Teoria dos Grafos
Árvore Geradora Mínima

Alexandre Romanelli, Gabriel Dardengo

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

11 de junho de 2018

Estrutura da apresentação

Definições – Árvore Geradora

Dado um grafo $G = (V, E)$, uma **árvore geradora** é um subgrafo $T_i = (V, E_i)$, com $E_i \subseteq E$ e T_i é conexo e acíclico.

O custo C_i da árvore geradora T_i é:

$$C_i = \sum_{e \in E_i} custo(e)$$

Definições – Árvore Geradora Mínima

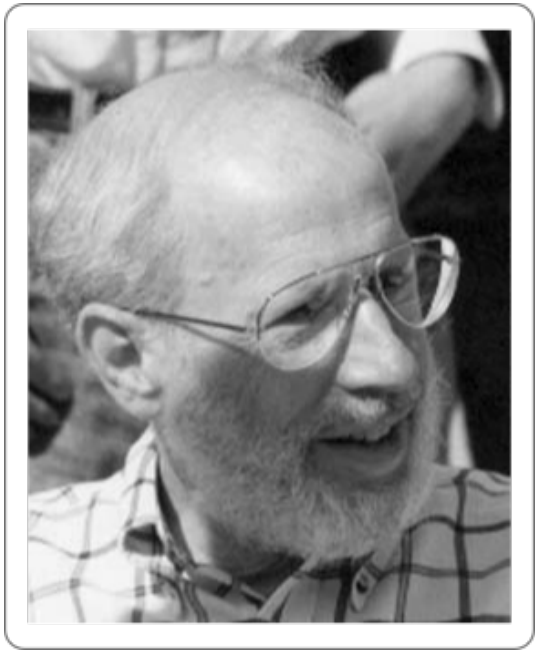
Dado o conjunto de árvores geradoras de um grafo G:

$$\mathcal{T} = \{T_i \mid T_i \text{ é uma árvore geradora de } G\}$$

Uma **árvore geradora mínima** de um grafo G é definida como abaixo:

$$T_{\text{min}} = T_i \in \mathcal{T} \mid \forall T_j \in \mathcal{T}, C_i \leq C_j$$

Algoritmo de Kruskal



Joseph Bernard Kruskal, Jr.

O algoritmo proposto em [Kruskal, 1956] encontra uma árvore geradora mínima para um grafo G com o seguinte método guloso:

- Inicialmente, as arestas de G são ordenadas por seus custos (ordem não-decrescente).
- É feito um percurso na lista ordenada de arestas. Para cada aresta, é verificado se esta pode ser incluída na solução, sem gerar ciclo.
- Se puder, a aresta é incluída na solução.

Algoritmo de Kruskal – Pseudocódigo

Algoritmo 1: Árvore Geradora Mínima por Kruskal.

entrada: grafo $G = (V, E)$
saída : árvore geradora mínima T

```
begin
    T ← ∅
    Gerar-Conjuntos-Disjuntos(V)
    L ← arestas de E em ordem não-decrescente de custo
    for i de 1 até |E| do
        (u, v) ← L.obterElemento(i)
        if Conjunto-Que-Contém(u) ≠ Conjunto-Que-Contém(v) then
            T ← T ∪ {(u, v)}
            Unir-Conjuntos-Que-Contém(u, v)
        end
    end
    return T
end
```

Fonte: adaptado de [Cormen et al., 2009].

Implementação do algoritmo de Kruskal

A implementação do algoritmo de Kruskal foi feita na linguagem C, com atenção aos seguintes termos:

- Os métodos *Gerar-Conjuntos-Disjuntos*, *Conjunto-Que-Contém* e *Unir-Conjuntos-Que-Contém* usam estrutura de conjuntos disjuntos, que foram testados com duas variantes:
  - "*Linked Lists*" Cada conjunto tem um número e é uma lista encadeada de itens. Cada item representa uma aresta e possui um número de conjunto. A união requer correção dos itens de um conjunto.
  - "*Rooted Trees*" Cada conjunto é uma árvore com raiz. Cada item da árvore representa uma aresta e aponta para o seu "pai", exceto a raiz, que aponta para si. Os itens têm também informação da altura da sua ramificação. A união de conjuntos equivale a fazer a raiz de uma árvore apontar para a raiz da outra. Uma operação de compactação é feita a cada consulta.

  Nos dois casos, há um vetor que associa arestas a itens da estrutura.

Implementação do algoritmo de Kruskal

- A ordenação das arestas de E foi feita com o algoritmo *Quick-sort*, que não requer memória adicional (ordenação "*in place*") e o "tempo" esperado de execução é $\Theta(n \log n)$ [Cormen et al., 2009].

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Figura: Algoritmos *Quick-Sort* e *Partition*. Fonte: [Cormen et al., 2009].

Complexidade

```
begin
    T ← ∅
    Gerar-Conjuntos-Disjuntos(V)
    L ← arestas de E em ordem não-decrescente de custo
    for i de 1 até |E| do
        (u, v) ← L.obterElemento(i)
        if Conjunto-Que-Contém(u) ≠ Conjunto-Que-Contém(v) then
            T ← T ∪ {(u, v)}
            Unir-Conjuntos-Que-Contém(u, v)
        end
    end
    return T
end
```

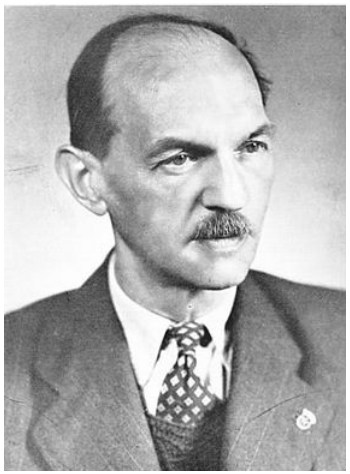
- Com *rooted trees* para representar conjuntos disjuntos, implementando união por ranqueamento e compressão de caminhos, as operações *Conjunto-Que-Contém*$(\bullet)$ e *Unir-Conjuntos-Que-Contém*$(\bullet, \bullet)$ têm complexidades de tempo:

$$O(m\,\alpha(|V|)) \ ,$$

  $\alpha(\bullet)$ é uma função de crescimento bem lento e m é o número de operações realizadas.
- O número de operações de união é igual a $|V| - 1$, portanto, a complexidade de tempo do algoritmo é $O(|E| \times |V|)$ (*considerando-se* $\alpha(|V|)$ *como uma constante*).

