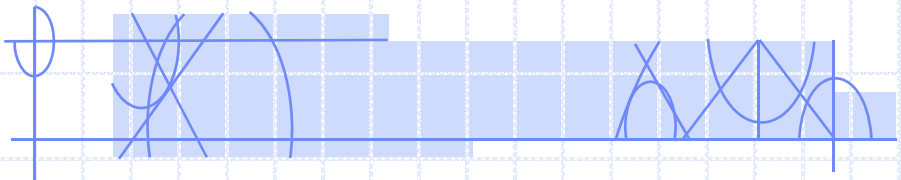
COMP 3760

Algorithm Analysis and Design

Lesson 20: Dijkstra's Algorithm (intro)



Rob Neilson

rneilson@bcit.ca

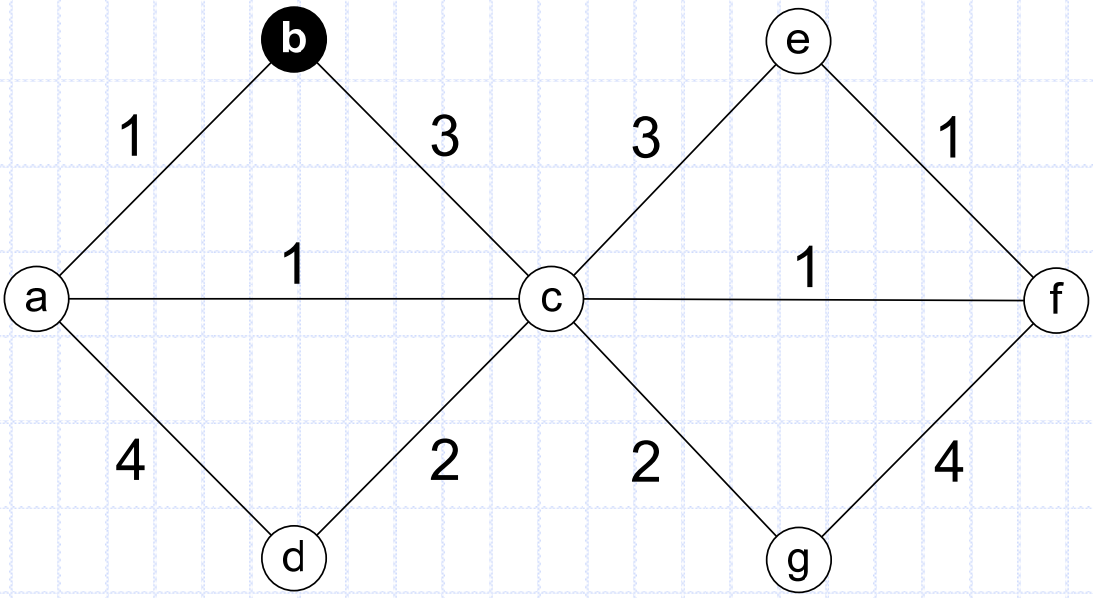
Reading

- Reading: Chapter 9.3

Shortest Path Problems

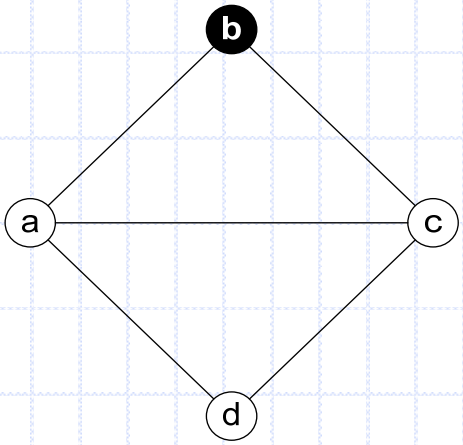
Problem: Single-Source Shortest Path

- find the shortest path from one source vertex v to every other vertex in the graph
  - "source" means "starting vertex"

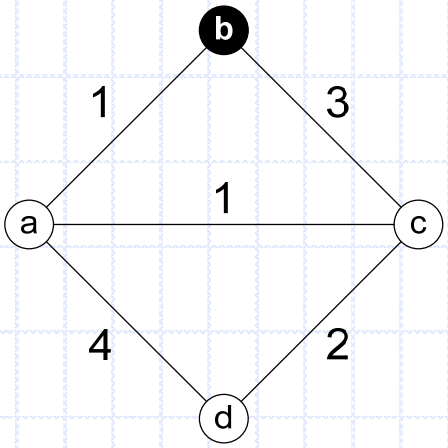


What about BFS?

