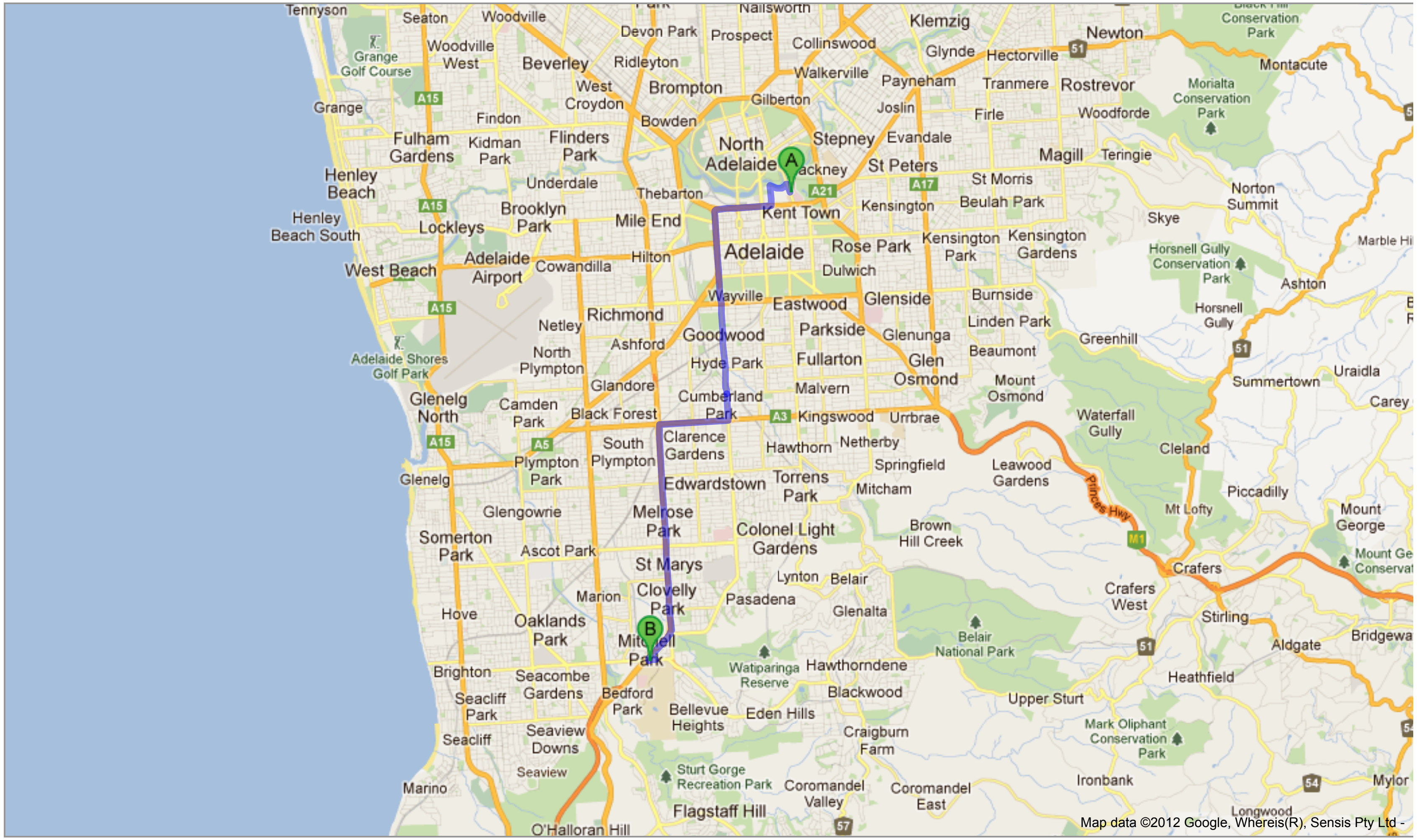
Algorithm and Data Structure Analysis (ADSA)

Shortest Paths

Problem

- Computation of shortest path is one of the classical problems.
- Classical application is route planning.

Uni Adelaide – Flinders Uni



Problem Statement

Given a directed graph G=(V,E) and a cost function $c : E \rightarrow R$ on the edges.

Given a path $p = (e_1, e_2, \ldots, e_k)$ consisting of k edges the cost of the path is

$$c(p) = \sum_{i=1}^{k} c(e_i)$$

A shortest path from a node s to a node v is a path of minimal cost among all possible paths from s to v.

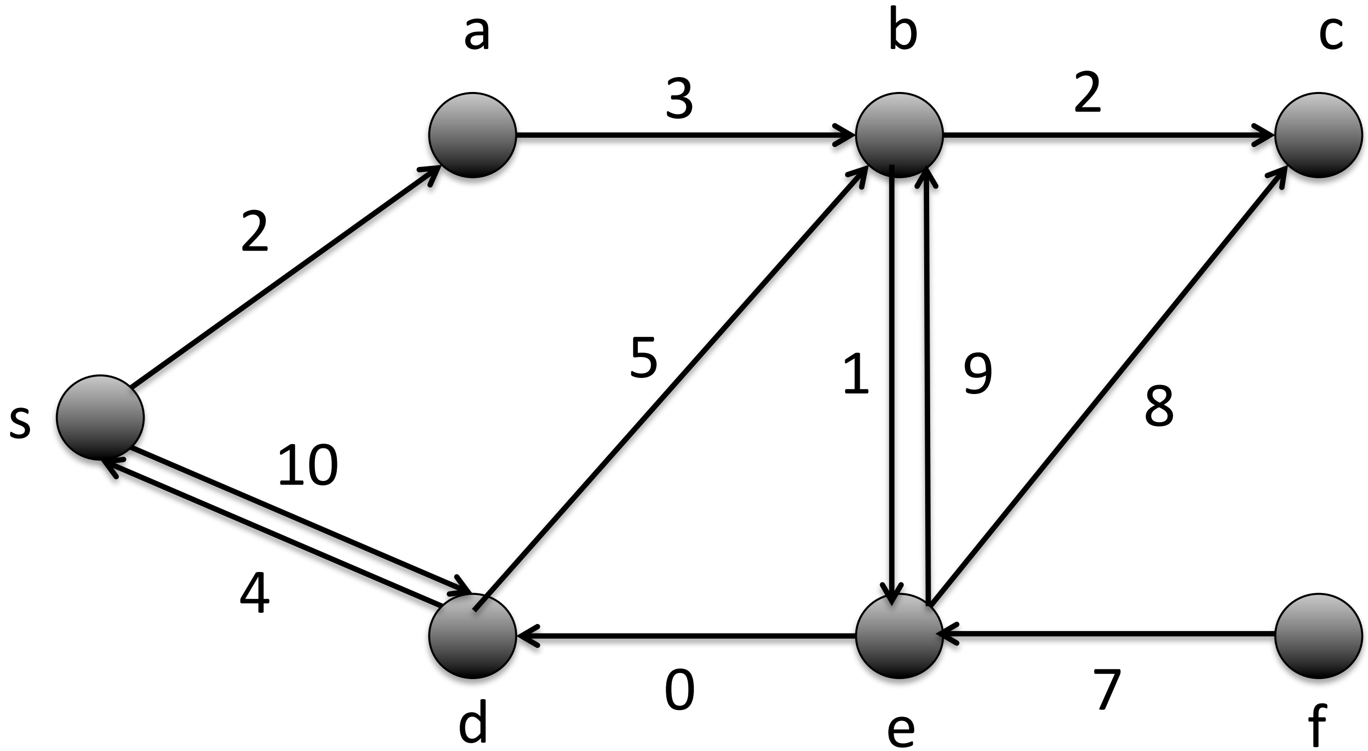
Problem Statement

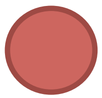
Single-source shortest path problem:

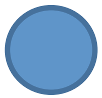
Compute for a given node s of V a shortest path to any other node in V (if it exists).

We assume that edge weights are non-negative.

