UDESC – Universidade do Estado de Santa Catarina

DCC – Departamento de Ciência da Computação

Curso: BCC

Disciplina: Teoria dos Grafos – TEG0001

Notas de Aula – Parte V

Referências bibliográficas:

1. Cormen, T. Introduction to Algorithms. Third edition, 2009. The MIT Press
2. Bondy, J.A. Graph Theory with applications. Fifth edition.
3. Rosen K. Matemática discreta e suas aplicações, sexta edição, 2009
4. Gersting, Judith L. Fundamentos Matemáticos para a Ciência da Computação. Rio de Janeiro. 5a Ed. Editora. LTC,

Complementar:

1.) WEST, Douglas, B. Introduction to Graph Theory, second edition, Pearson, 2001. (*)

2.) SEDGEWICK, R. Algorithms in C – part 5 – Graph Algorithms, third edition, 2002, Addison-Wesley. (*)

3.) GOLDBARG, M., GOLDBARG E., Grafos: Conceitos, algoritmos e aplicações. Editora Elsevier, 2012. (*)

4.) FEOFILOFF, P., KOHAYAKAWA, Y., WAKABAYASHI, Y., uma introdução sucinta à teoria dos grafos. 2011. (www.ime.usp.br/~pf/teoriadosgrafos)

5.) DIESTEL, R. Graph Theory, second edition, springer, 2000.

6.) Bastante material disponível na internet.

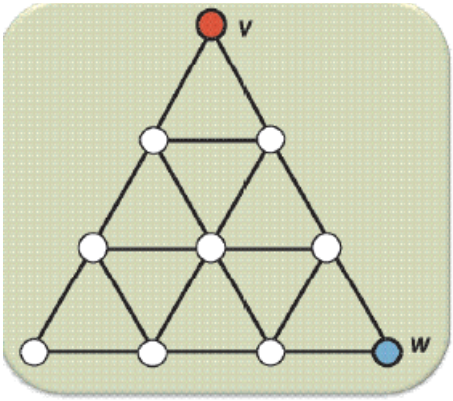
Modelos de redes

Caminhos em grafos

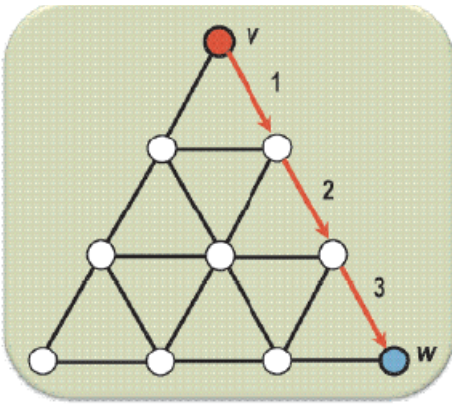
Caminho mínimo (mais curto) de um grafo G

O caminho mais curto em grafos possui diversas variantes.

O caminho mais curto entre os vértices v e w de um grafo não ponderado é aquele que acumula o menor número de arestas entre os referidos vértices.

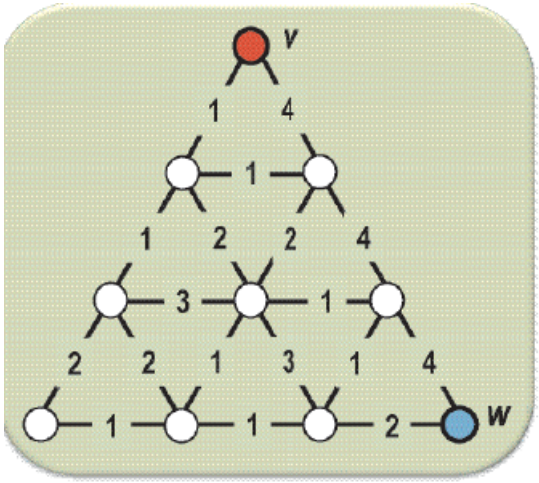


(1) Grafo não ponderado

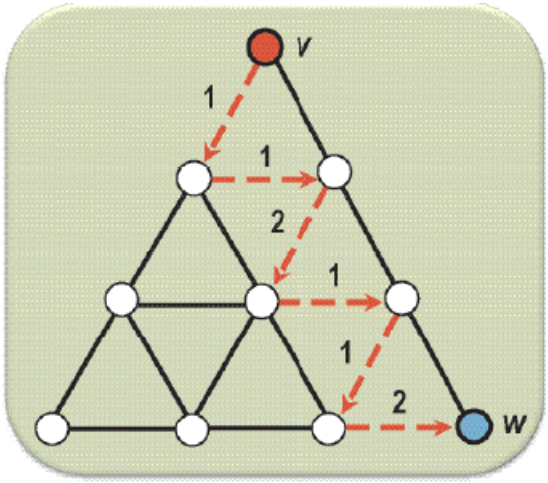


(2) Caminho mais curto entre *v* e *w*

O caminho mais curto entre os vértices v e w de um grafo G ponderado em arestas é aquele cuja soma dos pesos das arestas tem o menor valor possível dentre todos os caminhos existentes entre v e w.



(1) Grafo ponderado



(2) Caminhos mais curto entre *v* e *w*

Dado um grafo G(V,E) com **arestas ponderadas**, tal que o peso de um caminho P ={V0,V1,...,Vk} é dado pela soma dos pesos de cada uma das arestas que compõem o caminho.

1. ***Como obter um caminho mínimo entre o nó u e todos os demais nós v Ɛ V deste grafo?***

2. ***Como obter um caminho mínimo entre todos os nós deste grafo?***

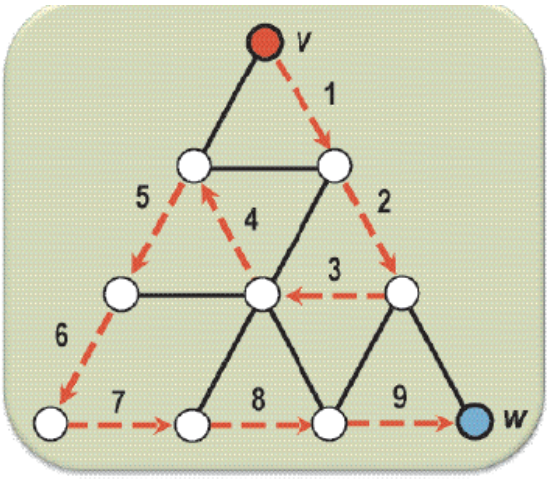
Um caminho mais curto do nó **u** ao nó **v** é definido como algum caminho P com peso w(p) = δ(u,v).

Um grafo com **comprimento ou peso** é um grafo onde cada aresta possui um **valor** representado por um **número real**.

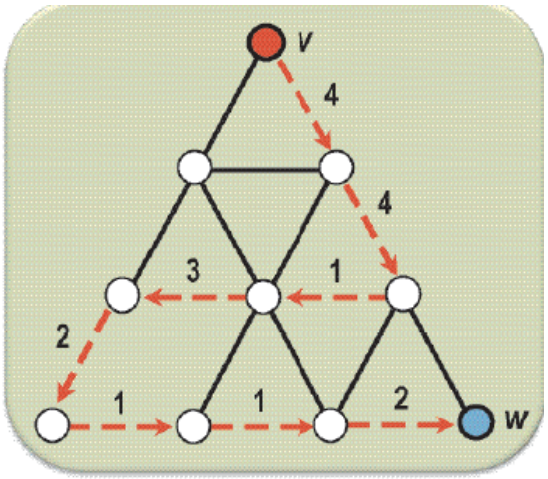
A soma de todos os pesos de todas as arestas é o **peso total do grafo**. Ou seja, o **comprimento** de um caminho é igual à soma dos comprimentos (distâncias) dos arcos que formam o caminho.

Caminho mais longo de um grafo G

O caminho mais longo em um grafo G é aquele que acumula o maior valor possível dentre todos os caminhos existentes entre v e w (grafo ponderado) ou percorre o maior número de arestas entre os referidos vértices (grafo não ponderado).



(1) Caminho mais longo do grafo não ponderado



(2) Caminho mais longo do grafo ponderado

O **comprimento ou peso** de um arco pode ter diversas interpretações dependendo da aplicação. Por exemplo: o peso ou comprimento de um arco pode estar relacionado à: custos, distâncias, consumo de combustível, dentre outros.

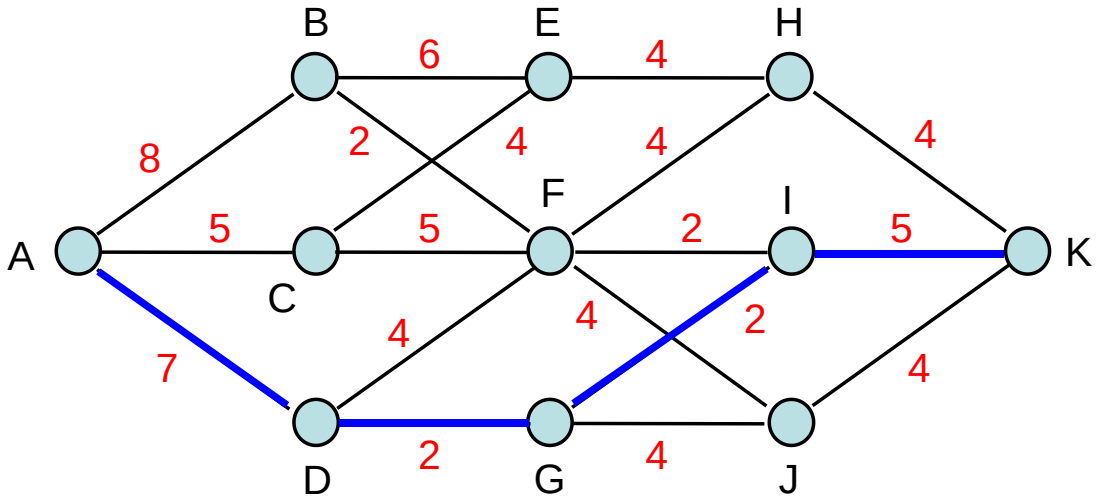
Vários problemas podem ser modelados usando grafos com pesos associados às suas arestas. Alguns exemplos:

. **Sistema de aviação;**
. **distâncias entre cidades;**
. **horários de voos;**

. redes de computadores.

Exemplo_ilustrativo: Mapa rodoviário.

O grafo G abaixo representa um mapa rodoviário de uma determinada região do Brasil, que interliga diversas cidades.
As cidades são representadas por vértices e os arcos representam as distâncias a serem percorridas entre as cidades. Dado este grafo determine qual o menor trajeto a ser percorrido entre as cidades A e K ?

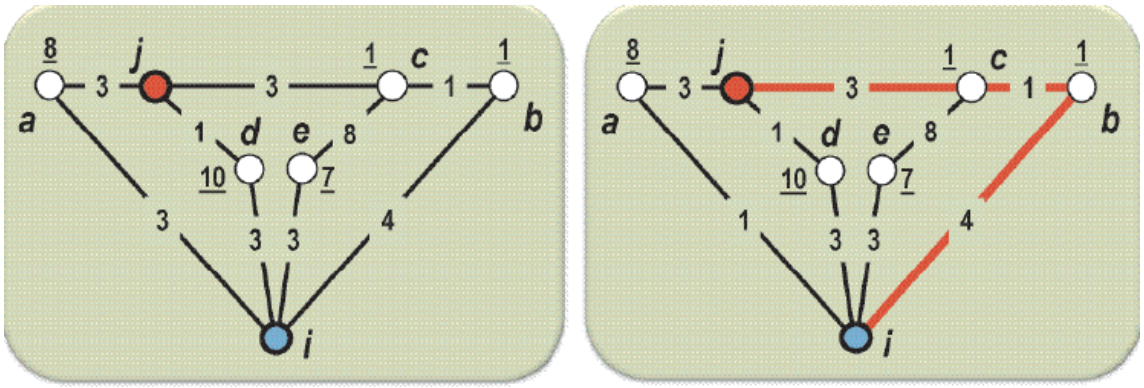


O trajeto mais curto pode ser identificado analisando-se o grafo e é calculado somando-se os valores das arestas que interligam as cidades (A,D,G,I,K).

Caminho mais curto com custos nos vértices

O caminho mais curto entre os vértices i e j de um grafo G ponderado em vértices e arestas é aquele cuja soma das arestas

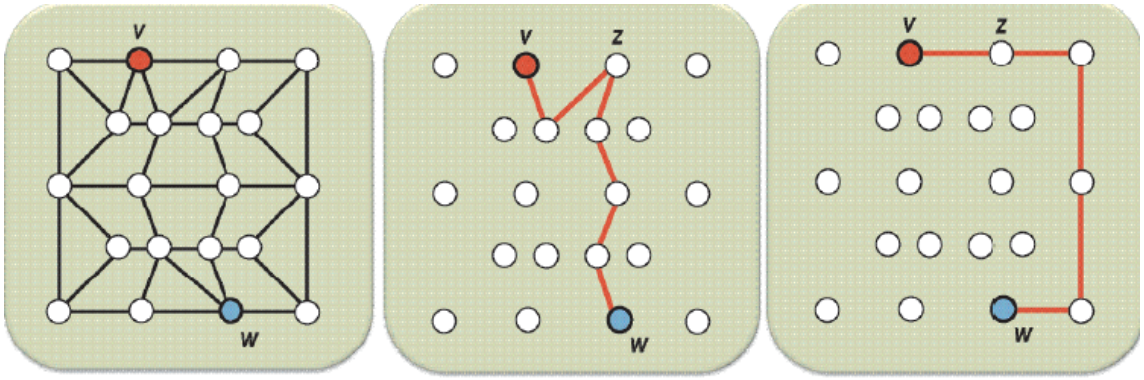
e dos vértices tem o menor valor possível dentre todos os caminhos existentes entre i e j.



(1) Grafo *G* (2) Caminho mais curto com custos nos vértices

Caminho disjunto em arestas

Dois caminhos v-w são ditos disjuntos em arestas quando não possuem aresta em comum.

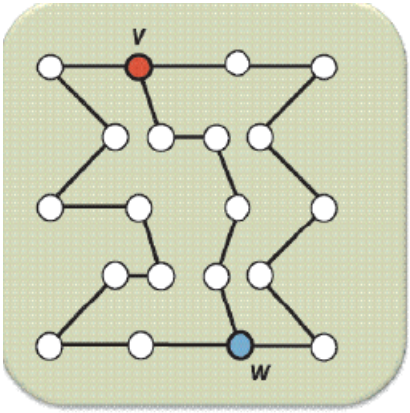


(1) Grafo *G* (2) Caminho 1 (3) Caminho 2

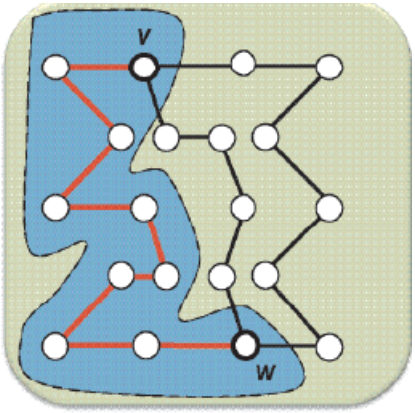
Partição de G em caminhos disjuntos

Dado um grafo G=(N,M), uma partição de G em caminhos disjuntos em vértices é um conjunto de caminhos P1 =

(N1,M1),..., Pr = (Nr,Mr) em G, tais que N1 U ... U Nr = N e Ni ∩ Nj = ○, para quaisquer i, j, i ≠ j, 1 <= i,j >= r.
Pode-se verificar que todos os vértices do grafo G, exceto v e w participam de algum caminho.



