Resumo Prova 3 - Aeds III

- **Algoritmo de Dijkstra - Bellman-ford**
  - **Estrutura de Dados - Aula 28 - Grafos - Algoritmo de Dijkstra**
  - **Algoritmo de Dijkstra - Algoritmos em Grafos**

Em grafos não ponderados podemos encontrar o caminho entre dois vértices utilizando os algoritmos de **largura** e **profundidade**, mas isso muda em grafos ponderados(com peso).

Algoritmos de caminhos mínimos, são algoritmos para encontrar o menor caminho em grafos ponderados.

Algoritmo de Dijkstra
Dijkstra é considerado um algoritmo **guloso** ou **programação dinâmica.**

Utiliza quatro vetores auxiliares:

$$\begin{array}{lcl} \pi[u] & \rightarrow & \text{pai do vértice } u \\ d[u] & \rightarrow & \text{distância da raiz até } u \\ Q & \rightarrow & \text{vértices com o caminho mínimo provisório} \\ S & \rightarrow & \text{vértices com o caminho mínimo garantido} \end{array}$$

Esses vetores são utilizados nas funções auxiliares que é utilizada na função Dijkstra

Pai do vértice u: suponha que temos três vértices A,B e C. A até o vértice C passa pelo vértice B, isso significa que B é o "pai" de C nesse contexto. Em termos simples, o "pai" de um vértice é o vértice pelo qual você chegou até ele durante a execução do algoritmo.

Distância da raiz até u: Dizemos que d[ ] é uma estimativa de caminho mais curto,inicialmente em cada índice do vetor com infinito sendo a quantidade de indice a quantidade Máxima de vértices do grafo.

vértices com o caminho mínimo provisório:
vértices com o caminho mínimo garantido:

Utiliza duas funções auxiliares:

```
INICIALIZA(G = (V, A), s)
    para cada v ∈ V
        d[v] = ∞
        π[v] = NULL
    fim para
    d[s] = 0
fim
```

```
RELAXA(u, v, w)
    se d[v] > (d[u] + w(u, v)) então
        d[v] ← d[u] + w(u, v)
        π[v] = u
    fim se
fim
```

Código Dijkstra utilizando as duas funções acima auxiliares:

```
DIJSKSTRA(G = (V, A), w, s)
    INICIALIZA(G, s)
    S ← { }
    Q ← V
    Enquanto | Q |≠ 0
        u ← extrair Minino(Q)
        S ← S ⋃ {u}
        para cada v ∈ Adj[u]
            relaxa(u, v, w)
        fim para
    fim enquanto
fim
```

https://github.com/Abner-Guimaraes/Algoritmo_em_Grafos

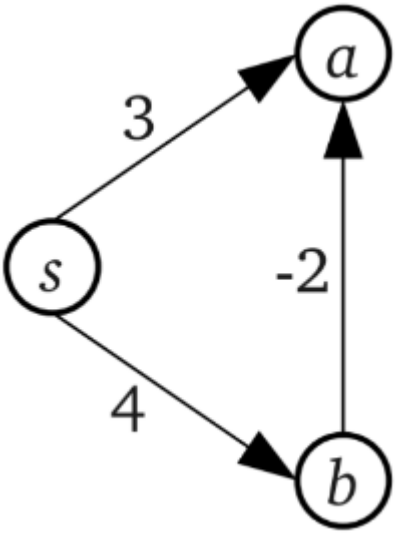
INICIALIZA:inicializa os vetores auxiliares para ser utilizado no Dijkstra.
representação de **Infinito**
https://petbcc.ufscar.br/limits/
Para representar infinito e colocar no vetor podemos utilizar a biblioteca **limits**.**h** que tem a função **INT_MAX**,valor máximo que um número inteiro pode assumir(pode ser utilizado a biblioteca **math**.**h**).

RELAXA: Atualiza a menor distância conhecida até um vértice, se uma nova rota oferecer um caminho mais curto até ele.

