Single-Source Shortest-Paths Problem

The Problem: Given a digraph with non-negative edge weights $G = (V, E)$
and a distinguished *source vertex*, $s \in V$,
determine the distance and a shortest path from the source vertex to every vertex in the digraph.

Question: How do you design an efficient algorithm for this problem?

Implementing the Idea of Relaxation

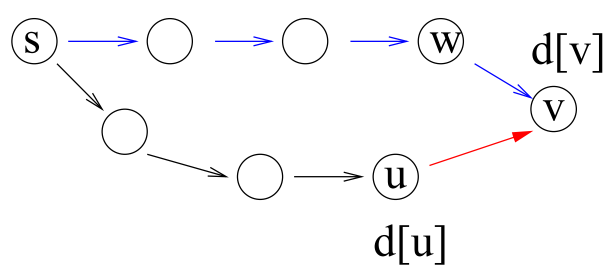
Consider an edge from a vertex u to v whose weight is $w(u, v)$. Suppose that we have already processed u so that we know $d[u] = \delta(s, u)$ and also computed a current estimate for $d[v]$. Then

- There is a (shortest) path from s to u with length $d[u]$.
- There is a path from s to v with length $d[v]$.

Combining this path from s to u with the edge (u, v), we obtain another path from s to v with length $d[u] + w(u, v)$.

If $d[u]+w(u, v) < d[v]$, then we replace the old path $\langle s, \ldots, w, v\rangle$ with the new shorter path $\langle s, \ldots, u, v\rangle$. Hence we update

- $d[v] = d[u] + w(u, v)$
- $pred[v] = u$ (originally, $pred[v] == w$).



The Algorithm for Relaxing an Edge

Relax(u,v)

$\{$

if $(d[u] + w(u,v) < d[v])$

$\{ \quad d[v] = d[u] + w(u,v);$

$\quad\quad pred[v] = u;$

$\}$

$\}$

Remark: The predecessor pointer $pred[\,]$ is for determining the shortest paths.

Idea of Dijkstra's Algorithm: Repeated Relaxation

- Dijkstra's algorithm operates by maintaining a subset of vertices, $S \subseteq V$, for which we know the true distance, that is $d[v] = \delta(s, v)$.

- Initially $S = \emptyset$, the empty set, and we set $d[s] = 0$ and $d[v] = \infty$ for all others vertices v. One by one we select vertices from $V \setminus S$ to add to S.

- The set S can be implemented using an array of vertex colors. Initially all vertices are white, and we set $color[v] =$ black to indicate that $v \in S$.

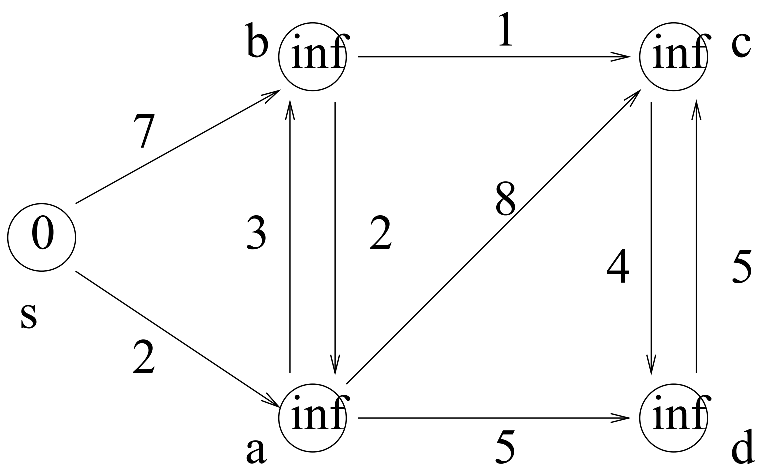
Description of Dijkstra's Algorithm

```
Dijkstra(G,w,s)
{                                              % Initialize
    for (each u ∈ V)
    {
        d[u] = ∞;
        color[u] =white;
    }
    d[s] = 0;
    pred[s] = NIL;
    Q = (queue with all vertices);

    while (Non-Empty(Q))                       % Process all vertices
    {
        u = Extract-Min(Q);                    % Find new vertex
        for (each v ∈ Adj[u])
            if (d[u] + w(u,v) < d[v])          % If estimate improves
            {
                d[v] = d[u] + w(u,v);            relax
                Decrease-Key(Q, v, d[v]);
                pred[v] = u;
            }
        color[u] =black;
    }
}
```

Dijkstra's Algorithm

Example:



Step 0: Initialization.

