PAA - Aula ~~15~~ 16

Árvores Geradoras Mínimas (MST), Algoritmo de Kruskal

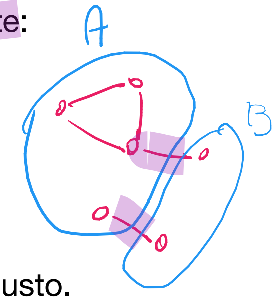
Relembrando o problema da árvore geradora mínima,

- na entrada temos um grafo $G = (V, E)$
    - com custo $c(e) > 0$ em cada aresta $e \in E$
- Quiz1: Os custos precisam ser não negativos?

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

- árvore → um subgrafo conexo que não tem ciclos,
    - geradora → e que contém todos os vértices de
- 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.

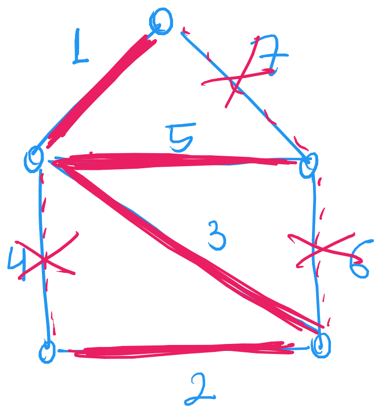
Nosso guia no projeto de um algoritmo guloso ainda é a Propriedade do Corte:

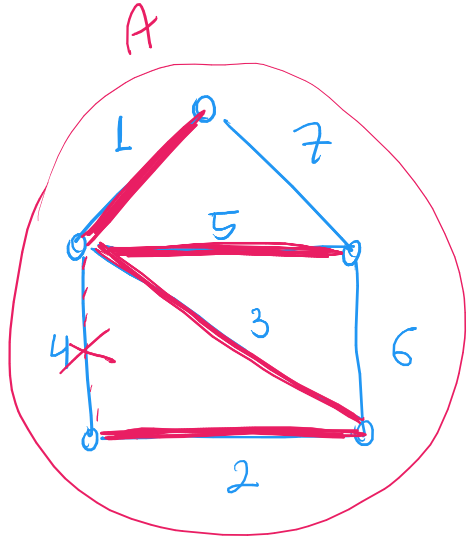


- Dado um grafo $G = (V, E)$ considere uma aresta $e \in E$
    - e tome um corte qualquer (A, B) tal que
        - e é a aresta mais barata de G que cruza este corte.
- Então e está na árvore geradora mínima de G
    - Podemos falar DA árvore geradora mínima
        - porque supomos que não existem 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
  - que não produz um ciclo.





Pseudocódigo do algoritmo de Kruskal: $n = |V|$ $m = |E|$

Kruskal (G=(V,E), custos c):

ordene as arestas em ordem crescente de custo $O(m \lg n)$

$T = \emptyset$

$O(m)$ iterações

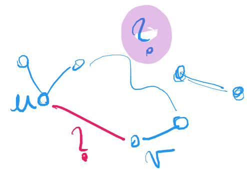
para cada aresta $e = (u, v)$ seguindo a ordenação:

se $T \cup \{e\}$ não forma ciclo:

$T \leftarrow T \cup \{e\}$ $O(n)$

devolve T

$O(m.n)$



Implementação básica e eficiência do algoritmo de Kruskal:

- Temos de ordenar m arestas, o que leva tempo $O(m \lg n) = O(m \lg m)$
  - Curiosidade: $O(\lg m) = O(\lg n)$ pois $m \leq n^2$ 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 $O(m \log n) + O(m.n) = O(m.n)$

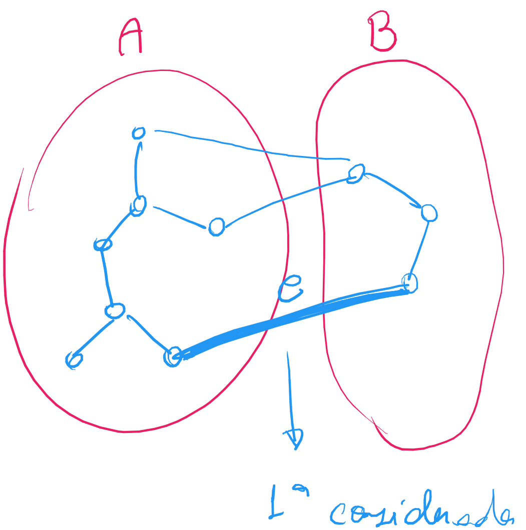
Prova de corretude do algoritmo de Kruskal:

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

T **é geradora e conexa:** Com G sendo conexo,

- será que T pode terminar desconexa?

