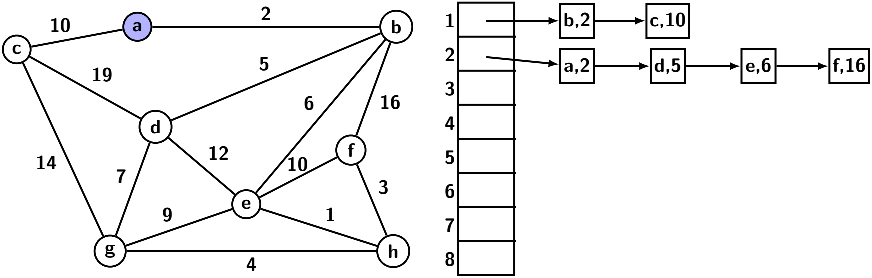
Example



10
2
5
19
6
16
14
7
12
10
3
9
1
4
a
b
c
d
e
f
g
h
1
2
3
4
5
6
7
8
b,2
c,10
a,2
d,5
e,6
f,16

Shortest Paths

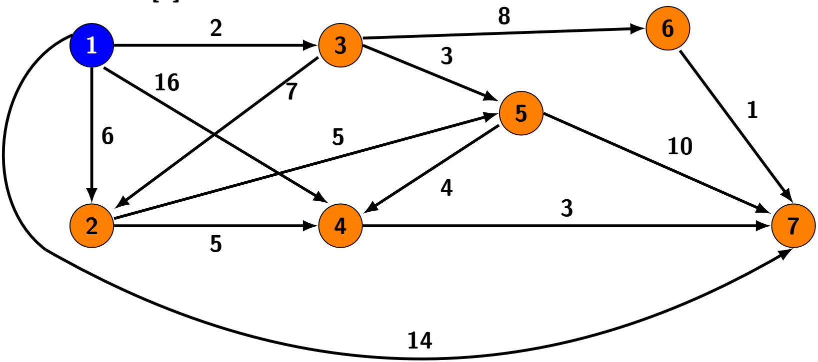
- Two possible variations in problems:
  1. Single source shortest paths
  2. All pairs of shortest paths
- Single source shortest path can be executed n times each with a different source vertex for all pairs shortest path.
- So, let us first examine how single source shortest path can be solved.

Djikstra's Algorithm

- Dijkstra's algorithm partitions the set of vertices logically into three partitions.
  - Set S: consists of vertices $u \in V$ such that dist[v] (from source s) already known.
  - Set I_1: consists of vertices $v \in V - S$ such that each $v \in I_1$ is directly connected to a vertex $x \in S$.
  - Set I_2: consists of vertices $w \in V - S - I_1$.
- Dijkstra's algorithm iteratively expands set S to include all vertices in V.

Example

Source vertex: d[1]=0



Shortest Paths

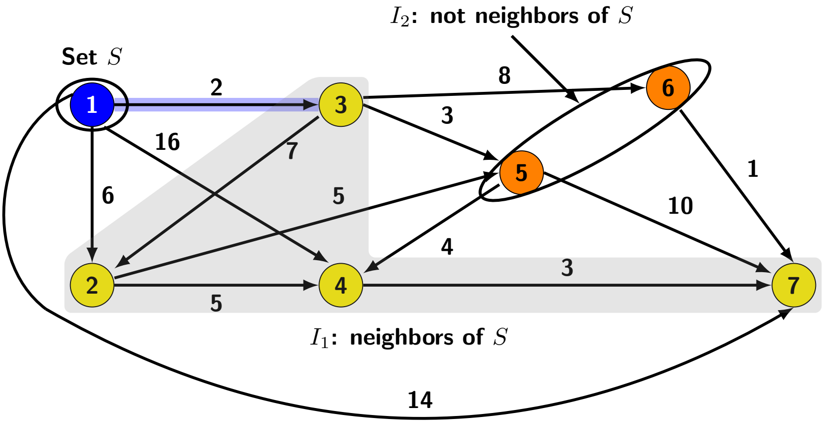
With source vertex 1, set $S = \{1\}$ and consider edges incident on vertices of S for relaxation.

