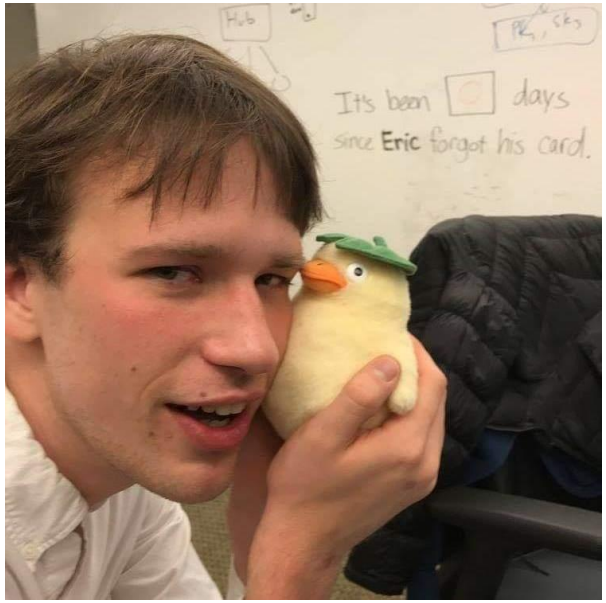
CSE 417 Algorithms

Winter 2020
Lecture 10
Dijkstra's algorithm

About me

Philip Garrison (he/him)

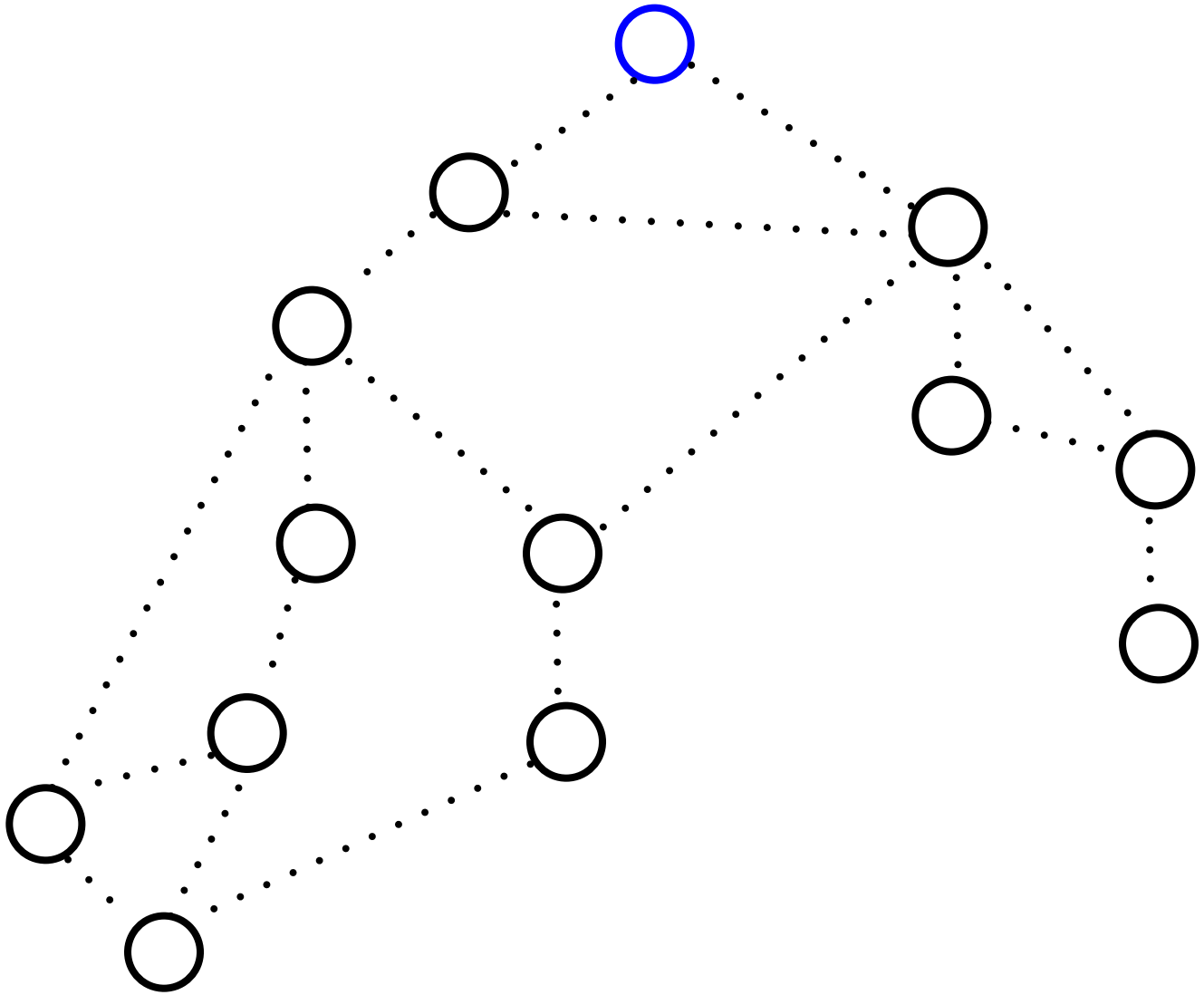
- CSE PhD student
- advised by Richard
- introducing CSE first years to proofs
- musician



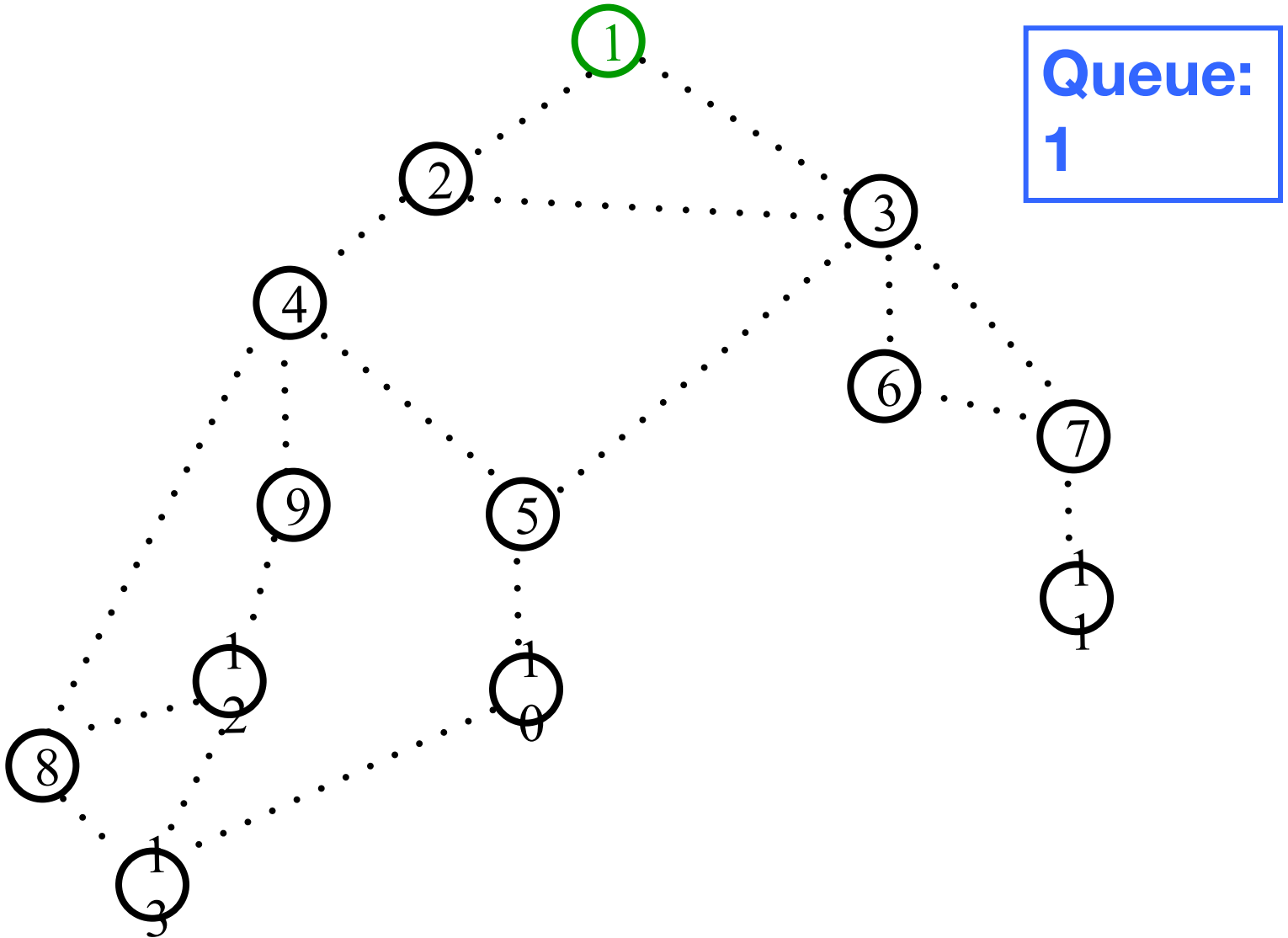
Upcoming lectures

- Topics
  - Dijkstra's Algorithm (Section 4.4)
  - Friday: Shortest Paths / Minimum Spanning Trees
  - Monday: Minimum Spanning Trees
- Reading
  - 4.4, 4.5, 4.7, 4.8

Breadth-first search (BFS)

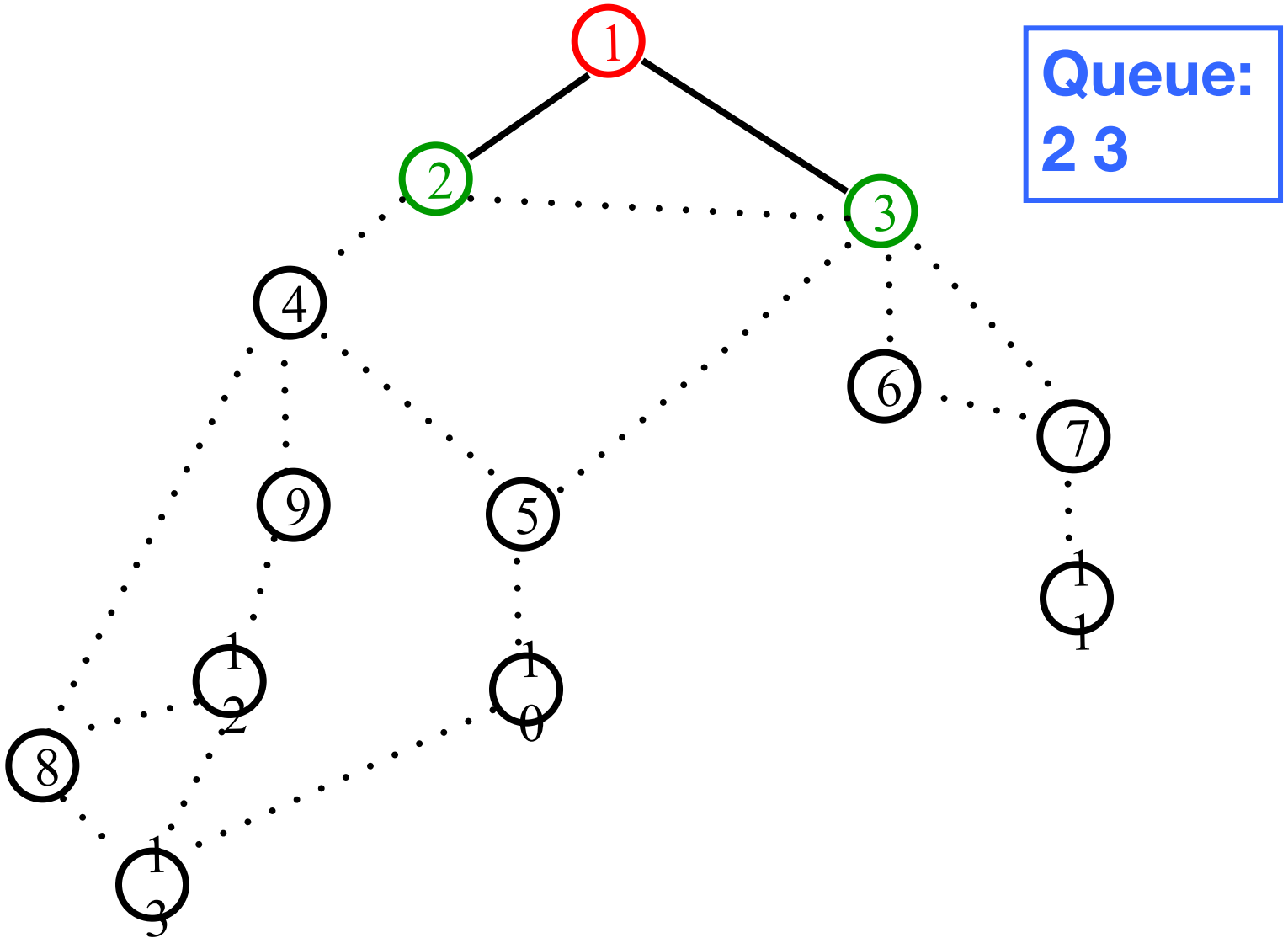


Breadth-first search (BFS)



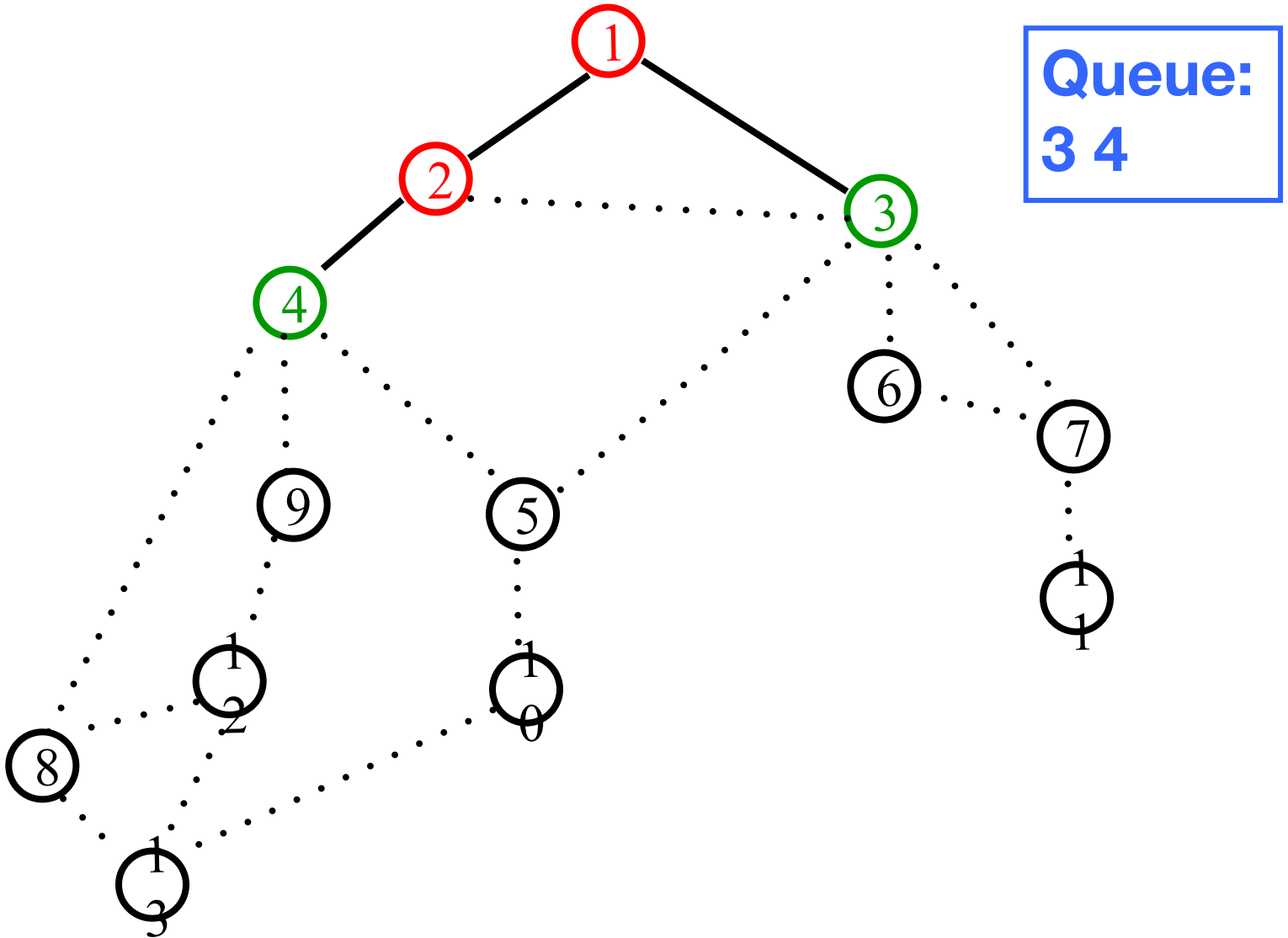
Queue:
1

Breadth-first search (BFS)



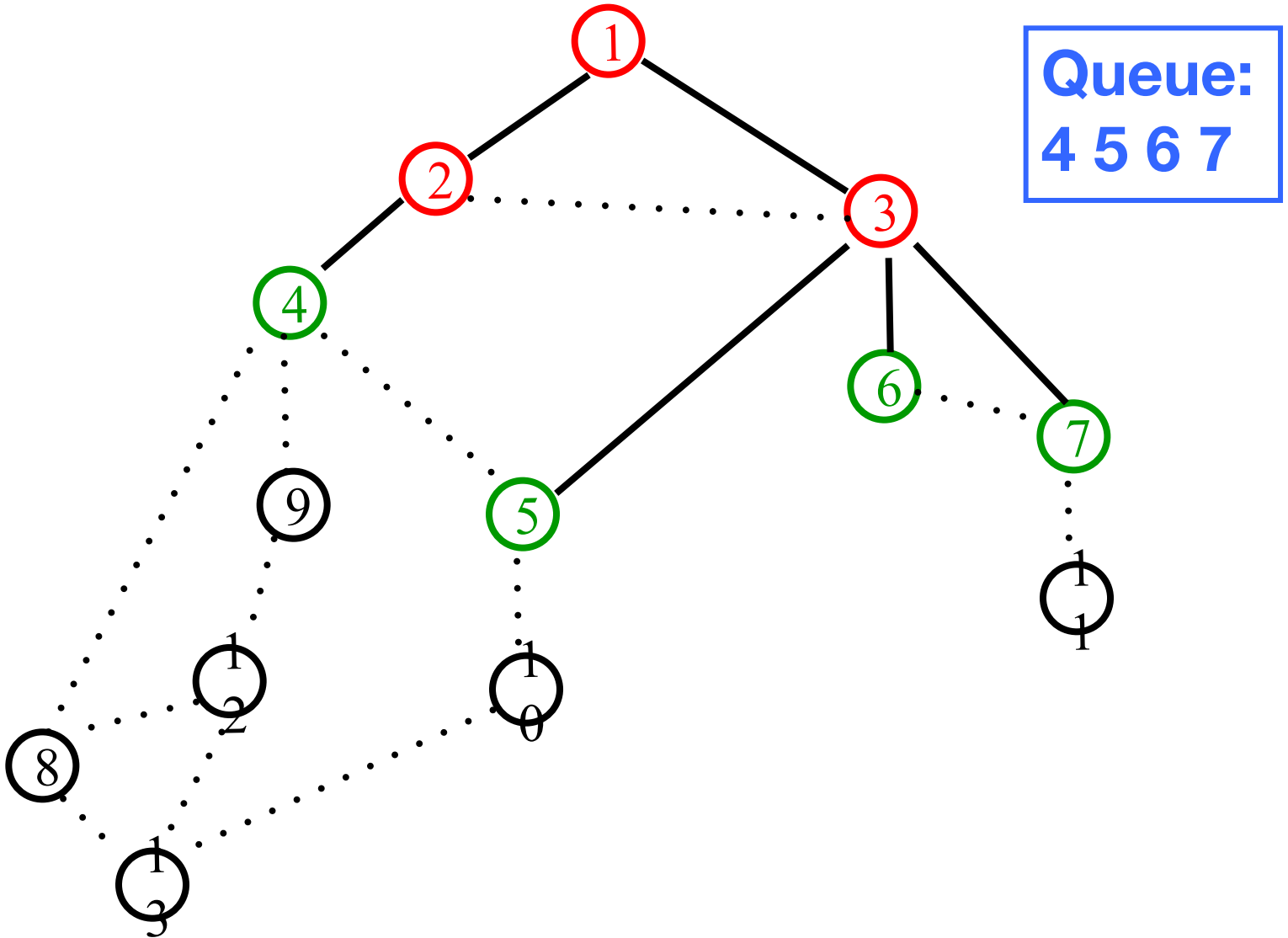
Queue:
2 3

Breadth-first search (BFS)