- we know how to do this for an unweighted graph
  - BFS



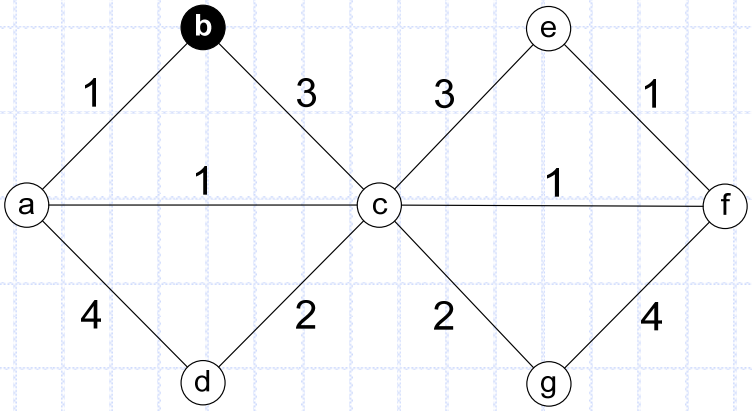
- but BFS doesn't work for weighted graphs, consider:



- *the algorithm to find shortest paths in weighted graphs needs to consider the weight on the edge before including it in the solution*

Dijkstra's Algorithm

- Greedy Algorithm
  - builds a tree of shortest paths rooted at the starting vertex
  - it is greedy because it adds the closest vertex, then the next closest, and so on (until all vertices have been added)



  - returns the *shortest distance to each vertex* in array **d[]**
  - returns the *parent of each vertex* in array **prev[]**

```
1. Initialise d and prev
2. Add all vertices to a PQ with distance
   from source as the key
3. While there are still vertices in PQ
4.     Get next vertex u from the PQ
5.     For each vertex v adjacent to u
6.         If v is still in PQ, relax v
```

Relaxation

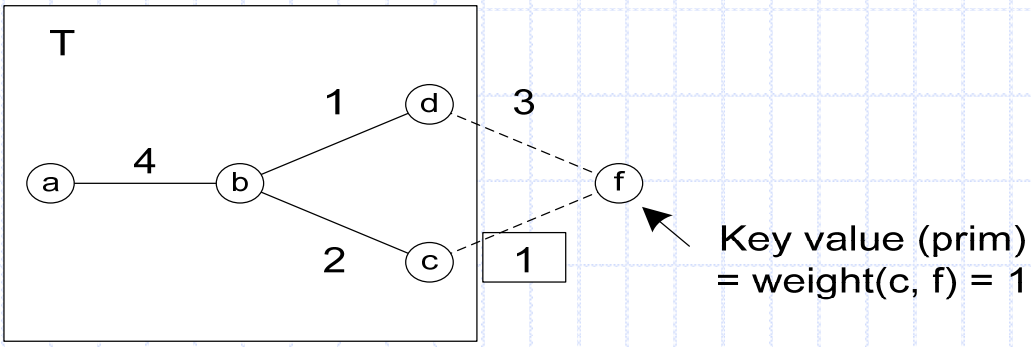
- Dijkstra always refers to “relaxing” a vertex
- this means “update the best known shortest path to v, and re-insert in the PQ”

- the pseudocode for relaxation should read:

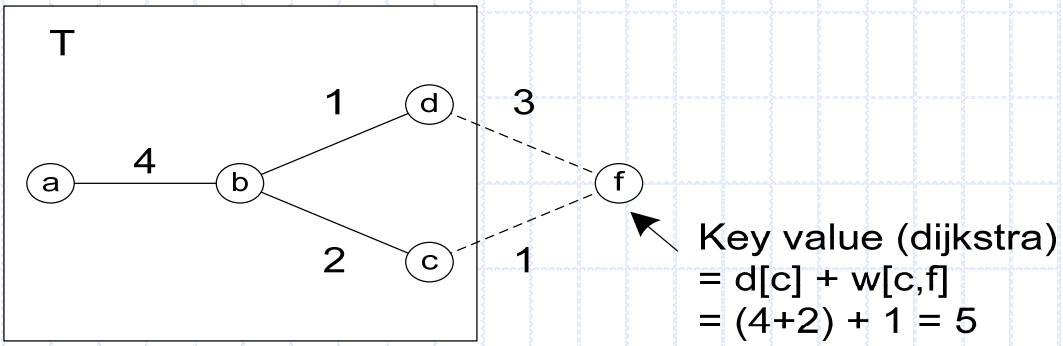
```
if d[u] + w(u,v) < d[v]
    d[v] ← d[u] + w(u,v)
    prev[v] ← u
    PQ.updateKey(d[v], v)
```

