Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 4

University of Southern California

Spring 2023

Greedy Algorithms

Reading: chapter 4

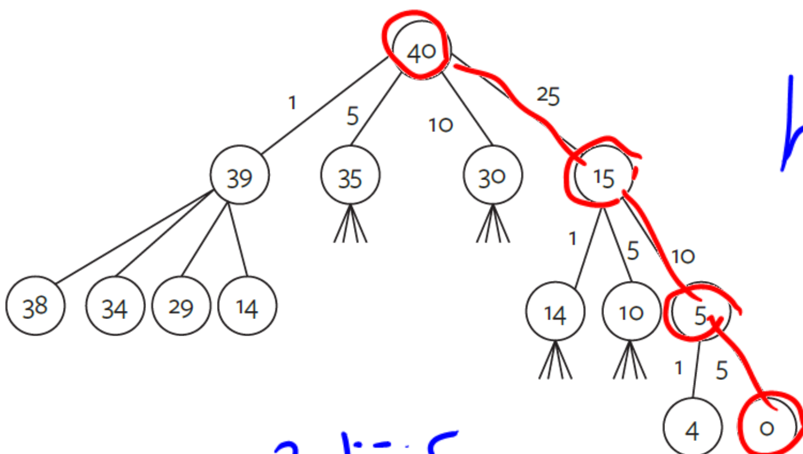
Heaps for Priority Queue

	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)

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,



h

greedy algo:

$O(h)$

PQ:

heap

Run BFS

$O(4^h)$ exponential

$40 = 25 + 10 + 5$

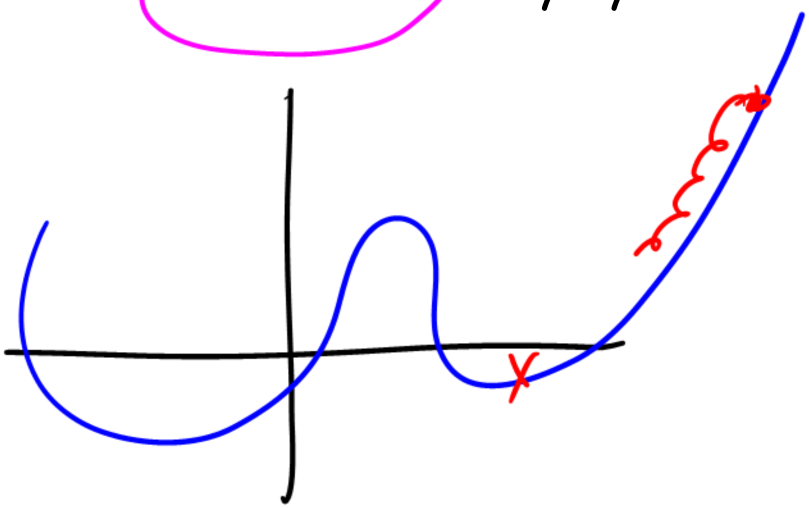
During the algorithm execution, we don't consider all available choices at any given point, but use a <u>heuristic</u> (greedy choice) to pick just one.

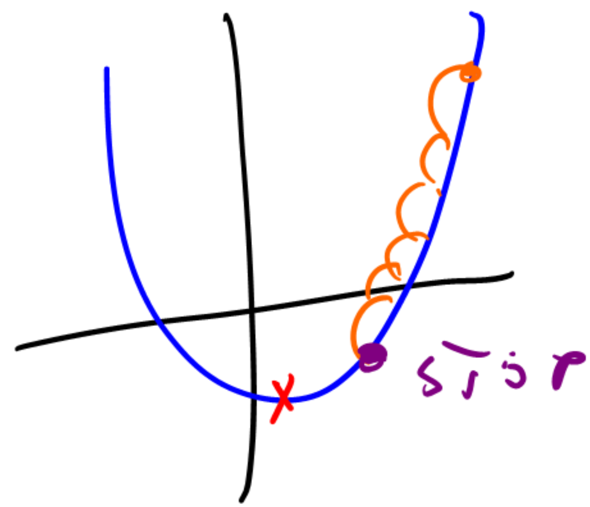
Counterexample: suppose 1, 5, 10, 20, 25, ..

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

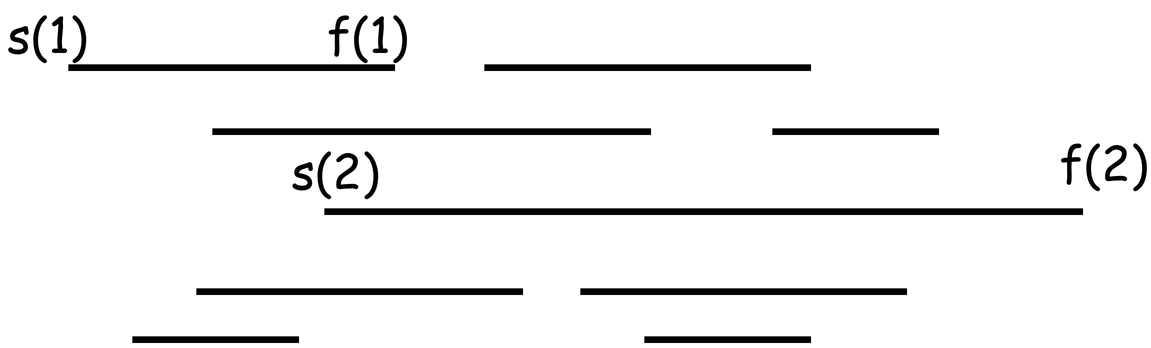
optimal substructure.

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 $s(i) \leq f(i)$. Our goal is to develop a greedy algorithm that finds the <u>largest</u> compatible (non-overlapping) subset of requests.



How do we choose requests?

1. Sort by $s(i)$, choose the smallest

ALG: A
OPT: B & C

2. Sort by $f(i) - s(i)$

ALG: C
OPT: A & B

Proof

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

Proof.

ALG: $i_1, i_2, \ldots, i_k$

OPT: $j_1, j_2, \ldots, j_m$

Induction on the output

Prove: $f(i_r) \leq f(j_r)$

Base case: $r = 1$ $\quad f(i_1) \leq f(j_1)$

IH: $f(i_{r-1}) \leq f(j_{r-1})$ $\quad$ the earliest finish time

IS: prove it for the next request

$f(i_{r-1}) \le f(j_{r-1}) \le s(j_r)$

by IH ? cannot overlap

What does this mean?

$\exists$ one more request: j_r

it means that ALG will pick that request

$f(i_r) \le f(j_r)$

Next step: prove $k = m$.

Proof by contradiction. comp.

Assume $k < m$.

① $f(j_k) \le s(j_{k+1})$

② $f(i_k) \le f(j_k)$, IH

Combine: $f(i_k) \le s(j_{k+1})$

it means that j_{k+1} request does not overlap with i_k request.

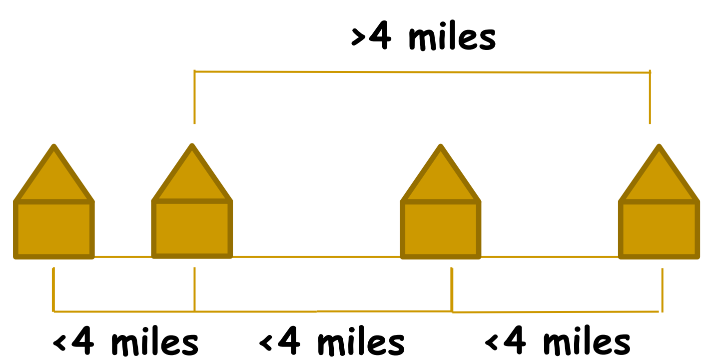
it follows, ALG will choose j_{k+1}.

Contradiction.

Break!

Discussion Problem 1

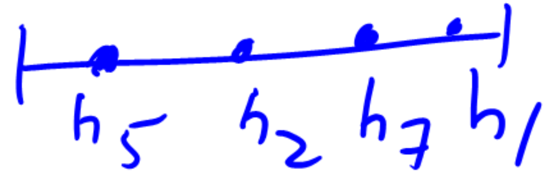
Let's consider a long, quiet country road with n houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. You want to place cell phone base stations at certain points along the road so that every house is within four miles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



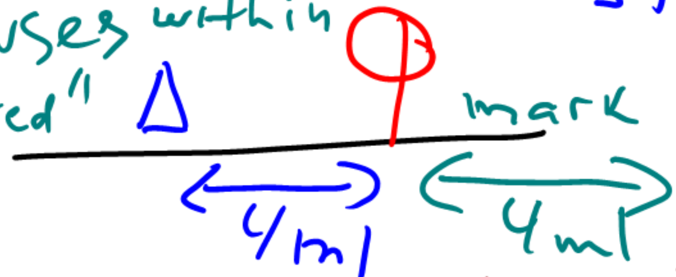
Algorithm: input: array of houses h_K

(1) sort h_K, from W to E

(2) ~~go~~ go to the first house then walk 4 mi, put a station



Mark all houses within 4mi as "covered"



Complexity: $O(n \log n)$

Proof of the correctness:

ALG: $s_1, s_2, \dots, s_K$

Induction on houses OPT: $t_1, t_2, \dots, t_m$

Base case:

IH: assume that is true for $(c-1)$ houses

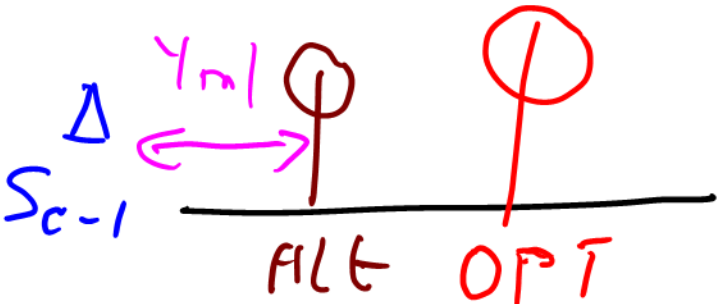
IS: prove it for c houses

what do we have?

