

# The Minimum Spanning Tree Problem (4.5)

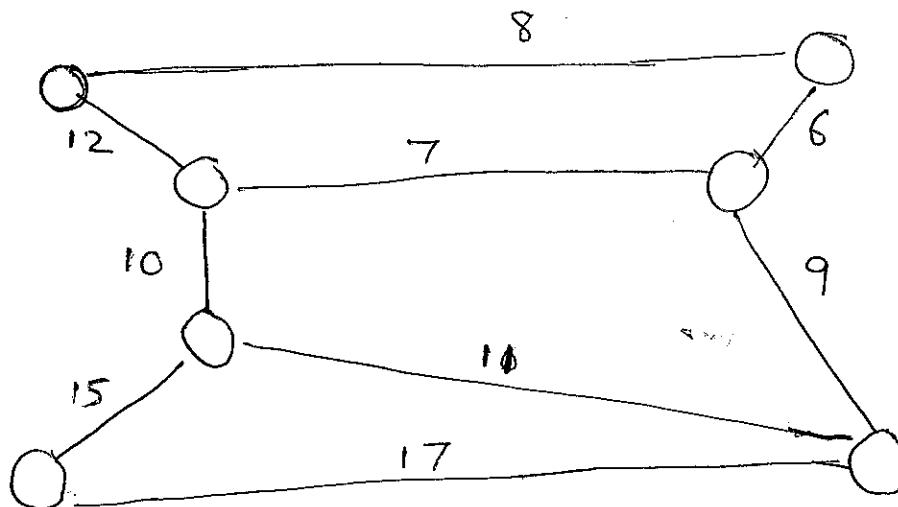
We are given a graph  $G = (V, E)$

$V = \{v_1, v_2, \dots, v_n\}$  is the set of vertices

$E$  is the set of edges - each element of  $E$  is of the form  $(v_i, v_j)$ , an unordered pair.

We are given for each edge

$e = (v_i, v_j)$  a cost which we denote by  $c_e$  or  $c(v_i, v_j)$ .



Let's imagine that vertices are locations,  
 $e = (v_i, v_j)$   
and the cost of an edge  $e$  is the  
cost of laying a direct connection  
from  $v_i$  to  $v_j$ .

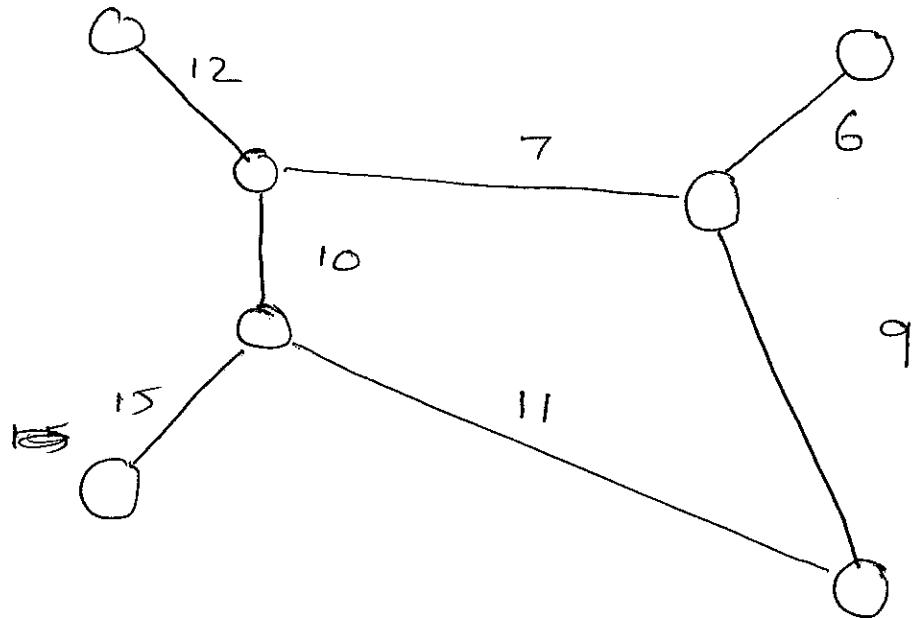
Our goal is to lay a set of  
direct connections so that the resulting  
network is connected, that is, it  
has a path between any pair of vertices.  
We want to do this in a way that  
minimizes cost.

In other words: We want to find the  
subset  $T \subseteq E$  of minimum cost  
 $c(T) = \sum_{e \in T} c_e$  so that subgraph  
 $\textcircled{B} (V, T)$  is connected.