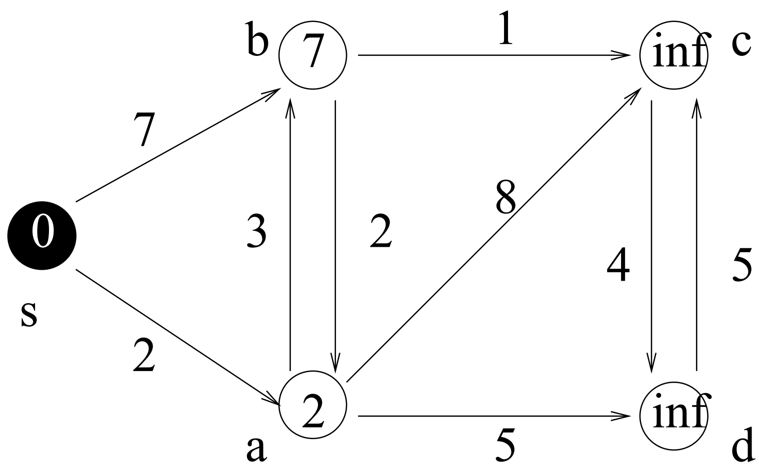
v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞
$pred[v]$	nil	nil	nil	nil	nil
$color[v]$	W	W	W	W	W

Priority Queue:

v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞

Dijkstra's Algorithm

Example:



Step 1: As $Adj[s] = \{a, b\}$, work on a and b and update information.

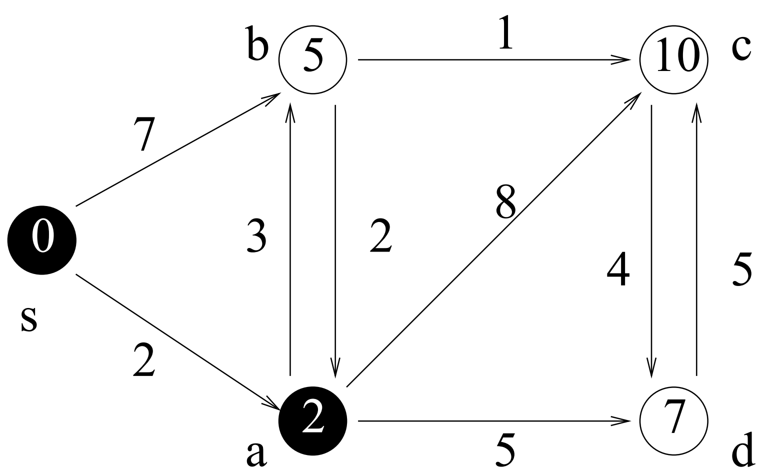
v	s	a	b	c	d
$d[v]$	0	2	7	∞	∞
$pred[v]$	nil	s	s	nil	nil
$color[v]$	B	W	W	W	W

Priority Queue:

v	a	b	c	d
$d[v]$	2	7	∞	∞

Dijkstra's Algorithm

Example:



Step 2: After Step 1, a has the minimum key in the priority queue. As $Adj[a] = \{b, c, d\}$, work on b, c, d and update information.

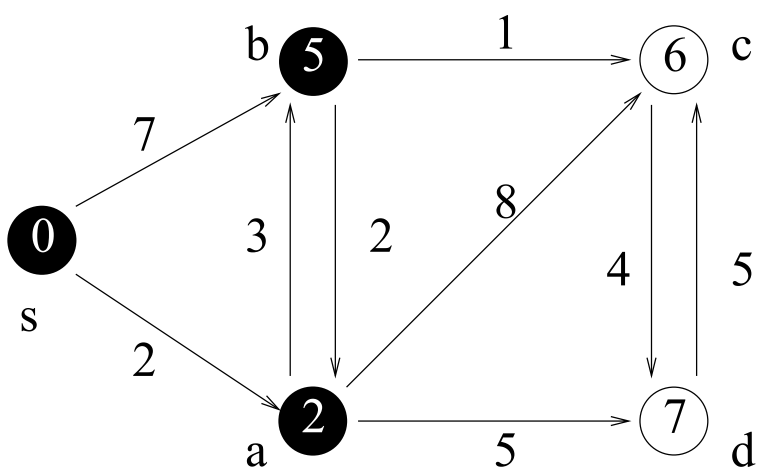
v	s	a	b	c	d
$d[v]$	0	2	5	10	7
$pred[v]$	nil	s	a	a	a
$color[v]$	B	B	W	W	W

Priority Queue:

v	b	c	d
$d[v]$	5	10	7

Dijkstra's Algorithm

Example:



Step 3: After Step 2, b has the minimum key in the priority queue. As $Adj[b] = \{a, c\}$, work on a, c and update information.

