Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 4

University of Southern California

Fall 2023

Greedy Algorithms

Reading: chapter 4

Heaps for Priority Queue

•

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$ (ac)
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)
merge	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)

Review

1. (T/[✓ F]) By using a binomial heap we can sort data of size *n* in *O*(*n*) time.

2. Prove that it is impossible construct a min-heap (not necessarily binary) in a comparison-based model with *both* the following properties:

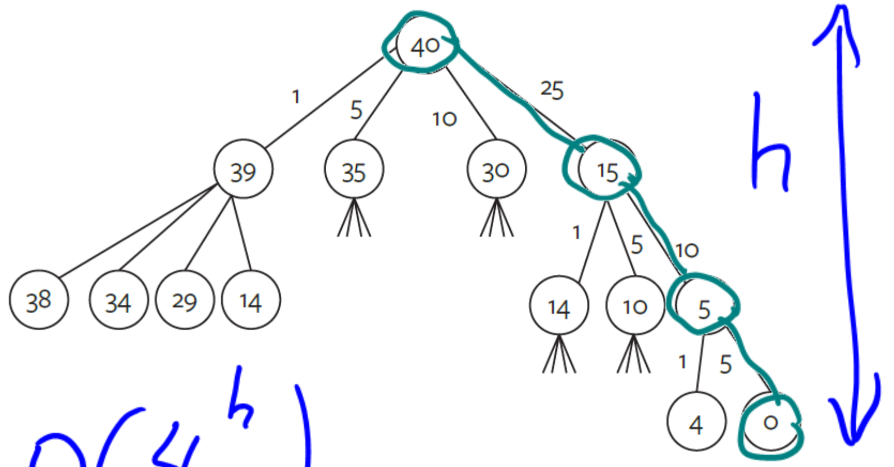
a. deleteMin() runs in *O*(1),

b. buildHeap() runs in *O*(*n*),

where *n* is the input size.

The Money Changing Problem

We are to make a change of $0.40 using US currency and assuming that there is an unlimited supply of coins. The goal is to compute the minimum number of coins. penny, nickel, dime, ...



greedy approach:
choose the largest coin

Runtime: $O(h)$

$O(4^h)$

Run BFS

During the algorithm execution, we don't consider all available choices at any given point, but use a <u>heuristic</u> (greedy choice) to pick just one.

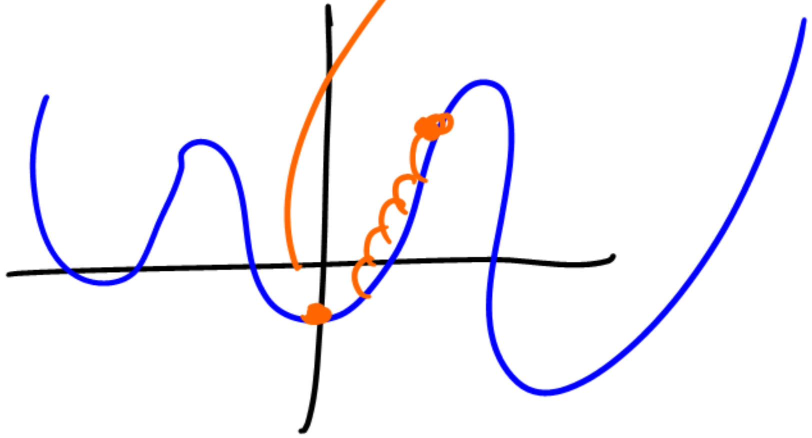
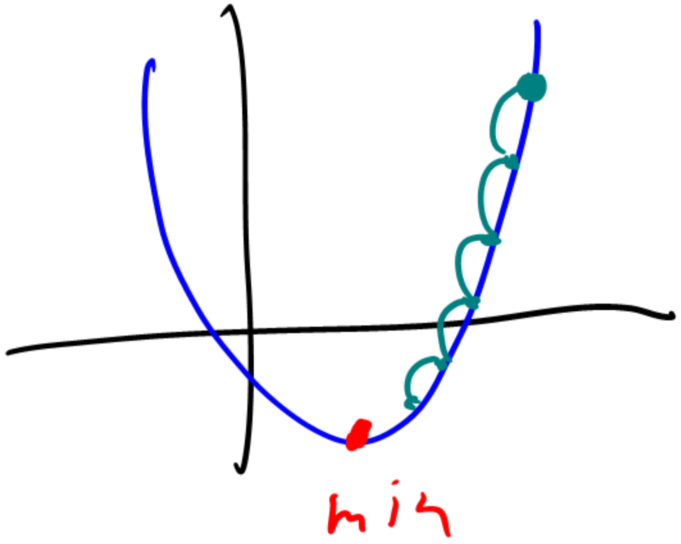
Coin system: 1, 5, 10, $\underset{\text{new}}{\underbrace{20}}$, 25 Greed: 40 = 3 coins

OPT = 2coins

What is Greedy Algorithm?

There is no formal definition…

- It is used to solve optimization problems
- It makes a local optimal choice at each step
- Earlier decisions are never undone
- Does not always yield the optimal solution

Elements of the greedy strategy

There is no guarantee that such a greedy algorithm exists, however a problem to be solved must obey the following two common properties:

greedy-choice property

and

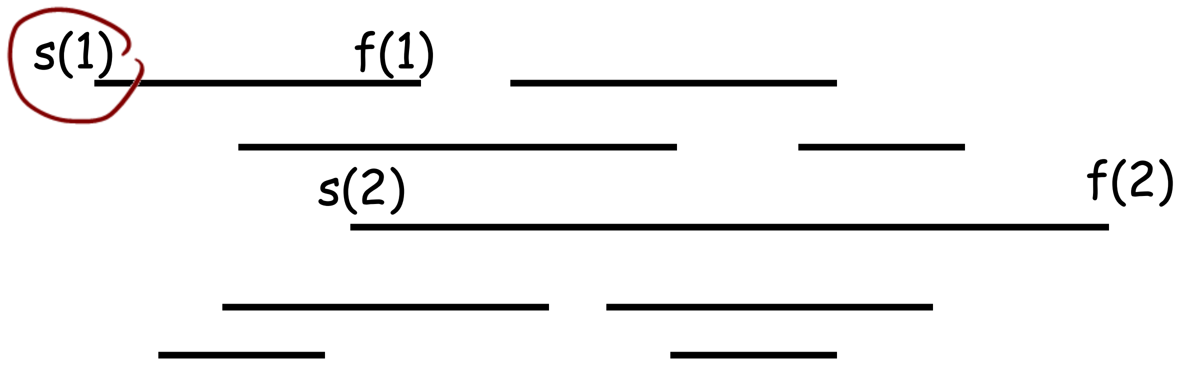
optimal substructure.

The proof of optimal substructure correctness is usually by induction.

The proof that a greedy choice for each subproblem yields a globally optimal solution is usually by contradiction.

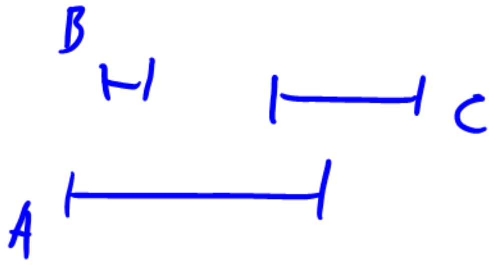
Scheduling Problem

There is a set of n requests. Each request i has a starting time s(i) and finish time f(i). Assume that all requests are equally important and s(i) ≤ f(i). Our goal is to develop a greedy algorithm that finds the largest compatible (non-overlapping) subset of requests.



How do we choose requests?

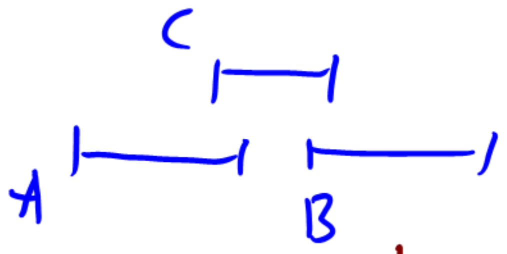
① Sort by $s(i)$, choose the earliest time



ALG: A

OPT: B, C

② Sort by $f(i)-s(i)$



ALG: C

OPT: A, B

③ Sort by $f(i)$

ex. 1 ALG: B, C

ex. 2 ALG: A, B

same

same

Proof

In this approach we sort requests with respect to *f*(*i*) in ascending order. Pick a request that has the earliest finish time.

ALG: $i_1, i_2, \ldots, i_k$ OPT: $j_1, j_2, \ldots, j_m$

Prove $f(i_r) \le f(j_r)$ by induction on r

Base case: $r=1$, first request

$$f(i_1) \le f(j_1)$$

IH: assume $f(i_{r-1}) \le f(j_{r-1})$ for $(r-1)$ requests

IS: prove it for the next request

$$f(i_{r-1}) \underbrace{\le}_{\text{IH}} f(j_{r-1}) \le s(j_r)$$

what does it mean?

$\exists$ another request

ALG will take it

$=$

Next step: prove $k \geq m$

proof by contradiction

Assume $k < m$.

① $f(j_k) \leq s(j_{k+1})$ compatible requests

② $f(i_k) \leq f(j_k)$ by I.H.

