CS 401: Computer Algorithm I

Single Source Shortest Path

Xiaorui Sun

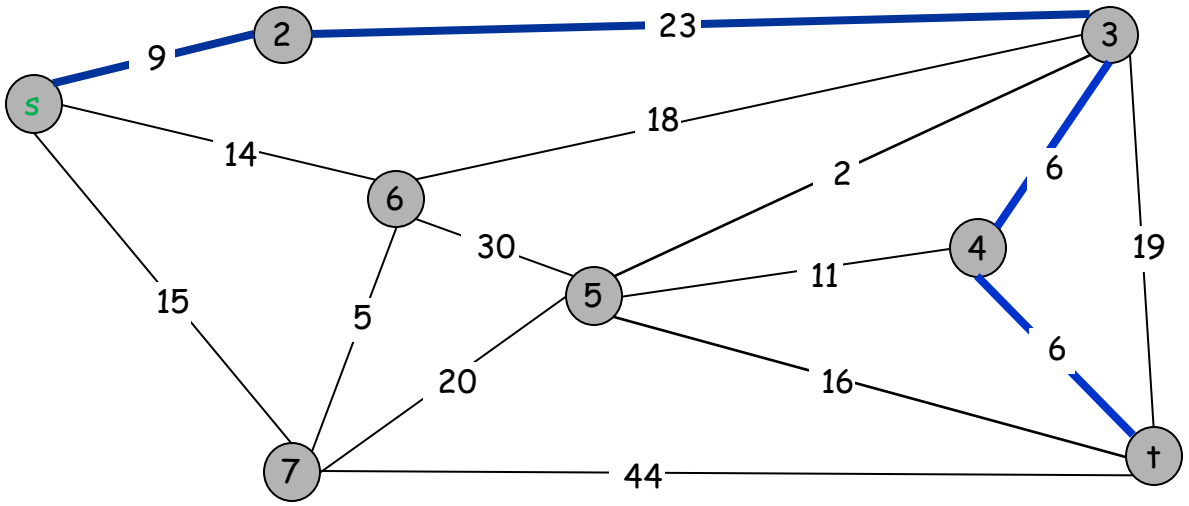
Single Source Shortest Path

Single Source Shortest Path

Given an (un)directed connected 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

length of path = sum of edge weights in path



Cost of path s-2-3-4-t
= 9 + 23 + 6 + 6
= 44.

Single Source Shortest Path

Greedy Recipe:

- Order the input in some way (the most 'important' element will be considered first)
- Go through the input according to the order
- Determine the strategy to construct best current partial solution in each step

For shortest path: How do we order the input?

- Observation: suppose $(s, v_1, v_2, \dots, v_k, t)$ is a shortest path from s to t, then $(s, v_1, v_2, \dots, v_k)$ is a shortest path from s to v_k.
- Idea: Instead of ordering the input, we order the output

Question: How to remove the assumption?
Relaxation: If we know the shortest paths from s to all the other vertices with length smaller than $d[t]$, it is sufficient.
Conclusion: Compute shortest paths in the ascending order of shortest path lengths

Ob … n
fro … to
v_k.

Goal: compute the shortest path from s to t

Assumption: know the shortest paths from s to all the other vertices except t

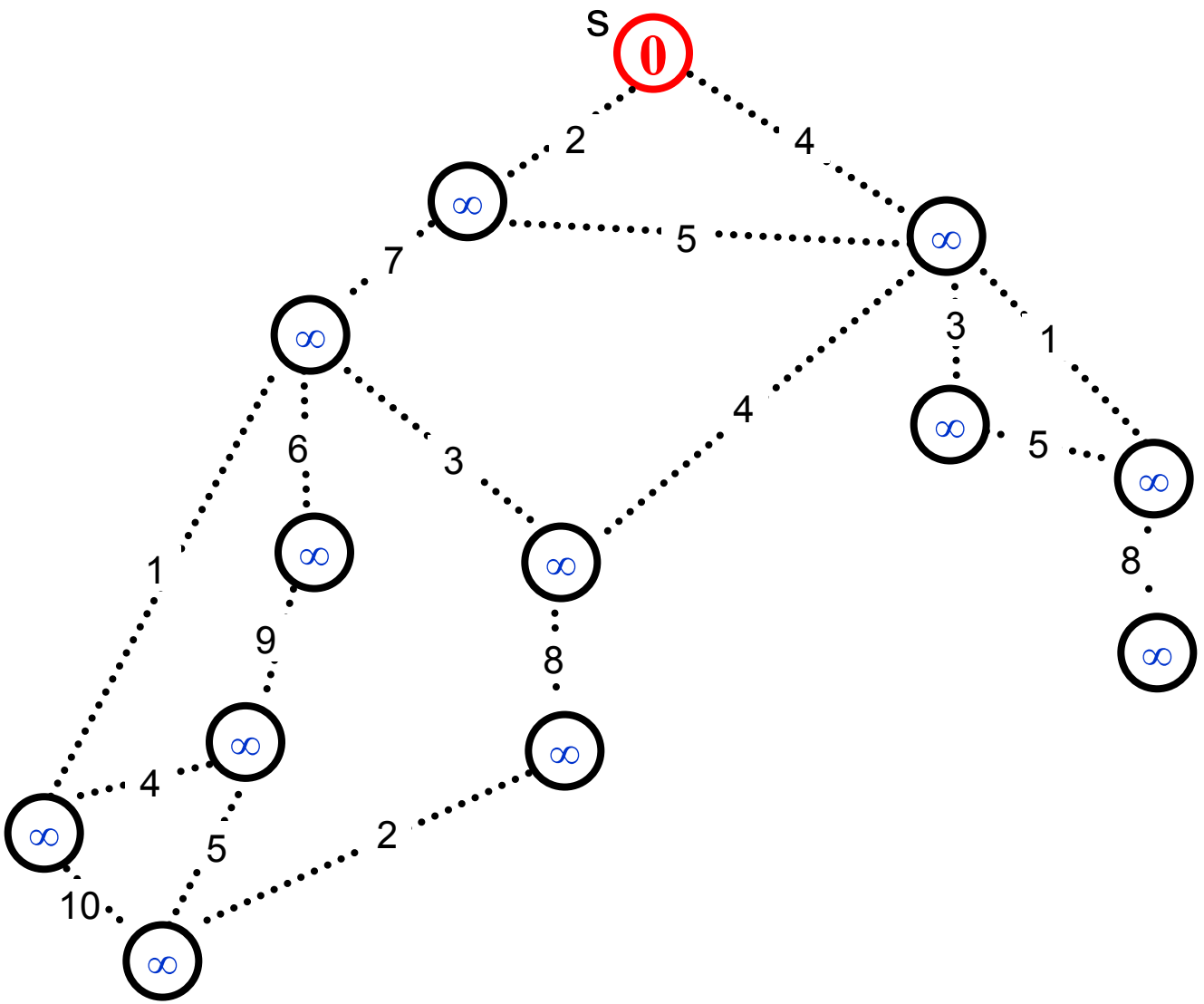
$$d[t] = \min_{v \in V \setminus \{t\}} (d[v] + c_{(v,t)})$$

($d[v]$ denotes the length of the shortest path from s to v, and $c_{(v,t)}$ denotes the weight of the edge (v, t))

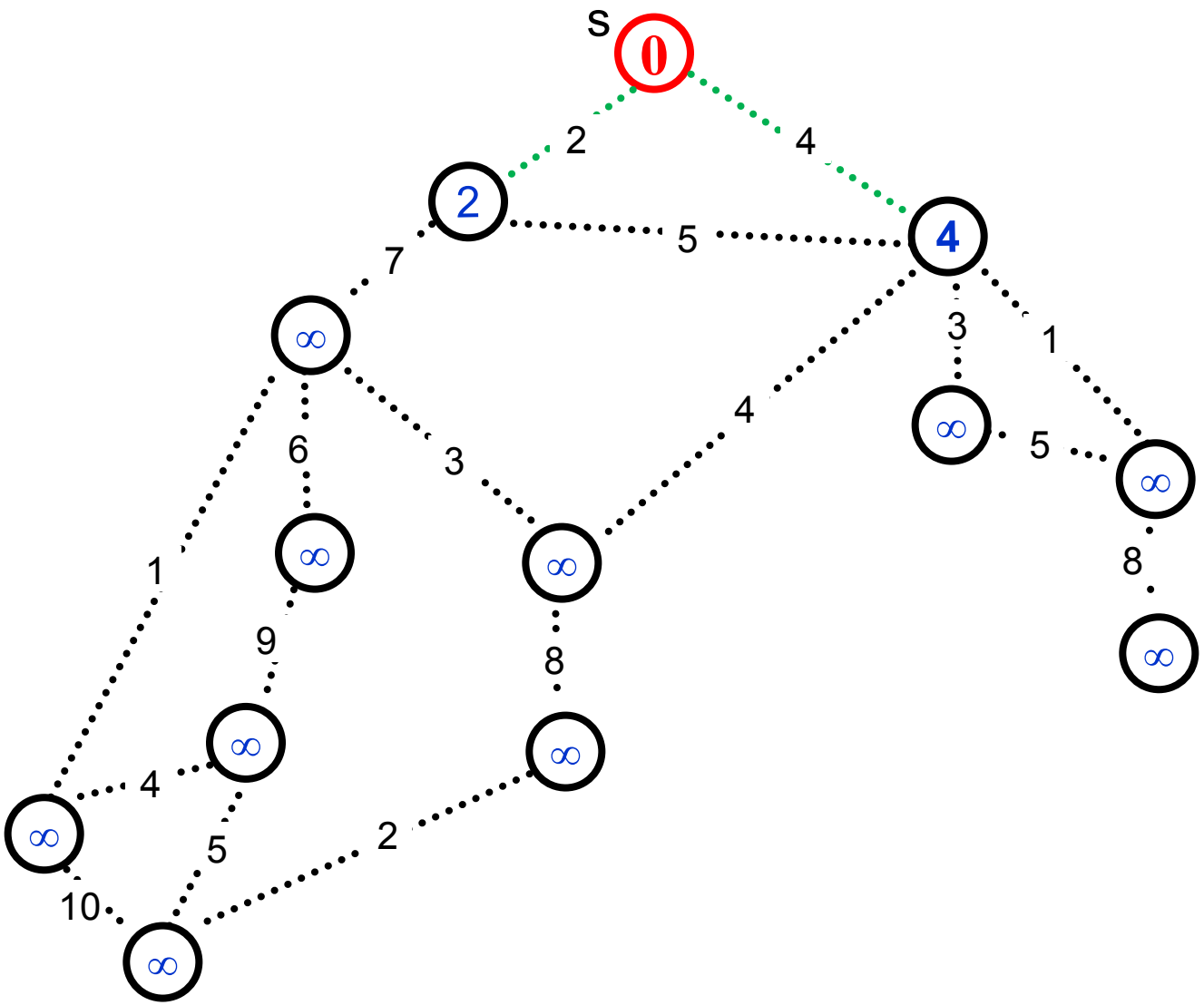
Dijkstra's Algorithm

```
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.
    }
```

