UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE RUSSAS

Algoritmos em Grafos

Aula 14: Árvore Geradora Mínima(Prim)

Professor Pablo Soares

2020.1

Árvore Geradora Mínima

Construindo o Conhecimento

Árvore Geradora Mínima

Construindo o Conhecimento

Árvore

Árvore Geradora Mínima

Construindo o Conhecimento

Árvore

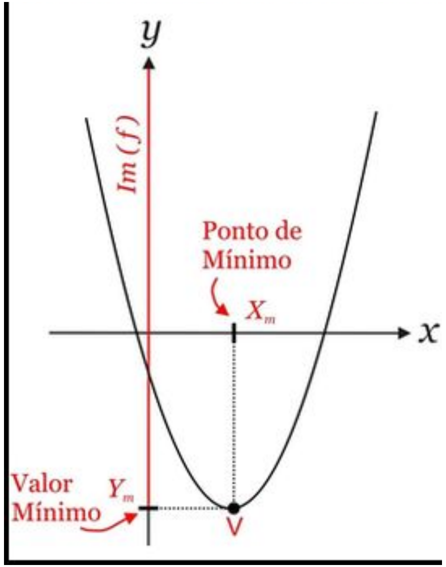
Gerador

Árvore Geradora Mínima

Construindo o Conhecimento

Árvore

Gerador



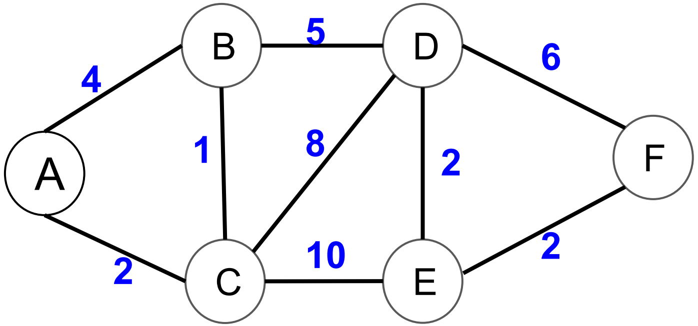
Mínimo

Árvore Geradora Mínima

Construindo o Conhecimento(Era isso mesmo?)

Árvore Geradora Mínima

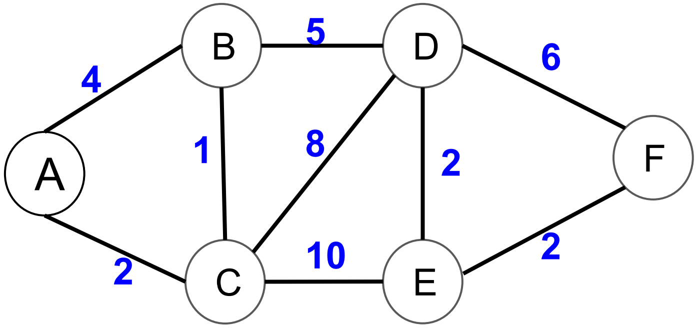
Construindo o Conhecimento(em Grafos)



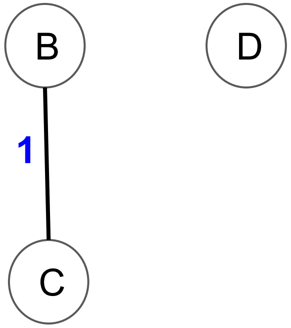
Grafo G

Árvore Geradora Mínima

Construindo o Conhecimento(em Grafos)



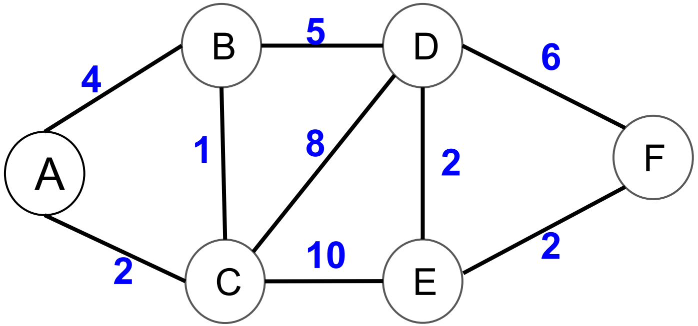
Grafo G



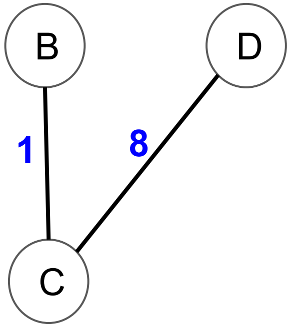
É árvore???

Árvore Geradora Mínima

Construindo o Conhecimento(em Grafos)



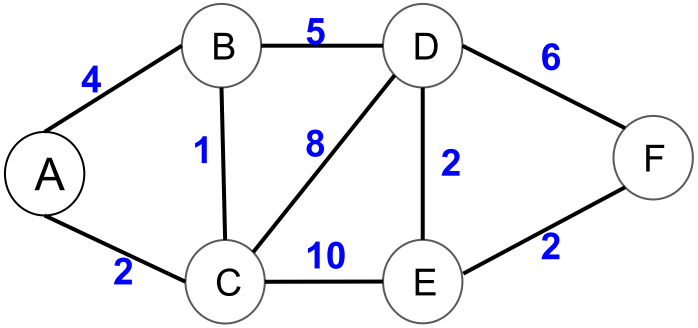
Grafo G



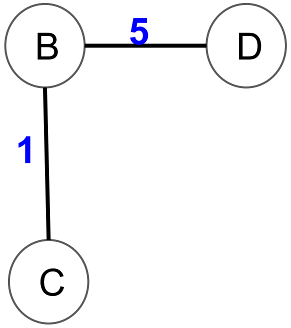
Agora sim

Árvore Geradora Mínima

Construindo o Conhecimento(em Grafos)



Grafo G

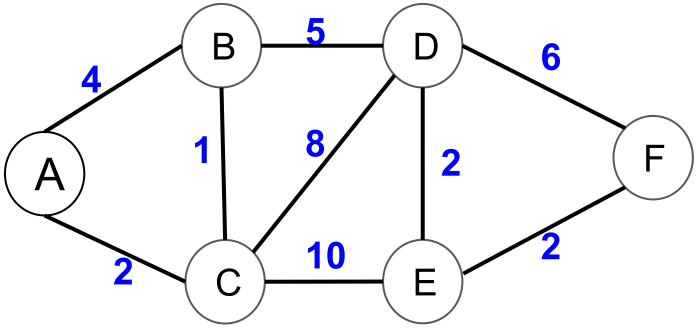


Poderia ser assim também

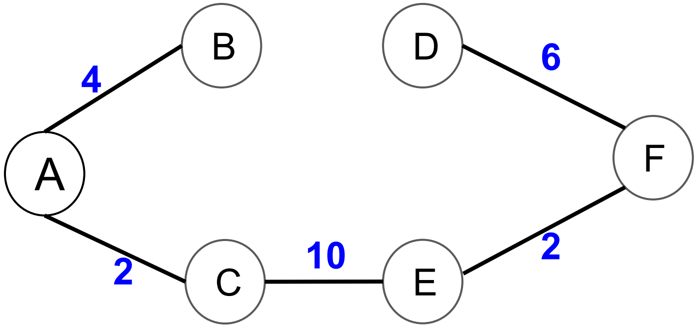
Ela é geradora?????

Árvore Geradora Mínima

Construindo o Conhecimento(em Grafos)



Grafo G

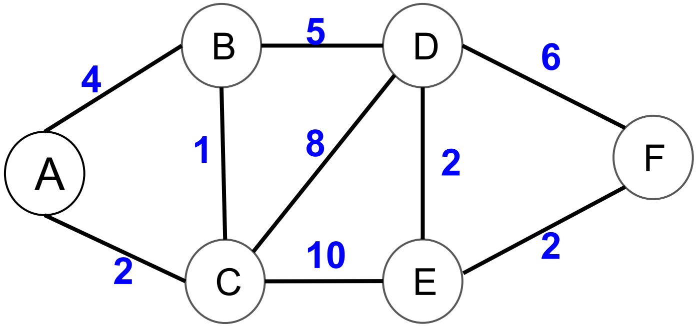


Árvore Geradora

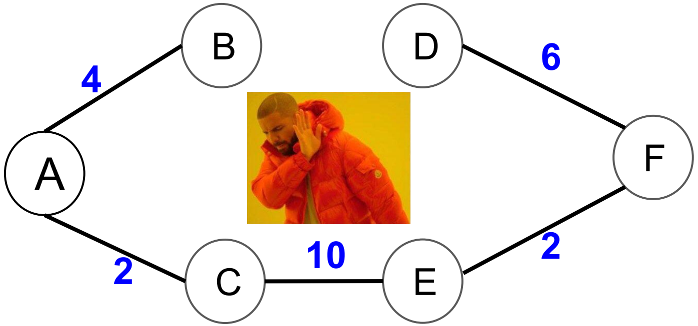
Ela é mínima?????

