Dijkstra's Algorithm

Dr. Yasir Ali

College of E&ME,
National University of Sciences and Technology,
Islamabad, Pakistan.

December 26, 2013

Weighted Graph, Shortest Path & Negative Cycle

- $w : E \to \mathbf{R}$ is known as *weight function*
- w to an edge is called *weight of the edge*
- A graph $G = (V, E)$ in which each edge has a weight is called a *weighted graph.*
- Let $S = v_1, v_2, \ldots, v_k$ be a path in a weighted graph G then weight of S is given by

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

- A *shortest path* from a vertex v to a vertex v' is defined as any $v - v'$ path S with weight $w(S) = d(v, v')$ where $d(v, v')$ is given by

$$d(v, v') = \begin{cases} \min\{w(\widetilde{P}) \mid \widetilde{P} \text{ is a } v - v' \text{ path}\} & \text{if } v' \text{ is reachable from } v \\ +\infty & \text{otherewise.} \end{cases}$$

- A cycle C in a weighted graph with $w(C) < 0$ is called a *negative cycle.*

Dijkstra's Algorithm (Main Idea)

- The idea is to visit the nodes in order of their closeness to *source vertex* s.

The closest node to s, say x, must be adjacent to s and the next closest node, say y, must be either adjacent to s or x. The third closest node to s must be either adjacent to s or x or y, and so on.

Predecessor $\pi[v]$ & Shortest Path Estimate $d[v]$

We set the π attribute so that the chain of predecessors originating at a vertex v runs backwards along the shortest path form vertex s to v.

- Given a graph $G(V, E)$, we maintain for each vertex $v \in V$ a *predecessor* $\pi[v]$ that is either a vertex or `NIL`

For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to vertex v.

- We call $d[v]$ a *Shortest Path Estimate*

Initialize . . .

We initialize the Shortest Path Estimate and Predecessor by the following procedure:

`INITIALIZE-SINGLE-SOURCE` (G, s)

1. **for** each vertex $v \in V$
2. **do** $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$`NIL`
4. $d[s] = 0$

After initialization, $\pi[v] =$`NIL`, $d[v] = 0$ for $v = s$ and $d[v] = \infty$ for all $v \in V - s$

Relaxation

- The process of relaxing an edge (u, v) consist of testing whether we can improve the shortest path to v found so far by going through u and, if so, update $d[v]$ and $\pi[v]$.

A relaxation step may decrease the shortest path estimate $d[v]$ and updates's v's predecessor field $\pi[v]$.

`RELAX` (u, v, w)

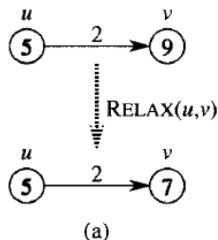
1. **if** $d[v] > d[u] + w(u, v)$
2. **then** $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

Relaxation

RELAX (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$ (if new shortest path found)
2. **then** $d[v] \leftarrow d[u] + w(u, v)$ (set new value of shortest path)
3. $\pi[v] \leftarrow u$ (add u to predecessor of v)

Relaxation of an edge (u, v).



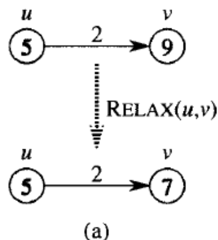
(a) Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases.

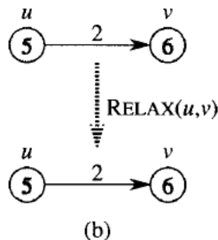
Relaxation

`RELAX` (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$ (if new shortest path found)
2. **then** $d[v] \leftarrow d[u] + w(u, v)$ (set new value of shortest path)
3. $\pi[v] \leftarrow u$ (add u to predecessor of v)

Relaxation of an edge (u, v).





(a) Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases.

(b) Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, so $d[v]$ is unchanged by relaxation.

Dijkstra's Algorithm

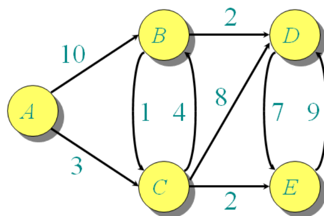
Dijkstra's algorithm maintains a set S of those vertices for which the shortest path weights from the source s have already been determined. The algorithm repeated selects the vertices $u \in V \setminus S$ with shortest path estimates, and insert u into S, and relaxes all the edges leaving u In the following implementation, we maintain a priority queue Q that contains all the vertices in $V \setminus S$ keyed by their d values.

DIJKSTRA (G, w, s)

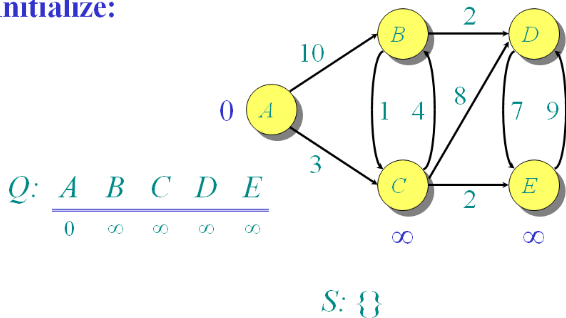
	Pseudocode	Comment
1	INITIALIZE-SINGLE-SOURCE (G, s)	(distance to source vertex is zero)
		(set all other distances to infinity)
2	$S \leftarrow \emptyset$	(S, the set of visited vertices is initially empty)
3	$Q \leftarrow V[G]$	(Q, the queue initially contains all vertices)
4	**while** $Q \neq \emptyset$	(while the queue is not empty)
5	**do** $u \leftarrow EXTRACT - MIN(Q)$	(select the element of Q with the min. distance)
6	$S \leftarrow S \cup \{u\}$	(add u to list of visited vertices)
7	**for** each vertex $v \in Adj[u]$	
8	**do** RELAX(u, v, w)	(if new shortest path found)
		(set new value of shortest path)

