Algorithms

Saptadeep Pal
saptadeep@ucla.edu

This is a modified version of slides by Dr. John Lee,
Dr. Rani Ghaida,
Dr. Abde Ali Kagalwalla and Dr. Yasmine Badr
A lot of the material is based on
Introduction to Algorithms by Cormen et al.

Variability Expedition

UCLA

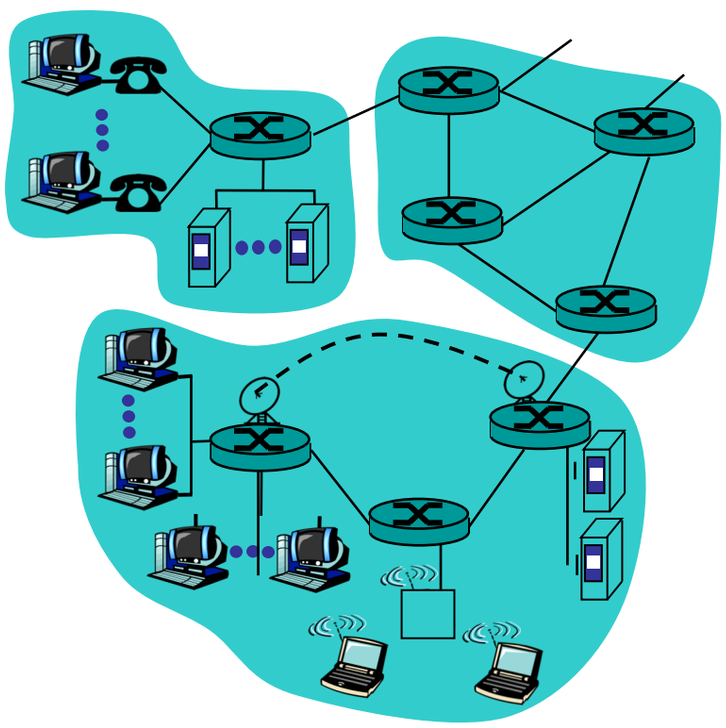
What is an Algorithm?

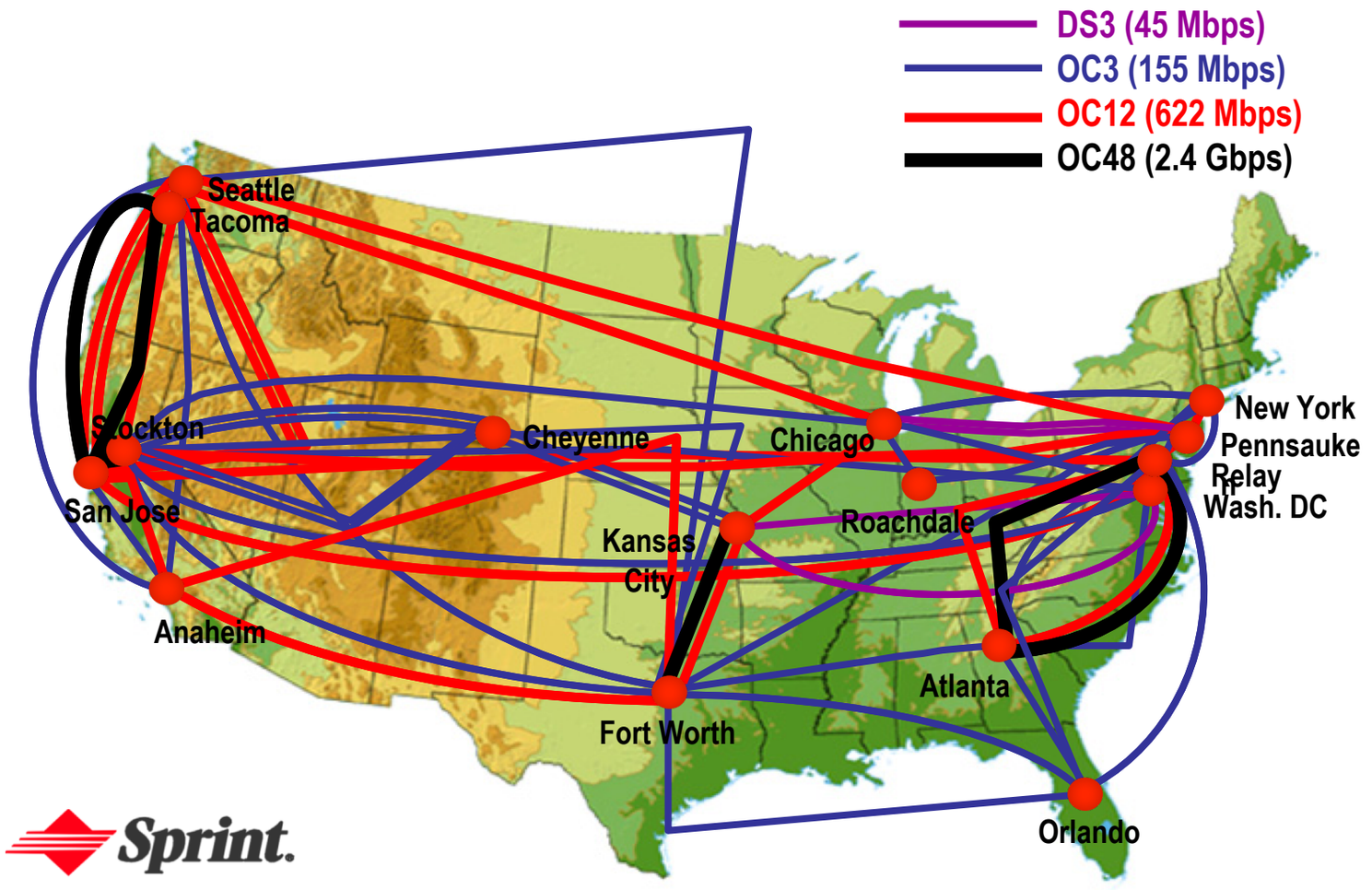
- An algorithm is a well-defined computational procedure that takes an input and produces some output.
  It is thus a set of computational steps that transform the input into the output.

Agenda

- ➢ What sort of problems are solved with algorithms?
- ➢ Examples of Algorithms
  - ○ Search Algorithms
  - ○ Gale Shapley Stable Marriage Algorithm
  - ○ Shortest-Path Problem
- ➢ Efficiency of Algorithms
- o Sorting Algorithms
  - ○ *Exercise*: Bubble Sort
- ➢ Algorithmic Techniques
  - ○ Greedy Algorithm
    - ▪ Example: Activity Selection Problem
- ➢ *Mini-project*: Traveling Salesman Problem
- ➢ Research Topics

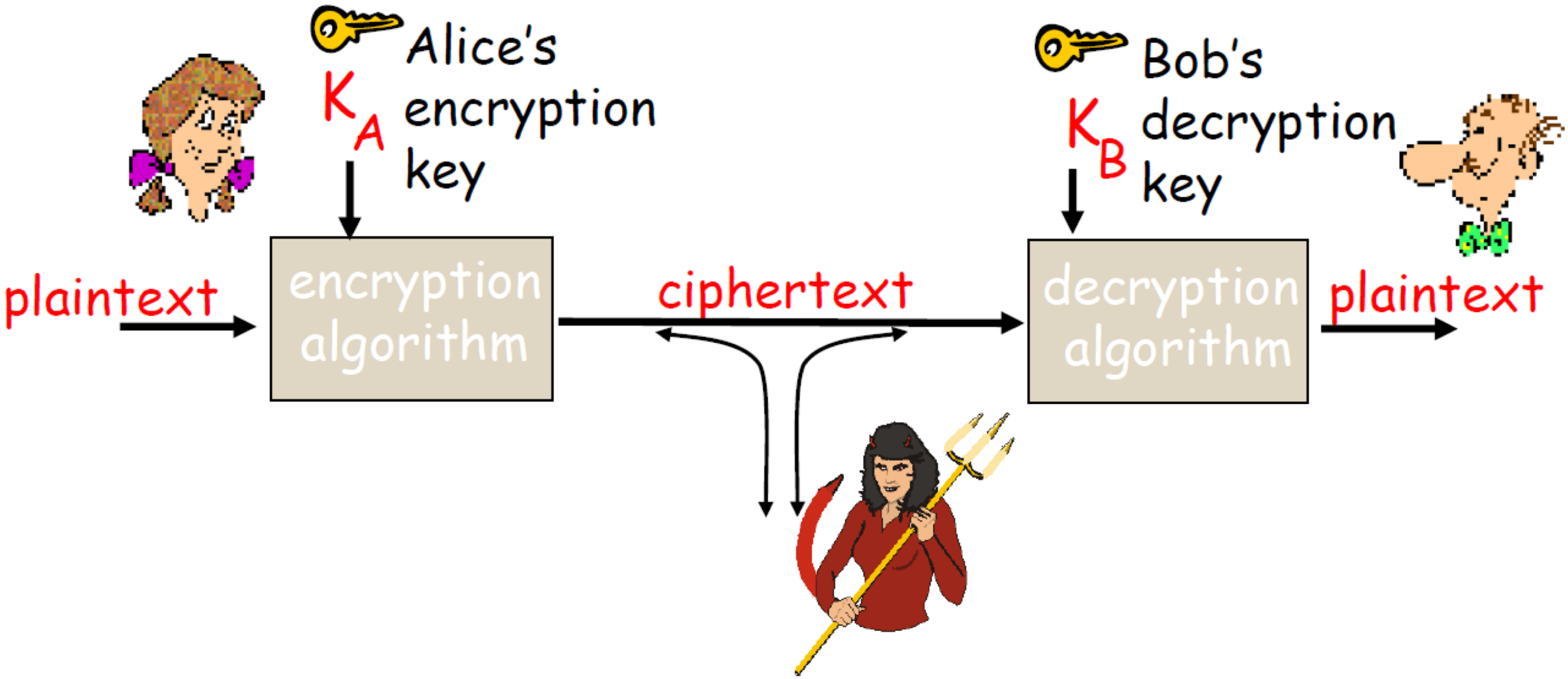
What sort of problems are solved with algorithms?





- **Internet - Manage and manipulate data**
  - E.g., (1) search engines, (2) finding good routes on which data will travel

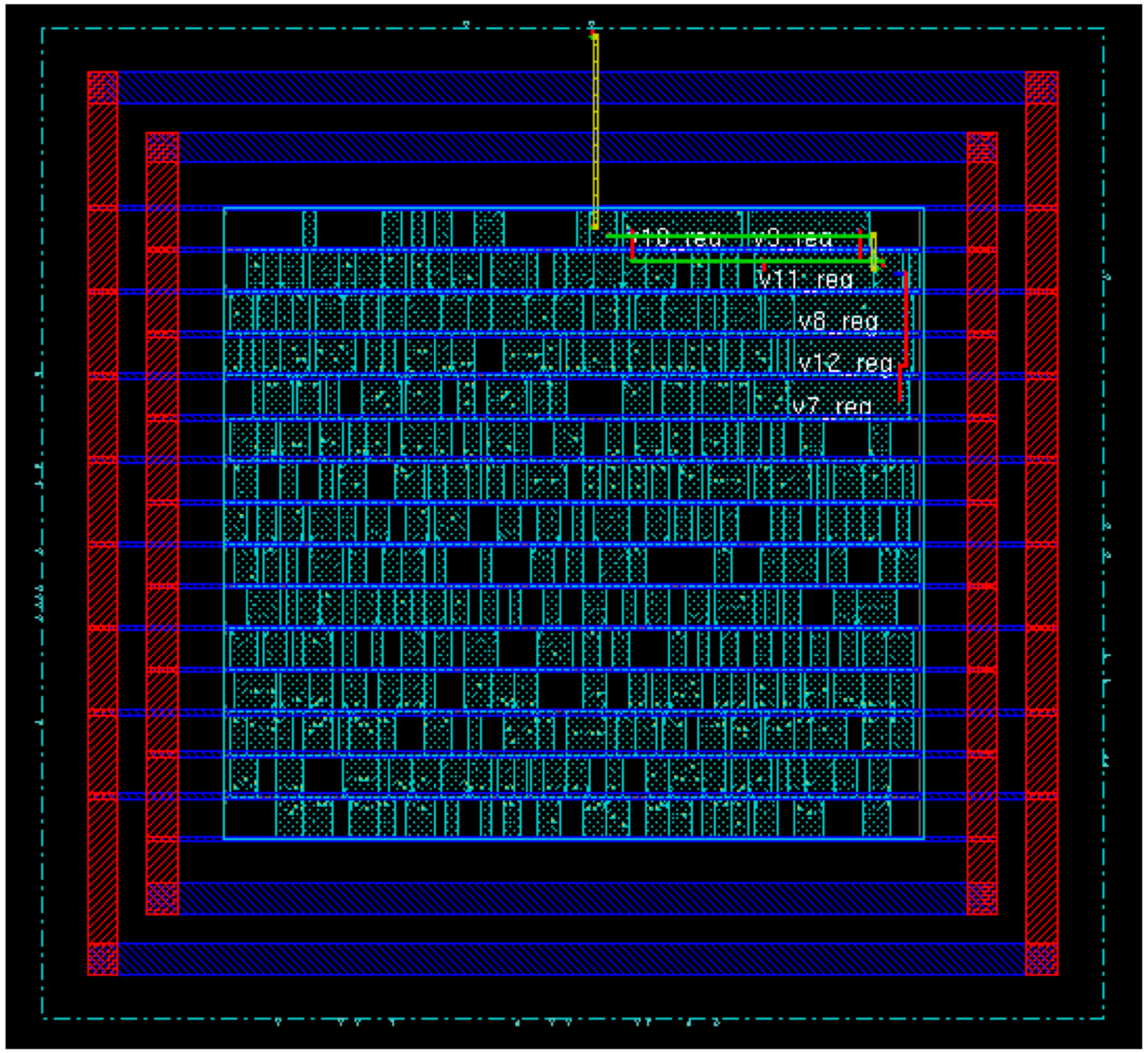
What sort of problems are solved with algorithms?

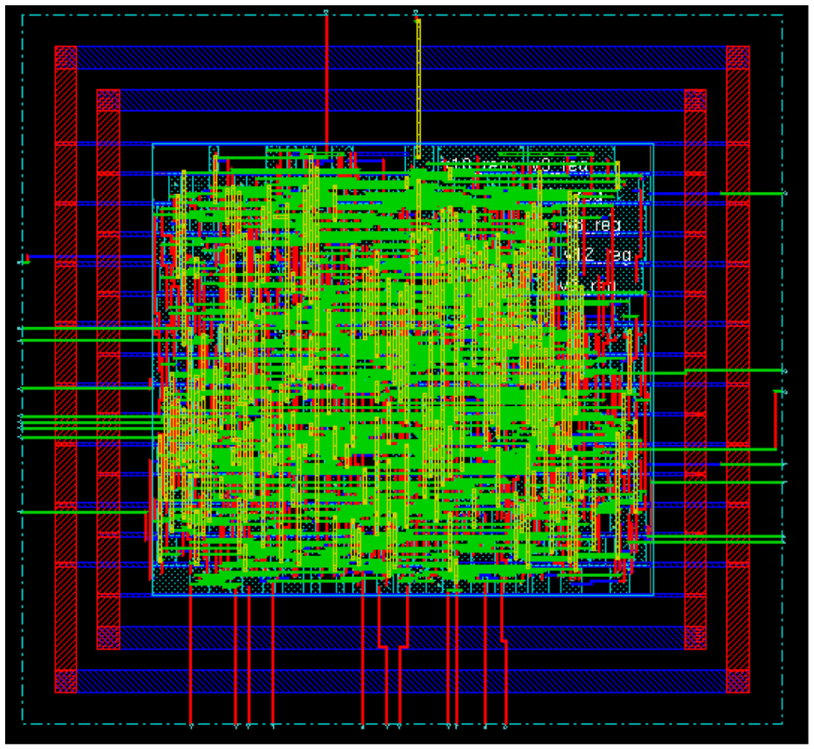


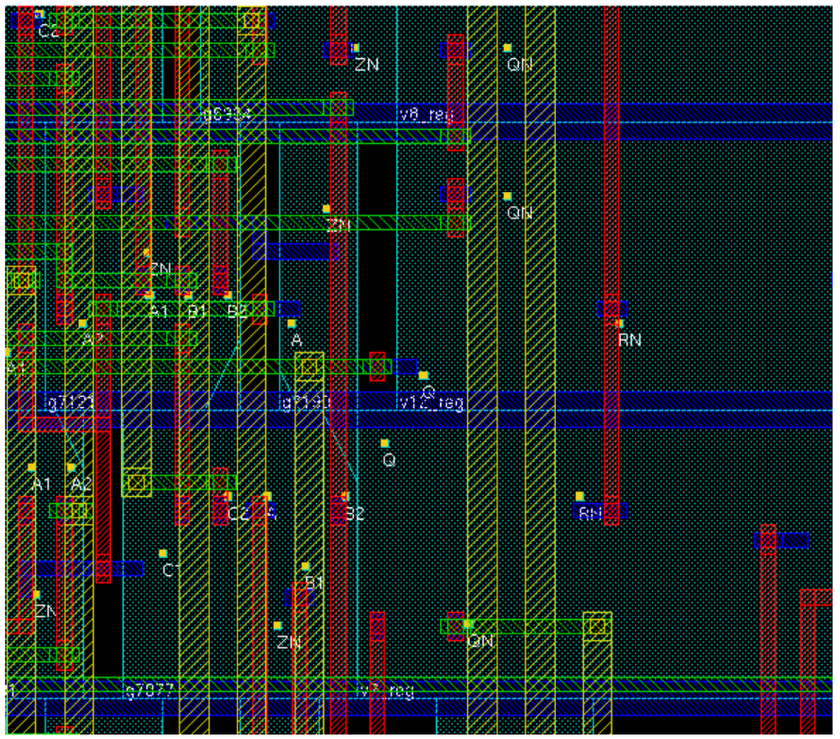
- **Data security – cryptography**
  - Encryption algorithm → computational steps to transform plaintext to ciphertext
  - Decryption algorithm → computational steps to transform ciphertext back to plaintext

What sort of problems are solved with algorithms?

- **NanoEletronics – Design Automation**
  - E.g., components placement and routing







What sort of problems are solved with algorithms?

- **Scheduling problems**
  - E.g., Flight scheduling
    - Crew preferences
    - Airplane types
    - Weather
    - Fuel Costs
    - Delays

From
http://www.worldofstock.com/slides/TRA2599.jpg

What sort of problems are solved with algorithms?

- **Artificial Intelligence**
  - E.g., IBM Watson supercomputer beats all-time best Jeopardy! players on Feb 2011

Daily Double betting

Final Jeopardy betting

Clue selection

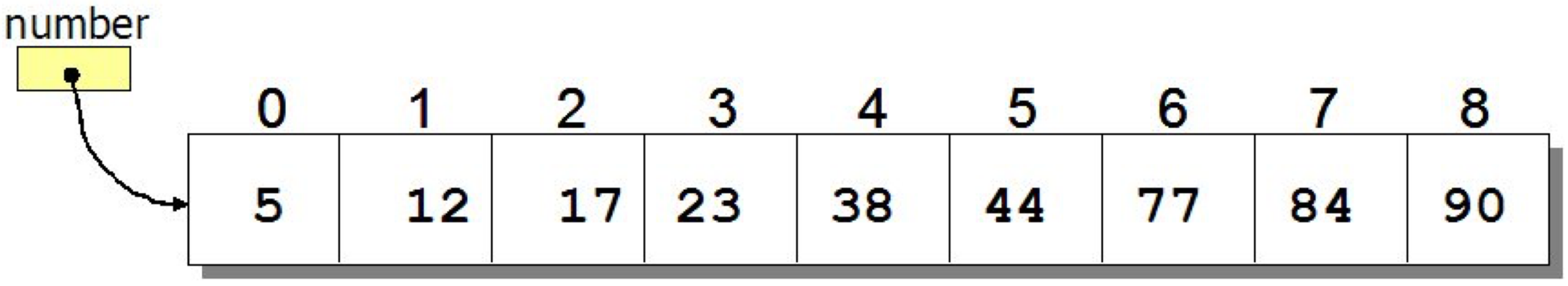
Confidence Threshold for Buzz-in

Search Algorithms

- How to search for an element in a list?
  1. **Sequential search**: Go through each element and check if it is the required element
     - Best case?
     - Worst case?
  2. What if the list is sorted?

Can we do better?