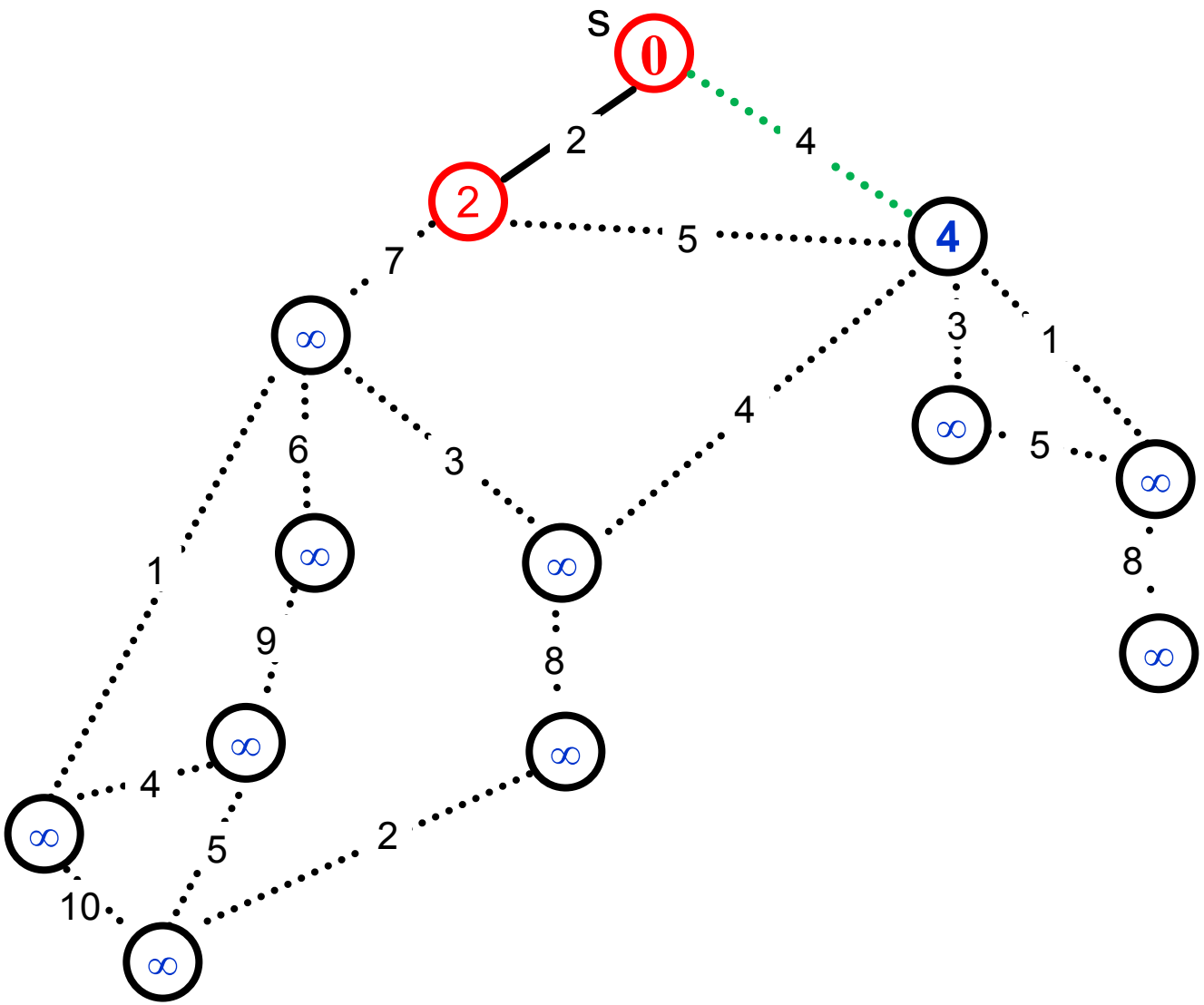
Dijkstra's Algorithm: Example



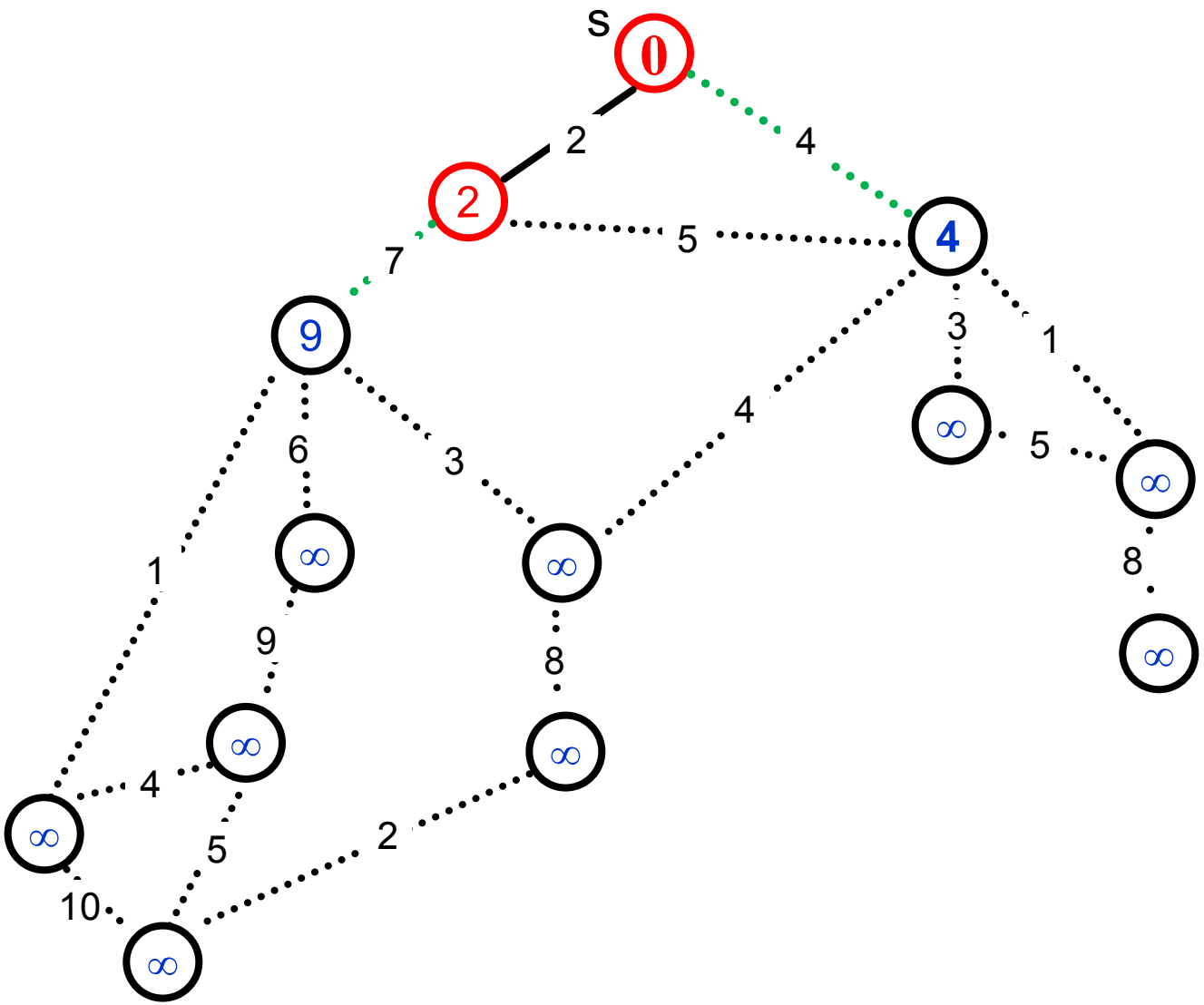
Dijkstra's Algorithm: Example



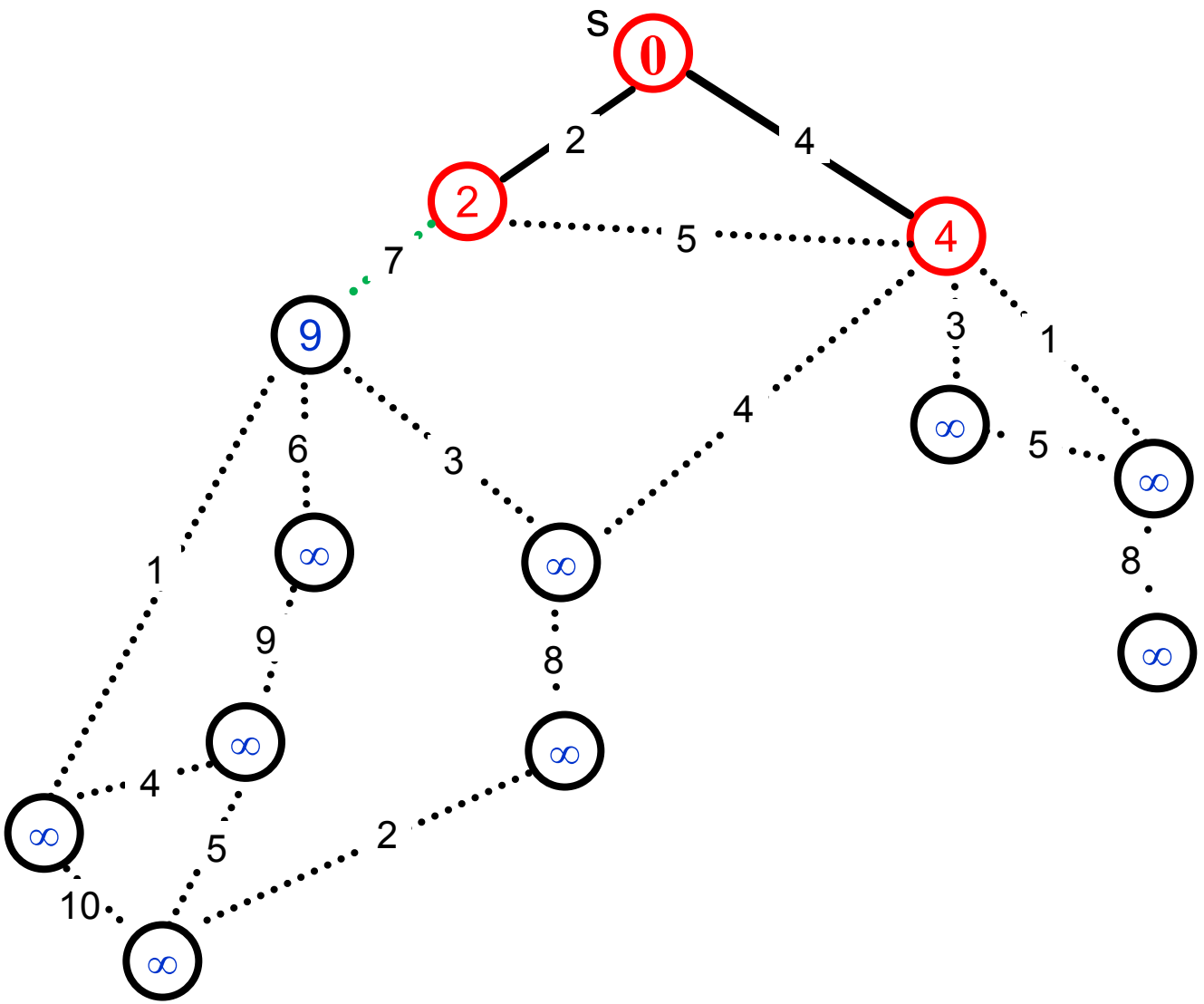
Dijkstra's Algorithm: Example



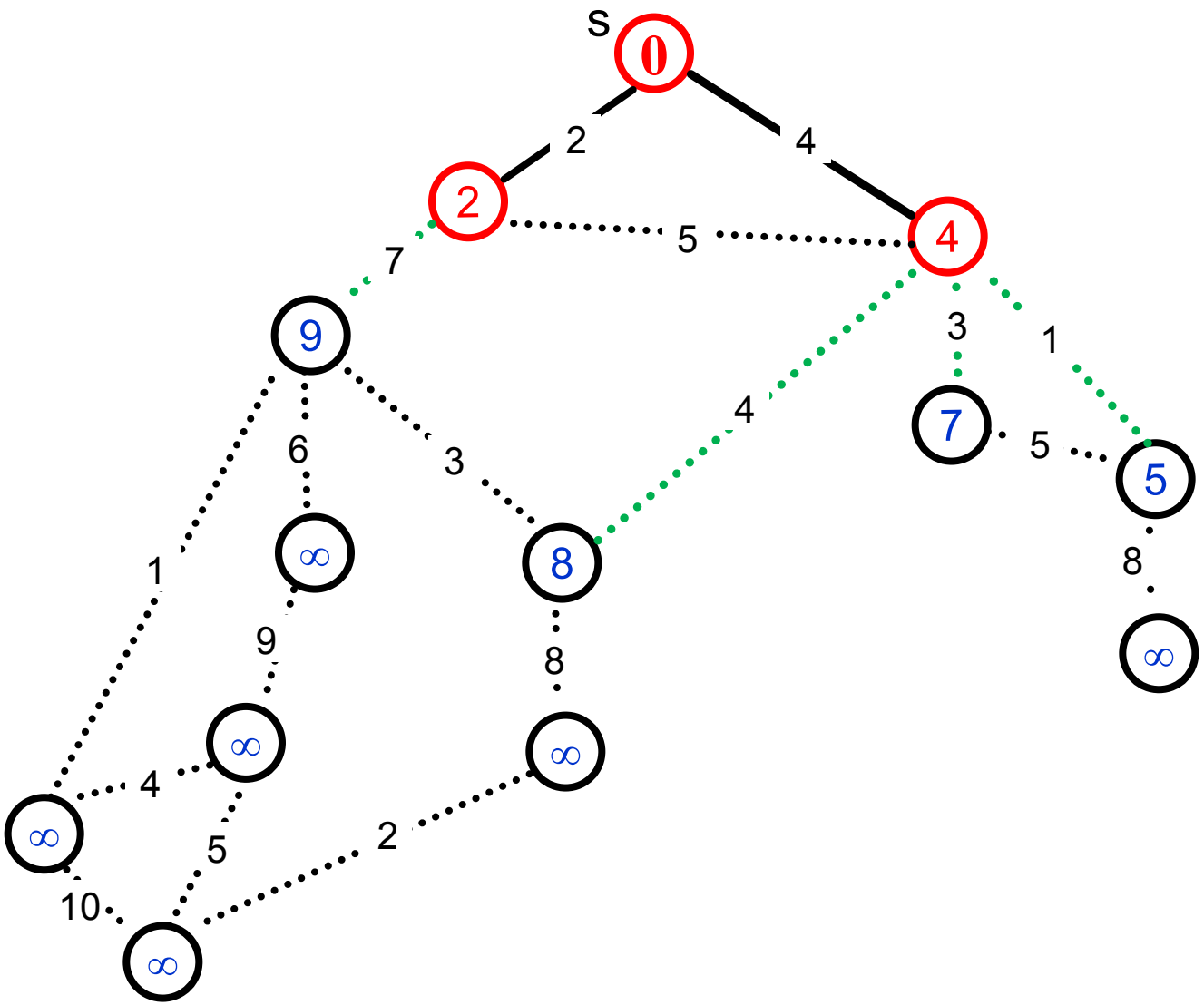
Dijkstra's Algorithm: Example



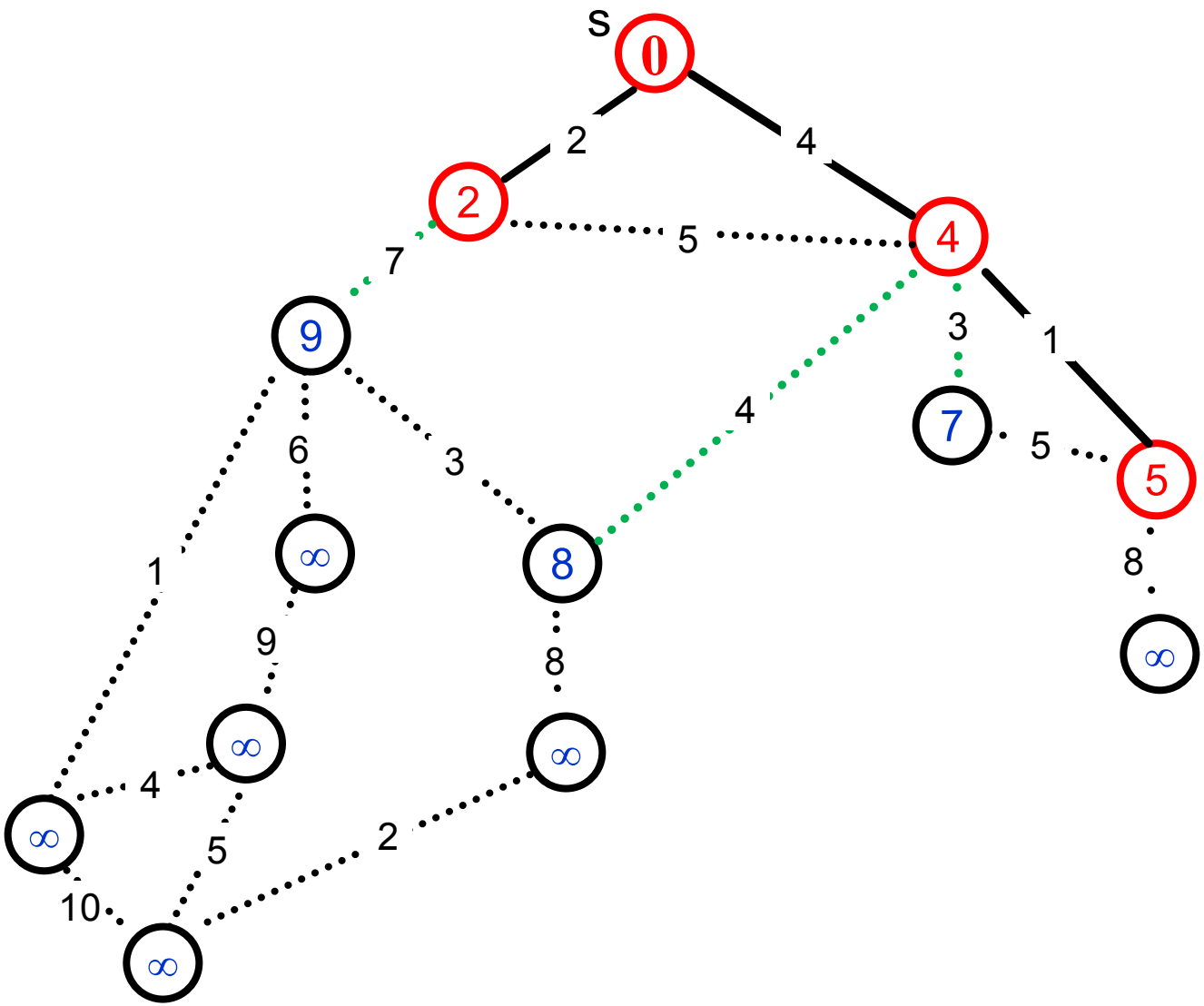
