EN.530.663
Robot Motion Planning
Spring 2022

Dijkstra's Algorithm

Jin Seob Kim, Ph.D.

Senior Lecturer

ME Dept., LCSR,

Johns Hopkins University

Overview

- Goal: Determine the shortest path between an initial and goal node in a graph
- Optimal greedy search algorithm
- Necessary Graph Properties:
  - Connected – i.e., there are no stand alone nodes or groups of nodes
  - Nonnegative weights on every edge (The Bellman Ford Algorithm can handle negative weights)

Dijkstra's Algorithm

1. Initialize the distance to the initial node as 0 and initialize temporary distances for the remaining nodes to ∞. Additionally, initialize the closest neighbor as NULL.
2. Set the initial node n_{init} as current and create a set of all unvisited nodes $U = \{n_1, n_2, \dots, n_p\}$, where p is the total number of nodes.
3. For the current node, calculate the distance to each unvisited neighbor (if one exists) by summing the edge weights from the initial node to the unvisited neighbor. If the calculated distance to the unvisited neighbor is smaller than the current assigned distance, update the distance of the unvisited neighbor and change the closest neighbor to the current node.
4. When all unvisited neighbors have been visited, mark the current node as visited and, if applicable, remove it from the unvisited set.
5. If the goal node n_{goal} has been marked visited, stop and return the shortest path.
6. If n_{goal} has not been visited, mark the node with the smallest tentative distance as current and repeat Steps 3 – 6.

Pseudocode

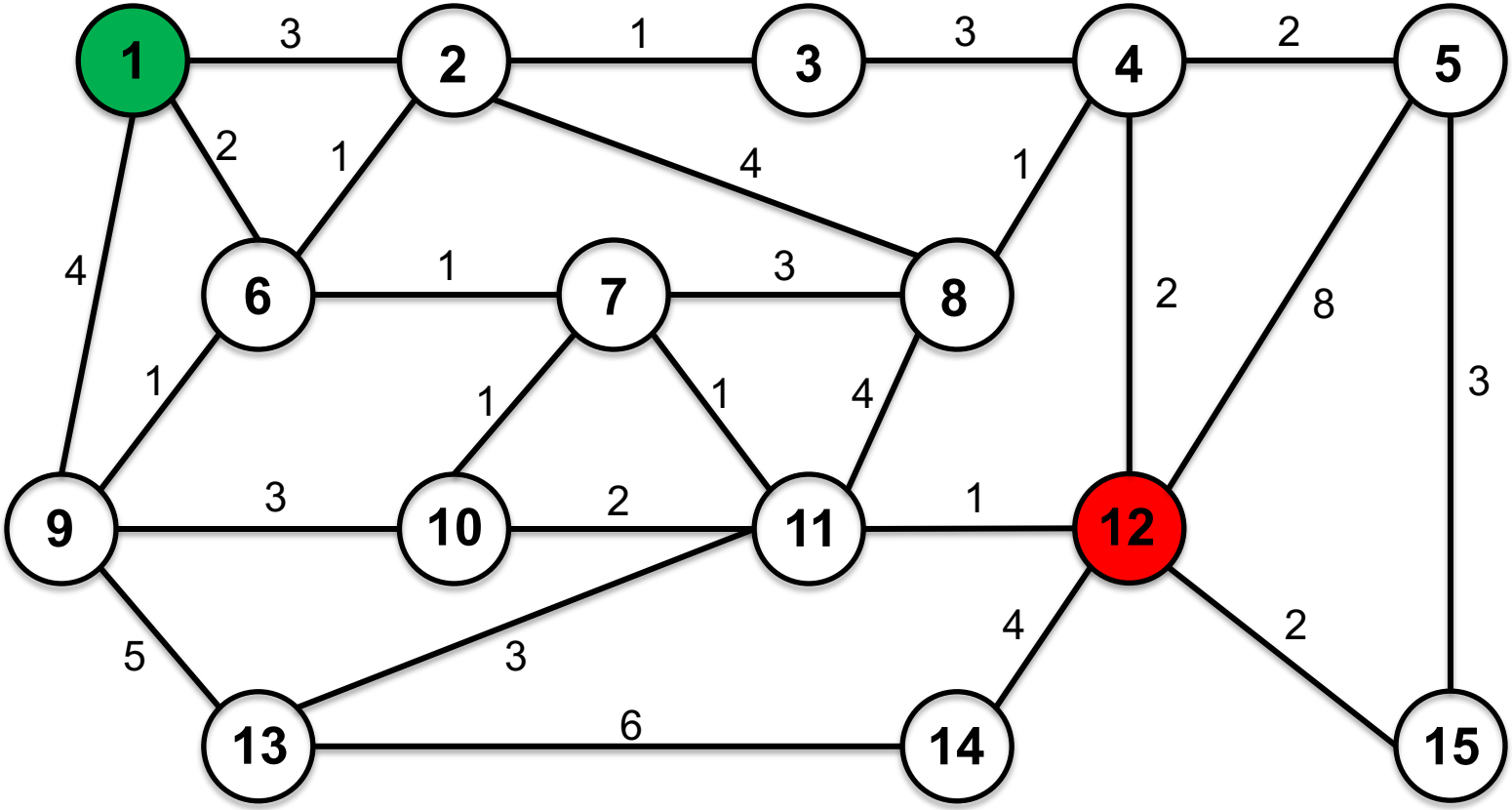
function Dijkstra(graph, n_{init}, n_{goal})
  for each node n in *graph* **do**:
    dist[n] $\doteq \infty$
    prev[n] $\doteq$ NULL
  end for
  dist[n_{init}] $\doteq 0$
  $U \doteq$ set of all nodes
  while $n_{goal} \in U$ **do**:
    $C \doteq$ node in U with smallest distance
    remove C from U
    for each neighbor v of C **do**:
      alt $\doteq$ dist[C] + distance between C and v
      if alt $<$ dist[v] **then**:
        dist[v] $\doteq$ alt
        prev[v] $\doteq C$
      end if
    end for
  end while
  return dist[n_{goal}], prev[n_{goal}]

Note: You have to write an additional code for constructing a shortest path mainly based on "prev" (and "dist")

Loosely borrowed from: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html

Example (Undirected Graph)

