Árvore Geradora Mínima

prof. Leandro G. M. Alvim

Árvore Geradora Mínima (MST)

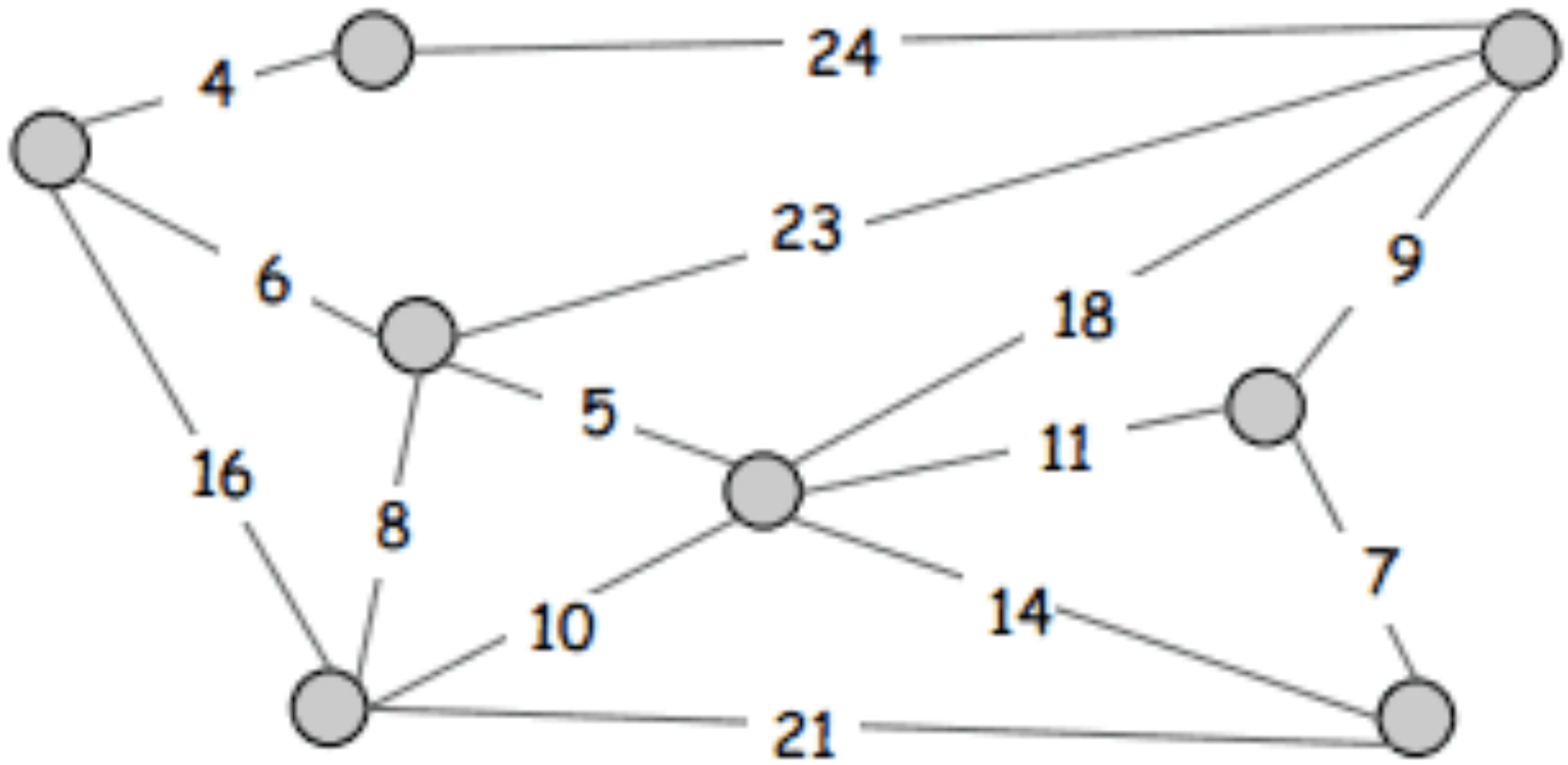
- Seja um grafo G=(V,E) com custos c_e nas aretas, uma MST é um subconjunto de arestas T c E tal que T é um árvore geradora cuja soma dos custos das aretas é a menor possível. (Kleinberg e Tardos, 2005)

Árvore Geradora Mínima (MST)

- Seja um grafo G=(V,E) com custos c_e nas aretas, uma MST é um subconjunto de arestas T c E tal que T é um árvore geradora cuja soma dos custos das aretas é a menor possível. (Kleinberg e Tardos, 2005)

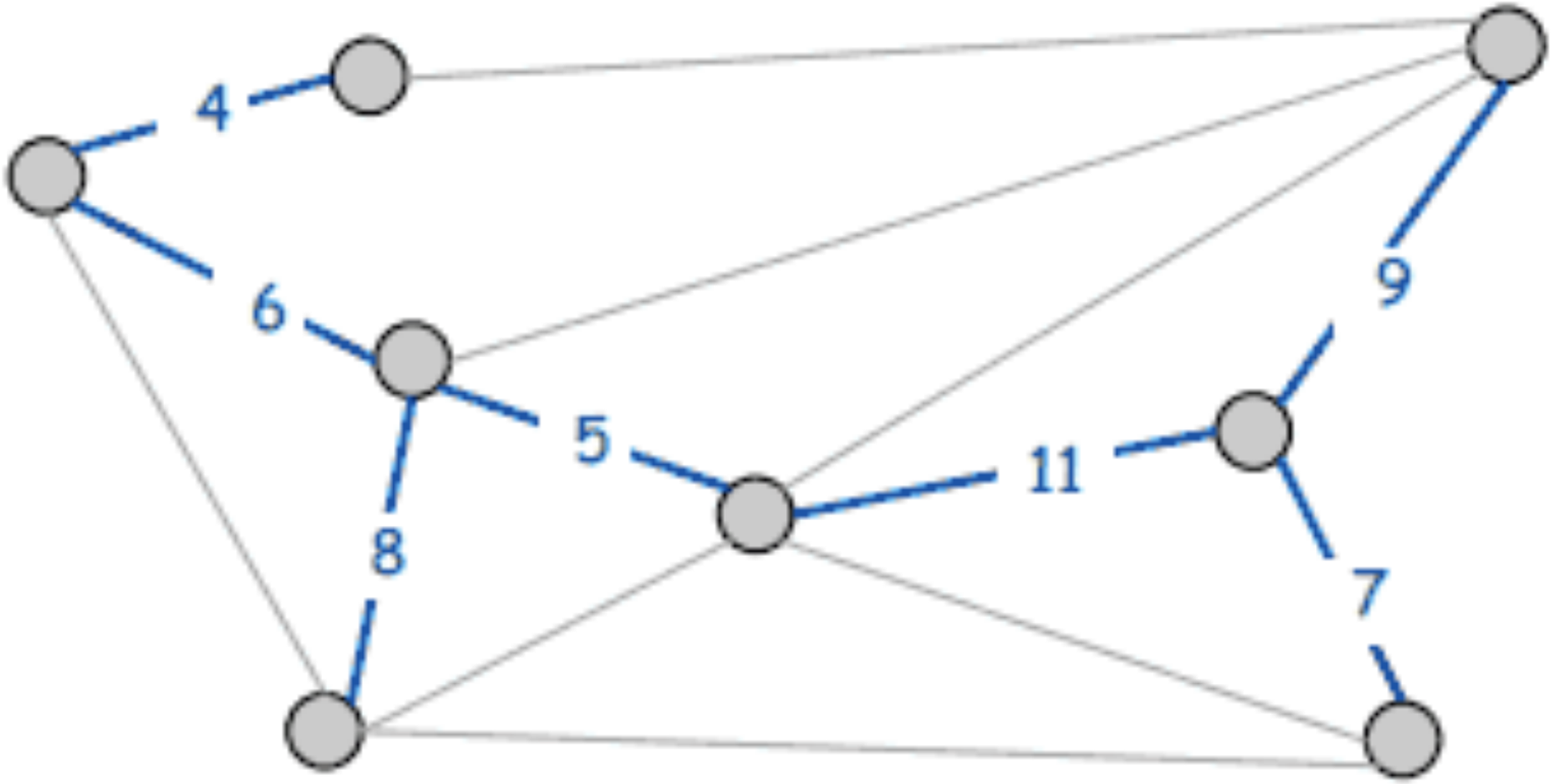
- Teorema de Caley (n^(n-2) árvores)
  - Inviável por força bruta

Árvore Geradora Mínima (MST)



$G = (V, E)$

Árvore Geradora Mínima (MST)



T, $\Sigma_{e \in T} c_e = 50$

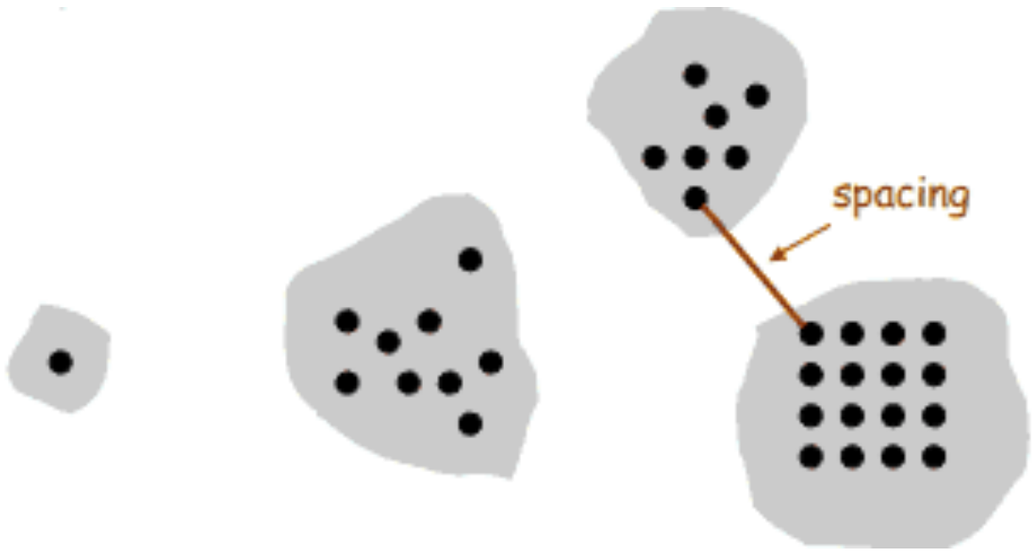
Algumas Aplicações

- Projeto de Redes
  - Tv a cabo, rede elétrica, hidráulica, computadores, estradas
- Agrupamento
- Algoritmos aproximativos para problemas da classe NP-Difícil (ex. Caixeiro Viajante)

TV a Cabo

- Conectar regiões com o menor custo possível
  - Certos "caminhos" precisam de cabos maiores
  - Em certos "caminhos" é necessário passar o cabo em um maior profundidade

Agrupamento



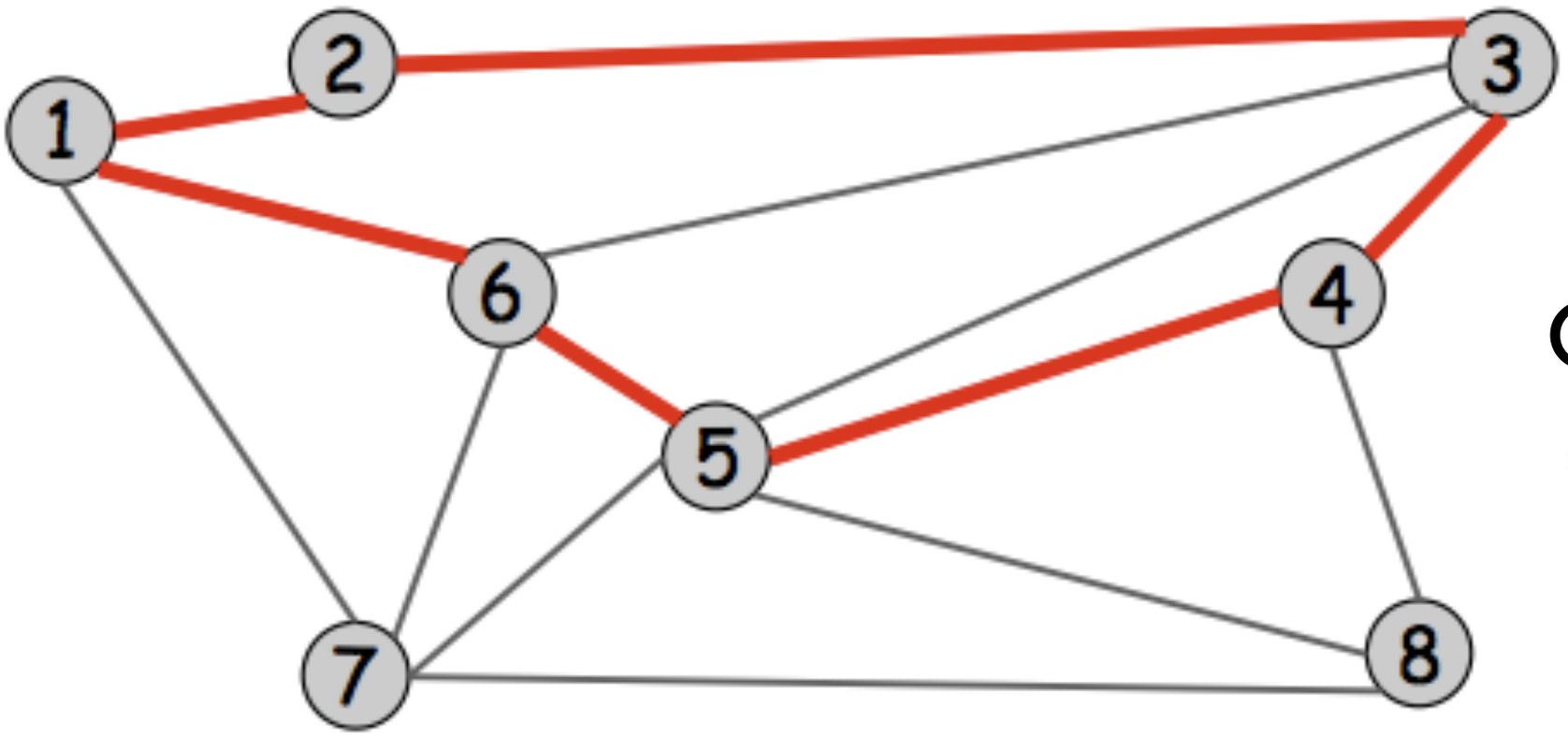
- Agrupar vértices em k grupos
  - Encontrar MST
  - Remover k-1 maiores arestas

Algoritmos para MST

- Kruskal
  - Comece com T={}. Insira, em ordem crescente de custo, arestas que não gerem ciclos
- Prim
  - Escolha um vértice s. Construa uma árvore T a partir de s adicionando o vértice cuja aresta possui menor custo e que possua apenas um vertice em T.
- Remoção Reversa
  - Comece com T=E. Seja as arestas em ordem decrescente, remova cada aresta e que não desconecta T.

