Algorithms: Greedy Method

Shortest Path Problems

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Shortest-Path

- Given a graph (directed or undirected) $G = (V, E)$ with weight function $w: E \to \mathbf{R}$ and a vertex $s \in V$, find for all vertices $v \in V$ the minimum possible weight for path from s to v.

- The weight of path $p = v_1 \to v_2 \to \ldots \to v_k$ is

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

- Shortest path = a path of the minimum weight

- Algorithm will compute a shortest-path tree.

Shortest-Path Problems

- Shortest-Path problems
  - **Single-Source:** Find a shortest path from a given source (vertex *s*) to each of the vertices.
  - **Single-Destination:** Find a shortest path to a given destination (vertex *t*) from each of the vertices.
  - **Single-Pair:** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
  - **All-Pairs:** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.

Single-Source Shortest Path

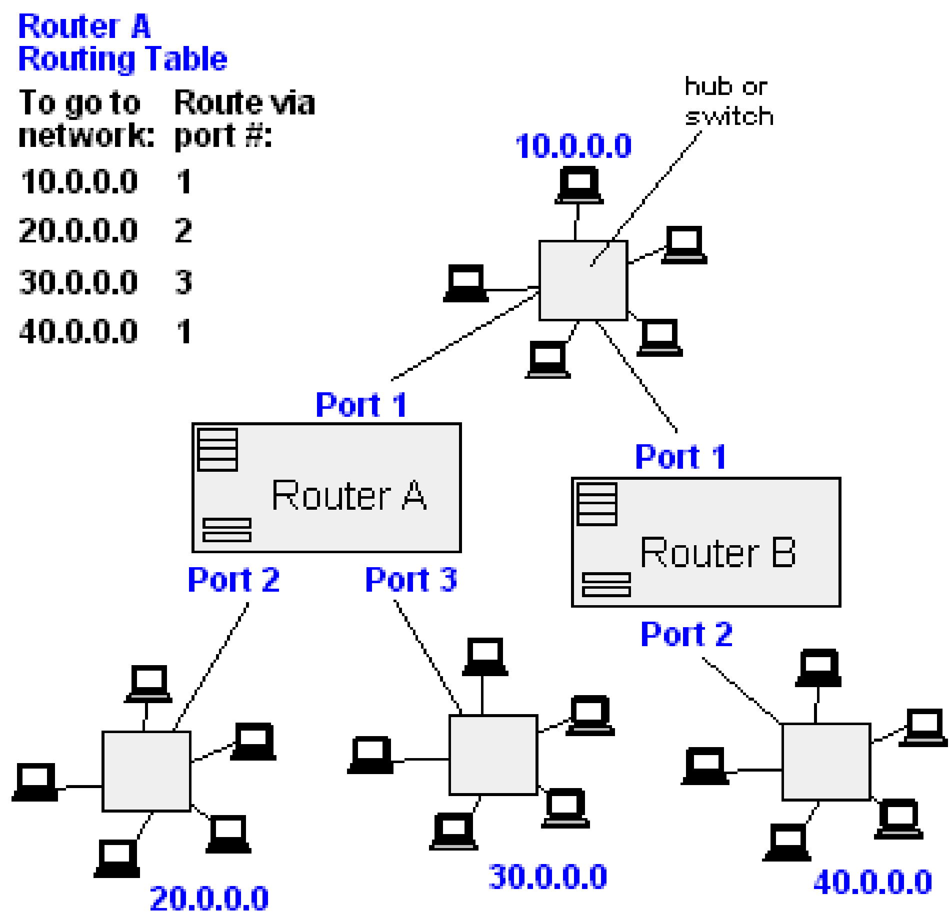
- Given a graph (directed or undirected) $G = (V, E)$ with weight function $w: E \rightarrow \mathbf{R}$ and a vertex $s \in V$, find for all vertices $v \in V$ the minimum possible weight for path from s to v.

- We will discuss two general case algorithms:
  - **Dijkstra's Algorithm** (positive edge weights only)
  - **Bellman-Ford Algorithm** (positive and negative edge weights)

- If all edge weights are equal (let's say 1), the problem is solved by BFS in $\Theta(V + E)$ time.

