香港中文大學（深圳）
The Chinese University of Hong Kong

CSC3100 Data Structures
Lecture 22: Graph shortest path

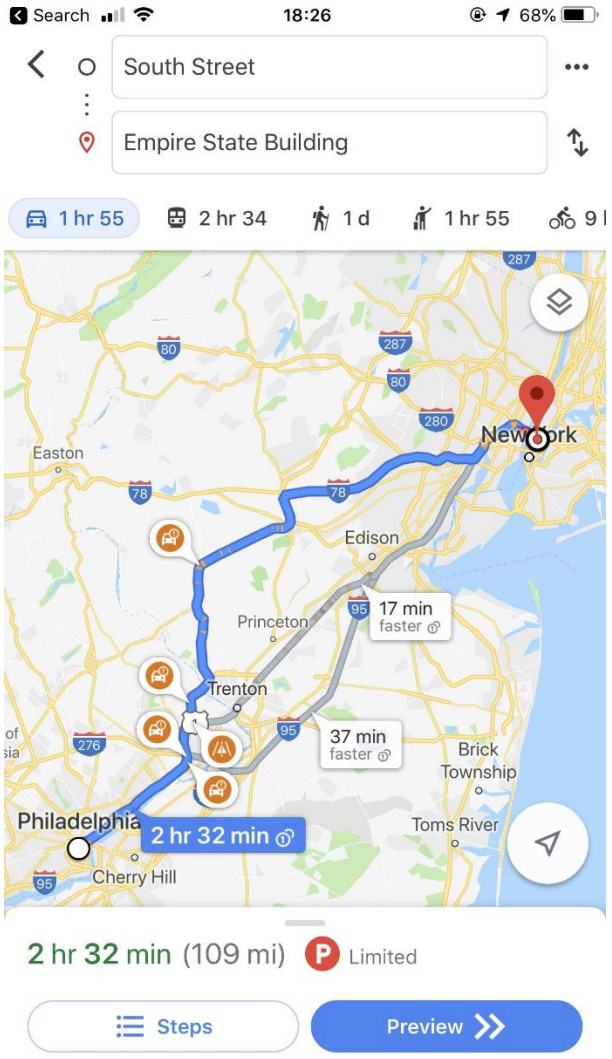
Yixiang Fang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Outline

- We focus on weighted graphs
- Graphs with non-negative weights
  - Single-Source Shortest Path: Dijkstra's algorithm
- All-Pair Shortest Path: Floyd's algorithm
- Graphs with negative weights
  - Bellman-Ford algorithm

Weighted graphs

- In real world graphs, each edge may have weight
  - On road networks, each edge (a road segment) have a weight, which may be the distance between two road junctions or the travel time from one junction to another
  - In navigation systems, e.g., Google Map, we may want to find the path with minimum travel time between two locations



Shortest path problems

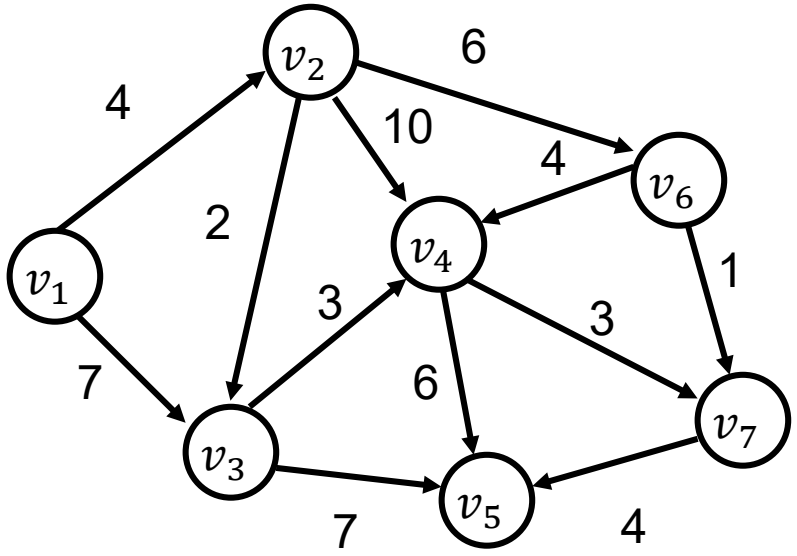
- How can we find the shortest route between two points on a road map?

- Model the problem as a graph problem:
  - Road map is a weighted graph:
    vertices = cities
    edges = road segments between cities
    edge weights = road distances

  - Goal: find a shortest path between two vertices (cities)

An example weighted graph

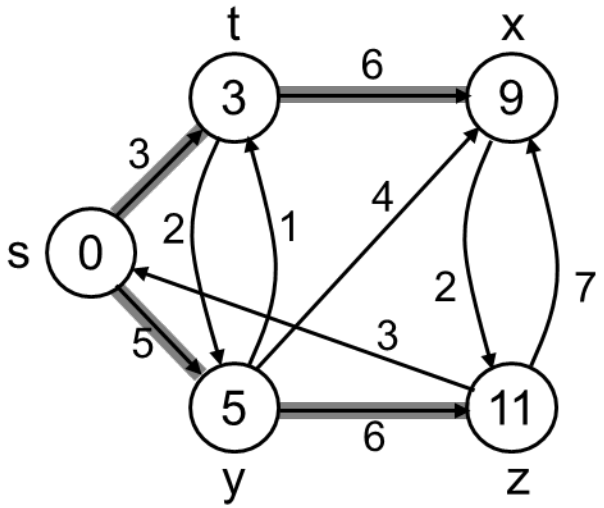
- A weighted graph is a graph such that each edge e is associated with a weight $w(e)$
  - We focus on directed graphs
  - The solution can be easily extended to undirected graphs



