# Agenda for Module 2

Part 1:

- Introduction to Algorithms
- Data Structures

Part 2:

- Search Algorithms
- Time Complexity

Part 3:

- Sorting Algorithms

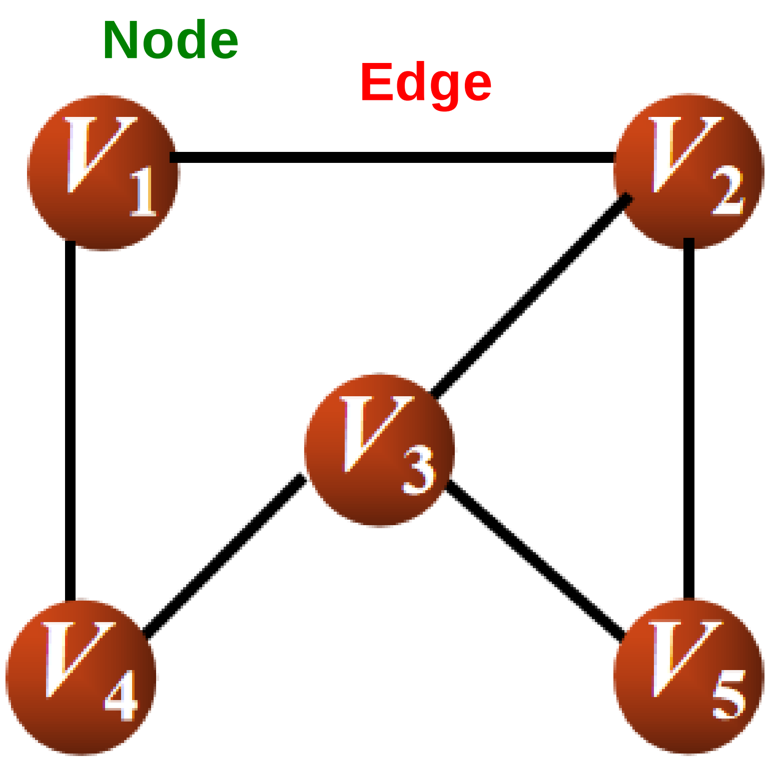
**Part 4:**

- **Graph Algorithms**
- **Algorithmic Techniques**

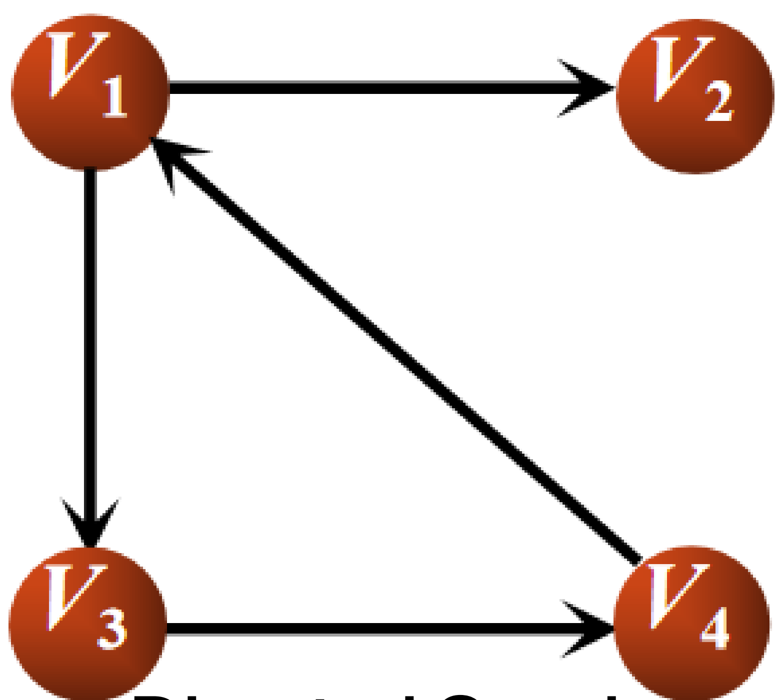
Ack: Some slides materials are based on *Introduction to Algorithms* by Cormen et al. and *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne
Resources: https://algs4.cs.princeton.edu/home/

# Graph Basics

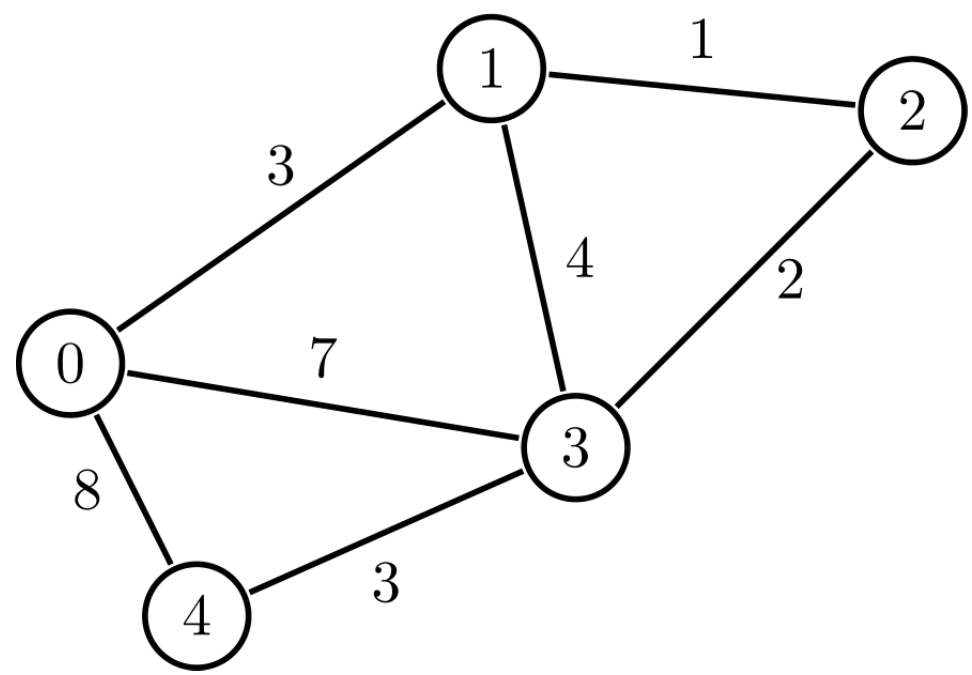
- Set of :
  - **Nodes/ Vertices** contains values and links to its neighbors
  - **edges** connecting the nodes
- Edges can be
  - Undirected: no distinction between the 2 nodes
  - Directed: has a source and a destination
  - Weighted




Undirected Graph




Directed Graph




Weighted Graph

# Applications - Navigation

- Google Maps - determining the fastest route

# Graph applications

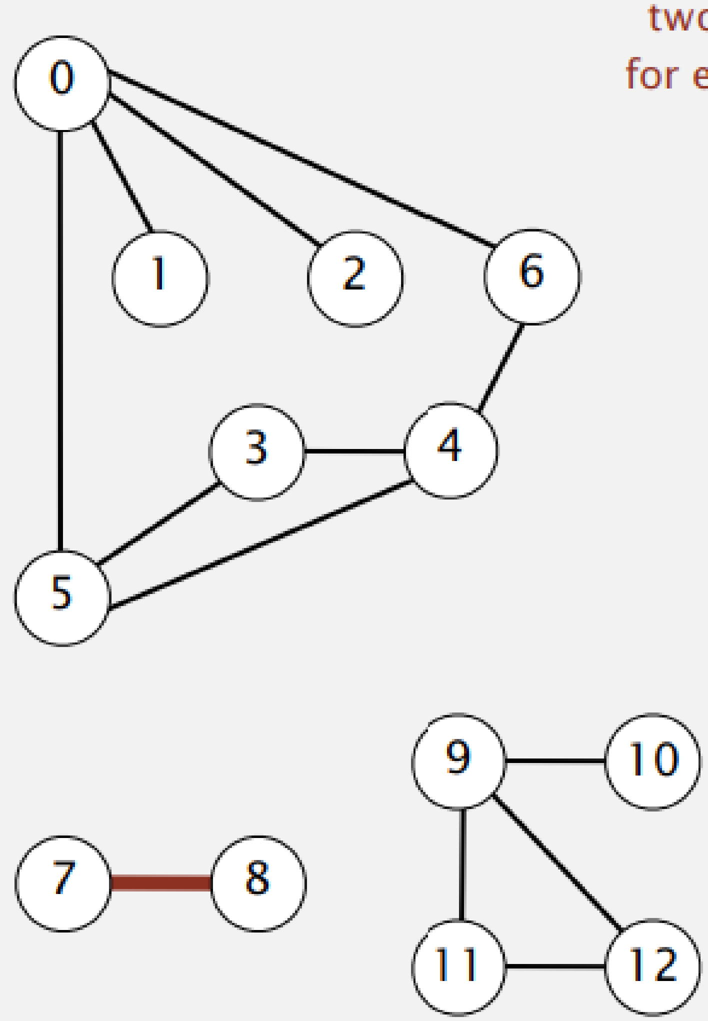
| graph | vertex | edge |
|---|---|---|
| **communication** | telephone, computer | fiber optic cable |
| **circuit** | gate, register, processor | wire |
| **mechanical** | joint | rod, beam, spring |
| **financial** | stock, currency | transactions |
| **transportation** | intersection | street |
| **internet** | class C network | connection |
| **game** | board position | legal move |
| **social relationship** | person | friendship |
| **neural network** | neuron | synapse |
| **protein network** | protein | protein-protein interaction |
| **molecule** | atom | bond |

# Computer Representation of Graph

- Adjacent Table/Matrix: Pros/ Cons?

Maintain a two-dimensional $V$-by-$V$ boolean array;

for each edge $v$–$w$ in graph: `adj[v][w] = adj[w][v] = true.`

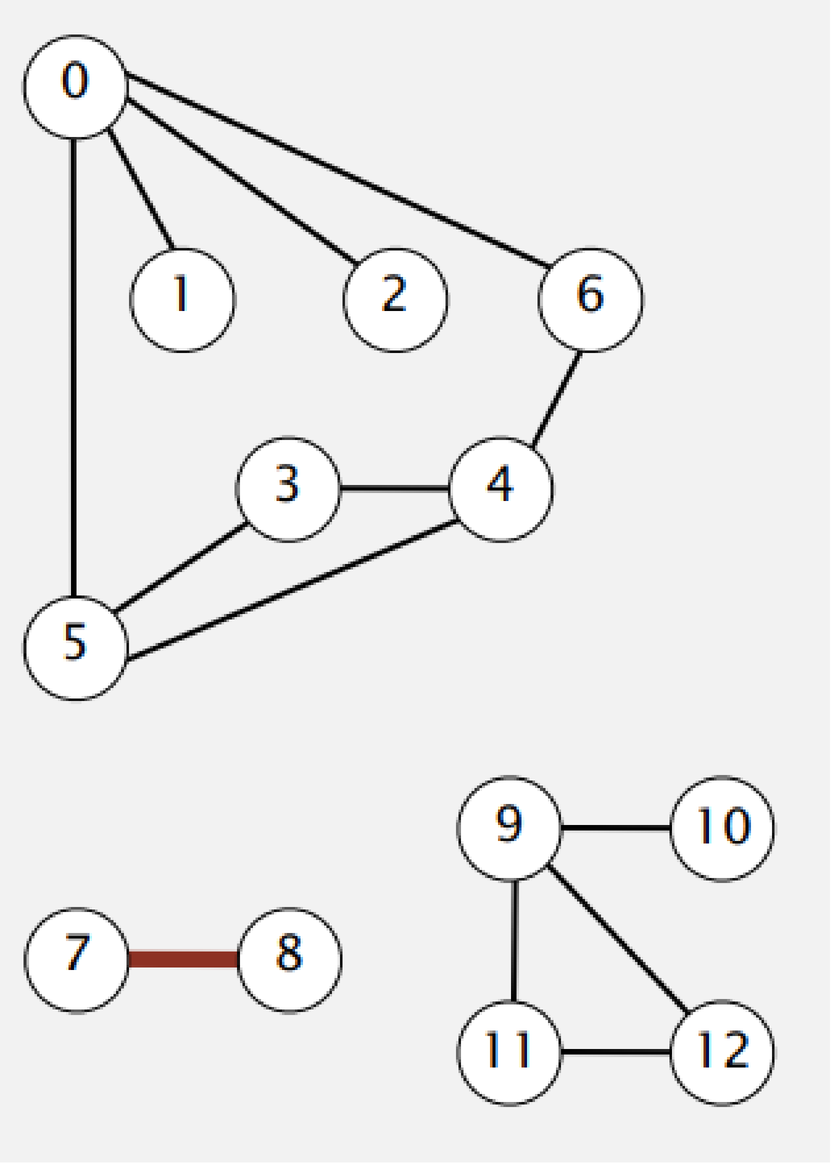


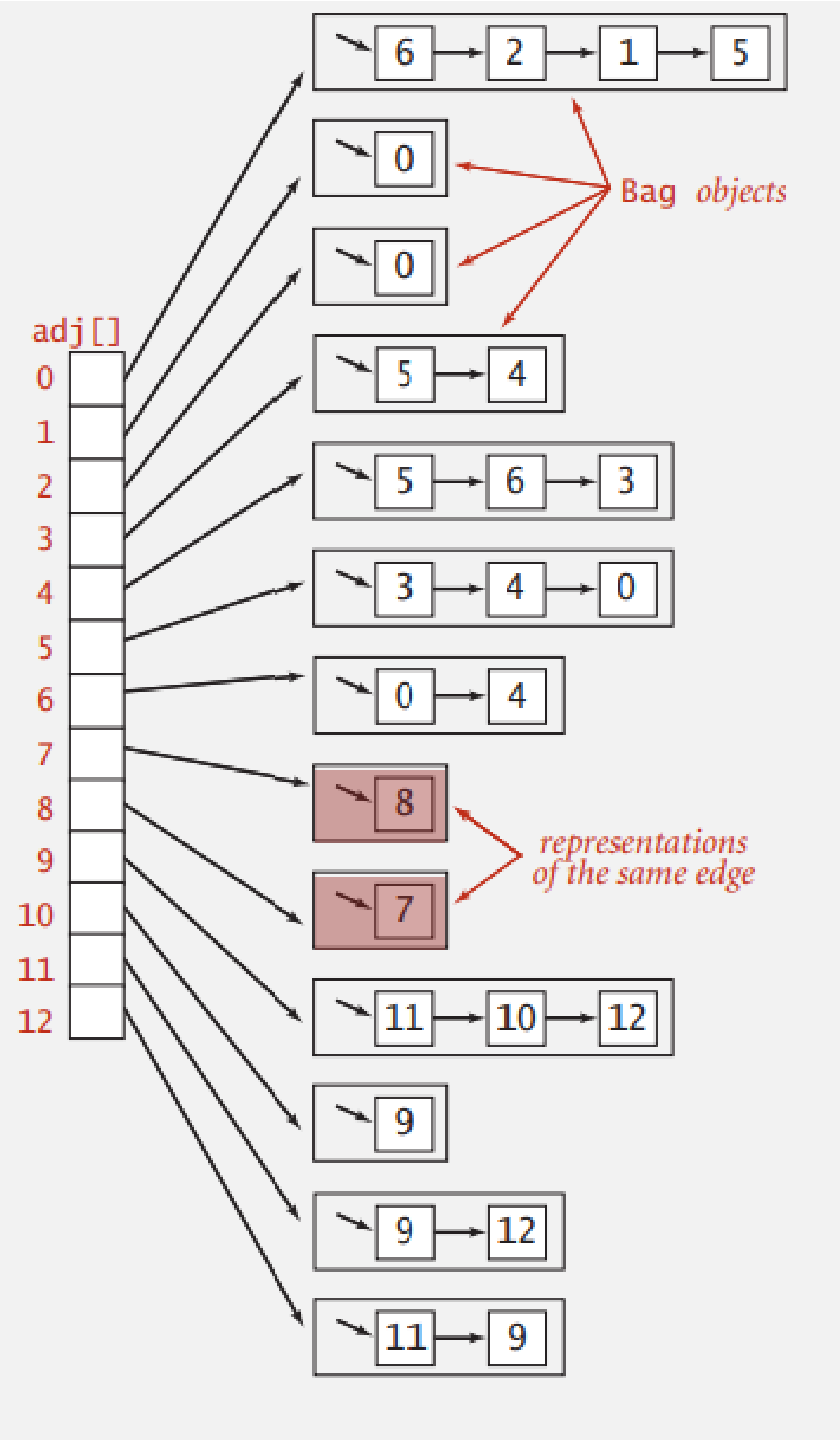

two entries
for each edge

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

# Computer Representation of Graph

- Adjacent List: Pros/ Cons?