Combine them together:

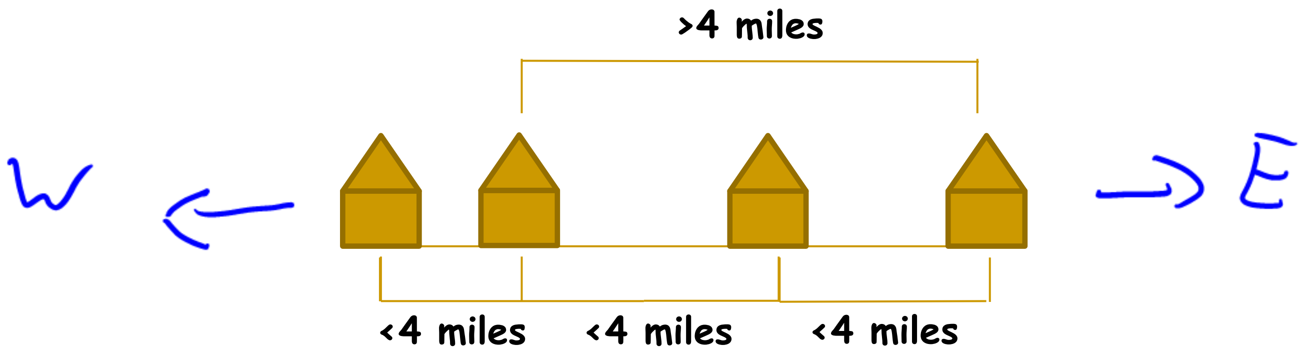
$$f(i_k) \leq f(j_k) \leq s(j_{k+1})$$

$\exists$ another request, ALG will pick it.

Discussion Problem 1

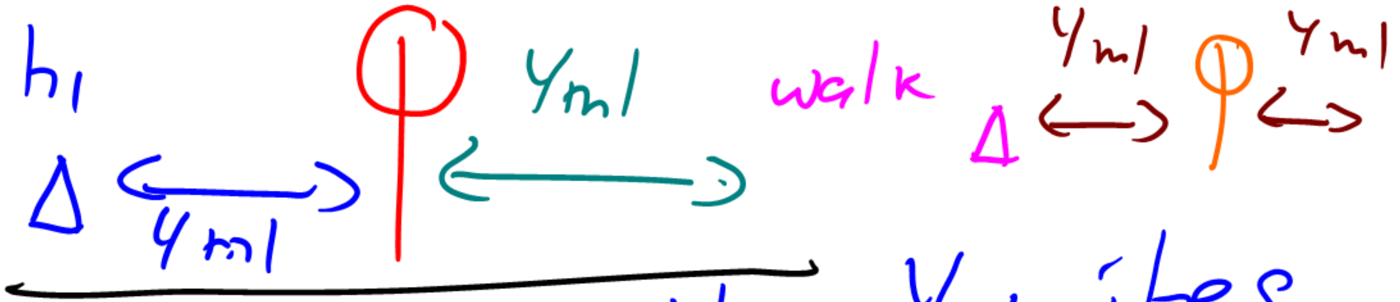
Let's consider a long, quiet country road with n houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. You want to place cell phone base stations at certain points along the road so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal and uses as few base stations as possible.

input: array of houses, h_k



Algorithm:

1. Sort h_k, from W to E
2. start with h_1, walk 4 miles, put a station



4 miles

3. mark all houses within

Runtime: $O(n \log n + n)$

Proof of correctness

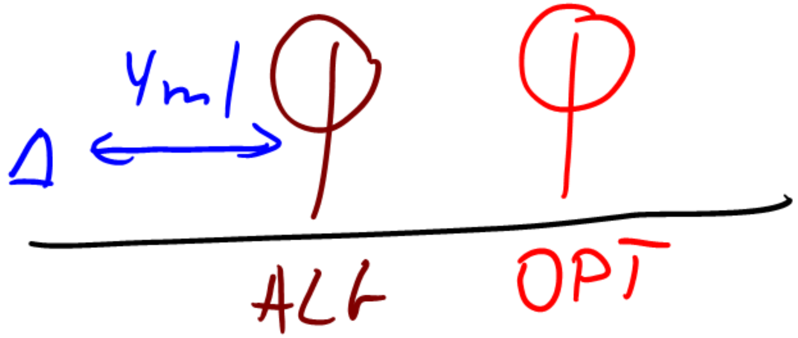
ALG: $s_1, s_2, \dots, s_k$ OPT: $t_1, t_2, \dots, t_m$

Base case: first station

IH: assume $(c-1)$ stations

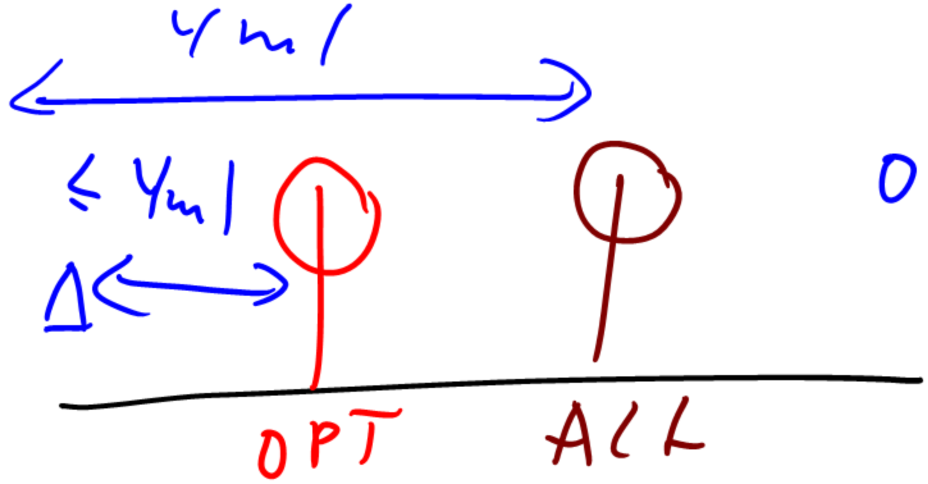
IS: prove for the next c-th station

A)



impossible

B)



OPT # stations $\geq$ ALG # stations

Break

The Minimum Spanning Tree

Given a weighted undirected graph. Find a spanning tree of the minimum total weight.

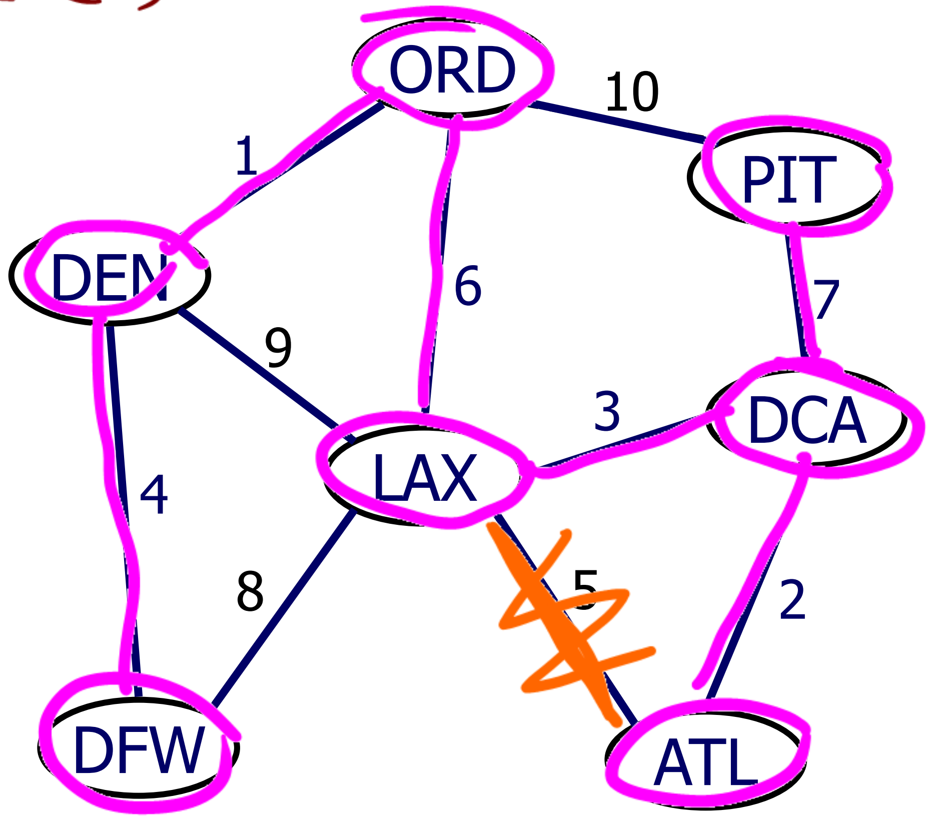
MST is fundamental problem with diverse applications.

The Minimum Spanning Tree

no cycles

Find a spanning tree of the minimum total weight.

cost: 1+2+3+4+6+7 = 23



Kruskal's Algorithm

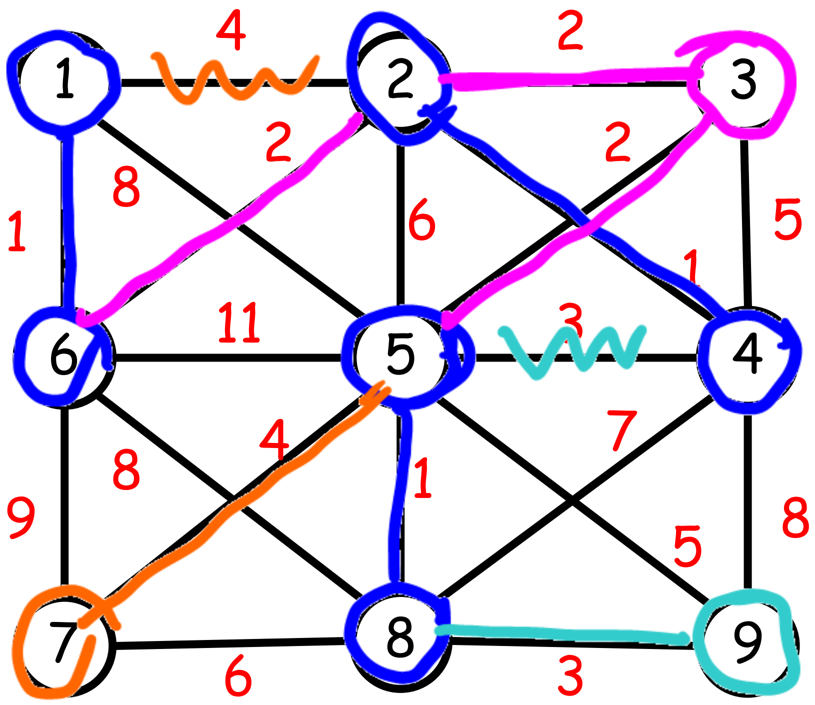
The algorithm builds a tree one EDGE at a time:

- Sort all edges by their weights. $O(E \log E)$
- Loop:
  - Choose the minimum weight edge and join correspondent vertices (subject to cycles). detecting a cycle $O(V)$
  - Go to the next edge.
  - Continue to grow the forest until all vertices are connected.

