ENG. INFORMÁTICA E DE COMPUTADORES

Algoritmos e Estruturas de Dados

(parte 16)

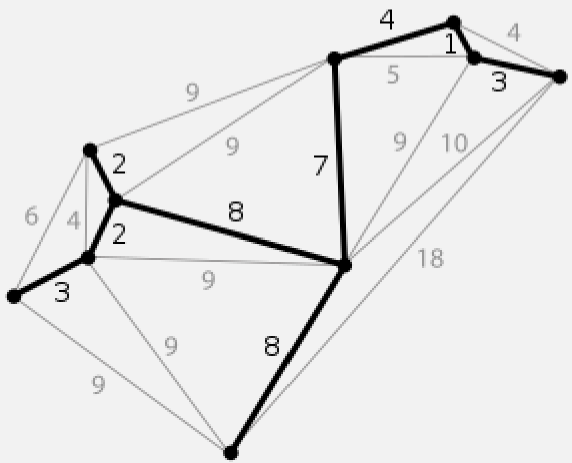
2º Semestre 2021/2022

Instituto Superior de Engenharia de Lisboa
Paula Graça

ALGORÍTMOS DE PRIM

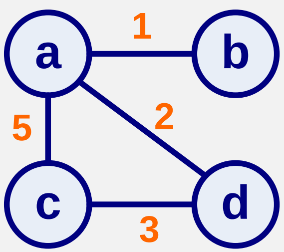
- Árvore de Abrangência Mínima (AAM) ou Minimum Spanning Tree (MST)

  - Permite calcular a forma mais económica de interligar um conjunto de vértices de um grafo pesado
  - O grafo tem que ser conexo, senão a árvore de abrangência não se consegue determinar

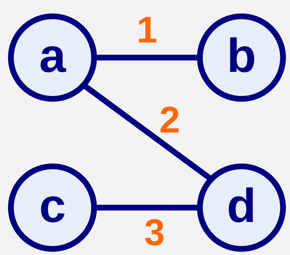


ÁRVORE DE ABRANGÂNCIA MÍNIMA

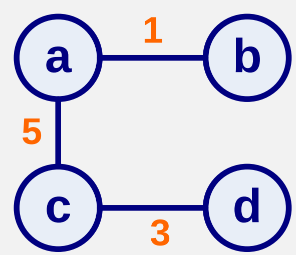
- Árvore de Abrangência
  - É uma árvore (um sub-grafo acíclico) que contém todos os vértices do grafo original
- Árvore de Abrangência Mínima
  - É uma árvore de abrangência na qual a soma dos pesos de todas as suas arestas é mínima



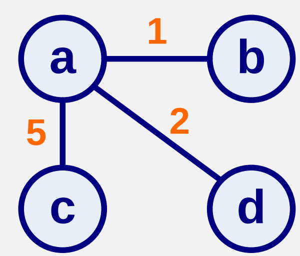
GRAFO



AAM = 6



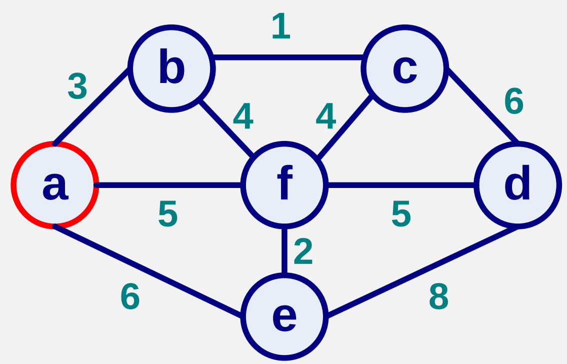
AA = 9



AA = 8

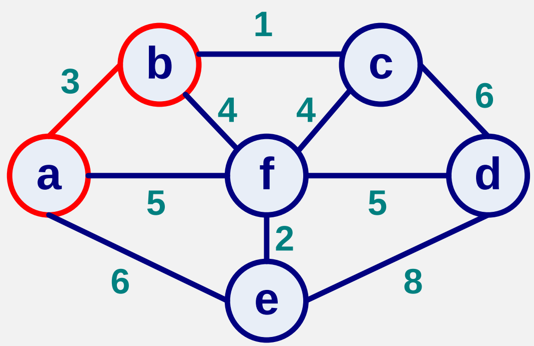
ALGORITMO DE PRIM

- A árvore de abrangência mínima (AAM) inicial, consiste num vértice simples escolhido arbitrariamente do conjunto de vértices V do grafo G



- Em cada iteração, a árvore é expandida de forma gananciosa (*greedy*) acrescentando-lhe o vértice menor custo ainda não adicionado à AAM

ALGORITMO DE PRIM

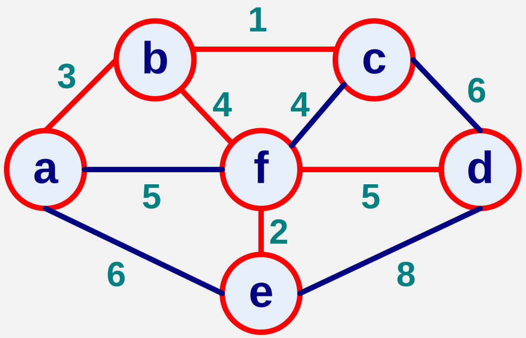


- O vértice mínimo é escolhido de entre os vértices adjacentes daqueles que fazem parte da AAM

