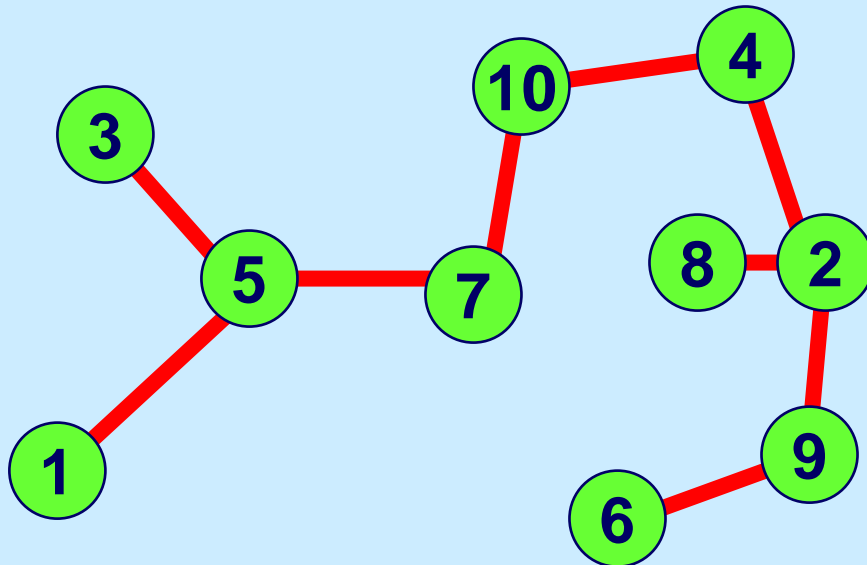

15.082 and 6.855

**The Minimum Cost Spanning Tree
Problem**

Communications Systems

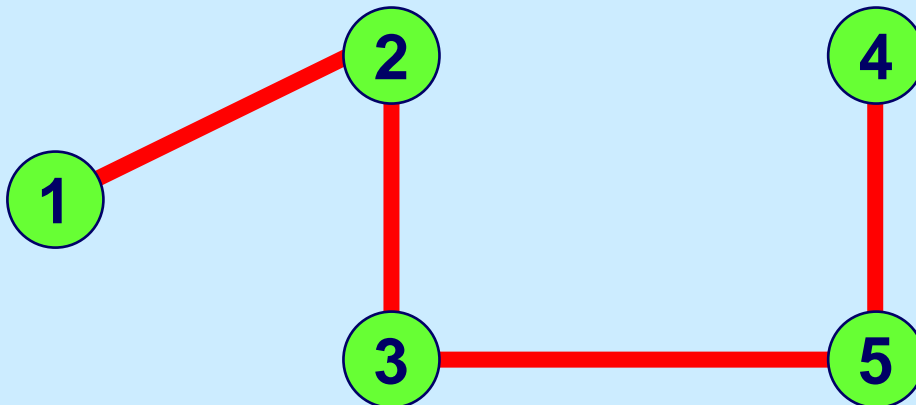
Consider a communications company, such as AT&T or GTE that needs to build a communication network that connects n different users. The cost of making a link joining i and j is c_{ij} . What is the minimum cost of connecting all of the users?



Common assumption: the only links possible are the ones directly joining two nodes.

Electronic Circuitry

Consider a system with a number of electronic components. In order to make two pins i and j of different components electrically equivalent, one can connect i and j by a wire. How can we connect n different pins in this way to make them electrically equivalent to each other so as to minimize the total wire length.



Minimum Cost Spanning Tree Problem

Undirected network $G = (N, A)$.

(i, j) is the same arc as (j, i) .

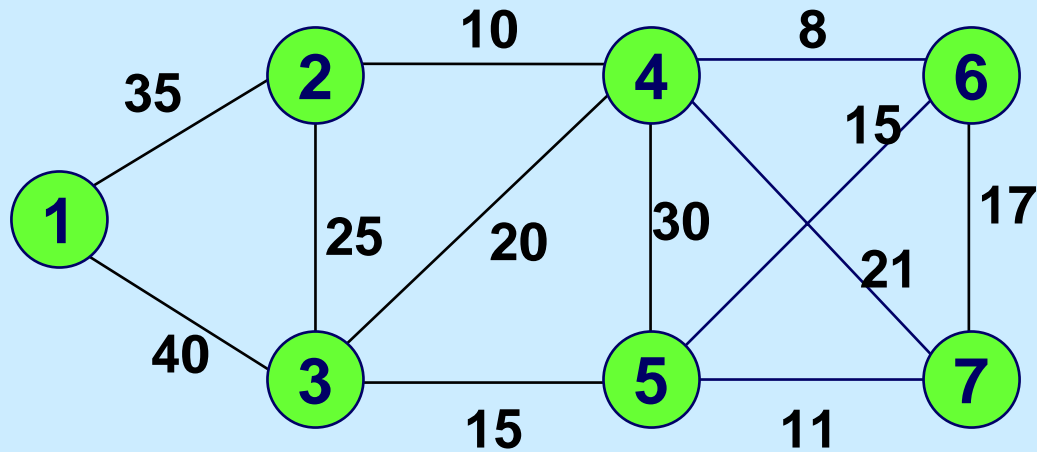
We associate with each arc $(i, j) \in A$ a cost c_{ij} .

A **spanning tree** T of G is a connected acyclic subgraph that spans all the nodes.

