



CSE 332: Data Structures & Parallelism

Lecture 17: Minimum Spanning Trees

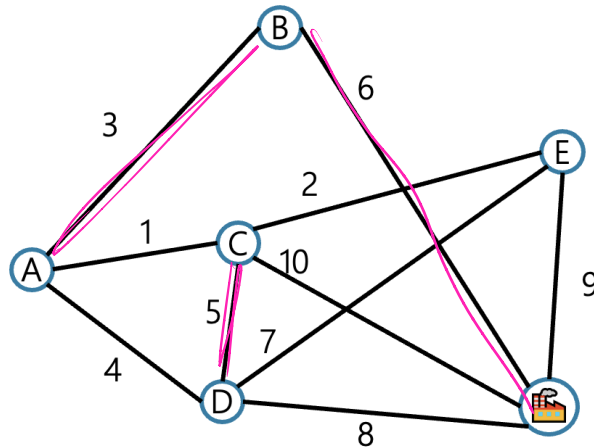
Yafqa Khan
Summer 2025

Announcements

- EX07 due today
- EX08 due Friday
- EX09 released today
- Exam 2 information posted here:
 - <https://courses.cs.washington.edu/courses/cse332/25su/exams/final.html>
 - **Note: it will be hard to accommodate makeups; only four days to grade**
 - If you can't make proposed makeup dates (e.g., sickness/emergency), some options:
 - Option 1: Exam 1 is worth 40% instead of 20% of overall grade
 - Option 2: Take the final exam in the next CSE 332 offering

MSTs

It's 2025! Your friend works at an internet company. They want to know where to build internet cables to connect all cities to the Internet.



They know how much it would cost to lay internet cables, and they want the cheapest way to make sure everyone can reach the server.

Minimum Spanning Trees

Given an undirected graph $G=(V, E)$, find a graph $G'=(V, E')$ such that:

- E' is a subset of E
- $|E'| = |V| - 1$ ← trees have $|V| - 1$
- G' is connected

G' is a minimum spanning tree.

- $\sum_{(u,v) \in E'} c_{uv}$ is minimal

Applications:

