Single Source Shortest Path

Dr. Gur Saran Adhar

Reference clrs, Chapter 24, Page 580-

Introduction to Problem: Single Source Shortest Path

`Reference:  clrs, page-580`

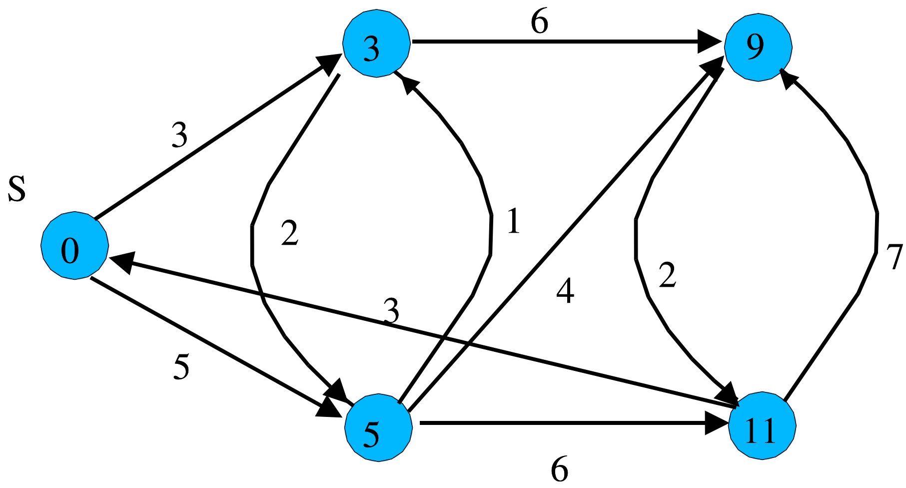
Given a **weighted, directed graph** $G = (V, E)$, with weight function $w : E \rightarrow R$ mapping edges to real-values weights. The **weight** of a path $p = \langle v_0, v_1, \ldots v_k \rangle$ is the sum of the weights of edges in the path

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

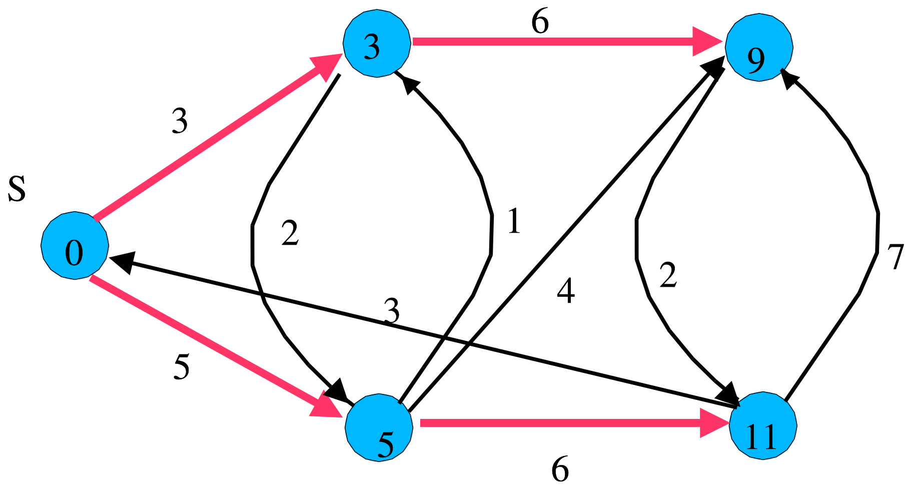
The **shortest-path weight** from u to v is defined as

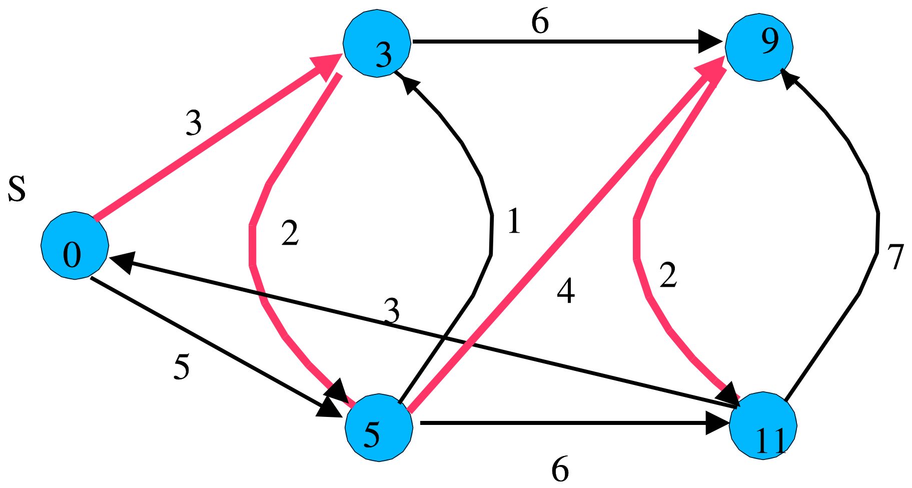
$$\delta(u,v) = \begin{cases} \text{min}\{w(p) : u \rightarrow v\} & \text{if there is a path } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

