

# Grafos

*Algoritmo de Prim*

**Prof. Edson Alves**

**Faculdade UnB Gama**

## Proponentes



**Vojtěch Jarník**  
(1930)



**Robert Clay Prim**  
(1957)



**Edsger Wybe Dijkstra**  
(1959)

## **Características do algoritmo de Prim**

## Características do algoritmo de Prim

- ★ O algoritmo de Prim encontra uma MST usando uma abordagem gulosa

## Características do algoritmo de Prim

- ★ O algoritmo de Prim encontra uma MST usando uma abordagem gulosa
- ★ Um vértice  $u$  é escolhido para iniciar um componente conectado  $C$

## Características do algoritmo de Prim

- ★ O algoritmo de Prim encontra uma MST usando uma abordagem gulosa
- ★ Um vértice  $u$  é escolhido para iniciar um componente conectado  $C$
- ★ Enquanto  $C \neq V$ , deve se identificar o vértice  $u \notin C$  mais próximo de  $C$

## Características do algoritmo de Prim

- ★ O algoritmo de Prim encontra uma MST usando uma abordagem gulosa
- ★ Um vértice  $u$  é escolhido para iniciar um componente conectado  $C$
- ★ Enquanto  $C \neq V$ , deve se identificar o vértice  $u \notin C$  mais próximo de  $C$
- ★ Então  $u$  é inserido em  $C$  e a aresta que uniu  $u$  a  $C$  faz parte de uma MST

## Características do algoritmo de Prim

- ★ O algoritmo de Prim encontra uma MST usando uma abordagem gulosa
- ★ Um vértice  $u$  é escolhido para iniciar um componente conectado  $C$
- ★ Enquanto  $C \neq V$ , deve se identificar o vértice  $u \notin C$  mais próximo de  $C$
- ★ Então  $u$  é inserido em  $C$  e a aresta que uniu  $u$  a  $C$  faz parte de uma MST
- ★ Complexidade:  $O(E \log V)$



# Pseudocódigo

# Pseudocódigo

**Entrada:** um grafo ponderado  $G(V, E)$

**Saída:** uma MST de  $G$

# Pseudocódigo

**Entrada:** um grafo ponderado  $G(V, E)$

**Saída:** uma MST de  $G$

1. Escolha um vértice  $u \in V$  e faça  $C = \{u\}, M = \emptyset$

# Pseudocódigo

**Entrada:** um grafo ponderado  $G(V, E)$

**Saída:** uma MST de  $G$

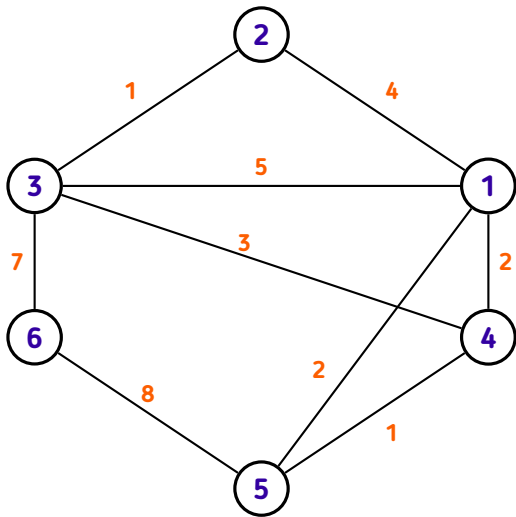
1. Escolha um vértice  $u \in V$  e faça  $C = \{u\}, M = \emptyset$
2. Enquanto  $C \neq V$ :
  - (a) Escolha o vértice  $v \notin C$  mais próximo de  $C$
  - (b) Inclua  $v$  em  $C$  e a aresta que une  $v$  a  $C$  em  $M$

