Graph Algorithms

Fall 2020

School of Software Engineering

South China University of Technology

Minimum Spanning Tree

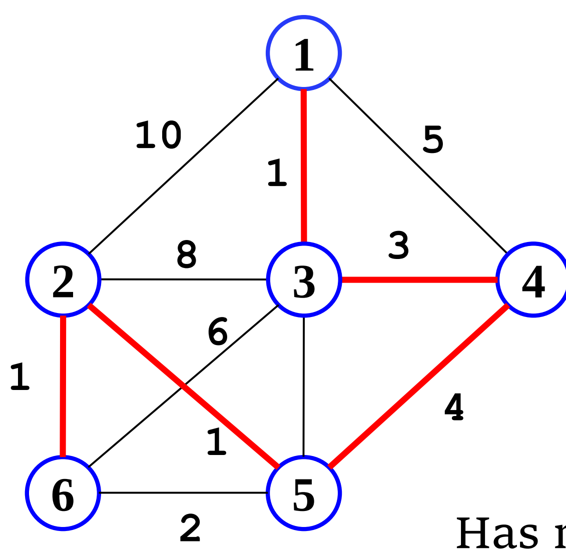
Section 9.5

Recall Spanning Tree

- Given (connected) graph G(V,E),
  a spanning tree T(V',E'):
  - Spans the graph (V' = V)
  - Forms a tree (no cycle);
  - E' has |V| -1 edges

Minimum Spanning Tree

- 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



Has minimum total cost

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

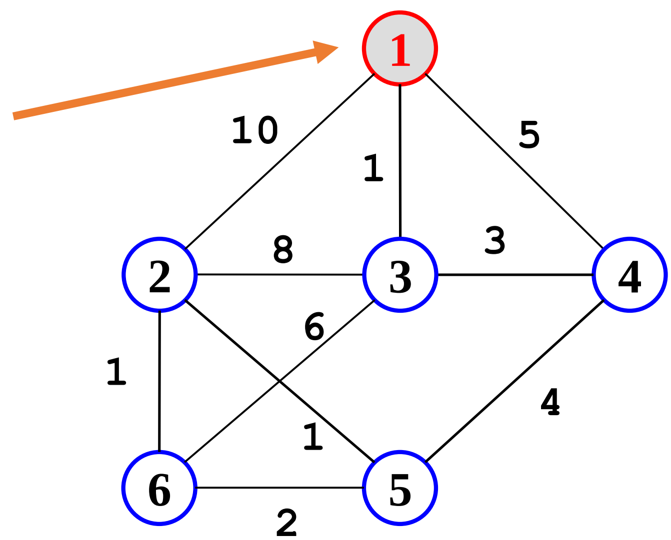
- Prim: (build tree incrementally)
  - Pick lower cost edge connected to known (incomplete) spanning tree that does not create a cycle and expand to include it in the tree

- Kruskal: (build forest that will finish as a tree)
  - Pick lower cost edge not yet in a tree that does not create a cycle and expand to include it somewhere in the forest

