
Grafos: árvores geradoras mínimas

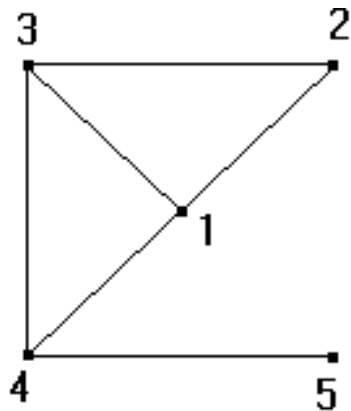
SCC0216 Modelagem Computacional em Grafos

Thiago A. S. Pardo

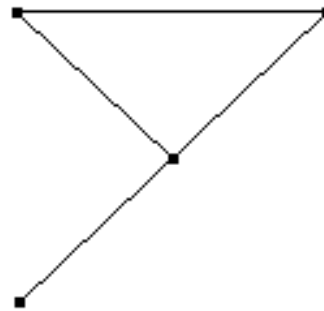
Maria Cristina F. Oliveira

Sub-grafo

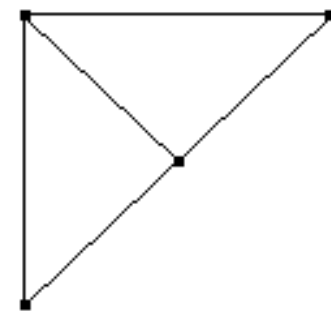
Dado um grafo $G_1(V_1, A_1)$, um grafo $G_2(V_2, A_2)$ é um sub-grafo de G_1 se V_2 está contido em V_1 e A_2 está contido em A_1



(a)



(b)

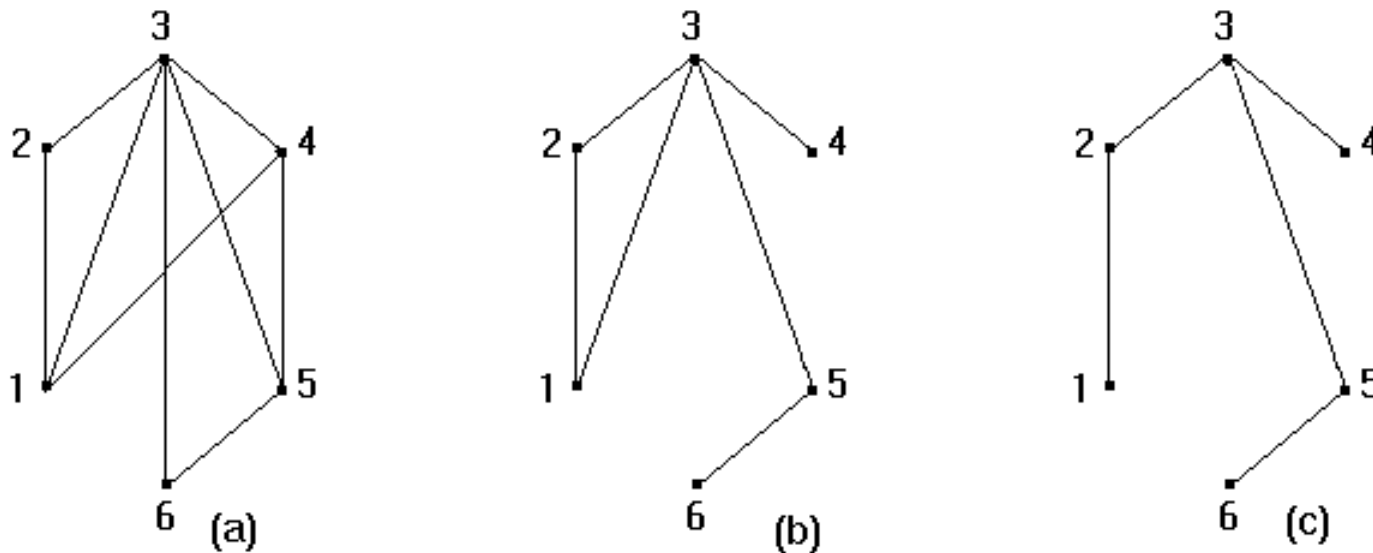


(c)

os grafos em (b) e (c) são sub-grafos do grafo em (a)

Sub-grafo gerador

Sub-grafo Gerador de um grafo $G_1(V_1, A_1)$ é um sub-grafo $G_2(V_2, A_2)$ de G_1 tal que $V_1 = V_2$. Quando o sub-grafo gerador é uma **árvore**, ele recebe o nome de árvore geradora



(b) e (c) são sub-grafos geradores do grafo em (a)

(c) é árvore geradora de (a) e (b)

Sub-grafo gerador de custo mínimo

■ Formalmente

- Dado um grafo não orientado $G(V,A)$
 - em que uma função $w: A \rightarrow \mathbb{R}^+$ define os custos das arestas
 - queremos encontrar um sub-grafo gerador mínimo conexo T de G tal que, para todo sub-grafo gerador conexo T' de G

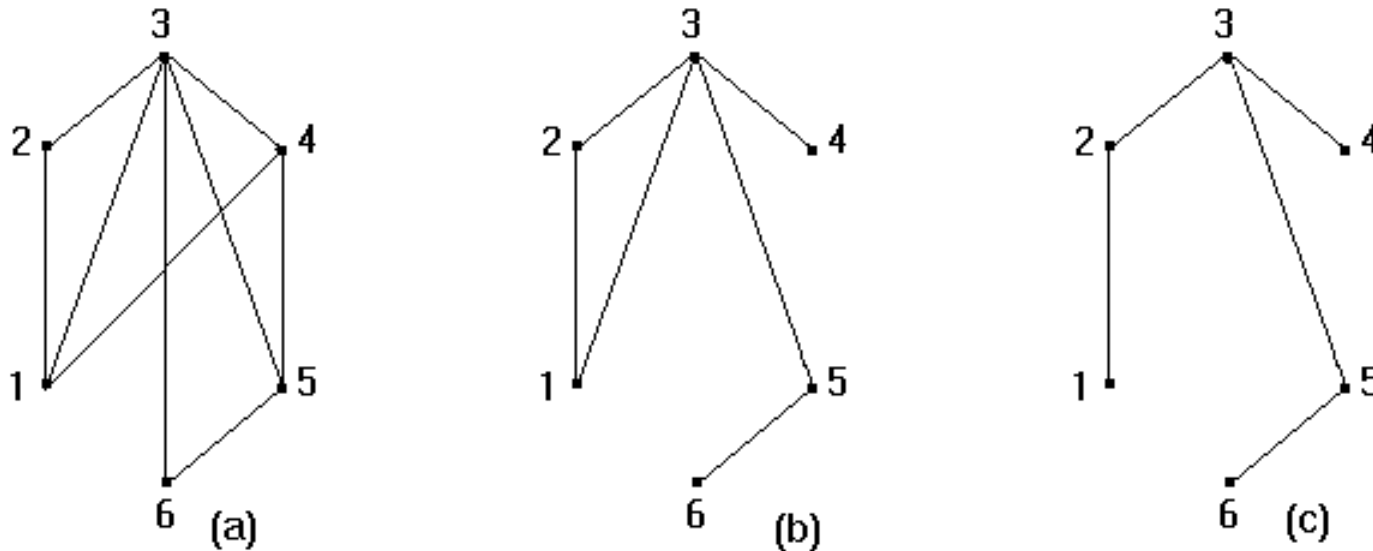
$$\sum_{a \in T} w(a) \leq \sum_{a \in T'} w(a)$$

Árvore geradora mínima

- Claramente, o problema só tem solução se G é conexo
 - A partir de agora, assumimos G conexo
- Também não é difícil ver que a solução para esse problema será sempre uma árvore
 - Por que?

Sub-grafo gerador

Sub-grafo Gerador de um grafo $G_1(V_1, A_1)$ é um sub-grafo $G_2(V_2, A_2)$ de G_1 tal que $V_1 = V_2$. Quando o sub-grafo gerador é uma **árvore**, ele recebe o nome de árvore geradora



(b) e (c) são sub-grafos geradores do grafo em (a)

(c) é árvore geradora de (a) e (b)

Árvore geradora mínima

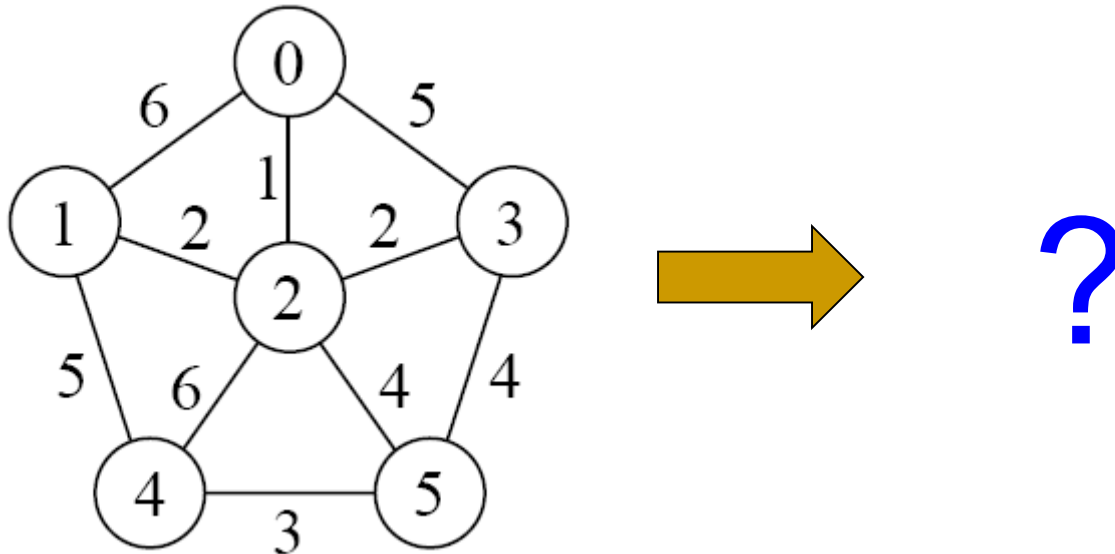
- Claramente, o problema só tem solução se **G é conexo**
 - A partir de agora, assumimos G conexo
- Também não é difícil ver que a solução para esse problema será **sempre uma árvore**
 - Basta notar que T não terá ciclos pois, caso contrário, poderíamos obter um outro sub-grafo T', ainda conexo e com custo menor do que o custo de T, removendo o ciclo (i.e., uma das arestas que o compõem)