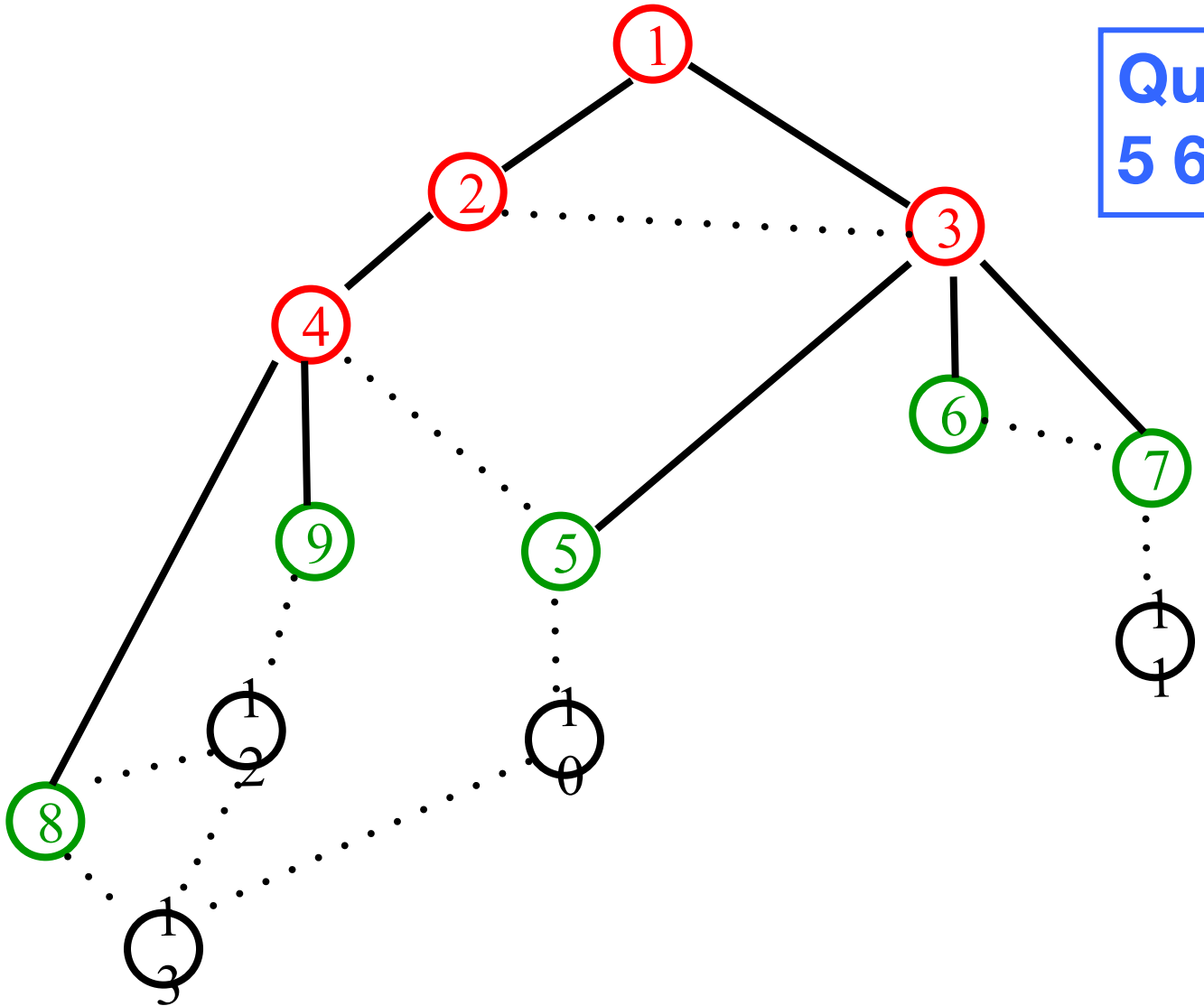


Queue:
3 4

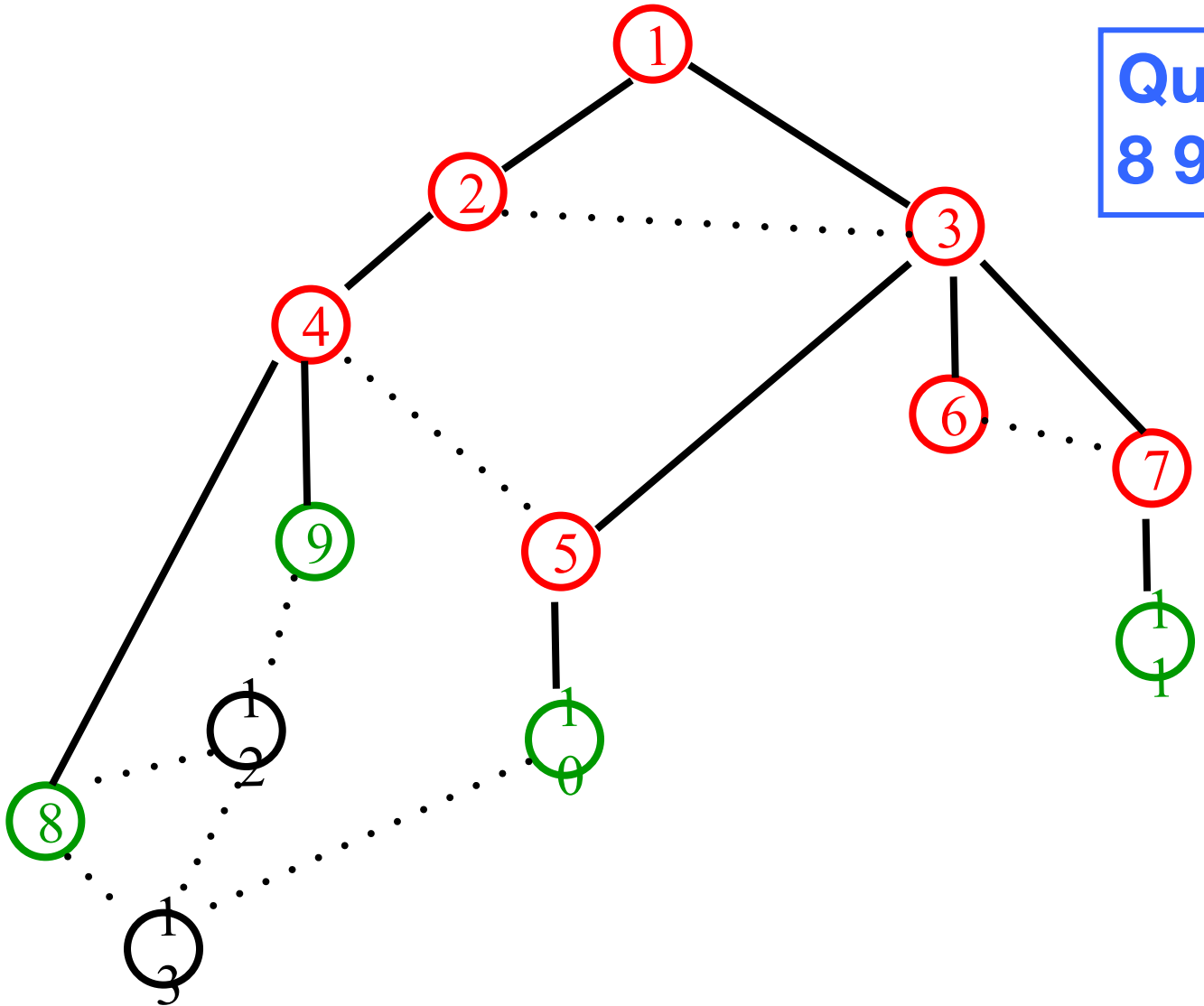
Breadth-first search (BFS)



Breadth-first search (BFS)

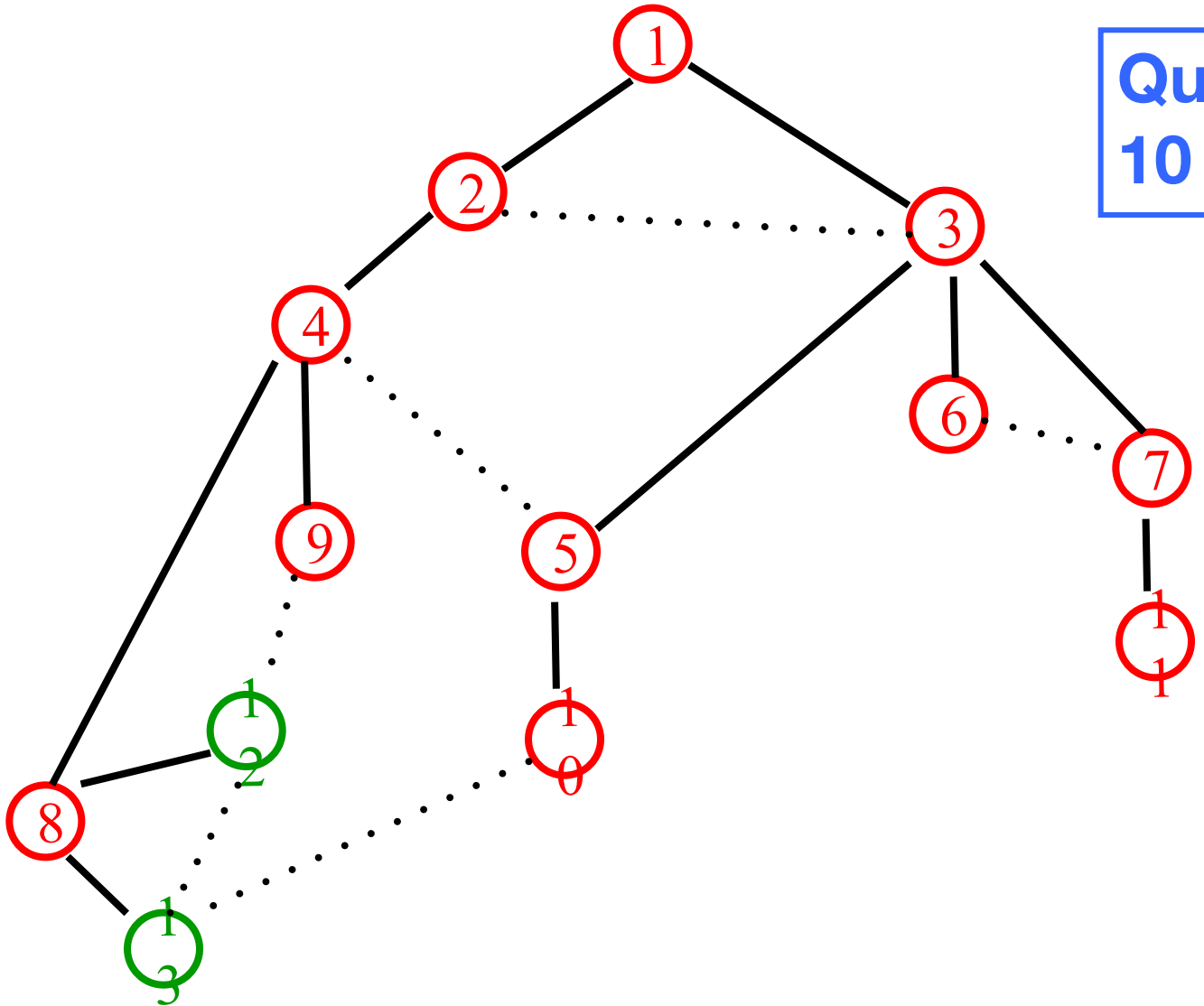


Breadth-first search (BFS)



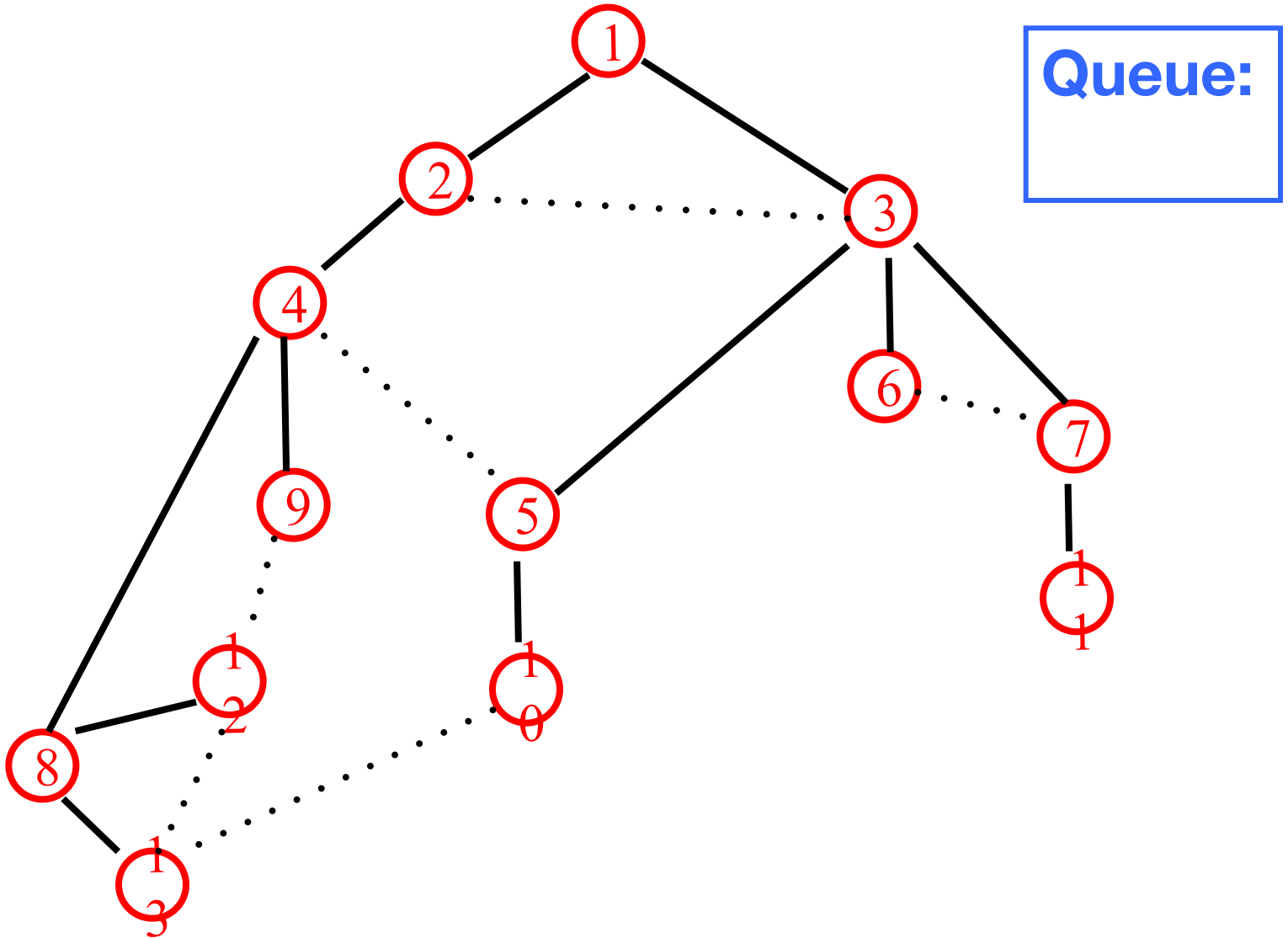
Queue:
8 9 10 11

Breadth-first search (BFS)



Queue:
10 11 12 13

Breadth-first search (BFS)

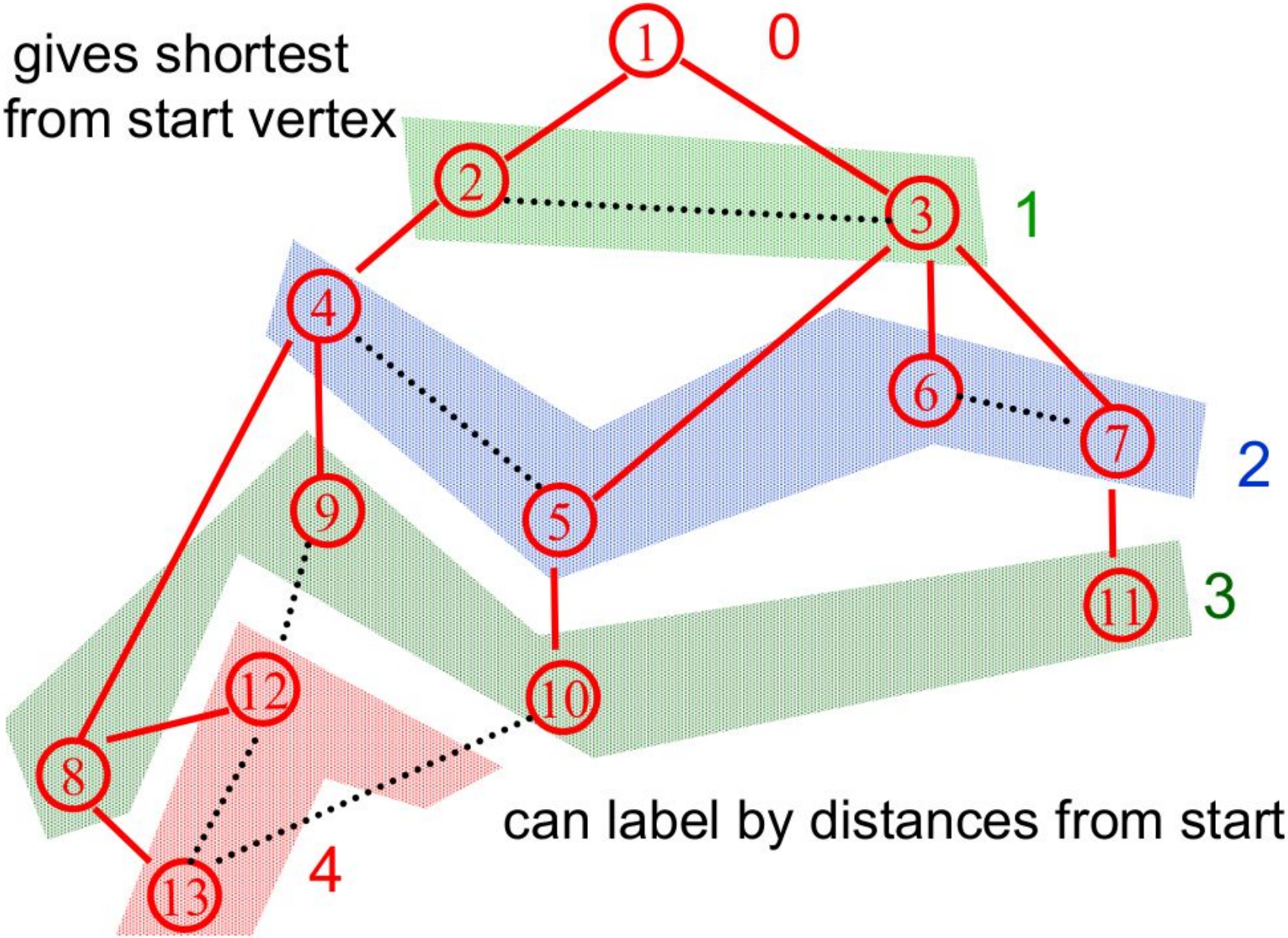


Breadth-first search (BFS)

- BFS can make a tree with the shortest paths from the start to any vertex
  - Q: Why is the set of edges traversed a tree?