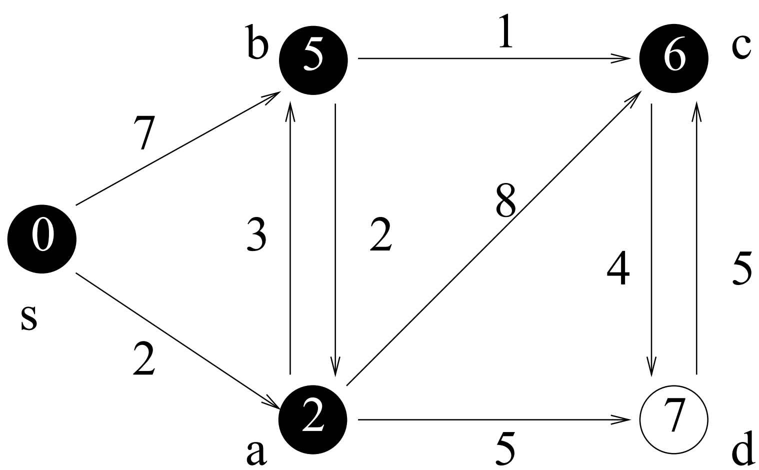
v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	W	W

Priority Queue:

v	c	d
$d[v]$	6	7

Dijkstra's Algorithm

Example:



Step 4: After Step 3, c has the minimum key in the priority queue. As $Adj[c] = \{d\}$, work on d and update information.

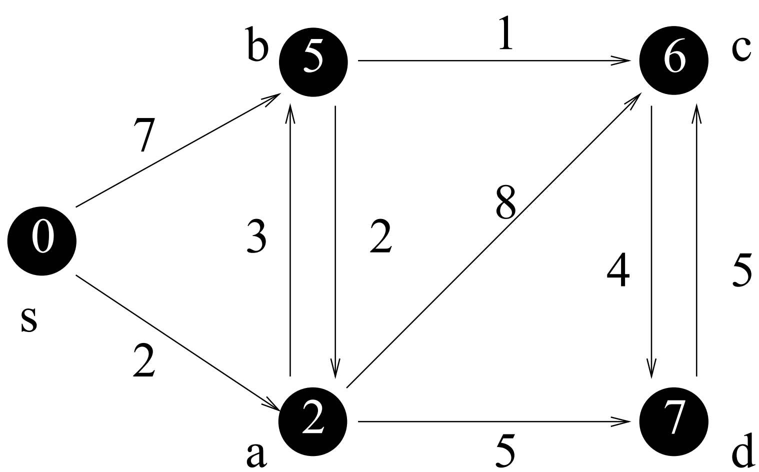
v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	W

Priority Queue:

v	d
$d[v]$	7

Dijkstra's Algorithm

Example:



Step 5: After Step 4, d has the minimum key in the priority queue. As $Adj[d] = \{c\}$, work on c and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	B

Priority Queue: $Q = \emptyset$.

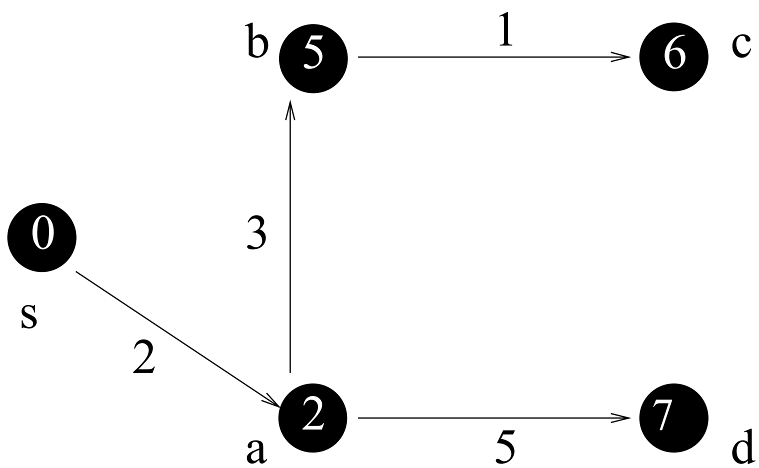
We are done.

Dijkstra's Algorithm

Shortest Path Tree: $T = (V, A)$, where

$$A = \{(pred[v], v) | v \in V \setminus \{s\}\}.$$

The array $pred[v]$ is used to build the tree.



Example:

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a

Analysis of Dijkstra's Algorithm:

The initialization uses only $O(n)$ time.

Each vertex is processed exactly once so `Non-Empty()` and `Extract-Min()` are called exactly once, e.g., n times in total.

The inner loop for (each $v \in Adj[u]$) is called once for each edge in the graph. Each call of the inner loop does $O(1)$ work plus, possibly, one `Decrease-Key` operation.

Recalling that all of the priority queue operations require $O(\log |Q|) = O(\log n)$ time we have that the algorithm uses

$$nO(1+\log n)+O(e)+O(e\log n) = O((n+e)\log n)$$

time.