Minimum Spanning Trees

CSE 373

Data Structures

Spanning Trees

- Given (connected) graph $G(V,E)$,
  a spanning tree $T(V',E')$:
  - Is a subgraph of G; that is, $V' \subseteq V$, $E' \subseteq E$.
  - Spans the graph ($V' = V$)
  - Forms a tree (no cycle);
  - So, E' has $|V| - 1$ edges

Minimum Spanning Trees

- Edges are weighted: find minimum cost spanning tree
- Applications
  - › Find cheapest way to wire your house
  - › Find minimum cost to send a message on the Internet

Strategy for Minimum Spanning Tree

- For any spanning tree T, inserting an edge e_{new} not in T creates a cycle
- But
  - › Removing any edge e_{old} from the cycle gives back a spanning tree
  - › If e_{new} has a lower cost than e_{old} we have progressed!

Strategy

- Strategy for construction:
  - Add an edge of minimum cost that does not create a cycle (greedy algorithm)
  - Repeat |V| -1 times
  - Correct since if we could replace an edge with one of lower cost, the algorithm would have picked it up

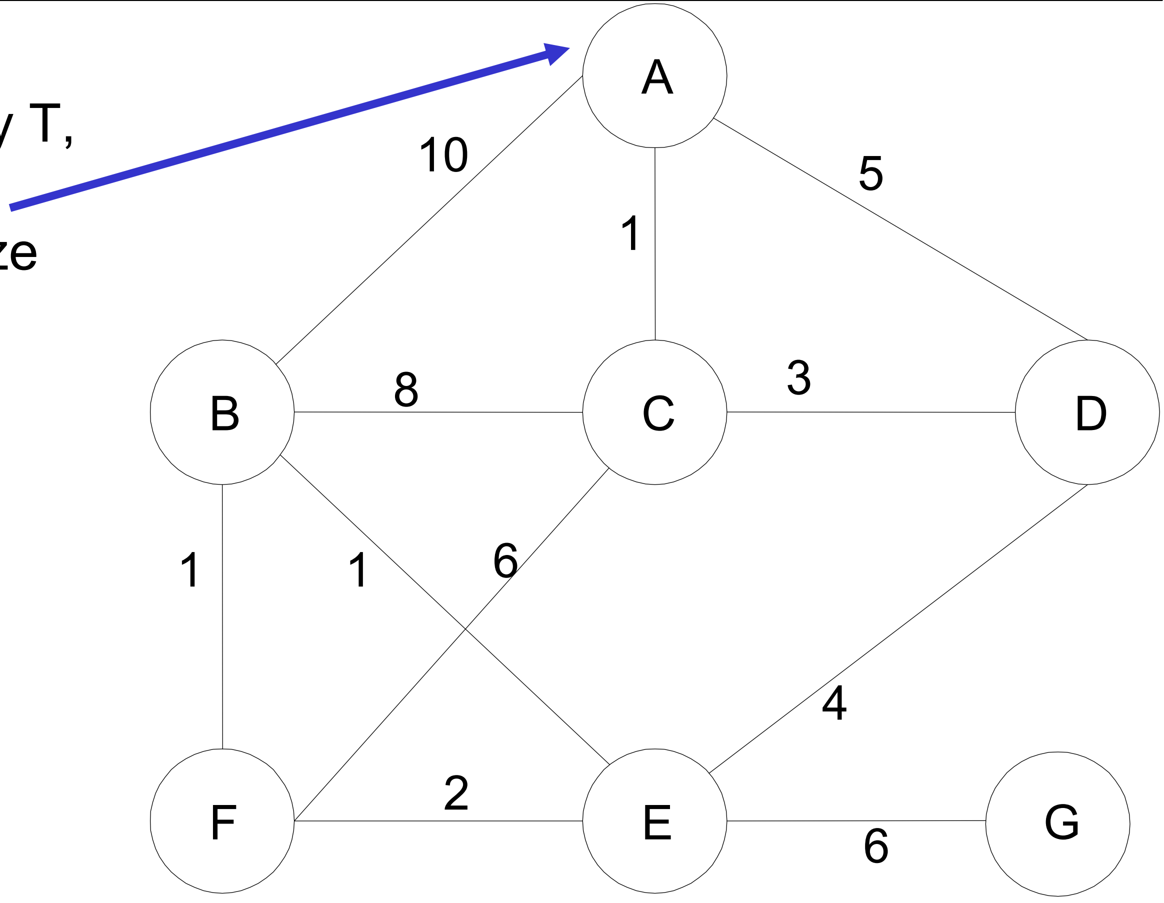
Two Algorithms

- Prim: (build tree incrementally)
  - Pick lower cost edge connected to known (incomplete) spanning tree that does not create a cycle and expand to include it in the tree
- Kruskal: (build forest that will finish as a tree)
  - Pick lowest cost edge not yet in a tree that does not create a cycle. Then expand the set of included edges to include it. (It will be somewhere in the forest.)