Sorting edges – O(E log E)
Cycle detection – O(V) for each edge

Total: $O(\underbrace{V*E}_{\text{cycle detection}} + \underbrace{E*\log E}_{\text{sort}})$

Kruskal's Algorithm



Exercise: suppose you do not sort edges. What is the runtime?

$O(\underbrace{V \cdot E}_{\text{cycle detection}} + \underbrace{E \cdot E}_{\text{findMin}})$

Discussion Problem 2

You are given a graph G with all distinct edge costs. Let T be a minimum spanning tree for G. Now suppose that we replace each edge weight c_e by its square, c_e^2, thereby creating a new graph G_1 with the different distinct weights. Prove or disprove whether T is still an MST for this new graph G_1. : $c \Rightarrow c^2$

$MST(G) = T$

$MST(G_1) = T_1$

Question: $T = T_1$?

Case 1) $\forall c \geq 0$, then $T = T_1$

$\frac{1}{2}, 1, 2$

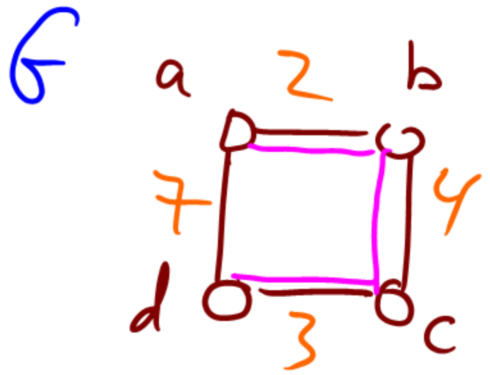
$\frac{1}{4}, 1, 4$

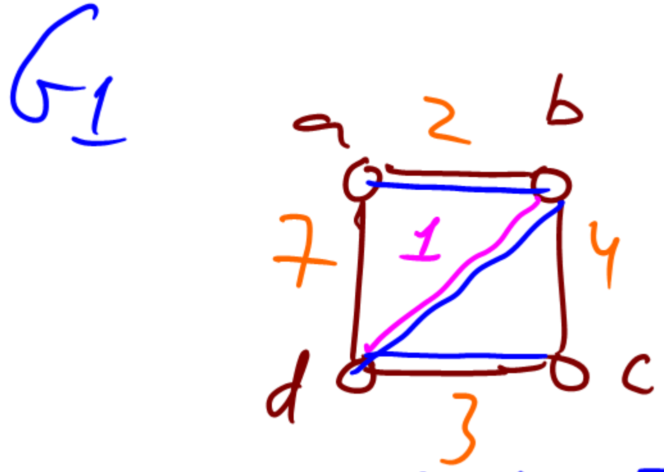
Case 2) $\exists c < 0$, then $T \neq T_1$

sorting order may change

Discussion Problem 3

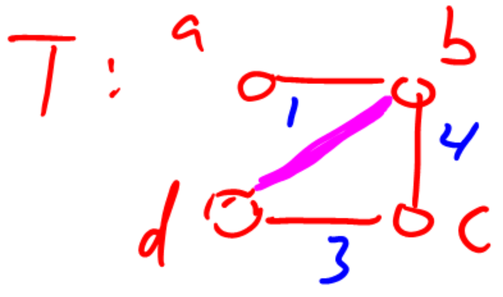
You are given a minimum spanning tree T in a graph G = (V, E). Suppose we add a new edge (without introducing any new vertices) to G creating a new graph G_1. Devise a linear time algorithm to find an MST in G_1.





$MST(G) = T$

$MST(G_1) = T_1$



Algorithm

① add that new edge to T

② traverse the cycle, remove the largest

Prim's Algorithm

The algorithm builds a tree one VERTEX at a time:

- Start with an arbitrary vertex as a sub-tree *C*.

- Expand *C* by adding a vertex having the minimum weight edge of the graph having exactly one end point in *C*.

- Update distances from *C* to adjacent vertices.

- Continue to grow the tree until *C* gets all vertices.

Prim's Algorithm: Example

T={a}

Heap: distances from T to all vertices

root ← delete Min

b-4	c-2	d-1	e-infty	f-infty

decreaseKey

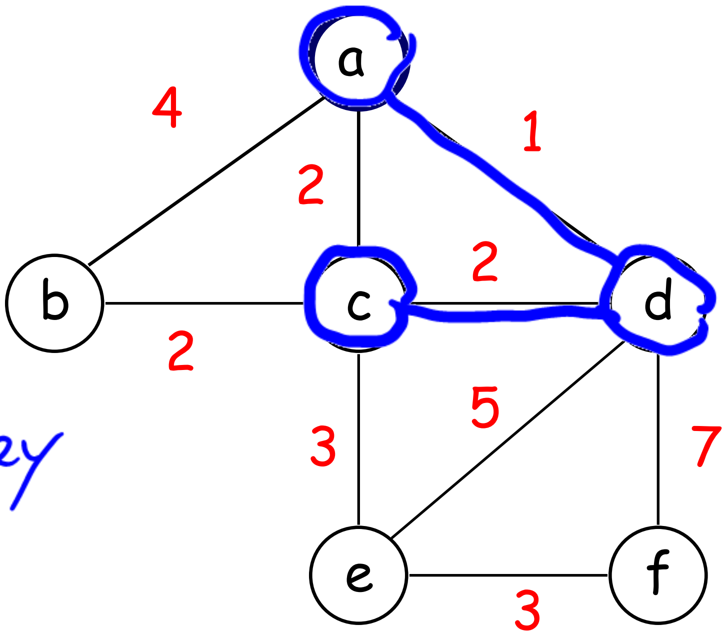
T={a, d}

root

b-4	c-2	e-5	f- 7

T={a, d, c}

b-2	e-3	f- 7



Complexity of Prim's Algorithm

Algorithm

① deleteMin, run V times

② decreaseKey, run E times

Runtime

a) binary heap $O(V \cdot \log V + E \cdot \log V)$

b) Fibonacci heap $O(V \cdot \log V + E \cdot 1)$

amortised cost

Discussion Problem 4

① Assume that an unsorted array is used as a heap. What would the running time of the Prim algorithm?

$O(V \cdot V + E \cdot 1)$ ← decreaseKey

② $E = O(V^2)$

Assume that we need to find an MST in a dense graph using Prim's algorithm. Which implementation (heap or array) shall we use?

Binary heap: $O(V^2 \log V)$

array: $O(V^2)$

Discussion T/F Questions

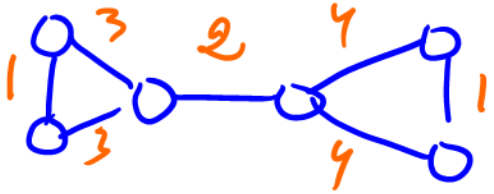
Break

Proof by example.

([✓ T]/F) The first edge added by Kruskal's algorithm can be the last edge added by Prim's algorithm.

Proof by example

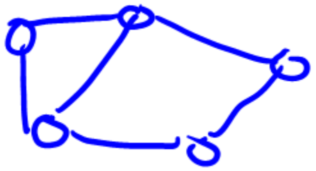
(T/[✓ F]) Suppose we have a graph where each edge weight value appears at most twice. Then, there are at most two minimum spanning trees in this graph.

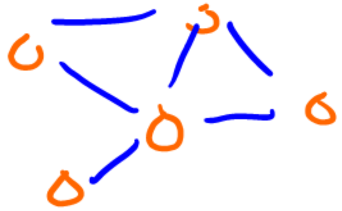


how many MSTs? 4

([✓ T]/F) If a connected undirected graph $G = (V, E)$ has $V + 1$ edges, we can find the minimum spanning tree of G in $O(V)$ runtime.

