# Mathematical Foundations of Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

Spring 2019


**Group: navigator**

| | |
|---|---|
| Xu Huan | 517021910724 |
| Tianyao Shi | 517021910623 |
| Chenxiao Yang | 517021910540 |
| Jiaqi Zeng | 517021910882 |

# 7 Spanning Trees

## 7.1 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 7.1.** 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 $X$ is good, $e \notin X$, and $X \cup e$ is good(acyclic and totoal weight is smallest for $|X| + 1$ edges, we guess?), we have $|X| < n - 1$. Since we need at least $n - 1$ edges to connect all vertices in a graph, there must exist a cut where no edge from $X$ crosses this cut, or we would use less than $n - 1$ edges to connect all vertices, which is impossible. (i) is now proved. Then it is assumed that $e$ crosses such a not-crossed-by-any-edge-in-$X$ cut.

For (ii), suppose $e$ is not a minimum weight edge of $G$ crossing this cut, say there exists some $e^{'}$ crossing this cut with $w(e) > w(e^{'})$. Then we cen infer that $X \cup e$ is not good since $X \cup e^{'}$ has smaller total weight. By contradiction, (ii) is correct. □

**Definition 7.2.** *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 7.3.** *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 7.4.** Illustrate Lemma 7.3 with an example!

**Solution** See Figure 7.1. In $G_3$, no edge crosses the cut, and so are E,F not connected in $T_3$. It's easy to see the lemma holds for $c$ equals other cases.

**Exercise 7.5.** Prove the lemma.

*Proof.* **Necessity** If two vertices are connected in $T_c$, then the edges on the path between the two vertices $\in E$, all satisfying $w(e) \leq c$, which indicates they are also in $G_c$. Therefore in $G_c$ the two vertices are also connected.

**Sufficiency** Suppose that two vertices are connected in $G_c$ yet not in $T_c$. This suggests that there exists some edge $e$ with $w(e) \leq c$ connecting the two vertices in $G$, and in the original $T$ weights of edges on the path between the two vertices is greater than $c$. If we replace the path in $T$ with that in $G_c$, we get a smaller minimum spanning tree, which is contradictory. □

**Definition 7.6.** *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 7.7.** *Let $T, T'$ be two minimum spanning trees of $G$. Then $m_c(T) = m_c(T')$.*

**Exercise 7.8.** Illustrate Lemma 7.7 with an example!

**Solution** See Figure 7.1.Let the left MST be $T$ and another $T'$, $m_1(T) = m_1(T') = 1$, $m_2(T) = m_2(T') = m_3(T) = m_3(T') = 2$, $m_4(T) = m_4(T') = 4$, $m_5(T) = m_5(T') = 5$.

**Exercise 7.9.** Prove the lemma.

*Proof.* Assume that for two minimum spanning tree $T$, $T'$, $\exists c$ s.t. $m_c(T) \neq m_c(T')$. Without loss of generality, say $m_c(T) < m_c(T')$, and $c_0$ is the first $c \in \mathbb{N}$ s.t. $m_c(T) < m_c(T')$. Say it is edge $a$ with $w(a) = c_0$, so that $a \notin T$ yet $a \in T'$. Add $a$ to $T$ we must get a cycle, otherwise $a$ would belong to $T$, as all other edges would have weight greater than $a$. Grab this cycle we can see there is at least one edge $a'$ that is in $T$ yet not in $T'$. And $a'$ must have weight smaller than $a$, since $a$, corresponding to $c_0$, is the first to appear resulting in $m_c(T) < m_c(T')$. Replacing $a$ with $a'$ in $T'$ will result in a smaller minimum spanning tree, which is contradictory. □

**Exercise 7.10.** Suppose no two edges of $G$ have the same weight. Show that $G$ has exactly one minimum spanning tree!

*Proof.* Assume that $G$ has more than one unique minimum spanning tree, $T, T'$. They have the same nodes yet different edges, since they are diffrent. Therefore there are at least two edges that belongs to one tree but not another. Let $e$ be the one with the smallest weight in such edges, and without loss of generality, say that $e \in T$. Add $e$ to $T'$ we get a cycle $C$ containing $e$, which also contains at least one edge $e'$ that is not in $T$. Since $e$ is chosen as the unique one with smallest weight, replace $e'$ with $e$ breaks the cycle to regain a minimum spanning tree, but with total weights smaller than $T'$, which is contradictory. □

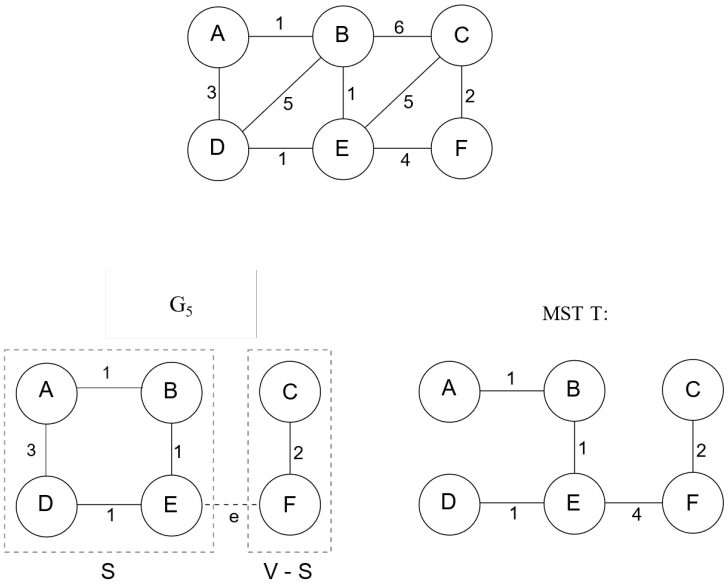



Figure 1: An illustration for Lemma 7.3. Figure source: https://en.wikipedia.org/wiki/Minimum_spanning_tree

