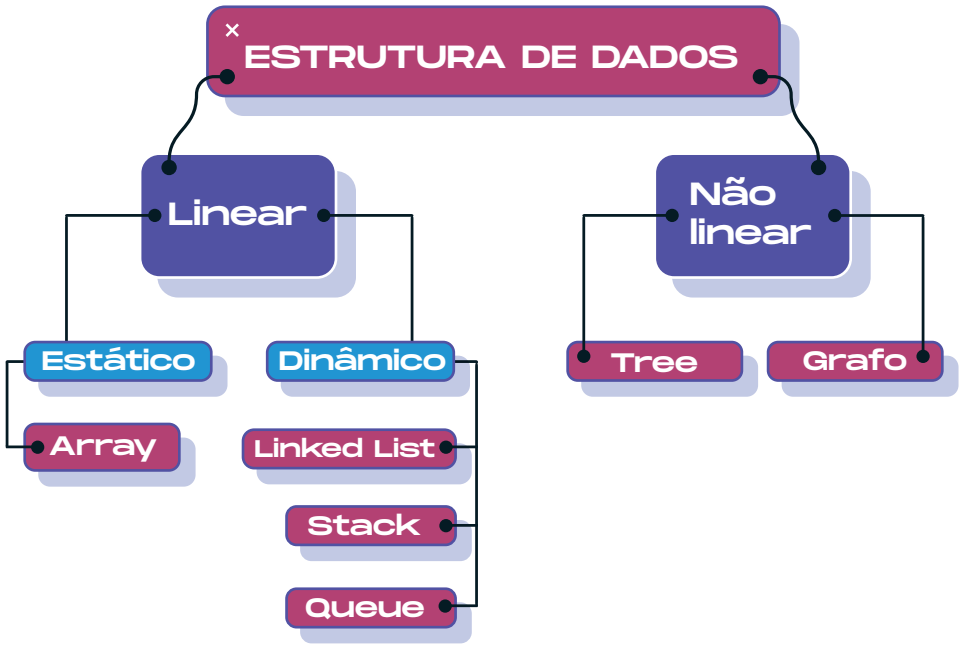
Google olabi

<ESTRUTURADEDADOS>
{GRAPHS}

{ Estrutura de dados

Conforme discutido nas aulas anteriores, estruturas de dados são usadas para armazenar e organizar dados. Também abordamos *arrays*, LinkedLists, *stack, queues* (estrutura de dados lineares) e *trees* (não lineares).

Nesta aula, continuaremos estudando estruturas de dados não lineares, agora com *graphs*, ou grafos.

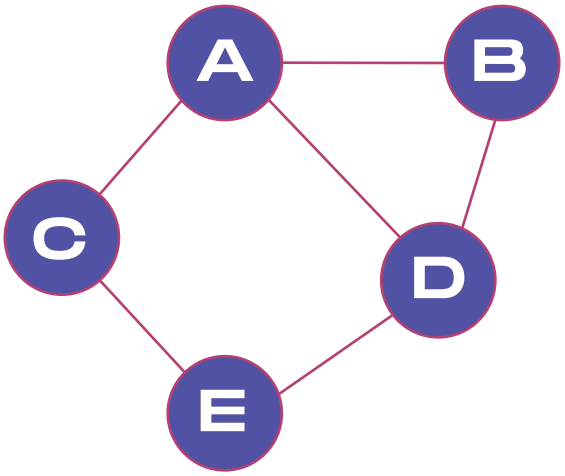


{ Grafos

Uma estrutura de dados *graphs*, ou grafos, representa uma rede que conecta vários pontos, chamados de vértices. Os links que conectam estes vértices são chamados de *edges*, ou bordas. Um grafo é definido como um conjunto de vértices e bordas que os conectam.

Grafos são usados, por exemplo, para representar redes, que, por sua vez, podem incluir caminhos em uma cidade, rede telefônica, ou circuitos. Grafos também são usados em redes sociais como LinkedIn e Facebook - onde cada pessoa é representada por um vértice ou nó, e cada nó, uma estrutura, com informações como identidade, nome, gênero e localização do usuário.

