TEG

Gilmário B. Santos

gilmario.santos@udesc.br

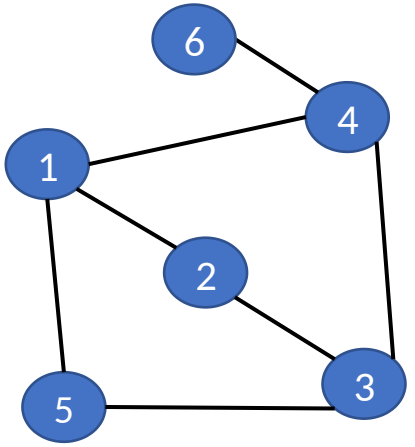
http://www.joinville.udesc.br/portal/pagina/gilmario

Conjunto desconectante

Um conjunto desconectante: um conjunto de arestas que, se removidas, desconectam o grafo gerando ou originando vários componentes.

Um corte de arestas é um conjunto minimal desconectante, ou seja, não contém nenhum subconjunto próprio que também desconecta o grafo.

Exemplo:



A_1={(1,5);(3,5)}

A_2={(1,4);(3,3)}

A_3={(1,2);(2,3)}

A_4={(1,2);(2,3);(1,5);(3,5)}

A_5={(6,4)}

A_1, A_2 , A_3 e A_5 são cortes de arestas

A_4 Não é corte de arestas pois contém A_1 e A_3, subconjuntos próprios que já são cortes de arestas

A_3 é um conjunto minimal desconectante, porém, não é mínimo (A_4 é um conjunto desconectante mínimo)

Conjunto desconectante

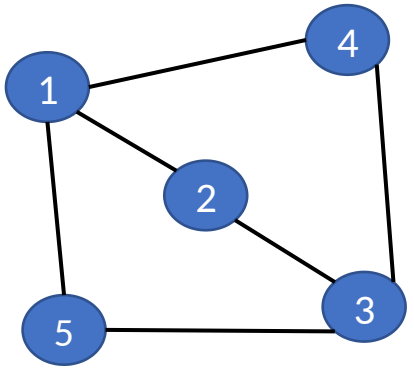
Aresta conectividade (λ) corresponde à cardinalidade do menor corte de arestas. Para o grafo abaixo, λ=2 (para desconectar o grafo eu preciso retirar pelo menos duas arestas):

Exemplo:

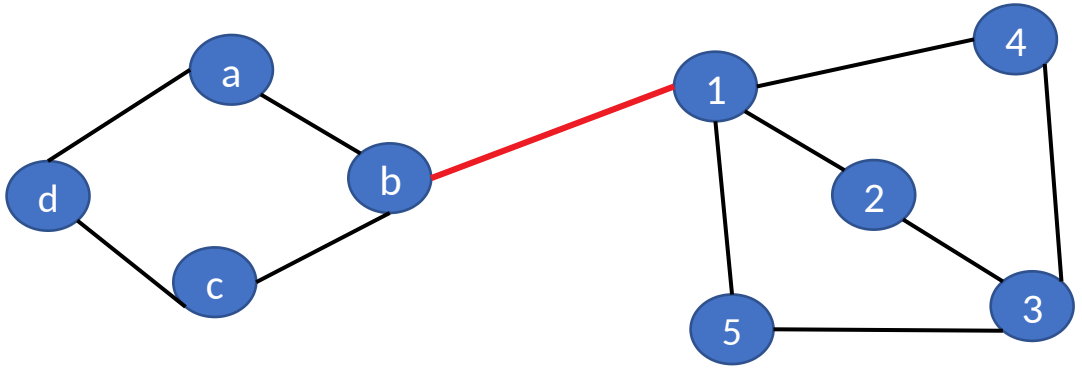
A_1={(1,5);(3,5)}

A_2={(1,4);(3,3)}

A_3={(1,2);(2,3)}



Se λ=1, a aresta que desconecta o grafo é chamada de aresta de ponte.



Conjunto separador

Um conjuntos separadores: conjunto de vértices que, se removidos, desconectam o grafo gerando ou originando vários componentes.

Um corte de vértices é um conjunto minimal desconectante, ou seja, não contém nenhum subconjunto próprio que também desconecta o grafo.

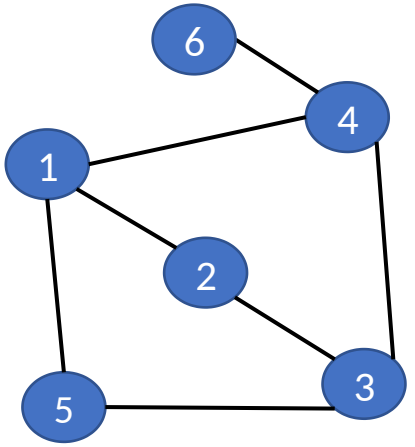
Exemplo:

C_1={1,3}

C_2={5,4}

C_3={{1,3},{5,4}}

C_4={4}



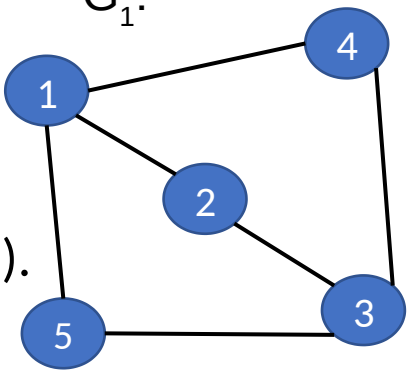
C_1 , C_2 e C_4 são cortes de vértices

C_3 Não é corte de vértices pois contém cortes

Conjunto separador

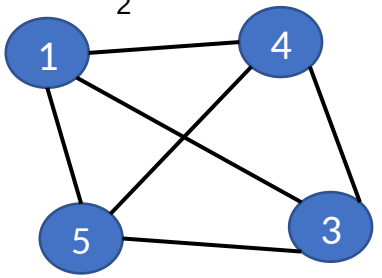
Vértice conectividade (β) corresponde à cardinalidade do menor corte de vértices.

G_1:



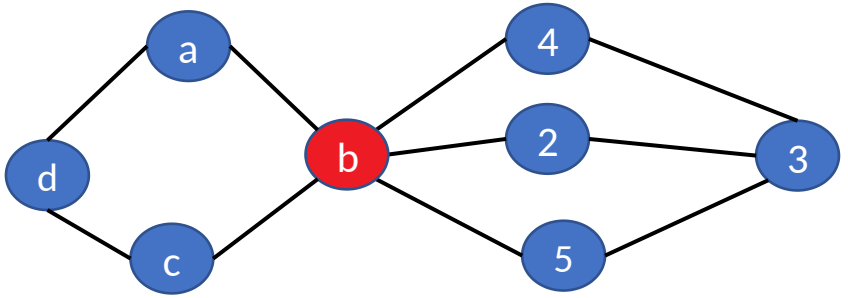
G_1: β=2 (para desconectar o grafo eu preciso retirar pelo menos dois vértices {1,3}).

G_2:



G_2: é grafo completo → não há conjunto separador que desconecte K_n

Se β=1, o vértice que desconecta o grafo é chamada de vértice de articulação:



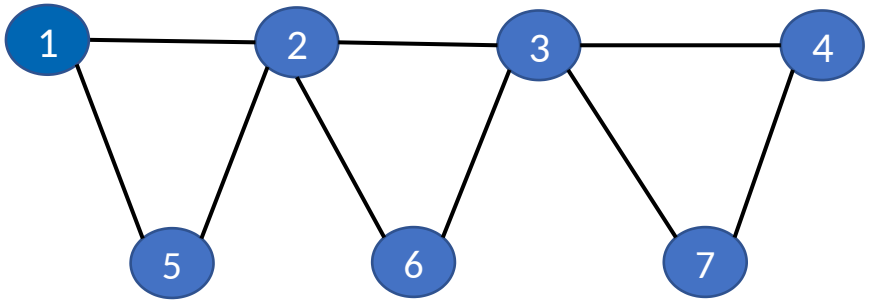
Árvore geradora

Uma árvore é um grafo conexo acíclico, no qual, toda aresta é uma ponte;

Denomina-se subgrafo gerador (*spanning*) de um grafo G1(V1,E1) a um subgrafo G2(V2,E2) de G1 tal que V1=V2;

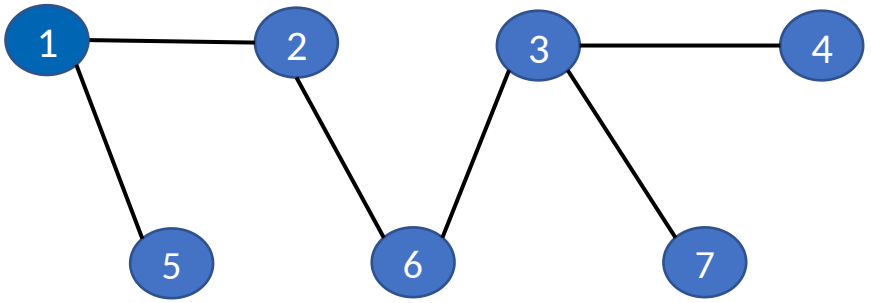
Quando o subgrafo gerador é uma árvore, ele recebe o nome de árvore geradora (*spanning tree*);

G

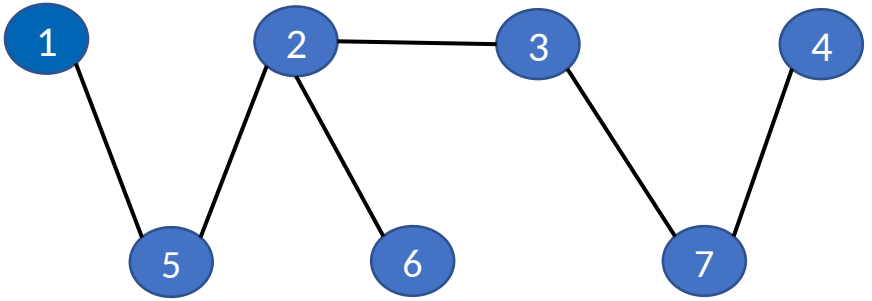


Onde está a raiz? Qualquer vértice pode ser a raiz para T (árvore geradora de G)...

