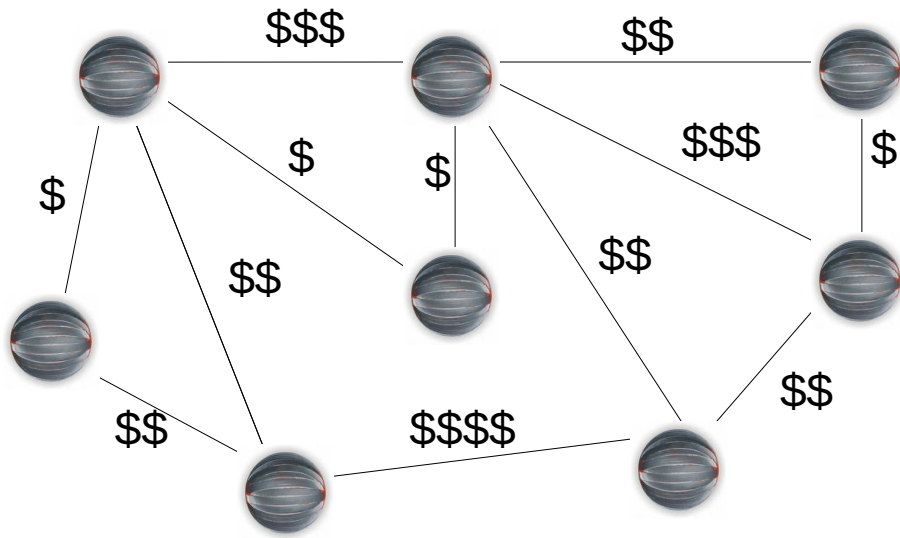


Grafos – Aula 11

Roteiro

- MST
- Algoritmo de Prim
- Algoritmo de Kruskal
- Propriedades da MST
- Corretude dos algoritmos

Projetando uma Rede



- Conjunto de localidades (ex. cidades)
- Custo para conectá-los diretamente (ex. construir estradas)

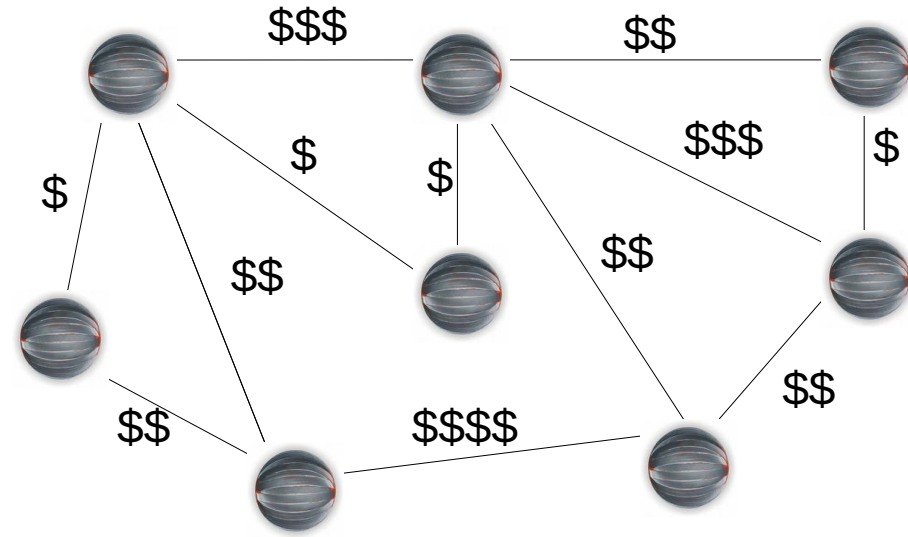
- Garantir conectividade

- de qualquer lugar, chegamos a qualquer outro

- **Problema:** Como conectar as localidades de forma a minimizar o custo total?

