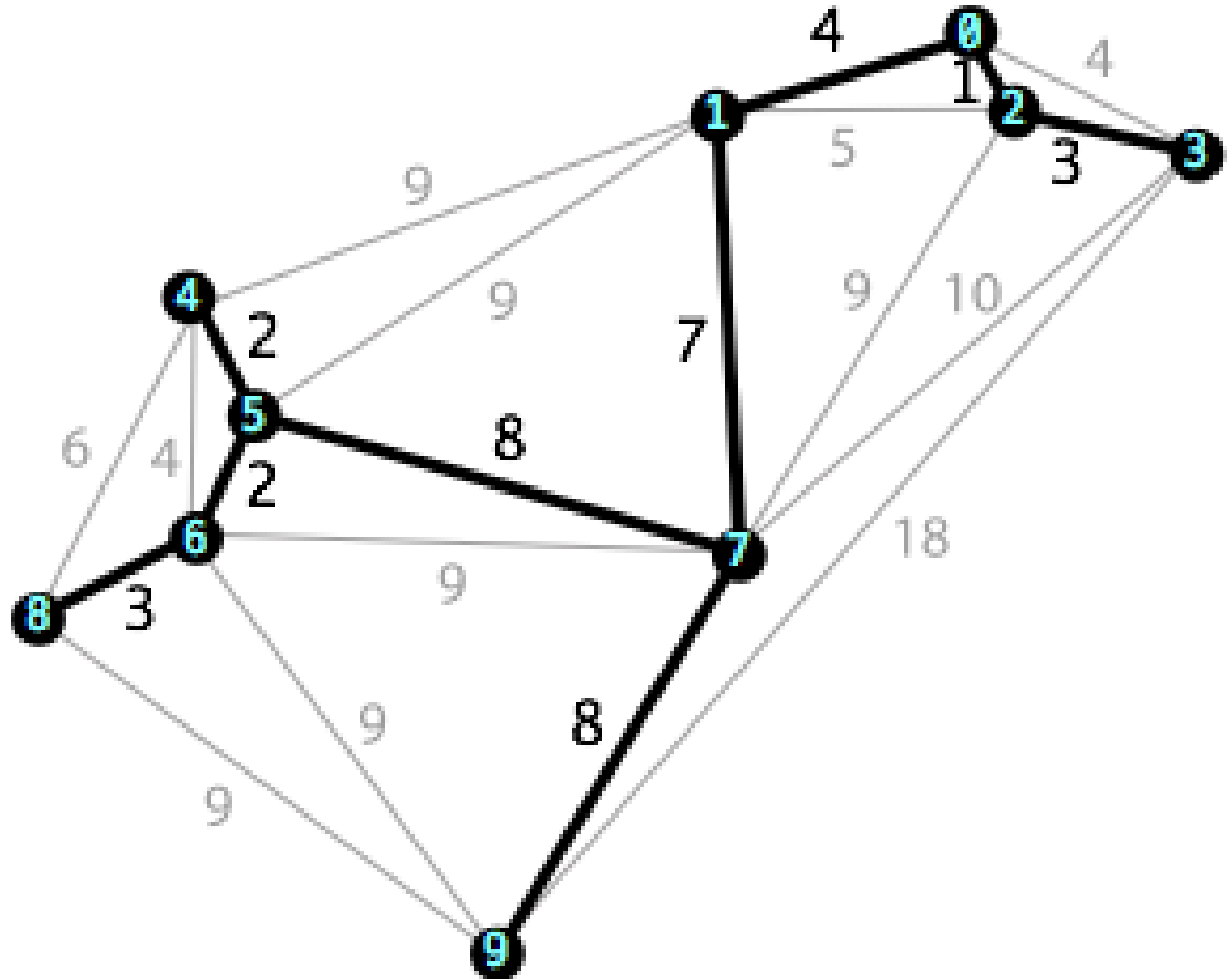


TEORIA DOS GRAFOS

ÁRVORE GERADORA
MÍNIMA (MINIMUM
SPANNING TREE - MST)

PROF. ANDRÉ
KAWAMOTO



AGENDA

- Motivação
- O que é uma Árvore Geradora Mínima?
- Algoritmos para MST
 - Prim
 - Kruskal
 - Complexidade dos Algoritmos

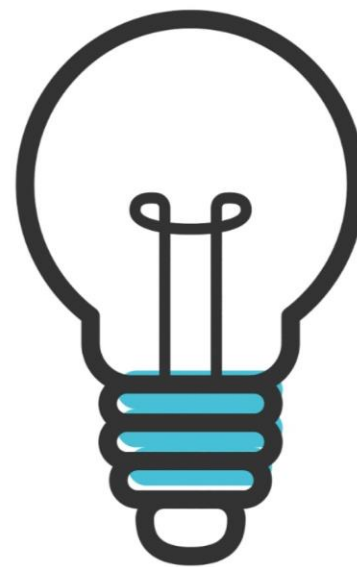
MOTIVAÇÃO

- Considere as situações
 - Em um circuito, vários pinos devem ser eletricamente **carregados ao mesmo tempo**. Preciso conectar esses pinos usando o mínimo de fiação.
 - Em um edifício, desejo interconectar vários pontos numa mesma rede (elétrica, hidráulica ou de dados, por exemplo), usando a menor quantidade de material
 - Em um roteiro turístico numa mata densa preciso de trilhas para que as pessoas possam passar por vários pontos de interesse derrubando menos árvores.
 - Como fazer isso?



