CS 2009 – Design and Analysis of Algorithms

Shortest Path Problem – I

Rizwan Ul Haq

Shortest Path

- In graph theory the shortest path between two vertices in a graph is a path between those two vertices in such way that the sum of weights of edges in that path is the smallest than the sum of weights of edges in any path between those two vertices provided the graph is a weighted graph.

- So, the shortest path problem is to find such a path between the given vertices.

Shortest path rooted tree

- A shortest-paths tree rooted at s is a directed subgraph $G' = \acute{V}, \acute{E}$ where $\acute{V} \subseteq$ V and $\acute{E} \subseteq$ E, such that:

  - V\` is the set of vertices reachable from ***s*** in G,
  - G\` forms a rooted tree with root ***s***, and
  - for all v ∈ V\`, the unique simple path from ***s*** to ***v*** in G\` is a shortest path from ***s*** to ***v*** in G.

Not necessarily unique!

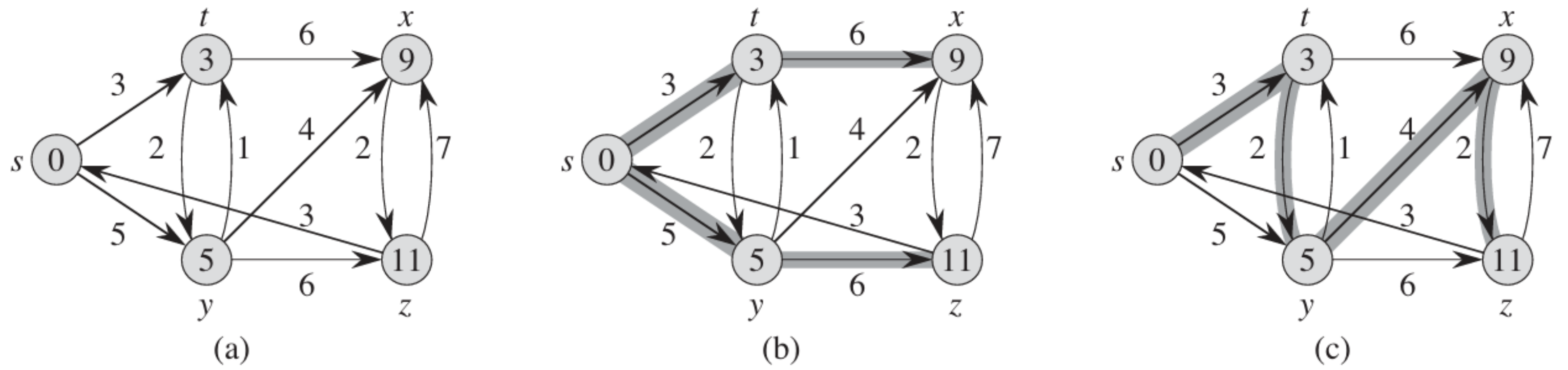
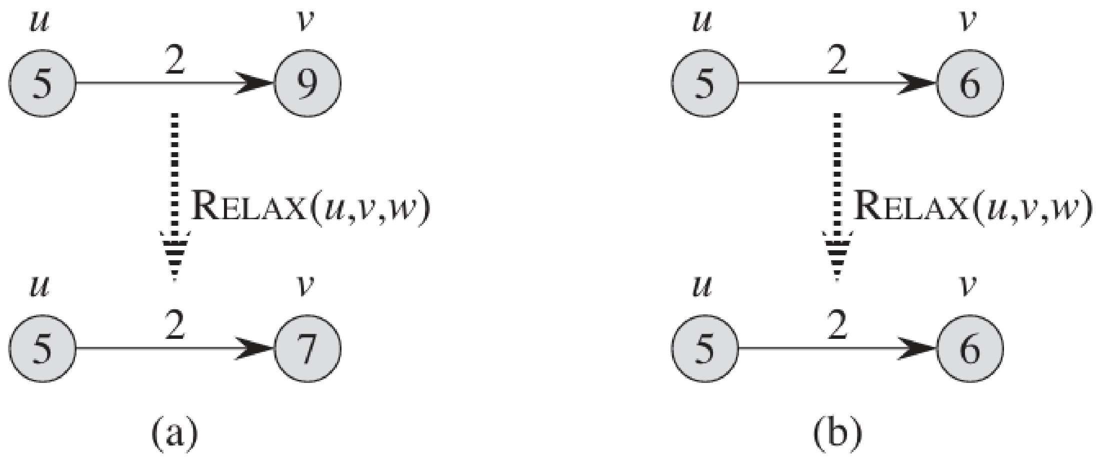


Figure 24.2 **(a)** A weighted, directed graph with shortest-path weights from source *s*. **(b)** The shaded edges form a shortest-paths tree rooted at the source *s*. **(c)** Another shortest-paths tree with the same root.