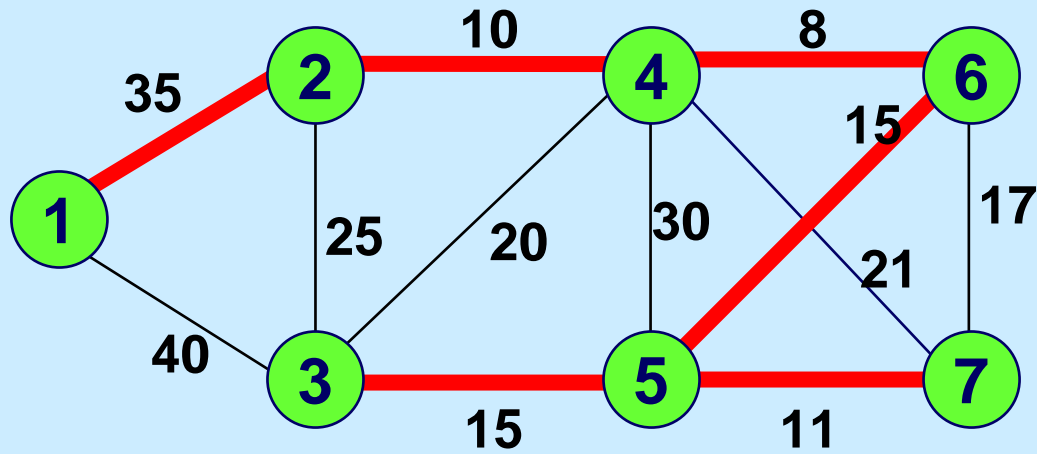
A connected graph with n nodes and $n - 1$ arcs is a spanning tree.

The minimum cost spanning tree problem is to find a spanning tree of minimum cost.

