Shortest Path Algorithms

Kuan-Yu Chen (陳冠宇)

2020/12/16 @ TR-212, NTUST

Review

- They each use a specific rule to determine a safe edge in line 3 of GENERIC-MST

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

  - In Prim's algorithm
    - The set A forms a single tree
    - The safe edge added to A is always a least-weight edge **connecting the tree to a vertex not in the tree**
  - In Kruskal's algorithm
    - The set A is a forest whose vertices are all those of the given graph
    - The safe edge added to A is always a least-weight edge in the graph that **connects two distinct components (trees)**

Shortest-paths Problem

- In a ***shortest-paths problem***
  - Given a weighted directed graph $G = (V, E)$
    - The weight function $w: E \rightarrow \mathbb{R}$ mapping edges to real-valued weights
  - The ***weight*** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

  - We define the ***shortest-path weight*** $\delta(u, v)$ from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p): u \overset{p}{\leadsto} v\}, & if\ there\ is\ a\ path\ from\ u\ to\ v \\ \infty & , otherwise \end{cases}$$

  - We shall focus on the ***single-source shortest-paths problem***
    - Given a graph $G = (V, E)$, we want to find a shortest path from a given ***source*** vertex $s \in V$ to each vertex $v \in V$

Shortest Path Algorithms

- The representative algorithms, which are used to calculate the shortest path, are:
  - Dijkstra's algorithm
  - Bellman-Ford algorithm

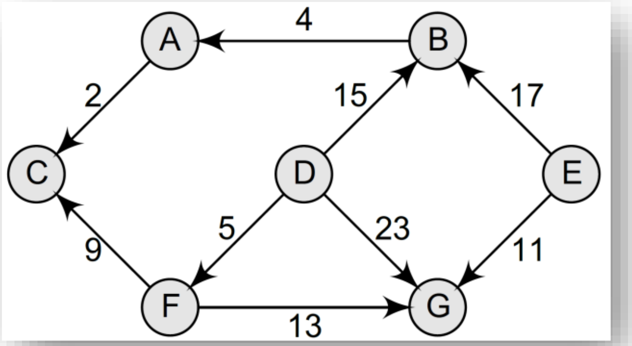
Dijkstra's Algorithm.

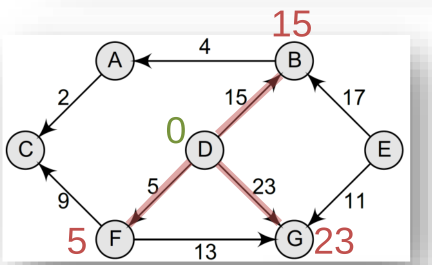
- Dijkstra's algorithm, given by a Dutch scientist Edsger Dijkstra in 1959, is used to find the shortest path tree
  - Given a graph *G* and a source node *A*, the algorithm is used to find the shortest path (one having the lowest cost) between *A* (source node) and every other node

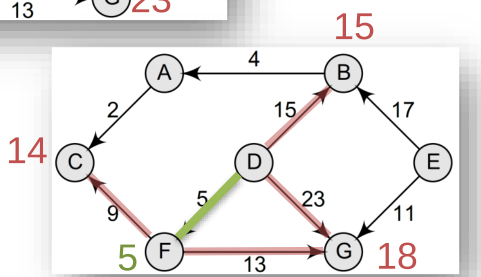
```
1. Select the source node also called the initial node
2. Define an empty set N that will be used to hold nodes to which a shortest
   path has been found.
3. Label the initial node with 0, and insert it into N.
4. Repeat Steps 5 to 7 until the destination node is in N or there are no more
   labelled nodes in N.
5. Consider each node that is not in N and is connected by an edge from
   the newly inserted node.
6. (a) If the node that is not in N has no label then SET the label of the
       node = the label of the newly inserted node + the length of the edge.
   (b) Else if the node that is not in N was already labelled, then SET its new
       label = minimum (label of newly inserted vertex + length of edge, old
       label)
7. Pick a node not in N that has the smallest label assigned to it and add it
   to N.
```

Dijkstra's Algorithm..

