CSE 421

Greedy Algorithms / Dijkstra's Algorithm

Yin Tat Lee

Single Source Shortest Path

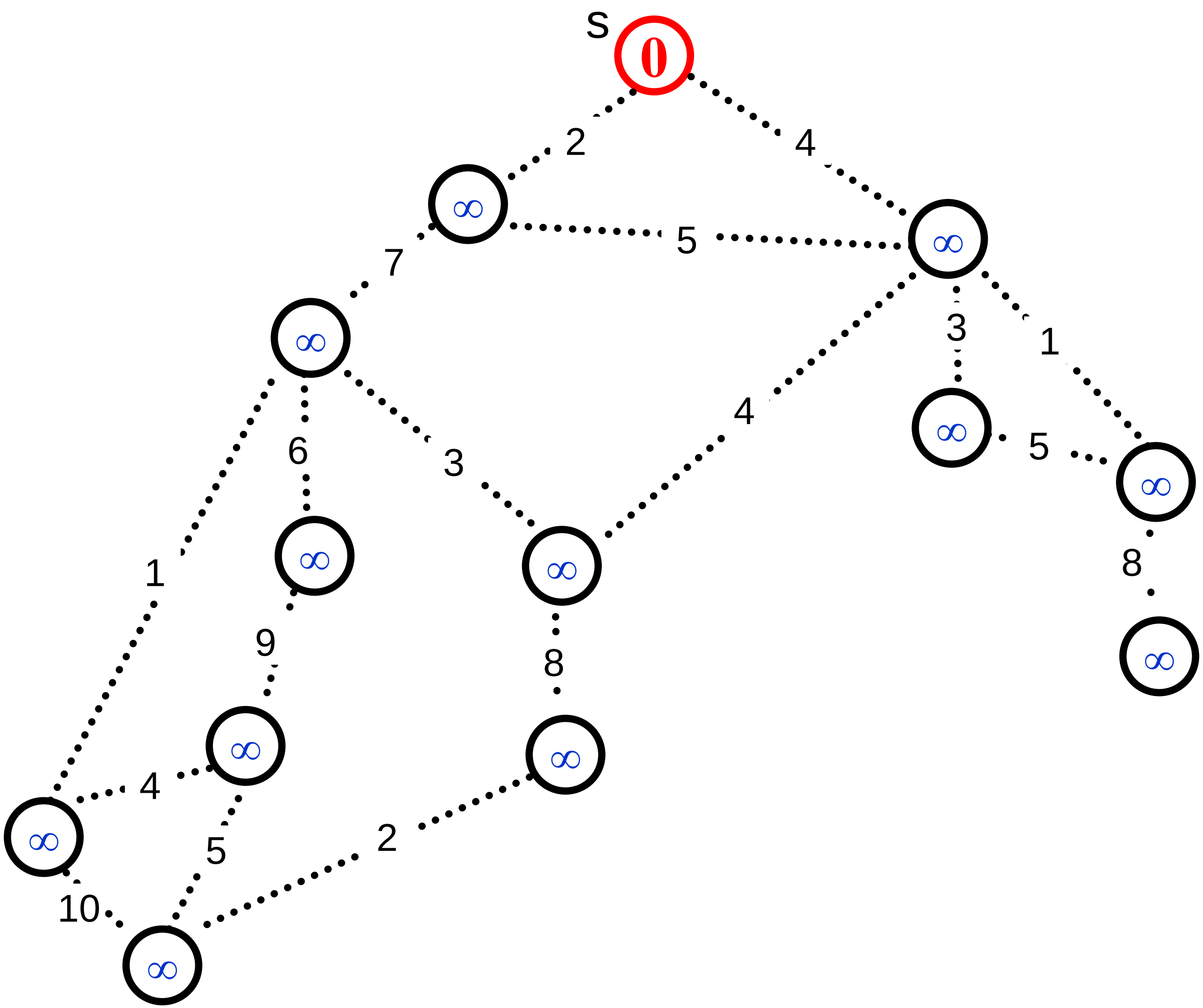
Given an (un)directed graph $G = (V, E)$ with non-negative edge weights $c_e \geq 0$ and a start vertex s.

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

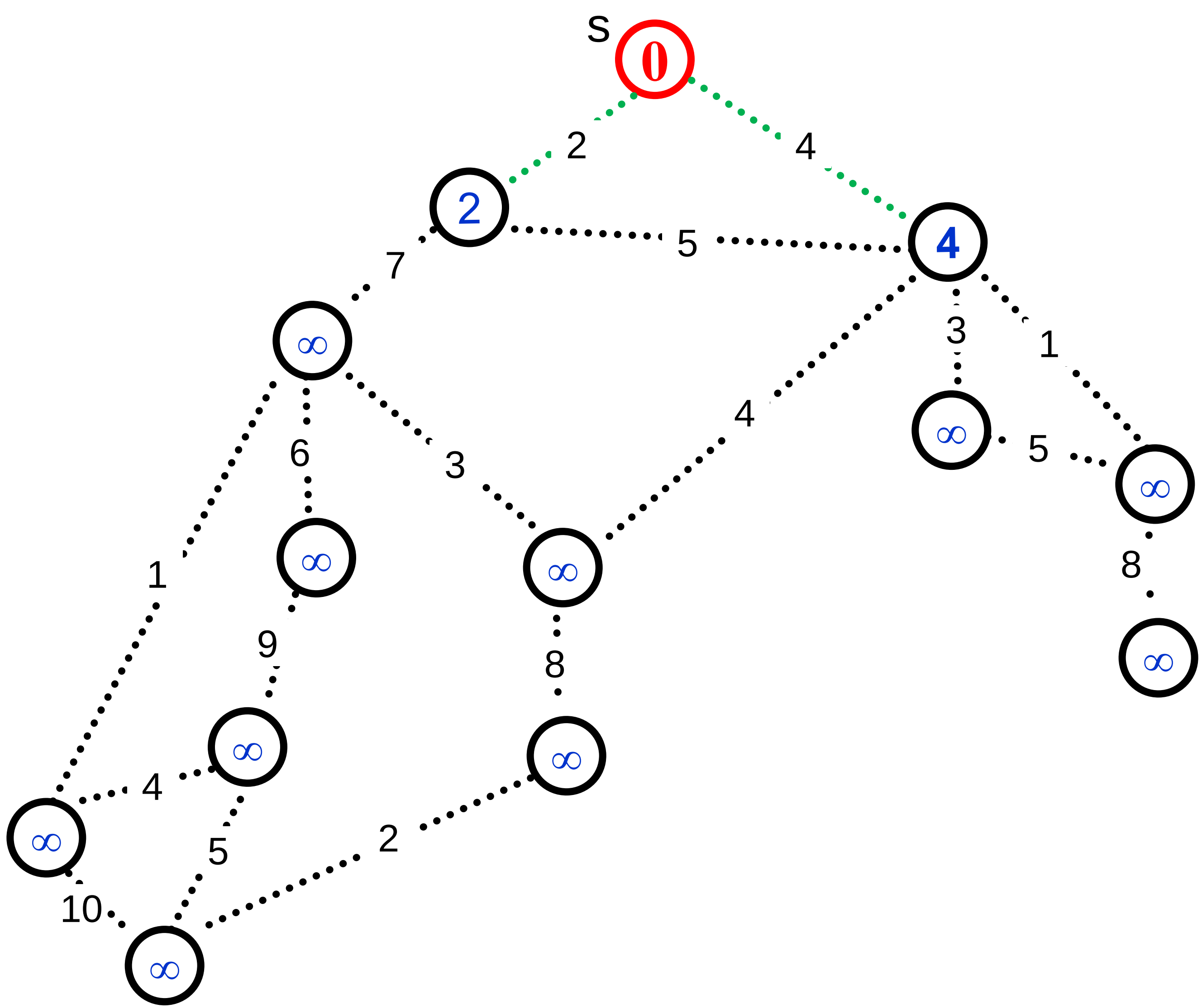
```
Dijkstra(G,c,s) {