Source: http://www.spinachandyoga.com/how-to-deal-with-a-long-to-do-list-with-zen/



Source: http://www.csit.parkland.edu/~mbrandyberry/CS1Java/Lessons/Lesson27/BinarySearch.htm

Search Algorithms

- How to search for an element **x** in a list?
  1. **Sequential search**: Go through each element and check if it is the the required element
     - Best case?
     - Worst case?
  2. What if the list is sorted?

     Can we do better?

     YES

     BINARY SEARCH

Source: http://www.spinachandyoga.com/how-to-deal-with-a-long-to-do-list-with-zen/

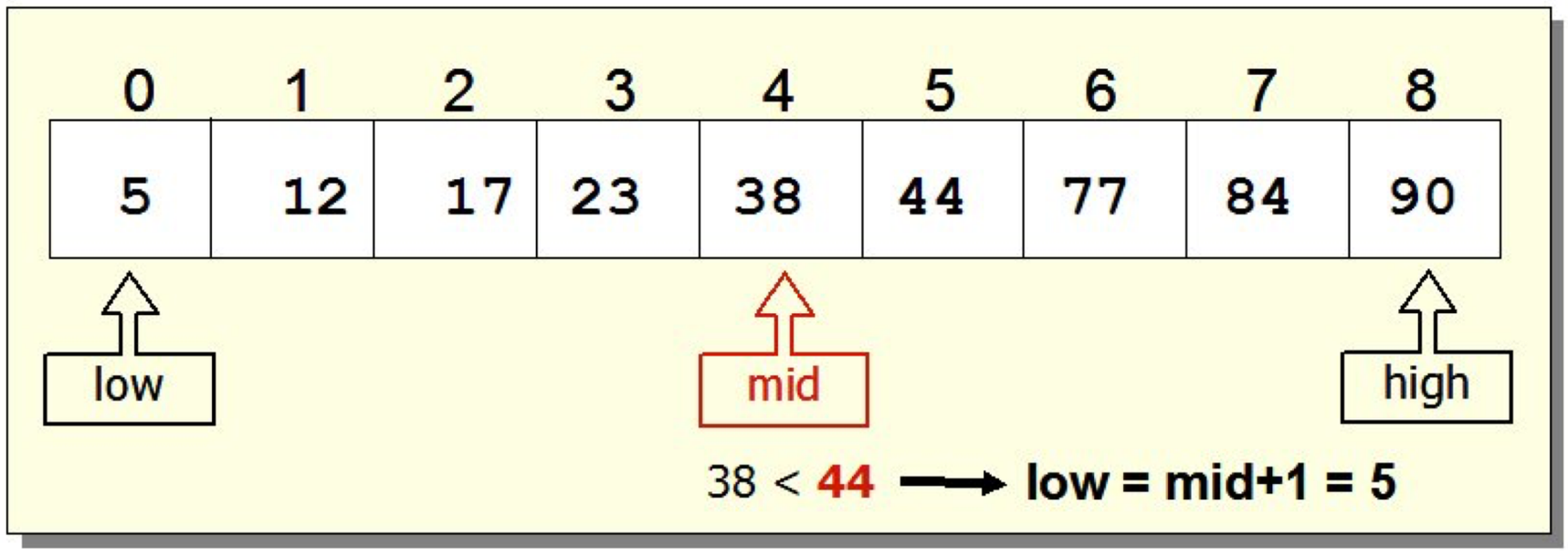
Search Algorithms: Binary Search

- If list is **sorted**:
  - Split list into two halves. If x is greater than element in the middle then discard the left half. Otherwise discard the right half. Repeat until you find x.

	low	high	mid
#1	0	8	4

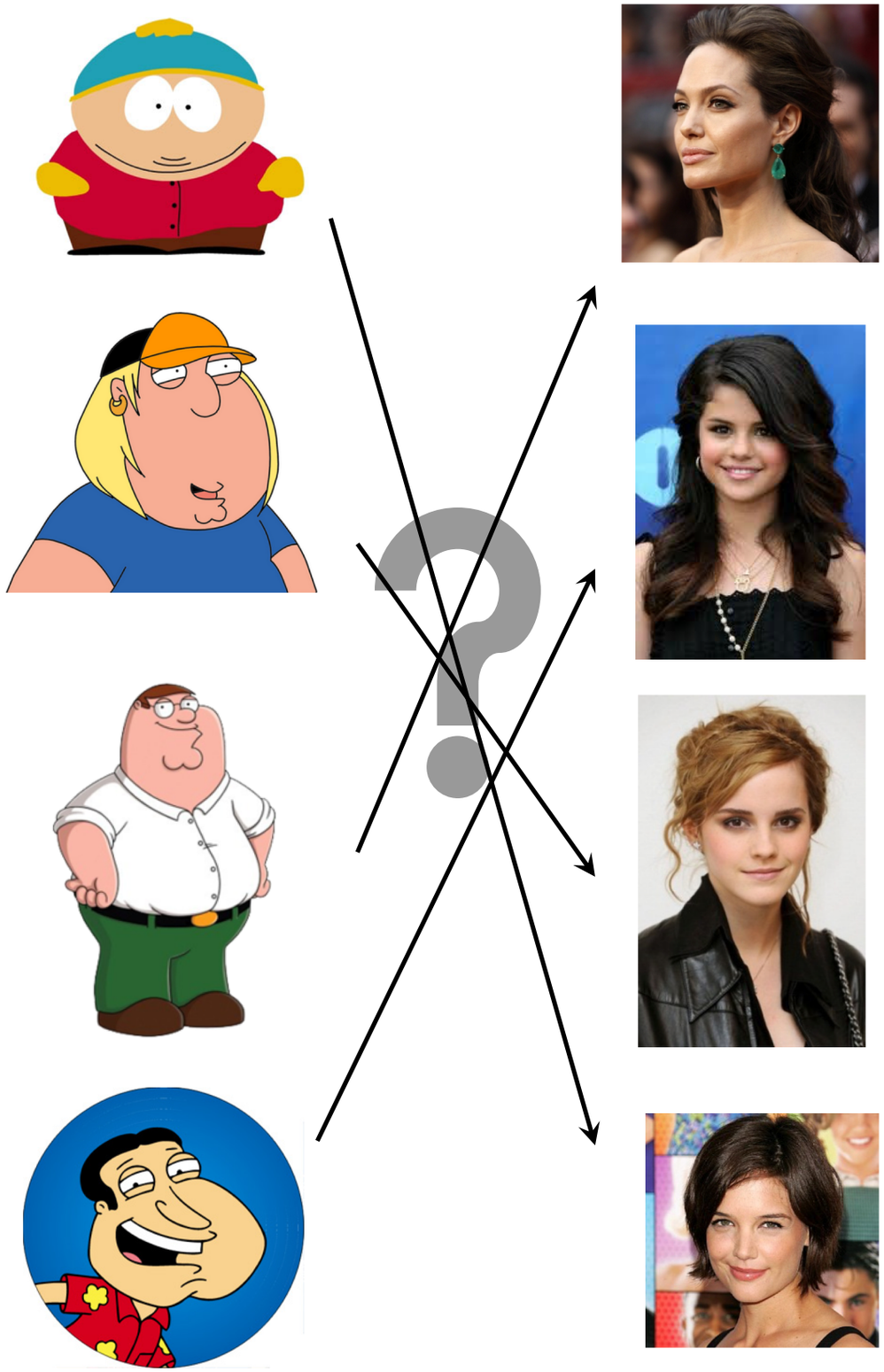
search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



Source: http://www.csit.parkland.edu/~mbrandyberry/CS1Java/Lessons/Lesson27/BinarySearch.htm

Example: Stable Marriage Problem



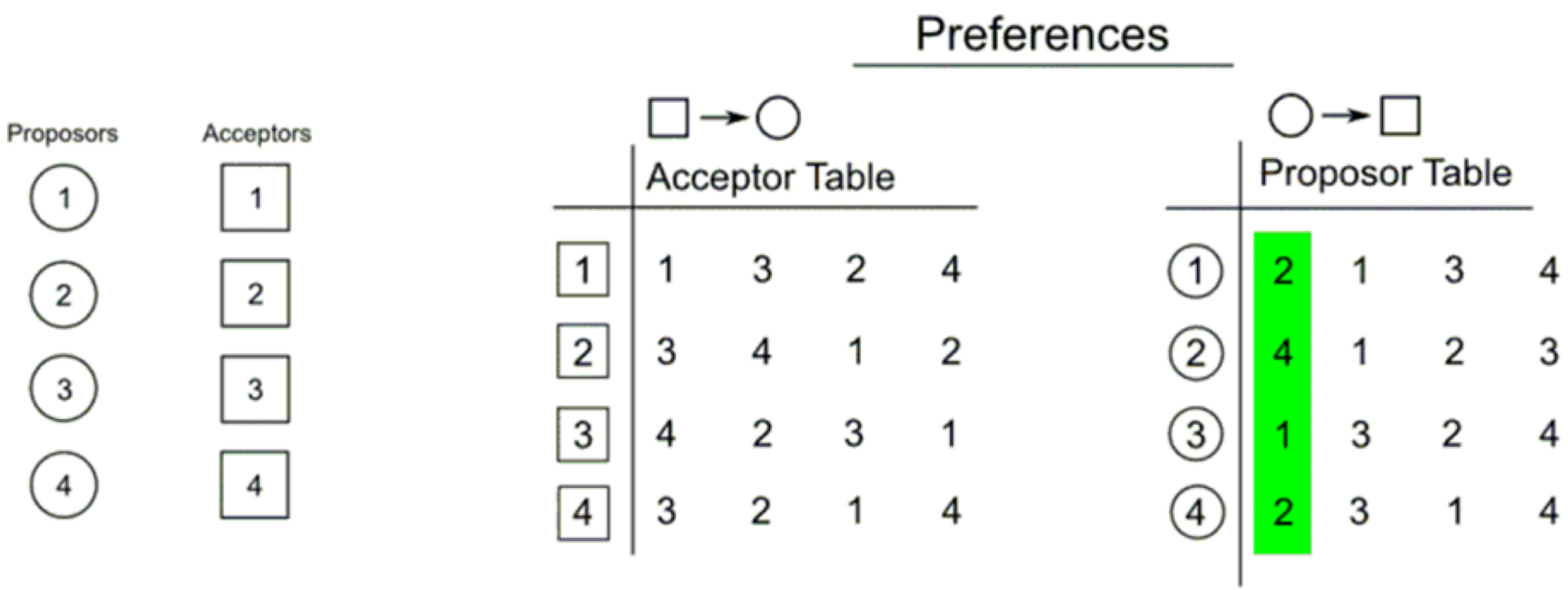
Input:
Men, Women, Preferences

Algorithm

Output:
1- All married
2- Stable marriages (i.e., no one prefers someone else who also prefers them

Example: Stable Marriage Problem

- Algorithm to solve this?



1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

1. In each round, unengaged man "proposes" to the most-preferred woman to whom he has not yet proposed
2. Each woman then considers all her suitors and tells the one she most prefers "Maybe" and all the rest of them "No".
3. She is then provisionally "engaged" to the suitor she most prefers so far, and that suitor is likewise provisionally engaged to her

Pseudocode

```
function stableMatching {
    Initialize all m ∈ M and w ∈ W to free
    while ∃ free man m who still has a woman w to propose to {
       w = m's highest ranked such woman to whom he has not yet proposed
       if w is free
          (m, w) become engaged
       else some pair (m', w) already exists
         if w prefers m to m'
            (m, w) become engaged
           m' becomes free
         else
            (m', w) remain engaged
    }
}
```

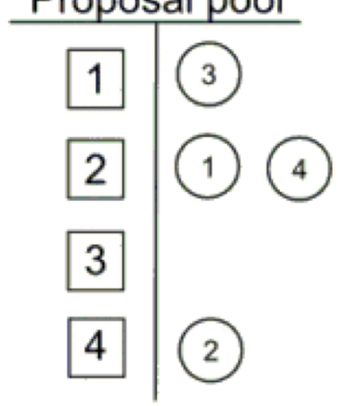
Gale Shapley Stable Marriage Algorithm

Round : 1

Proposors

Acceptors

Proposal pool



- 1-4 propose, as none are currently tentatively attached

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

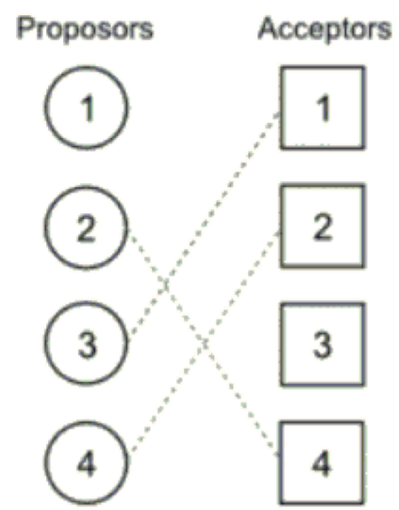
○ → □

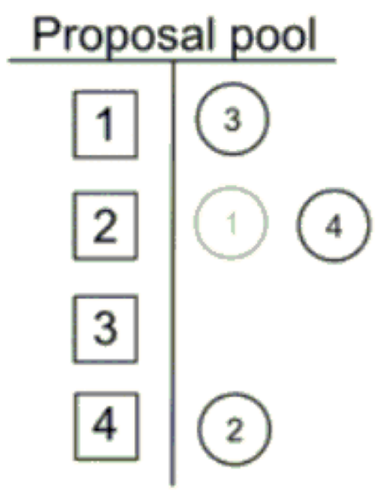
Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 1





- 1 accepts 3's proposal—no better offer.
- 2 accepts 4's proposal as 4 is more prefereable to 1.
- 3 recieves no offer.
- 4 accepts 2's proposal—no better offer.

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

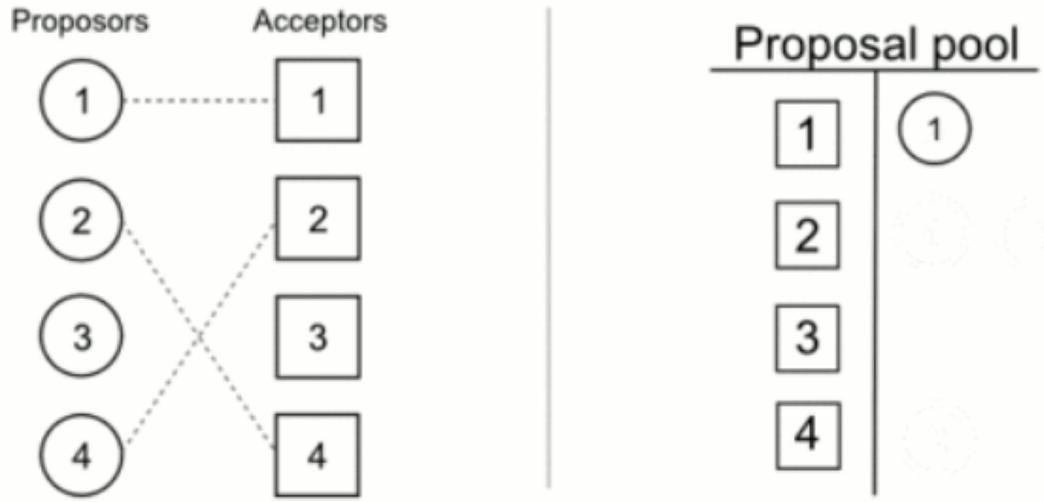
○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 2



• 1 drops 3's propsal in favour of 1 as this is higher in its preference table. 3 returns to the proposal pool.

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

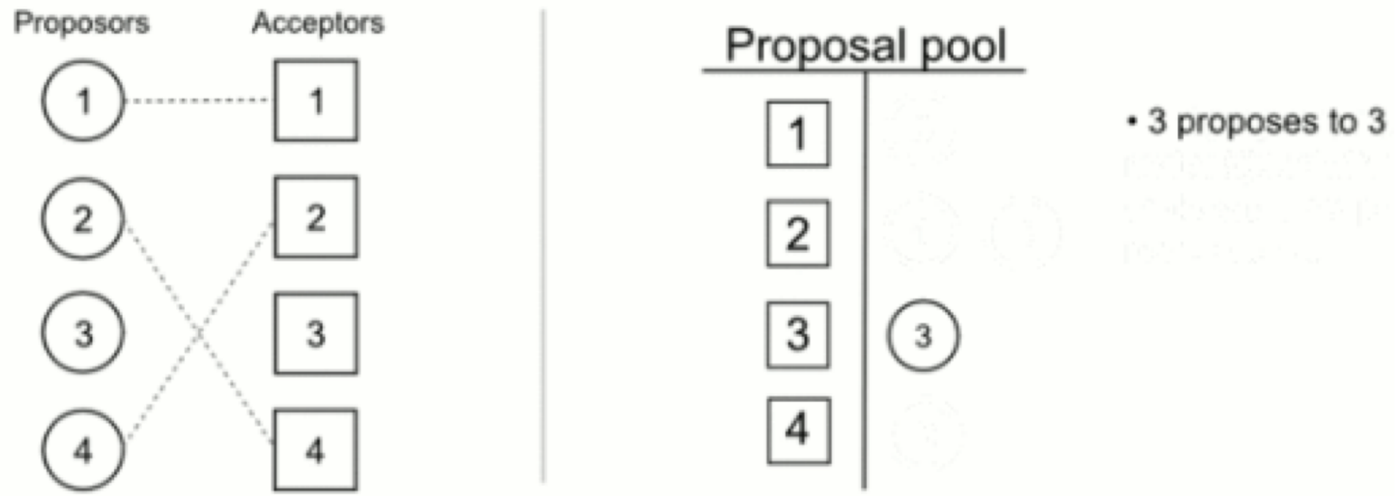
○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 3



Preferences

□ → ○ Acceptor Table

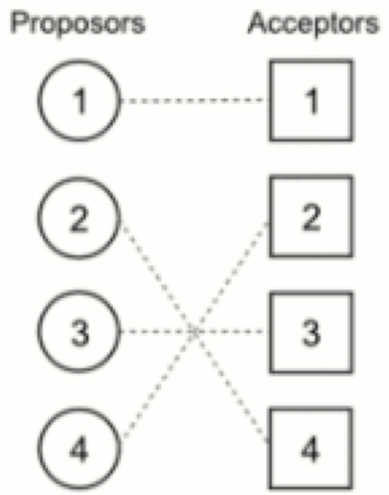
1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

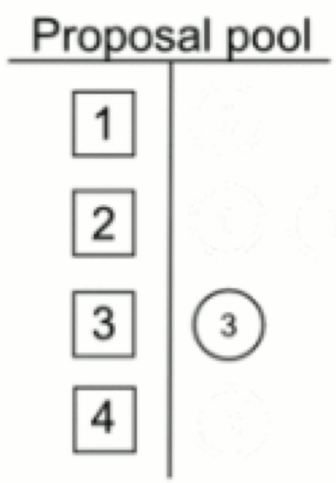
○ → □ Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 3





- 3 accepts 3, not having a better offer

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

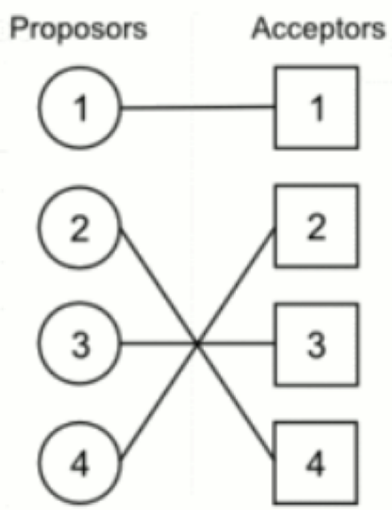
○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Finish



- No two members {P,A} would prefer one-another over their current pairing

Preferences

□ → ○

Acceptor Table

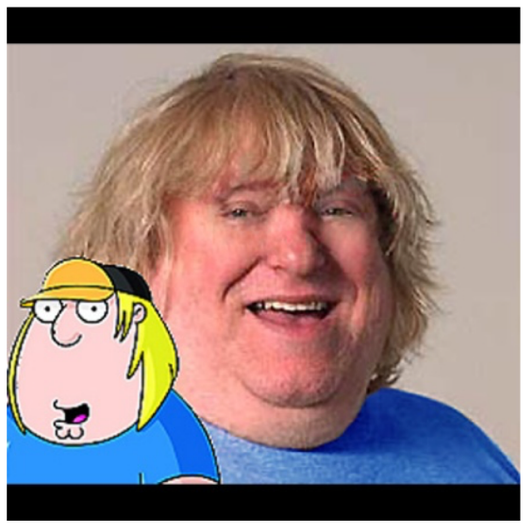
1	**1**	3	2	4
2	3	**4**	1	2
3	4	2	**3**	1
4	3	**2**	1	4

○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Our Happy Couples!