ALG: $s_1, s_2, \ldots, s_{c-1}, s_c$

Δ

(1)



impossible

(2)

The Minimum Spanning Tree

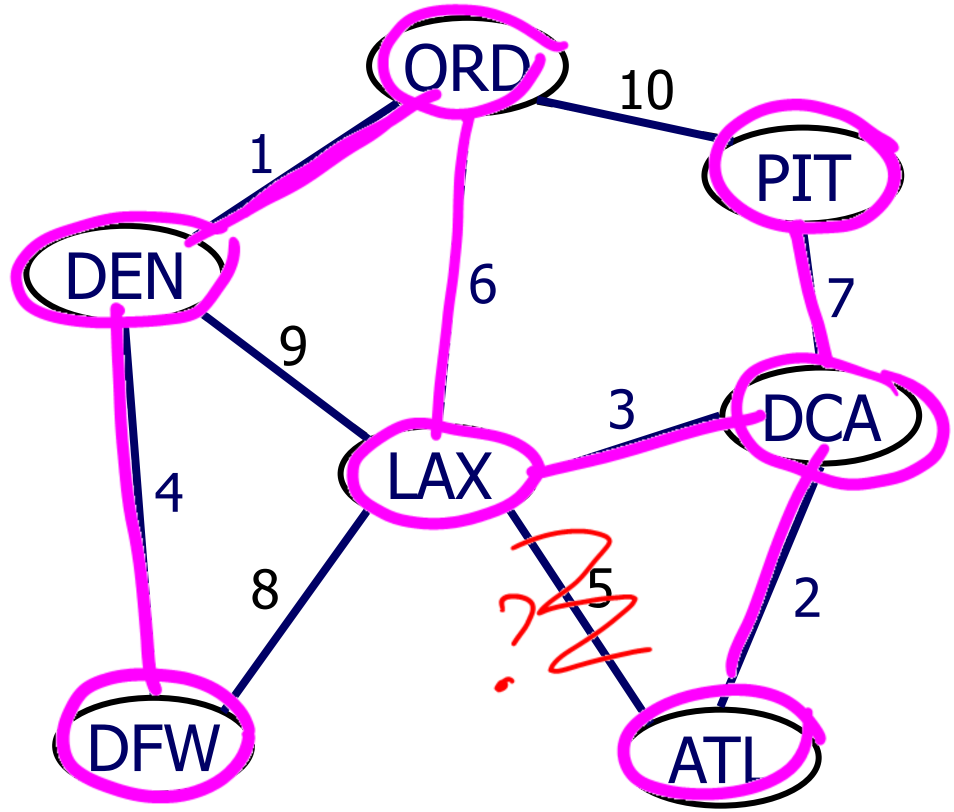
Given a weighted undirected graph. Find a spanning tree of the minimum total weight.

MST is fundamental problem with diverse applications.

The Minimum Spanning Tree

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:

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

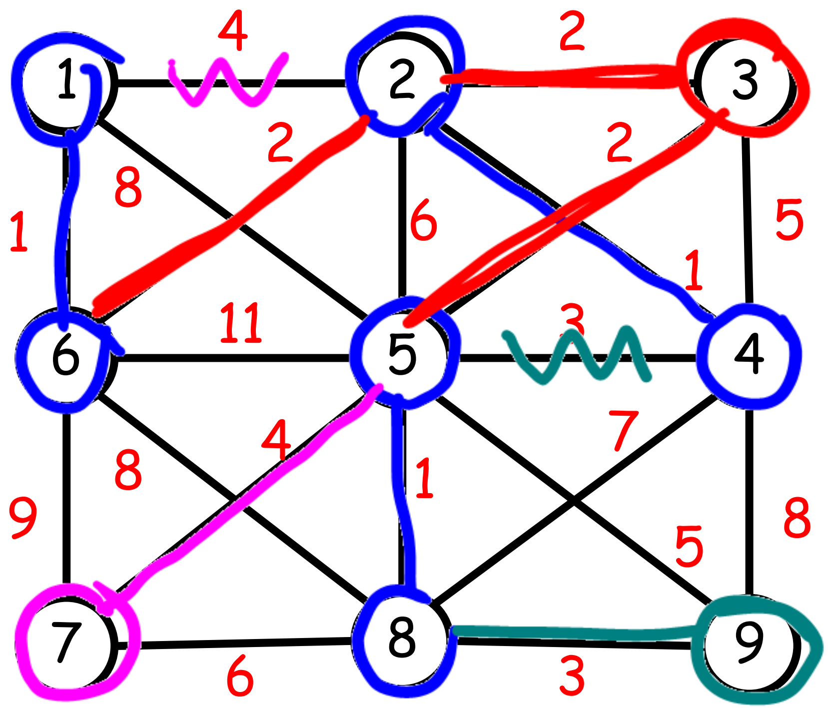
Sorting edges – O(E log E)

Cycle detection – O(V) for each edge

with sorting

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

Kruskal's Algorithm



Exercise: assume that you do not sort edges.

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

Discussion Problem 2

G $\quad a > b > 0 \quad \frac{1}{2} > \frac{1}{3} \quad \frac{1}{4} > \frac{1}{9} \quad$ G_1 $\quad a^2 > b^2 > 0$

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.

$MST(G) = T \qquad G_1: c \to c^2$

$MST(G_1) = T_1$

Question: $T = T_1$

Cases:

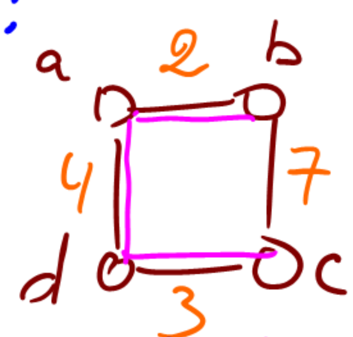
a) $c_e \geq 0$, then $T = T_1$. Prove it!

b) $\exists c_k < 0$, 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.

G:

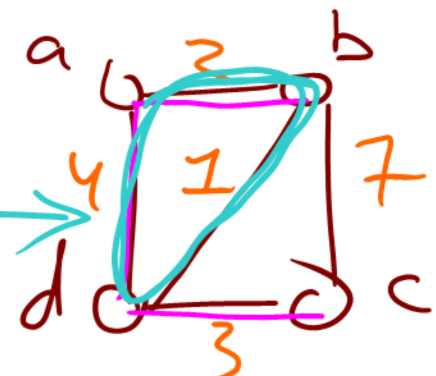


MST(G) = T

① add edge to T
we will get a cycle in T

② traverse that cycle
and delete the largest edge

G_1

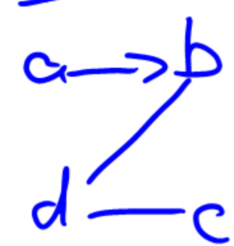


MST(G_1) = T_1

do not run Kruskal

remove edg (a, d)

T_1:



Prim's Algorithm

The algorithm builds a tree one VERTEX at a time:

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

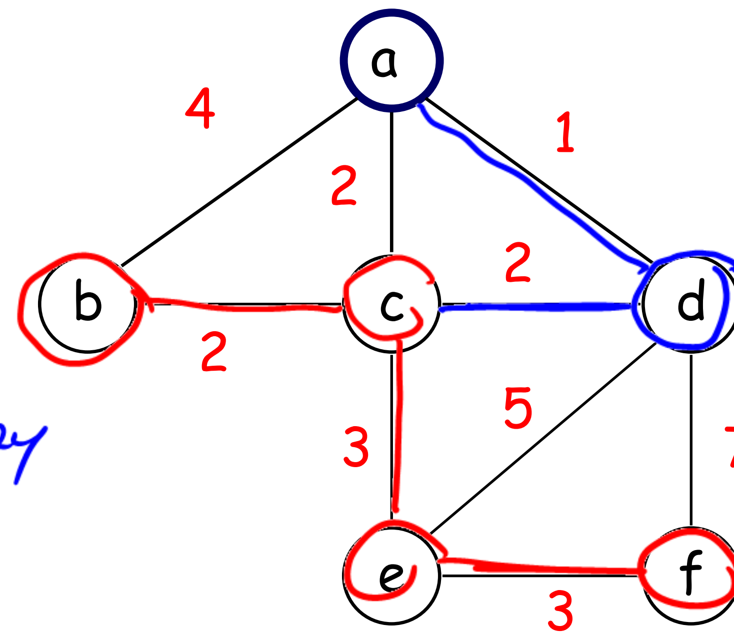
- Expand *C* by adding a vertex having the minimum weight edge of the graph having exactly one end point in *C*.

- Update distances from *C* to adjacent vertices.

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

Prim's Algorithm: Example

pick vertex a



T={a}

Heap: distances from T to all vertices

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

heap.deleteMin

decreaseKey

T={a, d}

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

deleteMin

T={a, d, c}

b-2	e-3	f- 7

decreaseKey (update)

Complexity of Prim's Algorithm

Algorithm:

1 deleteMin $O(\log V)$, run V times

2. (update\decreaseKey $O(\log V)$, run E times

Runtime: $O(V \log V + E \log V)$ worst-case

Binary Heap

Break.

