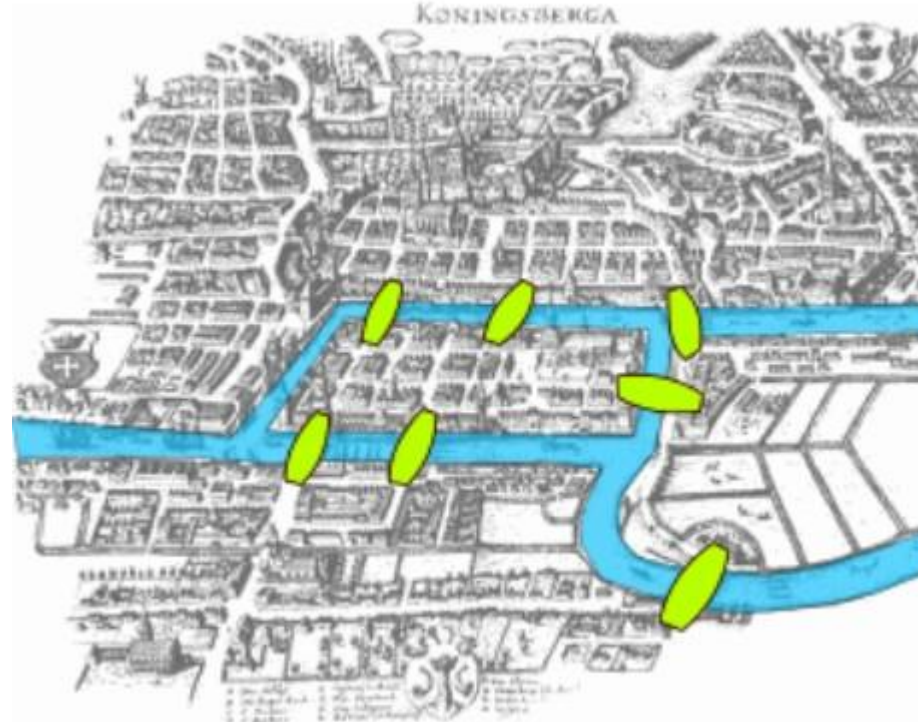


Conceitos Básicos e Representação

RODRIGO LYRA

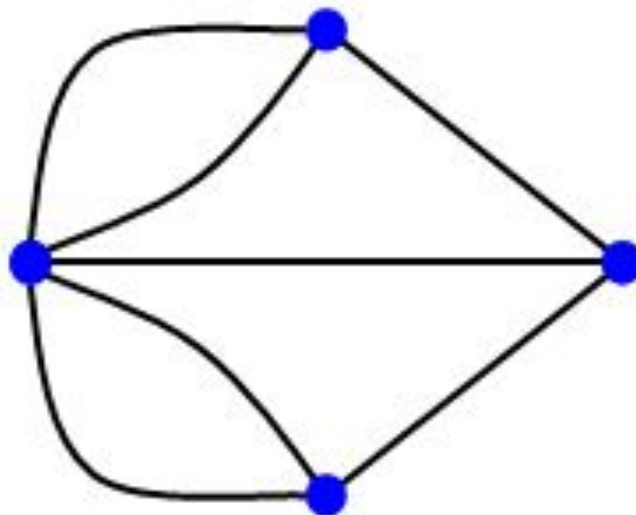
As pontes de Königsberg



As pontes de Königsberg

- A cidade de Königsberg (Pertencia a Prússia, atualmente território russo) tinha um dilema em relação as 7 pontes presentes na cidade.
 - **“Seria possível atravessar as 7 pontes, sem cruzar uma delas mais de uma vez?”**
- O problema foi solucionado por Leonhard Euler em 1736 utilizando uma representação mais simples do problem, eliminando qualquer informação dispensável.

As pontes de Königsberg



As pontes de Königsberg

- Esta representação deu origem à teoria dos grafos. E sua simplicidade permitiu um estudo mais preciso de diversos problemas, juntamente com a criação de teoremas.
- O problema acima por exemplo pode ser definido pela seguinte regra:
Um grafo G conexo possui caminho euleriano se e somente se ele tem exatamente zero ou dois vértices de grau ímpar.

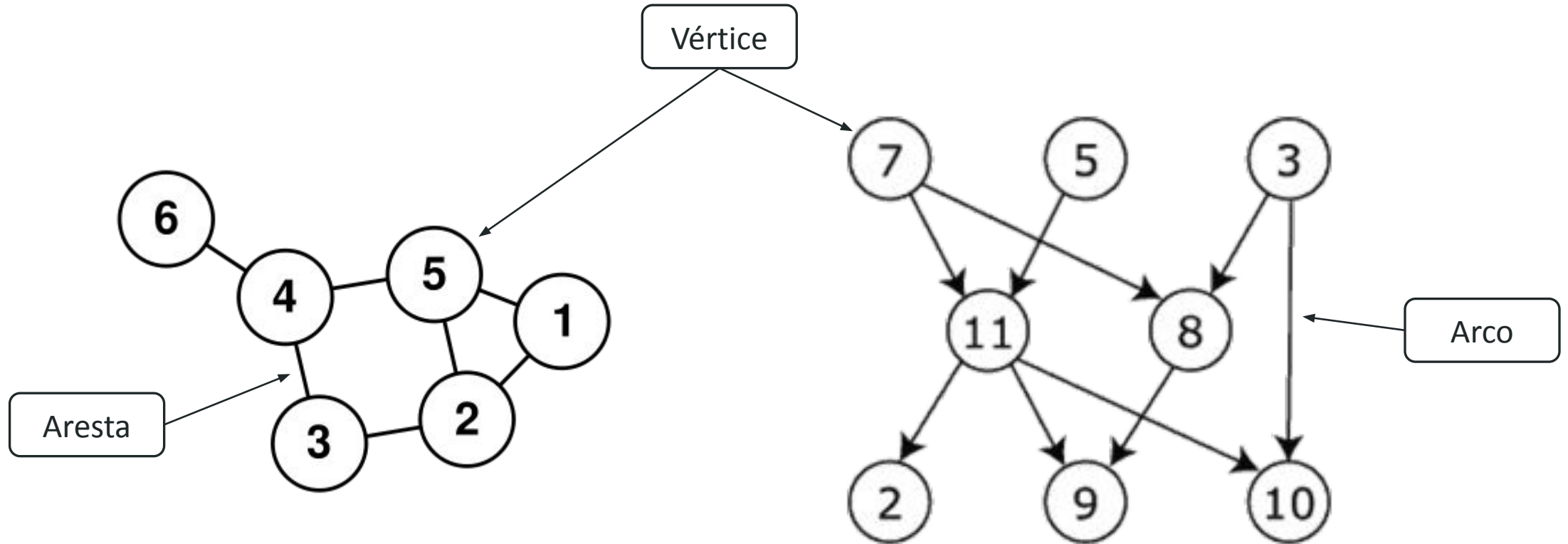
O que é um grafo?

- Utilizamos os grafos como um modo de estruturar nossos dados, com o objetivo de aproveitar sua base teórica para a resolução de problemas.

O que é um grafo?

- Ele é um conjunto de elementos, chamados vértices, e suas relações, que podem ser arestas, para grafos não direcionados, ou arcos, para grafos direcionados.
- Dois vértices conectados por uma aresta são considerados como adjacentes.

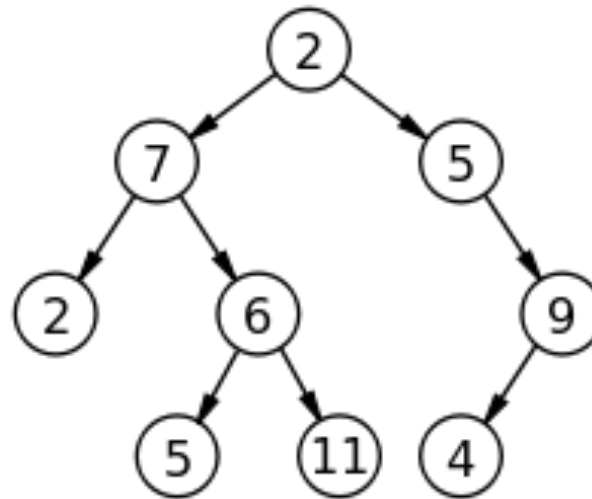
Exemplo - Grafos



Representação formal

- Notação: **$G = (V, A)$**
 - **G** : grafo
 - **V** : conjunto de vértices
 - **A** : conjunto de arestas ou arcos

Exemplos – Árvore?



O que representar?

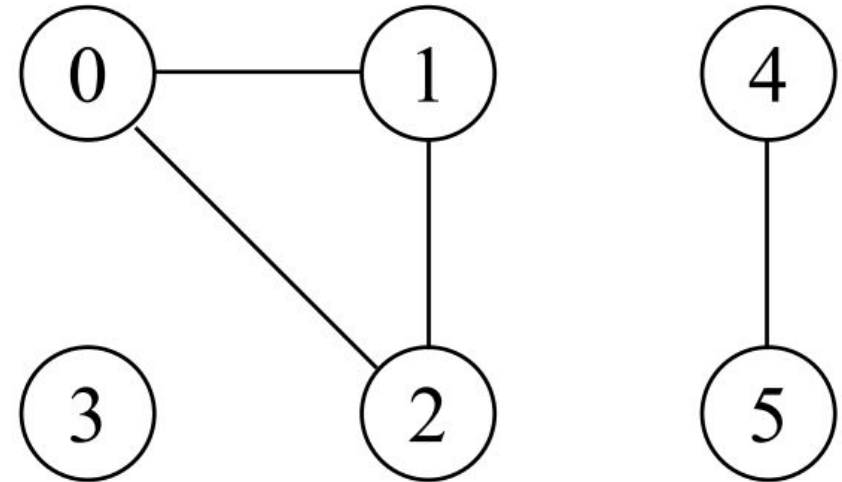


O que representar?



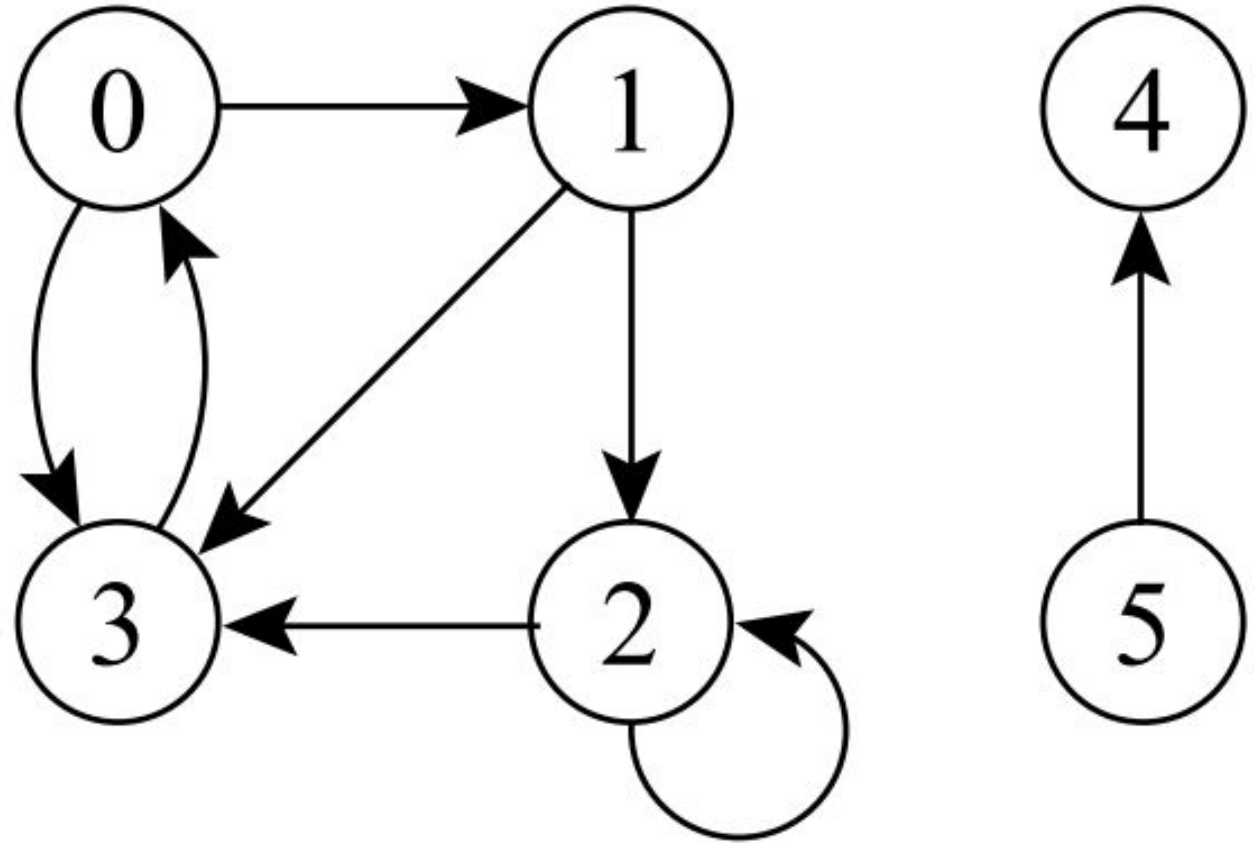
Grafos Não Direcionados

- Um grafo não direcionado **G** é um par **(V, A)**, onde o conjunto de arestas **A** é constituído de pares de vértices não ordenados.
 - Os arcos **(u, v)** e **(v, u)** são consideradas como uma única aresta. A relação de adjacência é simétrica.