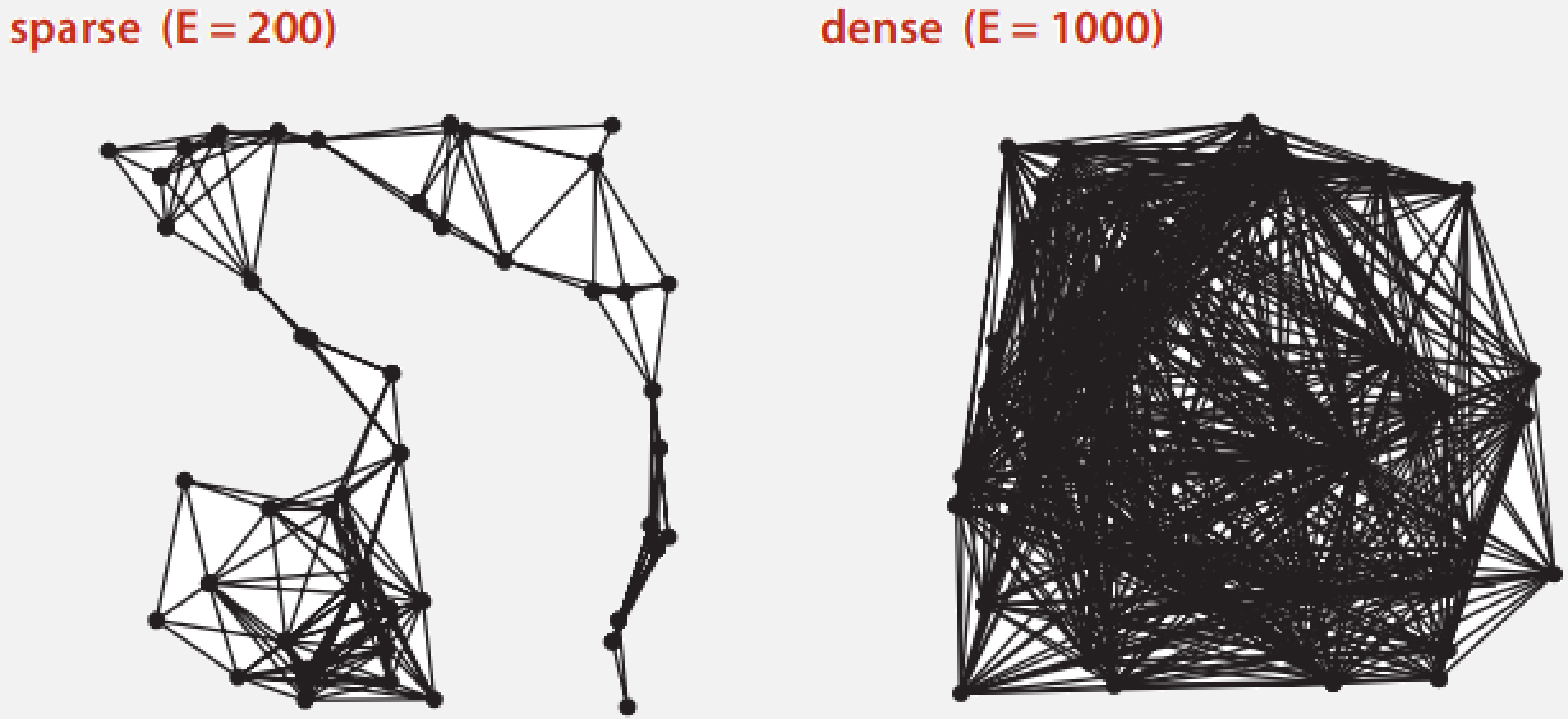



# Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to $v$.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree




Two graphs (V = 50)

# Depth-first search

Goal. Systematically traverse a graph.

Idea. Mimic maze exploration. ← function-call stack acts as ball of string

**DFS (to visit a vertex v)**

**Mark v as visited.**

**Recursively visit all unmarked vertices w adjacent to v.**

Typical applications.

- Find all vertices connected to a given source vertex.
- Find a path between two vertices.

Design challenge. How to implement?


http://algs4.cs.princeton.edu

# 4.1 DEPTH-FIRST SEARCH DEMO


http://algs4.cs.princeton.edu

# 4.1 BREADTH-FIRST SEARCH DEMO

# BFS & DFS Exercises

- Depth First Search exercise:
  - Find the depth of a binary tree (a tree is also a graph!)
  - https://leetcode.com/problems/maximum-depth-of-binary-tree/
- Breadth First Search exercise:
  - Find cousins in a binary tree.
  - Cousins have the same depth (level of the tree), but different parents.
  - https://leetcode.com/problems/cousins-in-binary-tree/

# Other Classical Problems in Graph：Example

- Topology Sort
  DFS + Reverse Post Order
- Minimum Spanning Tree
  Greedy / Kruskal / Prim
- Shortest Path
  Dijkstra / Acyclic / Bellman-Ford

# Algorithmic Techniques

- Brute-force algorithms
  - Take the most direct or obvious solution approach
  - Enumerate all possible candidate solutions
- Divide-and-conquer algorithms
  - break problem down into sub-problems that similar to the original problem, but smaller in size
  - Typically leads to recursive algorithms
  - E.g., MERGESORT
- Greedy algorithms
  - Often applied to optimization problems involving a sequence of choices
  - At each step, the locally optimal choice is made
- Dynamic programming algorithms
  - Break problem down into sub-problems
  - Use the solutions to these sub-problems to solve larger sub-problems (reusing results)

# Greedy Algorithm: Change Making

Given coins of denominations 1,5,10,25 cents; find out a way to give a customer an amount with the fewest number of coins.

Example: 147 cents

Greedy Algorithm Exercise:
https://leetcode.com/problems/largest-perimeter-triangle/

# Dynamic programming : Fibonacci Sequence

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it:

the 2 is found by adding the two numbers before it (1+1),
the 3 is found by adding the two numbers before it (1+2),
the 5 is (2+3),
and so on!

You can try it yourself: https://leetcode.com/problems/fibonacci-number/

# Exercise / Q&A

# ADDITIONAL MATERIAL

https://leetcode.com/problems/linked-list-cycle/

# Tortoise and Rabbits

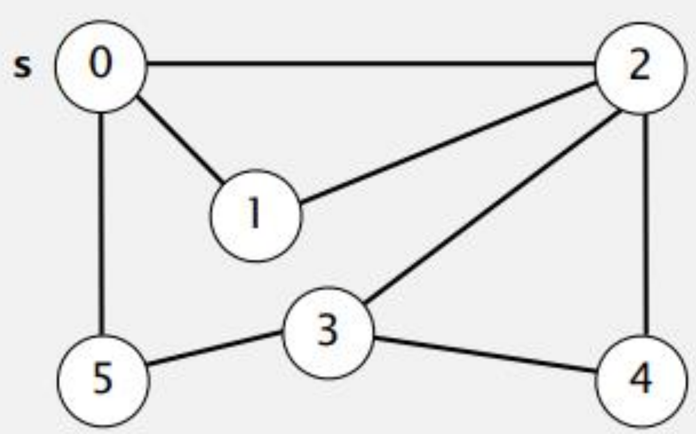
## Breadth-first search properties

Q. In which order does BFS examine vertices?

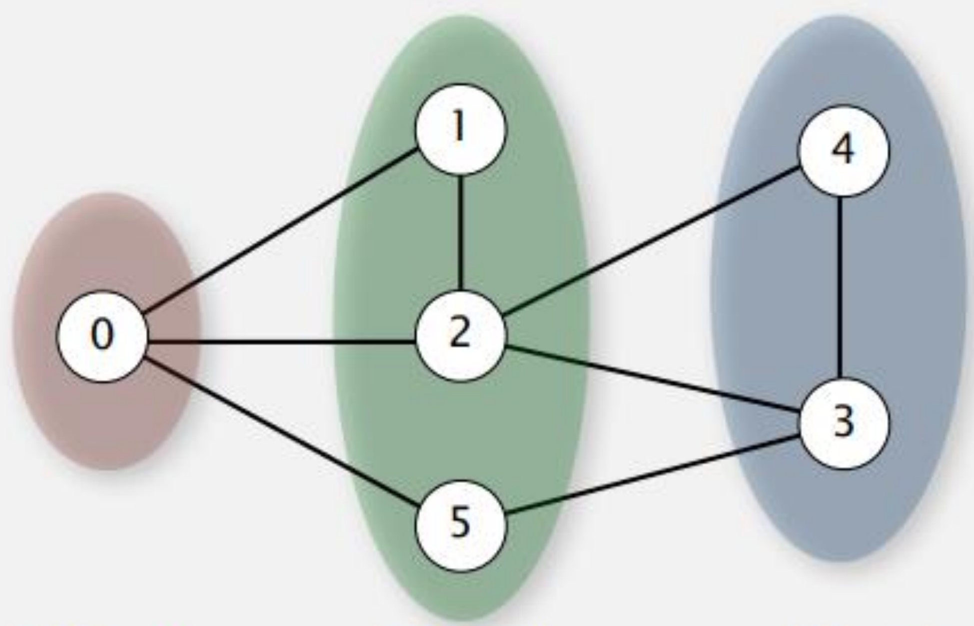
A. Increasing distance (number of edges) from $s$.

queue always consists of $\geq 0$ vertices of distance $k$ from $s$,
followed by $\geq 0$ vertices of distance $k+1$

Proposition. In any connected graph $G$, BFS computes shortest paths from $s$ to all other vertices in time proportional to $E + V$.




graph G

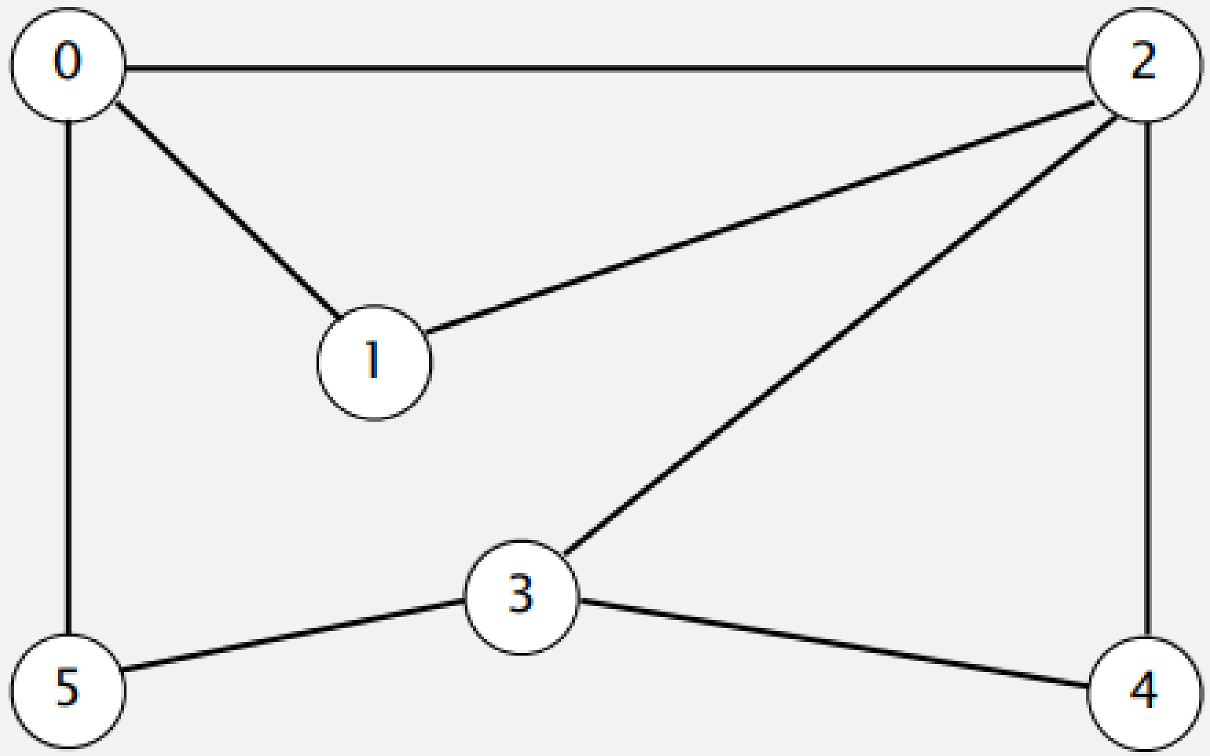



dist = 0 dist = 1 dist = 2

# Breadth-first search demo

Repeat until queue is empty:

- Remove vertex $v$ from queue.
- Add to queue all unmarked vertices adjacent to $v$ and mark them.




graph G

tinyCG.txt

```
V  6
   8  E
   0 5
   2 4
   2 3
   1 2
   0 1
   3 4
   3 5
   0 2
```

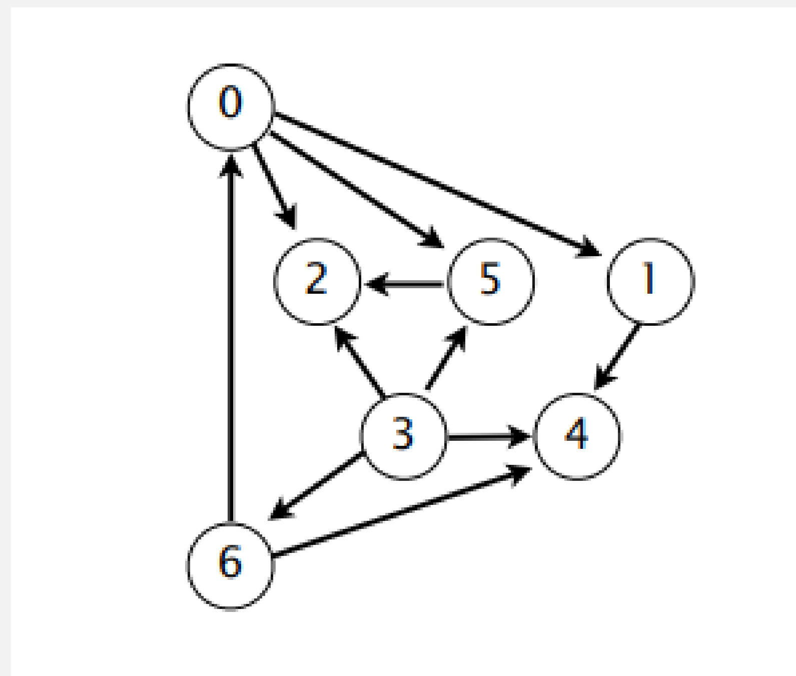
# Precedence scheduling

Goal. Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?

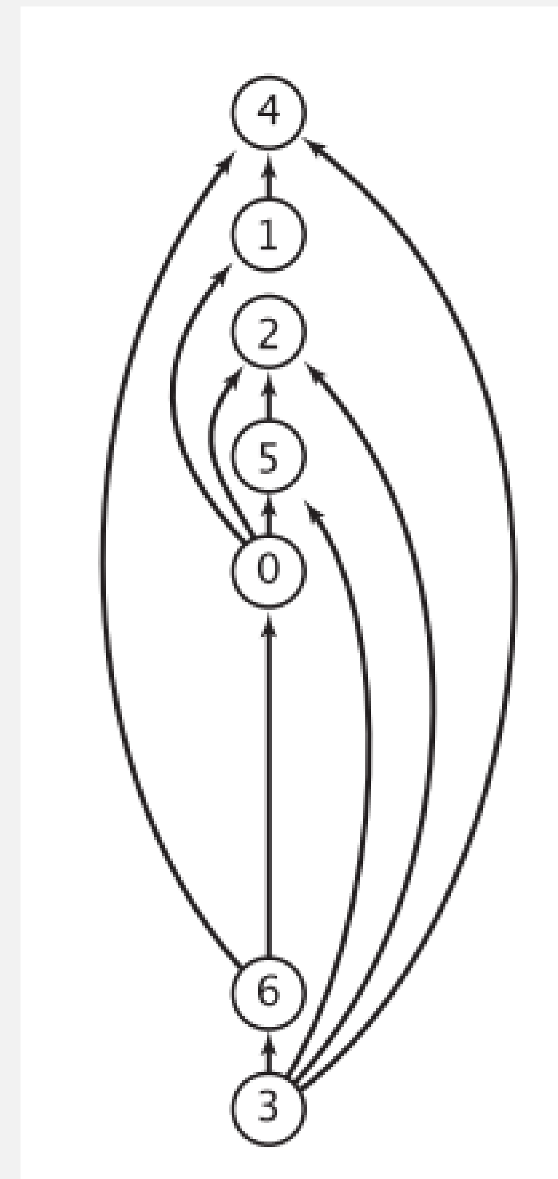
Digraph model. vertex = task; edge = precedence constraint.

0. Algorithms
1. Complexity Theory
2. Artificial Intelligence
3. Intro to CS
4. Cryptography
5. Scientific Computing
6. Advanced Programming

**tasks**




**precedence constraint graph**




**feasible schedule**

DAG. Directed acyclic graph.

Topological sort. Redraw DAG so all edges point upwards.

Solution. DFS. What else?