- Given a graph G, please take D as the initial (source) node, and execute the Dijkstra's algorithm on the graph
  - Step 1:
    - Set the label of D = 0 and N = {D}
  - Step 2:
    - Label of B = 15, G = 23, and F = 5
    - Therefore, N = {D, F}
  - Step 3:
    - B = 15
    - Re-label G = minimum(5 + 13, 23) = 18
    - Label C = 14 (5 + 9)
    - Therefore, N = {D,F, C}

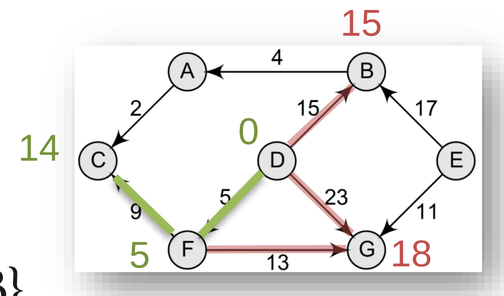


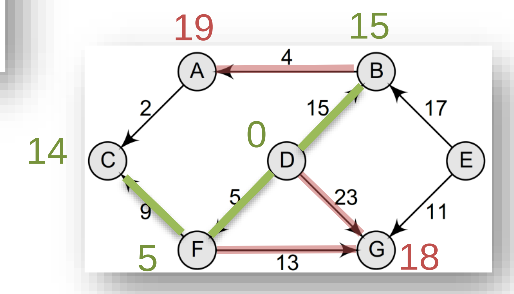




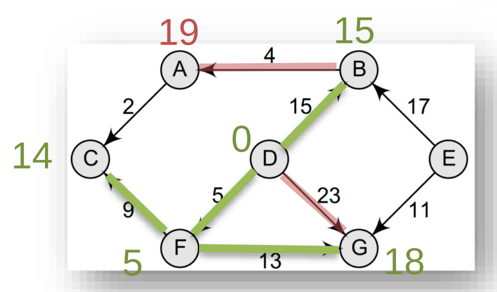
Dijkstra's Algorithm…

- Step 4:
  - B = 15
  - G = 18
  - Therefore, N = {D, F, C, B}

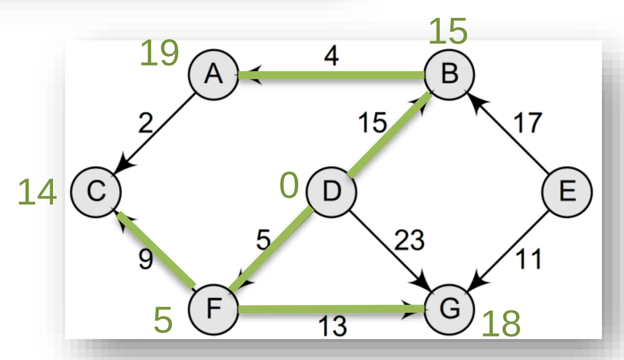




- Step 5:
  - G = 18
  - Label A = 19 (15 + 4)
  - Therefore, N = {D, F, C, B, G}