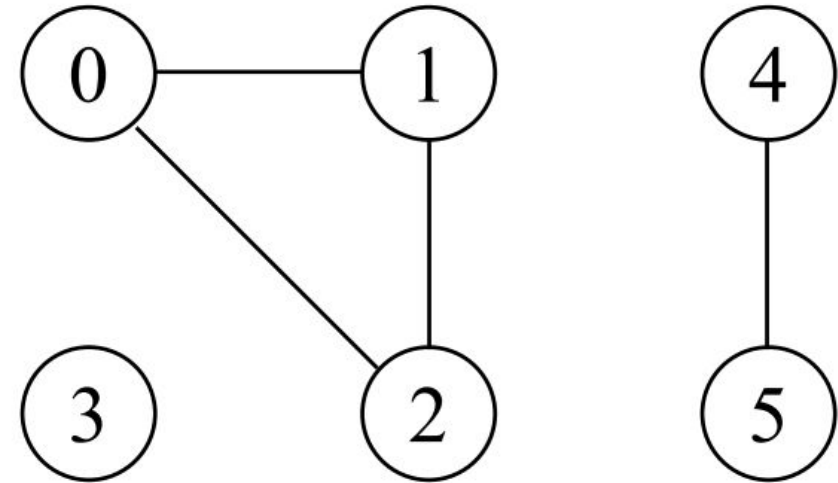
Grafos Direcionados

- Um grafo direcionado \mathbf{G} é um par (\mathbf{V}, \mathbf{A}) , onde \mathbf{V} é um conjunto finito de vértices e \mathbf{A} é uma relação binária em \mathbf{V} .
 - Um arco (\mathbf{u}, \mathbf{v}) sai do vértice \mathbf{u} e entra no vértice \mathbf{v} . O vértice \mathbf{v} é **adjacente** ao vértice \mathbf{u} .
 - Podem existir arestas de um vértice para ele mesmo, chamadas de **self-loops**.



Grau de um vértice

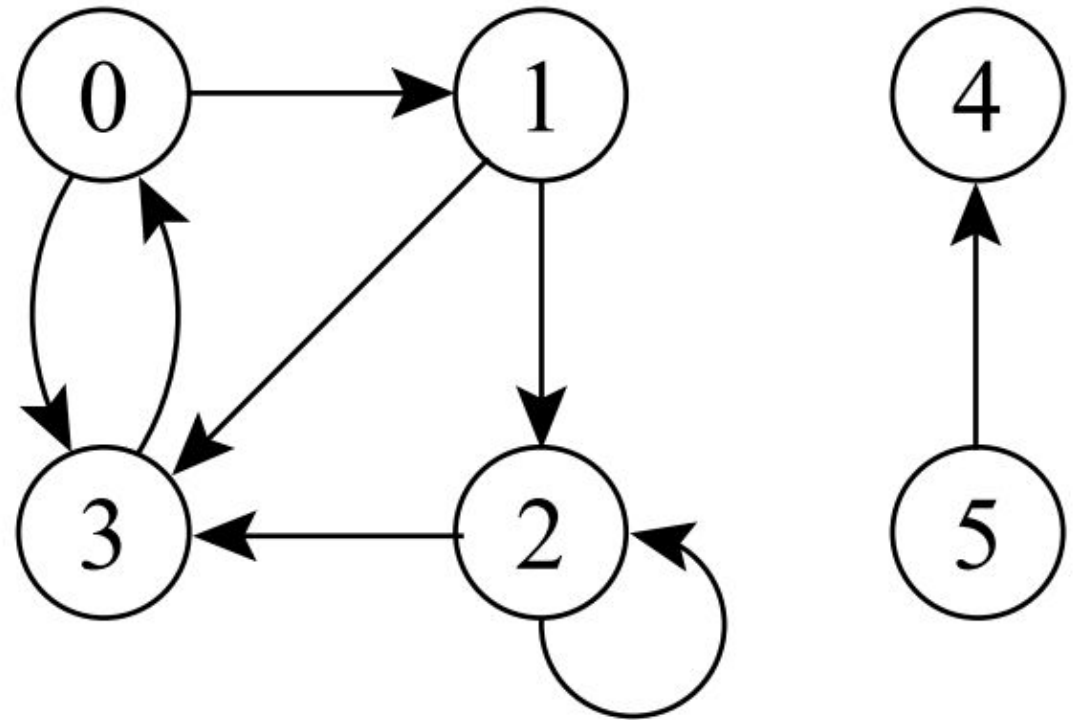
- Em grafos não direcionados:
 - O grau de um vértice é o número de arestas que incidem nele.
 - Um vértice de grau zero é dito isolado ou não conectado.
- Ex.: O vértice 1 tem grau 2 e o vértice 3 é isolado.



Grau de um vértice

Em grafos direcionados

- O grau de um vértice é o número de arestas que saem dele(out-degree) mais o número de arestas que chegam nele(in-degree).
- Ex.: O vértice 2 tem grau de entrada 2, grau de saída 2 e grau total 4.



Caminho entre vértices

- Um caminho de comprimento k de um vértice x a um vértice y em um grafo $G = (V, A)$ é uma sequência de vértices $(v_0, v_1, v_2, \dots, v_k)$ tal que $x = v_0$ e $y = v_k$, e $(v_{i-1}, v_i) \in A$ para $i = 1, 2, \dots, k$.

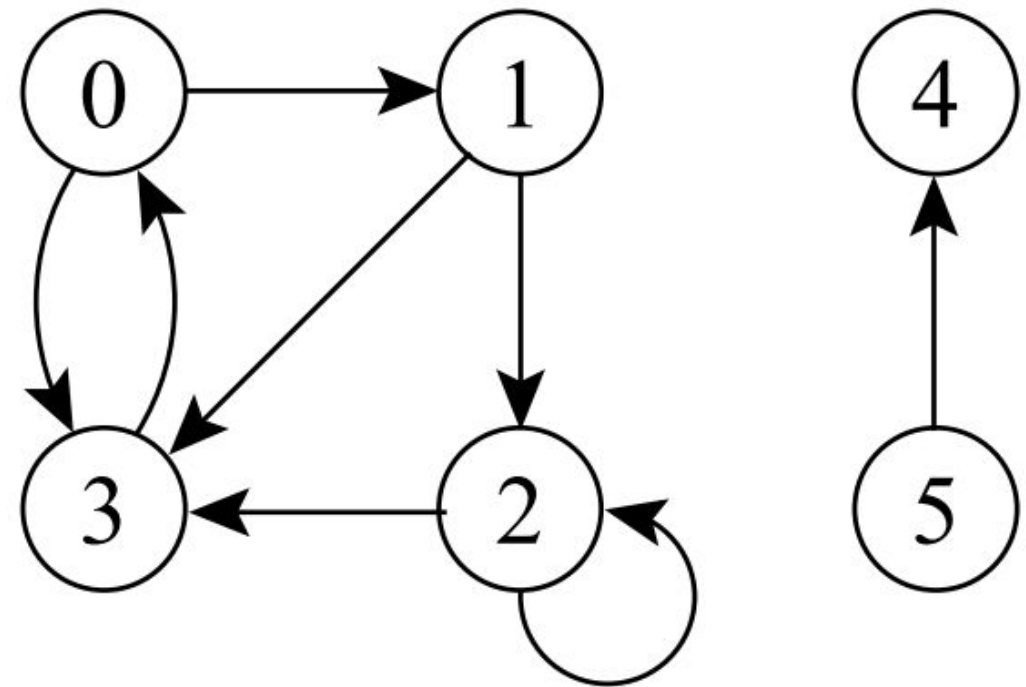
O comprimento de um caminho é o número de arestas nele, isto é, o caminho contém os vértices $v_0, v_1, v_2, \dots, v_k$ e as arestas $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$.

Se existir um caminho c de x a y então y é alcançável a partir de x via c .

Um caminho é simples se todos os vértices do caminho são distintos.

Caminho entre vértices

- Ex.: O caminho **(0, 1, 2, 3)** é simples e tem comprimento 3. O caminho **(1, 3, 0, 3)** não é simples



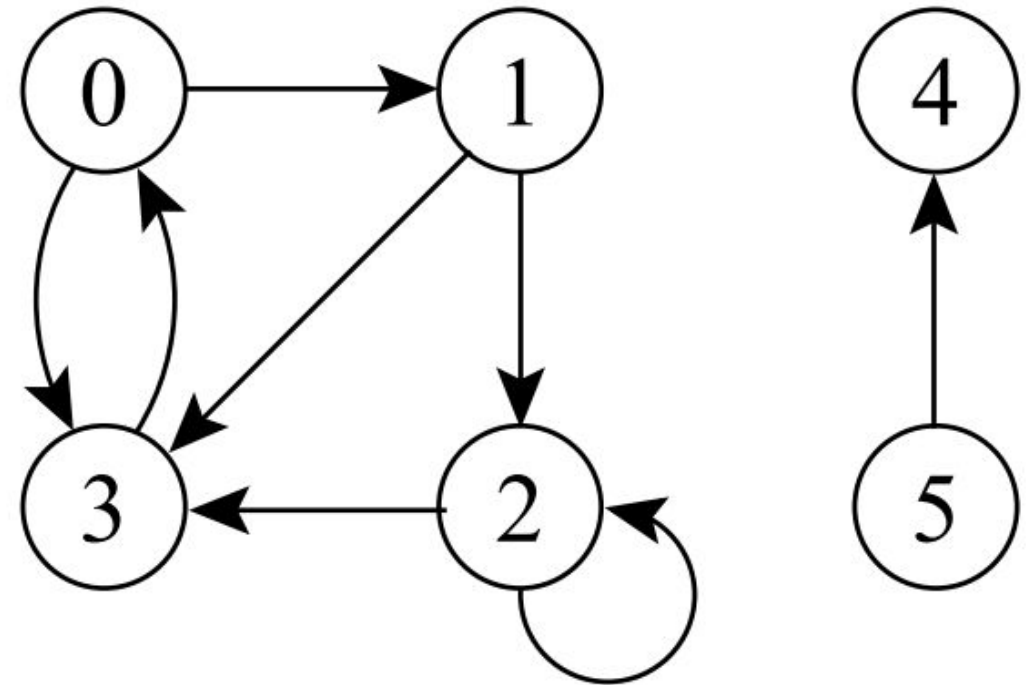
Ciclos



- Um caminho $(v_0, v_1, v_2, \dots, v_k)$ forma um ciclo se $v_0 = v_k$ e o caminho contém pelo menos uma aresta.
- O ciclo é simples se os vértices v_1, v_2, \dots, v_k são distintos.
- O **self-loop** é um ciclo de tamanho 1.
- Dois caminhos (v_0, v_1, \dots, v_k) e $(v'_0, v'_1, \dots, v'_k)$ formam o mesmo ciclo se existir um inteiro j tal que $v'_i = v'_{(i+j) \bmod k}$ para $i = 0, 1, \dots, k-1$.