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

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

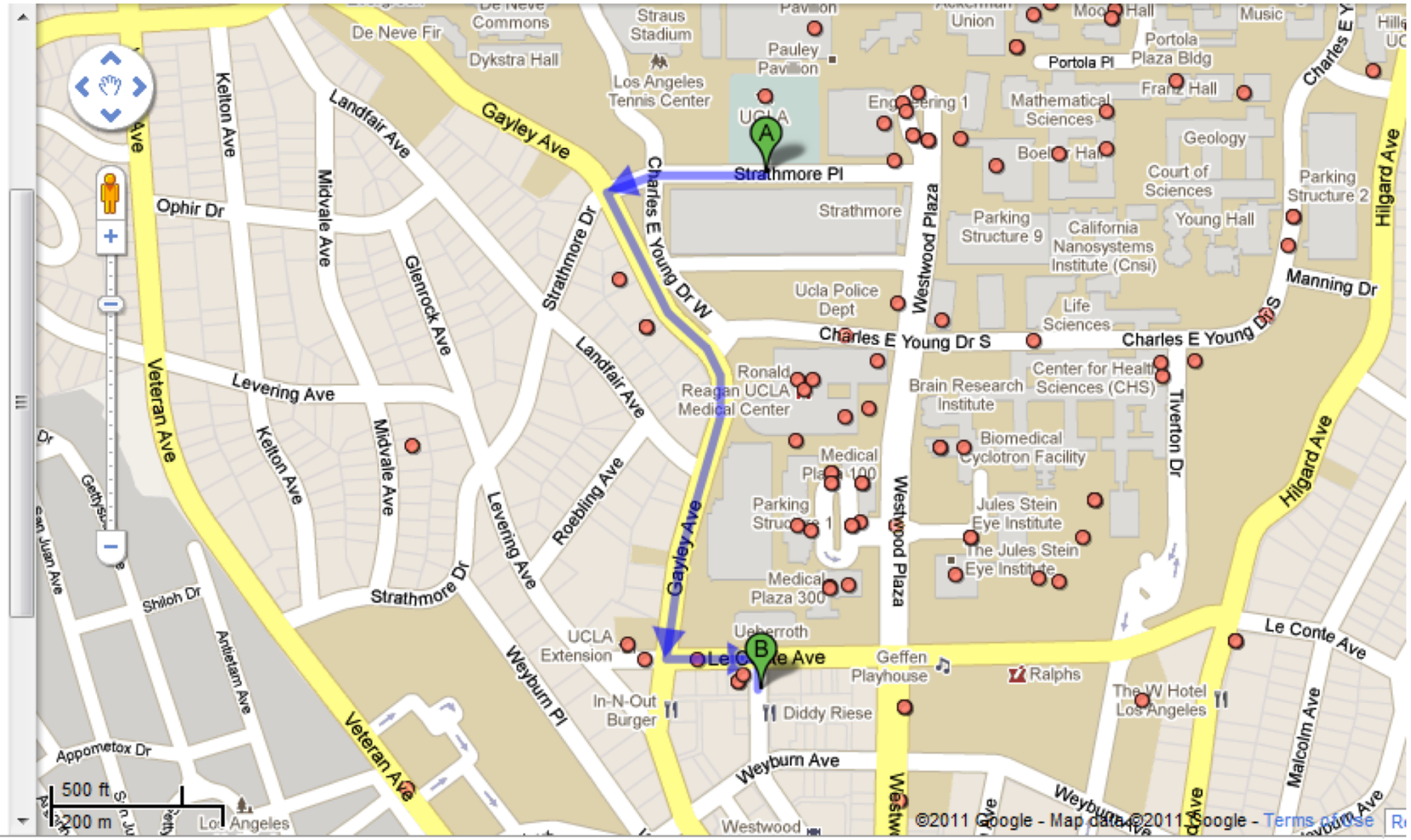
3. Turn left at **Le Conte Ave**

354 ft

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

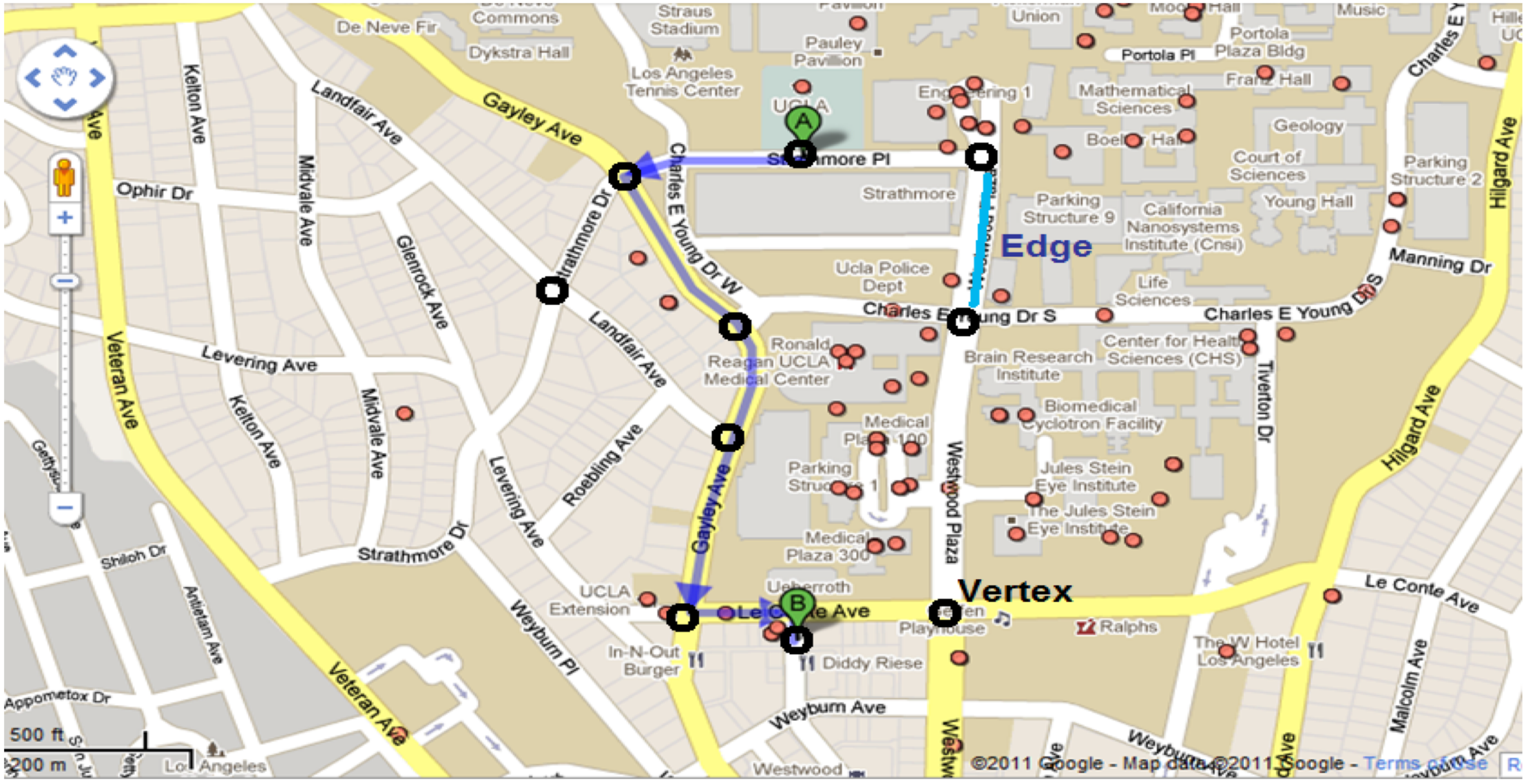
115 ft

Diddy Riese
926 Broxton Avenue



Shortest-Path Problem

- Google Maps - determining the fastest route

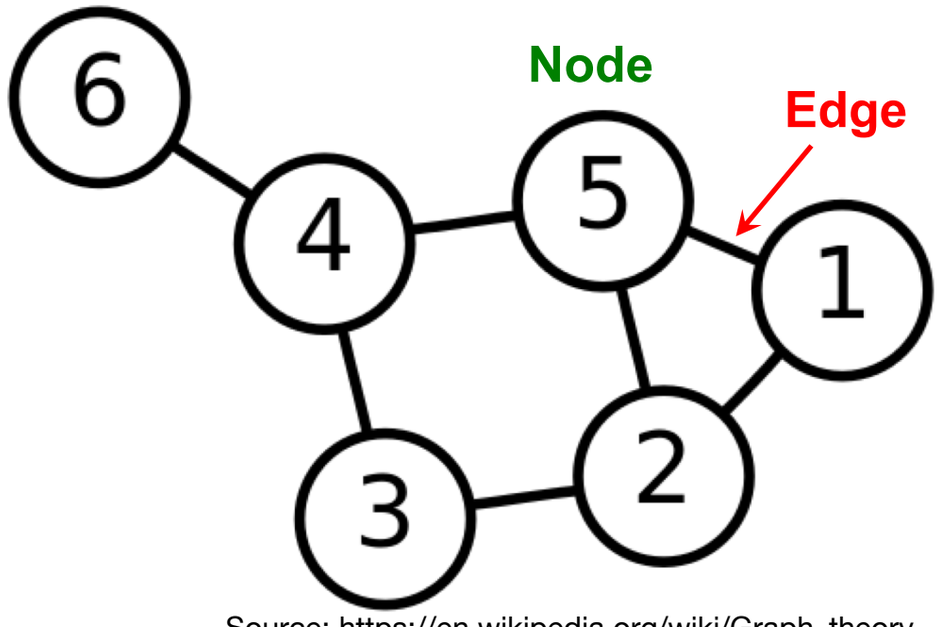


Graph [In Brief]

- Set of :
  - **Nodes/vertices**
  - **edges** connecting the nodes

- Edges can be
  - Undirected: no distinction between the 2 nodes
  - Directed: has a source and a destination

Undirected graph

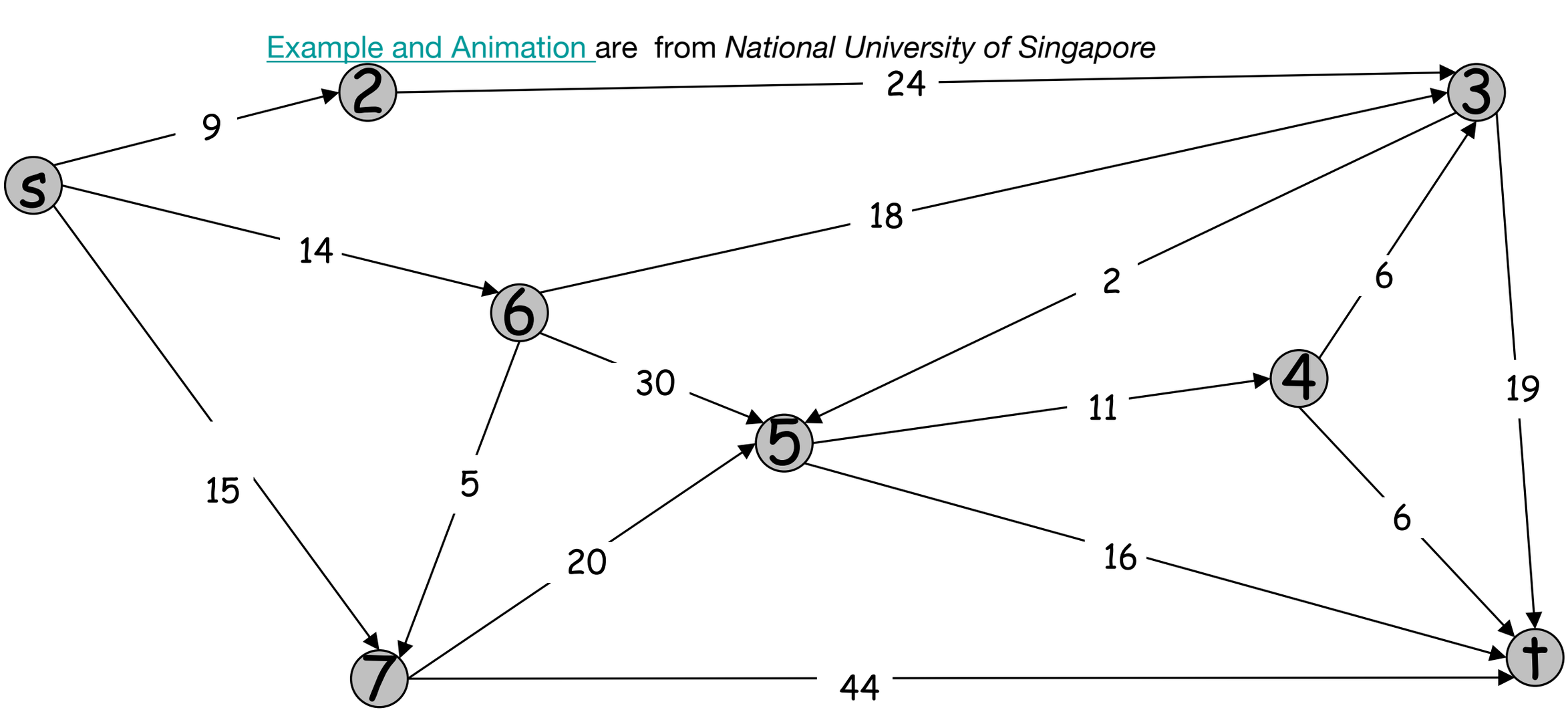


Source: https://en.wikipedia.org/wiki/Graph_theory

Class Activity: Shortest Path Algorithm

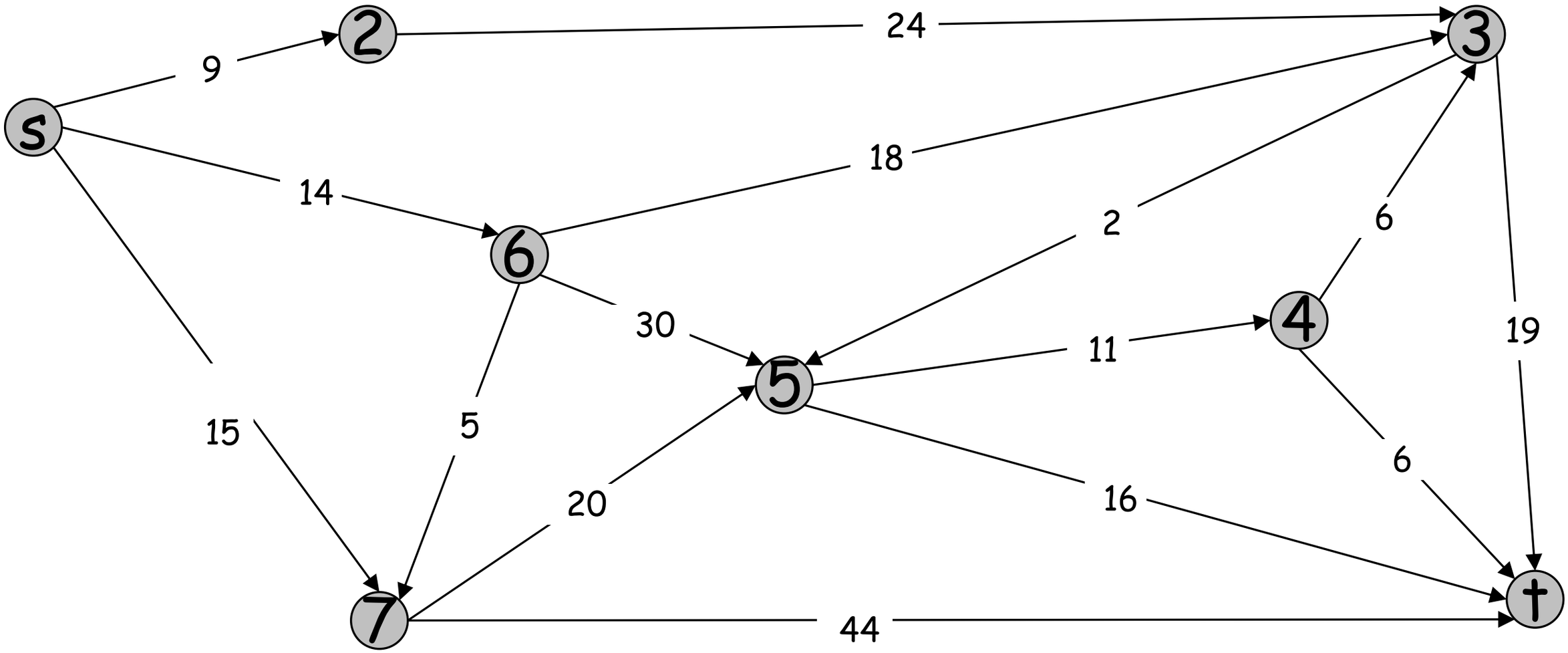
- Can you come up with an algorithm that solves the shortest path problem?
- Work in groups
- What is the shortest path from **s** to **t**?

Example and Animation are from *National University of Singapore*



Dijsktra's Shortest Path Algorithm

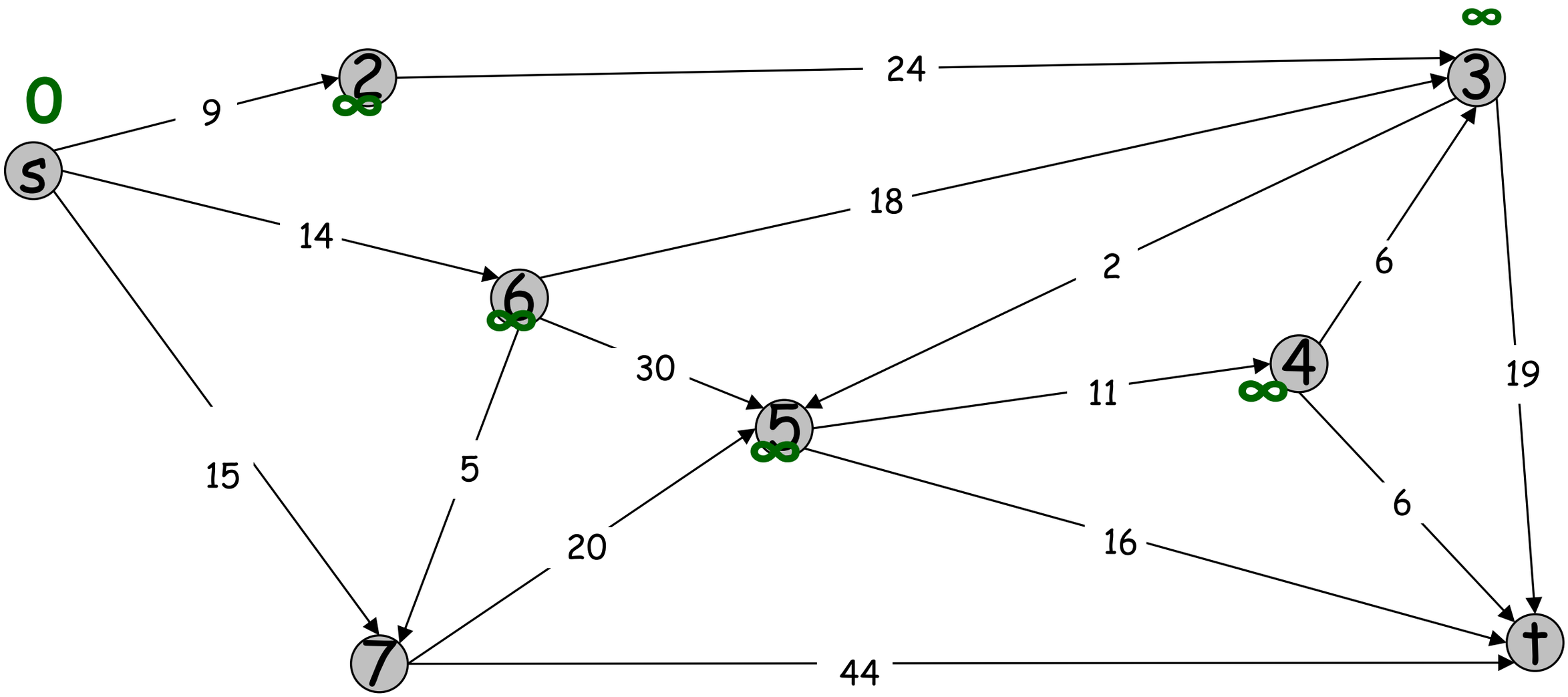
Find shortest path from s to t.



