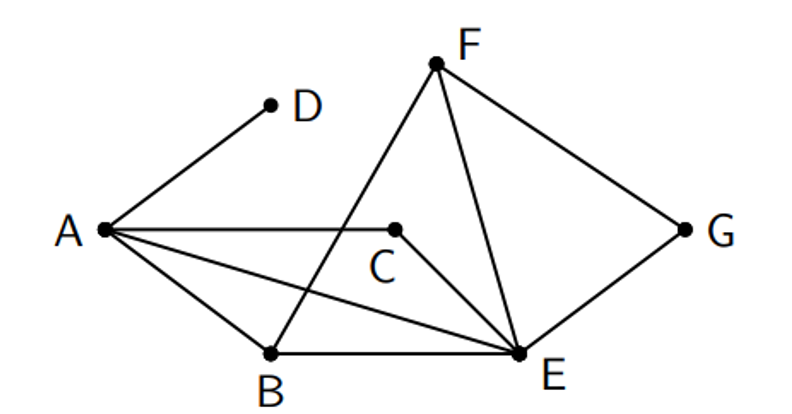
**Resumo – Teoria dos Grafos**

Iniciou-se na cidade de Königsberg em 1736, pelo matemático suíço Leonhard Euler.

**Grafos conexos**: aquele em que todos os vértices possuem pelo menos uma aresta.

**Grafos não conexos:** aquele em que alguma parte do grafo encontra-se separada.



Dada esse grafo,

**Grau de um vértice – d(v):** é quantas arestas são conectadas em um determinado vértice.

Ex. d(A) = 4

**Caminho:** é o percurso de um vértice até outro.

Ex. Caminho de A à G = A -> C -> E -> F -> G

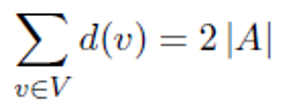
**Circuito:** é um caminho fechado.

Ex. A -> C -> E -> A

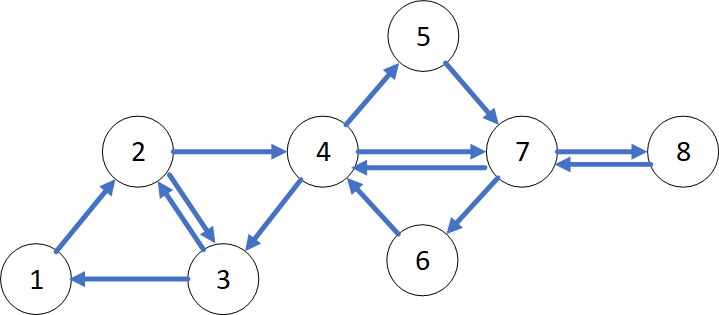
Ciclo de tamanho 3. | C3

**Grafo Simples:** é aquele que não possui loops e/ou arestas paralelas.

Em um grafo simples, a soma dos graus de todos os seus vértices é igualao dobro do número de arestas.



**Dígrafo / Grafo orientado:** o grafo em que as arestas são orientadas

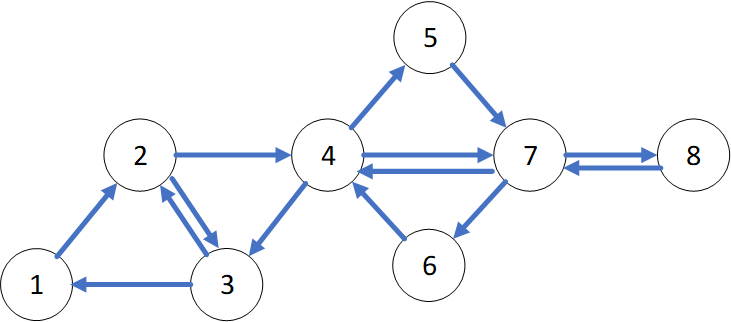


Uma **aresta orientada** (vi – vj) é uma

**aresta emergente** do vi e uma

**aresta incidente** no vértice vj

**Dígrafo com circuito euleriano:** aquele em que todos os vértices semigrau emergente = semigrau incidente.



**Caminho orientado:** é uma sequência de vértices conectados por arestas em um grafo orientado.

**Caminhos simples orientado:** é um caminho orientado, mas sem revisitar vértices.

**Caminhos elementares orientados:** é como um caminho simples orientado, mas nenhum arco se repete.

**Circuitos orientados:** é como um caminho orientado, mas você retorna ao vértice de partida.

**Circuitos simples orientados:** é como um circuito simples orientado, mas sem revisitar vértices, exceto o inicial e final.

