Minimum Spanning Trees

EECS 233

Course Evaluations

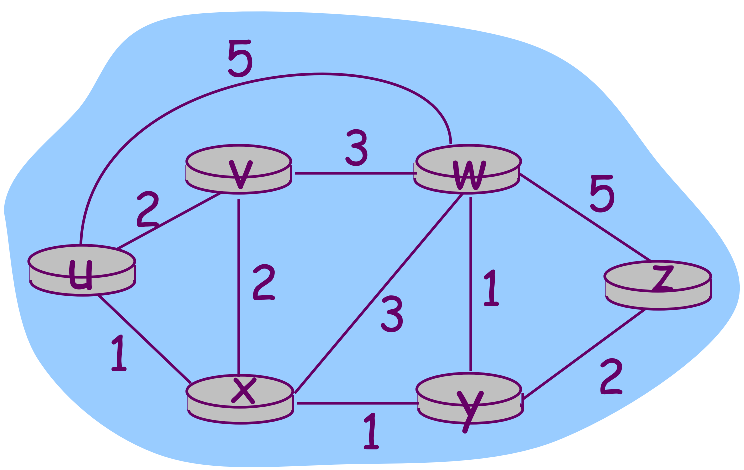
Course evaluations are
now open for student responses through
11:59 PM Wednesday, December 16

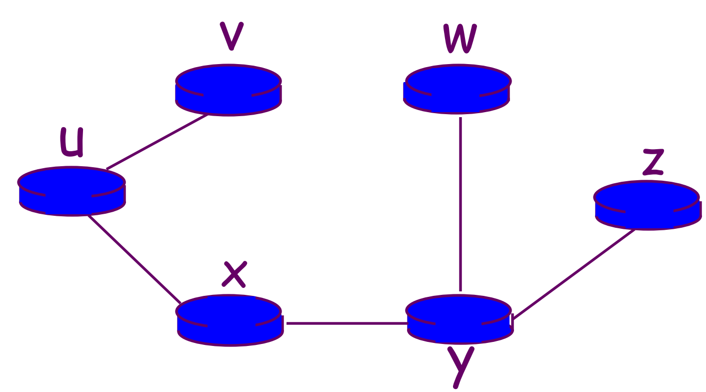
Students can find their evaluations
here:
https://webapps.case.edu/courseevals/

1% of the course
(whoever will upload the receipt/conformation
screenshot will get 1%)
But please do so by December 12th

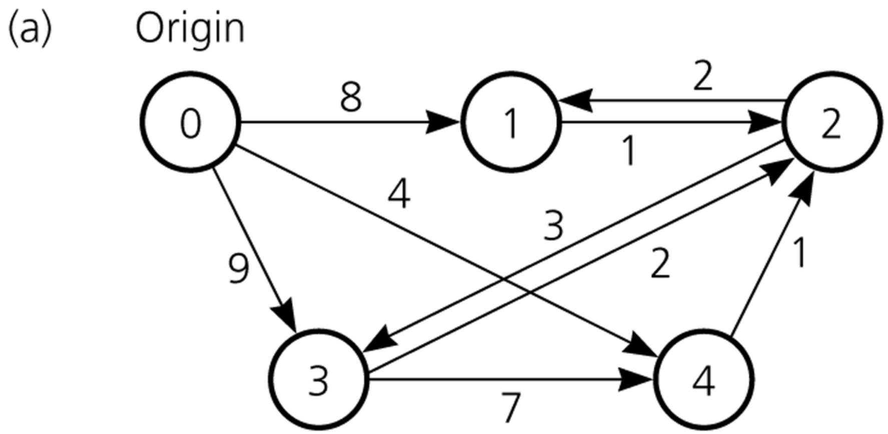
Dijkstra's algorithm: example

Step	N	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞





Shortest Path - Example



A Weighted Directed Graph

(b)

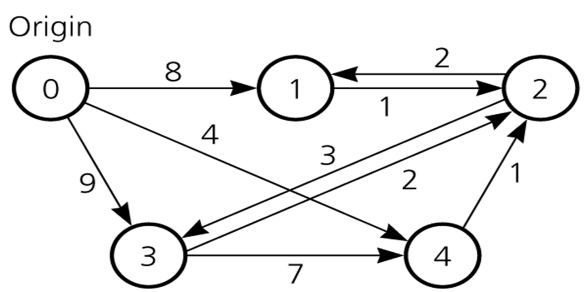
	0	1	2	3	4
0	∞	8	∞	9	4
1	∞	∞	1	∞	∞
2	∞	2	∞	3	∞
3	∞	∞	2	∞	7
4	∞	∞	1	∞	∞

