

Grafos

Algoritmo de Kruskal

Prof. Edson Alves

Faculdade UnB Gama

Proponente



Joseph Bernard Kruskal, Jr.
(1962)

Características do algoritmo de Kruskal

Características do algoritmo de Kruskal

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

Características do algoritmo de Kruskal

- ★ O algoritmo de Kruskal encontra uma MST usando uma abordagem gulosa
- ★ As arestas são ordenadas, ascendentemente, por peso

Características do algoritmo de Kruskal

- ★ O algoritmo de Kruskal encontra uma MST usando uma abordagem gulosa
- ★ As arestas são ordenadas, ascendentemente, por peso
- ★ Inicialmente os vértices formam uma floresta de vértices isolados

Características do algoritmo de Kruskal

- ★ O algoritmo de Kruskal encontra uma MST usando uma abordagem gulosa
- ★ As arestas são ordenadas, ascendentemente, por peso
- ★ Inicialmente os vértices formam uma floresta de vértices isolados
- ★ Na ordem estipulada, cada aresta que une dois componentes conectados fará parte da MST e unirá estes componentes distintos

Características do algoritmo de Kruskal

- ★ O algoritmo de Kruskal encontra uma MST usando uma abordagem gulosa
- ★ As arestas são ordenadas, ascendentemente, por peso
- ★ Inicialmente os vértices formam uma floresta de vértices isolados
- ★ Na ordem estipulada, cada aresta que une dois componentes conectados fará parte da MST e unirá estes componentes distintos
- ★ Complexidade: $O(E \log V)$

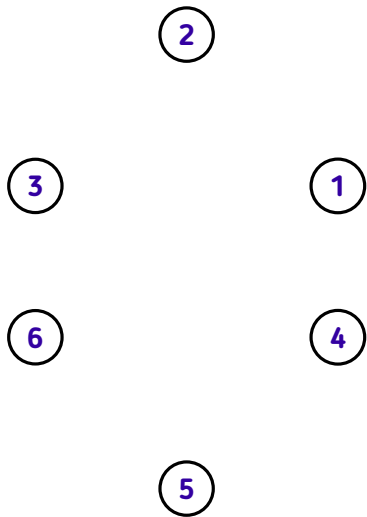
Pseudocódigo

Pseudocódigo

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

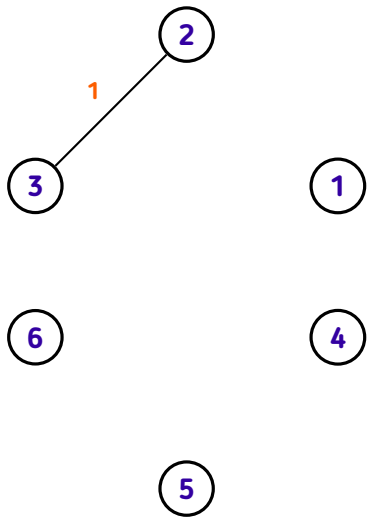
Saída: uma MST de G

1. Faça $M = \emptyset$ e seja $F(V, \emptyset)$ uma floresta de vértices isolados
2. Ordene E ascendentemente, por peso
3. Para cada $(u, v, w) \in E$, se u e v estão em componentes distintos de F :
 - (a) Una estes componentes em F
 - (b) Inclua (u, v, w) no conjunto M
4. Retorne M



