CS 217 – Algorithm Design and Analysis

Shanghai Jiaotong University, Fall 2019

Handed out on Thursday, 2019-09-26
First submission and questions due on Monday, 2019-09-30
You will receive feedback from the TA.
Final submission due on Thursday, 2019-10-10

3 Minimum Spanning Trees

Throughout this assignment, let G be a weighted graph, i.e., $G = (V, E, w)$ with $w : E \to \mathbb{R}^+$. For $c \in \mathbb{R}$ and a weighted graph $G = (V, E, w)$, let $G_c := (V, \{e \in E \mid w(e) \le c\})$. That is, G_c is the subgraph of G consisting of all edges of weight at most c.

Exercise 1 Let T be a minimum spanning tree of G, and let $c \in \mathbb{R}$. Show that T_c and G_c have exactly the same connected components. (That is, two vertices $u, v \in V$ are connected in T_c if and only if they are connected in G_c). You are encouraged to draw pictures to illustrate your proof!

Proof: In other words, we need to prove the following statements are equivalent , given $v_1, v_2 \in V$

$$\exists \text{ path } e_1, \ldots, e_n \in E, \text{ the path connects } v_1, v_2, \text{and } w(e_i) \le c \quad (*).$$

and

$$\exists \text{ path } e_1, \ldots, e_n \in E(T), \text{ the path connects } v_1, v_2, \text{and } w(e_i) \le c \quad (**).$$

Notice that $(**) \implies (*)$ is obvious. We only need to prove $(*) \implies (**)$. The path connects v_1, v_2 in T is unique, otherwise there are cycles.

Let the unique path connects $v_1 = u_1, \ldots, u_t = v_2$. And assume $\exists j, 1 \leq j < t$ such that

$$w\left(e(u_j, u_{j+1})\right) > c.$$

we remove the edge $e\left(u_j, u_{j+1}\right)$ from T, and get 2 sets of vertices A, B. A, B are connected respectively. We prove that $\exists e \in E$ that connects A, B plus $w\left(e\right) \leq c$. This is obvious from $(*)$. Hence it leads that T is not a MST, a contradiction. $\square$

Exercise 2 For a weighted graph G, let $m_c(G) := |\{e \in E(G) \mid w(e) \leq c\}|$, i.e., the number of edges of weight at most c (so G_c has $m_c(G)$ edges). Let T, T' be two minimum spanning trees of G. Show that $m_c(T) = m_c(T')$.

Proof: Let the edge set of T_c be

$$E\left(T_c\right) = \{e_1, e_2, \ldots, e_r\}.$$

We know that $E\left(T_c\right)$ forms several connected componets $A_1, A_2, \ldots, A_t \subset V$. And from the last exercise we know that $A_1, A_2, \ldots, A_t$ are also connected componets in T'. And we can assert the connected components in T' are exactly these A_i, otherwise apply the conclusion from last exercise, we can derive more components for T. And specifically, those components must be trees. Hence

$$m_c\left(T\right) = \sum_{i=1}^{t}\left(|A_i| - 1\right) = m_c\left(T'\right).$$

$\square$

Exercise 3 Suppose G is connected, and no two edges of G have the same weight. Show that G has exactly one minimum spanning tree!

Proof: Otherwise consider 2 minimum spanning tree T_1, T_2, let

$$E\left(T_1\right) = \{a_1, a_2, \ldots, a_r\}.$$

in which $a_i < a_{i+1}$, and similarly

$$E\left(T_2\right) = \{b_1, b_2, \ldots, b_s\}.$$

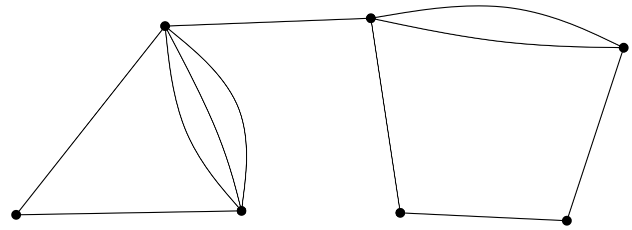
Let j be the minimum such that $a_j \neq b_j$. We can assert j exists since T_1, T_2 are different.

W.L.O.G let $a_j < b_j$, it leads that

$$m_{a_j}(T_1) = j \neq j - 1 = m_{a_j}(T_2).$$

Which contradicts with **Exercise 2**. $\square$

A *multigraph* is a graph that can have multiple edges, called "parallel edges". Without defining it formally, we illustrate it:



A multigraph.

All other definitions, like connected components and spanning trees are the same as for normal (simple) graphs. However, when two spanning trees use different parallel edges, we consider them different:

The same multigraph with two different spanning trees.