Árvore Geradora Mínima

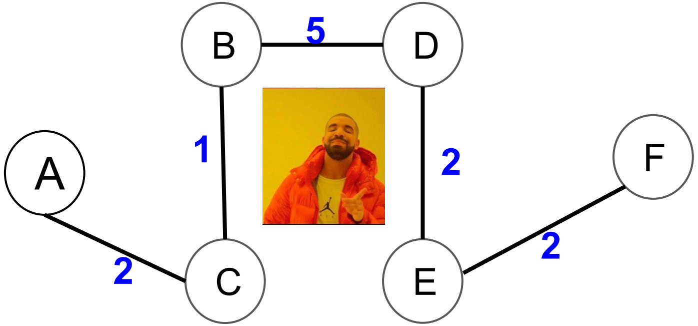
Construindo o Conhecimento(em Grafos)



Grafo G



Árvore Geradora



Árvore Geradora Mínima

Árvore Geradora Mínima (Motivação)

1. Projeto de redes conectando N nós
   a. $G = (V, E) \rightarrow$ grafo conexo, não direcionado;
      i. V → Nós (PCs);
      ii. E → Links.
         - Cada link com custo $w(u, v) \rightarrow$(peso da aresta).

Árvore Geradora Mínima (Motivação)

1. Solução:
   a. Subconjunto **T** = (*V*, ***E’***)
      i. Acíclico;
      ii. Conexo;
      iii. ***E’*** $\subseteq$ *E, tal que*;

Peso(**T**) $=\sum w(u, v), \forall (u, v) \in$ ***E’***

seja minimizado

Árvore Geradora Mínima (Guloso)

- Algoritmo Guloso para encontrar
  - |V| - 1 arestas de ***T***

AGM-Genérico(G, w)

1. ***T*** $\leftarrow \varnothing$
2. ***enquanto*** ***T*** *não formar uma árvore geradora mínima*
3.   *Encontre uma <u>aresta (u, v) que é segura</u> para* ***T***
4.   ***T*** $\leftarrow$ ***T*** $\cup$ *(u, v)*
5. ***fimenquanto***
6. ***Retorne T***

Fim.

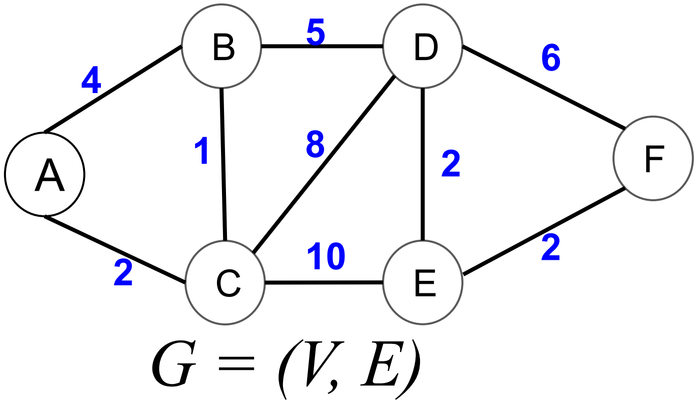
Árvore Geradora Mínima (Propriedades)

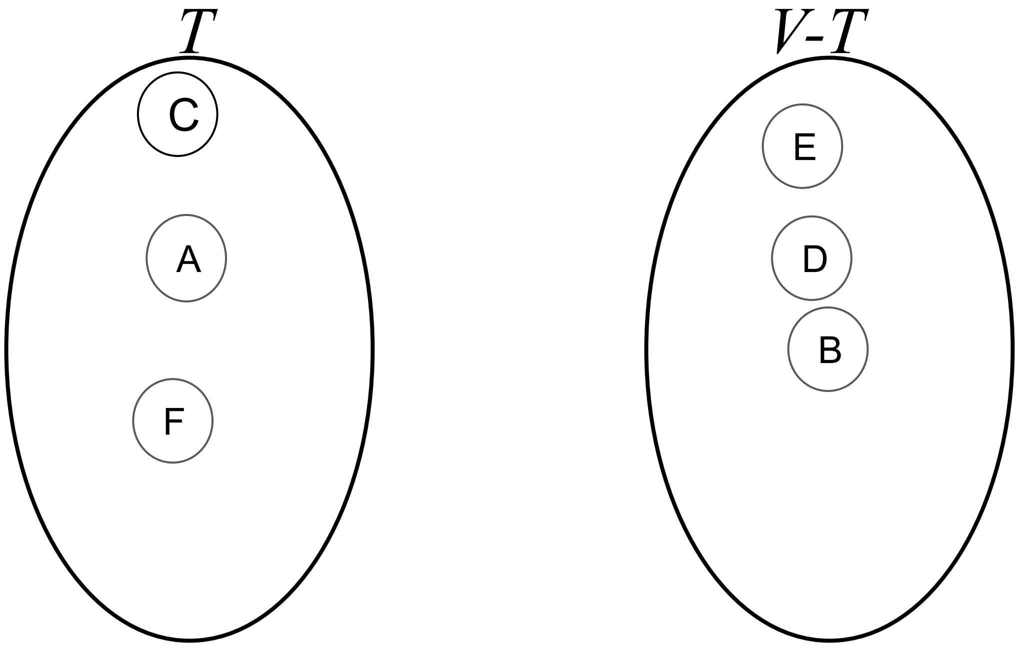
Teorema: Seja $G = (V, E)$ um grafo conexo não orientado e ponderado. Seja T um subconjunto de E que está incluído em alguma árvore geradora mínima correspondente a G, seja $(T, V\text{-}T)$ **qualquer corte de *G*** que **respeita *T*** e seja (u, v) uma **aresta leve** cruzando $(T, V\text{-}T)$. Então, a aresta (u, v) é **segura** para T.

1. Corte de G;
   a. Arestas que cruzam o corte.
2. Corte que respeita certo subconjunto de E;
3. Aresta Leve.

Árvore Geradora Mínima (Corte de G)

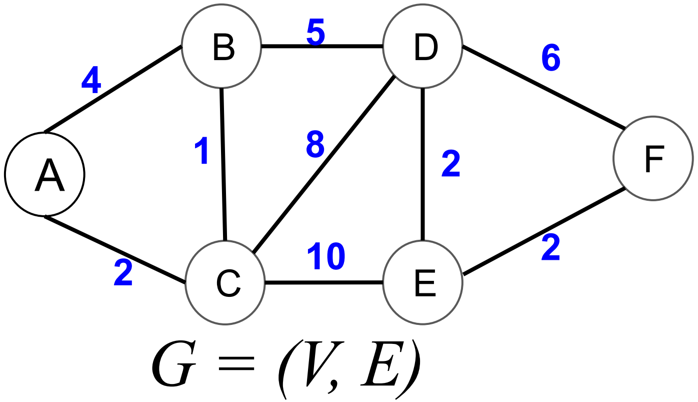
1. Um corte (T, V-T) de um grafo não orientado G é uma partição de V;

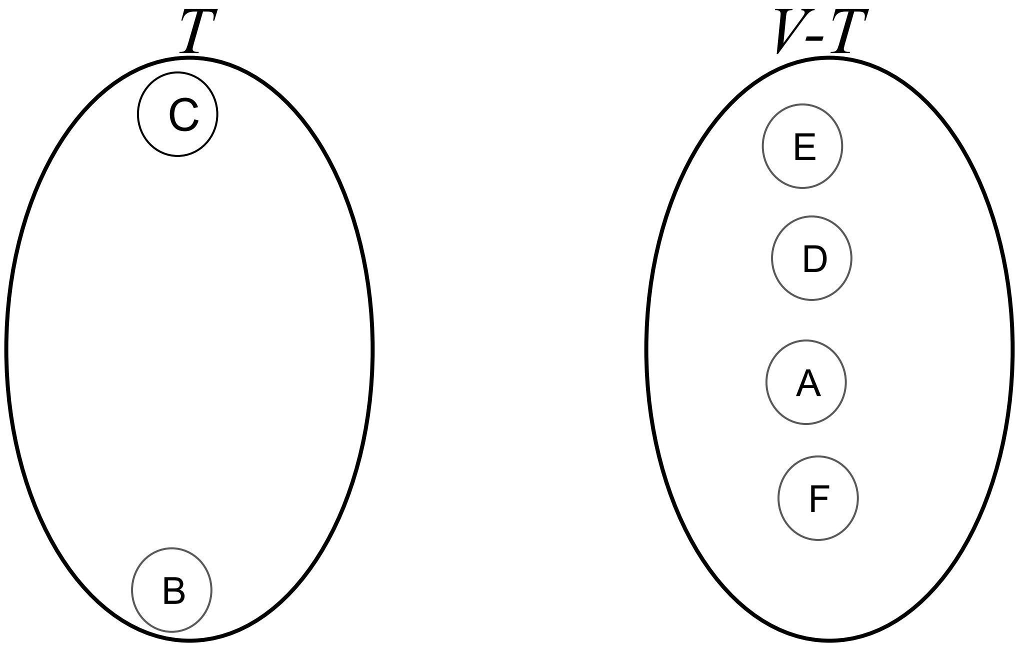




Árvore Geradora Mínima (Corte de G)

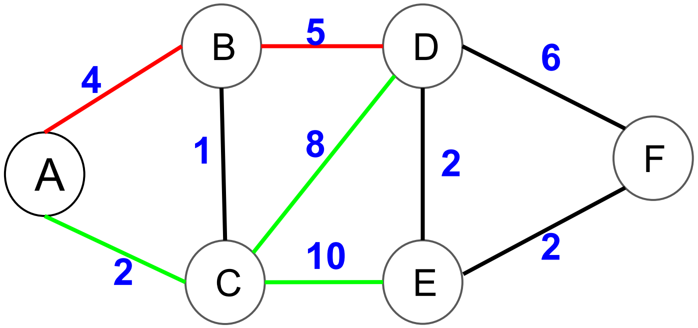
1. Um corte (T, V-T) de um grafo não orientado G é uma partição de V;