Edge from S	old d[v]	new d[v]	old p[v]	new p[v]
2	∞	6	undef	1
3	∞	2	undef	1
4	∞	16	undef	1
7	∞	14	undef	1

Select vertex 3 (minimum d value) for inclusion into set S. So, $S = \{1, 3\}$ and $p[3] = 1$.

Shortest Paths

Blue colored vertices are in set S, yellow colored vertices are in set I_1, and organge colored are in set I_2.



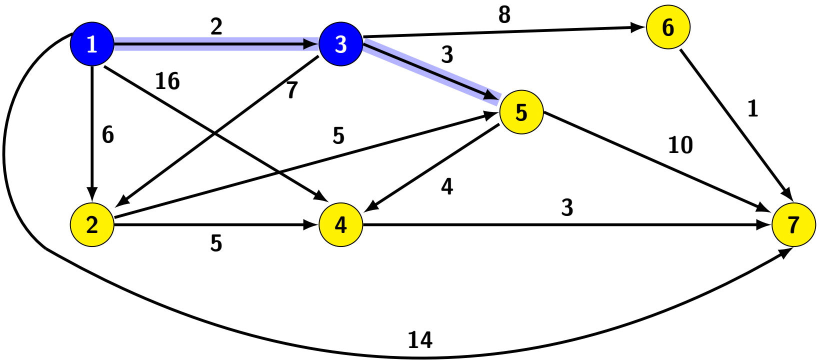
Now set $S = \{1, 3\}$ and only edges with end points 1 and 3 are considered for relaxation.

Edge from S	old d[v]	new d[v]	old p[v]	new p[v]
2	6	6	1	1
4	16	16	1	1
5	∞	5	undef	3
6	∞	10	undef	3
7	14	14	1	1

Select vertex 5 for inclusion into set S.
So $S = \{1, 3, 5\}$, and $p[3] = 1, p[5] = 3$

Shortest Paths

Blue colored vertices are in set S, yellow colored vertices are in set I_1.



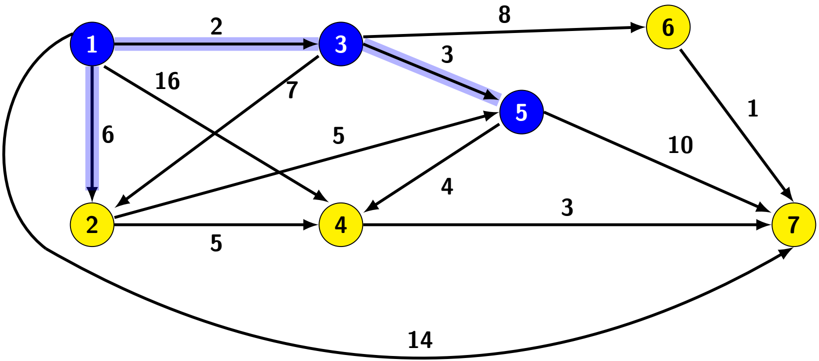
Shortest Paths

Now set $S = \{1, 3, 5\}$ and only edges with end points 1, 3 and 5 are considered for relaxation.

Edge from S	old d[v]	new d[v]	old p[v]	new p[v]
2	6	6	1	1
4	16	9	1	5
6	10	10	3	3
7	14	14	1	1

Select vertex 2 for inclusion into set S. So $S = \{1, 2, 3, 5\}$.
$p[3] = 1, p[5] = 3, p[2] = 1$

Blue colored vertices are in set S, yellow colored vertices are in set I_1.