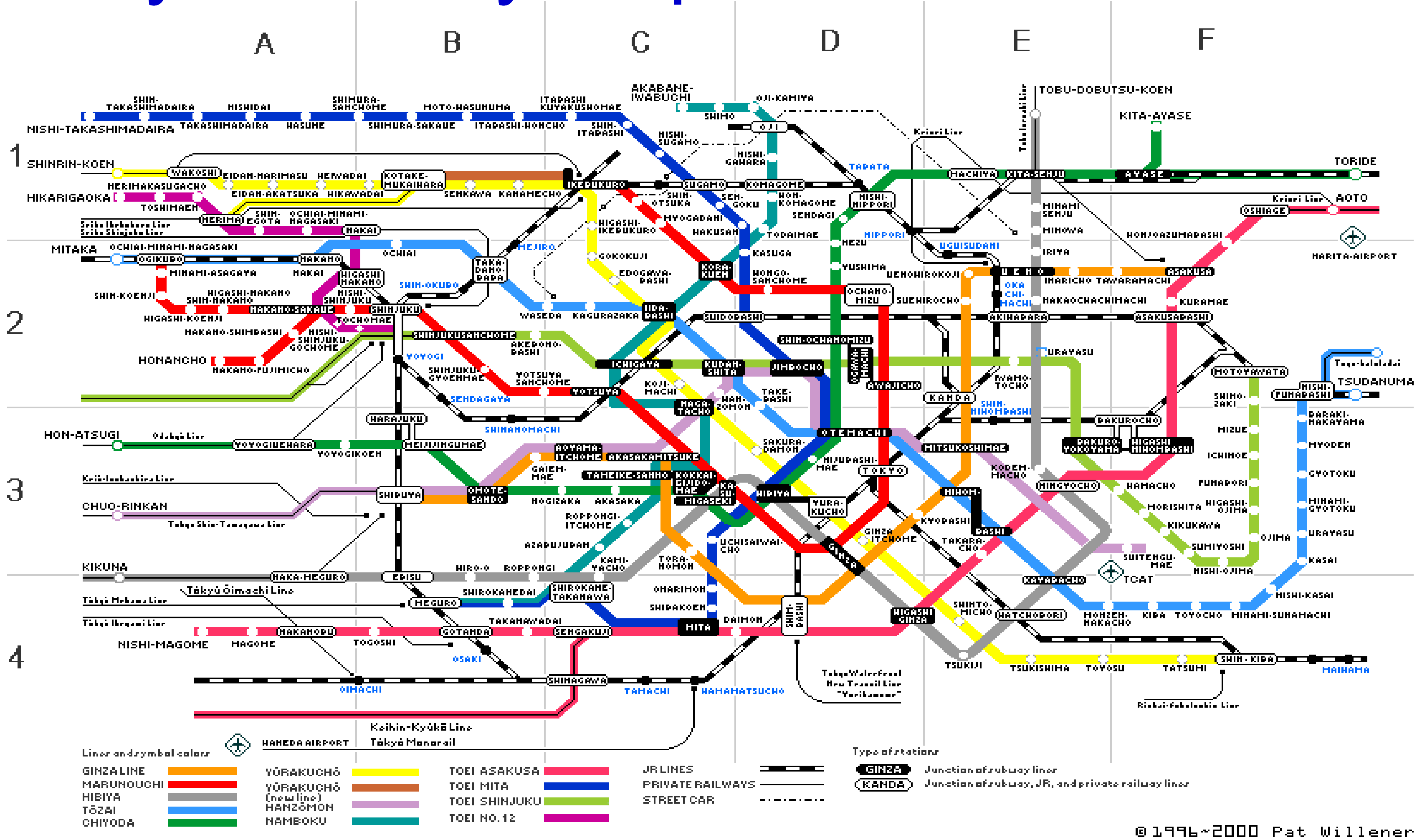
Single-Source Shortest Path

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



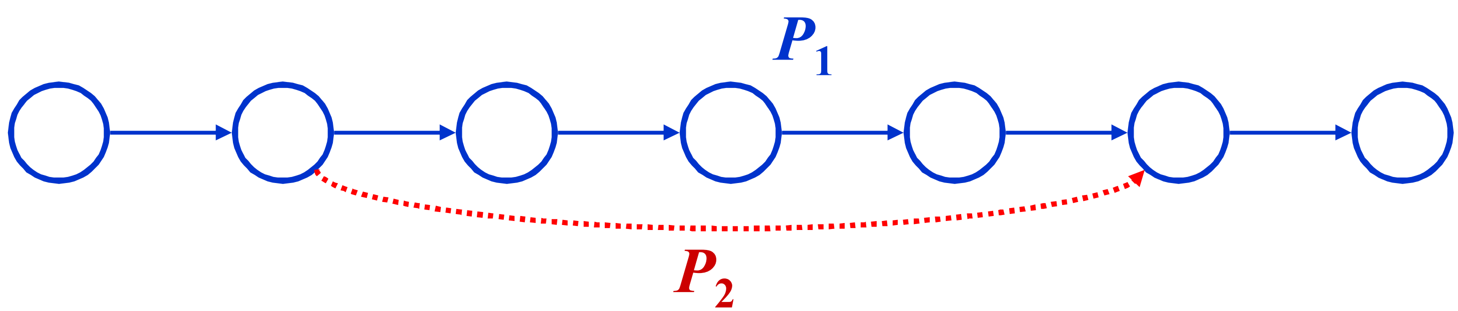
Single-Source Shortest Path

Tokyo Subway Map



Shortest Path Properties

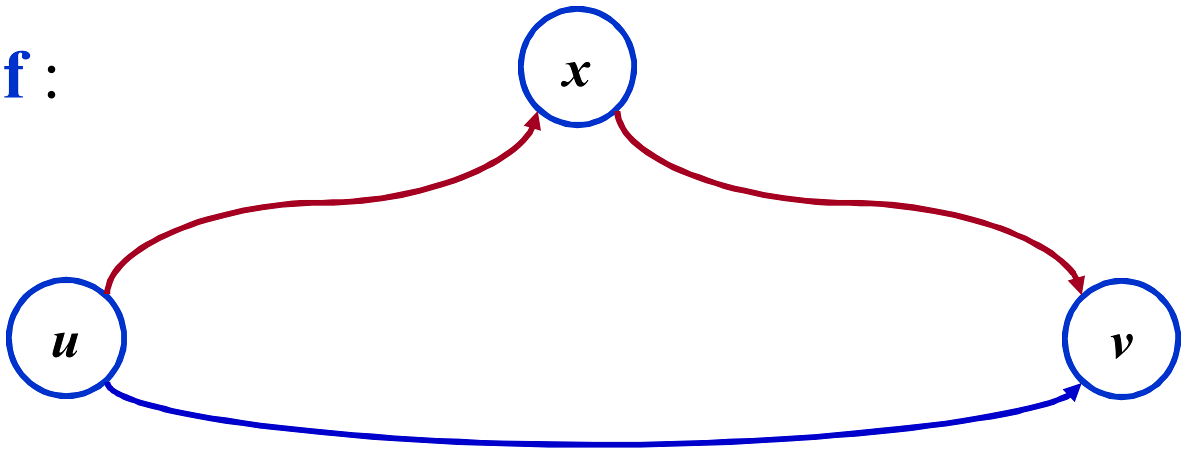
- The shortest path problem satisfies the *optimal substructure property*:
  - Subpaths of shortest paths are shortest paths.



  - **Proof**: suppose some subpath P_1 is not a shortest path
    - There must then exist a shorter subpath P_2
    - Could substitute the subpath P_1 by the shorter path P_2
    - But then overall path is not the shortest path. Contradiction

Shortest Path Properties

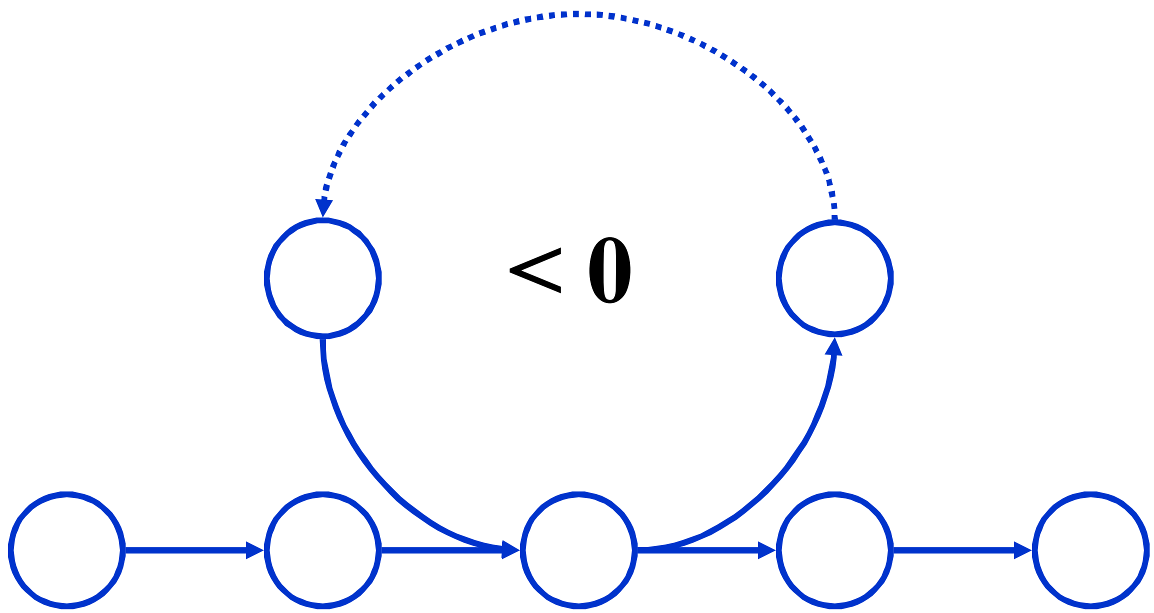
- Define $\delta(u, v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the *triangle inequality*: $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$
- **Proof** :



This path is no longer than any other path

Shortest Path Properties

- In graphs with negative weight cycles, some shortest paths will not exist (*Why ?*):



Relaxation

- A key technique in shortest path algorithms is *relaxation*
  - Idea: for all *v*, maintain upper bound d[*v*] on δ(s, v)

```
Relax(u,v,w) {
    if (d[v] > d[u]+w(u,v))
        then d[v] = d[u]+w(u,v);
}
```

Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;

   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v,w);

   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

Initialize d[], which will converge to shortest-path value δ

Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution
Under what condition do we get a solution?

```
Relax(u,v,w): if (d[v] > d[u]+w(u,v))
                 then d[v]=d[u]+w(u,v)
```