Projetando uma Rede

- Abstração via grafos
 - Vértices: localidades
 - Arestas com pesos: custo de conexão direta entre localidades



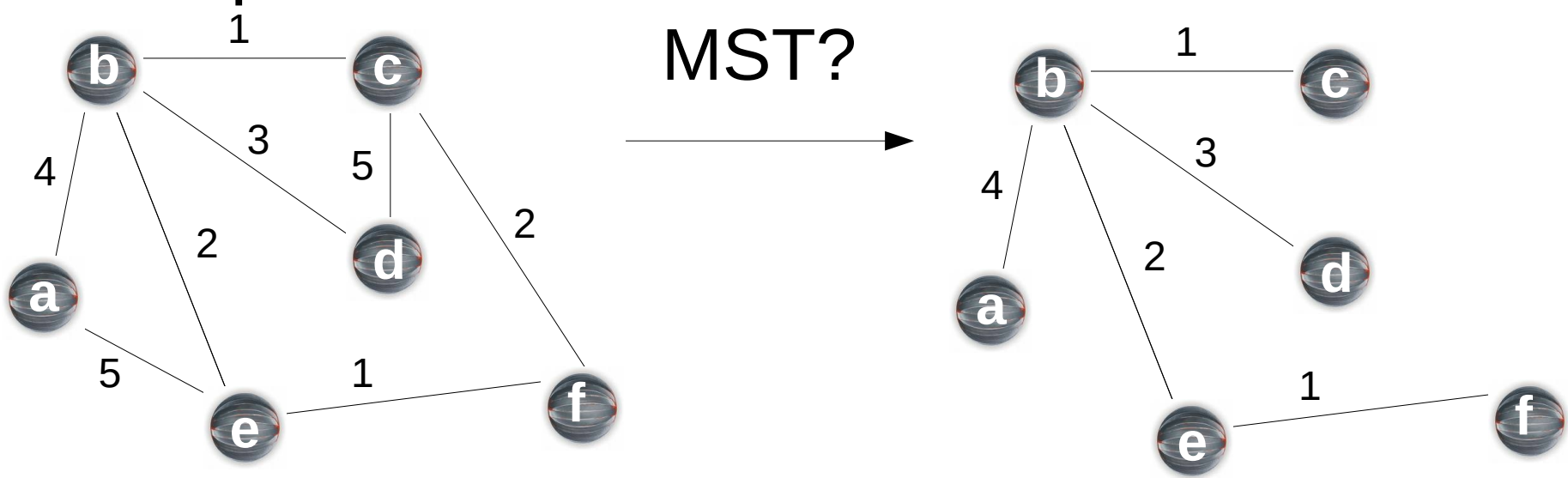
- Que tipo de rede (grafo) é o resultado?
- Subgrafo de G, uma árvore geradora
- Qualquer árvore?

Árvore Geradora de Custo Mínimo!

MST

- Minimum Spanning Tree (MST)
 - árvore geradora de custo mínimo
 - custo = soma dos pesos das arestas

Exemplo



Custo desta MST? **11**

- MST é única?

Descobrimos a MST

- **Problema:** Obter uma MST de um grafo
- Grafo com pesos idênticos? ← *BFS to the rescue!*
 - qualquer árvore geradora tem custo mínimo
- Grafo com pesos diferentes?



Idéias?

Descobrendo a MST – Idéias I

- BFS constrói uma árvore geradora
- Dado vértice inicial s , construir árvore geradora mínima
- **Ideia:** expandir fronteira na direção correta

Qual é a direção correta?

- Direção de menor custo do ponto de vista da árvore

Descobrendo a MST – Idéias I

- Dado $G=(V, E)$, construir MST, $T=(S, E')$
- Inicialmente S e E' estão vazios
- Selecionar s qualquer, vértice inicial
- Adicionar vértices em T na ordem mais barata possível
 - próximo vértice aumenta custo da árvore o mínimo possível