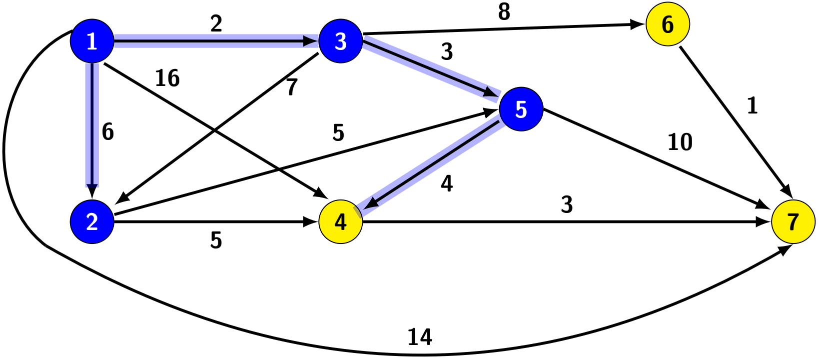
Shortest Paths

Now set $S = \{1, 2, 3, 5\}$ and only edges with end points 1, 2, 3 and 5 are considered for relaxation.

Edge from S	old d[v]	new d[v]	old p[v]	new p[v]
4	16	9	5	5
6	10	10	3	3
7	14	14	1	1

Select vertex 4 for inclusion into set S. So $S = \{1, 2, 3, 5\}$.
$p[3] = 1, p[5] = 3, p[2] = 1, p[4] = 5$

Blue colored vertices are in set 1, yellow colored vertices are in set 2.



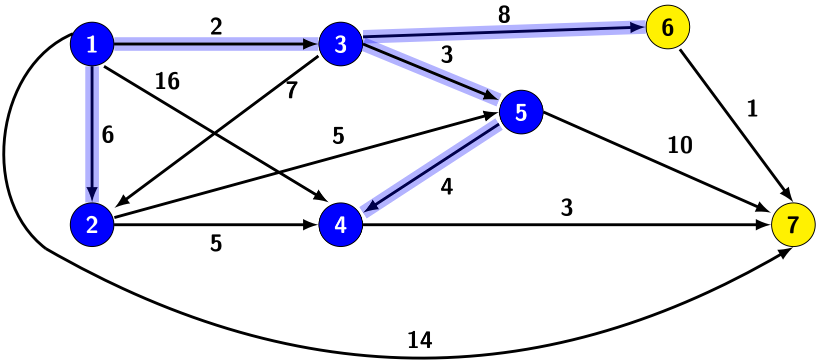
Shortest Paths

Now set $S = \{1, 2, 3, 4, 5\}$ and only edges with end points in S are considered for relaxation.

Edge from S	old d[v]	new d[v]	old p[v]	new p[v]
6	10	10	3	3
7	14	14	1	1

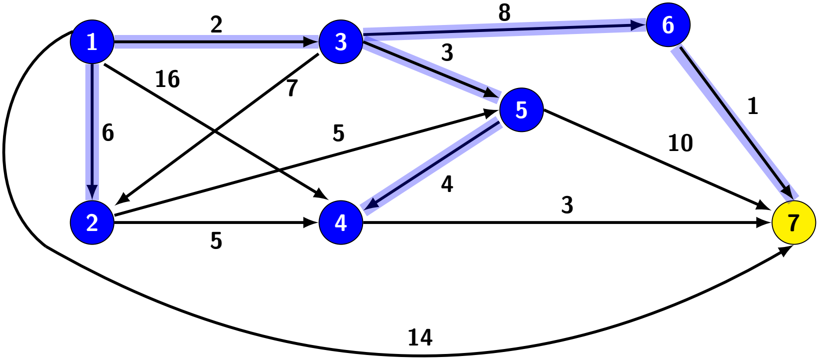
Select vertex 6 for inclusion into set S. So $S = \{1, 2, 3, 4, 5\}$.
$p[3] = 1, p[5] = 3, p[2] = 1, p[4] = 5, p[6] = 3$

Blue colored vertices are in set 1, yellow colored vertices are in set 2.



Shortest Paths

The remaining vertex 7 is included in S the last iteration.
$S = \{1, 2, 3, 4, 5, 6, 7\}$ and
$p[3] = 1, p[5] = 3, p[2] = 1, p[4] = 5, p[6] = 3$, $p[7] = 6$.