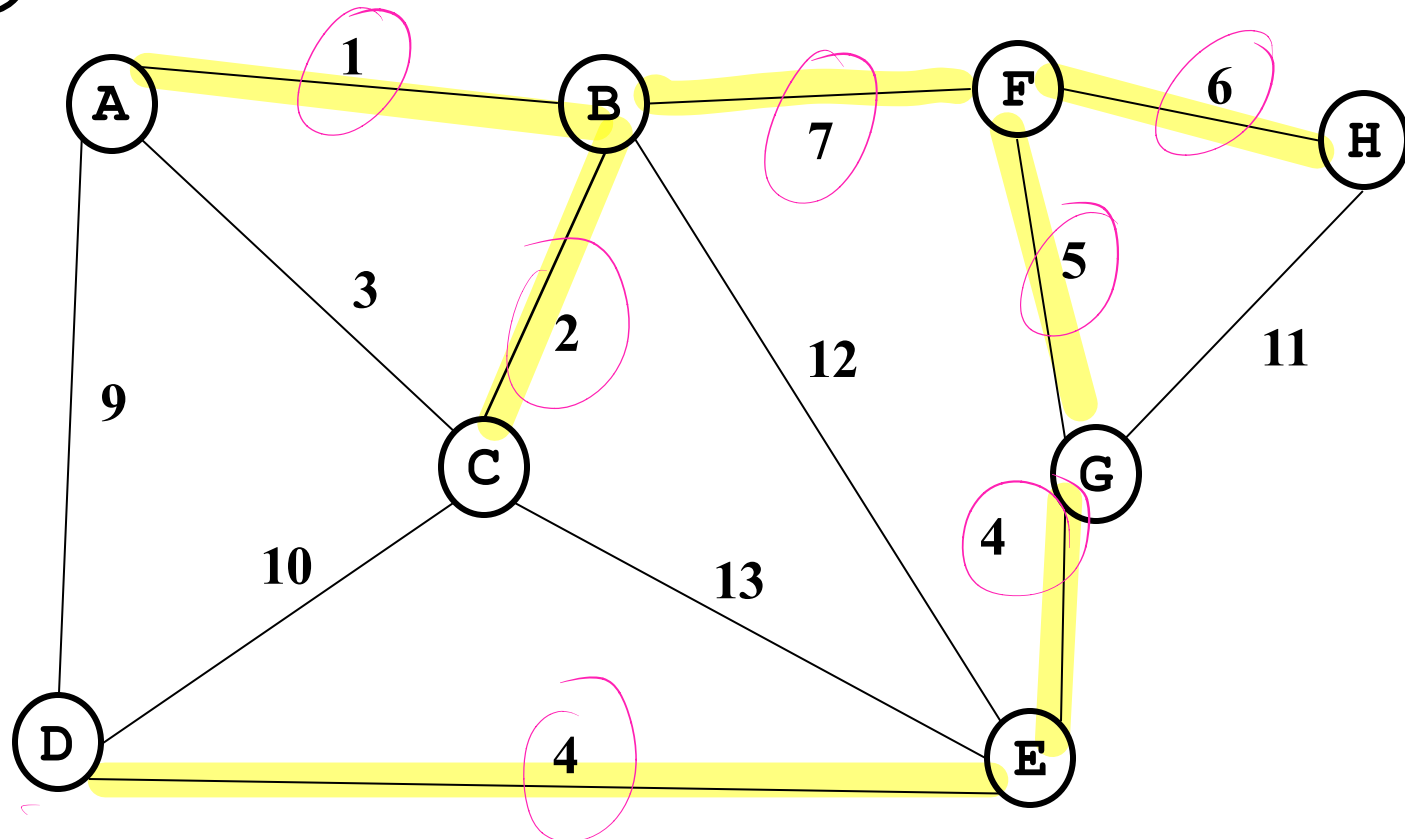
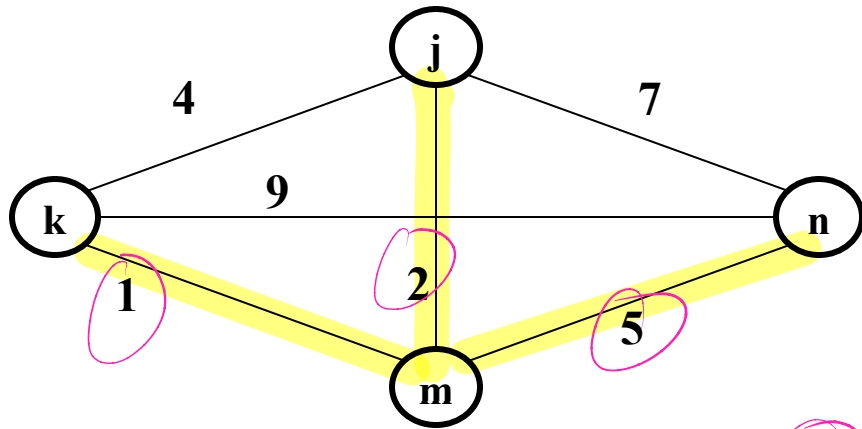
- Example: Electrical wiring for a house or clock wires on a chip
- Example: A road network if you cared about asphalt cost rather than travel time

