

Árvores Geradoras Mínimas (MST), Propriedade do Corte, Algoritmo de Prim

Minimum Spanning Tree

O problema da Árvore Geradora Mínima (MST)

- é central nas áreas de grafos e de otimização combinatória.
- Também é uma oportunidade para exercitar o projeto de algoritmos gulosos.

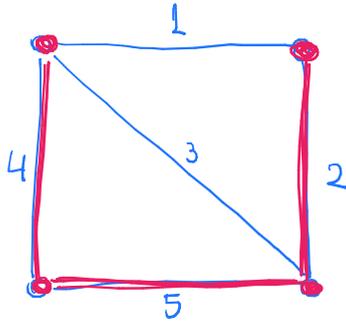
Na entrada temos um grafo $G = (V, E)$

- com custo $c(e) > 0$ em cada aresta $e \in E$

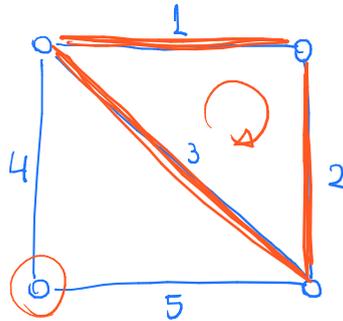
Uma solução é uma árvore geradora T de custo mínimo, isto é,

- 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.

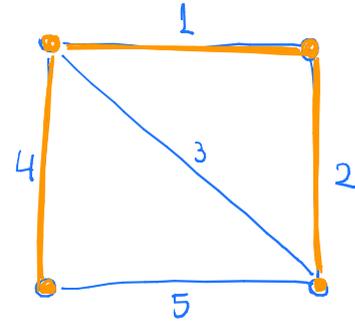
Exemplos:



Spanning Tree, mas não é mínima



Não é árvore
Não é quadrado



MST

Suposições para simplificar:

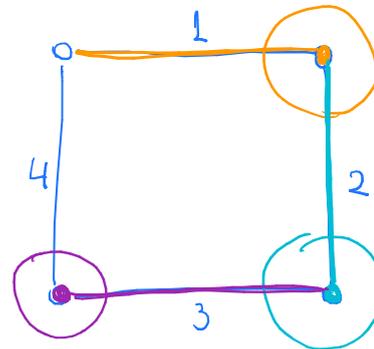
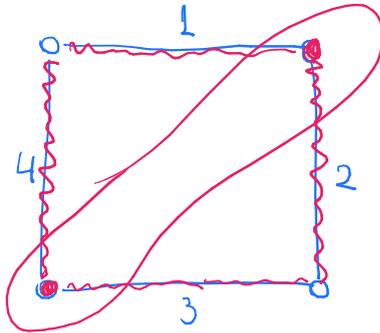
- G é **conexo**, caso contrário não faz sentido falar em **árvore**,
 - mas sim em floresta geradora mínima.
 - É fácil identificar os componentes conexos
 - usando busca em largura ou em profundidade.
- **Custos** das arestas são **distintos**.
 - Algoritmos continuam **corretos e eficientes** se isso não é verdade,
 - mas as provas ficam mais chatas,
 - basicamente porque não conseguimos usar
 - argumentos por contradição e precisamos usar
 - argumentos iterativos que são mais enrolados.

Um corte (A, B) é uma bipartição dos vértices do grafo,

- i.e., $A \cup B = V$ e $A \cap B = \emptyset$
- Uma aresta e cruza um corte se cada extremo está em uma parte distinta,
 - i.e., $e = \{u, v\}$ cruza (A, B) se $u \in A$ e $v \in B$ ou vice-versa

⇒ **Propriedade do corte:** Dado um grafo $G = (V, E)$, considere uma aresta $e \in E$

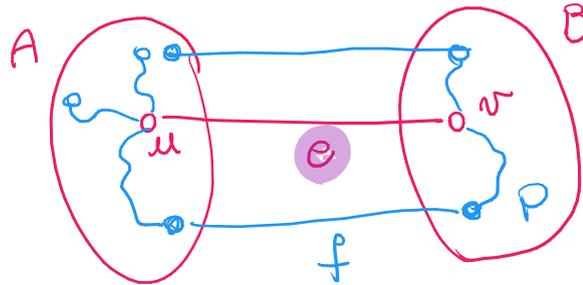
- Suponha que existe um corte (A, B) tal que
 - e é a aresta mais barata de G que cruza este corte.
- Então e está na árvore geradora mínima de G .



