Mathematical Foundations
of
Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

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. Since $X \cup \{e\}$ is good, let $\{e\} \in T$, and T is a minimum spanning tree. Delete e in T and then we will get two connected components. Let S be the set of vertices in one connected component.

(i) Because obviously $X \subset T$, and e is the only element that crosses the cut, X has no edge that crosses the cut.

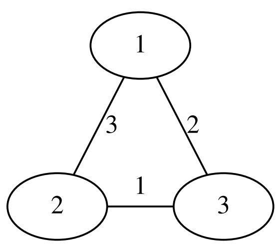
(ii) Assume there exists an edge e' with a smaller weight than e. Because it is not chosen when forming the spanning tree, it must form a cycle in S or $V \setminus S$. Otherwise, at that step of considering e', e has not been in the spanning tree, and then e' cannot form a cycle across the cut. $\square$

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.*

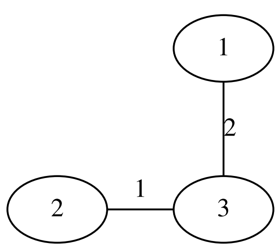
Lemma 8.3. *Let T be a minimum spanning tree of G, and let $c \in \mathbb{R}$. Then T_c and G_c have exactly the same connected components. (That is, two vertices $u, v \in V$ are connected in T_c if and only if they are connected in G_c).*

Exercise 8.4. Illustrate Lemma 8.3 with an example!

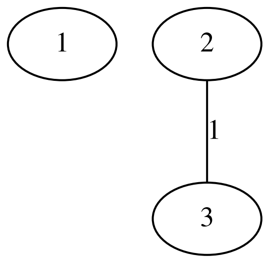
Solution. For example, G is:



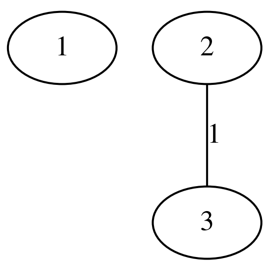
Then, T is:



$G_{1.5}$ is:



$T_{1.5}$ is:



$G_{1.5}$ and $T_{1.5}$ have the same connected components.

Exercise 8.5. Prove the lemma.

Proof. In *Kruskal Algorithm*, $\forall c \in \mathbb{R}, \forall e_i$ that $w(e) \leq c$, at the step considering it, if we can put e_k in T, then we can put it in T_c, which will result in T_c's connected component reduced by one, and since it does not cause a cycle in T_c, it does not cause a cycle in G_c, which will reduce G_c's connected components by one. If we cannot add it into T, obviously this edge does not affect T_c, and since it causes a cycle in T, it will not reduce the connected components when added to G_c. In conclusion, each edge has the same influence on T_c and G_c at the step selecting them.

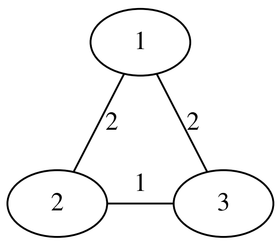
$\square$

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).*

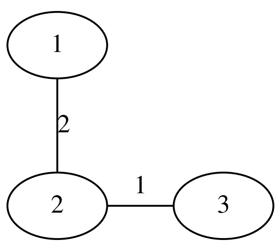
Lemma 8.7. *Let T, T' be two minimum spanning trees of G. Then $m_c(T) = m_c(T')$.*

Exercise 8.8. Illustrate Lemma 8.7 with an example!

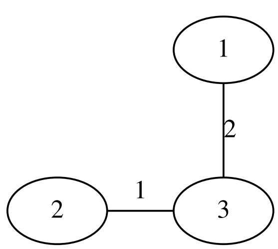
Solution. For example, G is:



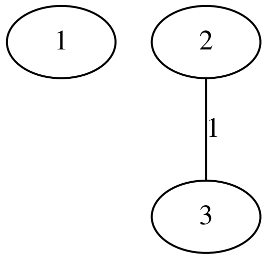
T is:



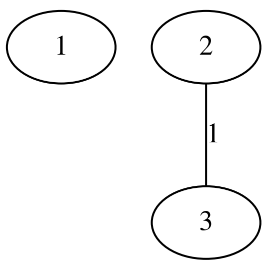
T' is:



$T_{1.5}$ is:



$T'_{1.5}$ is:



$T_{1.5}$ and $T'_{1.5}$ all have 1 edge.

Exercise 8.9. Prove the lemma.

