Exemplo de atividade de treinamento para resolução de problemas

Tiago Montalvão

Competições de Algoritmos e Programação UFRJ

Ideia

A ideia desta apresentação é fornecer ferramentas para a resolução de alguns problemas. Os problemas apresentados serão:

- Geração aleatória de um labirinto
- Resolução deste labirinto
- Uso da ideia da resolução para outros problemas, como:
  - Resolução de um Sudoku
  - Problema das N rainhas

Durante a apresentação, as ferramentas necessárias para resolver tais problemas serão apresentadas.

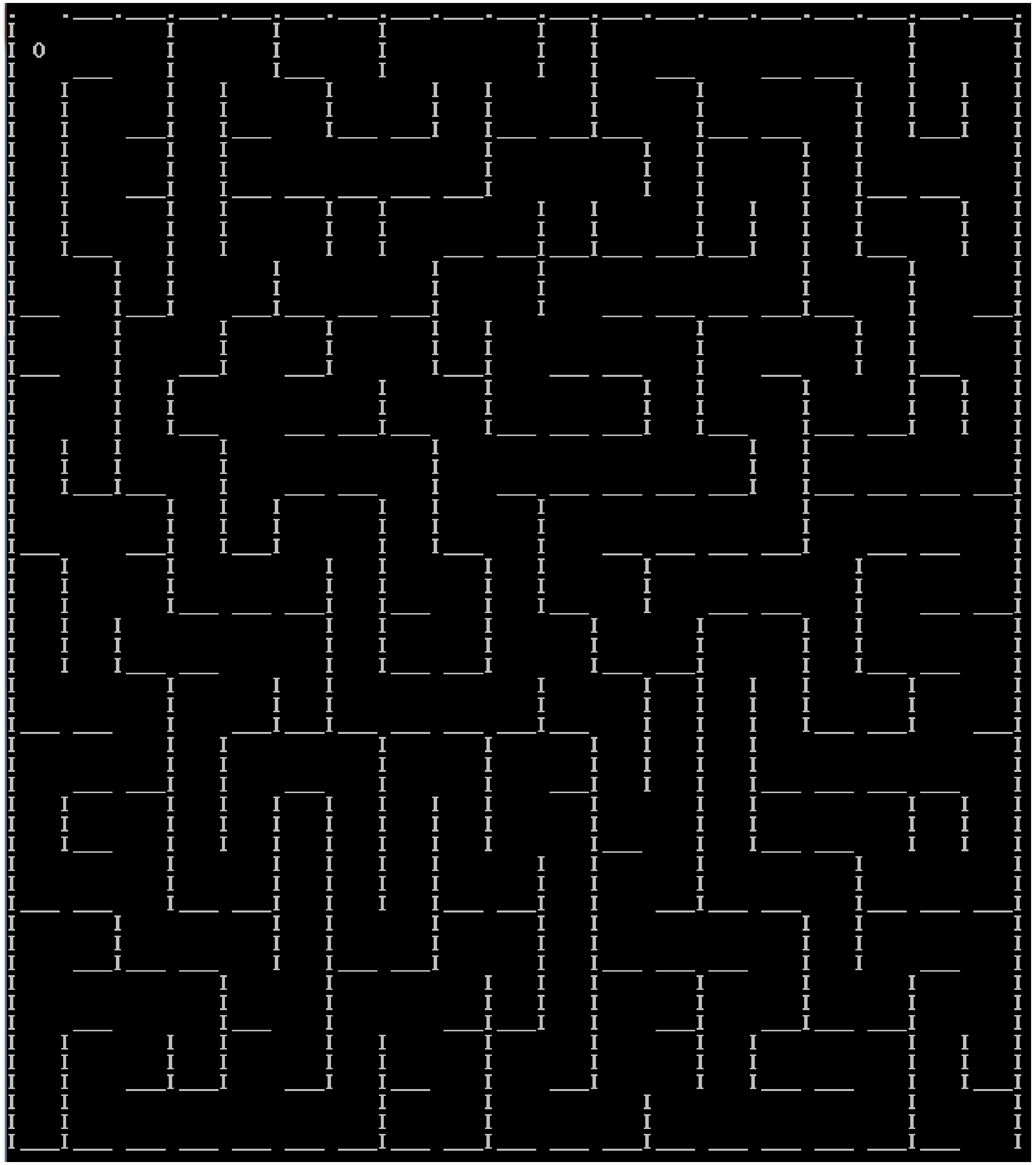
Motivação

A motivação por trás desta apresentação é fornecer um pouco do conhecimento obtido através das Competições de Algoritmos e Programação. Com o treinamento para tais competições, adquire-se bastante conhecimento e prática de vários algoritmos, como alguns aqui mostrados.

Problema inicial

Como gerar um labirinto de maneira aleatória, de forma a apresentar apenas um caminho de sua entrada até a saída?

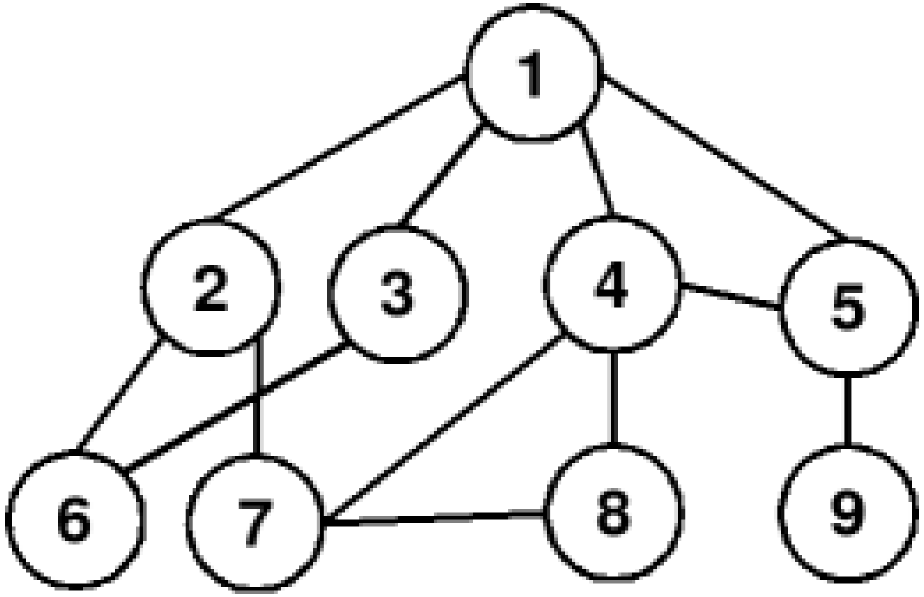
Entrada



Saída

Mas antes, precisamos de alguns conceitos...

Grafos



Grafos

Um grafo consiste de:

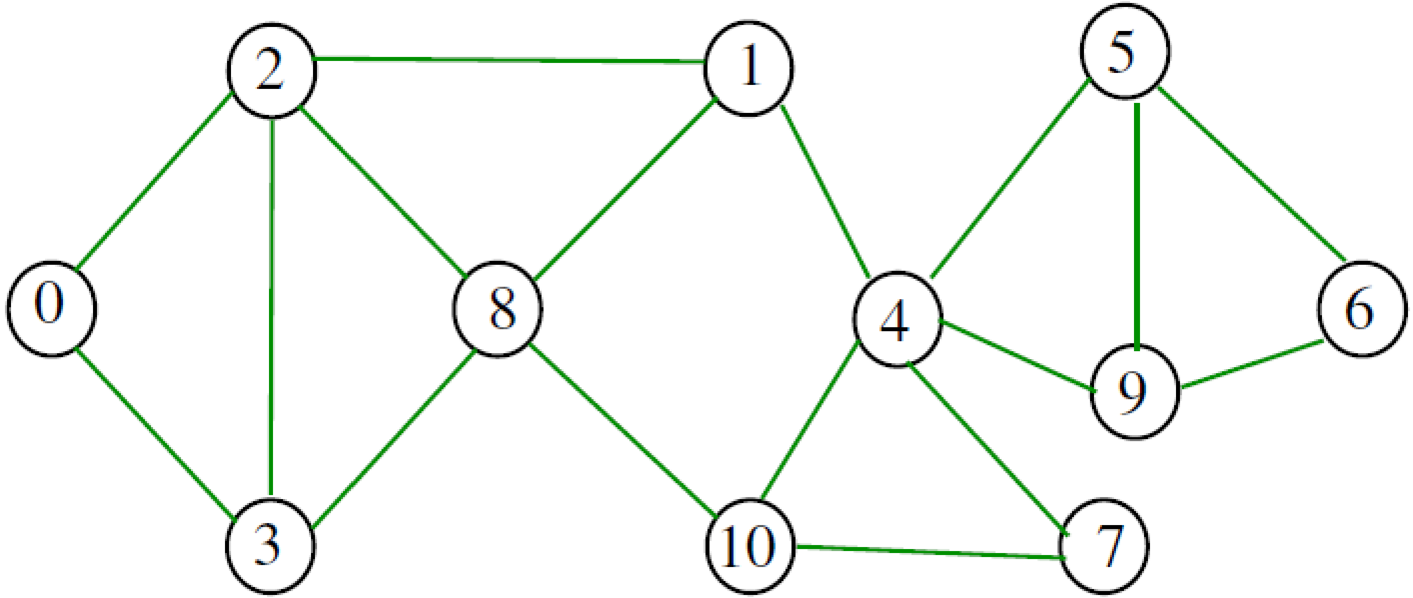
- Um conjunto V de vértices ou nós (pontos)
- Um conjunto E de arestas (ligações, relações)

Um vértice é dito vizinho de outro se houver uma aresta entre eles

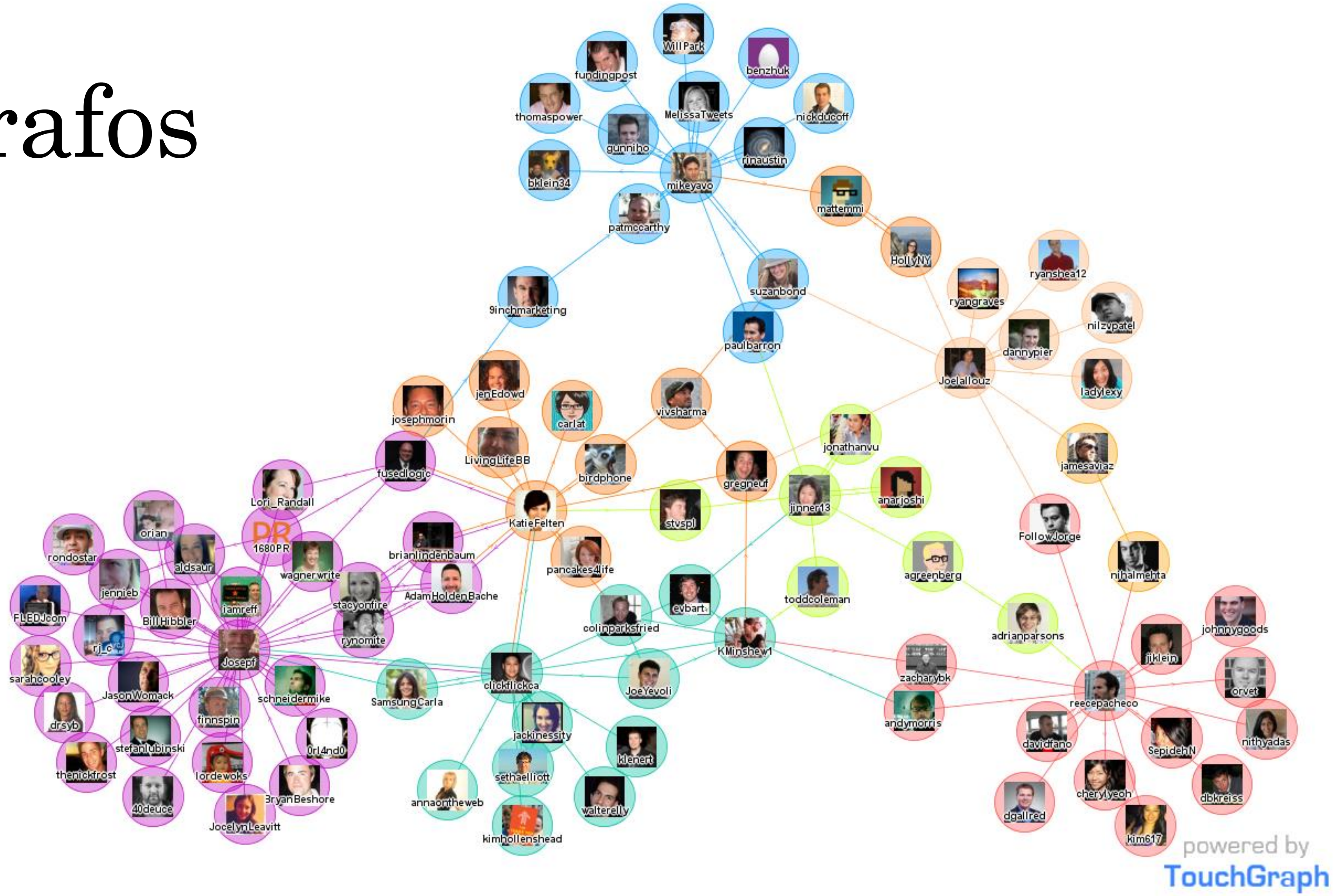
Usado para representar conexões entre elementos

