

# Lecture 22

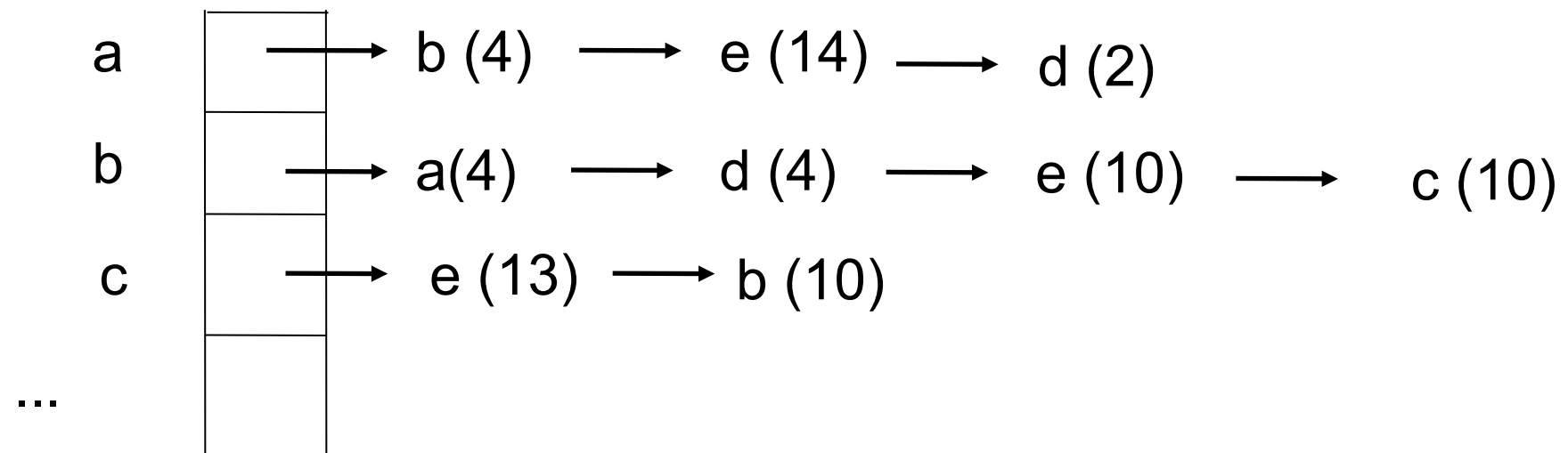
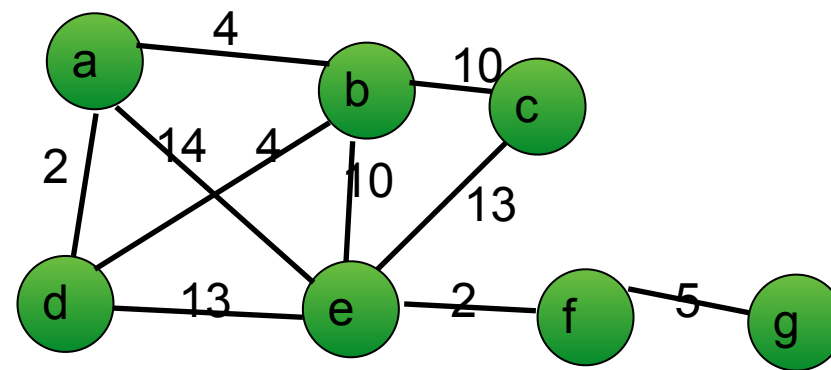
## Weighted Graphs and Dijkstra's Algorithm

If the graph was a computer network where the edge weights were the time it took for a packet to traverse the edge, how could we find the path (route) our message (packets) as fast as possible?