T1:



T2:

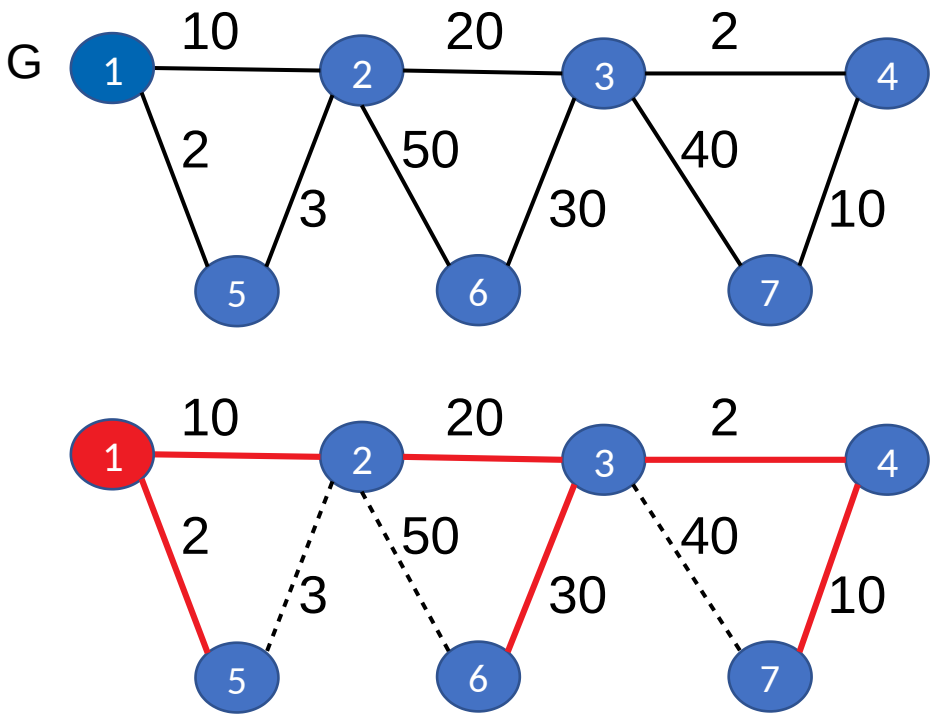


Árvore Geradora de Custo Mínimo

Um grafo pode apresentar diversas árvores geradoras, em certas aplicações pode nos interessar uma dessas versões;

Das muitas árvores geradoras em grafos de arestas ponderadas, nos interessa a que apresenta a soma mínima (ou máxima) dos pesos;

Esta é a árvore geradora de custo mínimo (*Minimum Spanning Tree - MST*)

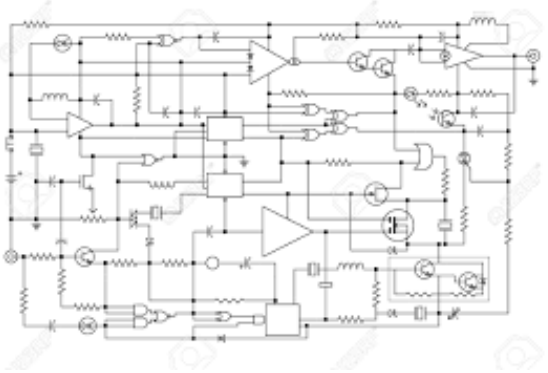


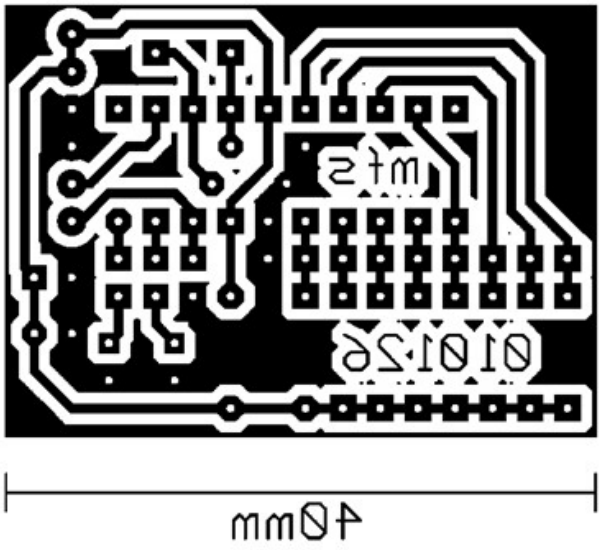
Árvore Geradora de Custo Mínimo – aplicações

Em projeto de circuitos eletrônicos, muitas vezes, é necessário conectar eletricamente os pinos de vários componentes.

Para interconectar um conjunto de *n* pinos, podemos usar um arranjo de *n – 1* trilhas condutoras, cada qual conectando dois pinos.

De todos os arranjos possíveis, aquele que utiliza a mínima metragem de trilhas é normalmente o mais desejável;



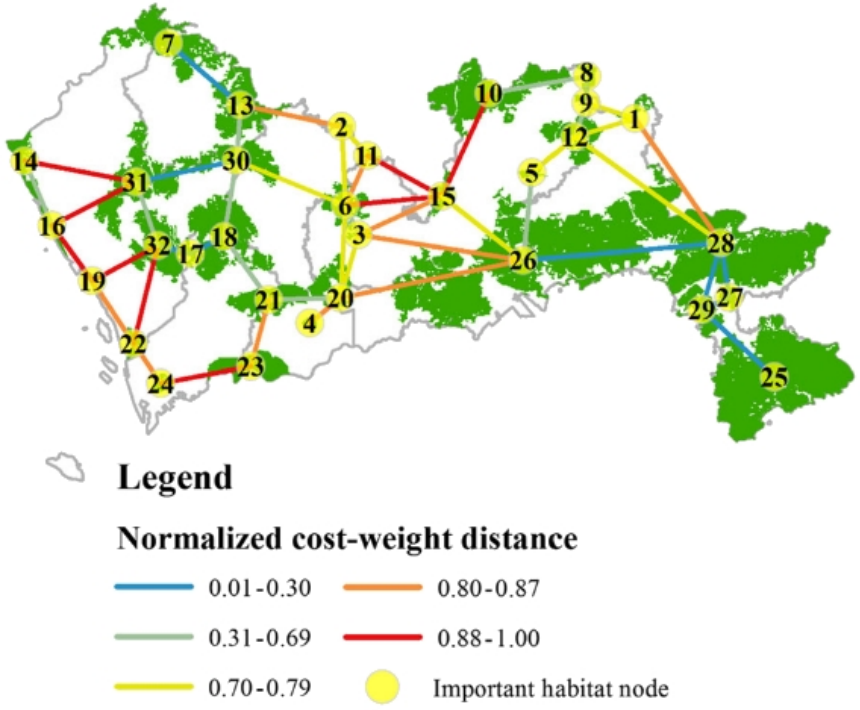


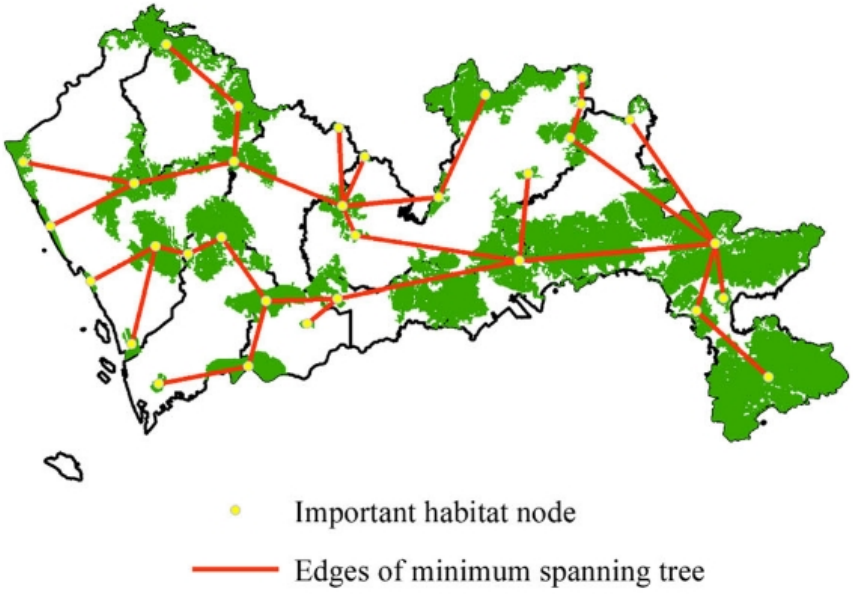
Aplicação em processamento de imagens LINK:

Árvore Geradora de Custo Mínimo – aplicações

Luo, Y., Wu, J. Linking the minimum spanning tree and edge betweenness to understand arterial corridors in an ecological network. Landscape Ecol 36, 1549–1565 (2021). https://doi.org/10.1007/s10980-021-01201-1

- Objetivo: construir uma rede ecológica identificando corredores/artérias entre habitats ecologicamente importantes;
- Um dos fundamentos do método aplicado está no cálculo de uma árvore geradora (minimum spanning tree - MST);
- A MST identificou os corredores ecológicos na rede, os demais corredores da rede foram considerados redundantes (as arestas correspondentes a esses corredores não constam na MST).