Prim’s algorithm

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

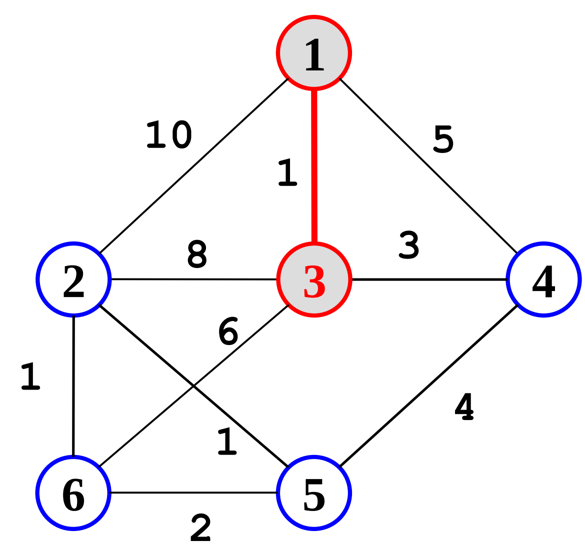
V = {1}
E’ ={}



Prim’s algorithm

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

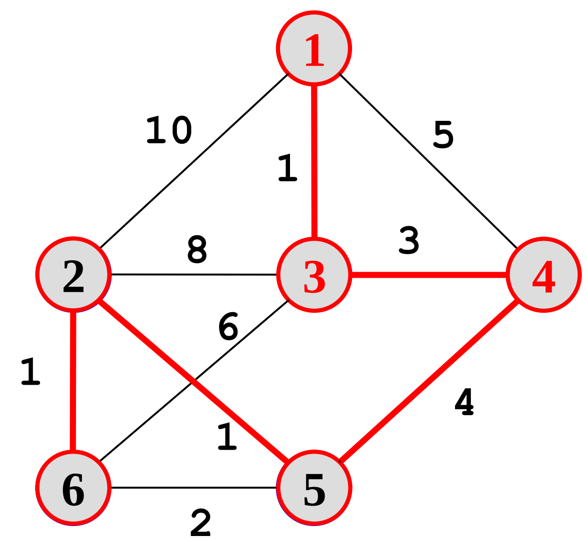
V={1,3}
E’= {(1,3) }



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= {1,3,4}
E’= {(1,3),(3,4)}

V={1,3,4,5}
E’={(1,3),(3,4),(4,5)}

……

V={1,3,4,5,2,6}
E’={(1,3),(3,4),(4,5),(5,2),(2,6)}

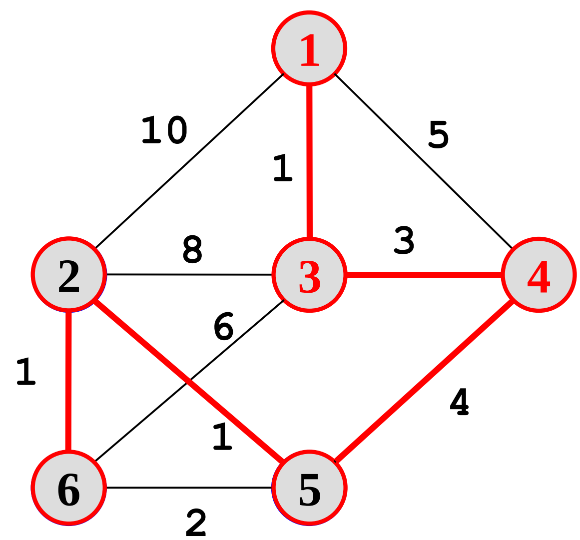
Prim’s algorithm

Repeat until all vertices have been chosen

V={1,3,4,5,2,6}

E’={(1,3),(3,4),(4, 5),(5,2),(2,6)}

Final Cost: 1 + 3 + 4 + 1 + 1 = 10



Prim's Algorithm Implementation

- Assume adjacency list representation

- (1) Initialize connection cost of each node to "infinite" and "unmark" them
- (2) Choose one node, say v and set cost[v] = 0 and prev[v] =0
- (3) While there 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**

- Looks a lot like Dijkstra's algorithm!

Prim’s Algorithm Implementation

```
//Implementation of Prim’s algorithm for MST
void Prim(Graph* G, int* D, int s) {
 int V[G->n()];  // store lowest cost vertex
 int i, w;
 for (int i=0; i<G->n(); i++) // Initialize
   D[i] = INFINITY; // cost
 D[0] = 0;

 for (i=0; i<G->n(); i++) {
   int v = minVertex(G, D);
   G->setMark(v, VISITED);

   if (v != s)
    AddEdgetoMST(V[v], v); //add edge to MST
   if (D[v] == INFINITY) return; //unreachable vertices

   for (w=G->first(v); w<G->n(); w=G->next(v,w))
    if (D[w] > G->weight(v,w)) {
      D[w] = G->weight(v,w); //Update cost
      V[w] = v; // where it came from
    }
   }
}
```