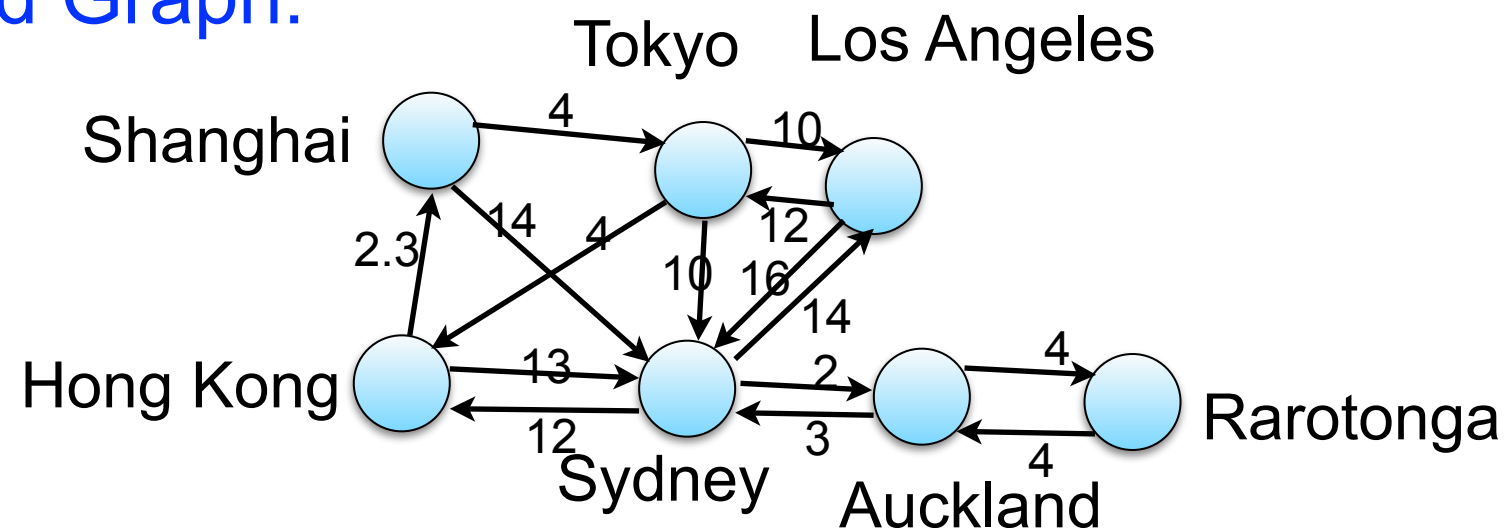
CS2134

# Weighted Graphs

- For many applications, it's useful to associate a “weight” (cost) with each edge:
  - Transportation: price of ticket between two cities, or physical distance, or travel time, or ...
  - Communication: congestion on link between two routers
  - ...
- Adjacency list
  - Store weight along with target node for each edge
- Adjacency matrix
  - Store weight, rather than true/false for each pair of vertices with some default value indicating “no edge”



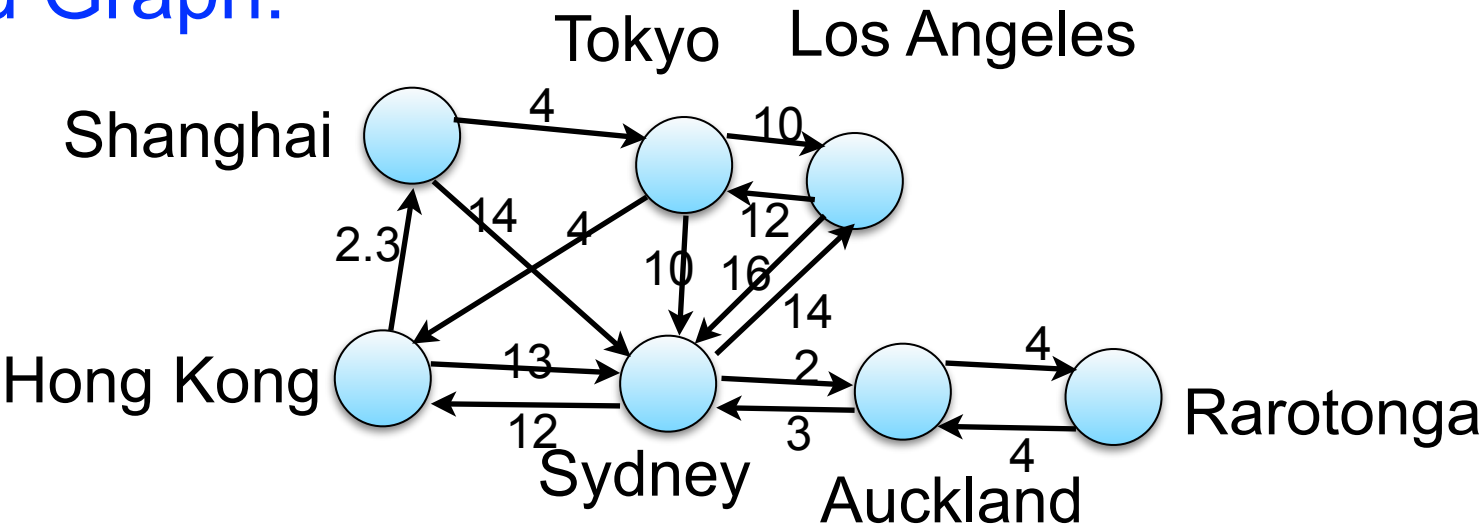
## Directed Graph:



## Adjacency List:

Shanghai	→	Tokyo (4)	→	Sydney (14)		
Tokyo	→	Hong Kong (4)	→	Sydney (10)	→	Los Angeles (10)
Hong Kong	→	Sydney (13)	→	Shanghai (2.3)		
...						