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 = { }

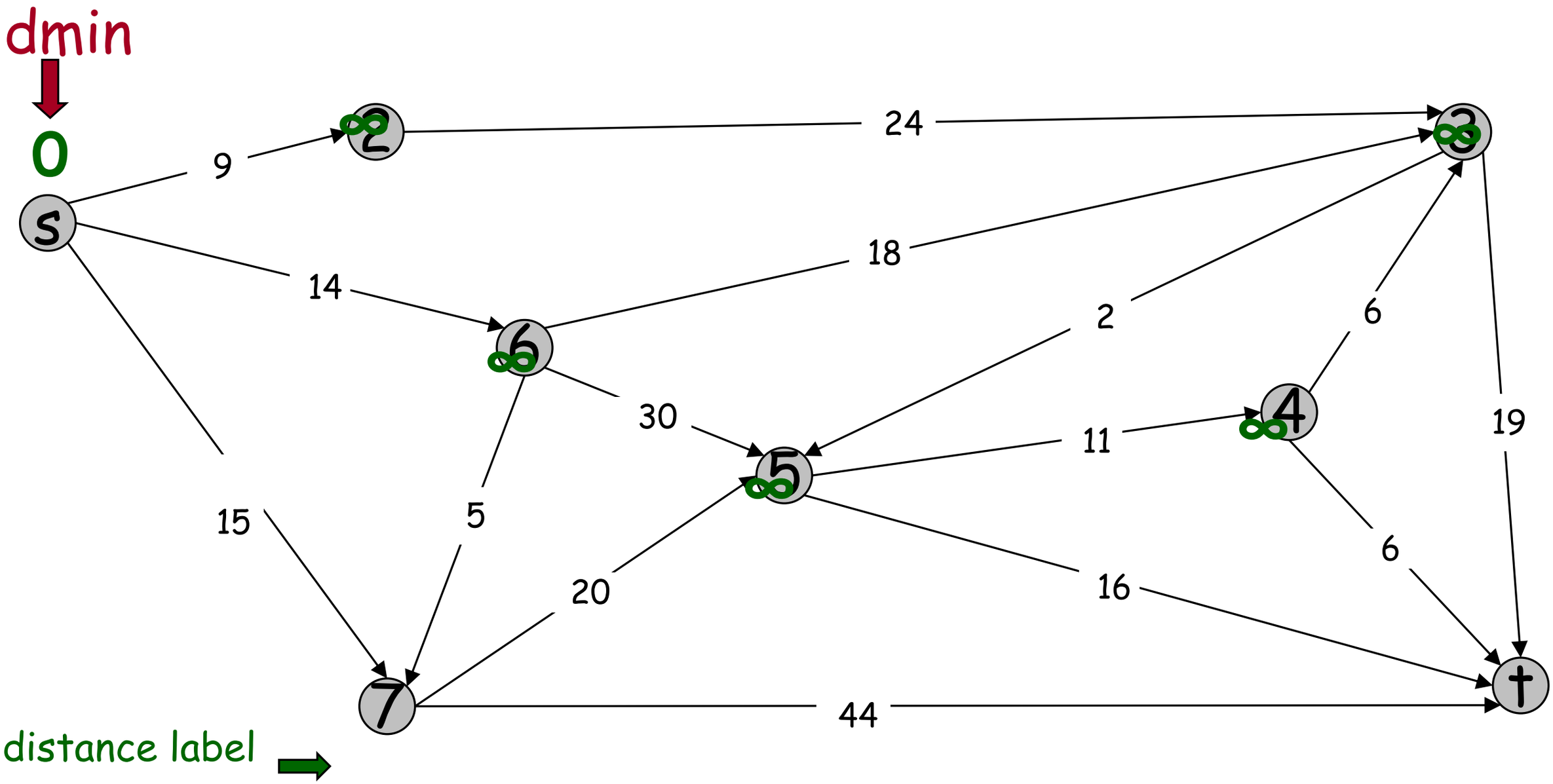


distance label ➡ ∞

Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

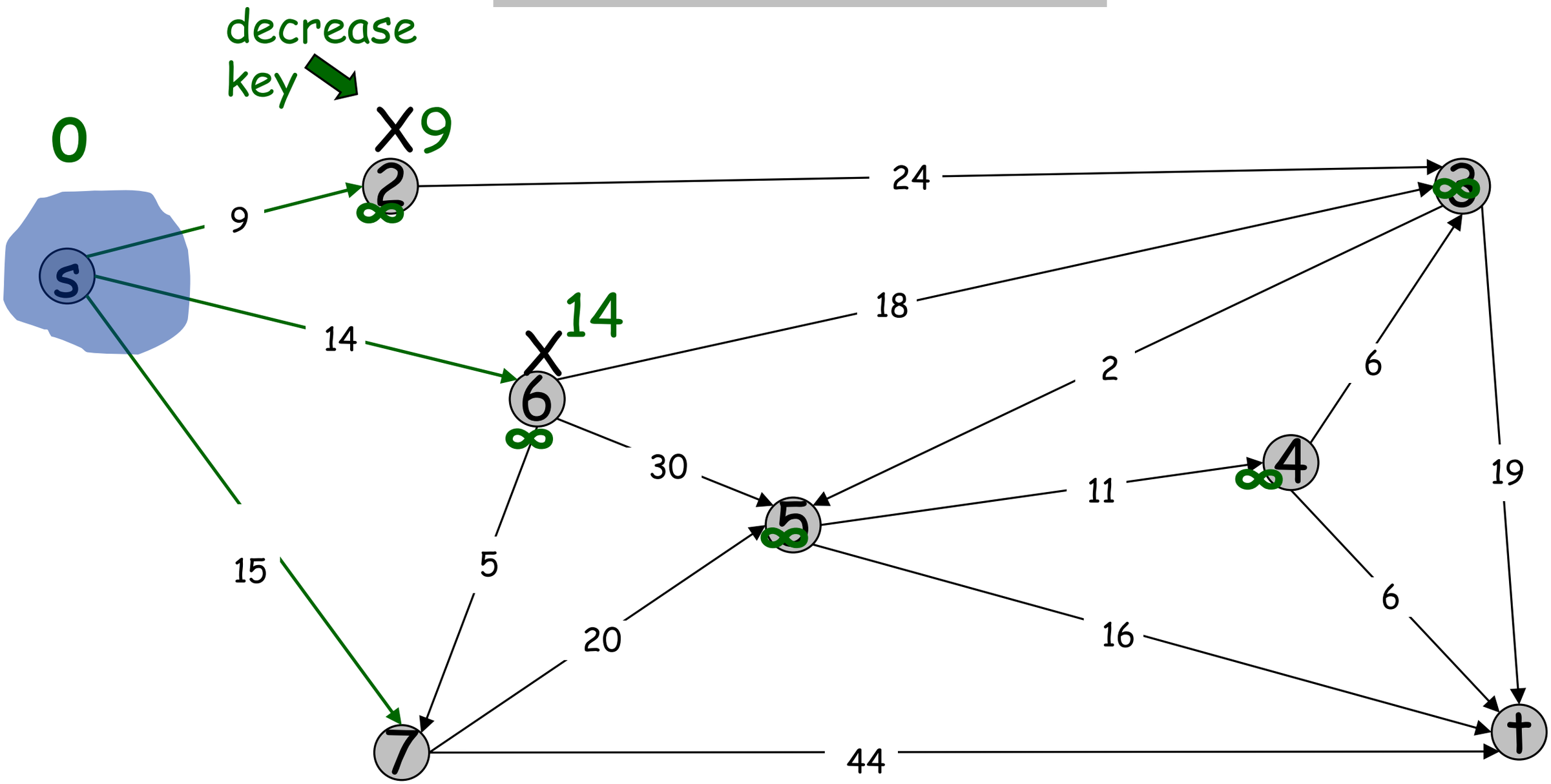
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }
B = { }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

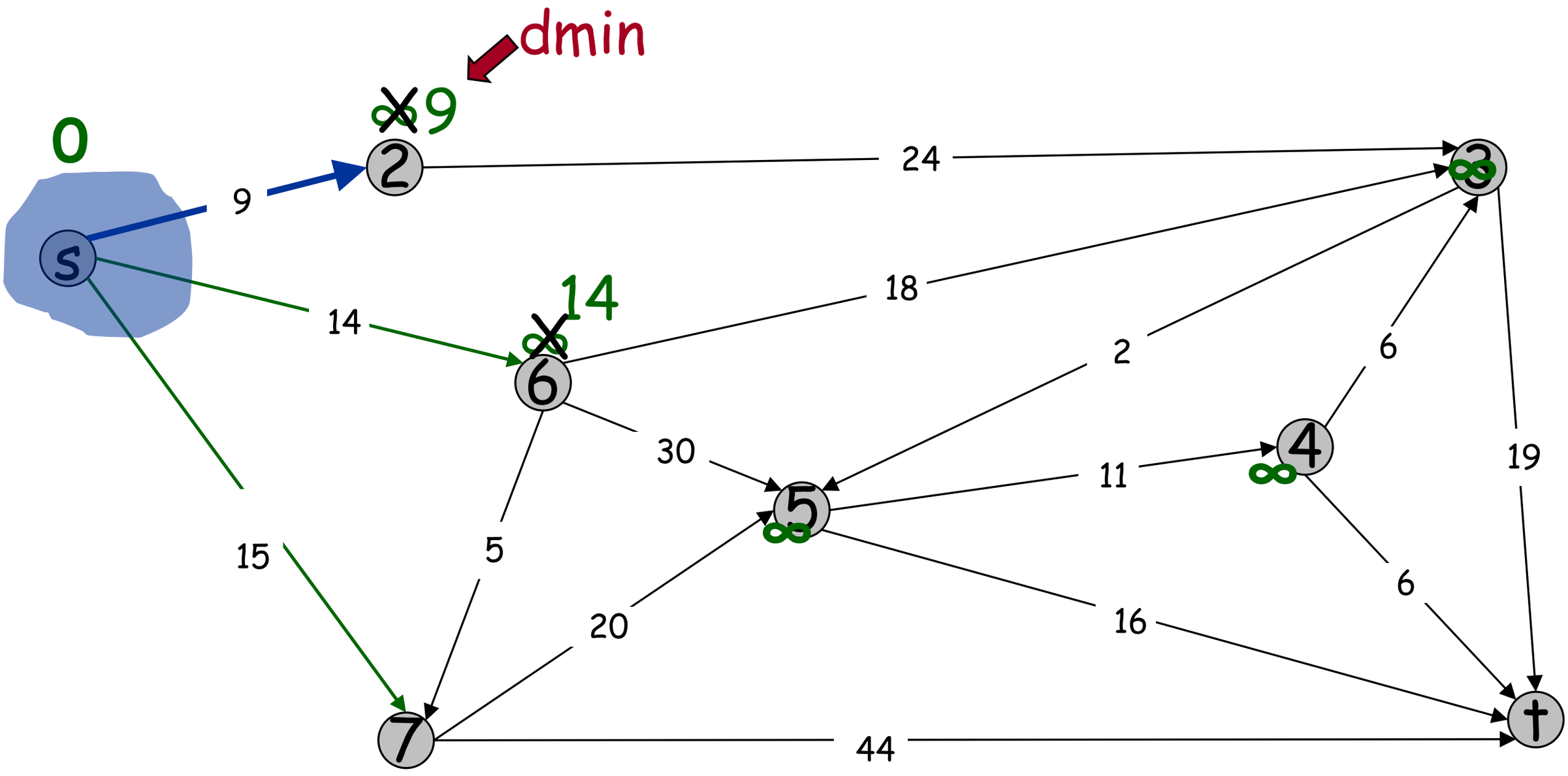
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }
B = { s }



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 }

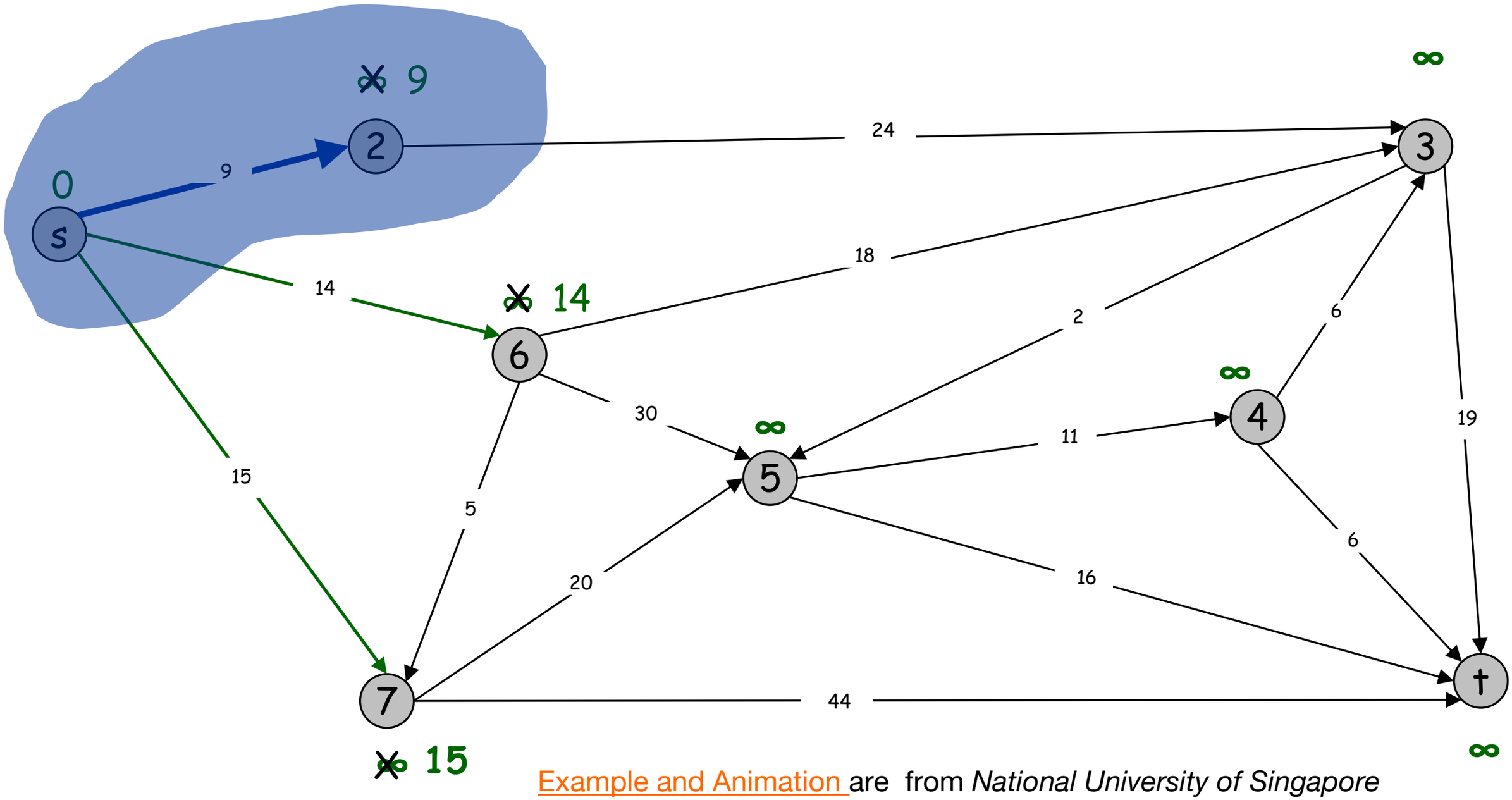


distance label ➡ ∞ 15

Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

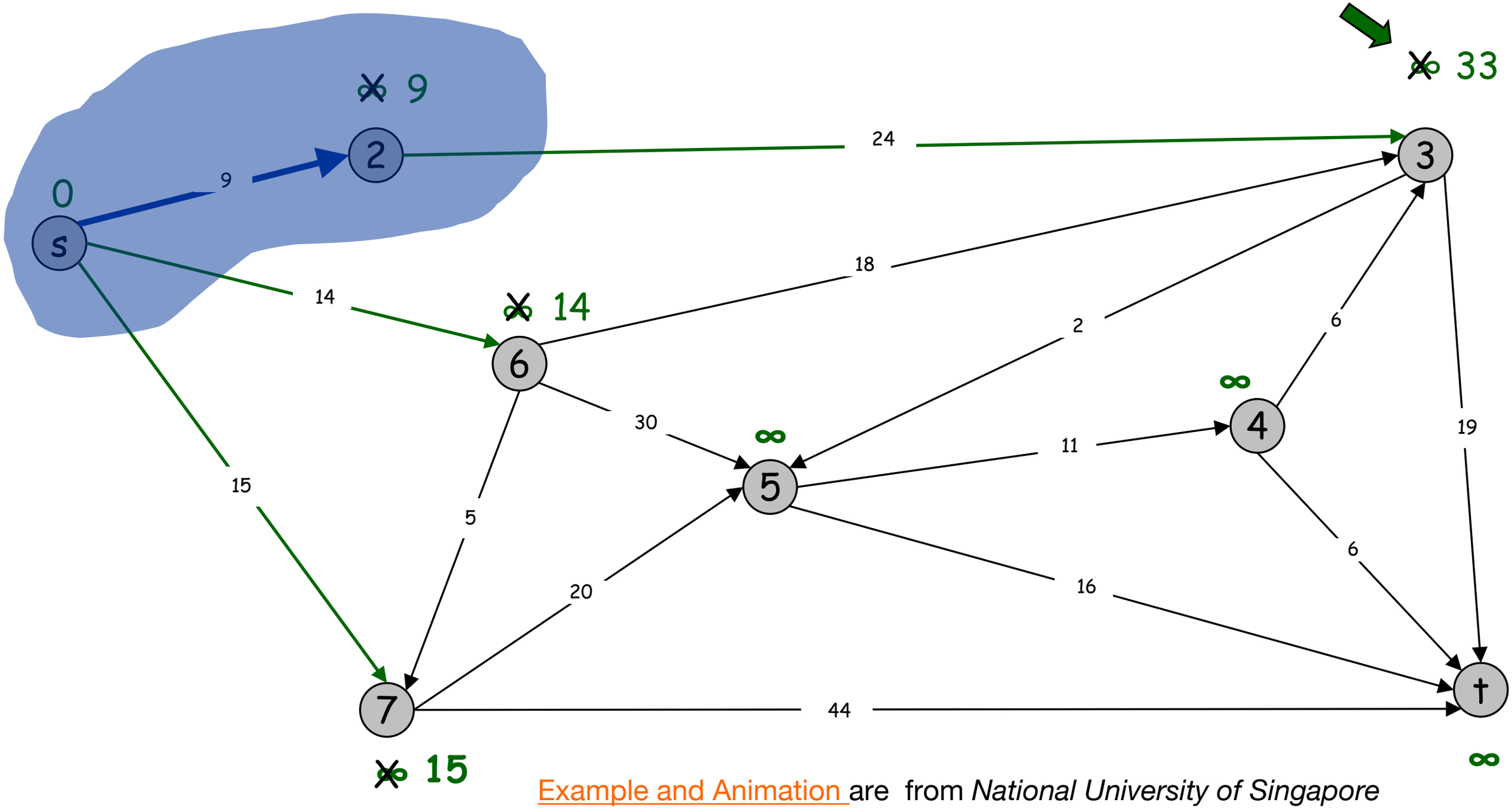
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