Student Activity

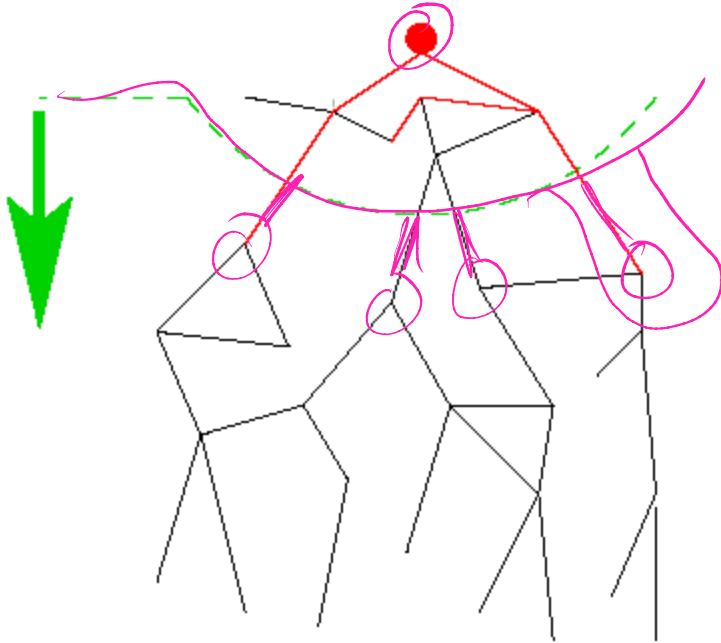
5 min

Find the MSTs

- a tree that
- connects all vertices
- $\sum w$ is minimal

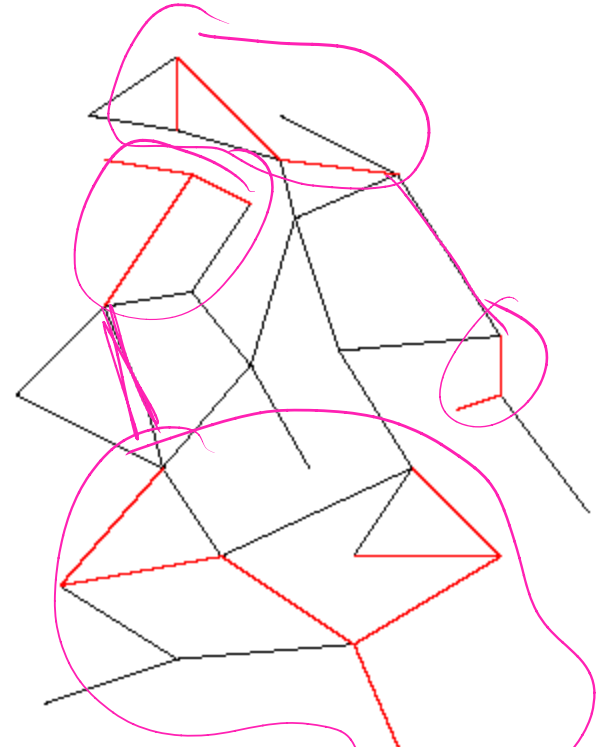


Two Different Approaches



Prim's Algorithm

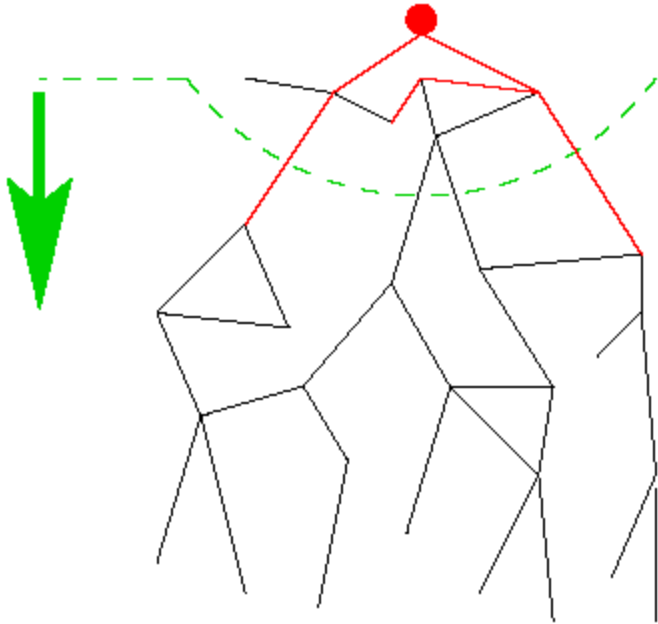
Almost identical to Dijkstra's



Kruskals's Algorithm

Completely different!

