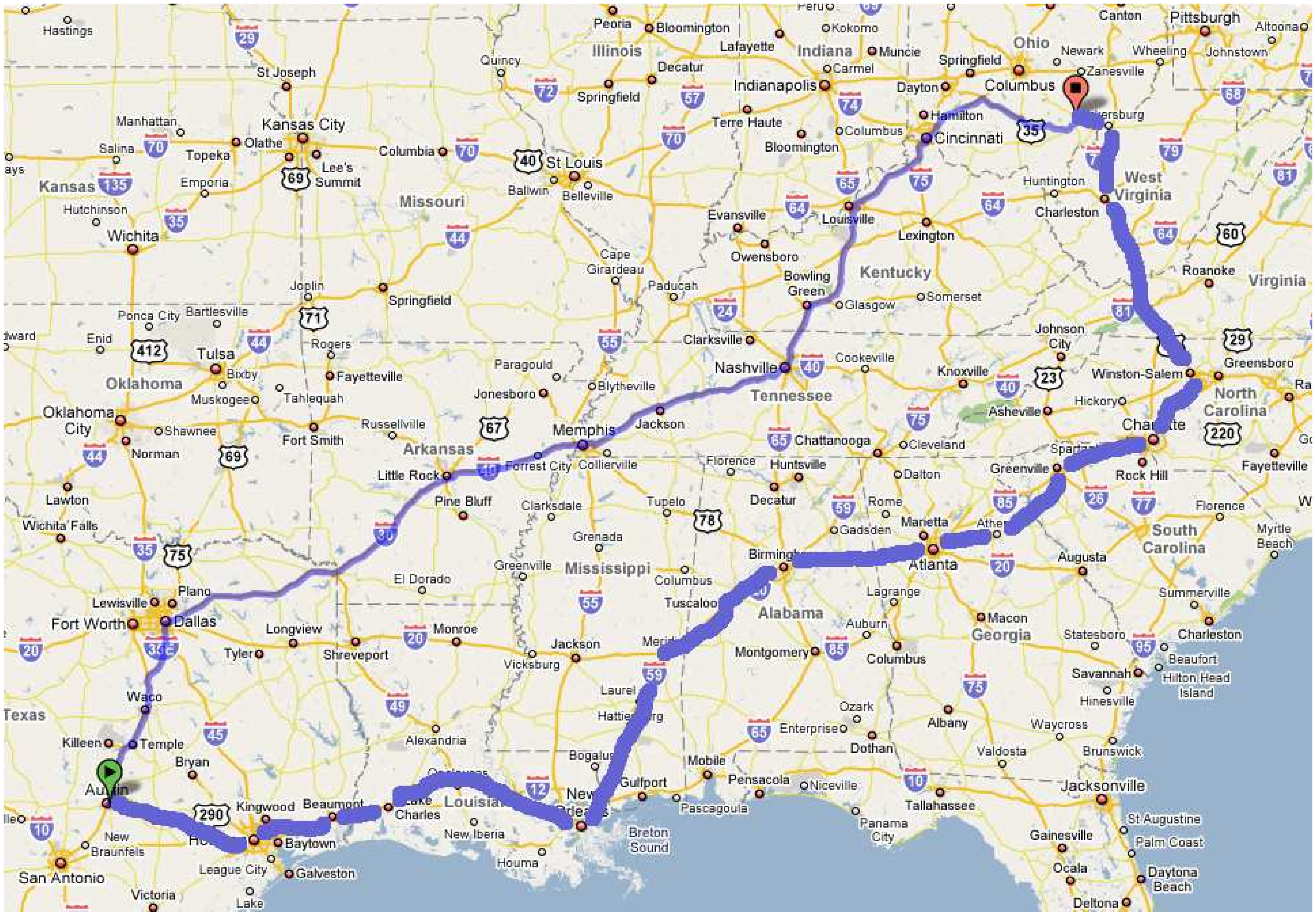
Driving directions

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 $p =< v_1, ..., v_k >$, 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 $v \in V$.