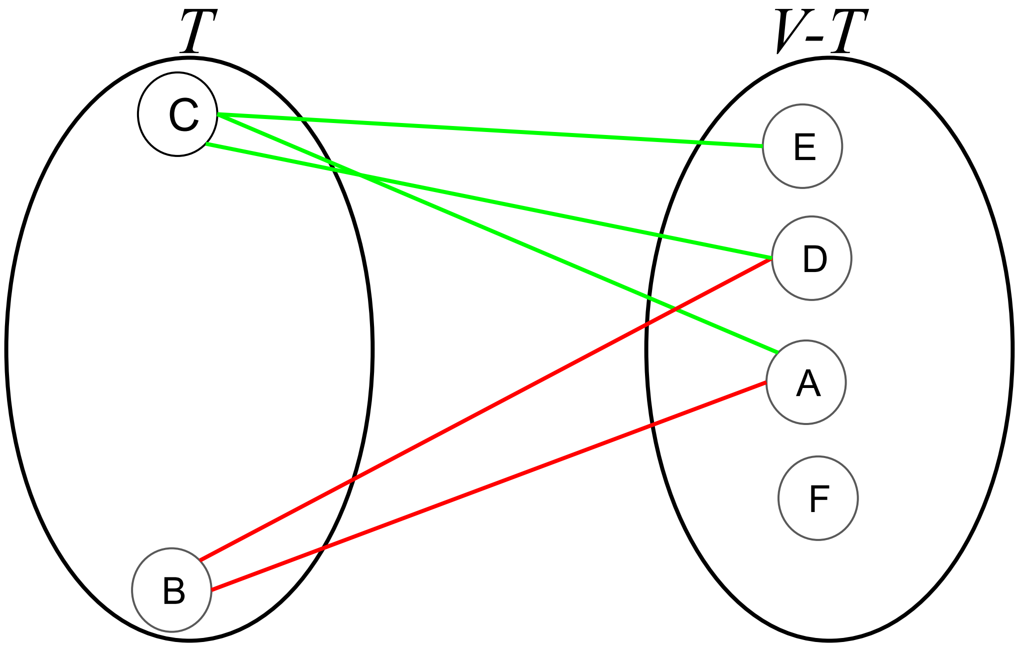


Árvore Geradora Mínima (Corte de G)

1. Um corte (*T*, *V*-*T*) de um grafo não orientado *G* é uma partição de *V*;
   a. Arestas que **cruzam** o corte

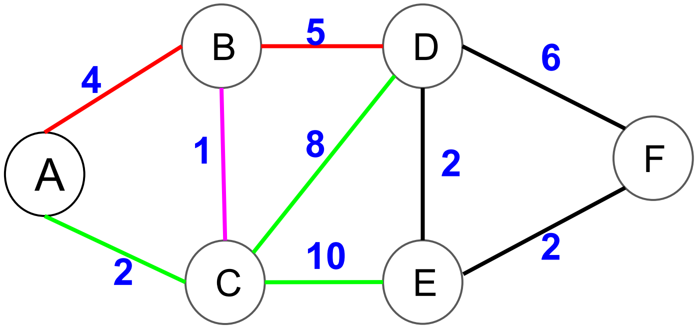


$G = (V, E)$

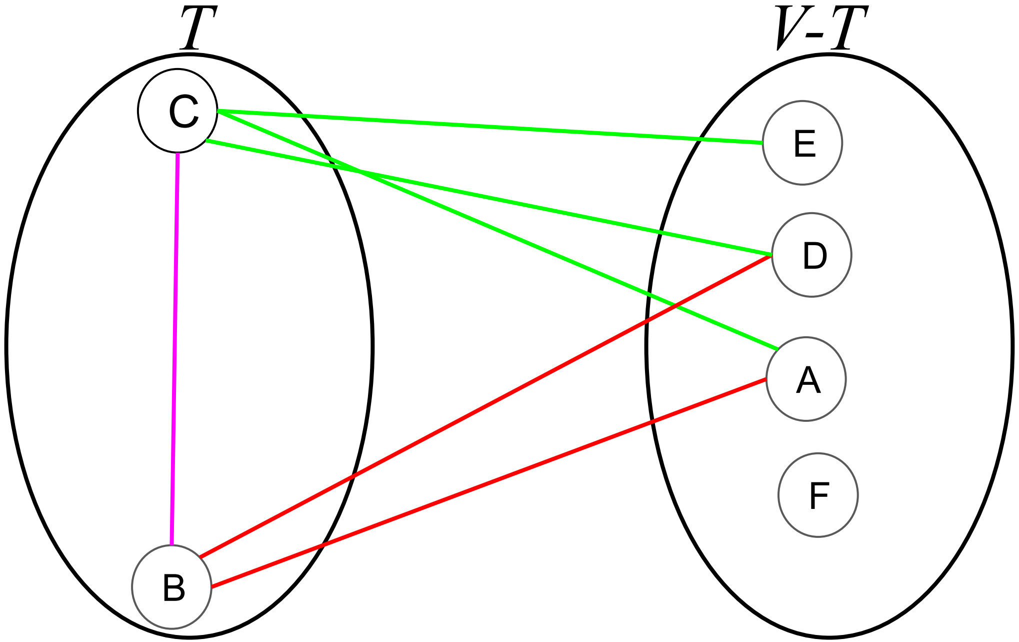


Árvore Geradora Mínima (Respeita subconjunto)

2. Corte que respeita certo subconjunto de E;

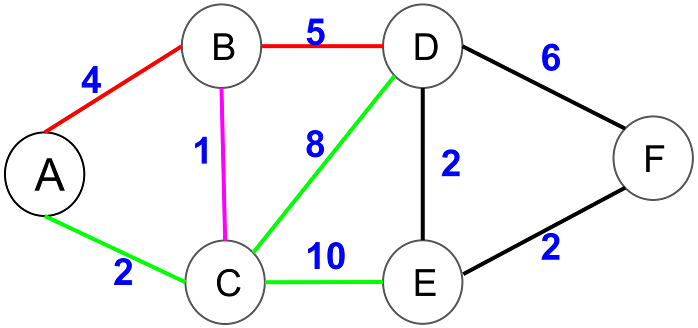


$G = (V, E)$

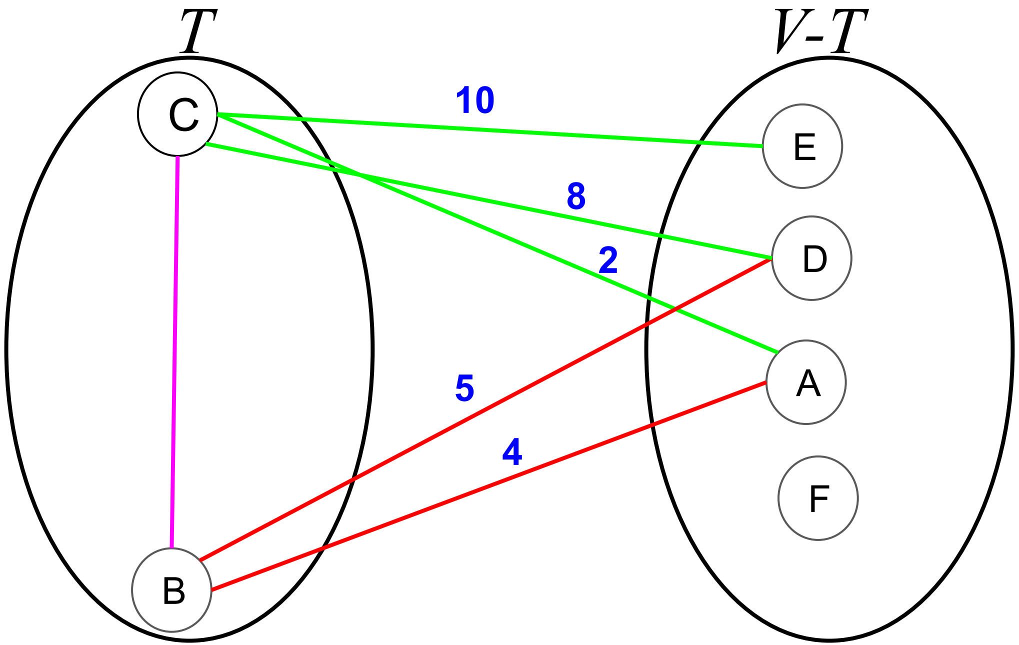


Árvore Geradora Mínima (Aresta Leve)