Ciclos

- Ex.: O caminho **(0, 1, 2, 3, 0)** forma um ciclo. O caminho **(0, 1, 3, 0)** forma o mesmo ciclo que os caminhos **(1, 3, 0, 1)** e **(3, 0, 1, 3)**

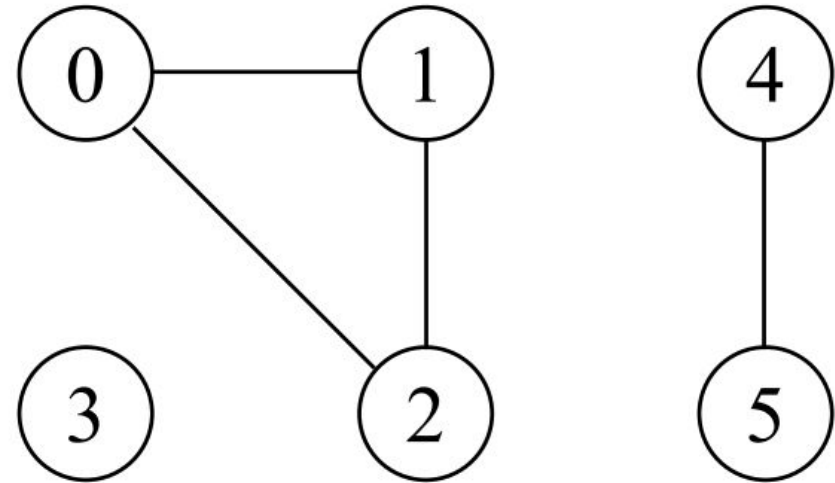


Componentes conectados

- Um grafo não direcionado é conectado se cada par de vértices está conectado por um caminho.
- Os componentes conectados são as porções conectadas de um grafo.
- Um grafo não direcionado é conectado se ele tem exatamente um componente conectado.

Componentes conectados

- Ex.: Os componentes são: **{0, 1, 2}**, **{4, 5}** e **{3}**

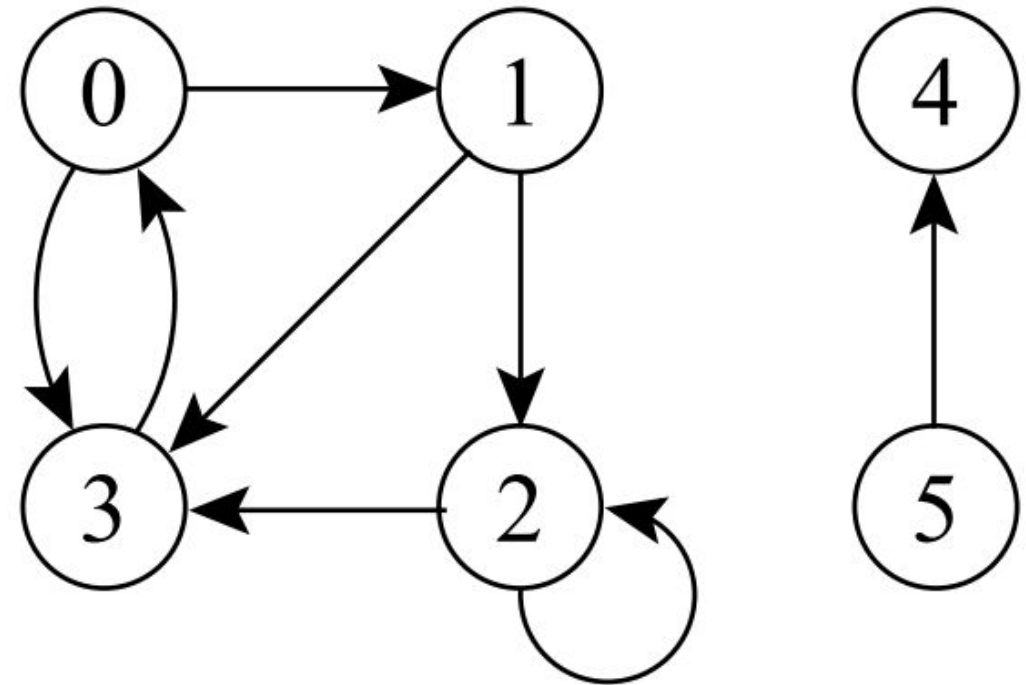


Componentes fortemente conectados

- Um grafo direcionado $\mathbf{G} = (\mathbf{V}, \mathbf{A})$ é fortemente conectado se cada dois vértices quaisquer são alcançáveis a partir um do outro.
- Os componentes fortemente conectados de um grafo direcionado são conjuntos de vértices sob a relação “são mutuamente alcançáveis”.
- Um grafo direcionado fortemente conectado tem apenas um componente fortemente conectado.

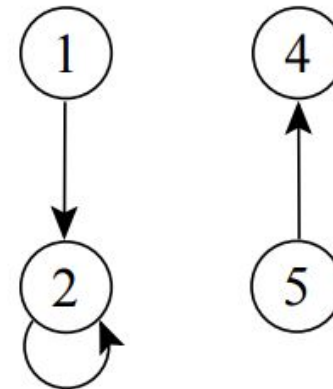
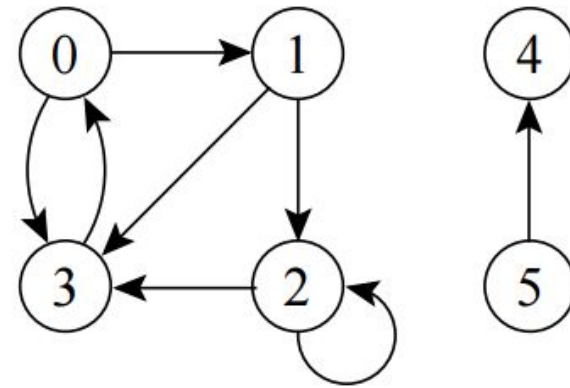
Componentes fortemente conectados

- Ex.: $\{0, 1, 2, 3\}$, $\{4\}$ e $\{5\}$ são os componentes fortemente conectados, $\{4, 5\}$ não o é pois o vértice **5** não é alcançável a partir do vértice **4**.



Subgrafos

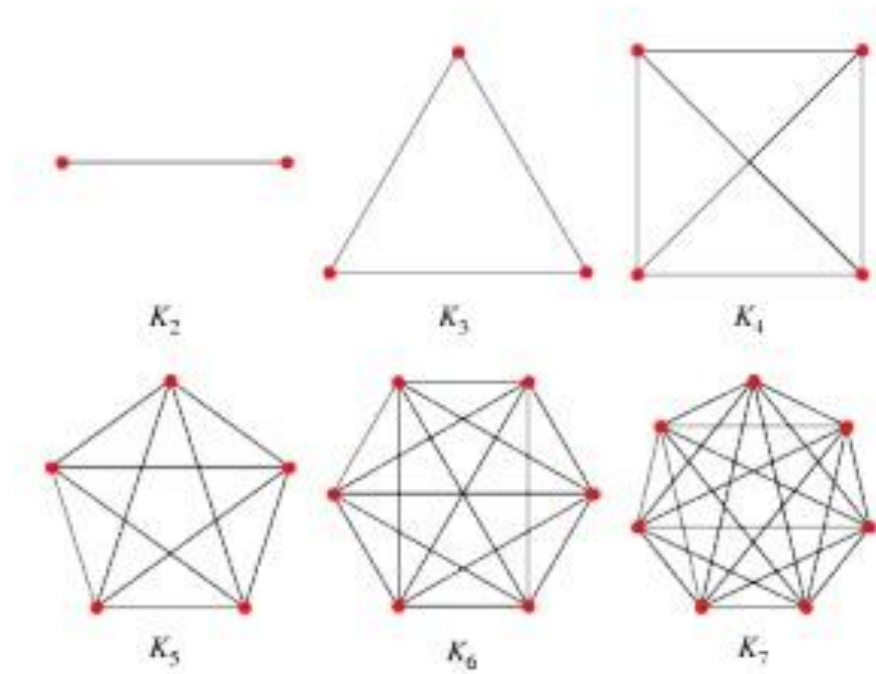
- Um grafo $\mathbf{G}' = (\mathbf{V}', \mathbf{A}')$ é um subgrafo de $\mathbf{G} = (\mathbf{V}, \mathbf{A})$ se $\mathbf{V}' \subseteq \mathbf{V}$ e $\mathbf{A}' \subseteq \mathbf{A}$.
- Dado um conjunto $\mathbf{V}' \subseteq \mathbf{V}$, o subgrafo induzido por \mathbf{V}' é o grafo $\mathbf{G}' = (\mathbf{V}', \mathbf{A}')$, onde $\mathbf{A}' = \{(u, v) \in \mathbf{A} \mid u, v \in \mathbf{V}'\}$.
- Ex.: Subgrafo induzido pelo conjunto de vértices **{1, 2, 4, 5}**



Grafos completos

- Um grafo completo é um grafo não direcionado no qual todos os pares de vértices são adjacentes.
- Possui $(|V|^2 - |V|) / 2$ arestas, pois do total de $|V|^2$ pares possíveis de vértices devemos subtrair $|V|$ **self-loops** e dividir por 2 (cada aresta ligando dois vértices é contada duas vezes).

Grafos completos



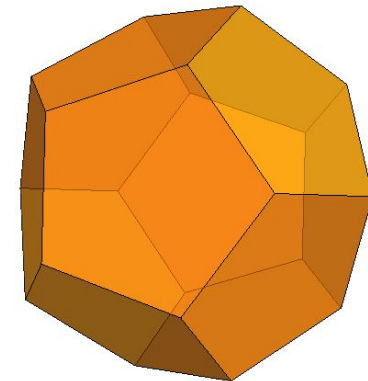
Outras definições de grafos

- **Grafo bipartido:** Grafo não direcionado $G = (V, A)$ no qual V pode ser particionado em dois conjuntos V_1 e V_2 tal que $(u, v) \in A$ implica que $u \in V_1$ e $v \in V_2$ ou $u \in V_2$ e $v \in V_1$ (todas as arestas ligam os dois conjuntos V_1 e V_2).

Grafo ponderado: possui pesos associados às arestas.

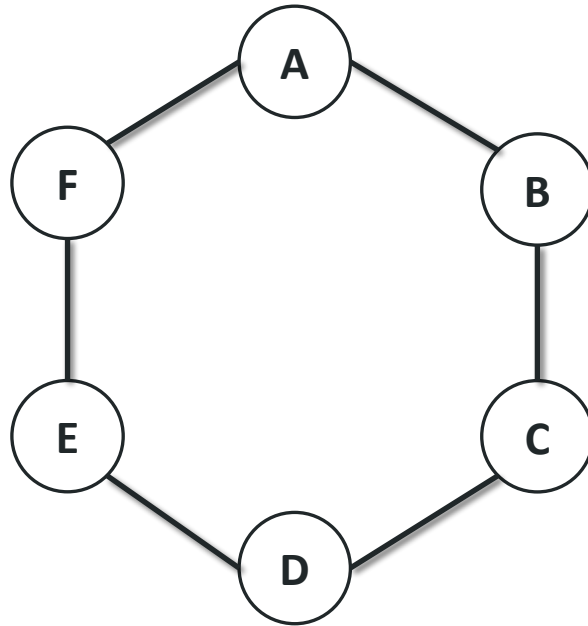
Grafos Hamiltonianos

- Um grafo Hamiltoniano é aquele que possui um ciclo Hamiltoniano.
- Um ciclo Hamiltoniano é um ciclo (em um grafo não dirigido) onde cada vértice é visitado exatamente uma vez retornando ao ponto de partida.
- Um caminho Hamiltoniano visita cada vértice uma vez.
- O nome vem de Sir William Rowan Hamilton (~1850), que criou um jogo chamado Icosian, onde o objetivo era achar um ciclo deste tipo em um dodecaedro.