(4) Partição em caminhos disjuntos



(5) Destaque de um caminho *v-w*

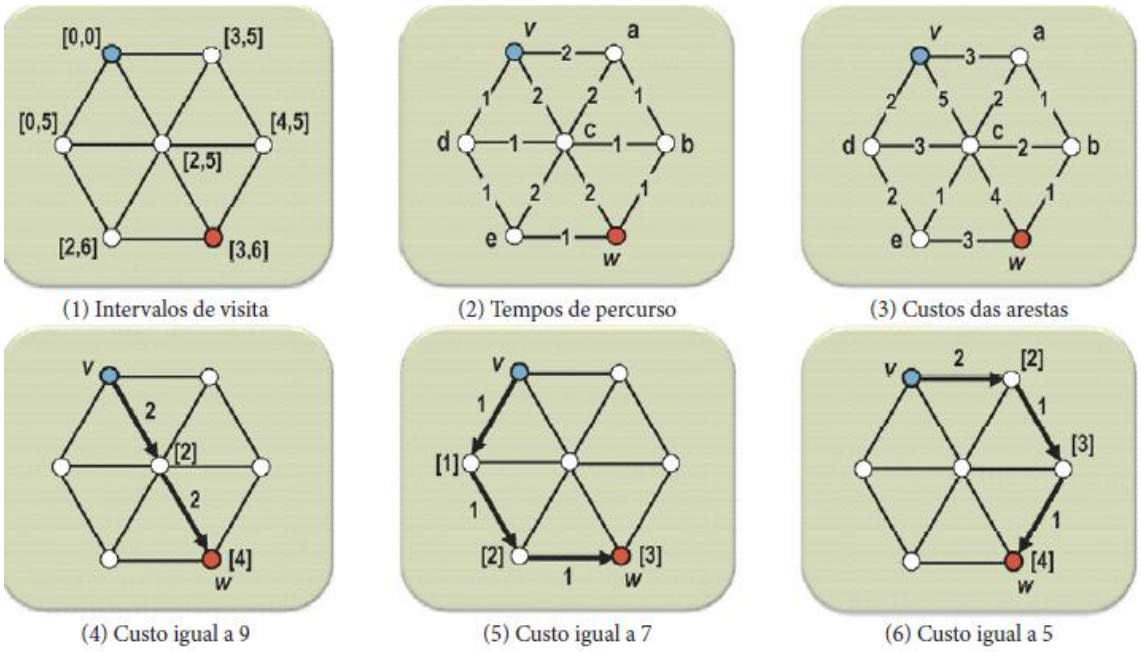
Dois caminhos disjuntos em arestas não são necessariamente disjuntos em vértices. Porém, caminhos disjuntos em vértices são sempre disjuntos em arestas.

Os conceitos de caminho disjunto em arestas e partição de G em caminhos disjuntos estão relacionados a importantes problemas de otimização combinatória.

Variantes do caminho mais curto

Os caminhos em grafos podem ser associados à restrições diversas. Por exemplo, um conjunto de restrições pode ser associado ao tempo, quando define-se os intervalos de tempo (mínimo e máximo) para realizar a visita aos vértices.

Dada a figura abaixo, o mesmo é representado por intervalos de tempo (vértices), e pelo custo (arestas). Observe na figurX.1 que cada vértice do grafo é caracterizado pelos tempos mínimos e máximos de visitação [Ti, Tj].

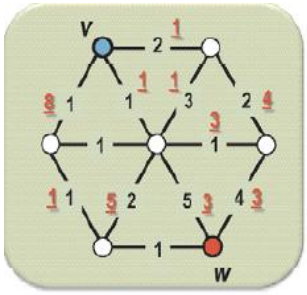


(1) Intervalos de visita
(2) Tempos de percurso
(3) Custos das arestas
(4) Custo igual a 9
(5) Custo igual a 7
(6) Custo igual a 5

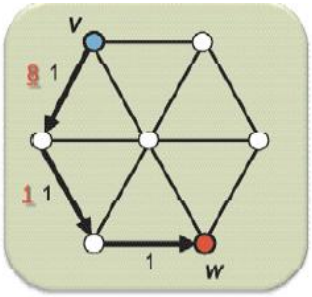
Caminho mais curto com restrição em peso

Dado um grafo G com pesos Wij e custos Dij associados às suas arestas (i,j), o caminho mais curto com restrição de peso é dado pelo menor caminho entre um dado par de vértices v e w tal que o valor dos pesos não ultrapasse um limite K.

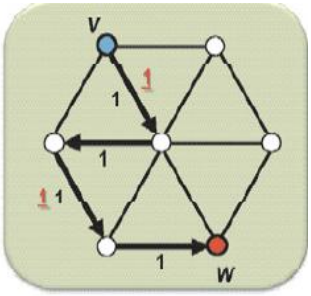
A figura apresenta o grafo G com pesos apresentados em vermelho e sublinhados e os custos estão representados em preto.



(1) Grafo G

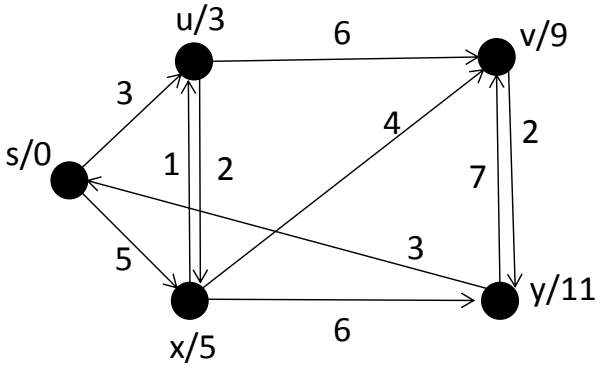


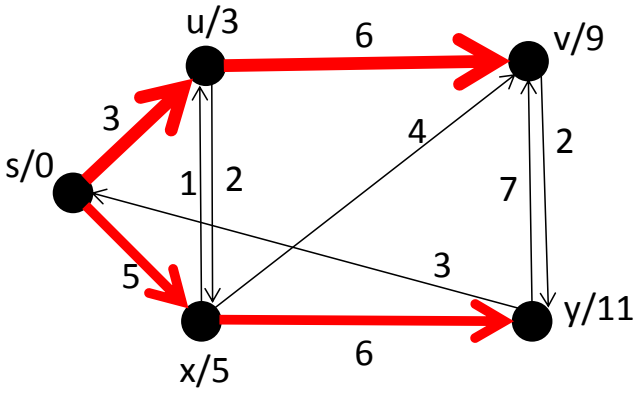
(2) Caminho=3 / Peso=9

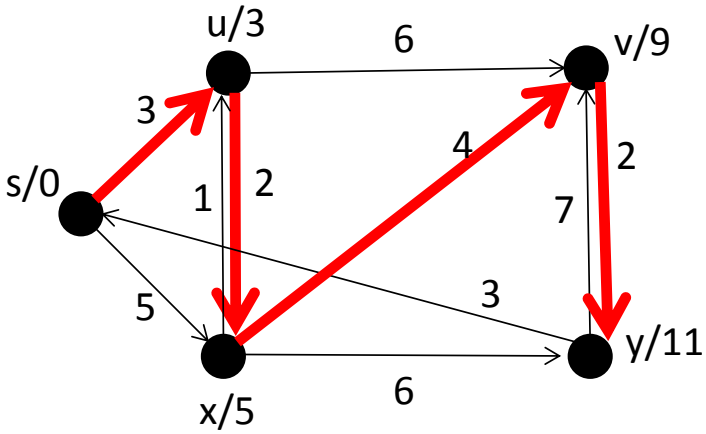


(3) Caminho=4 / Peso=2

Os caminhos mais curtos em um grafo G não são necessariamente únicos. A figura abaixo apresenta um grafo G que possui duas árvores distintas (com a mesma raíz em s) com caminhos mais curto.







A solução ótima que determina o caminho de menor custo será sempre dada por um caminho sem ciclos !

Os algoritmos a serem apresentados utilizam **o método de relaxação**.

Considere que d[v] contem o valor do caminho mais curto até o nó v; e que Π (v) contem o nó predecessor à v.

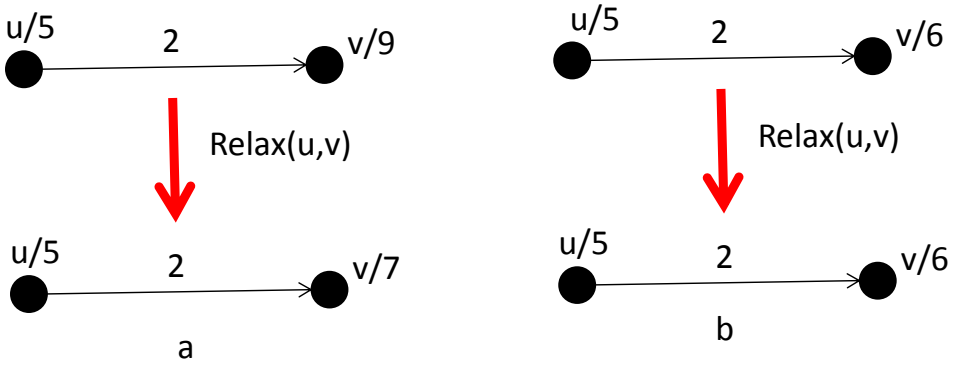
O **processo de relaxação** de uma aresta (u,v) consiste em testar a aresta (u,v) e verificar se o caminho mais curto para o nó v pode ser realizado através do nó u, e se for possível, o algoritmo deve atualizar **d[v] e Π (v).**

O algoritmo abaixo apresenta o procedimento de iniciação na determinação do caminho mínimo, bem como o processo de relaxação de uma determinada aresta (u,v).

```
INITIALIZE-SINGE-SOUCE (G,s)
FOR < each vertex v Ɛ V >   DO
        d[v] = infinito
        Π (v) =  NIL;
D[s] = 0;          /* nó incial

RELAX (u,v,w)
If < d[v] > d[u] + w[u,v] >
        Then
                d[v] = d[u] + w(u,v);
                Π (v) = u;
```

Ressalta-se que um passo no processo de relaxação pode decrementar o valor do d[v] (caminho mais curto) e atualizar o valor do predecessor **Π (v).**

A figura abaixo apresenta dois exemplos de relaxação de uma aresta, em que um caminho mais curto (estimado) é decrementado, e no segundo nenhuma alteração é realizada.



Propriedades da relaxação:

O peso **W** do caminho **p = {V0,V1,...,Vk}** é dado pela soma dos pesos de suas respectivas arestas:

$$\mathbf{W(p)} = \sum_{i=1}^{k} \mathbf{W}(V\,i-1, Vi)$$

Define-se o peso do caminho mais curto de u a v como sendo:

δ (u,v) = { min {w(p) : u → v}; se existir um caminho de u para v;

∞ ; caso contrário.

Portanto, um caminho mais curto do nó u ao nó v é então definido como qualquer caminho p com peso w(p) = δ (u,v).

. Considere o grafo G(V,E) , valorado, direcionado, com função de peso W que associa um peso Wi a cada uma das arestas (u,v) Ɛ E. Então, imediatamente após o relaxamento da aresta (u,v), tem-se que: d[v] <= d[u] + w(u,v)

. Considere o grafo G(V,E) , valorado, direcionado, com função de peso W que associa um peso Wi a cada uma das arestas (u,v) Ɛ E. Considere que o nó s é o nó inicial, e que o grafo é devidamente iniciado. Então d[v] >= δ (u,v) para todo v Ɛ V.

Exercício: Realize a prova destas duas propriedades acima de relaxamento.

Um dos problemas clássicos que envolvem distâncias entre cidades é o problema do **caixeiro viajante**. Este problema consiste em identificar uma ordem de visita aos vértices (cidades) do grafo em que o caixeiro-viajante visite cada cidade exatamente uma vez, de modo que ele percorra uma distância total mínima.

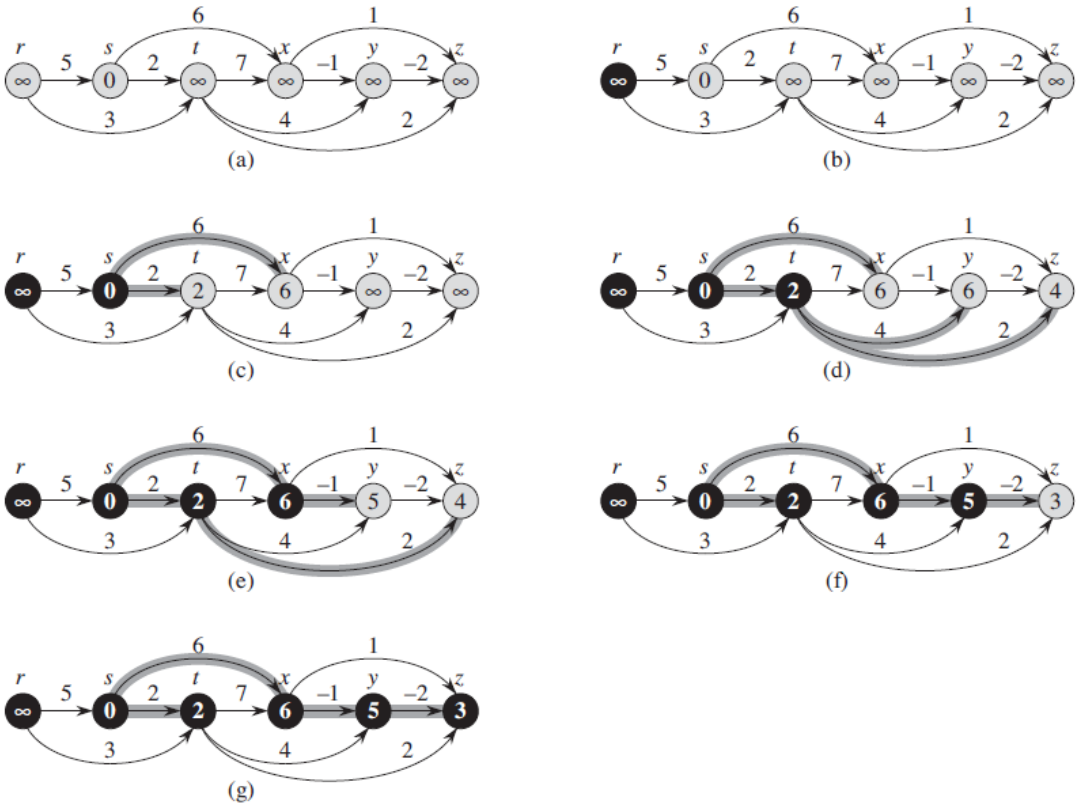
Caminho mínimo entre o nó u e todos os demais nós v Ɛ V de um grafo G, acíclico e direcionado (DAG).