Bellman-Ford Algorithm

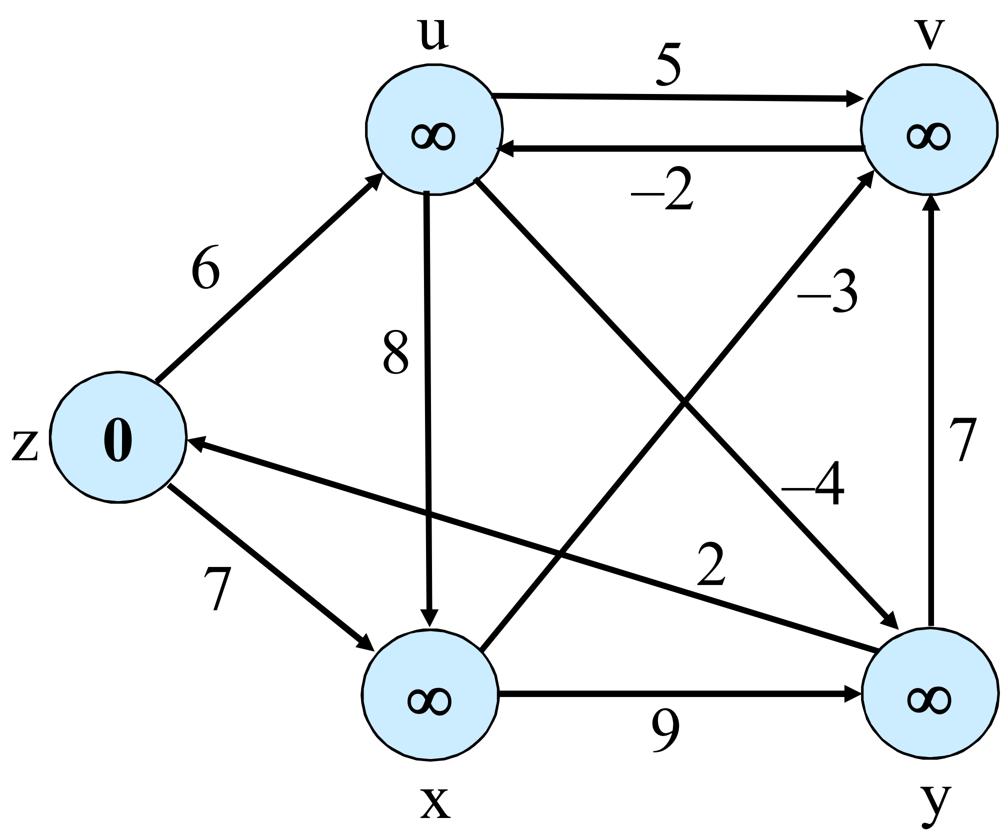
```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;

   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v,w);

   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

Q: What will be the running time?

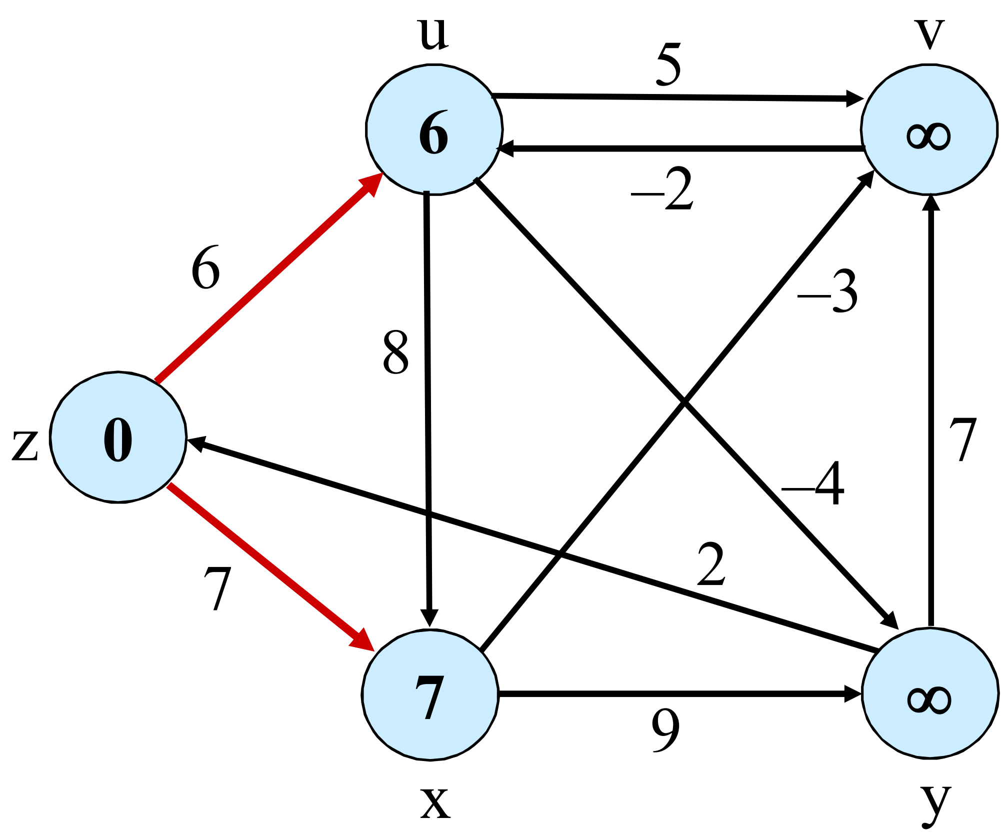
A: $O(V\,E)$