**Ciclos orientados:** é um circuito simples orientado, mas nenhum arco se repete.

**Vértice Fonte:** aquele com grau de entrada nulo. Apenas com aresta(s) saíndo.

**Vértice Sumidouro:** possui grau de saída nulo. Apenas com aresta(s) chegando.

**Dígrafo Acíclico:** quando possui um vértice fonte e um vértice sumidouro.

**Dígrafo Fortemente Conexo:** onde todos os vértices possuem um caminho para os demais.

**Dígrafo Fracamente Conexo:** todos os vértices possuem um caminho orientado de Vi para Vj, mas não de Vj para Vi.

**Dígrafo Pesado:** aquele que possui pesos em suas arestas

**Fecho Transitivo:** expande a conectividade de um grafo revelando caminhos indiretos entre vértices.

**Representação matemática de um Grafo:**

Conjunto de vértices V = {1, 2, 3, 4}

Conjunto de arestas E = {{1,2}, {1,3}, {1,4}, {2,4}, {3,4}}

**Representação Computacional:**

Vetor de vértices V = {1, 2, 3, 4}

Int vértices [4];

Lista de arestas E = {{1,2}, {1,3}, {1,4}, {2,4}, {3,4}}

#define qtdArestas 5

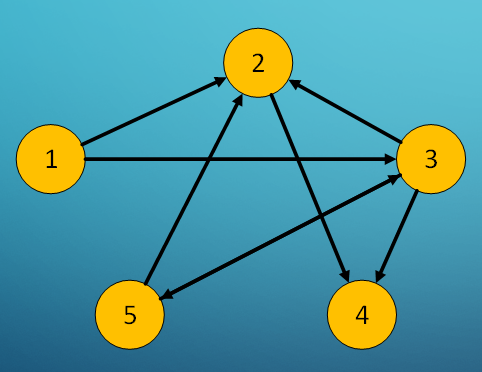
Struct Aresta{

Int vEmergente;

Int vIncidente;

}aresta[qtdAresta];

**Representação Matriz de Adjacência:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |

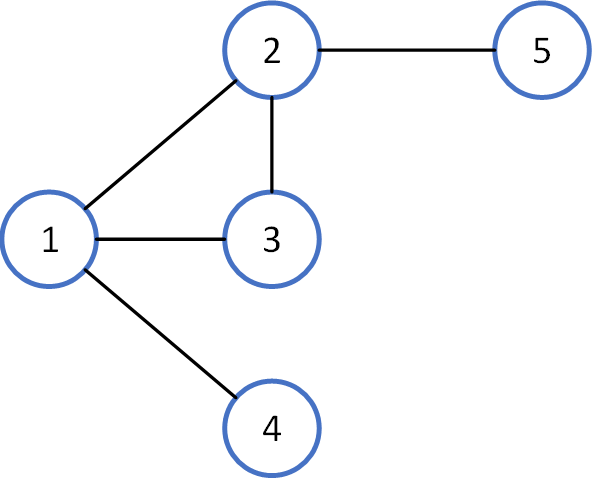
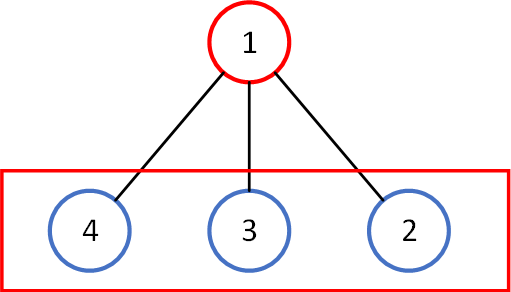
**Lista –** melhor para grafos esparsos

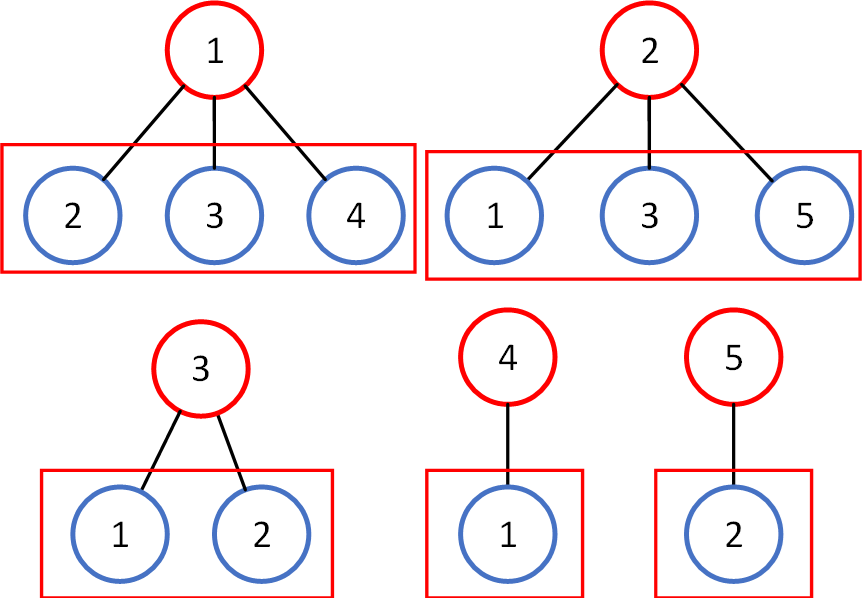
**Matriz –** melhor para grafos muito conexos

