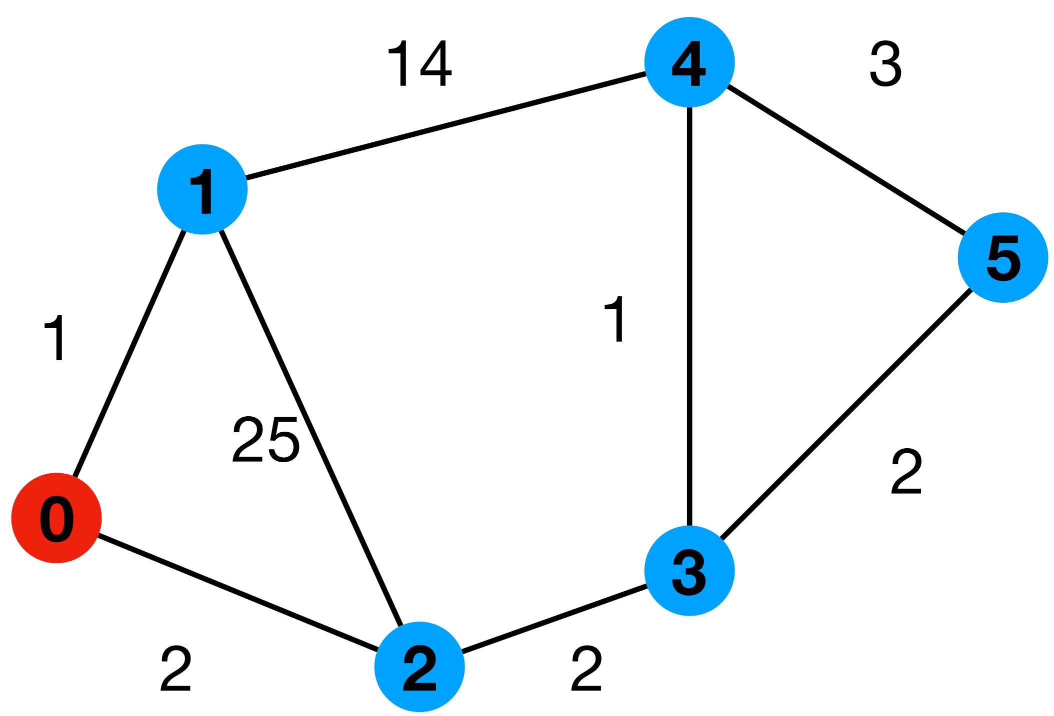
Shortest Paths

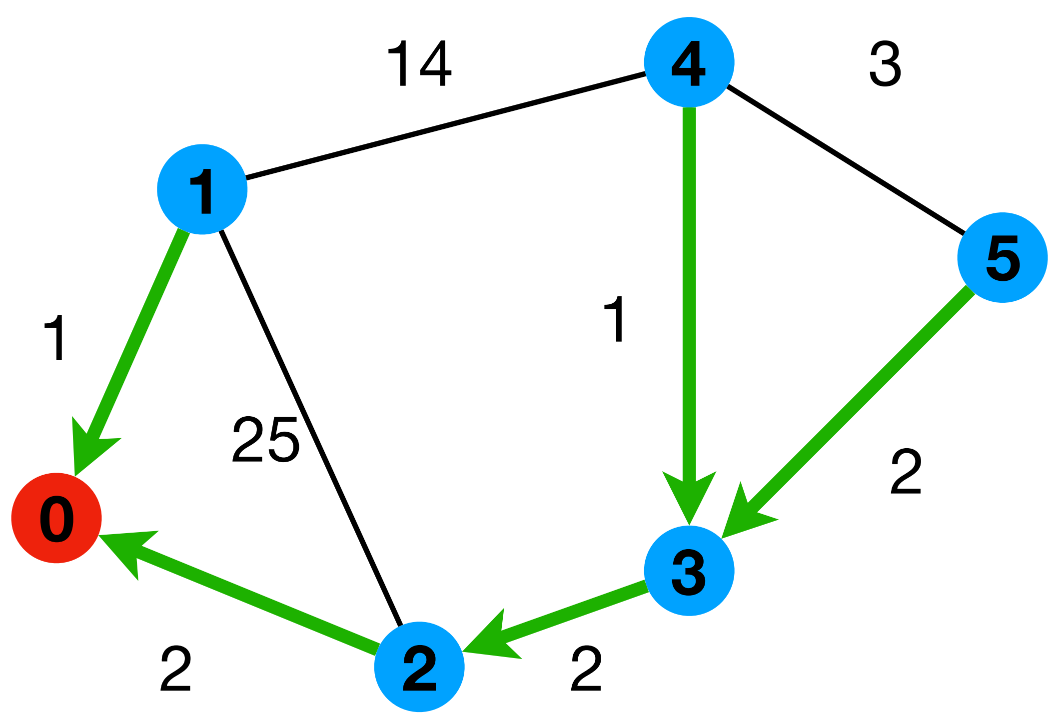
Single-Source Shortest Paths

- **Input:** Weighted graph *(G, w)*. Source vertex *s*.
  Output: Shortest path tree encoding shortest path from s to every other node.



Single-Source Shortest Paths

- **Input:** Weighted graph *(G, w)*. Source vertex *s*.
  Output: Shortest path tree encoding shortest path from s to every other node.




```
tree: [ -1,0,0,2,3,3 ]
# idx    0 1 2 3 4 5

# Quick and dirty recover path
# from root to t

def getPath(tree, t)
  list = []
  while (t != -1)
    list.add(t)
    t = tree[t]
  end
  list.reverse()
  return list
end
```