Relaxation

- The process of relaxing an edge **(*u*,*v*)** consists of testing whether we can improve the shortest path to ***v*** found so far by going through ***u*** and, if so, updating ***v.d*** and ***v.π***
- Relaxation step may decrease the value of the shortest-path



```
RELAX(u, v, w)
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$
```

Single-Source Shortest Path Problem

- **Single-Source Shortest Path Problem** - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.
  - Dijkstra's Algorithm
  - Bellman Ford Algorithm

Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

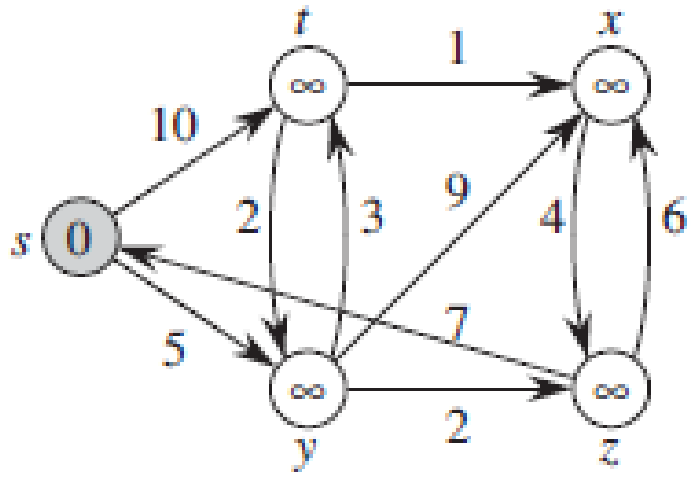
Works on both directed and undirected graphs.

Approach: Greedy

Input: Weighted graph G={E,V} and source vertex $s \in V$,

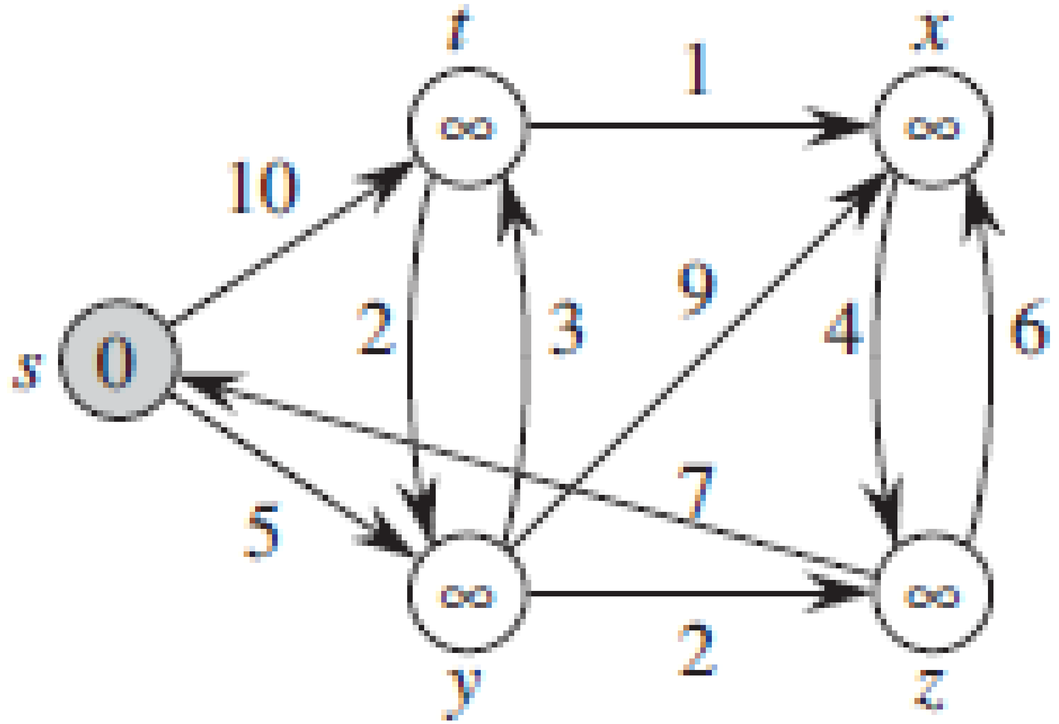
Output: the shortest path from a given source vertex $s \in V$ to all other vertices

