Minimum Spanning Trees

Assistant Professor Mr. Vidhu Vohra
PG Deptt of Computer Science & IT

In the design of electronic circuitry, it is often necessary to make the pins of several components electrically equivalent by wiring them together. To interconnect a set of n pins, we can use an arrangement of $n - 1$ wires, each connecting two pins. Of all such arrangements, the one that uses the least amount of wire is usually the most desirable. We can model this wiring problem with a connected, undirected graph $G = (V, E)$, where V is the set of pins, E is the set of possible interconnections between pairs of pins, and for each edge $(u, v) \in E$, we have a weight $w(u, v)$ specifying the cost (amount of wire needed) to connect u and v.

We then wish to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

is minimized. Since T is acyclic and connects all of the vertices, it must form a tree, which we call a ***spanning tree*** since it "spans" the graph G. We call the problem of determining the tree T the ***minimum-spanning-tree problem***.

Graph: set of "objects" and their "connections"

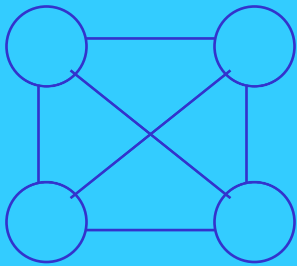
Formal definition:

- $G = (V, E)$, $V=\{v_1, v_2, \ldots, v_n\}$, $E=\{e_1, e_2, \ldots, e_m\}$
- V: set of vertices (nodes), E: set of edges (links, arcs)
- Directed graph: $e_k = (v_i, v_j)$
- Undirected graph: $e_k=\{v_i, v_j\}$
- Weighted graph: $w: E \rightarrow R$, $w(e_k)$ is the "weight" of e_k.

Weighted graphs

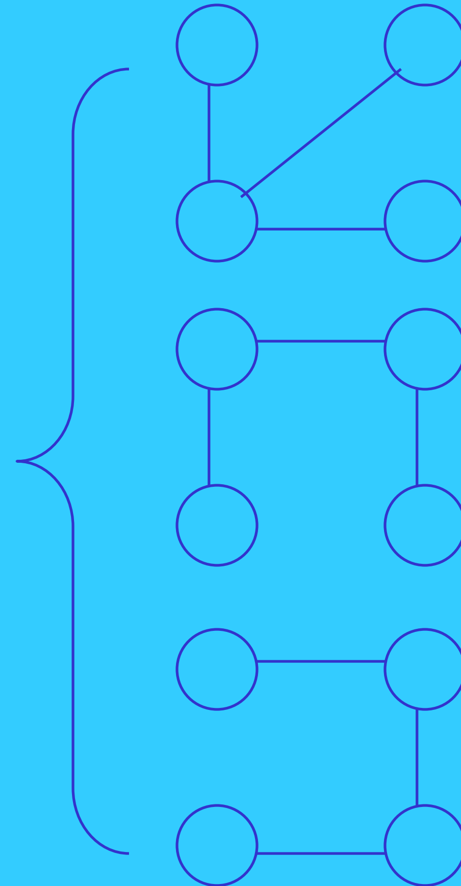
- **each edge e has an integer *weight*, wt(e)**
  - **graph may be undirected or directed**
  - **weight may represent length, cost, capacity, etc**
  - **adjacency matrix becomes *weight matrix***
  - **adjacency lists include weight in node**

Undirected graph and 3 of its spanning trees



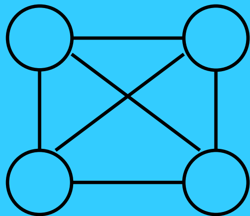
Undirected Graph

Spanning Trees

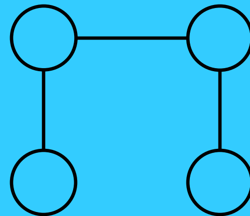
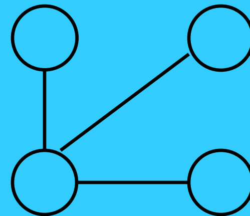
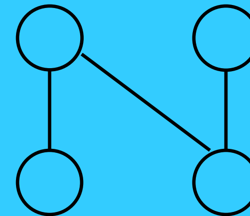
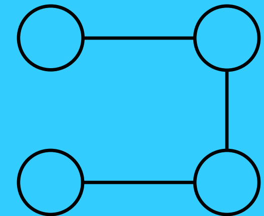


Spanning trees

- Suppose you have a connected undirected graph
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction
- ...then a spanning tree of the graph is a connected subgraph in which there are no cycles



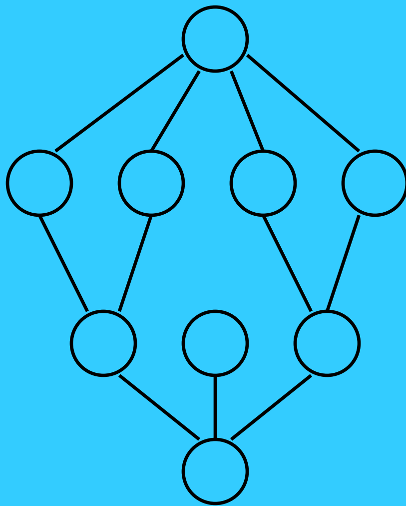
A connected, undirected graph

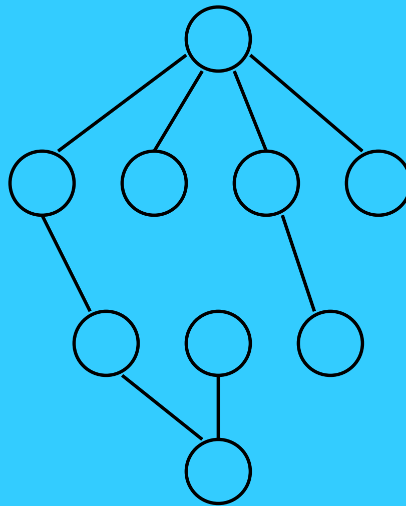
Four of the spanning trees of the graph

Finding a spanning tree

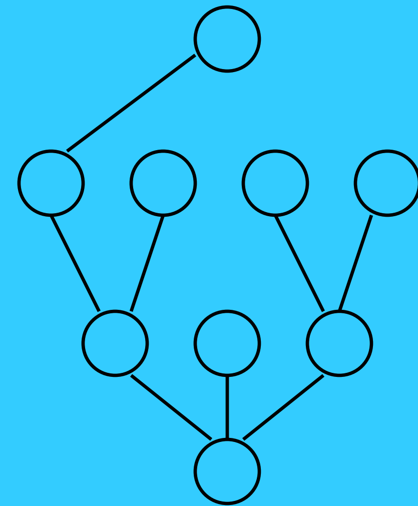
- To find a spanning tree of a graph,
  - pick a node and call it part of the spanning tree
  - do a search from the initial node:
    - each time you find a node that is not in the spanning tree, add to the spanning tree both the new node *and* the edge you followed to get to it