Its Adjacency Matrix

Dijkstra’s Shortest-Path Algorithm – Trace

Step	v	vertexSet	weight [0]	weight [1]	weight [2]	weight [3]	weight [4]
1	–	0	0	8	∞	9	4
2	4	0, 4	0	8	5	9	4
3	2	0, 4, 2	0	7	5	8	4
4	1	0, 4, 2, 1	0	7	5	8	4
5	3	0, 4, 2, 1, 3	0	7	5	8	4

(a)

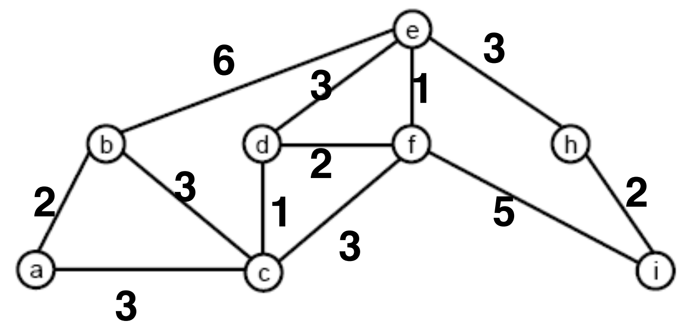


(b)

	0	1	2	3	4
0	∞	8	∞	9	4
1	∞	∞	1	∞	∞
2	∞	2	∞	3	∞
3	∞	∞	2	∞	7
4	∞	∞	1	∞	∞

Minimum Spanning Tree

- A minimum spanning tree (MST) of a connected (weighted) graph has the lowest total cost among all possible spanning trees.