- Graph with weights
  - In the inputs, G denotes the weighted adjacency matrix

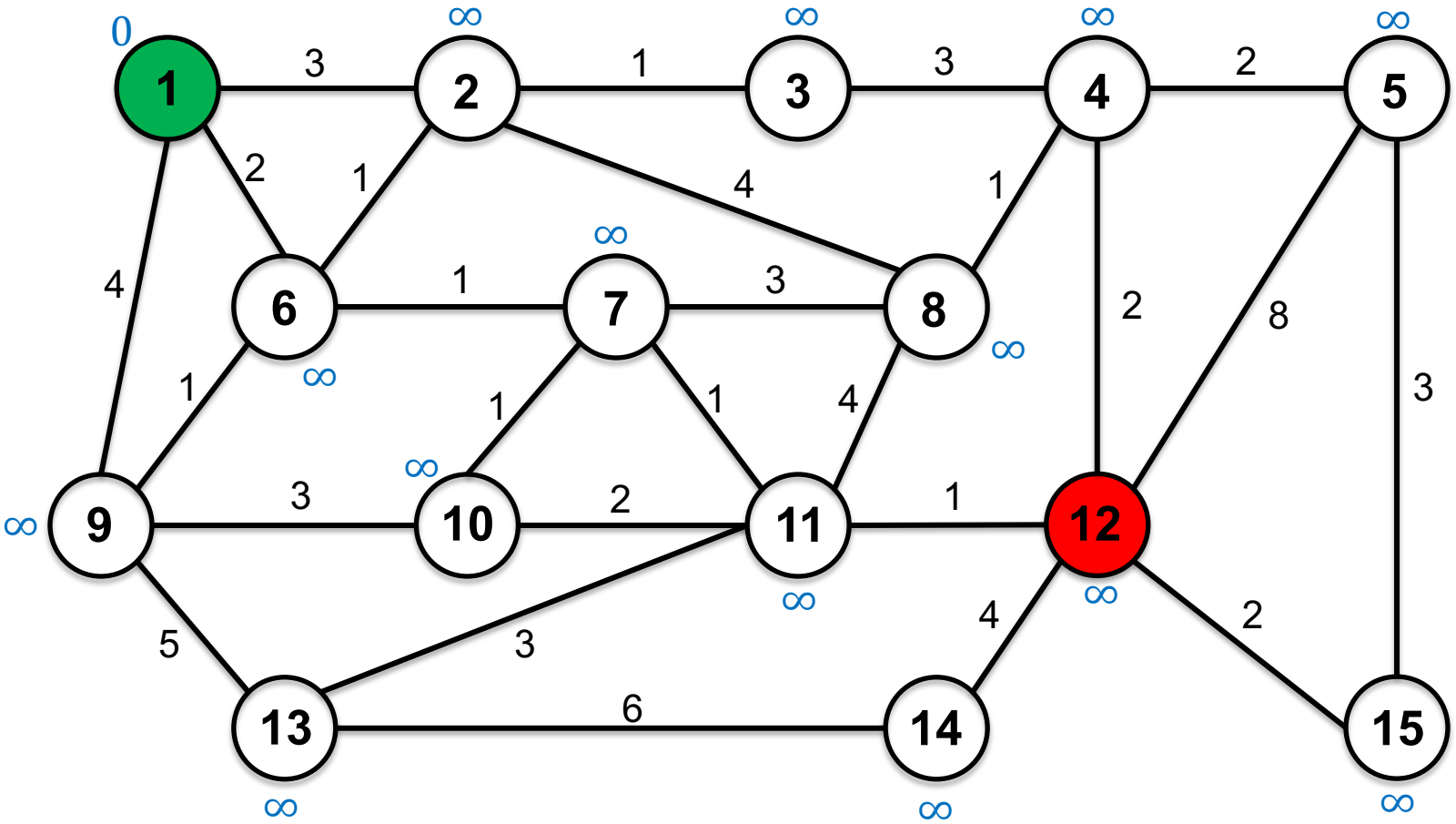


Example (Undirected Graph)

Initialize the distance to the initial node as 0 and initialize distances for the remaining nodes to ∞.
Additionally, initialize the closest neighbor as NULL.

The set of unvisited nodes U = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

Blue-colored numbers → dist

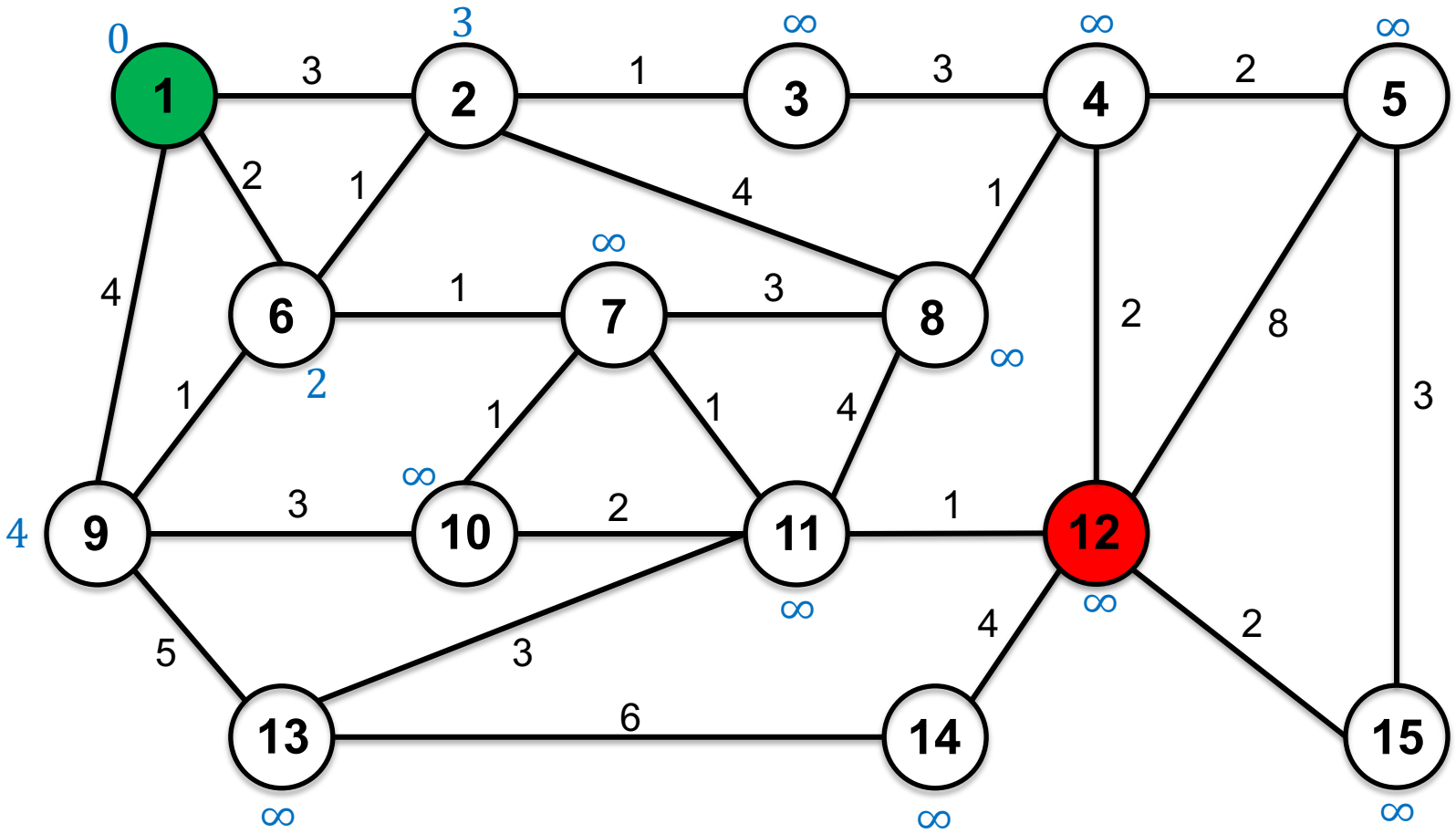


Example (Undirected Graph)

Current or closed node, C = 1, and U = {2,3,4,5,6,7,8,9,10,11,12,13,14,15}

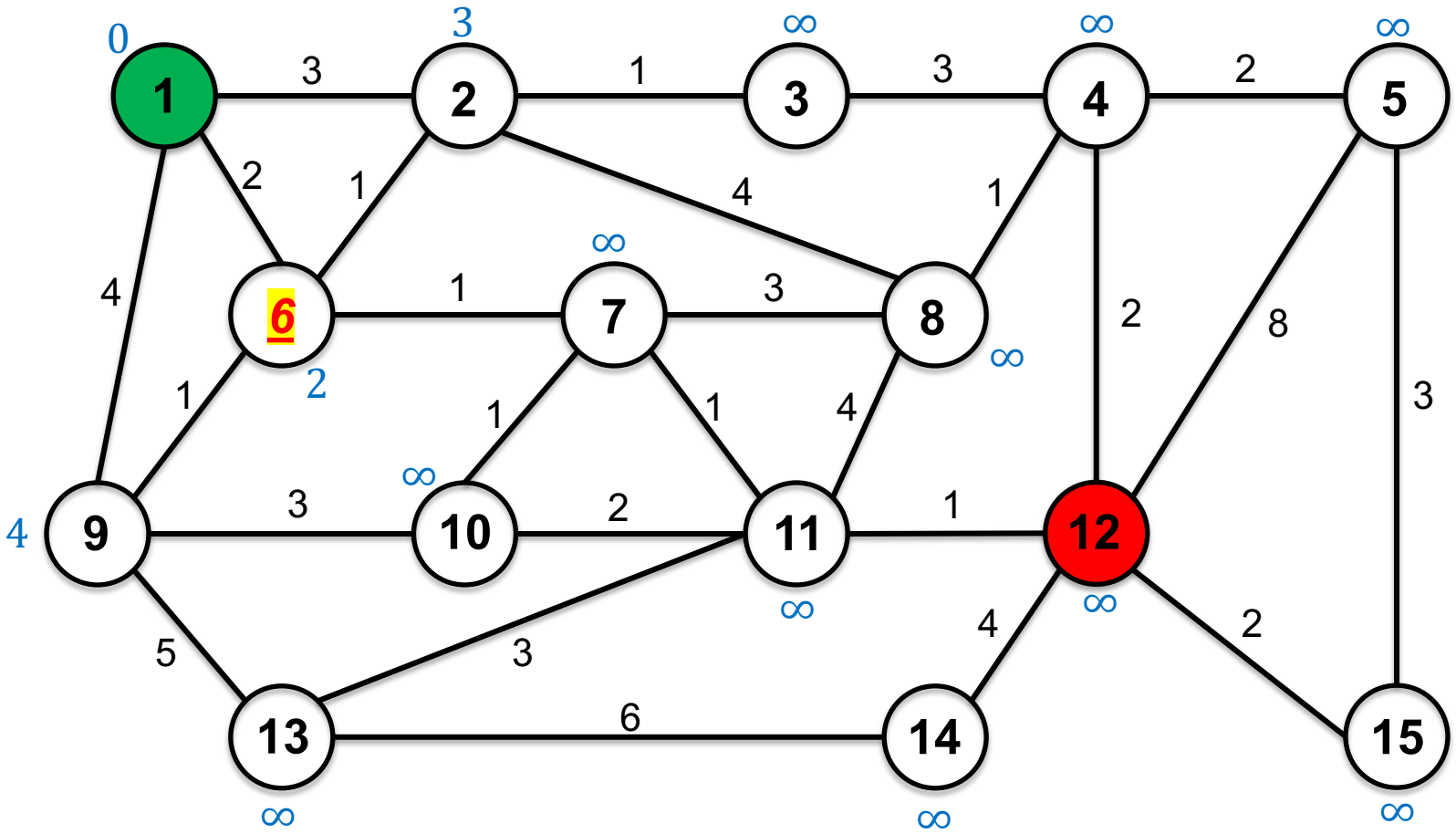
Update distance array

prev[2]=1, prev[6]=1, prev[9] = 1



Example (Undirected Graph)

