Fatec
Rio Claro

Estrutura de Dados

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Grafos

Fatec
Rio Claro

Grafo é uma estrutura de dados não linear, na qual o problema é representado como uma rede conectando um conjunto de nós com bordas, como uma rede telefônica ou rede social.

Por exemplo, em um grafo, os nós podem representar cidades diferentes, enquanto os links entre eles representam arestas.

São usados para resolver muitos problemas de computação, especialmente quando o problema é representado na forma de objetos e sua conexão, por exemplo, para descobrir o caminho mais curto de uma cidade para outra aos quais o problema pode ser representado como uma estrutura semelhante a uma rede.

Grafos

Grafo

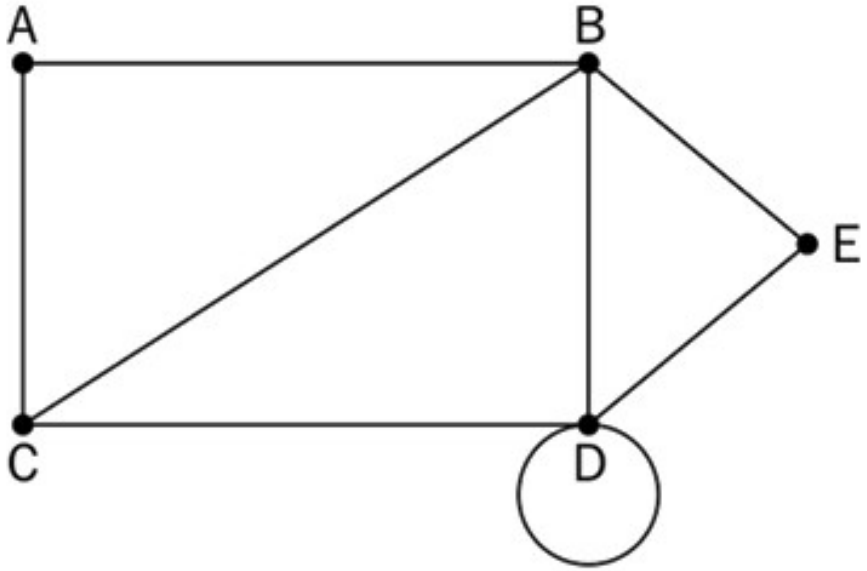
Fatec
Rio Claro

Um grafo é um conjunto de um número finito de vértices (também conhecidos como nós) e arestas, em que as arestas são as ligações entre os vértices, e cada aresta em um grafo une dois nós distintos.

Além disso, um grafo é uma representação matemática formal de uma rede, ou seja, um grafo G é um par ordenado de um conjunto V de vértices e um conjunto E de arestas, dado como G = (V, E) em notação matemática formal.

Grafo

Fatec
Rio Claro



O gráfico G = (V, E) pode ser descrito como abaixo:

V = {A, B, C, D, E}

E = {{A, B}, {A, C}, {B, C}, {B, D}, {C, D}, {D, D}, {B, E}, {D, E}}

G = (V, E)

Definições de Grafo

Fatec
Rio Claro

Nó ou vértice: Um ponto ou nó em um grafo é chamado de vértice. No diagrama anterior, os vértices ou nós são A, B, C, D e E e são denotados por um ponto.

Aresta: Esta é uma conexão entre dois vértices. A reta que conecta A e B é um exemplo de aresta.

Laço: Quando uma aresta de um nó retorna a si mesma, essa aresta forma um laço, por exemplo, o nó D.

Grau de um vértice/nó: O número total de arestas incidentes em um determinado vértice é chamado de grau desse vértice. Por exemplo, o grau do vértice B no diagrama anterior é 4.

Definições de Grafo

Fatec
Rio Claro

Adjacência: Refere-se à(s) conexão(ões) entre quaisquer dois nós; portanto, se houver uma conexão entre quaisquer dois vértices ou nós, então eles são considerados adjacentes um ao outro. Por exemplo, o nó C é adjacente ao nó A porque há uma aresta entre eles.

Caminho: Uma sequência de vértices e arestas entre quaisquer dois nós representa um caminho. Por exemplo, CABE representa um caminho do nó C ao nó E.

Vértice folha (também chamado de vértice pendente): Um vértice ou nó é chamado de vértice folha ou vértice pendente se tiver exatamente um grau.

Grafos direcionados e não direcionados

Fatec
Rio Claro

Grafos são representados pelas arestas entre os nós. As arestas de conexão podem ser consideradas direcionadas ou não direcionadas.

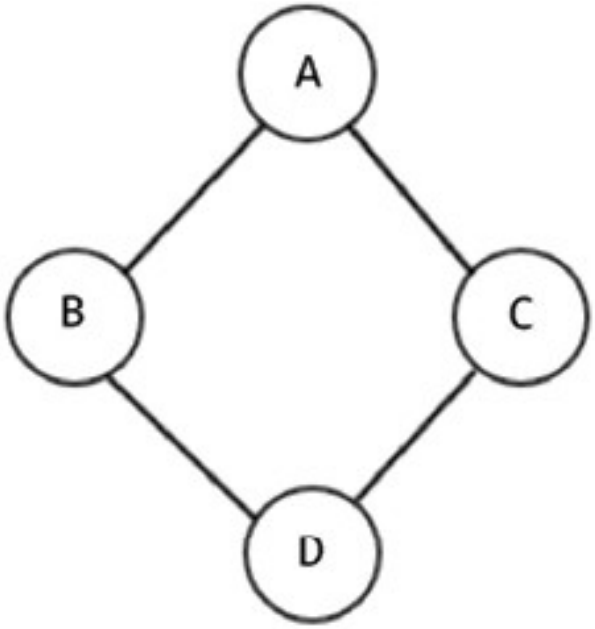
Se as arestas de conexão em um grafo são não direcionadas, então o grafo é chamado de grafo não direcionado, e se as arestas de conexão em um grafo são direcionadas, então ele é chamado de grafo direcionado.

Um grafo não direcionado simplesmente representa as arestas como linhas entre os nós. Não há nenhuma informação adicional sobre a relação entre os nós, além do fato de que eles estão conectados.

Grafos direcionados e não direcionados

Fatec
Rio Claro

Por exemplo, na imagem a seguir, demonstramos um grafo não direcionado de quatro nós, A, B, C e D, que são conectados por arestas



Grafos direcionados e não direcionados

Fatec
Rio Claro

Em um grafo direcionado, as arestas fornecem informações sobre a direção da conexão entre quaisquer dois nós em um grafo.

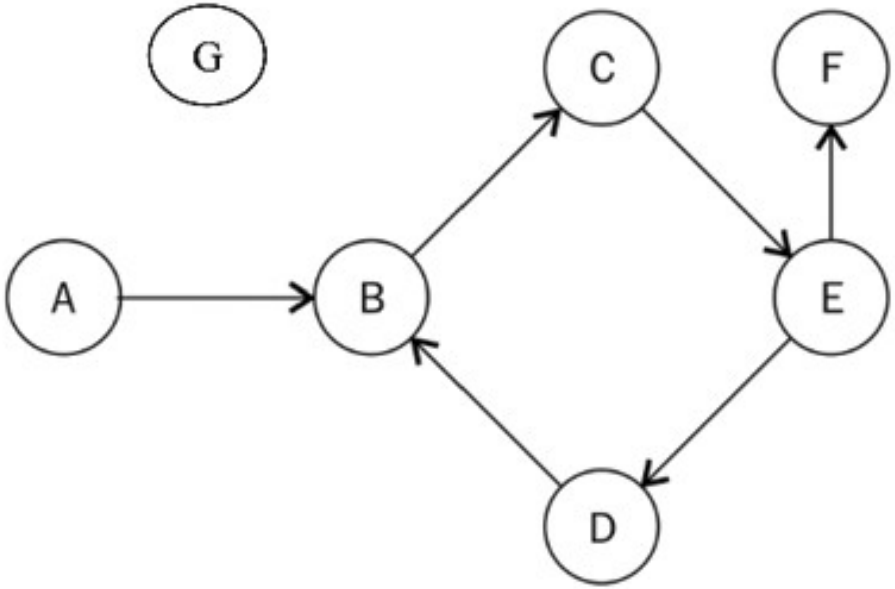
Se uma aresta do nó A ao nó B for dita direcionada, então a aresta (A, B) não seria igual à aresta (B, A).

As arestas direcionadas são desenhadas como linhas com setas, que apontarão para qualquer direção em que a aresta conecte os dois nós.

Grafos direcionados e não direcionados

Fatec Rio Claro

Por exemplo, na imagem a seguir, mostramos um grafo direcionado onde muitos nós são conectados por arestas direcionadas:



Grafos direcionados e não direcionados

Fatec
Rio Claro

A seta de uma aresta determina o fluxo da direção. Só é possível mover-se de A para B, como mostrado no diagrama anterior — não de B para A. Em um grafo direcionado, cada nó (ou vértice) tem um grau de entrada e um grau de saída.

Grau de entrada: O número total de arestas que entram em um vértice no grafo é chamado de grau de entrada desse vértice. Por exemplo, no diagrama anterior, o nó E tem 1 grau de entrada, devido à aresta CE entrar no nó E.

Grau de saída: O número total de arestas que saem de um vértice no grafo é chamado de grau de saída desse vértice. Por exemplo, o nó E no diagrama anterior tem um grau de saída 2, pois tem duas arestas, EF e ED, saindo desse nó.

Grafos direcionados e não direcionados

Fatec
Rio Claro

Vértice isolado: Um nó ou vértice é chamado de vértice isolado quando tem grau zero, como mostrado pelo nó G na imagem anterior.

Vértice de origem: Um vértice é chamado de vértice de origem se tiver um grau de entrada igual a zero. Por exemplo, no diagrama anterior, o nó A é o vértice de origem.

Vértice de destino: Um vértice é um vértice de destino se tiver um grau de saída igual a zero. Por exemplo, no diagrama anterior, o nó F é o vértice de destino.

Prática

Fatec
Rio Claro

Pense e compartilhe um exemplo de problema que pode ser modelado com grafos.

Grafos acíclicos direcionados

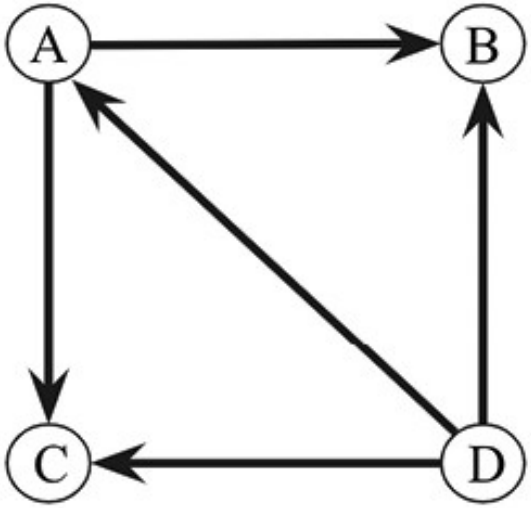
Fatec
Rio Claro

Um grafo acíclico direcionado (*directed acyclic graph* (DAG) é um grafo direcionado sem ciclos; em um DAG, todas as arestas são direcionadas de um nó para outro, de modo que a sequência de arestas nunca forma um laço fechado.

Um ciclo em um grafo é formado quando o nó inicial da primeira aresta é igual ao nó final da última aresta de uma sequência.

Grafos acíclicos direcionados

Fatec
Rio Claro



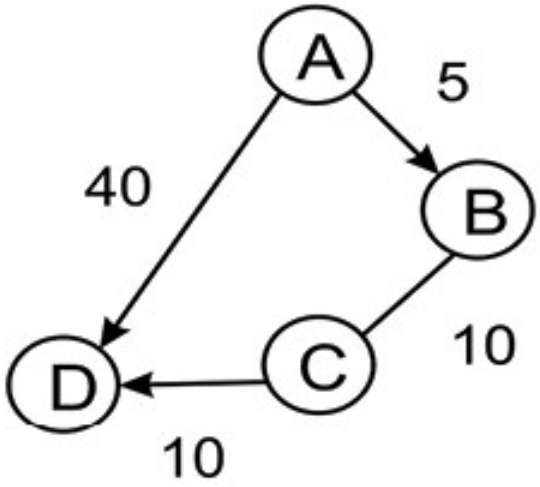
Todas as arestas do grafo são direcionadas e o grafo não possui ciclos.

Portanto, em um grafo acíclico direcionado, se começarmos em qualquer caminho a partir de um determinado nó, nunca encontraremos um caminho que termine no mesmo nó. Um DAG tem muitas aplicações, como em agendamento de tarefas, grafos de citações e compressão de dados.

Grafos ponderados

Fatec
Rio Claro

Um grafo ponderado é um grafo que possui um peso numérico associado às arestas do grafo. Um grafo ponderado pode ser direcionado ou não direcionado. O peso numérico pode ser usado para indicar distância ou custo, dependendo da finalidade do grafo.



Grafos ponderados

Fatec Rio Claro

A imagem anterior indica diferentes maneiras de chegar do nó A ao nó D.

Existem dois caminhos possíveis, como do nó A ao nó D, ou podem ser os nós A-B-C-D através do nó B e do nó C.