Se um DAG G = (V,E) possui um nó s e não contêm ciclos, então ao final do procedimento de busca do caminho mais curto ter-se-á identificada uma árvore de caminho que representará o caminho mais curto.

A determinação do menor caminho a partir de um nó fonte em um DAG é realizada através do processo de relaxação e seguindo a ordenação dos vértices.

Se o grafo contem um caminho entre dois nós u e v, então o nó u precede o nó v. Apenas uma passagem por cada nó é realizada.

A figura abaixo apresenta a execução do algoritmo a partir do nó s.



Uma aplicação interessante da aplicação do algoritmo de busca em uma DAG é destinada as redes PERT. As arestas representam tarefas a serem executadas, e o peso da aresta representa o tempo necessário para executar a respectiva tarefa.

Por exemplo, se um nó u antecede o nó v, então a tarefa (u,v) deve obrigatoriamente anteceder a tarefa v. Portanto, o DaG representa a ordem da sequencia das tarefas que devem ser executadas em uma ordem específica.

Um caminho crítico representa o caminho mais longo a ser preenchido.

Exercício: Altere o algoritmo para determinação do caminho mais curto em um DAG, apresentado no exemplo anterior, considerando que os pesos são agora atribuídos aos nós que representam as tarefas a serem executadas. A aresta que interliga os nós (u,v) indica que tem-se uma restrição temporal, que obrigatoriamente a tarefa u tem que ser executada antes da tarefa v.

Caminho mínimo entre o nó u e todos os demais nós v Ɛ V de um grafo G.

Para a determinação do caminho mínimo serão apresentados dois algoritmos: **Dijsktra e Belmann-Ford**.

Os dois algorítmos apresentados a seguir diferem pelo fato de que o algoritmo de **Dijkstra** considera pesos positivos, e o de **Bellmann-Ford** trata pesos positivos e negativos.

Outro ponto importante é que no algoritmo de **Dijkstra** cada aresta é “relaxada” apenas 1 vez, enquanto que no de **Bellmann-Ford** cada aresta pode ser “relaxada” 1 ou mais vezes.

Algorítmo de Dijkstra (1959)

Existem vários algoritmos diferentes para encontrar um caminho mais curto entre dois nós. Apresentaremos o algorítmo para o caminho mínimo conhecido como ***algoritmo de Dijkstra*** (matemático holandês), apresentado em 1959.

Suponha um **grafo simples G(V,E), conexo e com pesos positivos em suas arestas**. Portanto assume-se que w(u,v) >= 0 para cada aresta (u,v) **Ɛ V.**

. O algoritmo mantêm um conjunto S de vértices em que os pesos do caminho mais curto desde o nó inicial foram previamente determinados. Ou seja, para todos os vértices v Ɛ V tem-se que d[v] = δ (u,v).
. O algorítmo repetidamente seleciona o vértice u Ɛ V-S com o mínimo caminho mais curto "estimado", insere u em S, e realiza o processo de relaxação de todas as arestas que deixam o vértice u.
. Uma fila de prioridades dada por Q contêm todos os vértices em V-S, de acordo com os seus respectivos valores de d.
. O pseudocódigo apresentado abaixo assume que o grafo G está representado através de uma lista de adjacência.

. O algoritmo inicia associando o valor 0 ao vértice a e o valor ∞ aos outros vértices.
. A linha 1 realiza a iniciação dos valores de d e de Π.
. A linha 2 inicia o conjunto S;
. A linha 3 inicia a fila de prioridades Q para conter todos os vértices em V-S=V-0=V.

A cada passo do loop do While (linhas de 4 a 8) um nó u é extraído de Q=V-S e inserido no conjunto S.

. As linhas 7 e 8 realizam o processo de relaxação da aresta (u,v), deixando na sequencia o nó u. Na sequencia atualiza-se o d[v] e o predecessor Π[v] se o caminho mais curto para v puder ser melhorado através do nó u.

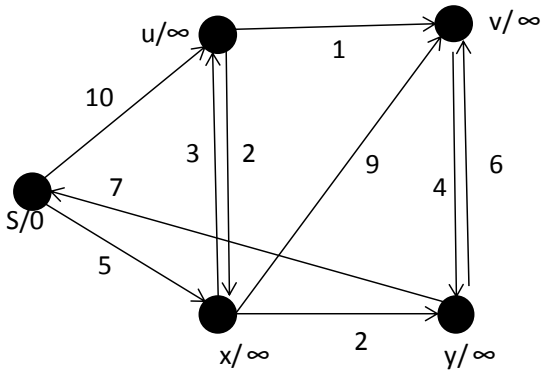
. Observe que os nós nunca são inseridos em Q após a linha 3 e que cada nó é extraído de Q e inserido em S exatamente uma vez. Assim, o loop while realiza exatamente |V| iterações.

. O algorítmo de Dijkstra sempre faz a busca e escolhe os nós que estão mais próximos (greedy strategy).

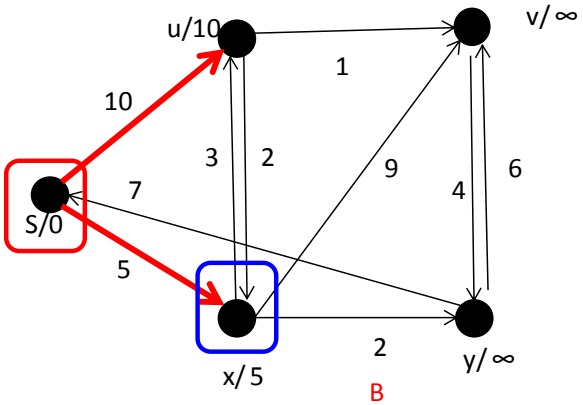
```
DIJKSTRA(G,w,s)
1 INITIALIZE-SINGE-SOUCE (G,s)
2 S = 0;
3 Q = V[G];
4 WHILE Q ≠ 0 DO
5       u = EXTRACT-MIN(Q);
6       S = S U {u};
7       FOR <each vertex v Ɛ Adj[u]> DO
8               RELAX(u,v,w)

INITIALIZE-SINGE-SOUCE (G,s)
FOR < each vertex v Ɛ V >    DO
        d[v] = infinito
        Π (v) =  NIL;
D[s] = 0;           /* nó incial

RELAX (u,v,w)
If < d[v] > d[u] + w[u,v] >
        Then
                d[v] = d[u] + w(u,v);
                Π (v) = u;
```

Exemplo_a: Dado o grafo G(V,E) da figura abaixo encontrar o caminho de menor custo entre o nó s e os demais nós do grafo G

Passos 01 (iniciação do grafo) e 02:



A

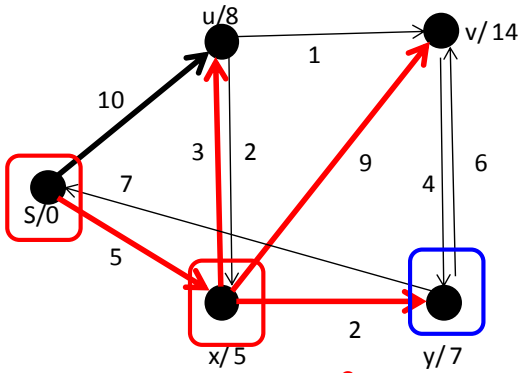


B

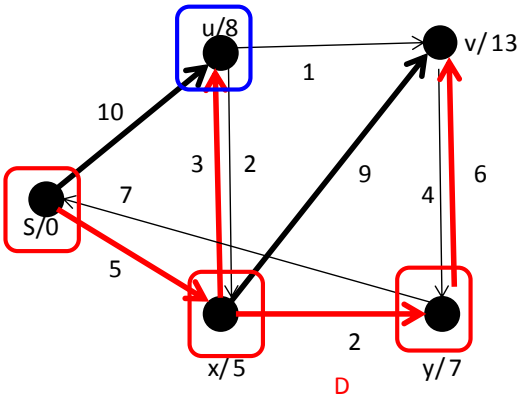
vértices	s	u	v	x	y
estimativas	0	∞	∞	∞	∞
precedentes	-	-	-	-	-

vértices	s	u	v	x	y
estimativas	0	10	∞	5	∞
precedentes	s	s	-	s	-

Passos 03 e 04:



C

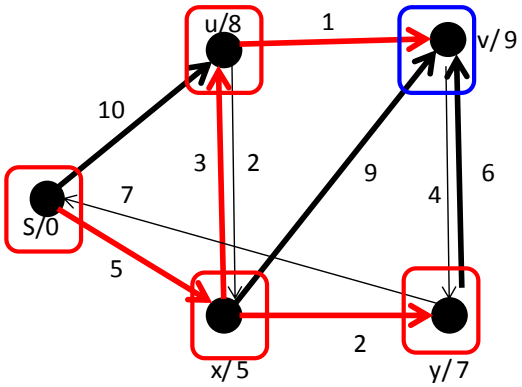


D

vértices	s	u	v	x	y
estimativas	0	8	14	5	7
precedentes	s	x	x	s	x

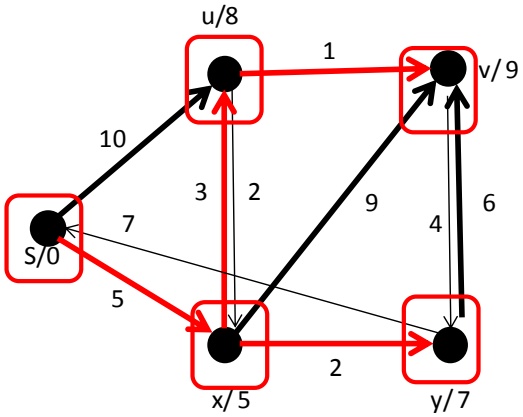
vértices	s	u	v	x	y
estimativas	0	8	13	5	7
precedentes	s	x	y	s	x

Passos 05 e 06:



E

vértices	s	u	v	x	y
estimativas	0	8	9	5	7
precedentes	s	x	u	s	x



F

vértices	s	u	v	x	y
estimativas	0	8	9	5	7
precedentes	s	x	u	s	x

predecessor

conjunto S de nós

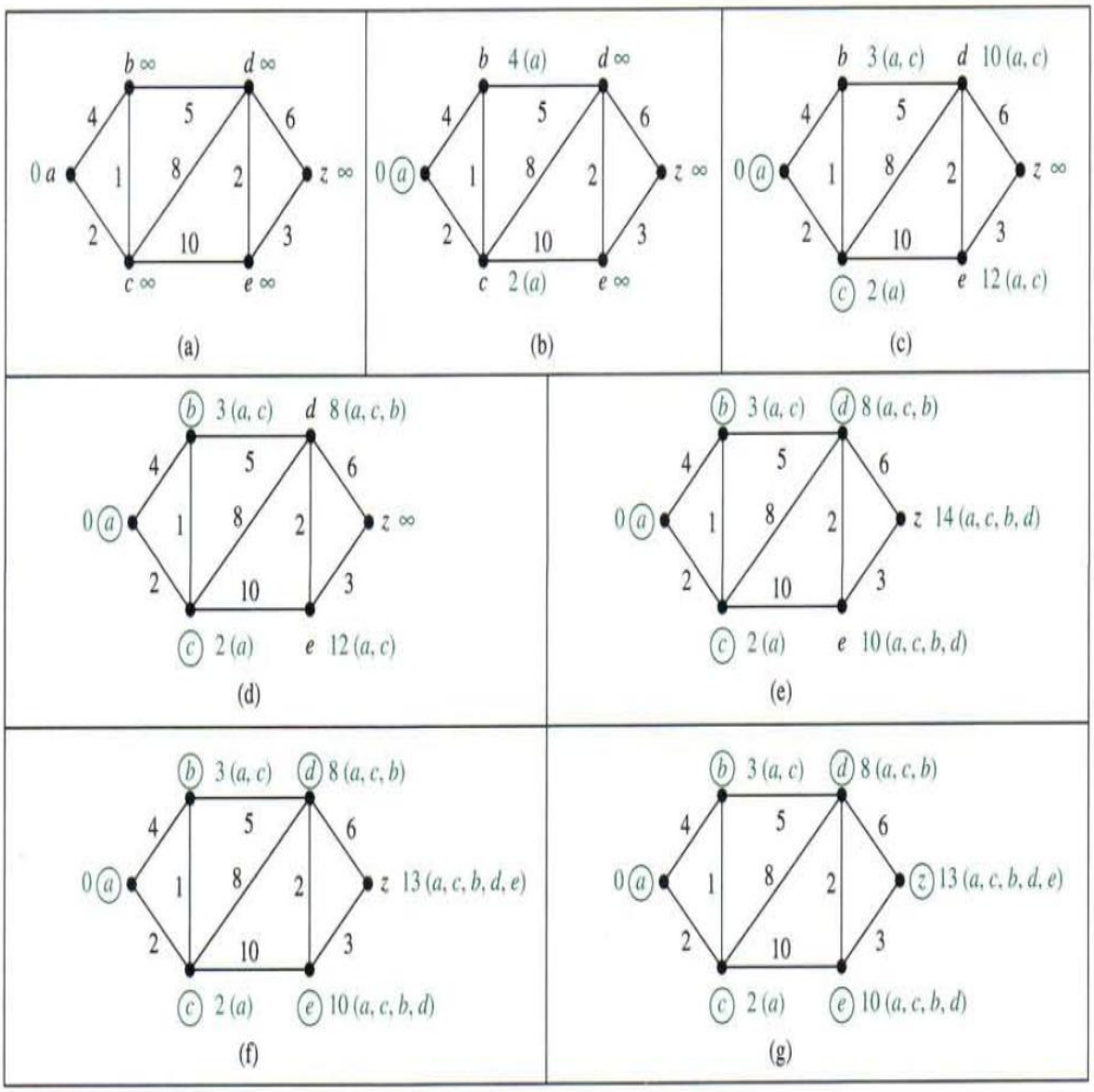
Exercício: Refaça o problema anterior de determinação do caminho mínimo Supondo que o peso do arco (y,v) no grafo acima é de 2

Resposta: Verifique que existem dois caminhos de custo mínimo do vértice s para v. Esta duplicidade resulta em dois precedentes para o vértice v.

vértices	s	u	v	x	y
estimativas	0	8	9	5	7
precedentes	s	x	u,y	s	x

Exemplo_b: Utilize o algoritmo de Dijkstra para encontrar o comprimento de um caminho mais curto entre os vértices de a à z no grafo abaixo.

