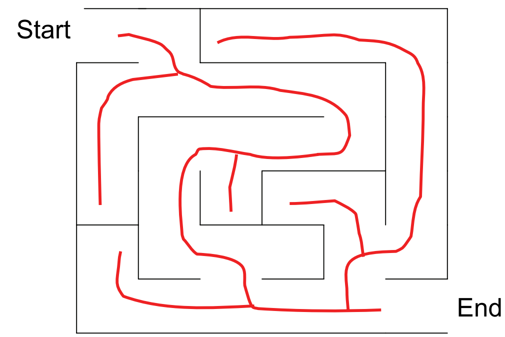
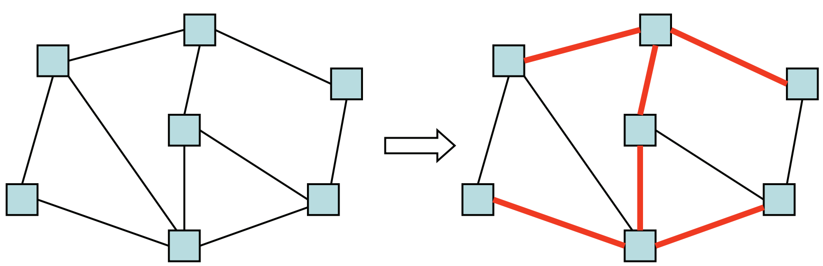
CSE 326: Data Structures Spanning Trees

Brian Curless
Spring 2008

A Hidden Tree



Spanning Tree in a Graph

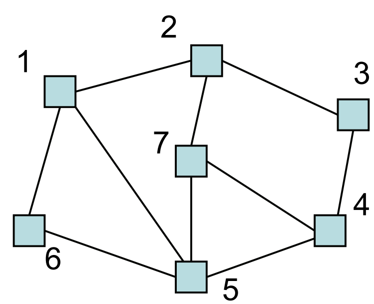


Vertex = router
Edge = link between routers

Spanning tree
- Connects all the vertices
- No cycles

Undirected Graph

- G = (V,E)
  – V is a set of vertices (or nodes)
  – E is a set of unordered pairs of vertices



V = {1,2,3,4,5,6,7}
E = {(1,2),(1,6),(1,5),(2,7),(2,3),
(3,4),(4,7),(4,5),(5,6)}

2 and 3 are adjacent
2 is incident to edge (2,3)

Spanning Tree Problem

- Input: An undirected graph G = (V,E). G is connected.
- Output: T contained in E such that
  - (V,T) is a connected graph
  - (V,T) has no cycles

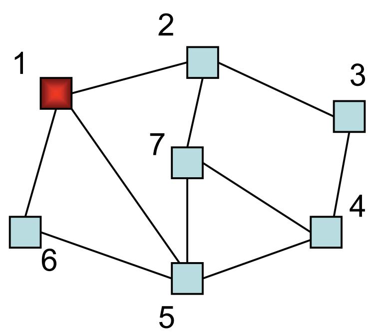
Spanning Tree Algorithm

```
ST(Vertex i) {
   mark i;
   for each j adjacent to i  {
      if (j is unmarked) {
         Add (i,j) to T;
         ST(j);
      }
   }
}
```

```
Main( ) {
  T = empty set;
  ST(1);
}
```

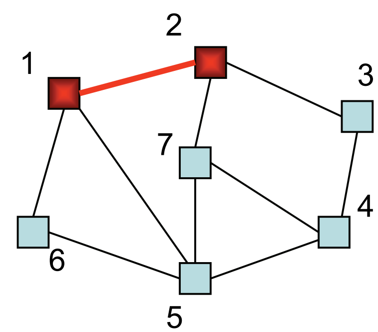

Example of Depth First Search

ST(1)



Example Step 2

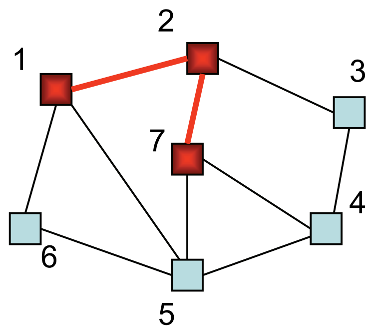
ST(1)
ST(2)



(1,2)