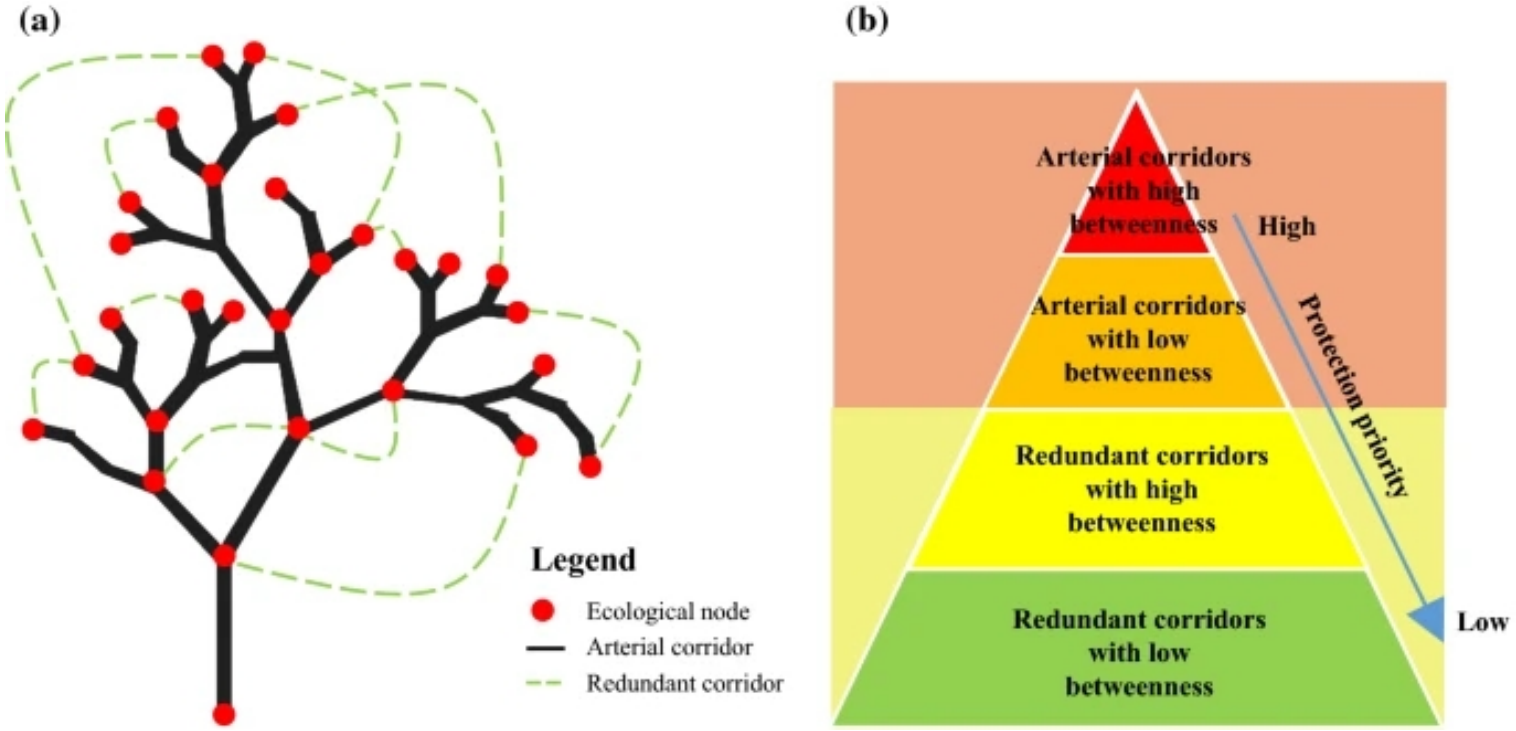




Árvore Geradora de Custo Mínimo – aplicações

Luo, Y., Wu, J. Linking the minimum spanning tree and edge betweenness to understand arterial corridors in an ecological network. Landscape Ecol 36, 1549–1565 (2021). https://doi.org/10.1007/s10980-021-01201-1

- A MST identificou os tais corredores ecológicos na rede, ao passo que os outros corredores da rede foram considerados redundantes (as arestas correspondentes a esses corredores não constam na MST).



Árvore Geradora de Custo Mínimo

Podemos modelar esse tipo de problema com um grafo conexo não dirigido G = (V, E):

V: corresponde ao conjunto de pinos (ou habitats);

E: corresponde ao conjunto de interconexões possíveis entre pares de pinos (ou entre habitats) e,

Para cada aresta (u, v) $\in$ E, há um peso w(u, v) que especifica o custo (a medida da trilha ou a distância entre habitats) para conectar o par de vértices *u* e *v*.

Desejamos encontrar um subconjunto acíclico T $\subseteq$ E que conecte todos os vértices e cujo peso total W(T) seja minimizado.

Visto que T é acíclico e conecta <u>todos</u> os vértices, T deve formar uma árvore que denominaremos árvore geradora, já que “gera” o grafo G.

$$W(T) = \sum_{(u,v) \in T} w(u,v)$$

O problema de determinar a árvore T é denominado problema da árvore geradora mínima.

Árvore Geradora de Custo Mínimo

Existem algoritmos para o cálculo da MST, os mais populares são algoritmo de Kruskal e o algoritmo de Prim.

Assim como Dijkstra, Kruskal e Prim utilizam a estratégia gulosa sendo que Prim guarda similaridade com o algoritmo de Dijkstra;

[Cormen] Cada etapa de um algoritmo guloso deve fazer uma entre várias opções possíveis. A estratégia gulosa faz a escolha que é a melhor no momento. Em geral, tal estratégia não garante que sempre encontrará soluções globalmente ótimas para problemas. Porém, no caso do problema da árvore geradora mínima, podemos provar que certas estratégias gulosas realmente produzem uma árvore geradora com peso mínimo.

Árvore Geradora de Custo Mínimo

Antes de cada iteração, A é um subconjunto de alguma árvore geradora mínima.

Em cada etapa, aresta (u, v) deve ser adicionada a A sem violar o invariante A = A ∪ {(u, v)} é sempre um subconjunto de uma árvore geradora mínima. Denominamos tal aresta aresta (u,v) como segura para A, já que ela pode ser adicionada com segurança a A e, ao mesmo tempo, manter o invariante.

Algoritmo genérico:

```
GENERIC-MST(G, w)
A = ∅
while A não formar uma árvore geradora
    encontre uma aresta (u, v) que seja segura para A
    A = A ∪ {(u, v)}
return A
```

Árvore Geradora de Custo Mínimo

Precisaremos de algumas definições:

Um corte $(S, V - S)$ de um grafo não dirigido $G = (V, E)$ é uma partição de V.

Uma aresta $(u, v) \in E$ cruza o corte $(S, V - S)$ se um de seus pontos extremos está em S e o outro está em $V - S$.

Um corte respeita um conjunto A de arestas se nenhuma aresta em A cruza o corte.

Uma aresta é considerada "*aresta leve que cruza um corte*" se o seu peso é o mínimo dentre quaisquer arestas que cruzam o corte (pode haver mais de uma aresta leve que cruza um corte no caso de laços, para um grafo que permita tal ocorrência.)

Kruskal e Prim montam a árvore identificando e acrescentando arestas seguras a um conjunto inicial (A) que contém apenas a raiz da árvore.

A aresta segura adicionada a A é sempre uma aresta leve que cruza um corte, ou seja, uma aresta de peso mínimo que conecta a árvore a um vértice não presente na árvore.

Árvore geradora de custo mínimo via Prim

O algoritmo de Prim tem a seguinte propriedade: as arestas de um conjunto *A* sempre formam uma árvore única.

A árvore (*A*) começa em um vértice raiz arbitrário *r* e aumenta até que a árvore abranja todos os vértices em *V*.

Cada etapa adiciona à árvore *A* uma aresta leve que conecta *A* a um vértice no qual nenhuma aresta de *A* incide.

Corolário 23.2 (Cormen et al.), essa regra adiciona apenas arestas que são seguras para *A*; portanto, quando o algoritmo termina, as arestas em *A* formam uma árvore geradora mínima. Essa estratégia se qualifica como gulosa, já que a cada etapa ela adiciona à árvore uma aresta que contribui com a mínima quantidade possível para o peso da árvore.

Árvore geradora de custo mínimo via Prim

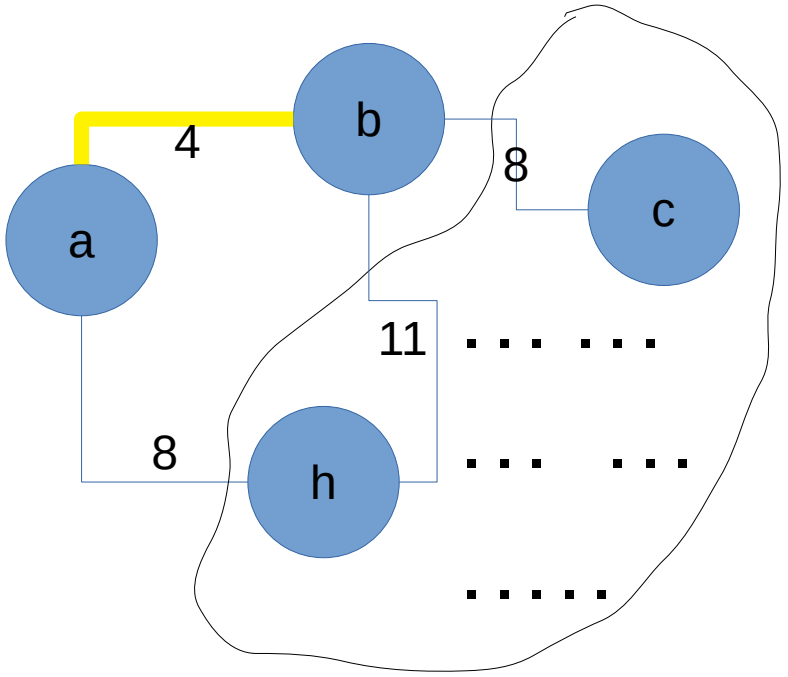
Durante a execução do algoritmo, todos os vértices que não estão na árvore *A* residem em uma fila de prioridade mínima *Q* ordenada pelo atributo *v.chave* (peso de aresta entre v e um vértice da árvore *A*).

A e *Q* definem um corte do grafo conexo, até que todos os vértices façam parte de *A* via arestas seguras;

A={a}

	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	-	a	-	-	-	-	-	a	-

FILA	**a**	b	h	d	e	f	g	h	**i**
chave	0	4	8	∞	∞	∞	∞	∞	∞
frente		Δ							
cauda									Δ



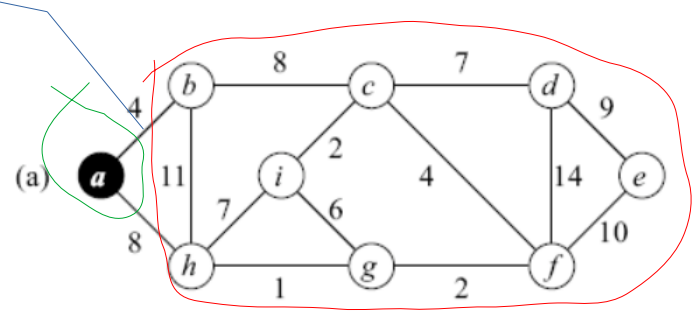
Árvore geradora de custo mínimo via Prim

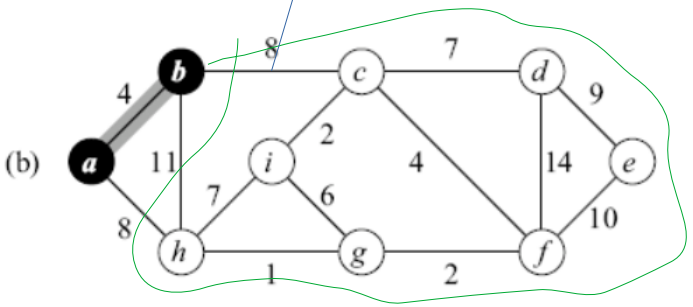
Para cada vértice v, o atributo v.chave é o peso mínimo de qualquer aresta que conecta v a um vértice na árvore; por convenção, v.chave = ∞ se não existe nenhuma aresta desse tipo.

