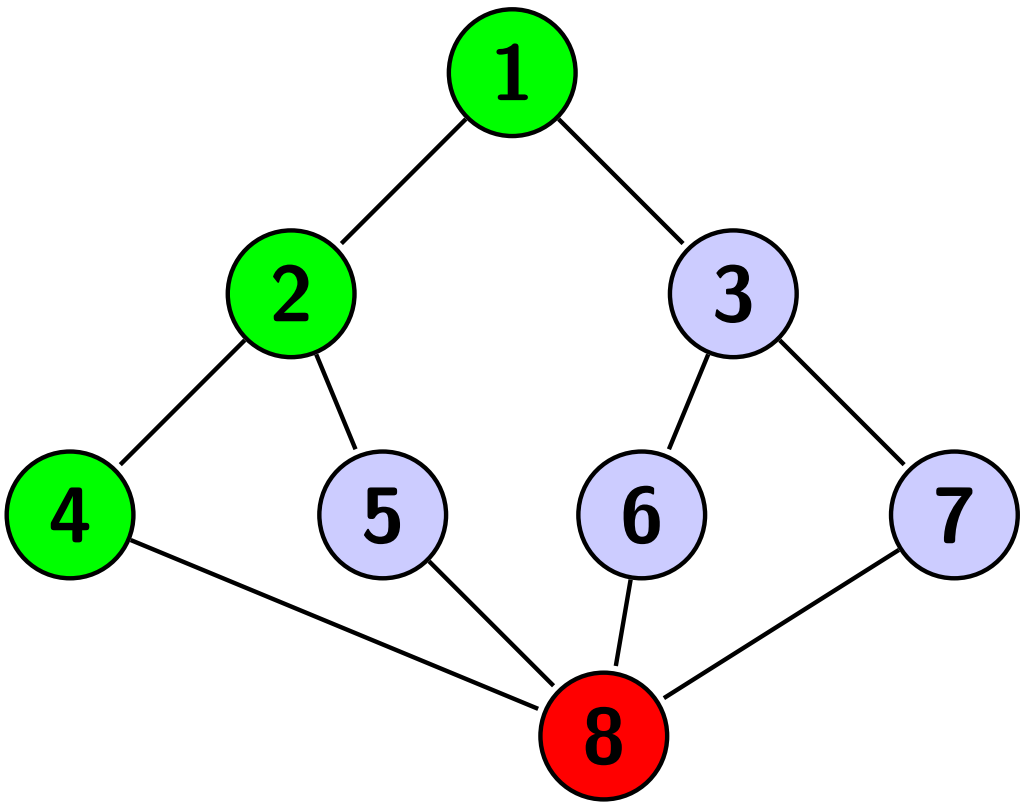
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 8.

A cor verde significa que vértice foi visitado.



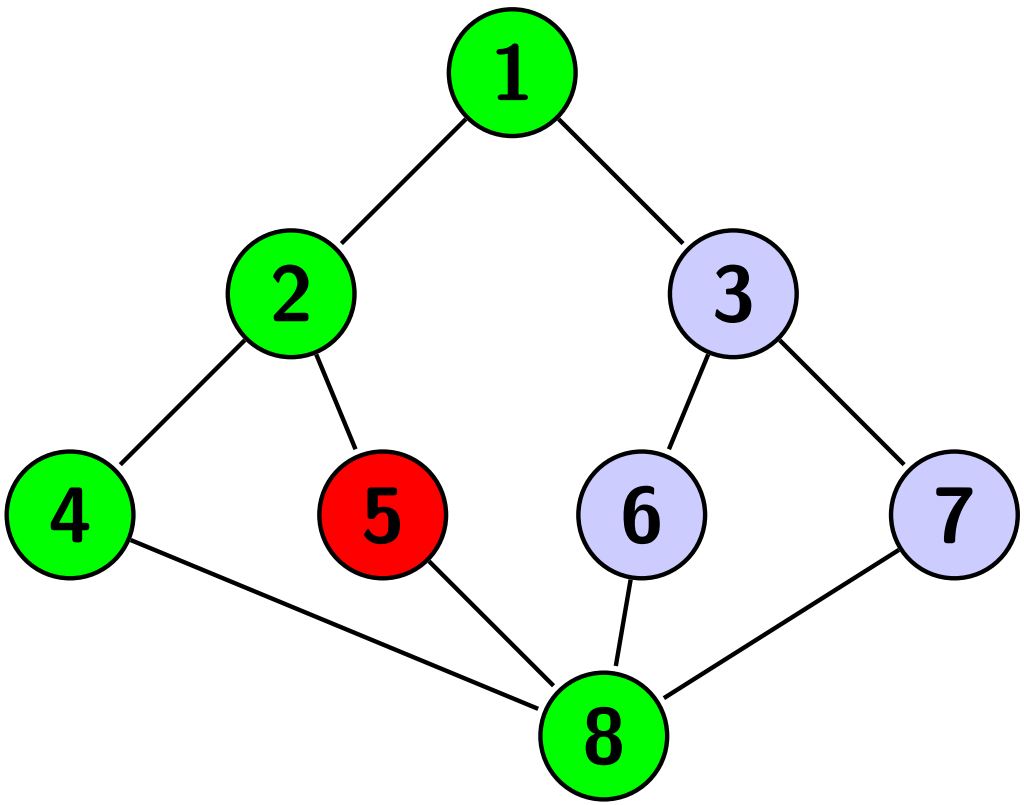
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 5.

A cor verde significa que vértice foi visitado.



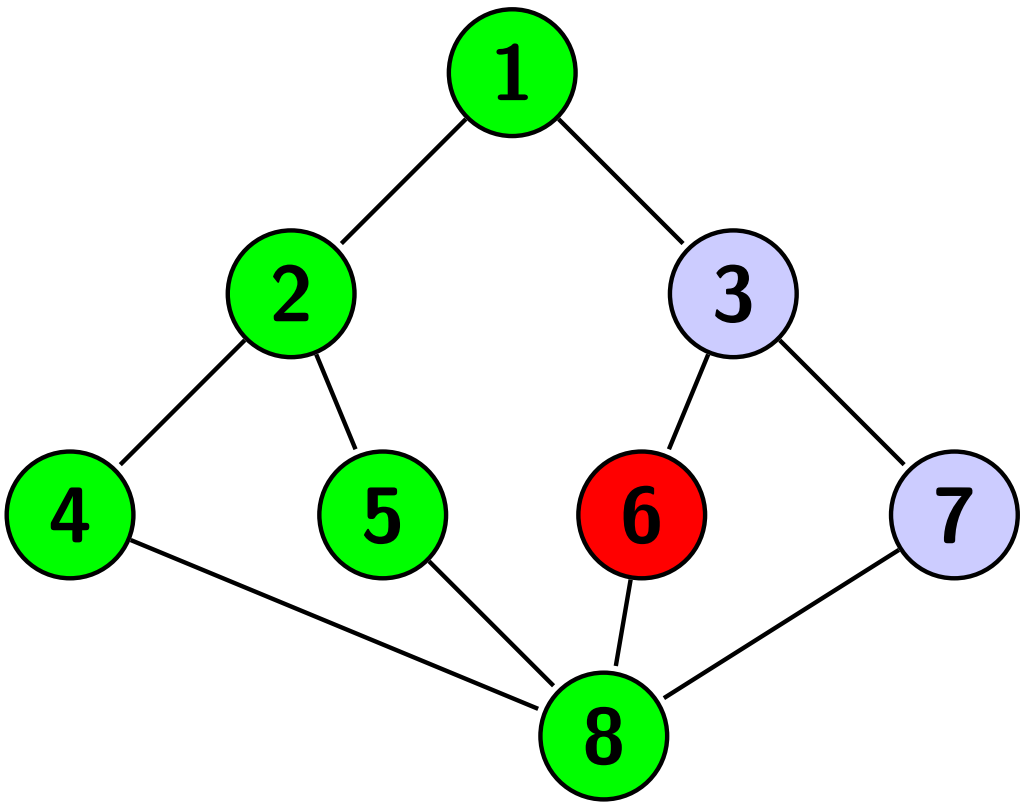
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 6.

A cor verde significa que vértice foi visitado.