An undirected graph



Result of a BFS starting from top



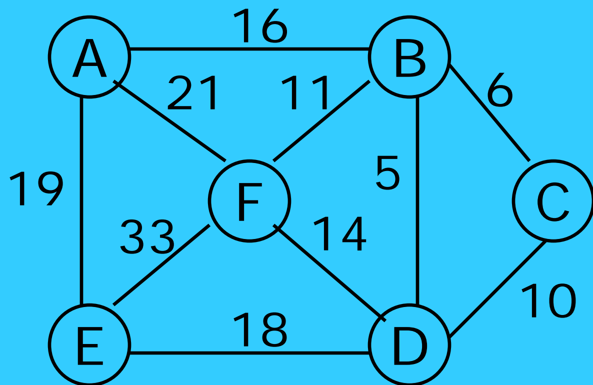
Result of a DFS starting from top

Minimizing costs

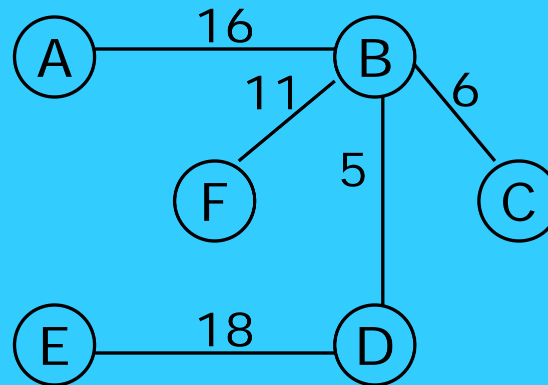
- Suppose you want to supply a set of houses (say, in a new subdivision) with:
  - electric power
  - water
  - sewage lines
  - telephone lines
- To keep costs down, you could connect these houses with a spanning tree (of, for example, power lines)
  - However, the houses are not all equal distances apart
- To reduce costs even further, you could connect the houses with a *minimum-cost* spanning tree

Minimum-cost spanning trees

- Suppose you have a connected undirected graph with a weight (or cost) associated with each edge
- The cost of a spanning tree would be the sum of the costs of its edges
- A minimum-cost spanning tree is that spanning tree that has the lowest cost



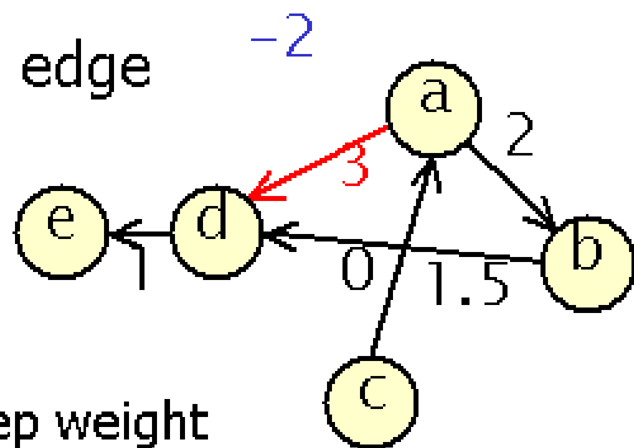
A connected, undirected graph



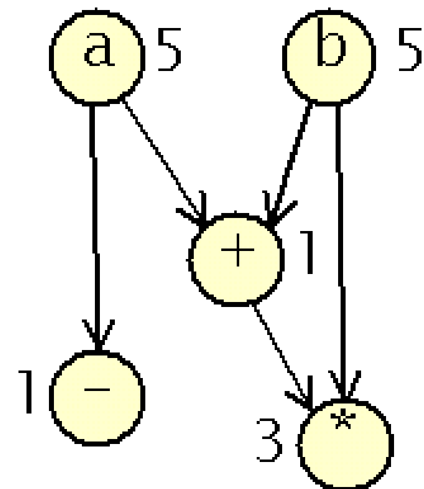
A minimum-cost spanning tree

Edge / Vertex Weights in Graphs

- Edge weights
  - Usually represent the "cost" of an edge
  - Examples:
    - Distance between two cities
    - Width of a data bus
  - Representation
    - Adjacency matrix: instead of 0/1, keep weight
    - Adjacency list: keep the weight in the linked list item



- Node weight
  - Usually used to enforce some "capacity" constraint
  - Examples:
    - The size of gates in a circuit
    - The delay of operations in a "data dependency graph"

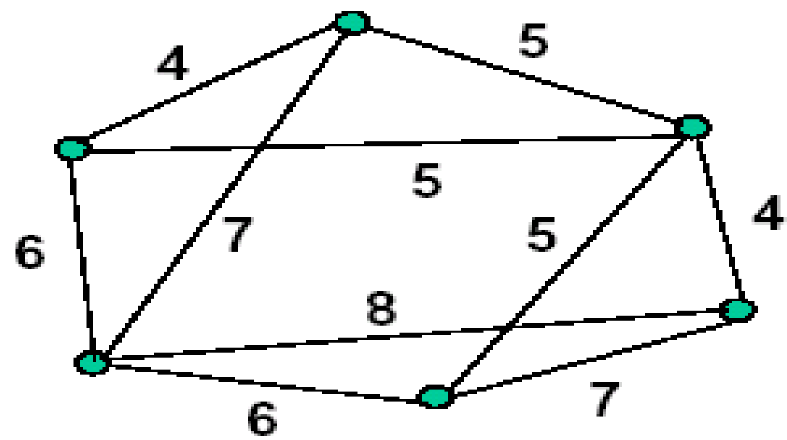


- a *spanning tree* is obtained from a connected graph by deleting edges to obtain a tree

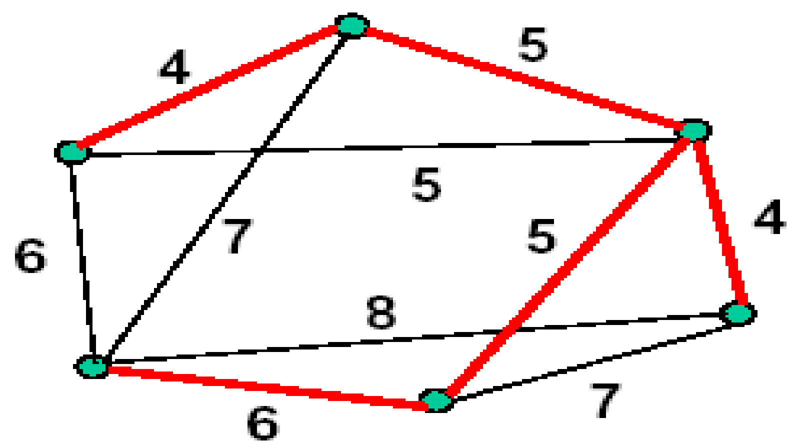
- the *weight* of a spanning tree is the sum of the weights of the edges it contains

- problem - for a weighted connected undirected graph, find a minimum weight spanning tree

  - represents the 'cheapest' way of interconnecting the nodes



a weighted graph G



a minimum weight spanning tree for G

- example of a problem in *combinatorial optimisation*
  - find the 'best' way of doing something among a (large) number of candidates

- can always be solved, in theory, by *exhaustive search*
  - but this may be infeasible
  - e.g. a clique of size n has n^{n-2} spanning trees
- a much more efficient algorithm *may* be possible (and is in this case)