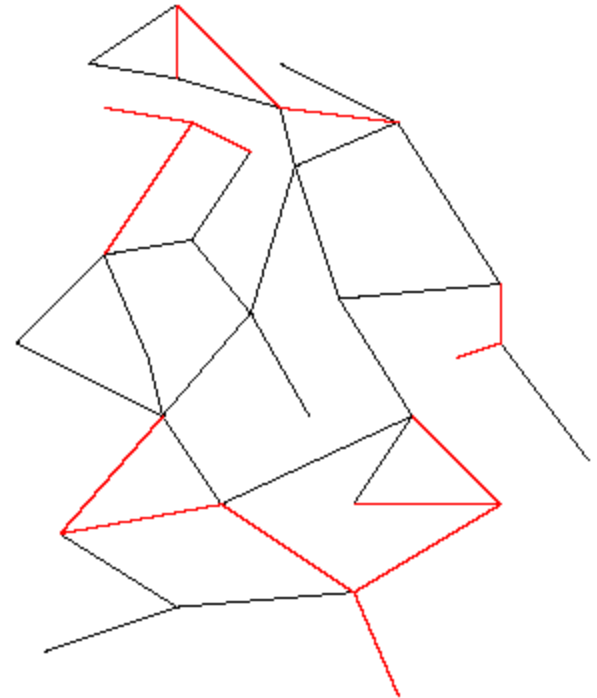
Two Different Approaches



Prim's Algorithm

Almost identical to Dijkstra's

One node, grow greedily



Kruskals's Algorithm

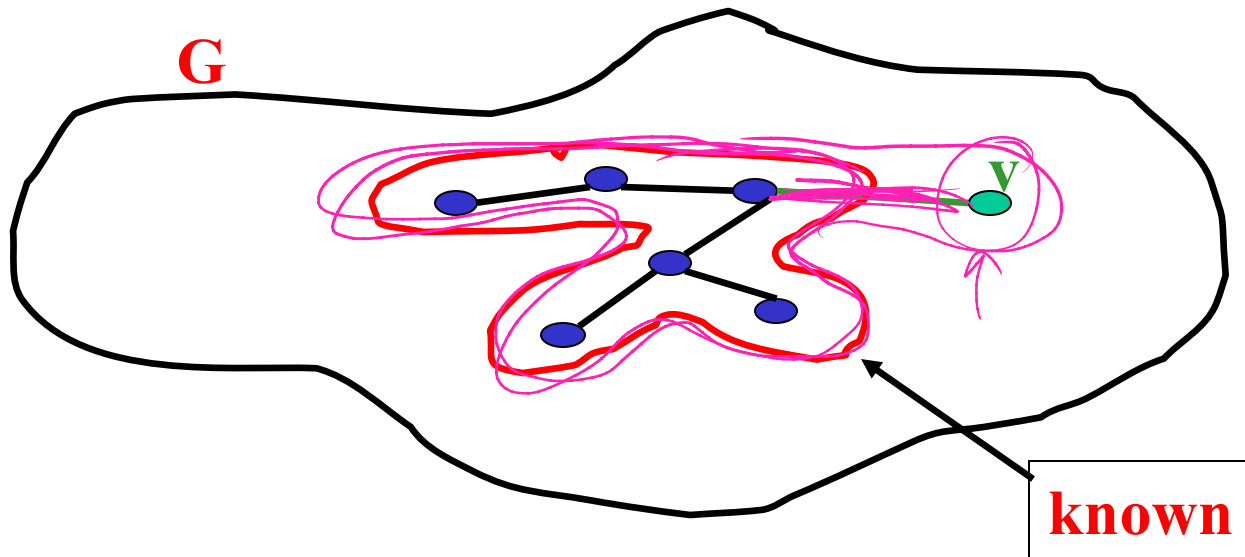
Completely different!