UMA SOLUÇÃO

- Represente a situação desejada como um grafo
- Aplique um algoritmo de **Árvore Geradora Mínima**
- Outros nomes:
 - Minimum spanning Tree (MST)
 - Árvore Geradora de Custo Mínimo
 - Árvore de Extensão Mínima




O QUE É UMA MST?

Uma **Árvore** é um grafo conexo e sem ciclos



Em um grafo qualquer, uma **Árvore Geradora** é um conjunto de arestas que resulta em uma Árvore



Um mesmo grafo pode possuir várias Árvore Geradoras



Uma **Árvore Geradora Mínima** é aquela cuja soma dos pesos nas arestas é o menor possível.

ALGORITMOS PARA GERAR MST

- Existem dois algoritmos bastante conhecidos para gerar MSTs
 - Kruskal
 - Prim
- Ambos são Algoritmos Gulosos (*Greedy*)*
 - Kruskal – adição de arestas
 - Prim – adição de vértices

*Algoritmos Gulosos tentam resolver um problema fazendo a escolha localmente ótima **em cada fase** na tentativa de encontrar uma solução ótima global.

ALGORITMO DE KRUSKAL PARA MST

ADIÇÃO DE ARESTAS

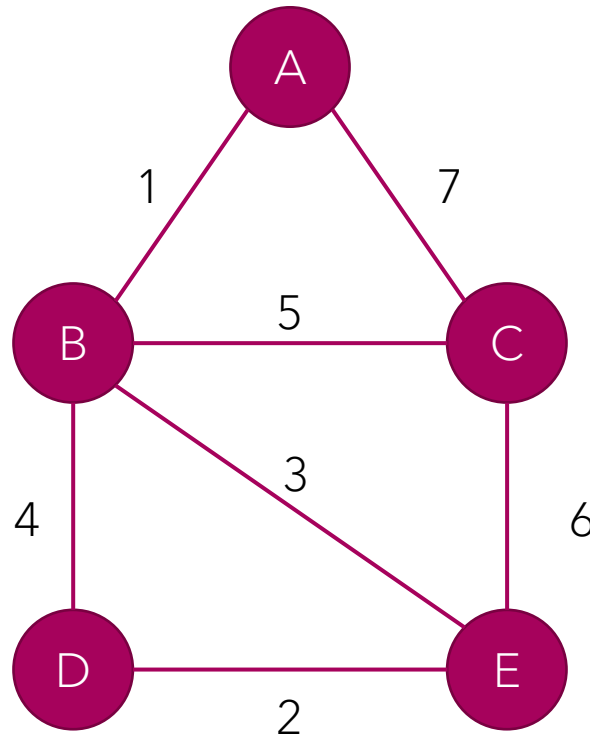
ALGORITMO DE KRUSKAL

- Funciona em grafos não-direcionados e conexos
- Uma MST terá $(V-1)$ arestas, sendo V o número de vértices do grafo
- Para reflexão
 - Por que $V-1$ arestas?
 - Se o grafo não for conexo, faz sentido pensar em uma MST?

ALGORITMO DE KRUSKAL

1. Crie um conjunto A , inicialmente vazio
2. Considere cada vértice de V uma única árvore (Make-Set)
3. Ordene as arestas do grafo em ordem crescente
4. Para cada aresta $(u-v)$, em ordem crescente, até atingir $(V-1)$ arestas:
 1. Se u e v não pertencem à mesma árvore ($\text{Find-}u \neq \text{Find-}v$)
 2. Adicione a aresta $(u-v)$ ao conjunto A , combinando as duas árvores em uma só
5. Retorna A