Árvore geradora mínima

- **Árvore Geradora** de um grafo G é um sub-grafo de G que contém **todos os seus vértices** e, ainda, é uma árvore
- **Árvore Geradora Mínima** é a árvore geradora de um grafo valorado cuja soma dos **pesos associados às arestas é mínimo**, i.e., é uma árvore geradora de custo mínimo

Árvore geradora mínima

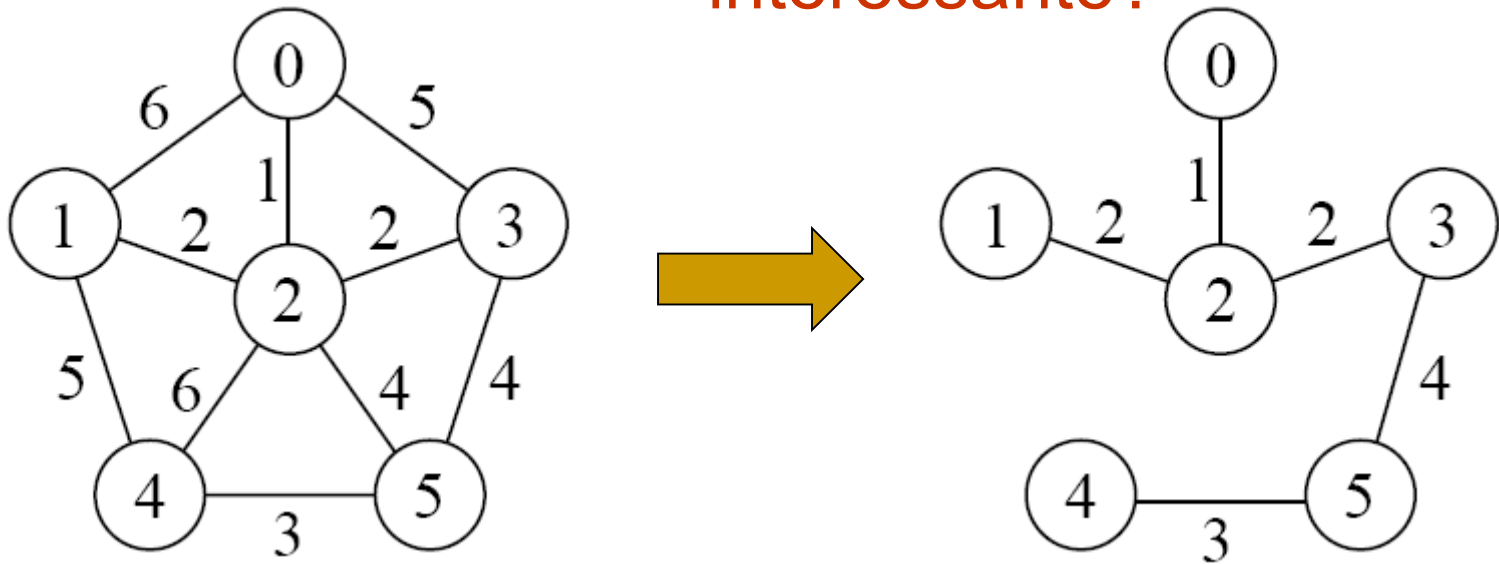
■ Exemplo



Árvore geradora mínima

■ Exemplo

Porque esse problema é interessante?



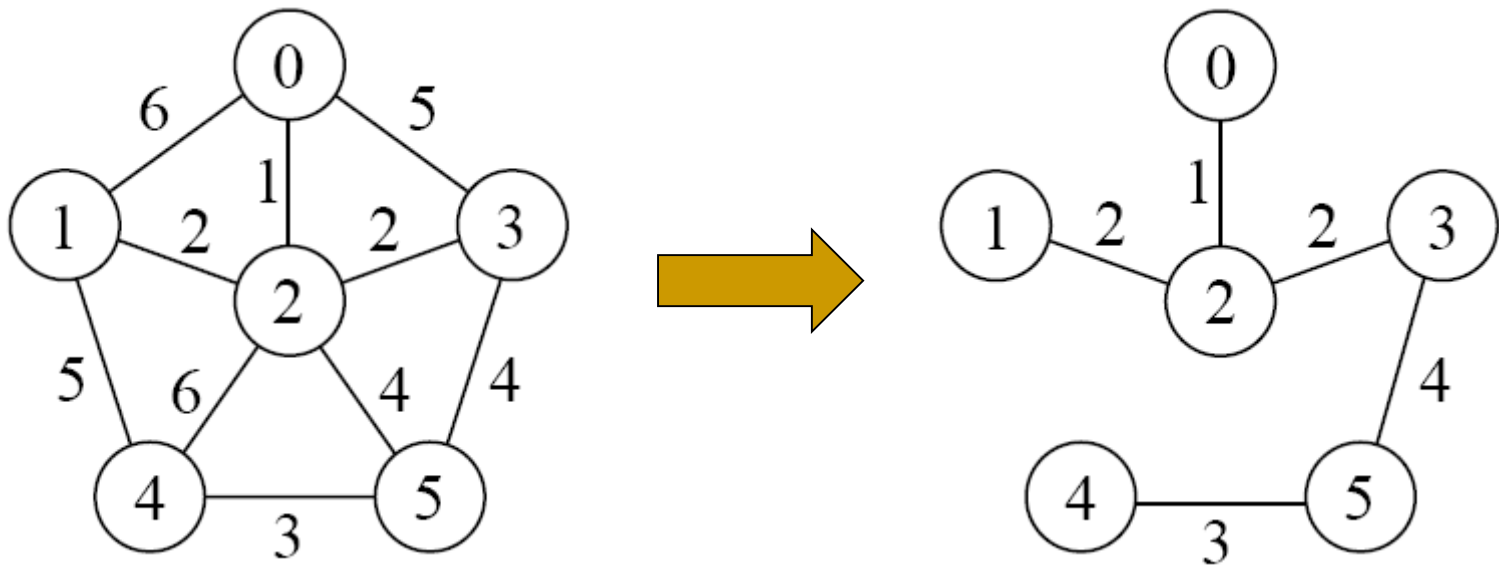
Por que é um problema interessante?

- Suponha que queremos montar uma **rede de distribuição de energia elétrica** para interligar N localidades em uma cidade
 - Cada conexão direta entre localidades i e j tem um **custo associado** (preço do cabeamento, mão de obra, tempo, etc.)
 - Nem todas as localidades precisam ser ligadas diretamente, desde que todas tenham **acesso à energia elétrica**
- Como determinar as linhas distribuição de forma **a minimizar o custo total** de distribuição de energia?

Exercício

Para pensar

- Proponha um algoritmo para encontrar a árvore geradora mínima de um grafo



Árvore geradora mínima

- Como encontrar a árvore geradora mínima de um grafo G ?
 - Algoritmo genérico: ilustra a estratégia 'gulosa'
 - Algoritmo de Prim
 - Algoritmo de Kruskal

Árvore geradora mínima

Algoritmo Genérico

```
procedimento genérico(G)
A = ∅
enquanto A não define uma árvore geradora
    encontre uma aresta (u,v) segura para A
    A = A ∪ {(u,v)}
retorna A
```

G conexo, não direcionado, ponderado

A - conjunto de arestas

Árvore geradora mínima

Algoritmo Genérico

```
procedimento genérico(G)
A = ∅
enquanto A não define uma árvore geradora
    encontre uma aresta (u,v) segura para A
    A = A ∪ {(u,v)}
retorna A
```

Abordagem 'gulosa' -> adiciona uma aresta (u,v) '*segura*' a cada iteração, de modo que a árvore continue sendo uma árvore

Árvore geradora mínima

Algoritmo Genérico

```
procedimento genérico (G(V,E))  
A =  $\emptyset$   
enquanto A não define uma árvore geradora  
    encontre uma aresta (u,v) segura para A  
    A = A  $\cup$  { (u,v) }  
retorna A
```

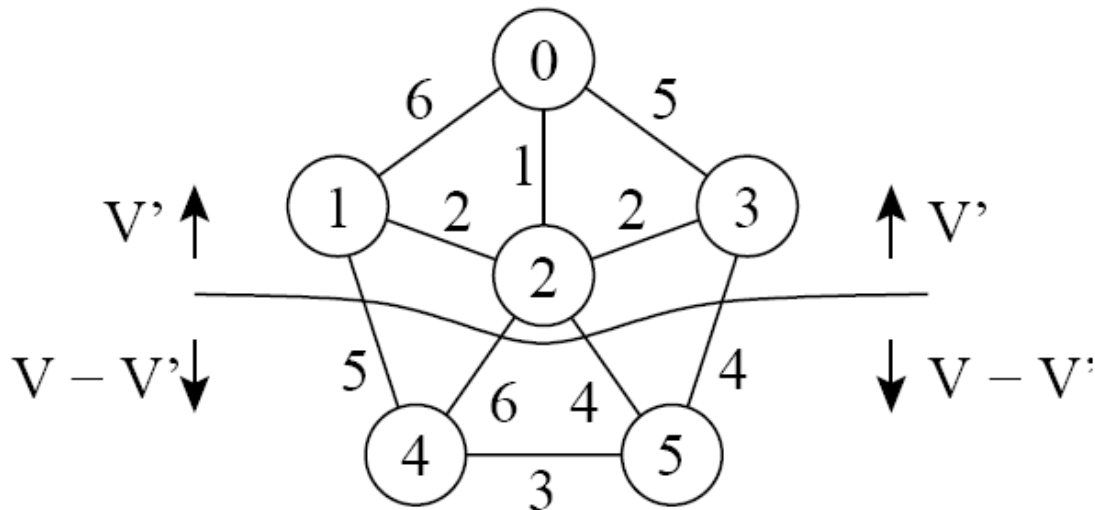
Condição invariante do laço: a cada iteração, a árvore obtida até o momento é um subconjunto de uma árvore geradora mínima

aresta *segura* (u,v): respeita o invariante, i.e., não introduz um ciclo, e não poderia ser substituída por outra aresta equivalente de custo menor