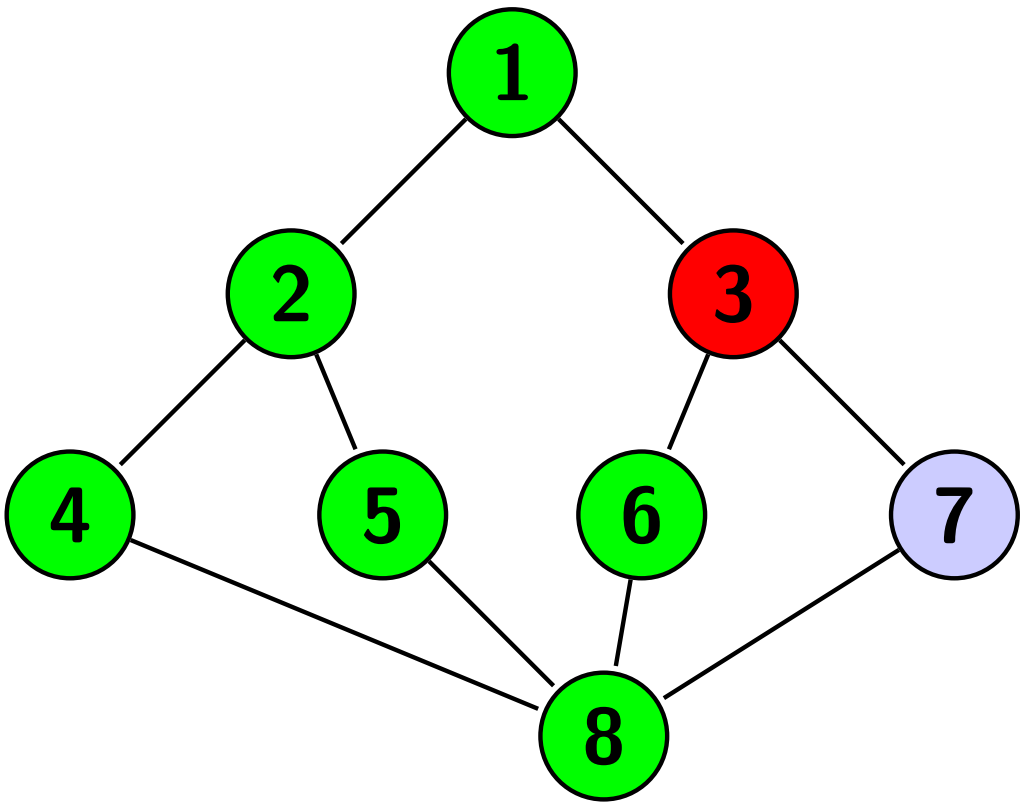
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

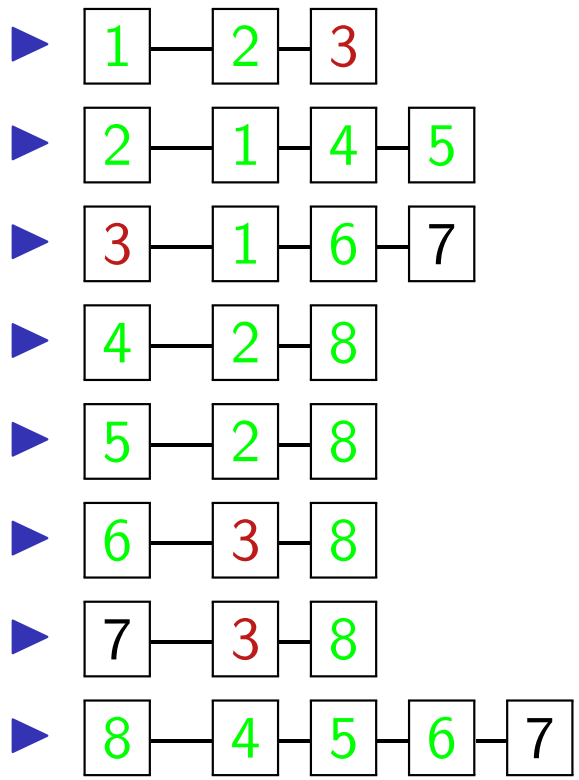
Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 3.

A cor verde significa que vértice foi visitado.



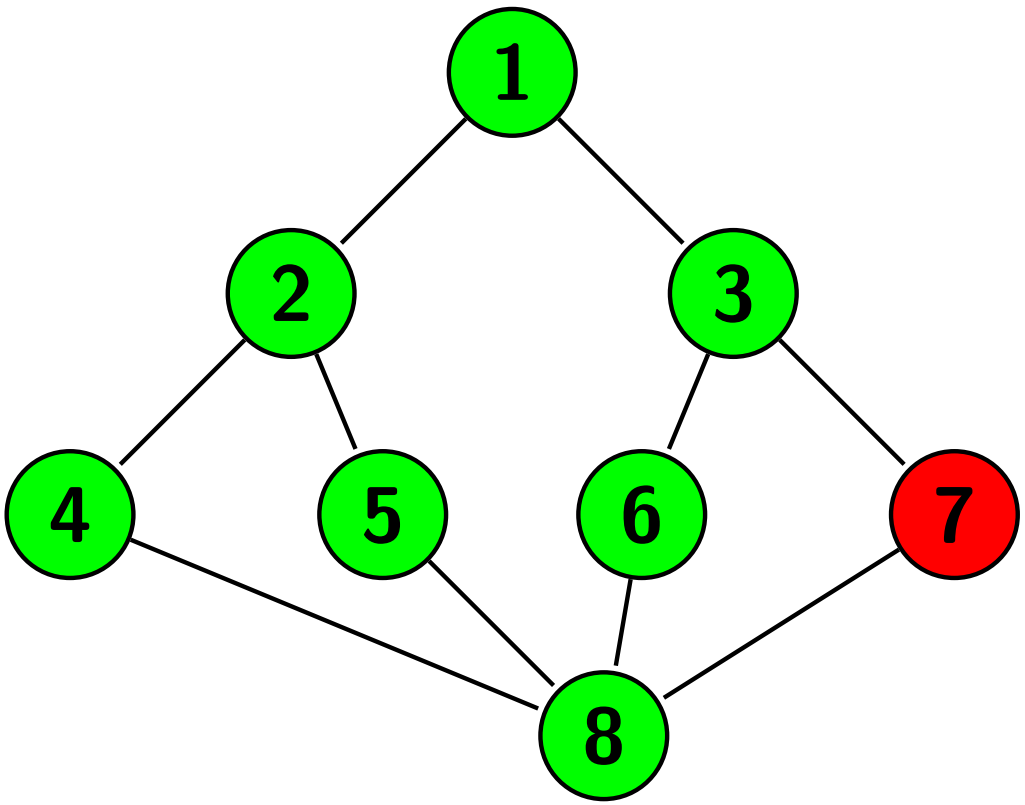


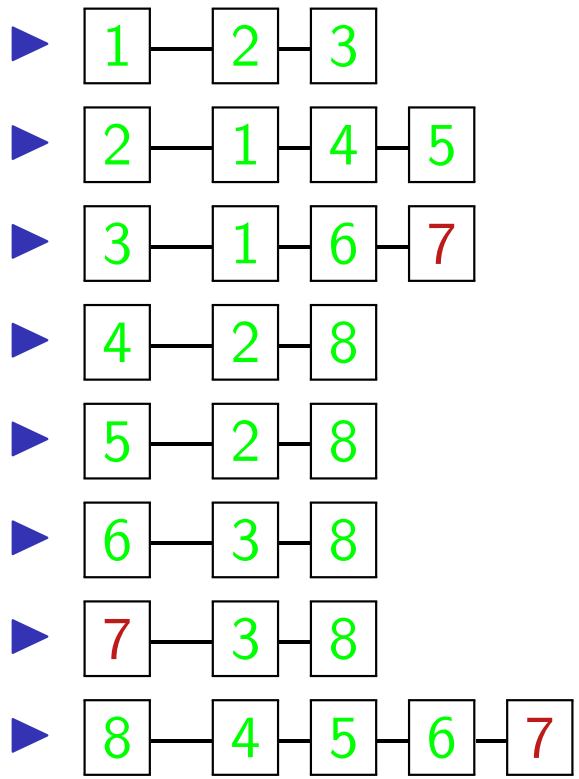
Qual é o próximo?