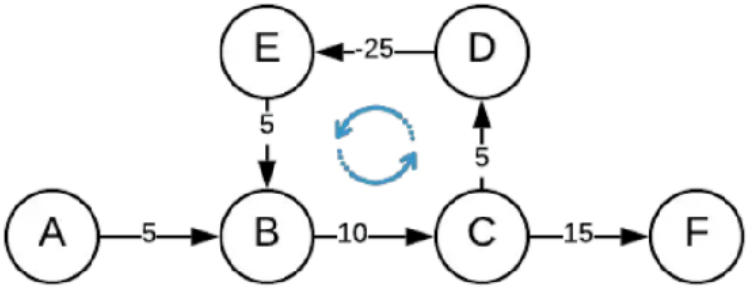
Limitações de utilizar esse código:

Grafos com pesos negativos

O algoritmo de Dijkstra não pode ser usado em grafos com pesos negativos porque ele assume que todas as arestas têm pesos não negativos. Se houver arestas com pesos negativos, o algoritmo pode ficar preso em um loop infinito tentando reduzir a distância devido à presença de ciclos de peso negativo. Para lidar com grafos com pesos negativos, é necessário usar algoritmos como o de **Bellman-Ford**, que podem detectar ciclos de peso negativo e lidar com essa situação.
Para saber o custo mínimo em grafos com pesos não negativos a melhor escolha é utilizar **Dijkstra**, mas em pesos negativos temos o algoritmo de **Bellman-Ford.**



O exemplo acima e abaixo ajuda na compreensão, utilizando o algoritmo de Dijkstra em um grafo com esse ciclo, vai acarretar em um loop infinito;

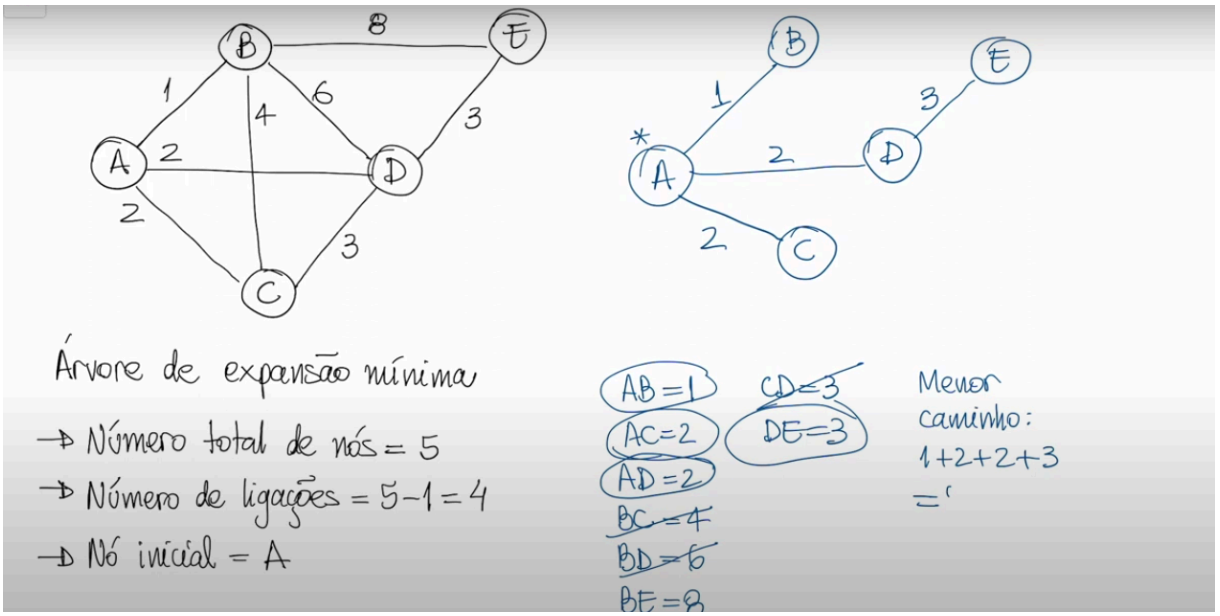


- **Algoritmo de Kruskal e algoritmo de Prim**

Árvore de expansão mínima: um exemplo

Resolvendo um outro exemplo de árvore de expansão mínima

Os algoritmos de **árvore geradora mínima(AGM)** tem como objetivo encontrar a subárvore com os mesmo vértices da árvore principal mas com as arestas com menos pesos e sem ter nenhum ciclo entre eles. No final faz a soma dos pesos das arestas. Isso é importante em muitas aplicações, como em redes de comunicação, onde se deseja conectar todos os pontos com a menor quantidade de fios ou com menor custo possível.



