TREES

7 Basic Properties

DEFINITION 7.1: A connected graph G is called a tree if the removal of any of its edges makes G disconnected.

A tree can be defined in a variety of ways as is shown in the following theorem:

THEOREM 7.1: The following statements are equivalent:

1. G is a tree.
2. There exists a unique path between every two vertices of G.
3. G does not hold any closed paths and for $n > 2$ every additional edge creates a closed path in G.
4. G is connected and contains no closed path.
5. G does not contain a closed path and $e = n - 1$.
6. G is connected and $e = n - 1$.

Proof: 1. $\Rightarrow$ 2.) There exists at least one path between u and v as G is connected. If there was a second path between u and v, any edge on this path (not belonging to the first path) could be removed without making G disconnected.
2. $\Rightarrow$ 3.) G cannot contain a closed path that holds the vertices u and v as this results in two paths between u and v. Let $u'v' \notin E(G)$, i.e., $u'v'$ is an edge that is not a part of G, then adding $u'v'$ would create two paths between u' and v'.
3. $\Rightarrow$ 4.) If G is not connected, we could take u and v such that they do not reside in the same component. Adding the edge uv would not create a

closed path.
4. $\Rightarrow$ 5.) Apply induction on n (the case $n = 1$ is trivial). Let $H = G - uv$, where uv is an arbitrary edge in G. u and v have to belong to a different component of H (say, H_u and H_v), otherwise G would hold a closed path containing u and v. For each of these components criteria 4. is met, thus $e = e_{H_u} + e_{H_v} + 1 = n_{H_u} + n_{H_v} - 1 = n - 1$.
5. $\Rightarrow$ 6.) Assume G has k components (say, H_1, H_2, ..., H_k). For each component H_i we have $e_{H_i} = n_{H_i} - 1$, meaning that $e = \sum_i e_{H_i} = \sum_i (n_{H_i} - 1) = n - k$. Hence, $k = 1$.
6. $\Rightarrow$ 1.) Removing any edge makes G disconnected, because a graph with n vertices clearly needs at least $n - 1$ edges to be connected. Q.E.D.

DEFINITION 7.2: A tree T is called a subtree of the graph G if $T \subseteq G$. A spanning tree T of G is defined as a maximum subtree of G.

It should be clear that any spanning tree of G contains all the vertices of G. Moreover, for any edge e, there exists at least one spanning tree that contains e [Proof: Take an arbitrary tree T and assume $e \notin T$. When we add the edge e to T, the graph $T + e$ holds a closed path. Removing any edge $e' \neq e$ from this path provides us with a spanning tree T' that contains e].

THEOREM 7.2 (Cayley (1889)): Let $|V| = n$, there are n^{n-2} spanning trees on V, that is, K_n has n^{n-2} spanning trees.

Proof: Let v be any vertex in K_n and define $X(k)$ as the number of spanning trees where v has a degree equal to k. Assume $k > 1$ and $n > 2$ (the cases $n = 1, 2$ are trivial). Let T be a spanning tree such that $d_T(v) = k$.
(A) Choose an edge uu' that is not incident to v (i.e., $v \neq u, u'$). $T - uu'$ is disconnected and u and u' belong to a different component. Without loss of generality assume v belongs to the component of u. Then $T' = T - uu' + vu'$ is a spanning tree. The number of couples (T, T') equals $X(k-1)(n-k)$ as there where $n - 1 - (k - 1)$ edges in T not incident to v.
(B) Let $\bar{T}'$ be a spanning tree with $d_{\bar{T}'}(v) = k$. If we remove the i-th edge vu_i incident to v ($i = 1, \ldots, k$), we obtain a disconnected graph with two components T_i and $\bar{T}' - T_i$, where T_i is said to hold u_i. If we now connect u_i with any vertex $v' \neq v$ in $\bar{T}' - T_i$, we obtain a spanning tree $\bar{T}$ with $d_{\bar{T}}(v) = k - 1$. There are $X(k)(\sum_i (n_{\bar{T}'-T_i} - 1)) = X(k)(nk - (n-1) - k) = X(k)(n-1)(k-1)$ couples $(\bar{T}, \bar{T}')$.