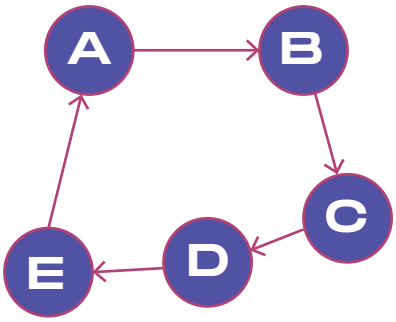
Considere, por exemplo, o grafo de cinco vértices {A, B, C, D, E} e bordas abaixo {{AB}, {AC}, {AD}, {BD}, {CE}, {ED}}}. Como as bordas não mostram nenhuma direção, trata-se de um grafo não direcionado.



{ Variantes de Grafo

1. Direcionado

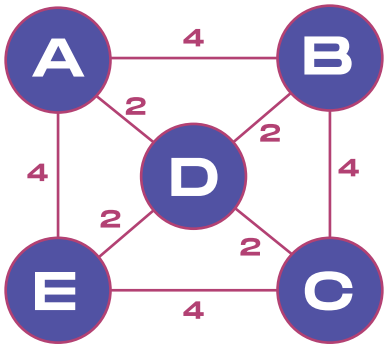
Um grafo dirigido, ou dígrafo, representa uma estrutura de dados na qual as bordas têm direção específica e originam de um vértice culminando em outro.



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

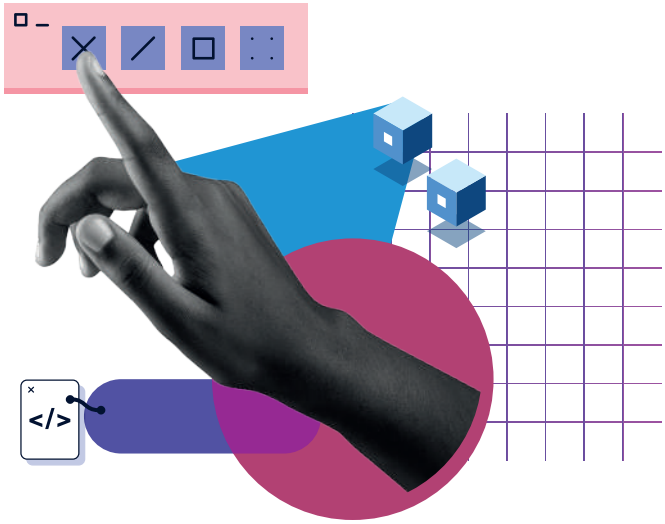
2. Ponderado

Em um grafo ponderado, um peso é associado a cada borda e usualmente indica a distância entre os dois vértices. O diagrama a seguir mostra um grafo de peso; como não são exibidas direções, ele não é direcionado.



{ Como criar um grafo?

Usualmente, implementamos grafos usando a coleção **HashMap**. Os elementos HashMap são representados por pares de valores chave. Podemos, por exemplo, representar uma lista de adjacências gráficas em um HashMap.



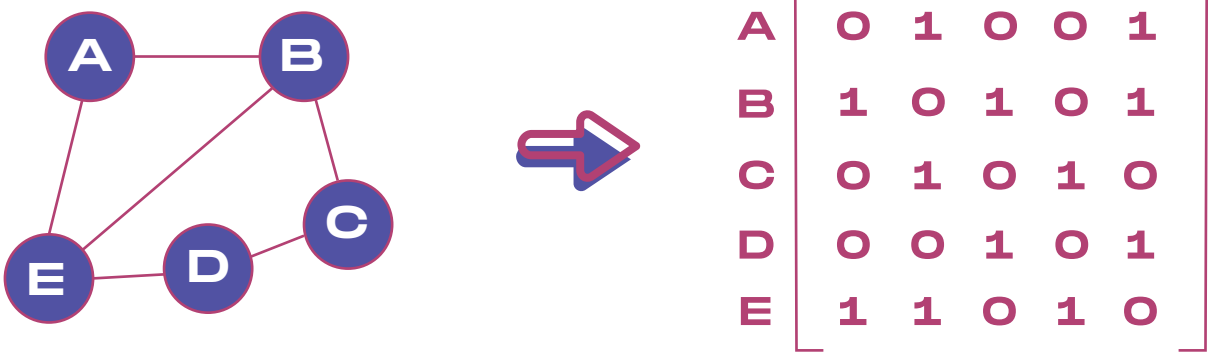
{ Representação gráfica em Java

Uma representação gráfica trata da abordagem ou técnica pela qual dados grafos são armazenados na memória do computador.