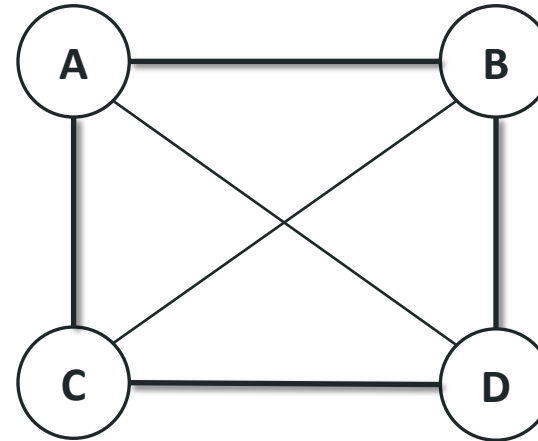
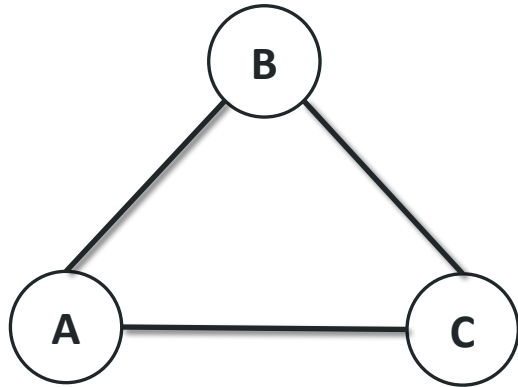
Grafos Hamiltonianos - Exemplos

- Todo grafo cíclico (composto por um único ciclo) é Hamiltoniano:



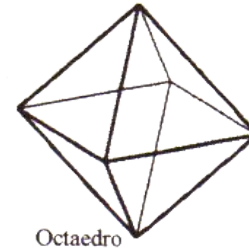
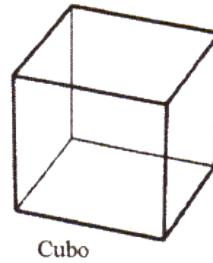
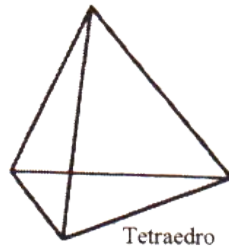
Grafos Hamiltonianos - Exemplos

- Um grafo completo com mais de dois vértices é Hamiltoniano



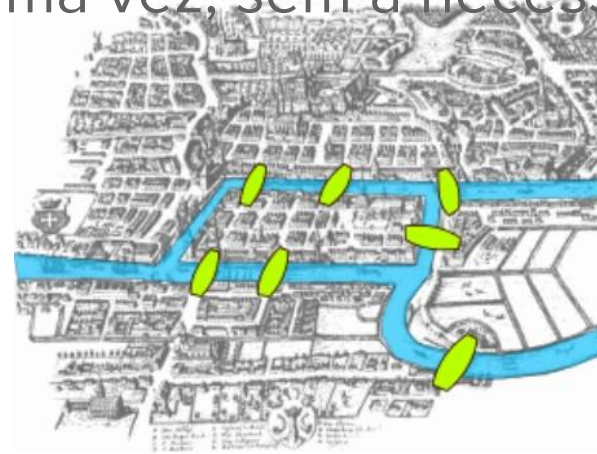
Grafos Hamiltonianos - Exemplos

- Os sólidos platônicos (polígonos regulares convexos) são Hamiltonianos



Grafo Euleriano

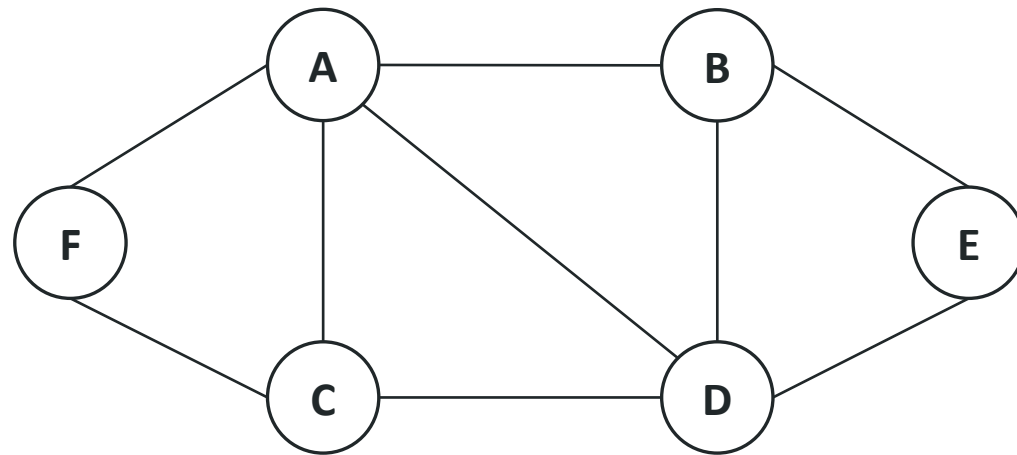
- Um grafo Euleriano é aquele que contém um ciclo Euleriano.
- Um ciclo Euleriano é um caminho em um grafo que visita cada aresta exatamente uma vez, iniciando e terminando no mesmo vértice.
- Euler – as Sete Pontes de Königsberg (1736).
- Um caminho Euleriano percorre cada aresta uma vez, sem a necessidade de voltar ao ponto de partida.



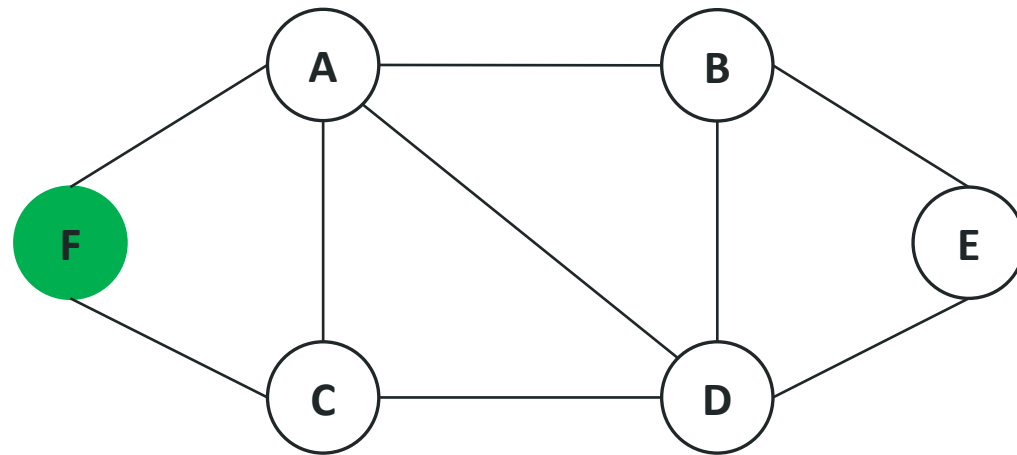
Grafo Euleriano

- Um grafo não-dirigido é Euleriano se não tiver nenhum vértice de grau ímpar
- Um grafo dirigido é Euleriano se todos os vértices tiverem grau de entrada igual ao seu grau de saída.
- Um grafo semi-Euleriano (que permite um caminho Euleriano) possui exatamente dois vértices de grau ímpar, um é o ponto de partida e outro é o ponto de chegada.