   Initialize set of explored nodes S ← {s}

   // 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.

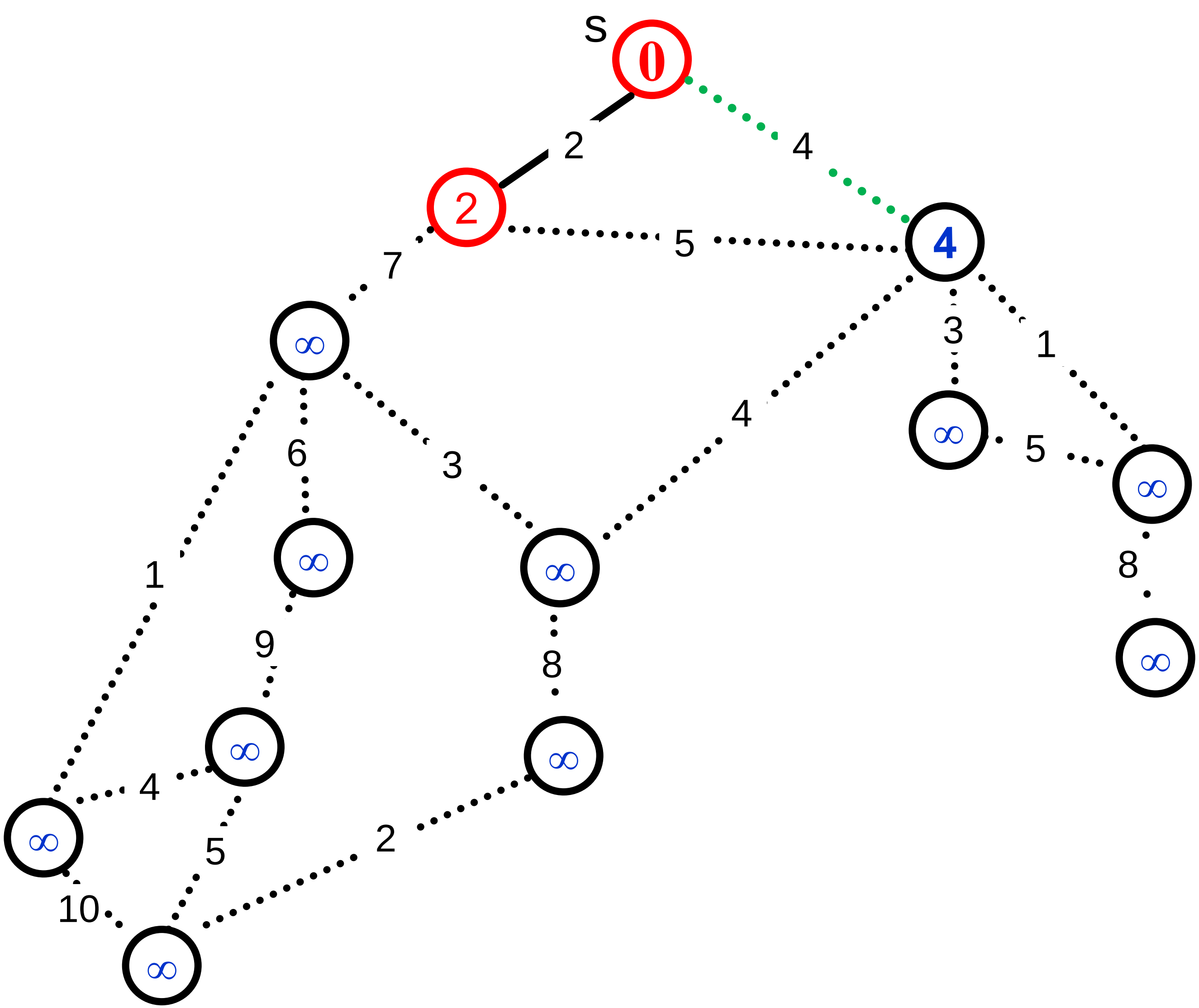
Dijkstra’s Algorithm: Example



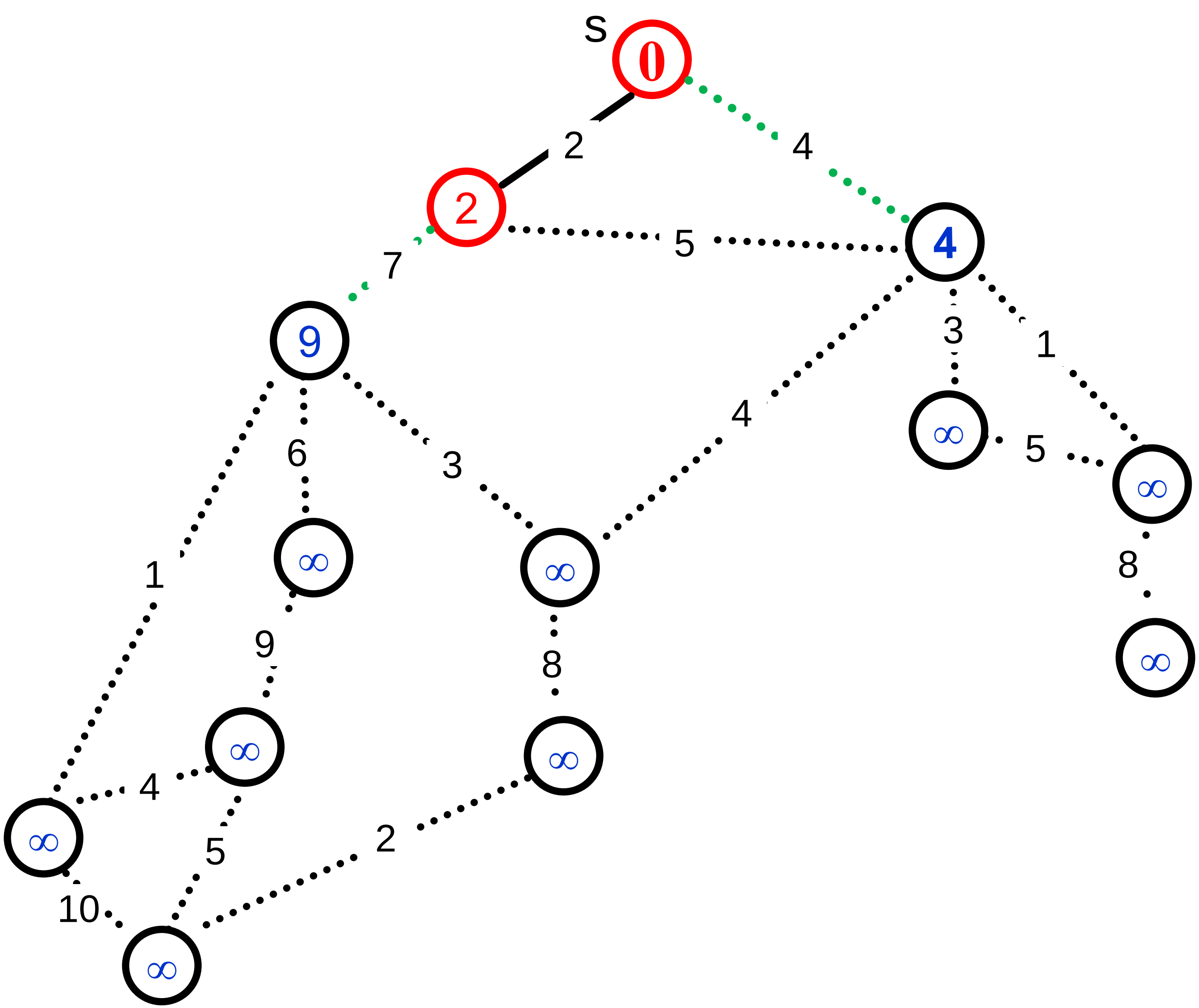
Dijkstra's Algorithm: Example



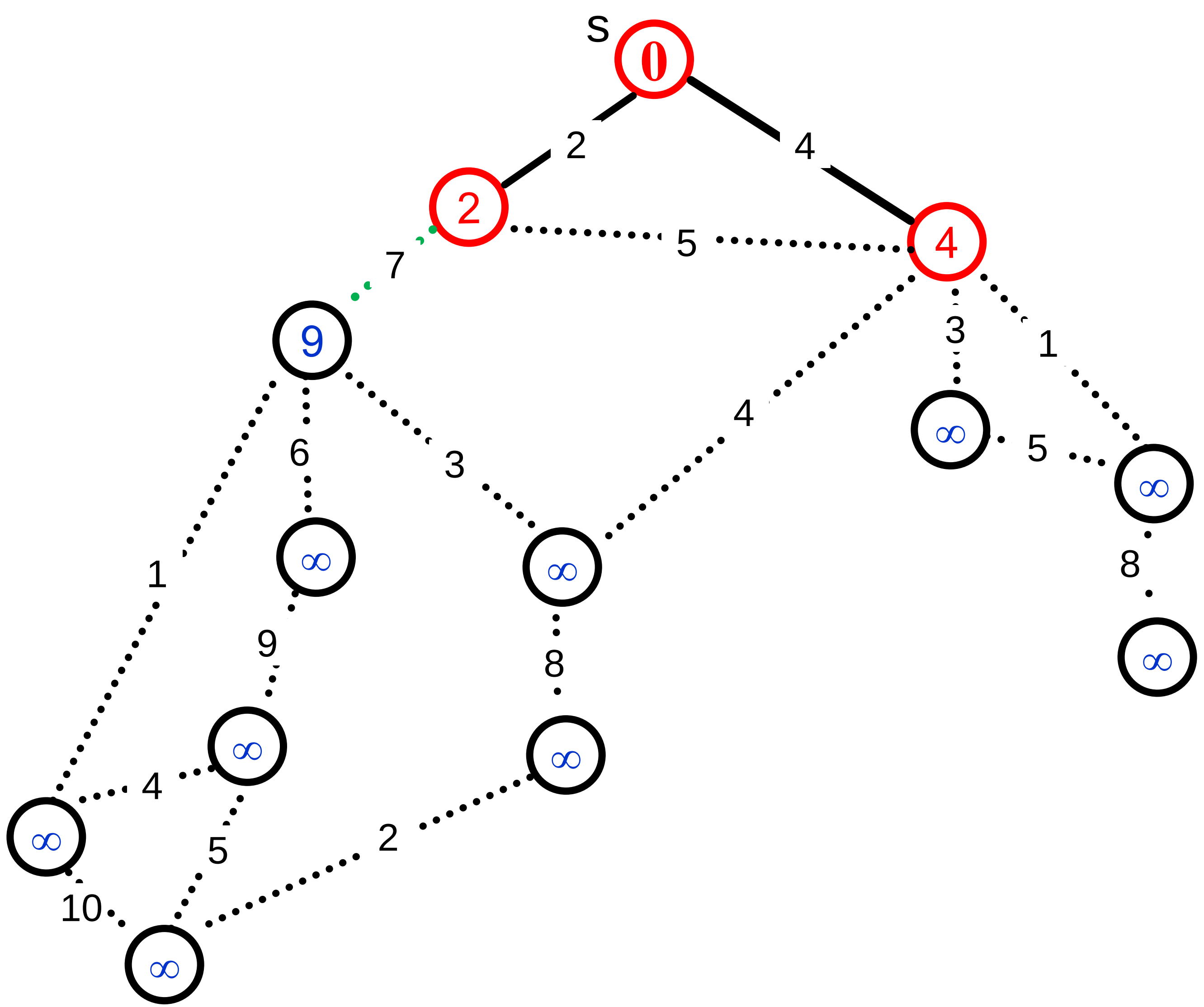
Dijkstra’s Algorithm: Example