Ciclo

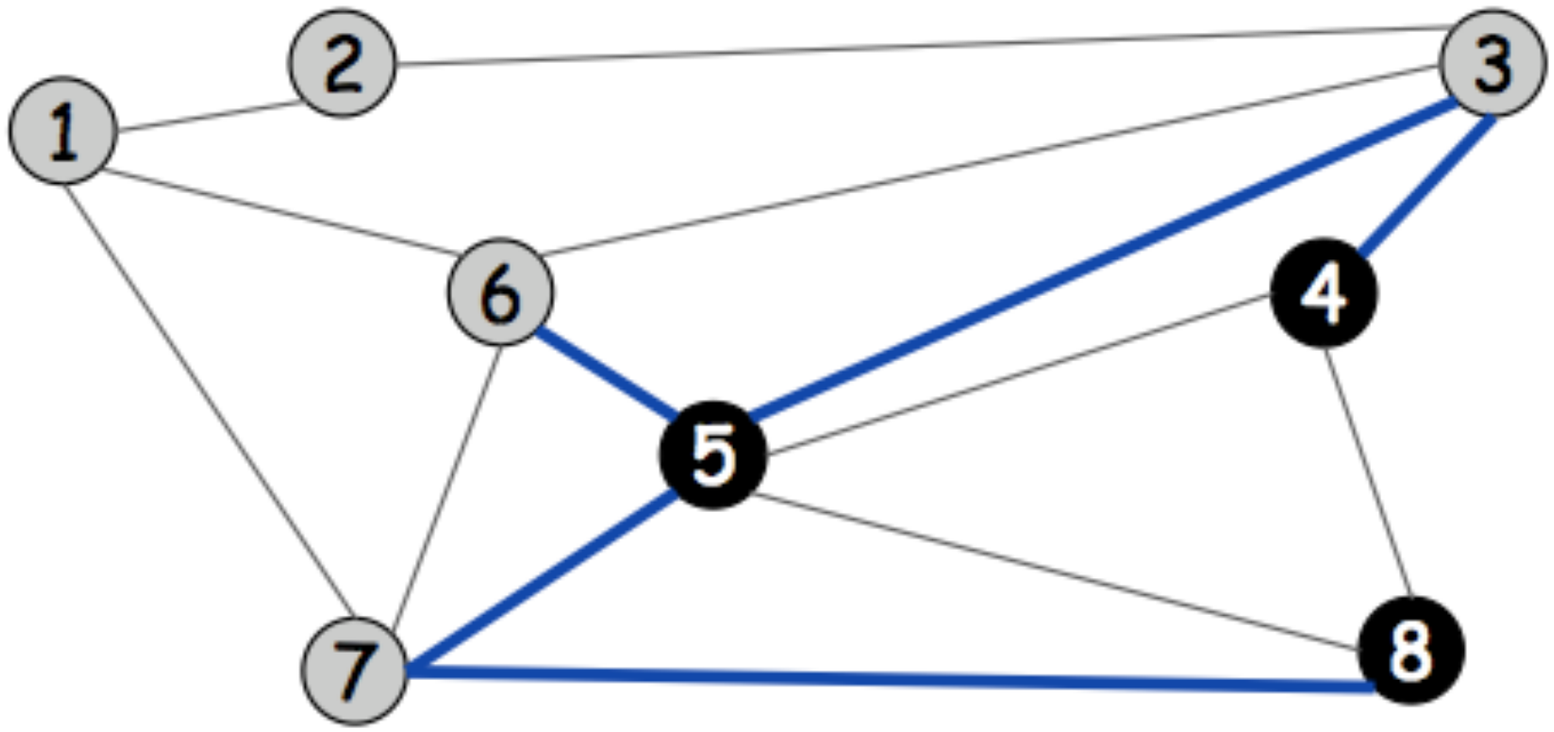
- Conjunto de arestas na forma (a,b), (b,c), ..., (y,z).



Ciclo C = {(1,2), (2,3),(3,4), (4,5), (5,6), (6,1)}

Conjunto de Corte

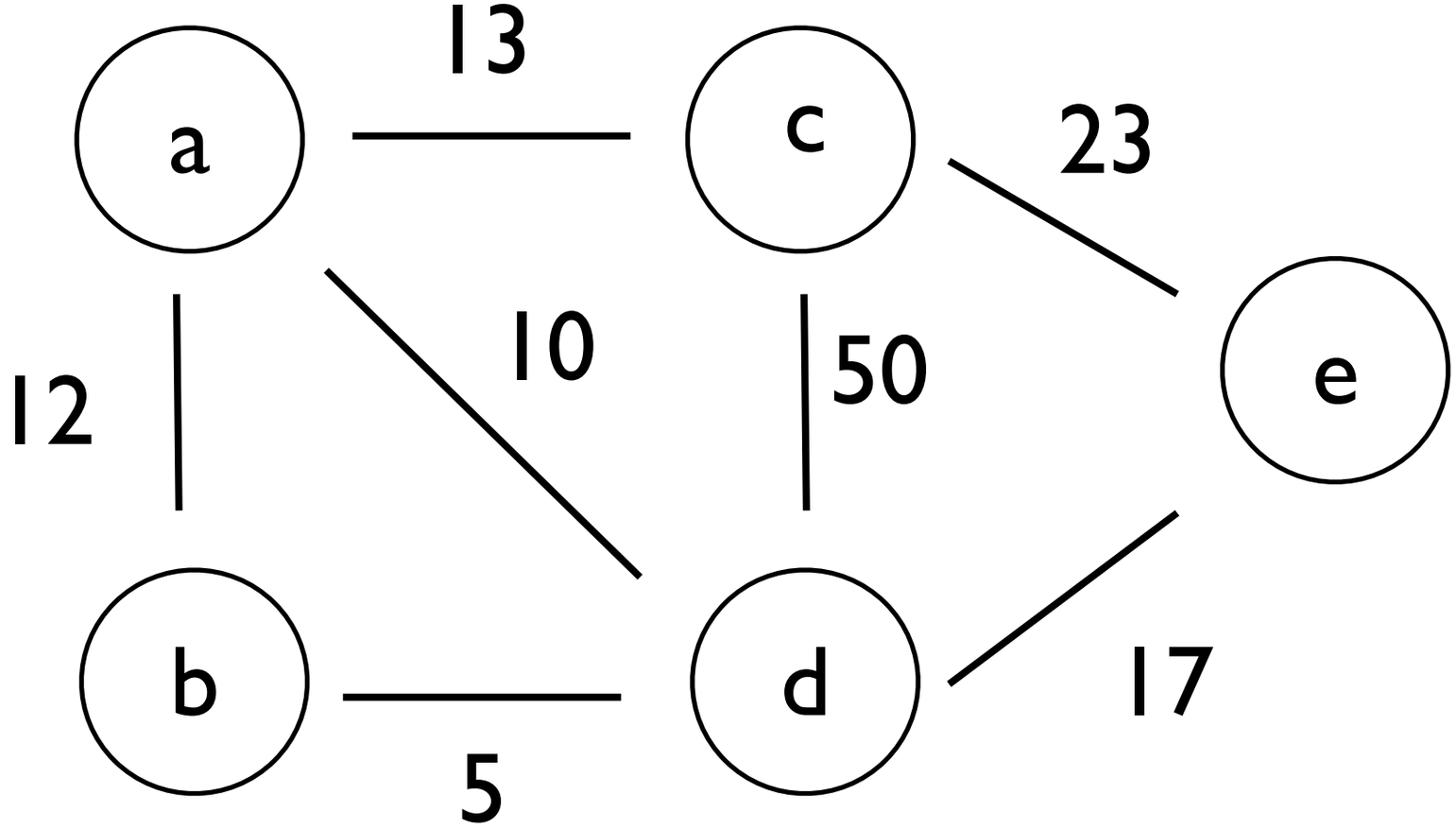
- Um corte é um subconjunto dos vértices de S. O correspondente conjunto de corte D é o subconjunto de arestas com exatamente um vértice em S.