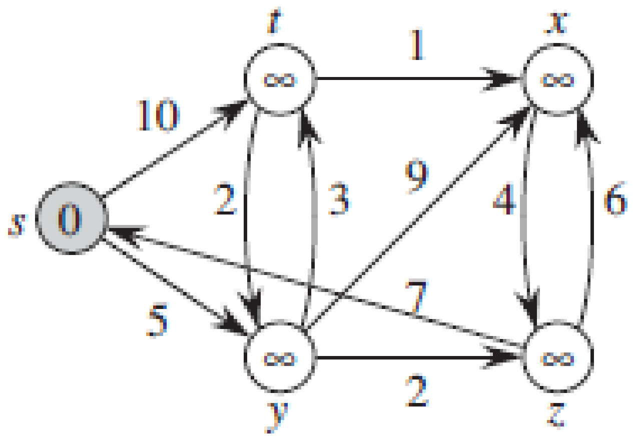
Dijkstra’s Algorithm



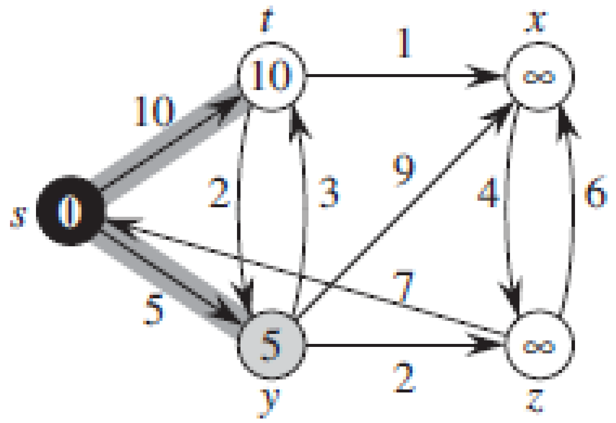
```
1.  d[s] -> 0
2.  for each vertex $v \in \{V[G] - S\}$
3.      do   $d[v] = \infty$
4.           $\pi[v] = NIL$
5.  $S = \emptyset$
6.  $Q = G.V$
7.  While $Q \neq \emptyset$
8.  do   $u = EXTRACT \min(Q)$
9.       $S = SU\{u\}$
10.      for each vertex $v \in Adj[u]$
11.          do   $if\ (v \in Q\ and\ d[v] > d[u] + w(u,v)\ )$
12.                   $d[v] = d[u] + w(u,v))$
13                    $\pi[v] = u$
```

