# CSE 421

## Greedy Algorithms / Dijkstra's Algorithm

Yin Tat Lee

# Homework 1 Comments

- Except for EC, each question should take less than 1 page.

(You don't have so much time in midterm/final/interviews)

- Q2: $(\log n)^{\sqrt{\log n}} < 2^{(\log n)^{2/3}}$

$(\log n)^{\sqrt{\log n}} = 2^{\sqrt{\log n} \log \log n} \sim 2^{\sqrt{\log n}}$. $\sqrt{\log n}$ is less than $(\log n)^{2/3}$

- Q3: Instead of $(n-1)/3$, one can get $(n-1)/5$

Induction on the number of vertices.

- Q4: Reduction-type Question

Algo: Transform input. Call the class algorithm. Transform output.

Proof: Why the input is valid. Why the output is what we want.

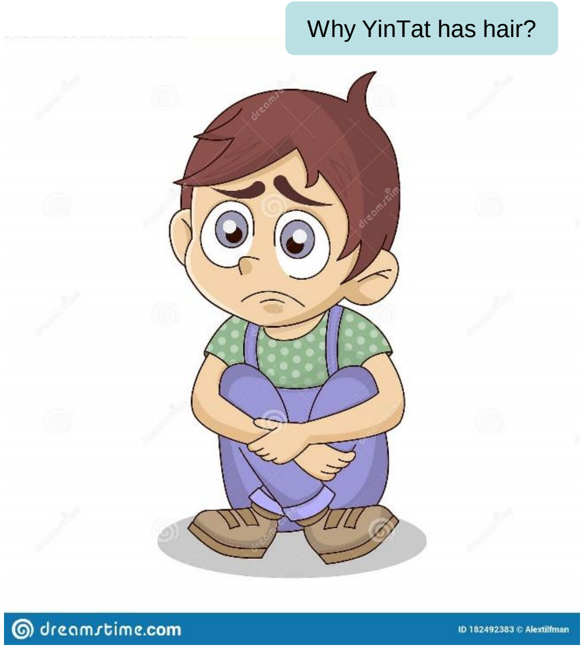
# Office Hour

- Please do HW earlier.

- You can start doing HW once it is announced.
  (I won’t ask things that hasn’t covered)

- HW3 is out

- If no time slot works for you,
  fill in this [https://bit.ly/3fCEgaU](https://bit.ly/3fCEgaU)




TAs and YinTat are lonely

# Experiment Y

Axiom:

- My teaching needs improvement.
- I care about teaching.
- Improvements are possible.

Algorithm:

- For each week

  I will make a poll to collect suggestions.

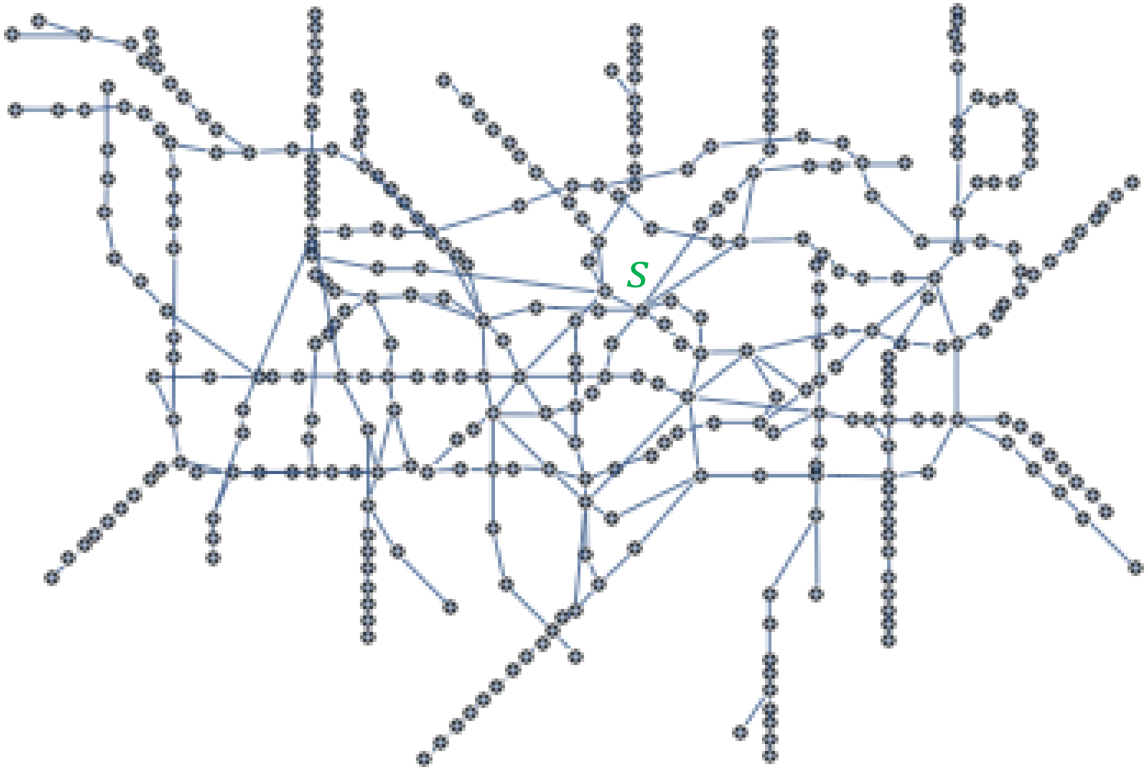
  I will implement the suggestion with the most hearts.

- For the benefit of everyone, please participate.

# Single Source Shortest Path

Given an (un)directed graph $G = (V, E)$ with non-negative edge weights $c_e \geq 0$ and a start vertex $s$.

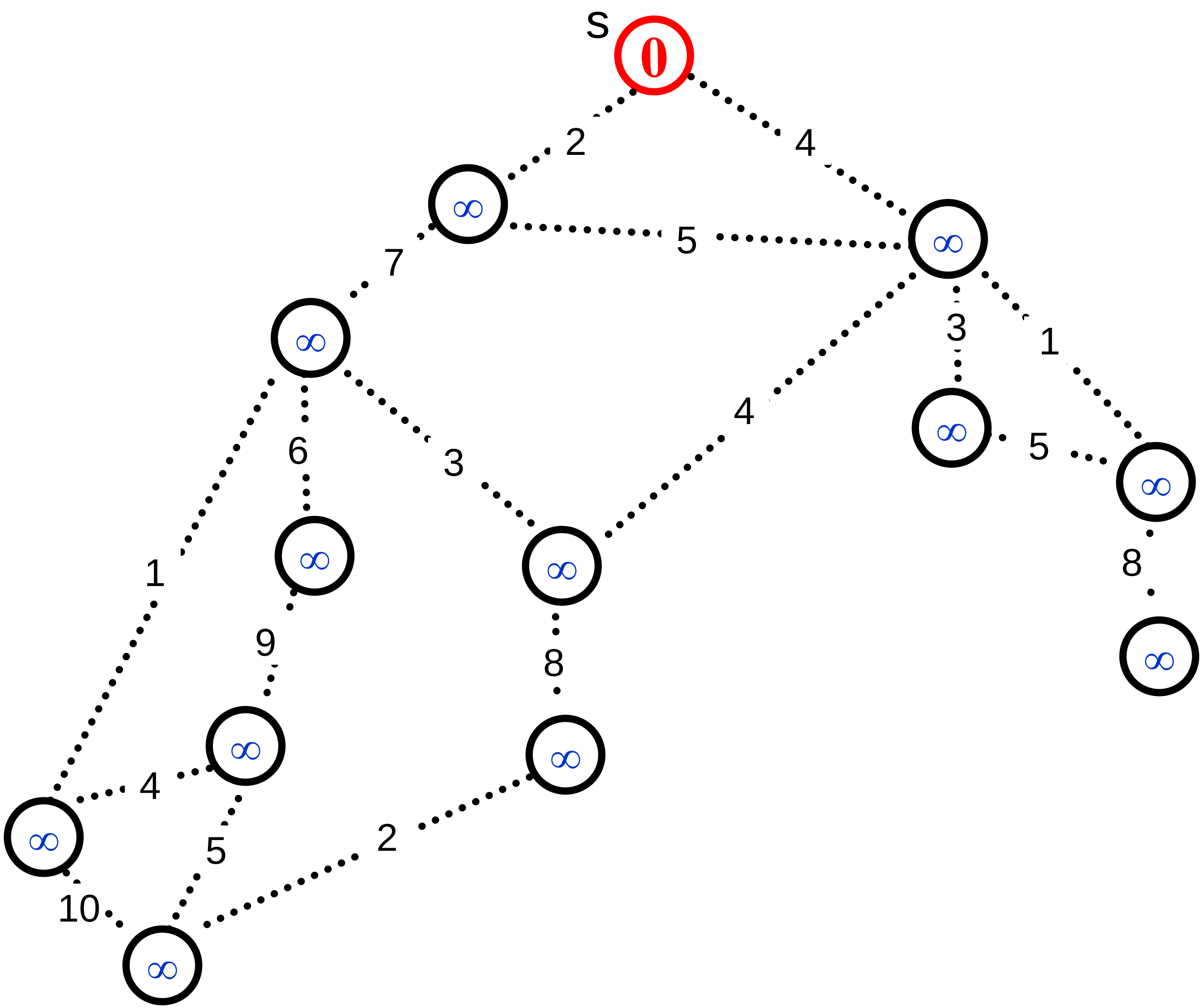
Find length of shortest paths from $s$ to each vertex in $G$



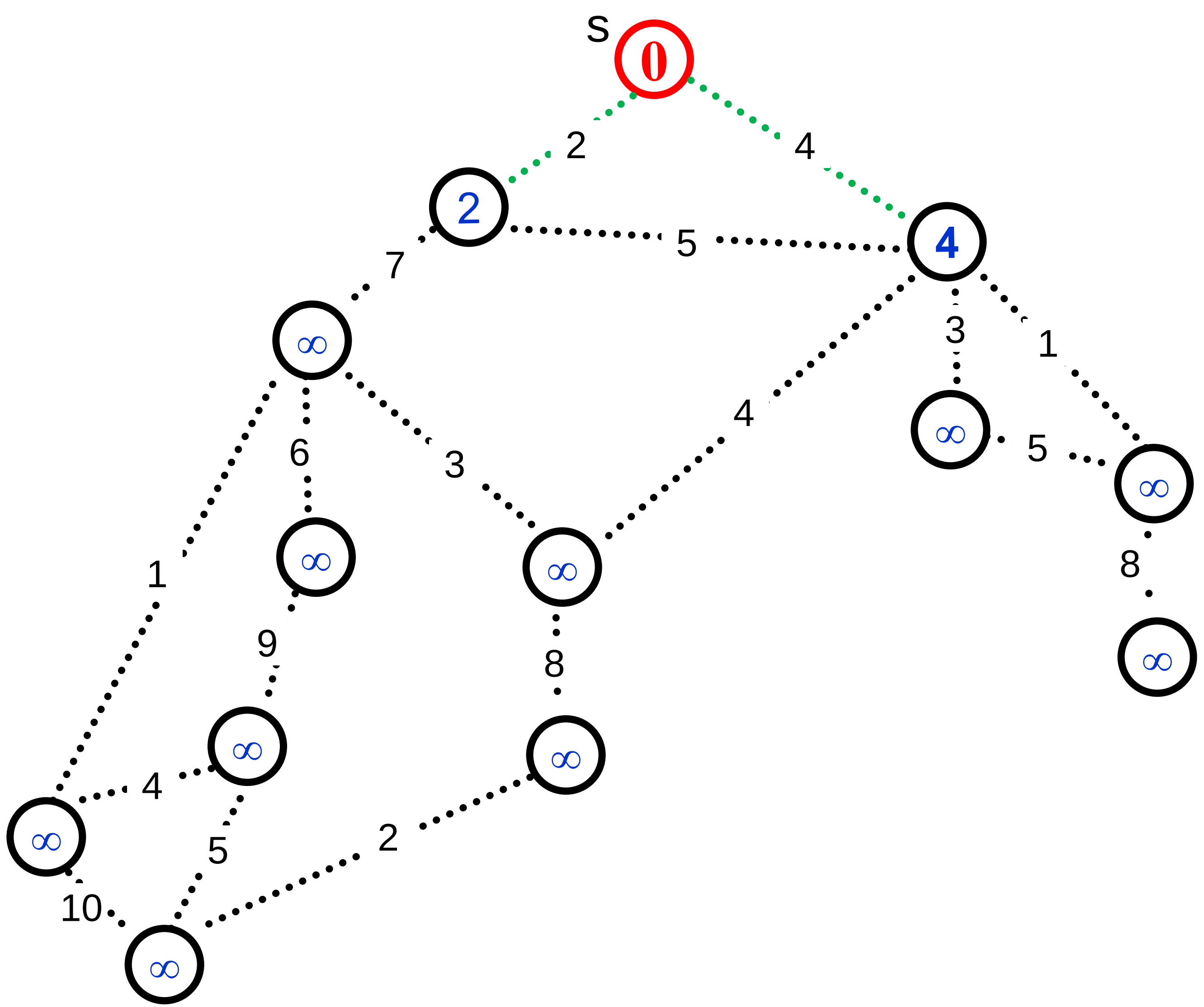
```
Dijkstra(G,c,s) {
   Initialize set of explored nodes S ← {s}

   // Maintain distance from s to each vertices in S
   d[s] ← 0

   while (S ≠ V)
   {
      Pick an edge (u,v) such that u ∈ S and v ∉ S and
            d[u] + c_(u,v) is as small as possible.

      Add v to S and define d[v] = d[u] + c_(u,v).
      Parent(v) ← u.
   }
```

Set $S$ is all vertices to which we have found the shortest path.