Corte S = {5,4,8}

Conjunto de corte D = {(5,6), (5,7), (5,3), (4,3), (8,7)}

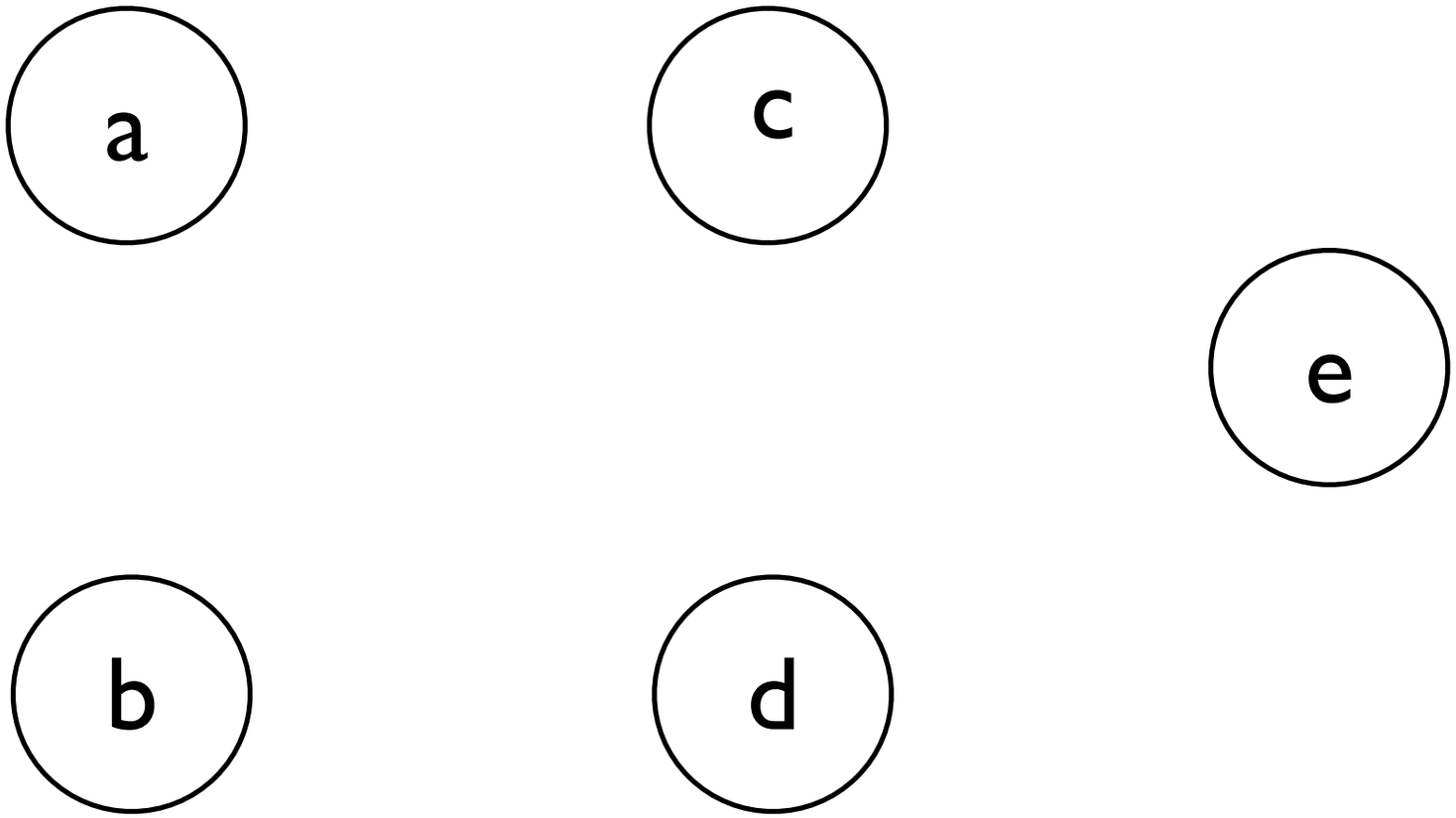
Kruskal



Arestas ordenadas

Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal

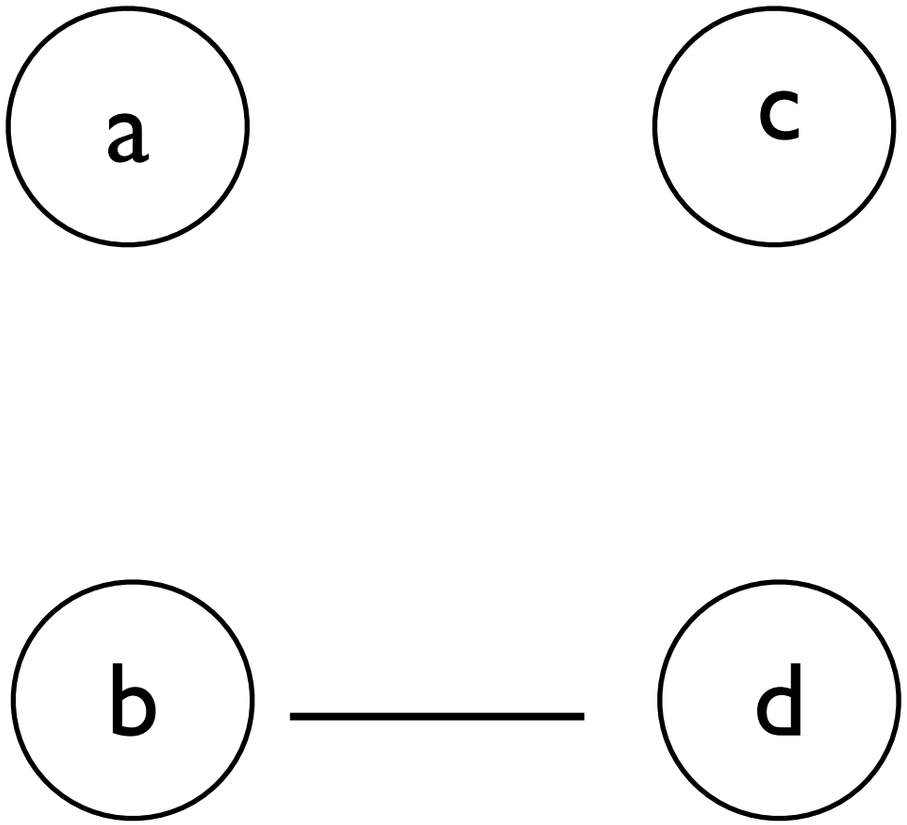


Floresta

Arestas ordenadas

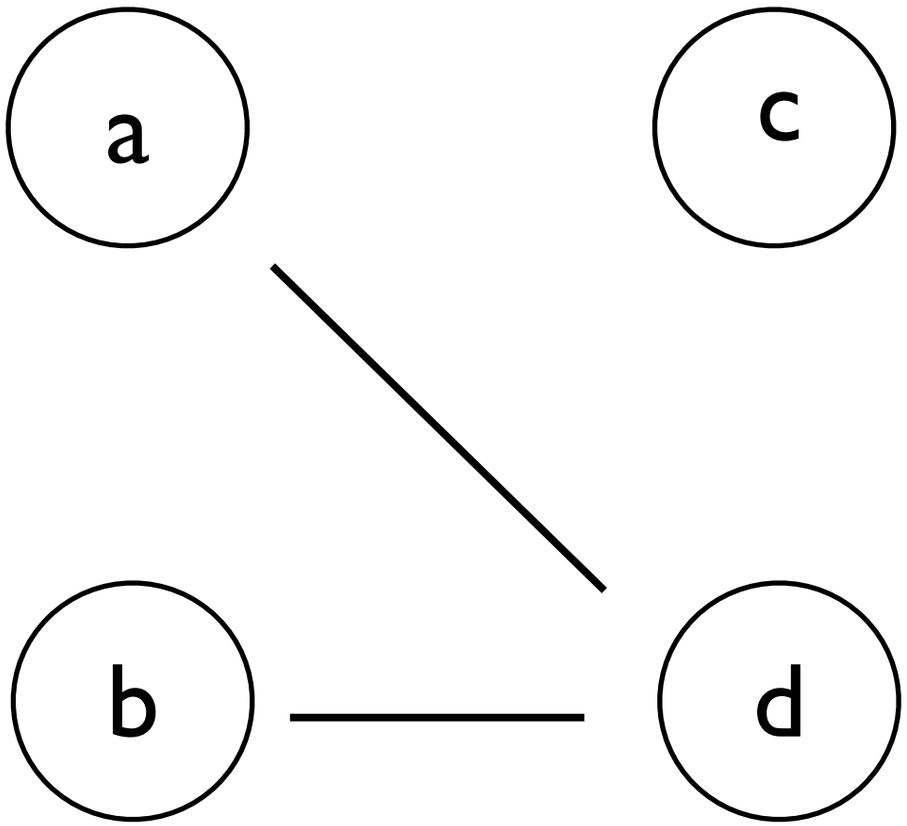
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



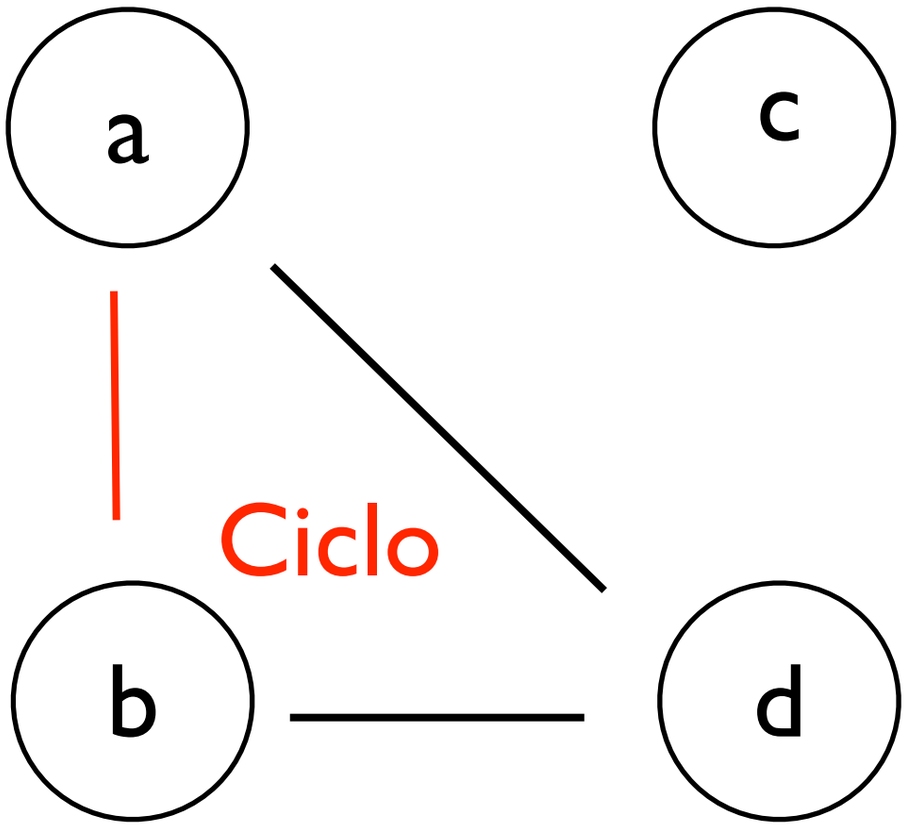
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



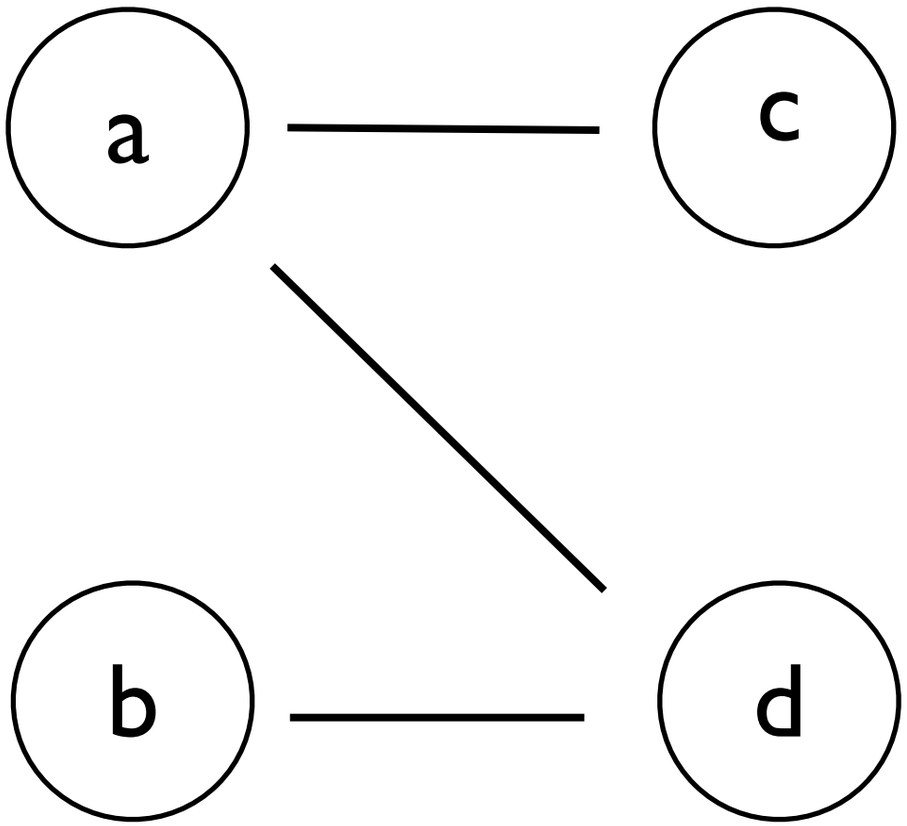
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



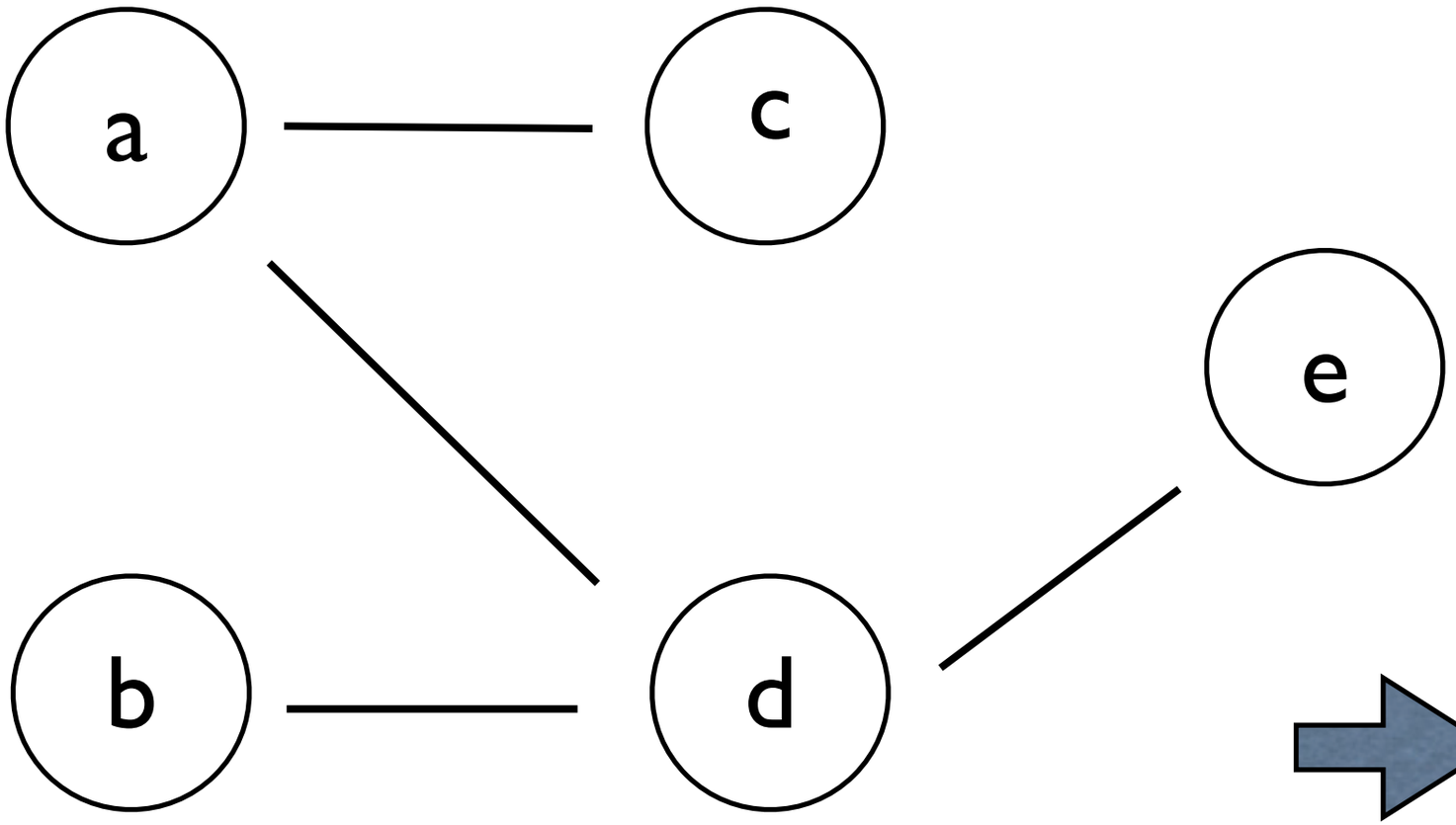
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