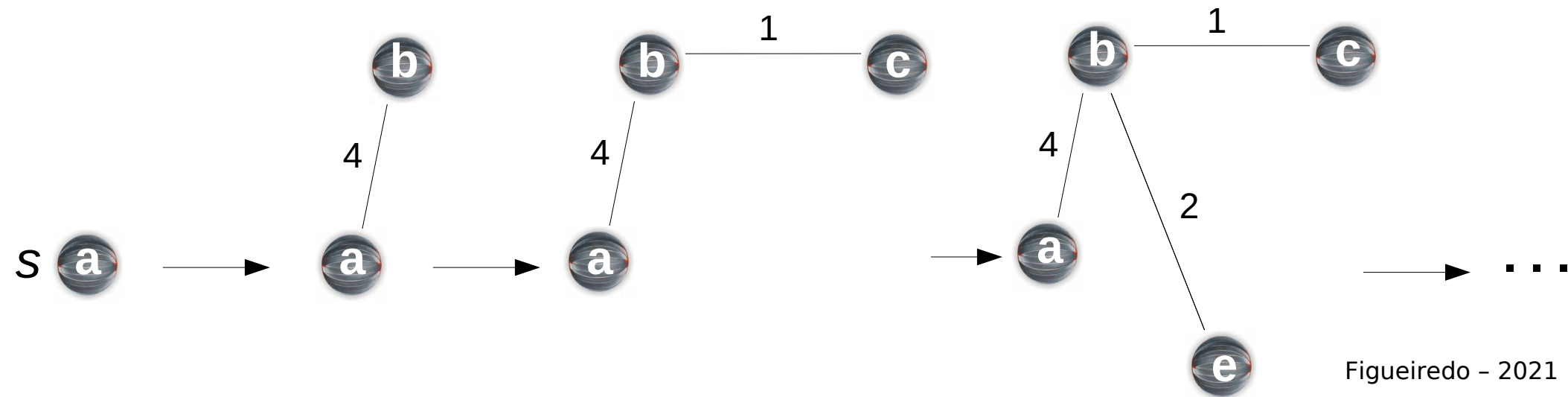
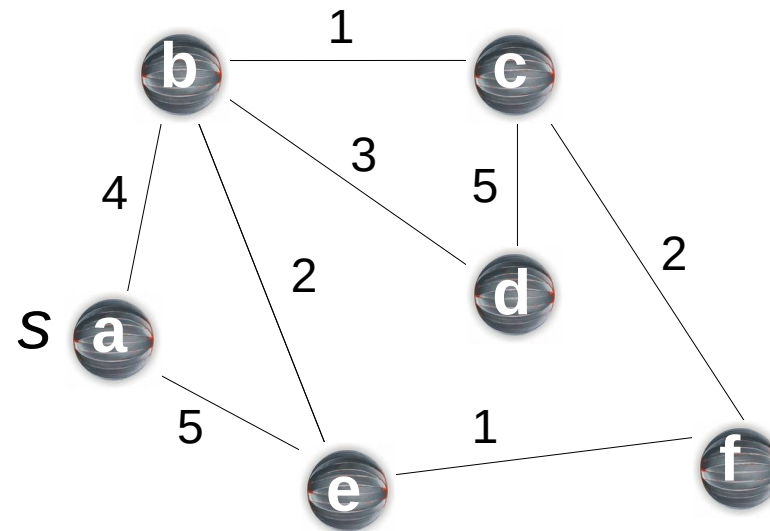
Algoritmo de *Prim*

- Muito parecido com algoritmo de Dijkstra

Algoritmo de *Prim*

■ **Idéia:** crescer T de forma mais barata possível

■ **Exemplo**



Algoritmo de *Prim*

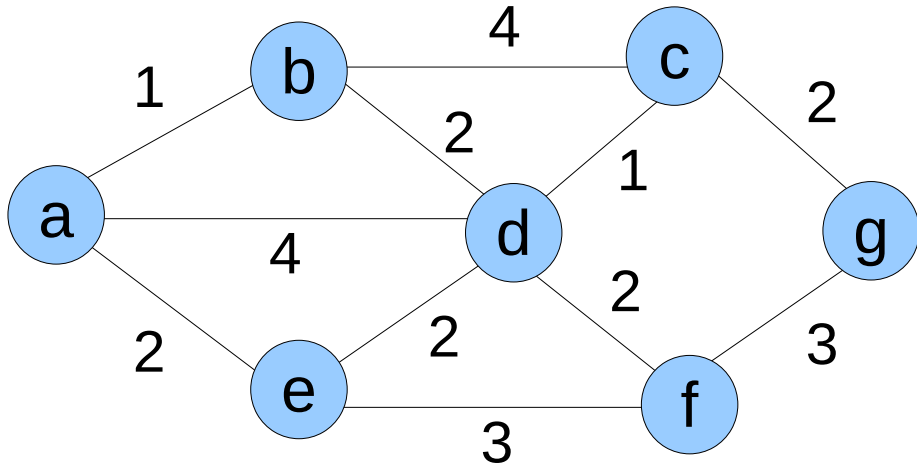
- Como tornar a idéia em algoritmo (eficiente)?
 - adicionar o vértice que aumenta o custo da árvore o menos possível
- **Idéias:**
 - Manter um conjunto de vértices da árvore (vértices explorados)
 - Manter custo para adicionar cada vértice até o momento (vértices descobertos)
 - Adicionar o vértice de menor custo
 - Atualizar custos