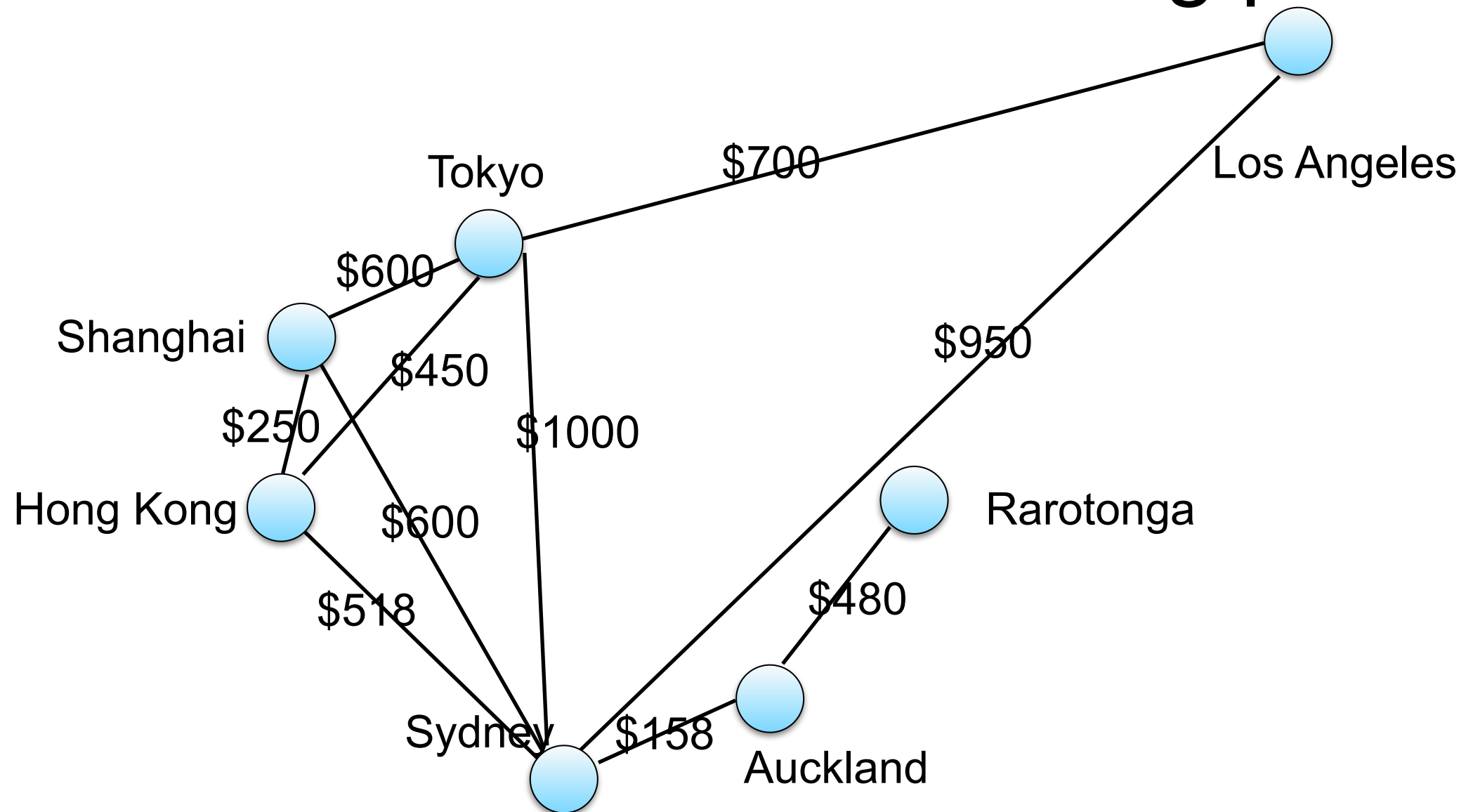
Directed Graph:



Adjacency Matrix:

	Shanghai	Hong Kong	Sydney	Tokyo	Auckland	Los Angeles	Rarotonga
Shanghai			14	4			
Hong Kong	2.3		13				
Sydney		12			2	14	
Tokyo		4	10			10	
Auckland			3				4
Los Angeles			16	12			
Rarotonga					4		

# What is the cost of the following path?

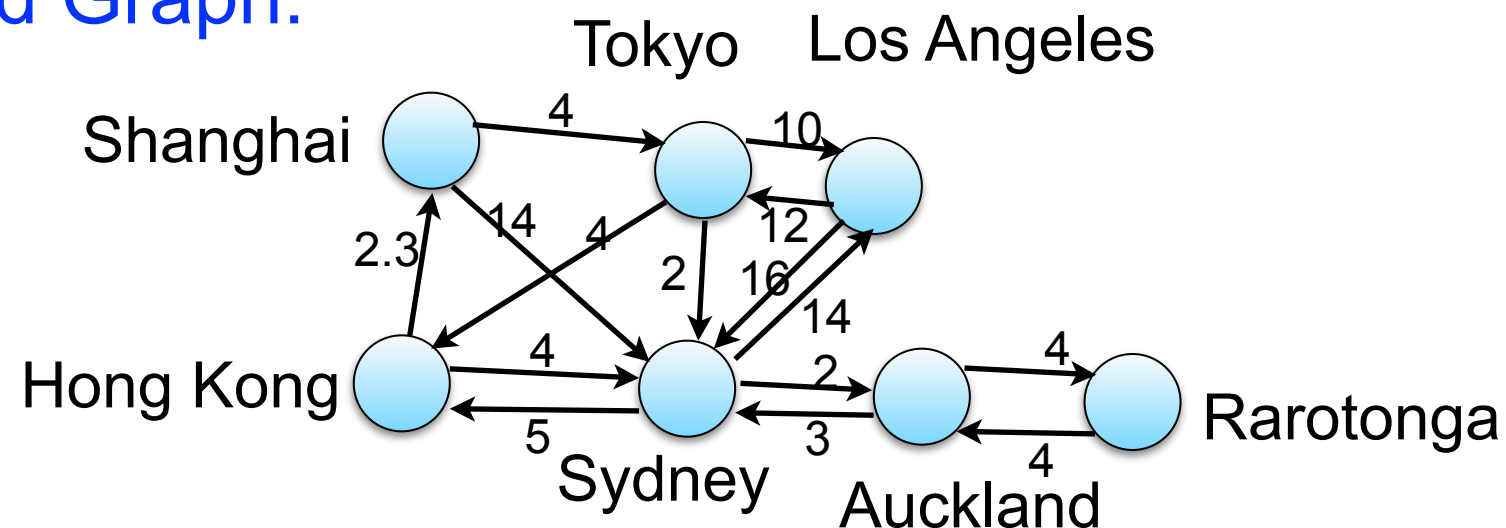


A path: Los Angeles, Tokyo, Shanghai, Sydney, Auckland **Not the best path!**  
This weighted path has length 2058.

# Shortest Path Problem in Weighted Graphs

- The weight/cost of a path is the sum of the weights/costs of its edges
- Shortest path between vertices  $v$  and  $w$  is the path connecting  $v$  and  $w$  with minimal weight/cost
- Note: shortest path from  $v$  to  $w$  is not necessarily the path with fewest edges.