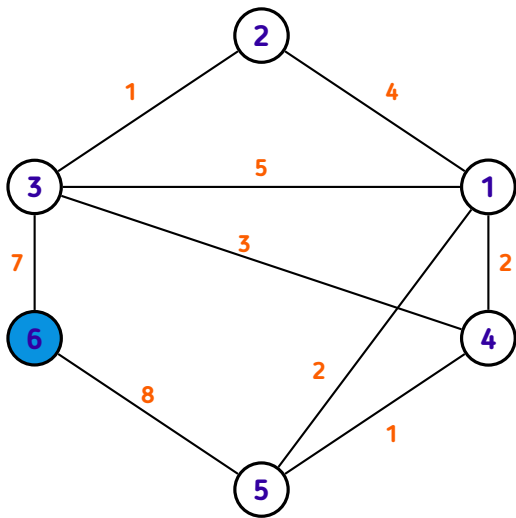
# Pseudocódigo

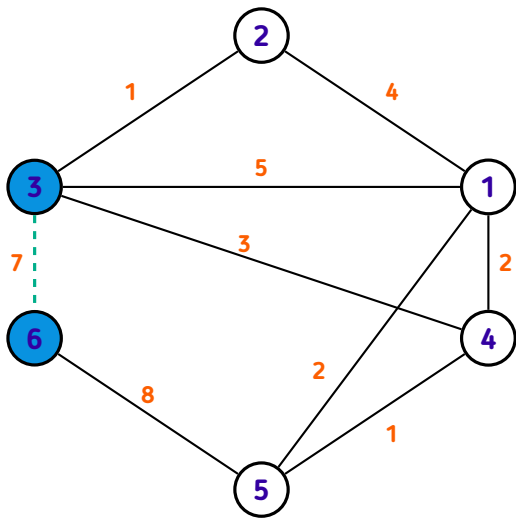
**Entrada:** um grafo ponderado  $G(V, E)$

**Saída:** uma MST de  $G$

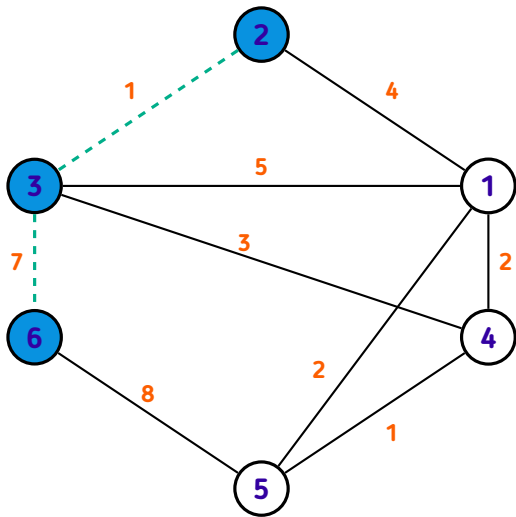
1. Escolha um vértice  $u \in V$  e faça  $C = \{u\}$ ,  $M = \emptyset$
2. Enquanto  $C \neq V$ :
  - (a) Escolha o vértice  $v \notin C$  mais próximo de  $C$
  - (b) Inclua  $v$  em  $C$  e a aresta que une  $v$  a  $C$  em  $M$
3. Retorne  $M$