(C) Clearly, the number of couples (T, T') and $(\bar{T}, \bar{T}')$ are identical. Thus, $X(k-1) = (n-1)(k-1)X(k)/(n-k)$. Replacing k by $n-j$ gives $X(n-j-1)j/[(n-j-1)(n-1)] = X(n-j)$. Moreover, $X(n-1) = 1$, therefore, it is easy to show by induction on j that $X(n-j) = (n-1)^{j-1}\binom{n-2}{j-1}$. The total number of spanning trees equals $\sum_{j=1}^{n-1} X(n-j) = \sum_{j=1}^{n-1} \binom{n-2}{j-1}(n-1)^{j-1} = ((n-1)+1)^{n-2} = n^{n-2}$ by Newton's binomial theorem. Q.E.D.

EXERCISES 7.1: On spanning trees:

1. We state that two graphs G and G' are isomorphic if there exists a bijection $\alpha : V(G) \to V(G')$ such that $uv \in E(G) \Leftrightarrow \alpha(u)\alpha(v) \in E(G')$. Thus, α simply renames the vertices of G. Determine the number of spanning trees in K_n *up to isomorphisms* for $n = 1, \ldots 7$.

8 The Connector Problem

To build a network connecting n nodes (towns, computers, chips in a computer) it is desirable to decrease the cost of construction of the links to the minimum. This is the connector problem. In graph theoretical terms we wish to find an optimal spanning subgraph of a weighted graph. Such an optimal subgraph is clearly a spanning tree, for, otherwise a deletion of any edge that is part of a cycle will reduce the total weight of the subgraph. Let then G^α be a graph G together with a weight function $\alpha : E(G) \to R^+$ (positive reals) on the edges. Kruskal's algorithm (also known as the greedy algorithm) provides a solution to the connector problem.

ALGORITHM 8.1 (Kruskal's algorithm (1956)): For a connected and weighted graph G^α of order n:

1. Let e_1 be an edge of smallest weight, and set $E_1 = \{e_1\}$.
2. For each $i = 2, 3, \ldots, n-1$ in this order, choose an edge $e_i \notin E_i$ of smallest possible weight such that e_i does not produce a cycle when added to $G[E_{i-1}]$, and let $E_i = E_{i-1} \cup \{e_i\}$. Notice, $G[E_i]$ need not to be connected.

The final outcome is $T = (V(G), E_{n-1})$ is a spanning tree with a minimal weight.

Proof: By the construction, $T = (V(G), E_{n-1})$ is a spanning tree of G, because it contains no cycles, it must be connected and has $n-1$ edges[6]. We now show that T has the minimum total weight among the spanning trees of G. Suppose T_1 is any spanning tree of G. Let e_k be the first edge produced by the algorithm that is not in T_1. If we add e_k to T_1, then a cycle C containing e_k is created. Also, C must contain an edge e that is not in T. When we replace e by e_k in T_1, we still have a spanning tree, say T_2. However, by the construction, $\alpha(e_k) \leq \alpha(e)$, and therefore $\alpha(T_2) \leq \alpha(T_1)$. Note that T_2 has more edges in common with T than T_1. Repeating the above procedure, we can transform T_1 to T by replacing edges, one by one, such that the total weight does not increase. We deduce that $\alpha(T) \leq \alpha(T_1)$.

Q.E.D.

The outcome of Kruskal's algorithm need not be unique. Indeed, there may exist several optimal spanning trees (with the same weight, of course) for a graph. The runtime of this algorithm equals $O(e \log e)$. Another algorithm that can be used for the same purpose is Prim's, it has a runtime complexity of $O(n^2)$ [Prim's algorithm can be implemented in $O(e + n \log n)$ time using more advanced data structures (in particular, Fibonacci heaps), this will not be worth the trouble unless you have extremely large, fairly sparse graphs]. Thus, Prim's algorithm is faster on dense graphs, while Kruskal's is faster on sparse graphs. Prim's algorithm was also invented earlier by Jarnick (1930).