Example Step 3

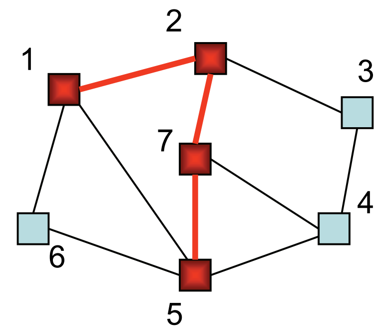
ST(1)
ST(2)
ST(7)



(1,2) (2,7)

Example Step 4

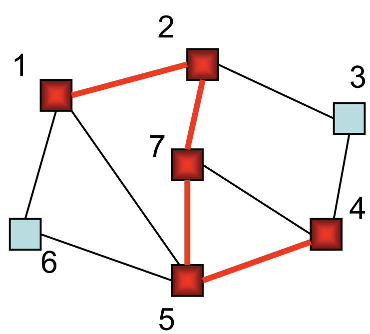
ST(1)
ST(2)
ST(7)
ST(5)



(1,2) (2,7) (7,5)

Example Step 5

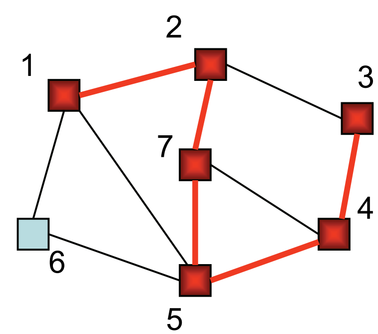
ST(1)
ST(2)
ST(7)
ST(5)
ST(4)



(1,2) (2,7) (7,5) (5,4)

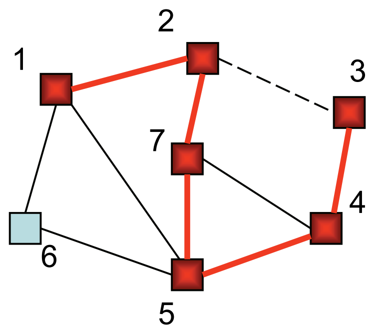
Example Step 6

ST(1)
ST(2)
ST(7)
ST(5)
ST(4)
ST(3)



(1,2) (2,7) (7,5) (5,4) (4,3)

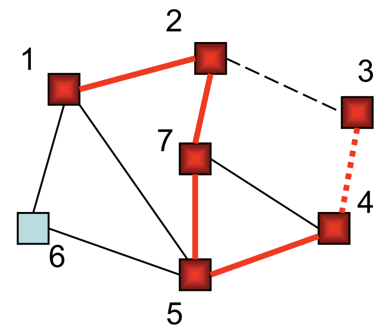
Example Step 7



ST(1)
ST(2)
ST(7)
ST(5)
ST(4)
ST(3)

(1,2) (2,7) (7,5) (5,4) (4,3)

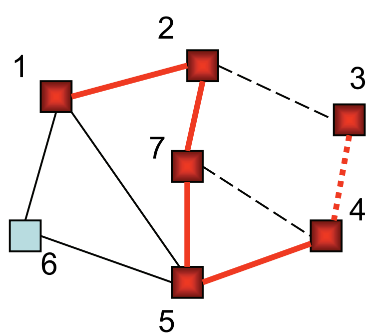
Example Step 8



ST(1)
ST(2)
ST(7)
ST(5)
ST(4)

(1,2) (2,7) (7,5) (5,4) (4,3)

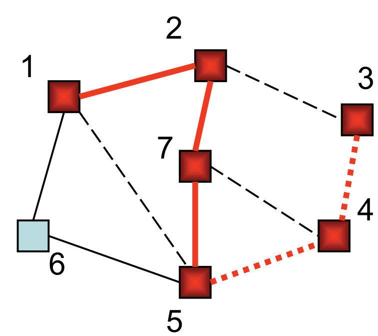
Example Step 9



ST(1)
ST(2)
ST(7)
ST(5)
ST(4)

(1,2) (2,7) (7,5) (5,4) (4,3)

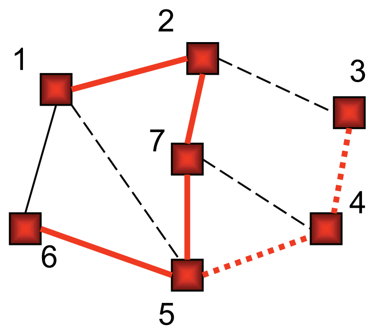
Example Step 10



ST(1)
ST(2)
ST(7)
ST(5)