Alguns exemplos:

Grafos

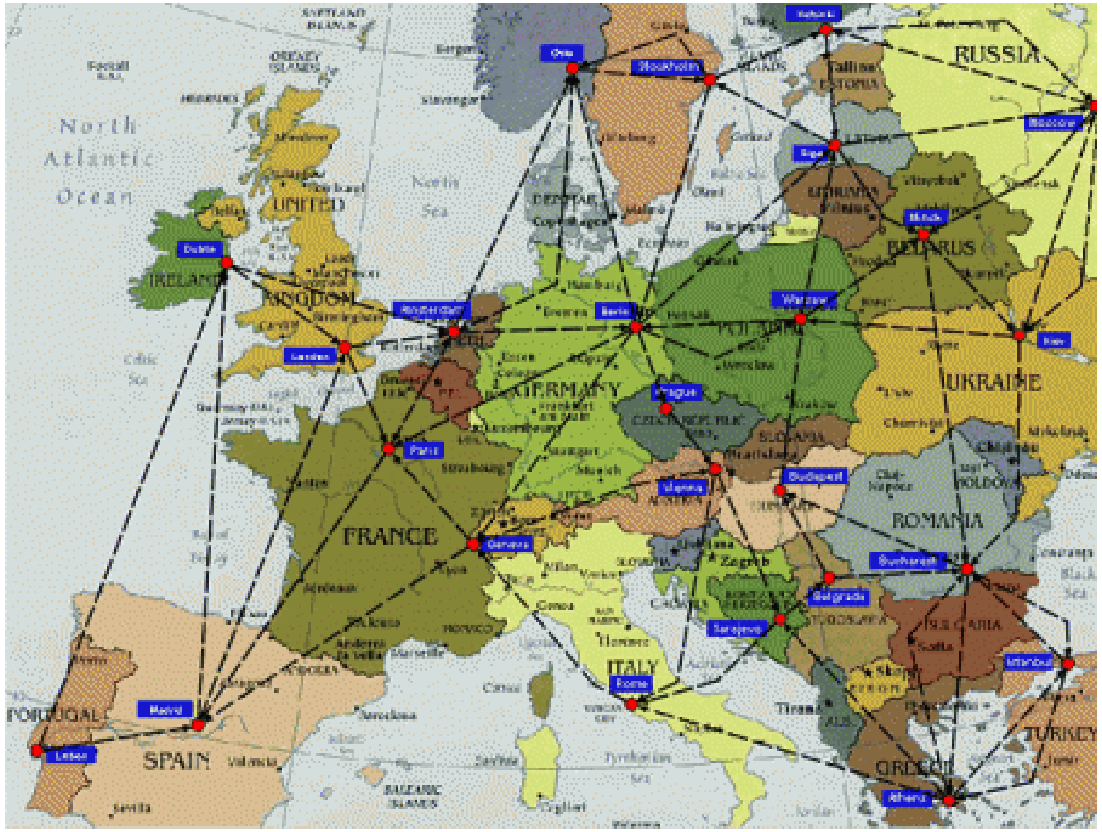


Grafos



Visualização de redes sociais

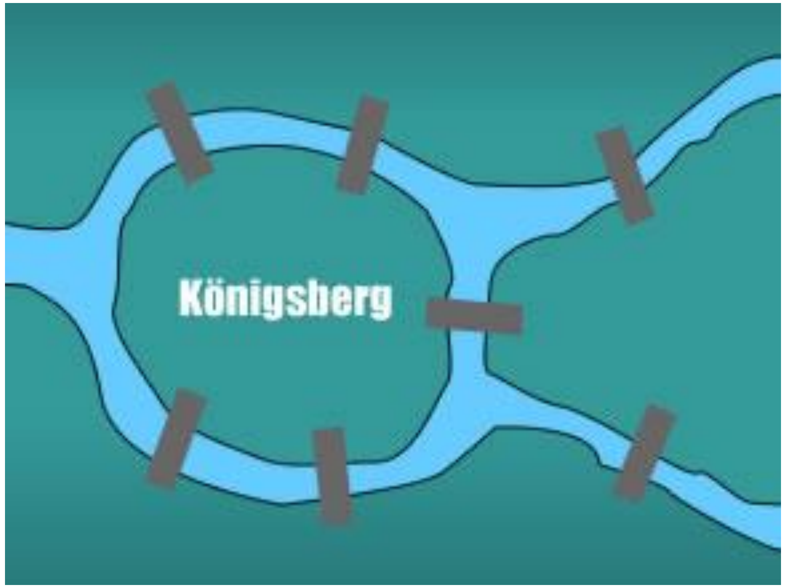
Grafos



Rede de voos cruzando a Europa

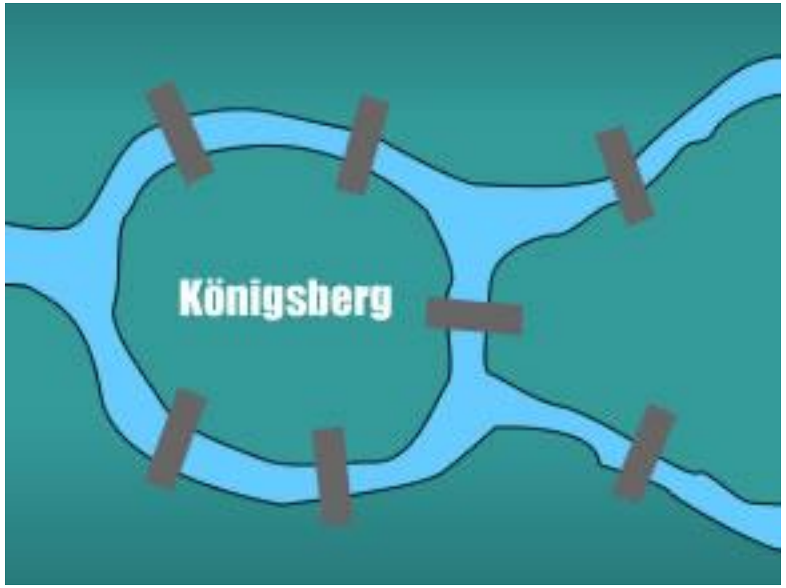
História

- Leonhard Euler é considerado o precursor da teoria de grafos, com a publicação, em 1736, de um artigo sobre as sete pontes de Königsberg

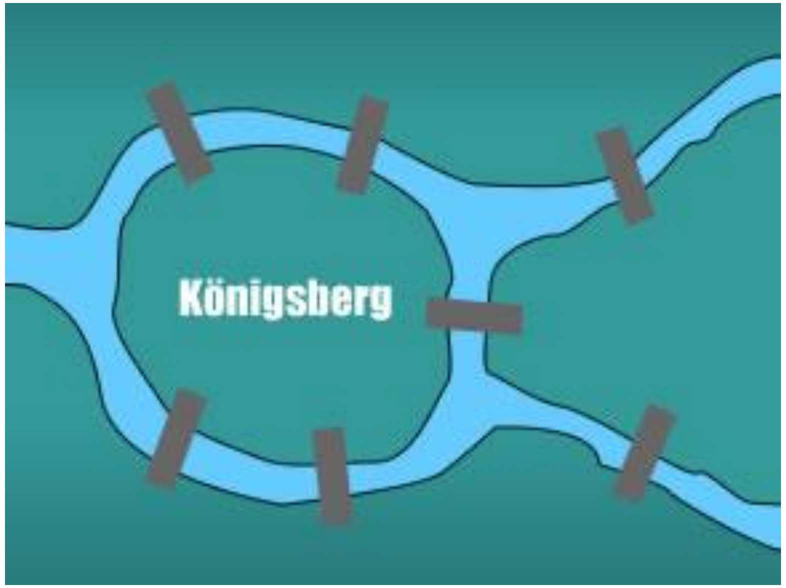


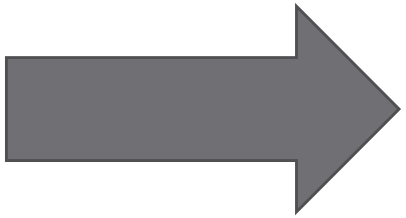
História

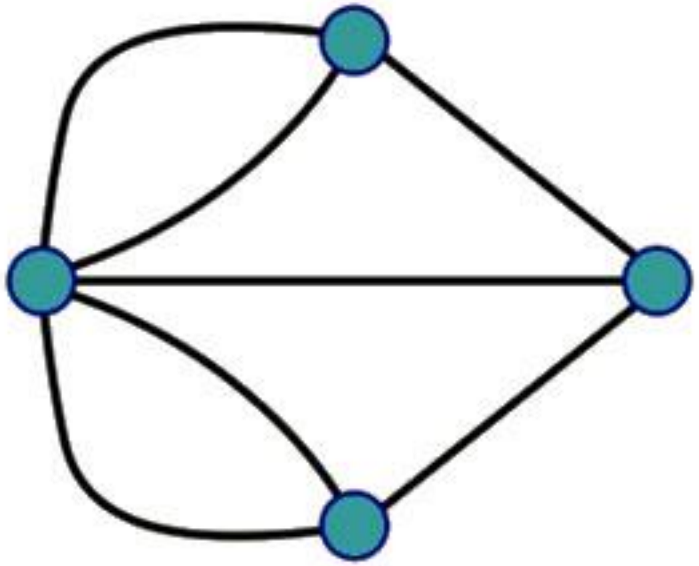
O problema a ser resolvido era se era possível, em Königsberg, passar por todas as pontes exatamente uma vez.



Interpretação do problema

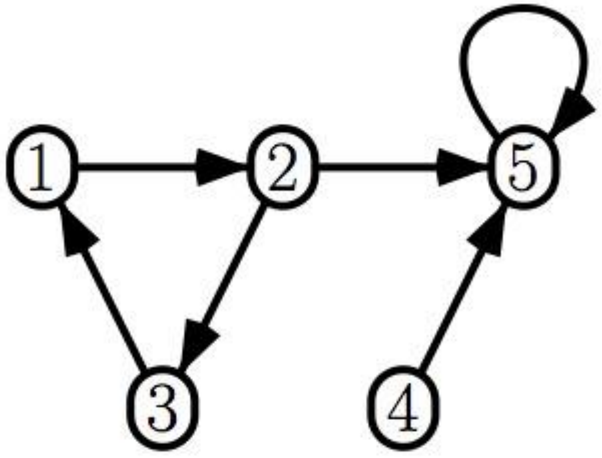




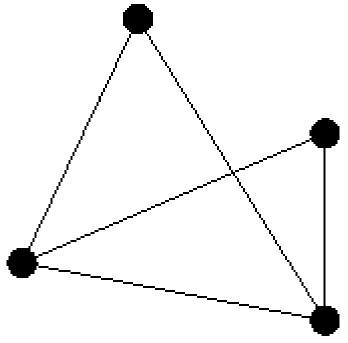


Tipos de grafo

- Grafo direcionado – relações unidirecionais (exemplo com laço)

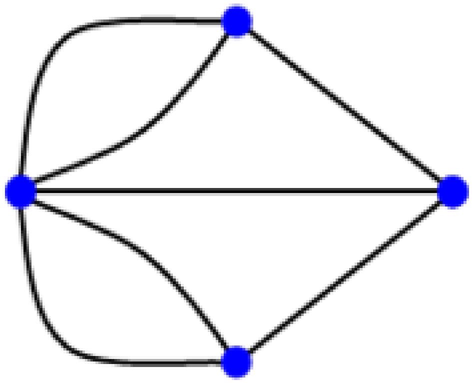


- Grafo não direcionado – relações sem importância de sentido

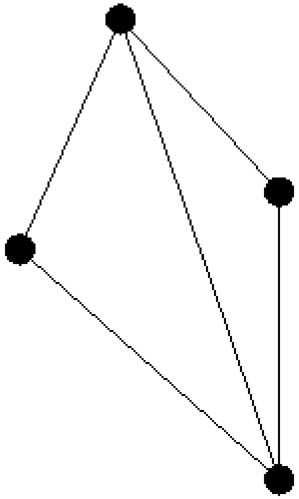


Tipos de grafo

- Multigrafo – grafo que permite que mais de um aresta ligue o mesmo par de vértices

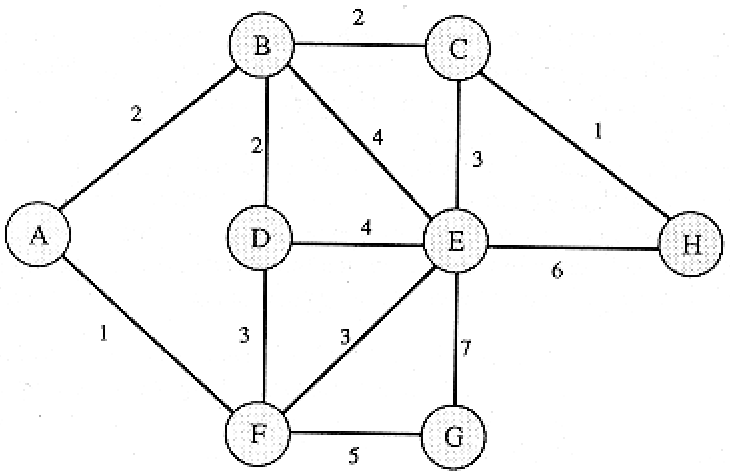


- Grafo simples – Grafo não direcionado, sem laços e com no máximo uma aresta ligando um par de vértices



