CSE 421

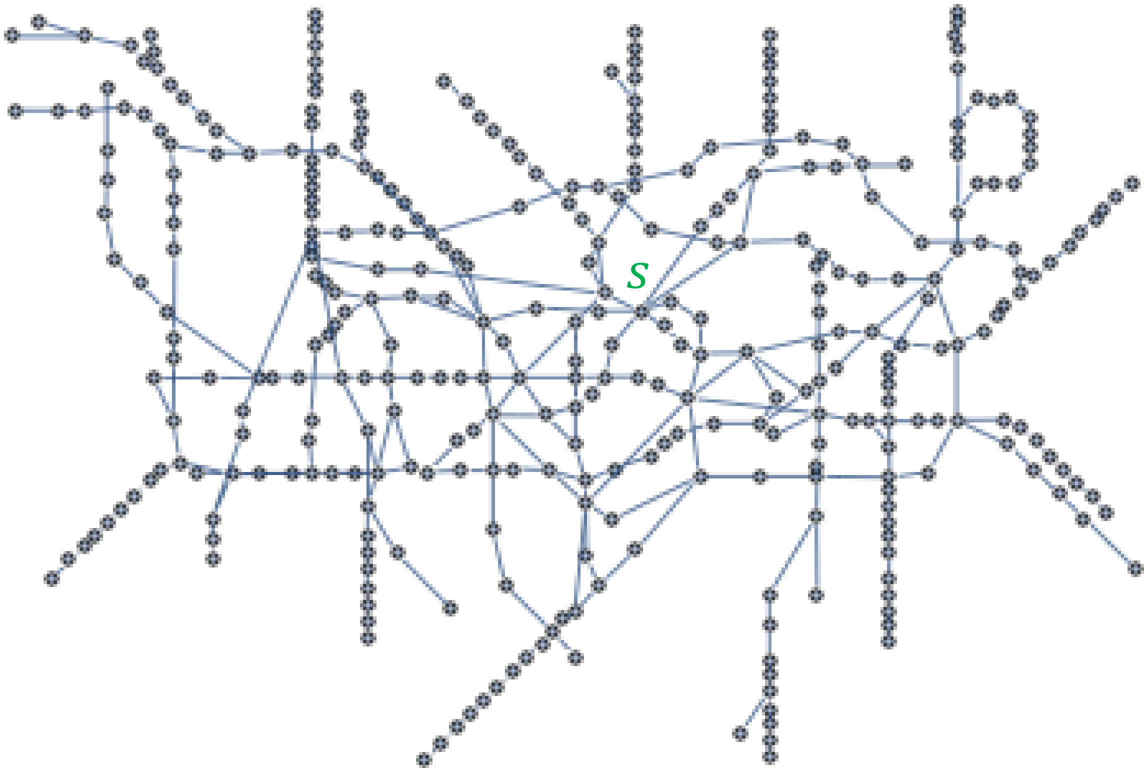
Greedy Algorithms / Dijkstra's Algorithm

Yin Tat Lee

Single Source Shortest Path

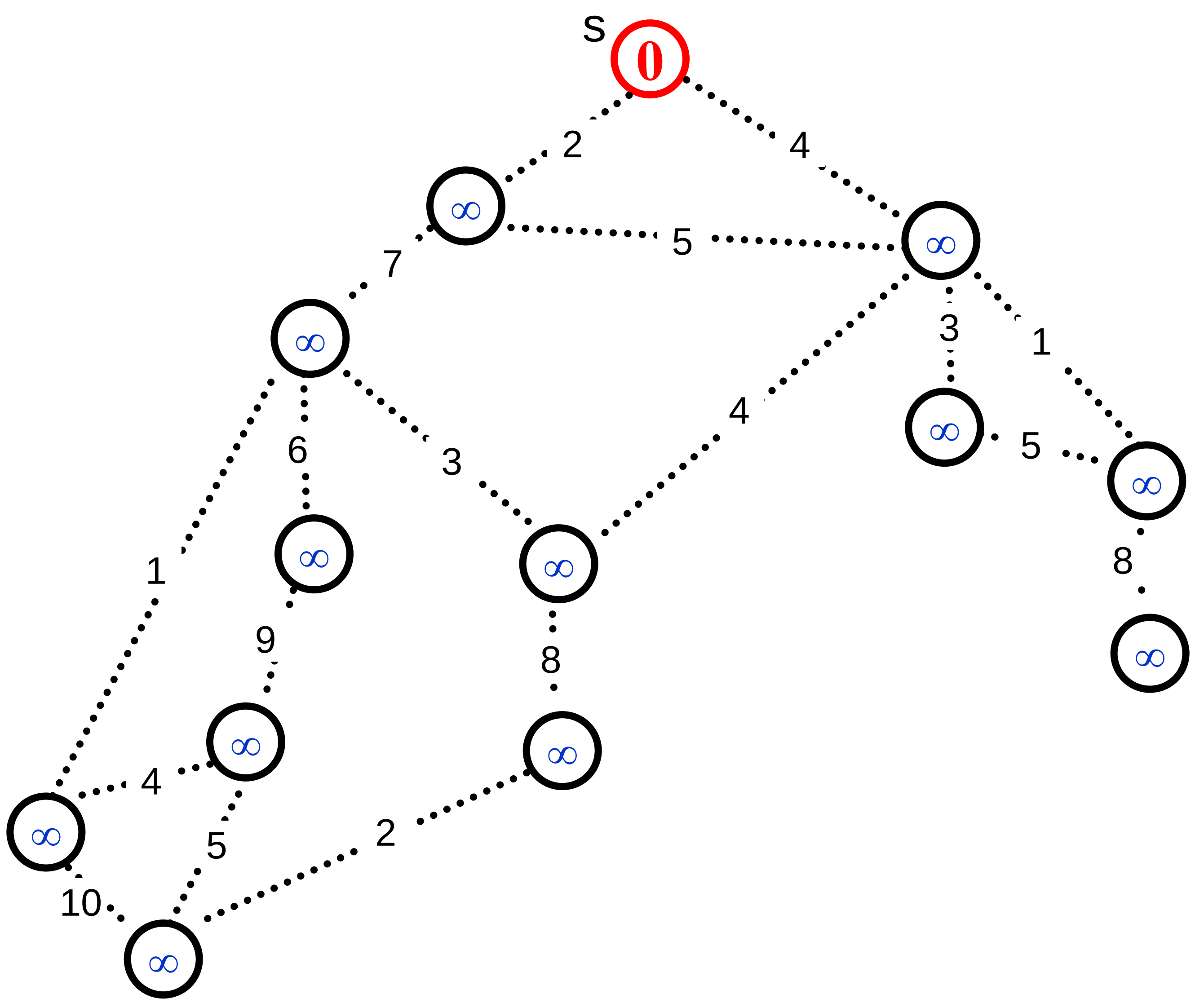
Given an (un)directed graph $G = (V, E)$ with non-negative edge weights $c_e \geq 0$ and a start vertex s.

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

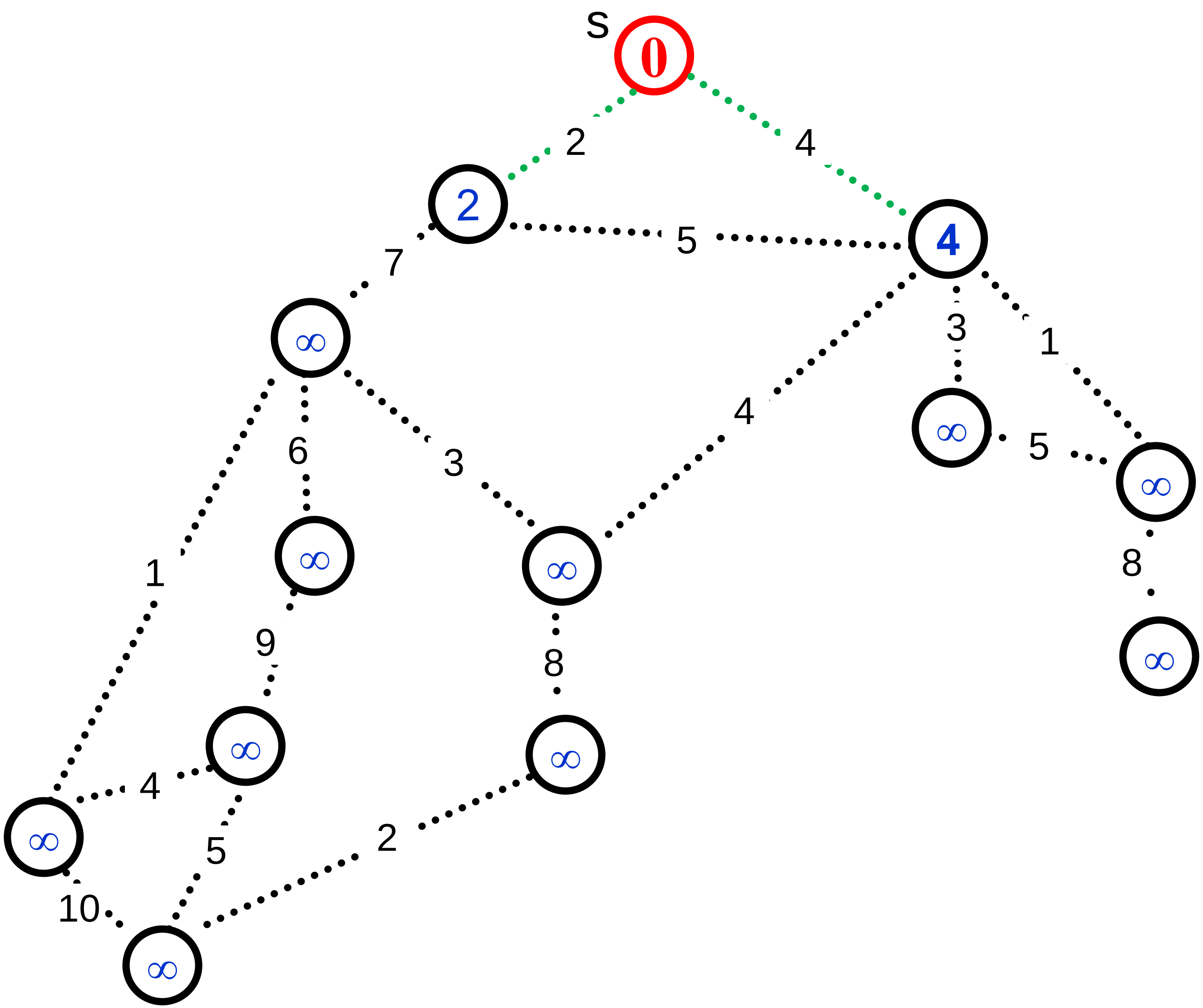


```
Dijkstra(G,c,s) {
   Initialize set of explored nodes S ← {s}

   // 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.
   }
```