3. Aresta Leve cruzando o corte
   - Seu peso é o <u>mínimo</u> de todas que cruzam o corte

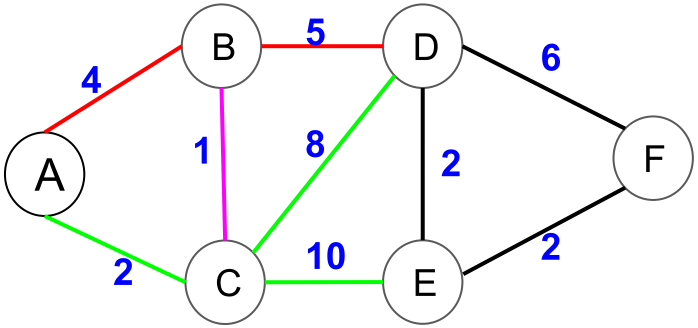


$G = (V, E)$

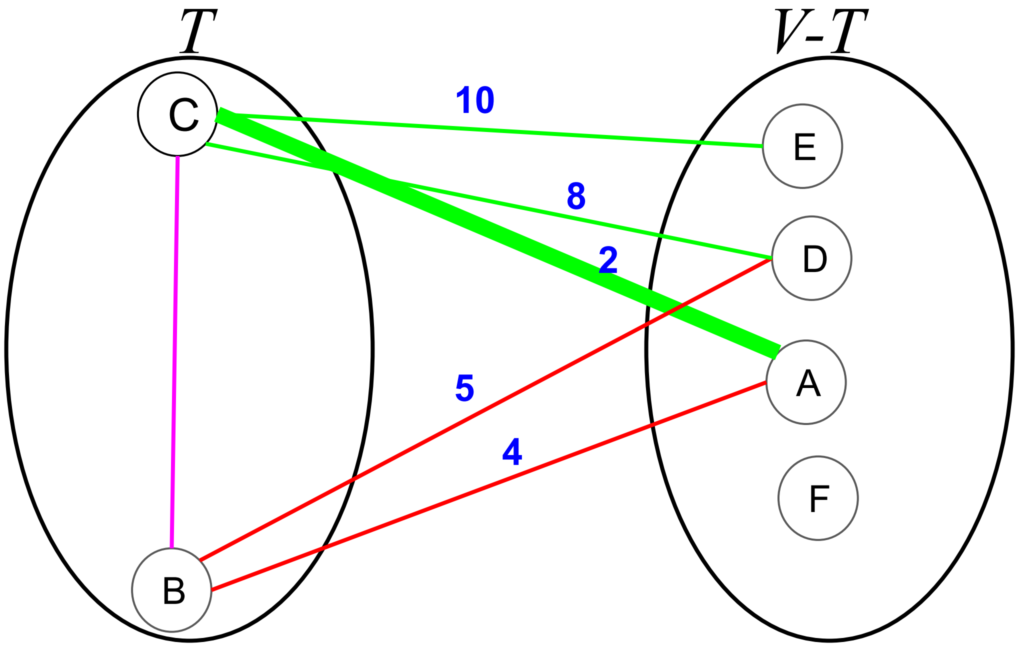


Árvore Geradora Mínima (Aresta Segura)

3. Aresta Leve cruzando o corte → Aresta Segura
   - Seu peso é o mínimo de todas que cruzam o corte

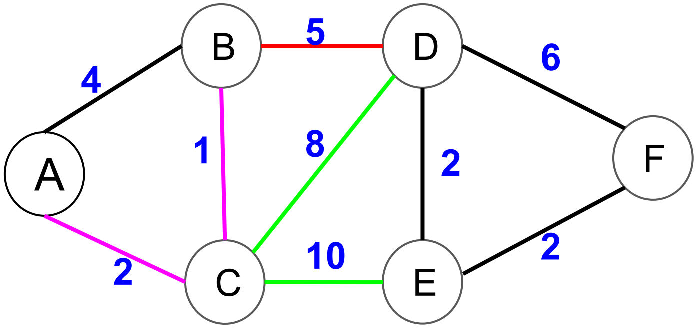


$G = (V, E)$

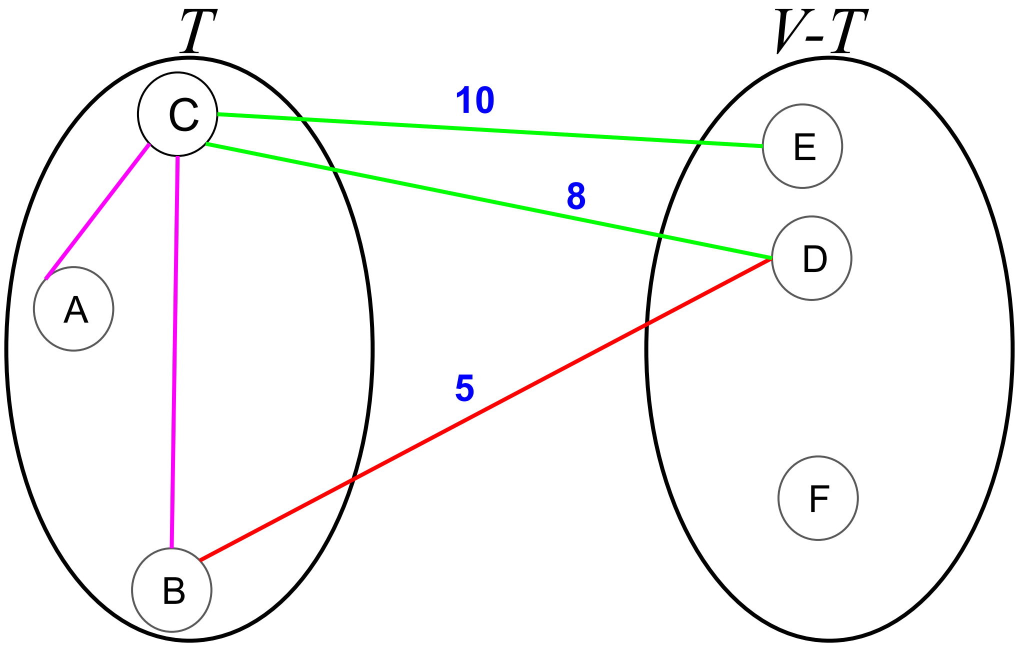


Árvore Geradora Mínima (Aresta Segura)

<u>Aresta Segura</u> adicionada ao subconjunto T



$G = (V, E)$



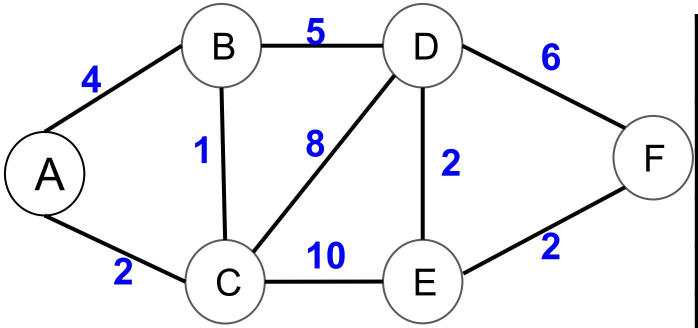
Pseudocódigo do Algoritmo de Prim

```
Prim(G, w, v₀)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v₀] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```

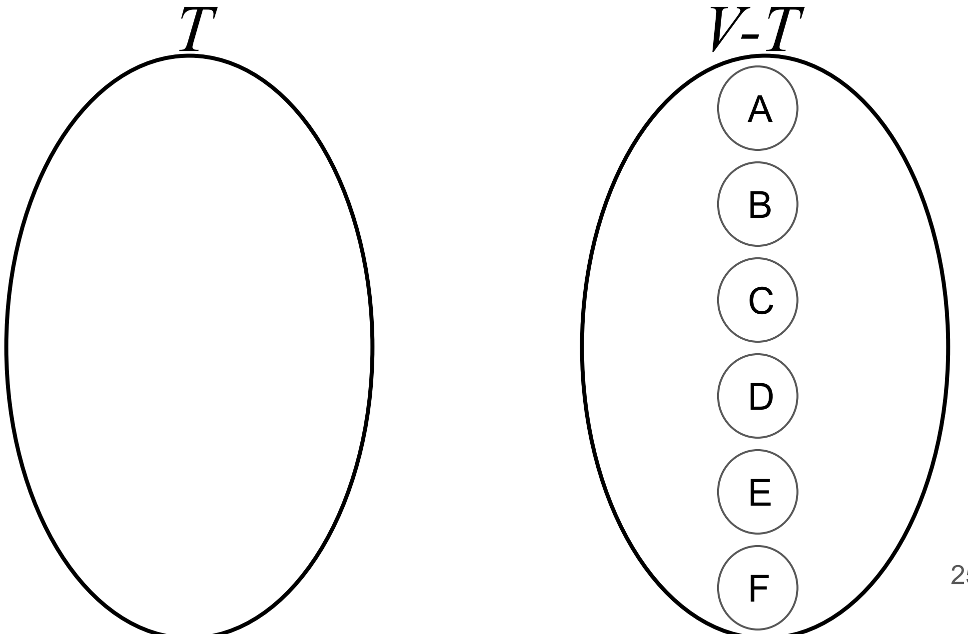
- Estruturas auxiliares:
  - Fila de Prioridade → Q
    - heap-min
  - Vetores
    - Chave → peso mínimo da aresta
    - $\pi \rightarrow$ *Predecessor*

- *Complexidade do Tempo de Execuçã*
  - $O(|V|+|E|)lg|V|$

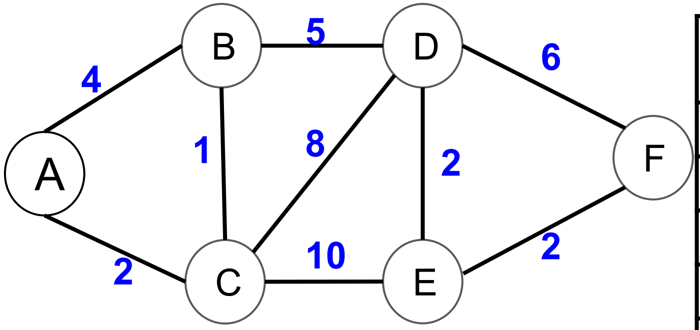
```
Prim(G, w, v0)
para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
fimpara
chave[v0] ← 0
Q←FilaDePrioridade( )
InsereNaFila(Q, V[G])
enquanto Q ≠ vazio
    u ←RemoveDaFila(Q)
    para cada vértice v ∈ L.adj(u)
        se v ∈ Q && chave[v] > w(u, v)
            chave[v] ← w(u, v)
            π[v] ←u
        fimse
    fimpara
fimenquanto
Fim.
```



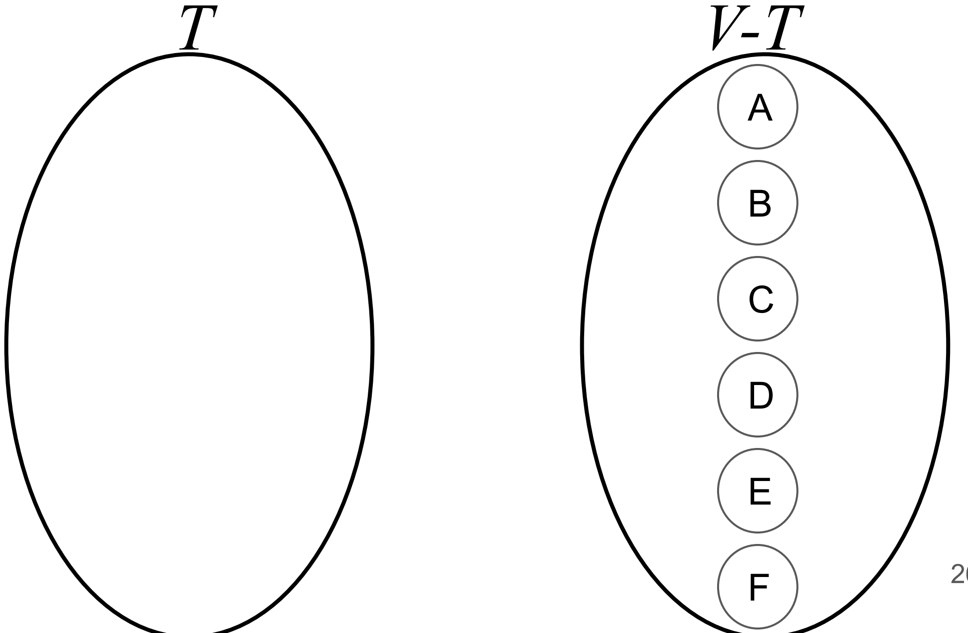
Vértice	*π*	*Chave*
A		
B		
C		
D		
E		
F		