Dijkstra's Algorithm: Example



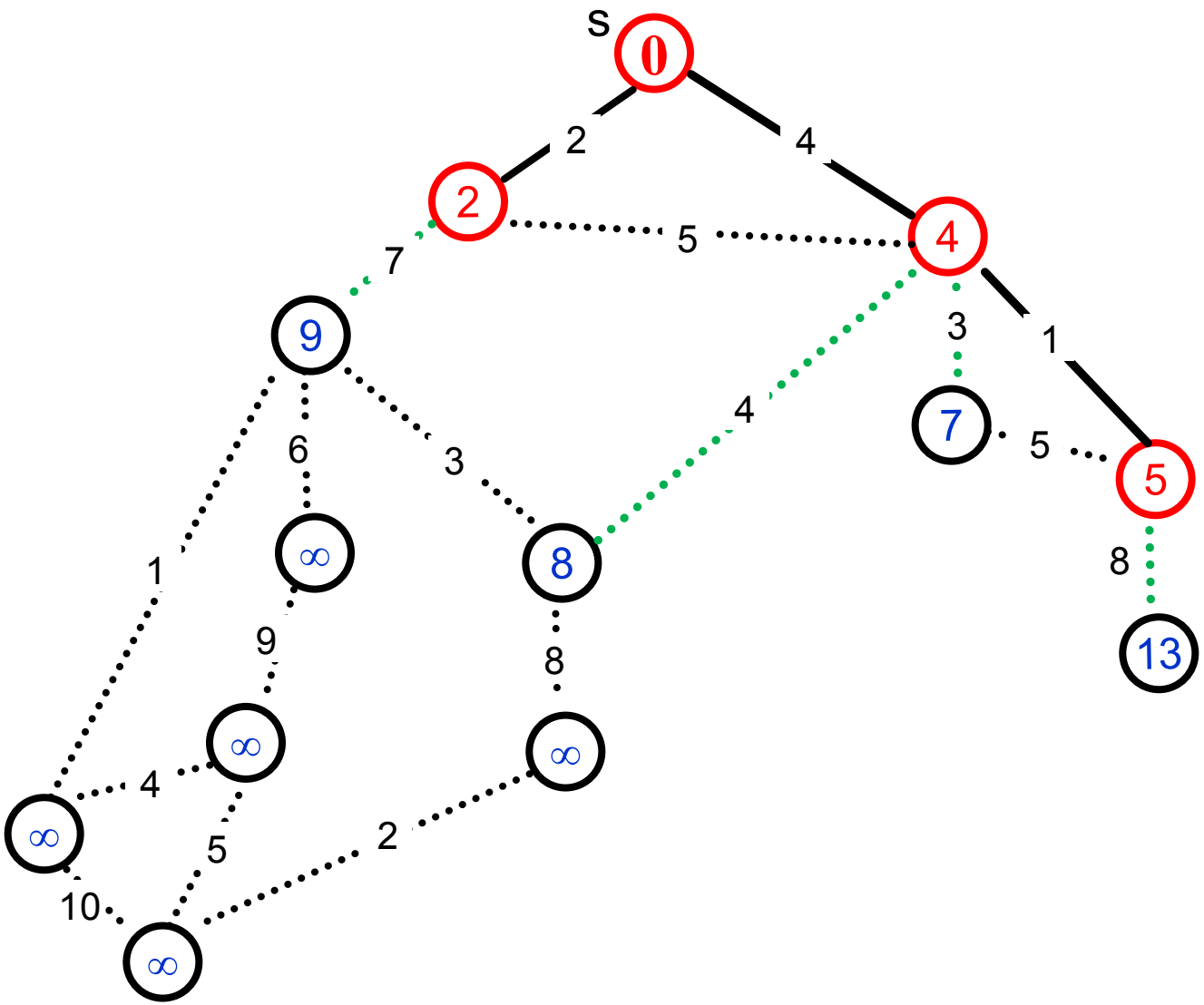
Dijkstra's Algorithm: Example



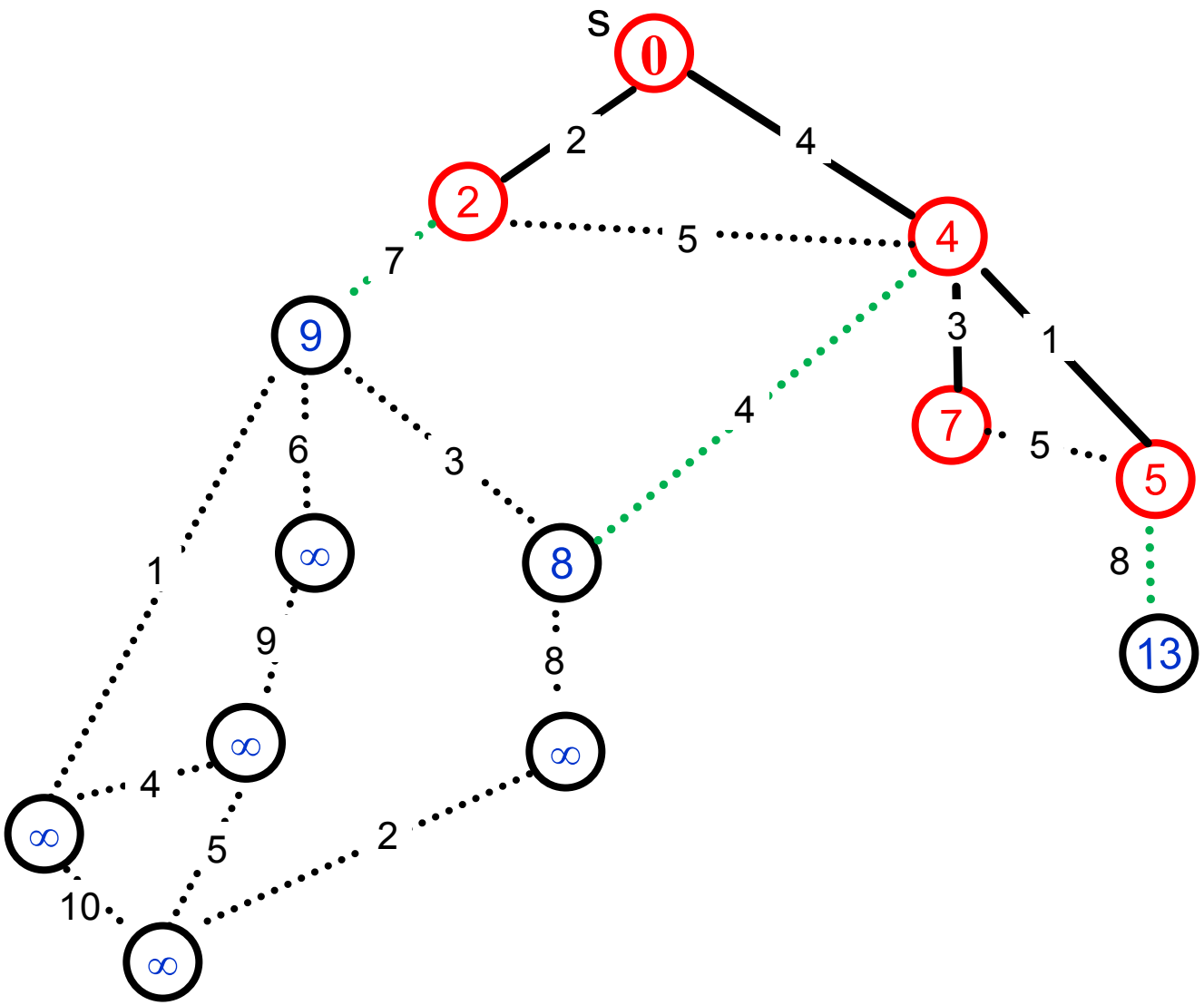
Dijkstra's Algorithm: Example



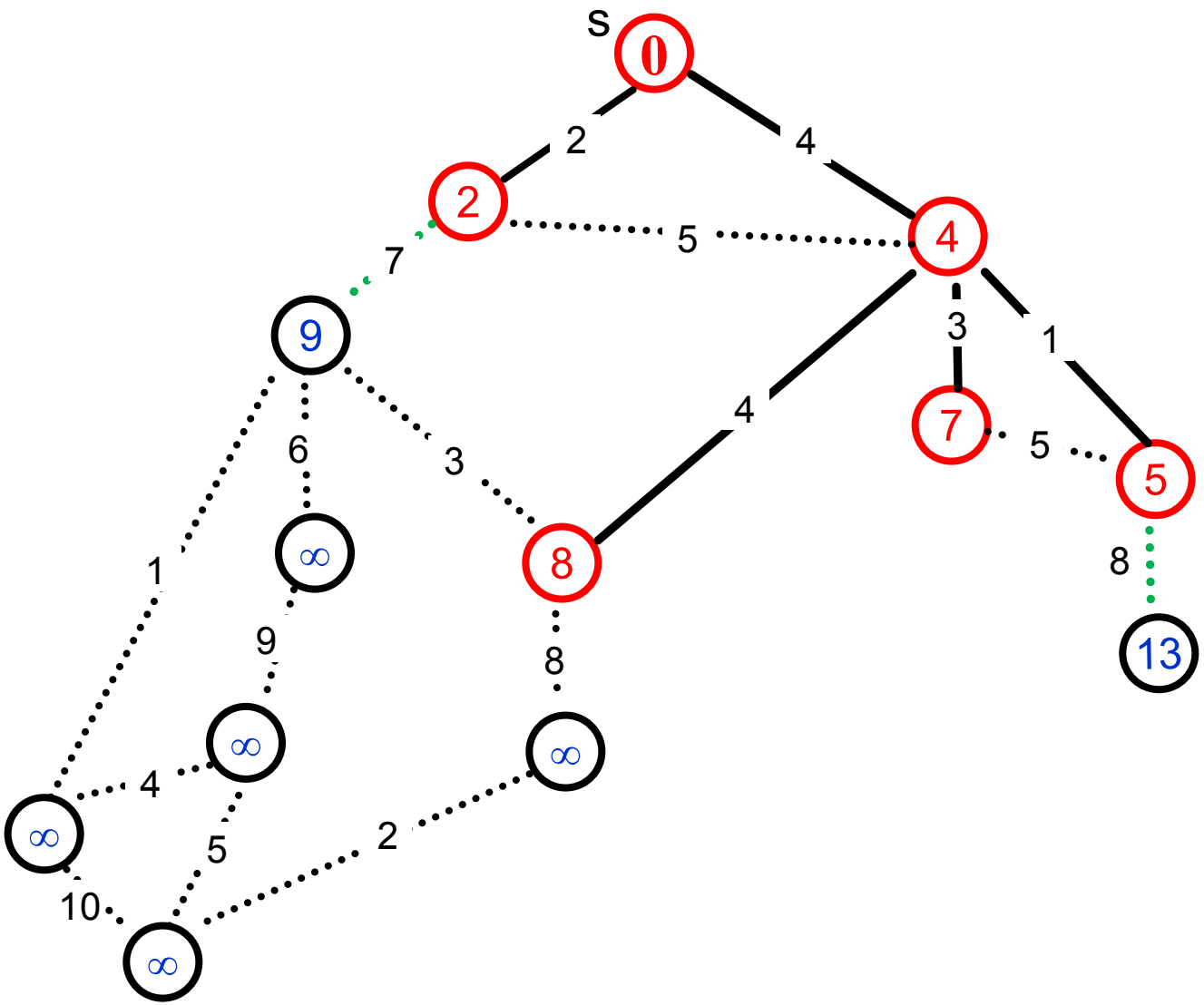
Dijkstra's Algorithm: Example



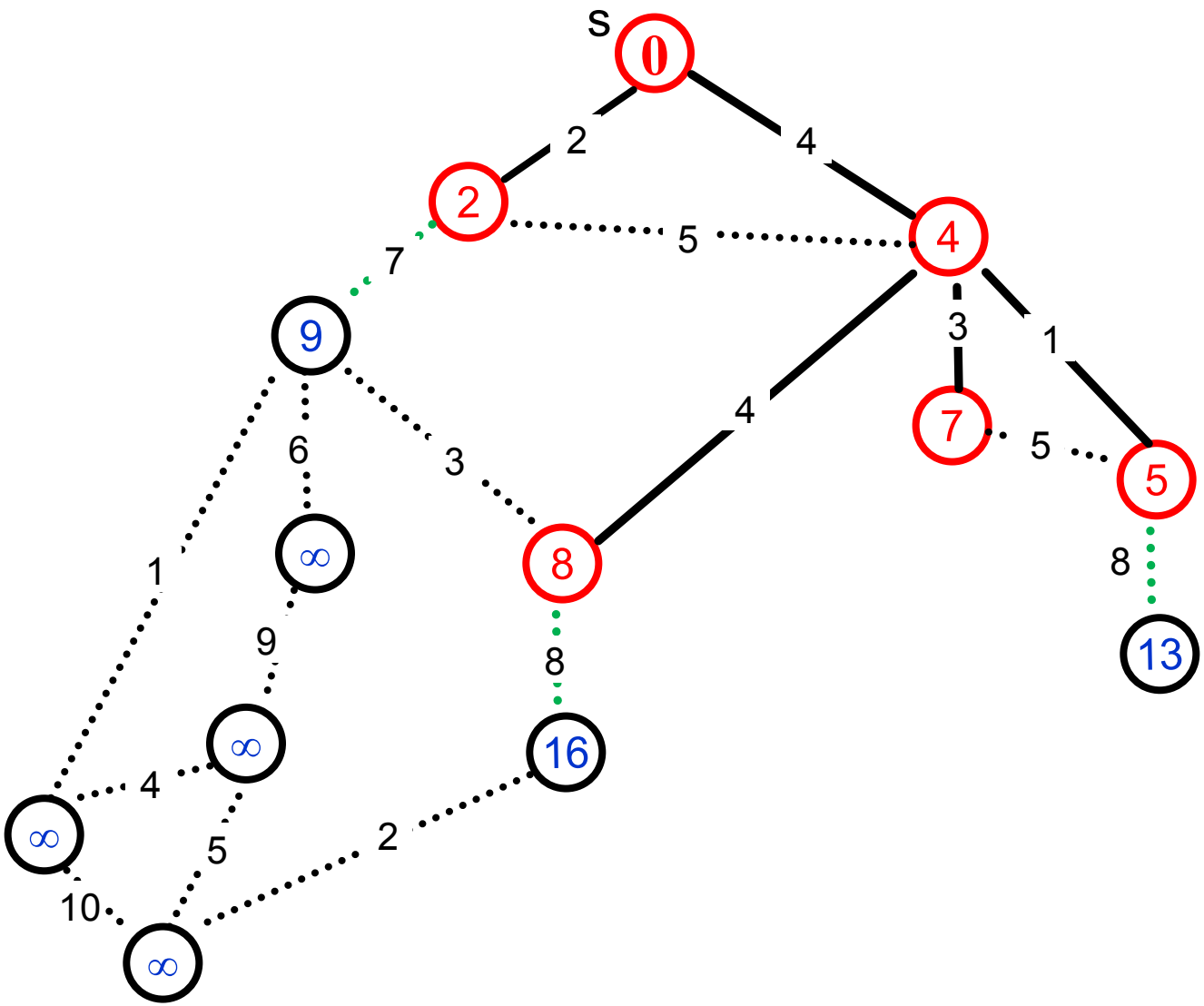
Dijkstra's Algorithm: Example