Exemplo de caminho em profundidade

Visitando o 7.

A cor verde significa que vértice foi visitado.





Terminou.

Análise da busca em profundidade

- Qual é o pior caso?
  - Custo de ir para cada vértice é proporcional a $|V|$
  - Custo de transitar em cada aresta é proporcional $|A|$
  - Complexidade de pior caso $O(|V| + |A|)$

Operações da API

```
1  boolean temLigacao(int w) {
2    return marcado[w];
3  }
```

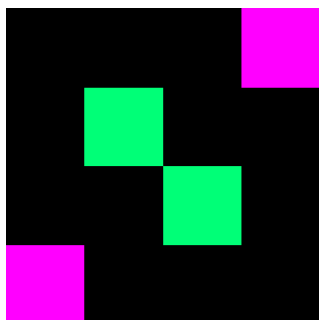
```
1  Iterable<Integer> caminhoPara(int w) {
2    if (temLigacao(w) == false) return null;
3
4    Stack<Integer> caminho = new Stack<Integer>();
5
6    for (int x = w; x != origem; x = veioDe[x])
7      caminho.push(x); // empilha da onde veio
8
9    caminho.push(origem);
10
11   return caminho;
12 }
```

Aplicação: ferramenta "balde" de programas de imagens

Formato PPM ASCII:

- linha 1: código do formato do arquivo "P3"
- linha 2: um exemplo de comentário. inicia-se com #
- linha 3: dimensões da imagem (horizontal vertical)
- linha 4: valor máximo de cor na imagem
- linhas seguintes:
  - cores: (vermelho verde azul) de cada ponto, 0 é preto
  - linhas ou espaços extras não interferem

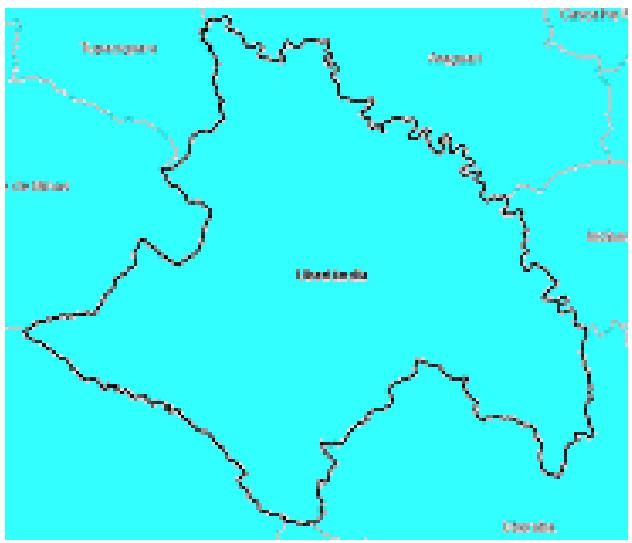
```
P3
# feep.ppm
4 4
15
0  0  0    0   0  0  0   0  0   15  0  15
0  0  0    0  15  7  0   0  0   0  0   0
0  0  0    0   0  0  0  15  7   0  0   0
15  0  15  0   0  0  0   0  0   0  0   0
```



Aplicação: ferramenta "balde" de programas de imagens

Ideia

- Montar grafo com pontos vizinhos similares
  - exemplo: pontos vizinhos são similares se suas cores diferem no máximo 10%
- Obter caminho com busca em profundidade para o ponto onde o balde foi jogado
- Pintar todos os pontos conectados com o ponto do balde



Busca em largura

Estratégia

- avança por fronteiras
- marca (ou visita) todos os adjacentes ainda não marcados primeiro

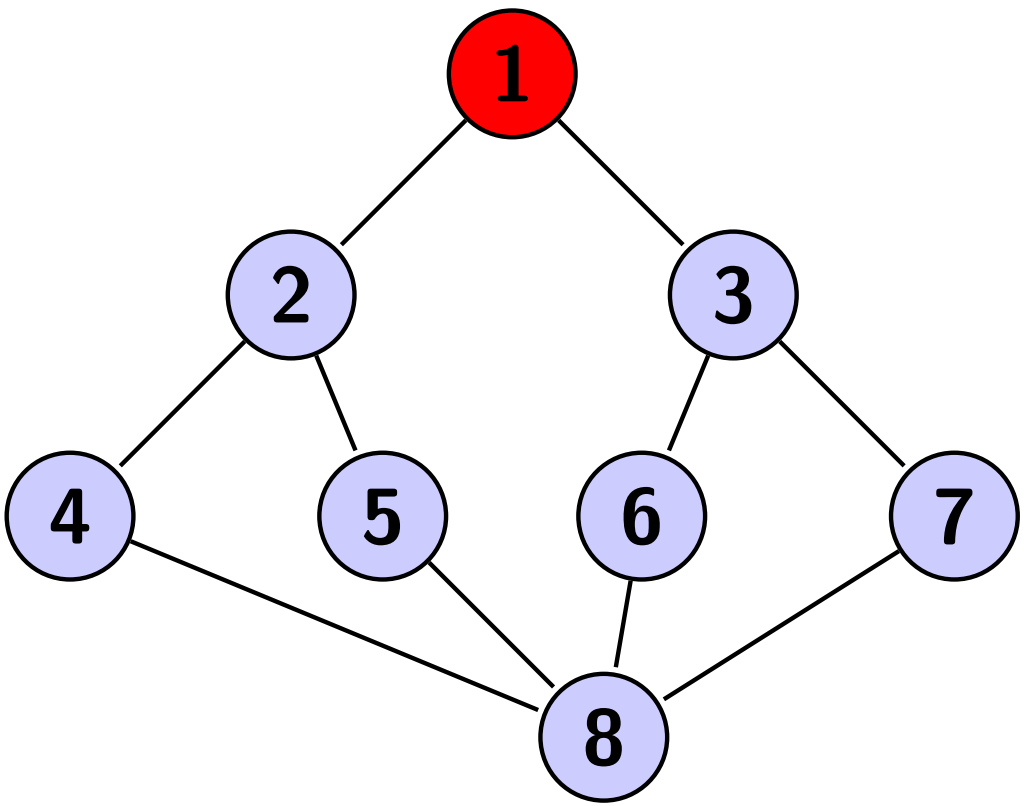
Algoritmo

Iniciar em um vértice arbitrário u

1. colocar u em uma fila F
2. Pegar e remover o primeiro elemento u da fila F
3. Para cada vértice adjacente que ainda não visitado v ao u
   - 3.1 visitar vértice adjacente v
   - 3.2 colocar vértice v na fila F

Exemplo de caminho em largura

Início com 1.
Fila [1]