## Directed Graph:



The shortest **unweighted** path from Shanghai to Auckland has length 2.

The path is Shanghai, Sydney, Auckland

This path has cost 16.

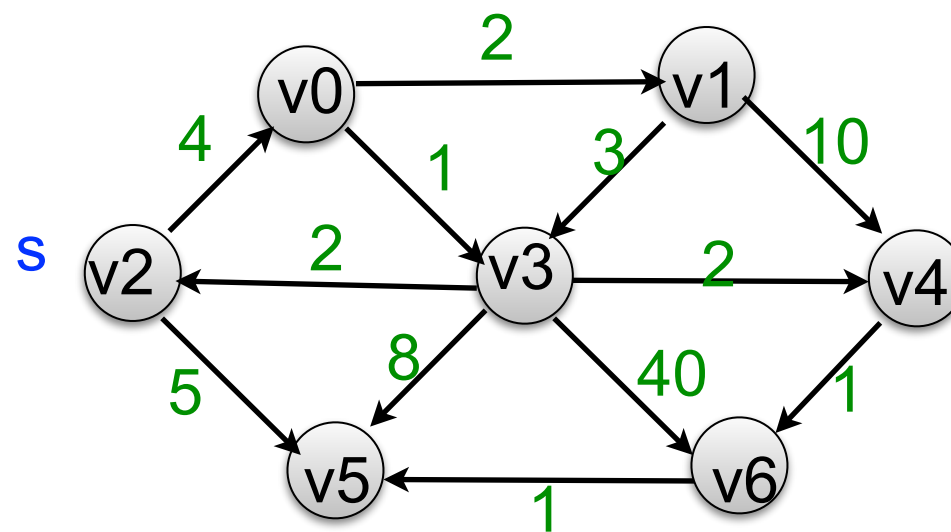
The shortest **weighted** path from Shanghai to Auckland has length 3.

The path is Shanghai, Tokyo, Sydney, Auckland

This path has cost 8.



# Weighted Shortest-path



What is the shortest unweighted path from v2 to v6?

What is the shortest weighted path from v2 to v6?

# Dijkstra's Algorithm

*Finds shortest weighted path*

- requires all edge weights to be positive
- finds the shortest path from a source node  $s$  to all the other nodes in  $G$
- $G$  can be directed or undirected (also true for unweighted shortest graph algorithm)

# Dijkstra's Algorithm

- Modification of BFS:
  - Three categories of nodes:
    - visited (finished),
    - discovered (but not visited),
    - undiscovered;
  - initially **s** is discovered and has distance 0. And all other nodes are undiscovered and have infinite distance.
  - At each stage, choose the discovered/unvisited node, **v**, that has smallest distance (total cost) from **s**:
    - For each edge adjacent to **v** (that is not finished)
      - ▶ Discover neighbor OR
      - ▶ Reduce cost of neighbor already discovered, if a shorter path has been found
    - set **v** to having been visited (finished)

# Pseudocode for Dijkstra's Algorithm

Input: Graph G, source vertex s

## Initialization

For every vertex,  $v$ , set  $\text{dist}[v]$  to  $\infty$

For every vertex,  $v$ , set  $\text{prev}[v]$  to sentinel

Mark all nodes as **undiscovered**

Mark  $s$  as **discovered**

$\text{dist}[s] = 0$

## main loop

While there exists a **discovered** (and not **visited**) node {

Find a **discovered** vertex  $v$  with  $\min \text{dist}[v]$

For each neighbor  $w$  of  $v$  that has not been **visited** {

If  $(\text{dist}(v) + \text{weight}(v,w) < \text{dist}(w))$ {

update  $\text{dist}(w)$  to be  $\text{dist}(v) + \text{weight}(v,w)$

update  $\text{prev}[w]$  to be  $v$

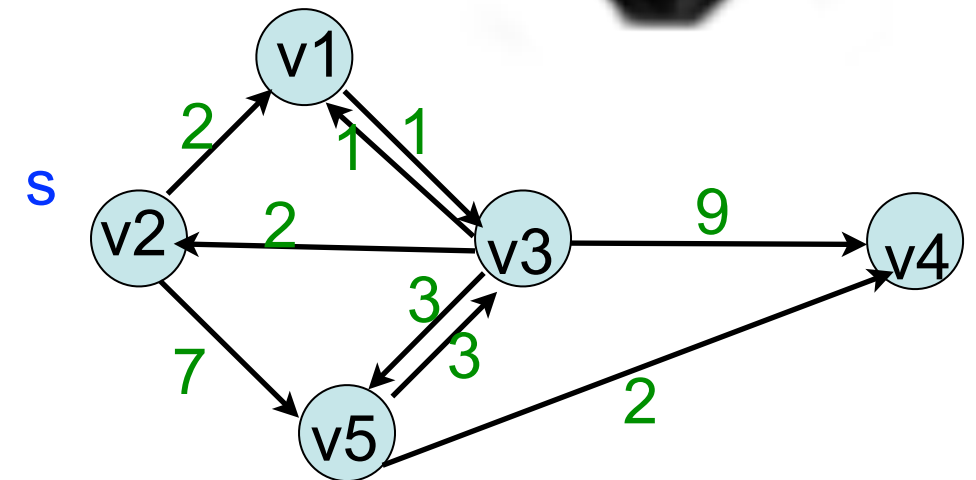
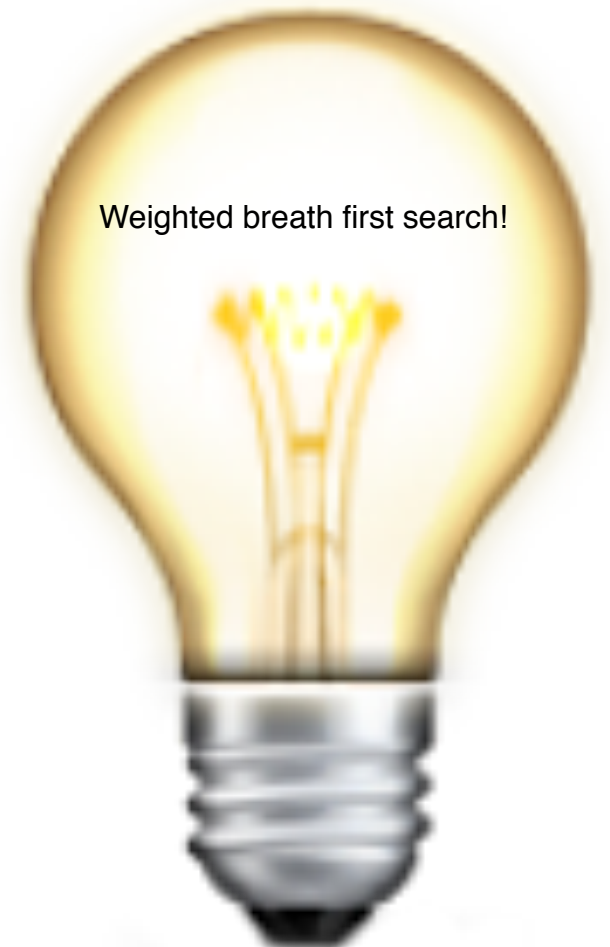
mark  $w$  as **discovered**