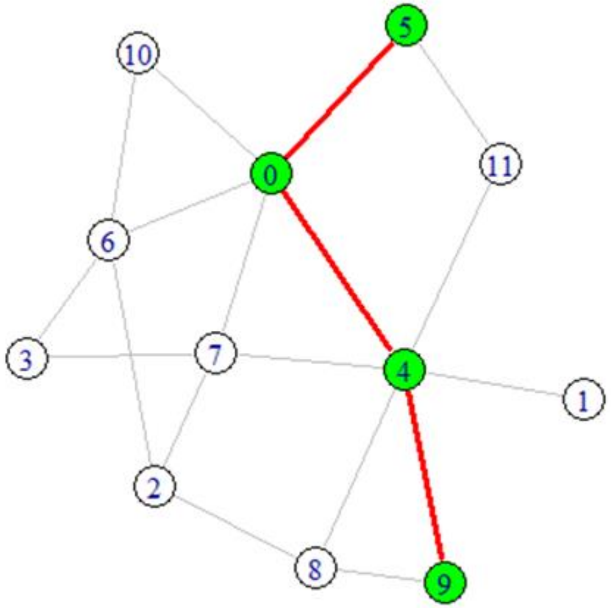
Tipos de grafo

- Grafo com pesos nas arestas – arestas apresentam pesos (ou custos) para ir de um vértice a outro



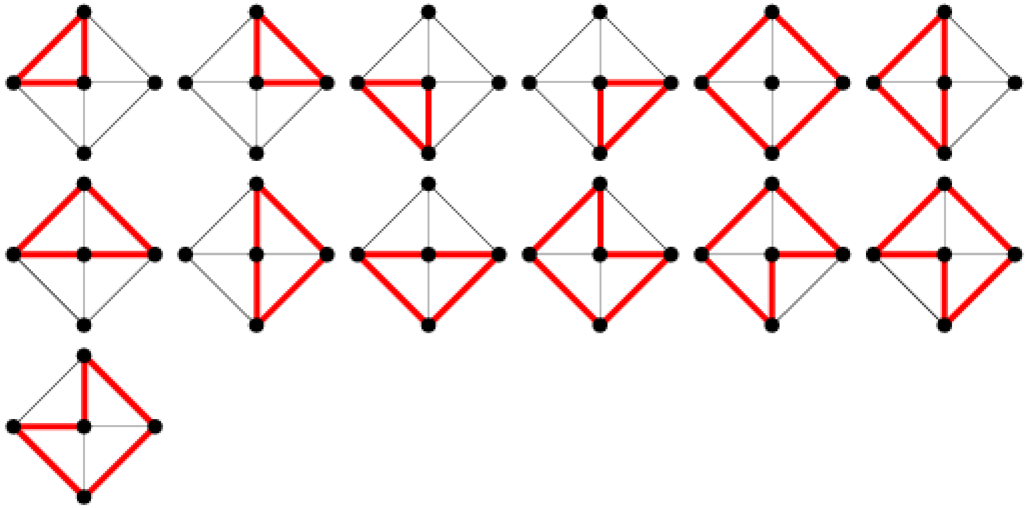
Caminho em um grafo

Sequência consecutiva de vértices e arestas



Ciclo em um grafo

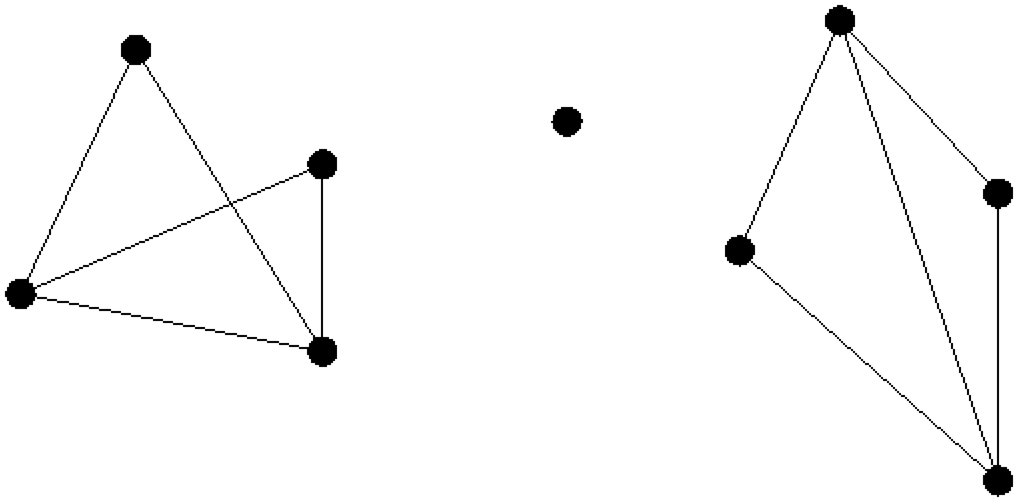
Qualquer caminho que comece e termine no mesmo vértice



Componentes conexas

Cada conjunto de vértices e arestas de um grafo, tal que exista um caminho de um vértice a qualquer outro.

Um grafo é dito conexo se for composto por apenas uma componente conexa.

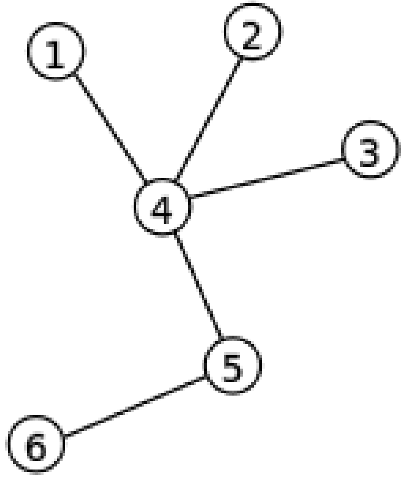


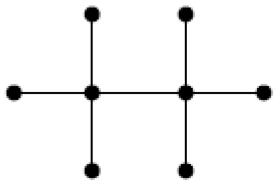
Árvores

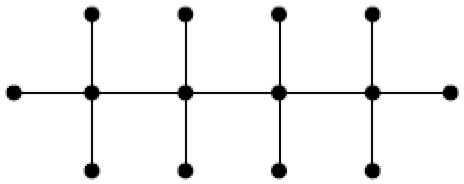
Grafos simples, acíclicos e conexos

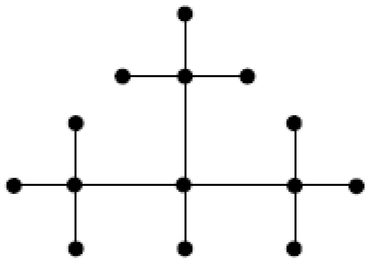
Em uma árvore, só existe um caminho entre qualquer par de vértices

O número de arestas é uma unidade menor que o número de vértices





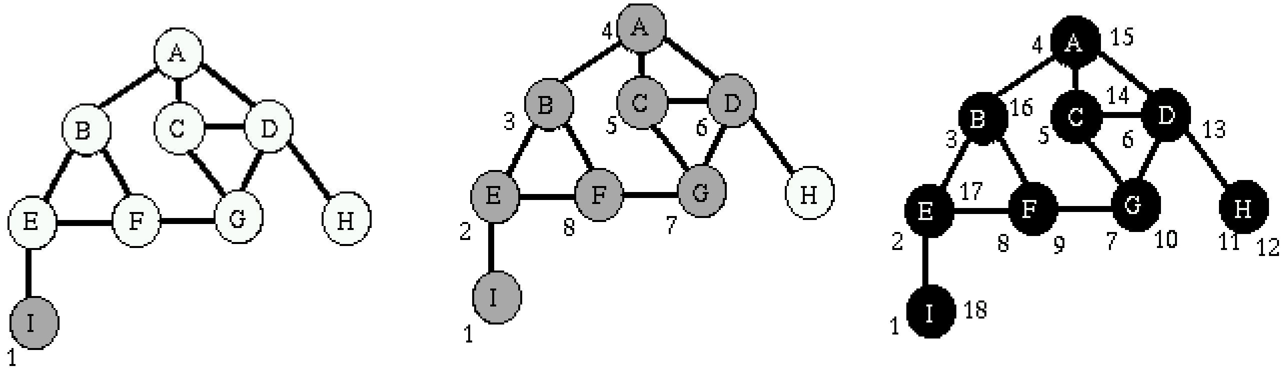




Como explorar todos os vértices em um grafo?

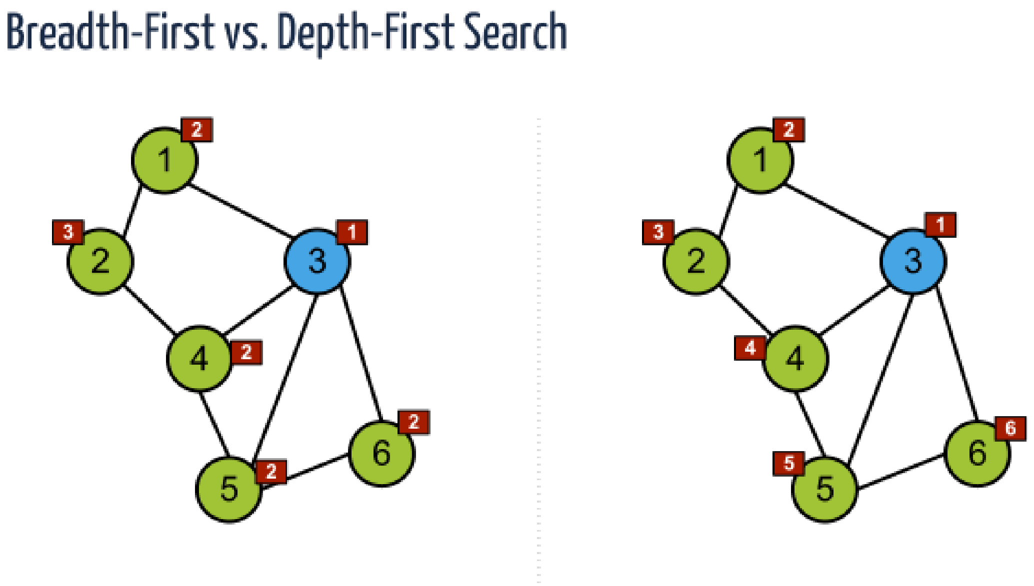
Buscas em um grafo

Técnica utilizada para visitar todos os vértices de um grafo e explorar cada aresta



Buscas em um grafo

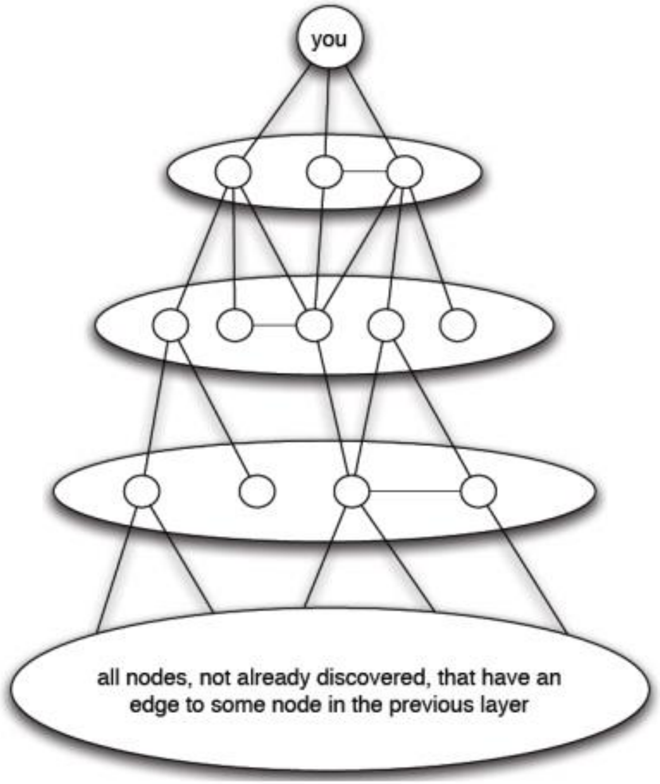
- Busca em largura (Breadth-first search ou BFS) – Busca por camadas
- Busca em profundidade (Depth-first search ou DFS) – Busca enquanto der

