CS 401: Computer Algorithm I

Lateness Minimization / Weighted Shortest Path

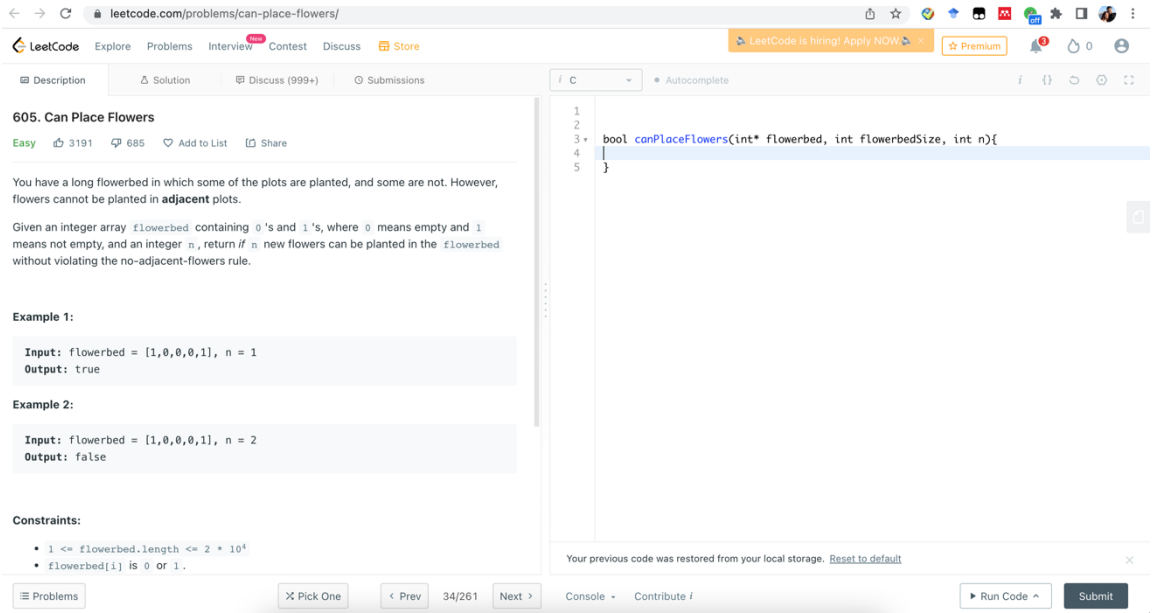
Xiaorui Sun

Homework 2

Homework 2 is out (due at 11:59pm March 4)

Programming homework on Leetcode

- Register a Leetcode account (free), and programming on Leetcode
- You can use any programming language
- Submit your source code to gradescope
- Score for each problem is proportional to the test cases on Leetcode you can pass (if you can pass all, you get full score on the problem)

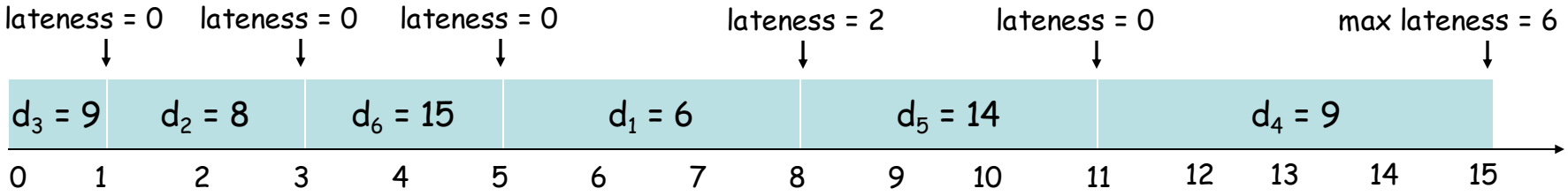


Lateness Minimization
Technique: Exchange Argument

Scheduling to Minimizing Lateness

- Instead of start and finish times, job i has
  - Time Requirement t_i which must be scheduled in a contiguous block
  - Deadline d_i by which time the job would like to be finished
- Jobs are scheduled into time intervals $[s_i, f_i]$ s.t. $t_i = f_i - s_i$.
- Lateness for job i is
  - If $d_i < f_i$ then job i is late by $L_i = f_i - d_i$ otherwise its lateness $L_i = 0$
- **Goal:** Find a schedule that minimize the Maximum lateness $L = \max_i L_i$

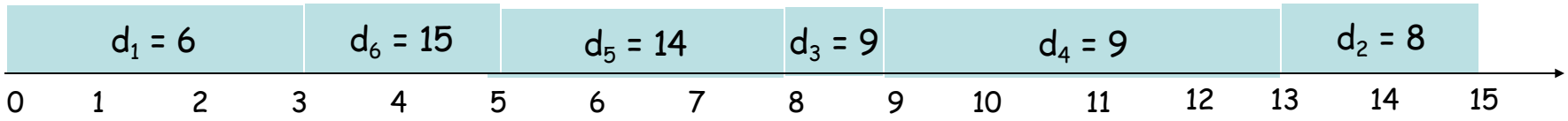
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Question

What is the maximum lateness of the following scheduling?

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Answer: 7

Minimizing Lateness: Greedy Algorithms

Greedy template. Run jobs in some order.

- [Shortest processing time first]
  Consider jobs in ascending order of processing time t_j.

- [Smallest slack]
  Consider jobs in ascending order of slack $d_j - t_j$.

- [Earliest deadline first]
  Consider jobs in ascending order of deadline d_j.

Minimizing Lateness: Greedy Algorithms

Greedy template. Run jobs in some order.

- [Shortest processing time first]

  Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack]

  Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

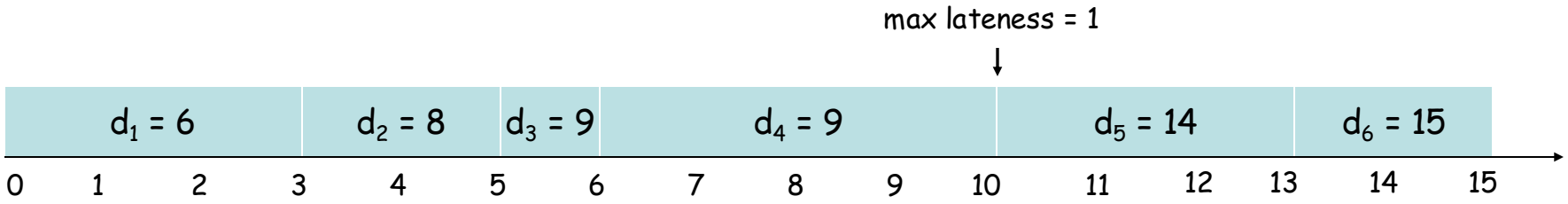
- [Earliest deadline first]

  Consider jobs in ascending order of deadline d_j.

Greedy Algorithm: Earliest Deadline First

Sort deadlines in increasing order $(\boldsymbol{d_1 \leq d_2 \leq \cdots \leq d_n})$
$\boldsymbol{f \leftarrow 0}$
for $\boldsymbol{i \leftarrow 1}$ to $\boldsymbol{n}$ to
    $\boldsymbol{s_i \leftarrow f}$
    $\boldsymbol{f_i \leftarrow s_i + t_i}$
    $\boldsymbol{f \leftarrow f_i}$
end for

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



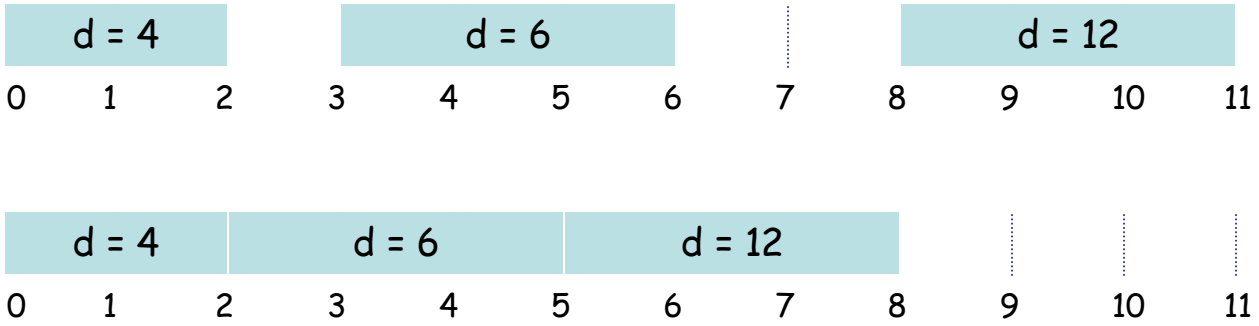
Proof for Greedy Algorithm: Exchange Argument

- We will show that if there is another schedule $\boldsymbol{O}$ (think optimal schedule) then we can gradually change $\boldsymbol{O}$ so that
  - at each step the maximum lateness in $\boldsymbol{O}$ never gets worse
  - it eventually becomes the same cost as $\boldsymbol{A}$

Minimizing Lateness: No Idle Time

Observation.

- There exists an optimal schedule with no idle time.



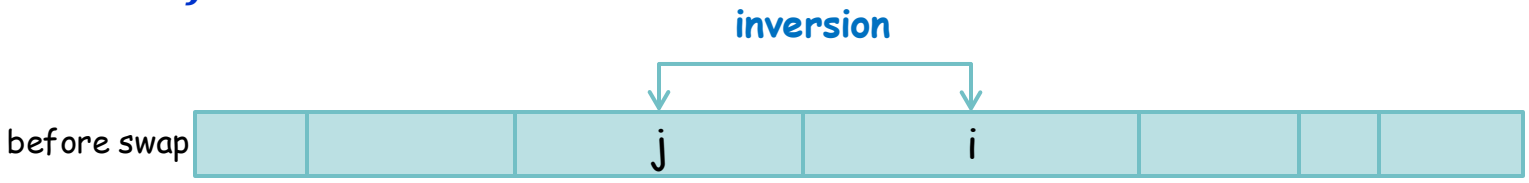
Observation.

- The greedy schedule has no idle time.

Minimizing Lateness: Inversions

Definition

- An inversion in schedule S is a pair of jobs i and j such that $d_i < d_j$ but j scheduled before i.



Observation

- Greedy schedule has no inversions.

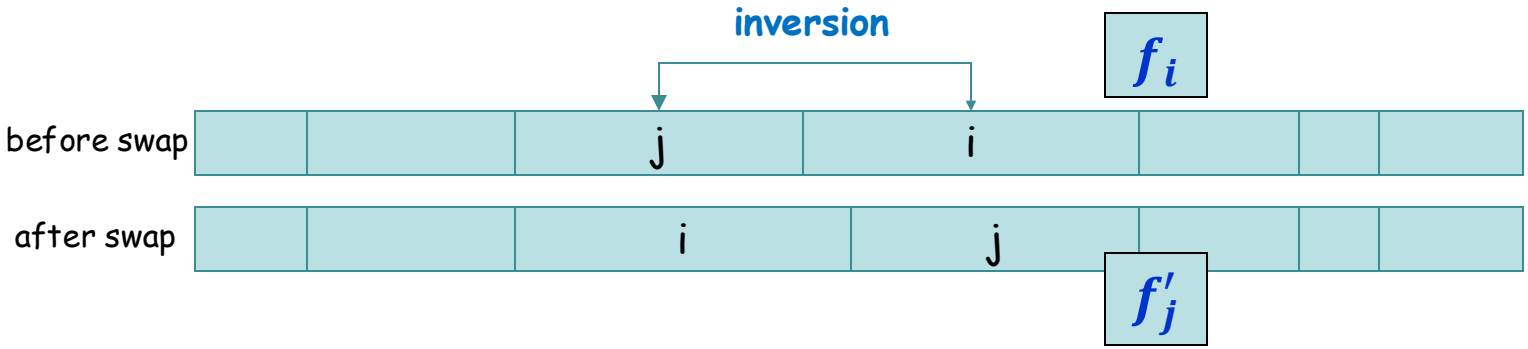
Observation

- If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled **consecutively**.
  - Why? If no inversion, then $d_i \leq d_{i+1}$ for all i .

Minimizing Lateness: Inversions

Definition

- An inversion in schedule S is a pair of jobs i and j such that $d_i < d_j$ but j scheduled before i.

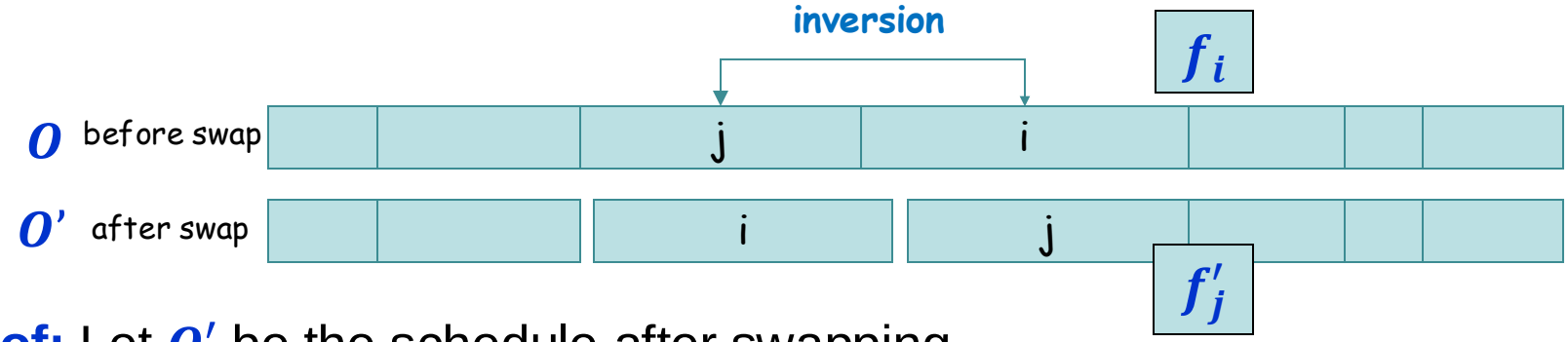


Claim

- Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Minimizing Lateness: Inversions

Lemma: Swapping two adjacent, inverted jobs does not increase the maximum lateness.



Proof: Let O' be the schedule after swapping.

- All other jobs $k \neq i, j$ have $L_k' = L_k$
- Lateness $L_i' \leq L_i$ since i is scheduled earlier in O' than in O
- Jobs i and j together occupy the same total time slot in both schedules
  - $f_j' = f_i$ so $L_j' = f_j' - d_j = f_i - d_j < f_i - d_i = L_i$
- Maximum lateness has not increased!

Optimal schedules and inversions

Did we finish the proof for greedy?

Claim: There is an optimal schedule with no idle time and no inversions

Proof:

- By previous argument there is an optimal schedule $\boldsymbol{O}$ with no idle time
- If $\boldsymbol{O}$ has an inversion then it has a **consecutive** pair of jobs in its schedule that are inverted and can be swapped without increasing lateness
- Eventually these swaps will produce an optimal schedule with no inversions
  - Each swap decreases the number of inversions by $\mathbf{1}$
  - There are at most $\boldsymbol{n(n-1)/2}$ inversions. (we only care that this is finite.)

Idleness and Inversions are the only issue

Claim: All schedules with no inversions and no idle time have the same maximum lateness

Proof:

- Schedules can differ only in how they order jobs with equal deadlines
- Consider all jobs having some common deadline d
- Maximum lateness of these jobs is based only on the finish time of the last of these jobs but the set of these jobs occupies the same time segment in both schedules
  - Last of these jobs finishes at the same time in any such schedule.

Earliest Deadline First is optimal

We know that

- There is an optimal schedule with no idle time or inversions
- All schedules with no idle time or inversions have the same maximum lateness
- EDF produces a schedule with no idle time or inversions

Therefore

- EDF produces an optimal schedule

Life Wisdom:

- Finish your jobs according to deadline!
- ☹ Unfortunately, we don’t see all jobs when born.

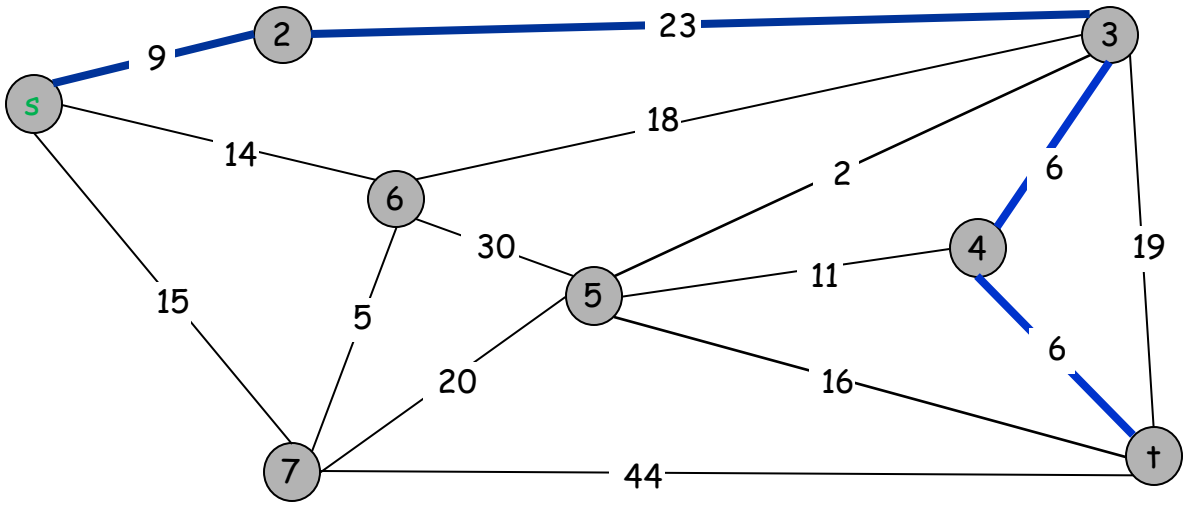
Single Source Shortest Path

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

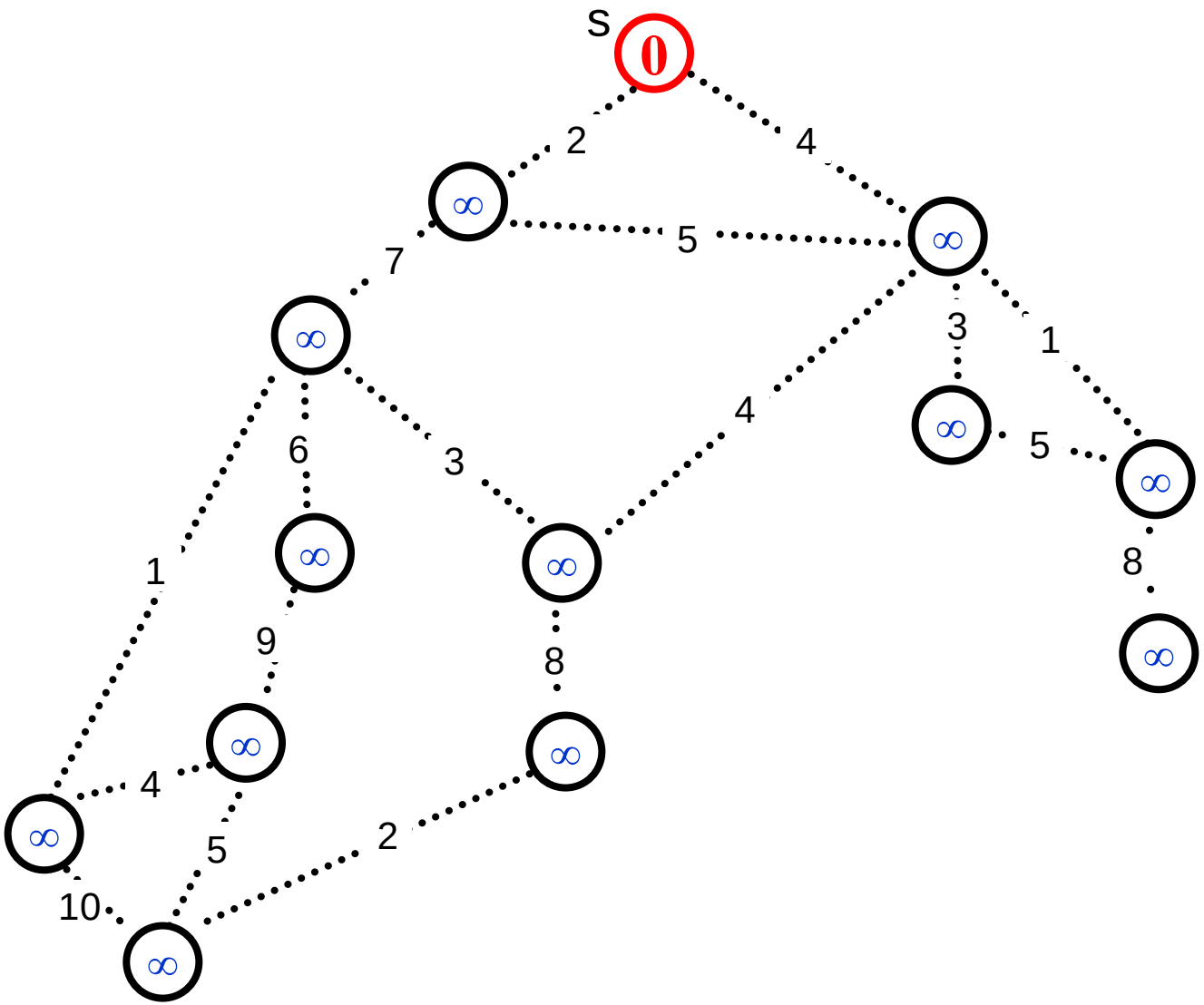
For shortest path: How do we order the input?

- Idea: Instead of ordering the input, we order the output

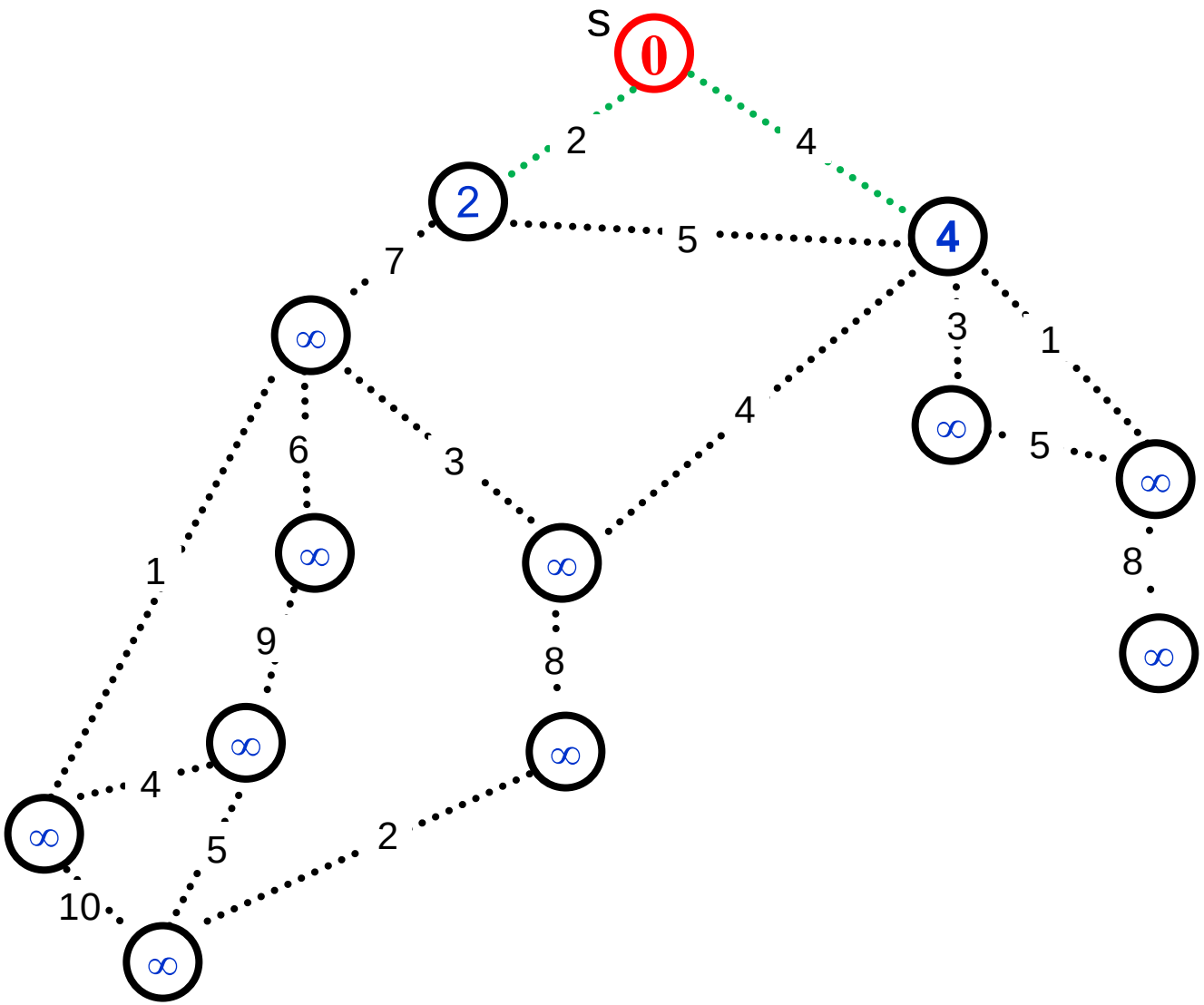
Dijkstra's Algorithm

```
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.
    }
```