EXEMPLO



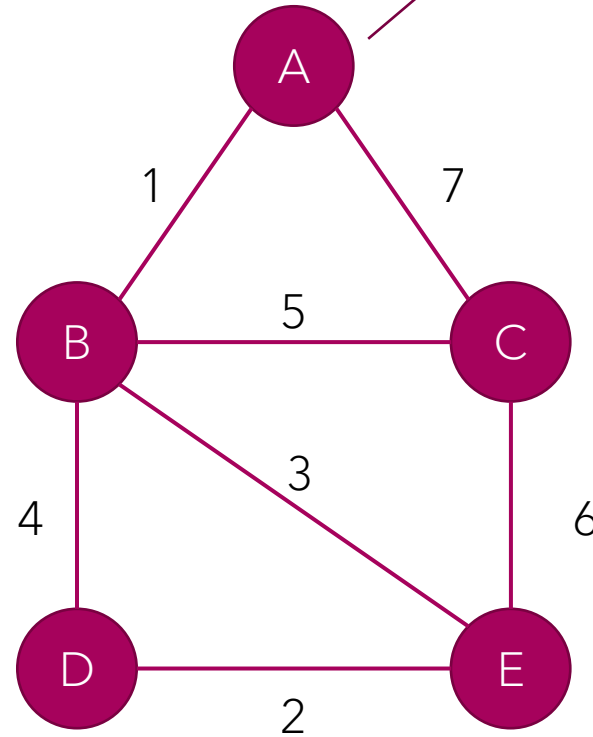
EXEMPLO

Origem	Destino	Peso
A	B	1
D	E	2
B	E	3
B	D	4
B	C	5
C	E	6
A	C	7

$A = \{\}$

Arestas em ordem
crescente de valor

Cada vértice do grafo é
considerado uma árvore

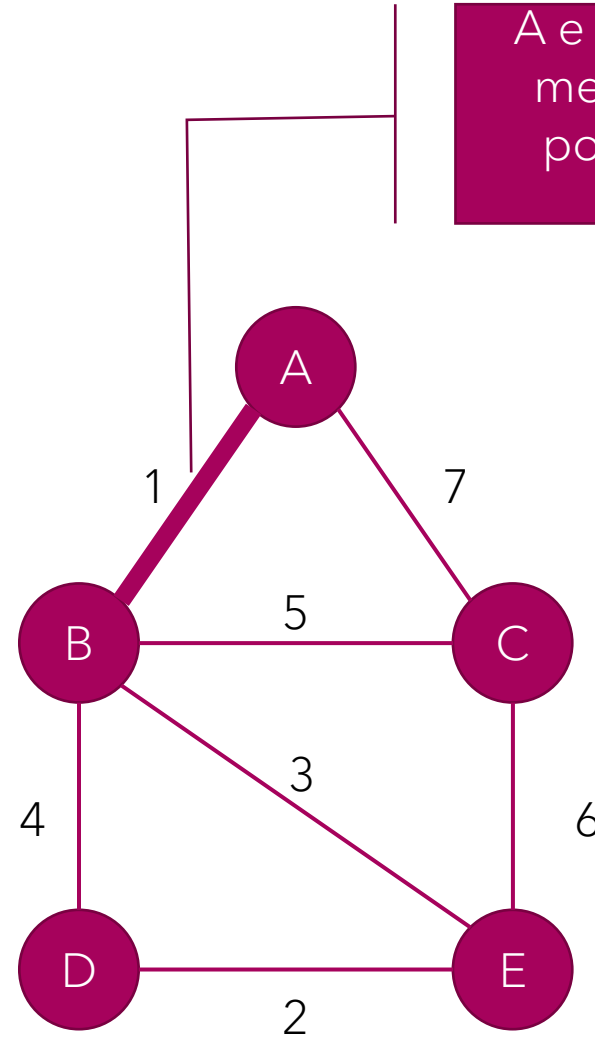


EXEMPLO

Origem	Destino	Peso
A	B	1
D	E	2
B	E	3
B	D	4
B	C	5
C	E	6
A	C	7

Aresta de menor peso

$$A = \{(a,b)\}$$

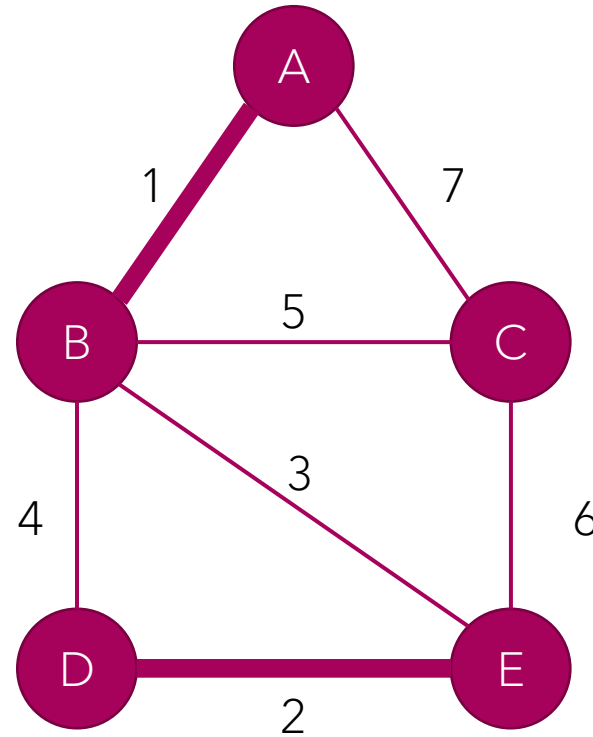


A e B não pertencem à mesma árvore, então pode adicionar essa aresta

EXEMPLO



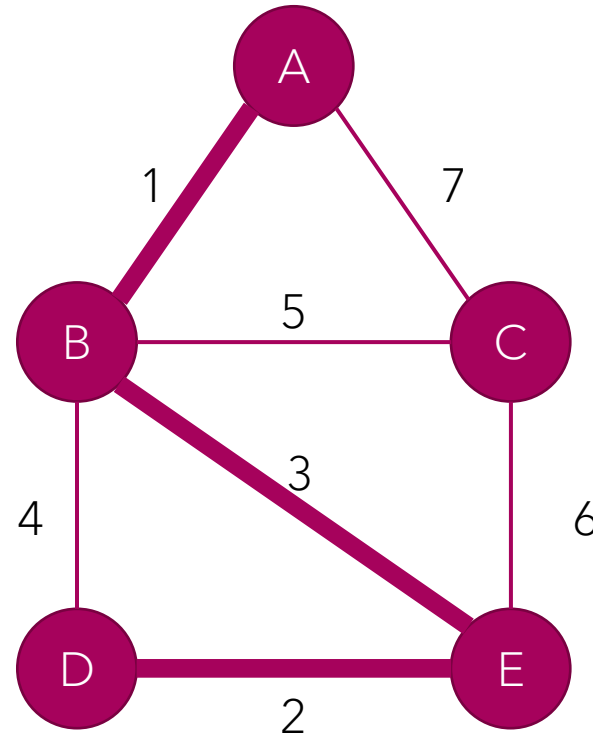
Origem	Destino	Peso
A	B	1
D	E	2
B	E	3
B	D	4
B	C	5
C	E	6
A	C	7



