Google olabi

<ESTRUTURADEDADOS>
{COMPLEXAS}

{ Estrutura de dados complexas

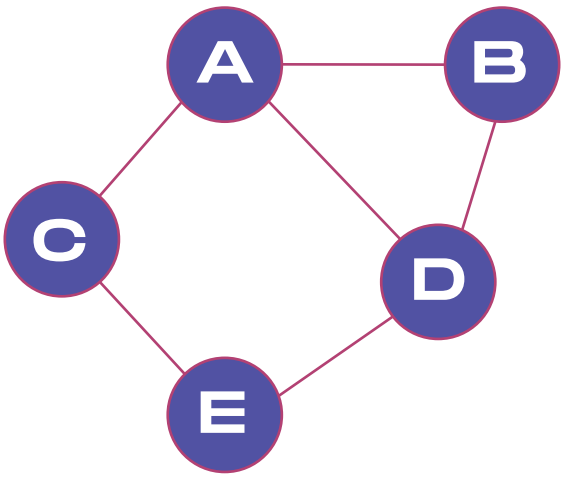
Conforme discutimos nas aulas anteriores, as estruturas de dados são usadas para armazenar e organizar os dados. Até este momento, abordamos Listas, Pilhas, filas (estruturas de dados lineares) e Árvores (estrutura de dados não lineares).

Nesta aula, continuaremos o estudo das estruturas de dados não lineares, mas agora com Grafos.

{ Grafos

Uma estrutura de dados Grafos, ou Graphs, representa uma rede que conecta vários pontos. Estes pontos são chamados de vértices e os ponteiros que conectam estes vértices são chamados de arestas. Assim, um grafo é definido como um conjunto de vértices e arestas que conectam estes vértices.

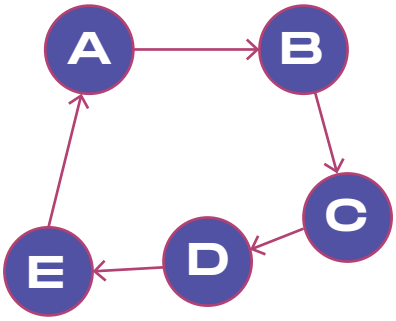
Os Grafos são usados, por exemplo, para representar as redes. As redes podem incluir caminhos em uma cidade ou rede telefônica ou rede de circuitos. Os Grafos também são usados em redes sociais como linkedIn, Facebook. Por exemplo, no Facebook, cada pessoa é representada com um vértice (ou nó). Cada nó é uma estrutura e contém informações como id de pessoa, nome, gênero e localização.



{ Diferentes Variantes de Grafo

1. Grafo Direcionado

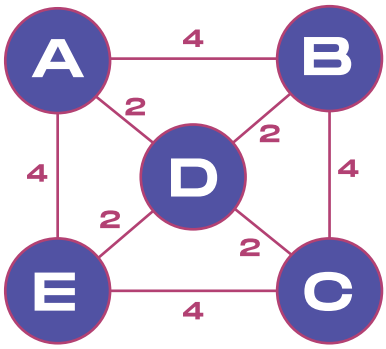
Um grafo dirigido é uma estrutura de dados na qual as arestas têm uma direção específica. Elas se originam de um vértice e culminam em outro vértice.



No diagrama, há uma aresta do vértice A ao vértice B. Note que A à B não é o mesmo que B à A como no grafo não direcionado a menos que haja uma aresta especificada de B à A.

2. Grafo Ponderado

Em um grafo ponderado, um peso é associado a cada aresta do grafo. O peso normalmente indica a distância entre os dois vértices. O diagrama a seguir mostra o grafo de peso. Como não são mostradas direções, este é o grafo não direcionado.



{ Como criar um grafo?

Normalmente, implementamos Grafos usando a coleção **HashMap**. Os elementos HashMap estão na forma de pares de valores chave. Podemos representar a lista de adjacências gráficas em um HashMap.

<Representação gráfica em Java>

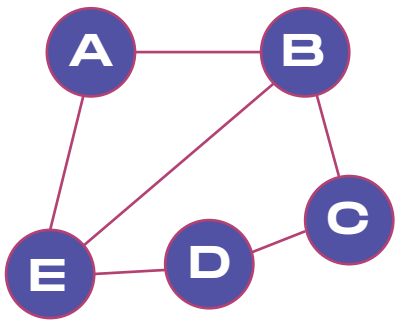
Representação gráfica significa a abordagem ou técnica através da qual os dados Grafos são armazenados na memória do computador.