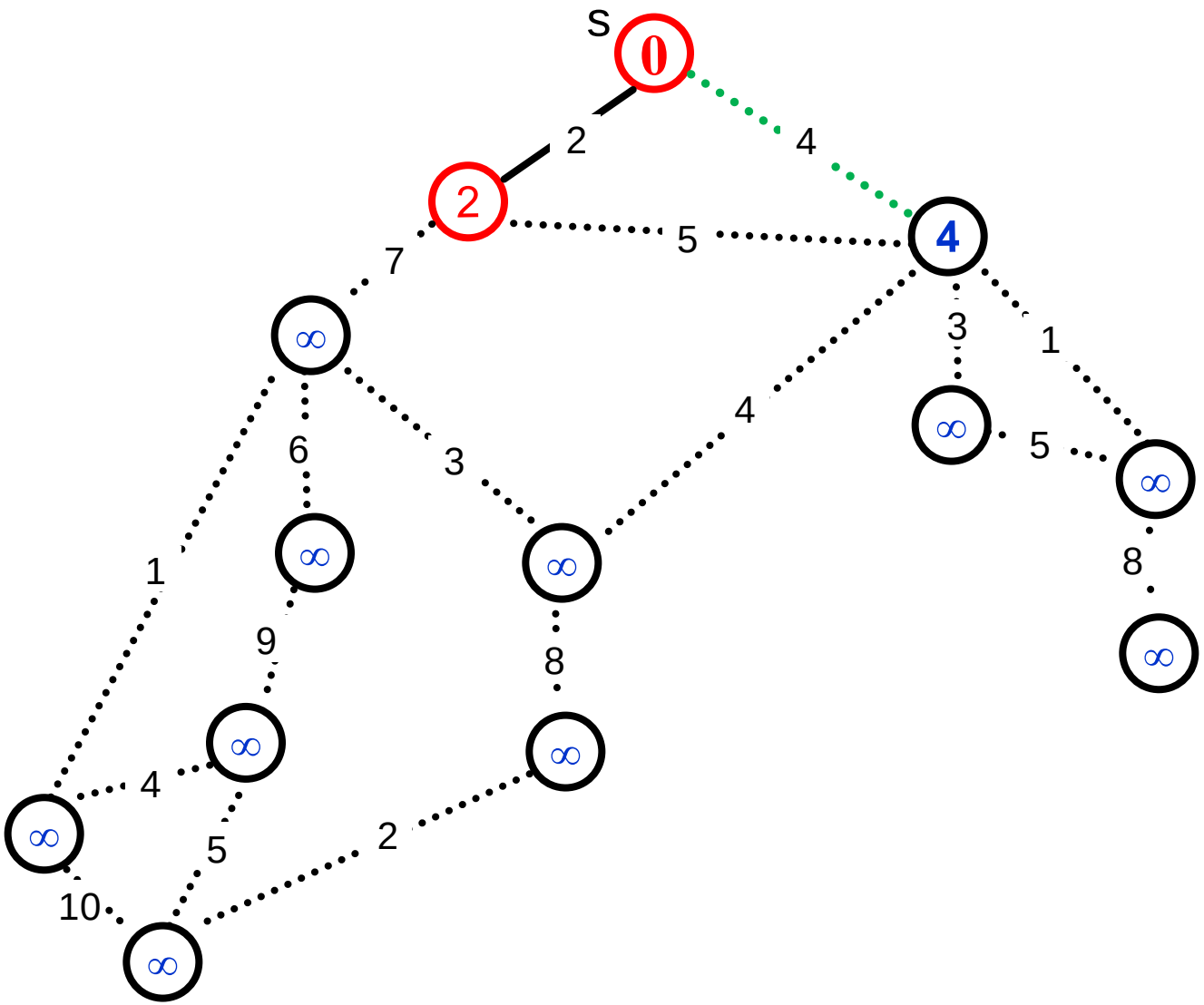
Dijkstra's Algorithm: Example



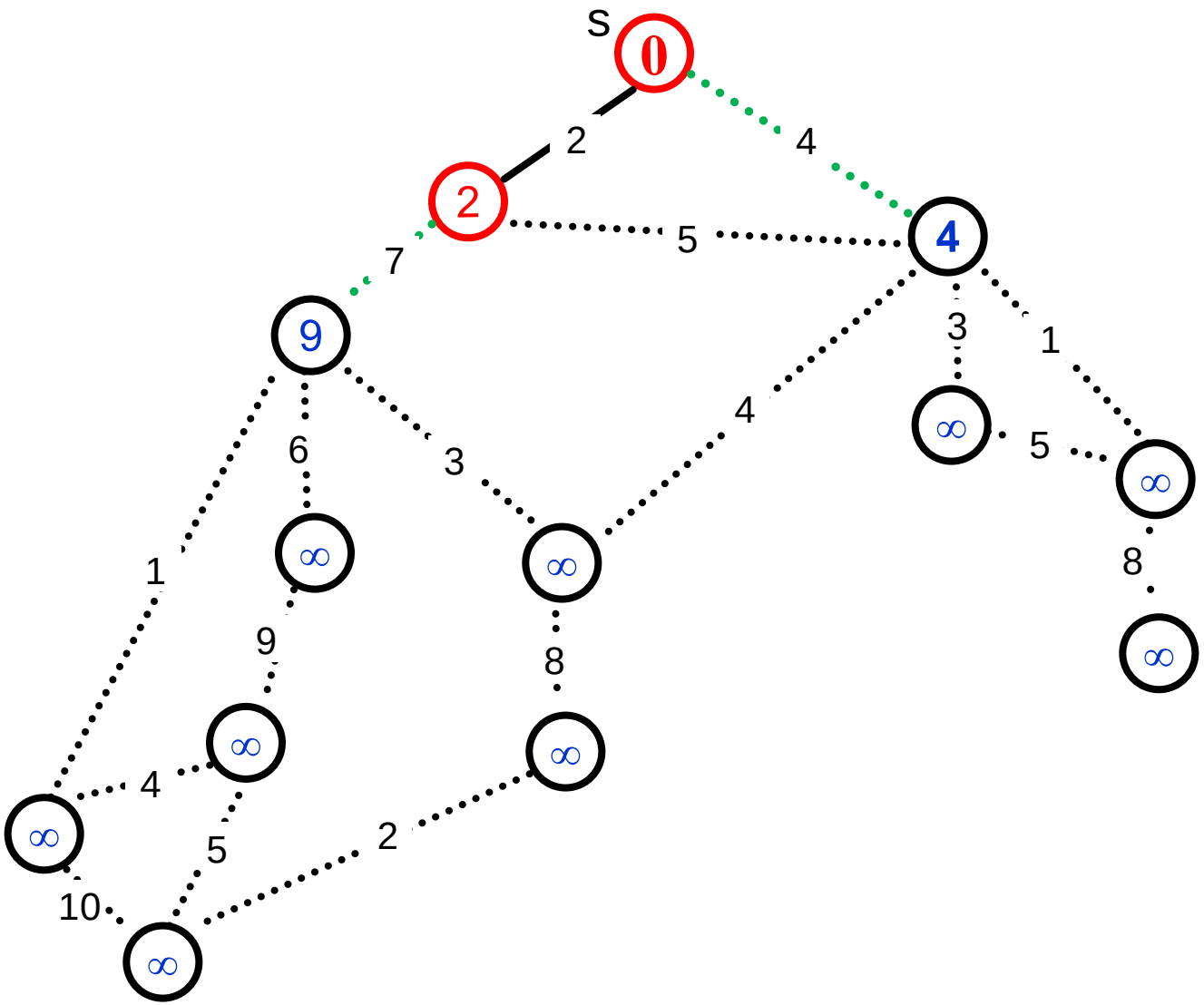
Dijkstra's Algorithm: Example



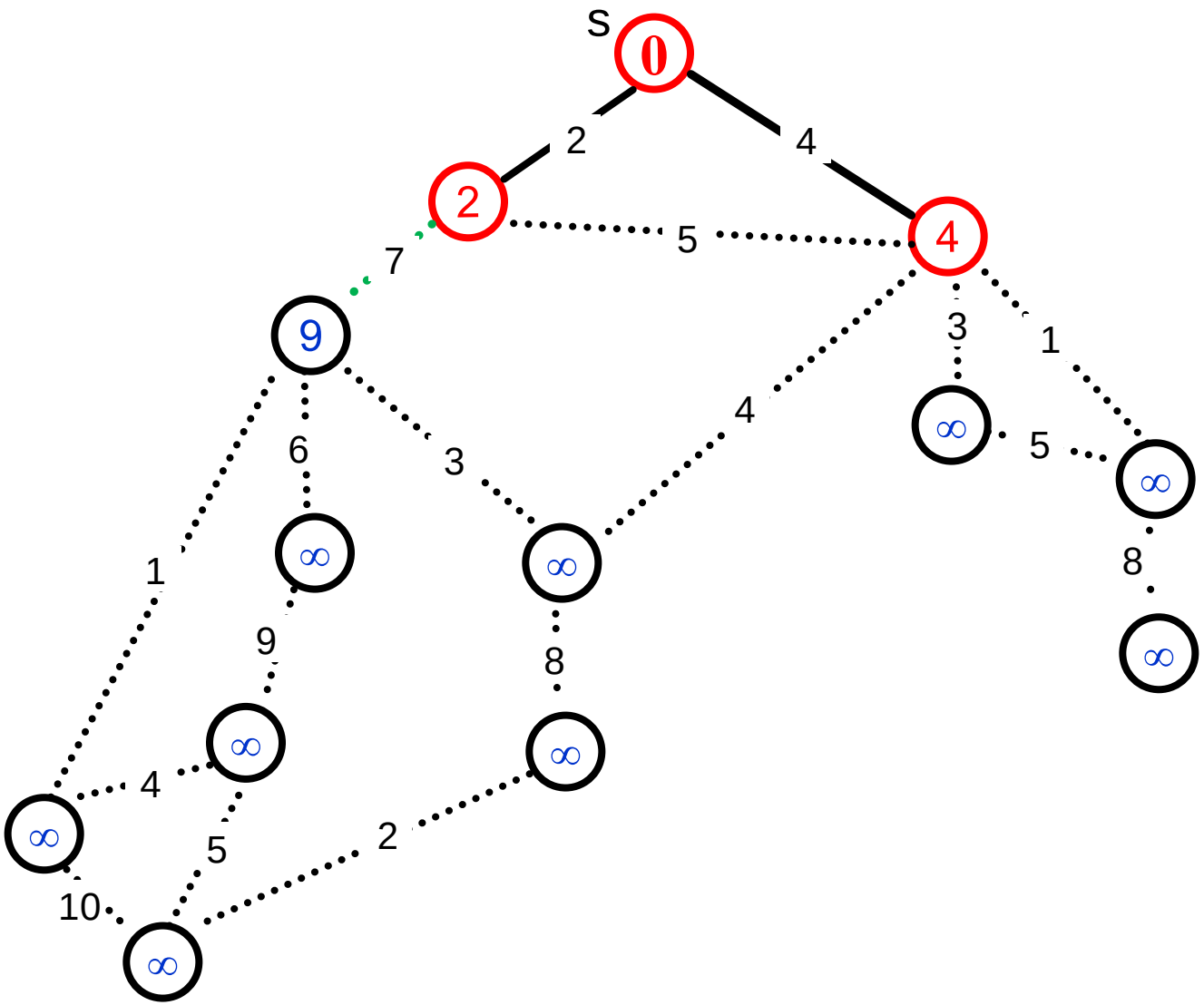
Dijkstra's Algorithm: Example



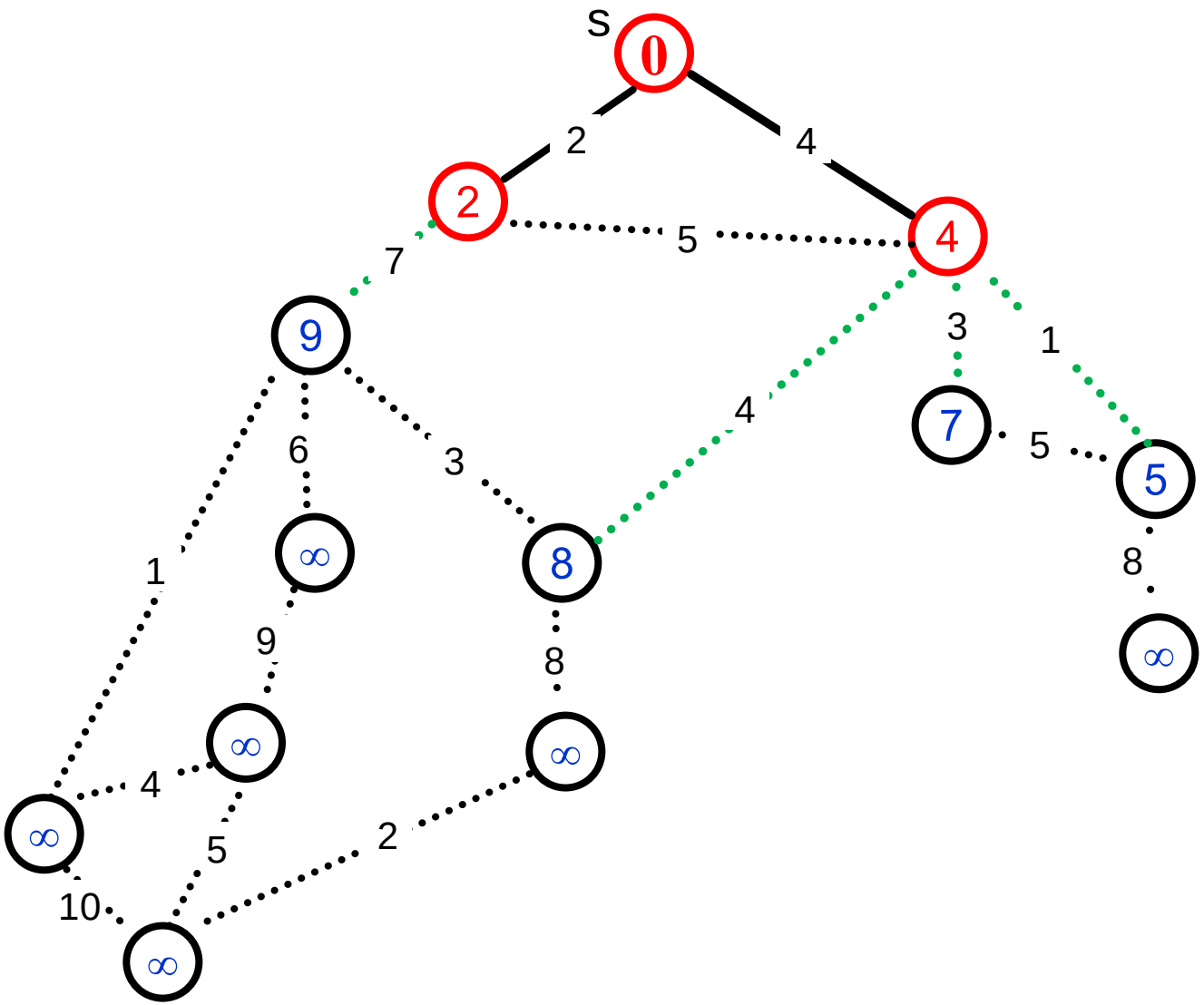
Dijkstra's Algorithm: Example