As iterações realizadas estão demonstradas na figura abaixo. A cada iteração os nós que fazem parte do conjunto Sk são circundados. O algoritmo termina quando o nó Z fizer parte do conjunto Sk.



. Teorema: Prove que ao executar o algorítmo de Dijkstra em um grafo G(V,E) direcionado e valorado com peso w não negativo, então ao término, d[u] = δ (s,u) para todos os vértices u Ɛ V.

. Prove que ao executar o algorítmo de Dijkstra em um grafo G(V,E) direcionado e valorado com peso w não negativo, e fonte no nó s, então ao término da execução do algoritmo o subgrafo predecessor G_π é uma árvore de caminhos mais curtos enraizada em s.

Algorítmo de Bellman-Ford

Este algoritmo resolve o problema do caminho mais curto, porém, de forma mais abrangente. **Considera arestas com pesos negativos.**

Dado um **grafo G(V,E), direcionado e com arestas valoradas**, com fonte no nó s e função de peso w: E → R, o algoritmo de **Bellman-Ford** retorna um valor booleano indicando se existe ou não um ciclo de peso negativo, que é alcançável a partir da fonte.

Se, por acaso, existir o ciclo negativo então o algoritmo informa que **não há solução para o problema**. Por outro lado, se não existir tal ciclo no grafo G, o algoritmo identifica os caminhos mais curtos e seus respectivos pesos.

Tal como o algoritmo de Dijkstra, o Bellman-Ford utilize a técnica de **relaxação**, progressivamente decrementando o d[v] (estimado) da fonte, para cada um dos vértices v Ɛ V até que alcance o peso do caminho mais curto δ (u,v).

O algorítmo retorna TRUE, se e somente se, o grafo não contêm ciclos com pesos negativos que são alcançáveis a partir da fonte.

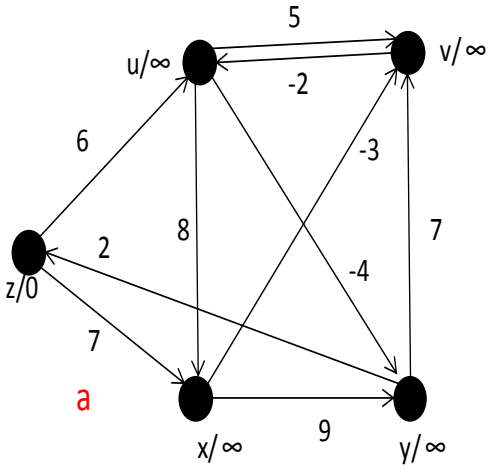
O algoritmo de Bellmann relaxa todas as arestas simultaneamente e realiza |V|-1 passagens sobre as arestas de G, onde |V| corresponde ao número de vértices do grafo. Veja exemplo abaixo.

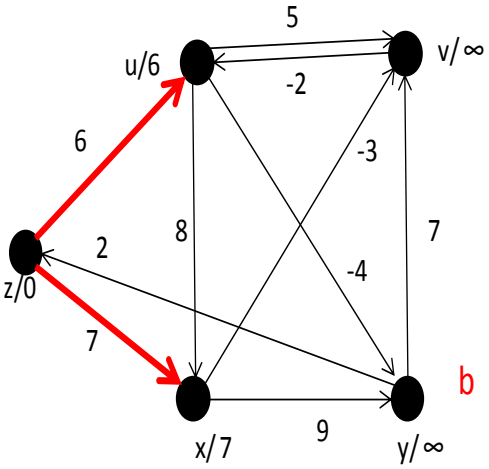
```
BELMANN-FORD(G,w,s)
1 INITIALIZE-SINGE-SOUCE (G,s)
2 FOR  < i=1 TO |V(G)| - 1 > DO
3       FOR < each edge (u,v) Ɛ E[G]> DO
4               RELAX(u,v,w)
5 FOR < each edge (u,v) Ɛ E[G] >  DO
6       IF < d[v] > d[u] + w(u,v) >
7               THEN return FALSE
8  RETURN TRUE
```

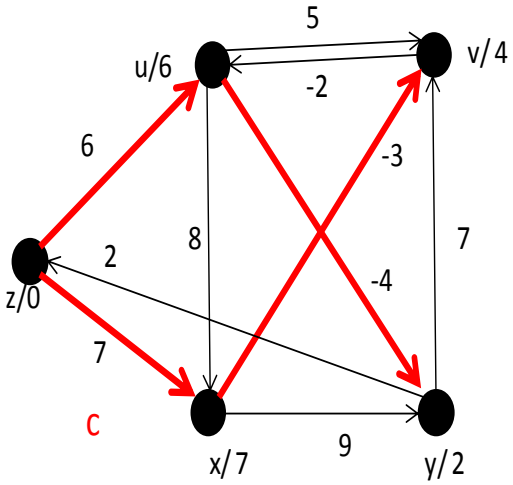
Cada uma das passagens corresponde a uma iteração do loop FOR (linhas 2-4), e consiste no processo de relaxação de cada aresta.

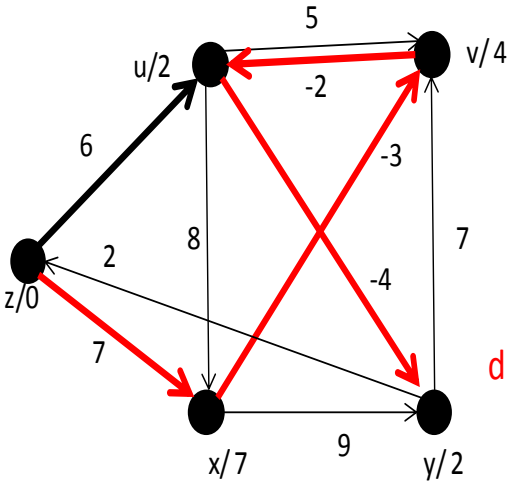
Após executar |V|-1 vezes, na sequencia as linhas 5 à 8 do algoritmo verificam ciclos de peso negativos e retornam o valor booleano apropriado.

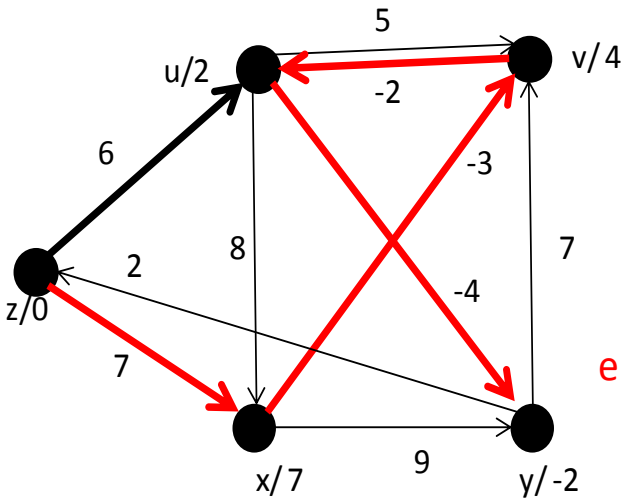
A figura a seguir apresenta a **execução do algorítmo de Bellman-Ford sobre o grafo G dado, com Z sendo o nó fonte**.



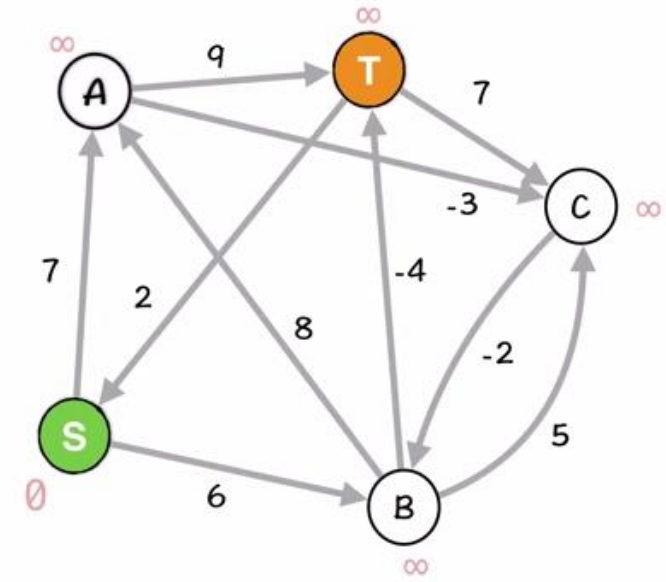








Exercício: Dado o grafo G abaixo utilize o algoritmo de Belmann-Ford para calcular o menor caminho entre o nó S e os demais nós do grafo.



Prove que dado um grafo G=(V,E) direcionado e valorado com fonte no nó S e função de peso w: E → R. Assumindo que G não contem ciclos de peso negativos que são alcançáveis a partir de s. Então ao final da execução do algoritmo de Bellman-Ford tem-

se: d[v] = δ (s,v) para todos os vértices v que são alcançáveis a partir de s.

Teorema: dado um grafo G=(V,E) direcionado e valorado com fonte no nó S e função de peso w: E → R. Assumindo que G não contem ciclos de peso negativos que são alcançáveis a partir de s. Então o algoritmo retorna TRUE, tem-se d[v] = δ (s,v) para todos os vértices v Ɛ V, e o subgrafo predecessor dado por Gπ é uma árvore de caminho mais curto enraizada em s. Se G contem um ciclo de peso negativo alcançável a partir de do nó s , então o algoritmo retorna FALSE.

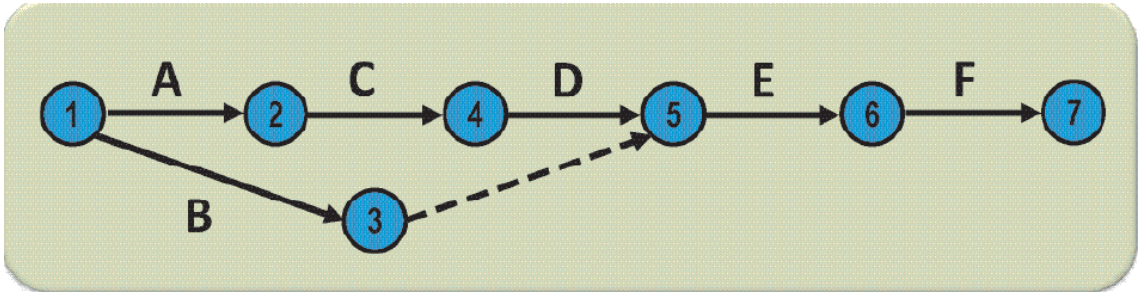
CPM – Critical Path Method

O CPM é um método de identificação do caminho crítico dada uma sequencia de atividades. Isto é, quais as atividades de uma sequência não podem sofrer alteração de duração sem que isso reflita na duração final, por exemplo, de um projeto.

Com exemplo, tem-se a tabela abaixo que contempla um conjunto encadeado de atividades que fazem parte do processo de fabricação de uma estante de madeira.

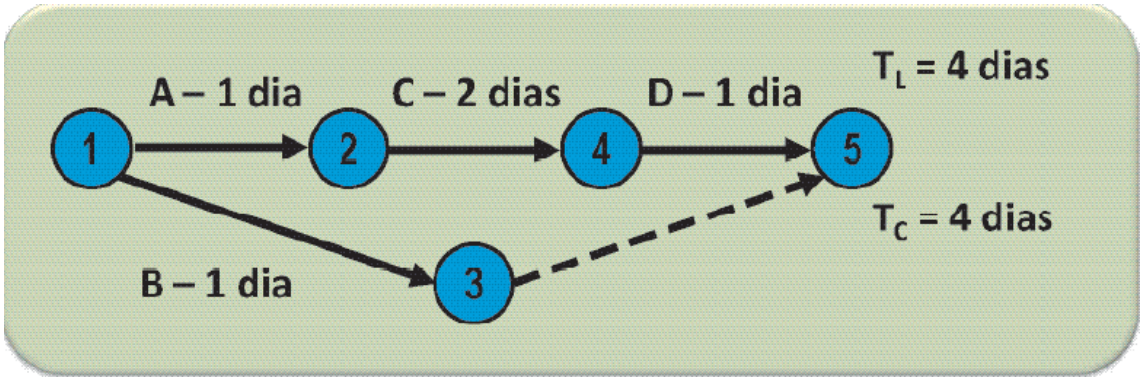
Atividade	Discrição	Duração	Anterior	Posterior
A	Comprar tábuas	1 dia	–	3
B	Comprar parafusos	1 dia	–	5
C	Cortar as tábuas	2 dias	1	4
D	Pintar as tábuas	1 dia	2	5
E	Montar as tábuas com parafusos	1 dia	4,2	6
F	Transportar a estante	1 dia	5	–

Com base nos dados da tabela anterior monta-se o grafo G com a sequencia lógica (precedência) das atividades de fabricação da estante. Verifique que no grafo abaixo as atividades fluem da esquerda para a direita e o grafo não possui ciclos.



As atividades estão representadas pelas setas. Os vértices representam os eventos instantâneos que caracterizam o início e

o término das atividades especificadas nas setas. A seta pontilhada representa a dependência lógica da atividade de montagem das tábuas em relação à atividade de compra dos parafusos. A duração desta atividade é igual à zero.



Pelo caminho 1-2-4-5 a conclusão da atividade 4-5 representada pelo vértice 5 somente poderá ocorrer em 4 dias. Pelo caminho 1-3-5, em somente 1 dia. A diferença entre esses tempos é denominada de folga.

Na solução da rede por eventos, o caminho mais longo vai determinar o tempo final da conclusão de todo o projeto. As atividades que pertencem ao caminho mais longo não possuem folga em sua execução.

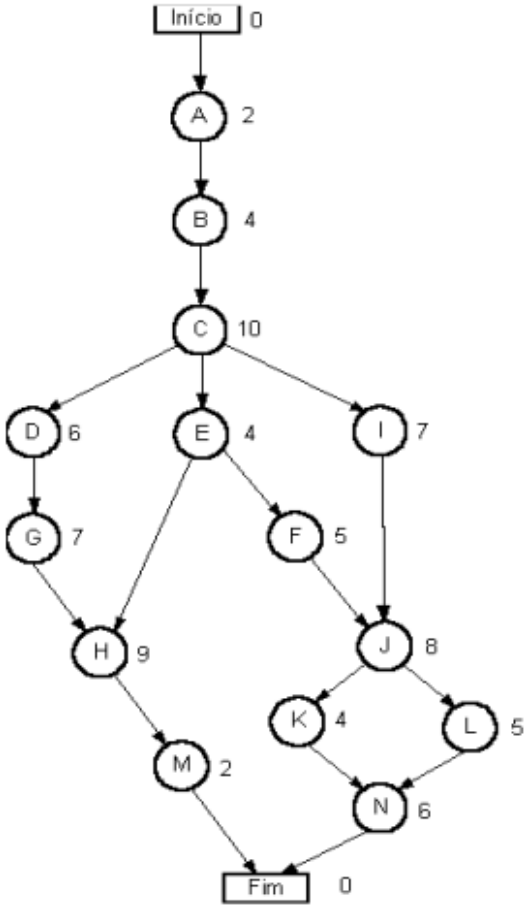
Exercício: Uma empresa ganhou uma concorrência para a construção de um galpão industrial. A relação de atividades a serem realizadas pela empresa a fim de terminar a construção do galpão estão descritas na tabela abaixo. Modele o problema utilizando grafos e determine o tempo mínimo necessário para que a obra seja finalizada. Algumas atividades podem ser realizadas em paralelo.