Existem duas representações principais de grafos:

1. Matriz de adjacência

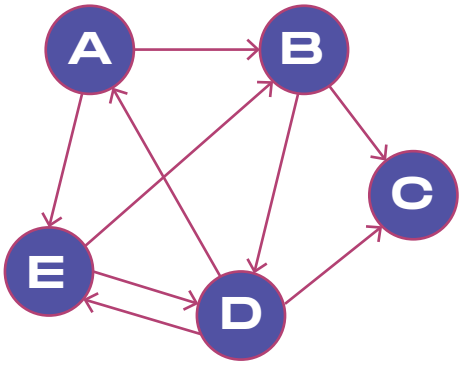
A matriz de adjacência é uma representação linear de grafos que armazena o mapeamento dos vértices e bordas deles. Na matriz adjacente, os vértices do grafo em questão representam linhas e colunas; se o grafo tiver N vértices, a matriz adjacente terá o tamanho N x N e, se V representar um conjunto de vértices do grafo, a interseção Mij na lista de adjacências = 1 confirma a existência de uma aresta entre os vértices i e j.



Como visto no diagrama acima, no vértice A, as interseções AB e AE estão definidas para 1, uma vez que há uma borda de A até B e de A até E.

Da mesma forma, a interseção BA está definida para 1, já que este é um grafo não direcionado e AB = BA. Definiremos todas as outras interseções para as quais haja uma borda em 1.

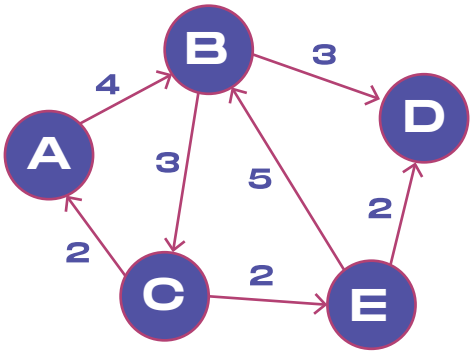
Caso o grafo seja direcionado, a interseção Mij será definida para 1 somente se houver uma borda clara direcionada de Vi para Vj.



	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

No diagrama acima, há uma borda de A até B. A intersecção AB está definida em 1, mas a interseção BA, em 0, uma vez que não há nenhuma borda direcionada de B até A. Nos vértices E e D, podemos observar bordas de E a D e de D a E. Por esse motivo, fixamos estas duas interseções em 1 na matriz adjacente.

Para o grafo ponderado, um número inteiro - também conhecido como peso - é associado a cada aresta. Representamos este peso na matriz de adjacência para a aresta que já existe, e o especificamos sempre que houver uma borda de um vértice a outro, ao invés de “1”.