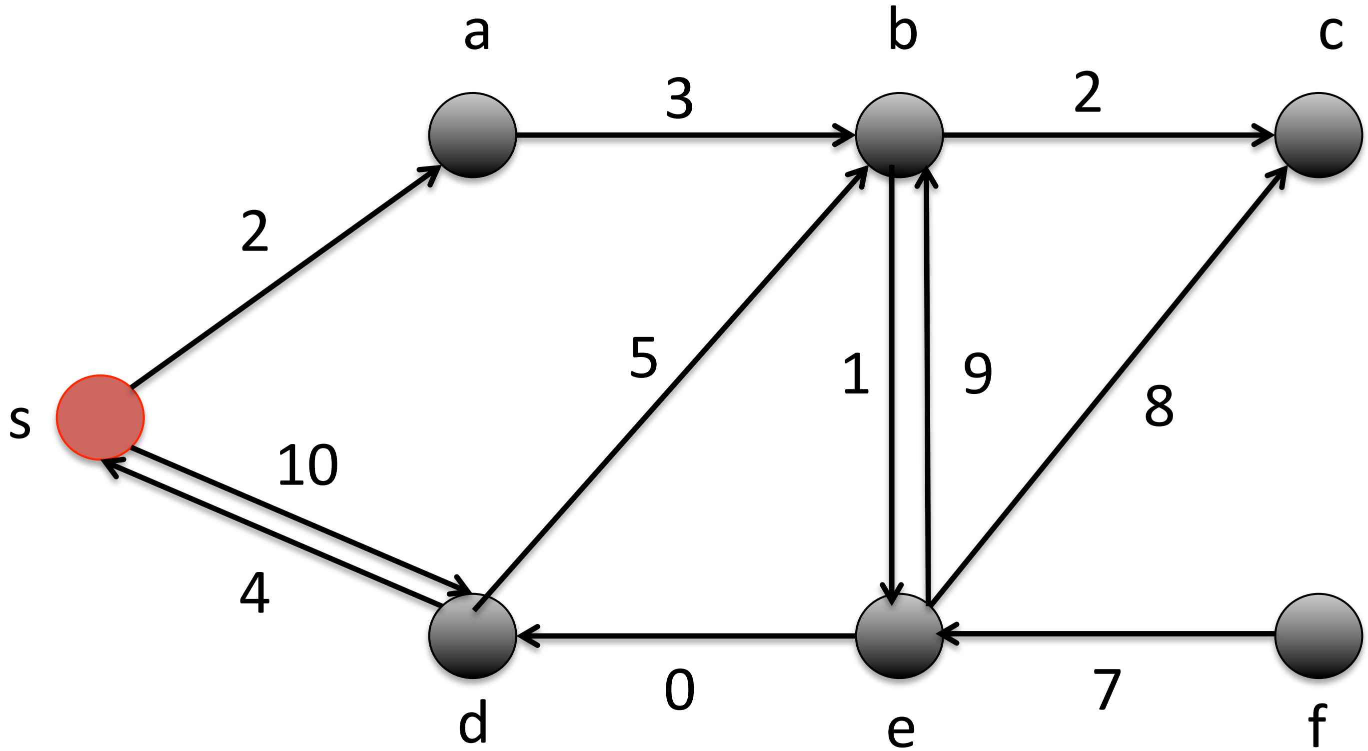
Example



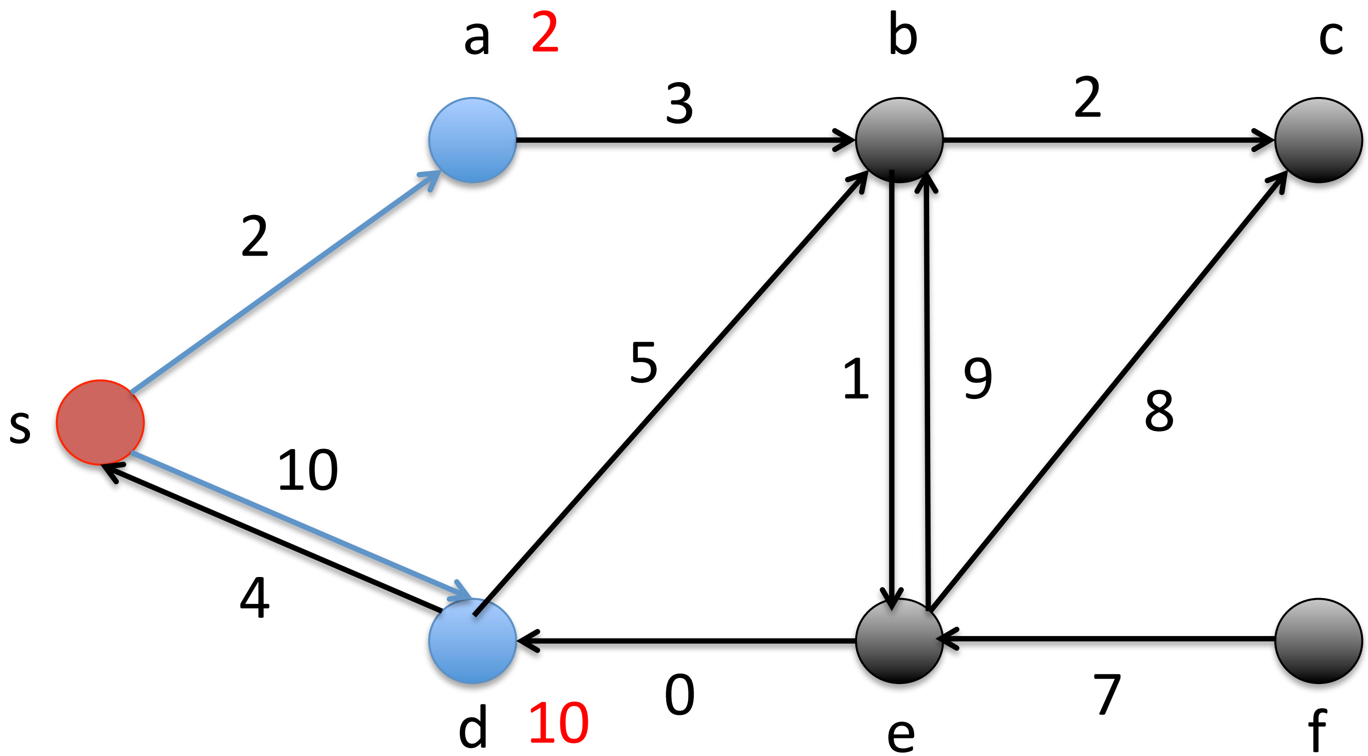

Nodes for which a shortest path has been computed


Nodes that have already been reached

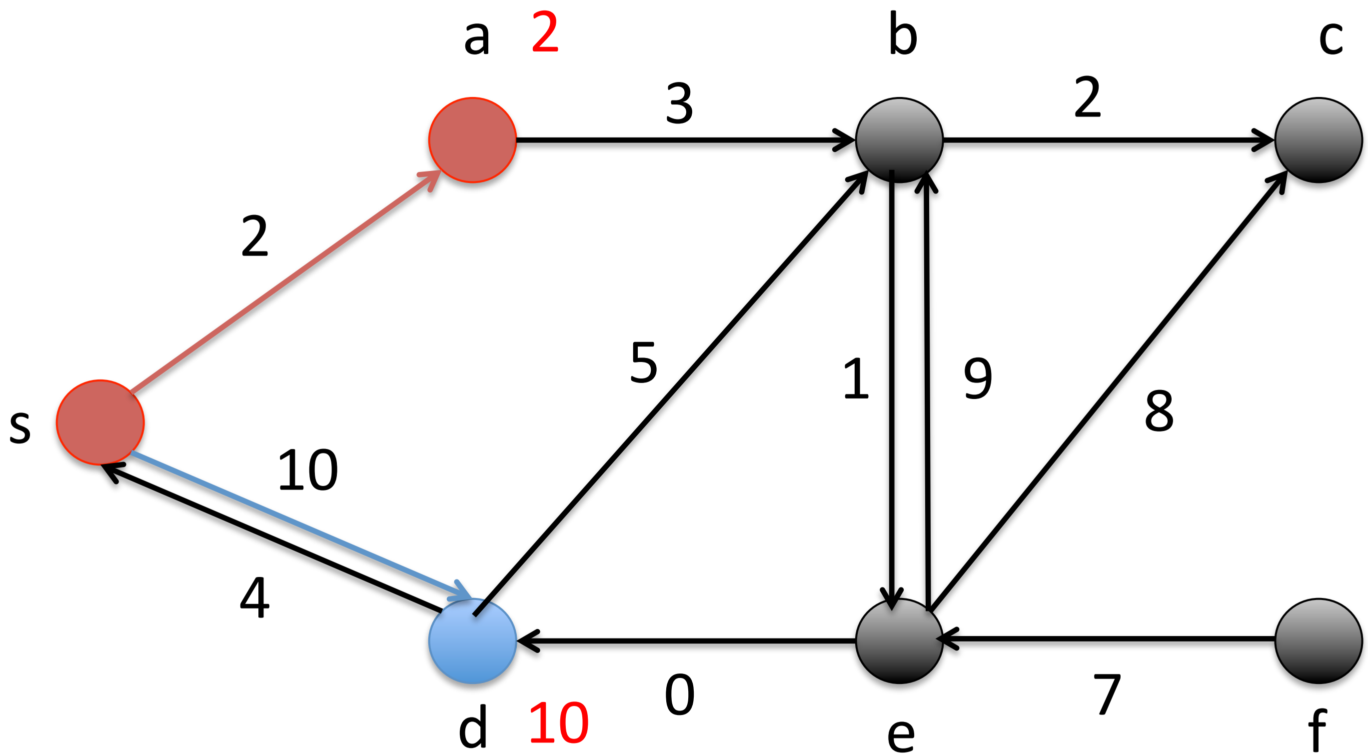
Example



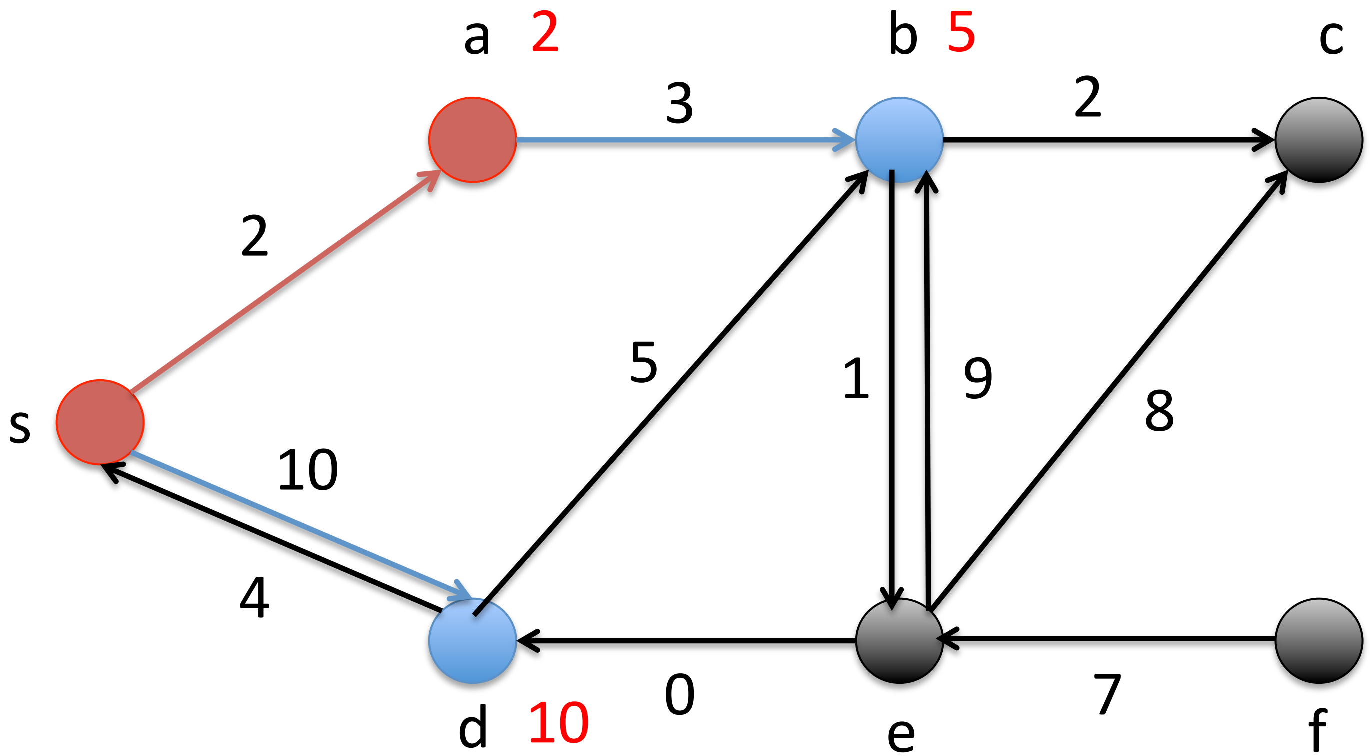
Example