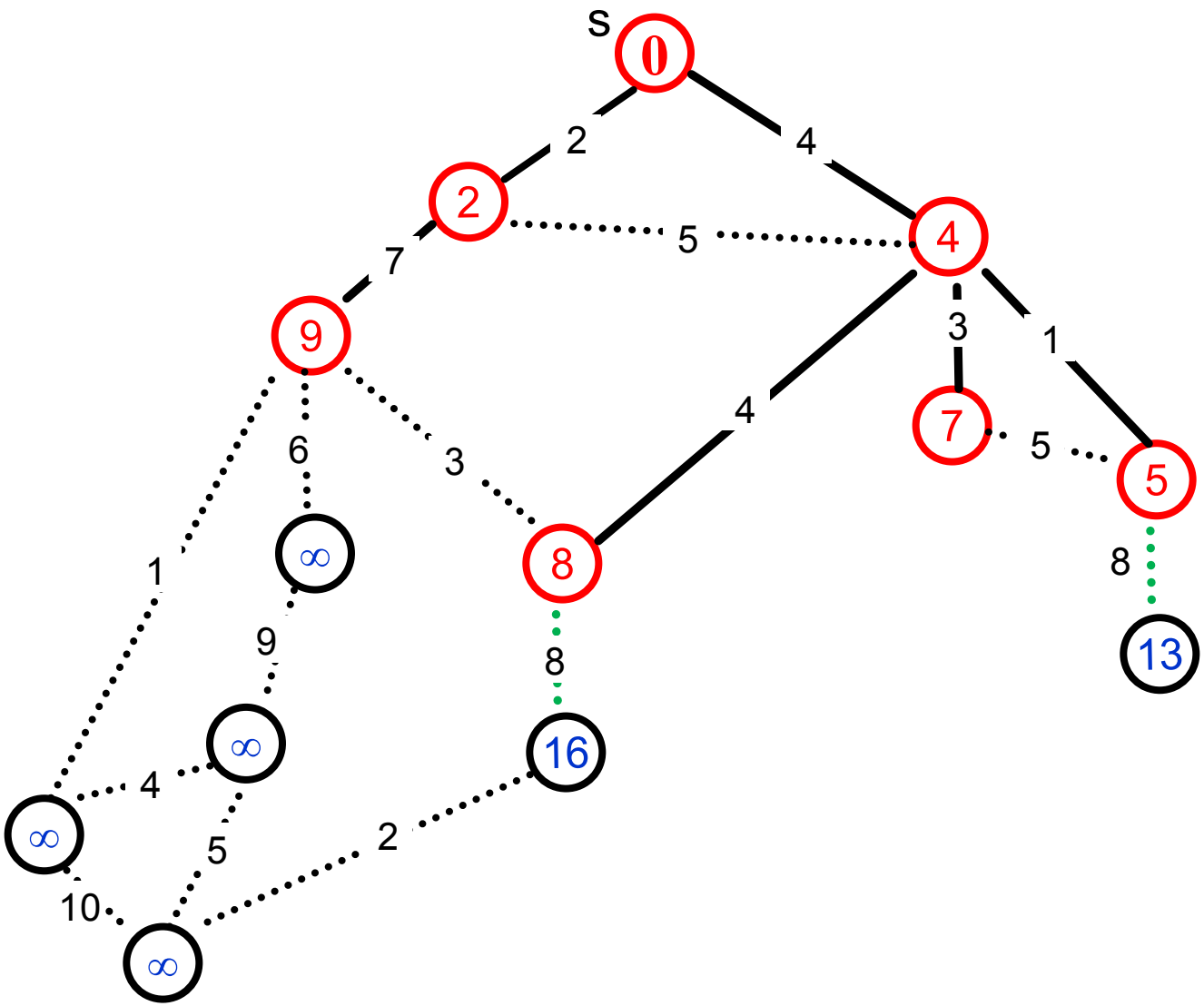
Dijkstra's Algorithm: Example



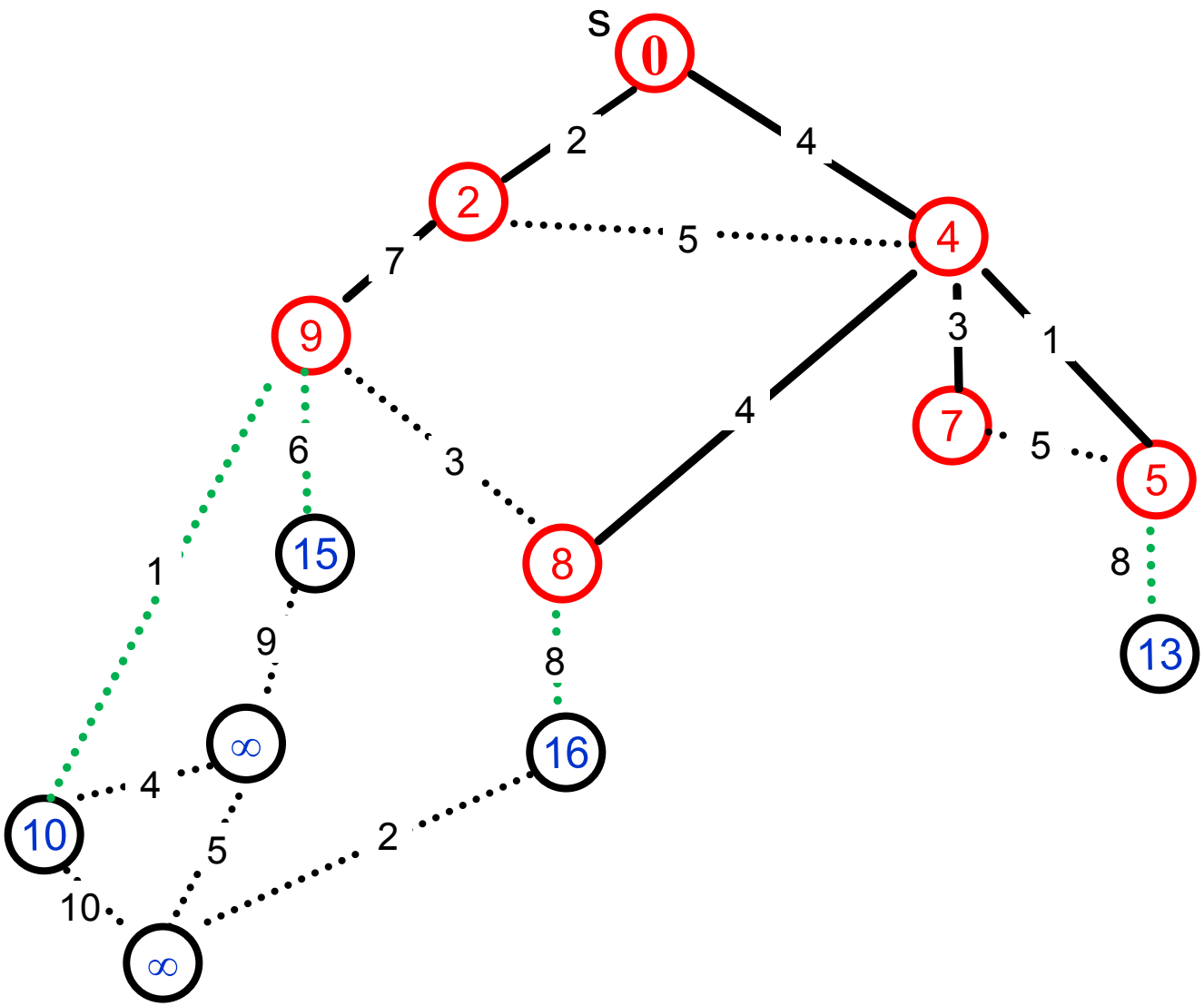
Dijkstra's Algorithm: Example



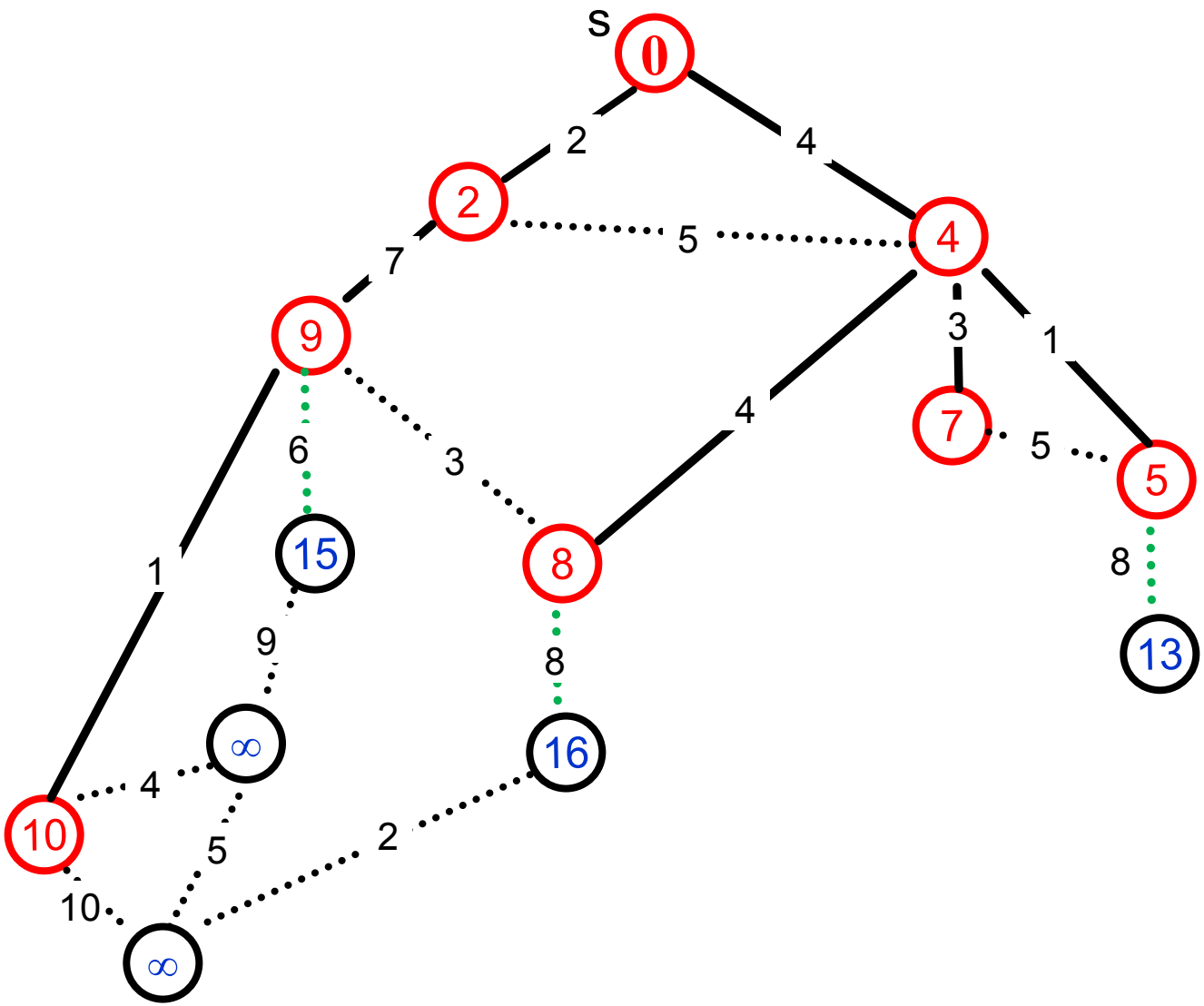
Dijkstra's Algorithm: Example



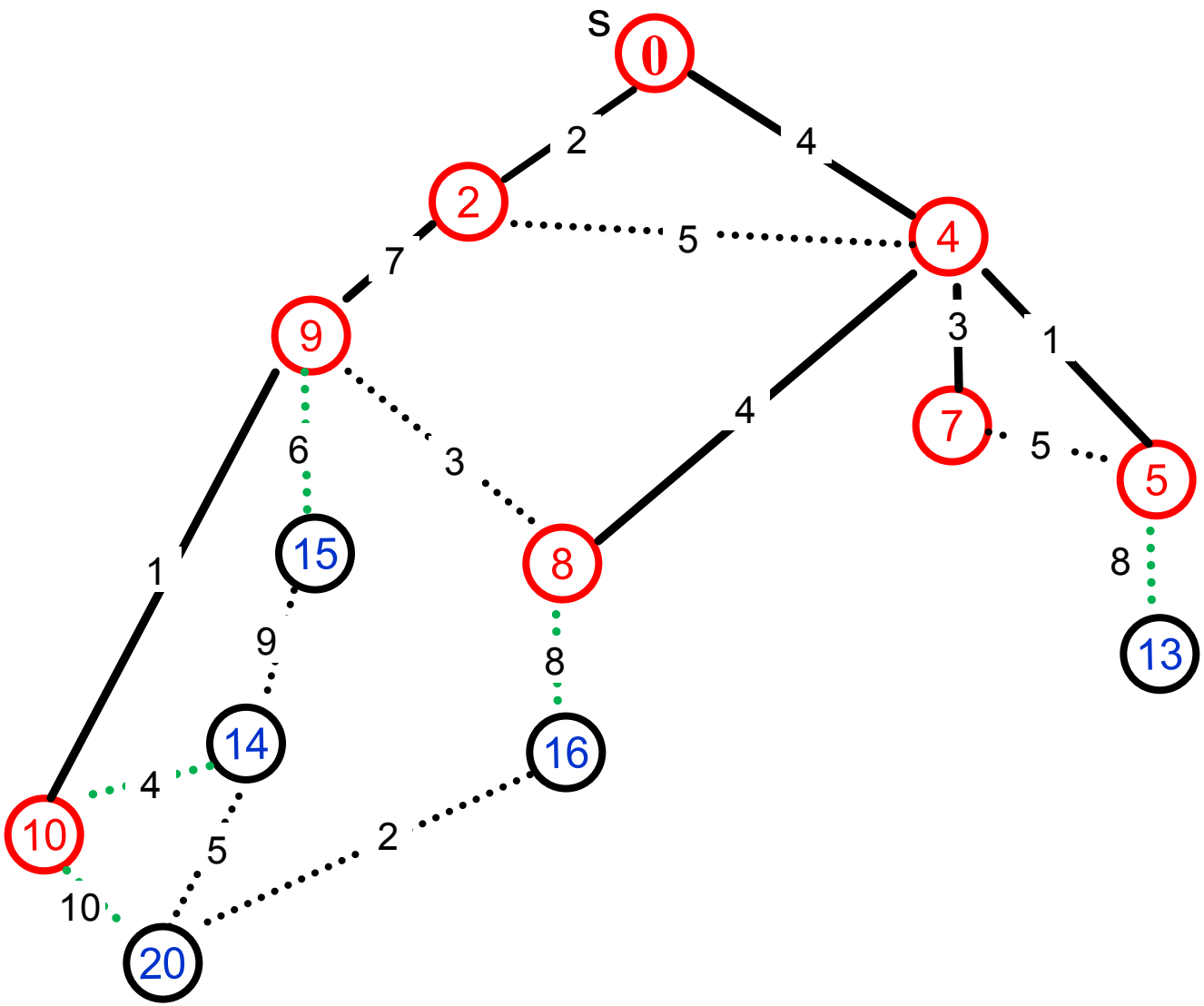
Dijkstra's Algorithm: Example



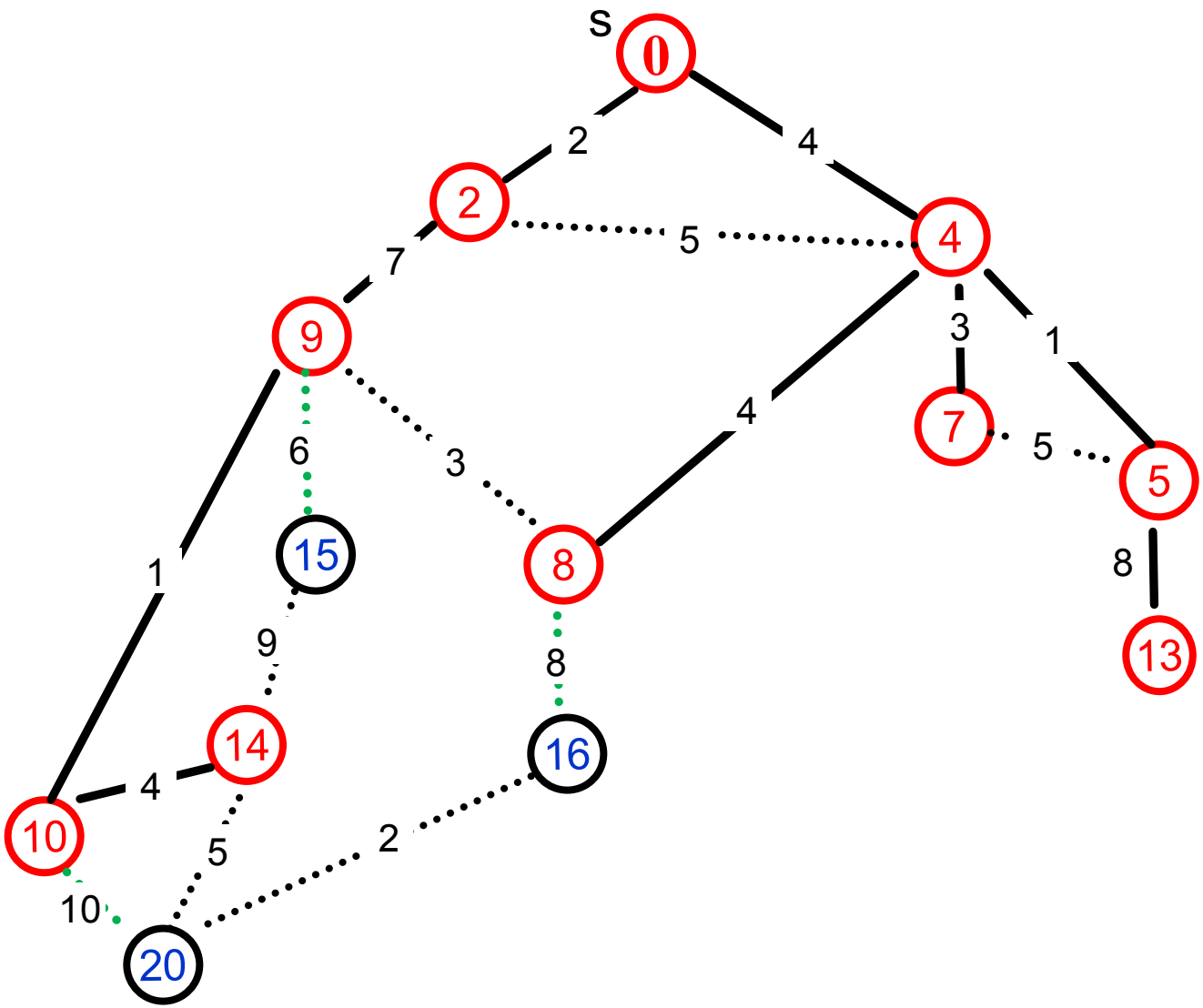
