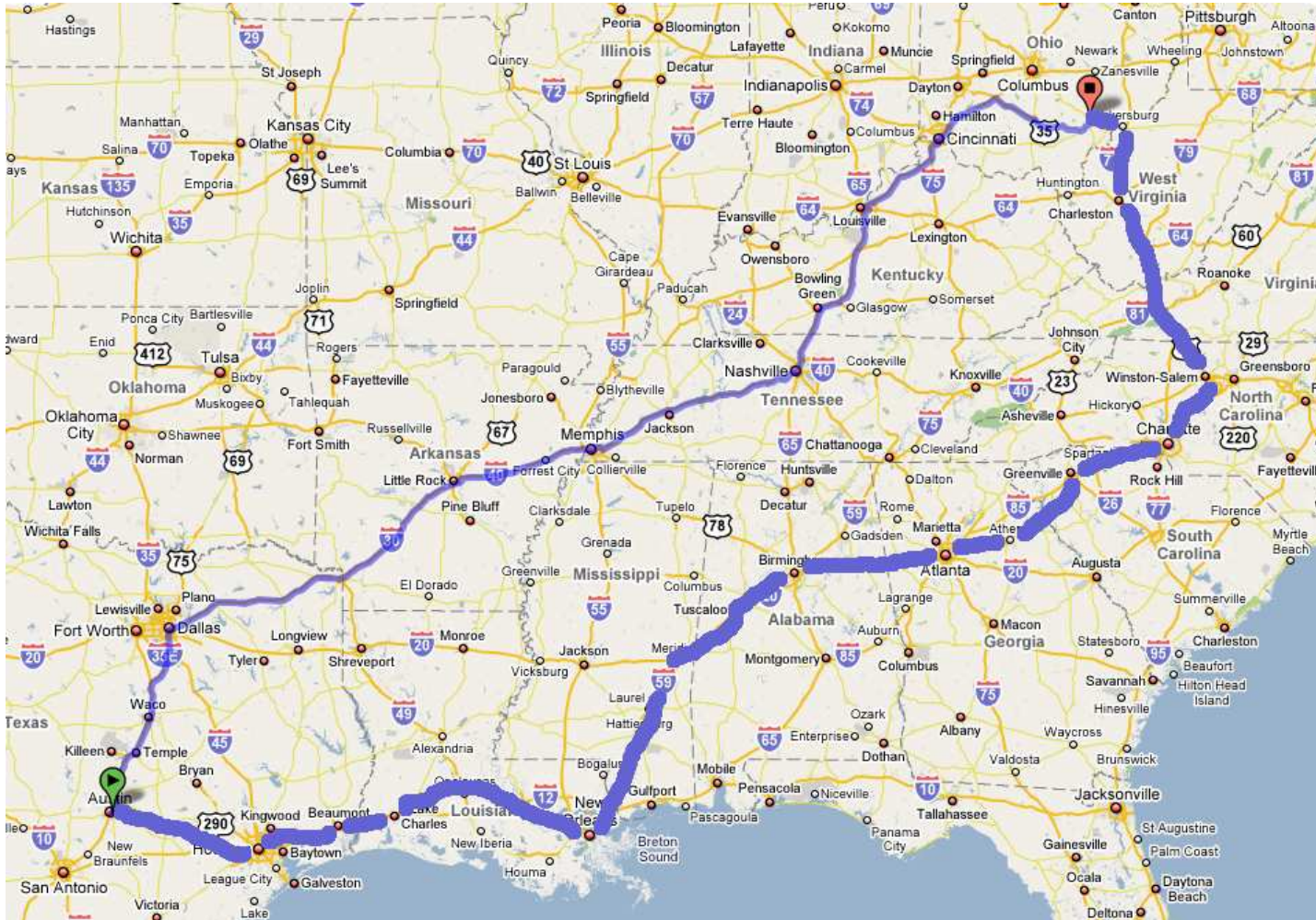


Driving directions

From Austin to Athens – many possible routes:



Problem: Single-Pair Shortest Path

Input: A directed graph $G = (V, E)$ where each edge (v_i, v_j) has a weight $w(i, j)$.

Output: A “shortest” path from u to v .

Weight of path: Given a path $p = \langle v_1, \dots, v_k \rangle$, its weight is:

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

“shortest” path = path of minimum weight. We use $\sigma(u, v)$ to denote this minimum weight.