}

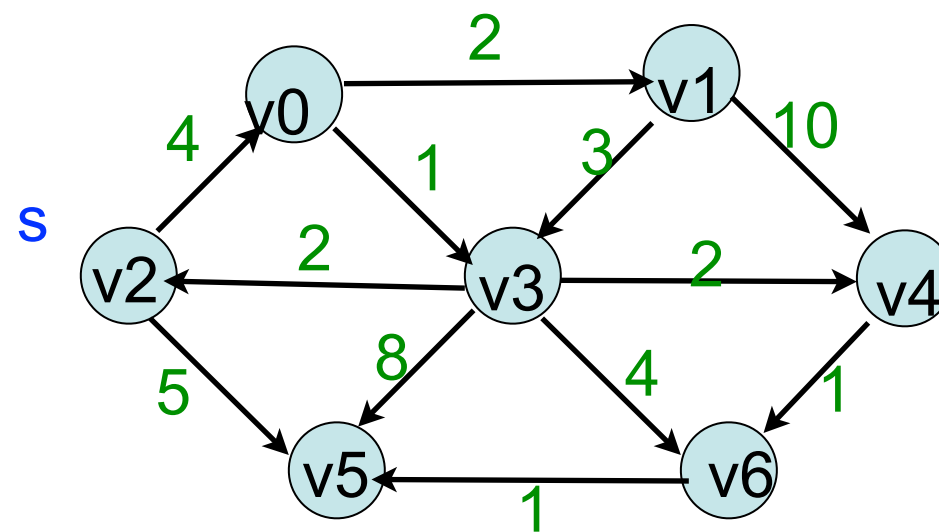
mark  $v$  as **visited** ( $v$  is no longer marked as **discovered**)