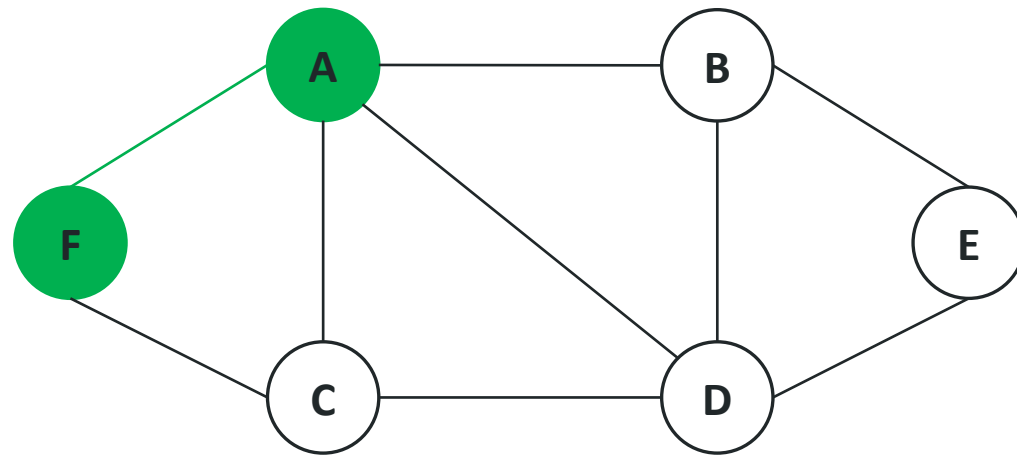
Exemplo - Hamiltoniano



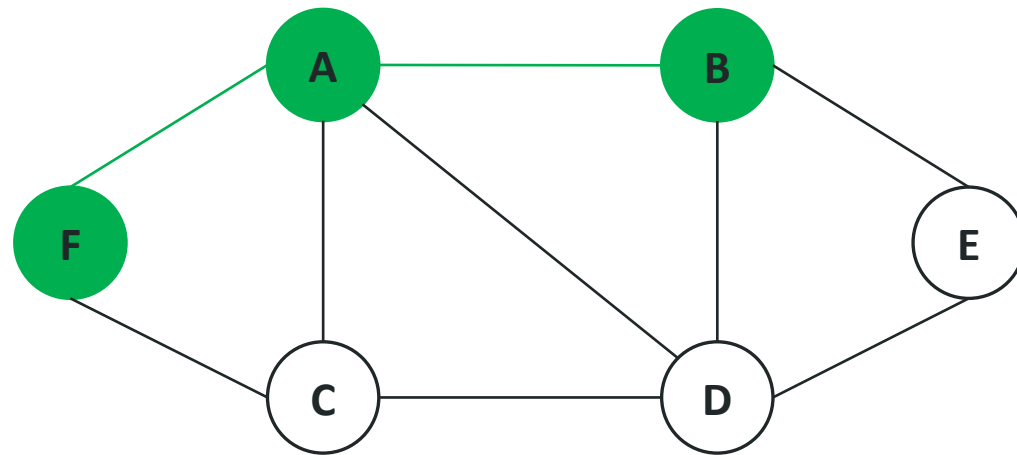
Exemplo - Hamiltoniano



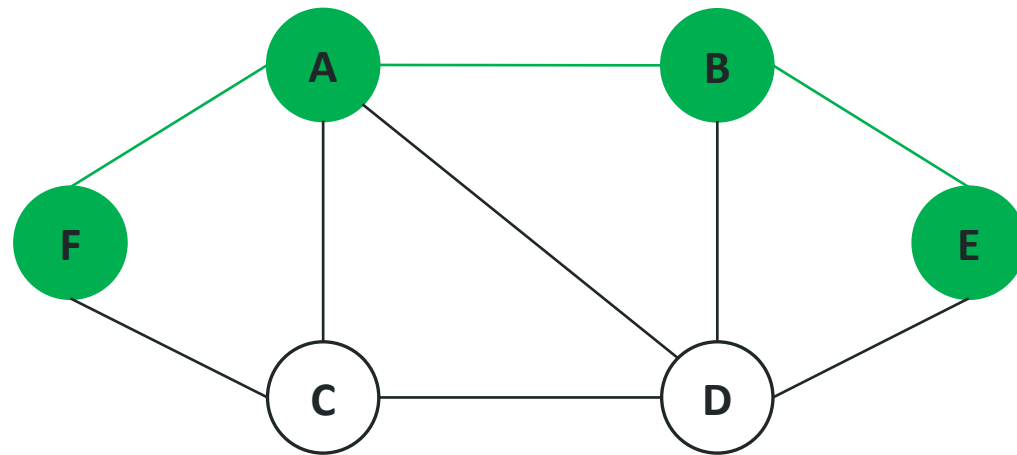
Exemplo - Hamiltoniano



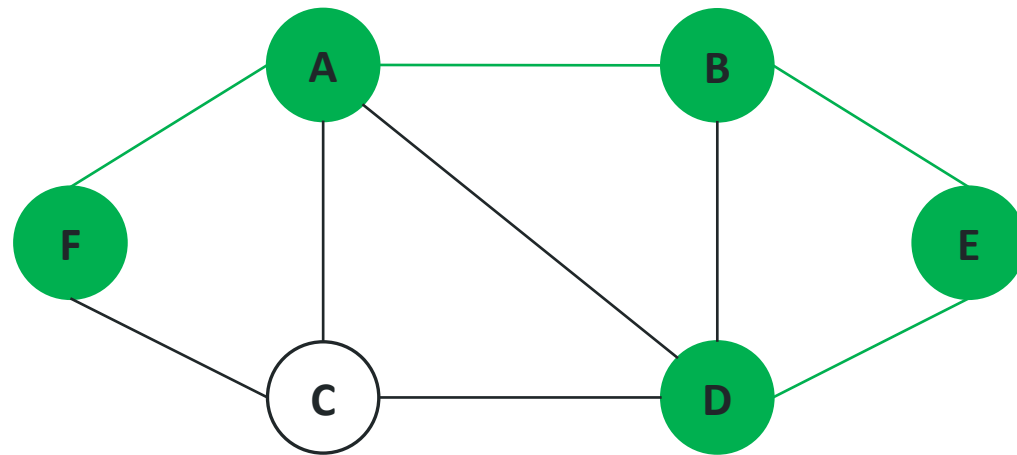
Exemplo - Hamiltoniano



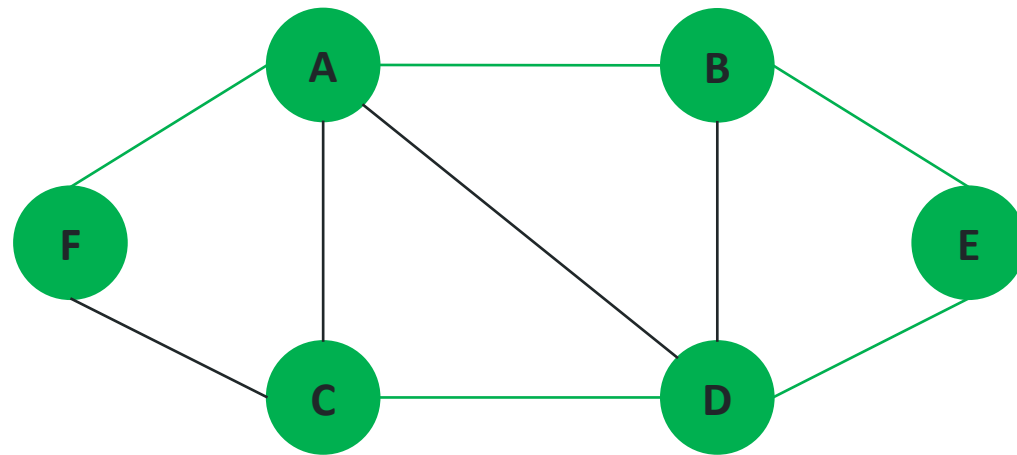
Exemplo - Hamiltoniano



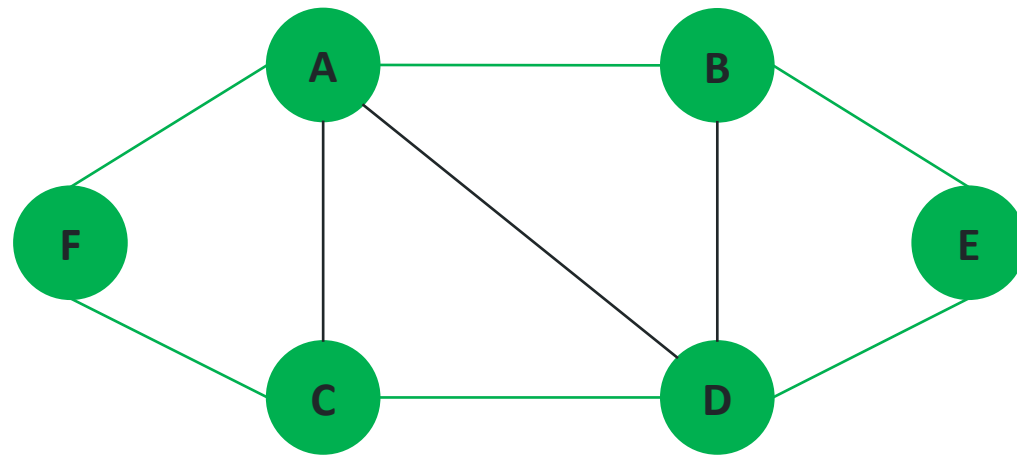
Exemplo - Hamiltoniano



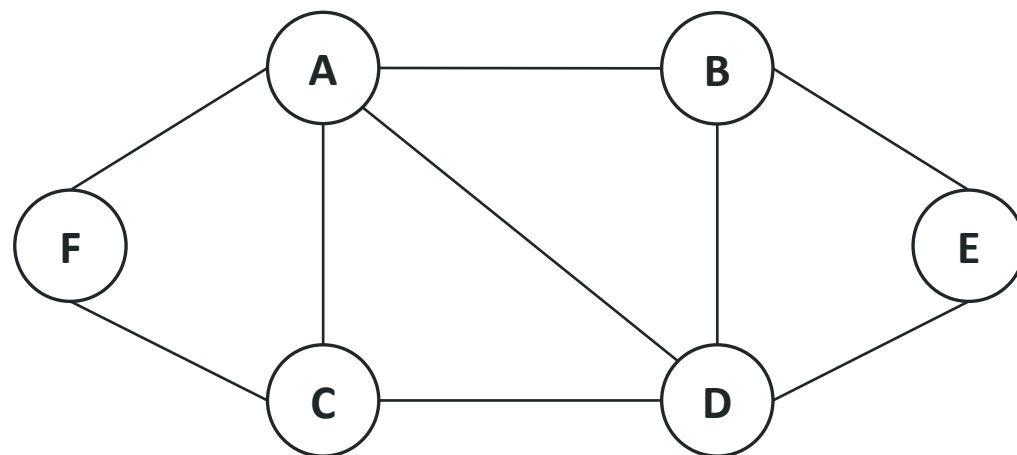
Exemplo - Hamiltoniano



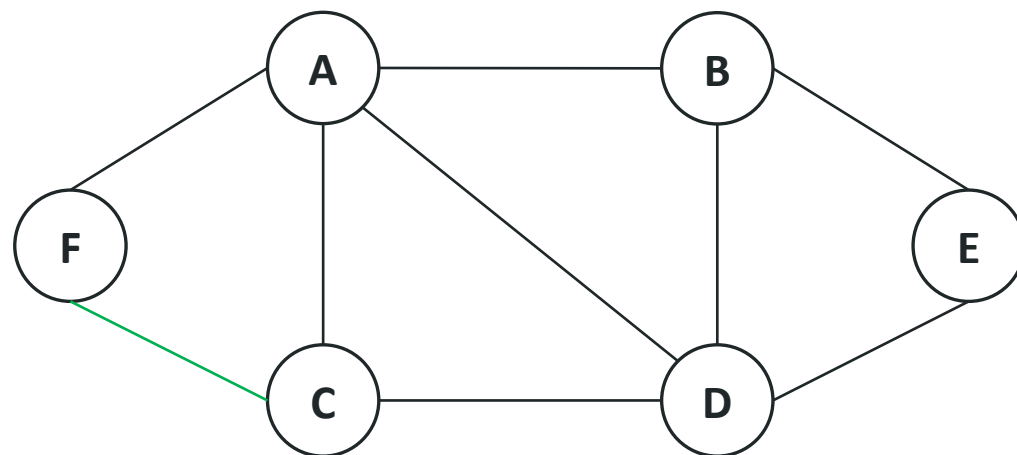
Exemplo - Hamiltoniano



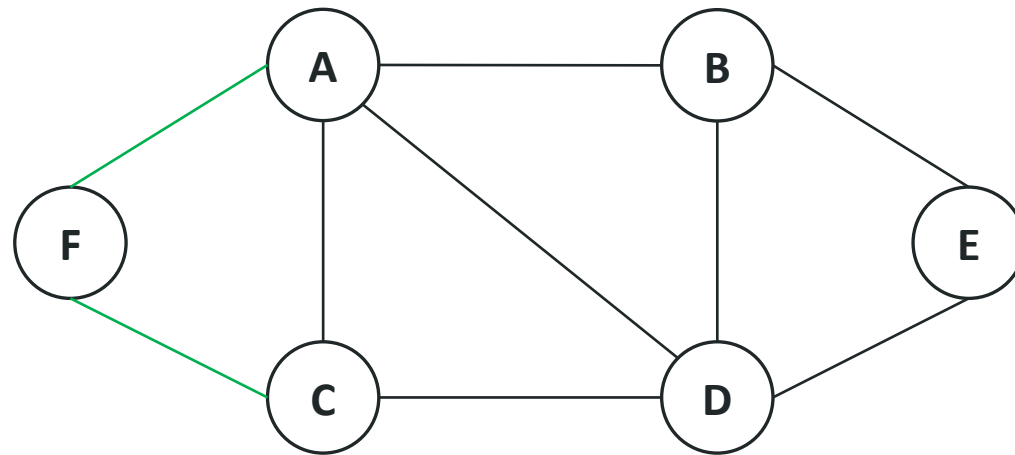
Exemplo - Semi-Euleriano



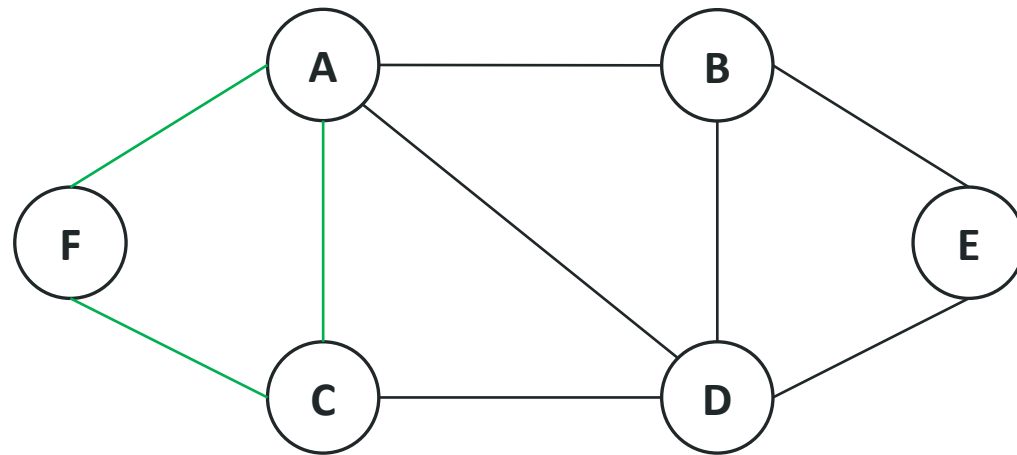
Exemplo - Semi-Euleriano



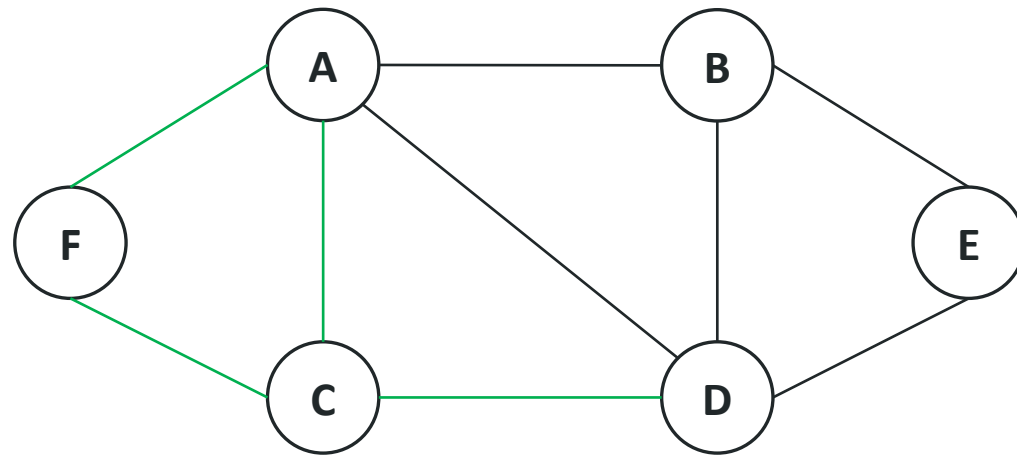
Exemplo - Semi-Euleriano



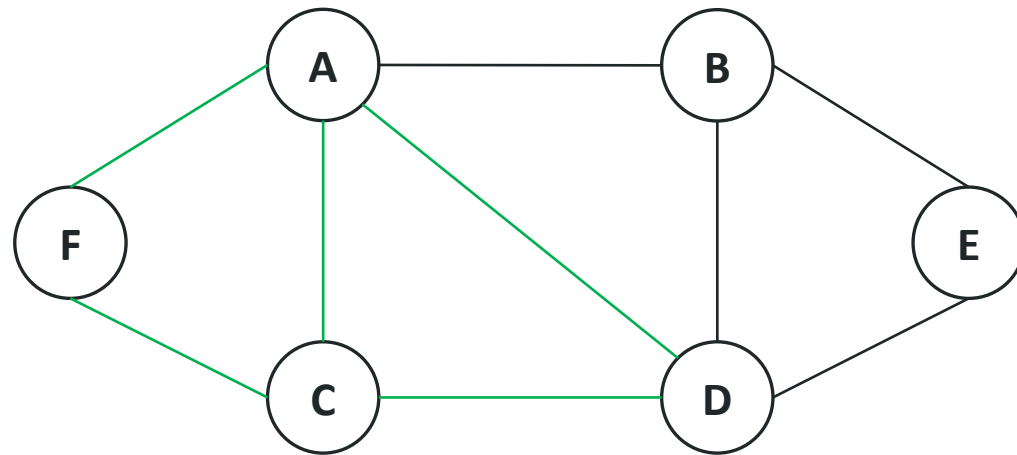
Exemplo - Semi-Euleriano



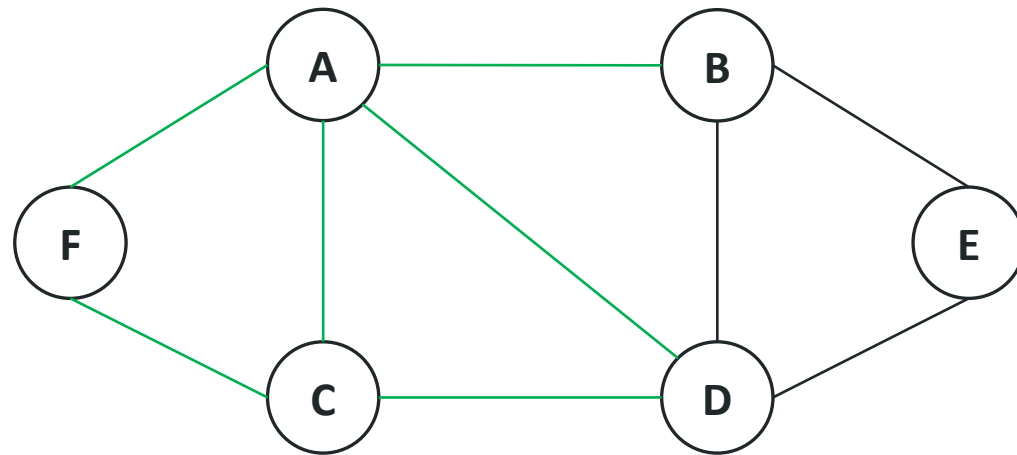
Exemplo - Semi-Euleriano



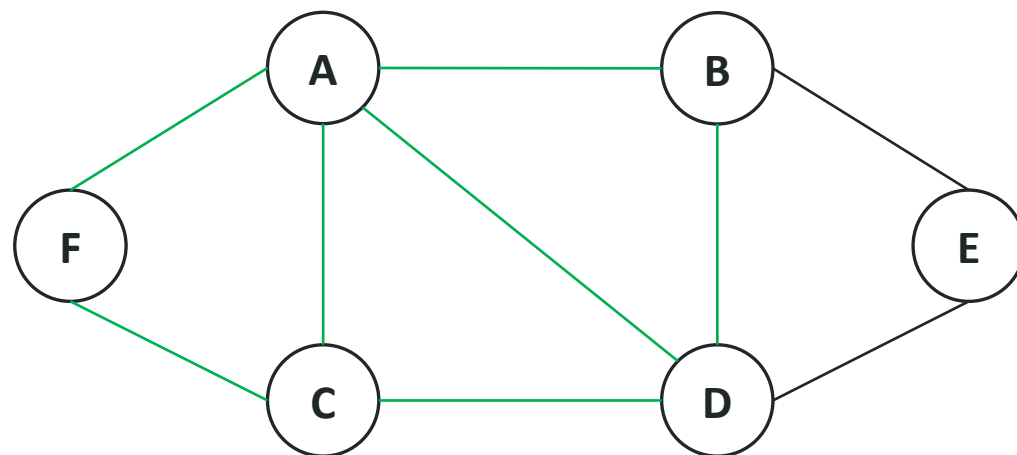
Exemplo - Semi-Euleriano



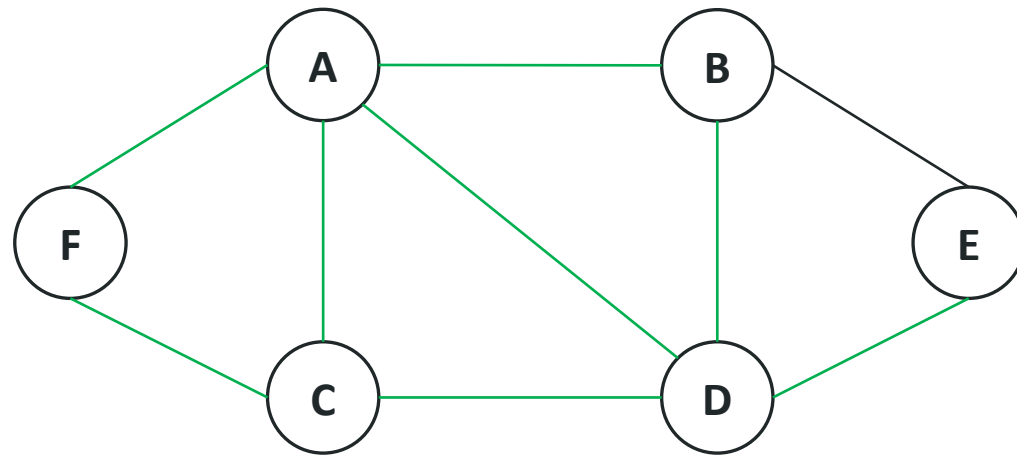
Exemplo - Semi-Euleriano



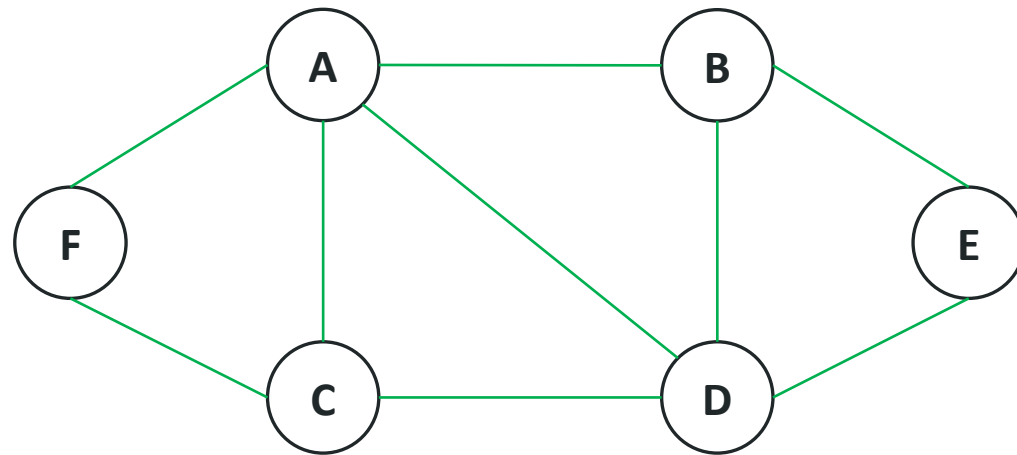
Exemplo - Semi-Euleriano



Exemplo - Semi-Euleriano



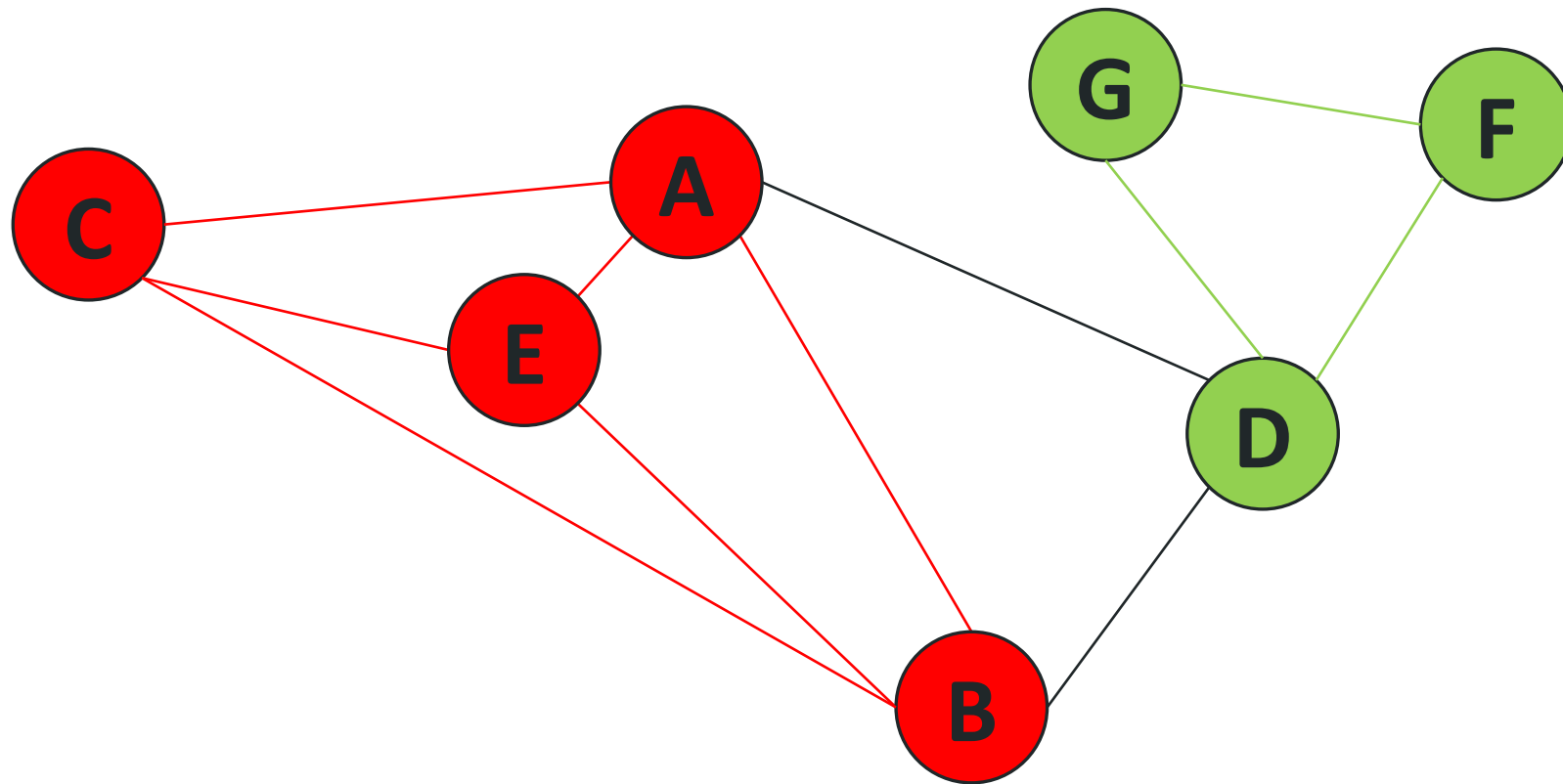
Exemplo - Semi-Euleriano