Dijkstra's Algorithm Implementation

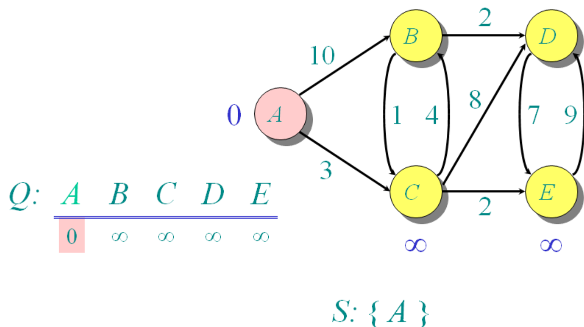
Graph with nonnegative edge weights:



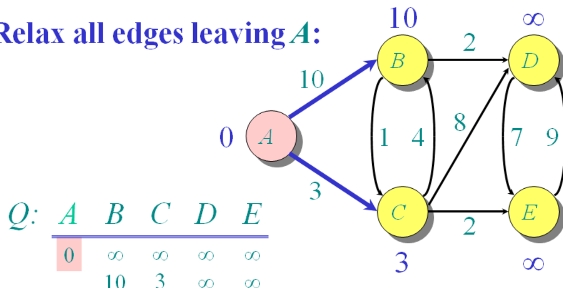
Dijkstra's Algorithm Implementation



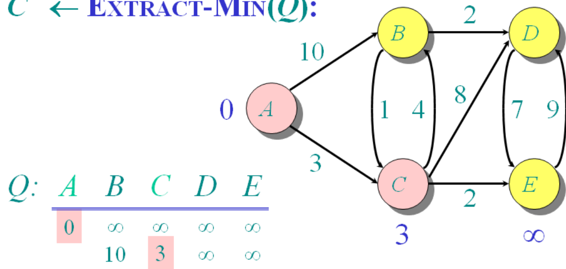
Dijkstra's Algorithm Implementation



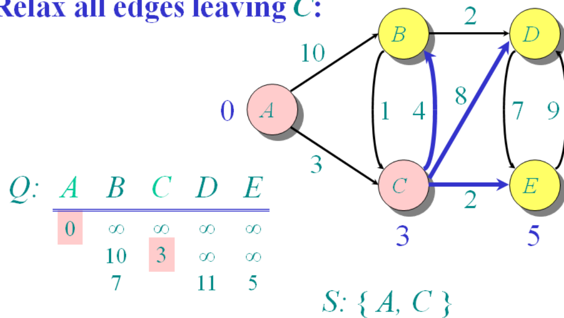
Dijkstra's Algorithm Implementation



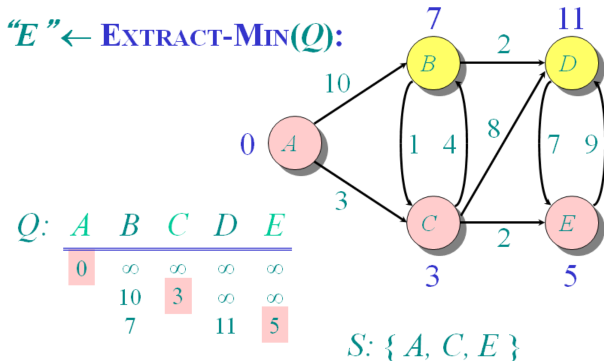
Dijkstra's Algorithm Implementation



Dijkstra's Algorithm Implementation

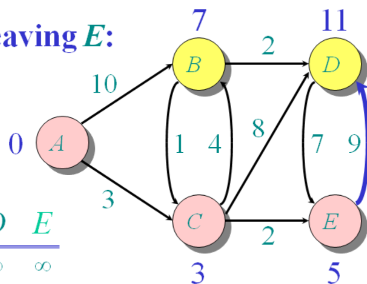


Dijkstra's Algorithm Implementation



Dijkstra's Algorithm Implementation

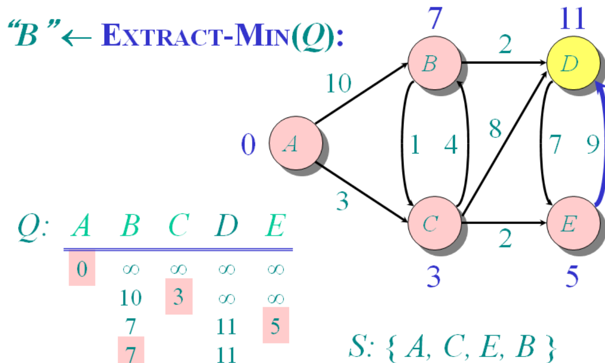
Relax all edges leaving *E*:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

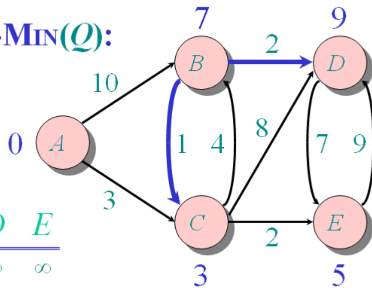
S: { A, C, E }

Dijkstra's Algorithm Implementation



Dijkstra's Algorithm Implementation

"D" $\leftarrow$ EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }