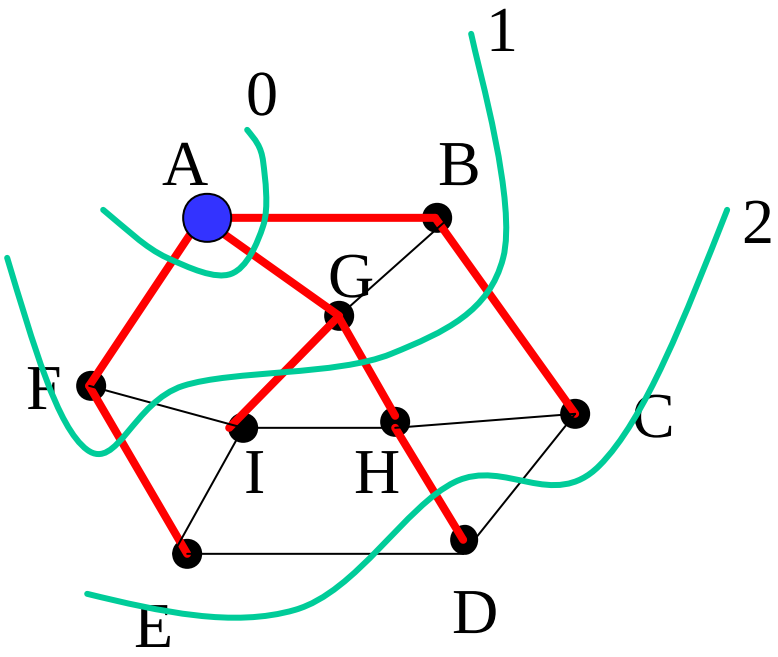
Minimal Spanning Trees

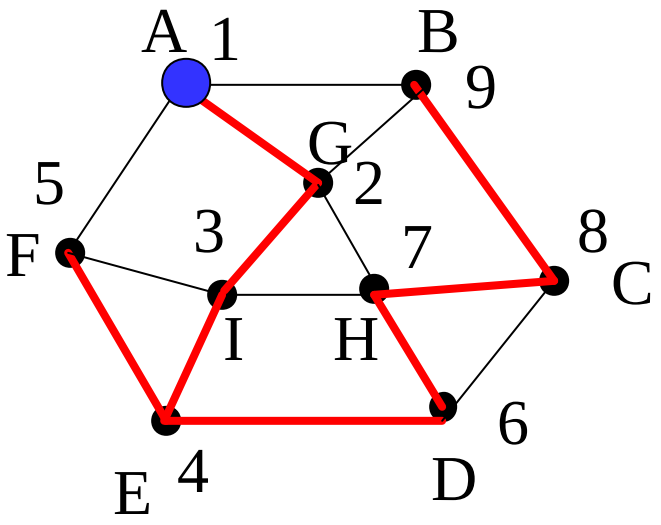
Spanning Tree

- Assume you have an undirected graph G = (V,E)
- Spanning tree of graph G is tree $T = (V, E_T \subseteq E, R)$
  - Tree has same set of nodes
  - All tree edges are graph edges
  - Root of tree is R
- Think: “smallest set of edges needed to connect everything together”

Spanning trees



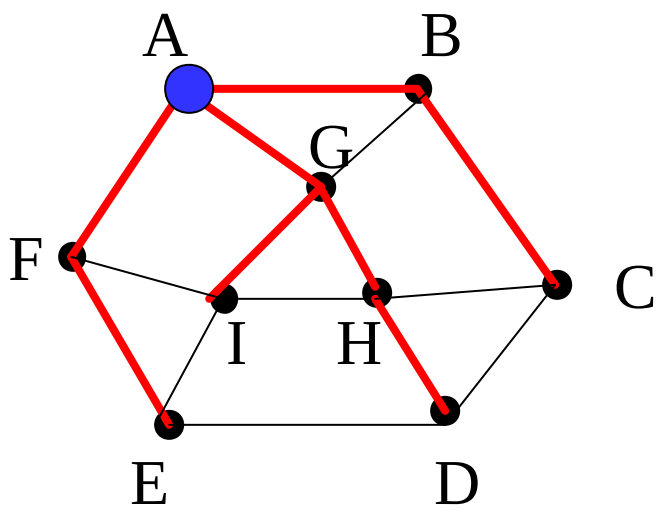
Breadth-first Spanning Tree



Depth-first spanning tree

Property 1 of spanning trees

- Graph: G = (V,E) , Spanning tree: T = (V,E_T,R)
- For any edge c in G but not in T, there is a simple cycle containing only edge c and edges in spanning tree.