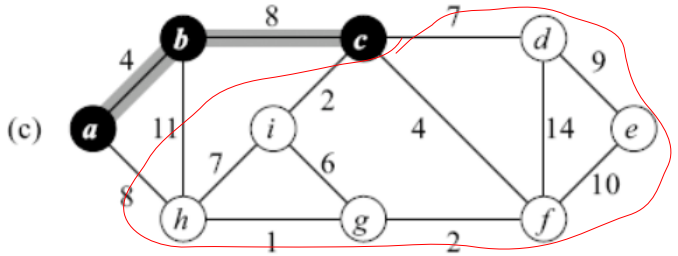
O atributo v.π nomeia o pai de *v* na árvore.

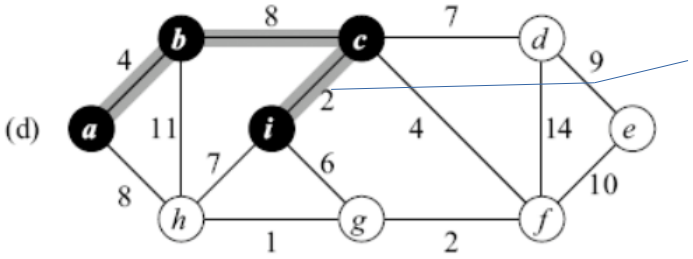
Aresta
Leve: segura

Aresta
Leve: segura









Aresta
Leve: segura

. . . .

Árvore geradora de custo mínimo via Prim

$A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.

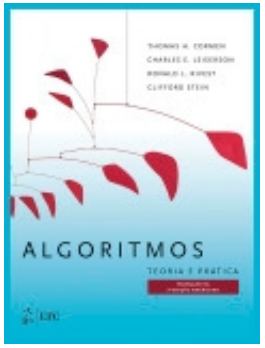
Quando o algoritmo termina, a fila de prioridade mínima Q está vazia; portanto, a árvore geradora mínima A para G é

$A = \{(v, v.\pi) : v \in V - \{r\}\}$.

```
MST-PRIM(G, w, r)
1     for cada u ∈ V[G]
2         u.chave = ∞
3         u.π = NIL
4     r.chave = 0
5     Q = V[G]
6     while Q ≠ ∅
7         u = EXTRACT-MIN(Q)
8         for cada v ∈ G. Adj[u]
9             if v ∈ Q e w(u, v) < v.chave
10                v.π = u
11                v.chave = w(u, v)
```

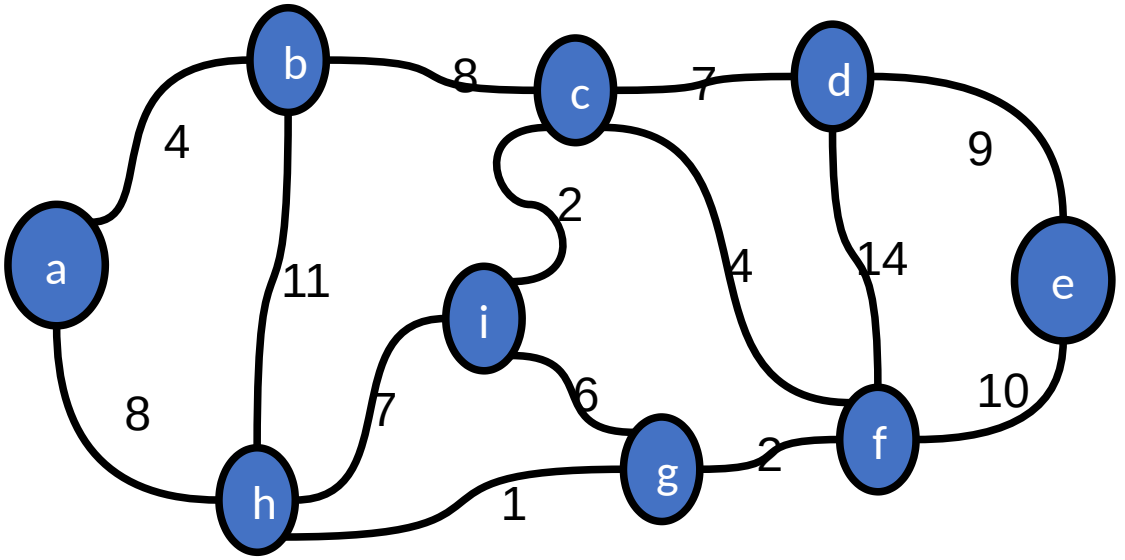
Árvore geradora de custo mínimo (Minimum Spanning Tree)

Vamos ao algoritmo de Prim para determinação da Minimum Spanning Tree na seção 23.2, pg 460 do livro do Cormen (disponível no formato ebook e fisicamente no acervo da biblioteca do CCT).



MST – PRIM
(Cormen pg 461)

	a	b	c	d	e	f	g	h	i
a	∞	4	∞	∞	∞	∞	∞	8	∞
b	4	∞	8	∞	∞	∞	∞	11	∞
c	∞	8	∞	7	∞	4	∞	∞	2
d	∞	∞	7	∞	9	14	∞	∞	∞
e	∞	∞	∞	9	∞	10	∞	∞	∞
f	∞	∞	4	14	10	∞	2	∞	∞
g	∞	∞	∞	∞	∞	2	∞	1	6
h	8	11	∞	∞	∞	∞	1	∞	7
i	∞	∞	2	∞	∞	∞	6	7	∞



MST – PRIM
(Cormen pg 461)

Inicialização:

Raiz: r=a
Tabela:

chave(v) : peso da aresta que conecta *v* à árvore

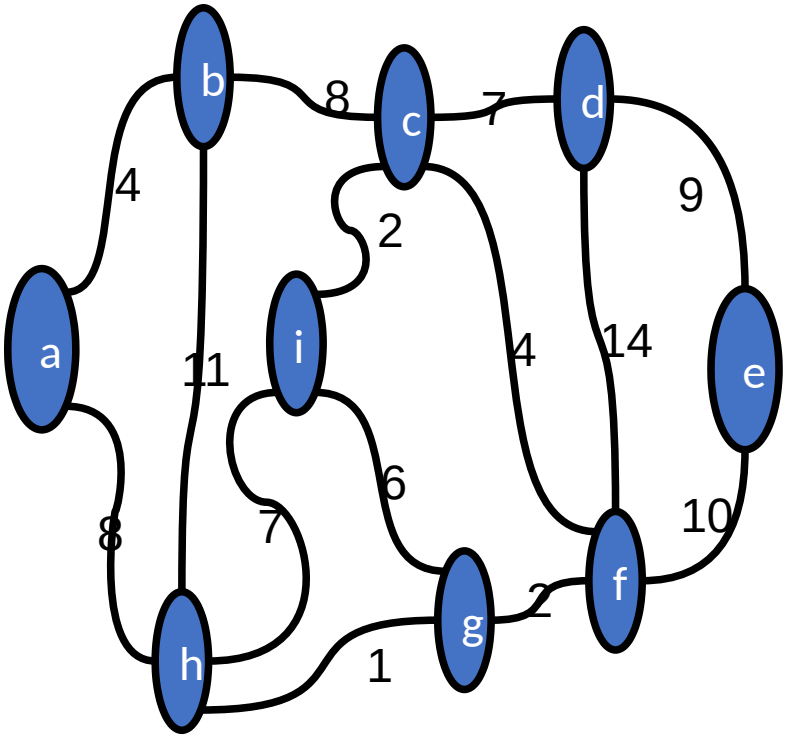
π(v) : vértice que pertence à árvore e é adjacente a *v*

Fila de prioridade mínima baseada em chave

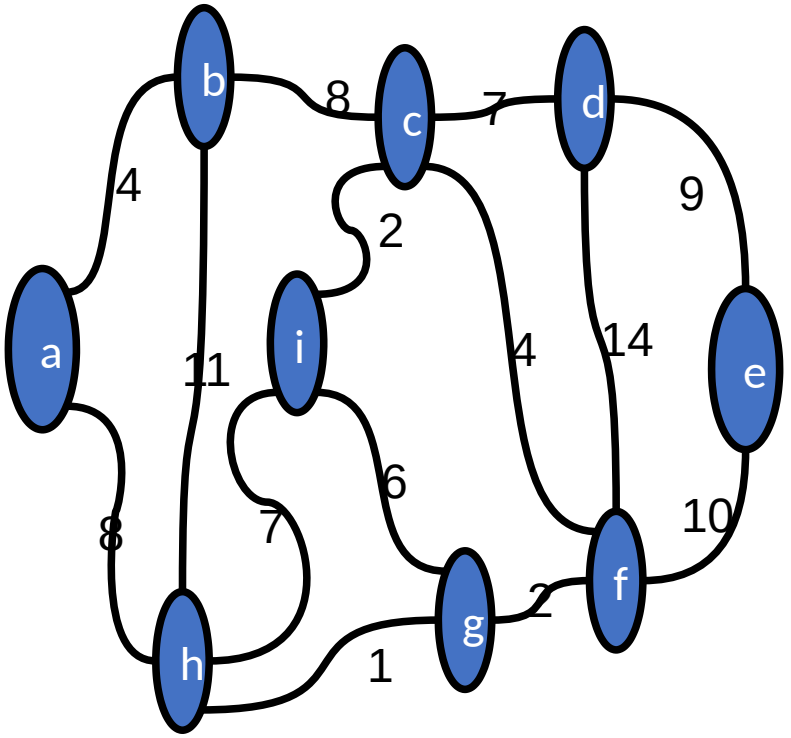
FILA	**a**	b	c	d	e	f	g	h	**i**
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
	F								C

