PAA - Aula 15
Árvores Geradoras Mínimas (MST), Algoritmo de Kruskal

Relembrando o problema da árvore geradora mínima,

- na entrada temos um grafo G = (V, E)
    - com custo c(e) > 0 em cada aresta e \in E.

Uma solução é uma árvore geradora T de custo mínimo, isto é,

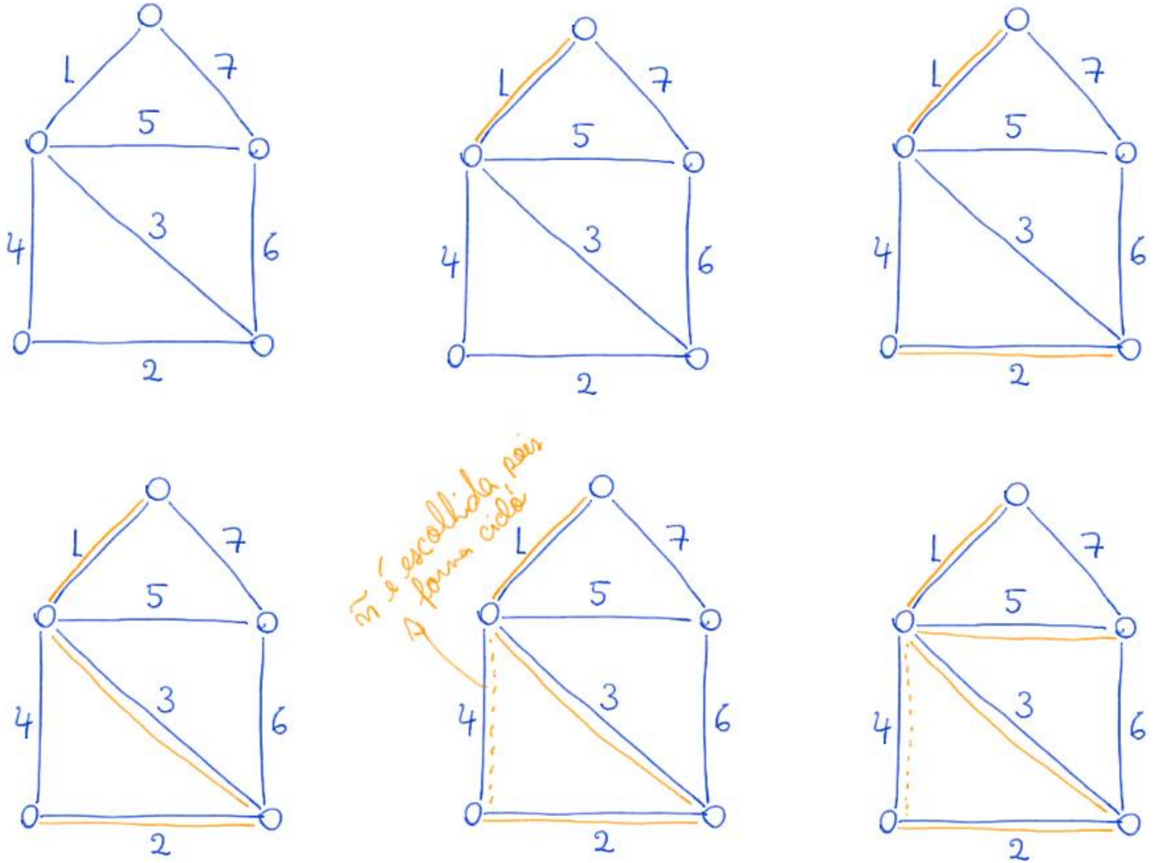
- um subgrafo conexo que não tem ciclos,
    - e que contém todos os vértices de G, i.e., V(T) = V.
- Note que, existe um caminho em T entre qualquer par de vértices de V.
- Além disso, entre todas as árvores geradoras de G,
    - o custo de T, i.e., c(T) = \sum_{e \in T} c(e) deve ser mínimo.