- BFS can also record all the distances of nodes from the start

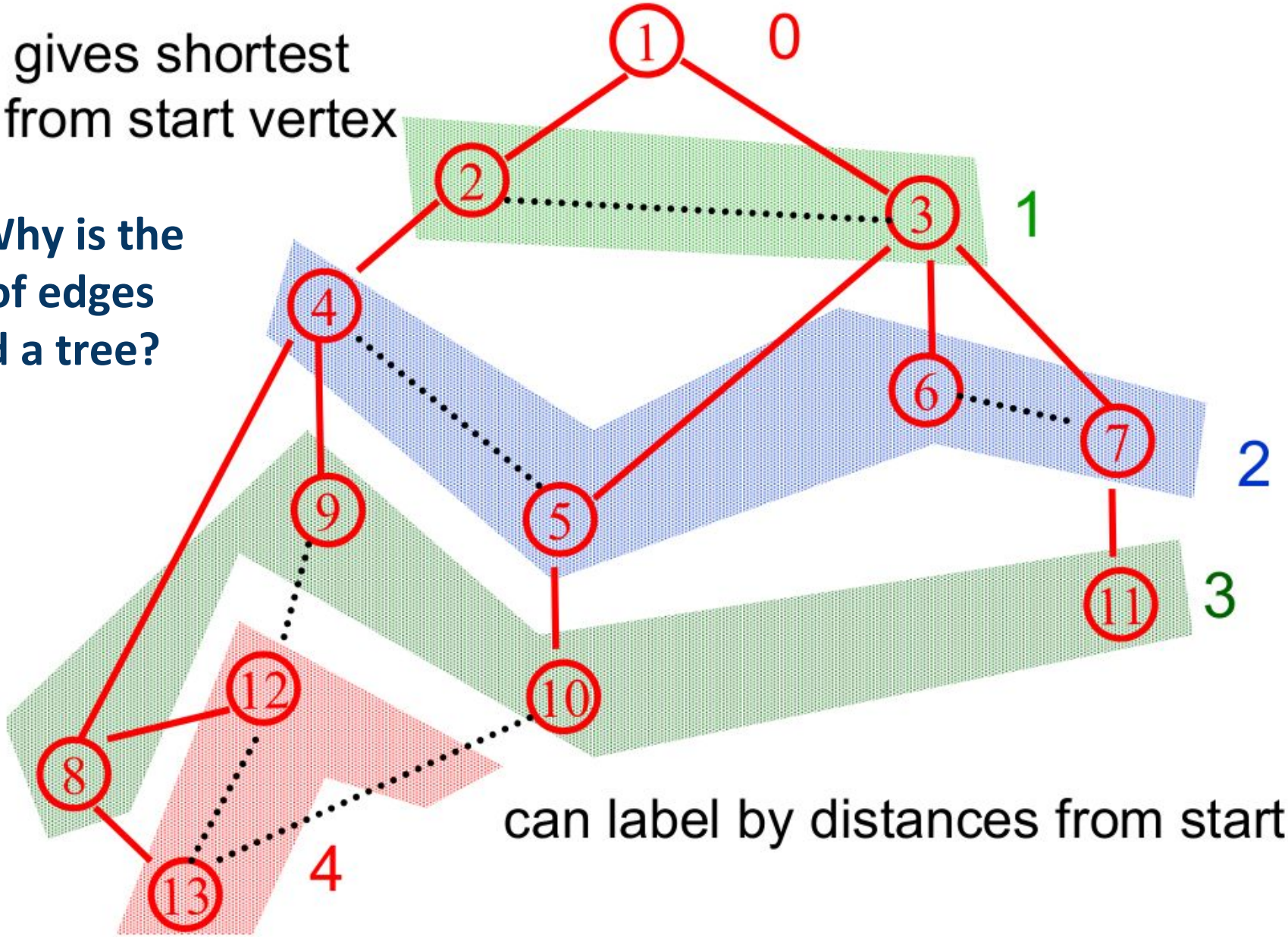
Tree gives shortest paths from start vertex



can label by distances from start

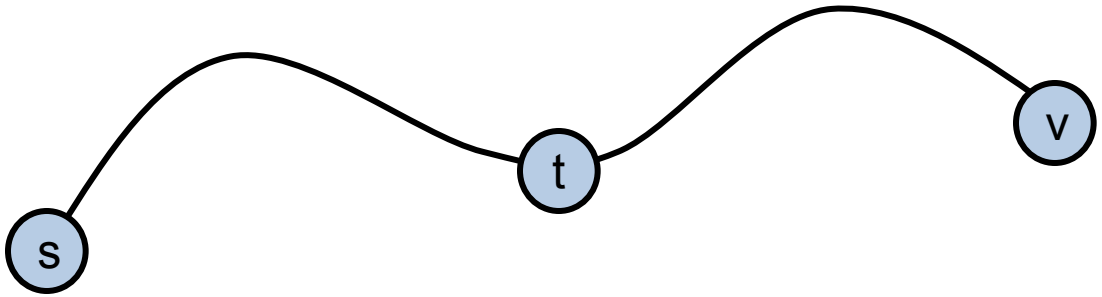
Tree gives shortest paths from start vertex

Q: Why is the set of edges used a tree?



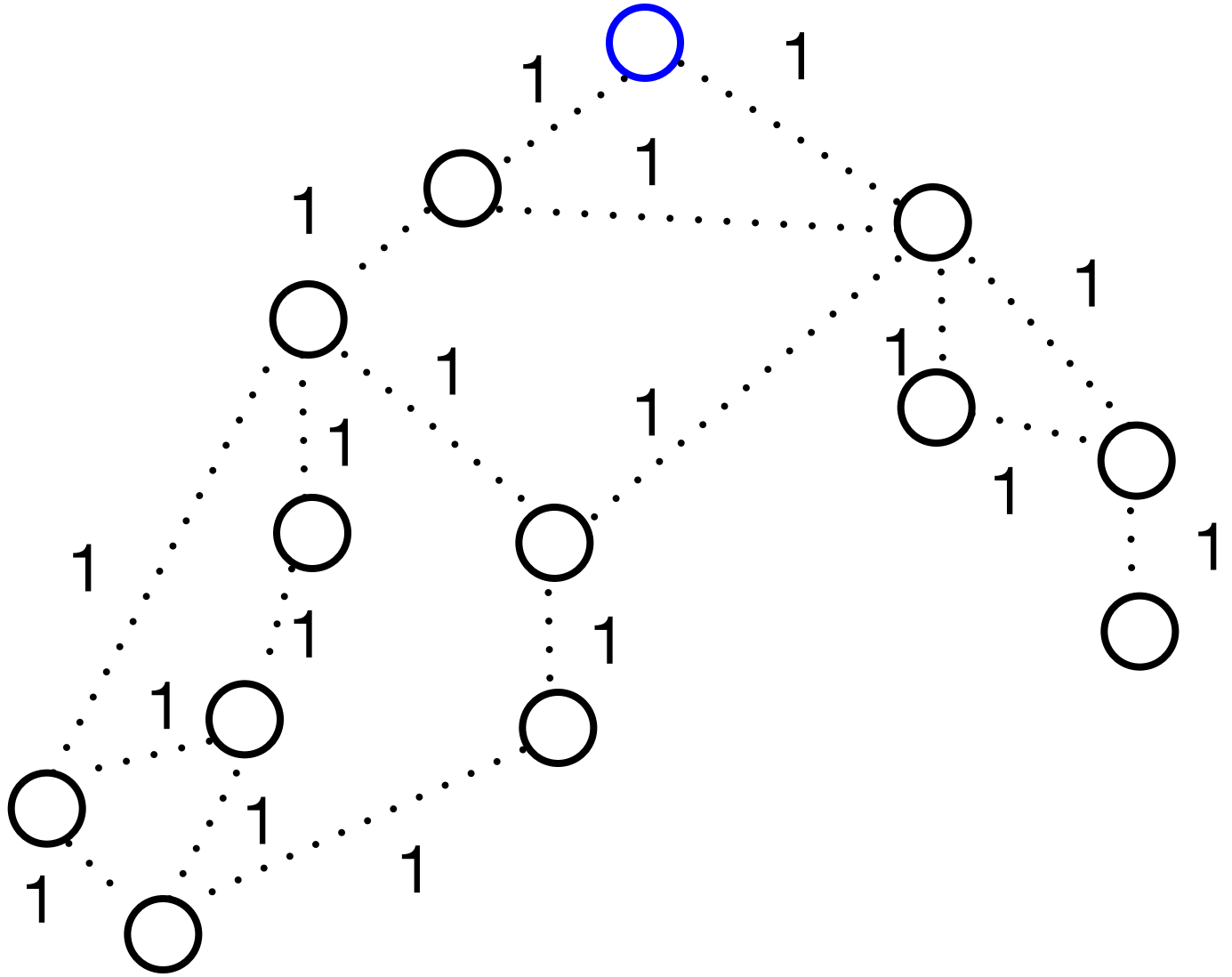
can label by distances from start

- If P is a shortest path from s to v, and if t is on the path P, the segment from s to t is a shortest path between s and t

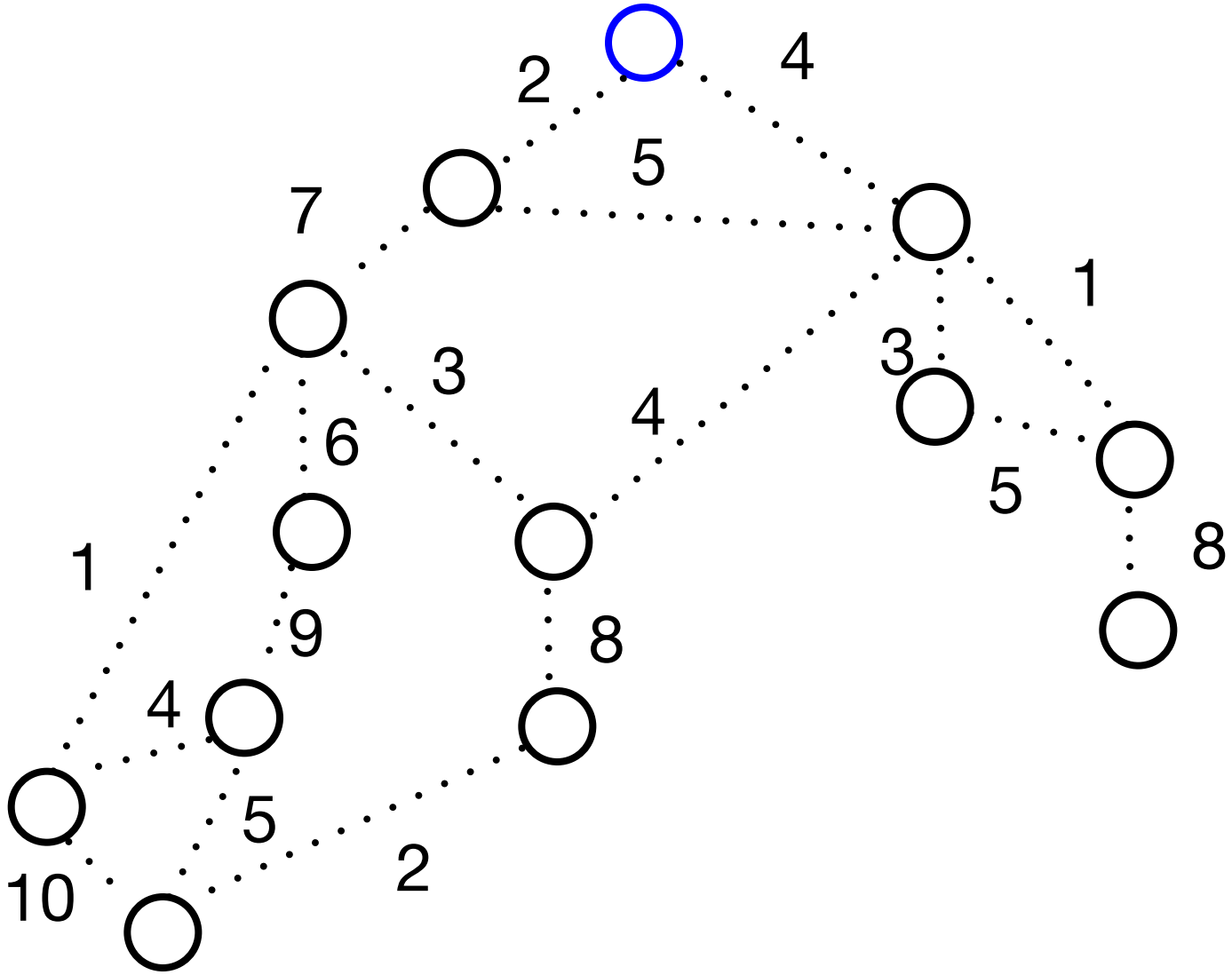


- WHY?

Weighted graphs

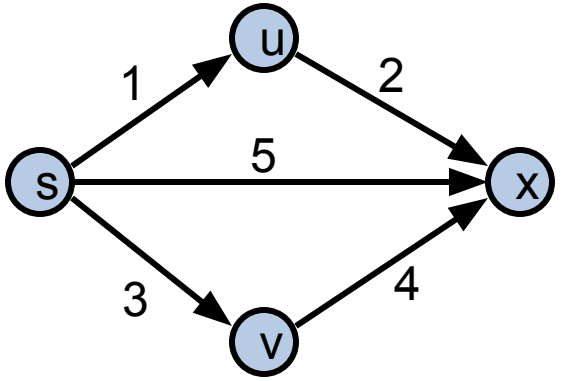


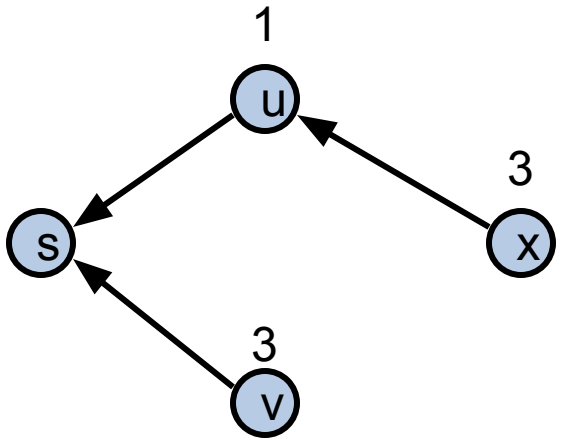
Weighted graphs



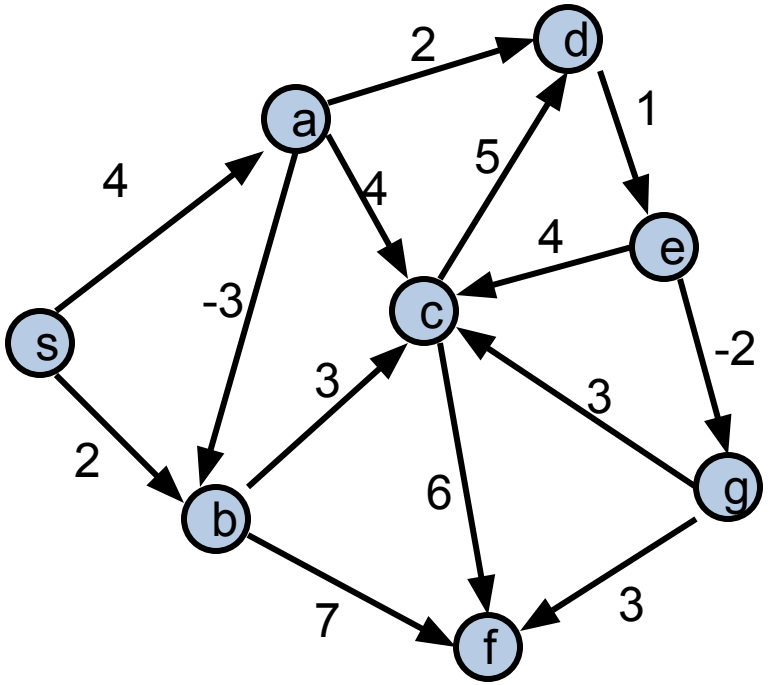
Single Source Shortest Path Problem

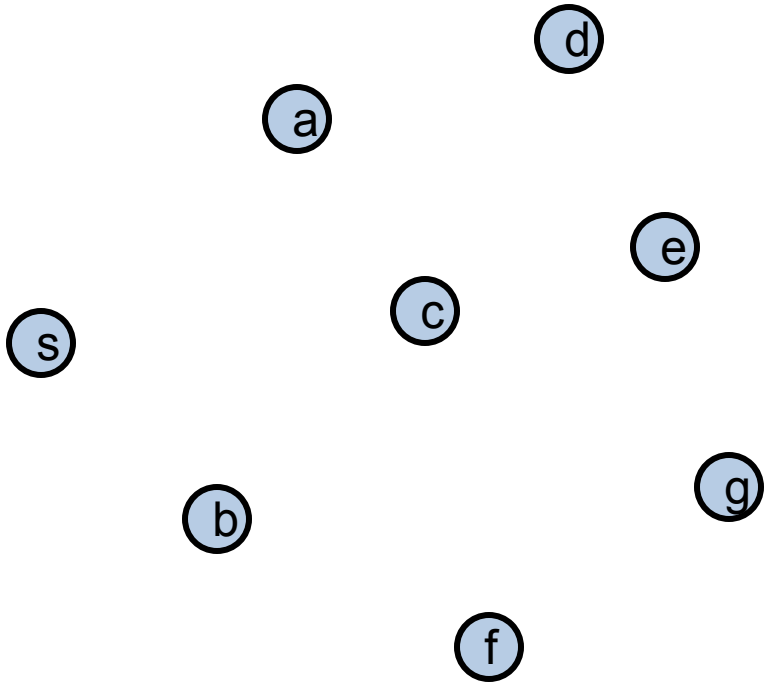
- Given a graph and a start vertex s
  - Determine distance of every vertex from s
  - Identify shortest paths to each vertex
    - Express concisely as a “shortest paths tree”
    - Each vertex has a pointer to a predecessor on shortest path





Construct Shortest Path Tree from s



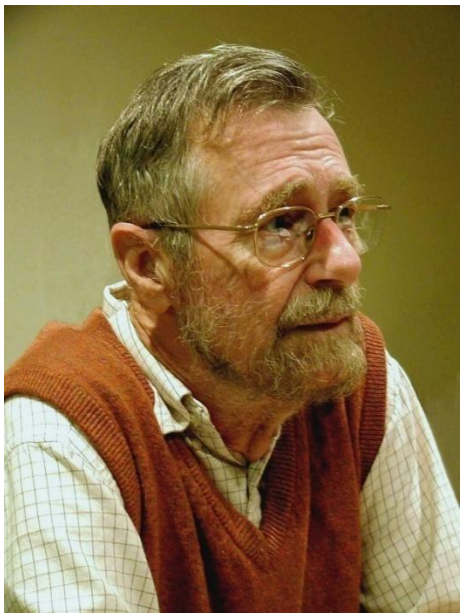


Who was Dijkstra?

- What were his major contributions?

http://www.cs.utexas.edu/users/EWD/

- Edsger Wybe Dijkstra was one of the most influential members of computing science's founding generation. Among the domains in which his scientific contributions are fundamental are
  - algorithm design
  - programming languages
  - program design
  - operating systems
  - distributed processing
  - formal specification and verification
  - design of mathematical arguments



Dijkstra's algorithm

"In 1956 I did two important things, I got my degree and we had the festive opening of the ARMAC. **We had to have a demonstration**... For a demonstration for noncomputing people you have to have a problem statement that non-mathematicians can understand; they even have to understand the answer. So I designed a program that would find the shortest route between two cities in the Netherlands"

Image: http://cs-exhibitions.uni-klu.ac.at/index.php?id=29
Quote: https://dl.acm.org/doi/pdf/10.1145/1787234.1787249

Assume all edges have non-negative cost

Dijkstra's Algorithm

S = { }; d[s] = 0; d[v] = infinity for v != s

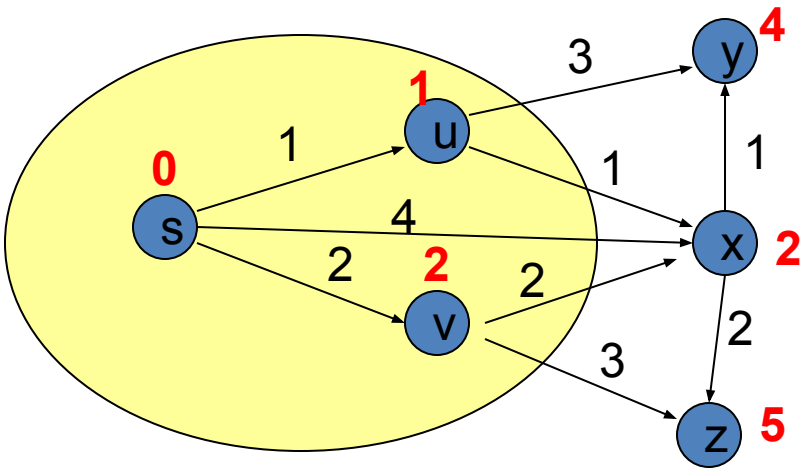
While S != V

    Choose v in V-S with minimum d[v]