Utilizar os nós para representar as atividades e os arcos para representar as relações de precedência.

Tabela 1 - Atividades, Atividades Precedentes e Duração Estimada

Atividade	Descrição	Atividades Precedentes	Duração Estimada (semanas)
A	Escavação	-	2
B	Fundação	A	4
C	Paredes	B	10
D	Telhado	C	6
E	Encanamento Exterior	C	4
F	Encanamento Interior	E	5
G	Muros	D	7
H	Pintura Exterior	E,G	9
I	Instalação Elétrica	C	7
J	Divisórias	F,I	8
K	Piso	J	4
L	Pintura Interior	J	5
M	Acabamento Exterior	H	2
N	Acabamento Interior	K,L	6

Resposta:



Caminhos mínimos entre todos os pares de um grafo G

Dado um grafo G(V,E) deseja-se determinar para todos os pares de vértices u,v Ɛ V, o caminho mais curto ou menor peso de u para v, onde o peso do caminho é dado pela somatória dos respectivos pesos das arestas que compõem o caminho.

Uma solução pode ser executar com os algoritmos de **Dijkstra** ou **Bellman-Ford** por um número |V| de vezes. Porém, vamos apresentar **algoritmos que são mais eficientes** para este propósito.

Tal como os algoritmos de determinação de caminhos mínimos a partir de um vértice s, os algoritmos de caminhos mínimos entre todos os pares de um grafo G também utilizam matriz de adjacência como forma de representar o grafo G.

A **entrada de dados** dos algorítmos de determinação dos caminhos mais curtos entre os pares de vértices é dada pela matriz de adjacência W de ordem NxN, representando os pesos das arestas de um grafo direcionado **G(V,E).** Ou seja, W_{ij} pode assumir os seguintes valores:

. 0 , se i = j;

. o valor do peso correspondente a aresta direcionada (i,j), se i ≠ j e (i,j) Ɛ E;

. ∞ , se i ≠ j e (i,j) (não Ɛ) E

A **saída de dados** dos algorítmos de determinação dos caminhos mais curtos entre os pares de vértices é dada pela matriz D de

ordem NxN. Cada entrada da matriz é dada por d_{ij} que contêm o peso do caminho mais curto do nó i ao nó j.

Ou seja se denotarmos **δ(i,j)** como sendo o peso do caminho mais curto, então d_{ij} **= δ (i,j).**

Para resolver os problemas de determinação dos caminhos de menor custo entre todos os pares de um grafo G necessita-se computar não somente os pesos dos caminhos de menor custo, mas também determinar a **matriz predecessora ∏,** em que cada elemento (i,j) desta matriz é dado por:

. NIL, se i=j OU se não existir um caminho de i para j;

. Caso contrário, o valor de (i,j) é de algum predecessor de j em um caminho mais curto a partir de i.

Algorítmo de Floyd-Warshall

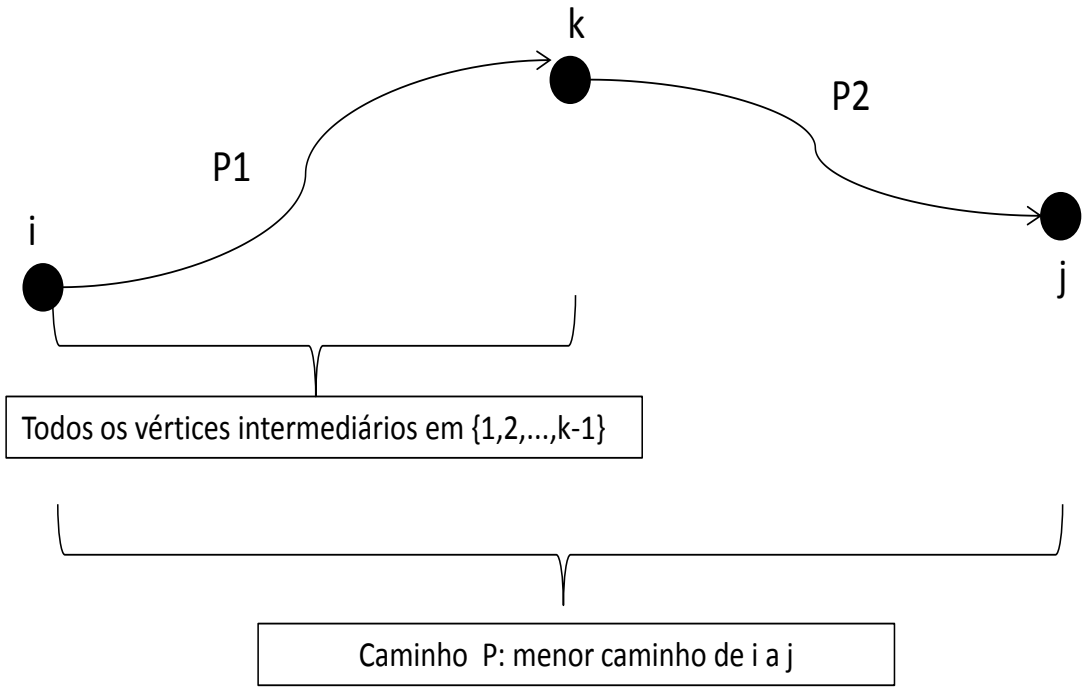
No algoritmo de **Floyd-Warshall** é permitido grafos com arestas de pesos negativos, porém, sem ciclos negativos.

Este algoritmo considera os nós intermediários de um caminho mais curto, em que um nó intermediário de um caminho simples p = {V1, V2, ..., Vl} é dado por qualquer nó de p que não seja V1 ou Vl, portanto, qualquer nó do conjunto {V2, V3, ..., V(l-1)}.

Considerando que os vértices de um grafo G são dados por V={1,2,...,n}, e que {1,2,...,k} corresponde a um subconjunto de V para algum valor de K.

Para qualquer par de vértices i,j Ɛ V, considera-se todos os caminhos de i para j em que os nós intermediários são todos escolhidos a partir de {1,2,...,k}. Considere que P é um caminho simples de menor peso dentre todos.

O algoritmo de **Floyd-Warshall** explora o relacionamento entre o caminho P e os caminhos mais curtos do nó I ao nó J, considerando todos os nós intermediários que fazem parte do conjunto {1,2,3...,k-1}.



Uma **solução recursiva** que calcula o caminho mais curto pode ser dada por:

$$Dij^{(k)} = Wij \qquad \text{Se K=0;}$$

$$= \min\left(Dij^{(k-1)} ;\ Dik^{(k-1)} + Dkj^{(k-1)}\right) \qquad \text{Se K >=1.}$$

Para qualquer caminho, todos os nós intermediários estão no conjunto {1,2,...,n}. A matriz $D^{(n)} = dij^{(n)}$ **fornece o resultado final que é dado por:** $dij^{(n)}$ **= δ (u,v), para todo i,j Ɛ V.**

FLOYD-WARSHALL(w)

1 n = W.rows

2 $D^{(0)} = W$

3 For k=1 TO n

4 Let $D^{(k)} = dij^{(k)}$ be a new NxN matrix

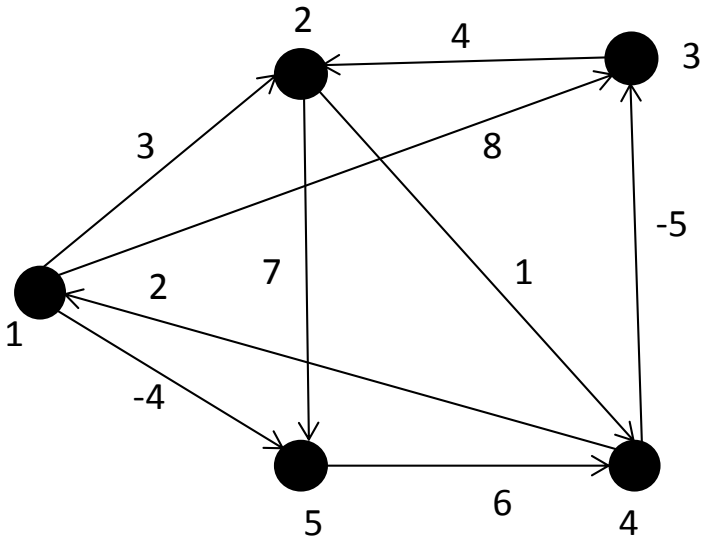
5 For I = 1 to N

6 For J = 1 to N

7 $Dij^{(k)} = min\,(Dij^{(k-1)}\,;\, Dik^{(k-1)} + Dkj^{(k-1)})$

8 return $D^{(n)}$

Exemplo: Dado o grafo G abaixo utilize o algoritmo de FLOYD para determinar o menor caminho entre os vértices V de G.



D^0

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

D^1

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

D^2

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

D^3

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	2
∞	∞	∞	6	0

D^4

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

D^5

0	1	-3	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

Fecho transitivo de um grafo direcionado G

Fecho transitivo direto: O Fecho Transitivo Direto de um vértice corresponde ao conjunto dos vértices de um grafo alcançáveis a partir de v.
Os vértices do conjunto do fecho transitivo são chamados de **descendentes** de v.

Dado um grafo direcionado G = (V,E) com um conjunto de vértices V = {1,2,...,n}. Temos como objetivo determinar se o grafo G contêm um caminho de nó i ao nó j através de todos os nós que compõem o grafo G, ou seja, i,j Ɛ V.

Define-se **fecho transitivo** de G como sendo o grafo G^* **= (V,** E^***), onde,**

E^* **= {(i,j) : existe um caminho do nó i ao nó j em G}**

Uma maneira de determinar o fecho transitivo de um grafo consiste em atribuir a cada uma das arestas de G o peso 1, e aplicar o algoritmo de FLOYD-WARSHALL. Se existir um caminho do nó i para o nó j, então:

Obtem-se: $dij < n$ **;**

Caso contrário: $dij = \infty$

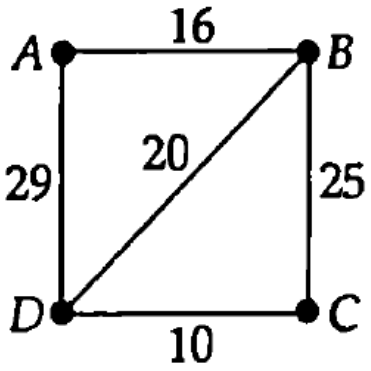
Fecho transitivo indireto: O Fecho Transitivo Indireto de um vértice corresponde ao conjunto dos vértices de um grafo a partir dos quais v é alcançável. Os vértices que fazem parte do conjunto do fecho transitivo indireto são chamados de **ascendentes** de v.

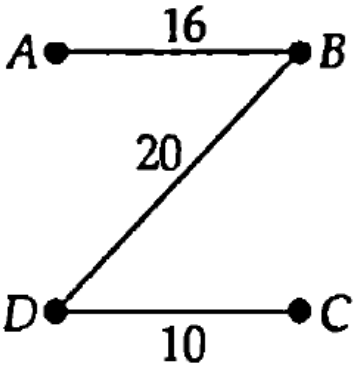
Árvore geradora mínima

Dado um grafo G, que seja conectado e com pesos nas arestas. O problema consiste em **identificar uma árvore geradora que apresente um custo mínimo**, ou seja, um subgrafo T que apresente um peso total mínimo, passando por cada um dos vértices do grafo G.

Portanto, uma árvore geradora mínima para um grafo com peso é uma árvore geradora que tem o menor peso total possível dentre todas as possíveis árvores geradoras do grafo.

Dado o grafo G da figura abaixo à esquerda. O grafo à direita corresponde a uma árvore geradora mínima de G.





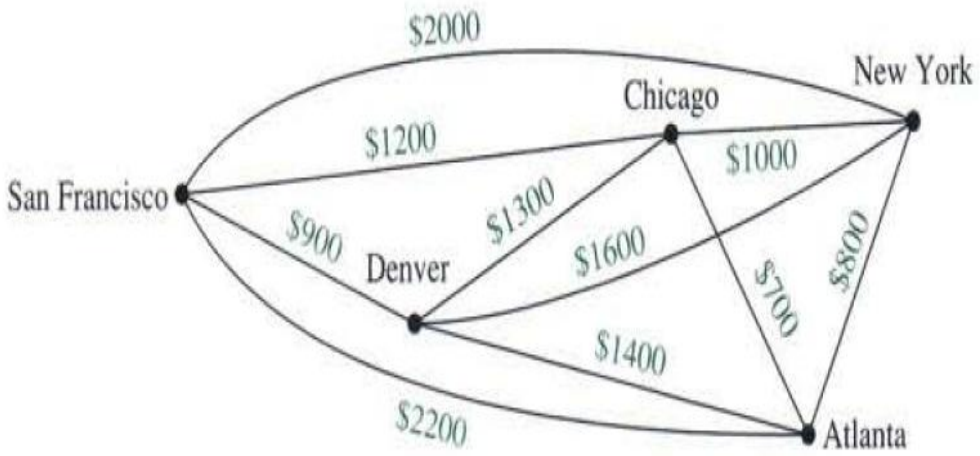
Apresenta-se **dois algoritmos** que podem ser utilizados para construir **árvores geradoras mínimas**.

Algorítmo de Prim (1957)

Dado um grafo G(V,E), o algoritmo de PRIM inclui um a um os vértices de uma árvore.

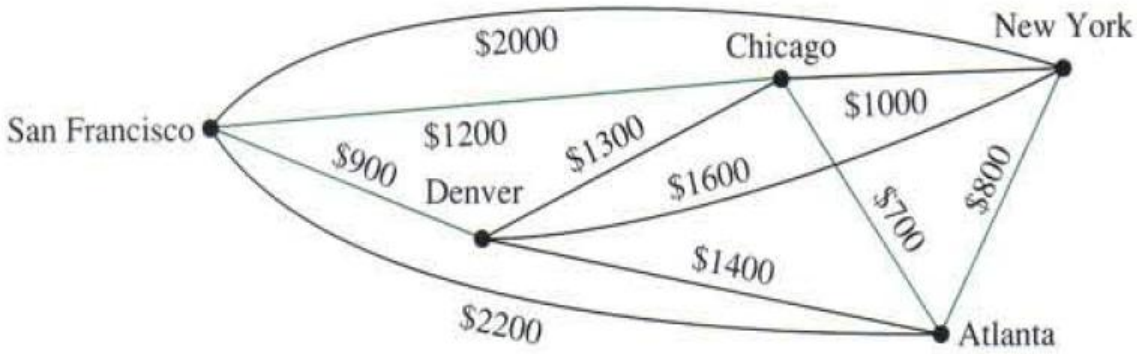
. Comece a construir a árvore geradora escolhendo-se um vértice do grafo;
. Sucessivamente, adicione à árvore arestas de peso mínimo que sejam incidentes a um vértice que já foi selecionado, ou seja, que já está na árvore e que possui uma extremidade fora deste conjunto de vértices. A aresta selecionada também não pode formar um ciclo simples com as arestas que já estão na árvore;
. **Parar o algoritmo quando tiverem sido adicionadas (n-1) arestas**.