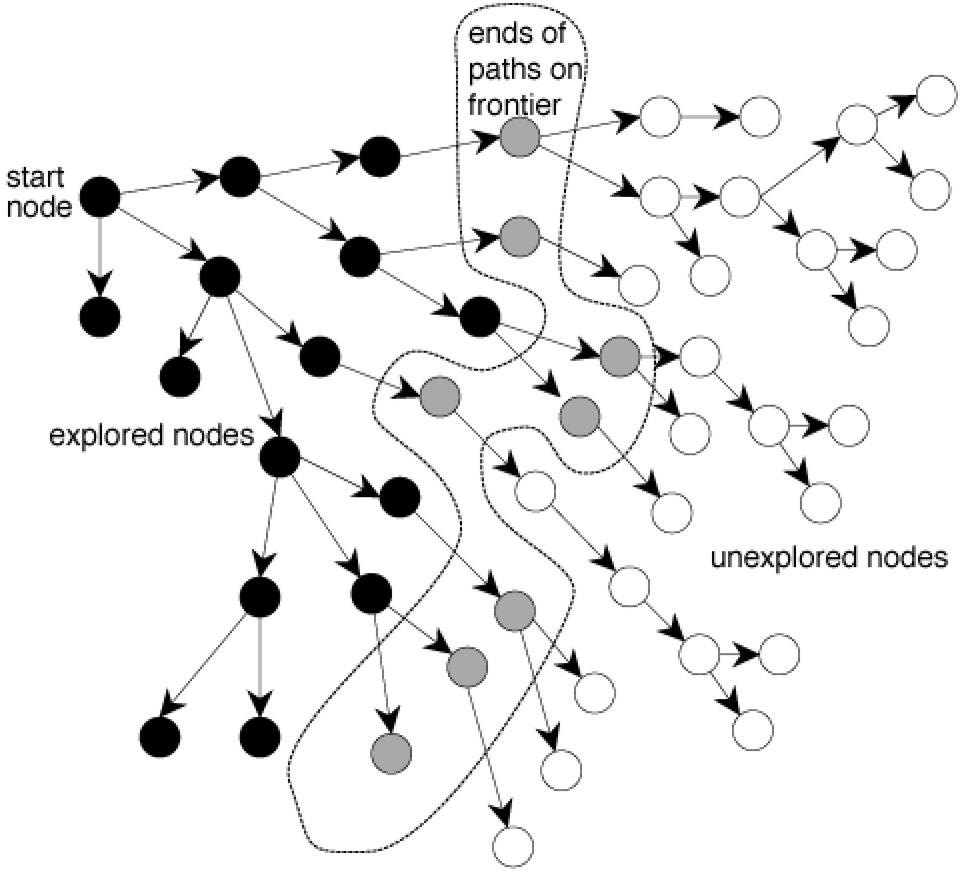


Busca em largura

1. Visita um vértice dado;
2. Visita todos os vizinhos deste vértice;
3. Visita todos os vizinhos dos vértices já visitados;
4. Repete o passo 3 até não haver mais vizinhos não visitados.

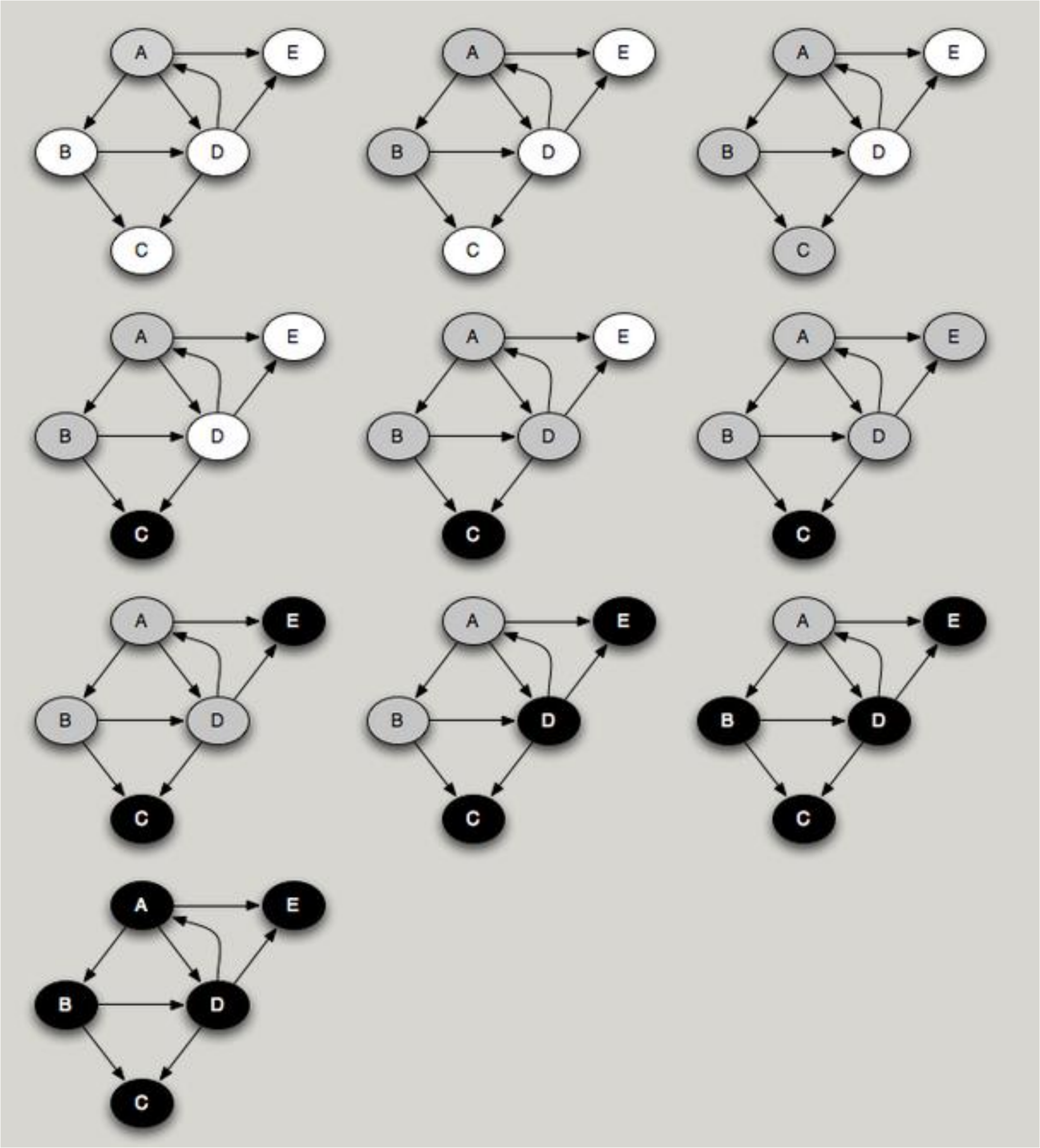
Busca em largura





Busca em profundidade

1. Visita um vértice dado;
2. Visita um vizinho deste vértice;
3. Visita um vizinho deste novo vértice que ainda não foi visitado;
4. Repete o passo 3 até não haver mais vizinhos não visitados;
5. Retorna ao vértice anterior;
6. Retorna ao passo 3 até não haver mais vizinhos não visitados.

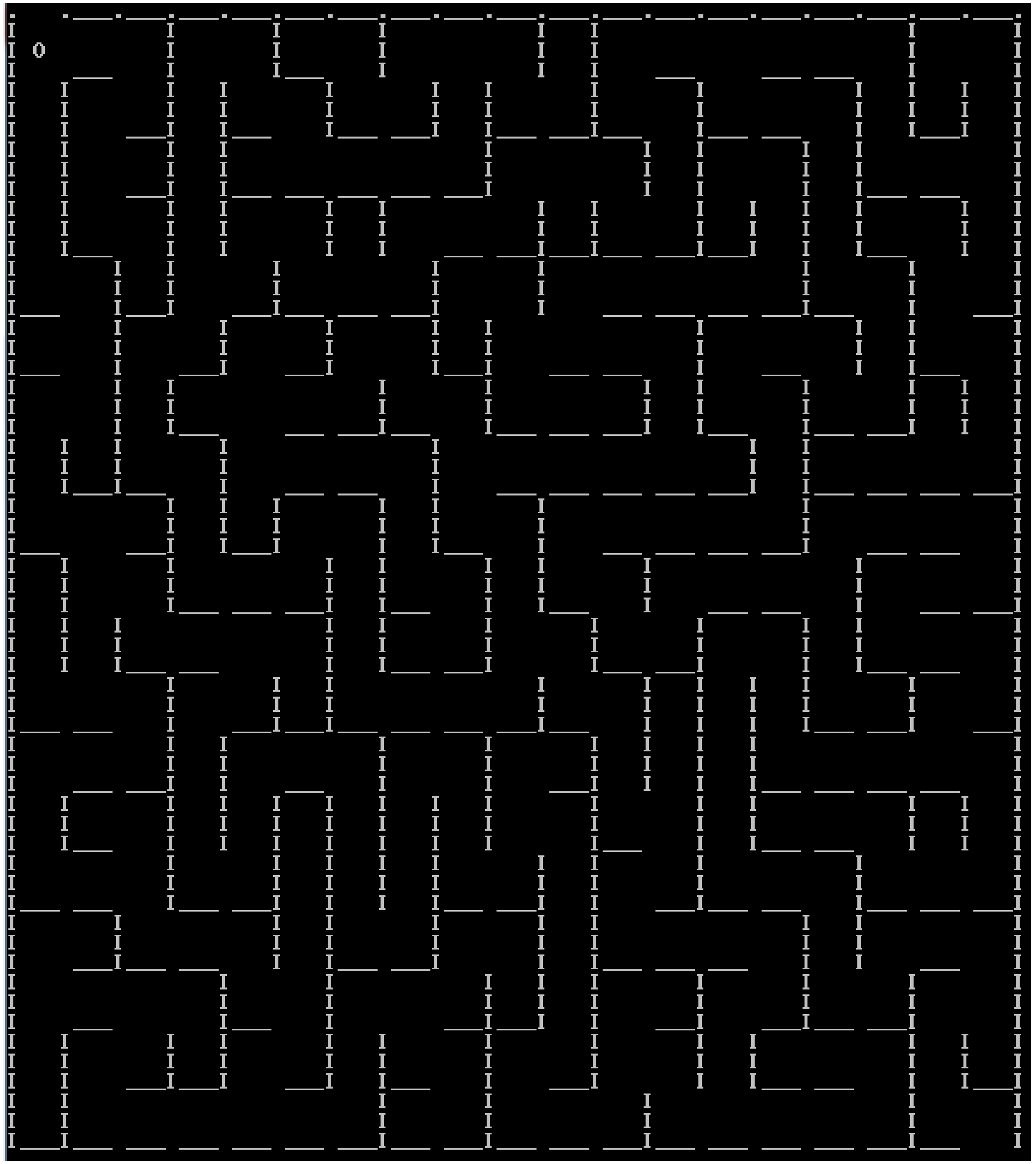


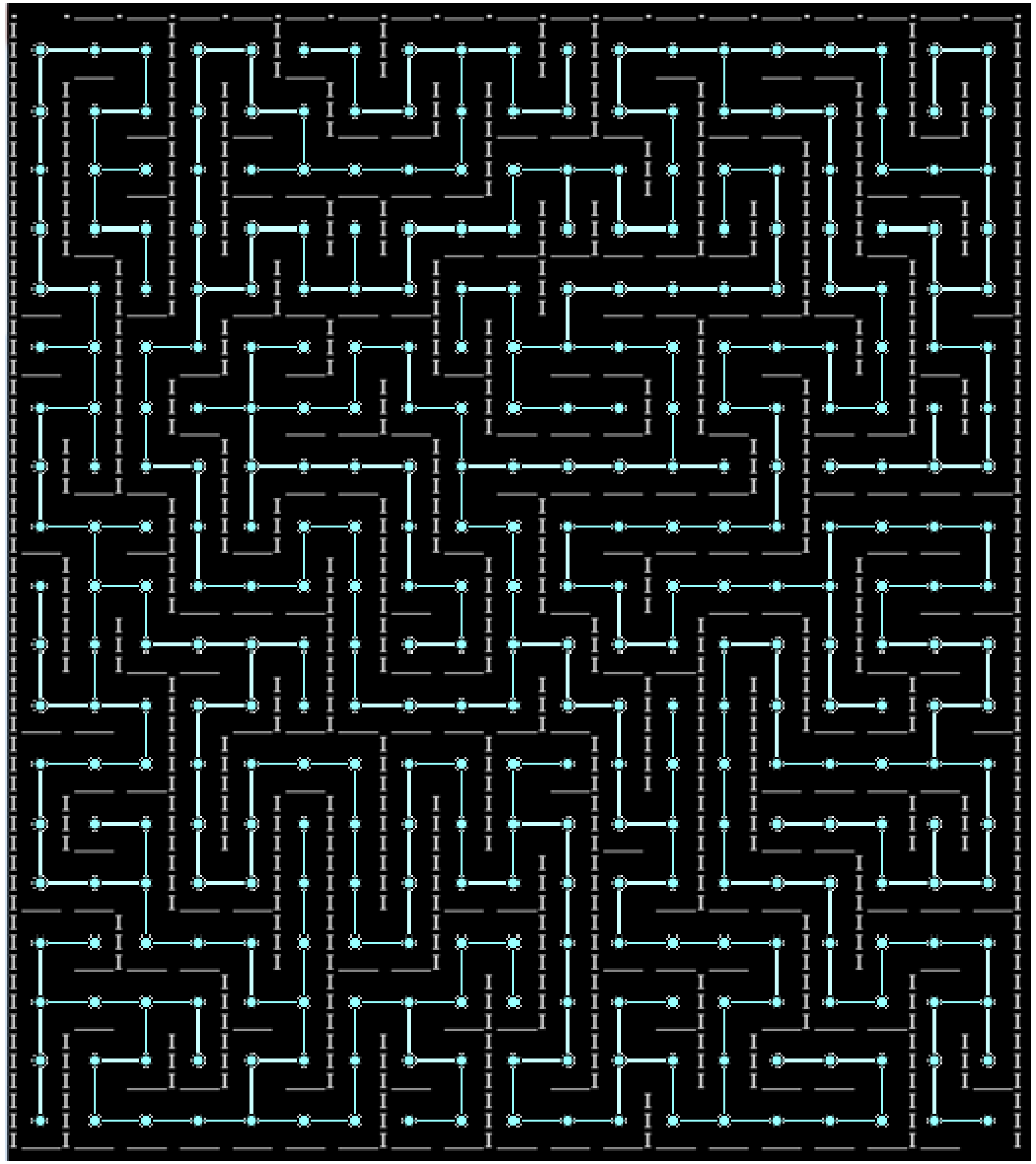
Busca em profundidade a partir do vértice A

Voltando ao problema original...