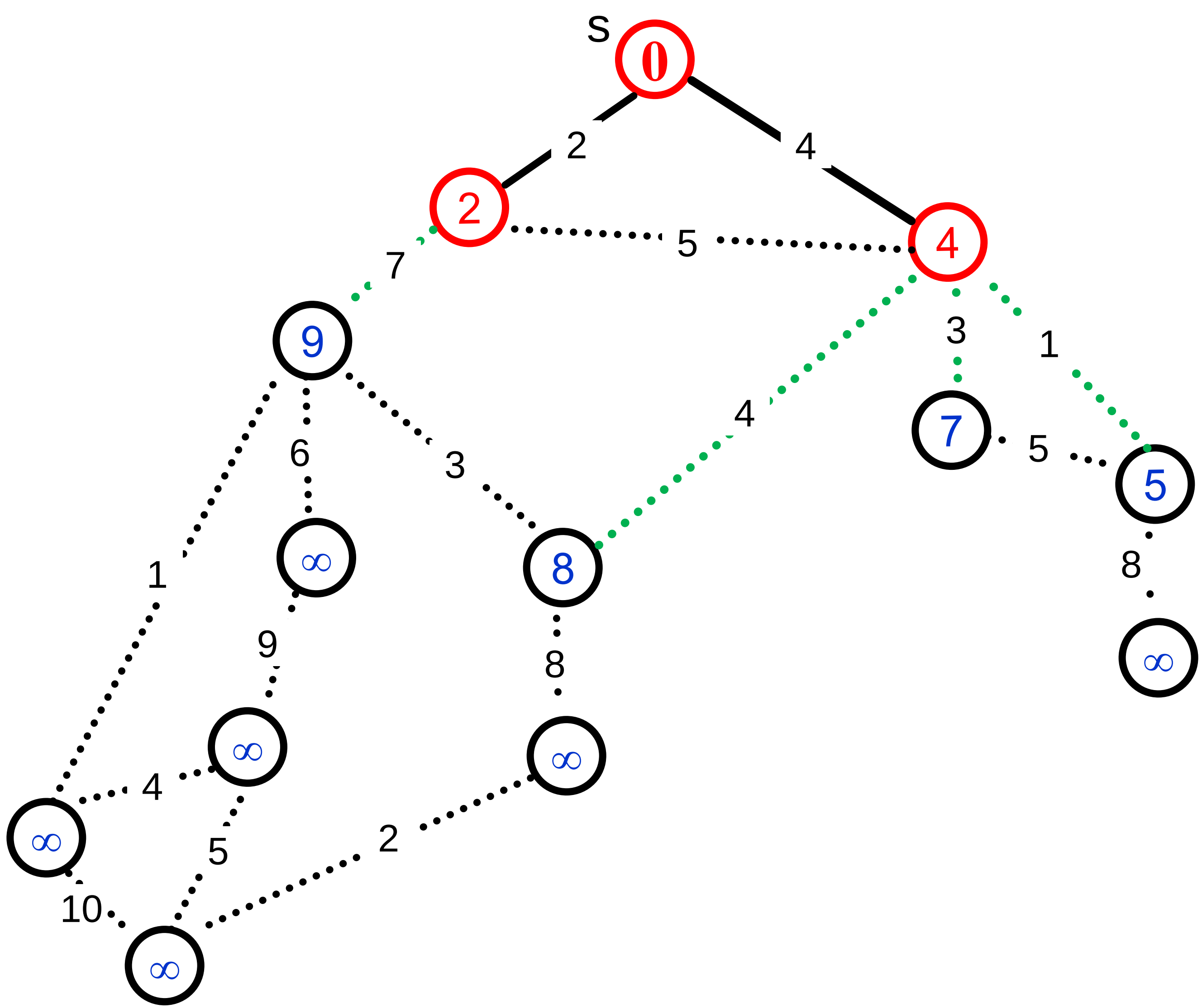
Dijkstra's Algorithm: Example



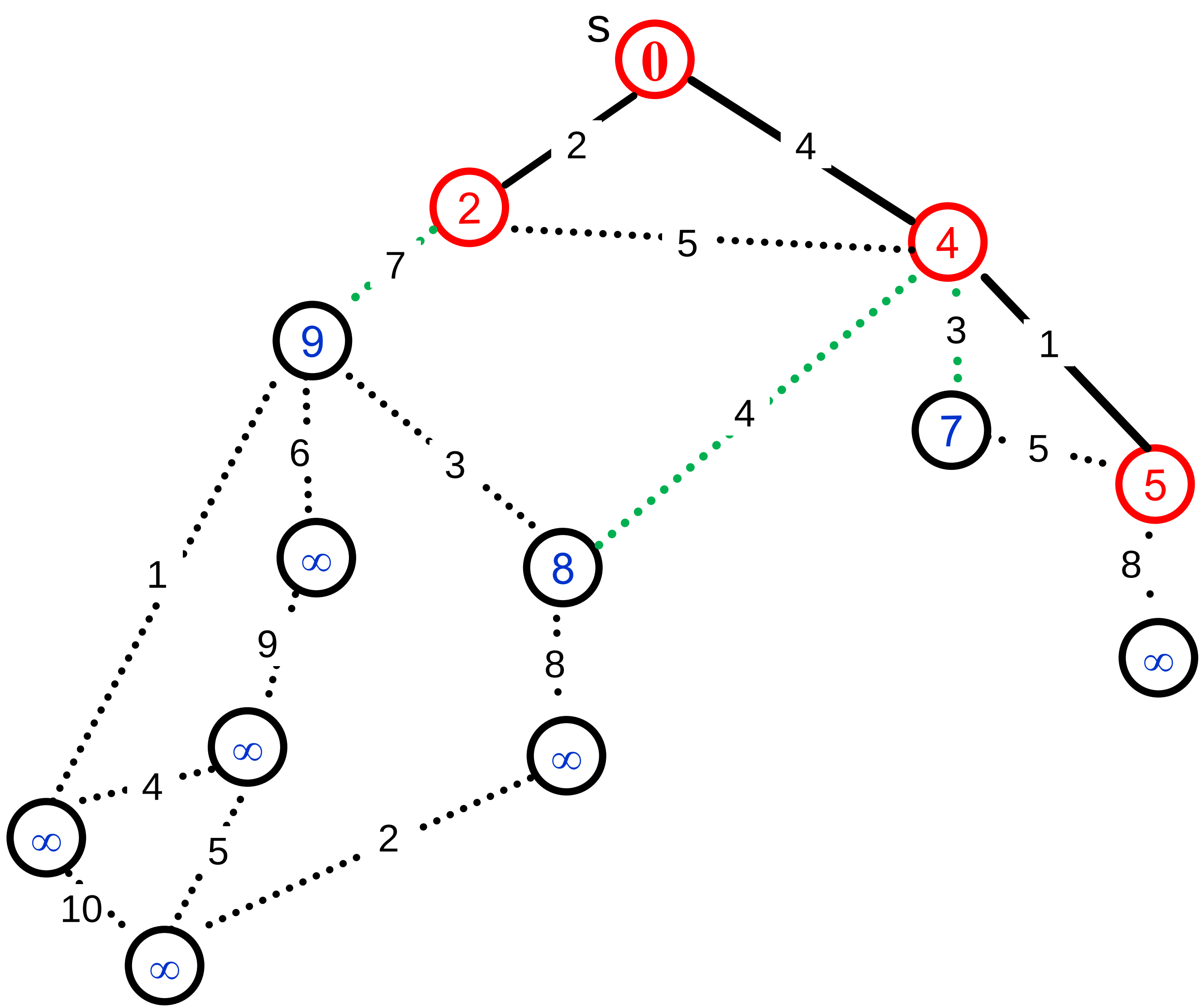
Dijkstra's Algorithm: Example



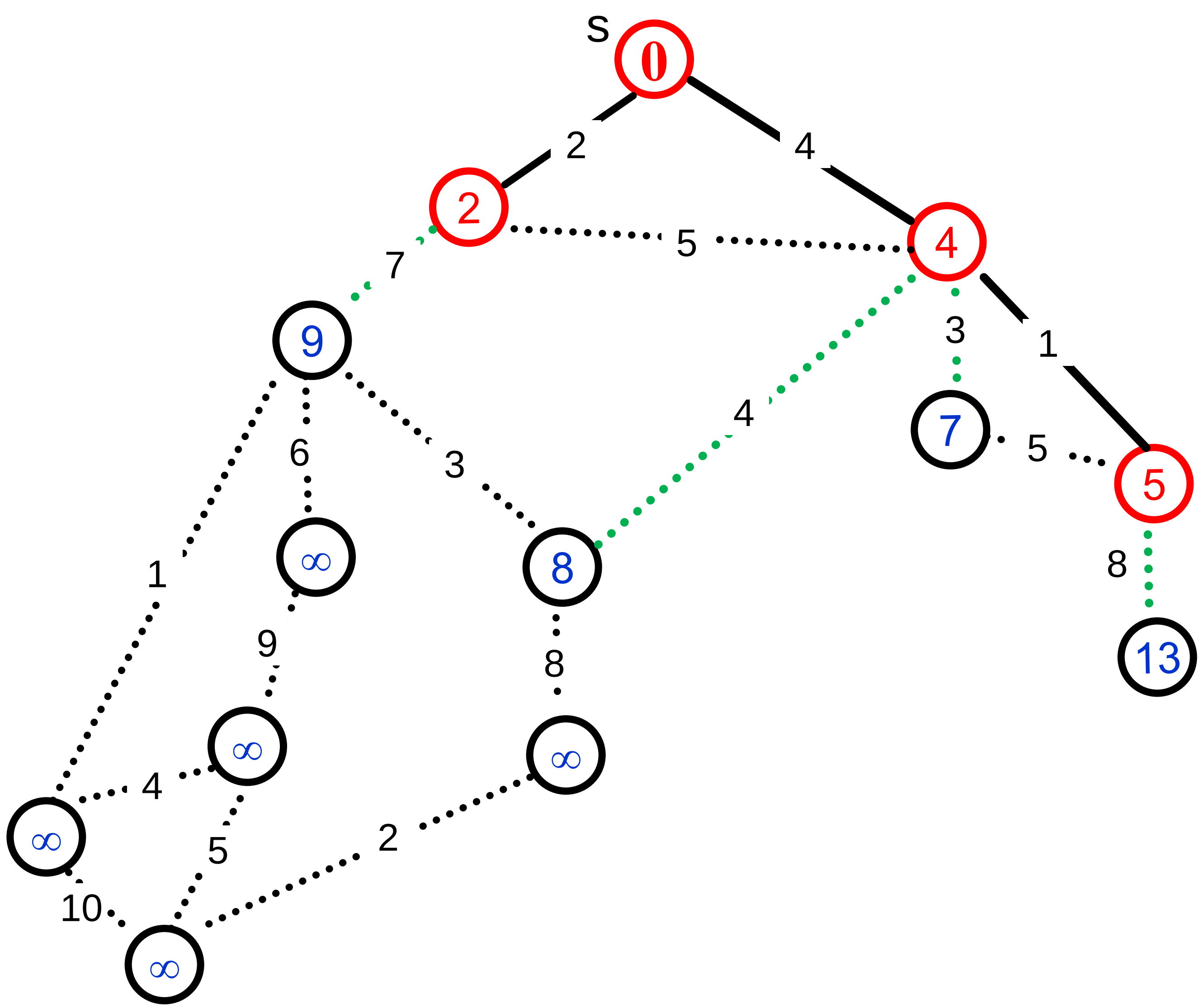
Dijkstra's Algorithm: Example



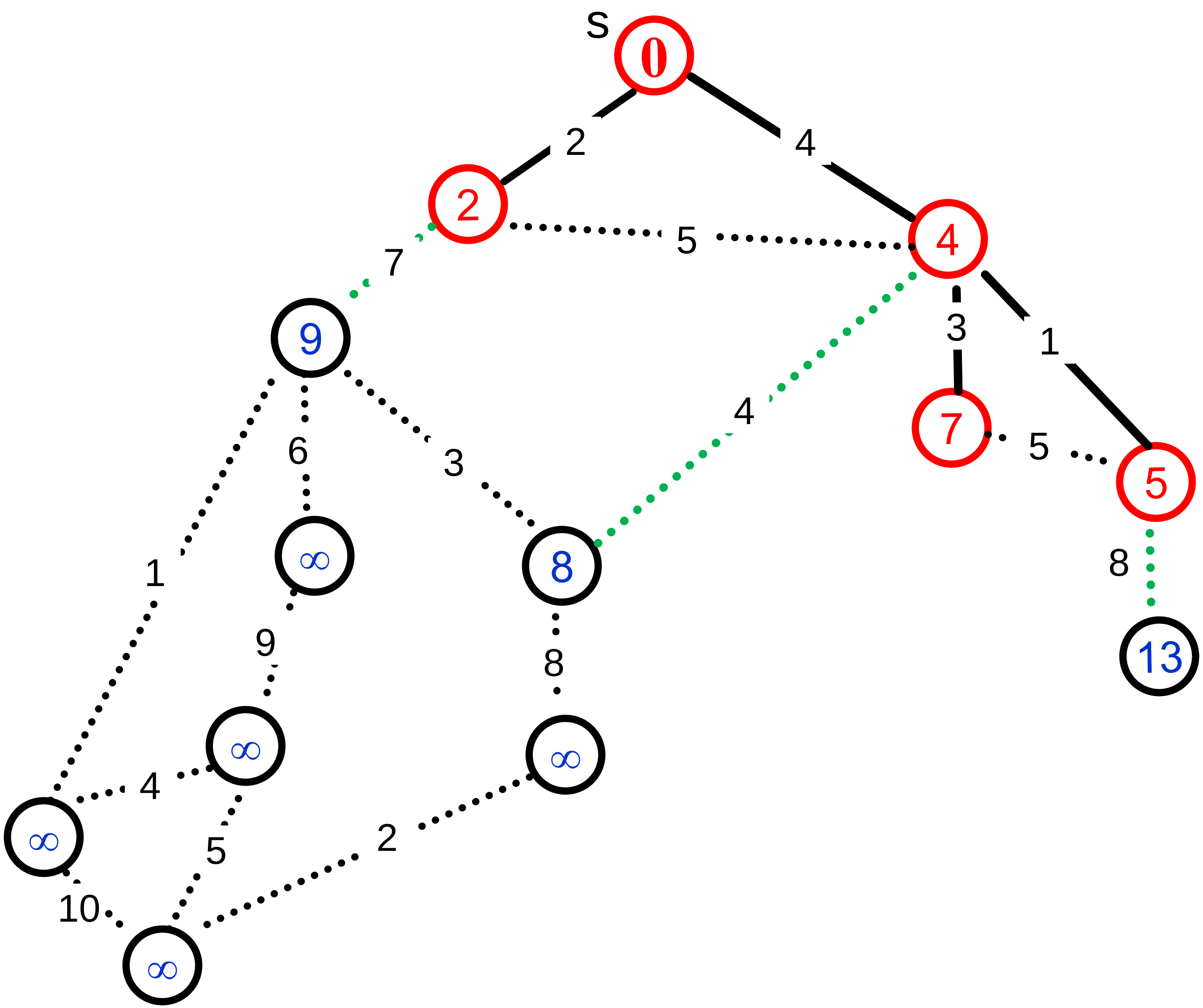
Dijkstra's Algorithm: Example



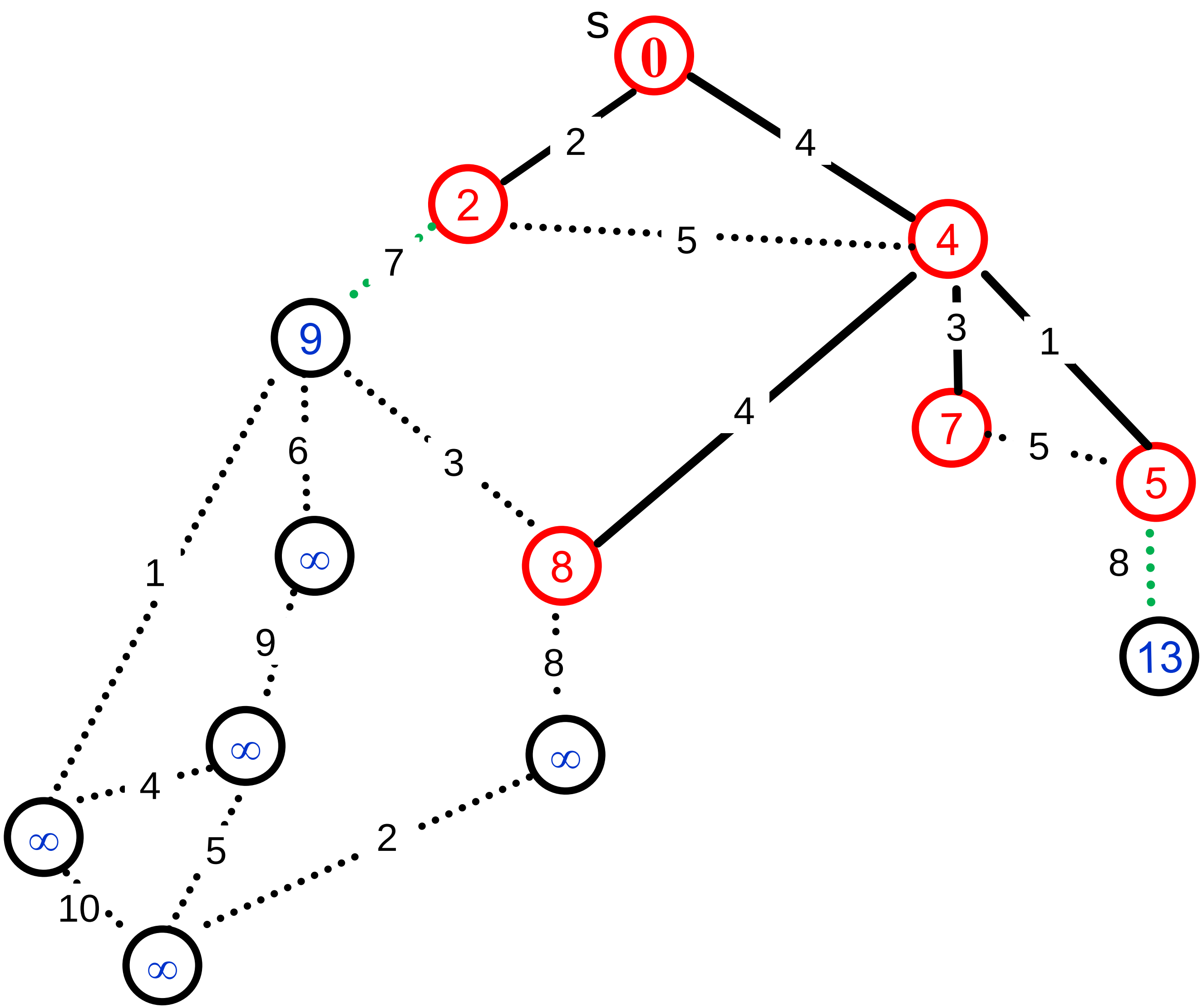
Dijkstra’s Algorithm: Example