Similarity to Prim

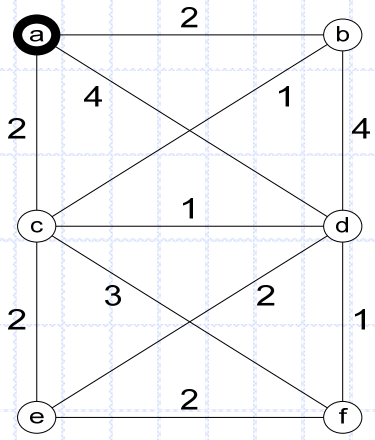
- algorithm is similar to Prim's algo
  - needs to select the minimum priority edge from the set of edges adjacent to the tree that has been built so far
  - in Prim's algo the "priority" of an edge *(u, v)* is defined by the weight of the edge

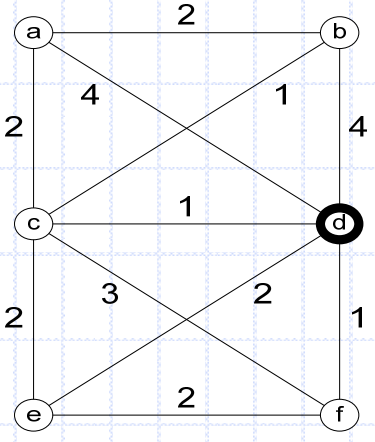


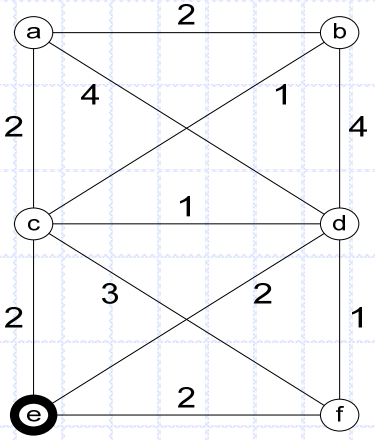
  - in Dijkstra the "priority" is given by the weight of the edge *(u, v)* plus the distance from the start to the parent of *v*