Forest of MSTs,
Union them together.
(Need a new data structure for this)

Prim's algorithm

Idea: Grow a tree by picking a vertex from the unknown set that has the smallest cost. Here cost = cost of the edge that connects that vertex to the known set. *Pick the vertex with the smallest cost that connects “known” to “unknown.”*


A node-based greedy algorithm
Builds MST by greedily adding nodes



Prim's Algorithm vs. Dijkstra's

Recall:

 **Dijkstra** picked the unknown vertex with smallest cost where
cost = distance to the source.

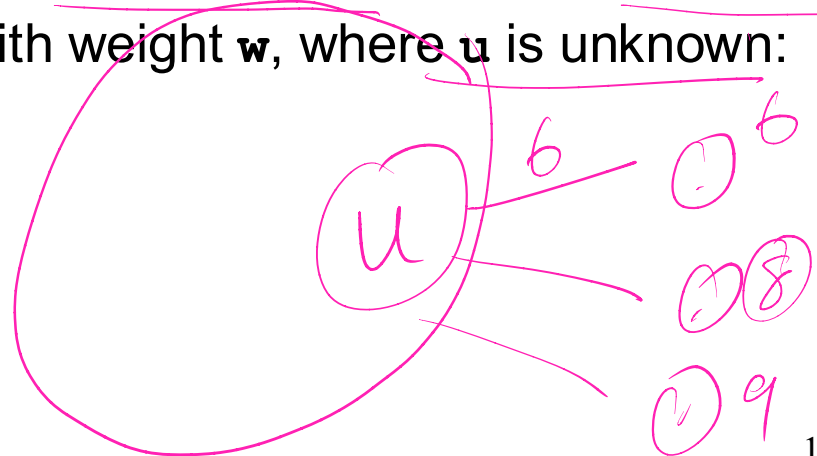
 **Prim's** pick the unknown vertex with smallest cost where
cost = distance from this vertex to the known set (in other words,
the cost of the smallest edge connecting this vertex to the known
set)

- Otherwise identical
- Compare to slides in Dijkstra lecture!

Prim's Algorithm for MST

1. For each node v , set $v.cost = \infty$ and $v.known = false$
2. Choose any node v . (this is like your "start" vertex in Dijkstra)
 - a) Mark v as known
 - b) For each edge (v, u) with weight w :
set $u.cost = w$ and $u.prev = v$
3. While there are unknown nodes in the graph
 - a) Select the unknown node v with lowest cost
 - b) Mark v as known and add $(v, v.prev)$ to output (the MST)
 - c) For each edge (v, u) with weight w , where u is unknown:

```
if ( $w < u.cost$ ) {  
     $u.cost = w$ ;  
     $u.prev = v$ ;  
}
```



Prim's Analysis

- Run-time
 - Same as Dijkstra
 - $O(\underbrace{|E| \log |V|})$ using a priority queue