A Minimum Cost Spanning Tree Problem

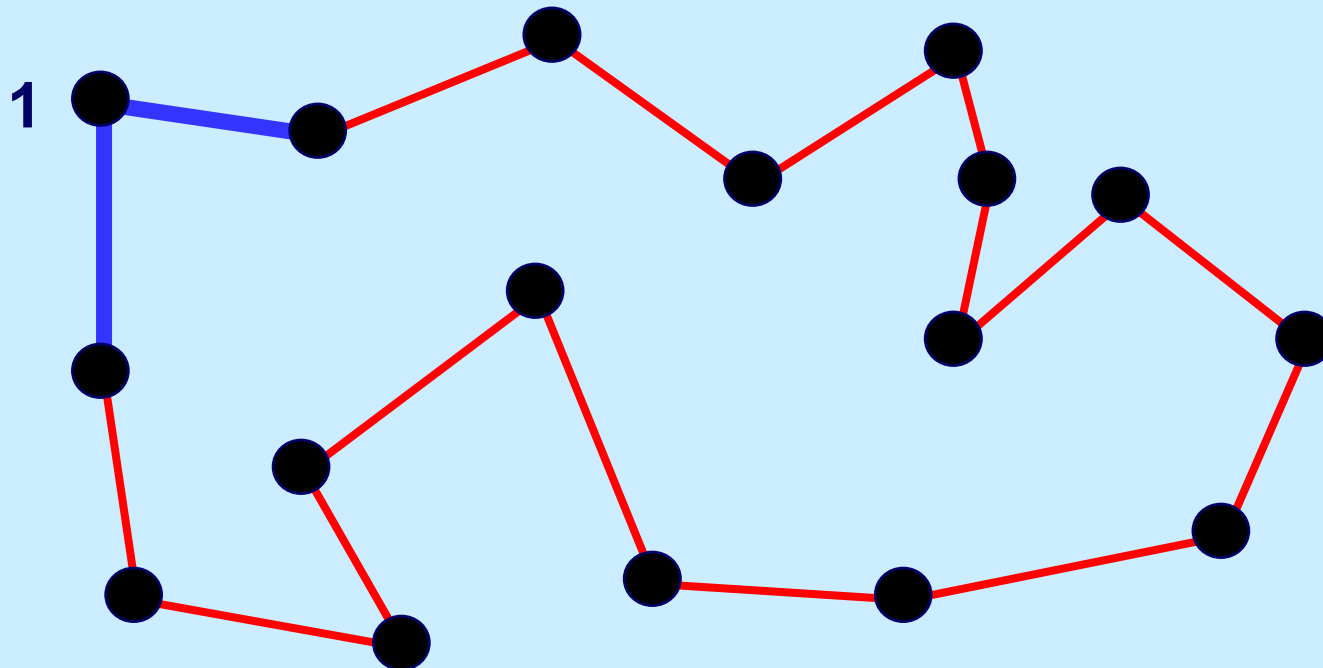


A Minimum Cost Spanning Tree



The Traveling Salesman Problem

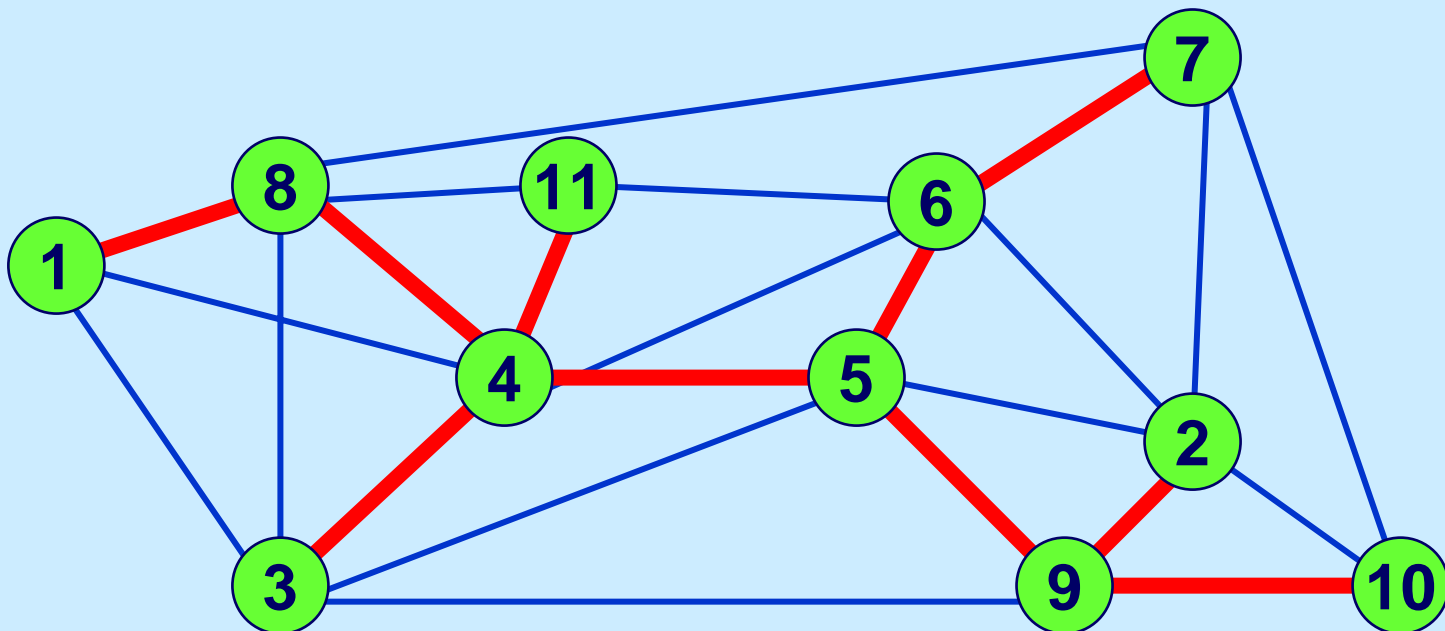
- ◆ Consider the traveling salesman problem of finding a minimum cost tour linking n cities.
- ◆ One way of formulating this problem is using minimum spanning trees.



Definitions

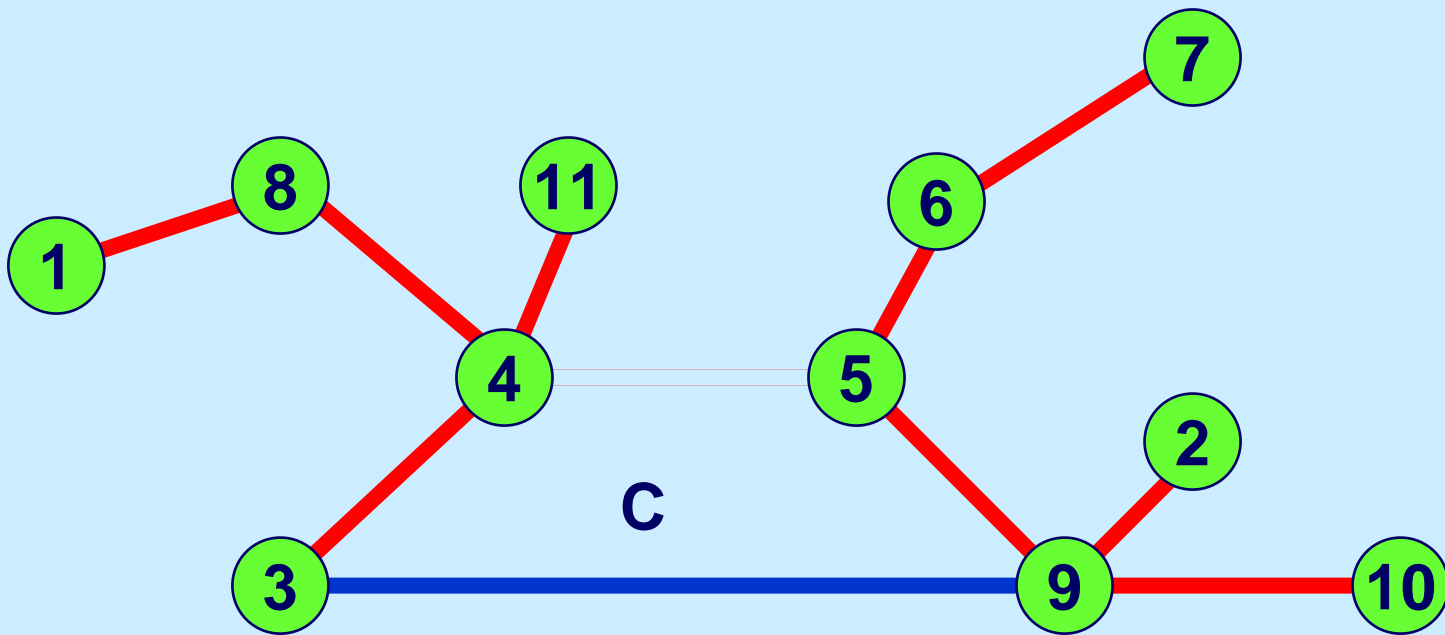
- ◆ Let T^* be a spanning tree. We refer to the arcs in T^* as *tree arcs* and to those arcs not in T^* as *nontree arcs*.

The thin blue lines below are all non-tree arcs.