A **shortest-path** from u to v is then defined as any path p with weight $w(p) = \delta(u, v)$



S
0
3
5
9
11
3
5
6
2
1
4
3
6
2
7


S
0
3
5
9
11
3
5
6
6
2
1
4
3
2
7


S
0
3
5
9
11
3
5
2
1
6
4
3
6
2
7

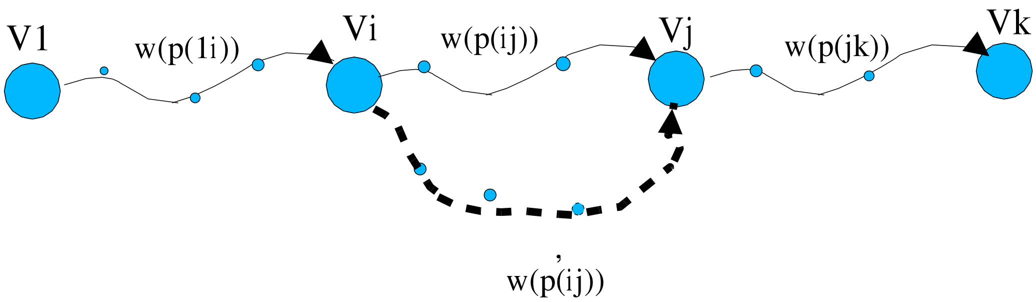
Variants of the Problem:

1. **Single-destination shortest-paths problem**: Find shortest path to a given **destination** vertex t from each vertex v. By reversing the direction of each edge, we can reduce it to a single source problem
2. **Single-pair shortest-path problem**: Find a shortest path from u to v for given vertices u and v.
3. **All-pair shortest-path problem**: Find a shortest path from u to v for every pair of vertices u and v (chapter-25).

Optimal Substructure

Property: The shortest path between two vertices contains other shortest path within it.

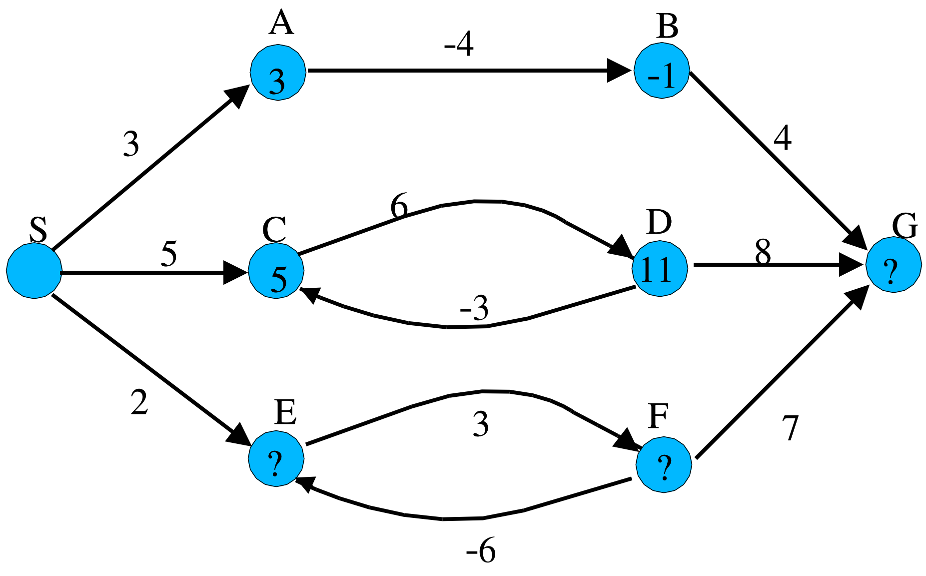
Sketch of the proof: Decompose the shortest path $p =< v_1, v_2, \ldots, v_k >$ into $p_{1i} = v_1 \rightarrow v_i$, $p_{ij} = v_i \rightarrow v_j$ and $p_{jk} = v_j \rightarrow v_k$. The weight of the shortest path $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. Now if there is a shorter path $p^{'}_{ij}$ between i and j. That is, $w(p^{'}_{ij}) < w(p_{ij})$ then there is shorter path with weight $w(p_{1i}) + w(p^{'}_{ij}) + w(p_{jk})$ than the original shortest path p. Which means the p could not have been shortest.


V1
w(p(1i))
Vi
w(p(ij))
Vj
w(p(jk))
Vk
w(p'(ij))

Negative Weight Edges

Can shortest path contain negative cycles?