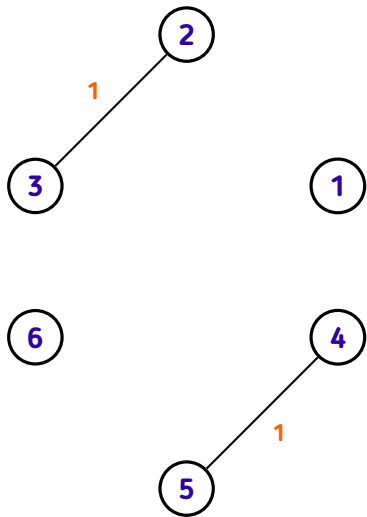
E

1 2 3
1 4 5
2 4 1
2 1 5
3 3 4
4 2 1
5 3 1
7 3 6
8 6 5



E

1 2 3 ✓
1 4 5
2 4 1
2 1 5
3 3 4
4 2 1
5 3 1
7 3 6
8 6 5



E

1 2 3 ✓

1 4 5 ✓

2 4 1

2 1 5

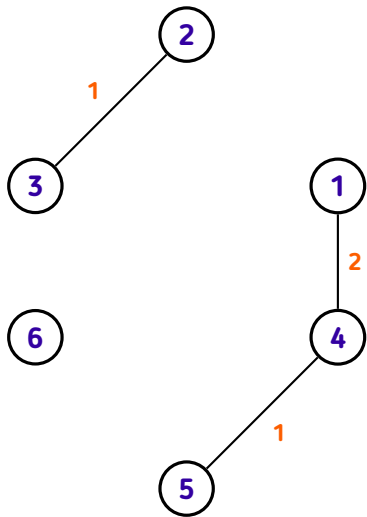
3 3 4

4 2 1

5 3 1

7 3 6

8 6 5



E

1 2 3 ✓

1 4 5 ✓

2 4 1 ✓

2 1 5

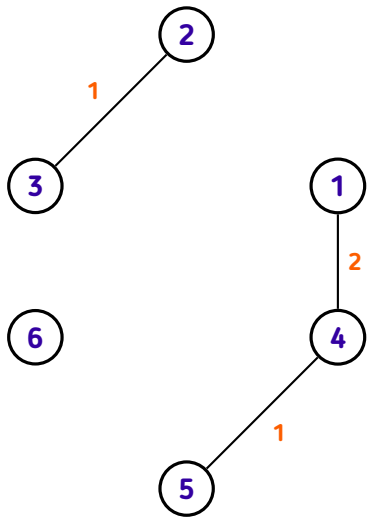
3 3 4

4 2 1

5 3 1

7 3 6

8 6 5



E

1 2 3 ✓

1 4 5 ✓

2 4 1 ✓

2 1 5 ✗

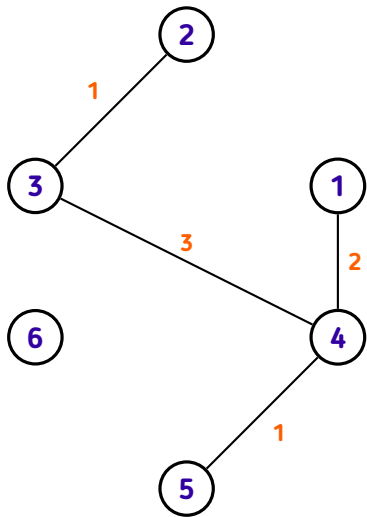
3 3 4

4 2 1

5 3 1

7 3 6

8 6 5



E

1 2 3 ✓

1 4 5 ✓

2 4 1 ✓

2 1 5 ✗

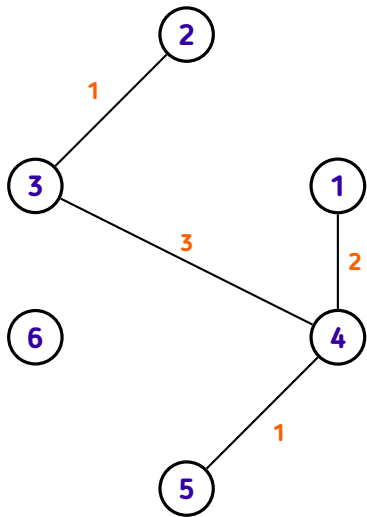
3 3 4 ✓

4 2 1

5 3 1

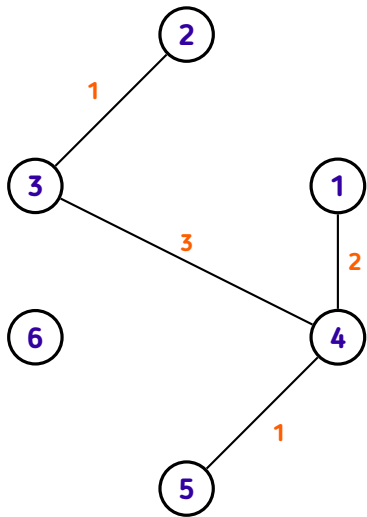
7 3 6

8 6 5



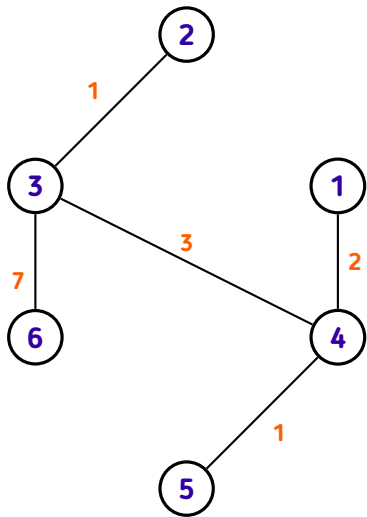
E

1	2	3	✓
1	4	5	✓
2	4	1	✓
2	1	5	✗
3	3	4	✓
4	2	1	✗
5	3	1	
7	3	6	
8	6	5	



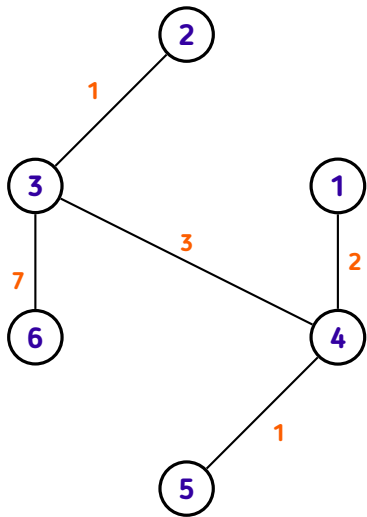
E

1	2	3	✓
1	4	5	✓
2	4	1	✓
2	1	5	✗
3	3	4	✓
4	2	1	✗
5	3	1	✗
7	3	6	
8	6	5	



E

1	2	3	✓
1	4	5	✓
2	4	1	✓
2	1	5	✗
3	3	4	✓
4	2	1	✗
5	3	1	✗
7	3	6	✓
8	6	5	



E

1	2	3	✓
1	4	5	✓
2	4	1	✓
2	1	5	✗
3	3	4	✓
4	2	1	✗
5	3	1	✗
7	3	6	✓
8	6	5	✗

Identificação e união dos componentes conectados

Identificação e união dos componentes conectados

★ A complexidade do algoritmo de Kruskal depende da identificação e união eficiente dos componentes conectados

Identificação e união dos componentes conectados

- ★ A complexidade do algoritmo de Kruskal depende da identificação e união eficiente dos componentes conectados
- ★ A *union-find disjoint sets* (UFDS) (também conhecida como *disjoint set union* – DSU) é uma estrutura de dados que atende a esta demanda

Identificação e união dos componentes conectados