$w(v_1, v_2) = 4$
$w(v_1, v_3) = 7$
…

Shortest path problems

- **Input:**
  - Directed graph G = (V, E)
  - Weight function w : E → **R**



- **Weight of path** p = $\langle v_0, v_1, \ldots, v_k \rangle$

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

- **Shortest-path weight** from u to v:

$$\delta(u, v) = \min \begin{cases} w(p) : u \overset{p}{\rightsquigarrow} v & \text{if there exists a path from u to v} \\ \infty & \text{otherwise} \end{cases}$$

- **Note:** there might be multiple shortest paths from u to v

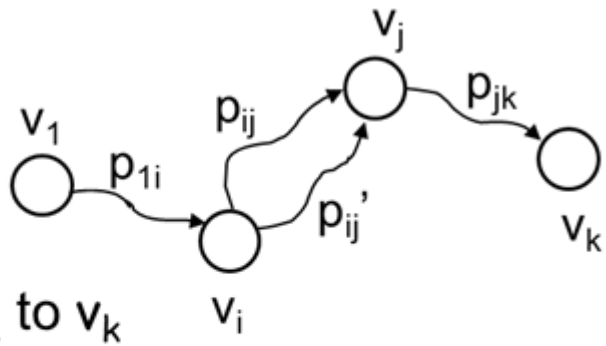
Variants of shortest path

- **Single-source shortest paths**
  - $G = (V, E) \Rightarrow$ find a shortest path from a given source vertex **s** to each vertex $v \in V$
- **Single-destination shortest paths**
  - Find a shortest path to a given destination vertex **t** from each vertex v
  - Reversing the direction of each edge $\Rightarrow$ single-source
- **Single-pair shortest path**
  - Find a shortest path from u to v for given vertices u and v
- **All-pairs shortest-paths**
  - Find a shortest path from u to v for every pair of vertices u and v

Optimal substructure theorem

Given:

- A weighted, directed graph G = (V, E)
- A weight function w: E → **R**,
- A shortest path $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$



Then: p_{ij} is a shortest path from v_i to v_j

Proof: $p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$$

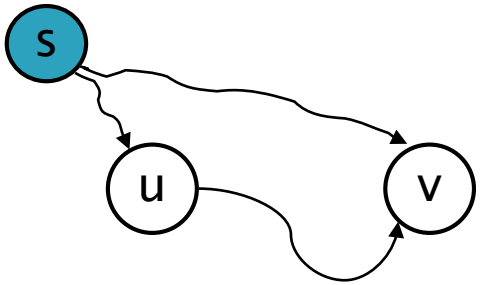
Assume $\exists\ p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$ contradiction!

Triangle inequality

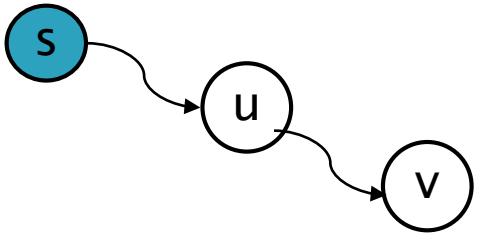
- For all $(u, v) \in E$, we have:

  $$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$$



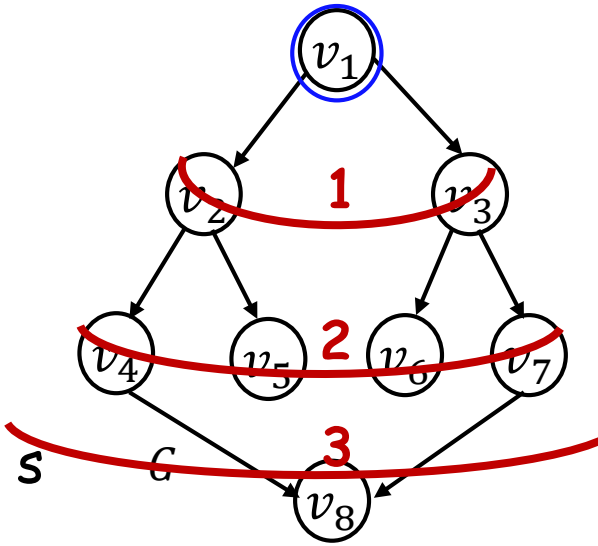
Proof?

- If u is on the shortest path to v we have the equality sign



Shortest path algorithm

- A simple case: unweighted graph
  - How to find the shortest path? Use BFS!



- A simple algorithm

  1. Mark the starting vertex, s
  2. Find and mark all unmarked vertices adjacent to s
  3. Find and mark all unmarked vertices adjacent to the marked vertices
  4. Repeat Step 3 until all vertices are marked

- For each vertex, keep track of
  - whether the adjacent vertex has been marked
  - its distance from $s (d_v)$
  - previous vertex of the path from $s (p_v)$

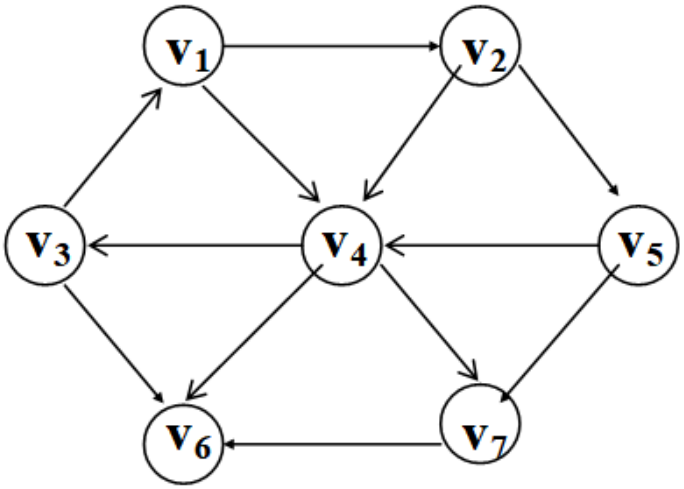
Shortest path algorithm

/* Pseudocode for better unweighted shortest-path algorithm with O(|E| + |V|) running time*/

```
void unweighted(Vertex s) {
    Queue<Vertex> q = new Queue<Vertex>();
    for each vertex { v.dist = INFINITY;}
    s.dist = 0;
    q.enqueue(s);
    while(!q.isEmpty()){
        Vertex v = q.dequeue();
        for each Vertex w adjacent to v
            if(w.dist == INFINITY){
                w.dist = v.dist + 1;
                w.path = v;
                q.enqueue(w);
            }
        }
    }
}
```