Para compreender melhor essa imagem recomendo assistir os vídeos do título(não seja preguiçoso/a o vídeo tem 2 minutos)

Observação:não estou explicando com detalhes os códigos pelo fato de ser dois códigos parecidos, a função deles é criar AGM de um grafo específico com arestas com peso. **e são algoritmos para grafos não direcionados.**

Prim

Algoritmo de Prim - Árvores Geradoras Mínimas - Algoritmos em Grafos

Kruskal

Kruskal - Algoritmos em Grafos - Árvores Geradoras Mínimas

nesses vídeos acima o professor mostra o pseudocódigo e faz o acompanhamento utilizando um desenho de um grafo

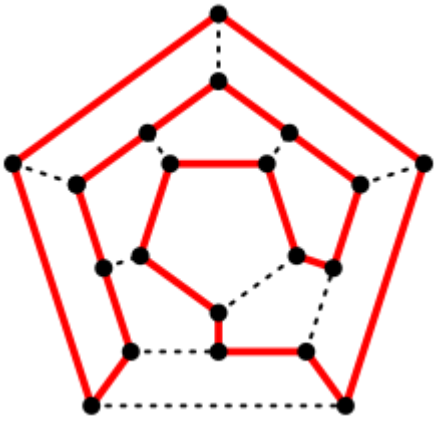
Diferenças entre os códigos Prim e Kruskal

Identificação de ciclos

No algoritmo de Prim, ao adicionar uma aresta à árvore geradora mínima, estamos garantindo implicitamente que não haverá ciclos na árvore. Isso ocorre porque, em cada iteração do algoritmo, selecionamos apenas a aresta de menor peso que conecta um vértice já na árvore a um vértice fora dela. Como começamos com um único vértice e expandimos a árvore gradualmente, sempre adicionando a aresta mais leve que conecta a árvore atual a um vértice fora dela, nunca criamos ciclos. Em contraste, **no algoritmo de Kruskal**, a verificação de ciclos é feita explicitamente. Isso ocorre porque o algoritmo seleciona arestas com base em seus pesos e verifica se a adição de uma nova aresta criará um ciclo na árvore parcial construída até o momento. Então, para resumir, enquanto o algoritmo de Prim garante que não haverá ciclos ao construir a árvore geradora mínima de maneira incremental, o algoritmo de Kruskal verifica ativamente a presença de ciclos antes de adicionar cada nova aresta.

- **Ciclos eurialinos e hamiltonianos - Caixeiro viajante e viajante chinês**

Ciclo hamiltoniano
um ciclo que passa por todos os vértices sem repetir nenhum e que volta para o vértice de início



nesse exemplo é possível começar em qualquer vértice, percorrer todos os vértices e chegar no vértice que começou sem repetir nenhum

temos algumas representações para um grafo hamiltoniano ou semi-hamiltoniano
se diz que o **grafo é hamiltoniano** quando tem um **ciclo hamiltoniano** nele
se diz que o **grafo é semi-hamiltoniano** quando tem um **caminho hamiltoniano** nele

caminho hamiltoniano: percorre todos os vértices **uma única vez** mas não necessariamente precisa terminar no vértice que começou
Como descobrir se o Grafo é hamiltoniano
NP-Completo
Verificar o Teorema de Dirac: O Teorema de Dirac é uma condição suficiente para um grafo ser hamiltoniano. Seja G um grafo simples com n vértices onde n ≥ 3. Se o grau de cada vértice em G é pelo menos n/2, então G é hamiltoniano.

Verificar o Teorema de Ore: O Teorema de Ore fornece uma condição suficiente para que um grafo seja hamiltoniano. Seja G um grafo simples com n vértices onde n ≥ 3. Se para cada par de vértices não adjacentes u e v, a soma dos graus de u e v é maior ou igual a n, então G é hamiltoniano. Você pode aplicar essa condição para verificar se o grafo em questão atende a ela.

Teorema de Bondy e Chvátal: Seja G um grafo simples com n vértices onde n ≥ 3. Se, para cada par não adjacente de vértices u e v em G, o grau de u + o grau de v é maior ou igual a n, então G é hamiltoniano.