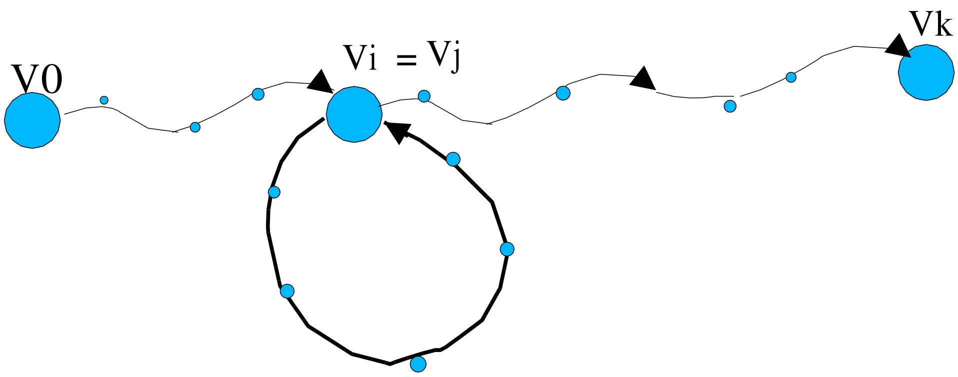
If there are no **negative edge cycles** reachable from the source s, the shortest path weight $\delta(s, v)$ remains well defined for all vertices v, even if there may be edges with negative weights.
If there is a negative weight cycle reachable from s, shortest path weights are not well defined. No path from s to a vertex on the cycle can be a shortest path.
A "lesser-weight" path can always be found.


A
B
-4
3
-1
4
3
S
6
D
C
G
5
8
5
11
?
-3
2
E
3
F
7
?
?
-6

Cycles

Can a shortest path contain any cycle?

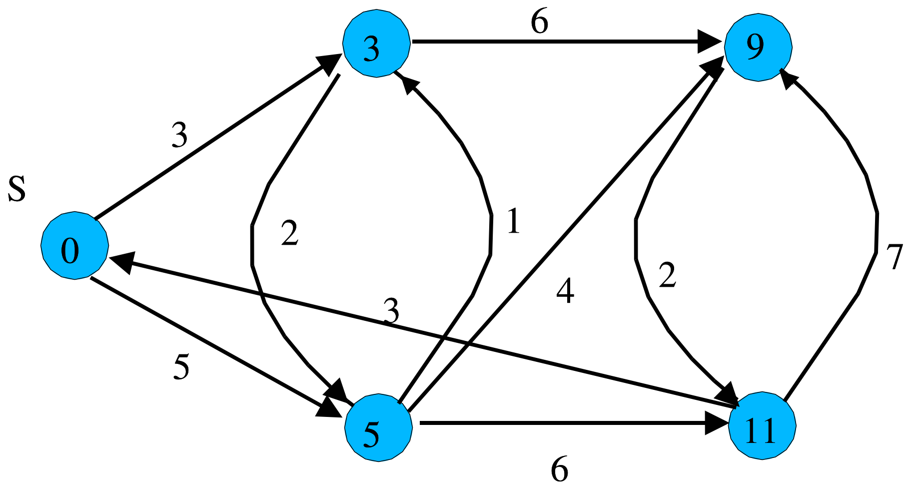
Sketch of the proof: Suppose p is a shortest path $p =< v_0, v_1, \ldots, v_k >$ and c is a positive weight cycle $c =< v_i, v_{i+1}, \ldots v_j >$ (so that $v_i = v_j$ and $w(c) > 0$), then another path $p^{'} =< v_0, v_i, v_{j+1}, \ldots v_k >$ (the path p minus the cycle c) has weight $w(p^{'}) = w(p) - w(c)$ which is less then $w(p)$ and therefore p could not have been the shortest path.

