

# Algoritmos

Pedro Hokama

## Fontes

- [cirs] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [timr] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms  
<https://www.youtube.com/playlist?list=PLXFMmk03Dt7Q0xr1PIAriY5623ckIH7V>  
<https://www.youtube.com/playlist?list=PLXFMmk03Dt5EMI2s2WQBsLsZ17A5HEK6>
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420  
Qualquer erro é de minha responsabilidade.

1 / 18

2 / 18

## Árvore Geradora Mínima: definição do problema

- Suponha que queiramos construir estradas para interligar  $n$  cidades, sendo que, a cada estrada entre duas cidades  $i$  e  $j$  que pode ser construída, há um custo de construção associado.
- *Como determinar eficientemente quais estradas devem ser construídas de forma a minimizar o custo total de interligação das cidades ?*
- Este problema pode ser modelado por um problema em grafos não orientados ponderados onde os vértices representam as cidades, as arestas representam as estradas que podem ser construídas e o peso de uma aresta representa o custo de construção da estrada.

Árvore Geradora Mínima

3 / 18

4 / 18

## Árvore Geradora Mínima: definição do problema

Nessa modelagem, o problema que queremos resolver é encontrar um **subgrafo gerador** (que contém todos os vértices do grafo original), **conexo** (para garantir a interligação de todas as cidades) e cuja soma dos custos de suas arestas seja a menor possível.

Mais formalmente, definimos o problema da seguinte forma:

### Problema da Árvore Geradora Mínima:

Dado um grafo não orientado ponderado  $G = (V, E)$ , onde  $w : E \rightarrow \mathbb{R}^+$  define os custos das arestas, encontrar um subgrafo gerador conexo  $T$  de  $G$  tal que, para todo subgrafo gerador conexo  $T'$  de  $G$

$$\sum_{e \in T} w_e \leq \sum_{e \in T'} w_e.$$

5 / 18

## Árvore Geradora Mínima

- Claramente, o problema só tem solução se  $G$  é conexo.
- **Portanto, a partir de agora, vamos supor que  $G$  é conexo.**
- Além disso, não é difícil ver que a solução para esse problema será sempre uma árvore: basta notar que  $T$  não conterá ciclos pois, caso contrário, poderíamos obter um outro subgrafo  $T'$ , ainda conexo e com custo menor ou igual ao de  $T$ , removendo uma aresta do ciclo.
- Portanto, dizemos que este problema de otimização é o problema de encontrar a *Árvore Geradora Mínima* (AGM) em  $G$ .
- *Como projetamos um algoritmo para resolver esse problema ?*

6 / 18

## Algoritmo para AGM

- Uma primeira abordagem exaustiva para solucionar o problema poderia ser enumerar todas as árvores geradoras do grafo, computar seus custos e retornar uma árvore de custo mínimo.
- No entanto, esse não é um bom algoritmo pois um grafo completo de  $n$  vértices possui um número **exponencial** ( $n^{n-2}$ ) de árvores geradoras.
- Para obter um algoritmo eficiente (polinomial !) devemos então procurar alguma propriedade do problema que nos permita restringir o espaço de possíveis soluções a ser analisado.

7 / 18

## Algoritmo genérico para AGMs

- Os dois algoritmos que serão vistos usam uma **estratégia gulosa**, diferindo apenas no modo em que esta é aplicada.
- A estratégia gulosa é resumida no *algoritmo genérico* mostrado a seguir, onde a AGM é construída aresta a aresta.
- A cada iteração do algoritmo é mantido um conjunto  $A$  de arestas que satisfaz à seguinte **invariante**:

Antes de cada iteração,  $A$  é um subconjunto de arestas de uma AGM.

- Em cada iteração, determina-se ainda uma aresta  $(u, v)$  tal que  $A \cup \{(u, v)\}$  que também satisfaz à invariante, i.e., está contido em alguma AGM. Tal aresta é dita ser **segura**.

8 / 18

## Algoritmo genérico para AGMs

### AGM-GENERICO( $G, w$ )

1.  $A \leftarrow \emptyset$ ;
2. **enquanto**  $A$  não forma uma árvore geradora **faça**
3.     encontre uma aresta **segura**  $(u, v)$ ;
4.      $A \leftarrow A \cup \{(u, v)\}$ ;
5. **retorne**  $A$ .