```
Relax(u,v,w): if (d[v] > d[u]+w(u,v))
                 then d[v]=d[u]+w(u,v)
```

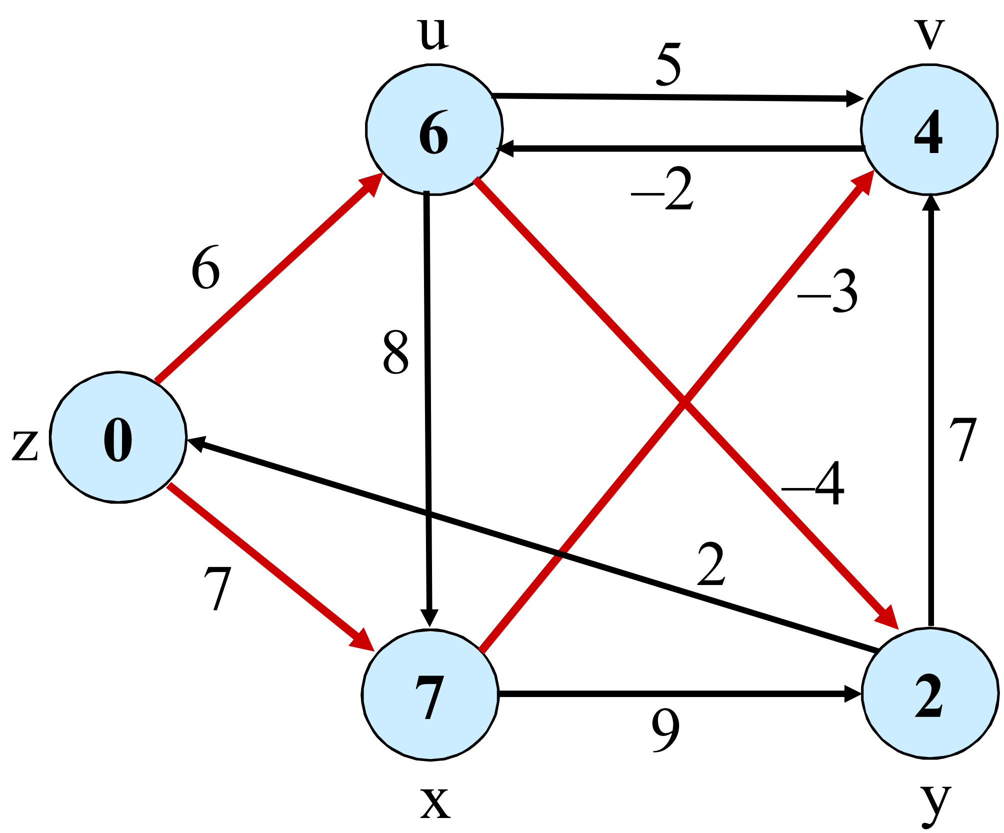
Bellman-Ford Algorithm: Example



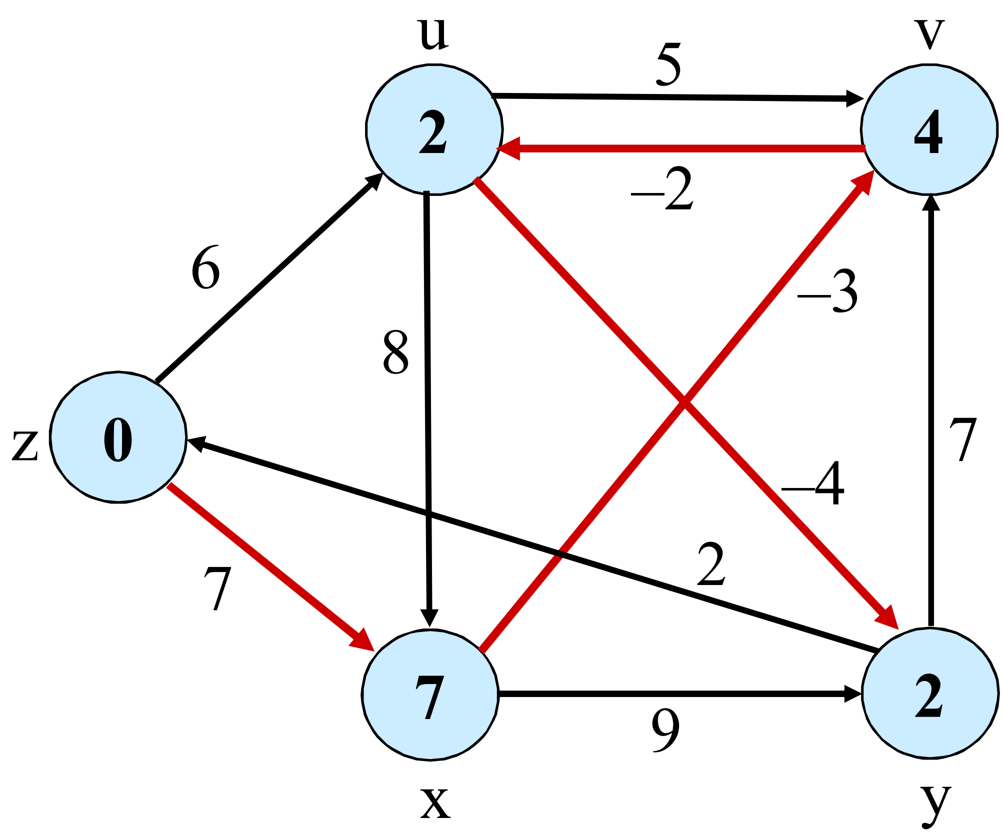
Bellman-Ford Algorithm: Example



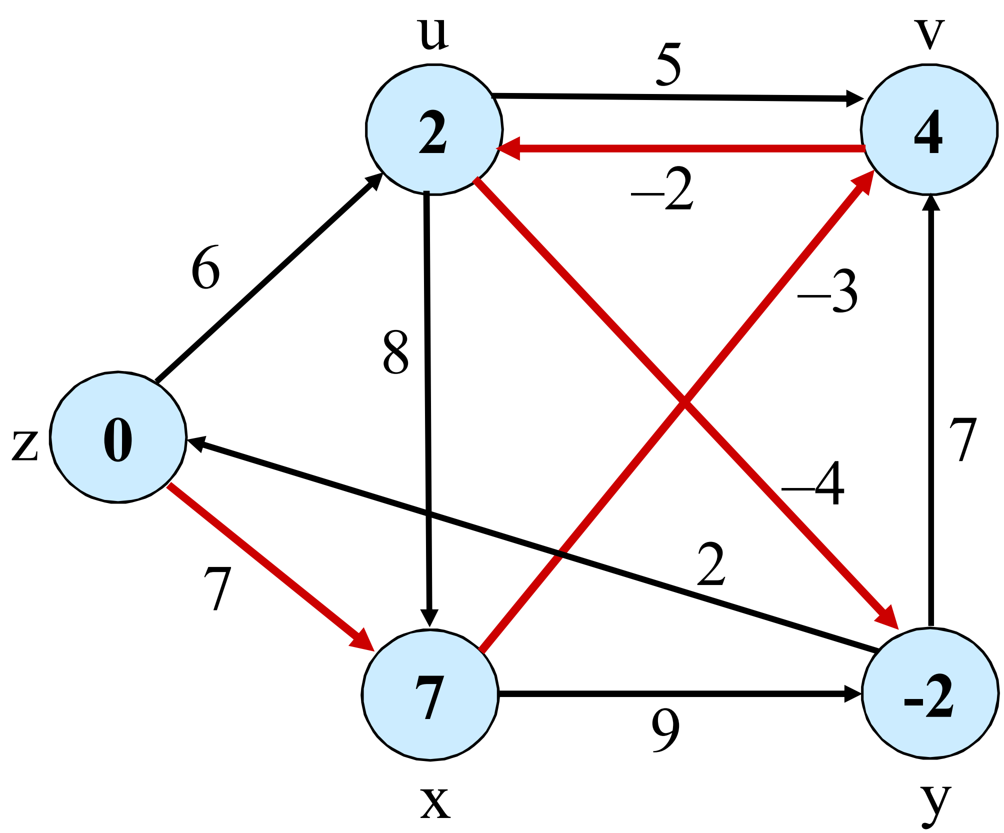
Bellman-Ford Algorithm: Example



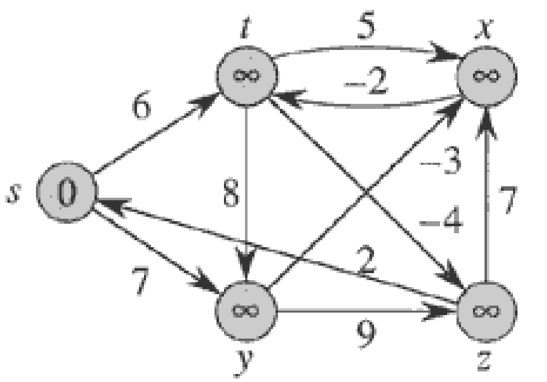
Bellman-Ford Algorithm: Example



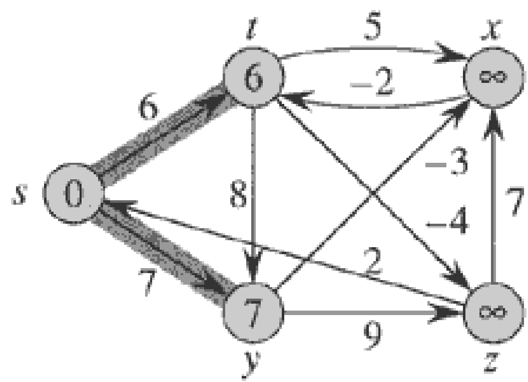
Bellman-Ford Algorithm: Example



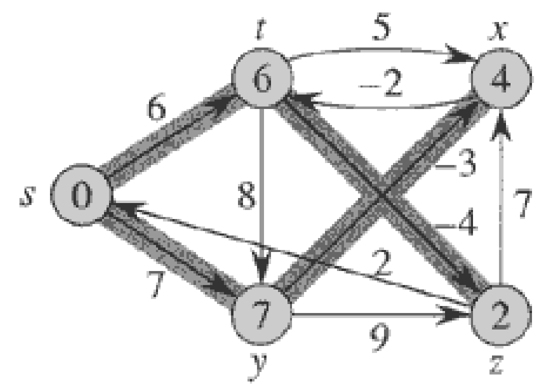
Bellman-Ford Algorithm: Example



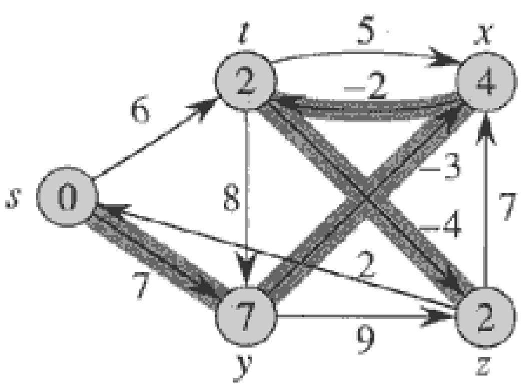
(a)



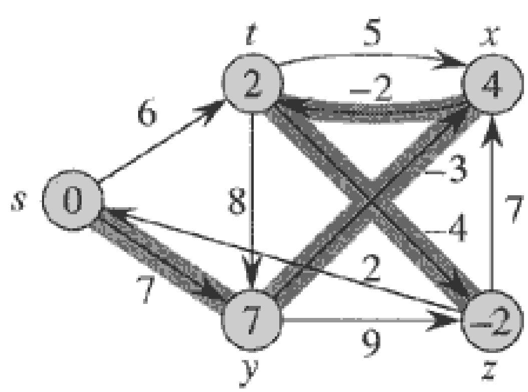
(b)



(c)



(d)



(e)

Correctness of Bellman-Ford Algorithm

We prove that Bellman-Ford Algorithm returns TRUE correctly, and FALSE correctly.

- Firstly: If G contains no negative-weight cycles that are reachable from s, then we show that the algorithm returns TRUE.

  - We prove that $d[v] = \delta(s, v)$, for all v, after $|V|-1$ passes
    - Lemma: $d[v] \geq \delta(s, v)$ always
      - Initially true
      - For a contradiction, let v be the first vertex for which $d[v] < \delta(s, v)$
      - Let u be the vertex that caused d[v] to change:

        $$d[v] = d[u] + w(u, v)$$
      - By triangle inequality, we have

        $$\delta(s, v) \leq \delta(s, u) + w(u, v)$$
      - Then

        $$\begin{aligned} d[v] &< \delta(s, v) \\ d[u] + w(u, v) &< \delta(s, u) + w(u, v) \\ d[u] &< \delta(s, u) \end{aligned}$$