Árvore geradora mínima

- Alguns conceitos

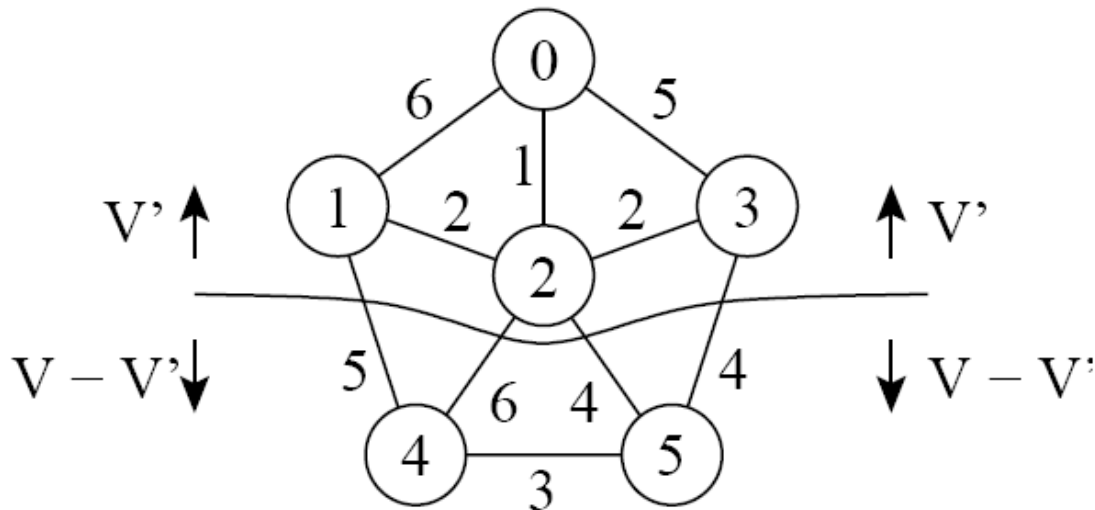
- Um **corte** $(V'; V - V')$ de um grafo não direcionado $G=(V,E)$ é uma partição de V
- Uma aresta **segura** (u,v) **crusa o corte** se um dos vértices extremos pertence a V' e o outro pertence a $V - V'$



Árvore geradora mínima

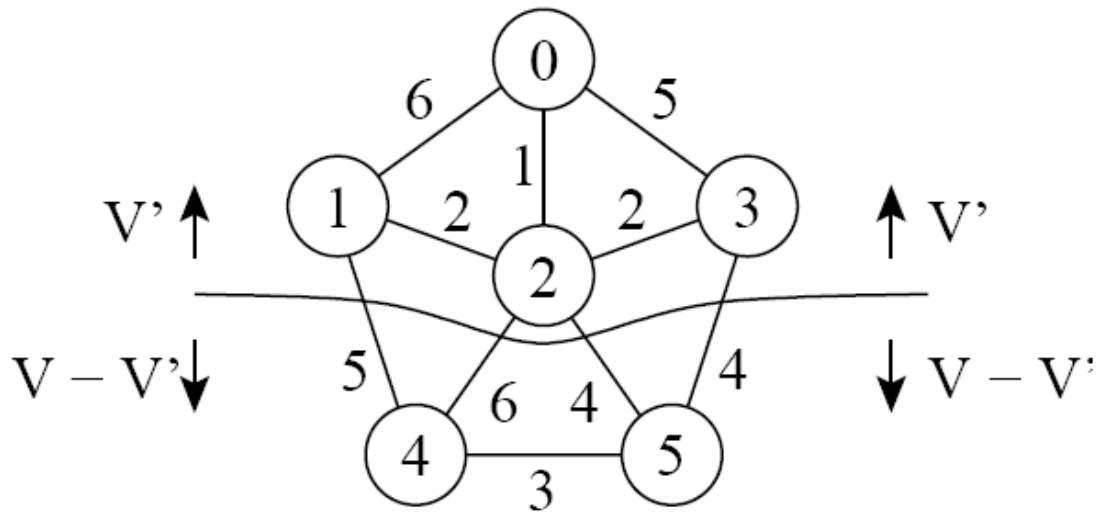
- Alguns conceitos

- Um corte *respeita* um conjunto A de arestas se nenhuma aresta em A cruza o corte
- Uma aresta que cruza o corte e tem custo mínimo em relação a todas as arestas cruzando o corte é denominada uma *aresta leve*



Árvore geradora mínima

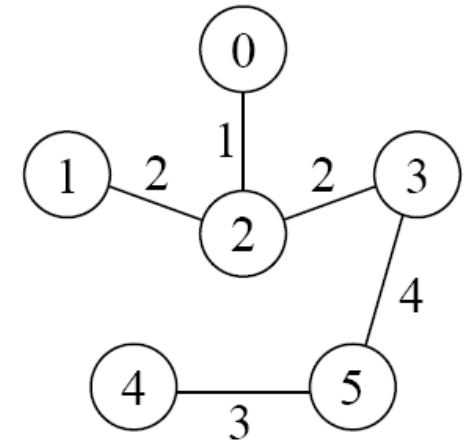
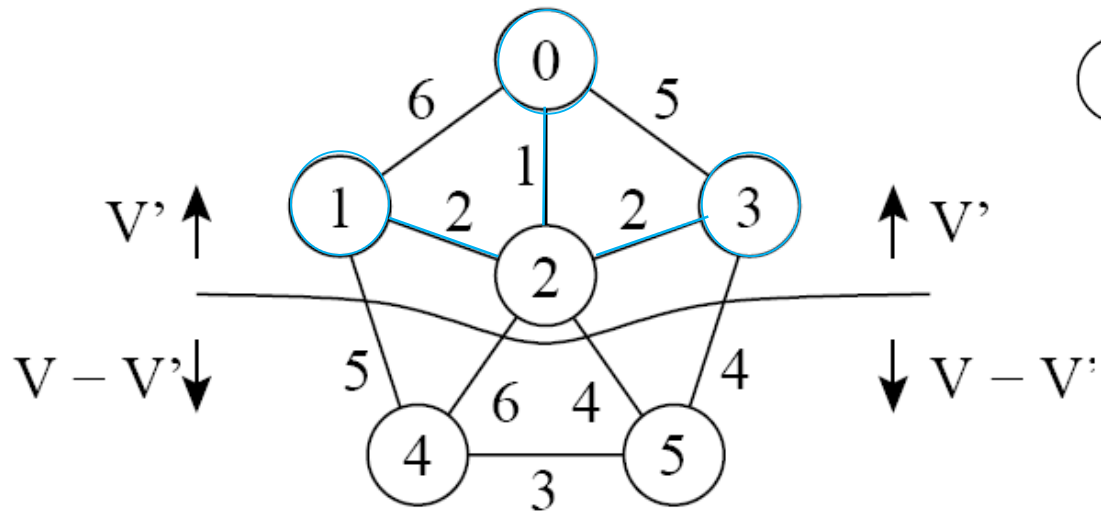
- Exemplo



- Se T é uma árvore geradora mínima de um sub-grafo e há um corte $(V'; V - V')$ que respeita T , a aresta leve (u, v) é uma aresta segura para T

Árvore geradora mínima

- Exemplo



- Se T é uma **árvore geradora mínima** de um sub-grafo e há um **corte** $(V'; V - V')$ que respeita T , a **aresta leve** (u, v) é uma **aresta segura** para T

Algoritmo de Prim

procedimento Prim(G)

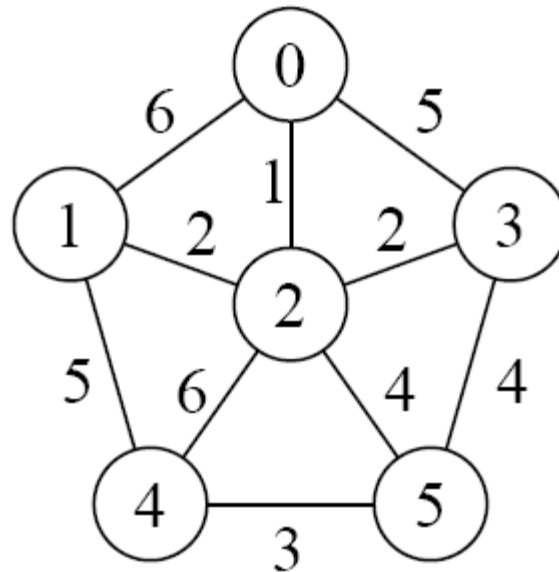
escolha um vértice s para iniciar a árvore
enquanto há vértices fora da árvore
 selecione uma aresta segura
 insira a aresta e seu vértice na árvore