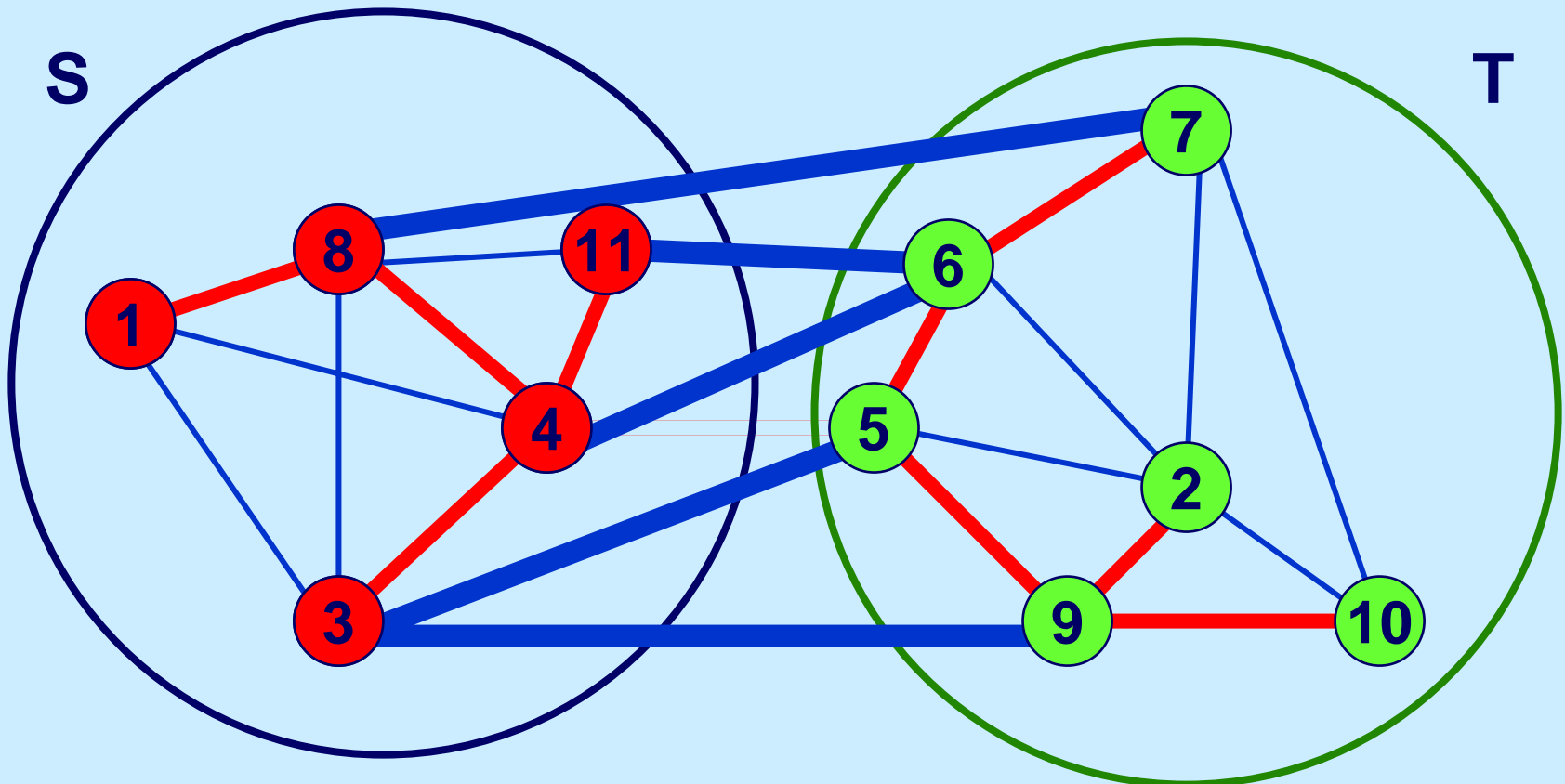
Lemma

Lemma. Let T be any spanning tree. Let a be any arc not on the tree. Then adding a to T creates a unique cycle C . Deleting any arc of C creates a spanning tree T' . (left as exercise).



Definitions

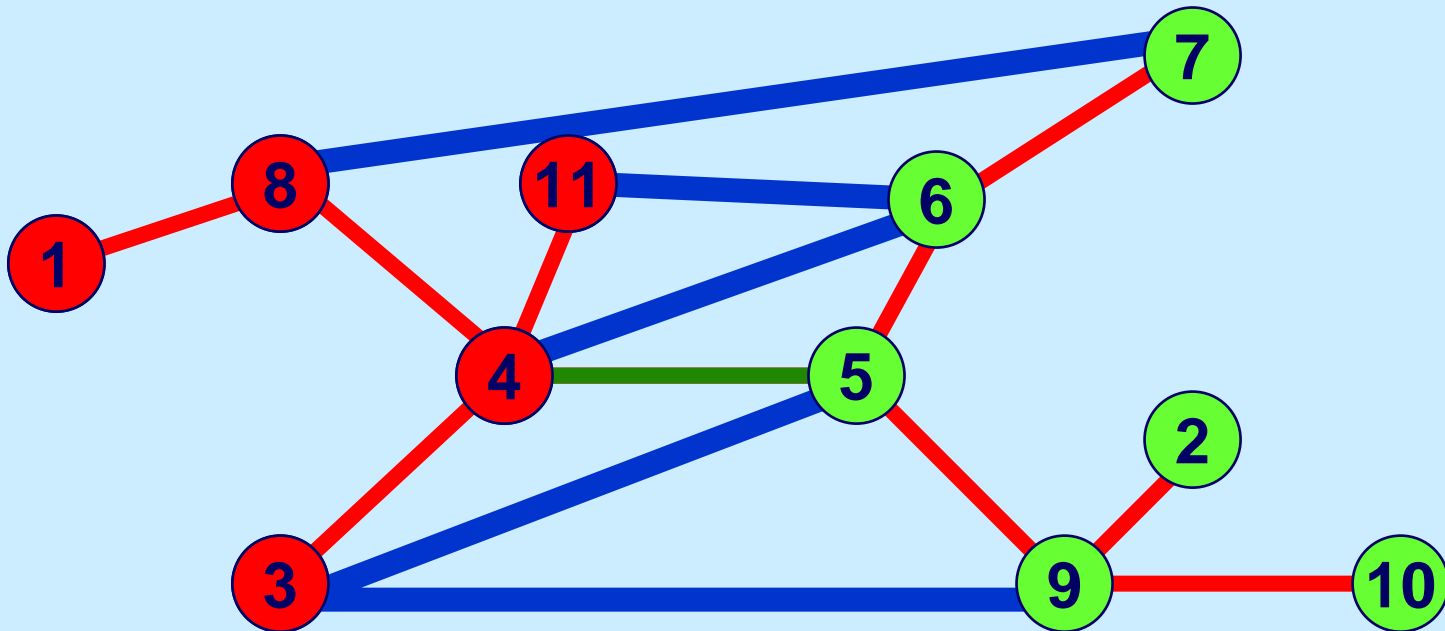
- ◆ If (i,j) is a tree arc, then $T^*(i,j)$ has two components, say with node sets S and $N-S$. The arcs in $(S,N-S)$ constitute a *cut*. For example, suppose that $(i,j) = (4, 5)$.