        , a contradiction of v be the first vertex.

Correctness of Bellman-Ford Algorithm

- After |V|-1 passes, all *d* values are correct
  - Consider shortest path from s to v:
    $s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_k = v$, where $k \leq |V|-1$
    - Initially, d[s] = 0 is correct, and doesn't change
    - After 1 pass through edges, $d[v_1]$ is correct and doesn't change
    - After 2 passes, $d[v_2]$ is correct and doesn't change
    - …
    - Terminates in |V| - 1 passes.
  - By the path-relaxation property:
    - $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

Correctness of Bellman-Ford Algorithm

- At termination, we have for all edges $(u, v) \in E$
  - $$\begin{aligned} d[v] &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \quad \text{[by triangle inequality]} \\ &= d[u] + w(u, v) \end{aligned}$$

- So, none of the tests in Line 8 of Bellman-Ford algorithm returns FALSE.

  Therefore, it returns TRUE

Correctness of Bellman-Ford Algorithm

Conversely, suppose that graph G contains a negative-weight cycle that is reachable from the source s; let this cycle be $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$. Then,

$$(24.1) \quad \sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 .$$

Assume for the purpose of contradiction that the Bellman-Ford algorithm returns TRUE. Thus, $d[v_i] \le d[v_{i-1}] + w(v_{i-1}, v_i)$ for $i = 1, 2, \ldots, k$. Summing the inequalities around cycle c gives us