Temos duas representações principais de Grafos: Matriz de Adjacência e Lista de Adjacência.

Matriz de Adjacência

A Matriz de Adjacência é uma representação linear de Grafos. Esta matriz armazena o mapeamento dos vértices e arestas do grafo. Na matriz adjacente, os vértices do grafo representam as linhas e colunas. Isto significa que se o Grafo tiver N vértices, então a matriz adjacente terá o tamanho NxN.

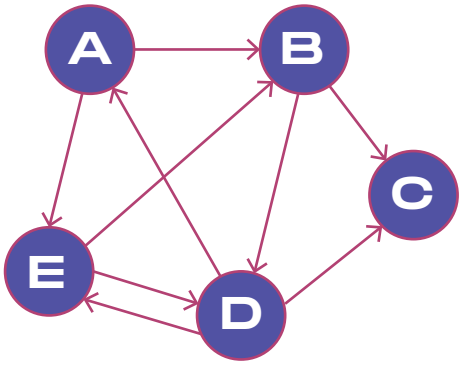
Se V for um conjunto de vértices do Grafo, a interseção Mij na lista de adjacências = 1 significa que existe uma aresta entre os vértices i e j.



	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

Como visto no diagrama acima, vemos que para o vértice A, as interseções AB e AE estão definidas para 1, pois há uma aresta de A à B e A à E. Da mesma forma, a interseção BA está definida para 1, pois este é um Grafo não direcionado e AB = BA.

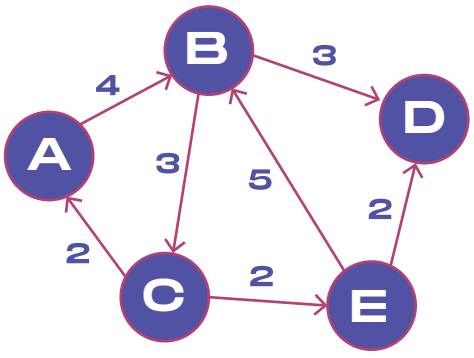
Caso o Grafo seja direcionado, a interseção Mij será definida para 1 somente se houver uma aresta clara direcionada de Vi para Vj.



	A	B	C	D	E
A	0	1	0	0	1
B	0	0	1	1	0
C	0	0	0	0	0
D	1	0	1	0	1
E	0	1	0	1	0

Como podemos ver no diagrama acima, há uma aresta de A à B. Assim, a interseção AB está definida em 1, mas a interseção BA está definida em 0. Isto porque não há nenhuma aresta direcionada de B à A.

Para os Grafos ponderados, sabemos que um número inteiro também conhecido como peso está associado a cada aresta. Representamos este peso na Matriz de adjacência para a aresta que existe. Este peso é especificado sempre que há uma aresta de um vértice a outro, em vez de '1'.



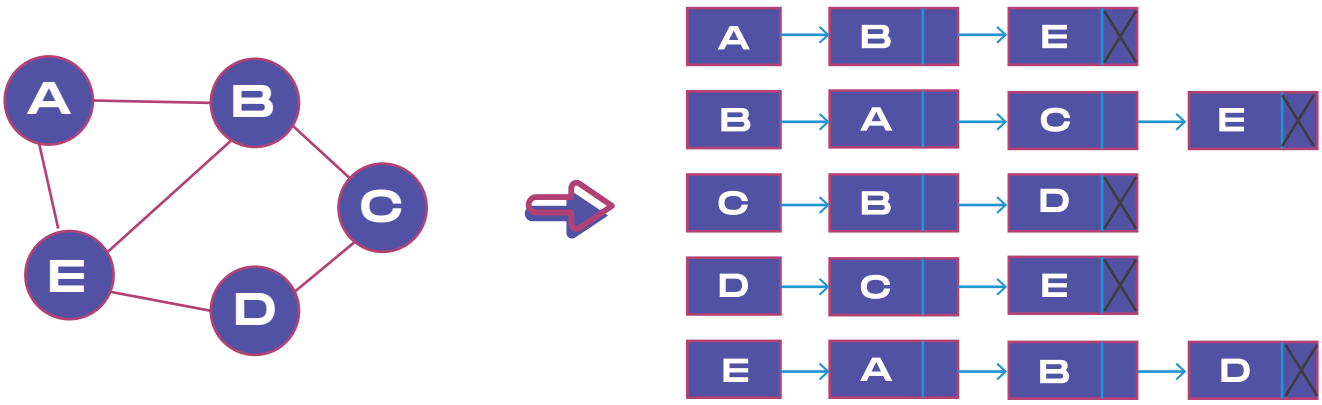
	A	B	C	D	E
A	0	4	0	0	0
B	0	0	3	3	0
C	2	0	0	0	2
D	0	0	0	0	0
E	0	5	0	2	0