}

}

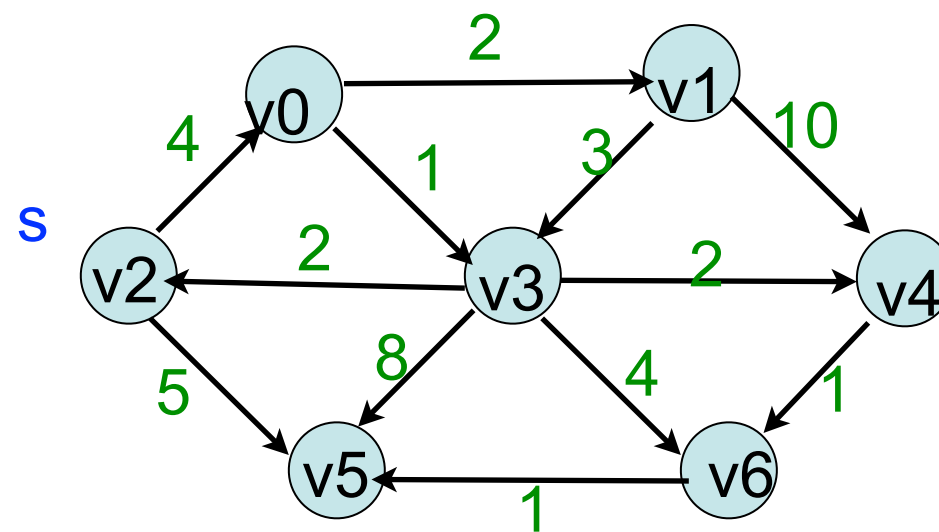


# Weighted Shortest-path Initialization



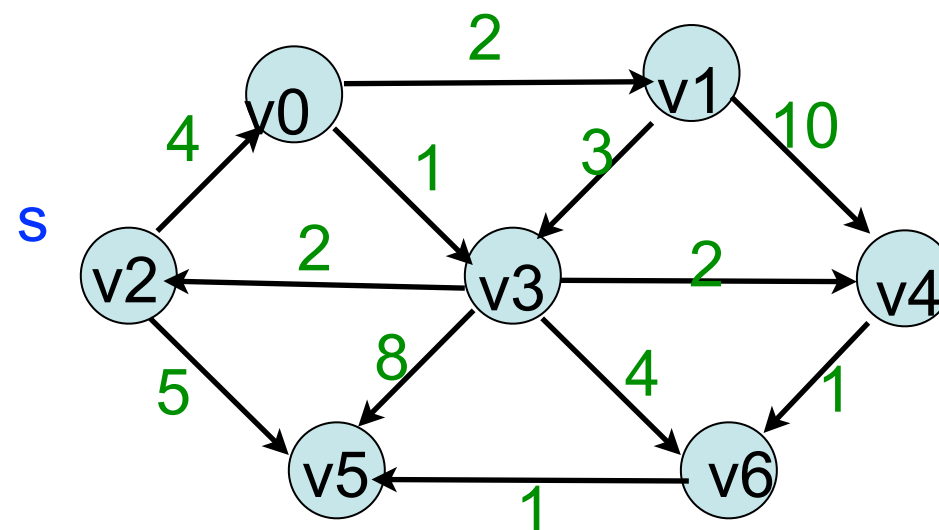
phase	distance/predecessor							visiting	discovered&not visited
	v0	v1	v2	v3	v4	v5	v6		
init	$\infty$	$\infty$	0/-	$\infty$	$\infty$	$\infty$	$\infty$		v2

# Weighted Shortest-path



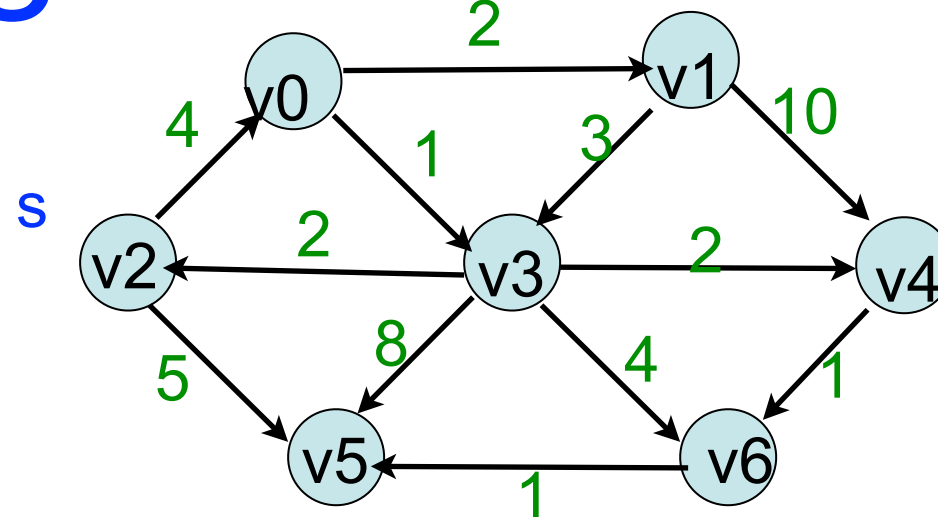
phase	distance/predecessor							visiting	discovered&not visited
	v0	v1	v2	v3	v4	v5	v6		
init	$\infty$	$\infty$	0/-	$\infty$	$\infty$	$\infty$	$\infty$		v2
1	4/v2	$\infty$	0/-	$\infty$	$\infty$	5/v2	$\infty$	v2	v0 , v5

# Weighted Shortest-path



phase	distance/predecessor								discovered&not visited
init	$\infty$	$\infty$	0/-	$\infty$	$\infty$	$\infty$	$\infty$		v2
1	4/v2	$\infty$	0/-	$\infty$	$\infty$	5/v2	$\infty$	v2	v0, v5
2	4/v2	6/v0	0/-	5/v0	$\infty$	5/v2	$\infty$	v0	v5 v1 v3

# Weighted Shortest-path



phase	distance/predecessor							visiting	discovered&not visited
	v0	v1	v2	v3	v4	v5	v6		
init	$\infty$	$\infty$	0/-	$\infty$	$\infty$	$\infty$	$\infty$		v2
1	4/v2	$\infty$	0/-	$\infty$	$\infty$	5/v2	$\infty$	v2	v0, v5
2	4/v2	6/v0	0/-	5/v0	$\infty$	5/v2	$\infty$	v0	v5 v1 v3
3									



# Why does it work?

Invariants:

- Whenever a node is **finished**, the shortest estimated distance cannot be improved. A shortest path from  $s$  to this node has been found.
- At any stage the distances to **discovered** nodes represent the shortest distance via path consisting only of **finished** nodes.
  - Can prove this by induction.

At the final stage, all nodes are **finished**

=> algorithm is correct

# Details/Variations

Data structure for finding min weight  
discovered nodes?

# Version 0

- Just use the distance vector
  - Mark nodes when **finished**
  - At each stage, find **discovered** node  $v$  with minimum  $\text{dist}(v)$  and process its neighbors
- Running time  $O(|V|^2)$ 
  - Find **discovered** node with smallest distance  $|V|$  times,  $O|V|$  each time =  $O(|V|^2)$
  - Total for neighbor processing is  $O(|E|)$
  - Running time is  $O(|V|^2 + |E|) = O(|V|^2)$
- Can we do better for sparse graphs?