```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```

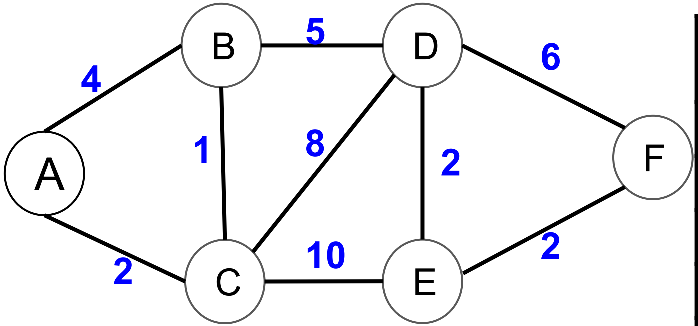


Vértice	*π*	*Chave*
A	N	∞
B	N	∞
C	N	*0*
D	N	∞
E	N	∞
F	N	∞

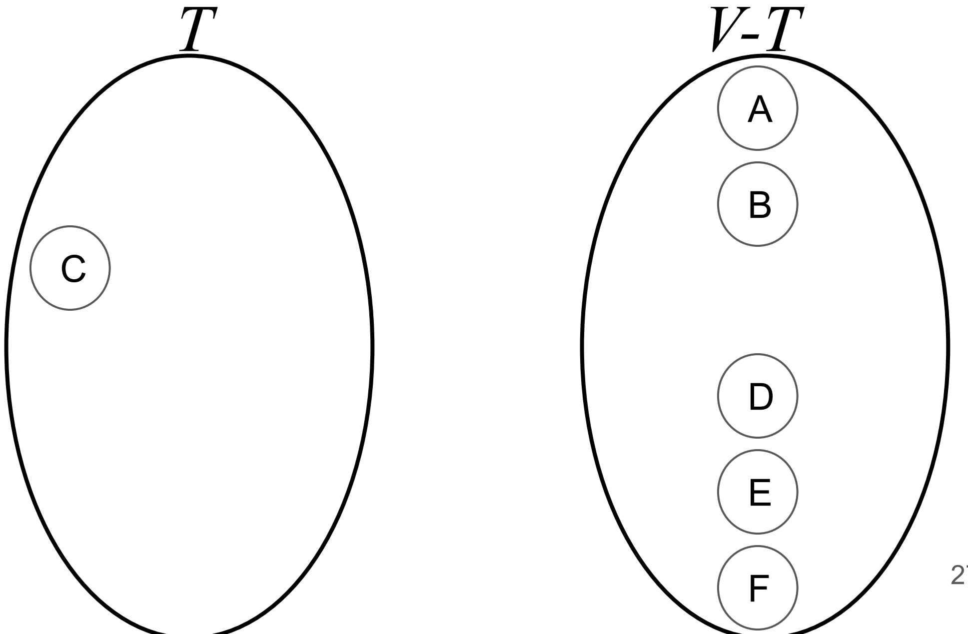


Prim(G, w, v_0)

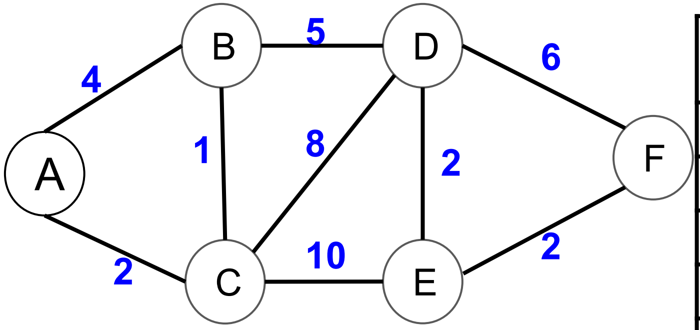
```
para cada vértice u ← V[G]
    chave[u] ←∞
    π[u] ← NULL
fimpara
chave[v0] ← 0
Q←FilaDePrioridade( )
InsereNaFila(Q, V[G])
enquanto Q ≠ vazio
    u ←RemoveDaFila(Q)
    para cada vértice v ∈ L.adj(u)
        se v ∈ Q && chave[v] > w(u, v)
            chave[v] ← w(u, v)
            π[v] ←u
        fimse
    fimpara
fimenquanto
Fim.
```



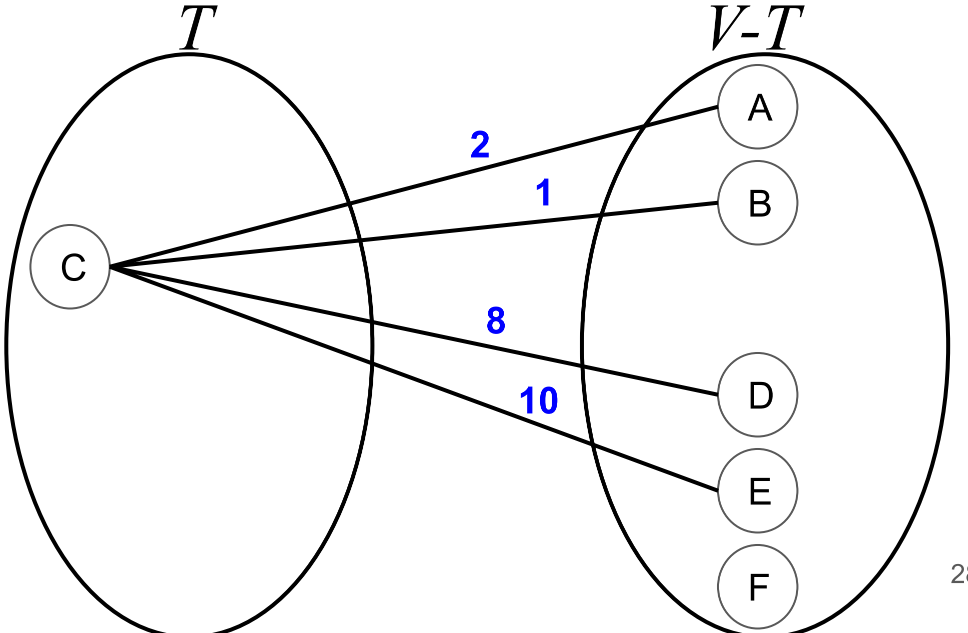
Vértice	π	Chave
A	N	∞
B	N	∞
C	N	0
D	N	∞
E	N	∞
F	N	∞



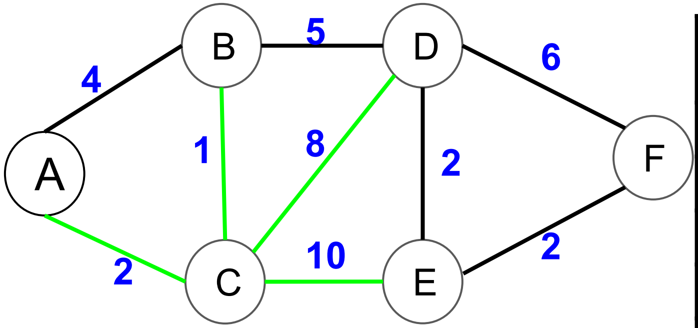
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