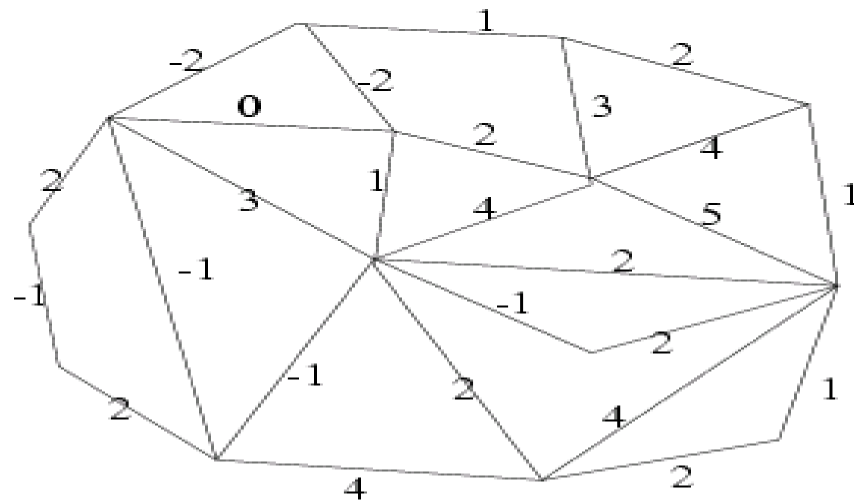
Maximum weight trees

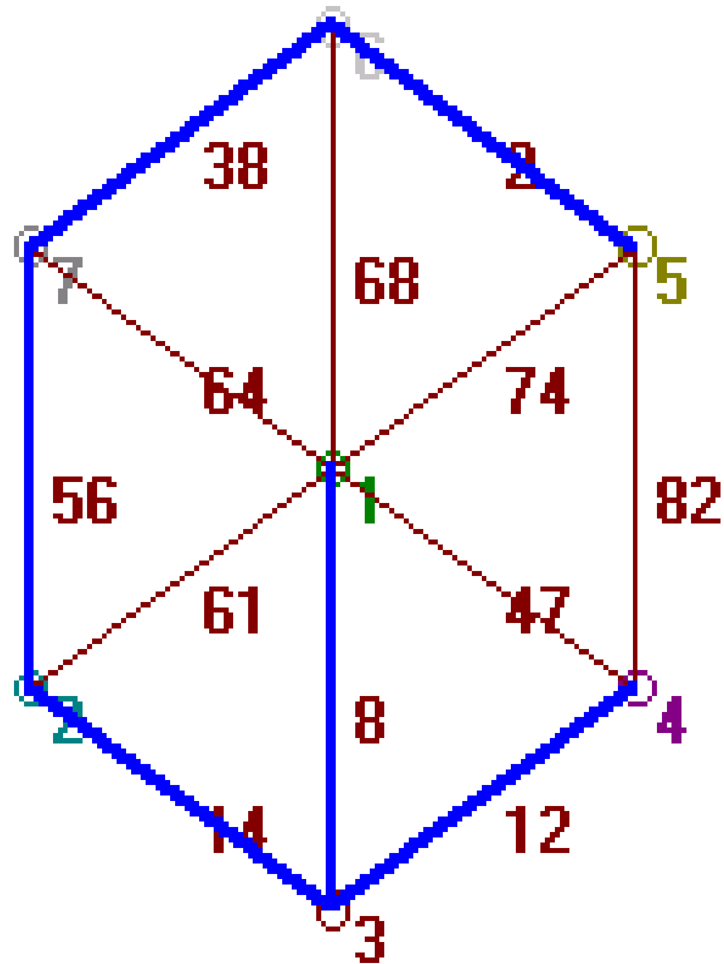
$G = (V, E)$ is a connected graph.

$w : E \to \boldsymbol{R}$. $w(e)$ is the *weight* of edge e.

For spanning tree T, $w(T) = \sum_{e \in T} w(e)$.

Problem: find a spanning tree of maximum weight.





A weighted graph with the minimal spanning tree in blue.

Minimum Spanning Tree (MST)

- Tree (usually undirected):
  - Connected graph with no cycles
  - $|E| = |V| - 1$
- Spanning tree
  - Connected subgraph that covers all vertices
  - If the original graph not tree,
    graph has several spanning trees
- Minimum spanning tree
  - Spanning tree with minimum sum of edge weights (among all spanning trees)
  - Example: build a railway system to connect N cities, with the smallest total length of the railroad

The minimum spanning tree algorithm

- an example of a *greedy* algorithm
- makes a sequence of decisions based on *local optimality*
- ends up with the *globally optimal* solution

- for many other problems, greedy algorithms do not yield optimal solutions
  - e.g. *travelling salesman problem*, finding a *largest clique* in a graph, etc.

Minimum Spanning Tree Algorithms

- Basic idea:
  - Start from a vertex (or edge), and expand the tree, avoiding loops
  - Pick the minimum weight edge at each step
- Known algorithms
  - Prim: start from a vertex, expand the connected tree
  - Kruskal: start with the min weight edge, add min weight edges while avoiding cycles (build a forest of small trees, merge them)

The *Prim-Dijkstra* algorithm

- the tree is constructed by choosing a sequence of edges

pseudocode

```
set an arbitrary vertex v to be a
  tree-vertex (tv);
set all other vertices to be
  non-tree-vertices (ntvs);
while number of ntvs > 0 loop
  find edge e = {x,y} such that
    x is a tv, y is an ntv, and
    wt(e) is minimised;
  adjoin edge e to the tree;
  make y a tv;
end loop;
```

Prim's version

- outer loop executed n-1 times
- inner loop checks all tv - ntv edges
- there are $O(n^2)$ of these
- so overall algorithm is $O(n^3)$
  (n is the number of vertices)

Dijkstra's refinement

- introduce an array best_tv
  - for each ntv x, best_tv(x) is set to the tv y for which wt({x,y}) is minimised

```
for each vertex x loop
   status(x) := ntv;
   best_tv(x) := v;
end loop;
status(v) := tv;
while number of ntvs > 0 loop
   find ntv x for which
     wt({x, best_tv(x)})is min;
   adjoin {x, best_tv(x)} to tree;
   status(x) := tv;
   for each ntv y loop
    if wt({y,x})<wt({y,best_tv(y)})
                                      then
       best_tv(y) := x;
    end if;
   end loop;
end loop;
```