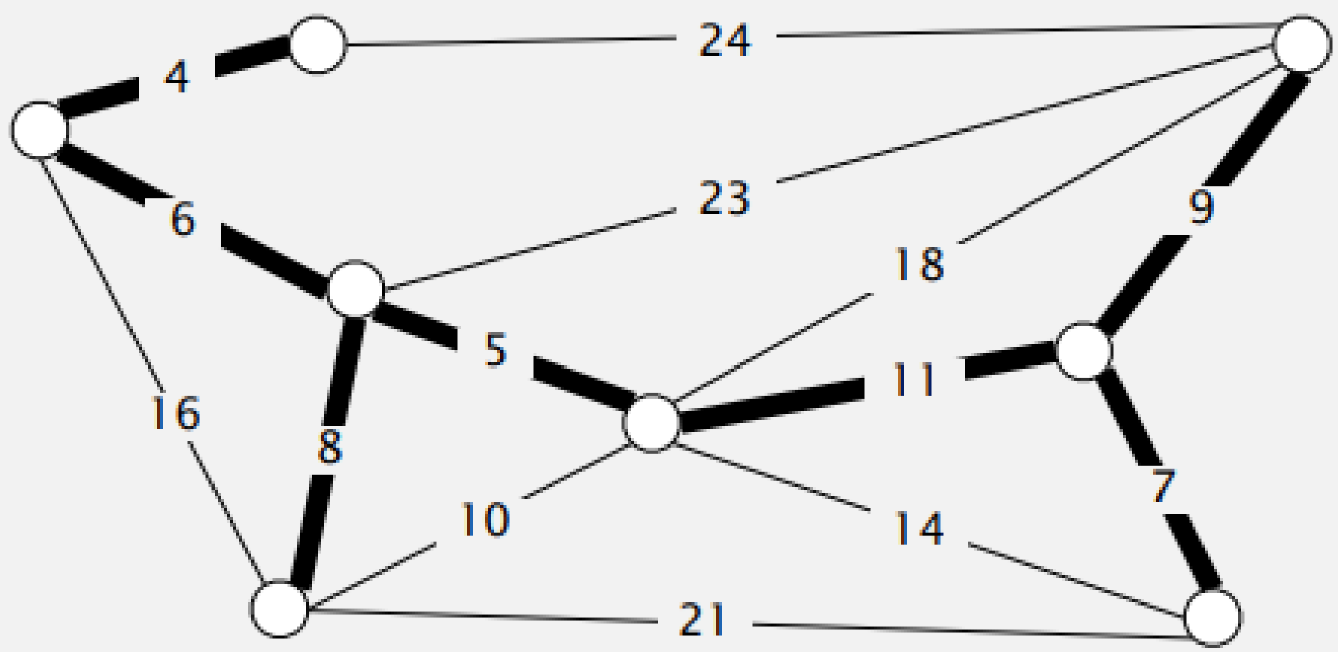
# Minimum spanning tree

Def. A spanning tree of $G$ is a subgraph $T$ that is:

- Connected.
- Acyclic.
- Includes all of the vertices.

Given. Undirected graph $G$ with positive edge weights (connected).

Goal. Find a min weight spanning tree.




**minimum spanning tree T**
**(cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7)**

# Shortest-Path Problem

- Google Maps - determining the fastest route

**Driving directions to Diddy Riese**

Suggested routes

| | |
|---|---|
| **1. Strathmore Pl and Gayley Ave** **0.6 mi** | **3 mins** |
| 2. Strathmore Pl and Westwood Plaza 0.6 mi | 3 mins |
| 3. Strathmore Pl 0.6 mi | 3 mins |

A **University of California Los Angeles**
405 Hilgard Ave
Los Angeles, CA 90095

**1.** Head **west** on **Strathmore Pl** toward **Charles E Young Dr**

0.1 mi

**2.** Turn left at **Gayley Ave**

0.4 mi

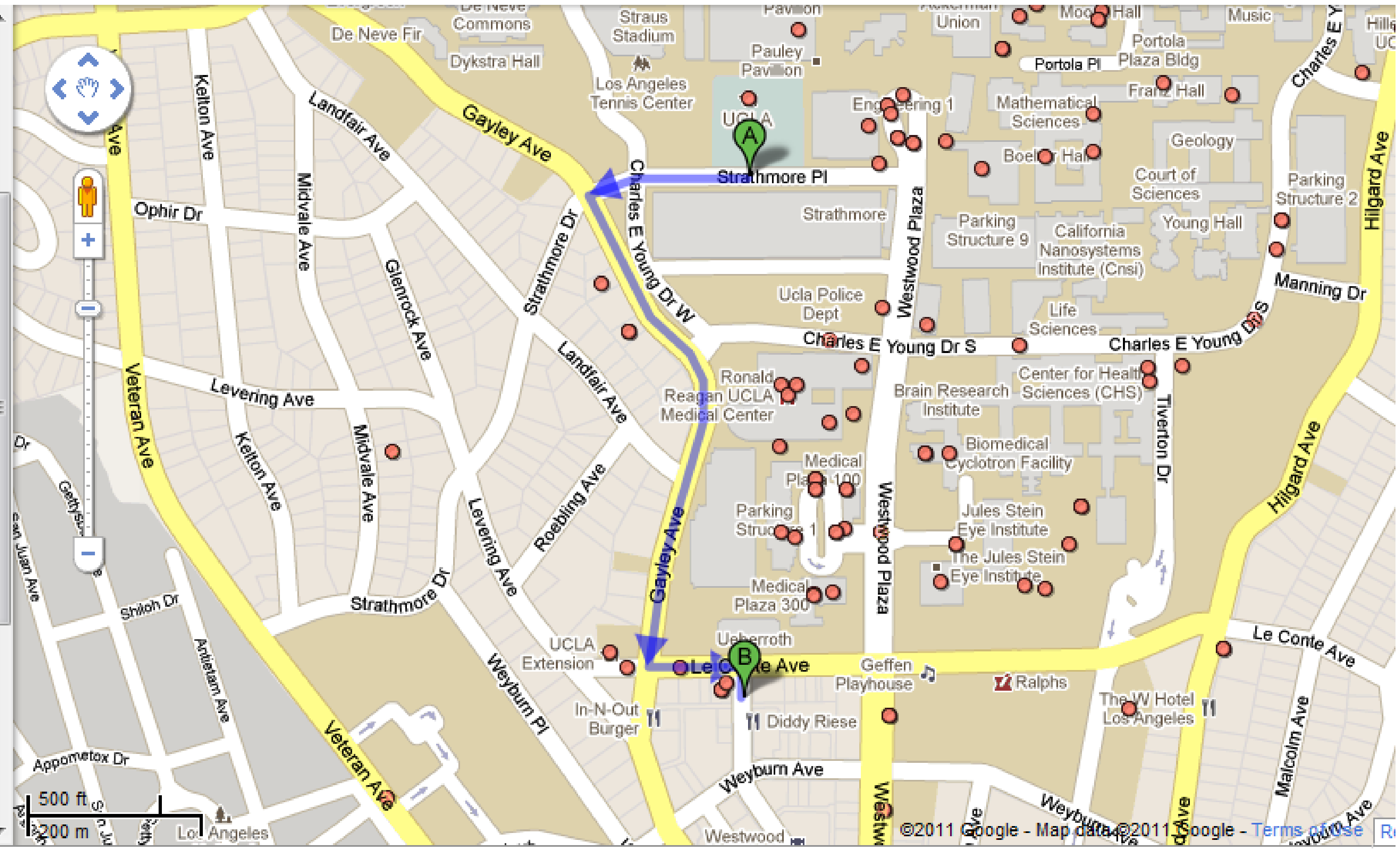
**3.** Turn left at **Le Conte Ave**

354 ft

**4.** Take the 1st right onto **Broxton Ave**

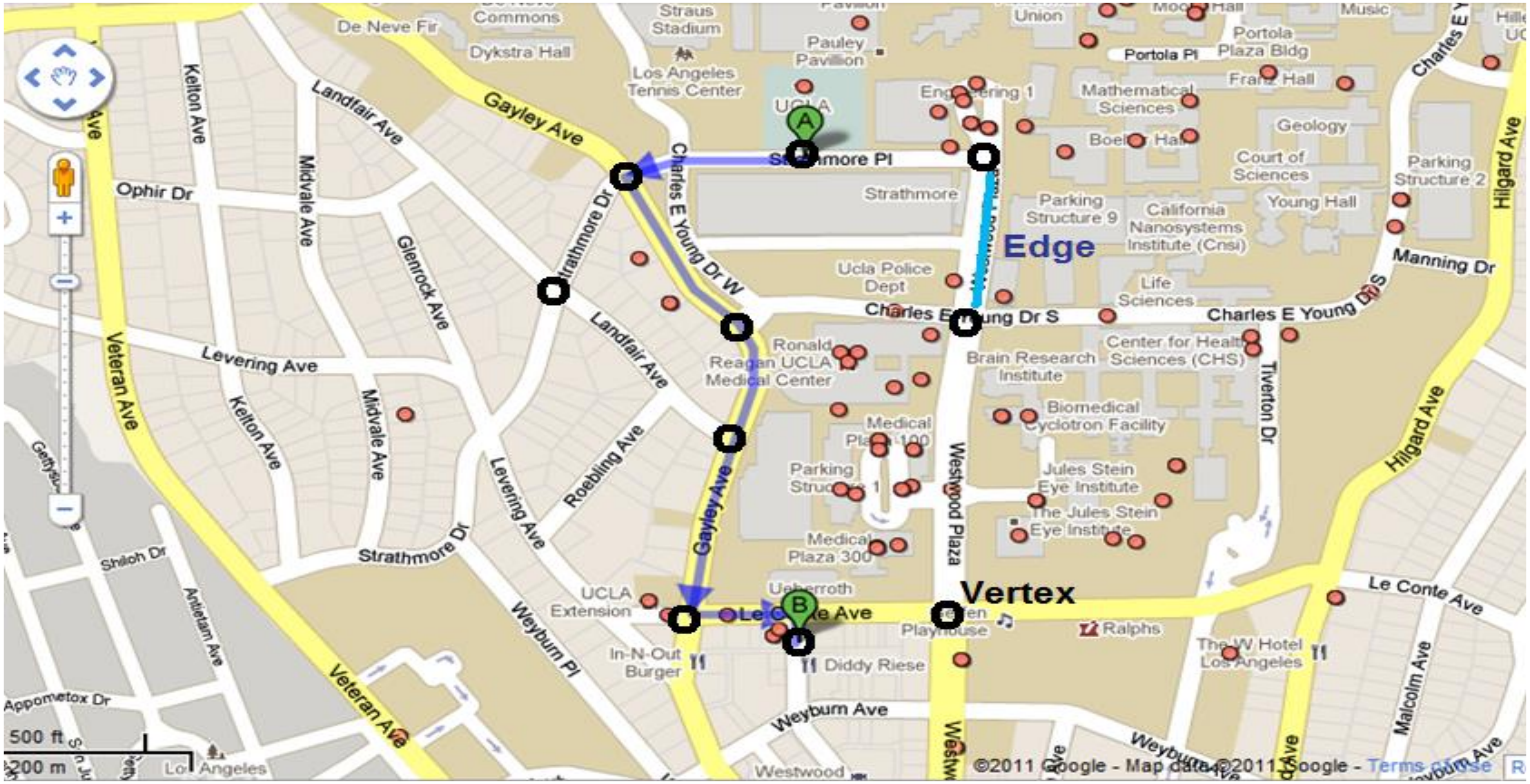
115 ft

B **Diddy Riese**
926 Broxton Avenue



# Shortest-Path Problem

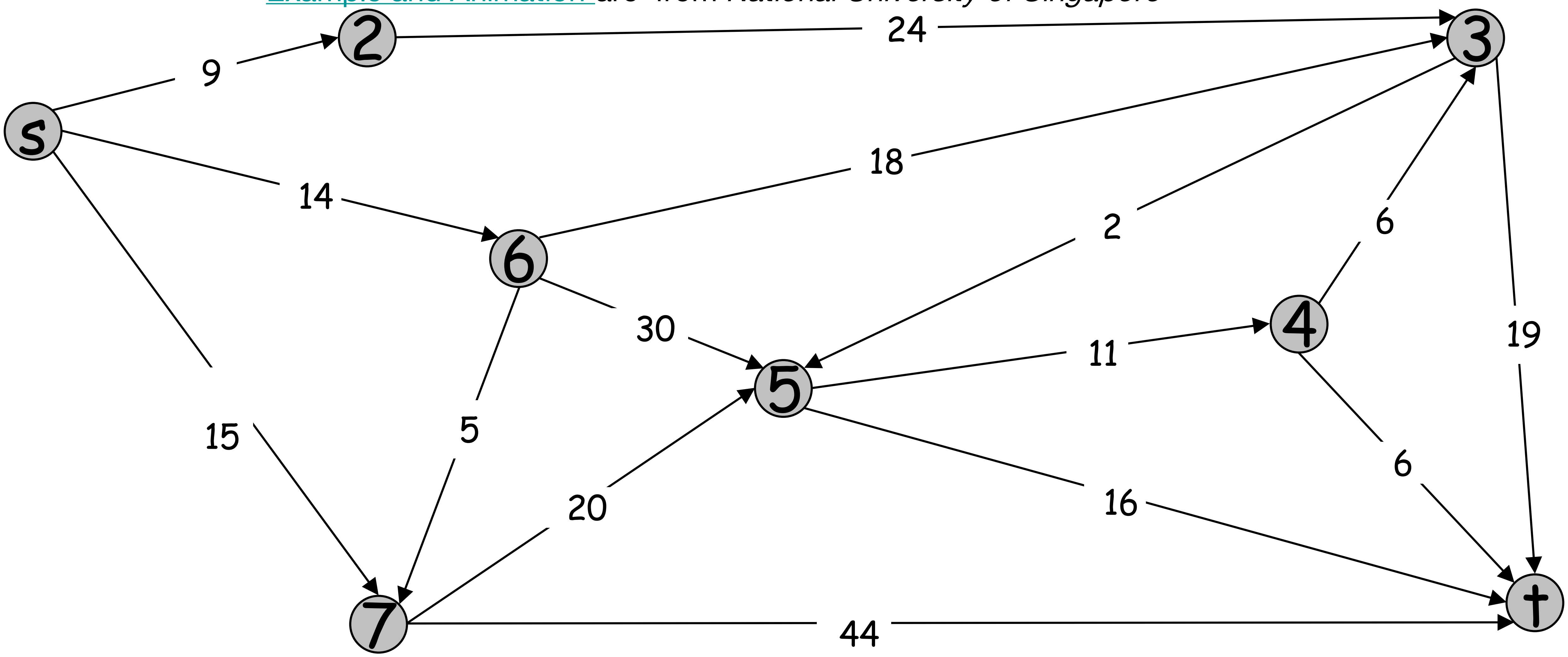
- Google Maps - determining the fastest route



# Class Activity: Shortest Path Algorithm

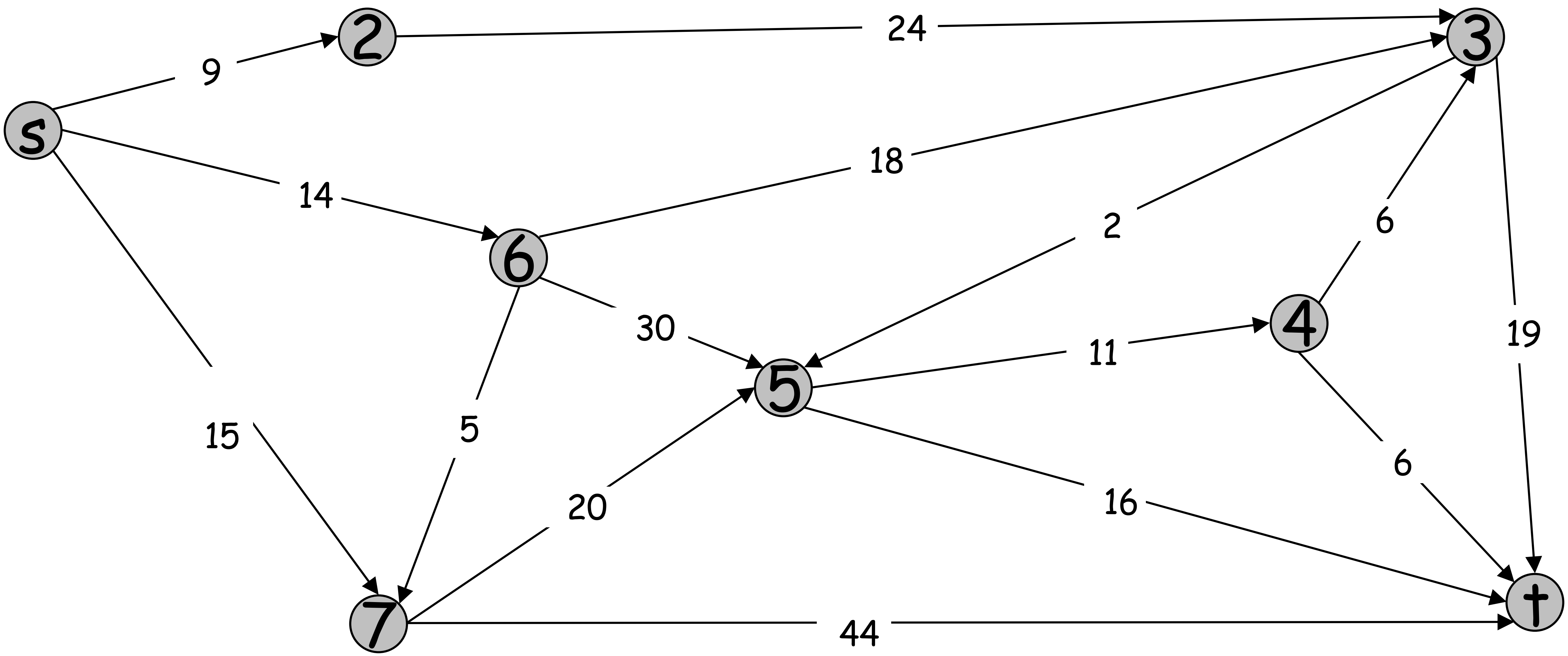
- Can you come up with an algorithm that solves the shortest path problem?
- Work in groups
- What is the shortest path from **s** to **t**?

