Percursos

- **Passeio (walk):** é uma sequência de vértices $v_1, v_2, \dots, v_k$ tal que cada par consecutivo de vértices (v_j, v_{j+1}) forma uma aresta do grafo. O comprimento de um passeio é o número de arestas que ele percorre.
  - **Trilha (Trail):** é um passeio que não repete arestas.
    - **Trilha fechada:** é uma trilha em que o primeiro e último vértices são iguais, ou seja ela forma um ciclo, mas pode repetir vértices, desde que não repita arestas.
- **Caminho (Path):** é um passeio que não repete vértices.
  - **Caminho fechado:** é um caminho que começa e termina no mesmo vértice formando um ciclo sem repetir vértices intermediários
- **Ciclo (Cycle):** é um caminho fechado, ou seja, começa e termina no mesmo vértice, sem repetição dos demais vértices (exceto o primeiro/último). O ciclo deve conter pelo menos 3 vértices, ou seja $K \geq 3$.

Percursos Especiais:

- **Trilha fechada Euleriana:** uma trilha fechada que percorre todas as arestas do grafo sem repeti-las. Se um grafo possui uma trilha euleriana, ele é chamado de grafo euleriano. Nessa trilha a repetição de vértices é permitida.
- **Trilha Euleriana (não fechada):** uma trilha que percorre todas as arestas do grafo sem repeti-las, mas não retorna ao vértice de origem. Um grafo que possui esse tipo de trilha é chamado de grafo semi-Euleriano. Vértices podem se repetir.
- **Ciclo Hamiltoniano:** um ciclo que passa por todos os vértices do grafo exatamente uma vez (exceto vértice de início / fim). O grafo é chamado de Hamiltoniano
- **Caminho Hamiltoniano:** caminho aberto que visita todos os vértices do grafo sem repetições. O grafo é chamado de semi-Hamiltoniano.

Distância em Grafos

- **Distância $d(v,w)$:** Número mínimo de arestas no caminho entre dois vértices v e w.
- **Excentricidade:** A maior distância entre um vértice v e qualquer outro vértice w do grafo.
- **Centro do grafo:** conjunto dos vértices com a menor excentricidade, os mais centrais do grafo.

Componentes Conexos

- **Grafo conexo:** existe um caminho entre qualquer par de vértices.
- **Grafo desconexo:** há vertices entre os quais não existe caminho. Um grafo sem arestas é totalmente desconexo.
- **Componente conexo:** subgrafo conexo maximal, que não é possível adicionar vértices sem perder a conectividade

Buscas em grafos:

buscas percorrem todos os vértices do grafo, geralmente ignorando pesos.

- **Busca em profundidade (DFS):** vai o mais fundo possível em cada ramificação antes de voltar e explorar caminhos alternativos. Usa pilha ou recursão.
- **Busca em largura (BFS):** visita todos os vizinhos do vértice atual antes de explorar seus vizinhos, e assim por diante, usa fila.

Dijkstra

- Determina o caminho mais curto entre dois vértices predeterminados em um grafo. Ele é projetado para grafos com arestas ponderadas com pesos positivos.
- **Algoritmo:** x = origem, y = destino.
  - $d[v]$: guarda o custo mínimo para chegar ao vértice v a partir da origem
  - $s[v]$: Predecessor (ou pai) de cada vértice $s[v]$ é predecessor de v
  - $vis[v]$: indica se v foi visitado.

  1. **Inicialização:**
     - Define $d[x]=0$ (distância da origem a ela mesma).
     - Para todos os outros vértices z, define $d[z] = \infty$ e $s[z] = X$.
     - $vis[\,] = \{false\}$, inicia todos falso.
  2. **laço principal:**
     - Enquanto o destino y não estiver visitado, $vis[y] == false$:
       - escolhe o vértice p não visitado com menor $d[p]$.
       - $vis[p] = true$.
       - Para cada vizinho z de p ainda não visitado, aplica o relaxamento:
         - $d[z] = \min(d[z], d[p] + Adj[p,z])$
         - se atualizado, $s[z] = p$.
  3. **Reconstrói o caminho:** quando $vis[y] = true$, reconstrói o caminho de y a x usando vetor s.