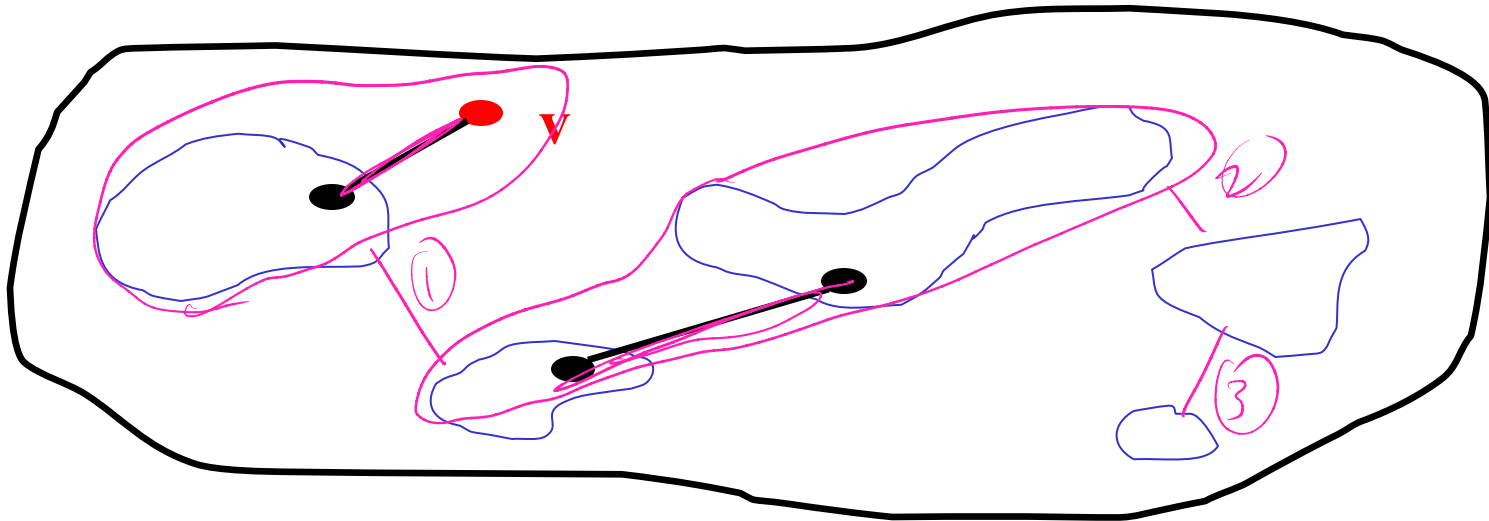
min heap

Kruskal's MST Algorithm

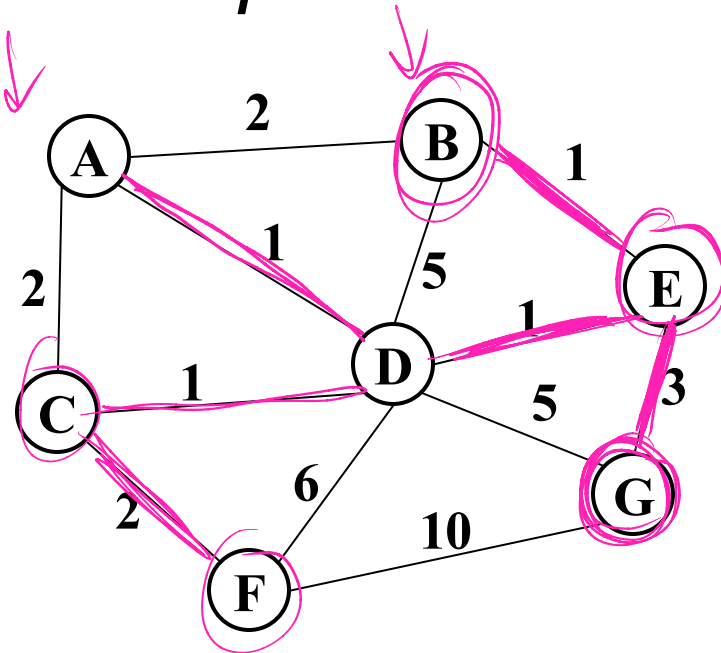
a set of tree

Idea: Grow a **forest** out of edges that do not create a cycle. Pick an edge with the smallest weight.

$G=(V,E)$



Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: ~~(A,B)~~, ~~(C,F)~~, ~~(A,C)~~

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

10: (F,G)

Output:

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Algorithm for MST

An edge-based greedy algorithm

Builds MST by greedily adding edges

1. Initialize with
 - empty MST
 - all vertices marked unconnected
 - all edges unmarked
2. While all vertices are not connected
 - a. Pick the lowest cost edge (u, v) and mark it
 - b. If u and v are not already connected, add (u, v) to the MST and mark u and v as connected to each other

Aside: Union-Find aka Disjoint Set ADT

- **Union(x,y)** – take the union of two sets named x and y
 - Given sets: {3, 5, 7}, {4, 2, 8}, {9}, {1, 6}
 - **Union(5,1)**
Result: {3, 5, 7, 1, 6}, {4, 2, 8}, {9},
- To perform the union operation, we replace sets x and y by $(x \cup y)$

- **Find(x)** – return the name of the set containing x.
 - Given sets: {3, 5, 7, 1, 6}, {4, 2, 8}, {9},
 - **Find(1)** returns 5
 - **Find(4)** returns 8

add edge (A,B)
to MST

find(A)
fin(B)

- We can do Union in constant time.
- We can get Find to be **amortized** constant time
(worst case $O(\log n)$ for an individual Find operation).

