Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 4

University of Southern California

Spring 2023

Greedy Algorithms

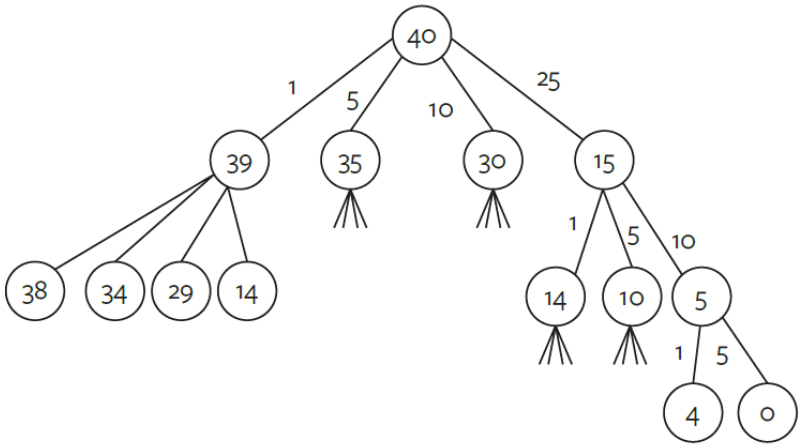
Reading: chapter 4

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.



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

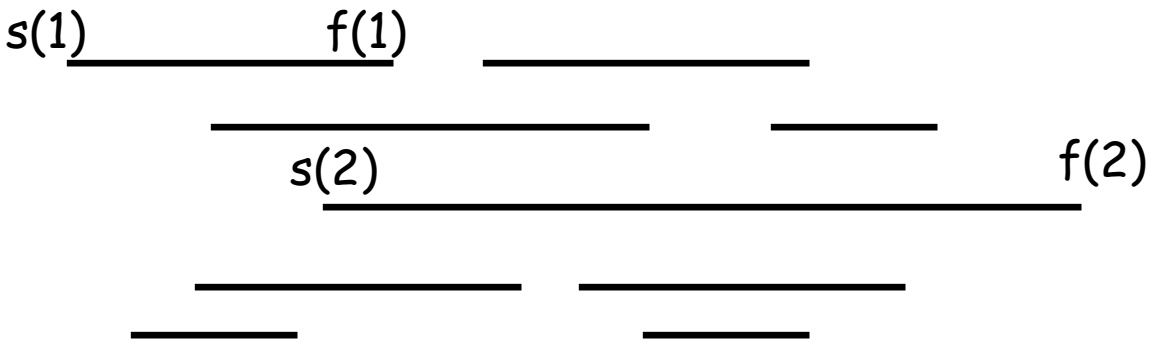
optimal substructure.

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 largest compatible (non-overlapping) subset of requests.



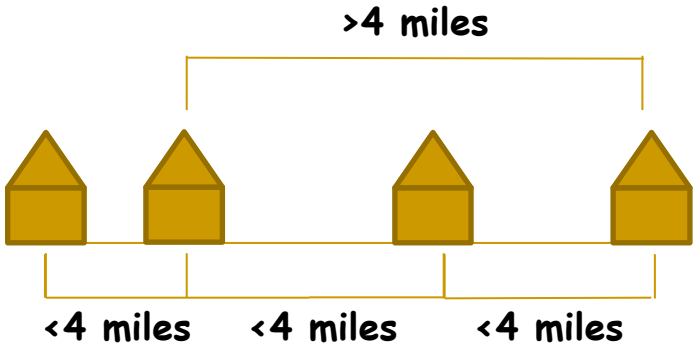
How do we choose requests?

Proof

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

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.



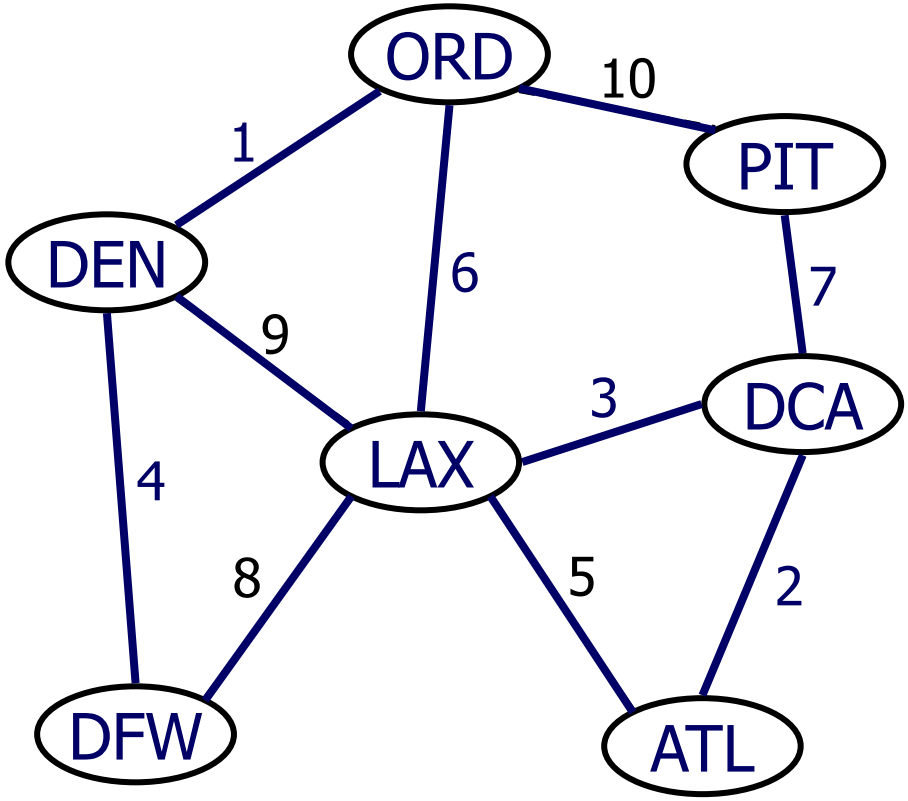
The Minimum Spanning Tree

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.



Kruskal's Algorithm

The algorithm builds a tree one EDGE at a time:

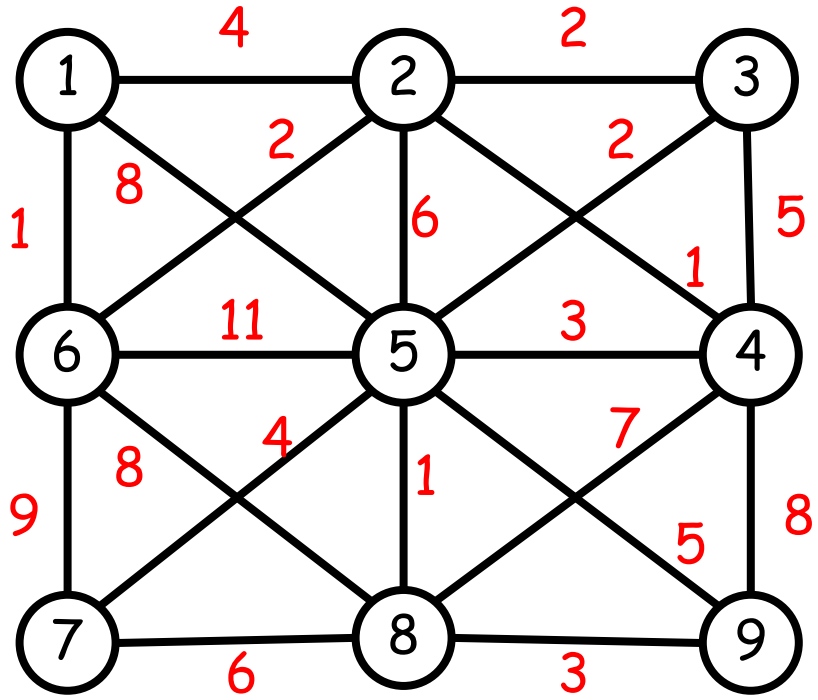
- Sort all edges by their weights.
- Loop:
  - Choose the minimum weight edge and join correspondent vertices (subject to cycles).
  - Go to the next edge.
  - Continue to grow the forest until all vertices are connected.

Sorting edges – O(E log E)

Cycle detection – O(V) for each edge

Total: O(V*E + E*log E)

Kruskal's Algorithm



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.

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.

Prim's Algorithm

The algorithm builds a tree one VERTEX at a time:

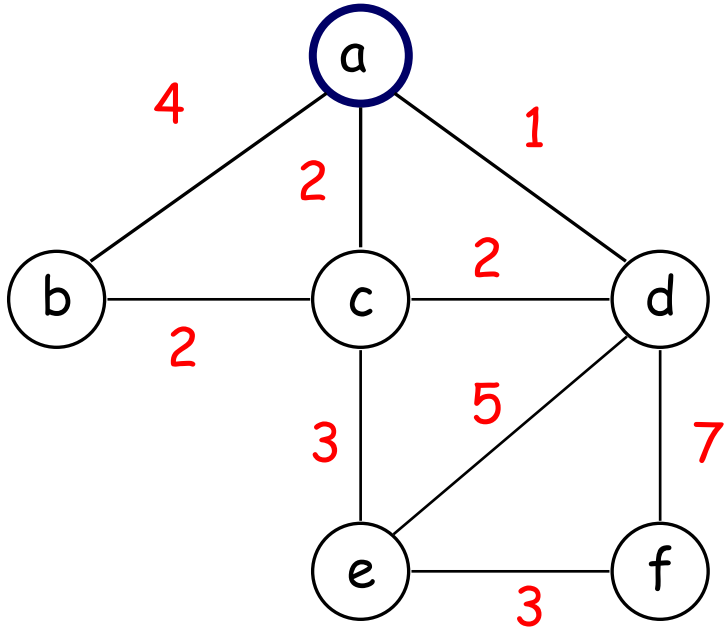
- Start with an arbitrary vertex as a sub-tree *C*.

- 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



Complexity of Prim's Algorithm

Discussion Problem 4

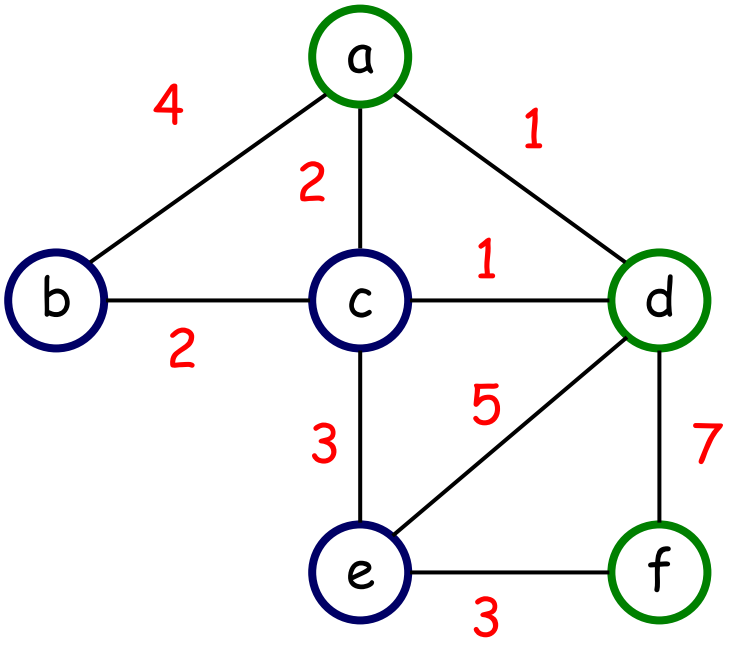
Assume that an unsorted array is used as a heap.
What would the running time of the Prim algorithm?

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

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.)

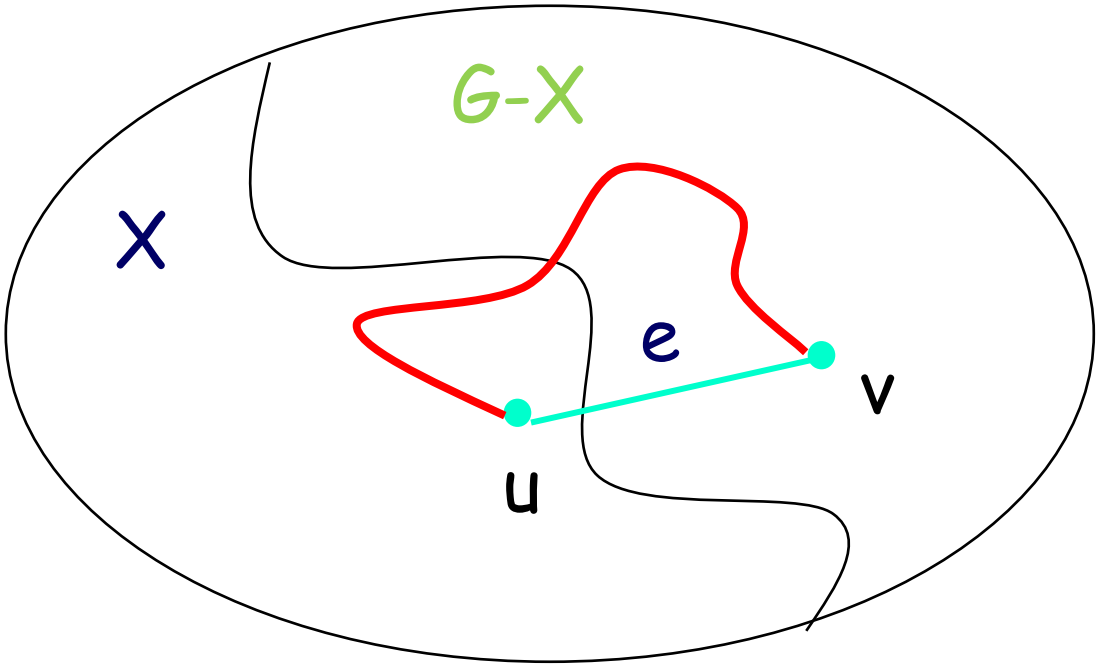
A crossing edge is an edge that connects a vertex in one set with a vertex in the other.



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

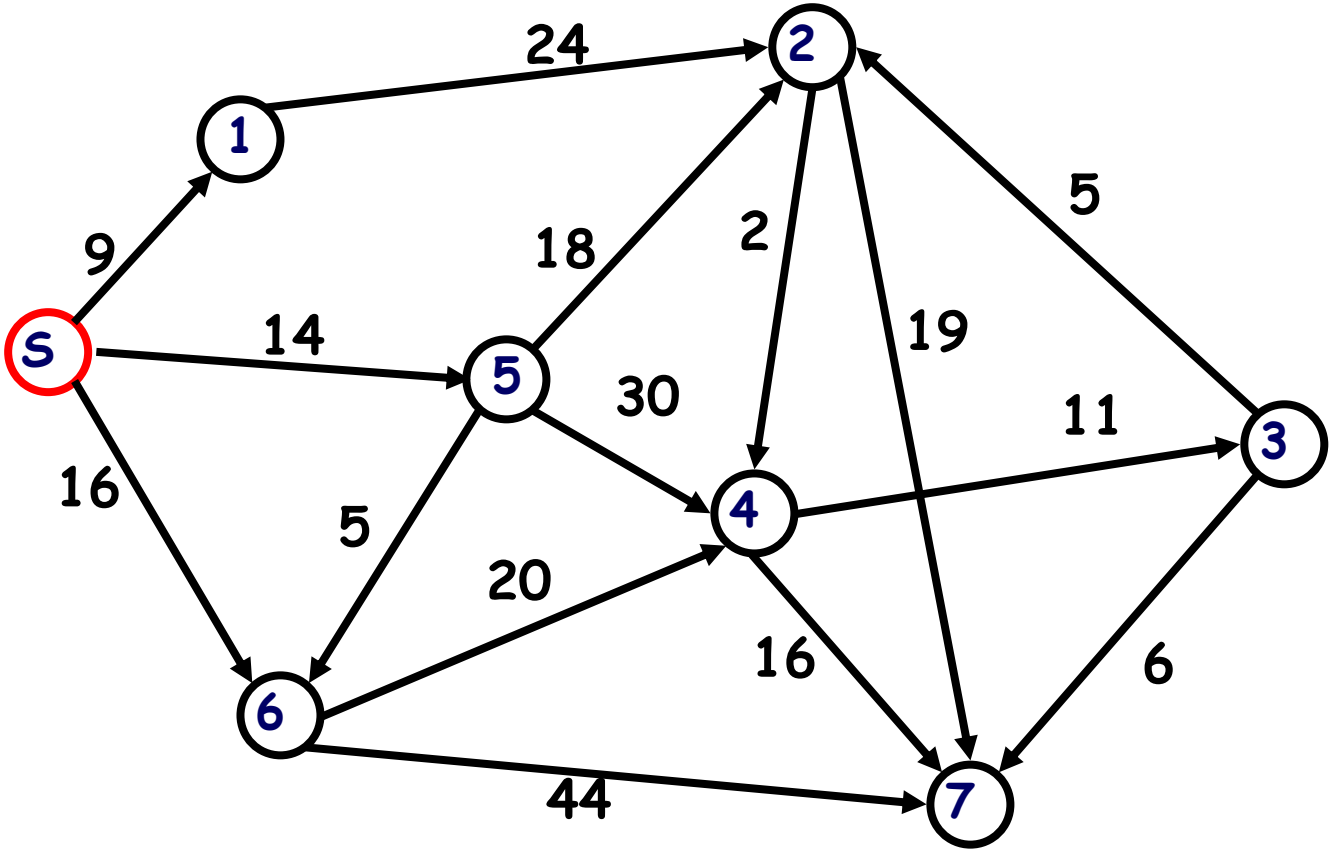
(**T/F**) The first edge added by Kruskal's algorithm can be the last edge added by Prim's algorithm.

(**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.

(**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.

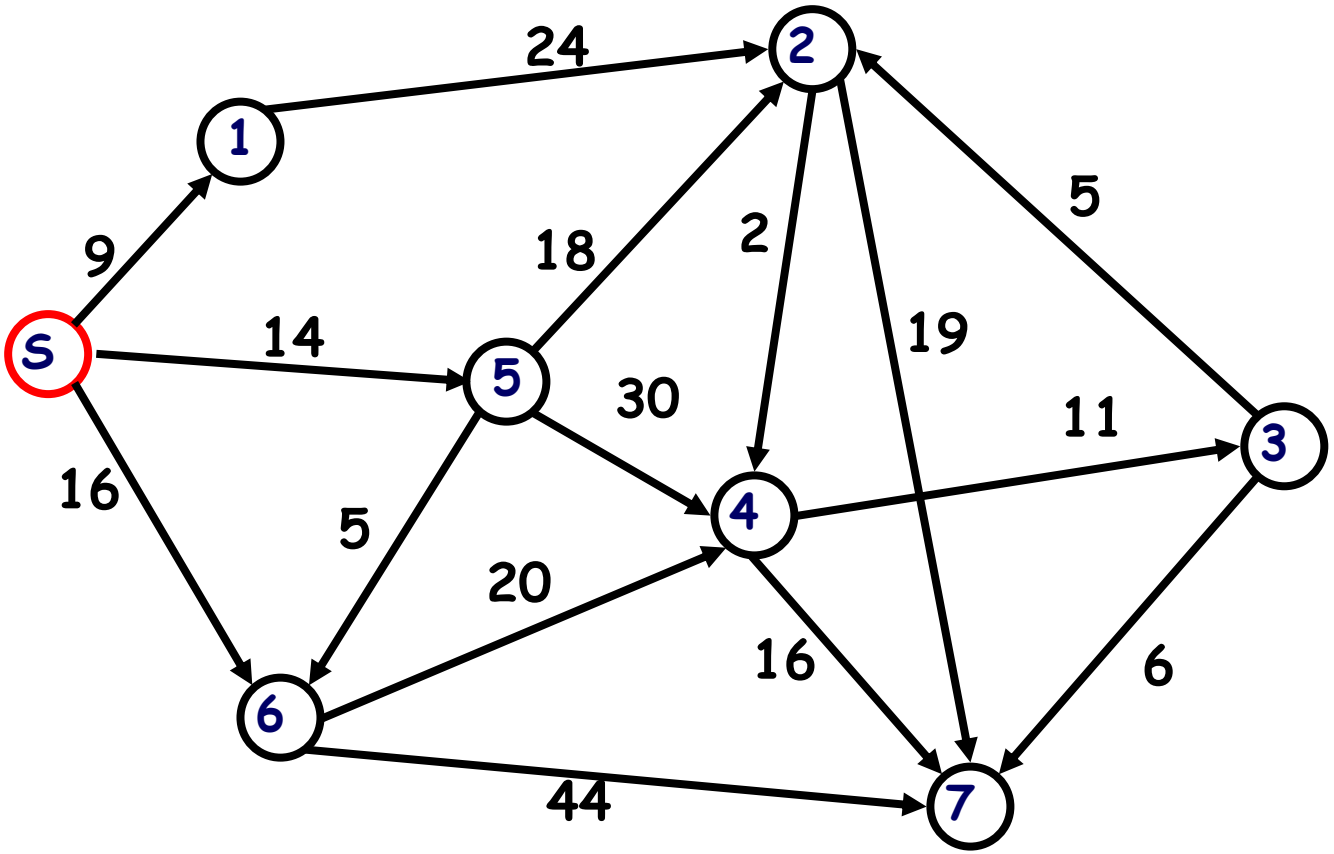
The Shortest Path Problem

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?



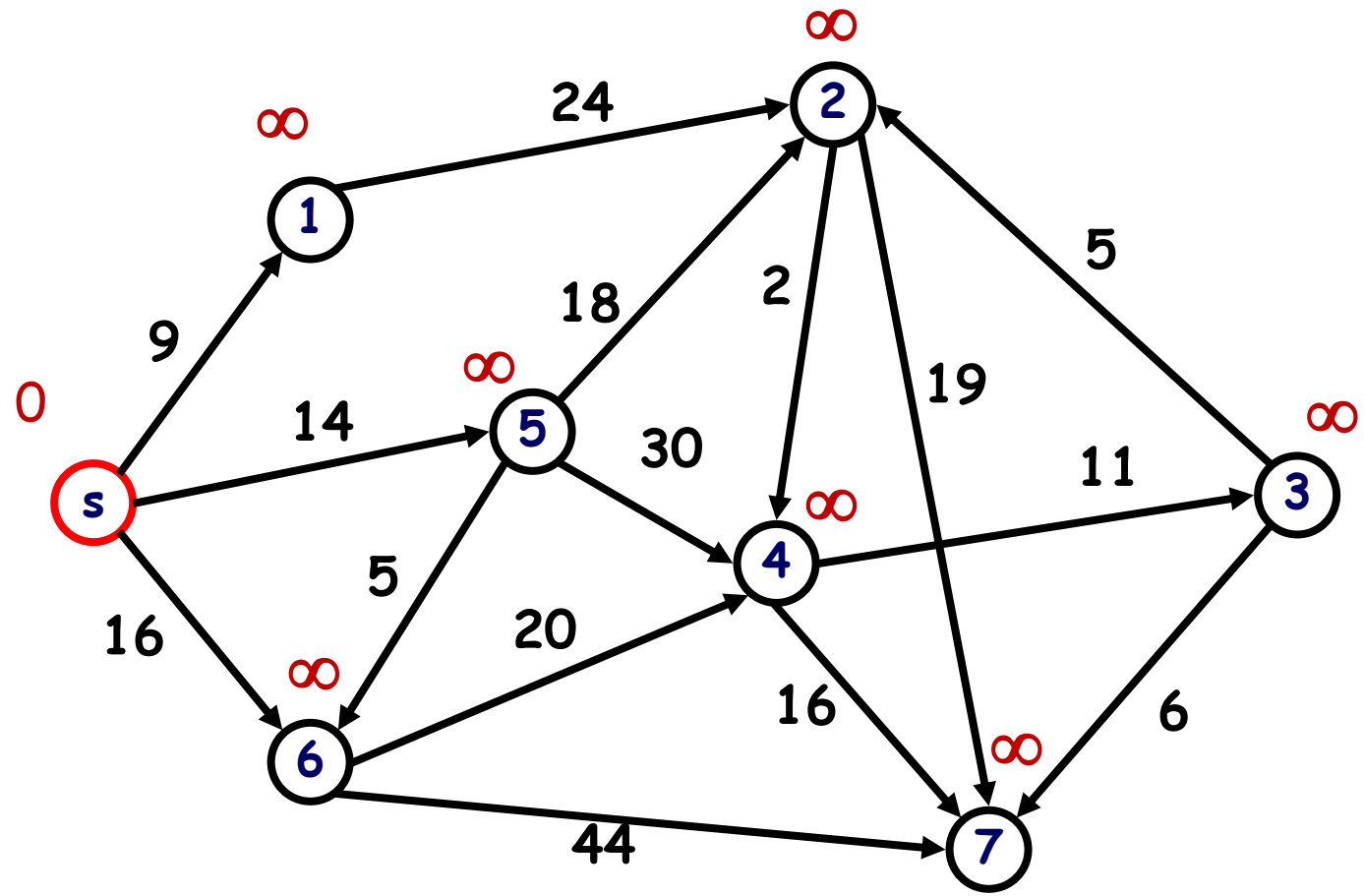
Greedy Approach

When algorithm proceeds all vertices are divided into two groups

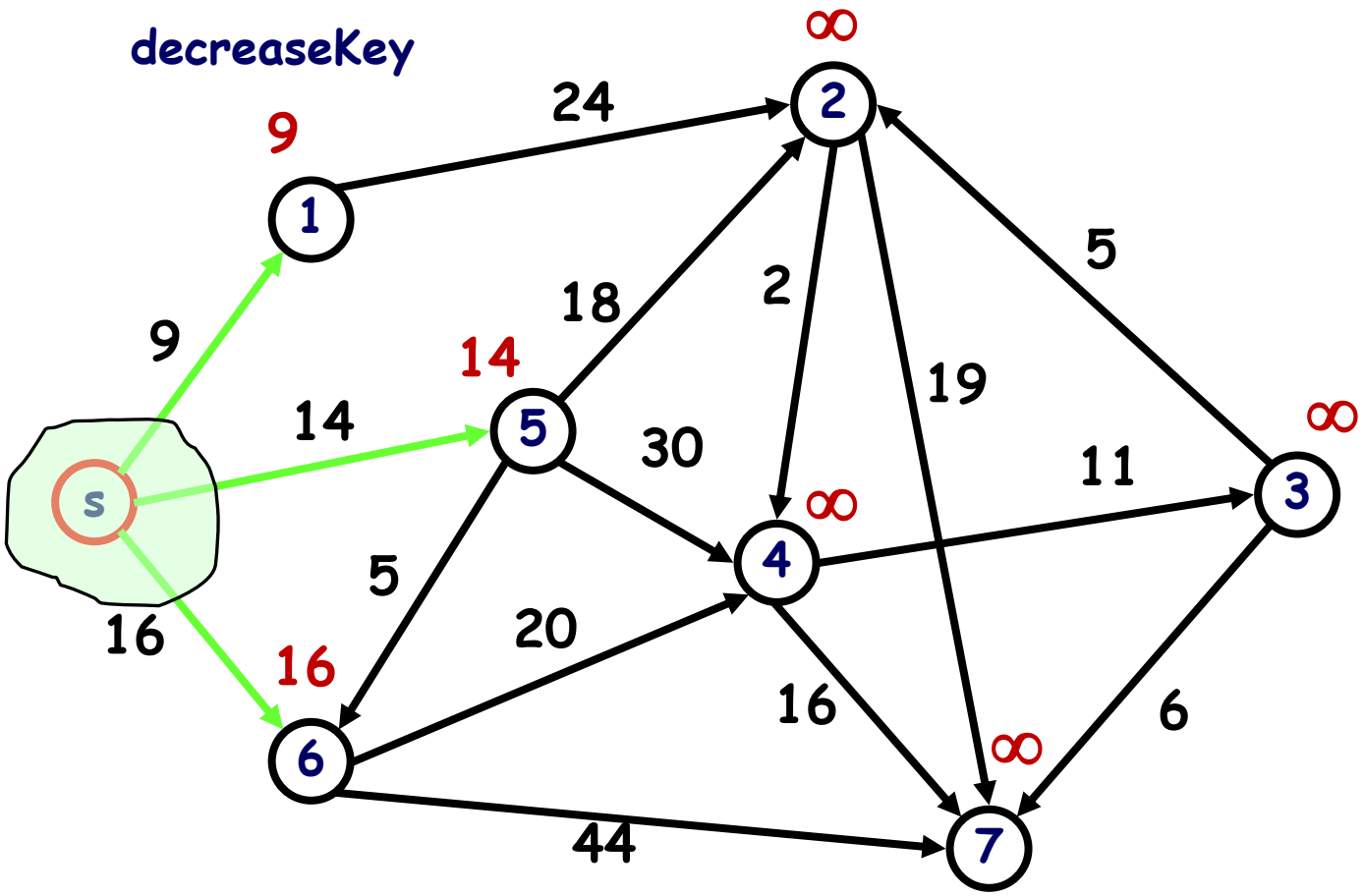
- vertices whose shortest path from the source is known
- vertices whose shortest path from the source is NOT discovered yet.

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.

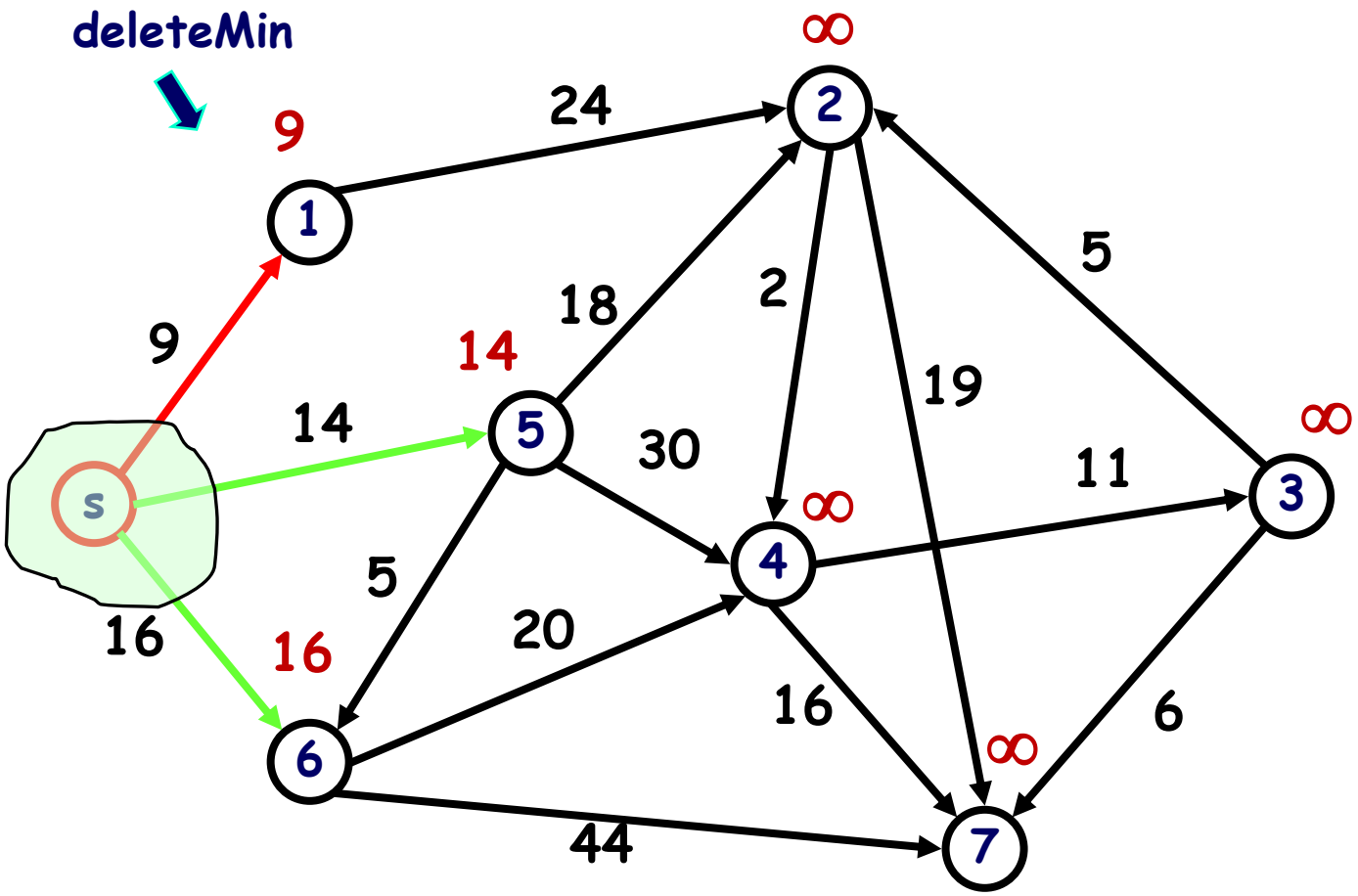
solution tree = s
heap = {1, 2, 3, 4, 5, 6, 7}



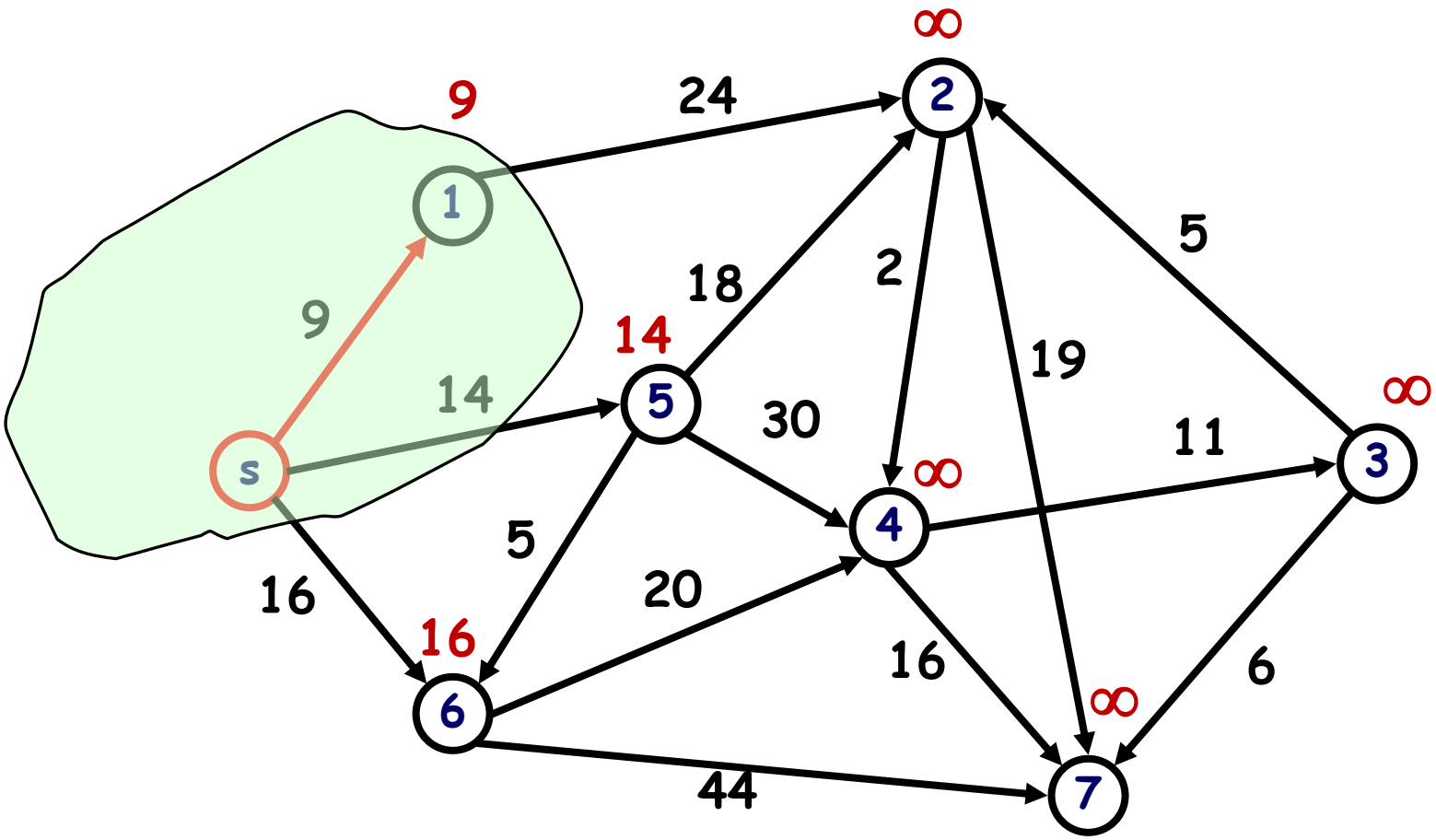
solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}



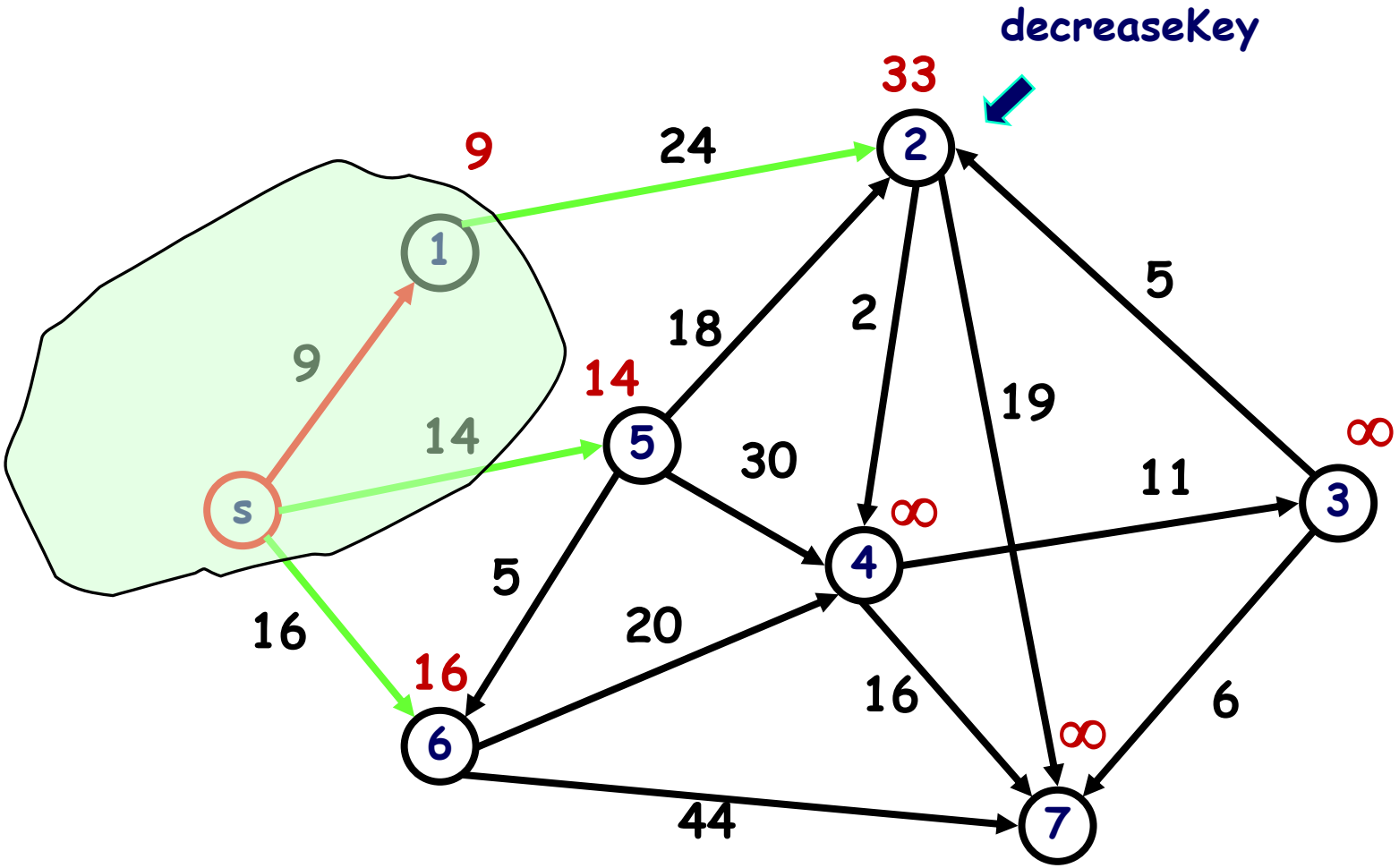
solution tree = { s }
heap = {1, 2, 3, 4, 5, 6, 7}



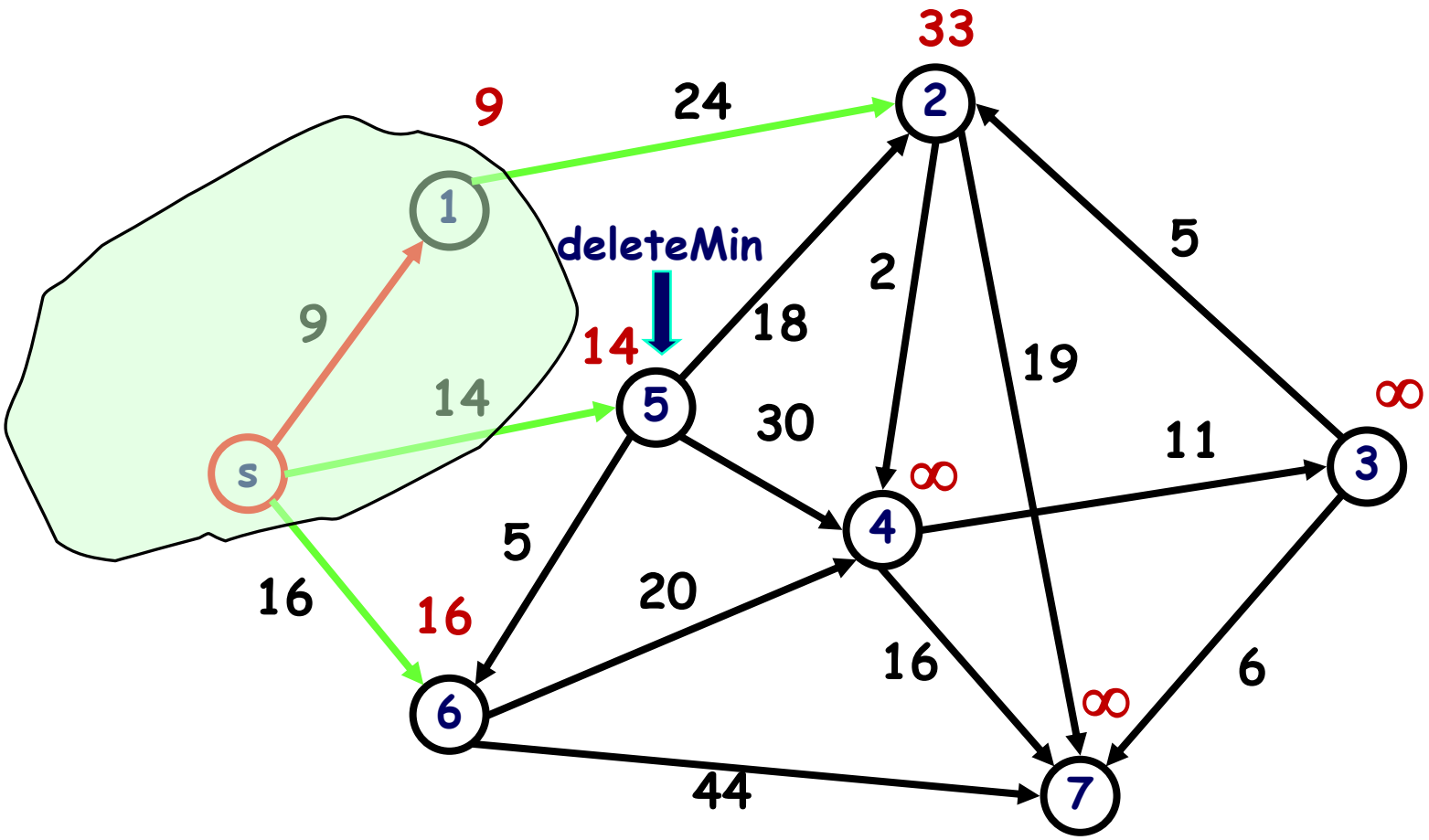
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



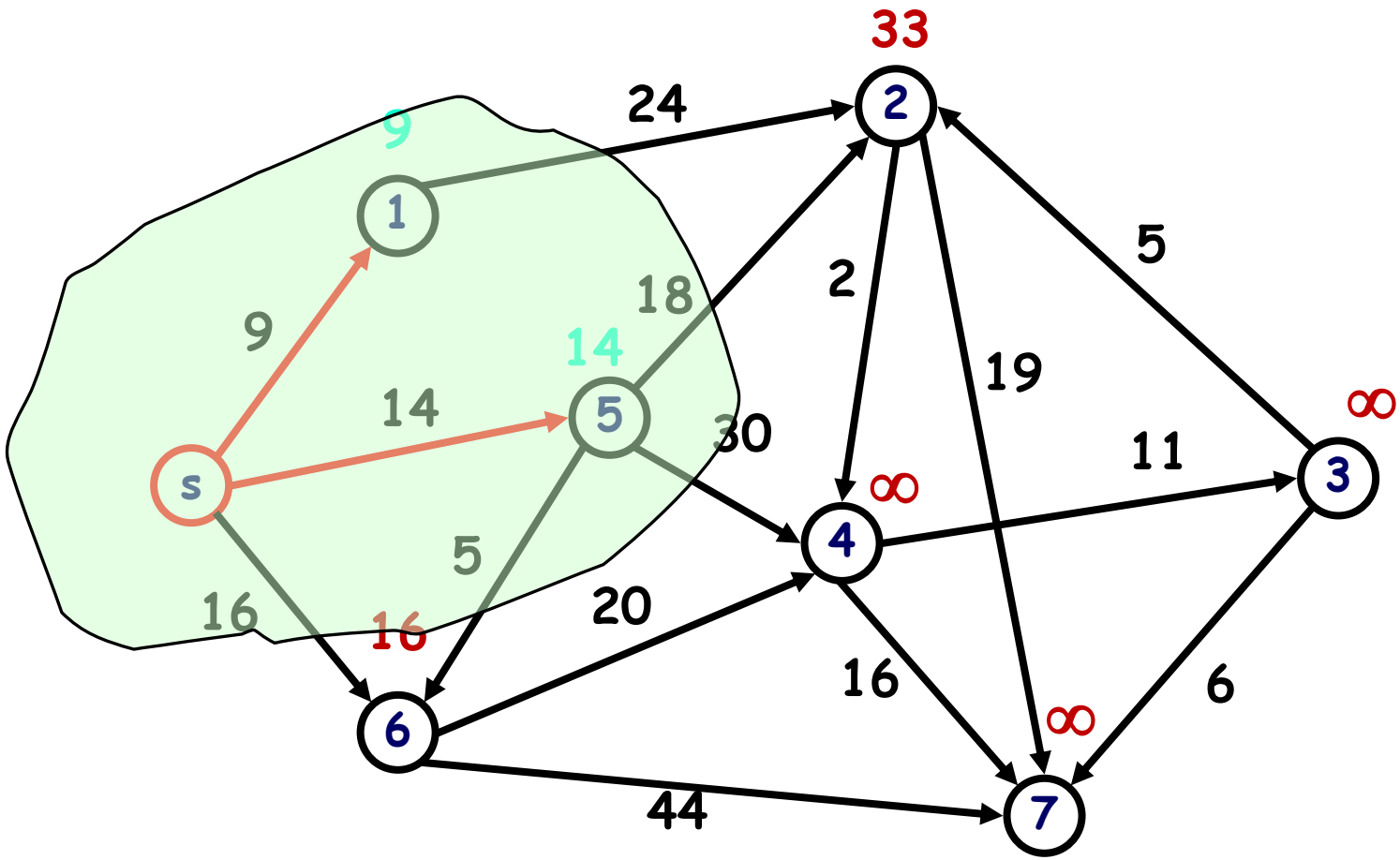
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



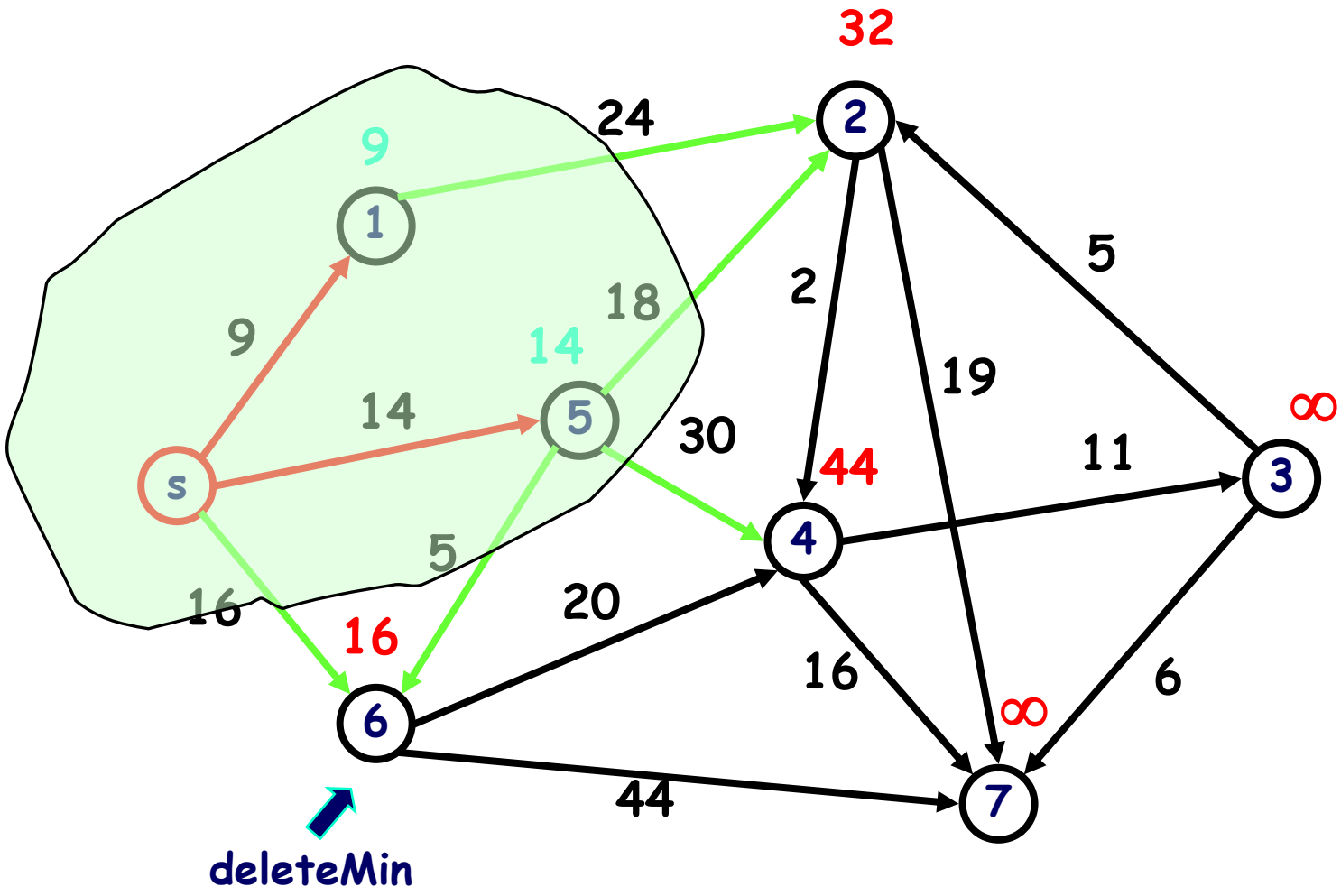
solution tree = { s, 1 }
heap = {2, 3, 4, 5, 6, 7}