Proof.

$$\begin{aligned} m_c(T) &= m_{+\infty}(T_c) \\ &= |E| - \text{connected component}(T_c) \\ &= |E| - \text{connected component}(G_c) \\ &= |E| - \text{connected component}(T'_c) \\ &= m_{+\infty}(T'_c) \\ &= m_c(T') \end{aligned}$$

$\square$

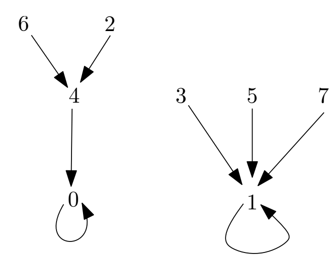
Exercise 8.10. Suppose no two edges of G have the same weight. Show that G has exactly one minimum spanning tree!

Proof. Suppose there are two minimum spanning tree T and T'. Consider in *Kruskal Alogorithm*, the first step where they add an edge to themselves is that T chooses e_i while T' chooses e_j. However, since they are precisely the same after the last step, e_i and e_j must have the same weight so that they are taken into consideration at the same time, which contradicts with the conditions. □

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.

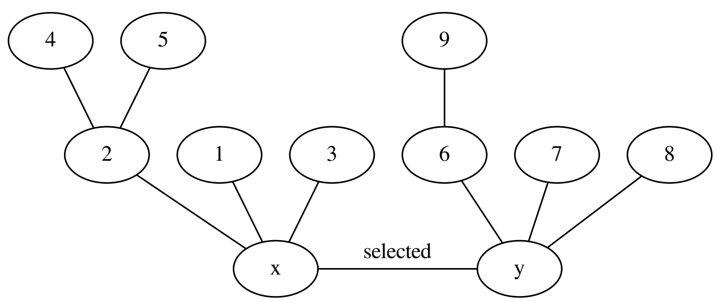
Solution.

It is obvious that there are n^{n-2} trees on n vertices, with $n-1$ many edges. Because there is a function from V to V, so there must be n edges in the graph. The one left forms a cycle and there are n ways to choose the cycle because there are n vertices. In total ,there are $n \times n^{n-2} = n^{n-1}$ functions.

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.

For a function f, when $\text{core}(f) = \{x, y\}$ with $f(x) = x$ and $f(y) = y$, we can convert it to a vertebrate (T, x, y):



x, y are *head* and *butt* and there exists one and only one edge between x and y. So every function f that $\text{core}(f) = \{x, y\}$ with $f(x) = x$ and $f(y) = y$ is a vertebrate with *head* and *butt* connected, and correspondingly, a tree with an edge selected.

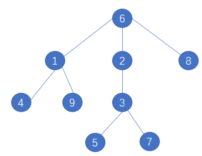
For n vertices, there are n^{n-2} trees and every tree has $n-1$ edges. Thus, there are $n^{n-2} \times (n-1)$ trees with an edge selected. So, $(n-1) \times n^{n-2}$ functions have a core of size 2 that consists of two 1-cycles.

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 $V = \{1, \ldots, 9\}$ 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 $\mathbf{p} = (p_1, p_2, \ldots, p_{n-2})$ 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?

Solution.

We assume the times vertex i appear in the Prüfer code is i_t, and it is obvious the degree of vertex is $i_t + 1$, and the the leaves of T are the vertices which don't appear in the Prüfer code.

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

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

Justify and explain your answers.

Proof. According to the algorithm to generate Prüfer code:
1. Figure1: Cut the leaf 2,3,4...n-1 in turn, and so it's Prüfer code is $(1, 1, \ldots, 1)$.
2. Figure2: Cut the leaf n-1,1,2,3,4...n-3 in turn, and so it's Prüfer code is $(1, 2, 3, \ldots, n - 2)$.

Figure 1:

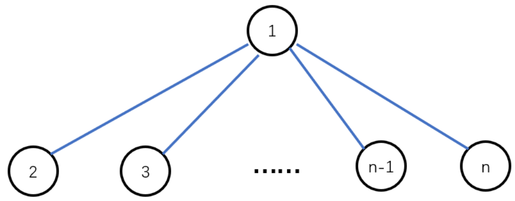


Figure 2:

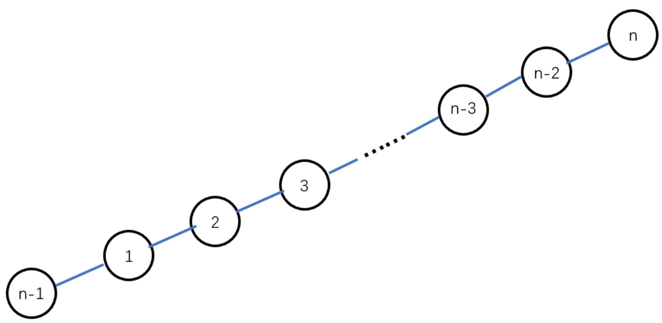


Figure 3:

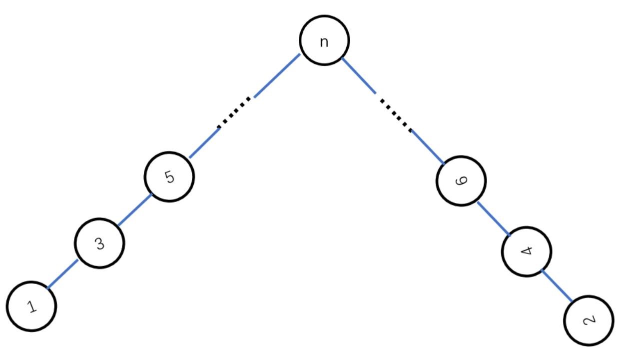


Figure 4:

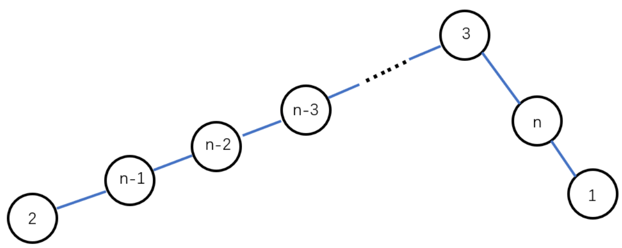


Figure 5:

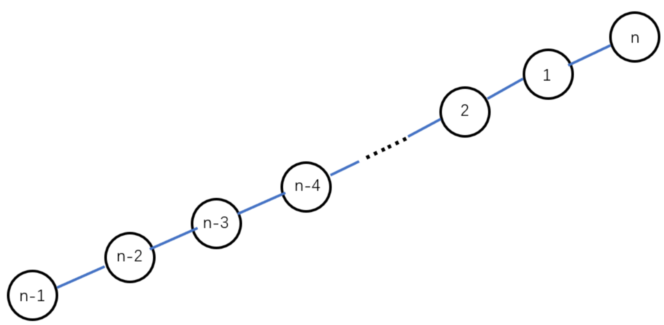
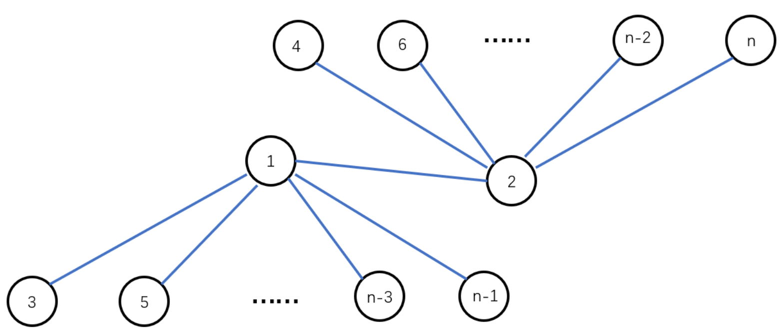


Figure 6:



3. Figure3: Cut the leaf 1,2,3...n-2 in turn, and so it's Prüfer code is $(3, 4, 5, \ldots, n)$.
4. Figure4: Cut the leaf 1,2,n-1,n-2,n-3,...4 in turn, and so it's Prüfer code is $(n, n-1, n-2, \ldots, 4, 3)$.
5. Figure5: Cut the leaf n-1,n-2,n-3,...2 in turn, and so it's Prüfer code is $(n-2, n-3, \ldots, 2, 1)$.
6. Figure6: Cut the leaf 3,4,5,...,n-1,1 in turn, and so it's Prüfer code is $(1, 2, 1, 2, \ldots, 1, 2)$.

$\square$

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 $[0, 1]$.

Solution.

We may generate a random tree in the following way:

1. Given the number of vertices in the tree n, we use `randomInt(`n`)` to generate $n-2$ random integers and get a Prüfer code.

2. Generate a tree according to Prüfer code. That is,

a. Init P as sorted Prüfer code, U as inetegers not appearing in the Prüfer code, R as vertices removed.

b. Take the first element p from P and first element u from U, add an edge u, p to the tree. Remove p from P and u from U.

c. Check if p appear in P. If not, add p to U.

d. Repeat a, b, c until there is no element of P. Take two elements from U and add an edge between them.

Obviously, every Prüfer code has the same probability of appearing. So every possible tree has the same probability of appearing.

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?

> **Solution.**
> $P(\text{u is a leaf}) = (1 - \frac{1}{n})^{n-2}$
> $E(\#leaf) = n(1 - \frac{1}{n})^{n-2}$

Proof. u is a leaf means u does not appear in Prüfer code. So for each element in the Prüfer code, it has $n - 1$ choices. The number of trees in which u is leaf is $(n - 1)^{n-2}$. So the probability that u is a leaf is $(1 - \frac{1}{n})^{n-2}$, expected number of leaves is $n(1 - \frac{1}{n})^{n-2}$. □

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

> **Solution.**
> $P(degree(u) = 2) = \frac{(n-2)(n-1)^{n-3}}{n^{n-2}}$