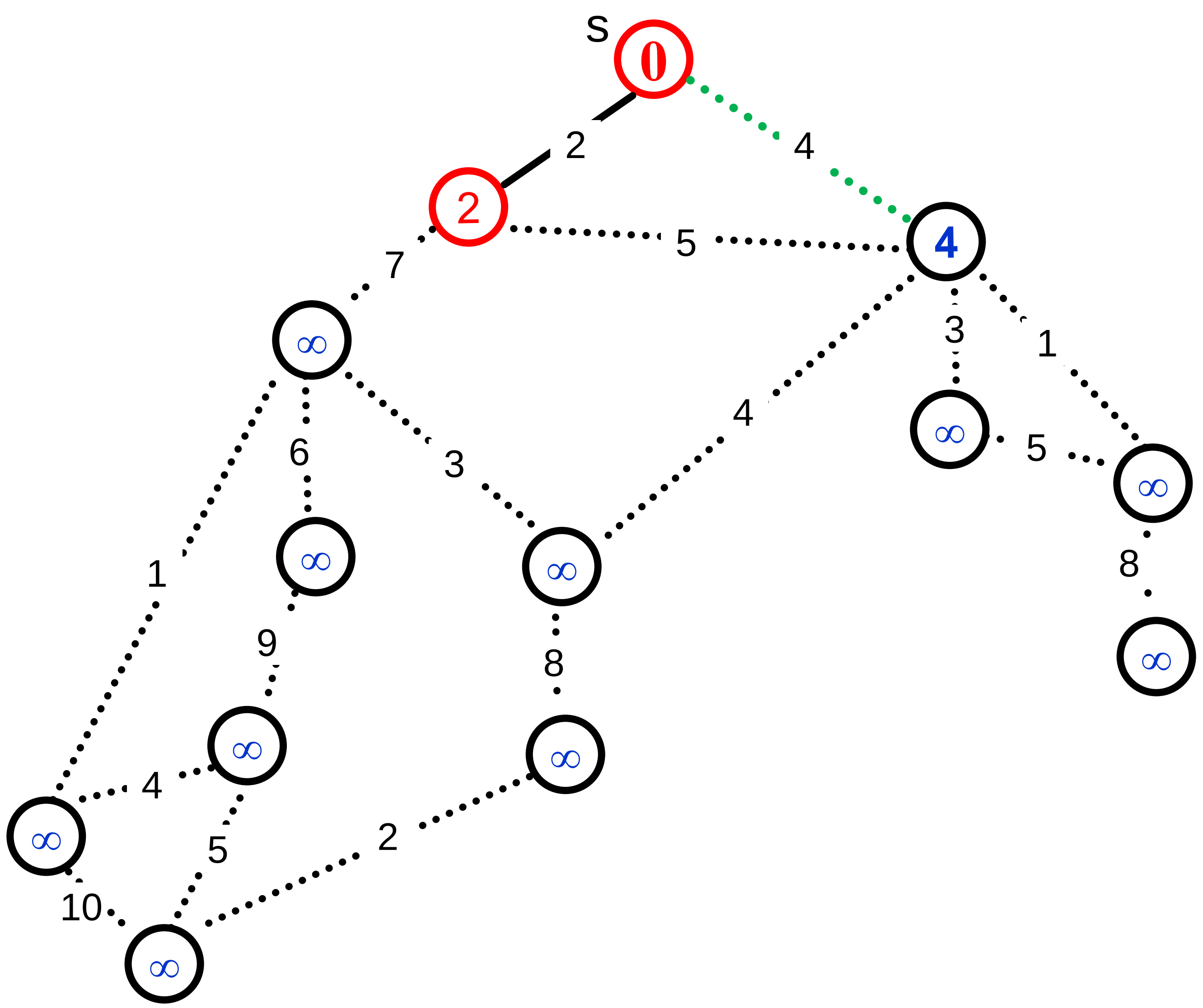
Dijkstra's Algorithm: Example



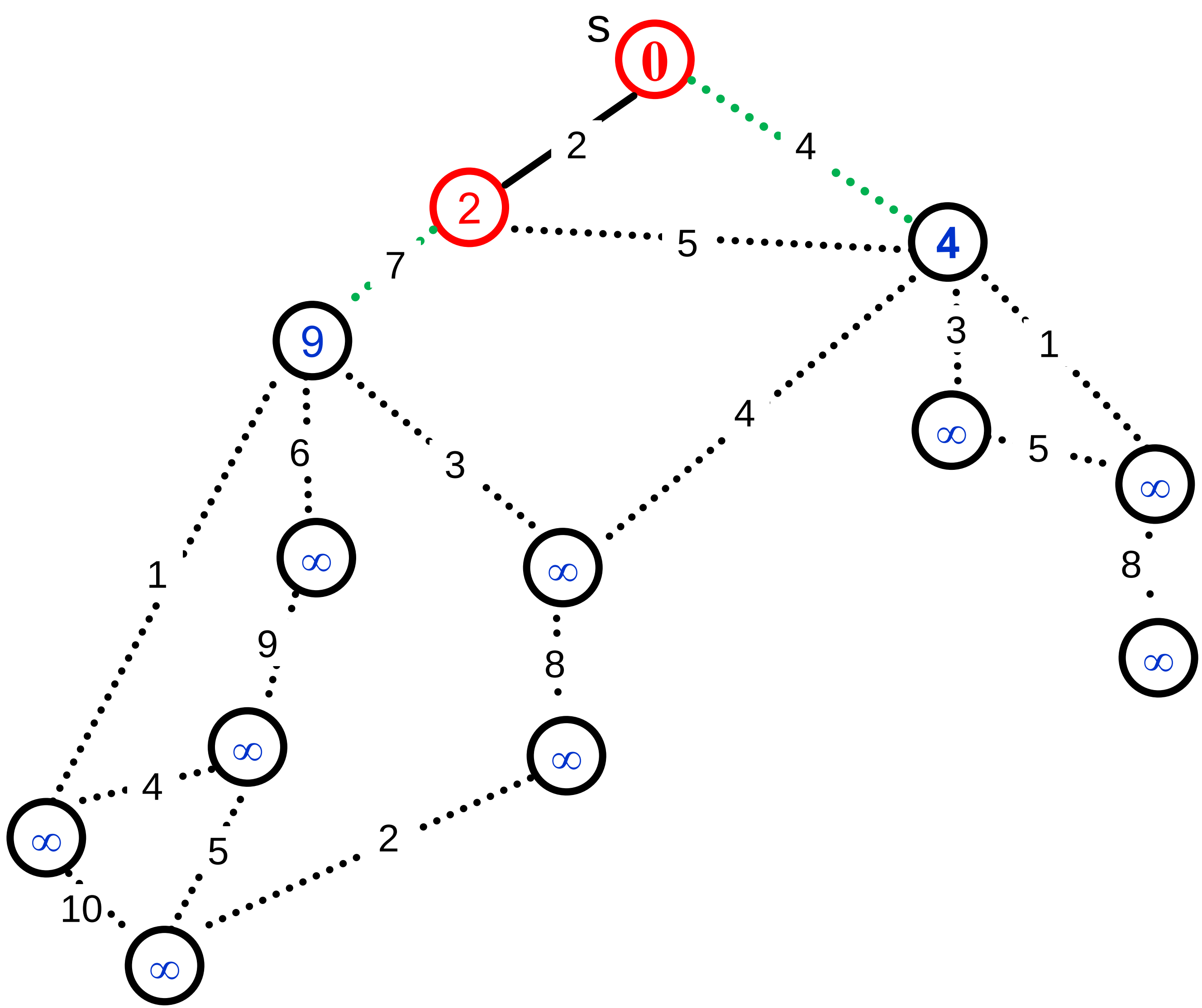
Dijkstra's Algorithm: Example



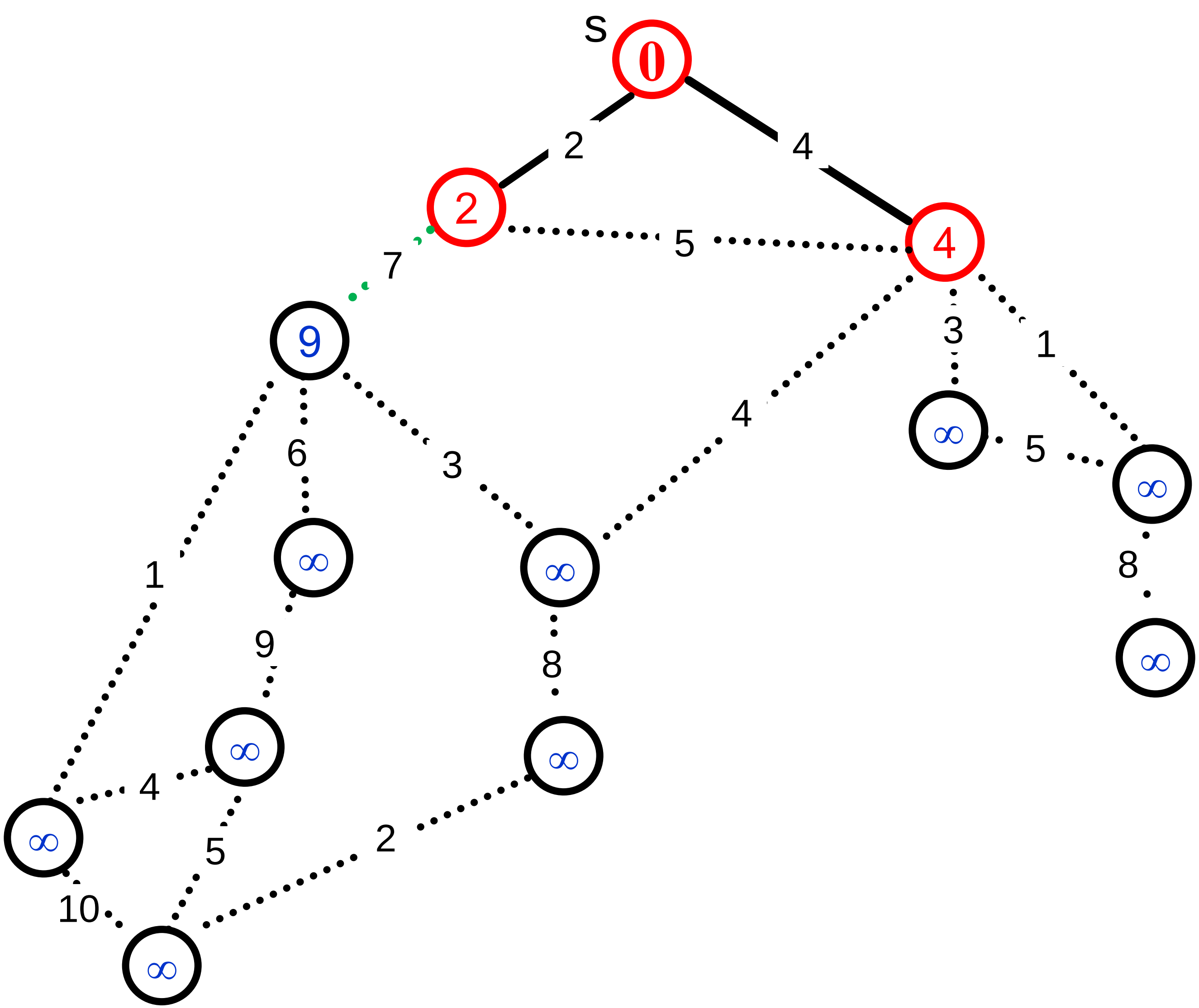
Dijkstra's Algorithm: Example



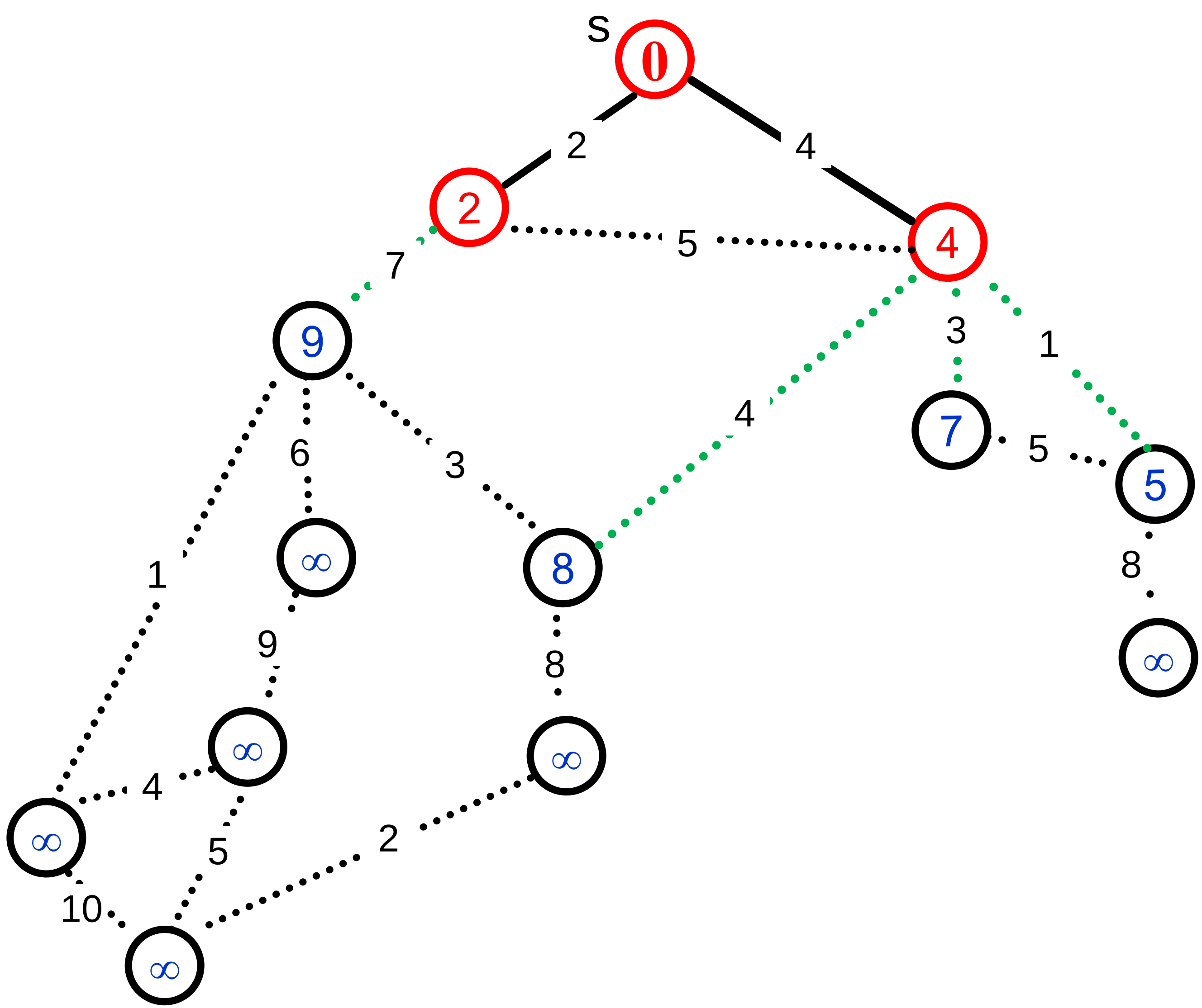
Dijkstra's Algorithm: Example



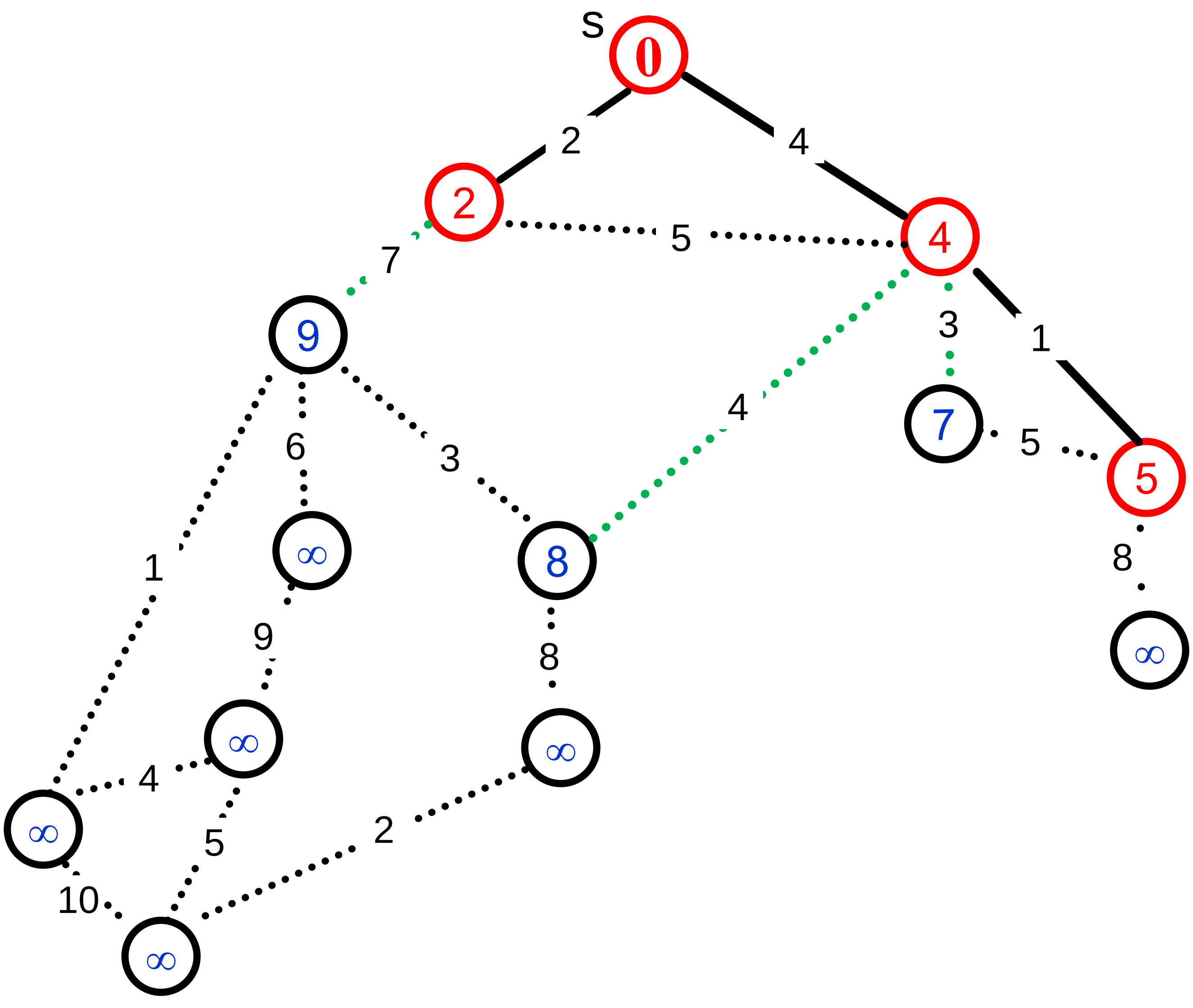
Dijkstra’s Algorithm: Example



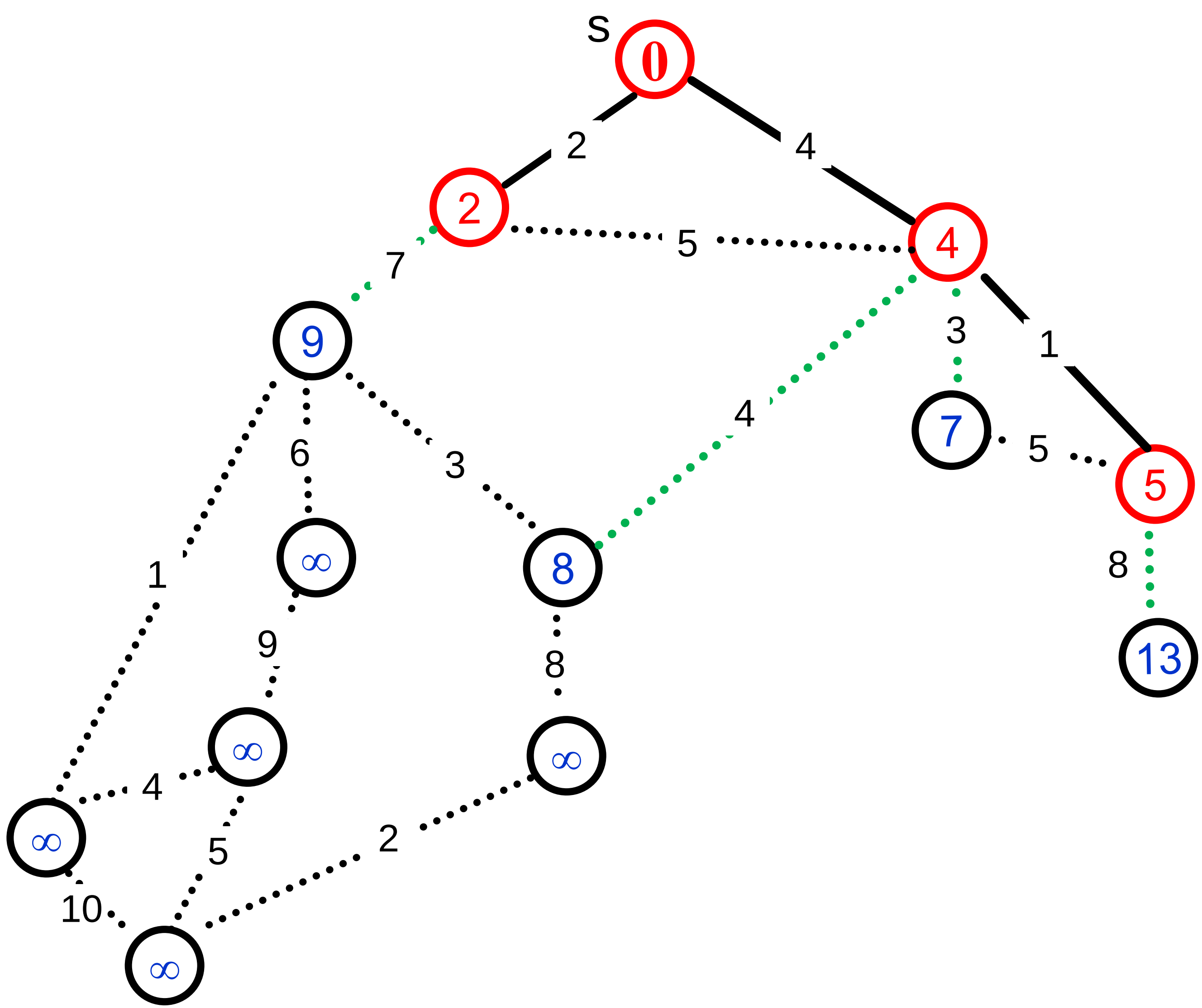
Dijkstra's Algorithm: Example



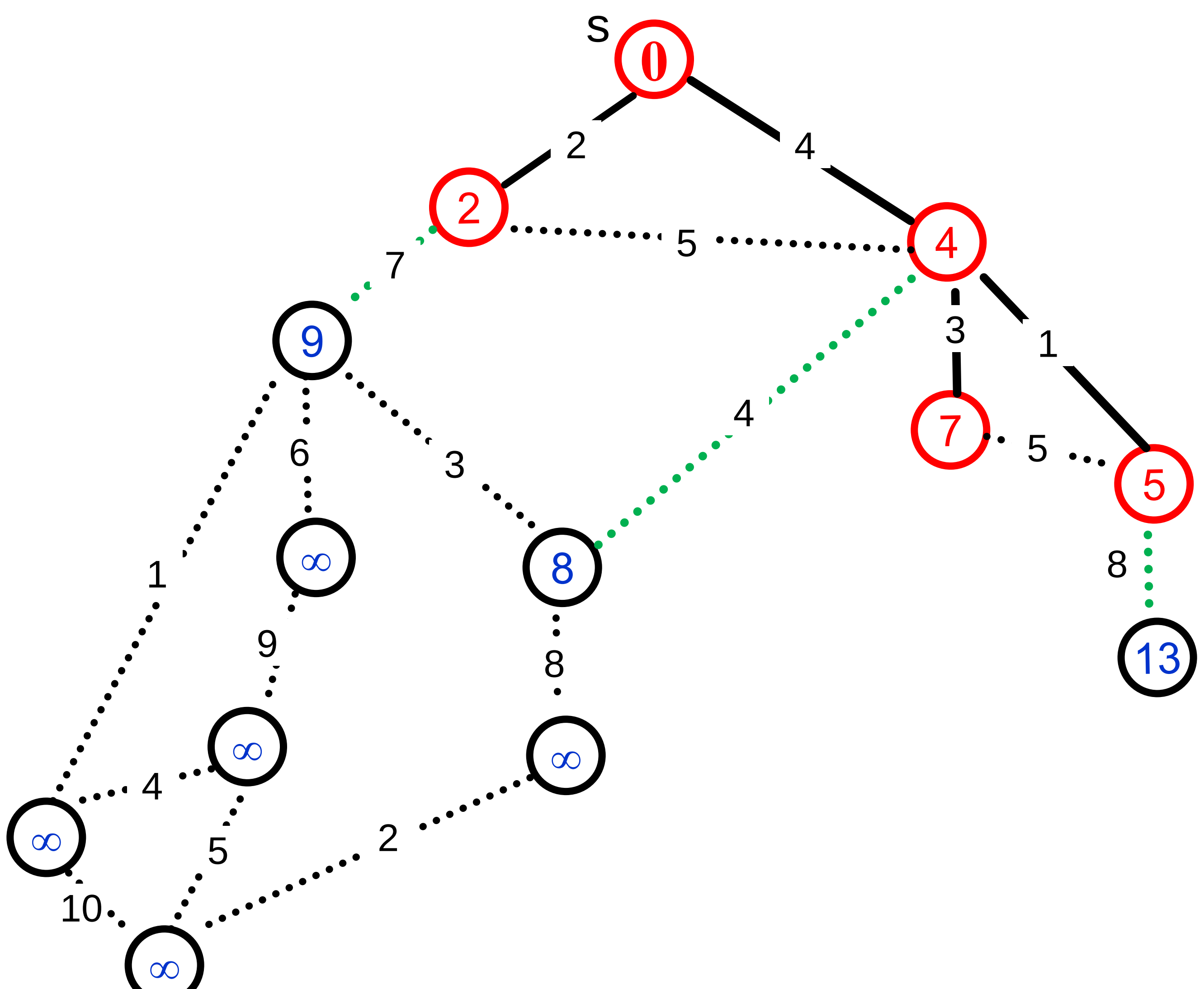
Dijkstra’s Algorithm: Example



