Summary on Lecture 21, February 29, 2016

Optimal spanning trees

1. Warm-up: spanning trees for a regular (non-weighted) graph. First, we would like to understand how to find just a spanning tree of a given graph without any special restrictions. By itself, it is an important problem. Let $G = (V(G), E(G))$ be a graph, and we fix a vertex $v \in V(G)$. Here is the first algorithm which starts with the vertex v and gives a spanning tree of G:

```
**Tree**$(G, v)$
Input: A vertex $v$ of the finite graph $G$
Output: A set $E$ of edges of a spanning tree for the component of $G$ that contains $v$
Let $V := \{v\}$ and $E := \emptyset$
   (where $V$ is a list of visited vertices).
while there are edges of $G$ joining vertices in $V$ to vertices that are not in $V$ do
   Choose such an edge $\{u, w\}$ with $u \in V$ and $w \notin V$.
   Put $w$ in $V$ and put the edge $\{u, w\}$ in $E$.
return $E$
```

Exercise. Use the algorithm **Tree**(G, v) for the following graph, where the vertex v is labeled by 1:

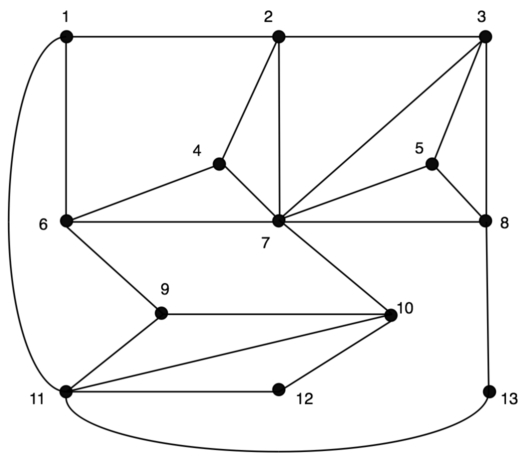

Fig. 1.

We emphasize that there is no requirement that G is a connected graph. In a general case, the algorithm **Tree**(G, v) finds a spanning tree for the component of G that contains the initial vertex v. To get a *spanning forest* for G, we just keep growing trees. Here is the algorithm how to make this work:

```
**Forest**$(G)$
Input: A finite graph G.
Output: A set $E^*$ of edges of a spanning forest for $G$
Set $V^* := \emptyset$, and $E^* := \emptyset$.
while $V^* \neq V(G)$ do
   Choose $v \in V(G) \setminus V^*$.
   Let $E$ be the edge set for the tree **Tree**$(G, v)$ spanning the
   the component of $G$ containing $v$, and let $V$ be its vertex set.
   Put the members of $V$ in $V^*$ and put the members of $E$ in $E^*$.
return $E^*$
```

Exercise. Use the algorithm **Forest** (G) for the following graph:

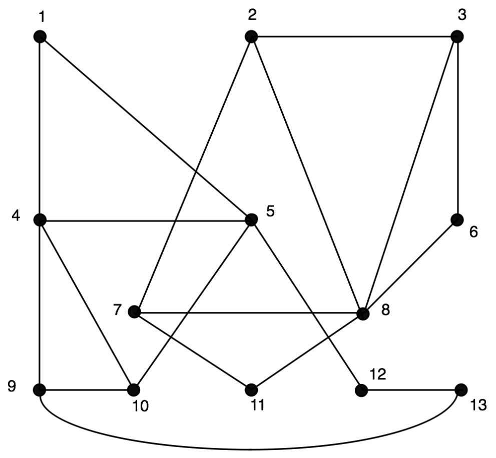

Fig. 2.

Theorem 1. *Let $G = (V(G), E(G))$ be a finite graph, and $v \in V(G)$. Then* **Tree** (G, v) *is a spanning tree for the component of G containing v. Hence the algorithm* **Forest** (G) *produces a spanning forest for G.*

Proof. First, we notice that each pass through the `while` loop increases the size of V, so the algorithm must stop eventually since G is a finite graph.

Consider the statement:

$\mathbf{S} :=$ `‘‘the sets` V `and` E `are the vertices and edges of a tree with` v `as vertex’’`

Clearly, the statement **S** is true when we first enter the loop. We show that the statement **S** is an invariant of the loop, so when the algorithm stops, it stops with a tree containing v.

Indeed, we have seen before that attaching a new vertex to a tree with a new edge yields a tree.[1] Since this is precisely what the while loop does, the statement **S** remains true after passing through the loop, i.e., it is a loop invariant. It remains to show that the algorithm does not stop as long as there are vertices in the component V' containing the vertex v.

Suppose that there is a vertex $v' \in V' \setminus V$ at the end of a `while` loop. There will be a path in V' from v to v', and along this path there will be a first edge $\{u, w\}$ with initial vertex $u \in V$ and terminal vertex $w \in V' \setminus V$ (the vertex u might be v, and w might be v'). Hence the guard

`‘‘there are edges of` G `joining vertices in` V `to vertices that are not in` V`’’`

still holds, and the algorithm does not stop. $\square$

Remark. The time that the algorithm **Tree** (G, v) takes depends on the scheme for making choices and on how the list of available edges is maintained. If we mark each vertex when we choose it and, at the same time, tell its neighbors that it's marked, then each vertex and each edge are handled only once. If G is connected, then **Tree** (G, v) builds a spanning tree in time $O(|V(G)| + |E(G)|)$. The same argument shows that, in the general case, Forest also runs in time $O(|V(G)| + |E(G)|)$. $\lozenge$

Remark. The algorithm **Forest** (G) can be used to test connectivity of a graph G; just check whether the algorithm produces more than one tree. Forest can also be used to give a relatively fast test for the presence of cycles in a graph. If G is acyclic, then the spanning forest produced is just G itself; otherwise, $|E^*| < |E(G)|$ at the conclusion of the algorithm. $\lozenge$

[1] Prove it!