Dijkstra's Algorithm: Example



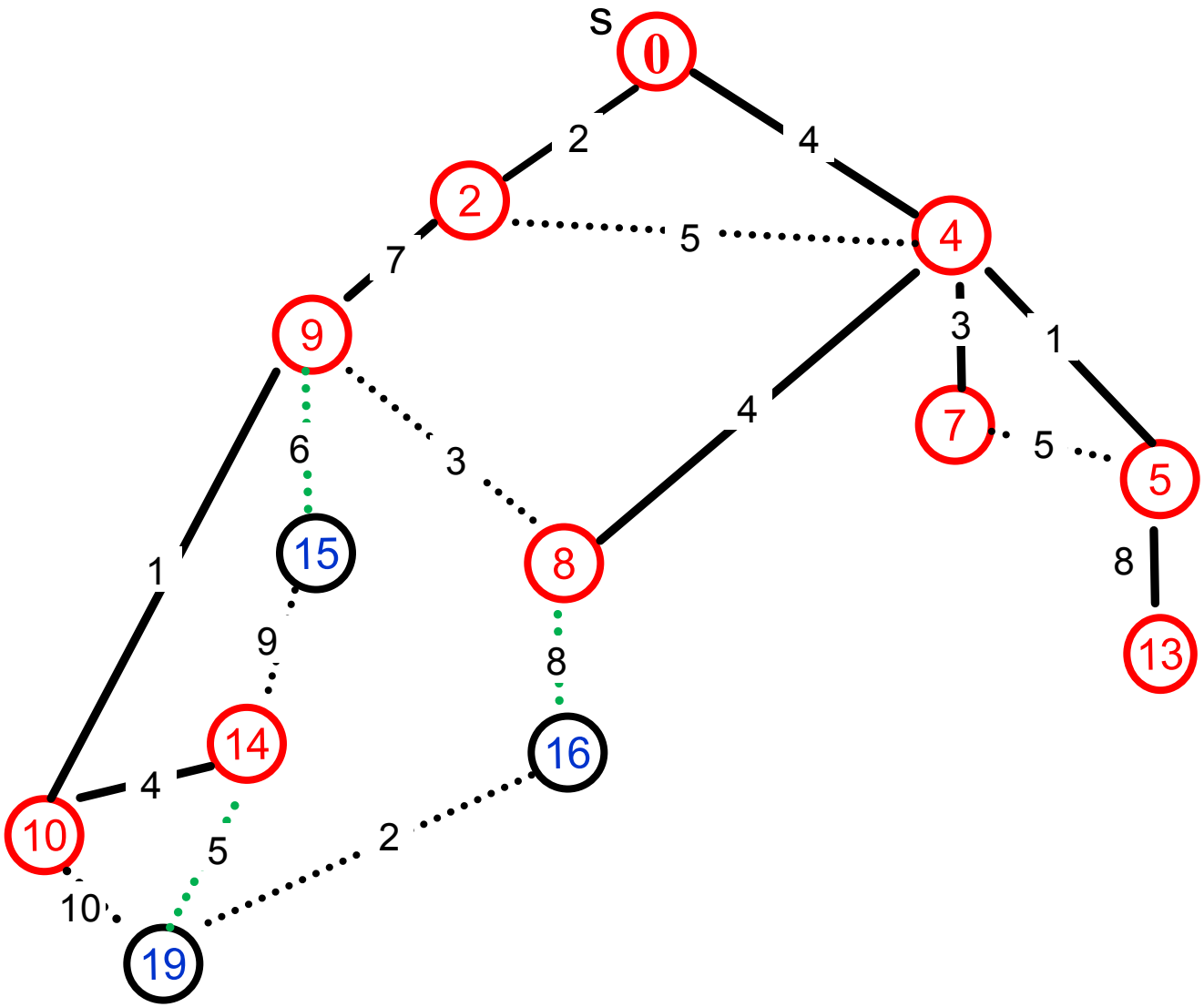
Dijkstra's Algorithm: Example



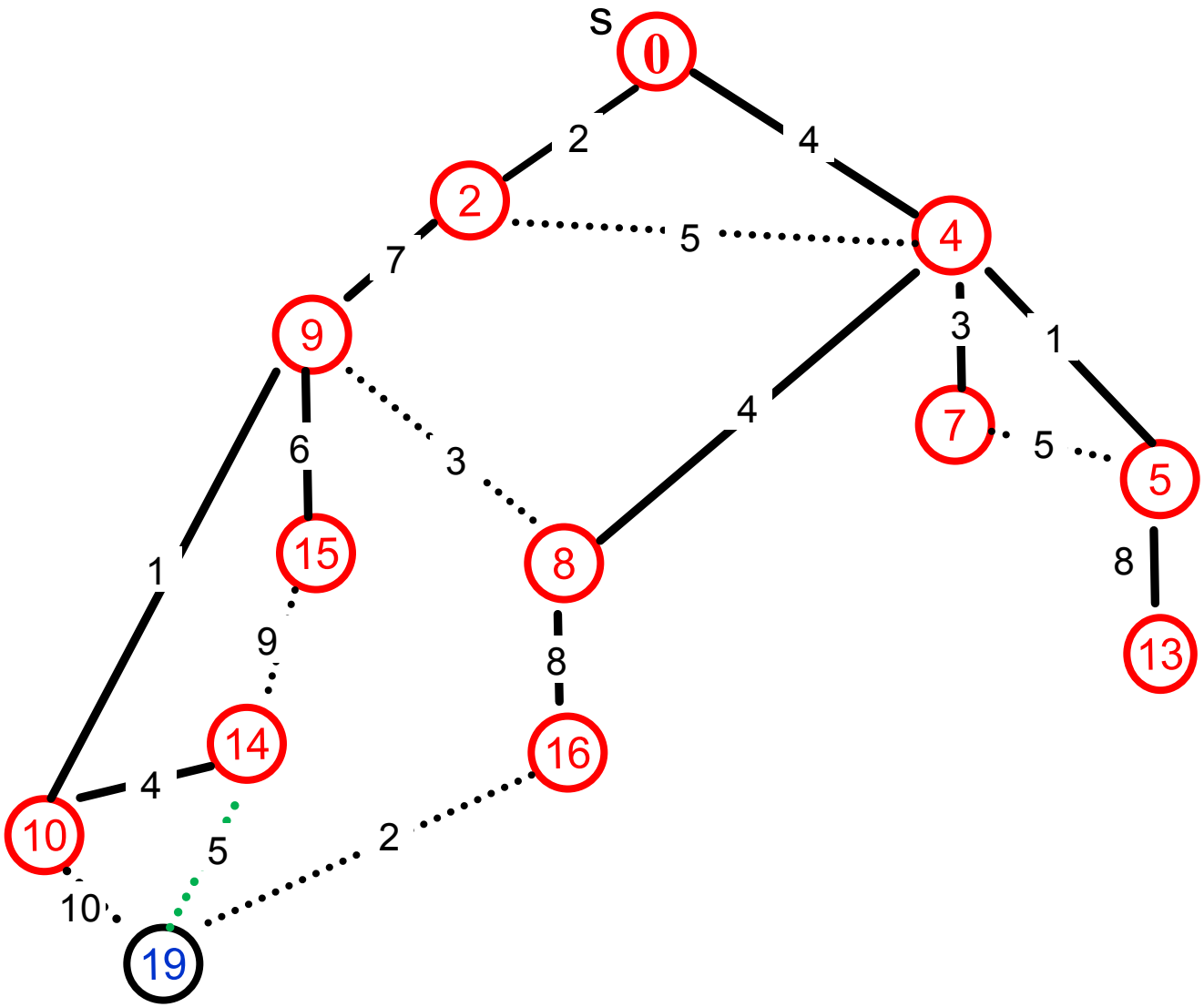
Dijkstra's Algorithm: Example



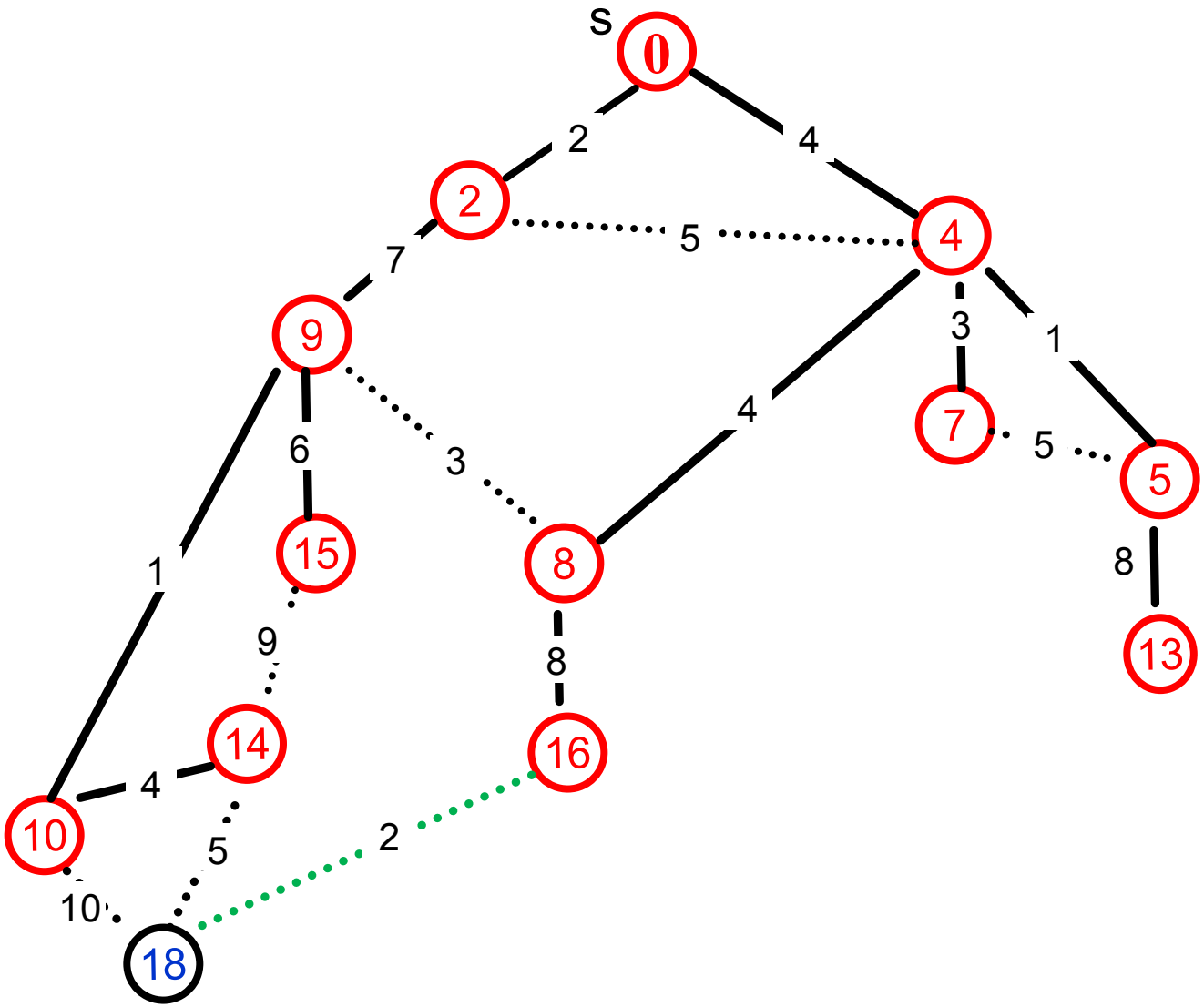
Dijkstra's Algorithm: Example



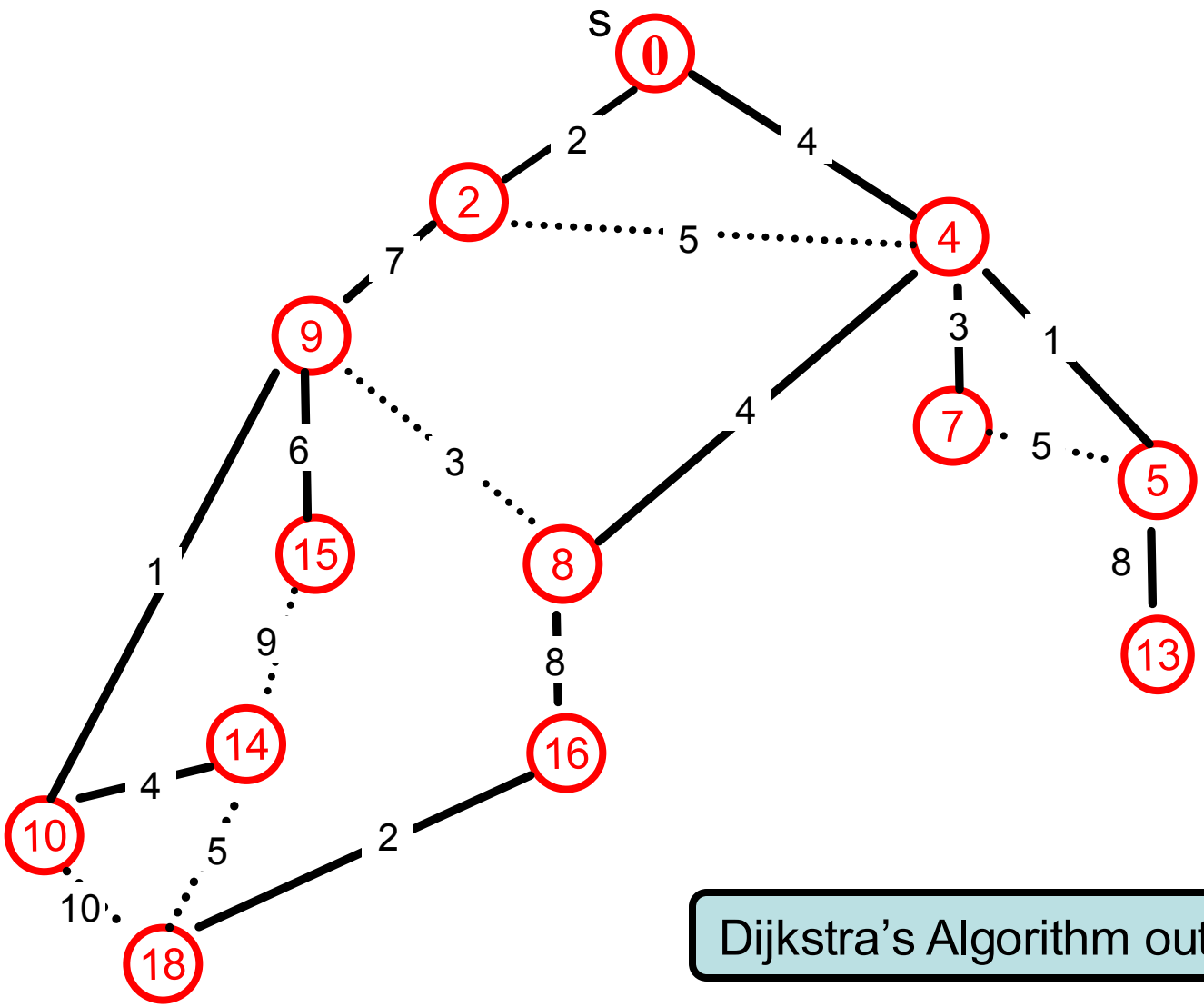
Dijkstra's Algorithm: Example