Exemplo_a: Um companhia planeja construir uma infra estrutura de redes de comunicações conectando os seus cinco Data Centers. Cada par deste data centers pode ser interligado através de enlaces de fibra ótica que são alugados da operadora de serviços de telecomunicação.
A figura abaixo apresenta o grafo com as arestas interligando cada um dos data centers. A cada aresta está associado o custo mensal do enlace de comunicação.
Problema: Quais ligações devem ser realizadas a fim de garantir que existe um caminho entre quaisquer dois data centers, de modo que o custo total da rede seja minimizado ?



Solução:

Escolhe-se a aresta inicial como sendo a que interliga os vértices de **Chicago a Atlanta**.

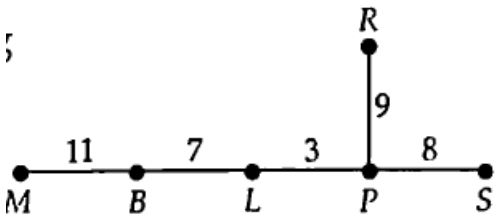


Choice	Edge	Cost
1	{Chicago, Atlanta}	$ 700
2	{Atlanta, New York}	$ 800
3	{Chicago, San Francisco}	$1200
4	{San Francisco, Denver}	$ 900
	Total:	$3600

Exemplo_b: A tabela abaixo fornece as distâncias entre seis cidades. Utilize o algoritmo de PRIM para calcular a árvore geradora mínima.

	Berlin	London	Moscow	Paris	Rome	Seville
Berlin	—	7	11	7	10	15
London	7	—	18	3	12	11
Moscow	11	18	—	18	20	27
Paris	7	3	18	—	9	8
Rome	10	12	20	9	—	13
Seville	15	11	27	8	13	—

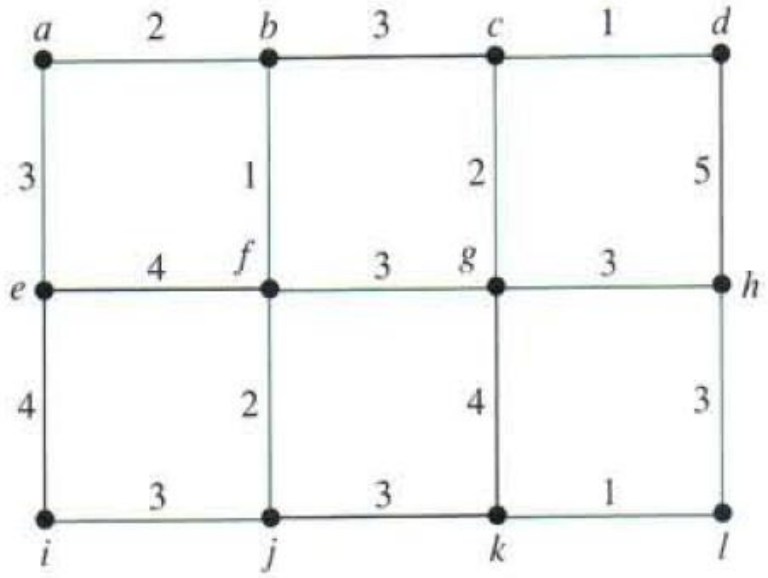
Resposta:



A figura abaixo apresenta o pseudo-código do algoritmo de Prim.

procedure *Prim*(G: weighted connected undirected graph with n vertices)
T := a minimum-weight edge
for $i := 1$ **to** $n - 2$
begin
  e := an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T
  $T := T$ with e added
end {T is a minimum spanning tree of G}

Exemplo_c: Dado o grafo com pesos abaixo discriminados, encontrar a árvore geradora mínima utilizando o algoritmo de Prim.



Solução:

Choice	Edge	Weight
1	{b, f}	1
2	{a, b}	2
3	{f, j}	2
4	{a, e}	3
5	{i, j}	3
6	{f, g}	3
7	{c, g}	2
8	{c, d}	1
9	{g, h}	3
10	{h, l}	3
11	{k, l}	1
	Total:	24

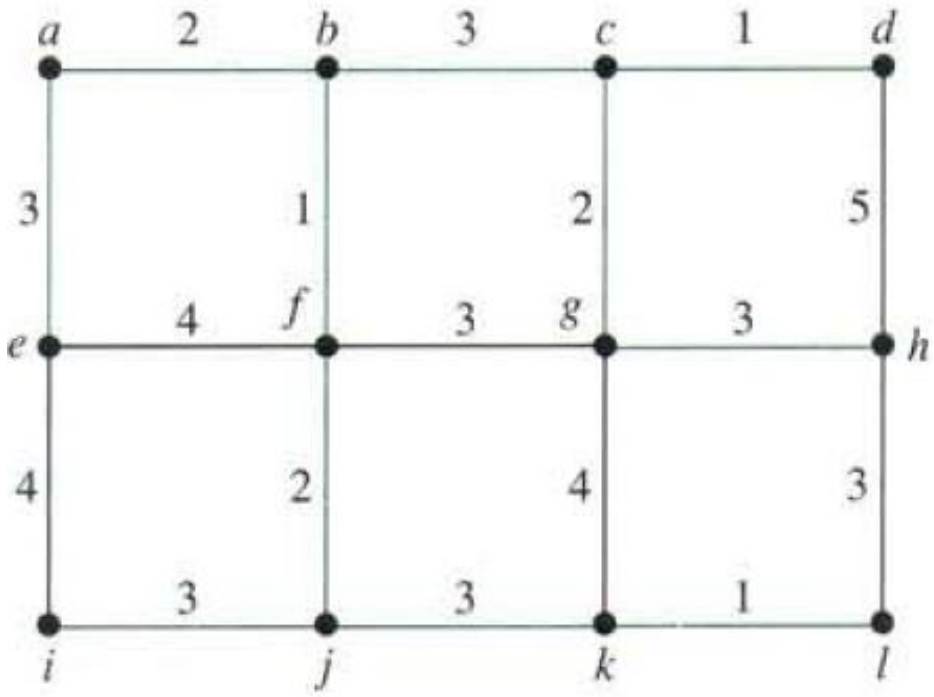
Algorítmo de Kruskal (1956)

Dado um grafo G(V,E): O algoritmo está voltado para formação da árvore através de inclusões de arestas, e não de vértices, como no algoritmo de PRIM.

. Escolha uma aresta no grafo com peso mínimo;
. Adicione sucessivamente arestas com peso mínimo que não formem um ciclo simples com as arestas já escolhidas;
. Parar depois de (n-1) arestas terem sido selecionadas.

Observação:
No algoritmo de Prim, as arestas de peso mínimo que são incidentes a um vértice já na árvore e que não formem um ciclo são escolhidas.
Enquanto que no algoritmo de Kruskal as arestas de peso mínimo que não são necessariamente incidentes a um vértice já na árvore e que não formem um ciclo são escolhidas.
No algoritmo de Prim as arestas precisam estar ordenadas para que o procedimento possa ser determinístico.

Exemplo_d: Utilize o algoritmo de Kruskal para calcular uma árvore geradora mínima no grafo com pesos apresentados abaixo.



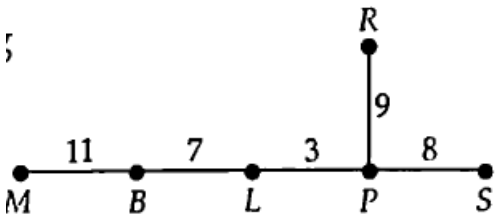
Solução:

Choice	Edge	Weight
1	{c, d}	1
2	{k, l}	1
3	{b, f}	1
4	{c, g}	2
5	{a, b}	2
6	{f, j}	2
7	{b, c}	3
8	{j, k}	3
9	{g, h}	3
10	{i, j}	3
11	{a, e}	3
	Total:	24

Exemplo_e: Utilize o algoritmo de Kruskal para calcular uma árvore geradora mínima do exemplo_b

	Berlin	London	Moscow	Paris	Rome	Seville
Berlin	—	7	11	7	10	15
London	7	—	18	3	12	11
Moscow	11	18	—	18	20	27
Paris	7	3	18	—	9	8
Rome	10	12	20	9	—	13
Seville	15	11	27	8	13	—

Resposta:



Trabalhos para casa

Pesquise sobre como implementar as seguintes árvores geradoras:

. Árvore geradora de mínimo diâmetro (AGMD)

. Árvore geradora mínima limitada em diâmetro (AGMLD)

. Árovre geradora mínima de um grafo bi-ponderado em arestas, onde Pi é o peso e Ci o custo da aresta.

Problema do Carteiro Chinês (1962): Este problema aborda a situação em que um carteiro precisa visitar todas as ruas de sua cidade, entregando correspondências, e retornando ao local de origem. Para isto precisa-se calcular o menor caminho a ser percorrido pelo carteiro.

Pode-se modelar o problema como um grafo onde os vértices representam as ruas que precisam ser visitadas, e as arestas representam as distâncias para se deslocar entre as ruas.

Se o problema for modelado como um grafo euleriano, a solução é dada através do algoritmo de Fleury.

Porém, se não for o caso, o algoritmo precisará calcular o caminho mais curto, passando, por todos os vértices.

Observação: Pode ser demonstrado que, para encontrar uma árvore geradora mínima de um grafo com **e** arestas e **v** vértices, o algoritmo de Kruskal pode ser executado usando O(e*loge) operações.

O algoritmo de Prim pode ser executado usando O(e*logv) operações.

Portanto, é preferível usar o algoritmo de Kruskal para **grafos que são esparsos**, isto é, nos quais **e** é muito pequeno quando comparado a C(v,2) = v(v-1)/2, o número total de arestas possíveis em um grafo não orientado com v vértices.

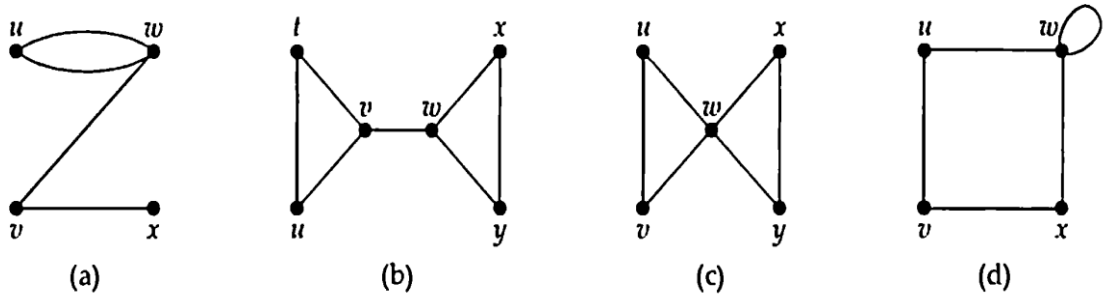
Caso contrário, há pouca diferença na complexidade destes dois algoritmos.

Conectividade de grafos

Para várias aplicações que são modeladas através de grafos é necessário considerar a conectividade (ou conexidade) entre vértices com mais detalhes. Ou seja, as relações que envolvem os vértices formam uma estrutura contínua, em que os vértices são todos interligados através das relações.

A conexidade possui importante aplicações nas comunicações, planejamento da produção, logística, dentre outros. Por exemplo, em uma rede de telecomunicações é importante identificar/simular como os enlaces (ou arestas) de comunicação podem ser rompidos, porém, sem afetar a comunicação entre dois pontos (ou vértices).

Exemplo: Dado os seguintes abaixo:



O grafo a.) pode ser dividido em dois componentes pela remoção de uma das arestas vw e vx. A remoção de qualquer das duas arestas torna o grafo desconexo.

O grafo b.) pode também ser desconexo pela remoção da aresta vw.

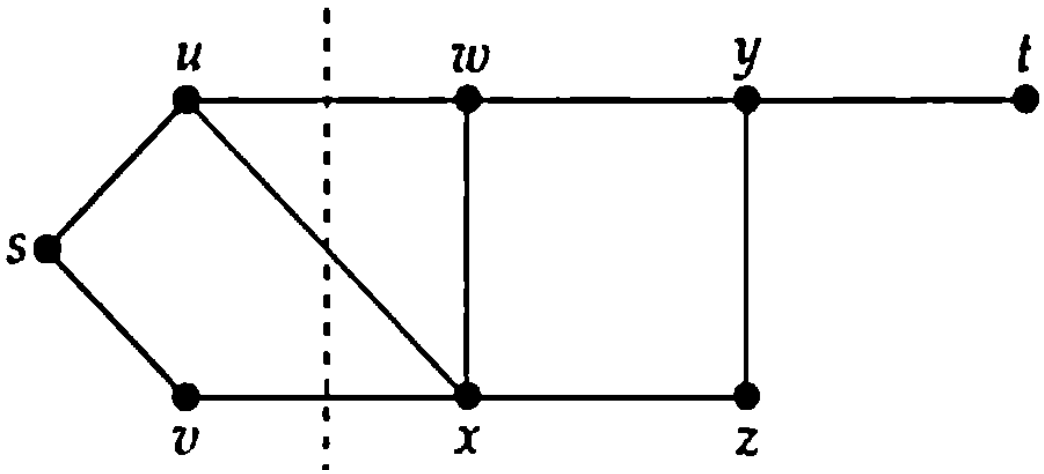
O grafo c.) pode se tornar desconexo pela remoção das arestas uw e vw.

O grado d.) pode também se tornar desconexo pela remoção de duas arestas: uw e wx.

Conectividade de aresta: A conectividade de aresta, **$\tilde{\lambda}$ (G),** de um grafo conexo corresponde ao menor número de arestas em que as suas remoções provocam a desconexão de G.

Por exemplo, no exercício anterior os grafos a.) e b.) possuem ambos conectividade de aresta igual à 1.

Conjunto de cortes (edge cutset): representa o conjunto de arestas não redundantes que se todas retiradas tornam o grafo G desconexo. Por exemplo, no grafo da figura abaixo, os conjuntos: {uw,ux,vx}, {wy, xz} ou {yt} representam 3 conjuntos de corte distintos.