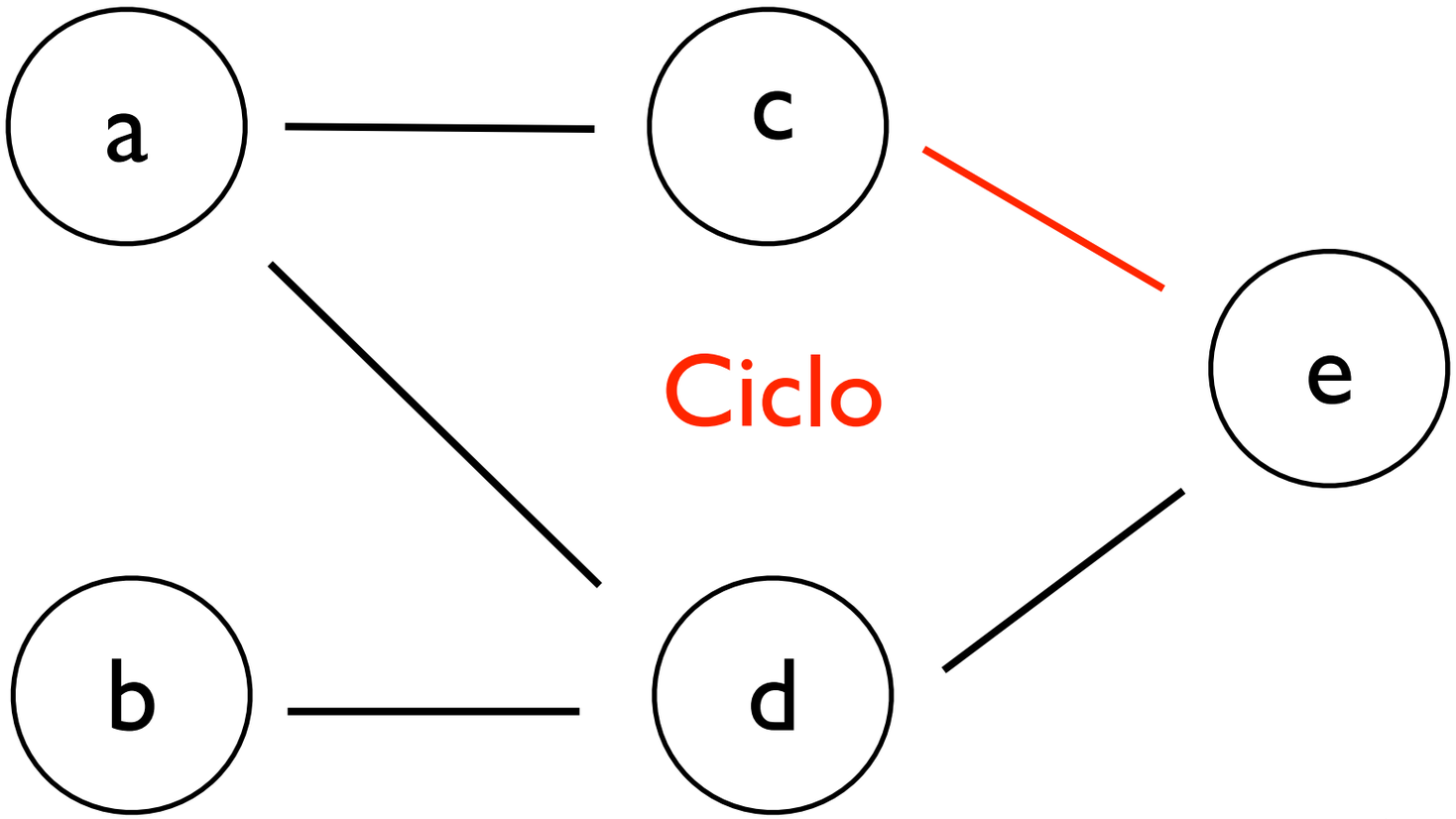
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



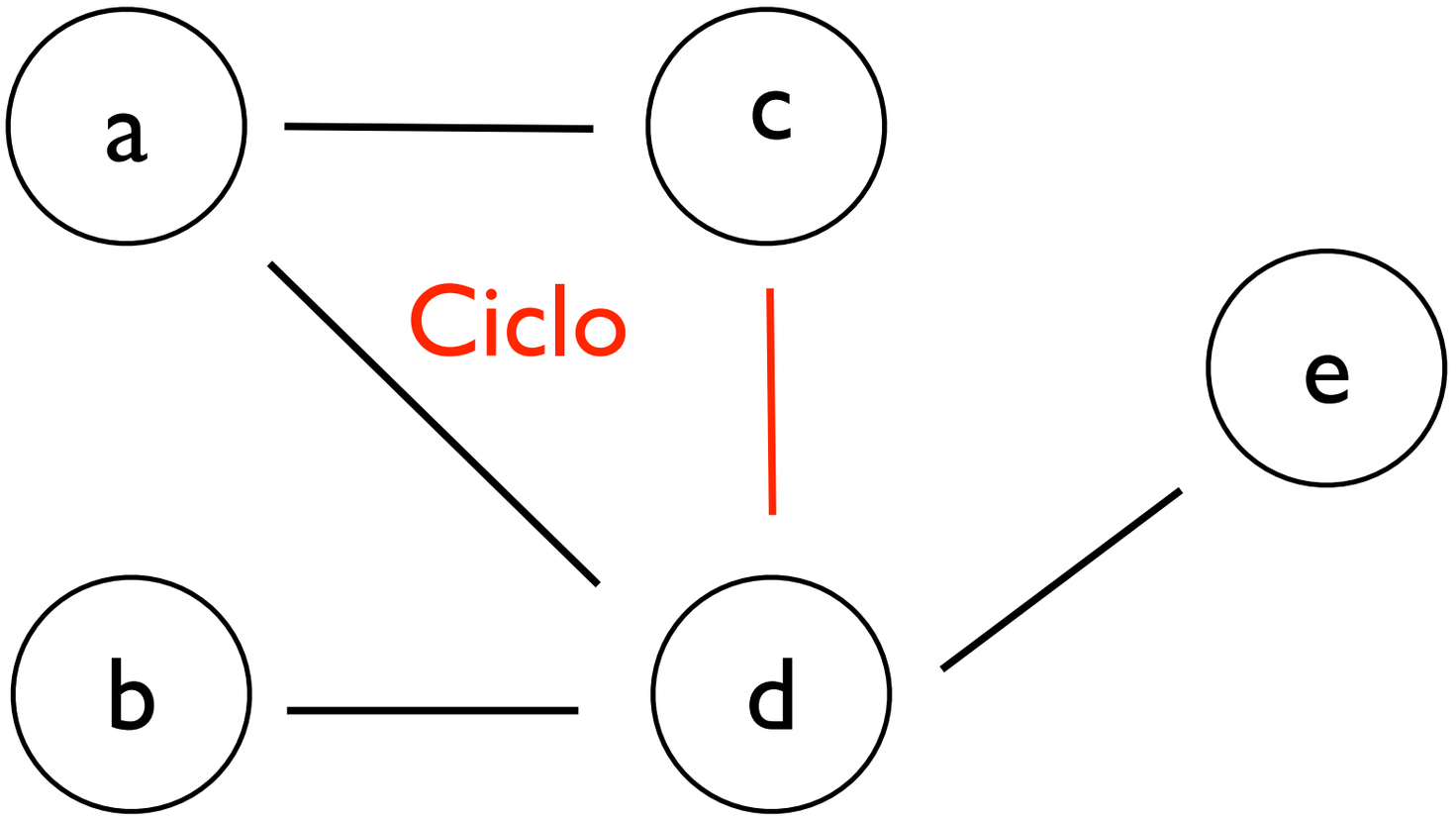
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



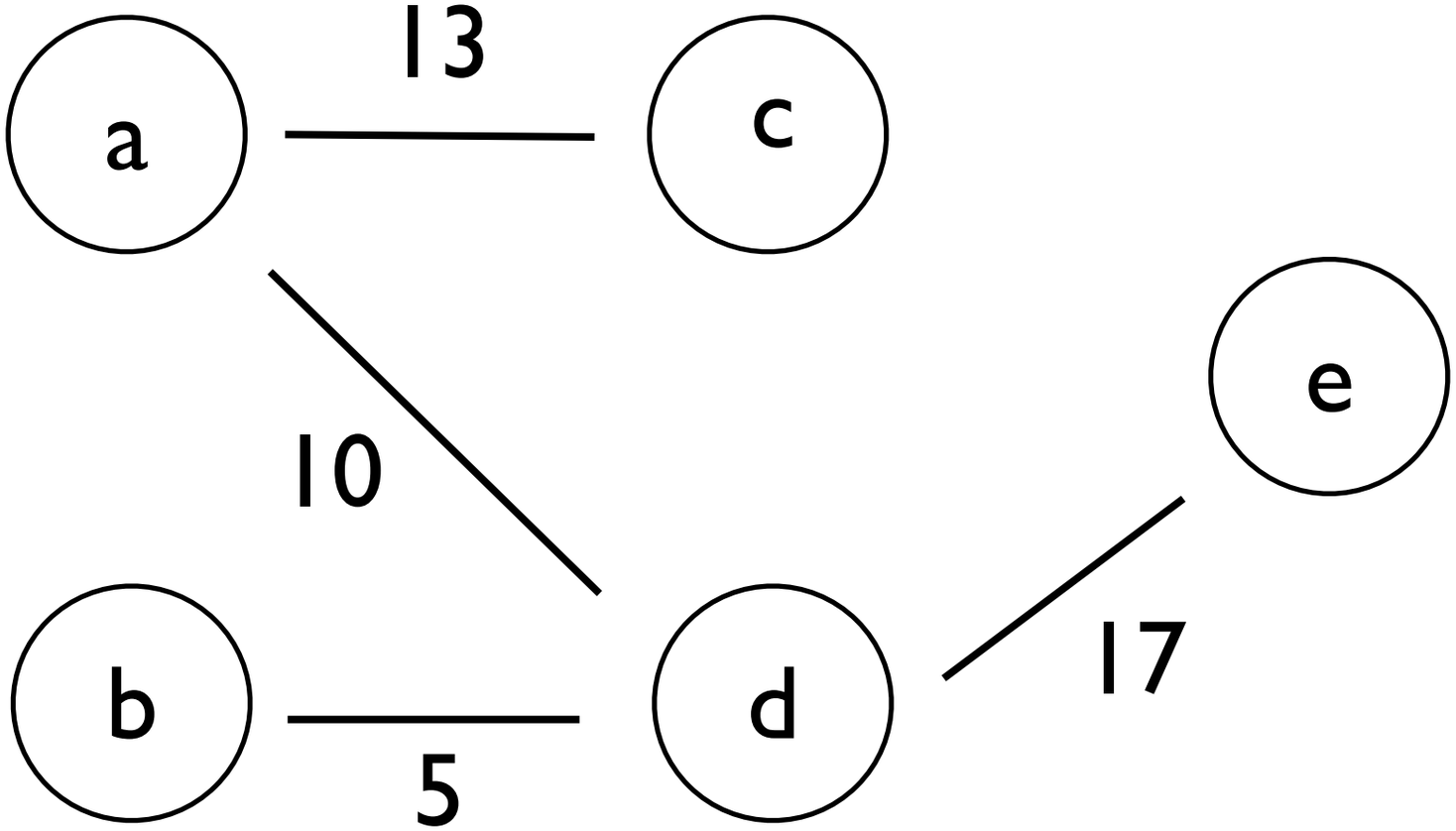
Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

Kruskal



Aresta	Peso
{b,d}	5
{a,d}	10
{a,b}	12
{a,c}	13
{d,e}	17
{c,e}	23
{c,d}	50

C(MST) = 45

Kruskal

- Passos
  - Comece com uma floresta
  - Adicione a menor aresta que não gere ciclo
  - Volte ao passo anterior até que a floresta se transforme em árvore
- Observação: Ao contrário do Prim, que cresce a partir de uma árvore, kruskal cresce com florestas até que se obtenha

Implementação

- Requer ordenacão de arestas: O(E log(E))
- Requer estrutura de dados Union-Find
  - createSet(u): O(1)
  - findSet(u): O(log(V))
  - union(u,v): O(log(V))

Implementação

- sorted(E) O(E log(E))
- a = {}
- **for** u **in** V: O(V)
  - createSet(u)
- **for** (u,v) **in** E: O(E log(V))
  - **if** findSet(u) != findSet(v):
    - a.append((u,v))
    - union(u,v)

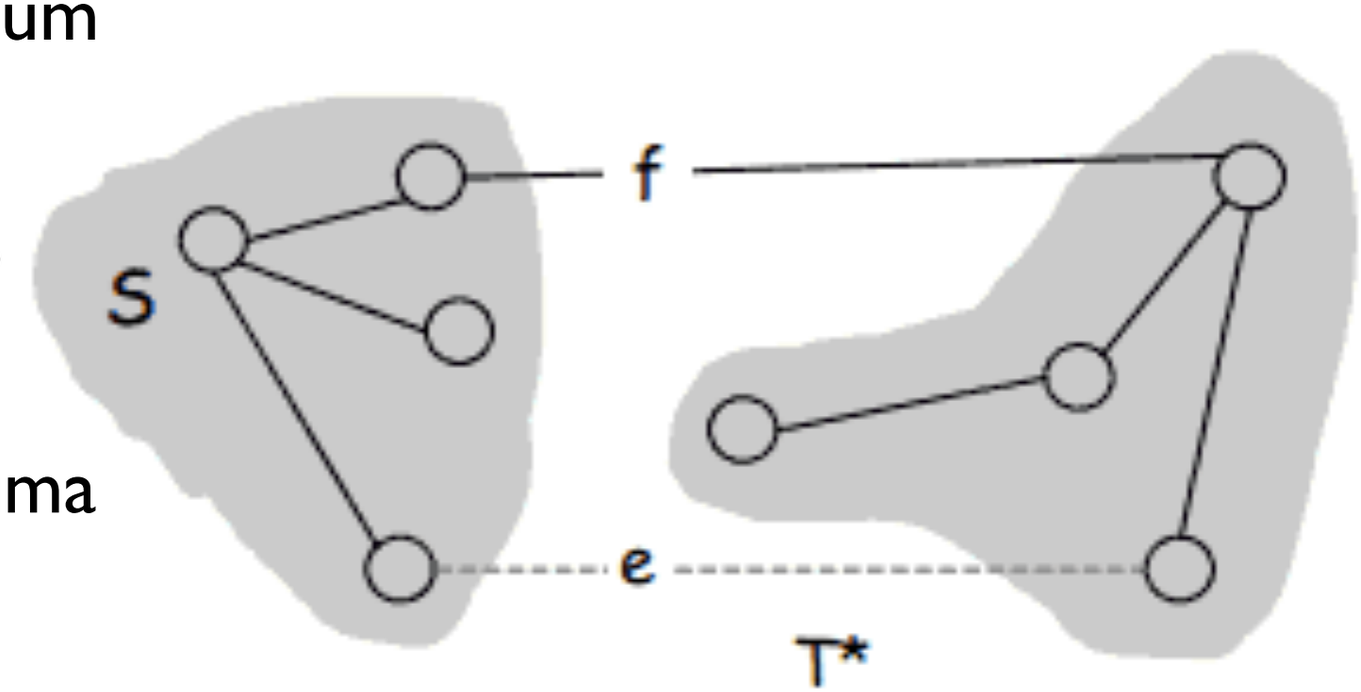
Implementação

- Kruskal
  - Com implementação union-find eficiente
  - O(E log(E))
  - Como E < V^2, então log(E) = O(log(V))
    - O(Elog(V))