Assume Fibonacci Heap:

Runtime: $O(V \cdot \log V + E \cdot \underline{1})$ amort. cost

Discussion Problem 4

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

PQ: binary heap – $O(V \log V + E \log V)$

PQ: unsorted array – $O(V \cdot V + E \cdot 1)$

$= E = O(V^2)$

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?

a) heap

b) array

c) I do not know

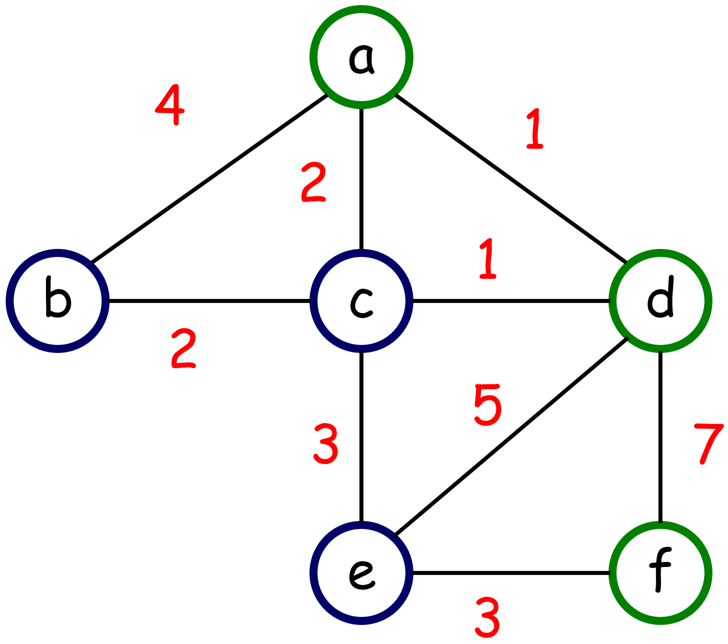
array: $O(V^2)$

bin. heap: $O(V^2 \log V)$

MST: Proof of the correctness

A cut of a graph is a partition of its vertices into two disjoint sets (blue and green vertices below.)

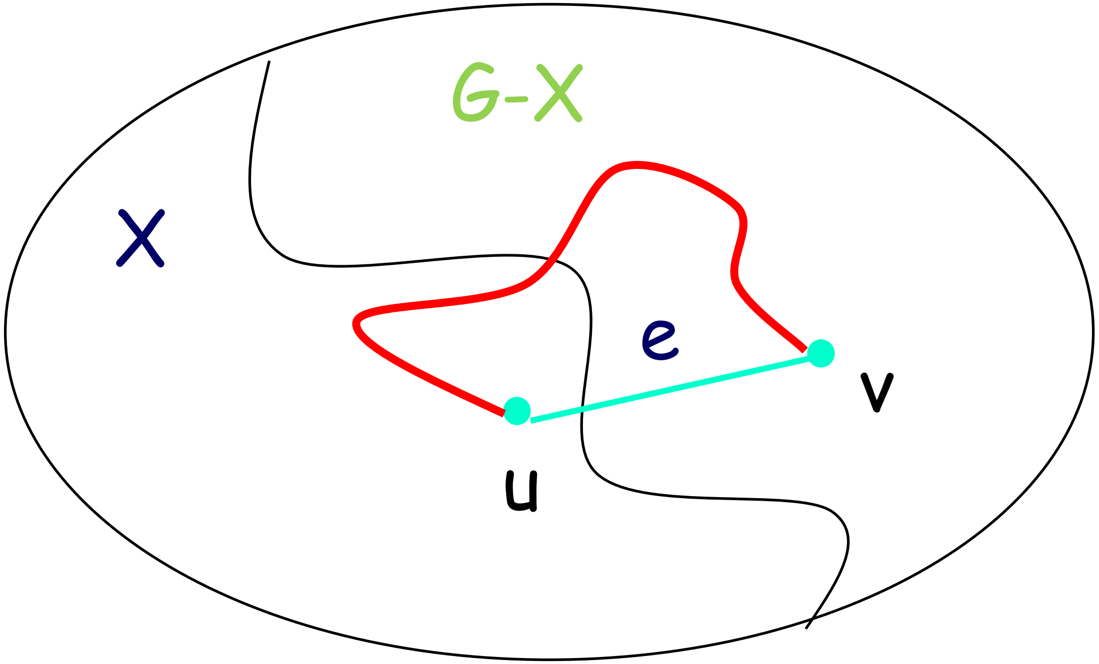
A crossing edge is an edge that connects a vertex in one set with a vertex in the other.



The smallest crossing edge must be in the MST.

MST: Proof of the correctness

Lemma: Given any cut in a weighted graph , the crossing edge of minimum weight is in the MST of the graph.



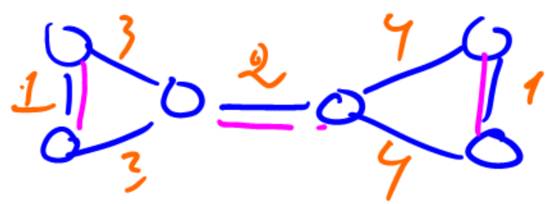
Discussion T/F Questions

proof by example

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

counterexample

(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.

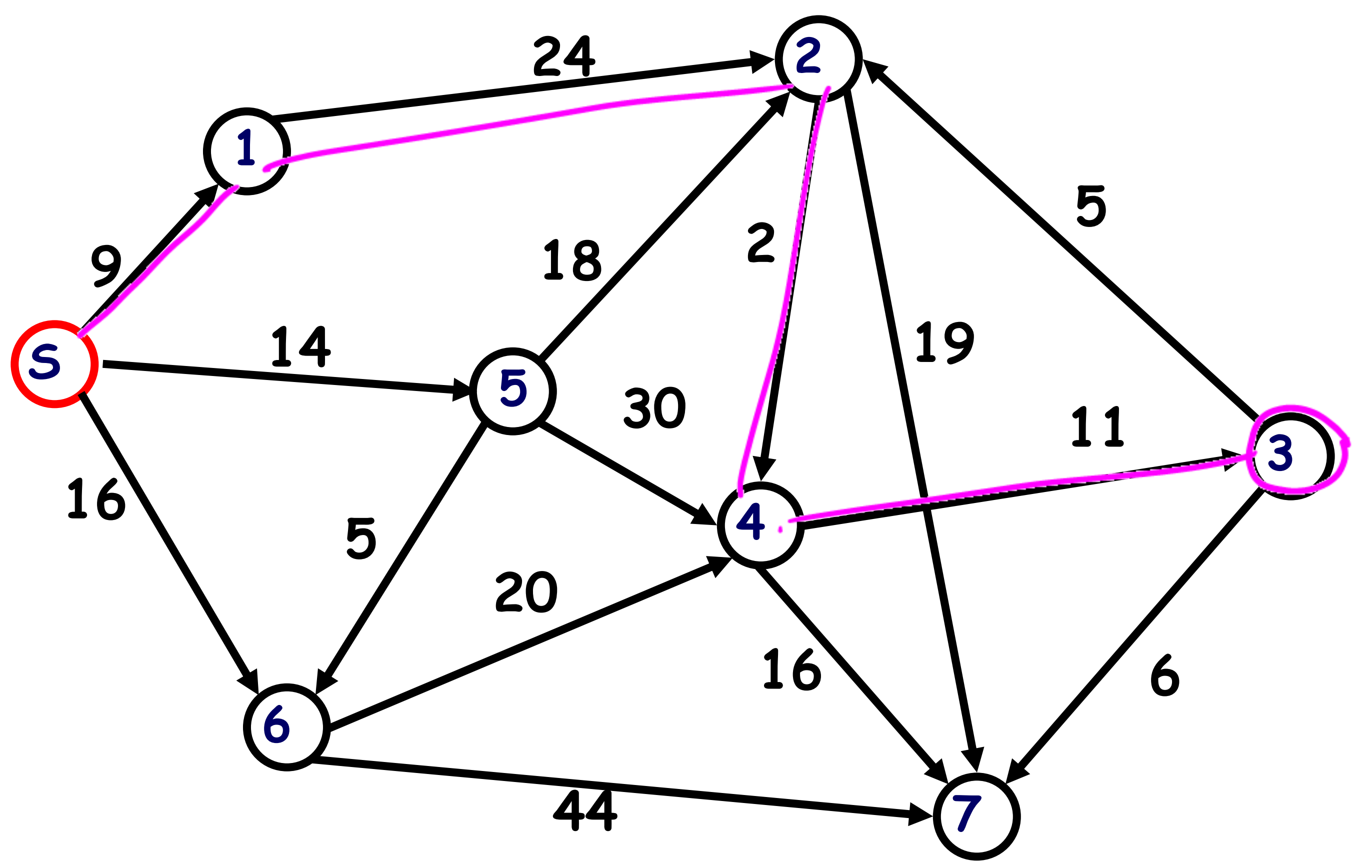


4 MST

(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.