(1,2) (2,7) (7,5) (5,4) (4,3)

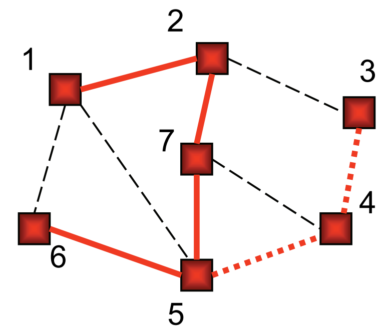
Example Step 11



ST(1)
ST(2)
ST(7)
ST(5)
ST(6)

(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

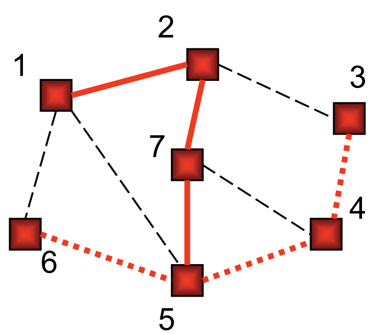
Example Step 12



ST(1)
ST(2)
ST(7)
ST(5)
ST(6)

(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

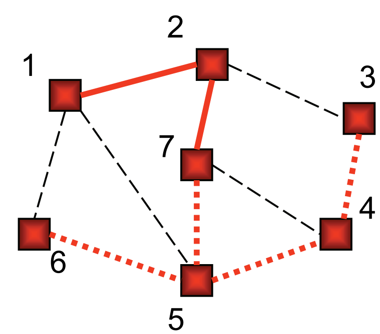
Example Step 13



ST(1)
ST(2)
ST(7)
ST(5)

(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

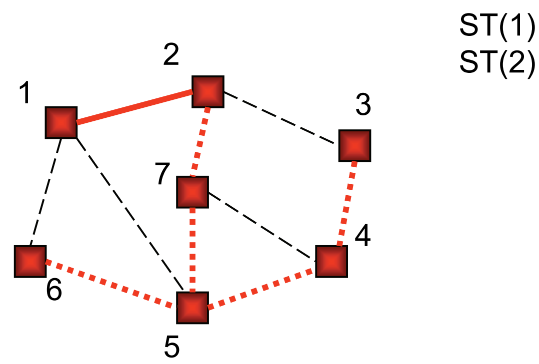
Example Step 14



ST(1)
ST(2)
ST(7)

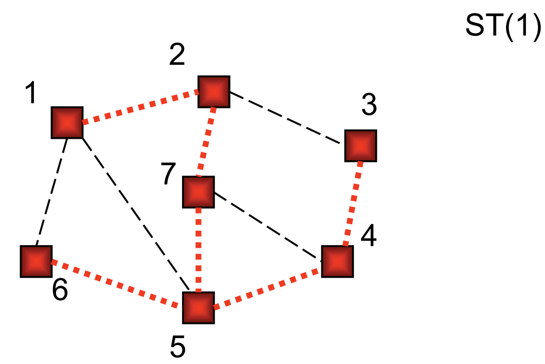
(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

Example Step 15



(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

Example Step 16



(1,2) (2,7) (7,5) (5,4) (4,3) (5,6)

How many edges in a spanning tree?

Before moving on, it will help us to know how many edges a spanning tree must have.

First, a couple of properties of trees…

A tree has one vertex of degree one

Property: a tree with |V|>1 has at least one vertex of degree one (i.e., with only one edge incident to it).

Proof by contradiction:

- Assume a “tree” with no such vertex.
- The graph has no endpoints and must contain a cycle.
- The graph is not a tree.

Removing a vertex of degree one gives a tree

Property: removing a vertex of degree one and the corresponding edge from a tree results in a tree.

Proof:

- Removing an edge cannot introduce cycles.
- Removing a vertex of degree one will not result in a disconnected graph.

So, the graph is acyclic and connected, i.e., a tree.

A spanning tree has |V|-1 edges

Property: a spanning tree over a graph with |V| vertices has |V|-1 edges.

Proof by induction:

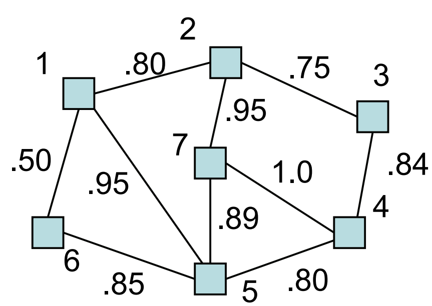
- Base case: 1 vertex → |V|-1 = 0 edges.
- Inductive hypothesis: a spanning tree with k-1 vertices has k-2 edges.
- Inductive step: spanning tree with k vertices has k-1 edges.