Prim's algorithm

Starting from empty T, choose a vertex at random and initialize

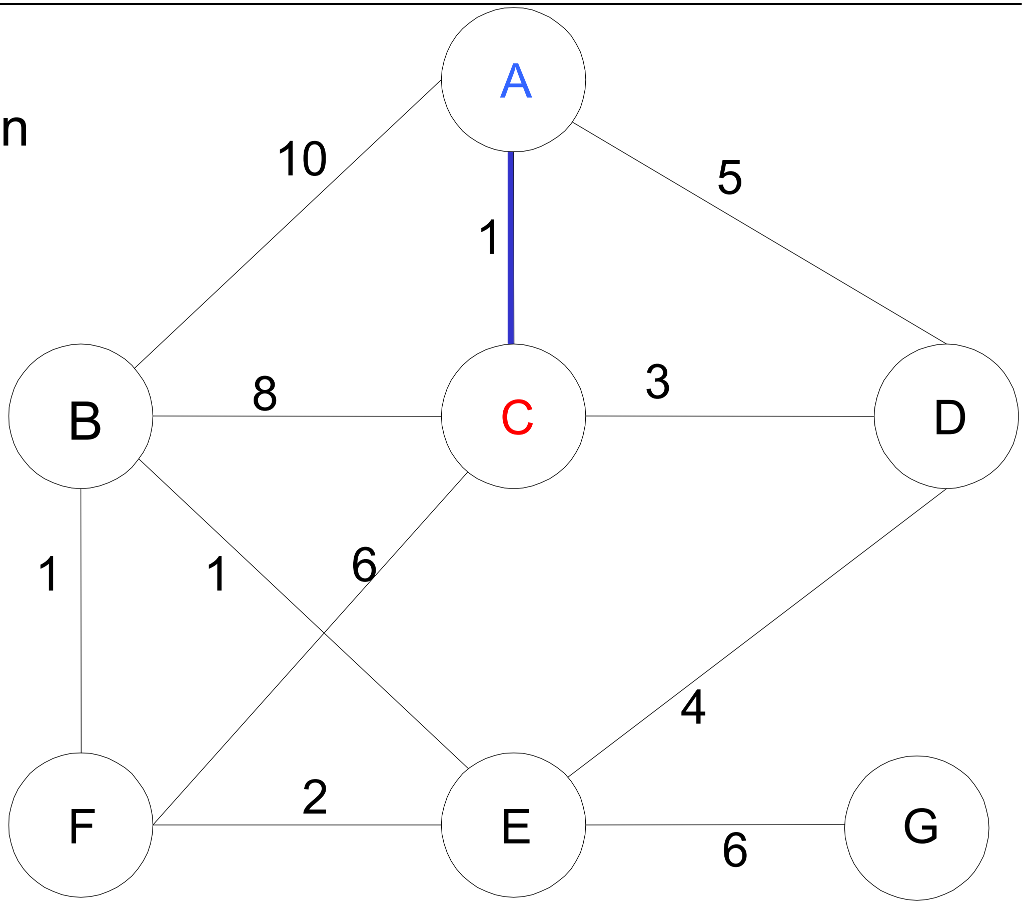
V = {A}, E' ={}



Prim's algorithm

Choose the vertex u not in V such that edge weight from u to a vertex in V is minimal (greedy!)

V={A,C} E'= {(A,C) }



Prim's algorithm

Repeat until all vertices have been chosen

Choose the vertex u not in V such that edge weight from v to a vertex in V is minimal (greedy!)

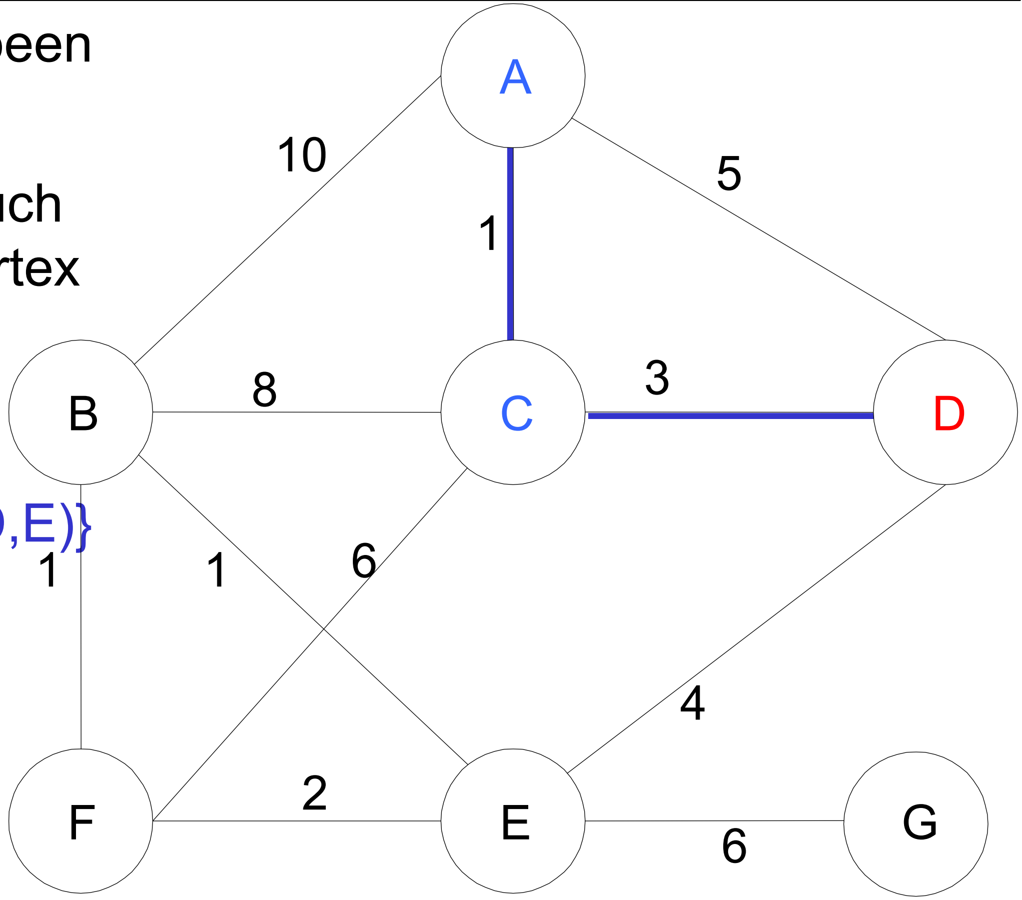
V= {A,C,D} E'= {(A,C),(C,D)}

V={A,C,D,E} E'={(A,C),(C,D),(D,E)}

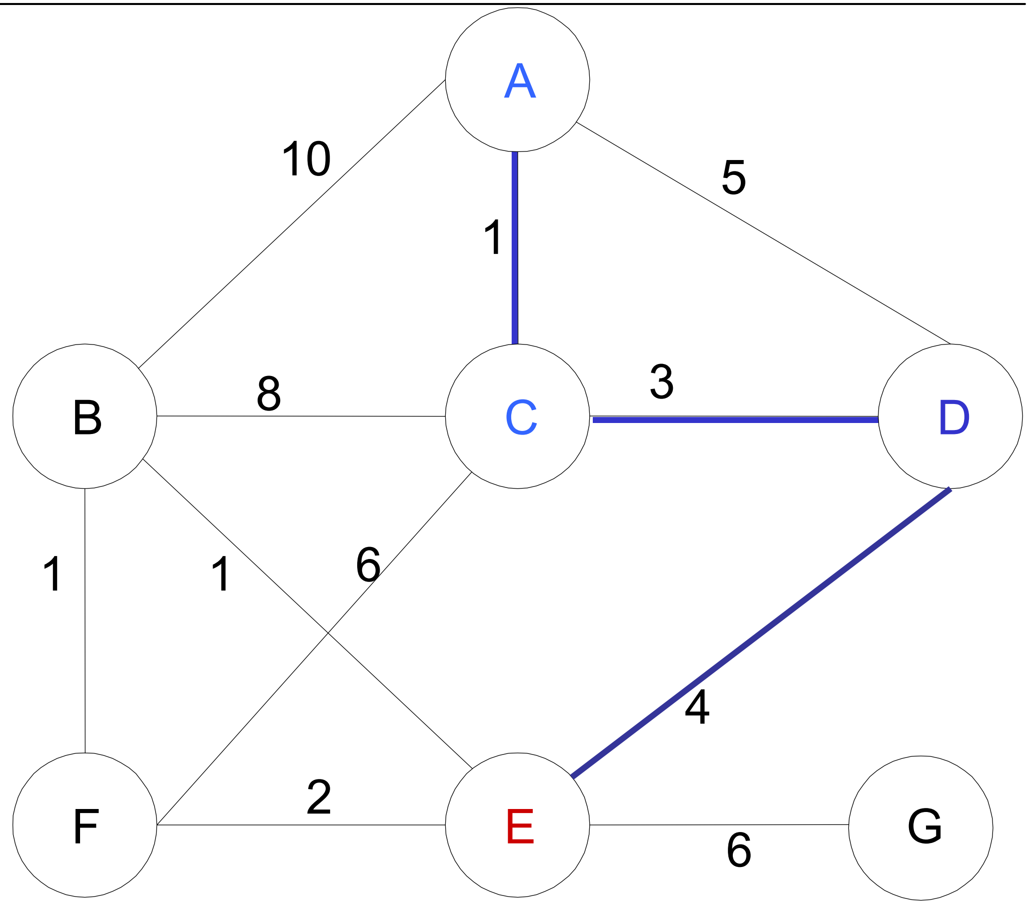
….