- ★ A complexidade do algoritmo de Kruskal depende da identificação e união eficiente dos componentes conectados
- ★ A *union-find disjoint sets* (UFDS) (também conhecida como *disjoint set union* – DSU) é uma estrutura de dados que atende a esta demanda
- ★ A operação `same_set(u , v)` retorna verdadeiro se ambos u e v pertencem ao mesmo componente conectado, em $O(\log V)$

Identificação e união dos componentes conectados

- ★ A complexidade do algoritmo de Kruskal depende da identificação e união eficiente dos componentes conectados
- ★ A *union-find disjoint sets* (UFDS) (também conhecida como *disjoint set union* – DSU) é uma estrutura de dados que atende a esta demanda
- ★ A operação `same_set(u , v)` retorna verdadeiro se ambos u e v pertencem ao mesmo componente conectado, em $O(\log V)$
- ★ A operação `union_set(u , v)` une os componentes distintos onde estão localizados u e v , também em $O(\log V)$

```
int kruskal(int N, vector<edge>& es)
{
    sort(es.begin(), es.end());

    int cost = 0;
    UnionFind udfs(N);

    for (auto [w, u, v] : es)
    {
        if (not udfs.same_set(u, v))
        {
            cost += w;
            udfs.union_set(u, v);
        }
    }

    return cost;
}
```

Aplicação: Floresta mínima geradora

Aplicação: Floresta mínima geradora

★ Uma floresta mínima geradora F_k pode ser identificada por meio de uma modificação simples no algoritmo de Kruskal

Aplicação: Floresta mínima geradora

- ★ Uma floresta mínima geradora F_k pode ser identificada por meio de uma modificação simples no algoritmo de Kruskal
- ★ Inicialmente, a floresta de vértices isolados contém V componentes