Example

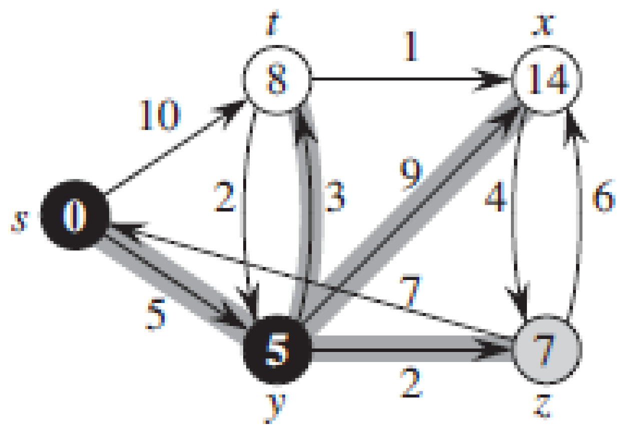




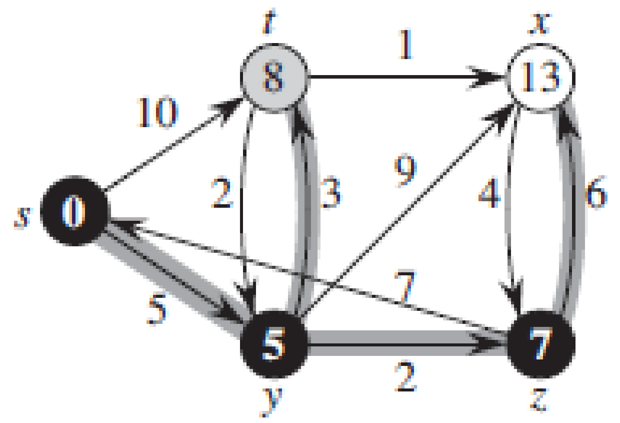
t
x
s
y
z
0
∞
10
1
2
3
9
4
6
7
5
2



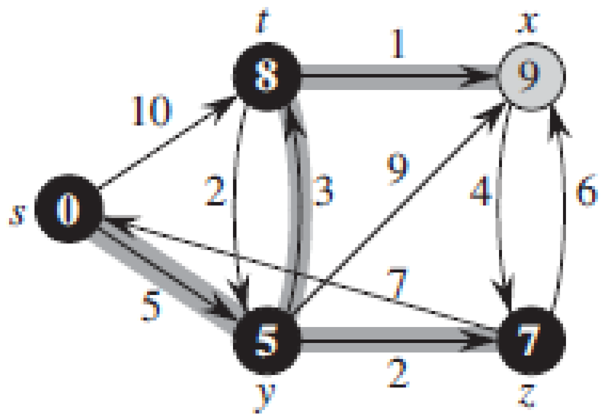
t
x
s
y
z
0
10
5
∞
10
1
2
3
9
4
6
7
5
2

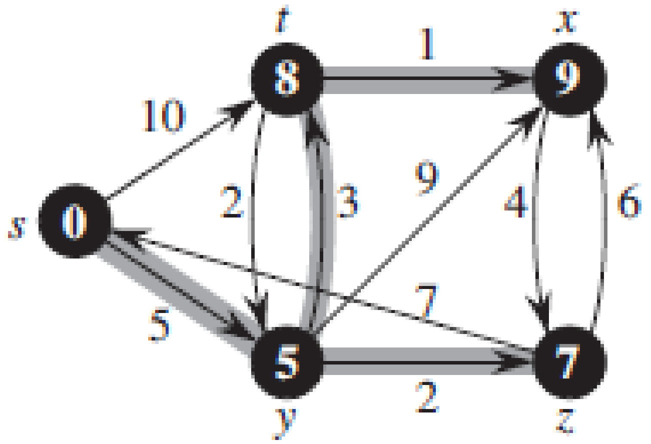


t
x
8
14
1
10
2
3
9
4
6
s
0
5
7
5
7
2
y
z

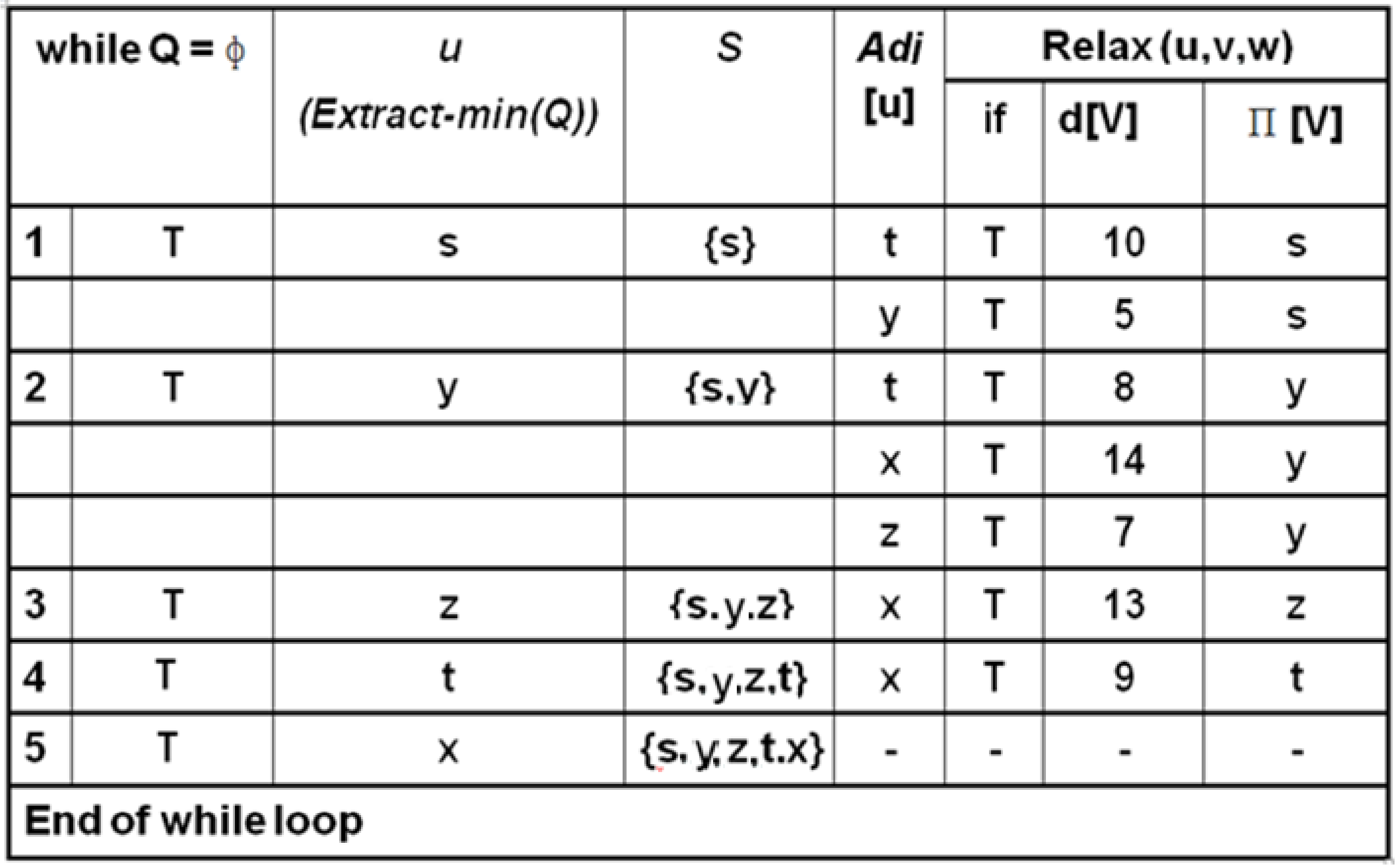


t
x
8
13
1
10
2
3
9
4
6
s
0
5
7
5
7
2
y
z


t
x
s
y
z
8
9
0
5
7
1
10
2
3
9
4
6
7
5
2


t
x
s
y
z
8
9
0
5
7
1
10
2
3
9
4
6
7
5
2

Example



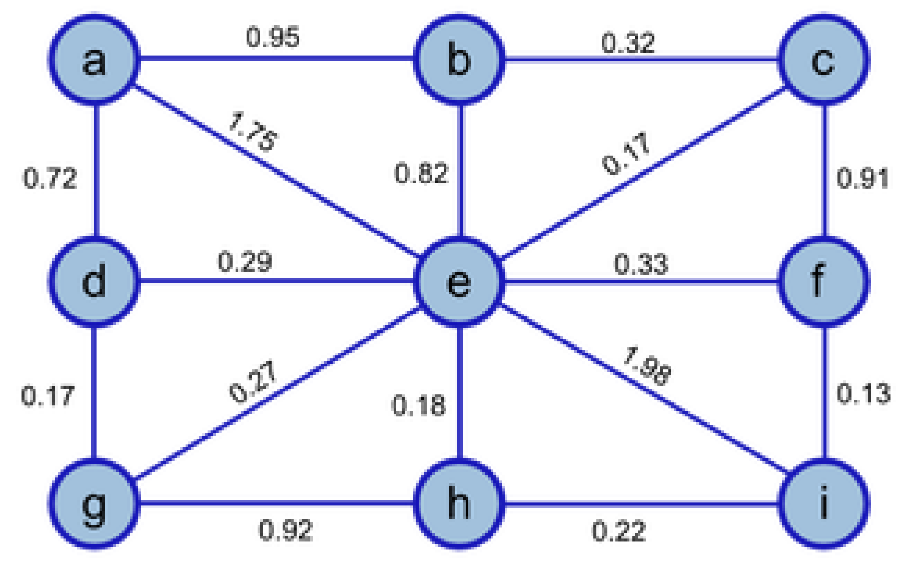
while Q = ϕ		*u* *(Extract-min(Q))*	*S*	*Adi* [u]	Relax (u,v,w)		
					if	d[V]	Π [V]
1	T	s	{s}	t	T	10	s
				y	T	5	s
2	T	y	{s,y}	t	T	8	y
				x	T	14	y