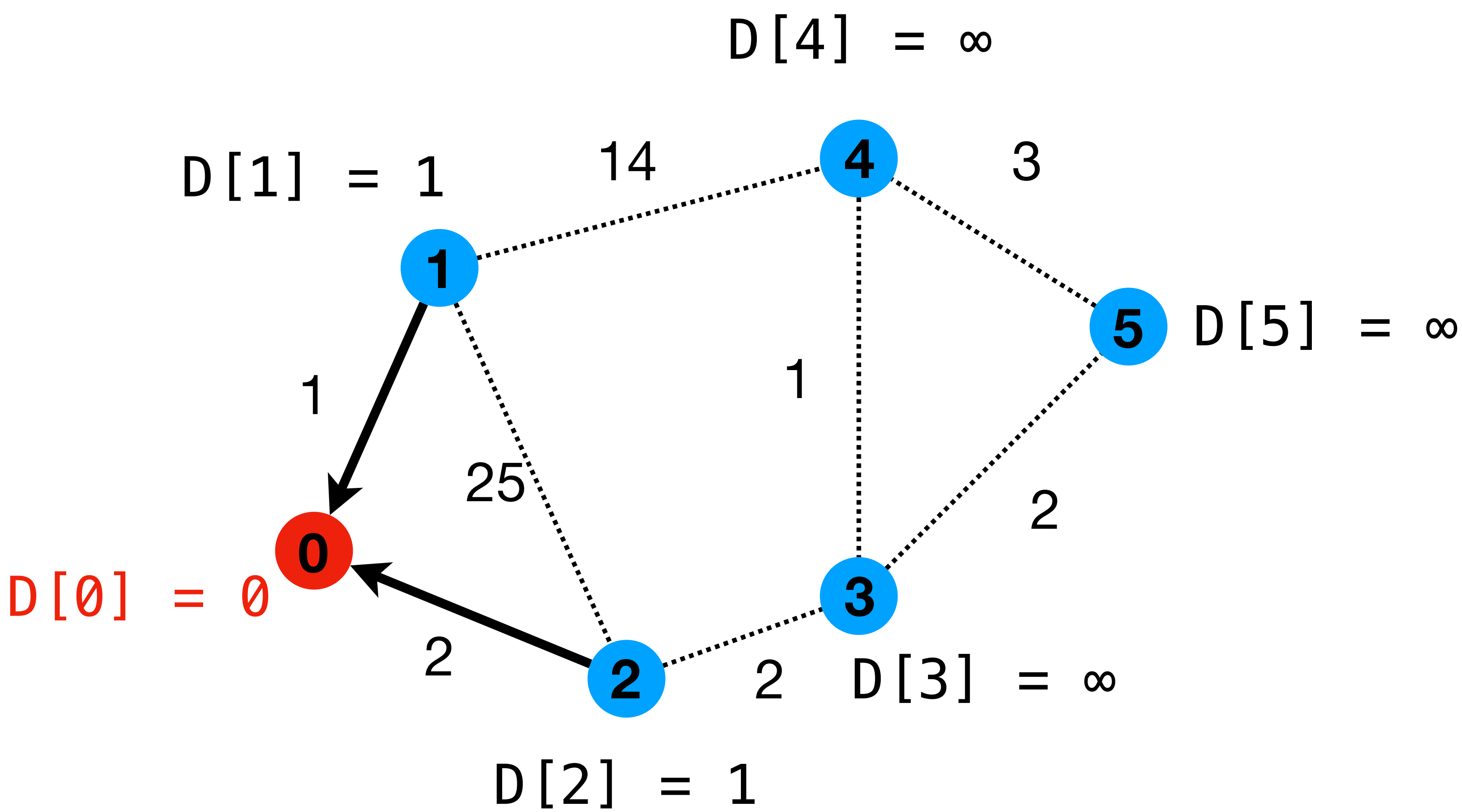
Dijkstra’s Algorithm

- Solution to the Single-Source shortest path.
    - A greedy algorithm (but non-trivial greedy algorithm)
    - Uses a Priority Queue
    - Running time is $O((V+E) \log V)$
- If your graph is un-weighted, *just use breadth-first search.*

Dijkstra’s Algorithm

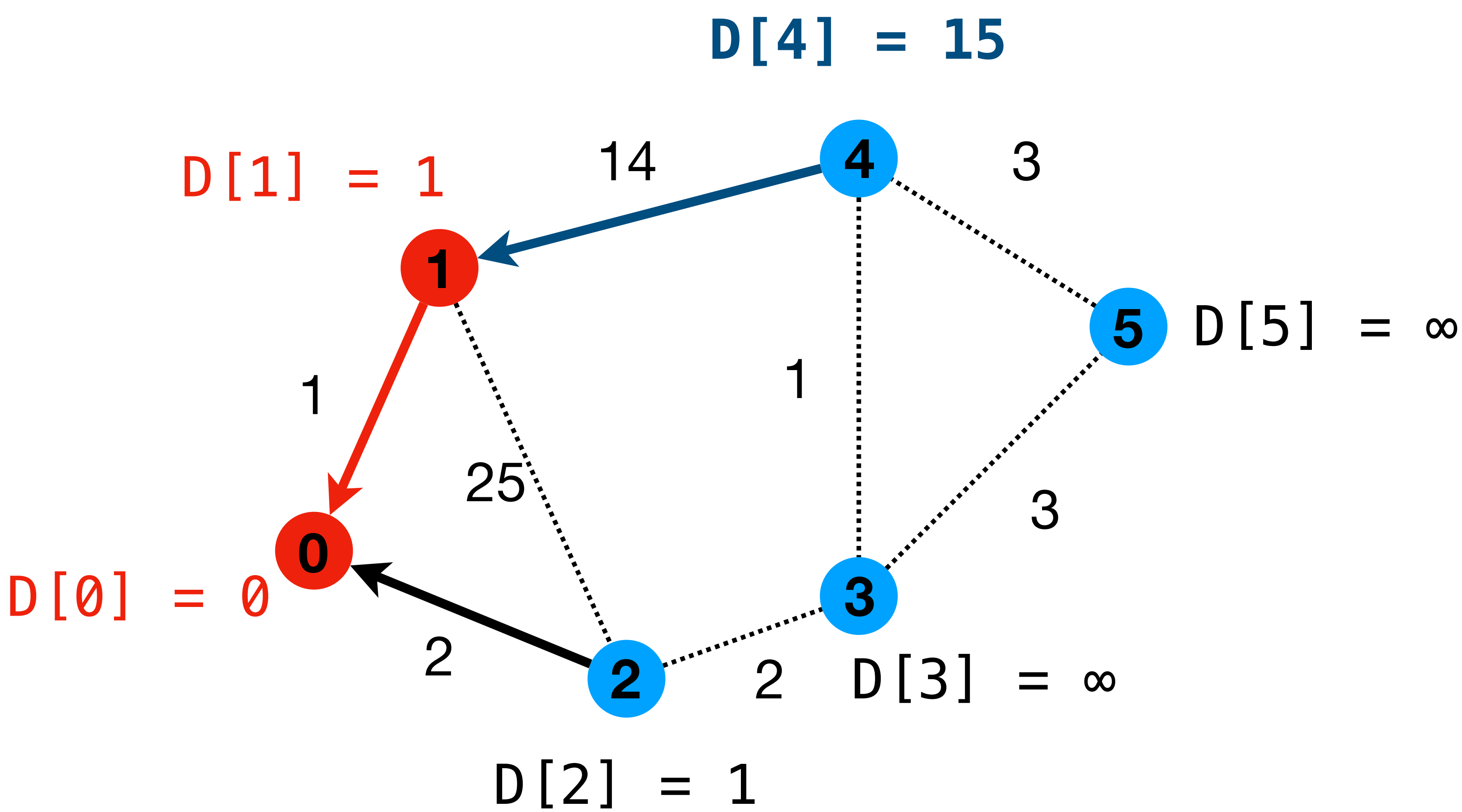
- Main Idea:
    - Grow the shortest path tree from the start vertex.
    - Maintain the “best edge” connecting every vertex not in the tree to the growing shortest path tree along with its distance across that edge back to the root.
    - Each iteration, add whatever vertex is closest to the current tree to the current tree.