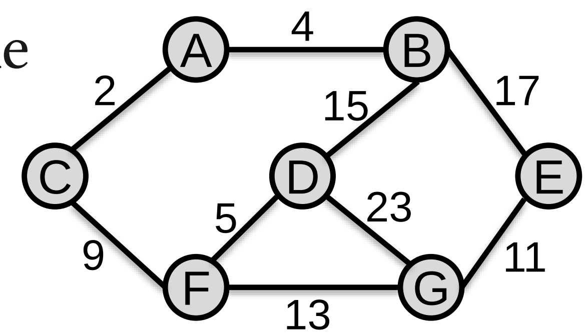


- Step 6:
  - A = 19
  - Therefore, N = {D, F, C, B, G, A}

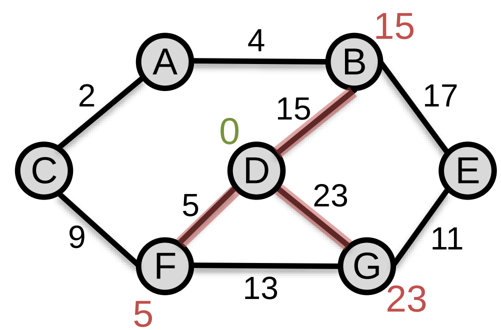


Dijkstra's Algorithm….

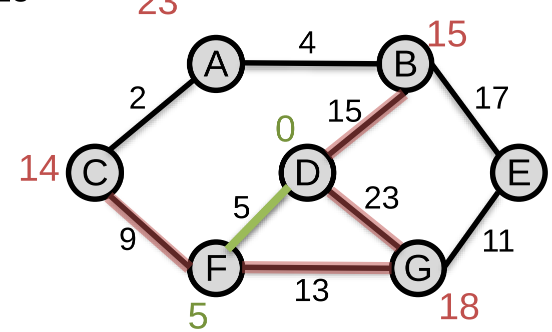
- Given a **undirected** graph G, please take D as the initial node, and execute the Dijkstra's algorithm on it



  – Step 1:
    - Set the label of D = 0 and N = {D}



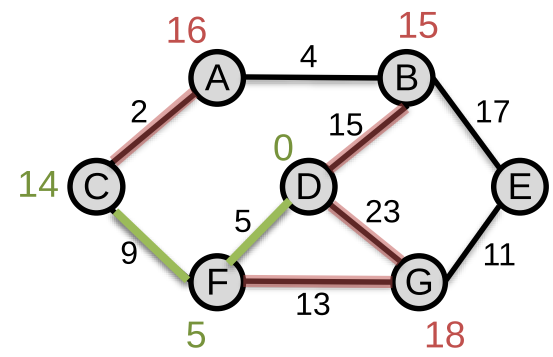
  – Step 2:
    - Label B = 15, G = 23, and F = 5
    - Therefore, N = {D, F}



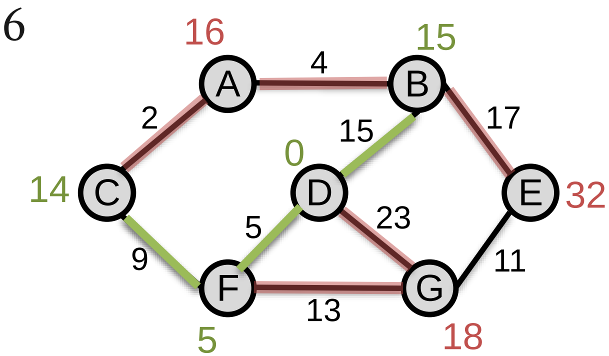
  – Step 3:
    - B = 15
    - Re-label G = minimum(5 + 13, 23) = 18
    - Label C = 14 (5 + 9)
    - Therefore, N = {D,F, C}

Dijkstra's Algorithm.....

- Step 4:
  - B = 15
  - G = 18
  - Label A = 16 (5+9+2)
  - Therefore, N = {D, F, C, B}



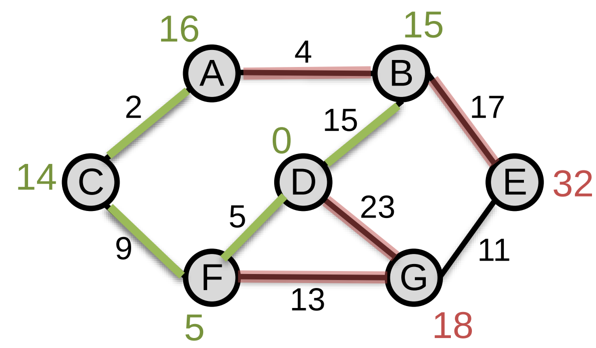
- Step 5:
  - G = 18
  - Re-label A = minimum(14+2=16,15+4=19)=16
  - Label E = 32 (15+17)
  - Therefore, N = {D, F, C, B, A}