Running time is O(|E| + |V|)

Shortest path algorithm



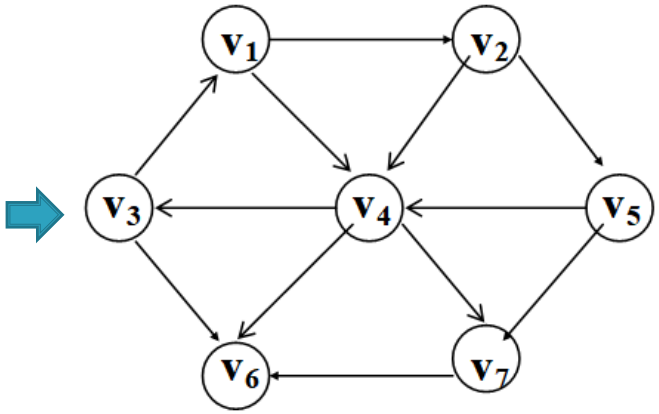
Start vertex

v	*Known*	d_v	p_v
v_1	F	∞	0
v_2	F	∞	0
v_3	F	0	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Shortest path algorithm

	Initial State		
v	*Known*	d_v	p_v
v_1	F	∞	0
v_2	F	∞	0
v_3	F	0	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0
Q	v_3		

	v_3 Dequeued		
v	*Known*	d_v	p_v
v_1	F	1	v_3
v_2	F	∞	0
v_3	1	0	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	1	v_3
v_7	F	∞	0
Q	v_1, v_6		



	v_1 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	F	2	v_1
v_3	T	0	0
v_4	F	2	v_1
v_5	F	∞	0
v_6	F	1	v_3
v_7	F	∞	0
Q	v_6, v_2, v_4		

	v_6 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	F	2	v_1
v_3	T	0	0
v_4	F	2	v_1
v_5	F	∞	0
v_6	T	1	v_3
v_7	F	∞	0
Q	v_2, v_4		

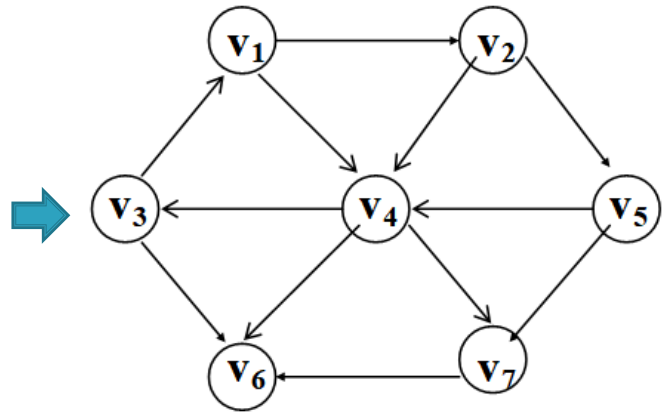
Shortest path algorithm

	v_2 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	F	2	v_1
v_5	F	3	v_2
v_6	T	1	v_3
v_7	F	∞	0
Q	v_4, v_5		

	v_4 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	F	3	v_2
v_6	T	1	v_3
v_7	F	3	v_4
Q	v_5, v_7		

	v_5 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	T	3	v_2
v_6	T	1	v_3
v_7	F	3	v_4
Q	v_7		

	v_7 Dequeued		
v	*Known*	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	T	3	v_2
v_6	T	1	v_3
v_7	T	3	v_4
Q	empty		

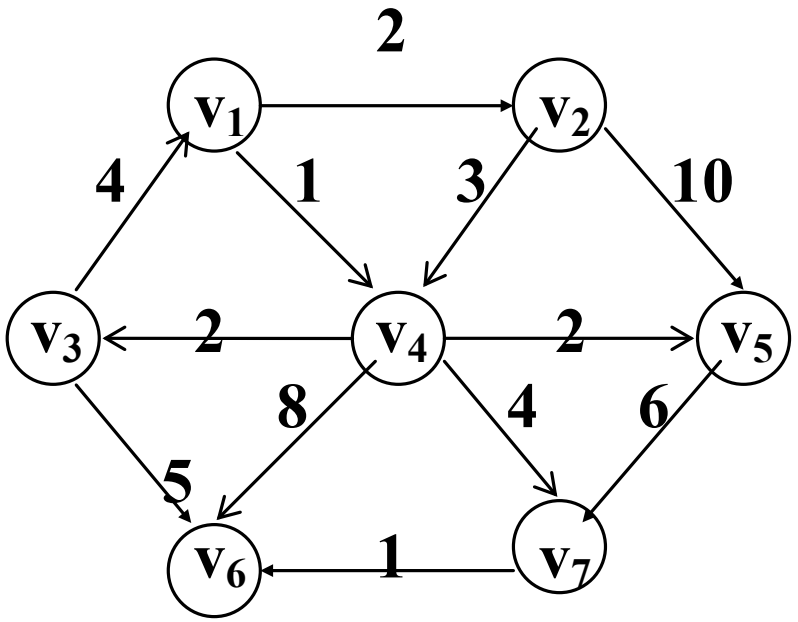


Shortest path algorithm

- Dijkstra's Algorithm for weighted graphs
  - A greedy algorithm, solving a problem by stages by doing what appears to be the best thing at each stage
  - Select a vertex v, which has the smallest d_v among all the unknown vertices, and declare that the shortest path from s to v is known
  - For each adjacent vertex, w, update $d_w = d_v + c_{v,w}$ if this new value for d_w is an improvement

