Spanning trees

Lecture 10

Coding of labeled trees

Among ways to code a graph are adjacency and incidence matrices. For labeled trees, there are nicer and shorter ways to code.

Consider the following procedure for a tree T with vertex set $\{1, \ldots, n\}$:

Prüfer algorithm. Let $T_0 = T$. For $i = 1, \ldots, n-1$,
(a) let b_i be the smallest leaf in T_{i-1},
(b) denote by a_i the neighbor of b_i in T_{i-1}, and
(c) let $T_i = T_{i-1} - b_i$.

Coding of labeled trees

Among ways to code a graph are adjacency and incidence matrices. For labeled trees, there are nicer and shorter ways to code.

Consider the following procedure for a tree T with vertex set $\{1, \ldots, n\}$:

Prüfer algorithm. Let $T_0 = T$. For $i = 1, \ldots, n-1$,
(a) let b_i be the smallest leaf in T_{i-1},
(b) denote by a_i the neighbor of b_i in T_{i-1}, and
(c) let $T_i = T_{i-1} - b_i$.

The Prüfer code of T is the vector $(a_1, \ldots, a_{n-2})$.

Properties of Prüfer algorithm

(P1) $a_{n-1} = n$.

Properties of Prüfer algorithm

(P1) $a_{n-1} = n$.

(P2) Any vertex of degree s in T appears in $(a_1, \ldots, a_{n-2})$ exactly $s - 1$ times.

Properties of Prüfer algorithm

(P1) $a_{n-1} = n$.

(P2) Any vertex of degree s in T appears in $(a_1, \ldots, a_{n-2})$ exactly $s - 1$ times.

(P3) $b_i = \min\{k : k \notin \{b_1, \ldots, b_{i-1}\} \cup \{a_i, a_{i+1}, \ldots, a_{n-2}\}\}$ for each i.

Properties of Prüfer algorithm

(P1) $a_{n-1} = n$.

(P2) Any vertex of degree s in T appears in $(a_1, \ldots, a_{n-2})$ exactly $s - 1$ times.

(P3) $b_i = \min\{k : k \notin \{b_1, \ldots, b_{i-1}\} \cup \{a_i, a_{i+1}, \ldots, a_{n-2}\}\}$ for each i.

Proofs. (P1) follows from the fact that we always have a leaf distinct from n.

(P2) follows from the facts that at the moment some k appears in $(a_1, \ldots, a_{n-2})$, its degree decreases by 1 and for $s \geq 3$ the neighbors of leaves in s-vertex trees are not leaves.

(P3) follows from the algorithm and (P2).

Theorem 2.4 (Prüfer, 1918): Every vector $(a_1, \ldots, a_{n-2})$ with $a_i \in \{1, \ldots, n\}$ for each i is the Prüfer code of exactly one labeled n-vertex tree.

Theorem 2.4 (Prüfer, 1918): Every vector $(a_1, \ldots, a_{n-2})$ with $a_i \in \{1, \ldots, n\}$ for each i is the Prüfer code of exactly one labeled n-vertex tree.

Proof. **Uniqueness**. By (P1) we know $a_{n-1} = n$. Then by (P3), we can reconstruct b_i for all $1 \leq i \leq n-1$. Thus the edges are $a_1 b_1, \ldots, a_{n-1} b_{n-1}$.

Theorem 2.4 (Prüfer, 1918): Every vector $(a_1, \ldots, a_{n-2})$ with $a_i \in \{1, \ldots, n\}$ for each i is the Prüfer code of exactly one labeled n-vertex tree.

Proof. **Uniqueness**. By (P1) we know $a_{n-1} = n$. Then by (P3), we can reconstruct b_i for all $1 \leq i \leq n-1$. Thus the edges are $a_1 b_1, \ldots, a_{n-1} b_{n-1}$.