Dijkstra's Algorithm……

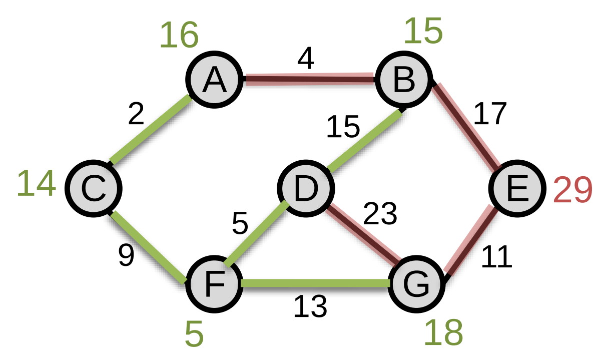
– Step 6:

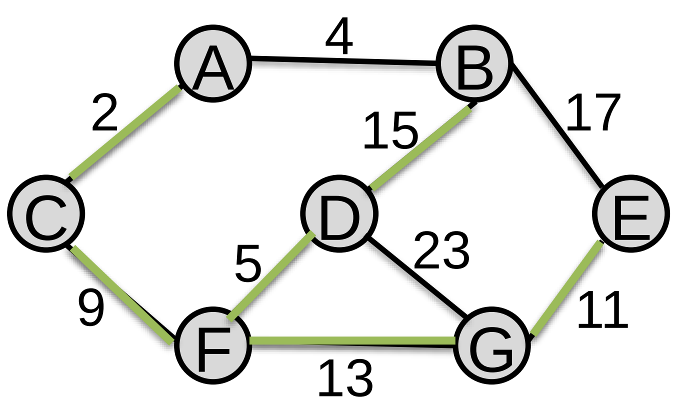
- G = 18
- E = 32
- Therefore, N = {D, F, C, B, A, G}



– Step 7:

- Re-label E = minimum(18+11,15+17) = 29
- Therefore, N = {D, F, C, B, A, G, E}



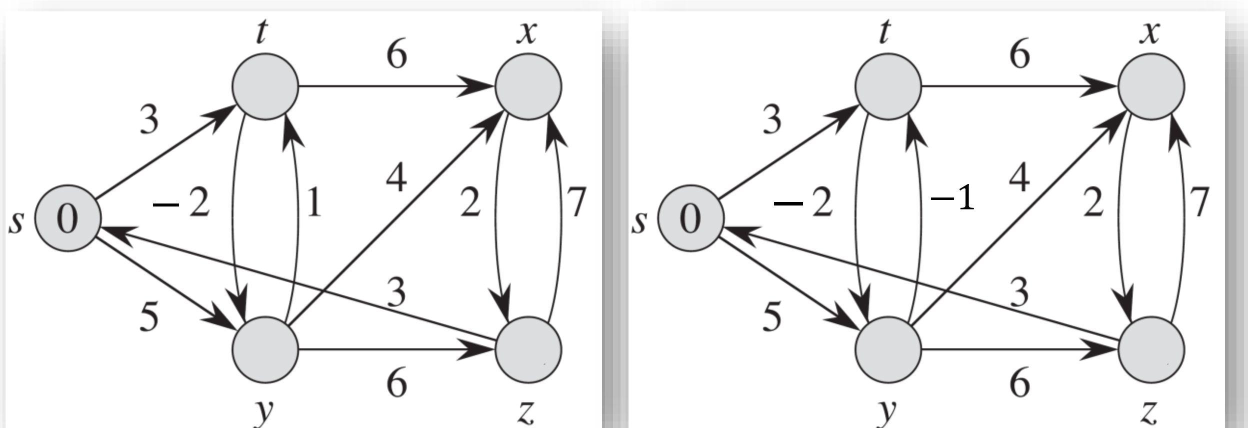


MST & Shortest Path Algorithms

- Dijkstra's algorithm is very similar to Prim's algorithm
  - Both the algorithms begin at a specific node and extend outward within the graph, until all other nodes in the graph have been reached
  - The difference is while Prim's algorithm stores a minimum cost edge, Dijkstra's algorithm stores the total cost from a source node to the current node