**Conexidade:** quando os vértices são conexos.

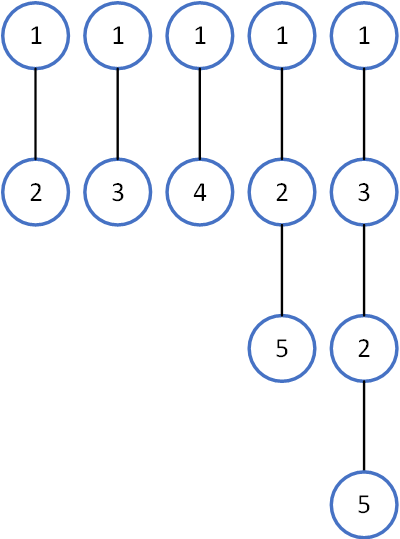
**Conectividade:** se há um caminho entre V origem até V destino.

**Busca em largura | Busca Horizontal | Breadth first**

** **

****

**Busca em profundidade | Depth first**

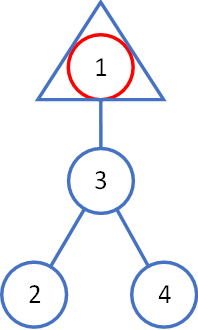


**Árvore:** é um grafo conexo que não possui circuitos.

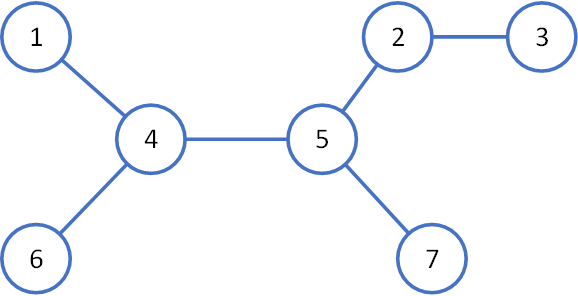
**Árvore orientada:** é um dígrafo conexo que não possui circuitos ou semi-circuitos.

**Teorema:** um grafo é uma árvore se, e somente se, existir um e apenas um caminho entre cada par de vértices.

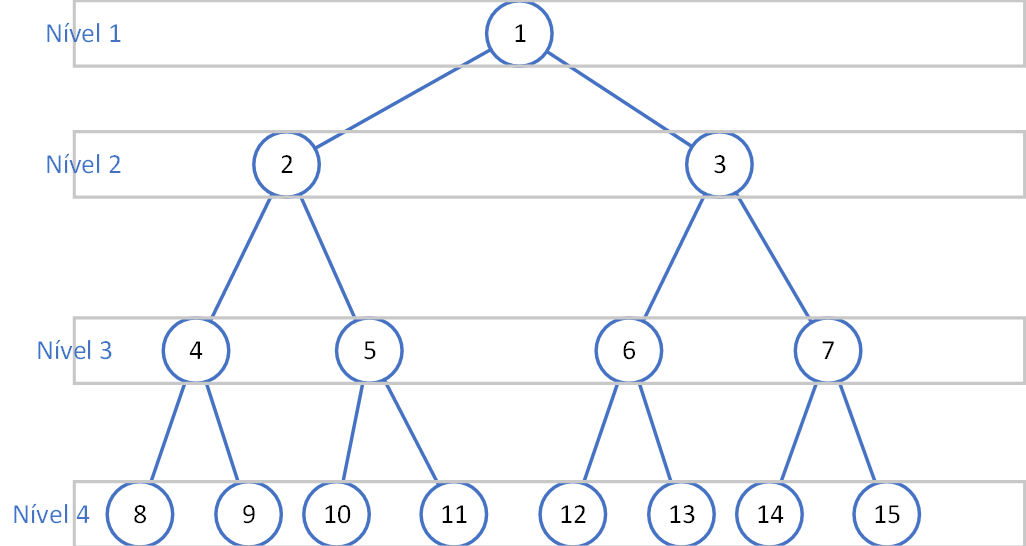
**Árvore enraizada:** aquela onde pode-se distinguir um vértice raiz.

 <- Exemplo

**Árvore não enraizada é chamada de árvore livre.**



Se enraizada, podemos identificar os **níveis** da árvore.



**Árvore Geradora T**

Procedimento para obter a árvore geradora:

1. Se G não possui circuitos, G é sua própria árvore geradora.
2. Se G possui circuitos, retire uma aresta do circuito. O subgrafo resultante é conexo.
3. Se existirem mais circuitos, repita até retirar uma aresta do último circuito do grafo.
4. O subgrafo resultante é conexo, sem circuitos e possui todos os vértices de G. Portanto é uma árvore geradora de G.

