

# AA de Grafos e Algoritmos

## Algoritmo de Kruskal

Jamile Santos<sup>1</sup>, Liliane Neves<sup>1</sup>, Victor Pedro<sup>1</sup>

<sup>1</sup> Departamento de Ciência da Computação – Instituto Multidisciplinar  
Universidade Federal Rural do Rio de Janeiro (UFRRJ)

`milysantos18@gmail.com, lilianeneves8@gmail.com, victorp@r7.com`

**Abstract.** *This paper describes the study about how to implement an algorithm capable of divide a valued graph  $G = (V, E)$  in  $k$ -groups of vertices in a way that the weight of the sum of the edges is the smallest possible. To solve this problem, the algorithm have to find a minimum spawning tree in the graph and after that eliminate the  $k - 1$  edges. To find the minimum spawning tree will be used the Kruskal's algorithm.*

**Resumo.** *Este relatório descreve o estudo sobre como implementar um algoritmo capaz de separar um grafo valorado  $G = (V, E)$  em  $k$ -grupos de vértices de forma que a soma do peso das arestas seja a menor possível. Para resolver este problema, o algoritmo deve encontrar uma árvore geradora mínima no grafo e após isso eliminar as  $k - 1$  arestas. Para encontrar a árvore geradora mínima será utilizado o algoritmo de Kruskal.*

## 1. Introdução

Dado um grafo não orientado conectado, uma árvore de dispersão deste grafo é um sub-grafo, que é uma árvore que conecta todos os vértices.

Um único grafo pode ter diferentes árvores de extensão. Nós podemos assinalar um peso a cada aresta, que é um número que representa quão desfavorável ela é, e atribuir um peso a árvore de extensão calculado pela soma dos pesos das arestas que a compõem.

Uma árvore de extensão mínima (também conhecida como árvore de extensão de peso mínimo ou árvore geradora mínima) é então uma árvore de extensão com peso menor ou igual a cada uma das outras árvores de extensão possíveis. Generalizando, qualquer grafo não direcional (não necessariamente conectado) tem uma floresta de árvores mínimas, que é uma união de árvores de extensão mínimas de cada uma de suas componentes conexas.

Este trabalho objetiva dissertar sobre o uso do algoritmo de Kruskal no desafio de encontrar  $k$  árvores a partir do grafo inicial de forma que a soma dos pesos das arestas destes sub-grafos seja a menor possível.

## 2. Árvore Geradora Mínima

Este problema, como muitos semelhantes, tem seu foco em encontrar uma Árvore Geradora Mínima (MST - Minimum Spawning Tree) para um grafo conhecido.

Há uma solução genérica para este problema, que serve de base para o algoritmo de Kruskal, que é o discutido neste trabalho. Este algoritmo genérico mantém um conjunto de arestas  $A$  e a seguinte invariante:

A cada iteração,  $A$  é subconjunto de arestas de uma MST.

O algoritmo mantém esta invariante adicionando ao conjunto  $A$ , a cada iteração, uma aresta  $(u, v)$  de maneira que a união de  $A$  com  $(u, v)$  seja um subconjunto de uma MST [Feofiloff 2015].

No algoritmos da figura 1 temos as seguintes variáveis:  $A$  é o vetor auxiliar onde a MST será armazenada,  $u$  e  $v$  são os vértices que serão ligadas pela menor aresta possível.

```

GENERIC-MST( $G, w$ )
1  $A := \{\}$ 
2 while  $A$  não forma uma árvore espalhada
3     do encontre uma aresta  $(u, v)$  que é segura para  $A$ 
4      $A := A$  união  $\{(u, v)\}$ 
5 return  $A$ 

```

Figura 1. Algoritmo MST Genérico

7

### 3. O Algoritmo de Kruskal

O algoritmo de Kruskal propõe que um grafo qualquer seja considerado uma floresta de árvores compostas por apenas um vértice e procura, a cada iteração, conectar duas árvores distintas da floresta através de uma aresta que possua peso mínimo [Thomas H. Cormen 2009].

Dessa forma, suponha que, numa determinada iteração, o algoritmo escolher a aresta  $(u, v)$  para ser inserida no conjunto  $A$  e que a aresta  $(u, v)$  conecte a árvore  $C_i$  à árvore  $C_j$ . Observe que, dentre todas as arestas que conectam duas componentes distintas nesta iteração,  $(u, v)$  é uma das que possui menor peso, pois foi escolhida. Logo, nesta iteração, para qualquer corte que separe  $u$  de  $v$ ,  $(u, v)$  é uma aresta leve pelo fato de não existir aresta de menor peso separando duas componentes nesta iteração [Thomas H. Cormen 2009].

```

MST-KRUSKAL( $G, w$ )
1  $A := \{\}$ 
2 for cada vértice  $v$  em  $V[G]$ 
3     do MAKE-SET( $v$ )
4 Ordene as arestas de  $E$  em ordem crescente de peso ( $w$ )
5 for cada aresta  $(u, v)$ , tomadas em ordem crescente de peso ( $w$ )
6     do if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7         then  $A := A$  união  $\{(u, v)\}$ 
8         UNION( $u, v$ )
9 return  $A$ 

```

Figura 2. Algoritmo MST Kruskal

A Figura 3 exemplifica uma execução do algoritmo de Kruskal. É possível notar que as arestas são visitadas em ordem crescente de peso e, para que uma aresta seja inserida no conjunto  $A$ , seus vértices adjacentes devem pertencer a componentes diferentes. Neste pseudocódigo,  $V$  representa o conjunto de vértices do grafo  $G$ , e  $u$  e  $v$  representam os vértices que serão ligados pela aresta selecionada, e  $E$  é o conjunto de arestas do grafo.

