Single Source/All Destinations: Bellman-Ford Algorithm

- $dist^k[u]$
  - The length of a shortest path from the source vertex v to vertex u under the constraint that the shortest path contains at most k edges

- When there are no cycles of negative length, we can find exact shortest path which has at most n-1 edges

Single Source/All Destinations: Bellman-Ford Algorithm

- Goal
  - Compute $dist^{n-1}[u]$ for all u
  - This can be done using dynamic programming methodology
- Observation
  - If the shortest path from v to u with at most k, k>1, edges has no more than k-1 edges
    - $dist^{k}[u] = dist^{k-1}[u]$
  - If the shortest path from v to u with at most k, k>1, edges has exactly k edges
    - It is comprised of a shortest path from v to some vertex j followed by the edge <j,u>. The path from v to j has k-1 edges, and its length is $dist^{k-1}[j]$
    - All vertices i such that the edge <i,u> is in the graph are candidates for j
    - $dist^{k}[u] = \min\{dist^{k-1}[u], \min\{dist^{k-1}[i]+length[i][u]\}\}$

Single Source/All Destinations: Bellman-Ford Algorithm

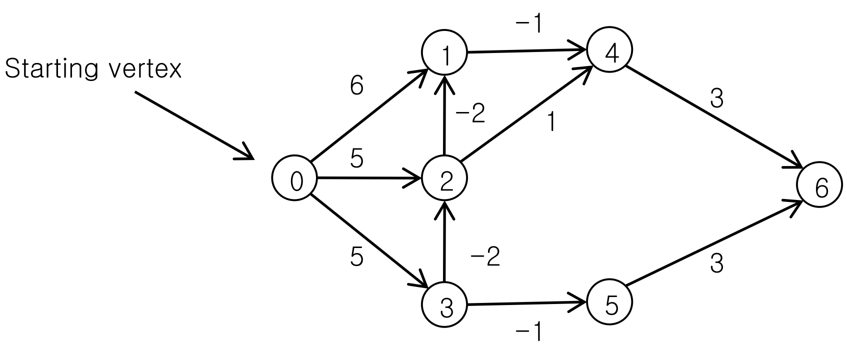
```
1. void MatrixWDigraph::BellmanFord(const int n, const int v)
2. { // Single source all destination shortest paths with negative edge lengths
3.       for(int i=0; i<n; i++) dist[i] = length[v][i]; // initialize dist

4.       for(int k=2; k<=n-1; k++)
5.             for(each u such that u != v and u has at least one incoming edge)
6.                   for(each <i,u> in the graph)
7.                         if(dist[u] > dist[i] + length[i][u]) dist[u] = dist[i] + length[i][u];
8. }
```

Program 6.9: Bellman and Ford algorithm to compute shortest paths

- Time Complexity
  - lines 5 to 7
    - $O(n^2)$ when adjacency matrices are used
    - $O(e)$ when adjacency lists are used
  - Overall
    - $O(n^3)$ when adjacency matrices are used
    - $O(ne)$ when adjacency lists are used

Single Source/All Destinations: Bellman-Ford Algorithm



(a) A directed graph

k	$dist^k[7]$						
	0	1	2	3	4	5	6
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

(b) $dist^k$

Figure 6.31: Shortest paths with negative edge lengths

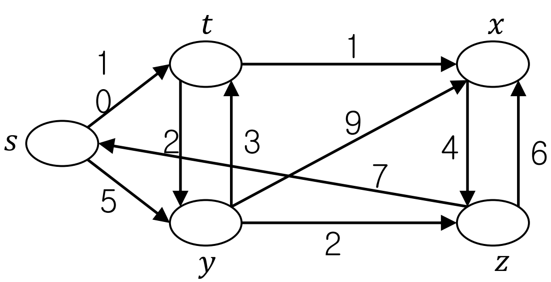
Dijkstra’s Algorithm

Dijkstra's Algorithm

- On weighted, directed graph G=(V,E) for which all edge weights are nonnegative.
- The running time of Dijkstra's algorithm is lower than that of the Bellman-Ford algorithm.