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.

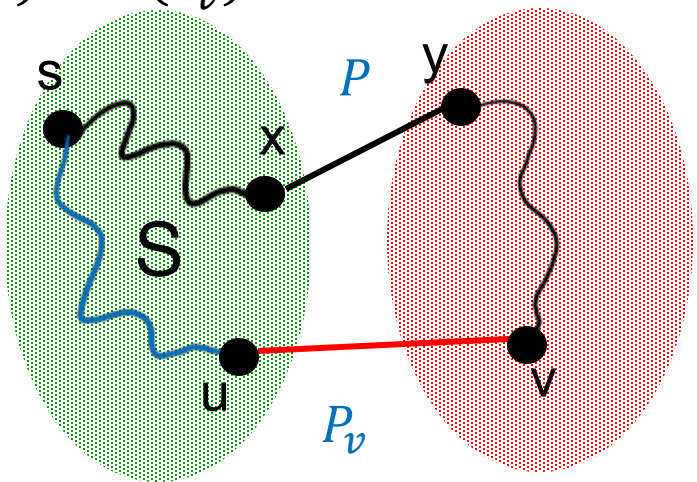
Consider the first time that P leaves S with edge (x,y).

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

P is the shorter path.

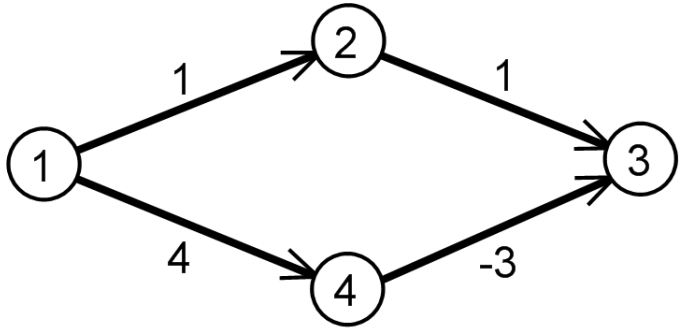
Due to the choice of v

A contradiction.



