# Graphs

## Algorithms for Weighted Graphs

Brad Miller  David Ranum[1]

[1]Department of Computer Science
Luther College

12/19/2005

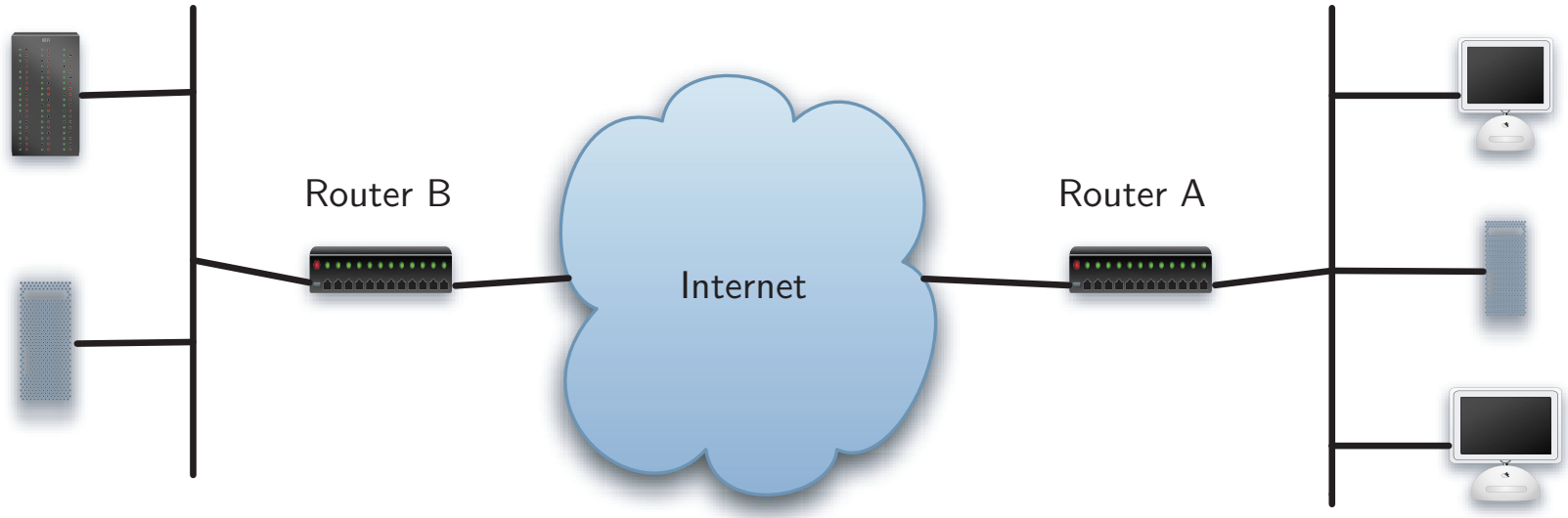
# Outline

- Shortest Path Problems
- Spanning Trees

# Outline

- Shortest Path Problems
- Spanning Trees

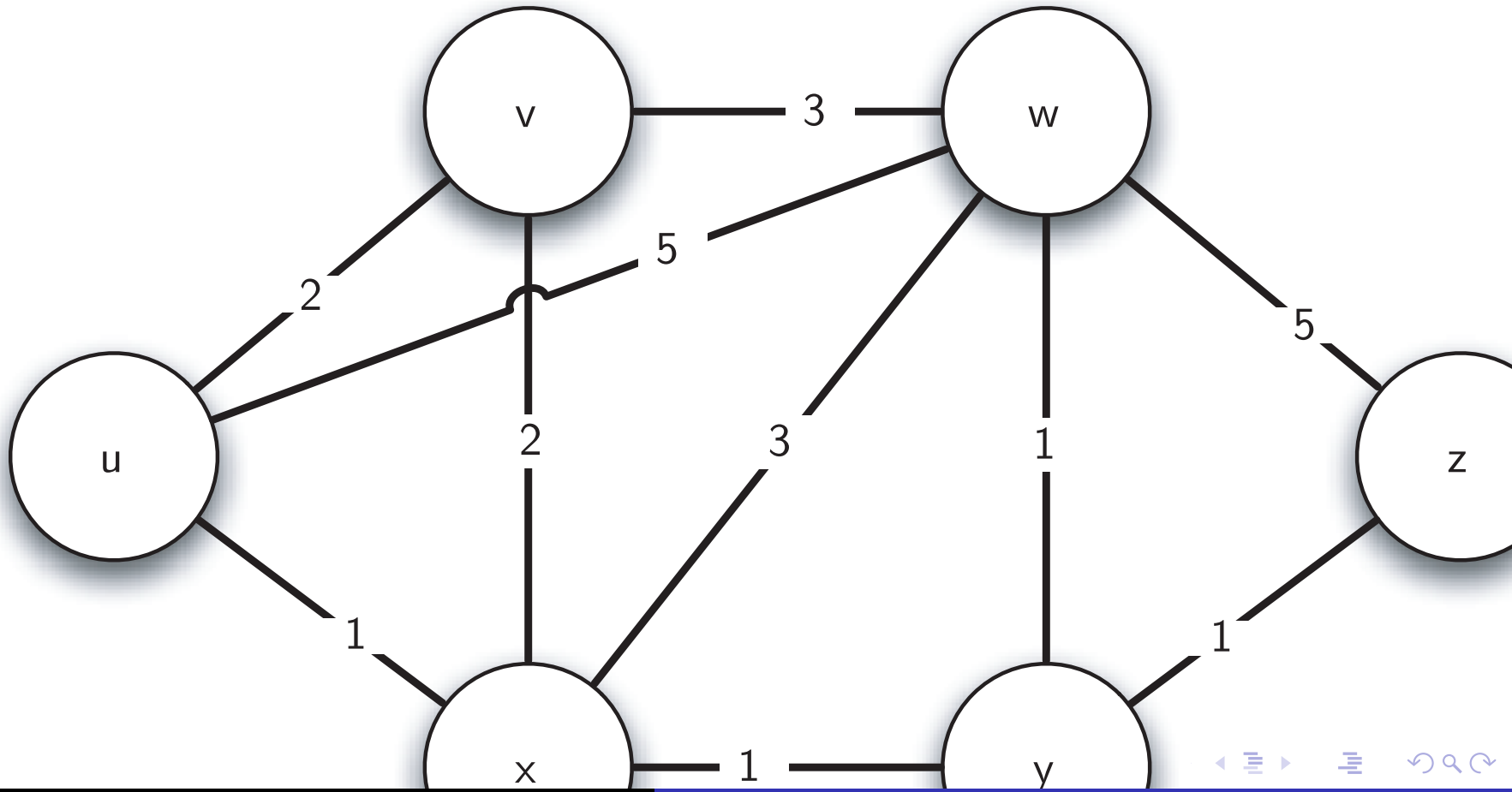
# Overview of Connectivity in the Internet



# Routers from One Host to the Next over the Internet

```
 1  192.203.196.1
 2  hilda.luther.edu (216.159.75.1)
 3  ICN-Luther-Ether.icn.state.ia.us (207.165.237.1
 4  ICN-ISP-1.icn.state.ia.us (209.56.255.1)
 5  p3-0.hsa1.chi1.bbnplanet.net (4.24.202.13)
 6  ae-1-54.bbr2.Chicago1.Level3.net (4.68.101.97)
 7  so-3-0-0.mpls2.Minneapolis1.Level3.net (64.159.
 8  ge-3-0.hsa2.Minneapolis1.Level3.net (4.68.112.1
 9  p1-0.minnesota.bbnplanet.net (4.24.226.74)
10  TelecomB-BR-01-V4002.ggnet.umn.edu (192.42.152.
11  TelecomB-BN-01-Vlan-3000.ggnet.umn.edu (128.101
12  TelecomB-CN-01-Vlan-710.ggnet.umn.edu (128.101.
13  baldrick.cs.umn.edu (128.101.80.129)(N!)  88.63
```