Corretude

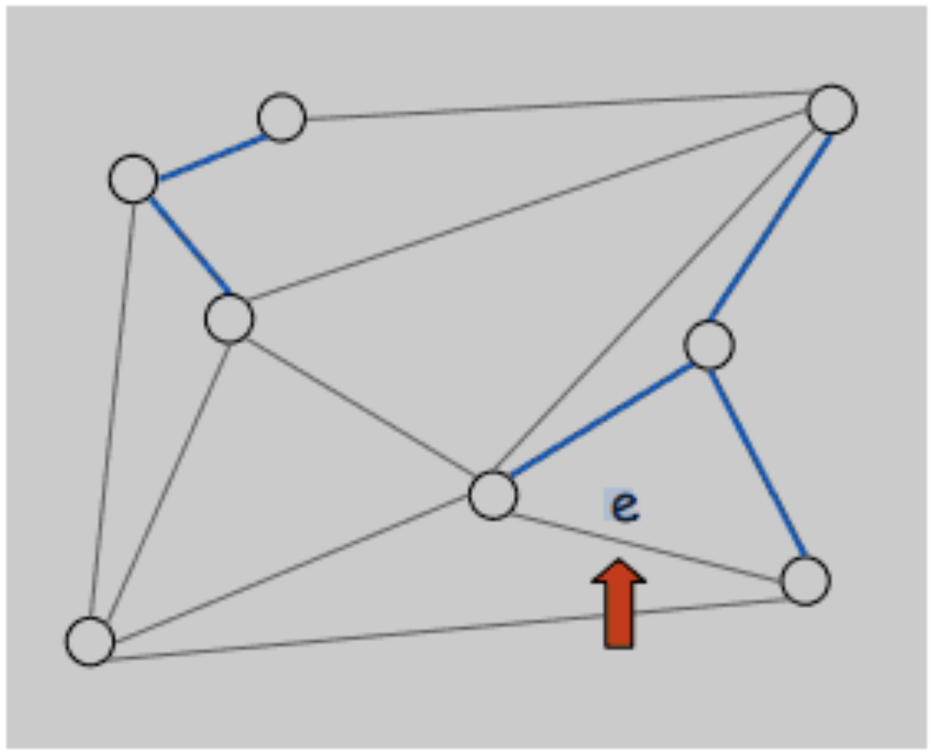
Propriedade do corte: Seja S um subconjunto qualquer de vértices e e a aresta de menor custo com exatamente um vértice em S. Então MST T* contém e.

- Prova
  - Suponha que e não pertença a T*,
  - então, adicionar e em T* cria um ciclo C
  - Aresta e está no ciclo C e no conjunto de corte D
  - T'= T* U {e} - {f} também é uma MST
  - Como $c_e < c_f$, Custo(T') < Custo(T*) (Contradição)



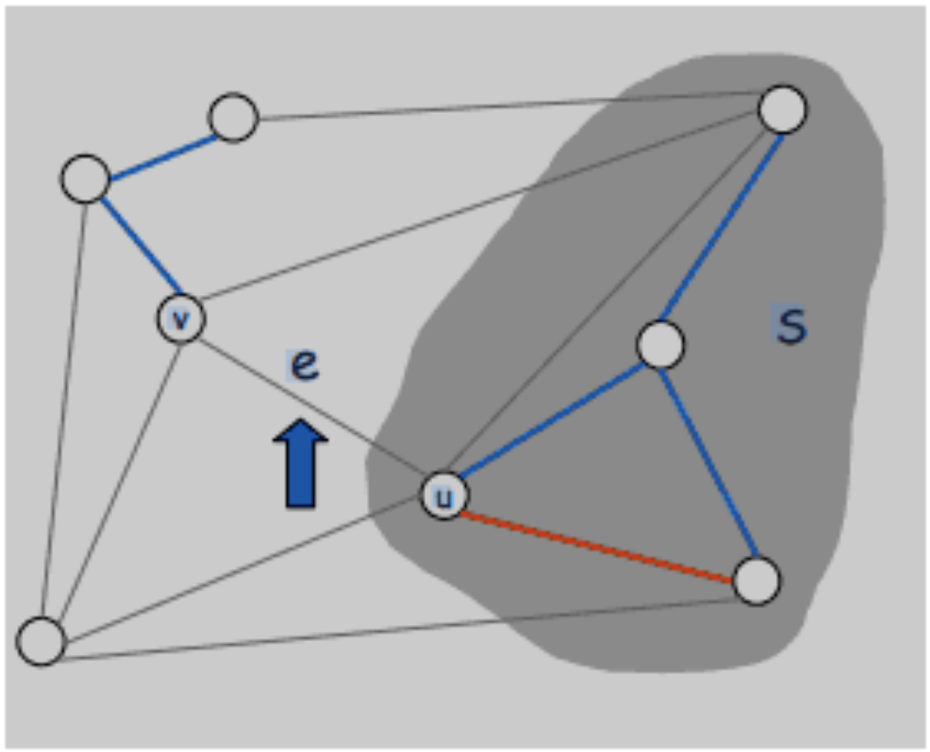
Corretude

Arestas em ordem crescente



Case 1

Caso 1: Se aresta e gera um ciclo, descarte.



Case 2

Caso 2: Caso contrário, insira aresta e em T de acordo com a propriedade do corte em que S é o conjunto de vértices da mesma componente conexa de u

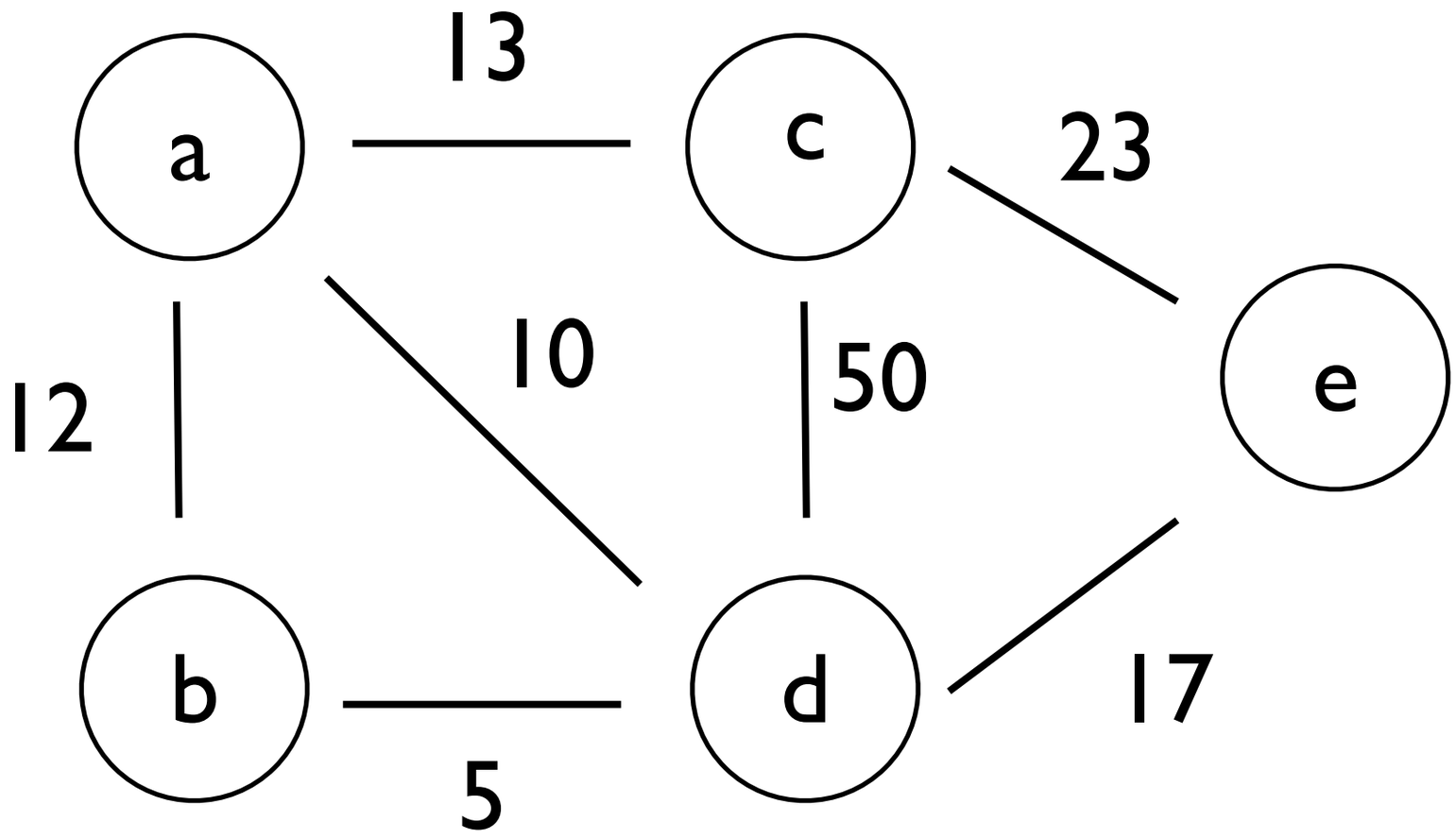
Corretude

Theorem 13.3 *Kruskal's algorithm correctly finds a minimum spanning tree of the given graph.*

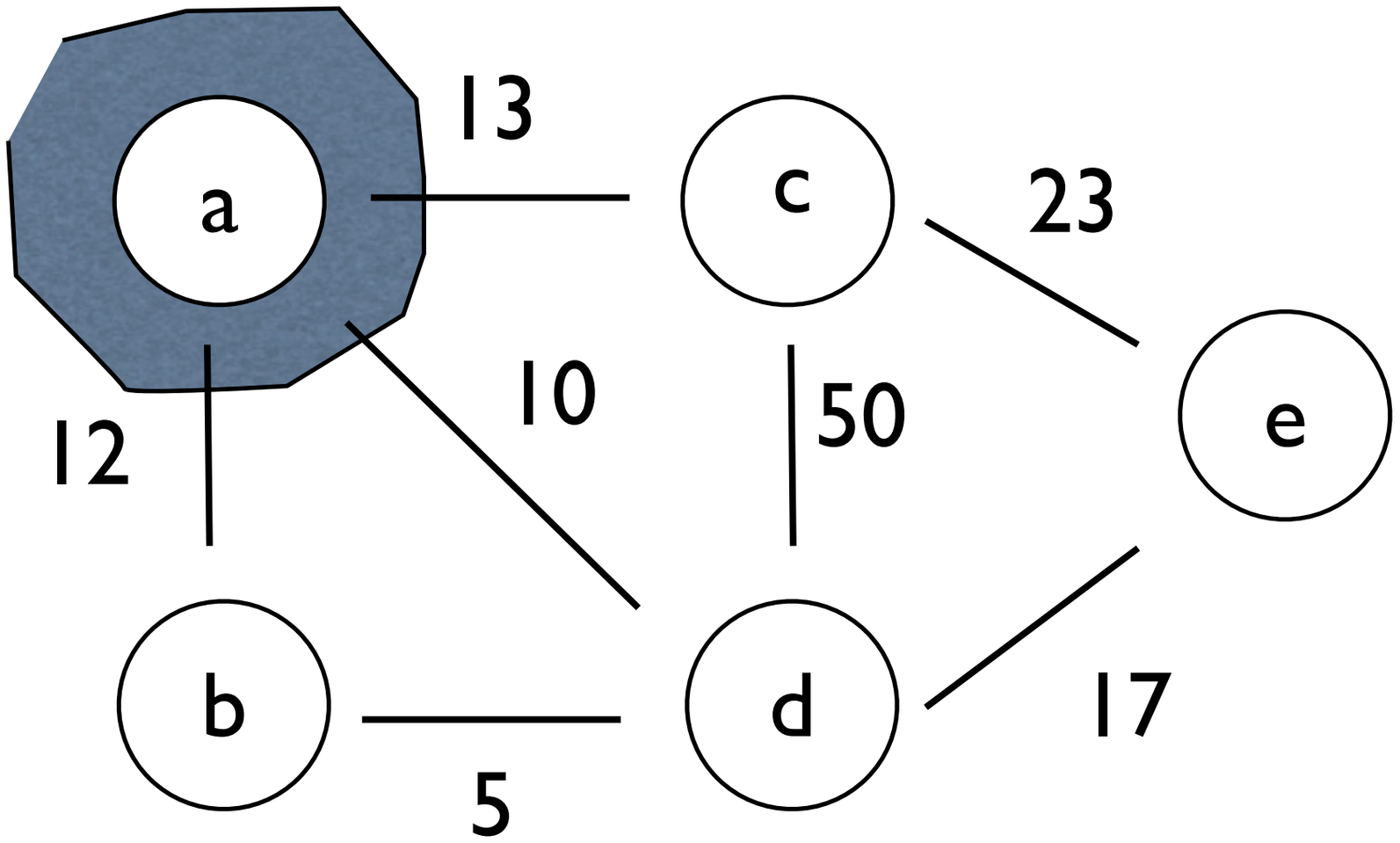
Proof: We can use a similar argument to the one we used for Prim's algorithm. Let G be the given graph, and let F be the forest we have constructed so far (initially, F consists of n trees of 1 node each, and at each step two trees get merged until finally F is just a single tree at the end). Assume by induction that there exists an MST M of G that is consistent with F, i.e., all edges in F are also in M; this is clearly true at the start when F has no edges. Let e be the next edge added by the algorithm. Our goal is to show that there exists an MST M' of G consistent with $F \cup \{e\}$.