- A parte mais *engenhosa* do algoritmo é encontrar a aresta **segura** na linha 3.
- Esta aresta deve existir pois, por hipótese, no laço das linhas 2–4,  $A$  é um subconjunto **próprio** de uma árvore geradora  $T$ . Logo existe uma aresta segura  $(u, v) \in T - A$ .

9 / 18

## Árvores geradoras: reconhecendo arestas seguras

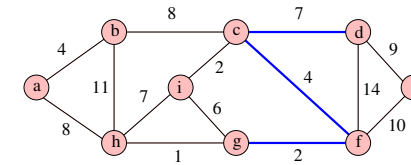
### Notação:

Considere um grafo não orientado  $G = (V, E)$  e tome  $S \subseteq V$ . O conjunto  $V - S$  é denotado por  $\bar{S}$ .

### Definição:

O *corte de arestas* de  $S$ , denotado por  $\delta(S)$ , é o conjunto de arestas de  $G$  com um extremo em  $S$  e outro em  $\bar{S}$ .

**Exemplo:**  $S = \{a, b, c, g, h, i\}$ .



10 / 18

## Árvores geradoras: reconhecendo arestas seguras

Diz-se que um corte  $\delta(S)$  **respeita** um conjunto de arestas  $A$  se  $A \cap \delta(S)$  é vazio.

Diz-se que  $(u, v)$  é uma **aresta leve** de um corte  $\delta(S)$  se  $w_{uv} := \min_{(x,y) \in \delta(S)} \{w_{xy}\}$ , i.e.,  $(u, v)$  é a aresta de menor peso no corte  $\delta(S)$ .

### Teorema 23.1: (Cormen 2ed)

Seja  $G = (V, E)$  um grafo conexo não orientado ponderado nas arestas por uma função  $w : E \rightarrow \mathbb{R}$ . Seja  $A$  um subconjunto de  $E$  que está contido em alguma AGM de  $G$ , seja ainda  $\delta(S)$  um corte de  $G$  que *respeita*  $A$  e  $(u, v)$  uma *aresta leve* de  $\delta(S)$ . Então,  $(u, v)$  é uma *aresta segura* para  $A$ .

11 / 18

## Árvores geradoras: reconhecendo arestas seguras

O Teorema anterior deixa claro o funcionamento do algoritmo genérico para AGM. À medida que o algoritmo avança, o grafo induzido por  $A$  é acíclico (pois pertence à uma árvore). Ou seja  $G_A = (V, A)$  é uma floresta, sendo que algumas árvores podem ter um único vértice. Cada aresta segura  $(u, v)$  conecta duas componentes distintas de  $G_A$ .

Inicialmente a floresta tem  $|V|$  árvores formadas por vértices isolados. Cada iteração reduz o número de componentes da floresta de uma unidade. Portanto, após a inclusão de  $|V| - 1$  arestas seguras, o algoritmo termina com uma única árvore.

12 / 18

## Árvores geradoras: reconhecendo arestas seguras

### Corolário 23.2: (Cormen 2ed)

Seja  $G = (V, E)$  um grafo conexo não orientado ponderado nas arestas por uma função  $w : E \rightarrow \mathbb{R}$ . Seja  $A$  um subconjunto de  $E$  que está contido em alguma AGM de  $G$ , e seja  $C = (V_C, E_C)$  uma componente conexa (árvore) da floresta  $G_A = (V, A)$ . Se  $(u, v)$  é uma **aresta leve** de  $\delta(C)$ , então  $(u, v)$  é **segura** para  $A$ .

13 / 18

## Os algoritmos de Prim e de Kruskal

- Os algoritmos de Prim e de Kruskal especializam o algoritmo genérico para AGM visto anteriormente, fazendo uso Corolário 23.3.
- No **algoritmo de Prim**, o conjunto de arestas  $A$  é sempre uma **árvore**. A **aresta segura** adicionada a  $A$  é sempre uma **aresta leve** do corte  $\delta(C)$ , onde  $C$  é o conjunto de vértices que são extremidades de arestas em  $A$ .
- No **algoritmo de Kruskal**, o conjunto de arestas  $A$  é uma **floresta**. A **aresta segura** adicionada a  $A$  é sempre a aresta de menor peso dentre todas as arestas ligando dois vértices de componentes distintas de  $G_A$ .