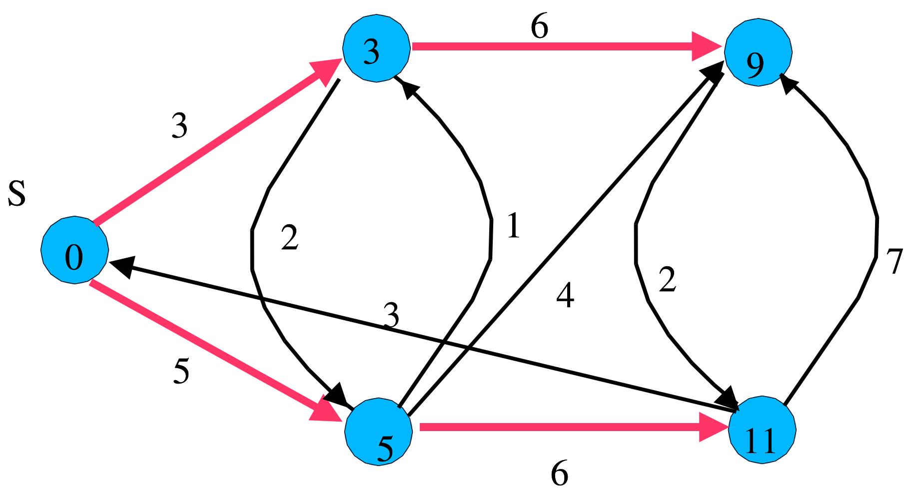

V0
Vi = Vj
Vk

Representation Of shortest path

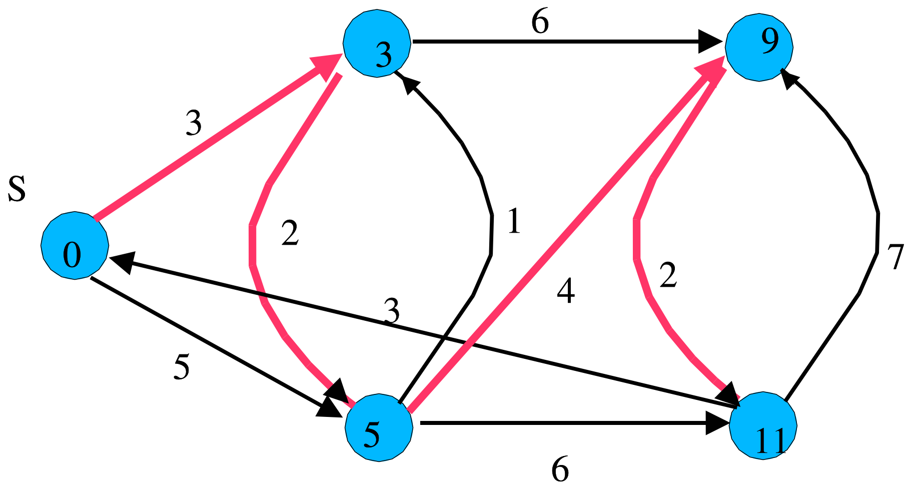
Since no cycles are permissible single-source shortest paths are represented by **shortest path trees** rooted at s.

`Note: shortest paths and shortest path trees are not unique`


S
0
3
5
9
11
3
5
6
2
1
4
3
6
2
7



S
0
3
5
9
11
3
5
6
2
1
4
3
2
7
6



S
0
3
9
5
11
3
6
2
1
4
2
7
3
5
6

INITIALIZE-SINGLE-SOURCE(G, s)

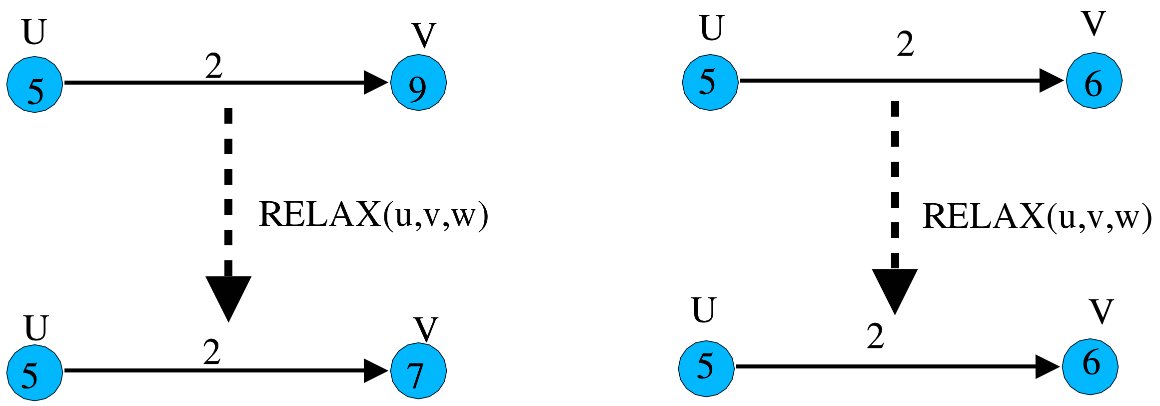
```
for each vertex v ∈ V(G)
    do d[v] = ∞
       π[v] = NIL
d[s] ← 0
```

Reference clrs 586

RELAX(u, v, w)

if $d[v] > d[u] + w(u, v)$
then $d[v] = d[u] + w(u, v)$
$\pi[v] = u$ { set u the predecessor of v }