Some Dijkstra Examples





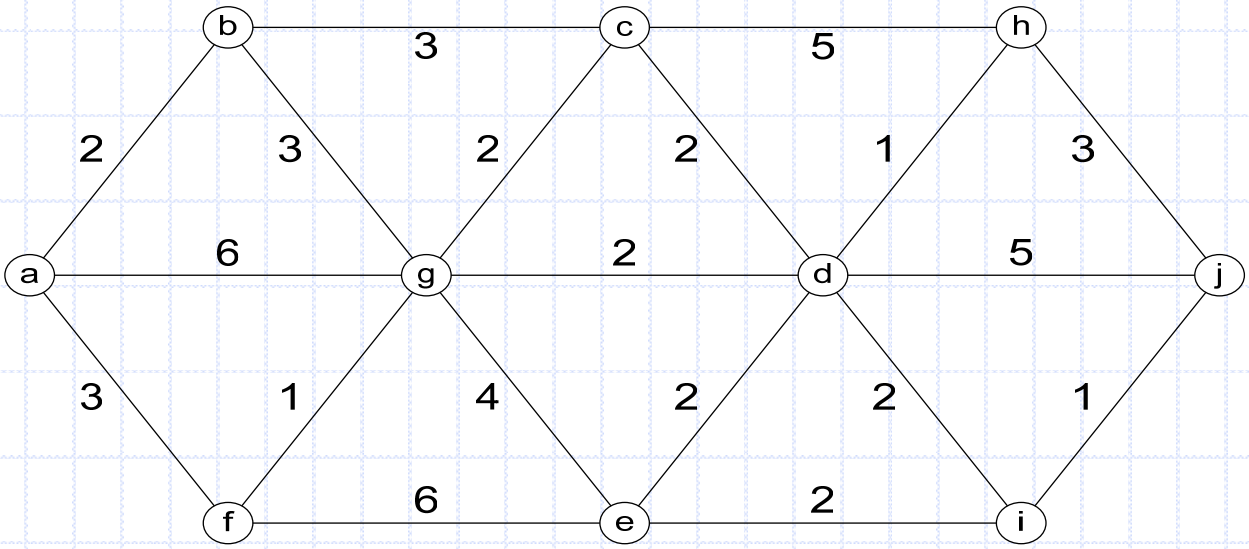


Dijkstra's Algorithm (more formally)

```
algo: Dijkstra(G, w, v_s)

    T ← empty set
    for each vertex v in V
         d[v] ← infinity
         prev[v] ← null

    d[V_s] ← 0

    for each vertex v in V
         PQ.add(d[v], v])

    while PQ is not empty
         u ← PQ.Extract_Min()
         T ← T union {u}
         for each v in V-T that is adjacent to u
                 if d[u] + w(u,v) < d[v]
                         d[v] ← d[u] + w(u,v)
                         prev[v] ← u
                         PQ.updateKey(d[v], v)
    return {d[], prev[]}
```

Note: if you just want shortest path to one node, stop when that vertex is retrieved in the Extract_Min() step

Dijkstra: example 1



How do we actually get the path?

- So the algorithm can return any of the following: (depending on implementation):
  1. a map or array giving the distance from start vertex s to any other vertex v
     - this is useful for some problems (where we need distances), but it doesn't give us the path

  or ...

  2. a map or array giving "previous vertex"
     - this is what we need
     - we can iterate over this structure and reconstruct the path as follows:

Recursively:

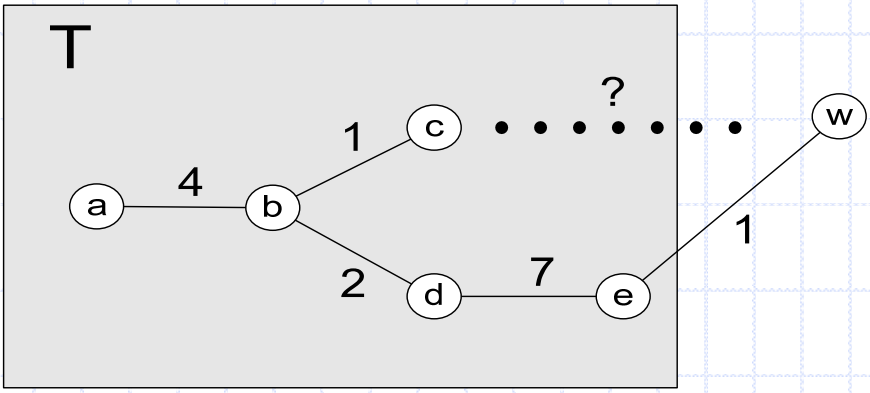
```
printPathToVertex(Vertex v)
{
   if prev(v) is not null {
      printPathToVertex( prev(v) )
   }
   print v
   print " "
}
```

Iteratively:

```
printPathToVertex(Vertex v)
{
   String Path = v
   while ( prev[v] != null ) {
       Path = v + " " + Path   // concatenate
       v = prev[v]
   }
}
```

Dijkstra: why does it work?

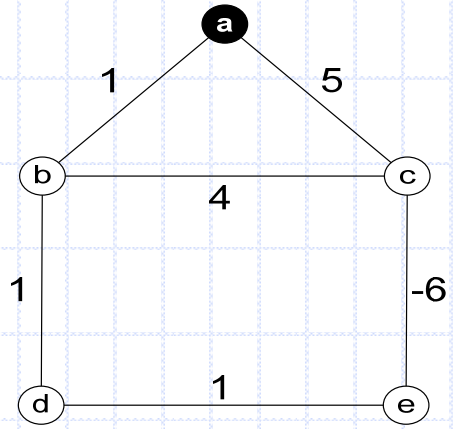
- The only way it couldn't work is if we added a vertex into T when there was a shorter path which we had not yet discovered.



- This is not possible. Consider the above graph.
  - vertices in the shaded box have been added
  - suppose "?" is such that an "undiscovered" path through w has shorter distance than 7
  - for this to be true the total weight on edge (c,w) must be less than 6
  - assume the weight on edge c,w is small – say 5
    - this means vertex w must be added to the tree T before vertex e
    - edge (d,e) would never get added
  - therefore any path (u-v) that has been added to T is guaranteed to be the shortest that exists from u-v

Dijkstra: negative weight edges?