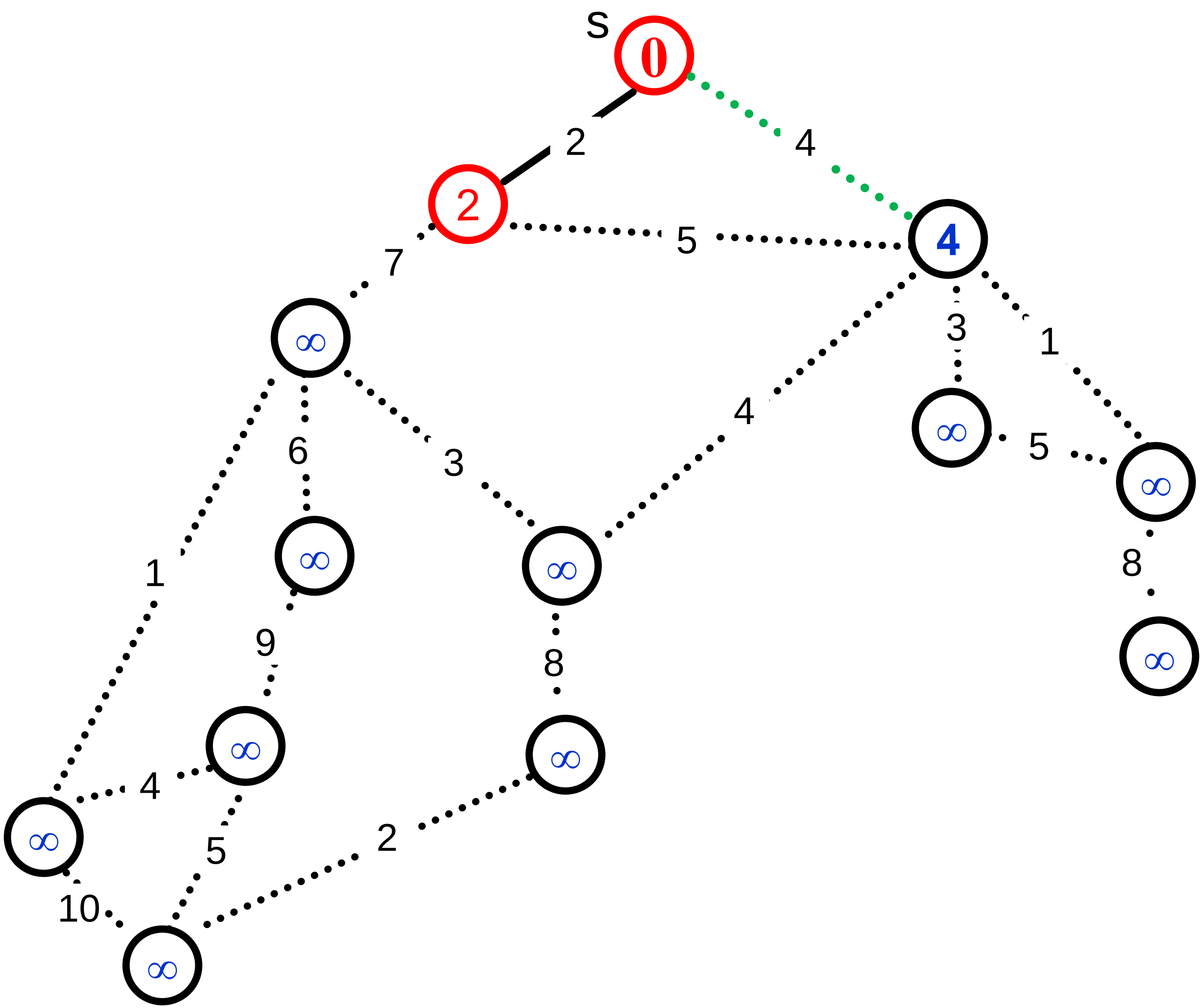
# Dijkstra's Algorithm: Example



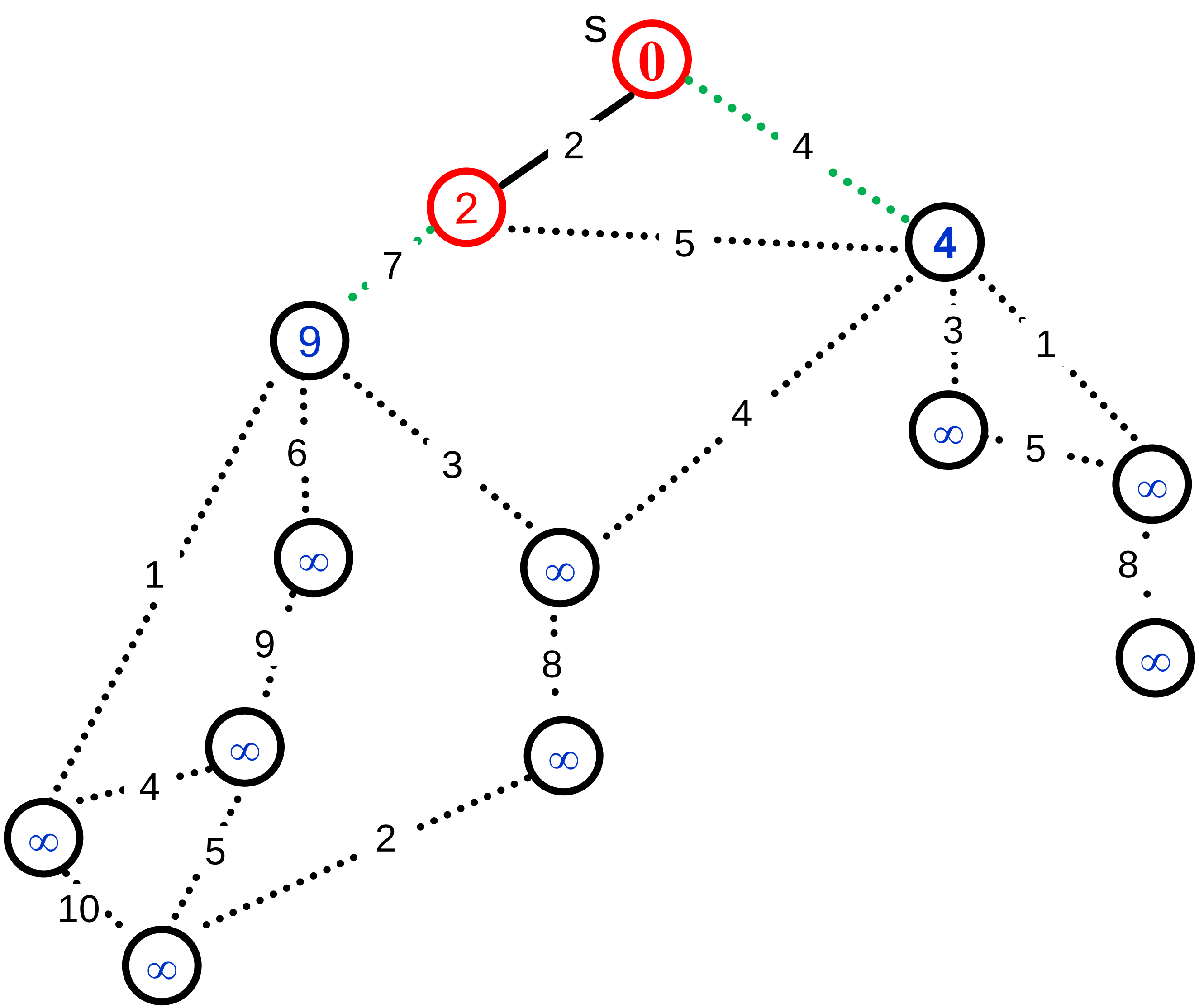
# Dijkstra’s Algorithm: Example



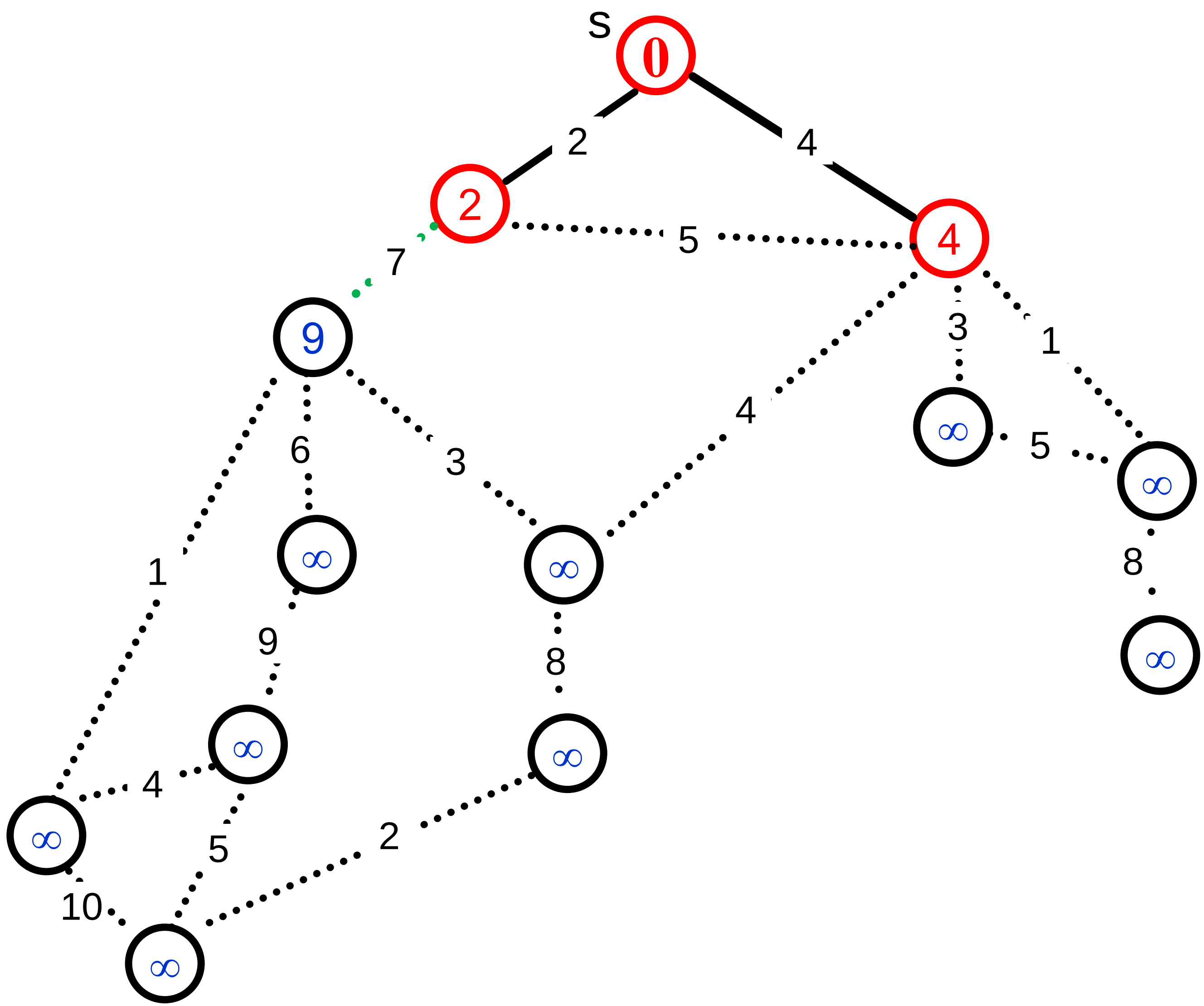
# Dijkstra's Algorithm: Example



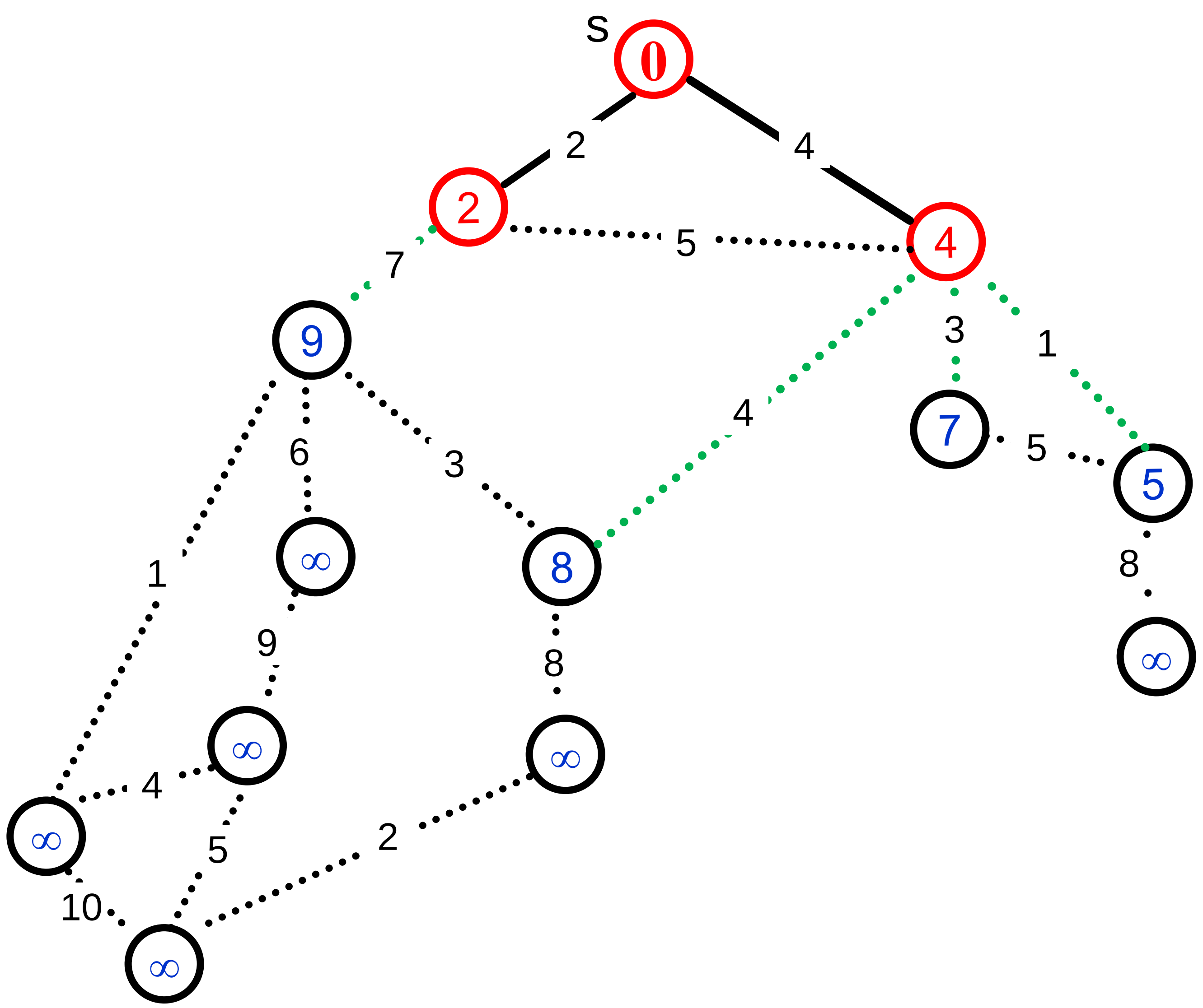