$V = 5$
$E = 6$

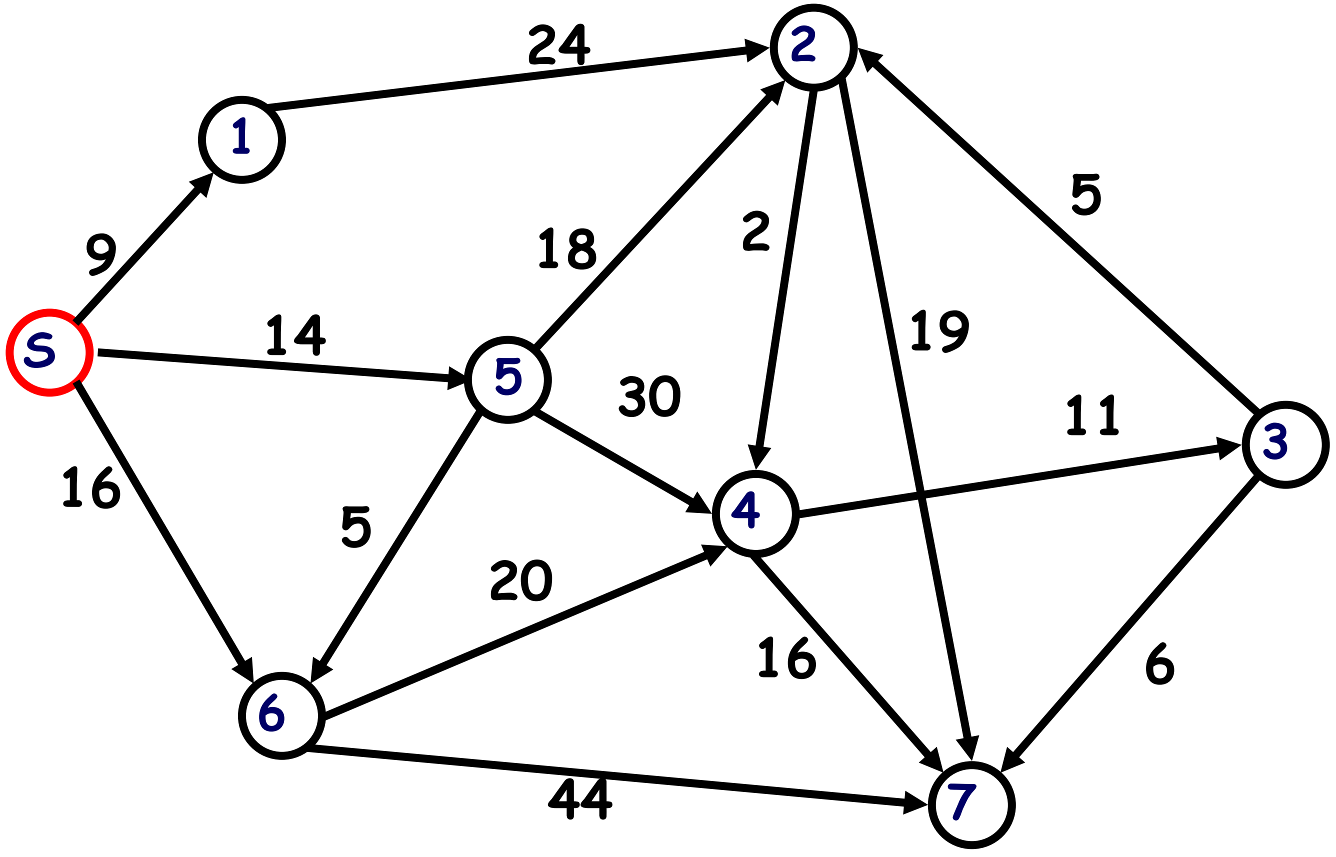




The Shortest Path Problem

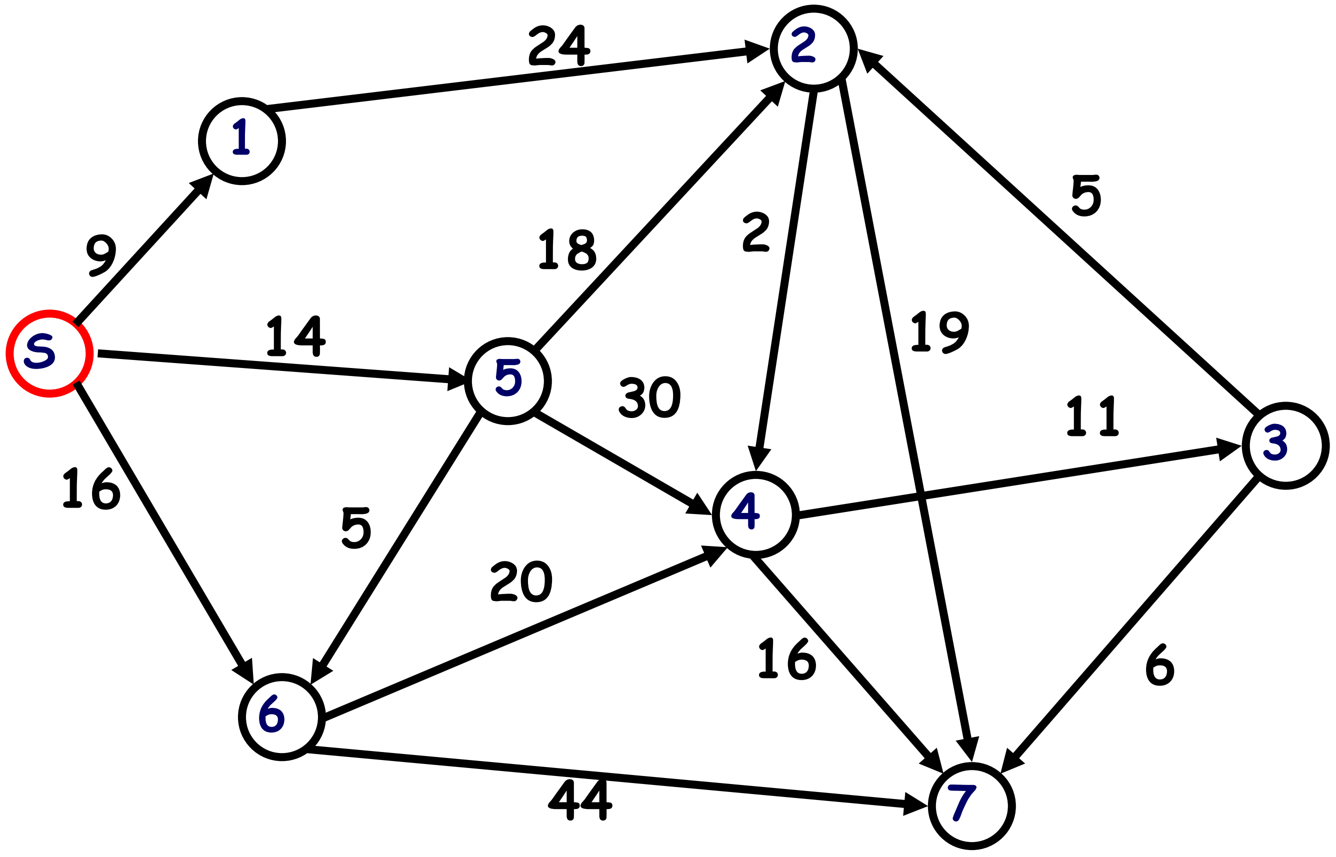
Given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.

SSSP



The Shortest Path Problem

What is the shortest distance from s to 4?

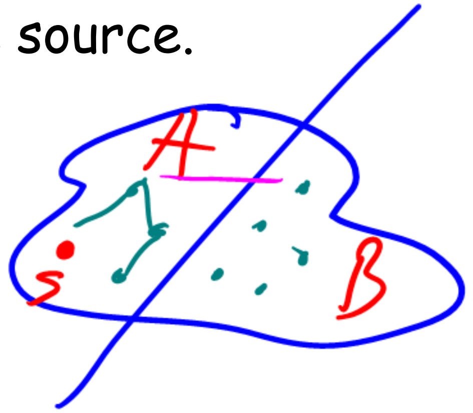


Greedy Approach

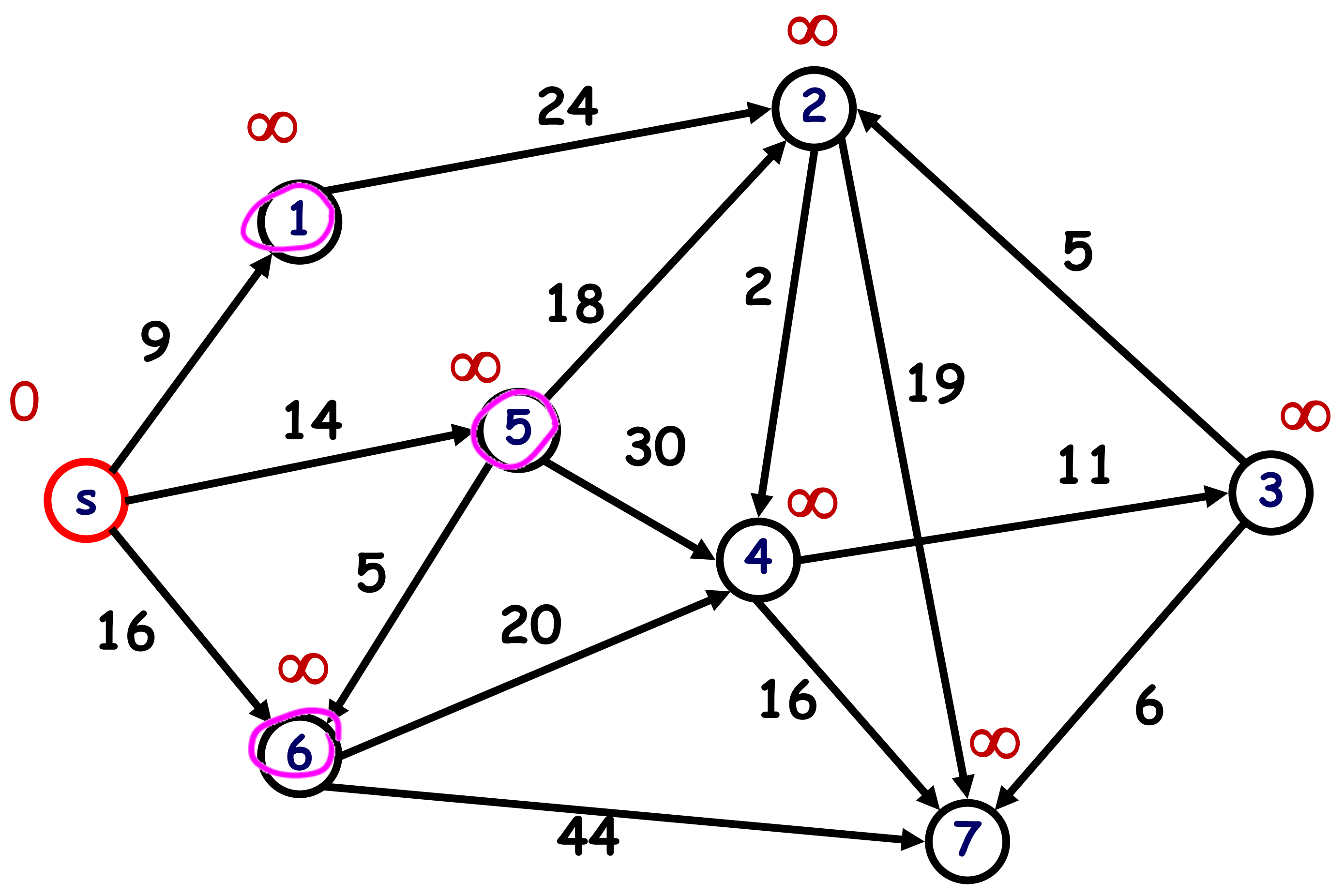
When algorithm proceeds all vertices are divided into two groups
- vertices whose shortest path from the source is known
- vertices whose shortest path from the source is NOT discovered yet.