$$\begin{aligned} \sum_{i=1}^{k} d[v_i] &\le \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i) . \end{aligned}$$

Correctness of Bellman-Ford Algorithm

Since $v_0 = v_k$, each vertex in c appears exactly once in each of the summations $\sum_{i=1}^{k} d[v_i]$ and $\sum_{i=1}^{k} d[v_{i-1}]$, and so

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}] .$$

Moreover, $d[v_i]$ is finite for $i = 1, 2, \ldots, k$. Thus,

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) ,$$

which contradicts inequality (24.1). We conclude that the Bellman-Ford algorithm returns TRUE if graph G contains no negative-weight cycles reachable from the source, and FALSE otherwise.

Dijkstra's Algorithm

- If no negative edge weights, we can beat Bellman-Ford Algorithm
- Similar to breadth-first search
  - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
  - Use a priority queue keyed on d[v]

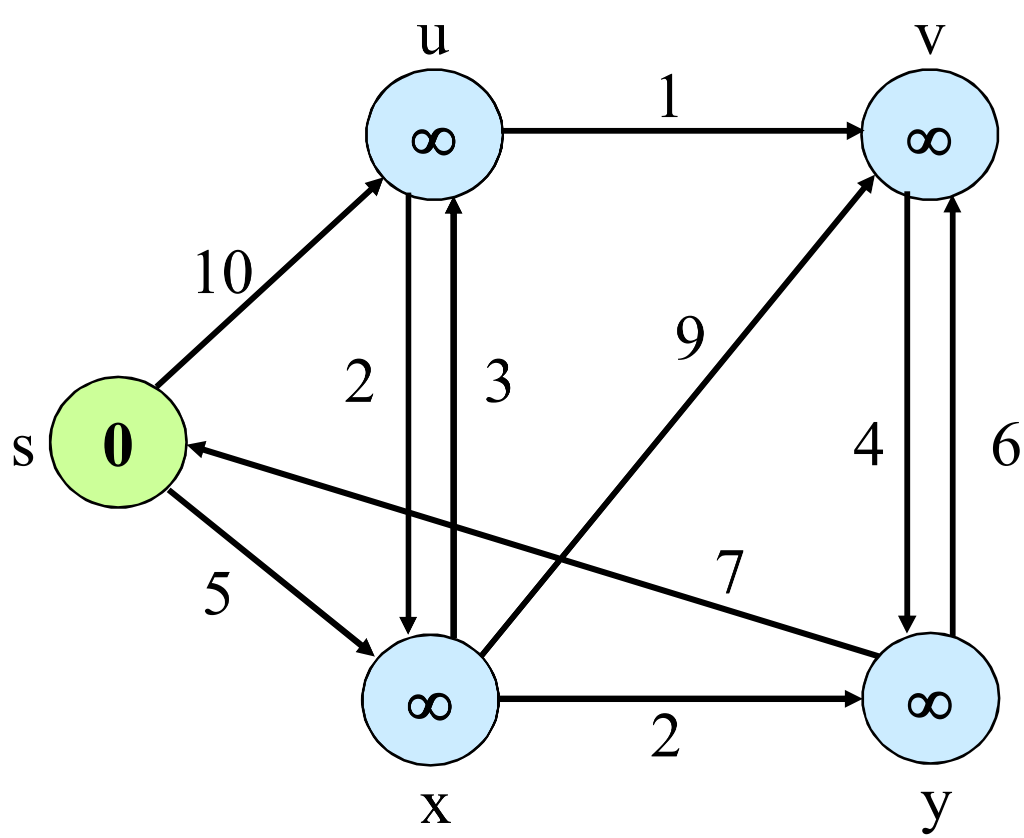
Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

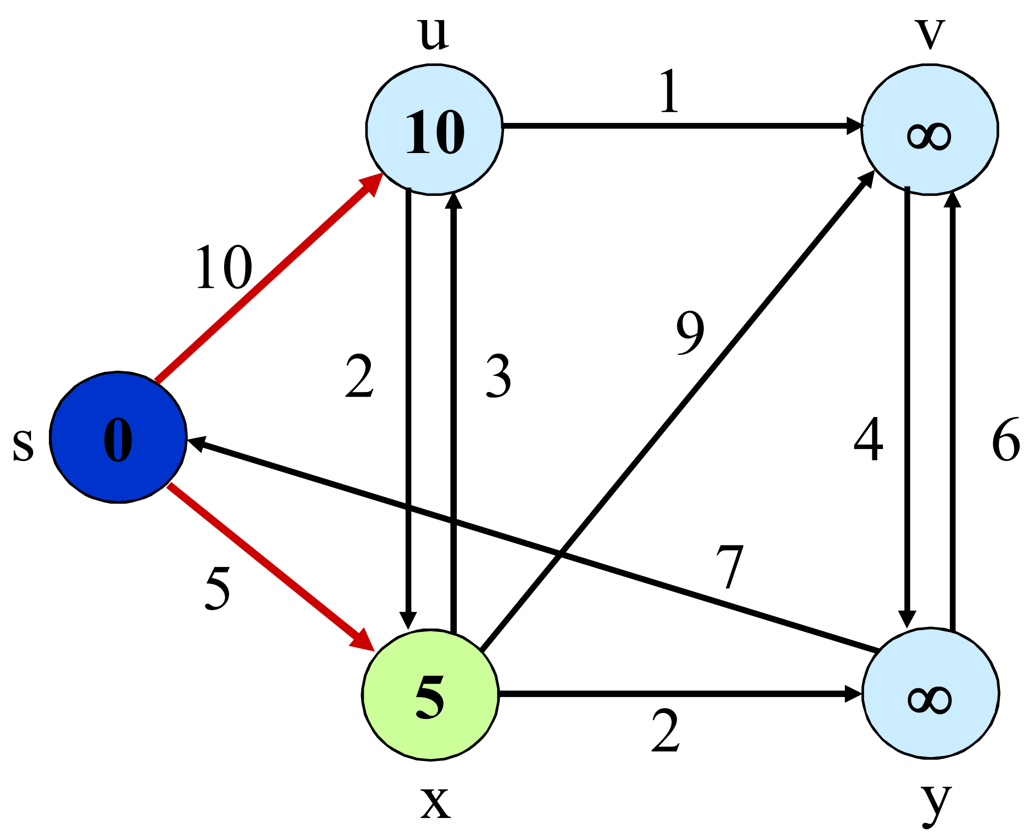
Relaxation Step

Note: this is really a call to Q->DecreaseKey()

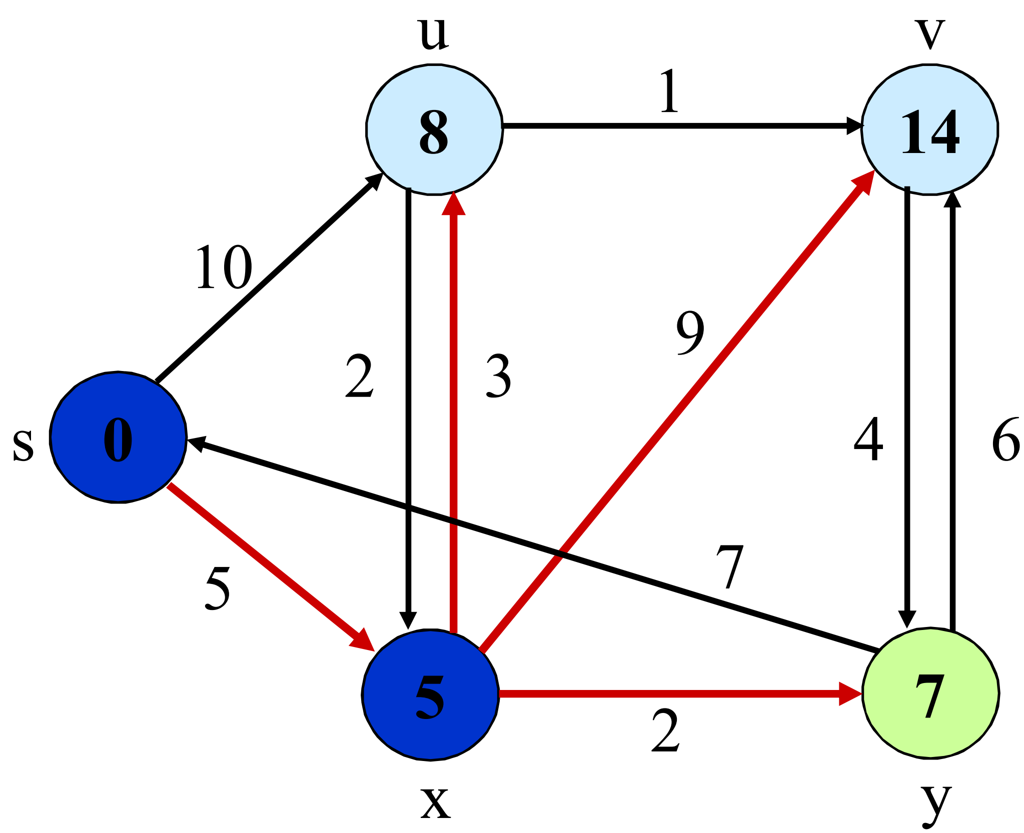
Dijkstra's Algorithm: Example



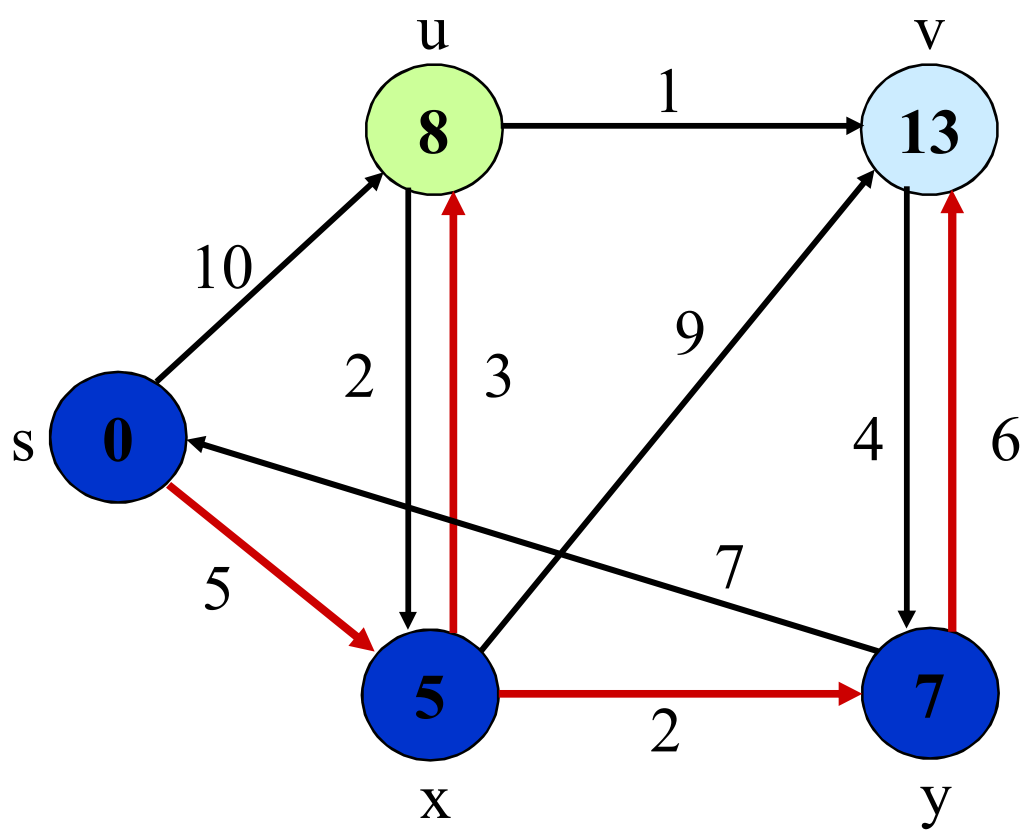
Dijkstra's Algorithm: Example



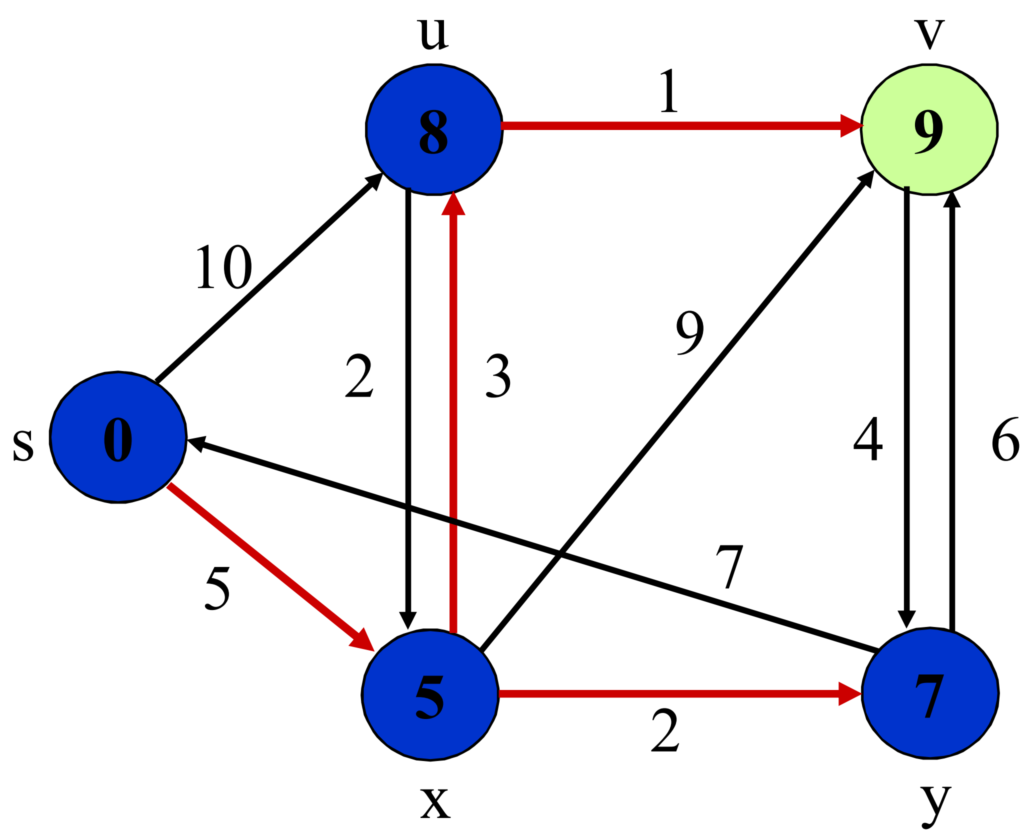
Dijkstra's Algorithm: Example



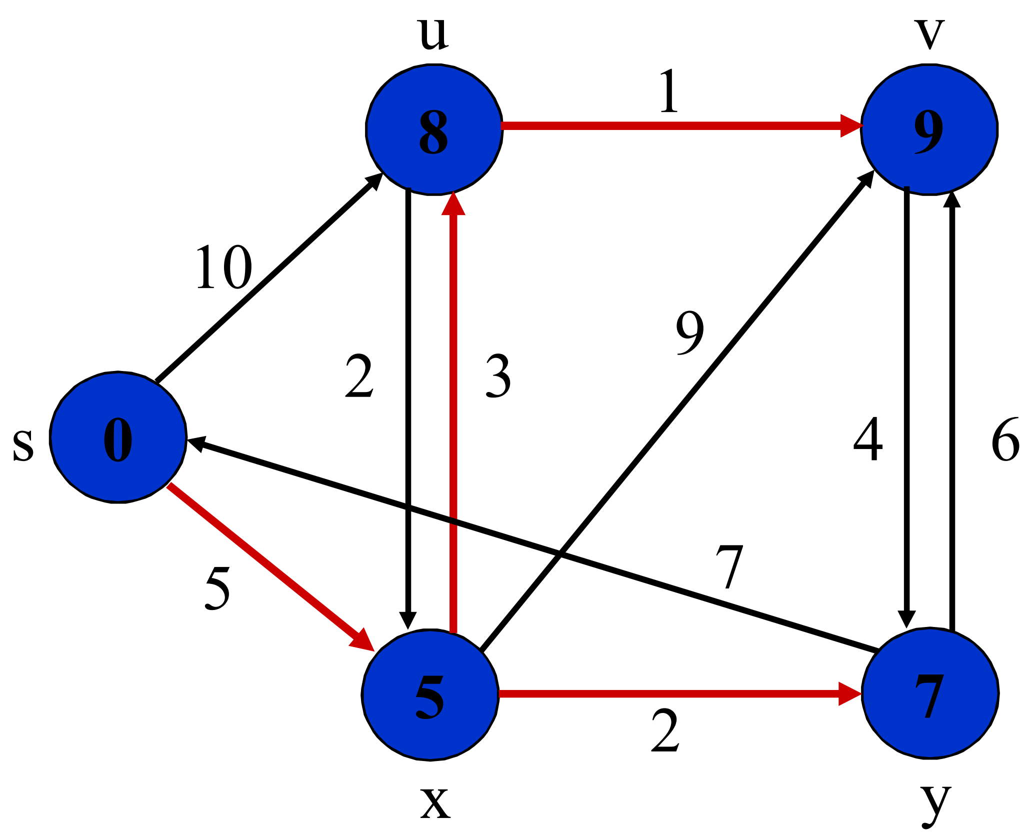
Dijkstra's Algorithm: Example