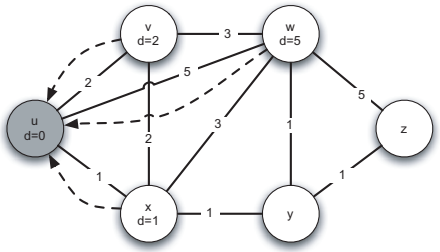
# Connections and Weights Between Routers in the Internet



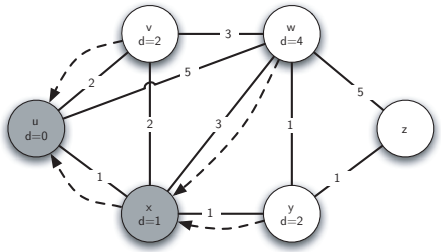
# Dijkstra's Algorithm I

```
1  def dijkstra(G,start):
2      PQ = PriorityQueue()
3      start.setDistance(0)
4      PQ.buildHeap([(v.getDistance(),v) for v in G])
5      while not PQ.isEmpty():
6          w = PQ.delMin()
7          for v in w.getAdj():
8              newDist = w.getDistance() + w.getCost(v)
9              if newDist < v.getDistance():
10                 v.setDistance( newDist )
11                 v.setPred(w)
12                 PQ.decreaseKey(v,newDist)
```

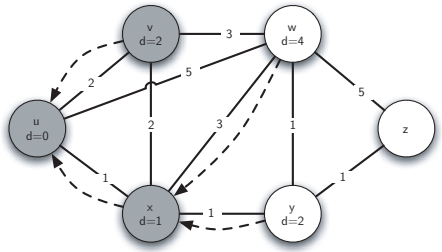
# Tracing Dijkstra's Algorithm



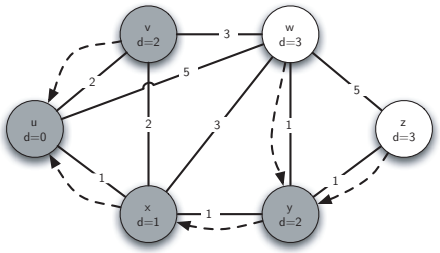

(a)



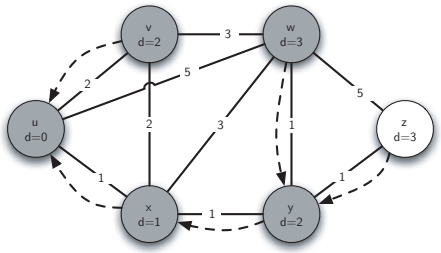

(b)



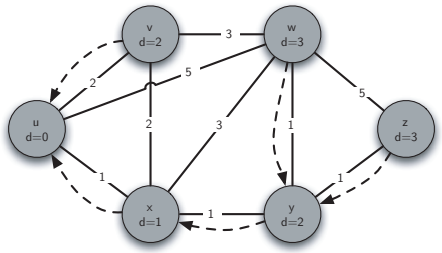

(c)




(d)




(e)