C = 6, U = {2,3,4,5,7,8,9,10,11,12,13,14,15}

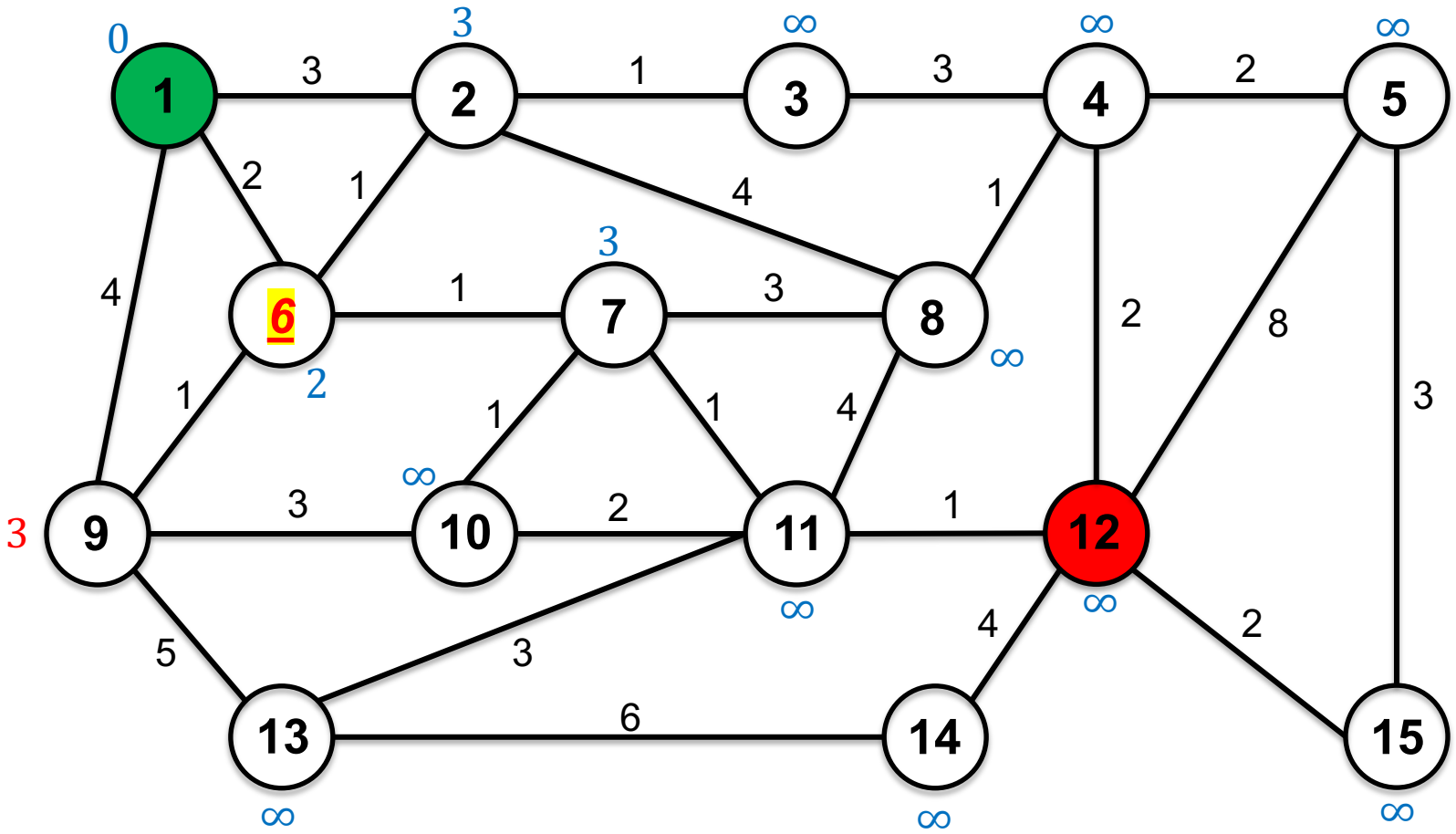


Example (Undirected Graph)

C = 6, U = {2,3,4,5,7,8,9,10,11,12,13,14,15}

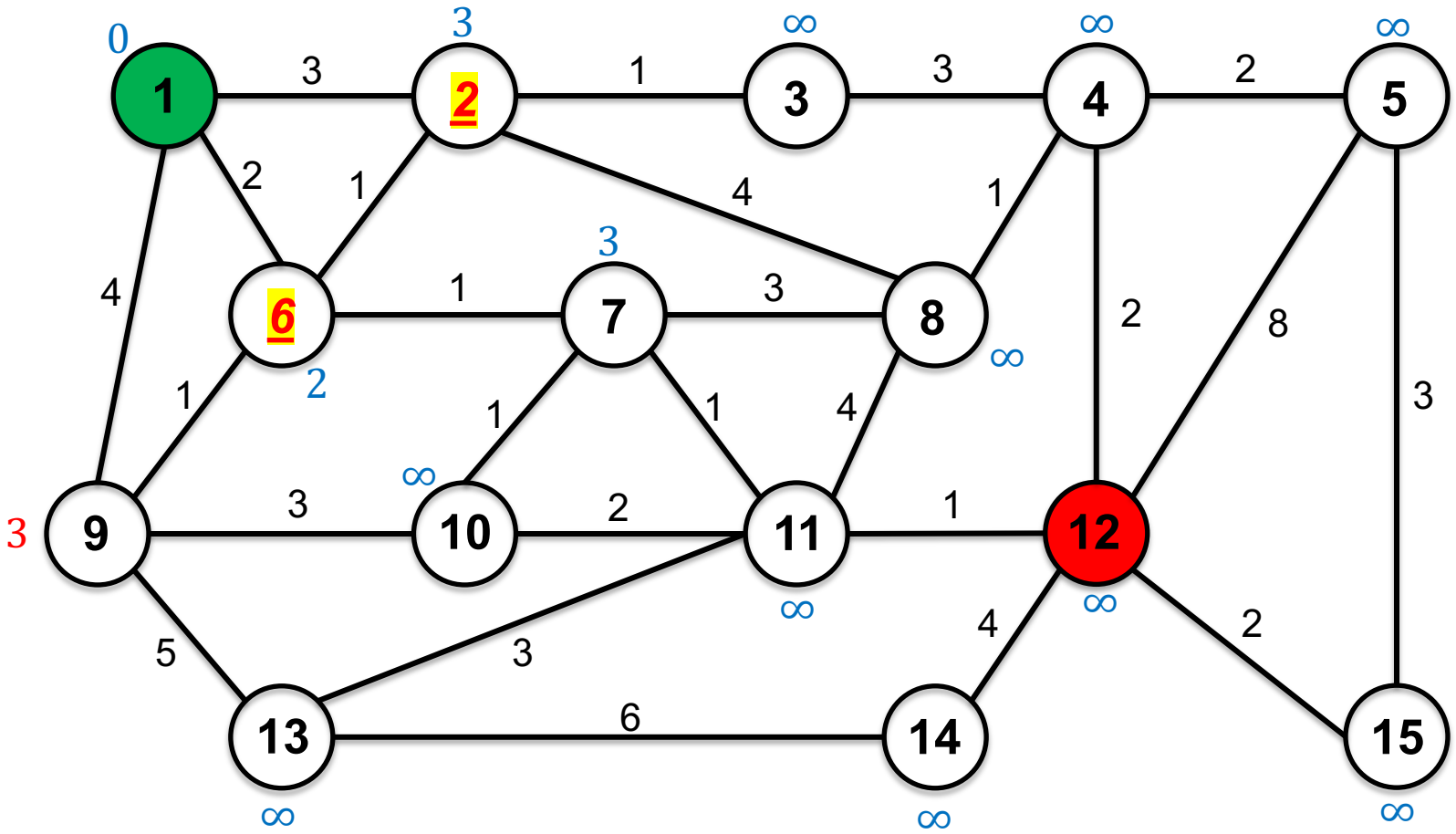
Update distance

prev[2] = 1, prev[6] = 1, prev[9] = 6



Example (Undirected Graph)