V={A,C,D,E,B,F,G}

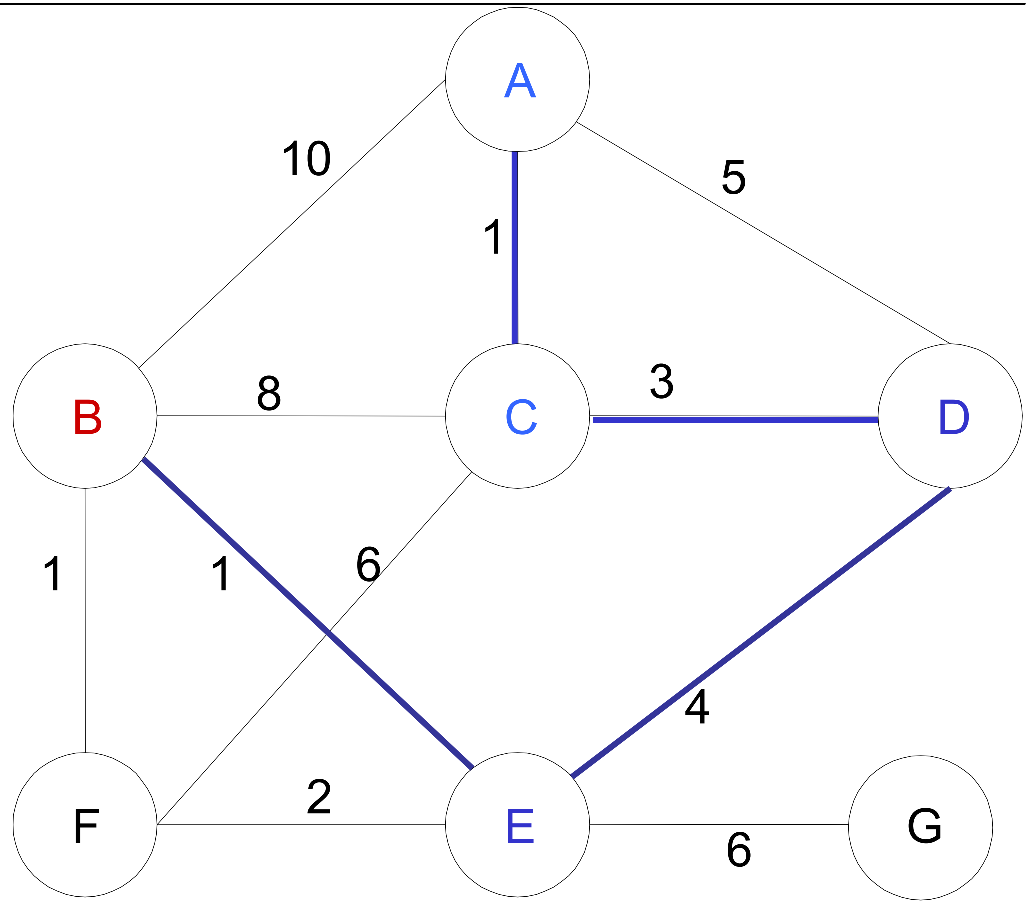
E'={(A,C),(C,D),(D,E),
(E,B),(B,F),(E,G)}