Prim's algorithm Analysis

- Like Dijkstra's algorithm
- If the "**Select the unmarked node u with minimum cost**" is done with binary heap then $O((n+m)\log n)$

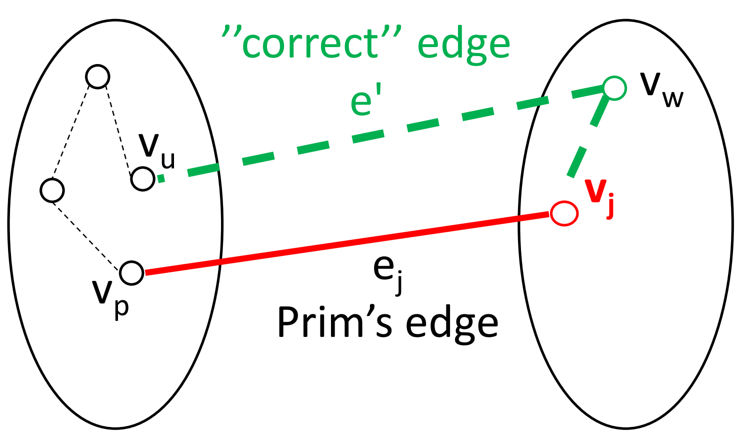
Prim’s Algorithm

- Prim’s algorithm is a greedy algorithm
  - At each step in the `for` loop, we always select the least-cost edge that connects some marked vertex to some unmarked vertex.

- Proof of Correctness
  - Prim’s algorithm produces a minimum-cost spanning tree.
  - Proof by contradiction

Proof of Prim's Algorithm

- Define an ordering on the vertices $v_0, v_1, \ldots, v_{n-1}$
  - according to the order in which they were added to the MST by the algorithm
- Suppose edge e_j is the first edge (the lowest numbered) where the algorithm "went wrong" – differ from the "true" MST
  - $e_j = (v_p, v_j)$ where $p<j$
- There must exist a path $v_j \text{->} v_w \text{->} v_u \text{->} v_p$ ($u<j$ and $w>j$) in the true MST that connect v_p and v_j
- Edge e' must be of lower cost than edge e_j
- Contradict the "Prim's algorithm would have selected the least-cost edge available"



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)

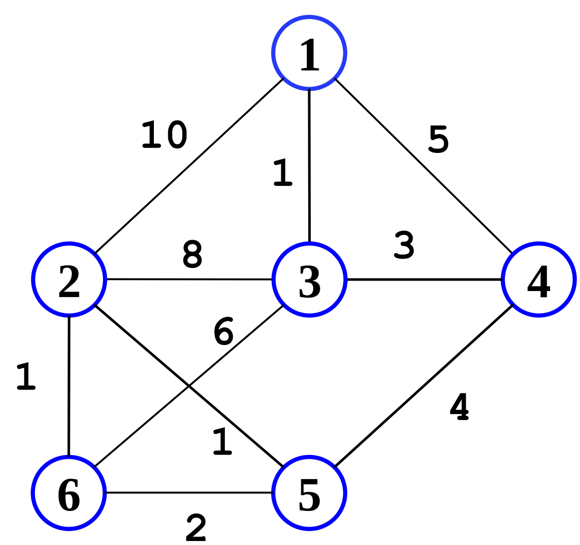
Kruskal's Algorithm

```
vector<Edge> kruskal( vector<Edge> edges, int numVertices )
{
   DisjSets ds{ numVertices };
   priority_queue pq{ edges };
   vector<Edge> mst;

   while( mst.size( ) != numVertices - 1 ){
      Edge e = pq.pop( ); // Edge e = (u, v)
      SetType uset = ds.find( e.getu( ) );
      SetType vset = ds.find( e.getv( ) );

      if( uset != vset ){
         // Accept the edge
         mst.push_back( e );
         ds.union( uset, vset );
      }
   }
   return mst;
}
```