$$A = \{(a,b), (d,e)\}$$

EXEMPLO

Origem	Destino	Peso
A	B	1
D	E	2
→ B	E	3
B	D	4
B	C	5
C	E	6
A	C	7

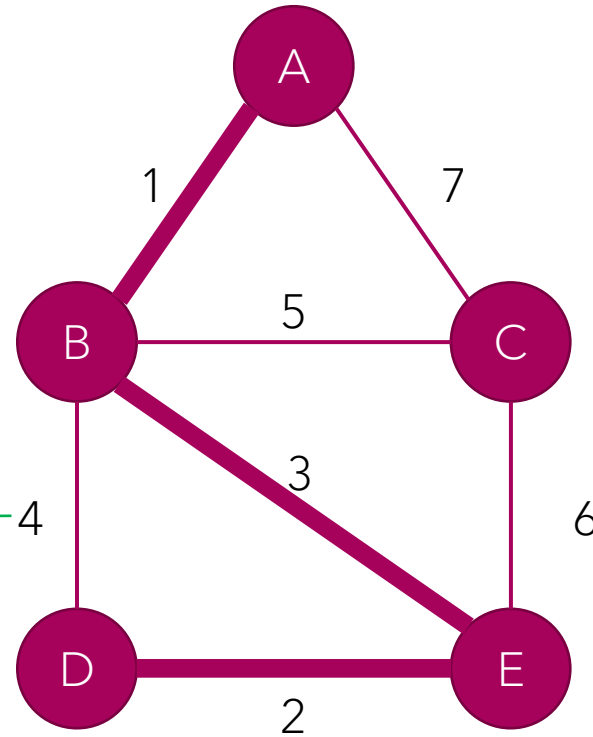


$$A = \{(a,b), (d,e), (b,e)\}$$

EXEMPLO

Origem	Destino	Peso
A	B	1
D	E	2
B	E	3
B	D	4
B	C	5
C	E	6
A	C	7

B e D pertencem à mesma árvore,
logo não podemos inserir essa aresta



$$A = \{(a,b), (d,e), (b,e)\}$$

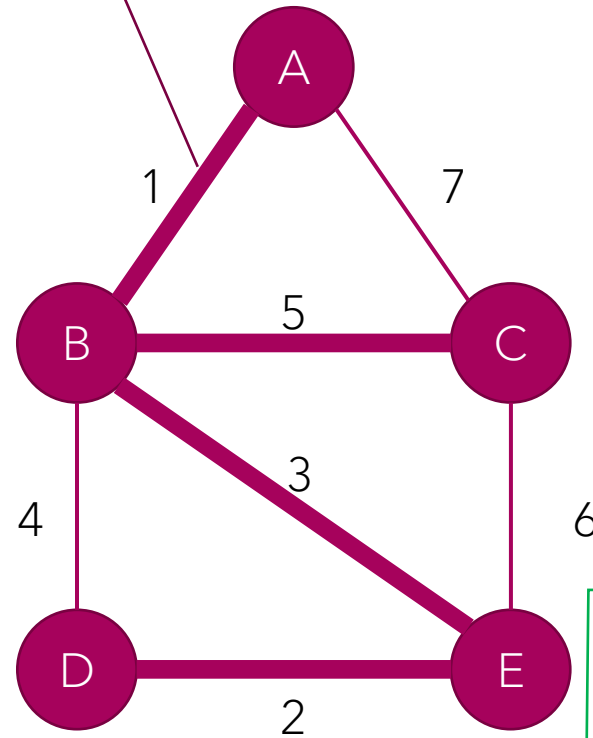
EXEMPLO

Origem	Destino	Peso
A	B	1
D	E	2
B	E	3
B	D	4
B	C	5
C	E	6
A	C	7



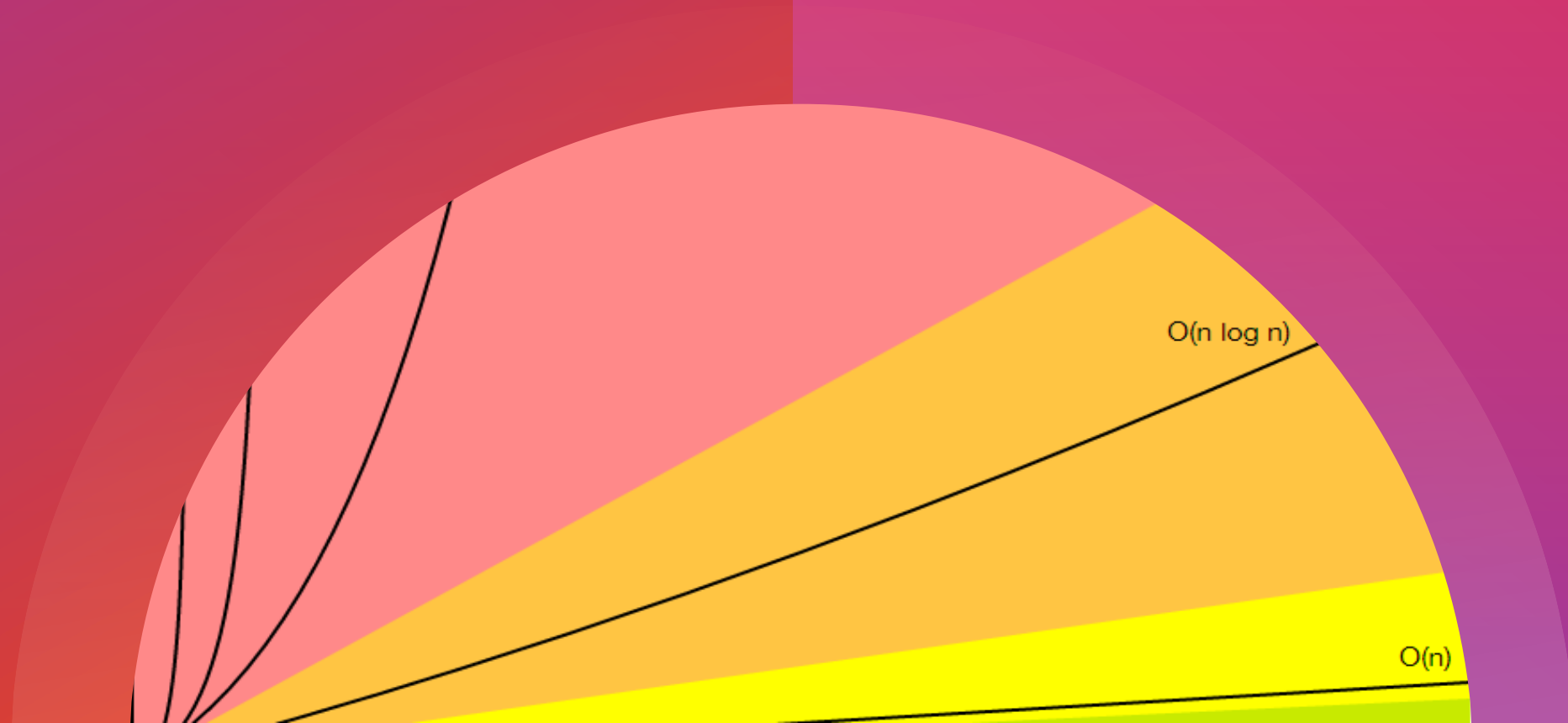
$A = \{(a,b), (d,e), (b,e), (b,c)\}$

Essa é a Árvore Geradora Mínima



Temos V-1 arestas em A.
Fim

COMPLEXIDADE DO ALGORITMO DE KRUSKAL



COMPLEXIDADE DO ALGORITMO DE KRUSKAL

1. Ordenar as arestas pode ser feito em $O(E \log E)$ (usando quicksort, p. ex.)
2. Após a ordenar, iteramos pelas as arestas (E) e aplicamos um algoritmo* para verificar se suas extremidades pertencem à mesma árvore.
 - A complexidade desse algoritmo é de no máximo $O(\log V)$
 - Logo, temos $O(E \log V)$ para todas as arestas
3. A complexidade geral é: $O(E \log E + E \log V)$, ou seja, o tempo para ordenar as arestas + tempo de aplicar a iteração pelas arestas.