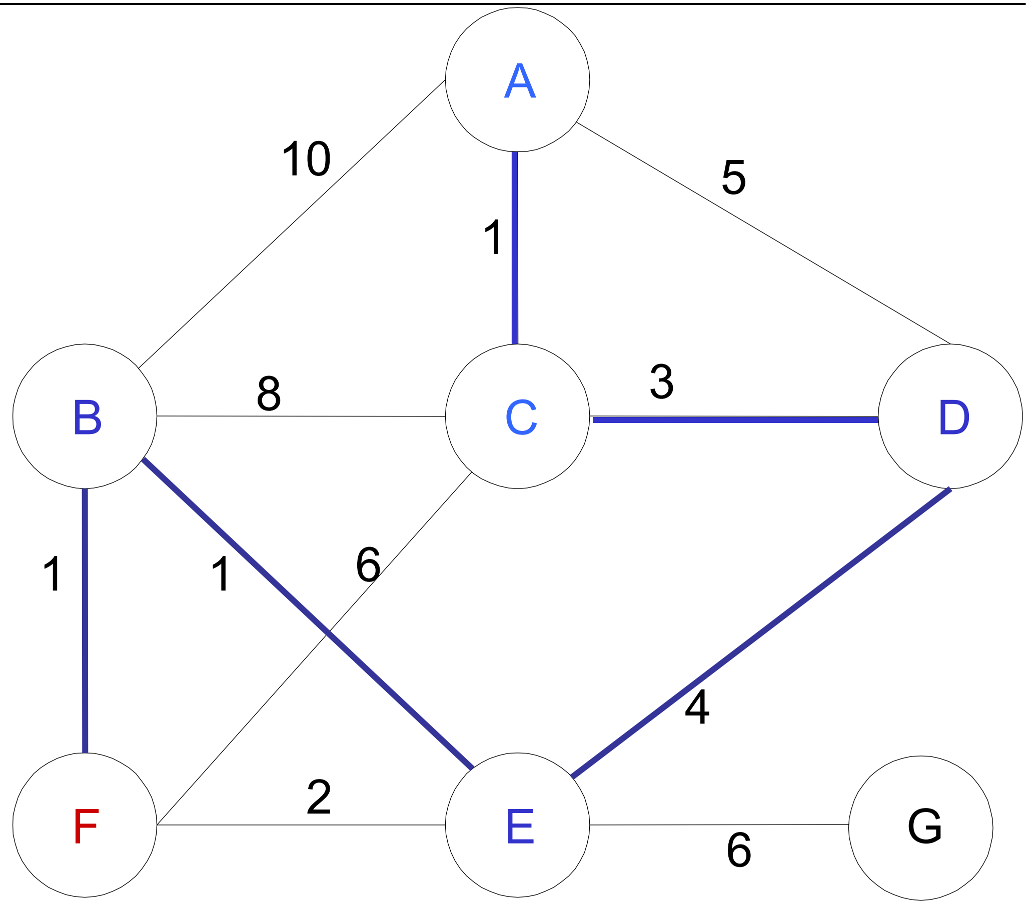
Prim's algorithm



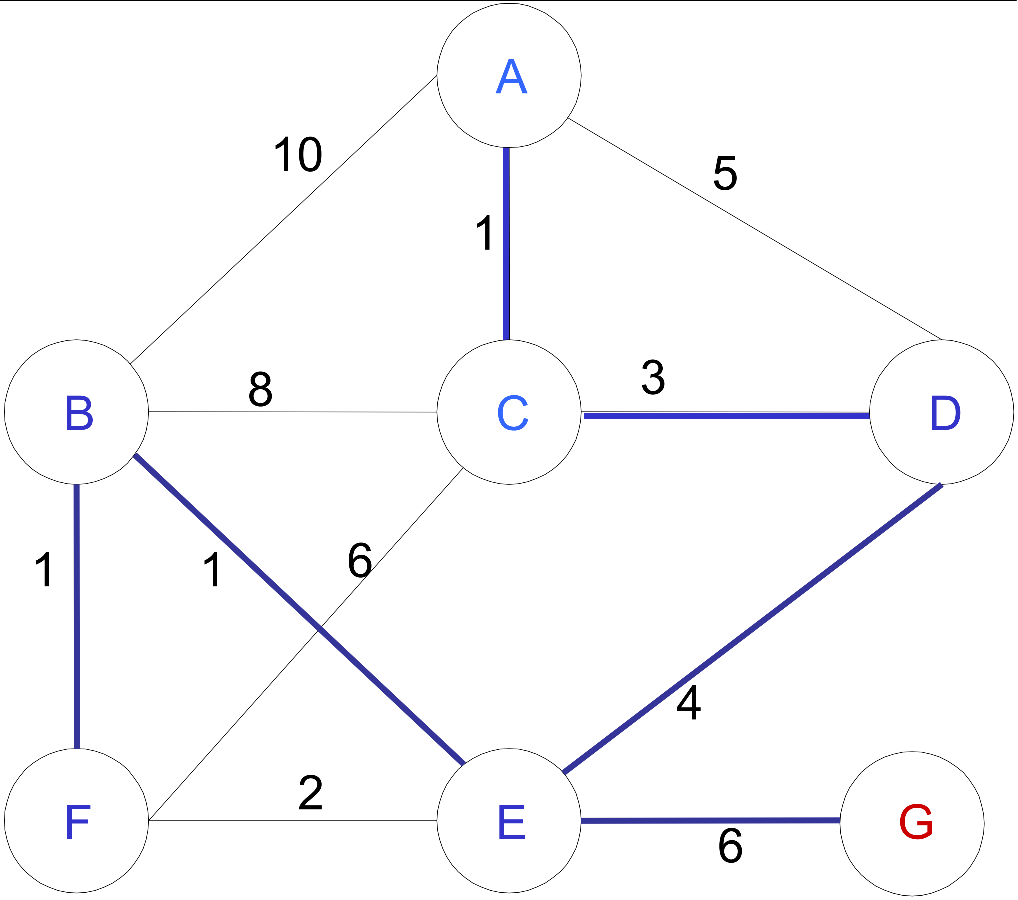
Prim's algorithm



Prim's algorithm



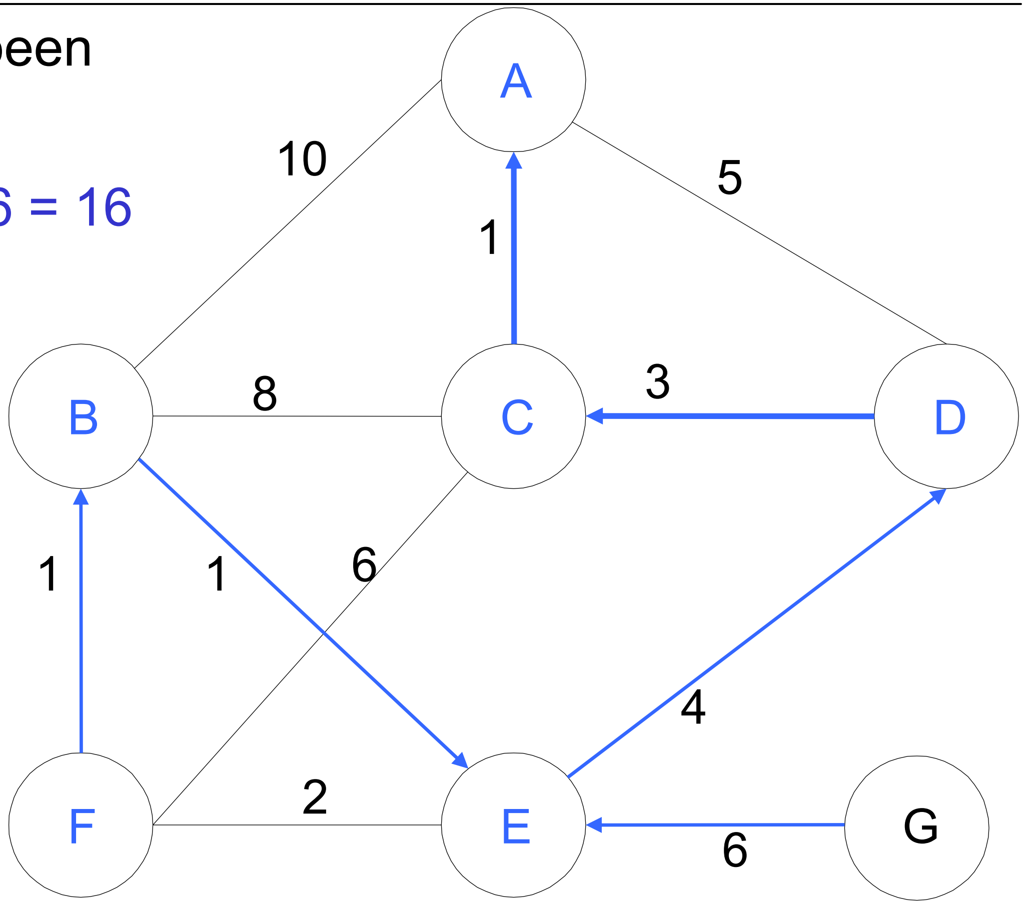
Prim's algorithm



Prim's algorithm

Repeat until all vertices have been chosen

Final Cost: 1 + 3 + 4 + 1 + 1 + 6 = 16



Prim’s Algorithm Implementation

- Assume adjacency list representation

Initialize connection cost of each node to “inf” and “unmark” them

Choose one node, say v and set cost[v] = 0 and prev[v] =0

While they are unmarked nodes

    Select the unmarked node **u** with minimum cost; mark it

    For each unmarked node **w** adjacent to **u**

        if cost(u,w) < cost(w) then cost(w) := cost (u,w)

        prev[w] = u

Prim’s algorithm Analysis

- If the “Select the unmarked node u with minimum cost” is done with binary heap then O((n+m)logn)