frente cauda

	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-



MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)
---

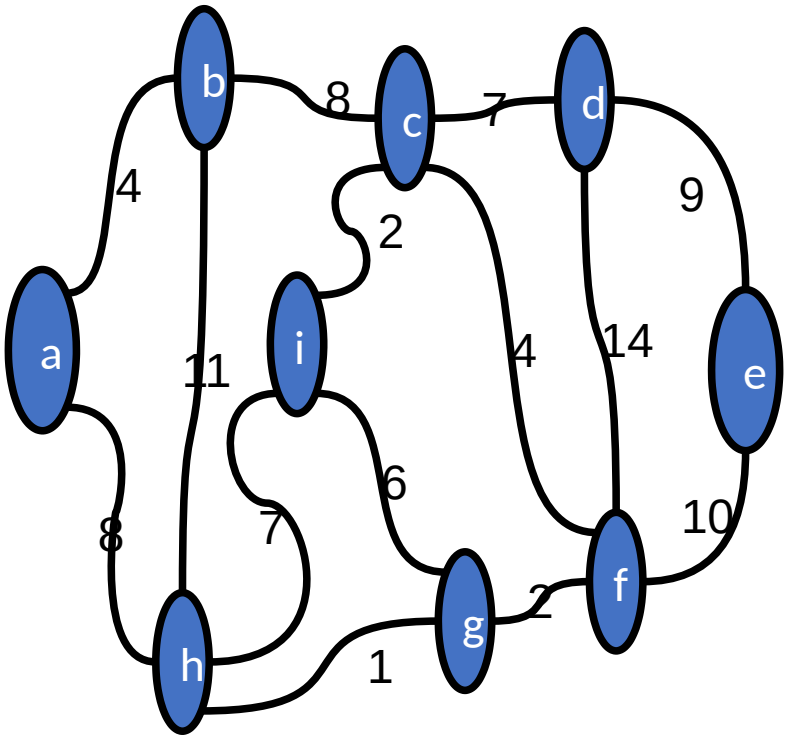
u=a

FILA	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
	F								C

FILA	~~a~~	b	h	d	e	f	g	c	i
chave	~~0~~	∞	∞	∞	∞	∞	∞	∞	∞
		F							C

	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=a

true true

v=h: h ∈ Q AND (w(a,h) < chave(h)) → TRUE

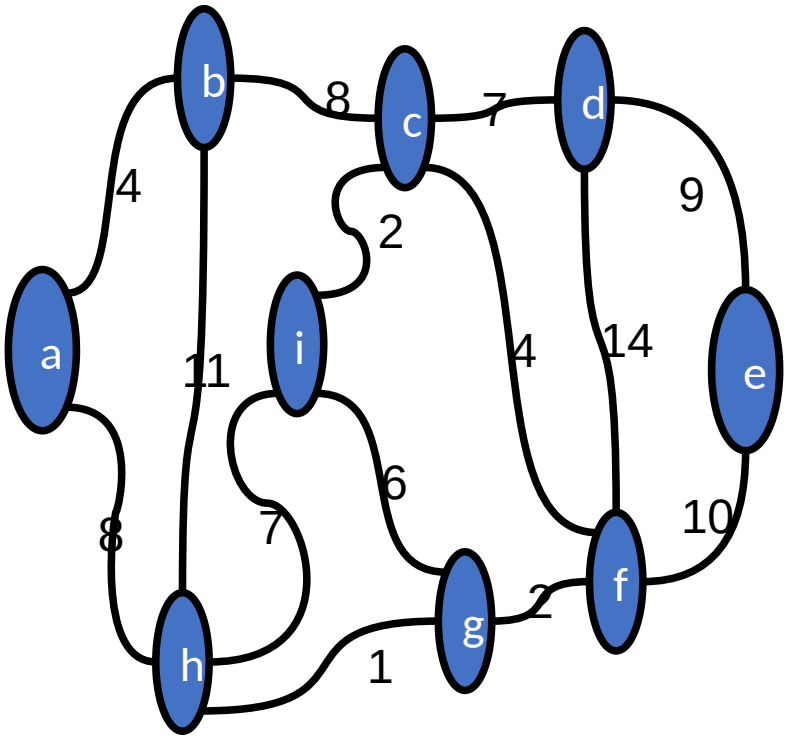
π(h) = a chave(h) = w(a, h) = 8

FILA	~~a~~	b	h	d	e	f	g	c	i
chave	~~0~~	∞	∞	∞	∞	∞	∞	∞	∞
		F							C

	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-

	a	b	c	d	e	f	g	**h**	i
chave	0	∞	∞	∞	∞	∞	∞	**8**	∞
π	-	-	-	-	-	-	-	**a**	-

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=a

v=h: h ∈ Q AND (w(a,h) < chave(h)) → TRUE
    π(h) = a   chave(h) = w(a, h) = 8

true true

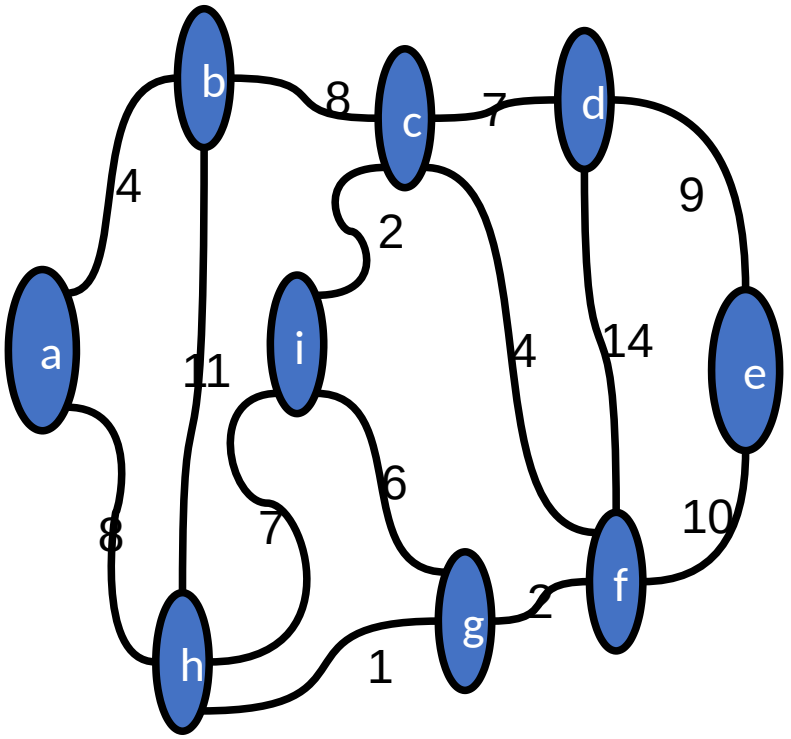
v=b: b ∈ Q AND (w(a,b) < chave(b)) → **TRUE**
    π(b) = a   chave(b) = w(a, b) = 4

FILA	~~a~~	b	h	d	e	f	g	c	i
chave	~~0~~	∞	∞	∞	∞	∞	∞	∞	∞
		F							C

	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-

	a	**b**	c	d	e	f	g	h	i
chave	0	**4**	∞	∞	∞	∞	∞	8	∞
π	-	**a**	-	-	-	-	-	a	-

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=a

v=h: h ∈ Q AND (w(a,h) < chave(h)) → TRUE
    π(h) = a   chave(h) = w(a, h) = 8

v=b: b ∈ Q AND (w(a,b) < chave(b)) → TRUE
    π(b) = a   chave(b) = w(a, b) = 4

FILA	~~a~~	b	h	d	e	f	g	c	i
chave	~~0~~	∞	∞	∞	∞	∞	∞	∞	∞
		F							C

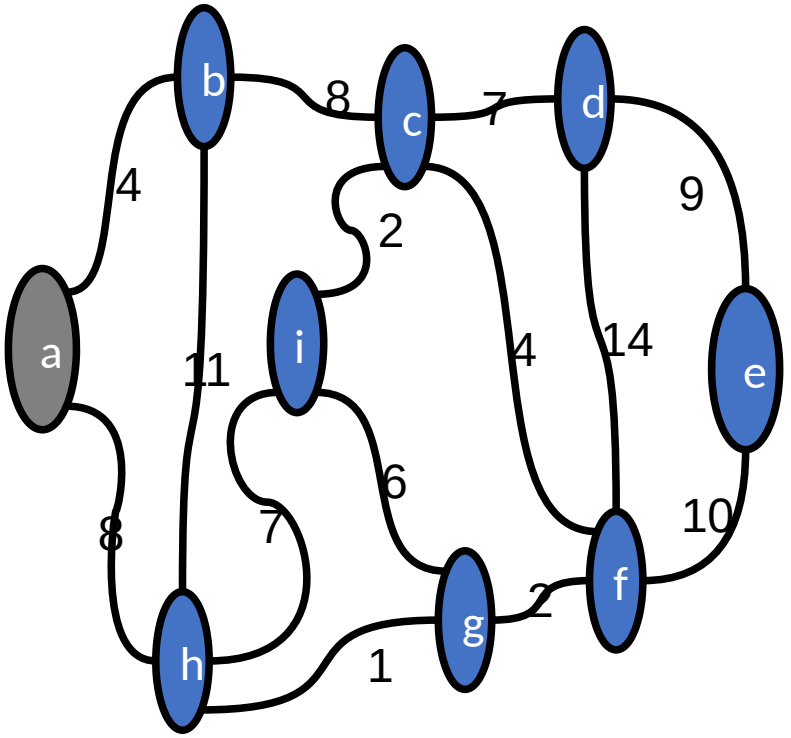
	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-

FILA	~~a~~	b	h	d	e	f	g	c	i
chave	~~0~~	4	8	∞	∞	∞	∞	∞	∞
		F							C

Atualiza a fila

	a	**b**	c	d	e	f	g	**h**	i
chave	0	**4**	∞	∞	∞	∞	∞	**8**	∞
π	-	**a**	-	-	-	-	-	**a**	-

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=b
    v=a: ~~a ∈ Q~~ AND (w(b,a) < chave(a)) → FALSE
    v=h: h ∈ Q AND ~~(w(b,h) < chave(h))~~ → FALSE
    v=c: c ∈ Q AND (w(b,c) < chave(c)) → TRUE
        π(c) = b chave(c) = w(b,c) = 8

FILA		~~b~~	h	d	e	f	g	c	i
chave		~~4~~	8	∞	∞	∞	∞	∞	∞
			F						C

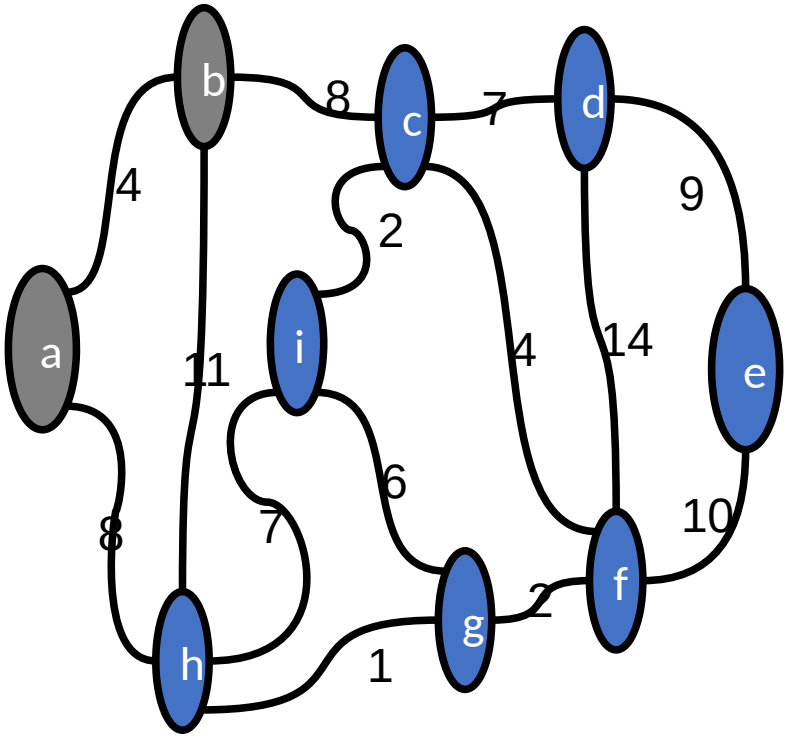
FILA		c	h	d	e	f	g	i	
chave		8	8	∞	∞	∞	∞	∞	
		F						C	