Example and Animation are from *National University of Singapore*



# Dijsktra's Shortest Path Algorithm

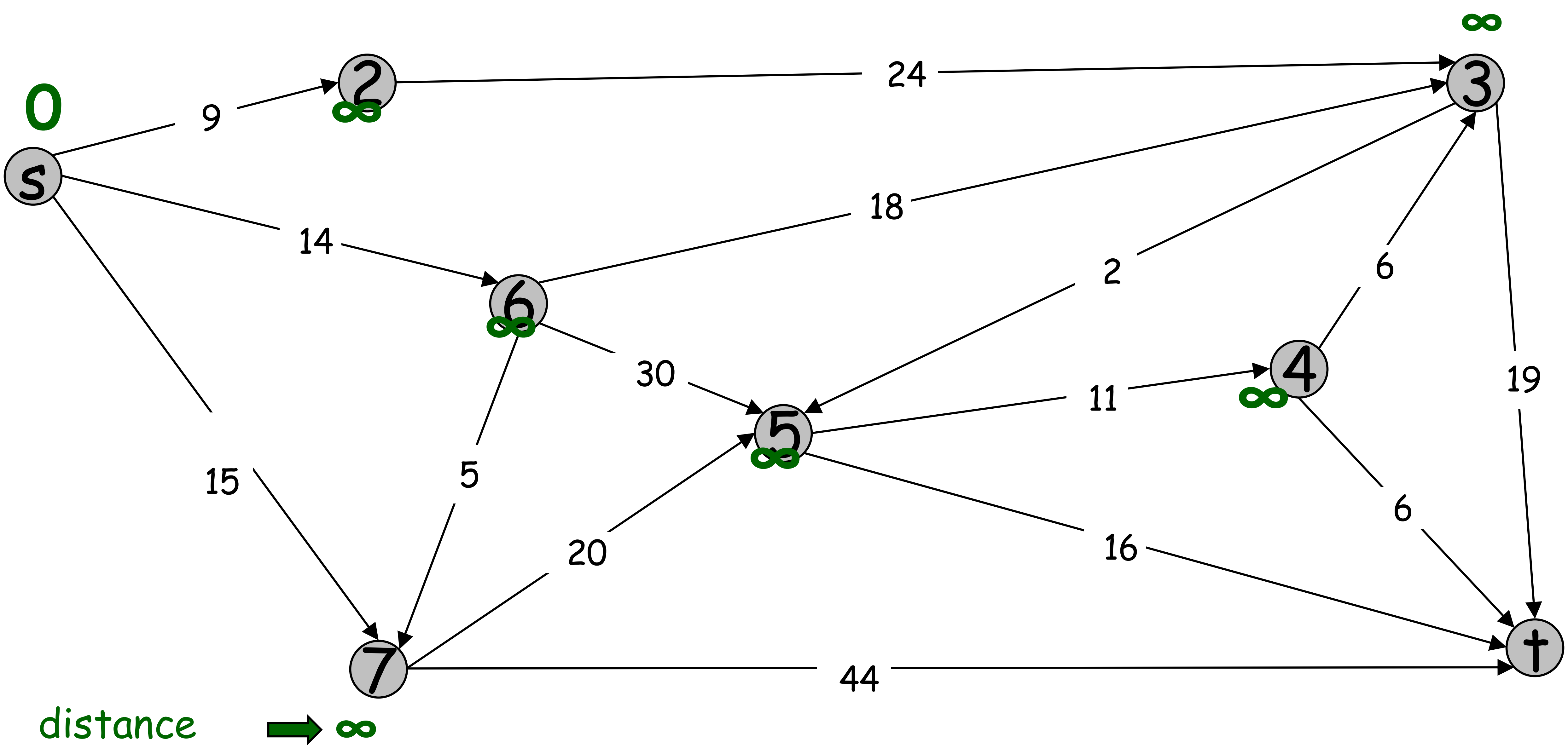
Find shortest path from s to t.




Example and Animation are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

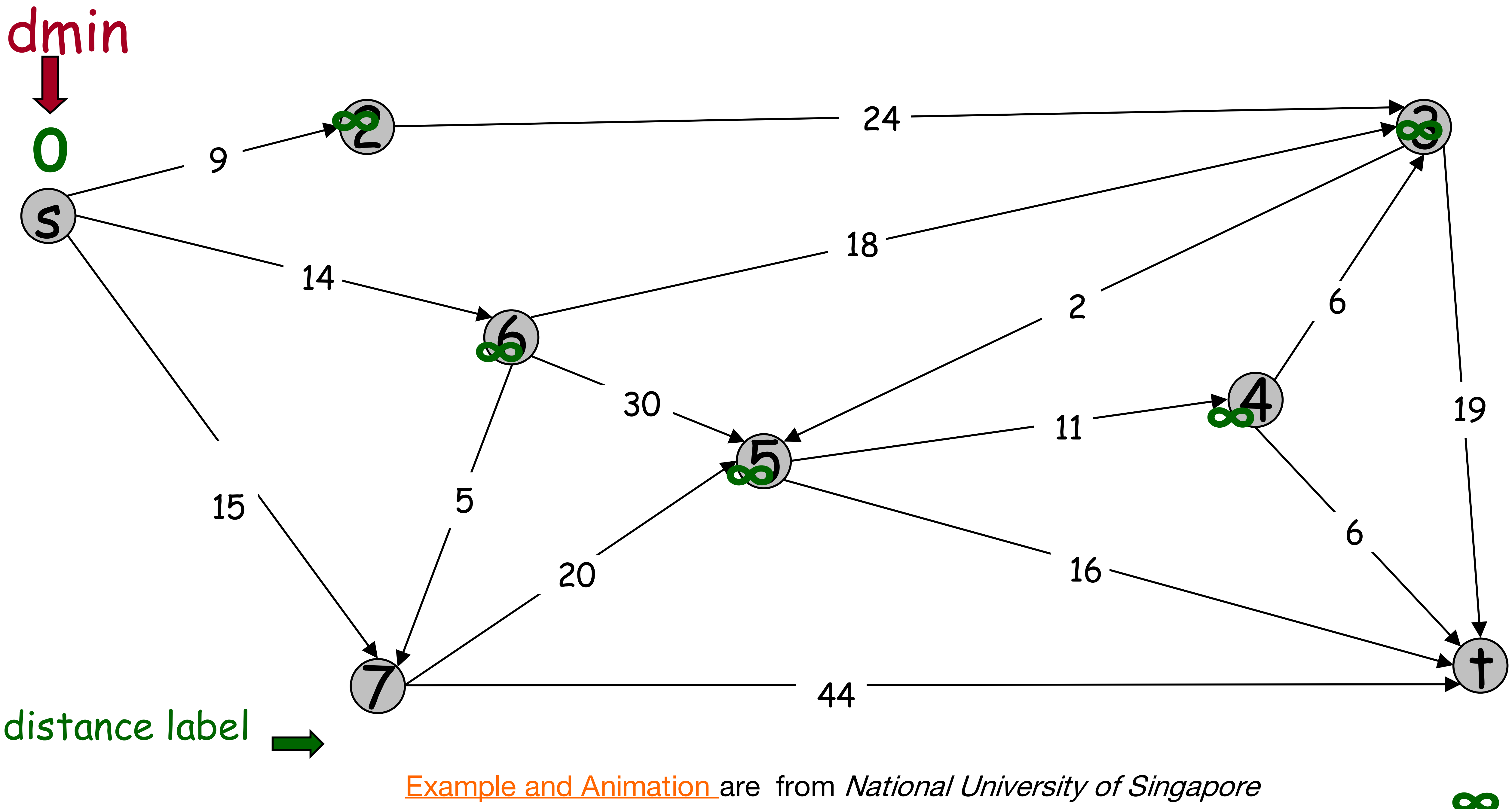
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }
B = { }




[Example and Animation](#) are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

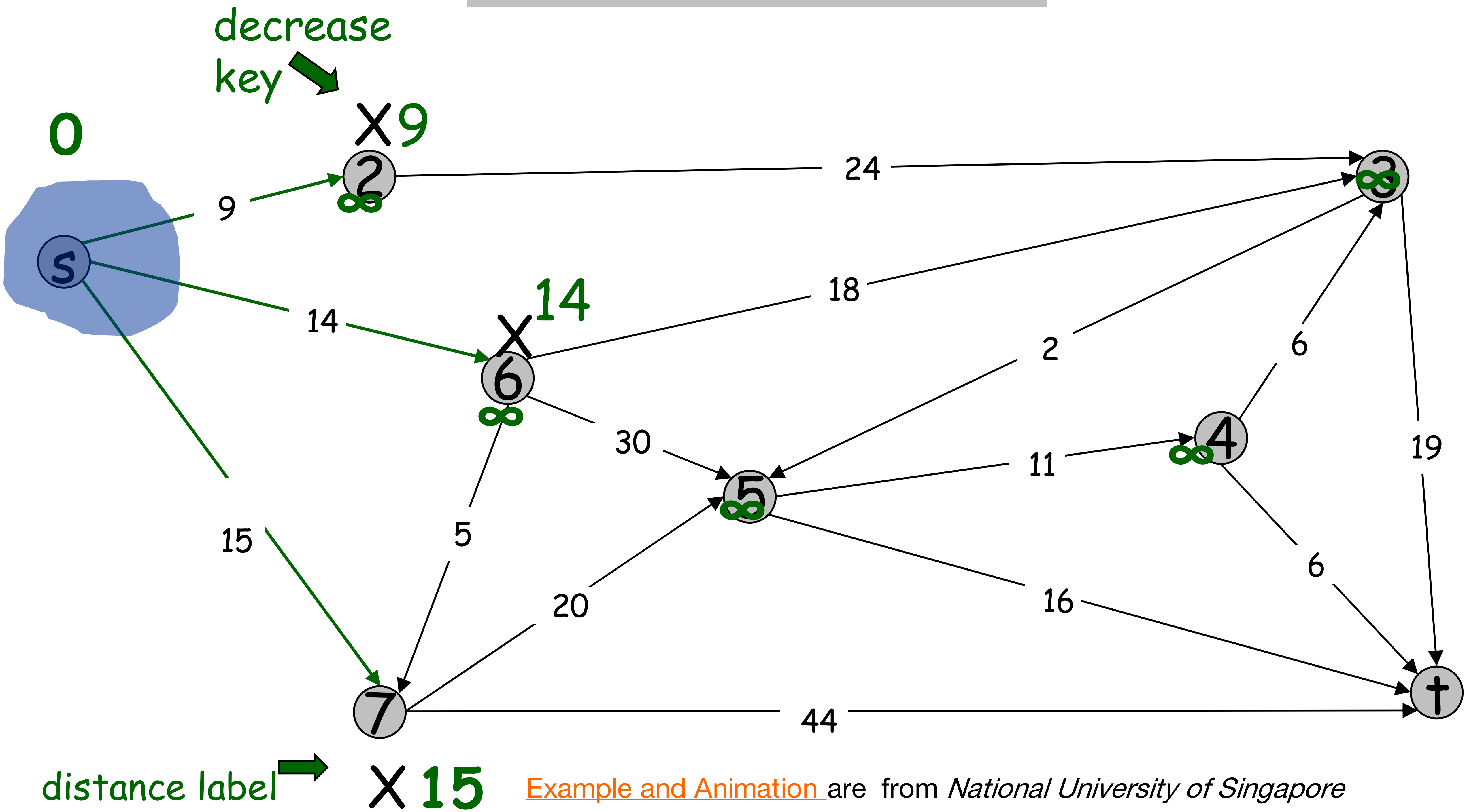
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }
B = { }




Example and Animation are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

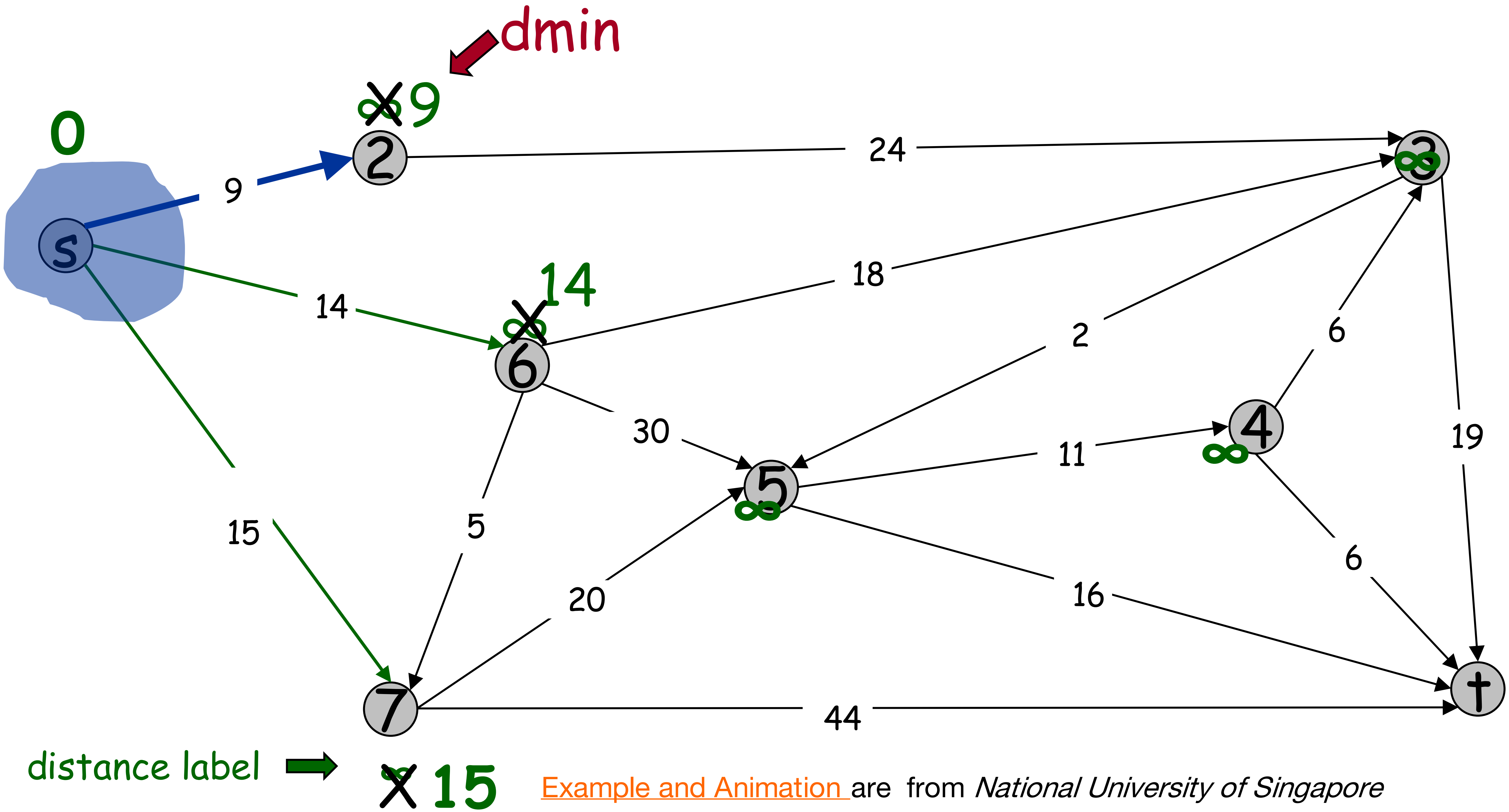
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }
B = { s }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