C = 2, U = {3,4,5,7,8,9,10,11,12,13,14,15}

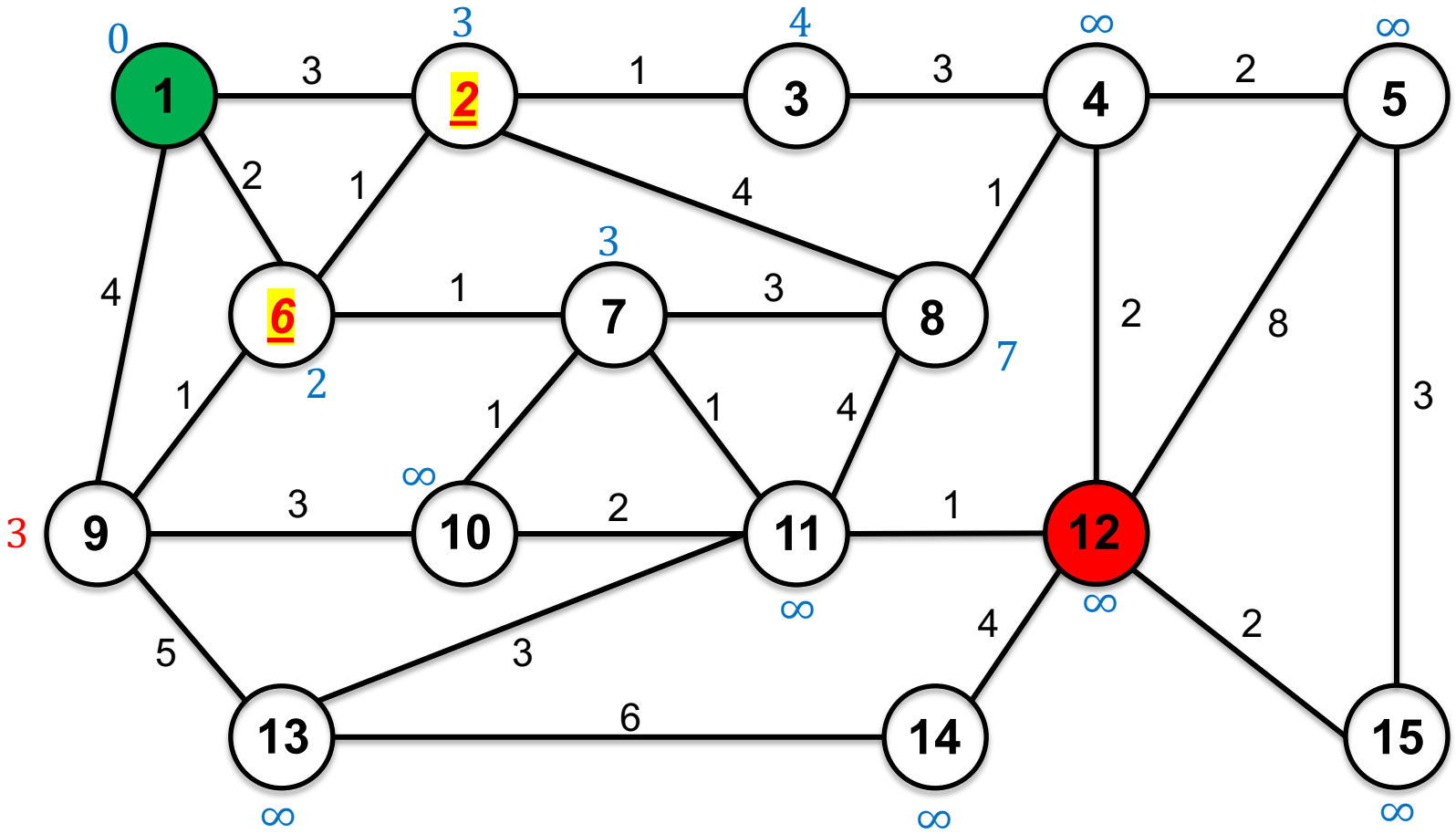


Example (Undirected Graph)

C = 2, U = {3,4,5,7,8,9,10,11,12,13,14,15}

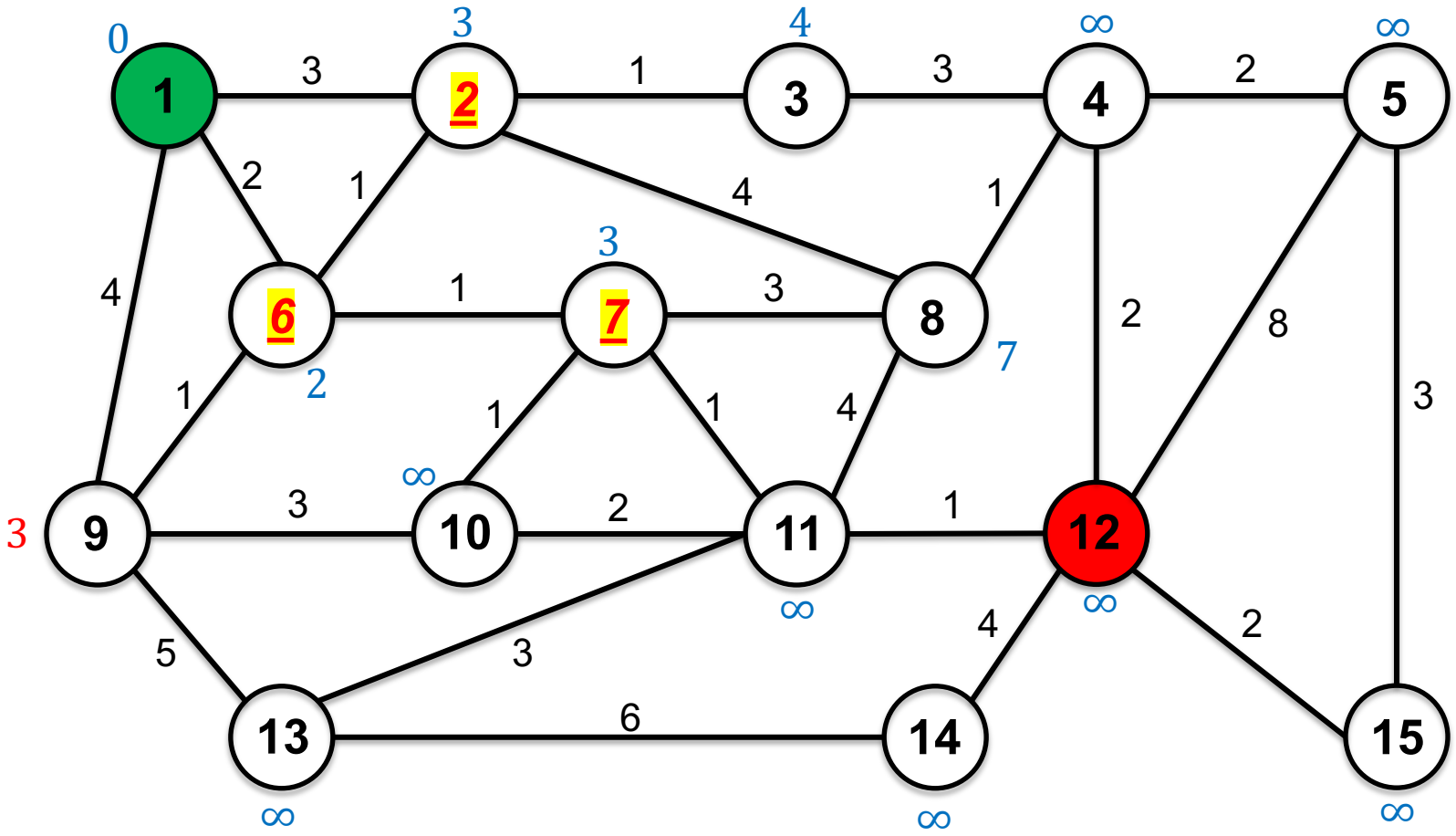
Update distance

prev[3] = 2, prev[7] = 6, prev[8] = 2



Example (Undirected Graph)