Dijkstra's version

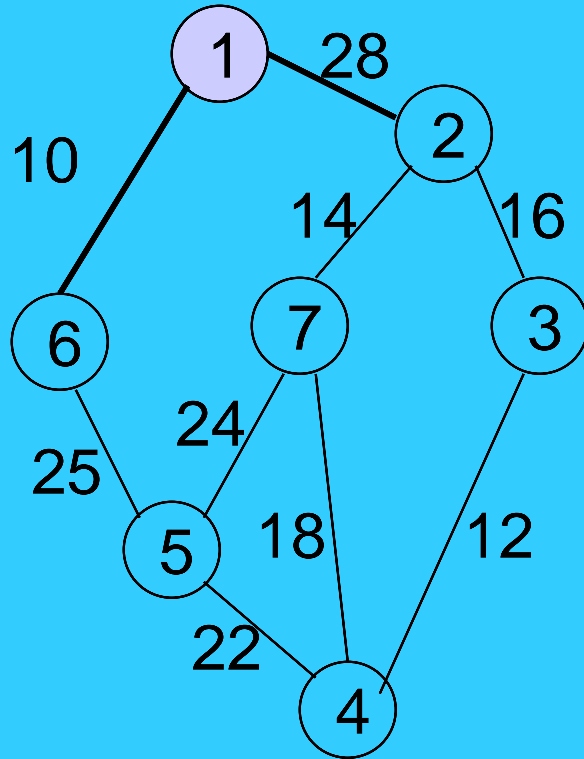
- outer loop executed n-1 times
- each inner loop is executed O(n) times
- so overall algorithm is $O(n^2)$

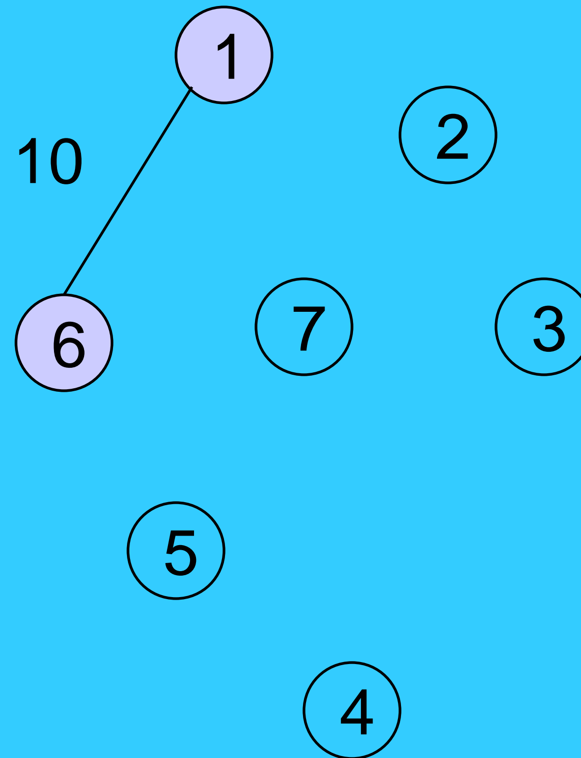
- Proof that it works
  - Suppose it doesn't - compare the tree T produced with an MST X
  - Let e be the first edge chosen to be in T that is not in X, & let S be the set of tvs at that moment
  - add e to X to give a cycle C
  - C must have another edge f that connects S to V - S, and weight of f $\geq$ weight of e (Greedy Alg chose e)
  - replacing f by e in X gives another spanning tree Y and weight of Y $\geq$ weight of X (X is an MST)
  - hence Y must have same weight as X
  - continue in the same way, thereby converting X to T without changing the weight
  - so T is also an MST

Applications of Spanning Trees

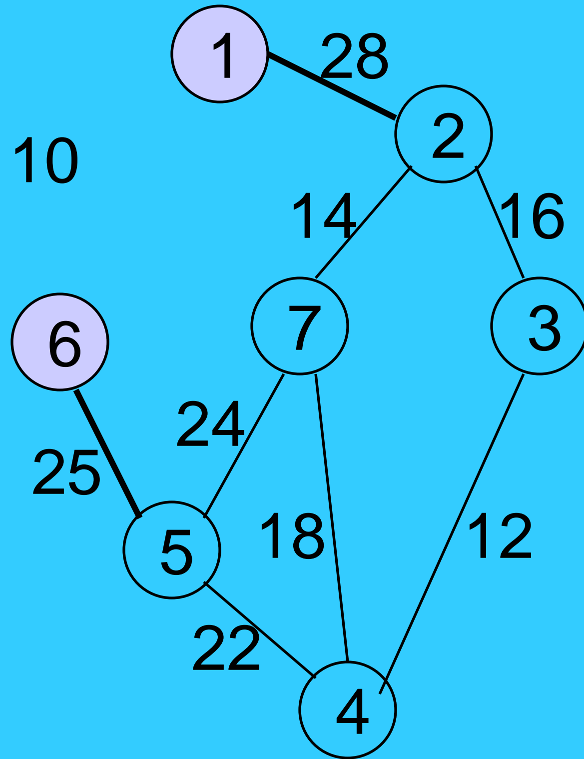
- Minimal path routing in all kinds of settings
  - circuits, networks, roads and sewers