Algoritmo de *Prim*

1. Prim(G, o)
2. Para cada vértice v
3. custo[v] = infinito
4. Define conjunto $S = \emptyset$ // vazio
5. custo[o] = 0
6. Enquanto $S \neq V$
7. Selecione u em $V - S$, tal que custo[u] é mínimo
8. Adicione u em S
9. Para cada vizinho v de u faça
10. Se custo[v] > $w(u, v)$ então
11. custo[v] = $w(u, v)$

↖
Custo do vértice depende apenas do peso da aresta incidente a ele

Executando o Algoritmo



- Manter tabela com passos e custos

$c(u)$ é igual ao custo $[u]$ no algoritmo

Passo	Conjunto S	c(a)	c(b)	c(c)	c(d)	c(e)	c(f)	c(g)
0	{}	0	inf	inf	inf	inf	inf	inf
1	{a}	-	1	inf	4	2	inf	inf
2	{a,b}		-	4	2	2	inf	inf
3	{a,b,e}			4	2	-	3	inf
4	{a,b,e,d}			1	-		2	inf
5	{a,b,e,d,c}			-			2	2
6	{a,b,e,d,c,f}						-	2
7	{a,b,e,d,c,f,g}							-

Complexidade

- Mesmo funcionamento que algoritmo de Dijkstra
 - mesma complexidade

```
1. Prim(G, o)
2. Para cada vértice v
3.     custo[v] = infinito
4. Define conjunto S = ∅ // vazio
5. custo[o] = 0
6. Enquanto S ≠ V
7.     Selecione u em V-S, tal que custo[u] é mínimo
8.     Adicione u em S
9.     Para cada vizinho v de u faça
10.        Se custo[v] > w(u,v) então
11.            custo[v] = w(u,v)
```

- Usando filas de prioridade baseada em heap
 - n operações de remoção, m de atualização
- $O((m+n)\log n) = O(m \log n)$

Descobrimos a MST – Idéias II

- Outra abordagem, diferente de *BFS*
 - mas também gulosa
- Observações
 - aresta de menor peso sempre está na MST
 - aresta de segundo menor peso sempre está na MST
 - aresta de terceiro menor *pode* estar na MST
 - vai estar se não formar um ciclo

Cuidado para não formar ciclos!

Descobrendo a MST – Idéias II

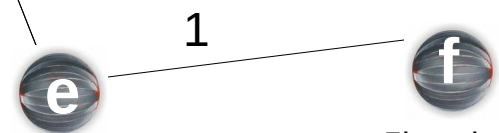
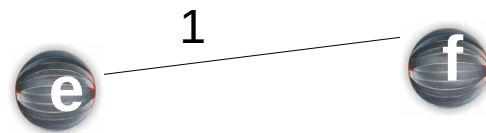
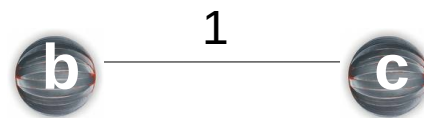
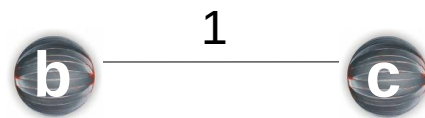
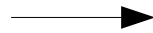
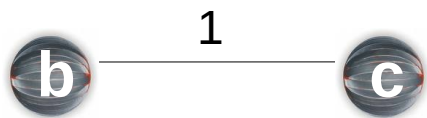
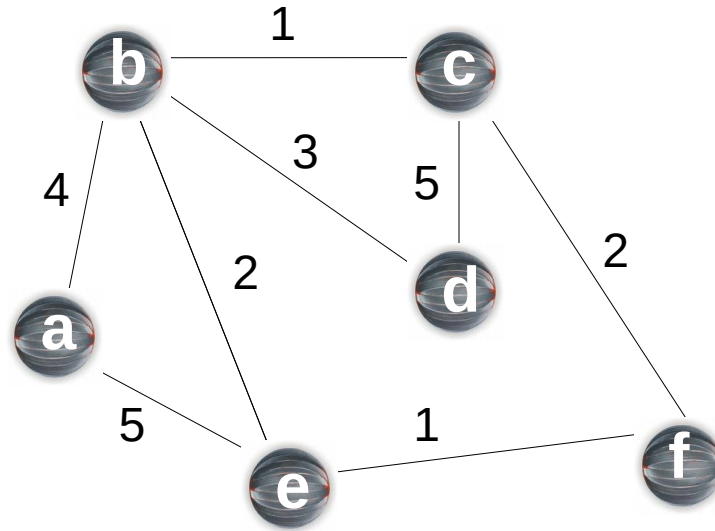
- Dado $G=(V, E)$, construir MST, $T=(V, E')$
- Ordenar arestas por peso
- Inicialmente E' está vazio
- Adicionar arestas em ordem crescente de peso
- Se aresta formar um ciclo em T , então descarte e continue