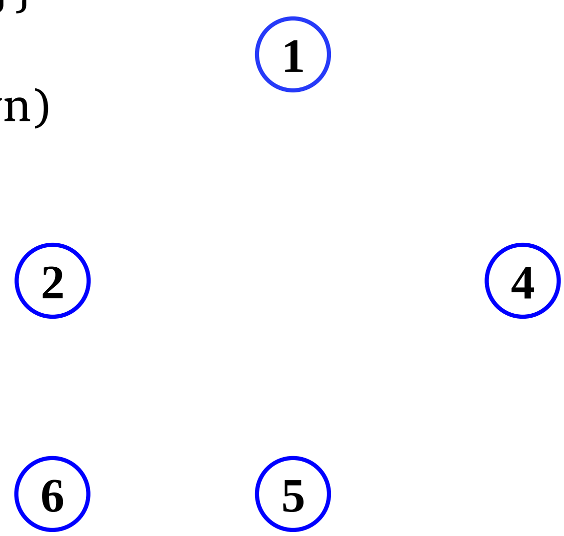
Example

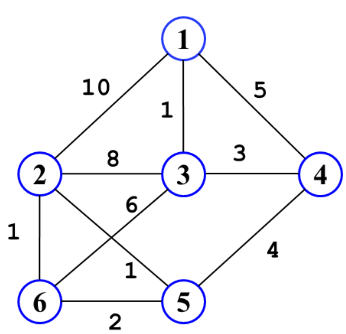


Initialization

Initially, Forest of 6 trees
F= {{1},{2},{3},{4},{5},{6}}

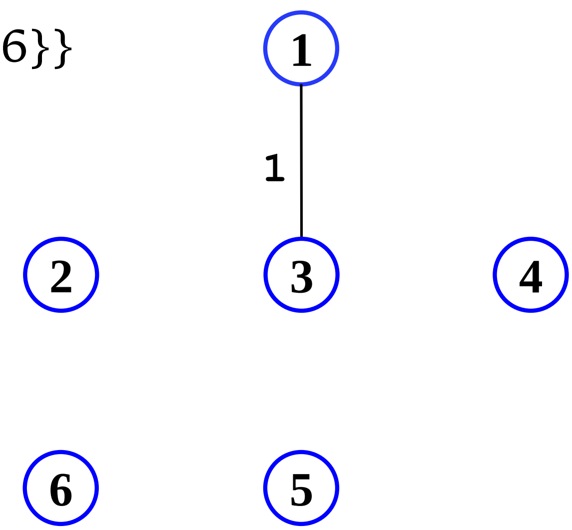
Edges in a heap (not shown)

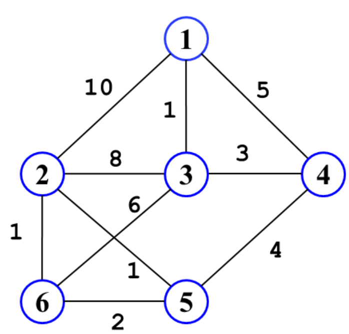




Step 1

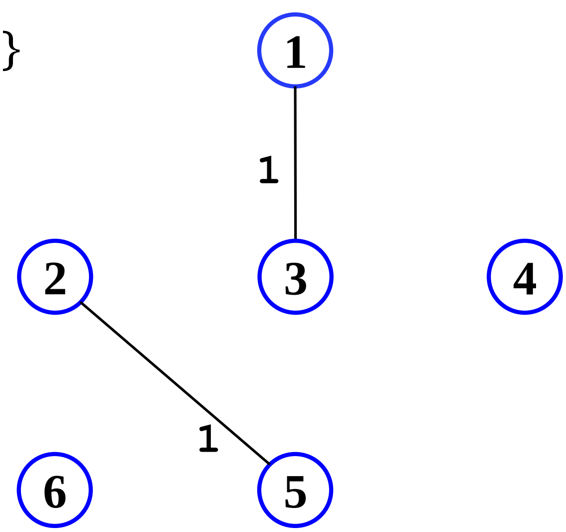
Select edge with lowest cost (1,3)
Find(1) = 1, Find (3) = 3
Union(1,3)
F= {{1,3},{2}, {4},{5},{6}}
1 edge accepted