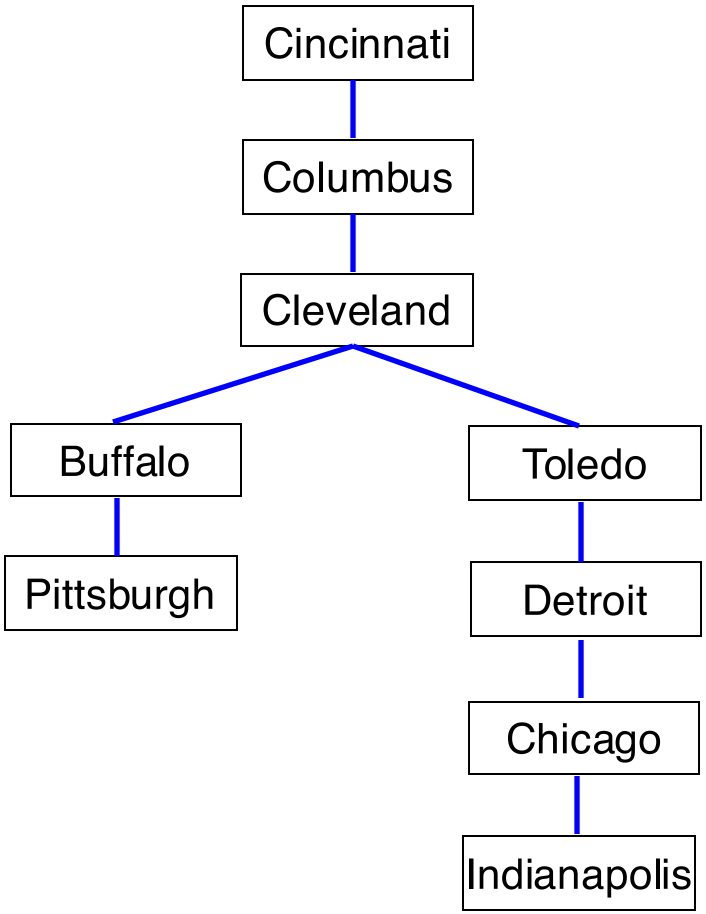


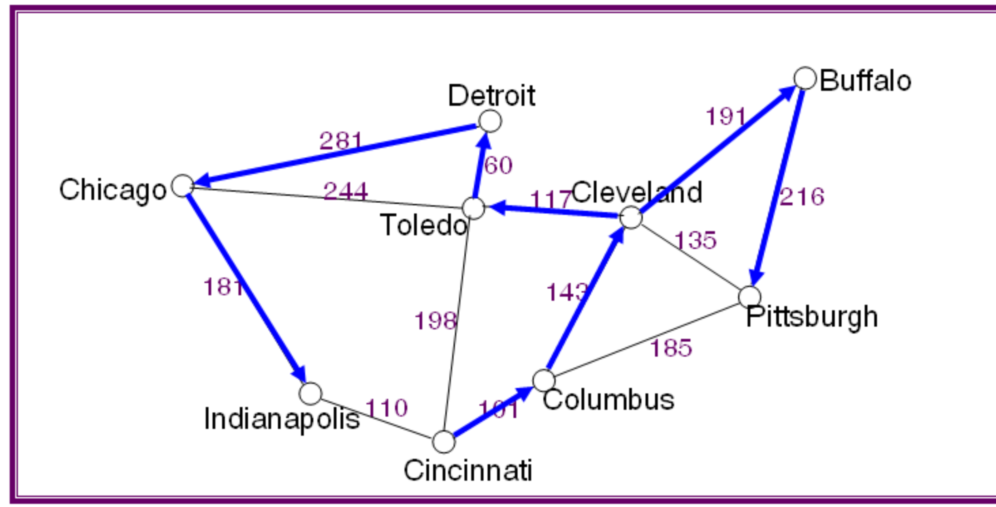
- MST may not be unique

- Applications: finding a MST could be used to
  - determine the shortest highway system to connect a set of cities
  - calculate the smallest length of cables needed to connect a network of computers
  - Group communication on the Internet (cost = hop distance to reduce bandwidth consumption)

Recall: Depth/Breadth-First Spanning Trees

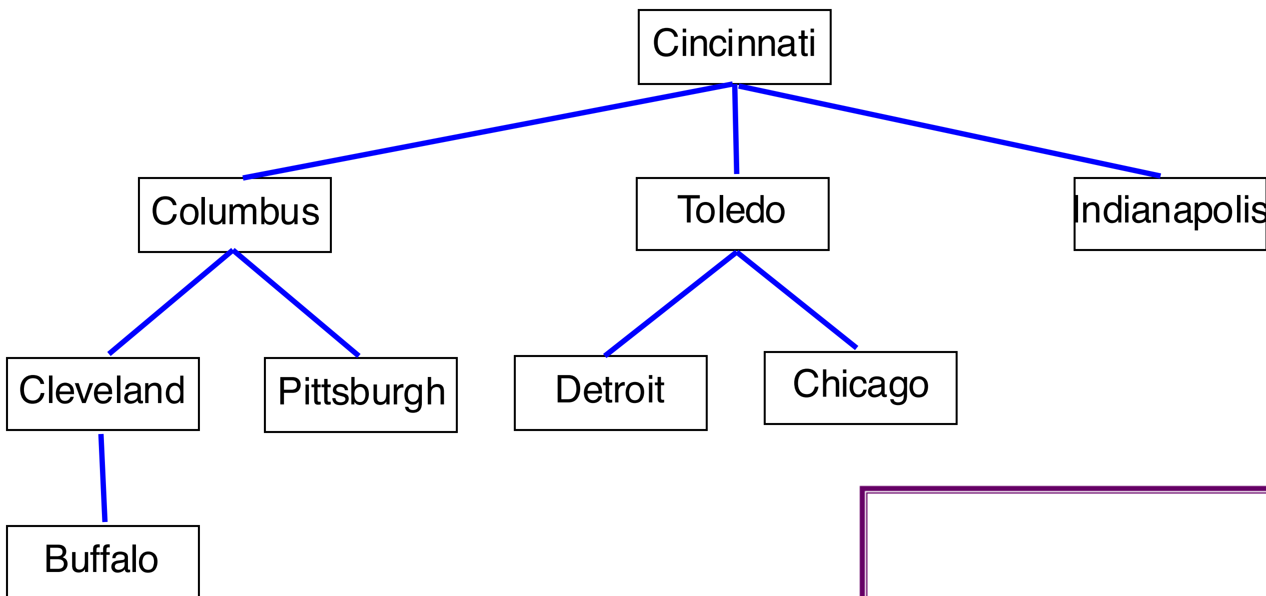
Depth-First traversal from Cincinnati: is it an MST?