Ponto importante do algoritmo: **seleção de uma aresta segura**

Outro ponto: os vértices que já estão na árvore definem um corte de V

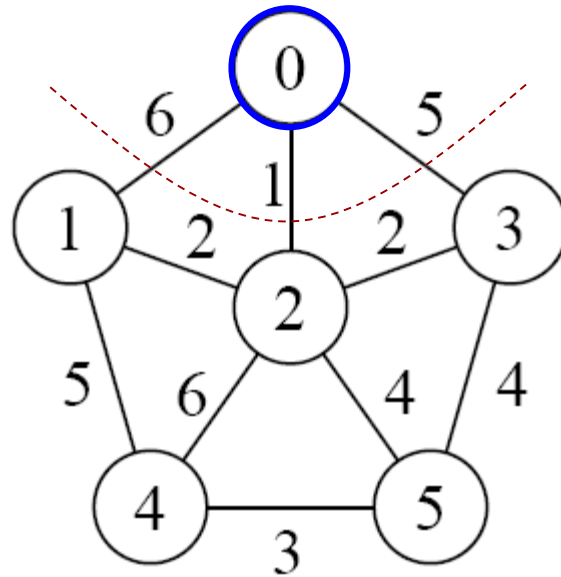
Algoritmo de Prim

- Exemplo: iniciando o algoritmo pelo vértice 0



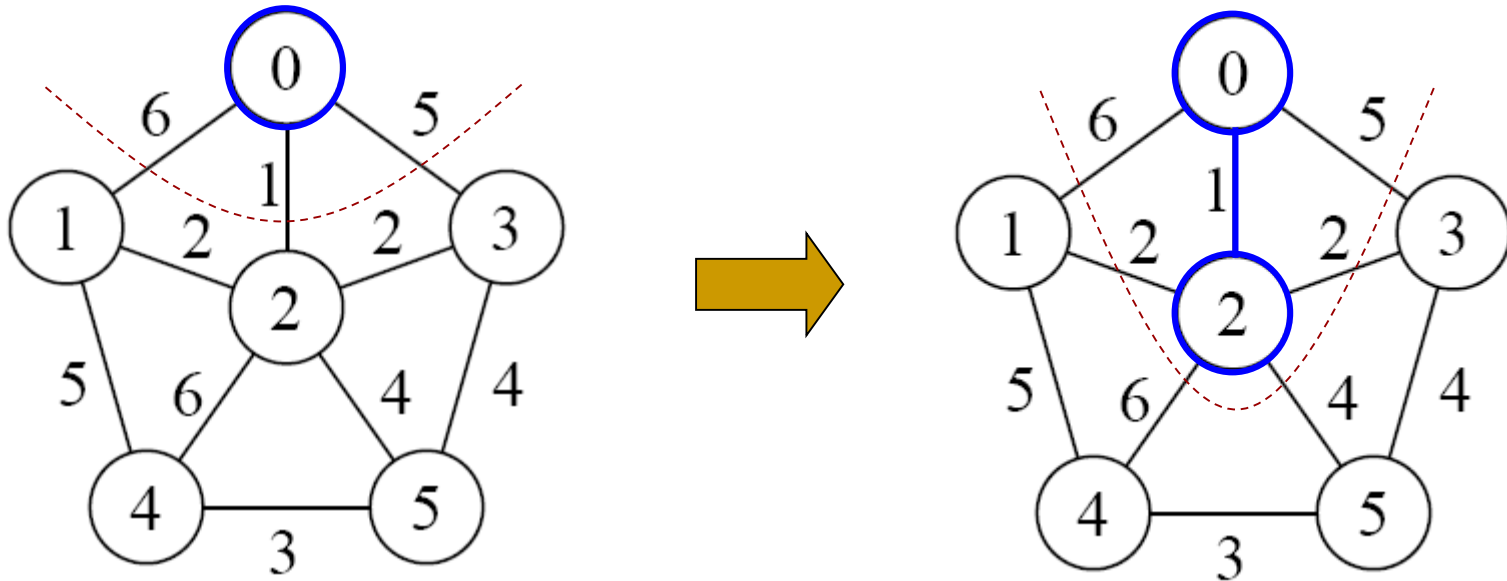
Algoritmo de Prim

- Exemplo: iniciando o algoritmo pelo vértice 0

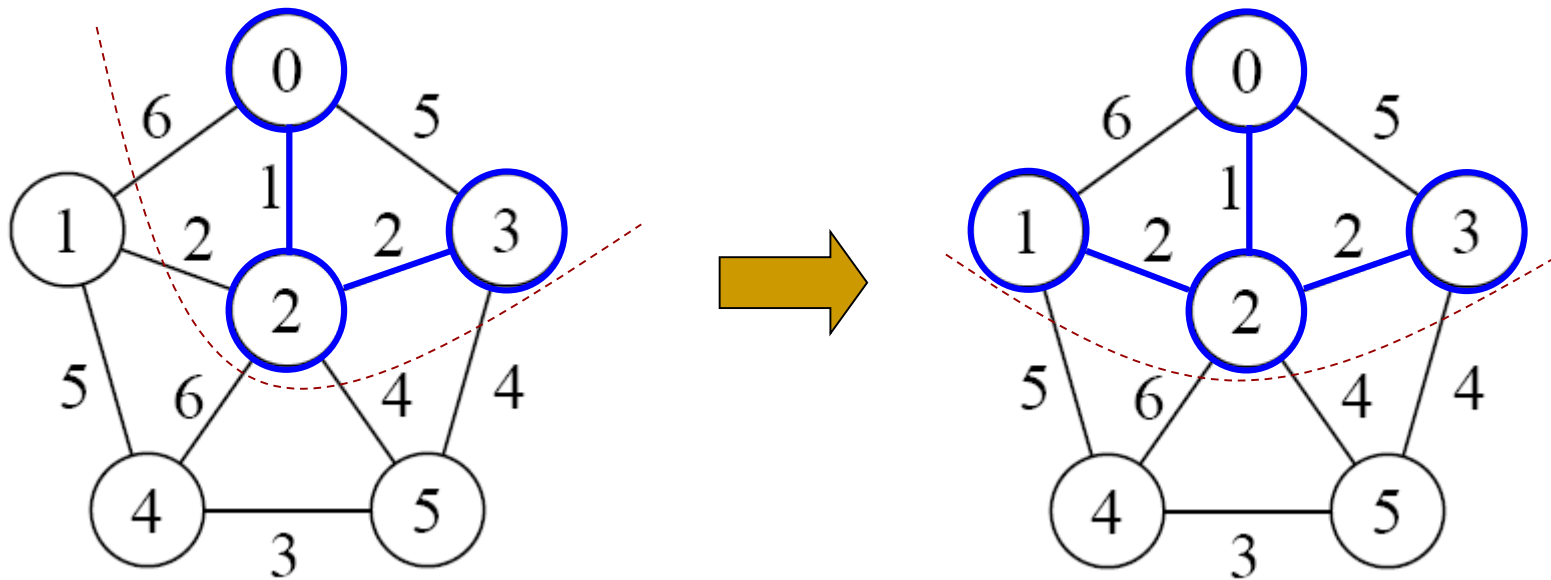


Algoritmo de Prim

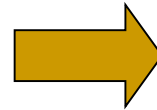
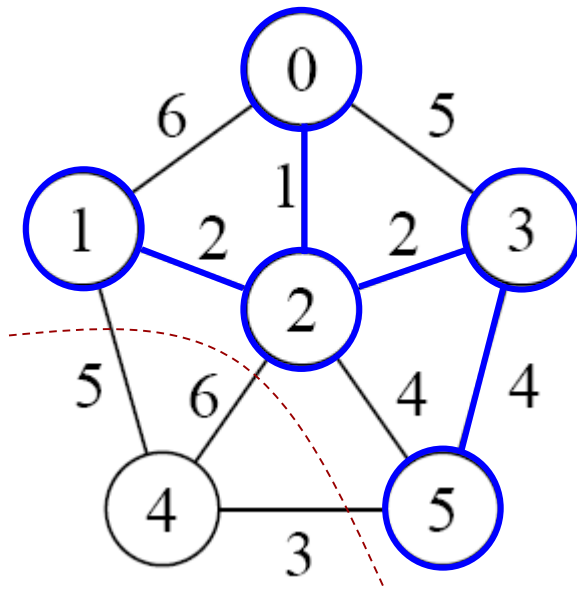
- Exemplo: iniciando o algoritmo pelo vértice 0