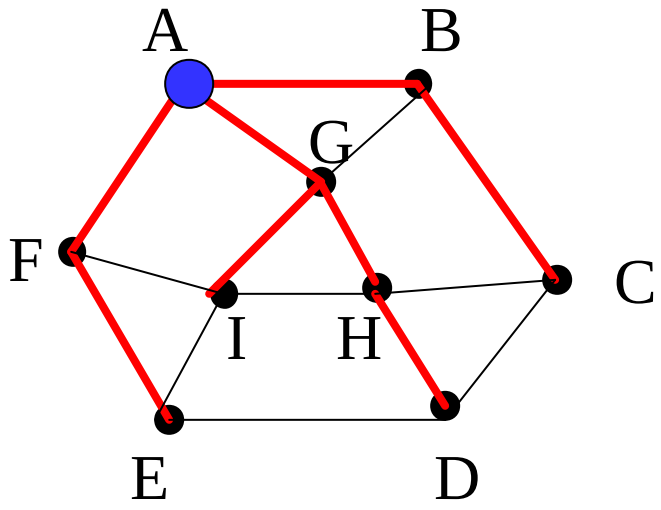
edge (I,H):
simple cycle is (I,H,G,I)

edge (H,C):
simple cycle is (H,C,B,A,G,H)

Proof?

Proof of Property 1

- Edge is c goes u ⟷ v
- If u is ancestor of v, result is easy (u ↷ v, then v ↔ u form a cycle)
- Otherwise, there are paths root ↷ u and root ↷ v (b/c it is a tree). Let p be the node furthest from root on both of these paths. Now p ↷ u, then u ↔ v, then v ↷ p form a cycle.



edge (I,H):
p is node G
simple cycle is (I,H,G,I)

edge (H,C):
p is node A
simple cycle is (H,C,B,A,G,H)

Useful lemma about trees

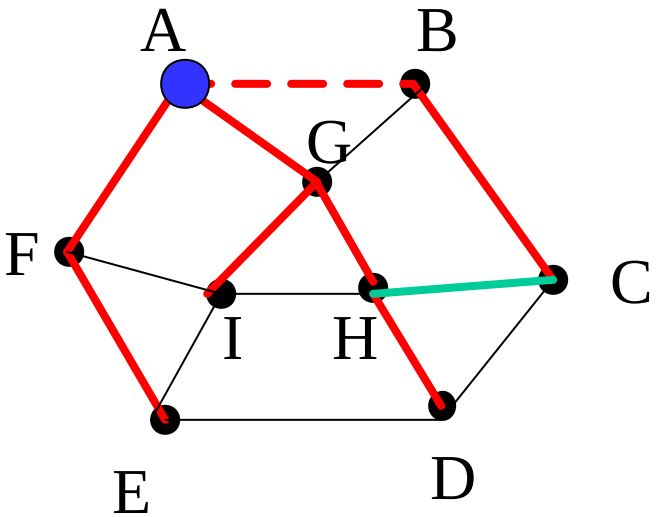
- In any tree T = (V,E), |E|=|V| -1
  - Proof?

Useful lemma about trees

- In any tree T = (V,E), |E|=|V| -1
  - Proof: (by induction on |V|)
    * If |V| = 1, we have the trivial tree containing a single node, and the result is obviously tree.
    * Assume result is true for all trees for which 1 <= |V| <n, and consider a tree S=(E_S, V_S) with |V| = n. Such a tree must have at least one leaf node; removing the leaf node and edge incident on that node gives a smaller tree T with less than n nodes. By inductive assumption, $|E_T| = |V_T|+1$. Since $|E_S| = |E_T|+1$ and $|V_S|=|V_T|+1$, the required result follow.
- Converse also true: an undirected graph G = (V,E) which
  (1) has a single connected component, and
  (2) has |E| = |V|-1
  → must be a tree.