Algoritmo de *Kruskal*

Algoritmo de *Kruskal*

■ **Idéia:** construir T adicionado arestas de menor peso

■ **Exemplo**

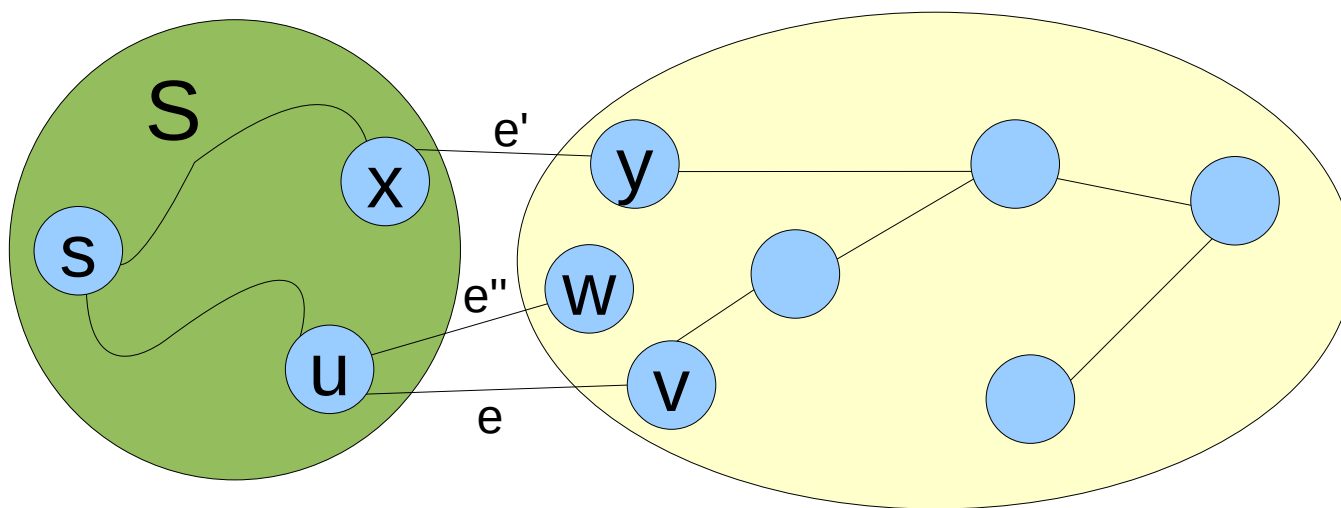


Analizando o Algoritmo

- Algoritmos de *Prim* e *Kruskal* sempre retornam uma MST
 - mas isto é óbvio?
- Como provar que algoritmo sempre produz resultado desejado - uma MST?
- Duas propriedades de uma MST
 - Propriedade do Corte (*cut property*)
 - Propriedade do Ciclo (*cycle property*)

Propriedade do Corte

- Considere um conjunto de vértices S e a aresta $e=(u, v)$ de menor peso com uma ponta em S e outra em $V-S$. Então **toda MST contém $e=(u,v)$** .
- O que isto está dizendo?