Example of Dijkstra's algorithm

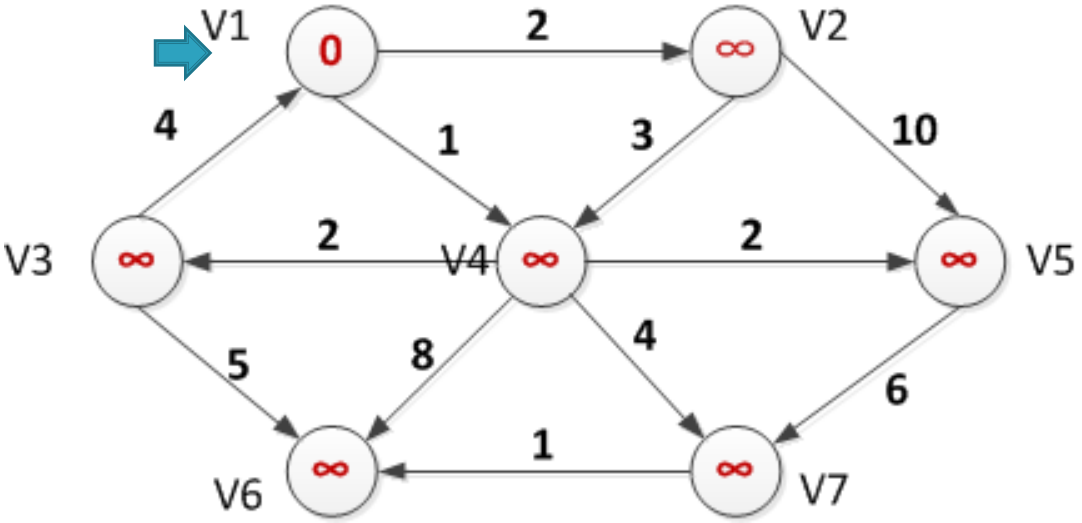
Given a graph, find the shortest path starting from v_1:



Intermediate results of Dijkstra's algo.

v	*Known*	d_v	p_v
v_1	0	0	0
v_2	0	∞	0
v_3	0	∞	0
v_4	0	∞	0
v_5	0	∞	0
v_6	0	∞	0
v_7	0	∞	0

Initial configuration

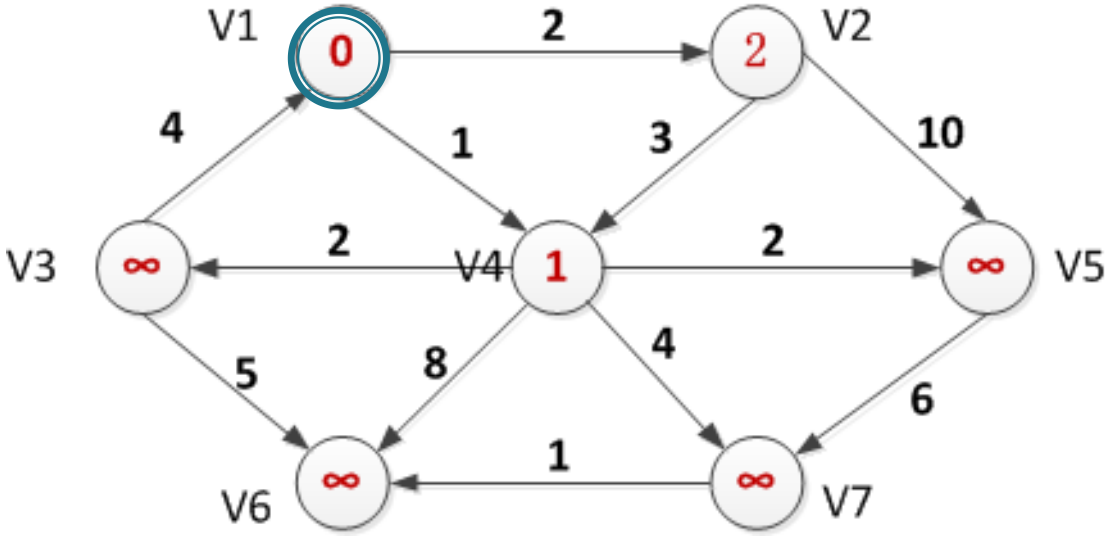


where v_1 is the start vertex

Intermediate results of Dijkstra's algo.

After v_1 is declared known

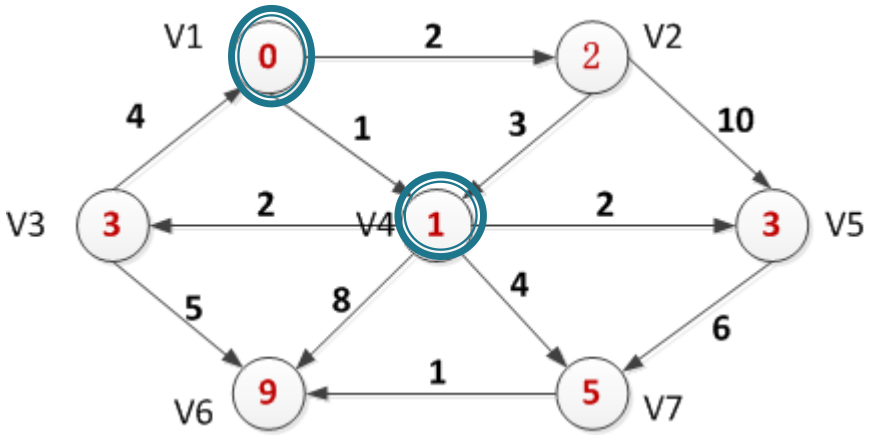
v	*Known*	d_v	p_v
v_1	1	0	0
v_2	0	2	v_1
v_3	0	∞	0
v_4	0	1	v_1
v_5	0	∞	0
v_6	0	∞	0
v_7	0	∞	0



Intermediate results of Dijkstra's algo.