# Dijkstra's Algorithm: Example



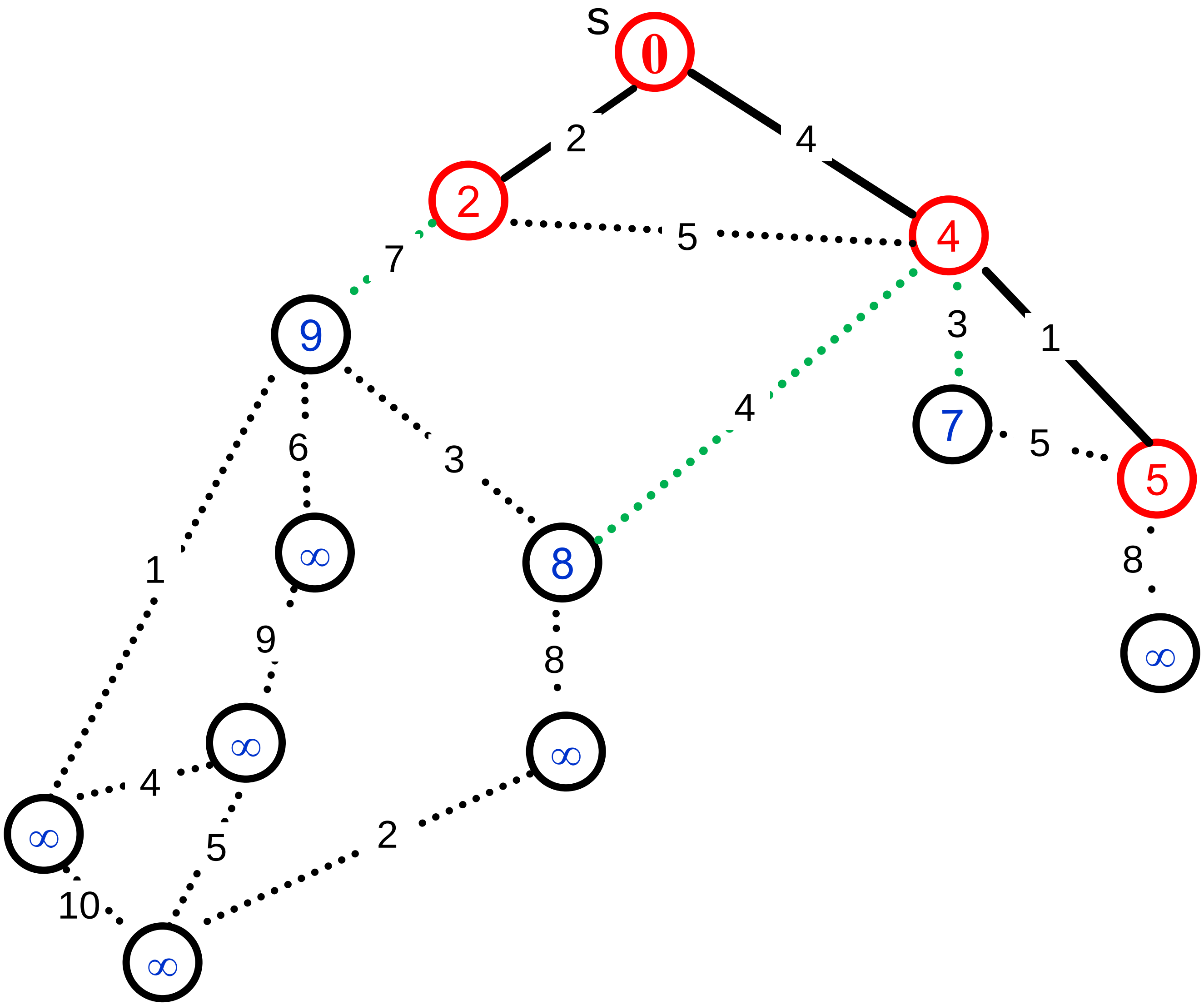
# Dijkstra's Algorithm: Example



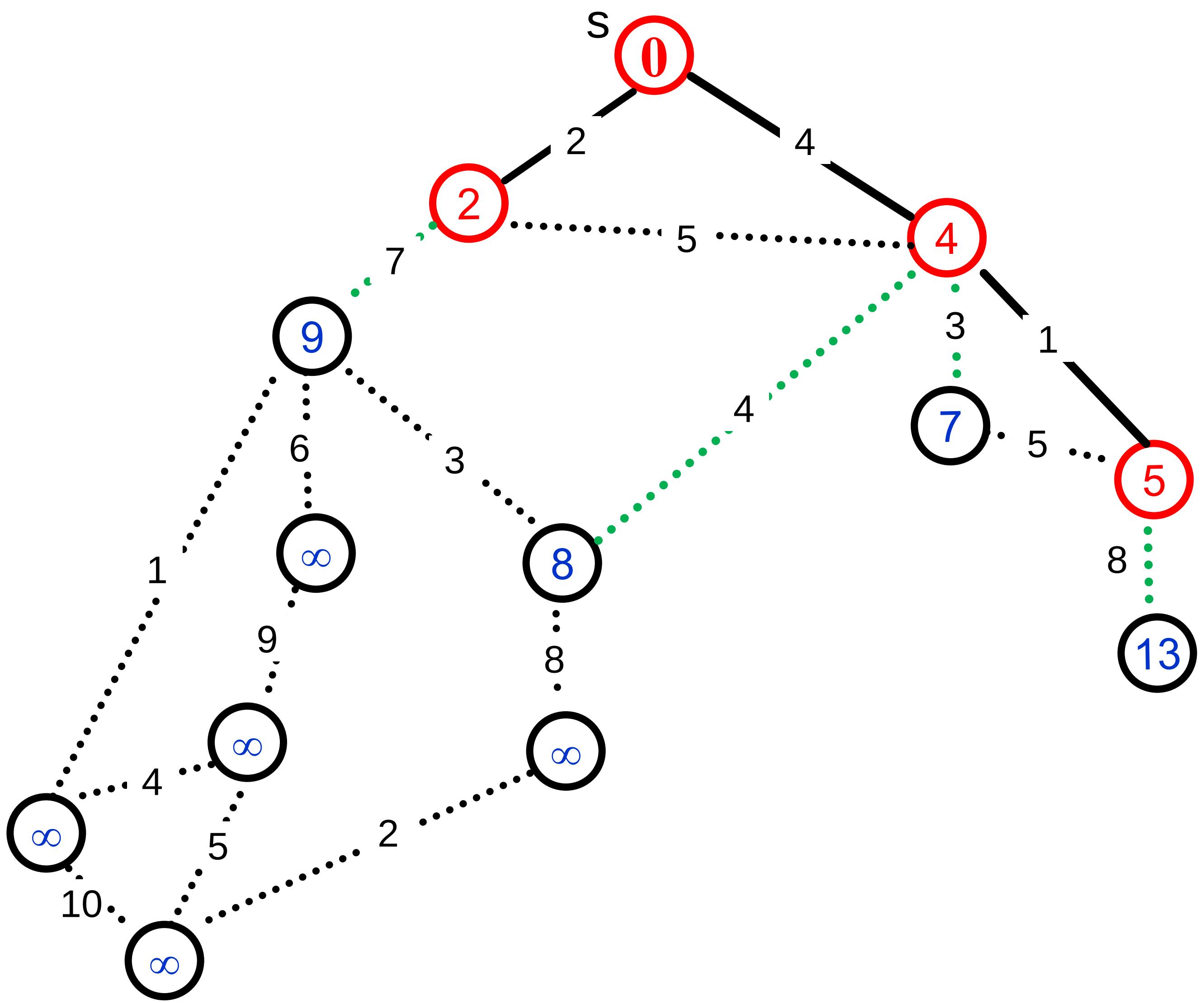
# Dijkstra's Algorithm: Example



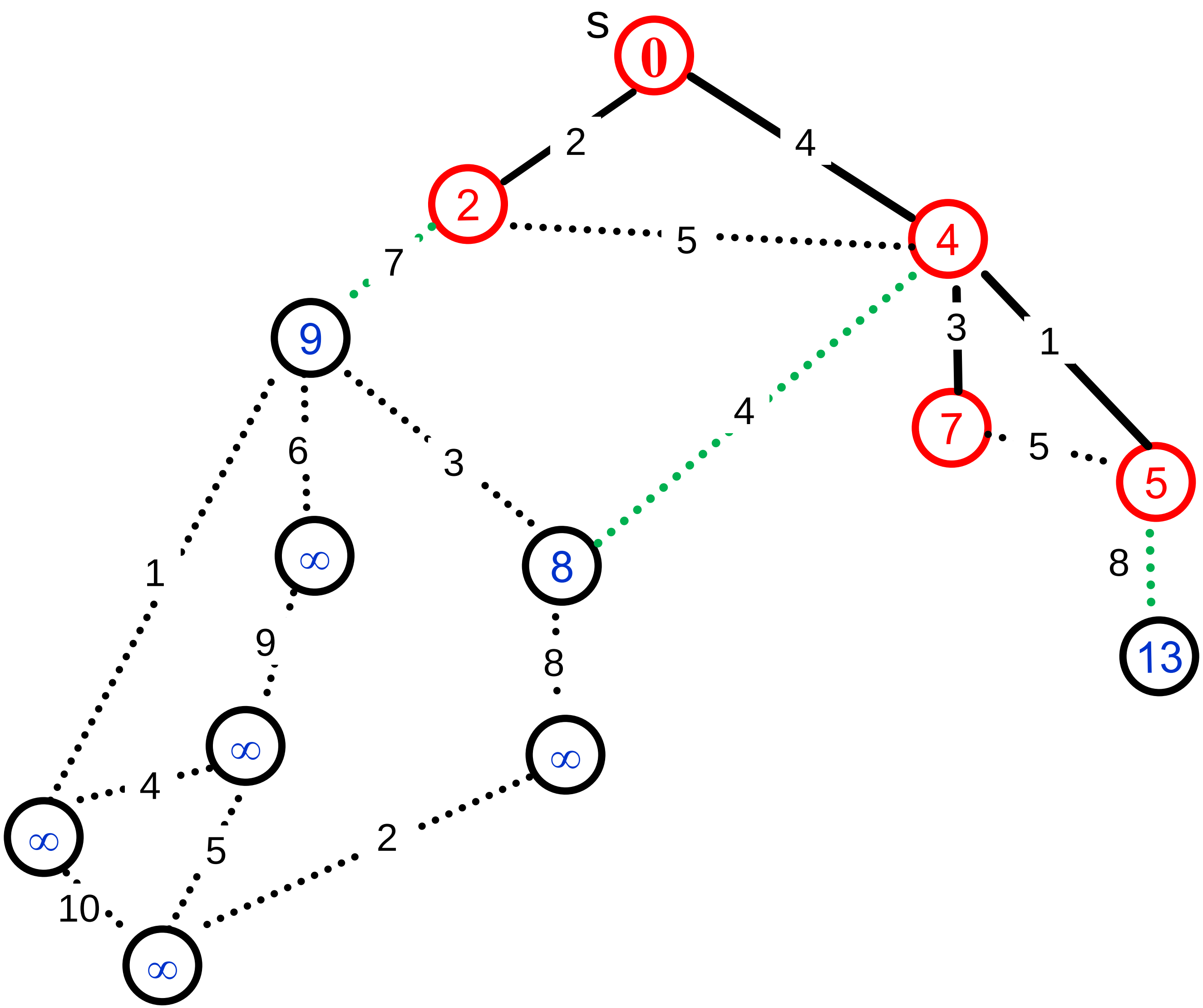
# Dijkstra's Algorithm: Example



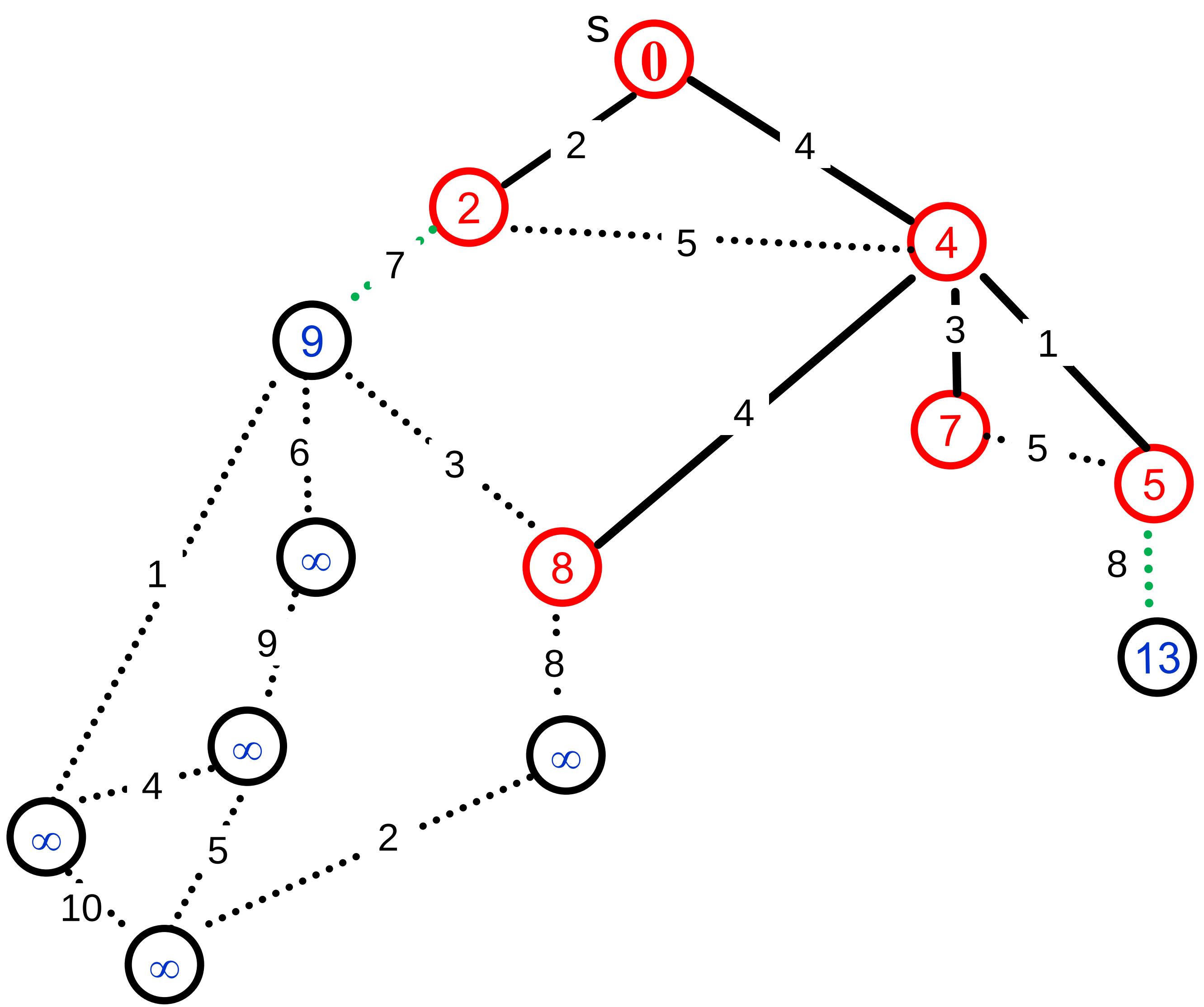