Of course, we assume that  $G = (V, E)$   
is connected.

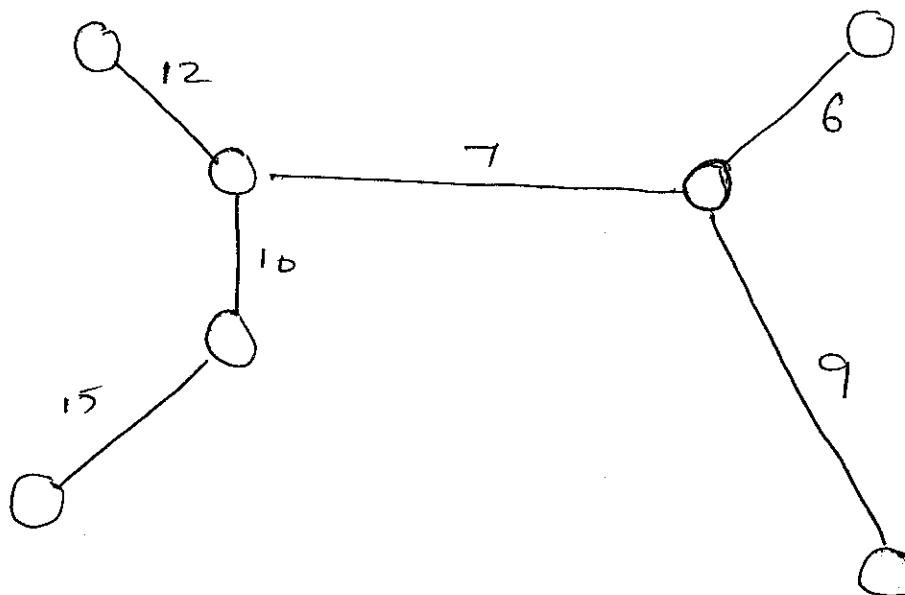
This is the minimum spanning subgraph problem.

Here is one spanning subgraph in the example:



$$\text{Its cost} = 12 + 10 + 15 + 7 + 11 + 6 + 9$$

Clearly, there is a lower cost solution:



We simply eliminated the cycle. This illustrates a general phenomenon:

Claim Let  $T$  be a min-cost solution to the minimum spanning subgraph problem.

Then  $(V, T)$  does not have a cycle.

For this reason, the problem is also called the minimum spanning tree problem.

(Spanning tree = Connected + acyclic)

Here are some heuristics for the problem.

- ① Consider edges of  $E$  in order of increasing cost, and add an edge if it is useful.

In particular:

Sort  $E$  by increasing cost. Suppose sorted set is  $e_1, e_2, \dots, e_m$ .

$T \leftarrow \emptyset$

For  $i \leftarrow 1$  to  $m$  do

If adding  $e_i$  to  $T$  does not

create a cycle in  $(V, T)$ , add

$e_i$  to  $T$

end for

Return  $T$ .

② Starting from some vertex  $s \in V$ , grow a connected component in a greedy way.

In particular:

$S \leftarrow \{s\}$

$T \leftarrow \emptyset$

while ( $S \neq V$ ) do

    Find the cheapest edge connecting

$S$  to  $V-S$ . Suppose this is

$e = (u, v)$  where  $u \in S$ ,  $v \in V-S$ .

    Add  $e$  to  $T$ , and  $v$  to  $S$

endwhile

Return  $T$ .

Examples of both greedy algorithms in class. Both algorithms actually work - the first is called Kruskal's algorithm, and the second is called Prim's. Notice that there are data structural issues we need to address in both algorithms. We'll talk about

there later. First we'll show that both algorithms work. Let us fix  $\text{OPT}$ , one optimal solution to the ~~ps~~ minimum spanning subgraph problem on  $G = (V, E)$ .

### Correctness of Kruskal's algorithm:

(1) We first argue that  $(V, T)$ , is connected, where  $T$  is the set of edges returned by Kruskal. This is relatively easy.

(2) We'll show by induction on  $i$  that:

If  $e_i$  is added to  $T$  by Kruskal, then  $e_i \in \text{OPT}$ .

This requires some more work. Details in class.

(1) implies that  $(V, T)$  is connected, and (2) implies that  $c(T) \leq c(OPT)$ . So  $T$  is optimal.

Correctness of Prim's: Along some lines.

- (1) Argue that  $(V, T)$  is connected for the final  $T$ . This is almost obvious
- (2) Argue that if an edge is added to  $T$ , it must be in  $OPT$ .