Árvore geradora de caminho mínimo (MST)

- **Conceitos fundamentais:**
  - **Conjunto desconectante:** conjunto de arestas que, se removidos de um grafo, o desconectam, gerando múltiplos componentes
  - **Corte de arestas:** é um conjunto desconectante que é minimal, ou seja, não contém nenhum subconjunto próprio que também desconecta o grafo.
  - **Arestas conectividade (λ):** corresponde a cardinalidade do menor corte de arestas necessário para desconectar um grafo. Se $\lambda = 1$, a aresta que desconecta o grafo é chamada de aresta de ponte.
  - **Conjunto separador:** um conjunto de vértices que, se removidos de um grafo, o desconectam.
  - **Corte de vértices:** um conjunto separador que é minimal.
  - **Vértices conectividade (β):** corresponde à cardinalidade do menor corte de vértices. Se $\beta = 1$, o vértice que desconecta o grafo é chamado de vértice de articulação.
- **Árvore geradora (Spanning Tree):**
  - Uma árvore é um grafo conexo e acíclico, onde cada aresta é uma ponte.
  - Um subgrafo gerador (spanning subgraph) de um grafo $G_1(V_1, E_1)$ é um subgrafo $G_2(V_2, E_2)$ tal que $V_1 = V_2$ (contém todos os vértices do grafo original).
  - Quando o subgrafo gerador é uma árvore, ele é chamado de árvore geradora.
  - qualquer vértice pode ser considerado a raiz de uma árvore geradora
- **Árvore geradora de custo mínimo (MST - Minimum Spanning Tree):**
  - Um grafo pode ter vários árvores geradoras.
  - A MST é a árvore geradora que apresenta a soma mínima (ou máxima, dependendo do problema) dos pesos de suas arestas. O peso total $W(T)$ é a soma dos pesos de todas as arestas em T.

↳ Algoritmos para MST

↳ os algoritmos mais populares para calcular MST são Kruskal e Prim.

↳ ambos são algoritmos gulosos (greedy), embora algoritmos nem sempre garantam soluções globalmente ótimas para todos os problemas, no caso da MST, eles provadamente produzem uma Árvore de peso mínimo.

↳ **Algoritmo genérico para MST:** começa com um conjunto vazio de arestas A e, a cada passo, adiciona uma aresta segura a A até que A forme uma árvore geradora.

↳ **Aresta segura:** uma aresta que pode ser adicionada a A sem violar o invariante de que A é sempre um subconjunto de uma árvore geradora mínima. Uma aresta leve é sempre uma aresta de peso mínimo que conecta a árvore parcialmente construída a um vértice ainda não incluído nela.

↳ Algoritmo de Prim:

↳ as arestas no conjunto A sempre formam uma única árvore.

↳ começa em um vértice raiz arbitrário r e cresce adicionando a aresta leve que conecta a árvore A a um vértice ainda não incluído.

↳ Durante a execução, os vértices não incluídos na árvore A são mantidos em uma fila de prioridade mínima, ordenada pelo peso mínimo da aresta que os conecta a A. Um atributo (ou vetor) pai de um vértice armazena o seu pai.

↳ Floresta Geradora de custo mínimo (MSF - Minimum spanning forest)

↳ É uma generalização da MST para grafos desconectados. Para cada componente conexo de um grafo desconexo, calcula-se sua MST, e a coleção resultante dessas MSTs forma uma floresta geradora mínima.

Grafos Eulerianos:

↳ **Definição:** um grafo é considerado Euleriano se possui um passeio fechado que contém todas as arestas do grafo sem repeti-las. Esse passeio é conhecido como trilha Euleriana fechada. Para que um grafo conexo não orientado seja Euleriano, é uma condição necessária e suficiente que todos os seus vértices possuam graus pares. Alternativamente, um grafo conexo é Euleriano sse o conjunto de duas arestas puder ser dividido em ciclos disjuntos, que não compartilham arestas.

↳ **Grafos Semieulerianos:** possui uma trilha não fechada, um grafo conexo é dito semieuleriano sse houver exatamente dois vértices com graus ímpares. Nesse caso a trilha inicia em um nodo de grau ímpar e encerra no outro.

↳ **Algoritmo de determinação de trilhas Eulerianas**

1. Verificar os graus, se todos forem pares é Euleriano, se tiver 2 ímpares é semieuleriano.
2. **Algoritmo de Fleury:** determina a trilha euleriana ou semi-Euleriana.
   - No caso semieuleriano, o vértice inicial é um dos vértices de grau ímpar.
   - A cada iteração, uma aresta é escolhida. Uma regra importante é que uma aresta que seja uma ponte no grafo induzido pelas arestas ainda não marcadas só deve ser escolhida se não houver outra opção.
   - O algoritmo continua até que todas as arestas tenham sido removidas do grafo e adicionadas à trilha construída

Grafos Hamiltonianos:

um grafo Hamiltoniano possui um ciclo que passa por todos os vértices sem repetições, chamado ciclo Hamiltoniano. Já um grafo semi-Hamiltoniano contém um caminho Hamiltoniano, que percorre todos os vértices uma única vez, mas não precisa ser um ciclo. Todo grafo Hamiltoniano também é semi-Hamiltoniano.