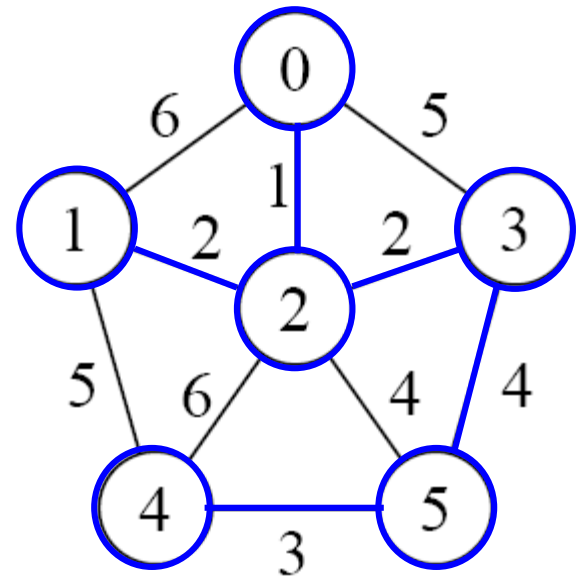
Algoritmo de Prim



Algoritmo de Prim

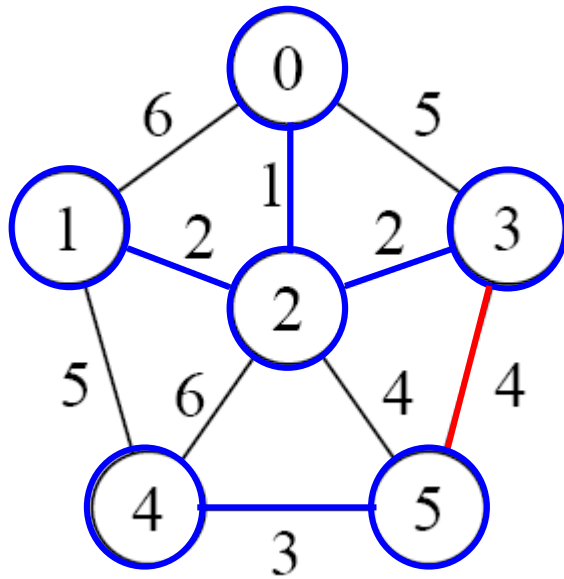


Árvore geradora mínima

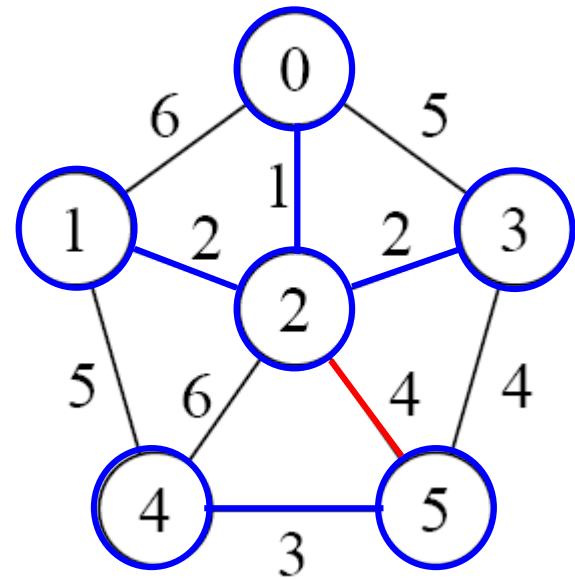


Algoritmo de Prim

- Há mais de uma árvore geradora mínima para um mesmo grafo



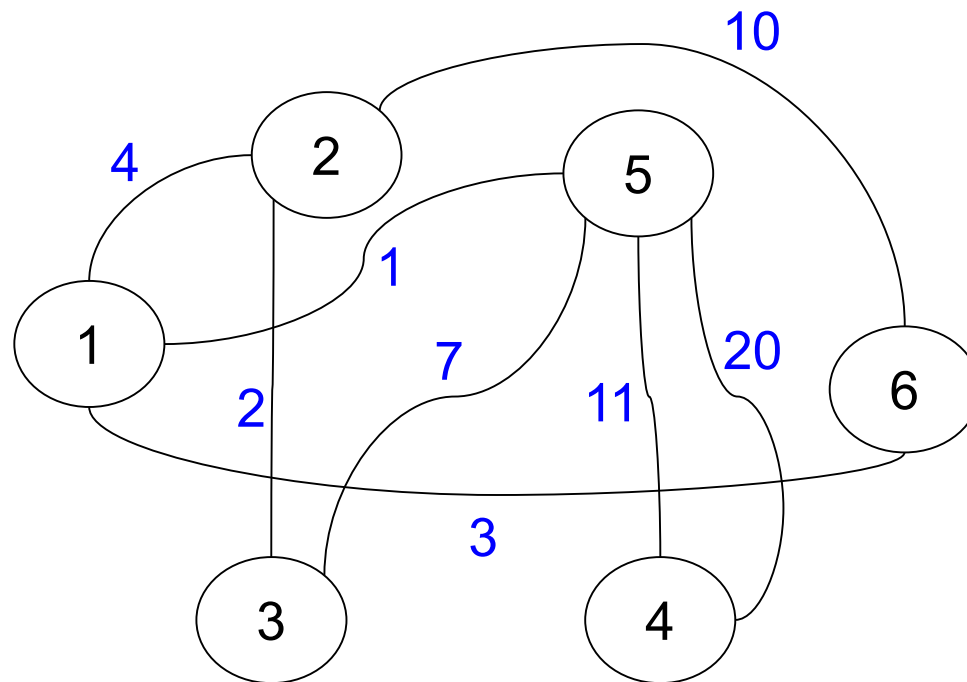
ou



Exercícios

Algoritmo de Prim

- Encontre uma árvore geradora mínima para o grafo abaixo utilizando o algoritmo de Prim



Algoritmo de Prim

- **Implemente** o algoritmo de Prim!
 - Qual a complexidade de sua implementação?
 - Como melhorar?

-
- A grande questão é ter uma maneira eficiente de, a cada passo, achar uma aresta segura!
 - Os dois algoritmos (Prim e Kruskal) variam na estratégia que adotam para fazer isso!
 - Ambos podem ser vistos como especializações do algoritmo genérico (estratégia ‘greedy’)

Algoritmo de Prim

- Maneira mais eficiente de determinar a aresta segura
 - Mantém todos os vértices que ainda não estão na árvore em uma **fila de prioridade**
 - Um valor de chave é usado para determinar a prioridade do vértice na fila (‘key’): o vértice v prioritário é o vértice extremo de alguma aresta segura
 - Para cada vértice v , a aresta segura é a aresta de menor peso que conecta v a um vértice já na árvore (aresta leve que cruza o corte!)

Algoritmo de Prim

- Maneira mais eficiente de determinar a aresta segura
 - Mantém todos os vértices que ainda não estão na árvore em uma **fila de prioridade**
 - Qual a maneira mais eficiente de representar uma fila de prioridade?
 - Qual o custo das operações de inserção e remoção nessa representação?

Procedure MST-PRIM(G, r)

1. Cria array $parent[]$ de tamanho $|V|$, inicializa com NIL
2. Cria minHeap H (ou fila de prioridade) de tamanho $|V|$
3. Insere todos os vértices em H , tal que $H.key[v]$ é 0 para o vértice inicial e $H.key[v]$ é INF (infinito) para todos os demais
4. **While** not emptyHeap(H) **do**
 - $u := extractMin(H)$
 - For all** v in $adj(u)$ **do**
 - If** v in H **then** // seja w o peso da aresta (u, v)
 - If** $w < H.key[v]$ **then**
 - $H.key[v] := w$ // (u, v) é aresta segura (até o momento)
 - End If**
 - $parent[v] := u$
 - End If**
 - End For**
- End While**

Procedure MST-PRIM(G, r)

1. Cria array $parent[]$ de tamanho $|V|$, inicializa com NIL
2. Cria minHeap H (ou fila de prioridade) de tamanho $|V|$
3. Insere todos os vértices em H , tal que $H.key[r] := 0$ (vértice inicial) e $H.key[v] := INF$ (infinito) para todos os demais
4. **While** not emptyHeap(H) **do**
 - $u := extractMin(H)$ // operação tem custo $O(\log(|V|))$
 - For all** v in $adj(u)$ **do**
 - If** v in H **then** // atualiza $key[v]$ em H : custo $O(\log(|V|))$
 - If** $w < H.key[v]$ **then** // w é o peso da aresta (u,v)
 - $H.key[v] := w$
 - End If**
 - $parent[v] := u$
 - End If**
 - End For**
- End While**

Procedure MST-PRIM(G, r)

1. Cria array $parent[]$ de tamanho $|V|$, inicializa com NIL
2. Cria minHeap H (ou fila de prioridade) de tamanho $|V|$
3. Insere todos os vértices em H , tal que $H.key[v]$ é 0 para o vértice inicial e $H.key[v]$ é INF (infinito) para todos os demais
4. **While** not emptyHeap(H) **do** // executado $|V|$ vezes
 $u := extractMin(H)$ // operação tem custo $O(\log(|V|))$
 For all v in $adj(u)$ **do** // executado $|A|$ vezes **no total**
 If v in H **then** // atualiza $key[v]$ em H : custo $O(\log(|V|))$
 If $w < H.key[v]$ **then** // w é o peso da aresta (u, v)
 $H.key[v] := w$
 End If
 $parent[v] := u$
 End If
 End For
End While

Algoritmo de Prim

- Complexidade de tempo:

$$O((|A| + |V|)^* \log(|V|)) \quad \text{ou}$$

$$O(|A| \log(|V|))$$

- Supondo representação por listas de adjacências e a fila de prioridade implementada como uma *heap* binária

Algoritmo de Prim

- Complexidade de tempo:

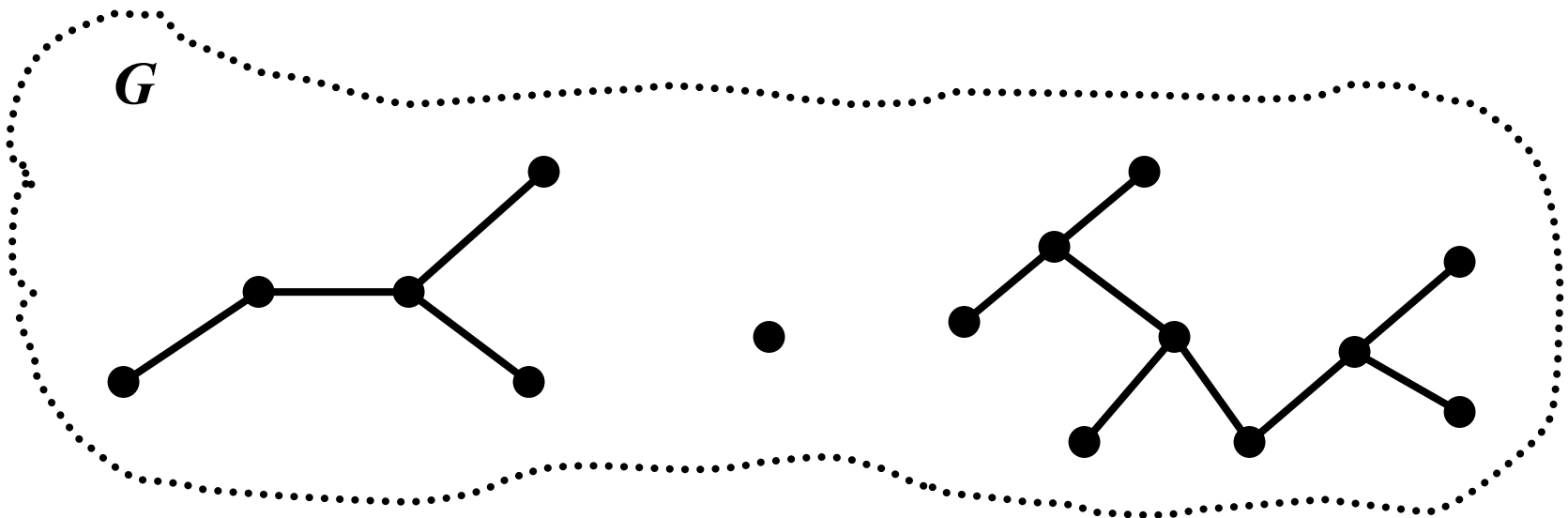
$$O((|A| + |V|)^* \log(|V|)) \text{ ou}$$

- Supondo representação por listas de adjacências e a fila de prioridade implementada como uma *heap* binária

-
- Os dois algoritmos (Prim e Kruskal) variam na estratégia adotada para identificar uma aresta segura na iteração corrente...