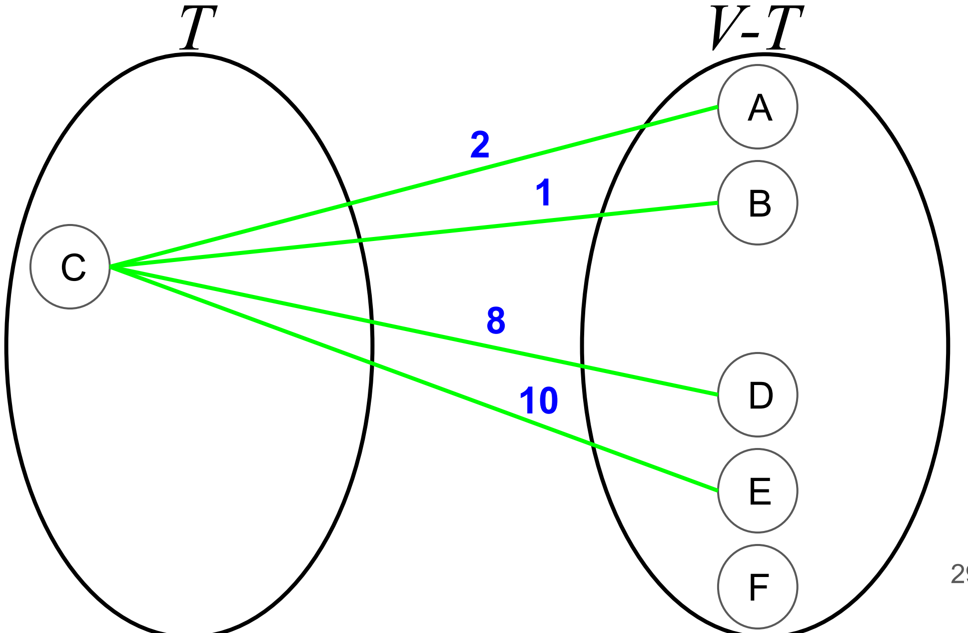
Vértice	π	Chave
A	N	∞
B	N	∞
C	N	0
D	N	∞
E	N	∞
F	N	∞



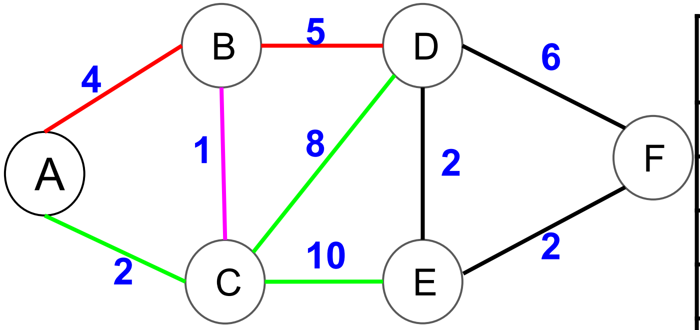
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



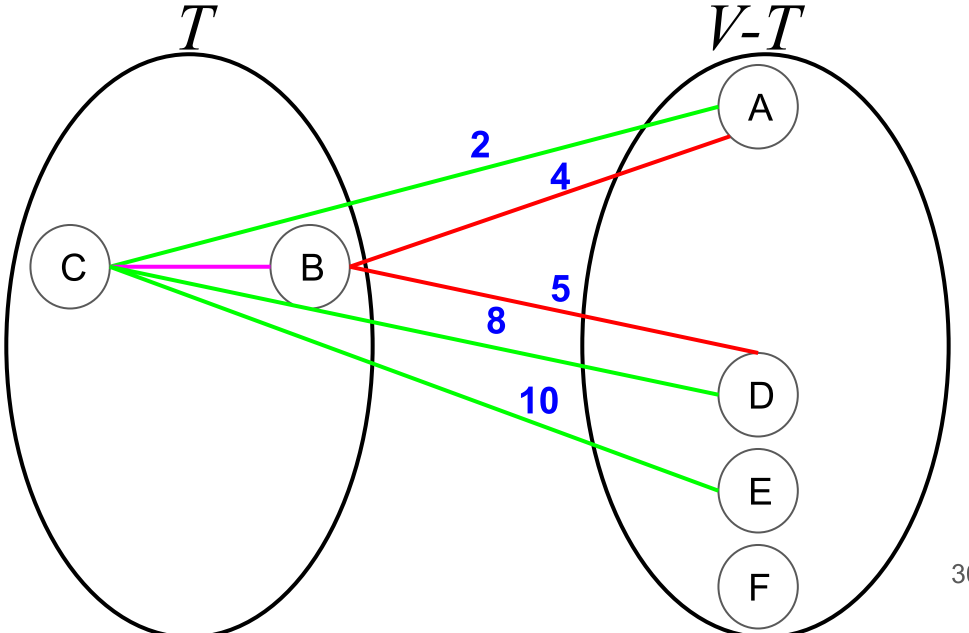
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	C	8
E	C	10
F	N	∞



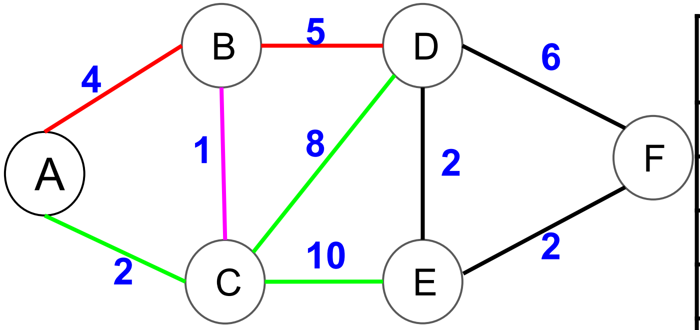
```
Prim(G, w, v0)
para cada vértice u ← V[G]
      chave[u] ←∞
      π[u] ← NULL
fimpara
chave[v0] ← 0
Q←FilaDePrioridade( )
InsereNaFila(Q, V[G])
enquanto Q ≠ vazio
    u ←RemoveDaFila(Q)
    para cada vértice v ∈ L.adj(u)
        se v ∈ Q && chave[v] > w(u, v)
            chave[v] ← w(u, v)
            π[v] ←u
        fimse
    fimpara
fimenquanto
Fim.
```



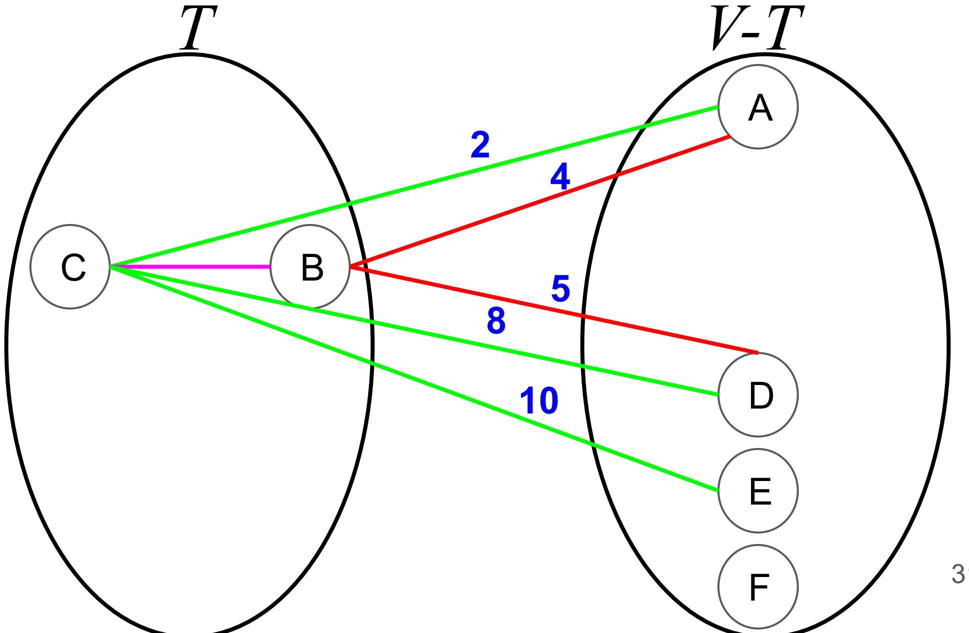
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	C	8
E	C	10
F	N	∞



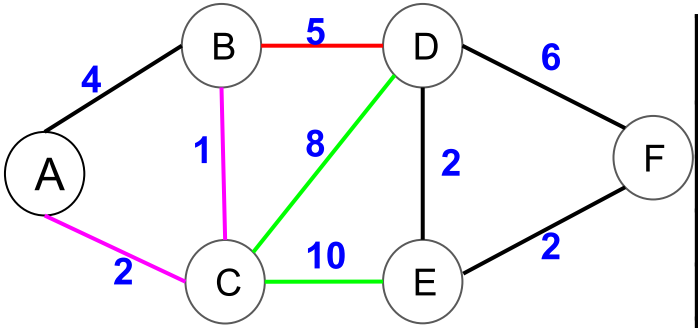
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