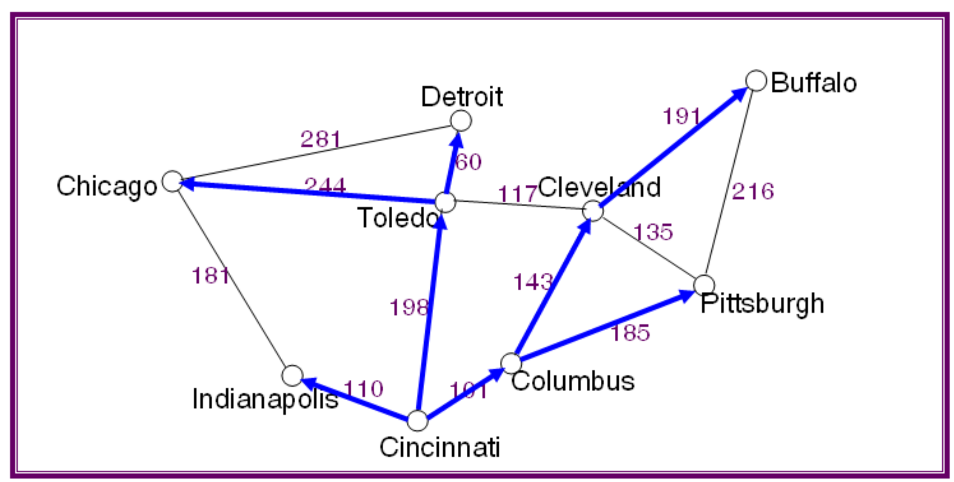




Recall: Depth/Breadth-First Spanning Trees

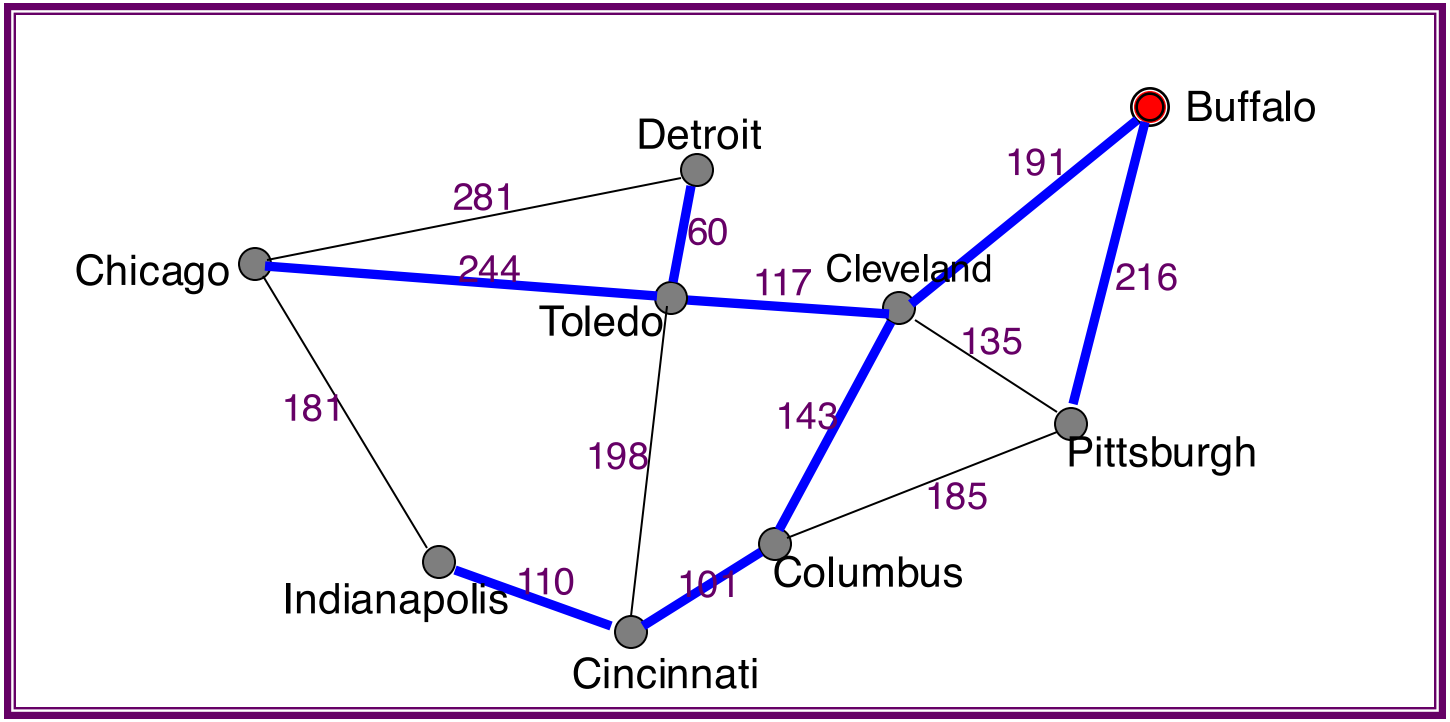
Breadth-First traversal from Cincinnati: is it an MST?





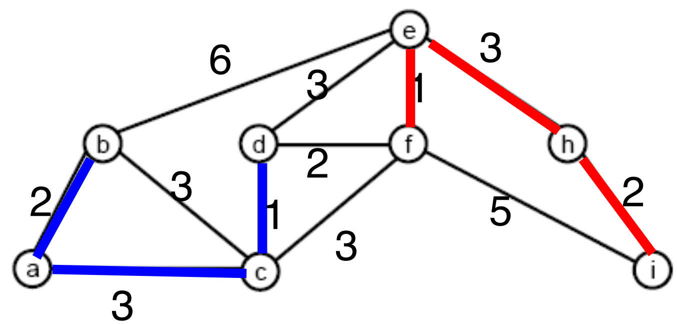
Recall: Shortest path spanning tree

- Shortest path tree from Buffalo – is it an MST?

