Mathematical Foundations of Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

Group: NOIDEA

8 Spanning Trees

- Homework assignment published on Monday, 2018-04-16
- Submit questions and first solution by Sunday, 2018-04-22, 12:00, by email to me and the TAs.
- Submit your final solution by Sunday, 2018-04-29.

8.1 Minimum Spanning Trees

Throughout this assignment, let $G = (V, E)$ be a connected graph and $w : E \to \mathbb{R}^+$ be a weight function.

Exercise 8.1. Prove the inverse of the cut lemma: If X is good, $e \notin X$, and $X \cup e$ is good, then there is a cut $S, V \setminus S$ such that (i) no edge from X crosses this cut and (ii) e is a minimum weight edge of G crossing this cut.

Proof. X is good and $X \cup e$ is good, means that there's a minimum spanning tree T such that $X \subseteq E(T)$ and $X \cup e \subseteq E(T)$. Also, X contains two or more components.

- Proof by contradiction that there exists a cut $V = S \cup \overline{S}$ of X such that no edge from X crosses this cut but e crosses this cut:

  If e does not cross any cut of X, meaning that the two vertices that e connected is within the same connected component of X, which leads to a cycle in this component in $X \cup e$. Then it can't be good, a contradiction.

- Proof by contradiction that e is a minimum weight edge of G crossing this cut:

  If e is not the minimum weight edge of G crossing this cut, which means that there exists a edge e' such that $w(e') < w(e)$. Moreover, $w(X \cup e') < w(X \cup e)$. Consider a tree T' such that $E(T') = (E(T) \setminus e) \cup e'$. T is a spanning tree, so that there are exactly 2 connected components if the edge e is subtracted from it. Then by adding the edge e', T' is also a spanning tree since no circles are incidented. However $w(T) - w(T') = w(e) - w(e') > 0$, which means that $w(T') < w(T)$. But T is the minimum spanning tree of G, thus a contradiction.

$\square$

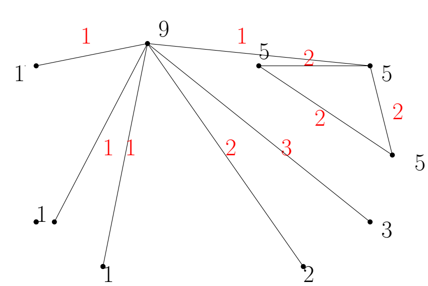
Definition 8.2. *For $c \in \mathbb{R}$ and a weighted graph $G = (V, E)$, let $G_c := (V, \{e \in E \mid w(e) \leq c\})$. That is, G_c is the subgraph of G consisting of all edges of weight at most c.*

Lemma 8.3. *Let T be a minimum spanning tree of G, and let $c \in \mathbb{R}$. Then 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).*

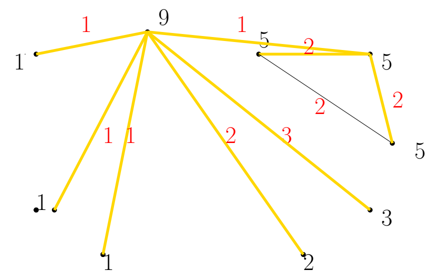
Exercise 8.4. Illustrate Lemma 8.3 with an example!

Solution. $\blacksquare$

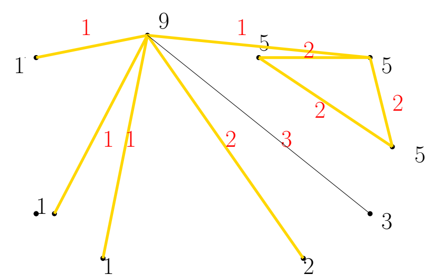
For a figure as below:



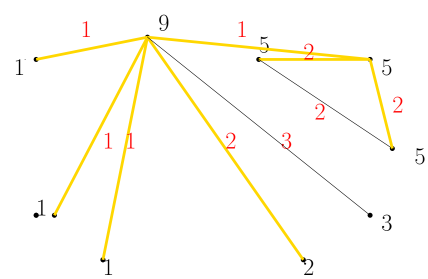
The MST T would be



Assume that $c = 2$, then G_2 is



And T_2 is



Lemma 8.3, obviously, holds for the example above.

Exercise 8.5. Prove the lemma.

Proof. The proof is given in 2 directions as below: $\square$

- two vertices $u, v \in V$ are connected in T_c if they are connected in G_c:
  - $\forall e \in E$, $c \geq w(e)$. Then T_c is the spanning tree which connects every 2 vertices. And G_c is the G itself. The lemma obviously holds.