Algoritmo de Kruskal

- Relembrando: uma floresta é um conjunto de árvores



Algoritmo de Kruskal

- Mais eficiente do que Prim em grafos esparsos
- Não precisa manter uma árvore desde o começo
- Não inicia em nenhum vértice em particular
 - Verifica se as arestas podem (ou não) pertencer à árvore, analisando-as em ordem crescente de custo
 - As árvores que compõem a floresta são identificadas pelos conjuntos S_i , que contém os vértices que a compõem

Algoritmo de Kruskal

- Basicamente, o algoritmo **constrói uma árvore geradora** a partir de uma floresta
 - Estado inicial: corresponde à floresta formada por $|V|$ árvores triviais (i.e., um só vértice cada)

Algoritmo de Kruskal

procedimento Kruskal (G)

```
definir conjuntos  $S_j = \{v_j\}$ ,  $1 \leq j \leq |V|$   
inserir as arestas de G em uma fila de prioridade,  
em ordem crescente de peso  
enquanto houver arestas na fila faça  
  (v,w) = remove aresta da fila  
  se  $v \in S_p$  e  $w \in S_q$ ,  $S_p \cap S_q = \emptyset$  então  
     $S_p = S_p \cup S_q$   
    eliminar  $S_q$ 
```

Ponto importante do algoritmo: **deve-se evitar ciclos!**

Algoritmo de Kruskal

procedimento Kruskal (G)

```
definir conjuntos  $S_j = \{v_j\}$ ,  $1 \leq j \leq |V|$   
inserir as arestas de G em uma fila de prioridade,  
em ordem crescente de peso  
enquanto houver arestas na fila faça  
  (v,w) = remove aresta da fila  
  se  $v \in S_p$  e  $w \in S_q$ ,  $S_p \cap S_q = \emptyset$  então  
     $S_p = S_p \cup S_q$   
    eliminar  $S_q$ 
```

Ponto importante do algoritmo: deve-se evitar ciclos! Por que?

Algoritmo de Kruskal

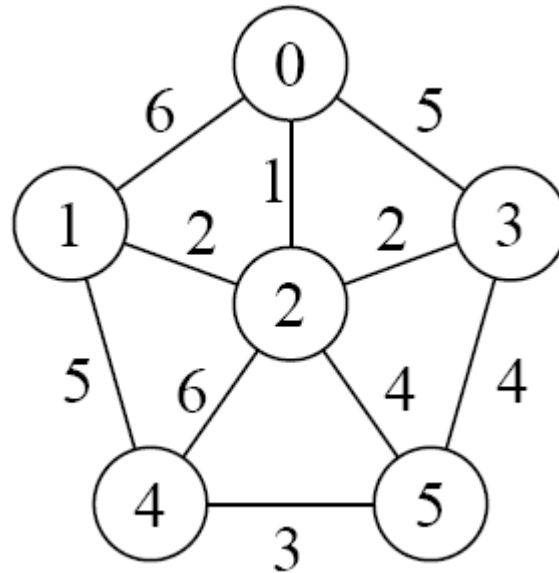
procedimento Kruskal (G)

```
definir conjuntos  $S_j = \{v_j\}$ ,  $1 \leq j \leq |V|$   
inserir as arestas de G em uma fila de prioridade,  
em ordem crescente de peso  
enquanto houver arestas na fila faça  
  (v,w) = remove aresta da fila  
  se  $v \in S_p$  e  $w \in S_q$ ,  $S_p \cap S_q = \emptyset$  então  
     $S_p = S_p \cup S_q$   
    eliminar  $S_q$ 
```

Ponto importante do algoritmo: deve-se evitar ciclos! Onde está isso?

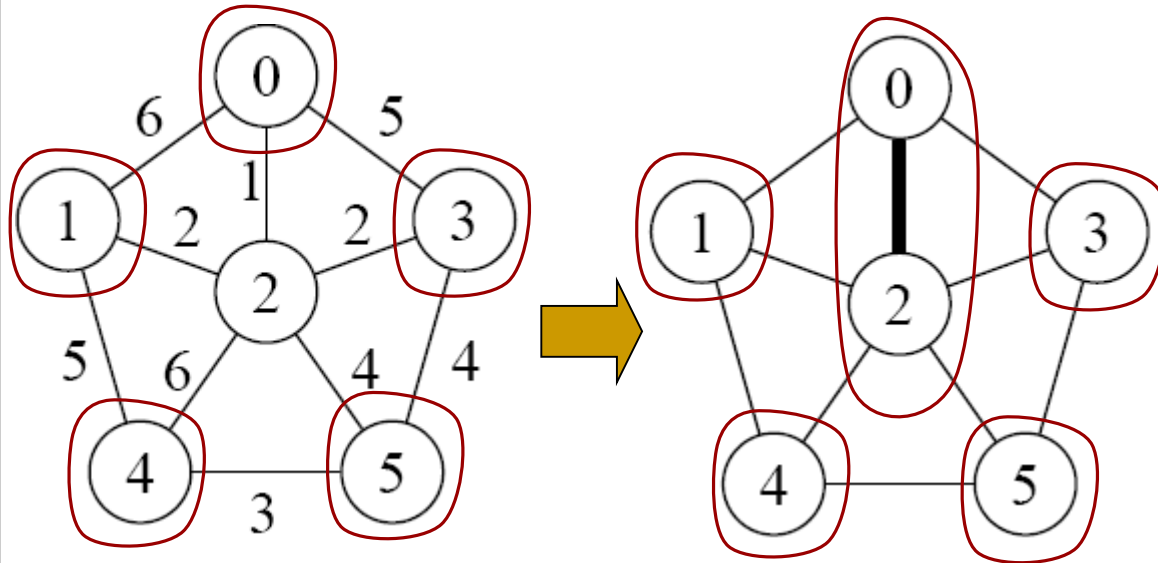
Algoritmo de Kruskal

■ Exemplo



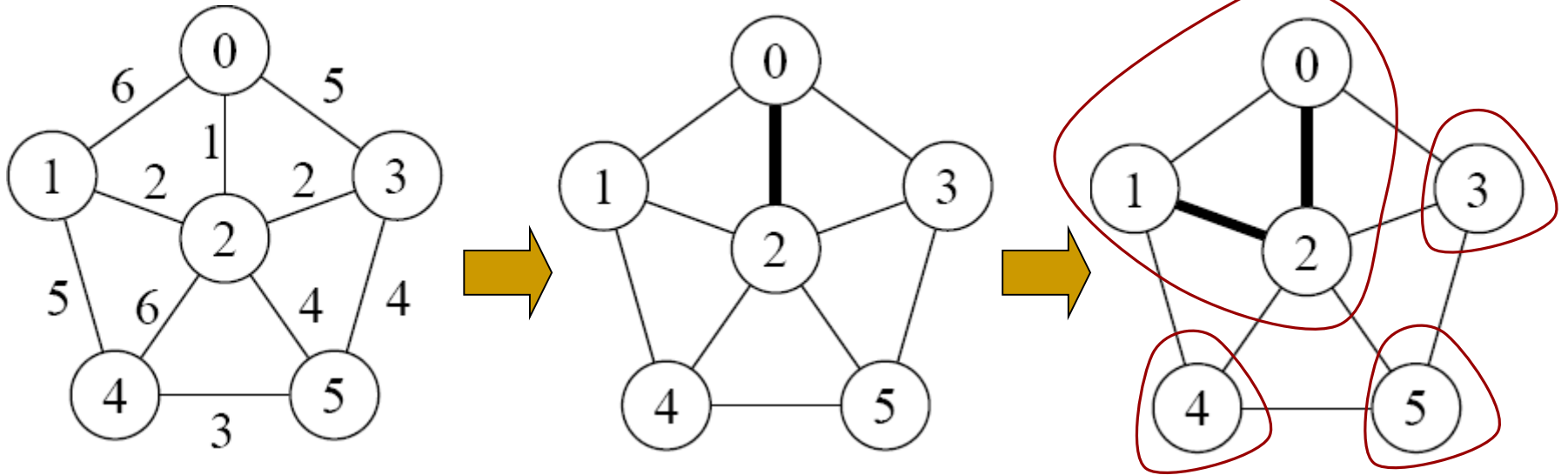
Fila: (0,2,1) (1,2,2) (2,3,2) (4,5,3) (3,5,4) (2,5,4) (1,4,5) (0,3,5) (0,1,6) (2,4,6)

Algoritmo de Kruskal: exemplo



Fila: ~~(0,2,1)~~ (1,2,2) (2,3,2) (4,5,3) (3,5,4) (2,5,4) (1,4,5) (0,3,5) (0,1,6) (2,4,6)