# Dijkstra's Algorithm: Example



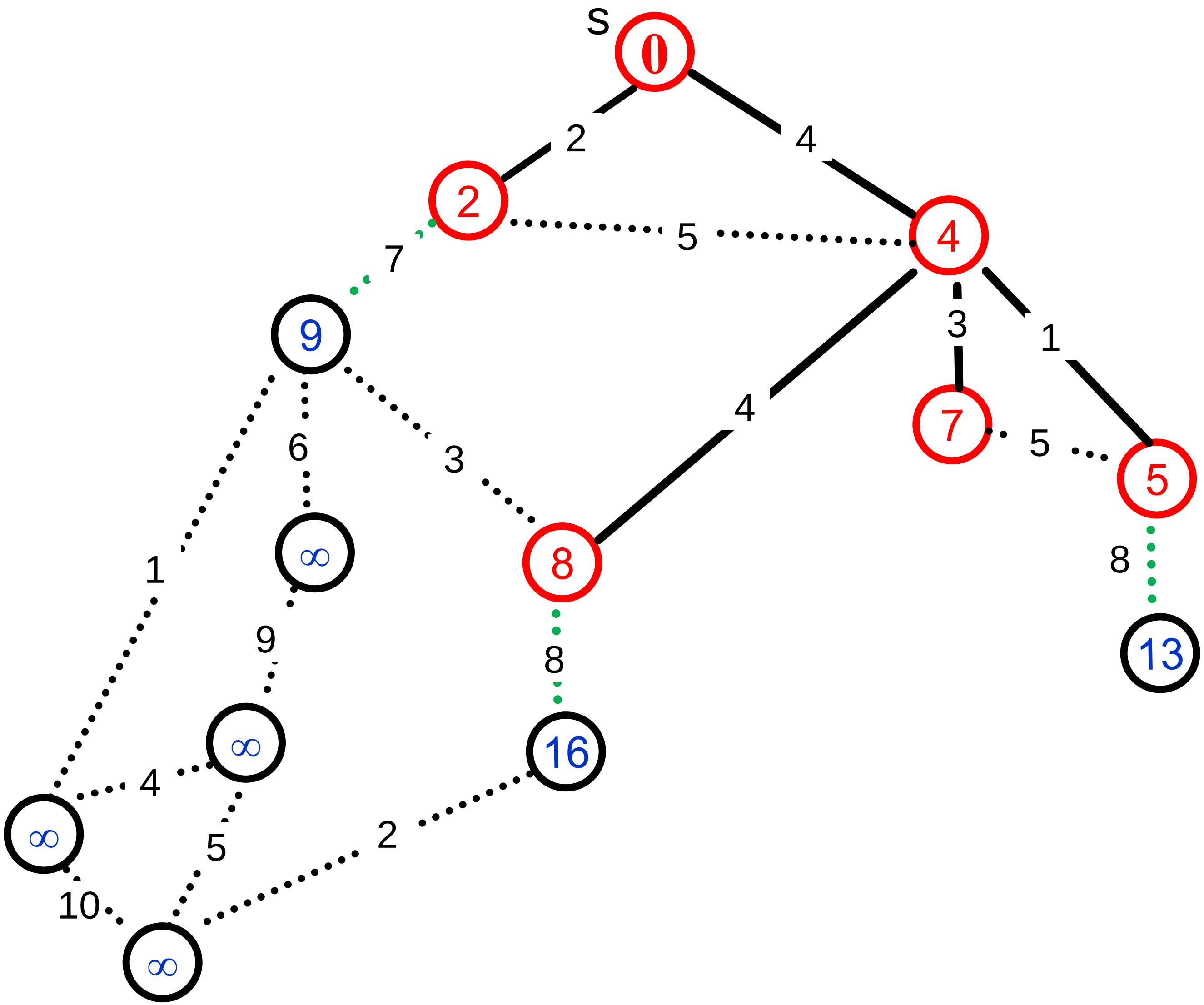
# Dijkstra's Algorithm: Example



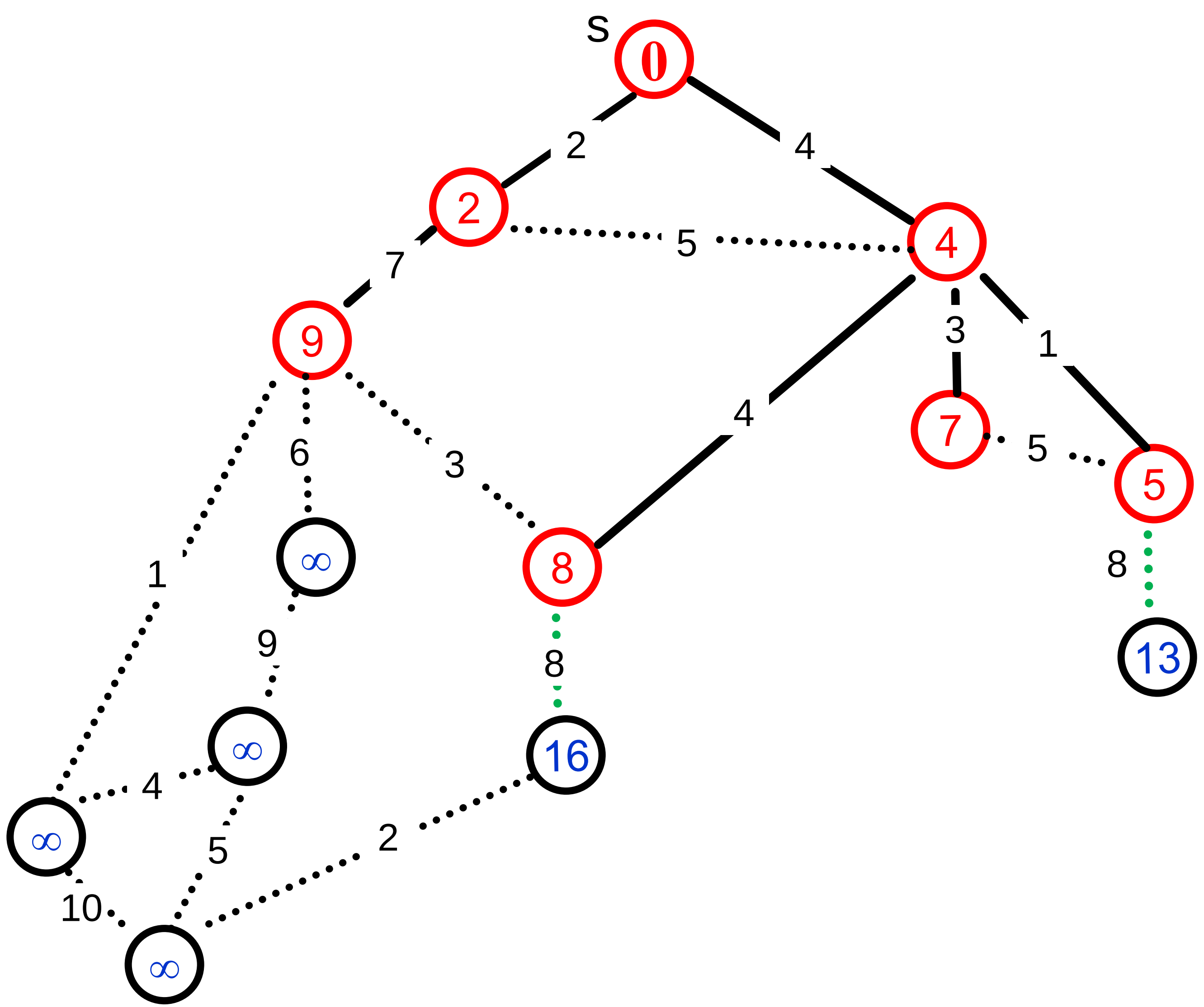
# Dijkstra's Algorithm: Example



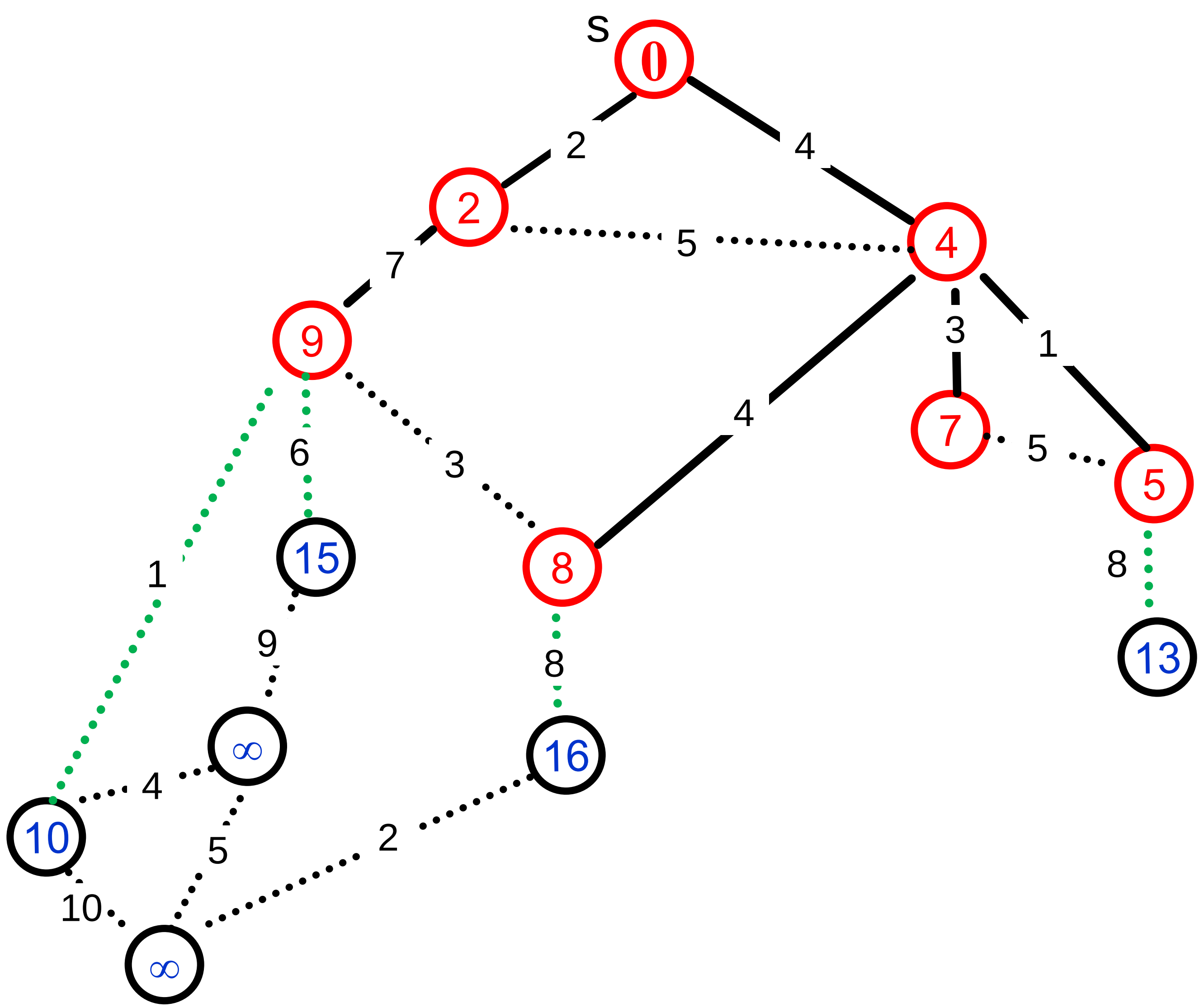
# Dijkstra’s Algorithm: Example



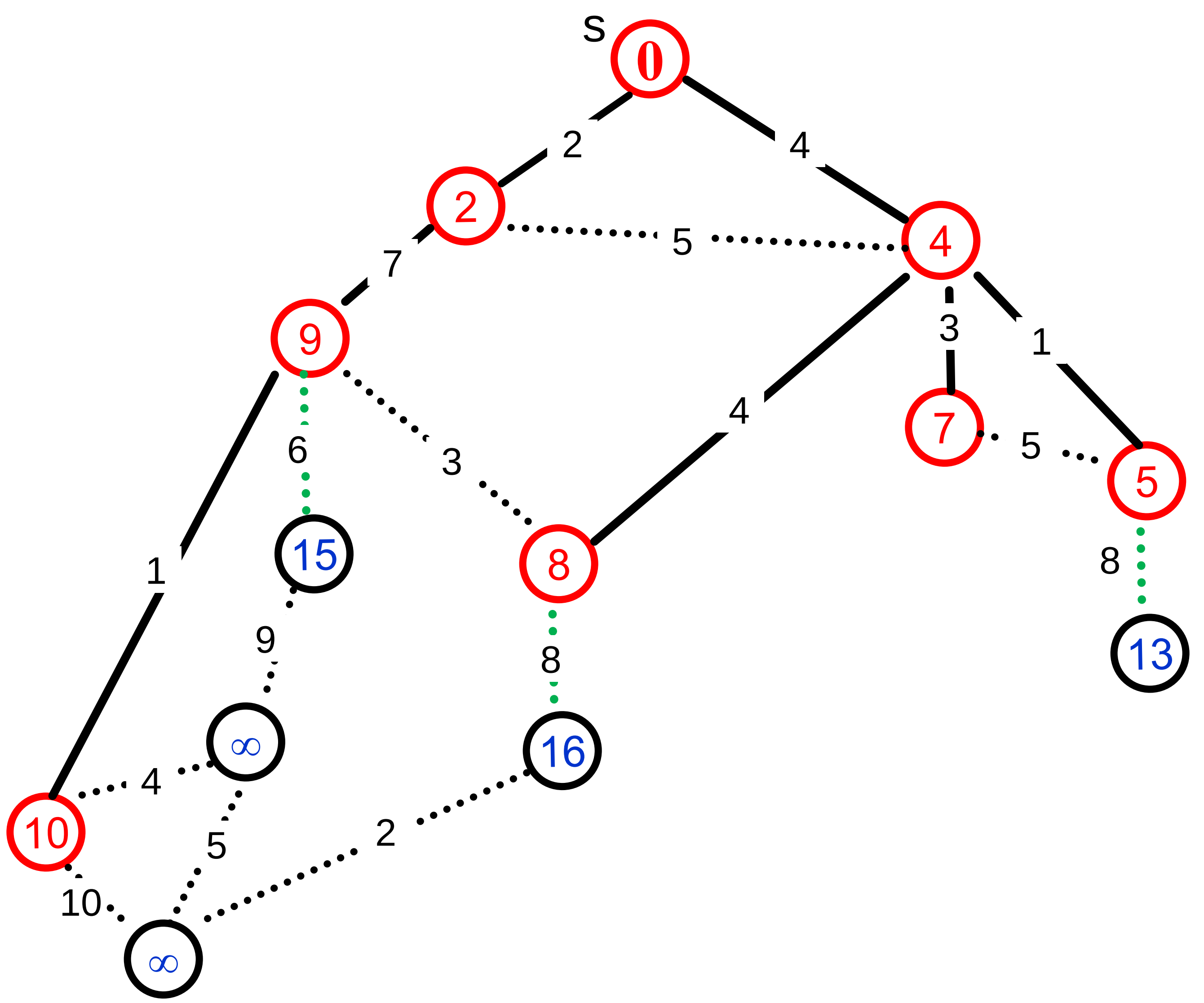