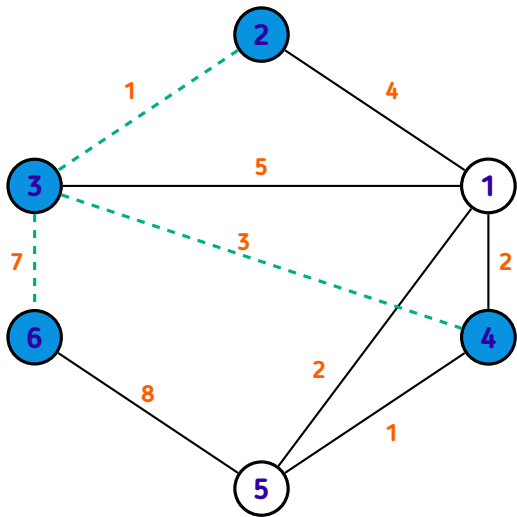


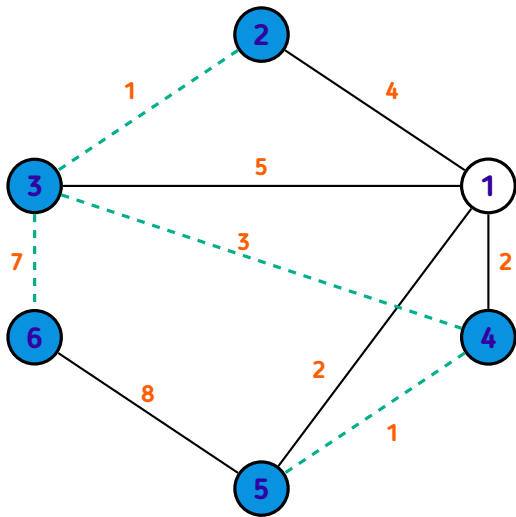


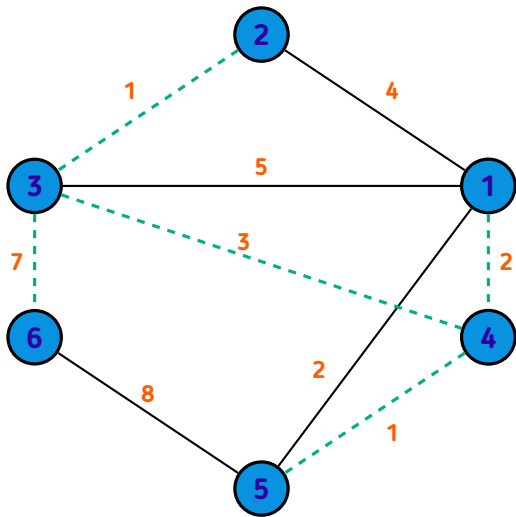












**Identificação eficiente do vértice mais próximo de  $C$**

## Identificação eficiente do vértice mais próximo de $C$

- ★ A complexidade do algoritmo de Prim depende da identificação eficiente do vértice  $v$  mais próximo de  $C$

## Identificação eficiente do vértice mais próximo de $C$

- ★ A complexidade do algoritmo de Prim depende da identificação eficiente do vértice  $v$  mais próximo de  $C$
- ★ O vértice  $v \notin C$  é o mais próximo de  $C$  se  $v$  minimiza a distância

$$\text{dist}(v, C) = \min_{u \in C} \{ \text{dist}(v, u) \}$$

## Identificação eficiente do vértice mais próximo de $C$

★ A complexidade do algoritmo de Prim depende da identificação eficiente do vértice  $v$  mais próximo de  $C$

★ O vértice  $v \notin C$  é o mais próximo de  $C$  se  $v$  minimiza a distância

$$\text{dist}(v, C) = \min_{u \in C} \{ \text{dist}(v, u) \}$$

★ Uma forma de se identificar  $v$  é manter uma fila com prioridades  $q$



## Identificação eficiente do vértice mais próximo de $C$

- ★ Inicialmente,  $q$  estará vazia

## Identificação eficiente do vértice mais próximo de $C$

- ★ Inicialmente,  $q$  estará vazia
- ★ Esta fila será ordenada, de forma ascendente, pelas distâncias até  $C$

## Identificação eficiente do vértice mais próximo de $C$

- ★ Inicialmente,  $q$  estará vazia
- ★ Esta fila será ordenada, de forma ascendente, pelas distâncias até  $C$
- ★ A cada vértice adicionado a  $C$  (inclusive o  $u$  inicial), insira em  $q$  pares  $(w, v)$ , onde  $w$  o peso da aresta que une o vértice  $v \notin C$  a  $u$

## Identificação eficiente do vértice mais próximo de $C$

- ★ Inicialmente,  $q$  estará vazia
- ★ Esta fila será ordenada, de forma ascendente, pelas distâncias até  $C$
- ★ A cada vértice adicionado a  $C$  (inclusive o  $u$  inicial), insira em  $q$  pares  $(w, v)$ , onde  $w$  o peso da aresta que une o vértice  $v \notin C$  a  $u$
- ★ O vértice mais próximo  $v$  será dado pelo par mais próximo do início da fila tal que  $v \notin C$

```
int prim(int u, int N)
{
    set<int> C;
    C.insert(u);

    priority_queue<ii, vector<ii>, greater<ii>> pq;

    for (auto [v, w] : adj[u])
        pq.push(ii(w, v));

    int mst = 0;

    while ((int) C.size() < N)
    {
        int v, w;
```

```
    do {  
        w = pq.top().first, v = pq.top().second;  
        pq.pop();  
    } while (C.count(v));  
  
    mst += w;  
    C.insert(v);  
  
    for (auto [s, p] : adj[v])  
        pq.push(ii(p, s));  
}  
  
return mst;  
}
```