Lista de Adjacência

Em vez de representar um Grafo como uma matriz adjacente que é sequencial por natureza, também podemos usar a representação vinculada. Esta representação vinculada é conhecida como a lista de adjacências. Uma lista de adjacências nada mais é do que uma lista vinculada e cada nó da lista representa um vértice.

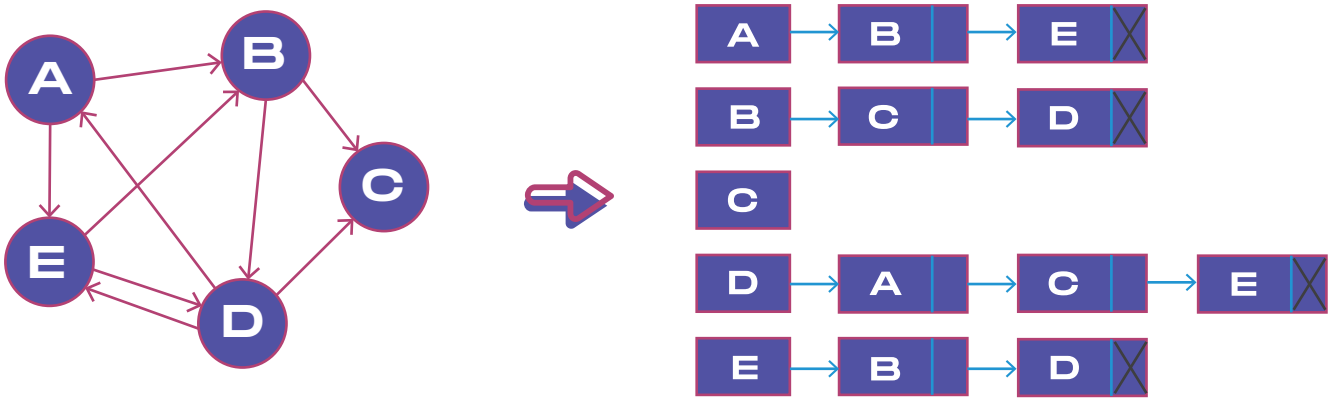
A presença de uma aresta entre dois vértices é denotada por um ponteiro do primeiro vértice para o segundo. Esta lista de adjacências é mantida para cada vértice no Grafo.

Quando percorremos todos os nós adjacentes para um determinado nó, armazenamos NULL no campo do ponteiro seguinte do último nó da lista de adjacências.



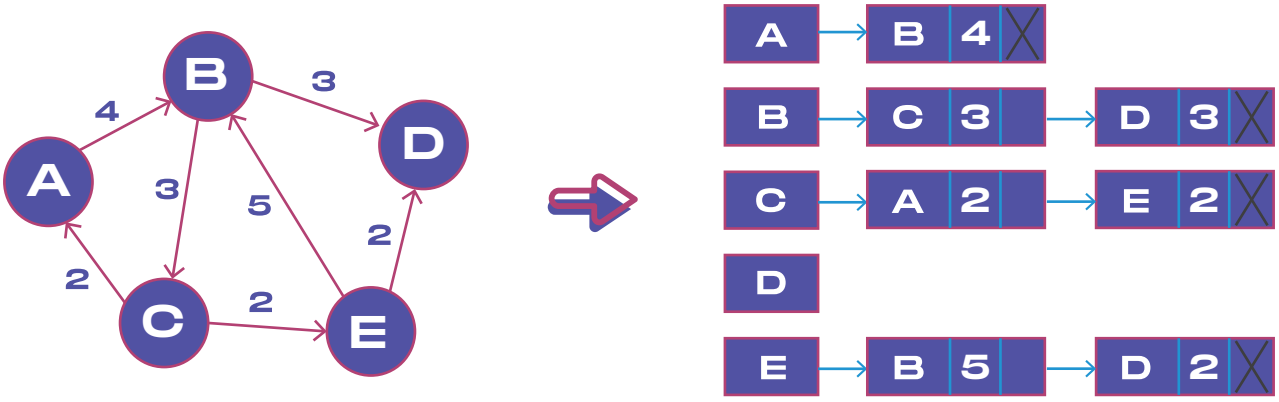
A figura acima mostra a lista de adjacências para o Grafo não direcionado. Vemos que cada vértice ou nó tem sua lista de adjacências.