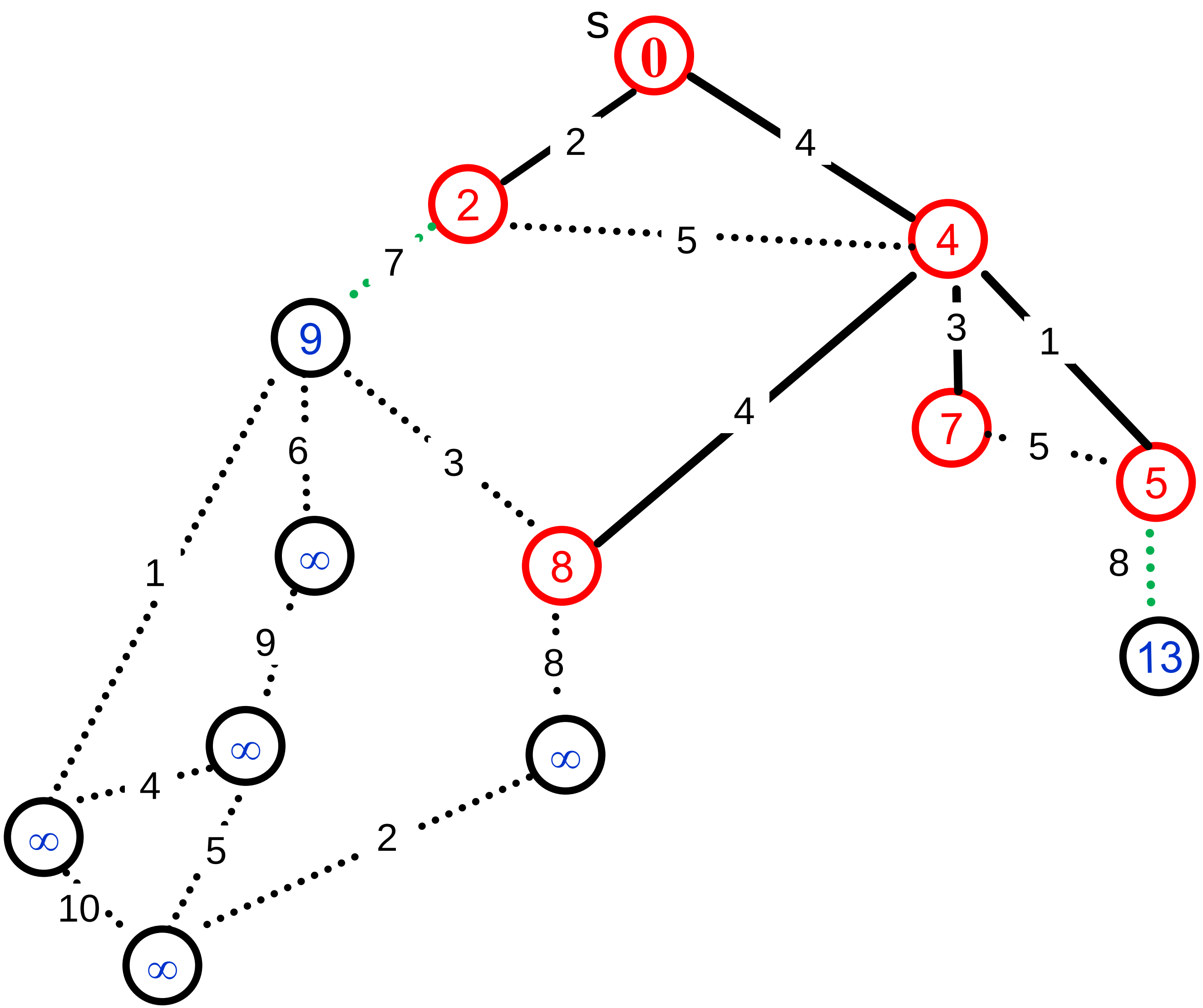
Dijkstra’s Algorithm: Example



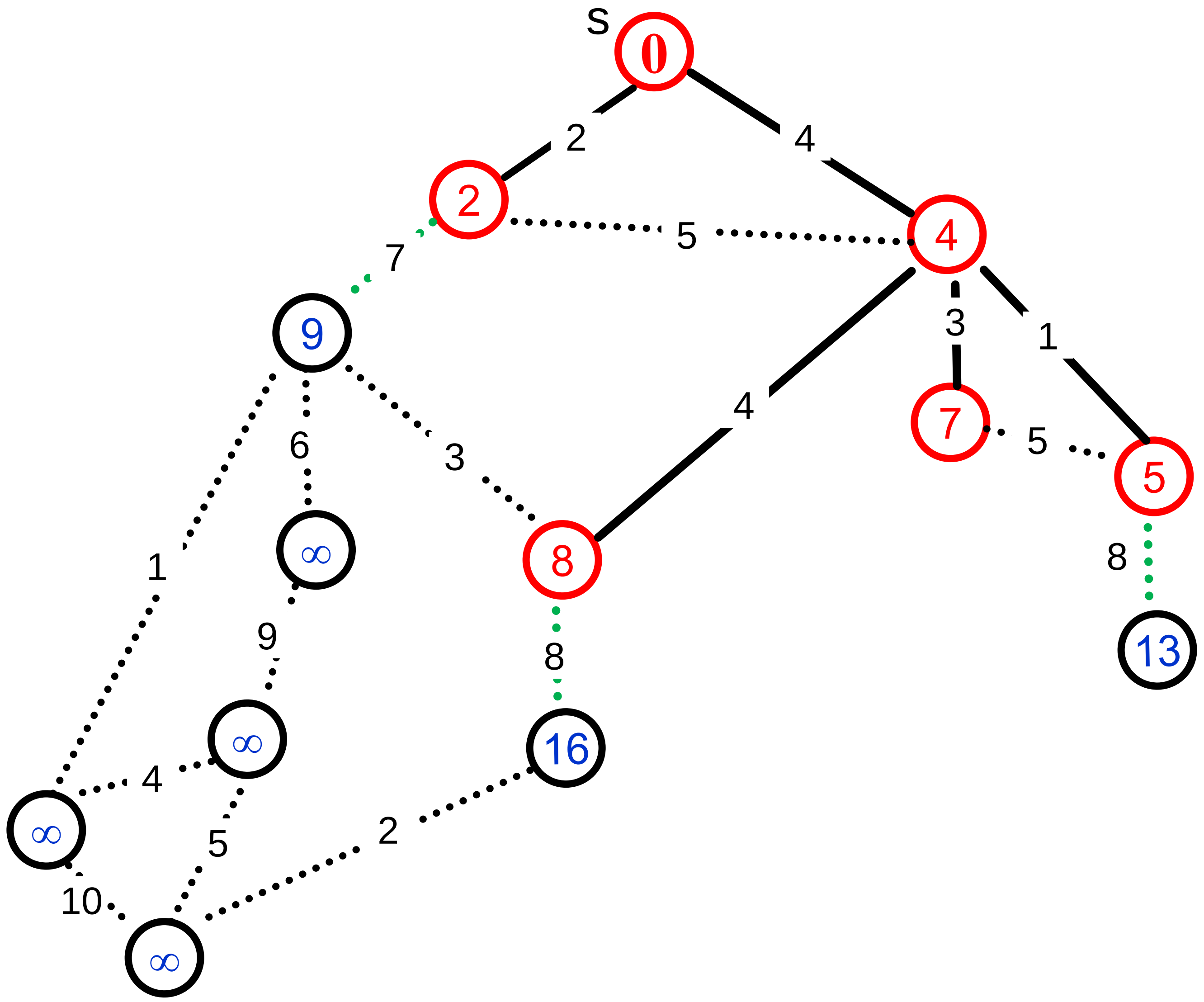
Dijkstra’s Algorithm: Example



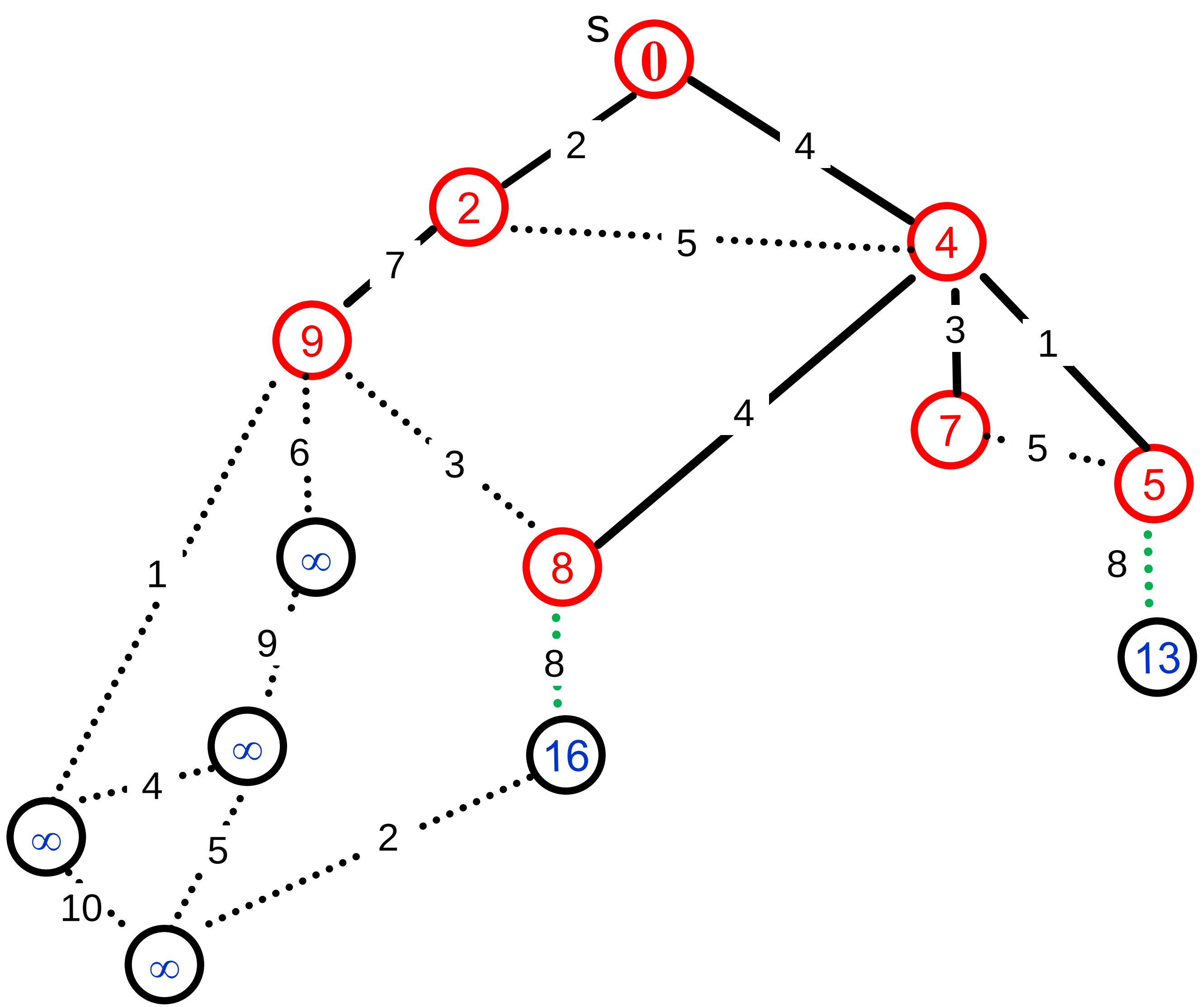
Dijkstra’s Algorithm: Example



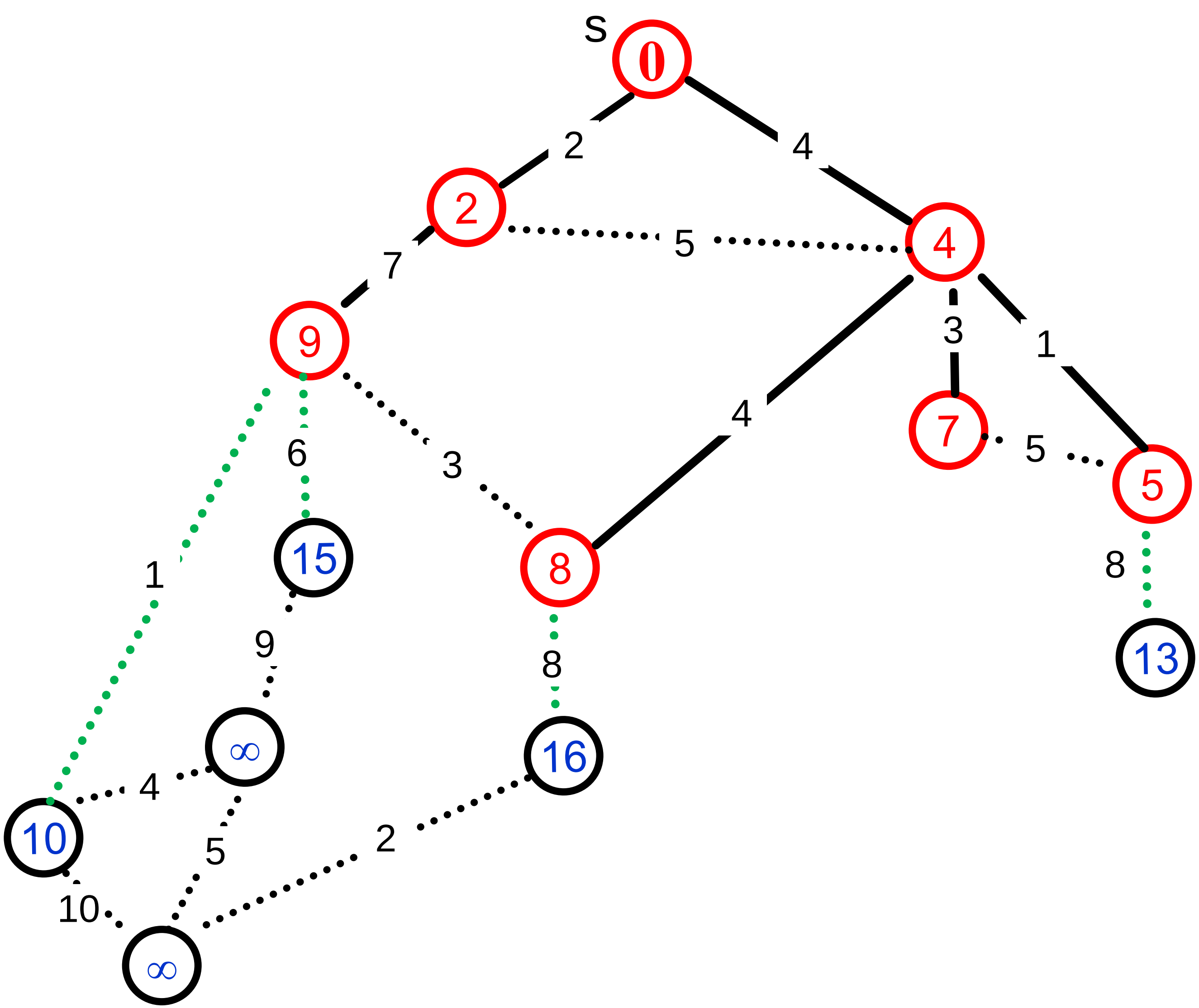
Dijkstra’s Algorithm: Example



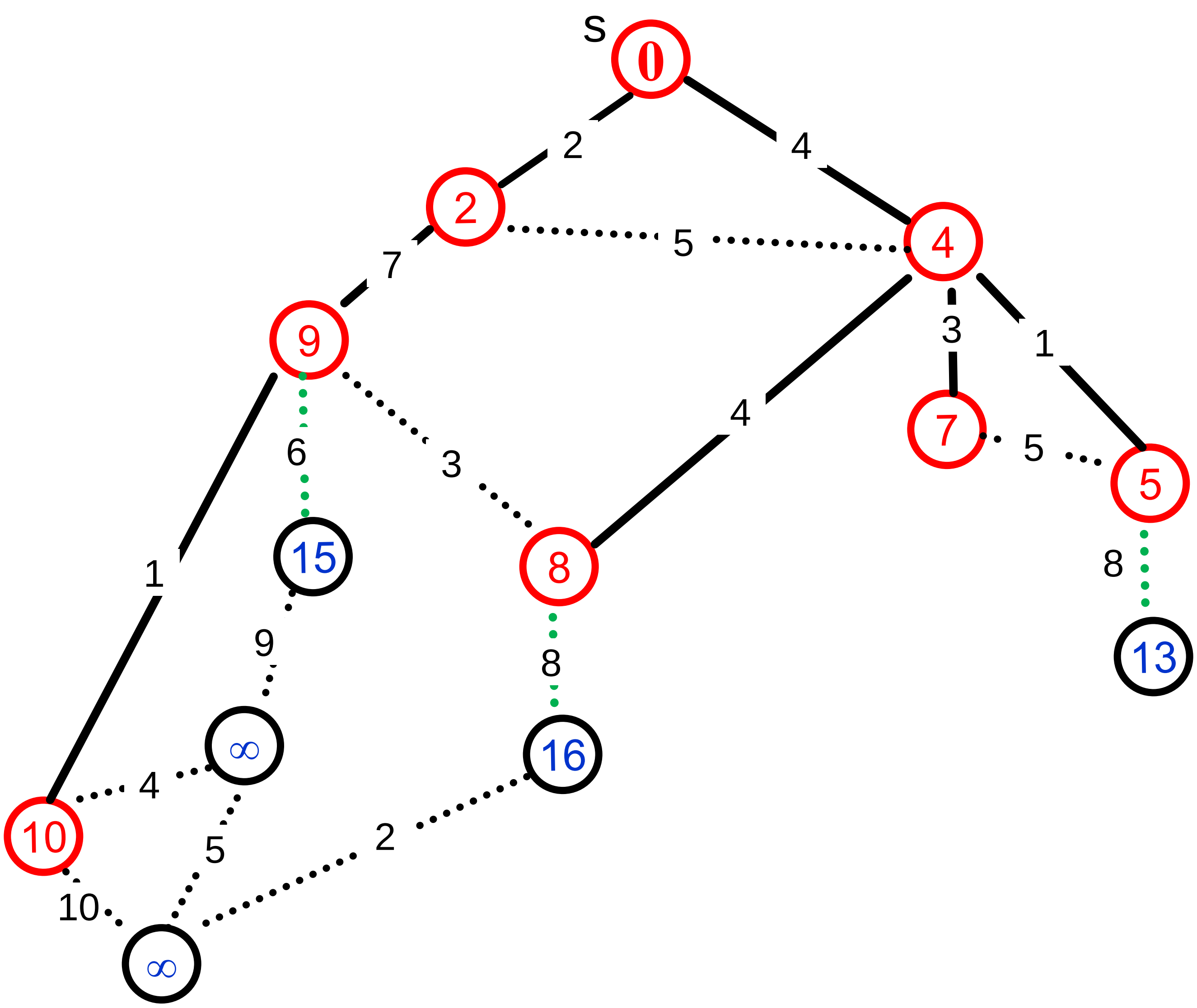
Dijkstra’s Algorithm: Example



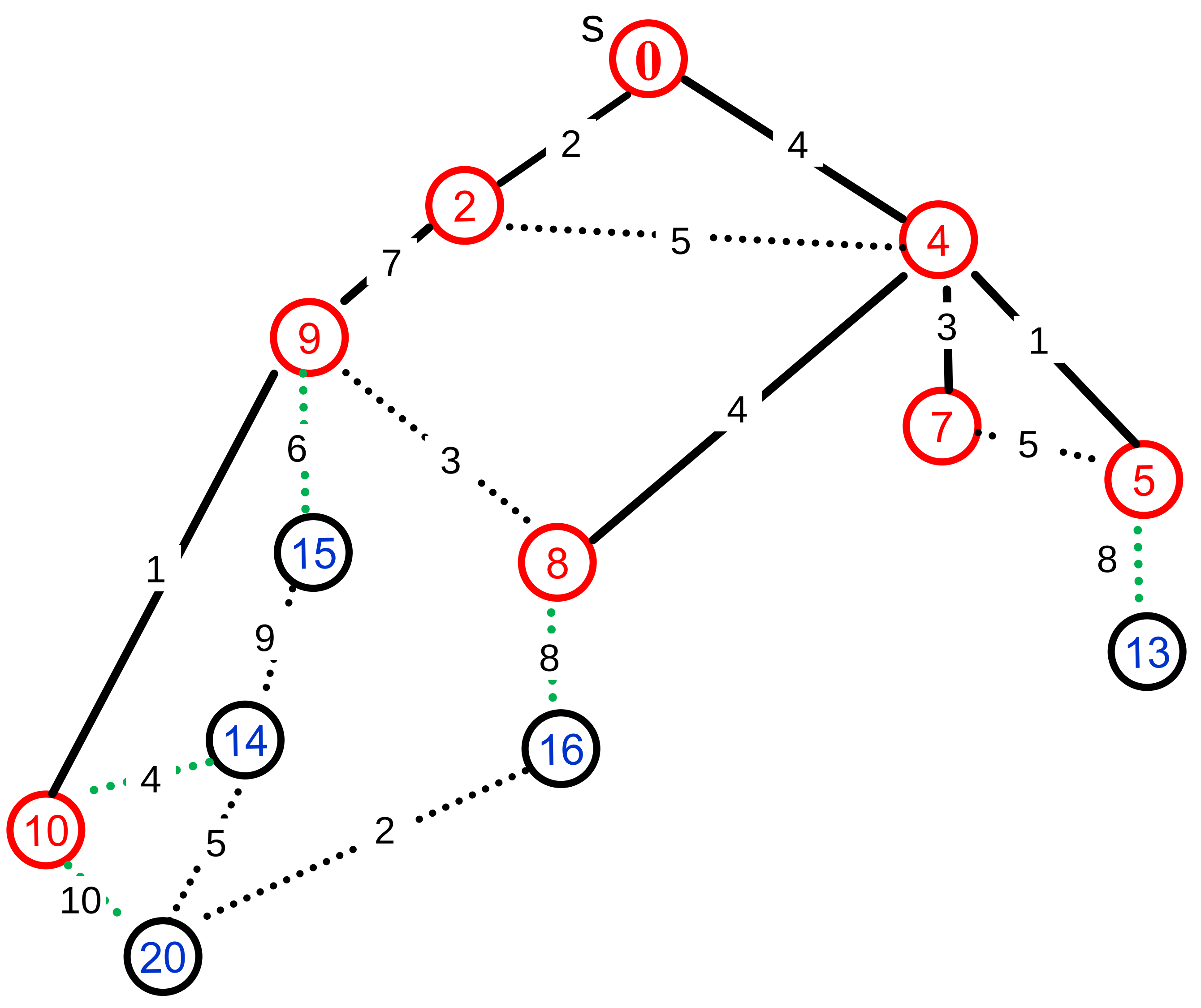
Dijkstra's Algorithm: Example



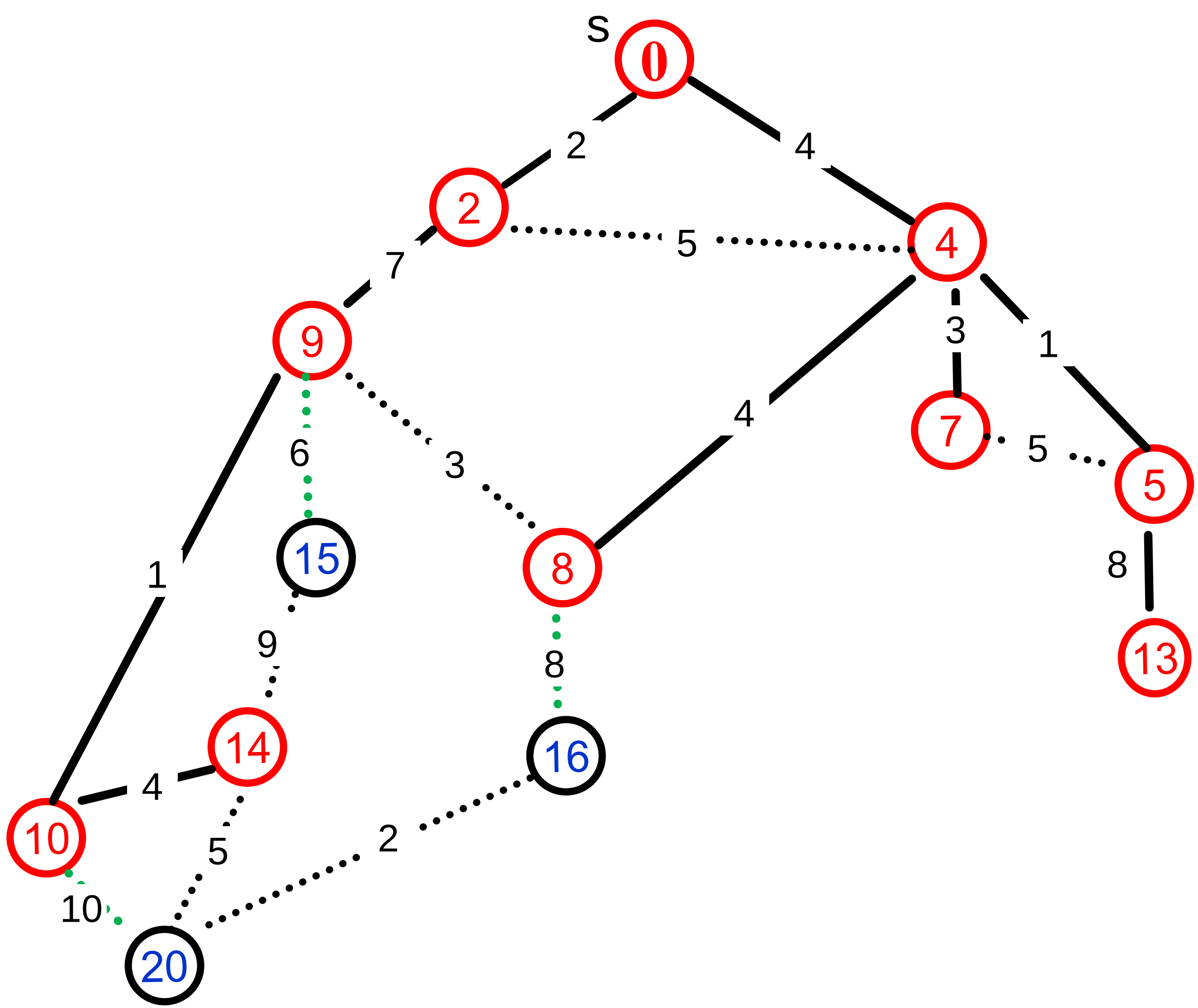
Dijkstra's Algorithm: Example