- negative weight edges do not work
- if we added a new edge to T, and it had a negative weight, then there could exist a shorter path (through this new vertex) to vertices already in T
  - this violates the underlying assumption that the path to vertex v is the shortest known when we add v to T



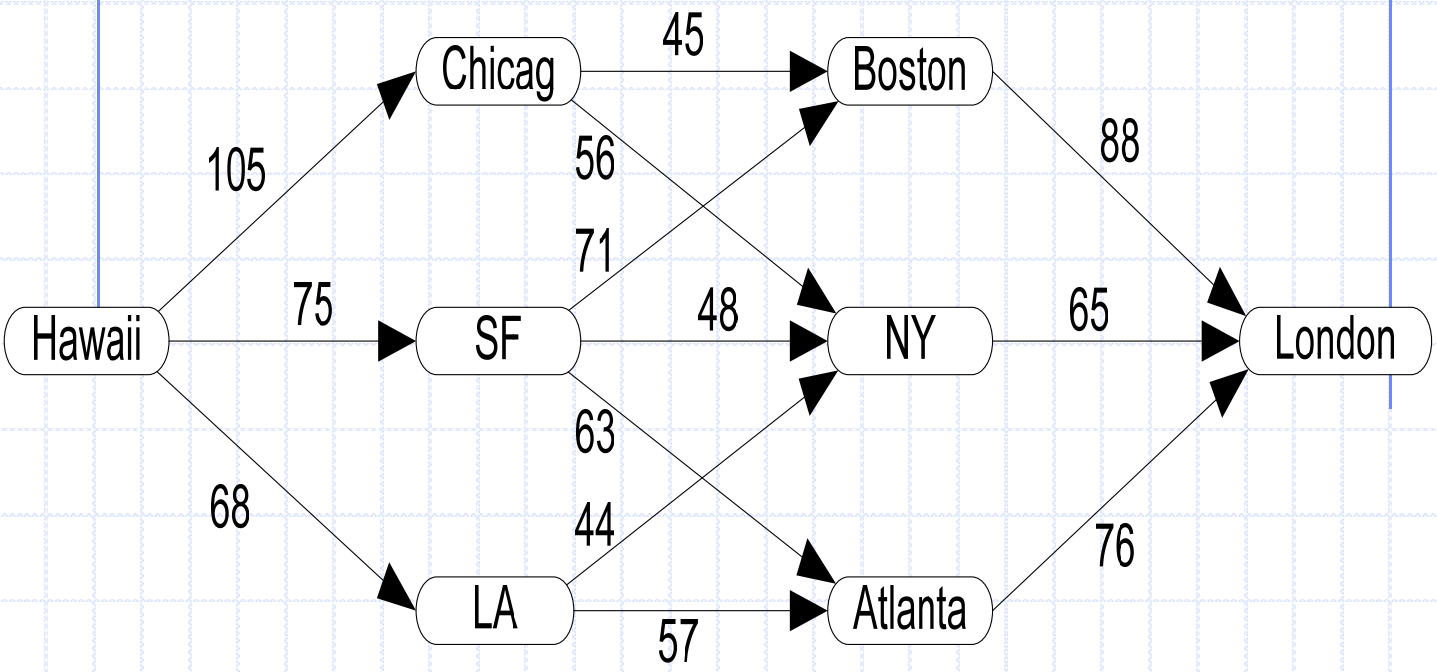
Example shortest path problem

- Whole pineapples are served in a restaurant in London. To ensure freshness, the pineapples are purchased in Hawaii and air freighted from Honolulu to Heathrow in London.
- There are various airline routes that the shipments can take, but each possible route has a different shipping cost.
- Which route will result in the lowest shipping cost?

Input: (start, destination, cost)

```
Honolulu Chicago 105
Honolulu SanFran 75
Honolulu LA 68
Chicago Boston 45
Chicago NewYork 56
SanFran Boston 71
SanFran NewYork 48
SanFran Atlanta 63
LA NewYork 44
LA Atlanta 57
Boston London 88
NewYork London 65
Atlanta London 76
```

Build a model …



- now we just apply Dijkstra to find the shortest route from Honolulu to London

Note: sample problem from http://www.csc.fi

The End