- Se existem dois ou mais vértices com igual custo, então escolhe-se um deles arbitrariamente

ALGORITMO DE PRIM

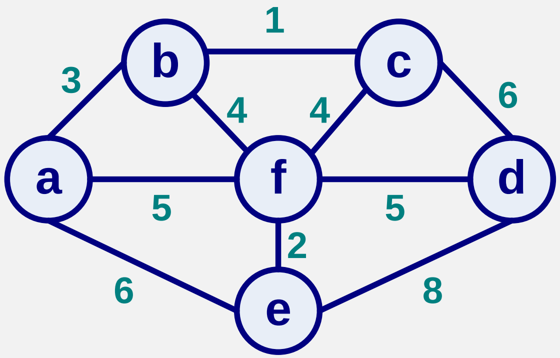
- O algoritmo pára quando todos os vértices do grafo tiverem sido introduzidos na AAM em construção



- O número total de iterações é n-1, sendo n o número de vértices do grafo

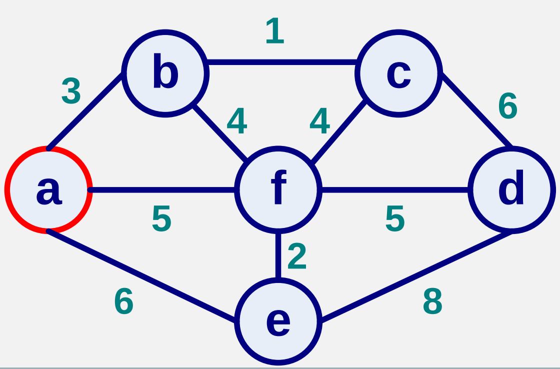
ALGORITMO DE PRIM

- Exemplo passo a passo do algoritmo de Prim, considerando o seguinte grafo não dirigido:

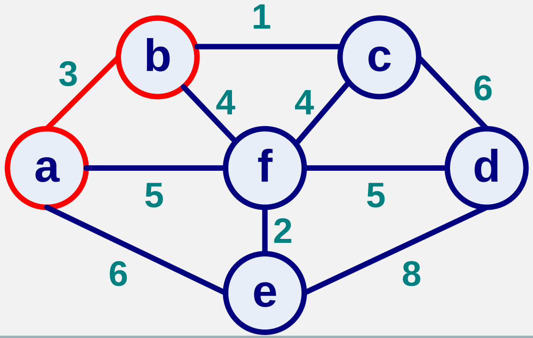
A_{AAM} Grafo	V_{AAM} Vértices AAM	$V-V_{AAM}$ Restantes Vértices
		

ALGORITMO DE PRIM

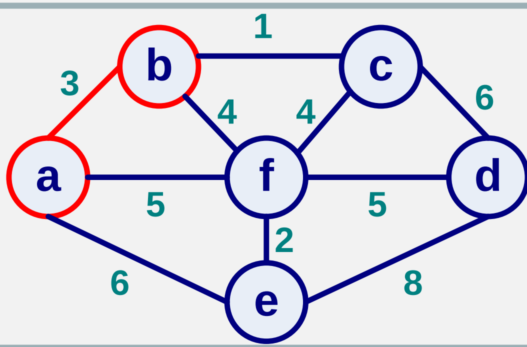
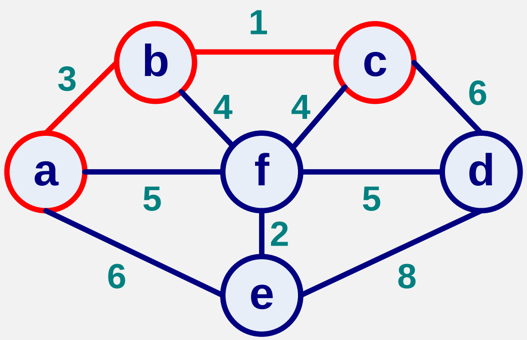
- Primeira iteração através da técnica *greedy*:

A_{AAM} Grafo	V_{AAM} Vértices AAM	$V\text{-}V_{AAM}$ Restantes Vértices
	a(-, 0)	**b(a, 3)** e(a, 6) f(a, 5)

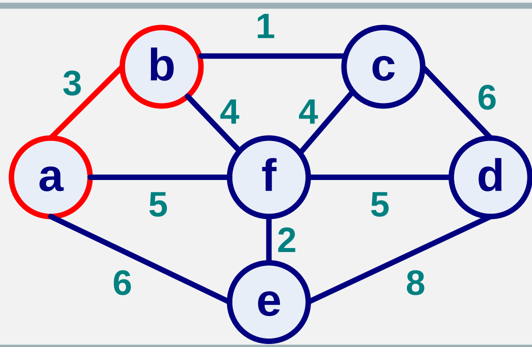
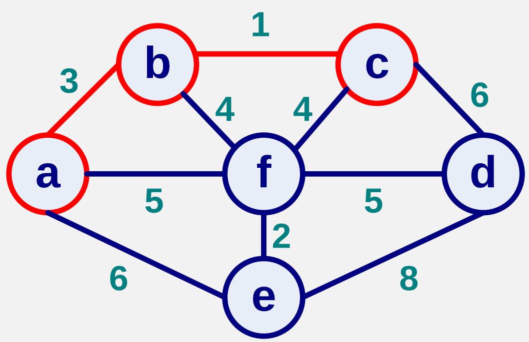
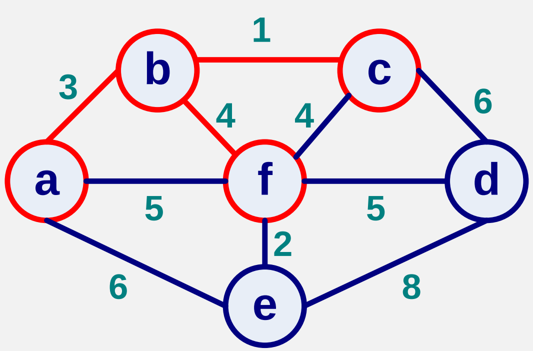
ALGORITMO DE PRIM