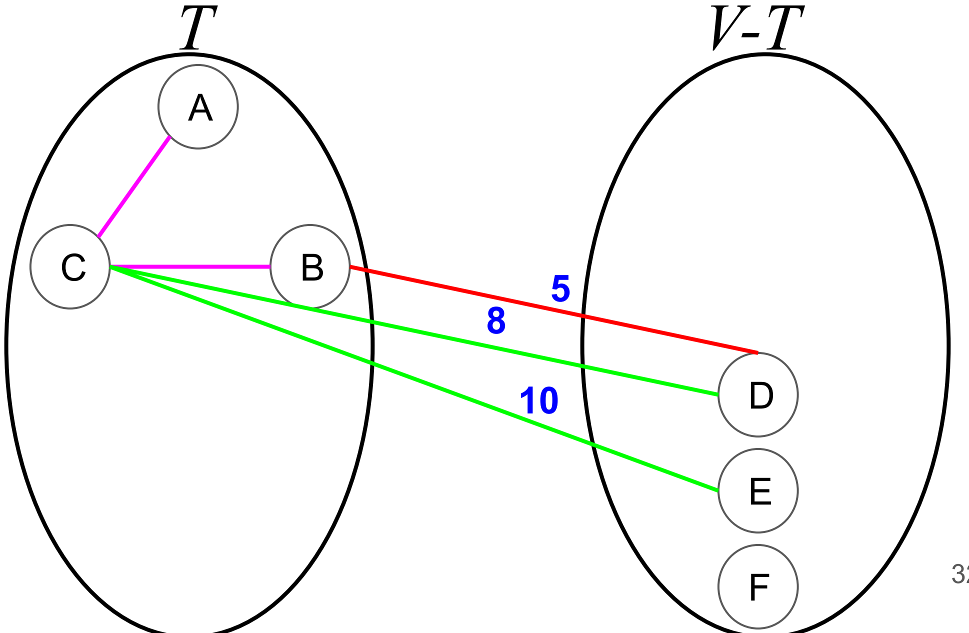
Vértice	π	*Chave*
A	C	2
B	C	1
C	N	0
D	B	5
E	C	10
F	N	∞



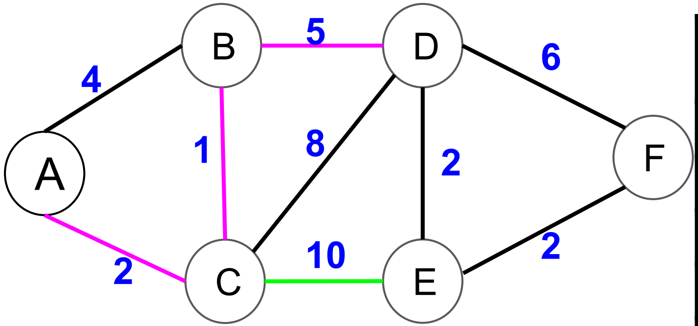
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



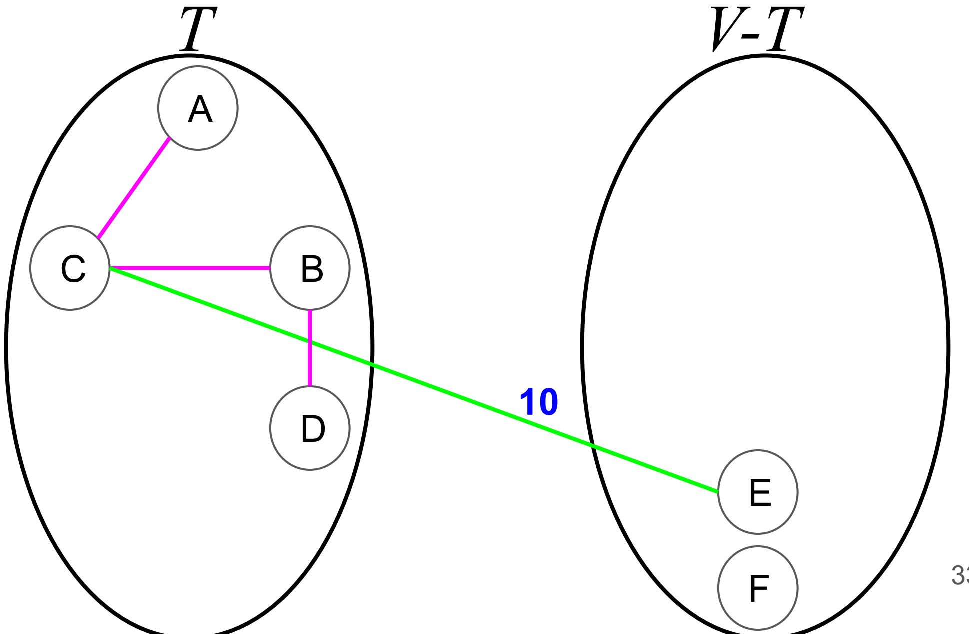
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	B	5
E	C	10
F	N	∞



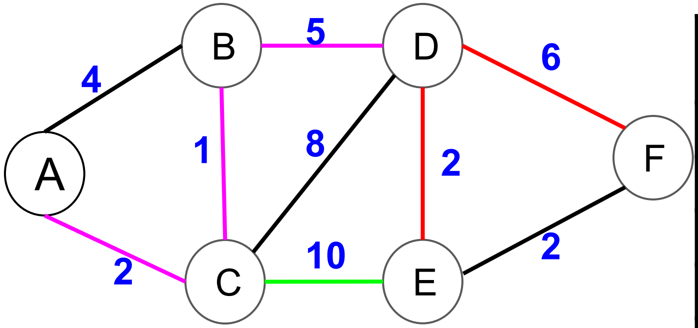
```
Prim(G, w, v_0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v_0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



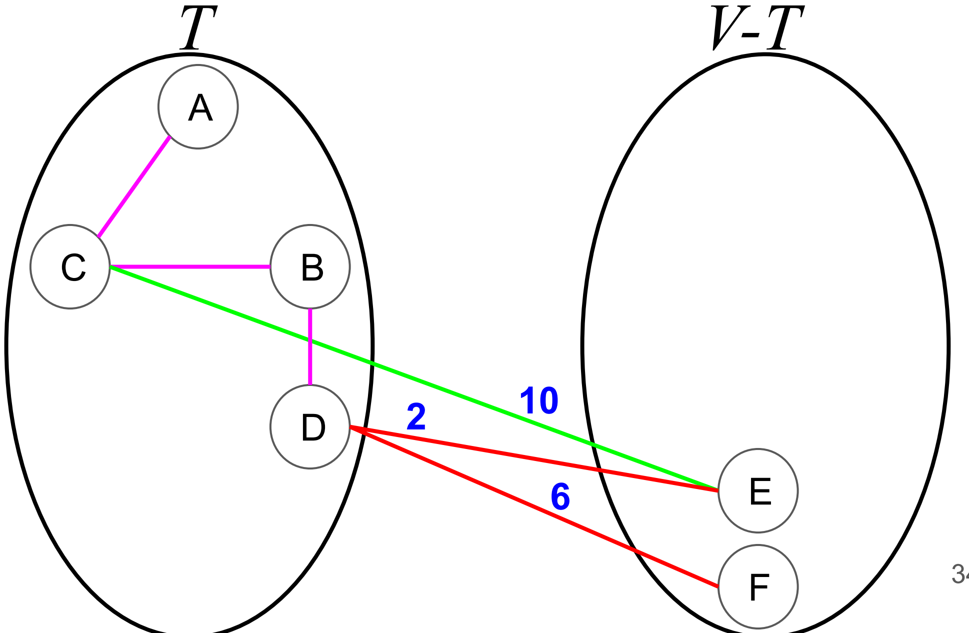
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	B	5
E	C	10
F	N	∞



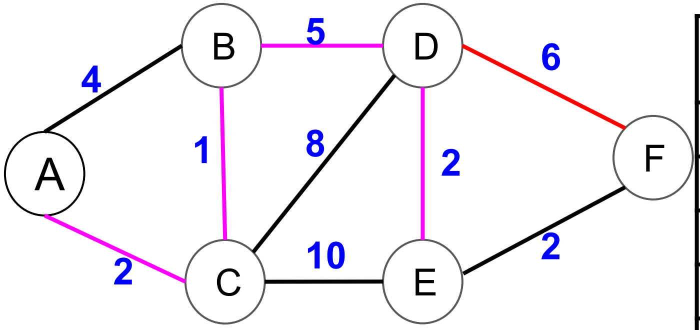
```
Prim(G, w, v_0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v_0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



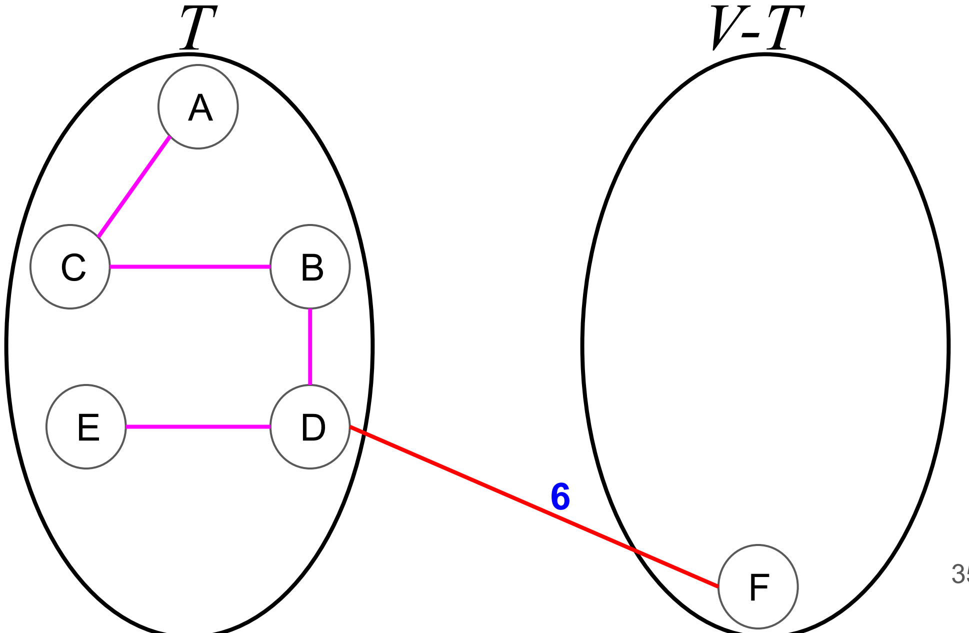
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	B	5
E	D	2
F	D	6