Diferente dos grafos Eulerianos, não há uma condição necessária e suficiente para identificar grafos Hamiltonianos. O problema de decidir se um grafo possui um ciclo ou caminho Hamiltoniano é NP-Completo, ou seja, não se conhece um algoritmo polinomial eficiente para resolvê-lo, e métodos exatos têm complexidade exponencial. Assim, são utilizadas heurísticas, que buscam boas soluções sem garantia de serem ótimas. Algumas propriedades e teoremas auxiliam na identificação:

↳ Um vértice de grau 1 impede ciclo Hamiltoniano.

↳ Se um vértice tem grau 2, suas arestas devem estar no ciclo.

↳ Grafos completos K_n (com $n \geq 3$) e bipartidos completos $K_{n,m}$ são Hamiltonianos.

↳ **Teorema de Ore:** se para todo par de vértices não adjacentes v e w, $deg(v) + deg(w) \geq n$, sendo $n = |V(G)|$, então o grafo possui um ciclo Hamiltoniano.

↳ **Teorema de Dirac:** se todo vértice v tem grau $deg(v) \geq n/2$, então o grafo possui um ciclo Hamiltoniano.

↳ Os teoremas confirmam que o grafo é Hamiltoniano, mas a sua não validade não exclui a possibilidade.

↳ **Fecho Hamiltoniano (Bondy-Chvátal):** o fecho Hamiltoniano de um grafo G, denotado por $\phi(G)$ é obtido adicionando arestas entre pares de vértices não adjacentes cujo grau combinado seja maior ou igual a n ($n = |V(G)|$), ou seja, todo par de vértices $v, w \mid deg(v) + deg(w) \geq n$, repetindo esse processo até que nenhuma nova aresta seja adicionada. Se $\phi(G)$ for um K_n (grafo completo), então G é Hamiltoniano. Além disso G é Hamiltoniano sse $\phi(G)$ for Hamiltoniano. O processo pode ser interrompido se, durante a sua construção, o grafo satisfizer as condições de Ore ou Dirac.

Branch and Bound:

é uma técnica geral utilizada para resolver problemas de otimização combinatória, especialmente aqueles que envolvem variáveis inteiras ou discretas, onde a enumeração exaustiva de todas as soluções possíveis é inviável computacionalmente. Seu objetivo é encontrar a solução ótima sem explorar todas as possibilidades, sendo eficaz para problemas NP-Difíceis. O método combina dois pilares:

- **Branching** (ramificação): divide o problema original em subproblemas menores, representados por nós em uma árvore de busca, onde as ramificações indicam decisões tomadas.
- **Bounding** (limitação): calcula um limite da função objetivo para cada subproblema. Em problemas de minimização, usa-se o limite inferior (Lower Bound - LB), que representa o menor valor possível da solução ótima para aquele subproblema.

Coloração de Vértices:

é uma atribuição de cores a vértices de um grafo $G(V,E)$ de forma que vértices adjacentes não tenham a mesma cor. Formalmente, é uma função $f: V \to C$, tal que, se $(v,w) \in E$, então $f(v) \neq f(w)$. Uma K-coloração usa K cores distintas. O objetivo é encontrar o menor número de cores necessário para uma coloração válida. O número cromático, $\chi(G)$, é o menor número de cores necessário para uma coloração válida. O número cromático, $\chi(G)$ é o menor K para o qual o grafo admite uma K-coloração. Determinar $\chi(G)$ é um problema NP-difícil para $n \geq 3$, exigindo o uso de heurísticas, como o algoritmo guloso, que colore cada vértice com a menor cor disponível, embora o resultado dependa da ordem dos vértices e pode não ser ótimo

↳ Casos e propriedades importantes:
- Vértice com laço não admite coloração;
- Grafo nulo (sem arestas): $\chi(G) = 1$;
- Grafo bipartido: $\chi(G) = 2$;
- Grafo planar sem laços: $\chi(G) \leq 4$ (teorema das quatro cores)

↳ Conceitos úteis:
- Conjunto independente: vértices não adjacentes; seu tamanho máximo é $\alpha(G)$. Vértices em um conjunto independente podem ter a mesma cor.
- Clique: subgrafo completo; seu tamanho máximo é $\omega(G)$; $\chi(G) \geq \omega(G)$.

↳ Limites para $\chi(G)$:
- Inferior: $\omega(G) \leq \chi(G)$, pois vértices de uma clique precisam de cores distintas.
- Superior: $\chi(G) \leq \Delta(G) + 1$, onde $\Delta(G)$ é o grau máximo do grafo.
- Assim: $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$.

↳ Polinômio Cromático $P_G(K)$: conta quantas colorações válidas existem com K cores.
- Se $K < \chi(G)$, então $P_G(K) = 0$
- Se $K \geq \chi(G)$, então $P_G(K) > 0$
- $\chi(G)$ é o menor K tal que $P_G(K) > 0$.