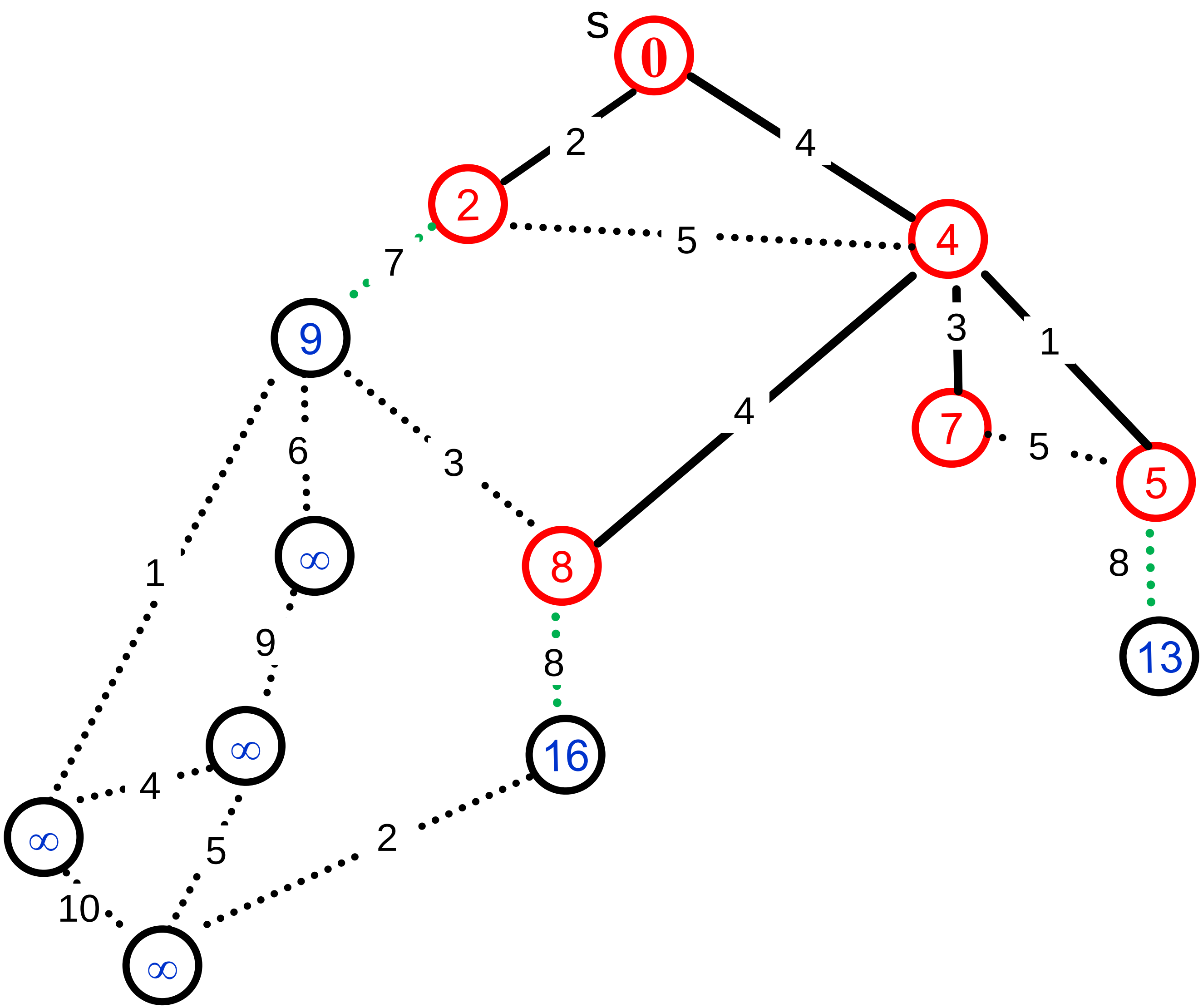
Dijkstra's Algorithm: Example



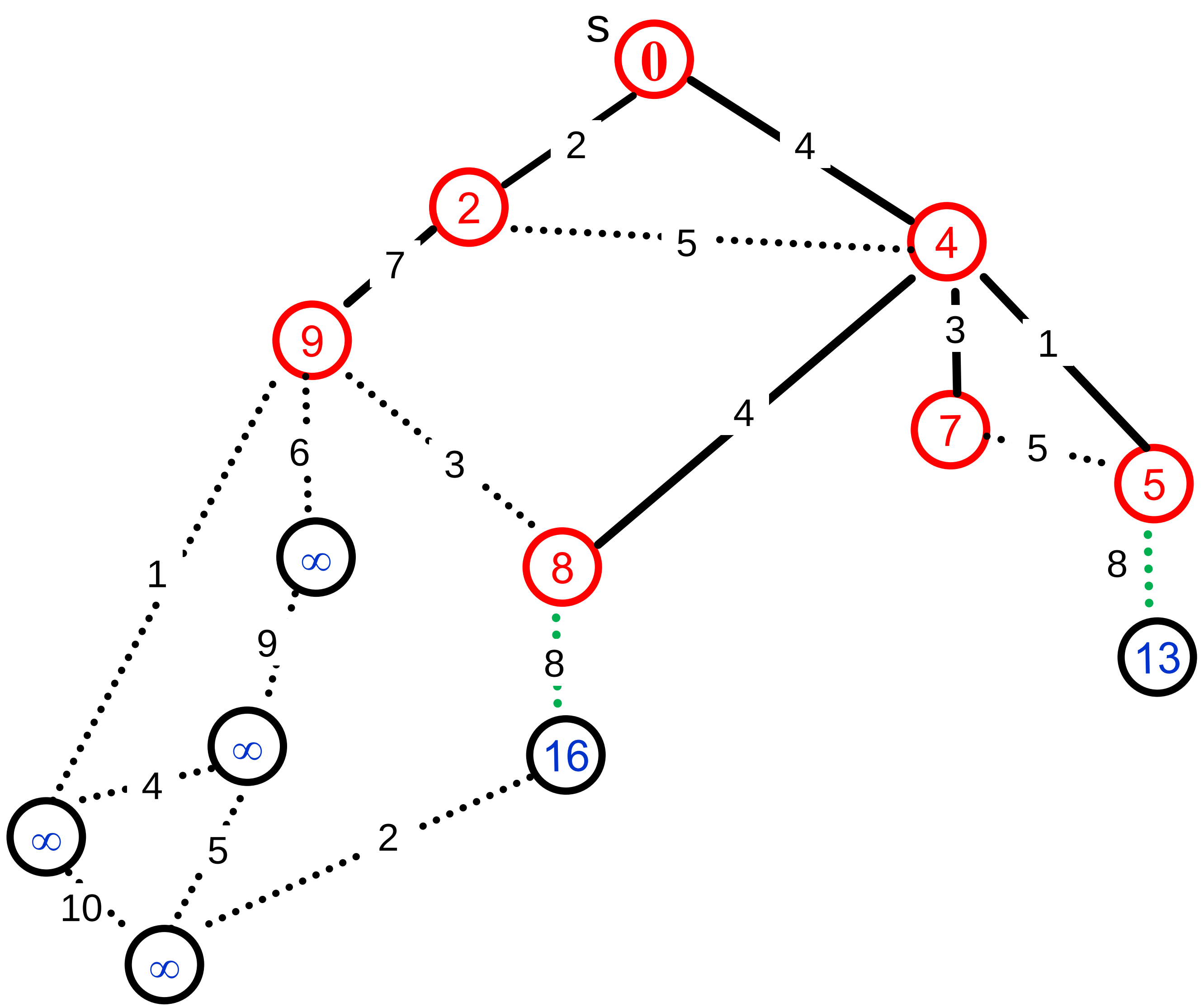
Dijkstra’s Algorithm: Example



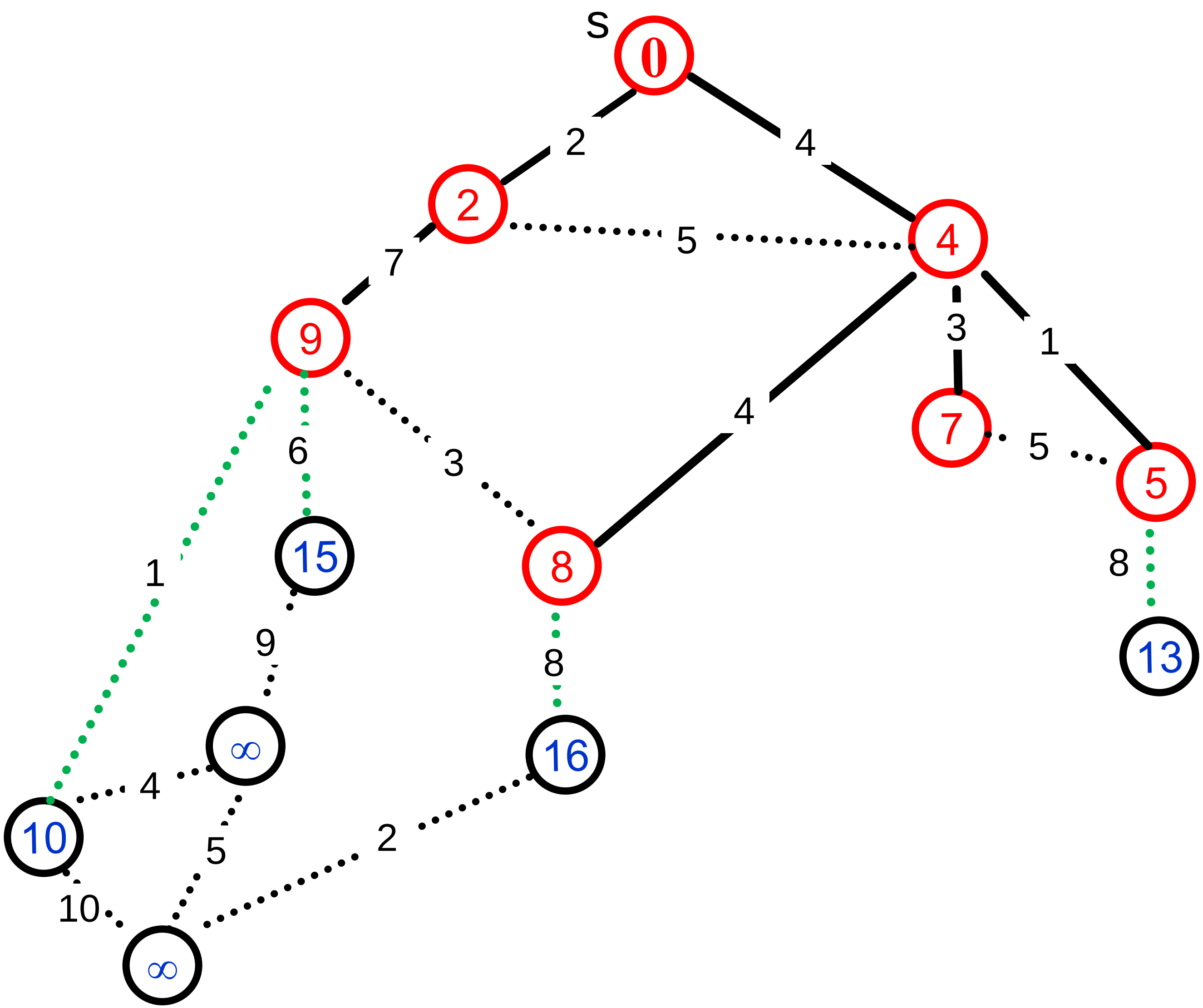
Dijkstra’s Algorithm: Example



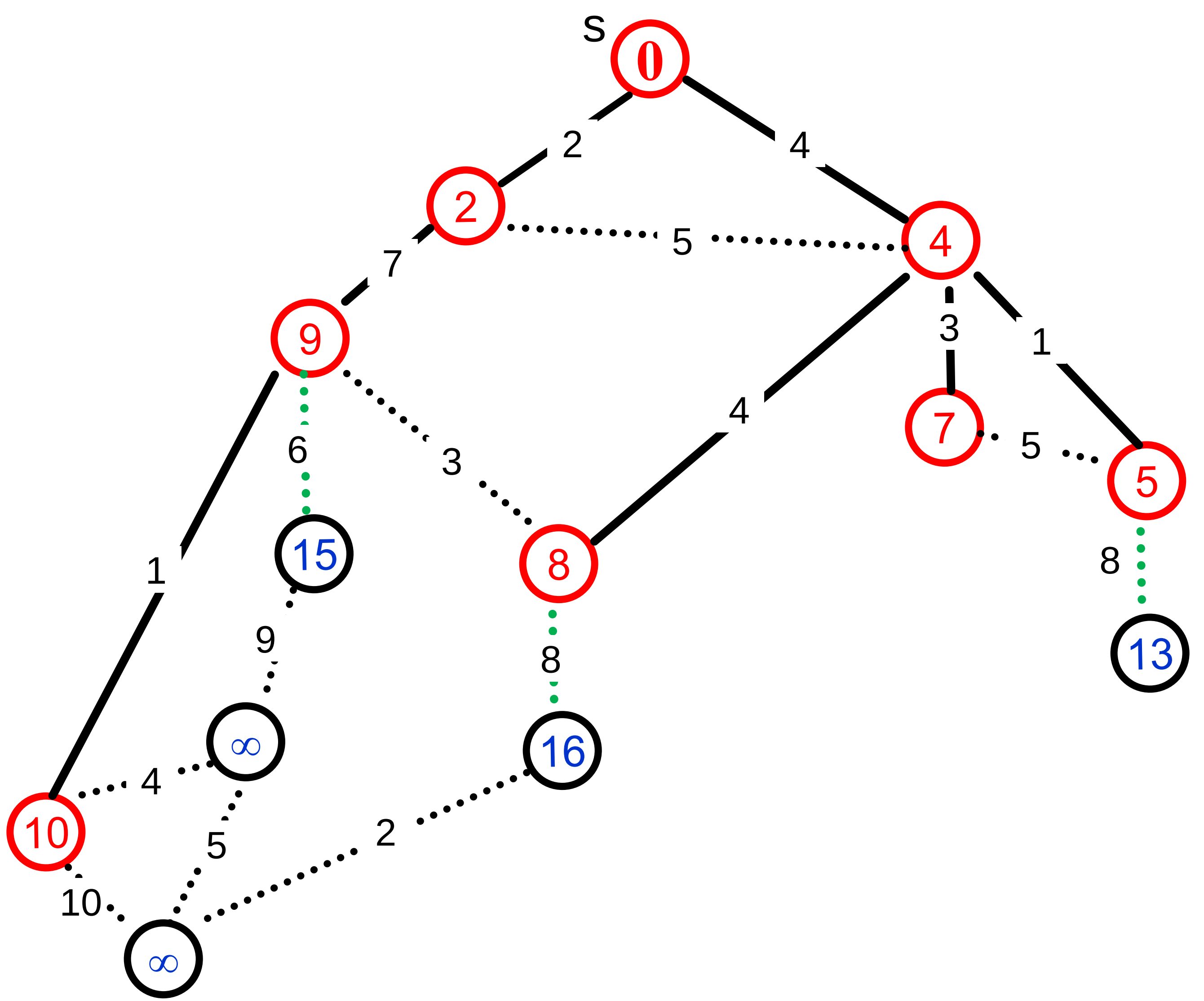
Dijkstra's Algorithm: Example



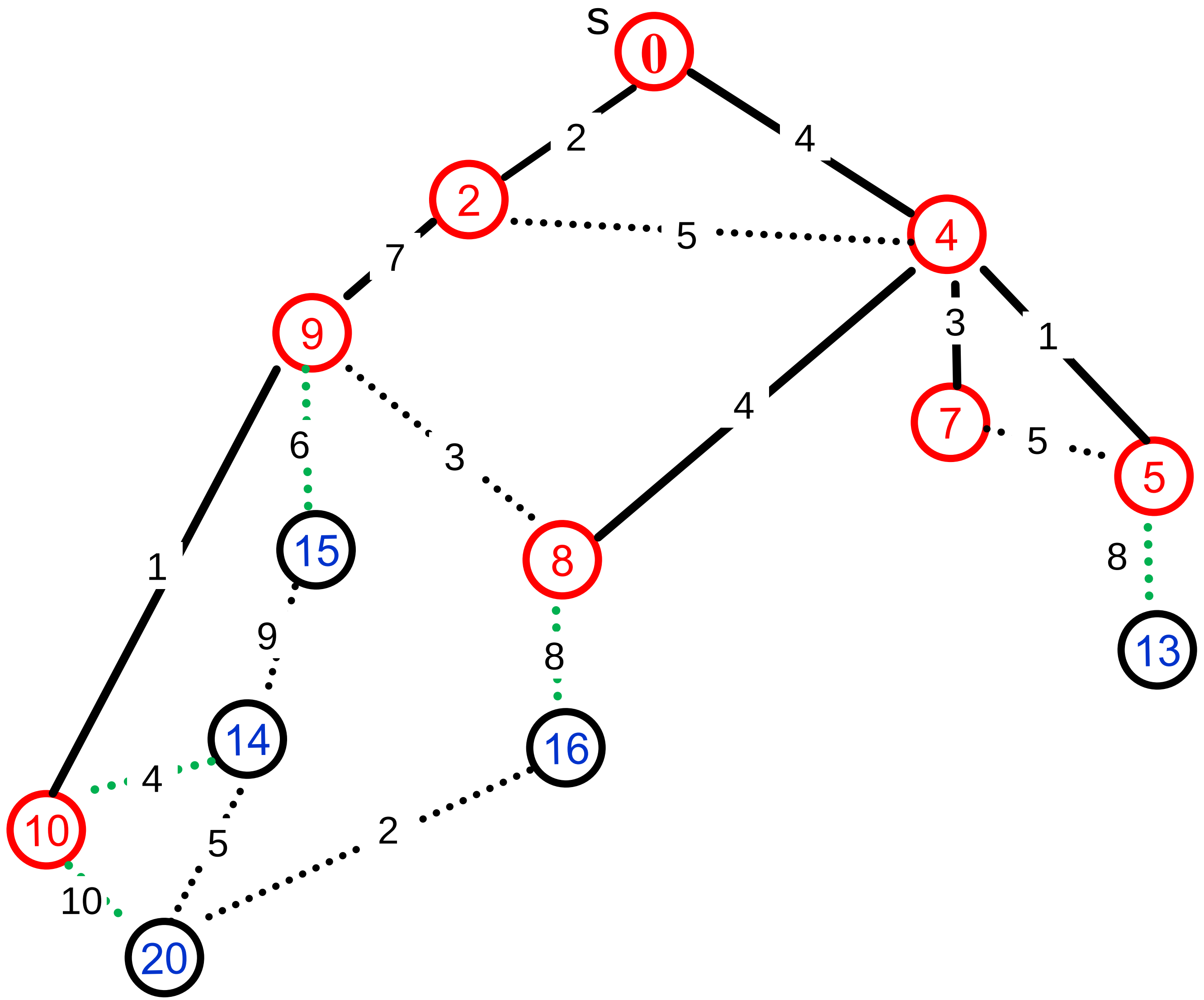