Existence. Given $(a_1, \ldots, a_{n-2})$, we let $a_{n-1} = n$ and define numbers b_i by (P3). Now consider the edges going from $a_{n-1} b_{n-1}$ backwards and check that for each i, b_i is a leaf in the graph formed by the edges $a_i b_i, \ldots, a_{n-1} b_{n-1}$. □

Theorem 2.4 (Prüfer, 1918): Every vector $(a_1, \ldots, a_{n-2})$ with $a_i \in \{1, \ldots, n\}$ for each i is the Prüfer code of exactly one labeled n-vertex tree.

Proof. **Uniqueness**. By (P1) we know $a_{n-1} = n$. Then by (P3), we can reconstruct b_i for all $1 \leq i \leq n-1$. Thus the edges are $a_1 b_1, \ldots, a_{n-1} b_{n-1}$.

Existence. Given $(a_1, \ldots, a_{n-2})$, we let $a_{n-1} = n$ and define numbers b_i by (P3). Now consider the edges going from $a_{n-1} b_{n-1}$ backwards and check that for each i, b_i is a leaf in the graph formed by the edges $a_i b_i, \ldots, a_{n-1} b_{n-1}$. □

Corollary 2.5 (Cayley's Formula, Borchardt 1860): There are n^{n-2} labeled n-vertex trees.

A subgraph of a graph G is spanning if its vertex set is $V(G)$.

A spanning tree in a graph G is a spanning subgraph of G that is a tree.

A graph has a spanning tree if and only if ?????

Cayley's Formula tells us the number of spanning trees in K_n.

A subgraph of a graph G is spanning if its vertex set is $V(G)$.

A spanning tree in a graph G is a spanning subgraph of G that is a tree.

A graph has a spanning tree if and only if ?????

Cayley's Formula tells us the number of spanning trees in K_n.

The number of spanning trees, $\tau(G)$, of a graph G is useful in some applications.

Theorem 2.6 (Matrix Tree Theorem) Let G be a loopless graph with $V(G) = \{v_1, \ldots, v_n\}$ and $a_{i,j}$ edges connecting v_i and v_j. Let $Q = (q_{i,j})_{i,j=1}^{n}$, where $q_{i,j} = \begin{cases} d(v_i), & \text{if } j = i; \\ -a_{i,j}, & \text{if } j \neq i. \end{cases}$ Let $Q_{s,t}$ be obtained from Q by deleting row s and column t. Then $\tau(G) = (-1)^{s+t} \det Q_{s,t}$.

Minimum spanning trees

In many applications, it makes sense to consider an **edge-weighted graph**, which is a graph $G = (V(G), (E))$ along with a weight function $w : E(G) \to \mathbb{R}$ that associates a real number (the weight) to each edge.

An application might be if you have multiple villages you want to connect with roads, the villages are all vertices, while the edges can be weighted with the cost to build a road between those two villages. You might want to minimize the cost of road construction.

Similarly, you may have a set of computers that you want to connect into a network, and the cost of connecting computer i with computer j is $c_{i,j}$. Again you may want to economize.

In both examples, we are looking for a spanning connected subgraph of our graph with the sum of the weights of the edges as small as possible.

Of course, if we have edges with negative weights, we'd better include all of them. If the resulting graph is connected, then we are done. If not, we can shrink each component into a vertex and consider the resulting graph with modified weights.

In both examples, we are looking for a spanning connected subgraph of our graph with the sum of the weights of the edges as small as possible.

Of course, if we have edges with negative weights, we'd better include all of them. If the resulting graph is connected, then we are done. If not, we can shrink each component into a vertex and consider the resulting graph with modified weights.

In both examples, we are looking for a spanning connected subgraph of our graph with the sum of the weights of the edges as small as possible.

Of course, if we have edges with negative weights, we'd better include all of them. If the resulting graph is connected, then we are done. If not, we can shrink each component into a vertex and consider the resulting graph with modified weights.

In view of this observation, we will assume all edge weights are non-negative. In this case, among spanning subgraphs of minimum total weight there always are spanning trees.

This motivates us to study the Minimum Spanning Tree Problem in a graph. As we know, K_n has n^{n-2} distinct spanning trees, so the idea to look at all such trees and choose among them a tree of minimum weight is not a great idea.