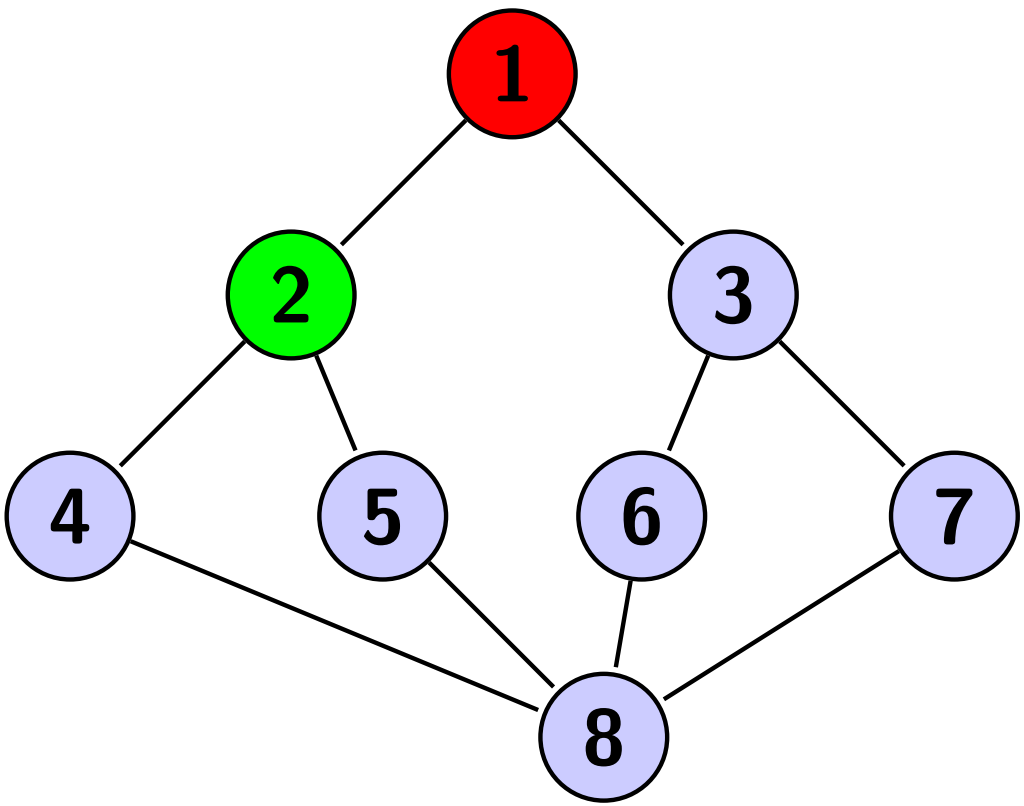


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em largura

Primeiro da fila é 1.
Fila 2

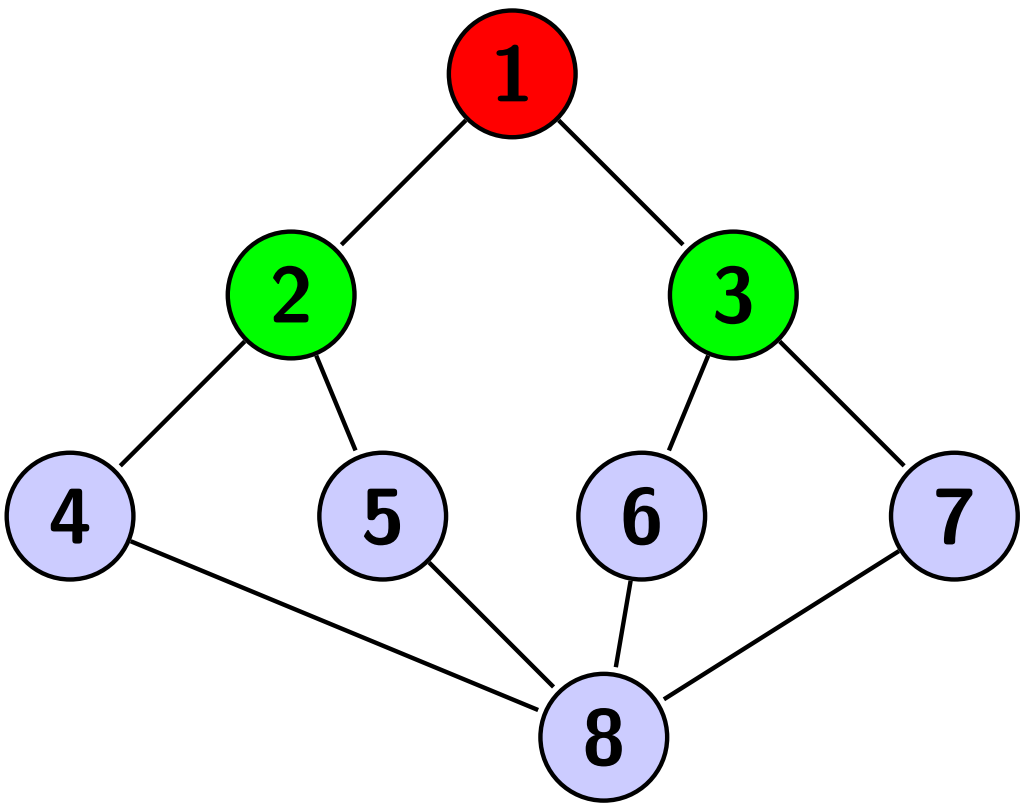


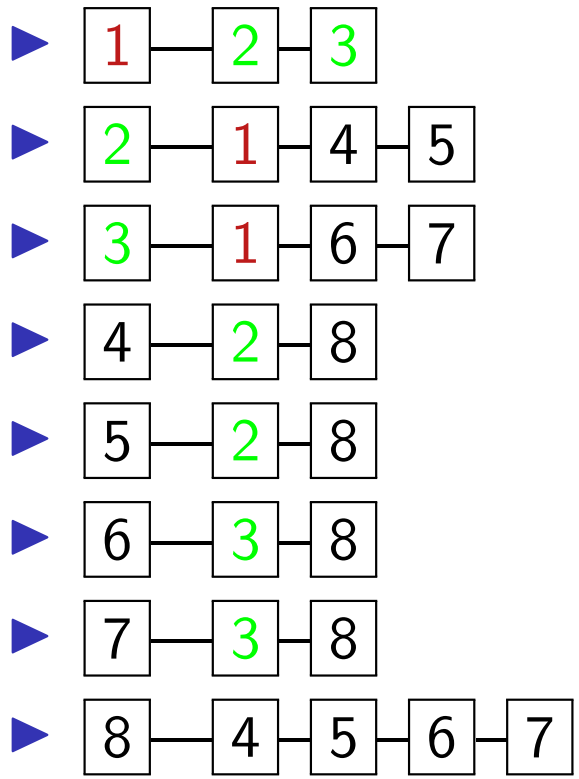
- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo?

Exemplo de caminho em largura

Primeiro da fila é 1.
Fila 2 3.

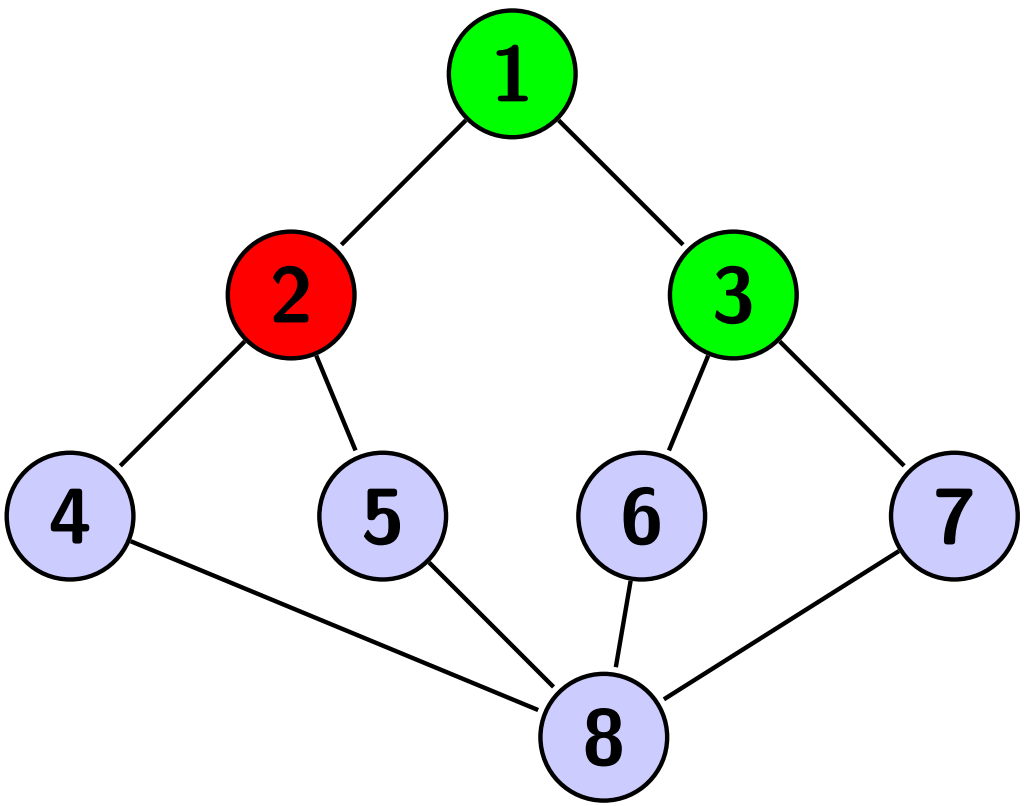




Qual é o próximo? Pega próximo da fila: 2

Exemplo de caminho em largura

Primeiro da fila é 2.
Fila [3].