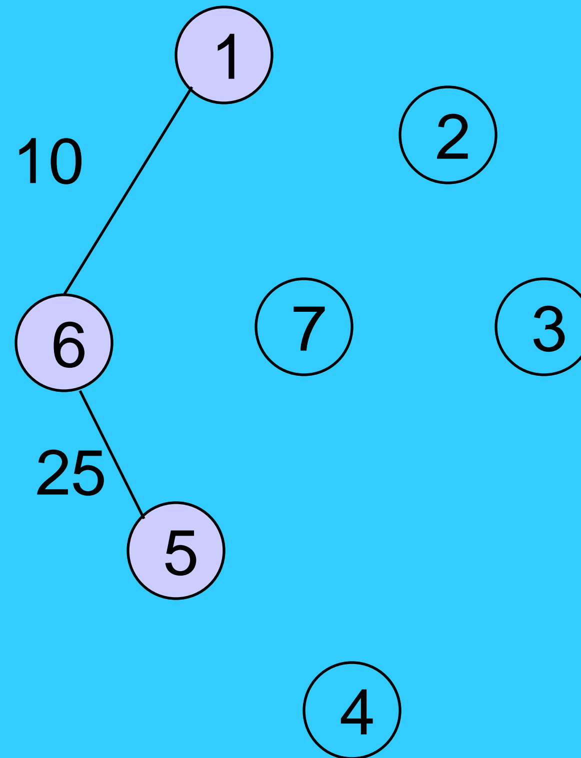
Prim's algorithm



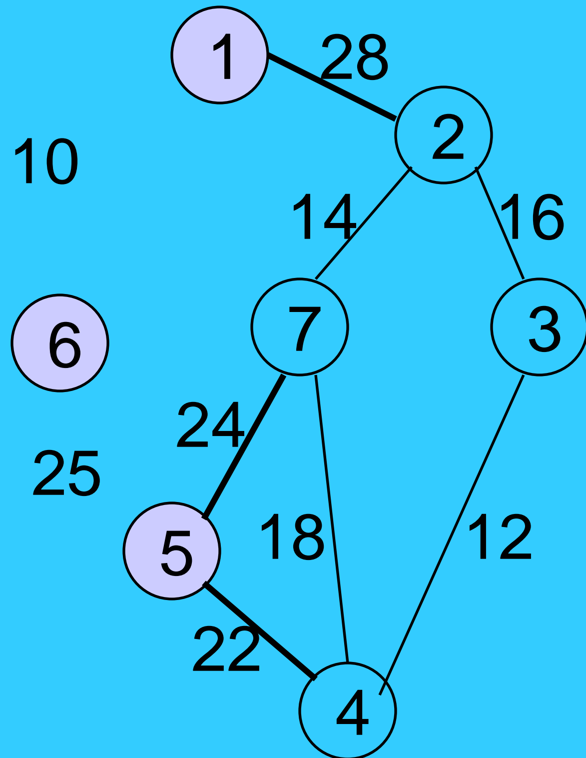


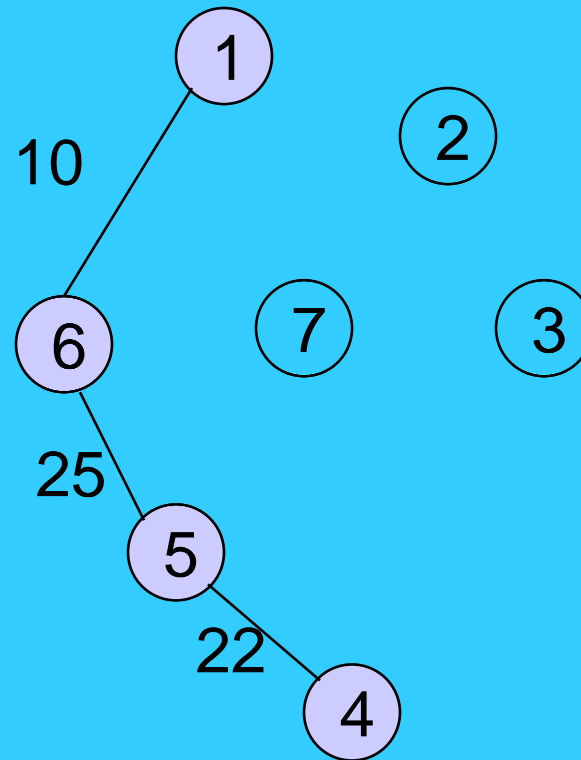
Prim's algorithm



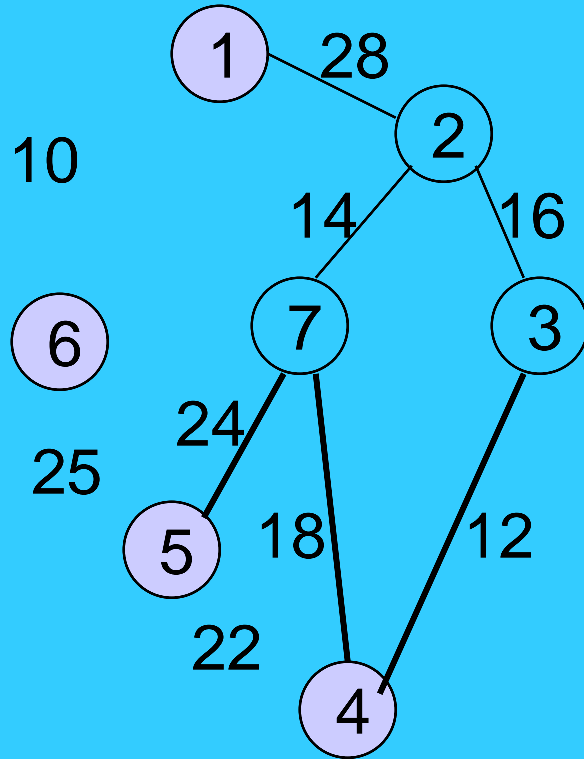


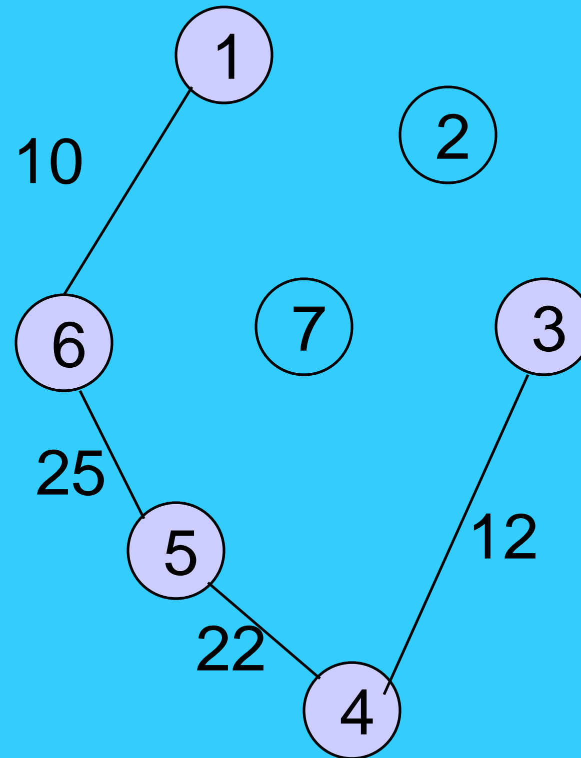
Prim's algorithm



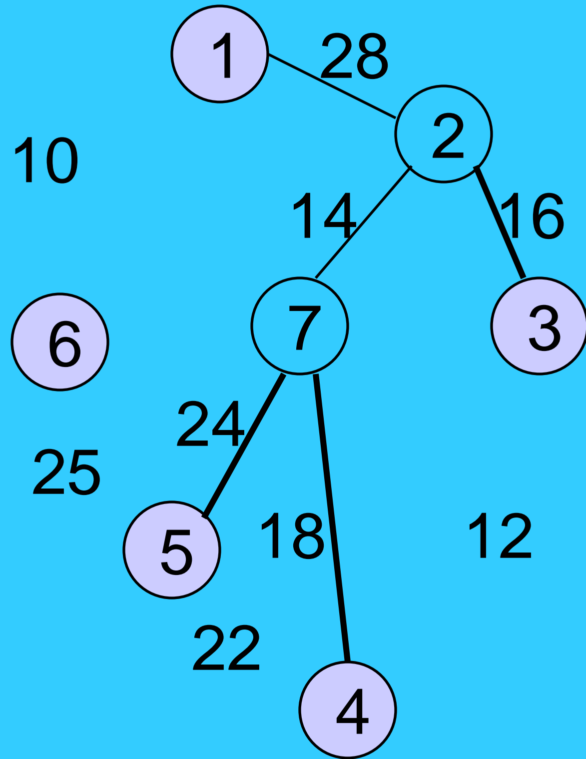


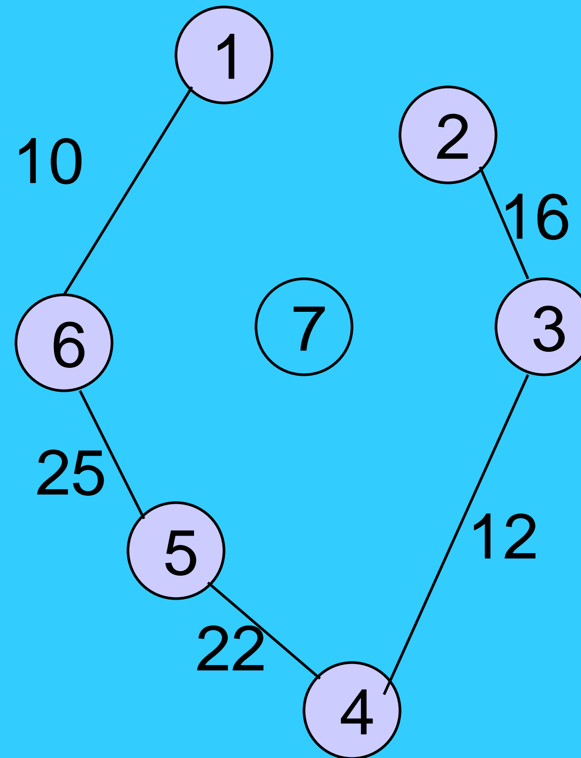
Prim's algorithm



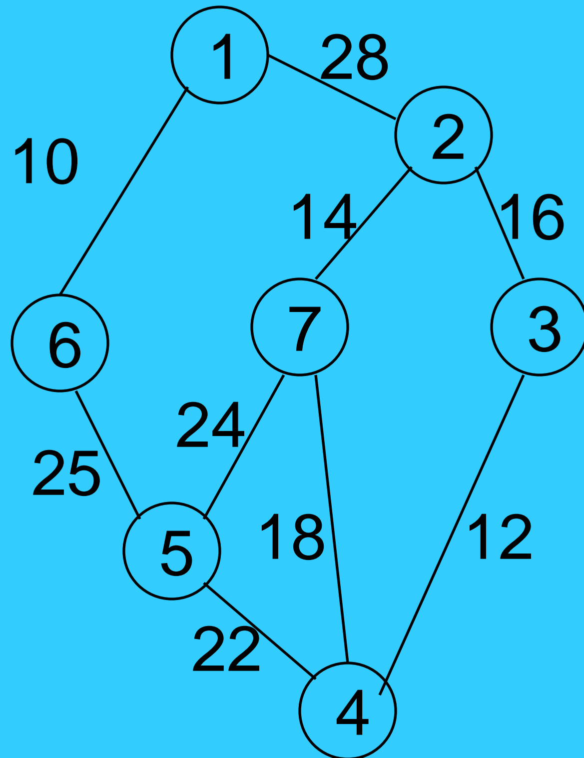


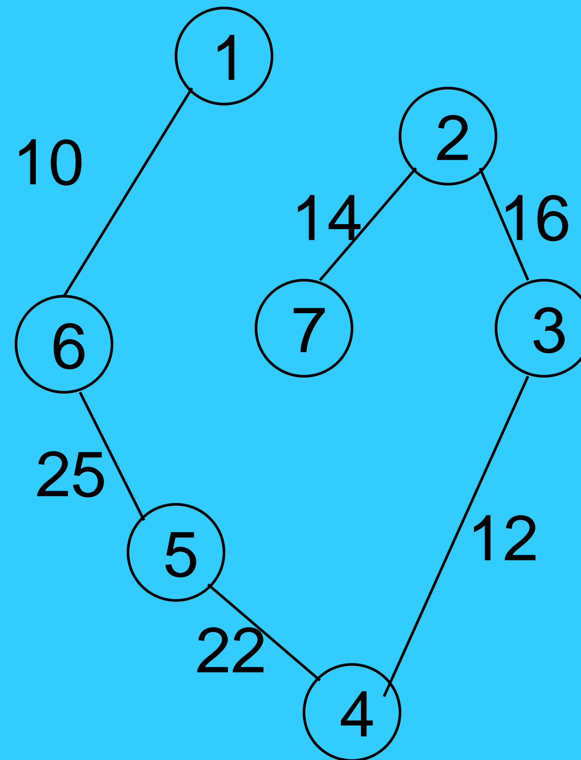
Prim's algorithm





Prim's algorithm



