CSE 421

Greedy Alg: Union Find/Dijkstra's Alg

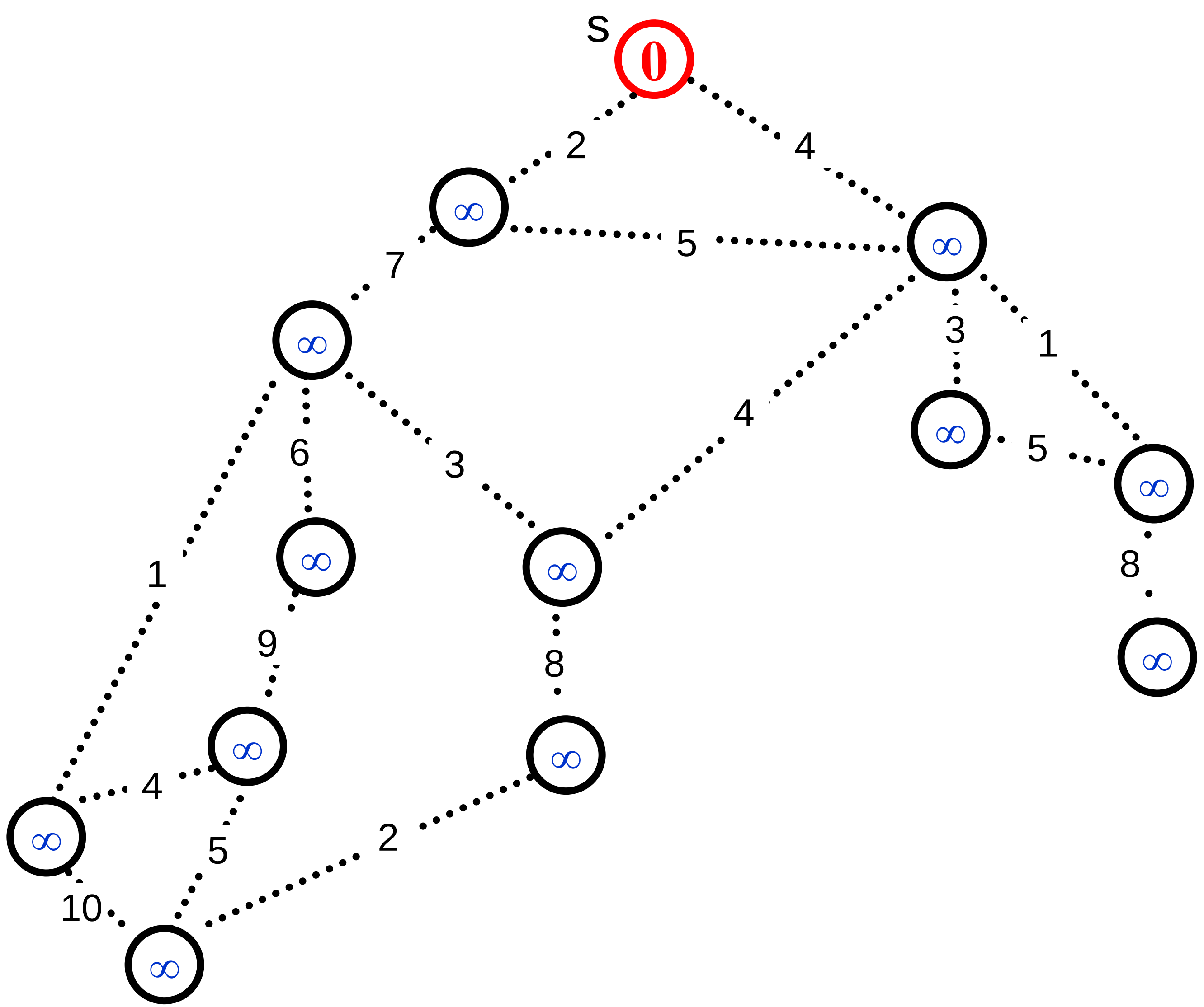
Shayan Oveis Gharan

Dijkstra's Algorithm

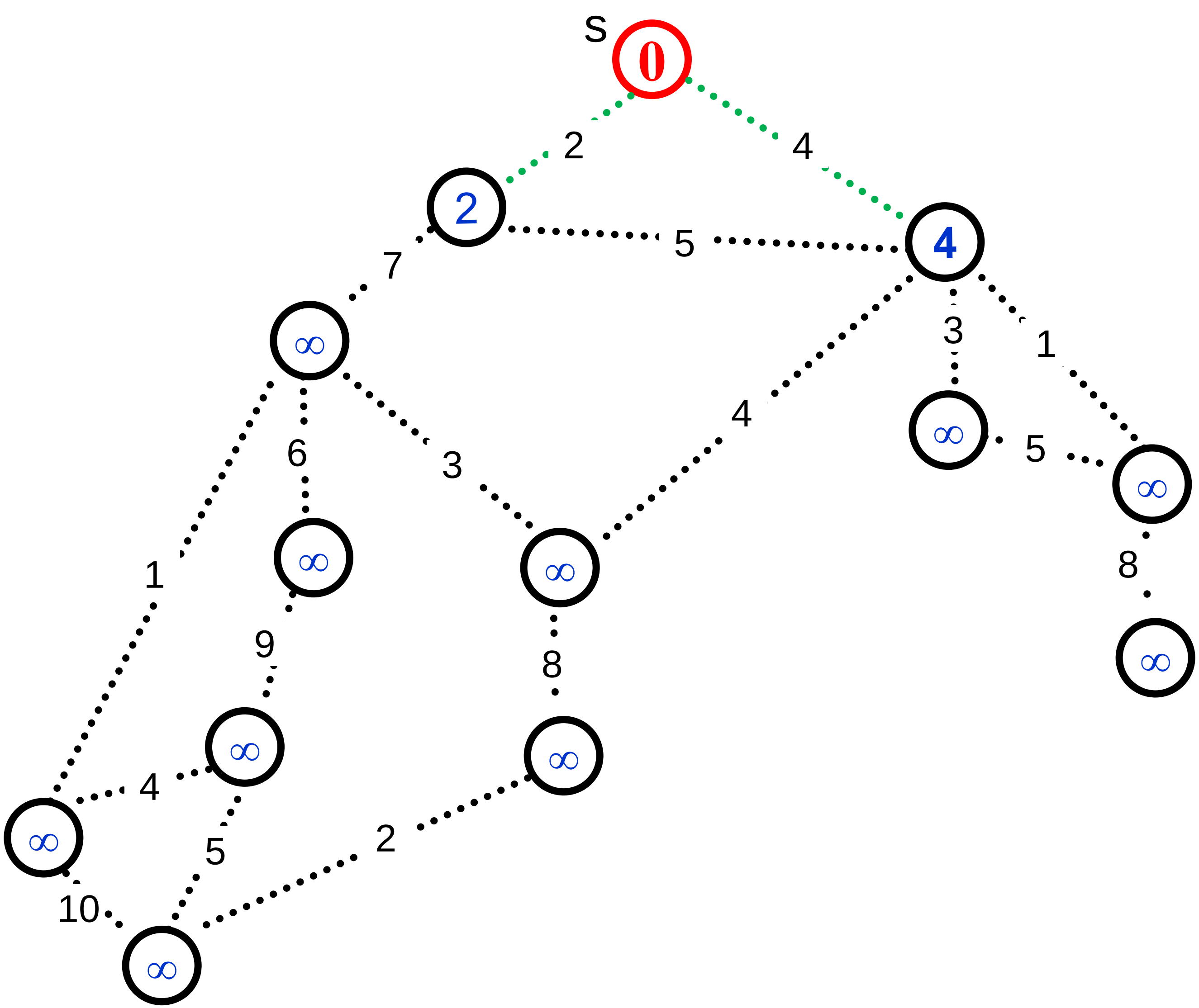
```
Dijkstra(G, c, s) {
   d[s] ← 0
   foreach (v ∈ V) d[v] ← ∞ //This is the key of node v
   foreach (v ∈ V) insert v onto a priority queue Q
   Initialize set of explored nodes S ← {s}

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (d[u]+c_e < d[v]))
             d[v] ← d[u] + c_e
             Decrease key of v to d[v].
             Parent(v) ← u
}
```