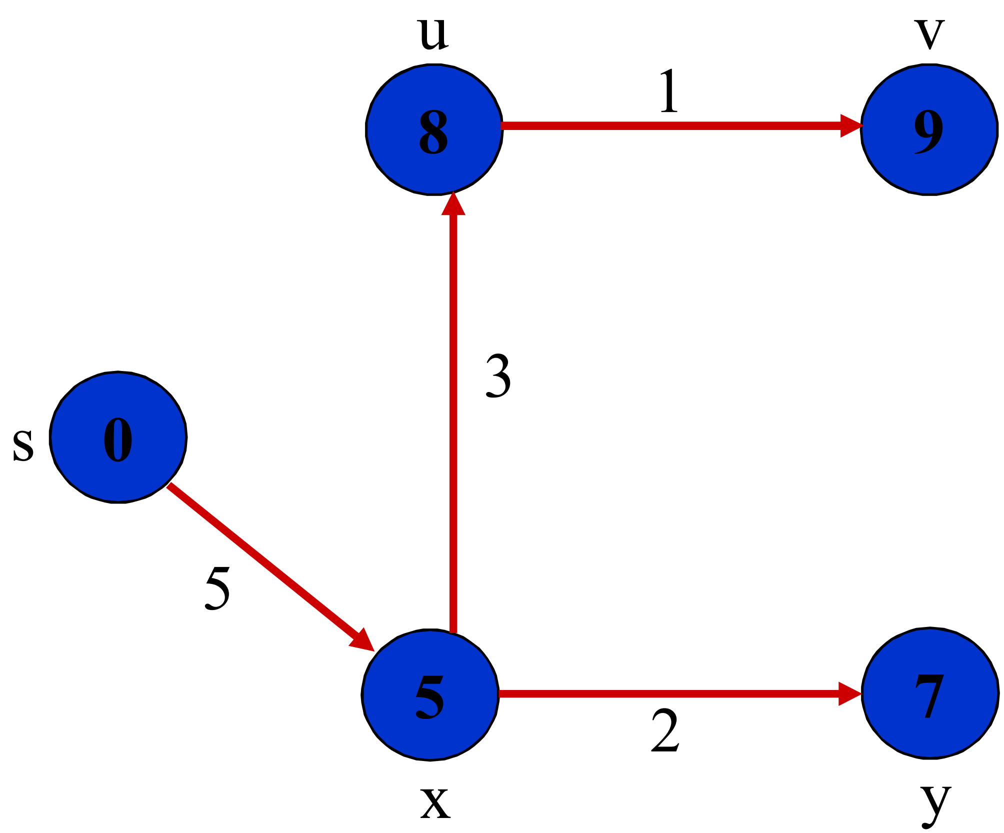
Dijkstra’s Algorithm: Example



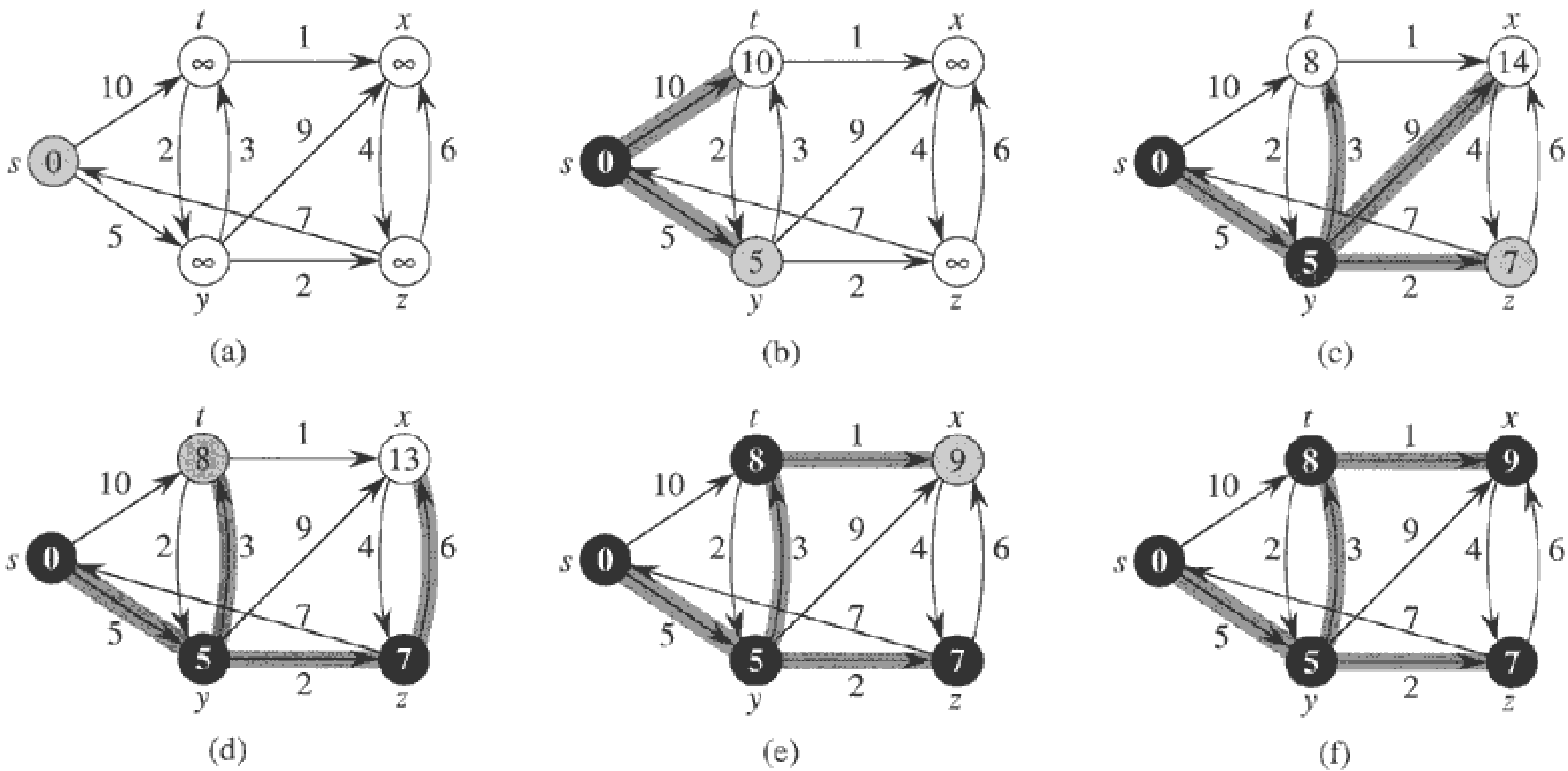
Dijkstra's Algorithm: Example



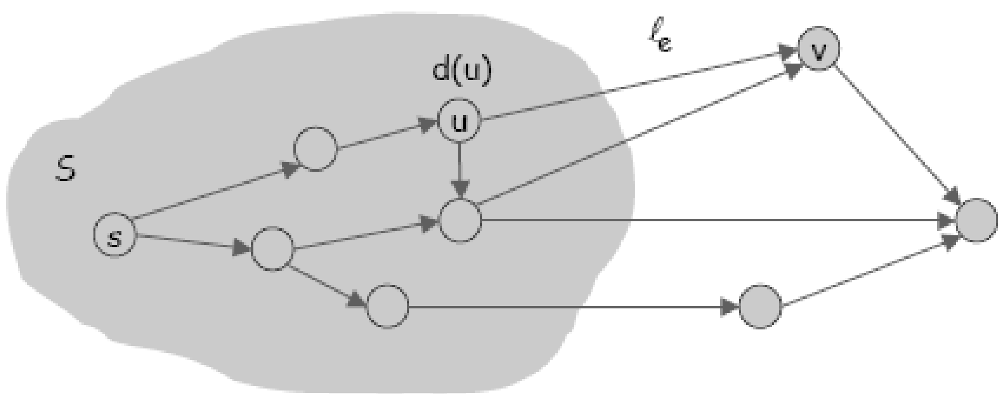
Dijkstra's Algorithm: Example



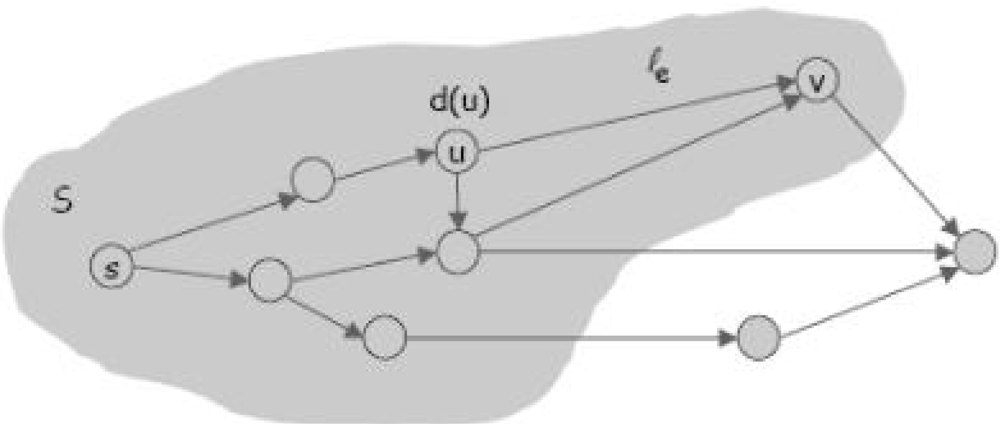
Dijkstra’s Algorithm: Example



Dijkstra’s Algorithm: Idea



Dijkstra's Algorithm: Idea



Dijkstra's Algorithm: Time Complexity

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

How many times is ExtractMin() called?

How many times is DecraseKey() called?

What will be the total running time?

Dijkstra's Algorithm: Time Complexity

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

A: *O*(E lg V) using binary heap for Q
Can achieve *O*(V lg V + E) with Fibonacci heaps

Dijkstra's Algorithm: Time Complexity

Using Array:

The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary array

- Good for dense graphs (many edges)

- $|V|$ vertices and $|E|$ edges
- Initialization $O(|V|)$
- While loop $O(|V|)$
  - Find and remove a min distance vertex $O(|V|)$
  - Potentially $|E|$ updates
    - Each update costs $O(1)$
- Reconstruct path $O(|E|)$

Total time $O(|V^2| + |E|) = O(|V^2|)$

Dijkstra's Algorithm: Time Complexity

Using Priority Queue:

For sparse graphs, Dijkstra's Algorithm can be implemented more efficiently by *priority queue*

- Initialization $O(|V|)$ using $O(|V|)$ buildHeap()
- While loop $O(|V|)$