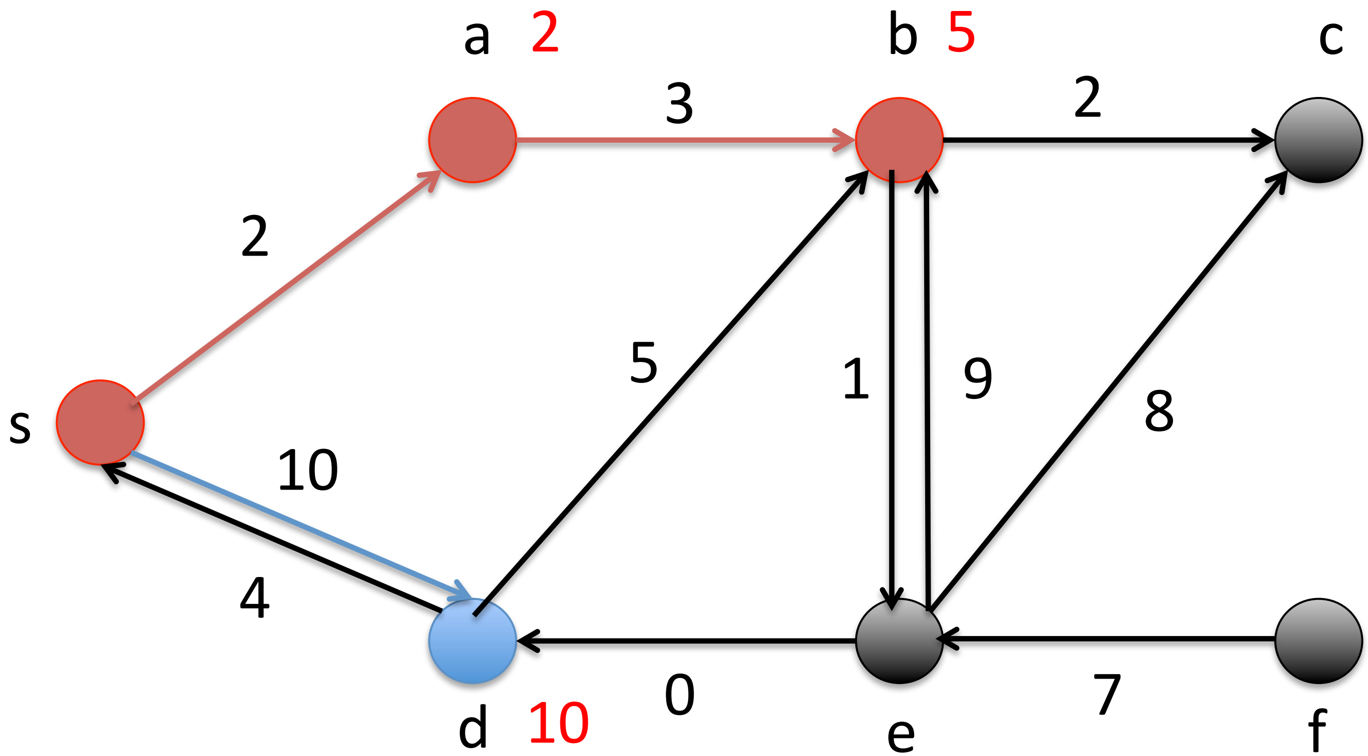
Example



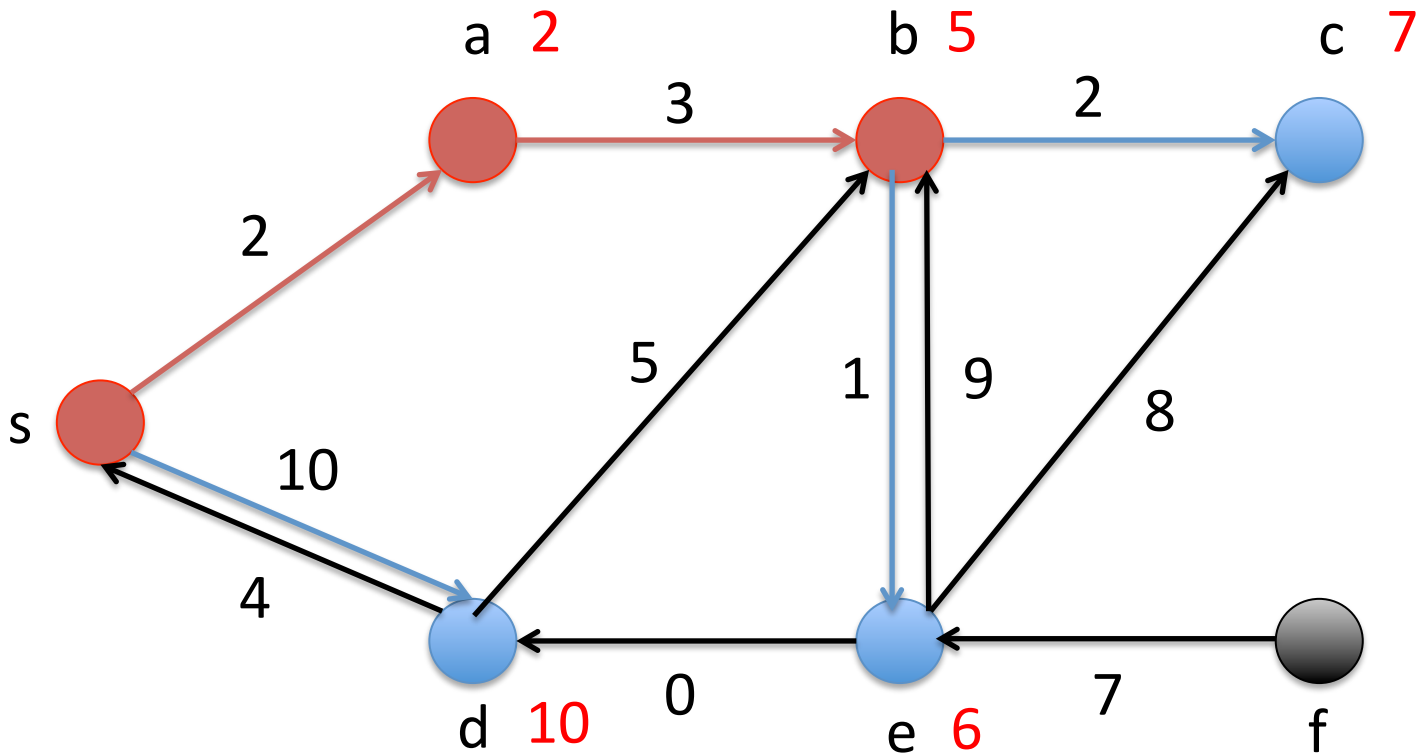
Example



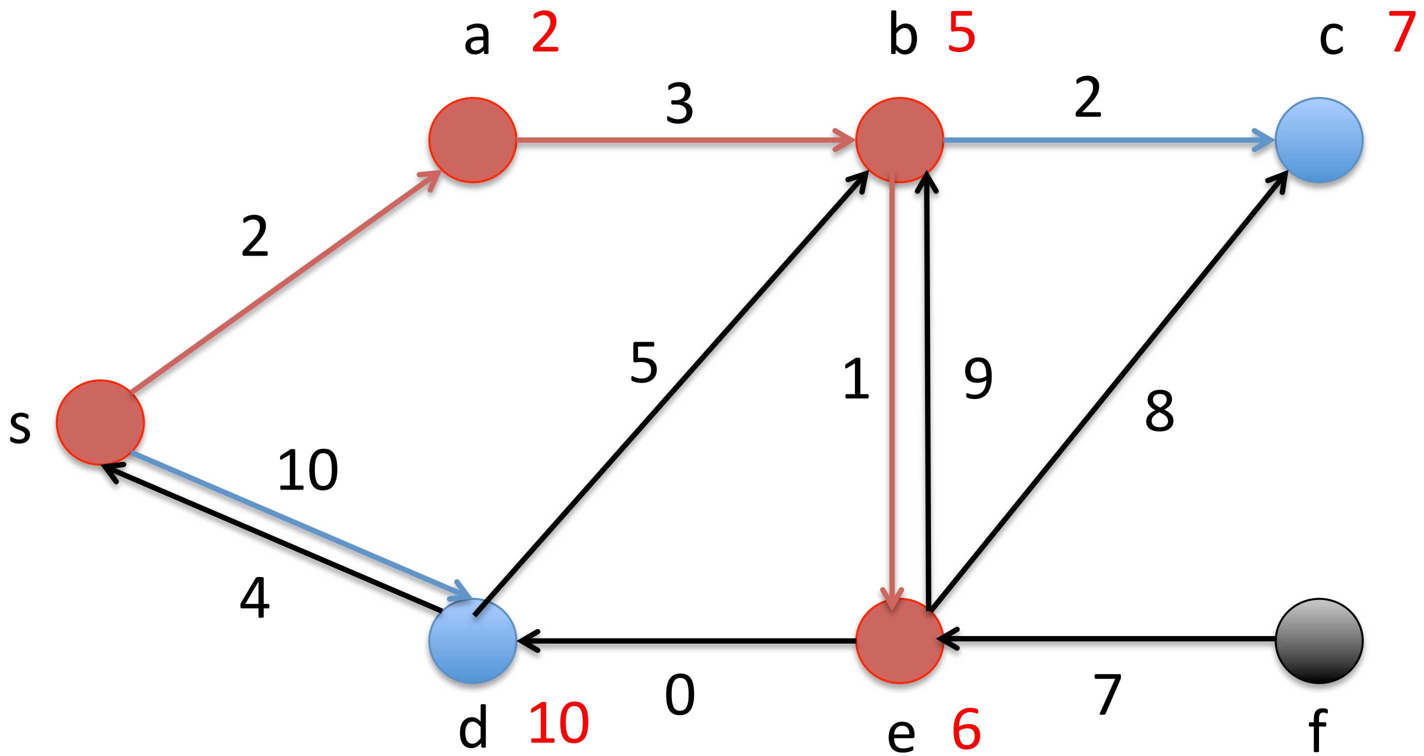
Example



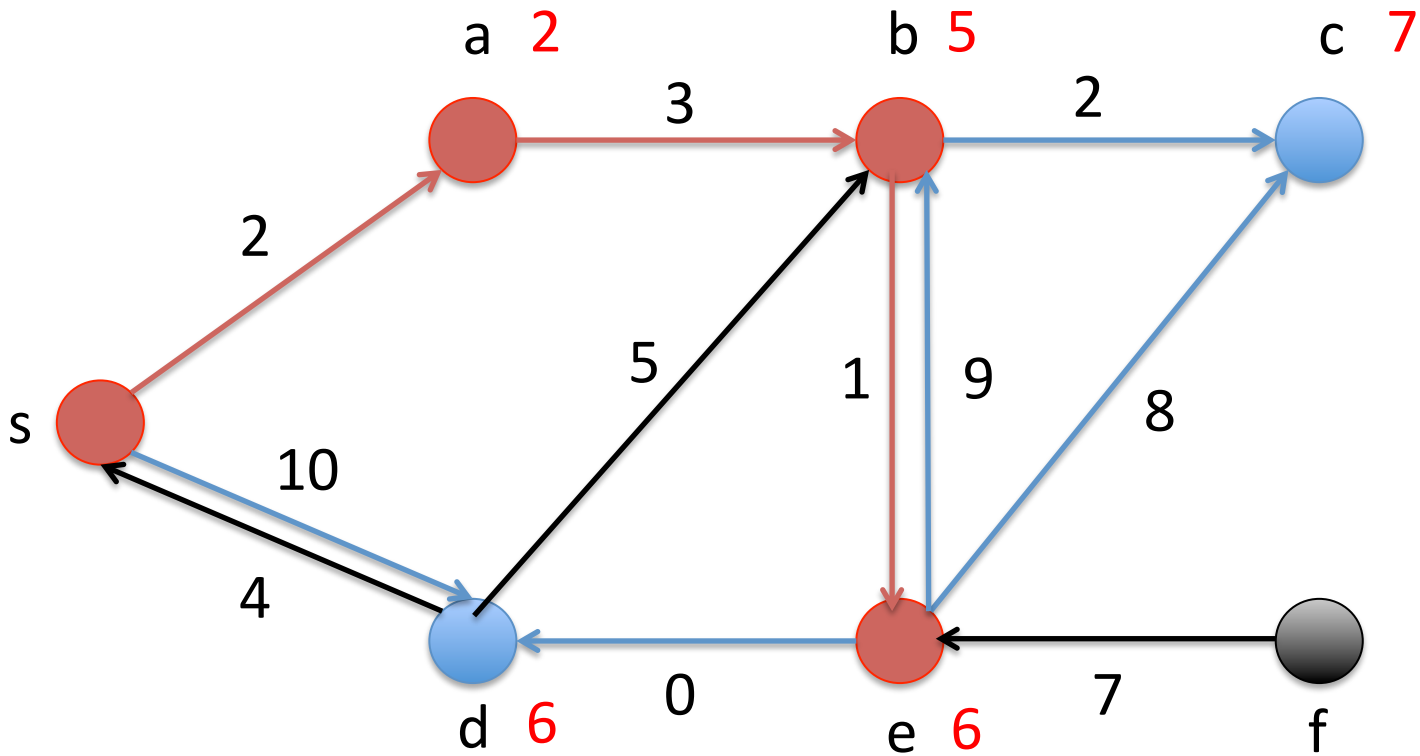
Example



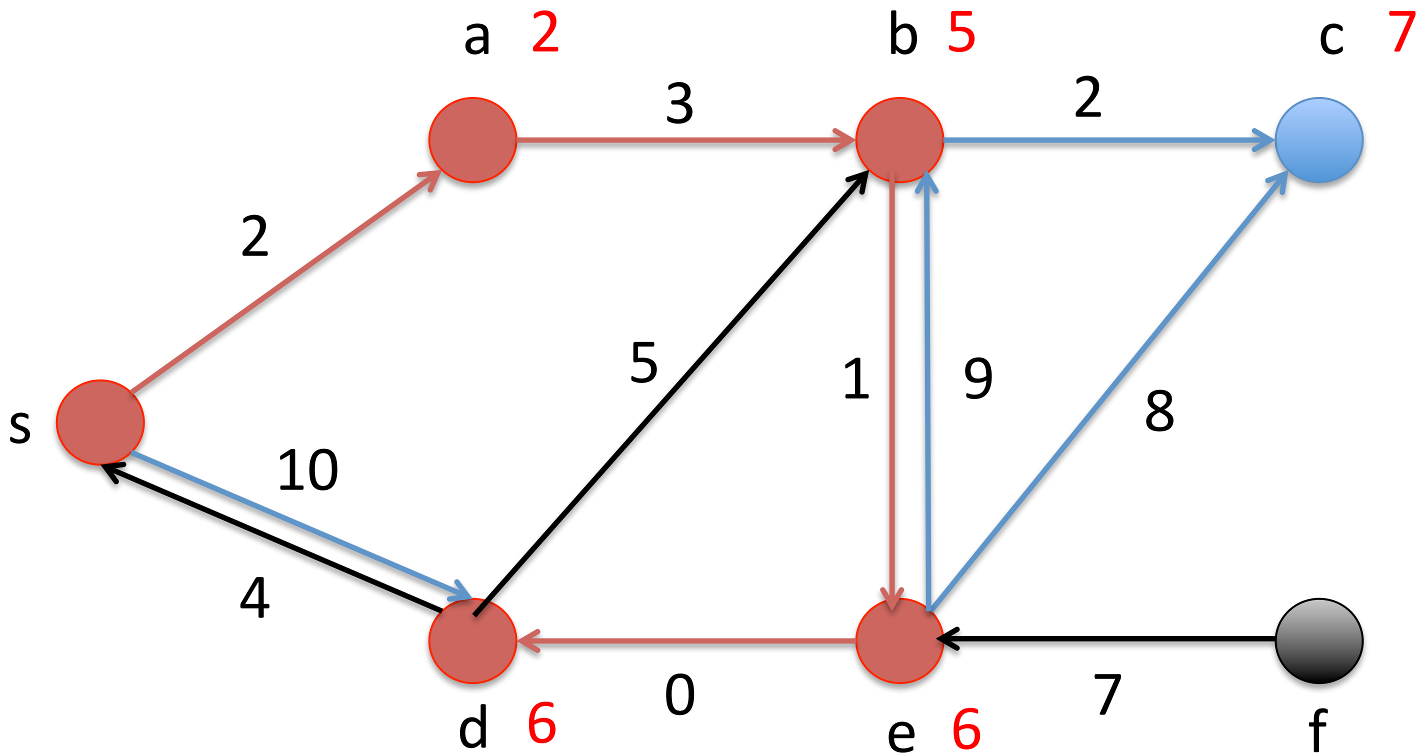
Example



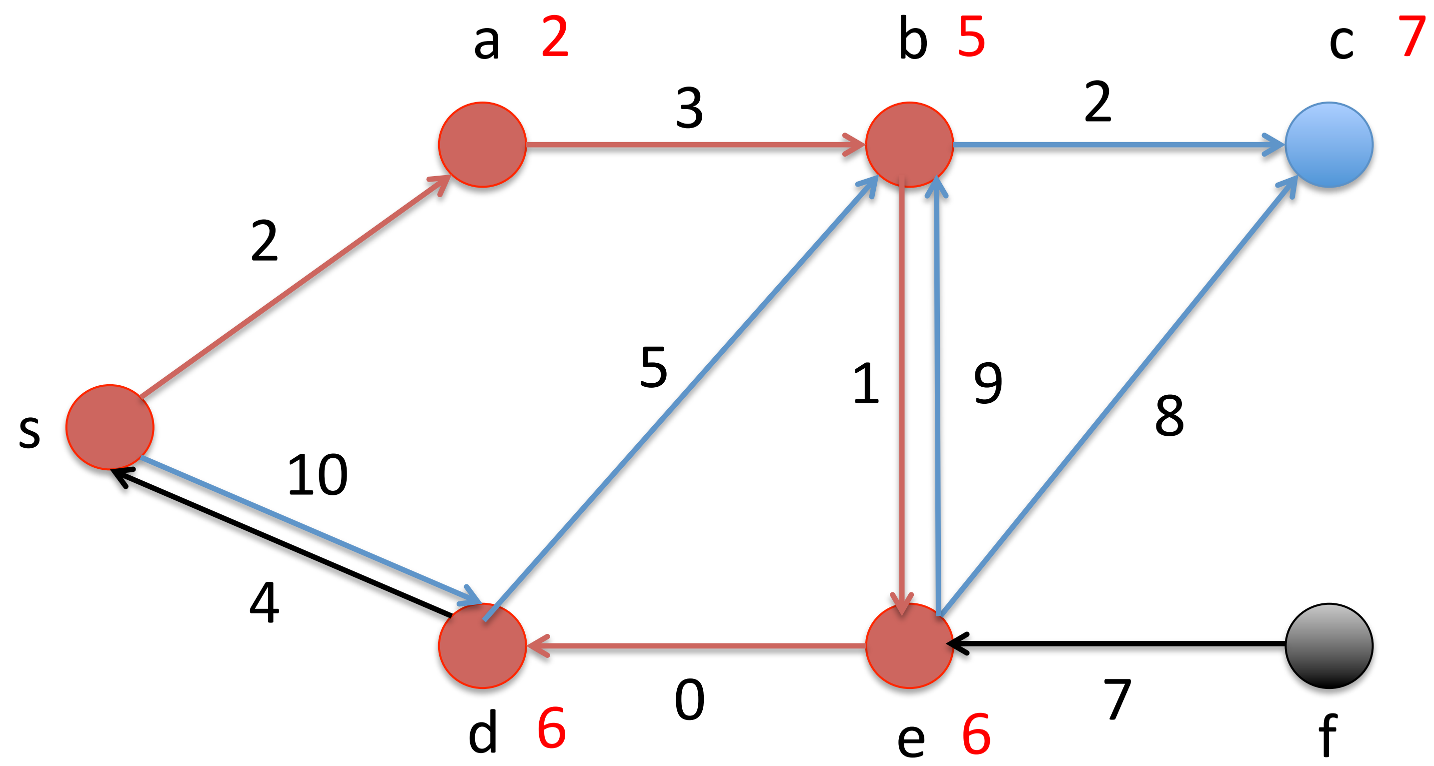
Example