C = 7, U = {3,4,5,8,9,10,11,12,13,14,15}

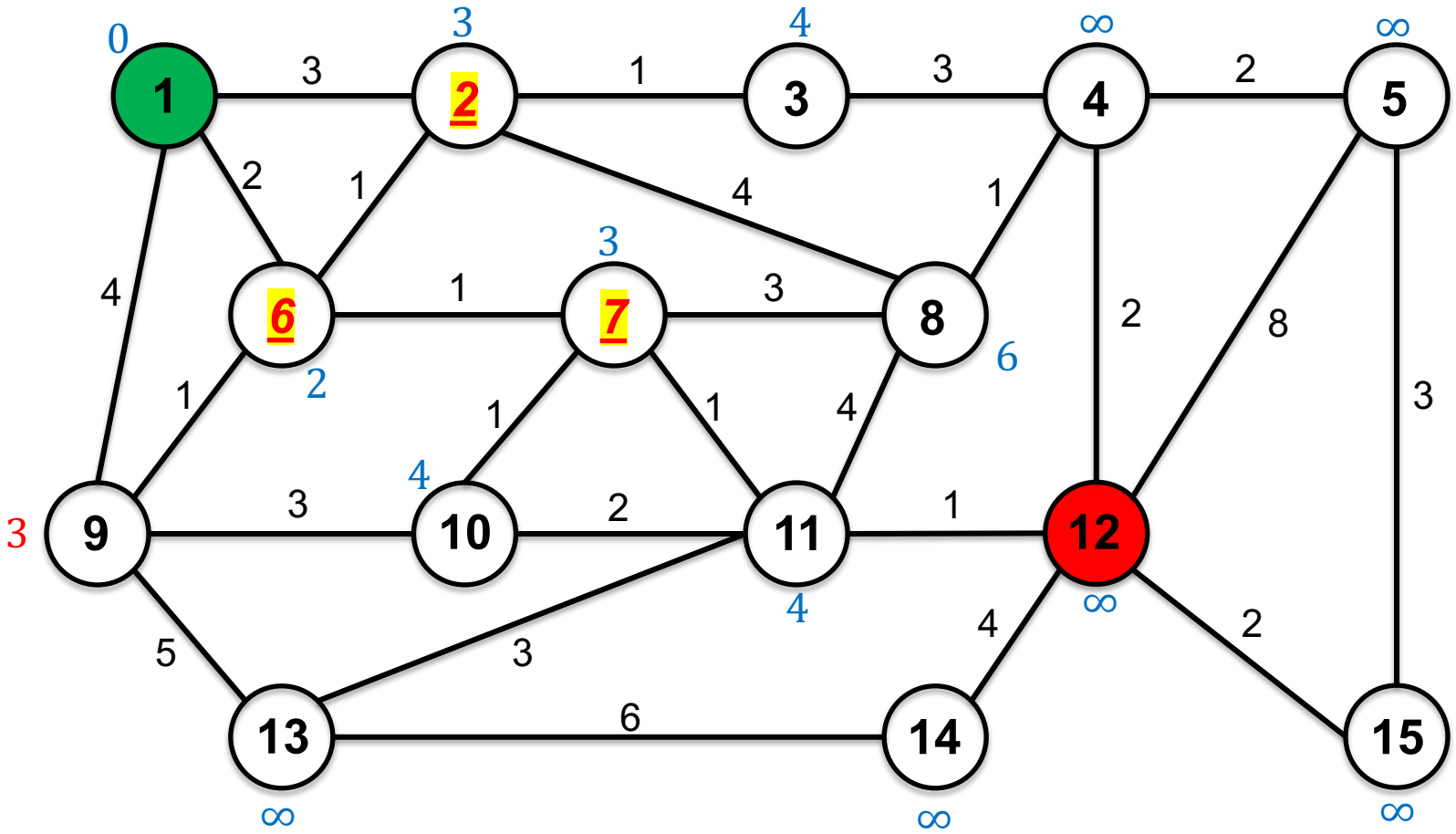


Example (Undirected Graph)

C = 7, U = {3,4,5,8,9,10,11,12,13,14,15}

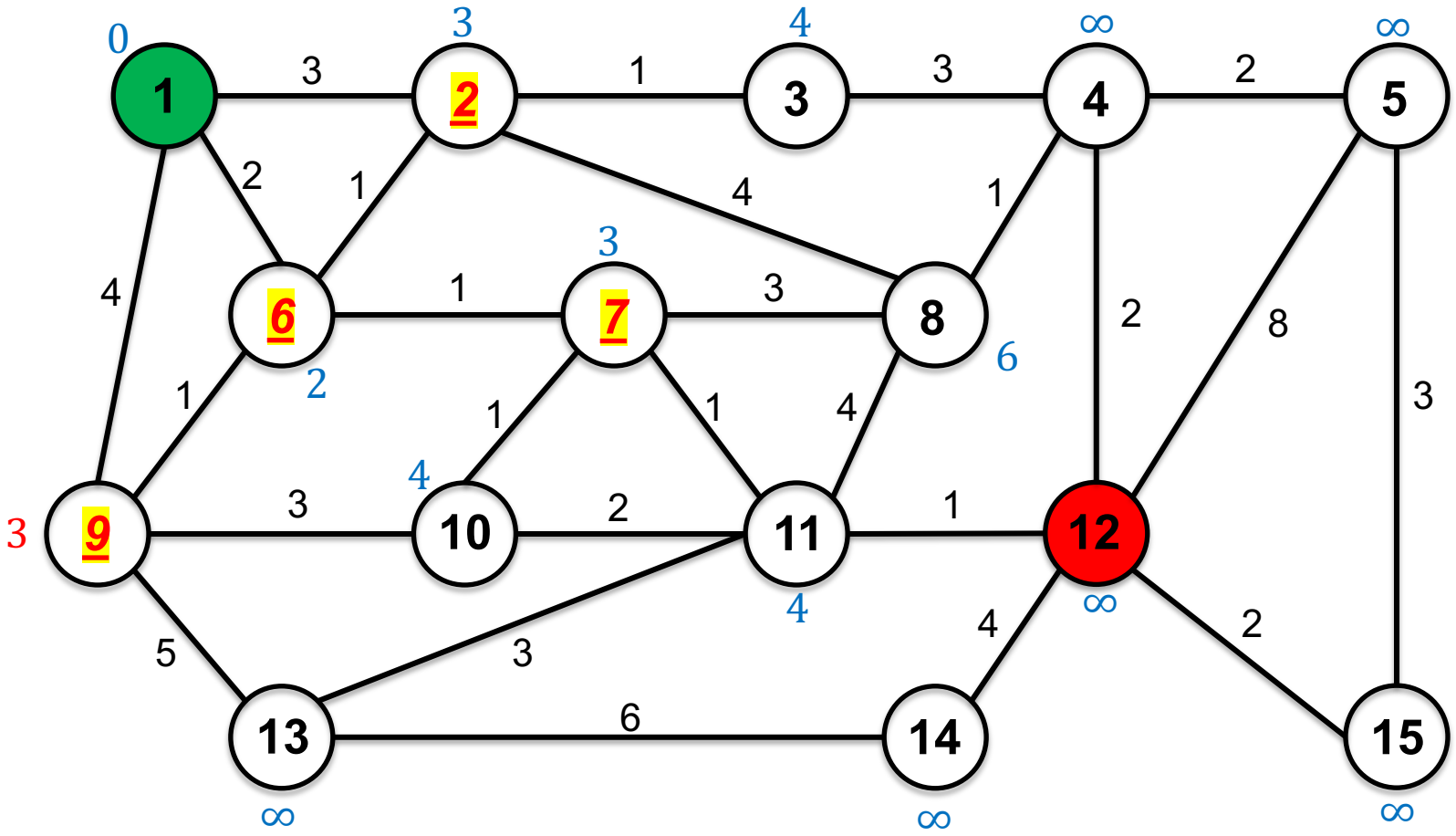
Update distance

prev[7] = 6, prev[8] = 7, prev[10] = 7, prev[11] = 7



Example (Undirected Graph)