- Podemos falar DA árvore geradora mínima
 - porque supomos que não existem arestas com mesmo custo.

Demonstração da propriedade do corte

- ⇒ Suponha, por contradição, que a árvore geradora mínima T^* não usa
- o uma aresta e que é a mais barata que atravessa um corte (A, B)



- Note que, T^* precisa atravessar esse corte,
 - o já que ela conecta todos os vértices do grafo.
- Sejam u e v os extremos da aresta e , ou seja, $e = \{u, v\}$
 - o Assim, existe um caminho $P \subseteq T^*$ que conecta u e v
- Como u e v estão em partes diferentes do corte,
 - o pelo menos uma aresta $f \in P$ cruza o corte.
- Construa uma nova árvore T' fazendo uma troca, i.e., $T' = T^* \cup \{e\} \setminus \{f\}$
 - o T' continua conexa, pois $P \cup \{e\}$ forma um ciclo.
- Como e é a aresta de menor custo do corte (A, B) , temos $c(T') < c(T^*)$
 - o Portanto, a nova árvore T' é mais barata que T^* , contradição.

Algoritmo de Prim

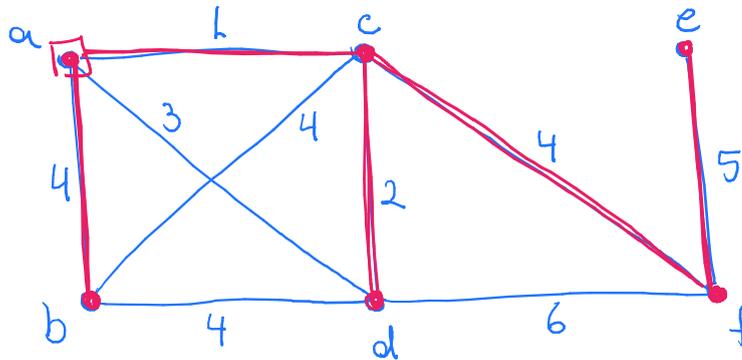
Técnica de projeto de algoritmos gulosos: iterativamente realizar

- escolhas míopes buscando atingir o ótimo.
- A propriedade do corte nos dá um critério para escolhas míopes.

Ideia para um algoritmo: dado um grafo ponderado $G = (V, E)$

- considere um corte (A, B) e seja e a aresta mais barata que crossa este corte.
 - Então e deve ser colocado na árvore geradora em construção.
- Podemos derivar diversos algoritmos que utilizam essa propriedade/ideia.
 - O primeiro que veremos é o algoritmo de Prim.

Exemplo do algoritmo de Prim:



Pseudocódigo do algoritmo de Prim:

Prim ($G = (V, E)$, custos c):

escolher 's' arbitrariamente como inicial

$X = \{s\}$ $T = \emptyset$

$O(m)$ iterações

enquanto $X \neq V$:

• seja $e = \{u, v\}$ a aresta de menor custo que atravessa $(X, V \setminus X)$
 $c/u \in X$ e $v \notin X$

$T = T \cup \{e\}$ -

$X = X \cup \{v\}$ -

devolva T

$\hookrightarrow O(m)$ arestas p/verificar

$n = |V|$

Implementação e eficiência do algoritmo de Prim:

- A implementação mais simples leva tempo $O(m \cdot m)$
 - o que não é trivial pois o número de árvores geradoras é imenso.
- Note que, assim como no algoritmo de Dijkstra,
 - estamos sucessivamente escolhendo elementos mínimos.
- Por isso, parece interessante usar um heap de mínimo na implementação.
- É curioso observar que a implementação com heap mais simples e eficiente
 - utiliza o heap para armazenar vértices, e não arestas.