Dijkstra's Algorithm: Example



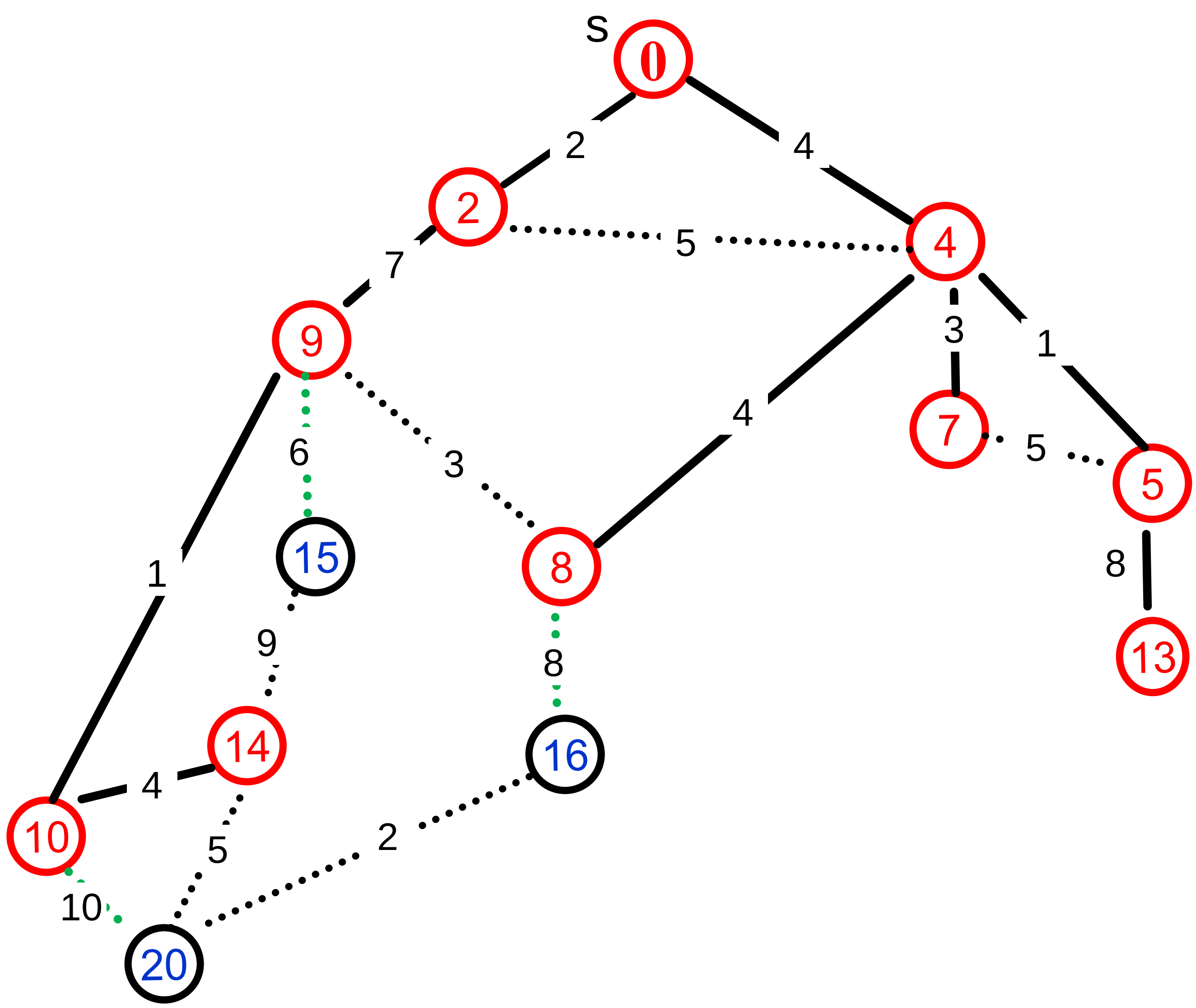
Dijkstra's Algorithm: Example



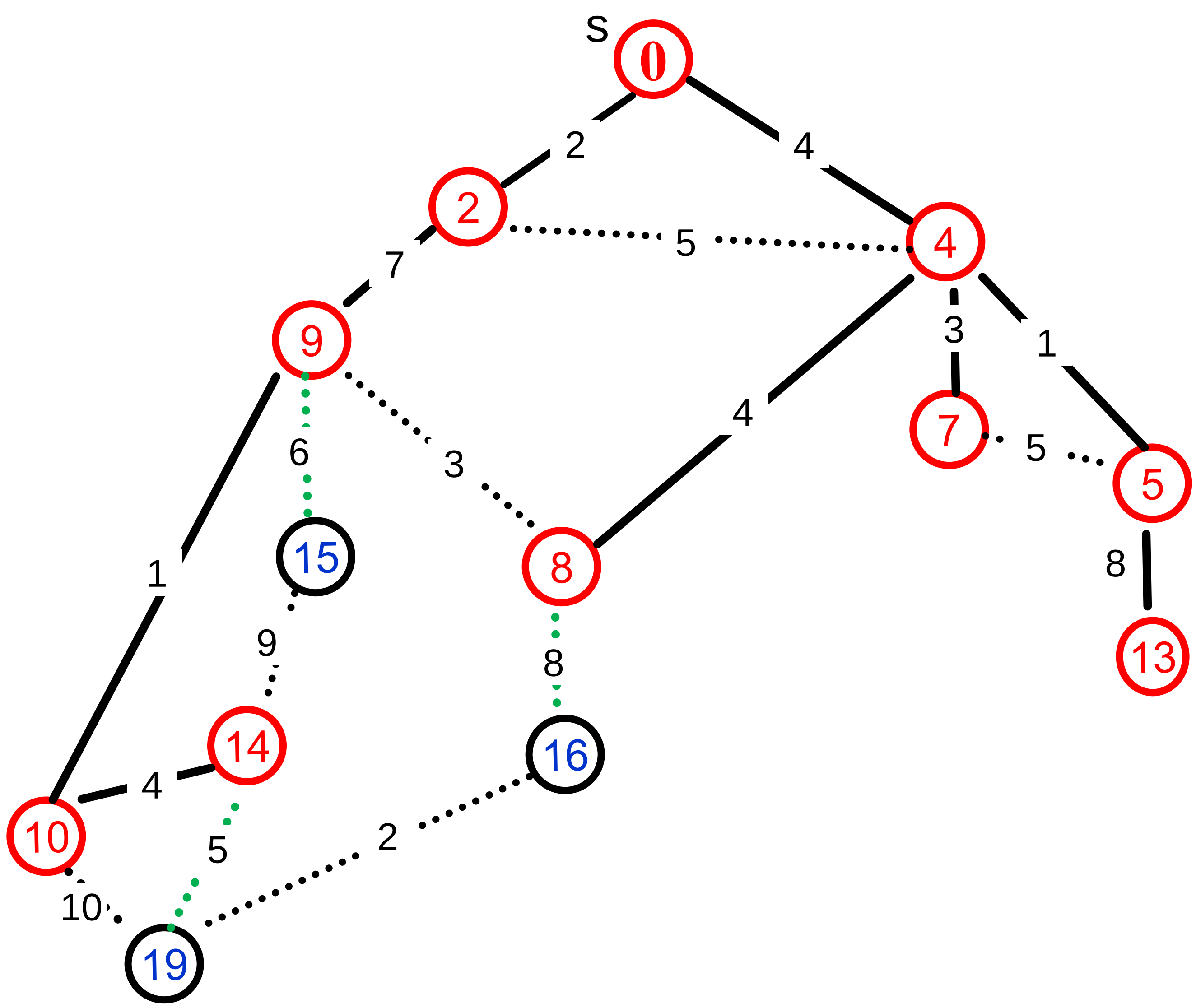
Dijkstra's Algorithm: Example



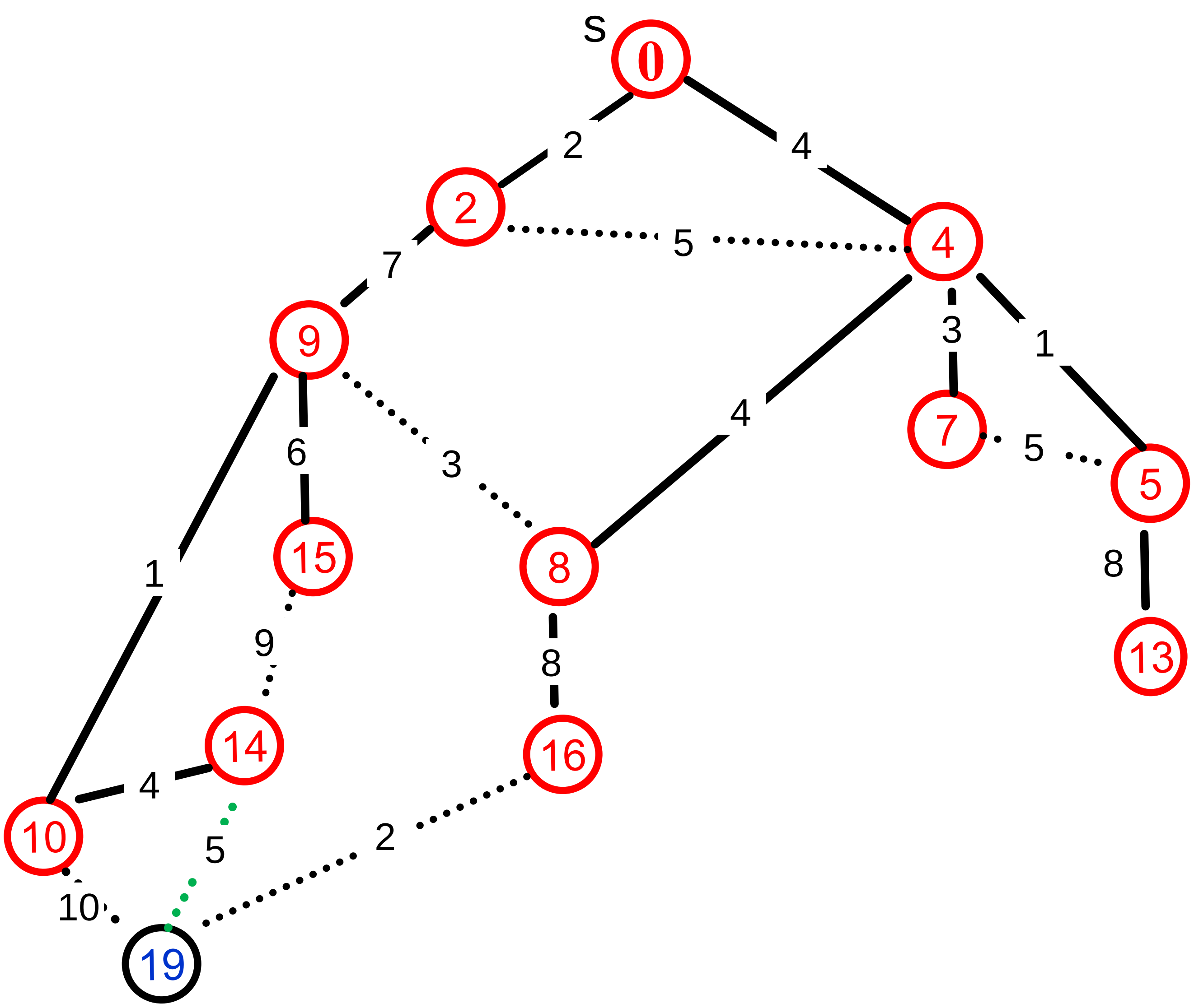
Dijkstra's Algorithm: Example



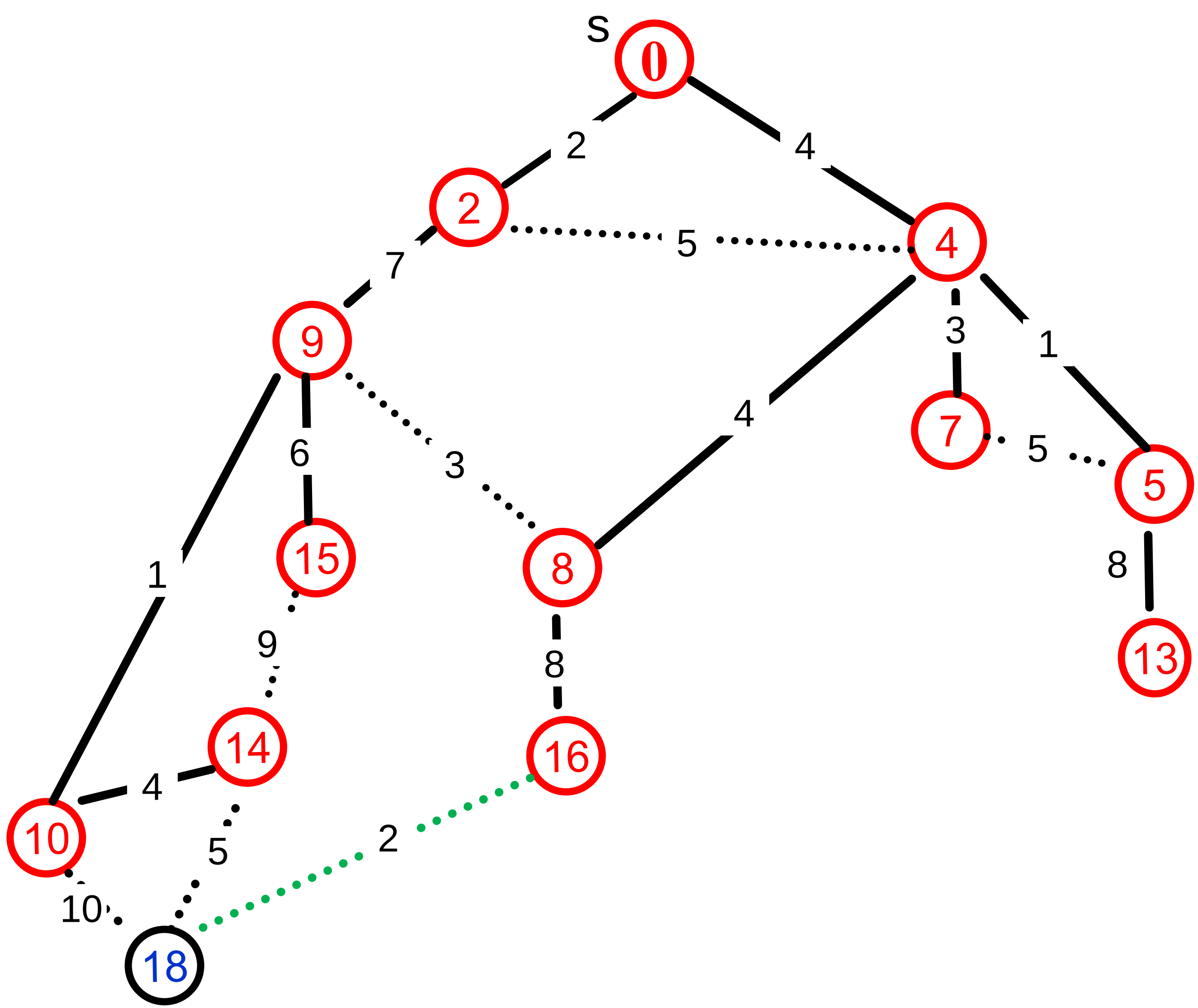
Dijkstra’s Algorithm: Example



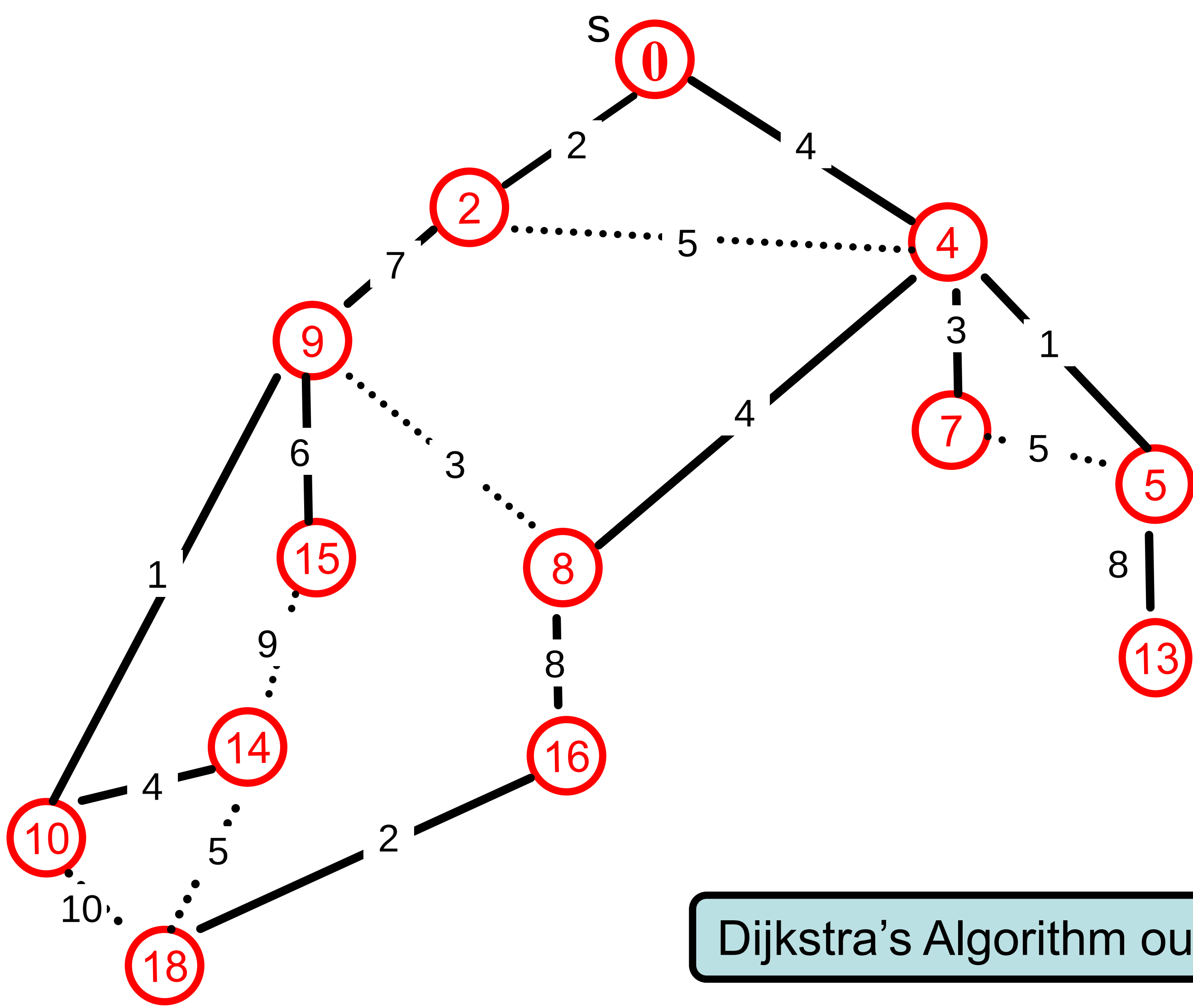