A_{AAM}	V_{AAM}	$V\text{-}V_{AAM}$
	a(-, 0) b(a, 3)	**c(b, 1)** e(a, 6) f(b, 4)

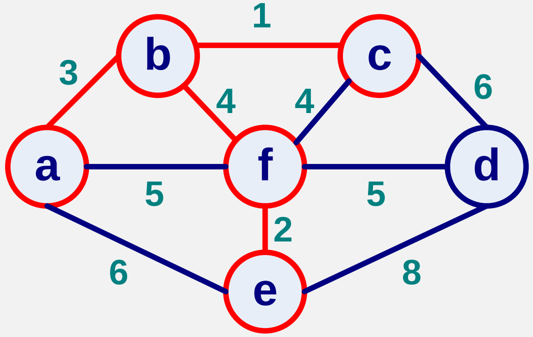
ALGORITMO DE PRIM

<table>
<tr><th>A_{AAM}</th><th>V_{AAM}</th><th>$V\text{-}V_{AAM}$</th></tr>
<tr><td>
</td><td>a(-, 0)
b(a, 3)</td><td>c(b, 1) e(a, 6) f(b, 4)</td></tr>
<tr><td>
</td><td>a(-, 0)
b(a, 3)
c(b, 1)</td><td>d(c, 6) e(a, 6) f(b, 4)</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

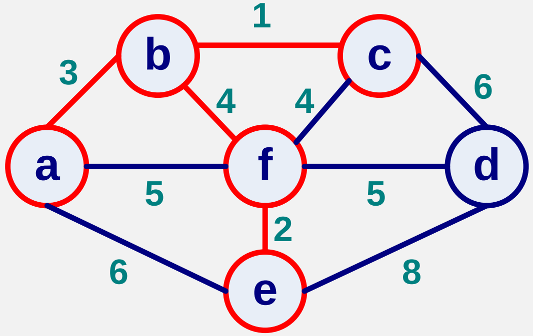
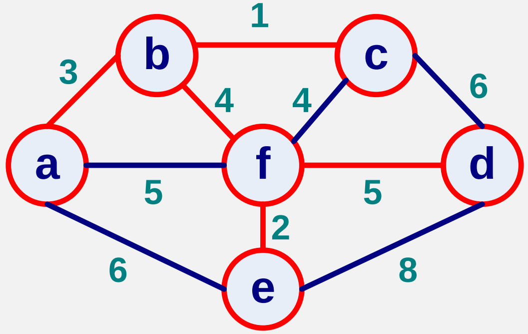
ALGORITMO DE PRIM

A_{AAM}	V_{AAM}	$V\text{-}V_{AAM}$
	a(-, 0) b(a, 3)	**c(b, 1)** e(a, 6) f(b, 4)
	a(-, 0) b(a, 3) c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**
	a(-, 0) b(a, 3) c(b, 1) f(b, 4)	d(f, 5) **e(f, 2)**

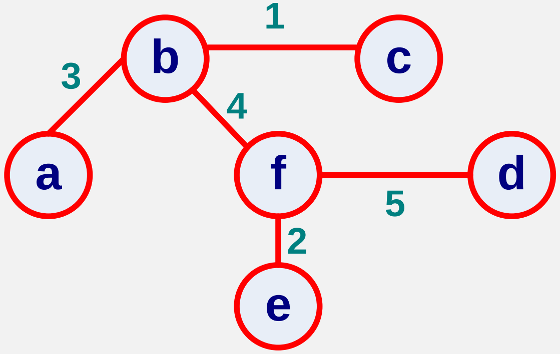
ALGORITMO DE PRIM

A_{AAM}	V_{AAM}	$V\text{-}V_{AAM}$
	a(-, 0) b(a, 3) c(b, 1) f(b, 4) e(f, 2)	d(f, 5)

ALGORITMO DE PRIM

A_{AAM}	V_{AAM}	$V\text{-}V_{AAM}$
	a(-, 0) b(a, 3) c(b, 1) f(b, 4) e(f, 2)	**d(f, 5)**
	a(-, 0) b(a, 3) c(b, 1) f(b, 4) e(f, 2) d(f, 5)	

ALGORITMO DE PRIM

A_{AAM}	V_{AAM}	$V\text{-}V_{AAM}$
	a(-, 0) b(a, 3) c(b, 1) f(b, 4) e(f, 2) d(f, 5)	

ALGORITMO DE PRIM

// Implementa o algoritmo de Prim para a construção da AAM

// Entrada: um grafo pesado ligado G = (V, A)

// Saída: A_{AAM}, conjunto de arestas que compõe a AAM de G

Prim(G)