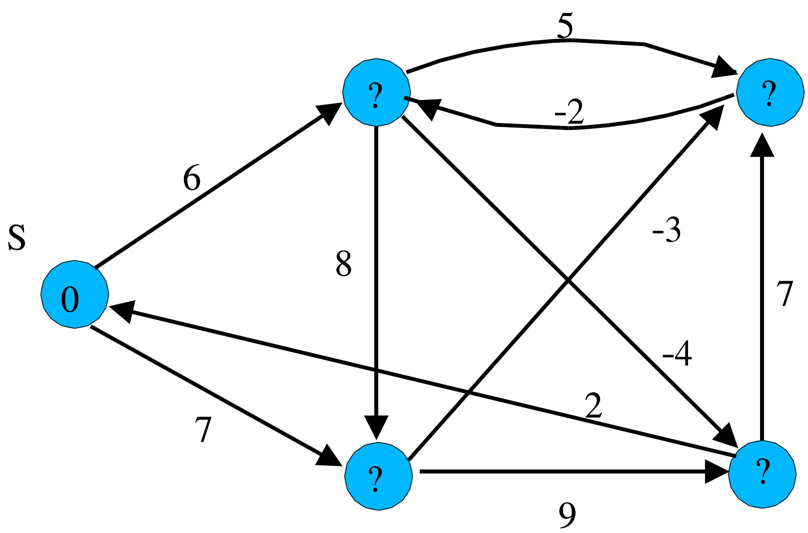
`Reference clrs 586`

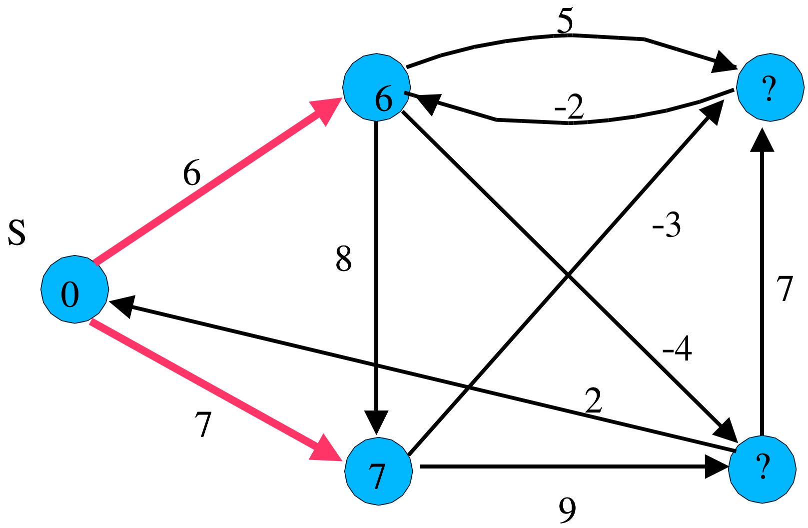

U
V
2
5
9
RELAX(u,v,w)
U
V
2
5
7
U
V
2
5
6
RELAX(u,v,w)
U
V
2
5
6

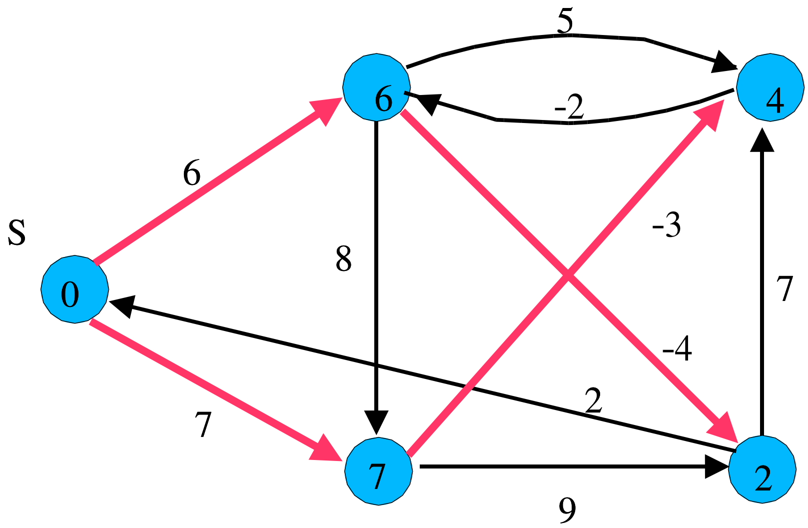
BELLMAN-FORD(G, w, s)

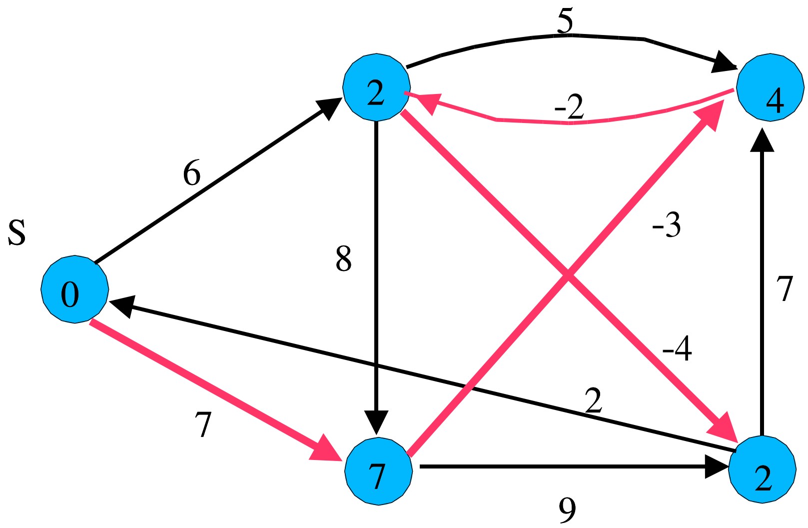
INITIALIZE-SINGLE-SOURCE(G, s)
for $i = 1$ to $\mid V(G) \mid -1$
    do for each edge $(u, v) \in E(G)$
        do RELAX(u, v, w)
for each edge $(u, v) \in E(G)$
    do if $d[v] > d[u] + w(u, v)$
        then return FALSE
return TRUE