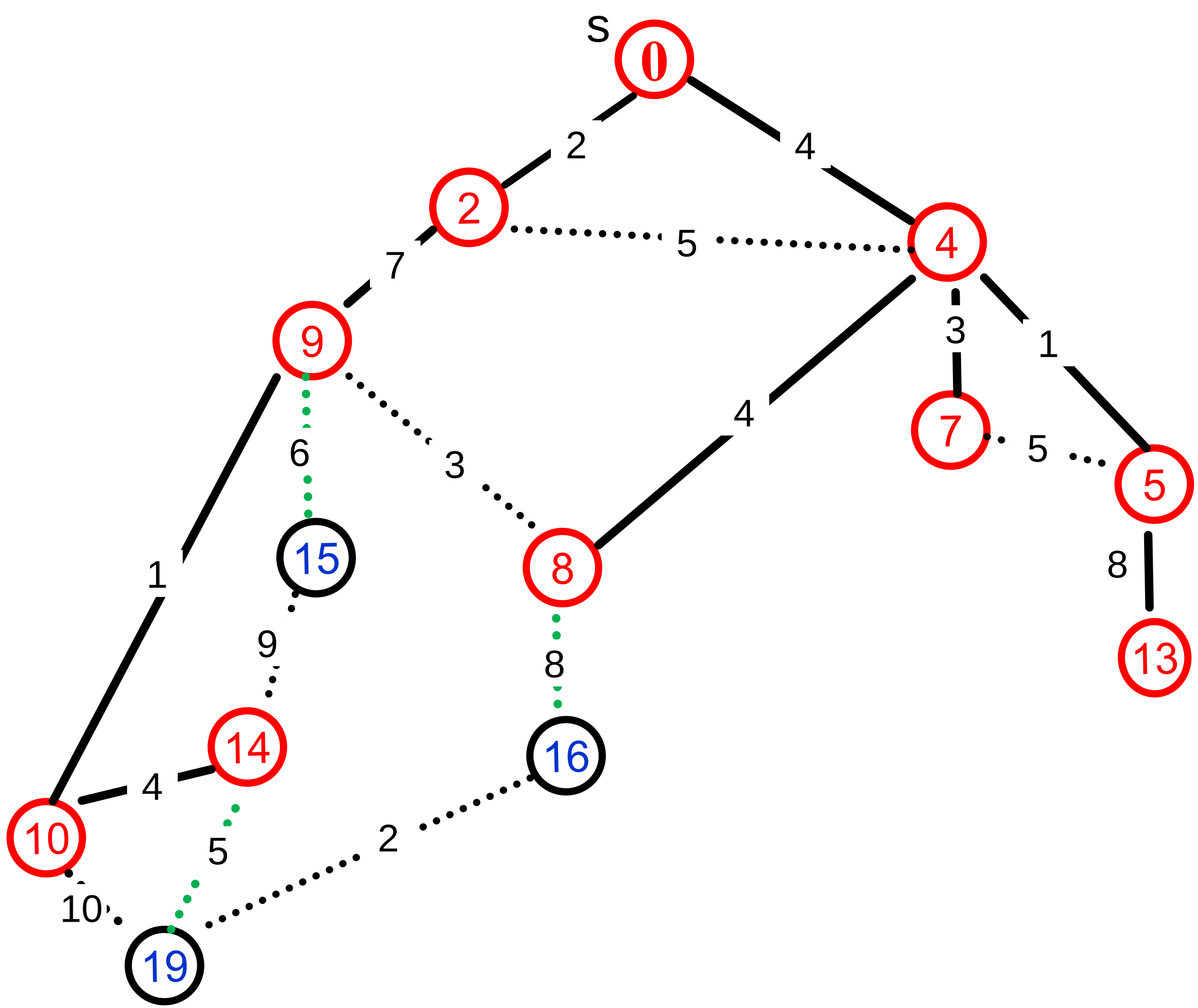
Dijkstra's Algorithm: Example



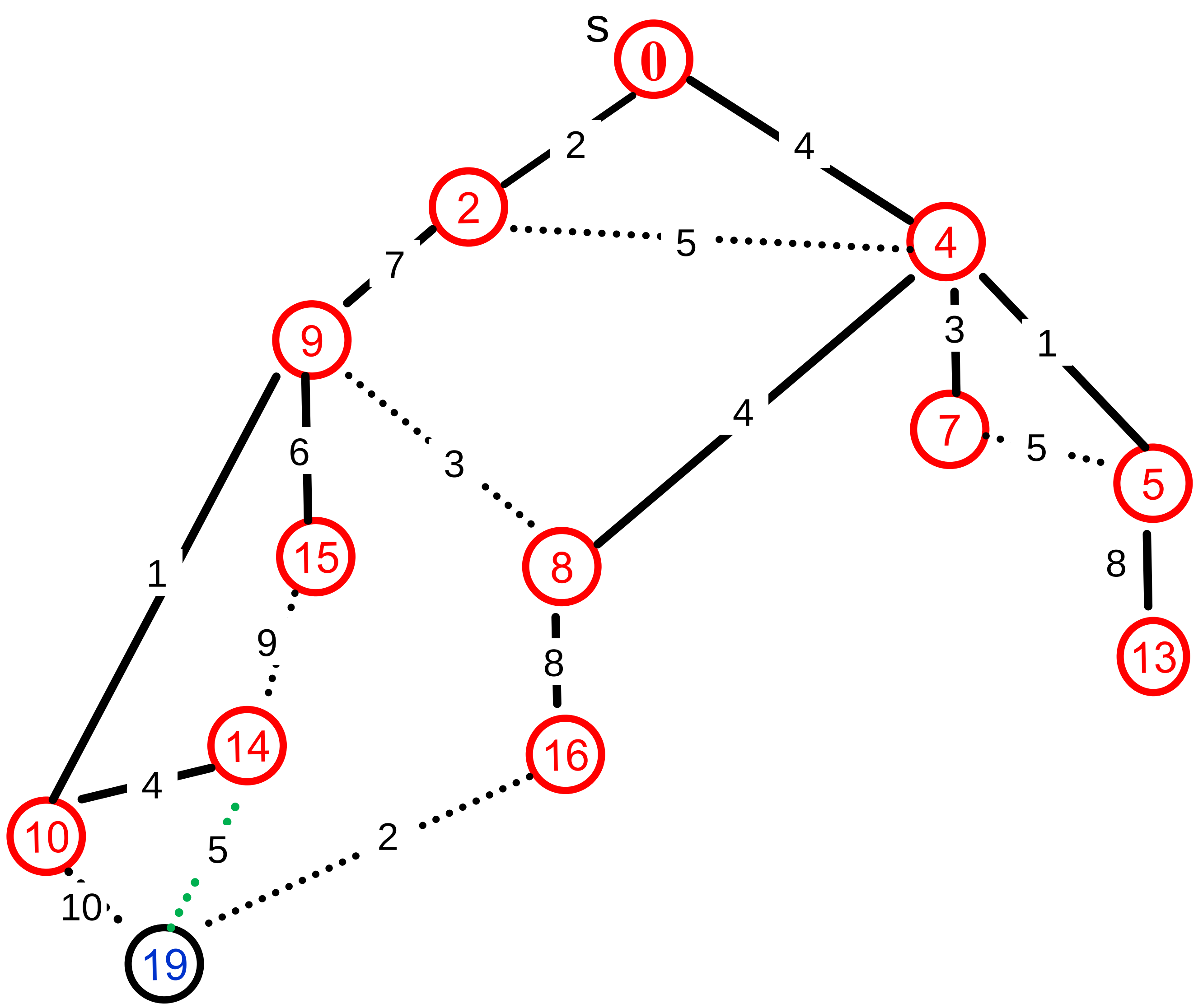
Dijkstra's Algorithm: Example



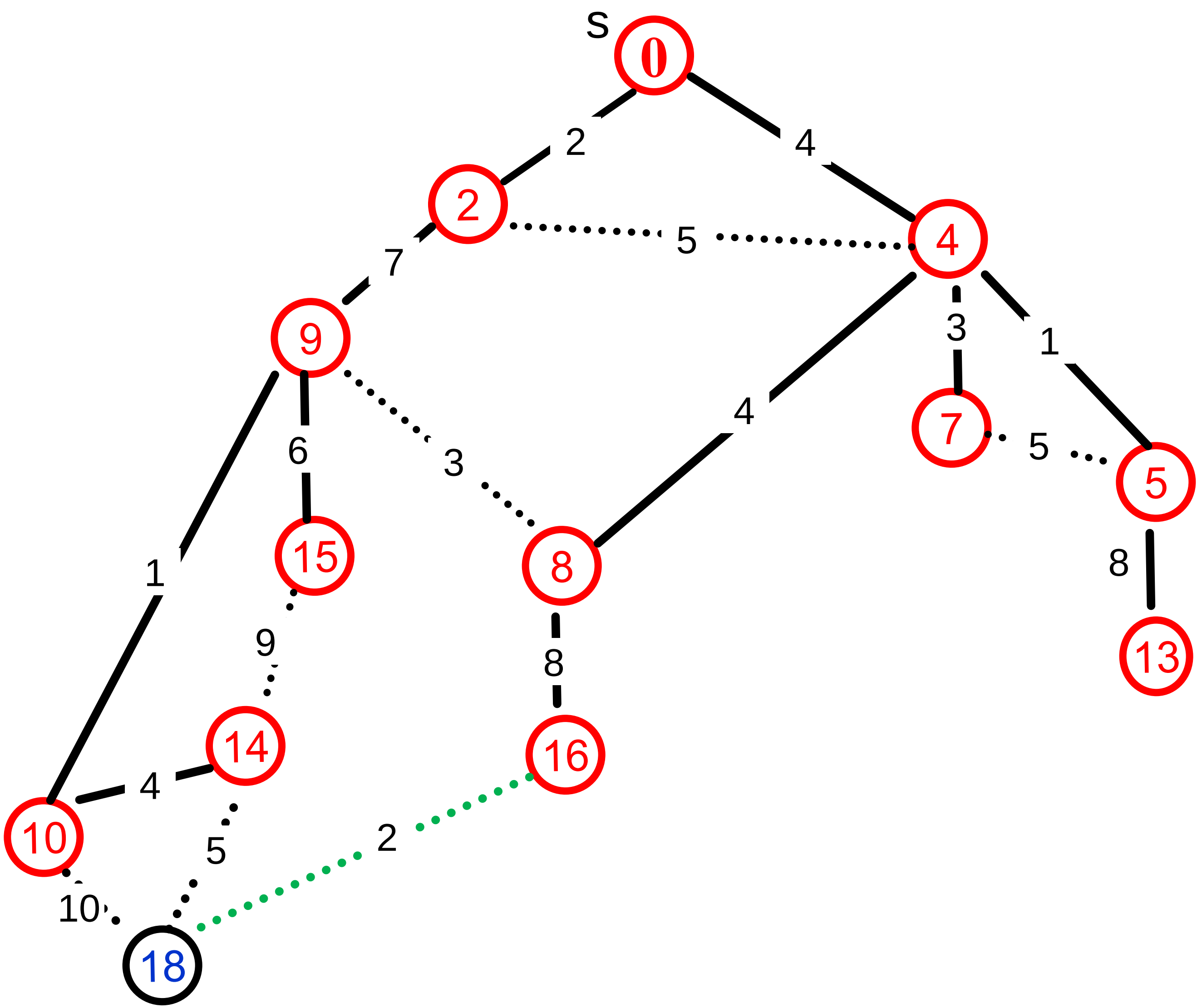
Dijkstra’s Algorithm: Example



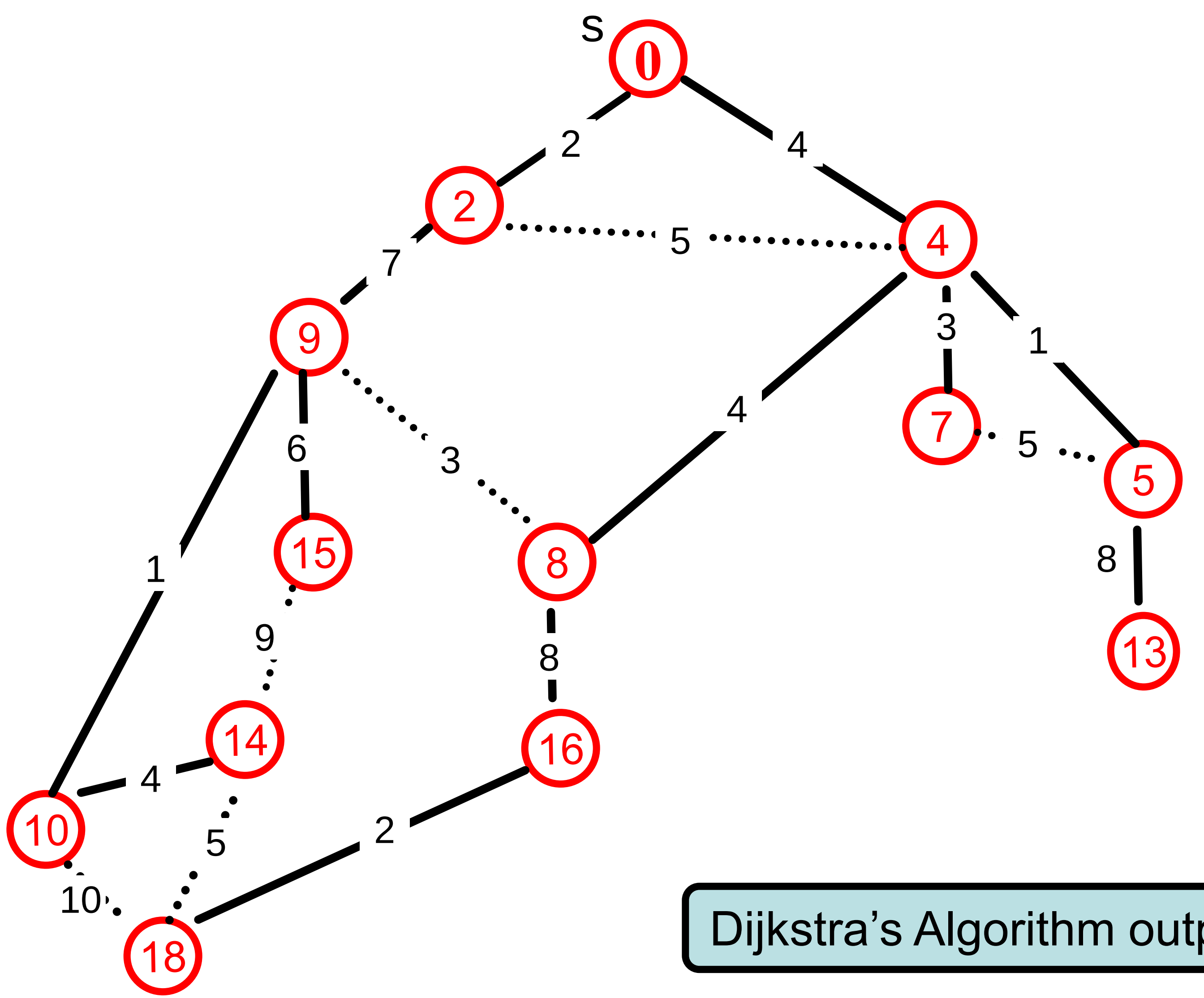
Dijkstra's Algorithm: Example



Dijkstra’s Algorithm: Example



Dijkstra's Algorithm: Example



Dijkstra's Algorithm outputs a tree.

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.

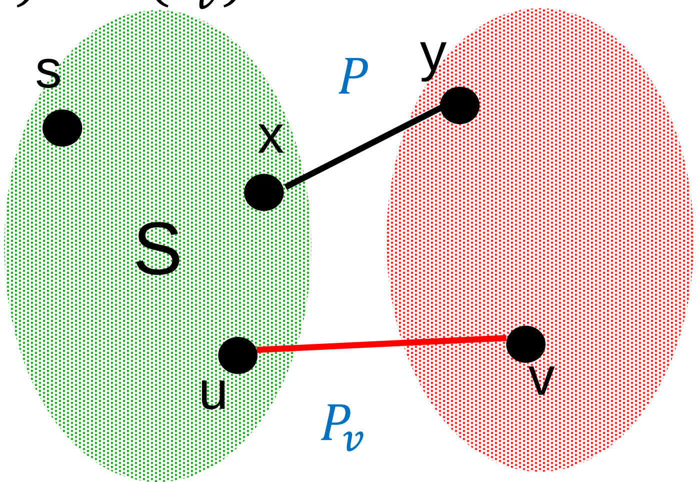