Dijkstra's Algorithm: Example



Dijkstra’s Algorithm: Example



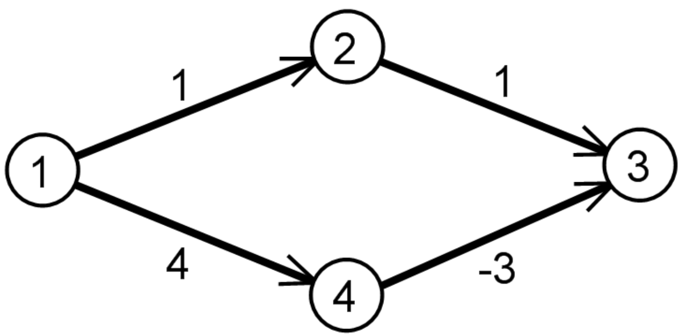
Dijkstra's Algorithm: Example



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 in 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.

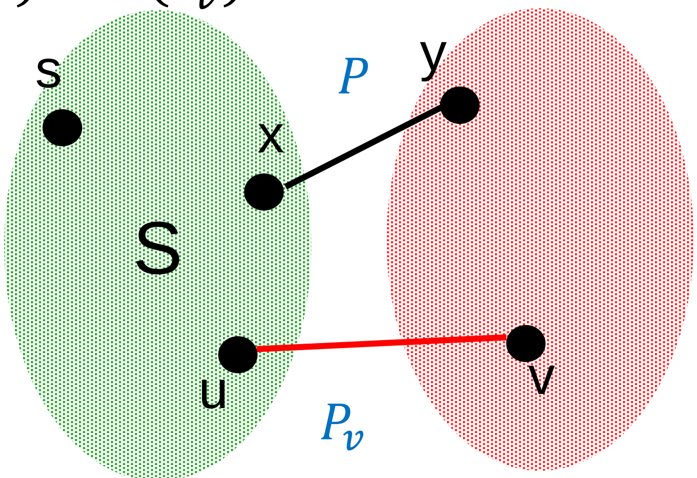
If P_v is not the shortest path, there is a shorter path P to S.