Pseudocódigo do algoritmo de Prim usando heap de mínimo:

PrimHeap ($G=(V,E)$, custos c):

$X = \emptyset$

$T = \emptyset$

$O(m)$

para todo $v \in V$:

$d[v] = +\infty$

aresta[v] = Null

escolha um vértice s qualquer como inicial

$O(m)$

$d[s] = 0$

H = heap de mínimo com todo $v \in V$ e usando $d[v]$ como prioridade
enquanto $H \neq \emptyset$:

v = remove mínimo de H \Leftarrow no total $O(m \log n)$

$X = X \cup \{v\}$

$T = T \cup \text{aresta}[v]$

para cada aresta (v, w) :

se $c(v, w) < d[w]$:

$d[w] = c(v, w)$

aresta[w] = (v, w)

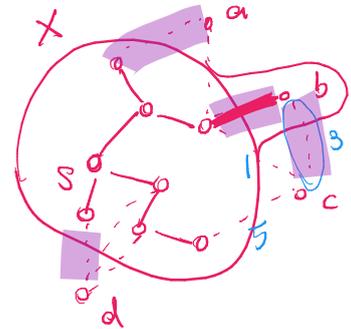
atualize a posição de w em H \Leftarrow no total

$O(m \log n)$

devolva T

no total $O(m)$ iterações

$O(m)$
iterrações



Análise de eficiência do algoritmo de Prim implementado com Heap:

- a inicialização leva tempo $O(n)$
- o laço principal é executado $O(n)$ vezes,
 - já que em cada iteração um vértice é removido do heap.
- Cada operação de remoção do heap custa $O(\log n)$
 - Assim, essas operações custam, no total, $O(n \log n)$
- Cada aresta do grafo pode desencadear
 - uma operação de atualização no heap, i.e., $O(n)$ atualizações
- Como cada atualização custa $O(\log n)$
 - no total essas operações custam $O(m \log n)$
- Portanto, a complexidade de tempo do algoritmo é $\Rightarrow O(m \log n)$
 - que é **quase linear** no tamanho do grafo.

Curiosidade: existe outra maneira eficiente de implementar este algoritmo

- sem usar funções para **atualizar** elementos internos do heap.
 - Quiz: Como fazer isso?
- Dica: envolve reinserir vértices no heap ao encontrar arestas de menor custo
 - e, ao remover um vértice, só considerá-lo se ele ainda não foi visitado.

Prova de corretude do algoritmo de Prim:

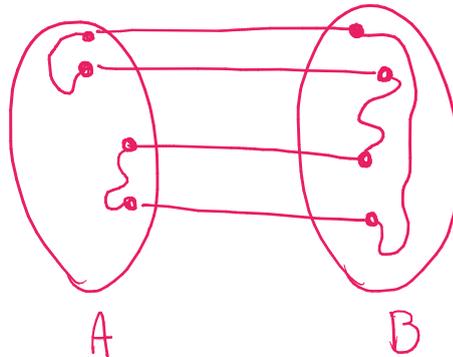
- Primeiro mostramos que ele produz uma árvore T que é geradora, ou seja
 1. T não possui ciclos
 2. T alcança todos os vértices do grafo
- Então mostramos que T é de custo mínimo, usando a propriedade do corte.

1) Começamos mostrando que T não possui ciclos.

- Para tanto basta mostrar que nenhuma aresta escolhida pelo algoritmo
 - pertence a algum ciclo.
- Existem resultados auxiliares que podem nos ajudar

Lema da Travessia Par - Se um corte (A, B) atravessa um ciclo C qualquer,

- então C tem um número par de arestas atravessando o corte.



Prova: Note que, se começarmos a percorrer o ciclo C

- cada vez que passamos por uma das arestas que atravessa o corte (A, B)
 - nós mudamos de parte, e só nesse caso mudamos de parte.
- Assim, após atravessarmos o corte C um número ímpar de vezes
 - estamos numa parte diferente da que começamos.
- Como o ciclo tem que fechar, precisamos atravessar
 - um número par dessas arestas para voltar ao começo.
- Portanto, o corte tem que atravessar um número par de arestas do ciclo C

Corolário da Aresta Solitária - Se e é a única aresta atravessando

- um certo corte (A, B) então e não pode pertencer a um ciclo.