Pseudo Code for Edge Relaxation

```
Relax(u, v) {
    new_d = min {d[v], d[u] + w(u,v)};
    if (new_d[v] < d[v]) {
        d[v] = new_d;
        p[v] = u;
    }
}
```

Pseudo Code for Dijkstra's Algorithm

```
for all (v ∈ V) {
    d[v] = ∞; // Initialize distances
    p[v] = undef;
}
choose(s); // Source
d[s] = 0; // Initialize source distance
Q = V; // Initialize queue
while (!isEmpty(Q)) {
   u = min(d[u]); // Add new vertex
   Q = Q − {u}; // Update Q
   for each (v ∈ ADJ_G(u))
        Relax(u,v);
}
```

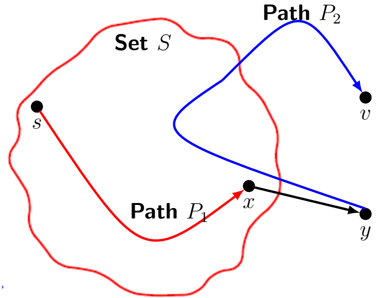
Optimal Substructure Property

- Consider a shortest path from s to t.
- Let v be an intermediate vertex on the path.
- Consider subpath $s \rightsquigarrow v$.
- If the subpath $s \rightsquigarrow v$ is not a shortest path, then $\exists$ a shorter path from s to v.
- It implies that that using the above shorter path we can reduce the cost of shortest path from s to t.

Correctness of Dijkstra

- Let v be the first vertex added to S such that d[v] $\neq$ dist[v].
- v cannot be s, because d[s] = 0.
- There must be path from s to v, otherwise d[v] would be ∞.
- Since, there is a finite path , a shortest path also exists.
- Furthermore the shortest path to $s \rightsquigarrow v$ does lie completely inside S.

Correctness of Dijkstra



- Consider shortest path $P : s \rightsquigarrow v$.
- Consider two partitions of

$$P = P_1 \cup P_2,$$

where P_1 completely belongs to S.

- Let y be first vertex on shortest path from s to v that does not belong to S.

Correctness of Dijkstra

- When x is included in S, d[x] = dist[x] and edge $x \rightarrow y$ must have been used to update
$$\text{d}[y] = \text{dist}[y] \leq \text{dist}[v] \leq \text{d}[v] \implies \text{d}[y] \leq \text{d}[v].$$
- Since both y and v are in $V - S$ when v was chosen,
$$\text{d}[v] \leq \text{d}[y].$$
- The above two set of inequalities imply
$$\text{d}[y] = \text{dist}[y] = \text{dist}[v] = \text{d}[v].$$
- So, d[v] = dist[v] is a contradiction.

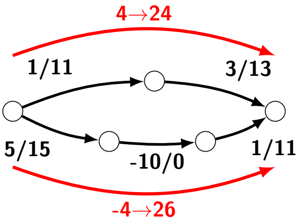
Running Time of Dijkstra's Algorithm

- Once Q is empty the algorithm terminates.
- On each iteration one vertex (with minimum d value) is added to S. Finding minimum requires $|V| - 1$ time.
  - So, with $|V| - 1$ iterations, it gives running time $|V|^2$.
- After a vertex is included in S it performs updates for $d[v]$s.
- However, each edge is used exactly only once during the execution of the algorithm.
  - So the cost of update cannot exceed $|E|$.
- Therefore, the running time is O($|E| + |V|^2$).

Running Time of Dijkstra's Algorithm

- If graph is dense, $|E| = \Theta(|V|^2)$.
- But we can do better than this by keeping $d[.]$ in form of a heap.
- We create a heap all $d[s]$.
- Use DeleteMIN to select the vertex to be included next.
  - It requires total of $|V| \log |V|$ unit of time.
- Relaxation step is equivalent to a decrease key operation, which requires $|E| \log |V|$ units of time.
- So, total time: O($|E| \log |V| + |V| \log |V|$).

Handling Negative Weight



- Can we handle negative edge weight as long as there are no negative cycles?
- One simple solution one can imagine is to add a big number turn negative edge weight to zero.
- It does not work, because the cost of path becomes $M \times$ (hop length) + cost(path).
- So with big value of M hop length begins to dominate.

