Department of
Computer Engineering

Data Structure & Algorithms

Shortest Path

Example Problem

- The ticket prices for traveling by bus between pairs of cities are known. Implement a travel planner for planning a journey from a source city to a destination city, using buses such that the total cost of the journey is minimum.

Find a shortest path from source to destination,
taking into account that edge weights are positive!

Properties of Shortest paths

1. The optimal substructure property: Any subpath of a shortest path is a shortest path.
2. Shortest paths cannot contain cycles.

Designing shortest path algorithms

- The *starting point* is the optimal substructure property: *Subpaths of shortest paths are also shortest paths*
- If we know the shortest paths composed of maximum k vertices, we can build shortest paths of maximum k+1 vertices by adding a new vertex to one of the paths

Dijkstra’s algorithm – The Idea

- Consider all the vertices in the order of their shortest paths from the source vertex s
  - Initially, we check all outgoing edges from s. Let (s, x) be the minimum edge outgoing from s. Because *all edges are positive*, it is also the shortest path from s to x.
  - Next step: find the shortest path from s to one more vertex (other than x). The only paths to consider are other edges from s or a path formed by (s, x) and an outgoing edge from x.

Relaxation

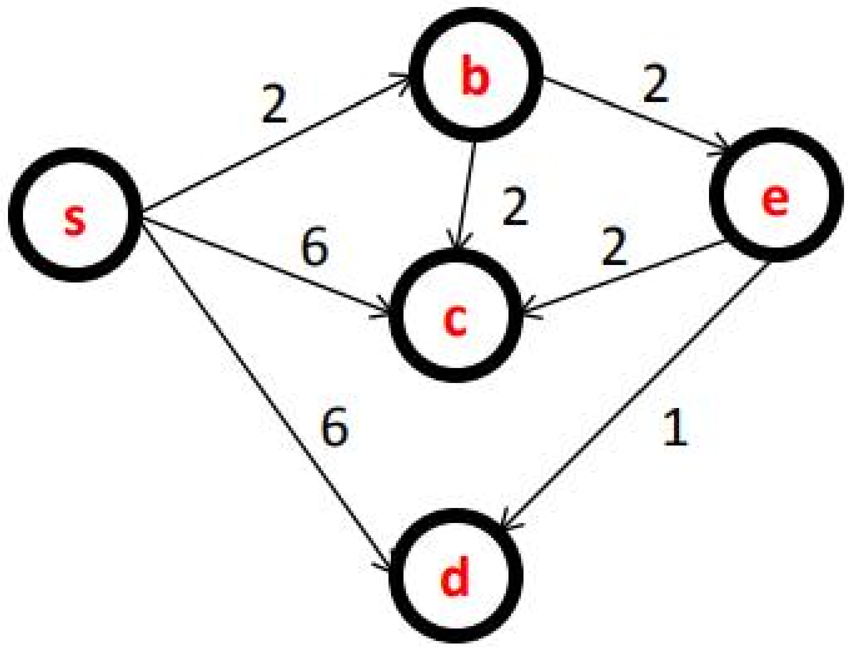
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

Dijkstra Algorithm - details

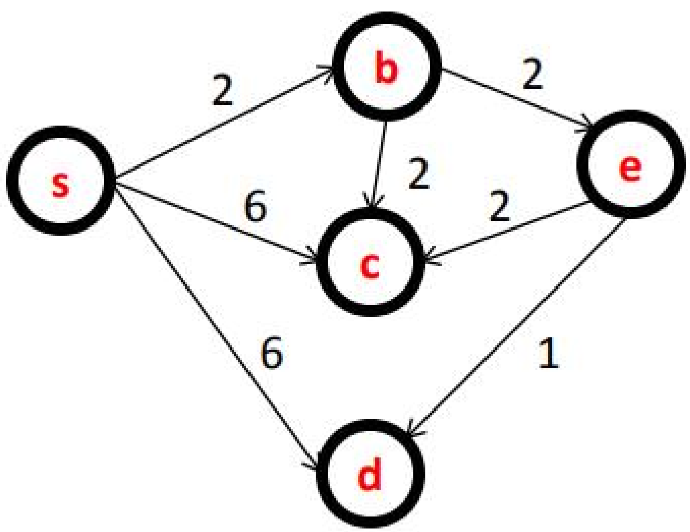
- D[i]: the weight of the shortest path from s to i
- $D[i] = \begin{cases} \infty & no\ edge\ from\ s\ to\ i \\ w[s,i] & an\ edge\ from\ s\ to\ i, whose\ weight\ is\ w[s,i] \end{cases}$
- w is the node with the shortest D[i]
- $D[v] = \min\{D[v],\ D[w] + w[w,v]\}$, for all v nodes which are neighbors to w

Dijkstra Algorithm – example

- A weighted graph with non-negative weights:



Dijkstra Algorithm – example (step 1)



	s	b	c	d	e
D(x)	0	2	6	6	∞

x <- extract min

//D(x) is the distance of s to x

// mark x as final

"relax" all the edges out of x

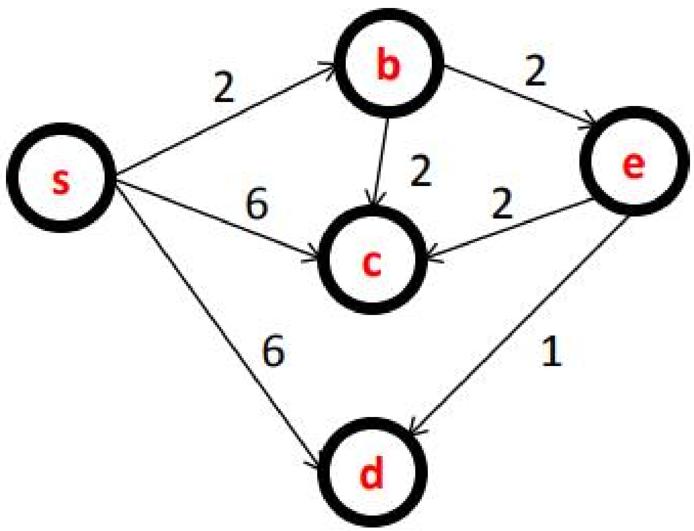
D(y) <- min (D(y), D(x) + w(x,y))