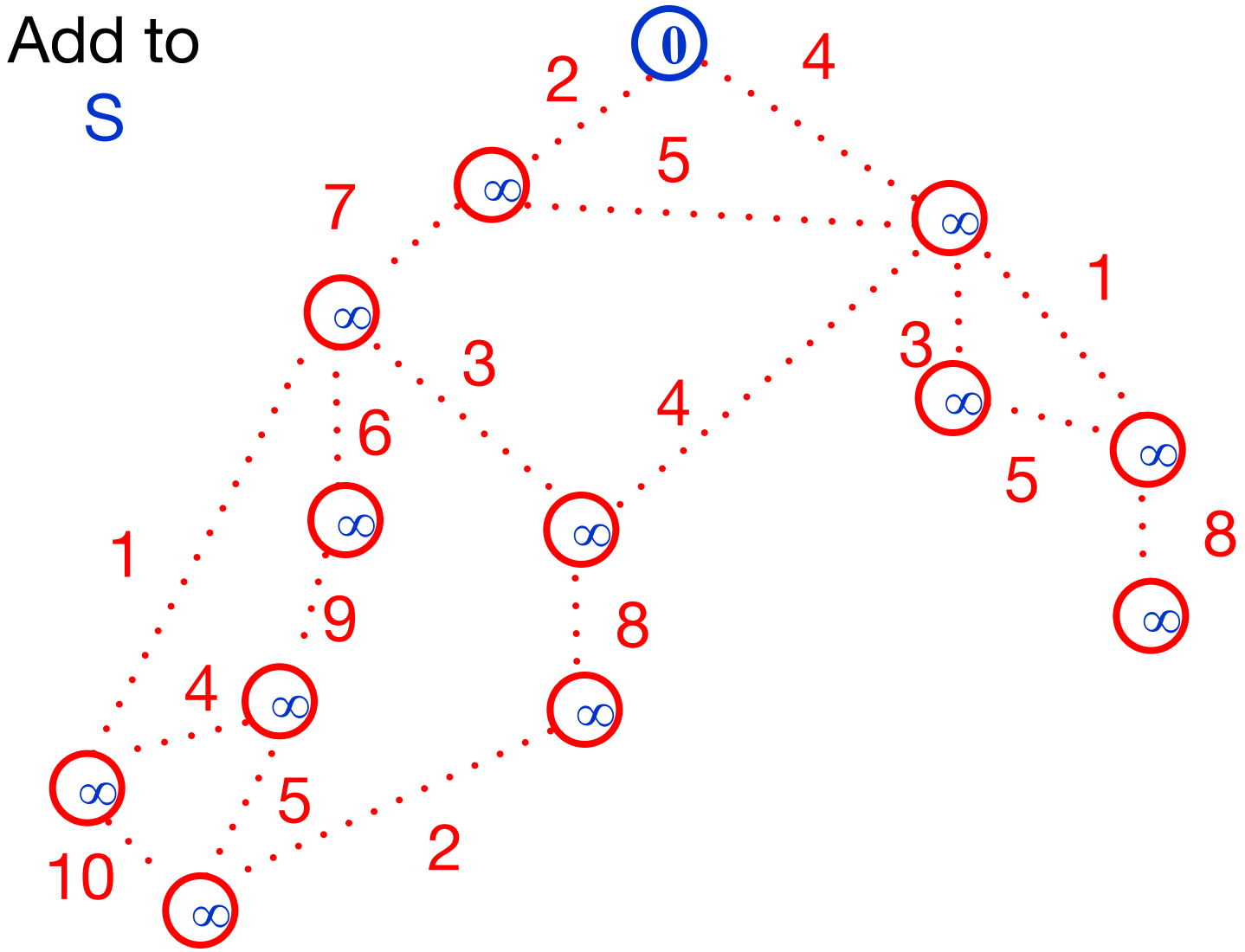
    Add v to S

    For each w in the neighborhood of v

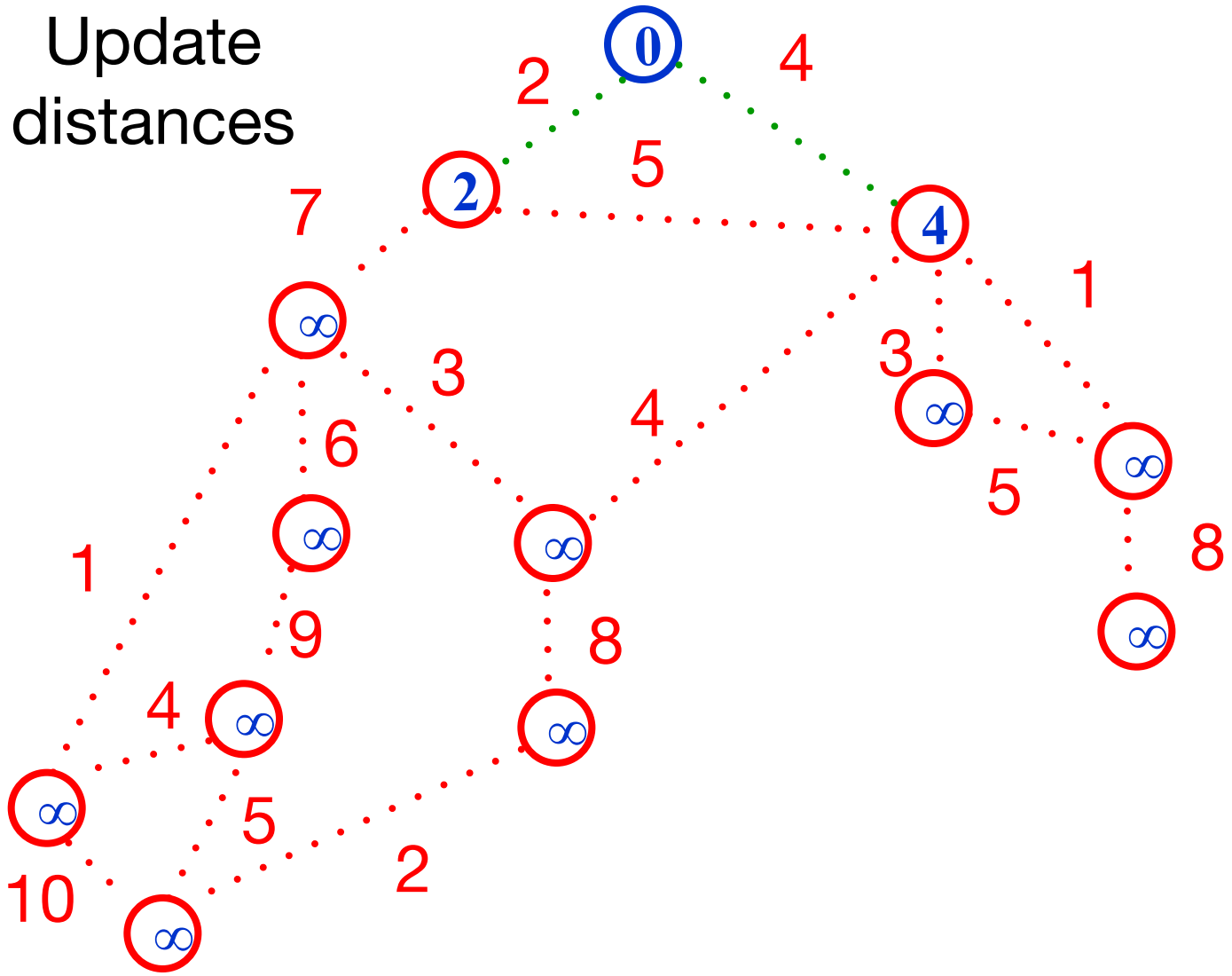
        d[w] = min(d[w], d[v] + c(v, w))