We must prove the inductive step:

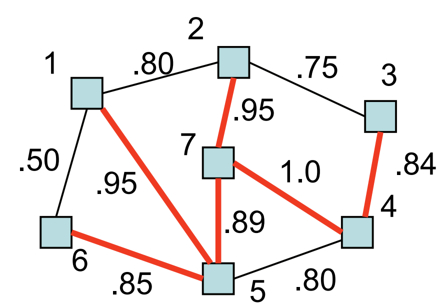
- Spanning tree with k vertices has at least one vertex of degree 1.
- Remove that vertex and its edge from the graph and the spanning tree.
- The result is a spanning tree of k-1 vertices, which must have k-2 edges (inductive hypothesis).
- Restoring that vertex and edge gives a tree with k vertices and k-1 edges.

Best Spanning Tree

- Each edge has the probability that it won't fail
- Find the spanning tree that is least likely to fail



Example of a Spanning Tree



Probability of success = .85 x .95 x .89 x .95 x 1.0 x .84
= .5735

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

G' is a **minimum spanning tree**.

Applications: wiring a house, power grids, Internet connections

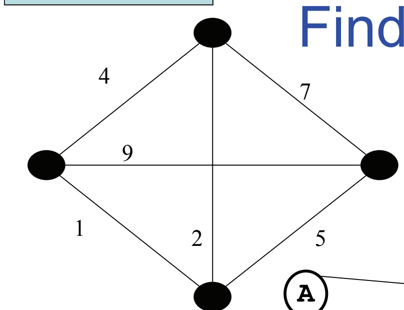
Minimum Spanning Tree Problem

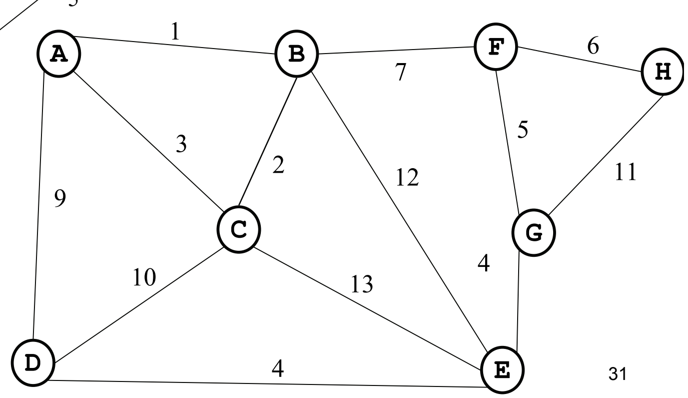
- Input: Undirected Graph G = (V,E) and a cost function C from E to the reals. C(e) is the cost of edge e.
- Output: A spanning tree T with minimum total cost. That is: T that minimizes

$$C(T) = \sum_{e\in T} C(e)$$

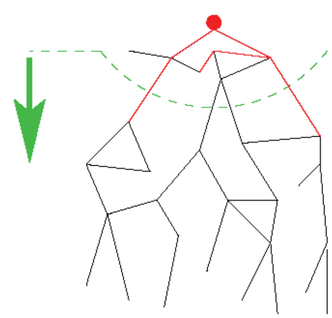
Your Turn

Find the MST



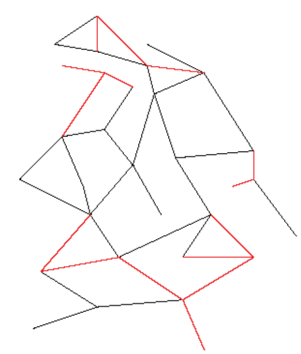


Two Different Approaches



Prim's Algorithm

Almost identical to Dijkstra's

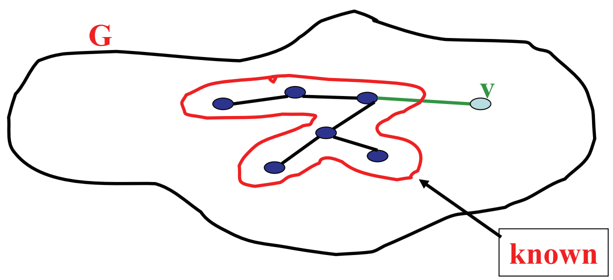


Kruskals's Algorithm

Completely different!