- $\exists e \in E$, $c < w(e)$. Proof by contradiction, if there exist 2 vertices u, v that are connected in G_c, but are not connected in T_c. Obviously, u, v are connected in T. Suppose the connecting path is $(e_1, e_2, ..., e_k)$. There exists an edge $e_i \in E(T)$, $1 \leq i \leq k$, such that $w(e_i) > c$. For the only cut V in the 2 connected components of T', where $E(T') = E(T) \setminus e_i$. According to the cut lemma, e_i is the minimum weight edge that connecting the 2 components. Hence, there can not exist a path in which all the edges' weights are less than c in G. So u, v can not be connected in G_c, a contradiction.

- two vertices $u, v \in V$ are connected in G_c if they are connected in T_c:

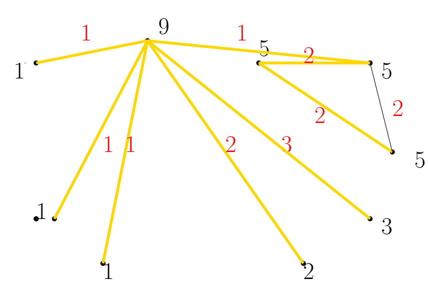
  The proof of this direction is much simpler. If u, v are connected in T_c, then there must exists a path $(e_1, e_2, ..., e_n)$ connecting u and v, in which every edge weight is no more than c. Further, all the edges in this path would be in G_c, which leads to a path connecting u, v in G_c as well.

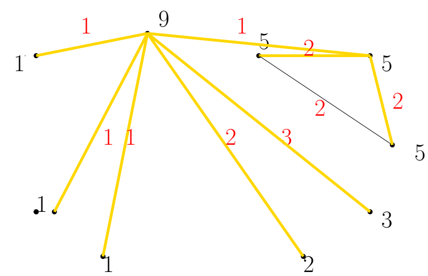
Definition 8.6. *For a weighted graph G, let $m_c(G) := |\{e \in E(G) \mid w(e) \leq c\}|$, i.e., the number of edges of weight at most c (so G_c has $m_c(G)$ edges).*

Lemma 8.7. *Let T, T' be two minimum spanning trees of G. Then $m_c(T) = m_c(T')$.*

Exercise 8.8. Illustrate Lemma 8.7 with an example!

Solution. Using the same example in **exercise 8.4**, the minimum spanning tree is





The lemma, obviously, holds. ■

Exercise 8.9. Prove the lemma.

Proof. T and T' both has exactly $|V| - 1$ edges.

Assume that $E(T) = (e_1, ..., e_k)$, in an order such that $w(e_1) \leq w(e_2) \leq ... \leq w(e_k)$, and $E(T') = (e'_1, ..., e'_k)$, in an order such that $w(e'_1) \leq w(e'_2) \leq ... \leq w(e'_k)$ and $k = |V| - 1$.

First, we proof that the weight sequence $w(e_1)w(e_2)...w(e_k)$ and $w(e'_1)w(e'_2)...w(e'_k)$ are exactly the same.

Consider constructing the minimum spanning tree by Kruskal's Algorithm. In this approach, each time when you want to expand your subgraph by adding an edge, you only care about the weight of an edge but not the edge, say (v_i, v_k), itself. So, each time, you will always get the exactly weight sequence, however, not always the same MST.

Therefore, for two MST trees of the same weight sequence, the lemma obviously holds. □

Exercise 8.10. Suppose no two edges of G have the same weight. Show that G has exactly one minimum spanning tree!

Proof. Consider constructing the minimum spanning tree by Kruskal's Algorithm. In this approach, each time when you want to expand your subgraph by adding an edge, you have only one choice, because no two edges have the same weight. Therefore, you'll get exactly the same MST each time you call the algorithm.

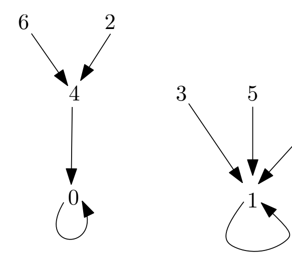
Besides, in this algorithm, the choice of the minimal weight edges is arbitrary. And different choice leads to different MST. So the Kruskal's Algorithm is somehow able to generate all possible MST from a given graph G (not only one MST from all possibilities).

Thus, G has exactly one minimum spanning tree. □

8.2 Counting Special Functions

In the video lecture, we have seen a connection between functions $f : V \to V$ and trees on V. We used this to learn something about the number of such trees. Here, we will go in the reverse direction: the connection will actually teach us a bit about the number of functions with a special structure.