C = 9, U = {3,4,5,8,10,11,12,13,14,15}

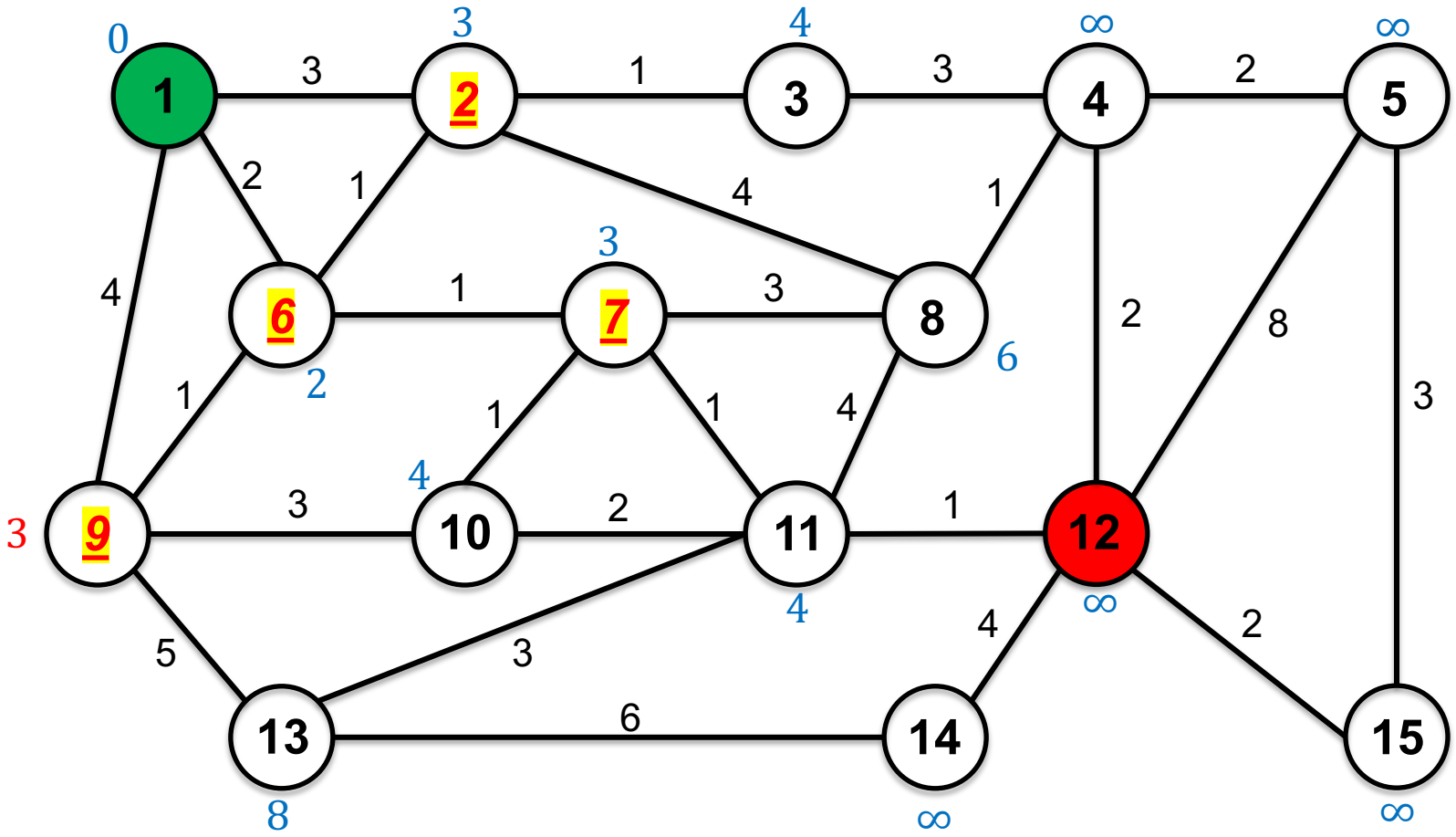


Example (Undirected Graph)

C = 9, U = {3,4,5,8,10,11,12,13,14,15}

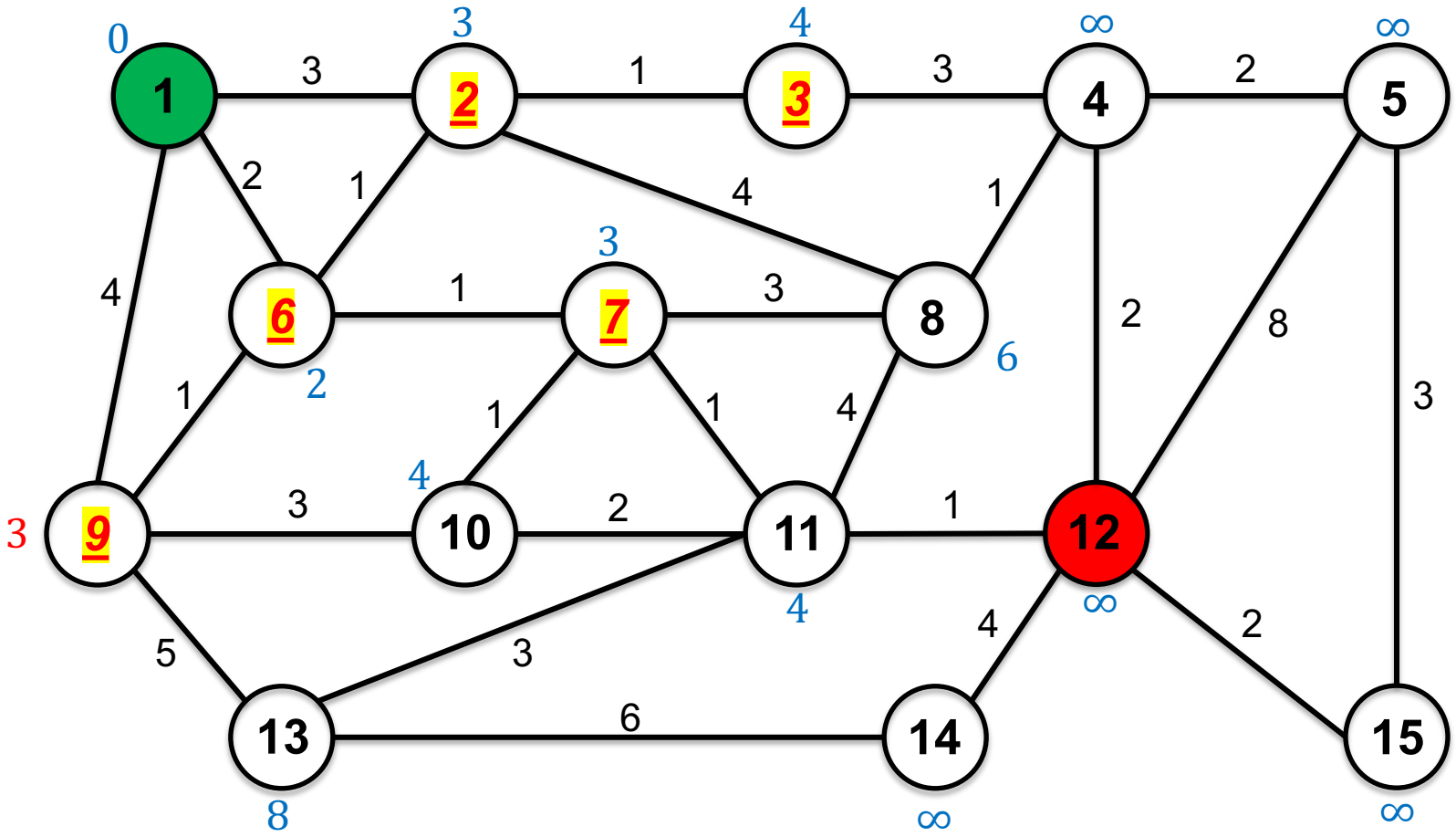
Update distance

prev[13] = 9



Example (Undirected Graph)