ALGORITHM 8.2 (Prim's algorithm (1957)): For a connected and weighted graph G^α of order n with $V(G) = \{v_1, \ldots, v_n\}$, define $f(v_1) = 0$ and $f(v_i) = \alpha(v_1 v_i)$ if $v_1 v_i \in E(G)$ and $f(v_i) = \infty$ otherwise ($i > 1$):

1. Let $E = \emptyset$ and set $U = \{v_1\}$.
2. Choose $w \in V(G) - U$ with $f(w)$ minimal.
3. Replace E by $E \cup \{e\}$, where e is an edge incident to w and U for which $\alpha(e) = f(w)$ and set $U = U \cup \{w\}$. If $U = V(G)$ stop.

[6] At each step $i = 2, \ldots, n-1$ we can always find at least one edge that does not create a cycle. Indeed, at step i, the graph $(V(G), E_{i-1} = \{e_1, \ldots, e_{i-1}\})$ has n nodes and less than $n-1$ edges, thus, it contains at least two components. As G is connected there has to exist a path P in G that connects these components. Any edge of this path can be added without creating a cycle.

4. For each $v \notin U$ for which $wv \in E(G)$: $f(v) = \min\{f(v), \alpha(wv)\}$. Return to Step 2.

The final outcome is $T = (U, E)$ is a spanning tree of G with minimal weight. Notice, $f(v)$ equals the minimal weight of any edge connecting v with U ($f(v)$ equals ∞ if there is no such edge). Thus, at each iteration we simply add a new edge e to T with a weight $\alpha(e)$ as small as possible.

Proof: Let G^α be a connected, weighted graph. By induction we show that $T = (U, E)$ is a spanning tree on the vertices of U during each iteration. At step 1 we have $n_T = 1$, $e_T = 0$, hence, it is a spanning tree on $\{v_1\}$. At every other iteration we augment n_T and e_T by one, meaning $n_T = e_T + 1$ and T is connected.
Let T_1 be any spanning tree for G. Let e be the first edge not belonging to T_1 that was added when $T = (U, E)$ was constructed. Then one endpoint of e was in U and the other was not. Since T_1 is a spanning tree of G, there is a path in T_1 joining the two endpoints of e. As one travels along the path, one must encounter an edge f joining a vertex in U to one that is not in U. Now, at the iteration when e was added to E, f could also have been added and it would be added instead of e if $\alpha(f) < \alpha(e)$. Since f was not added, we conclude that $\alpha(f) \geq \alpha(e)$. Let $T_2 = T_1 + e - f$, then $\alpha(T_2) \leq \alpha(T_1)$. Repeating the above procedure, we can transform T_1 to T by replacing edges, one by one, such that the total weight does not increase. We deduce that $\alpha(T) \leq \alpha(T_1)$.
Q.E.D.

Many other algorithms exist to solve this problem, e.g., Boruvka's algorithm (1926) with runtime $O(e \log n)$. A faster algorithm due to Karger, Klein and Tarjan runs in $O(n)$ time.

EXERCISES 8.1: On Kruskal's and Prim's Algorithm:

1. Apply Kruskal's Algorithm to the graph G depicted in Figure 8.

2. Apply Prim's Algorithm to the graph G depicted in Figure 8.

Minimum weight spanning trees are useful in a variety of disciplines. For instance, minimum spanning trees are useful in constructing networks, by describing the way to connect a set of sites using the smallest total amount of wire. Much of the work on minimum spanning (and related Steiner) trees

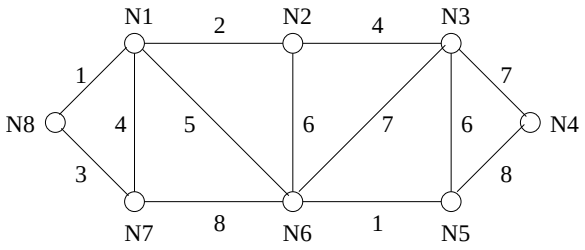

Fig. 8: Apply Kruskal's and Prim's algorithm on the graph G

has been conducted by the phone company. They also provide a reasonable way for clustering points in space into natural groups. When the cities are points in the Euclidean plane, the minimum spanning tree provides a good heuristic for traveling salesman problems. The optimum traveling salesman tour is at most twice the length of the minimum spanning tree.