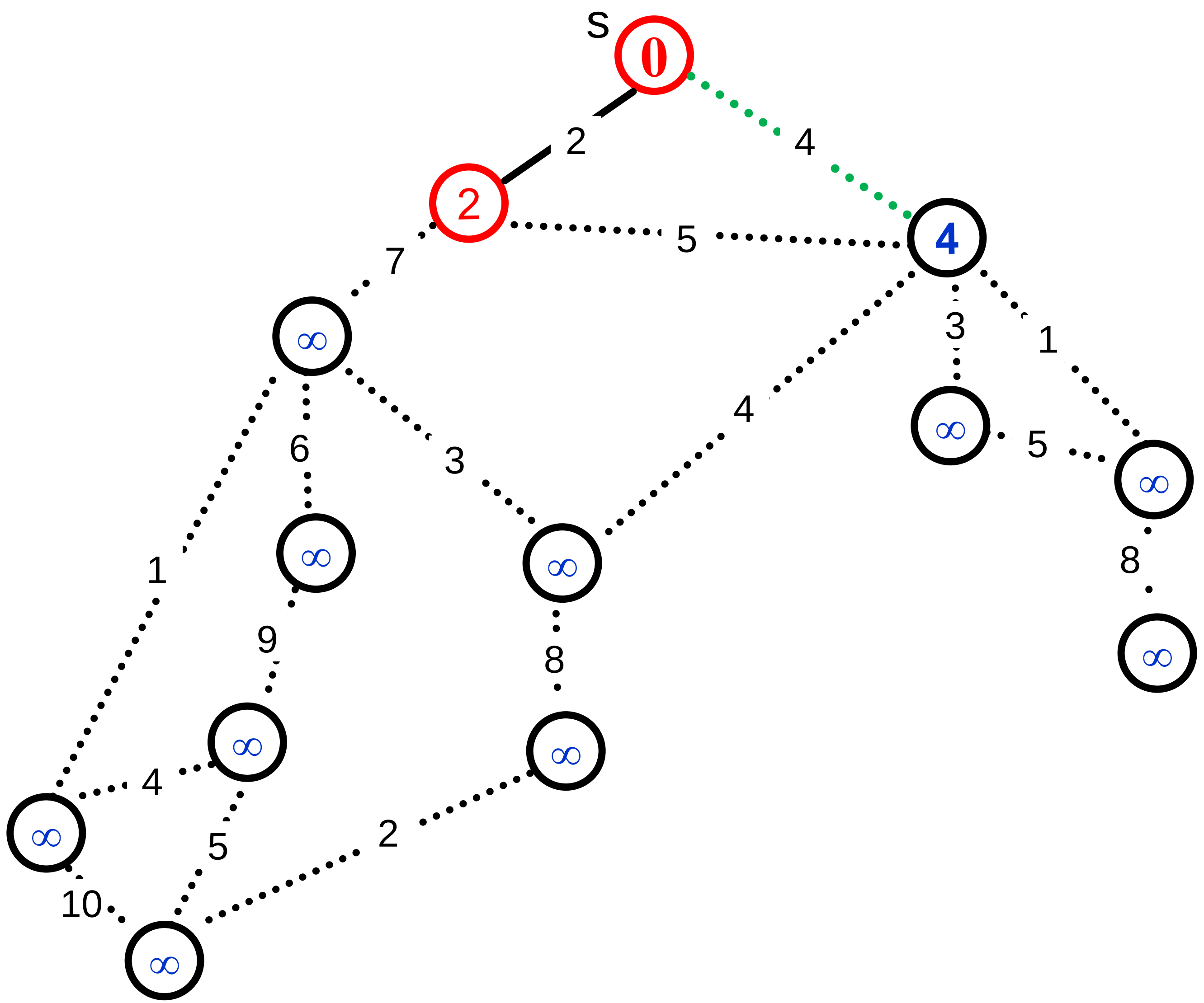
Dijkstra's Algorithm: Example



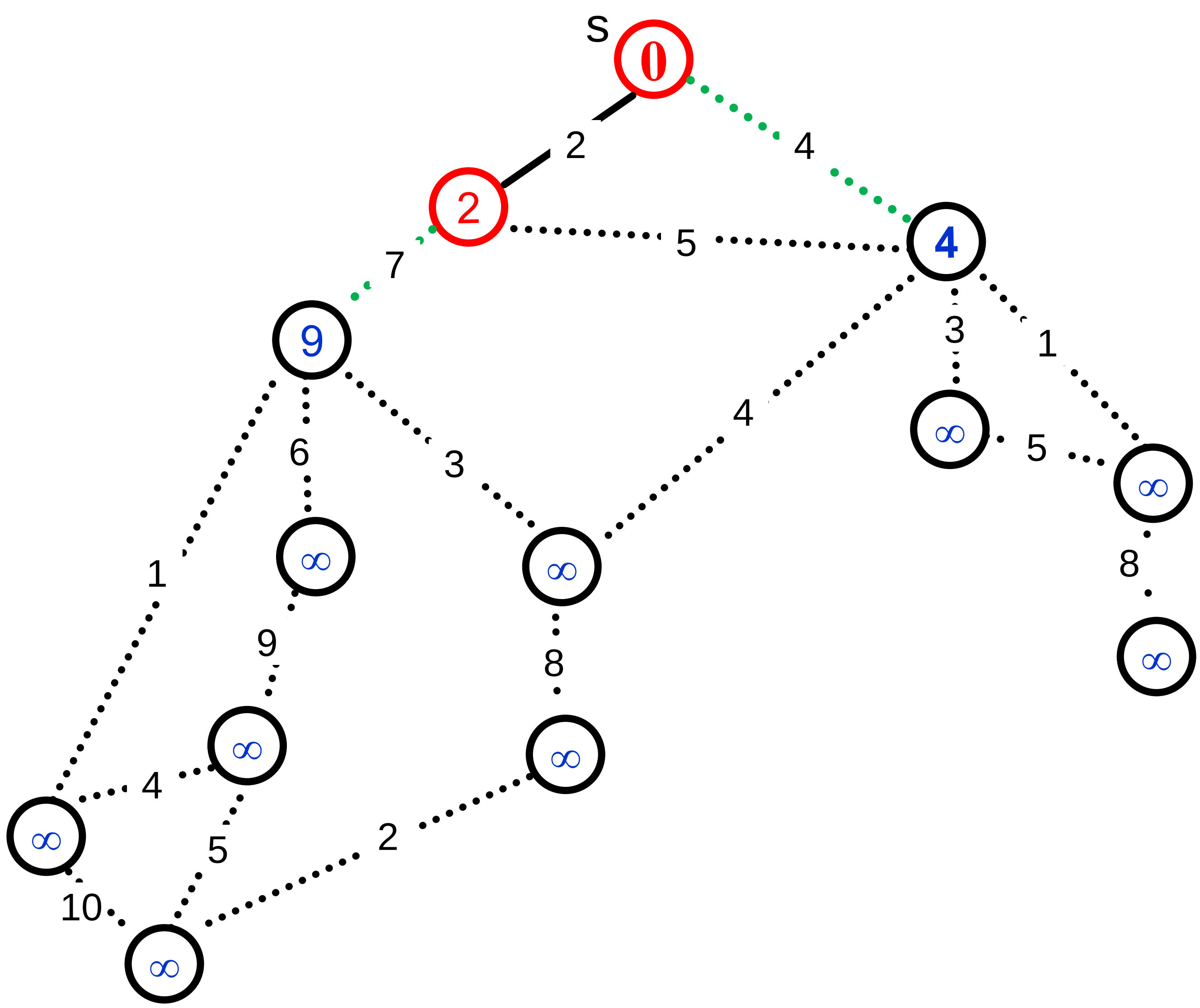
Dijkstra’s Algorithm: Example



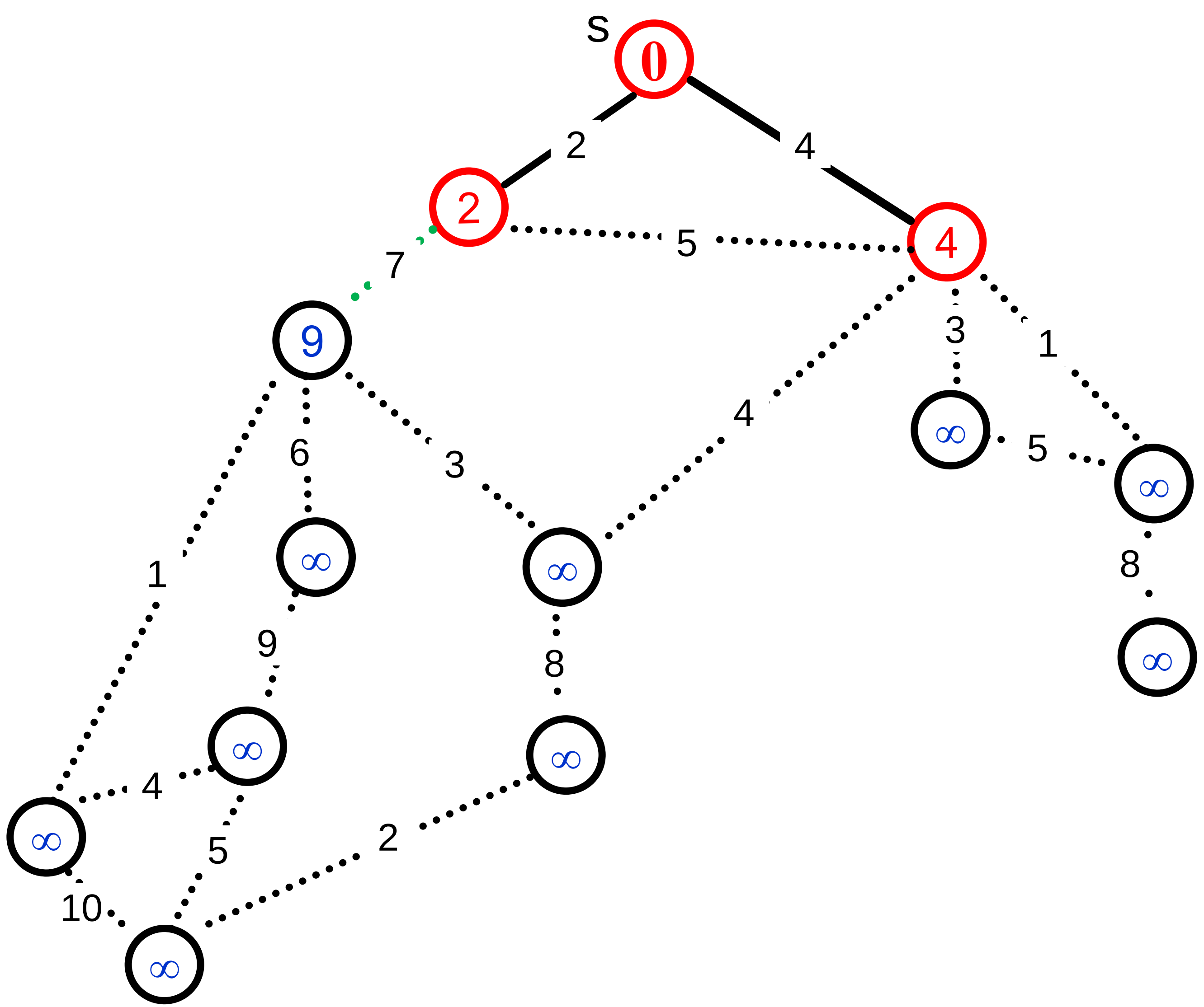
Dijkstra's Algorithm: Example



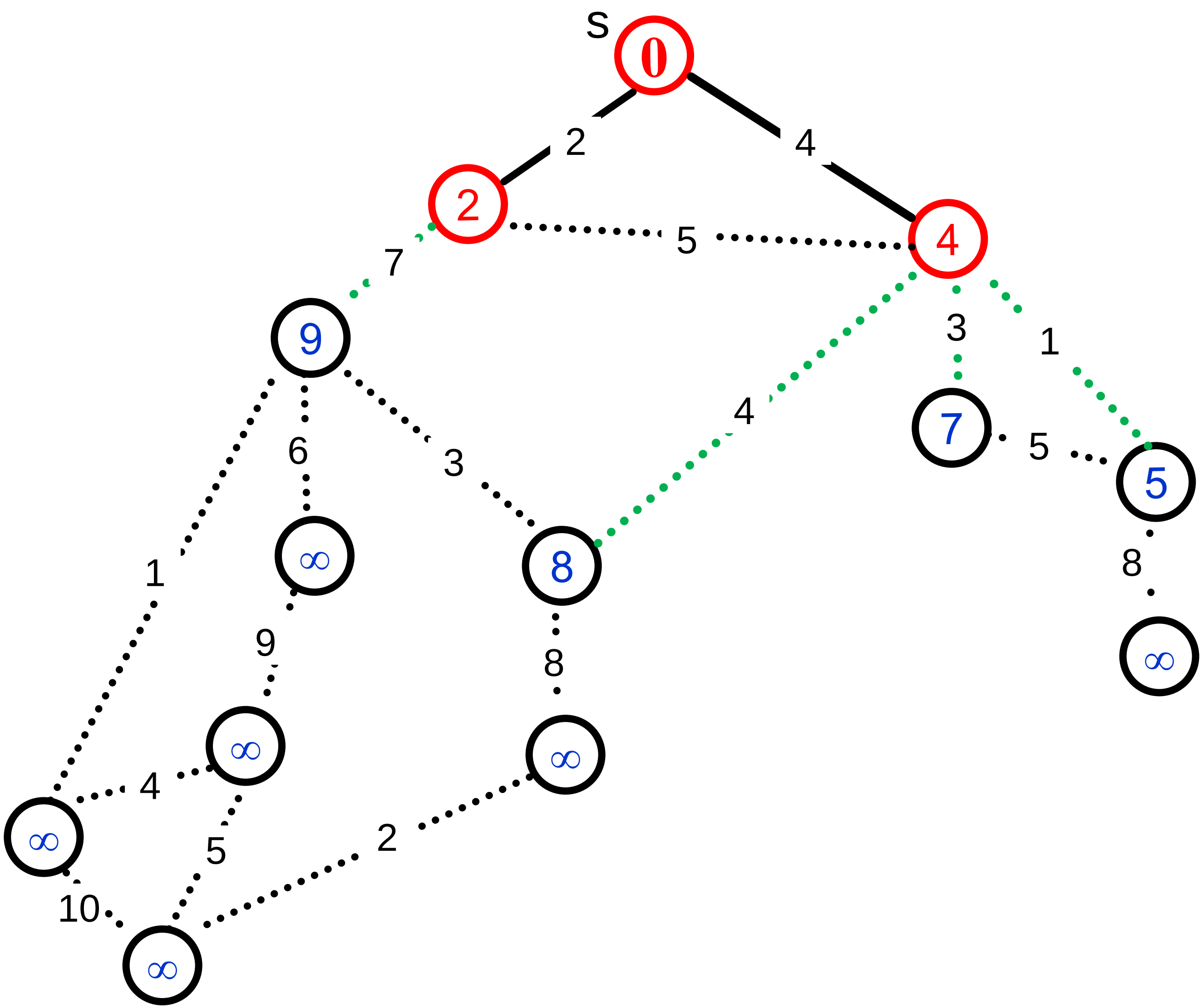
Dijkstra's Algorithm: Example



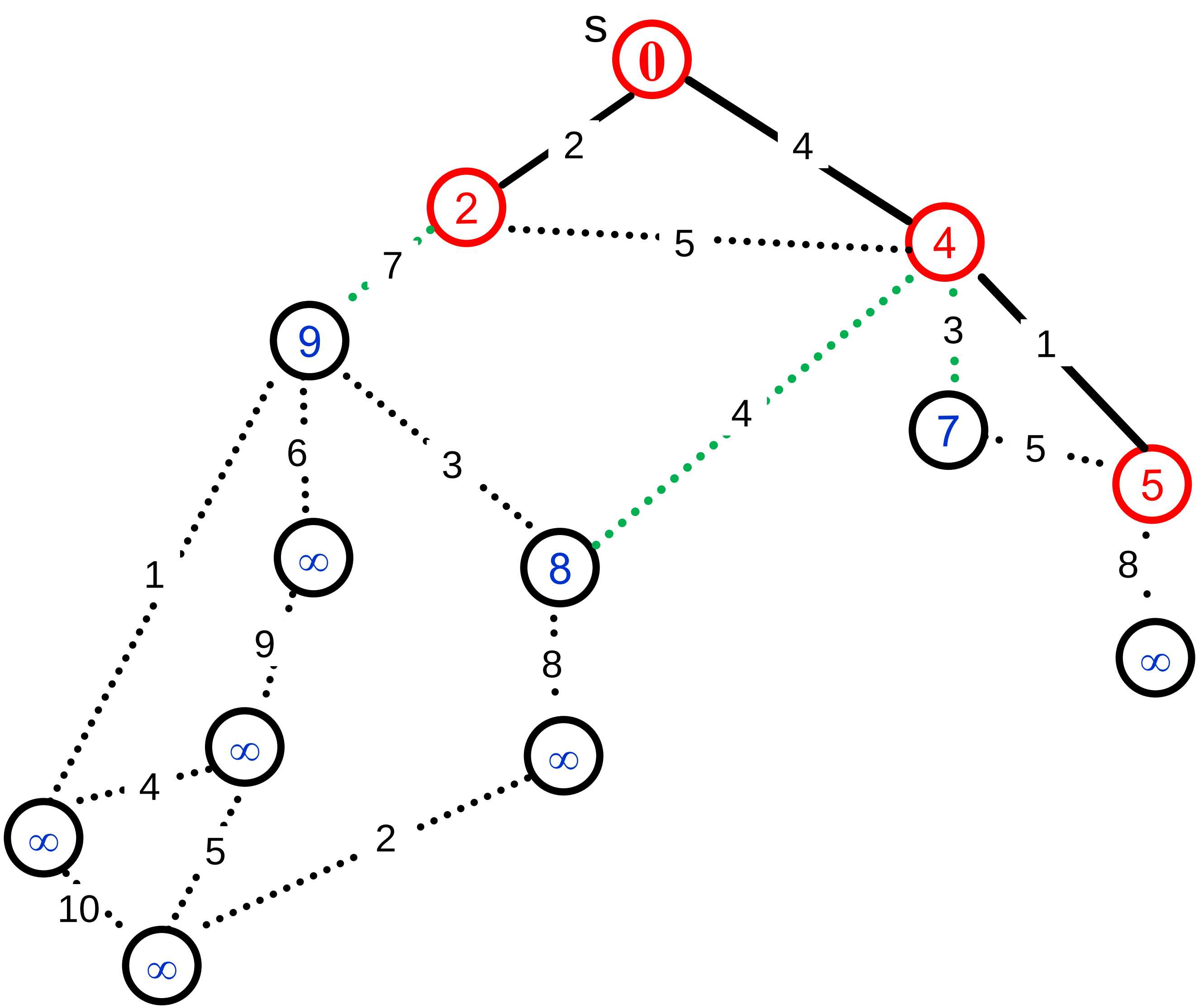
Dijkstra's Algorithm: Example



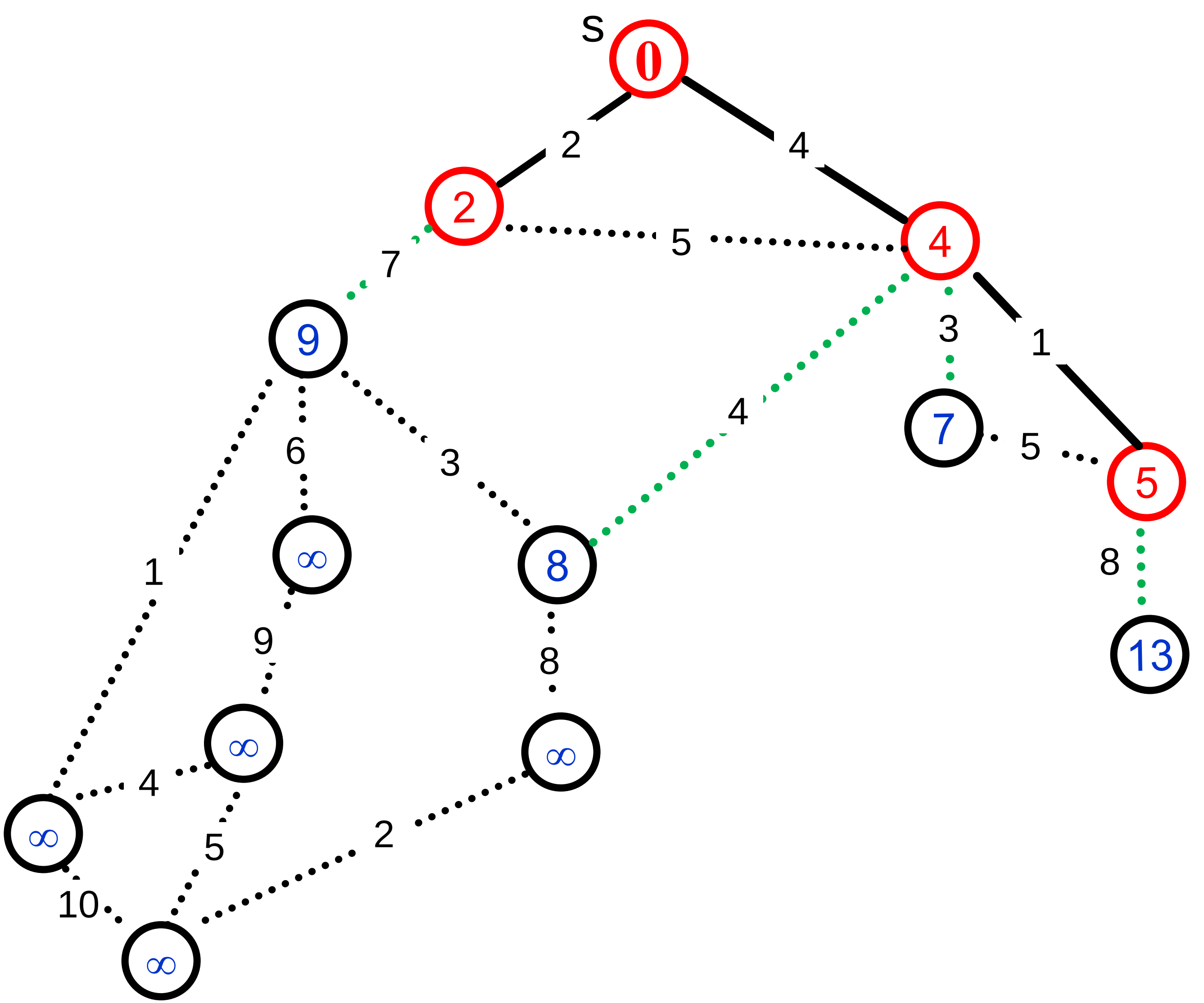
Dijkstra's Algorithm: Example



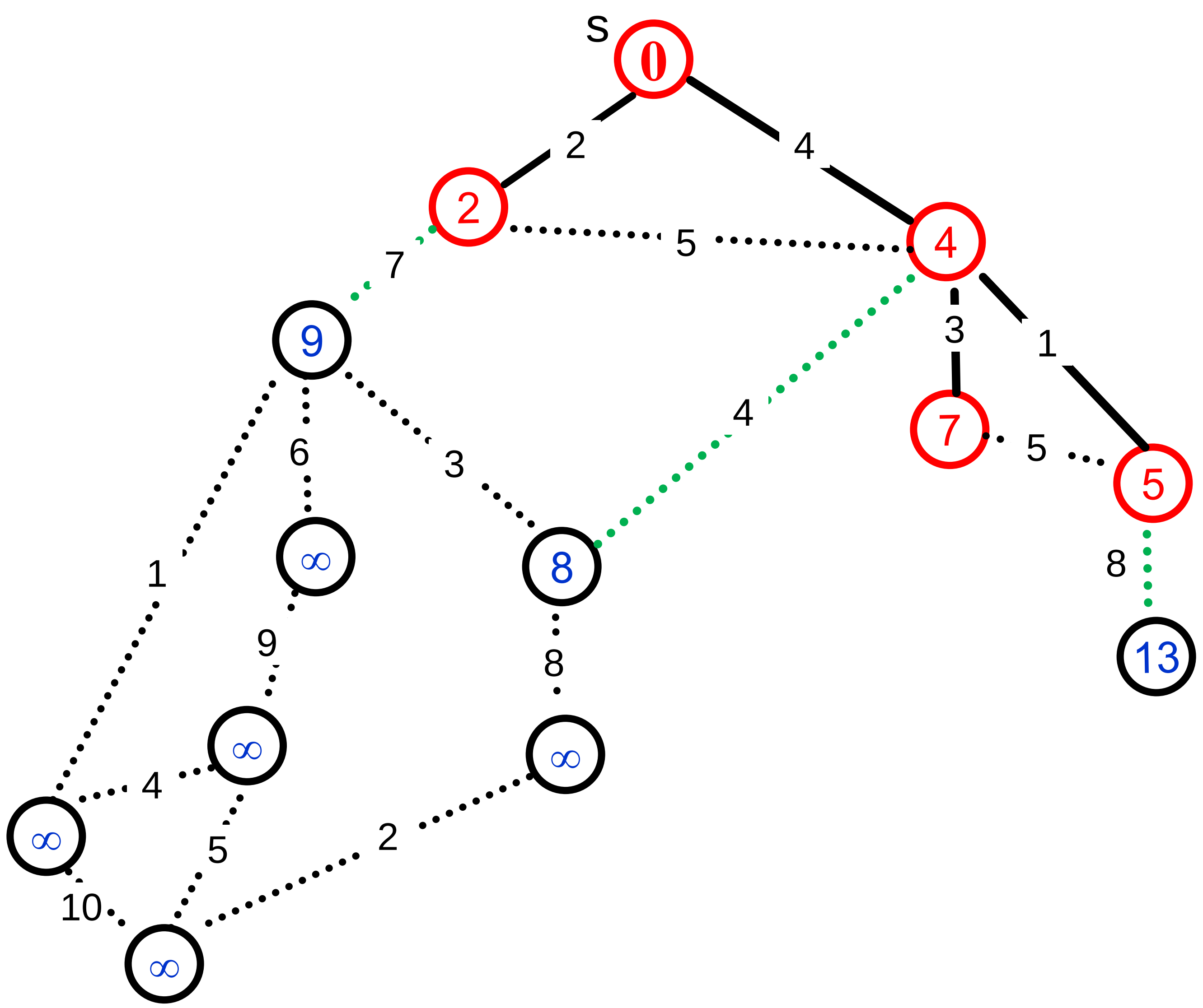
Dijkstra's Algorithm: Example



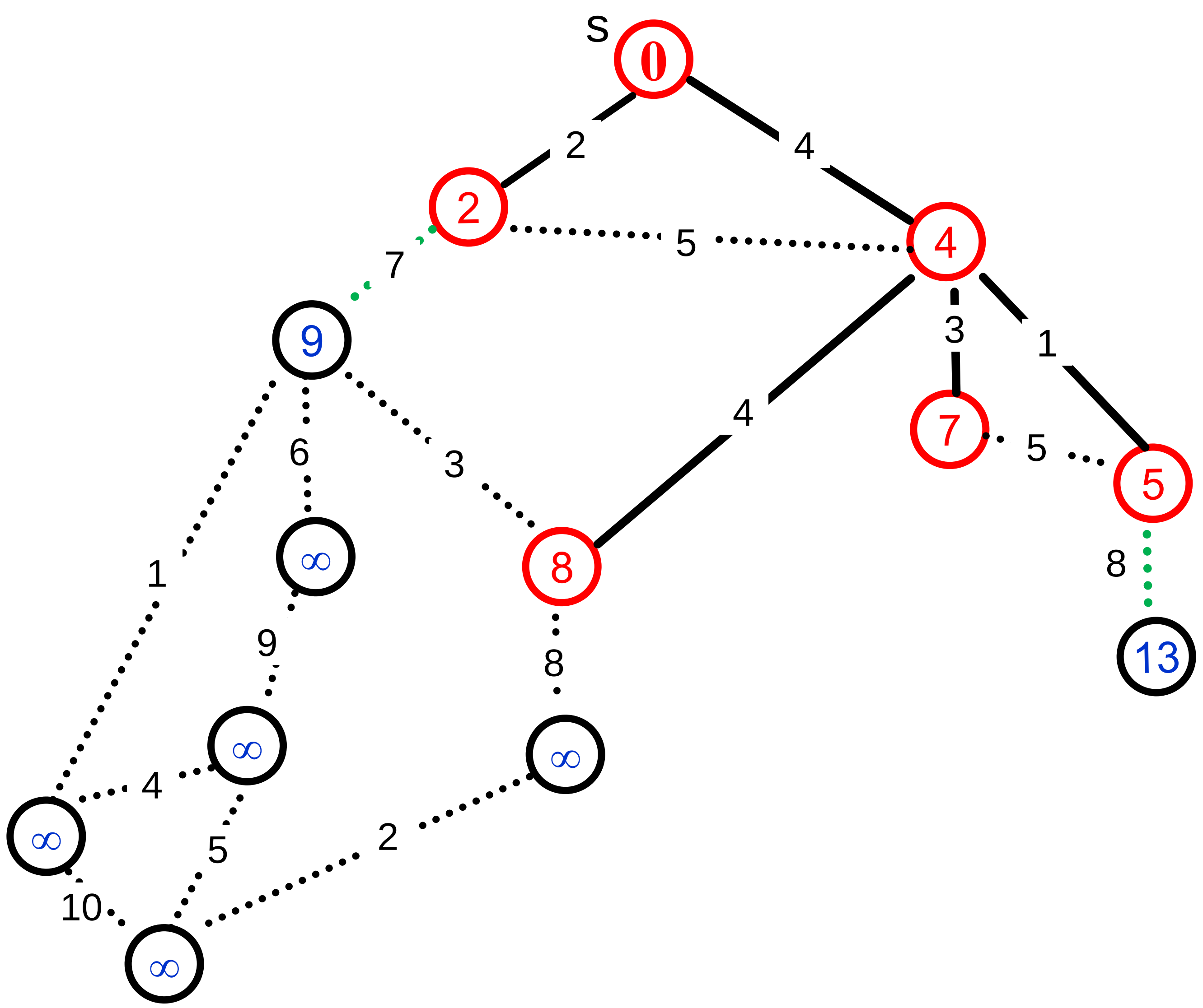
Dijkstra's Algorithm: Example