Agora, dependendo dos pesos associados às arestas, qualquer um dos caminhos pode ser considerado melhor do que os outros para a jornada – por exemplo, suponha que os pesos neste gráfico representam a distância entre dois nós, e queremos descobrir o caminho mais curto entre os nós A-D; então um caminho possível A-D tem um custo associado de 40, e outro caminho possível A-B-C-D tem um custo associado de 25.

Neste caso, o melhor caminho é A-B-C-D, que tem uma distância menor.

Grafos bipartidos

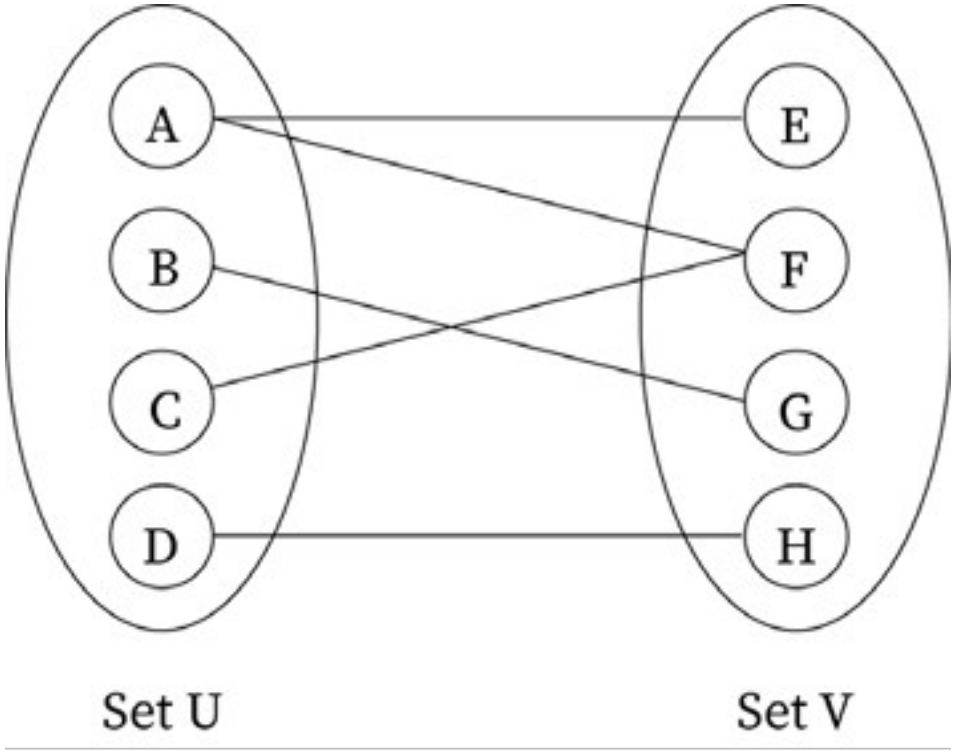
Fatec
Rio Claro

Um grafo bipartido (também conhecido como bigrafo) é um grafo especial no qual todos os nós do grafo podem ser divididos em dois conjuntos de tal forma que as arestas conectam os nós de um conjunto aos nós de outro conjunto.

Na próxima imagem há um exemplo de grafo bipartido; todos os nós dos grafos são divididos em dois conjuntos independentes, ou seja, conjunto U e conjunto V, de modo que cada aresta do grafo tenha uma extremidade no conjunto U e outra extremidade no conjunto V (por exemplo, na aresta (A, B), uma extremidade ou um vértice é do conjunto U e a outra extremidade ou outro vértice é do conjunto V).

Grafos bipartidos

Fatec
Rio Claro



Grafos bipartidos

Fatec
Rio Claro

Em grafos bipartidos, nenhuma aresta se conectará aos nós do mesmo conjunto.

Os grafos bipartidos são úteis quando precisamos modelar um relacionamento entre duas classes diferentes de objetos, por exemplo, um grafo de candidatos e empregos, no qual podemos precisar modelar o relacionamento entre esses dois grupos diferentes; outro exemplo pode ser um gráfico bipartido de jogadores de futebol e clubes, no qual podemos precisar modelar se um jogador jogou por um clube específico ou não.

Representação de Grafos

Representações de grafos

Fatec
Rio Claro

Uma técnica de representação de grafos significa como armazenamos o grafo na memória, ou seja, como armazenamos os vértices, arestas e pesos (se o grafo for ponderado).

Uma representação de **lista de adjacências** é baseada em uma lista encadeada. Nela, representamos o grafo mantendo uma lista de vizinhos (também chamada de nó adjacente) para cada vértice (ou nó) do grafo.

Em uma representação de **matriz de adjacência** de um grafo, mantemos uma matriz que representa qual nó é adjacente a qual outro nó no grafo; ou seja, a matriz de adjacência contém as informações de cada aresta do grafo, que são representadas pelas células da matriz.

Representações de grafos

Fatec Rio Claro

Qualquer uma dessas duas representações pode ser usada; no entanto, nossa escolha depende da aplicação em que usaremos a representação de grafos.

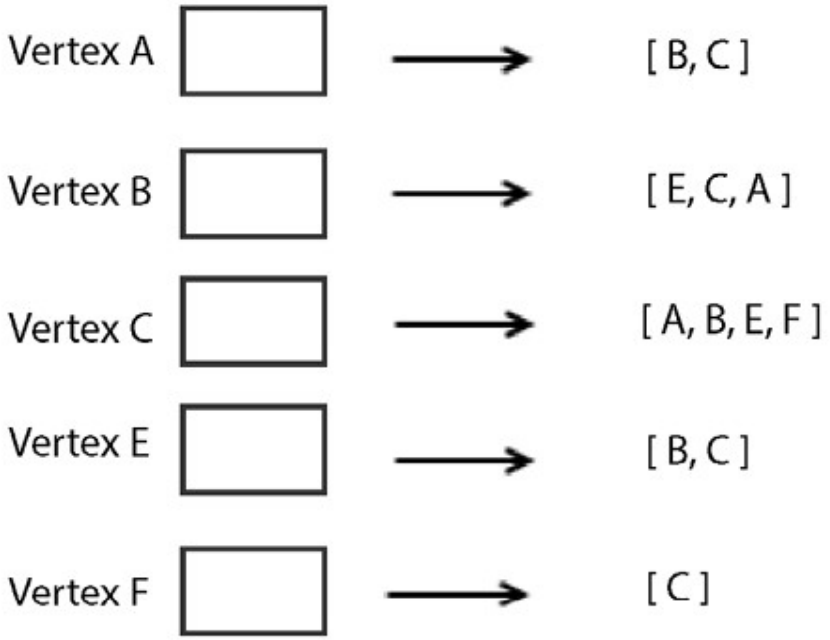
Uma lista de adjacência é preferível quando esperamos que o grafo seja esparso e tenhamos um número menor de arestas; por exemplo, se um grafo de 200 nós tiver, digamos, 100 arestas, é melhor armazenar esse tipo de grafo em uma lista de adjacência, porque se usarmos uma matriz de adjacência, o tamanho da matriz será 200x200 com muitos valores zero.

A matriz de adjacência é preferível quando esperamos que o grafo tenha muitas arestas e a matriz seja densa. Na matriz de adjacência, a consulta e a verificação da presença ou ausência de uma aresta são muito fáceis em comparação com a representação em lista de adjacência.

Listas de Adjacência

Fatec
Rio Claro

Uma lista encadeada pode ser usada para implementar a lista de adjacência. Para representar o grafo, precisamos que o número de listas encadeadas seja igual ao número total de nós no grafo. Em cada índice, os nós adjacentes àquele vértice são armazenados.



Listas de Adjacência

Fatec Rio Claro

Usar uma lista para a representação é bastante restritivo, pois não temos a capacidade de usar diretamente os rótulos dos vértices. Portanto, para implementar um grafo de forma eficiente usando Python, uma estrutura de dados de dicionário é usada, pois é mais adequada para representar o grafo.

```
graph = dict()
graph['A'] = ['B', 'C']
graph['B'] = ['E','C', 'A']
graph['C'] = ['A', 'B', 'E','F']
graph['E'] = ['B', 'C']
graph['F'] = ['C']
```

Matriz de adjacência

Fatec
Rio Claro

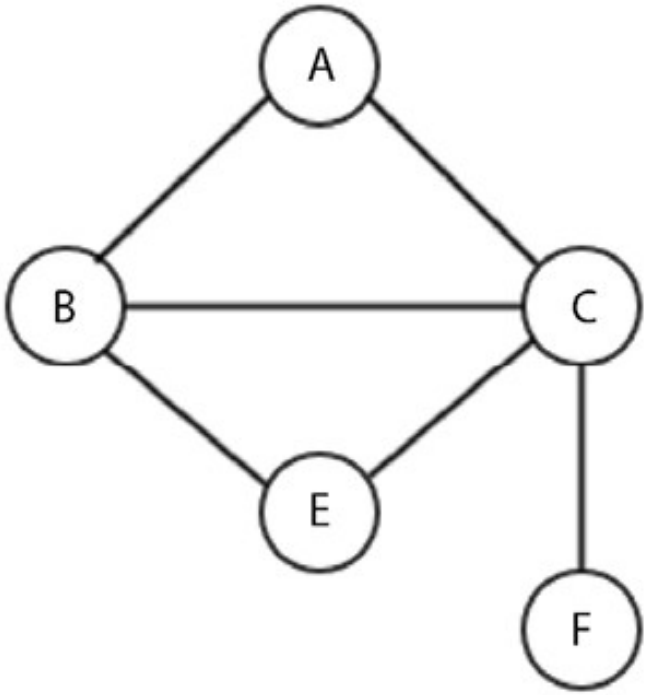
Outra abordagem para representar um grafo é usar uma matriz de adjacência.

Nela, o grafo é representado mostrando os nós e suas interconexões por meio de arestas. Usando esse método, as dimensões (V x V) de uma matriz são usadas para representar o grafo, onde cada célula denota uma aresta no grafo. Uma matriz é um arranjo bidimensional.

Portanto, a ideia aqui é representar as células da matriz com um 1 ou um 0, dependendo se dois nós estão conectados por uma aresta ou não.

Matriz de adjacência

Fatec
Rio Claro



Adjacency Matrix

	A	B	C	E	F
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	1
E	0	1	1	0	0
F	0	0	1	0	0

Prática

Fatec
Rio Claro

Observe o código **graph.py**

Representações de grafos

Fatec
Rio Claro

O uso da matriz de adjacência para representação de grafos é adequado quando precisamos consultar e verificar frequentemente a presença ou ausência de uma aresta entre dois nós no grafo, por exemplo, na criação de tabelas de roteamento em redes, na busca de rotas em aplicativos de transporte público e sistemas de navegação, etc.

Matrizes de adjacência não são adequadas quando nós são frequentemente adicionados ou excluídos dentro de um grafo; nessas situações, a lista de adjacências é uma técnica mais adequada.

Percurso em Grafos

Percurso de grafos

Fatec
Rio Claro

Um percurso em grafos significa visitar todos os vértices do grafo, mantendo o controle de quais nós ou vértices já foram visitados e quais não foram. Um algoritmo de percurso em grafos é eficiente se percorrer todos os nós do grafo no menor tempo possível.

O percurso em grafos, também conhecido como algoritmo de busca em grafos, é bastante semelhante aos algoritmos de percurso em árvores, como os algoritmos de pré-ordem, ordem interna, pós-ordem e ordem de níveis; semelhante a eles, em um algoritmo de busca em grafos, começamos com um nó e percorremos as arestas até todos os outros nós do grafo.

Percurso de grafos

Fatec
Rio Claro

Uma estratégia comum de percurso em grafos é seguir um caminho até chegar a um beco sem saída e, em seguida, percorrer de volta até encontrar um ponto onde encontramos um caminho alternativo.

Também podemos nos mover iterativamente de um nó para outro para percorrer o grafo inteiro ou parte dele.

Algoritmos de travessia de grafos são muito importantes para responder a muitos problemas fundamentais — eles podem ser úteis para determinar como ir de um vértice a outro em um grafo e qual caminho do nó A ao nó B em um grafo é melhor do que outros caminhos.

Por exemplo, algoritmos de travessia de grafos podem ser úteis para descobrir a rota mais curta de uma cidade a outra em uma rede de cidades.

Breadth-first traversal

Breadth-first search (BFS)

Fatec
Rio Claro

A busca em largura (BFS) funciona de forma muito semelhante a um algoritmo de travessia por ordem de nível em uma estrutura de dados em árvore.

O algoritmo BFS também funciona nível por nível; ele começa visitando o nó raiz no nível 0 e, em seguida, todos os nós do primeiro nível diretamente conectados ao nó raiz são visitados no nível 1.

O nó do nível 1 está a uma distância de 1 do nó raiz. Após visitar todos os nós no nível 1, os nós do nível 2 são visitados em seguida. Da mesma forma, todos os nós no grafo são percorridos nível por nível até que todos os nós sejam visitados. Portanto, os algoritmos de travessia em largura funcionam em largura no grafo.

Breadth-first search (BFS)

Fatec
Rio Claro

Uma estrutura de dados de fila é usada para armazenar as informações dos vértices que devem ser visitados em um grafo.