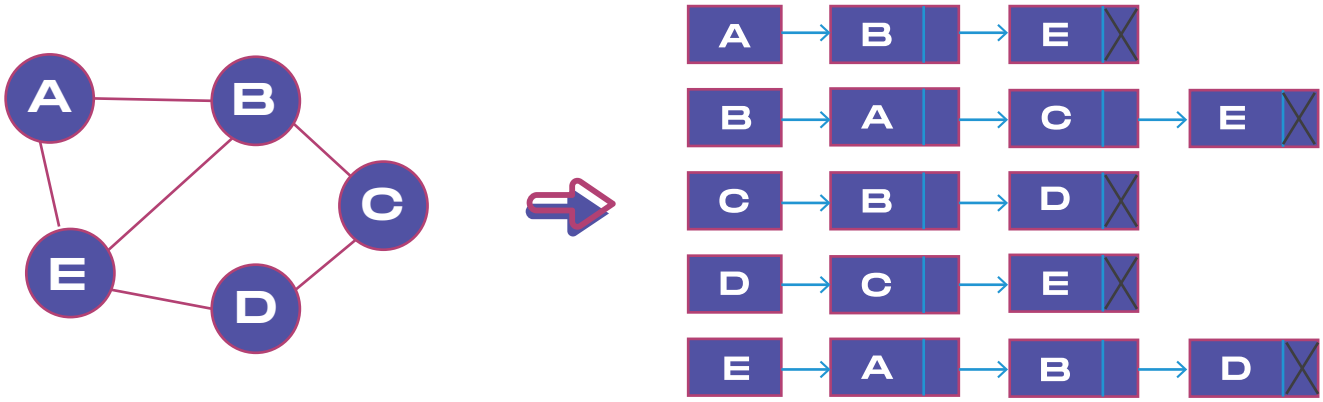


	A	B	C	D	E
A	0	4	0	0	0
B	0	0	3	3	0
C	2	0	0	0	2
D	0	0	0	0	0
E	0	5	0	2	0

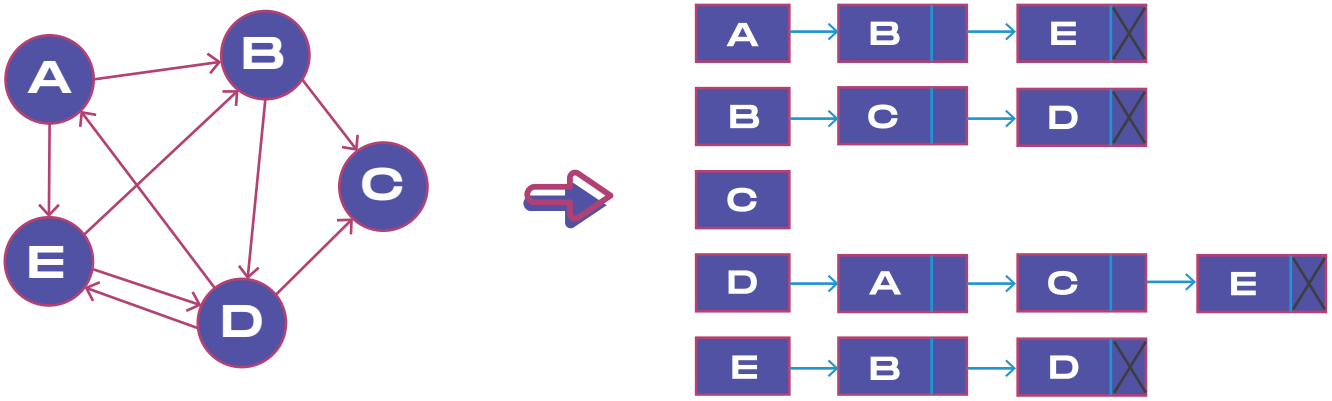
2. Lista de adjacência

Nesta representação, ao invés de representar um grafo como matriz adjacente (sequencial por natureza), podemos fazê-lo de forma vinculada, com cada nó representando um vértice.

A presença de uma borda entre dois vértices é identificada por um ponteiro do primeiro vértice para o segundo. Esta lista de adjacências é mantida para cada vértice do grafo. Quando percorremos todos os nós adjacentes em direção a outro nó, armazenamos *NULL* no campo do ponteiro seguinte do último nó da lista de adjacências.

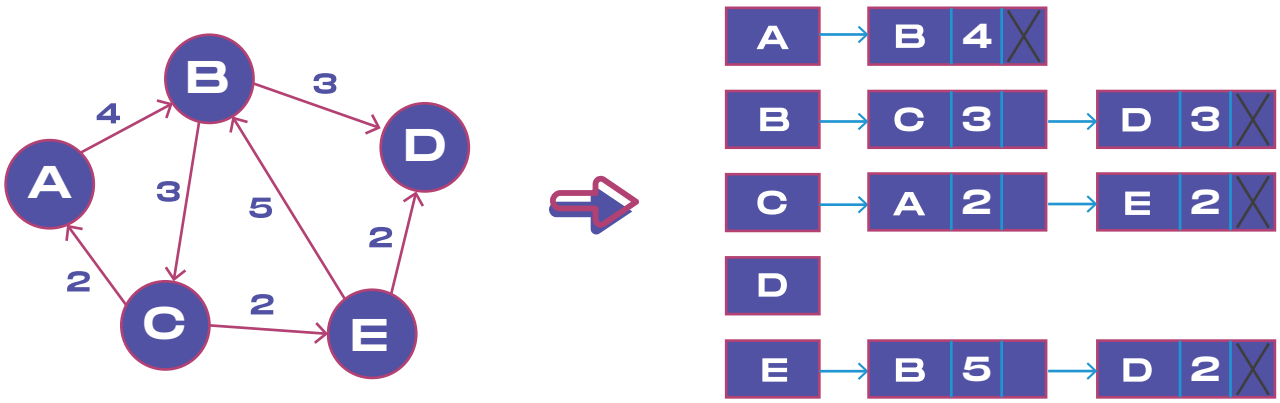


A figura acima mostra a lista de adjacências para grafo não direcionado. Nela, observamos que cada vértice ou nó tem sua própria lista de adjacências. No caso do grafo não direcionado, os comprimentos totais das listas de adjacências usualmente totalizam o dobro do número de bordas. No grafo acima, o número total de bordas é 6, e o total ou soma do comprimento de toda a lista de adjacências, 12.