				z	T	7	y
3	T	z	{s.y.z}	x	T	13	z
4	T	t	{s,y.z,t}	x	T	9	t
5	T	x	{s, y, z,t.x}	-	-	-	-
End of while loop							

Analysis: O(E log V)

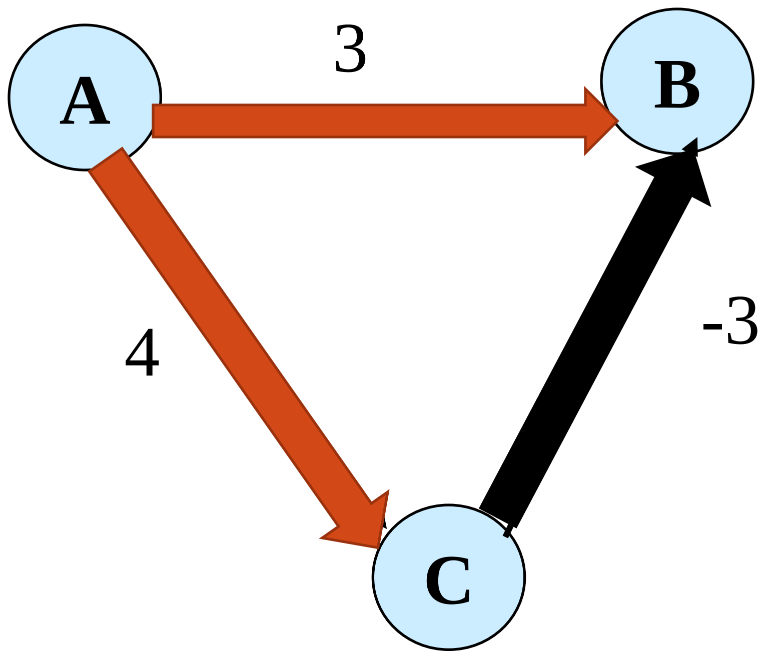
```
1.  d[s] -> 0
2.  for each vertex $v \in \{V[G] - S\}$
3.      do  $d[v] = \infty$
4.          $\pi[v] = NIL$
5.  $S = \emptyset$
6.  $Q = G.V$
7.  While $Q \neq \emptyset$
8.  do  $u = EXTRACT\min(Q)$
9.      $S = SU\{u\}$
10.     for each vertex $v \in Adj[u]$
11.         do  $if\ (v \in Q\ and\ d[v] > d[u] + w(u,v)\ )$
12.                 $d[v] = d[u] + w(u,v))$
13                  $\pi[v] = u$
```

- 2-6 -> O(V)
- 8 -> O(V log V)
- 10-13 -> O(E log (V))

Task
Start Node a.



Dijkstra's algorithm failure

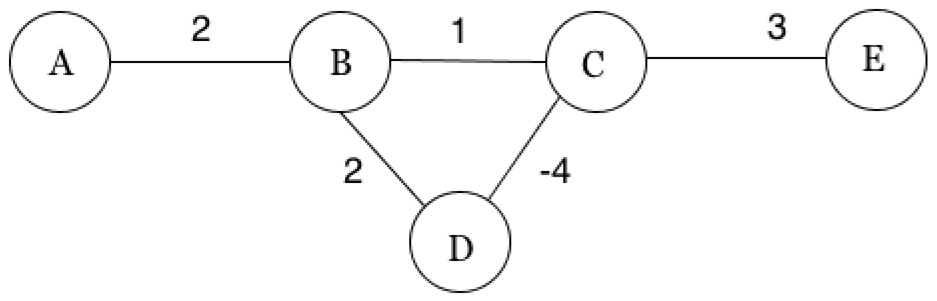


Negative edges

- The weight on edges can represent everything in real world.
- A negative edge is simply an edge having a negative weight.
- Example:
  - The weights can be used to represent the heat produced during a chemical reaction. Energy out of the reaction i.e released energy represented with +ve weights and absorbed energy with -ve.
  - Suppose if you doing job, to going from state ***a*** to ***b*** costs 2$ (e.g job is buying book costs 2$), after that you can do some project (you earn 2$, means cost function is -2), then total cost is 0, a--(+2) →b--(-2) → c : +2 - 2 = 0

Negative cycle

- A negative cycle is a cycle in a weighted graph whose edges sum to a negative value



Bellman-Ford algorithm

- Bellman-Ford SSSP algorithm can handle negative edge weights in directed graphs only.
- It even can detect negative weight cycles if they exist. but in presence of negative cycle don't find correct shortest path.

Bellman-Ford algorithm

```
Bellman-Ford (G,W,s)
1.  d[s] = 0
2.  $\pi[v] = NIL$
3.  for each vertex $v \in \{V[G] - S\}$
4.      do  $d[v] = \infty$
5.          $\pi[v] = NIL$
6.  for i=1 to $|G.V| - 1$
7.      for each edge (u,v) ∈ G.E
8.          if d[v] >d[u] + w(u,v)  then
9.              d[v] = d[u] + w(u,v)
10.             $\pi[v]$= u
11. for each edge (u,v) ∈ G.E
12.         if d[v] >d[u] + w(u,v)  then
13.           return False
```

Shortest path

Cycle detection

Bellman-Ford algorithm

Bellman-Ford Algo (G,W,s)

```
1.  d[s] = 0
2.  $\pi[v] = NIL$
3.  for each vertex $v \in \{V[G] - S\}$
4.      do  $d[v] = \infty$
5.          $\pi[v] = NIL$
6.  for i=1 to $|G.V| - 1$
7.      for each edge (u,v) $\in$ G.E
8.          if d[v] >d[u] + w(u,v) then
9.              d[v] = d[u] + w(u,v)
10.             $\pi[v]$= u
11. for each edge (u,v) $\in$ G.E
12.         if d[v] >d[u] + w(u,v) then
13.             return False
```

Dijkstra's Algo (G,W,s)

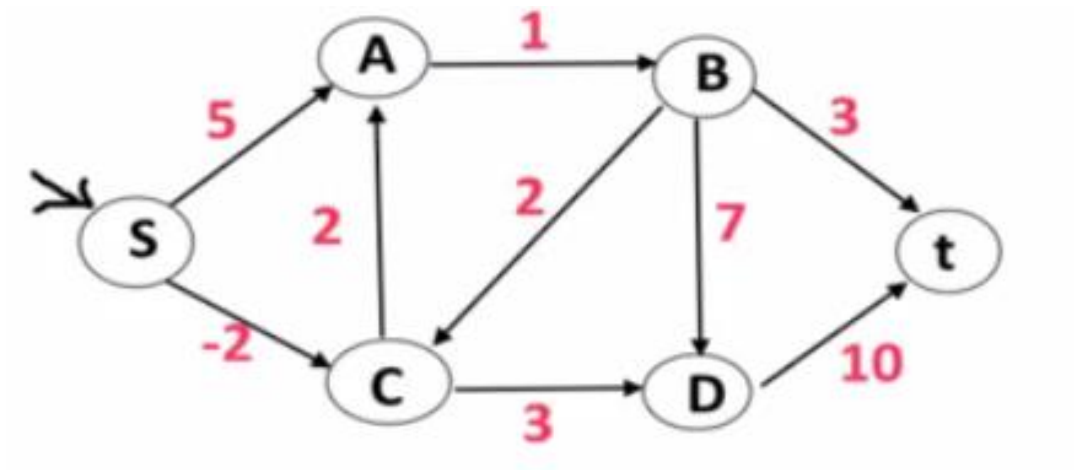
```
1.  d[s] -> 0
2.  for each vertex $v \in \{V[G] - S\}$
3.      do  $d[v] = \infty$
4.          $\pi[v] = NIL$
5.  $S = \emptyset$
6.  $Q = G.V$
7.  While $Q \neq \emptyset$
8.  do  $u = EXTRACT\min(Q)$
9.      $S = SU\{u\}$
10.     for each vertex $v \in Adj[u]$
11.         do  $if\ (v \in Q\ and\ d[v] > d[u] + w(u,v)\ )$
12.                 $d[v] = d[u] + w(u,v))$
13                  $\pi[v] = u$
```