Dijkstra's Algorithm

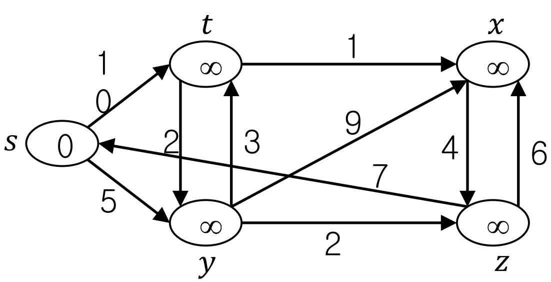
```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```




```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π=NIL
    r.key = 0
    Q = G.V
    while Q ≠ Ø
        u = Extract-Min(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u,v) < v.key
                v.π = u
                v.key = w(u,v)
```

Dijkstra’s Algorithm

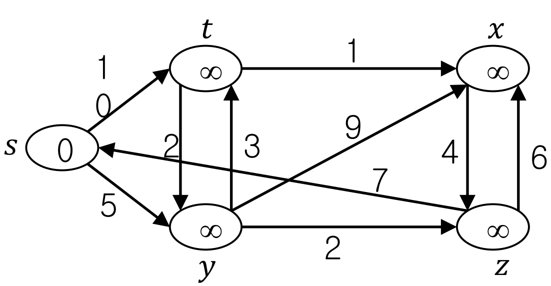
```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```



Dijkstra’s Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

S=∅



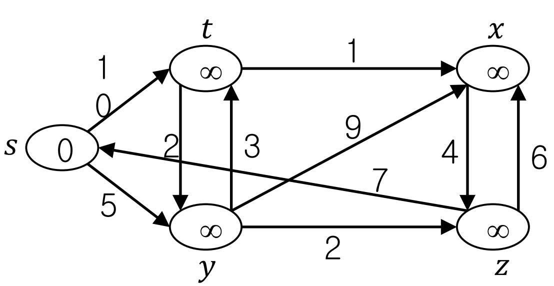
Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

S=∅

Q

G.V	*s*	*t*	*x*	*y*	*z*
d	0	∞	∞	∞	∞



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

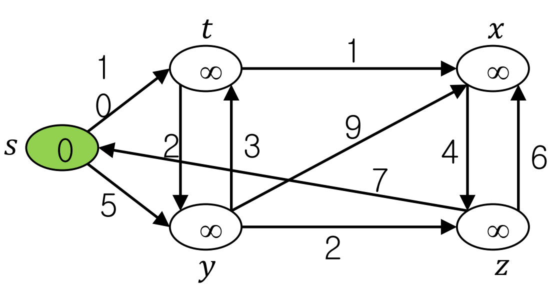
S={s}

Q

G.V	t	x	y	z
d	∞	∞	∞	∞

u=s

G.adj[s] = {t, y}



Dijkstra’s Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

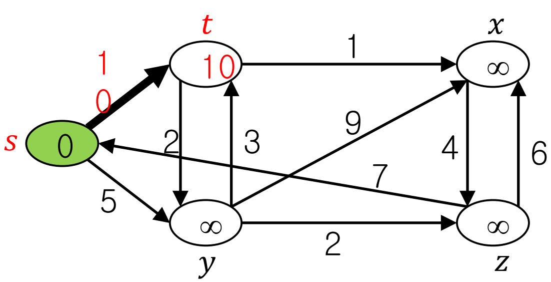
S={s}

Q

G.V	t	x	y	z
d	10	∞	∞	∞

u=s

G.adj[s] = {t, y}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

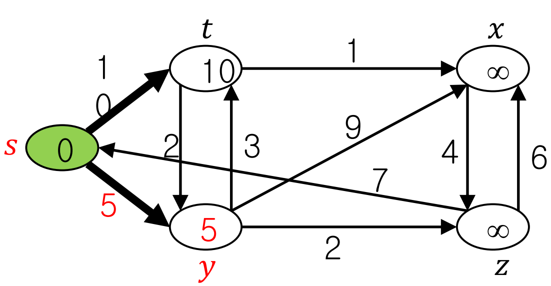
S={s}

Q

G.V	t	x	y	z
d	10	∞	5	∞

u=s

G.adj[s] = {t, y}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

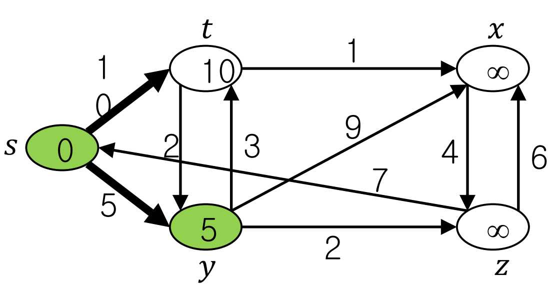
S=$\{s, y\}$

Q

G.V	t	x	z
d	10	∞	∞

u=y

G.adj$[y]$ = $\{t, x, z\}$



Dijkstra's Algorithm

DIJKSTRA(G,w,s)

1. INITIALIZE-SINGLE-SOURCE(G,s)
2. S = ∅
3. Q = G.V
4. **while** Q ≠ ∅
5. u = Extract-Min(Q)
6. S = S U {u}
7. **for** each vertex v ∈ G.Adj[u]
8. RELAX(u,v,w)