Let (u, v) be last edge on P_v.

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

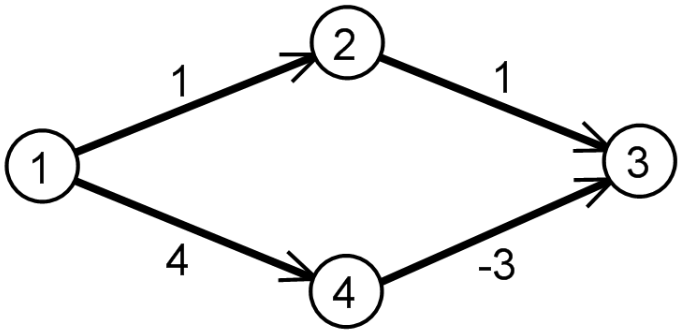
Due to the choice of v



A contradiction.

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.
  - e.g., some airline tickets
- Why does it fail?



- Dijkstra's algorithm is similar to BFS:
  - Substitute every edge with $c_e = k$ with a path of length k, then run 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

Arrays:

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

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

Read wiki!

```
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 $O(1)$

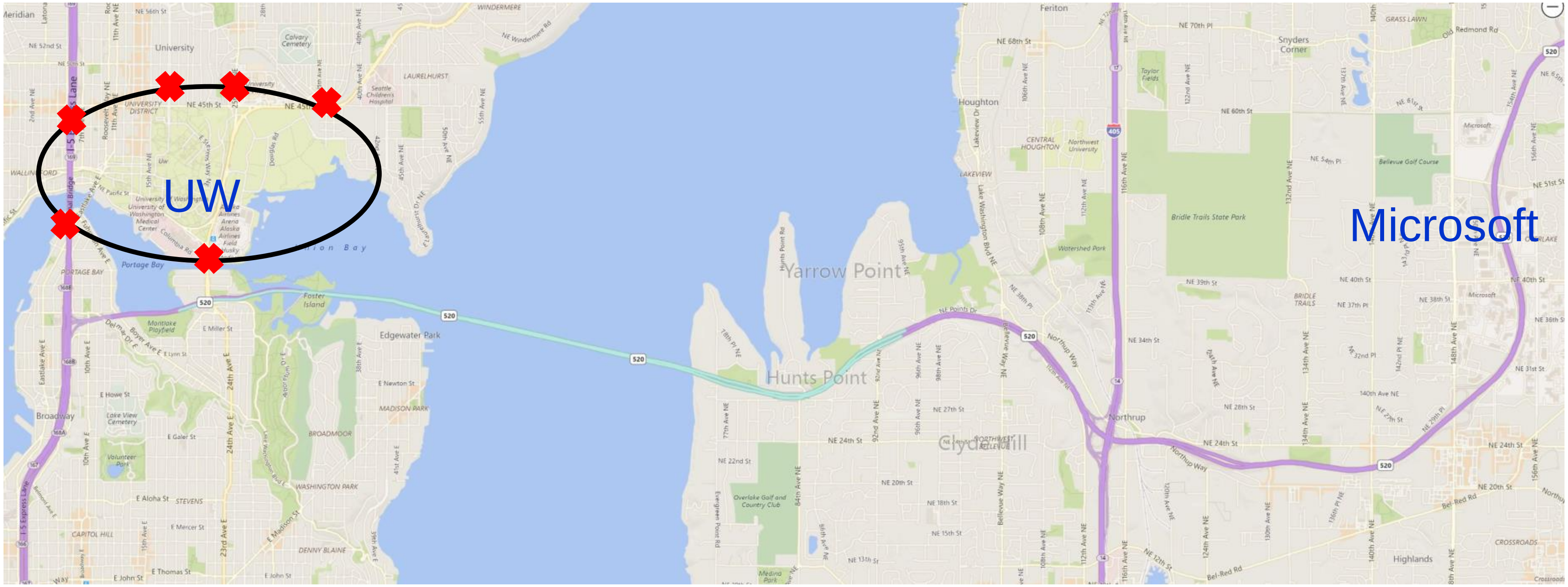
How does Bing Maps work?

Continent-sized road networks have 10s of millions intersections.

Dijkstra's algorithm: few seconds.

How do you go from UW to Microsoft?

For a region, there is a small set of nodes such that all sufficiently long shortest paths out of the region pass a node in the set