Dijkstra’s Algorithm



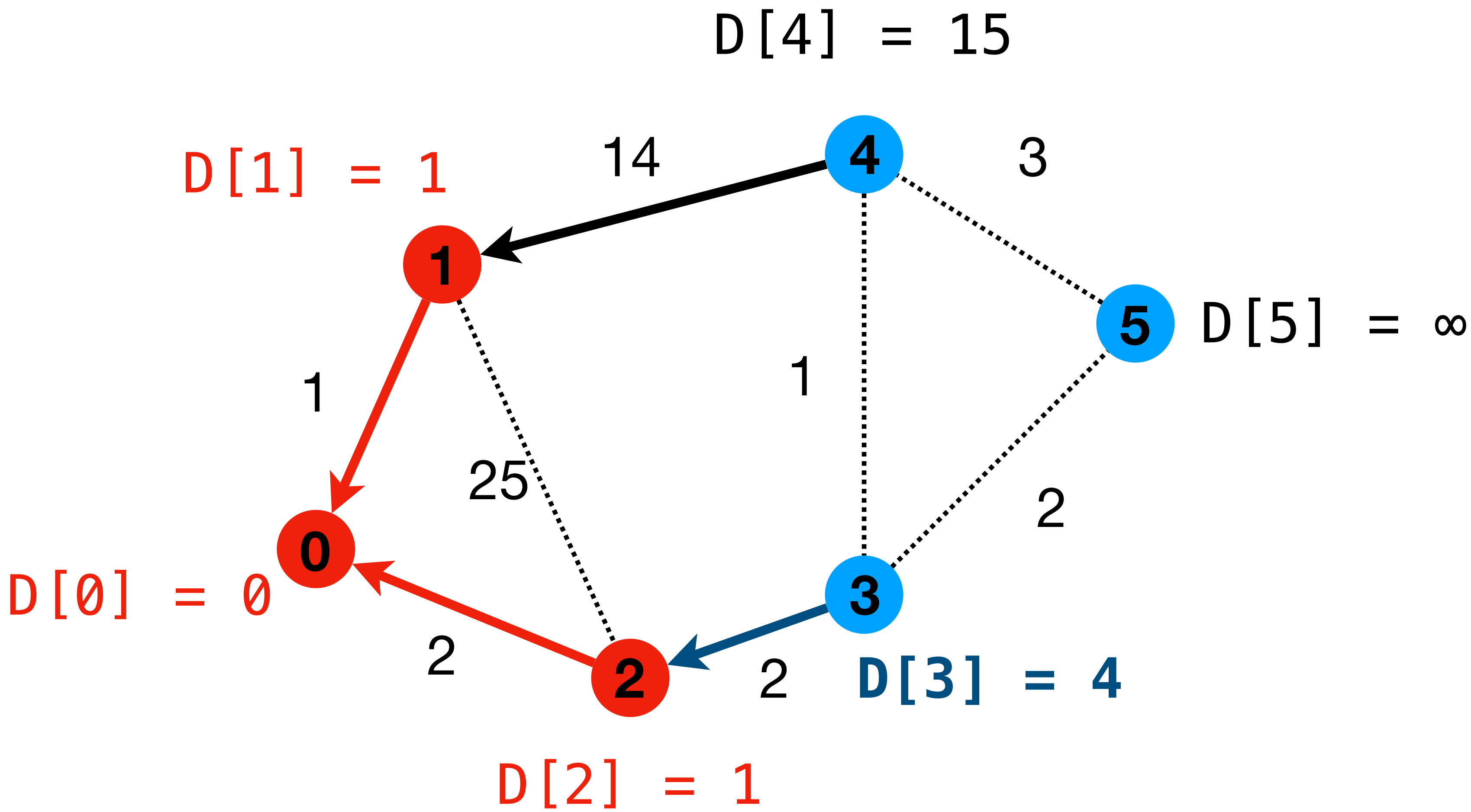

```
   D: [  0  1  2  ∞  ∞  ∞ ]
Tree: [ -1  0  0 -1 -1 -1 ]
```