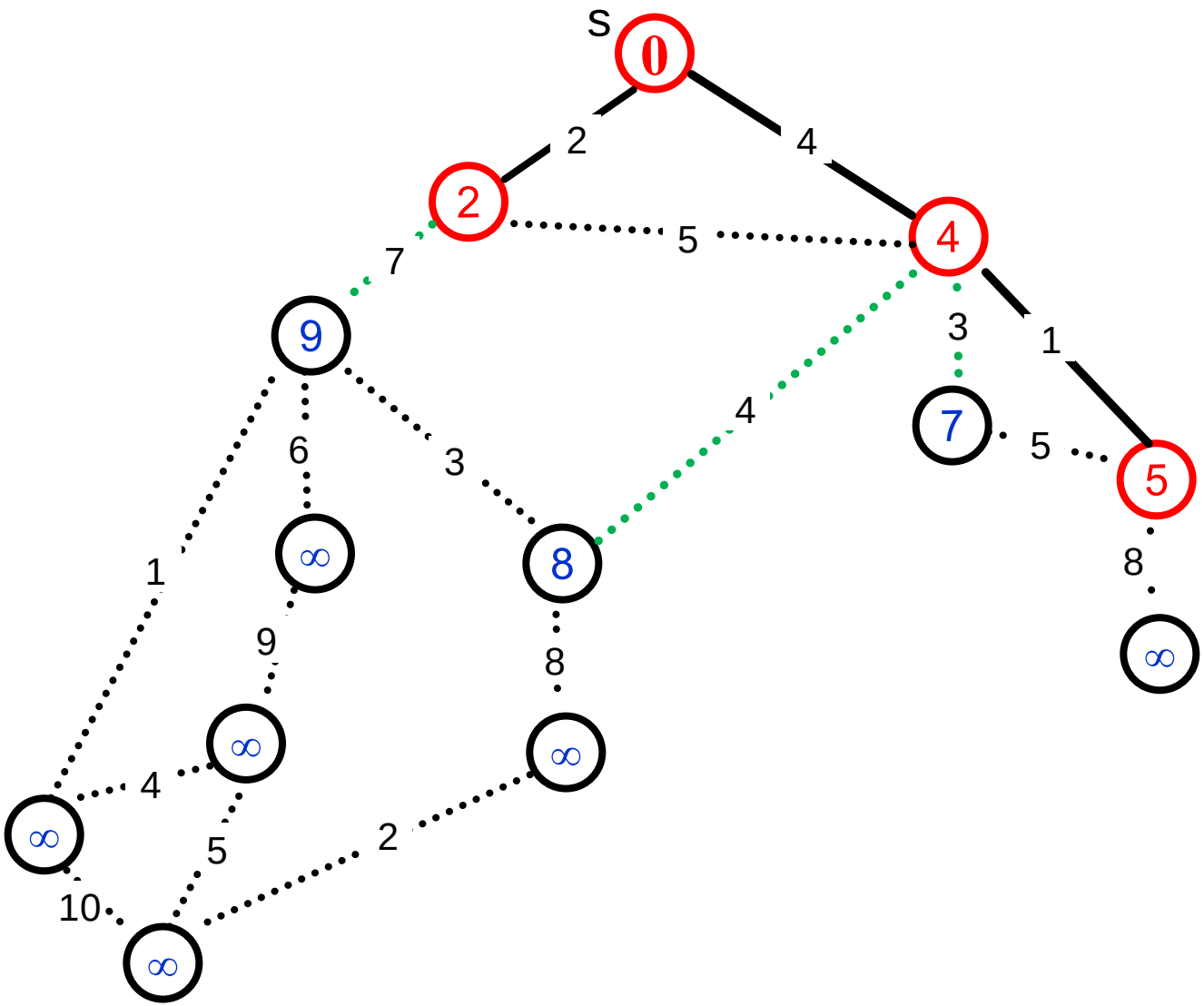
Dijkstra's Algorithm: Example



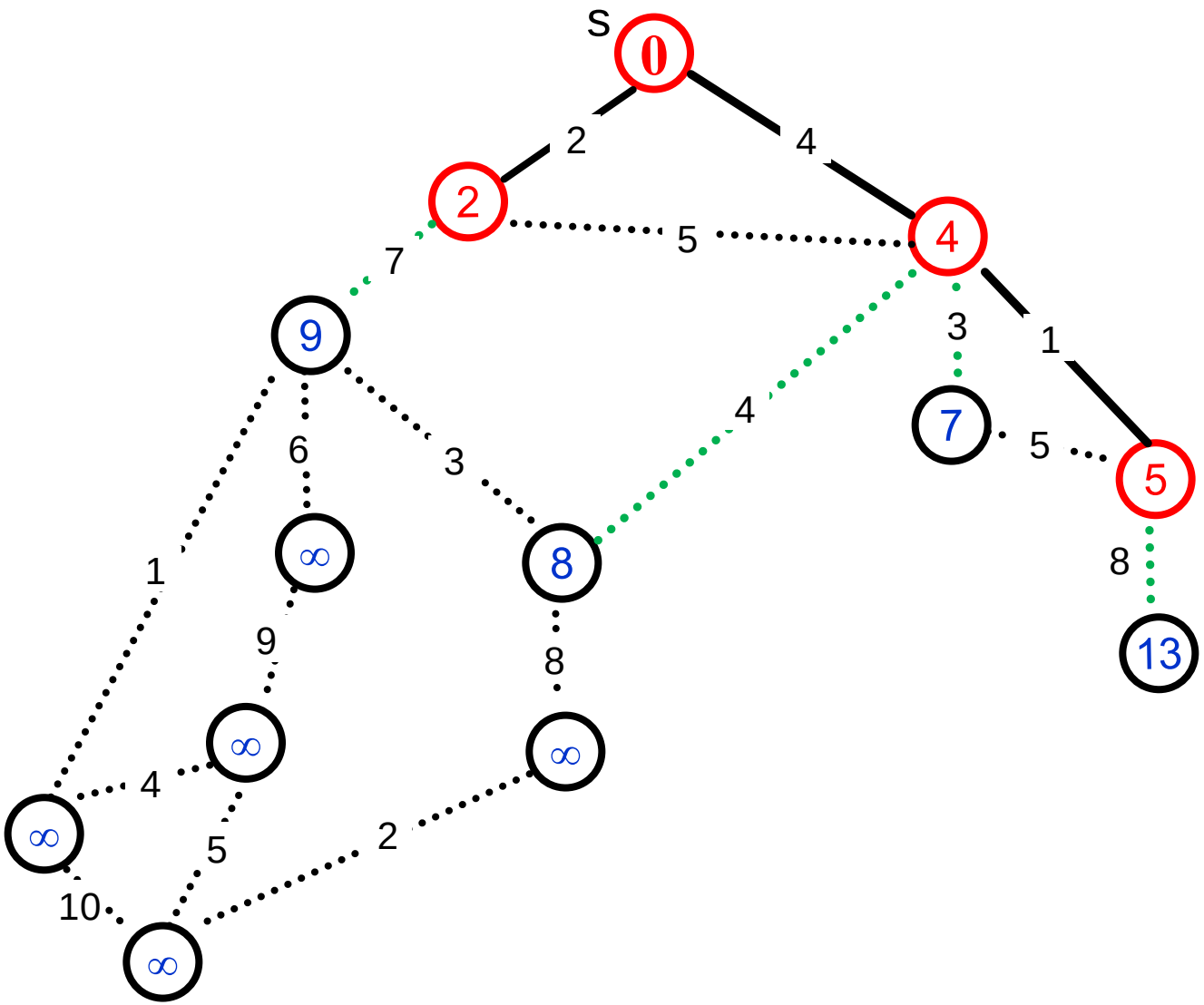
Dijkstra's Algorithm: Example



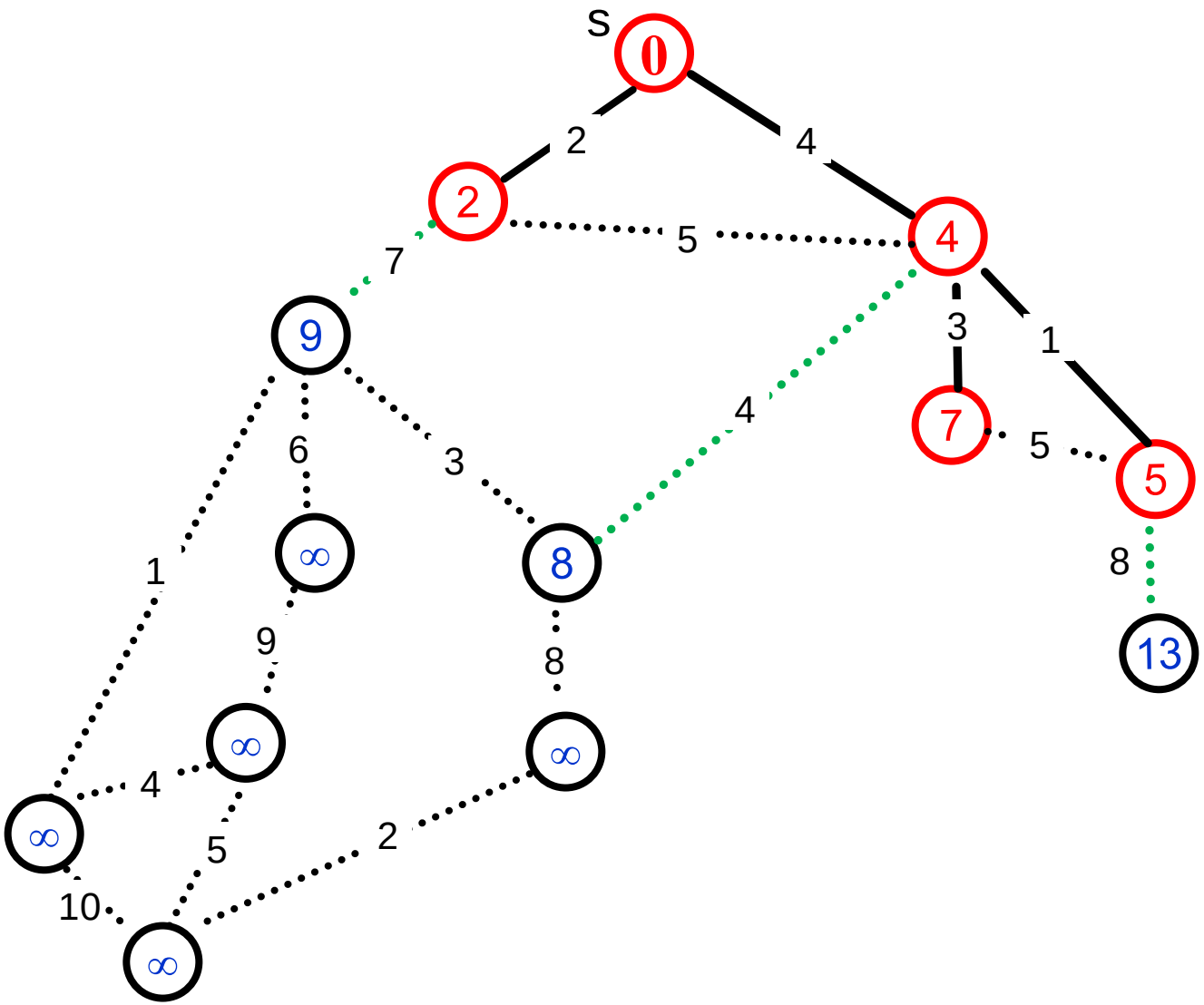
Dijkstra's Algorithm: Example



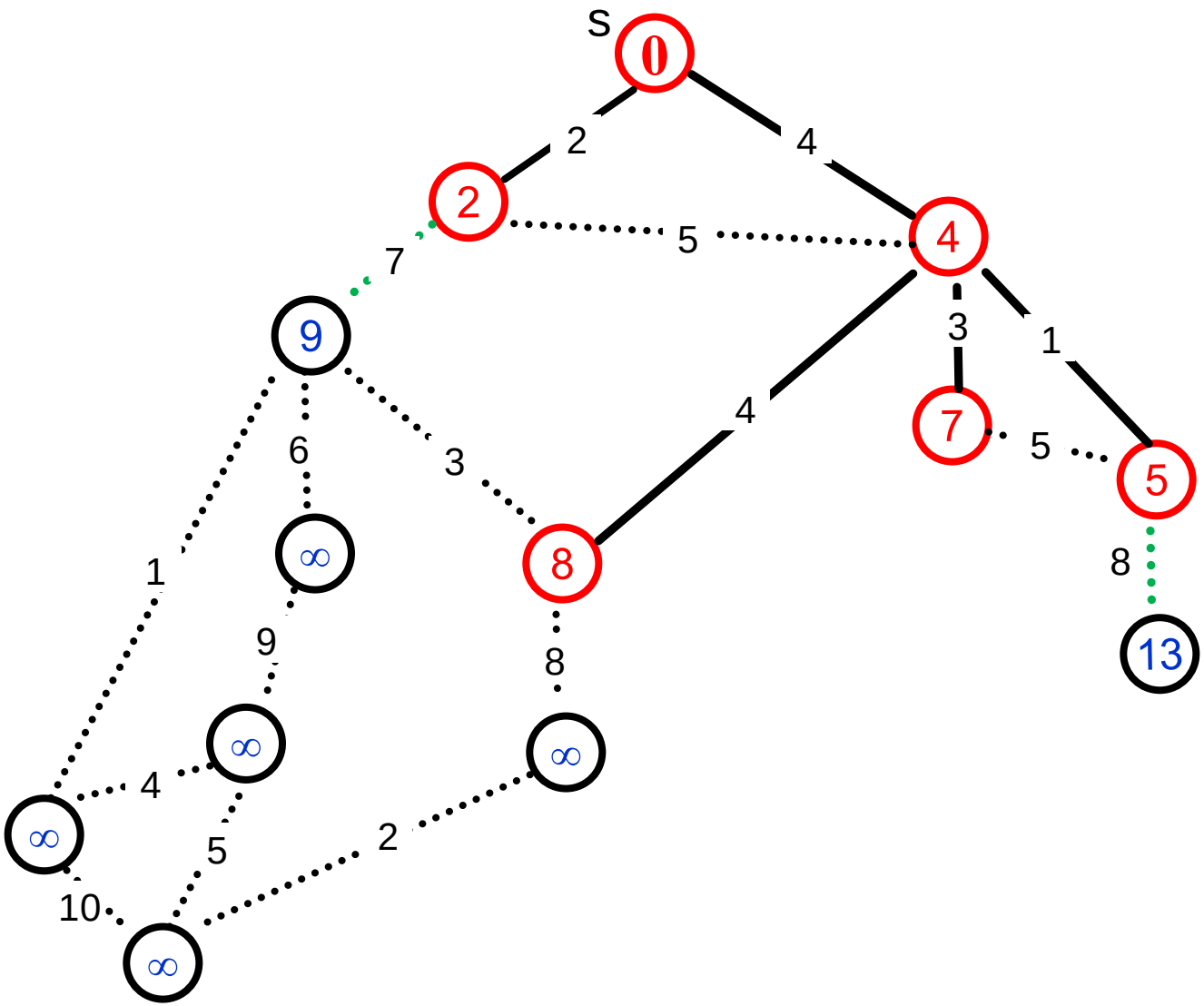