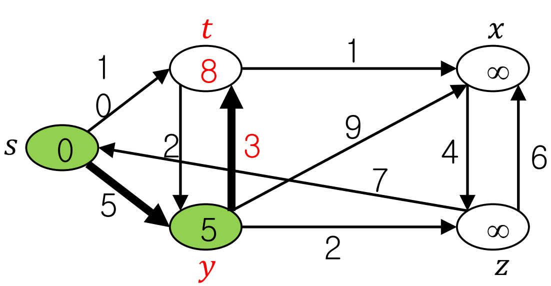
S={s, y}

Q

G.V	t	x	z
d	8	∞	∞

u=y

G.adj[y] = {t, x, z}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

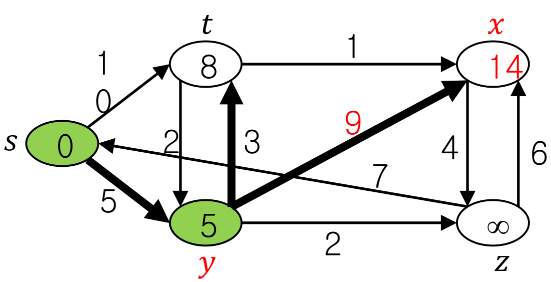
S={s, y}

Q

G.V	t	x	z
d	8	14	∞

u=y

G.adj[y] = {t, x, z}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

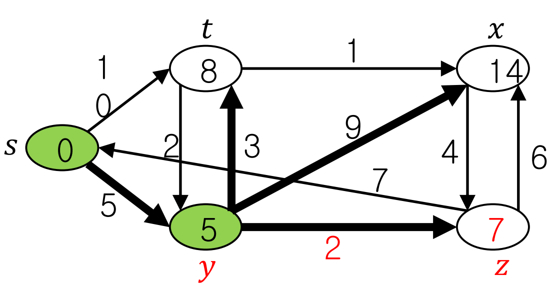
S={s, y}

Q

G.V	t	x	z
d	8	14	7

u=y

G.adj[y] = {t, x, z}



Dijkstra’s Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

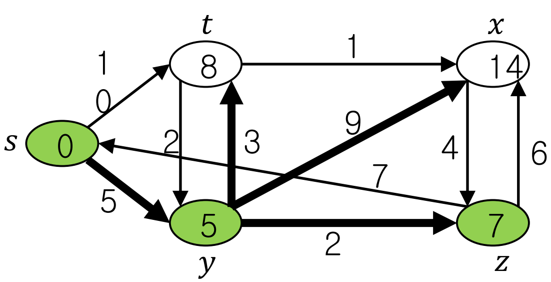
S={s, y, z}

Q

G.V	t	x
d	8	14

u=z

G.adj[z] = {x, s}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

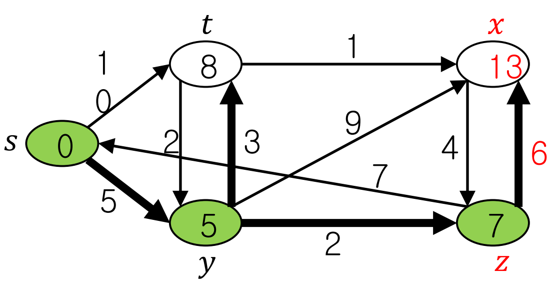
S={s, y, z}

Q

G.V	t	x
d	8	13

u=z

G.adj[z] = {x, s}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

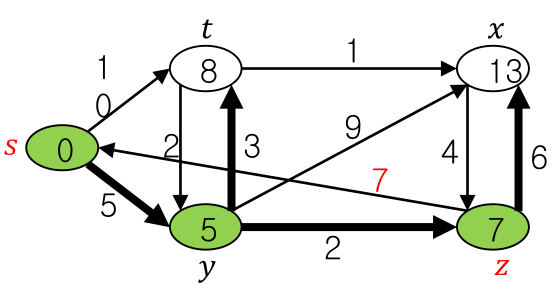
S={s, y, z}

Q

G.V	t	x
d	8	13

u=z

G.adj[z] = {x, s}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
      INITIALIZE-SINGLE-SOURCE(G,s)
      S = ∅
      Q = G.V
      while Q ≠ ∅
          u = Extract-Min(Q)
          S = S U {u}
          for each vertex v ∈ G.Adj[u]
              RELAX(u,v,w)
```

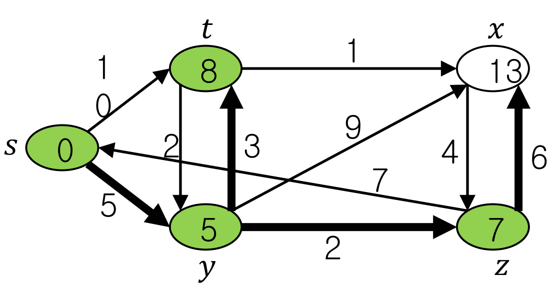
S={s, y, z, t}

Q

G.V	x
d	13

u=t

G.adj[t] = {x, y}



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

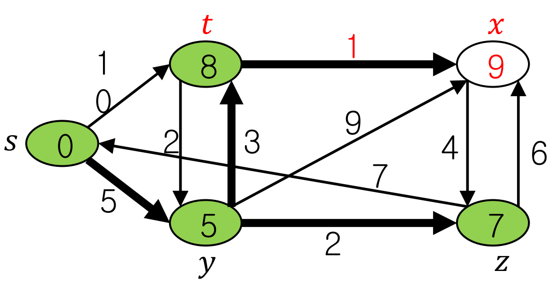
S=$\{s,\ y,\ z,\ t\}$

Q

G.V	x
d	9

u=t

G.adj$[t]$ = $\{x,\ y\}$



Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

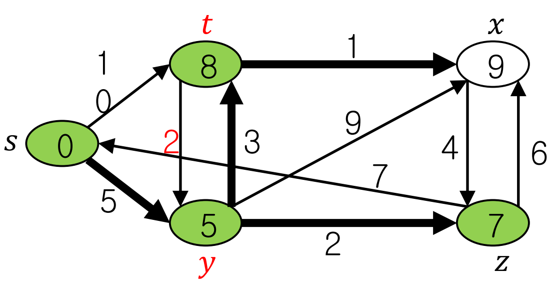
S={s, y, z, t}

Q

G.V	x
d	9

u=t

G.adj[t] = {x, y}



Dijkstra's Algorithm

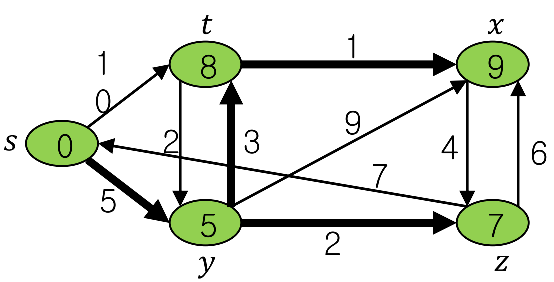
```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