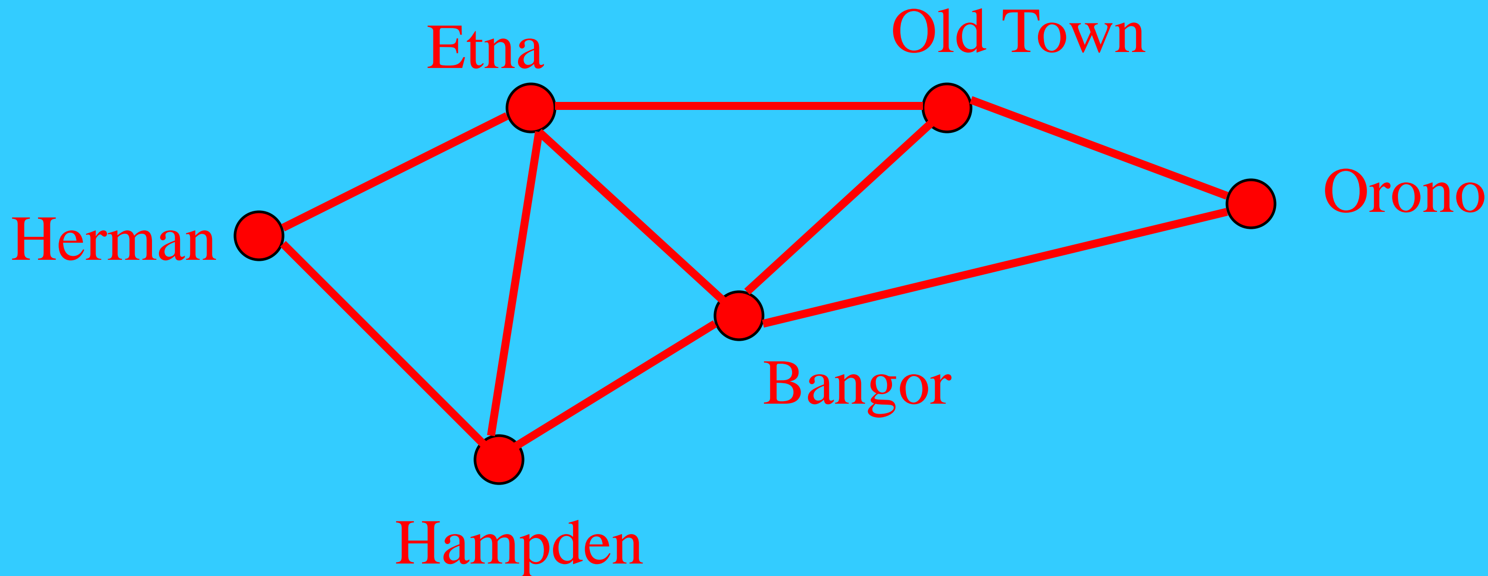


Cost = 99

Kruskal's algorithm

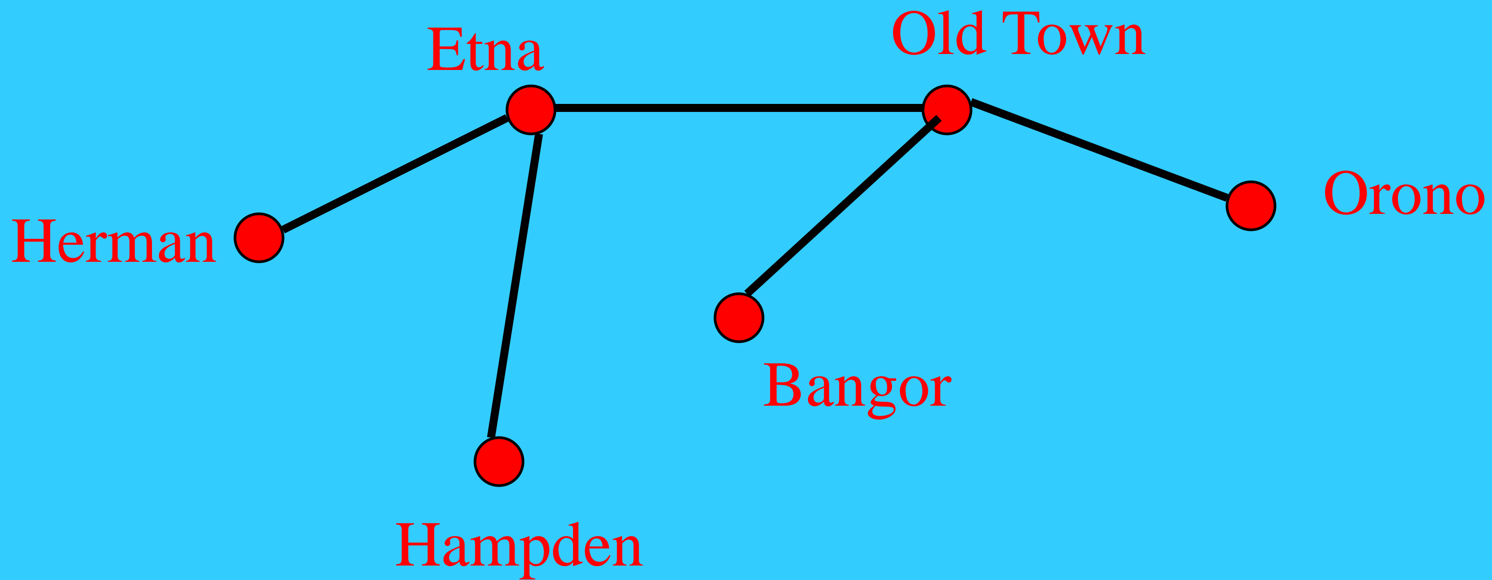
- Pick the cheapest edge that does not create a cycle in the tree
- Add the edge to the solution and remove it from the graph.
- Continue until all nodes are part of the tree.

Spanning Tree



How do we plow the fewest roads so there will always be cleared roads connecting any two towns?

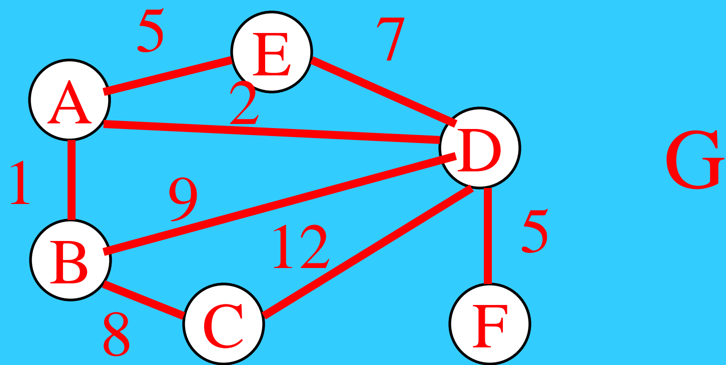
Spanning Tree

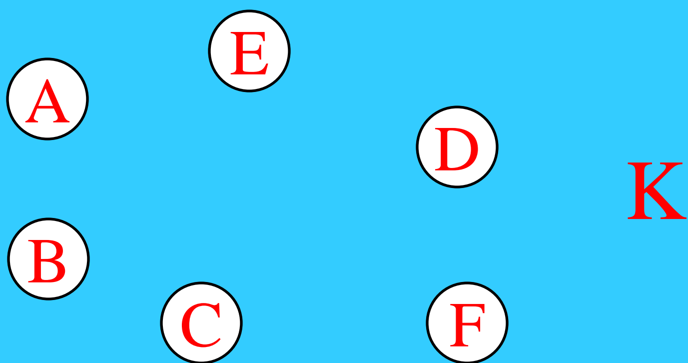