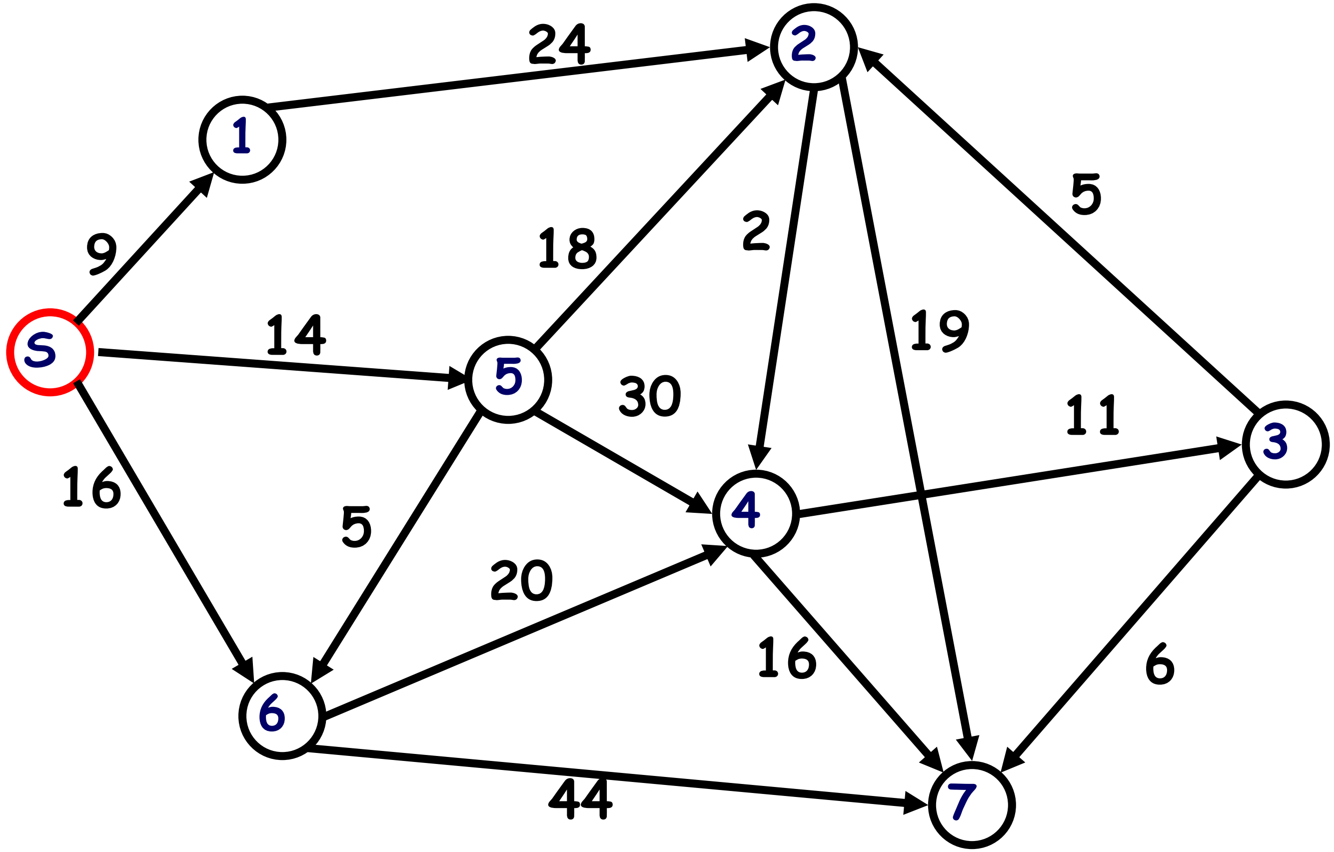
The Shortest Path Problem

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



The Shortest Path Problem

What is the shortest distance from s to 4?



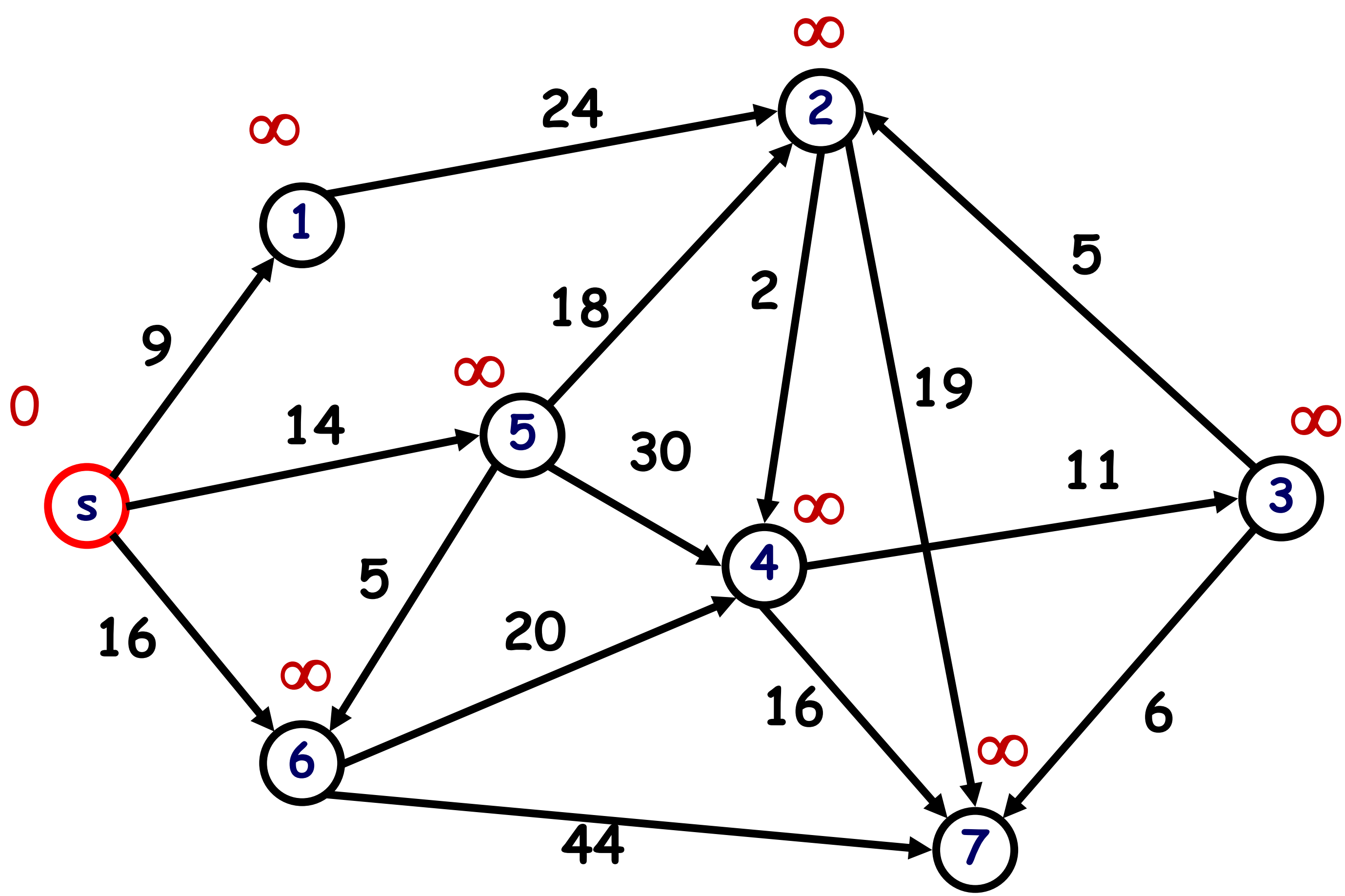
Greedy Approach

When algorithm proceeds all vertices are divided into two groups

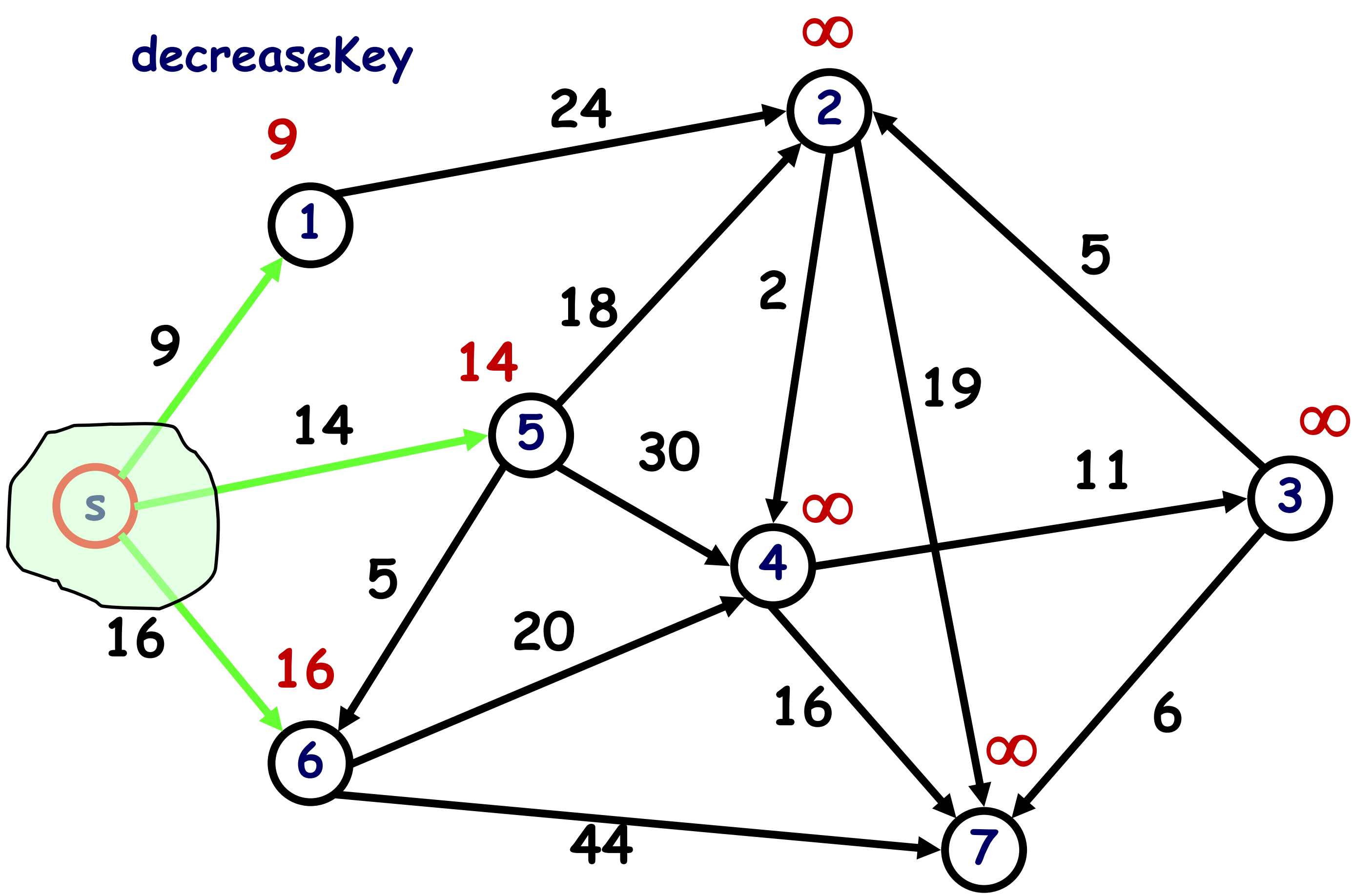
- vertices whose shortest path from the source is known
- vertices whose shortest path from the source is NOT discovered yet.