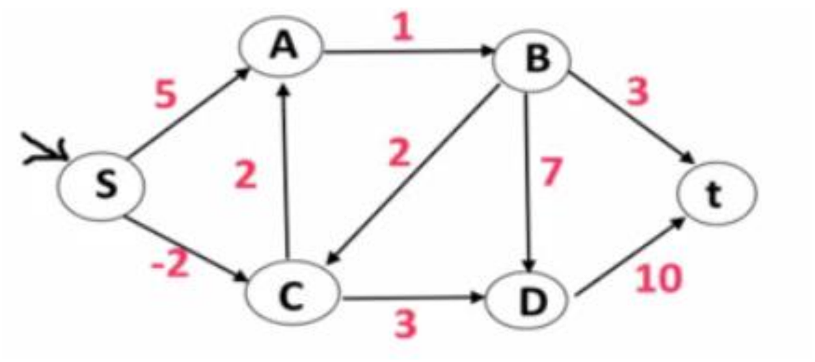
Bellman-Ford algorithm

- Consider each edge (***u***,*v*) and see if u offers *v* a cheaper path from ***s***
  - compare d[*v*] to d[***u***] + w(***u***,*v*)

- Repeat this process |V| - 1 times. We have no idea what is the ith iteration after which no relaxation change occurs, but we know that it will happen at some point that all edges are relaxed. (if there is no negative cycle)

Example



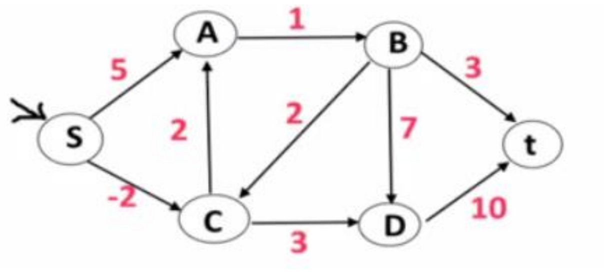


Initially

	S	A	B	C	D	T
d[v]	0	∝	∝	∝	∝	∝
$\pi[v]$	NIL	NIL	NIL	NIL	NIL	NIL

1st Iteration

	S	A	B	C	D	T
d[v]	0	5, 0	6	-2	13,1	9
$\pi[v]$	NIL	S, C	A	S	B,C	B



2nd Iteration

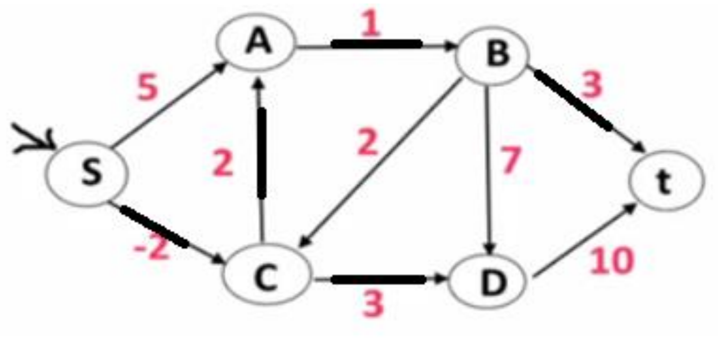
	S	A	B	C	D	T
d[v]	0	0	6,1	−2	1	9,4
$\pi[v]$	NIL	C	A,A	S	C	B,B