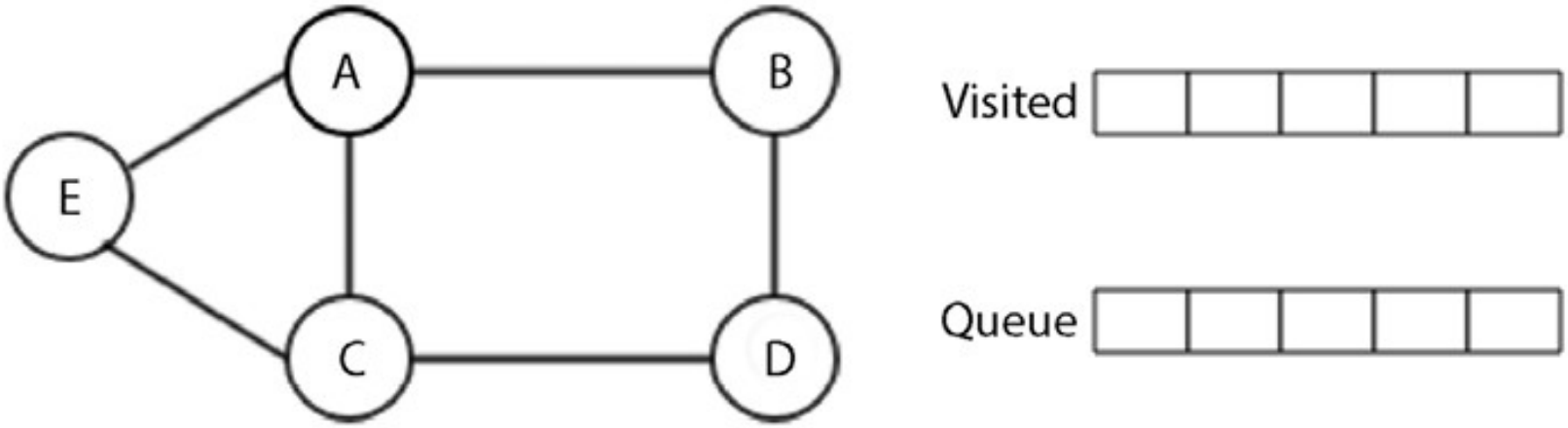
Começamos com o nó inicial. Primeiramente, visitamos esse nó e, em seguida, procuramos todos os seus vértices vizinhos ou adjacentes. Primeiro, visitamos esses vértices adjacentes um por um, enquanto adicionamos seus vizinhos à lista de vértices a serem visitados.

Seguimos esse processo até visitarmos todos os vértices do grafo, garantindo que nenhum vértice seja visitado duas vezes.

Breadth-first search (BFS)

Fatec Rio Claro

Vamos considerar um exemplo para entender melhor o funcionamento da travessia em largura para grafos.



Breadth-first search (BFS)

Fatec Rio Claro

Temos um grafo com cinco nós e uma estrutura de dados de fila para armazenar os vértices a serem visitados.

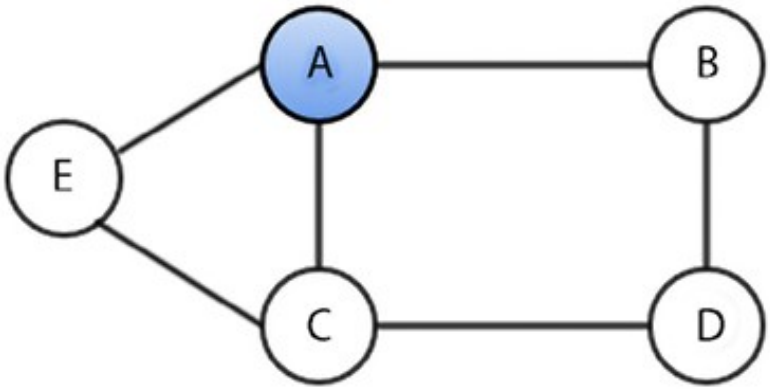
Começamos visitando o primeiro nó, ou seja, o nó A, e então adicionamos todos os seus vértices adjacentes, B, C e E, à fila.

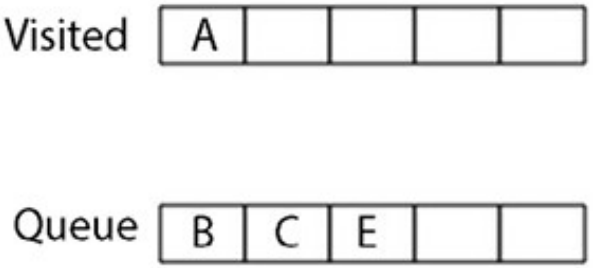
Aqui, é importante observar que existem várias maneiras de adicionar os nós adjacentes à fila, visto que existem três nós, B, C e E, que podem ser adicionados à fila como BCE, CEB, CBE, BEC ou ECB, cada um dos quais nos daria resultados diferentes para a travessia da árvore.

Todas essas soluções possíveis para a travessia do grafo estão corretas, mas neste exemplo, adicionamos os nós em ordem alfabética apenas para manter as coisas simples na fila, ou seja, BCE.

Breadth-first search (BFS)

Fatec
Rio Claro





Breadth-first search (BFS)

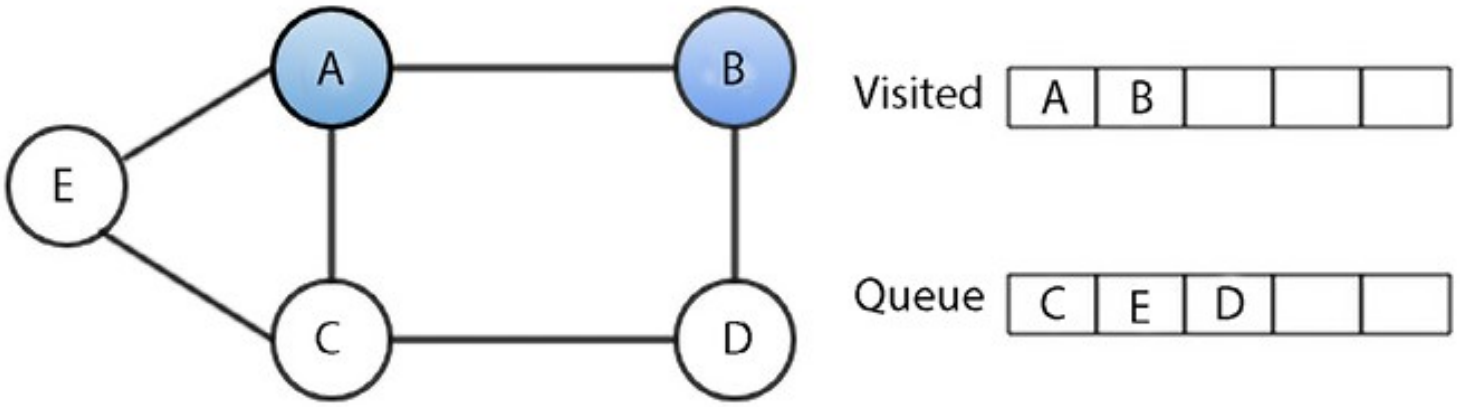
Fatec Rio Claro

Depois de visitar o vértice A, visitamos seu primeiro vértice adjacente, B, e adicionamos os vértices adjacentes do vértice B que ainda não foram adicionados à fila ou não foram visitados.

Nesse caso, precisamos adicionar o vértice D (já que ele possui dois vértices, os nós A e D, dos quais A já foi visitado) à fila.

Breadth-first search (BFS)

Fatec
Rio Claro



Breadth-first search (BFS)

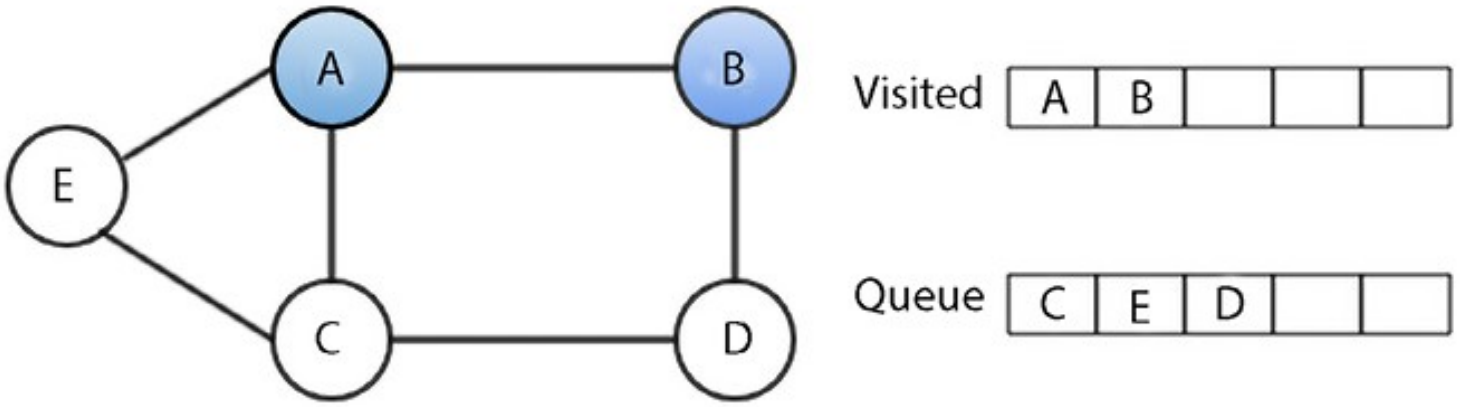
Fatec Rio Claro

Depois de visitar o vértice A, visitamos seu primeiro vértice adjacente, B, e adicionamos os vértices adjacentes do vértice B que ainda não foram adicionados à fila ou não foram visitados.

Nesse caso, precisamos adicionar o vértice D (já que ele possui dois vértices, os nós A e D, dos quais A já foi visitado) à fila.

Breadth-first search (BFS)

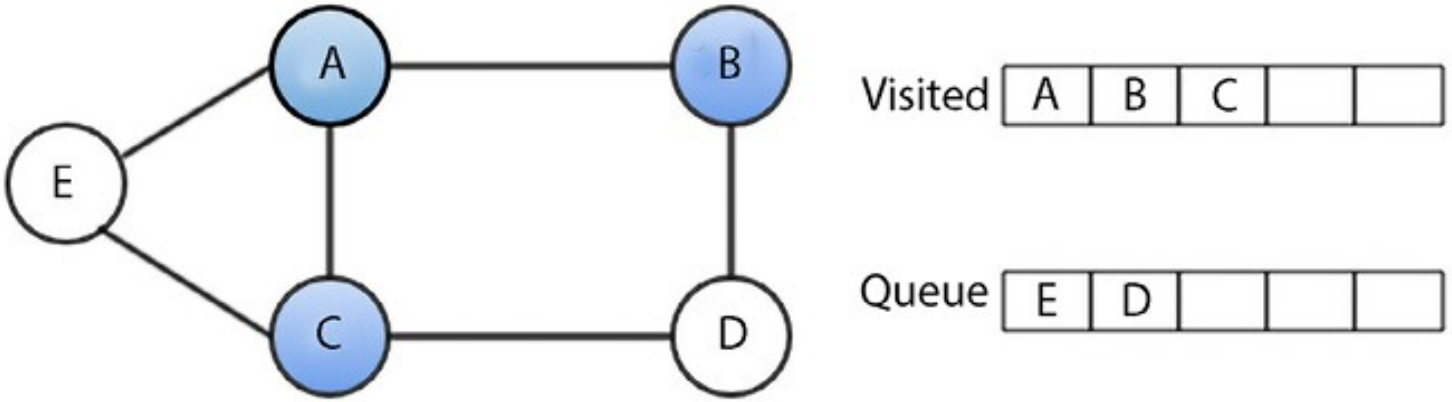
Fatec
Rio Claro



Breadth-first search (BFS)

Fatec
Rio Claro

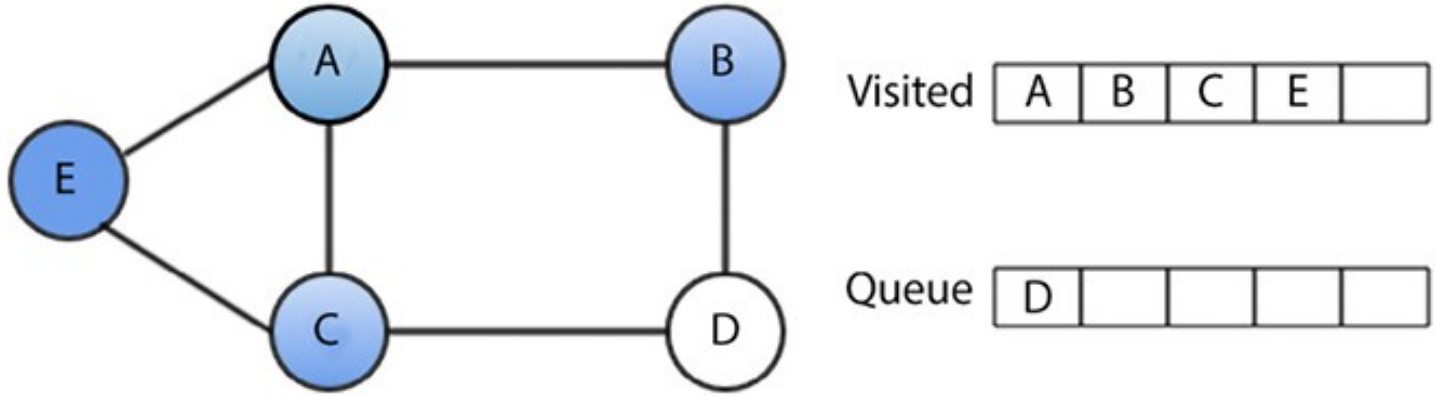
Agora, após visitar o vértice B, visitamos o próximo vértice da fila — o vértice C. E, novamente, adicionamos os vértices adjacentes que ainda não foram adicionados à fila. Neste caso, não há vértices não registrados restantes.