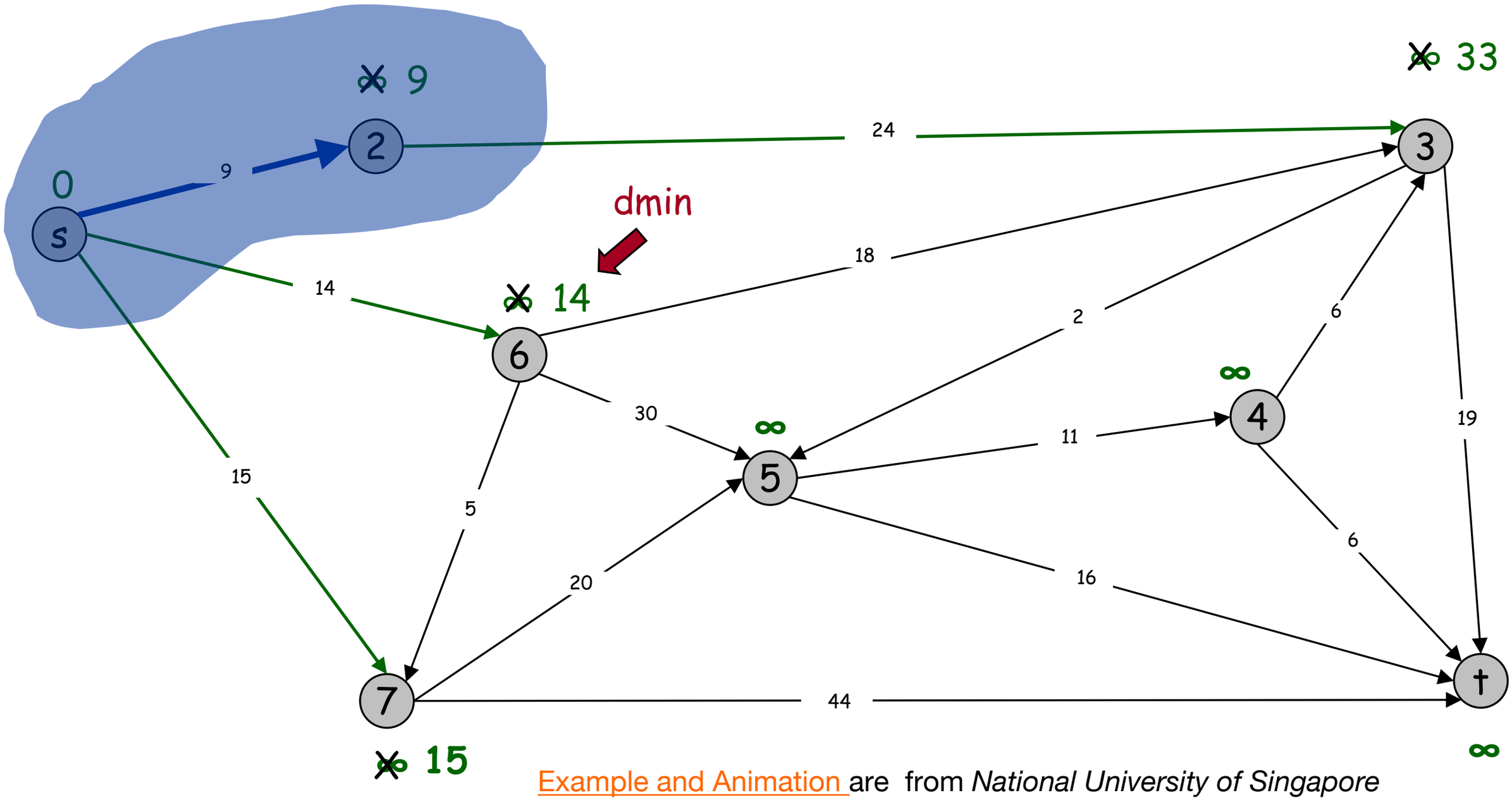
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

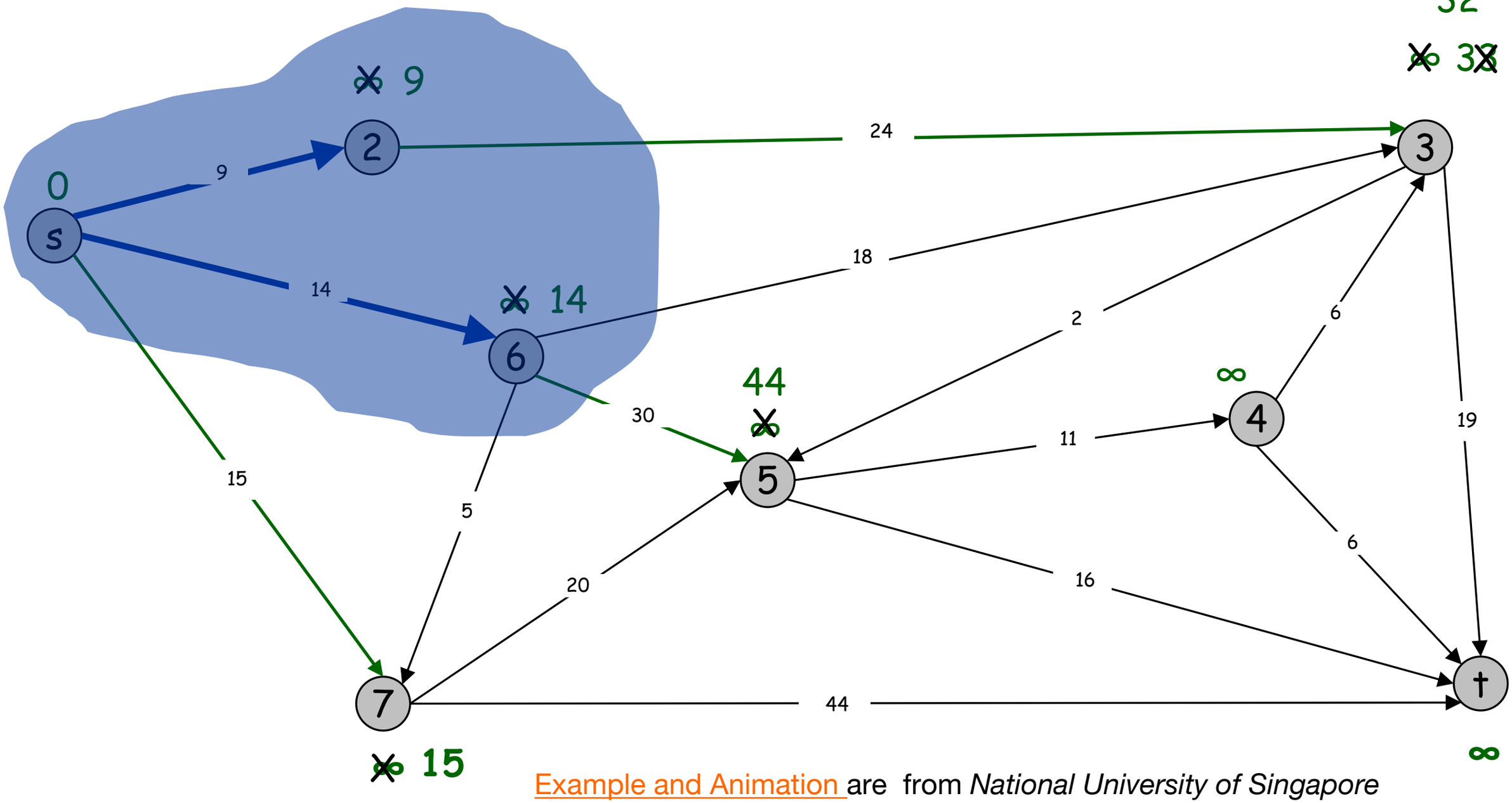
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }
B = { s, 2 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

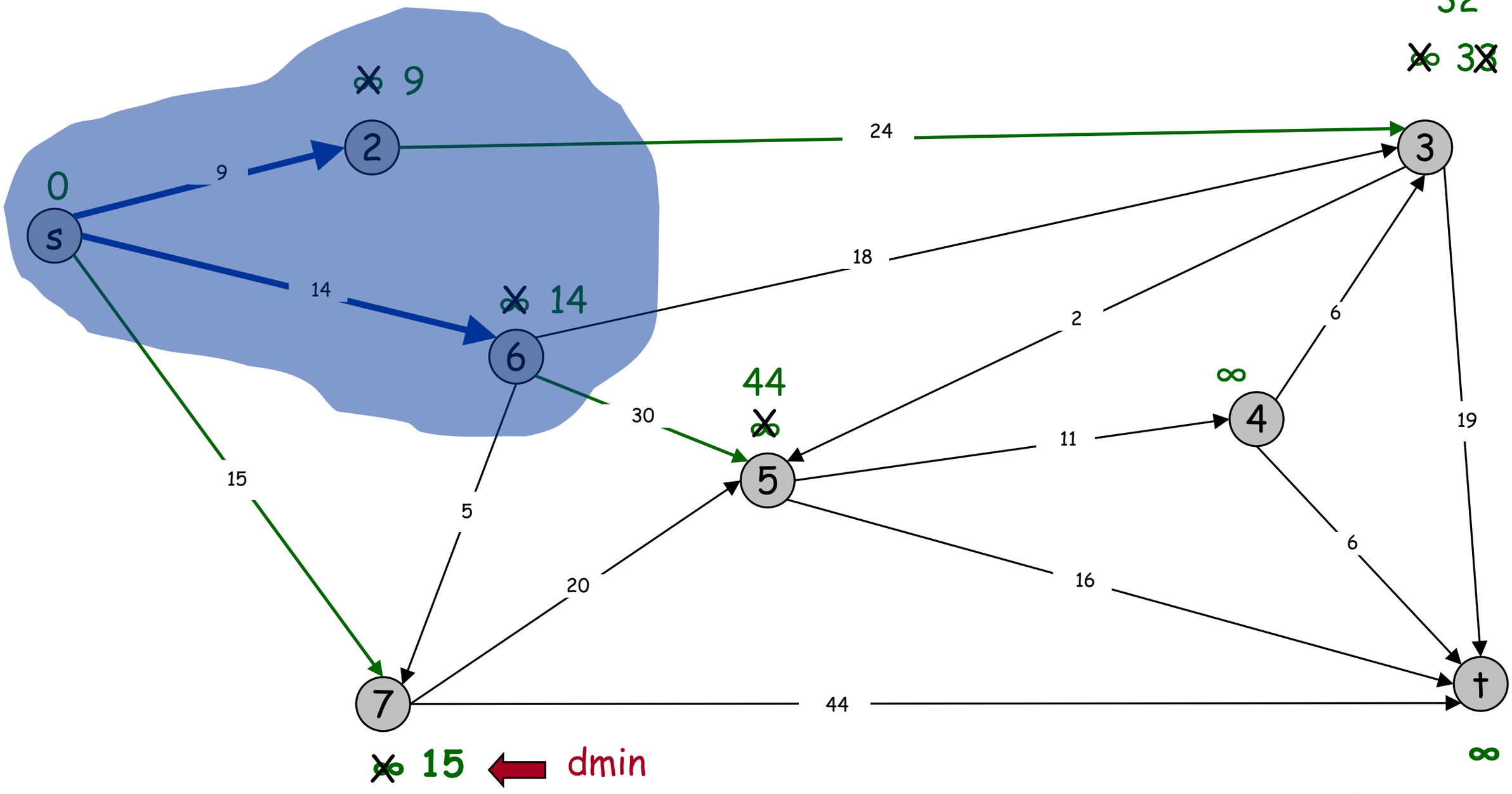
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }
B = { s, 6 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

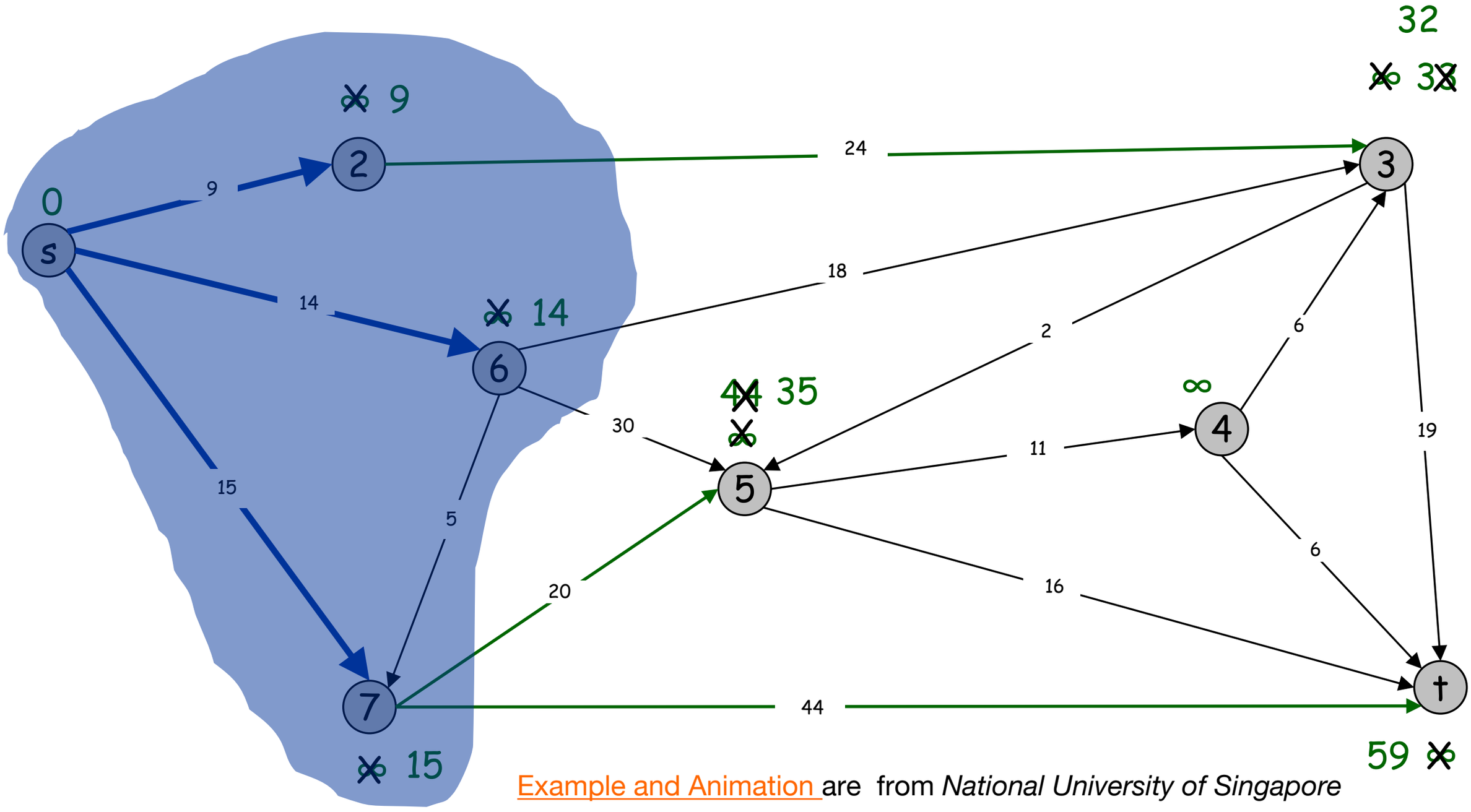
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }
B = { s, 6 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

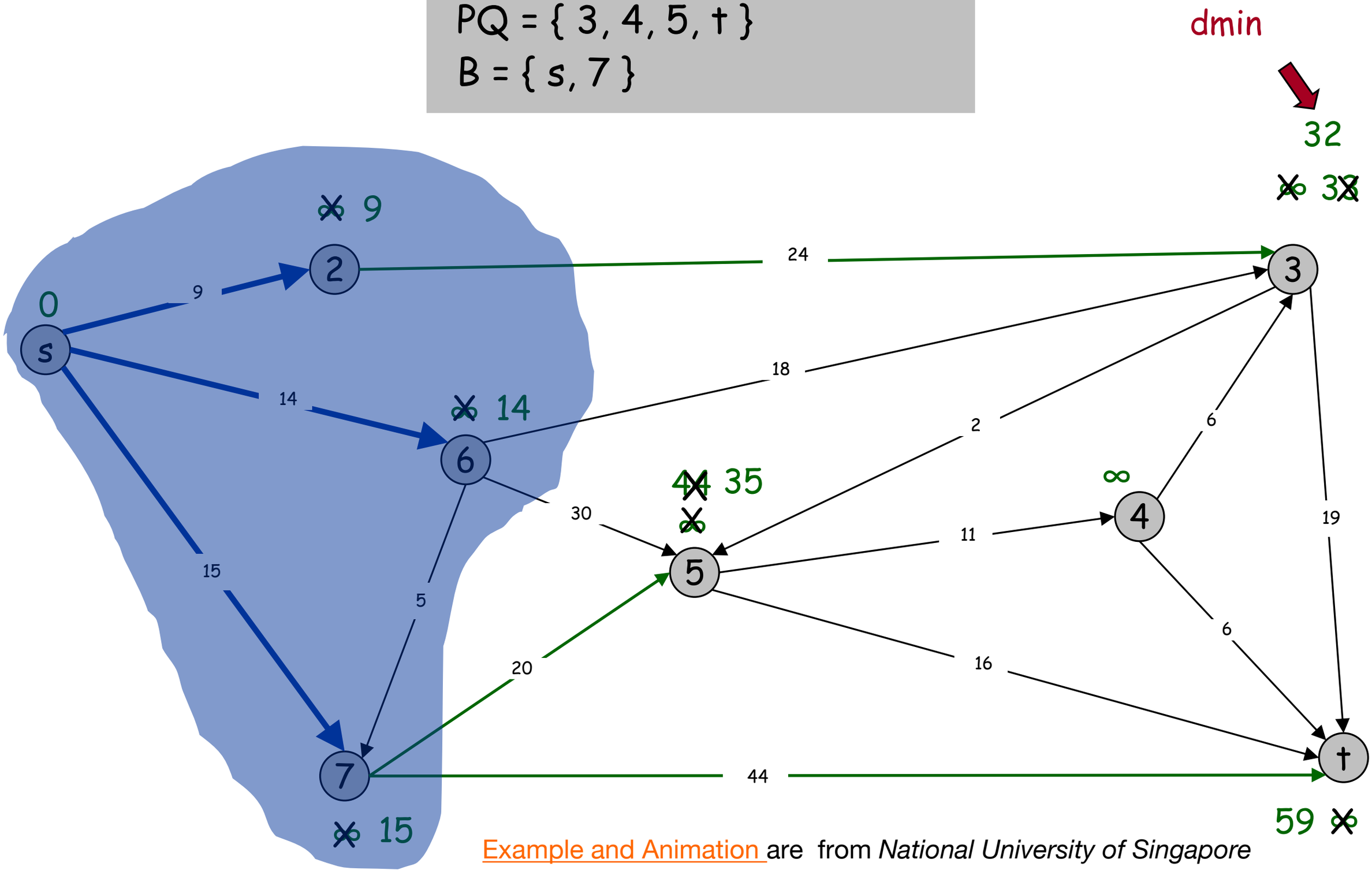
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }
B = { s, 7 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

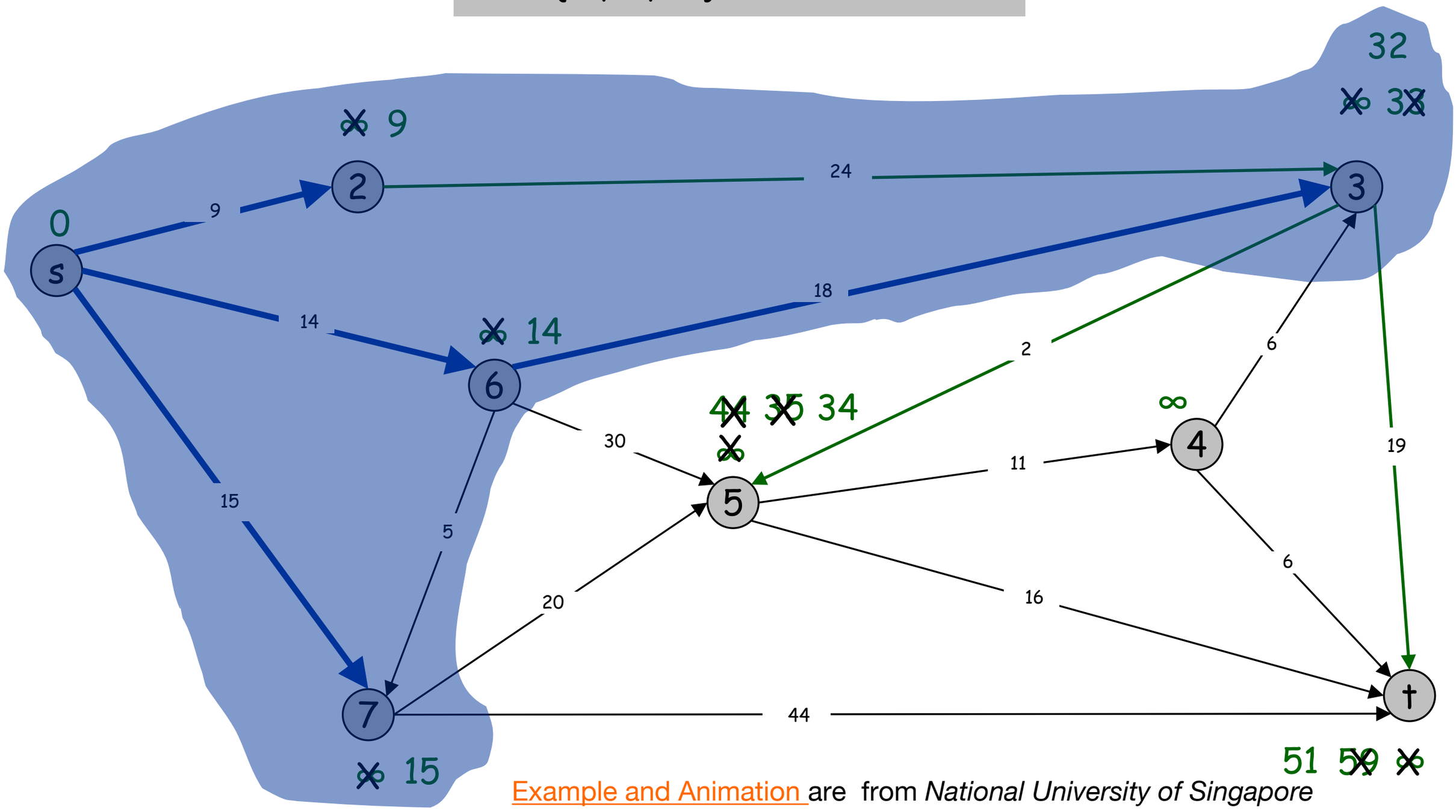
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }
B = { s, 7 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

