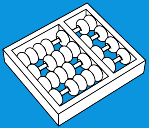
Algoritmos de Prim e Kruskal

MC558 - Projeto e Análise de Algoritmos II

09/24

09



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

"Encontrar todas as árvores geradoras mínimas é não trivial pois o número de soluções pode ser exponencial."

João Guilherme Martinez, Rosiane de Freitas e Altigran Silva

Algoritmo de Prim

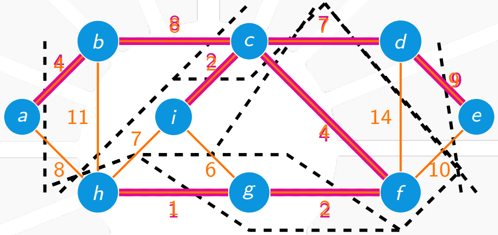
Ideia

- Escolhemos um vértice **r** arbitrariamente no início.
- O conjunto **A** são as arestas de uma árvore com raiz **r**.
- O conjunto **S** são os vértices dessa árvore.
- Em cada iteração, adicionamos uma **ARESTA LEVE** de $\delta(\mathbf{S})$.

Detalhe de implementação importante:

- Como encontrar essa aresta leve **EFICIENTEMENTE**?

Exemplo



Estruturas de dados

Como representar os vértices a serem adicionados?

- Mantemos uma fila de prioridade (de mínimo) Q.
- Ela contém todos vértices que **NÃO** estão na árvore.
- Cada vértice **v** na fila tem prioridade key[**v**] de ser inserido.

Qual a prioridade de escolher um vértice **v**?

- key[**v**] guarda o peso da menor aresta ligando **v** à árvore, ou vale ∞ se não houver uma tal aresta.

Como representar a árvore sendo construída?

- Mantemos um vetor π de pais de todos vértices.
- Os vértices da árvore são: $\mathbf{S} = \mathbf{V} \setminus \mathbf{Q}$.
- As arestas da árvore são: $\mathbf{A} = \{(\mathbf{u}, \pi[\mathbf{u}]) : \mathbf{u} \in \mathbf{S} \setminus \{\mathbf{r}\}\}$.

O algoritmo

Algoritmo: $\textsc{AGM-Prim}(G, w, r)$

para cada $u \in V[G]$
    $\mathrm{key}[u] \leftarrow \infty$
    $\pi[u] \leftarrow \mathrm{NIL}$
$\mathrm{key}[r] \leftarrow 0$
$Q \leftarrow V[G]$
enquanto $Q \neq \emptyset$
    $u \leftarrow \textsc{Extract-Min}(Q)$
    para cada $v \in \mathrm{Adj}[u]$
        se $v \in Q$ **e** $w(u, v) < \mathrm{key}[v]$
            $\pi[v] \leftarrow u$
            $\mathrm{key}[v] \leftarrow w(u, v)$

Correção do algoritmo

Teorema (Invariantes)

Considere a execução no início do laço **enquanto** *e defina* $\mathbf{S} = \mathbf{V} \setminus \mathbf{Q}$ *e* $\mathbf{A} = \{(\mathbf{u}, \pi[\mathbf{u}]) : \mathbf{u} \in \mathbf{S} \setminus \{\mathbf{r}\}\}$, *então:*

1. $\boldsymbol{A}$ *contém arestas de uma árvore* T *com vértices* $\boldsymbol{S}$ *e raiz* $\boldsymbol{r}$.
2. *Para cada* $\boldsymbol{v} \in \boldsymbol{Q}$:
   - *Se* $\pi[\boldsymbol{v}] \neq \text{NIL}$, *então* $\text{key}[\boldsymbol{v}]$ *é o peso de uma aresta com menor peso ligando* $\boldsymbol{v}$ *a algum vértice de* T.
   - *Se* $\pi[\boldsymbol{v}] = \text{NIL}$, *então não existe aresta ligando* $\boldsymbol{v}$ *a algum vértice de* T.

- As invariantes implicam que no início da iteração do laço, $(\mathbf{u}, \pi[\mathbf{u}])$ é uma **ARESTA SEGURA**.
- Portanto, o algoritmo está correto.

Complexidade

A complexidade depende da fila de prioridade Q:

- Cada teste $v \in Q$ (linha 9) leva tempo constante (por quê?).
- Vamos contar quantas vezes executamos cada operação:
  - INSERT é executada $|V|$ vezes (linhas 1–5).
  - EXTRACT-MIN é executada $|V|$ vezes (linha 6).
  - DECREASE-KEY é executada até $|E|$ vezes (linha 11).