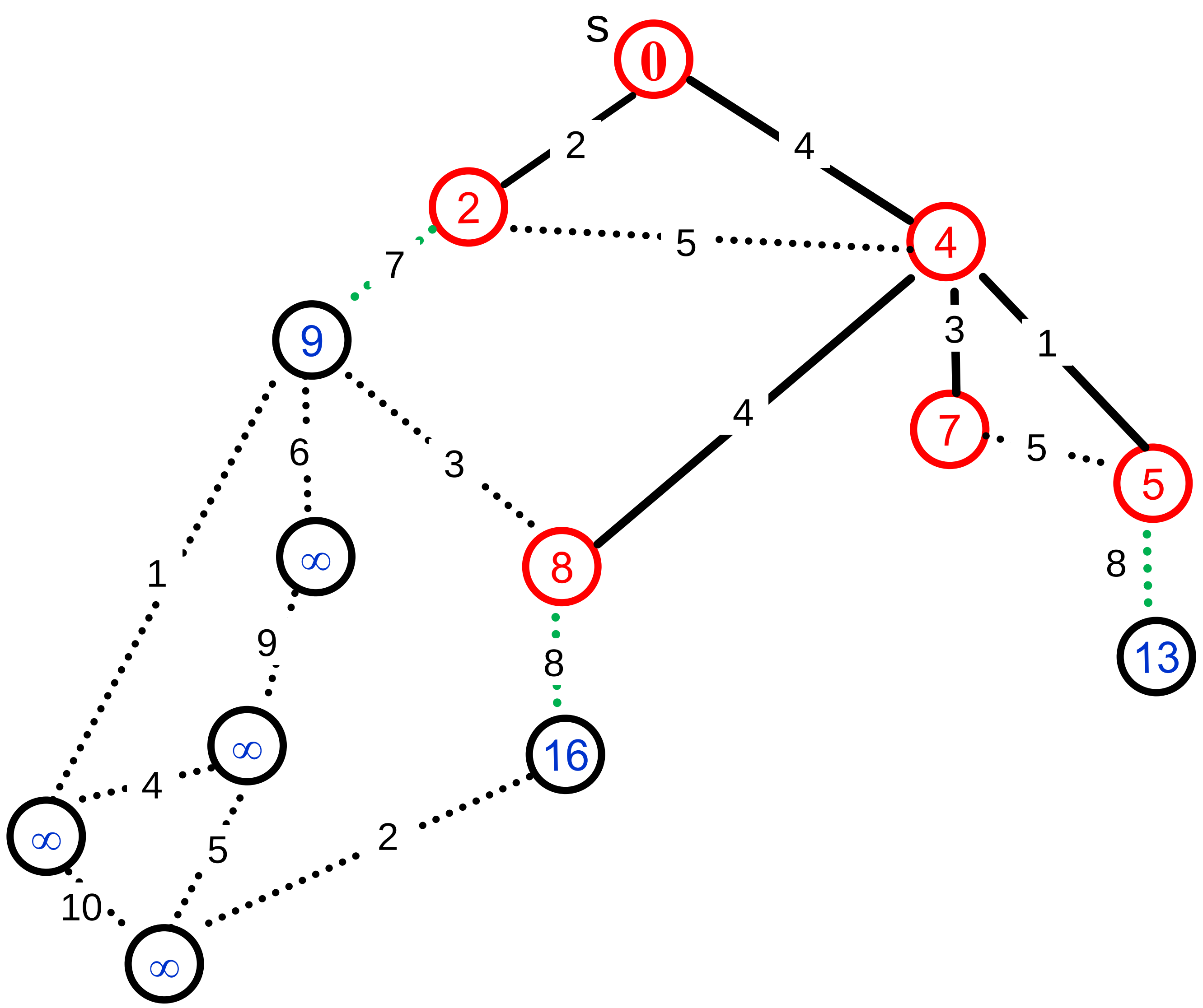
Dijkstra’s Algorithm: Example



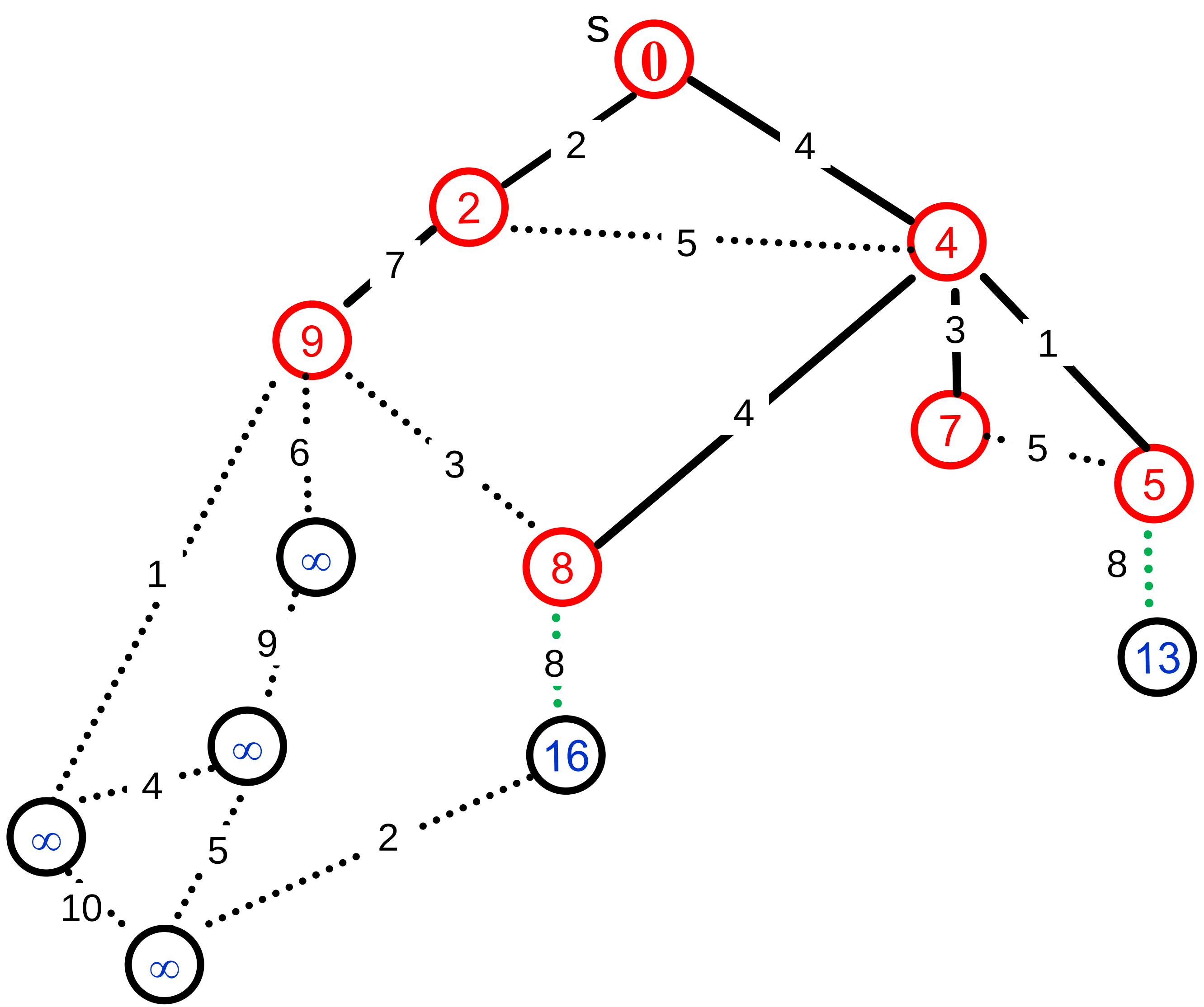
Dijkstra's Algorithm: Example



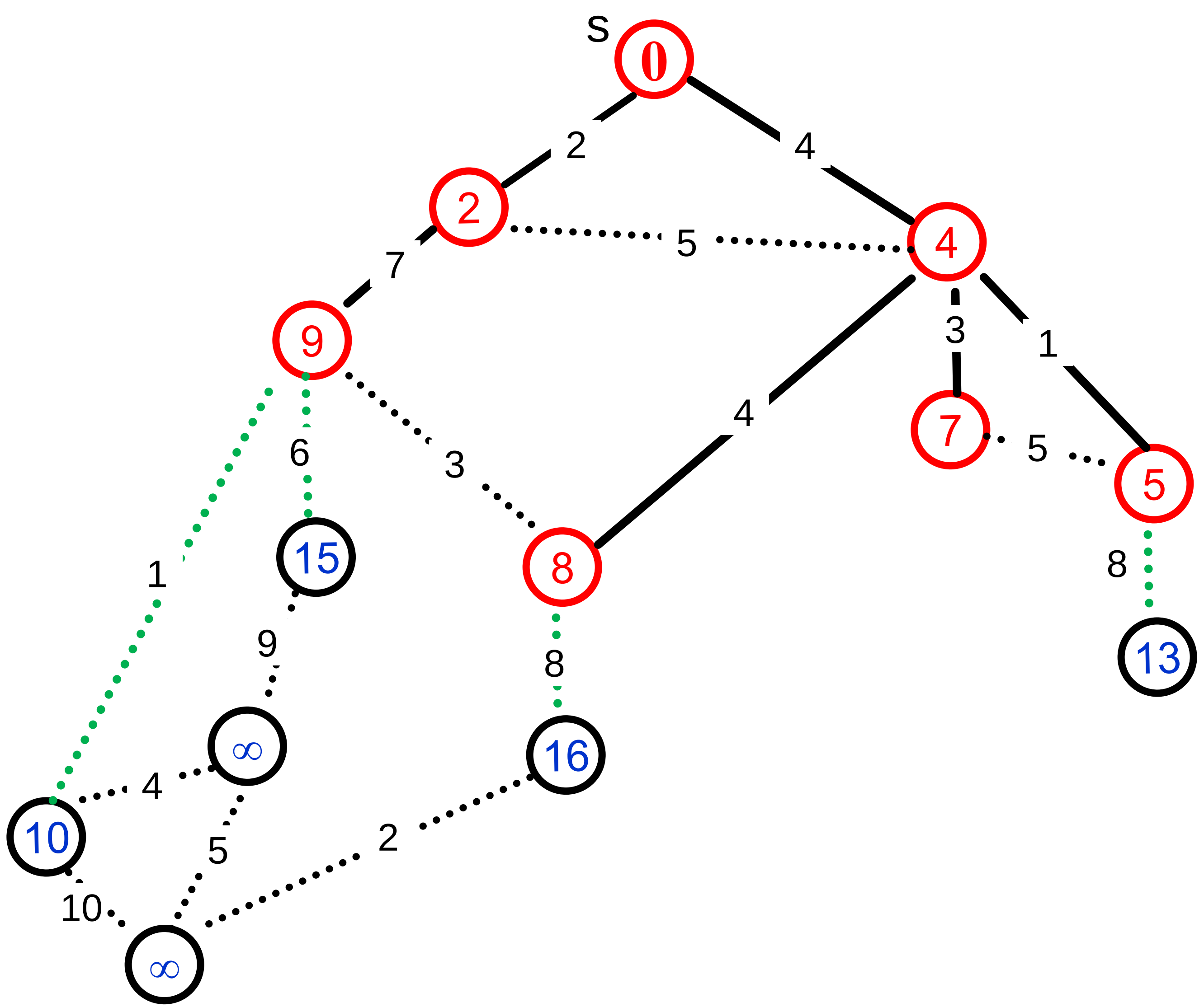
Dijkstra’s Algorithm: Example



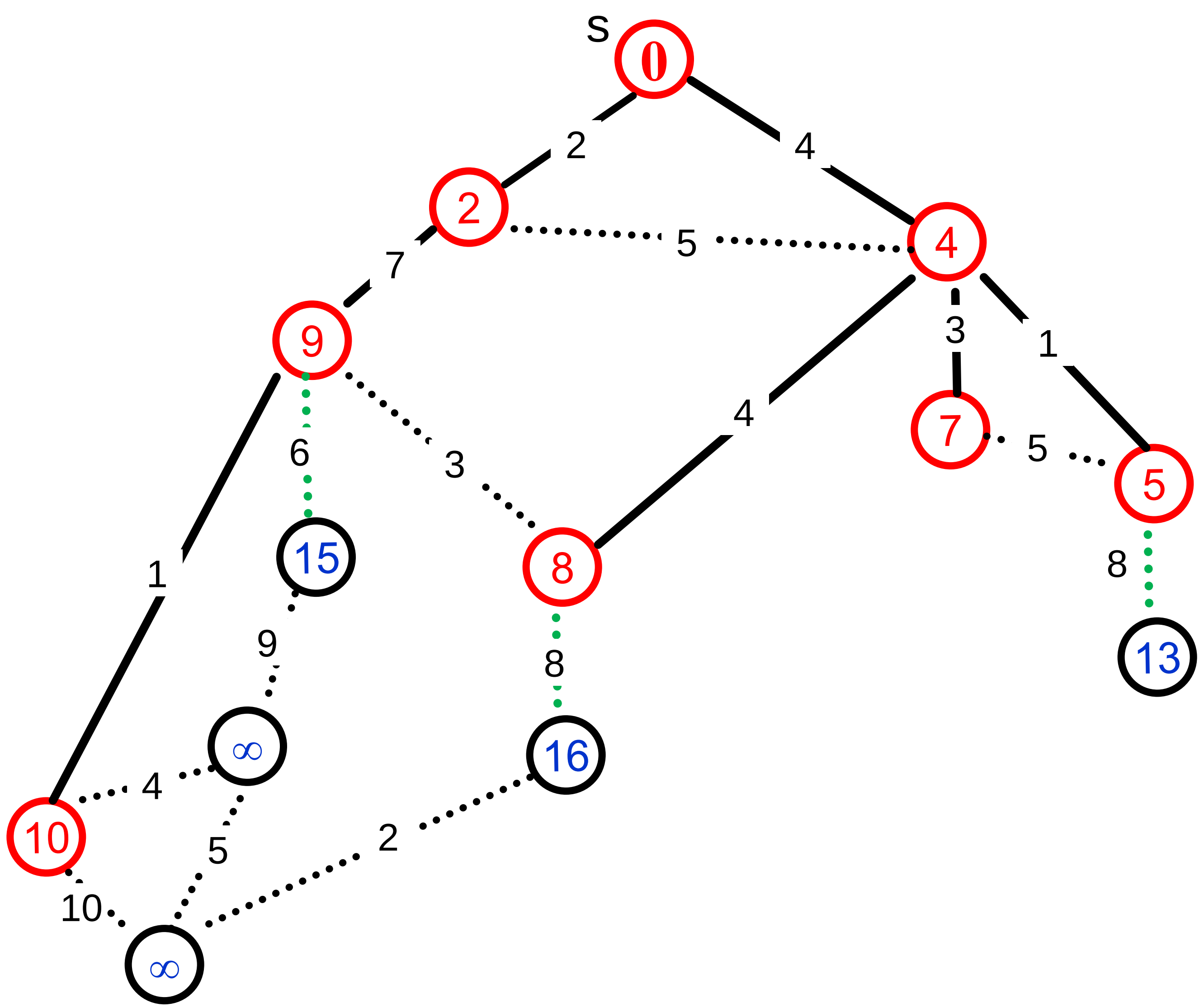
Dijkstra's Algorithm: Example



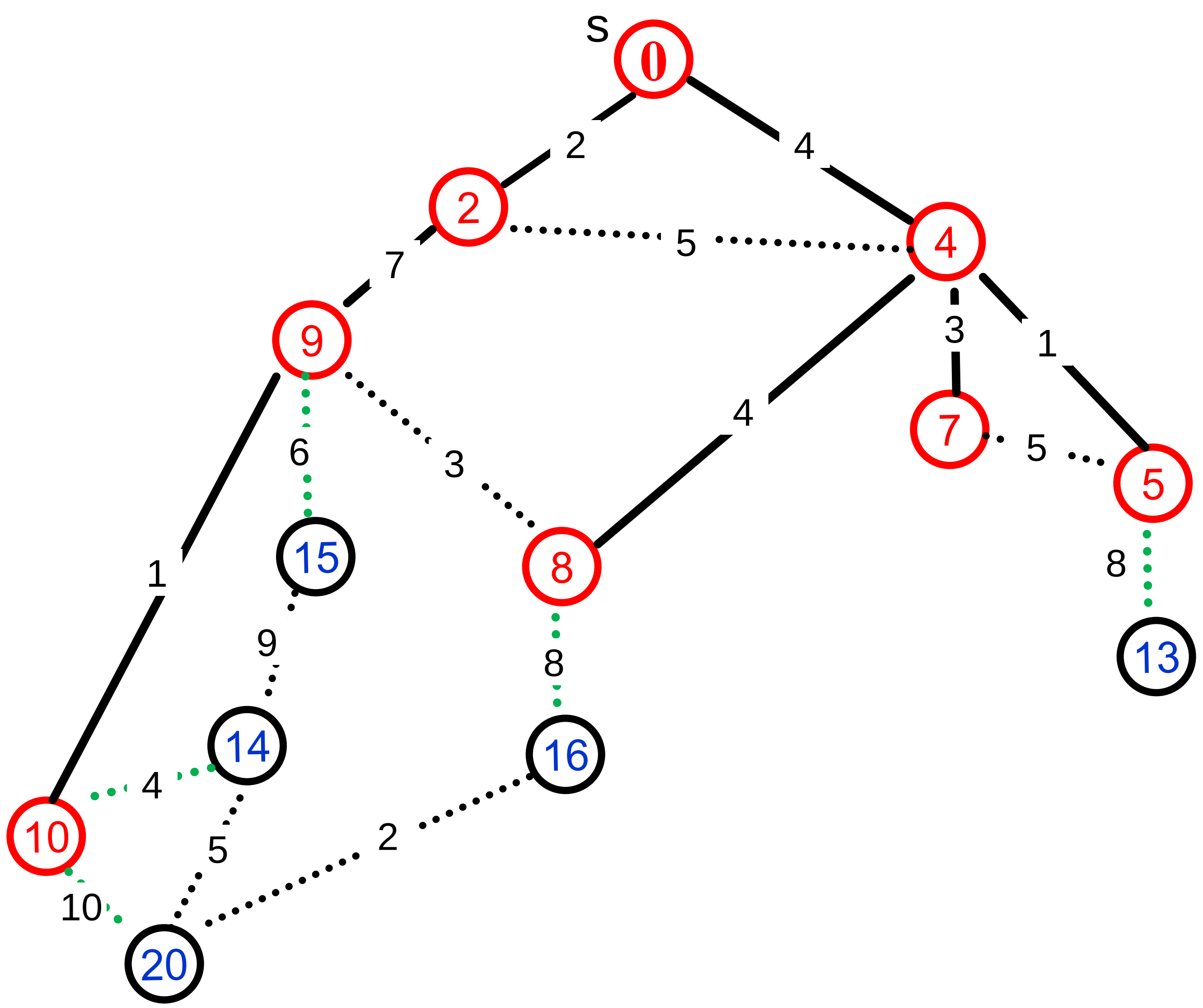
Dijkstra’s Algorithm: Example



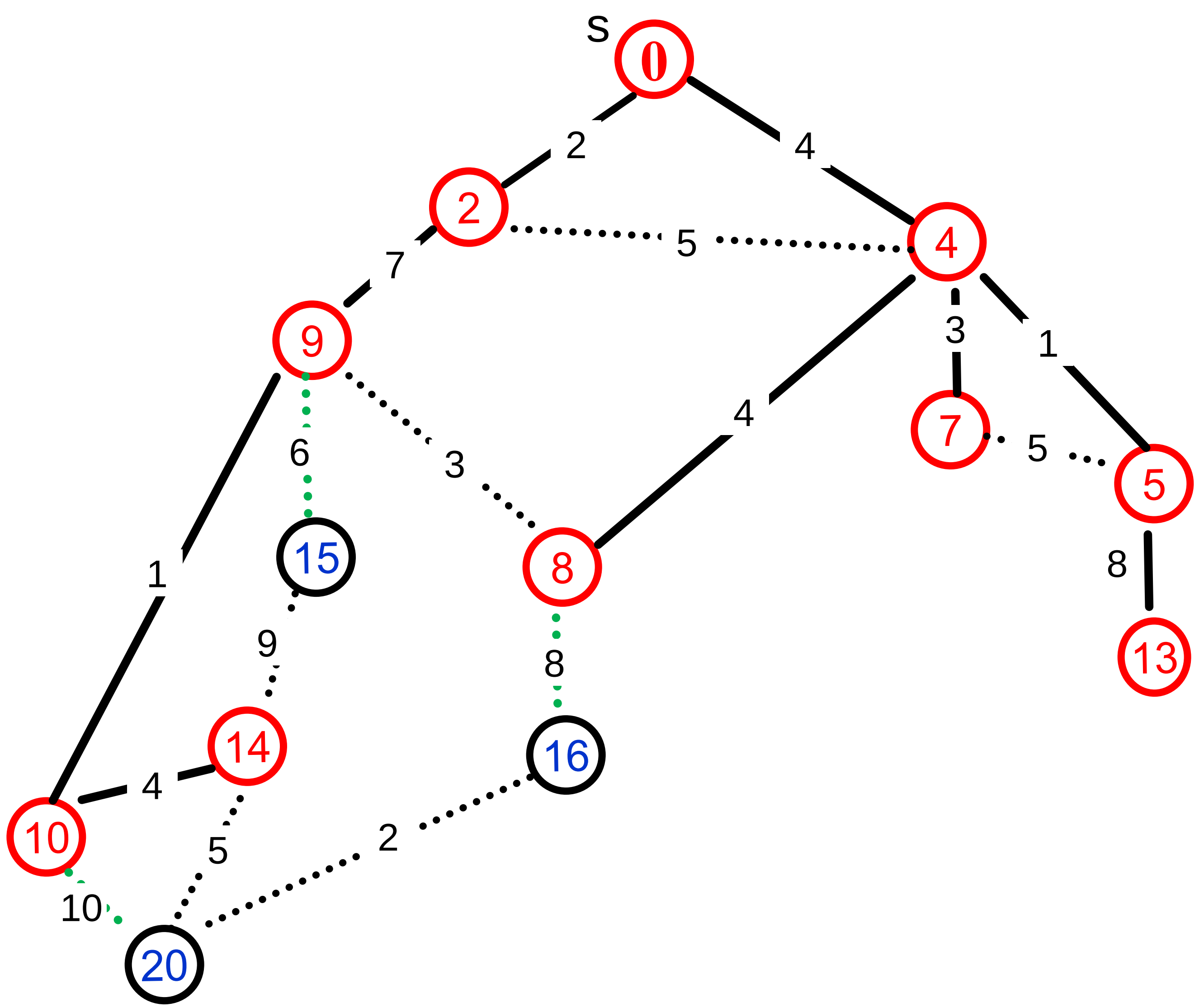
Dijkstra's Algorithm: Example



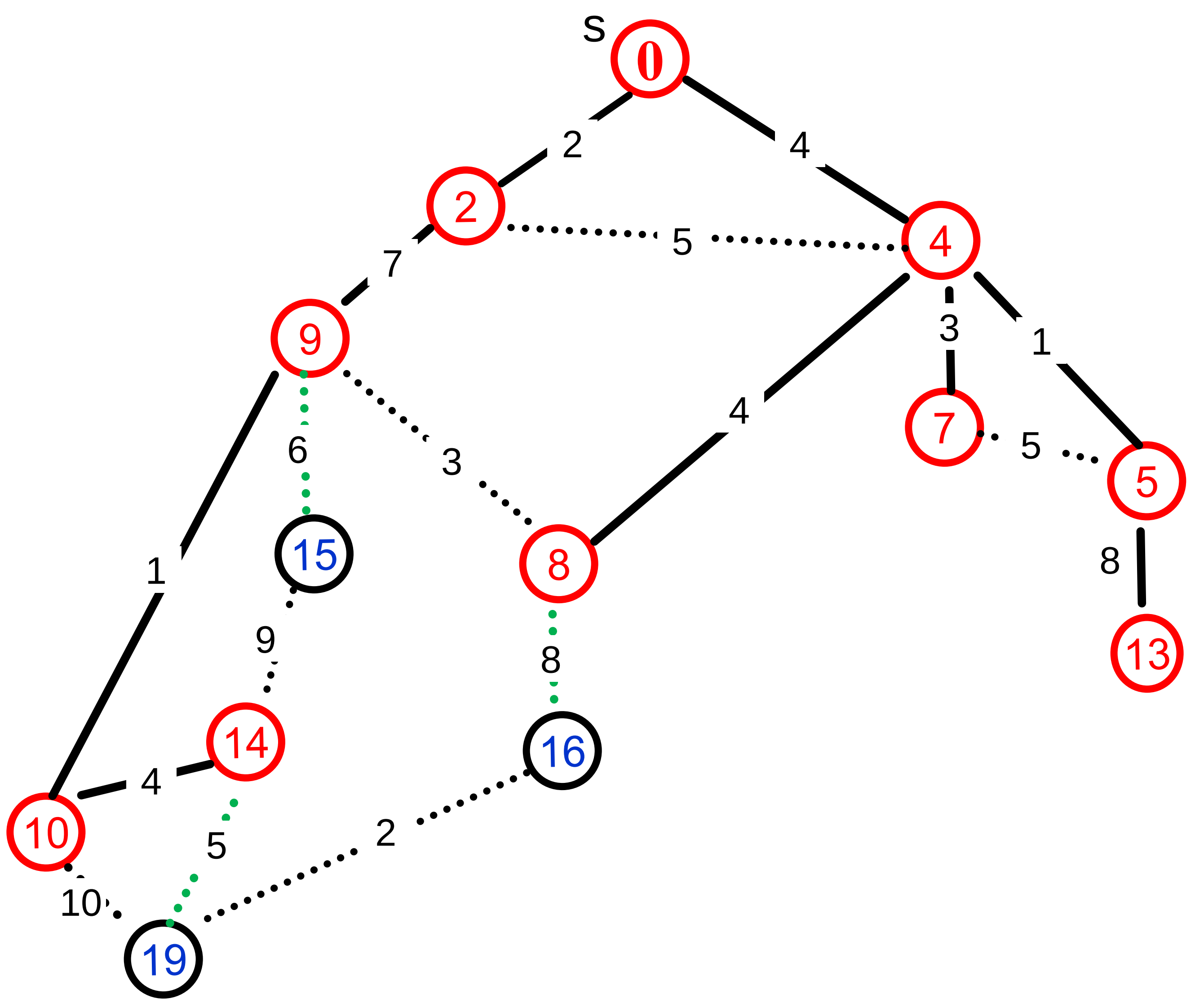
Dijkstra's Algorithm: Example



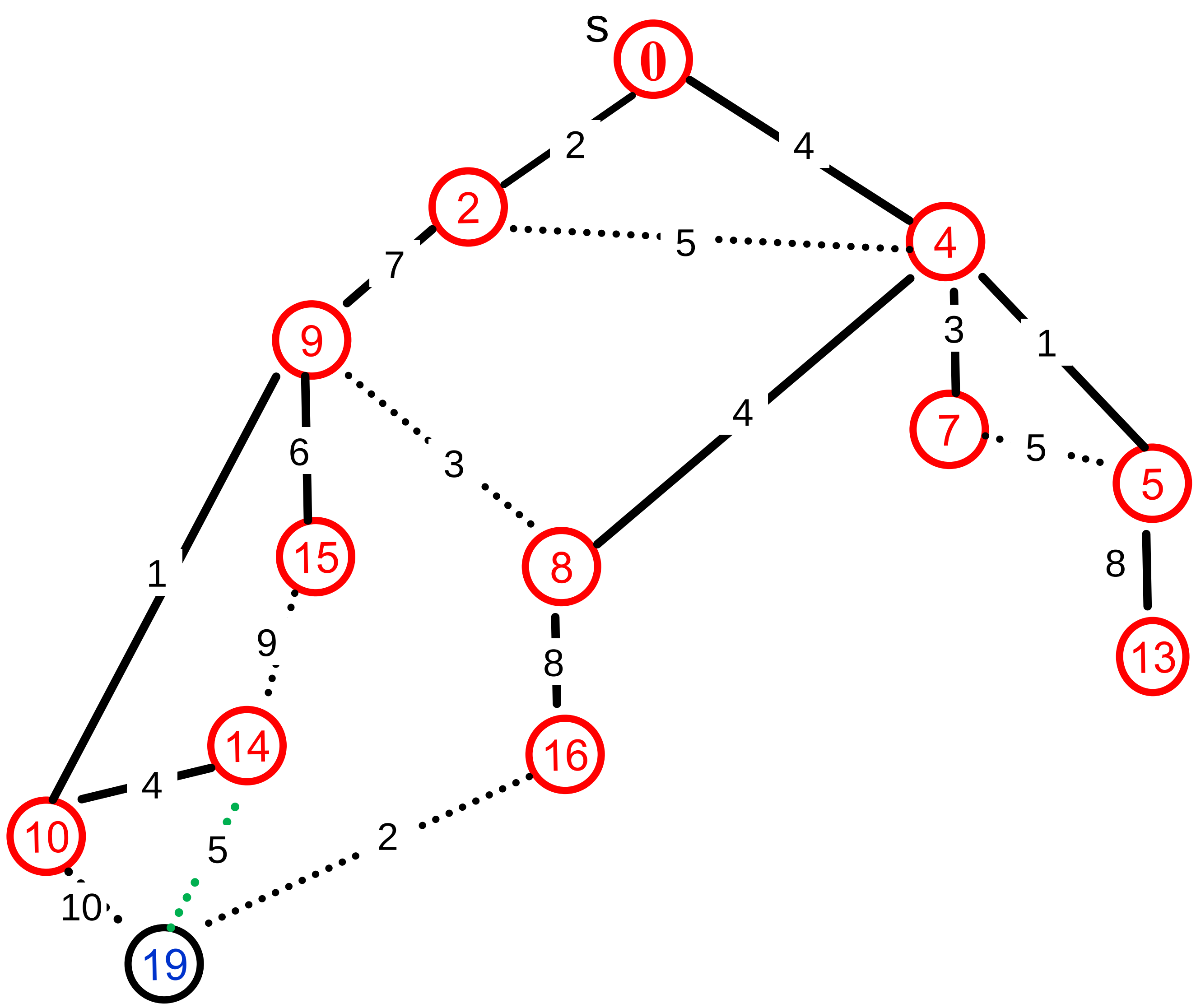
Dijkstra's Algorithm: Example