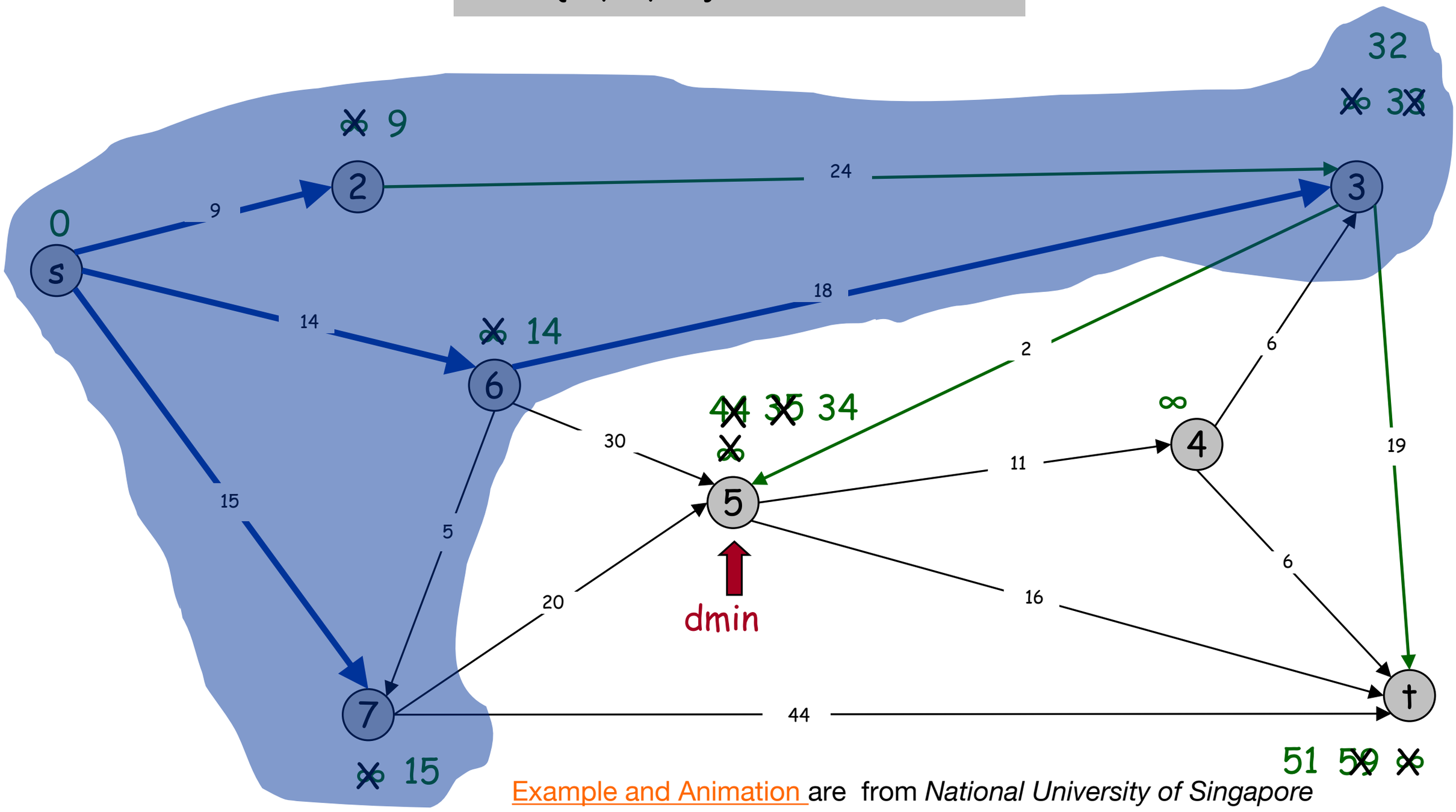
S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }
B = { s, 6, 3 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

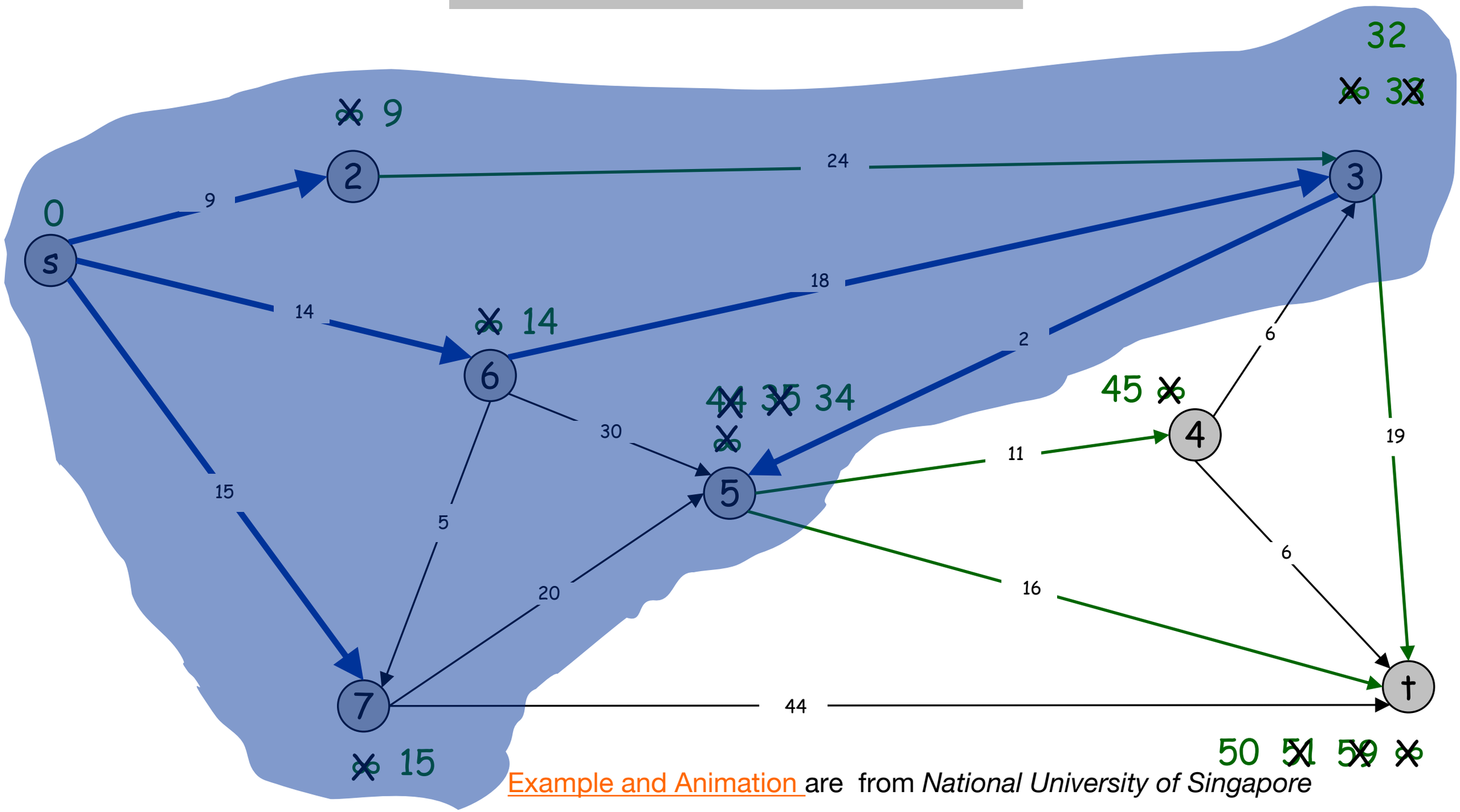
S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }
B = { s, 6, 3 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

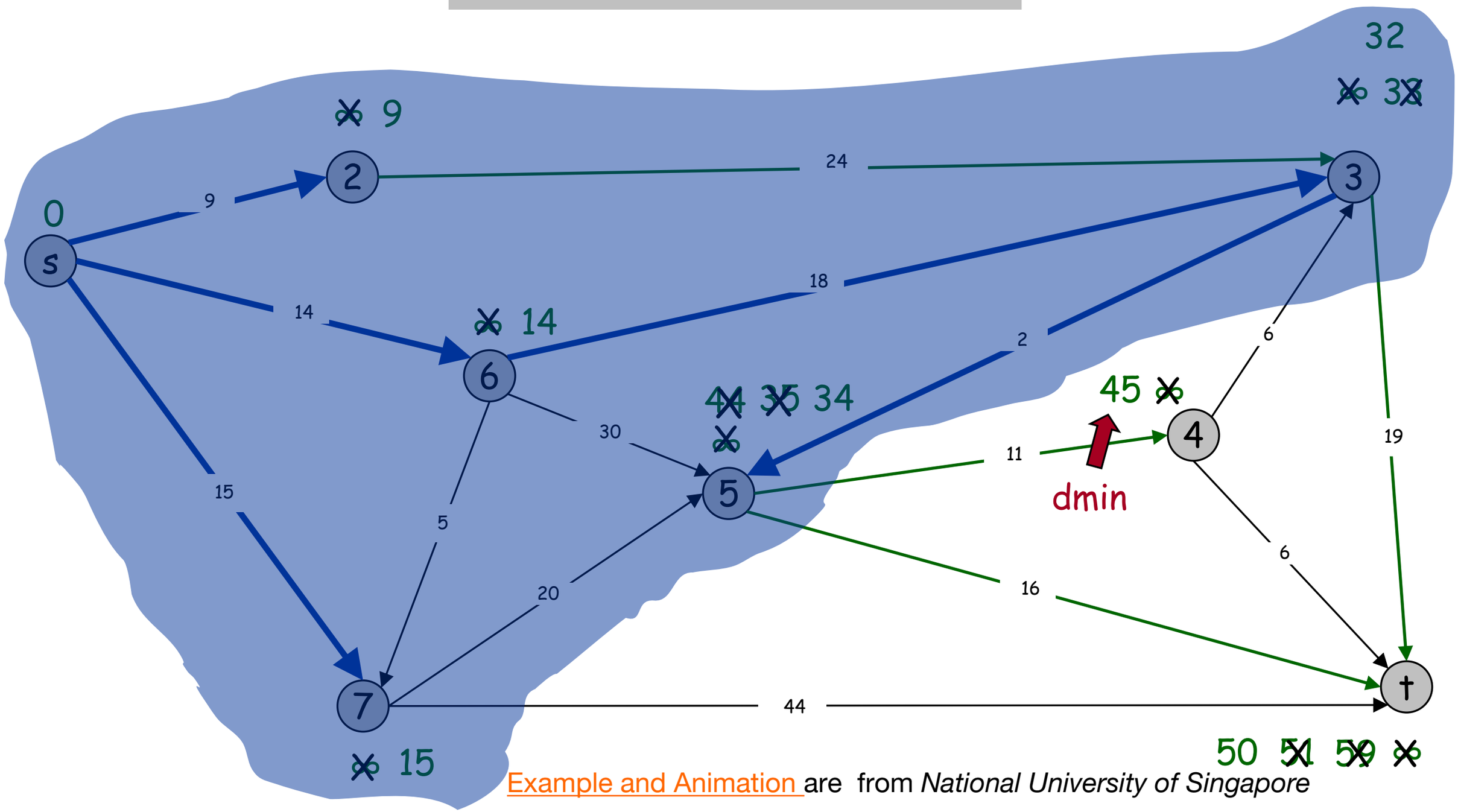
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }
B = { s, 6, 3, 5 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

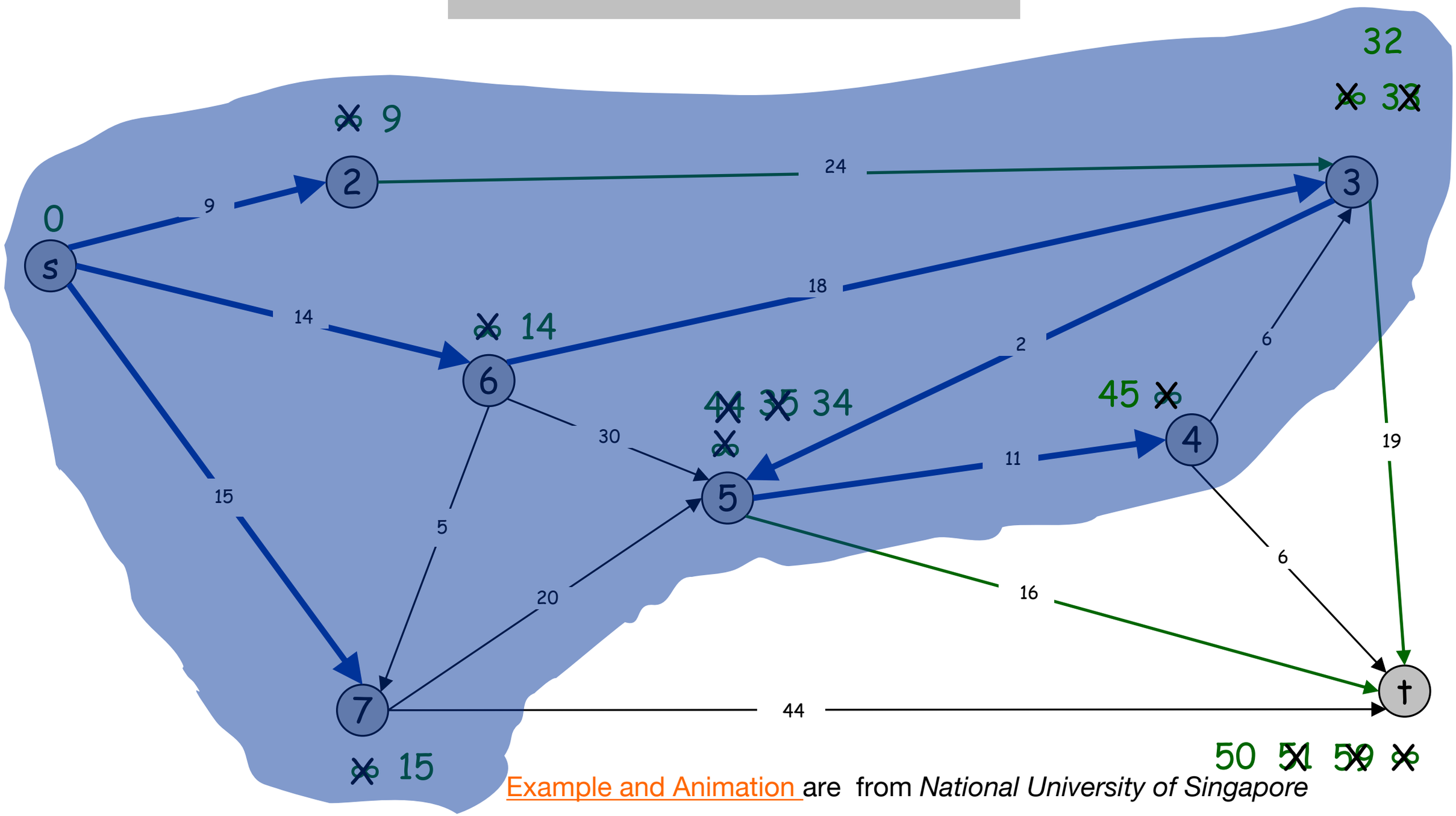
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }
B = { s, 6, 3, 5 }



Example and Animation are from *National University of Singapore*

Dijsktra's Shortest Path Algorithm

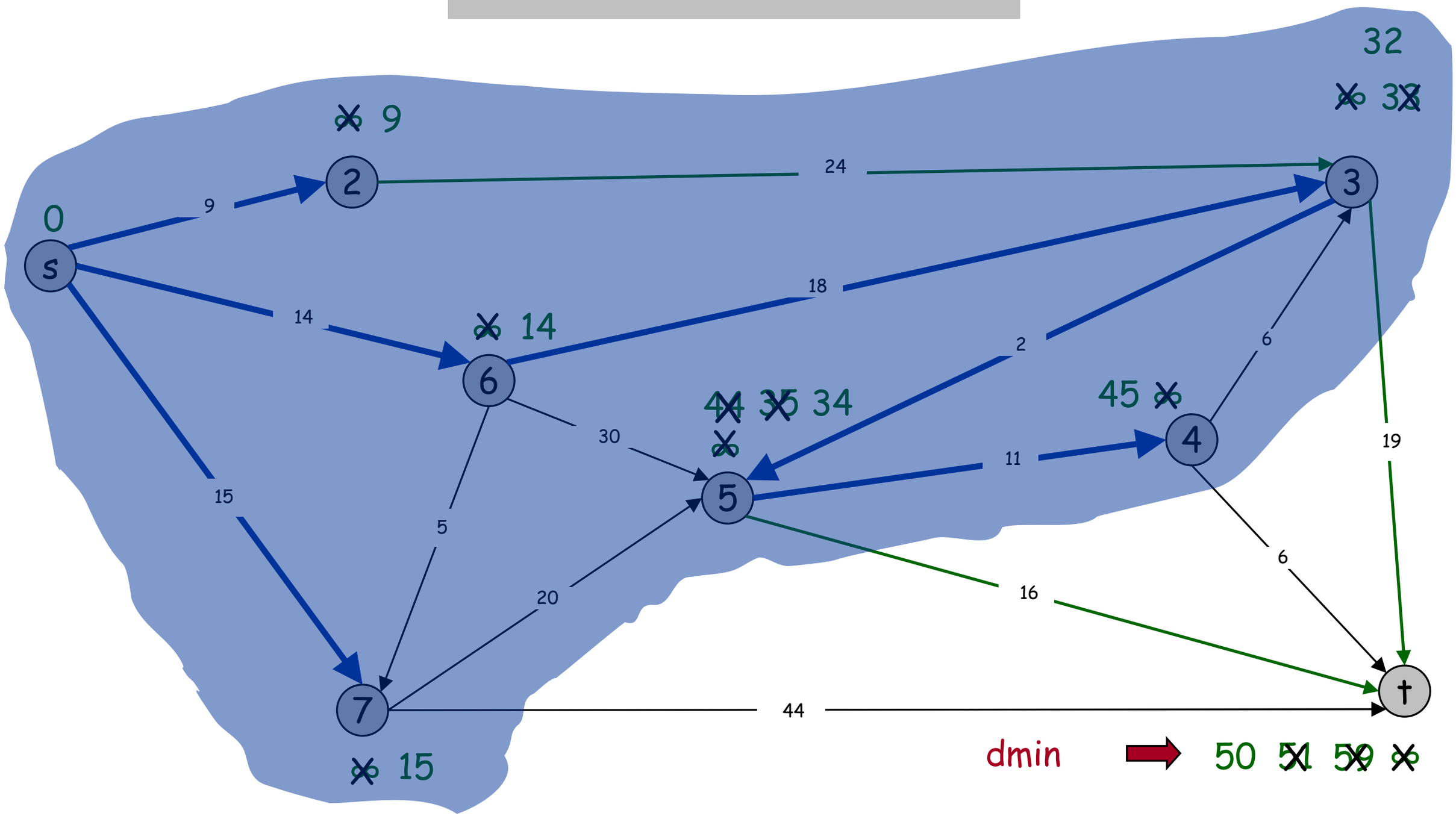
S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }
B = { s, 6, 3, 5, 4 }