Remarks on Dijkstra's Algorithm

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



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.
        v ← 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 $O(1)$

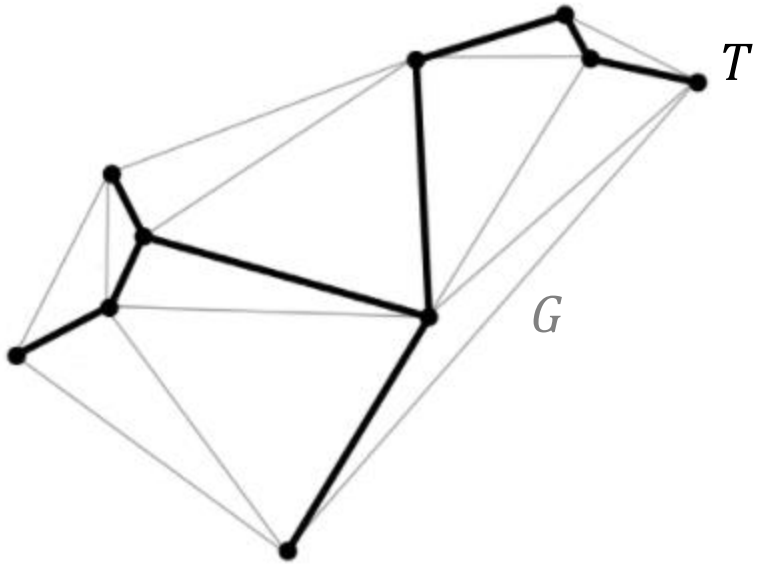
Minimum Spanning Tree

Spanning Tree

Given a connected undirected graph $G = (V, E)$.