Kruskal's Algorithm

- Start by initializing a graph K with all of G's nodes and none of G's edges.
- For each edge (x,y) in G (taken in increasing order of weight)
  - If x and y are not in same connected component
    - Add edge (x,y) to K

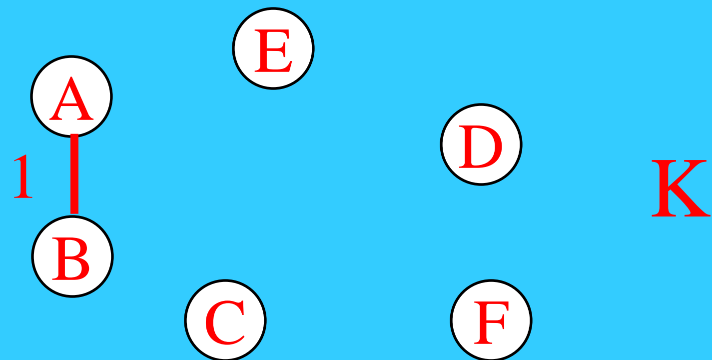
Kruskal's Algorithm





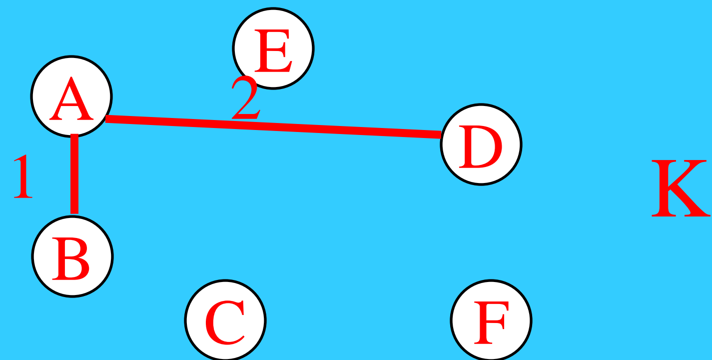
{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



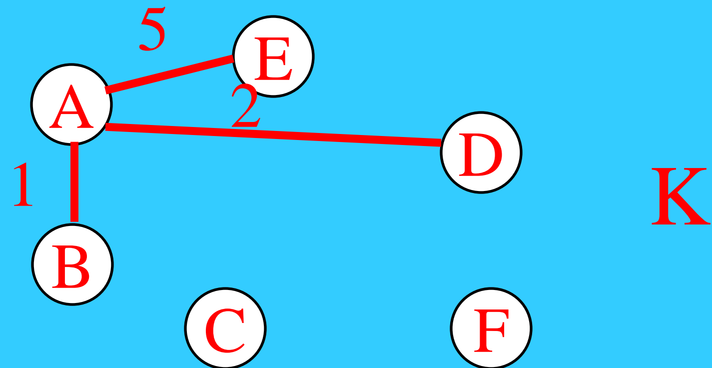
{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