Dijkstra's Algorithm: Example



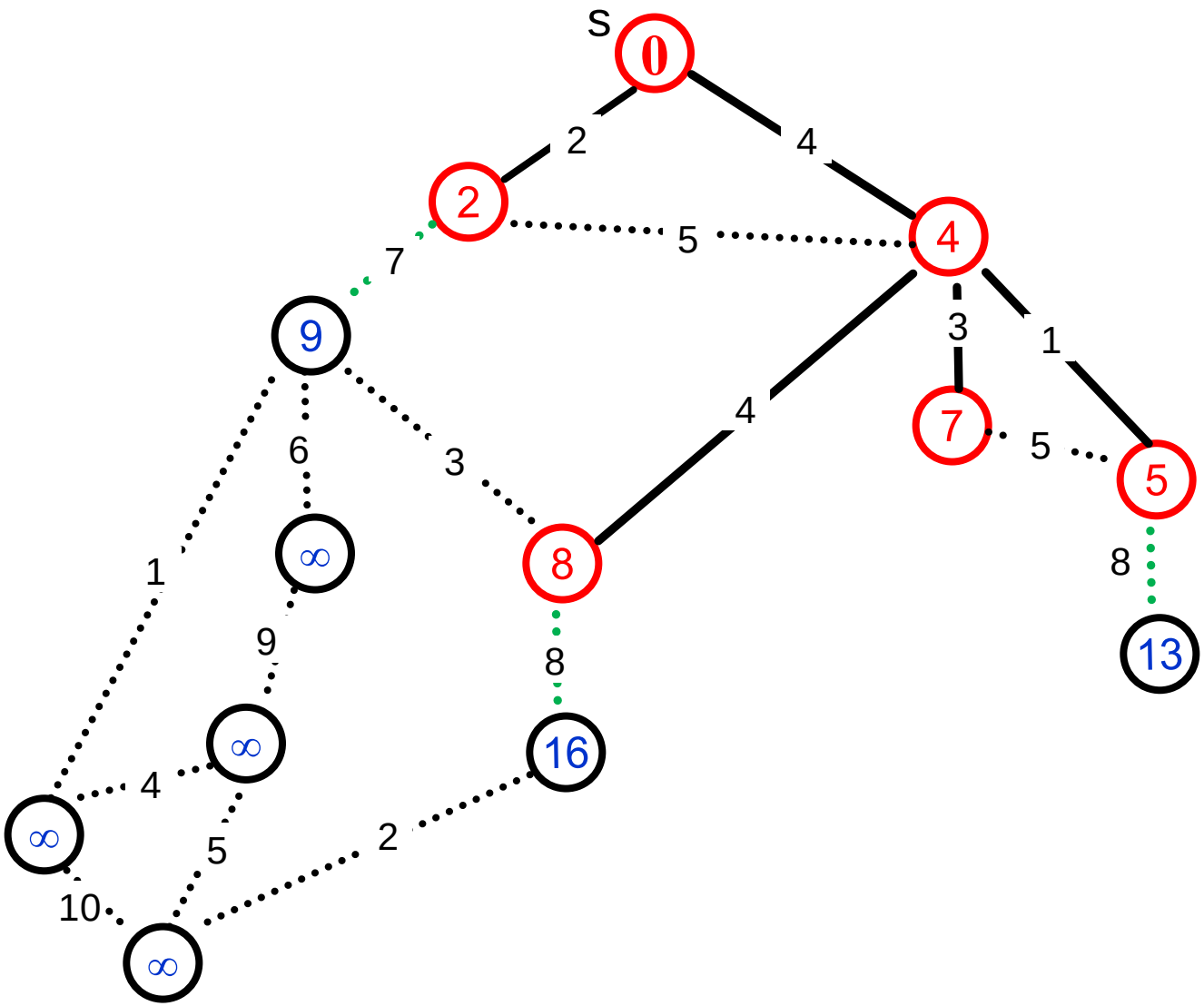
Dijkstra's Algorithm: Example



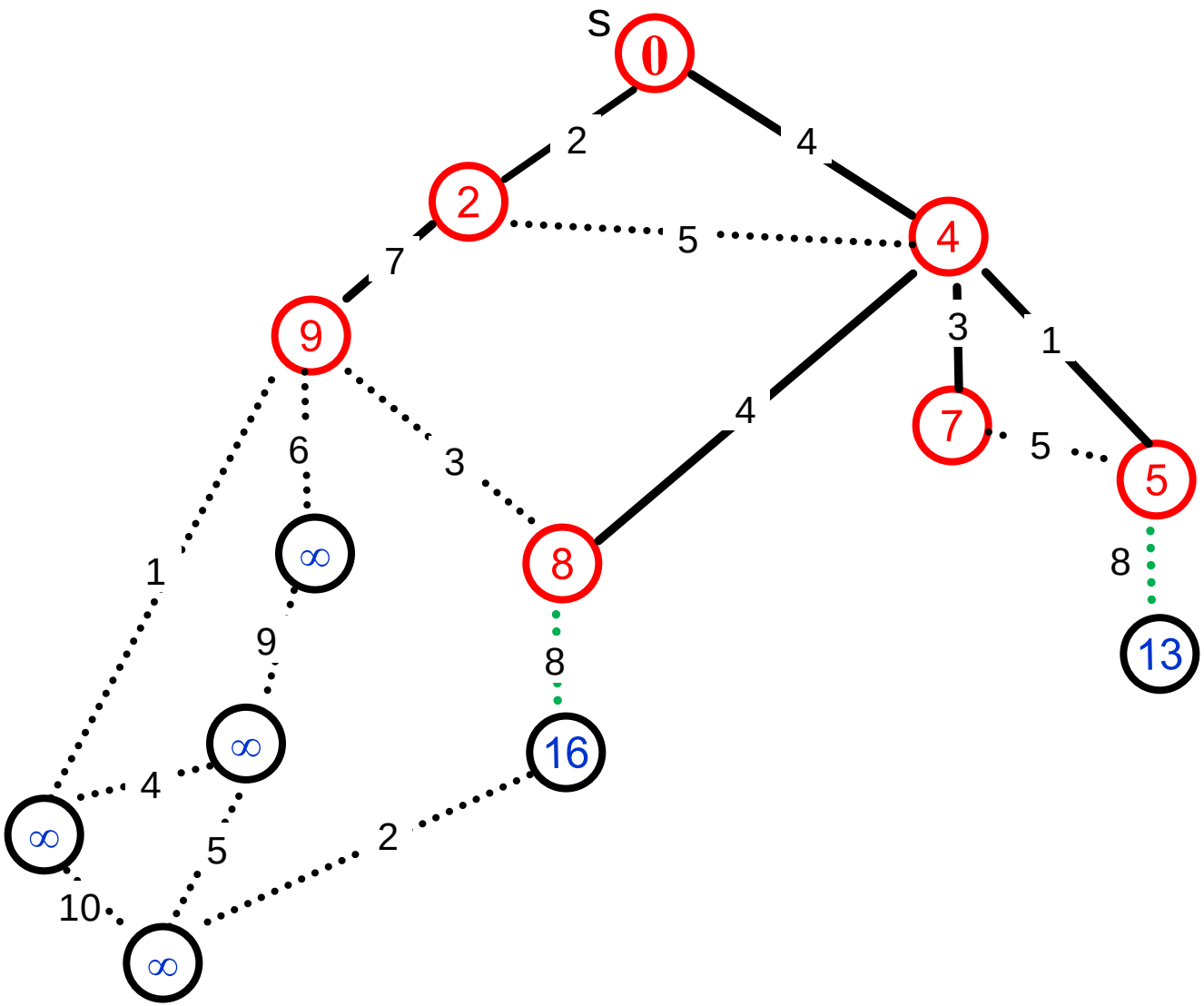
Dijkstra's Algorithm: Example



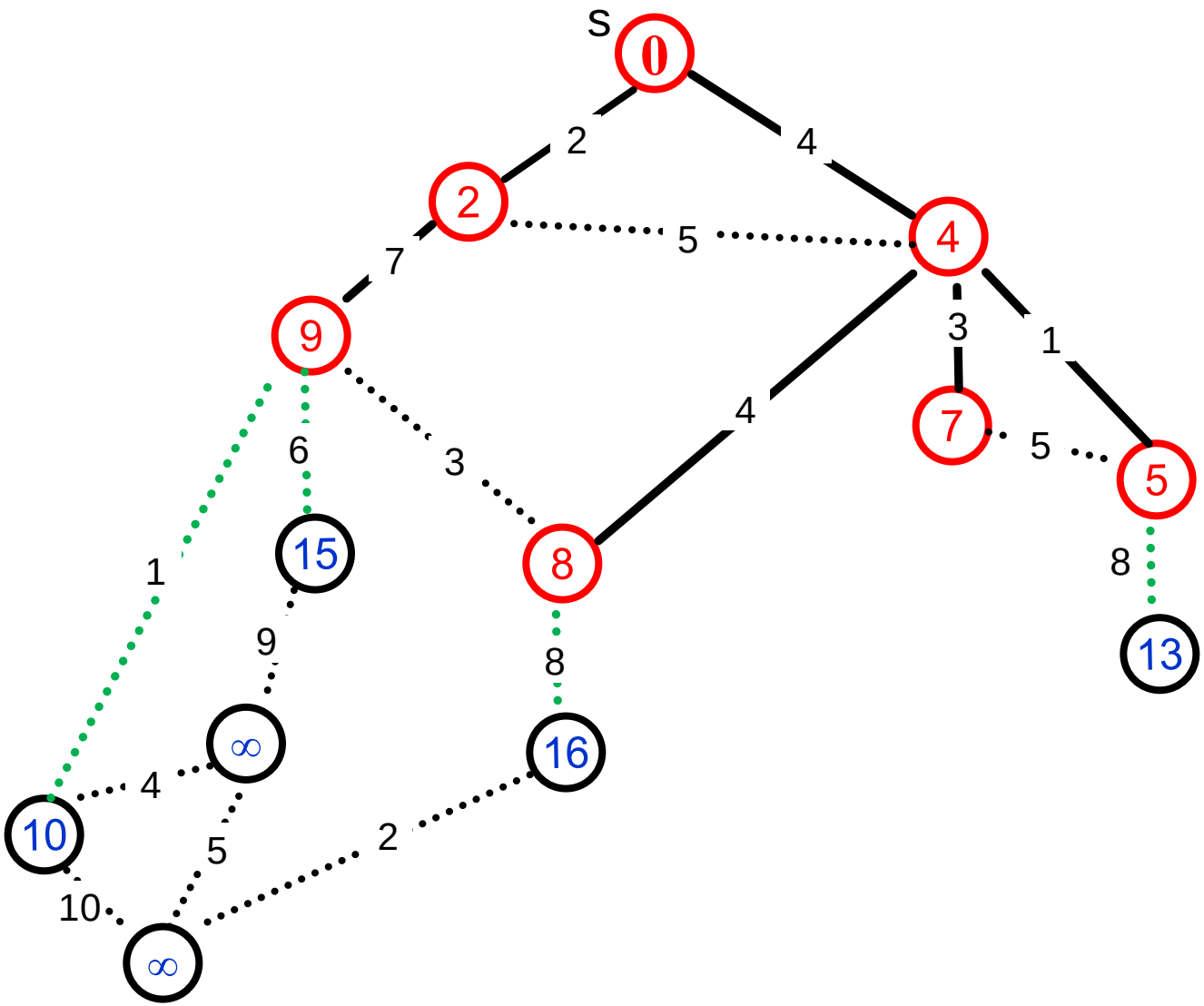
Dijkstra's Algorithm: Example



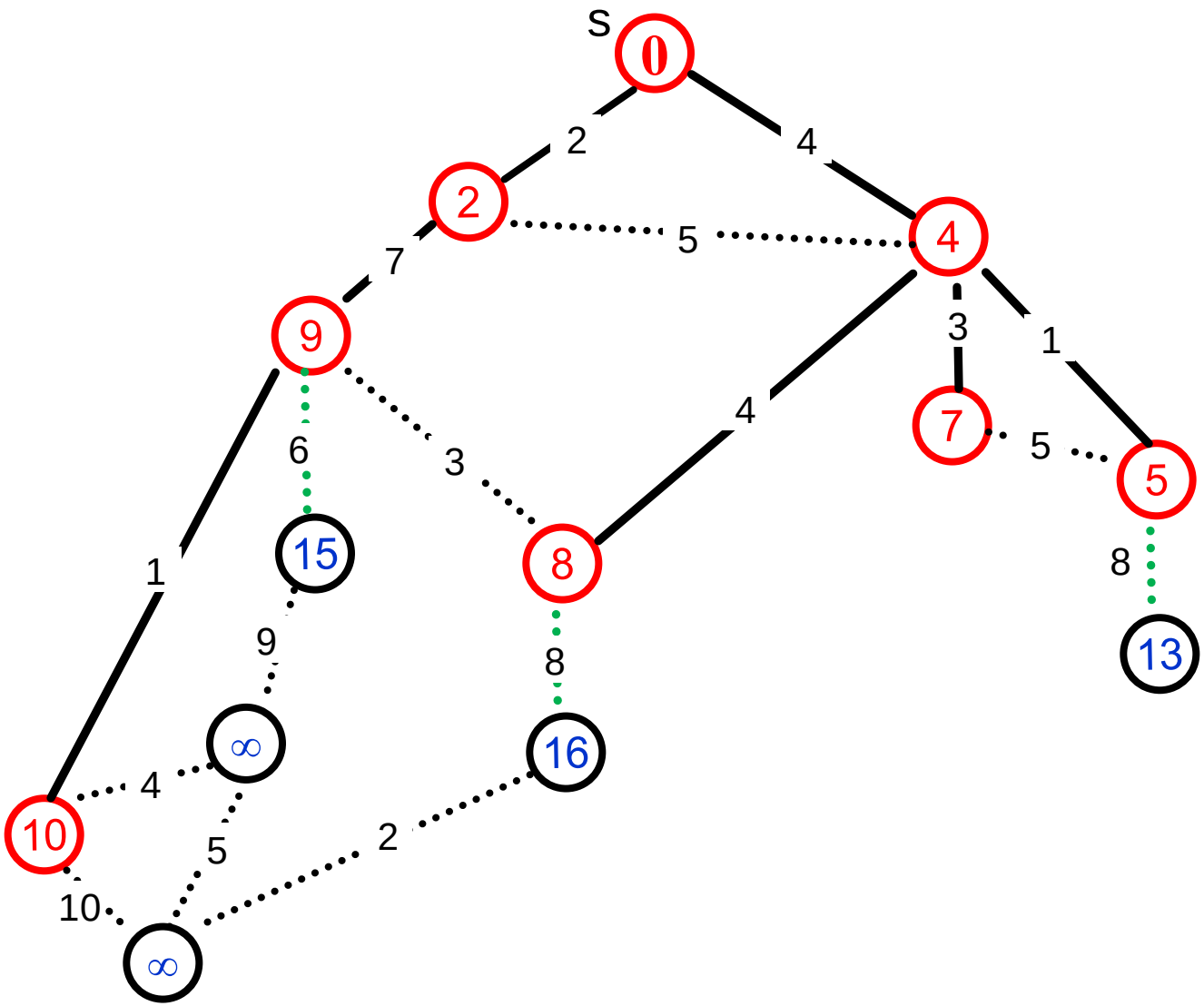