Bellman Ford

```
Initialize();
for (i=1; i < |V|; i++) {
    foreach edge (u, v) {
        Relax(u,v);
    }
    foreach edge (u, v) {
        if (d[v] > d[u] + w(u,v))
            report negative cycle;
    }
}
Relax(u, v, w) {
    if (d[v] > d[u]+w(u,v)) {
        d[v] = d[u]+w(u,v);
        pred[v] = u;
    }
}
```

Bellman-Ford Algorithm

- Relaxation is the most important part of the operation.
- Label edges as: $e_1, e_2, \ldots e_m$, where $m = |E|$.
- Relaxation is repeated $|V| - 1$ and carried out for each edge every time.

$$\underbrace{\underbrace{e_1, e_2, \ldots, e_m,}_{\textbf{relaxation}} \; \underbrace{e_1, e_2, \ldots, e_m,}_{\textbf{relaxation}} \; \cdots, \; \underbrace{e_1, e_2, \ldots, e_m,}_{\textbf{relaxation}}}_{\textbf{Repeat } |V|-1 \textbf{ times}}$$

Correctness of Bellman-Ford

Lemma (*Main lemma*)

After k iterations of the edge relaxations, $d[v]$ contains correct shortest path from the source s to every vertes $v \in V$ using at most k edges.

Proof.

- **Basis**: Initially $d[s] = 0$ and $d[v] = \infty$ for all $v \in V - \{s\}$.
  - The shortest path from s to itself is zero.
  - At this point no edge is used for relaxation, consequently, the shortest path to other vertices are not known.
  - So, $d[v]$ correctly gives shortest path where each path may have at most 0 edges.

$\square$

Proof.

- **Hypothesis**: Assume that after kth iteration of relaxation step, $d[v]$ is the shortest path from s by using at most k edges.
- **Induction step**: Now consider $k+1$th iteration of relaxation step.
  - Consider some arbitrary vertex v.
  - In $k+1$th iteration, relaxation is performed by each incoming edge (u, v).
  - At this point $d[u]$ is the shortest path from s to u which used upto k edges.
  - After $k+1$th iteration, since $d[v] = \min\{d[v], d[u] + w(u, v)\}$, $d[v]$ should have shortest path from source to v with at most $k+1$ edges.

□

Correctness of Bellman-Ford

- A corollary of the above induction proof leads to the following conclusion:
  - After $|V|-1$ iterations of the edge relaxation step, $d[v]$ for every $v \in V$, contains the weight of the shortest path from s to v with at most $|V|-1$ edges.
  - If there is no negative cycle, then every cycle has a nonnegative cost.
  - So, gnoring or deleting the cycle from the path can only lead to lower cost.
  - Therefore, $d[v]$ after completing $|V|-1$ relaxation steps will give correct shortest path from s to v.

Minimum Spanning Tree

Definition (Spanning Tree)

A spanning tree T of an undirected connected graph G is a connected acyclic graph containing all vertices of G.

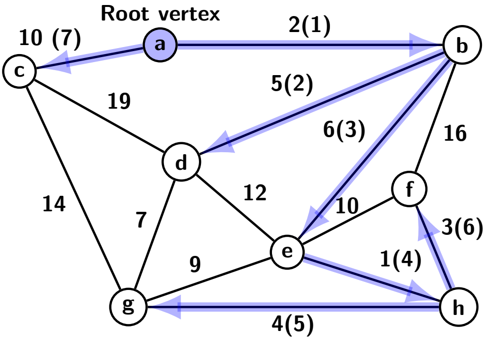
Definition (Minimum Spanning Tree)

Given a weighted graph find a spanning tree that has minimum weight.

Minimum Spanning Tree

- Uses a strategy similar to Dijkstra's algorithm.
- Initially, only one vertex v_0 is included in V_T.
- Then grow V_T by including exactly one vertex $u \in V - V_T$ by choosing cheapest edge $(v, u) \in E$ such that $v \in V_T$.
- Repeat previous step until $V_T = V$ or $V - V_T = \Phi$. in V_T

Minimum Spanning Tree



Edge weight(iteration#), Total weight = 30