Breadth-first search (BFS)

Fatec Rio Claro

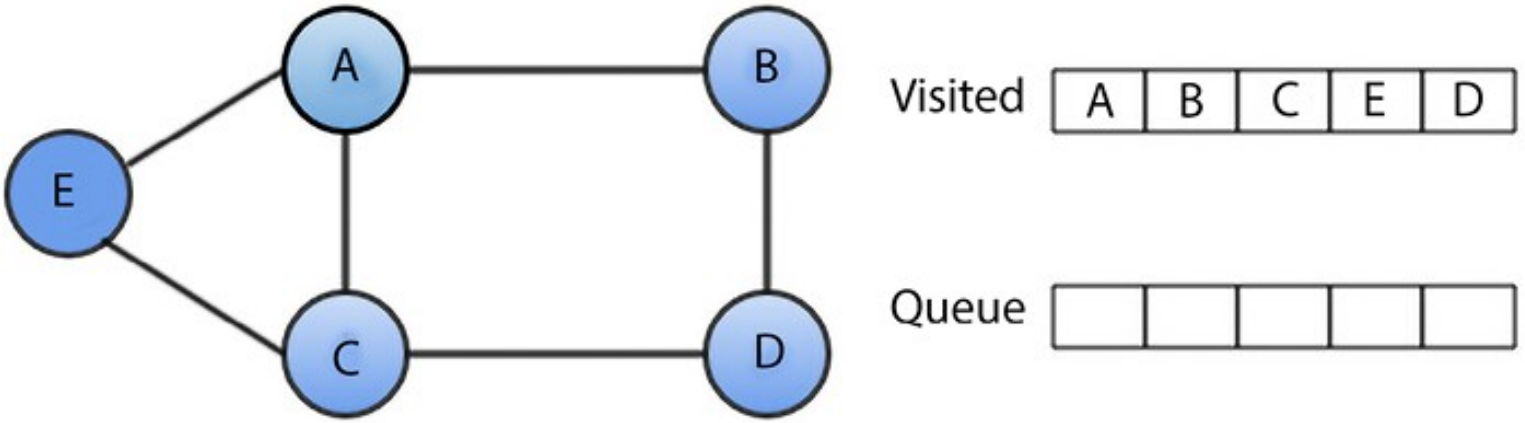
Depois de visitar o vértice C, visitamos o próximo vértice da fila, o vértice E.



Breadth-first search (BFS)

Fatec Rio Claro

Da mesma forma, após visitar o vértice E, visitamos o vértice D na última etapa, conforme mostrado:



Breadth-first search (BFS)

Fatec
Rio Claro

Portanto, o algoritmo BFS para percorrer o grafo anterior visita os vértices na ordem A-B-C-E-D. Esta é uma das soluções possíveis para a travessia BFS do grafo anterior, mas podemos obter muitas soluções possíveis, dependendo de como adicionamos os nós adjacentes à fila.

Prática

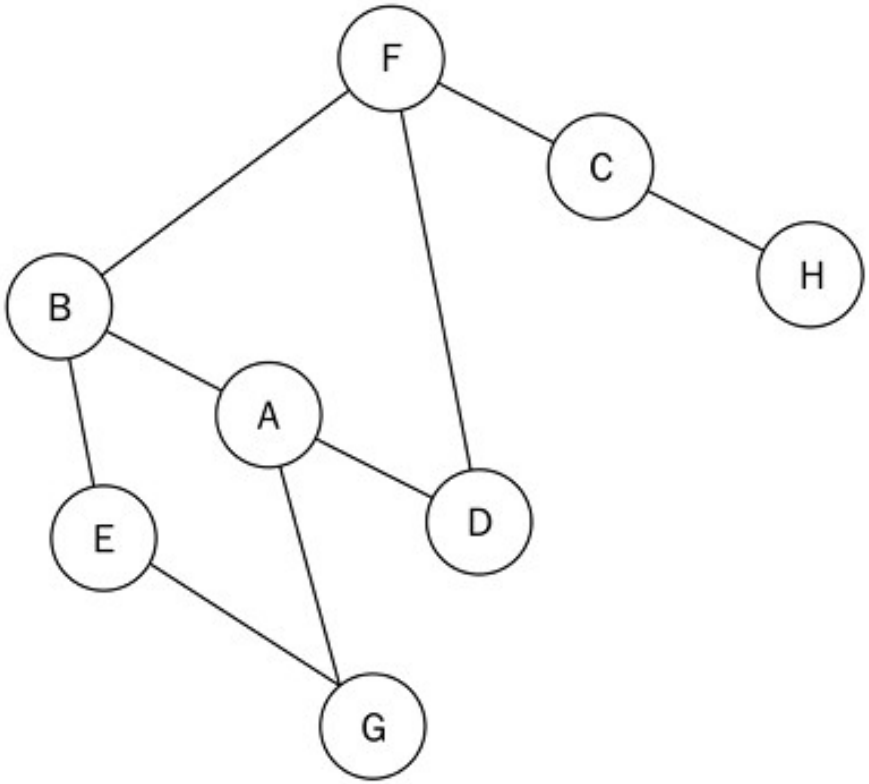
Fatec
Rio Claro

Observe o código **BFS.py**

Breadth-first search (BFS)

Fatec
Rio Claro

Para entender a implementação deste algoritmo em Python, usaremos outro exemplo de um grafo não direcionado



Breadth-first search (BFS)

Fatec Rio Claro

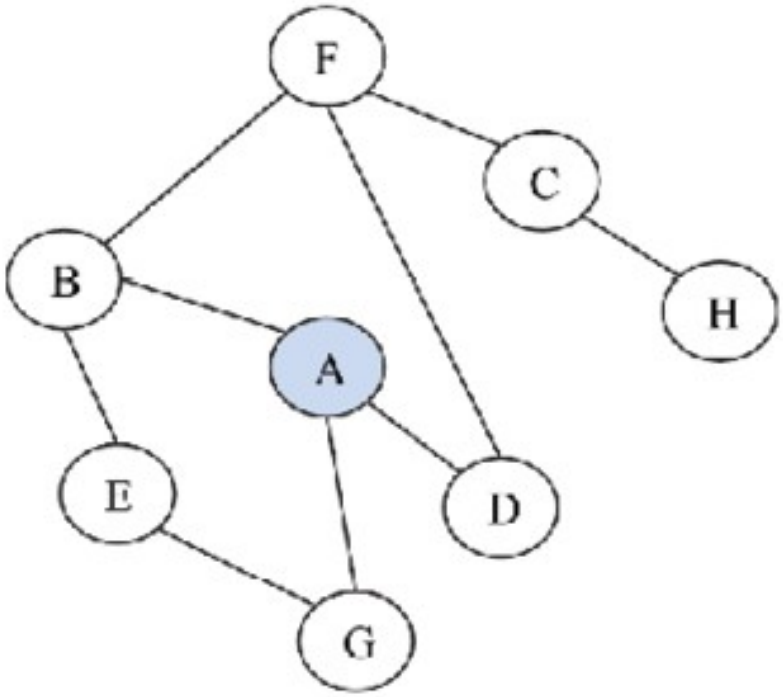
Para percorrer este grafo usando o algoritmo de largura, inicializamos a fila e o nó de origem. Iniciamos a travessia a partir do nó A. Primeiramente, o nó A é enfileirado e adicionado à lista de nós visitados. Em seguida, usamos um laço while para afetar a travessia do grafo.

Na primeira iteração do laço while, o nó A é desenfileirado.

Em seguida, todos os nós adjacentes não visitados do nó A, que são B, D e G, são classificados em ordem alfabética e enfileirados. A fila agora contém os nós B, D e G.

Breadth-first search (BFS)

Fatec
Rio Claro



Visited	A						
Queue	B	D	G				

Breadth-first search (BFS)

Fatec Rio Claro

Para implementação, adicionamos todos esses nós (B, D, G) à lista de nós visitados e, em seguida, adicionamos os nós adjacentes/vizinhos desses nós.

Nesse ponto, iniciamos outra iteração do loop while.

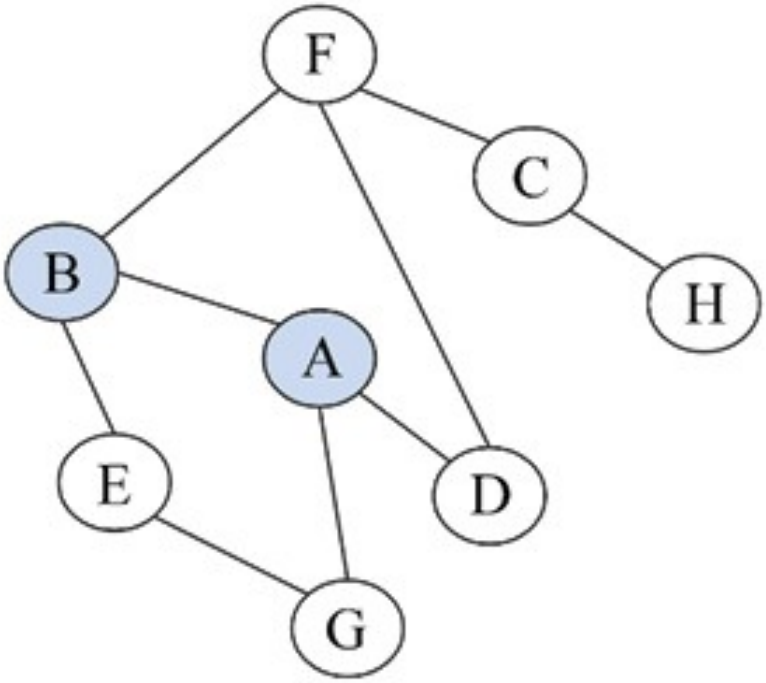
Após visitar o nó A, o nó B é retirado da fila.

Dos seus nós adjacentes (A, E e F), o nó A já foi visitado.

Portanto, enfileiramos apenas os nós E e F em ordem alfabética.

Breadth-first search (BFS)

Fatec
Rio Claro



Visited	A	B					
Queue	D	G	E	F			

Breadth-first search (BFS)

Fatec Rio Claro

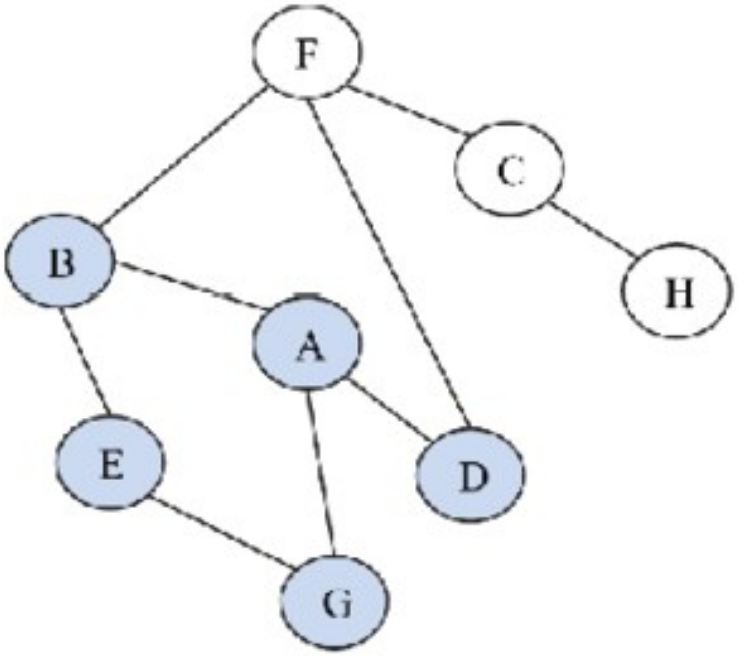
A fila agora contém os seguintes nós neste ponto: D, G, E e F. O nó D é desenfileirado, mas todos os seus nós adjacentes foram visitados, então simplesmente o desenfileiramos.

O próximo nó na frente da fila é G. Desenfileiramos o nó G, mas também descobrimos que todos os seus nós adjacentes foram visitados porque estão na lista de nós visitados.