# Dijkstra's Algorithm: Example



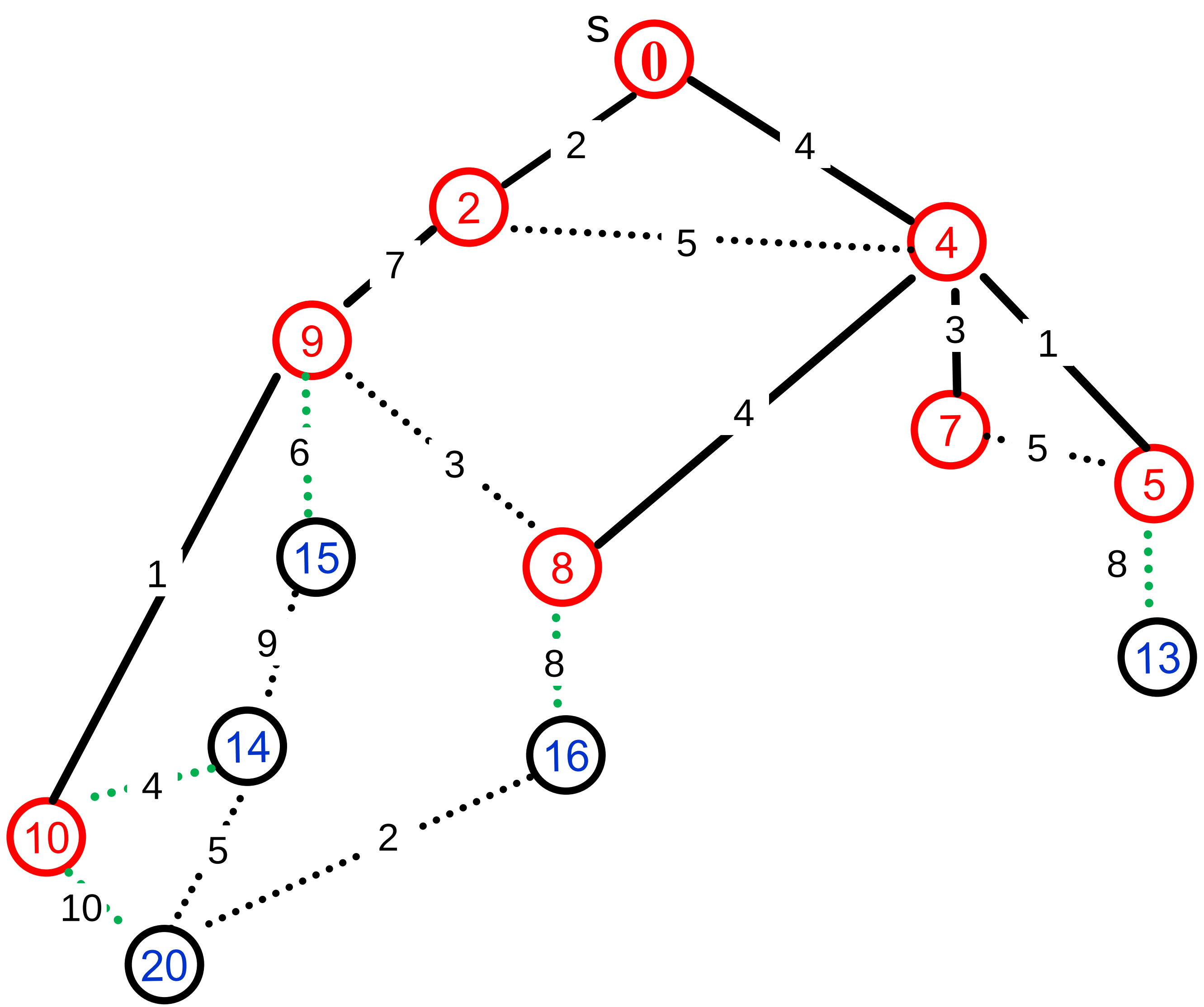
# Dijkstra's Algorithm: Example



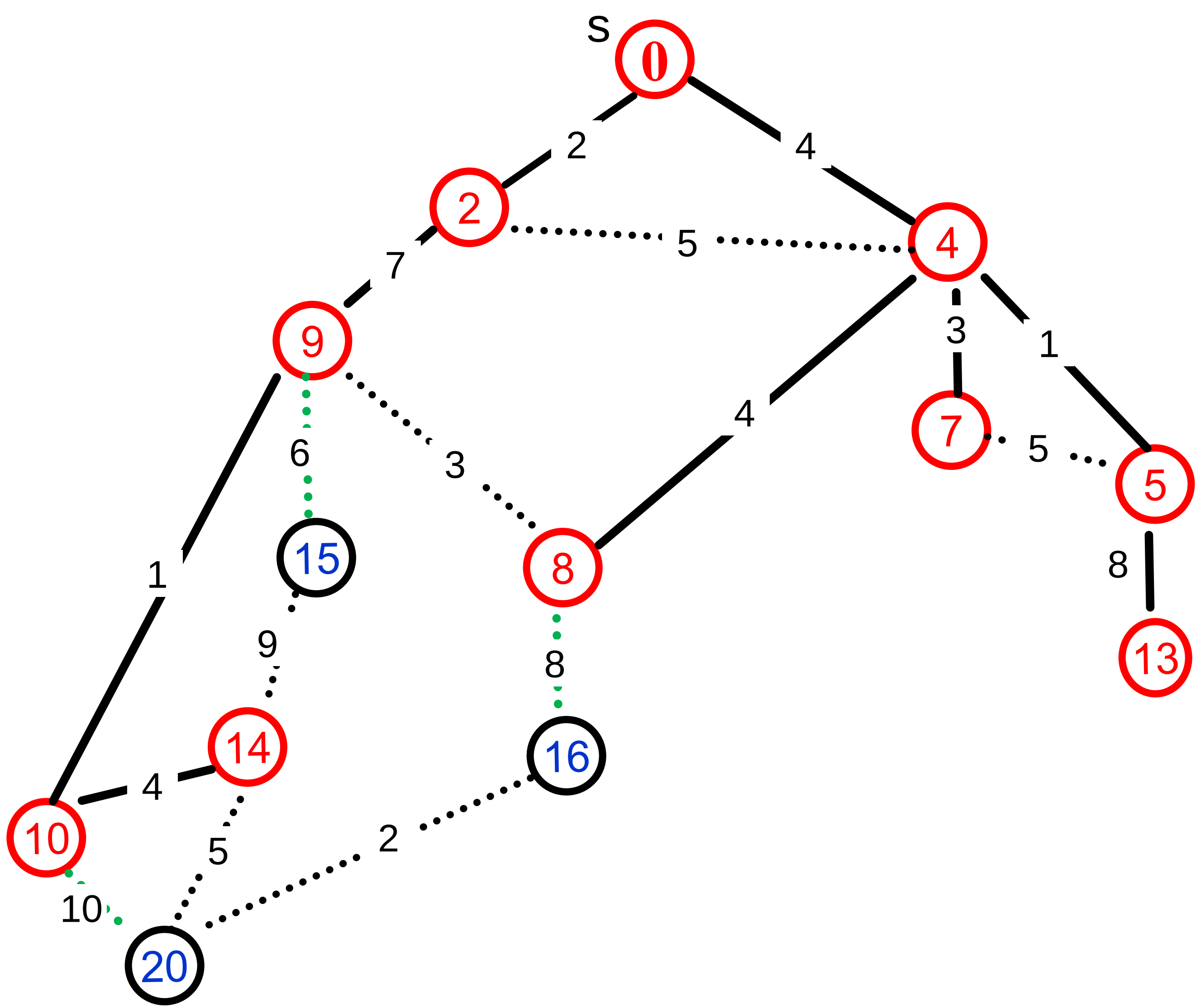
# Dijkstra's Algorithm: Example



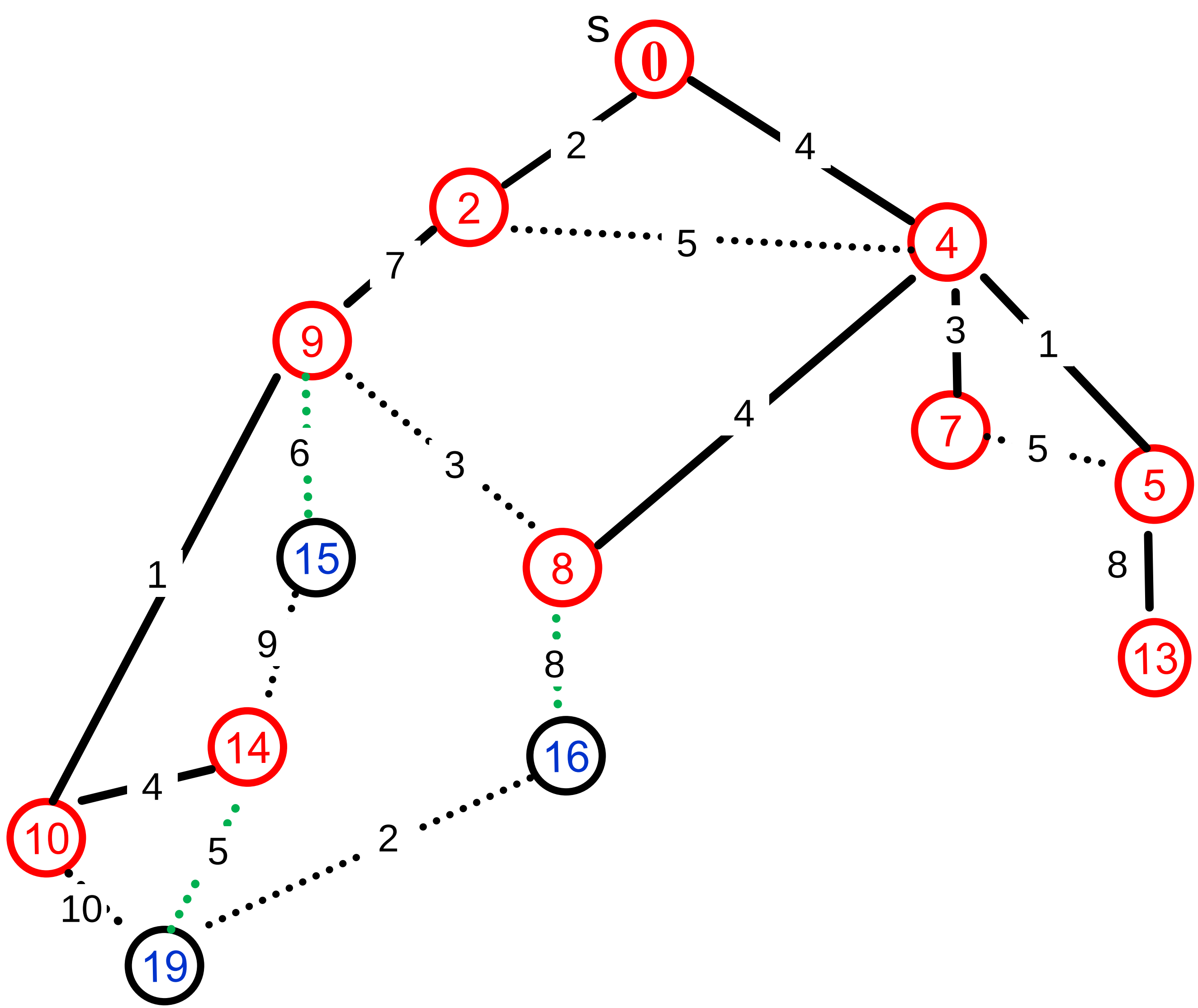
# Dijkstra’s Algorithm: Example



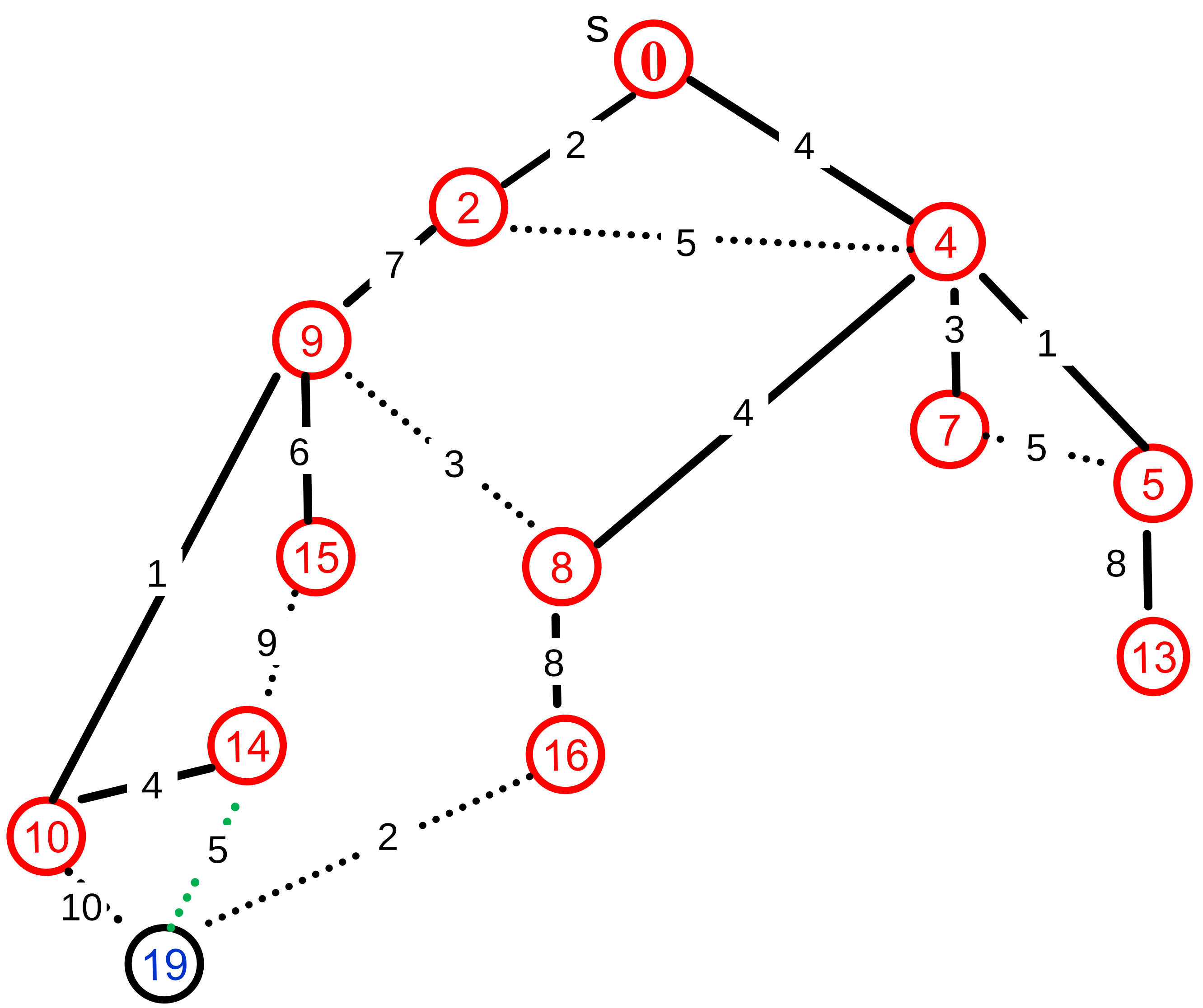