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

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

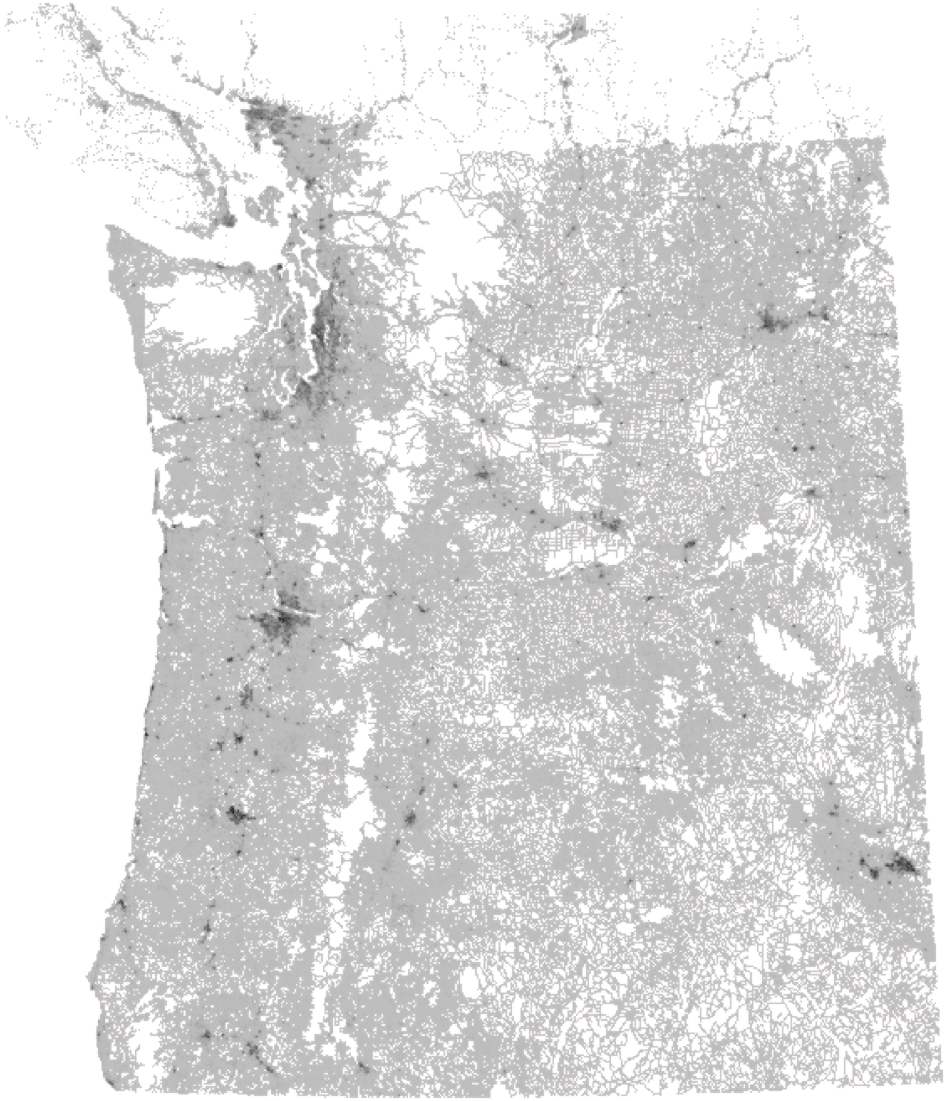
P is the shorter path.

Due to the choice of v



A contradiction.

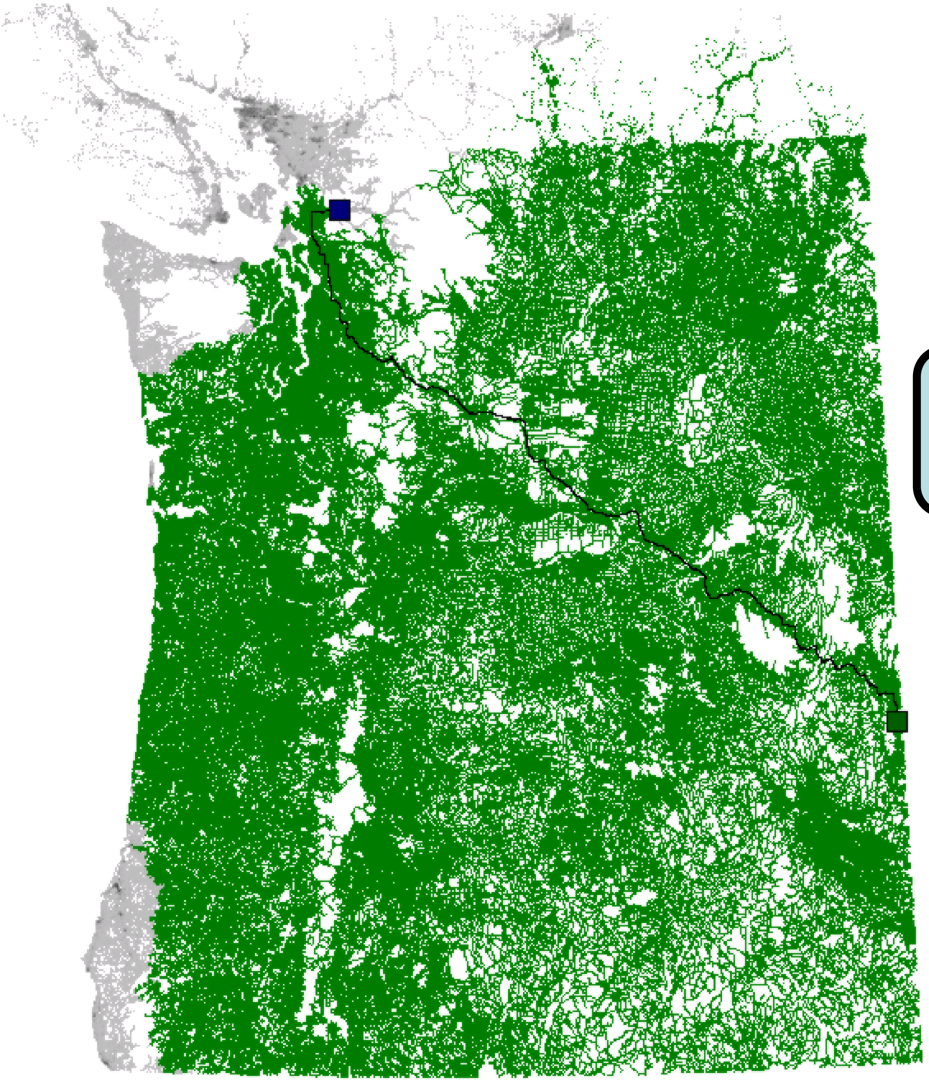
Dijkstra Example



1.6 million vertices
3.8 million edges
Distance = travel time.