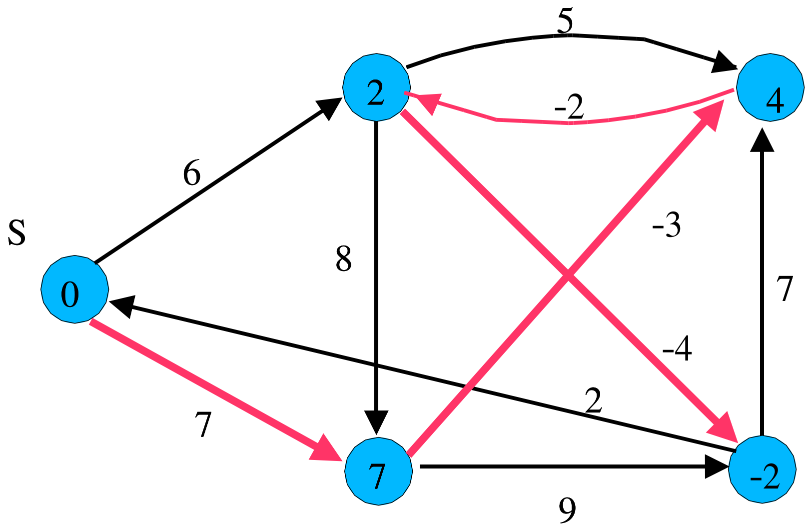
Reference clrs 588


5
-2
6
S
0
8
-3
7
-4
2
7
9
?
?
?
?


S
0
6
7
?
?
6
7
5
-2
8
-3
-4
7
2
9


5
-2
6
4
6
S
0
8
-3
7
-4
2
7
7
9
2


5
2
-2
4
6
S
0
8
-3
7
-4
2
7
7
2
9


5
2
-2
4
6
S
-3
8
7
0
-4
2
7
7
-2
9

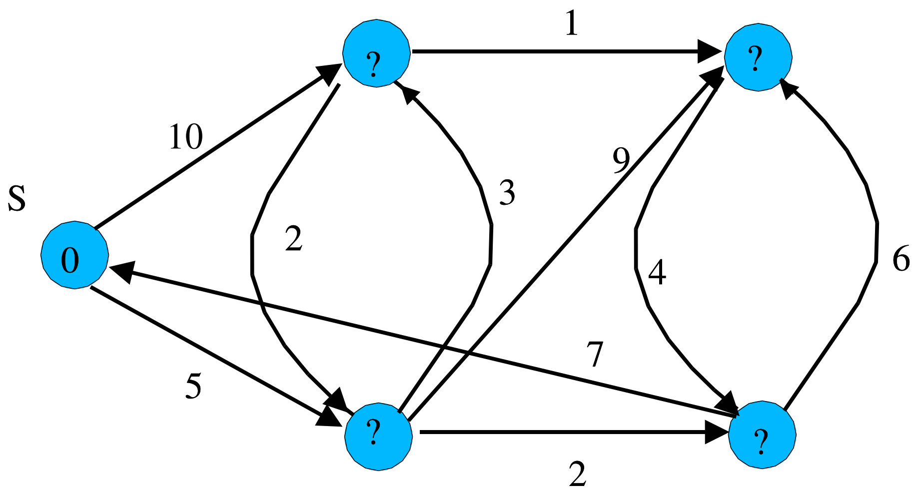
Dijkstra Algorithm: Main Idea

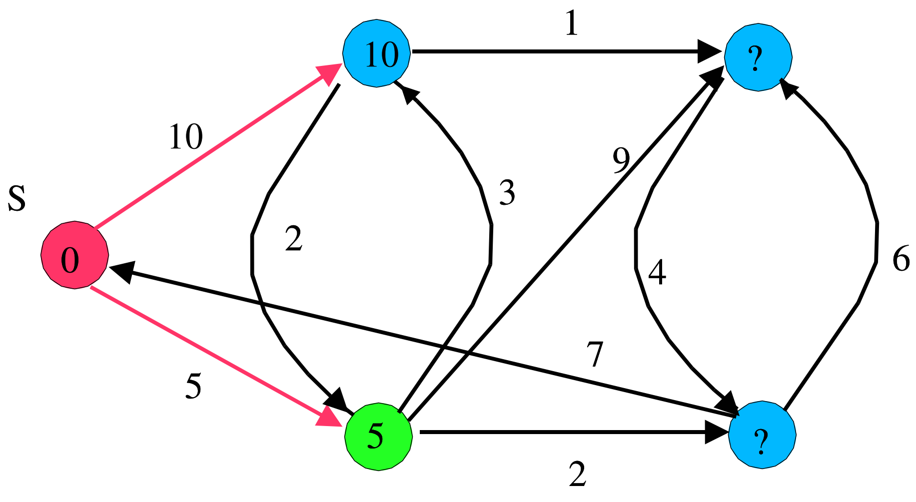
- Dijkstra's Algorithm solves single-source shortest paths problem on a weighted directed graph in which all edge weights are non-negative.
- Dijkstra's Algorithm maintains a set S of vertices whose final shortest paths from the source s have already been determined.
- The algorithm repeatedly selects one vertex $u \in V - S$ with the minimum shortest path, adds u to S, and *relaxes* all edges leaving u.
- A min-priority queue Q of vertices, keyed by their d values is used to select u as the algorithm proceeds.