# Dijkstra’s Algorithm: Example



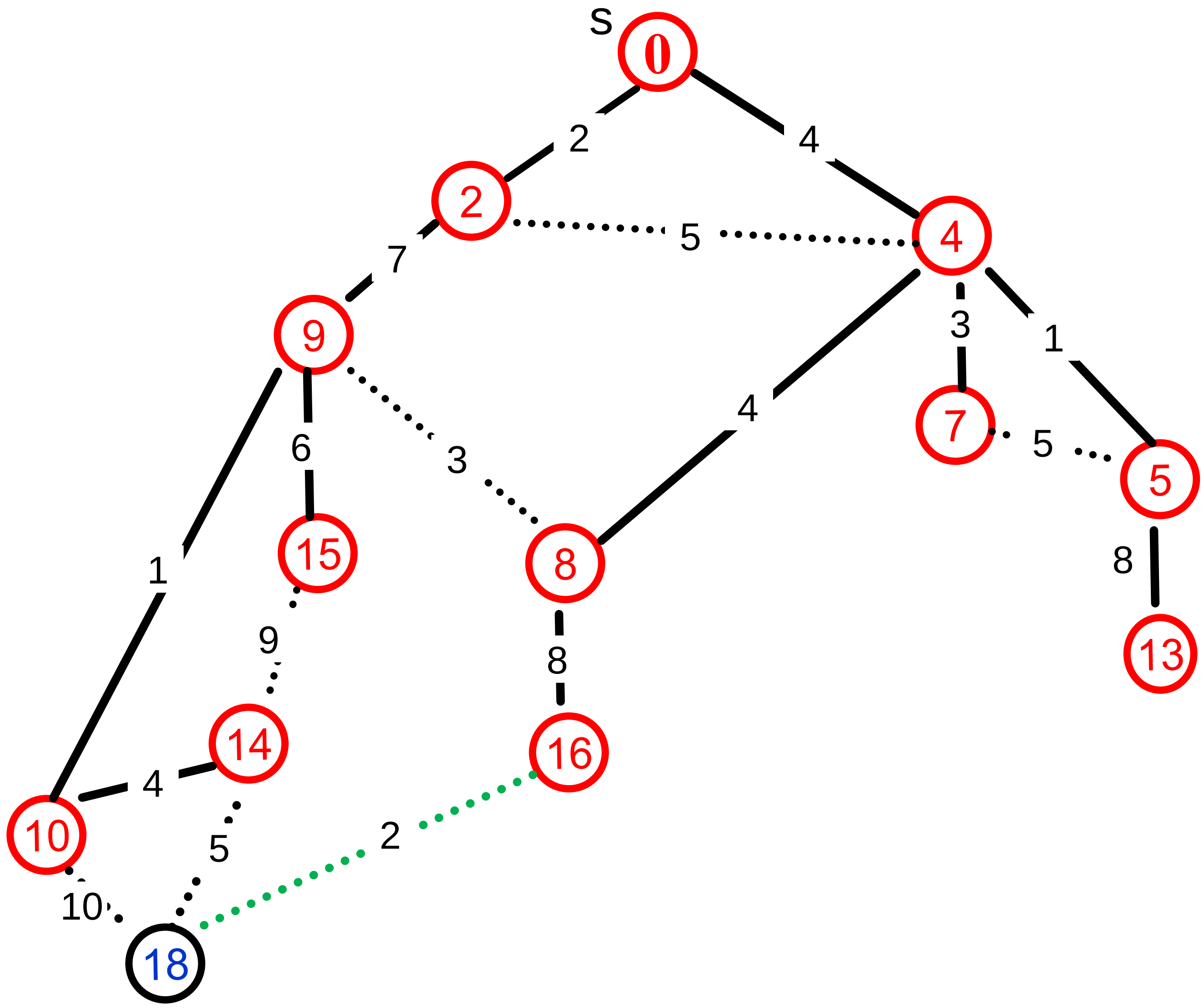
# Dijkstra's Algorithm: Example



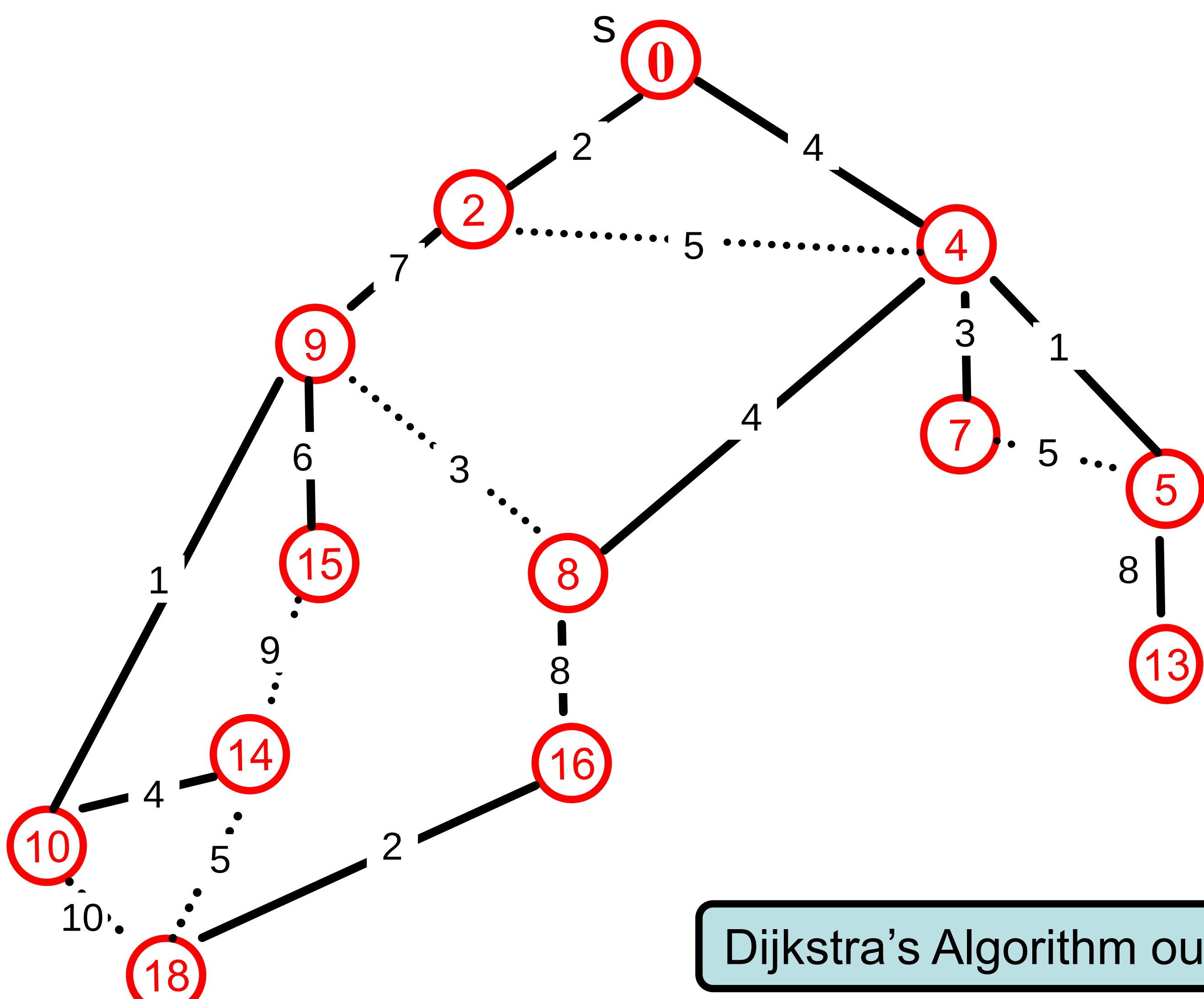
# Dijkstra’s Algorithm: Example



# Dijkstra's Algorithm: Example



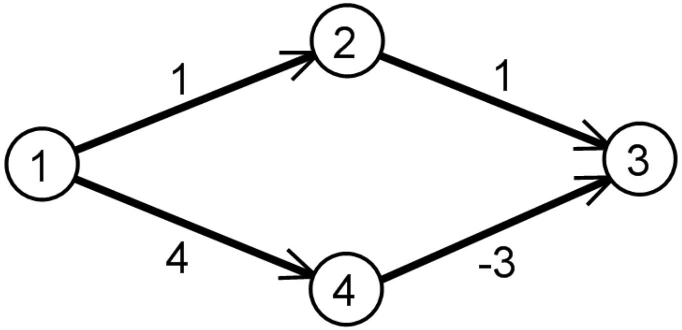
# Dijkstra's Algorithm: Example




Dijkstra's Algorithm outputs a tree.

# Remarks on Dijkstra's Algorithm

- Algorithm works on directed graph (with nonnegative weights)
- Algorithm produces a tree of shortest paths to $s$ following Parent links (for undirected graph)
- The algorithm fails with negative edge weights.
- Why does it fail?




- For unit length graph, Dijkstra's algorithm is same as BFS.

# Implementing Dijkstra's Algorithm

Priority Queue: Elements each with an associated key Operations

- Insert
- Find-min
  - Return the element with the smallest key
- Delete-min
  - Return the element with the smallest key and delete it from the data structure
- Decrease-key
  - Decrease the key value of some element

Implementations

Binary Heaps:

- $O(\log n)$ time insert/decrease-key/delete-min,
- $O(1)$ time find-min

Fibonacci heap:

- $O(1)$ time insert/decrease-key
- $O(\log n)$ delete-min
- O(1) time find-min

```
Dijkstra(G,c,s) {
   Initialize set of explored nodes S ← {s}

   // Maintain distance from s to each vertices in S
   d[s] ← 0
   Insert all neighbors v of s into a priority queue with value c_(s,v).

   while (S ≠ V)
   {
      // Pick an edge (u,v) such that u ∈ S and v ∉ S and
      //     d[u] + c_(u,v) is as small as possible.
      u ← delete min element from Q

      Add v to S and define d[v] = d[u] + c_(u,v).
      Parent(v) ← u.

      foreach (edge e = (v,w) incident to v)
         if (w ∉ S)
            if (w is not in the Q)
               Insert w into Q with value d[v] + c_(v,w)
            else (the key of w > d[v] + c_(v,w))
               Decrease key of v to d[v] + c_(v,w).
}
```

$O(n)$ of insert, each in $O(1)$

$O(n)$ of delete min, each in $O(\log n)$

$O(m)$ of decrease/insert key, each runs in $O(1)$

# Disjkstra's Algorithm: Correctness

Theorem: For any $u \in S$, the path $P_u$ on the tree is the shortest path from $s$ to $u$ on $G$. (For all $u \in S, d(u) = \text{dist}(s, u)$.)

Proof: Induction on $|S| = k$.

Base Case: This is always true when $S = \{s\}$.

Inductive Step: Say $v$ is the $(k+1)^{st}$ vertex that we add to S.

Let $(u, v)$ be last edge on $P_v$.

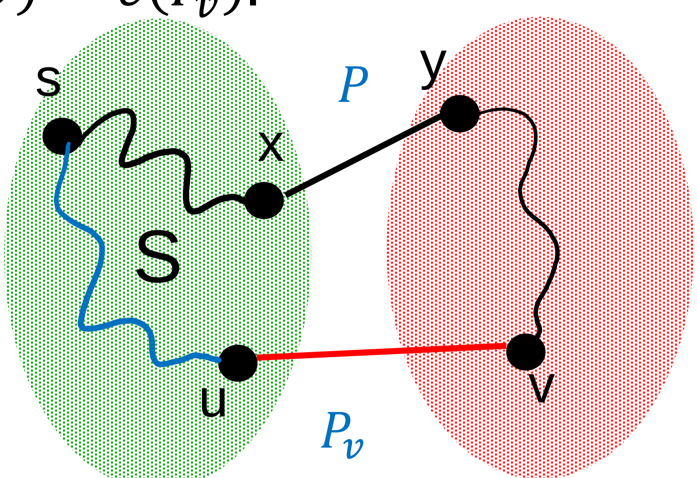
If $P_v$ is not the shortest path, there is a shorter path $P$ to $S$.

Consider the first time that $P$ leaves $S$ with edge $(x, y)$.

So, $c(P) \geq d(x) + c_{x,y} \geq d(u) + c_{u,v} = d(v) = c(P_v)$.

$P$ is the shorter path.

Due to the choice of $v$




A contradiction.

**Problem 4 (20 points).** Given a polynomial time algorithm to solve the following problem:

**Input:** An undirected graph $G = (V, E)$ and a positive integer edge length $l_e$ for each edge $e \in E$, and two vertices $s, t \in V$.

**Output:** A shortest path (in terms of the total edge length on the path) from $s$ to $t$ with the minimum number of edges.

Show how to use or modify Dijkstra's algorithm to solve the problem with the same time complexity. Prove the correctness and the runtime of the algorithm. (You can use any fact we proved about Dijkstra.)

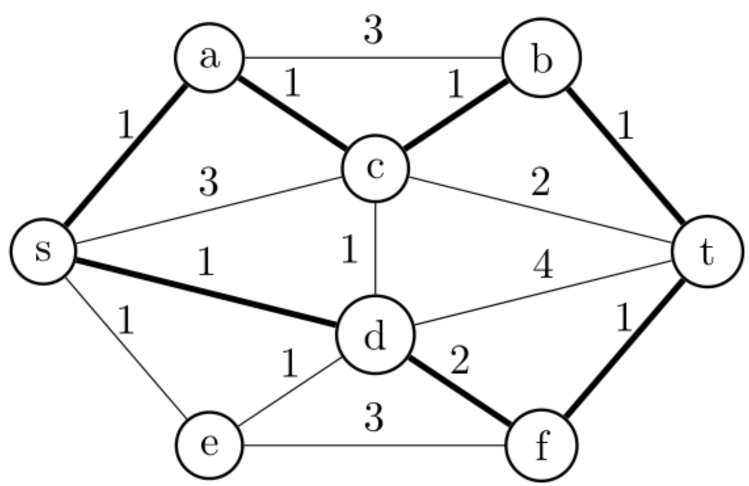
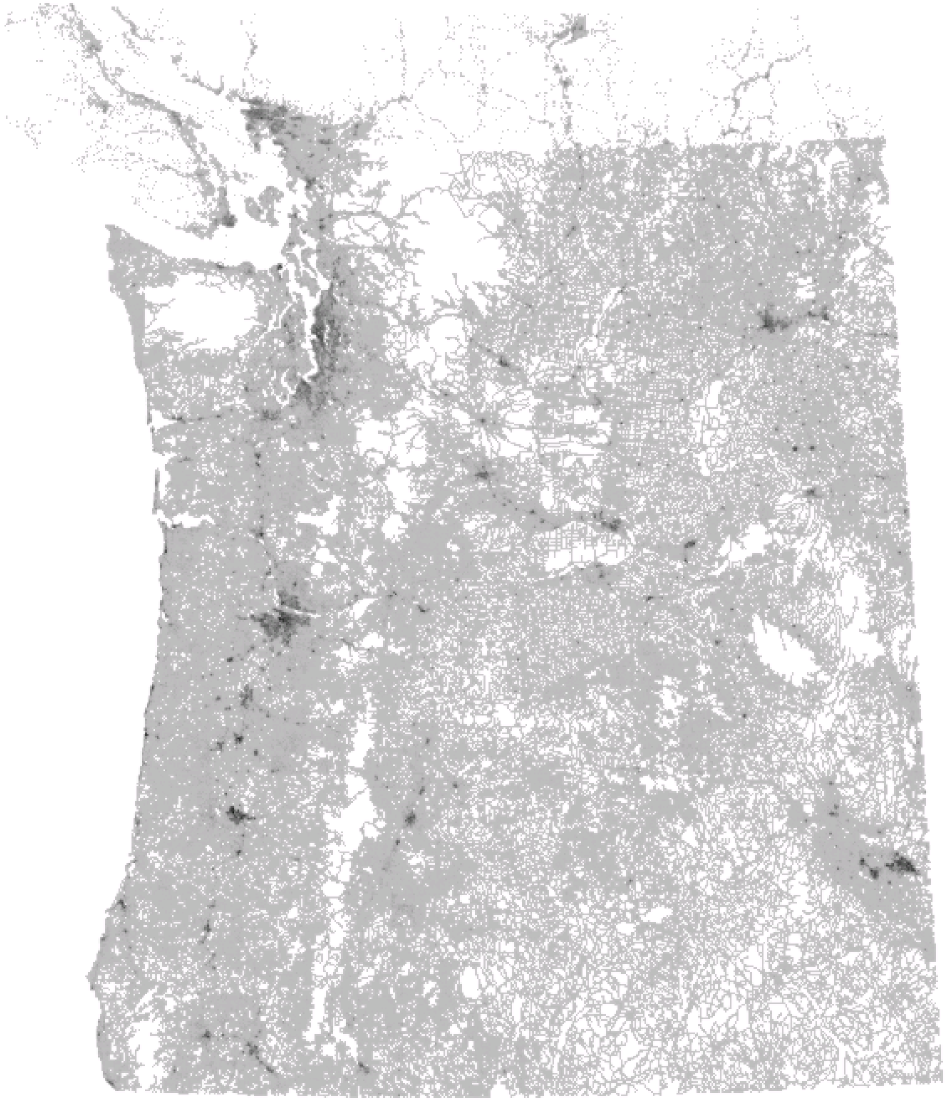



Figure 2: There are two paths (highlighted) of length 4 between $s$ and $t$. The path $s, d, f, t$ has only three edges and is the optimal solution in this example. Note that there are two $s - t$ path ($s, c, t$ and $s, d, t$) with only two edges, but they are of length 5 and are not shortest paths.

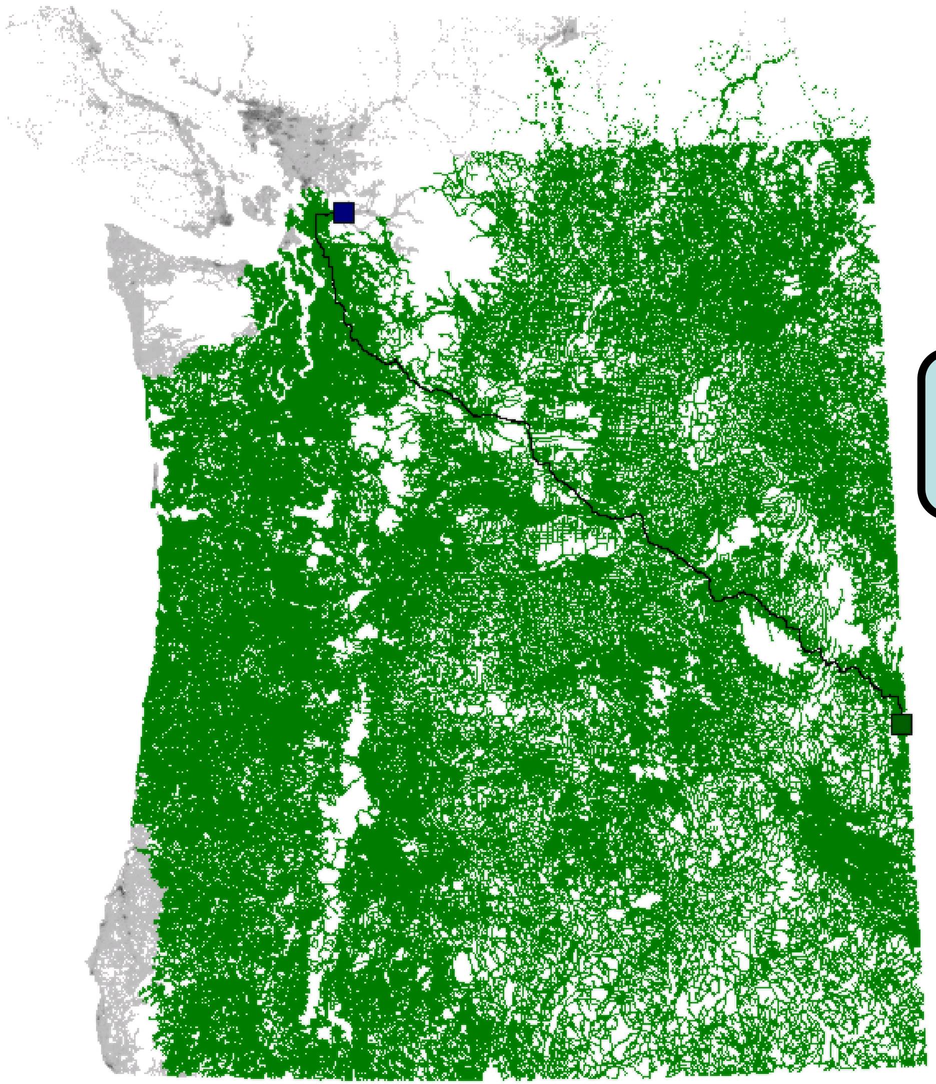
# Dijkstra Example



1.6 million vertices
3.8 million edges
Distance = travel time.