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	-	a	-	-	-	-	-	a	-

	a	b	**c**	d	e	f	g	h	i
chave	0	4	**8**	∞	∞	∞	∞	8	∞
π	-	a	**b**	-	-	-	-	a	-

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=c

v=b: ~~b ∈ Q~~ AND (w(c,b) < chave(b)) → FALSE
v=i: i ∈ Q AND (w(c,i) < chave(i)) → TRUE
v=d: d ∈ Q AND (w(c,d) < chave(d)) → TRUE
v=f: f ∈ Q AND (w(c,f) < chave(f)) → TRUE

FILA		~~c~~	h	d	e	f	g	i	
chave		~~8~~	8	∞	∞	∞	∞	∞	
			F					C	

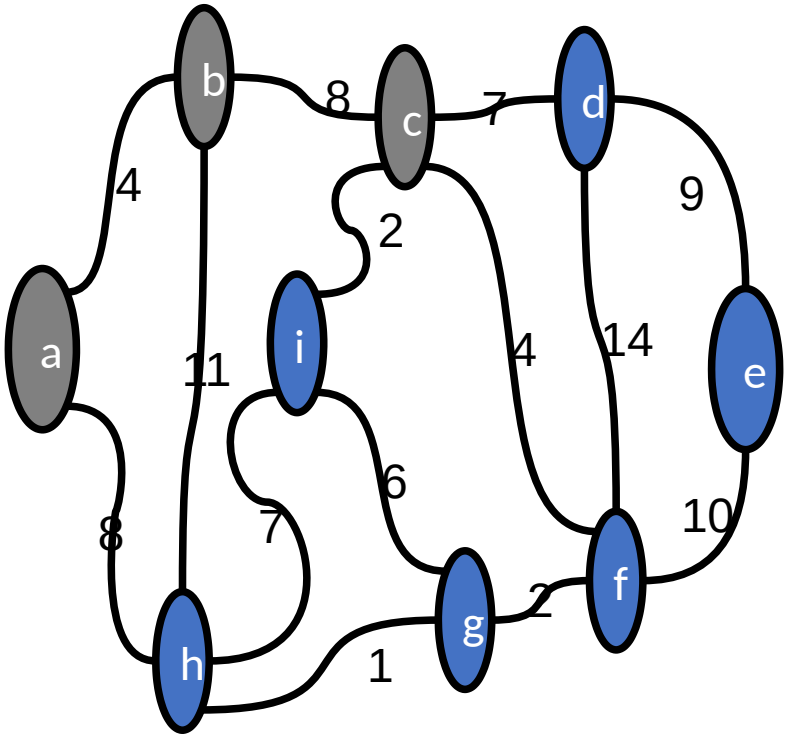
FILA		i	f	d	h	g	e		
chave		2	4	7	8	∞	∞		
		F					C		

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	∞	∞	∞	∞	8	∞
π	-	a	b	-	-	-	-	a	-

	a	b	c	**d**	e	**f**	g	h	**i**
chave	0	4	8	**7**	∞	**4**	∞	8	**2**
π	-	a	b	**c**	-	**c**	-	a	**c**

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=i

v=c: ~~c ∈ Q~~ AND (w(i,c) < chave(c)) → FALSE
v=h: h ∈ Q AND (w(i,h) < chave(h)) → TRUE
v=g: g ∈ Q AND (w(i,g) < chave(g)) → TRUE

FILA		~~i~~	f	d	h	g	e		
chave		~~2~~	4	7	8	∞	∞		
			F				C		

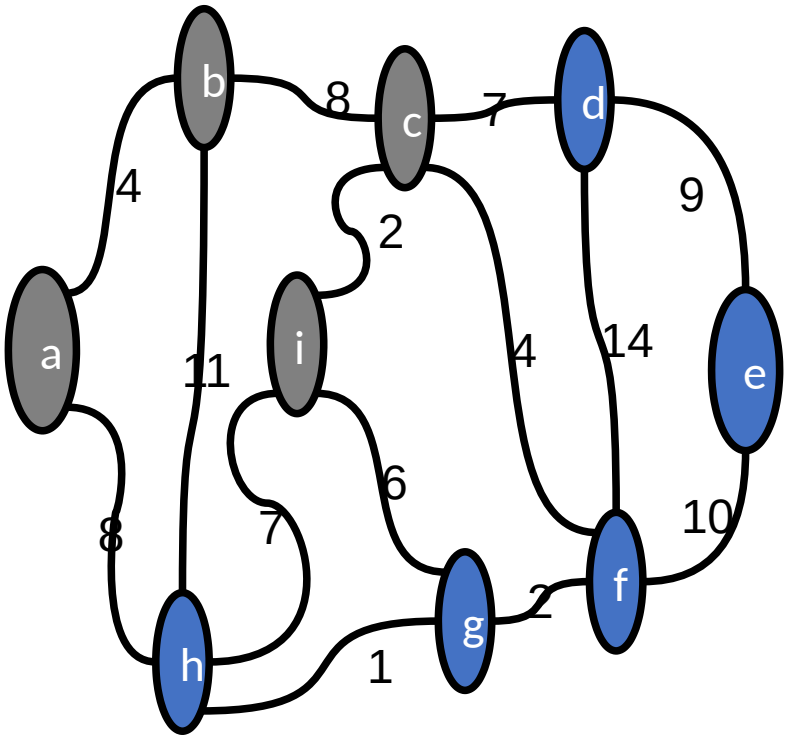
FILA			f	g	d	h	e		
chave			4	6	7	7	∞		
frente			F				C		

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	∞	8	2
π	-	a	b	c	-	c	-	a	c

	a	b	c	d	e	f	**g**	**h**	i
chave	0	4	8	7	∞	4	**6**	**7**	2
π	-	a	b	c	-	c	**i**	**i**	c

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=f

v=c: ~~c ∈ Q~~ AND (w(f,c) < chave(c)) → FALSE
v=d: d ∈ Q AND ~~(w(f,d) < chave(d))~~ → FALSE
v=e: e ∈ Q AND (w(f,e) < chave(e)) → TRUE
v=g: g ∈ Q AND (w(f,g) < chave(g)) → TRUE

FILA			~~f~~	g	d	h	e		
chave			~~4~~	6	7	7	∞		
				F			C		

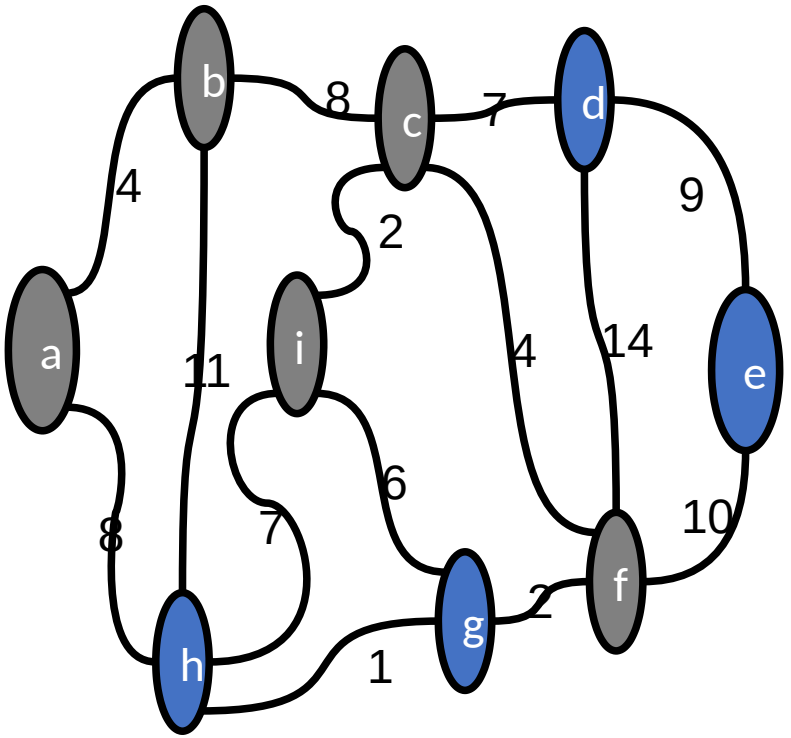
FILA				g	d	h	e		
chave				2	7	7	10		
				F			C		

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	6	7	2
π	-	a	b	c	-	c	i	i	c

	a	b	c	d	**e**	f	**g**	h	i
chave	0	4	8	7	**10**	4	**2**	7	2
π	-	a	b	c	**f**	c	**f**	i	c

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=g
    v=i: ~~i ∈ Q~~ AND (w(g,i) < chave(i)) → FALSE
    v=f: ~~f ∈ Q~~ AND (w(f,d) < chave(d)) → FALSE
    v=h: h ∈ Q AND (w(g,h) < chave(h)) → TRUE

FILA				~~g~~	d	h	e		
chave				~~2~~	7	7	10		
					F		C		

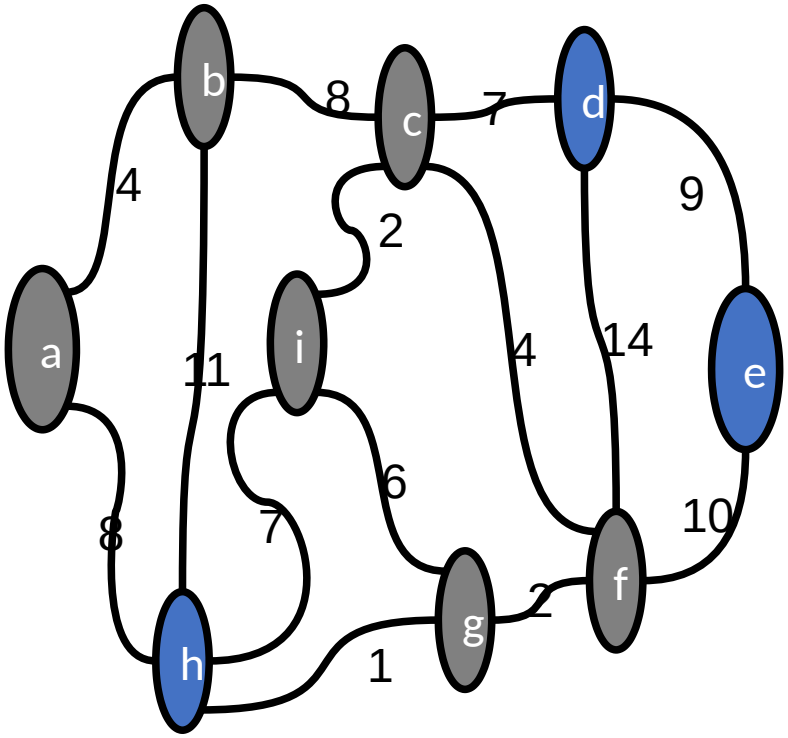
FILA				h	d	e			
chave				1	7	10			
				F		C			