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.

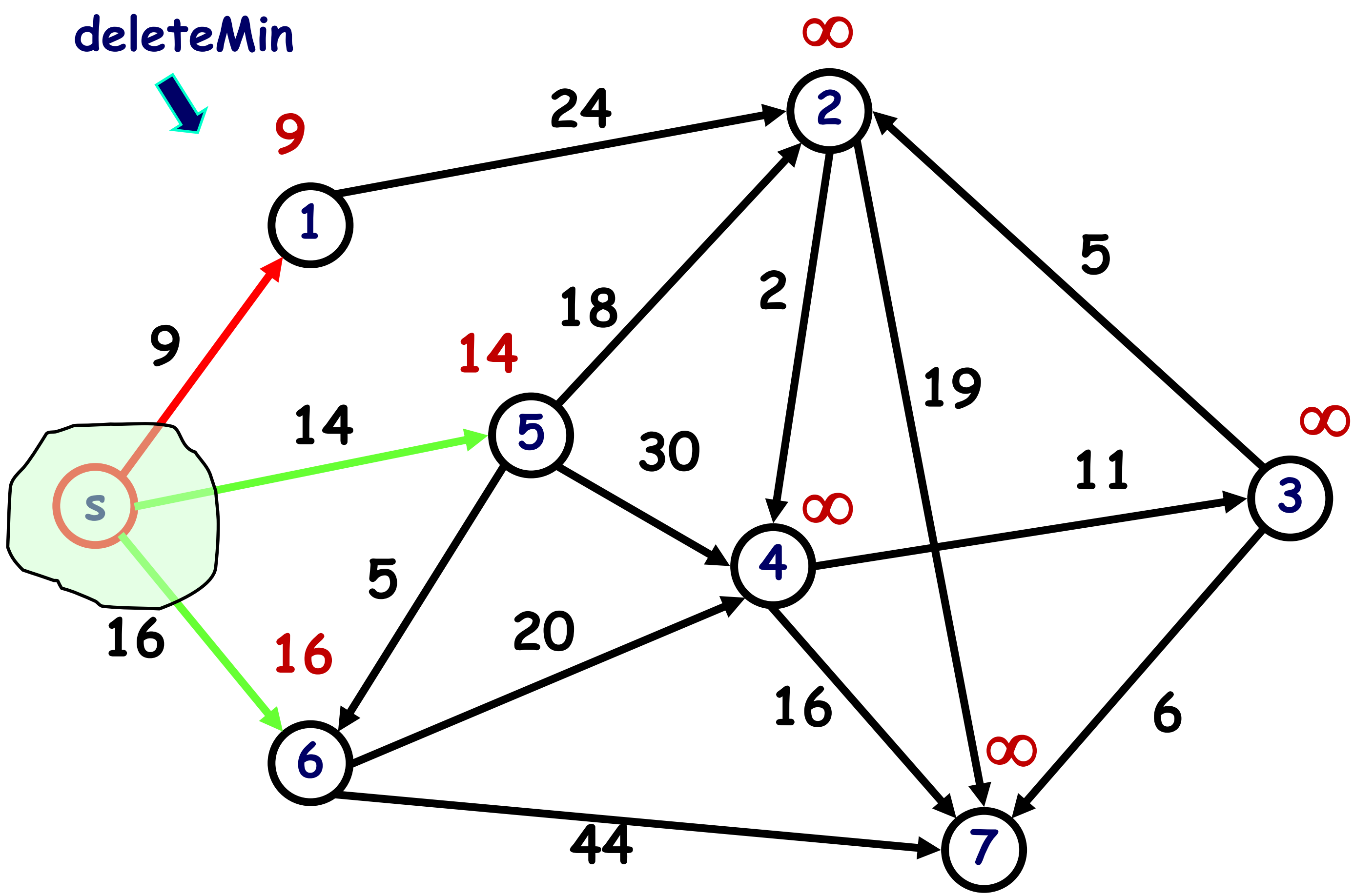
solution tree = s
heap = {1, 2, 3, 4, 5, 6, 7}



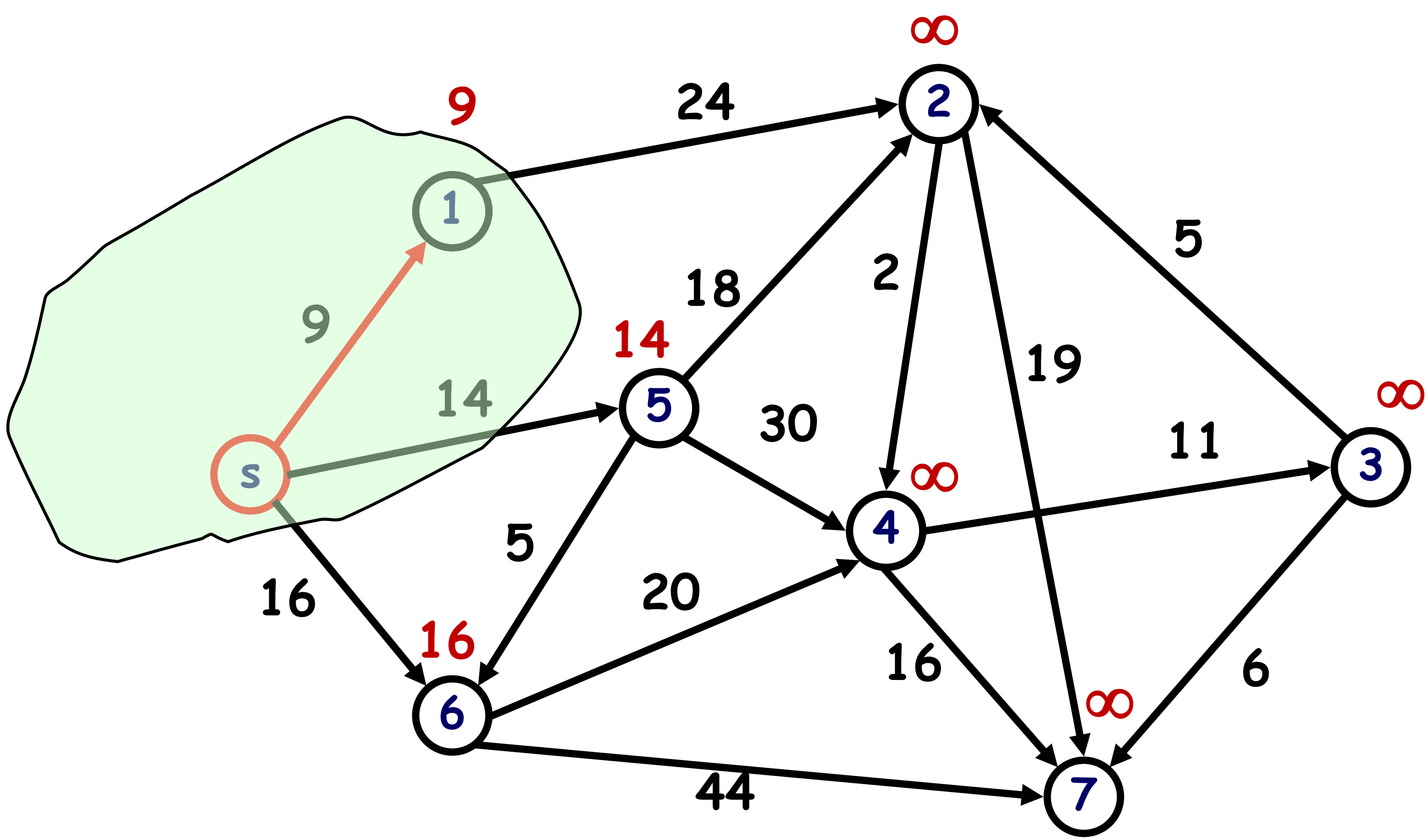
solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}