**Teorema:** Todo grafo conexo contém pelo menos uma árvore geradora.

**Obs:** o número de árvores geradoras de um grafo pode ser muito grande. O grafo de Petersen possui 2000 árvores geradoras.

**Grafos Bipartidos:** uma **bipartição** ou **bicoloração**, de um grafo não dirigido é uma coloração válida do grafo com duas cores.

O intuíto é que as duas pontas de cada aresta sempre tenham cores diferentes.

Um grafo não-dirigido é bipartido, ou bicolorido, se admite uma bicoloração.

**Algoritmo de Bipartição**

O algoritmo recebe um vértice incolor V e uma cor C e decide – sim ou não – se existe uma bicoloração (completa) de G que estrende a bicoloração incompleta e atribui cor C a V.

A(v, c)

Atribua cor C a V.

Se algum vizinho colorido de v tem cor c

Então diga “não atribui a cor c

Senão

Se A(w, 1-c) diz não para algum vizinho incolor w de v

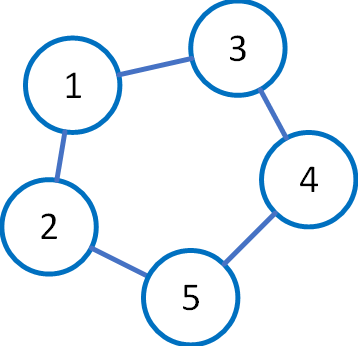
Então diga “não atribui a cor c”

Senão diga “sim atribui a cor c”

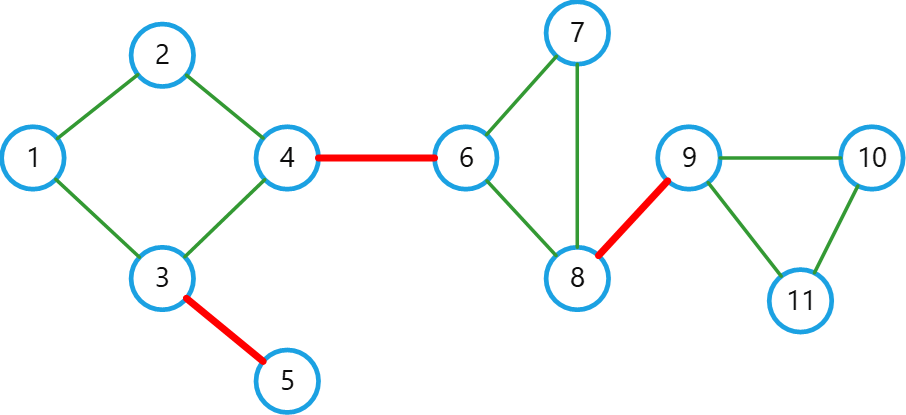
**Circuitos ímpares:** aquele que seu comprimento for um número ímpar.

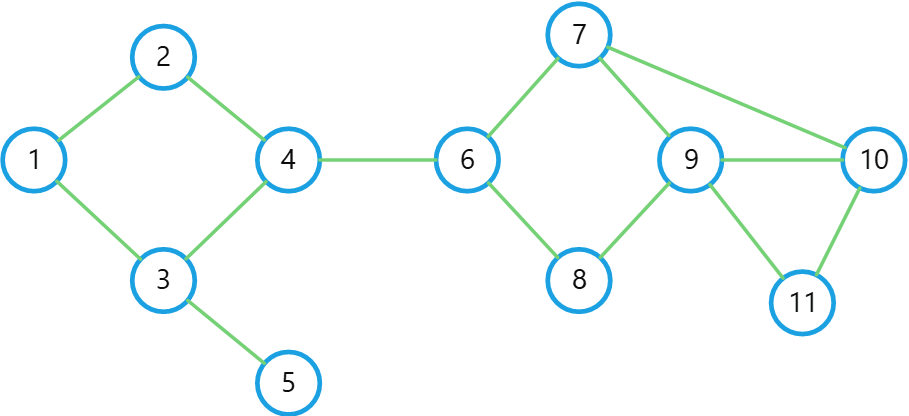
Diz-se comprimento o número de arestas

Se um grafo tem um circuito ímpar, então é obvio que não admite bipatição.



**Pontes em grafos:** são arestas que, caso sejam a única a atravessarem um corte do grafo, se retiradas transformam o grafo em desconexo.



**Articulação:** tem um conceito parecido com o da ponte, mas trata-se de um vértice que se caso removido transforma o grafo em desconexo. 