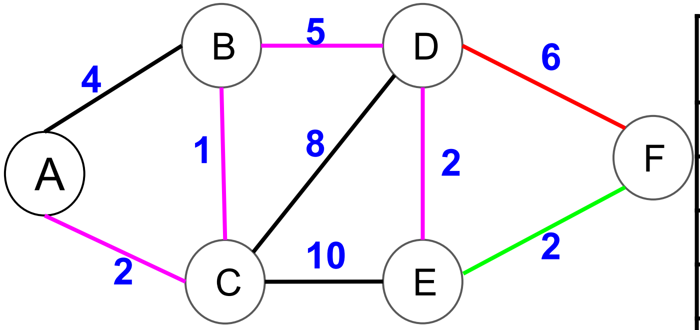
```
Prim(G, w, v0)
para cada vértice u ← V[G]
    chave[u] ←∞
    π[u] ← NULL
fimpara
chave[v0] ← 0
Q←FilaDePrioridade( )
InsereNaFila(Q, V[G])
enquanto Q ≠ vazio
    u ←RemoveDaFila(Q)
    para cada vértice v ∈ L.adj(u)
        se v ∈ Q && chave[v] > w(u, v)
            chave[v] ← w(u, v)
            π[v] ←u
        fimse
    fimpara
fimenquanto
Fim.
```



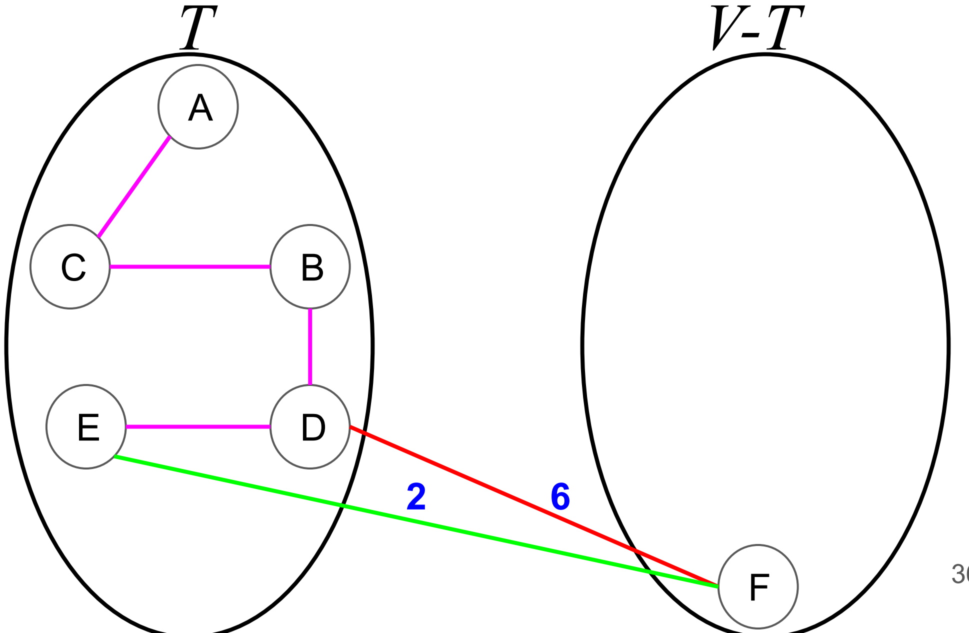
Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	B	5
E	D	2
F	D	6



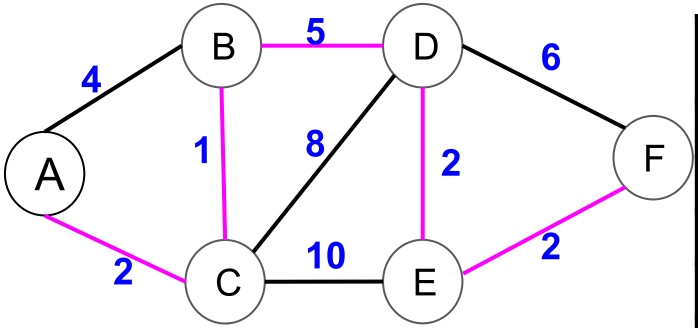
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



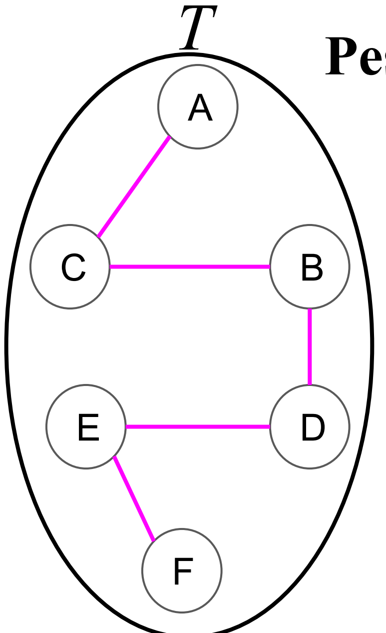
Vértice	π	*Chave*
A	C	2
B	C	1
C	N	0
D	B	5
E	D	2
F	E	2



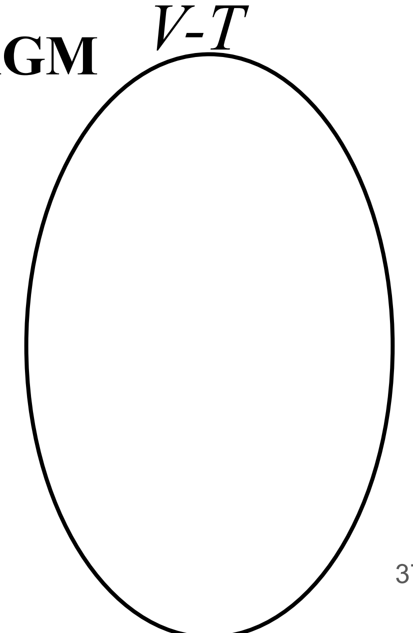
```
Prim(G, w, v0)
    para cada vértice u ← V[G]
        chave[u] ←∞
        π[u] ← NULL
    fimpara
    chave[v0] ← 0
    Q←FilaDePrioridade( )
    InsereNaFila(Q, V[G])
    enquanto Q ≠ vazio
        u ←RemoveDaFila(Q)
        para cada vértice v ∈ L.adj(u)
            se v ∈ Q && chave[v] > w(u, v)
                chave[v] ← w(u, v)
                π[v] ←u
            fimse
        fimpara
    fimenquanto
Fim.
```



Vértice	π	Chave
A	C	2
B	C	1
C	N	0
D	B	5
E	D	2
F	E	2

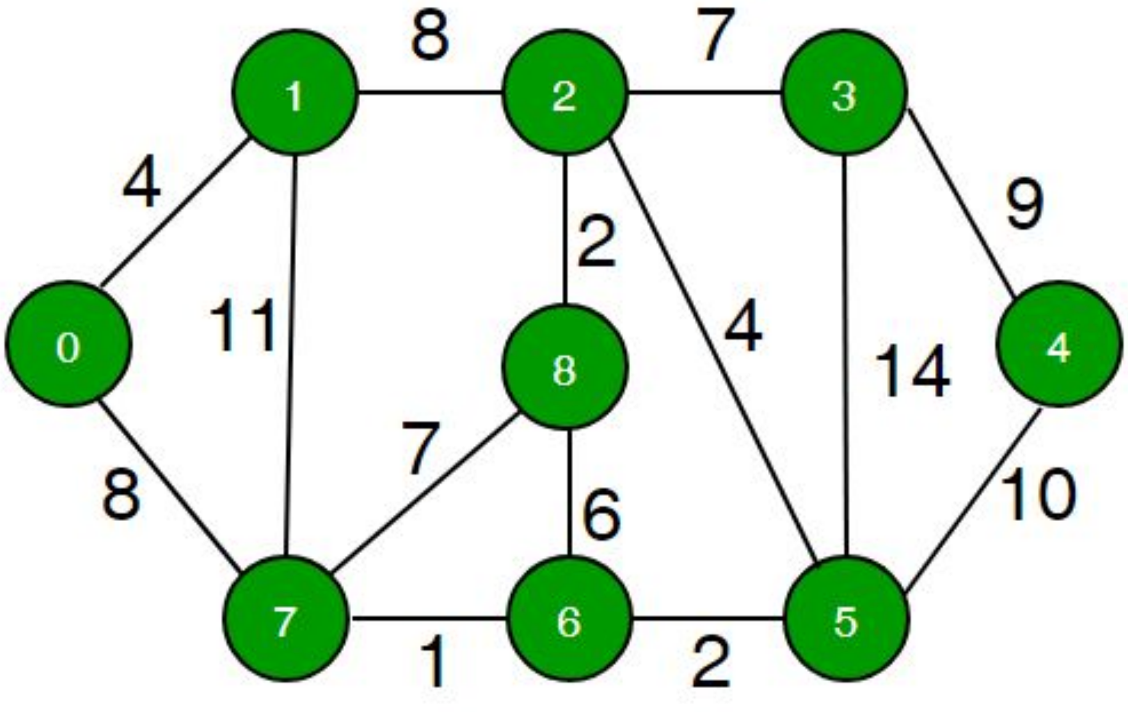


Peso da AGM
12



Exercício de Fixação

Encontre a árvore geradora mínima.



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE RUSSAS

Algoritmos em Grafos

Aula 14: Árvore Geradora Mínima

Professor Pablo Soares

2020.1