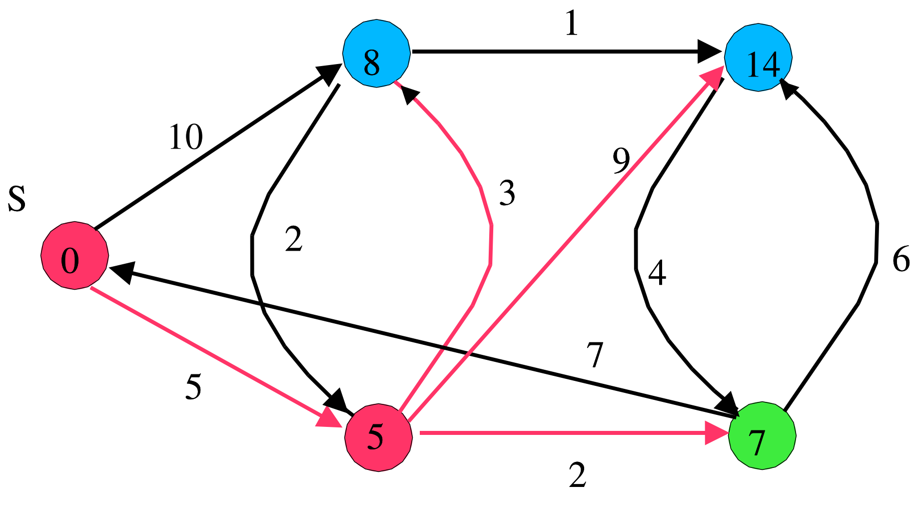
DIJKSTRA(G, w, s)

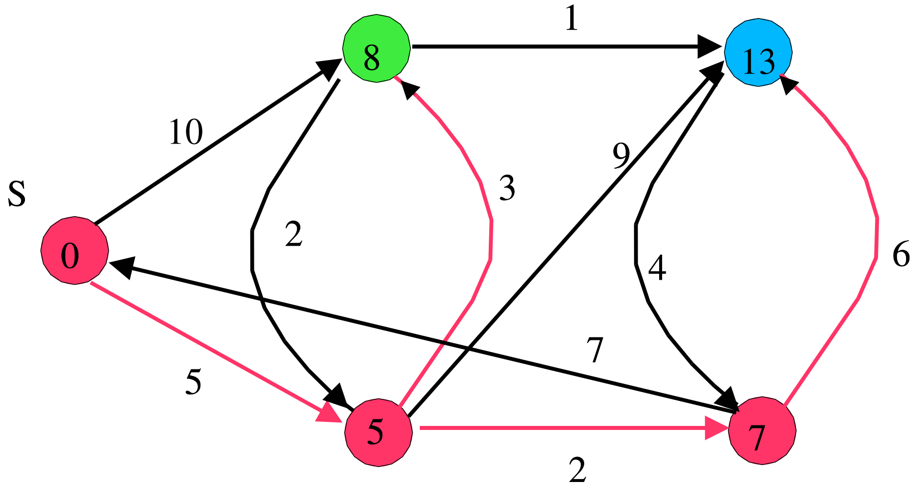
INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \phi$
$Q \leftarrow V(G)$
while $Q \neq \phi$
    do $u \leftarrow EXTRACT - MIN(Q)$
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do RELAX(u, v, w)