Prova: Se e estivesse em um ciclo, pelo Lema da Travessia Par,

- ao menos mais uma aresta também teria que atravessar o corte (A, B)

Observe que toda aresta que o algoritmo escolhe é a única aresta

- que atravessa o corte definido pelos vértices $(X, V \setminus X)$
- Assim, usando o Corolário da Aresta Solitária temos que
 - as arestas escolhidas pelo algoritmo não formam ciclos.
- De fato, também precisamos mostrar que T é conexo.
 - Mas isso é fácil de verificar por indução, já que
 - toda aresta que o algoritmo escolhe tem uma ponta em X

2) Agora vamos mostrar que **T é geradora**, ou seja, alcança todos os vértices de G.

- Como em cada iteração o algoritmo busca arestas
 - que atravessam o corte $(X, V \setminus X)$
- para que ele deixe de alcançar algum vértice
 - deve ser o caso de existir algum corte vazio.
- Existe um resultado auxiliar que pode nos ajudar.

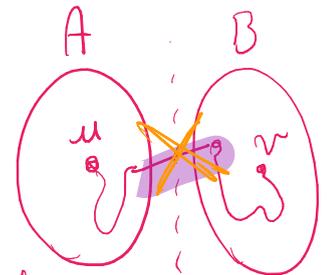
Lema do Corte Vazio - Um grafo não é conexo \iff existe um corte (A, B) que não é atravessado por arestas.

se, e somente se,

sem

Prova: (\leftarrow) Supondo que existe um corte (A, B) sem arestas,

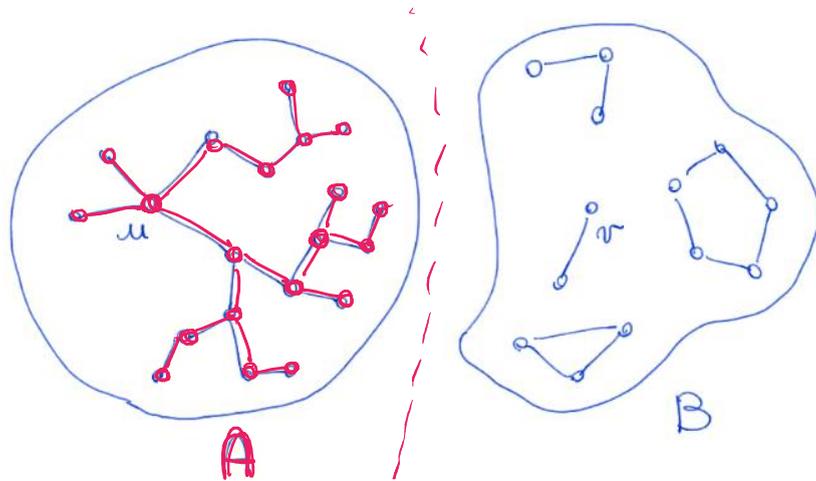
temos $u \in A$ e $v \in B$. Basta verificar que não existe caminho de u até v , caso contrário teríamos uma aresta atravessando (A, B) . Portanto, o grafo é desconexo



(\rightarrow) Supondo que o grafo não é conexo, seja u e v um par não conectado.

Seja A o conj. de todos os vértices alcançados a partir de u . Seja $B = V \setminus A$. Note que $v \in B$.

Assim, por construção (A, B) é um corte sem arestas criando sua fratura.



Como sabemos que o grafo $G = (V, E)$ é conexo, usando o **Lema do Corte Vazio**

- concluímos que ele não tem qualquer corte (A, B) sem arestas.
- Portanto, o algoritmo nunca vai parar no meio de uma iteração
 - por falta de arestas atravessando o corte $(X, V \setminus X)$

3) Finalmente mostramos que **T tem custo mínimo**.

Esse resultado sai direto da Propriedade do Corte, pois

- toda aresta escolhida pelo algoritmo é a mais barata de um corte $(X, V \setminus X)$
 - e, pela propriedade, deve estar na árvore geradora mínima.