## **Aplicação: Minimax**

## Aplicação: Minimax

- ★ Uma MST minimiza o maior peso entre as arestas presente em qualquer árvore geradora



## Aplicação: Minimax

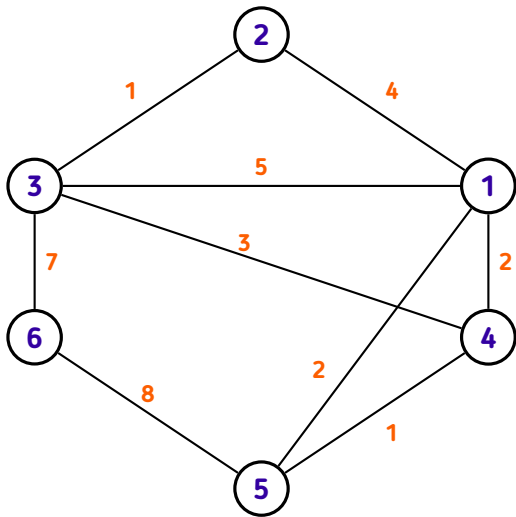
- ★ Uma MST minimiza o maior peso entre as arestas presente em qualquer árvore geradora
- ★ O problema de se minimizar tal peso é denominado *minimax*

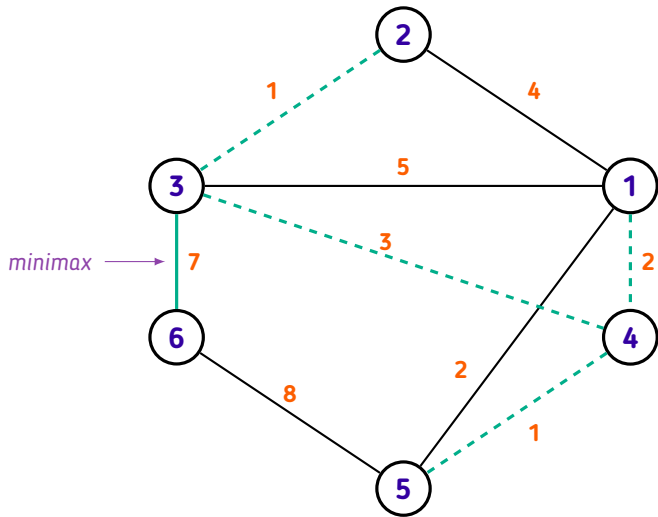
## Aplicação: Minimax

- ★ Uma MST minimiza o maior peso entre as arestas presente em qualquer árvore geradora
- ★ O problema de se minimizar tal peso é denominado *minimax*
- ★ Uma variante deste problema é o *maximin*, que maximiza o menor peso entre as arestas presentes em qualquer árvore geradora

## Aplicação: Minimax

- ★ Uma MST minimiza o maior peso entre as arestas presente em qualquer árvore geradora
- ★ O problema de se minimizar tal peso é denominado *minimax*
- ★ Uma variante deste problema é o *maximin*, que maximiza o menor peso entre as arestas presentes em qualquer árvore geradora
- ★ Uma variante simples do algoritmo de Prim resolve o *minimax*





```
int minimax(int u, int N)
{
    set<int> C;
    C.insert(u);

    priority_queue<ii, vector<ii>, greater<ii>> pq;

    for (auto [v, w] : adj[u])
        pq.push(ii(w, v));

    int minmax = -oo;

    while ((int) C.size() < N)
    {
        int v, w;
```

```
    do {  
        w = pq.top().first, v = pq.top().second;  
        pq.pop();  
    } while (C.count(v));  
  
    minmax = max(minmax, w);  
    C.insert(v);  
  
    for (auto [s, p] : adj[v])  
        pq.push(ii(p, s));  
}  
  
return minmax;  
}
```

## **Aplicação: Menor Subgrafo Gerador**



## Aplicação: Menor Subgrafo Gerador

★ Seja  $G(V, E)$  um grafo conectado e ponderado e  $E' \subset E$

## Aplicação: Menor Subgrafo Gerador

- ★ Seja  $G(V, E)$  um grafo conectado e ponderado e  $E' \subset E$
- ★ O menor subgrafo gerador  $S_{E'}$  de  $G$  é um subgrafo conectado que contém todas as arestas  $E'$  e que tem custo mínimo