V_{AAM} = {a} **// conjunto de vértices da AAM iniciado com o vértice "a"**

A_{AAM} = Ø **// conjunto de arestas da AAM iniciado a vazio**

for i = 1 to |V| - 1 do

encontrar a aresta mínima a* = (v*, u*) entre todas as arestas (v, u) tais que v ∈ V_{AAM} e u ∈ V – V_{AAM}

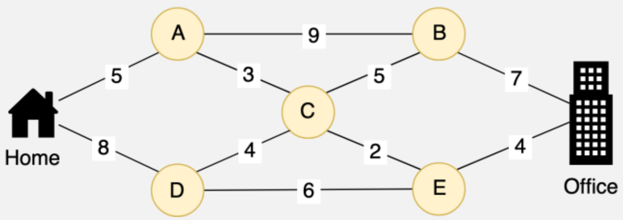
V_{AAM} = V_{AAM} U {u*}

A_{AAM} = A_{AAM} U {a*}

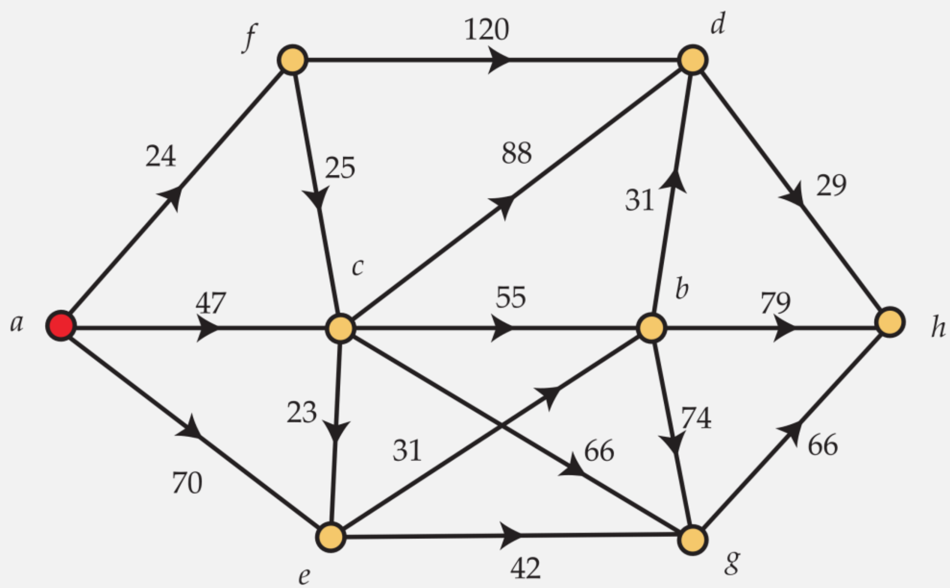
return A_{AAM}

ALGORITMO DE DIJKSTRA

- Caminho mais curto com fonte única



  - Calcula o caminho mais curto com início num determinado vértice fonte, até todos os outros vértices
  - Aplica-se a grafos com pesos não negativos



ALGORITMO DE DIJKSTRA

- Trata-se do melhor dos algoritmos e o mais conhecido para o problema do caminho mais curto com fonte única,

- A restrição é perfeitamente possível no contexto de redes de transportes, onde as arestas representam normalmente distâncias ou tempos médios de percurso

- Em aplicações onde as arestas apresentam pesos negativos, o algoritmo não funciona corretamente

ALGORITMO DE DIJKSTRA

- Descrição do algoritmo de Dijkstra:
  - A partir do vértice inicial (fonte), calculam-se as distâncias desde o vértice fonte até aos vértices adjacentes que ainda não foram selecionados como fazendo parte do conjunto de vértices dos caminhos mais curtos
  - As distâncias calculadas são registadas nos vértices adjacentes respetivos
  - Seleciona-se o vértice de menor distância de entre todos os vértices que ainda não fazem parte do conjunto dos caminhos mais curtos
  - Adiciona-se o vértice selecionado ao conjunto dos caminhos mais curtos
  - Repete-se o processo para o vértice selecionado até que todos os vértices façam parte do conjunto dos caminhos mais curtos

ALGORITMO DE DIJKSTRA

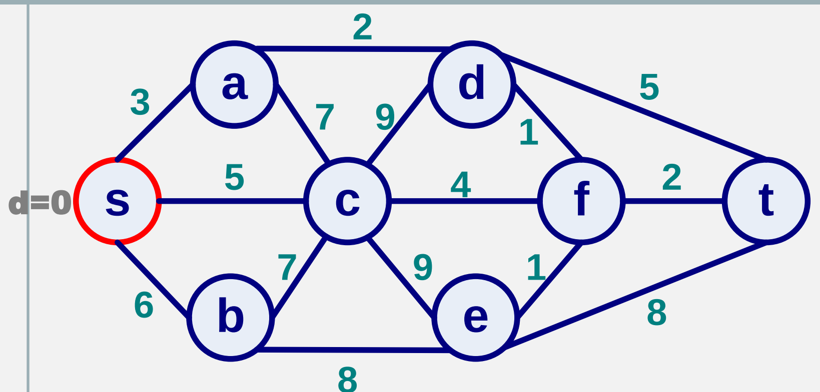
- Exemplo passo a passo do algoritmo de Dijkstra, considerando o seguinte grafo não dirigido:

  <u>Nota</u>:

  s – peso (ou distância) de **s** a **s**

  a – peso (ou distância) de **a** a **s**

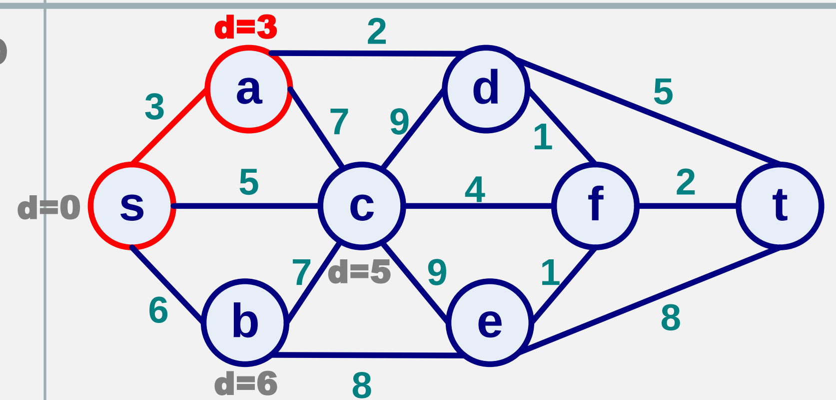
  b – ...

s	a	b	c	d	e	f	t	Grafo
0	∞	∞	∞	∞	∞	∞	∞	

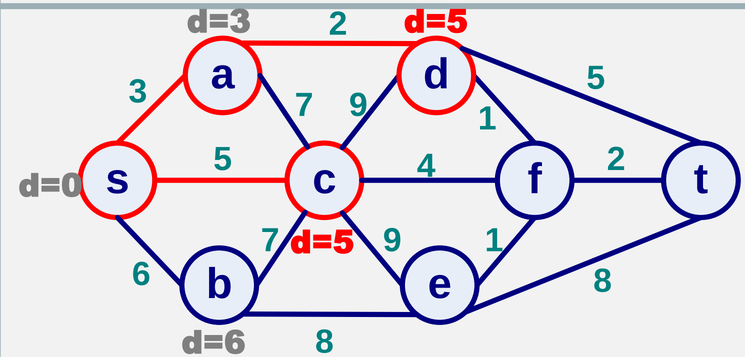
ALGORITMO DE DIJKSTRA

- Primeira iteração través da técnica *greedy*:

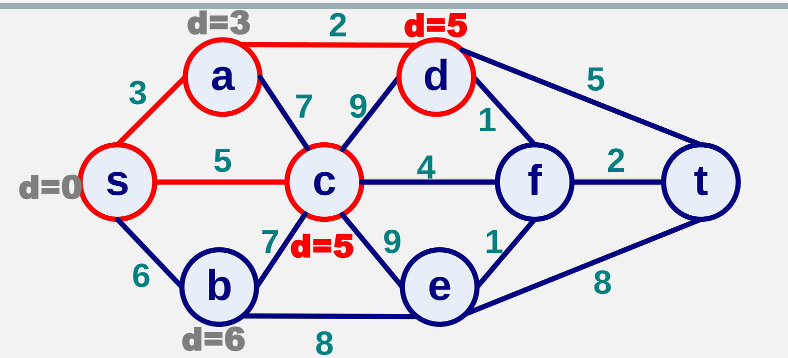
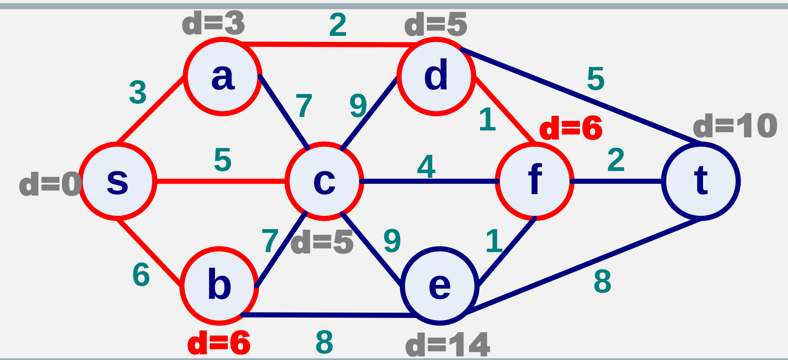
s	a	b	c	d	e	f	t	Grafo
0	3	6	5	∞	∞	∞	∞	
	3							