Kruskal's Algorithm

- Select edges in order of increasing cost
- Accept an edge to expand tree or forest only if it does not cause a cycle
- Implementation using adjacency list, priority queues and disjoint sets

Kruskal's Algorithm

Initialize a forest of trees, each tree being a single node

Build a priority queue of edges with priority being lowest cost

Repeat until |V| -1 edges have been accepted {

  Deletemin edge from priority queue

  If it forms a cycle then discard it

  else accept the edge – It will join 2 existing trees yielding a larger tree and reducing the forest by one tree

}

The accepted edges form the minimum spanning tree

Detecting Cycles

- If the edge to be added (u,v) is such that vertices u and v belong to the same tree, then by adding (u,v) you would form a cycle
  - › Therefore to check, Find(u) and Find(v). If they are the same discard (u,v)
  - › If they are different Union(Find(u),Find(v))

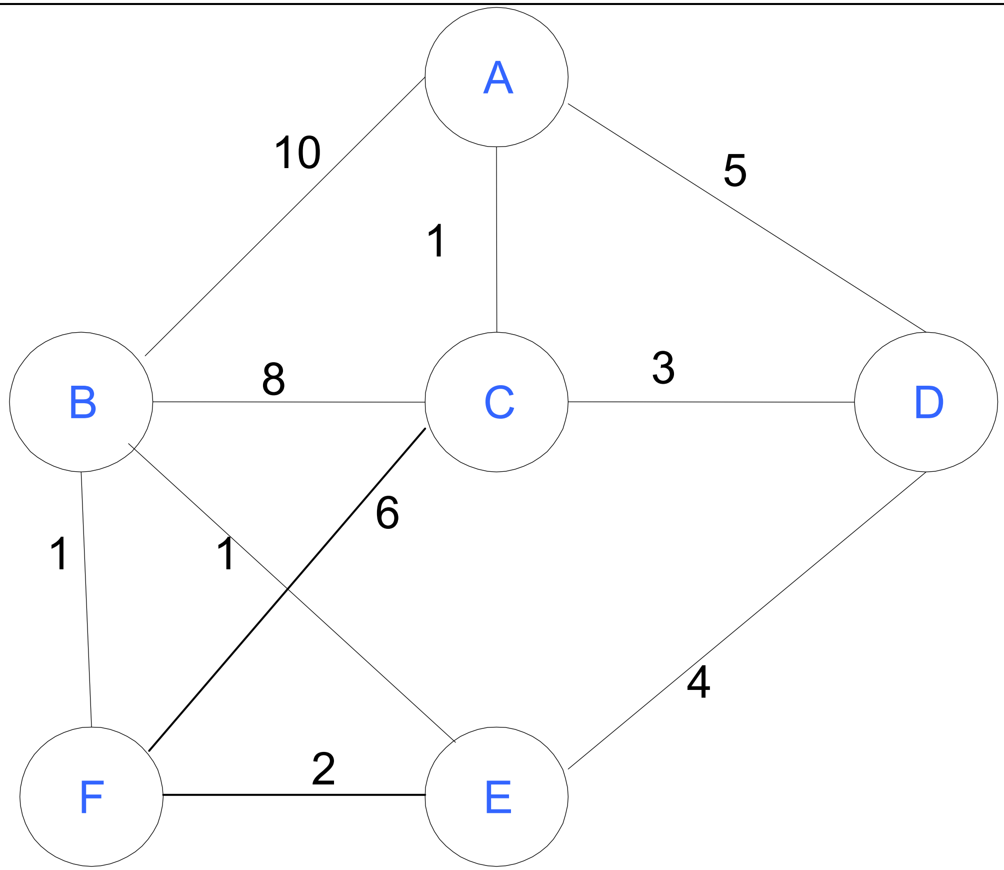
Properties of trees in K's algorithm

- Vertices in different trees are disjoint
  - True at initialization and Union won't modify the fact for remaining trees
- Trees form equivalent classes under the relation "is connected to"
  - u connected to u (reflexivity)
  - u connected to v implies v connected to u (symmetry)
  - u connected to v and v connected to w implies a path from u to w so u connected to w (transitivity)

K's Algorithm Data Structures

- Adjacency list for the graph
  - To perform the initialization of the data structures below
- Disjoint Set ADT's for the trees (recall Up tree implementation of Union-Find)
- Binary heap for edges