Aplicação: Floresta mínima geradora

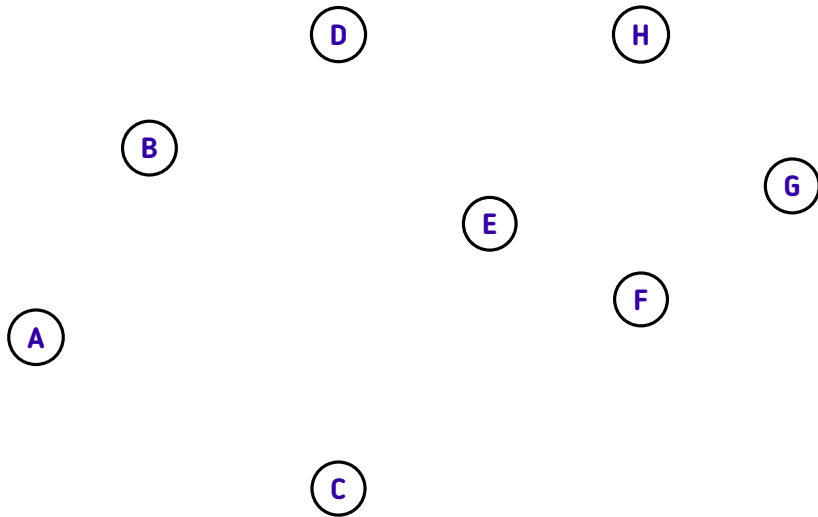
- ★ Uma floresta mínima geradora F_k pode ser identificada por meio de uma modificação simples no algoritmo de Kruskal
- ★ Inicialmente, a floresta de vértices isolados contém V componentes
- ★ A cada união de componentes, o total é reduzido em uma unidade

Aplicação: Floresta mínima geradora

- ★ Uma floresta mínima geradora F_k pode ser identificada por meio de uma modificação simples no algoritmo de Kruskal
- ★ Inicialmente, a floresta de vértices isolados contém V componentes
- ★ A cada união de componentes, o total é reduzido em uma unidade
- ★ Quando a contagem de componentes for igual a k , o algoritmo deve ser encerrado

$$k = 4$$

componentes: 8



$k = 4$

componentes: 8

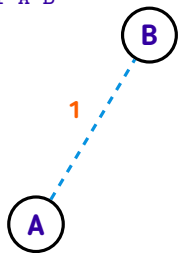
próxima aresta: 1 A B



$$k = 4$$

componentes: 7

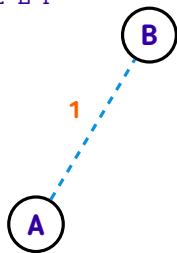
próxima aresta: 1 A B



$k = 4$

componentes: 7

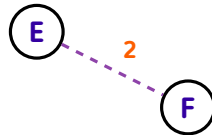
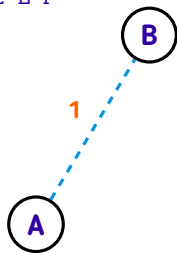
próxima aresta: 2 E F



$$k = 4$$

componentes: 6

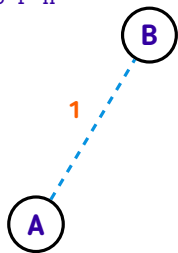
próxima aresta: 2 E F



$k = 4$

componentes: 6

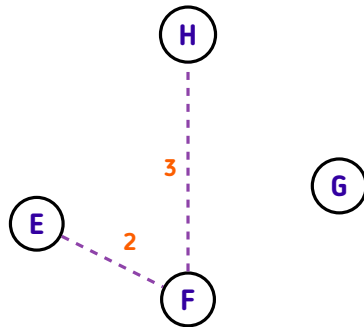
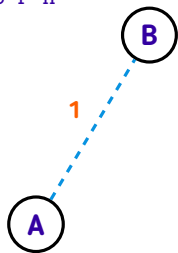
próxima aresta: 3 F H



$$k = 4$$

componentes: 5

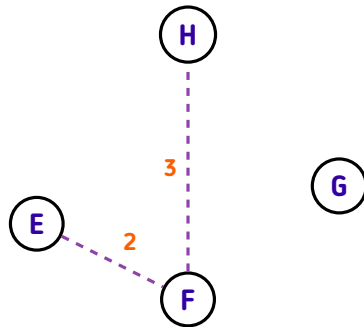
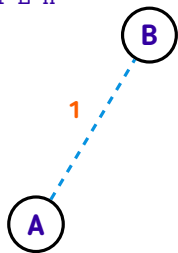
próxima aresta: 3 F H