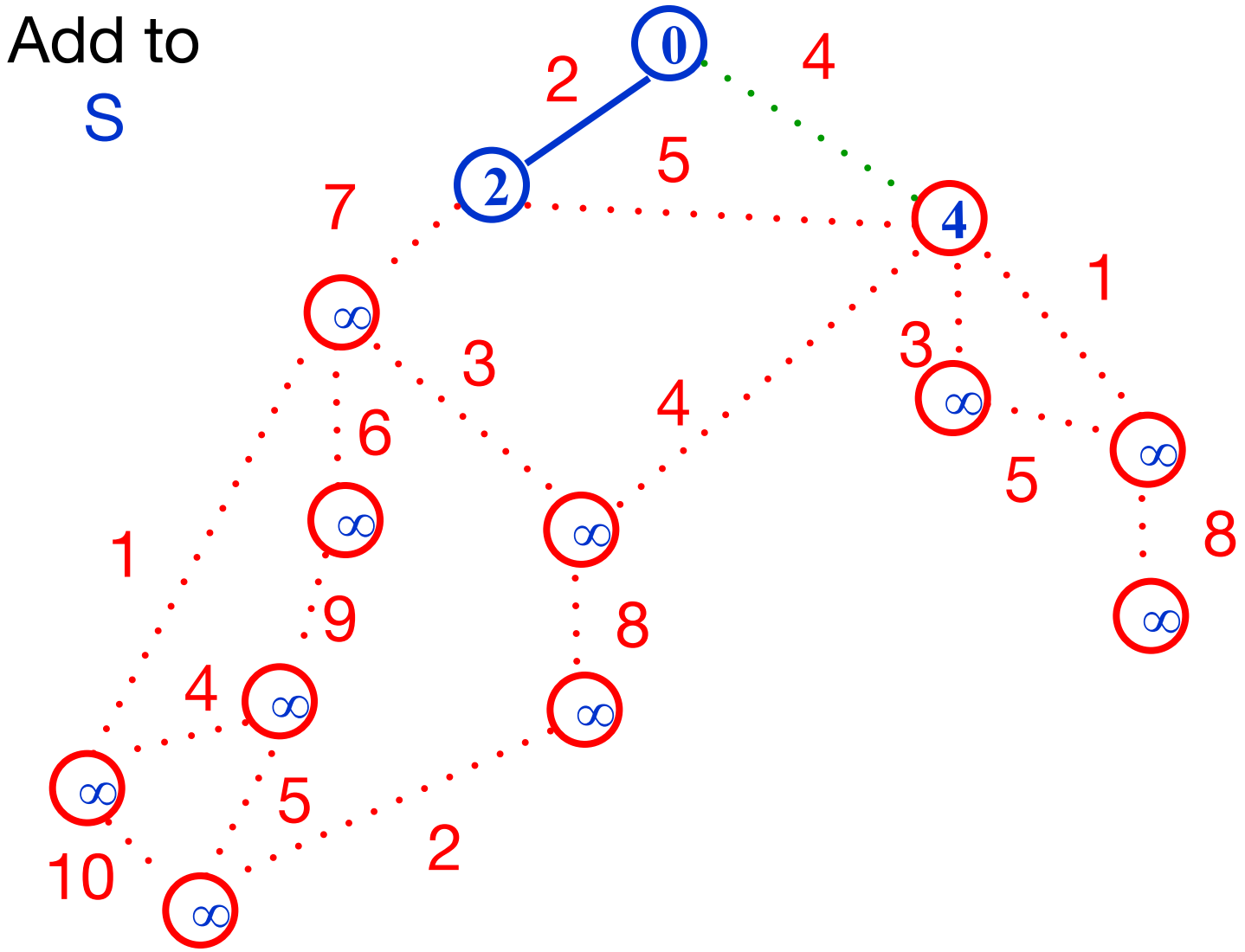
Dijkstra's Algorithm



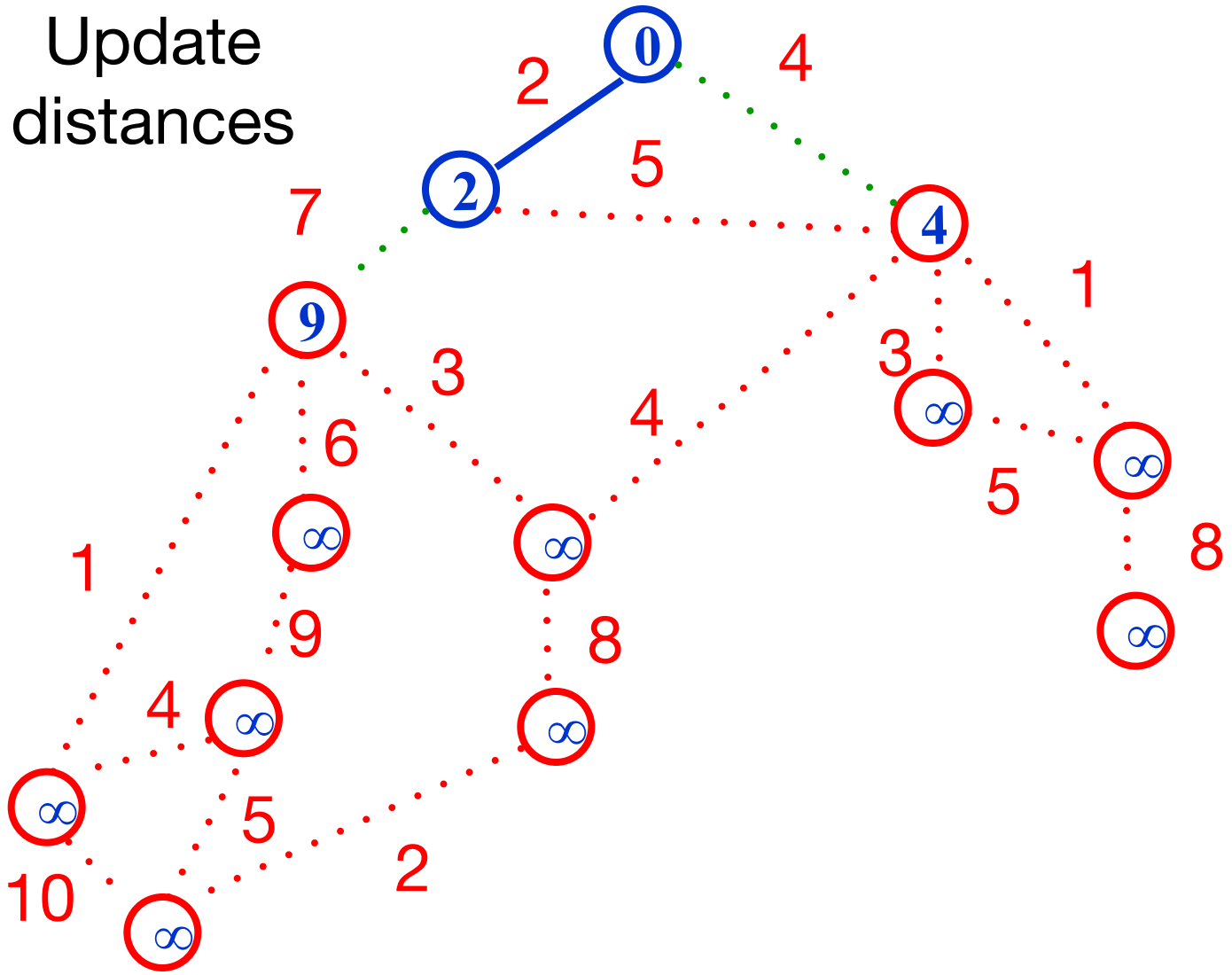
Dijkstra's Algorithm



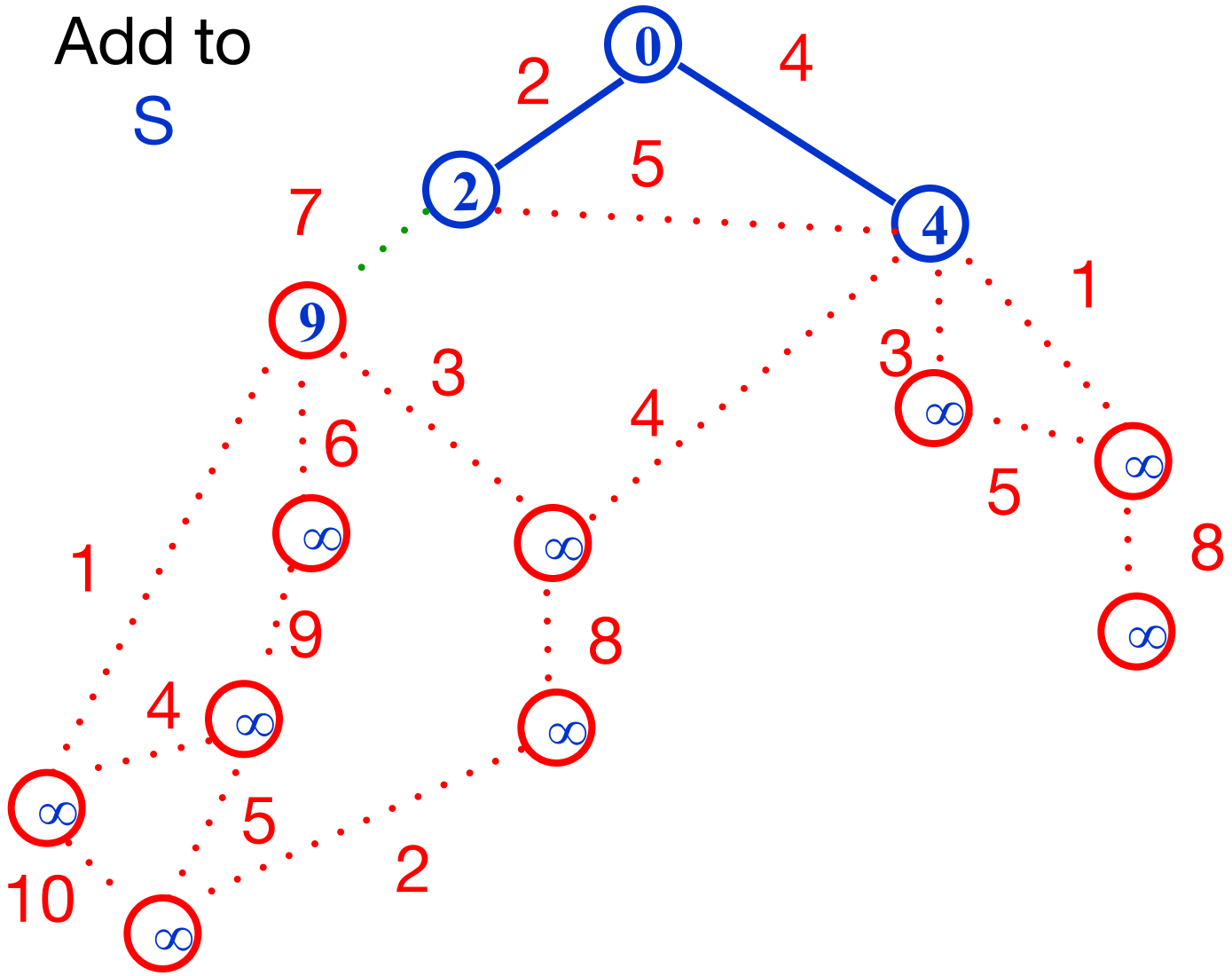
Dijkstra's Algorithm



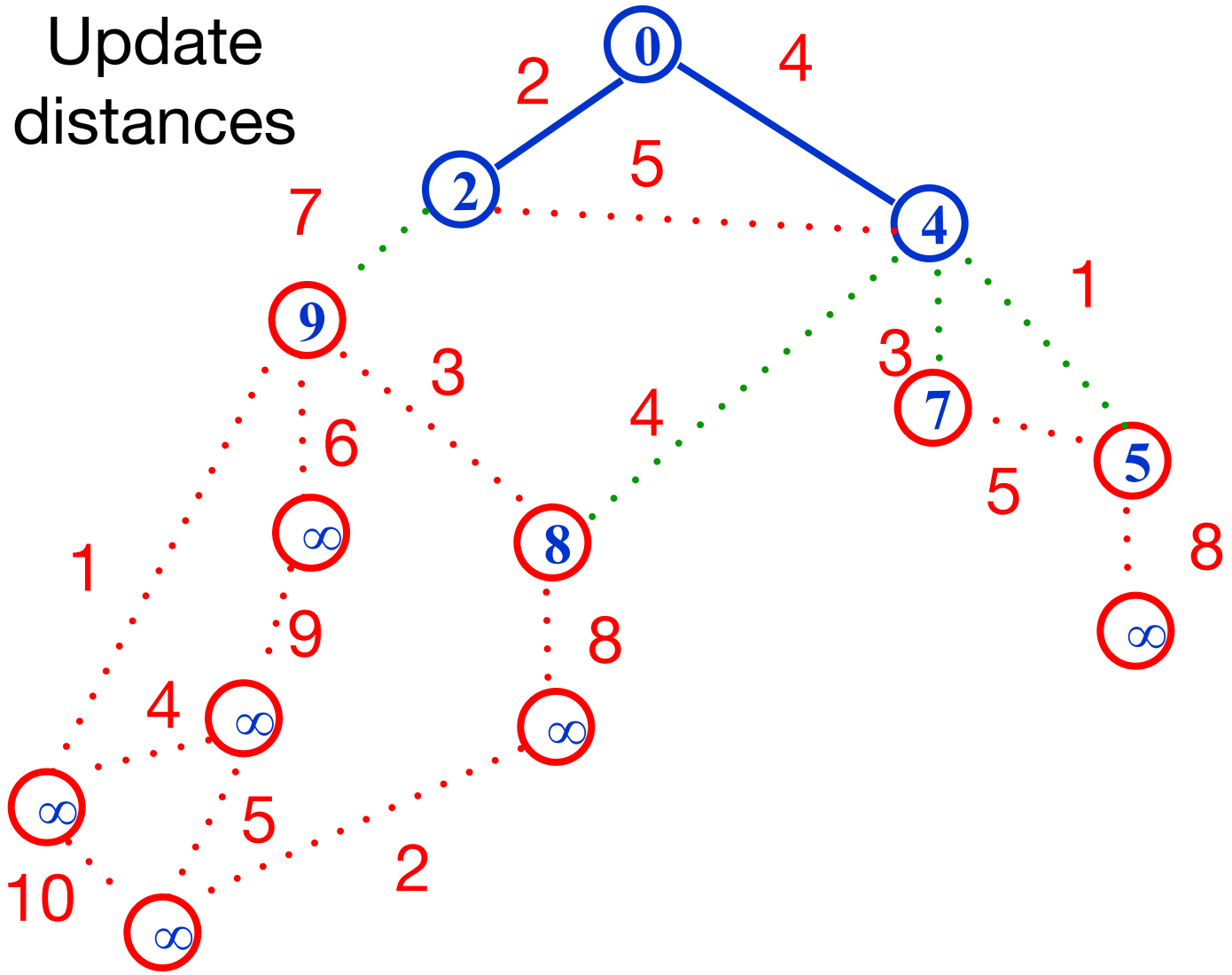
Dijkstra's Algorithm



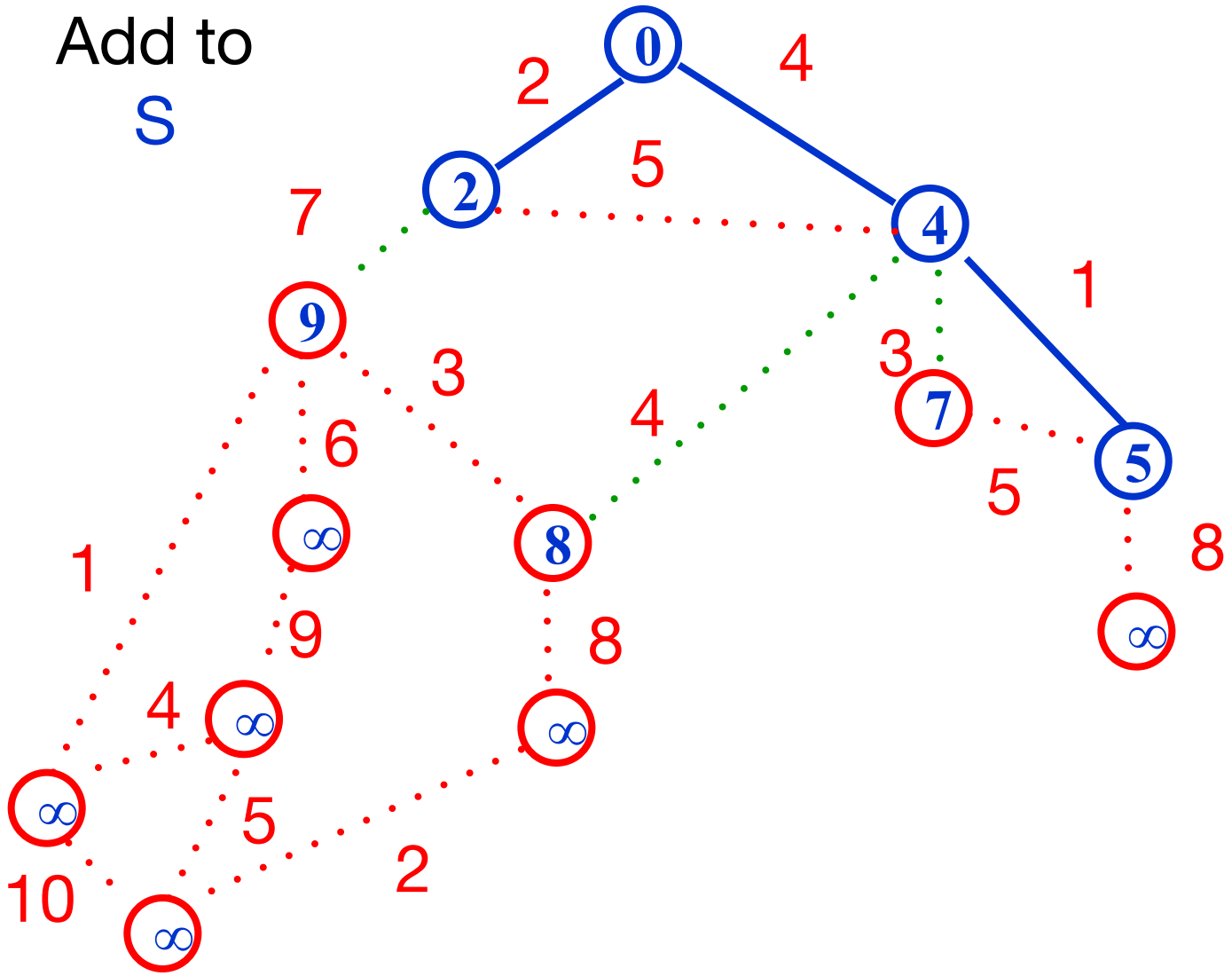
Dijkstra's Algorithm



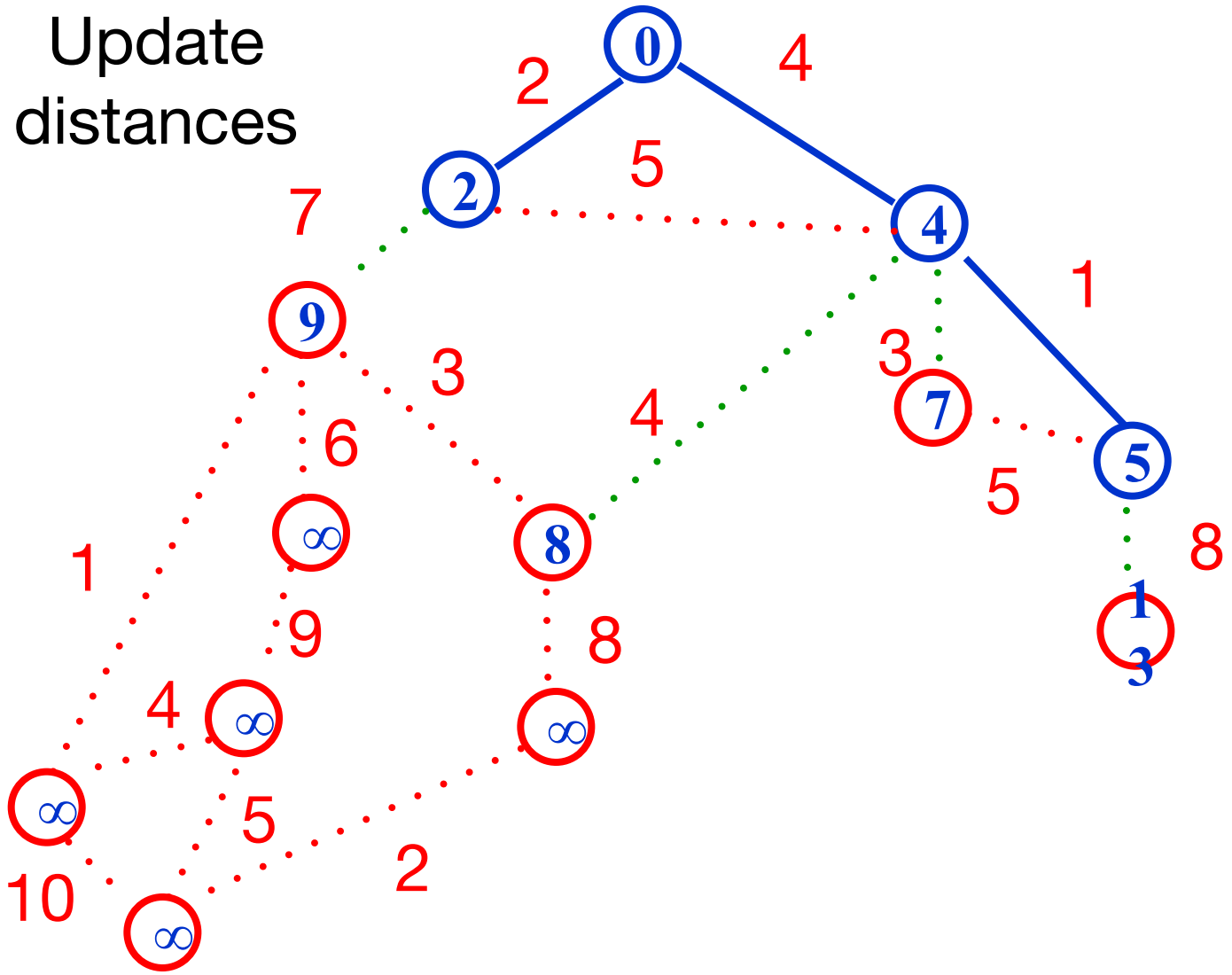
Dijkstra's Algorithm



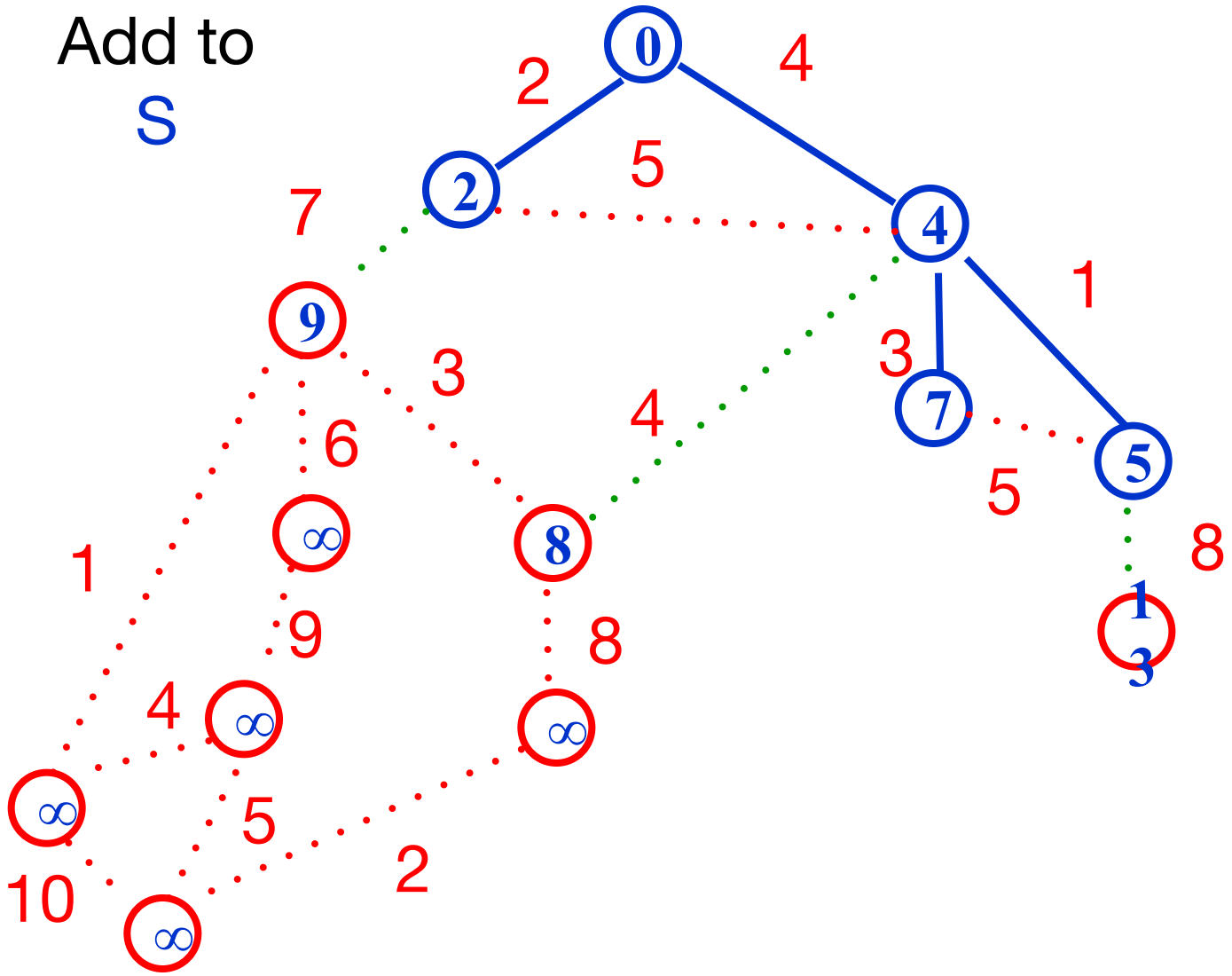
Dijkstra's Algorithm



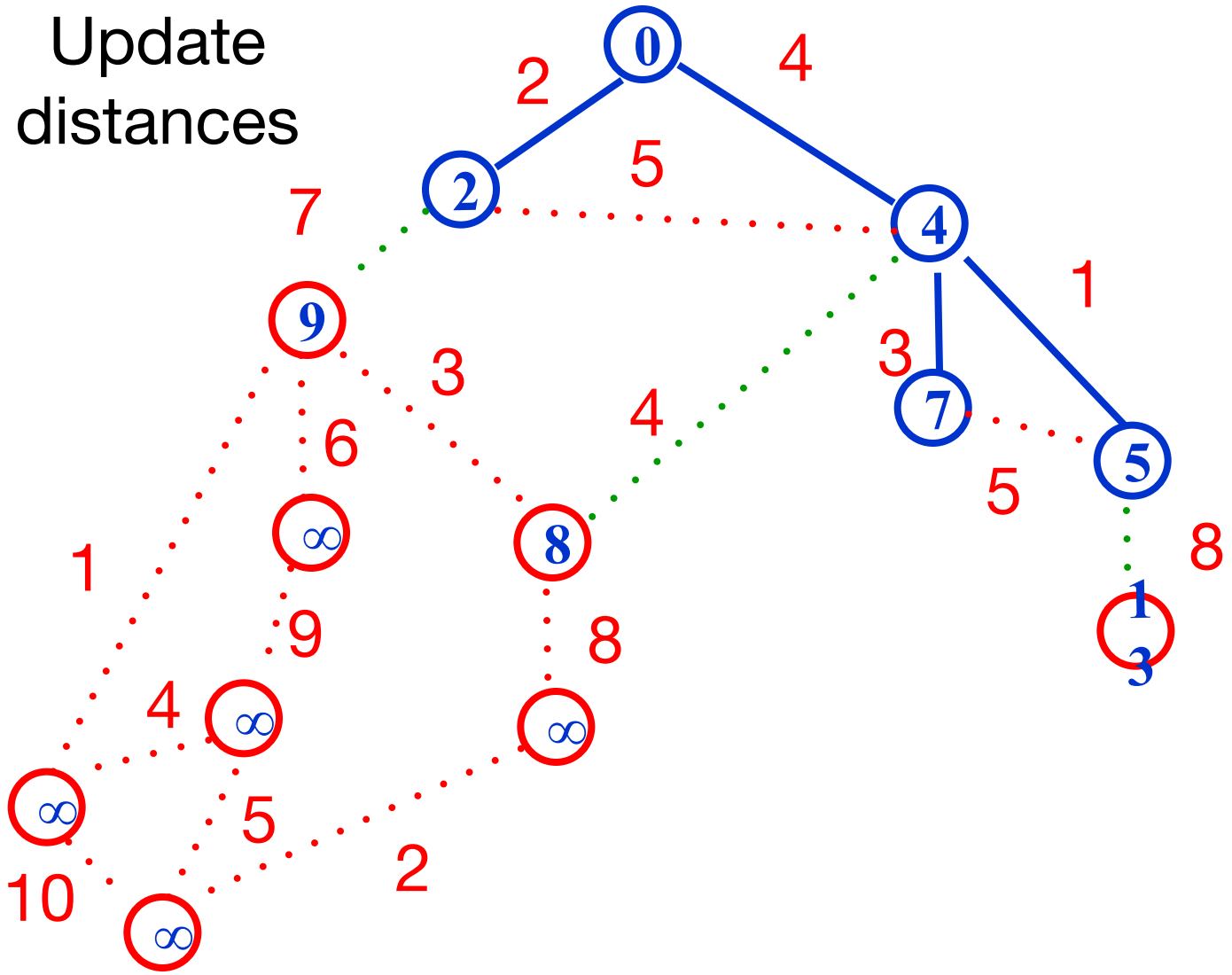
Dijkstra's Algorithm



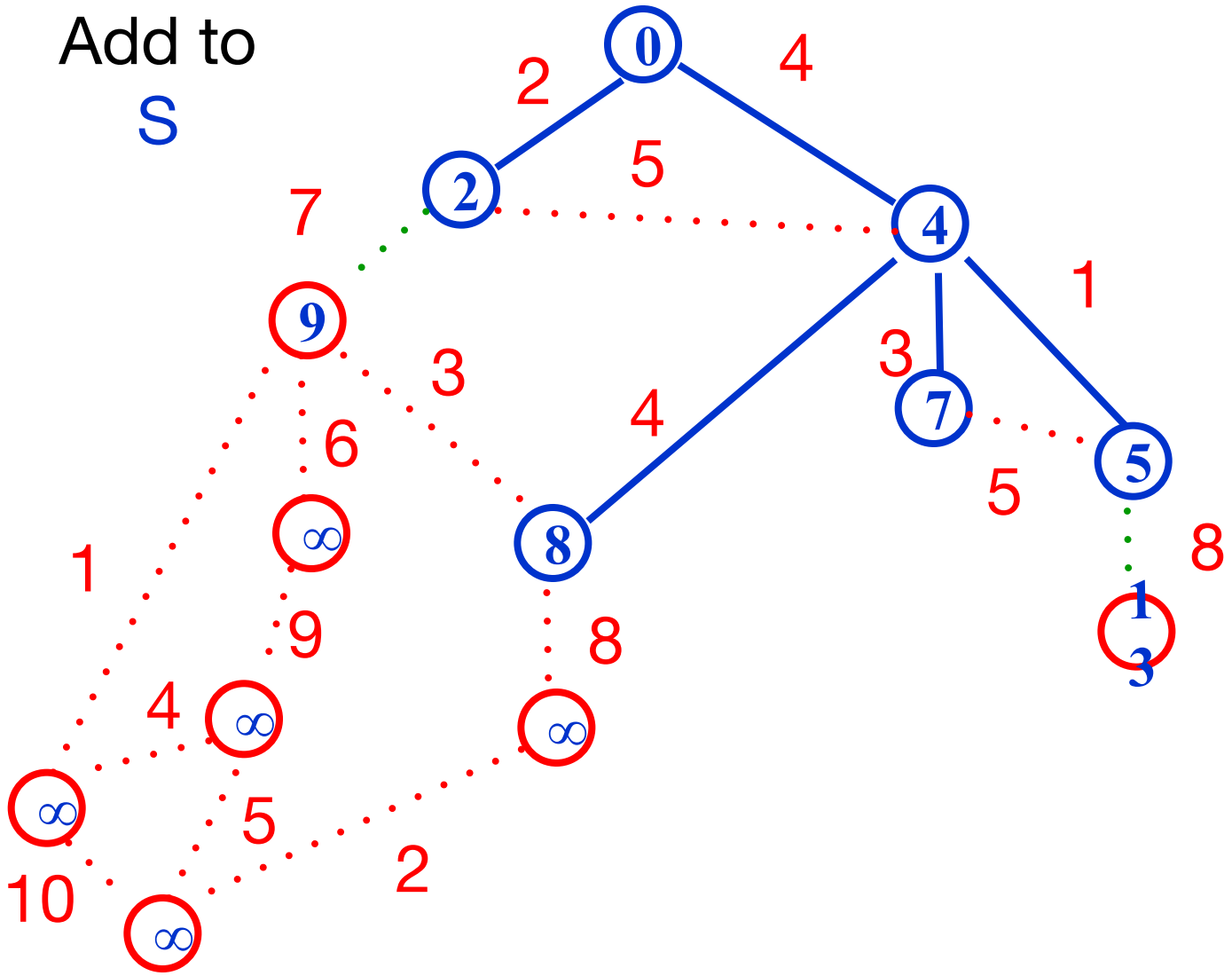
Dijkstra's Algorithm



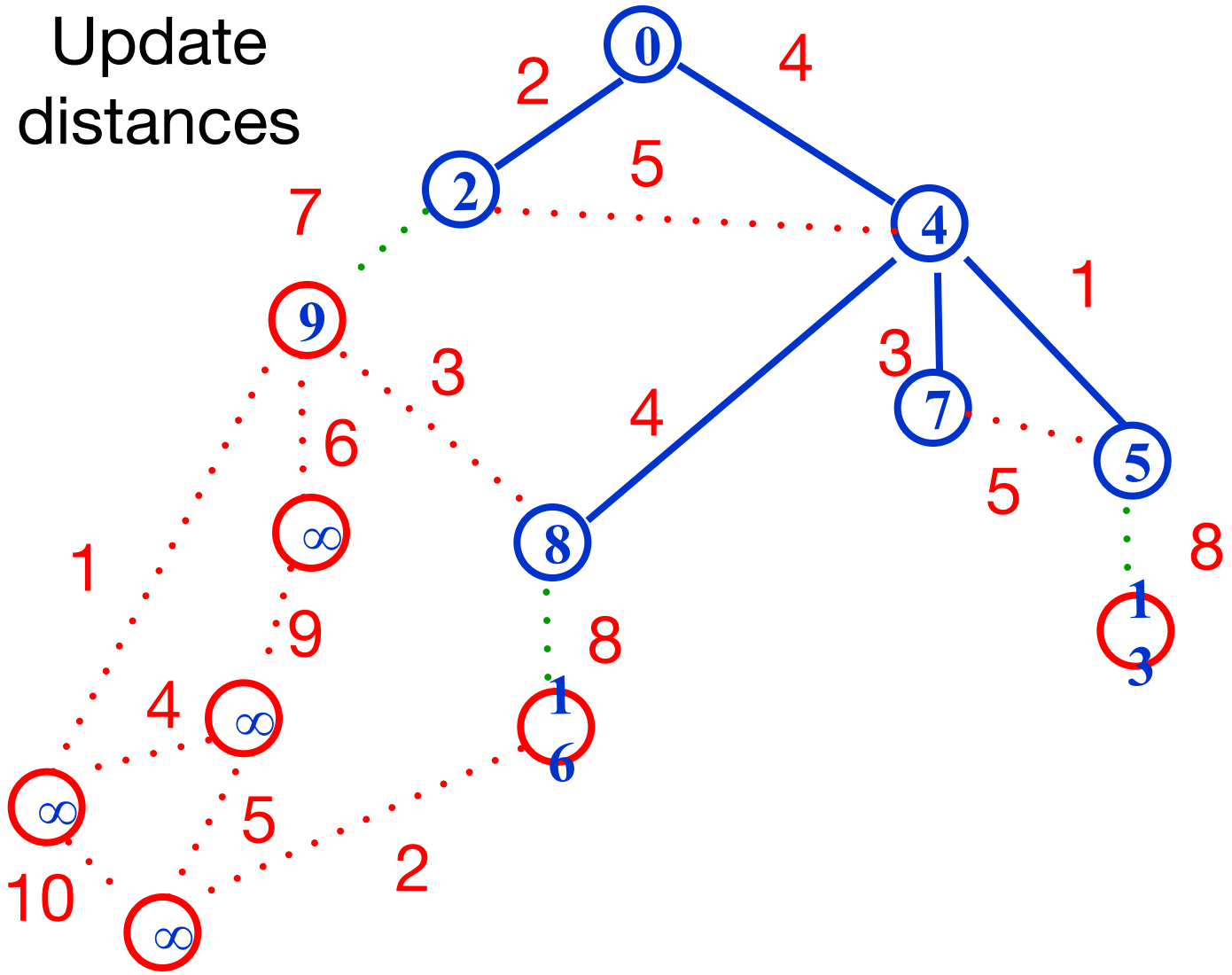
Dijkstra's Algorithm



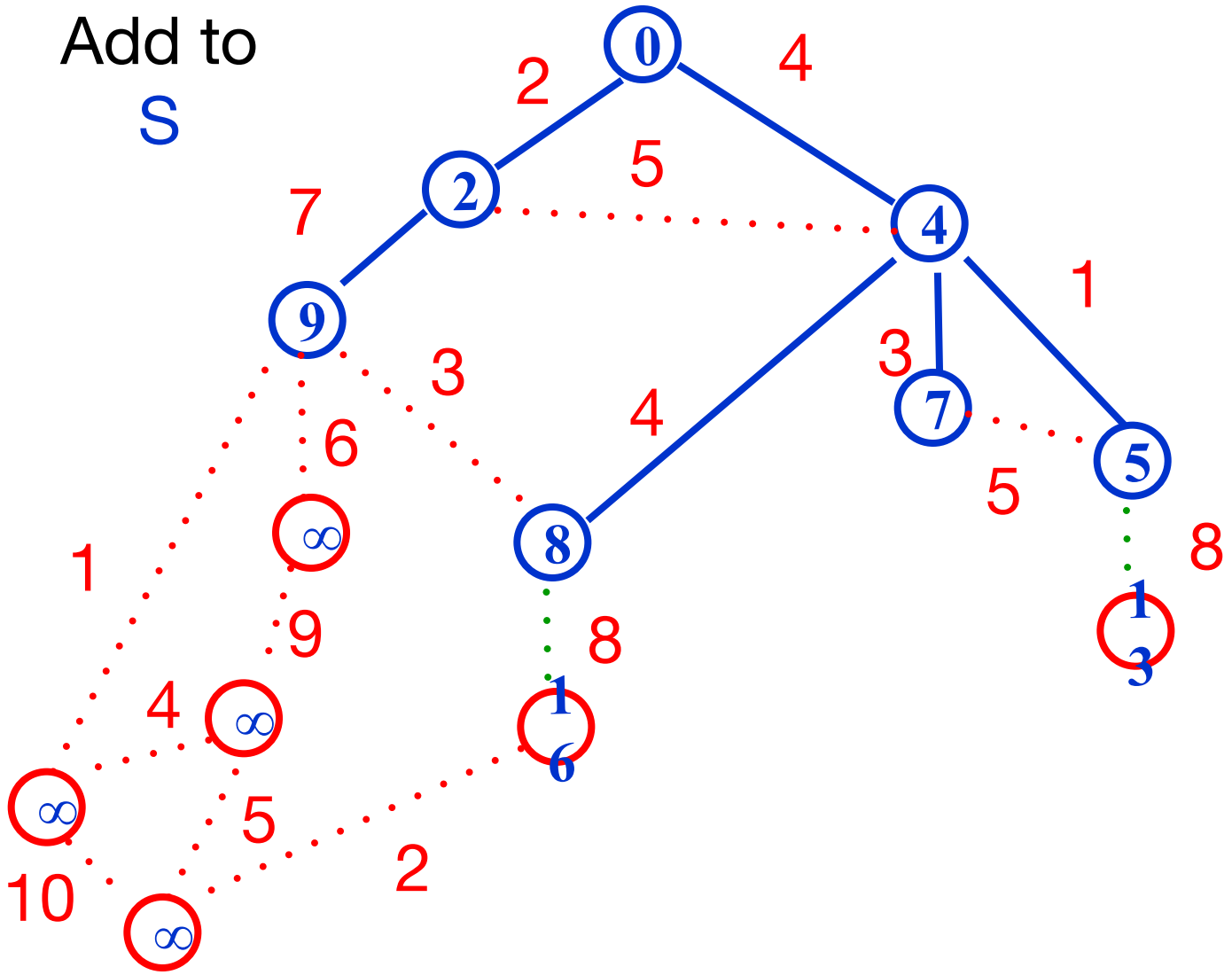
Dijkstra's Algorithm



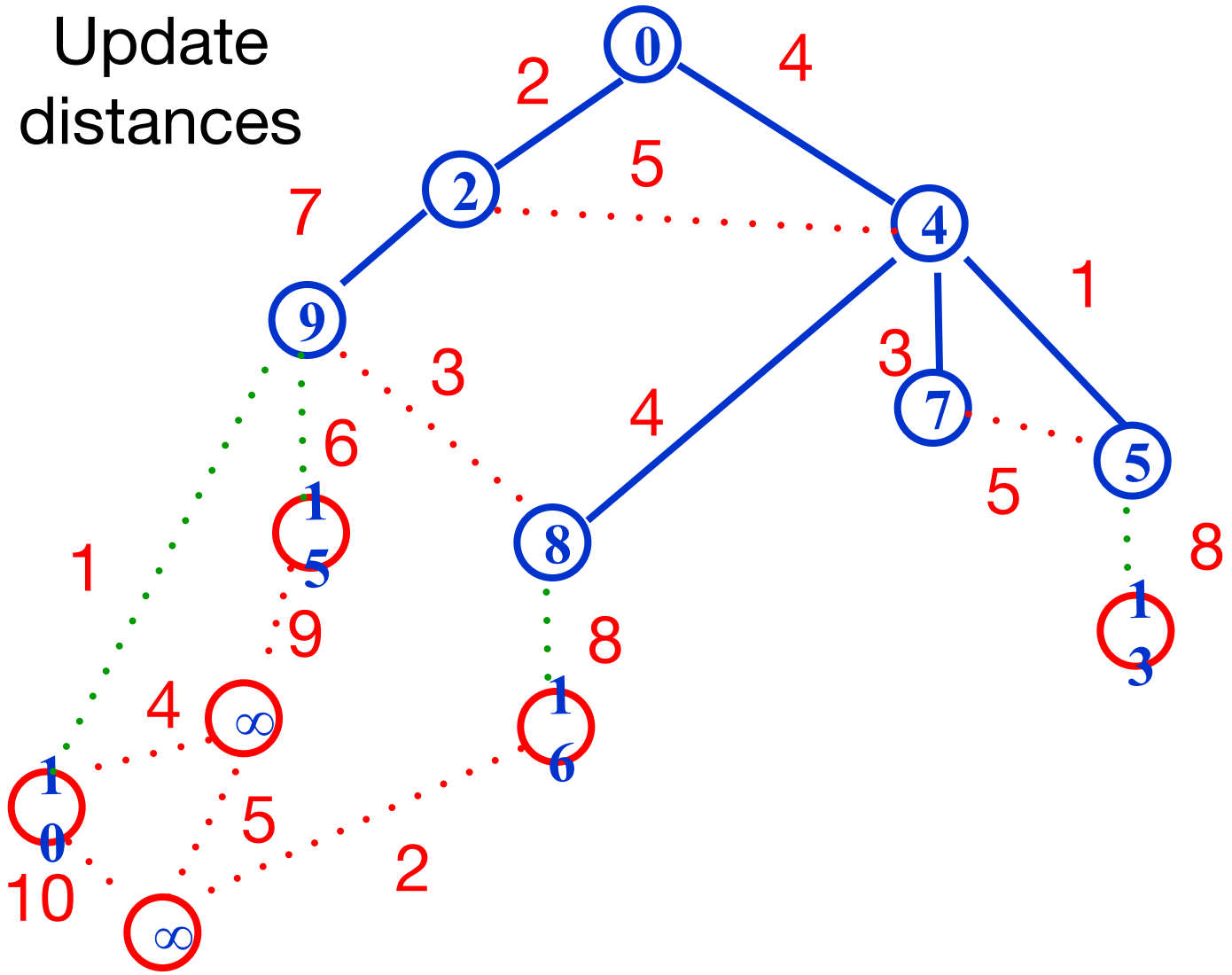
Dijkstra's Algorithm



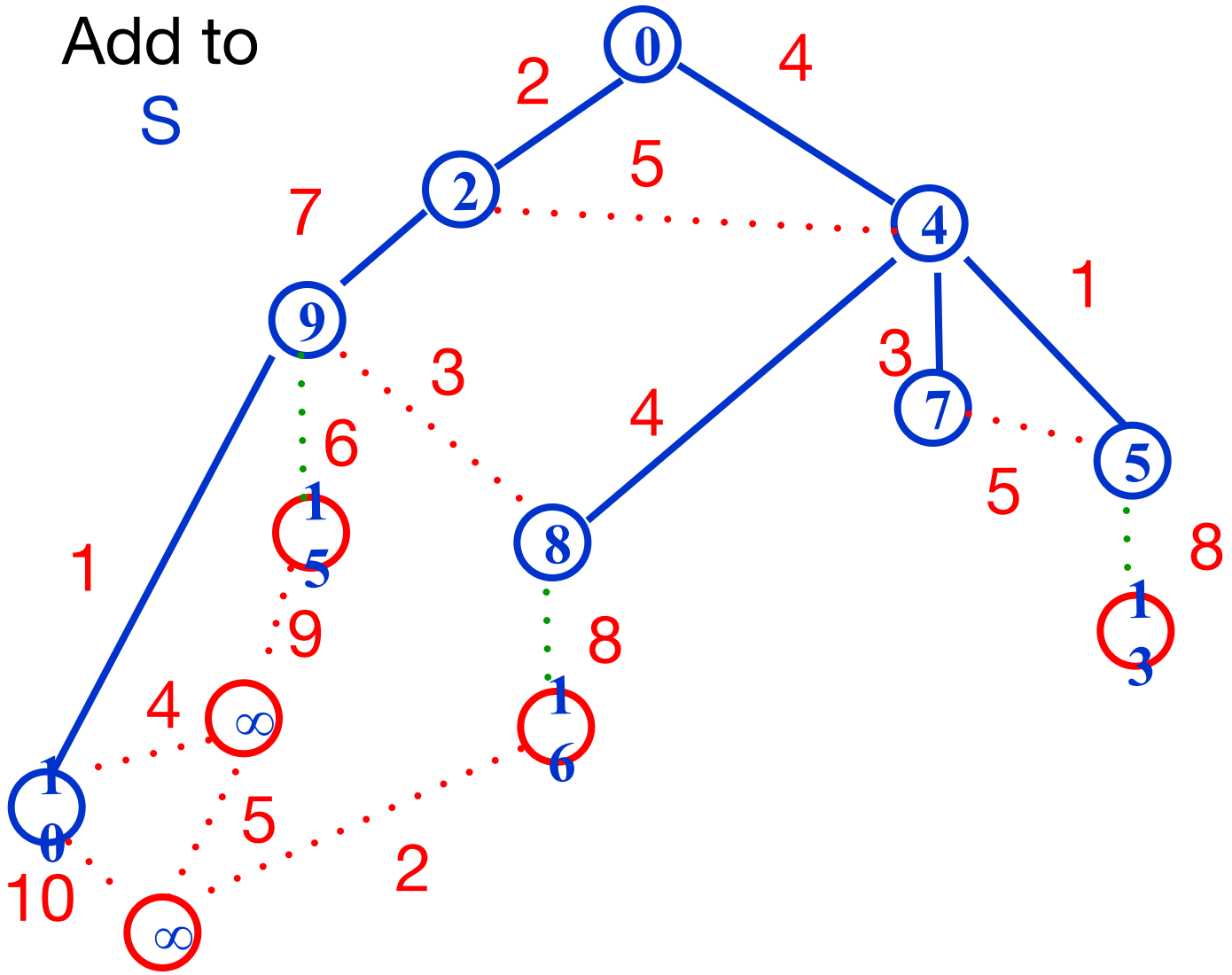
Dijkstra's Algorithm



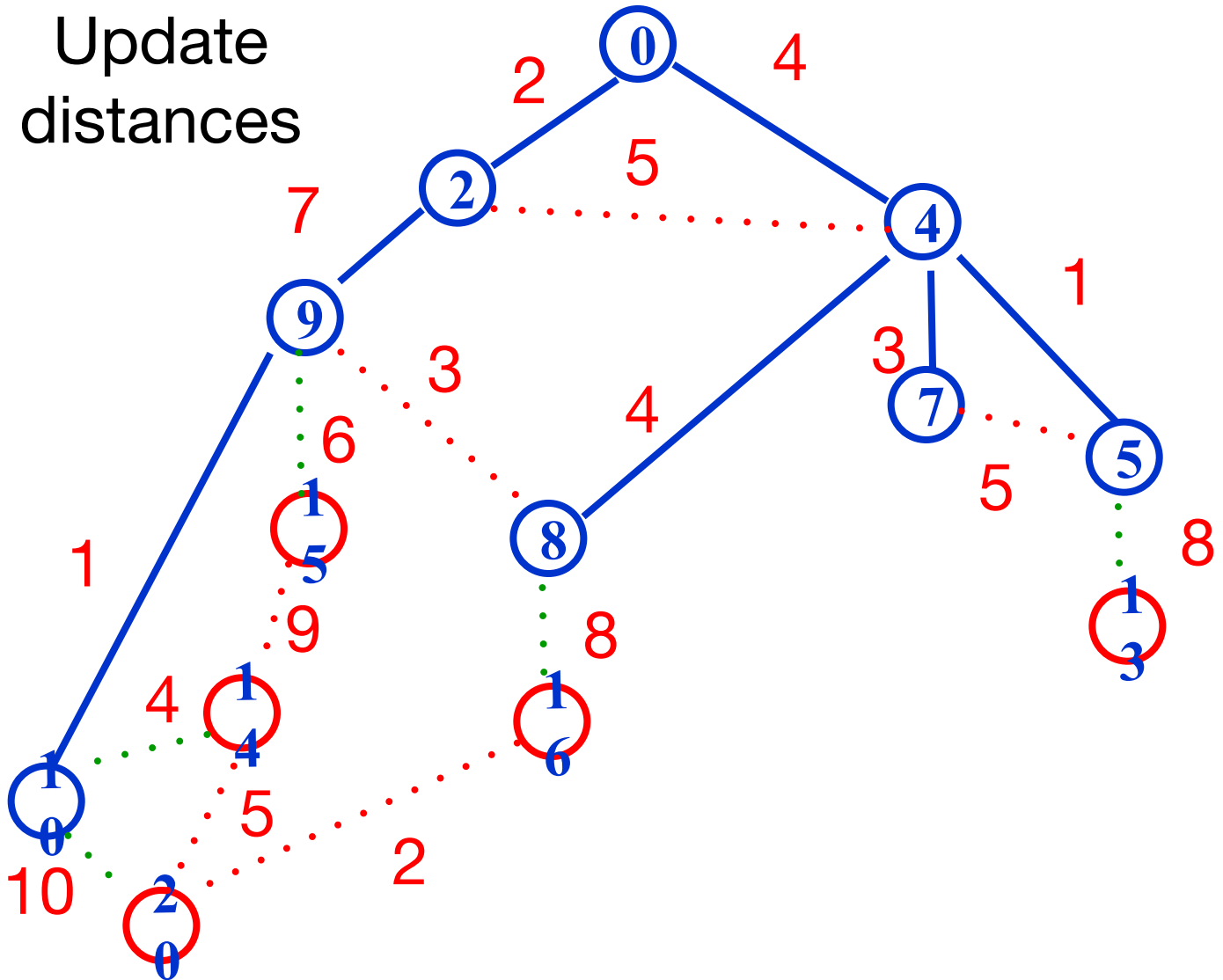
Dijkstra's Algorithm



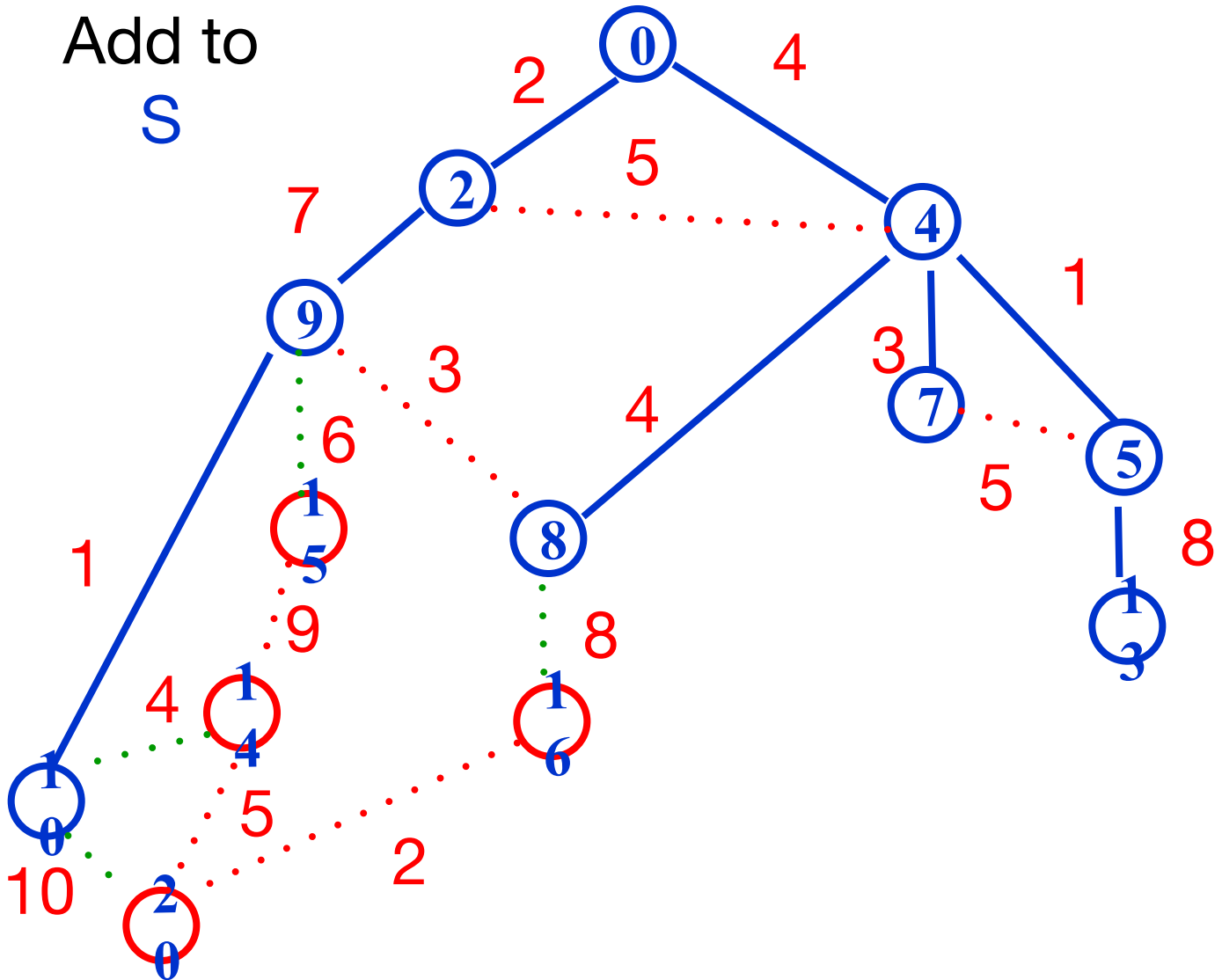
Dijkstra's Algorithm