Prim's algorithm

Idea: Grow a tree by adding an edge from the "known" vertices to the "unknown" vertices. Pick the edge with the smallest weight.



Prim's Algorithm for MST

A *node-based* greedy algorithm

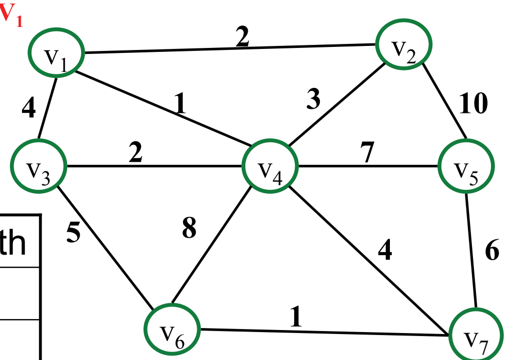
Builds MST by greedily adding nodes

1. Select a node to be the "root"
   - mark it as known
   - Update cost of all its neighbors
2. While there are unknown nodes left in the graph
   a. Select an unknown node *b* with the smallest cost from some *known* node *a*
   b. Mark *b* as known
   c. Add (*a*, *b*) to MST
   d. Update cost of all nodes adjacent to *b*

Your Turn

Find MST using Prim's

Start with V_1



V	Kwn	Distance	path
v1			
v2			
v3			
v4			
v5			
v6			
v7			

Order Declared Known:

V_1

Prim's Algorithm Analysis

Running time:

Same as Dijkstra's: O(|E| log |V| + |V| log |V|)

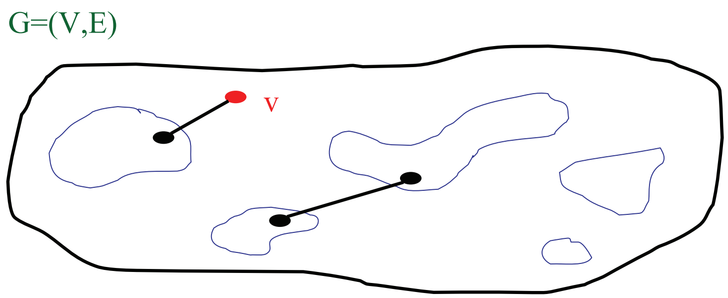
Can we reduce this?

Correctness:

Proof is similar to Dijkstra's

Kruskal's MST Algorithm

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



Kruskal's Algorithm for MST

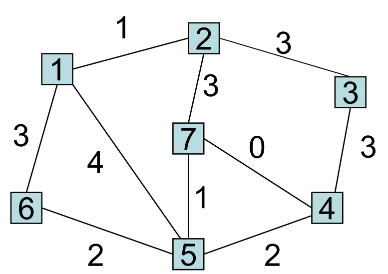
An *edge-based* greedy algorithm

Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   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

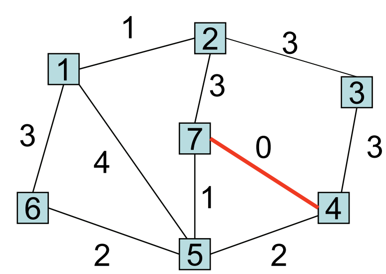
Doesn't it sound familiar?

Example of Kruskal 1



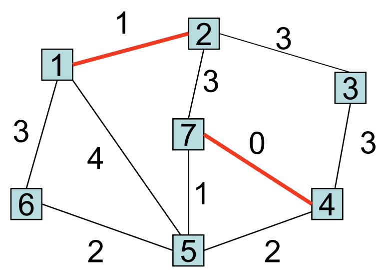
(7,4)	(2,1)	(7,5)	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
0	1	1	2	2	3	3	3	3	4

Example of Kruskal 2



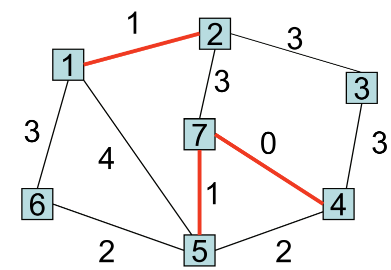
~~(7,4)~~	(2,1)	(7,5)	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	1	1	2	2	3	3	3	3	4