If $e \in M$ then we are done ($M' = M$). Else add e into M, creating a cycle. Since the two endpoints of e were in different trees of F, if you follow around the cycle you must eventually traverse some edge $e' \neq e$ whose endpoints are also in two different trees of F (because you eventually have to get back to the node you started from). Now, both e and e' were eligible to be added into F, which by definition of our algorithm means that $len(e) \leq len(e')$. So, adding e and removing e' from M creates a tree M' that is also a MST and contains $F \cup \{e\}$, as desired. ■

Prim



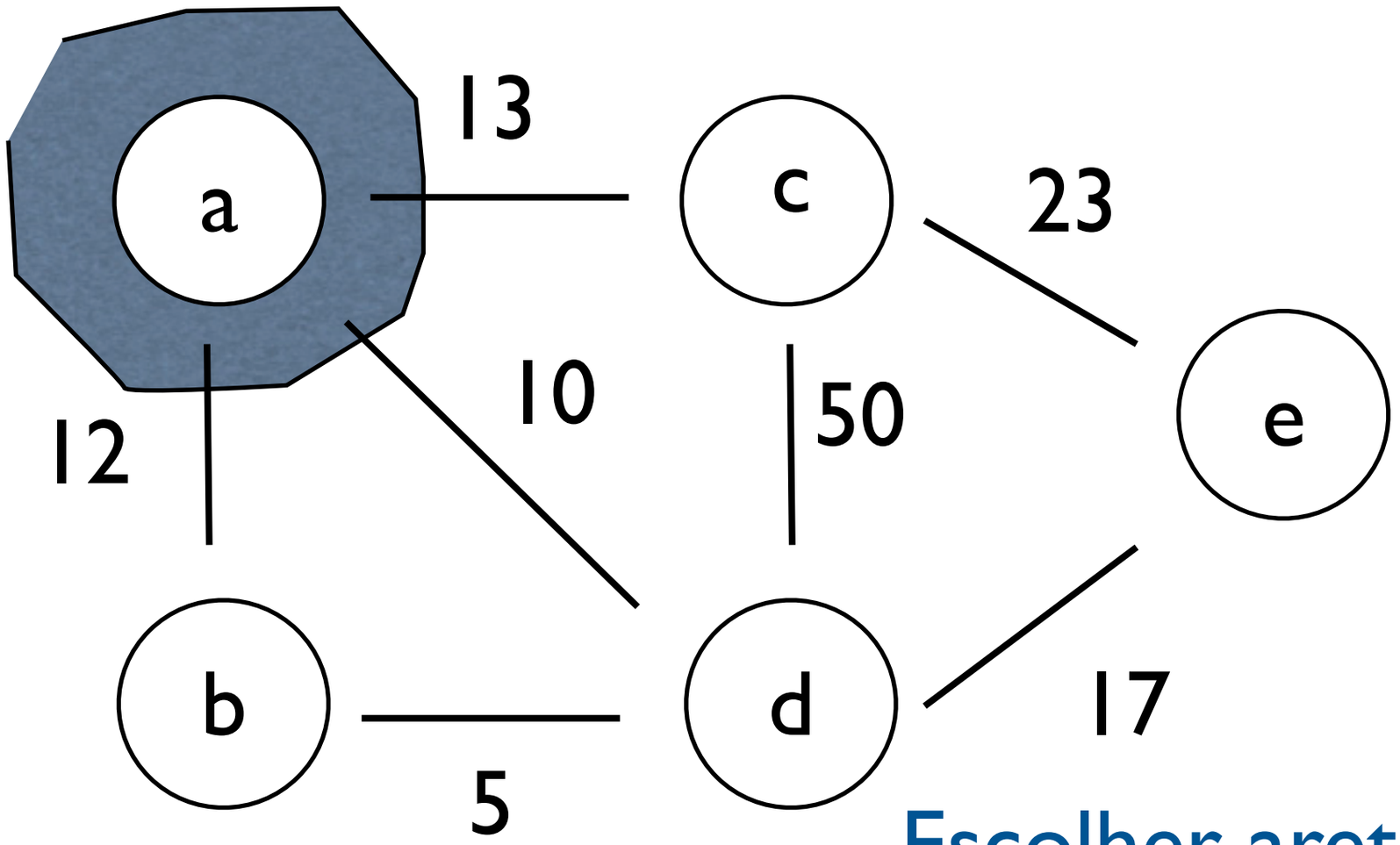
Prim



S = {a}

v	Pred(v)
a	-1

Prim

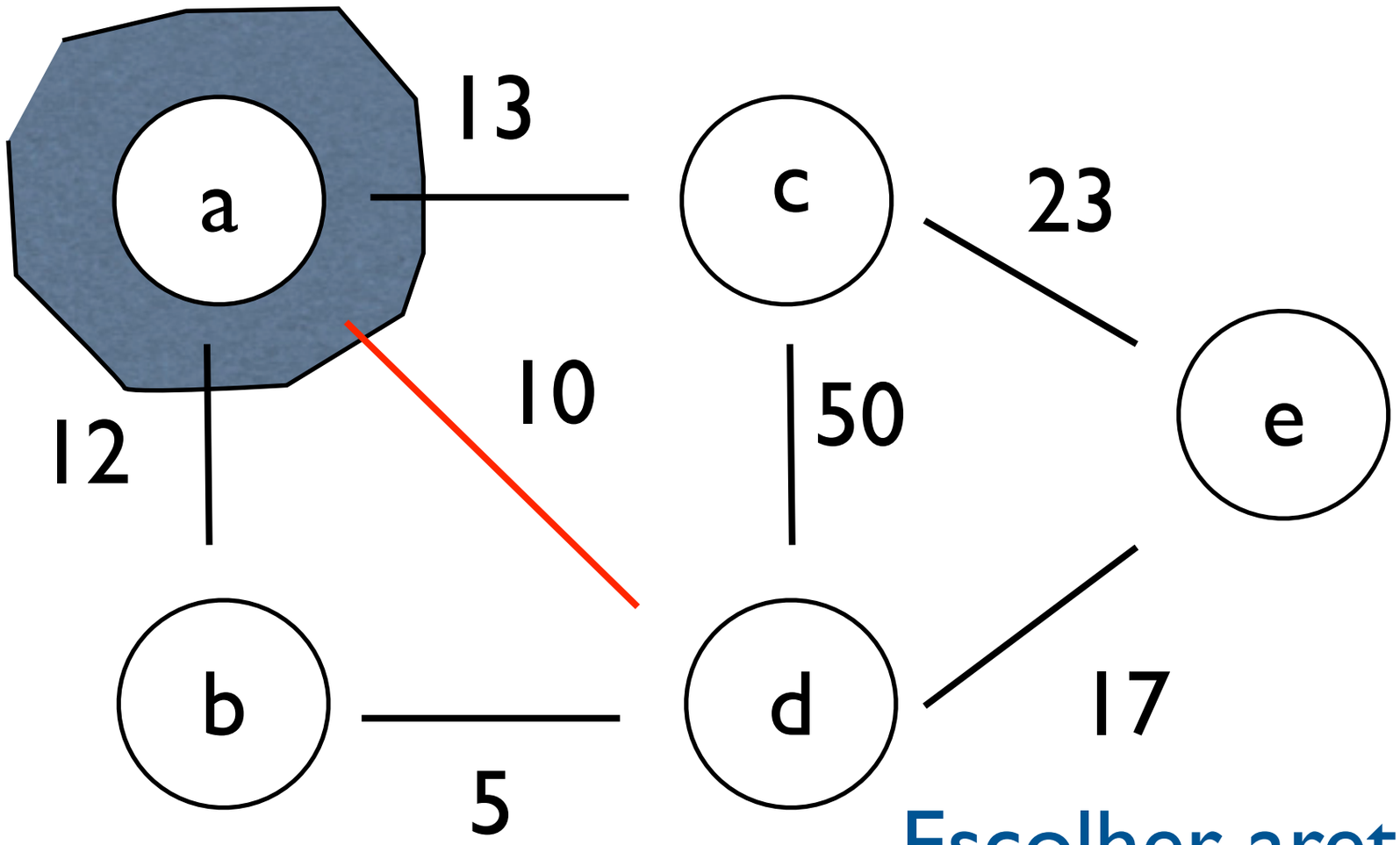


S = {a}

v	Pred(v)
a	-1

Escolher areta
dentro do corte
com custo mínimo

Prim

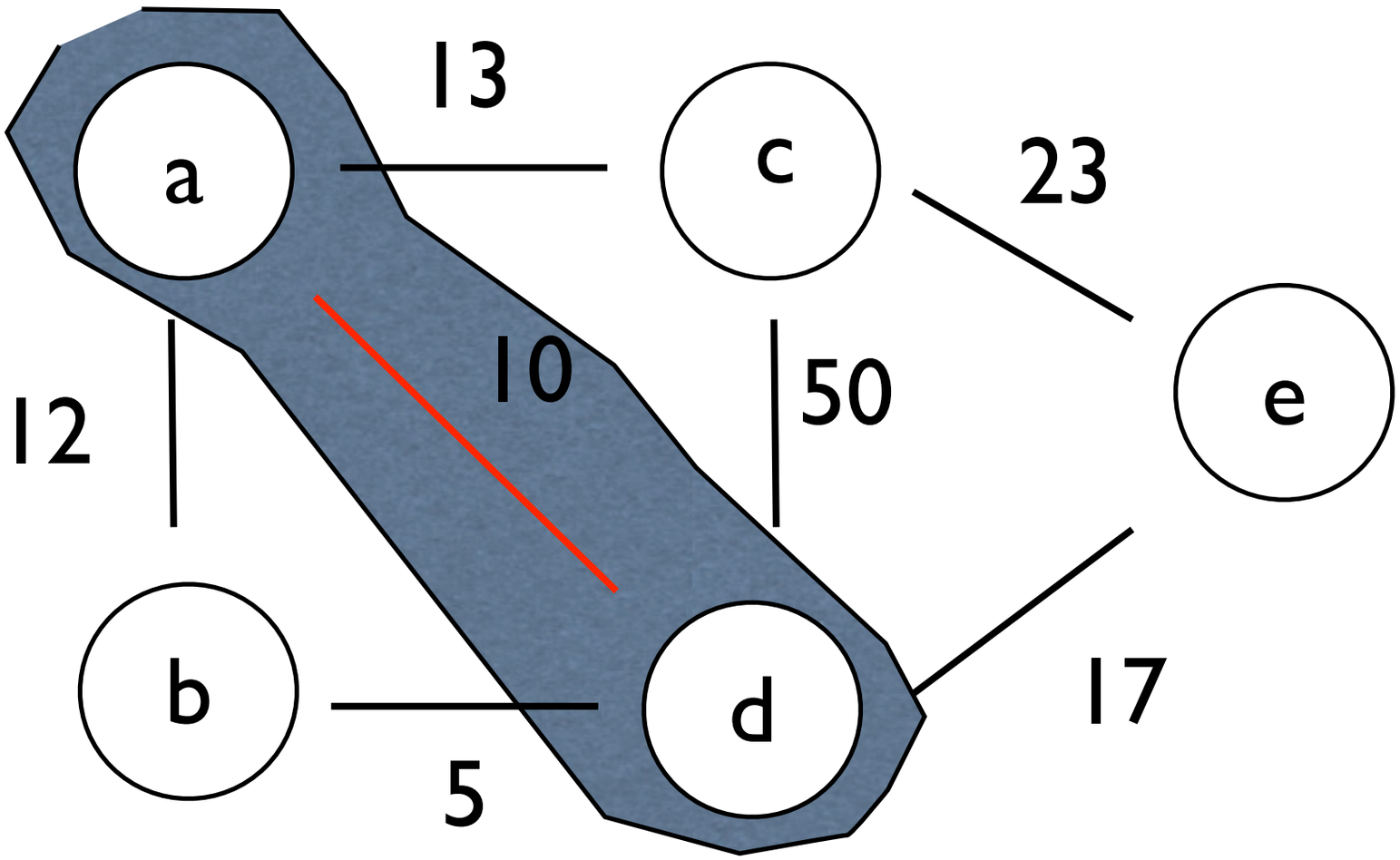


S = {a}

v	Pred(v)
a	-1

Escolher areta
dentro do corte
com custo mínimo

Prim

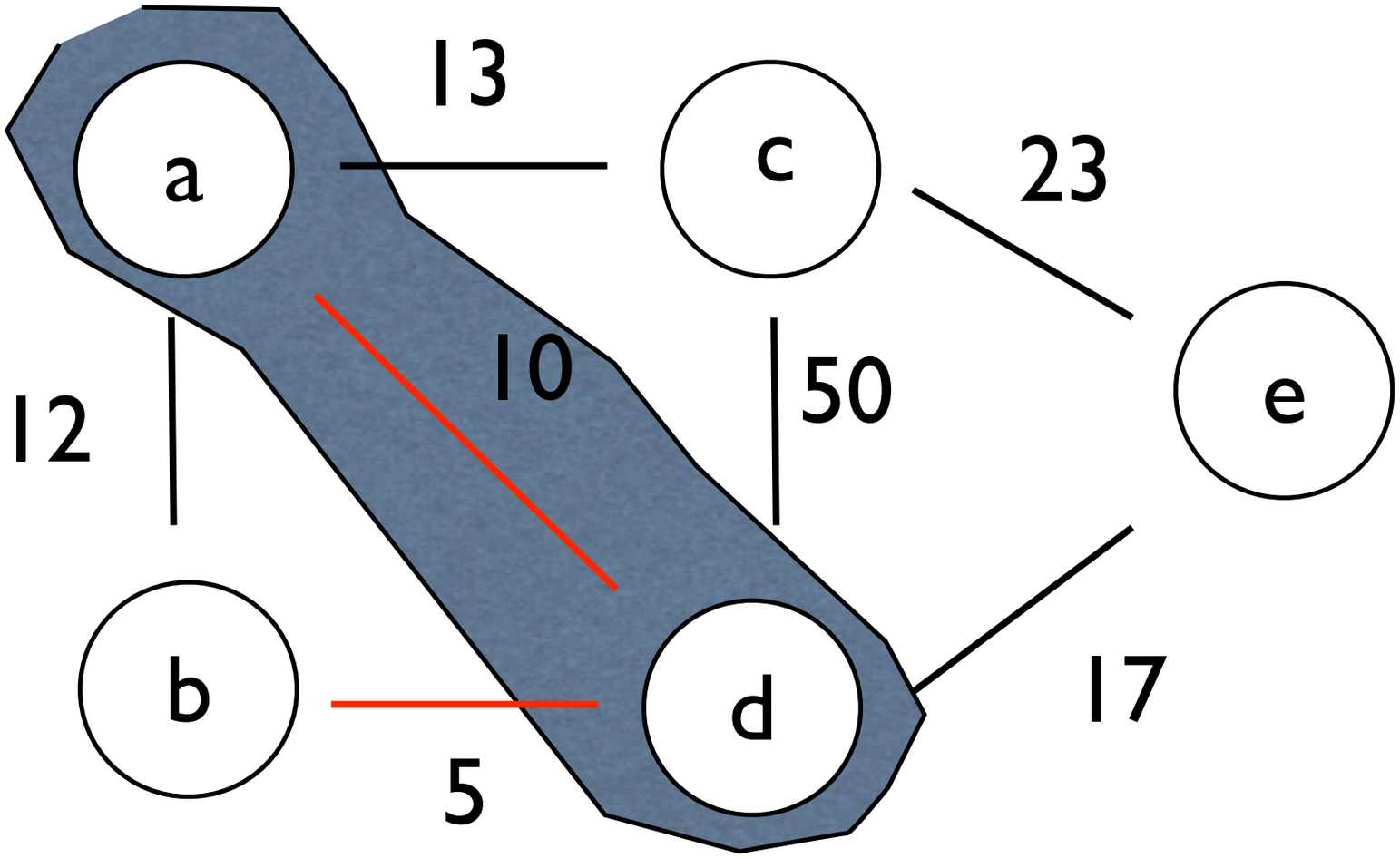


S = {a,d}

v	Pred(v)
a	-1
d	a

Adicionar novo vértice em S

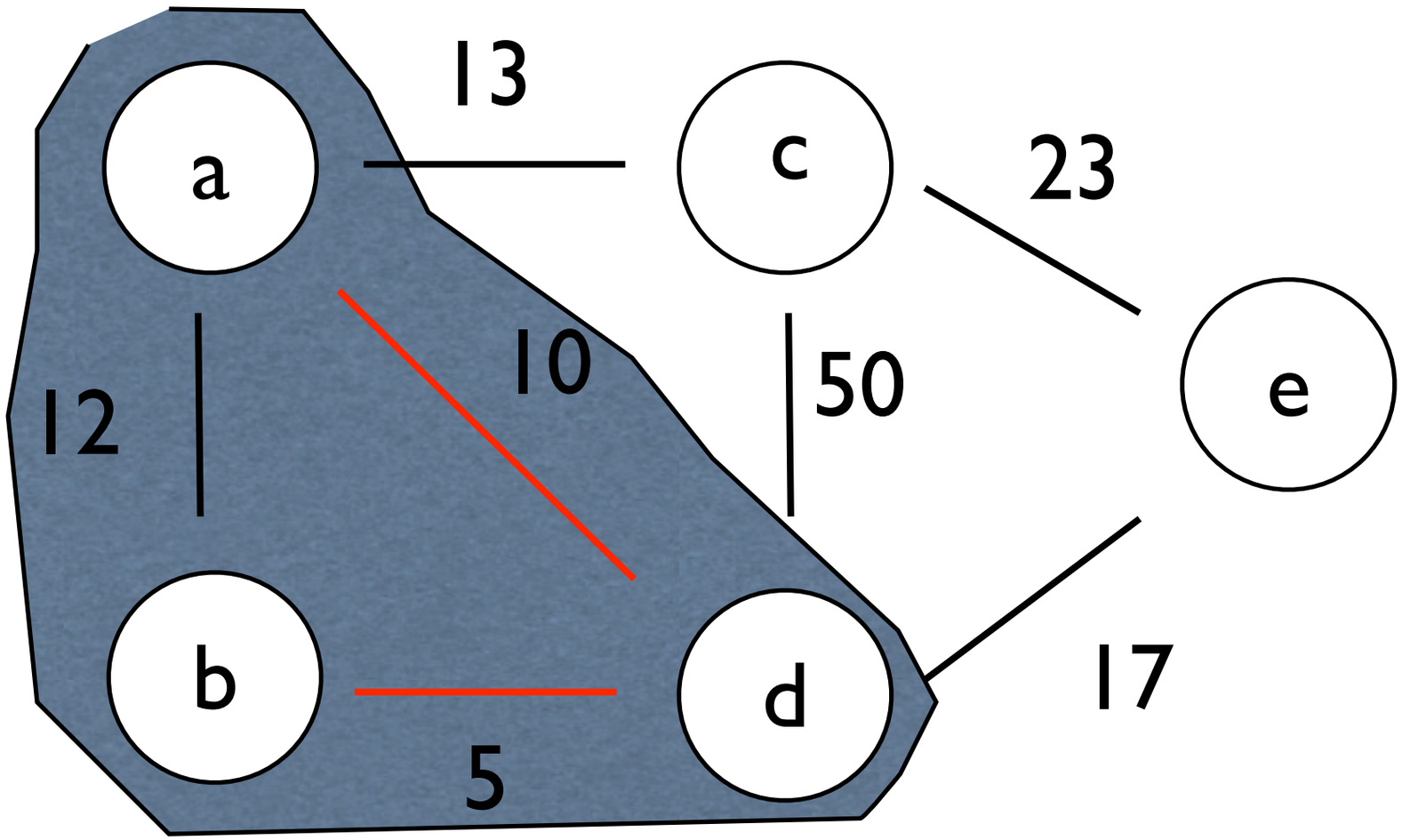
Prim



S = {a,d}

v	Pred(v)
a	-1
d	a

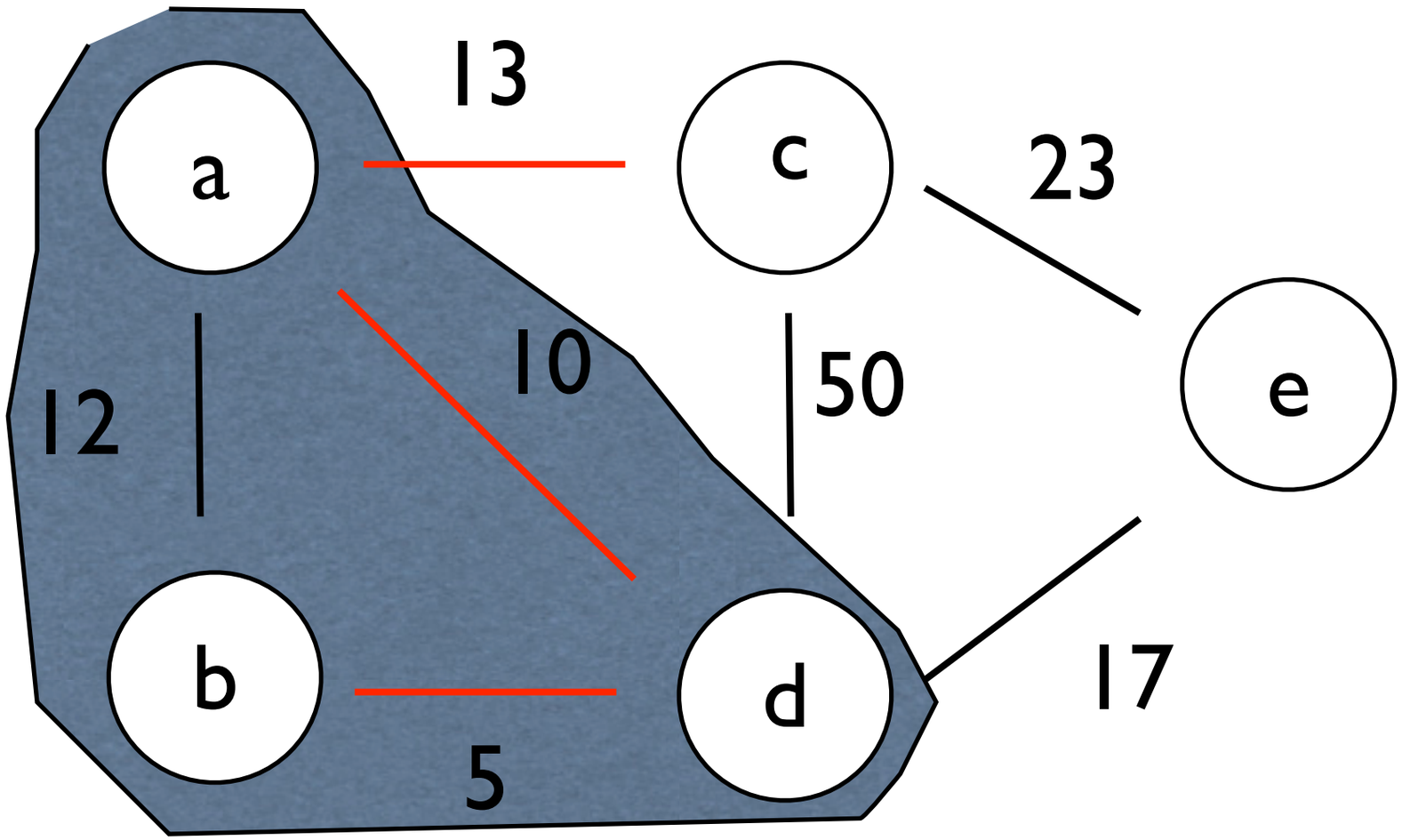
Prim



S = {a,d,b}

v	Pred(v)
a	-1
d	a
b	d

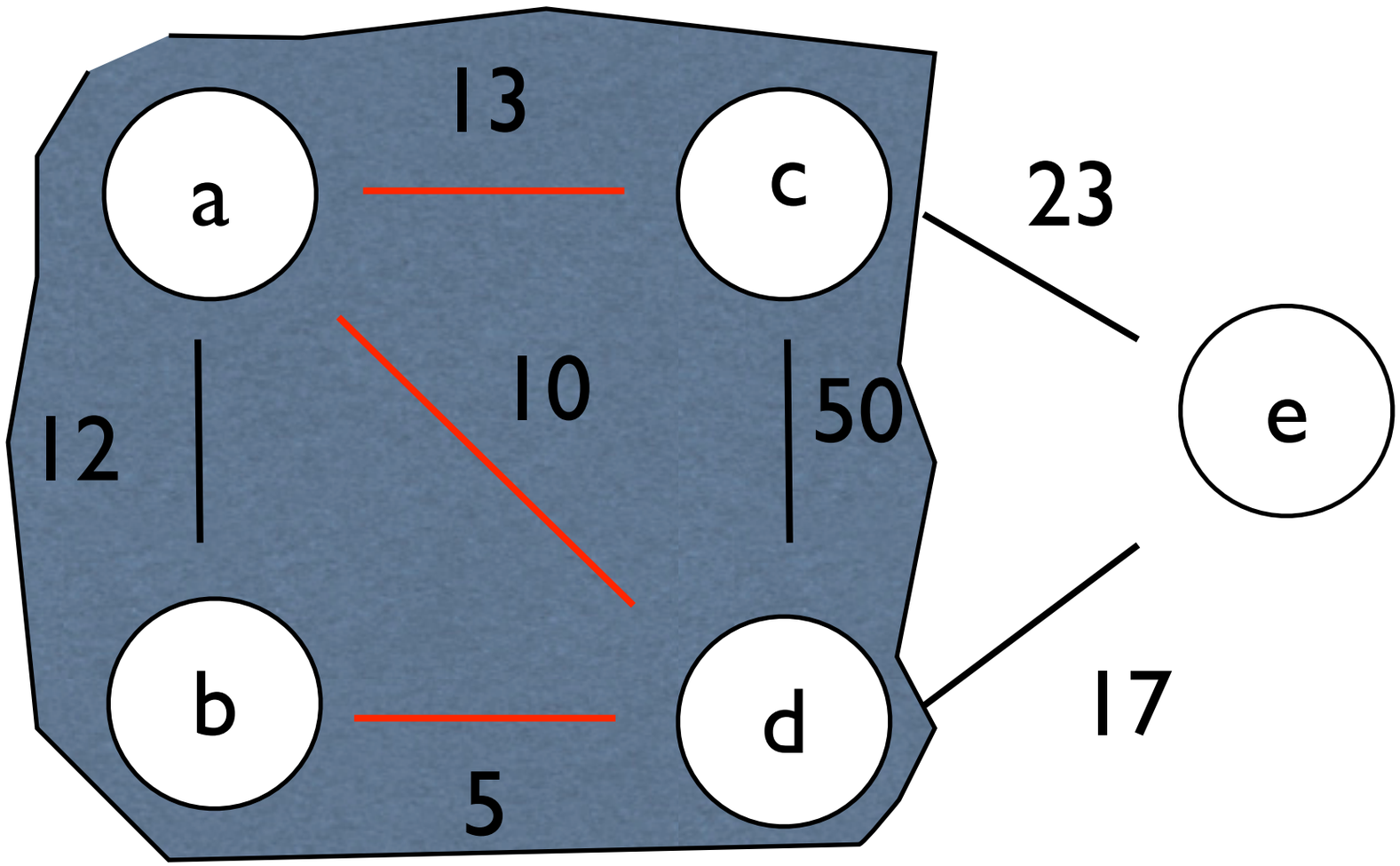
Prim



S = {a,d,b}

v	Pred(v)
a	-1
d	a
b	d

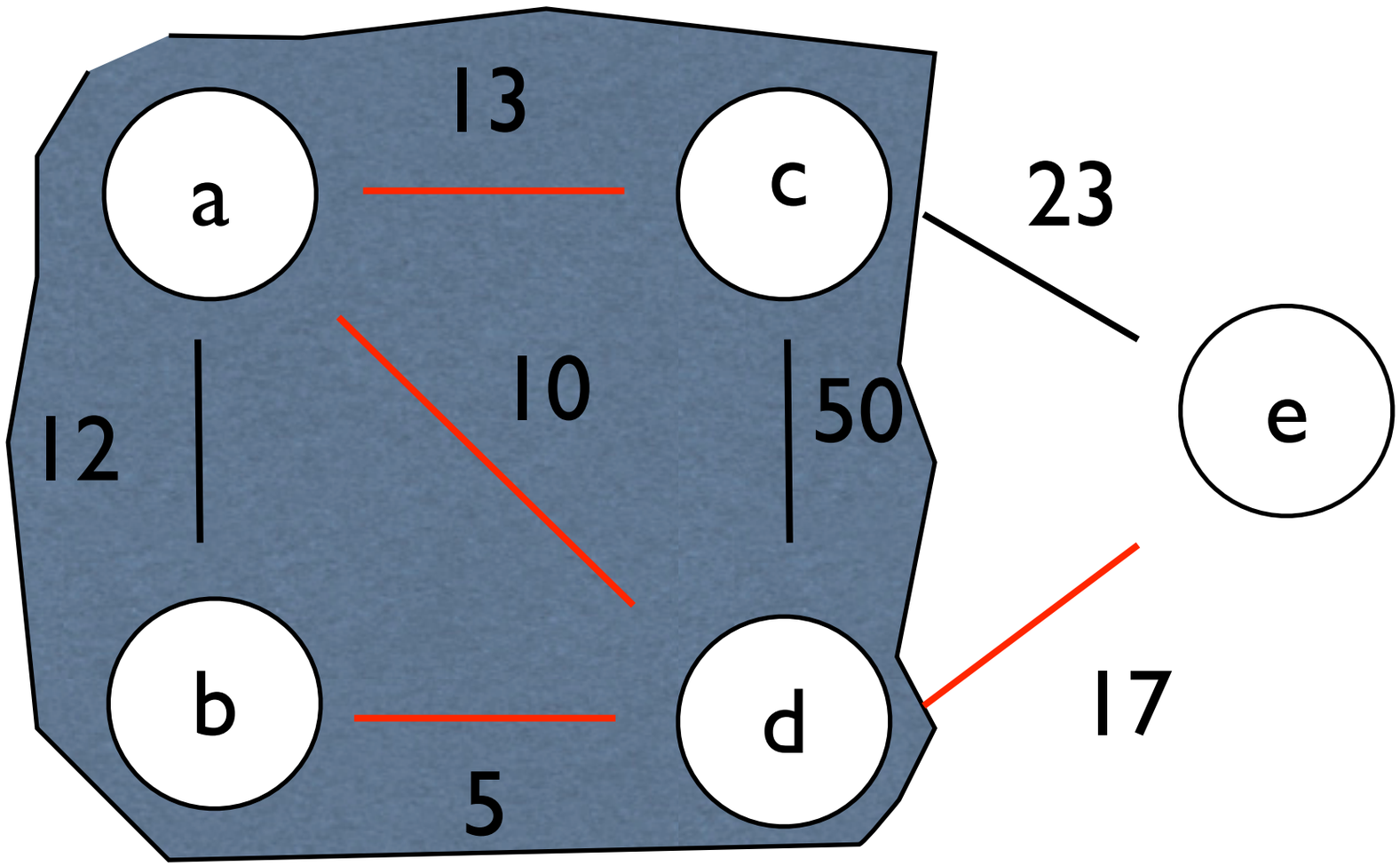
Prim



S = {a,d,b,c}

v	Pred(v)
a	-1
d	a
b	d
c	a

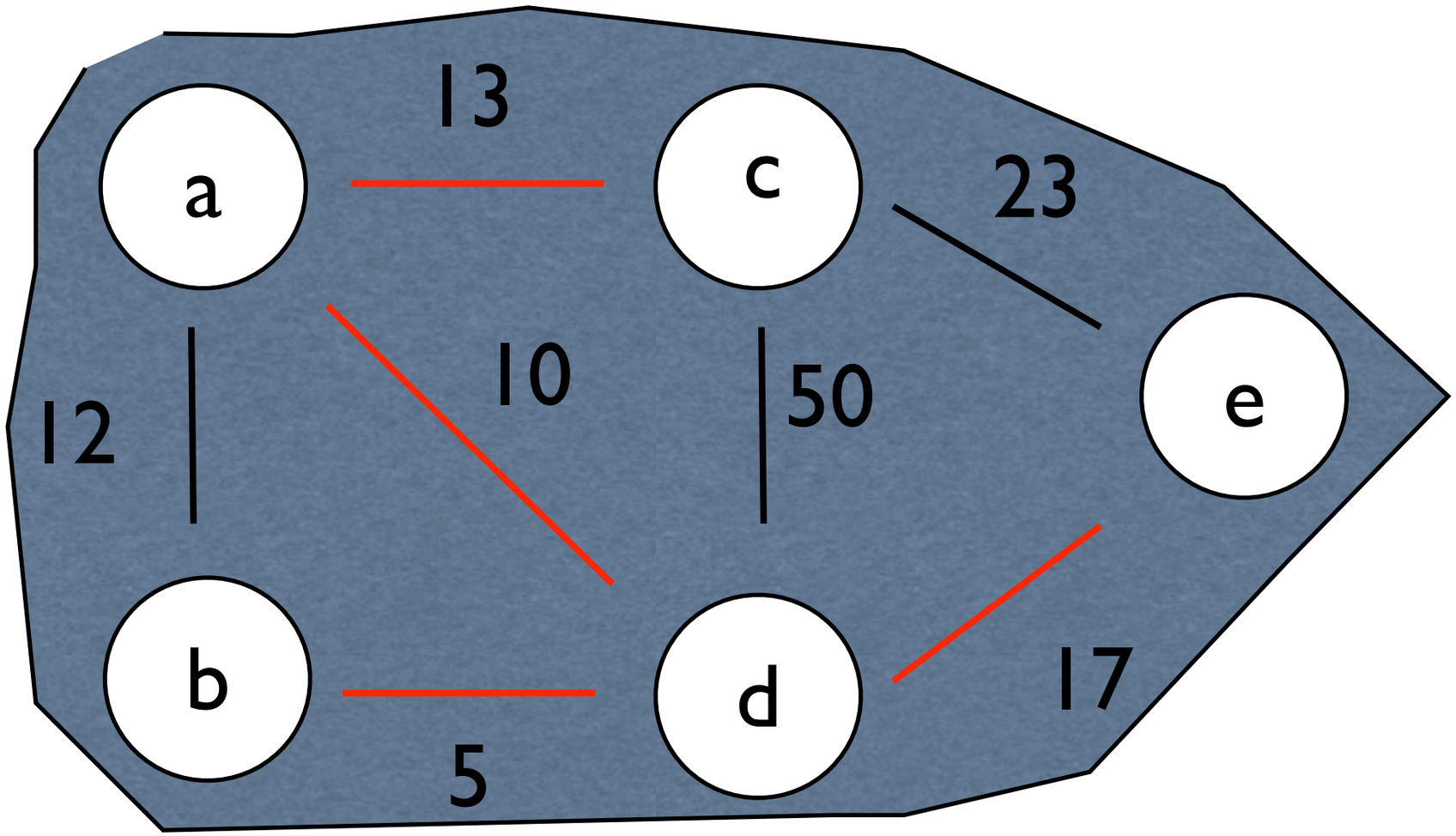
Prim



S = {a,d,b,c}

v	Pred(v)
a	-1
d	a
b	d
c	a

Prim



S = {a,d,b,c,e}

v	Pred(v)
a	-1
d	a
b	d
c	a
e	d