Kruskal's pseudo code

```
void Graph::kruskal() {  
    int edgesAccepted = 0;  
    DisjSet s(NUM_VERTICES);  
  
    while (edgesAccepted < NUM_VERTICES - 1) {  
        e = smallest weight edge not deleted yet;  
        // edge e = (u, v)  
        uset = s.find(u); ←  
        vset = s.find(v);  
        if (uset != vset) {  
            edgesAccepted++;  
            s.unionSets(uset, vset); ←  
        }  
    }  
}
```


Kruskal's pseudo code

```
void Graph::kruskal() {
    int edgesAccepted = 0;
    DisjSet s(NUM_VERTICES);
```

```
while (edgesAccepted < NUM_VERTICES - 1) {
    e = smallest weight edge not deleted yet;
    // edge e = (u, v)
```

```
    uset = s.find(u);
    vset = s.find(v);
```

```
    if (uset != vset) {
        edgesAccepted++;
```

```
        s.unionSets(uset, vset);
    }
```

```
}
}
```

```
}
```

On heap of
edges
Delete min =
 $\log |E|$

|E| heap ops

|E| delete Min

2|E| finds

One for each
vertex in the
edge
Find = $\log |V|$

|V| unions

Union = $O(1)$

$|E| \log |E| + 2|E| \log |V| + |V|$

$O(|E| \log |E|) = O(|E| \log |V|)$

b/c $\log |E| < \log |V|^2 = 2 \log |V|$

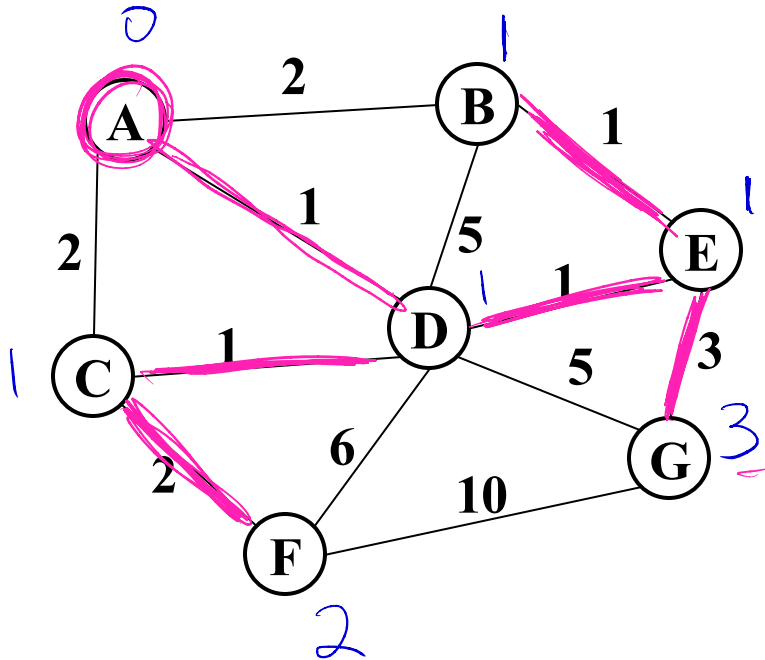
$O(|E| \log |E|)$

or $O(|E| \log |V|)$

Prim's

5 min.

Example: Find MST using ~~Kruskal's~~



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

10: (F,G)

Output:

Note: At each step, the union/find sets are the trees in the forest