Images comes from A.V. Goldberg

Dijkstra Example

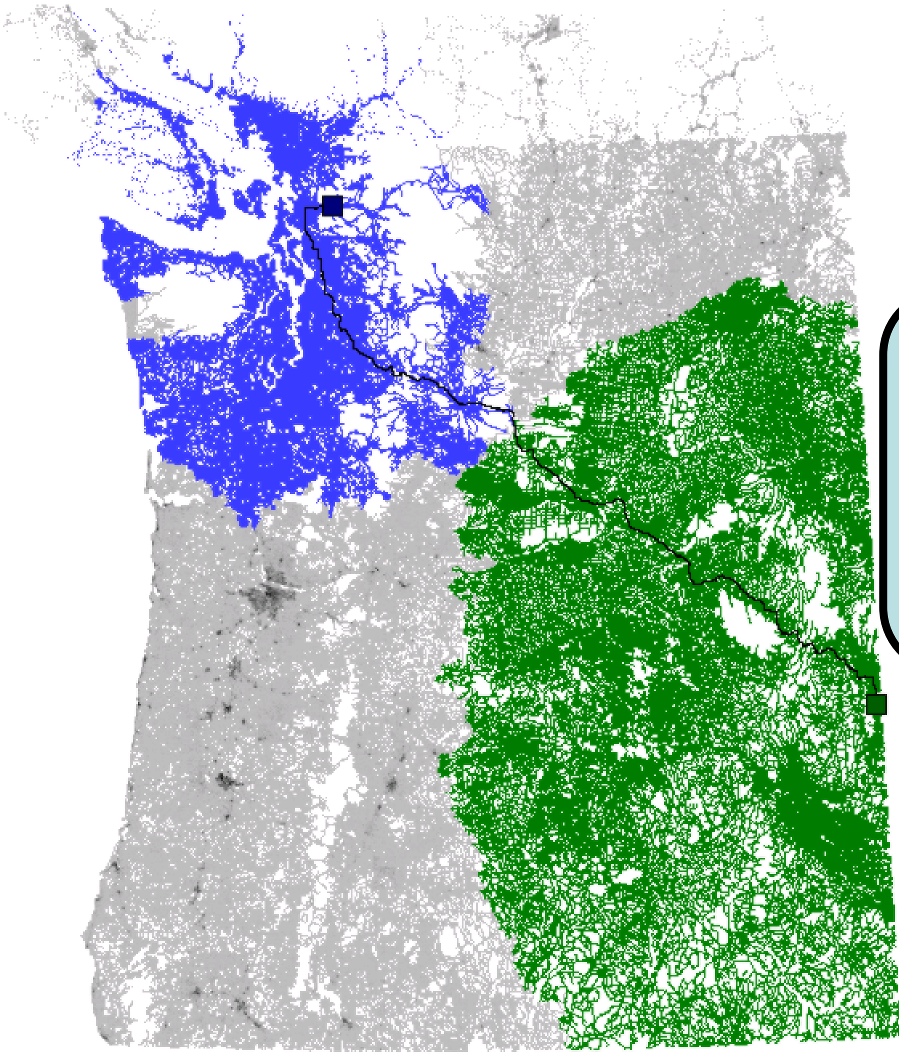


Searched Area
(starting from green point)

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.

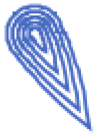
    Add v to S and define $d[v] = d[u] + c_{(u,v)}$.

    $Parent(v) \leftarrow u$.

  }

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)

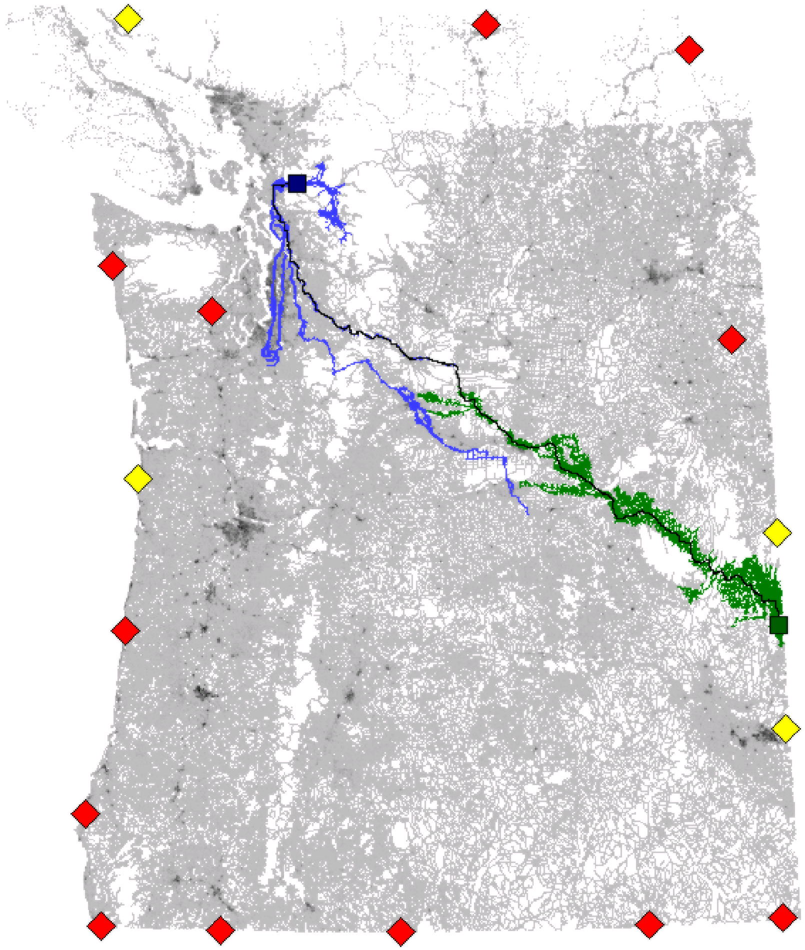
So, $dist(x, l) - dist(t, l)$ is a lower bound for $dist(x, t)$!

Algorithm: Select landmarks l_i, define

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

12ms

A^* + Landmarks + Triangle equality (ATL)



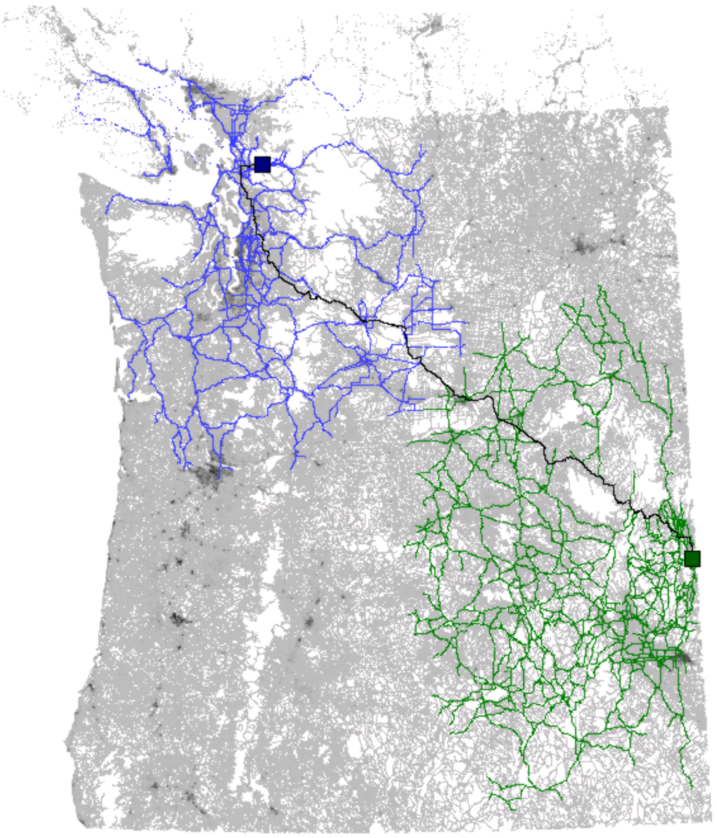
Forward search
Backward search
Inactive landmarks
Active landmarks

Problem of ATL:
We should stick with highway!

From now on, we allow to preprocess the graph.

30ms

Reach Algorithm

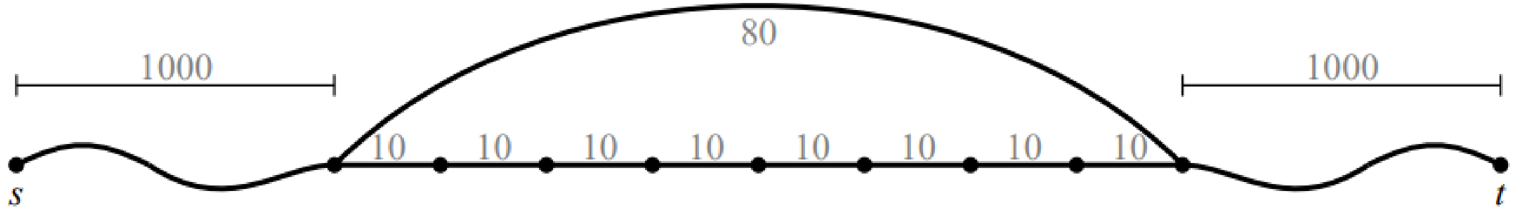


Use highway except for the beginning and the end of the journey!

Forward search
Backward search

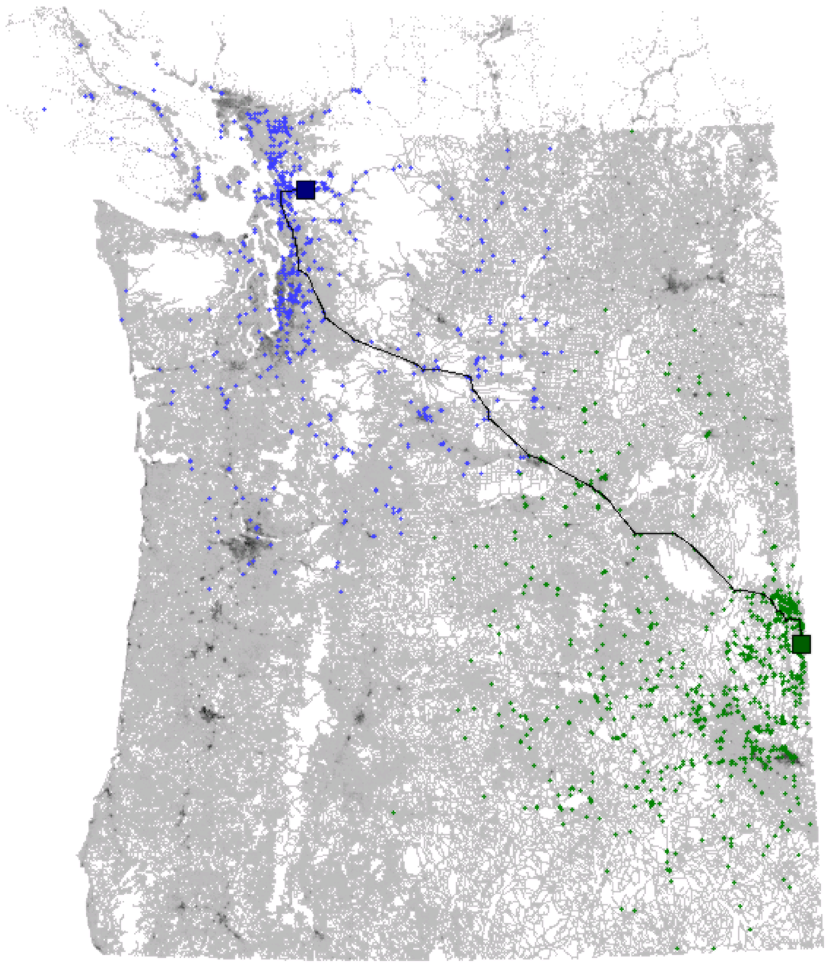
Creating shortcut in the graph

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