Nosso guia no projeto de um algoritmo guloso ainda é a Propriedade do Corte:

- Dado um grafo G = (V, E), considere uma aresta e \in E
    - e suponha que existe um corte (A, B) tal que
        - ‘e’ é a aresta mais barata de G que cruza este corte.
- Então ‘e’ está na árvore geradora mínima de G.
- Podemos falar DA árvore geradora mínima
    - porque supomos que não existem arestas com mesmo custo.

Algoritmo de Kruskal: vamos ver um exemplo do algoritmo de Kruskal,

- cuja ideia é reiteradamente selecionar a aresta de menor custo.



Pseudocódigo do algoritmo de Kruskal:

```
Kruskal (G=(V,E), custos c):
    ordene as arestas em ordem crescente de custo
    T = {}
    Para cada aresta e = (u, v) em ordem crescente de custo:
        se T unida com 'e' não forma um ciclo:
            Adicione e em T
    devolva T
```

Implementação básica e eficiência do algoritmo de Kruskal:

- Temos de ordenar as arestas, o que leva tempo O(m log n).
    - Curiosidade: ordenar m arestas leva tempo O(m log n),
        - pois O(log m) = O(log n) já que m <= n^2 e log n^2 = 2 log n.
- O laço principal testa se cada aresta forma um ciclo em T, o que
    - pode ser feito com um algoritmo de busca em grafo (BFS ou DFS).
- Esses algoritmos levam tempo proporcional ao tamanho do grafo,
    - i.e., número de vértice mais número de arestas.
- No entanto, o grafo em questão é a floresta T, cujo tamanho é O(n).
- Assim, O(m) iterações do laço principal, cada uma custando O(n) operações,
    - totalizam O(m * n).
- Portanto, o algoritmo leva tempo O(m log n) + O(m * n) = O(m * n)

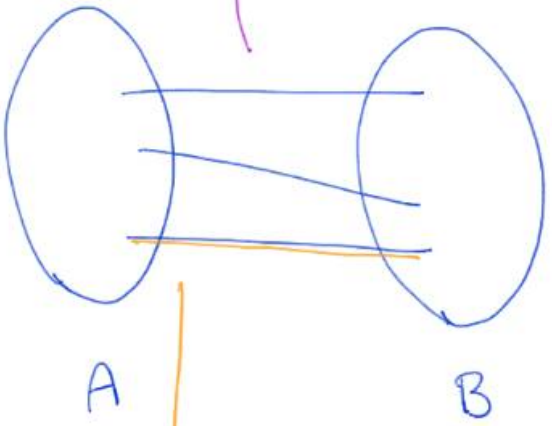
Prova de corretude do algoritmo de Kruskal:

T não possui ciclos, pois o algoritmo descarta qualquer aresta que os gere.

T é geradora (e conexa). Com G sendo conexo,

- será que o subgrafo T do algoritmo pode terminar desconexo?
- Do lema do corte vazio temos que, se o grafo for conexo
  - então não existe um corte vazio.
- Fixe um corte (A, B) qualquer e considere a primeira vez
  - que o algoritmo de Kruskal considera uma aresta deste corte.
- Pelo Corolário da Aresta Solitária, sabemos que
  - esta aresta não gera um ciclo em T.
- Portanto, o algoritmo vai adicionar esta aresta a T.
- Como isso vale para um corte (A, B) qualquer,
  - temos que o subgrafo T produzido pelo algoritmo
    - terá arestas cruzando qualquer corte.
- Assim, T é conexo e gera o grafo G.

A figura a seguir ilustra alguns argumentos dessa demonstração.