S=$\{s, y, z, t, x\}$

Q=$\varnothing$

u=x

G.adj$[x]$ = $\{z\}$



Dijkstra's Algorithm

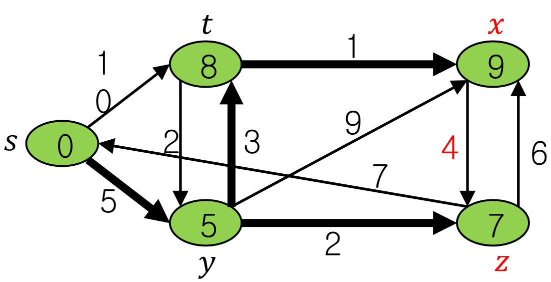
```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

S=$\{s,\ y,\ z,\ t,\ x\}$

Q=∅

u=x

G.adj$[x]$ = $\{z\}$

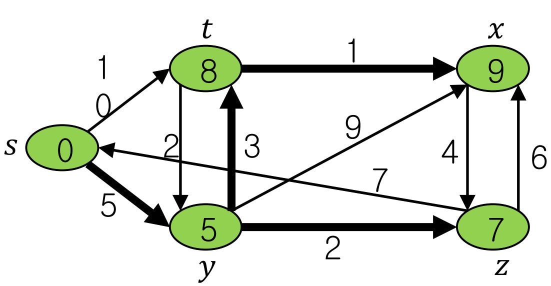


Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

S=$\{s, y, z, t, x\}$

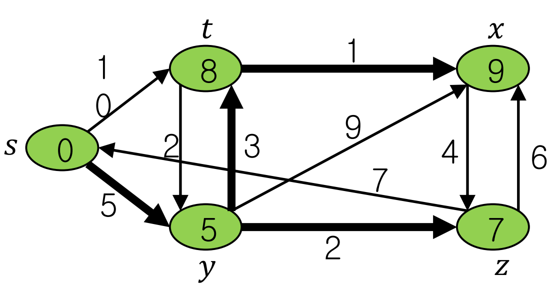
Q=$\emptyset$



Dijkstra’s Algorithm

```
DIJKSTRA(G,w,s)
        INITIALIZE-SINGLE-SOURCE(G,s)
        S = ∅
        Q = G.V
        while Q ≠ ∅
            u = Extract-Min(Q)
            S = S U {u}
            for each vertex v ∈ G.Adj[u]
                RELAX(u,v,w)
```

S=$\{s,\ y,\ z,\ t,\ x\}$



Running Time of Dijkstra's Algorithm

- It depends on implementations of the min-priority queue Q.
- If we implement Q as a binary min-heap,
  - EXTRACT-MIN takes O(lg |V|) time.
  - DECREASE-KEY takes O(lg |V|) time.
- If we implement Q as a simple array,
  - EXTRACT-MIN takes O(|V|) time.
  - DECREASE-KEY O(1) time.
- If we implement Q as a Fibonacci heap,
  - EXTRACT-MIN takes O(lg |V|) amortized time.
  - DECREASE-KEY O(1) amortized time.

Dijkstra’s Algorithm

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S U {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u,v,w)
```

Dijkstra's Algorithm

- min-priority queue : array

DIJKSTRA(G,w,s)