Dijkstra's Algorithm



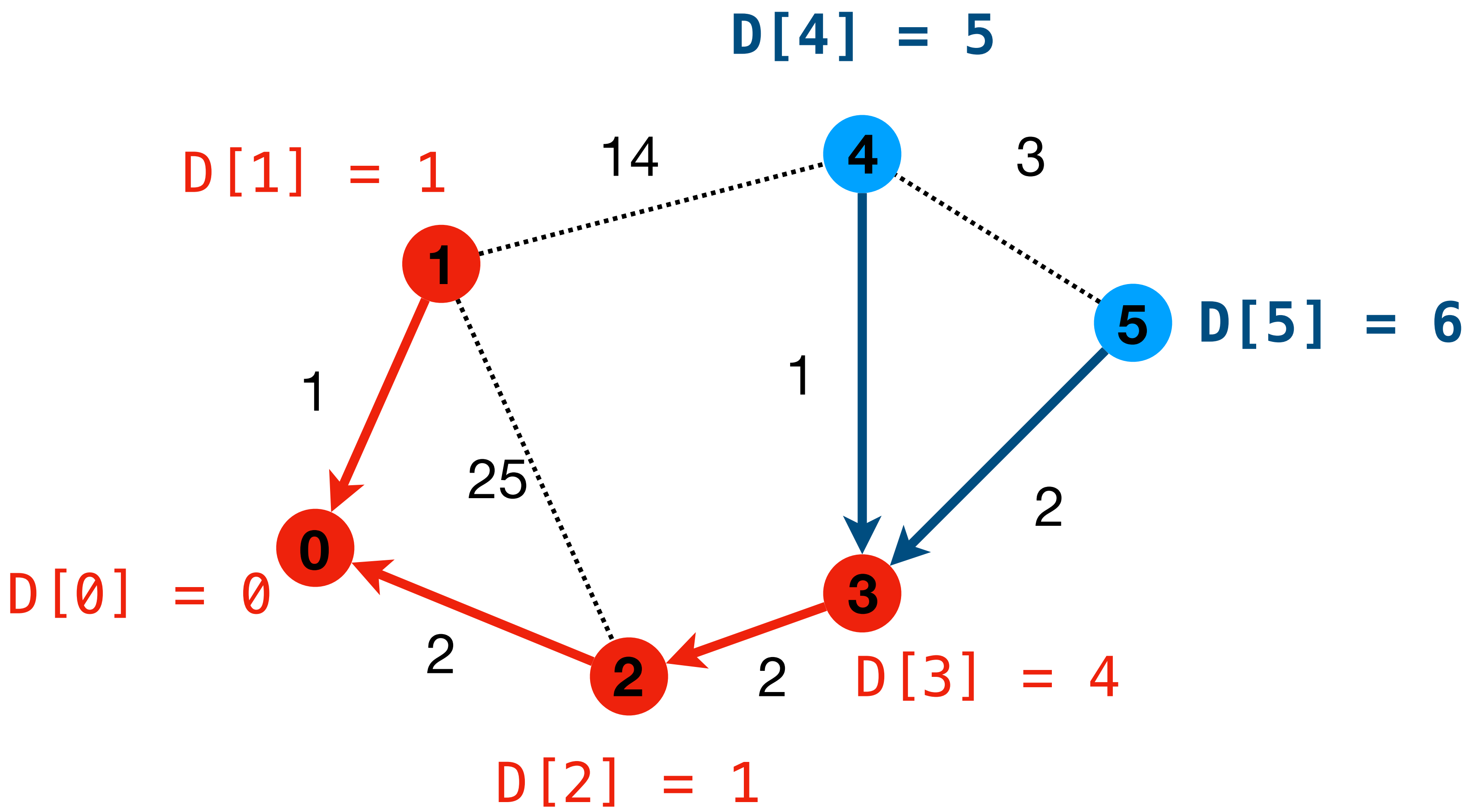

```
   D: [  0  1  2  ∞ 15  ∞ ]
Tree: [ −1  0  0 −1  1 −1 ]
```