Property 2 of spanning trees

- Graph: G = (V,E), Spanning tree: T = (V,E_T,R)
- For any edge c in G but not in T, there is a simple cycle Y containing only edge c and edges in spanning tree.
- Moreover, inserting edge c into T and deleting any edge in Y gives another spanning tree T'.



edge (H,C):

simple cycle is (H,C,B,A,G,H)

adding (H,C) to T and deleting (A,B) gives another spanning tree

Proof of Property 2 - Outline

- T’ is a connected component.
  - Proof?

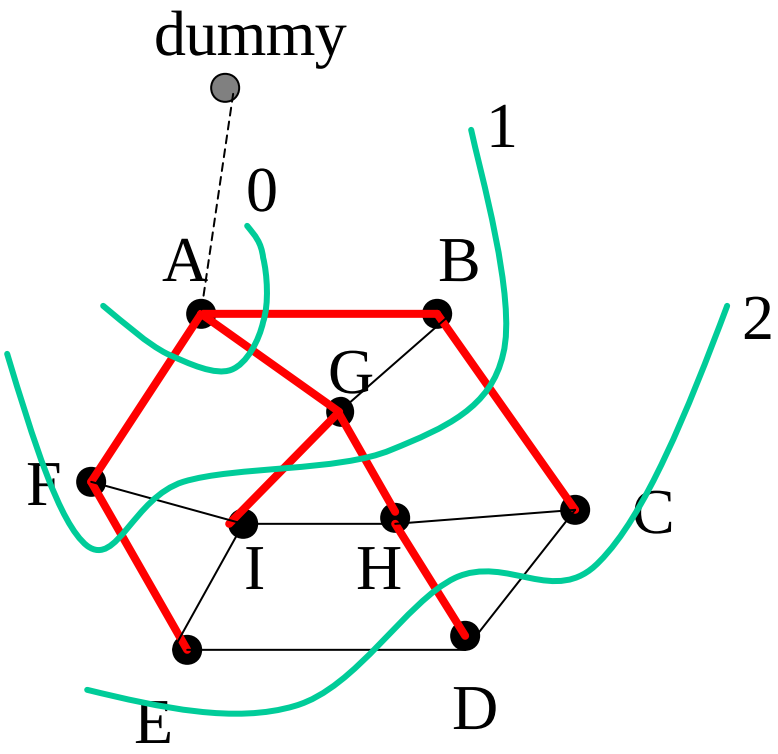
- In T’, numbers of edges = number of nodes –1
  - Proof ?

- Therefore, from lemma earlier, T’ is a tree.

Proof of Property 2

- T' is a connected component.
  - Otherwise, assume node a is not reachable from node b in T'. In T, there must be a path from b to a that contains edge (s? t). In this path, replace edge (s? t) by the path in T' obtained by deleting (s? t) from the cycle Y, which gives a path from b to a. Contradiction, thus a must be reachable from b
- In T', numbers of edges = number of nodes –1
  - Proof: by construction of T' and fact that T is a tree. T' is same as T, with one edge removed, one edge added.
- Therefore, from lemma, T' is a tree.

Building BFS/DFS spanning trees



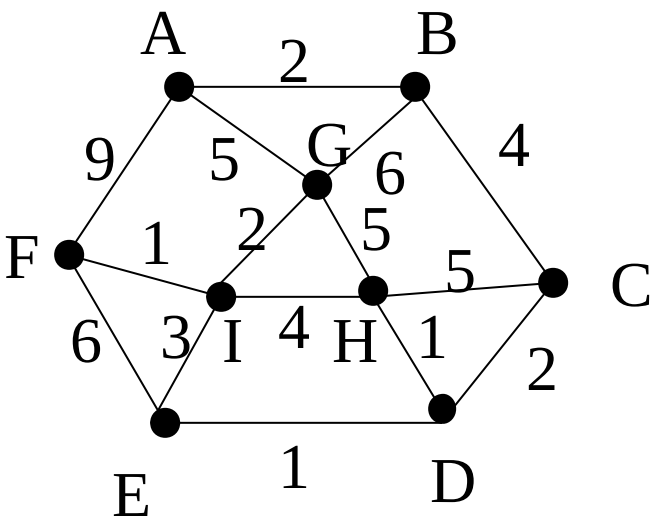
- Use sequence structure as before, but put/get edges, not nodes
  - Get edge (s,d) from structure
  - If d is not in done set,
    - add d to done set
    - (s,d) is in spanning tree
    - add out-edges (d,t) to seq structure if t is not in done set
- Example: BFS (Queue)