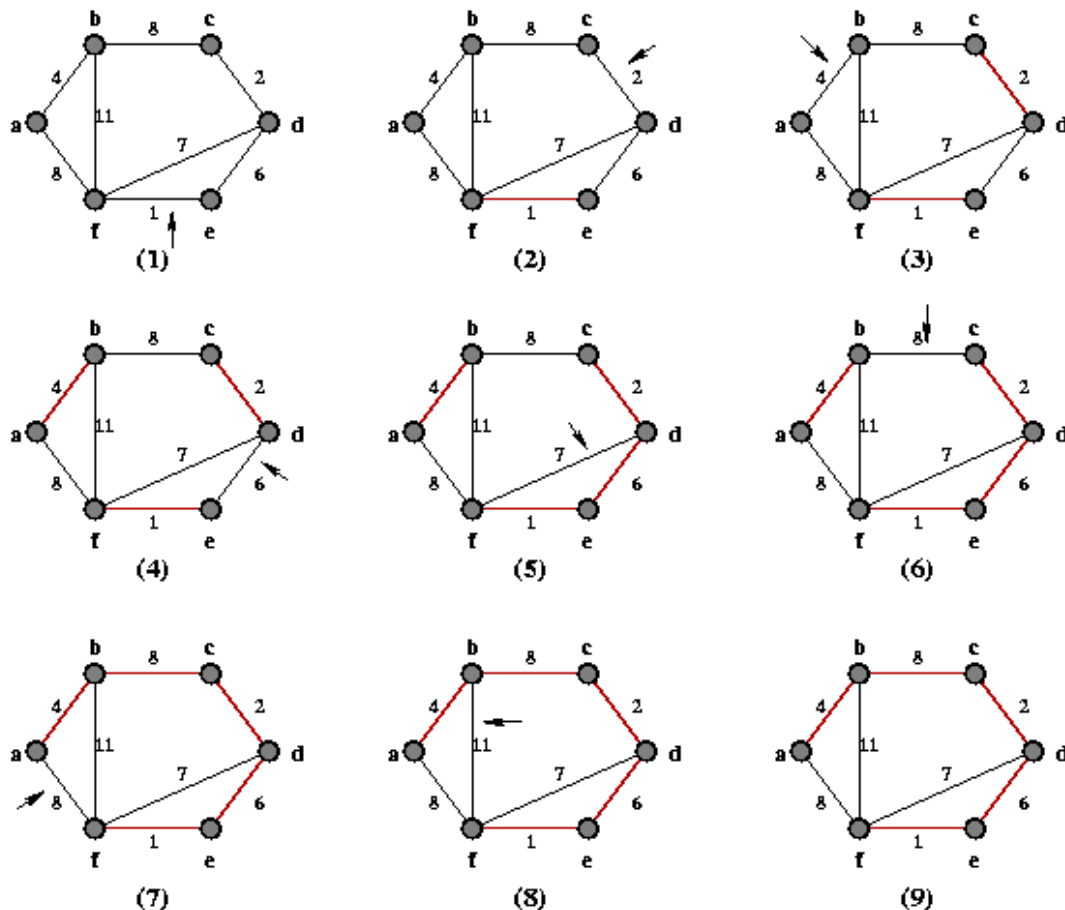


Figura 3. Algoritmo MST Genérico

#### 4. Análise de Complexidade do Algoritmo

A complexidade do Algoritmo é:  $T(N) = \Theta(E \log E)$ , ou  $\Theta(E \log V)$ . A ordenação das arestas tem tempo  $\Theta(E \log E)$ . Depois da ordenação, há uma iteração por todas as arestas e é aplicada a união, esta operação é aproximadamente  $\Theta(\log V)$ . Portanto, a complexidade geral do algoritmo é  $\Theta(E \log E + E \log V)$ . O valor de  $E$  pode ser no máximo  $V^2$ , logo,  $\Theta(\log V)$  e  $\Theta(\log E)$  são as mesmas. Dessa forma podemos afirmar que a complexidade geral do algoritmo é  $\Theta(E \log E)$  ou  $\Theta(E \log V)$ . [GeeksForGeeks 2012]

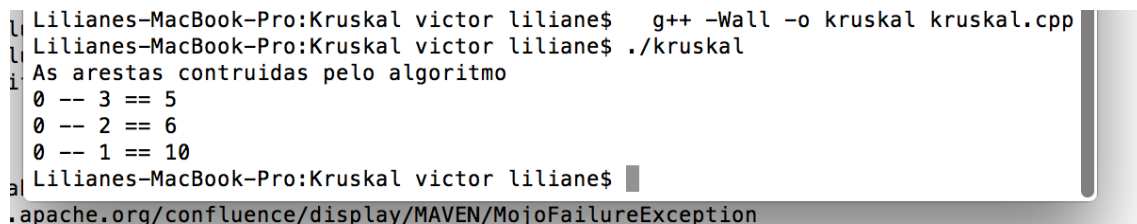
#### 5. Conclusões

Como foi proposto, a implementação é capaz de separar o grafo em árvores conexas pela aresta de menor peso possível, utilizando uma Árvore Geradora Mínima através do algoritmo de Kruskal.

Os resultados obtidos pelo programa nos testes realizados apresentam sempre resultados corretos e tempos de execução pequenos, mesmo aumentando o número de vértices e arestas.

### 5.1. Resultados

O algoritmo foi feito executando com G++ num MacBook Pro OSX Yosemite core i7, 16GB de RAM e em um MacBook OSX Mavericks, ambos utilizaram o Xcode 7 como IDE C++.

A screenshot of a terminal window on a Mac. The prompt is 'Lilianes-MacBook-Pro:Kruskal victor liliane\$'. The first command is 'g++ -Wall -o kruskal kruskal.cpp'. The second command is './kruskal'. The output of the program is: 'As arestas contruidas pelo algoritmo', followed by three lines: '0 -- 3 == 5', '0 -- 2 == 6', and '0 -- 1 == 10'. The prompt returns to 'Lilianes-MacBook-Pro:Kruskal victor liliane\$'.

```
Lilianes-MacBook-Pro:Kruskal victor liliane$ g++ -Wall -o kruskal kruskal.cpp
Lilianes-MacBook-Pro:Kruskal victor liliane$ ./kruskal
As arestas contruidas pelo algoritmo
0 -- 3 == 5
0 -- 2 == 6
0 -- 1 == 10
Lilianes-MacBook-Pro:Kruskal victor liliane$
```

Figura 4. Código executado

### Referências

- Feofiloff, P. (2015). Algoritmo de kruskal. [http://www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/aulas/kruskal.html](http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html). Acessado em: Jul. 2015.
- GeeksForGeeks (2012). Greedy algorithms — set 2 (kruskal's minimum spanning tree algorithm).
- Ime (2015). Algoritmo de kruskal. [https://pt.wikipedia.org/wiki/Algoritmo\\_de\\_Kruskal](https://pt.wikipedia.org/wiki/Algoritmo_de_Kruskal). Acessado em: Jul. 2015.
- Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. (2009). *Introduction to Algorithms*. MIT Press.
- Wikipedia (2015). Algoritmo de kruskal @ONLINE.