Example

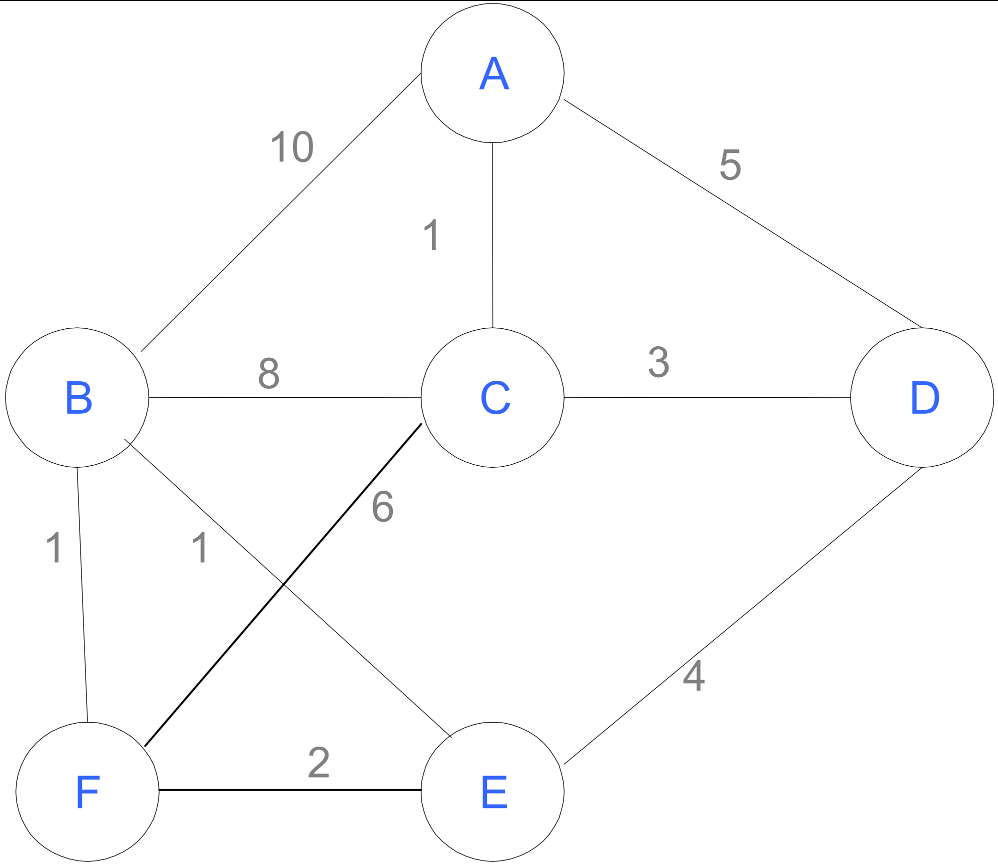


Initialization

Initially, Forest of 6 trees

F= {{A},{B},{C},{D},{E},{F}}

Edges in a heap (not shown)



Step 1

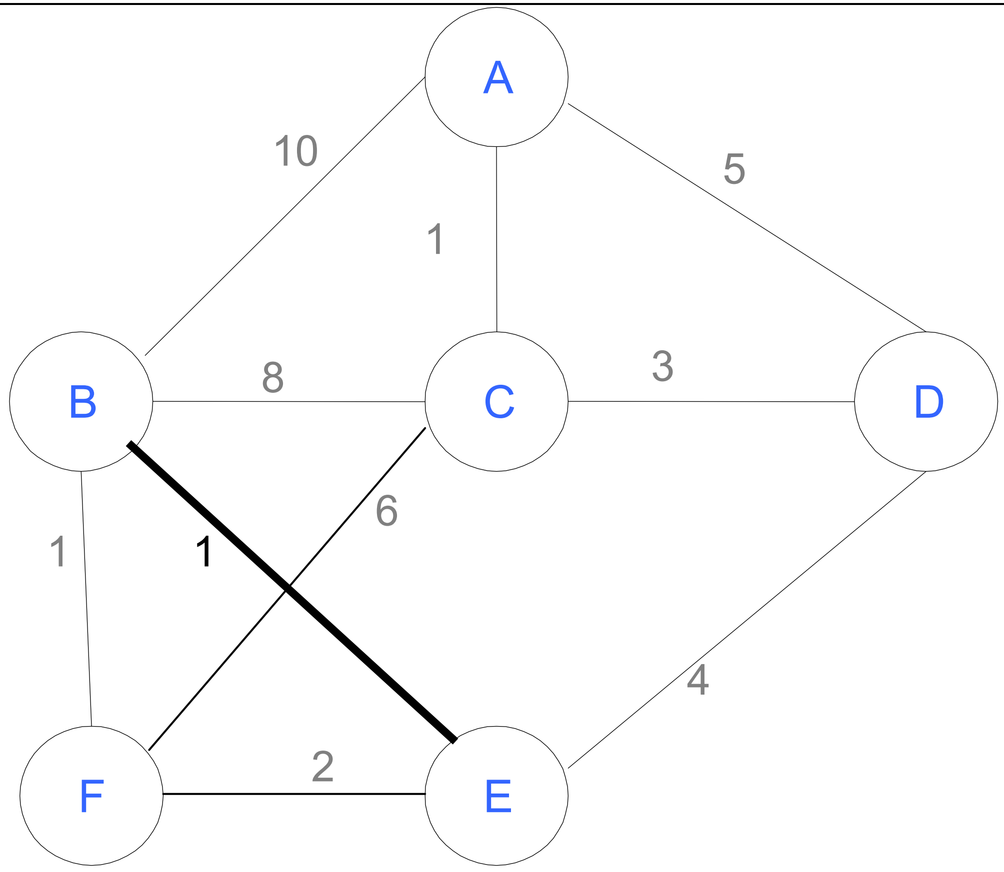
Select edge with lowest cost (B,E)

Find(B) = B, Find(E) = E

Union(B,E)

F= {{A},{B,E},{C},{D},{F}}

1 edge accepted



Step 2

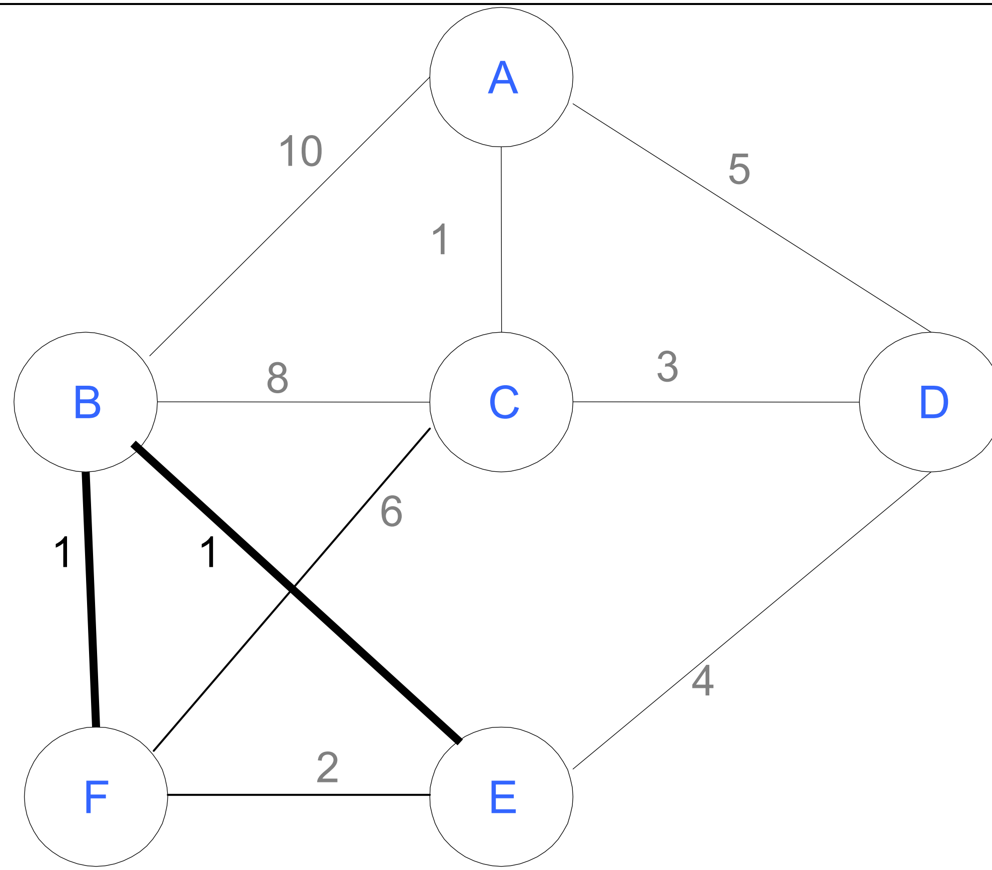
Select edge with lowest cost (B,F)