**Grafo Biconexo | 2-conexo:** se é conexo e não tem articulações

**Heurística Gulosa | Algoritmo de coloração sequencial:** produz uma coloração válida de qualquer grafo.

escolha um vértice incolor v

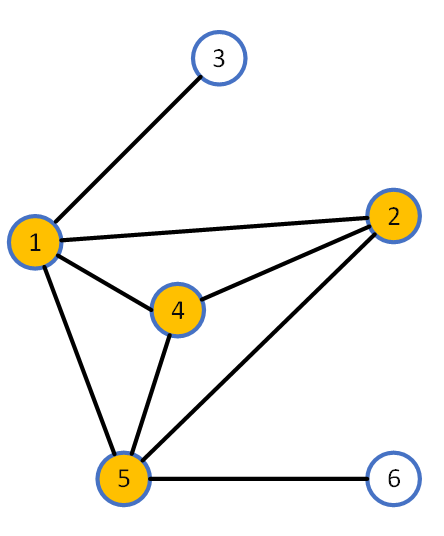
se alguma cor c não é usada por nenhum vizinho de v

então atribua cor c a v

senão atribua cor k a v e some 1 a k

O algoritmo não retorna para trocar a cor. Comportamento guloso (greedy).

**Clique**: em um grafo não-dirigido, é um conjunto de vértices adjacentes entre si.



Clique de tamanho 4.

**Problema da clique máxima:** encontrar uma clique de tamanho máximo num grafo não-dirigido.

**Obs:** se um grafo não-dirigido tem uma clique de tamanho q, então qualquer coloração válida precisa de pelo menos q cores.

**Árvores Geradora Mínima - AGM:**

Ideia de problema:

**Modelagem:**

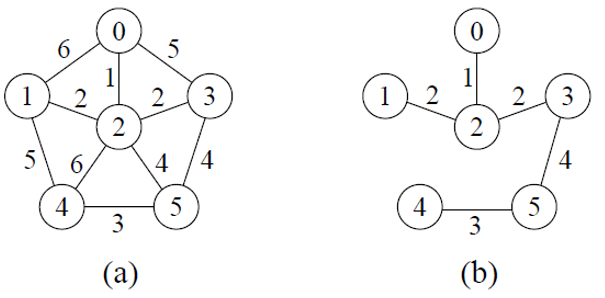
- G = (V,A): grafo conectado, não direcionado

- V: conjunto de cidades

- A: conjunto de possíveis conexões

- p(u,v): peso da aresta (u,v) pertencente a A, custo total de cabo para conectar u a v.

**Solução:** encontrar um subconjunto T contido em A, acíclico, que conecta todos os vértices de G e cujo peso total seja minimizado.



Uma AGM de peso 12.

**Generico AGM**

- Uma estratégia gulosa permite obter a AGM adicionando uma aresta de cada vez.

- Invariante: antes de cada iteração, S é um subconjunto de uma AGM.

A cada passo, adicionamos a S uma aresta (u,v) que não viola a invariante. (u,v) é chamada de uma **aresta segura**.

**AGM – Corte**

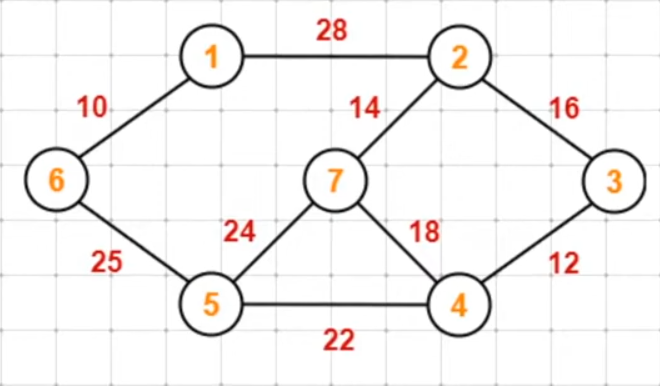
Um corte de um grafo não direcionado é uma partição de V.

Uma aresta pertencente a A **cruza** o corte se um de seus vértices pertence a V e o outro vértice pertence a V – V.

**Algoritmo de PRIM – AGM**

É derivado do algoritmo genérico. Serve para encontrar a AGM de um grafo.

Iniciamos em qualquer vértice.



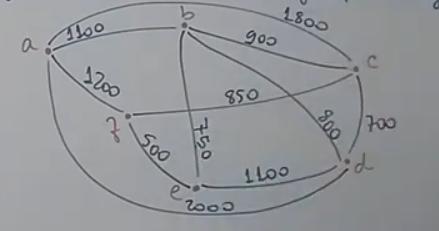
Por ex. o 7.

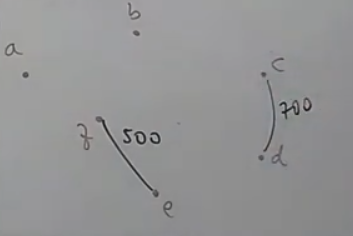
O objetivo é ir escolhendo o vizinho de peso mais leve, até que finalizem todos os vértices. Se caso em algum momento um vizinho mais leve for o vértice inicial, deve-se desconsiderá-lo escolhendo o segundo mais leve.