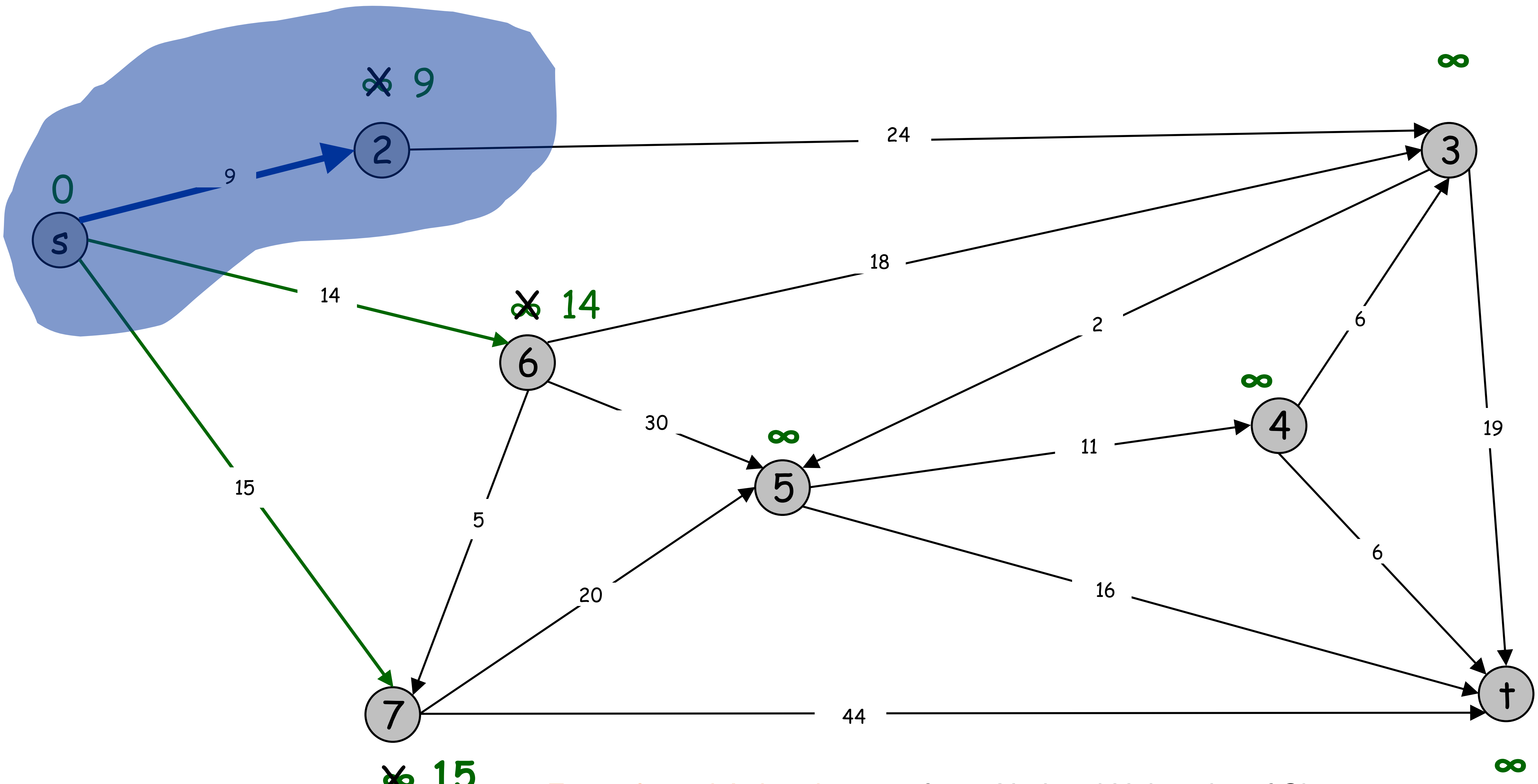
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }
B = { s }




[Example and Animation ](#)are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

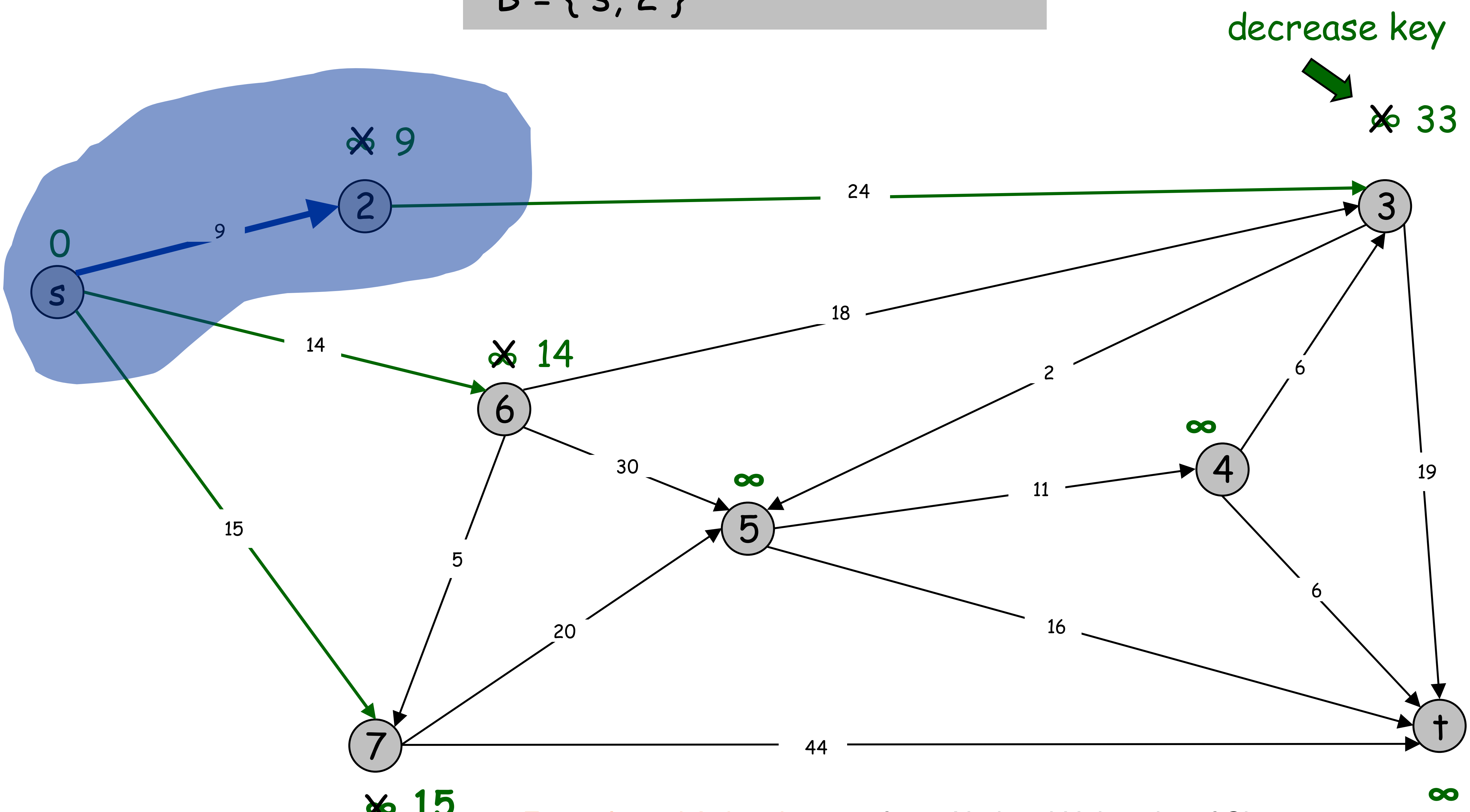
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

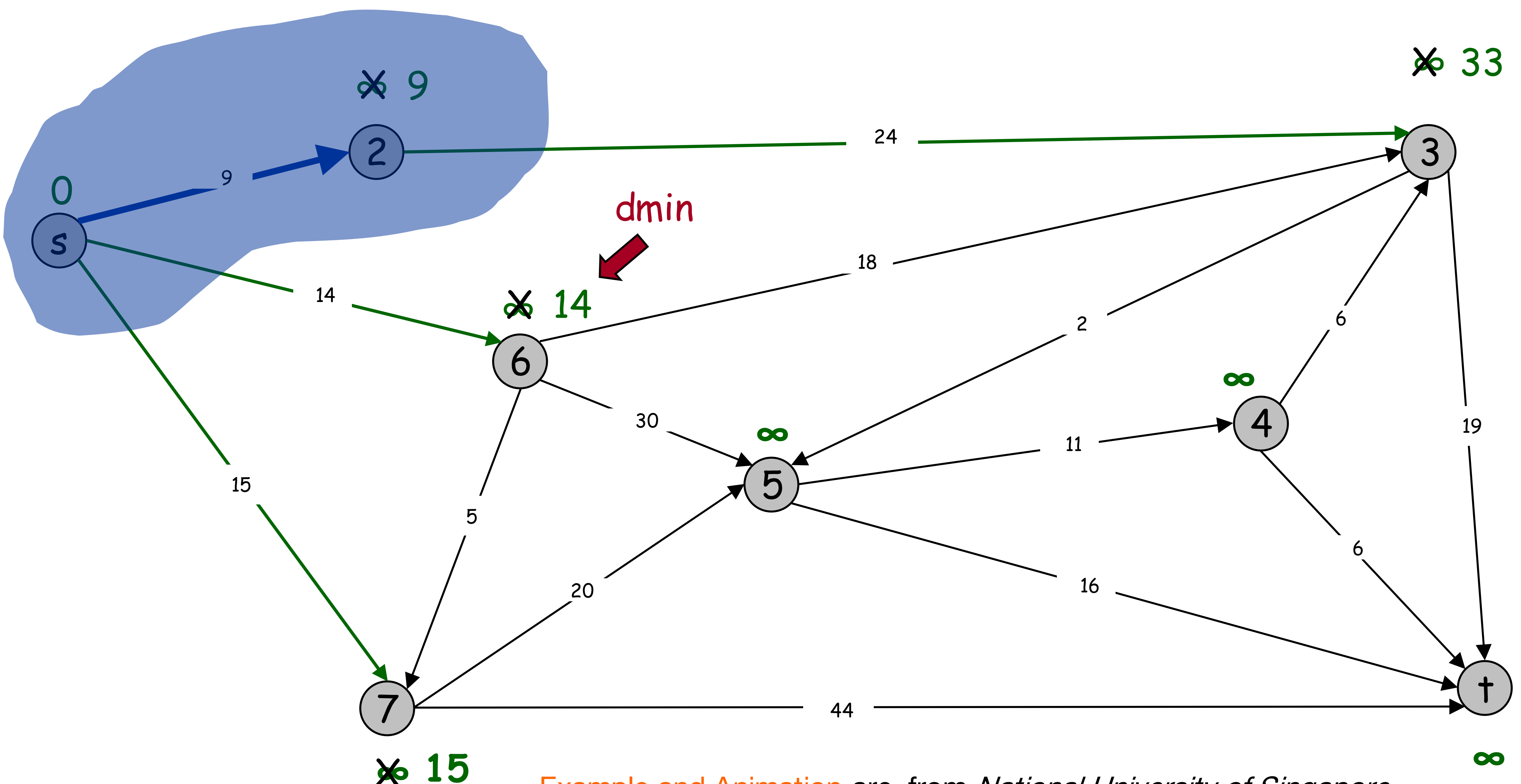
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

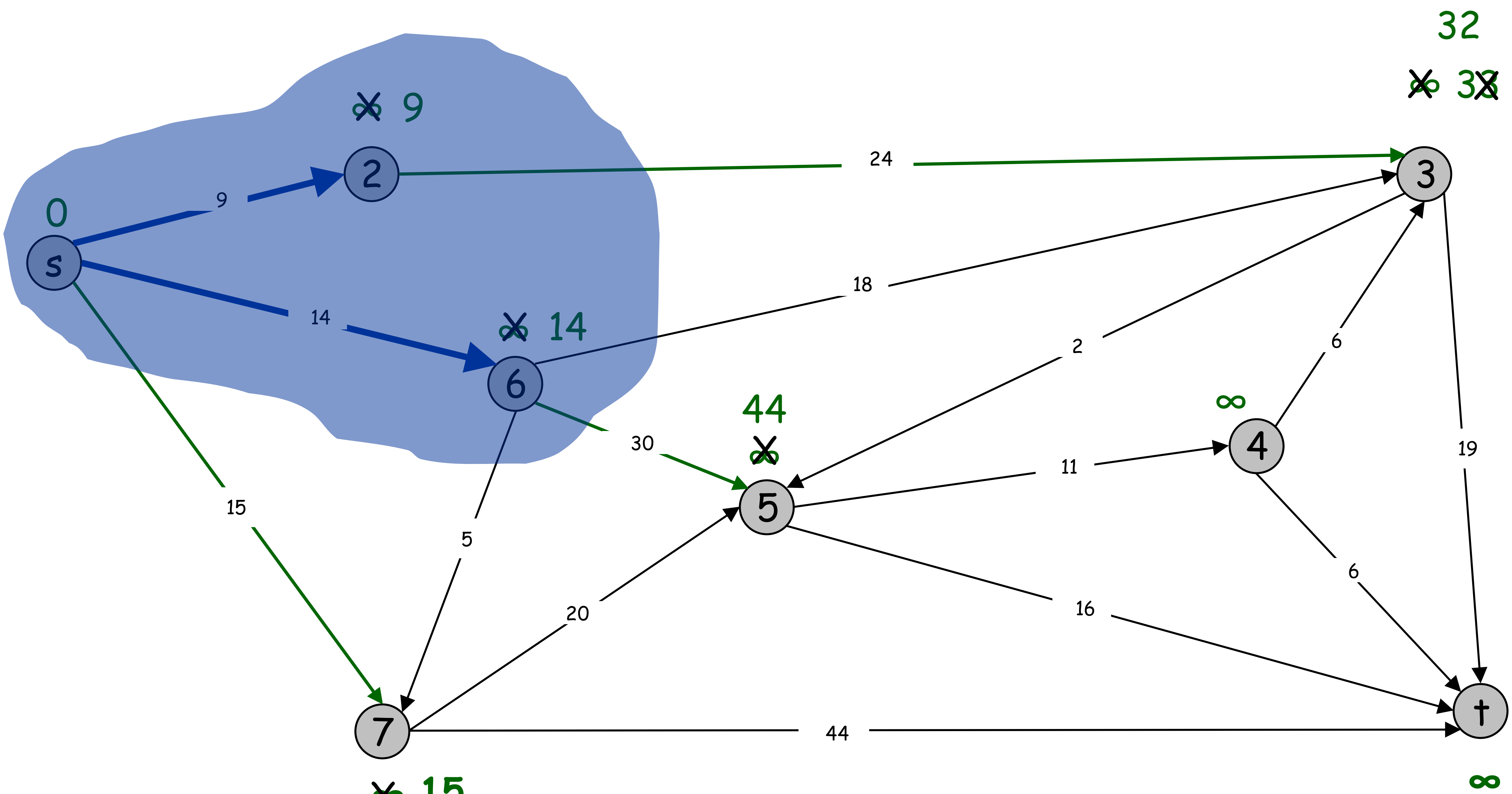
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }




[Example and Animation ]are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

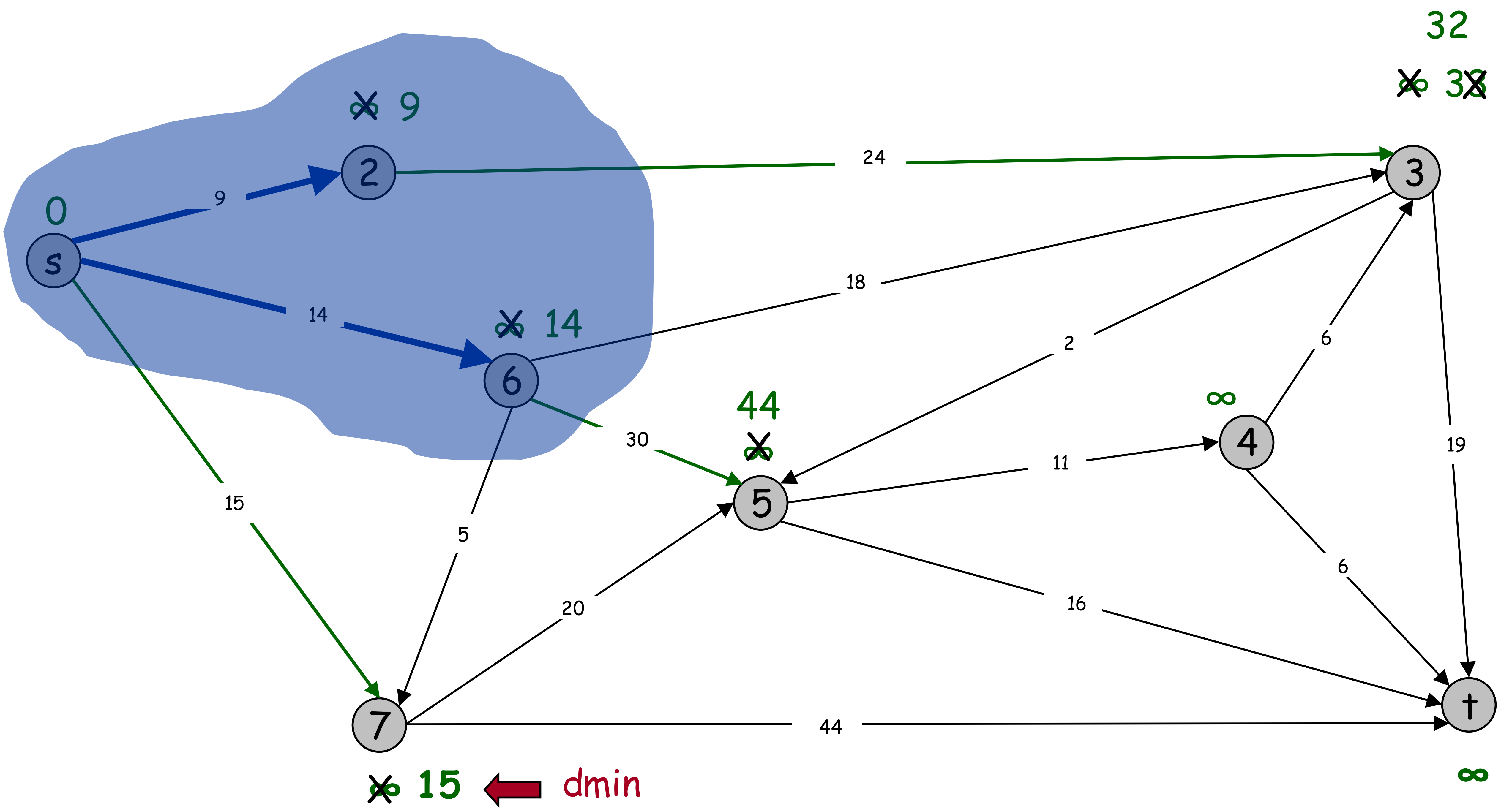
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }
B = { s, 6 }