Modelagem do problema

1. Primeiro dividimos o labirinto em células de uma grade;
2. Cada célula é interpretada como um vértice;
3. Se houver caminho entre células adjacentes, representamos como uma aresta;
4. No final teremos um grafo, como a seguir:





Como gerar o labirinto

Este grafo obtido é uma árvore, pois:

- Existe caminho de qualquer ponto a qualquer outro (grafo conexo)
- Não há mais de uma aresta entre pares de vértices (grafo simples)
- Não há ciclos (grafo acíclico)

Logo, precisamos saber gerar uma árvore a partir de um grafo qualquer, ou seja, mantendo todos vértices e escolhendo apenas arestas de maneira a não formar ciclos.

Árvore geradora

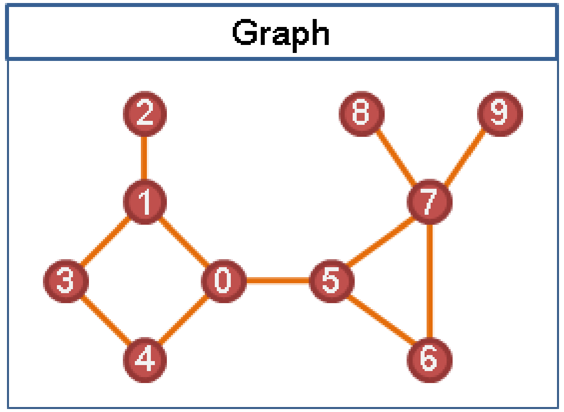
Alguns algoritmos são bem conhecidos por gerarem uma árvore de um grafo, como:

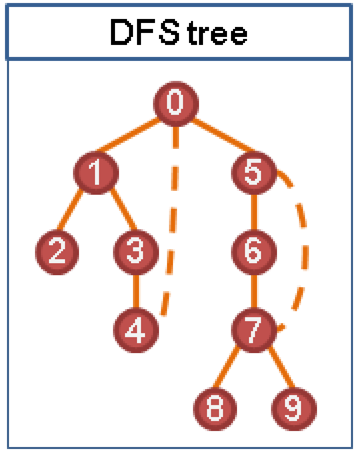
- Busca em profundidade
- Algoritmo de Prim
- Algoritmo de Kruskal

Os últimos dois são conhecidos como algoritmos de árvore geradora mínima, pois geram a árvore com menor soma dos pesos das arestas

Busca em profundidade

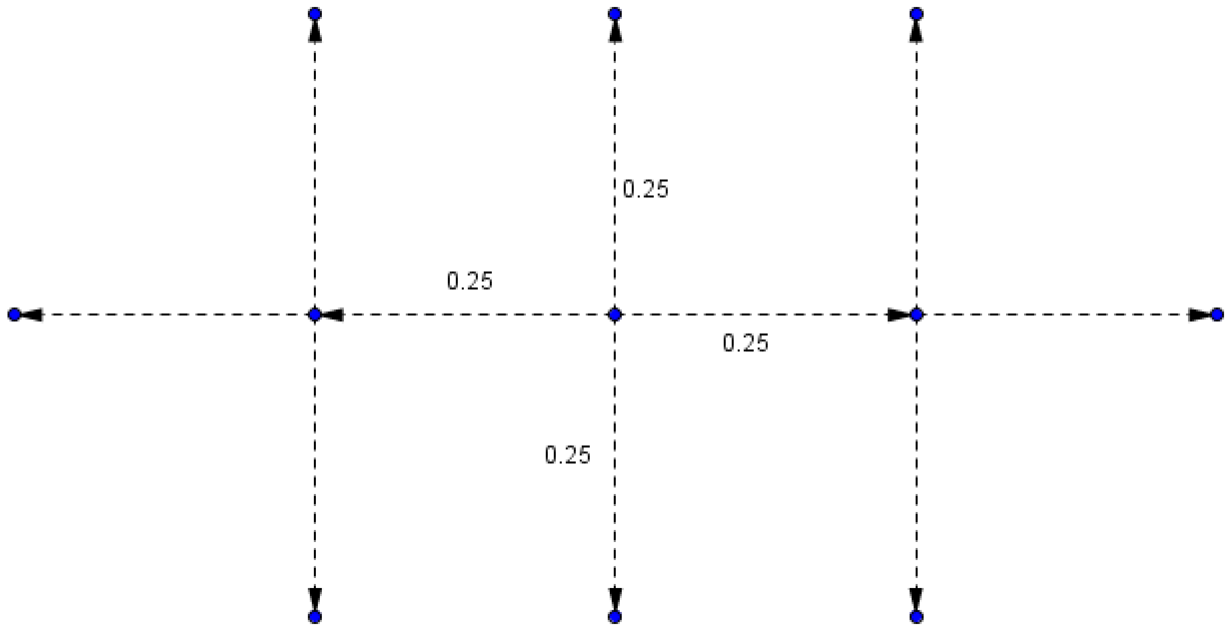
Ao explorar o grafo com a busca em profundidade, como nunca visitamos o mesmo vértice duas vezes por caminhos diferentes, o percurso obtido é uma árvore.



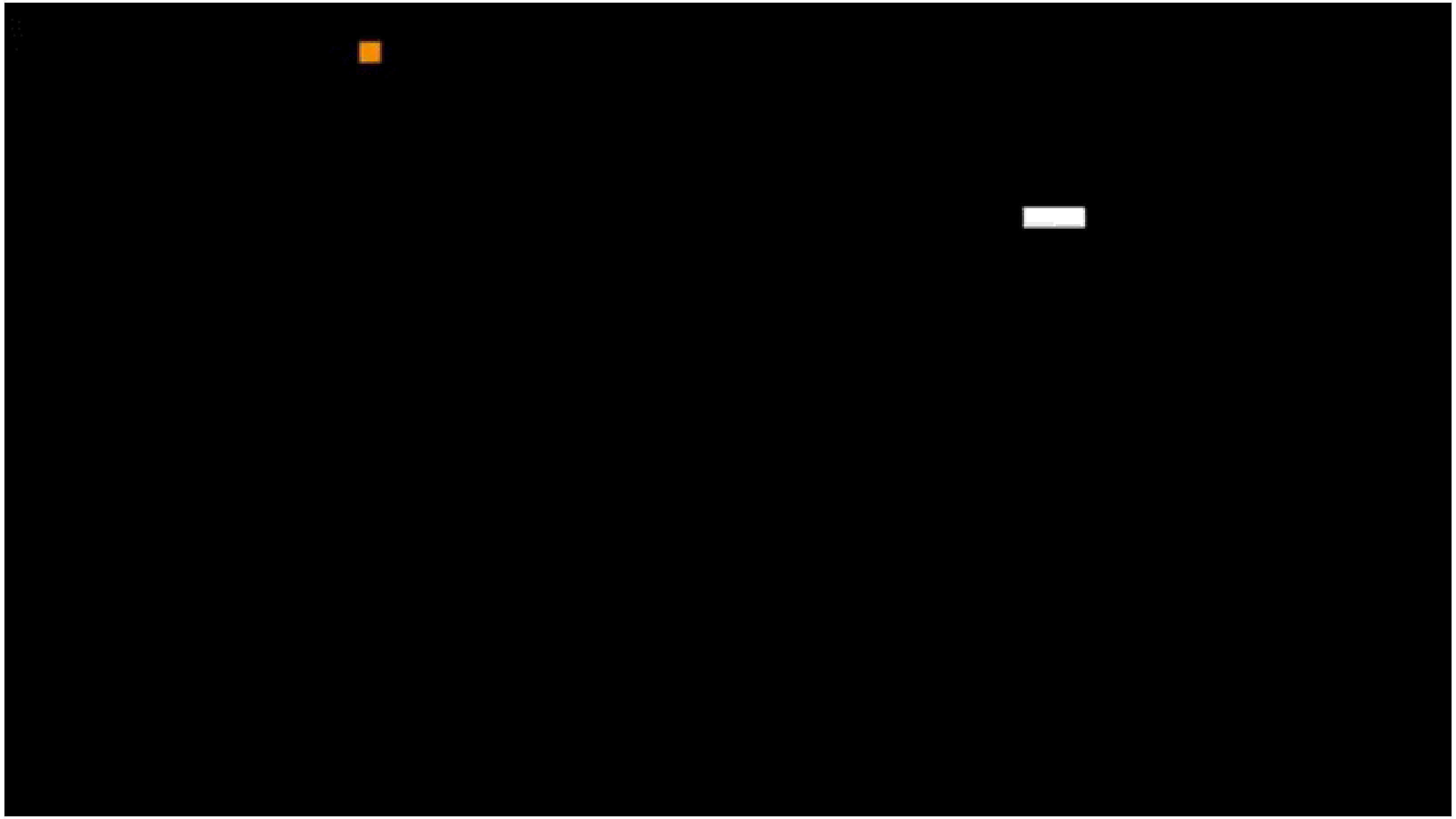


Busca em profundidade

O que fazemos aqui é, estando em um vértice qualquer, visitamos um vizinho de forma aleatória. Isto é feito para todo vértice. Ao final, a árvore gerada será aleatória



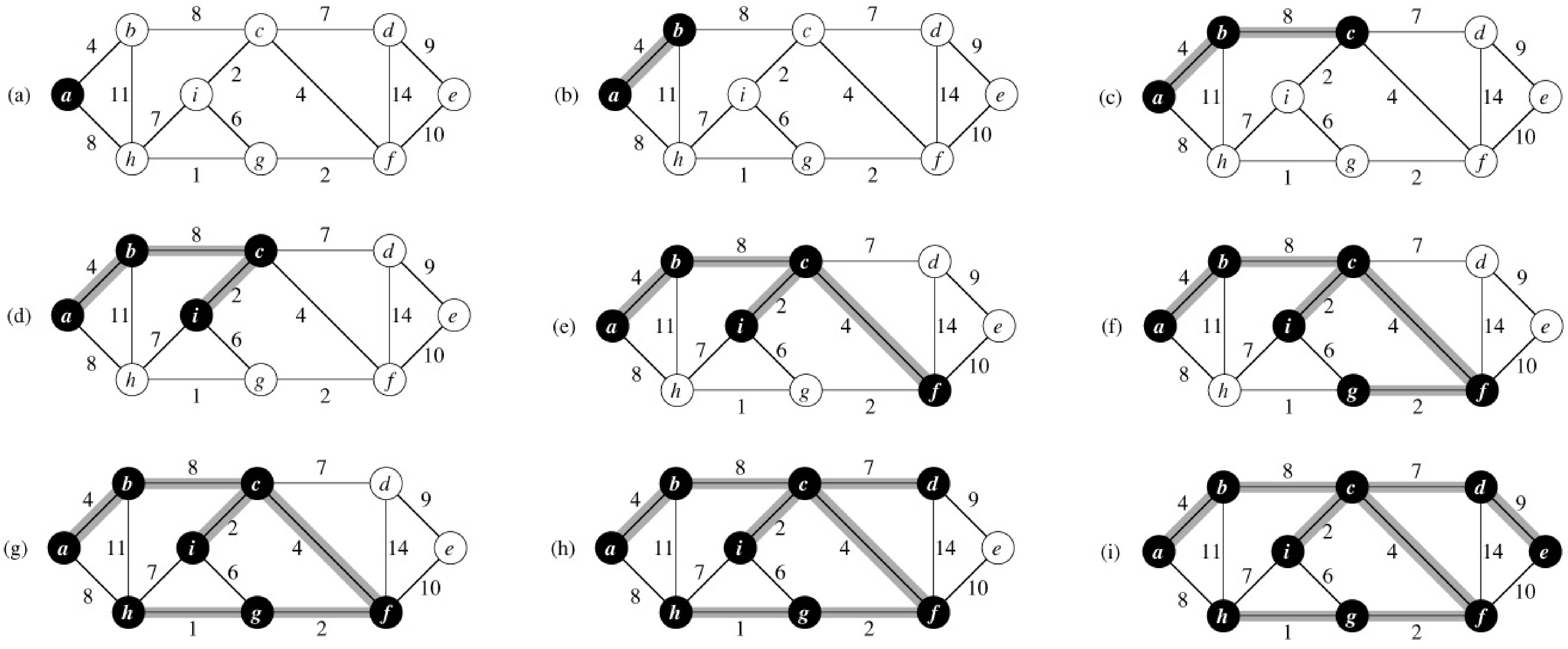
Busca em profundidade



Algoritmo de Prim

O algoritmo a seguir gera a árvore com soma mínima dos pesos nas arestas:

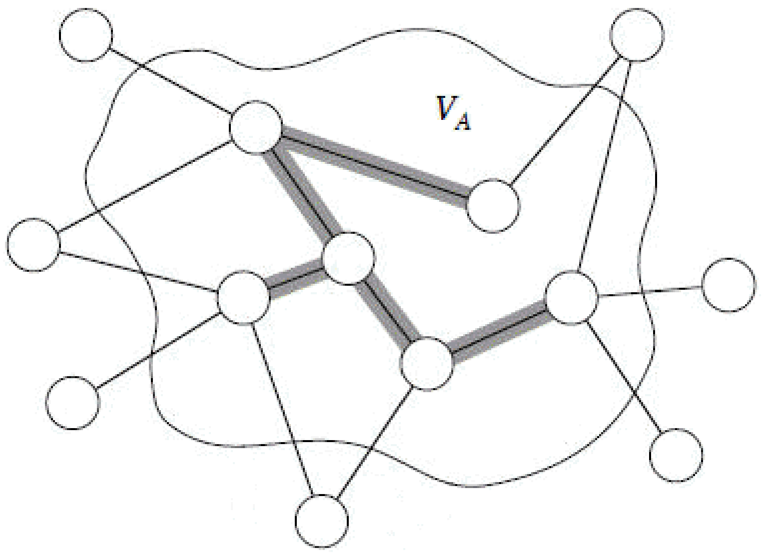
1. Adiciona à árvore um vértice arbitrário;
2. Adiciona à arvore uma aresta. Ela deve ser a de peso mínimo dentre aquelas que conectam a árvore a vértices ainda não visitados;
3. Repete o passo 2 até que todos vértices sejam visitados.



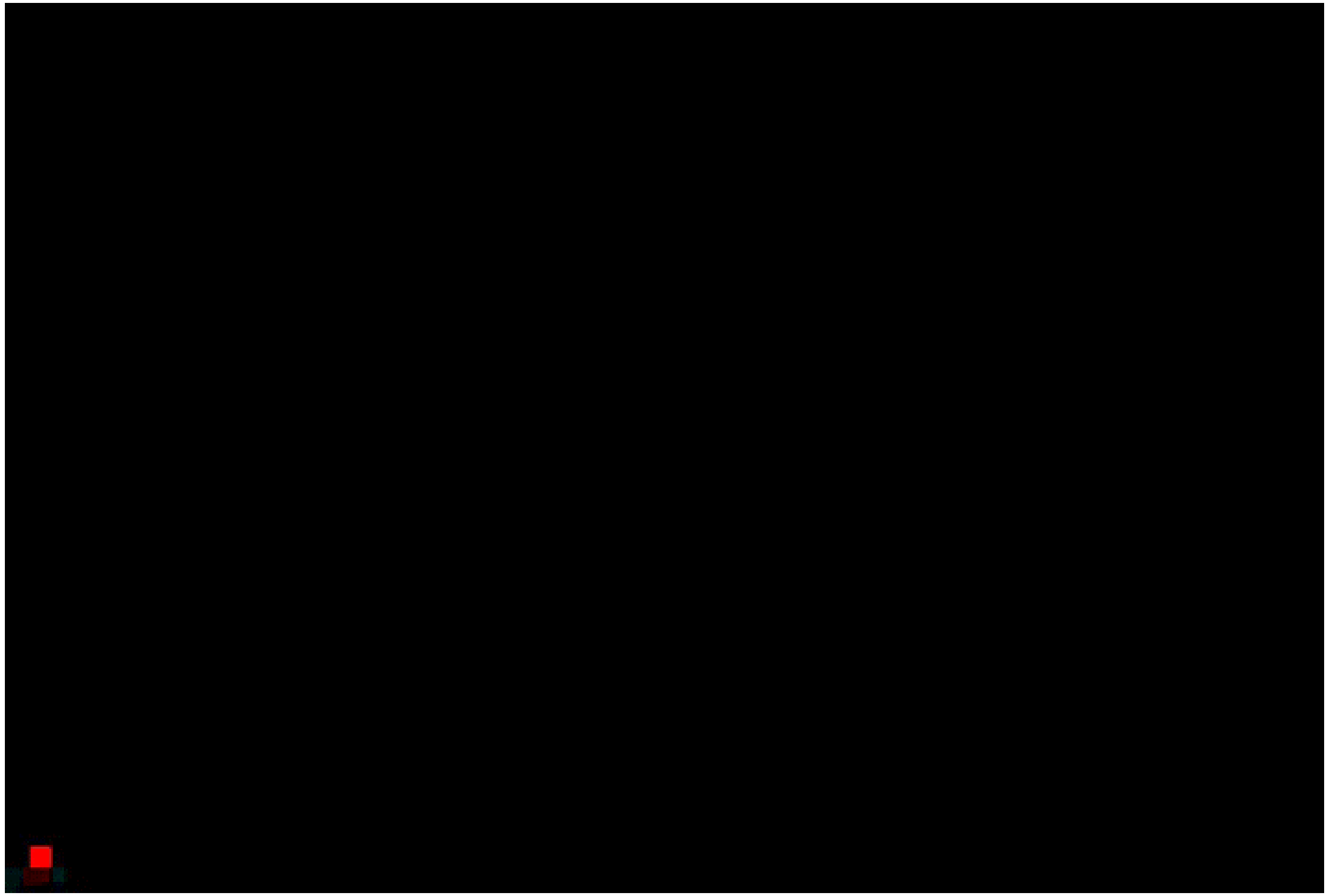
Algoritmo de Prim a partir do vértice a

Algoritmo de Prim

Nesta aplicação, podemos ignorar os pesos (todas arestas possuem o mesmo peso). Sendo assim, a cada passo, podemos adicionar à arvore qualquer aresta de forma aleatória.



Algoritmo de Prim

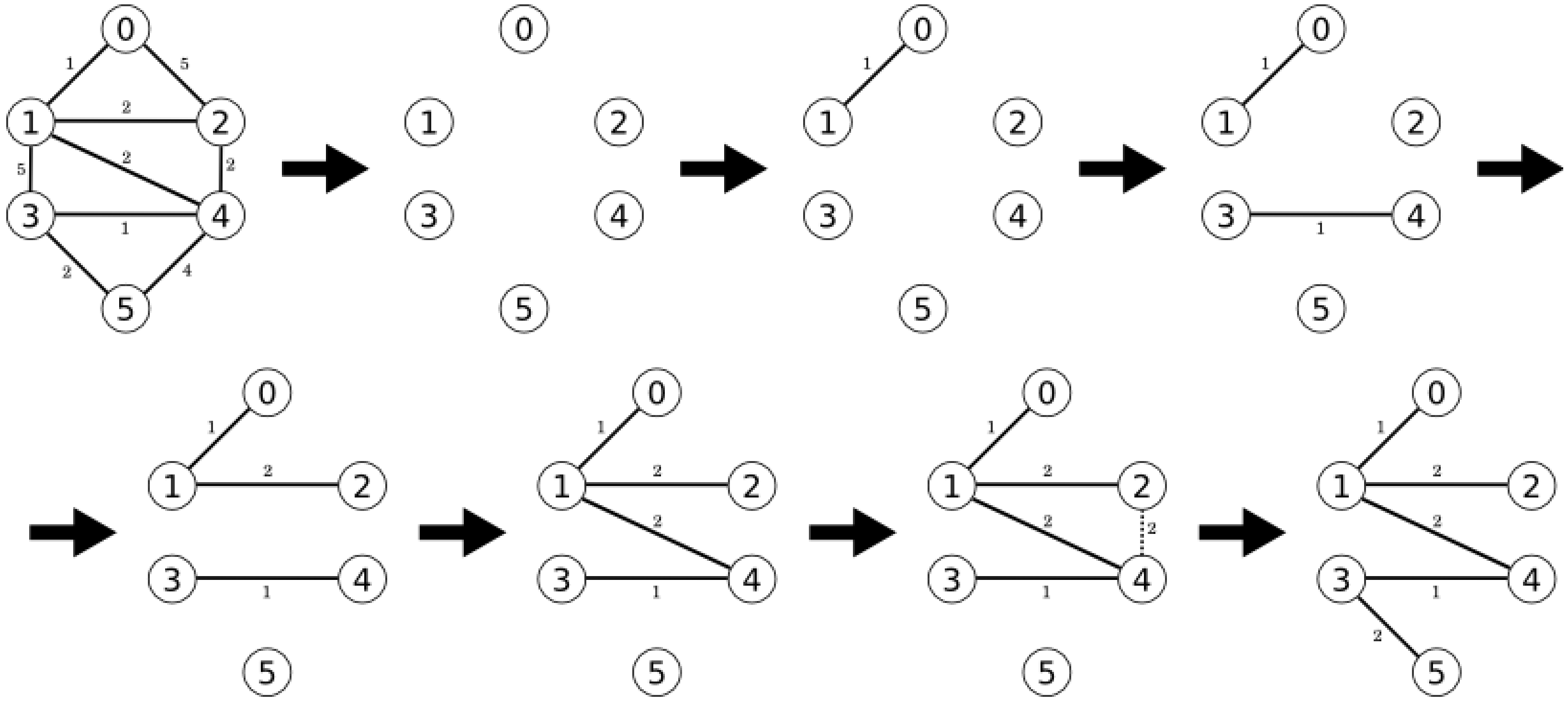


Algoritmo de Kruskal

O algoritmo a seguir gera a árvore com soma mínima dos pesos nas arestas:

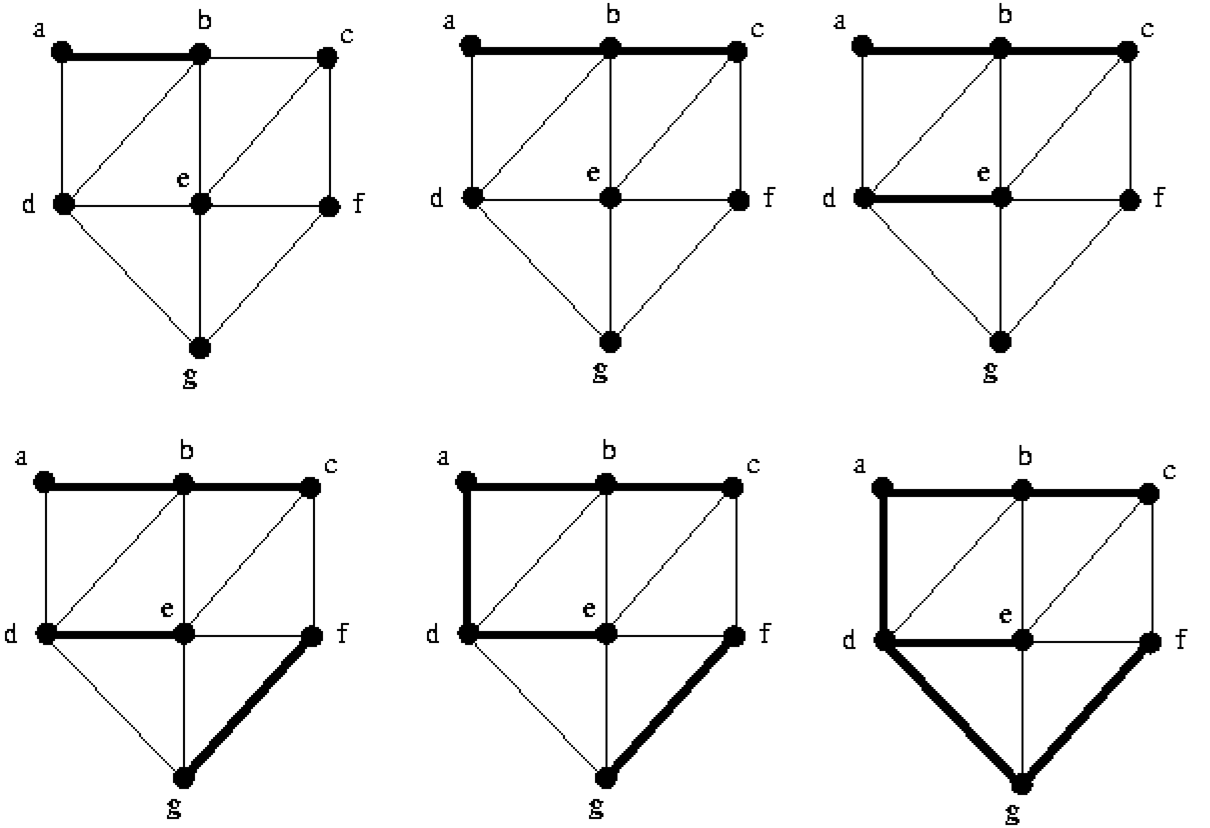
1. Cria uma lista com todas as arestas do grafo;
2. Ordena esta lista em relação aos pesos das arestas;
3. Percorre toda a lista adicionando, em ordem, as arestas à árvore, exceto se ela formar um ciclo.