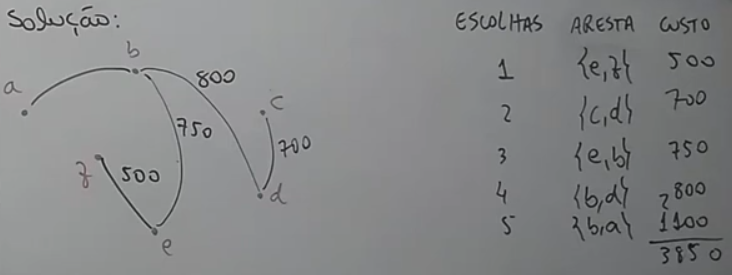
No exemplo da figura, partindo de 7 teríamos: 7-2-3-4-5-6-1

**Algoritmo de KRUSKAL – AGM**

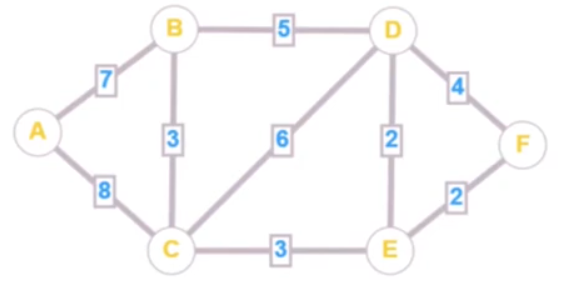
É derivado do algoritmo genérico. Serve para encontrar a AGM de um grafo.



De início, vou colocando as arestas de menor peso à vulso em um novo grafo.  


O cuidado é que a próxima aresta escolhida não forme circuito. Fazer isso até ter a AGM.

**Algoritmo de BORUVKA – AGM**



Dado o grafo,

Primeiro separam-se as relações:



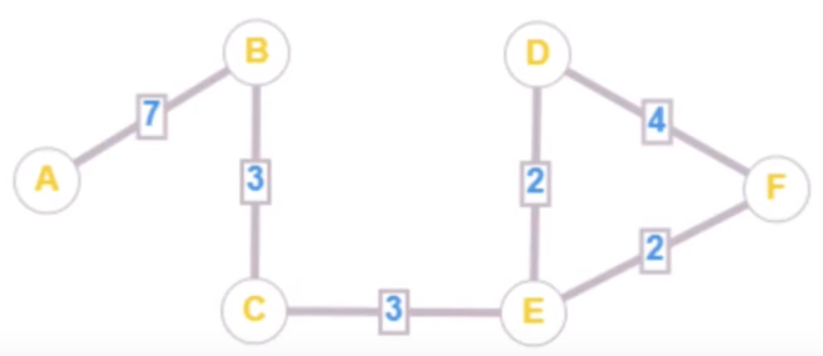
É escolhida a aresta mais leve de A, de B, de C… até a de menor valor de F



Vou ignorando as que já foram escolhidas



Finalizando dessa forma:



O último passo é retirar qualquer circuito, agrupando as arestas por ordem crescente e adicionando as arestas de menor valor sem que se forme circuito.



Terminando dessa forma.

**Caminhos Mínimos em Grafos**

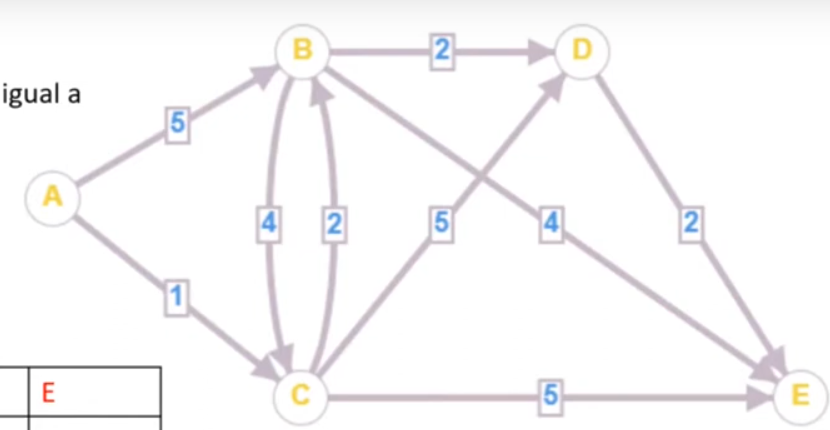
**Algoritmo de Bellman-Ford:** algoritmo de busca de caminho minimo em um dígrafo ponderado, ou seja, cujas arestas têm peso, **inclusive negativo**.

**Algoritmo de Dijkstra:** resolve o mesmo problema, num tempo menor, porém exige que todas as arestas tenham pesos positivos.

