

Projeto e Análise de Algoritmos

Flávio L. C. de Moura

22 de junho de 2023

Nesta aula estudaremos o problema de determinar a árvore geradora mínima (*minimum spanning tree*) de um grafo G [?]. Este problema possui diversas aplicações, como na construção de conexão em redes de computadores ou conexão viária entre diversos pontos de uma cidade. Formalmente, uma árvore geradora é definida como a seguir:

Definição 1. *Sejam $G = (V, E)$ um grafo e G' um subgrafo de G que seja uma árvore. Dizemos que G' é uma árvore geradora (spanning tree) de G se contém todos os vértices de G .*

Sabemos que árvores são conexas, e portanto se o grafo G possui árvore geradora então G é necessariamente conexo. Reciprocamente, se G é um grafo conexo então G possui pelo menos uma árvore geradora.

Definição 2. *Seja $G = (V, E)$ um grafo com função peso w . Uma árvore geradora mínima (Minimum Spanning Tree (MST)) de G é uma árvore geradora de custo mínimo, isto é, tal que a soma dos pesos de suas arestas é mínimo.*

Assim, G' é uma árvore geradora mínima do grafo G se contém todos os vértices de G e nenhuma outra árvore geradora de G possui custo estritamente menor do que G' .

Quantas árvores geradoras um qualquer grafo possui?

Teorema 3 (Cayley, 1889). *Existem V^{V-2} árvores geradoras em um grafo completo com V vértices.*

Portanto, utilizar um algoritmo força bruta para obter uma árvore geradora mínima não é uma boa ideia. Inicialmente, definiremos um método genérico que manipula o conjunto $A \subseteq E$ de arestas baseado na seguinte invariante:

Antes de cada iteração, o conjunto A é um subconjunto de alguma árvore geradora mínima de G .

A ideia é adicionar novas arestas ao conjunto A de forma a não violar a invariante acima. Chamaremos de *seguras* (*safe*) as arestas que podem ser adicionadas ao conjunto A sem violar a invariante. Desta forma, o procedimento genérico para construir árvores geradoras mínimas em grafos conexos é dado como a seguir:

```
1  $A = \emptyset$ ;  
2 while  $A$  does not form a spanning tree do  
3   | find a safe edge  $(u, v)$  for  $A$ ;  
4   |  $A = A \cup \{(u, v)\}$ ;  
5 end  
6 return  $A$ ;
```

Algoritmo 1: Generic-MST(G, w)

Como identificar uma aresta segura? A seguir veremos uma regra para reconhecer arestas seguras.

Definição 4. *Um corte $(S, V - S)$ de um grafo $G(V, E)$ é uma partição de V . Dizemos que a aresta (u, v) cruza o corte, se $u \in S$ e $v \in V - S$. Dizemos que o corte respeita o conjunto A de arestas se nenhuma aresta de A cruza o corte. Uma aresta que cruza um corte é dita leve se tem peso mínimo dentre todas as arestas que cruzam o corte.*

Teorema 5. *Sejam $G = (V, E)$ um grafo conexo com função peso w , e $A \subseteq E$ um conjunto contido em alguma árvore geradora mínima de G . Se $(S, V - S)$ é um corte de G que respeita A , e (u, v) é uma aresta leve que cruza o corte $(S, V - S)$, então (u, v) é segura para A .*

Corolário 6. *Sejam $G = (V, E)$ um grafo conexo com função peso w , $A \subseteq E$ um conjunto contido em alguma árvore geradora mínima de G , e $C = (V_C, E_C)$ uma componente conexa na floresta $G_A = (V, A)$. Se (u, v) é uma aresta leve que conecta C a outra componente em G_A então (u, v) é segura para A .*

Exercício 7. *Seja (u, v) uma aresta de peso mínimo em um grafo conexo G . Mostre que (u, v) pertence a alguma árvore geradora mínima de G .*

1 O algoritmo de Prim

O algoritmo de Prim (Robert C. Prim, 1957) consiste em uma especialização do algoritmo genérico dado acima. Para construir uma árvore geradora mínima de um grafo G conexo com função peso w , o algoritmo vai partir de um vértice r dado como entrada. A árvore então será desenvolvida a partir de r , que chamaremos de raiz da árvore geradora mínima. A ideia é que em cada passo, o algoritmo vai adicionar uma nova aresta leve à árvore construída a partir de r . Pelo Corolário 6, temos que cada aresta adicionada à árvore é segura. O algoritmo de Prim é classificado como algoritmo guloso porque em cada passo a aresta que é adicionada à árvore é a que tem menor peso dentre as que podem ser selecionadas. Uma implementação eficiente do algoritmo de Prim necessita de uma forma rápida para selecionar uma aresta segura a ser inserida na árvore construída até aquele momento. Para isto, todos os vértices que ainda não fazem parte da árvore em construção são armazenados em uma fila de prioridade, onde o atributo *key* de cada vértice corresponde à sua prioridade. Uma maneira eficiente de implementar filas de prioridade é utilizando *heaps*. No caso de um *heap* de máximo (resp. mínimo) teremos uma fila de prioridade de máximo (resp. mínimo). No caso do algoritmo de Prim utilizaremos filas de prioridade de mínimo que possuem as seguintes operações:

- $\text{Insert}(S, x)$: insere o elemento x no conjunto S , o que equivale à operação $S = S \cup \{x\}$;
- $\text{Minimum}(S)$: retorna o elemento de S com a maior prioridade, ou seja, o elemento de S que possui o menor valor de *key*;
- $\text{Extract-Min}(S)$: remove e retorna o elemento de S que possui o menor valor de *key*;
- $\text{Decrease-Key}(S, x, k)$: decrementa o valor de *key* do elemento x de S para o novo valor k , que deve ser menor ou igual a $x.\text{key}$.