Se $c(e) < c(e'), c(e'')$
Então e está em toda MST

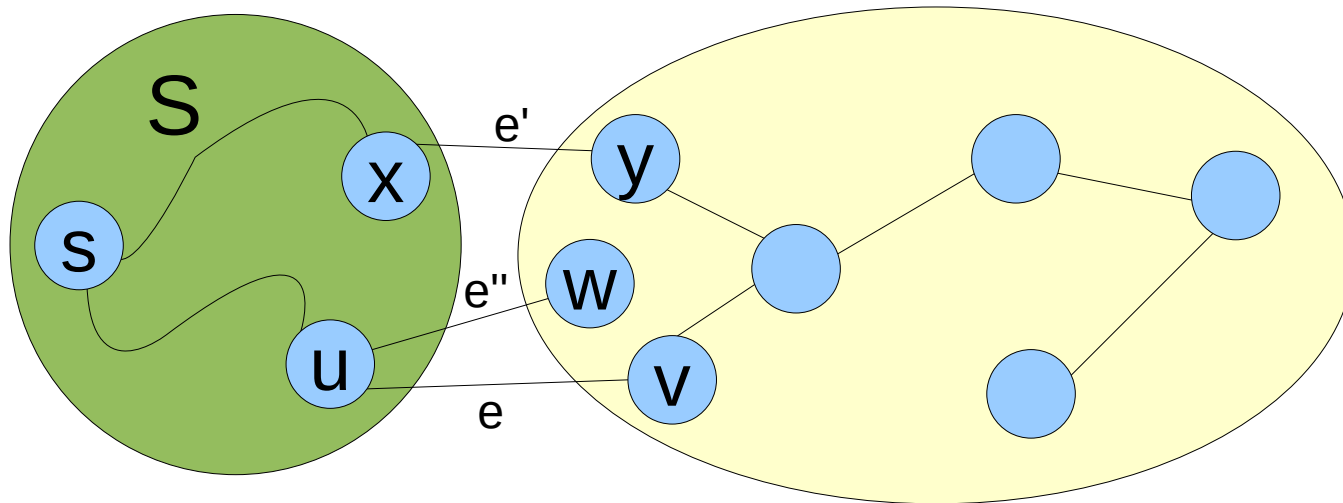
■ Como provar isto?

← Por contradição

Propriedade do Corte

- **Prova por contradição:** assumir que o oposto é verdade, e mostrar que nova afirmação não é verdadeira
- **Oposto:** Existe MST que não possui aresta e
- Como provar que isto não é verdade?
 - 1) T' possui aresta e' e algum peso total
 - 2) Mostrar que e pode substituir e' em T'
 - 3) Peso da árvore com e é menor, logo T' com e' não é mínima, e não pode ser MST

Propriedade do Corte



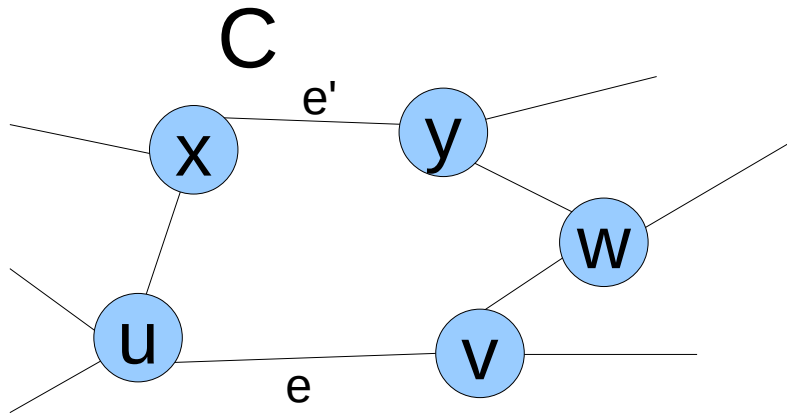
- Assumir árvore T' sem e
- Existe caminho P em T' entre u e v
- Seguir caminho P até encontrar vértice x em S e y em $V-S$, chamar $e'=(x,y)$
- Trocar e' por e cria nova árvore geradora T
- Nova T tem peso menor que T'
- Logo T' não é MST

Corretude de *Prim*

- Algoritmo de Prim
 - a cada passo, adiciona aresta de menor peso entre S e $V-S$ (corte)
- Pela Propriedade do Corte, aresta faz parte de qualquer MST
- Prim só inclui arestas em T que fazem parte de qualquer MST
- Prim gera uma MST!

Propriedade do Ciclo

- Seja C um ciclo em G e $e=(u,v)$ a aresta de maior custo de C . Então $e=(u,v)$ **não** faz parte de nenhuma MST.
- O que isto está dizendo?



- Considere ciclo $C = x,y,w,v,u,x$
- Considere aresta $e=(u,v)$ de maior peso neste ciclo
- Então aresta e não está em nenhuma MST

- Como mostrar isto é verdade?
 - por contradição

Corretude de *Kruskal*

- Algoritmo de Kruskal
 - processa arestas em ordem crescente de peso, não inclui aresta que fecha ciclo
- Pela Propriedade do Ciclo, aresta deixada de fora não pertence a nenhuma MST
- Kruskal não inclui arestas que não pertencem a nenhuma MST
- Kruskal gera uma MST!