C(MST) = 45

Implementação

- **for** u **in** V:
  - u.key = inf
  - u.pred = nil
- r.key = 0
- Q = V
- **while not** empty(Q):
  - u = min(Q)
  - **for** v **in** vizinhos(u):
    - **If** v **in** Q **and** w(u,v) < v.key:
    - pred(v) = u
    - v.key = w(u,v)

Complexidade

- **for** u **in** V: O(?)
  - u.key = inf
  - u.pred = nil
- r.key = 0
- Q = V
- **while not** empty(Q): O(?)
  - u = min(Q) O(?) // Ao todo
  - **for** v **in** vizinhos(u): O(?) // Ao todo
    - **If** v **in** Q **and** w(u,v) < v.key:
    - pred(v) = u
    - v.key = w(u,v) O(?) // atualizar heap

O(?) // Ao todo

Complexidade

- **for** u **in** V: O(V)
  - u.key = inf
  - u.pred = nil
- r.key = 0
- Q = V
- **while not** empty(Q): O(V)
  - u = min(Q)
  - **for** v **in** vizinhos(u):
    - **If** v **in** Q **and** w(u,v) < v.key:
    - pred(v) = u
    - v.key = w(u,v)

O(VlogV) // Ao todo

O(E) // Ao todo

O(logV) // atualizar heap

O(ElogV) // Ao todo

Complexidade

- Prim
  - Com implementação min-heap eficiente
  - O(Vlog(V) + Elog(V)) = O(ElogV)

A mesma que a do Kruskal!

Complexidade

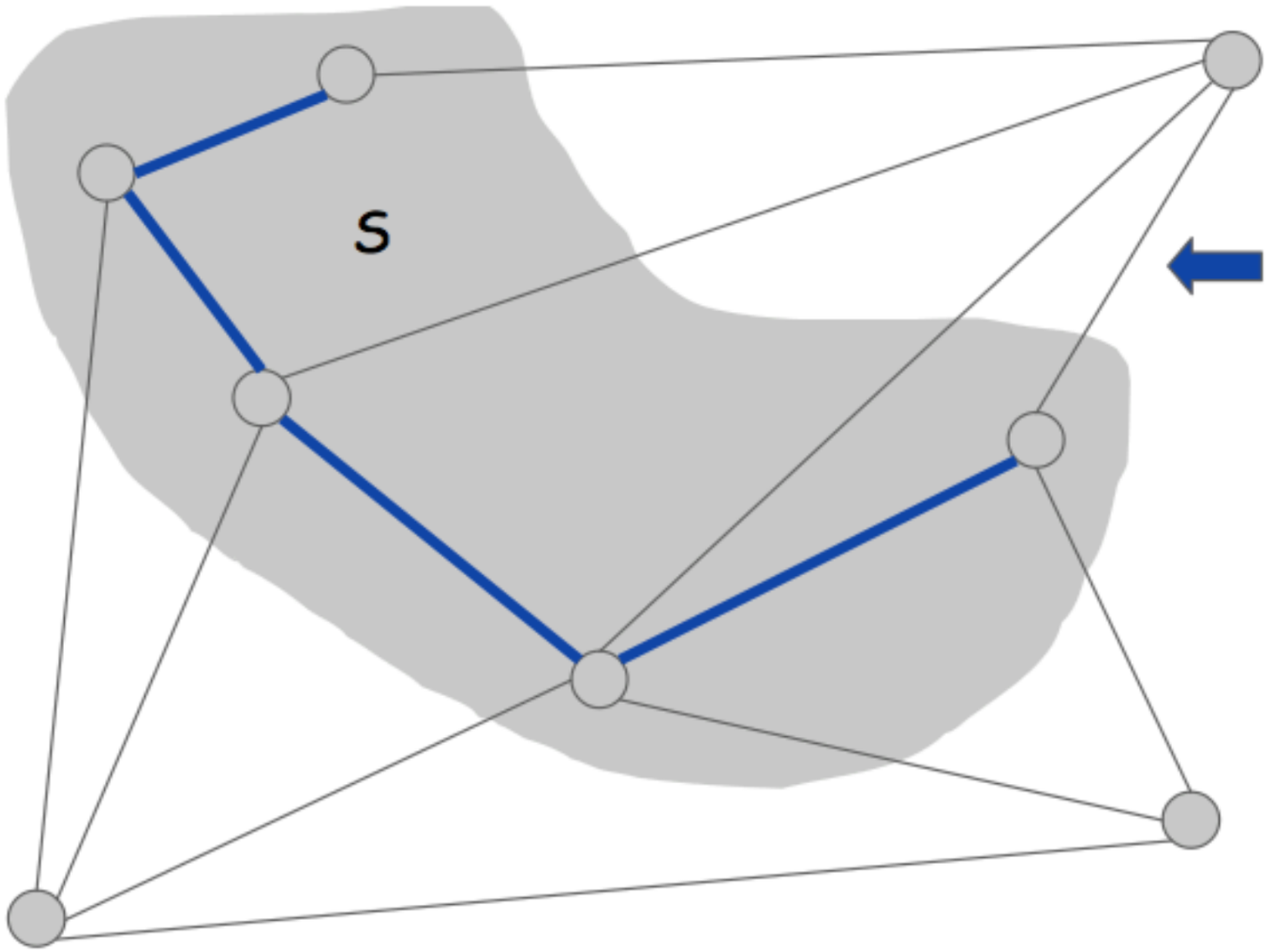
- Prim
  - Com implementação min-heap Fibonacci
    - O(E+Vlog(V))

Supera o Kruskal!

Semelhante ao Djkstra em complexidade, execução e implementação!

Corretude

- Adicione um nó v qualquer em S
- Aplicar a propriedade do corte
- Adicione uma aresta do corte, de custo mínimo, que incide em T



Corretude

Theorem 13.2 *Prim's algorithm correctly finds a minimum spanning tree of the given graph.*