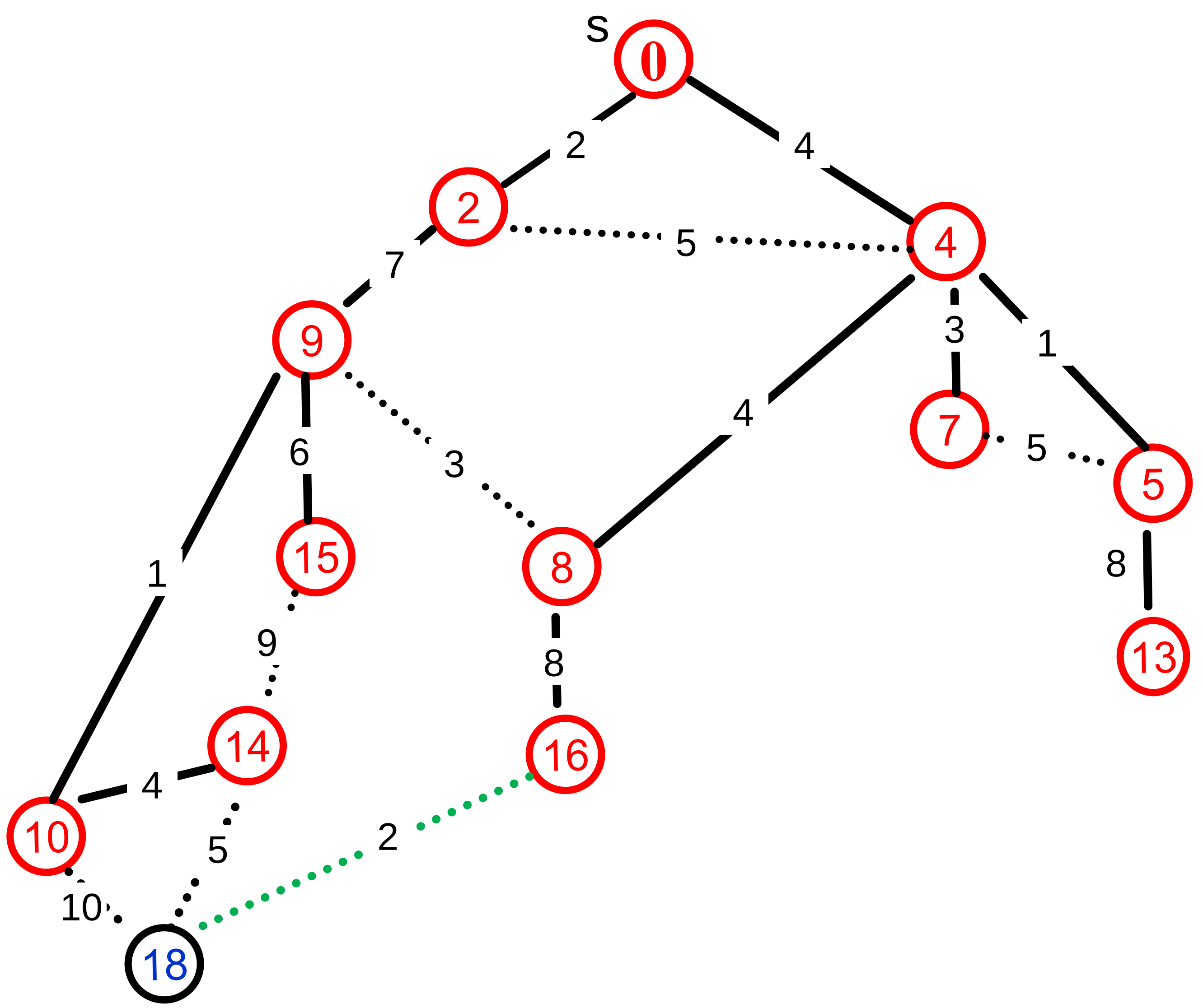
Dijkstra's Algorithm: Example



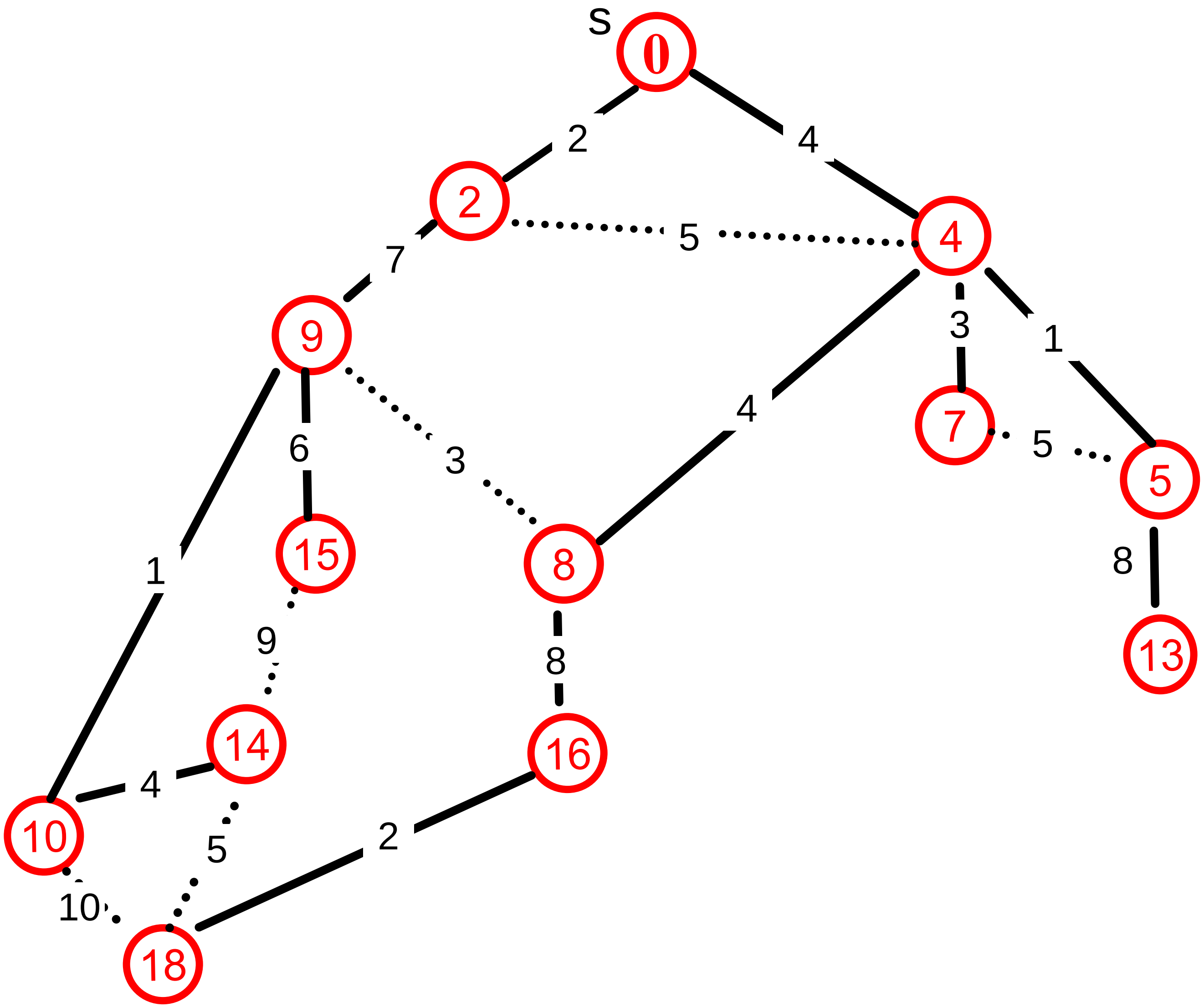
Dijkstra's Algorithm: Example



Dijkstra's Algorithm: Example



Dijkstra's Algorithm: Example



Disjkstra's Algorithm: Correctness

Prove by induction that throughout the algorithm, for any $u \in S$, the path P_u in the shortest from s to u.

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

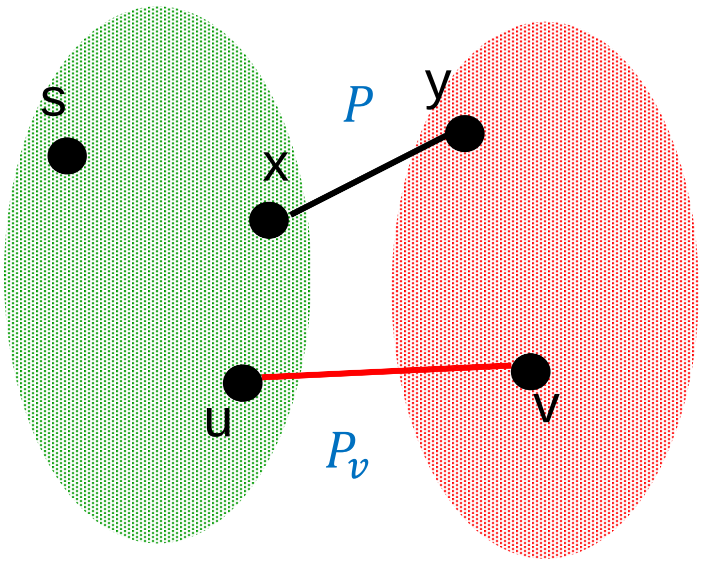
IH: Suppose $|S| = k$ and the claim holds for S

IS: 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 path P to S which is shorter. Consider the first time that P leaves S (with edge {x,y}).