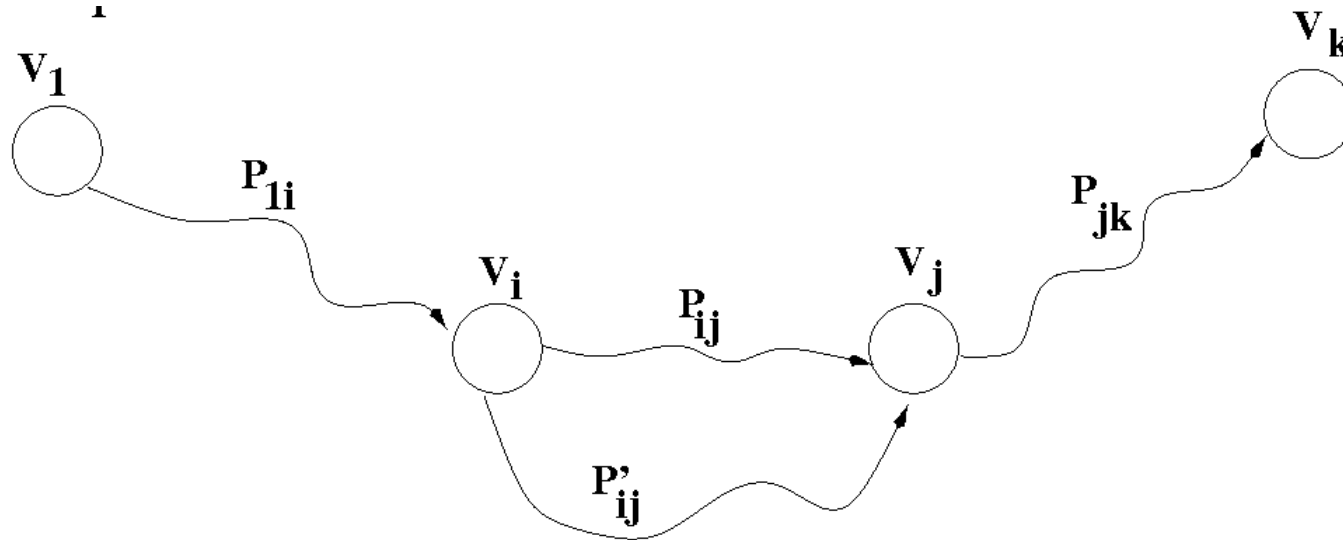
Different variants of shortest path problems

- **Single-pair shortest path (SPSP):**
Find a shortest path from u to v .
- **Single-source shortest paths (SSSP):**
Find a shortest path from source s to all vertices $v \in V$.
- **All-pairs shortest paths (APSP):**
Find a shortest path from u to v for all $u, v \in V$.
- No algorithm is known for computing a single-pair shortest path better than solving the SSSP problem in the worst case. So we will only focus on **SSSP**.

Properties of shortest paths (1): Optimal Substructure

Lemma 24.1: Subpaths of shortest paths are shortest paths.

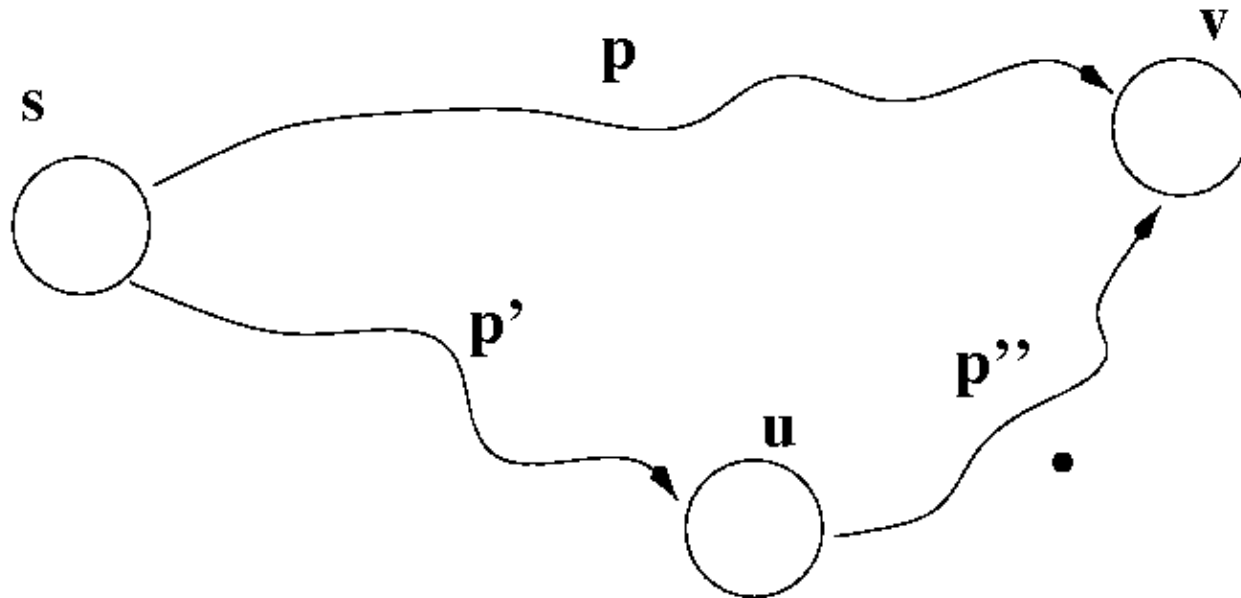
Proof: Cut and paste:



If some subpath were not a shortest path, we could substitute the shorter subpath and create an even shorter total path.

Properties of shortest paths (2): Triangle Inequality

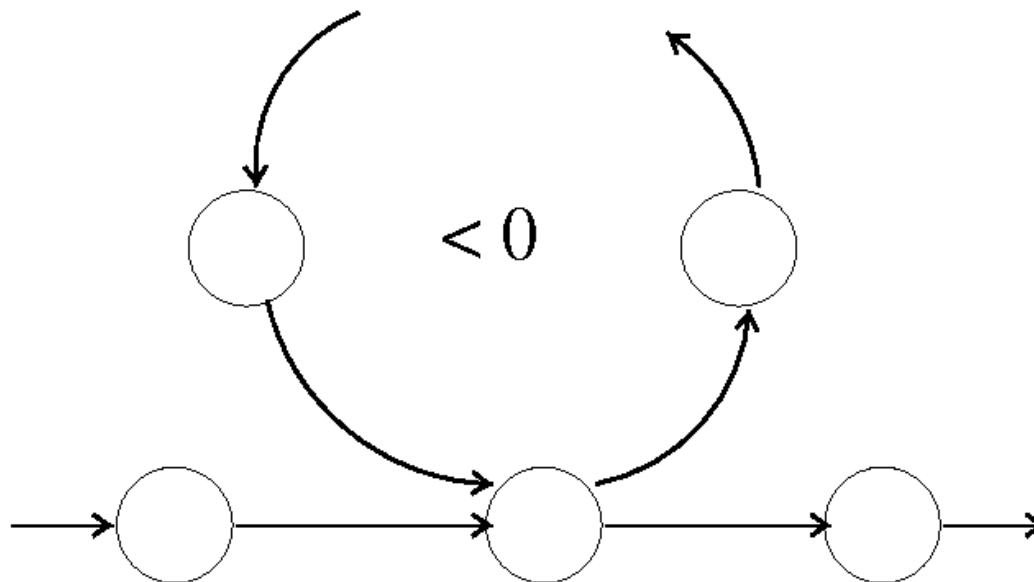
$$\sigma[s, v] \leq \sigma[s, u] + \sigma[u, v] \quad (2)$$



Is shortest-path well-defined?

Negative weight cycle \Rightarrow no shortest path.

Argument: path can be shortened by traversing a negative cycle.



Dijkstra's algorithm: Idea

- Maintain a set S of vertices whose final shortest-path weights from the source s have already been determined (S is just like the set A in Prim's algorithm).
- The set S initially contains only the source s .
- The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate (like the *key* in Prim's algorithm), adds u to S , and *relaxes* all edges leaving u .
- **Input requirement:** $w(u, v) \geq 0$, for all $(u, v) \in E$.

Dijkstra's algorithm: Data Structures

Data Structures:

S : Vertices whose shortest paths have already been determined.

$V - S$: Remainder.

d : $d[v]$ tells the current best estimate of shortest path to the source.