4. Em grafos densos, o valor de E pode chegar a V^2 , então $O(\log E)$ e $O(\log V)$ são praticamente a mesma grandeza.

Assim, a Complexidade Geral desse algoritmo é $O(E \log E)$ ou $O(V \log V)$



ALGORITMO DE PRIM PARA MST

ALGORITMO DE PRIM

- Funciona em grafos não-direcionados e conexos
- Inicia em um vértice aleatório qualquer
- Em vez de adicionar arestas, adiciona vértices
- Abordagem Gulosa

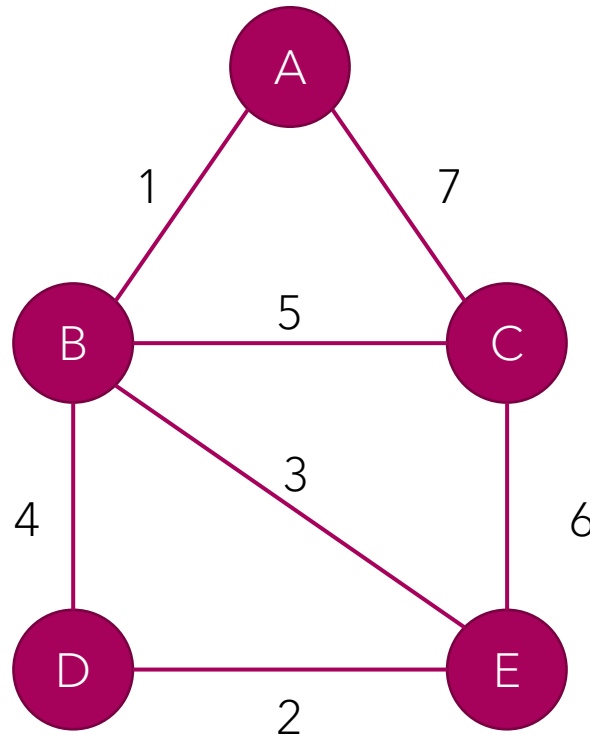
ALGORITMO DE PRIM

1. Crie um conjunto **mstSet** para armazenar os vértices da MST, inicialmente vazio
2. Atribua um valor de custo de entrada paraa todos os vértices no grafo. Esse valor inicial é **INFINITO**
3. Escolha um vértice inicial qualquer
4. Atribua o valor de custo 0 (zero) para o vértice inicial
5. Enquanto o mstSet não incluiu todos os vértices
 - a) Escolha um vértice u , adjacente à árvore, que não esteja no mstSet e tenha o menor valor de custo de entrada
 - b) Inclua u no mstSet.
 - c) Atualize o valor do custo de entrada para todos os vértices adjacentes a u .

*Para atualizar esses valores, itere através de todos os vértices adjacentes.