  [(dummy,A)]
  [(A,B),(A,G),(A,F)]
  [(A,G),(A,F),(B,G),(B,C)]…..

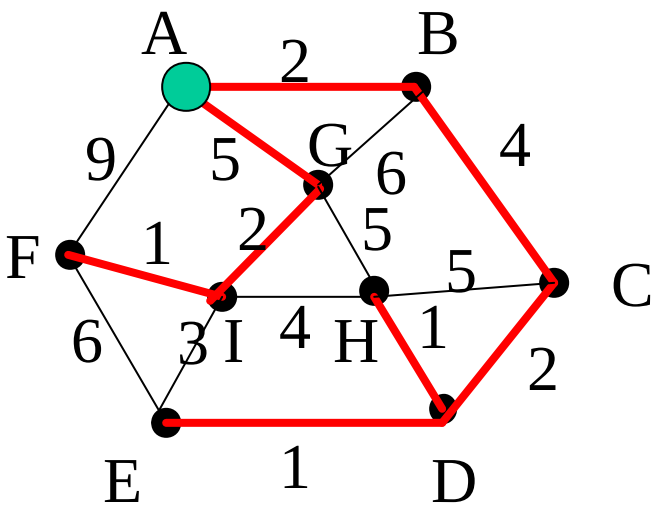
Weighted Spanning Trees

- Assume you have an undirected graph G = (V,E) with weights on each edge
- Spanning tree of graph G is tree $T = (V, E_T \subseteq E)$
  - Tree has same set of nodes
  - All tree edges are graph edges
  - Weight of spanning tree = sum of tree edge weights
- Minimal Spanning Tree (MST)
  - Any spanning tree whose weight is minimal
  - In general, a graph has several MST's
  - Applications: circuit-board routing, networking, etc.