Find(B) = B, Find(F) = F

Union(B,F)

F= {{A},{B,E,F},{C},{D}}

2 edges accepted



Step 3

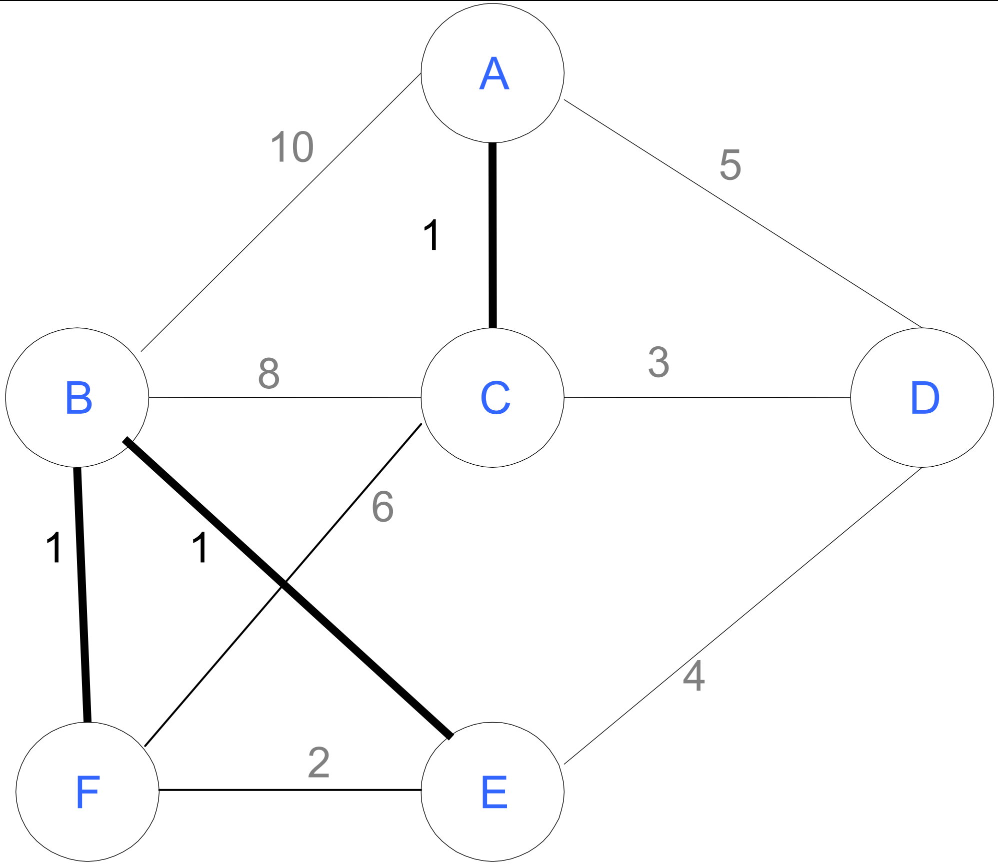
Select edge with lowest cost (A,C)

Find(A) = A, Find (C) = C

Union(A,C)

F= {{A,C},{B,E,F},{D}}

3 edges accepted



Step 4

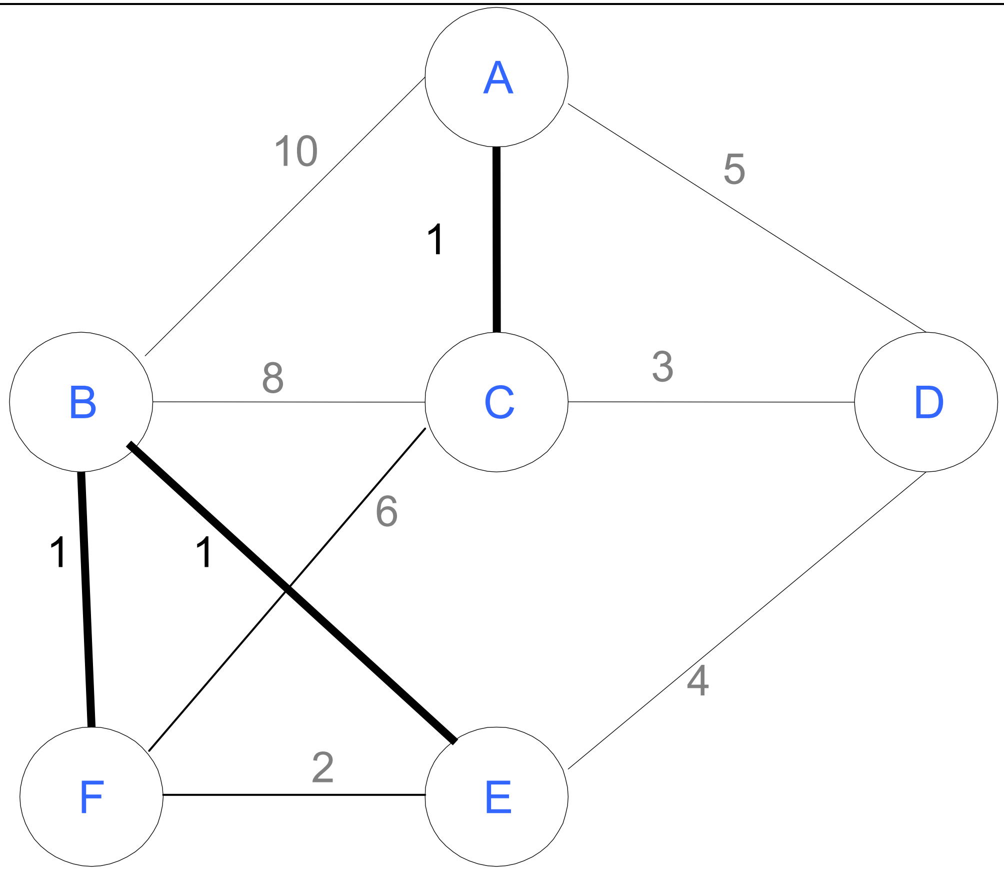
Select edge with lowest cost (E,F)