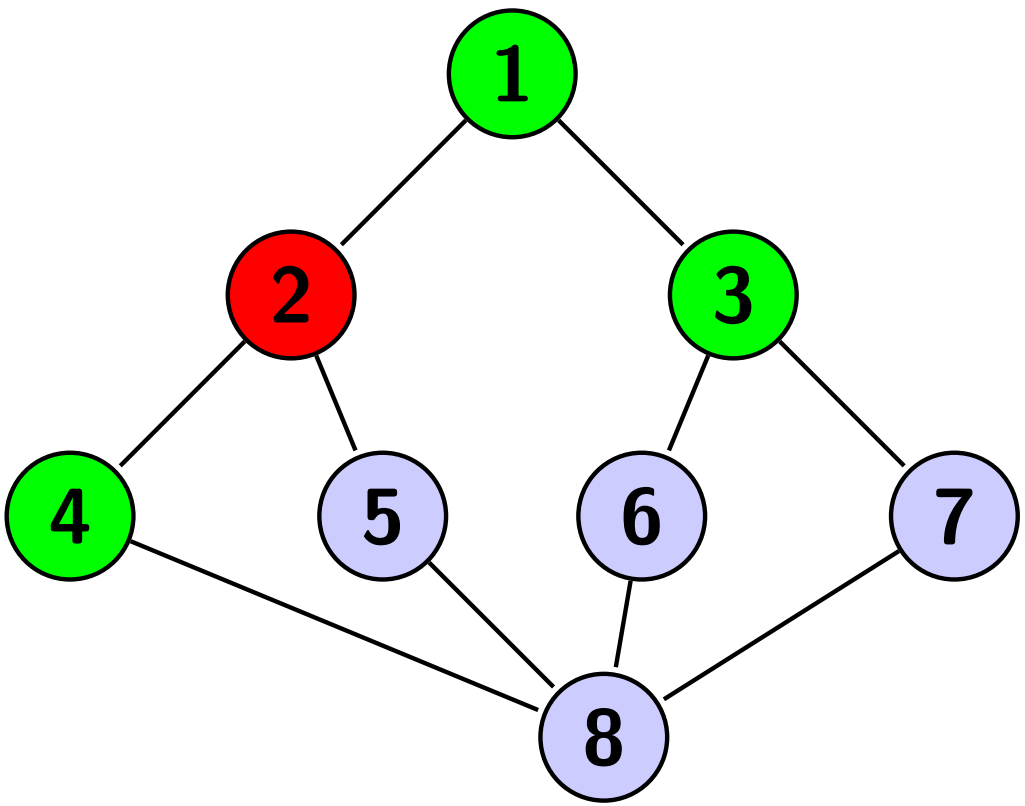


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Adjacente não visitado.

Exemplo de caminho em largura

Primeiro da fila é 2.
Fila de [2] é [4].

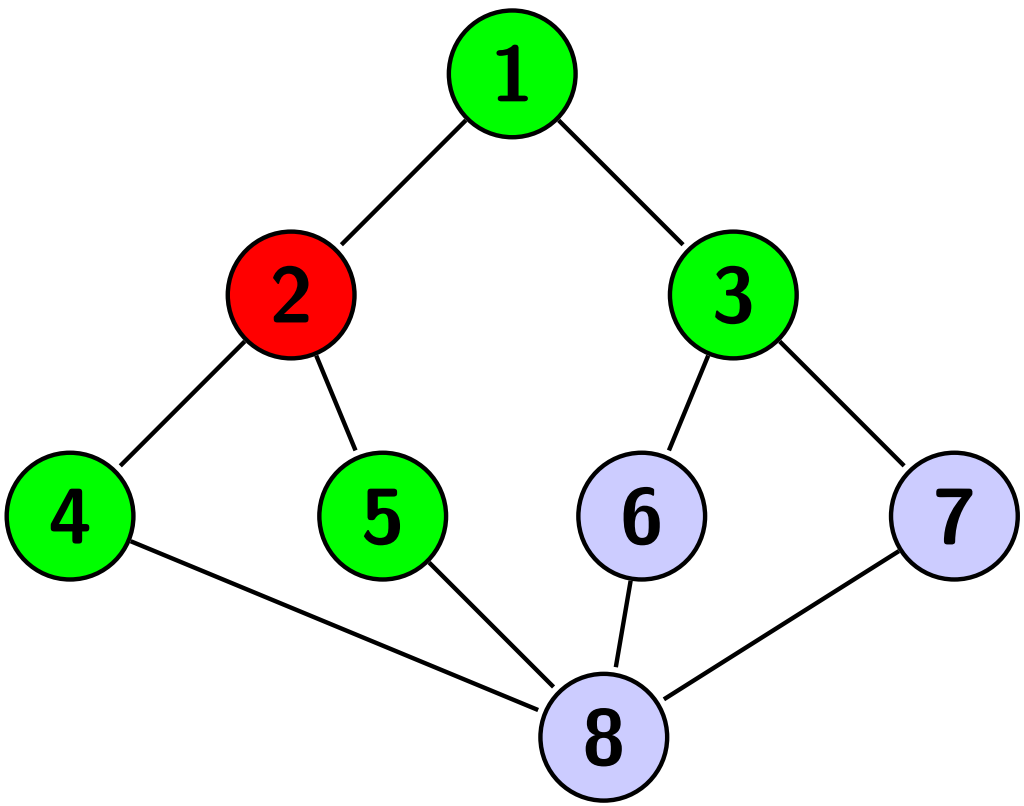


- [1] — [2] — [3]
- [2] — [1] — [4] — [5]
- [3] — [1] — [6] — [7]
- [4] — [2] — [8]
- [5] — [2] — [8]
- [6] — [3] — [8]
- [7] — [3] — [8]
- [8] — [4] — [5] — [6] — [7]

Qual é o próximo? Adjacente não visitado.

Exemplo de caminho em largura

Primeiro da fila é 2.
Fila 3 4 5.

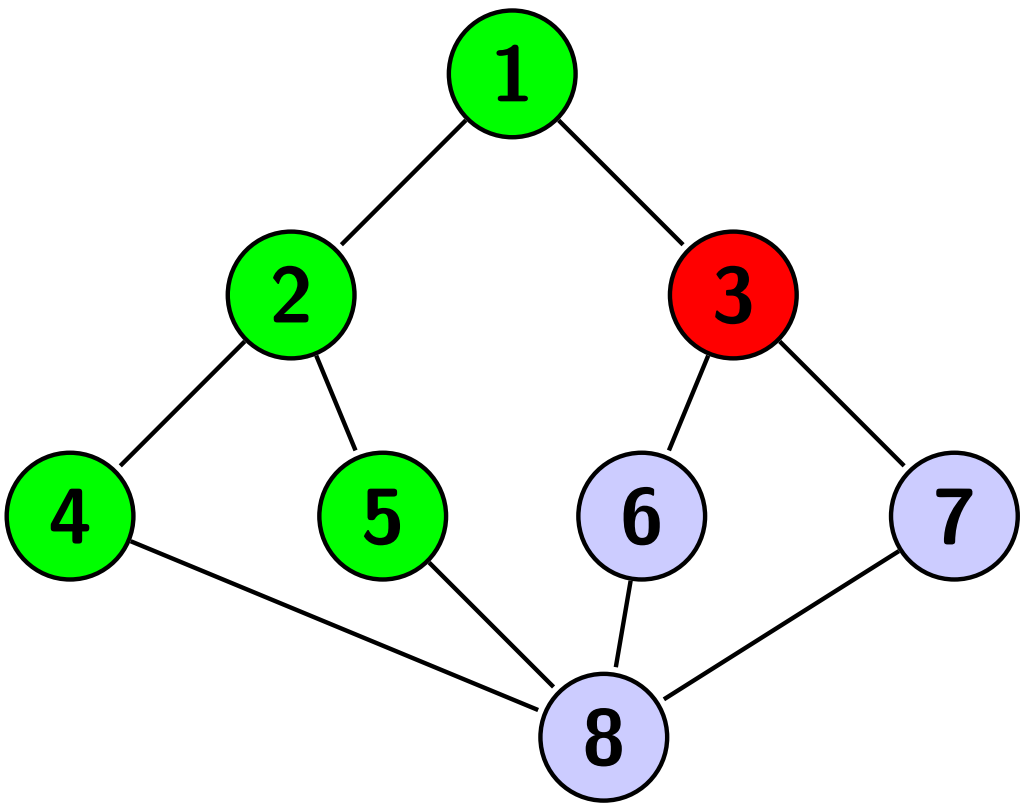


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Acabaram os adjacentes, pegar próximo da fila.