# Variation 1

- Use data structure that supports insert, deleteMin, and decreaseKey efficiently
- When discover a node, insert into data structure
- Need to decrease distances of neighbors
- Our binary heap doesn't support decreaseKey 😞
- Fancy data structures: Fibonacci heap (Omit details)

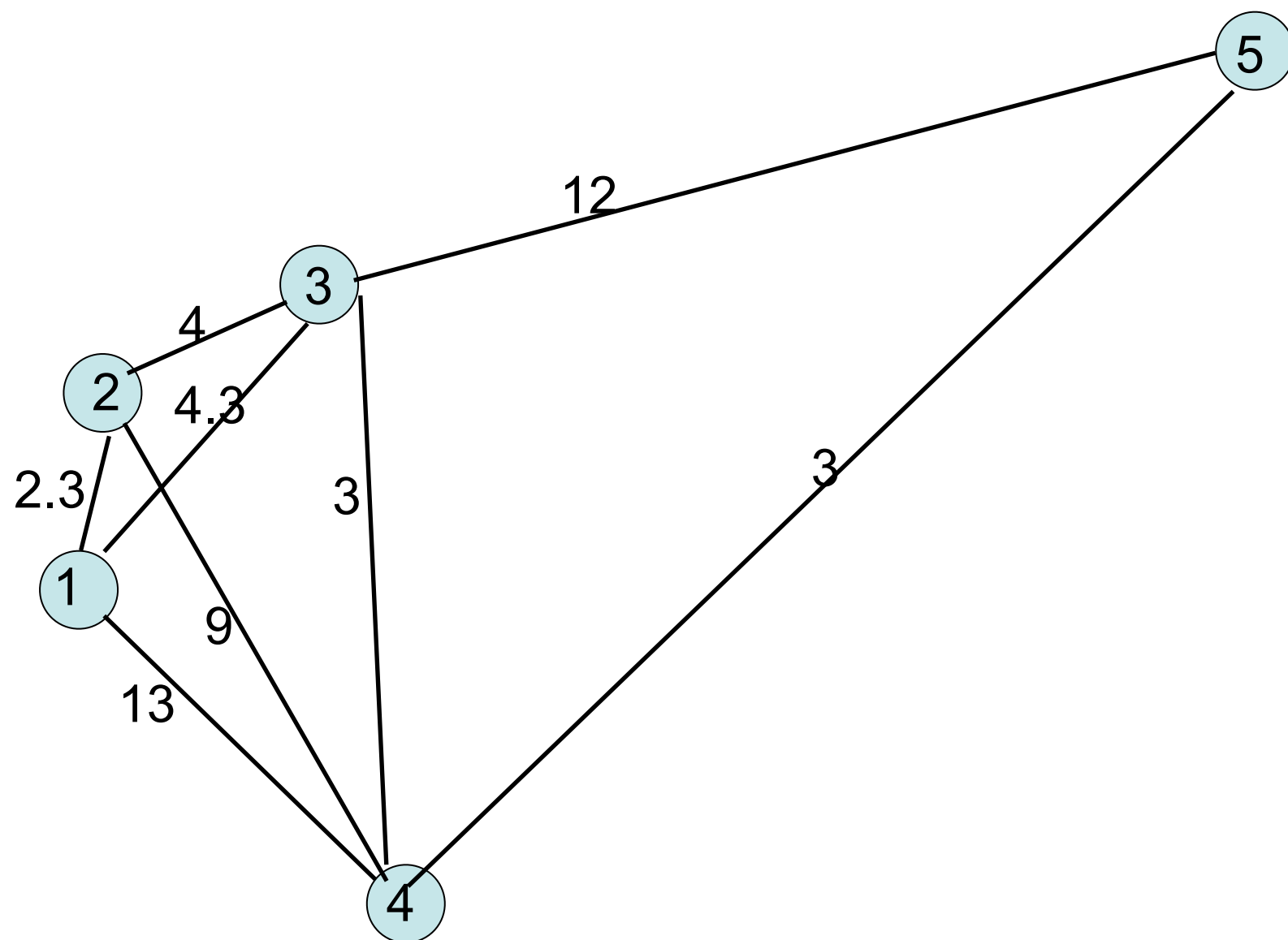
# Variation 2

- Store (node, distance) pairs in ordinary binary heap, with distances as keys
- Insert node (with distance) when **discovered**
- Instead of decreasing key of  $w$  from  $d_1$  to  $d_2$ , leave old pair  $(w, d_1)$  in heap and insert new pair  $(w, d_2)$
- When do deleteMin, check if node already **finished**. If so, discard and try again.
  - Note that if  $(w, d_1)$  and  $(w, d_2)$  are both in the heap, with  $d_2 < d_1$ , item  $(w, d_2)$  will be returned earlier, so  $(w, d_1)$  will eventually be discarded.