No grafo dirigido representado na figura acima, o comprimento total das listas adjacentes é igual ao número de bordas - 9 ao todo. A soma dos comprimentos das listas de adjacências para este grafo = 9.

Agora, considere um grafo direcionado ponderado onde cada aresta tem um peso associado. Ao representar este grafo com a lista de adjacências, precisamos adicionar um novo campo a cada nó da lista que denota o peso da borda.



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

{ Percorrendo Grafos

Para realizar qualquer ação significativa, como busca pela presença de qualquer dado, precisamos percorrer o grafo de tal forma que cada vértice e borda sejam visitados pelo menos uma vez.

Há dois algoritmos suportados para percorrer o grafo:

{ Depth-First

A busca em profundidade (*Depth-First Search* em inglês, ou DFS) é uma técnica usada para percorrer uma árvore ou grafo que começa com um nó raiz para, em seguida, percorrer os nós adjacentes do nó raiz de forma a se aprofundar no grafo em si.

Na técnica, os nós são percorridos em profundidade até que não haja mais "crianças" a explorar. Uma vez alcançado o nó foliar, a DFS retrocede, começa outros nós e realiza o percurso de maneira semelhante. A DFS usa uma estrutura de dados em pilha, para armazenar nós sendo percorridos.

<Aplicações>

1. Detecção de um ciclo

O DFS facilita a detecção de um ciclo em um grafo quando podemos retroceder até uma borda. Como já vimos no gráfico da DFS, em qualquer dupla de vértices, podemos encontrar o caminho entre os dois.

2. Árvore de espaçamento mínimo e caminho mais curto

Se executarmos a técnica DFS em um grafo não-pesado, ela nos trará a árvore de espaçamento mínimo e caminho mais curto.

3. Triagem topológica

Utilizada quando precisamos programar os trabalhos e possuímos dependências entre vários deles.

{ Breadth-first

A técnica de busca em largura (*Breadth-First Search*, ou BFS) usa uma fila para armazenar os nós do grafo. Na BFS, nós percorremos a largura (ou nível) do grafo de forma sensata. Após explorar todos os vértices ou nós de um nível, podemos passar para o nível seguinte.

<Aplicações>

1. Coleta de lixo

Um dos algoritmos usados pela técnica de coleta de lixo para copiá-la é chamado de "algoritmo de Cheney". Este algoritmo utiliza uma técnica de travessia ampla e simples.

2. Radiodifusão em redes

A transmissão de pacotes de um ponto a outro em uma rede é feita usando a técnica BFS.

3. Navegação por GPS

Podemos usar a técnica para encontrar nós adjacentes enquanto navegamos com GPS.

4. Sites de redes sociais

A técnica também é usada em redes sociais, para encontrar a rede de pessoas que cerca determinado usuário.

5. Caminho mais curto e árvore de alcance mínimo em grafo não-pesado

No grafo não-ponderado, BFS pode ser usada para encontrar uma árvore de alcance mínimo e o caminho mais curto entre nós.

Referências

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

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

Google olabi

<ESTRUTURADEDADOS>
{GRAPHS}

/exercícios/

{ Exercício coletivo I

Dada a lista adjacente de um gráfico bidirecional, sua tarefa é copiar/ clonar a lista de adjacências para cada vértice e retornar uma nova lista.

{ Exercício coletivo II:

Escreva um algoritmo para o caminho mais curto de 1 a n considerando um grafo dirigido cujos vértices são numerados de 1 a n. Considere também uma borda de um vértice i para um vértice j iif ou j = i + 1 ou j = 3 * i. Sua tarefa é encontrar o número mínimo de bordas em um caminho em G do vértice 1 ao vértice n.

{ Exercício coletivo III:

Considere um gráfico dirigido e projete um algoritmo para descobrir se existe uma rota entre dois nós.

<Leitura recomendada>

MLA. McDowell, Gayle Laakmann, 1982-. Cracking the Coding Interview : 150 Programming Questions and Solutions. Palo Alto, CA :CareerCup, LLC, 2011.

<Referência para os exercícios>

https://practice.geeksforgeeks.org/problems/

Professora: Renata Pereira | Monitora: Jessica Osko