C = 3, U = {4,5,8,10,11,12,13,14,15}

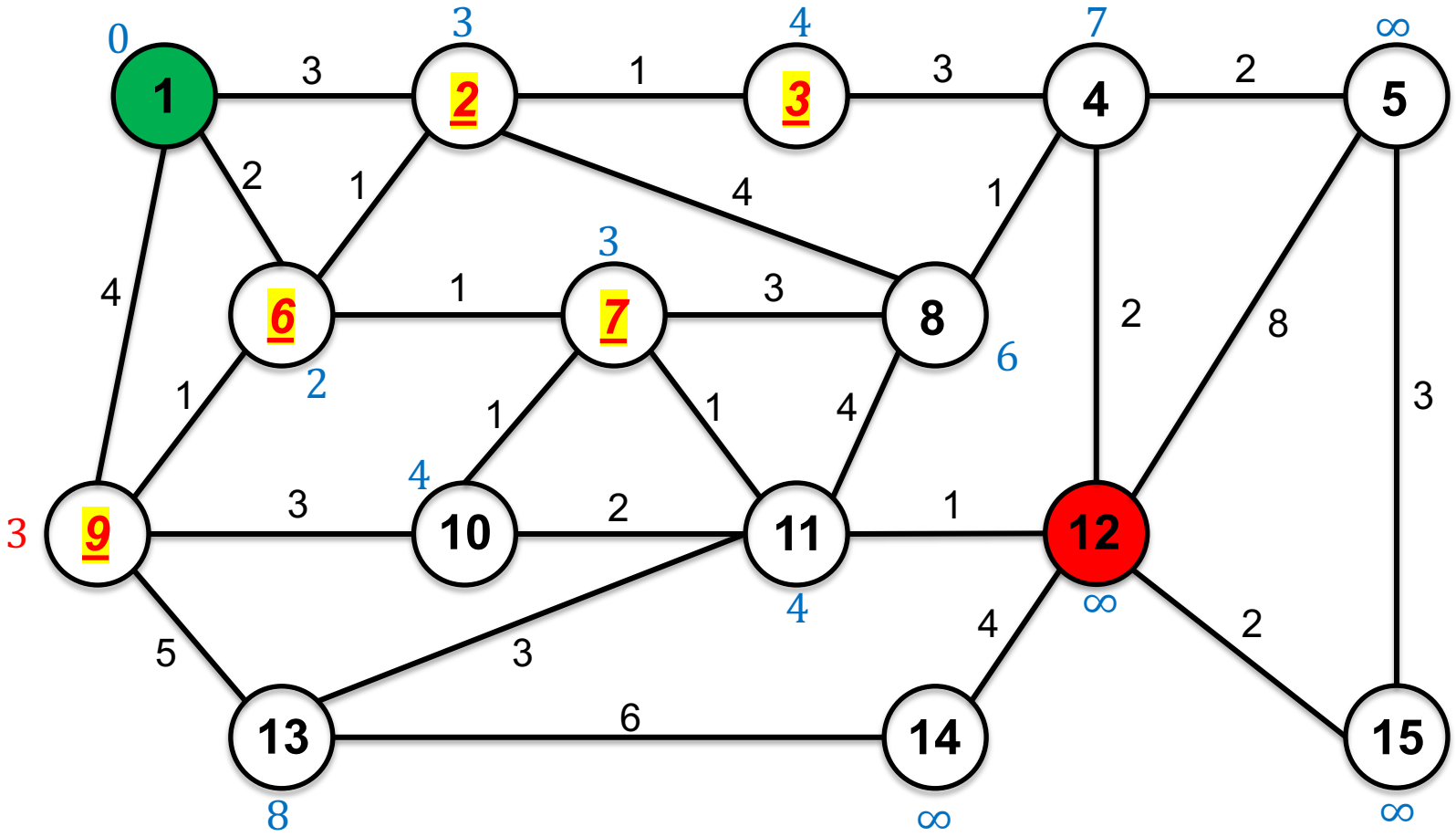


Example (Undirected Graph)

C = 3, U = {4,5,8,10,11,12,13,14,15}

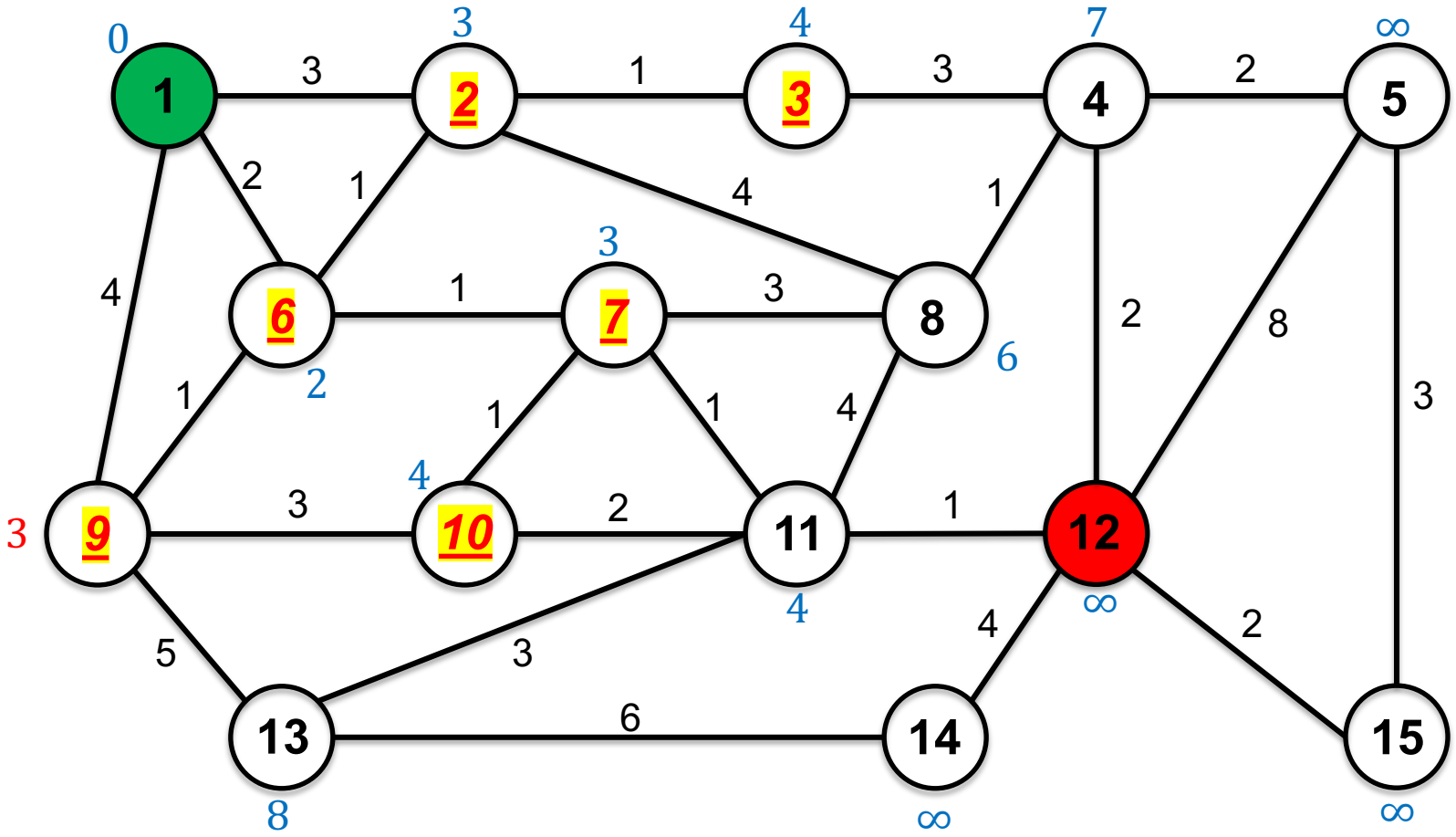
Update distance

prev[4] = 3



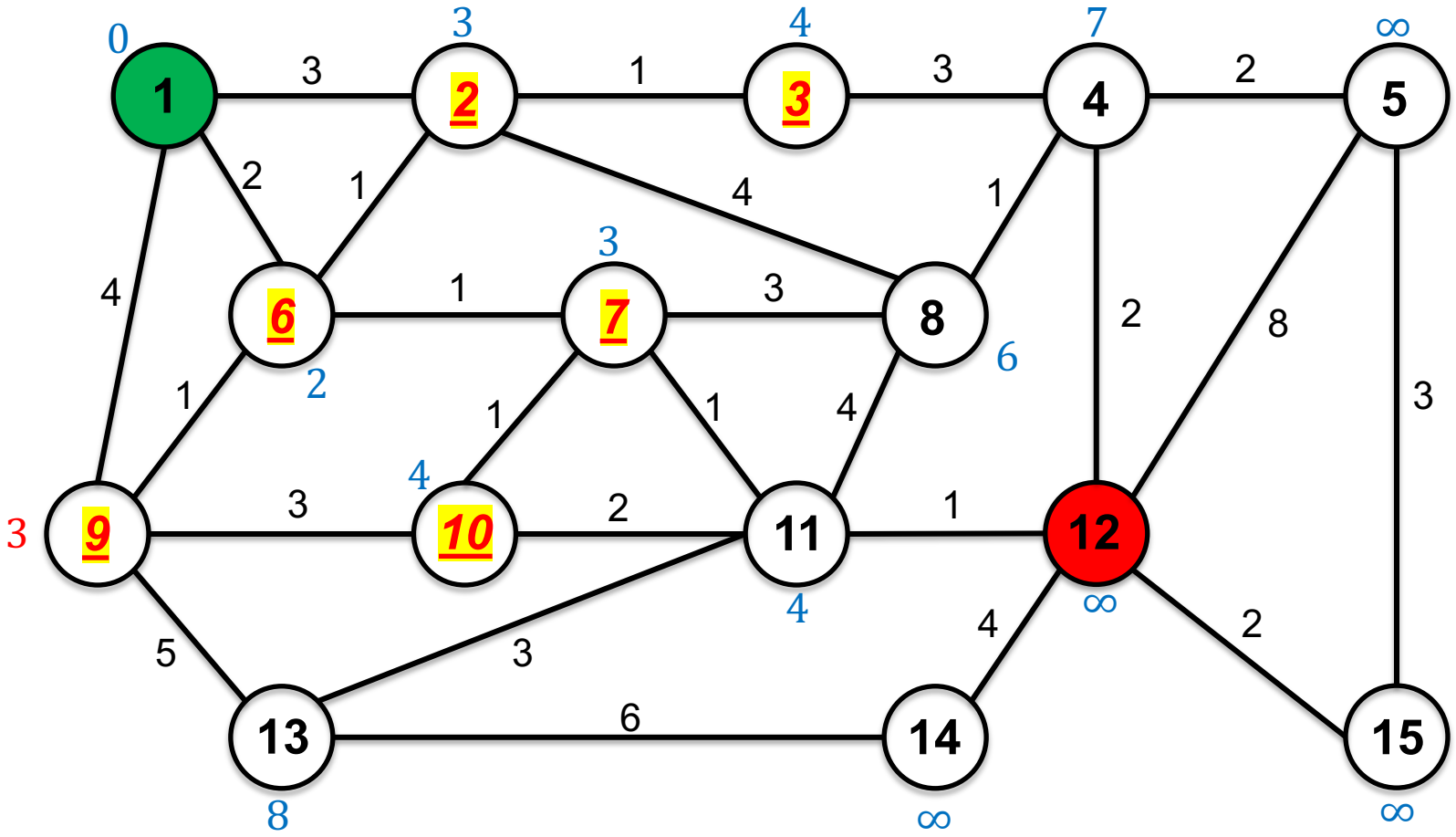
Example (Undirected Graph)