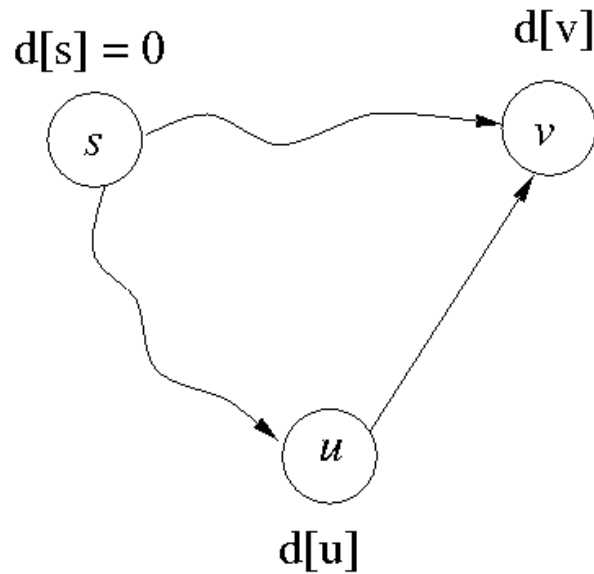
π : $\pi[v]$ tells the predecessor for vertex v in the current shortest path.

Dijkstra's Algorithm: Auxiliary Functions

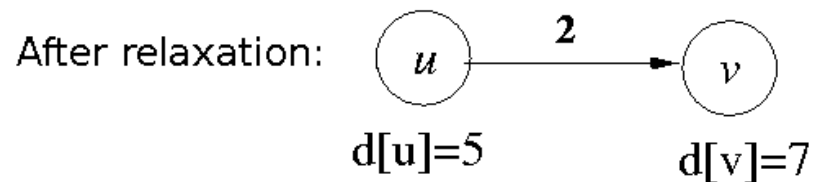
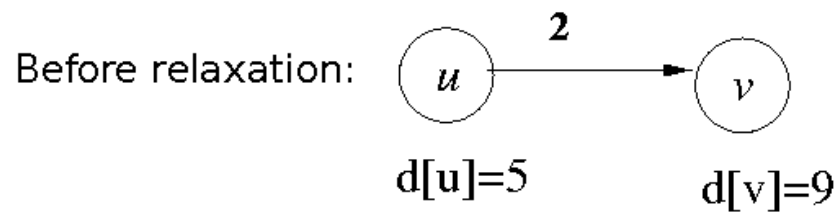
InitializeSingleSource(G, s) {
 for each vertex $v \in V$ do
 $d[v] = \infty$
 $\pi[v] = nil$
 $d[s] = 0$
}

Relax(u, v, w) {
 if $d[v] > d[u] + w(u, v)$ then
 $d[v] = d[u] + w(u, v)$
 $\pi[v] = u$
}

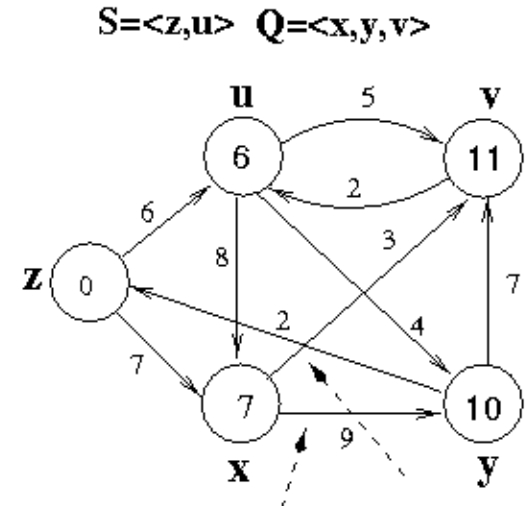
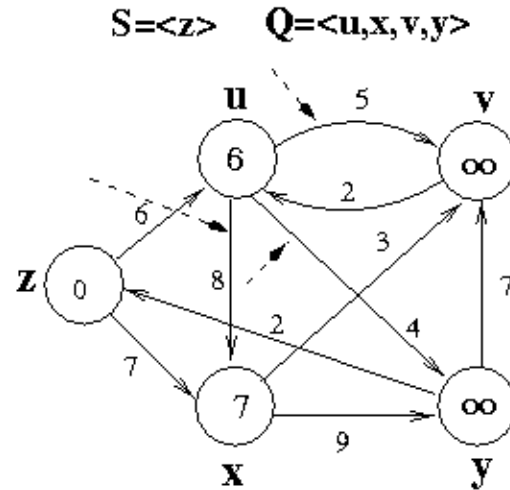
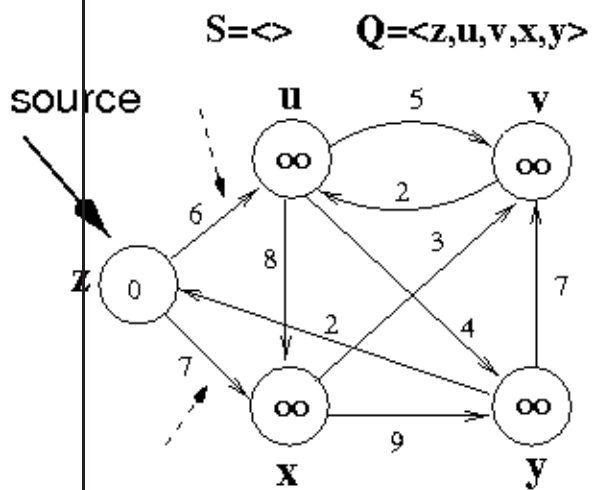
Relaxation