[Example and Animation](#) are from *National University of Singapore*

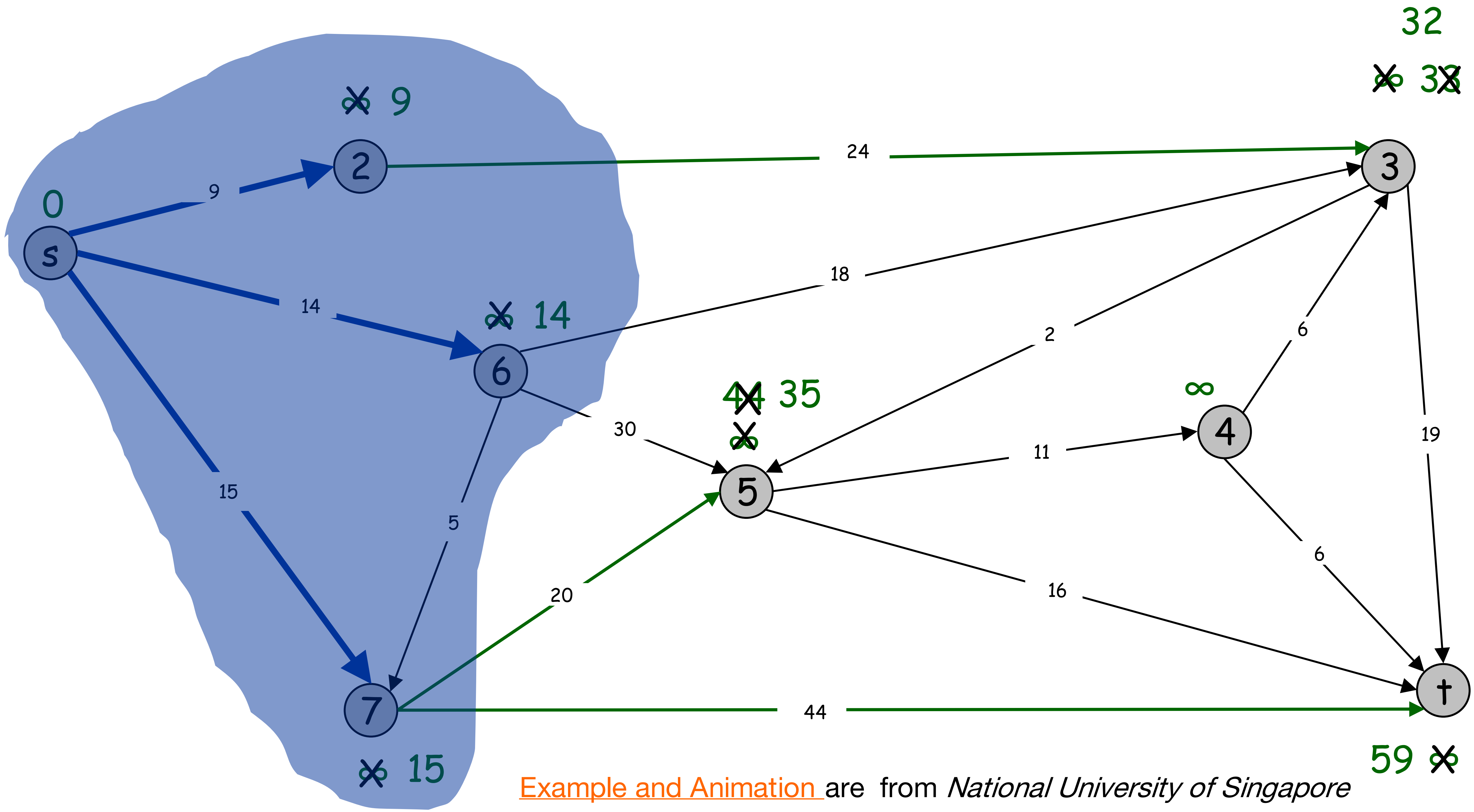
# Dijsktra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }
B = { s, 6 }



# Dijsktra’s Shortest Path Algorithm

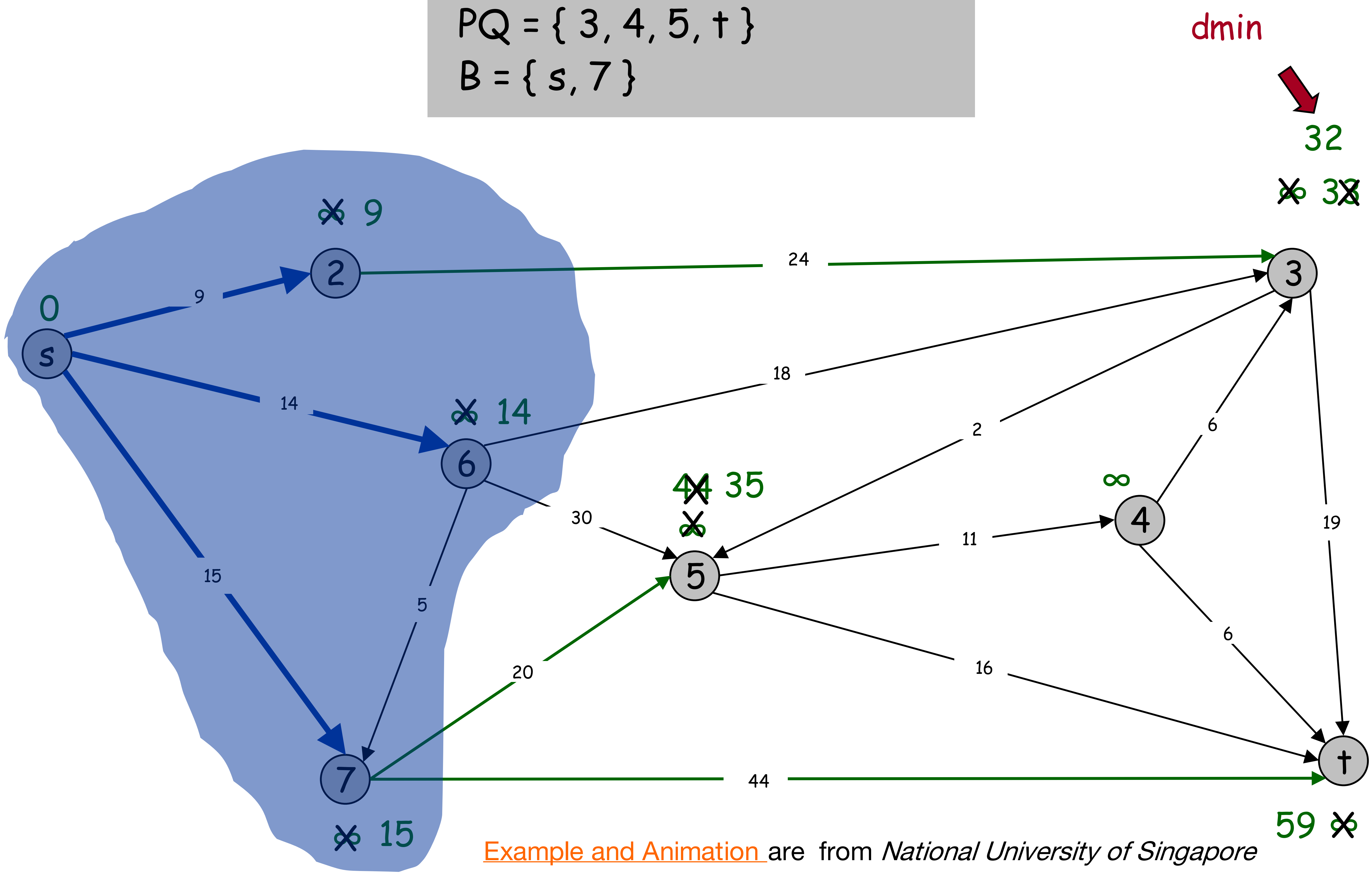
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }
B = { s, 7 }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

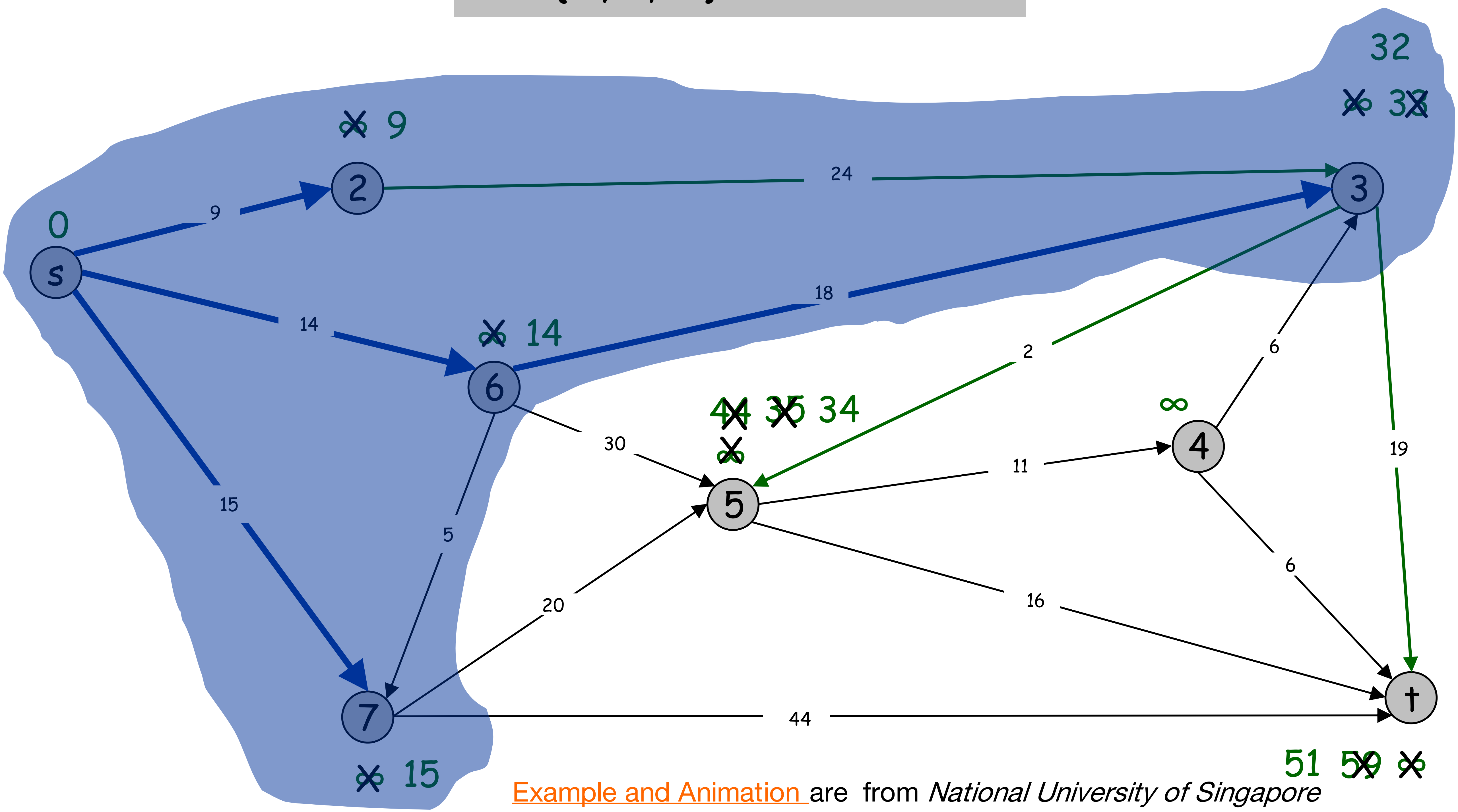
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }
B = { s, 7 }




[Example and Animation] are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

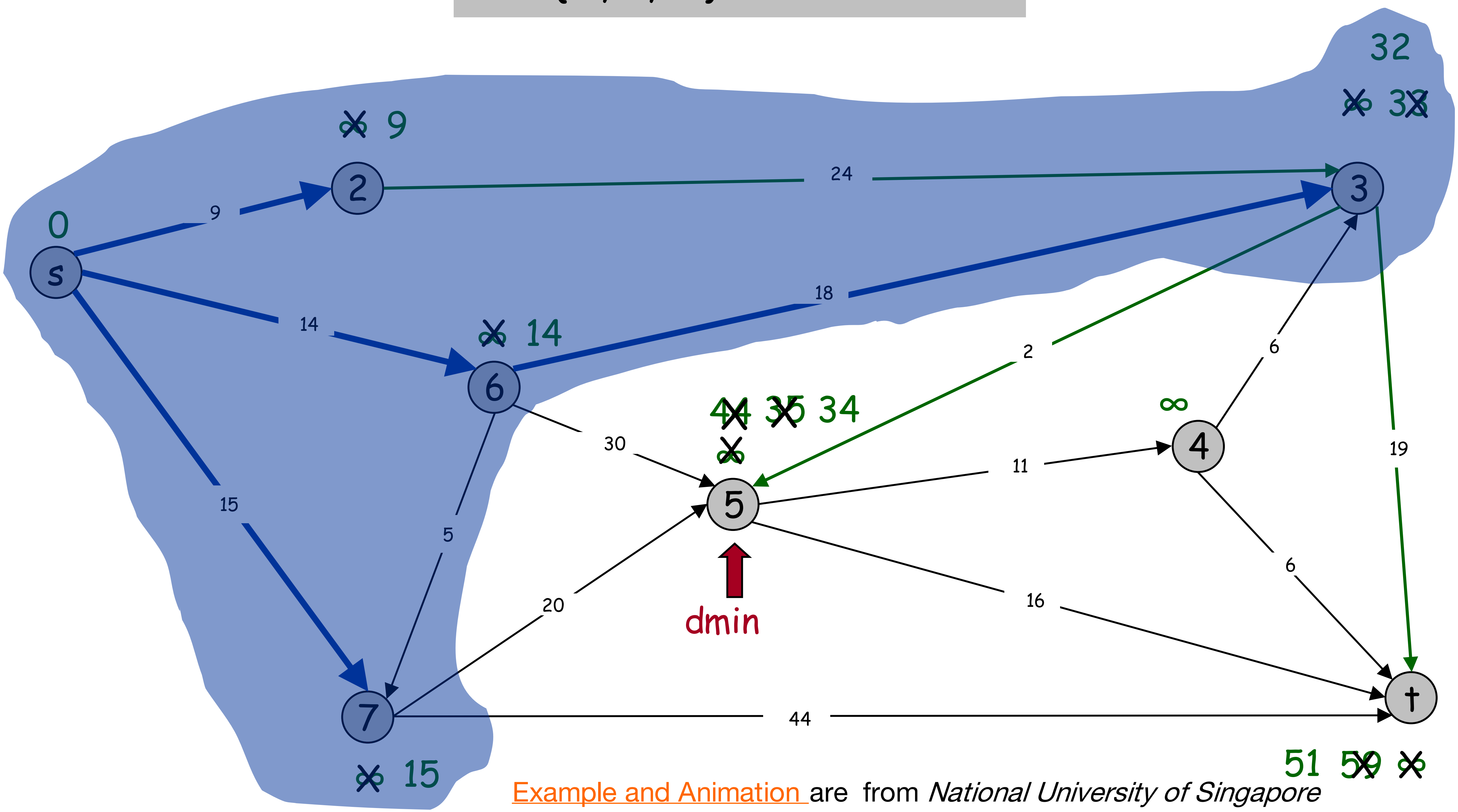
S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }
B = { s, 6, 3 }




[Example and Animation](#) are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