↳ Exemplos:
- Árvore com n vértices: $P_G(K) = K(K-1)^{(n-1)}$
- Grafo completo K_n: $P_G(K) = K(K-1)(K-2)\ldots(K-(n-1))$
- Ciclo C_n: $P_G(K) = (K-1)^n + (-1)^n(K-1)$

↳ Para grafos gerais: $P_G(K) = P_{G-e}(K) - P_{G/e}(K)$ onde
- $G-e$: grafo sem a aresta e.
- G/e: grafo com a aresta e contraída.

Esse processo é repetido até que o polinômio possa ser conhecido por fórmulas básicas.

Fluxo em redes: é um conceito de teoria dos grafos usado para modelar o transporte ou tráfego em digrafos ponderados, com o objetivo de determinar o fluxo máximo entre dois pontos. Aplica-se a sistemas como redes de transportes, comunicação e energia elétrica.

↳ Representação: A rede é um digrafo com capacidades (pesos) nas arestas. Dois vértices especiais têm:
- Fonte (v): é o vértice onde o fluxo se origina. Nenhuma aresta aponta para ele, ou seja, não recebe fluxo. Portanto, seu grau de entrada é zero: $indeg(v) = 0$.
- Sumidouro (w): é o vértice onde o fluxo termina, representando o destino final. Nenhuma aresta sai dele, ou seja, não envia fluxo para outros vértices. Portanto seu grau de saída é zero: $outdeg(w) = 0$.

↳ Conceitos centrais:
- $indeg(x)$: número de arestas que entram em um vértice x (grau de entrada).
- $outdeg(x)$: número de arestas que saem de um vértice x (grau de saída).
- Cada aresta (u,z) possui:
  - ↳ Capacidade $c(e)$ e fluxo $f(e)$
  - ↳ Fluxo legal: ocorre quando $0 \leq f(e) \leq c(e)$
  - ↳ Aresta saturada: quando $f(e) = c(e)$
- Para todo vértice exceto a fonte e o sumidouro, vale a conservação de fluxo: o fluxo total que entra é igual ao que sai.
- O fluxo total da rede é a soma dos fluxos que saem da fonte (ou entram no sumidouro), chamado de valor do fluxo.

↳ Tipos de fluxo
- Fluxo maximal: quando não existe nenhum caminho da fonte ao sumidouro no qual seja possível aumentar o fluxo sem ultrapassar a capacidade de alguma aresta.
- Fluxo máximo: maior valor de fluxo legal possível da fonte ao sumidouro. Todo fluxo máximo é necessariamente maximal, mas nem todo fluxo maximal é máximo pois pode existir outra distribuição de fluxo com valor maior.

↳ Algoritmo de Ford-Fulkerson:
- Encontra iterativamente caminhos de aumento da fonte ao sumidouro, com capacidade disponível.
- Aumenta o fluxo nesses caminhos até não restar mais nenhum caminho possível.
- Utiliza a rede residual, que representa a capacidade restante e reversa das arestas.

Ordenação topológica: A ordenação topológica organiza tarefas ou eventos em uma sequência linear, garantindo que cada tarefa ocorra após a conclusão de todas as tarefas das quais depende. É usada para definir a ordem correta de execução em situações com dependências, como montagem de projetos, compilação de código ou fluxos de trabalho.

↳ Definição: é uma permutação dos vértices de um grafo direcionado, onde para cada aresta $\{v_i, v_j\}$, o vértice v_i aparece antes de v_j na sequência ordenada.

↳ Grafo direcionado acíclico (DAG): apenas grafos direcionados sem ciclos podem ser ordenados topologicamente. A existência de um ciclo impede uma ordenação válida.

↳ Grau de entrada e saída:
- $indeg(x)$: número de arestas que entram no vértice x.
  - ↳ se $indeg(x) = 0$, o vértice não depende de nenhum outro.
- $outdeg(x)$: número de arestas que saem do vértice x.
  - ↳ se $outdeg(x) = 0$, o vértice não é pré-requisito de nenhuma outra tarefa.

↳ Ideia básica do algoritmo
1. Identificar um vértice com $indeg = 0$.
2. Adicionar esse vértice à ordenação.
3. Remover ou simular a remoção do vértice, atualizando o indeg dos seus vizinhos.
4. Repetir até processar todos os vértices

⚠ Se ao final ainda restarem vértices com $indeg > 0$, o grafo possui ciclo e não admite ordenação topológica.

4 Implementação.

- Usa uma fila para armazenar a ordenação.
- Utiliza:
  - Estrutura auxiliar para armazenar indeg e outdeg
  - Matriz de adjacência ou lista de adjacência.
- Em cada passo:
  1. Escolher o vértice com indeg = 0 e maior outdeg entre os candidatos
  2. Inserir na fila e atualizar o indeg dos vizinhos
  3. Repetir a escolha com o novo vértice ativo.
  - se não for possível encontrar mais vértices com indeg = 0, o grafo contém ciclos.