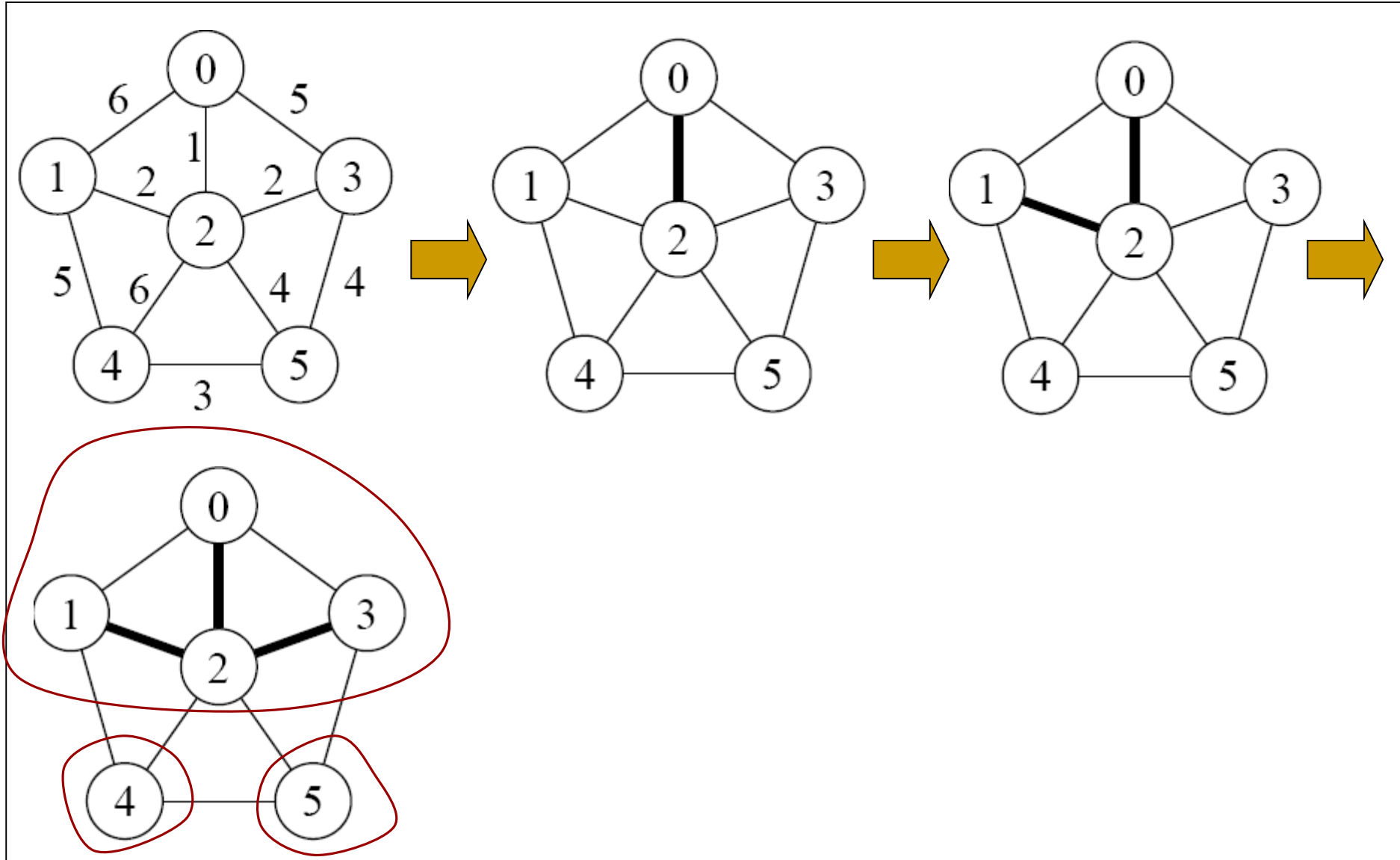
Algoritmo de Kruskal: exemplo



Fila: ~~(0,2,1)~~ ~~(1,2,2)~~ (2,3,2) (4,5,3) (3,5,4) (2,5,4) (1,4,5) (0,3,5) (0,1,6) (2,4,6)

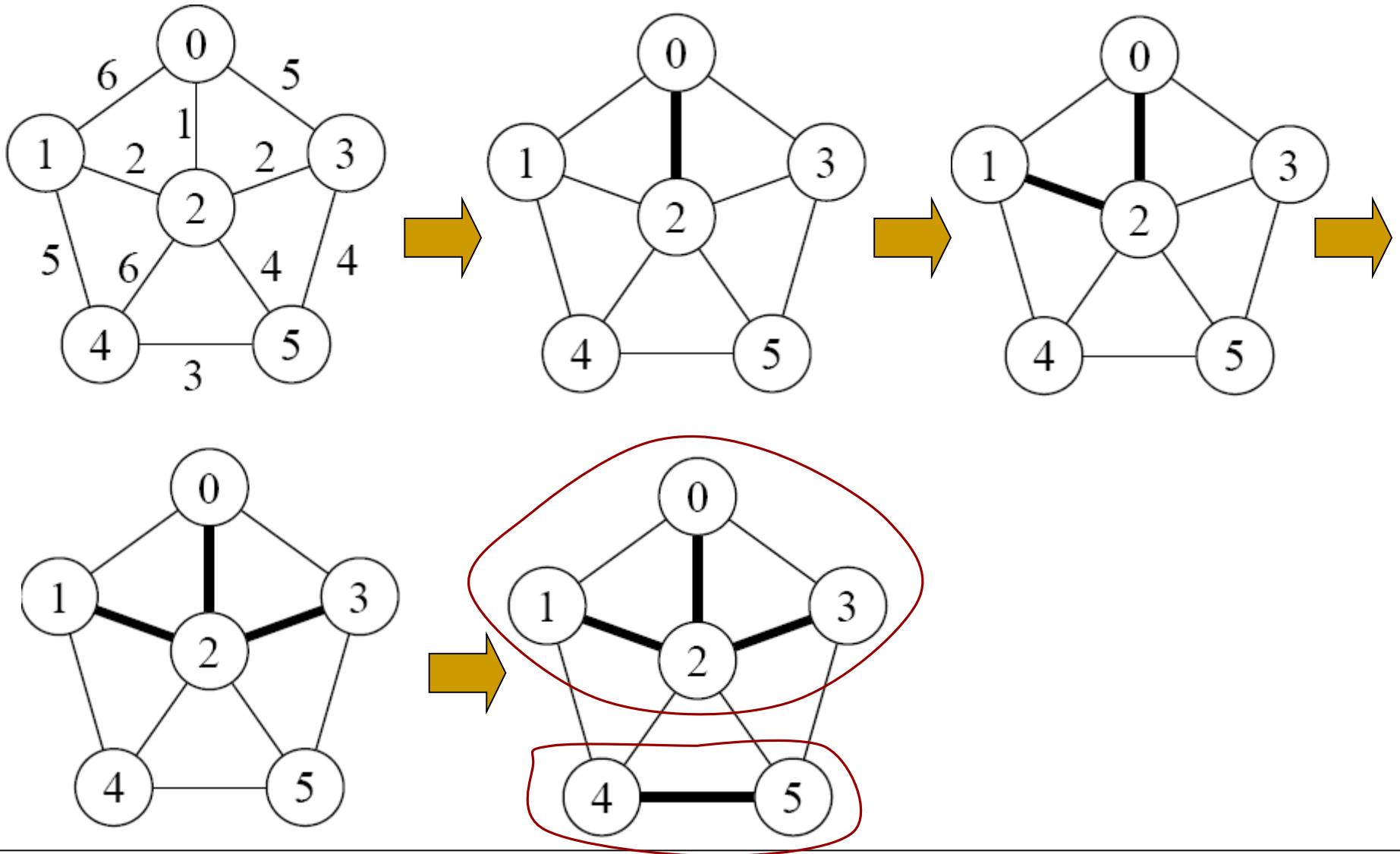
Algoritmo de Kruskal: exemplo

Fila: ~~(0,2,1)~~ ~~(1,2,2)~~ ~~(2,3,2)~~ (4,5,3) (3,5,4) (2,5,4) (1,4,5) (0,3,5) (0,1,6) (2,4,6)