Images comes from A.V. Goldberg

# Dijkstra Example

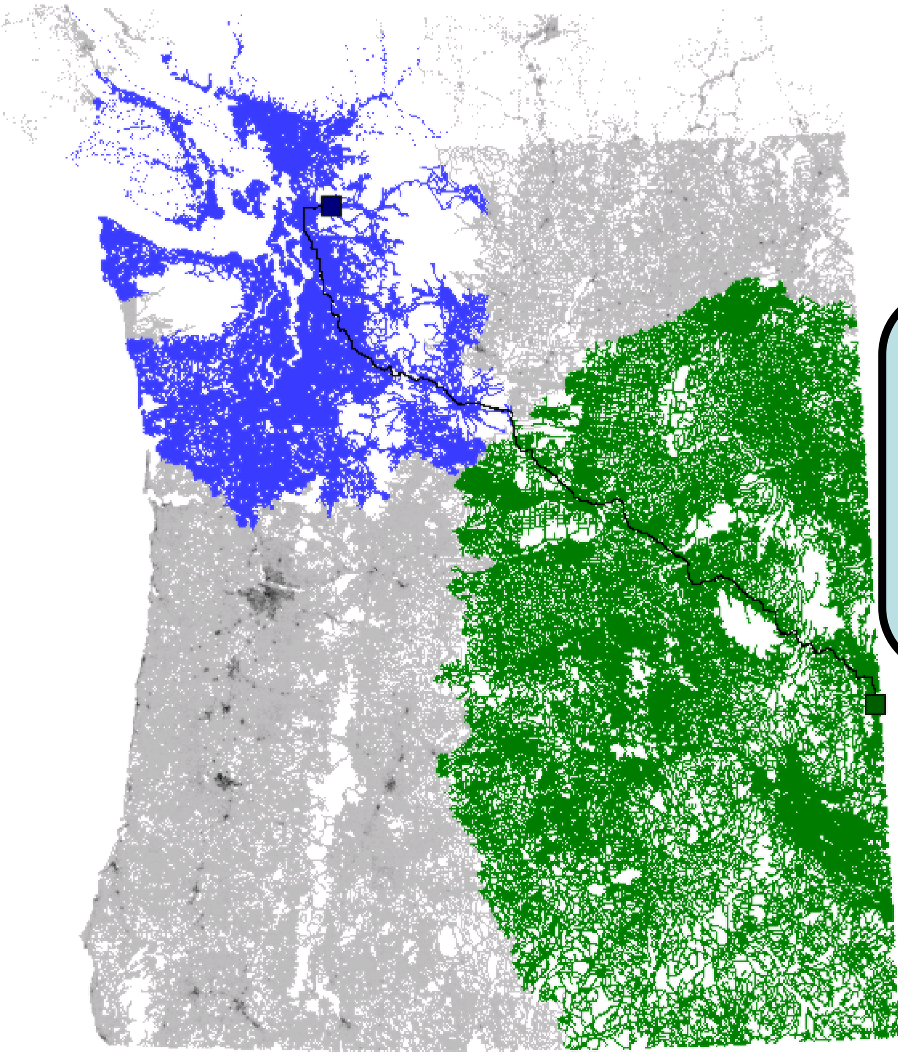


Searched Area
(starting from green point)

Problem of Dijkstra:
Didn't take account of where is $t$

340ms

# Bidirectional Dijkstra



Forward search
Backward search

Problem of Bidirectional Dijkstra:
Forward search did not take account of $t$
Backward search did not take account of $s$.

# $A^*$ Search

**AStar** $(G, c, s, t)$ **{**

  **Initialize set of explored nodes** $S \leftarrow \{s\}$

  **// Maintain distance from** $s$ **to each vertices in** $S$

  $d[s] \leftarrow 0$

  **while** $(S \neq V)$

  **{**

    **Pick an edge** $(u, v)$ **such that** $u \in S$ **and** $v \notin S$ **and**

      $d[u] + c_{(u,v)} + h(v)$ **is as small as possible.**

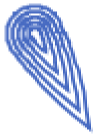
    **Add** $v$ **to** $S$ **and define** $d[v] = d[u] + c_{(u,v)}$**.**

    $Parent(v) \leftarrow u$**.**

  **}**

BFS

Dijkstra



$A^*$

- $h(v)$ is the estimate of distance from $v$ to $t$
- If $h(v)$ is exactly the shortest distance from $v$ to $t$, then the algorithm would go directly to $t$.

# $A^*$ Search

Let $h(v)$ be the estimate distance from $v$ to $t$.

Define the reduced cost $\tilde{c}_{u,v} = c_{u,v} - h(u) + h(v)$.

**Claim 1:** Shortest path on $\tilde{c}$ is same as shortest path on $c$.

**Claim 2:** If the reduced cost $\tilde{c}_{u,v}$ is non-negative,

Dijkstra on $\tilde{c}$ is equivalent to $A^*$ on $c$ with the estimate $h$.

Therefore, $A^*$ is correct.

# Estimating the distance

**Euclidean bounds:**

Limited applicability, not very good for driving directions.

**Triangle inequality:**

Let $dist(x, y)$ be the shortest path distance from $x$ to $y$.

For any node $l$, we can estimate the distance $dist(x, t)$ by

$$dist(x, l) - dist(t, l).$$

Note that $dist(x, t) + dist(t, l) \geq dist(x, l)$. (Triangle inequality)

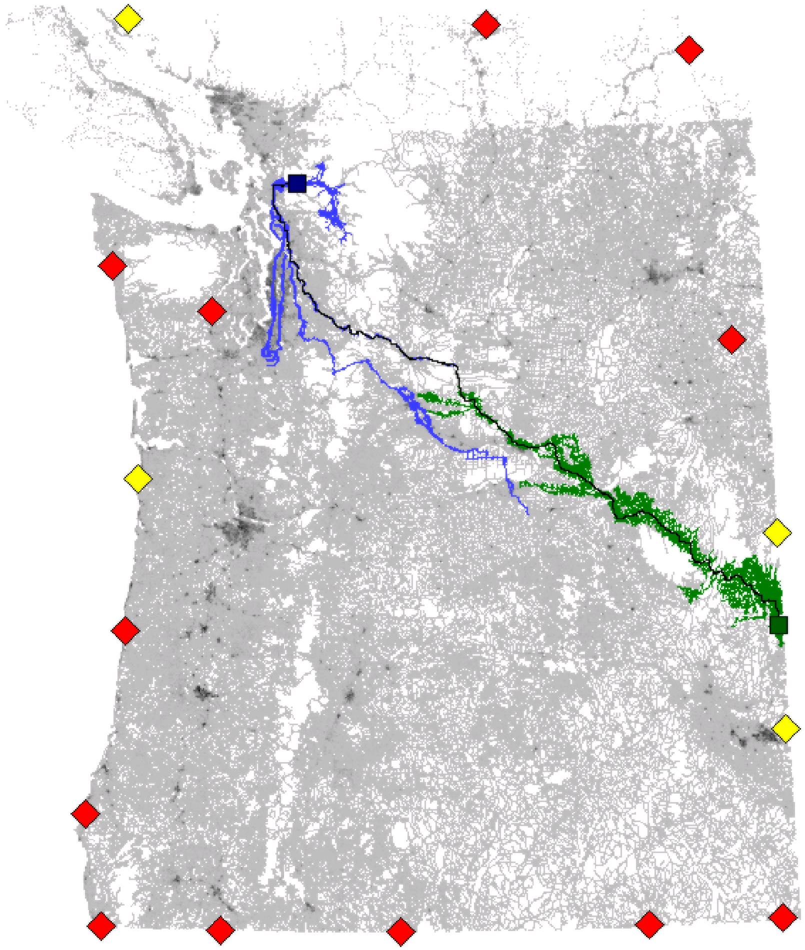
So, $dist(x, l) - dist(t, l)$ is a lower bound for $dist(x, t)$!

**Algorithm:** Select landmarks $l_i$, define

$$h(v) = \max_i dist(x, l_i) - dist(t, l_i).$$

12ms

# $A^*$ + Landmarks + Triangle equality (ATL)



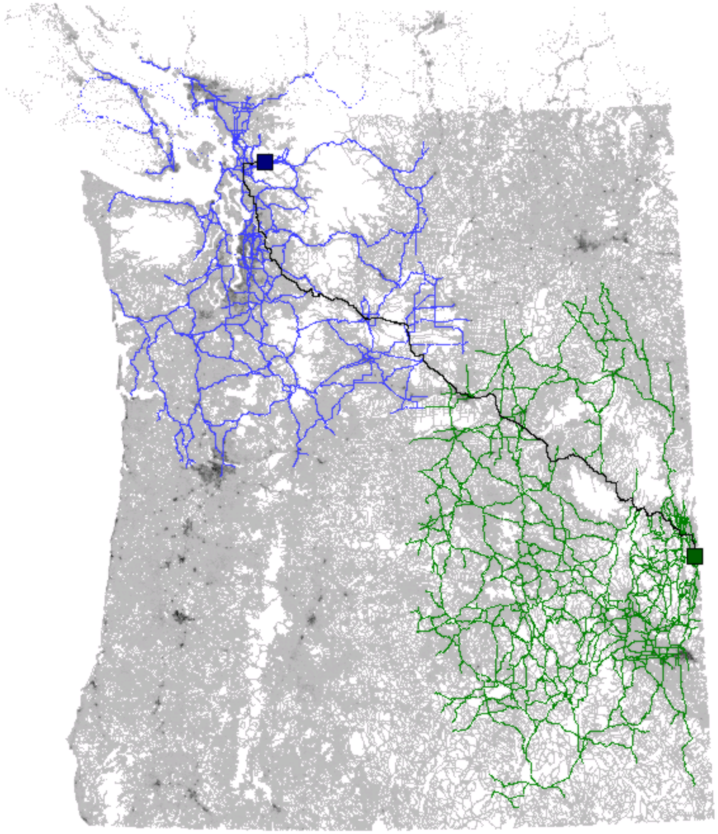
Forward search
Backward search
Inactive landmarks
Active landmarks

Problem of ATL:
We should stick with highway!

From now on, we allow to preprocess the graph.

30ms

# Reach Algorithm

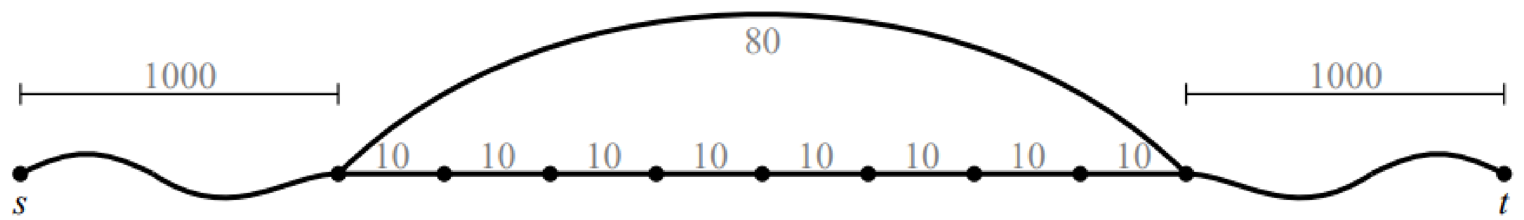


Use highway except for the beginning and the end of the journey!

Forward search
Backward search

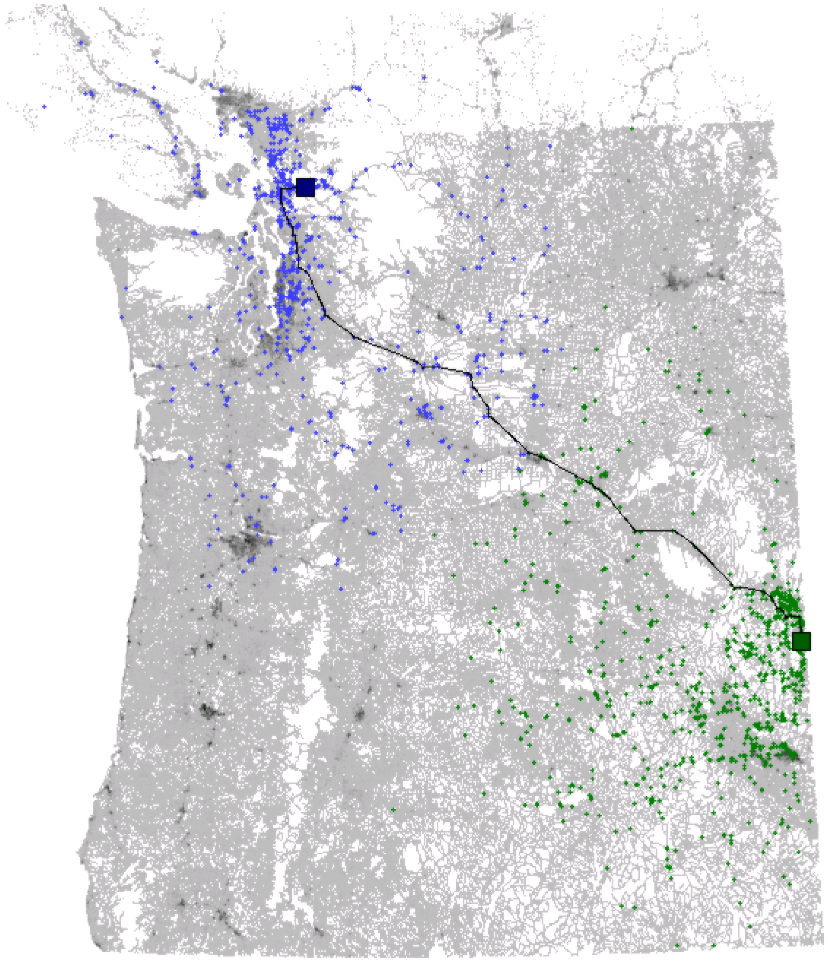
# Creating shortcut in the graph

When you are on the highway, don't need to keep checking the map until you are nearby!



2ms

# Reach + Shortcut Algorithm



Forward search
Backward search

0.7ms

# Reach + Shortcut + ATL Algorithm