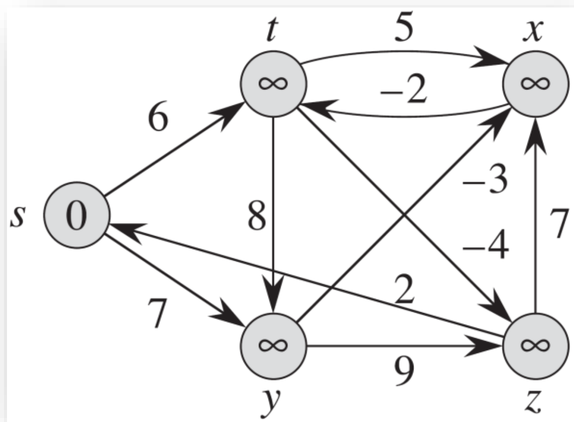
Bellman-Ford Algorithm

- The ***Bellman-Ford algorithm*** solves the single-source shortest-paths problem in the general case in which edge weights may be negative
  - The Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source
    - If there is such a cycle, the algorithm indicates that no solution exists (return false)
    - If there is no such cycle, the algorithm produces the shortest paths and their weights

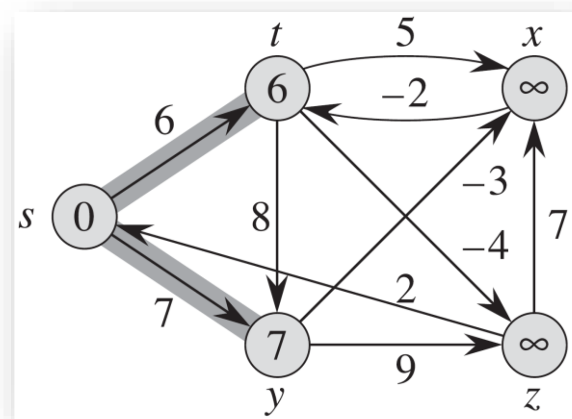


Example.

- The source is vertex s, and we assume that each pass relaxes the edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$

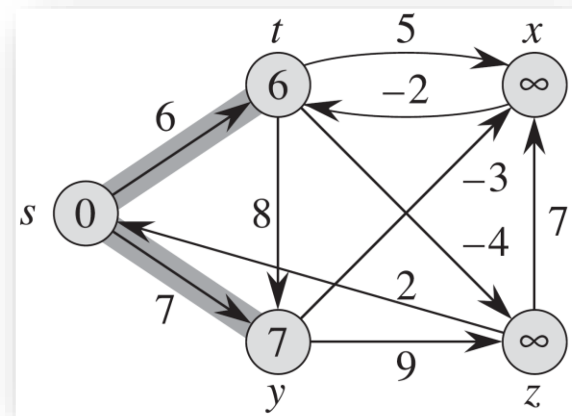


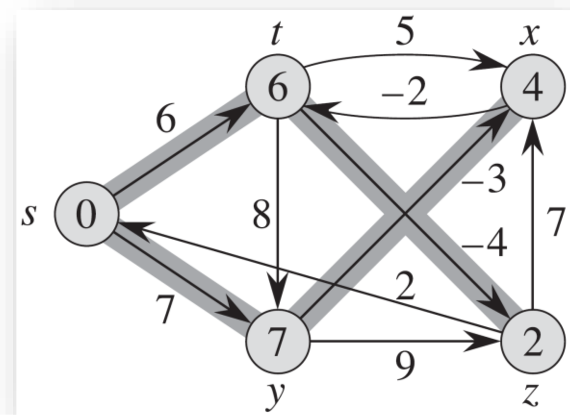
  - Iteration1:
    - (s,t): $t.d = 6$
    - (s,y): $y.d = 7$



Example..