## Aplicação: Menor Subgrafo Gerador

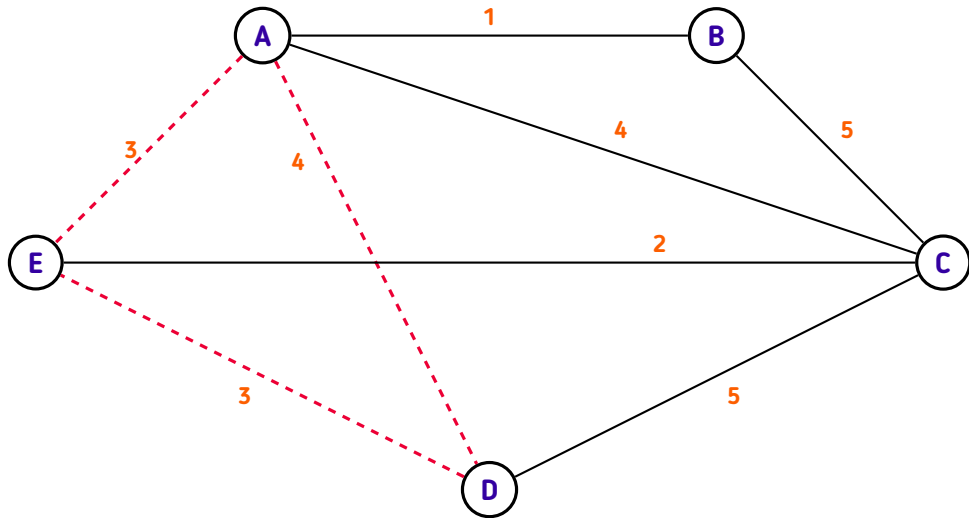
- ★ Seja  $G(V, E)$  um grafo conectado e ponderado e  $E' \subset E$
- ★ O menor subgrafo gerador  $S_{E'}$  de  $G$  é um subgrafo conectado que contém todas as arestas  $E'$  e que tem custo mínimo
- ★ Por conta da restrição  $E'$ ,  $S_{E'}$ , não é, necessariamente, uma MST