Dijkstra's Algorithm



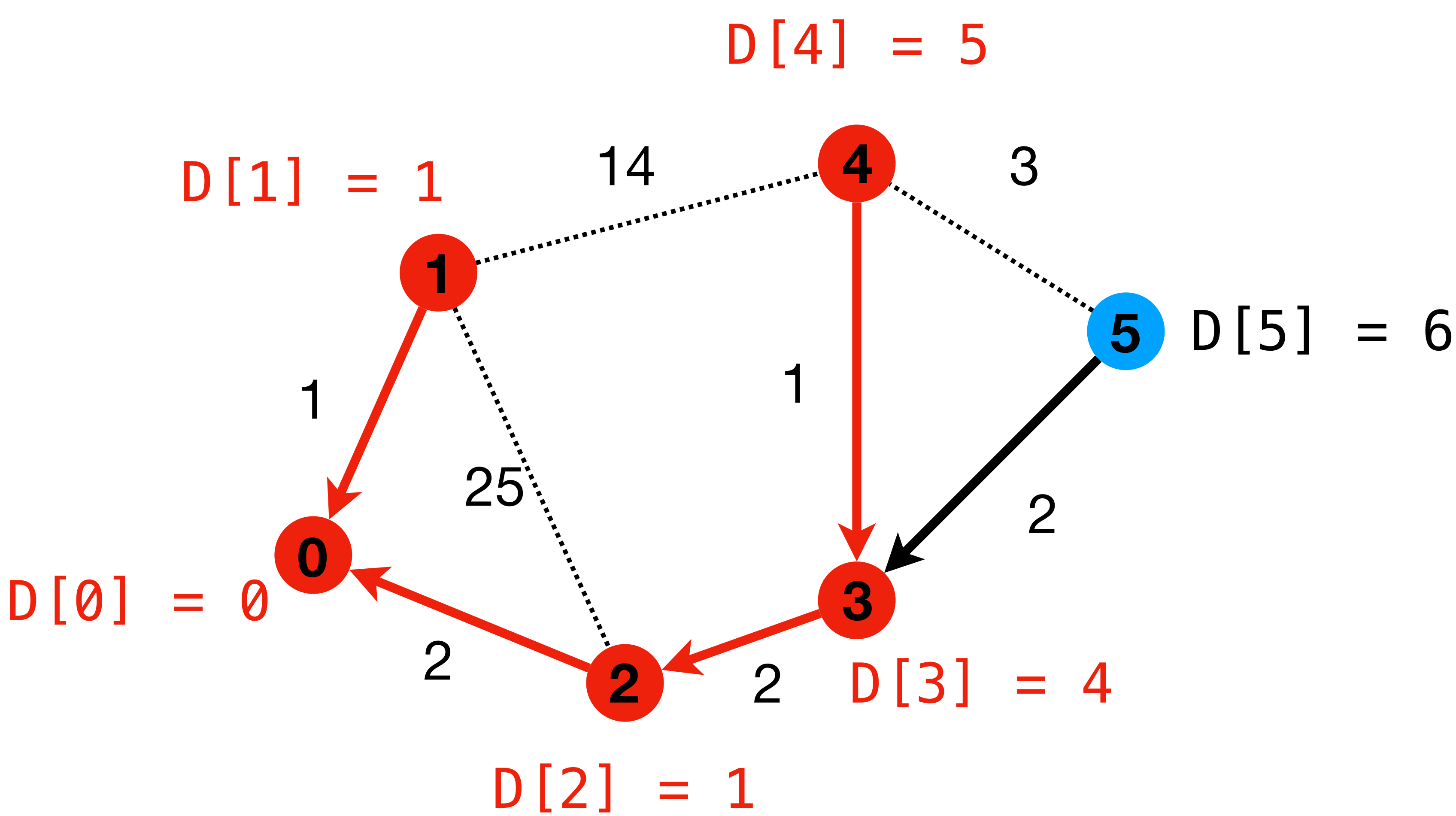

```
   D: [  0  1  2  4 15  ∞ ]
Tree: [ -1  0  0  2  1 -1 ]
```