Exemplo de caminho em largura

Primeiro da fila é 3.
Fila 4 5.

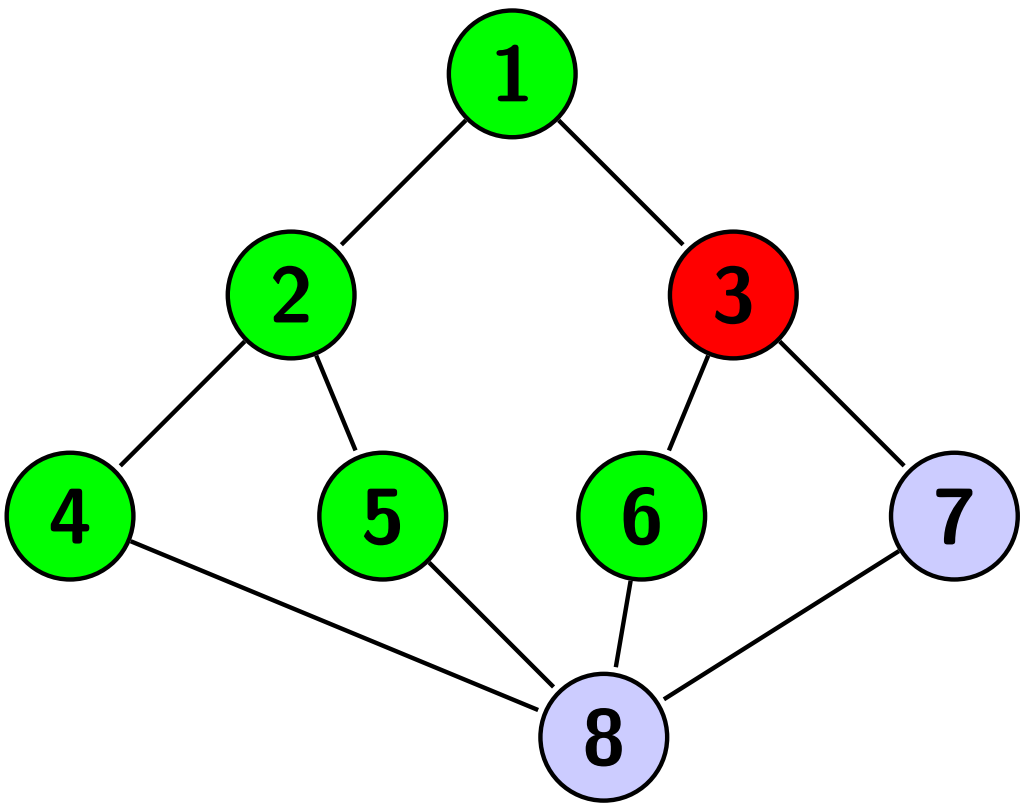


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Adjacente.

Exemplo de caminho em largura

Primeiro da fila é 3.
Fila | 4 | 5 | 6 |.

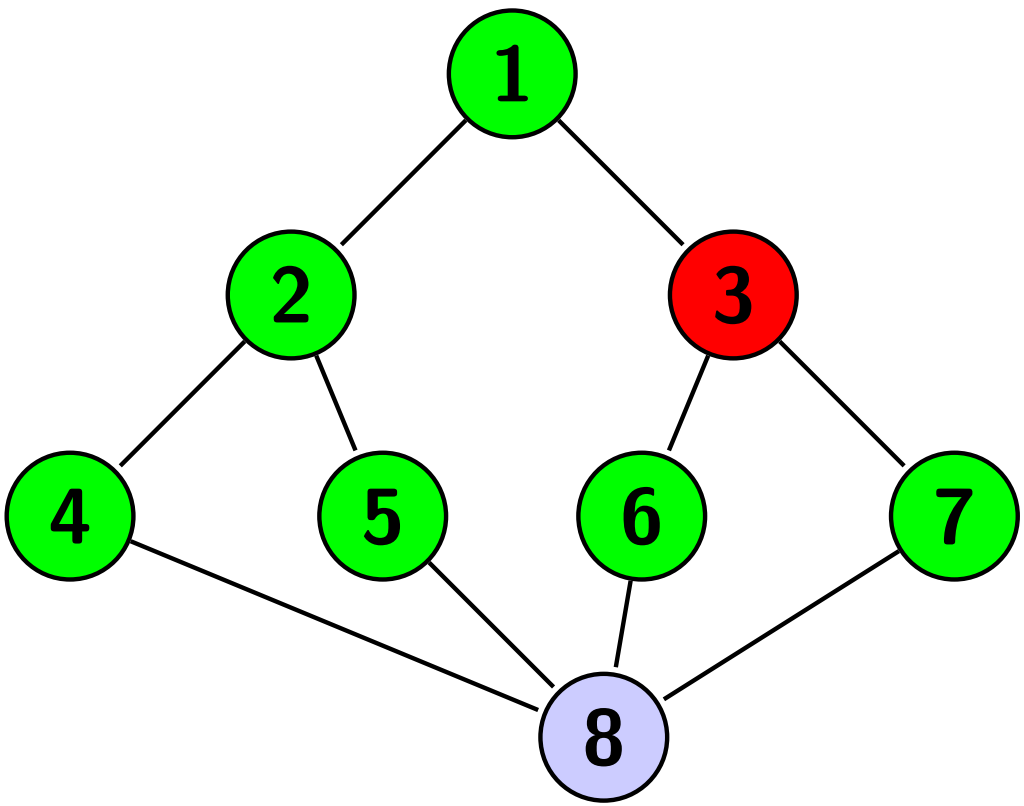


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Adjacente não visitado.

Exemplo de caminho em largura

Primeiro da fila é 3.
Fila | 4 | 5 | 6 | 7 |.

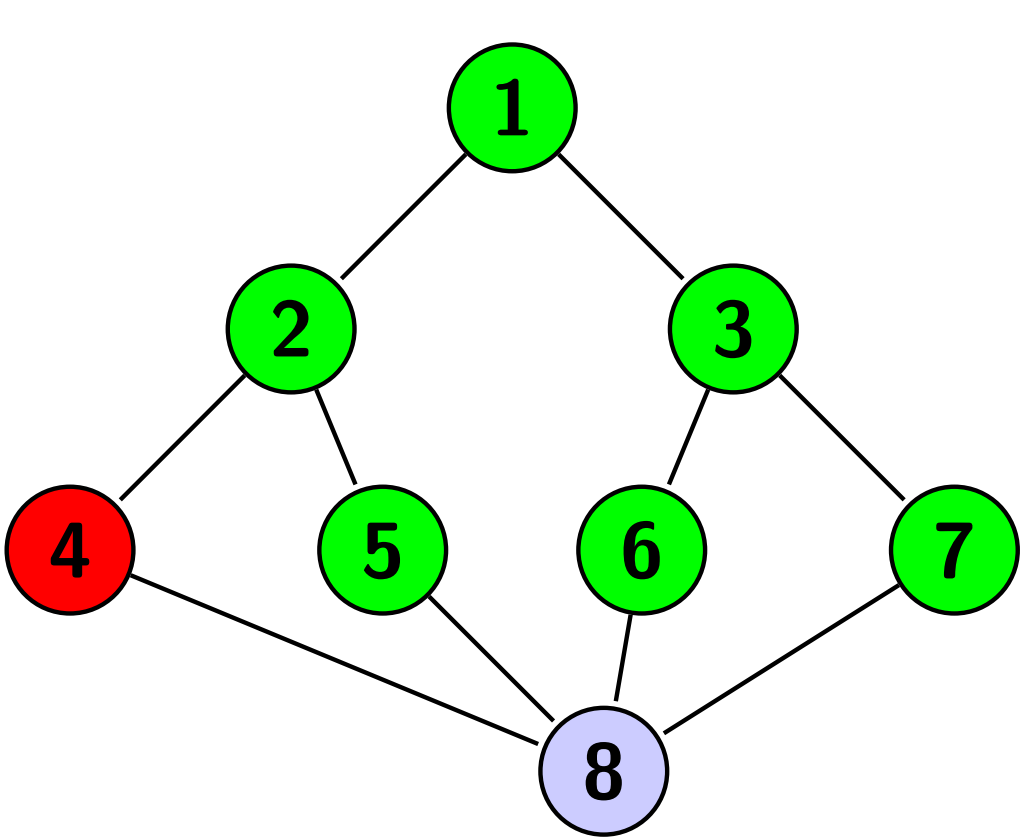


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Acabaram os adjacentes, pegar próximo da fila

Exemplo de caminho em largura

Primeiro da fila é 4.
Fila 5 6 7.