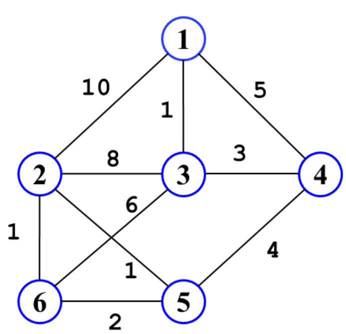




Step 2

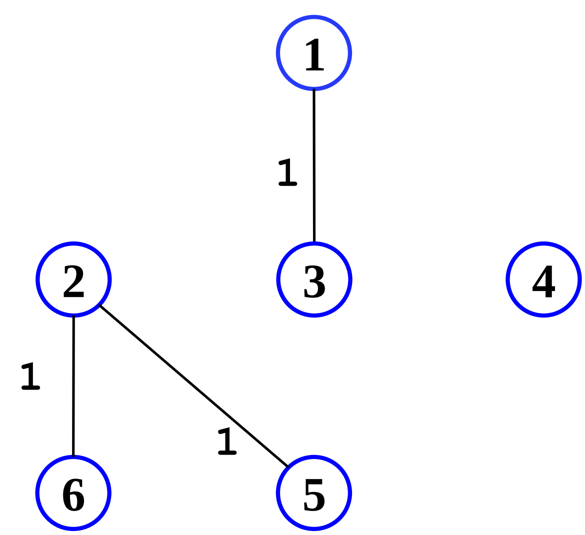
Select edge with lowest cost (2,5)
Find(2) = 2, Find (5) = 5
Union(2,5)
F= {{1,3},{2,5}, {4},{6}}
2 edge accepted

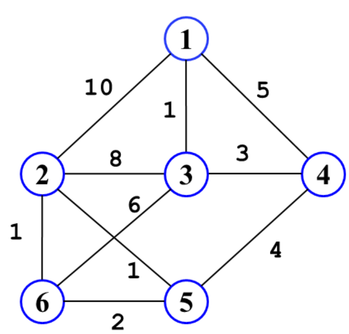




Step 3

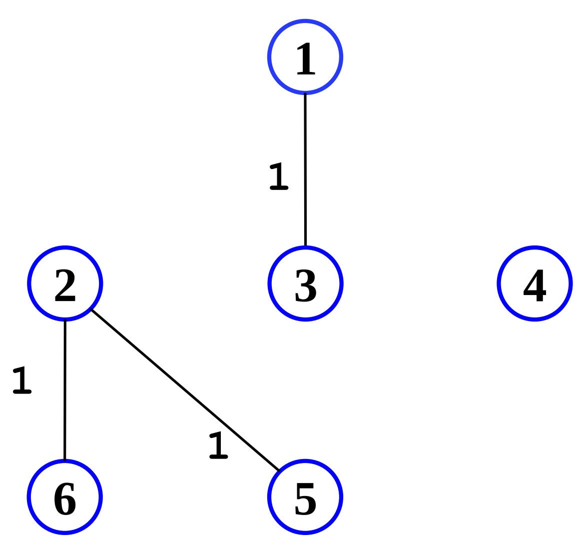
Select edge with lowest cost (2,6)
Find(2) = 2, Find (6) = 6
Union(2,6)
F= {{1,3},{2,5,6}, {4}}
3 edge accepted