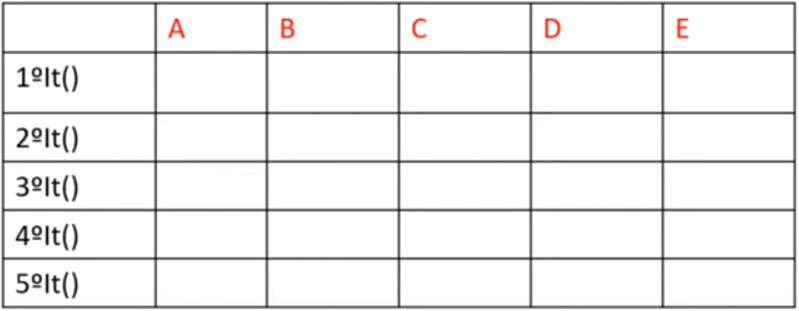
**Algoritmo de DIJKSTRA**

Determina, num grafo pesado orientado ou não, qual o caminho mais curto entre uma determinada **origem** e um **destino** específico.

Só pode ser aplicado em arestas positivas.

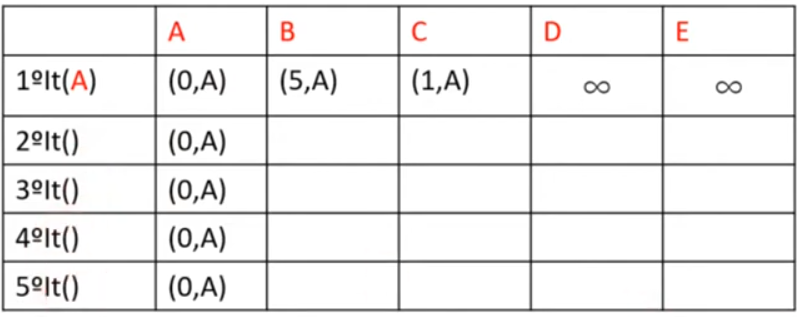


Sendo este o dígrafo,



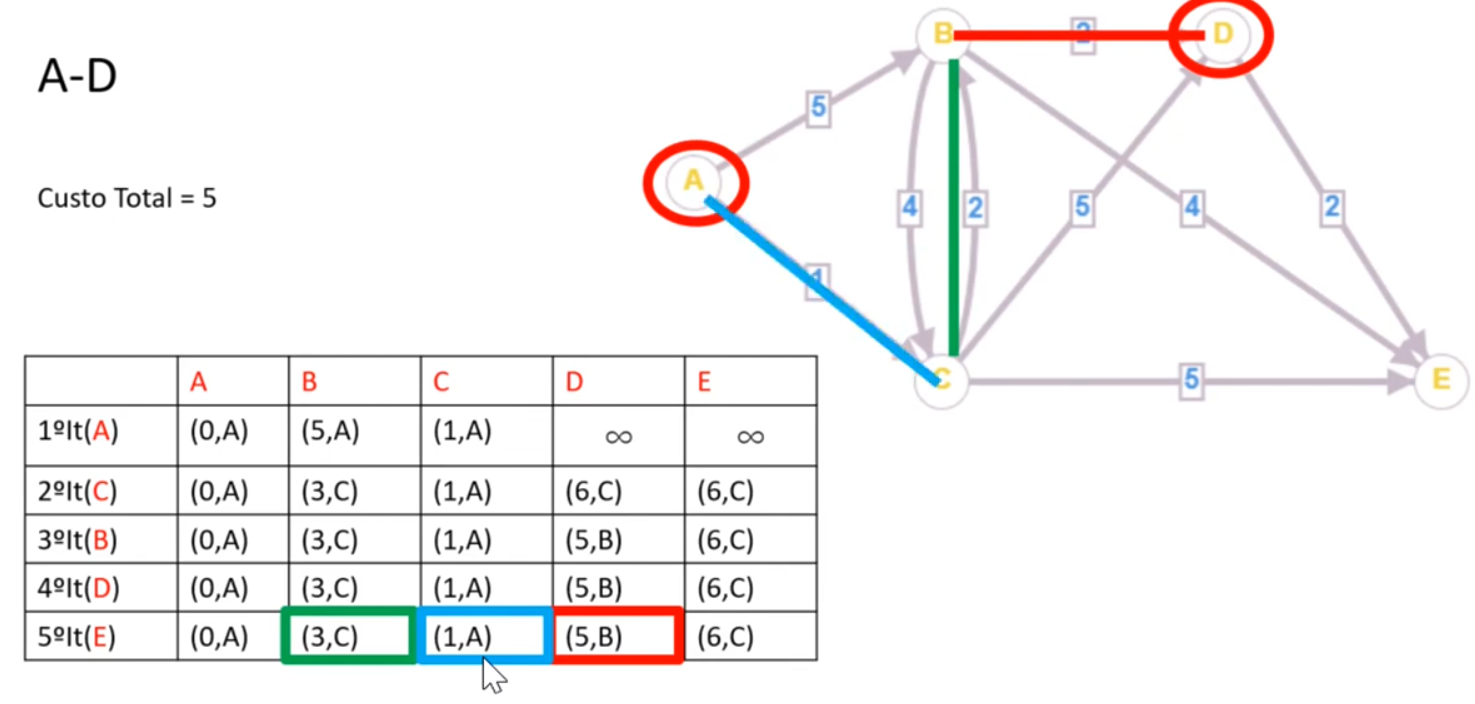
Sendo n (nº de vértices) o nº de iterações será igual a n.

n = 5



Vou buscando se há uma ligação do vértice da linha até o vértice da coluna.