Example of Kruskal 2



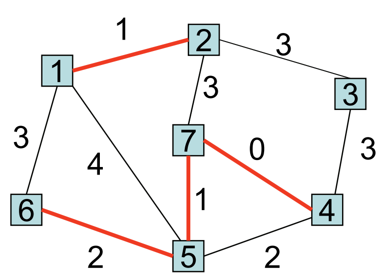
~~(7,4)~~	~~(2,1)~~	(7,5)	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	1	2	2	3	3	3	3	4

Example of Kruskal 3



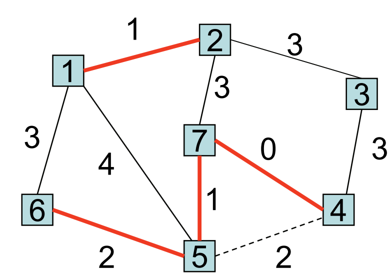
~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	2	2	3	3	3	3	4

Example of Kruskal 4



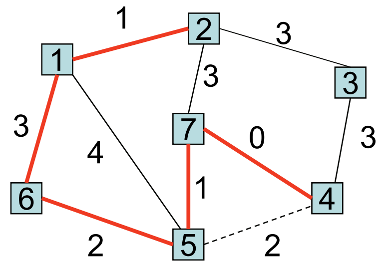
~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	~~2~~	2	3	3	3	3	4

Example of Kruskal 5



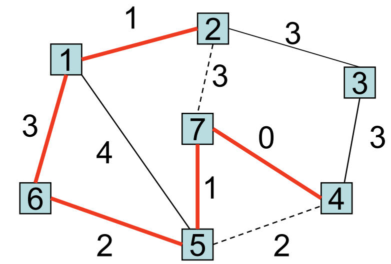
~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	~~(5,4)~~	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	3	3	3	3	4

Example of Kruskal 6



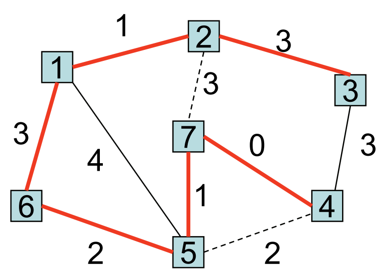
~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	~~(5,4)~~	~~(1,6)~~	(2,7)	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	3	3	3	4

Example of Kruskal 7



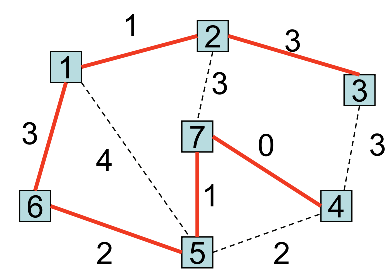
~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	~~(5,4)~~	~~(1,6)~~	~~(2,7)~~	(2,3)	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	3	3	4

Example of Kruskal 7



~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	~~(5,4)~~	~~(1,6)~~	~~(2,7)~~	~~(2,3)~~	(3,4)	(1,5)
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	3	4

Example of Kruskal 8,9



~~(7,4)~~	~~(2,1)~~	~~(7,5)~~	~~(5,6)~~	~~(5,4)~~	~~(1,6)~~	~~(2,7)~~	~~(2,3)~~	~~(3,4)~~	~~(1,5)~~
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	~~3~~	~~4~~

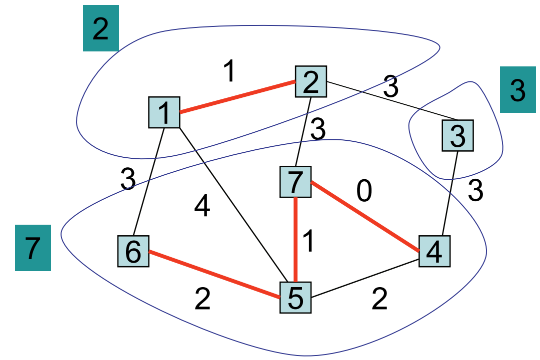
Data Structures for Kruskal

- Sorted edge list

  (7,4) (2,1) (7,5) (5,6) (5,4) (1,6) (2,7) (2,3) (3,4) (1,5)
  0 1 1 2 2 3 3 3 3 4

- Disjoint Union / Find
  – Union(*a,b*) - union the disjoint sets named by *a* and *b*
  – Find(a) returns the name of the set containing *a*