Portanto, o **TEMPO TOTAL** de execução é:

$$O(V) \cdot \textsc{Insert} + O(V) \cdot \textsc{Extract-Min} + O(E) \cdot \textsc{Decrease-Key}.$$

Complexidade usando min-heap

Se implementarmos Q como um min-heap, então:

- INSERT consome tempo $O(\log V)$.
- EXTRACT-MIN consome tempo $O(\log V)$.
- DECREASE-KEY consome tempo $O(\log V)$.

Então, o tempo total será:

$$O(V \log V + V \log V + E \log V) = O(E \log V).$$

Observações:

- Podemos inicializar o min-heap em tempo $O(V)$.
- Usamos $V = O(E)$ pois sabemos que G é conexo.

Análise amortizada

Refletindo sobre como analisamos uma operação:

- Supomos que todas as chamadas levam o mesmo tempo.
- Consideramos **SEMPRE** o tempo de pior caso.
- Na prática, o tempo de uma chamada pode ser bem menor.

Custo amortizado:

- Considere uma estrutura de dados abstrata S.
- Suponha que podemos realizar uma operação $p(S)$.
- Pode haver operações distintas (inserir, remover, etc.).
- Se executamos essas operações diversas vezes:
- Quanto tempo leva cada chamada em **MÉDIA**?

Análise amortizada

Ideia da análise amortizada:

- Suponha que na execução fazemos m chamadas a $p(S)$.
- Que o **TEMPO TOTAL** das operações p é $T(n)$.
- Então, o custo amortizado de p é $T(n)/m$.

Exemplo:

- se $T(n) = 4n$ e $m = 2n$, então o custo amortizado é 2.
- Isso **NÃO** significa que a operação leva tempo constante.
- Apenas que em média o tempo gasto por p é constante.

Revisitando a complexidade de Prim

Um **HEAP DE FIBONACCI** é uma estrutura de dados que:

- É utilizada para guardar um conjunto de $|V|$ elementos.
- Implementa as operações de fila de prioridade:
  - EXTRACT-MIN – tempo $O(\log V)$
  - DECREASE-KEY – tempo amortizado $O(1)$
  - INSERT – tempo amortizado $O(1)$
- Além de outras operações como UNION, etc.

Se usarmos um heap de Fibonacci para implementar Q:

- O tempo total melhora para $O(V + E + V \log V) = O(E + V \log V)$.
- Na prática, a implementação com min-heap é melhor.

O algoritmo de Kruskal

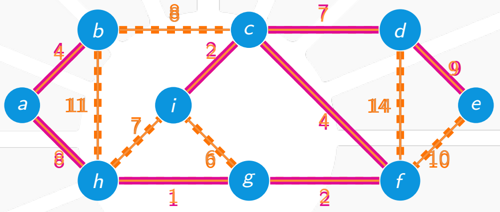
Ideia

- O subgrafo $G_A = (\mathbf{V}, \mathbf{A})$ é uma floresta.
- Consideramos cada uma das arestas em ordem de peso.
- Em cada iteração, adicionamos uma aresta **(u,v)** se ela ligar duas componentes distintas **C, C'** da floresta.
- Note que **(u,v)** é uma **ARESTA LEVE** de $\delta(\mathbf{C})$

Detalhe de implementação importante:

- Como saber se **(u,v)** liga componentes distintas eficientemente?

Exemplo



O algoritmo

Algoritmo: $\text{AGM-K}\textsc{ruskal}(G, w)$

1 $A \leftarrow \emptyset$
2 ordene as arestas em ordem não decrescente de peso
3 **para cada** $(u, v) \in E[G]$ *na ordem obtida*
4   **se** *u e v estão em componentes distintas de* (V, A)
5         $A \leftarrow A \cup \{(u, v)\}$
6 **devolva** A

Esta é uma versão preliminar do algoritmo:

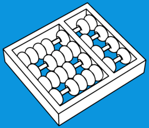
- Falta detalhar a implementação da linha 4.
- Como fazer isso **EFICIENTEMENTE**?

Algoritmos de Prim e Kruskal

MC558 - Projeto e Análise de Algoritmos II

09

09/24



UNICAMP

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br