Dijkstra's Algorithm: Example



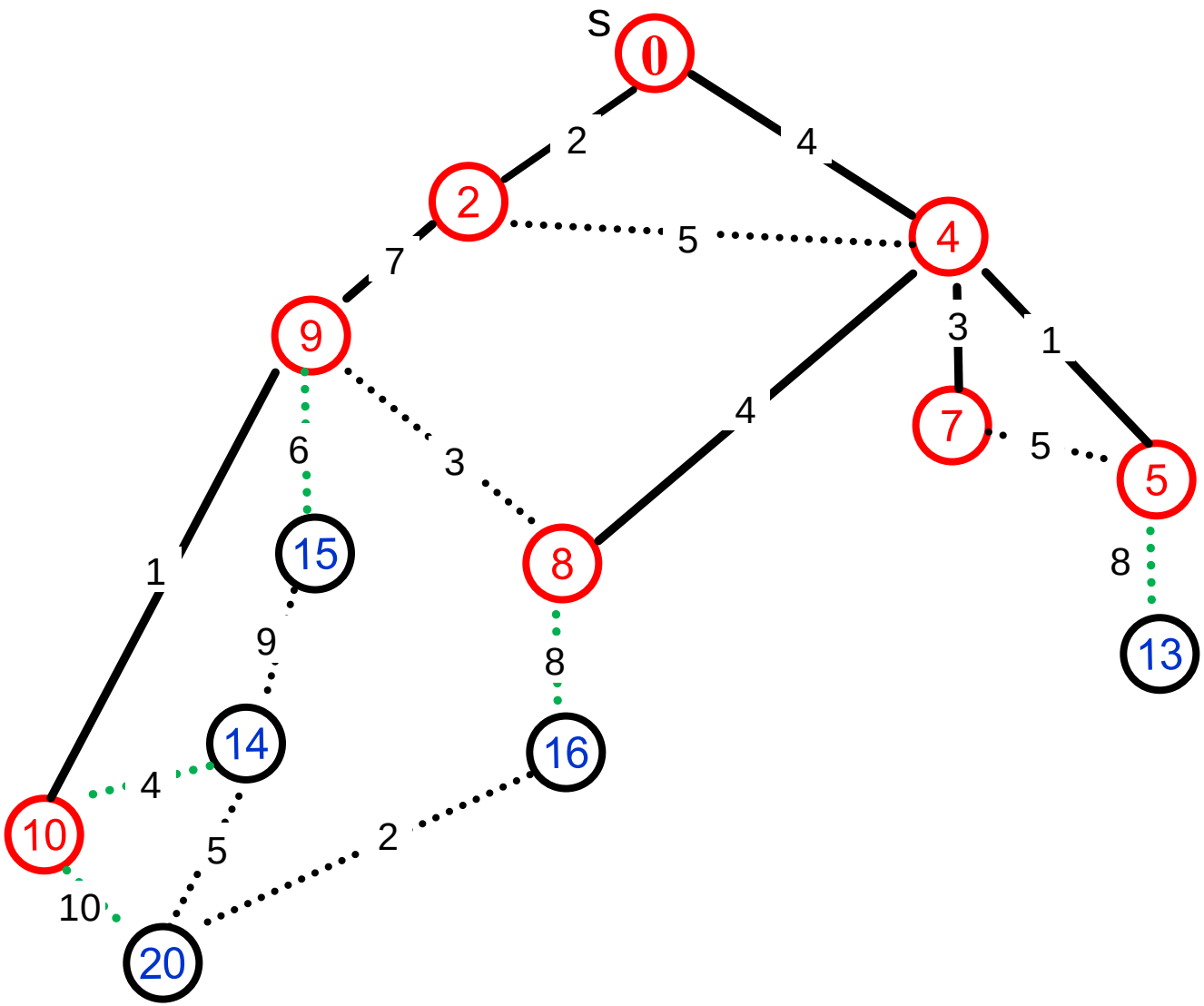
Dijkstra's Algorithm: Example



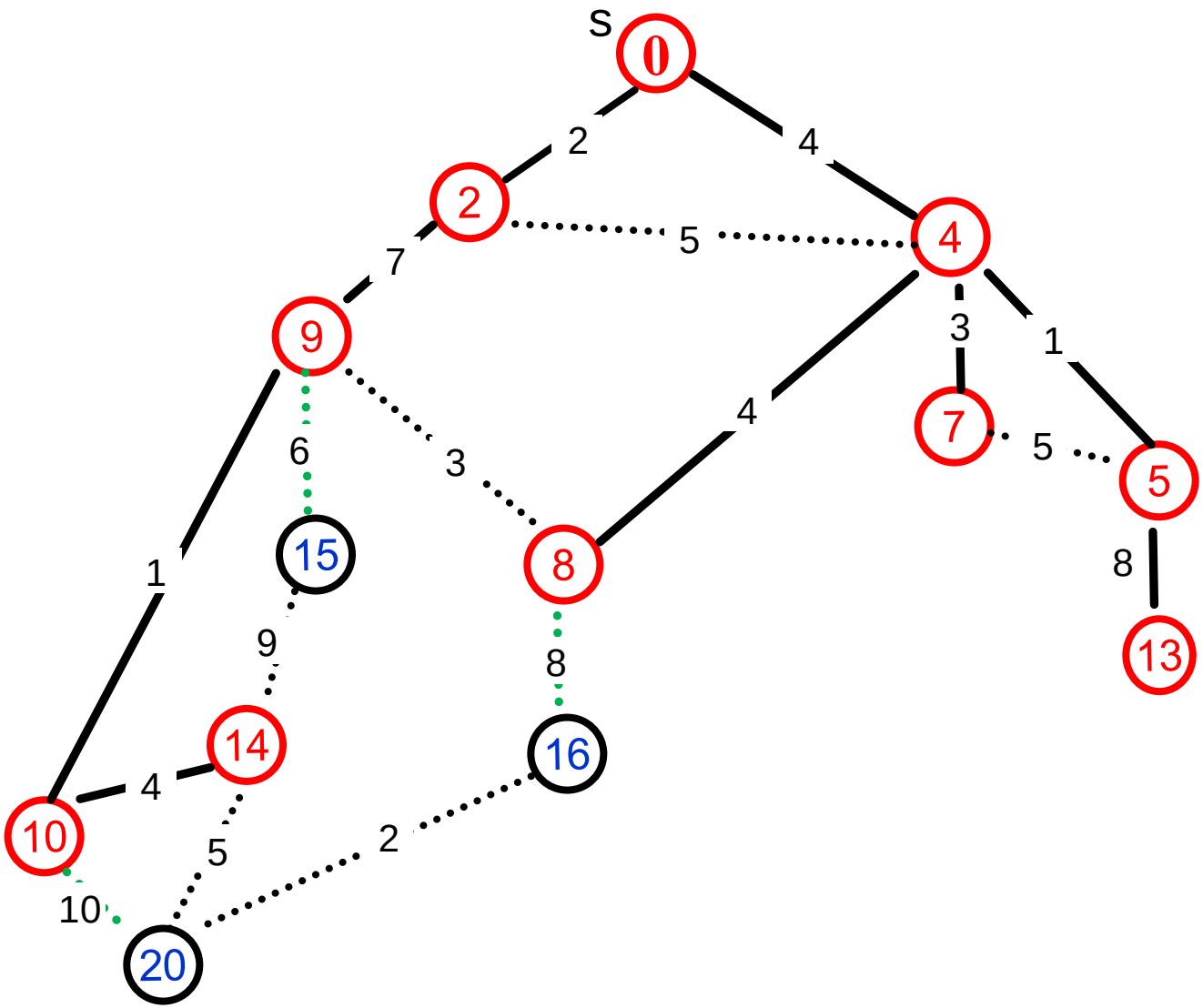
Dijkstra's Algorithm: Example



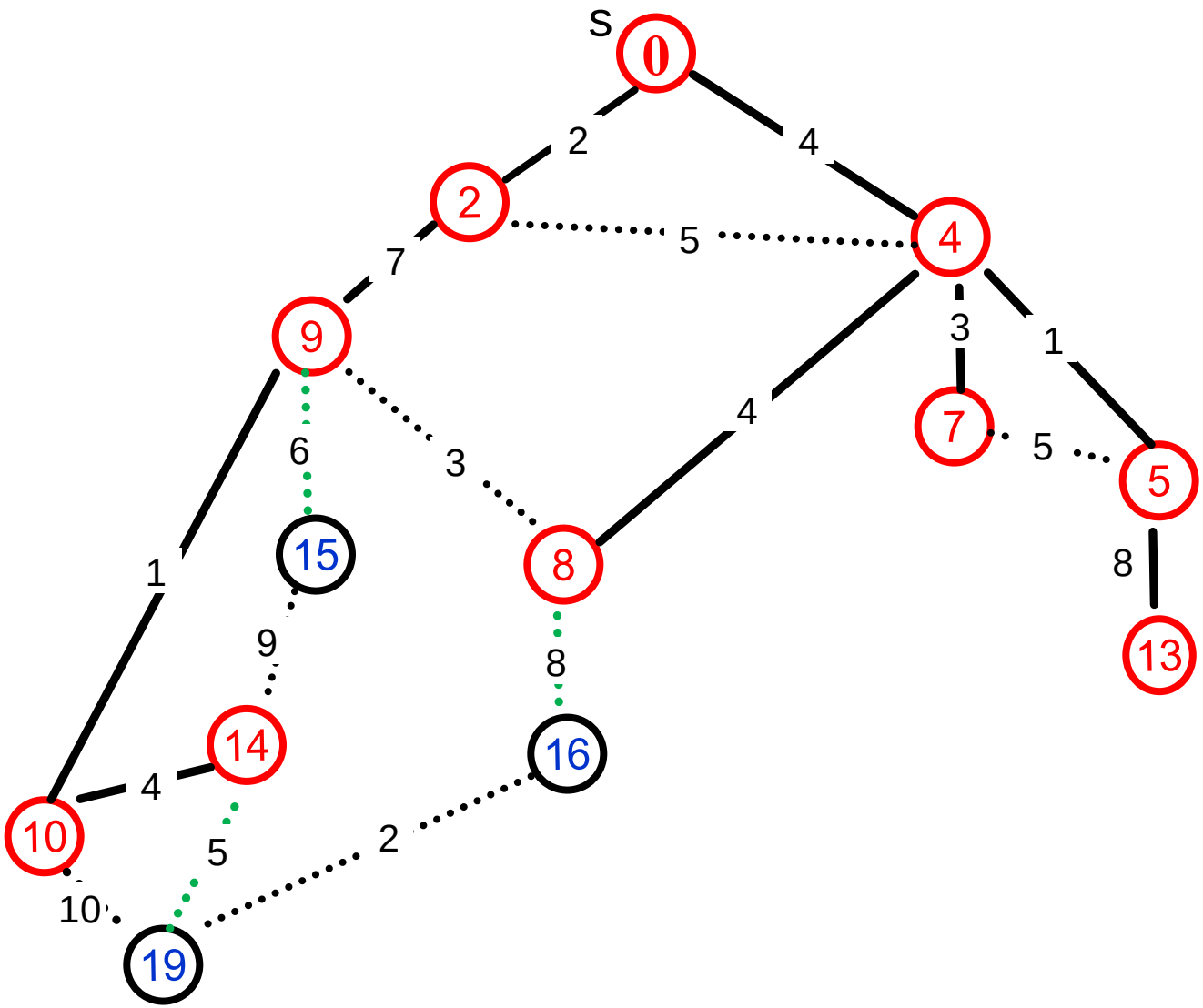
Dijkstra's Algorithm: Example



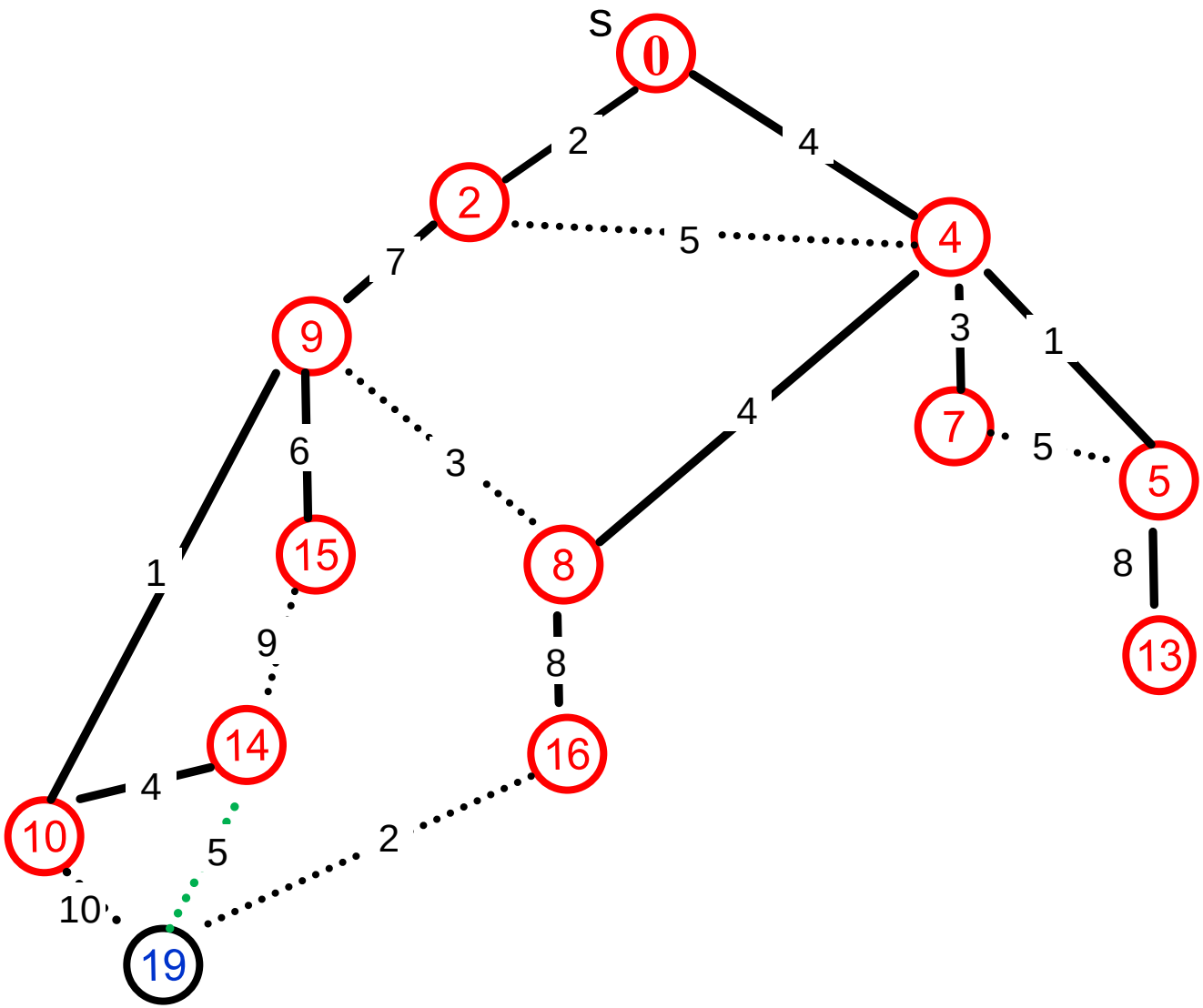