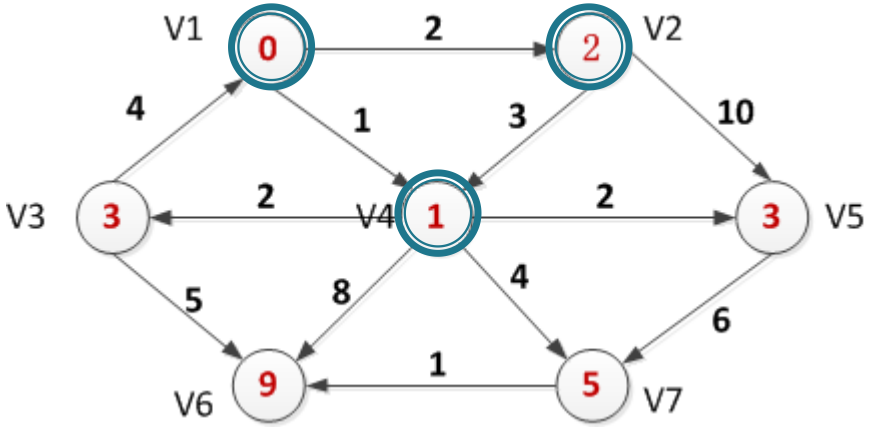
After v_4 is declared known

v	*Known*	d_v	p_v
v_1	1	0	0
v_2	0	2	v_1
v_3	0	3	v_4
v_4	1	1	v_1
v_5	0	3	v_4
v_6	0	9	v_4
v_7	0	5	v_4



After v_2 is declared known

v	*Known*	d_v	p_v
v_1	1	0	0
v_2	1	2	v_1
v_3	0	3	v_4
v_4	1	1	v_1
v_5	0	3	v_4
v_6	0	9	v_4
v_7	0	5	v_4



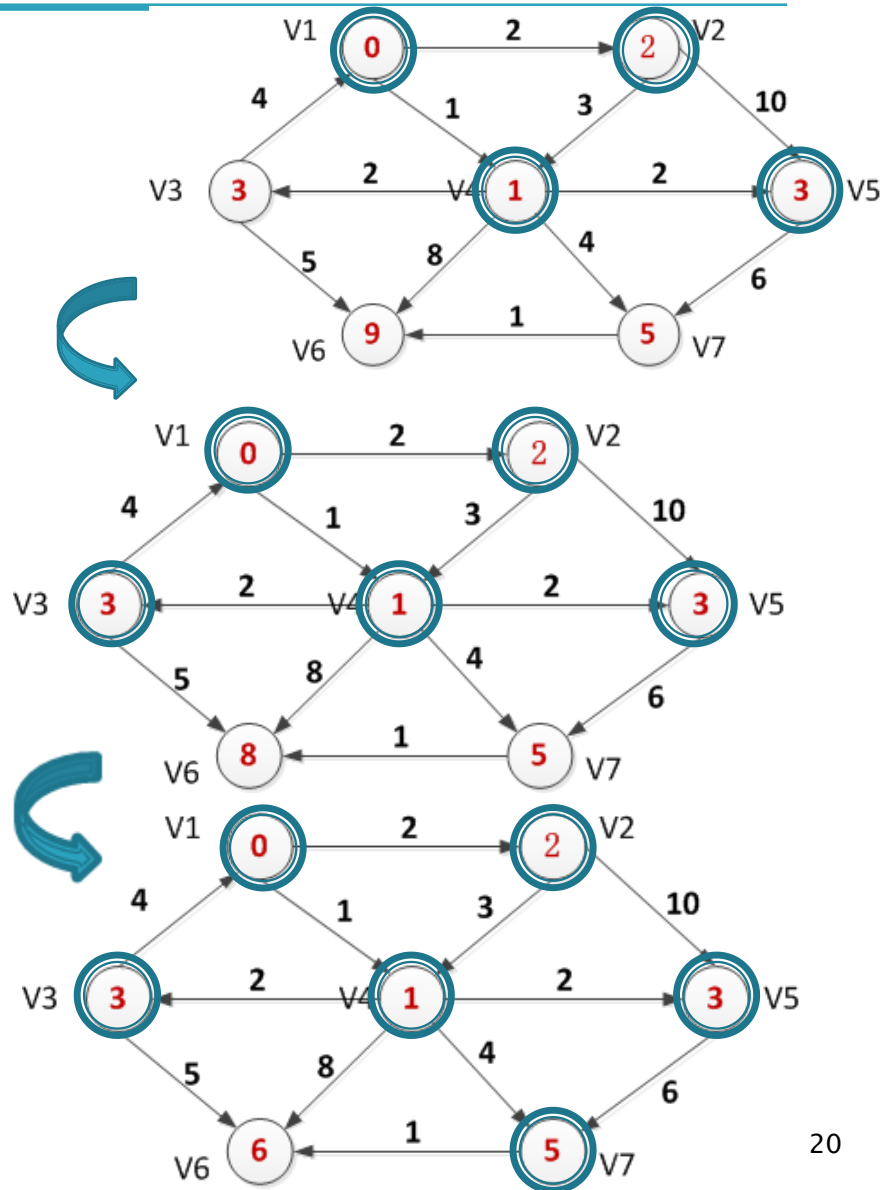
Intermediate results of Dijkstra's algo.