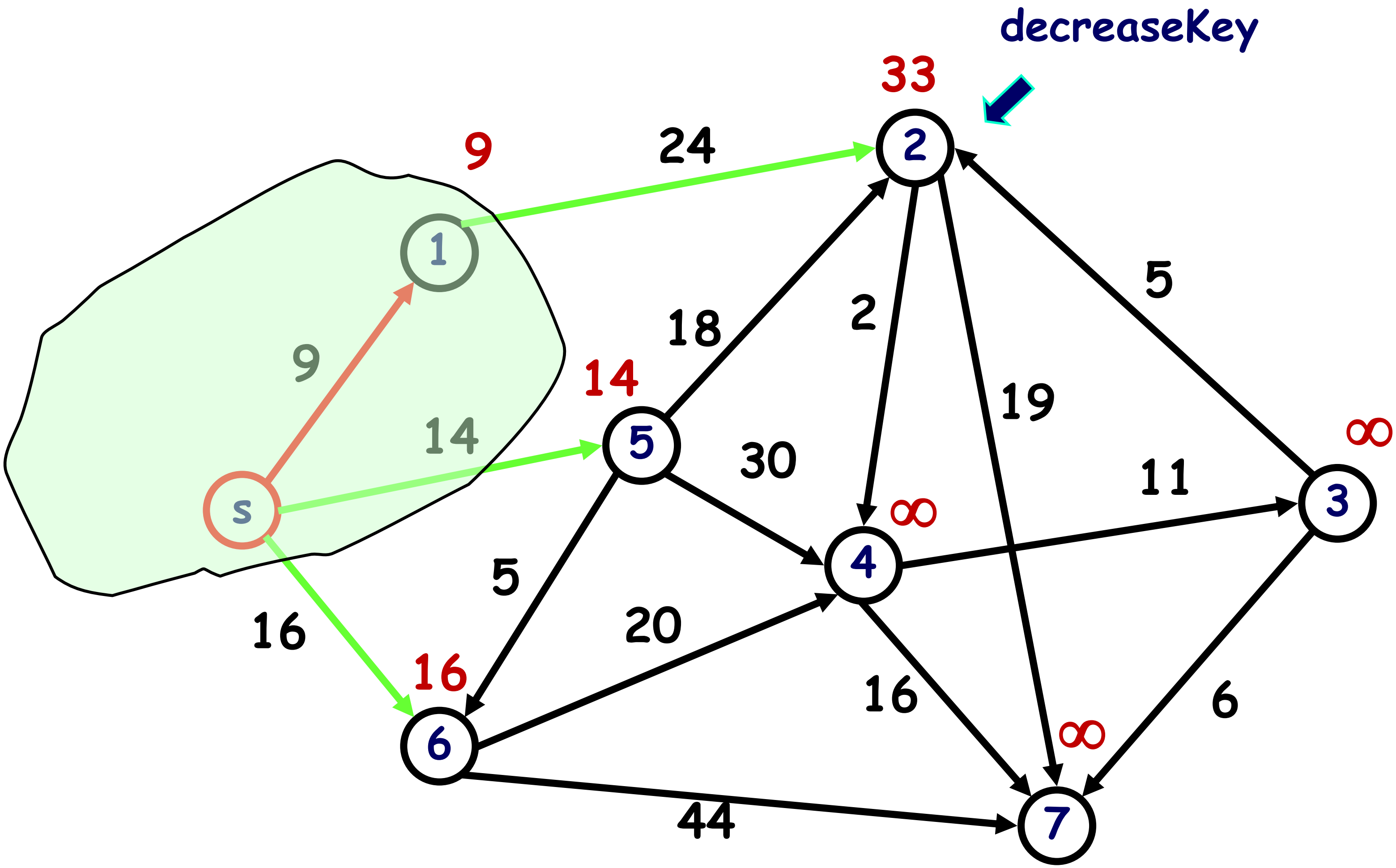
solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}



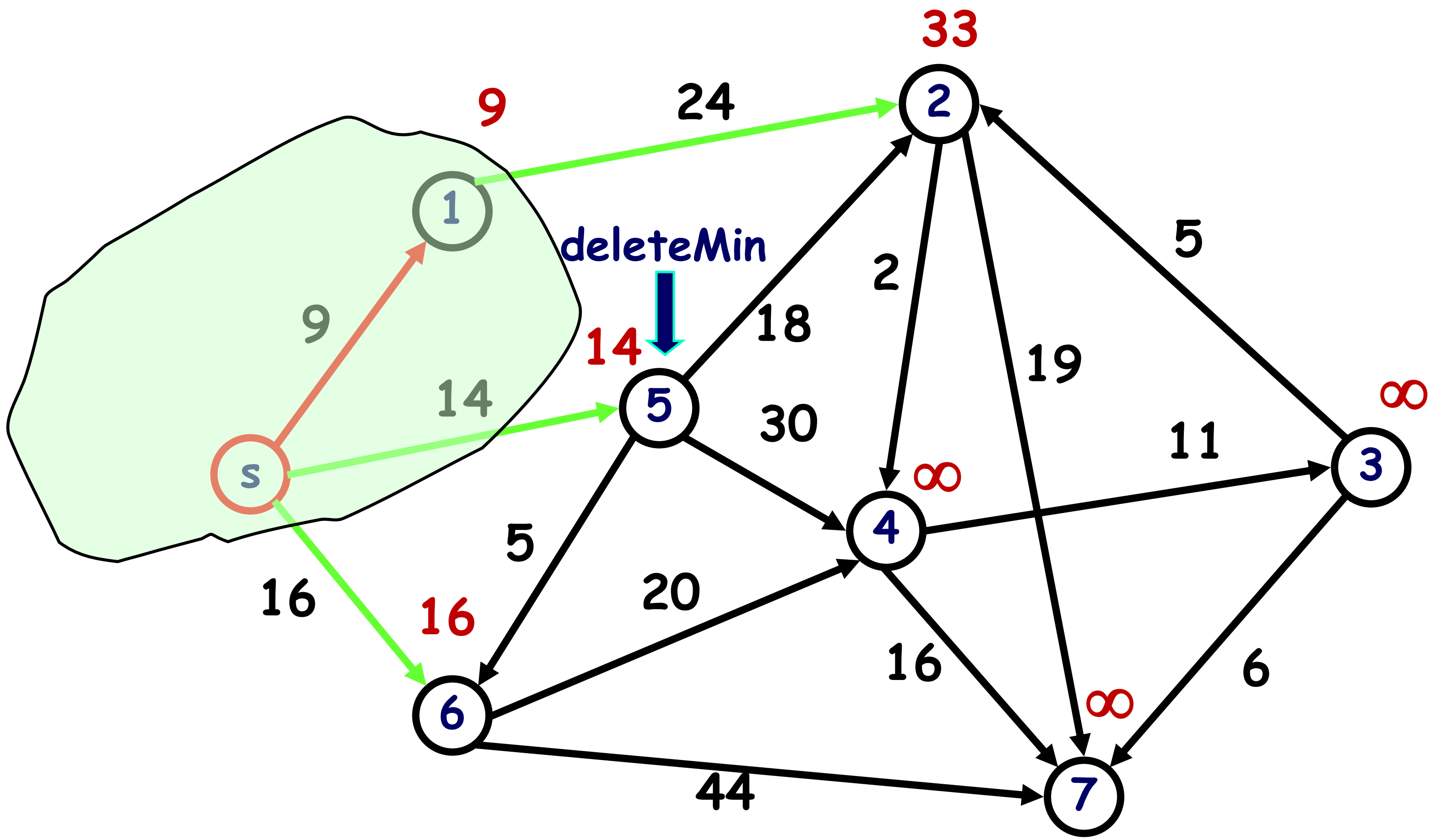
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



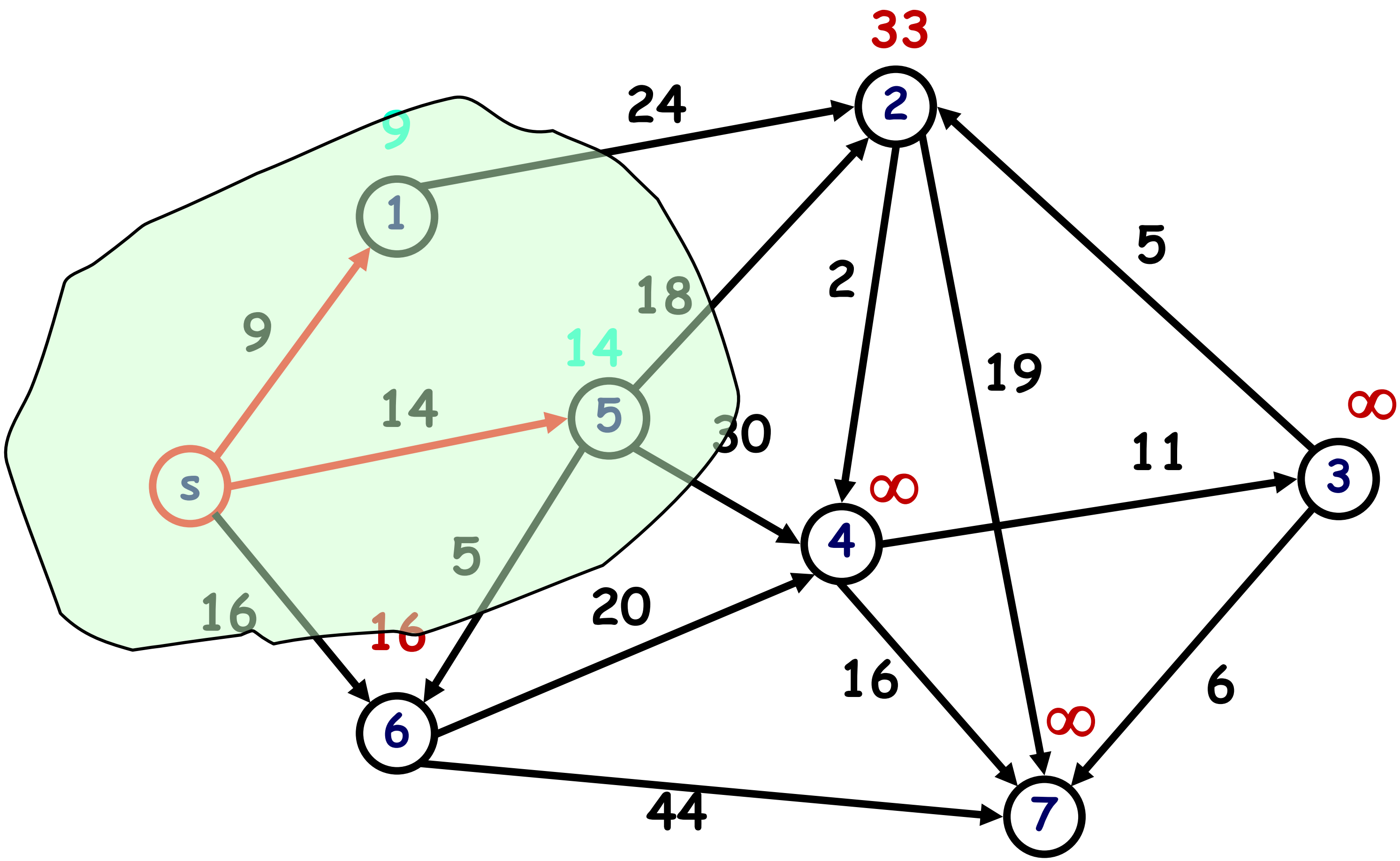
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