Algoritmo de Kruskal

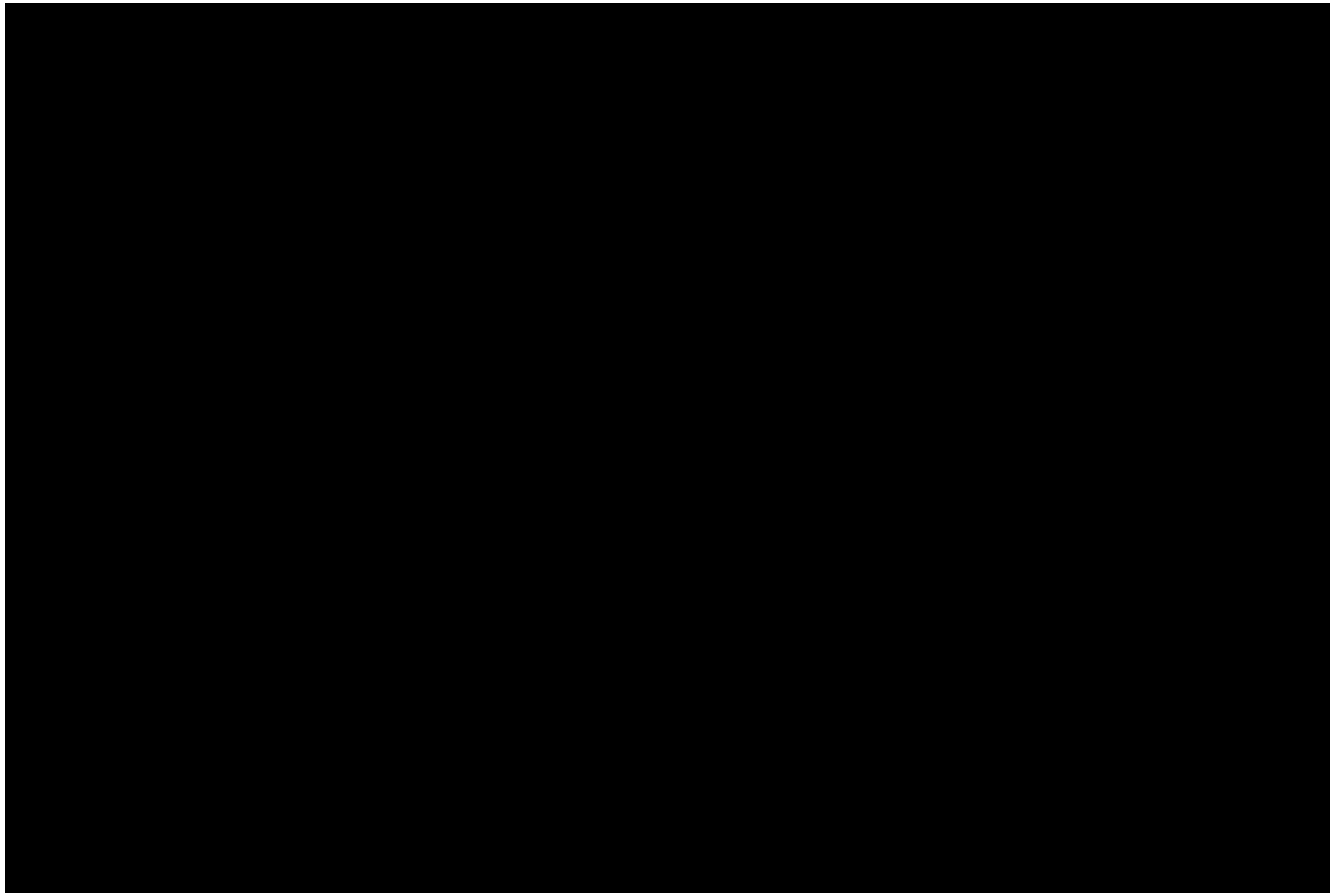


Algoritmo de Kruskal

- Nesta aplicação, podemos ignorar os pesos (todas arestas possuem o mesmo peso). Sendo assim, a cada passo, podemos adicionar à arvore qualquer aresta de forma aleatória, a menos que esta forme um ciclo.

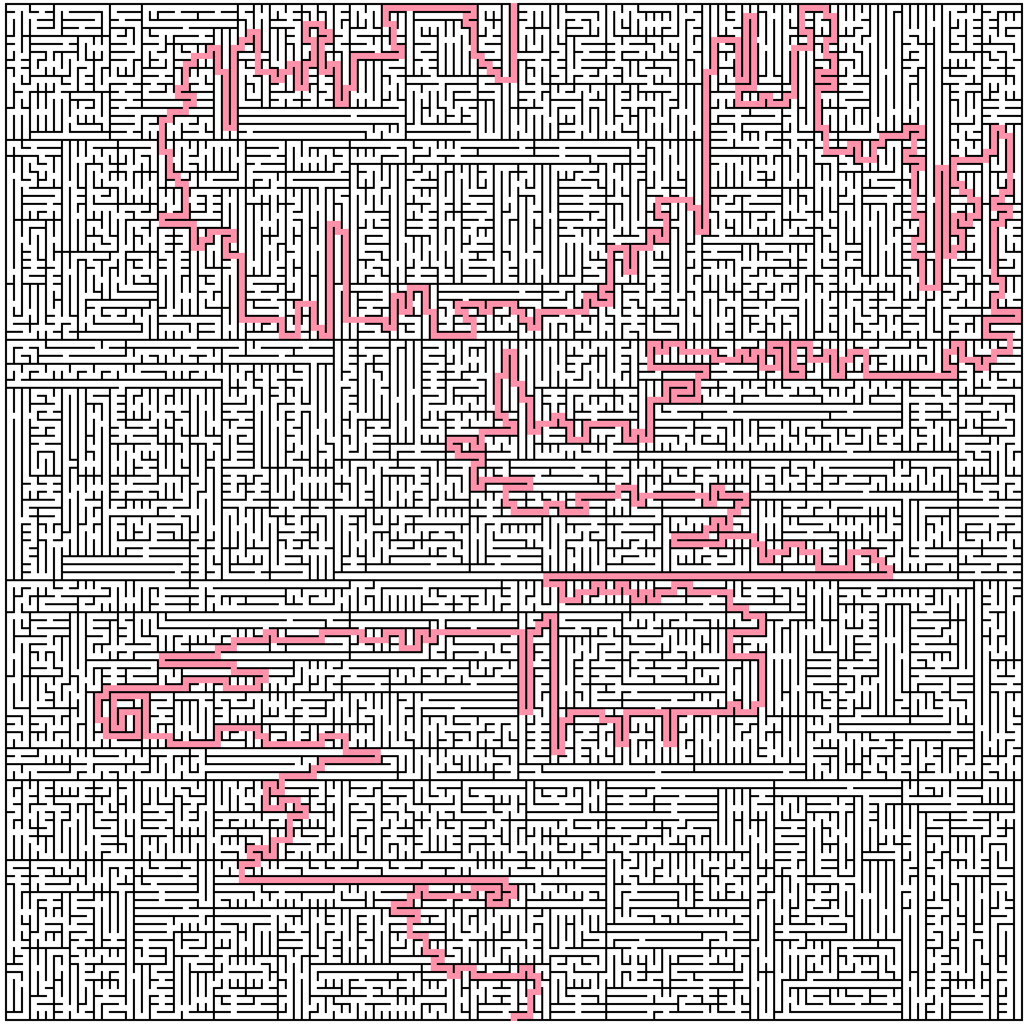


Algoritmo de Kruskal



Uma vez gerado, como resolvê-lo?

Backtracking



Backtracking

Backtracking é uma técnica de resolução de problemas que explora, a princípio, todas as soluções possíveis de dado problema. Algumas restrições podem ser impostas para que o programa não simule de fato todas estas soluções.

Alguns problemas que podem ser resolvidos com backtracking:

- Resolução de um labirinto
- Geração de todas as combinações e permutações de uma sequência
- Resolução do problema das N rainhas
- Resolução de um jogo de Sudoku

Backtracking

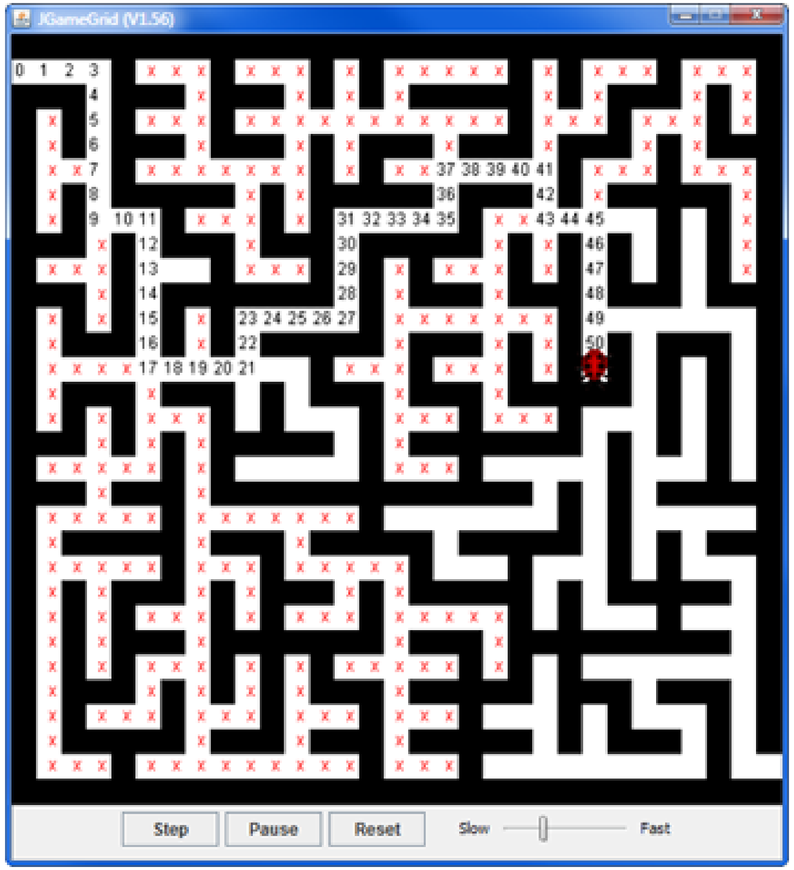
A ideia para o labirinto é, literalmente, testar todos os caminhos possíveis, parando de testar um caminho assim que ele não for mais possível

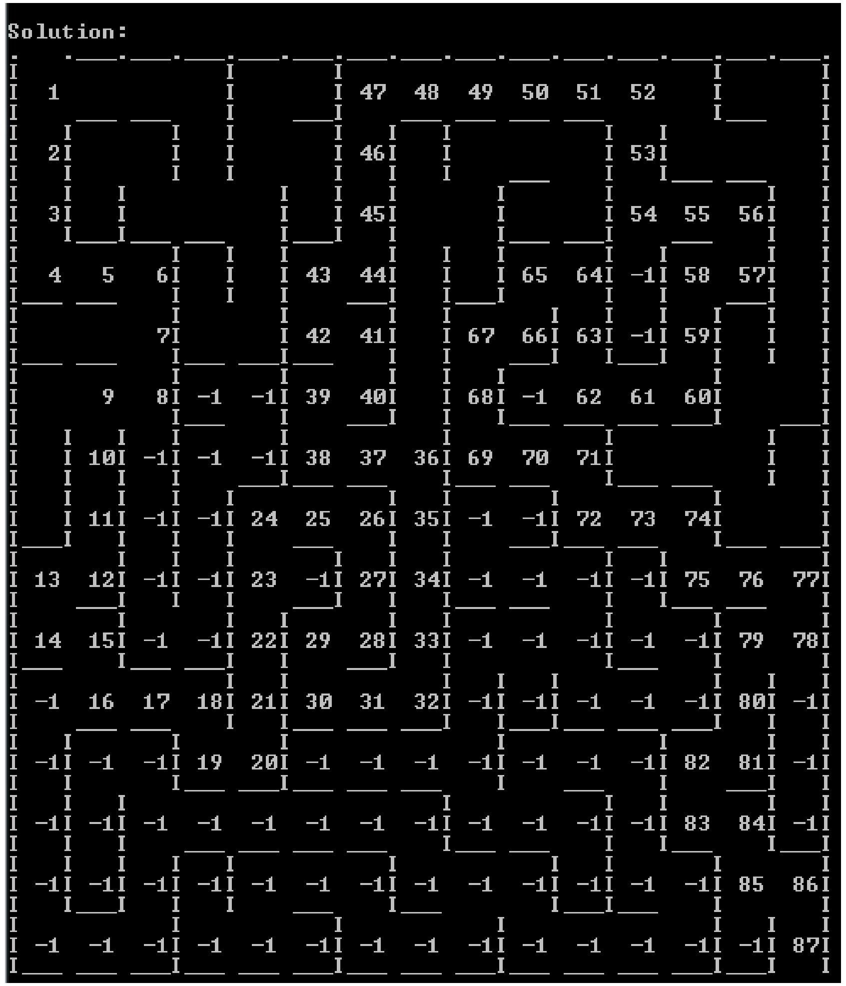
Para cada caminho parcialmente tomado, todas as possibilidades a partir dele são exploradas.