S -> x has weight (at least) d(x)

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

A contradiction.



Remarks on Dijkstra’s Algorithm

- Algorithm also produces a tree of shortest paths to s following Parent links
- Algorithm works on directed graph (with nonnegative weights)

- The algorithm fails with negative edge weights.
  - e.g., some airline tickets

Why does it fail?

- Dijkstra’s algorithm is similar to BFS:
  - Subtitute 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

Dijkstra's Algorithm

Runs in O((n+m)log n).

```
Dijkstra(G, c, s) {
   d[s] ← 0
   foreach (v ∈ V) d[v] ← ∞ //This is the key of node v
   foreach (v ∈ V) insert v onto a priority queue Q
   Initialize set of explored nodes S ← {s}

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (d[u]+c_e < d[v]))
             d[v] ← d[u] + c_e
             Decrease key of v to d[v].
             Parent(v) ← u
}
```

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

$O(m)$ of decrease key, each runs in $O(\log n)$

Summary (Greedy Algorithms)

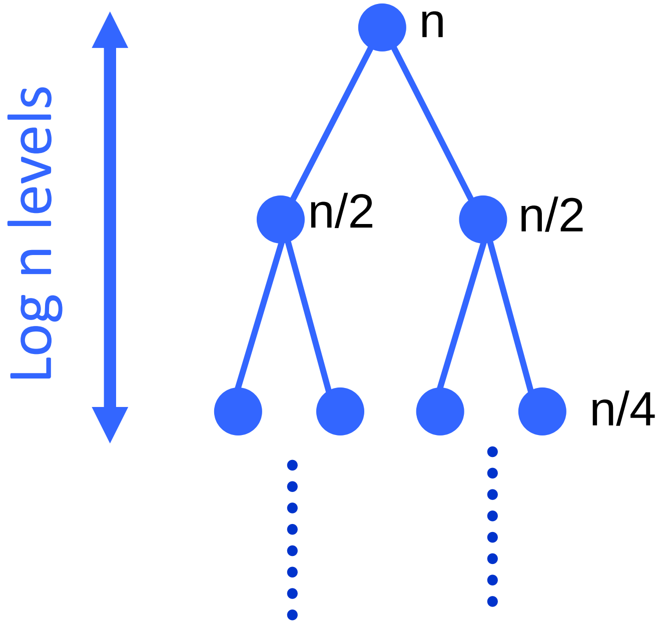
- Greedy Stays Ahead: Interval Scheduling, Dijkstra's algorithm