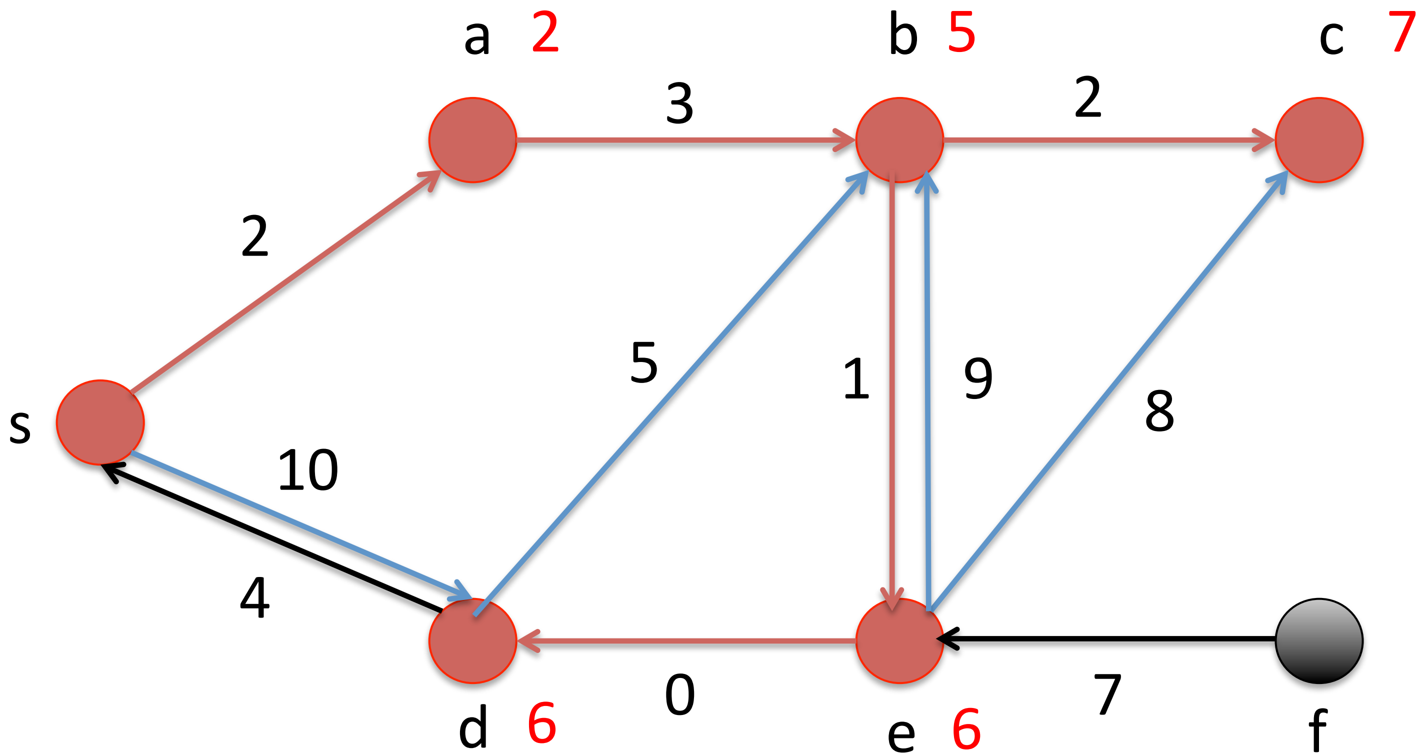
Example



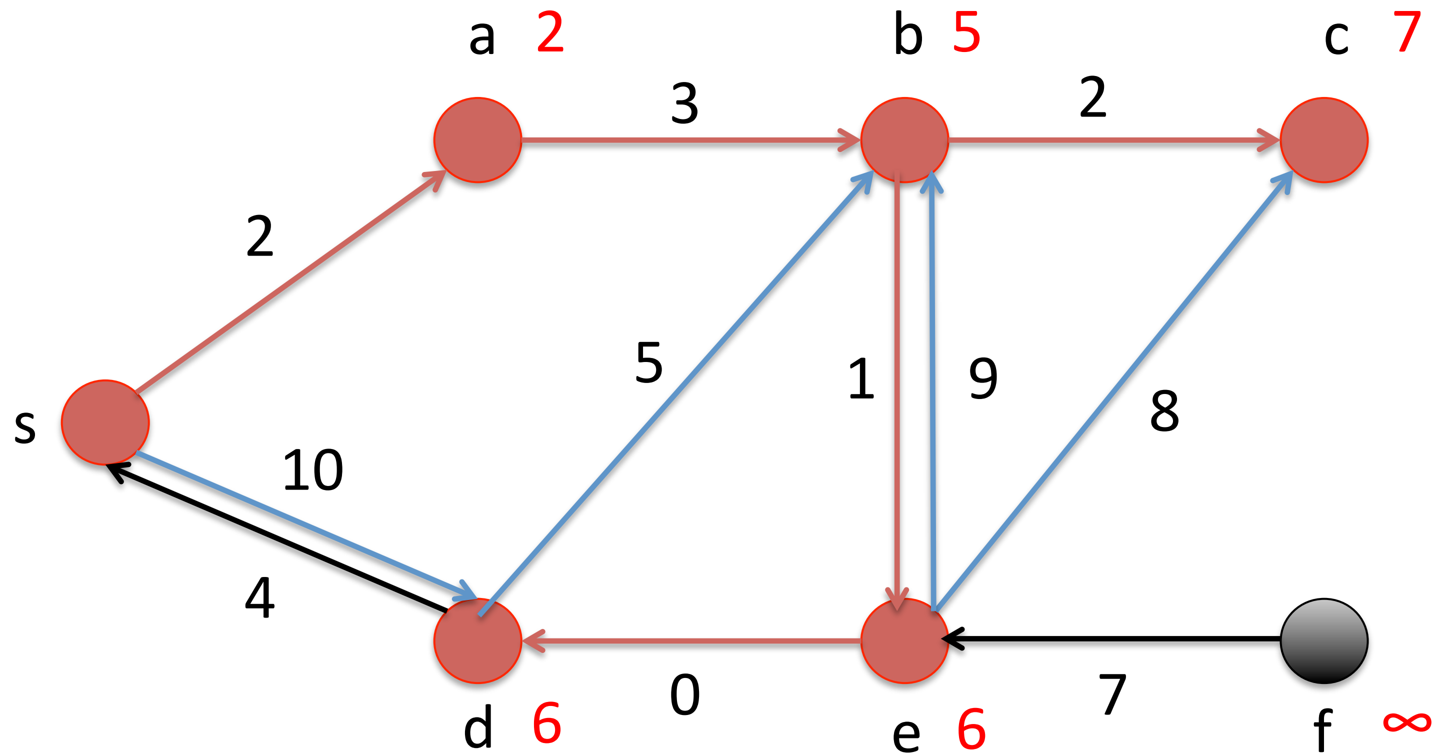
Example



Example



Example



Why non-negative edge costs?

If a path from s to v can contains a negative cycles then a shortest path does not exist (is not defined).

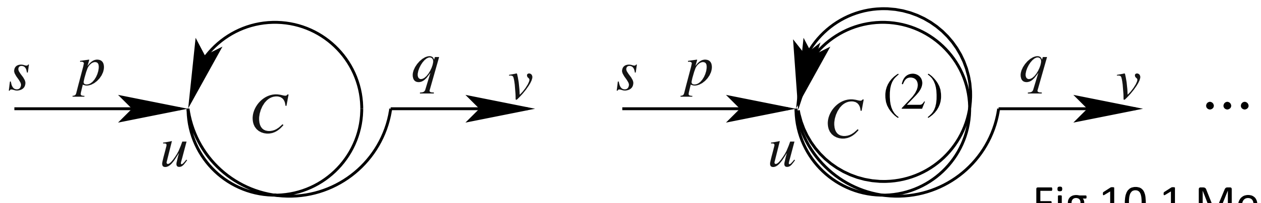


Fig 10.1 Mehlhorn/Sanders

Simple shortest path for non-negative edge costs

If edge costs are non-negative and v is reachable from s then a shortest path P from s to v exists. P can be chosen to be simple (cycle-free).

Properties of subpaths

Lemma: Subpaths of a shortest path are also shortest paths.
Proof (by contradiction):

- Assume that the path P is a shortest path from s to v.
- Assume that a subpath from a to b is not a shortest path from a to b

- This implies that there is a shorter path from a to b
- We can use this path to obtain a shorter path from s to v.
- Contradiction to P is shortest path from s to v.

