

Minimum Spanning Tree Algorithm (MST)

Rafael G. Nagel

2018-06-22

Outline

Introduction

Algorithm explanation (Kruskal)

Applications¹ and Problem

- ▶ Usually relates to optimizations in **network design**
 - ▶ telephone
 - ▶ electrical
 - ▶ hydraulic
 - ▶ TV cable
 - ▶ road
- ▶ Also indirect applications
 - ▶ learning salient features for real-time face verification
 - ▶ max bottleneck paths

¹[https:](https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/)

Conditions


- ▶ graph G with *positive* edge weights
- ▶ *undirected* edges
- ▶ find a **min weight** set of *edges* that connects **all** of the *vertices*
- ▶ number of *edges* we want:

$$\text{edges} = \text{number of vertices} - 1$$

Few algorithms that implement MST ²

1. Kruskal's algorithm
2. Prim's algorithm
3. Boruvka's algorithm

-
- ▶ Why did Kruskal?
 - ▶ easier to code and understand

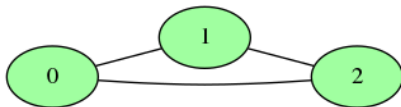
²<https://www.ics.uci.edu/~eppstein/161/960206.html> 

Overview

1. order *edges* by weight (ascendant)
2. pick edge and check if it forms a cycle with the spanning tree
 - ▶ if cycle formed then discard edge
 - ▶ else include edge
3. repeat 2) until $e = v - 1$

How to check if graph has cycle?

- ▶ Union-find³ algorithm
 - ▶ temporary vector to save parent of vertices being inserted



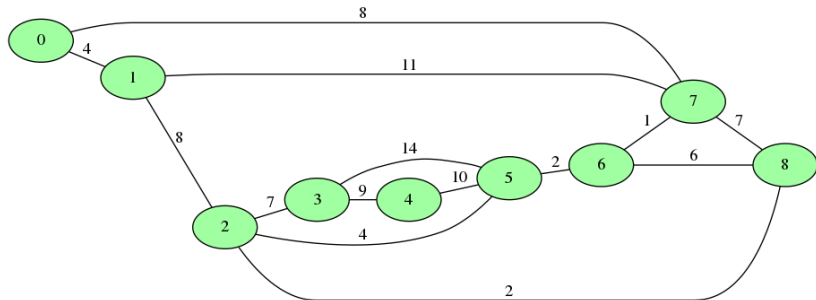
0	1	2
-1	-1	-1

0	1	2
1	-1	-1

0	1	2
1	2	-1

³<https://www.geeksforgeeks.org/union-find/>

Step by step with example⁴

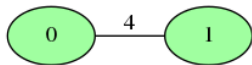
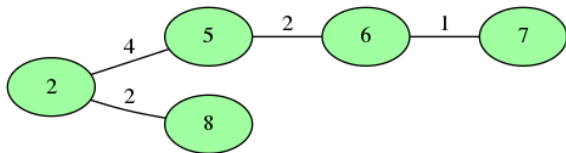


⁴<https://www.geeksforgeeks.org/greedy-algorithms-set-2-kruskals-minimum-spanning-tree-mst/>

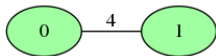
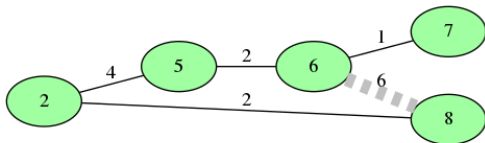
Graph sorted by edges

weight	src	dst
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

pick edges from sorted and try to include each one

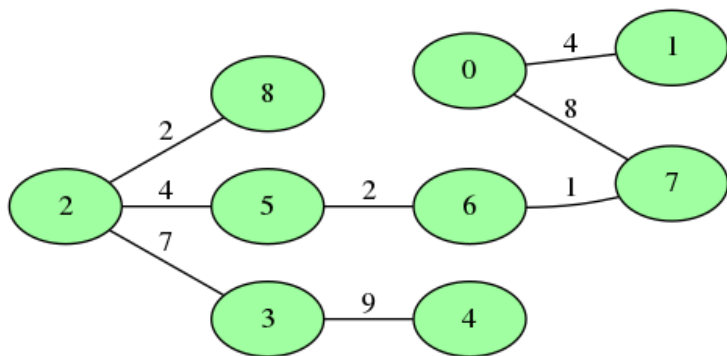


... no cycles so far.



... including 8-6 produces a cycle. Do not include it.

repeat until: $\text{edges} = \text{vertices} - 1$




Complexity

summing up:

1. sort graph ascendant (qsort $\rightarrow O(n \log n)$)
2. apply MST (Kruskal) algorithm
 - 2.1 for *each edge* in sorted list:
 - 2.1.1 include it in MST graph
 - 2.1.2 check if *formed cycle*; if so remove it
(find-union algorithm)

Kruskal complexity

- ▶ The find-union is $O(n)$ in the *present work*
 - ▶ we could improve it to $O(\log n)$ using *union by Rank or Height*⁵

⁵<https://www.geeksforgeeks.org/union-find/> 

Complexity

Consider the worst case

- ▶ Everytime we add a new vertice in a graph, we can have:

$$\text{edges} + = \text{number of vertices} - 1.$$

- ▶ Thus, number of edges in the worst case is:

$$\begin{aligned} \text{edges} &= v^2 \\ \text{when edges} &\rightarrow \infty \end{aligned}$$

This implementation:

$$\begin{aligned} &O(e \times \log e) + O(e \times O(v)) \\ &O(e \times \log e + v^2 \times v) \\ &O(e \times \log v + v^3) \end{aligned}$$

Complexity

Implementation with: union-find = $O(\log n)$

$$\begin{aligned} & O(e \times \log e) + O(e \times \log v) \\ e = v^2 \rightarrow \log e = \log v^2 = 2 \times \log v \approx \log v \\ & \quad \quad \quad \vdots \\ & O(e \times (\log v + \log v)) = O(e \times 2 \times \log v) \\ & \quad \quad \quad \vdots \\ & O(e \times \log v) \end{aligned}$$

Code

Structs

- ▶ grafo
- ▶ vertice
- ▶ lista
- ▶ nó

New functions

- ▶ hasCycle()
- ▶ union()
- ▶ find()

Conclusion

- ▶ Instead of creating new *structs* or *modules*, I add *members* to structs and add new *functions* to modules.

How to avoid checking double vertex addition without $O(n)$?

- ▶ This issue arise when adding edges to graph to check if *has cycle*
- ▶ With this issue, the complexity has one more v factor:

$$O(e \times \log v + v^4)$$

Kruskal algorithm is *much easier* to implement than other

- ▶ Tradeoffs: Complexity vs. Time of coding