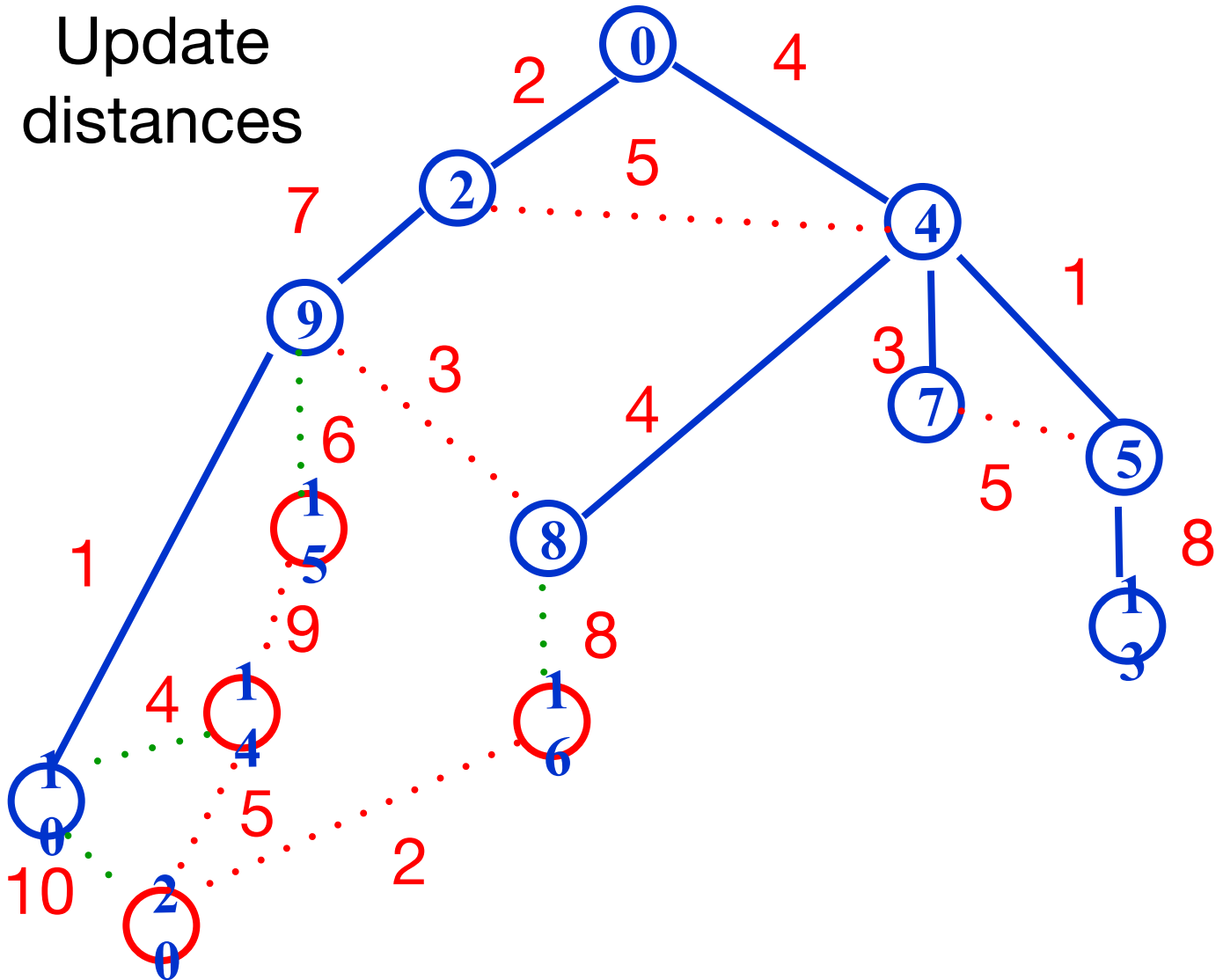
Dijkstra's Algorithm



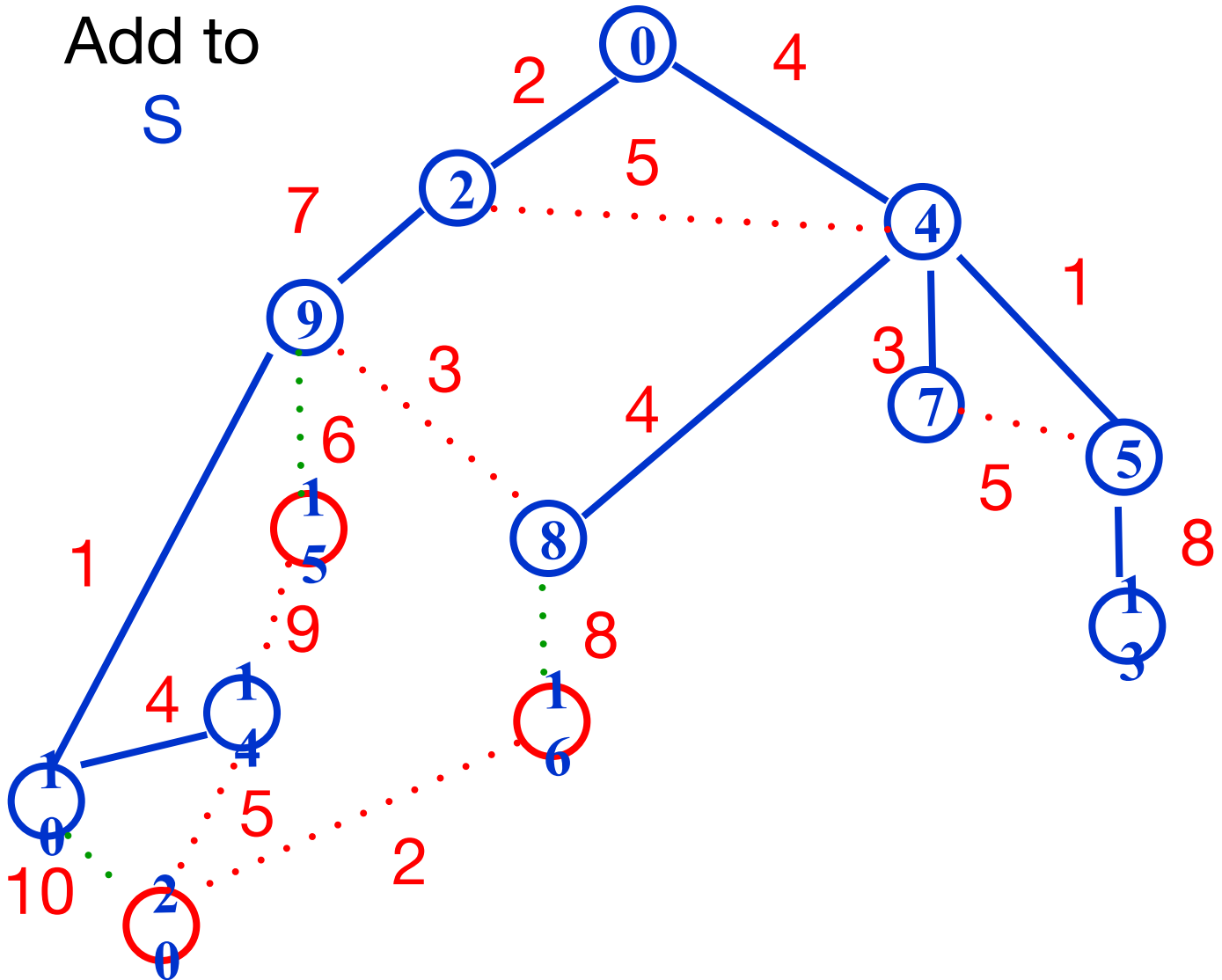
Dijkstra's Algorithm



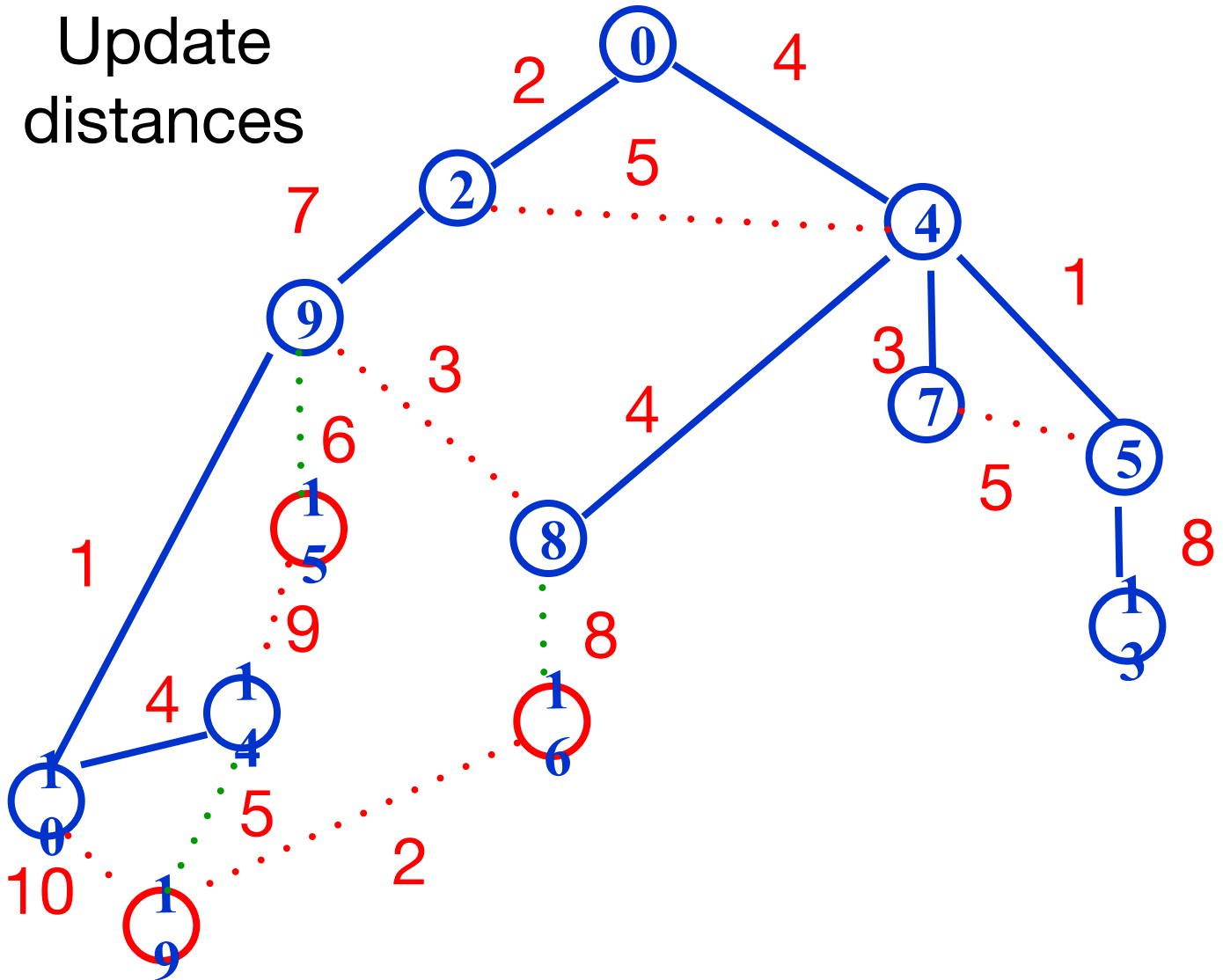
Dijkstra's Algorithm



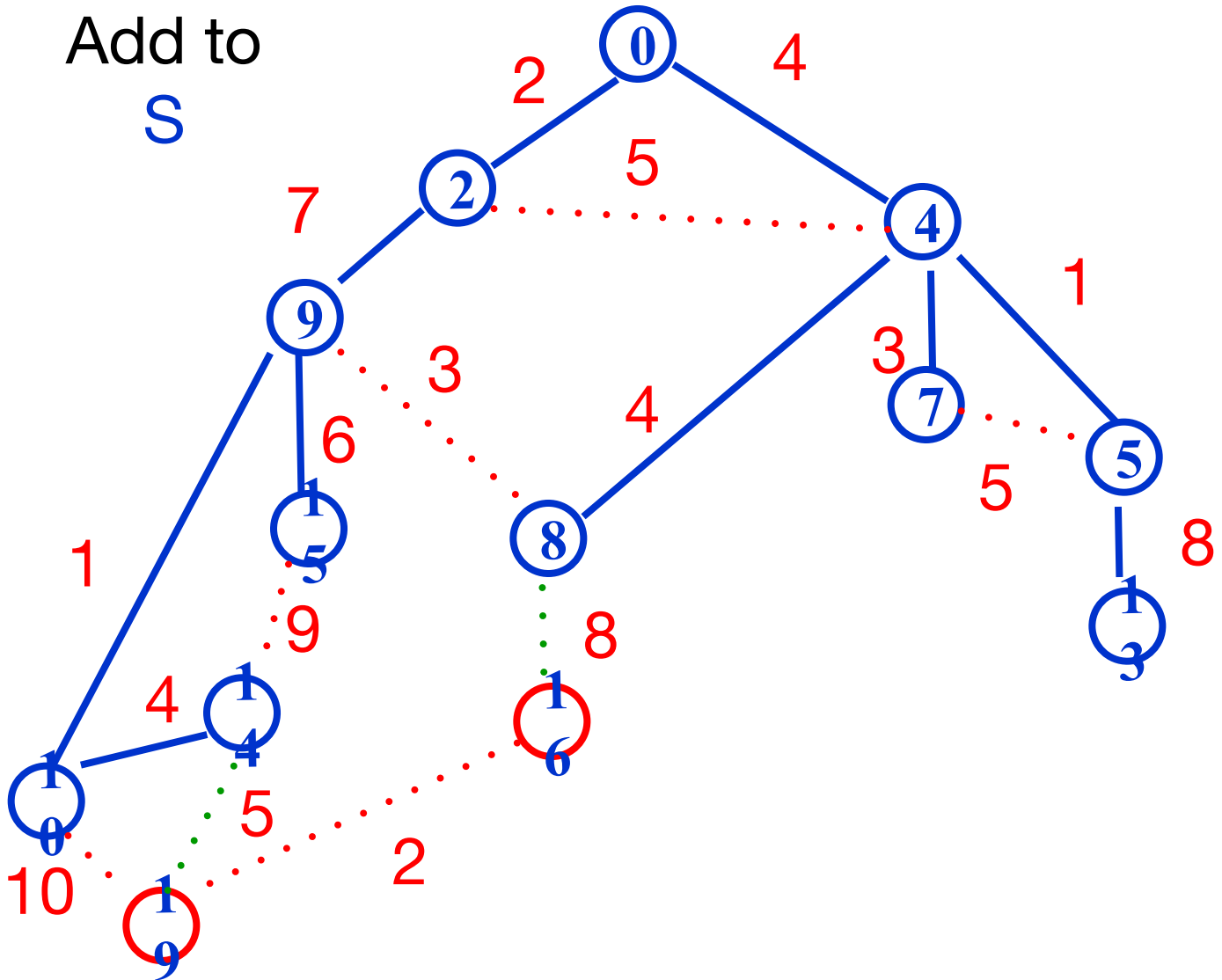
Dijkstra's Algorithm



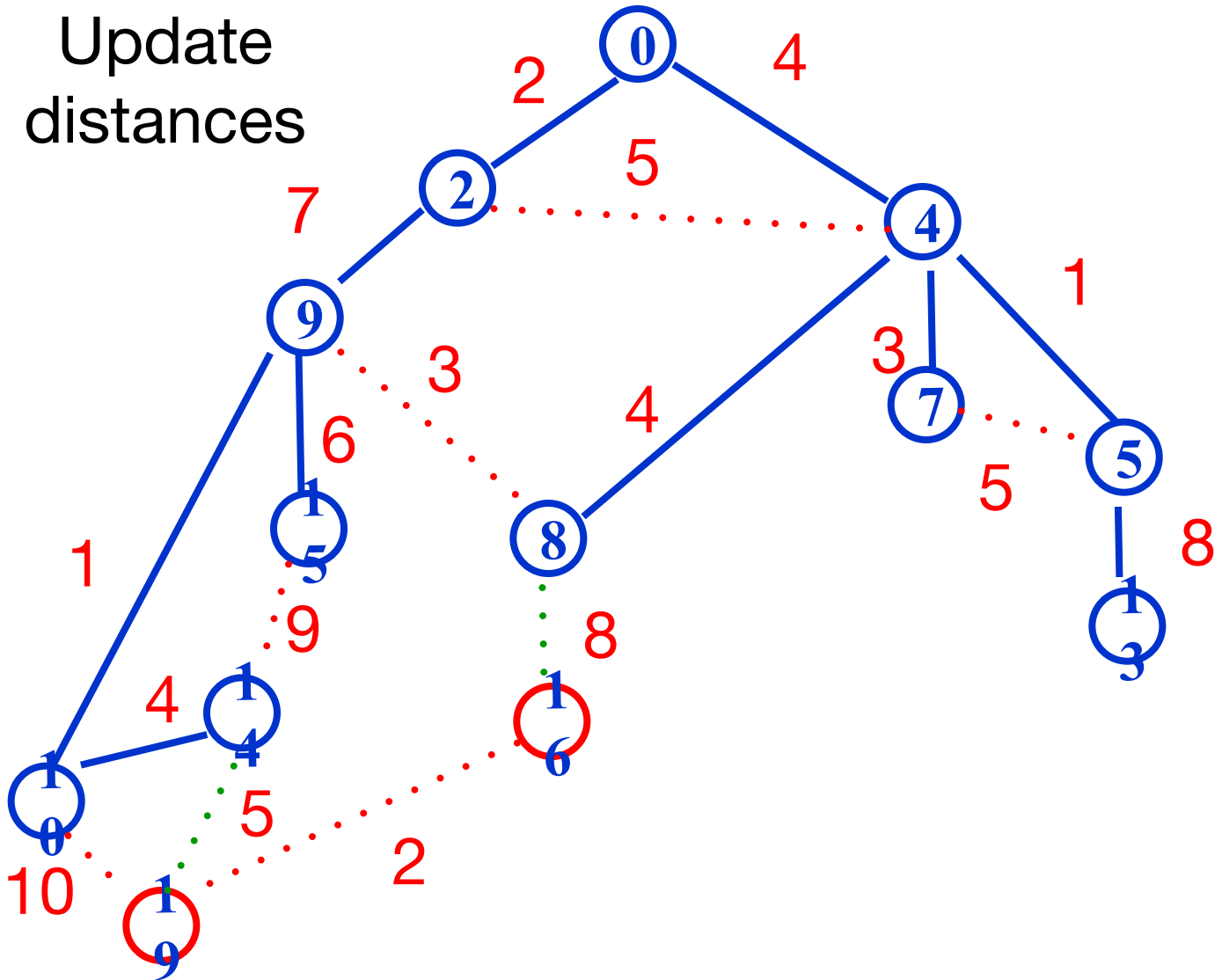
Dijkstra's Algorithm



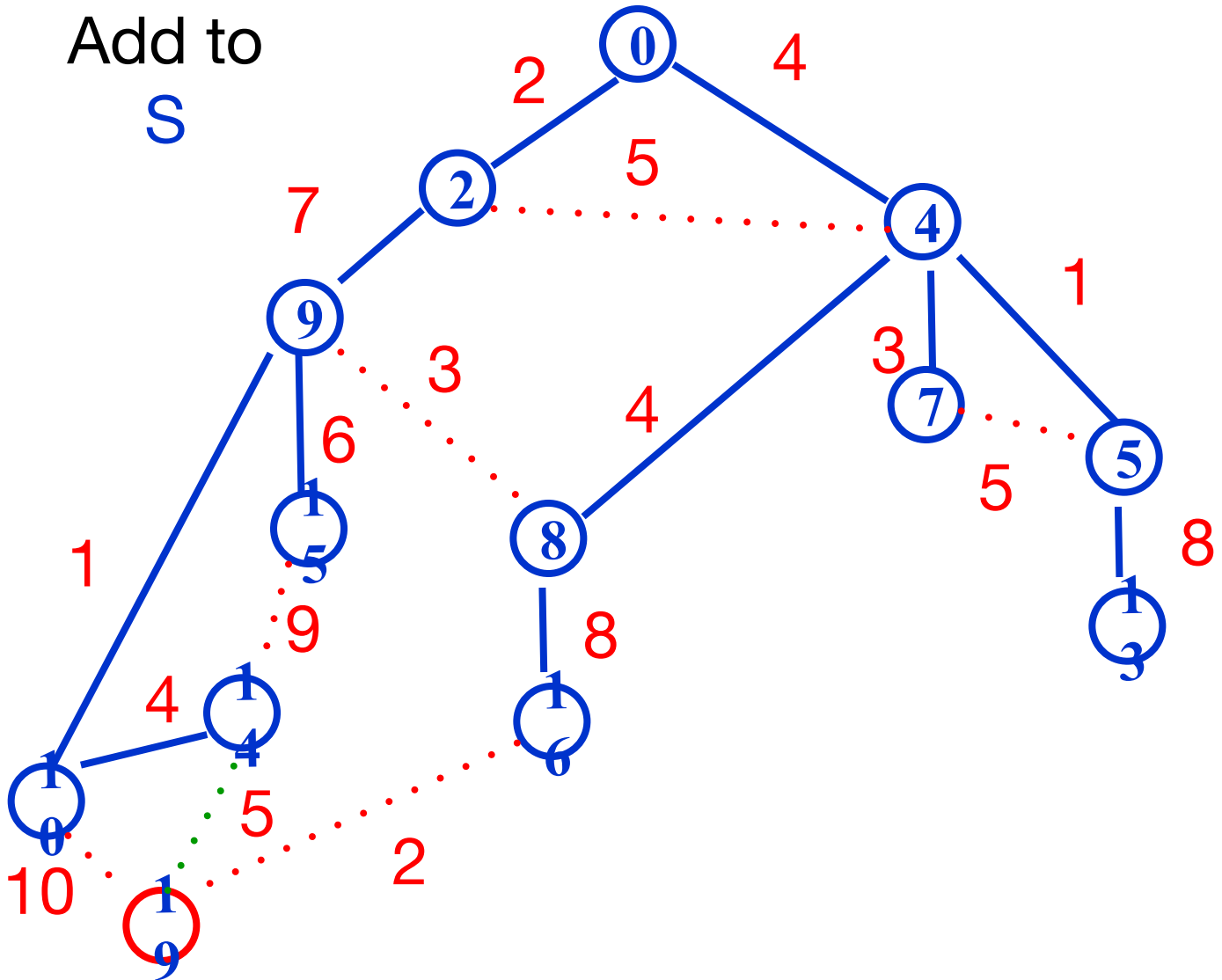
Dijkstra's Algorithm



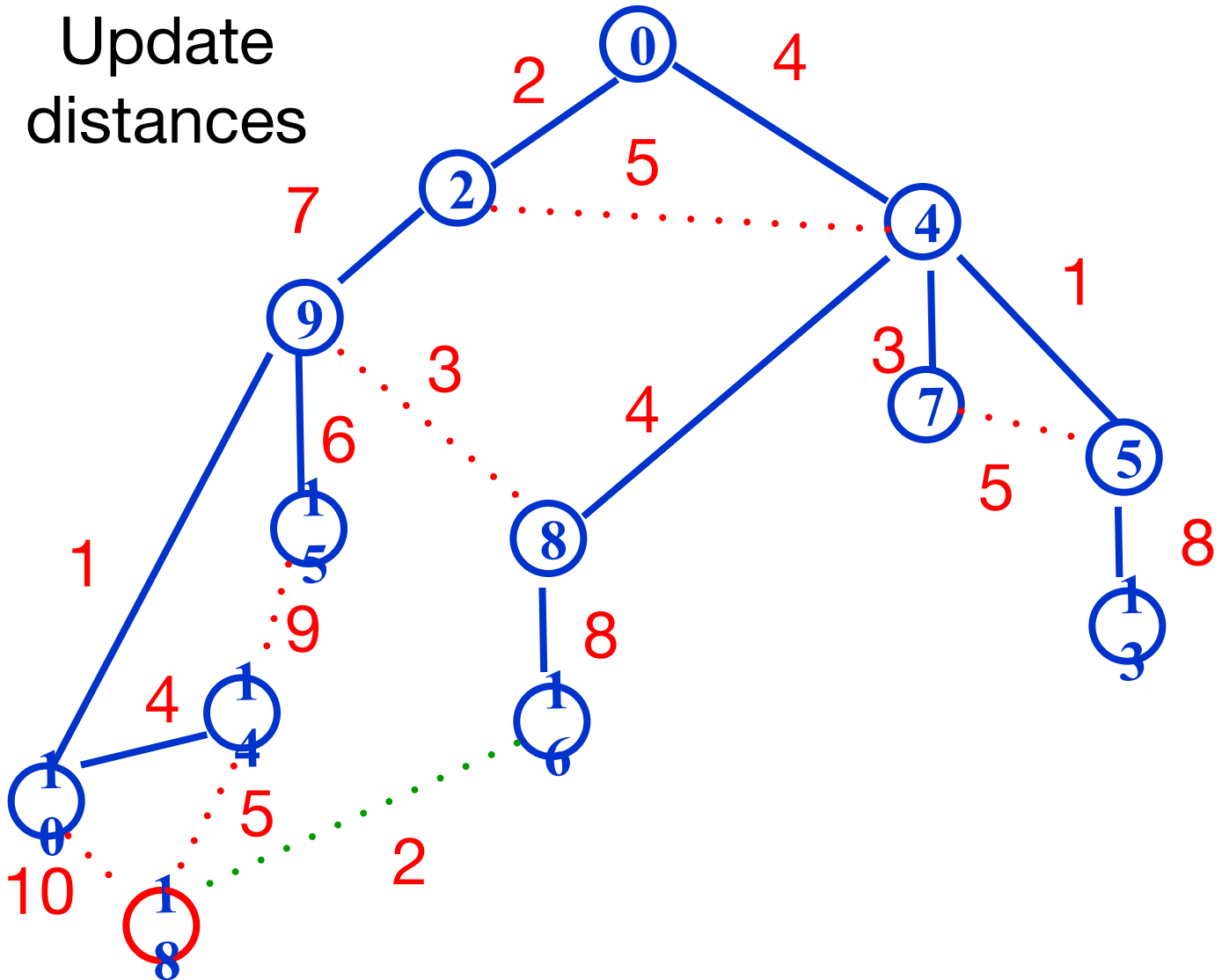
Dijkstra's Algorithm



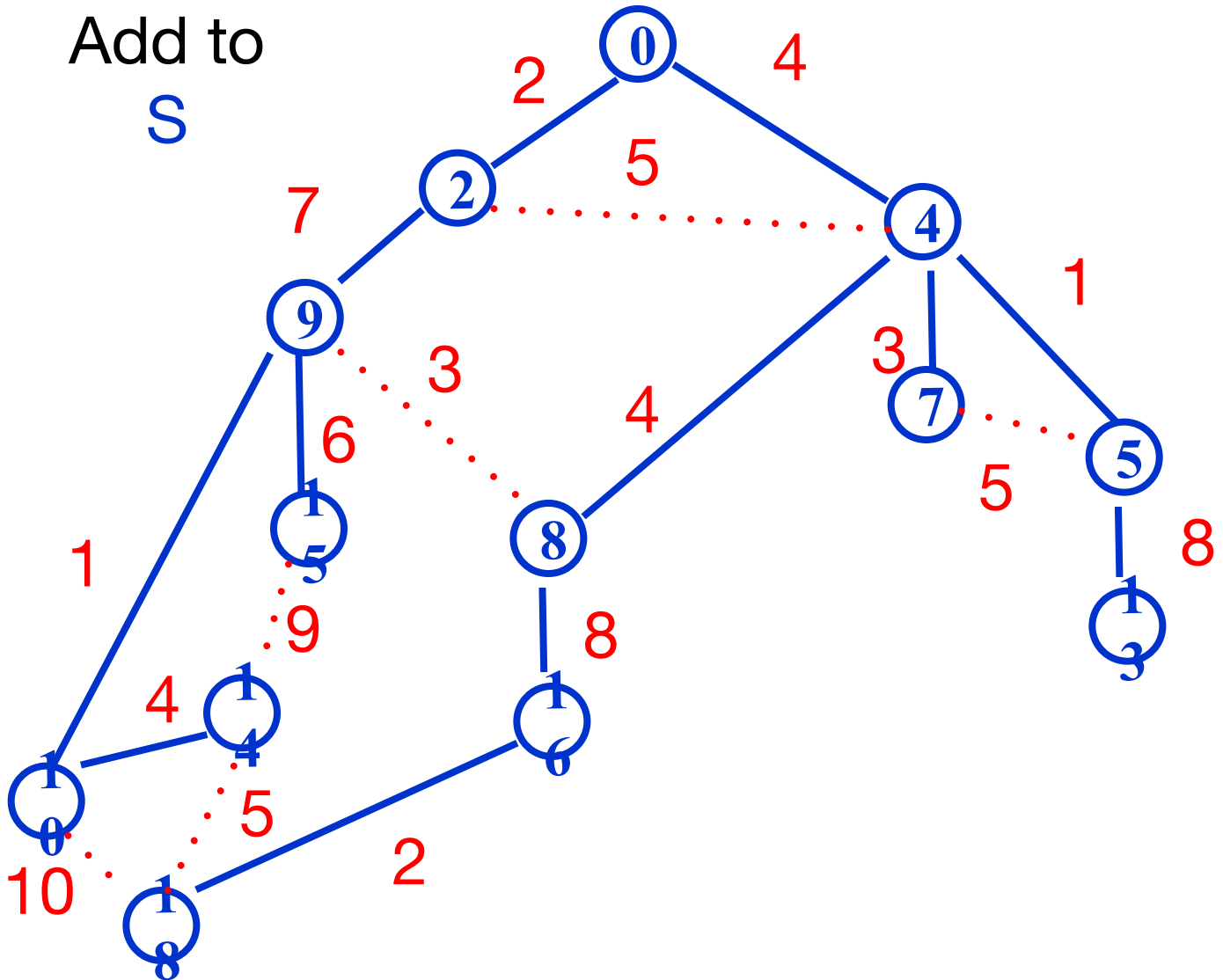
Dijkstra's Algorithm



Dijkstra's Algorithm



Dijkstra's Algorithm



Assume all edges have non-negative cost

Dijkstra's Algorithm

S = { }; d[s] = 0; d[v] = infinity for v != s

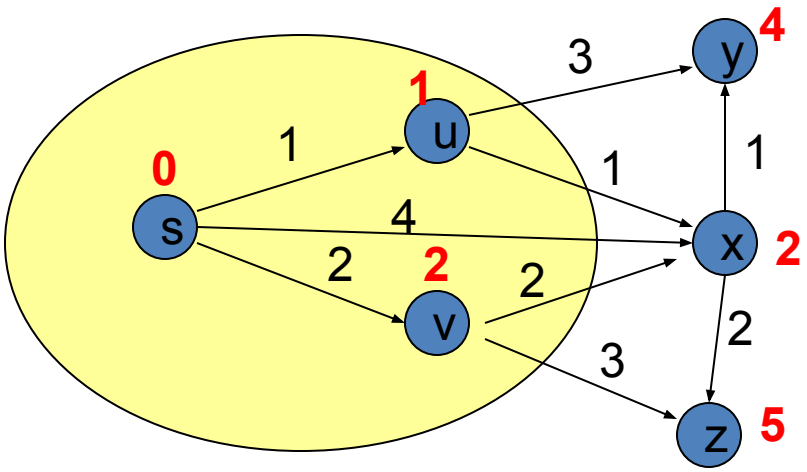
While S != V

    Choose v in V-S with minimum d[v]

    Add v to S

    For each w in the neighborhood of v

        d[w] = min(d[w], d[v] + c(v, w))

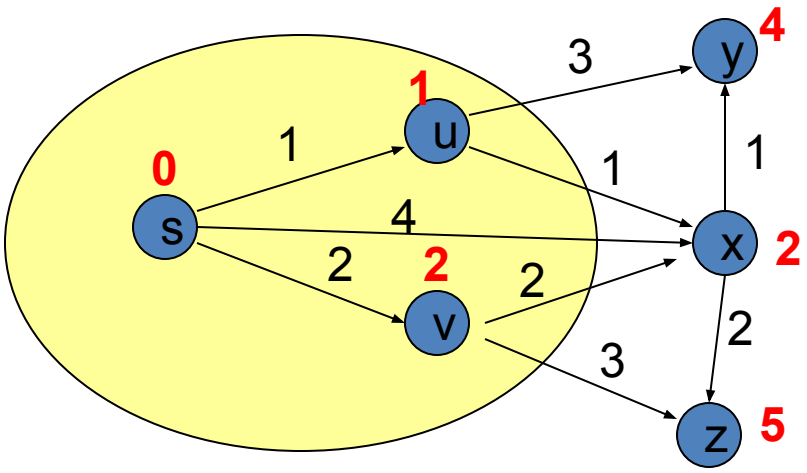


Assume all edges have non-negative cost

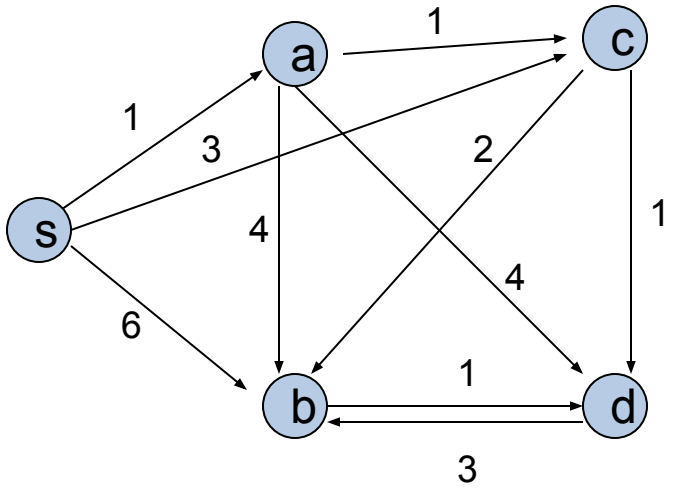