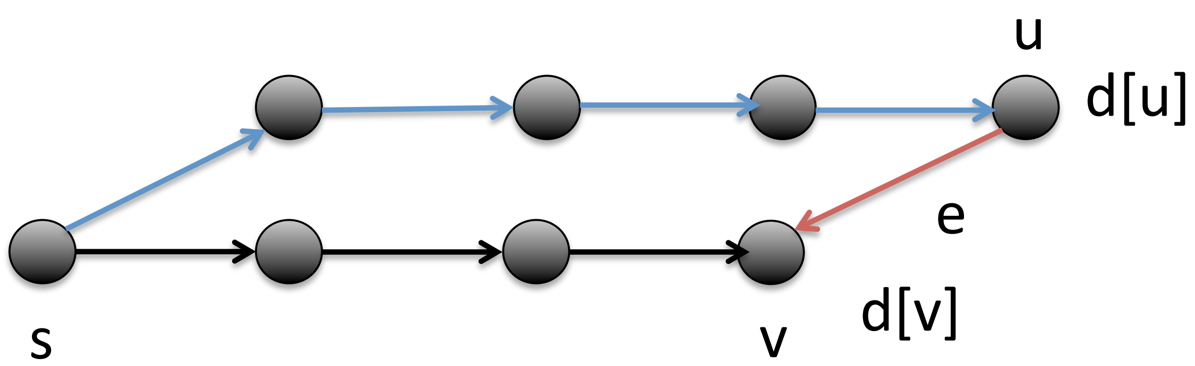
□

Dijkstra's Algorithm

- Remember BFS for computing all shortest paths in an unweighted graph.
- In iteration i, we computed all shortest paths having i edges.
- Dijkstra's algorithm obtains in iteration i a shortest path to the node of the ith smallest distance from s.
- We can represent all shortest paths from a node s by a tree rooted at s.

Updating

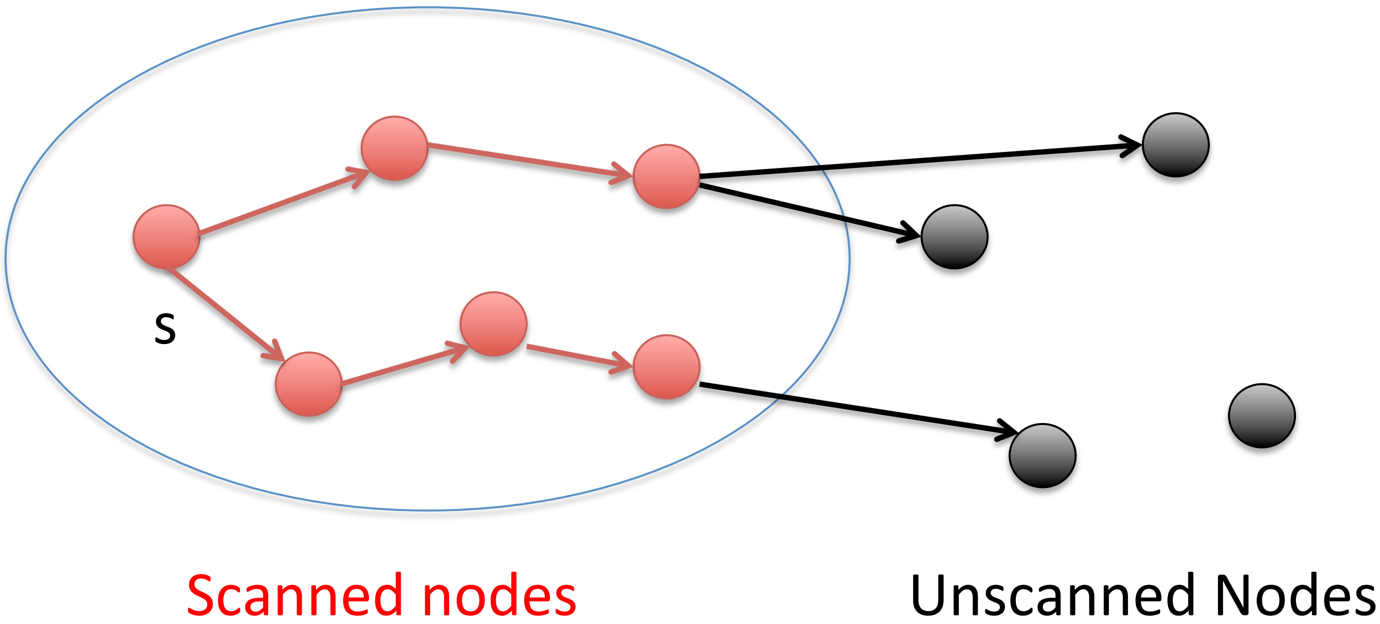
We may to update a previous path from s to v if we find a shorter path



Procedure $relax(e = (u,v) : Edge)$
  if $d[u] + c(e) < d[v]$ **then** $d[v] := d[u] + c(e); \quad parent[v] := u$

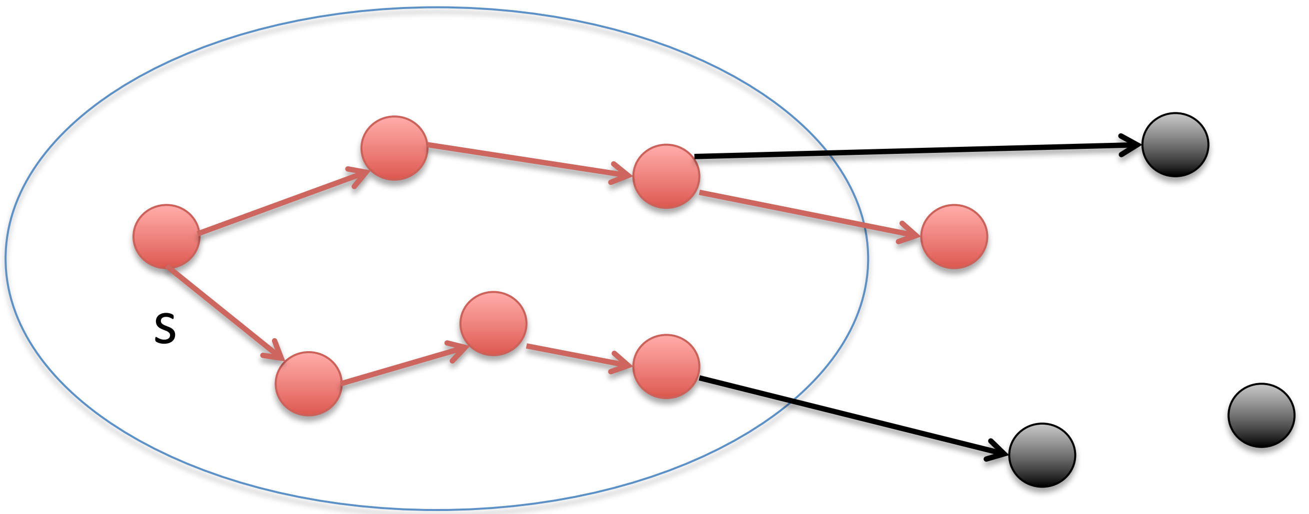
Dijkstras Algorithm

We call a node u unscanned if no shortest path from s to u has been found so far



Dijkstras Algorithm

Make unscanned node u scanned that would get the minimal tentative distance among all unscanned nodes.