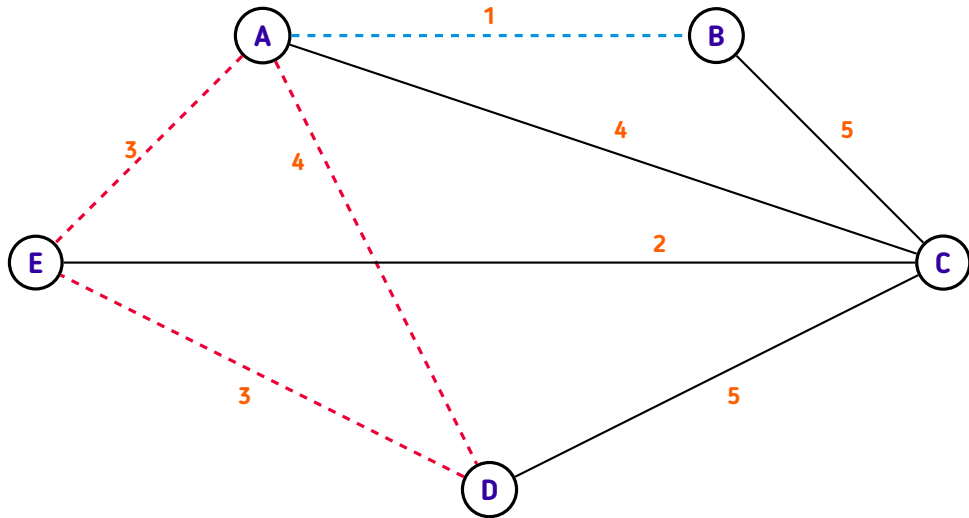
## Aplicação: Menor Subgrafo Gerador

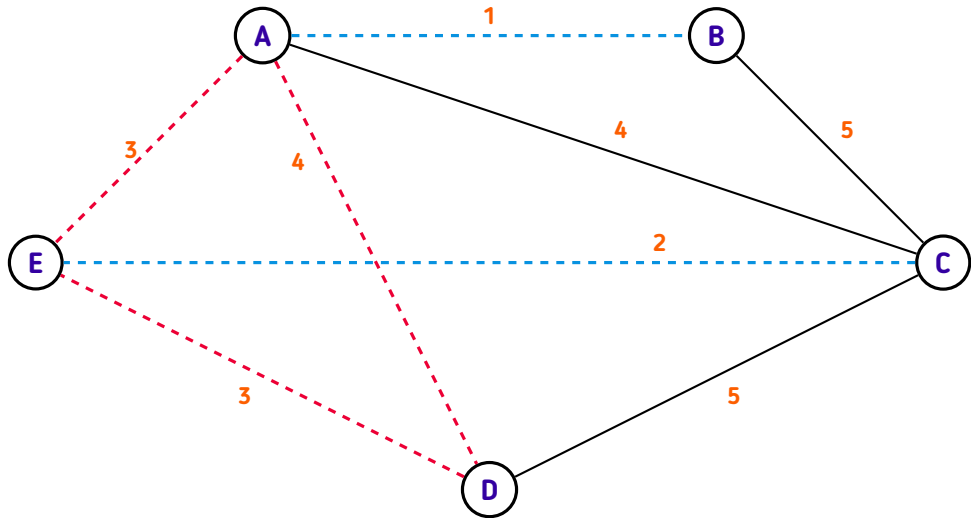
- ★ Seja  $G(V, E)$  um grafo conectado e ponderado e  $E' \subset E$
- ★ O menor subgrafo gerador  $S_{E'}$  de  $G$  é um subgrafo conectado que contém todas as arestas  $E'$  e que tem custo mínimo
- ★ Por conta da restrição  $E'$ ,  $S_{E'}$ , não é, necessariamente, uma MST
- ★ O menor subgrafo gerador pode ser encontrado atribuindo a  $C$  todos os vértices ligados por alguma aresta em  $E'$  no passo inicial do algoritmo de Prim

## Aplicação: Menor Subgrafo Gerador

- ★ Seja  $G(V, E)$  um grafo conectado e ponderado e  $E' \subset E$
- ★ O menor subgrafo gerador  $S_{E'}$  de  $G$  é um subgrafo conectado que contém todas as arestas  $E'$  e que tem custo mínimo
- ★ Por conta da restrição  $E'$ ,  $S_{E'}$ , não é, necessariamente, uma MST
- ★ O menor subgrafo gerador pode ser encontrado atribuindo a  $C$  todos os vértices ligados por alguma aresta em  $E'$  no passo inicial do algoritmo de Prim
- ★ A complexidade é a mesma do algoritmo original









```
int msg(int N, const vector<edge>& es)
{
    set<int> C;
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    int cost = 0;

    for (auto [u, v, w] : es)
    {
        cost += w;

        C.insert(u);
        C.insert(v);

        for (auto [r, s] : adj[u])
            pq.push(ii(s, r));

        for (auto [r, s] : adj[v])
            pq.push(ii(s, r));
    }
}
```

```
while ((int) C.size() < N)
{
    int v, w;

    do {
        w = pq.top().first, v = pq.top().second;
        pq.pop();
    } while (C.count(v));

    cost += w;
    C.insert(v);

    for (auto [s, p] : adj[v])
        pq.push(ii(p, s));
}

return cost;
}
```

## Problemas sugeridos

1. CSES 1675 – Road Reparation
2. OJ 10048 – Audiophobia
3. OJ 10099 – Tourist Guide
4. SPOJ IITKWPCG – Help the old King

## Referências

1. HALIM, Felix; HALIM, Steve. *Competitive Programming 3*, 2010.
2. IT History Society. *Dr. Robert Clay Prim*, acesso em 28/08/2021.
3. LAAKSONEN, Antti. *Competitive Programmer's Handbook*, 2018.
4. Wikipédia. *Edsger Wybe Dijkstra*, acesso em 28/08/2021.
5. Wikipédia. *Prim's algorithm*, acesso em 28/08/2021.
6. Wikipédia. *Vojtěch Jarník*, acesso em 28/08/2021.