After v_5 and then v_3 are declared known

v	*Known*	d_v	p_v
v_1	1	0	0
v_2	1	2	v_1
v_3	1	3	v_4
v_4	1	1	v_1
v_5	1	3	v_4
v_6	0	8	v_3
v_7	0	5	v_4

After v_7 is declared known

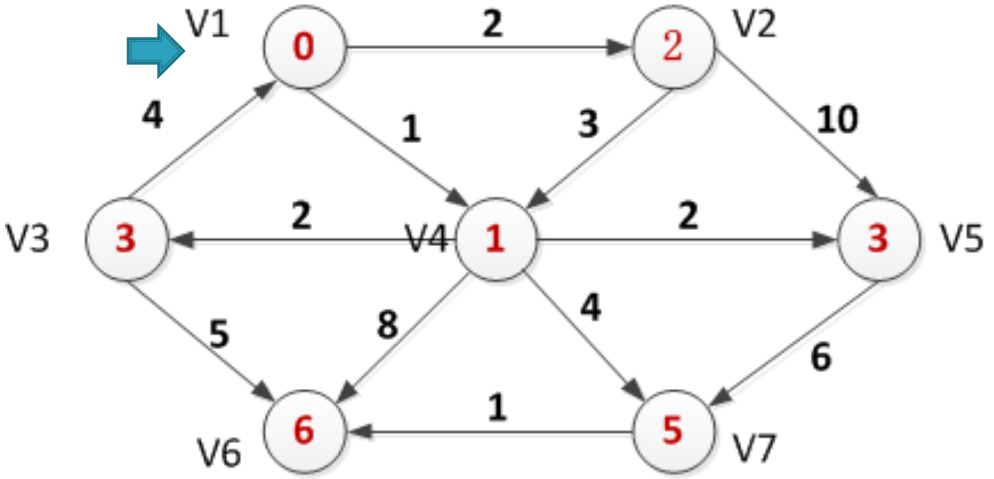
v	*Known*	d_v	p_v
v_1	1	0	0
v_2	1	2	v_1
v_3	1	3	v_4
v_4	1	1	v_1
v_5	1	3	v_4
v_6	0	6	v_7
v_7	1	5	v_4



Intermediate results of Dijkstra's algo.

After v_6 is declared known

v	*Known*	d_v	p_v
v_1	1	0	0
v_2	1	2	v_1
v_3	1	3	v_4
v_4	1	1	v_1
v_5	1	3	v_4
v_6	1	6	v_7
v_7	1	5	v_4



Initialization

Alg.: INITIALIZE-SINGLE-SOURCE(V, s)

1. **for** each $v \in V$
2. **do** $d[v] \leftarrow \infty$
3. $p[v] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

Relaxation step

- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

  If d[v] > d[u] + w(u, v)

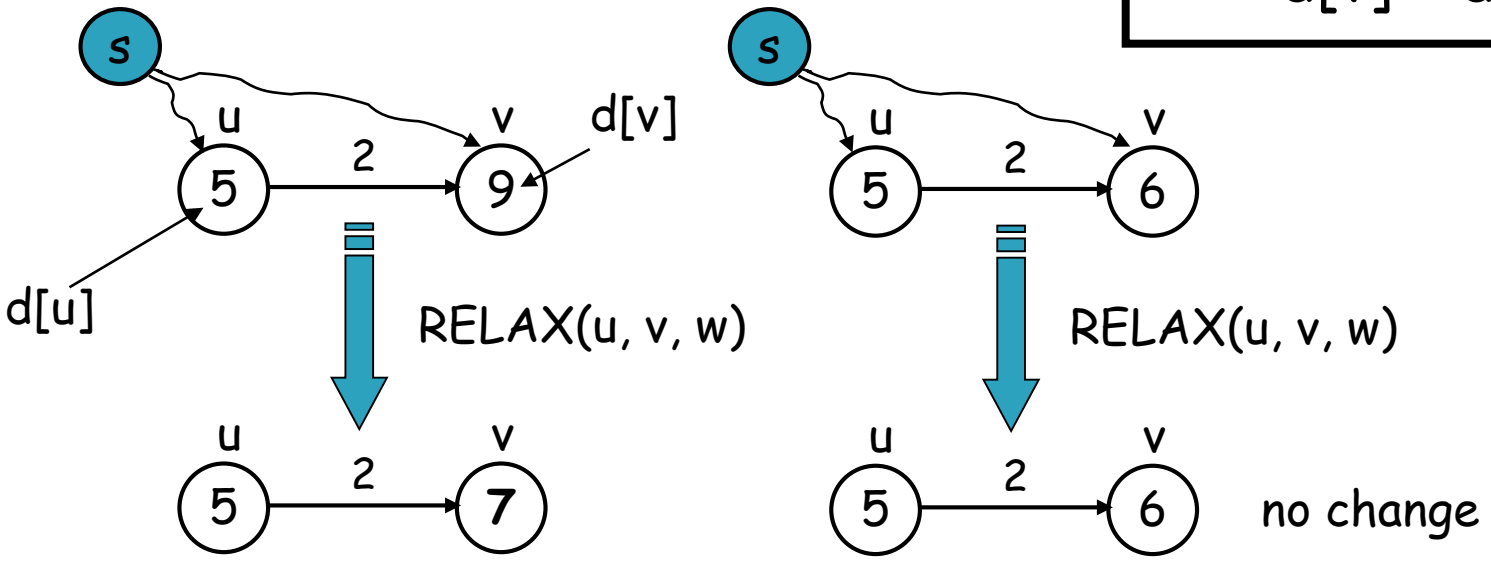
  we can improve the shortest path to v

  ⇒ d[v]=d[u]+w(u,v)

  ⇒ p[v] ← u

After relaxation:

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



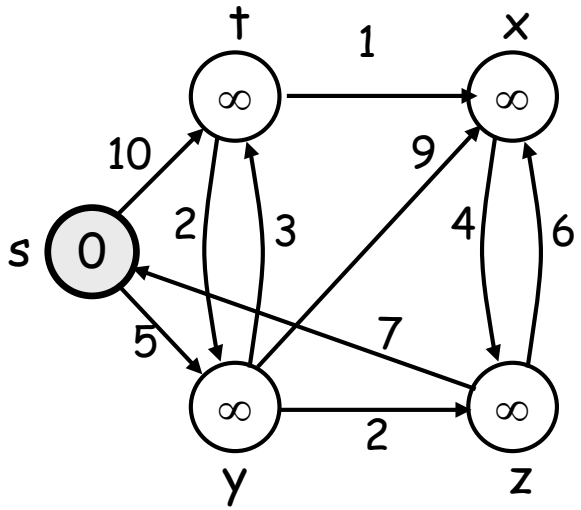
Dijkstra (G, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s) ⟵ $\Theta(V)$
2. $S \leftarrow \varnothing$
3. $Q \leftarrow V[G]$ ⟵ $O(V)$ build min-heap
4. **while** $Q \neq \varnothing$ ⟵ Executed $O(V)$ times
5. **do** $u \leftarrow$ EXTRACT-MIN(Q) ⟵ $O(\lg V)$ } $O(V \lg V)$
6. $S \leftarrow S \cup \{u\}$
7. **for** each vertex $v \in$ Adj[u] ⟵ $O(E)$ times (total)
8. **do** RELAX(u, v, w)
9. Update Q (DECREASE_KEY) ⟵ $O(\lg V)$ } $O(E \lg V)$