Portanto, o nó G também é desenfileirado. Desenfileiramos o nó E também porque todos os seus nós adjacentes também foram visitados. O único nó na fila agora é o nó F

Breadth-first search (BFS)

Fatec
Rio Claro

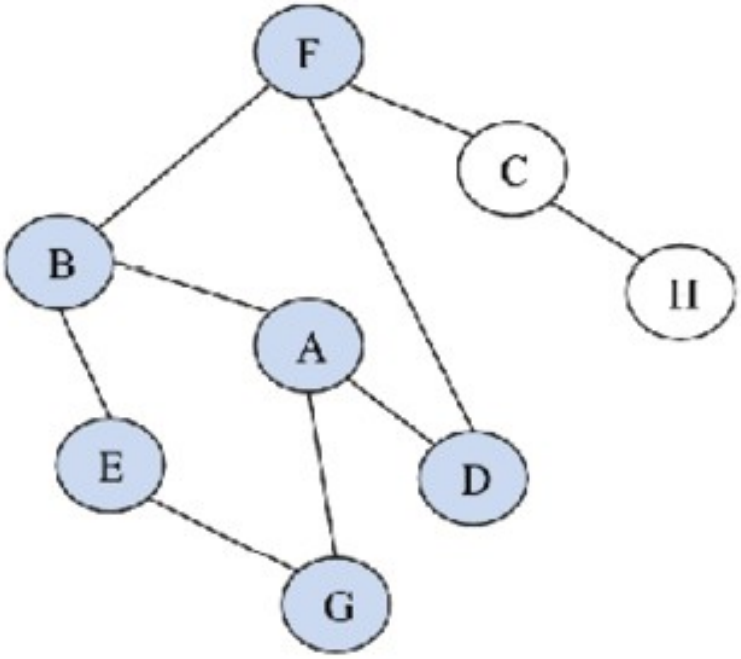


Visited	A	B	D	G	E			
Queue	F							

Breadth-first search (BFS)

Fatec Rio Claro

O nó F é retirado da fila e vemos que, dos nós adjacentes, B, D e C, apenas C não foi visitado. Em seguida, colocamos o nó C na fila e o adicionamos à lista de nós visitados.



Visited	A	B	D	G	E	F	
Queue	C						

Breadth-first search (BFS)

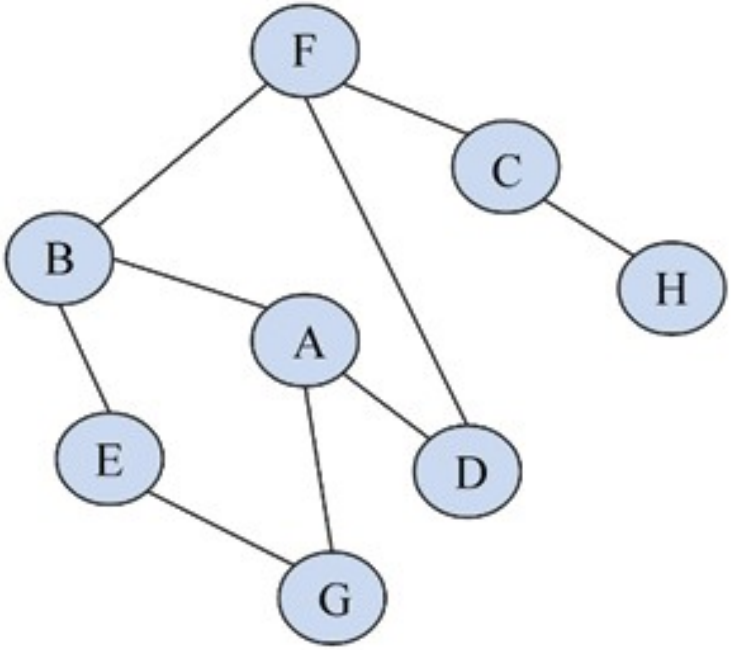
Fatec Rio Claro

Em seguida, o nó C é desenfileirado. C possui os nós adjacentes de F e H, mas F já foi visitado, deixando o nó H. O nó H é enfileirado e adicionado à lista de nós visitados. Finalmente, a última iteração do loop while levará à desenfileiração do nó H.

Seu único nó adjacente, C, já foi visitado. Assim que a fila estiver vazia, o loop será interrompido.

Breadth-first search (BFS)

Fatec
Rio Claro



Visited	A	B	D	G	E	F	C	H
Queue								

Breadth-first search (BFS)

Fatec Rio Claro

O algoritmo BFS é muito útil para construir o caminho mais curto em um grafo com iterações mínimas.

Em algumas aplicações práticas do BFS, ele pode ser usado para criar um rastreador web eficiente, no qual vários níveis de índices podem ser mantidos para mecanismos de busca, e pode manter uma lista de páginas web fechadas a partir de uma página web de origem.

O BFS também pode ser útil para sistemas de navegação nos quais localizações vizinhas podem ser facilmente recuperadas de um grafo de localizações diferentes.

Depth-first traversal

Depth-first search (DFS)

Fatec
Rio Claro

Como o nome sugere, o algoritmo de busca em profundidade (DFS) ou travessia percorre o grafo de forma semelhante ao funcionamento do algoritmo de travessia em pré-ordem em árvores.

No algoritmo DFS, percorremos a árvore na profundidade de qualquer caminho específico no grafo. Assim, os nós filhos são visitados primeiro, antes dos nós irmãos.

Depth-first search (DFS)

Fatec
Rio Claro

Neste caso, começamos com o nó raiz; primeiro o visitamos e, em seguida, vemos todos os vértices adjacentes do nó atual. Começamos visitando um dos nós adjacentes. Se a aresta levar a um nó visitado, retrocedemos para o nó atual. E, se a aresta levar a um nó não visitado, então vamos para esse nó e continuamos o processamento a partir dele. Continuamos o mesmo processo até chegarmos a um beco sem saída, quando não há nenhum nó não visitado; nesse caso, retrocedemos para os nós anteriores e paramos quando alcançamos o nó raiz durante o retrocesso.

Depth-first search (DFS)

Fatec
Rio Claro

Neste caso, começamos com o nó raiz; primeiro o visitamos e, em seguida, vemos todos os vértices adjacentes do nó atual. Começamos visitando um dos nós adjacentes. Se a aresta levar a um nó visitado, retrocedemos para o nó atual. E, se a aresta levar a um nó não visitado, então vamos para esse nó e continuamos o processamento a partir dele. Continuamos o mesmo processo até chegarmos a um beco sem saída, quando não há nenhum nó não visitado; nesse caso, retrocedemos para os nós anteriores e paramos quando alcançamos o nó raiz durante o retrocesso.

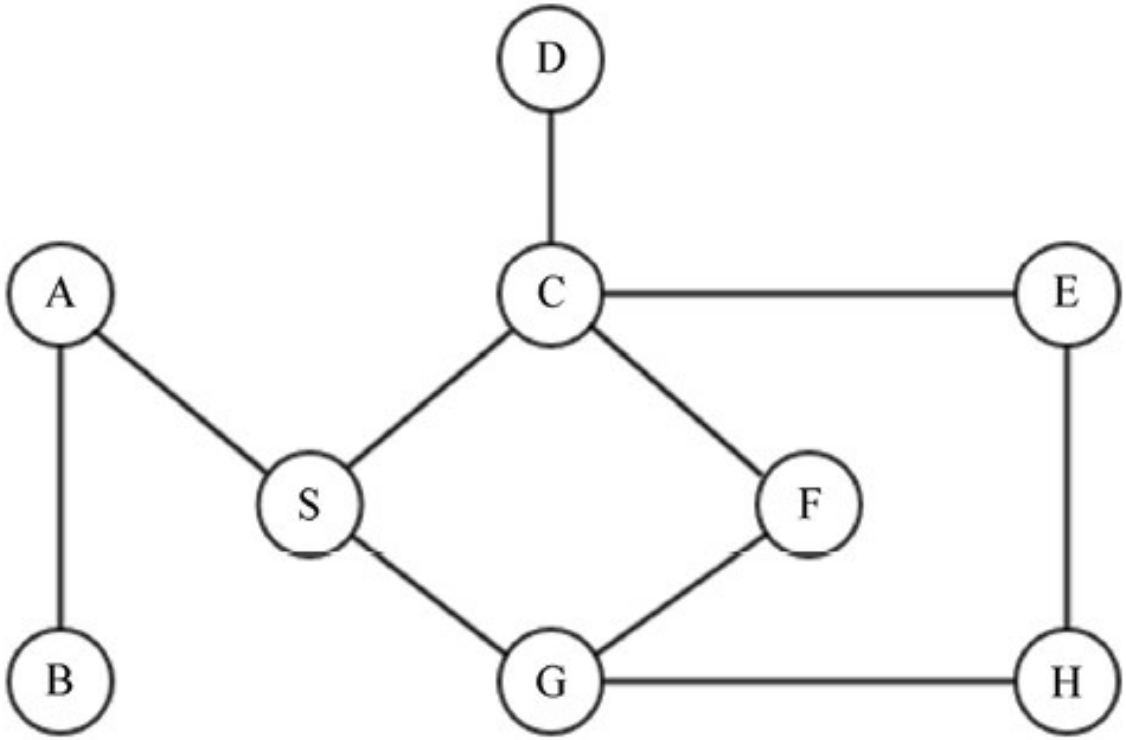
Prática

Fatec
Rio Claro

Observe o código **DFS.py**

Depth-first search (DFS)

Fatec
Rio Claro



Depth-first search (DFS)

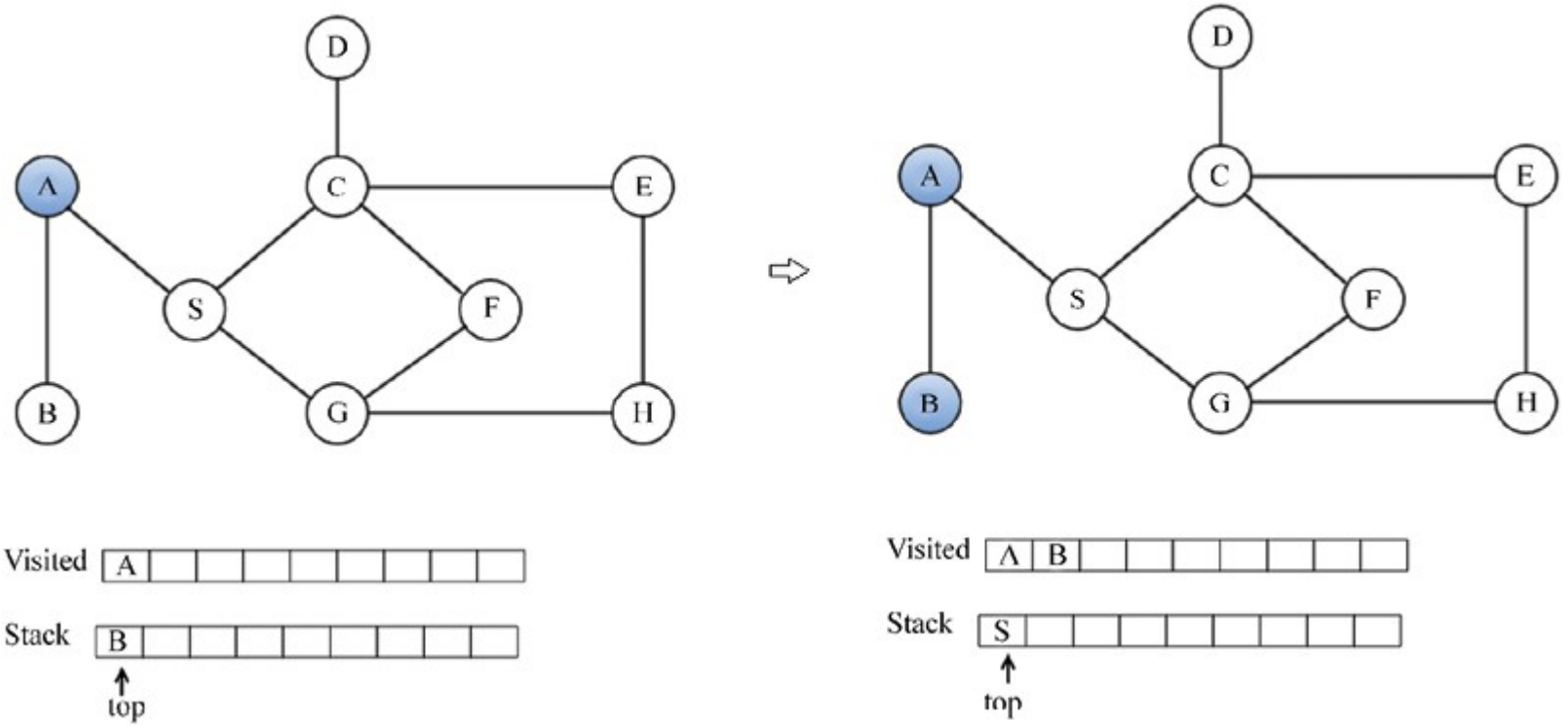
Fatec
Rio Claro

Começamos visitando o nó A e, em seguida, observamos os vizinhos do vértice A, depois um vizinho desse vizinho e assim por diante.

Após visitar o vértice A, visitamos um de seus vizinhos, B (no nosso exemplo, ordenamos em ordem alfabética; no entanto, qualquer vizinho pode ser adicionado).