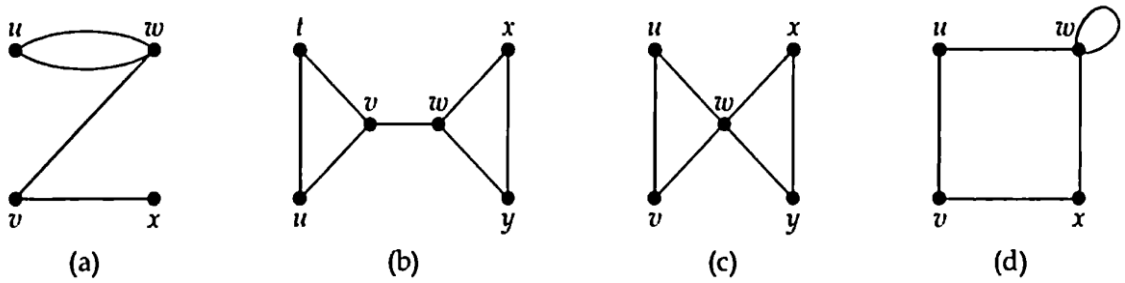


Conectividade de vértice: conectividade ou conectividade de vértices de um grafo conexo G, representada por **K(G),** corresponde ao menor número de nós cuja remoção torna o grafo G desconexo. Esta definição não é válida para grafos completos, já que não pode-se tornar um grafo completo desconexo removendo-se os respectivos vértices.

Portanto, a conectividade de um grafo completo Kn é dada por:

K(Kn) = n-1; para n>=3.

.



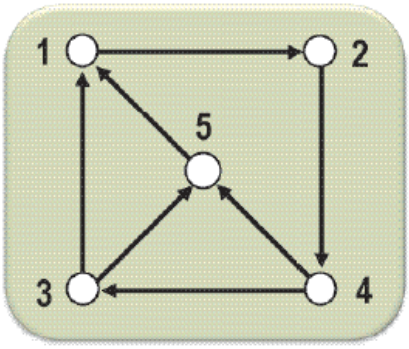
Os grafos a.) , b.) e c.) podem se tornar desconexos se removermos apenas um único vértice.

O grafo d.) pode se tornar desconexo se removermos dois vértices não adjacentes.

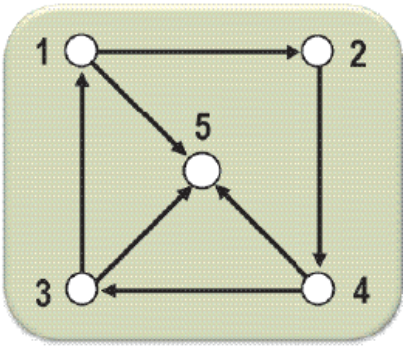
Vértices fortemente conectados: Dois vértices i e j estão fortemente conectados em um grafo direcionado G, se existe caminho direcionado de i para j e de j para i em G.

Dois vértices i e j estão fortemente conectados em um grafo não direcionado G, se existem dois caminhos distintos em arestas de i para j em G.

Vértices fracamente conectados: este conceito é exclusivo para grafos direcionados. Dois vértices i e j são fracamente conectados em um grafo direcionado G, se existe apenas um caminho direcionado de i para j ou de j para i em G

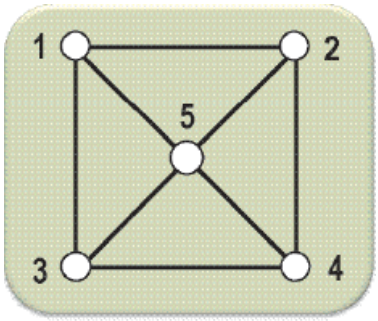


(1) Grafo fortemente conexo



(2) Grafo fracamente conexo

K-Conexidade: um grafo é dito k-conexo quando nele existem pelo menos k caminhos disjuntos em vértices ligando cada par de vértices. Um grafo dito 2-conexo é um grafo fortemente conexo.



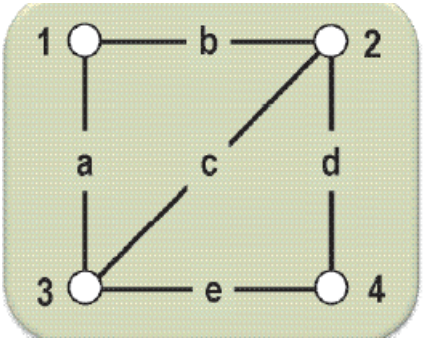
(1) Grafo 3-conexo

Ciclo em um grafo: um ciclo em um grafo implica a existência de pelo menos dois caminhos disjuntos para os vértices de G que fazem parte do ciclo.

Matriz de ciclos: Dado um grafo G = (N,M) com p ciclos distintos, uma matriz de ciclos de G, B = bij , é uma matriz tal que:

. bij = 1, se a aresta j pertence ao ciclo i;

. bij = 0, caso contrário.



	a	b	c	d	e
C_1	1	1	1	0	0
C_2	0	0	1	1	1
C_3	1	1	0	1	1

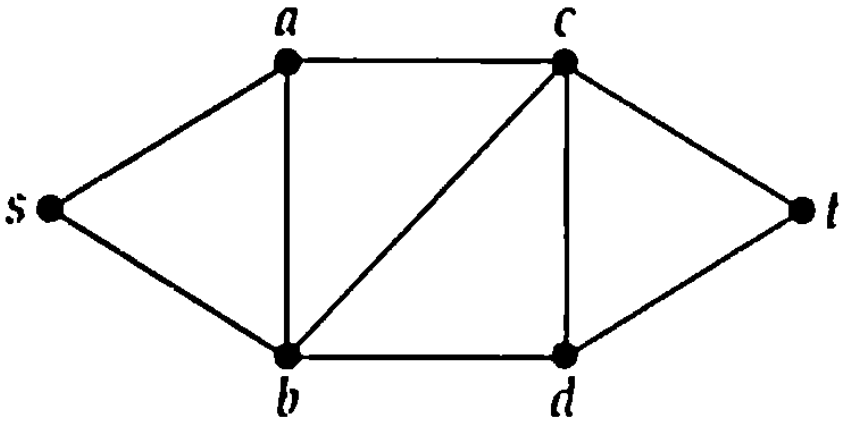
Conjunto de corte de vértices (vertex cutset): Dado um grafo G, o conjunto de corte de vértices representa o conjunto S de tal forma que a remoção de todos os vértices de S tornam o grafo G desconexo; e a remoção de alguns (mas não de todos) os vértices de S,não tornam G desconexo.

Vértices e arestas disjuntas: Dado um grafo G conexo. Considere s e t como vértices de G. Dois ou mais caminhos de s à t são **disjuntos por arestas** se estes caminhos não tiverem arestas em comum. São considerados como **disjuntos por vértices** se não tiverem vértices em comum.

Dado o grafo G abaixo, os conjuntos {sact} e {sbdt} são ambos disjuntos em arestas e em vértices para o caminho s-t.

Os conjuntos {sact} e {sbct} são ambos não disjuntos em arestas e em vértices .

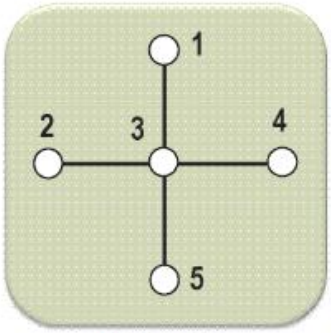
Os conjuntos {sact} e {sbcdt} são disjuntos em arestas, mas não em vértices, já que possuem o vértice c em comum.



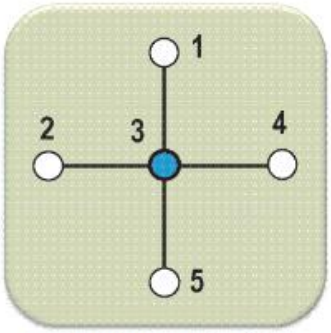
Cobertura de vértices: Dado um grafo G, denomina-se cobertura de vértices um conjunto V´ de vértices, tal que toda aresta de G tem pelo menos uma extremidade em V´.

Cobertura de arestas: Dado um grafo G, denomina-se um conjunto E´ de arestas tal que todo vértice de G seja incidente a pelo menos uma aresta de E´.

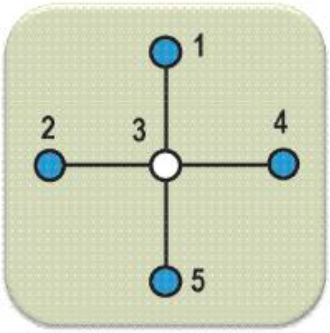
A figura abaixo apresenta dois exemplos de cobertura de vértices e de arestas, respectivamente.



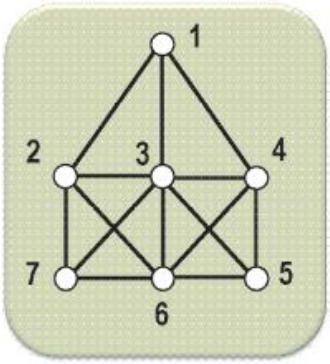
(1) Grafo *G*



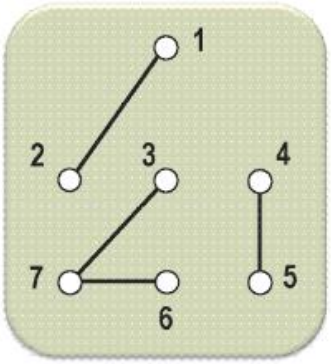
(2) Cobertura $C_1 = \{3\}$



(3) Cobertura $C_2 = \{1,2,4,5\}$



(4) Grafo *G*



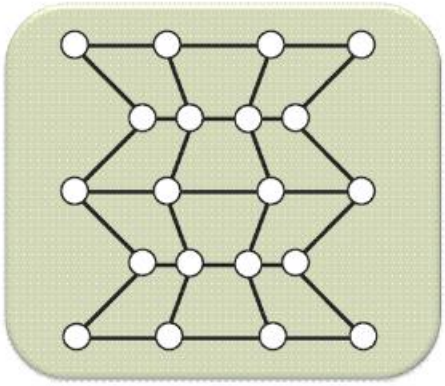
(5) Cobertura de arestas $E = \{(1\text{-}2),(3\text{-}7),(6\text{-}7),(4\text{-}5)\}$

Cobertura mínima em caminhos de G: corresponde a determinar a menor coleção possível de caminhos disjuntos que percorra todos os vértices de G.

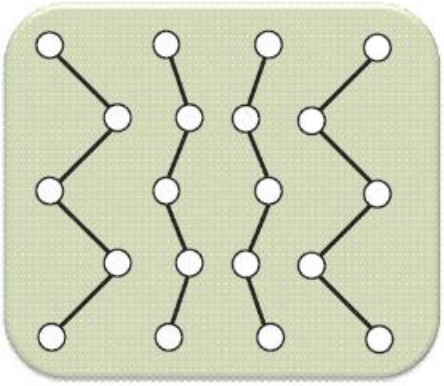
Cobertura em caminhos: Dado um grafo G , uma cobertura em caminhos em G corresponde a um conjunto de caminhos disjuntos em vértices, que passam por todos os vértices.

Cobertura em ciclos: Dado u grafo G , uma cobertura em ciclos em G corresponde a um conjunto de ciclos disjuntos em vértices, que passam por todos os vértices.

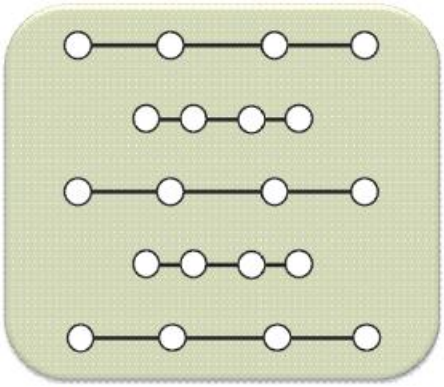
A figura abaixo apresenta duas coberturas em caminhos disjuntos do grafo G (2 e 3), e uma cobertura de ciclos (4) do mesmo grafo.



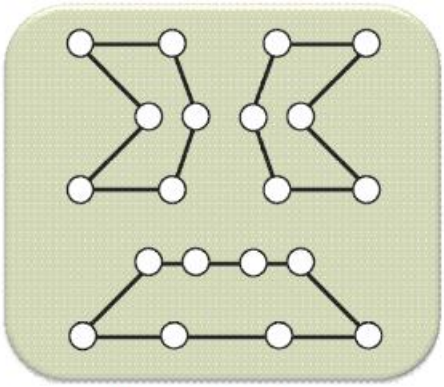
(1) Grafo G



(2) Primeira cobertura em caminhos



(3) Segunda cobertura em caminhos

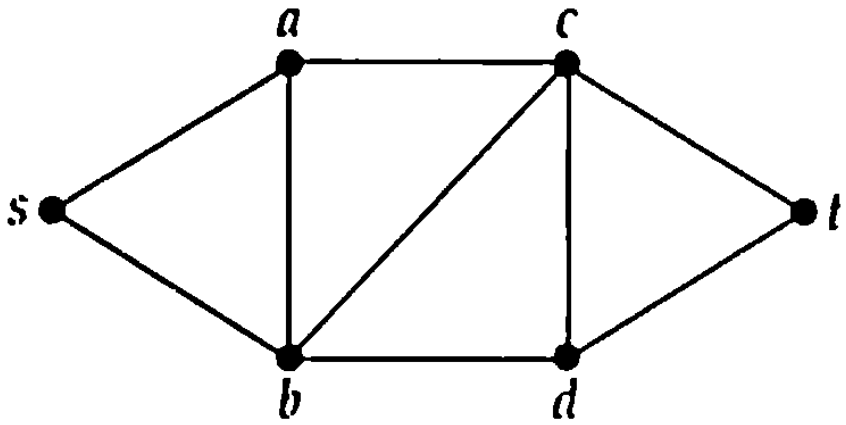


(4) Cobertura em ciclos

Teorema de Menger:

(considerando as arestas) Dado que G é um grafo conexo, e s e t vértices deste grafo. Então o número máximo de caminhos de arestas disjuntas para s-t é igual ao número mínimo de arestas que separam s de t.

Exemplo: Vide a figura abaixo.



(considerando os vértices) Dado que G é um grafo conexo, e considere que s e t são vértices não adjacentes de G. Então o número máximo de vértices disjuntos do caminho s –t é igual ao número mínimo de vértices separando s de t.

Dado um grafo G, quando encontra-se um valor K que corresponde tanto ao número de caminhos disjuntos de arcos de s para t, quanto ao número de arcos separando s de t, então o valor de k corresponde ao número máximo de arcos de caminhos s-t disjuntos, e o número mínimo de arcos separando s de t.

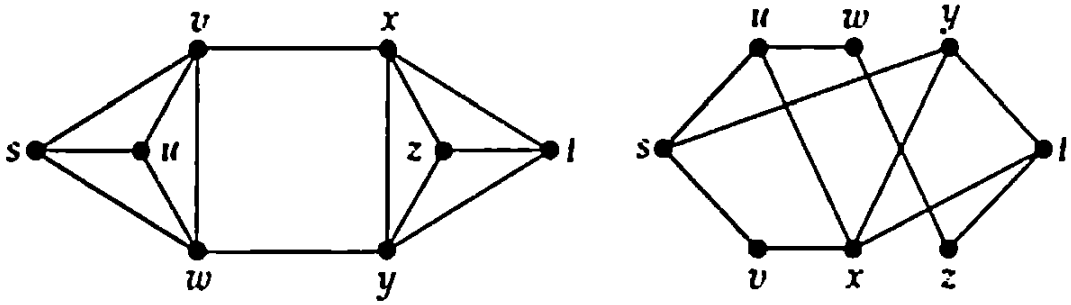
Exercício: **Transporte de valores**. Os dois grafos abaixo representam as possíveis rotas de duas cidades que dois caminhões, de uma empresa que transporta dinheiro para abastecer agências bancárias, utilizam para se deslocar através das ruas das cidades.

Considere que os vértices representam os pontos onde as agências estão localizadas, e as arestas representam as ruas que os caminhões precisam utilizar para se deslocar entre duas agências.

Considere que, por motivo de segurança, a empresa que realiza o transporte de valores, abastece uma agência com dois caminhões que percorrem obrigatoriamente rotas distintas.

Dado o grafo das duas cidades abaixo, determine quantas rotas distintas são possíveis para cada uma das cidades?

Obs: para as rotas distintas não existe(m) arestas em comum.



Exercício_implementação: Implemente o algoritmo que resolve o problema do transporte de valores através de uma região metropolitana, para atender a demanda do exercício anterior.

Exercício: Nos domingos em que ocorrem o jogo do JEC na cidade de Joinville, os torcedores que precisam se deslocar da zona Norte (utilize como referência a UDESC) para a Arena, se deparam com um grande congestionamento e muitos deles acabam chegando atrasados ao estádio para assistir ao jogo.

A Prefeitura de Joinville está fazendo uma campanha a fim de conscientizar o torcedor a buscar rotas alternativas entre a zona norte e a Arena, para assim tentar minimizar o problema de congestionamento nas vias próximas ao estádio.

Dado o problema exposto acima, cada uma das equipes, utilizando-se de um mapa da cidade de Joinville (utilize o Google maps) , modelará através de grafos o conjunto de ruas (arestas) e bairros (vértices) que permitem interligar a zona norte a Arena.

Após a modelagem utilizando grafos, a equipe irá determinar quantas rotas alternativas existem entre os dois pontos.

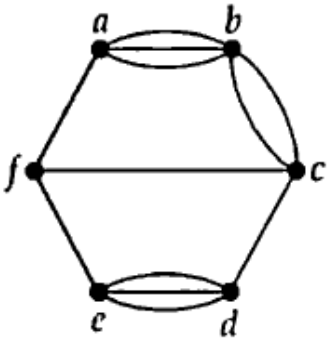
Como fica a questão do fluxo máximo permitido nas ruas (arestas)? O fluxo máximo interfere no problema do congestionamento ?

Coloração de arestas

Dado que G seja um **grafo sem loops**, uma coloração K-arestas consiste em uma atribuição de no máximo k cores às arestas de G de tal forma que qualquer duas arestas que se encontram em um vértice possuem cores diferentes.

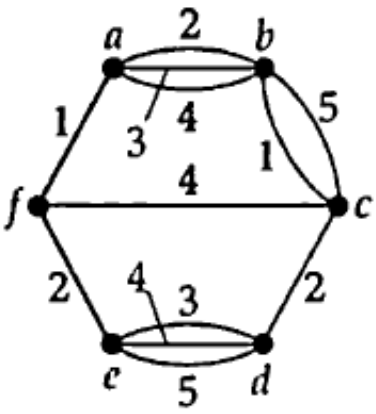
O índice cromático **X(G)** corresponde ao menor número K para o qual as arestas de um grafo podem ser coloridas.

Exercício: Dado o grafo G abaixo, determinar quantas cores (k) serão necessárias a fim de colorir todas as arestas do grafo.



Resposta:

O índice cromático do grafo é 5, pois existe um vértice (b) com grau 5.



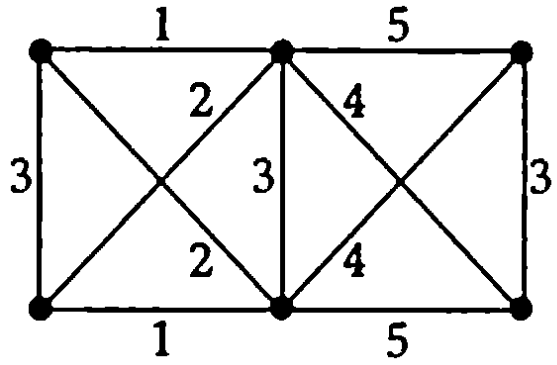
Dado um determinado grafo G, a determinação do índice cromático é dado pelos limites superiores e inferiores. Dado o exemplo abaixo, as arestas do grafo podem ser coloridas com 5 cores diferentes. Então tem-se que:

X(G) <= 5.

Por outro lado o grafo G não pode ser colorido com menos do que 5 cores, pois G contêm um vértice com 5 arestas (grau 5). Então tem-se que;

X(G) >= 5

Combinando estas duas inequações tem-se que **X(G) =5**.



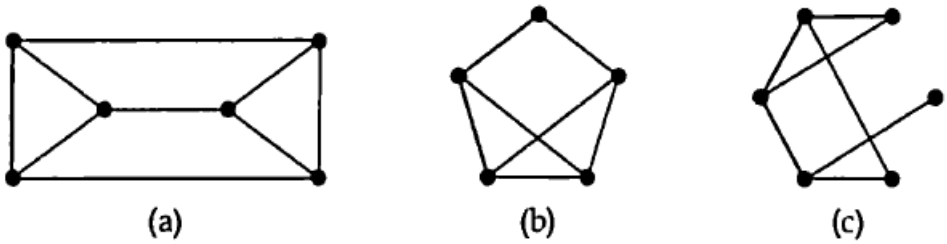
Teorema de Vizing:

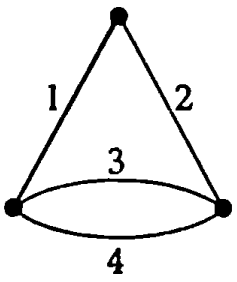
Dado um grafo G simples, cujo grau máximo de um vértice é de d, e h corresponde ao número máximo de arestas que unem um par de vértices.

Então:

$$d <= X(G) <= d + h$$

Exercícios: determine o número cromático para cada um dos grafos abaixo. :





Resposta:

a.)3 ; b.) 4 ; c.) 3

Aplicando-se o teorema de Vizing no grafo d.) temos que d=3 e h=2. Portanto, 3 <= X(G) <= 3+2

Teorema de Shannon:

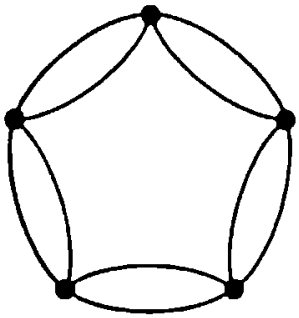
Dado um grafo G com o grau máximo de um vértice dado por **d.**

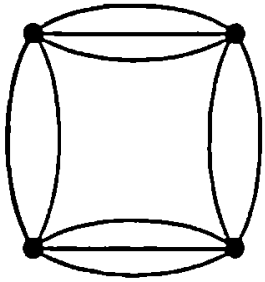
Então:

$$d <= X(G) <= 3d/2 \text{ ; se d é par}$$

$$d <= X(G) <= (3d-1)/2 \text{; se d é impar}$$

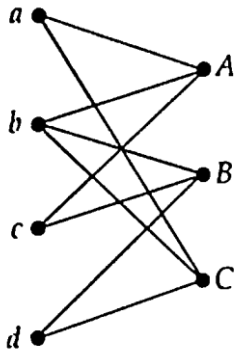
Exercício: Dado os dois grafos abaixo, utilize os teoremas de Vizing e de Shannon para calcular os limites inferiores e superiores de X(G).





Exercício: Escalonamento de horários para atendimento aos alunos do TEG. Quatro alunos de TEG necessitam tirar suas dúvidas sobre a disciplina e precisam agendar horários para que os três monitores da disciplina possam atendê-los. O atendimento é individual por aluno. Cada um dos monitores domina apenas partes da disciplina. No grafo bipartido abaixo consta os possíveis relacionamentos (competências) entre os alunos (a,b,c,d) e os monitores (A,B,C).

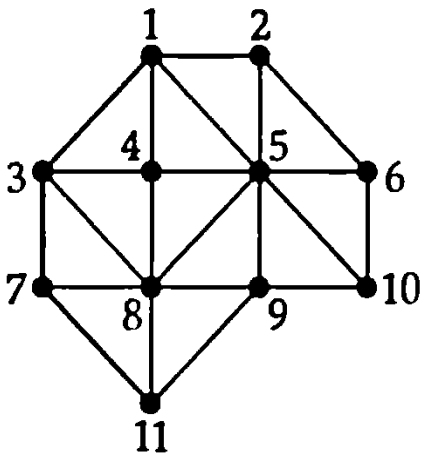
Utilize um grafo para modelar os relacionamentos entre os alunos e os monitores, e utiliza as cores para identificar quantos horários distintos serão necessários para atender aos alunos.



Implementação: implemente o exercício anterior.

Implementação: implemente um algoritmo que dado um grafo G determine o número cromático de arestas.

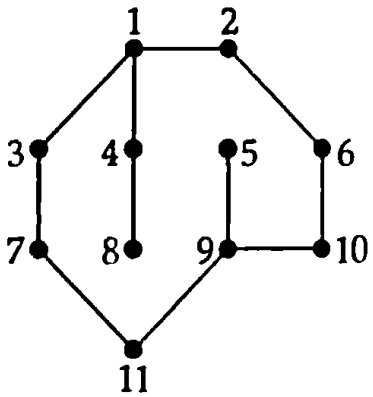
Implementação: O grafo G representa o mapa de um estado que é composta de onze cidades (vértices) interligadas por 22 estradas (arestas). O departamento de transporte precisa contratar duas empresas de ônibus, onde cada uma delas vai ficar responsável pelo transporte em pelo menos 10 rodovias. Cada cidadão pode se deslocar entre duas cidades quaisquer utilizando apenas 1 empresa.

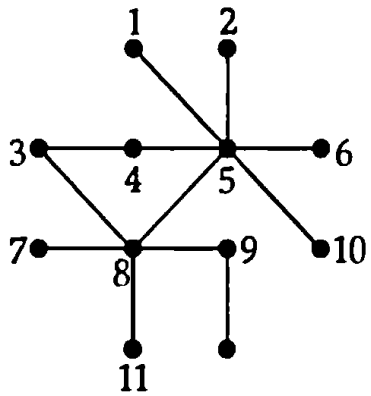


Os dois subgrafos abaixo ilustram possíveis rotas para as duas companhias.

a.) Implemente um algoritmo que gere dois possíveis subgrafos;

b.) Suponha que uma companhia não pode operar dois trechos (interligar três cidades), com o mesmo motorista, na sequencia. Quantos motoristas serão necessárias neste caso para cobrir todas as cidades de um mapa ?





Apêndice A – Algorítmo de Dikjstra

Este apêndice apresenta outra maneira de se analisar o algoritmo de Dijkstra

Suponha um **grafo simples, conexo e com pesos positivos em suas arestas**. O algoritmo do caminho mínimo encontra o caminho com peso mínimo entre dois nós **a e v** de um grafo.

O algoritmo inicia associando o valor 0 ao vértice a e o valor ∞ aos outros vértices.

Utiliza-se a indicação: $\mathbf{L_o\,(a) = 0}$ **e** $\mathbf{L_o\,(v) = \infty}$ para estes **rótulos** antes de ocorrer qualquer iteração. Atenção o subscrito 0 indica o **número de iterações**.

$\mathbf{S_k}$ = representa o conjunto depois de K iterações do procedimento de rotulação.

Os rótulos representam os comprimentos dos caminhos mais curtos de **a** aos outros vértices.

Iniciação com $\mathbf{S_k = 0}$, onde o conjunto $\mathbf{S_k}$ é formado a partir de $\mathbf{S_{k-1}}$, adicionando um vértice **u** que não está em $\mathbf{S_{k-1}}$ como menor rótulo.

Uma vez que **u** seja adicionado à $\mathbf{S_k}$, atualiza-se o rótulo de todos os vértices que não estão em $\mathbf{S_k}$, de modo que $\mathbf{L_k\,(v)}$,

que representa o rótulo do vértice **v** no estágio **k**, corresponde ao comprimento do caminho mais curto de **a** à **v** que contêm apenas vértices em $\mathbf{S}_{\boldsymbol{k}}$. Isto é, vértices que já estavam no conjunto especial juntamente com **u.**

Seja **v** um vértice que não está em Sk, e Lk(v) é o comprimento do caminho mais curto de **a** a **v** que contem apenas vértices em Sk.

Portanto, o caminho mais curto de **a** à **v** que contem apenas elementos em Sk **ou** é um caminho mais curto de **a** a **v** que contem apenas elementos em Sk-1 (isto é os nós especiais sem incluir o nó u) **ou** é um caminho mais curto de **a** a **u** no (k-1) estágio com a aresta **(u,v)** adicionada.

Este procedimento é iterado adicionando-se sucessivamente vértices ao conjunto especial até que o nó de destino (final) seja adicionado.

Quando o nó final for adicionado ao conjunto de nós **S**, o seu respectivo rótulo corresponderá ao comprimento de um caminho mais curto (ou menor custo) do nó origem até o nó destino.

Podemos expressar matematicamente o algoritmo de Dikjstra da seguinte maneira:

$$\left(\mathbf{L}_{\boldsymbol{k}} (\mathbf{a}, \mathbf{v}) \right) = \mathbf{min}\{ \left(\mathbf{L}_{\boldsymbol{k-1}} (\mathbf{a}, \mathbf{v}) \right), \ \left(\mathbf{L}_{\boldsymbol{k-1}} (\mathbf{a}, \mathbf{u}) \right) + \mathbf{w}(\mathbf{u}, \mathbf{v})\}$$

O algoritmo de Dijkstra está apresentado na figura abaixo.

procedure *Dijkstra*(G: weighted connected simple graph, with all weights positive)
{G has vertices $a = v_0, v_1, \ldots, v_n = z$ and weights $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for $i := 1$ **to** n
    $L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set}
while $z \notin S$
begin
    $u :=$ a vertex not in S with $L(u)$ minimal
    $S := S \cup \{u\}$
    for all vertices v not in S
        if $L(u) + w(u, v) < L(v)$ **then** $L(v) := L(u) + w(u, v)$
    {this adds a vertex to S with minimal label and updates the labels of vertices not in S}
end {$L(z) =$ length of a shortest path from a to z}