1. INITIALIZE-SINGLE-SOURCE(G,s) ← $\boxed{O(|V|)}$
2. S = ∅ ← $\boxed{O(1)}$
3. Q = G.V ← $\boxed{O(|V|)}$
4. **while** Q ≠ ∅
5. u = Extract-Min(Q) ← $\boxed{O(|V|^2)}$
6. S = S U {u} ← $\boxed{O(|V|)}$
7. **for** each vertex v ∈ G.Adj[u] ← $\boxed{O(|E|)}$
8. RELAX(u,v,w)

Dijkstra's algorithm running time is $O(|V|^2)$

Dijkstra’s Algorithm

- min-priority queue : binary min-heap

DIJKSTRA(G,w,s)

```
INITIALIZE-SINGLE-SOURCE(G,s)   ← O(|V|)
S = ∅                           ← O(1)
Q = G.V                         ← O(|V|)
while Q ≠ ∅
    u = Extract-Min(Q)          ← O(|V| lg |V|)
    S = S U {u}                 ← O(|V|)
    for each vertex v ∈ G.Adj[u]   ← O(|E| lg |V|)
        RELAX(u,v,w)
```

- Running time:
  - O((|V|+|E|) lg |V|) , if all vertices are reachable → O(|E| lg |V|).
  - Better than $O(|V|^2)$, if the graph is sufficiently sparse: $|E| = o(|V|^2 / \lg |V|)$.

Dijkstra's Algorithm

- min-priority queue : Fibonacci heap

```
DIJKSTRA(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)      O(|V|)
    S = ∅                              O(1)
    Q = G.V                            O(|V|)
    while Q ≠ ∅
        u = Extract-Min(Q)             O(|V| lg |V|)
        S = S U {u}                    O(|V|)
        for each vertex v ∈ G.Adj[u]   O(|E|)
            RELAX(u,v,w)
```

Dijkstra's algorithm running time is O(|V| lg |V|+ |E|)

Theorem 24.6 (Correctness of Dijkstra's Algorithm)

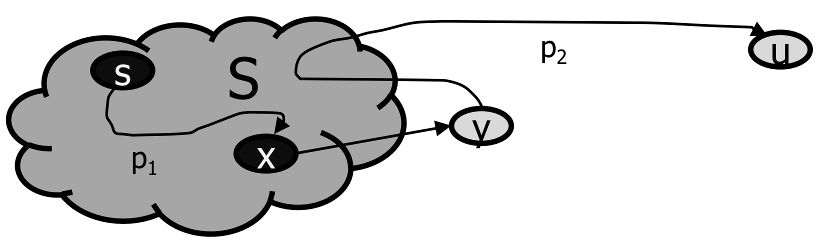
- Dijkstra's algorithm, run on a weighted, directed graph $G=(V,E)$ with non-negative weight function w and source s, terminates with $u.d=\delta(s,u)$ for all vertices $u \in V$.

Theorem 24.6 (Proof)

- Loop invariant
- At the start of each iteration of the while loop of lines 4-8, v.d= δ (s,v) for each vertex v in S
- Initialization: S={}, so true
- Maintenance:
  - Let u be the first vertex for which u.d ≠ δ(s,u) when it is added to set S
  - u ≠ s because s is the first vertex added to set S and s.d= δ(s,s)=0
  - Because u ≠ s, we also have that S ≠ {} just before u is added to S
  - There must be some path from s to u, for otherwise u.d=δ (s,u)= ∞ by no-path property
  - There is a shortest path p from s to u
  - Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in V-S, namely u.

Theorem 24.6 (Proof)

- Let us consider the first vertex y along p such that $y \in V-S$, and let $x \in S$ be y's predecessor along p
- Figure shown below illustrates, we can decompose path p into $s \rightsquigarrow x \rightarrow y \rightsquigarrow u$ ($s \rightsquigarrow x$: p_1 , $y \rightsquigarrow u$: p_2)
- Claim: $y.d=\delta(s,y)$ when u is added to S
  - $x.d=\delta(s,x)$ when x was added to s
    ($\because$ we chose u as the first vertex for which $u.d \neq \delta(s,u)$)
  - Edge (x,y) was relaxed at that time, and the claim follows from the convergence property