- Structural: Interval Partitioning
- Exchange Arguments: MST, Kruskal's Algorithm, Prim's Algorithm
- Data Structures: Union Find, Heap

Divide and Conquer Approach

Divide and Conquer

Similar to algorithm design by induction, we reduce a problem to several subproblems.

Typically, each sub-problem is at most a constant fraction of the size of the original problem



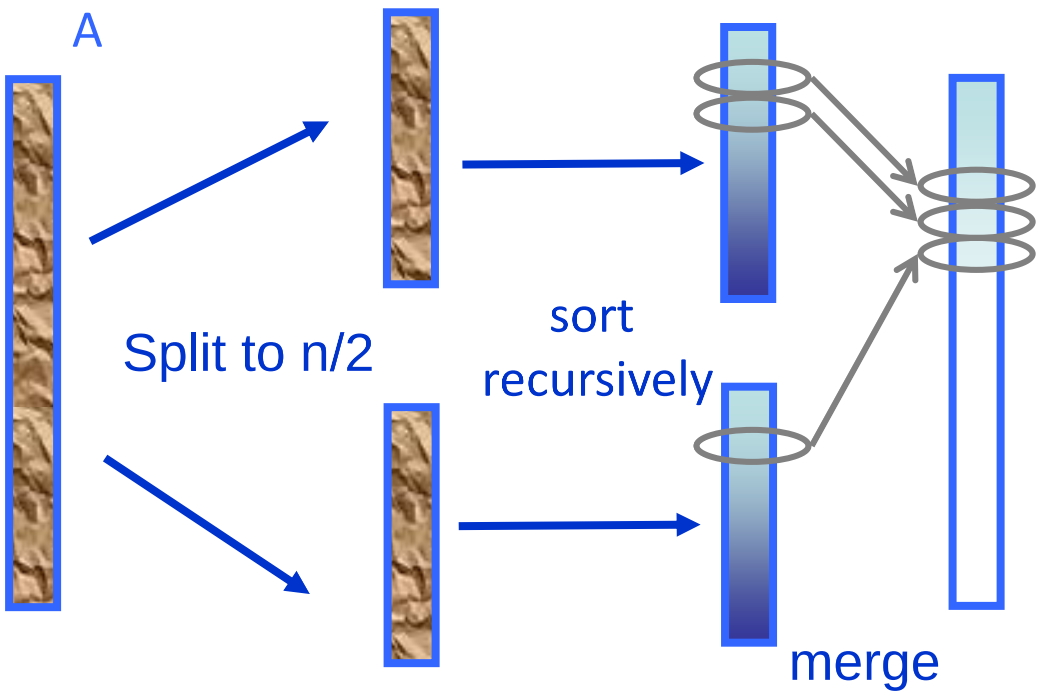
Recursively solve each subproblem

Merge the solutions

Examples:

- Mergesort, Binary Search, Strassen's Algorithm,

A Classical Example: Merge Sort



Why Balanced Partitioning?