14 / 18

## O algoritmo de Prim

### PRIM( $G, w, r$ )

▷  $r$  é um vértice escolhido para ser *raiz* da AGM

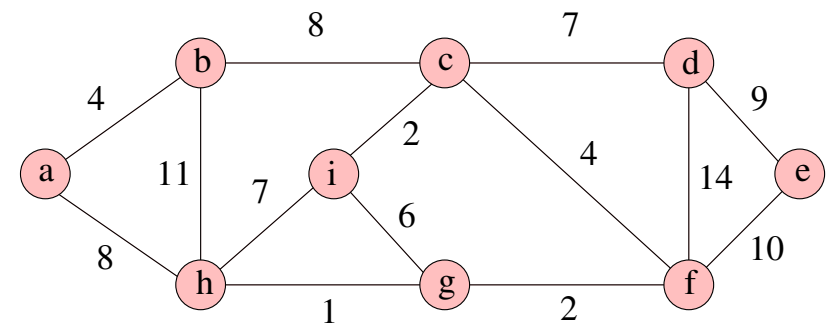
$Q$  é o conjunto de vértices a incluir na árvore

$\text{dist}[x]$ : peso da aresta mais leve ligando  $x$  a componente de  $r$

- $\text{dist}[r] \leftarrow 0; \quad Q \leftarrow V;$
- para** todo  $v \in V - \{r\}$  **faça**  $\text{dist}[v] \leftarrow \infty;$
- $\text{pred}[r] \leftarrow \text{nulo};$
- $A \leftarrow \{\};$    ▷ inicializa o conjunto de arestas da AGM
- $W \leftarrow 0;$    ▷ inicializa o peso da AGM
- enquanto** ( $Q$  não for vazio) **faça**
- Remover de  $Q$  o vértice  $u$  com o menor valor em  $\text{dist}$ ;
- $W \leftarrow W + \text{dist}[u];$
- se**  $\text{pred}[u] \neq \text{nulo}$ ,  $A \leftarrow A \cup \{(\text{pred}[u], u)\};$   
  ▷ atualiza o valor de  $\text{dist}[\cdot]$  para vértices adjacentes a  $u$
- para** todo  $v \in \text{Adj}[u]$  **faça**
- se** ( $v \in Q$ ) e ( $\text{dist}[v] > w[u, v]$ ) **então**
- $\{ \text{dist}[v] \leftarrow w[u, v]; \quad \text{pred}[v] \leftarrow u; \}$
- retorne** ( $A, W$ ).

15 / 18

## O algoritmo de Prim: exemplo



16 / 18

## O algoritmo de Prim: corretude

O algoritmo de Prim mantém a seguinte invariante, a qual é válida antes de cada execução do laço das linhas 6 a 12:

- 1  $A = \{(v, \text{pred}[v]) : v \in V - \{r\} - Q\}$ .
- 2 Os vértices já colocados na AGM são aqueles em  $V - Q$ .
- 3 Para todos os vértices em  $Q$ , se  $\text{pred}[v] \neq \text{nulo}$ , então  $\text{dist}[v] < \infty$  e  $\text{dist}[v]$  é o valor da aresta leve  $(v, \text{pred}[v])$  que conecta  $v$  a algum vértice já pertencente a AGM.

## O algoritmo de Prim: complexidade

- A complexidade depende de como o conjunto  $Q$  é *implementado*.
- Se  $Q$  for implementado como um **vetor simples** ( $Q$  poderia estar representado pelo próprio vetor  $\text{dist}$ ), a complexidade do algoritmo será  $O(|V|^2 + |E|) \equiv O(|V|^2)$ .
- Se  $Q$  for implementado como um **heap** ( $Q$  poderia estar representado pelo próprio vetor  $\text{dist}$ ), a complexidade do algoritmo será  $O((|V| + |E|) \log |V|) \equiv O(|E| \log |V|)$ .
- Portanto, para grafos **densos** ( $|E| \in O(|V|^2)$ ), a melhor alternativa é implementar  $Q$  como um vetor simples, enquanto que para grafos **esparsos** ( $|E| \in O(|V|)$ ), deve-se optar pela implementação de  $Q$  como uma **fila de prioridades**.
- Note que, com esta última opção, o algoritmo de Prim assemelha-se muito a uma **busca em largura**. **A diferença entre os algoritmos fica praticamente restrita à troca de uma fila simples por uma fila de prioridades!**