Running time: $O(V \lg V + E \lg V) = O(E \lg V)$

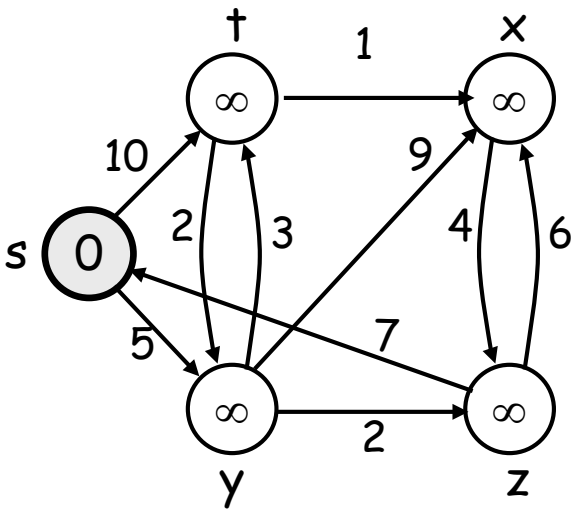
Exercise

- Show the steps of Dijkstra's algorithm

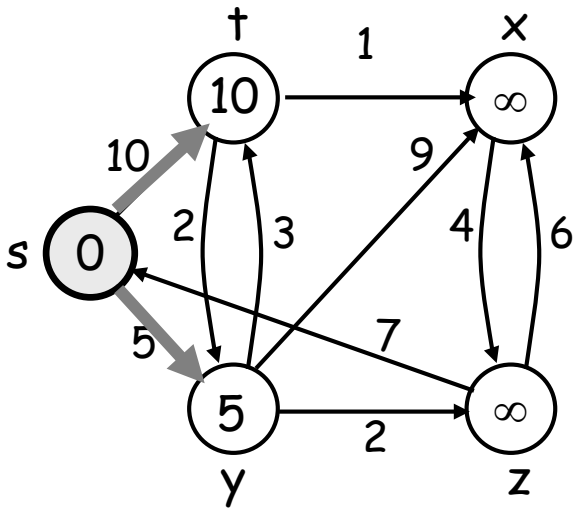


Dijkstra (G, w, s)

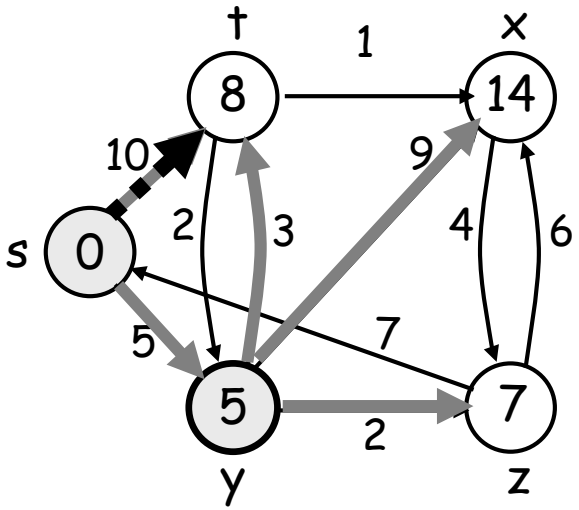
S=<> Q=<s,t,x,z,y>



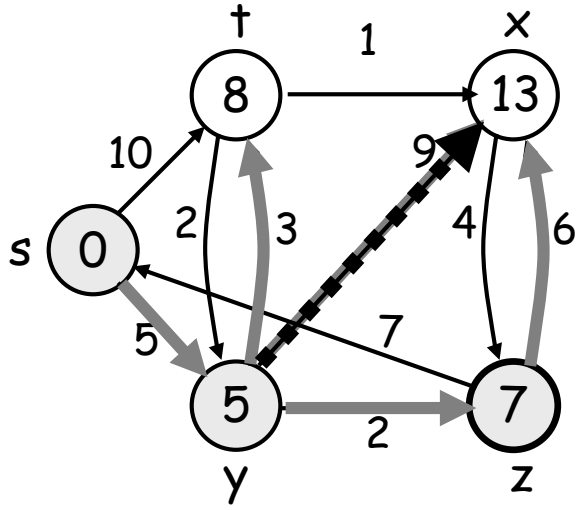
S=<s> Q=<y,t,x,z>



Example (cont.)