3rd Iteration

	S	A	B	C	D	T
d[v]	0	0	1	−2	1	4
$\pi[v]$	NIL	C	A	S	C	B

4th Iteration

	S	A	B	C	D	T
d[v]	0	0	1	−2	1	4
$\pi[v]$	NIL	C	A	S	C	B



5th Iteration

	S	A	B	C	D	T
d[v]	0	0	1	−2	1	4
$\pi[v]$	NIL	C	A	S	C	B

6th Iteration

	S	A	B	C	D	T
d[v]	0	0	1	−2	1	4
$\pi[v]$	NIL	C	A	S	C	B

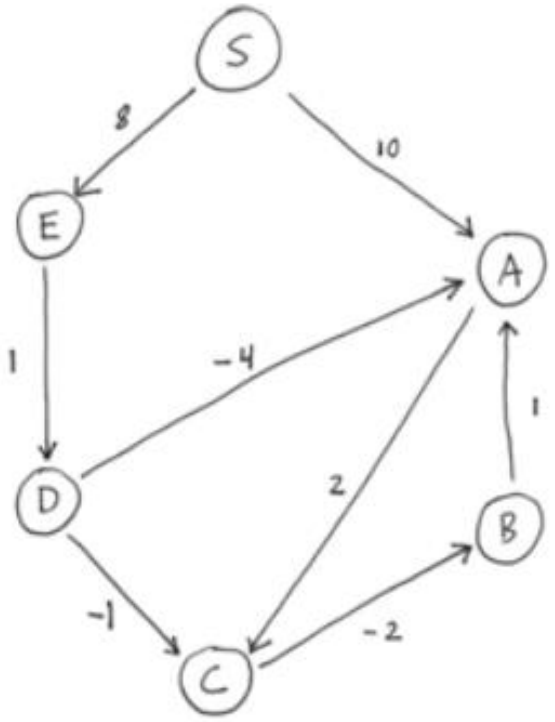
we have no idea what is the *ith* iteration after which no relaxation change occurs, but we know that it will happen at some point that all edges are relaxed. (if there is no negative cycle

Analysis

```
1.  d[s] = 0
2.  $\pi[v] = NIL$
3.  for each vertex $v \in \{V[G] - S\}$
4.      do  $d[v] = \infty$
5.          $\pi[v] = NIL$
6.  for i=1 to $|G.V| - 1$
7.      for each edge (u,v) ∈ G.E
8.          if d[v] >d[u] + w(u,v)  then
9.              d[v] = d[u] + w(u,v)
10.             $\pi[v]$= u
11. for each edge (u,v) ∈ G.E
12.         if d[v] >d[u] + w(u,v)  then
13.          return False
```

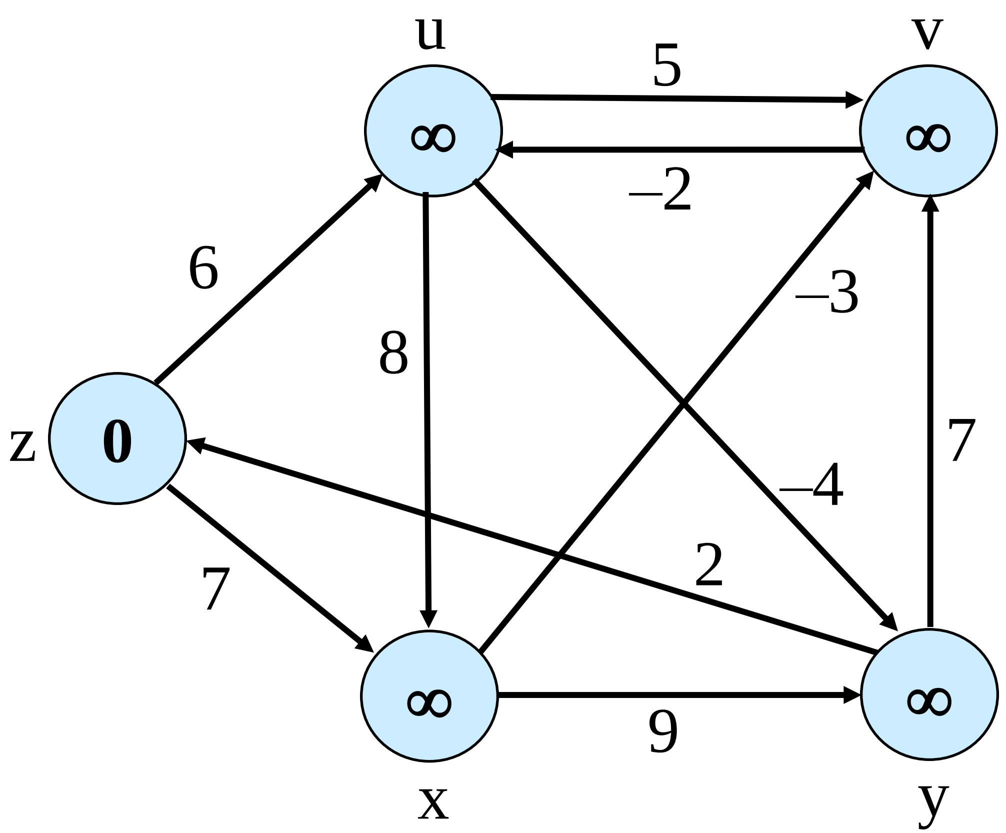
- O(V) iterations of outer for loop
- O(E) iterations of inner for loop
- O(VE) time total ≈ n^2
- ***What would be the case if the graph has maximum number of edges?***

Task:



6 vertices =
5 iterations

Homework



- Bellman-Ford has two relevant invariants that hold for all vertices u.
- There exists a path from the source to ***u*** of length dist[***u***] length (unless dist[***u***] is MAX).
- After ***i*** iterations of the outer loop, for all paths from the source to u with ***i*** or fewer edges, the distnce of that path is no less than dist[***u***].