Proof: We will prove correctness by induction. Let G be the given graph. Our inductive hypothesis will be that the tree T constructed so far is consistent with (is a subtree of) some minimum spanning tree M of G. This is certainly true at the start. Now, let e be the edge chosen by the algorithm. We need to argue that the new tree, $T \cup \{e\}$ is also consistent with some minimum spanning tree M' of G. If $e \in M$ then we are done ($M' = M$). Else, we argue as follows.

Consider adding e to M. As noted above, this creates a cycle. Since e has one endpoint in T and one outside T, if we trace around this cycle we must eventually get to an edge e' that goes back in to T. We know $len(e') \geq len(e)$ by definition of the algorithm. So, if we add e to M and remove e', we get a new tree M' that is no larger than M was and contains $T \cup \{e\}$, maintaining our induction and proving the theorem. ■

Evolução

- Comparativo de algoritmos determinísticos
  - O(Elog(V)) [Jarnik,Prim,Dijkstra,kruskal,Boruvka]
  - O(Elog(log(V))) [Cheriton-Tarjan 1976,Yao 1975]
  - O(Eb(E,V)) [Fredman-Tarjan 1987]
  - O(Elog(b(E,V))) [Gabow-Galil-Spencer-Tarjan 1986]
  - O(Ea(E,V)) [Chazelle 2000]
- Santo Graal O(E)
- Notáveis
  - O(E) aleatorizado [Karger-Klein-Tarjan 1995]
  - O(E) verificação [Dixon-Rauch-Tarjan 1992]

Questões

- Passo-a-passo Prim e Kruskal
- Desenvolva um algoritmo que encontre MST utilizando busca em largura
- Desenvolva um algoritmo que encontre MST por remoções de arestas