Dijkstra's Algorithm

```
S = { };    d[s] = 0;    d[v] = infinity for v != s
While S != V
    Choose v in V-S with minimum d[v]
    Add v to S
    For each  w in the neighborhood of v
        d[w] = min(d[w], d[v] + c(v, w))
```

Q: what's missing from this pseudo-code?



Simulate Dijkstra's algorithm (starting from s) on the graph



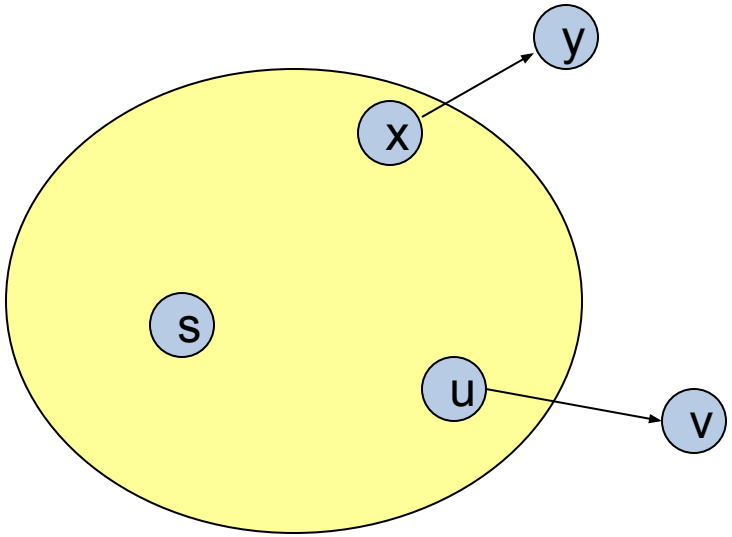
Round	Vertex Added	s	a	b	c	d
1						
2						
3						
4						
5						

Dijkstra's Algorithm as a greedy algorithm

- Elements committed to the solution by order of minimum distance
- Just like BFS

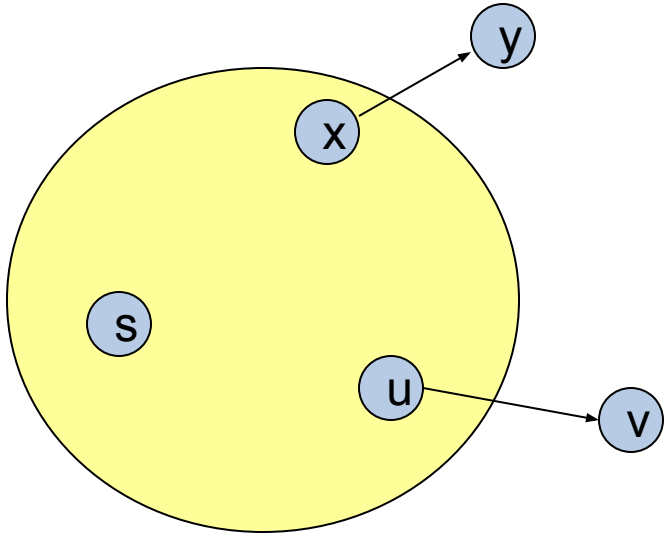
Correctness Proof

- Elements in S have the correct label
- Key to proof: when v is added to S, it has the correct distance label.



Proof

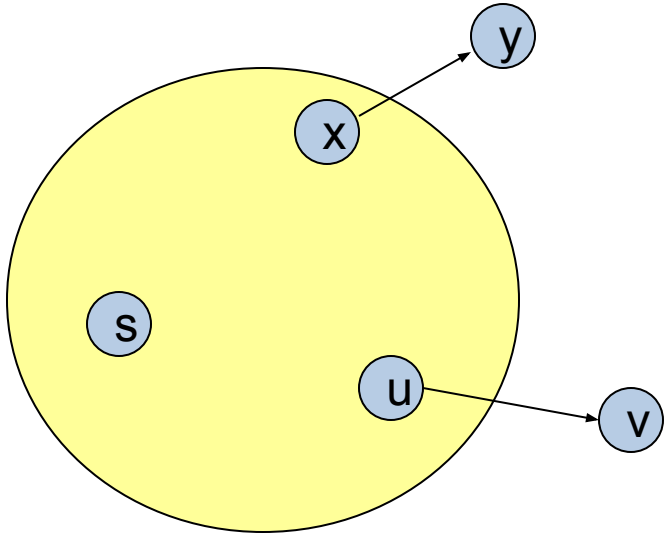
- Let v be a vertex in V-S with minimum d[v]
- Let P_v be a path of length d[v], with an edge (u,v)
- Let P be some other path to v. Suppose P first leaves S on the edge (x, y)