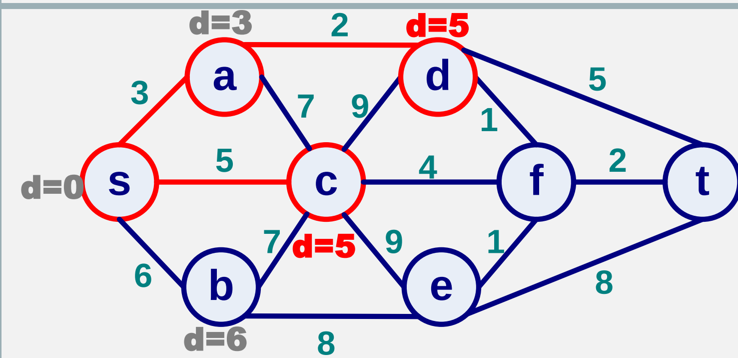
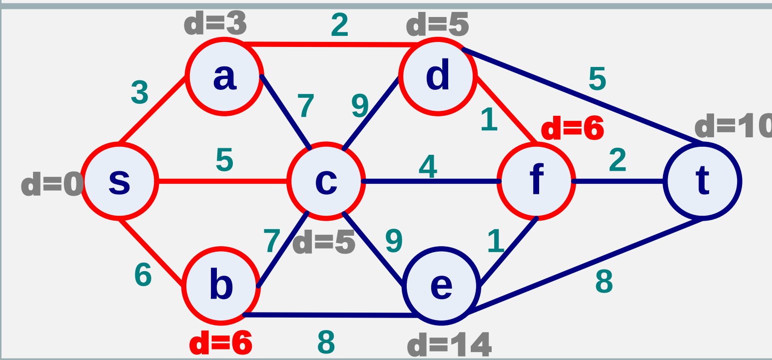
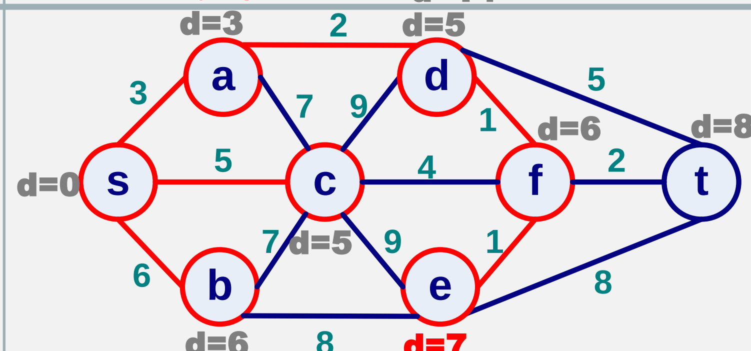
ALGORITMO DE DIJKSTRA

s	a	b	c	d	e	f	t	Grafo
0	3	6	5 5	5 5	∞	∞	∞	

ALGORITMO DE DIJKSTRA

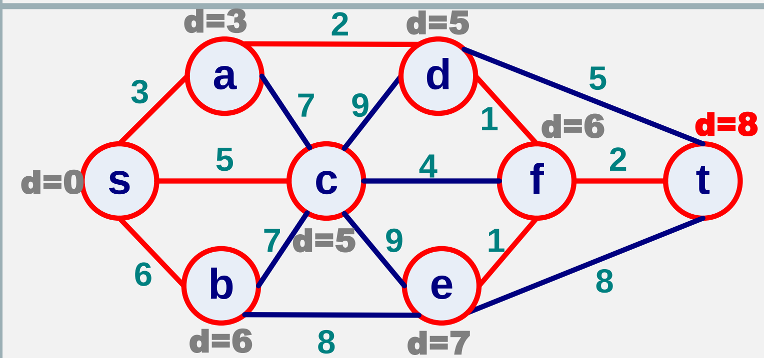
s	a	b	c	d	e	f	t	Grafo
0	3	6	5 5	5 5	∞	∞	∞	
0	3	6 6	5	5	14	6 6	10	

ALGORITMO DE DIJKSTRA

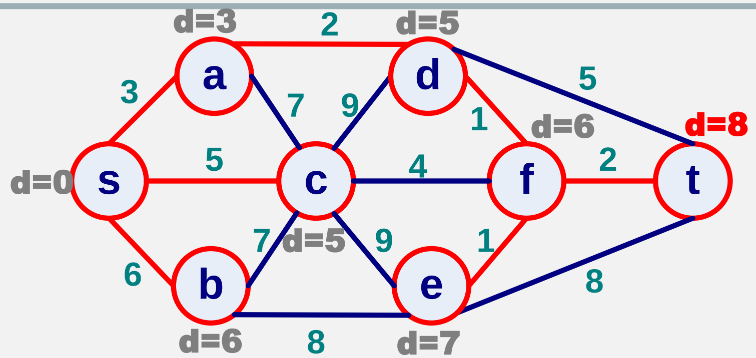
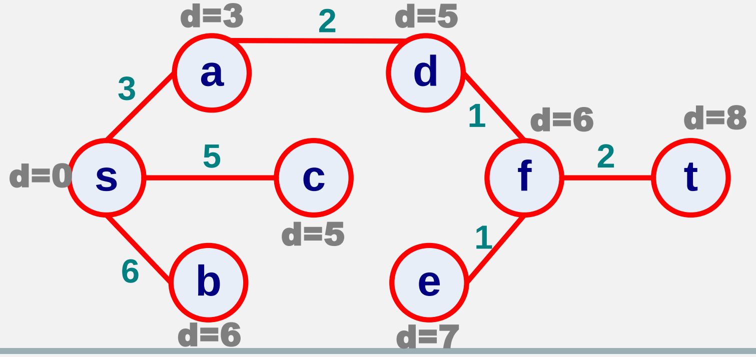
s	a	b	c	d	e	f	t	Grafo
0	3	6	5 5	5 5	∞	∞	∞	
0	3	6 6	5	5	14	6 6	10	
0	3	6	5	5	7 7	6	8	

ALGORITMO DE DIJKSTRA

s	a	b	c	d	e	f	t	Grafo
0	3	6	5	5	7	6	8 8	



ALGORITMO DE DIJKSTRA

s	a	b	c	d	e	f	t	Grafo
0	**3**	**6**	**5**	**5**	**7**	**6**	**8** **8**	
0	**3**	**6**	**5**	**5**	**7**	**6**	**8**	

ALGORITMO DE DIJKSTRA

// Implementa o algoritmo de Dijkstra para o cálculo do caminho mais curto
// Entrada: grafo direcionado pesado G = (V, A) e vértice inicial s (source)
// Saída: conjunto de vértices com os valores dos caminhos mais curtos a partir de s

```
Dijkstra(G, s)
Initialize-Single-Source(G, s) // inicializa todos os vertices do grafo
S = Ø    // conjunto de vertices com os caminhos  mais  curtos calculados
Q = G.V // min-priority queue iniciada com os vértices organizados por peso
while Q ≠ Ø
      u = Extract-Min(Q)
      S = S U {u}
      for each vertex v ∈ G.Adj[u] and v ∉ S
            Relax (u, v, w) // w (weight) calcula o peso da aresta entre u e v
```

ALGORITMO DE DIJKSTRA

// Inicializa todos os vértices do grafo

```
Initialize-Single-Source(G, s)
    for each vertex v ϵ G.V
        v.d = ∞
        v.p = NULL
    s.d = 0
```

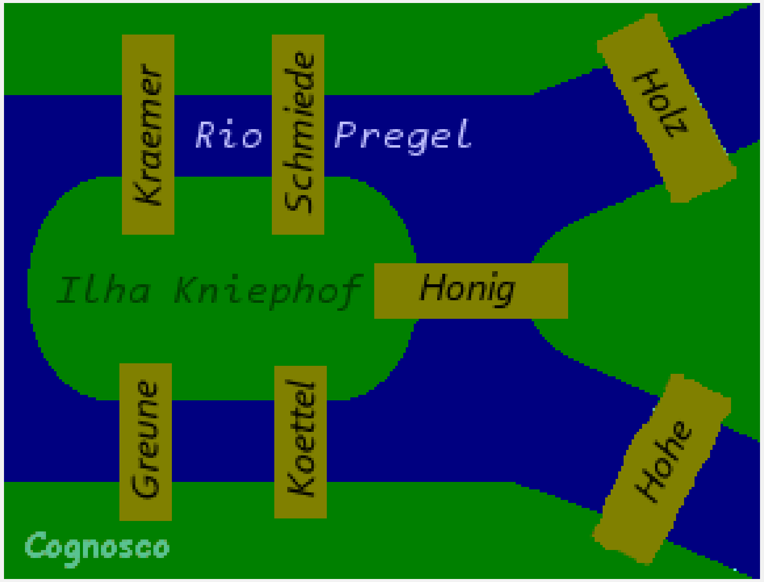
// O processo de *relaxing* de uma aresta (u, v), consiste na verificação se o caminho mais curto calculado até ao momento pode ser melhorado, somando a partir de u. Se for o caso, v.d e v.p são atualizados. Este processo pode diminuir o valor do caminho mais curto estimado v.d

```
Relax(u, v, w)
    if v.d > u.d + w(u, v) // w(u, v) devolve o peso da aresta entre u e v
        v.d = u.d + w(u, v)
        v.p = u
```

CIRCUITO DE EULER

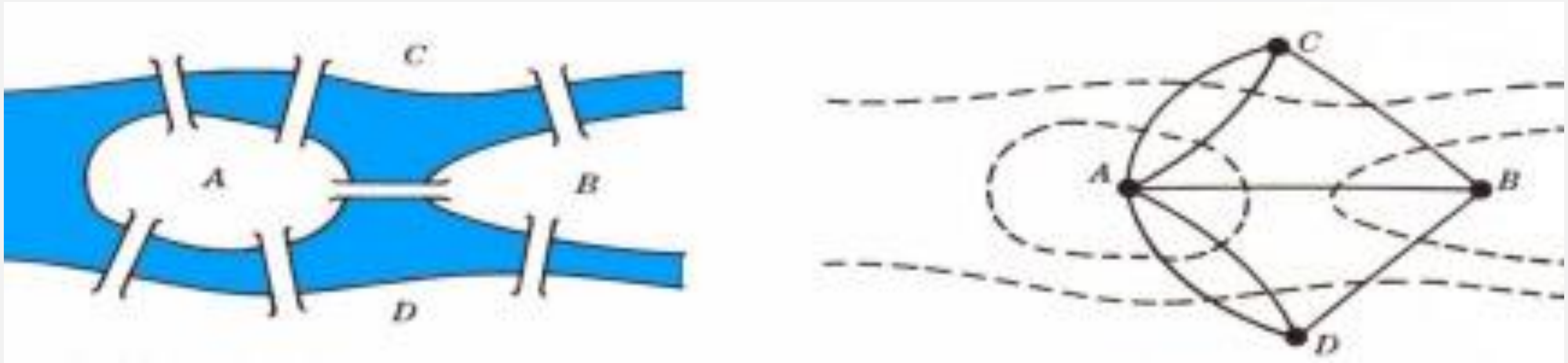
- Problema das Pontes de Königsberg

A cidade de Königsberg é banhada pelo rio Pregel que, ao atravessar a cidade se ramifica formando uma ilha (Kneiphof) que está ligada à restante parte da cidade por sete pontes. Dizia-se que os habitantes da cidade nos dias soalheiros de descanso, tentavam efectuar um percurso que os obrigasse a passar por todas as pontes, mas apenas uma vez em cada uma. Como as suas tentativas foram sempre falhadas, muitos deles acreditavam que não era possível encontrar tal percurso. Será que tinham razão?