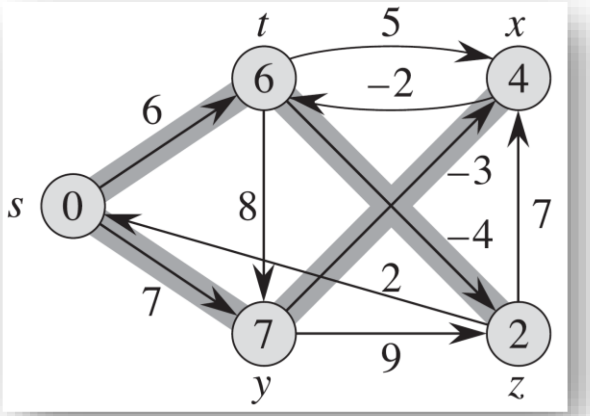
- The source is vertex s, and we assume that each pass relaxes the edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$
  - Iteration2:
    - (t,x): $x.d = 6+5 = 11$
    - (t,y): $y.d = 7 < 6+8$ *unchange*
    - (t,z): $z.d = 6-4 = 2$
    - (x,t): $t.d = 6 < 11-2 = 9$ *unchange*
    - (y,x): $x.d = 4 = 7-3 < 11$
    - (y,z): $z.d = 2 < 7+9$ *unchange*
    - (z,x): $x.d = 4 < 2+7 = 9$ *unchange*
    - (z,s): $s.d = 0 < 2+2 = 4$ *unchange*
    - (s,t): $t.d = 6 = 0+6$ *unchange*
    - (s,y): $y.d = 7 = 0+7$*unchange*

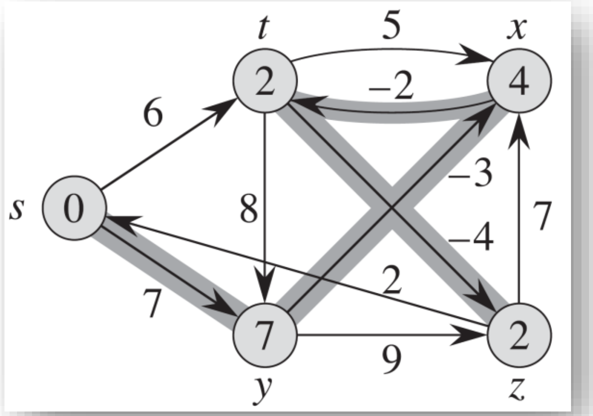




Example...

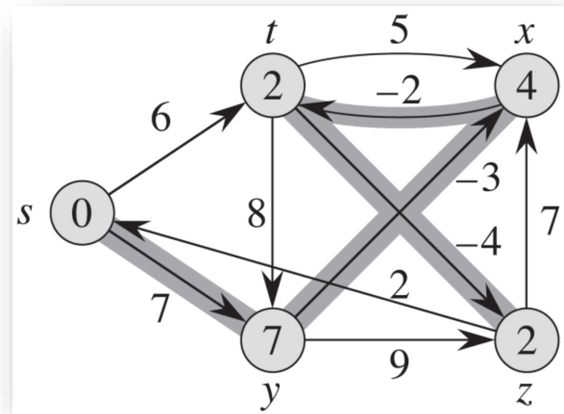
- The source is vertex s, and we assume that each pass relaxes the edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$
  - Iteration3:
    - (t,x): $x.d = 4 < 6 + 5$ *unchange*
    - (t,y): $y.d = 7 < 6 + 8$ *unchange*
    - (t,z): $z.d = 2 = 6 - 4$ *unchange*
    - (x,t): $t.d = 2 = 4 - 2 < 6$
    - (y,x): $x.d = 4 = 7 - 3$ *unchange*
    - (y,z): $z.d = 2 < 7 + 9$ *unchange*
    - (z,x): $x.d = 4 < 2 + 7 = 9$ *unchange*
    - (z,s): $s.d = 0 < 2 + 2 = 4$ *unchange*
    - (s,t): $t.d = 2 < 0 + 6 = 6$ *unchange*
    - (s,y): $y.d = 7 = 0 + 7$ *unchange*

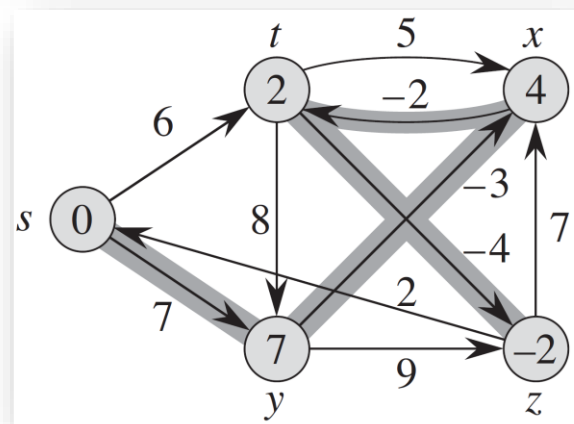




Example….

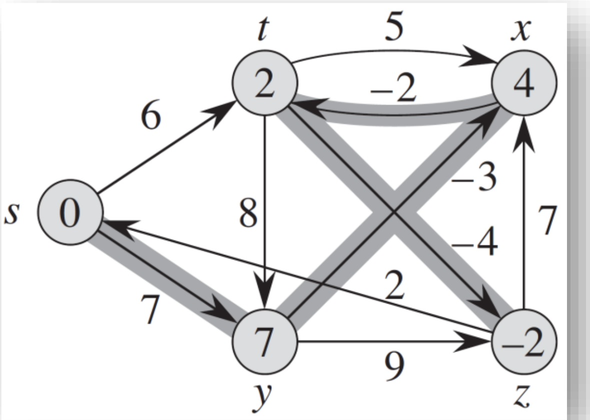
- The source is vertex s, and we assume that each pass relaxes the edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$
  - Iteration4:
    - (t,x): $x.d = 4 < 2 + 5$ *unchange*
    - (t,y): $y.d = 7 < 2 + 8$ *unchange*
    - (t,z): $z.d = -2 = 2 - 4 < 2$
    - (x,t): $t.d = 2 = 4 - 2$ *unchange*
    - (y,x): $x.d = 4 = 7 - 3$ *unchange*
    - (y,z): $z.d = -2 < 7 + 9$ *unchange*
    - (z,x): $x.d = 4 < -2 + 7 = 5$ *unchange*
    - (z,s): $s.d = 0 < -2 + 2 = 0$ *unchange*
    - (s,t): $t.d = 2 < 0 + 6 = 6$ *unchange*
    - (s,y): $y.d = 7 = 0 + 7$ *unchange*





Example.....

- The source is vertex s, and we assume that each pass relaxes the edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$
  - Check:
    - (t,x): $x.d = 4 < 2 + 5$ $unchange$
    - (t,y): $y.d = 7 < 2 + 8$ $unchange$
    - (t,z): $z.d = -2 = 2 - 4$ $unchange$
    - (x,t): $t.d = 2 = 4 - 2$ $unchange$
    - (y,x): $x.d = 4 = 7 - 3$ $unchange$
    - (y,z): $z.d = -2 < 7 + 9$ $unchange$
    - (z,x): $x.d = 4 < -2 + 7 = 5$ $unchange$
    - (z,s): $s.d = 0 < -2 + 2 = 0$ $unchange$
    - (s,t): $t.d = 2 < 0 + 6 = 6$ $unchange$
    - (s,y): $y.d = 7 = 0 + 7$ $unchange$