Dijkstra’s Algorithm



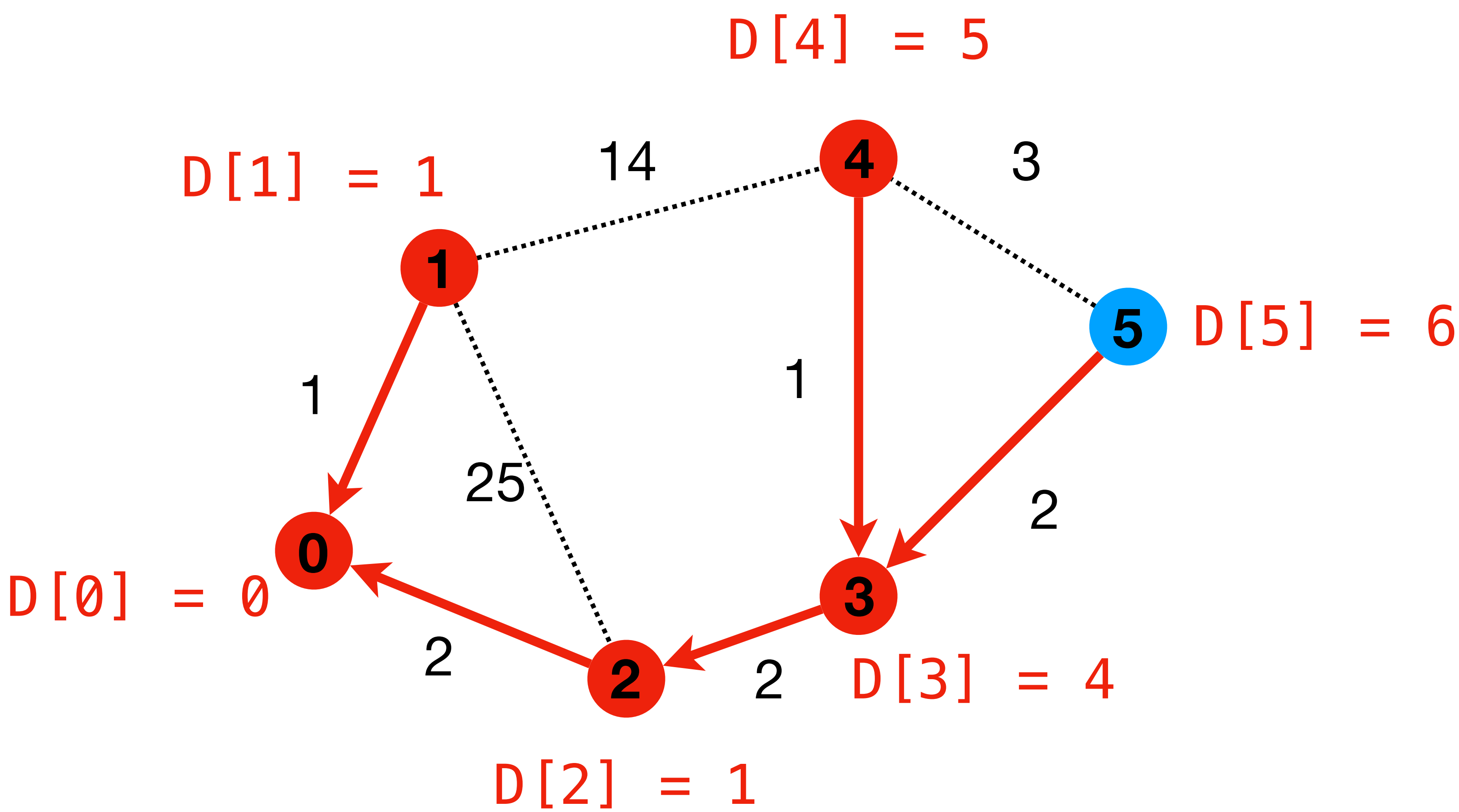

```
   D: [  0  1  2  4  5  6 ]
Tree: [ −1  0  0  2  3  3 ]
```