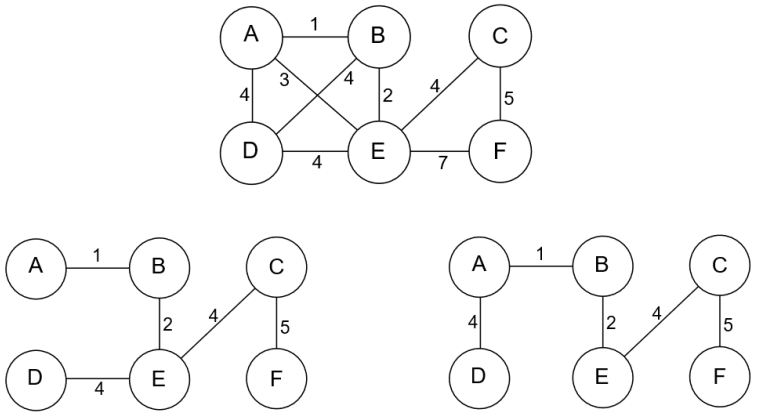
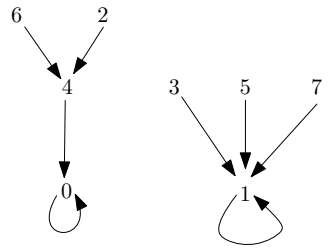



Figure 2: An illustration for Lemma 7.7. Figure source: https://en.wikipedia.org/wiki/Minimum_spanning_tree

## 7.2 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 7.11.** 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.** Recall that in the video lecture, we construct a function $f : V \to V$ according to the spanning tree and its vertebrate. For each spanning tree, there are totally $n^2$ corresponding vertebrates. Among those vertebrates, there are $n$ vertebrates whose head and butt is the same vertex. In this case, the core-size of the corresponding function is 1.

As we know, there are totally $n^{n-2}$ spanning tree. For each spanning tree, there are $n$ required functions. Therefore, the total function number is $n^{n-2} * n = n^{n-1}$.

**Exercise 7.12.** How many have a core of size 2 that consists of two 1-cycles? By this we mean that core$(f) = \{x, y\}$ with $f(x) = x$ and $f(y) = y$.

**Solution.** We arbitrarily choose an edge in the spanning tree and treat the vertex with smaller number as the head, another vertex as the butt. In this way, a spanning tree with a vertebrate is constructed. For each spanning tree, there are $n-1$ edges in the tree. So there are $n-1$ vertebrates for each spanning tree.

For such spanning tree with a vertebrate, the corresponding function only have two cores. For the head and the butt, $f(x) = x$.

Because there are totally $n^{n-2}$ spanning trees, the total number of required functions is $(n-1) * n^{n-2}$.

## 7.3 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*. I also presented a proof using Prüfer codes in the lecture. If you forgot to take notes, read Section 7.4 of the textbook.

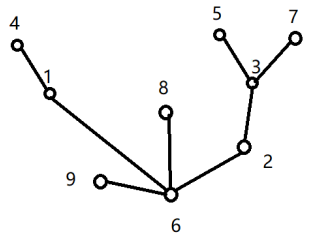
**Exercise 7.13.** Let $V = \{1, \ldots, 9\}$ and consider the code $(1, 3, 3, 2, 6, 6, 1)$. Reconstruct a tree from this code. That is, find a tree on $V$ whose Prüfer code is $(1, 3, 3, 2, 6, 6, 1)$.

*Proof.* The constructing process is shown in Table 1 .

Table 1:

| V | Code | Leaf |
|---|---|---|
| 123456789 | 1332661 | 4 |
| 12356789 | 332661 | 5 |
| 1236789 | 32661 | 7 |
| 123689 | 2661 | 3 |
| 12689 | 661 | 2 |
| 1689 | 61 | 8 |
| 169 | 1 | 6 |
| 19 | | |

The tree is shown as following.




□

**Exercise 7.14.** 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 $T$?

*Proof.*

1. The degree of vertex $i = 1 +$ the number of times that $i$ appears in $\mathbf{p}$.
2. Vertex $i$ is a leave if and only if $i$ does not appear in $\mathbf{p}$.

□

**Exercise 7.15.** 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.

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 7.16.** 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.** *Randomly create a sequence* $(p_1, p_2, \cdots, p_{n-2})$*, with* $1 \leq p_i \leq n$*. We can regard this sequence as prüfer code to construct a tree.*

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 7.17.** 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.**

1. *Number the vertices of tree* $T$ *from 1 to* $n$*. We can derive the prüfer code* $(p_1, p_2, \cdots, p_{n-2})$ *of the tree* $T$*, with* $1 \leq p_i \leq n$*. If* $u$ *is a leaf, then* $u$ *cannot appear in the prüfer code.*

$$P[u \text{ is a leaf}] = (n-1)^{n-2}/n^{n-2} = (\frac{n-1}{n})^{n-2}$$

2. *The number of leaves prüfer code of* $T$ *is denoted by the number of vertices which don't appear in the prüfer code.*

$$\mathbb{E}[\text{number of leaves of } T] = \sum_{k=1}^{n-2} \frac{(n-k)(\binom{n}{k}k^{n-2} - \frac{n-k+1}{k-1}\binom{n}{k-1}(k-1)^{n-2})}{n^{n-2}}$$

**Exercise 7.18.** For a fixed vertex $u$, what is the probability that $u$ has degree 2?

**Solution.**

$$P[u \text{ has degree } 2] = (n-1)^{n-3} \times 1 \times (n-2)/n^{n-2} = \frac{(n-2)(n-1)^{n-3}}{n^{n-2}}$$