An alternative "divide & conquer" algorithm:

- Split into n-1 and 1
- Sort each sub problem
- Merge them

Runtime

$$T(n) = T(n-1) + T(1) + n$$

Solution:

$$T(n) = n + T(n-1) + T(1)$$

$$= n + n - 1 + T(n-2)$$

$$= n + n - 1 + n - 2 + T(n-3)$$

$$= n + n - 1 + n - 2 + \cdots + 1 = O(n^2)$$

D&C: The Key Idea

Suppose we've already invented Bubble-Sort, and we know it takes n^2

Try just one level of divide & conquer:

Bubble-Sort(first n/2 elements)

Bubble-Sort(last n/2 elements)

Merge results

Time: $2\,(n/2)2 + n = n^2/2 + n \ll n^2$

Almost twice as fast!

D&C in a nutshell

D&C approach

- "the more dividing and conquering, the better"
  - Two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
  - Best is usually full recursion down to a small constant size (balancing "work" vs "overhead").

  In the limit: you've just rediscovered mergesort!
- Even unbalanced partitioning is good, but less good
  - Bubble-sort improved with a 0.1/0.9 split:

    $$(.1n)2 + (.9n)2 + n = .82n^2 + n$$

    The 18% savings compounds significantly if you carry recursion to more levels, actually giving $O(n \log n)$, but with a bigger constant.
- This is why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.

Finding the Root of a Function

Finding the Root of a Function

Given a continuous function f and two points a < b such that

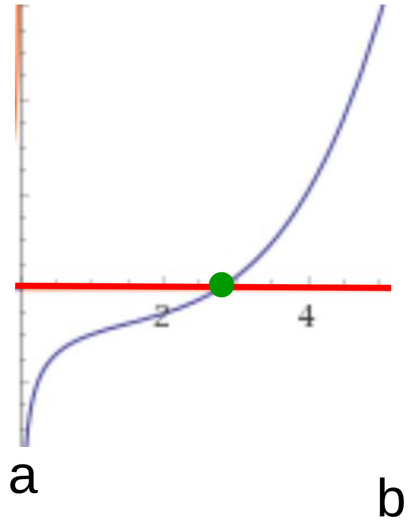
$$f(a) \leq 0$$
$$f(b) \geq 0$$

Find an approximate root of f (a point c where $f(c) = 0$).

f has a root in $[a, b]$ by intermediate value theorem

Note that roots of f may be irrational,
So, we want to approximate
the root with an arbitrary precision!

$$f(x) = \sin(x) - \frac{100}{\sqrt{x}} + x^4$$



A Naiive Approch

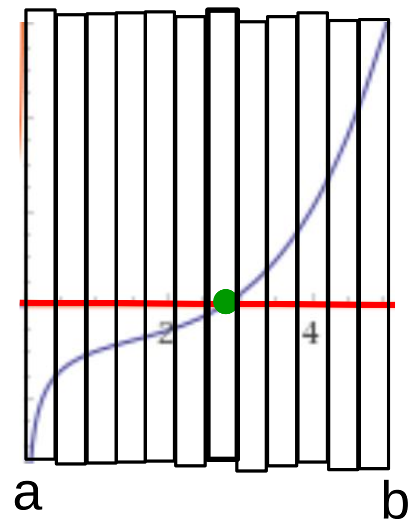
Suppose we want ϵ approximation to a root.

Divide [a,b] into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check

$$f(x) \leq 0, f(x+\epsilon) \geq 0$$

This runs in time $O(n) = O(\frac{b-a}{\epsilon})$

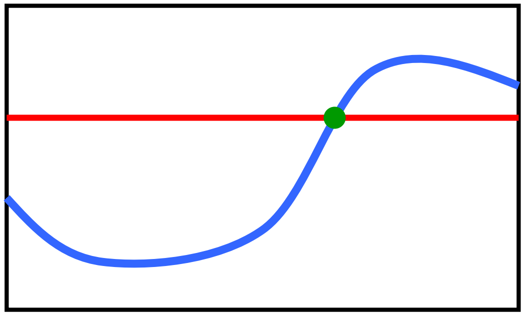
Can we do faster?

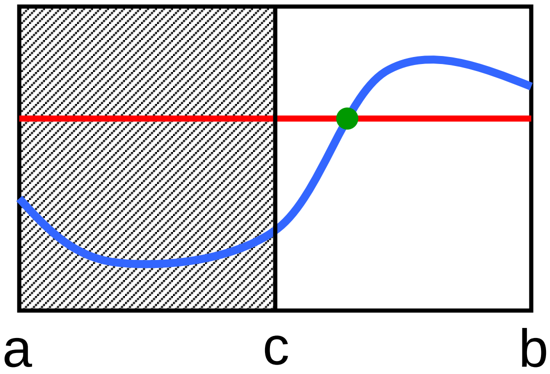


D&C Approach (Based on Binary Search)

Bisection(a,b, $\boldsymbol{\varepsilon}$)

- if $(b - a) < \boldsymbol{\epsilon}$ then
  - return (a)
- else
  - $m \leftarrow (a + b)/2$
  - if $f(m) \leq 0$ then
    - return(Bisection(c, b, ε))
  - else
    - return(Bisection(a, c, ε))



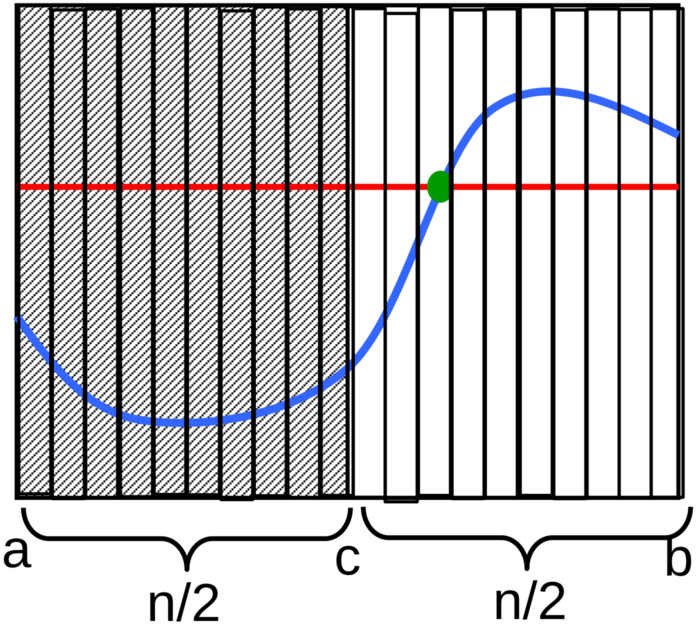


Time Analysis

Let $n = \frac{a-b}{\epsilon}$

And $c = (a + b)/2$

Always half of the intervals lie to the left and half lie to the right of c



So,

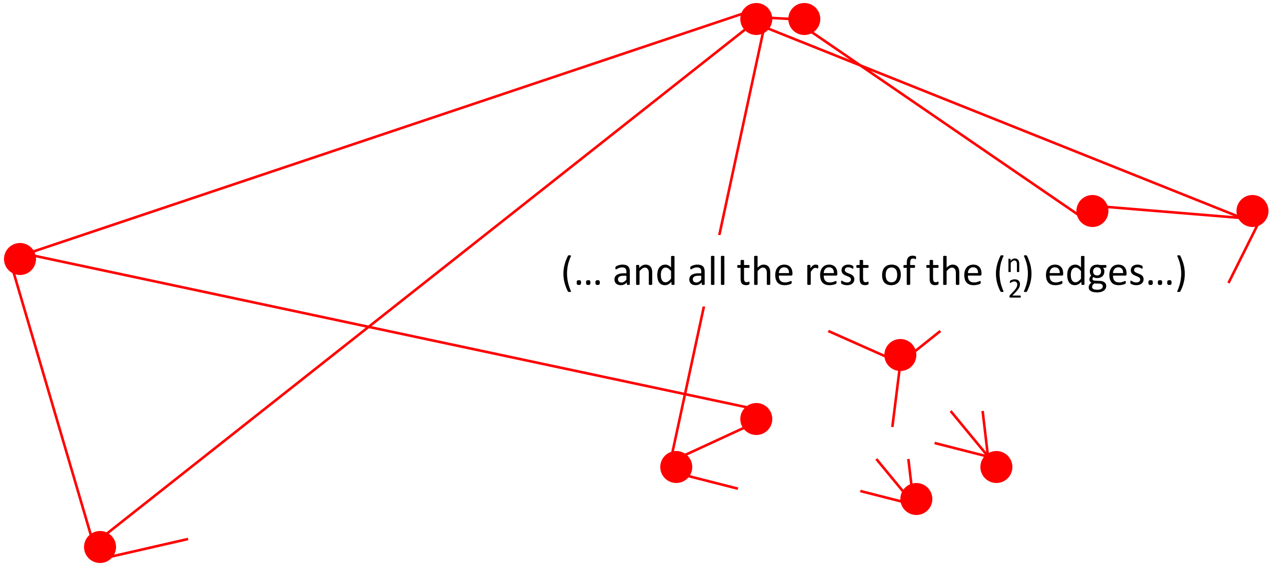
$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

i.e., $T(n) = O(\log n) = O(\log \frac{a-b}{\epsilon})$

Finding the Closest Pair of Points

Closest Pair of Points (non geometric)

Given n points and arbitrary distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)

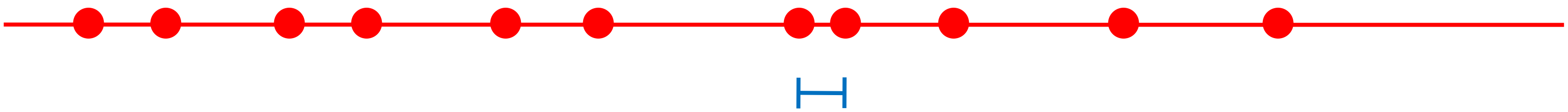


Must look at all n choose 2 pairwise distances, else any one you didn't check might be the shortest.

i.e., you have to read the whole input

Closest Pair of Points (1-dimension)

Given n points on the real line, find the closest pair



Fact: Closest pair is adjacent in ordered list

So, first sort, then scan adjacent pairs.

Time O(n log n) to sort, if needed, Plus O(n) to scan adjacent pairs

Key point: do *not* need to calc distances between all pairs: exploit geometry + ordering

Closest Pair of Points (2-dimensions)

Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force: Check all pairs of points p and q with $\Theta(n^2)$ time.

Assumption: No two points have same x coordinate.

Closest Pair of Points (2-dimensions)

Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force: Check all pairs of points p and q with $\Theta(n^2)$ time.

Assumption: No two points have same x coordinate.

A Divide and Conquer Alg

Divide: draw vertical line L with ≈ n/2 points on each side.

Conquer: find closest pair on each side, recursively.

Combine to find closest pair overall

Return best solutions

seems like $\Theta(n^2)$?