if ($d[v] > d[u] + w(u, v)$)
 $d[v] = d[u] + w(u, v)$
else donot change $d[v]$

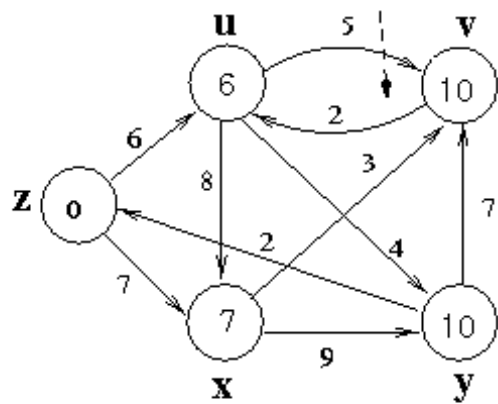


Dijkstra's algorithm: Example

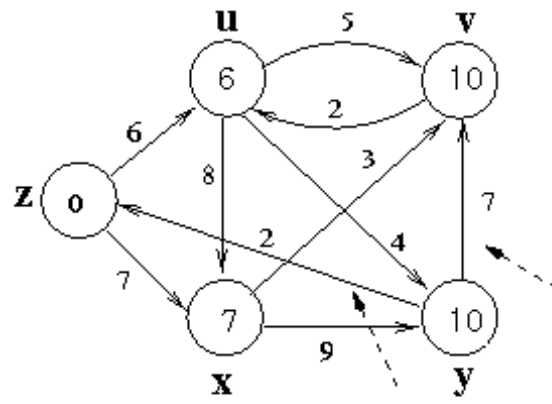


Example

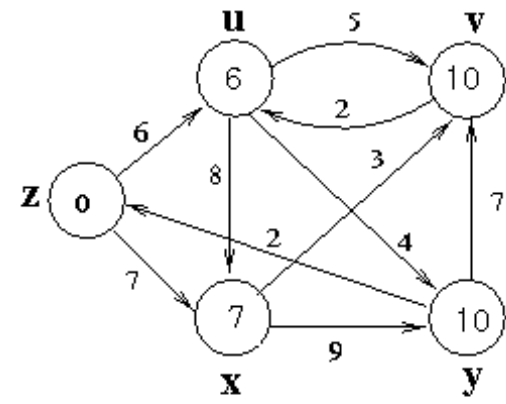
$S = \langle z, u, x \rangle$ $Q = \langle v, y \rangle$



$S = \langle z, u, x, v \rangle$ $Q = \langle y \rangle$



$S = \langle z, u, x, v, y \rangle$ $Q = \langle \rangle$



Dijkstra's algorithm

```
DIJKSTRA ( $G(V, E), w, s$ )      /*  $s$  is the source */
1  InitializeSingleSource( $G, s$ );

2   $S := \emptyset$ ;              /* Make  $S$  empty */
3   $Q := V$ ;                    /* put all vertices into
                               a Priority Queue */
4  while  $Q$  is not empty
5       $u := \text{Extract-Min}(Q)$ ; /* get the vertex which is
                                   closest to the source  $s$ , and
                                   remove it from the queue */
6       $S := S \cup u$ ;          /* Add  $u$  to  $S$  */
7      for each  $v \in \text{Adj}[u]$  /* update the  $d_s$  to  $s$  */
8          Relax ( $u, v, w, Q$ );
```

Similarity with Prim's algorithm

```
MST-PRIM ( $G(V, E), w, r$ ) /*  $r$  is the arbitrarily
                               selected starting point */
1   for each  $u \in V$ 
2        $key[u] := \infty$ ;

3    $key[r] := 0$ ;           /* the first to be picked into  $V_A$  */
4    $Q := V$ ;             /* put all vertices into a PQ */

5   while  $Q$  is not empty
6        $u := \text{Extract-Min}(Q)$ ; /* extract the vertex which
                                   is closest to the tree  $A$  */
7       for each  $v \in \text{Adj}[u]$  /* update the dist. to  $A$  */
8           if  $v \in Q$  and  $w(u, v) < key[v]$ 
9                $key[v] := w(u, v)$ 
```

Complexity depends on priority queue implementation

Use a **Binary Heap** to implement the min-priority queue.