We call T is a spanning tree of G if

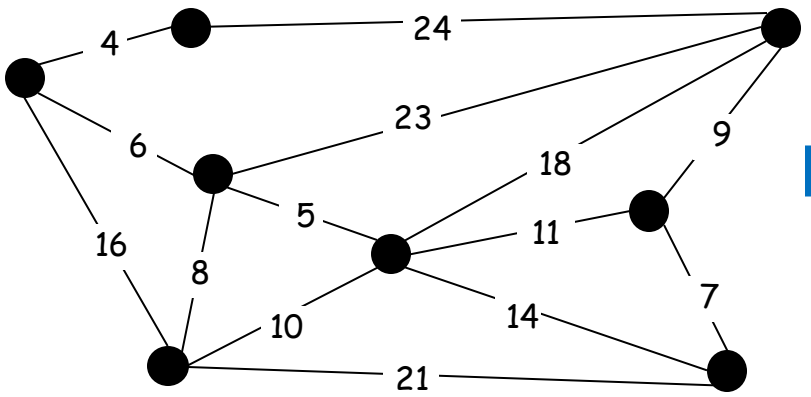
- All edges in T are from E.
- T includes all of the vertices of G.



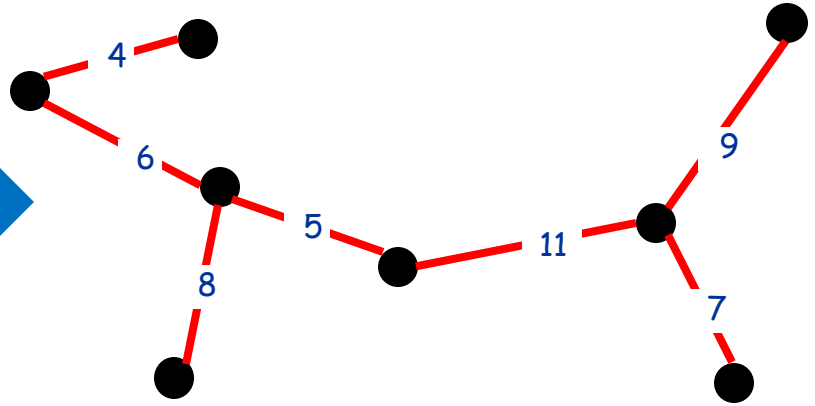
Minimum Spanning Tree (MST)

Given a connected undirected graph $G = (V, E)$ with real-valued edge weights $c_e \geq 0$.

An MST T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$

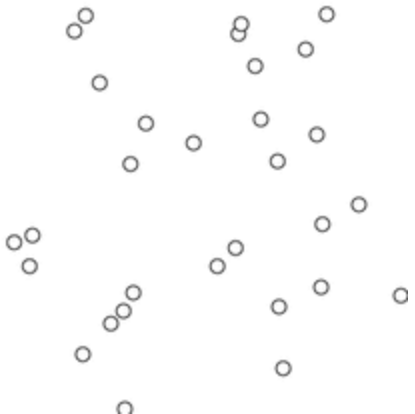


$$c(T) = \sum_{e \in T} c_e = 50$$

Kruskal's Algorithm [1956]

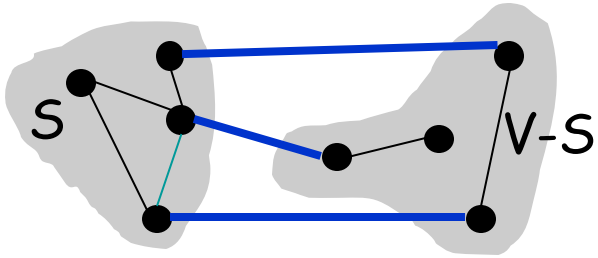
```
Kruskal(G, c) {
    Sort edges weights so that c_1 ≤ c_2 ≤ ⋯ ≤ c_m.
    T ← ∅

    foreach (u ∈ V) make a set containing singleton {u}

    for i = 1 to m
        Let (u, v) = e_i
        if (u and v are in different sets) {
            T ← T ∪ {e_i}
            merge the sets containing u and v
        }
    return T
}
```

Kruskal



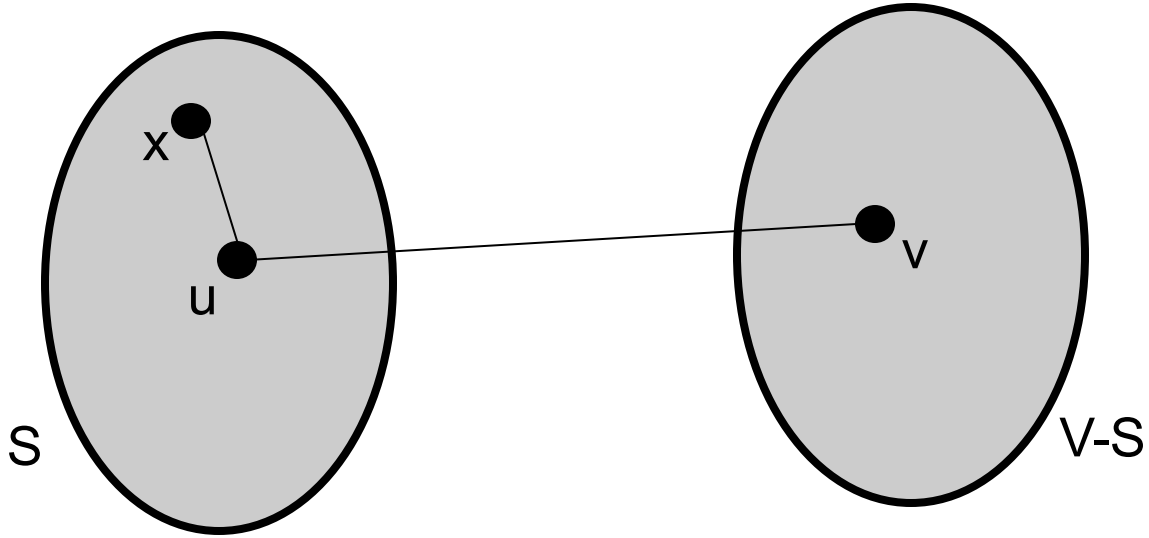
Sort edges weight.
Add edges whenever it does not create cycle.

Cuts



In a graph $G = (V, E)$, a cut is a bipartition of V into disjoint sets $S, V - S$ for some $S \subseteq V$. We denote it by $(S, V - S)$.

An edge $e = \{u, v\}$ is in the cut $(S, V - S)$ if exactly one of u, v is in S.

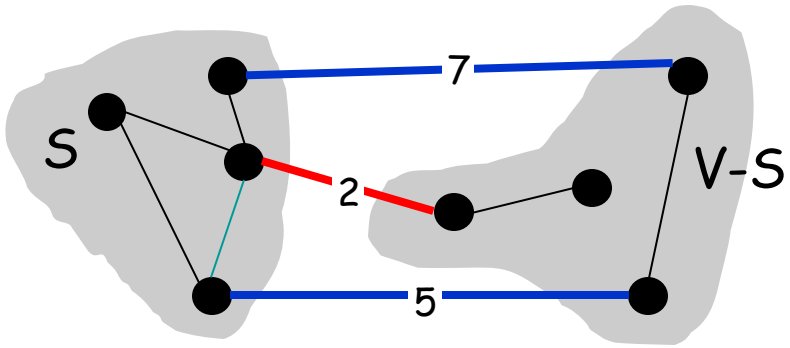


Properties of the OPT

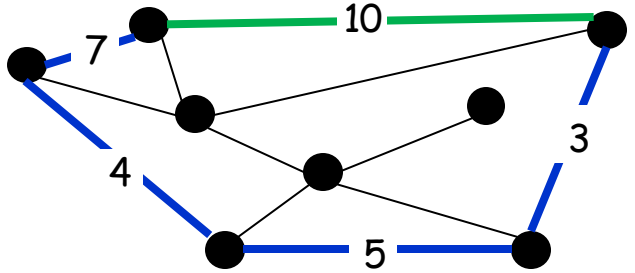
Simplifying assumption: All edge costs c_e are distinct.

Cut property: Let S be any subset of nodes (called a cut), and let e be the min cost edge with exactly one endpoint in S. Then every MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then no MST contains f.



red edge is in the MST



Green edge is not in the MST

Proof of Correctness (Kruskal)

Consider edges in ascending order of weight.

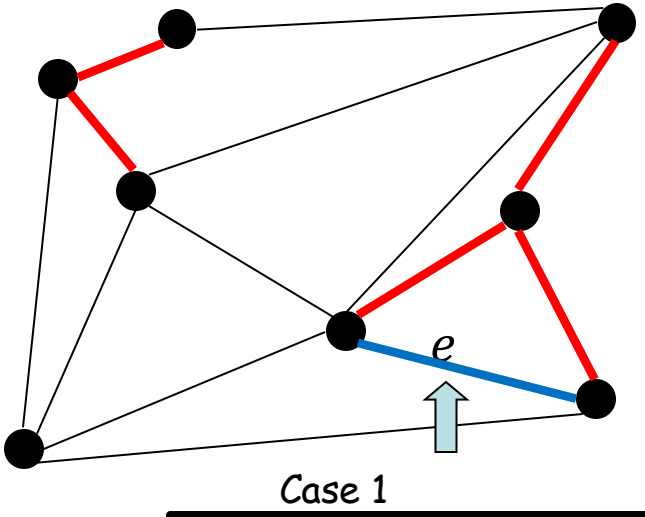
Case 1: adding e to T creates a cycle,

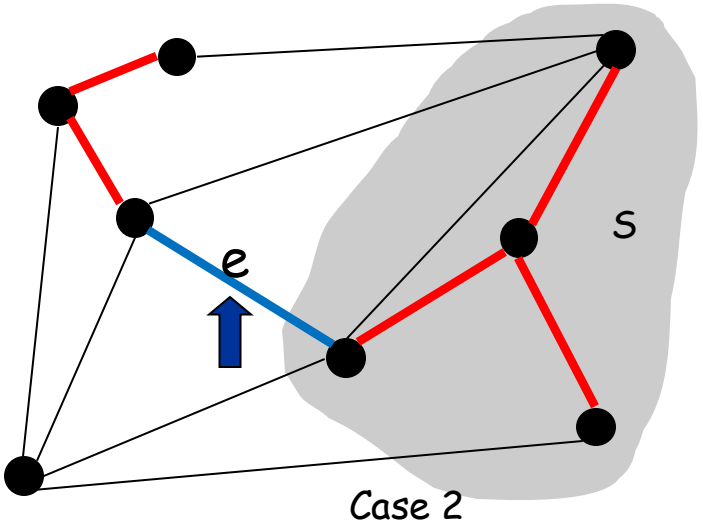
e is the maximum weight edge in that cycle.

cycle property show e is not in any minimum spanning tree.

Case 2: $e = (u, v)$ is the minimum weight edge in the cut S where S is the set of nodes in u's connected component.

So, e is in all minimum spanning tree.





This proves MST is unique if weights are distinct.

Summary

- Greedy algorithm: ‘Best’ current partial solution at each step

- Design greedy algorithm:
  - How to order your input
  - Strategy for every step

- Greedy Analysis Strategies
  - Greedy algorithm stays ahead
  - Structural
  - Exchange argument