Cut Optimality Conditions

Cut optimality conditions. For every tree arc $(i, j) \in T^*$, $c_{ij} \leq c_{kl}$ for every arc (k, l) contained in the cut formed by deleting arc (i, j) from T^* .

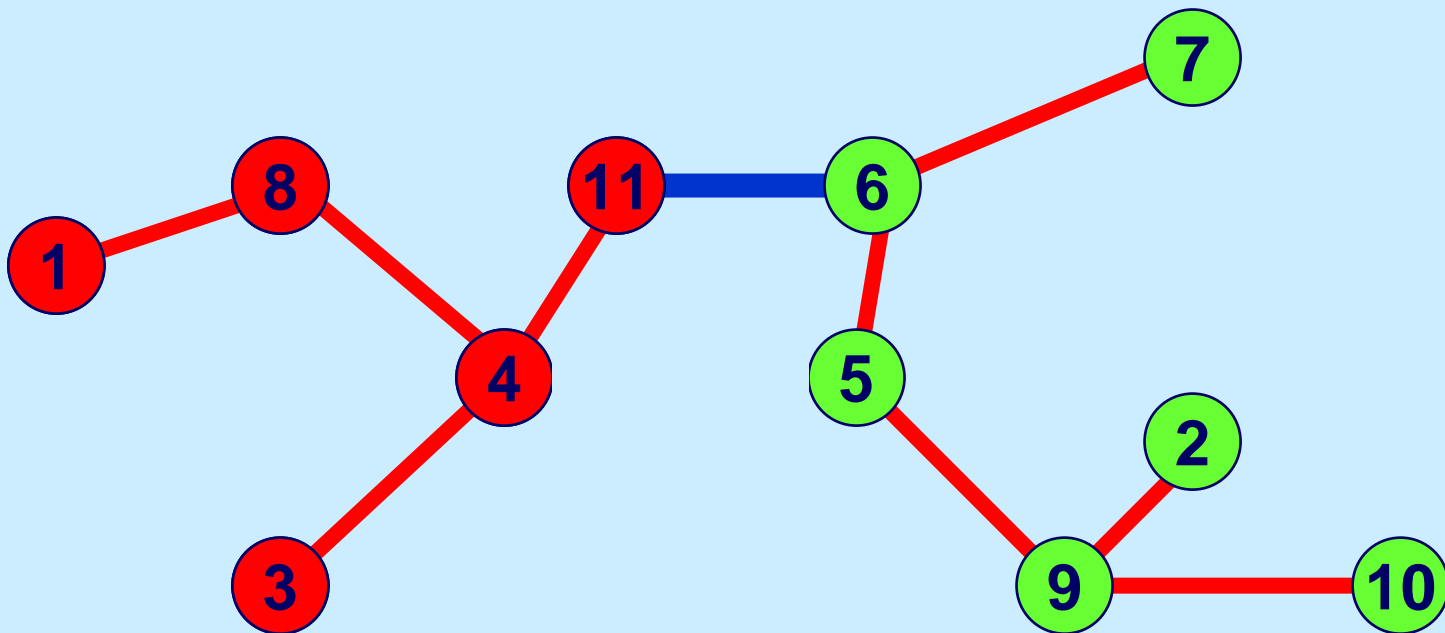
Example. Consider the cutset induced by deleting $(4,5)$. The cut optimality conditions are satisfied if every arc in the cut set has length at least that of $(4,5)$.



Cut Optimality Theorem

Theorem. A spanning tree T^* is a minimum spanning tree if and only if it satisfies the **cut optimality conditions**.