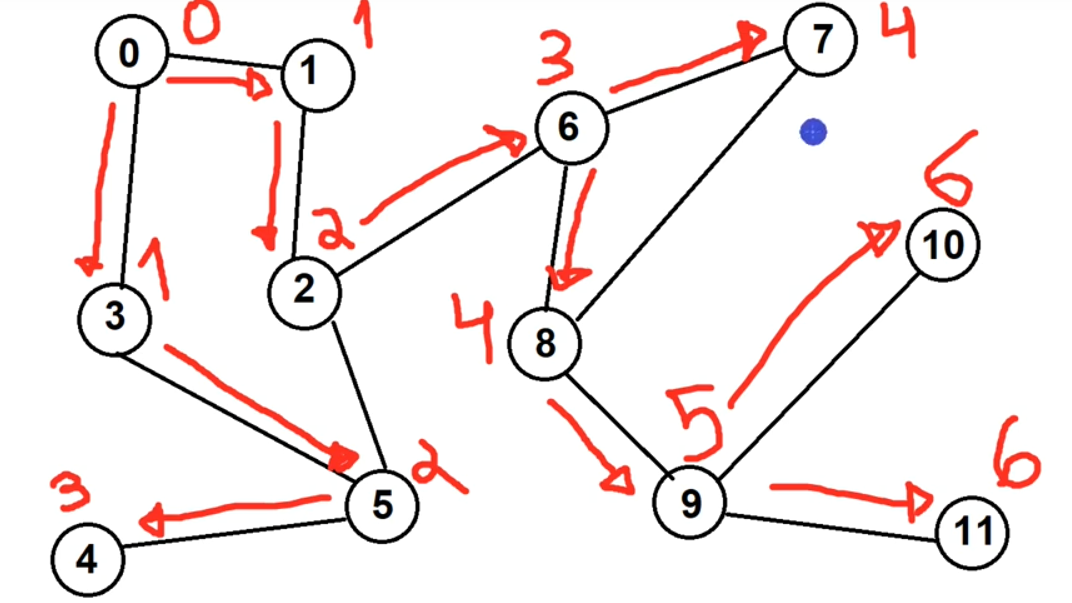
A coluna do A fica imutável e o vértice A não pode ser mais considerado. Dou seguimento.



É possível entar saber o caminho mais rápido de A até qualquer vértice.

**Algoritmo de Busca em Largura – INUNDAÇÃO OU BRODCASTING**

Encontra a menor distância com o menor número de arestas.



**Problema do Fluxo Máximo**

É uma poderosa ferramenta de modelagem, capaz de representar uma grande variedade de outros problemas.

**Corte Máximo**

Tem a função de prever em qual parte do grafo possui mais recursos alocados.

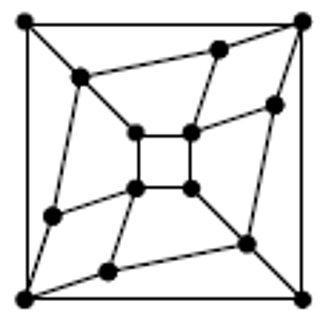
A ideia básica é deslocar os recursos de uma região altamente atendida para uma região que possua uma defasagem de conexões ou recursos.

**Corte Mínimo**

Tem a função de prever em qual parte do grafo possui um estrangulamento em algum recurso que diminui o desempenho do sistema.

A ideia é verificar qual é a parte que ocorre estrangulamento para poder melhorar a fluidez naquele laço ou parte do grafo.

**Problema Hamiltoniano**



A figura representa as ligações rodoviárias entre 14 cidades.

Existe um caminho passando por cada cidade exatamente uma vez?

O problema do Caixeiro Viajante. Cada vértice é visitado somente uma vez.

Não forma ciclo.

**Problema Euleriano**

O problema consiste em encontrar caminhos, de inicio e fim no mesmo vértice, passando por todas as arestas uma única vez. O mesmo vértice pode ser visitado mais de uma vez.

**Problema do Caixeiro Viajante**

É um clássico exemplo de **problema de otimização combinatória**.

Leva em conta:

* Combinatória de percursos (Todos os percursos)
* Verificação para cada percurso qual o tamanho do caminho
* Paraca cada um dos caminhos verica-se qual é o MENOR.
* Muito custo quando com muitas vértices.

**Problema do Carteiro Chinês**

O problema consiste em encontrar um ciclo de peso mínimo que passe por cada aresta **pelo menos** uma vez.