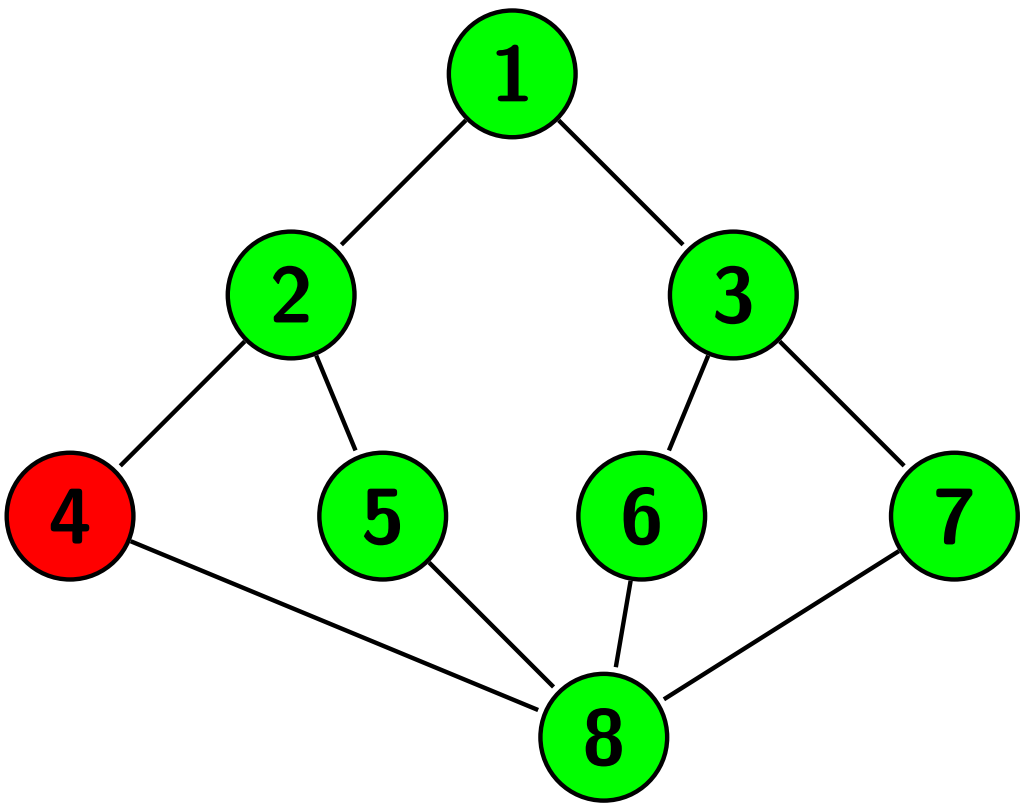


- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Qual é o próximo? Adjacente não visitado.

Exemplo de caminho em largura

Primeiro da fila é 4.
Fila 5 6 7.



- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

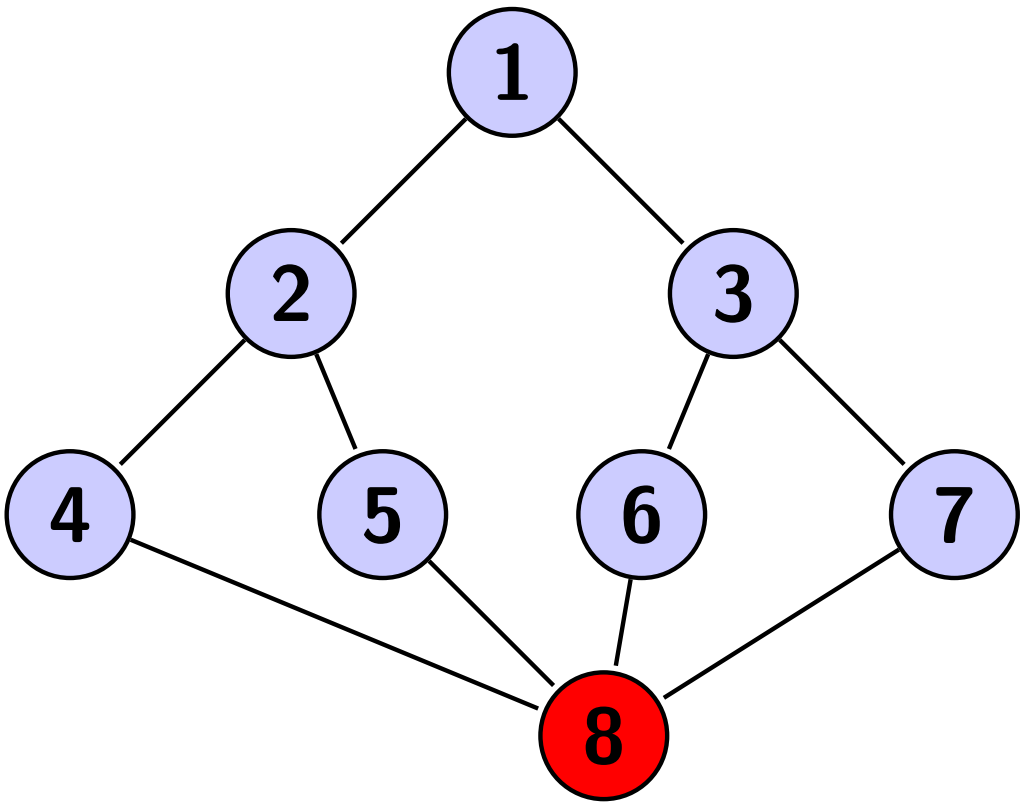
Qual é o próximo? Esvazia a fila.

Análise da busca em largura

- Qual é o pior caso?
  - Custo de inserção e remoção em fila é constante
  - Todos os vértices são enfilerados e removidos uma vez $O(|V|)$
  - Todas as arestas são utilizadas $|A|$
  - Complexidade de pior caso $O(|V| + |A|)$

Exercício de caminhos

Qual são os caminhos em profundidade e largura quando começarmos com vértice 8.



- 1 — 2 — 3
- 2 — 1 — 4 — 5
- 3 — 1 — 6 — 7
- 4 — 2 — 8
- 5 — 2 — 8
- 6 — 3 — 8
- 7 — 3 — 8
- 8 — 4 — 5 — 6 — 7

Consultas sobre conectividade

Definição

Vértices `v` e `w` são conectados se existe um caminho entre eles.

Meta

Pré-processar grafo para responder consultas da forma v é conectado a w? em tempo constante.

API de Componentes Conectados

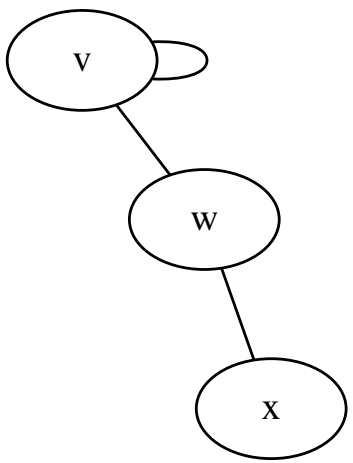
public class	CompConectado	
	CompConectado(Grafo G)	processa componentes conectados em G
boolean	conectado(int v, int w)	v e w são conectados?
int	contar()	número de componentes conectados
int	id(int v)	identificador do componente de v