Do início, temos que o algoritmo é o seguinte:

1. Explora um caminho até chegar ao final ou não ser mais possível avançar;
2. Se não tiver chegado ao final, retorna e tenta outro caminho;
3. Se não tiver chegado ao final, retorna ao passo 2 até chegar;
4. Se todas as tentativas forem simuladas e nenhuma solução for encontrada, o labirinto é impossível.

Backtracking





Sudoku com backtracking

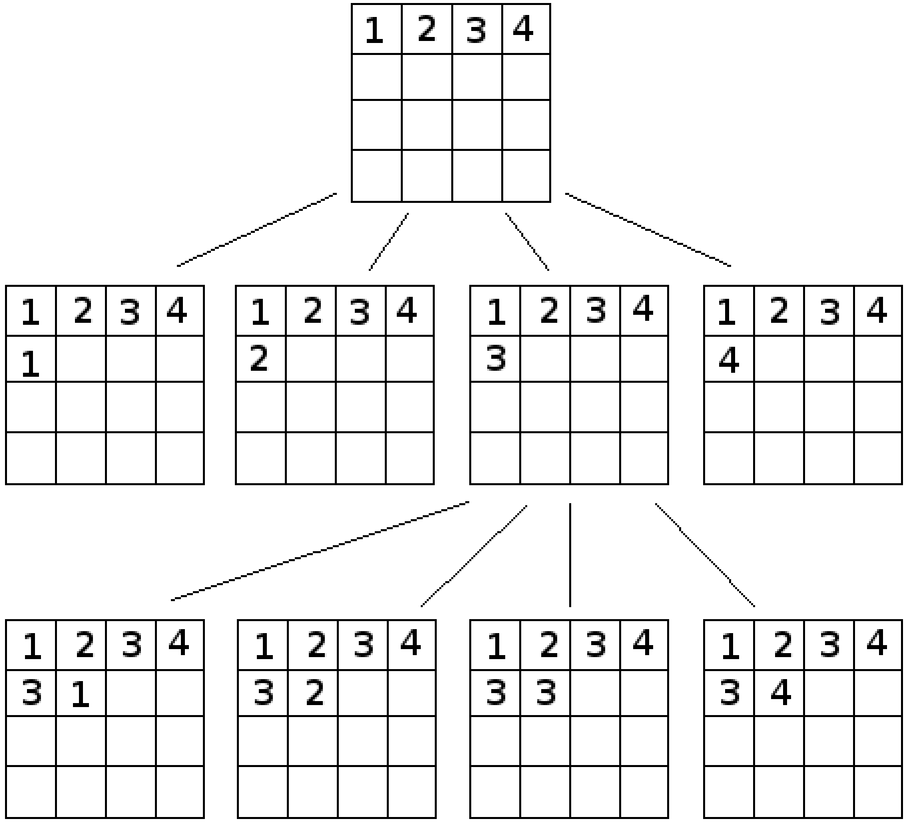
Como visto anteriormente, outros problemas podem ser resolvidos com esta técnica. Veremos como resolver o jogo de Sudoku:

Sudoku com backtracking

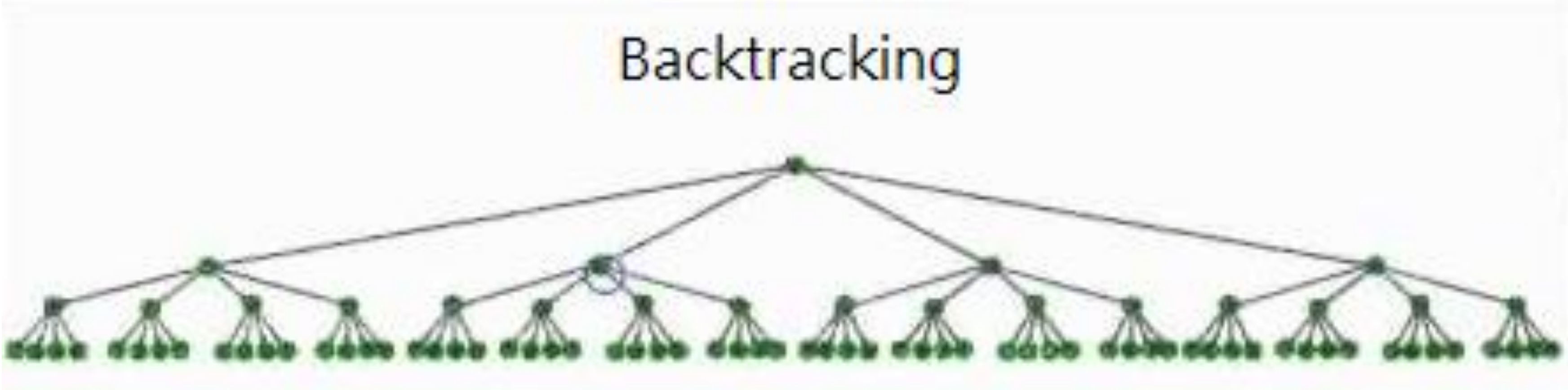
O algoritmo de resolução é o seguinte:

1. Procura a primeira célula vazia do tabuleiro e testa todos os valores possíveis nela, tais que não violem as regras do jogo (números repetidos na mesma linha, coluna ou quadrado 3x3);
2. Para cada valor atribuído à célula anterior, procura a próxima célula vazia e testa todos os valores possíveis nela, tais que não violem as regras do jogo;
3. Repete o passo 2 até que não haja mais células vazias (jogo ganho) ou não haver mais possibilidade de tentativa (jogo impossível).

Sudoku com backtracking



Sudoku com backtracking



Árvore de tentativas de um problema genérico envolvendo backtracking

Sudoku com backtracking

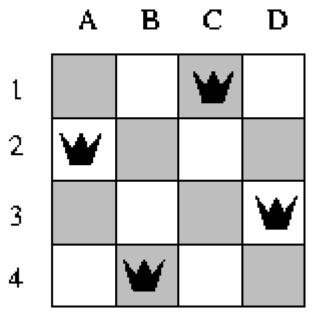
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Problema das N rainhas

Como último exemplo, o problema das N rainhas:

“Dado um tabuleiro de xadrez NxN, colocar N rainhas de tal forma que nenhuma delas possa atacar outra, ou seja, que nenhuma delas esteja na mesma linha, coluna e diagonal de outra”



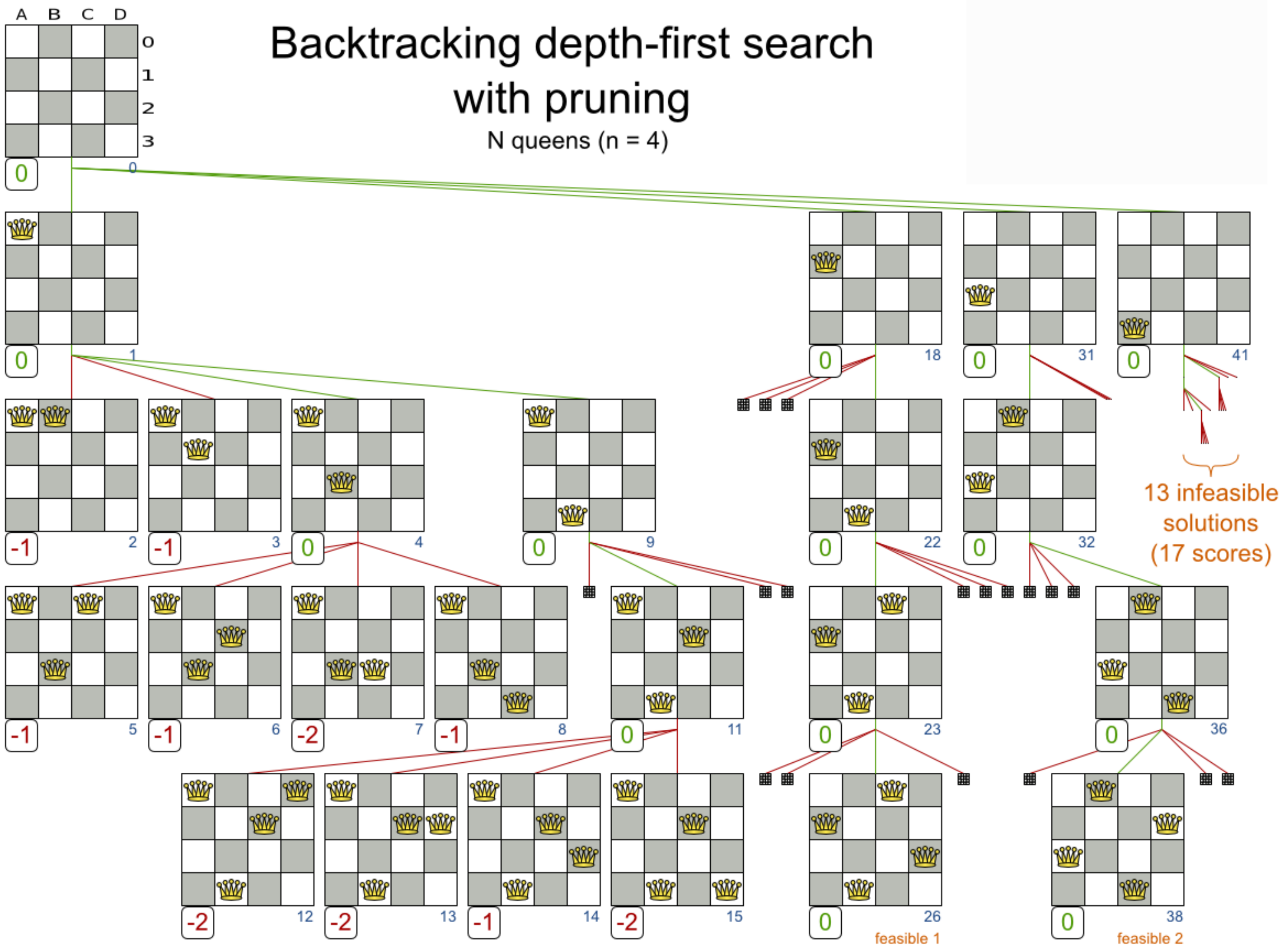
Exemplo para N = 4

Problema das N rainhas

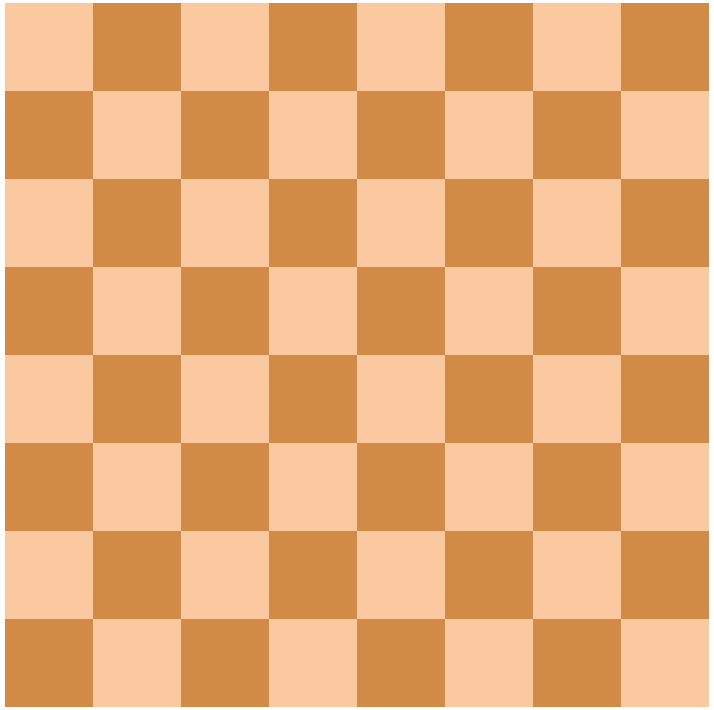
A ideia é a mesma: ir tentando todas as combinações possíveis, enquanto elas não violarem as regras do problema:

1. Comece na primeira linha. Coloque a rainha na primeira casa;
2. Avance para a próxima linha. Veja todas as possíveis casas para a rainha ficar e coloque-a;
3. Para cada configuração obtida anteriormente, avance para a próxima linha. Veja todas as possíveis casas para a rainha ficar e coloque-a;
4. Repita o passo 3 até completar as N linhas ou não ser mais possível inserir novas peças;
5. Se não for possível, volte ao passo 2 e tente novas configurações;
6. Se não for possível, volte ao passo 1 e tente novas configurações.

Problema das N rainhas



Problema das N rainhas



Animação para N = 8

Problema das N rainhas

```
# ####
### ##
##### 
 #####
## ###
#### #
```

Solução de N = 6

Obrigado!!

Competições
de Algoritmos
e Programação
UFRJ