Clique

- Um clique em um grafo não direcionado $G = (V, E)$ é um subconjunto de vértices $C \subseteq V$, tal que para cada dois vértices em C , existe uma aresta os conectando. Isso se equivale a dizer que um subgrafo induzido de C é completo (em alguns casos, o termo clique também pode ser referência ao subgrafo).

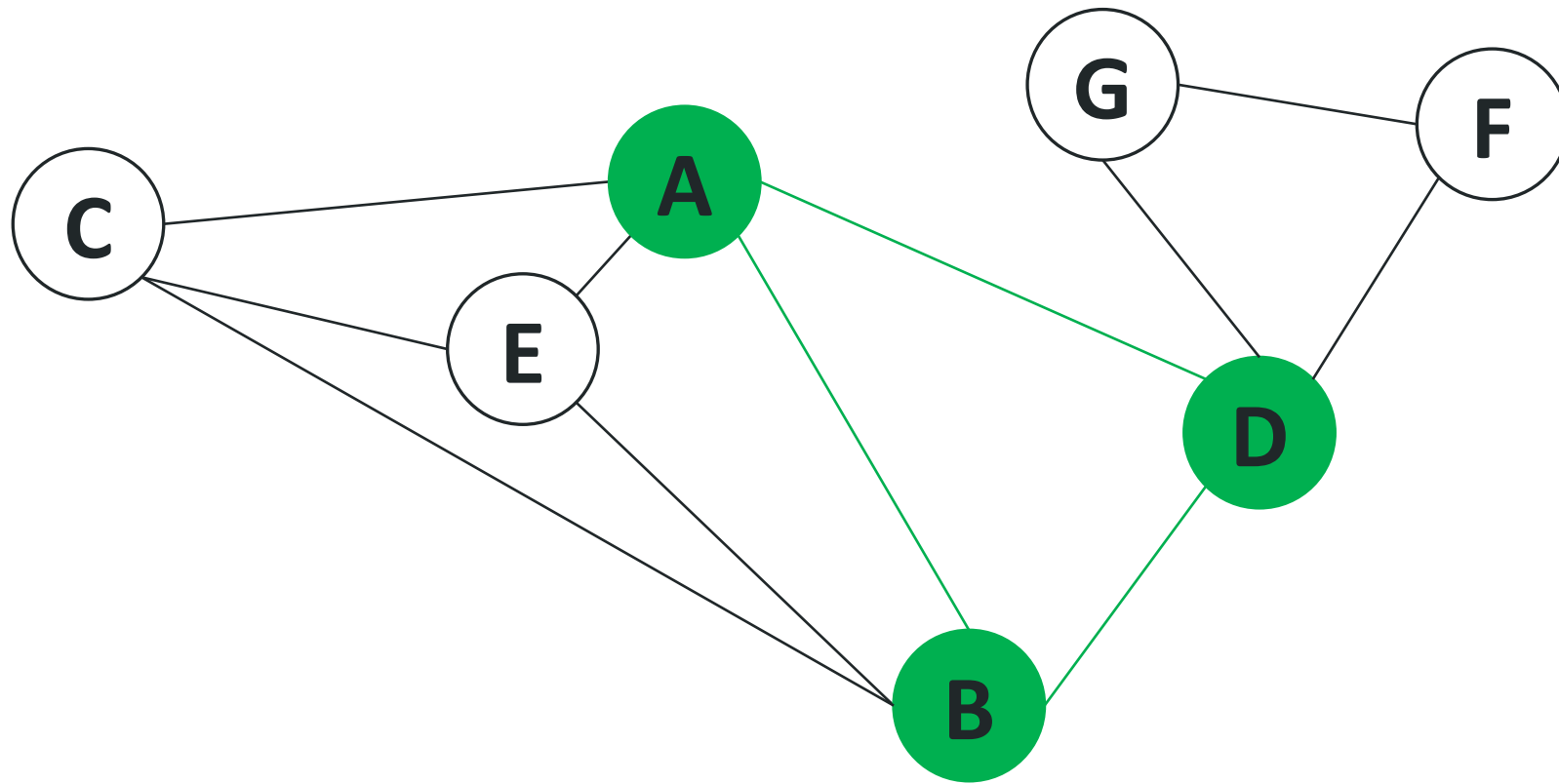
Clique



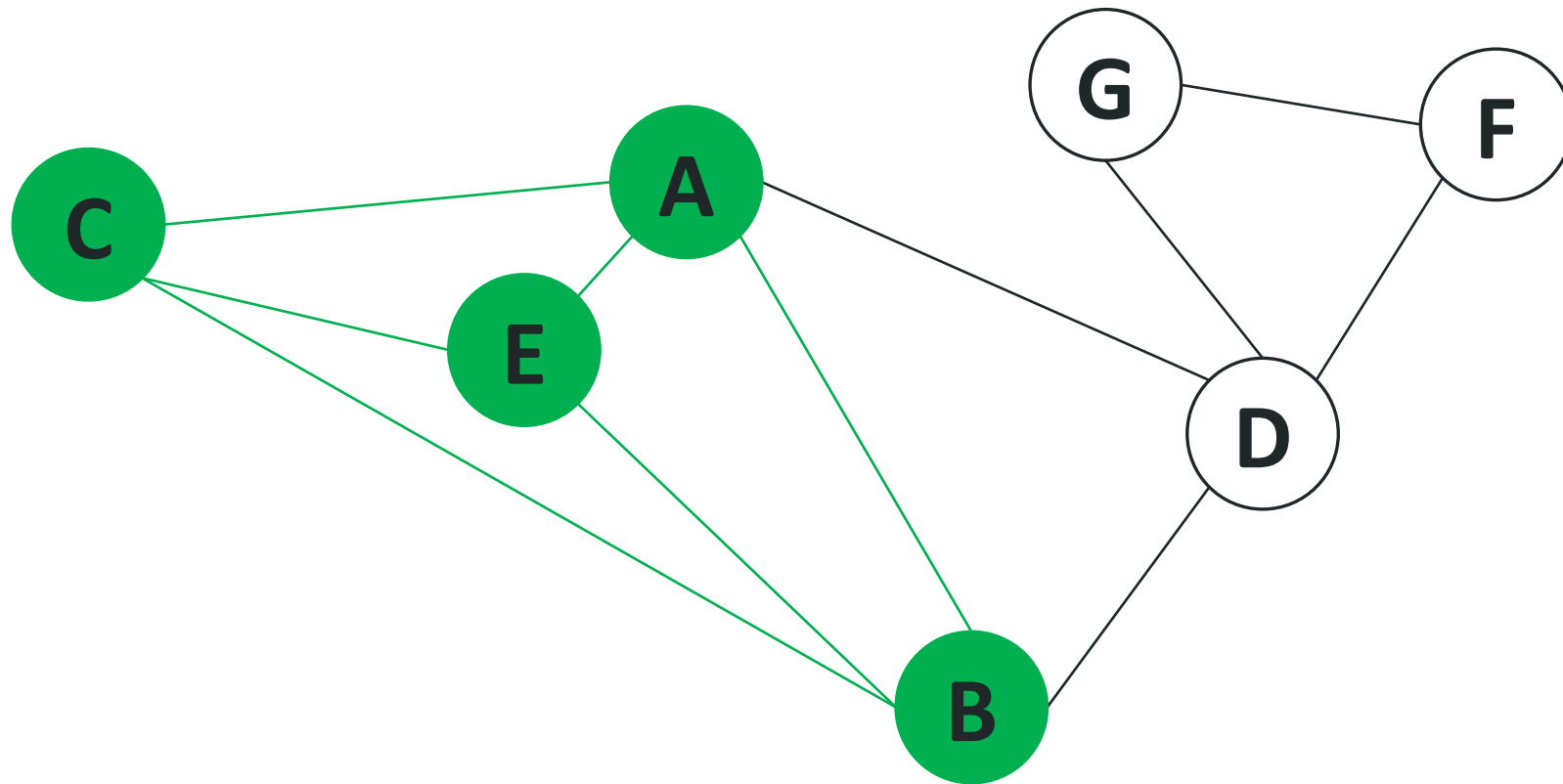
Clique

- Um **clique maximal** é um clique que não pode ser estendido ao se adicionar um ou mais vértices adjacentes, ou seja, um clique que não existe exclusivamente dentro de um conjunto de vértices de um clique maior.
- Um **clique máximo** é o maior clique possível em um dado grafo. O número do clique $\omega(G)$ de um grafo G é o número de vértices de um clique máximo em G . O número da intersecção de G é o menor número de cliques que, juntos, cobrem todas as arestas de G .

Clique Maximal



Clique Máximo



Aplicações

- A palavra "clique", no seu uso na teoria dos grafos, veio do trabalho de Luce & Perry (1949), que utilizou subgrafos completos para modelar cliques (grupos de pessoas que se conhecem) em redes sociais. Para trabalhos contínuos de como modelar cliques sociais, veja e.g. Alba (1973), Peay (1974), e Doreian & Woodard (1994).