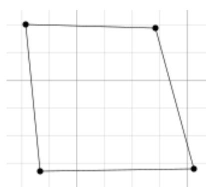
Exercise 4 How many spanning trees does the above multigraph on 7 vertices have? Justify your answer!

Proof: Obviously the edge connecting the left part of the graph and the right part must be included in a spanning tree. To construct a spanning tree, we now pay attention on these two subgraphs.

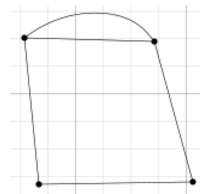
We first consider such a simple graph with 4 edges. Without any effort, it has 4 spanning trees, and this result can be applied to a graph with n edges.

Then we consider a simple multi-graph case. If we remove one of the parallel edges in the multi-graph, it's exactly a simple graph discussed above. So obviously, we know it has $4 \times 2 = 8$ spanning trees.

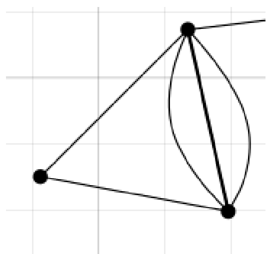
Based on these observations, we consider the subgraphs in *Exer.*4. Left part has 3 parallel edges, which leads to $3 \times 3 = 9$ spanning trees, and right



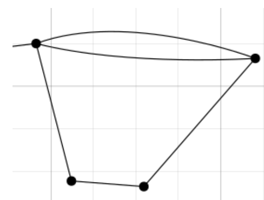
(a) simple graph



(b) simple multi graph

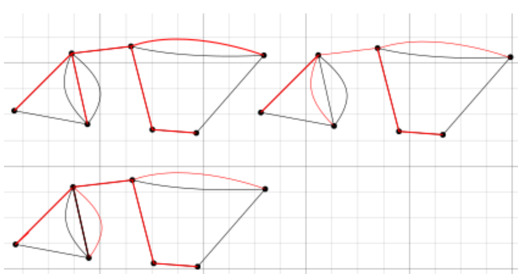


(a) left part

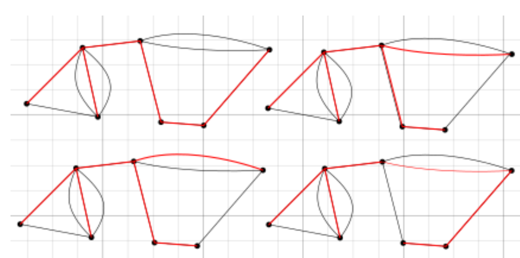


(b) right part

part has $2 \times 4 = 8$ spanning trees. Due to the simple counting method, there're $9 \times 8 = 72$ spanning trees for the graph.Below is some of them. $\square$



(a) left part examples



(b) right part examples

Exercise 5 Suppose you have a polynomial-time algorithm that, given a multigraph H, computes the number of spanning trees of H. Using this algorithm as a subroutine, design a polynomial-time algorithm that, given a weighted graph G, computes the number of minimum spanning trees of G.

Proof: We already know $Prim's$ and $Kruskal's$ algorithm to construct one of the MSTs of a graph, and only those edges having the same weight may lead to different spanning trees and we have a magic polynomial-time algorithm for cumputing the number of spanning trees. Based on these tools, we just need to find do some midifications for $Kruskal's$ algorithm.

We know that during the process of $Kruskal's$ algorithm, it gets the edges with different weights of an MST. Having these weights, we can get all other *candidate* edges for another MST as long as its weight is the same as one of the known MST's edge's. After that, we construct a subgraph g of graph G, and each spanning tree of g is a MST of G. Call the magic algorithm, we get the number of spanning tree of g, which is also the number of MSTs of G. The pseudocode is shown below.

```
Data: graph G
Result: number of MSTs of G
T = Kruskal(G);
toMergeEdges = ∅ ;
for e ∈ T(E) do
    for e_tmp ∈ G(E) do
        if e.weight == e_tmp.weight and e ≠ e_tmp then
            toMergeEdges = toMergeEdges ∪ e_tmp;
        end
    end
end
T(E) = T(E) ∪ toMergeEdges;
result = ComputeSpanningTreeNumber(T);
return result
```

Algorithm 1: Compute number of MST

The running time for this algorithm is really simple, since we know that $Kruskal$ and $ComputeSpanningTreeNumber$ are polynomial-time. And finding $toMergeEdges$ need at most $|E|^2$ comparisons,namely $O(|E|^2)$, the merge for $T(E)$ and $toMergeEdges$ is $O(|E|)$. In all running time is the sum of these polynomial subroutines, so in all the running time is polynomial. $\square$