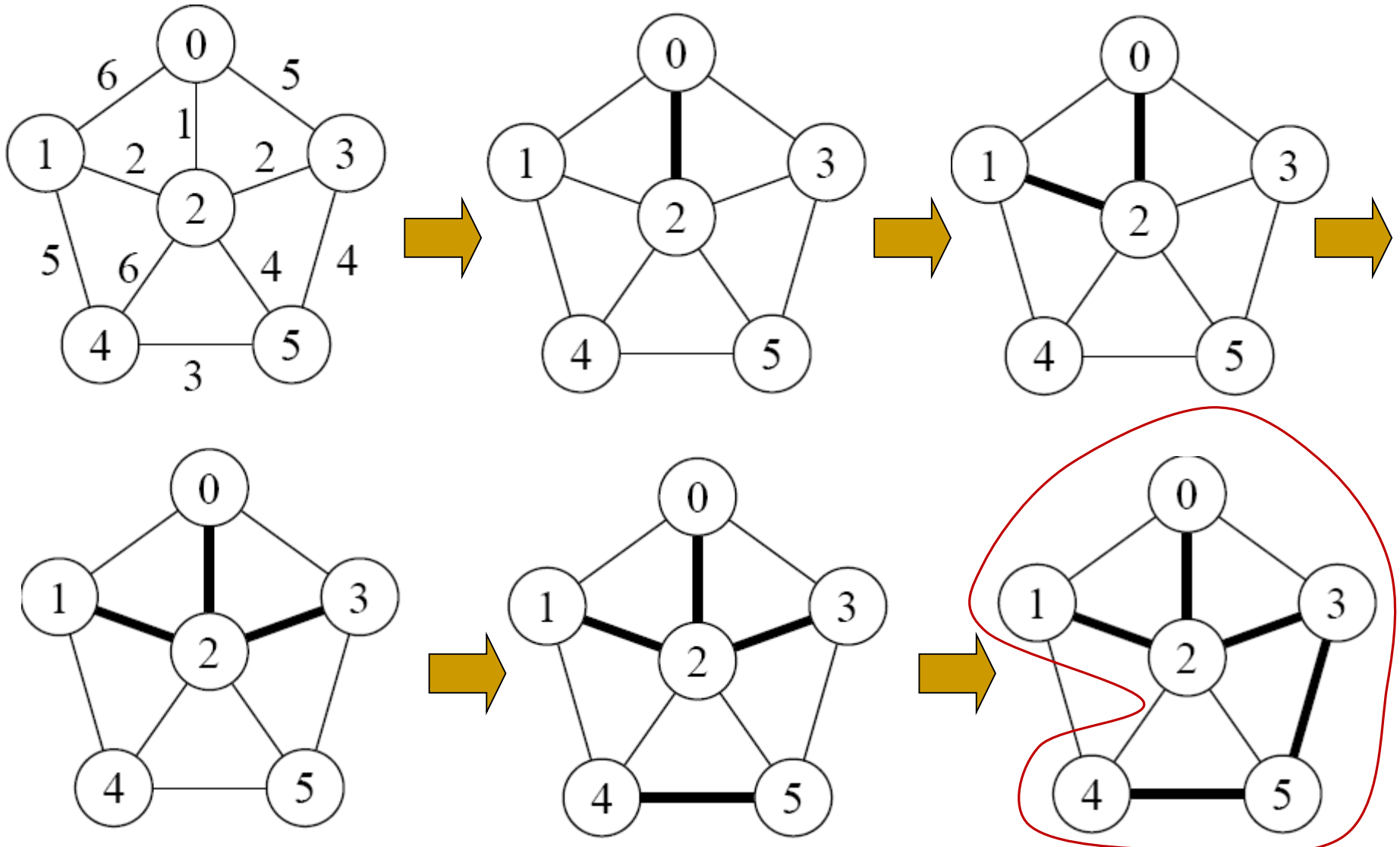
Algoritmo de Kruskal: exemplo

Fila: ~~(0,2,1)~~ ~~(1,2,2)~~ ~~(2,3,2)~~ ~~(4,5,3)~~ ~~(3,5,4)~~ (2,5,4) (1,4,5) (0,3,5) (0,1,6) (2,4,6)



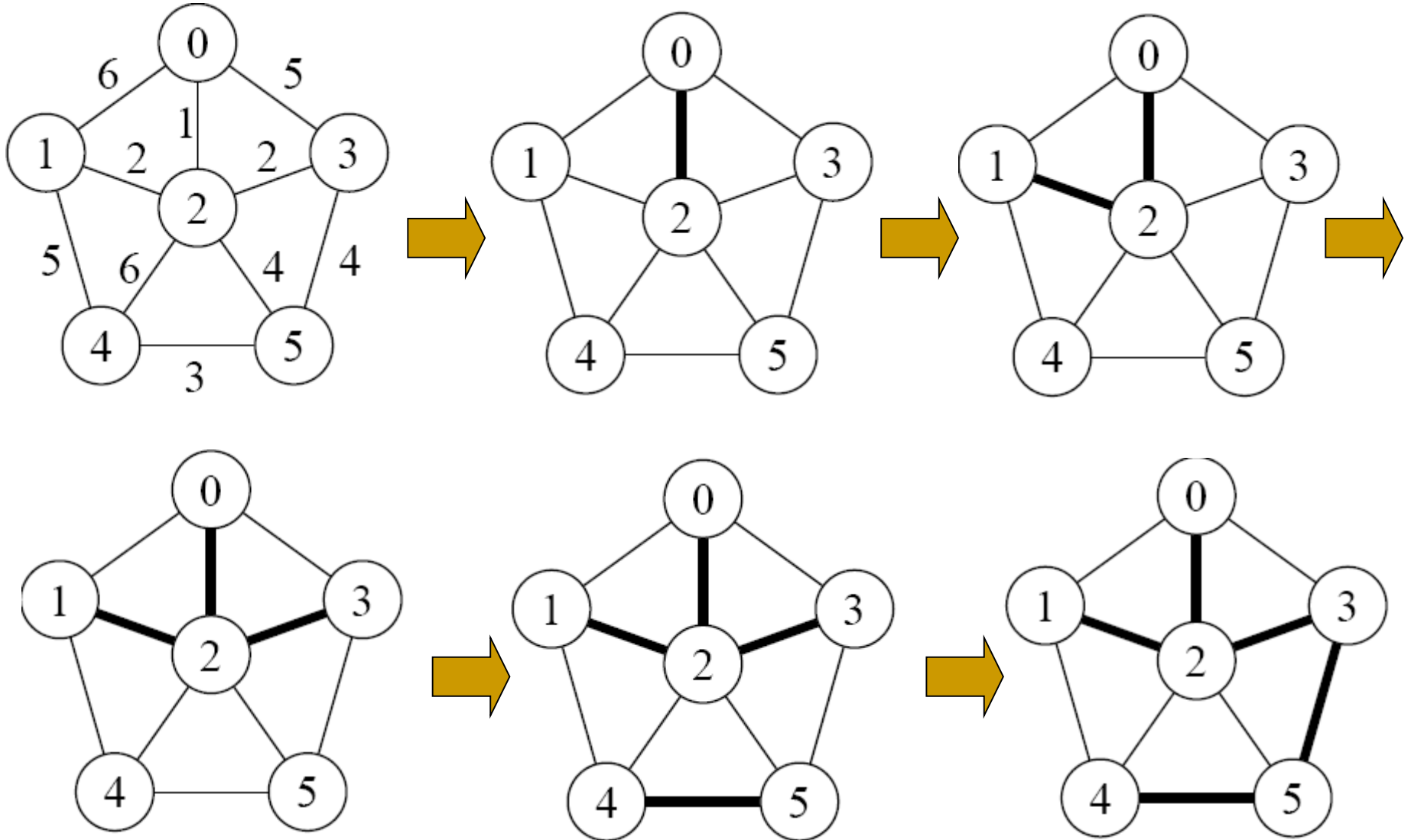
Algoritmo de Kruskal: exemplo

Fila: ~~(0,2,1)~~ ~~(1,2,2)~~ ~~(2,3,2)~~ ~~(4,5,3)~~ ~~(3,5,4)~~ ~~(2,5,4)~~ (1,4,5) (0,3,5) (0,1,6) (2,4,6)



Algoritmo de Kruskal: exemplo

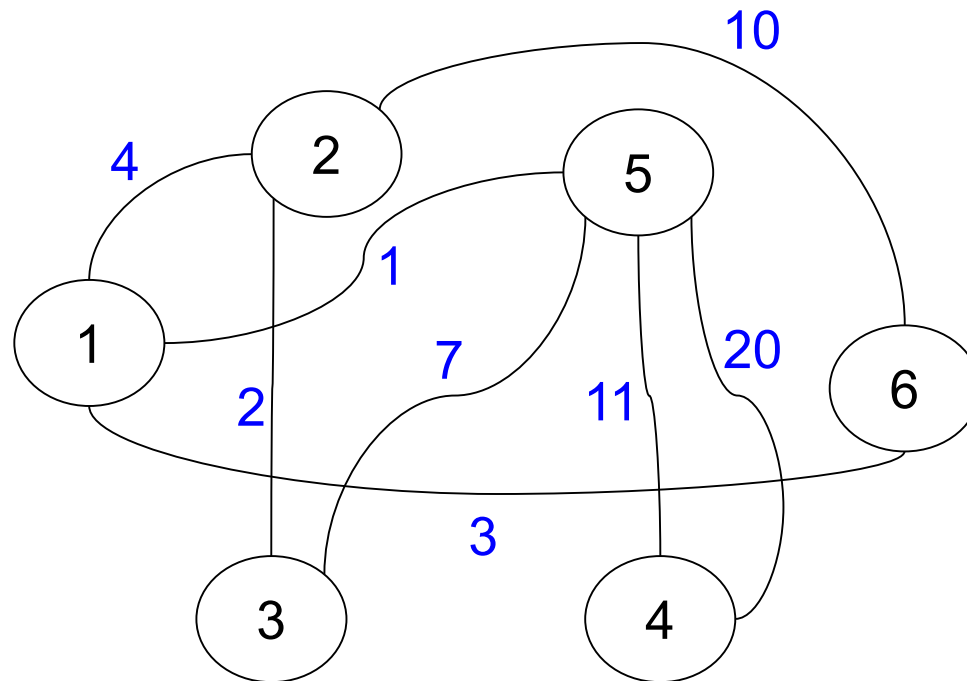
Fila: ~~(0,2,1)~~ ~~(1,2,2)~~ ~~(2,3,2)~~ ~~(4,5,3)~~ ~~(3,5,4)~~ ~~(2,5,4)~~ (1,4,5) (0,3,5) (0,1,6) (2,4,6)



Algoritmo de Kruskal

■ Exercício

- Encontre uma árvore geradora mínima para o grafo abaixo utilizando o algoritmo de Kruskal



Algoritmo de Kruskal: exemplo

- **Exercício**

- Implementação do algoritmo de Kruskal

Procedure MST-Kruskal(G)

1. $A := \emptyset$
2. **For** each vertex v in V **do**
 make-set(v)
End For
3. Ordenar arestas de A em ordem crescente de peso w
4. **For** each (u,v) in A (em ordem crescente de peso) **do**
 If *find-set*(u) \neq *find-set*(v) **then**
 $A := A \cup \{(u,v)\}$
 union(u,v)
 End If
End For

make-set(u): cria conjunto com vértice u

find-set(u): retorna um elemento representativo do conjunto que contém u

union(u,v): combina as árvores

Algoritmo de Kruskal

- Complexidade: $O(|A| \log(|V|))$
 - Se bem implementado (depende da estrutura de dados adotada para representar os conjuntos!)
 - discussão no livro do Cormen, cap. 23