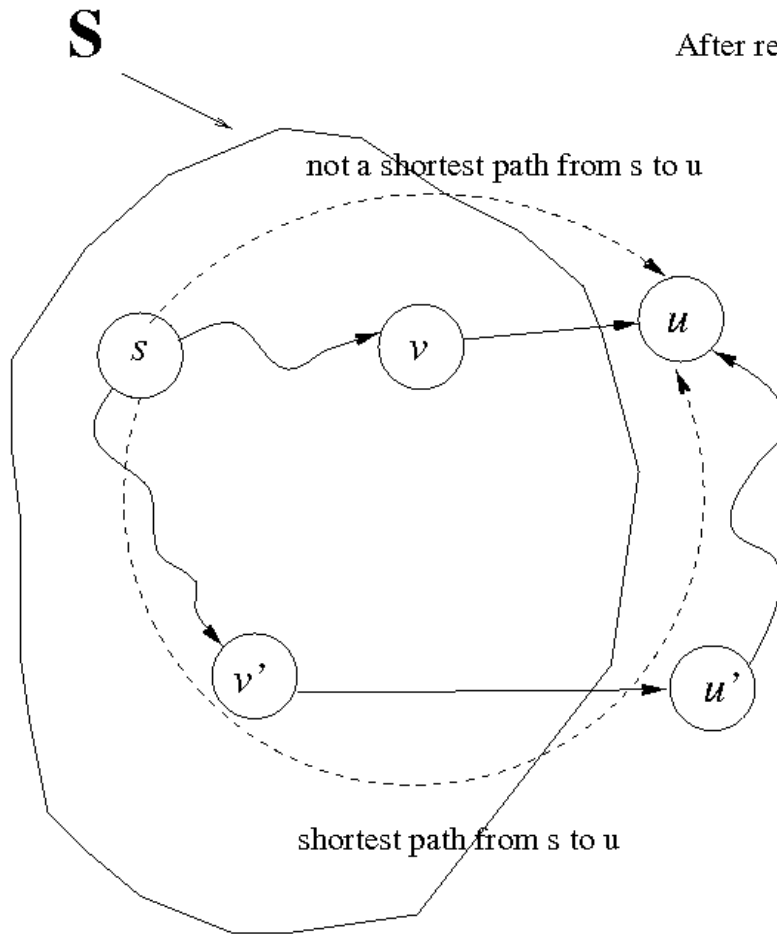
DIJKSTRA ($G(V, E), w, r$)

- 1 InitializeSingleSource(G, s); — $\Theta(|V|)$
- 2 $S := \emptyset$; — $\Theta(1)$
- 3 $Q := V$; — Build-Min-Heap: $O(|V|)$
- 4 while Q is not empty — $|V|$ times
- 5 $u := \text{Extract-Min}(Q)$; — Extract-Min: $O(\lg|V|)$
- 6 $S := S \cup u$; — $\Theta(1)$
- 7 for each $v \in \text{Adj}[u]$ — $O(|E|)$
- 8 $\text{relax}(u, v, w, Q)$; /*Decrease-Key */ $O(\lg(|V|))$

Correctness of Dijkstra's algorithm

- We need to show that when the algorithm finishes, $d[u] = \sigma[s, u]$ for every u in V .
- We'll show that when u is inserted to S , $d[u] = \sigma[s, u]$.

Assume: $d[u] > \sigma[s, u]$ — Proof by contradiction



What is the value of $d[u]$?

After relaxation: $d[u] \leq d[v] + w(v, u) = \sigma[s, v] + w(v, u)$

What is the value of $d[u']$?

after relaxation: $d[u'] \leq d[v'] + w(v', u')$
 $= \sigma[s, v'] + w[v', u']$

Of course, $d[u'] < \sigma[s, u]$

Since $d[u] > \sigma[s, u] \implies d[u'] < d[u]$

Contradiction!!

Priority Queue: $Q \langle u, \dots, u' \dots \rangle$ (note: $d[u] < \dots < d[u'] < \dots$)

Dijkstra's algorithm

- Where are we using the assumption that the weights are ≥ 0 ?

A historic note:

- Prim's algorithm was invented in 1957.
- Dijkstra's algorithm was invented in 1959, without the use of a priority queue.