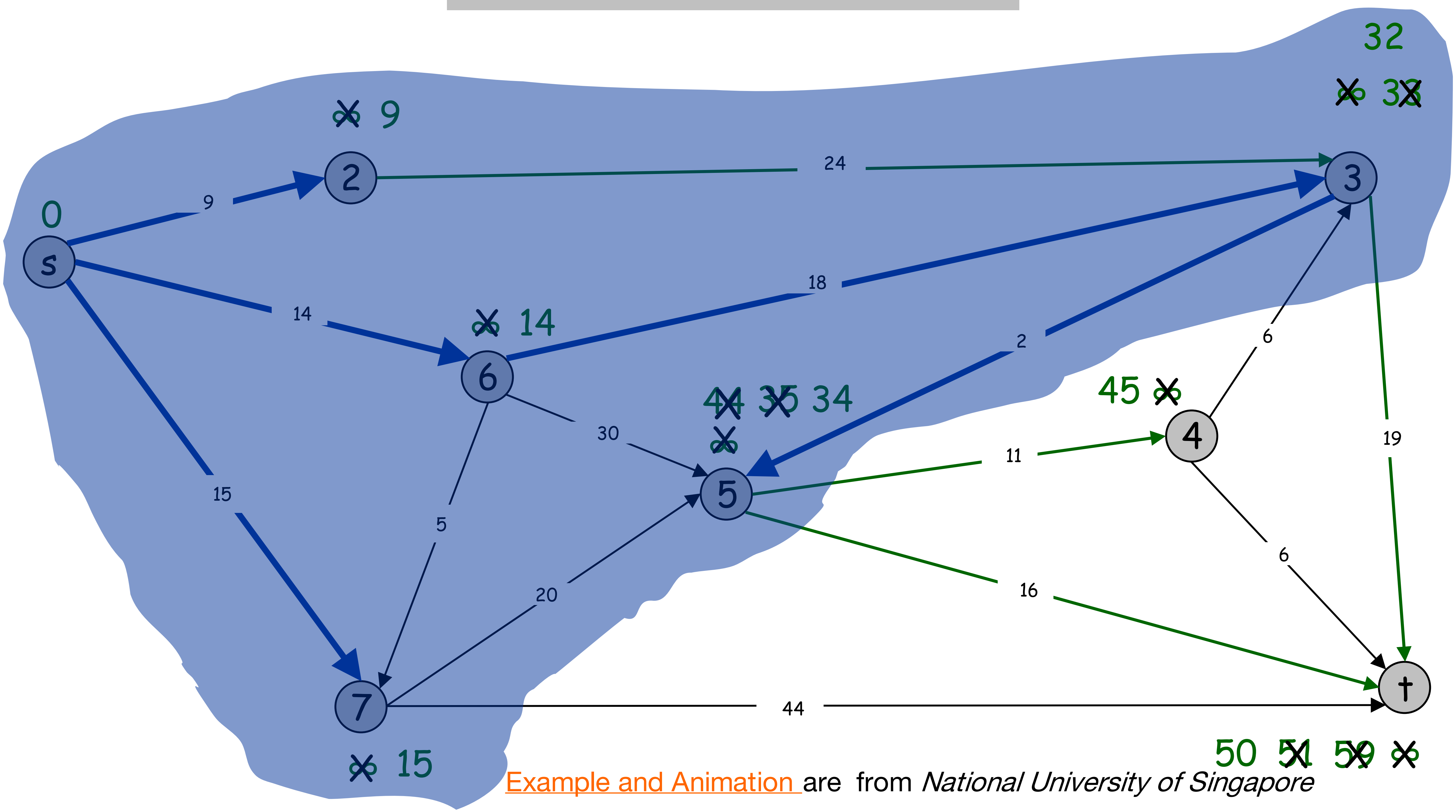
S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }
B = { s, 6, 3 }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

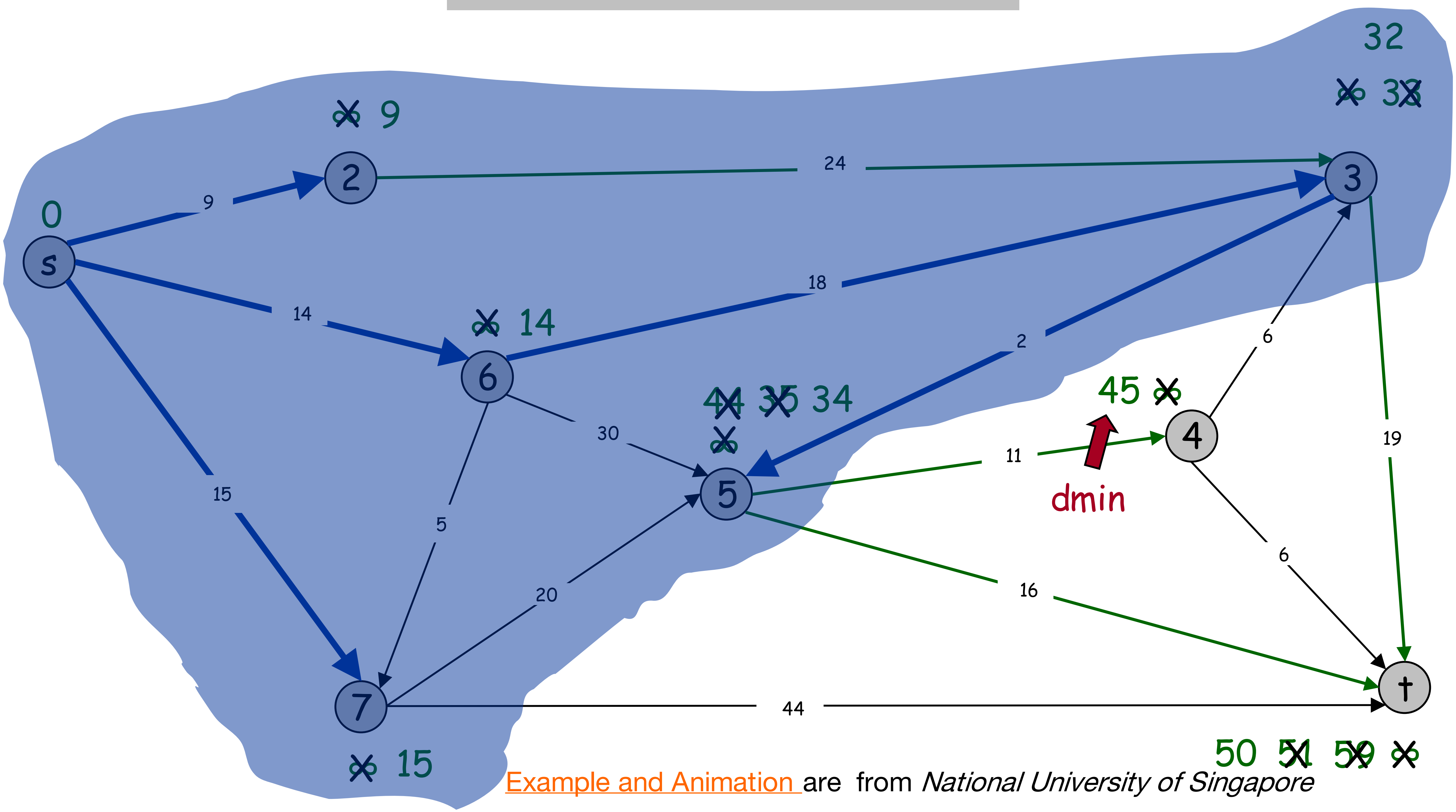
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }
B = { s, 6, 3, 5 }




Example and Animation are from *National University of Singapore*

# Dijsktra’s Shortest Path Algorithm

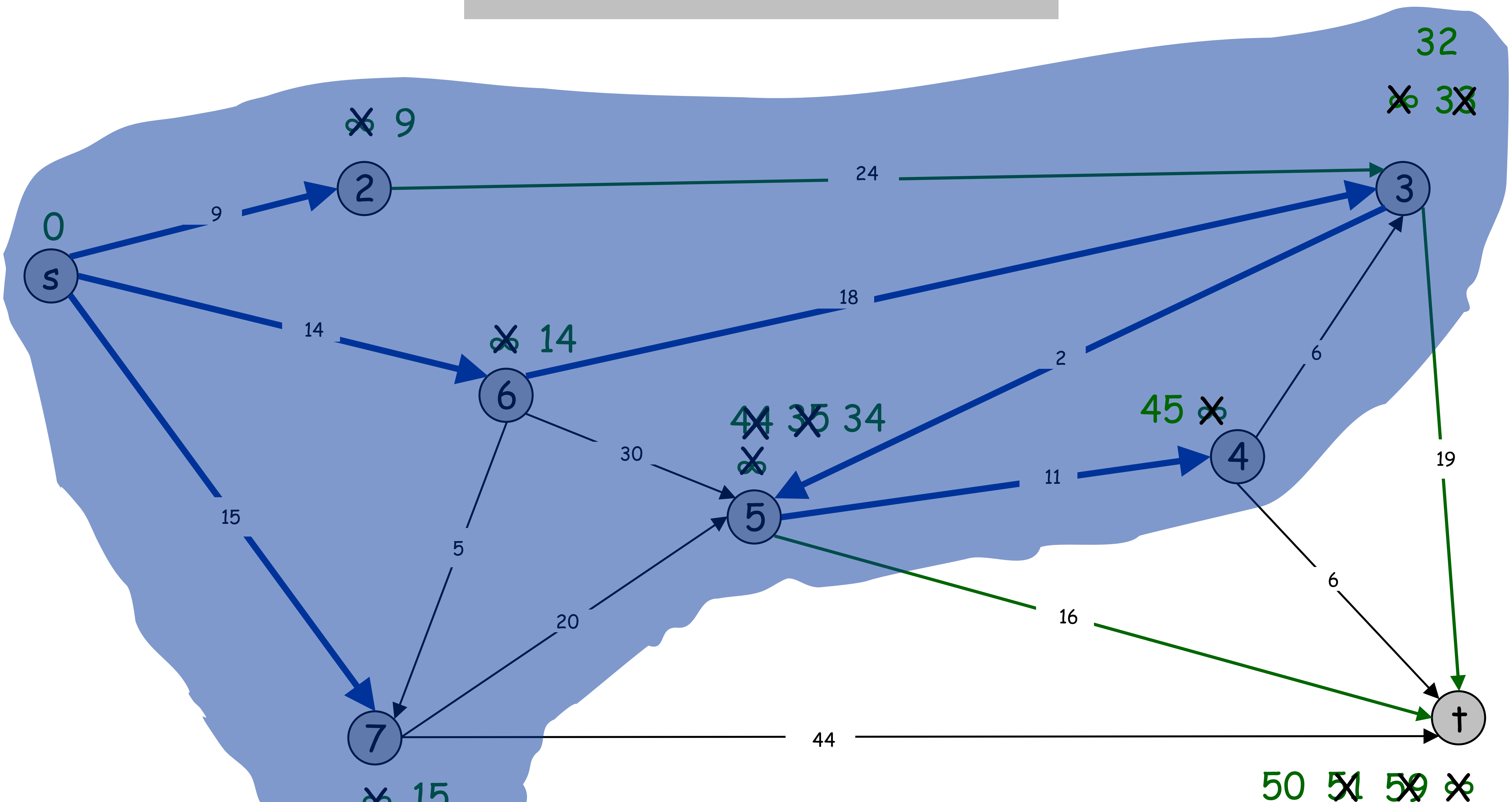
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }
B = { s, 6, 3, 5 }




Example and Animation are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

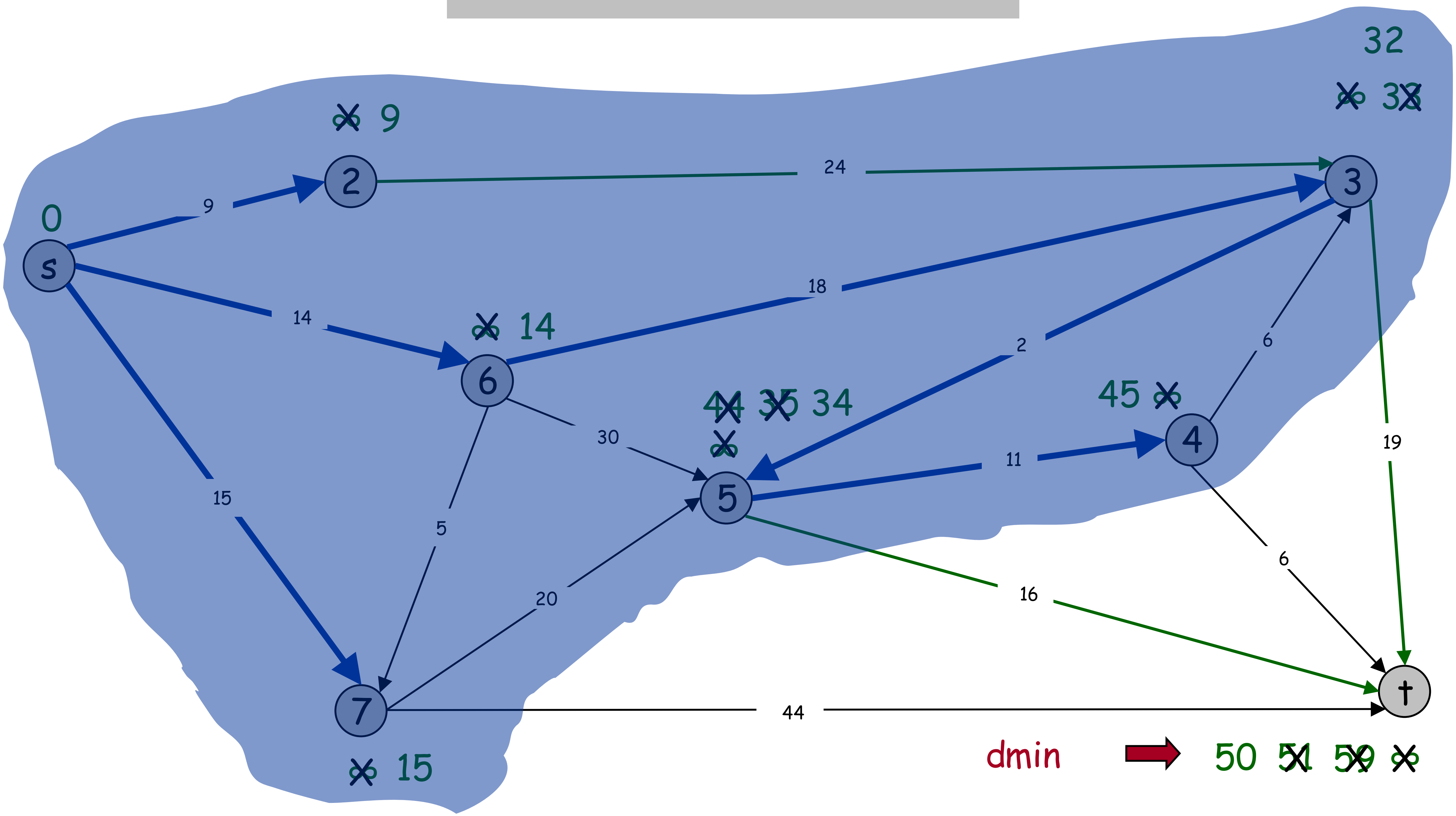
S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }
B = { s, 6, 3, 5, 4 }




Example and Animation are from *National University of Singapore*

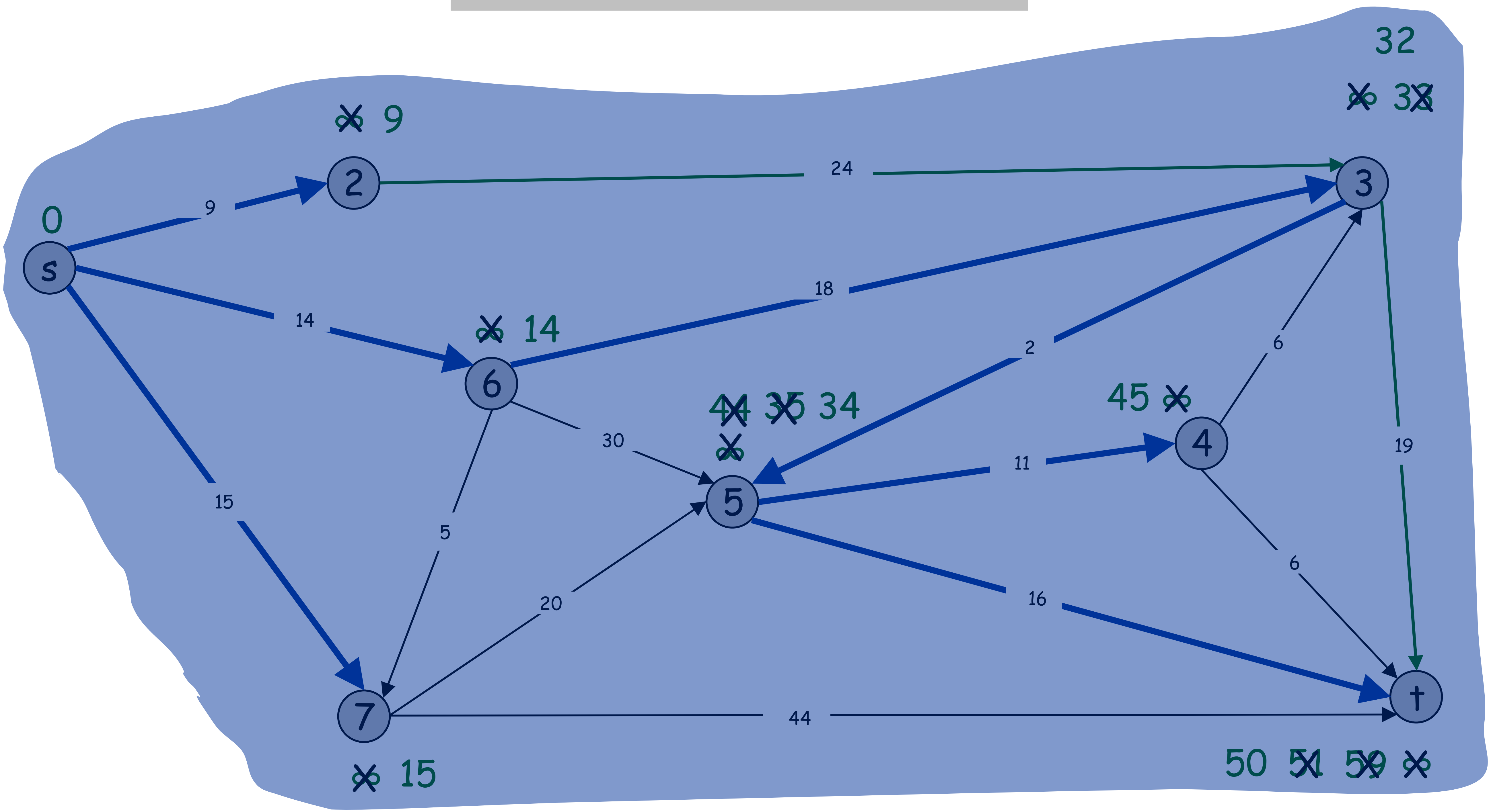
# Dijsktra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }
B = { s, 6, 3, 5, 4 }