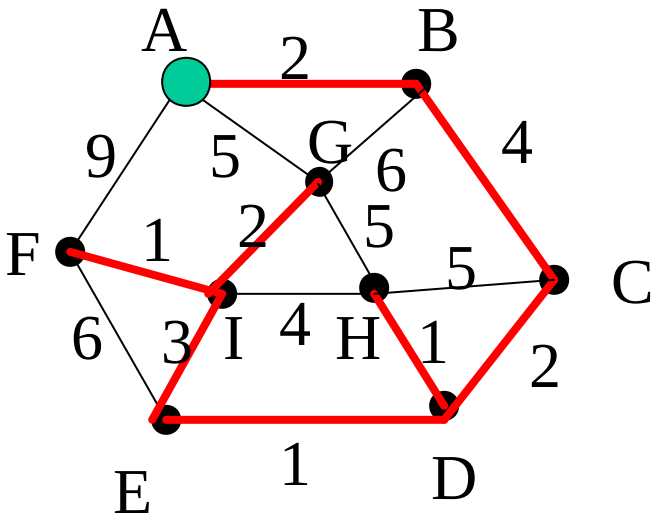
Example



Graph

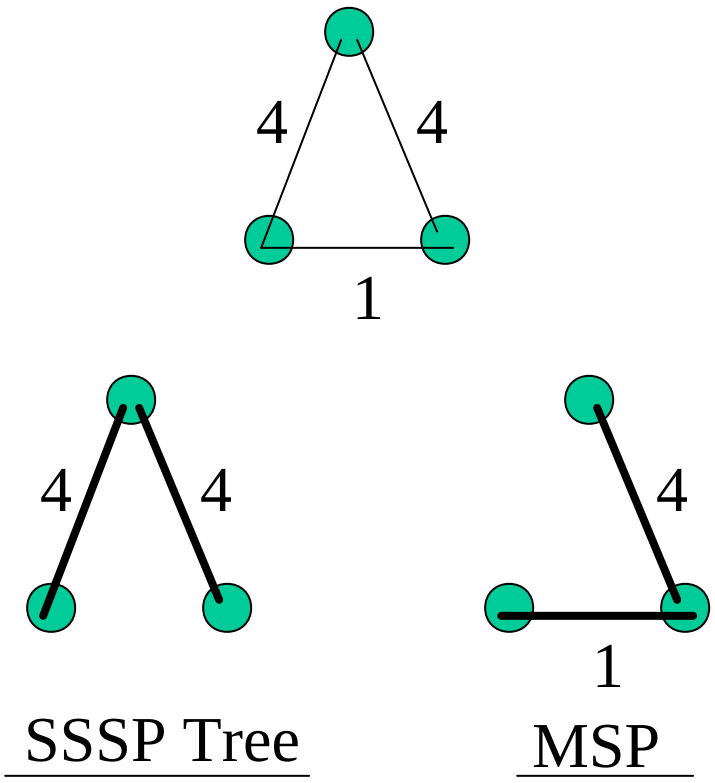


SSSP tree



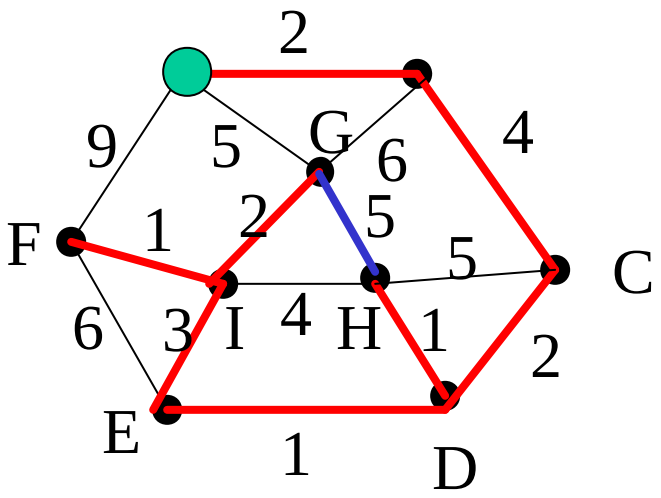
Minimal spanning tree

Caution: in general, SSSP tree is not MST



- Intuition:
  - SSSP: fixed start node
  - MST: at any point in construction, we have a bunch of nodes that we have reached, and we look at the shortest distance from any one of those nodes to a new node

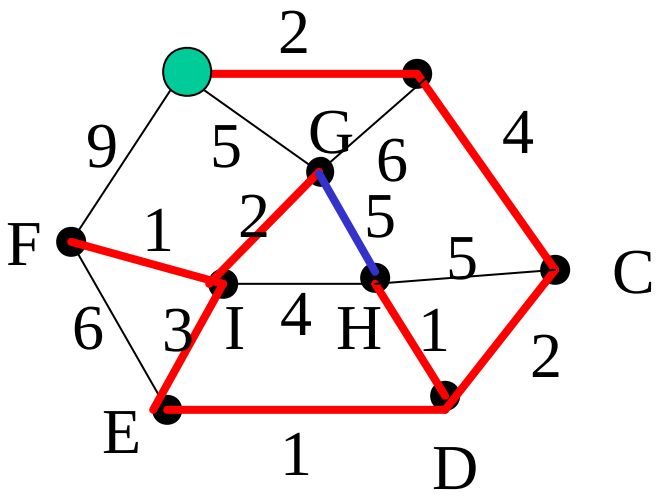
Property 3 of minimal spanning trees



Edge(G→H): 5
Cycle edges: (G→I), (I→E), (E→D),(H→D) all have weights less than (G→H)

- Graph: G = (V,E) , Spanning tree: T = (V,E_T,R)
- For any edge: c in G but not in T, there is a simple cycle Y containing only edge c and edges in spanning tree (already proved).
- Moreover, weight of c must be greater than or equal to weight of any edge in this cycle.
  - Proof?