  - $P = P_{sx} + c(x,y) + P_{yv}$
  - $Len(P_{sx}) + c(x,y) >= d[y]$
  - $Len(P_{yv}) >= 0$
  - $Len(P) >= d[y] + 0 >= d[v]$



Proof

- Let v be a vertex in V-S with minimum d[v]
- Let P_v be a path of length d[v], with an edge (u,v)
- Let P be some other path to v. Suppose P first leaves S on the edge (x, y)
  - $P = P_{sx} + c(x,y) + P_{yv}$
  - $Len(P_{sx}) + c(x,y) >= d[y]$
  - $Len(P_{yv}) >= 0$
  - $Len(P) >= d[y] + 0 >= d[v]$

Notice: this is another exchange argument

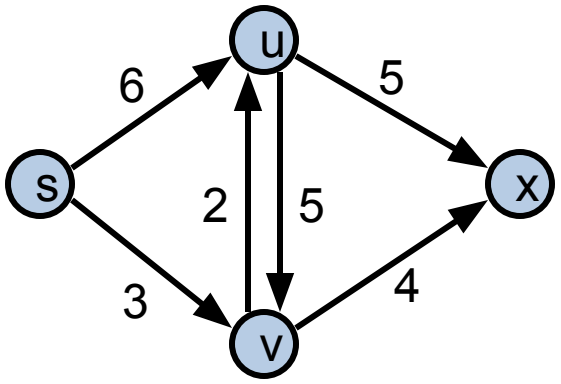


Negative Cost Edges

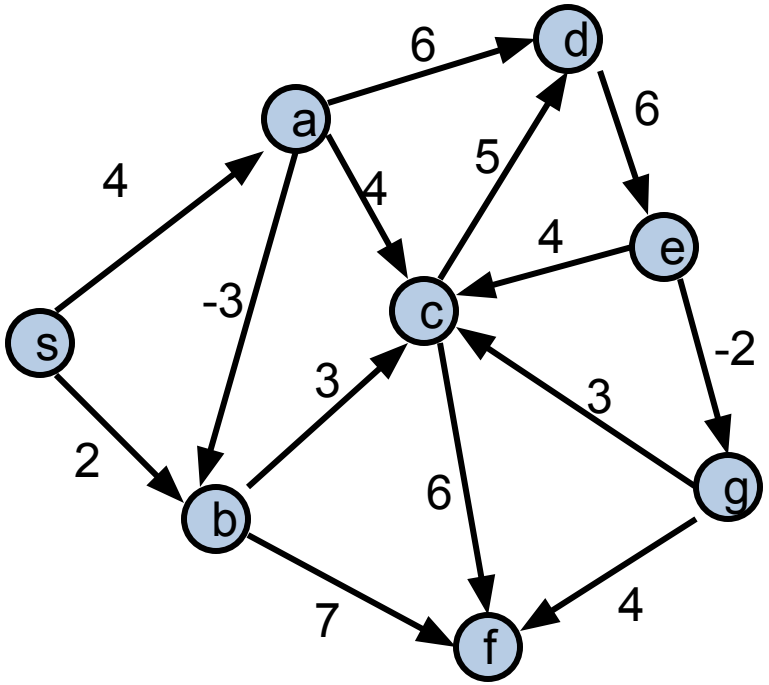
- Draw a small example with a negative cost edge and show that Dijkstra's algorithm fails on this example

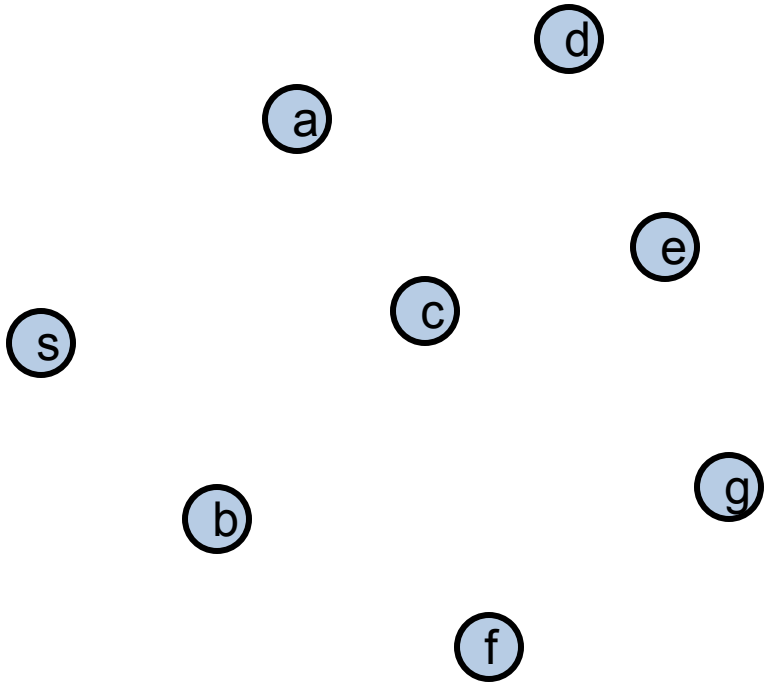
Bottleneck Shortest Path

- Define the bottleneck distance for a path to be the maximum cost edge along the path



Compute the bottleneck shortest paths





How do you adapt Dijkstra's algorithm to handle bottleneck distances

- Does the correctness proof still apply?