Para cada vértice adjacente v , se o custo de entrada da aresta $u-v$ for menor que o valor anterior de v , atualize esse valor com o peso da aresta $u-v$

EXEMPLO



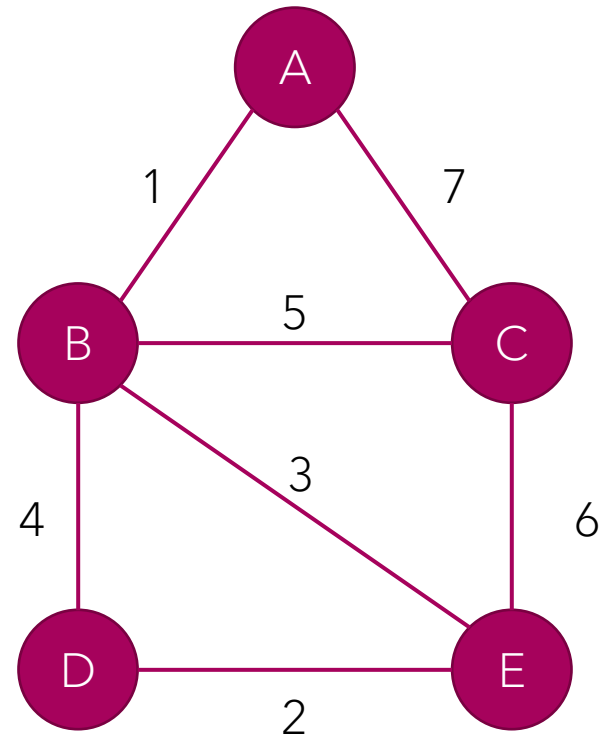
EXEMPLO

MST = {}

Custos

A	B	C	D	E
∞	∞	∞	∞	∞

V:



Inicialização

EXEMPLO

MST = {}

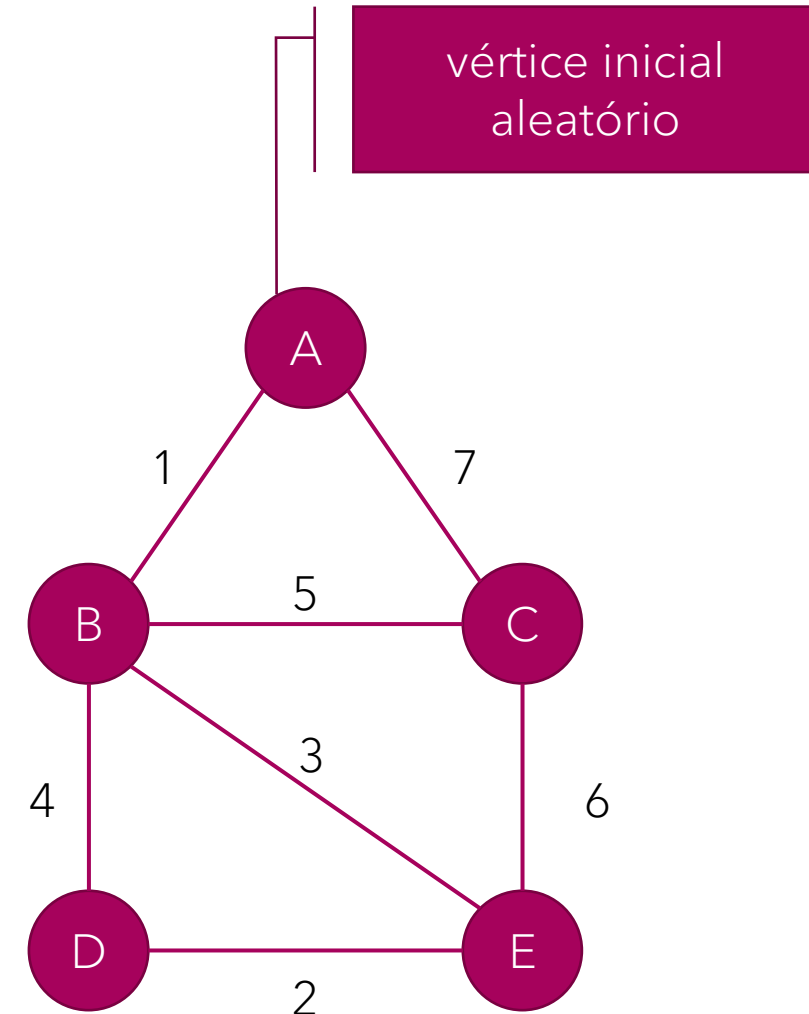
Custos

A	B	C	D	E
0	∞	∞	∞	∞

Custo do
vértice inicial

U:

Inicialização



EXEMPLO

MST = {A}

Custos

A	B	C	D	E
0	1	7	∞	∞

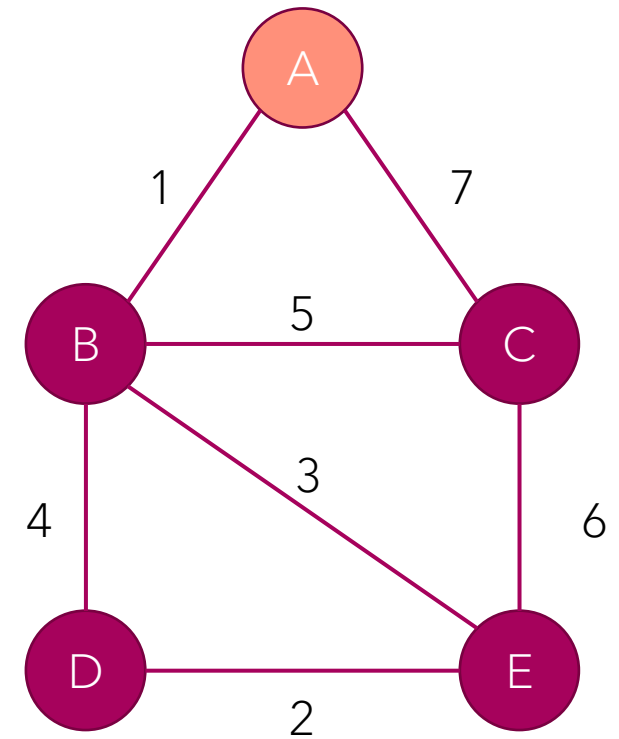
u: A

(2) Inseriu no MST

(1) Vértice de menor custo

(3) Atualiza custos

Iteração



EXEMPLO

MST = {A,B}

Custos

A	B	C	D	E
0	1	5	4	3

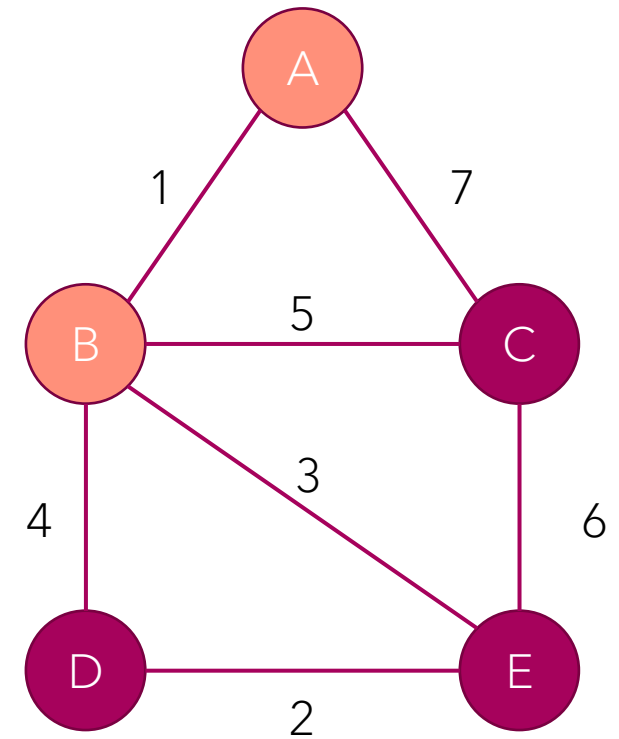
u: B

(2) Inseriu no MST

(1) Vértice de menor custo

(3) Atualiza custos

Iteração



EXEMPLO

MST = {A,B, E}

Custos

A	B	C	D	E
0	1	5	2	3

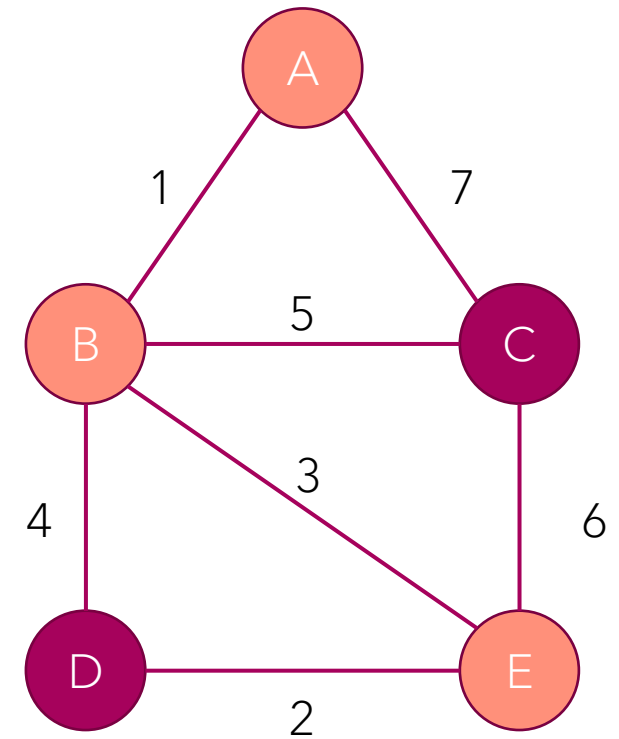
u: E

(2) Inseriu no MST

(1) Vértice de menor custo

(3) Atualiza custos

Iteração



EXEMPLO

MST = {A,B, E, D}

Custos

A	B	C	D	E
0	1	5	2	3

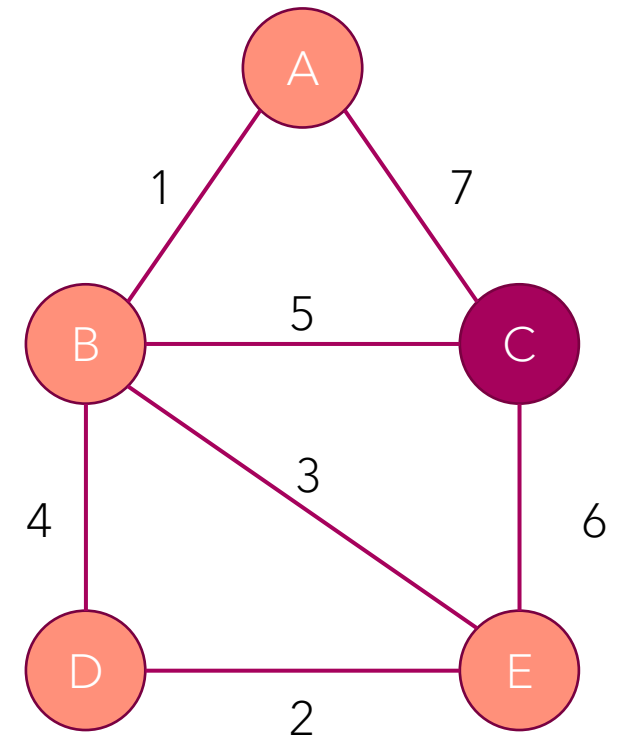
u: D

(2) Inseriu no MST

(1) Vértice de menor custo

(3) Atualiza custos

Iteração



EXEMPLO

MST = {A,B, E, D, C}

Custos

A	B	C	D	E
0	1	5	2	3

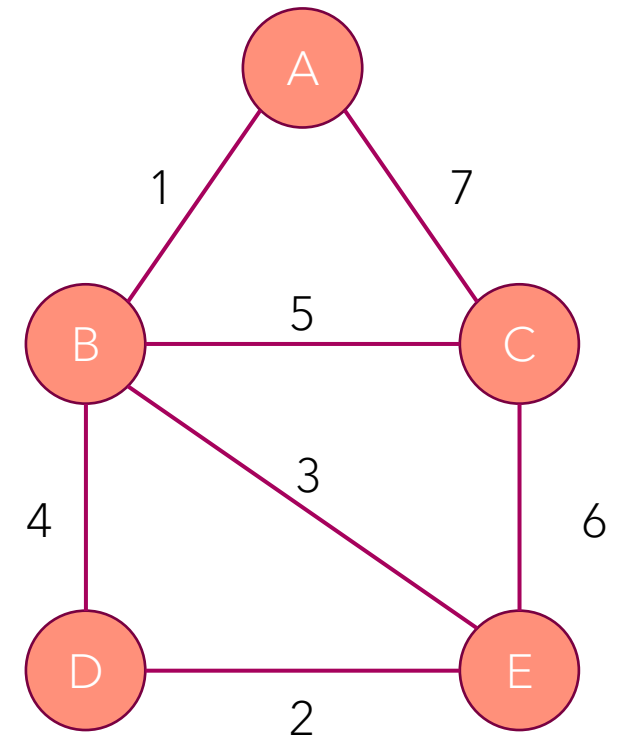
u: C

(2) Inseriu no MST

(1) Vértice de menor custo

(3) Atualiza custos

Iteração



EXEMPLO

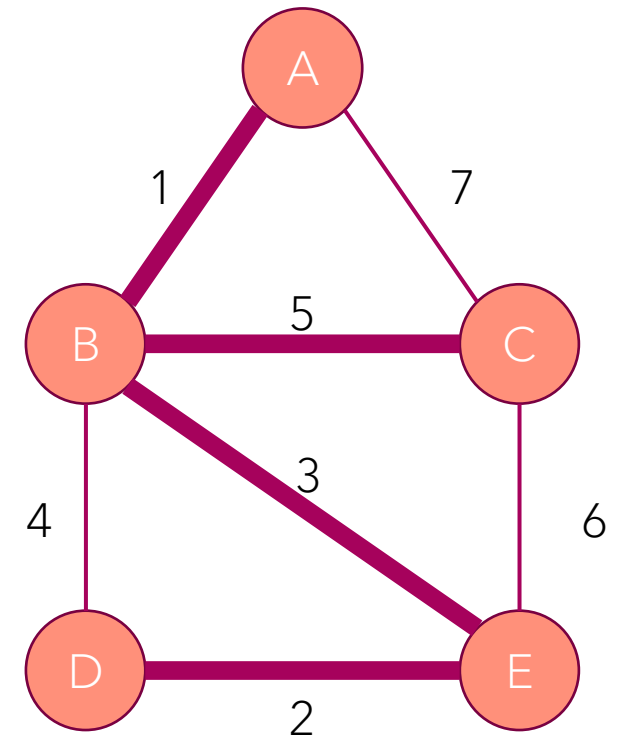
Todos os vértices pertencem a MST

MST = {A,B, E, D, C}

Custos

A	B	C	D	E
0	1	5	2	3

u: C



FIM