`Reference clrs 595`

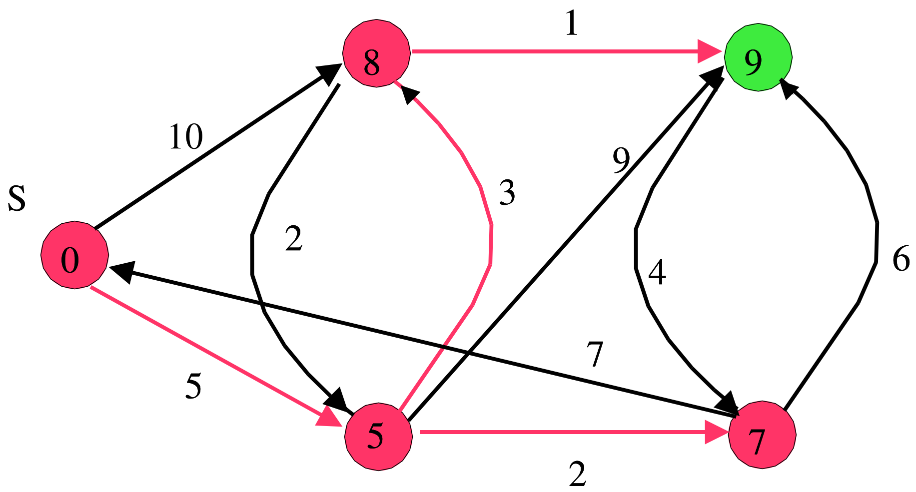

S
0
?
?
?
?
10
5
1
2
3
2
9
4
6
7
2


S
0
10
5
?
?
10
5
1
2
3
9
2
4
6
7

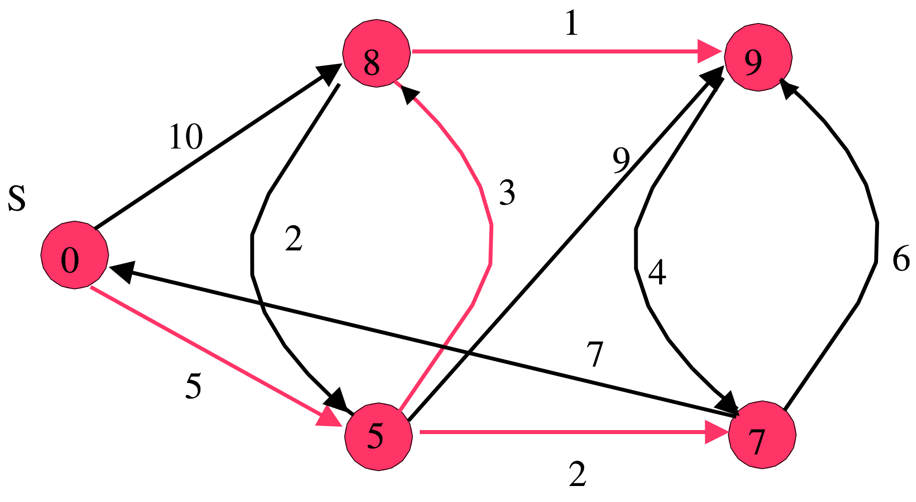

s
0
8
14
5
7
10
1
2
3
9
4
6
7
5
2



1
10
9
3
2
4
6
S
7
5
2
8
13
0
5
7



S
0
8
9
5
7
10
1
2
3
9
4
6
7
5
2


1
10
s
0
8
9
2
3
9
4
6
7
5
5
7
2

Arbitrage

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of currency into more than one unit of the same currency.

For example, suppose that 1 U.S. dollar buys 0.6 English Pounds; 1 English pound buys 120 Japanese Yen; and one Japanese Yen buys 0.014661 U.S. dollars. Then by converting currencies, a trader can turn a 1 U.S. dollar and 0.6*120*0.014661 = 1.055 U.S. dollar, thus turning a profit of 5.56 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots c_n$ and an nxn table of exchange rates R, such that one unit of currency i buys $R[i, j]$ units of currency j.

Give an algorithm to determine whether or not there exists a sequence of currencies $< c_{i_1}, c_{i_2}, \ldots, c_{i_k} >$ such that

$$R[i_1, i_2] * R[i_2, i_3] * \ldots * R[i_k, i_1] > 1$$

Difference Constraints

Solving a **system of difference constraints** when each constraint is a simple linear inequality of the form:

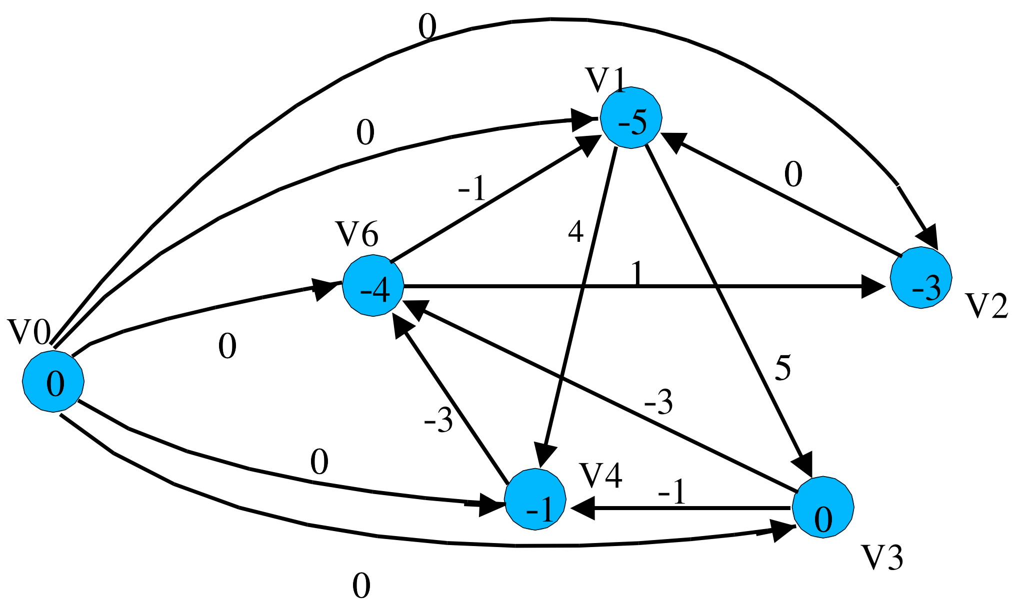
$$x_j - x_i \leq b_k$$

where $1 \leq i, j \leq n$ and $1 \leq k \leq m$

For example:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

$$x_1 - x_2 \leq 0, \tag{1}$$
$$x_1 - x_5 \leq -1, \tag{2}$$
$$x_2 - x_5 \leq 1, \tag{3}$$
$$x_3 - x_1 \leq 5, \tag{4}$$
$$x_4 - x_1 \leq 4, \tag{5}$$
$$x_4 - x_3 \leq -1, \tag{6}$$
$$x_5 - x_3 \leq -3, \tag{7}$$
$$x_5 - x_4 \leq -3, \tag{8}$$


0
V1
-5
0
0
-1
4
V6
-4
1
-3
V2
V0
0
0
5
-3
-3
0
V4
-1
-1
0
V3
0