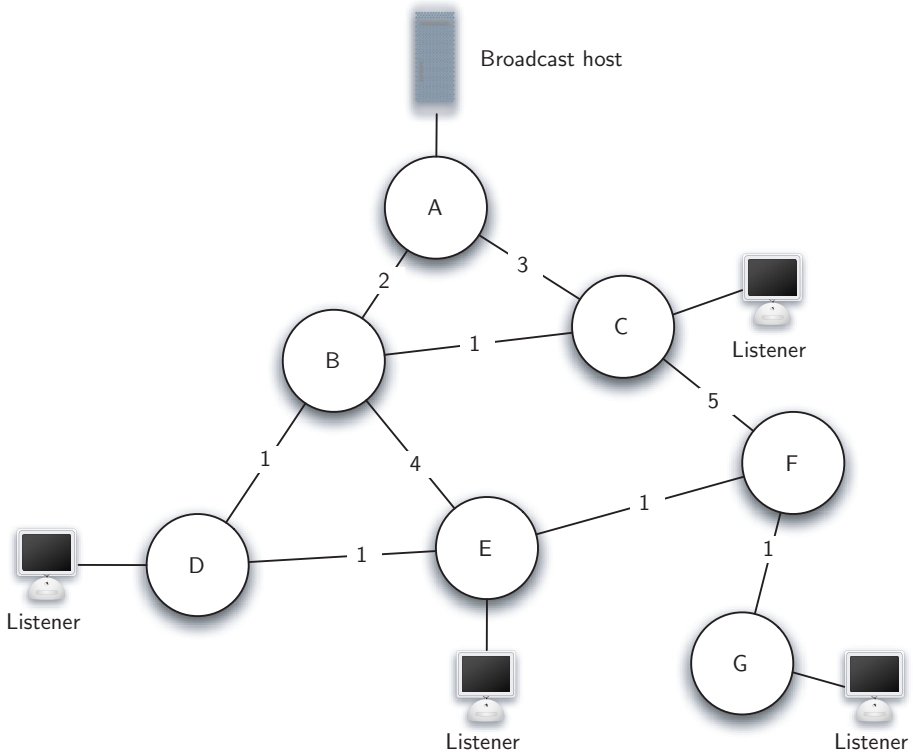



(f)