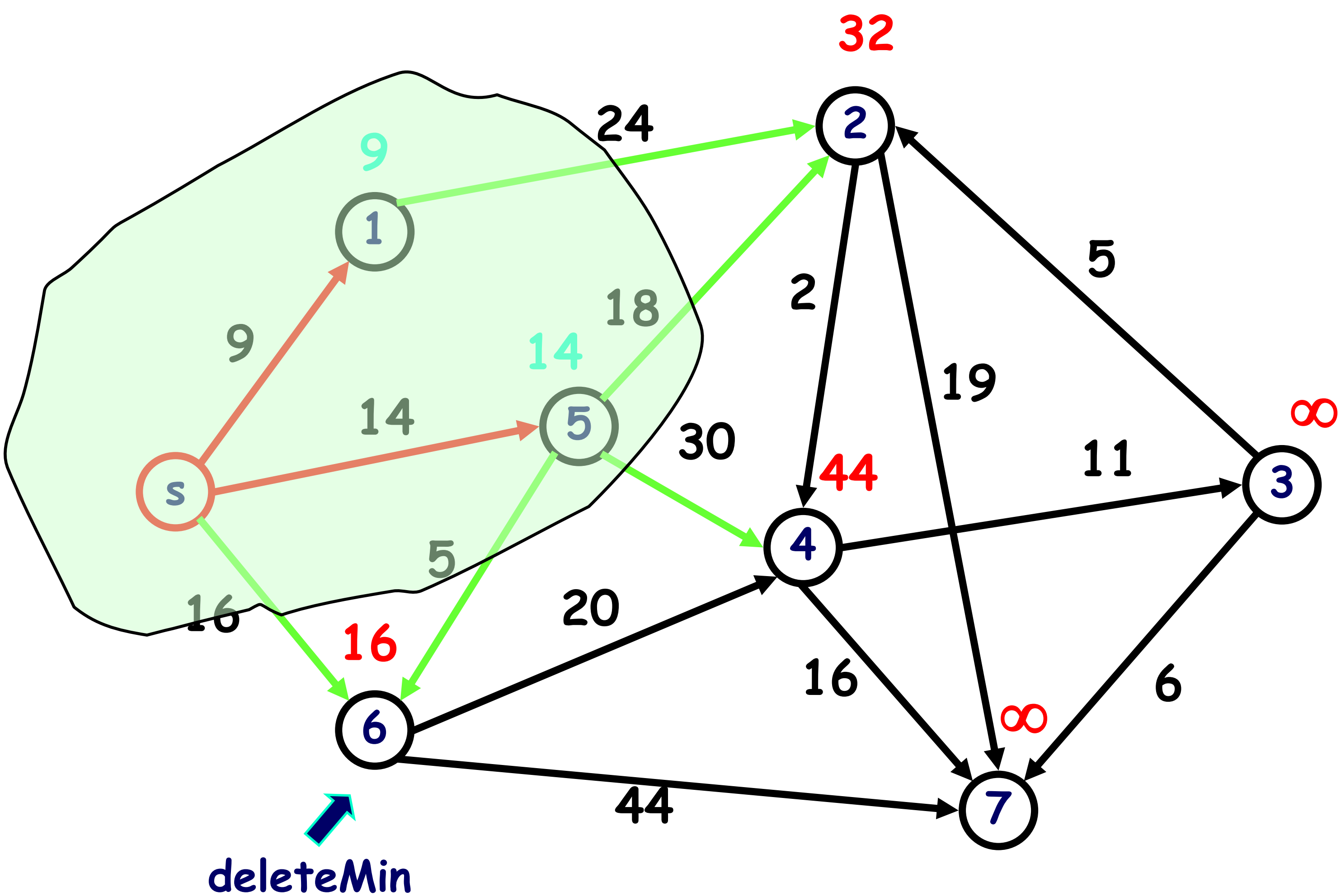
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



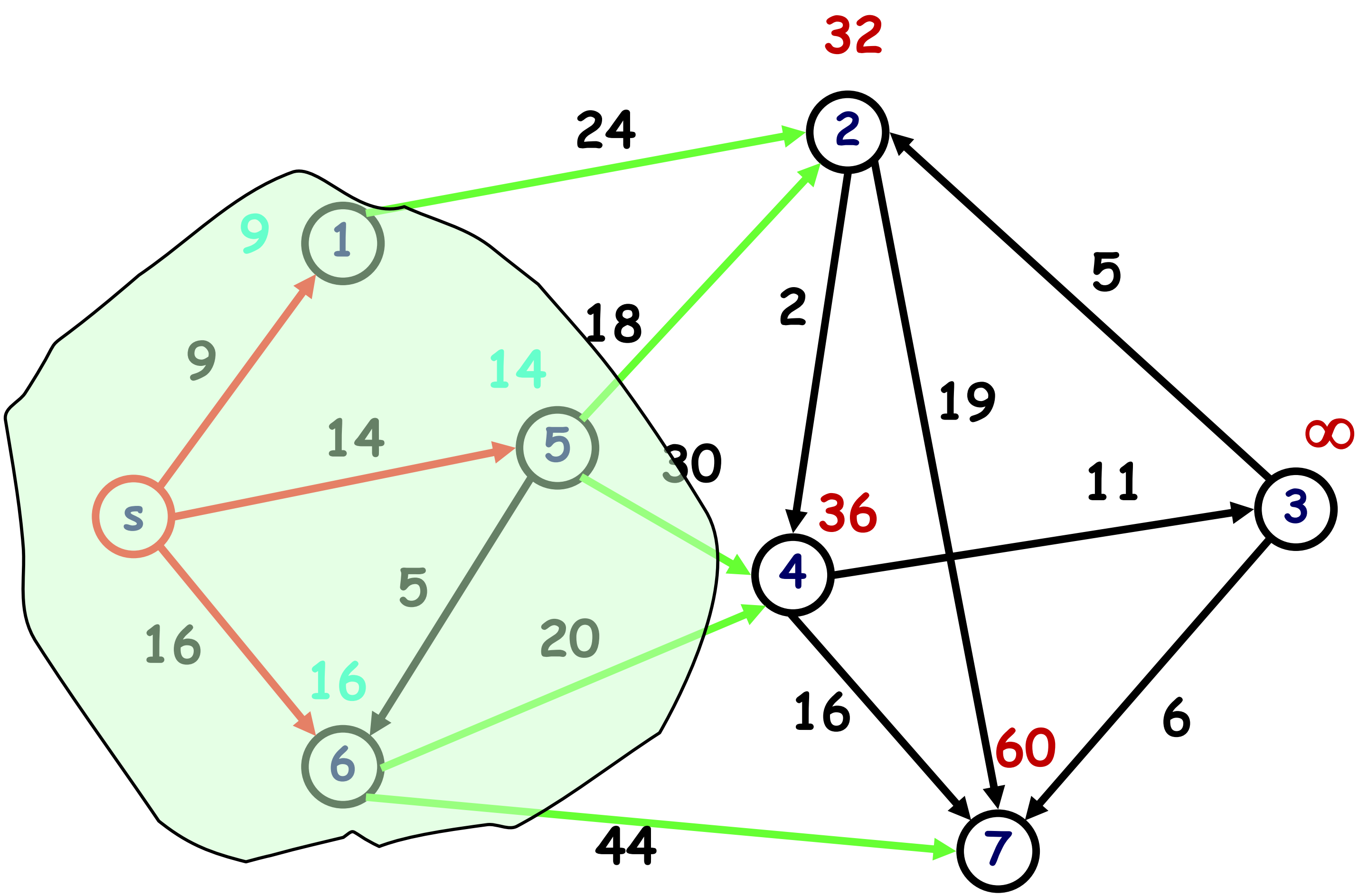
solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}