Move vertices one at a time from the undiscovered set of vertices to the known set of the shortest distances, based on the shortest distance from the source.

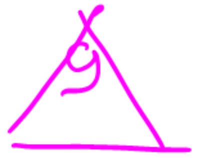
$G = A + B$

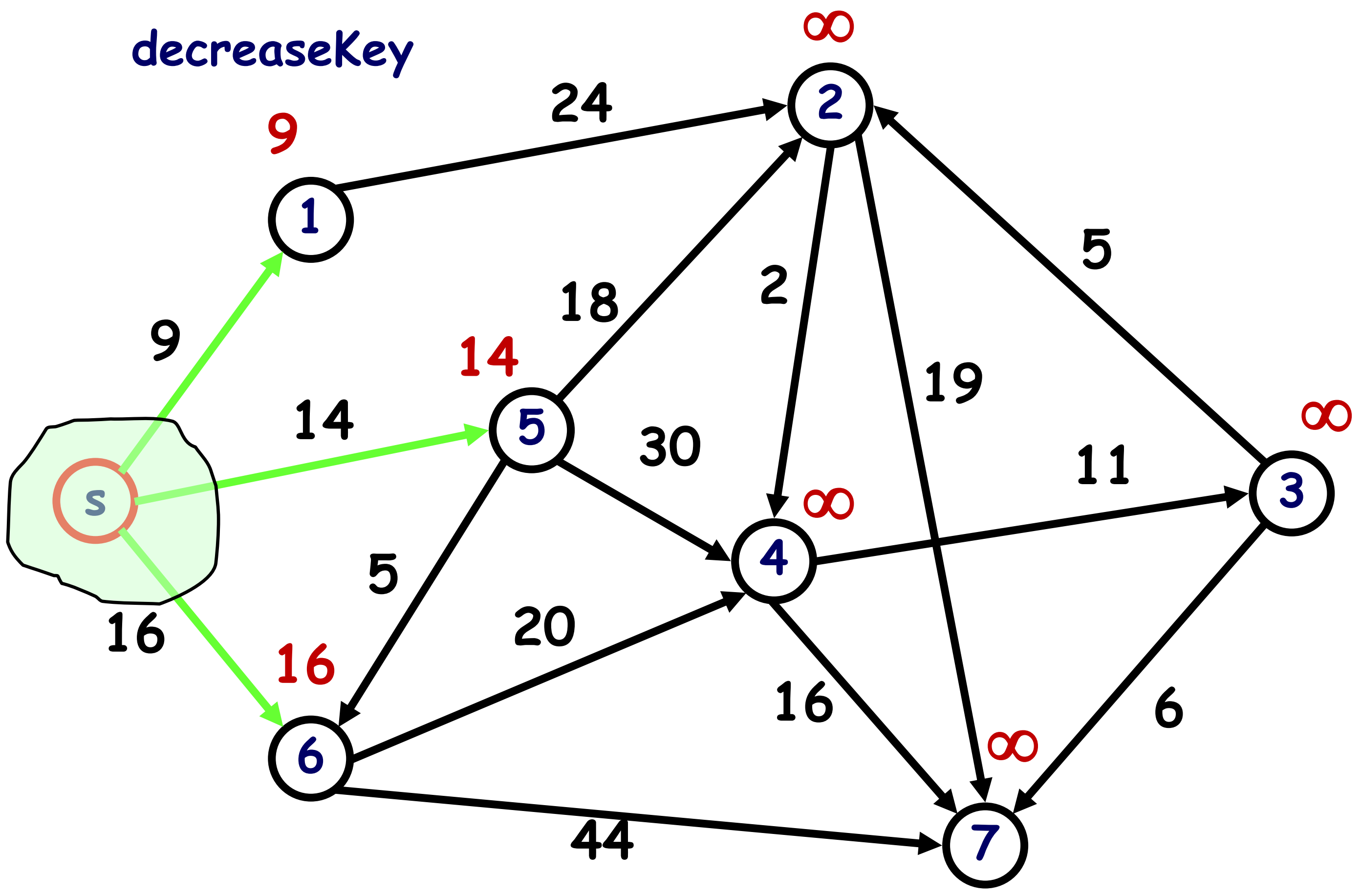


solution tree = s
heap = {1, 2, 3, 4, 5, 6, 7}