- Pelo Lema do Corte Vazio sabemos que,
  - se o grafo for conexo
    - não existe um corte vazio.
- Fixe um corte (A, B) qualquer e considere
  - a primeira vez em que o algoritmo
    - trata uma aresta e deste corte.
- Pelo Corolário da Aresta Solitária, sabemos
  - que e não gera um ciclo em T
- Portanto, o algoritmo vai adicionar e a T
- Como isso vale para um corte qualquer,
  - temos que o subgrafo T
    - produzido pelo algoritmo
  - terá arestas cruzando qualquer corte.
- Assim, T é conexa e gera o grafo G

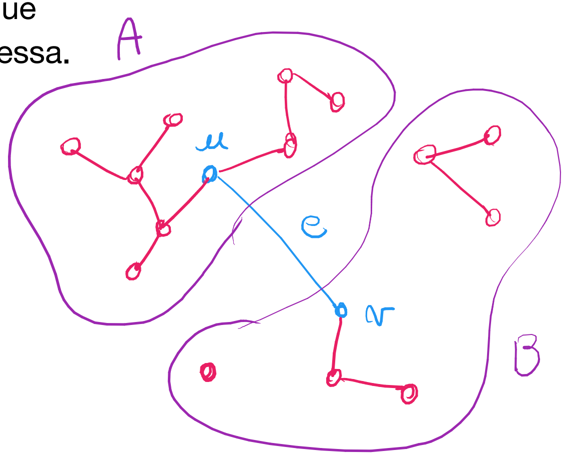


T **tem custo mínimo:** Vamos mostrar que qualquer aresta $e = (u, v)$

- escolhida pelo algoritmo de Kruskal respeita a propriedade do Corte.

o que queremos $\Rightarrow$

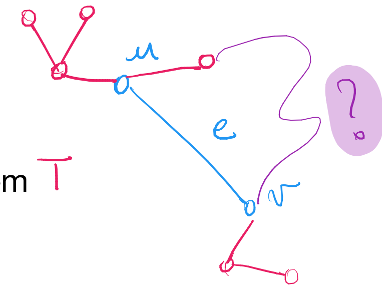
- Ou seja, vamos encontrar um corte (A, B) tal que
  - e é a aresta de menor custo que o atravessa.



- Considere a iteração em que e é escolhida e seja T o subgrafo no início dessa iteração.
- Note que não existe caminho de u para v em T. Senão e formaria um ciclo em T e não seria escolhida.
- Faça 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
- Então e é a primeira aresta escolhida que atravessa o corte (A, B)
  - mas será que e é aresta mais barata a atravessá-lo?
- Pelo Corolário da Aresta Solitária, a primeira aresta do corte
  - que for considerada pelo algoritmo é adicionada a
    - já que tal aresta não pode formar ciclo.
- Assim, e é a primeira aresta de (A, B) analisada pelo algoritmo.
- Como o algoritmo percorre as arestas em ordem crescente de custo,
  - $e = (u, v)$ é a aresta de menor custo que atravessa (A, B)
- Logo, pela Propriedade do Corte, e pertence à árvore geradora mínima.

Sabemos implementar o algoritmo de Kruskal com eficiência de pior caso $O(m\,n)$

- Será que conseguimos fazer melhor?



Se olharmos para o pseudocódigo básico do algoritmo de Kruskal,

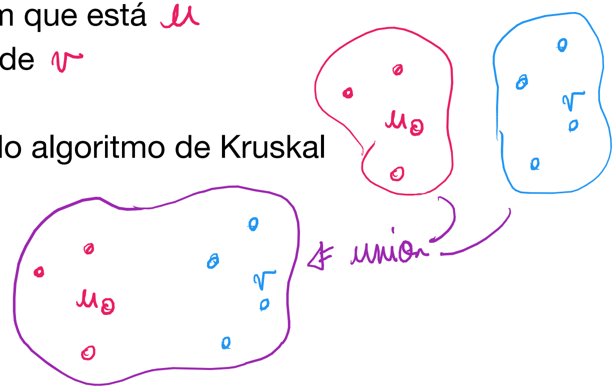
- percebemos que dentro do laço principal
  - temos uma operação que custa $O(n)$ para descobrir
    - se uma nova aresta $e = (u, v)$ produzirá um ciclo em T
  - i.e., para descobrir se os seus extremos u e v
    - estão no mesmo componente de T
- Felizmente, existe uma estrutura de dados chamada Union-Find
  - que é especializada na manutenção de conjuntos disjuntos.

Union-Find suporta três operações:

- makeSet(u) - cria uma parte (subconjunto) apenas com o elemento u
- find(u) - devolve o representante da parte em que está u
- union(u, v) - une a parte de u com a parte de v

A seguir apresentamos (e analisamos) uma versão do algoritmo de Kruskal

- que utiliza essa estrutura de dados.



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

KruskalUnionFind (G=(V,E), custos c):

$O(n)$ — para todo $v \in V$: makeSet(v) — $O(1)$

$O(m \lg n)$ — ordene as arestas em ordem crescente de custo

$T \leftarrow \emptyset$

$O(m)$ iterações — para cada aresta $e = (u,v)$ seguindo a ordenação:

se find(u) $\neq$ find(v):

$T \leftarrow T \cup \{e\}$

union(u, v)

$O(\log n)$

devolva T

$O(m \lg n)$

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 $O(m \lg n)$
- O laço principal é executado $O(m)$ vezes.
- Na implementação do Union-Find usando Union by Rank
    - cada operação find() ou union() leva tempo $O(\lg n)$
- Em cada iteração são executadas duas operações find() e uma union().
    - Portanto, o tempo total do laço principal é $O(m \lg n)$
- Assim, o tempo de execução do algoritmo é $O(m \lg n)$