Property 3 of minimal spanning trees



Edge(G→H): 5
Cycle edges: (G→I), (I→E), (E→D),(H→D) all have weights less than (G→H)

- Graph: G = (V,E) , Spanning tree: T = (V,E_T,R)
- Edge c … weight of c must be greater than or equal to weight of any edge in this cycle.
- Proof: Otherwise, let d be an edge on cycle with lower weight. Construct T' from T by removing c and adding d. T' is less weight than T, so T not minimal. Contradiction., so d can't exist.

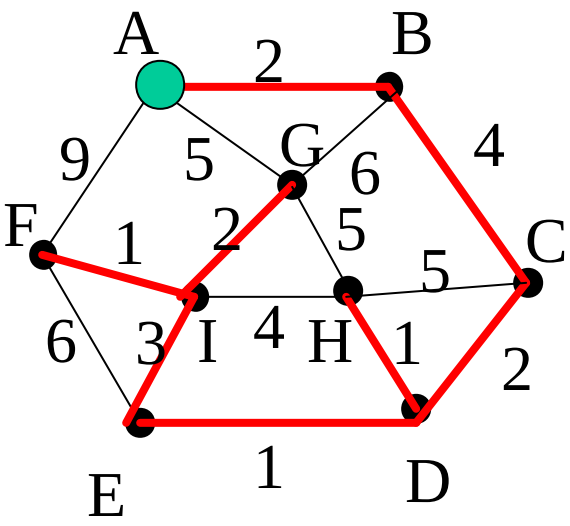
Building Minimal Spanning Trees

- Prim's algorithm: simple variation of Dijkstra's SSSP algorithm
  - Change Dijkstra's algorithm so the priority of bridge (f→n) is length(f,n) rather than minDistance(f) + length(f,n)
  - Intuition: Starts with any node. Keep adding smallest border edge to expand this component.
- Algorithm produces minimal spanning tree!

Prim's MST algorithm

```
Tree MST = empty tree;
Heap h = new Heap();
//any node can be the root of the MST
h.put((dummyRoot → anyNode), 0);
while (h is not empty)  {
    get minimum priority (= length) edge (t→f);
    if (f is not lifted) {
        add (t→f) to MST;//grow MST
        make f a lifted node;
        for each edge (f→n)
            if (n is not lifted)
                h.put((f→n), length(f,n));
    }
}
```

Steps of Prim's algorithm



[((dummy→A), 0)]

[] add (dummy→A) to MST
[((A→B),2), ((A→G),5),((A→F),9)]

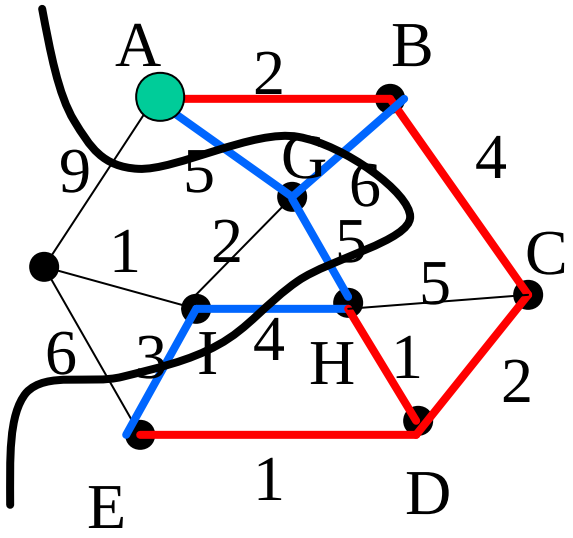
[((A→G),5),((A→F),9)] add (A→B) to MST
[((A→G),5),((A→F),9), (B→G),6),((B→C),4)]

[((A→G),5),((A→F),9),((B→G),6)]
add (B→C) to MST
[((A→G),5),((A→F),9),((B→G),6),((C,H),5),
((C,D), 2)]

……

Property of Prim's algorithm

- At each step of the algorithm, we have a spanning tree for "lifted" nodes.
- This spanning tree grows by one new node and edge at each iteration.