Relaxation Step

- The list of b neighbors are c and e, so:

$$\begin{cases} D[c] = \min\{D[c], D[b] + w[b,c]\} = 4 \\ D[e] = \min\{D[e], D[b] + w[b,e]\} = 4 \end{cases}$$

Dijkstra Algorithm – example (step 2)



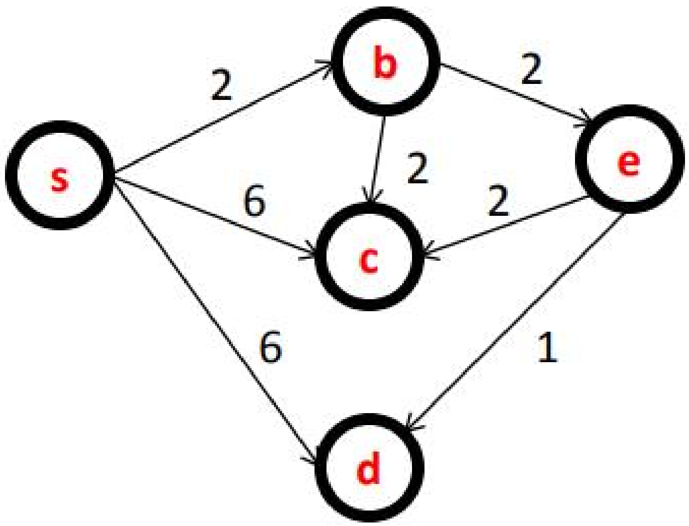
	s	b	c	d	e
D(x)	0	2	4	6	4

x <- extract min

//D(x) is the distance of s to x
// mark x as final

"relax" all the edges out of x
D(y) <- min (D(y) , D(x) + w(x,y))

Dijkstra Algorithm – example (step 3)



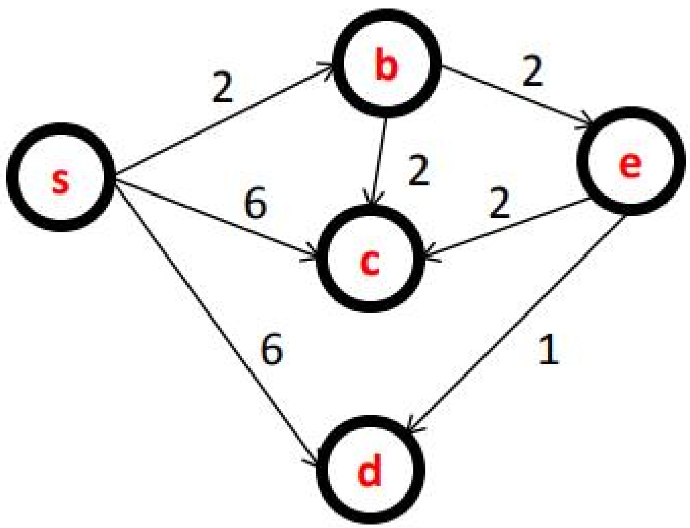
	s	b	c	d	e
D(x)	0	2	4	6	4

x <- extract min

//D(x) is the distance of s to x
// mark x as final

"relax" all the edges out of x
D(y) <- min (D(y) , D(x)+ w(x,y))

Dijkstra Algorithm – example (step 4)



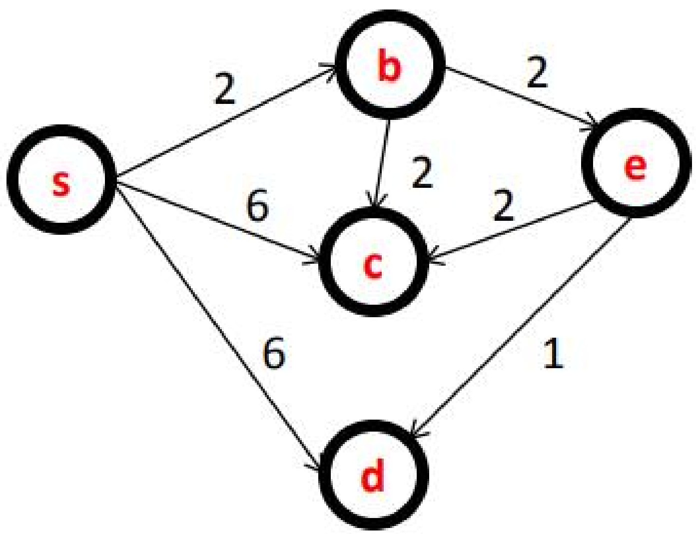
	s	b	c	d	e
D(x)	0	2	4	5	4

x <- extract min

//D(x) is the distance of s to x
// mark x as final

"relax" all the edges out of x
    D(y) <- min (D(y), D(x) + w(x,y))

Dijkstra Algorithm



	s	b	c	d	e
D(x)	0	2	4	5	4

x <- extract min

//D(x) is the distance of s to x
// mark x as final

"relax" all the edges out of x
D(y) <- min (D(y) , D(x) + w(x,y))

Dijkstra Algorithm - Analysis

- Dijkstra's algorithm uses a data structure for storing and querying partial solutions sorted by distance from the start.

- Arrays → $O(|V|^2)$

- Mean-Heap → $O(|E| + |V|\log|V|)$