CIRCUITO DE EULER

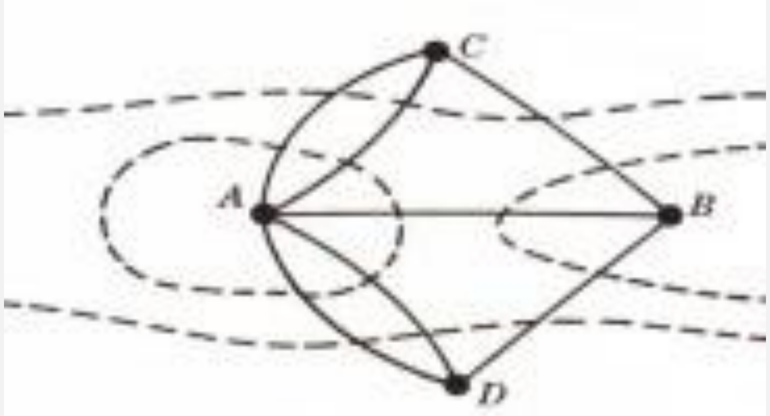
- Problema das Pontes de Königsberg
  - Euler representou cada zona A, B, C e D por um vértice de um grafo, correspondendo cada ponte a uma ligação entre as zonas, (arestas)
  - Obteve assim uma representação gráfica do problema:



CIRCUITO DE EULER

- Problema das Pontes de Königsberg
  - Como os graus de todos os vértices são ímpares, é fácil verificar que este grafo não apresenta nem um caminho nem um circuito euleriano, visto que não satisfaz o teorema de Euler
  - Os habitantes da cidade tinham razão ao concluir que tal percurso era impossível de realizar

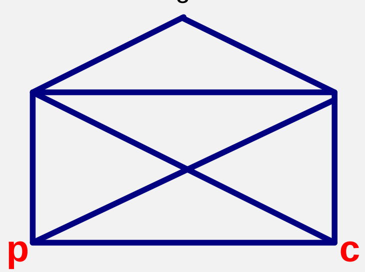
- Teorema de Euler
  - Um grafo G é euleriano se e somente se G é conexo e cada vértice de G tem grau par



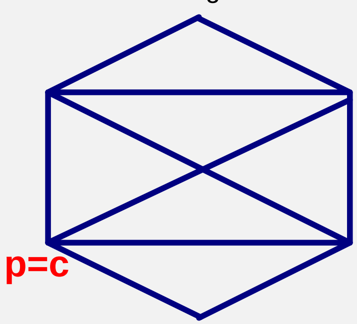
CIRCUITO DE EULER

- Tem também como base um puzzle antigo, que consiste em
  - Reconstruir as figuras usando uma caneta, desenhando todas as linhas de uma vez
  - A caneta não pode ser levantada do papel enquanto o desenho não estiver terminado
  - Desafio extra: o ponto de chegada deverá coincidir com o ponto de partida

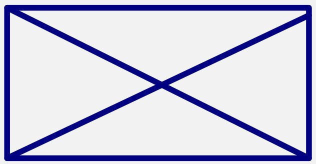
<u>Solução possível</u>
O ponto de partida é diferente do ponto de chegada



<u>Solução possível</u>
O ponto de partida coincide com o ponto de chegada



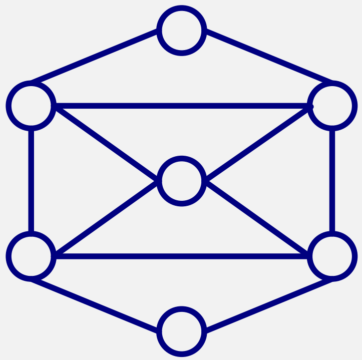
<u>Solução impossível</u>



CIRCUITO DE EULER

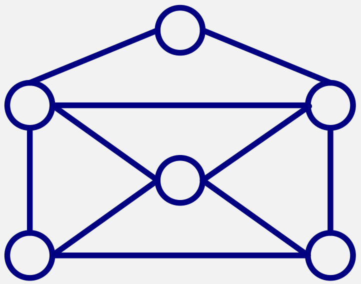
- Descoberto por Euler em 1736, o circuito de Euler marcou o início da teoria dos grafos
- O grafo tem que ser conexo

- Circuito de Euler
  - Encontrar um circuito (ou ciclo) no grafo que visite todas as aresta exatamente uma vez
  - O número de arestas em cada vértice (grau) tem que ser par. O circuito inicia-se e termina no mesmo vértice

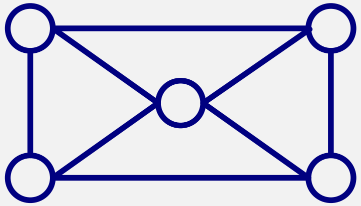


CIRCUITO DE EULER

- Caminho de Euler
  - Encontrar um caminho no grafo que visite todas as arestas exatamente uma vez
  - Só podem existir exatamente dois vértices com um número ímpar de arestas. O caminho inicia-se num desses vértices e termina no outro. Todos os outros vértices têm grau par



- Solução impossível
  - O número de vértices de grau ímpar é diferente de dois



CIRCUITO DE EULER

5, 4, 1, 3, 2, 8, 9, 12, 10, 9, 6, 3, 7, 4, 11, 10, 7, 9, 3, 4, 10, 5