Dijkstra’s Algorithm



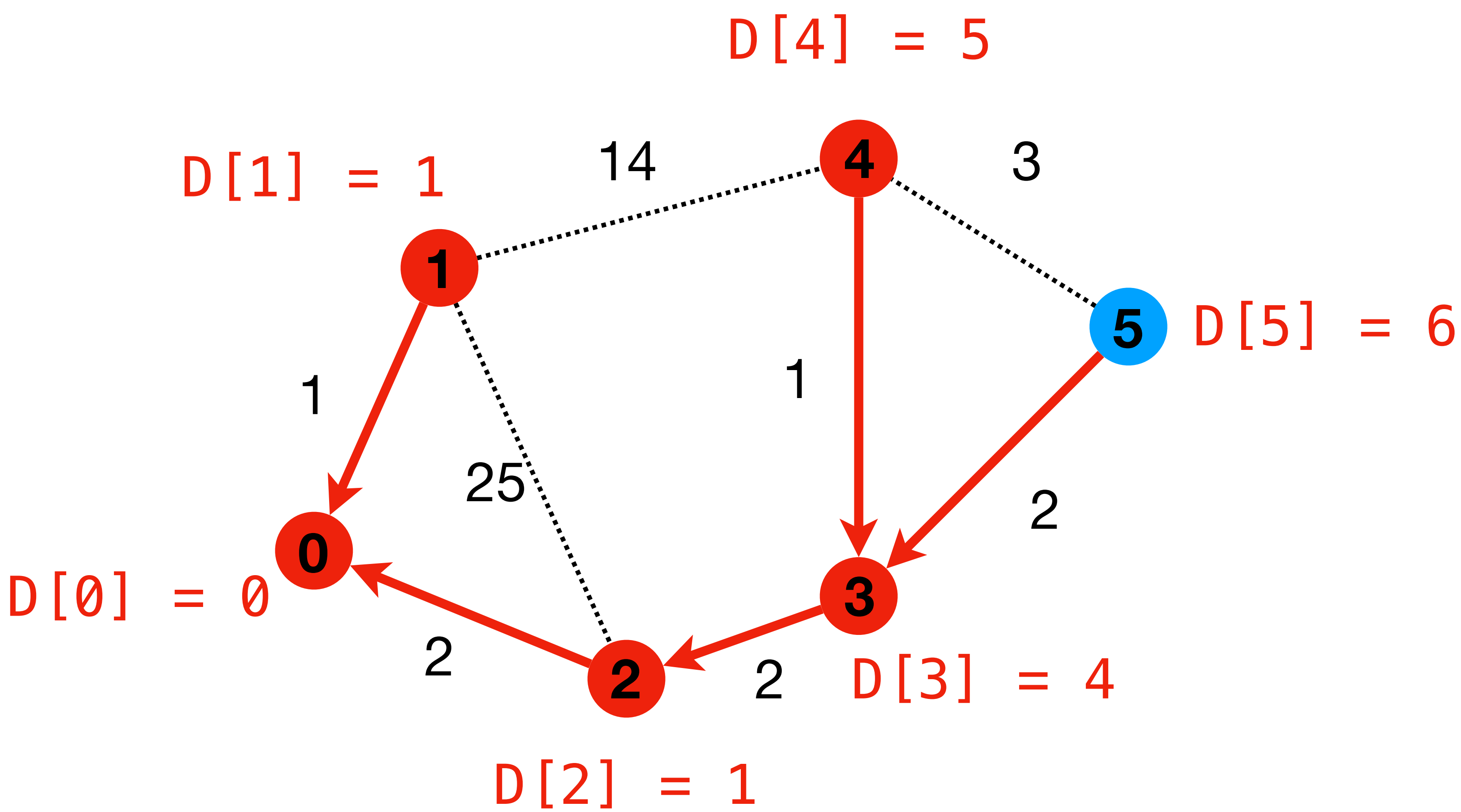

```
   D: [  0  1  2  4  5  6 ]
Tree: [ −1  0  0  2  3  3 ]
```

Dijkstra's Algorithm




```
   D: [  0  1  2  4  5  6 ]
Tree: [ −1  0  0  2  3  3 ]
```

Dijkstra’s Algorithm




```
   D: [  0  1  2  4  5  6 ]
Tree: [ −1  0  0  2  3  3 ]
```

Implementation Details

- For every vertex v not in the tree, we maintain a MIN-QUEUE on the vertices, with value given by D[v].
- Every iteration of the loop, we take the current minimum vertex v and add it to the tree.
    - This requires looping over its neighbors and updating their best estimate to the tree if adding the new vertex gives a better estimate. This also requires an updated for each such edge in the priority queue.