This slide modified slides from A.V. Goldberg (a former MSR researcher)

Transit Node (TN) Algorithm

Basic concepts

- divide a map into regions (a few thousand)
- for each region, optimal paths to far away places pass

through one of a small number of access nodes (≈ 10 on the average)

- the union of access nodes is the set of transit nodes (≈ 10 000)

Empirical observation: small number of access/transit nodes

Preprocessing algorithm

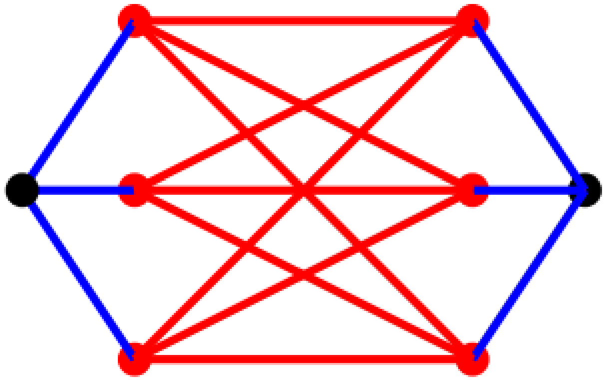
- find access nodes for every region
- connect each vertex to its access nodes
- compute all pairs of shortest paths between transit nodes

Transit Node (TN) Algorithm

The algorithm:

If the query (s, t) is far away, the shortest path is of the form

$$s - \text{access}(s) - \text{access}(t) - t$$



We can create a table look-up for $(\text{access}(s), \text{access}(t))$ pairs.

If the query (s, t) is close, you can do say use Dijkstra.

The precise algorithm is much more complicated.

It can find $s - t$ shortest path in < 300 ns instead of > 1 sec!