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	7	2
π	-	a	b	c	f	c	f	i	c

	a	b	c	d	e	f	g	**h**	i
chave	0	4	8	7	10	4	2	**1**	2
π	-	a	b	c	f	c	f	**g**	c

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=h
    v=a: ~~a ∈ Q~~ AND (w(h,a) < chave(a)) → FALSE
    v=f: ~~f ∈ Q~~ AND (w(h,f) < chave(f)) → FALSE
    v=i: ~~i ∈ Q~~ AND (w(h,i) < chave(i)) → FALSE
    v=g: ~~g ∈ Q~~ AND (w(h,g) < chave(g)) → FALSE

FILA				~~h~~	d	e			
chave				~~1~~	7	10			
					F	C			

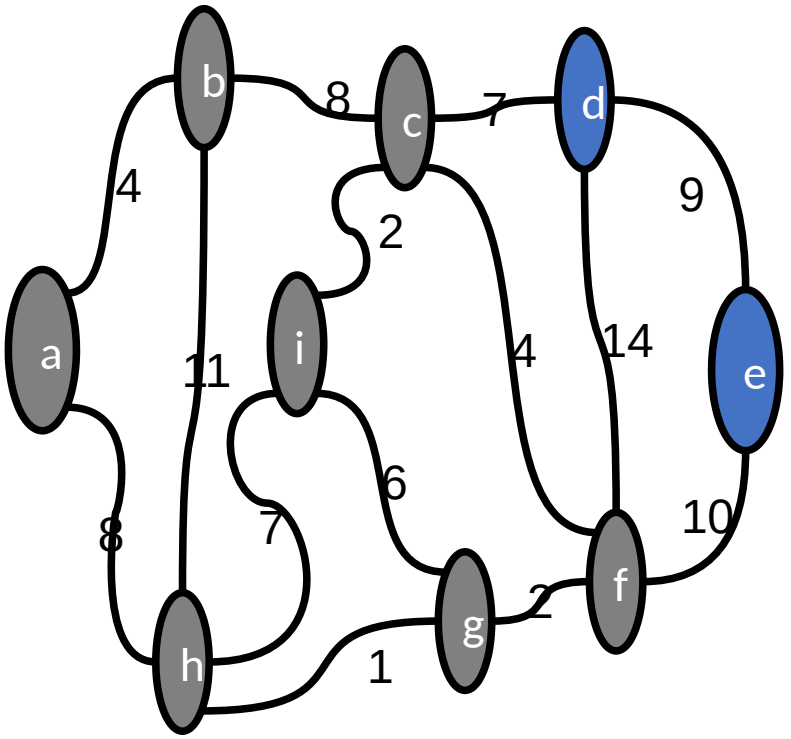
FILA					d	e			
chave					7	10			
					F	C			

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	-	a	b	c	f	c	f	g	c

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	-	a	b	c	f	c	f	g	c

MST – PRIM
(Cormen pg 461)



while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
    if v ∈ Q e w(u, v) < chave(v)
        π(v) = u
        chave(v) = w(u, v)

---

u=d

v=c: ~~c ∈ Q~~ AND (w(d,c) < chave(c)) → FALSE
v=f: ~~f ∈ Q~~ AND (w(d,f) < chave(f)) → FALSE
v=e: e ∈ Q AND (w(d,e) < chave(e)) → TRUE

FILA					~~d~~	e		
chave					~~7~~	10		
						F/C		

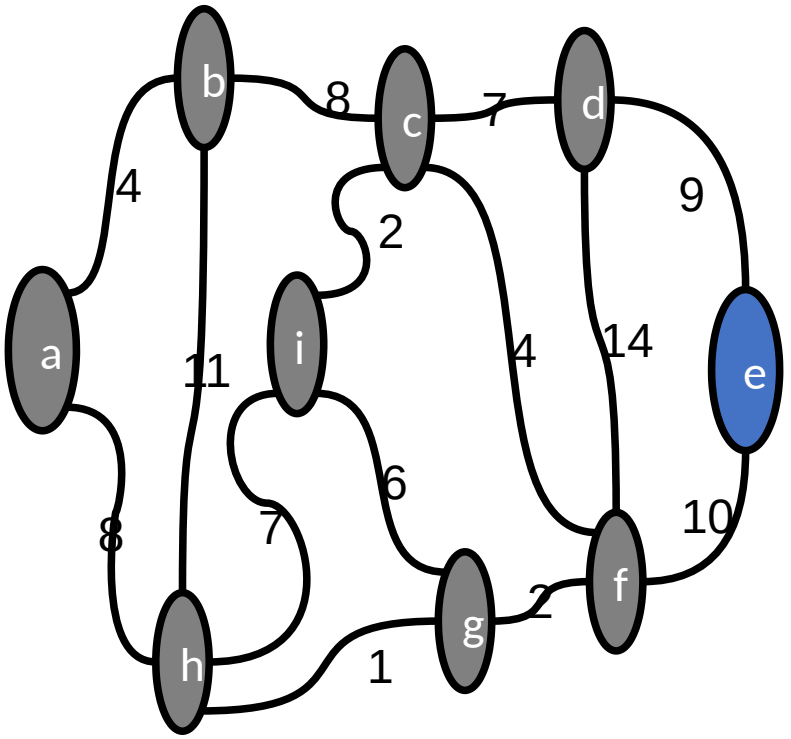
FILA						e		
chave						9		
						F/C		

Atualiza a fila

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	-	a	b	c	f	c	f	g	c

	a	b	c	d	**e**	f	g	h	i
chave	0	4	8	7	**9**	4	2	1	2
π	-	a	b	c	**d**	c	f	g	c

MST – PRIM
(Cormen pg 461)




```
while Q ≠ ∅
u = EXTRACT-MIN(Q)
for cada v ∈ G. Adj[u]
      if v ∈ Q    e    w(u, v) < chave(v)
            π(v) = u
            chave(v) = w(u, v)
----------------------------------------------------------
u=e
      v=d: ~~d ∈ Q~~ AND (w(e,d) < chave(d)) → FALSE
      v=f: ~~f ∈ Q~~ AND (w(e,f) < chave(f)) → FALSE
```

FILA						~~e~~			
chave						~~10~~			

FILA									
chave			FILA VAZIA critério de parada						

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	-	a	b	c	d	c	f	g	c

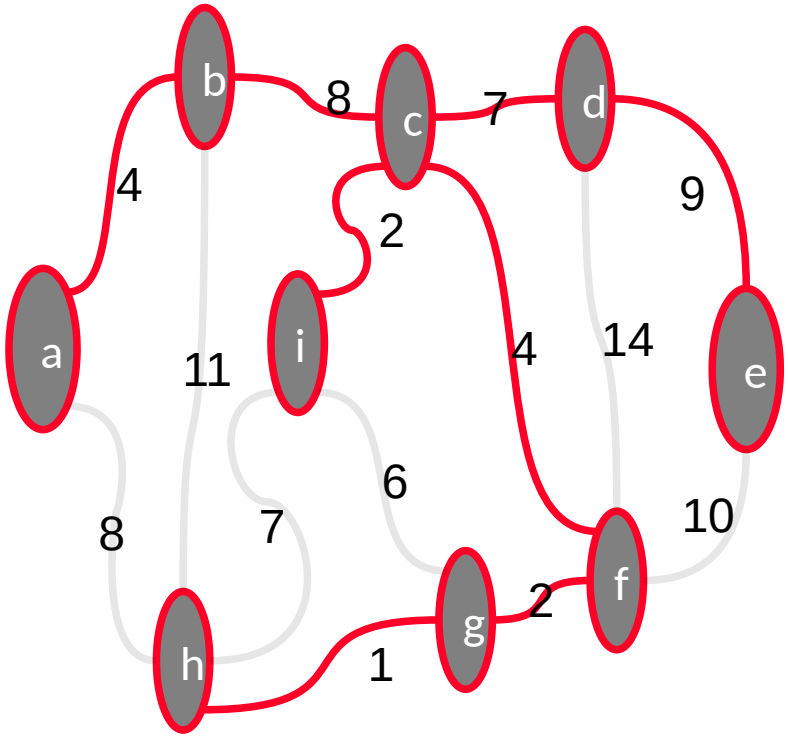
	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	-	a	b	c	d	c	f	g	c

MST – PRIM
(Cormen pg 461)

FILA									
chave		FILA VAZIA critério de parada							
frente									
cauda									

Montagem da Árvore:

	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	-	a	b	c	d	c	f	g	c

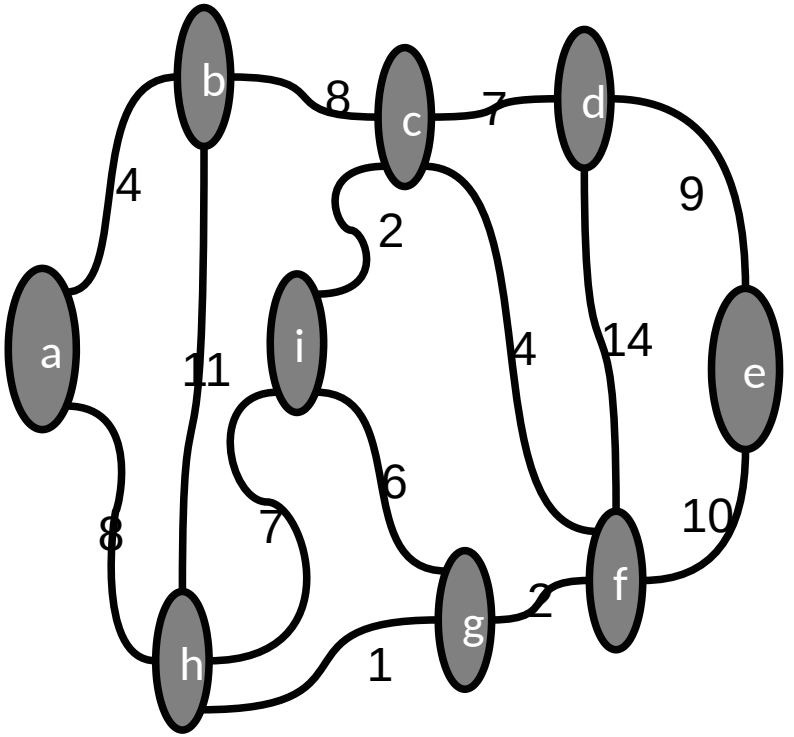


A={a, b, c, i, f, g, h, d, e}

Custo total = 37

MST – PRIM
(Cormen pg 461)