Depth-first search (DFS)

Fatec
Rio Claro



Depth-first search (DFS)

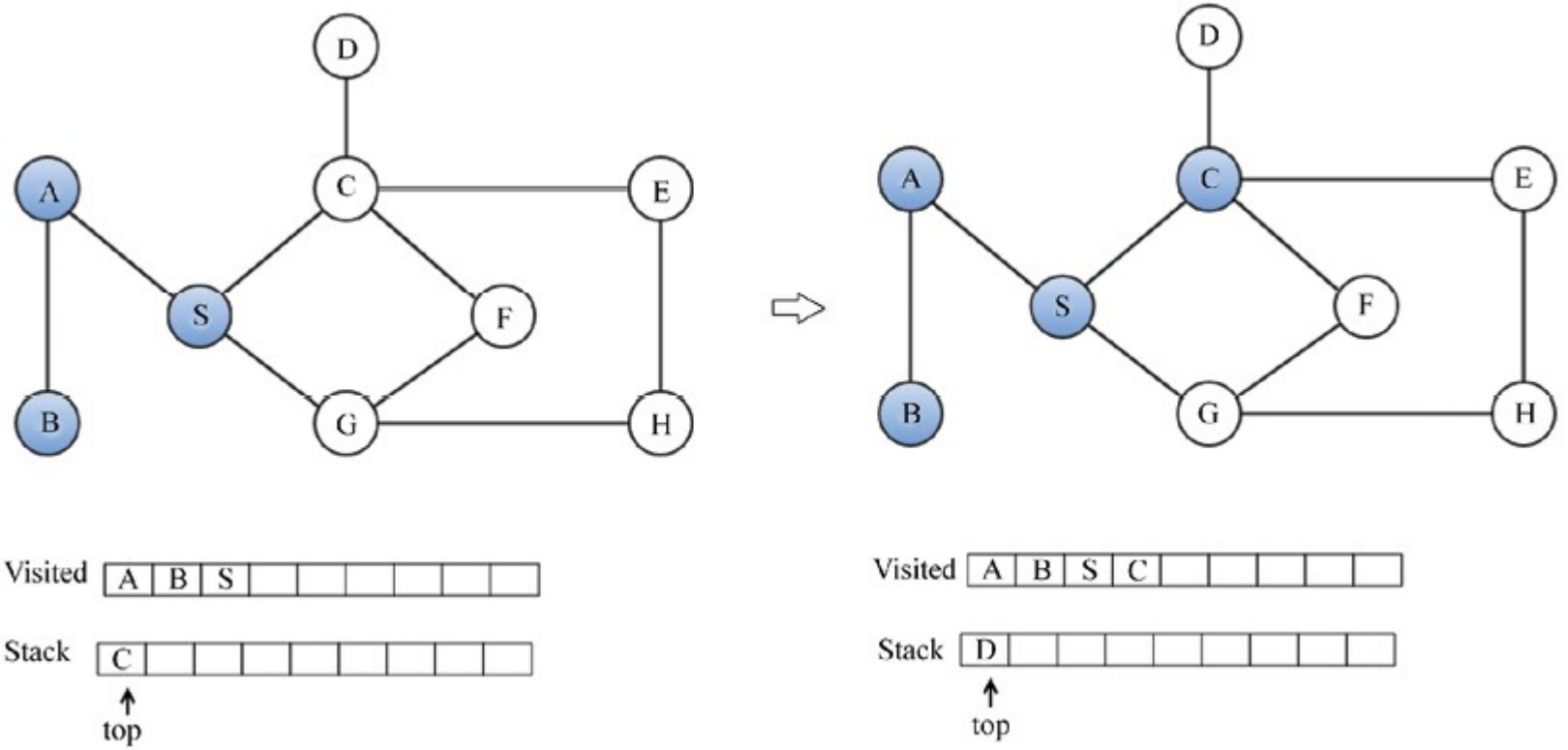
Fatec
Rio Claro

Após visitar o vértice B, olhamos para outro vizinho de A, isto é, S, visto que não há vértice conectado a B que possa ser visitado.

Em seguida, procuramos os vizinhos do vértice S, que são os vértices C e G. Visitamos C

Depth-first search (DFS)

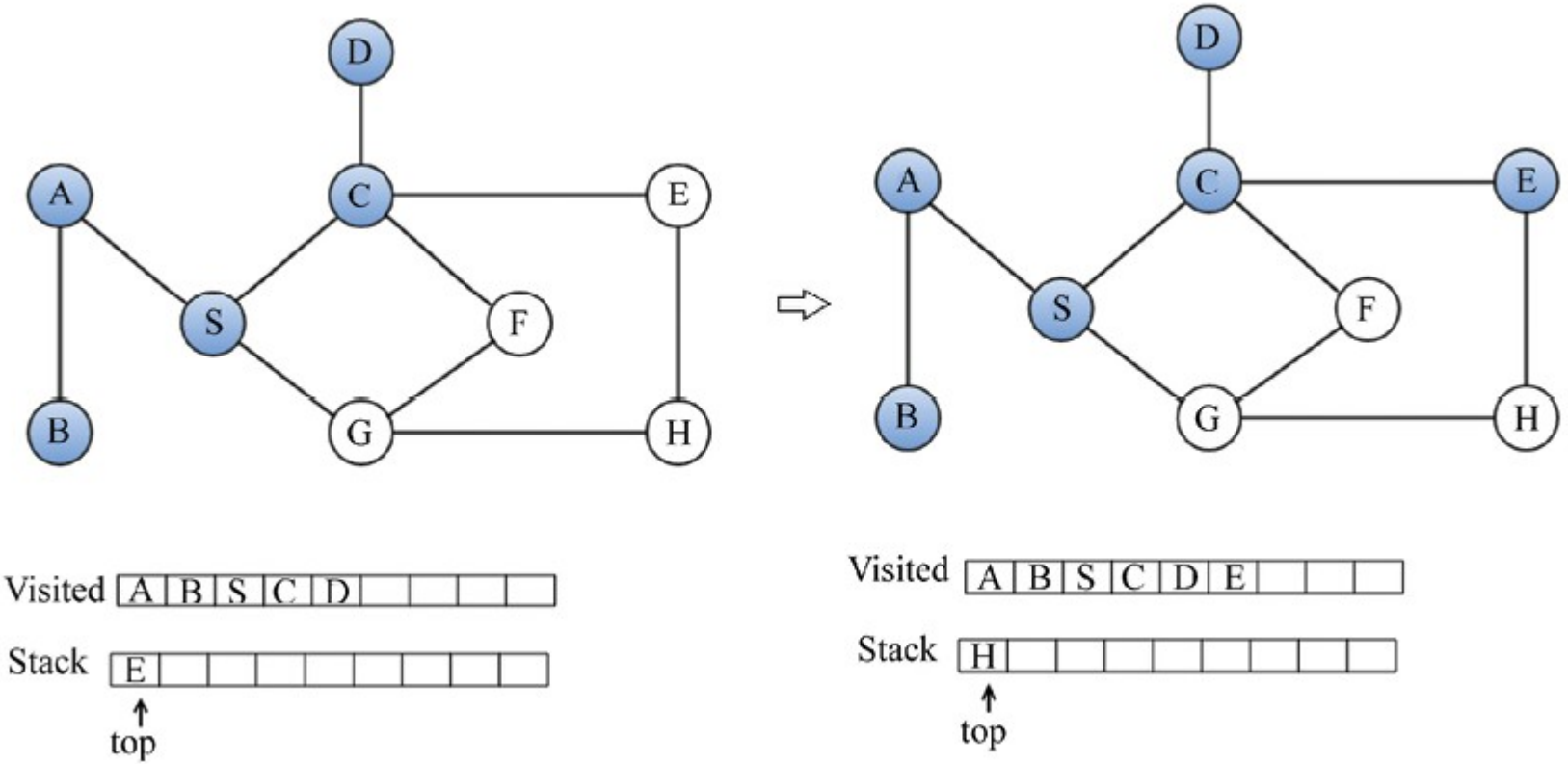
Fatec
Rio Claro



Depth-first search (DFS)

Fatec Rio Claro

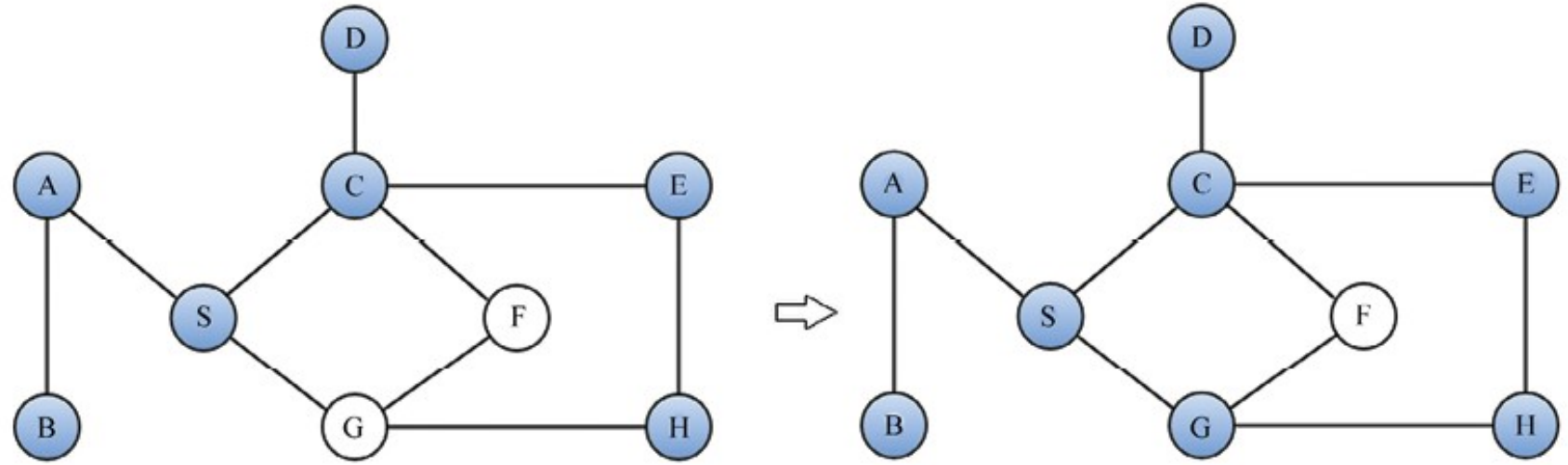
Após visitar o nó C, visitamos seus vértices vizinhos, D e E.

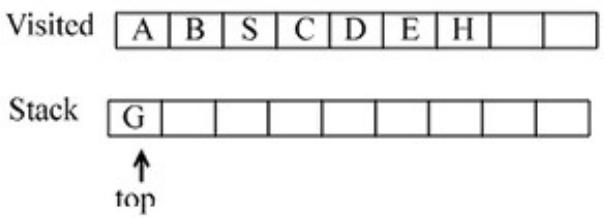


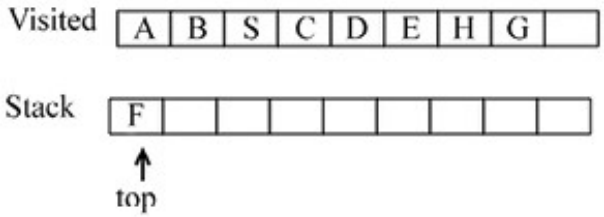
Depth-first search (DFS)

Fatec Rio Claro

Da mesma forma, depois de visitar o vértice E, visitamos os vértices H e G.

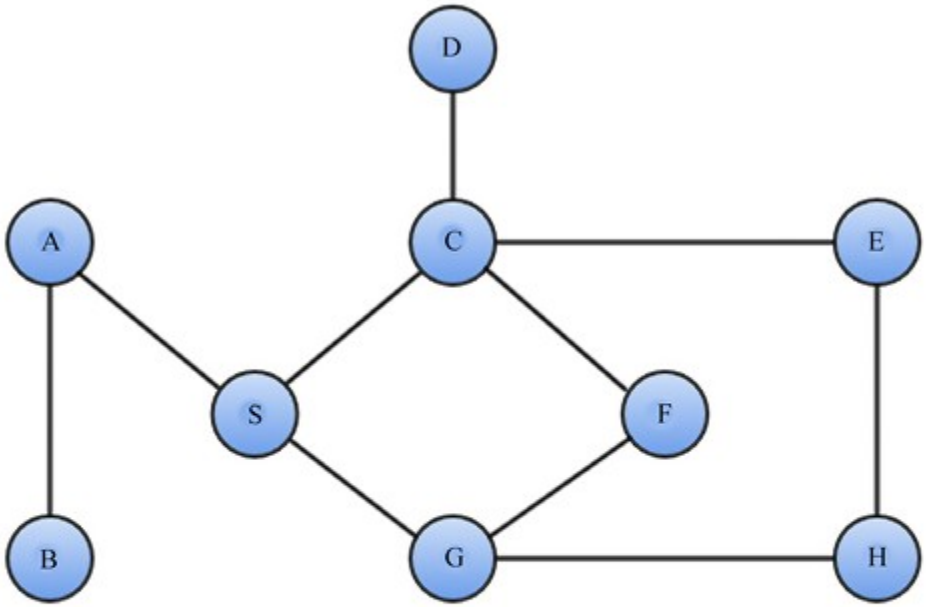






Depth-first search (DFS)

Fatec
Rio Claro



Visited	A	B	S	C	D	E	H	G	F

Stack

top

Outros métodos úteis para grafos

Minimum Spanning Tree

Fatec
Rio Claro

É muito comum precisarmos usar grafos para encontrar um caminho entre dois nós. Quando é necessário encontrar todos os caminhos entre os nós e, em algumas situações, podemos precisar encontrar o caminho mais curto entre os nós.

Para um grafo não ponderado, simplesmente determinaríamos o caminho com o menor número de arestas entre eles.

Se um grafo ponderado for fornecido, temos que calcular o peso total da passagem por um conjunto de arestas.