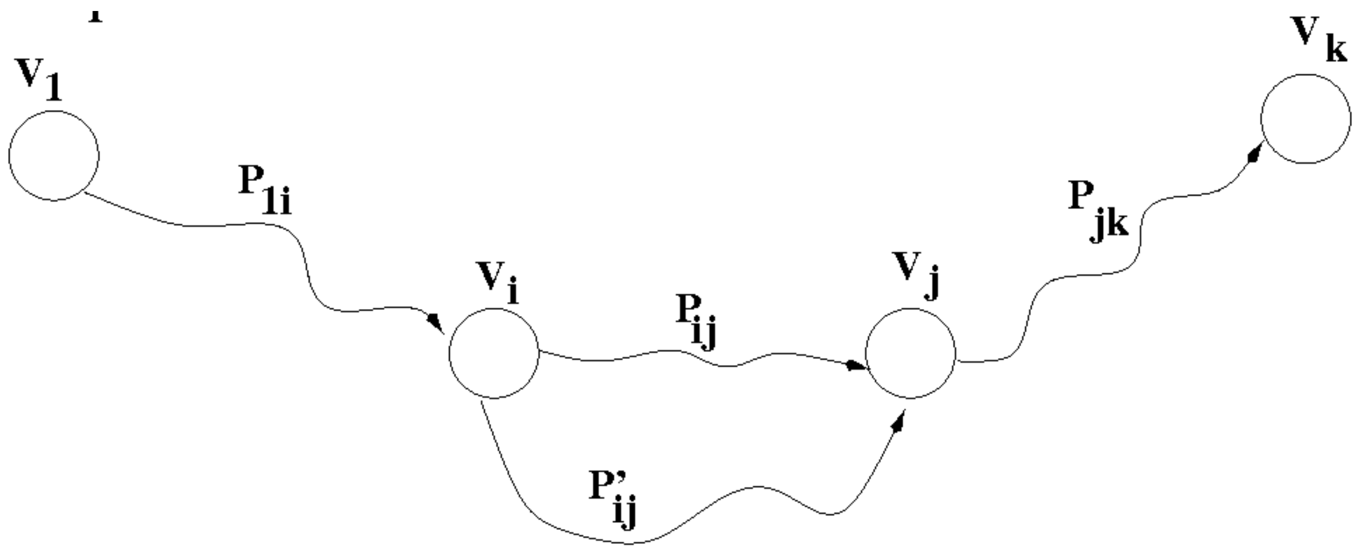
- **All-pairs shortest paths (APSP):**
  Find a shortest path from u to v for all $u, v \in V$.

- 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: **<u>Subpaths</u> of shortest paths are shortest paths.**

Proof: Cut and paste:

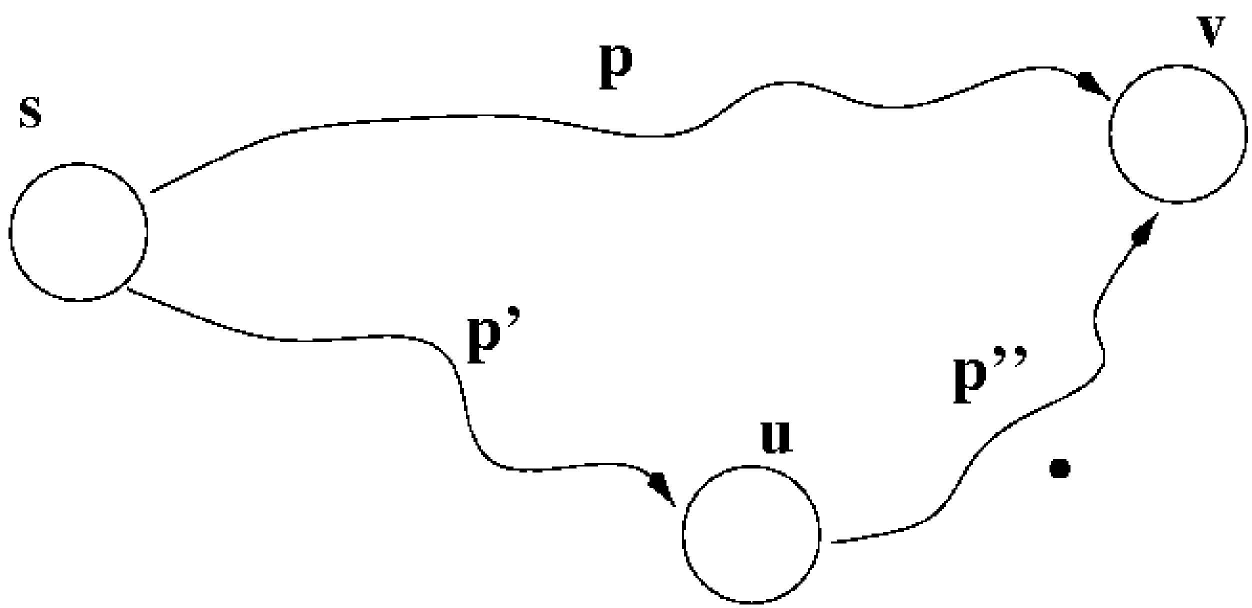


If some subpath were not a shortest path, we could substitute

the shorter subpath and create an even shorter total path.

Properties of shortest paths (2): Triangle Inequality

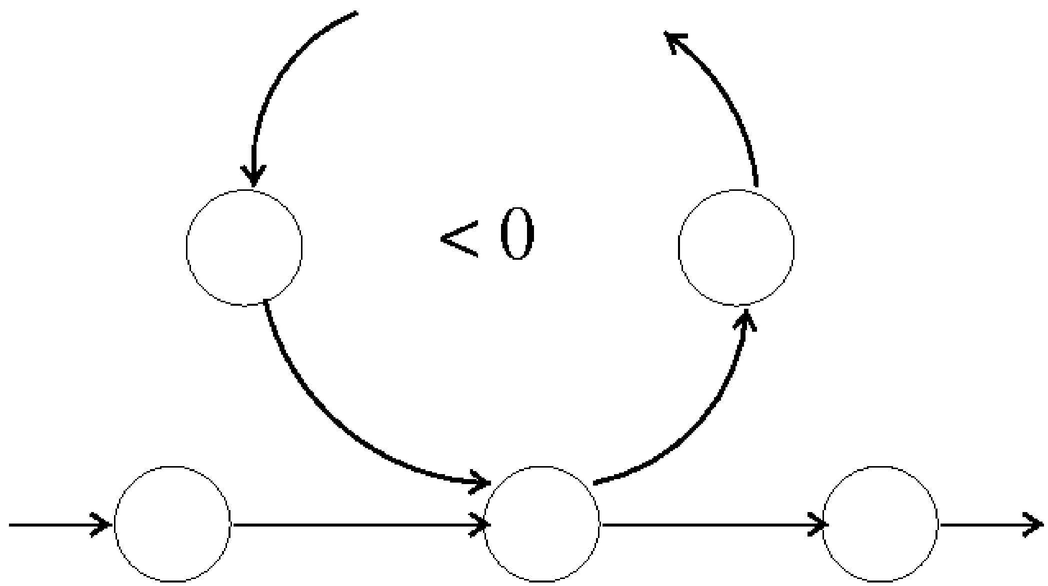
$$\sigma[s, v] \leq \sigma[s, u] + \sigma[u, v] \quad (2)$$



Is shortest-path well-defined?

Negative weigth cycle ⇒ 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 repeteadly 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**)** $\{$