Example and Animation are from *National University of Singapore*

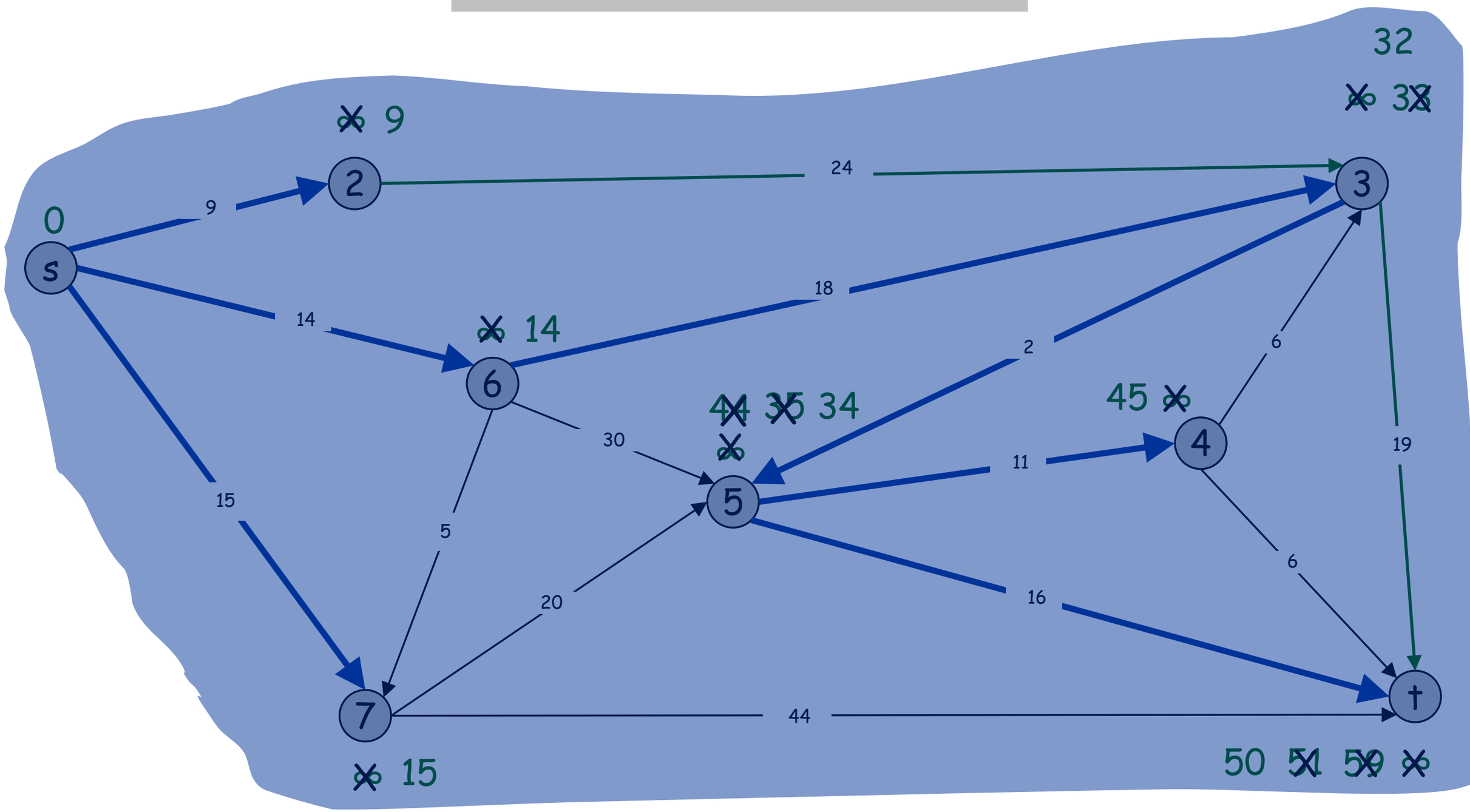
Dijsktra's Shortest Path Algorithm

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



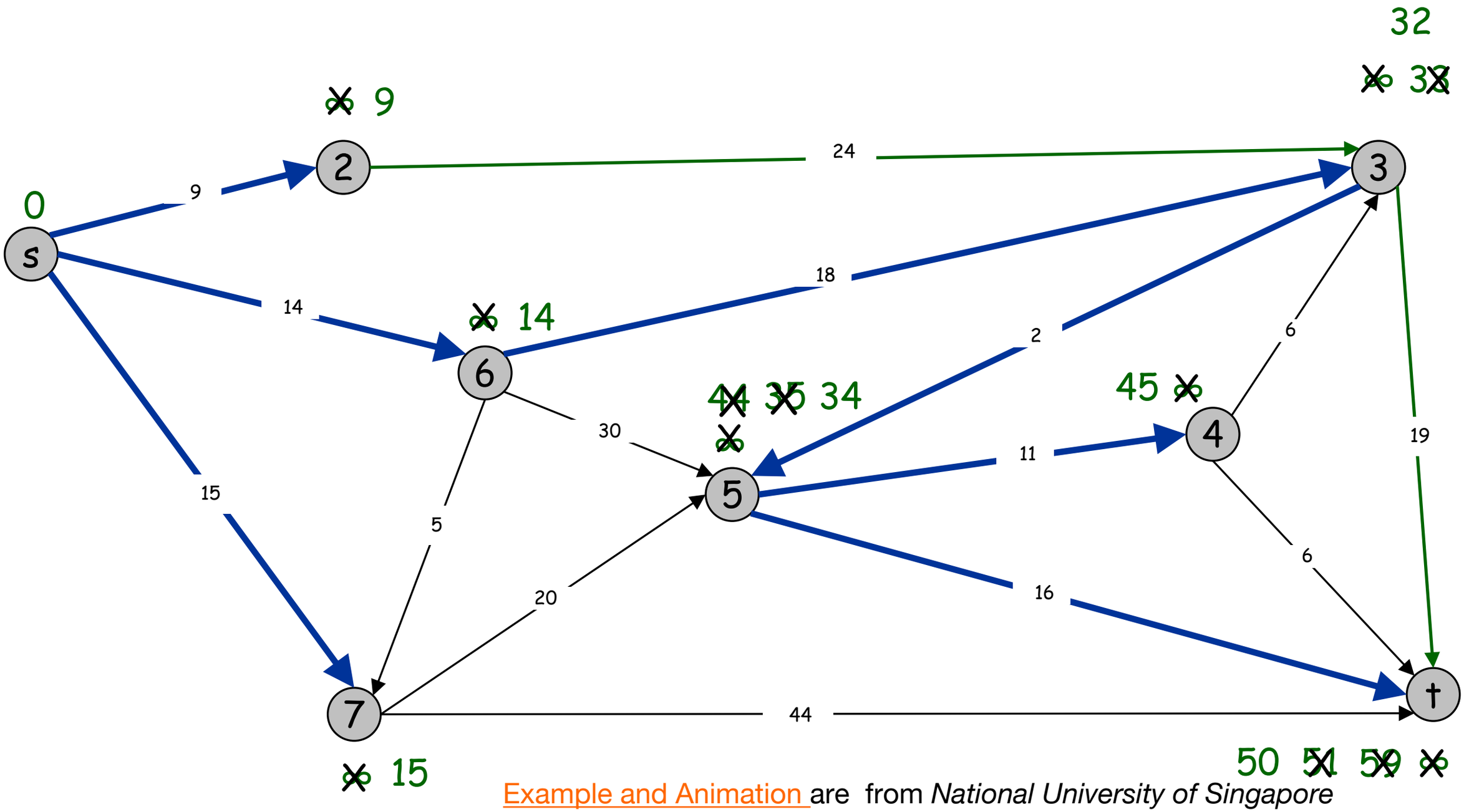
Dijsktra's Shortest Path Algorithm

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?

Θ-notation

```
for i = 0 to n
    a = a + 1
```

$\Theta(n)$

```
for i = 0 to n
    for j = 0 to n
        a = a + 1
```

$\Theta(n^2)$

Algorithm *X* executes in $10n^2 + n / 3.7 + 22$ steps $\rightarrow \Theta(n^2)$

Algorithm *Y* executes in $100n \log n + 14n + 22$ steps $\rightarrow \Theta(n \log n)$

Which algorithm is better? **Algorithm *Y***

Running Time Comparison

Good

Algorithm complexity	Size n					
	10	20	30	40	50	60
polynomial						
n	.00001sec	.00002sec	.00003sec	.00004sec	.00005sec	.00006sec
n^2	.0001sec	.0004sec	.0009sec	.0016sec	.0025sec	.0036sec
n^3	.001sec	.008sec	.027sec	.064sec	.125sec	.216sec
n^5	.1sec	3.2sec	24.3sec	1.7min	5.2min	13min
super-polynomial						
2^n	.001sec	1.0sec	17.9min	12.7days	35.7yr	366cen
3^n	.059sec	58min	6.5yr	3855cen	$2 \cdot 10^8$cen	$1 \cdot 10^{13}$cen
$n!$	3.63sec	771cen	$8 \cdot 10^{16}$cen	$3 \cdot 10^{32}$cen	$1 \cdot 10^{49}$cen	$3 \cdot 10^{66}$cen

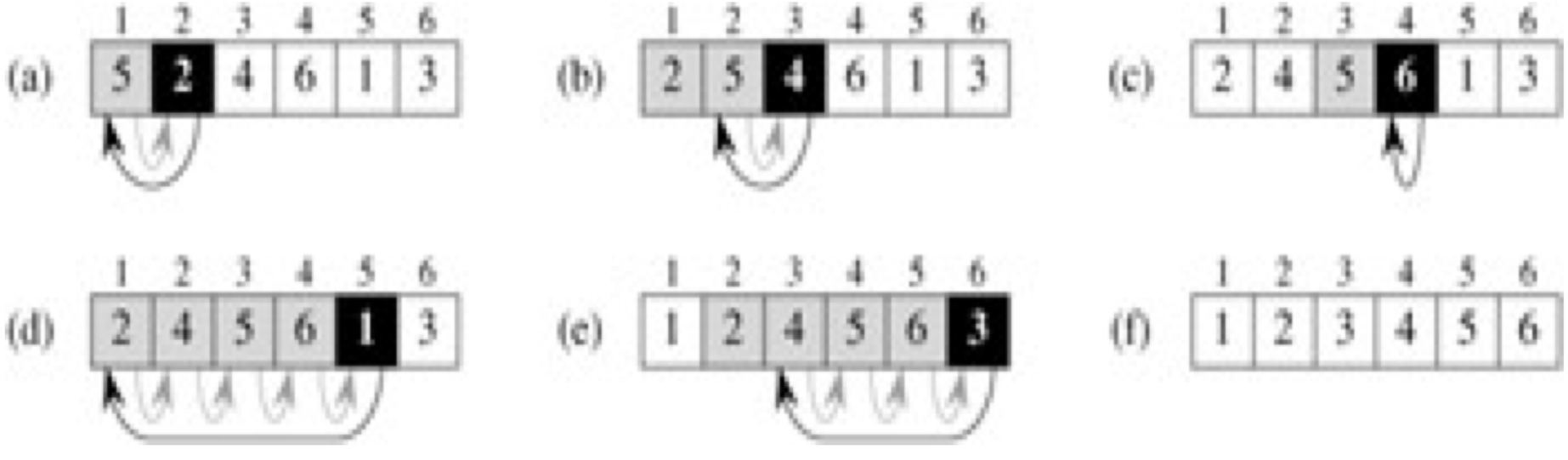
Bad!

Sorting Algorithms

- Sorting is a fundamental operation in computer science and algorithms
- Many algorithms use it as an intermediate step
- There are several sorting algorithms, differing in their efficiency.

Insertion Sort

- Start with empty left hand
- Pick one card at a time
- Insert it into the correct position in left hand
  - To find correct position, compare it with each of the cards already in hand

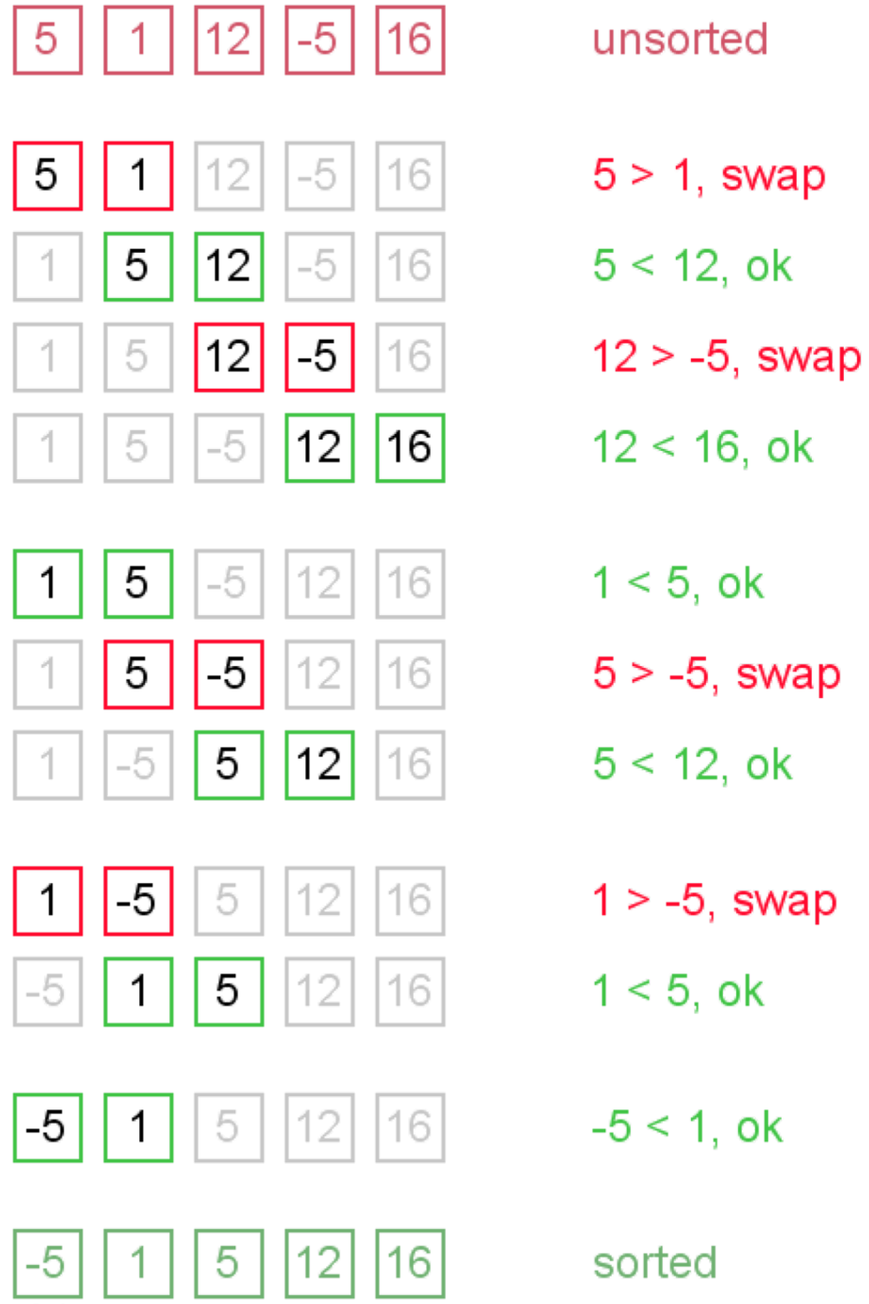


Insertion Sort

```
INSERTION-SORT(A)
1  for j ← 2 to length[A]
2       do key ← A[j]
3          ▹ Insert A[j] into the sorted sequence A[1 □ j - 1].
4          i ← j - 1
5          while i > 0 and A[i] > key
6              do A[i + 1] ← A[i]
7                 i ← i - 1
8          A[i + 1] ← key
```

Bubble Sort

- Go through the list:
  - If the next element is smaller:
    - SWAP!

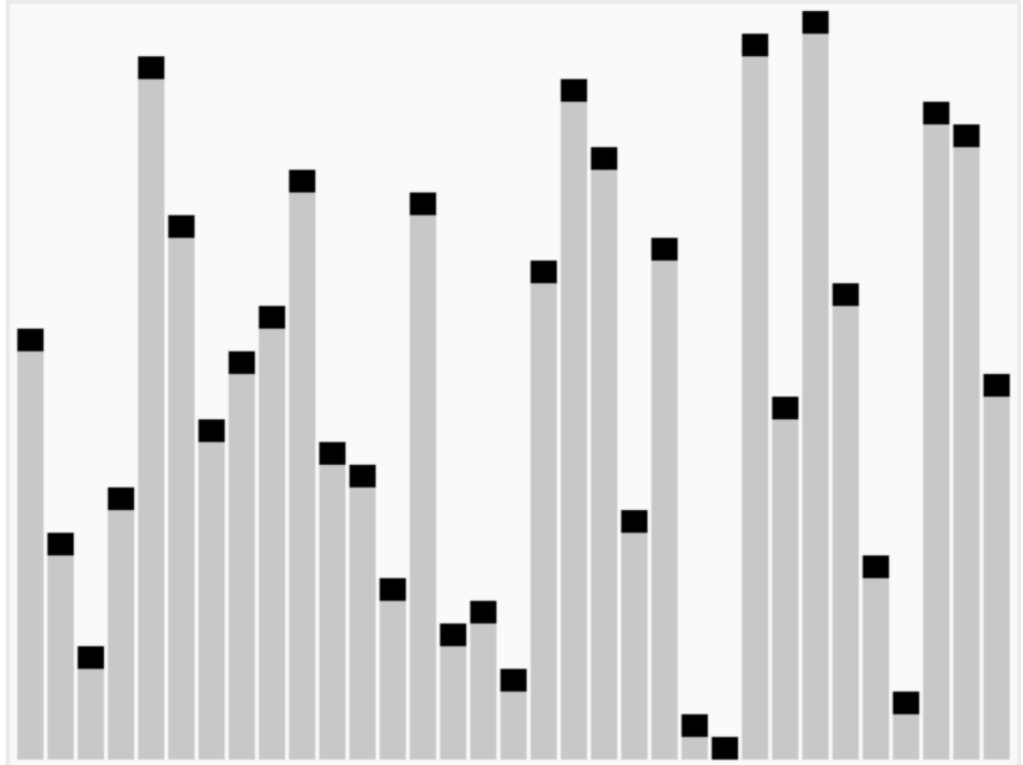


Bubble Sort

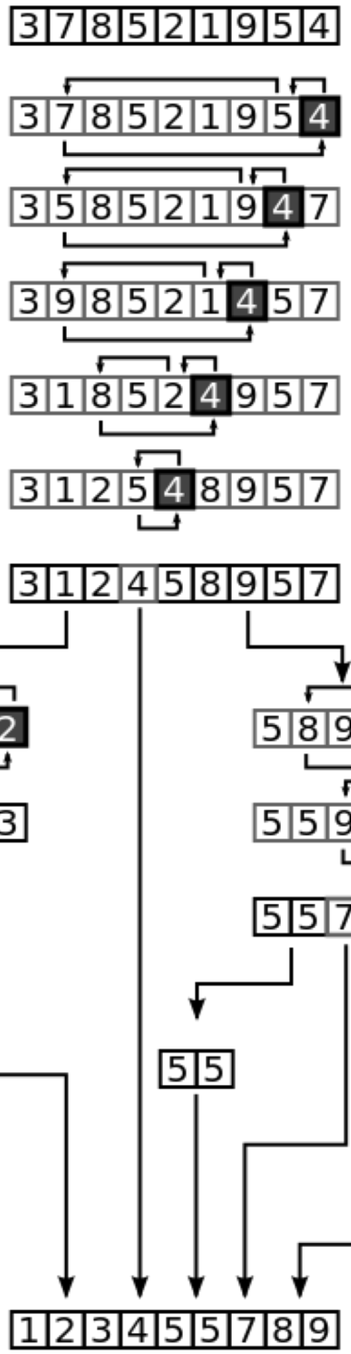
```
BUBBLESORT(A)
1 for i ← 1 to length[A]
2       do for j ← length[A] downto i + 1
3             do if A[j] < A[j - 1]
4                   then exchange A[j] ↔ A[j - 1]
```