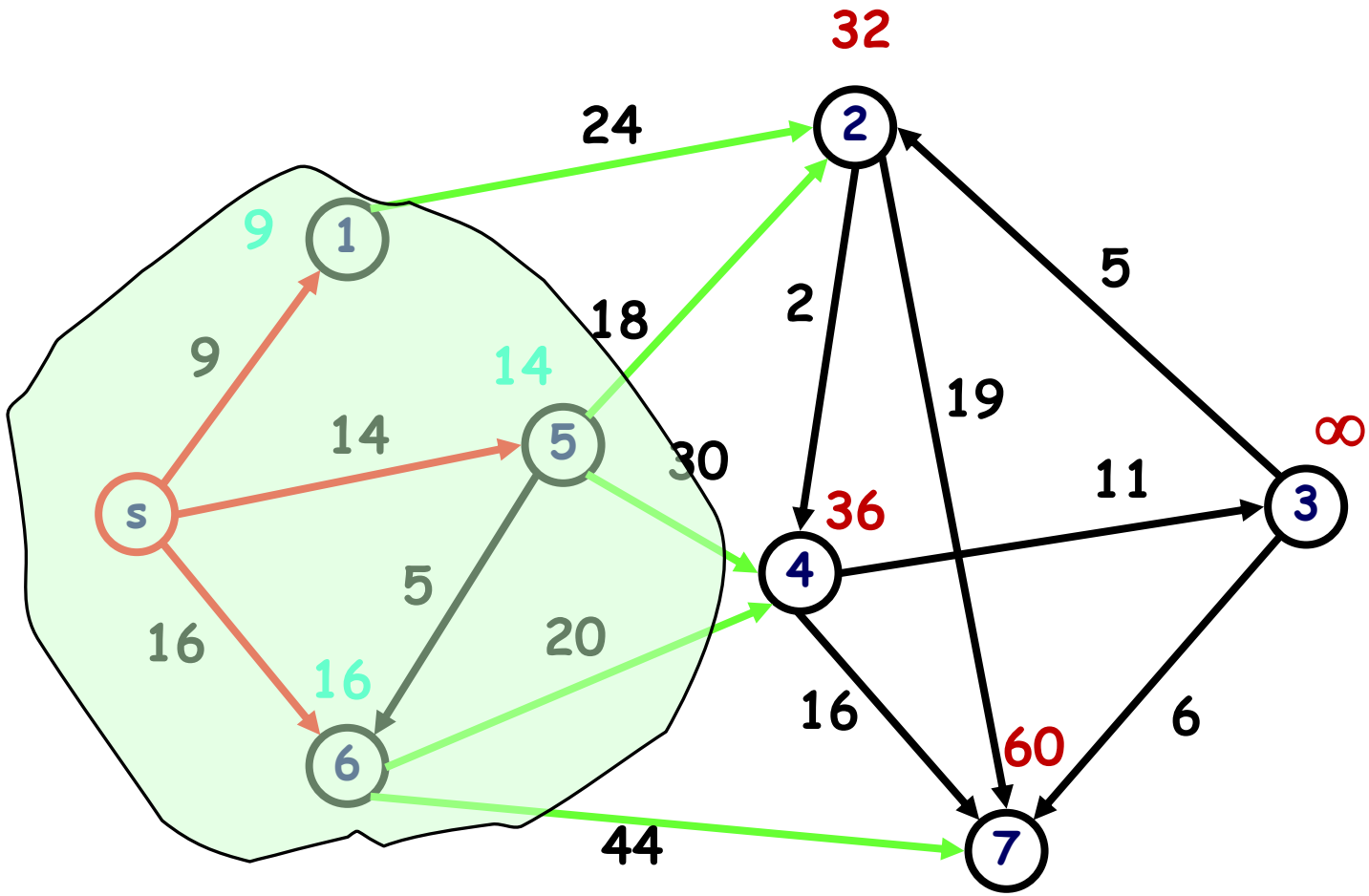
solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}



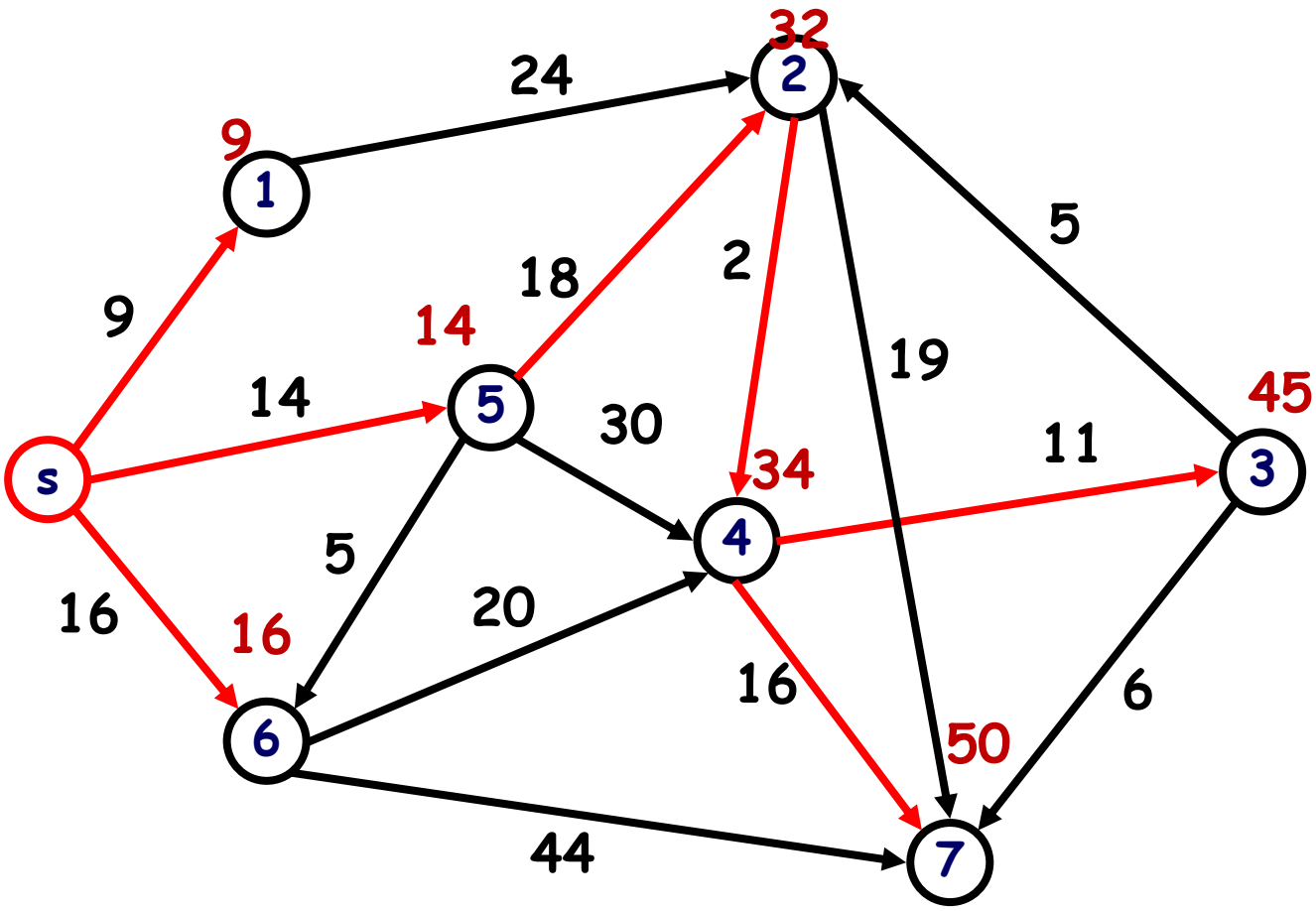
solution tree = { s, 1, 5 }
heap = {2, 3, 4, 6, 7}



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.