Dijkstra's Algorithm: Example



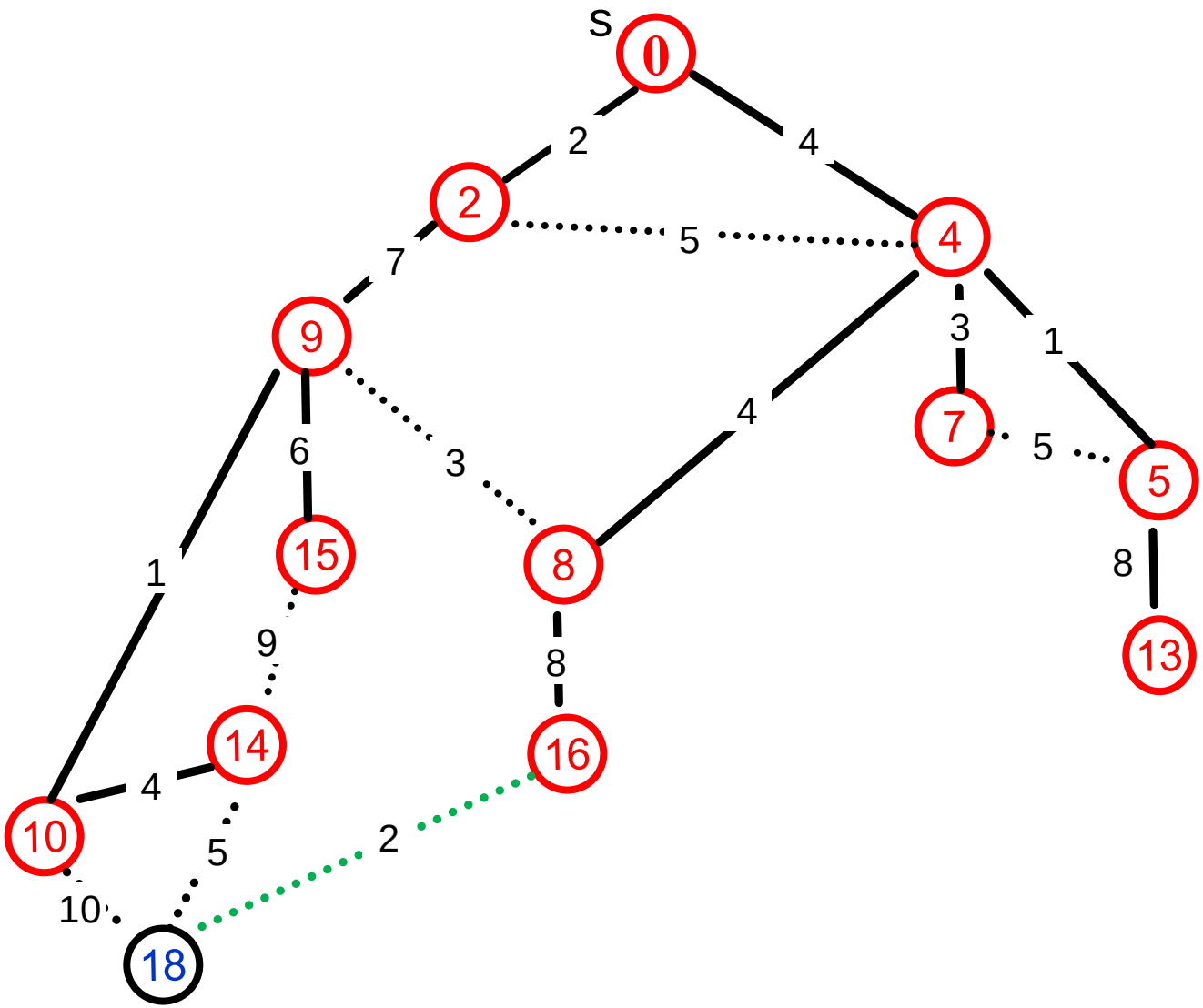
Dijkstra's Algorithm: Example



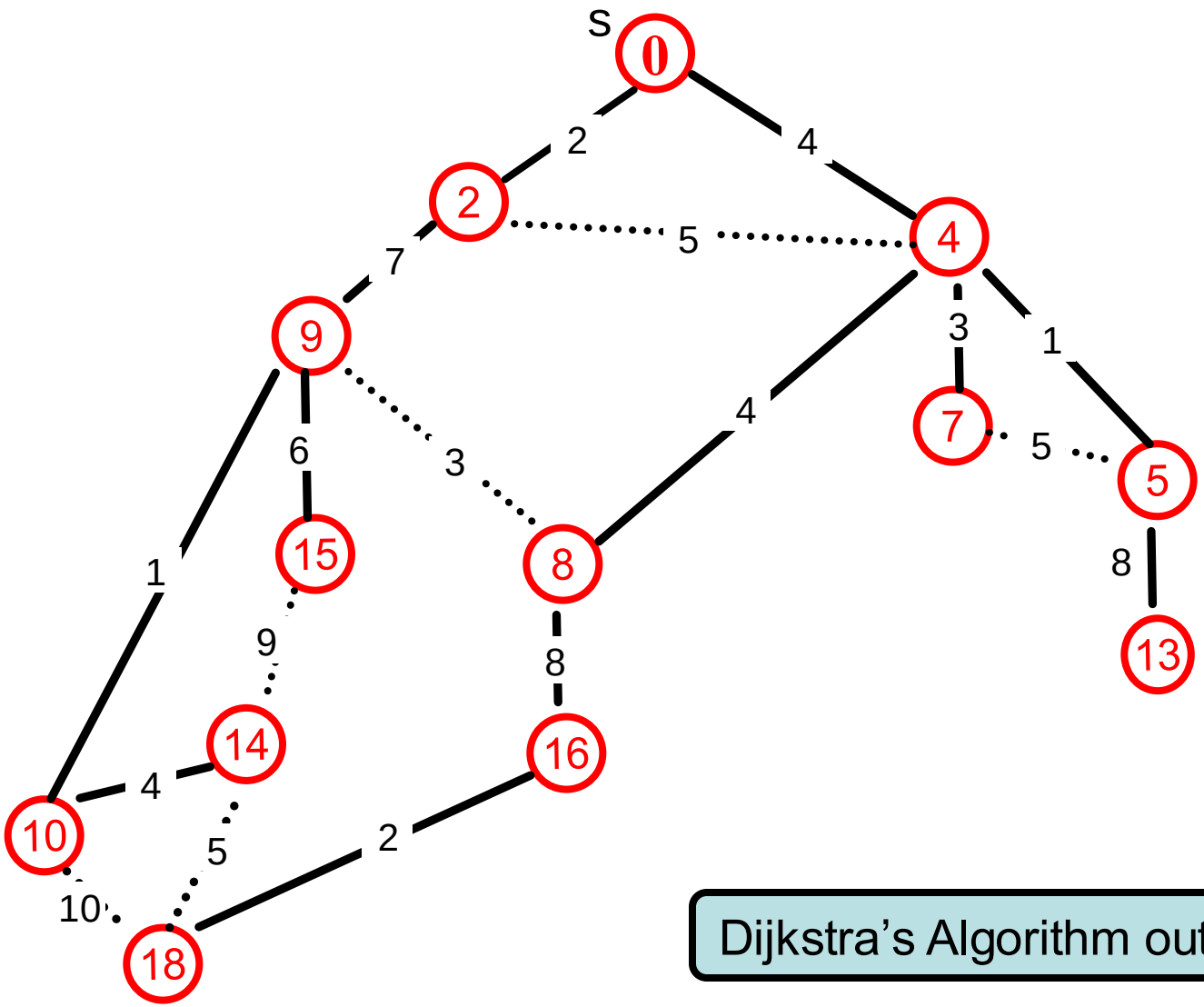
Dijkstra's Algorithm: Example



Dijkstra's Algorithm: Example



Dijkstra's Algorithm: Example



Dijkstra's Algorithm outputs a tree.