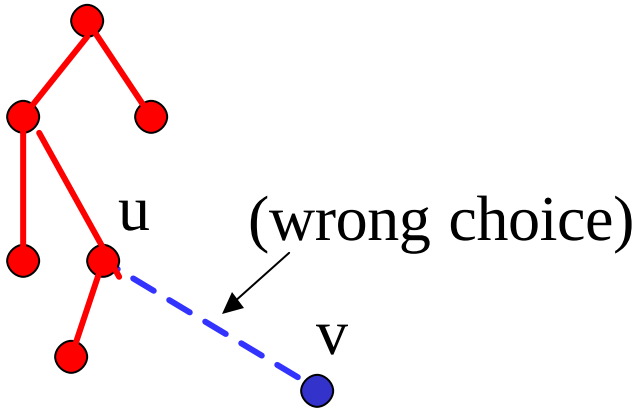


Proof of correctness (part 1)

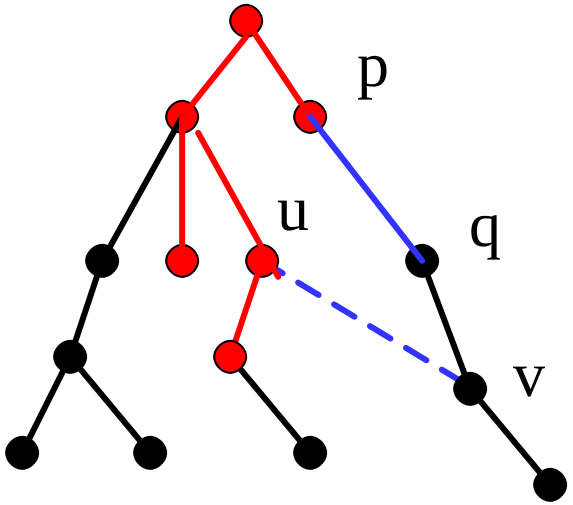
- Suppose the algorithm does not produce MST.
- Each iteration adds one new node and edge to tree.
- First iteration adds the root to tree, and at least that step is “correct”.
  - “Correct” means partial spanning tree built so far can be extended to an MST.
- Suppose first k steps were correct, and then algorithm made the wrong choice.
  - Partial spanning tree P built by first k steps can be extended to an MST M
  - Step (k+1) adds edge (u→v) to P, but resulting tree cannot be extended to an MST
  - Where to go from here?

Proof (contd.)

- Consider simple cycle formed by adding (u→v) to M. Let p be the lowest ancestor of v in M that is also in P, and let q be p's child in M that is also an ancestor of v. So (p→q) is a bridge edge at step (k+1) as is (u→v). Since our algorithm chose (u→v) at step (k+1), weight(u→v) is less than or equal to weight(p→q).
- From Property (3), weight of (u→v) must be greater than or equal to weight(p→q).



Partial spanning tree P



Minimal Spanning Tree M

Proof (contd.)

- Therefore, weight(p→q) = weight(u→v).
- This means that the tree obtained by taking M, deleting edge (p→q) and adding edge (u→v) is a minimal spanning tree as well, contradicting the assumption that there was no MST that contained the partial spanning tree obtained after step (k+1).
- Therefore (by induction!), our algorithm is correct.

Complexity of Prim's Algorithm

- Every edge is examined once and inserted into PQ when one of its two end points is first lifted.
- Every edge is examined again when its other end point is lifted.
- Number of insertions and deletions into PQ is $|E| + 1$
- Complexity = $O(|E|\log(|E|))$
- Same as Dijkstra's (of course)

Editorial notes

- Dijkstra's algorithm and Prim's algorithm are examples of greedy algorithms:
  - making optimal choice at each step of the algorithm gives globally optimal solution
- In most problems, greedy algorithms do not yield globally optimal solutions
  - (eg) TSP (Travelling Salesman Problem)
  - (eg) greedy algorithm for puzzle graph search: at each step, choose move that minimizes the number of tiles that are out of position
    - Problem: we can get stuck in "local" minima and never find the global solution