The Algorithm

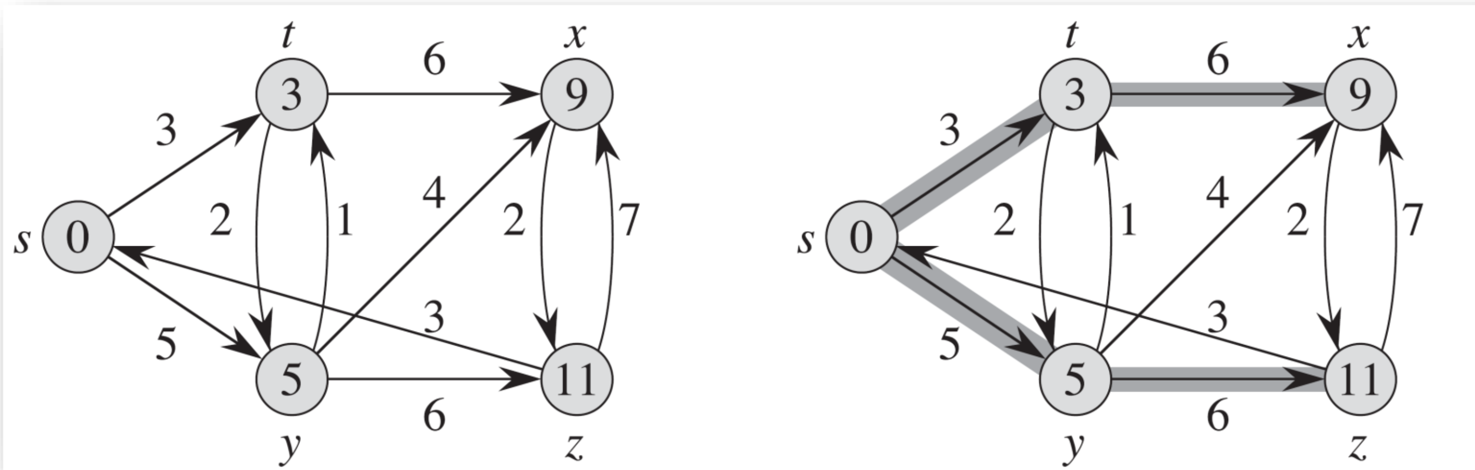
- The ***Bellman-Ford algorithm*** solves the single-source shortest-paths problem in the general case in which edge weights may be negative
  - The Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source
    - If there is such a cycle, the algorithm indicates that no solution exists (return false)
    - If there is no such cycle, the algorithm produces the shortest paths and their weights

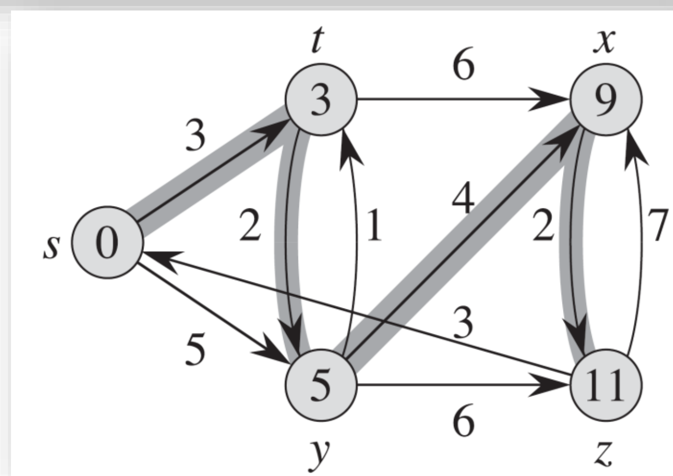
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Unique?

- Shortest paths are not necessarily unique





Questions?

kychen@mail.ntust.edu.tw