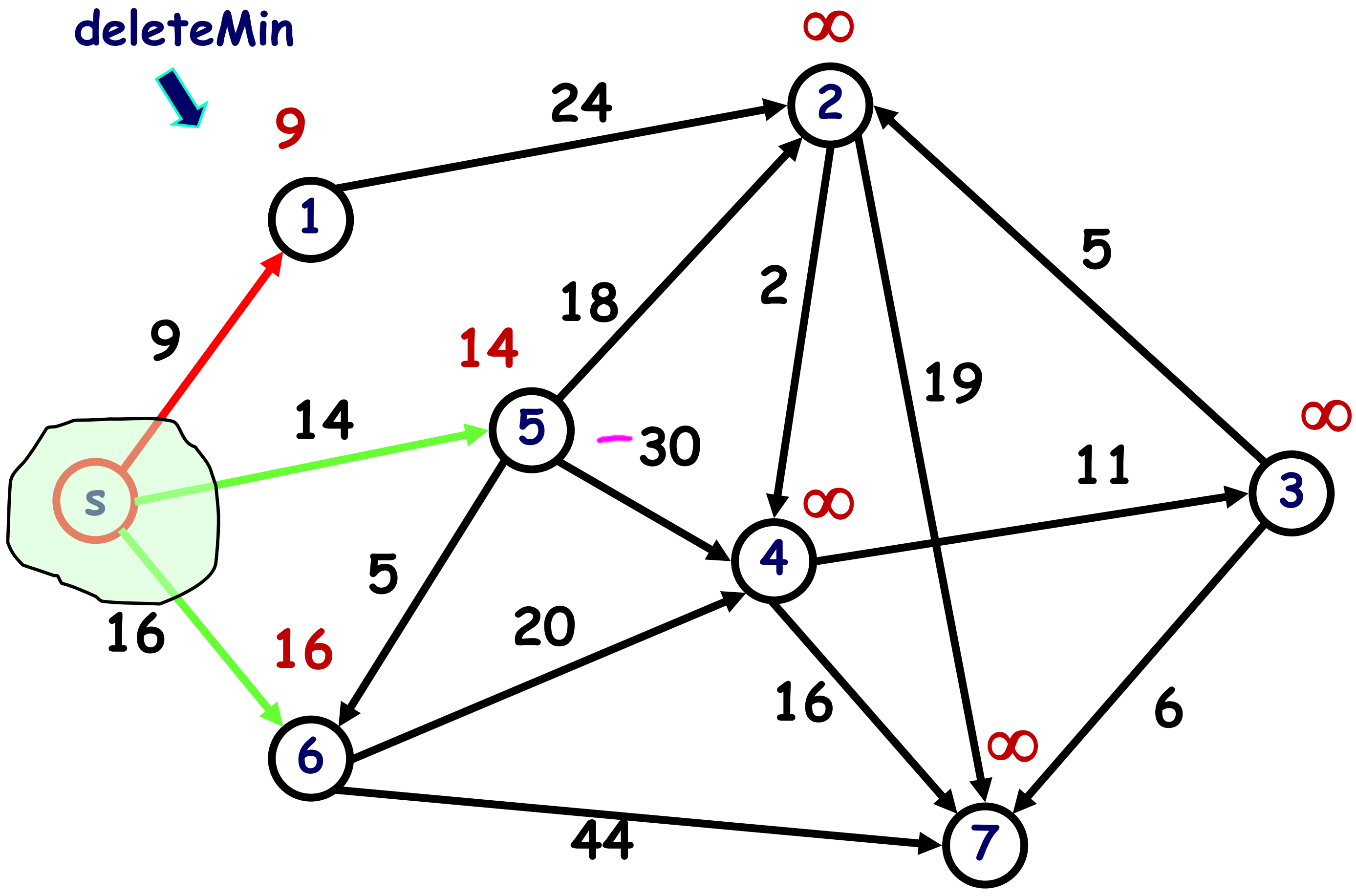


solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}

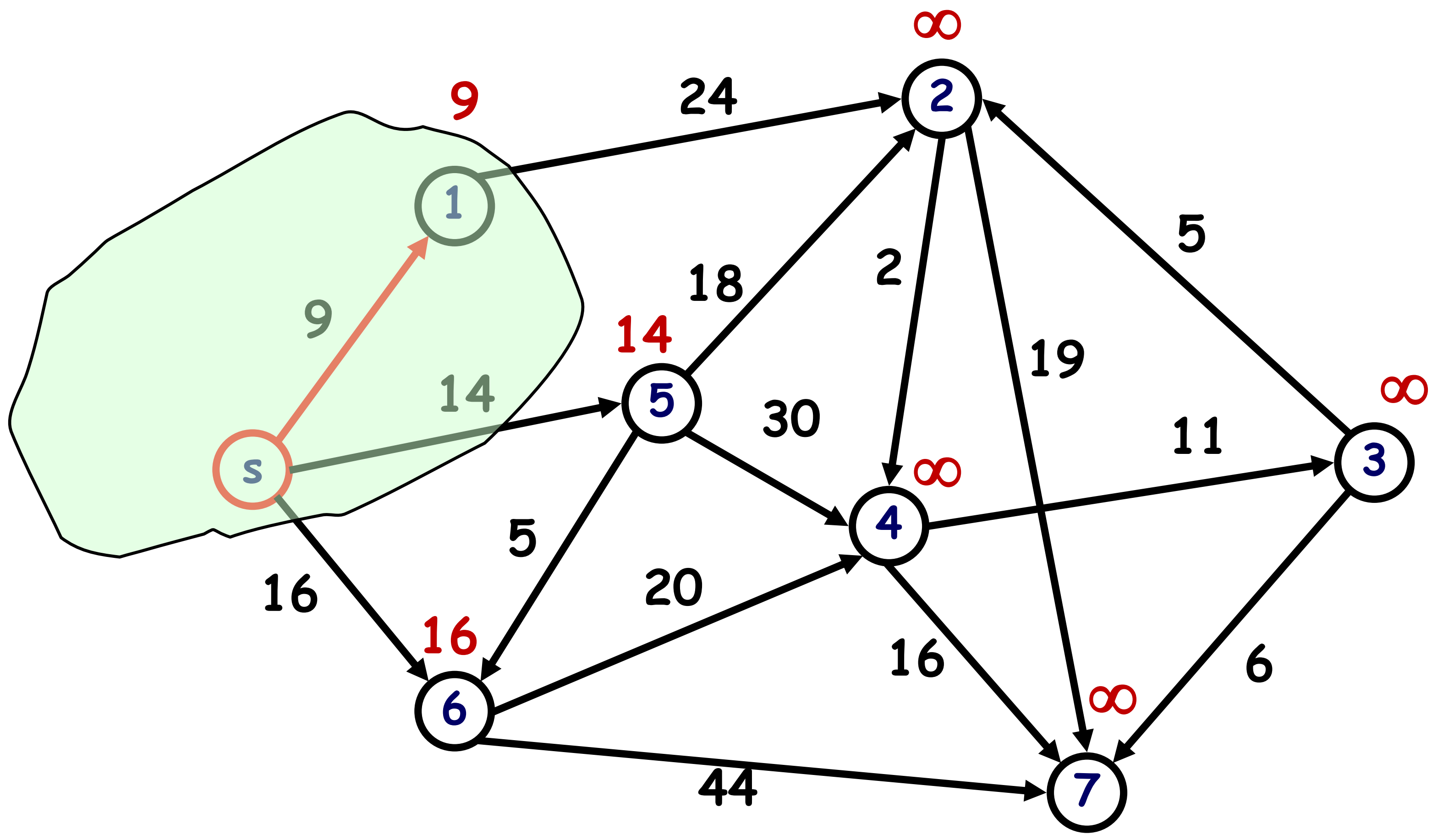




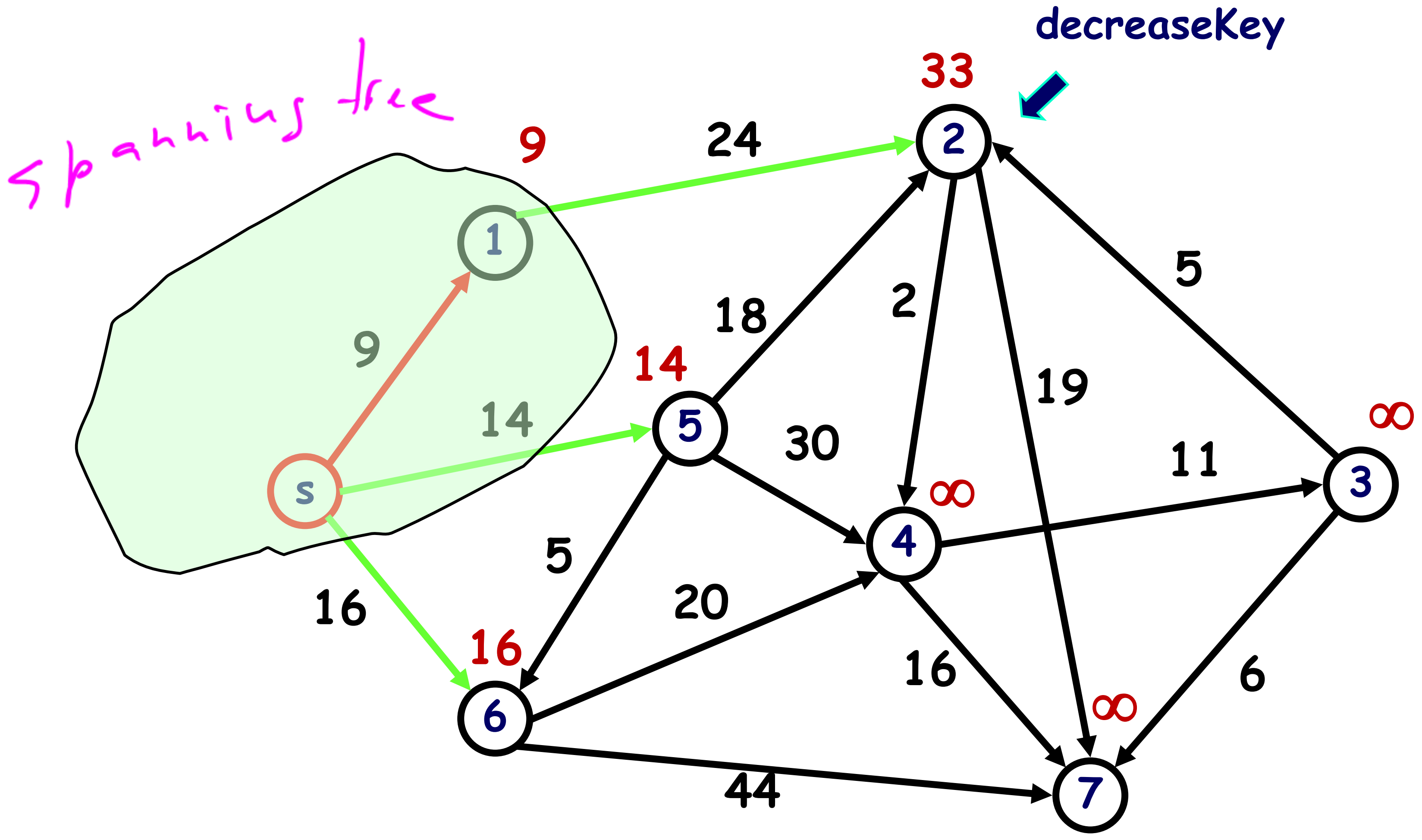
solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}



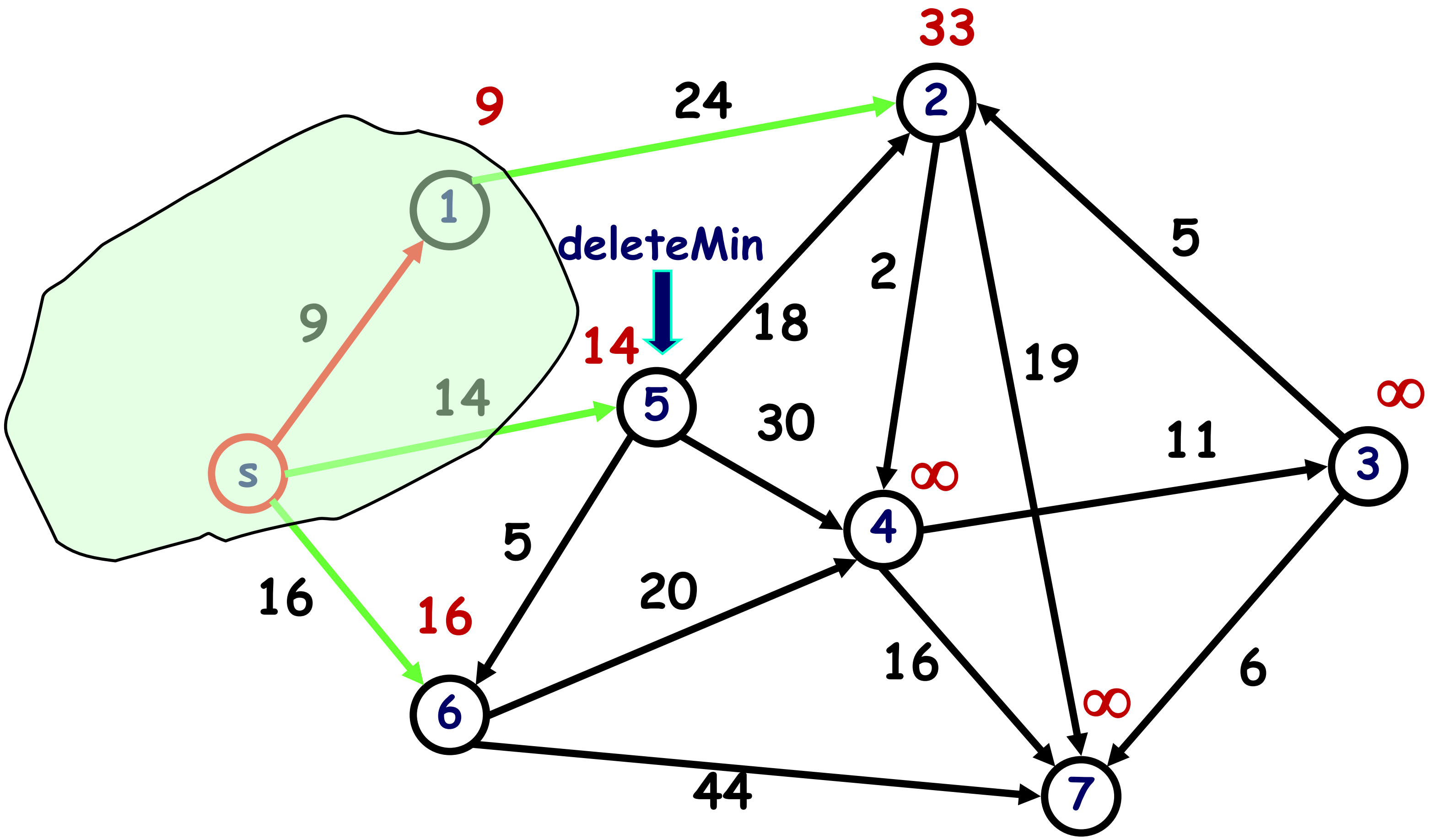
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