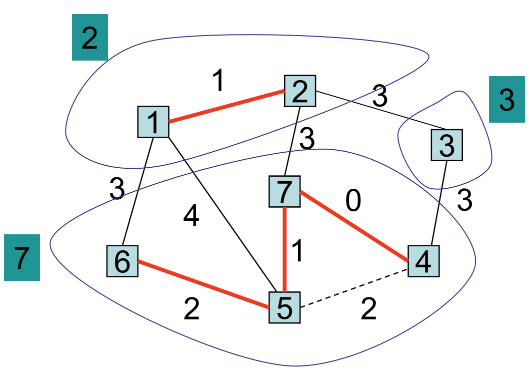
Example of DU/F 1



Find(5) = 7
Find(4) = 7

(7,4) (2,1) (7,5) (5,6) (5,4) (1,6) (2,7) (2,3) (3,4) (1,5)
0 1 1 2 2 3 3 3 3 4

Example of DU/F 2

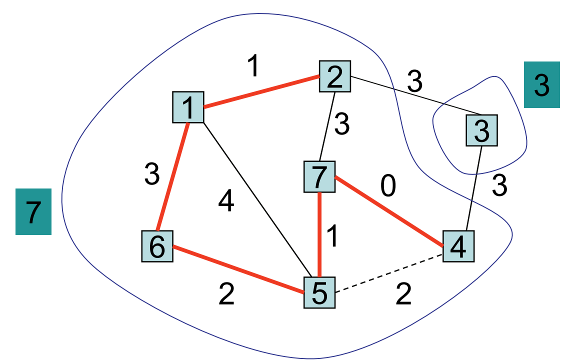


Find(1) = 2
Find(6) = 7

(7,4) (2,1) (7,5) (5,6) (5,4) (1,6) (2,7) (2,3) (3,4) (1,5)
0 1 1 2 2 3 3 3 3 4

Example of DU/F 3

Union(1,7)



(7,4) (2,1) (7,5) (5,6) (5,4) (1,6) (2,7) (2,3) (3,4) (1,5)
0 1 1 2 2 3 3 3 3 4

Kruskal's Algorithm with DU / F

```
Sort the edges by increasing cost;
Initialize A to be empty;
for each edge (i,j) chosen in increasing order do
   u := Find(i);
   v := Find(j);
   if not(u = v) then
      add (i,j) to A;
      Union(u,v);
```

This algorithm will work, but it goes through all the edges.

Is this always necessary?

Kruskal 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);
    }
  }
}
```

|E| heap ops

2|E| finds

|V| unions

Total Cost:

Kruskal's Algorithm: Correctness

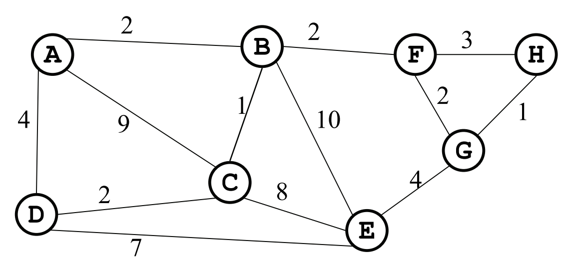
It clearly generates a spanning tree. Call it T_K.

Suppose T_K is *not* minimum:

- Pick another spanning tree T_{min} with *lower cost* than T_K
- Pick the smallest edge $e_1=(u,v)$ in T_K that is not in T_{min}
- T_{min} already has a path p in T_{min} from u to v
  - $\Rightarrow$ Adding e_1 to T_{min} will create a cycle in T_{min}
- Pick an edge e_2 in p that Kruskal's algorithm considered *after* adding e_1 (must exist: u and v unconnected when e_1 considered)
  - $\Rightarrow$ cost(e_2) $\geq$ cost(e_1)
  - $\Rightarrow$ can replace e_2 with e_1 in T_{min} without increasing cost!
- Keep doing this until T_{min} is identical to T_K
  - $\Rightarrow$ T_K must also be minimal – contradiction!

Your Turn

Find MST using Prim's and Kruskal's



Reducing Best to Minimum

Let P(e) be the probability that an edge doesn't fail.
Define:

$$C(e) = -\log_{10}(P(e))$$

Minimizing $\sum_{e \in T} C(e)$

is equivalent to maximizing $\prod_{e \in T} P(e)$

because $\prod_{e \in T} P(e) = \prod_{e \in T} 10^{-C(e)} = 10^{-\sum_{e \in T} C(e)}$