Algoritmo de Prim

O algoritmo de Prim é um algoritmo guloso (*greedy algorithm*) empregado para encontrar uma árvore geradora mínima (*minimal spanning tree*) num grafo conectado, valorado e não direcionado. Isso significa que o algoritmo encontra um subgrafo do grafo original no qual a soma total das arestas é minimizada e todos os vértices estão interligados.
O algoritmo foi desenvolvido em 1930 pelo matemático Vojtěch Jarník e depois pelo cientista da computação Robert Clay Prim em 1957, e redescoberto por Edsger Dijkstra em 1959.



(a) Vojtěch Jarník

(b) Robert Clay Prim

Algoritmo de Prim – Pseudocódigo

Algoritmo 2: Árvore Geradora Mínima por Prim.

```
begin
    s ← seleciona-um-elemento(G.V)
    foreach v ∈ G.V do
        π[v] ← nulo
    end
    Q ← {(0, s)}, S ← ∅
    while Q ≠ ∅ do
        v ← Extrair-Min(Q), S ← S ∪ {v}
        foreach u adjacente a v do
            if u ∈ S e Peso-Aresta(π[u]-u) > Peso-Aresta(v-u) then
                Q ← Q \ {(Peso-Aresta(π[u]-u), u)}
                Q ← Q ∪ {(Peso-Aresta(v-u), u)}
                π[u] ← v
            end
        end
    end
    return {(π[v], v) | v ∈ G.V e π[v] ≠ nulo}
end
```

Algoritmo de Prim – Complexidade

- $O(|V|^2)$ – com matriz de adjacência
- $O(|E| \log |V|)$ – com lista de adjacência/minHeap

Testes realizados

As implementações dos algoritmos de Kruskal e Prim foram testadas com as seguintes instâncias, obtidas de [Demetrescu et al., 2006]:

- USA-road-d.BAY e USA-road-t.BAY: 321.270 vértices e 800.172 arestas;
- USA-road-d.COL e USA-road-t.COL: 435.666 vértices e 1.057.066 arestas;
- USA-road-d.FLA e USA-road-t.FLA: 1.070.376 vértices e 2.712.798 arestas;
- USA-road-d.NY e USA-road-t.NY: 264.346 vértices e 733.846 arestas.

Resultados obtidos

Instância	|V|	|E|	Estrutura	Custo	Tempo de execução
USA-road-d.NY.gr	264346	733846	Rooted Trees	261159288	0.648898
			Linked Lists	261159288	6.889646
USA-road-d.FLA.gr	1070376	2712798	Rooted Trees	1806814846	6.081004
			Linked Lists	1806814846	108.819873
USA-road-d.COL.gr	435666	1057066	Rooted Trees	1323900090	0.608786
			Linked Lists	1323900090	19.971446
USA-road-d.BAY.gr	321270	800172	Rooted Trees	435798417	0.661261
			Linked Lists	435798417	7.833997
USA-road-t.NY.gr	264346	733846	Rooted Trees	628527136	0.500802
			Linked Lists	628527136	17.201385
USA-road-t.FLA.gr	1070376	2712798	Rooted Trees	4246472272	4.720503
			Linked Lists	4246472272	235.435439
USA-road-t.COL.gr	435666	1057066	Rooted Trees	3191475536	0.532046
			Linked Lists	3191475536	40.936742
USA-road-t.BAY.gr	321270	800172	Rooted Trees	1057008586	0.652855
			Linked Lists	1057008586	17.895229

Figura: Resultados obtidos com o algoritmo de Kruskal.

Resultados obtidos

Instância	|V|	|E|	Estrutura	Custo	Tempo de execução
USA-road-d.NY.gr	264346	733846	minHeap	261159288	0.328125
USA-road-d.FLA.gr	1070376	2712798	minHeap	1806814846	1.593750
USA-road-d.COL.gr	435666	1057066	minHeap	1323900090	0.531250
USA-road-d.BAY.gr	321270	1057066	minHeap	435798417	0.359375
USA-road-t.NY.gr	264346	733846	minHeap	628527136	0.328125
USA-road-t.FLA.gr	1070376	2712798	minHeap	4246472272	1.546875
USA-road-t.COL.gr	435666	1057066	minHeap	3191475536	0.562500
USA-road-t.BAY.gr	321270	800172	minHeap	1057008586	0.375000

Figura: Resultados obtidos com o algoritmo de Prim.

Referências

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009).
Introduction to Algorithms.
Computer science. MIT Press.

Demetrescu, C., Goldberg, A., and Johnson, D. (2006).
9th dimacs implementation challenge: Shortest paths.
`http://www.dis.uniroma1.it/~challenge9`.
acessado em 08/06/2018.

Kruskal, J. B. (1956).
On the shortest spanning subtree of a graph and the traveling salesman problem.
Proceedings of the American Mathematical society, 7(1):48–50.