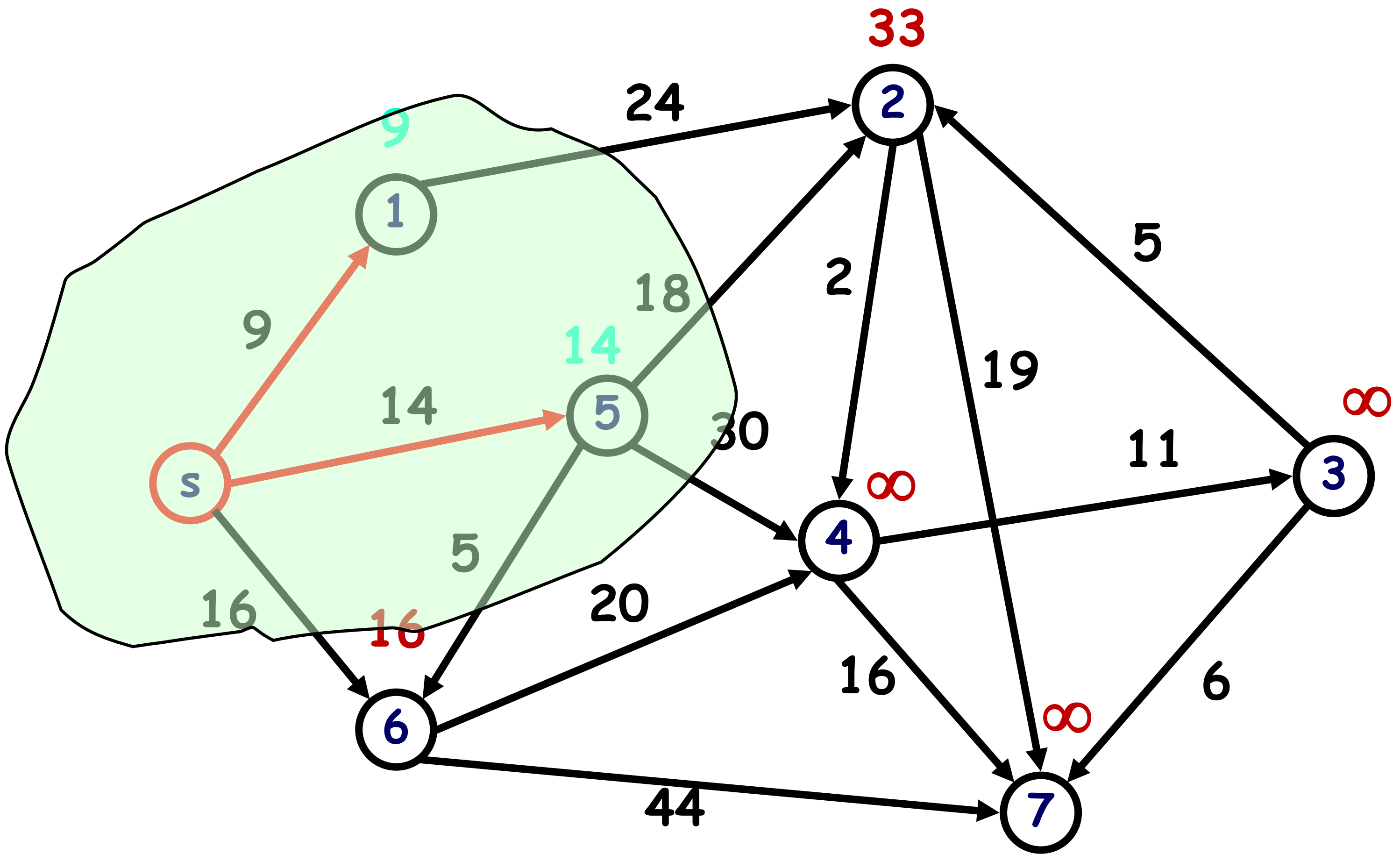
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



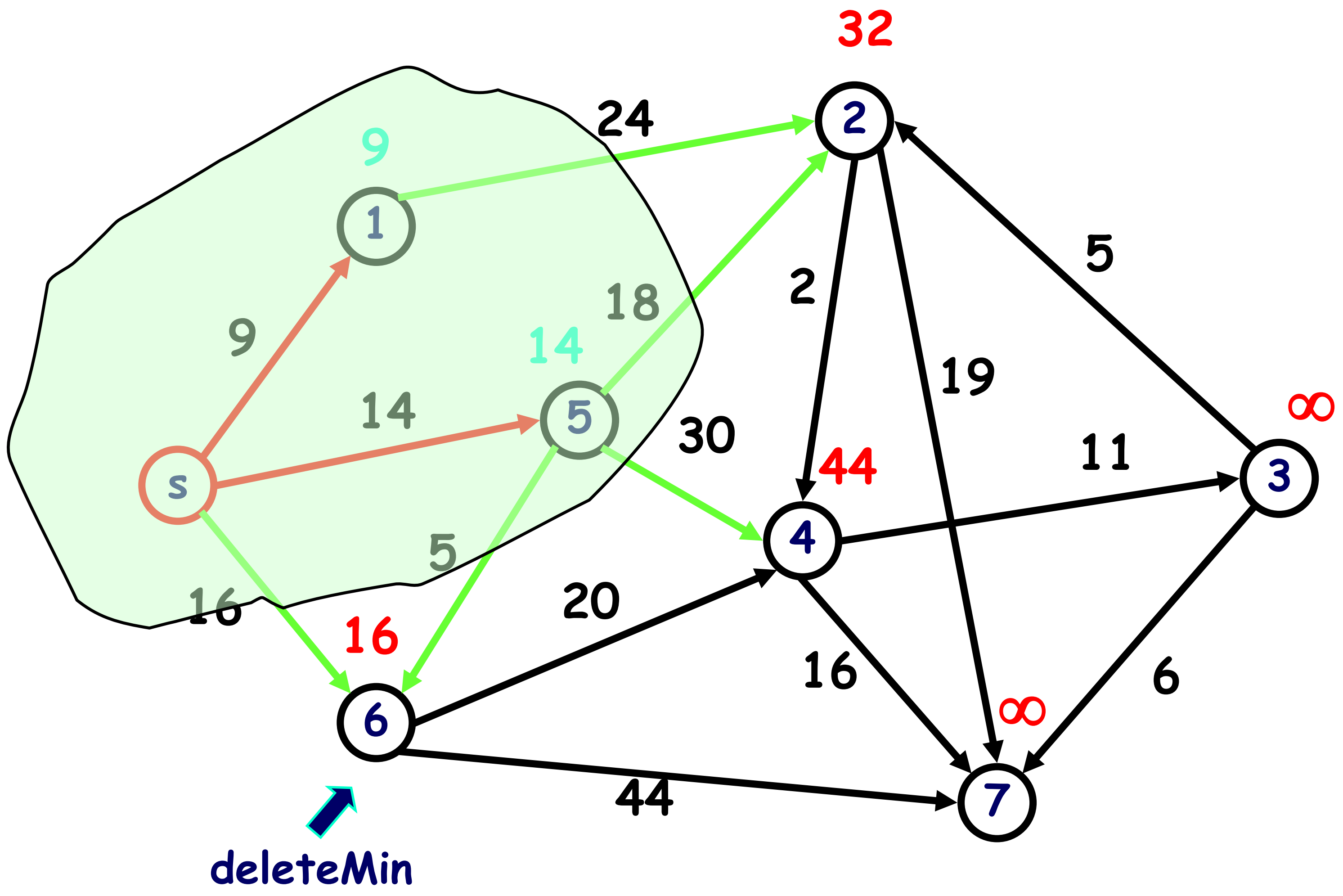
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