Ciclo Euleriano
um ciclo que passa por todas as arestas sem repetir nenhuma vez e volta para o vertice onde começou

temos algumas representações para um grafo **Euleriano** ou semi-**Euleriano**
se diz que o **grafo é Euleriano** quando tem um **ciclo Euleriano** nele
se diz que o **grafo é semi-Euleriano** quando tem um **caminho Euleriano** nele

Caminho Euleriano: percorre todas as arestas **uma única vez** mas não necessariamente precisa terminar no vértice que começou

Como descobrir se o Grafo é Euleriano

Grafo euleriano (Teorema de Euler, 1952)

Um grafo G é euleriano se e somente se todos seus vértices possuem grau par

Grafo não euleriano

Um grafo G não é euleriano se e somente se existem dois ou mais vértices de grau ímpar

Grafo semi-euleriano

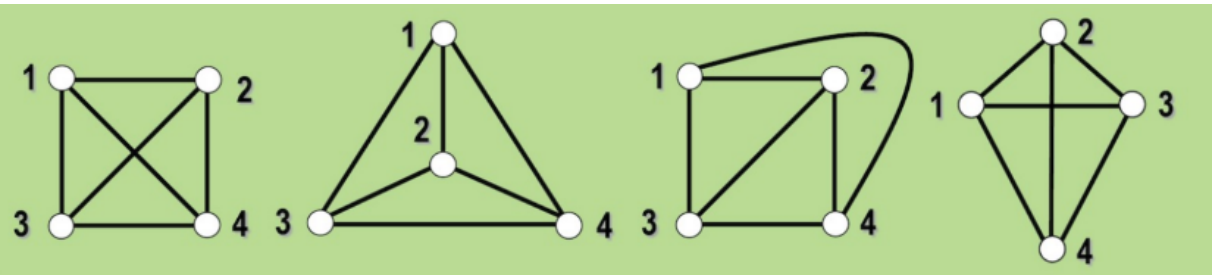
Um grafo G é semi-euleriano se e somente se existem exatamente dois vértices de grau ímpar

- **Isomorfismo**

ED190 Grafos Isomorfos
Teoria dos Grafos - Isomorfismo

é a comparação de grafos de modo a afirmar se a estrutura de um grafo(G) pode ser transformada para a outra estrutura do Grafo(E)

Nesse exemplo abaixo podemos transformar a primeira estrutura do primeiro Grafo(A) para os outros grafos e comprovar que são isomorfos



Como identificar se dois grafos são isomorfos ou mais
Quando não é isomorfismo ?

- **um grafo tem mais vértices que o outro**
- **um grafo tem mais arestas que o outro**
- **um grafo tem arestas paralelas e o outro não**
- **um grafo tem um laço e o outro não**
- **um grafo tem um vértice de grau k e o outro não**
- **um grafo é conexo e o outro não**
- **um grafo tem um ciclo e o outro não**

Grau: é a quantidade de conexões que o vértice tem, ou melhor, quantas arestas estão conectadas nesse vértice

- **Componentes conexões**

nesse tópico vamos identificar grafos e suas conexões, vamos determinar se é um grafo conexo ou não

Existem várias denominações para determinar as conexões de um grafo

Grafo desconexo:só pelo nome já dar pra saber, é um grafo onde tem um vértice fora da sua estrutura, ou seja, ele é desconexo do grafo
Grafo não orientado conexo: é um grafo onde não tem direção, é um grafo não orientado, e ele é conexo pelo fato de vc conseguir ir para qualquer vértice começando por um vértice qualquer do grafo

Mas e com grafos orientados?
como podemos denominar e analisar sua conectividade

Em grafos orientados temos três categorias de conexo:
Estrutura de Dados - Aula 23 - Grafos - Conceitos básicos 01:25
http://www.decom.ufop.br/marco/site_media/uploads/bcc204/03_aula_03.pdf

1. **Simplesmente conexo (s conexo)**

Todo grafo conexo é simplesmente conexo

2. **Semi-fortemente conexo(sf conexo)**

Quando tem u para v mas nem sempre de v para u

3. **Fortemente conexo (f conexo)**

u para v e v para u

- **Coloração em grafos**

Regras de coloração:
Um número par de vértices requer no mínimo 2 cores.
Um número ímpar de vértices requer no mínimo 3 cores.
essa regra também funciona em arestas e em vértices e aresta junto

Complexidade:
Computar um número cromático de um grafo é NP-difícil

- **Fluxo em redes**