COMPLEXIDADE DO ALGORITMO DE PRIM

COMPLEXIDADE DO ALGORITMO DE PRIM

- A complexidade desse algoritmo varia de acordo com a estrutura de dados usada para representar o grafo e da implementação
 - Usando Listas de Adjacência, pode chegar a $O(E \log V)$
 - Usando Matriz de Adjacência, pode chegar a $O(V^2)$



UM ALGORITMO ADICIONAL*

**COMO SABER SE 2
VÉRTICES
PERTENCEM À
MESMA ÁRVORE?**

BÔNUS*

- No algoritmo de Kruskal, é necessário saber se 2 vértices (**u** e **v**) pertencem à mesma árvore.
- Na explicação foi dito que é possível saber isso em tempo $O(\log V)$
- Se usarmos uma busca em profundidade a partir de **u** e verificar se chegamos em **v**, teremos uma operação de tempo $O(V+E)$
- Existe estrutura de dados que pode ajudar: **Disjoint Sets (Conjuntos Disjuntos)**
- Essa estrutura está disponível em material adicional

REFERÊNCIAS USADAS NESSE MATERIAL

- FEOFILOFF, P. Algoritmo de Prim. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/prim.html>. Acesso em: 24 jun. 2020.
- CORMEN, Thomas. **Desmistificando algoritmos**. Elsevier Brasil, 2017.