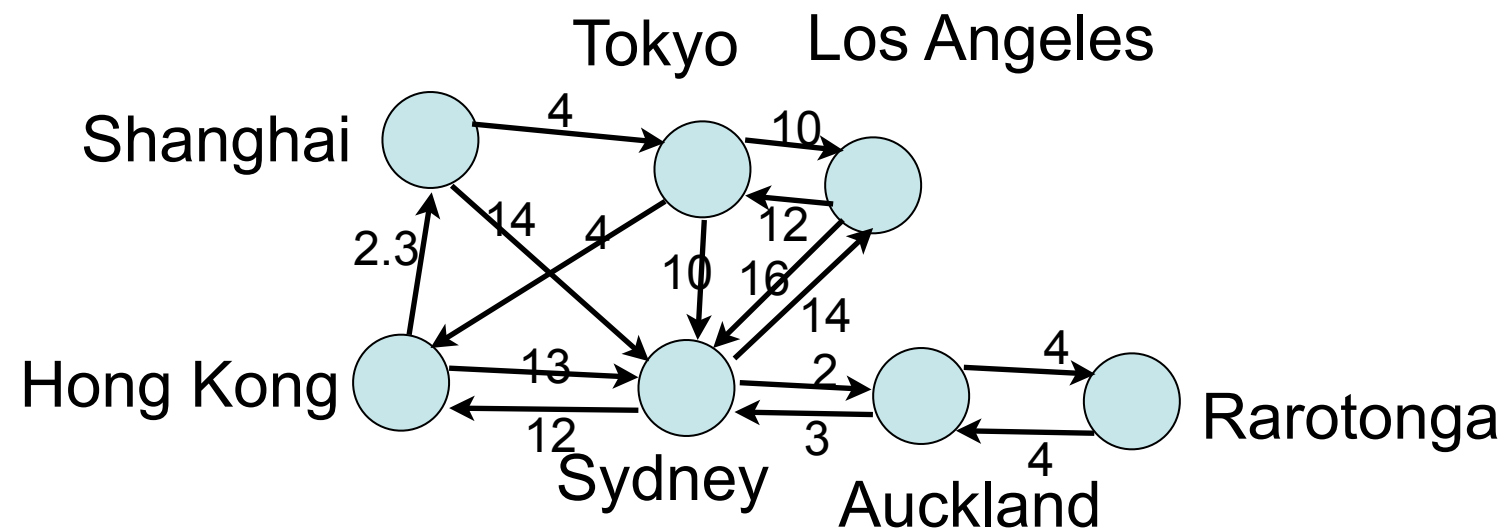
# Running time of Variation 2

$$O(|E| \log |V|)$$

- Binary heap size is at most  $|E|$ , so insert and deleteMin take  $O(\log |E|)$ .
- At most  $|E|$  inserts, and  $|E|$  deleteMins
- Initialization is  $O(|V|)$
- Assuming all nodes reachable from  $s$ , so  $|E| \geq |V|-1$ , total is  $O(|E| \log |E|)$   
 $= O(|E| \log |V|)$  since  $|E|$  is  $O(|V|^2)$



## Weighted Shortest Path From Shanghai?





## Loop Invariant (aka what is true at the top of the loop):

At the start of every iteration of the while loop,  $\text{dist}(v) = \text{optimal dist}(s, v)$  for all  $v$  which are marked as discovered.

## Maintenance (proving the loop invariant is true at the top of the loop):

Need to show that  $\text{dist}(u) = \text{optimal dist}(s, u)$  when  $u$  is marked as discovered in the loop

*Suppose not!* Suppose  $u$  is marked discovered, but  $\text{dist}(u) \neq \text{optimal dist}(s, u)$

WLOG let  $u$  be the first vertex marked as discovered that does not have the optimal path

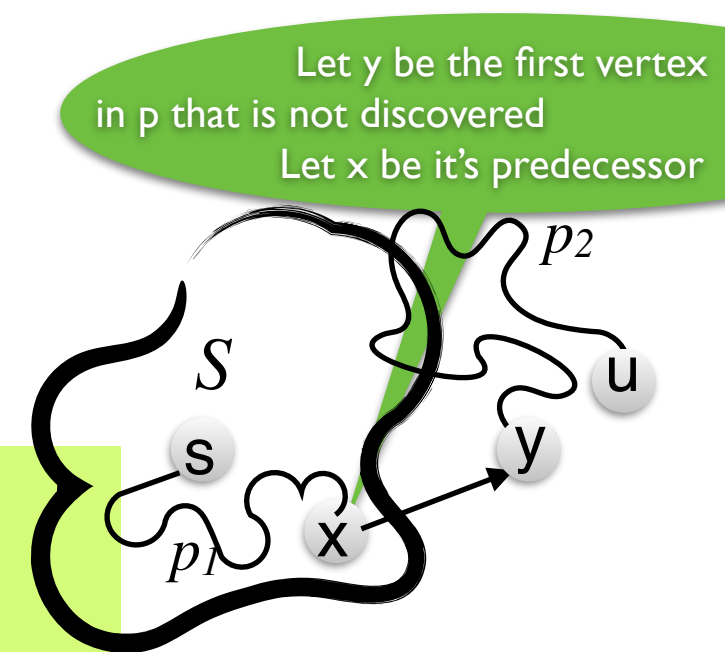
$u \neq s$  Let  $p$  be a shortest path  $s \xrightarrow{p} u$

Decompose  $p$  into subpaths:  $s \xrightarrow{p_1} x \rightarrow y \xrightarrow{p_2} u$

**Claim:**  $\text{dist}(y) = \text{optimal dist}(s, y)$  when  $u$  was marked discovered

**Proof:**  $\text{dist}(x) = \text{optimal dist}(s, x)$  when  $u$  was marked discovered

$\text{dist}(x, y)$  was updated when  $x$  was discovered  $\blacktriangleright \blacktriangleleft$



$y$  is on shortest path  $s \xrightarrow{p} u$ , and all edges are non-negative  $\Rightarrow \text{optimal dist}(s, y) \leq \text{optimal dist}(s, u)$

$\Rightarrow \text{dist}(y) = \text{optimal dist}(s, y) \leq \text{optimal dist}(s, u) \leq \text{dist}(u)$

But,  $u$  and  $y$  were in  $Q$ , and we chose  $u$  so,

$\text{dist}(u) \leq \text{dist}(y) \Rightarrow \text{dist}(u) = \text{dist}(y) \Rightarrow \text{optimal dist}(s, y) = \text{optimal dist}(s, u) \blacktriangleright \blacktriangleleft$