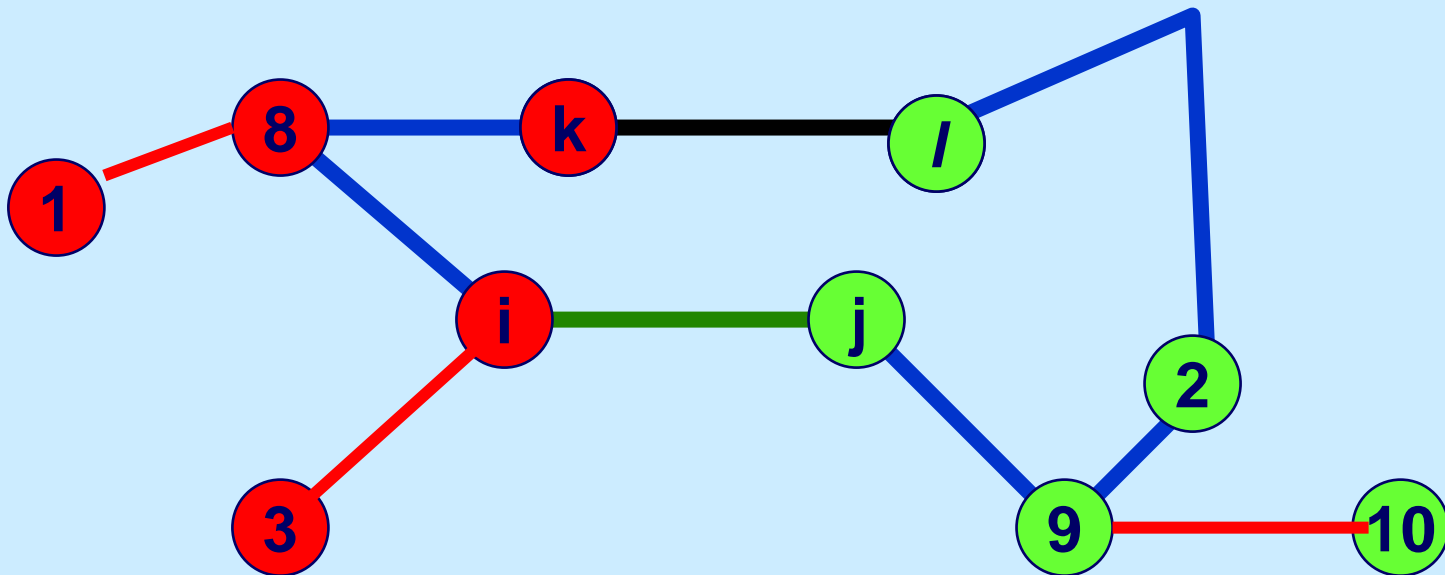
Proof. Suppose first that $c_{ij} > c_{kl}$ for some arc (k,l) in the cutset obtained from deleting (i,j) from T^* . Then $T' = T^* - (i,j) + (k,l)$ is a spanning tree with less cost.



Conversely, suppose that T^* satisfies the cut optimality conditions. Let T' be an optimal spanning tree, and among all optimal spanning trees let T' be the one with the most arcs in common with T^* . Let $(i,j) \in T^*$. We claim that $(i,j) \in T'$. Suppose that this were not true.

Let P be the path from i to j in T' . Then P has at least one arc (k,l) in the cutset induced by deleting (i,j) .

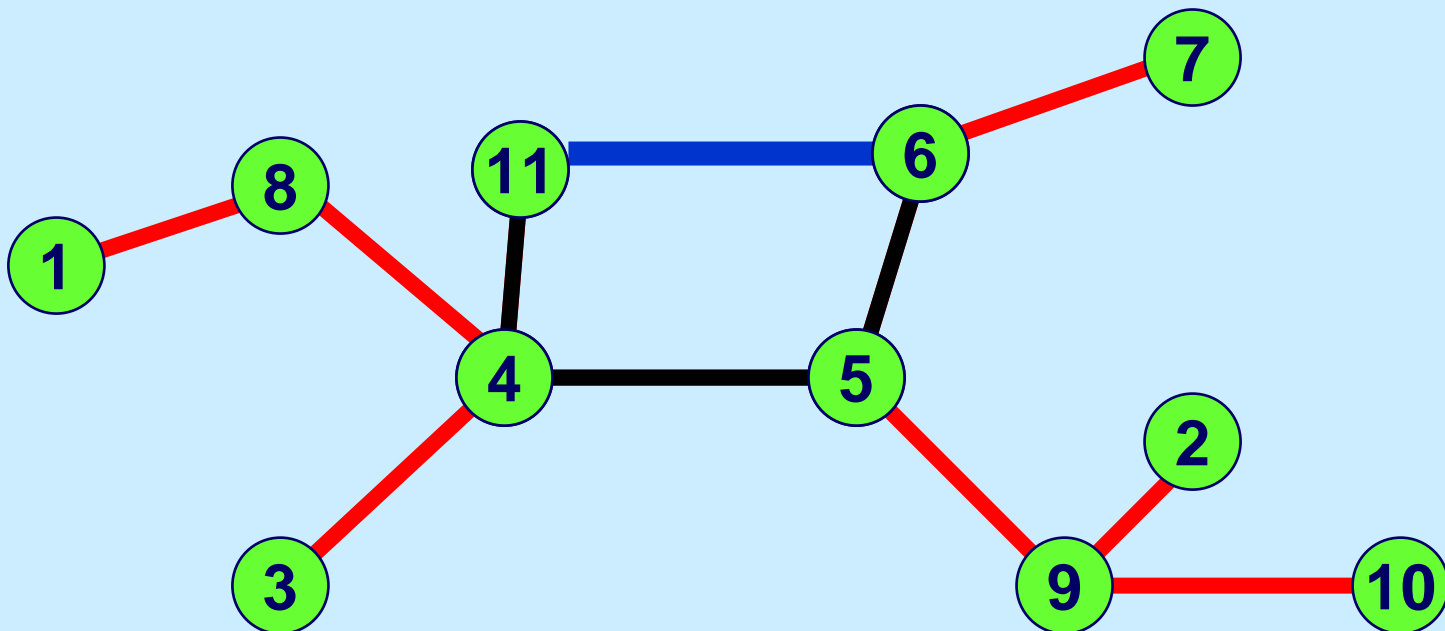
But then $T' - (k,l) + (i,j)$ is also a min cost tree, which is a contradiction.