  - Find and remove a min dist vertex needs $O(\log |V|)$ using $O(\log |V|)$ ExtractMin()
  - Potentially $|E|$ updates
    - Each update costs $O(\log |V|)$ using decreaseKey()
- Reconstruct path $O(|E|)$

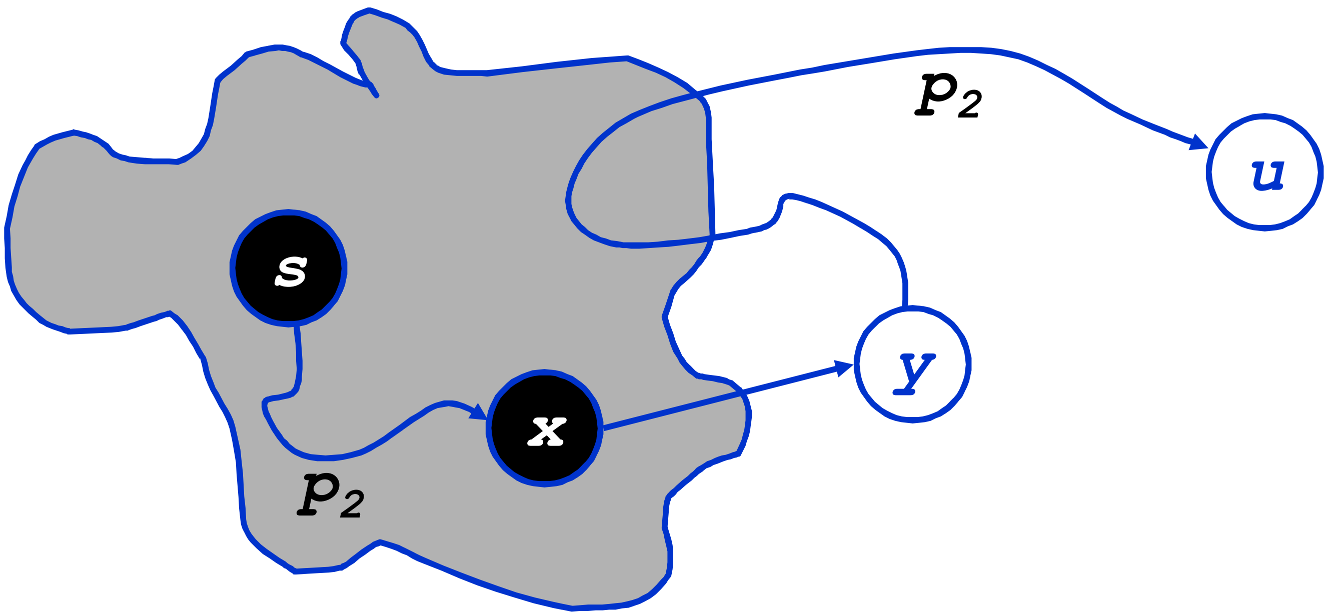
Total time $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$

Correctness of Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

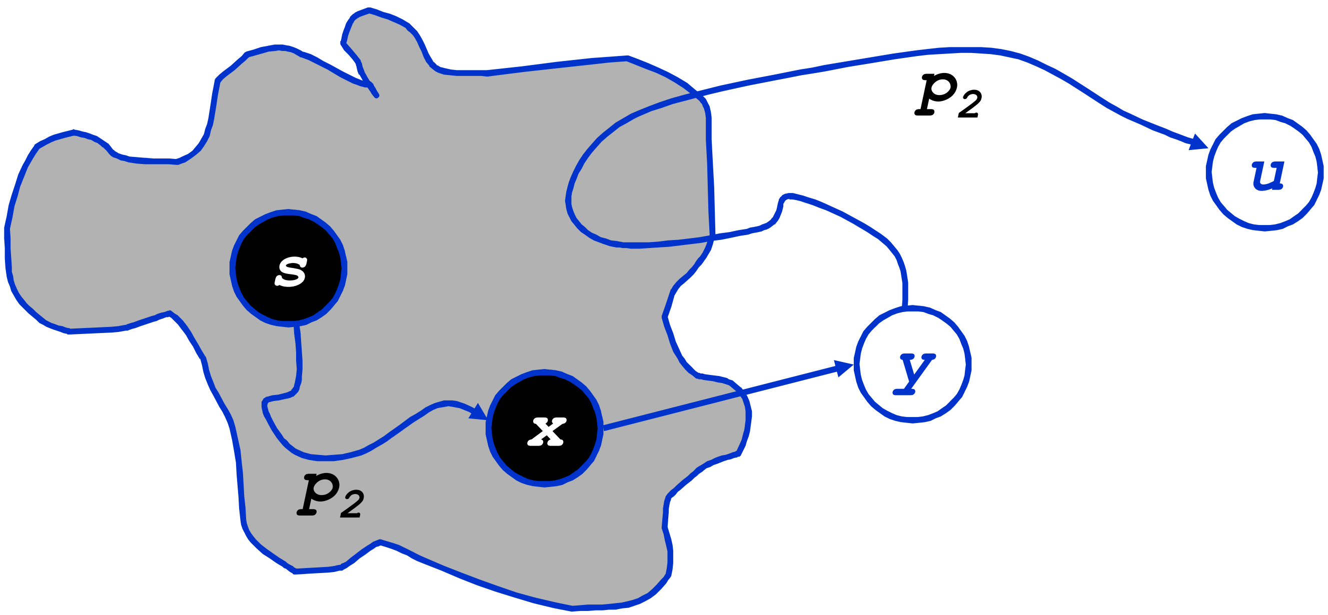
Correctness: we must show that when u is removed from Q, it has already converged

Correctness of Dijkstra's Algorithm



- Note that $d[v] \geq \delta(s,v)\ \forall v$
- Let u be the first vertex picked s.t. $\exists$ shorter path than d[u] $\Rightarrow d[u] > \delta(s,u)$
- Let y be the first vertex $\in V\text{-}S$ on actual shortest path from $s \rightarrow u$ $\Rightarrow d[y] = \delta(s,y)$
  - Because d[x] is set correctly for y's predecessor $x \in S$ on the shortest path, and
  - When we put x into S, we relaxed (x,y), giving d[y] the correct value

Correctness of Dijkstra's Algorithm



- Note that $d[v] \geq \delta(s,v)\ \forall v$
- Let u be the first vertex picked s.t. $\exists$ shorter path than d[u] $\Rightarrow d[u] > \delta(s,u)$
- Let y be the first vertex $\in$V-S on actual shortest path from $s \rightarrow u$ $\Rightarrow d[y] = \delta(s,y)$
- $d[u] > \delta(s,u)$
  $= \delta(s,y) + \delta(y,u)$ (*Optimal substructure property*)
  $= d[y] + \delta(y,u)$
  $\geq d[y]$
- But if $d[u] > d[y]$, wouldn't have chosen u. Contradiction.