- Vários problemas da bioinformática foram modelados utilizando clique. Na química, Rhodes et al. (2003) utilizou cliques para descrever produtos químicos em um banco de dados que possuía um alto grau de similaridade com a estrutura alvo. Kuhl, Crippen & Friesen (1983) utilizou cliques para modelar as posições no qual dois produtos químicos vão se associar mutualmente.

Visualização On-line

- Ferramenta on-line para grafos:
- - <https://graphonline.top/en/>

Formas de representação de grafos

- Em código, não podemos manter a representação visual para controle do grafo.
- Partimos de duas formas principais para a representação em código: Lista e Matriz de adjacência.
- Ambas as representações têm vantagens e desvantagens. Identificamos essas características ao decorrer da matéria.

Matriz de adjacência

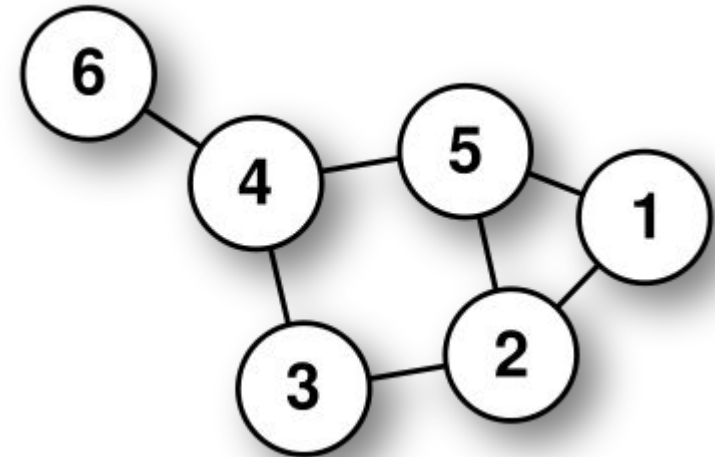
- Principal representação utilizada para a construção de grafos
- Alguns dos algoritmos estudados são facilmente aplicados em matrizes, enquanto tem um grande grau de dificuldade em outras representações.