Em particular, estamos interessados nas operações Extract-Min e Decrease-Key dadas a seguir.

```

1 if  $S.\text{heap\_size} < 1$  then
2   | error "heap underflow";
3 end
4  $\text{min} \leftarrow S[1]$ ;
5  $S[1] \leftarrow S[S.\text{heap\_size}]$ ;
6  $S.\text{heap\_size} \leftarrow S.\text{heap\_size} - 1$ ;
7  $\text{Min-Heapify}(S, 1)$ ;
8 return  $\text{min}$ ;

```

Algoritmo 2: $\text{Extract-Min}(S)$

onde Min-Heapify é dada por:

```

1  $l \leftarrow 2i$ ;
2  $r \leftarrow 2i + 1$ ;
3 if  $l \leq S.heap\_size$  and  $S[l] < S[i]$  then
4   |  $smallest \leftarrow l$ ;
5 end
6 else
7   |  $smallest \leftarrow i$ ;
8 end
9 if  $r \leq S.heap\_size$  and  $S[r] < S[smallest]$  then
10  |  $smallest \leftarrow r$ ;
11 end
12 if  $smallest \neq i$  then
13  | exchange  $S[i]$  with  $S[smallest]$ ;
14  | Min-Heapify( $S, smallest$ );
15 end

```

Algoritmo 3: Min-Heapify(S, i)

A complexidade de Min-Heapify é obtida a partir da recorrência $T(n) \leq T(2n/3) + O(1)$ que tem solução $O(\lg n)$ (Veja a aula sobre o algoritmo *heapsort*). Assim, a complexidade de Extract-Min é também $O(\lg n)$. A operação Decrease-Key é dada por:

```

1 if  $k > S[i]$  then
2   | error "new key is larger than current key";
3 end
4  $S[i] \leftarrow k$ ;
5 while  $i > 1$  and  $S[Parent(i)] > S[i]$  do
6   | exchange  $S[i]$  with  $S[Parent(i)]$ ;
7   |  $i \leftarrow Parent(i)$ ;
8 end

```

Algoritmo 4: Decrease-Key(S, i, k)

e tem complexidade $O(\lg n)$ que corresponde ao comprimento máximo da distância entre o elemento que teve sua prioridade alterada (linha 4), e a raiz do *heap*. O algoritmo Insert é dado por:

```

1  $S.heap\_size \leftarrow S.heap\_size + 1$ ;
2  $S[S.heap\_size] \leftarrow \infty$ ;
3 Decrease-Key( $S, S.heap\_size, key$ );

```

Algoritmo 5: Insert(S, key)

O algoritmo de Prim é dado como a seguir:

```

1 for each vertex  $u \in G.V$  do
2   |  $u.key \leftarrow \infty$ ;
3   |  $u.\pi \leftarrow NIL$ ;
4 end
5  $r.key \leftarrow 0$ ;
6  $Q \leftarrow \emptyset$ ; // Inicializa uma fila vazia
7 for each vertex  $u \in G.V$  do
8   |  $Insert(Q, u)$ ;
9 end
10 while  $Q \neq \emptyset$  do // Adiciona  $u$  à árvore
11   |  $u \leftarrow Extract-Min(Q)$ ;
12   | for each  $v \in G.Adj[u]$  do
13     | if  $v \in Q$  and  $w(u, v) < v.key$  then
14       |   |  $v.\pi \leftarrow u$ ;
15       |   |  $v.key \leftarrow w(u, v)$ ;
16       |   |  $Decrease-Key(Q, v, w(u, v))$ ;
17     |   | end
18   |   | end
19 end

```

Algoritmo 6: MST-Prim(G, w, r)

Agora conclua que a complexidade do algoritmo de Prim é $O((V + E) \lg V) = O(E \lg V)$, pois $E \geq V - 1$ em um grafo conexo.

Por fim, a correção do algoritmo de Prim pode ser estabelecida pelo seguinte teorema:

Teorema 8. *Seja $G = (V, E)$ um grafo conexo com função peso w , e $r \in V$. Após a execução de MST-Prim(G, w, r), o subgrafo $G' = (V', E')$ com $V' = \{v \in V : v.\pi \neq NIL\} \cup \{r\}$ e $E' = \{(v.\pi, v) : v \in V' - \{r\}\}$ é uma árvore geradora mínima de G .*