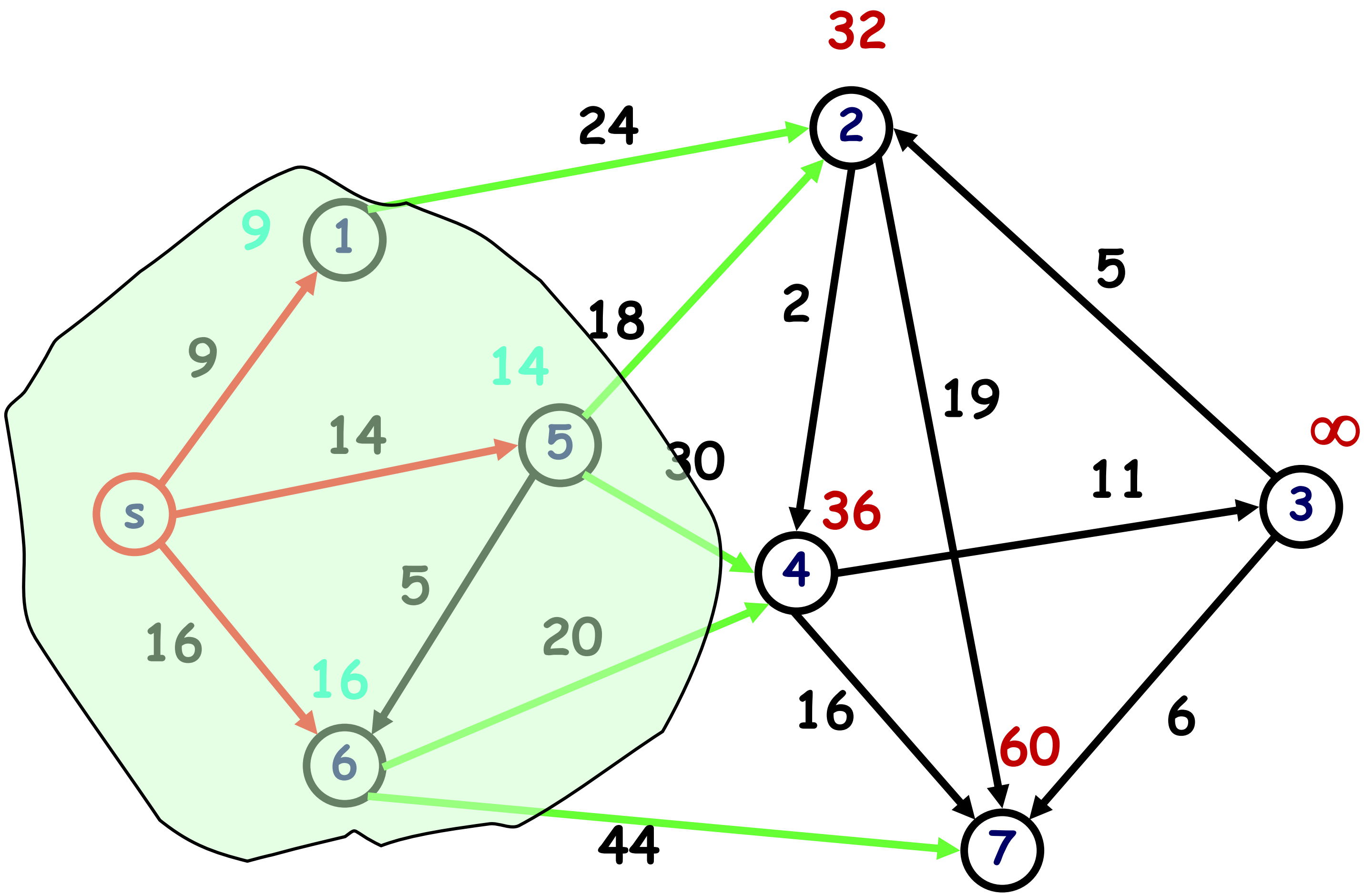
solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}



solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}

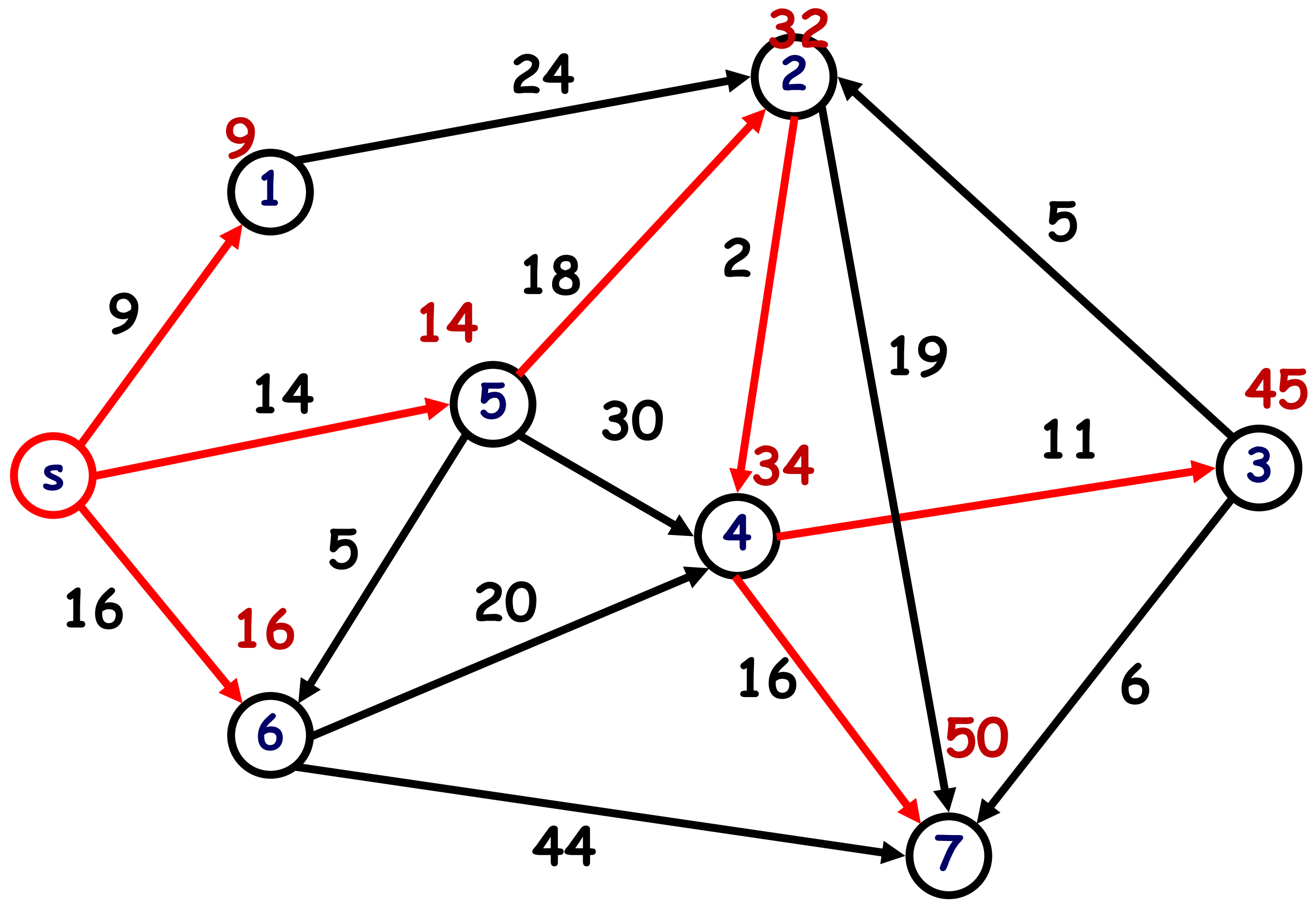


solution tree = { s, 1, 5, 6 }
heap = {2, 3, 4, 7}



solution tree = { s, 1, 5, 6, 2, 4, 3, 7 }
heap = {}

spanning tree ≠ MST



Runtime Complexity

Let D(v) denote a length from the source s to any vertex v.
We store distances D(v) in a binary heap.

Loop:

① deleteMin, V times

② decreaseKey (update distances), E times

$O(V \cdot \log V + E \cdot \log V)$

Discussion Problem 5

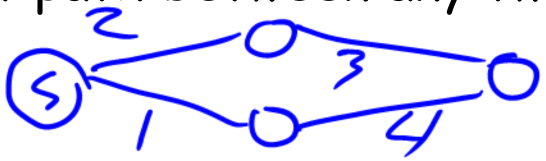
Assume that an unsorted array is used instead of a binary heap. What would the running time of the Dijkstra algorithm?

$$O(V \cdot V + E \cdot 1)$$

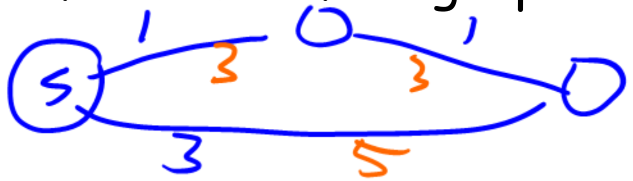
Discussion T/F Questions

Proof

(T/[✓ F]) If all edges in a connected undirected graph have distinct positive weights, the shortest path between any two vertices is unique.



(T/[✓ F]) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph, G, such that weights of all edges are increased by 2, then the shortest path tree of G is also the shortest path tree of the modified graph.



([✓ T]/F) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph G such that weights of all edges are doubled, then the shortest path tree of G is also the shortest path tree of the modified graph.

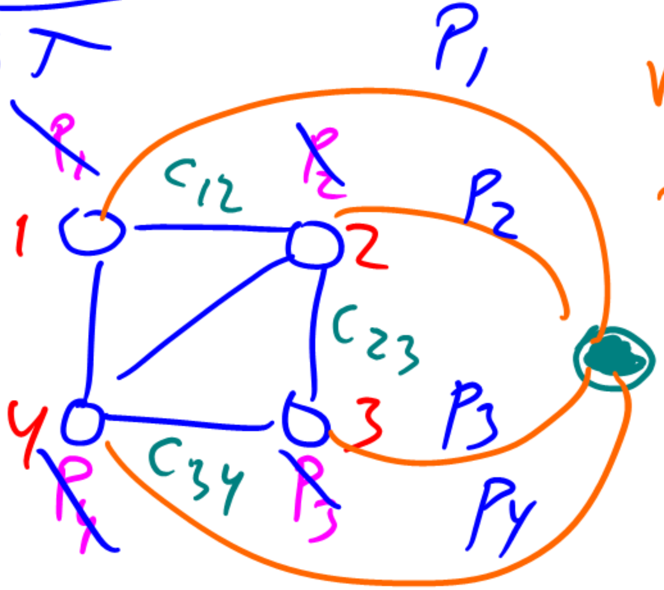
$x+y+z < a+b$

$2x+2y+2z < 2a+2b$

Discussion Problem 8

In this problem you are to find the most efficient way to deliver power to a network of n cities. It costs p_i to open up a power plant at city i. It costs $c_{ij} \geq 0$ to put a cable between cities i and j. A city is said to have power if either it has a power plant, or it is connected by a series of cables to some other city with a power plant. Devise an efficient algorithm for finding the *minimum cost* to power all the cities.

MST



remove weights P_i
from vertices ..

Add a new vertex
Connect it to all other vertices
Run MST algorithm.

? does our solution have at least one power plant?

Guidelines solving problems
- change the input
- design a new algorithm