Assim, em uma situação diferente, podemos ter que encontrar o caminho mais longo ou mais curto usando algoritmos diferentes, como uma Árvore Geradora Mínima (*Minimum Spanning Tree*)

Minimum Spanning Tree

Fatec
Rio Claro

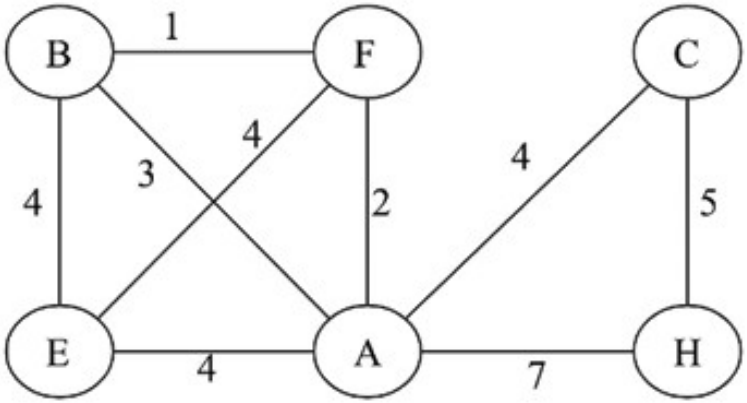
Uma Árvore Geradora Mínima (MST) é um subconjunto das arestas do grafo conectado com um grafo ponderado por arestas que conecta todos os nós do grafo, com os menores pesos totais de arestas possíveis e sem ciclo. Mais formalmente, dado um grafo conectado G, onde G = (V, E) com pesos de arestas de valor real, uma MST é um subgrafo com um subconjunto das arestas de forma que a soma dos pesos de arestas seja mínima e não haja ciclo.

Existem muitas árvores geradoras possíveis que podem conectar todos os nós do grafo sem qualquer ciclo, mas a árvore geradora de peso mínimo é uma árvore geradora que tem o menor peso total de arestas (também chamado de custo) entre todas as outras árvores geradoras possíveis.

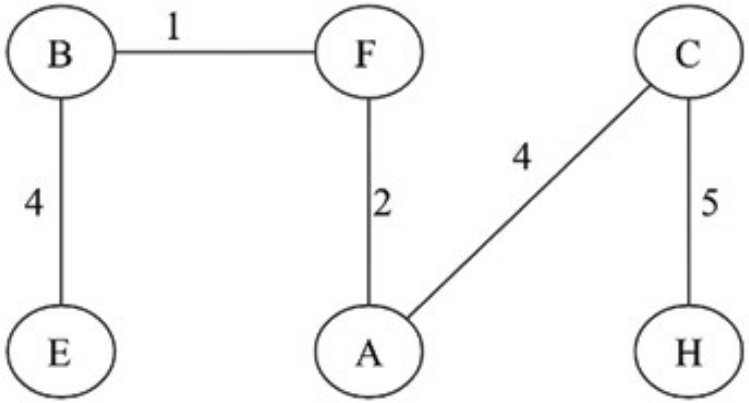
Minimum Spanning Tree

Fatec Rio Claro

O MST tem o menor peso total de todas as arestas, ou seja, (1+4+2+4+5 = 16) entre todas as outras árvores de extensão possíveis:



A graph



A minimum spanning tree

Minimum Spanning Tree

Fatec
Rio Claro

Uma Árvore Geradora Mínima (MST) é um subconjunto das arestas do grafo conectado com um grafo ponderado por arestas que conecta todos os nós do grafo, com os menores pesos totais de arestas possíveis e sem ciclo. Mais formalmente, dado um grafo conectado G, onde G = (V, E) com pesos de arestas de valor real, uma MST é um subgrafo com um subconjunto das arestas de forma que a soma dos pesos de arestas seja mínima e não haja ciclo.

Existem muitas árvores geradoras possíveis que podem conectar todos os nós do grafo sem qualquer ciclo, mas a árvore geradora de peso mínimo é uma árvore geradora que tem o menor peso total de arestas (também chamado de custo) entre todas as outras árvores geradoras possíveis.

Prática

Fatec
Rio Claro

O MST tem diversas aplicações no mundo real.

É usado principalmente em projetos de redes para congestionamento rodoviário, cabos hidráulicos, redes de cabos elétricos e até mesmo análise de cluster.

Apresente exemplos de outras aplicações reais

Algoritmo da MST de Kruskal

Fatec
Rio Claro

O algoritmo de Kruskal é amplamente utilizado para encontrar a árvore geradora a partir de um grafo ponderado, conectado e não direcionado.

Ele se baseia na abordagem gulosa, pois primeiro encontramos a aresta com o menor peso e a adicionamos à árvore.

Em seguida, em cada iteração, adicionamos a aresta com o menor peso à árvore geradora para que não formemos um ciclo.

Neste algoritmo, inicialmente, tratamos todos os vértices do grafo como uma árvore separada e, em cada iteração, selecionamos a aresta com o menor peso de forma que não forme um ciclo.

Essas árvores separadas são combinadas e crescem para formar uma árvore geradora. Repetimos esse processo até que todos os nós sejam processados.

Algoritmo da MST de Kruskal

Fatec
Rio Claro

O algoritmo funciona da seguinte forma:

1. Inicializar um MST (M) vazio com zero arestas
2. Ordenar todas as arestas de acordo com seus pesos
3. Para cada aresta da lista ordenada, adicionamos uma a uma ao MST (M) de forma que não forme um ciclo