Busca em profundidade

Relação de conectividade

A relação "é conectado a" é uma relação de equivalência

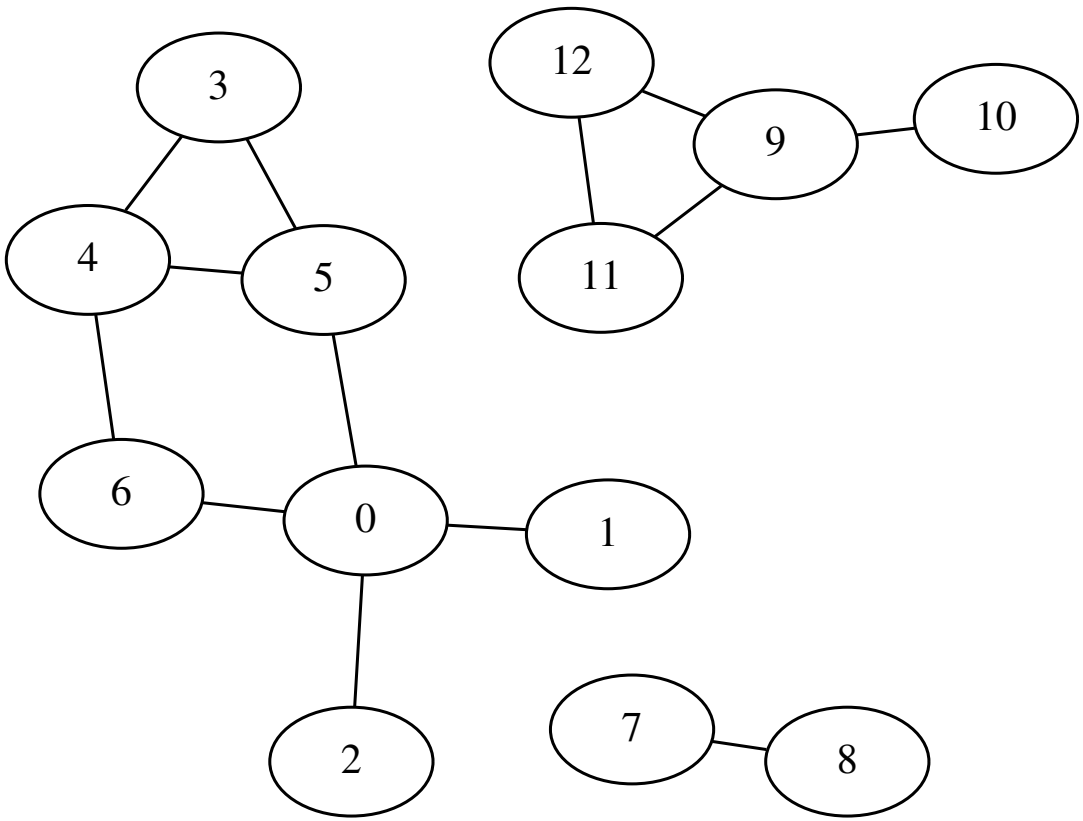
- Reflexivo: v é conectado a v
- Simétrico: se v é conectado a w, então w é conectado a v
- Transitivo: se v é conectado a w e w conectado a x, então v é conectado a x



Definição

Um componente conectado é um conjunto maximal de vértices conectados

Quantos componentes conectados neste grafo?



v	id[v]
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	2
10	2
11	2
12	2

Implementação

Meta

Particionar vértices em componentes conectados

Algoritmo

- Inicializar todos vértices v como não marcados
- Para cada vértice v não marcado, rodar busca em profundidade para identificar todos os vértices descobertos como parte do mesmo componente

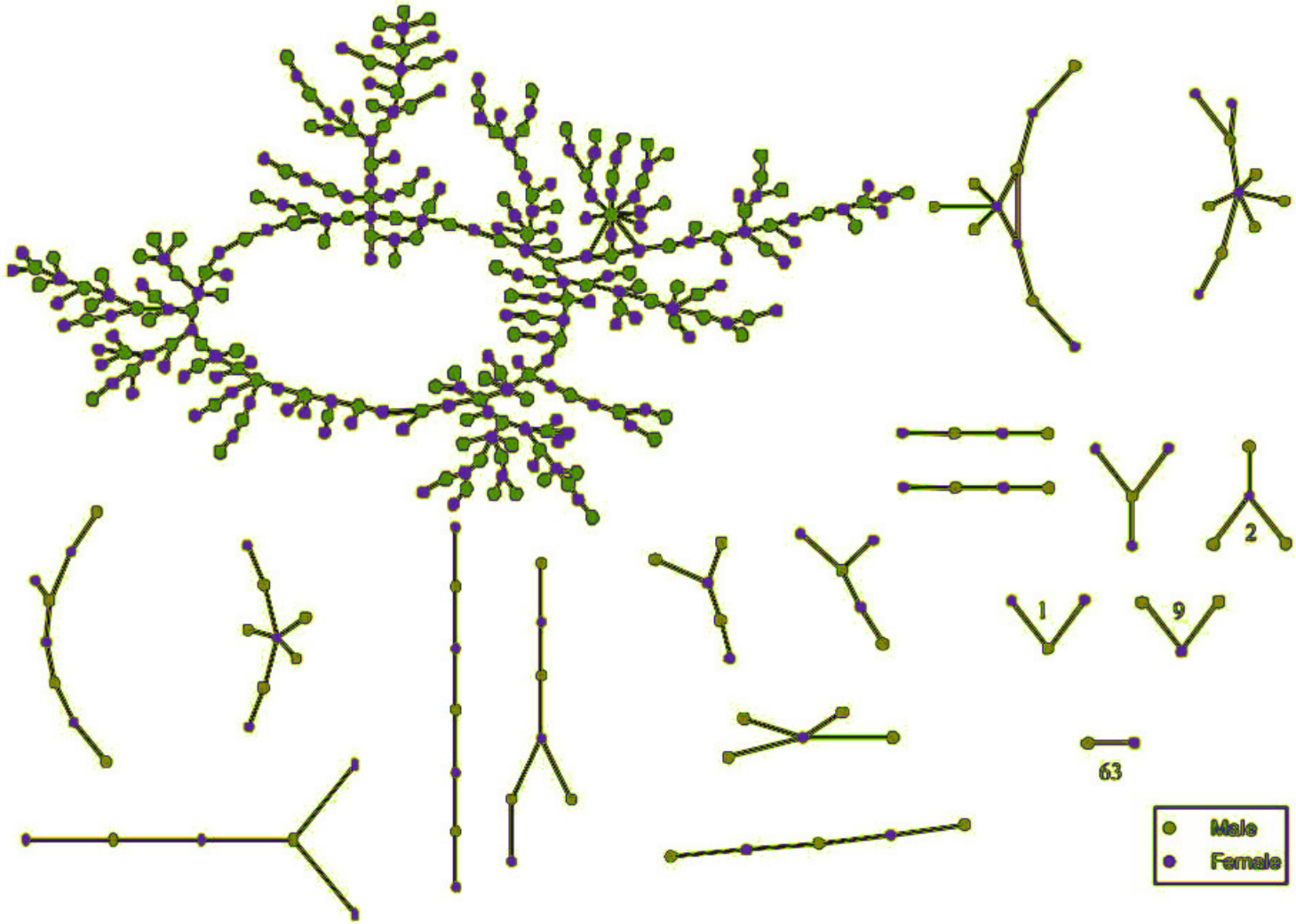


Figura: Peter Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. American Journal of Sociology, 110(1): 44-99, 2004.

Implementação para Componentes Conectados

```
1  class CompConectado {
2   boolean marcado[];
3   int id[];
4   int componentes;
5
6   public CompConectado(Grafo G) {
7     marcado = new boolean[G.V()];
8     id = new int[G.V()];
9
10    for (int v = 0; v < G.V(); v++) {
11      if (marcado[v] == false) {
12        dfs(G, v);
13        componentes++;
14      }
15    }
16  }
17
18  int count() {/* continua */}
19  int id(int v) {/* continua */}
20  void dfs(Grafo G, int v) {/* usar não recursivo */}
21 }
```

Operações auxiliares

```
1 int count() {
2   return componentes;
3 }
4 int id(int v) {
5   return id[v];
6 }
```

Busca em profundidade modificada

```
1  void dfs(Grafo G, int v) {
2   Stack pilha = new Stack();// pilha substitui recursão
3   pilha.push(origem);
4
5   while (pilha.empty() == false) {
6     int atual = (int) pilha.pop();//pega mais recente
7     if (marcado[atual] == false) {
8         marcado[atual] = true;
9         id[atual] = componentes; // MARCAR o vertice
10        for (int adjacente: G.adj(atual)) {
11            veioDe[adjacente] = atual; //marca caminho
12            pilha.push(adjacente);
13        }
14    }
15  }
16
17 }
```