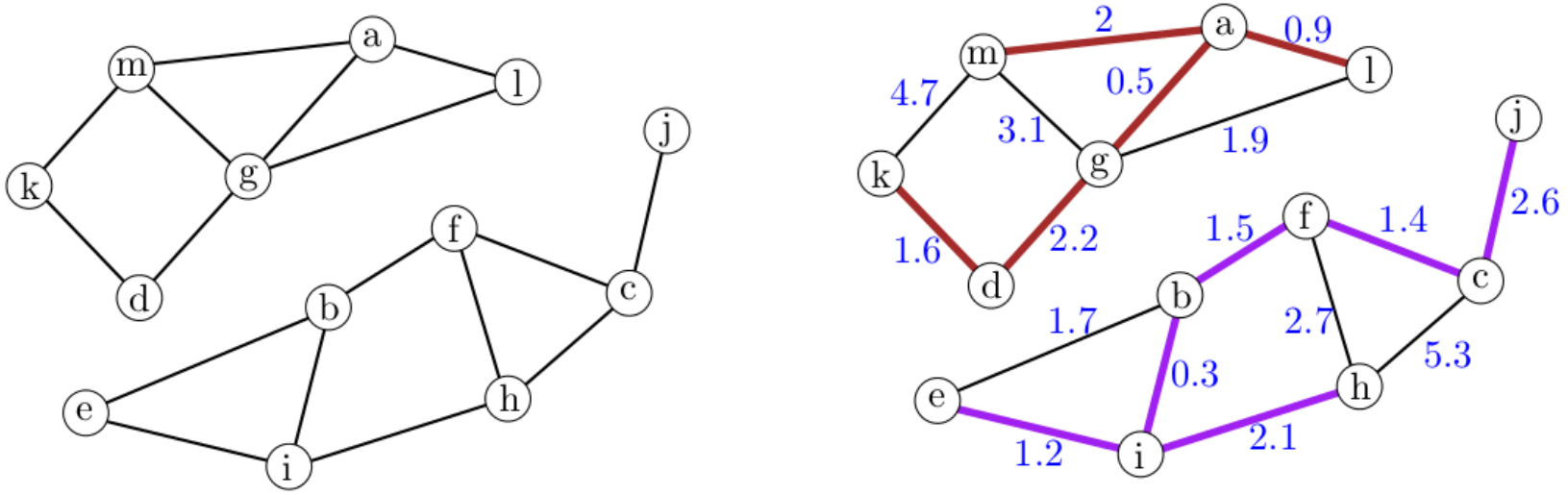
Execute o algoritmo de Kruskal para o mesmo grafo



Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

A floresta geradora mínima é uma generalização da árvore geradora mínima para grafos não conectados.

Para cada componente do grafo, pegue seu MST e a coleção resultante é uma floresta de extensão mínima



https://www.enseignement.polytechnique.fr/informatique/INF421/TD/TD5/images/trees.png

Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

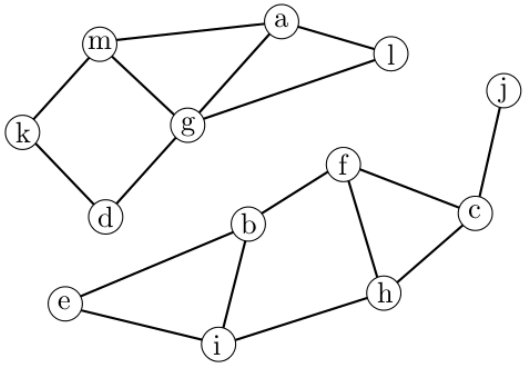
Não é necessário um outro algoritmo além dos usados para árvores geradoras – MST.

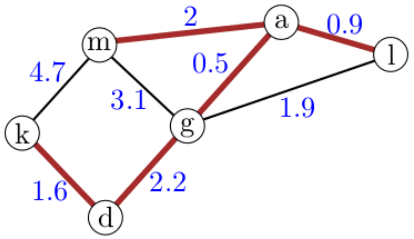
Por exemplo, você pode usar o algoritmo de Prim e se sua iteração parar, reinicie-a com um nó que ainda não está conectado (ou seja, com outra árvore da floresta).

https://stackoverflow.com/questions/43996928/what-algorithms-are-used-to-find-a-minimum-spanning-forest

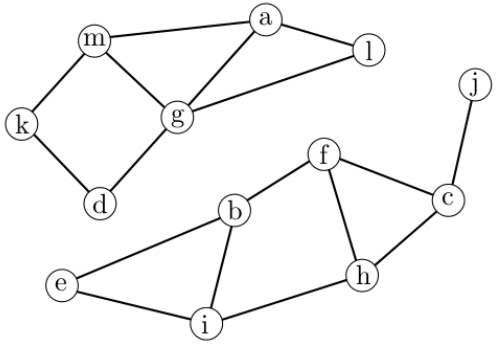
Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

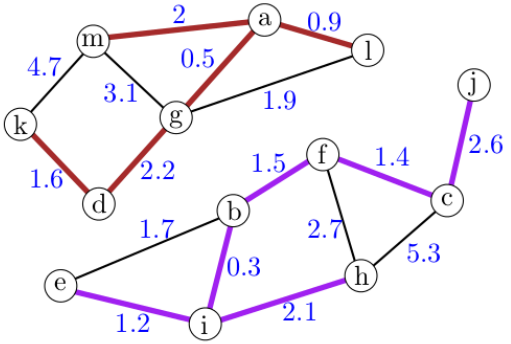
Pode usar o algoritmo de Prim e…





... se sua iteração parar, reinicie-a com um nó que ainda não está conectado (ou seja, com outra árvore da floresta).





https://www.enseignement.polytechnique.fr/informatique/INF421/TD/TD5/images/trees.png

Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

Mas, se você resolver utilizar o algoritmo de Kruskal, você obterá todas as arestas mais baratas em cada subgrafo/árvore de extensão mínima de sua floresta (agora também de custo mínimo).

Conclusão: os algoritmos usados para encontrar uma floresta geradora mínima são os mesmos usados para encontrar uma árvore geradora mínima - em alguns casos com adaptações e em alguns casos sem elas.

https://stackoverflow.com/questions/43996928/what-algorithms-are-used-to-find-a-minimum-spanning-forest

Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

Dijkistra é o mesmo que Prim?

Ambos são *Greedy Algorithms*, porém:

Dijkstra seleciona como próxima aresta aquela que sai da árvore para um nó ainda não escolhido mais próximo do nó **inicial**. Então, com esta escolha, as distâncias são recalculadas.

Prim escolhe como aresta a mais curta que sai da árvore construída até agora.

Portanto, ambos os algoritmos escolheram uma “borda mínima”. A principal diferença é o valor escolhido para ser mínimo. Para Dijkstra é o comprimento do caminho **completo** do nó inicial ao nó candidato, para Prim é apenas o peso daquela única aresta.

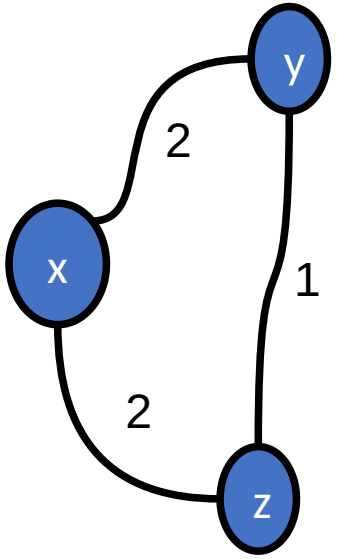
https://cs.stackexchange.com/questions/18797/minimum-spanning-tree-vs-shortest-path

Minimum Spanning Forest - MSF
Floresta geradora de custo Mínimo

Dijkistra é o mesmo que Prim?

O exemplo mais simples que mostra um comportamento diferente é um triângulo x,y,z com arestas $\{x,y\}$ e $\{x,z\}$ de comprimento 2, enquanto $\{y,z\}$ tem comprimento 1. Começando em x, Dijkstra escolherá $\{x\ ,y\}$ **ou** $\{x,z\}$ (dando dois caminhos de comprimento 2) enquanto Prim escolherá $\{x,y\}$ **e** $\{y,z\}$ (dando árvore geradora de peso 3).

https://cs.stackexchange.com/questions/18797/minimum-spanning-tree-vs-shortest-path



TEG

Bibliografia

Básica

LUCCHESI, C. L. et alli. Aspectos Teóricos da Computação, Parte C: Teoria dos Grafos, projeto Euclides, 1979.

SANTOS, J. P. O. et alli. Introdução à Análise Combinatória. UNICAMP; 1995.

SZWARCFITER, J. L. Grafos e Algoritmos Computacionais. Campus, 1986.

GERSTING, Judith L. Fundamentos Matemáticos para a Ciência da Computação. Rio de Janeiro. 3a Ed. Editora.

Complementar:

1.) CORMEN, T. Introduction to Algorithms, third edition, MIT press, 2009

2.) ROSEN, K. Discrete Mathematics and its applications, seventh edition, McGraw Hill, 2011.

3.) WEST, Douglas, B. Introduction to Graph Theory, second edition, Pearson, 2001.

4.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984.

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

6.) GOLDBARG, M., GOLDBARG E., Grafos: Conceitos, algoritmos e aplicações. Editora Elsevier, 2012.

7.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984

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

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

10.) FURTADO, A. L. Teoria de grafos. Rio de janeiro. Editora LTC. 1973.

11.) WILSON, R.J. Introduction to Graph Theory. John Wiley & Sons Inc., 1985

12.) BOAVENTURA NETTO , P. O. Grafos: Teoria, Modelos, Algoritmos. Edgard Blucher, SP, quinta edição

Tutoriais, artigos, notas de aula...