CS340 - Analysis of Algorithms

Greedy Algorithms

Announcements:

HW2 gradescope is up (sorry)

In class assignment gradescope is up

Check piazza for class announcements!

Upcoming deadlines:

- Lab2 due Tomorrow
- HW3 due next Monday (9/29)
- Project checkpoint 1 (10/5)

Agenda

1. Review: Binary Heaps

2. Shortest Path
    a. Description of problem
    b. A greedy solution (Dijkstra's algorithm)
    c. Runtime analysis
    d. Proof of correctness

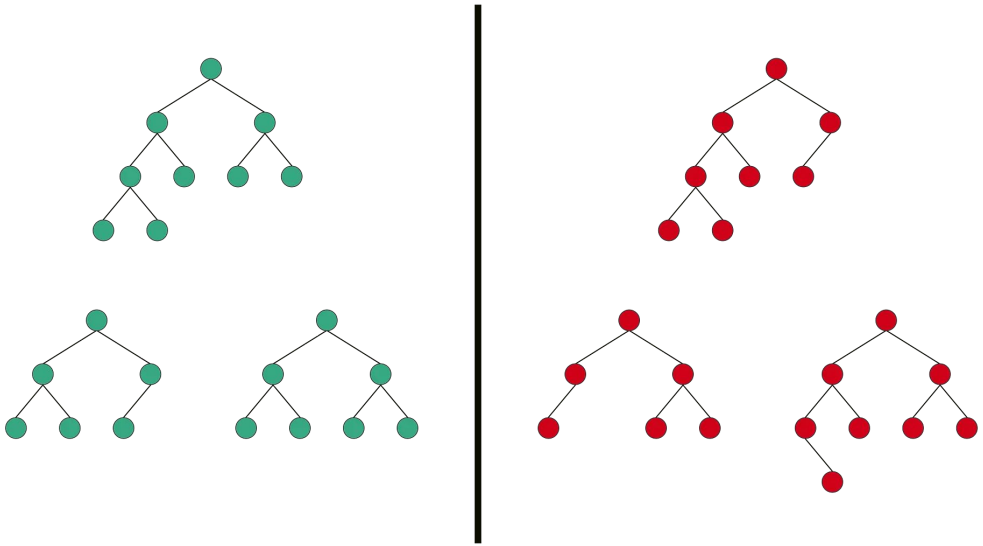
Review: Binary Min-Heaps

Min heaps maintain the following properties:

1. Completeness:
   a. all levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not fully filled, the nodes of that level are filled from left to right
2. For every node n, n.key >= parent(n.key)
   a. Root is the minimum

Complete Binary Trees

All levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right



Min heaps

After each operation, heaps maintain the properties of:

1. Completeness
2. Each node's value is larger than its parent's value

This is achieved by:

1. Always insert at the leftmost bottommost open slot
    a. Upheap if necessary to preserve second property
2. After poll, always replace root with leftmost bottomost node
    a. Downheap if necessary to preserve second property

Warmup: Draw the min-heap after each operation

Insert(A, 4)

Insert(B, 2)

Insert(C, 5)

Insert(D, 1)

Poll()

Insert(E, 3)

Poll()

Poll()

Dijkstra’s algorithm

Shortest Paths in a Graph

Given a graph one may ask:

- Given nodes u and v what is the shortest u-v path?
- Given a start node s, what is the shortest path from s to each other node?

Formally, we are given a directed graph G = (V, E) with a designated start node s.

We assume s has a path to every other node in G

Each edge e has a **weight** w(u,v)

Length of a path is the sum of weights along the edges of the path

Distance between any two vertices u and v is the min length of any path between the vertices.
Denoted δ(u,v)

Shortest Path

We will assume there is a single source s

Given a directed graph G = (V, E) with edge weights and a source vertex s $\in$ V, determine the distance $\delta(s,v)$, $\forall v \in V$

Assume the edge weights are non-negative

Dijkstra’s Algorithm

- Maintains a set S of vertices u for which we have determined a shortest-path distance.
  - The “explored” part of the graph

- For each node $v \in V \setminus S$, we determine the shortest path that can be constructed by traveling along a path through the explored part S to some $u \in S$, followed by the single edge (u, v).
  - We choose the node $v \in V \setminus S$ which minimizes d[u] + w(u, v)

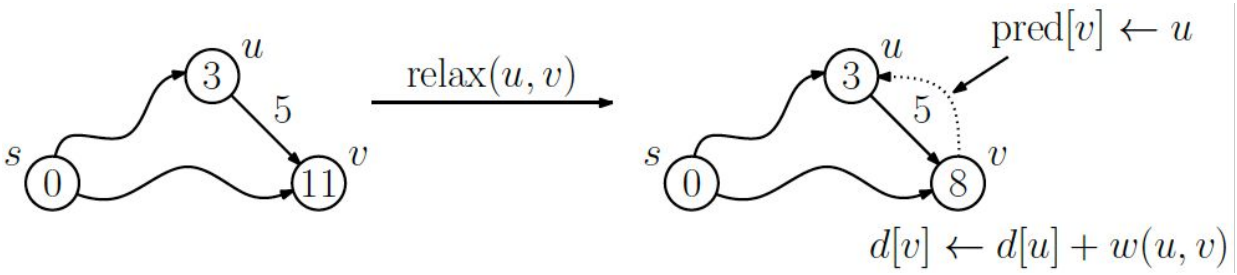
Dijkstra’s Algorithm

- Maintains an estimate of the shortest path for each vertex, d[v] from s
- d[v] stores the length of the shortest path from s to v that the algorithm currently knows of
- Initially, d[s] = 0 and d[v] = ∞, v != s
- The algorithm updates d[v] as it process more vertices
  - This is called relaxation

Relaxation

- Consider an edge (u,v)
- We have current values for d[u] and d[v]
- d[v] should be the min of (d[u] + w(u,v), d[v])
  - In the path from s to v, do we want to go through u?

```
relax(u, v){
  if (d[u]+w(u, v)<d[v]) {
    d[v] = d[u] + w(u, v)
    pred[v] = u
  }
}
```



Dijkstra's Algorithm

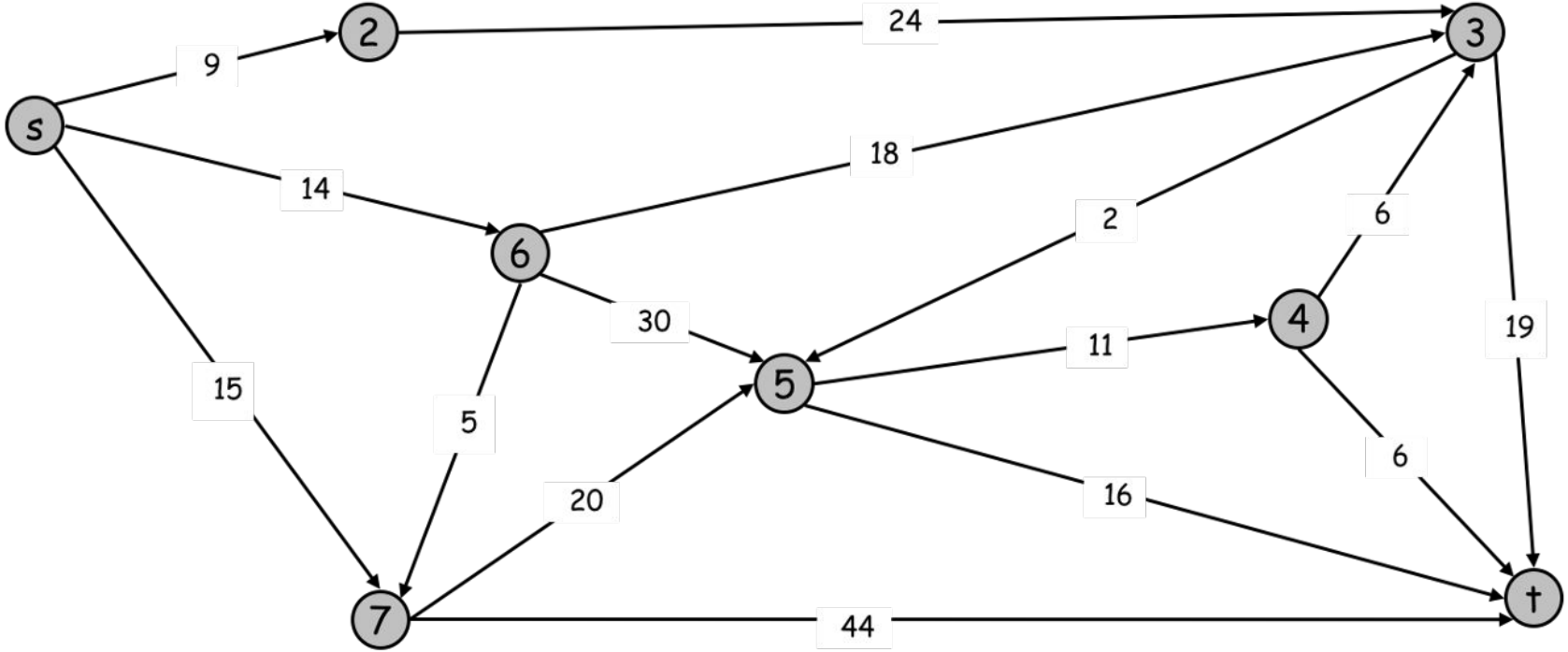
1. init:
   a. assign a init distance for each node
      - $d[s] = 0, d[v] = \infty, v \neq s$
   b. Q = { s }

2. while Q is non-empty:
   b. Select a u ∈ Q for which d[u] is the minimum
   c. For each neighbor of u:
      ii. distance of neighbor = d[u] + w(u, neighbor)
      iii. If this distance is less than the current dist, update it.
      iv. If distance changed, add to Q
      v. add u to S

Dijkstra's -

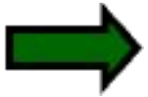
Input: graph with weighted (non-negative) edges

Output: shortest path from s to all V

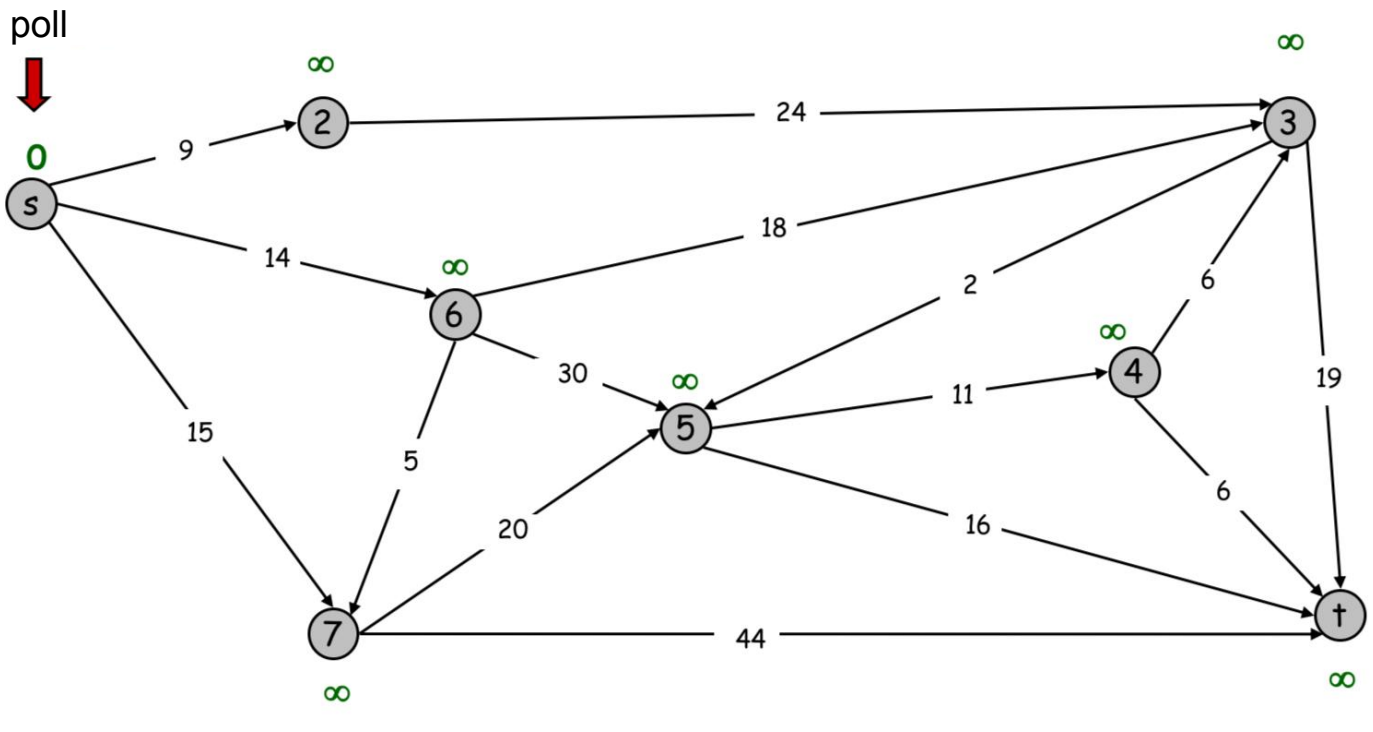


Notation

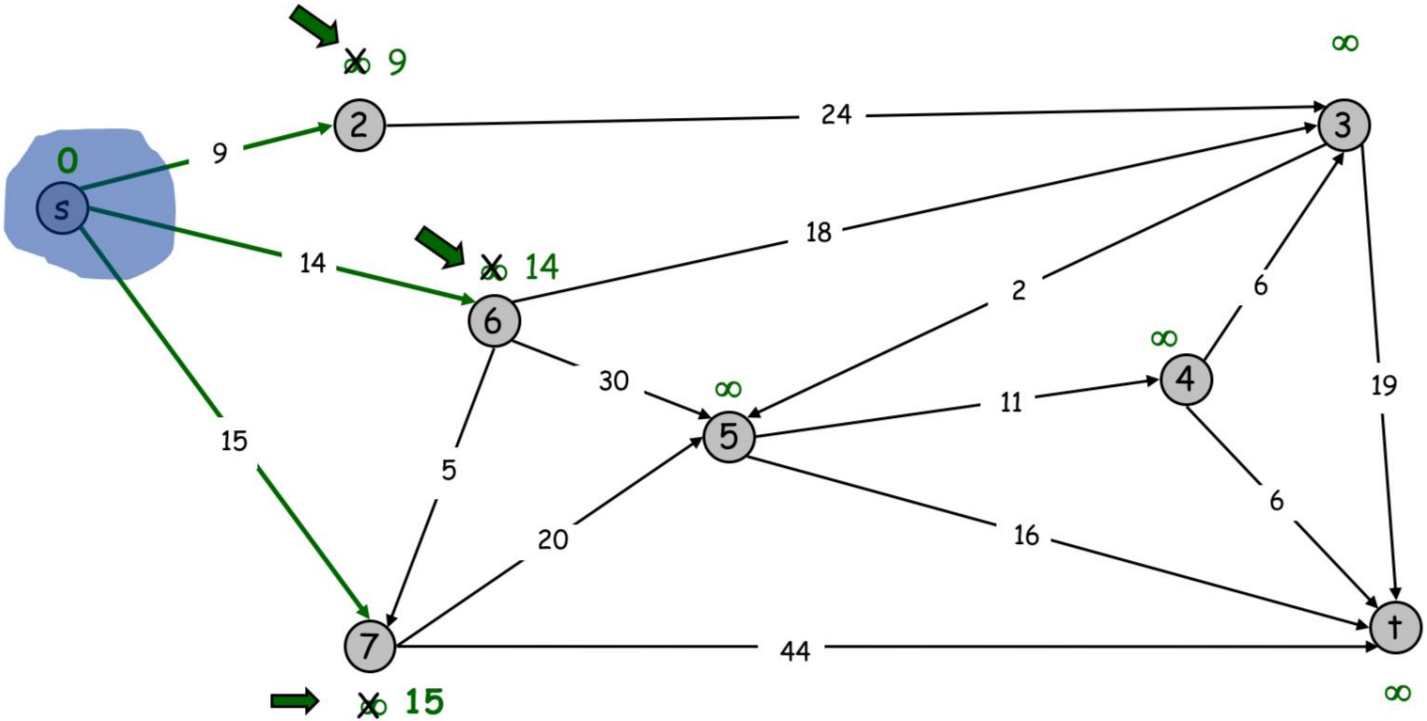
- blue blob: S 

- red arrows: poll ($u \in S$ for which d[u] is the minimum) 
- green arrows: Update d[v] 
- green numbers: current `d[v]` 34

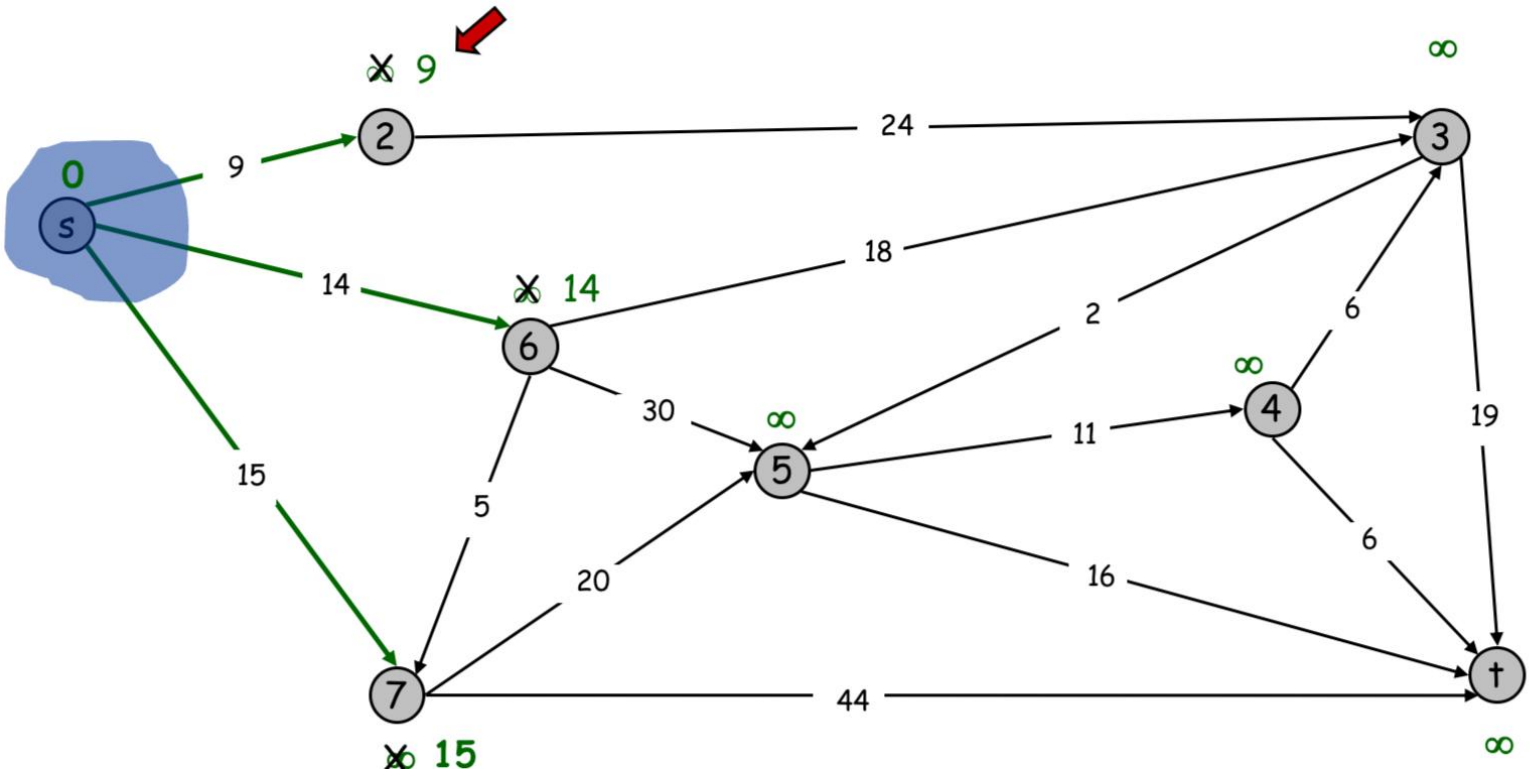
Dijkstra's algorithm



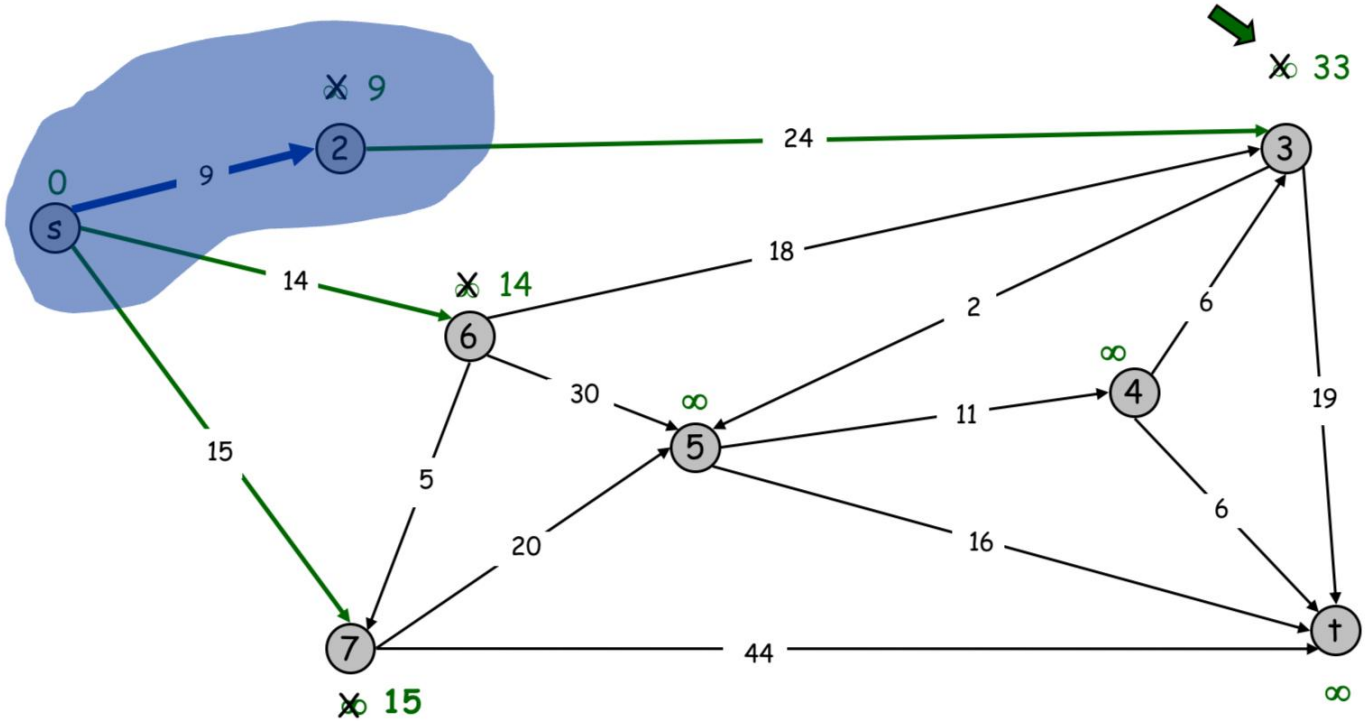
Dijkstra's algorithm



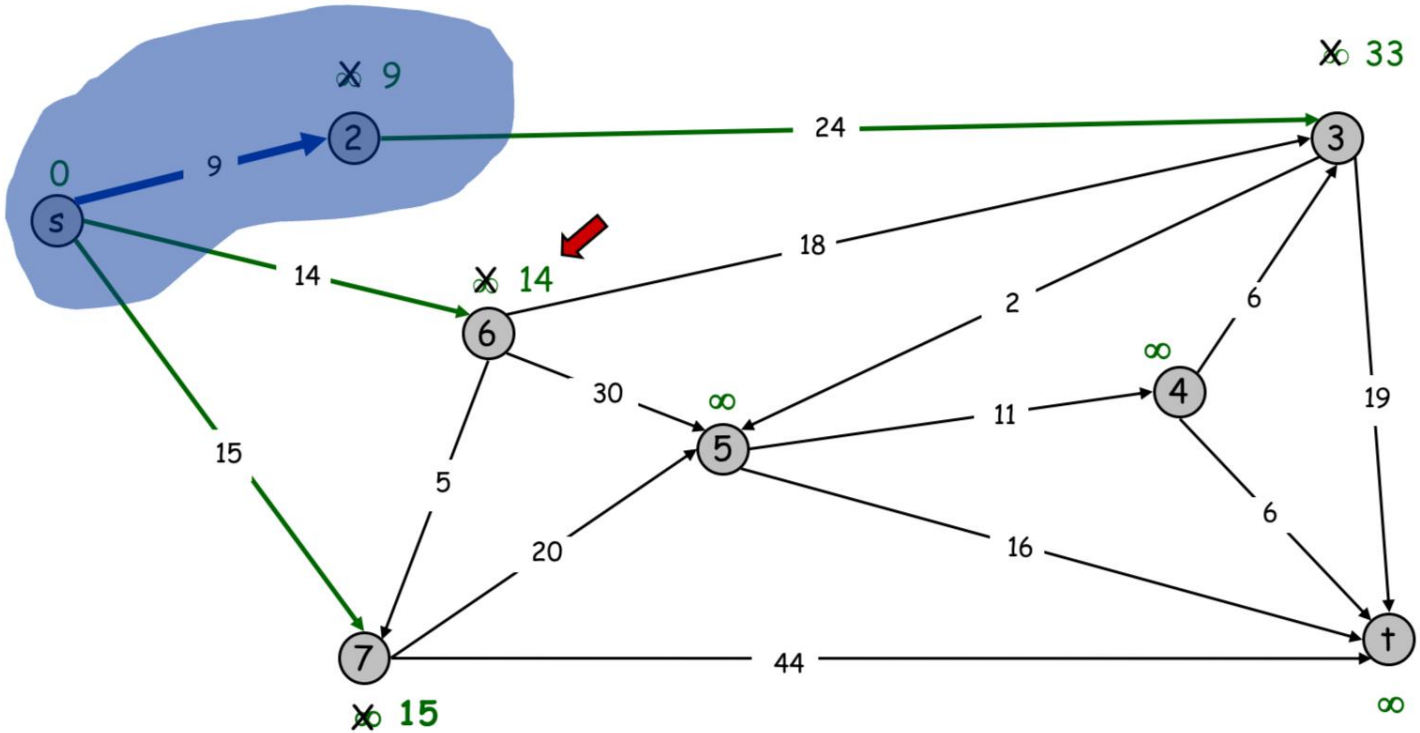
Dijkstra's algorithm



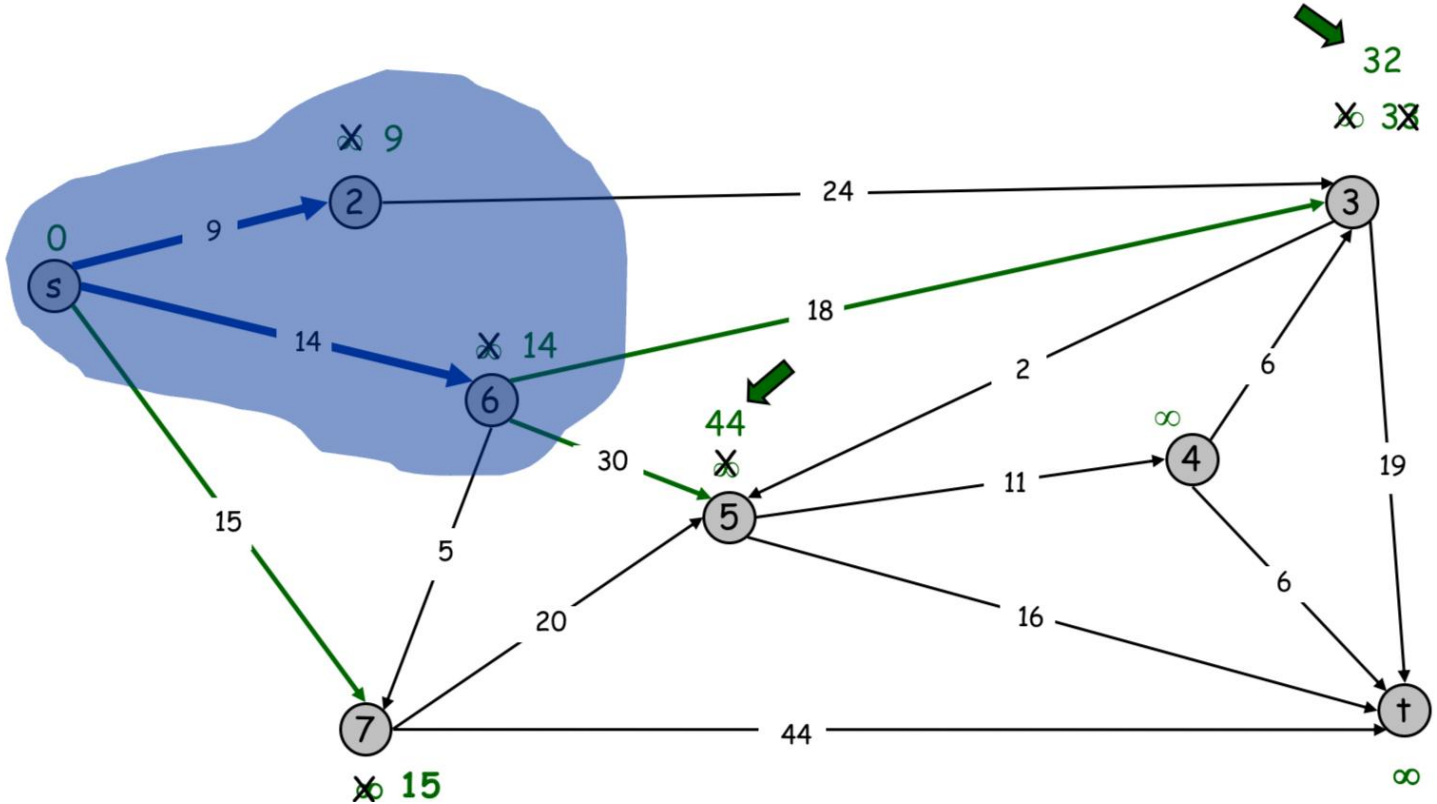
Dijkstra's algorithm



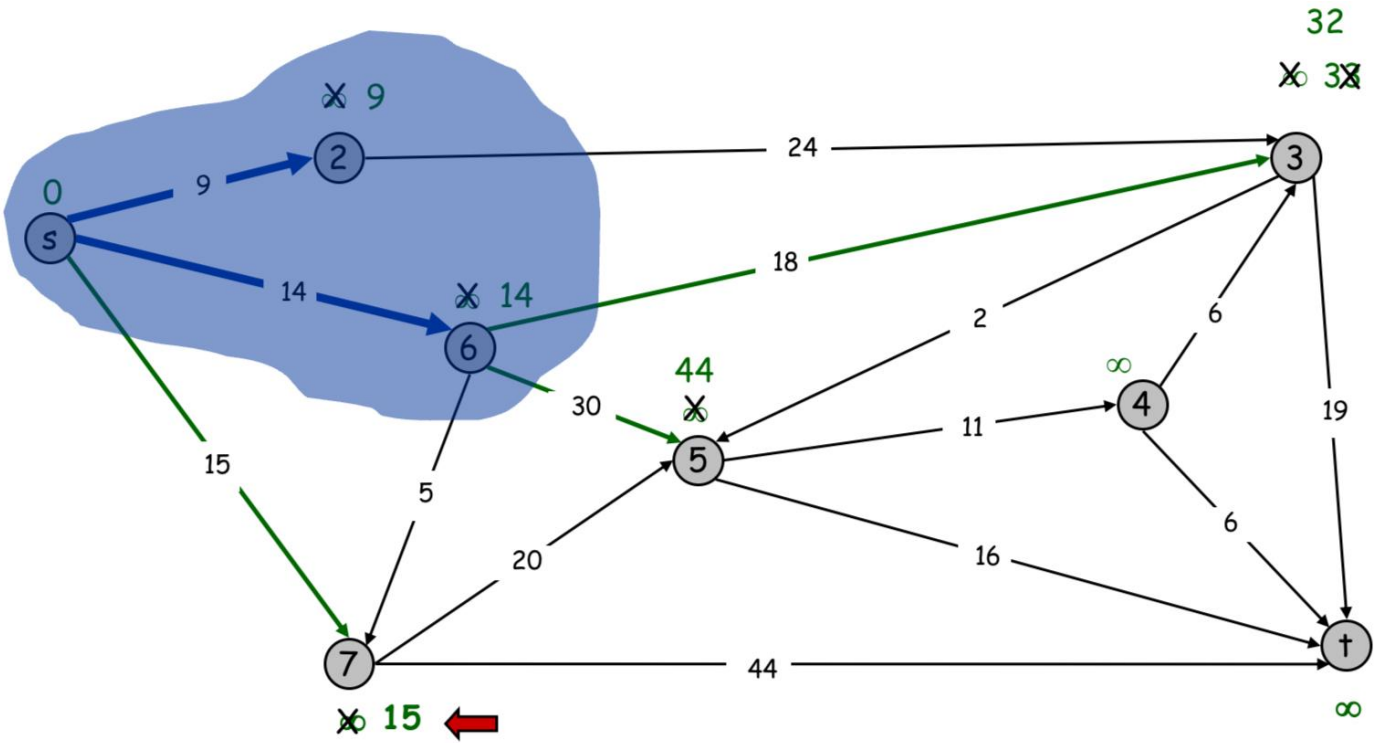
Dijkstra's algorithm



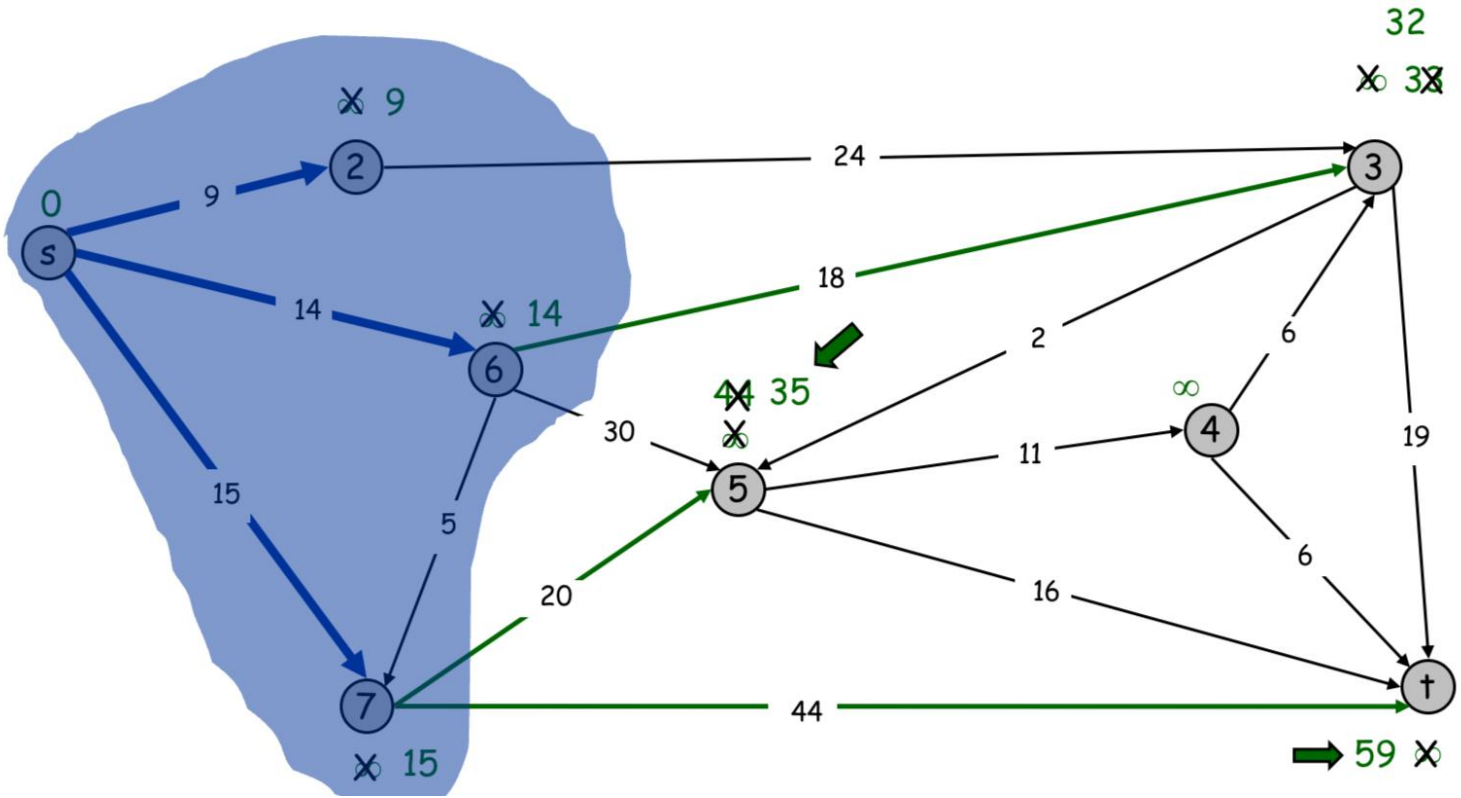
Dijkstra's algorithm



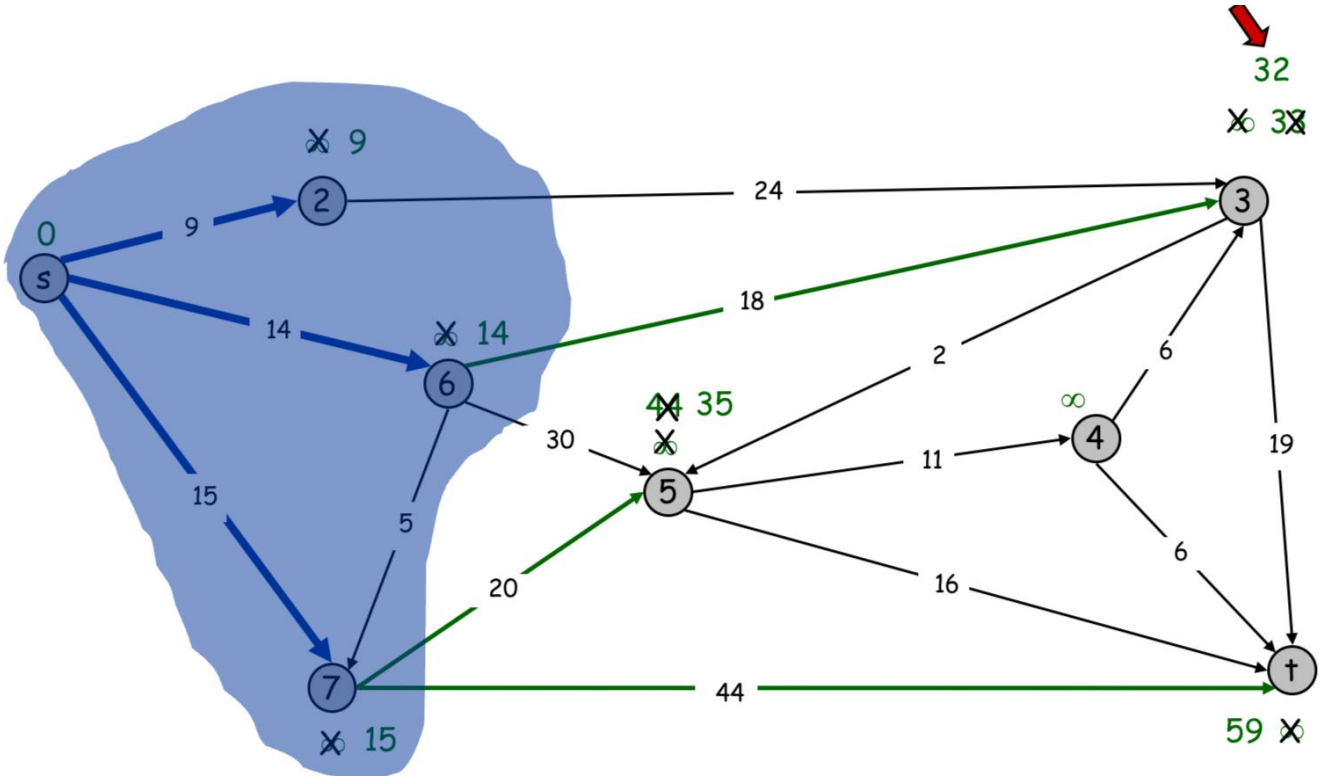
Dijkstra's algorithm



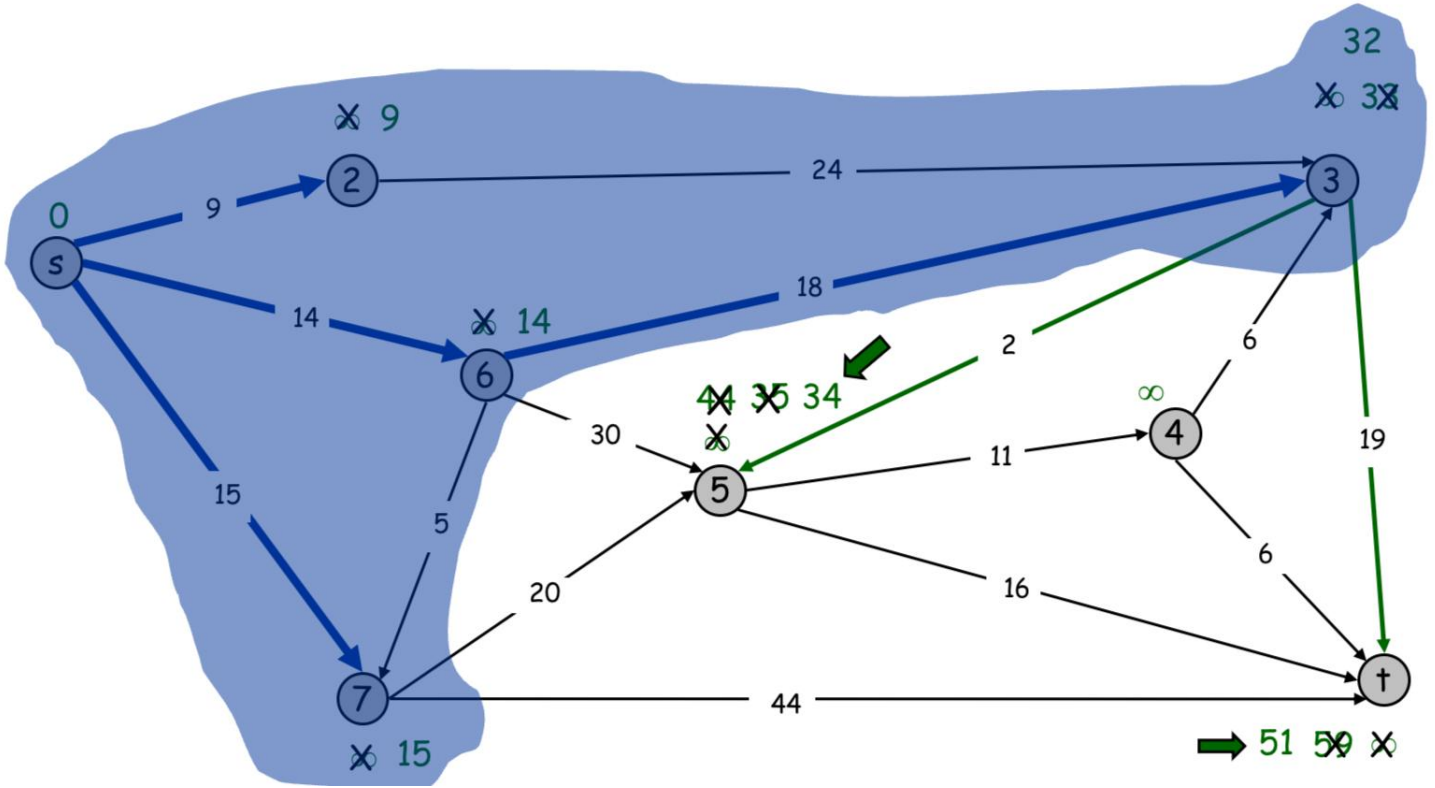
Dijkstra's algorithm



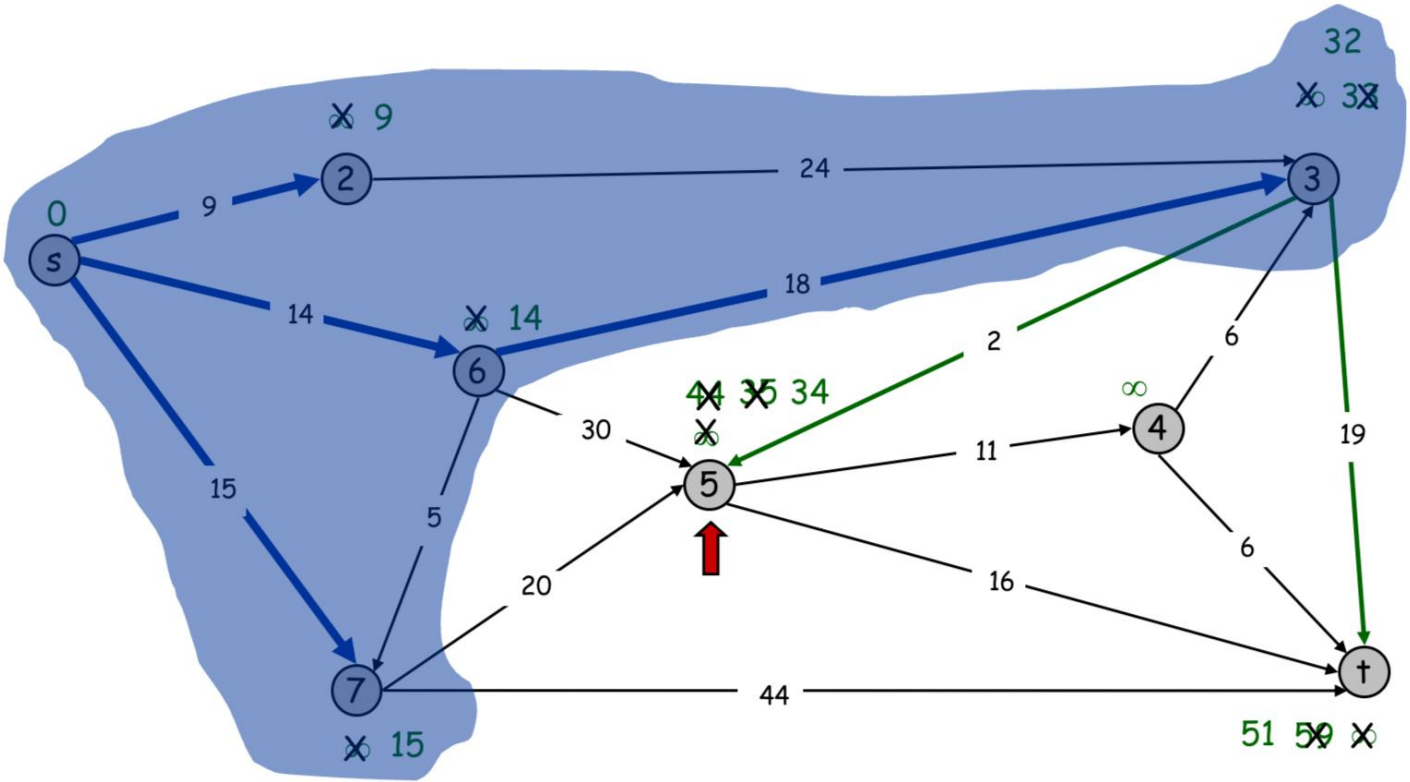
Dijkstra's algorithm



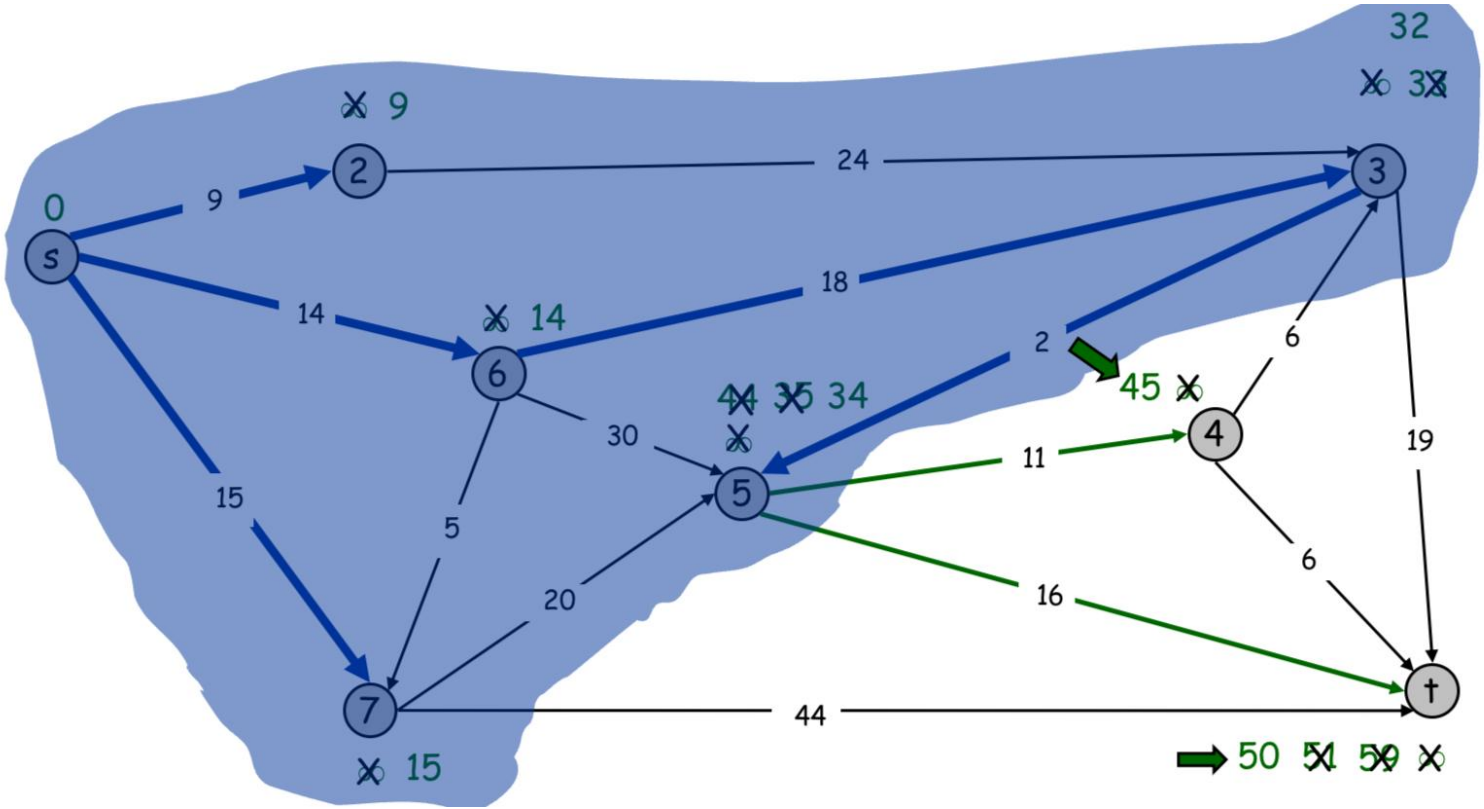
Dijkstra's algorithm



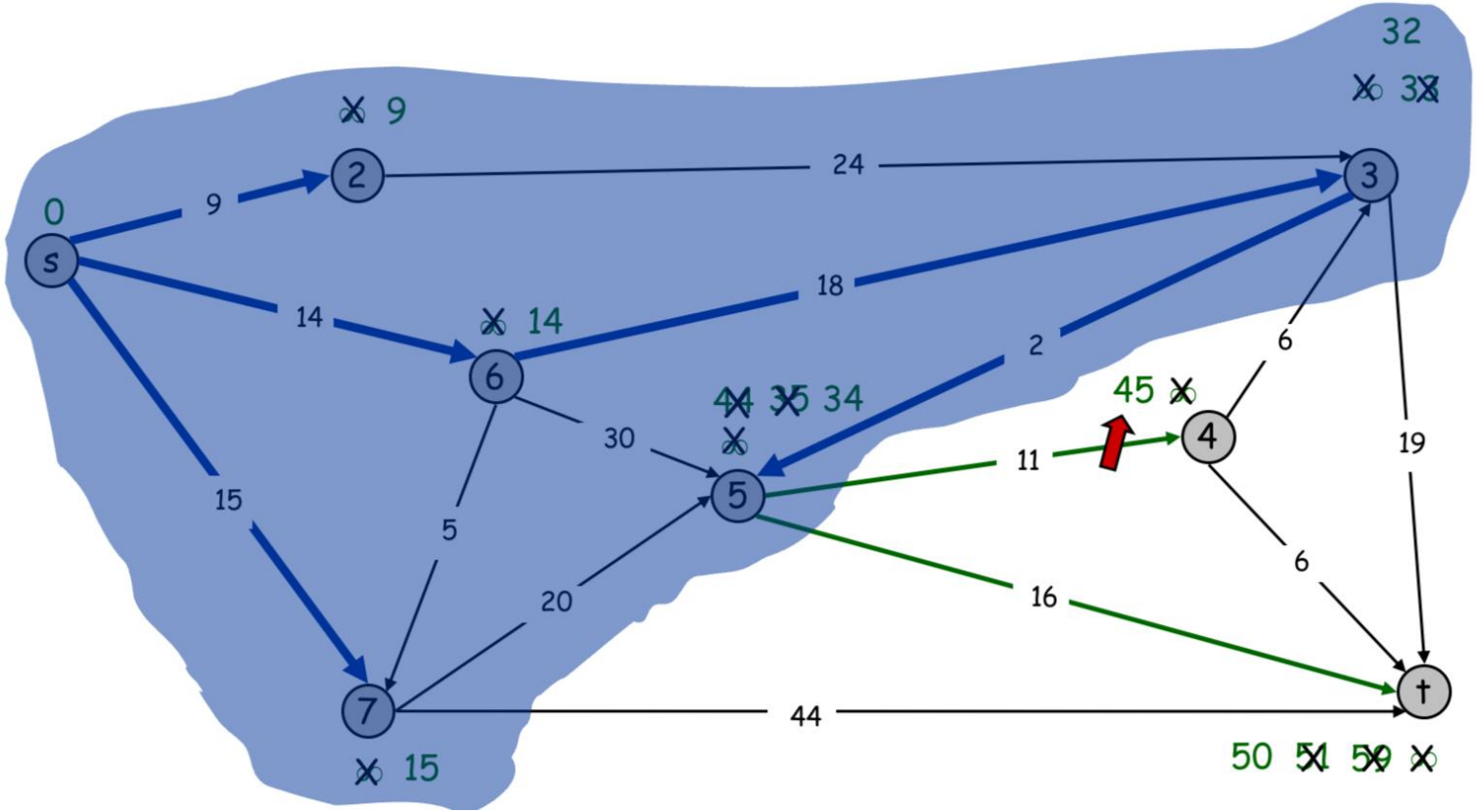
Dijkstra's algorithm



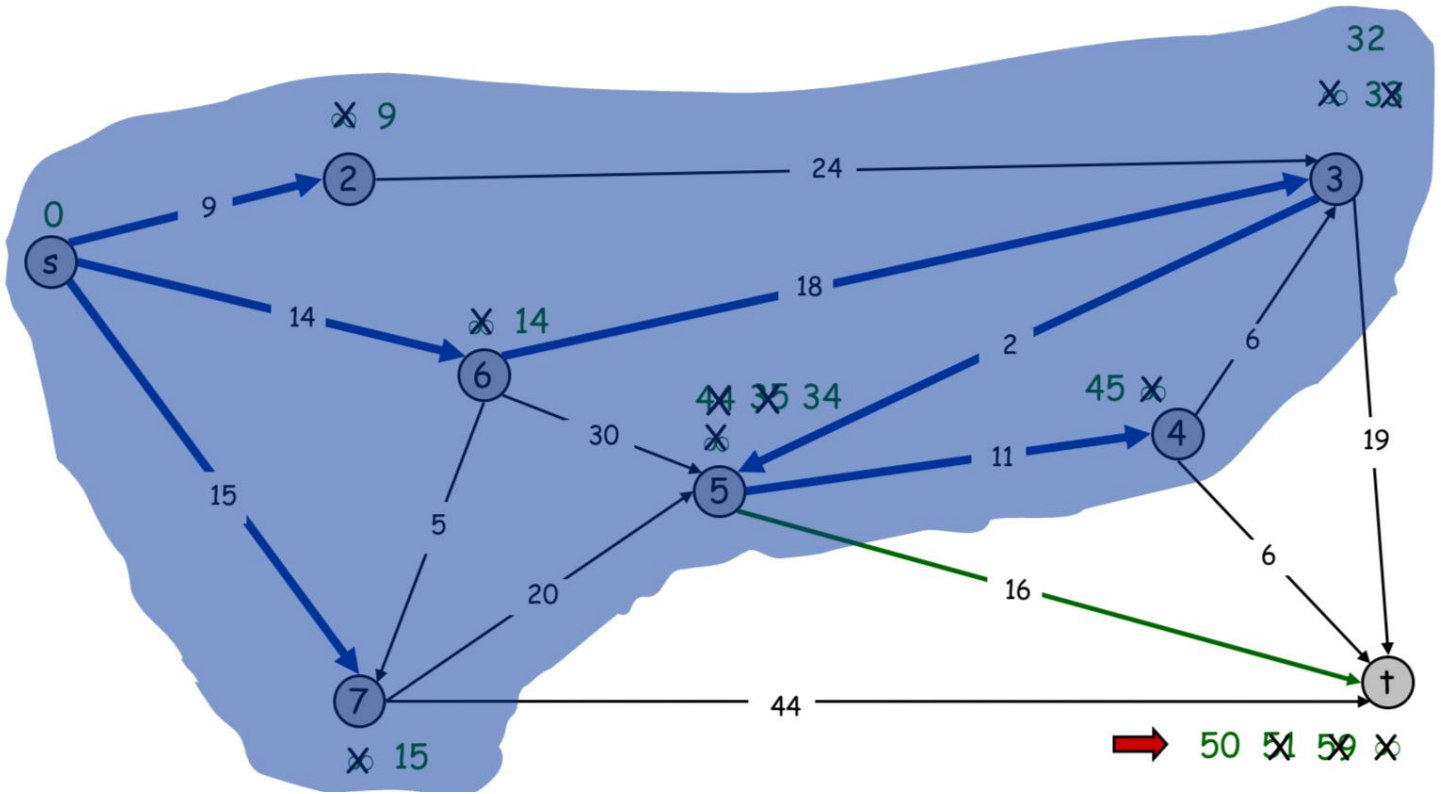
Dijkstra's algorithm



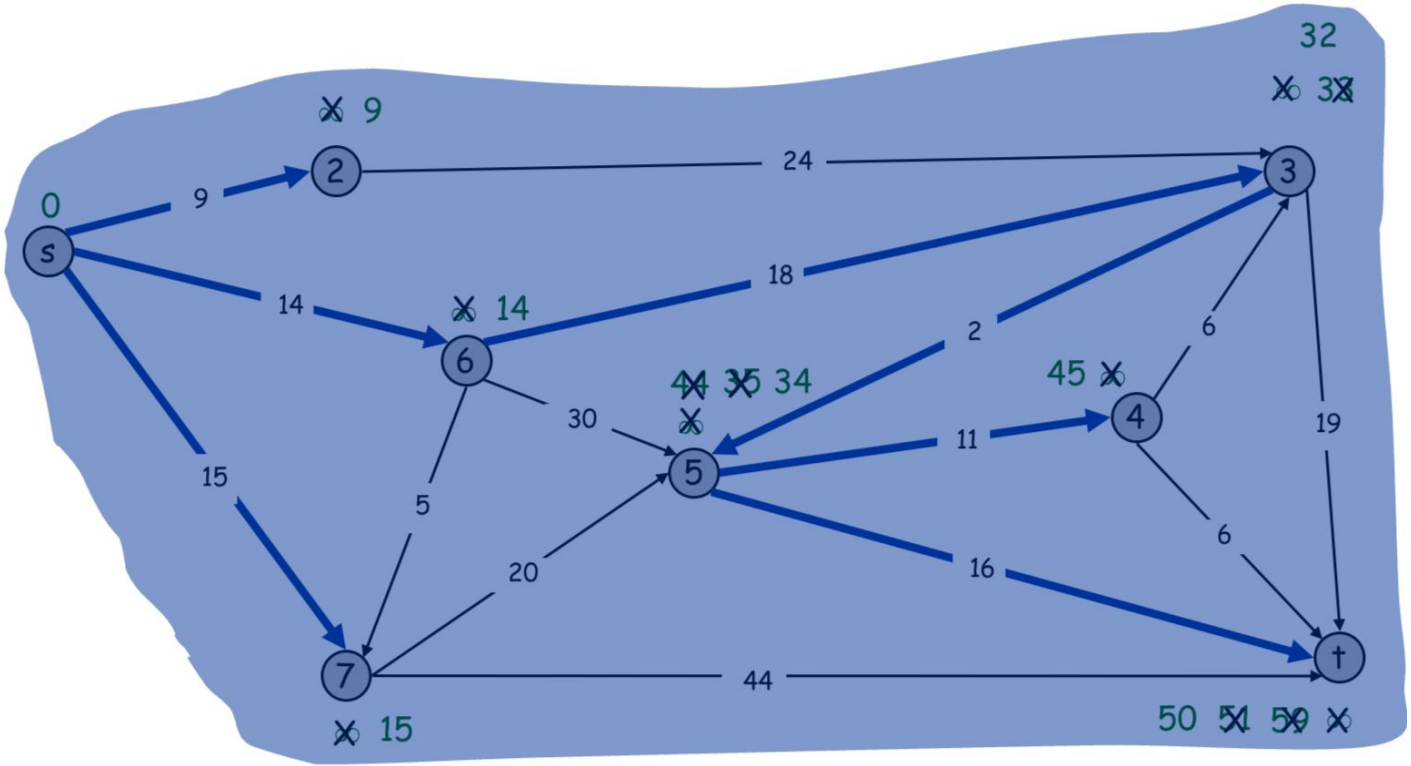
Dijkstra's algorithm



Dijkstra's algorithm



Dijkstra's algorithm



What path did we take for the min distance?

Pred map

As each node v is added to the set S, we simply record the edge (u, v) on which it achieved the minimum distance

(green edges in example)

```
dijkstra(G, s){
  for each (u in V) { d[u] = infinity }
  d[s] = 0 pred[s] = null
  Q = priority queue of all vertices u keyed by d[u]
  while (Q is not empty) {
    u = extractMin from Q
    for each (v in Adj[u]) {
      if (d[u] + w(u, v) < d[v]) {
        d[v] = d[u] + w(u, v)
        decrease v's key value in Q to d[v]
        pred[v] = u //keeps track of the tree
      }
    }
  }
}
```

Runtime Analysis

- Vertices $V \backslash S$ are stored in a priority queue via key value $d[u]$
- Priority queue operations (binary heap)
  - build $O(n)$
  - delmin (extract min) $O(logn)$
  - decrease key $O(logn)$
- $T(n,m) = n + n + \sum_{u \in V} (\log n + \deg(u) \cdot \log n)$
- $= 2n + \log n \sum_{u \in V} (1 + \deg(u))$
- $= 2n + \log n \, (n + 2m)$
- $= \Theta(2n + n \log n + 2mlogn) = O(m \log n)$

Is our choice of data structure important?

Dijkstra's V Op	Array	Binary Heap
Insert	1	$\log n$
ExtractMin	n	$\log n$
ChangeKey	1	$\log n$
IsEmpty	1	1
Total	n^2	$m \log n$

For an m = 10^10 edges and n = 10^9 vertices, this is a difference of 6 min vs 3k years

Proof of Correctness

Dijkstra's algorithm is greedy

Greedy criteria: we always form the shortest new s-v path we can make from a path in S followed by a single edge

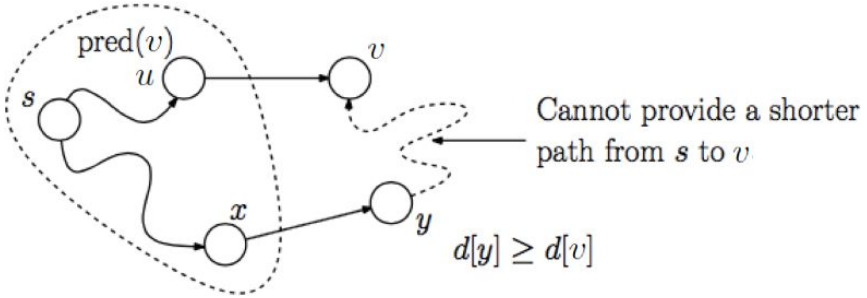
We can prove its correctness by showing that it always *stays ahead* of the optimal solution

Each time it selects a path to a node v, that path is shorter than every other possible path to v

Proof of Correctness

- Need to show that $d[v] = \delta(s, v), \forall v \in V$
- Invariant: $d[v] = \delta(s, v), \forall v \in S$
- Proof by induction on $|S|$
  - Base case: $|S| = 1, \delta(s, s) = 0$
  - Assume true for $|S| = k > 1$
  - $|S| = k + 1$

Proof of Correctness



- Let v be the next vertex added to S, along with (u, v)
- $d[v] = \delta(s, u) + w(u, v)$
- Consider any other $s - v$ path P. let (x, y) be the first edge taken by P where $x \in S$ and $y \in V \backslash S$
- $d[x] = \delta(s, x)$
- $d[y] \geq d[v]$
- $\text{len}(P) > \delta(s, x) + w(x, y) = d[y] \geq d[v]$

Full proof posted on resources tab

Summary

1. Djikstra's algorithm
   a. Finds shortest paths from s to all other nodes
   b. Greedy algorithm - forms a path from S to the next shortest edge
   c. Runtime O(mlogn) with a heap
   d. Prove by showing your solution "stays ahead" at each step

1. Upcoming deadlines:
   a. Lab 2 due tomorrow
   b. HW3 due Monday
   c. Submit your Lecture 6 class assignment

2. Next class: Minimum Spanning Trees