Theorem 24.6 (Proof)

- We can now obtain a contradiction to prove u.d= δ(s,u)
  - δ(s,y)≤δ(s,u) (∵ y appears before u on a shortest path from s to u and all edge weights are non-negative)
  - y.d=δ(s,y)≤δ(s,u)≤u.d (by the upper-bound property)
  - But because both vertices u and y were in V-S when u was chosen in line 5, u.d≤y.d
  - y.d=δ(s,y)=δ(s,u)=u.d
  - Consequently u.d=δ(s,u), which contradicts our choice of u.
- u.d=δ(s,u) when u is added to S, and that this equality is maintained at all times thereafter

- Termination : At termination, Q={} which, along with our earlier invariant that Q=V-S, implies that S=V. u.d=δ(s,u) for all vertices u ∈V

Shortest-Path Algorithms

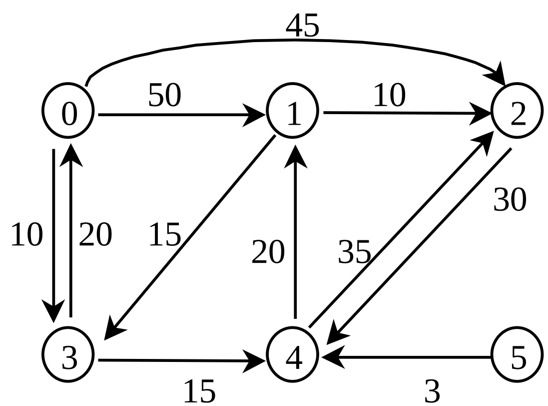
	Bellman-Ford	**Dijkstra**
Negative Edge	O	X
Positive Cycle	O	O
Negative Cycle	X	X
Time Complexity	O(|V||E|)	Array: O(|V|^2) Min-heap: O((|V|+|E|)lg|V|) Fibonacci heap: O(|V|lg|V|+|E|)

Single Source/All Destinations: Dijkstra’s Algorithm

- Definition of class Graph

```
class Graph
{
private:
  int length[nmax][nmax]; // w(u,v)
  int dist[nmax];              // v.d
  Boolean s[nmax];
public:
  void ShortestPath(const int, const int);
  int choose(const int);
};
```

Single Source/All Destinations: Dijkstra's Algorithm



(a) graph

Path	*Length*
1) 0, 3	10
2) 0, 3, 4	25
3) 0, 3, 4, 1	45
4) 0, 2	45

(b) shortest paths from 0

Figure 6.26 : Graph and shortest paths from vertex 0

Single Source/All Destinations: Dijkstra's Algorithm

- S: set of vertices to which the shortest paths have already been found
- dist[w], for w not in S
  - The length of the shortest path starting from v
  - Go through only the vertices that are in S
  - End at w
- A greedy algorithm will generate the shortest paths in nondecreasing order of path length

Single Source/All Destinations: Dijkstra's Algorithm

- We observe that when paths are generated in nondecreasing order of length
  ① If the next shortest path is to vertex u, then the path goes through only vertices that are in S
    - All of the intermediate vertices on the shortest path must be in S
    - Proof) Assume there is a vertex w on this path that is not in S. Then the v-to-u path also contains a path from v to w that is less than that of the v-to-u path. But, by the observation, paths are generated in nondecreasing order of length. so, the shorter path from v to w has been generated already. Hence, there is no intermediate vertex that is not in S

  ② The destination of the next path generated must be the vertex u that has the minimum distance, dist[u], among all vertices not in S
    - This follows from the definition of dist and observation ①
    - If there are several vertices not in S with the same dist, then any of these may be selected

  ③ The vertex u selected in ② becomes a member of S. At this point, the length of the shortest paths starting at v, going through vertices only in S, and ending at a vertex w not in S may decrease. Therefore, if dist[w] decreases, then the change is due to the path from v to u to w. The length of this path is dist[u]+length(<u,w>)