No caso do Grafo não direcionado, os comprimentos totais das listas de adjacências são normalmente o dobro do número de arestas. No Grafo acima, o número total de arestas é 6 e o total ou soma do comprimento de toda a lista de adjacências é 12.



Como visto na figura acima, no Grafo dirigido, o comprimento total das listas adjacentes do Grafo é igual ao número de arestas no Grafo. No Grafo acima, há 9 arestas e soma dos comprimentos das listas de adjacências para este Grafo = 9.

Agora vamos considerar o seguinte Grafo direcionado ponderado. Note que cada aresta do Grafo ponderado tem um peso associado a ele. Portanto, quando representamos este Grafo com a lista de adjacências, temos que adicionar um novo campo a cada nó da lista que denotará o peso da aresta.



O diagrama acima mostra o Grafo ponderado e sua lista de adjacências. Note que há um novo espaço na lista de adjacências que denota o peso de cada nó.

{ Percorrendo Grafos

Para realizar qualquer ação significativa como a busca pela presença de qualquer dado, precisamos percorrer o Grafo de tal forma que cada vértice e a arestas do Grafo seja visitada pelo menos uma vez. Da mesma forma que Árvores, há dois algoritmos suportados para percorre o Grafo: Depth-first e Breadth-first.

Referências:

https://www.softwaretestinghelp.com/java-graph-tutorial/

https://www.geeksforgeeks.org/graph-and-its-representations/

Google olabi

<GRAFOSEMJAVA>

/exercícios/

1 Implemente um simulador de redes sociais. Este simulador deve ser compilado em um módulo de nome “redesocial”.

Problema

Uma rede social de amizades pode ser representada por um grafo G (V,E) em que V é o conjunto de nós e E o conjunto de arestas. Cada um dos nós n_0, n_1, n_2... representa uma pessoa e, caso duas pessoas n_i e n_j sejam amigas, existe uma aresta (i,j) ∈ E. Umas das maneiras usuais para se representar um grafo é através de uma matriz de adjacência $n \times n$ de n colunas e n linhas. Cada linha (ou coluna) i contém as relações da pessoa n_i. Considere a matriz de adjacência abaixo:

ID	n_0	n_1	n_2	n_3	n_4
n_0	0	1	1	0	1
n_1	1	0	0	1	0
n_2	1	0	0	0	0
n_3	0	1	0	0	1
n_4	1	0	0	1	0

Esta matriz representa uma rede social entre 5 pessoas: n_0, n_1, n_2, n_3 e n_4. Além disso, quando a posição (i,j) da matriz é 1, então as pessoas n_i e n_j são amigas entre si. Caso a posição (i,j) da matriz é 0, então n_i e n_j não são amigas. Observe que a pessoa n_0 é amiga das pessoas n_1, n_2 e n_4, mas não é amiga da pessoa n_3.

Importante: a relação de amizade é simétrica: se n_i é amigo de n_j, então n_j é, necessariamente, amigo de n_i. Além disso, em redes sociais de amizade, não existe aresta entre a mesma pessoa, ou seja, não existem arestas do tipo (i,i).

1.1 Implementar um procedimento para inicializar a matriz de adjacência que gerencia a rede social. Inicialmente, ninguém é amigo de ninguém, ou seja, todas as posições da matriz são zeradas.

1.2 Implementar um procedimento para marcar duas pessoas como amigas na matriz de adjacência.

1.3 Implementar uma função para retornar o número de amigos em comum que duas pessoas têm. Essa função deve também imprimir os identificadores dos amigos em comum.