Find(E) = B, Find (F) = B

Do nothing

F= {{A,C},{B,E,F},{D}}

3 edges accepted



Step 5

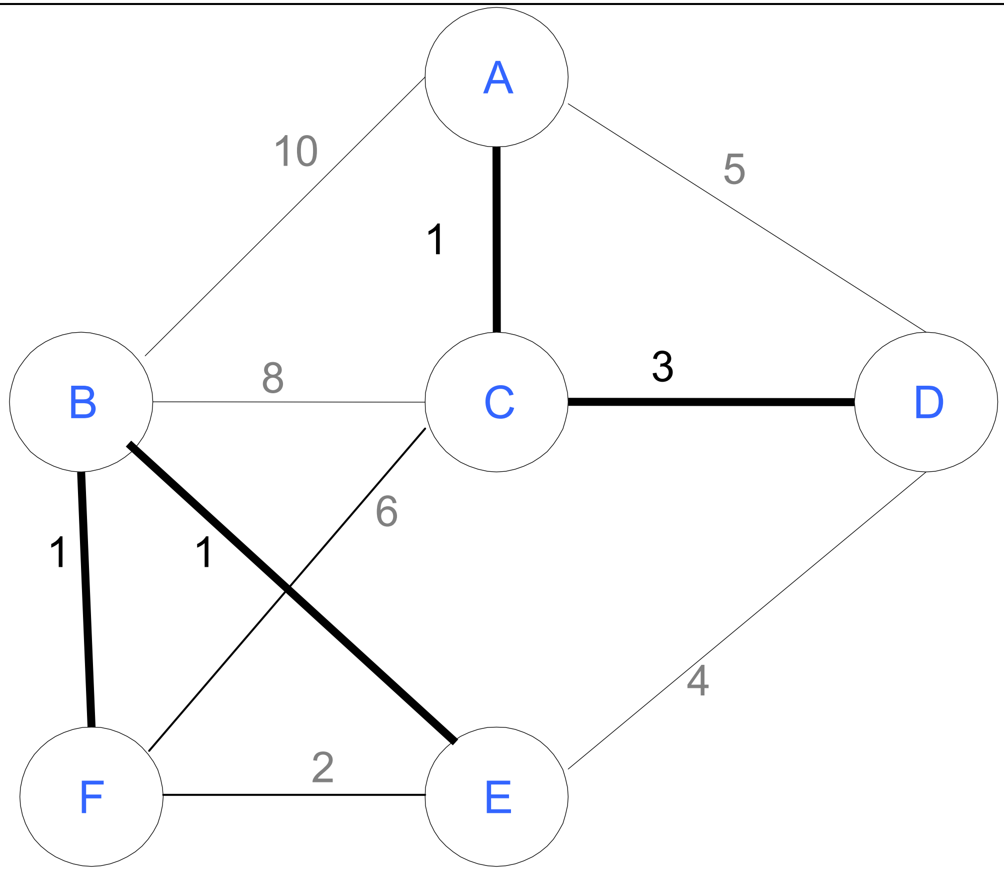
Select edge with lowest cost (C,D)

Find(C) = A, Find (D) = D

Union(A,D)

F= {{A,C,D},{B,E,F}}

4 edges accepted



Step 6

Select edge with lowest cost (D,E)

Find(D) = A, Find (E) = B

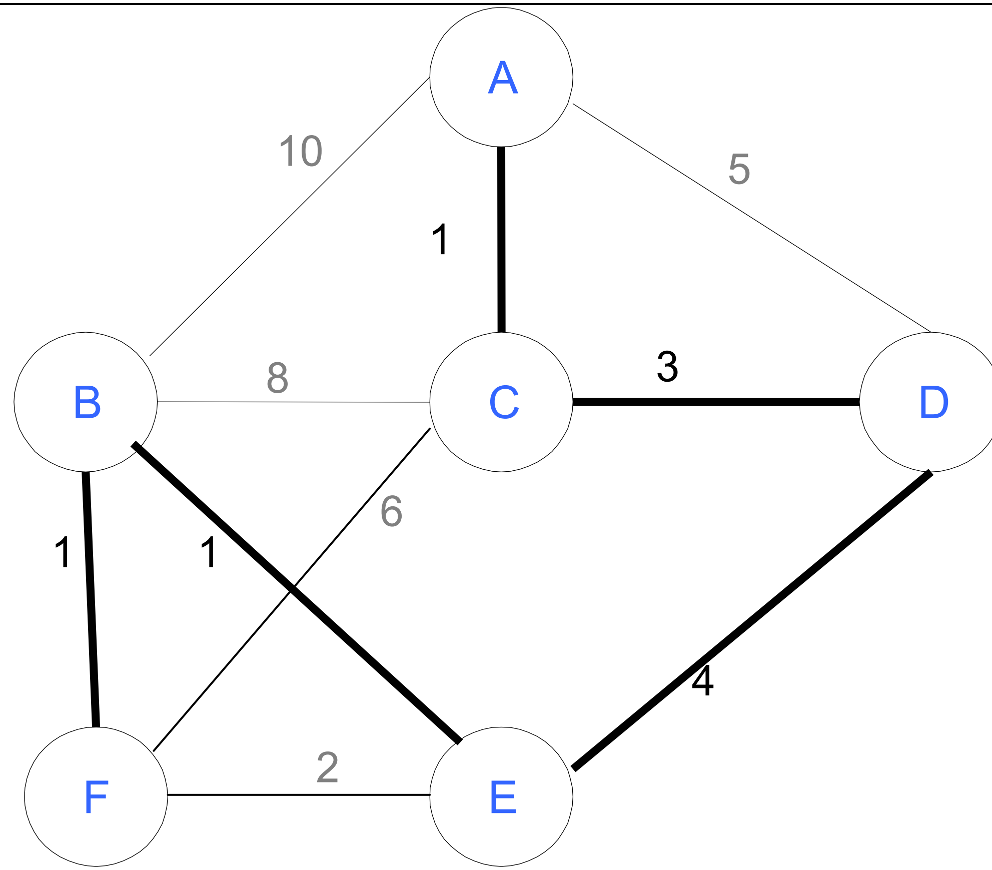
Union(A,B)

F= {{A,C,D,B,E,F}}

5 edges accepted : end

Total cost = 10

Although there is a unique spanning tree in this example, this is not generally the case



Kruskal's Algorithm Analysis

- Initialize forest O(n)
- Initialize heap O(m), m = |E|
- Loop performed m times
  - In the loop one deleteMin O(log m)
  - Two Find, each O(log n)
  - One Union (at most) O(1)
- So worst case O(m log m) = O(m log n)

Time Complexity Summary

- Recall that $m = |E| = O(V^2) = O(n^2)$
- Prim's runs in $O((n+m) \log n)$
- Kruskal runs in $O(m \log m) = O(m \log n)$
- In practice, Kruskal has a tendency to run faster since graphs might not be dense and not all edges need to be looked at in the deleteMin operations