Let V be a set of size n. We have learned that there are n^n functions $f : V \to V$. For such a function we can draw an "arrow diagram" by simply drawing an arrow from x to $f(x)$ for every V. For example, let $V = \{0, \ldots, 7\}$ and $f(x) := x^2 \mod 8$. The arrow diagram of f looks as follows:



The *core* of a function is the set of elements lying on cycles in such a diagram. For example, the core of the above function is $\{0, 1\}$. Formally, the core of f is the set

$$\{x \in V \mid \exists k \geq 1 f^{(k)}(x) = x\}$$

where $f^{(k)}(x) = f(f(\ldots f(x) \ldots))$, i.e., the function f applied k times iteratively to x.

Exercise 8.11. Of the n^n functions from V to V, how many have a core of size 1? Give an explicit formula in terms of n.

Solution. One mapping from V to V has a corresponding mapping graph and a corresponding vertebrate in a spanning tree of (V, E).

Hence, how many functions have a core of size 1 is equal to how many vertebrates generate a mapping graph of exactly one cycle in all the spanning tree with vertices V.

The number of spanning tree is n^{n-2}.

For each spanning tree,There are n choices to form a function of only 1 core, that is , by choosing both the *HEAD* and *BUTT* exactly the same.

In total, there are $n^{n-2} \times n = n^{n-1}$ functions that have a core of size 1.

■

Exercise 8.12. How many have a core of size 2 that consists of two 1-cycles? By this we mean that $\text{core}(f) = \{x, y\}$ with $f(x) = x$ and $f(y) = y$.

Solution. The number of spanning tree is n^{n-2}.

For each spanning tree, we choose 2 different elements, connected by the same edge, as *HEAD* and *BUTT* and let them form a cycle to itself in the corresponding mapping graph. Hence, there are $(n-1)$ choices.

In total, there are $n^{n-2} \times (n-1)$ functions. ■

Hint. For the previous two exercises, you need to use the link between functions $f : [n] \to [n]$ and vertebrates (T, h, b) from the video lecture.

8.3 Counting Trees with Prüfer Codes

In the video lecture, we have seen Cayley's formula, stating that there are exactly n^{n-2} trees on the vertex set $[n]$. We showed a proof using *vertebrates*. For this homework, read Section 7.4 of the textbook, titled "A proof using the Prüfer code".

Exercise 8.13. Let $V = \{1, \dots, 9\}$ and consider the code $(1, 3, 3, 2, 6, 6, 1)$. Reconstruct a tree from this code. That is, find a tree on V whose Prüfer code is $(1, 3, 3, 2, 6, 6, 1)$.

Exercise 8.14. Let $\mathbf{p} = (p_1, p_2, \dots, p_{n-2})$ be the Prüfer code of some tree T on $[n]$. Find a way to quickly determine the degree of vertex i only by looking at $\mathbf{p}$ and not actually constructing the tree T. In particular, by looking at $\mathbf{p}$, what are the leaves of T?

Exercise 8.15. Describe which tree on $V = [n]$ has the

1. Prüfer code $(1, 1, \dots, 1)$.
2. Prüfer code $(1, 2, 3, \dots, n-2)$.
3. Prüfer code $(3, 4, 5, \dots, n)$.
4. Prüfer code $(n, n-1, n-2, \dots, 4, 3)$.
5. Prüfer code $(n-2, n-3, \dots, 2, 1)$.
6. Prüfer code $(1, 2, 1, 2, \dots, 1, 2)$ (assuming n is even).

Justify and explain your answers.

The next two exercises use a bit of probability theory. Suppose we want to sample a random tree on $[n]$. That is, we want to write a little procedure (say in Java) that uses randomness and outputs a tree T on $[n]$, where each of the n^{n-2} trees has the same probability of appearing.

Exercise 8.16. Sketch how one could write such a procedure. Don't actually write program code, just describe it informally. You can assume you have access to a random generator `randomInt(`n`)` that returns a function in $\{1, \ldots, n\}$ as well as `randomReal()` that returns a random real number from the interval $[0, 1]$.

Clearly, a tree T on $[n]$ has at least 2 and at most $n - 1$ leaves. But how many leaves does it have on average? For this, we could use your tree sampler from the previous exercise, run it 1000 times and compute the average. However, it would be much nicer to have a closed formula.

Exercise 8.17. Fix some vertex $u \in [n]$. If we choose a tree T on $[n]$ uniformly at random, what is the probability that u is a leaf? What is the expected number of leaves of T?

Exercise 8.18. For a fixed vertex u, what is the probability that u has degree 2?