Matriz de adjacência

- O grafo é representado em uma matriz quadrada, onde cada linha é correspondente ao vértice de origem de uma ligação, e a coluna ao vértice de destino.
- Normalmente o valor de cada célula é definido como **0** para a ausência de aresta/arco entre os vértices, e diferente de **0** para a presença de uma aresta/arco.

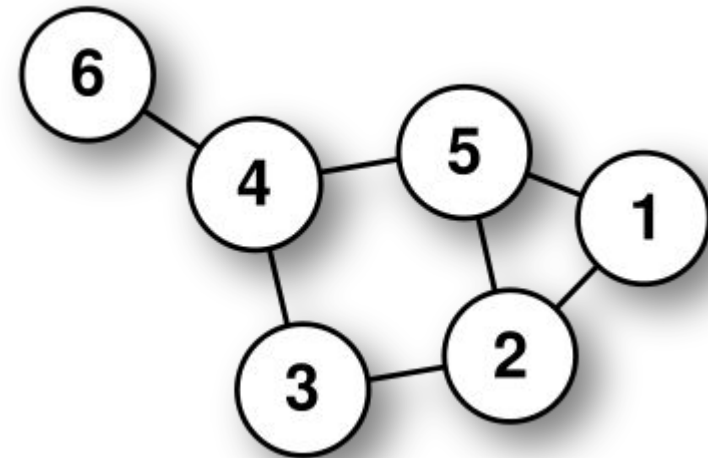
Matriz de adjacência

	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0



Matriz de adjacência

	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0



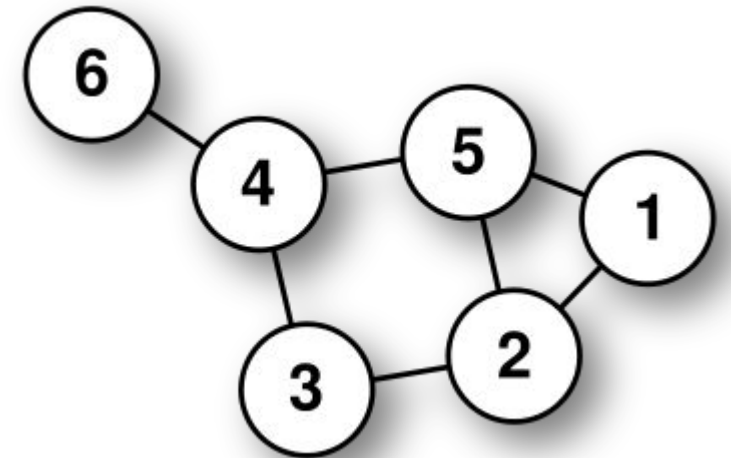
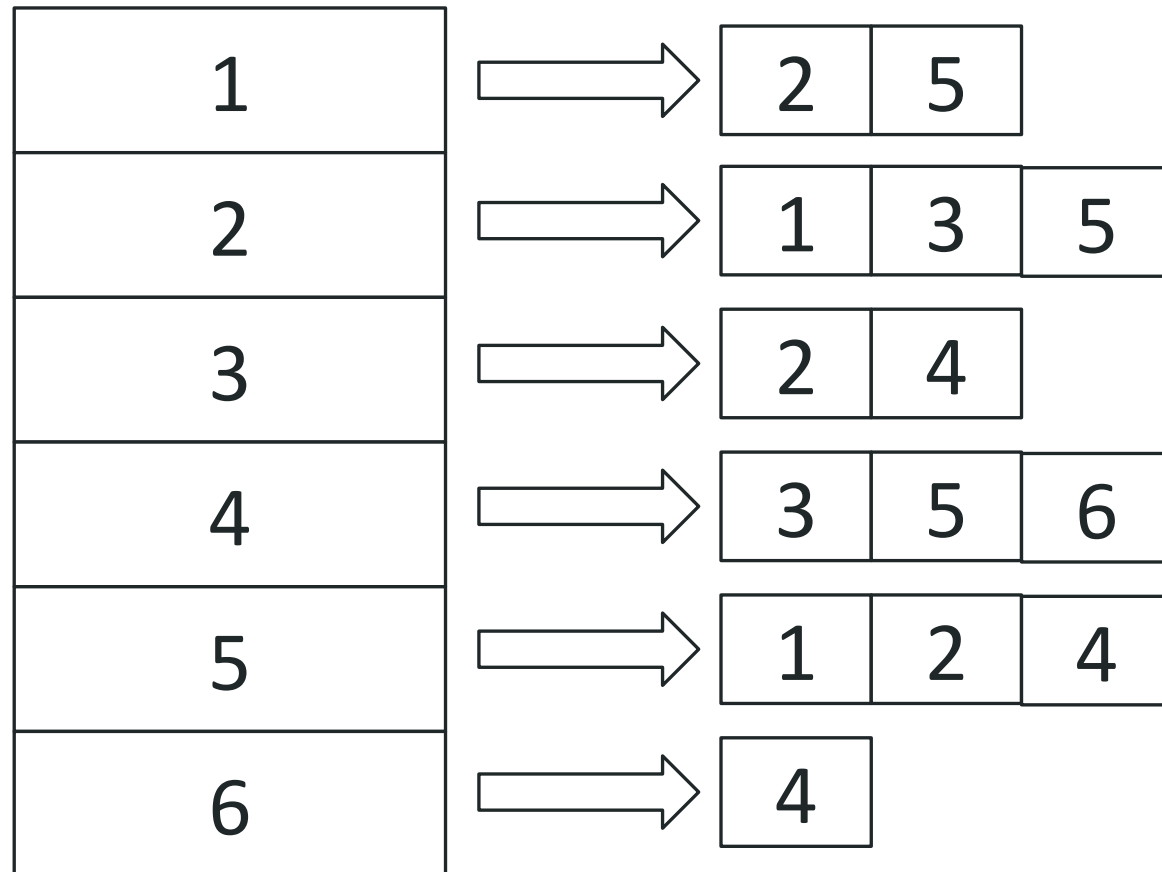
Matriz de adjacência

- Deve ser utilizada para grafos densos, onde $|\mathbf{A}|$ é próximo de $|\mathbf{V}|^2$.
- O tempo necessário para acessar um elemento é independente de $|\mathbf{V}|$ ou $|\mathbf{A}|$.
- É muito útil para algoritmos em que precisamos saber com rapidez se existe uma aresta ligando dois vértices.
- A maior desvantagem é que a matriz necessita $\Omega(|\mathbf{V}|^2)$ de espaço. Ler ou examinar a matriz tem complexidade de tempo $\mathbf{O}(|\mathbf{V}|^2)$.

Lista de adjacência

- Este é um método mais intuitivo pensando em orientação a objetos.
- Mantemos uma lista principal dos vértices que compõem o grafo.
- Por sua vez, cada um dos vértices possui uma lista de referência aos vértices com o qual possuem alguma ligação.

Lista de adjacência

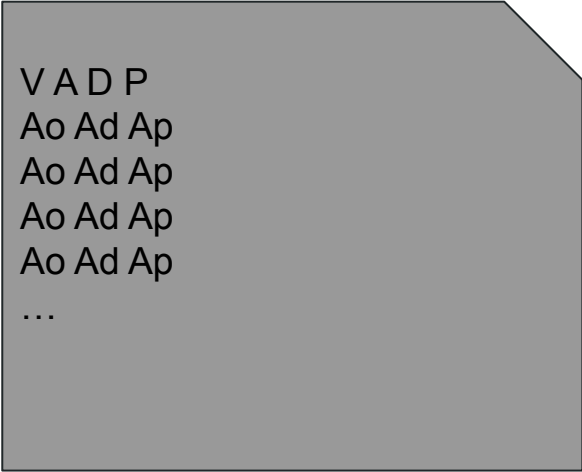


Lista de adjacência

- Os vértices de uma lista de adjacência são em geral armazenados em uma ordem arbitrária.
- Possui uma complexidade de espaço $O(|V| + |A|)$
- Indicada para grafos esparsos, onde $|A|$ é muito menor do que $|V|^2$.
- É compacta e usualmente utilizada na maioria das aplicações.
- A principal desvantagem é que ela pode ter tempo $O(|V|)$ para determinar se existe uma aresta entre o vértice i e o vértice j , pois podem existir $O(|V|)$ vértices na lista de adjacentes do vértice i .

Leitura de Grafos

Existem alguns formatos de descrição de grafos em arquivos, e alguns formatos livres, para o primeiro trabalho não vai ser pedido, mas vamos trabalhar com grafos carregados em arquivos, que serão texto simples com o seguinte formato:



```
V A D P
Ao Ad Ap
Ao Ad Ap
Ao Ad Ap
Ao Ad Ap
...
```

Onde:

V = Número de vértices

A = Número de Arestas

D = Direcionado ou não (0 para não e 1 para sim)

P = Ponderado ou não (0 para não e 1 para sim)

A primeira linha é sempre presente seguida de A linhas, uma linha para cada aresta, onde:

Ao = Vértice de origem da aresta

Ad = Vértice de destino da aresta

Ap = Peso da aresta (somente presente em grafos ponderados)

Lembrando que em grafos não direcionados as arestas devem ser criadas nos dois sentidos

Trabalho - Estrutura Básica

Estrutura Básica

- Classe Grafos:

- O construtor deve pedir duas informações:
 - Se é Direcionado ou não
 - Se é Ponderado ou não
- Isso afetará diretamente como as arestas serão inseridas.

- Duas especializações da classe Grafos:

- GrafoMatriz, que usa uma matriz de adjacência como representação do grafo.
- GrafoLista, que usa uma lista de adjacência como representação do grafo e tem uma estrutura Aresta como auxilio.

Operações básicas

- **bool inserirVertice(string label);**
- Adiciona um vértice sem nenhuma aresta associada a ele, pode parecer igual em ambos os casos, mas não é.
- Precisamos adicionar esse vértice no vetor de vértices e também alocar seu espaço para as arestas.

Operações básicas

- **bool removerVertice(int indice);**
- Remove um vértice do grafo, elimina a linha e coluna dele da matriz e a referência dele da lista, junto com todas as arestas que chegam e saem dele.

Operações básicas

- **string labelVertice(int indice);**
- Funções básicas para retornar o nome de um vértice.

Operações básicas

- **void imprimeGrafo();**
- Imprimir o grafo no console, tentem deixar próximo da representação dos slides (não precisa da grade da matriz).

Operações básicas

- **`bool inserirAresta(int origem, int destino, float peso = 1);`**
- Essa operação deve ter um cuidado especial, ela deve ser executada levando em conta o tipo do grafo. No caso de um grafo ponderado o peso deve ser aplicado e no caso de um grafo direcionado, uma ligação de volta deve ser adicionada;

Operações básicas

- **bool removerAresta(int origem, int destino);**
- Remove uma aresta entre dois vértices no grafo, lembrando que no grafo não direcionado deve ser removida a aresta de retorno também;

Operações básicas

- **bool existeAresta(int origem, int destino);**
- Verifica a existência de uma aresta, aqui vemos uma diferença grande entre matriz e lista.

Operações básicas

- **float pesoAresta(int origem, int destino);**
- Retorne o peso de uma aresta, aqui vemos uma diferença grande entre matriz e lista.

Operações básicas

- **`vector<int> retornarVizinhos(int vertice);`**
- Função para retorno dos vizinhos de um vértice, necessária pois não teremos acesso a estrutura das arestas para os próximos algoritmos.