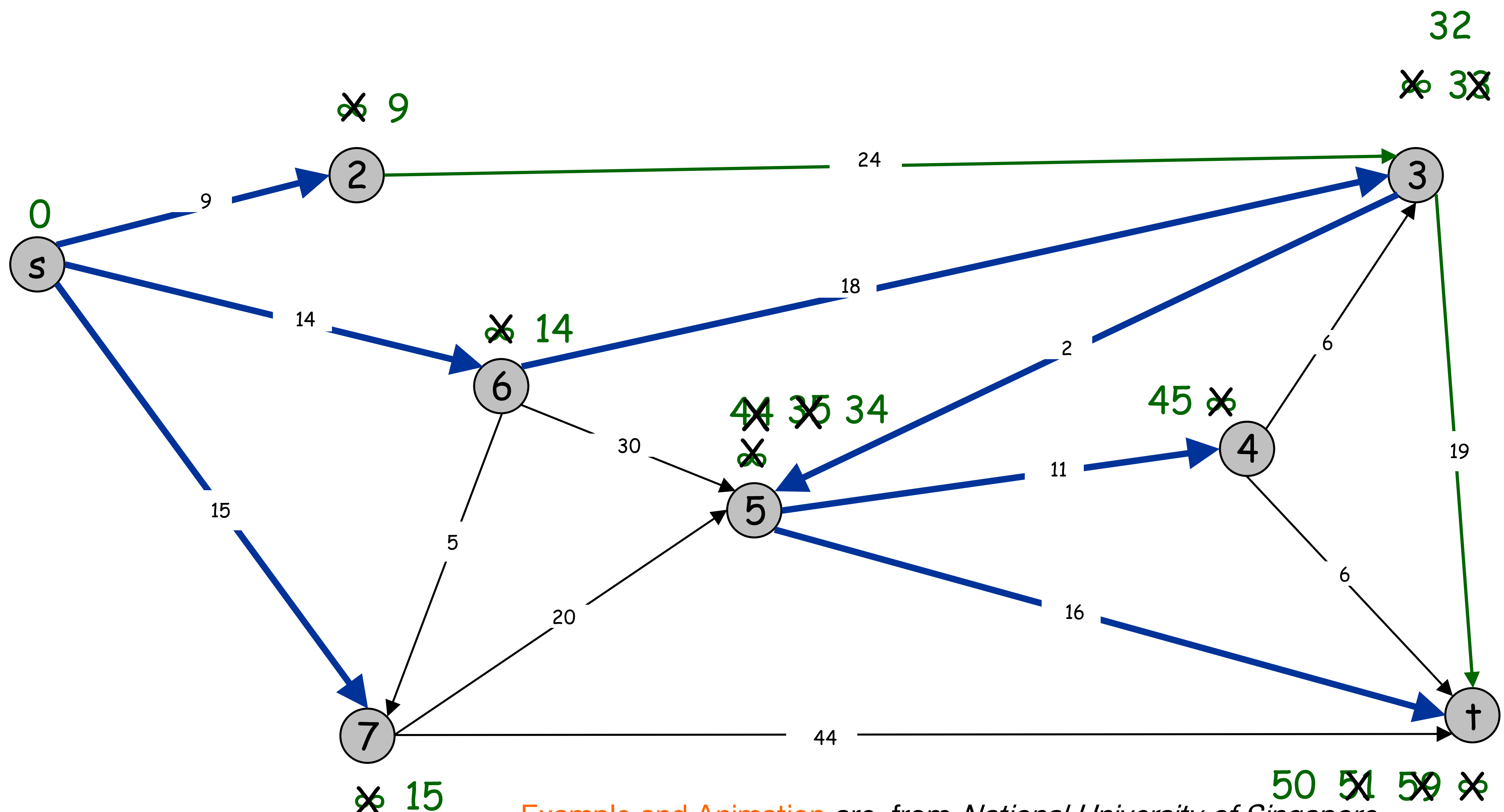
# Dijsktra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }
B = { s, 6, 3, 5, t }



# Dijsktra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }
B = { s, 6, 3, 5, t }




Example and Animation are from *National University of Singapore*

# Dijsktra's Shortest Path Algorithm

```
function Dijkstra(Graph, source):

    dist[source] ← 0                          // Distance from source to source
    prev[source] ← undefined                  // Previous node in optimal path initialization

    for each vertex v in Graph:  // Initialization
        if v ≠ source:           // Where v has not yet been removed from Q (unvisited nodes)
            dist[v] ← infinity                // Unknown distance function from source to v
            prev[v] ← undefined               // Previous node in optimal path from source
        end if
        add v to Q                        // All nodes initially in Q (unvisited nodes)
    end for

    while Q is not empty:
        u ← vertex in Q with min dist[u]  // Source node in first case
        remove u from Q

        for each neighbor v of u:              // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:                  // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u
            end if
        end for
    end while

    return dist[], prev[]

end function
```

Source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

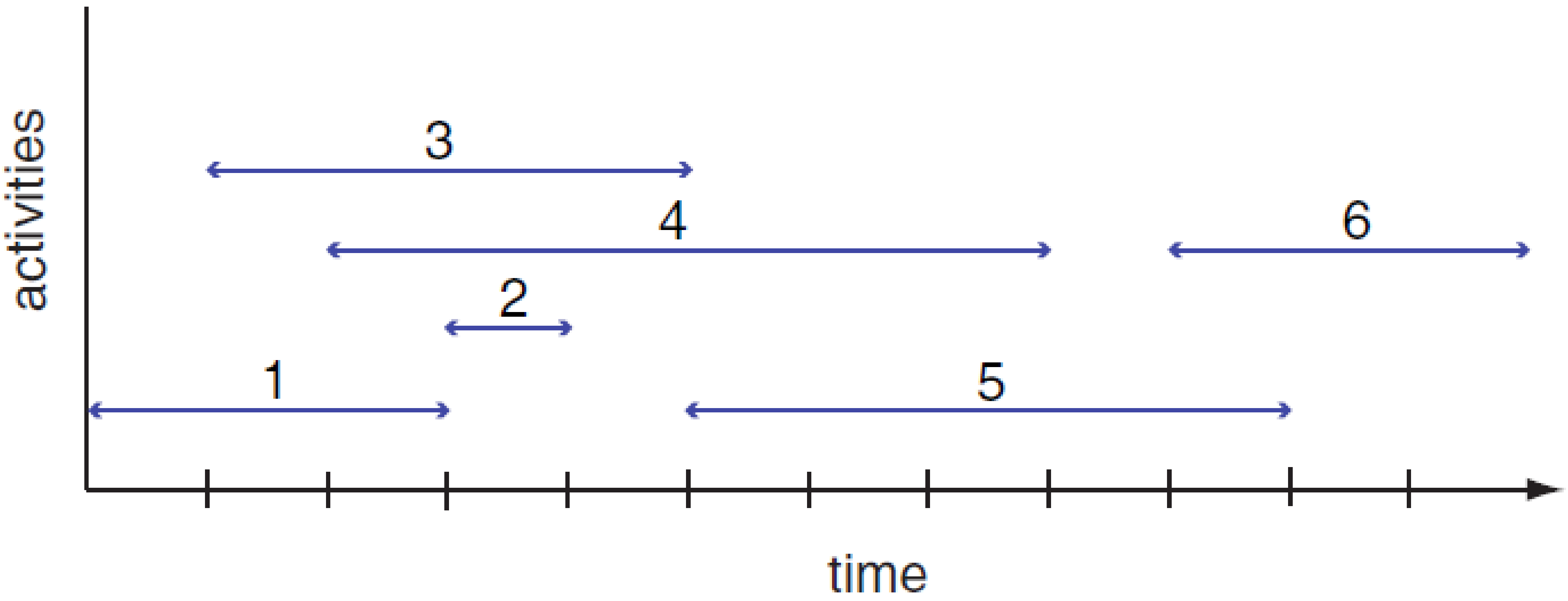
# Importance of Efficiency

- Computers have finite speed, finite memory.
- Need to minimize amount of computation that the machines needs to make.

Algorithm *X* executes in $10n^2 + n / 3.7 + 22$ steps
Algorithm *Y* executes in $100n \log n + 14n + 22$ steps

Which algorithm is better?

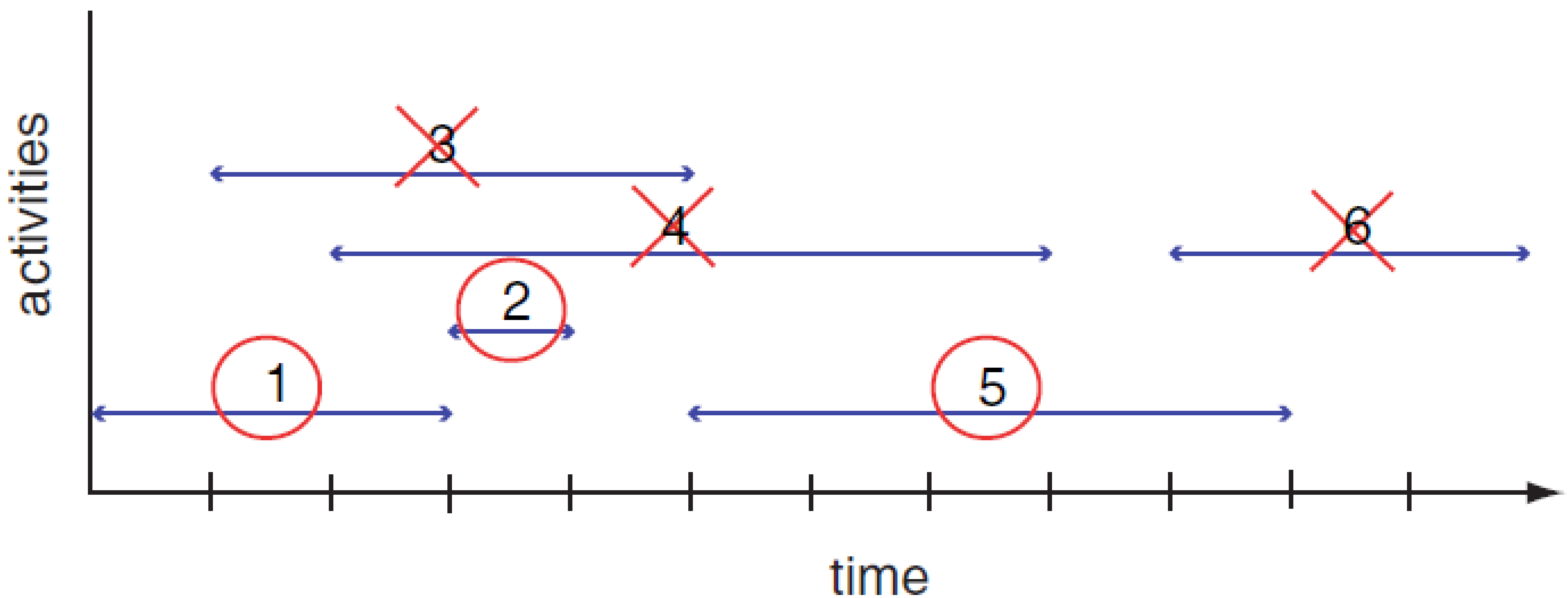
# Greedy Algorithm




**Activity Selection Problem**

- Given: A set of proposed activities that wish to use a resource, which can only be used by one activity at a time
- Problem: Find maximum-size set of activities that do not have a time conflict

# Greedy Algorithm




- Greedy choice: Pick the remaining compatible activity that has the earliest finish time

- Can you write an algorithm that does this?

# Algorithm for Activity Selection Problem

ActivitySelector($s[n], f[n]$)

$A \leftarrow 1$;   $j \leftarrow 1$

<u>for</u> $i \leftarrow 2$ <u>to</u> $n$ <u>do</u>

  <u>if</u> $s_i \geq f_j$ <u>then</u>   $\triangleright$ greedy choice

    $A \leftarrow A \cup \{i\}$

    $j \leftarrow i$

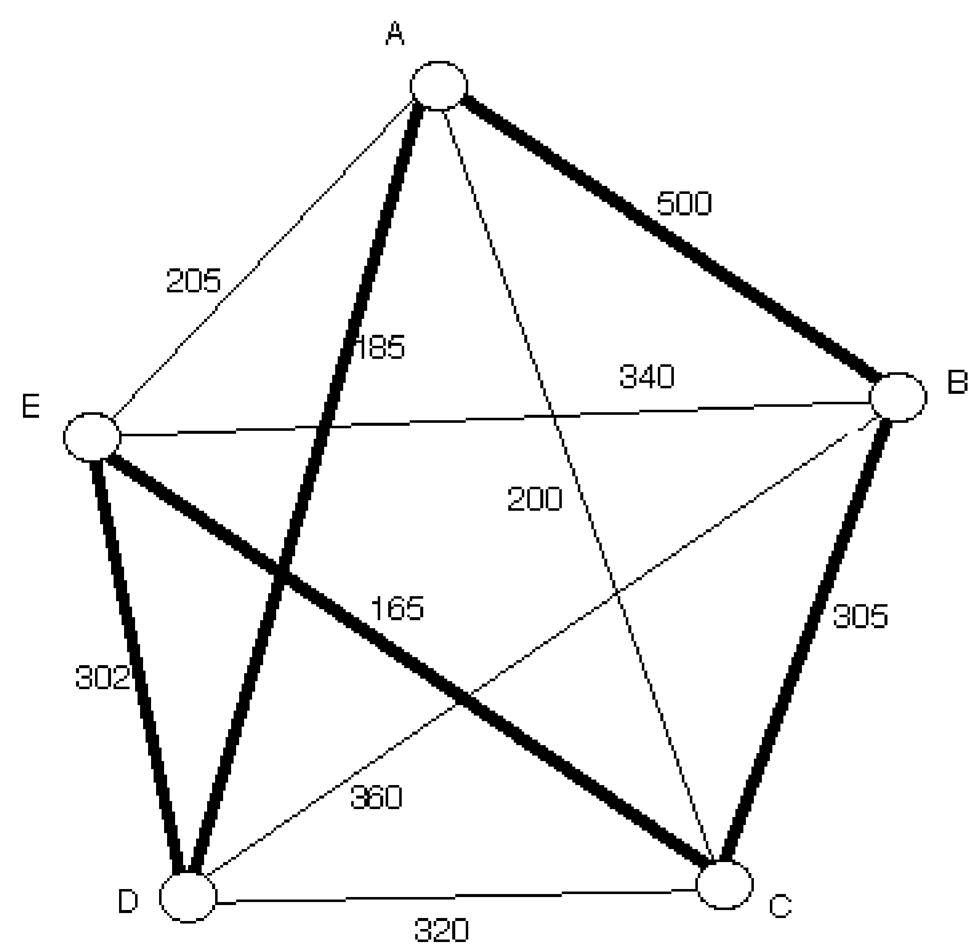
<u>return</u> $A$

- What is the running time for this algorithm?

# Nearest Neighbor Algorithm

1. Start at a vertex
2. Find the lightest edge connecting the current vertex and an unvisited vertex V
3. Set current vertex to V
4. Mark V as visited
5. Terminate when all the vertices have been visited
6. Repeat from step 2

- Example: Starts from Vertex A and returns
- to starting point at the end
  - Path: A - D - E - C - B - A



# Deficiencies of Nearest Neighbor

- Worst case: algorithm can result in a path much longer than the optimal path
- Won't work in case of incomplete graph
    - When some cities are not connected

- Any ideas for a better algorithm?