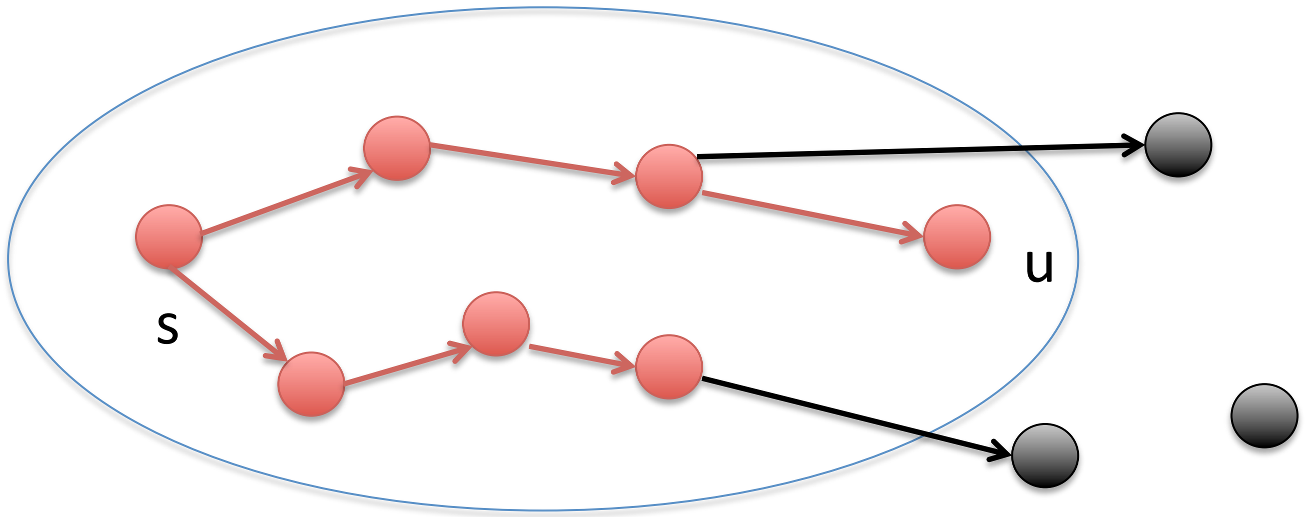


Scanned nodes

Unscanned Nodes

Dijkstras Algorithm

Make unscanned node u scanned that would get the minimal tentative distance among all unscanned nodes.

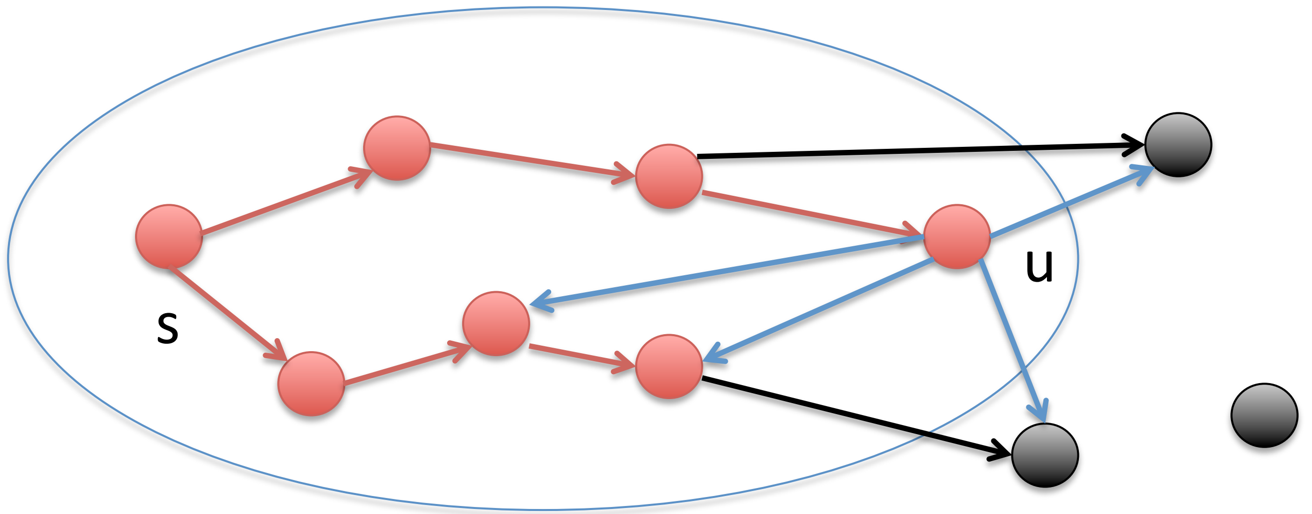


Scanned nodes

Unscanned Nodes

Dijkstras Algorithm

Consider all edges leaving u and update distances using relax.



Scanned nodes

Unscanned Nodes

Dijkstras Algorithm

Dijkstra's Algorithm
declare all nodes unscanned and initialize d and *parent*
while there is an unscanned node with tentative distance $< +\infty$ **do**

  $u :=$ the unscanned node with minimal tentative distance
  relax all edges (u, v) out of u and declare u scanned

Correctness

Theorem 10.5: Dijkstra's algorithm solves the single-source shortest path problem for graphs with nonnegative edge costs.

Proof:

We show two steps:

- All nodes reachable from s are scanned after termination.
- When a node v becomes scanned then the shortest path from s to v is obtained.

Correctness

Claim: All nodes reachable from s are scanned after termination.

Proof (by contradiction):

- Assume that there is a node v reachable from s, but never scanned.
- Consider a shortest path $p=(s=v_1, v_2, \ldots, v_k=v)$ from s to v
- Let $i>1$ be minimal such that v_i is unscanned.
- Implies node v_{i-1} has been scanned.
- When v_{i-1} is scanned $d[v_i]$ is set to $d[v_{i-1}]+c(v_{i-1},v_i) < \infty$.
- Hence, v_i must be scanned as only nodes u with $d[u]= \infty$ stay unscanned. Contraction to v_i is unscanned.

$\square$

Correctness

Claim: When a node v becomes scanned then the shortest path from s to v is obtained.

Proof (by contradiction):

- Denote by $\mu[v]$ the length of a shortest path from s to v.
- Consider the first point in time t when v has been scanned and $d[v] > \mu[v]$ holds.
- Consider a shortest path $p=(s=v_1, v_2, \ldots, v_k=v)$ from s to v.
- Let $i>1$ be minimal such that v_i has not been scanned before time t.

Correctness

Proof (continued):

- Node v_{i-1} was scanned before time t which implies $\mu[v_{i-1}]=d[v_{i-1}]$.
- When v_{i-1} is scanned $d[v_i]$ is set to $d[v_{i-1}]+c(v_{i-1},v_i)=\mu[v_{i-1}]+c(v_{i-1},v_i)$.
- We have $d[v_i]=\mu[v_i]\leq= \mu[v_k]< d[v_k]$ and hence v_i is scanned instead of v_k, a contradiction.

$\square$

Implementation

- Store all unscanned reached nodes in an addressable priority queue Q (using tentative distances as key values)

Pseudocode Dijkstra

Function $Dijkstra(s : NodeId) : NodeArray \times NodeArray$ // returns $(d, parent)$

  $d = \langle \infty, \ldots, \infty \rangle$: $NodeArray$ **of** $\mathbb{R} \cup \{\infty\}$ // tentative distance from root

  $parent = \langle \perp, \ldots, \perp \rangle$: $NodeArray$ **of** $NodeId$

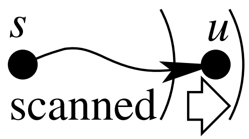
  $parent[s] := s$ // self-loop signals root

  $Q : NodePQ$ // unscanned reached nodes

  $d[s] := 0$; $Q.insert(s)$

  while $Q \neq \emptyset$ **do**

    $u := Q.deleteMin$ // we have $d[u] = \mu(u)$



    foreach $edge\ e = (u, v) \in E$ **do**

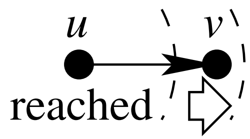
      if $d[u] + c(e) < d[v]$ **then** // relax

        $d[v] := d[u] + c(e)$

        $parent[v] := u$ // update tree

        if $v \in Q$ **then** $Q.decreaseKey(v)$

        else $Q.insert(v)$



  return $(d, parent)$

Fig 10.6 Mehlhorn/Sanders

Runtime

- Initialization (arrays, priority queue) takes time O(n).
- Every reachable note is inserted and removed once from Q.
- At most n deleteMin and insert operations.
- Each node is scanned at most once and each edge is relaxed at most once.
- Implies at most m decreaseKey operations.

Total runtime

$$T_{\text{Dijkstra}} = \mathrm{O}\left(m \cdot T_{decreaseKey}(n) + n \cdot (T_{deleteMin}(n) + T_{insert}(n))\right)$$

Runtime

Runtime depends on implementation of priority queue.

Original (Dijkstra 1959):

- Maintain the number of reached unscanned nodes.
- An array d storing the distances and an array storing for each node whether it is reached or unscanned.
- Insert and decreaseKey take time O(1)
- DeleteMin takes time O(n)
- Total Runtime: $O(m+n^2)$

Improvements:

- Binary Heaps: O((m+n) log n)
- Fibonacci Heaps: O(m + n log n)