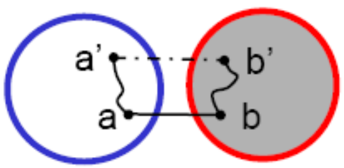


How to find A Minimum Spanning Tree?

- Key insight: if we divide the vertices into two disjoint subsets A and B, then the lowest-cost edge joining a vertex in A to a vertex in B – call it (a, b) – must be part of the MST.



- Proof by contradiction:
  - assume there is an MST (call it T) that doesn't include (a, b)
  - T must include a path from a to b, so it must include another edge (a', b') that connects subsets A and B with a path a to a' to b' to b
  - adding (a, b) to T introduces a cycle
  - removing (a', b') gives a spanning tree with lower cost, which contradicts the original assumption.



Prim’s Algorithm

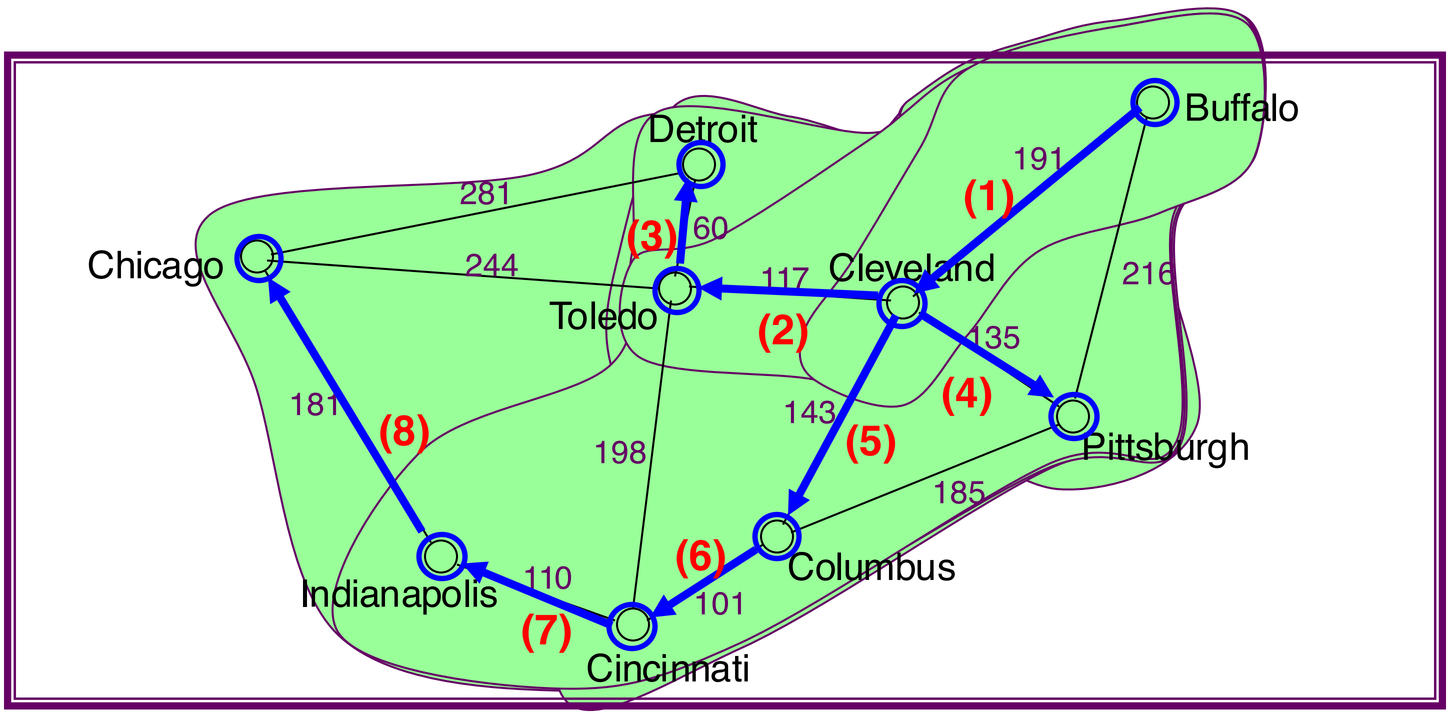
- Begin with the following subsets:
  - A = any one of the vertices
  - B = all of the other vertices

- Repeatedly select the lowest-cost edge (a, b) connecting a vertex in A to a vertex in B and do the following:
  - add (a, b) to the spanning tree
  - update the two sets: A = A U {b}, B = B – {b}

- Continue until A contains all vertices.

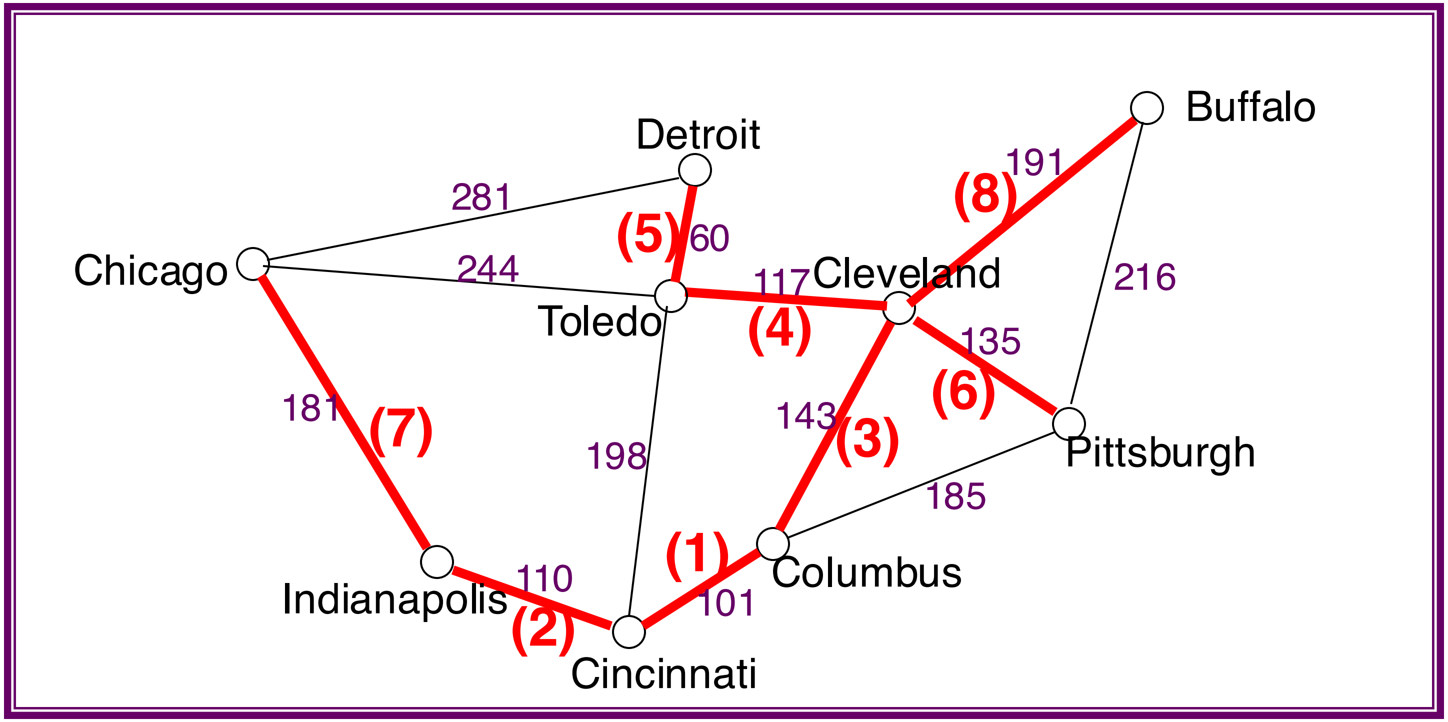
Prim’s Algorithm (starting from Buffalo)

- The algorithm will maintain the sets A and B
- Final result: the set of edges forming an MST



Prim's Algorithm (starting from Cincinnati)

- The same minimum spanning tree, but the order in which we include edges is different

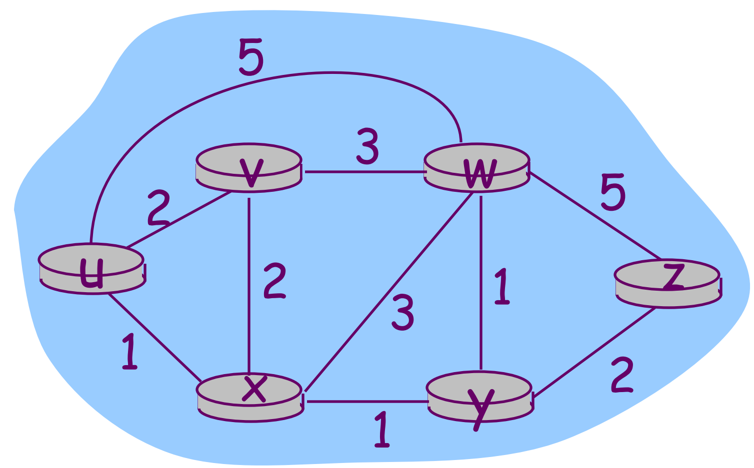


Reminder - Dijsktra's Algorithm

Notation:

c(x,y): link cost from node x to y;
∞ if not direct neighbors
D(v): current path cost estimate from source to dest. v
p(v): predecessor node along path from source to v
N: set of nodes whose least cost path definitively known
u: starting node

```
Initialization:
  N = {u}
  for all nodes v
    if v adjacent to u then
      D(v) = c(u,v)
      p(v) = u
    else D(v) = ∞

Loop
  find w not in N with smallest D(w)
  add w to N
  update D(v) for all v adjacent to w and not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
      Update p(v)
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes are in N
```



Prim's Algorithm

Notation:

c(x,y): link cost from node x to y;

∞ if not direct neighbors

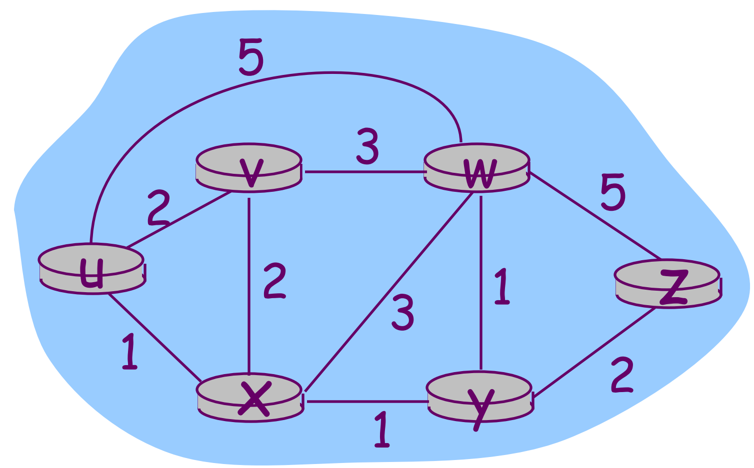
C(v): current estimate of cost-to-connect v to A

p(v): predecessor node along path from source to v

A: set of nodes added to MST

u: starting node

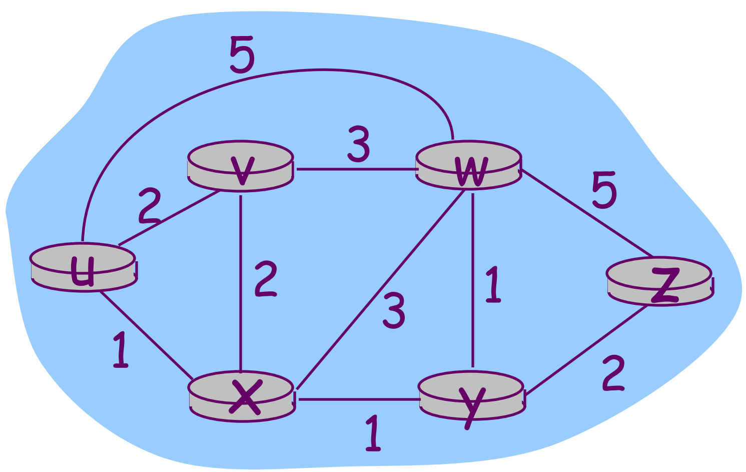
```
Initialization:
  A = {u}
    for all other nodes v
      if v adjacent to u then
          C(v) = c(u,v)
          p(v) = u
      else C(v) = ∞

 Loop
   find w not in A such that C(w) is a minimum
   add w to A
   update C(v) for all v adjacent to w and not in A :
           C(v) = min( C(v), c(w,v) )
           p(v) = w
 until all nodes are in A
```



Prim's vs. Dijkstra's Algorithms

```
Initialization:
 A = {u}
 for all nodes v
       if v adjacent to u then
        C(v) = c(u,v)
            p(v) = u
     else C(v) = ∞

Loop
  find w not in A such that C(w) is a minimum
   add w to A
   update C(v) for all v adjacent to w and not in A :
         C(v) = min( C(v), c(w,v) )
         p(v) = w
until all nodes are in A
```