$k = 4$

componentes: 5

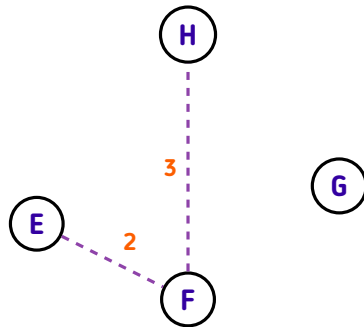
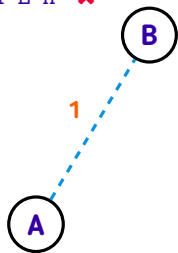
próxima aresta: 4 E H



$k = 4$

componentes: 5

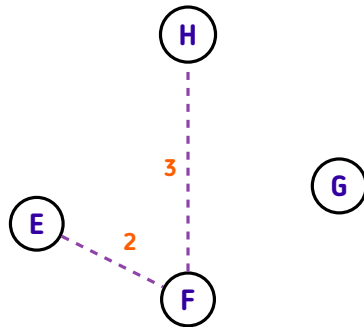
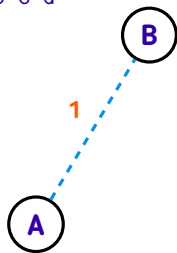
próxima aresta: 4 E H ✖



$k = 4$

componentes: 5

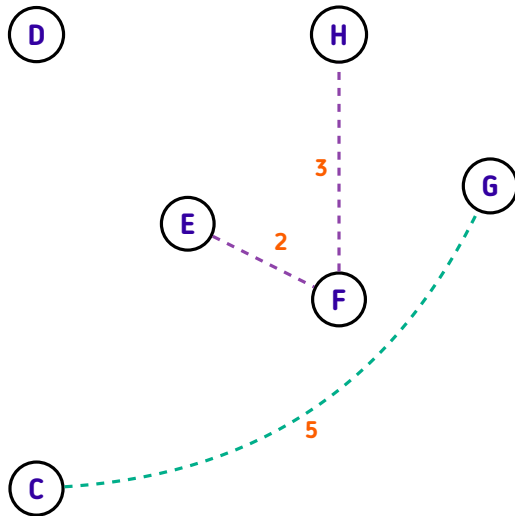
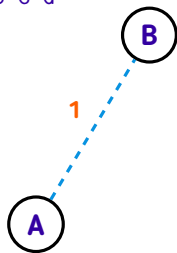
próxima aresta: 5 C G



$k = 4$

componentes: 4

próxima aresta: 5 C G



```
int msf(int k, int N, vector<edge>& es)
{
    sort(es.begin(), es.end());

    int cost = 0, cc = N;
    UnionFind udfs(N);

    for (auto [w, u, v] : es)
    {
        if (not udfs.same_set(u, v)) {
            cost += w;
            udfs.union_set(u, v);

            if (--cc == k)
                return cost;
        }
    }

    return cost;
}
```

Aplicação: Segunda melhor MST

Aplicação: Segunda melhor MST

- ★ Algumas aplicações demandam a identificação da segunda melhor MST

Aplicação: Segunda melhor MST

- ★ Algumas aplicações demandam a identificação da segunda melhor MST
- ★ Esta segunda melhor MST, se existir, pode ser identificada por meio de uma nova modificação no algoritmo de Kruskal

Aplicação: Segunda melhor MST

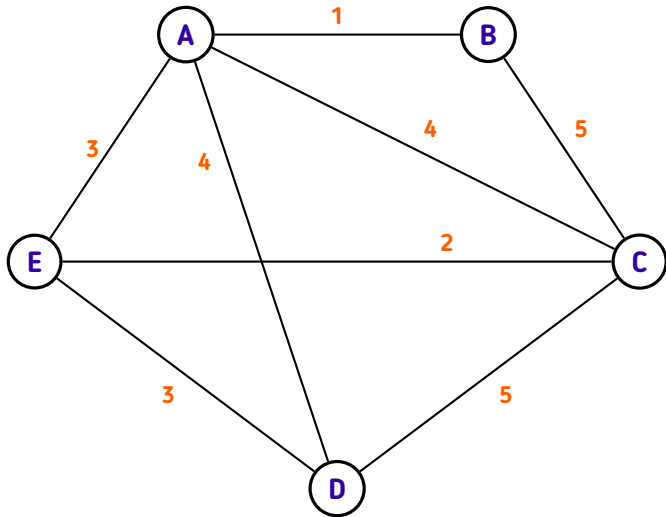
- ★ Algumas aplicações demandam a identificação da segunda melhor MST
- ★ Esta segunda melhor MST, se existir, pode ser identificada por meio de uma nova modificação no algoritmo de Kruskal
- ★ Primeiramente, deve ser identificada a MST