Path Optimality Conditions

Path optimality conditions : For every nontree arc (k, l) of G , $c_{ij} \geq c_{kl}$ for every arc (i, j) contained in the path of T^* connecting nodes k and l .

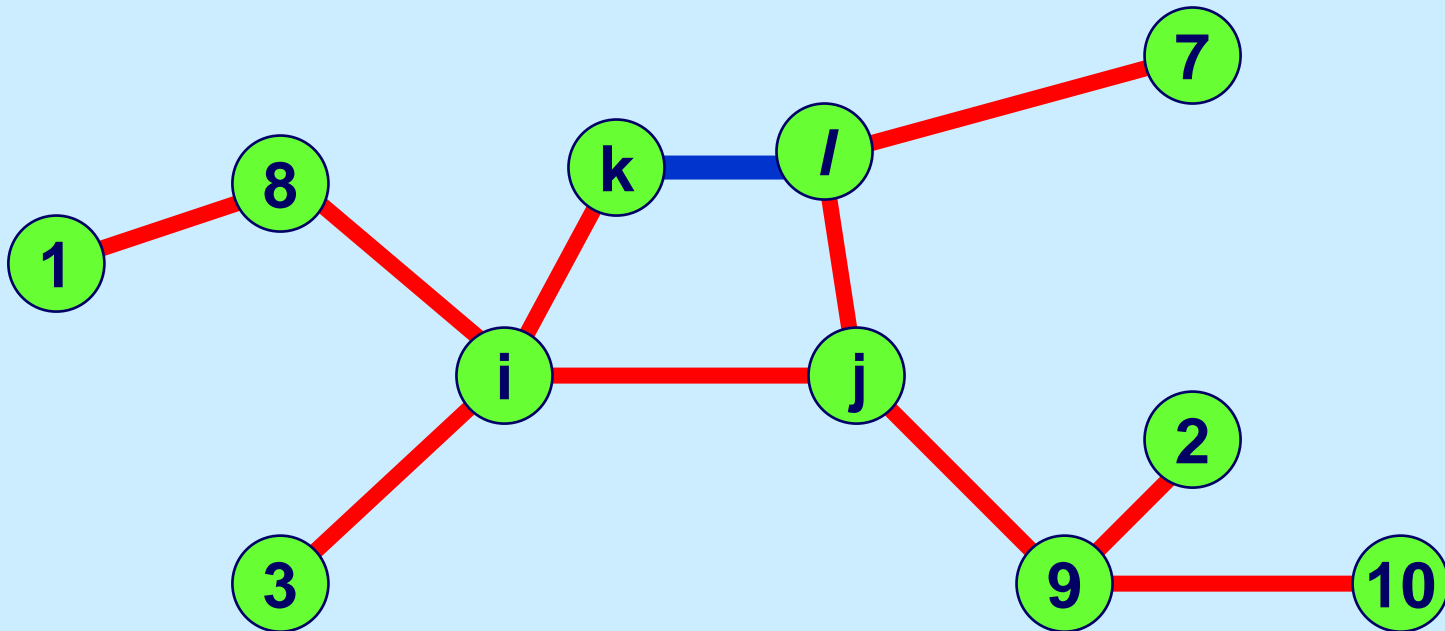
Example. Every arc in the path from 11 to 6 in T^* has length at least that of $(11,6)$



Path Optimality Theorem

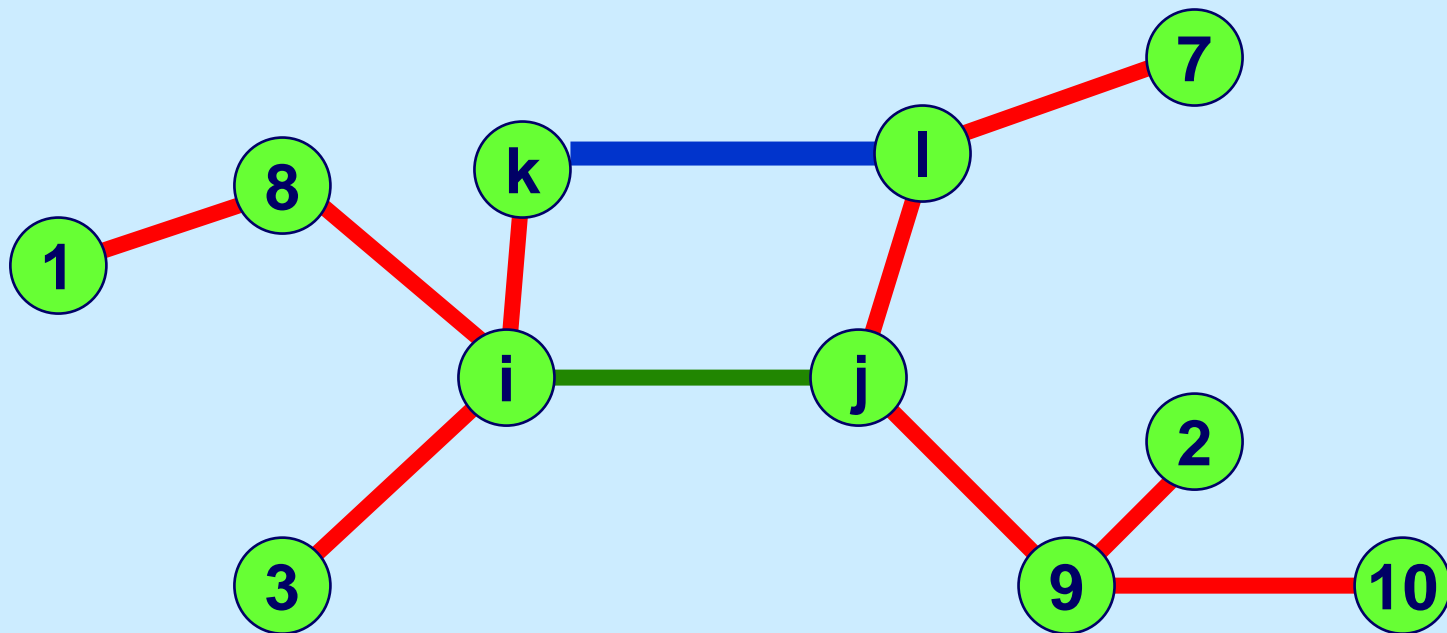
Theorem 13.2. (Path optimality theorem) *A spanning tree T^* is a minimum spanning tree if and only if it satisfies the path optimality conditions.*