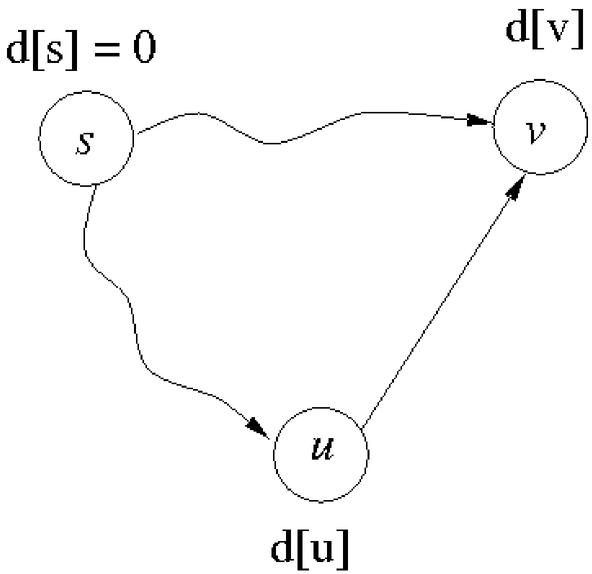
if $d[v] > d[u] + w(u, v)$ then

$d[v] = d[u] + w(u, v)$

$\pi[v] = u$

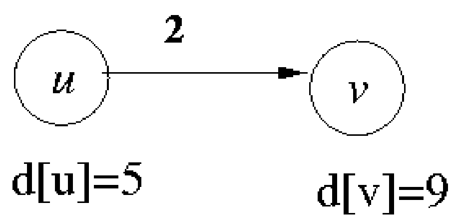
$\}$

Relaxation

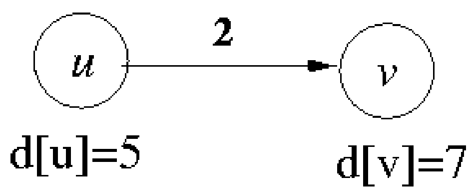



```
if (d[v] > d[u] + w(u, v))
    d[v] = d[u] + w(u, v)
else donot change d[v]
```

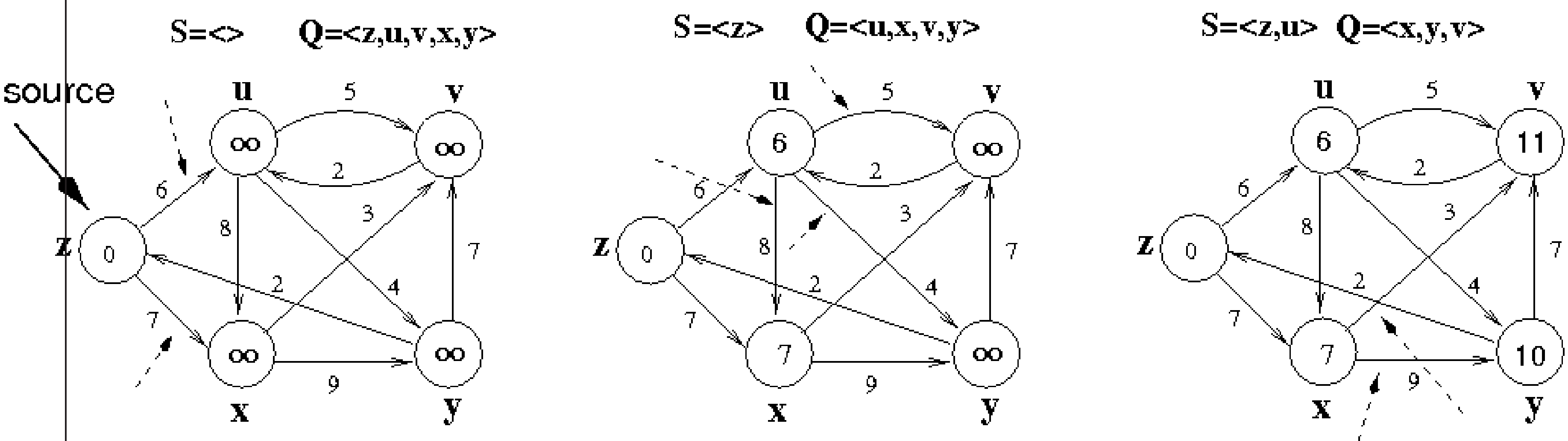
Before relaxation:



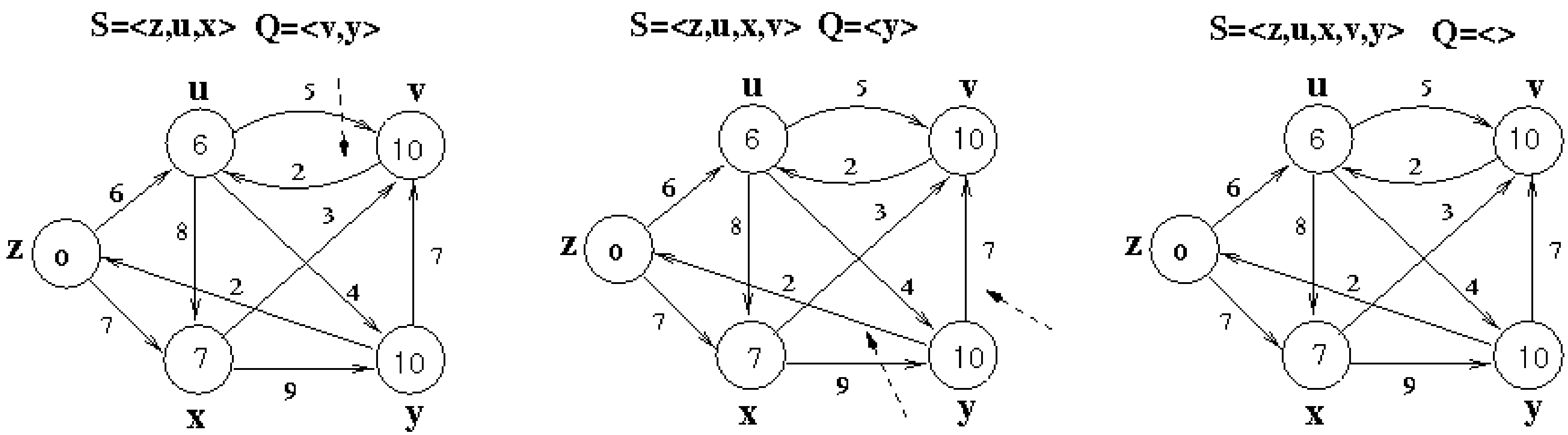
After relaxation:



Dijkstra's algorithm: Example



Example



Dijkstra's algorithm

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

    S := ∅;                        /* Make S empty */
    Q := V;                        /* put all vertices into
                                      a Priority Queue */
    while Q is not empty
        u := Extract-Min(Q);       /* get the vertex which is
                                      closest to the source s, and
                                      remove it from the queue */
        S := S ∪ u;                /* Add u to S */
        for each v ∈ Adj[u]        /* update the ds to s */
            Relax (u, v, w, Q);
```

Similarity with Prim's algorithm

```
MST-PRIM (G(V,E), w, r)   /* r is the arbitrarily
                             selected starting point */
    for each u ∈ V
        key[u] := ∞;

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

    while Q is not empty
        u := Extract-Min(Q);      /* extract the vertex which
                                     is closest to the tree A */
        for each v ∈ Adj[u]       /* update the dist. to A */
            if v ∈ Q and w(u,v) < key[v]
                key[v] := w(u,v)
```

Complexity depends on priority queue implementation

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

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

InitializeSingleSource(G, s); — $\Theta(|V|)$

$S := \emptyset$; — $\Theta(1)$
$Q := V$; — Build-Min-Heap: $O(|V|)$

while Q is not empty — $|V|$ times
    $u :=$ Extract-Min(Q); — Extract-Min: $O(lg|V|)$
    $S := S \cup u$; — $\Theta(1)$

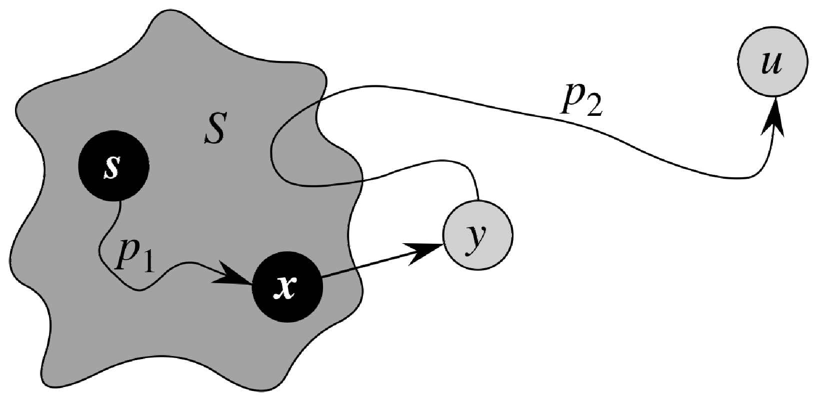
    for each $v \in$ Adj[u] — $O(|E|)$
        relax(u, v, w, Q); /*Decrease-Key */ $O(lg(|V|)$

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



Let u be the first vertex such added to S s.t. $d[u] > \sigma[s, u]$.

When x was added to S, $d[x] = \delta(s, x)$ and edge (x, y) was relaxed => $d[y] \leq \delta(s, x) + w(x, y) \leq \delta(s, u)$

Thus, $d[y] \leq \delta(s, u) \leq d[u]$. But $d[u] \leq d[y]$ because u was chosen to be added before y => $d[u] = \delta(s, u)$ => contradiction!

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.