Single Source/All Destinations: Dijkstra’s Algorithm

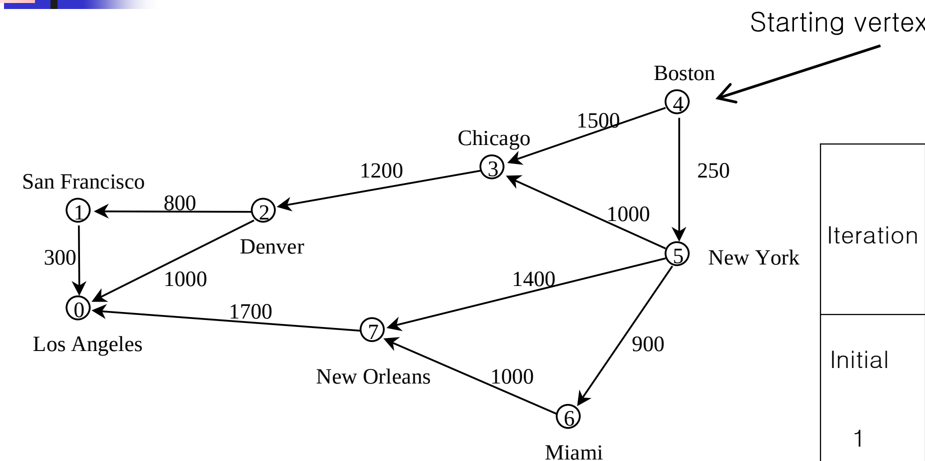
```
void MatrixWDigraph::ShortestPath(const int n, const int v)
{ // dist[j], 0 ≤ j < n, is set to the length of the shortest path from v to j
  // in a digraph G with n vertices and edge lengths given by length[i][j].
      for(int i = 0; i < n; i++) { s[i] = false; dist[i] = length[v][i]; } // initialize
      s[v] = true;
      dist[v] = 0;

      for(i = 0; i < n-1; i++) { // determine n-1 paths from vertex v
              int u = Choose(n); // .returns a value u such that:
                                           // dist[u] = minimum dist[w], where s[w] = false
              s[u] = true;
              for(int w = 0; w < n; w++)
                      if(!s[w] && dist[u] + length[u][w] < dist[w])
                              dist[w] = dist[u] + length[u][w];
      } // end of for(i = 0; …)
}
```

Program 6.8 : Determining the shortest paths

- Time Complexity
  - $O(n^2)$, n: number of vertices

Example 6.5



(a) Digraph

$$
\begin{array}{c|cccccccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & & & & & & & \\
1 & 300 & 0 & & & & & & \\
2 & 1000 & 800 & 0 & & & & & \\
3 & & & 1200 & 0 & & & & \\
4 & & & & 1500 & 0 & 250 & & \\
5 & & & & 1000 & & 0 & 900 & 1400 \\
6 & & & & & & & 0 & 1000 \\
7 & 1700 & & & & & & & 0
\end{array}
$$

(b) Length-adjacency matrix

Figure 6.27 : Digraph for Example 6.5

Iteration	S (shortest paths have already been found)	Vertex selected	Distance LA [0]	SF [1]	DEN [2]	CHI [3]	BOST [4]	NY [5]	MIA [6]	NO [7]
Initial	--	----	+∞	+∞	+∞	1500	0	250	+∞	+∞
1	{4}	5	+∞	+∞	+∞	1250	0	250	1150	1650
2	{4,5}	6	+∞	+∞	+∞	1250	0	250	1150	1650
3	{4,5,6}	3	+∞	+∞	2450	1250	0	250	1150	1650
4	{4,5,6,3}	7	3350	+∞	2450	1250	0	250	1150	1650
5	{4,5,6,3,7}	2	3350	3250	2450	1250	0	250	1150	1650
6	{4,5,6,3,7,2}	1	3350	3250	2450	1250	0	250	1150	1650
	{4,5,6,3,7,2,1}									

Figure 6.28 : Action of ShortestPath on digraph of Figure 6.27