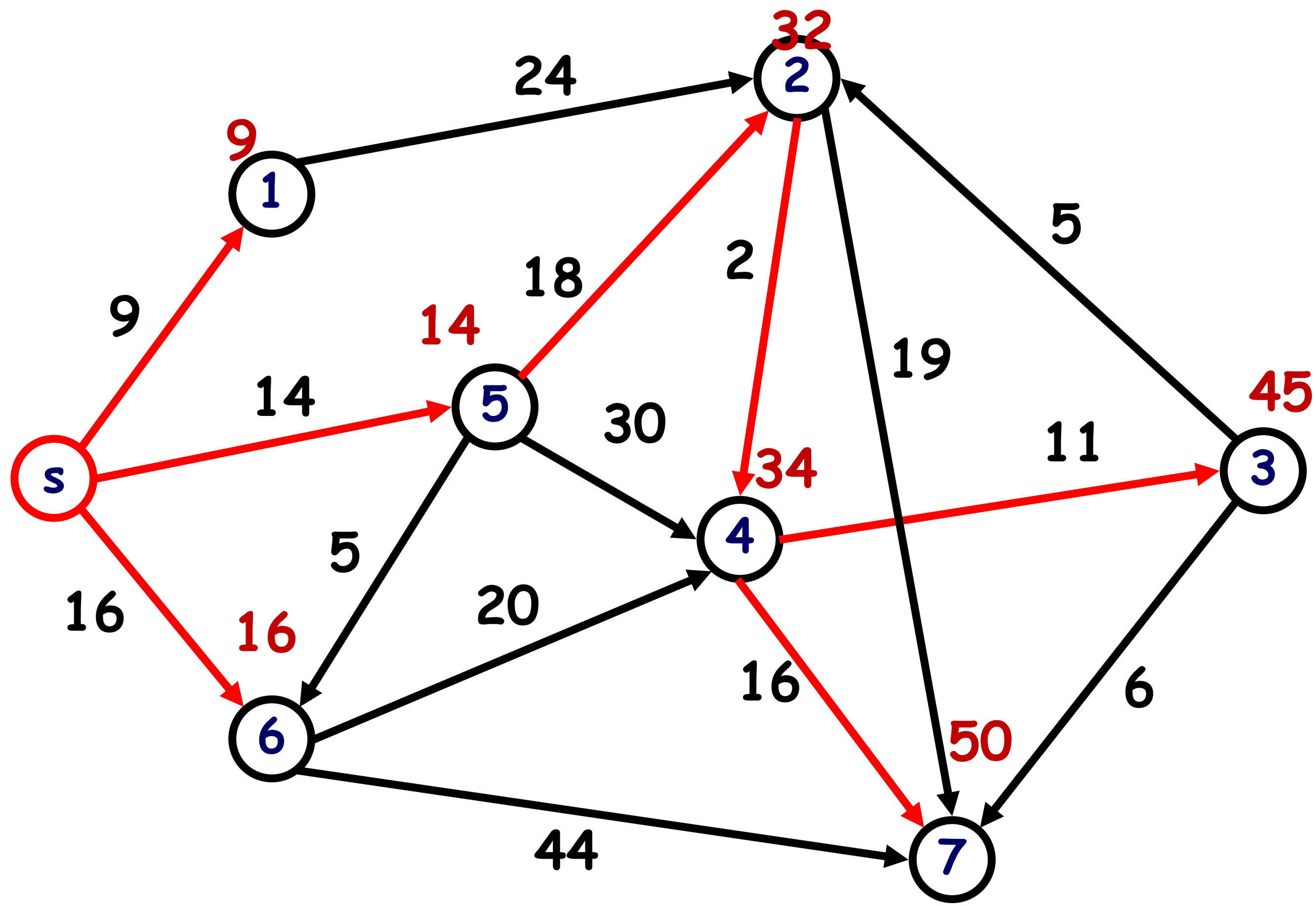
solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}



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



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



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.

Prim: heap of vertices
priority is a edge weith

Dijkstra: heap of vertices
priority is a distanse
from S to V

Algo:
1 DeleteMin, V times
2 DecreaseKey, 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?

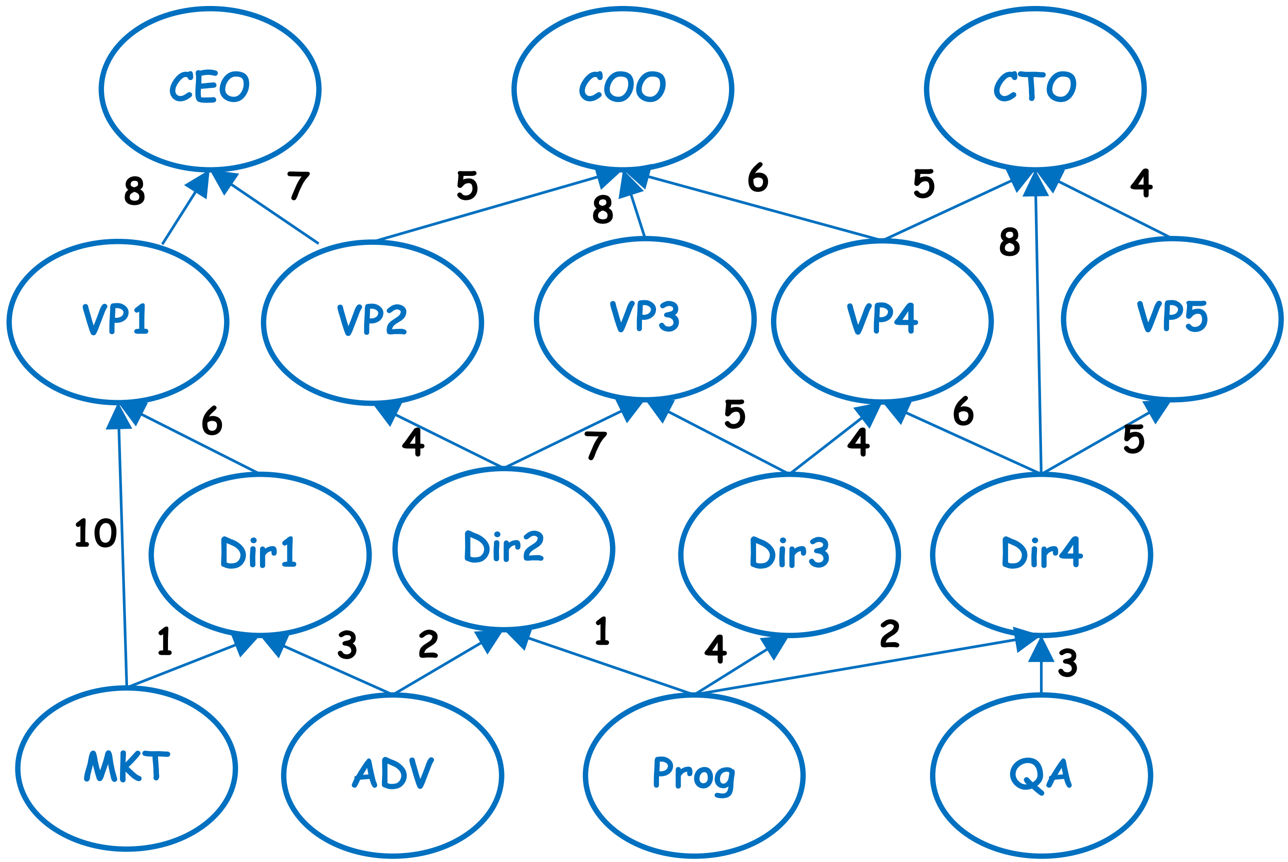
$O(V \cdot V + E \cdot 1)$

Use Fibonacci heap:

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

Discussion Problem 6

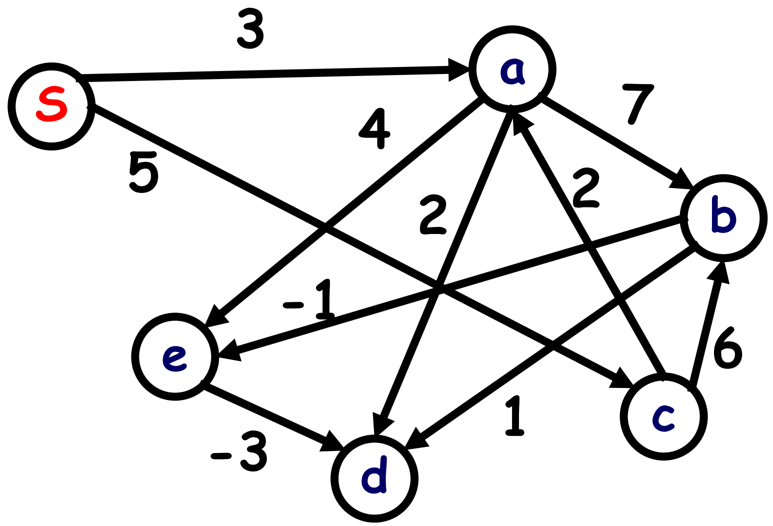
You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones which have no pre-requisites. The cost of an edge (v,u) is the effort required to go from one position v to position u. Salma wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's algorithm?



CEO
COO
CTO
8
7
5
8
6
5
8
4
VP1
VP2
VP3
VP4
VP5
6
4
7
5
4
6
5
10
Dir1
Dir2
Dir3
Dir4
1
3
2
1
4
2
3
MKT
ADV
Prog
QA

Discussion Problem 7

Design a linear time algorithm to find shortest distances in a DAG.



Discussion T/F Questions

(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.

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.