Aplicação: Segunda melhor MST

- ★ Algumas aplicações demandam a identificação da segunda melhor MST
- ★ Esta segunda melhor MST, se existir, pode ser identificada por meio de uma nova modificação no algoritmo de Kruskal
- ★ Primeiramente, deve ser identificada a MST
- ★ Em seguida, cada uma das arestas da MST deve ser excluída e o algoritmo deve identificar a MST formada pelas arestas restantes

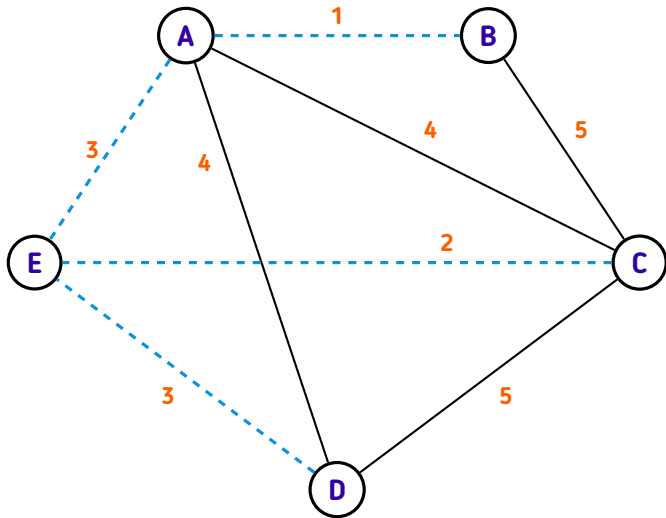
Aplicação: Segunda melhor MST

- ★ Algumas aplicações demandam a identificação da segunda melhor MST
- ★ Esta segunda melhor MST, se existir, pode ser identificada por meio de uma nova modificação no algoritmo de Kruskal
- ★ Primeiramente, deve ser identificada a MST
- ★ Em seguida, cada uma das arestas da MST deve ser excluída e o algoritmo deve identificar a MST formada pelas arestas restantes
- ★ Complexidade: $O(VE)$