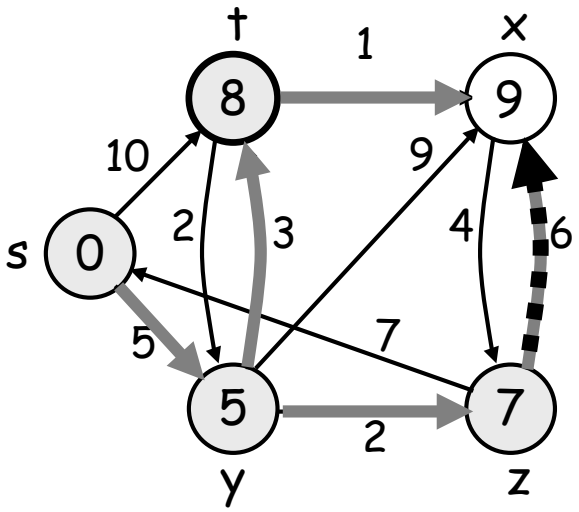
S=<s,y> Q=<z,t,x>



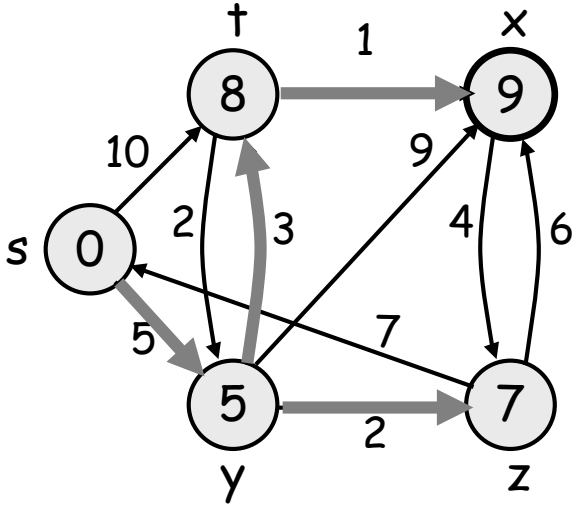
S=<s,y,z> Q=<t,x>

Example (cont.)

S=<s,y,z,t> Q=<x>



S=<s,y,z,t,x> Q=<>



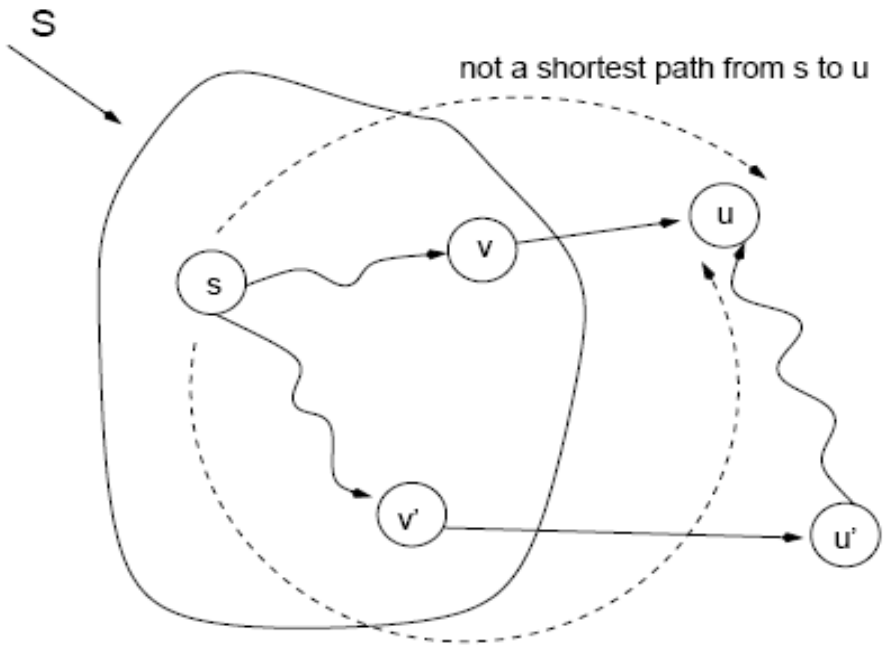
Correctness of Dijkstra's algorithm

- For each vertex $u \in V$, we must have $d[u] = \delta(s, u)$ at the time when u is added to S

Proof:

- Let u be the first vertex for which $d[u] \neq \delta(s, u)$ when added to S
- Let's look at a true shortest path p from s to u:

Correctness of Dijkstra's algorithm



0: Suppose the shortest to u is via u'

1. What is the value of d[u]?
$d[u] \le d[v]+w(v,u) = \delta(s,v)+w(v,u)$

2. What is the value of d[u']?
$d[u'] \le d[v']+w(v',u') = \delta(s,v')+w(v',u')$

3. Since u' is in the shortest path of u: $d[u'] < \delta(s,u)$

4. however, from definition: $d[u] >= \delta(s,u)$

$d[u'] < d[u]$

5. Contradiction!

Priority Queue Q: <u, …, u', ….> (i.e., d[u]<…<d[u']<…)

Exercise 1

- Given a directed graph $G=(V,E)$ where each edge (u, v) has an associated value $r(u,v)$, which is a real number in the range $0 \leq r(u,v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v
  - We interpret $r(u,v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent
  - Give an efficient algorithm to find the most reliable path between two given vertices

Exercise 1

- Solution 1: modify Dijkstra's algorithm
  - r(u,v) = Pr(channel from u to v will not fail)
  - Assuming that the probabilities are independent, the reliability of a path p = $\langle v_1, v_2, \ldots, v_k \rangle$ is:

    $r(v_1,v_2)\, r(v_2,v_3) \ldots r(v_{k-1},v_k)$
  - Find the channel with the highest reliability, i.e.,

$$\max_p \prod_{(u,v)\in p} r(u,v)$$

Exercise 1 (cont.)

- But Dijkstra's algorithm computes

$$\min_p \sum_{(u,v)\in p} w(u,v)$$

- Perform relaxation as follows:

  if d[v] < d[u] w(u,v) then
  d[v] = d[u] w(u,v)

- Use "EXTRACT_MAX" instead of "EXTRACT_MIN"

Exercise 1 (cont.)

- Solution 2: use Dijkstra's algorithm without any modifications!
  - Goal

$$\max_p \prod_{(u,v)\in p} r(u,v)$$

  - Take the lg

$$\lg(\max_p \prod_{(u,v)\in p} r(u,v)) = \max_p \sum_{(u,v)\in p} \lg(r(u,v))$$

Exercise 1 (cont.)

- Turn this into a minimization problem by taking the negative:

$$-\min_p \sum_{(u,v)\in p} \lg(r(u,v)) = \min_p \sum_{(u,v)\in p} -\lg(r(u,v))$$

- Run Dijkstra's algorithm using

$$w(u,v) = -\lg(r(u,v))$$

Recommended reading

- Reading materials
  - Textbook Chapters 24&25

- Next lecture
  - All pairs shortest path