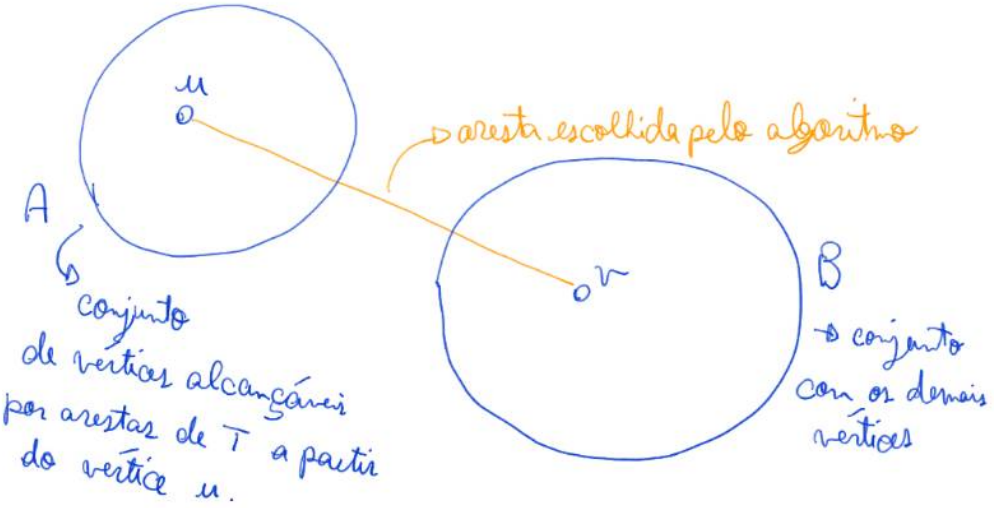
↳ o corte não pode estar vazio por conta do Lema do Corte Vazio

↳ 1ª aresta do corte que o alg. considera Pelo Corol. da Aresta Solitária esta aresta ñ forma ciclo. Por isso, será adicionada pelo alg. na árvore T.

T tem custo mínimo. Vamos mostrar que toda aresta

- escolhida pelo algoritmo de Kruskal respeita a propriedade do Corte.
- Para tanto, vamos encontrar um corte (A, B) que justifique
    - a escolha de uma aresta ‘e’ qualquer pega pelo algoritmo.
- Considere uma aresta e = (u, v) que foi escolhida pelo algoritmo,
    - e seja T o subgrafo do algoritmo no início da iteração dessa escolha.
- Note que não pode haver caminho em T que vai de ‘u’ para ‘v’.
    - Caso contrário, ‘e’ formaria um ciclo em T e não teria sido escolhida.
- Seja A o conjunto de todos os vértices alcançáveis a partir de u em T,
    - e seja B o restante dos vértices.
- Por construção, não existem arestas em T com uma ponta em A
    - e outra em B. Caso contrário, A alcançaria mais vértices.
- Pelo Corolário da Aresta Solitária, o algoritmo adiciona a T
    - a primeira aresta do corte (A, B) que for considerada,
        - já que tal aresta não pode formar ciclo.
- Portanto, ‘e’ é a primeira aresta deste corte que o algoritmo considerou.
- Como o algoritmo percorre as arestas em ordem crescente de custo,
    - e = (u, v) é a aresta de menor custo que atravessa o corte (A, B).
- Logo, pela Propriedade do Corte
    - ela pertence à árvore geradora mínima do grafo.

A figura a seguir ilustra argumentos argumentos dessa demonstração.



Note que a aresta $e = (u, v)$ é a aresta de menor custo que cruza o corte (A, B). Portanto, pela propriedade do corte ela deve estar na árvore geradora mínima.

Pseudocódigo do algoritmo de Kruskal usando estrutura de dados Union-Find:

```
KruskalUnionFind (G=(V,E), custos c):
    para todo v \in V:
        makeSet(v) // cria um conjunto para cada vértice
    ordene as arestas em ordem crescente de custo
    T = {}
    Para cada aresta e = (u, v) em ordem crescente de custo:
        se find(u) != find(v): // se 'u' e 'v' estão em conjuntos distintos
            adicione e em T
            union(u, v) // faça a união dos conjuntos de 'u' e de 'v'
    devolva T
```

Análise de eficiência do algoritmo de Kruskal implementado com Union-Find:

- makeSet leva tempo O(1), portanto, a inicialização leva tempo O(n).
- Ordenar as arestas leva tempo O(m log n).
- O laço principal é executado O(m) vezes.
- Na implementação do Union-Find usando Union by Rank
    - cada operação leva tempo O(log n).
- Em cada iteração do laço são executadas duas operações find e uma union.
    - Portanto, o tempo total do laço principal é O(m log n).
- Assim, o tempo de execução do algoritmo é O(m log n).