Proof. Suppose first that the path optimality conditions are not satisfied. Suppose $c_{ij} > c_{kl}$. Then $T' = T^* - (i,j) + (k,l)$ has lower cost than T^* .



Observation. Let (k,l) be an arc in $A-T^*$. Then (i,j) is on the cycle contained in $T^* + (k,l)$ if and only if (k,l) is in the cutset induced by $T^*-(i,j)$.

Thus, path optimality conditions are equivalent to cut optimality conditions.



Kruskal's Greedy Algorithm

begin

order the arcs a_1, \dots, a_m in non-decreasing
order of cost

$T = \emptyset$

for $i = 1$ to m **do**

if $T + a_i$ does not have a cycle then $T := T + a_i$

end

The Greedy Algorithm

Running Time Analysis

Step 1. Sort the arc costs.

$O(\min(m \log n, m \log_n C))$

Repeated Step: determine if $T + a_i$ has a cycle?

When arc (j,k) is considered, we want to know if j and k are in the same connected component. If they are not, they result in two components being merged.

Next: a very simple way of keeping track of components.

A simple, yet fairly efficient data structure

Store each component C as set.

- ◆ Let $\text{First}(C)$ denote the first node in the component of C .

For each element j in a component C , let $\text{First}(j) = \text{First}(C) =$ first node of C .

Remark: adding (i,j) to a forest F creates a cycle if and only if (i,j) are in the same component; i.e., $\text{First}(i) = \text{First}(j)$.

When merging components C and D , put the larger component first. If $|C| > |D|$, $\text{First}(C \cup D) := \text{First}(C)$.

The Greedy Algorithm with merging

Analysis of the Running Time

- ◆ Time to determine if $\text{First}(i) = \text{First}(j)$ for i, j : $O(1)$ per arc. $O(m)$ in total.
- ◆ Time to merge components S and T , assume $|S| \geq |T|$.
 - $O(1)$ per node of T (the smaller side)
 - Each node i is on the smaller side of a merge at most $\log n$ times. (Because the number of nodes in the component containing i at least doubles when it is merged.)
- ◆ Total time spent merging: $O(n \log n)$.
- ◆ Total running time.
 $O(\min (m \log n , m \log_n C + n \log n))$.

Prim's Algorithm, A Dijkstra-like Algorithm

begin

S = {1}; T = \emptyset ;

while S \neq N do

begin

find the minimum cost arc (i,j) from S to N-S;

add j to S; add (i,j) to T;

end

end

Remark. This algorithm builds a minimum cost spanning tree from node 1. The algorithm guarantees that the cut optimality conditions are satisfied at the end.

Prim's Algorithm

Analysis of Prim's Algorithm

- ◆ At each iteration, one determines the minimum arc using a priority queue.
- ◆ Running time: each arc enters the priority queue once and is deleted once.
- ◆ $O(m \log n)$ running time.
- ◆ Further improvements are possible by maintaining node labels, and using a more efficient priority queue.

Sollin's Algorithm

```
begin
  for each  $i \in N$  do  $N_i := \{i\}$ ;
   $T^* := \emptyset$ 
  while  $|T^*| < n - 1$  do
    begin
      for each tree  $N_k$  of the forest do
        find the least cost arc  $(i_k, j_k)$  from  $N_k$  to  $N \setminus N_k$  ;
        in case of ties, use a consistent tie breaking rule
        let  $T^* = T^* \cup \{(i_1, j_1), (i_2, j_2), \dots, \}$ 
    end
  end
```

At each iteration, the size of the smallest component at least doubles, and so there are $O(\log n)$ iterations. A straightforward implementation runs in $O(m \log n)$ time or $O(n \log n) +$ time to sort m arcs.

Summary

- ◆ **There are several algorithms for finding a minimum cost spanning tree**
 - **Kruskal's Algorithm (a greedy algorithm)**
 - **Prim's Algorithm (a Dijkstra-type algorithm)**
 - **Sollin's Algorithm (is great for parallel implementations)**
- ◆ **Proof of correctness relies on Path or Cut optimality conditions**
- ◆ **Proof of running times depend on good data structures, and careful analysis.**
- ◆ **Bottom Line: There is an algorithm for finding a minimum cost spanning tree that runs in $O(m)$ time. It is very complex and not practical.**