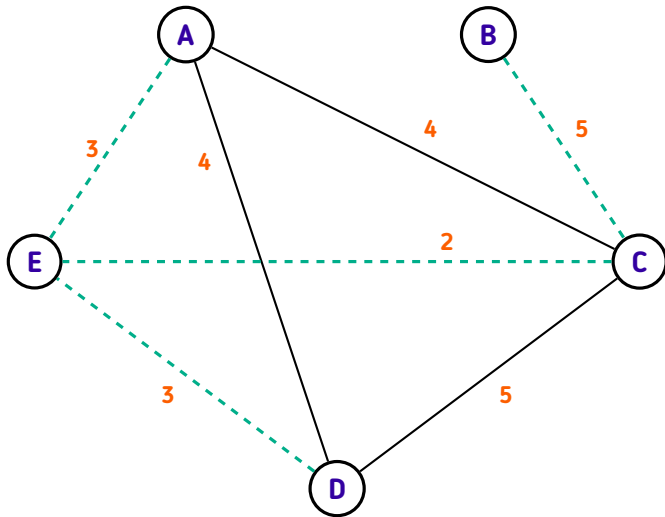
MST

$$c(\text{MST}) = 9$$



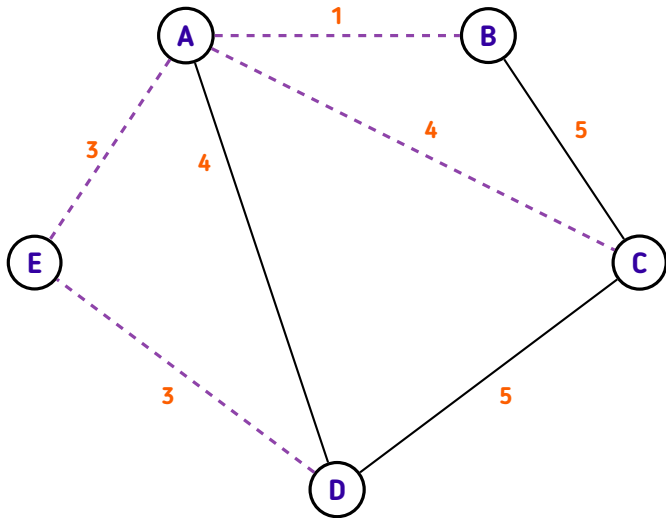
Árvore geradora T_1

$$c(T_1) = 13$$



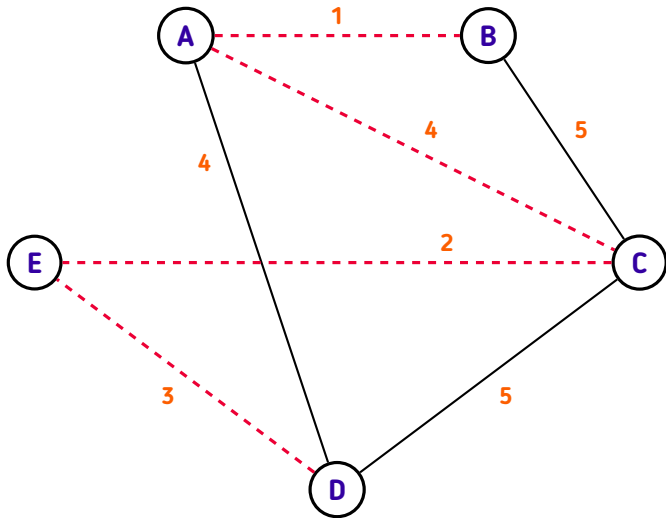
Árvore geradora T_2

$$c(T_2) = 11$$



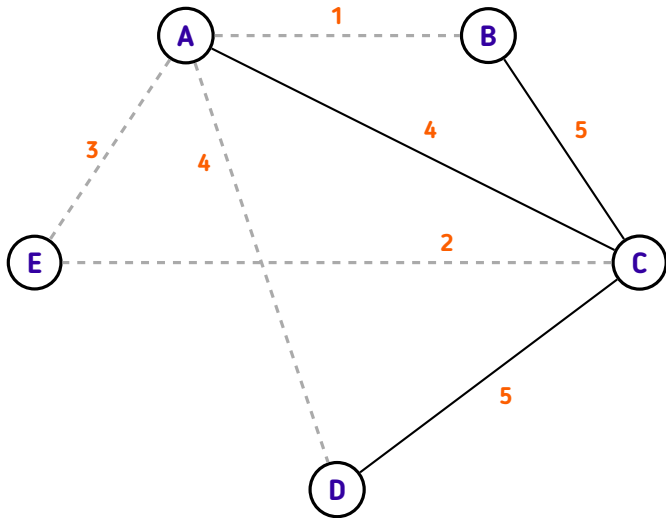
Árvore geradora T_3

$$c(T_3) = 10$$



Árvore geradora T_4

$$c(T_4) = 10$$



```

pair<int, vector<int>>
kruskal(int N, vector<edge>& es, int blocked = -1)
{
    vector<int> mst;
    UnionFind ufds(N);
    int cost = 0;

    for (int i = 0; i < (int) es.size(); ++i)
    {
        auto [w, u, v] = es[i];

        if (i != blocked and not ufds.same_set(u, v)) {
            cost += w;
            ufds.union_set(u, v);
            mst.emplace_back(i);
        }
    }

    return { (int) mst.size() == N - 1 ? cost : oo, mst };
}

```



```
int second_best(int N, vector<edge>& es)
{
    sort(es.begin(), es.end());

    auto [_ , mst] = kruskal(N, es);
    int best = oo;

    for (auto blocked : mst)
    {
        auto [cost, __] = kruskal(N, es, blocked);
        best = min(best, cost);
    }

    return best;
}
```

Problemas sugeridos

1. [Codechef CHEFELEC – Chefland and Electricity](#)
2. [OJ 10369 – Artic Network](#)
3. [OJ 10600 – ACM Contest and Blackout](#)
4. [SPOJ EC_MODE – Modems](#)

Referências

1. CP-Algorithm. *Minimum spanning tree – Kruskal's algorithm*, acesso em 24/08/2021.
2. HALIM, Felix; HALIM, Steve. *Competitive Programming 3*, 2010.
3. LAAKSONEN, Antti. *Competitive Programmer's Handbook*, 2018.
4. Wikipédia. *Joseph Kruskal*, acesso em 24/08/2021.
5. Wikipédia. *Kruskal's algorithm*, acesso em 24/08/2021.