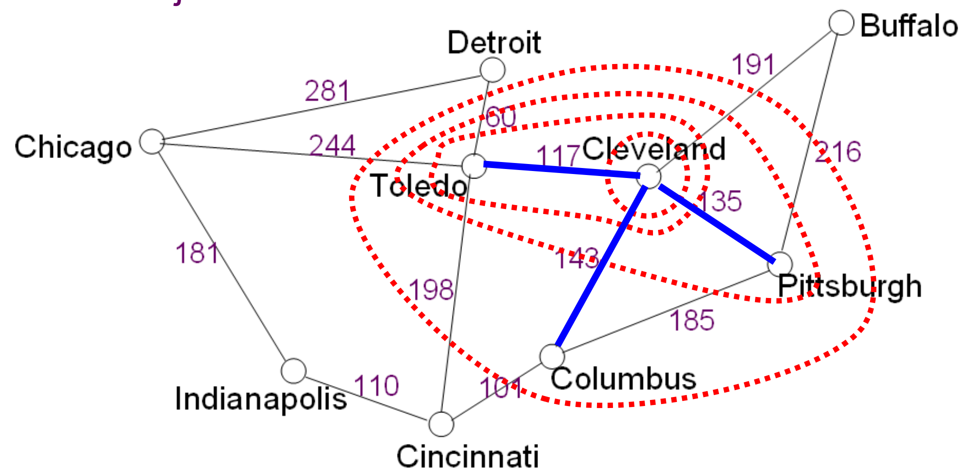
```
Initialization:
 N = {u}
 for all nodes v
       if v adjacent to u then
        D(v) = c(u,v)
            p(v) = u
     else D(v) = ∞

Loop
  find w not in N such that D(w) is a minimum
   add w to N
   update D(v) for all v adjacent to w and not in N :
         D(v) = min( D(v), D(w) + c(w,v) )
         p(v) = w
until all nodes are in N'
```



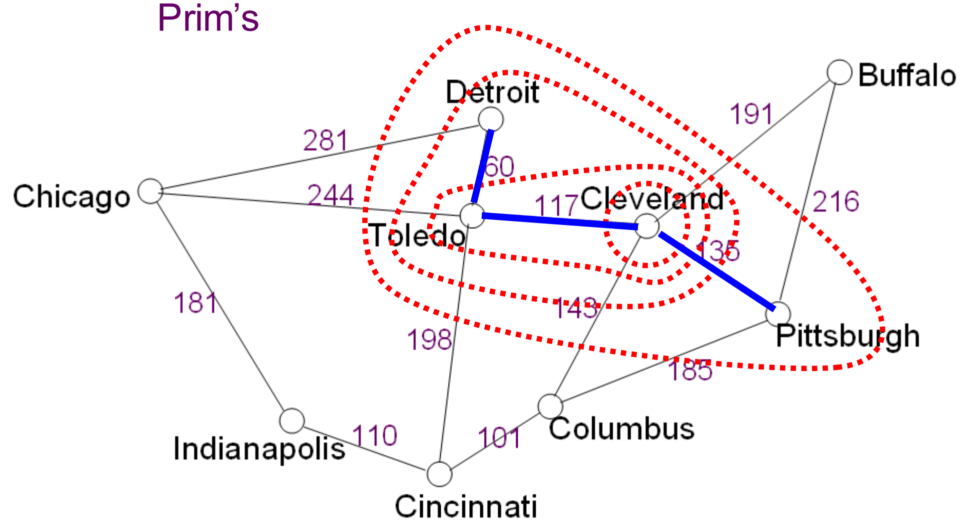
Intuitive Comparison of Dijkstra's vs. Prim's

- Both Dijkstra's and Prim's expand the search circle by one vertex at a time, until all vertices are included
- Dijkstra's includes the one with the shortest distance to **the source**
  - Shortest-path tree
- Prim's includes the one with the shortest distance to **any** of those already included
  - Minimum spanning tree

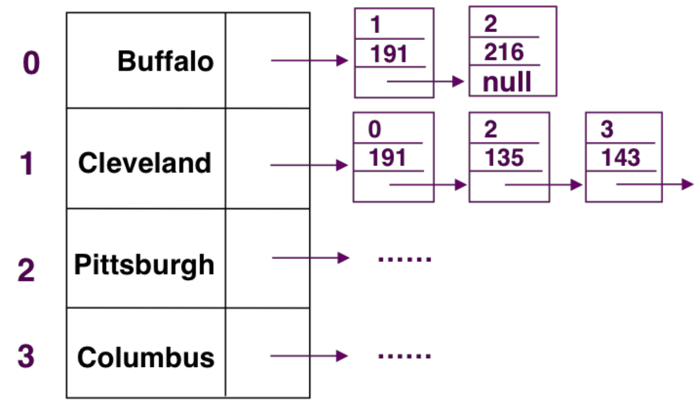
Dijkstra's



Prim's



Prim’s Algorithm: Complexity




```
Initialization:
  A = {u}
  for all nodes v                      // V iterations
      if v adjacent to u then          // Check of the total
          C(v) = c(u,v)                //   of up to E edges
          p(v) = u
      else C(v) = ∞
                                       // O(V) to build the heap
Loop                                   // V iterations
   find w not in A such that C(w) is a // logV to reorg the heap after w’s removal
   minimum
    add w to A
    update C(v) for all v adjacent to w and   // A step per every link (E total); logV
   not in A :                                 // per step to reorganize the heap
        C(v) = min( C(v), c(w,v) )
          p(v) = w
 until all nodes are in N'
```

$\boxed{\text{Overall complexity is } O(V + E + V\log V + E\log V) = O((E+V)\log V) = O(E\log V)}$