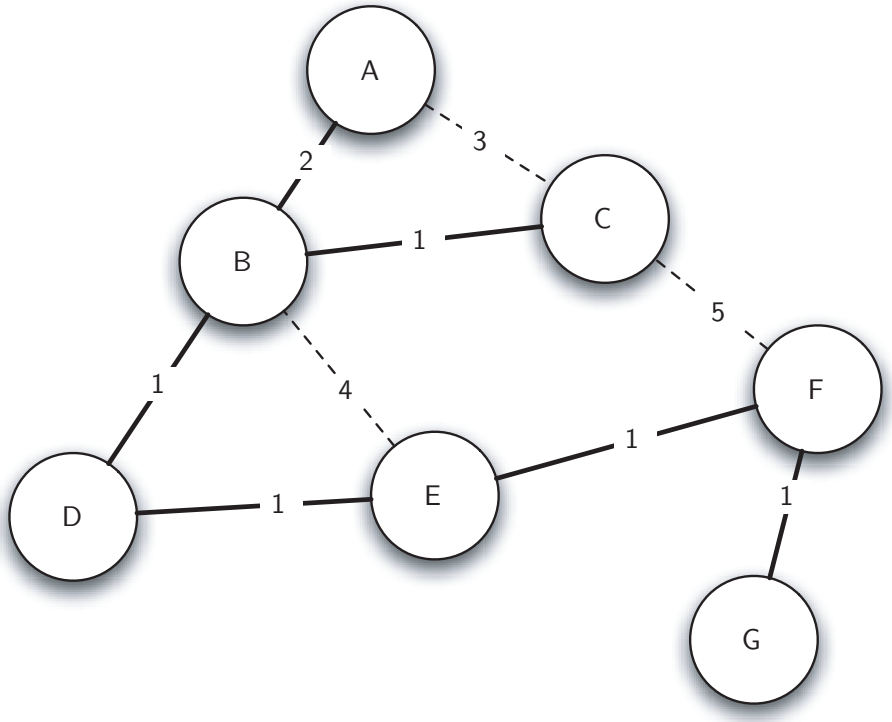
# Outline

- Shortest Path Problems
- Spanning Trees

# The Broadcast Problem



# Minimum Spanning Tree for the Broadcast Graph

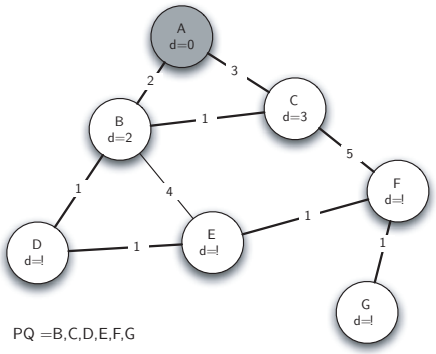


1. While $T$ is not yet a spanning tree
   1. Find an edge that is safe to add to the tree
   2. Add the new edge to $T$

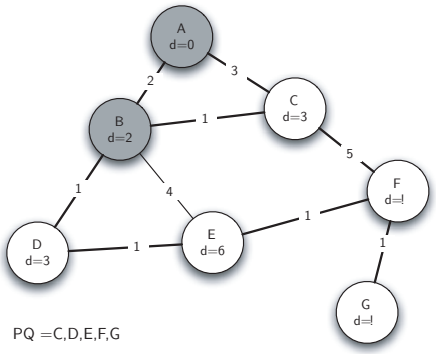
# Prim's Minimum Spanning Tree Algorithm I

```
1  def prim(G,start):
2      PQ = PriorityQueue()
3      for v in G:
4          v.setDistance(sys.maxint)
5          v.setPred(None)
6      start.setDistance(0)
7      PQ.buildHeap([(v.getDistance,v) for v in G])
8      while not PQ.isEmpty():
9          u = PQ.delMin()
10         for v in u.getAdj():
11             if v in PQ and u.cost(v) < v.getDistance():
12                 v.setPred(u)
13                 v.setDistance(u.cost(v))
14                 PQ.decreaseKey((u.cost(v)))
```

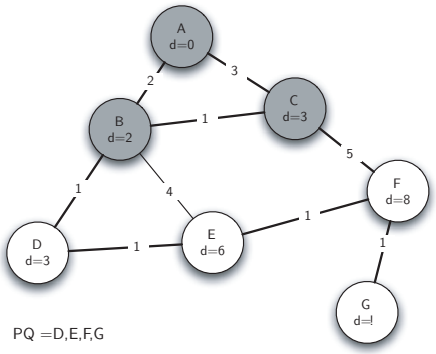
# Tracing Prim's Algorithm



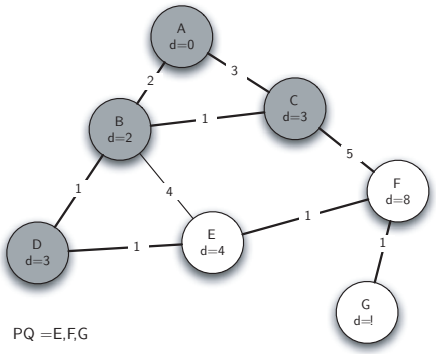

(a)



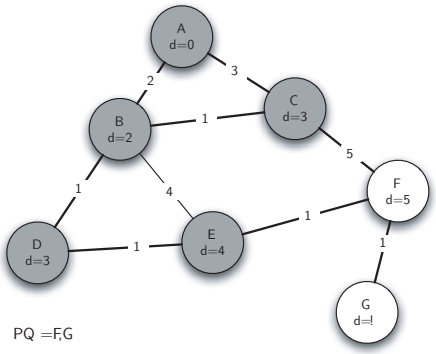

(b)



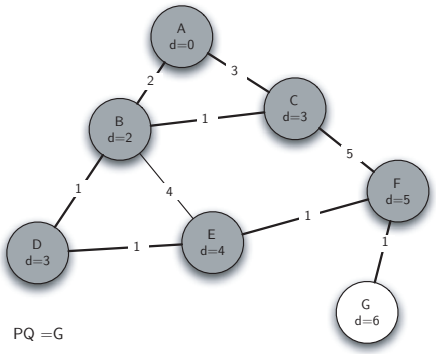

(c)




(d)




(e)




(f)