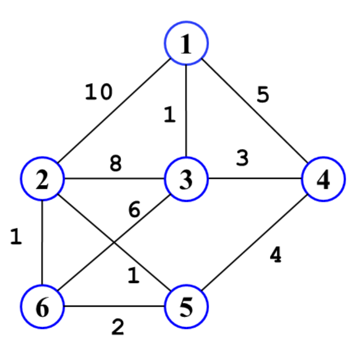




Step 4

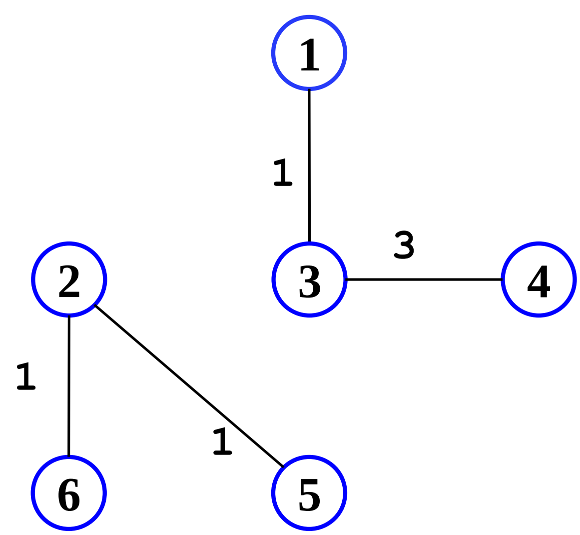
Select edge with lowest cost (5,6)
Find(5) = 2, Find (6) = 2
Do nothing
F= {{1,3},{2,5,6}, {4}}
3 edge accepted

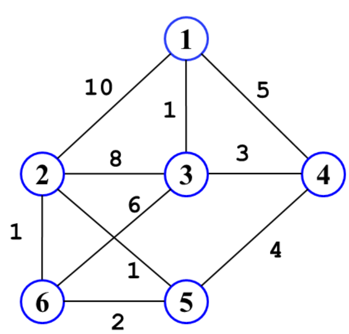




Step 5

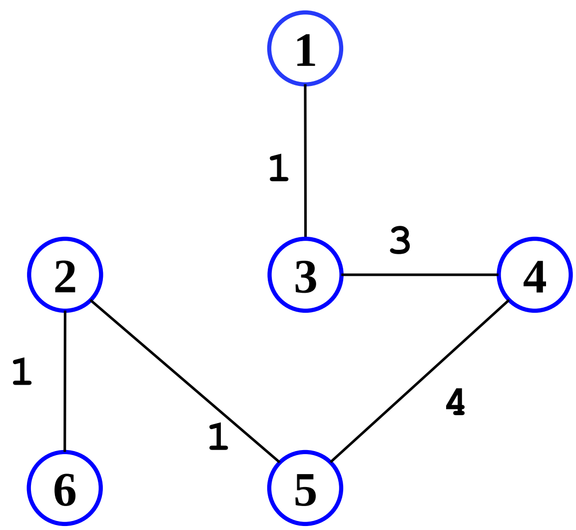
Select edge with lowest cost (3,4)
Find(3) = 1, Find (4) = 4
Union(1,4)
F= {{1,3,4},{2,5,6}}
4 edge accepted

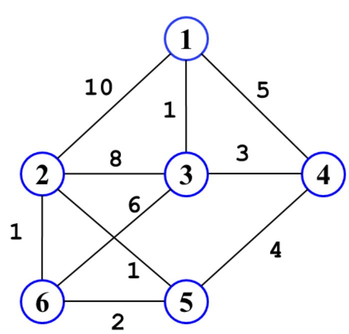




Step 6

Select edge with lowest cost (4,5)
Find(4) = 1, Find (5) = 2
Union(1,2)
F= {{1,3,4,2,5,6}}
5 edge 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's 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

Homework 7-3

- Textbook Exercises 9.15