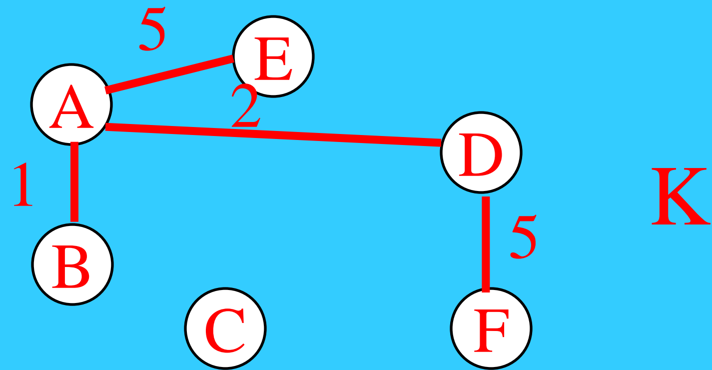
{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



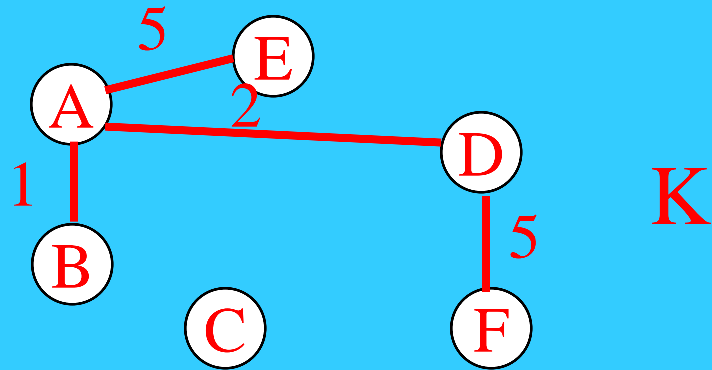
{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

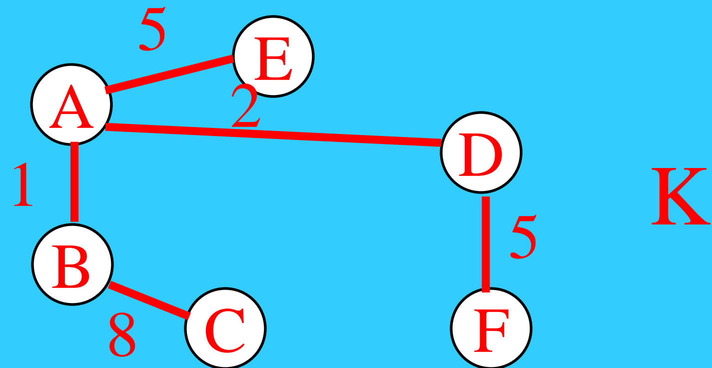
Kruskal's Algorithm



{E,D} already connected

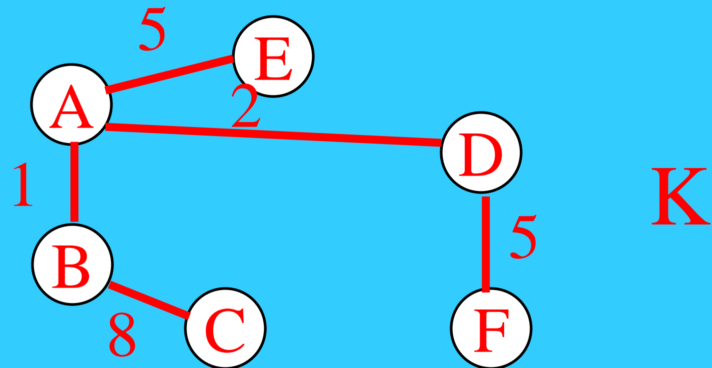
{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

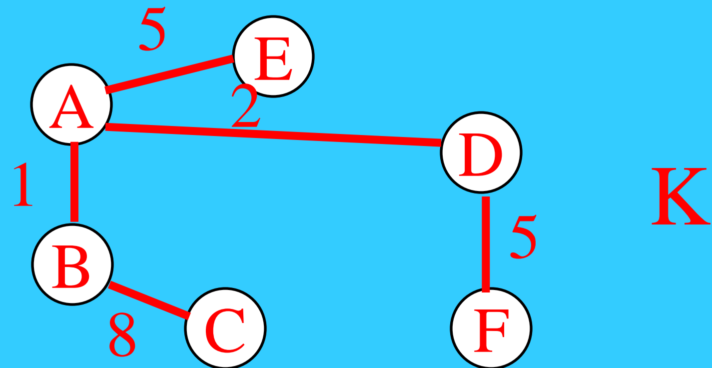
Kruskal's Algorithm



{B,D} already connected

{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm



{C,D} already connected

{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Compare Prim and Kruskal

- Which one is better?
- Which one would you use if…
  - you don't know the entire graph at the beginning?
  - you always want a tree in partial solutions?

Prim's Algorithm for MST

- Data structure:
  - S set of nodes added to the tree so far
  - S′ set of nodes not added to the tree yet
  - T the edges of the MST built so far
  - $\lambda(w)$ current length of the shortest edge (v, w) that connects w to the current tree
  - $\pi(w)$ potential parent node of w in the final MST (current parent that connects w to the current tree)

Prim's Algorithm

- Initialize S, S' and T
  - $S \leftarrow \{u\}$, $S' \leftarrow \mathbf{V} \setminus \{u\}$ // u is any vertex
  - $T \leftarrow \{\}$
  - $\forall v \in S'$, $\lambda(v) \leftarrow \infty$
- Initialize λ and π for the vertices adjacent to u
  - For each $v \in S'$ s.t. $(u,v) \in E$,
    - $\lambda(v) \leftarrow \omega\ \mathbf{((u,v))}$
    - $\pi(v) \leftarrow u$
- While ($S' \mathrel{!}= \phi$)
  - Find $u \in S'$, s.t. $\forall v \in S'$, $\lambda(u) \leq \lambda(v)$
  - $S \leftarrow S \cup \{u\}$, $S' \leftarrow S' \setminus \{u\}$, $T \leftarrow T \cup \{ (\pi(u), u) \}$
  - For each v s.t. $(u, v) \in E$,
    - If $\lambda(v) > \omega((u,v))$ then

      $\lambda(v) \leftarrow \omega((u,v))$

      $\pi(v) \leftarrow u$

The Greedy Algorithm (also known as Kruskal's Algorithm)

Let G be a connected weighted graph. To find a spanning tree T for G of minimal weight:

Step 1 Start with the forest T consisting of all vertices of G and no edges.

Step 2 Choose an edge e of G of minimal weight amongst all those not in T. Add e to T if this does not create a cycle in T.

Step 3 If T is connected then T is a minimal weight spanning tree so stop. Otherwise repeat from Step **2** .