Quick Sort

- Divide and conquer approach:
  - Pick an element from the list
  - Move elements smaller than the element before it
  - Move elements greater than it after it
  - Repeat for sub-problems



Animation and example from
https://en.wikipedia.org/wiki/Quicksort



Quicksort

```
QUICKSORT(A, p, r)
1 if p < r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q - 1)
4          QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p - 1
3  for j ← p to r - 1
4       do if A[j] ≤ x
5             then i ← i + 1
6                  exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Exercise: Bubble sort and comparison to Python sorting algorithm

---

➢Download material from
https://github.com/ucla-lacc-2019/2_Algorithms/tree/master/Sorting_Example

➢We will use the Jupyter server on 128.97.92.31

➢Enjoy writing a BUBBLE SORT ALGORITHM code!

Algorithmic Techniques

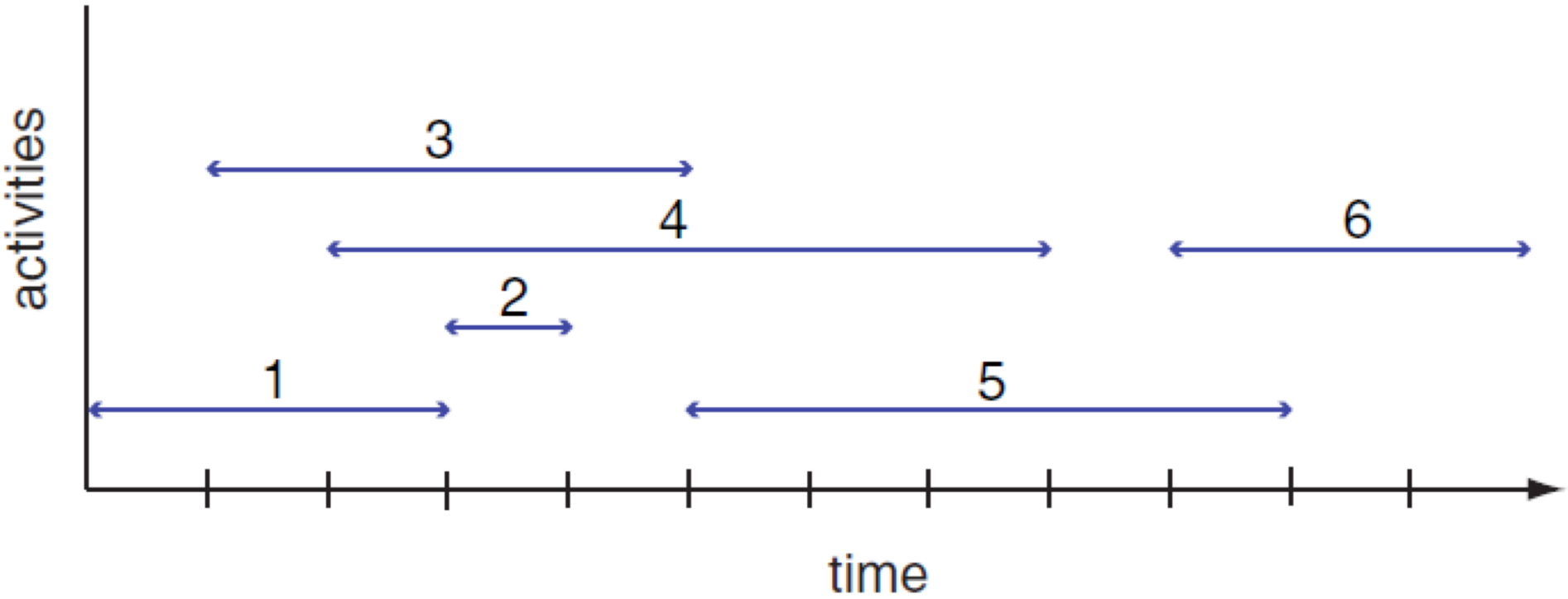
- Brute-force algorithms
  - Take the most direct or obvious solution approach
  - Enumerate all possible candidate solutions

- 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)

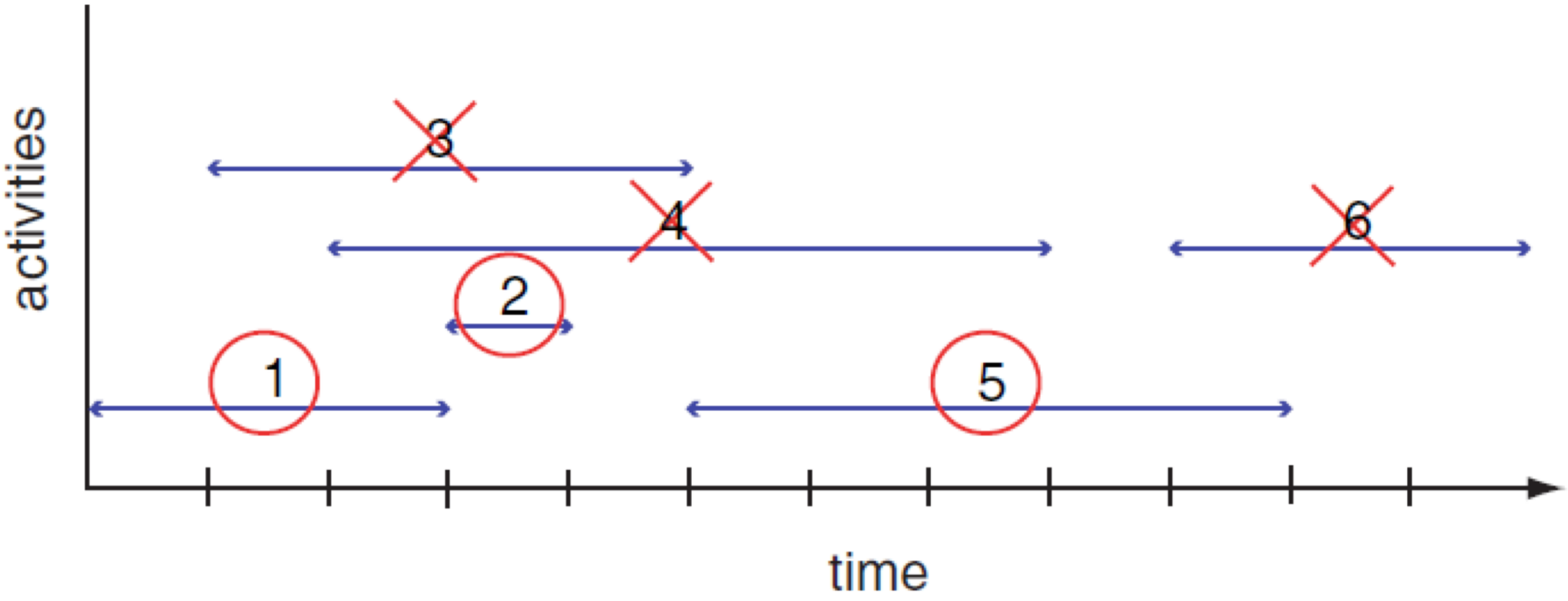
Greedy Algorithm



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

1 $A \leftarrow 1$; $j \leftarrow 1$
2 for $i \leftarrow 2$ to n do
3 if $s_i \geq f_j$ then ▷ greedy choice
4 $A \leftarrow A \cup \{i\}$
5 $j \leftarrow i$
6 return A

- What is the running time for this algorithm?

Traveling Salesman Problem

- Given a list of cities and their pairwise distances,
  - find the shortest possible route that visits each city exactly once and returns to the origin city
  - How would you tackle this problem?

Mini-project: Trump's travel plan in Python

1. **Think of a good algorithm to reduce the cost of Mr. Trump's trip**
   - Mr. Trump wants to visit 52 cities
   - Travel cost= air fuel cost * distance
   - Air fuel consumption rate= 5 gallons per mile (estimate for a Boeing 747)
   - **Implement your algorithm in the python function provided**
     - Edit the traveler_orig.py
     - You are provided a function get_distance() to calculate distance between two locations.
   - **Prepare 2-3 slides** describing your strategy and your final solution and total distance
   - Can we solve it optimally?
2. **Optional:** Due to weather conditions, direct flights between some pairs of states are not possible.
   - Find a generic solution
   - As an example: these direct flights are blocked:
     - Austin, TX – Miami,FL
     - Washington D.C. – Baltimore, MD
     - Ann Arbor, MI – St Louis, MO
     - Vancouver, BC –Raleigh-NC
     - Atlanta, GA – Miami, Fl

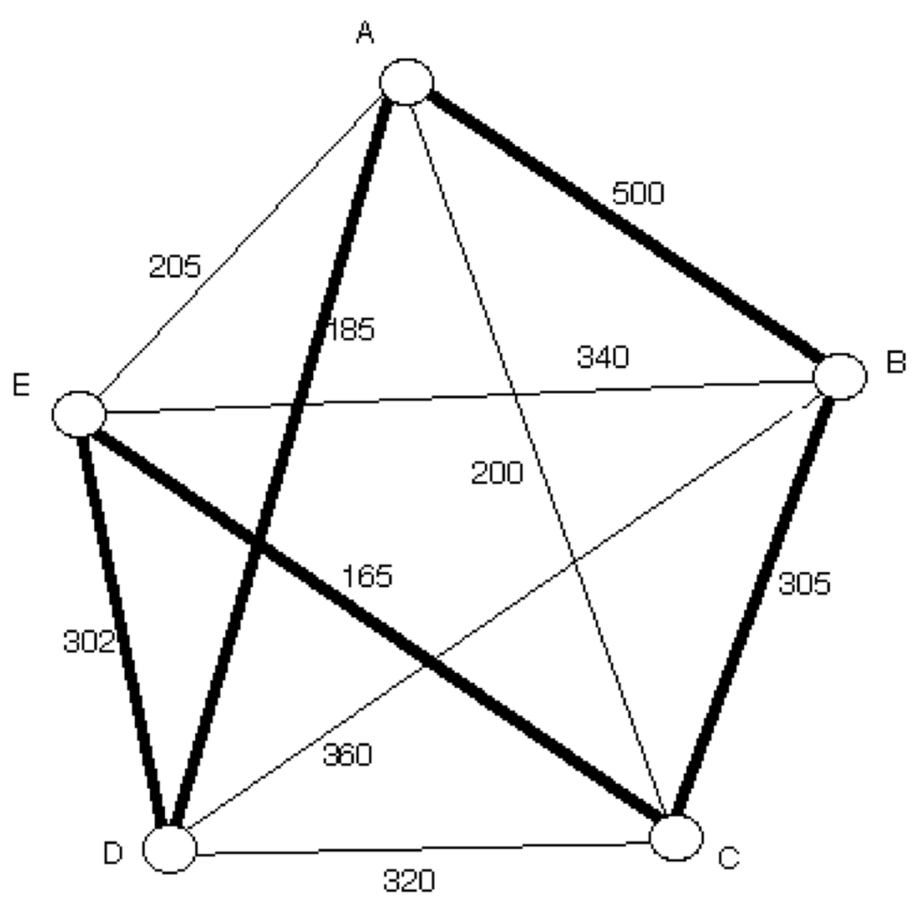
Mini-project: Trump's travel plan in Python

- Download code from
  https://github.com/ucla-lacc-2019/2_Algorithms/tree/master/Travelling_Salesman
  Submit code and slides by 3:00 pm on Thursday on Github

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

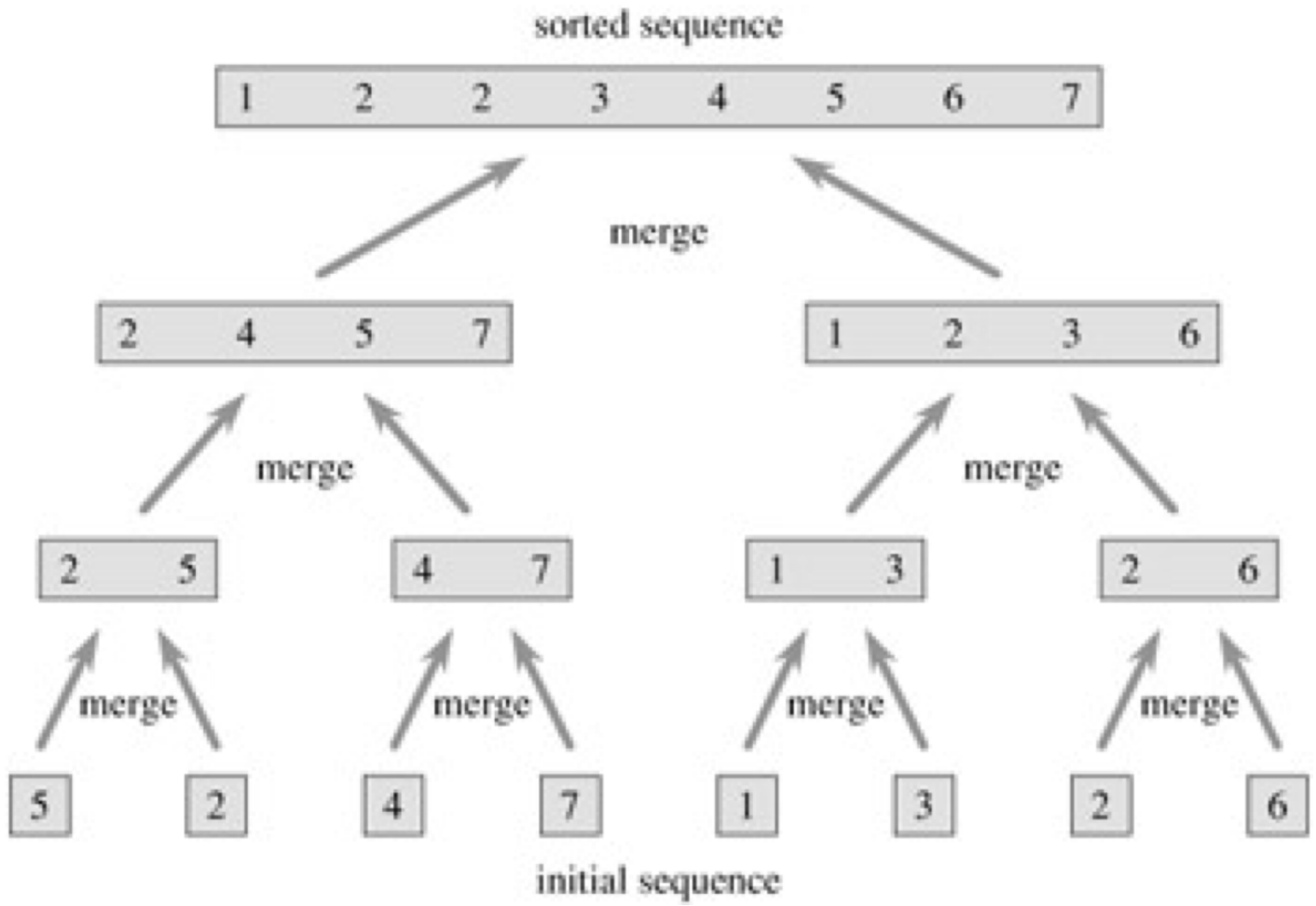


Deficiencies of Nearest Neighbor

- 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?

ADDITIONAL MATERIAL

Merge Sort



- Divide and conquer algorithm
- Recursive algorithm
  - Split the list
    - Apply merge sort to the smaller list
  - After the smaller lists are collected: MERGE
    - (Note that each of the sublists is already sorted)

Mergesort

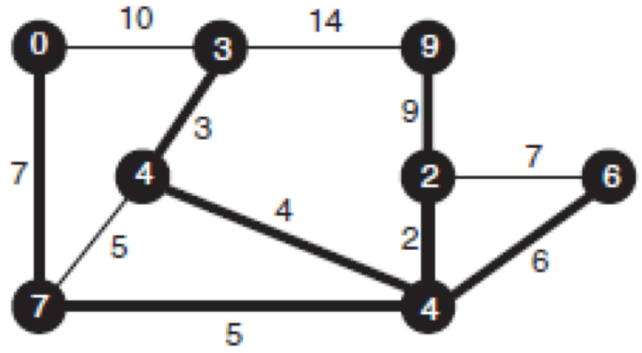
```
MERGE-SORT(A, p, r)
1 if p < r
2    then q ← ⌊(p + r)/2⌋
3         MERGE-SORT(A, p, q)
4         MERGE-SORT(A, q + 1, r)
5         MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
 1  n1 ← q - p + 1
 2  n2 ← r - q
 3  create arrays L[1 □ n1 + 1] and R[1 □ n2 + 1]
 4  for i ← 1 to n1
 5       do L[i] ← A[p + i - 1]
 6  for j ← 1 to n2
 7       do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13       do if L[i] ≤ R[j]
14            then A[k] ← L[i]
15                 i ← i + 1
16            else A[k] ← R[j]
17                 j ← j + 1
```

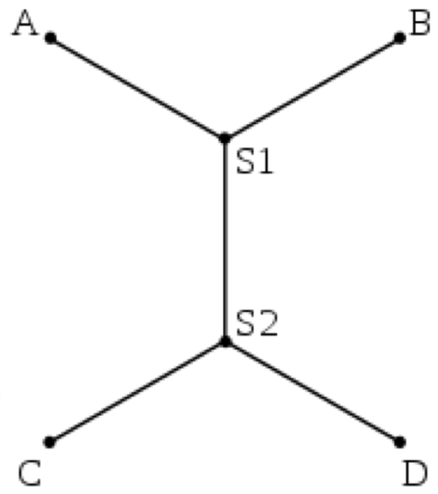
Optional Assignments

- Look into the following Algorithms

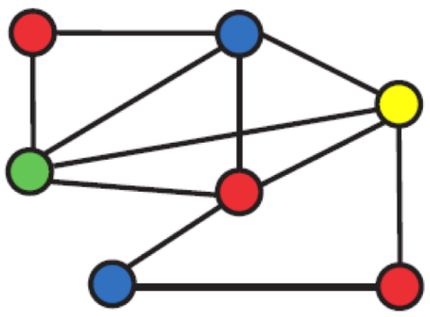
(1) Minimum spanning tree



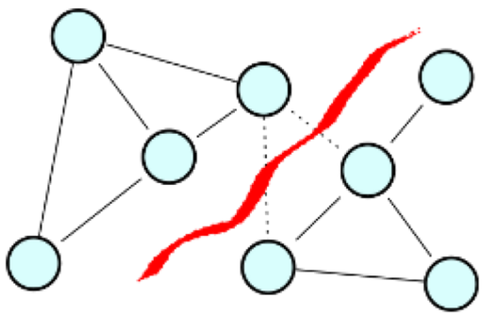
(2) Minimum Steiner tree



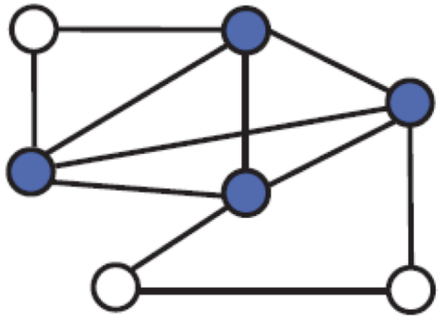
(3) Graph coloring problem



(4) Graph partitioning problem

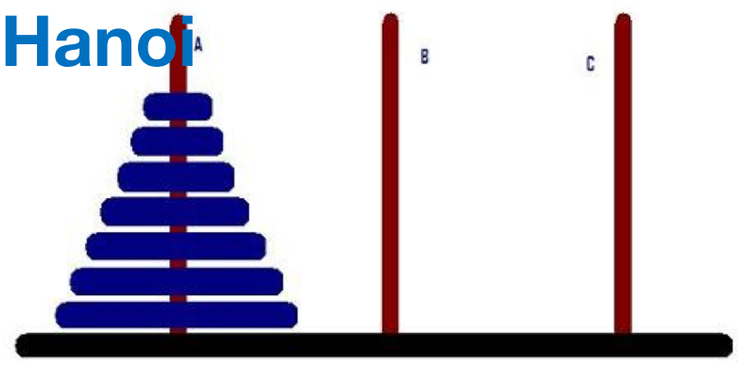


(4) Maximum Clique problem



(6) Knapsack problem

(7) Towers of Hanoi



Graph Exercises

- Graphs - nearest neighbor algorithm
  - http://nodebox.net/code/index.php/Graph