Prática

Fatec
Rio Claro

Observe os códigos

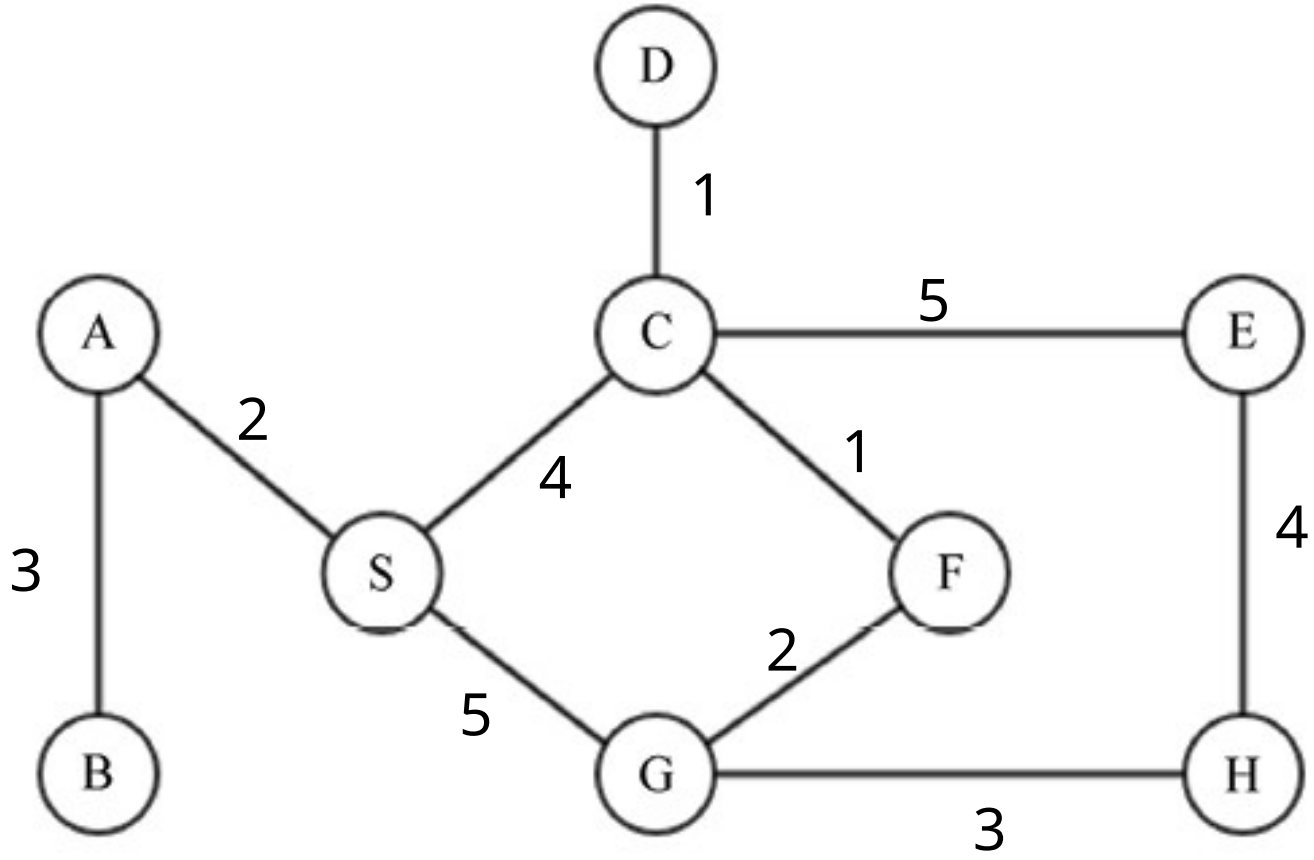
kruskal.py

prim.py

dijkstra.py

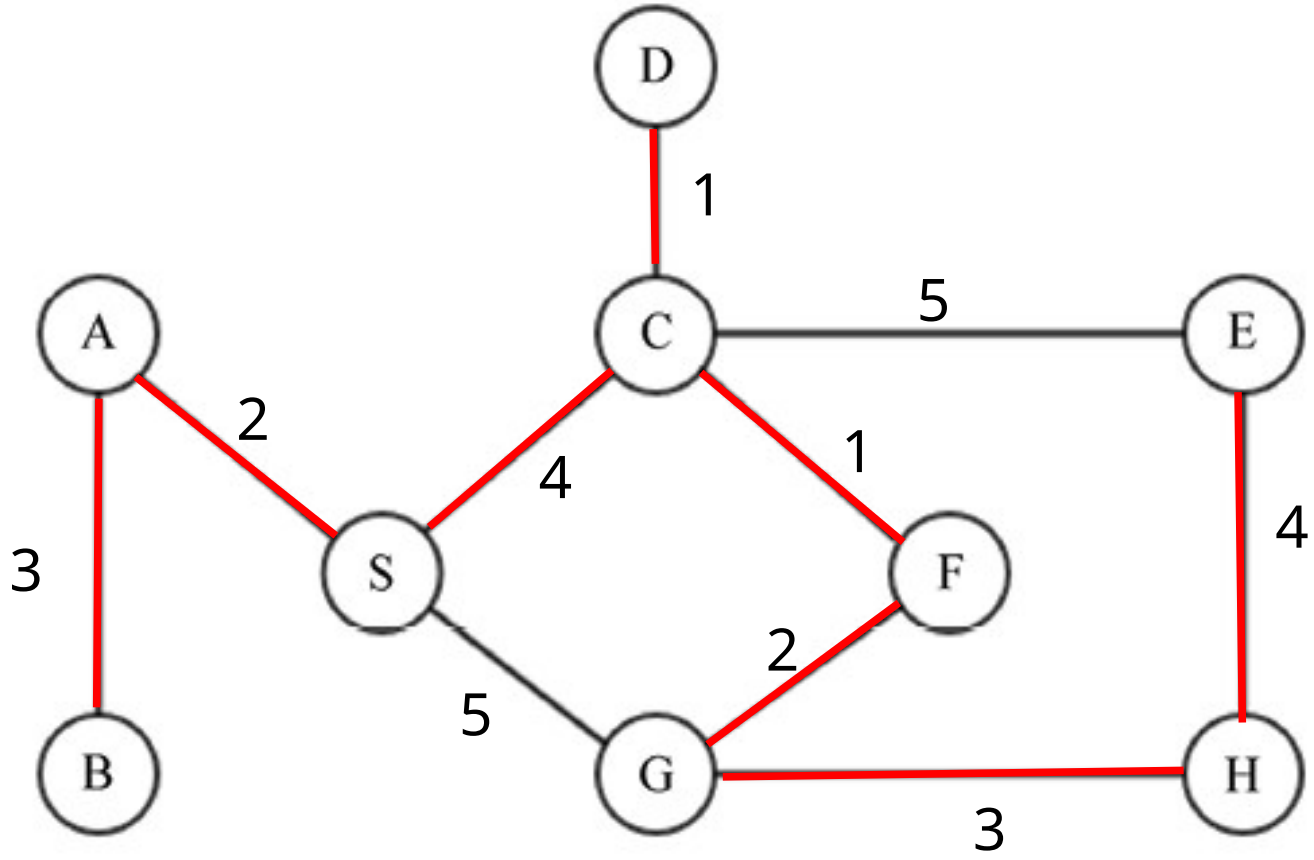
Prática

Fatec
Rio Claro



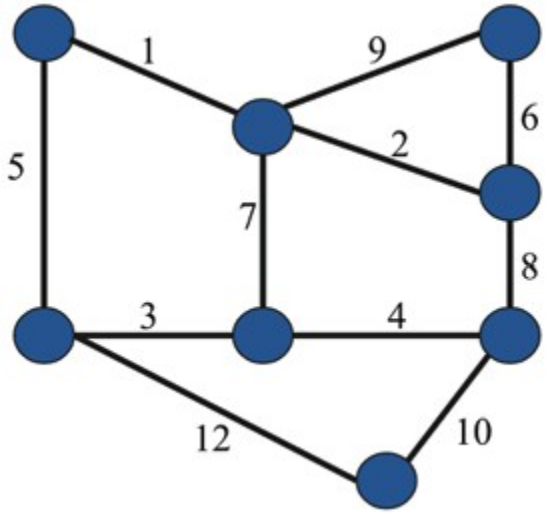
Prática

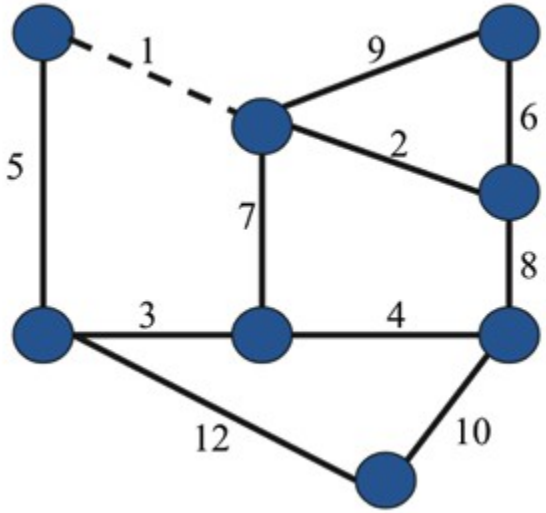
Fatec
Rio Claro



Algoritmo da MST de Kruskal

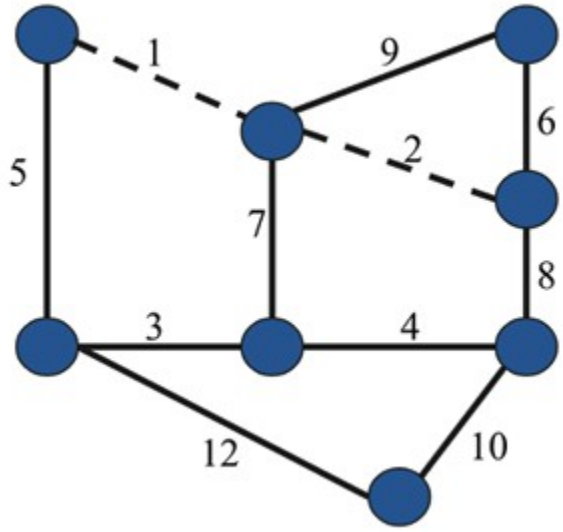
Fatec
Rio Claro

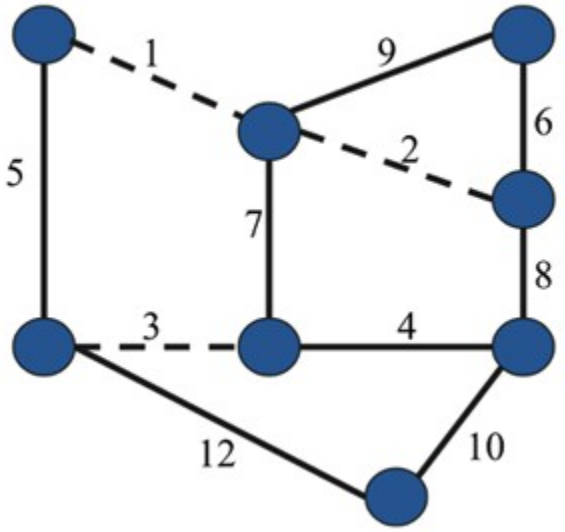




Algoritmo da MST de Kruskal

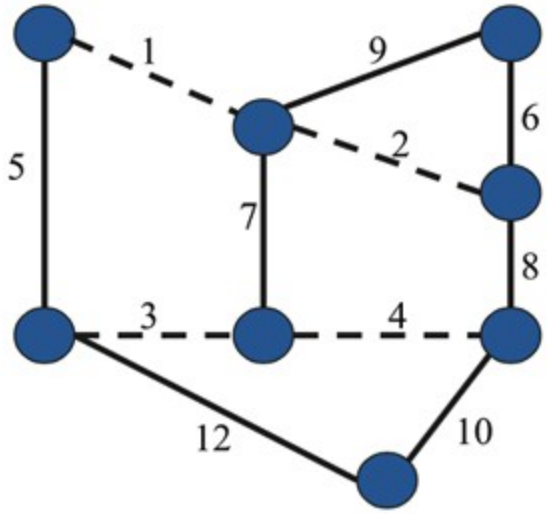
Fatec
Rio Claro

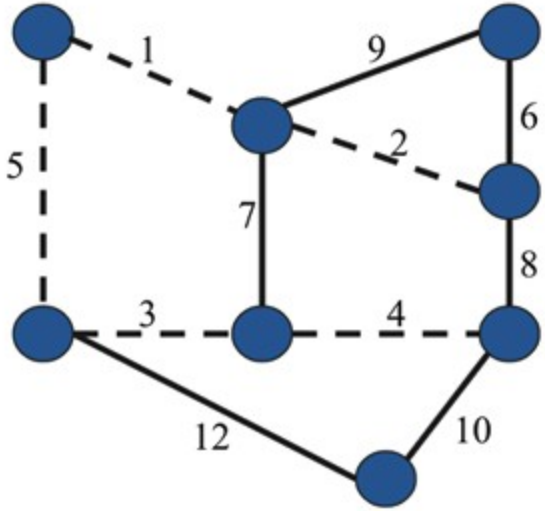




Algoritmo da MST de Kruskal

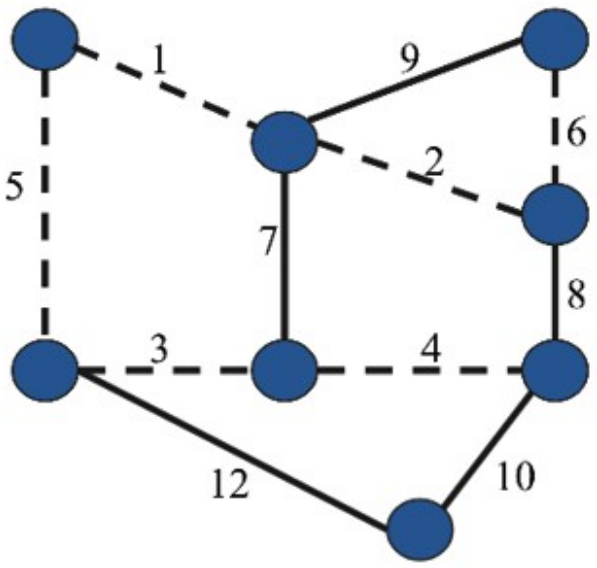
Fatec
Rio Claro

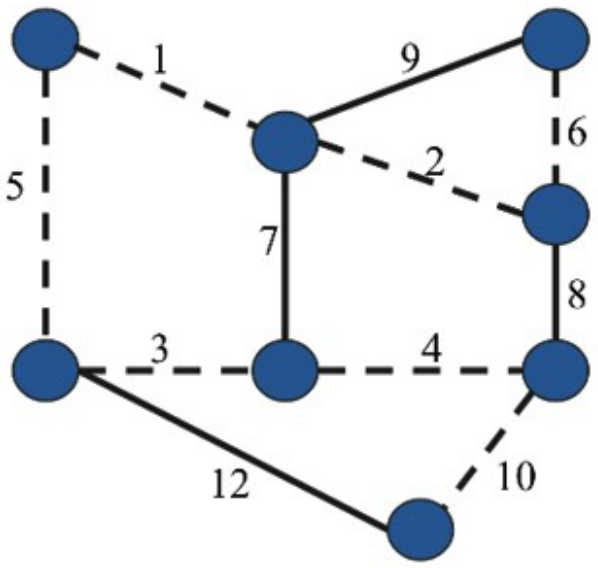




Algoritmo da MST de Kruskal

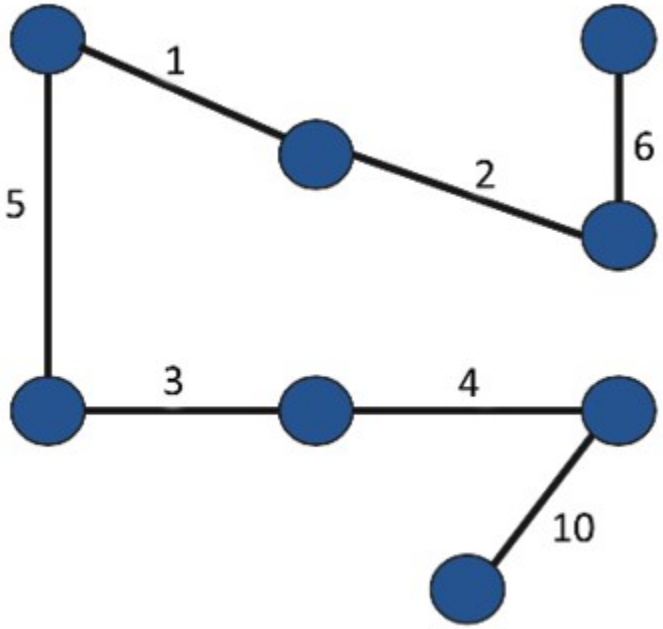
Fatec
Rio Claro





Algoritmo da MST de Kruskal

Fatec
Rio Claro



Algoritmo da Árvore Geradora Mínima de Prim

Fatec
Rio Claro

O algoritmo de Prim também se baseia em uma abordagem gulosa para encontrar a árvore geradora de custo mínimo.

O algoritmo de Prim é muito semelhante ao algoritmo de Dijkstra para encontrar o caminho mais curto em um grafo.

Neste algoritmo, partimos de um nó arbitrário como ponto de partida e, em seguida, verificamos as arestas de saída dos nós selecionados e percorremos a aresta que tem o menor custo (ou pesos).

Os termos custo e peso são usados indistintamente neste algoritmo. Assim, após começar a partir do nó selecionado, crescemos a árvore selecionando as arestas, uma a uma, que têm o menor peso e não formam um ciclo.

Algoritmo da Árvore Geradora Mínima de Prim

Fatec Rio Claro

O algoritmo funciona da seguinte forma:

1. Crie um dicionário que contenha todas as arestas e seus pesos.

2. Obtenha as arestas, uma a uma, que tenham o menor custo do dicionário e aumente a árvore de forma que o ciclo não seja formado.

3. Repita o passo 2 até que todos os vértices sejam visitados.

Qual Algoritmo escolher ?

Fatec
Rio Claro

Como os algoritmos MST de Kruskal e Prim são usados para o mesmo propósito, qual deles deve ser usado?

Em geral, depende da estrutura do grafo.

Para um grafo com C vértices e E arestas, a complexidade temporal do pior caso do algoritmo de Kruskal é O(E logV), e o algoritmo de Prim tem uma complexidade temporal de O(E + V logV).

Portanto, podemos observar que o algoritmo de Prim funciona melhor quando temos um grafo denso, enquanto o algoritmo de Kruskal é melhor quando temos um grafo esparso.

Qual Algoritmo escolher ?

Fatec
Rio Claro

Algoritmo	Objetivo	Tipo de Grafo	Resultado
Kruskal	Encontra a MST (ordena todas as arestas)	Não direcionado	MST mínima (pode começar em qualquer vértice)
Prim	Encontra a MST a partir de um vértice inicial	Não direcionado	MST mínima (expande por vizinhança)
Dijkstra	Encontra menor caminho de origem a todos os vértices	Direcionado ou não	Caminhos mínimos individuais

Síntese

Síntese

Fatec
Rio Claro

Um grafo é uma estrutura de dados não linear, muito importante devido ao grande número de aplicações no mundo real.

Discutimos diferentes maneiras de representar um grafo em Python, usando listas e dicionários.

Além disso, aprendemos dois algoritmos de travessia de grafos muito importantes: busca em profundidade (DFS) e busca em largura (BFS).

Também discutimos dois algoritmos muito importantes para encontrar uma MST: o algoritmo de Kruskal e o algoritmo de Prim.

Fatec
Rio Claro

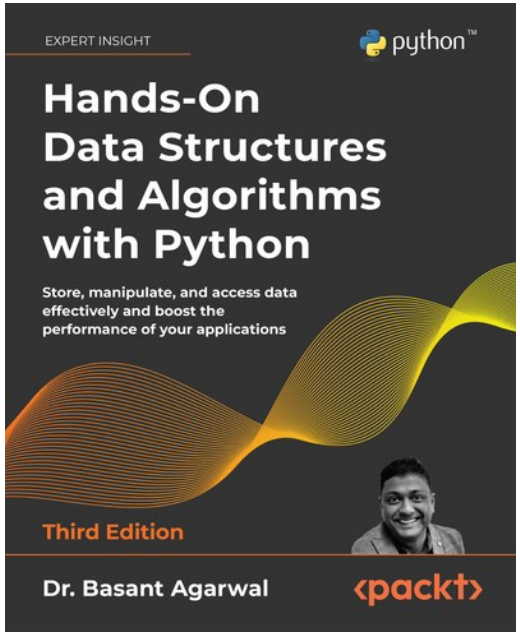
Dúvidas

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Prática

Bibliografia

AGARWAL, Basant. Hands-On Data Structures and Algorithms with Python. 3. ed. Birmingham: Packt Publishing, 2022.



Bibliografia

CANNING, John; BRODER, Alan; LAFORE, Robert. Data structures & algorithms in Python. Boston: Addison-Wesley Professional, 2019.