C = 10, U = {4,5,8,11,12,13,14,15}



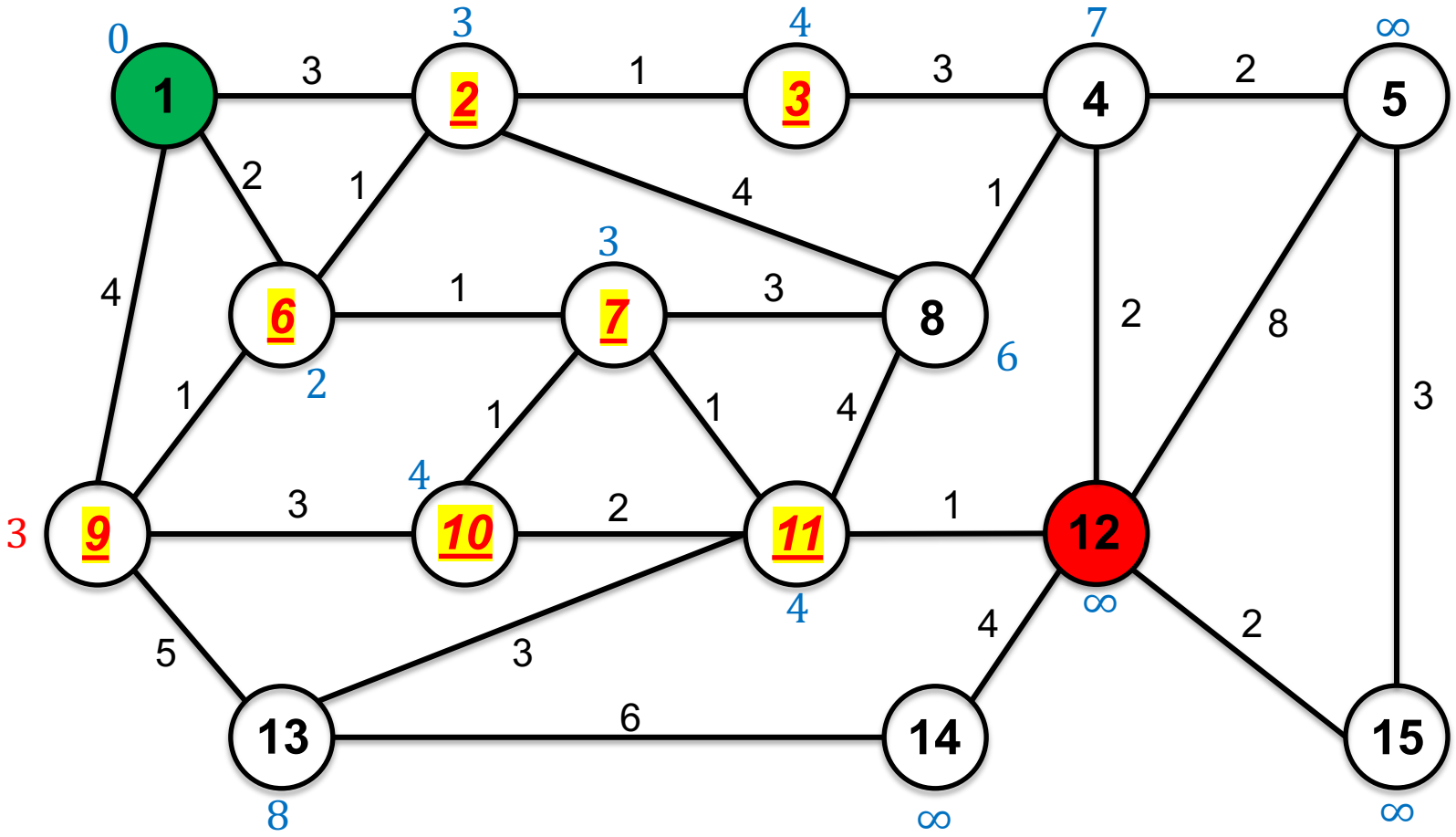
Example (Undirected Graph)

C = 10, U = {4,5,8,11,12,13,14,15}
Update distance



Example (Undirected Graph)

C = 11, U = {4,5,8,12,13,14,15}

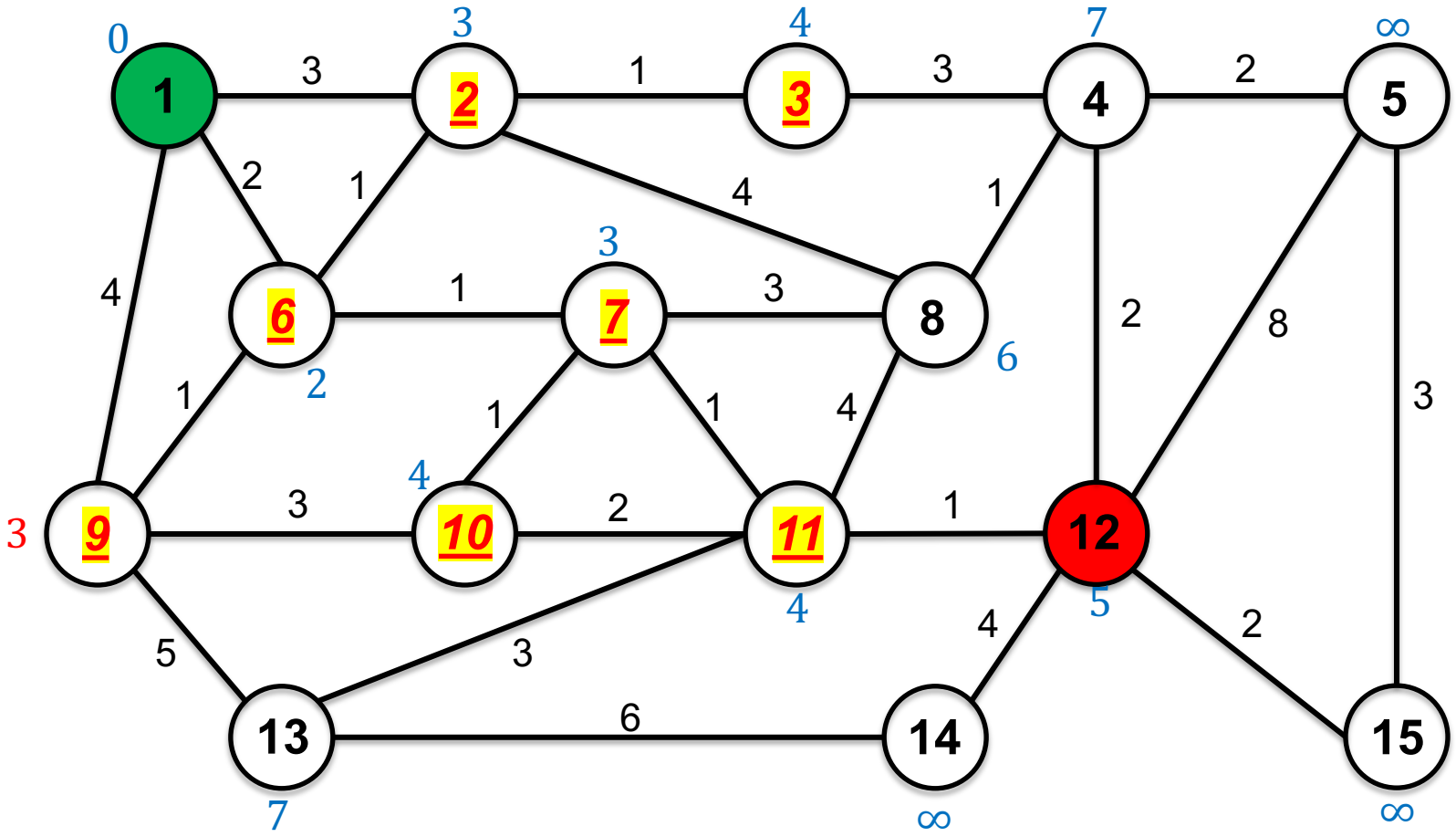


Example (Undirected Graph)

C = 11, U = {4,5,8,12,13,14,15}

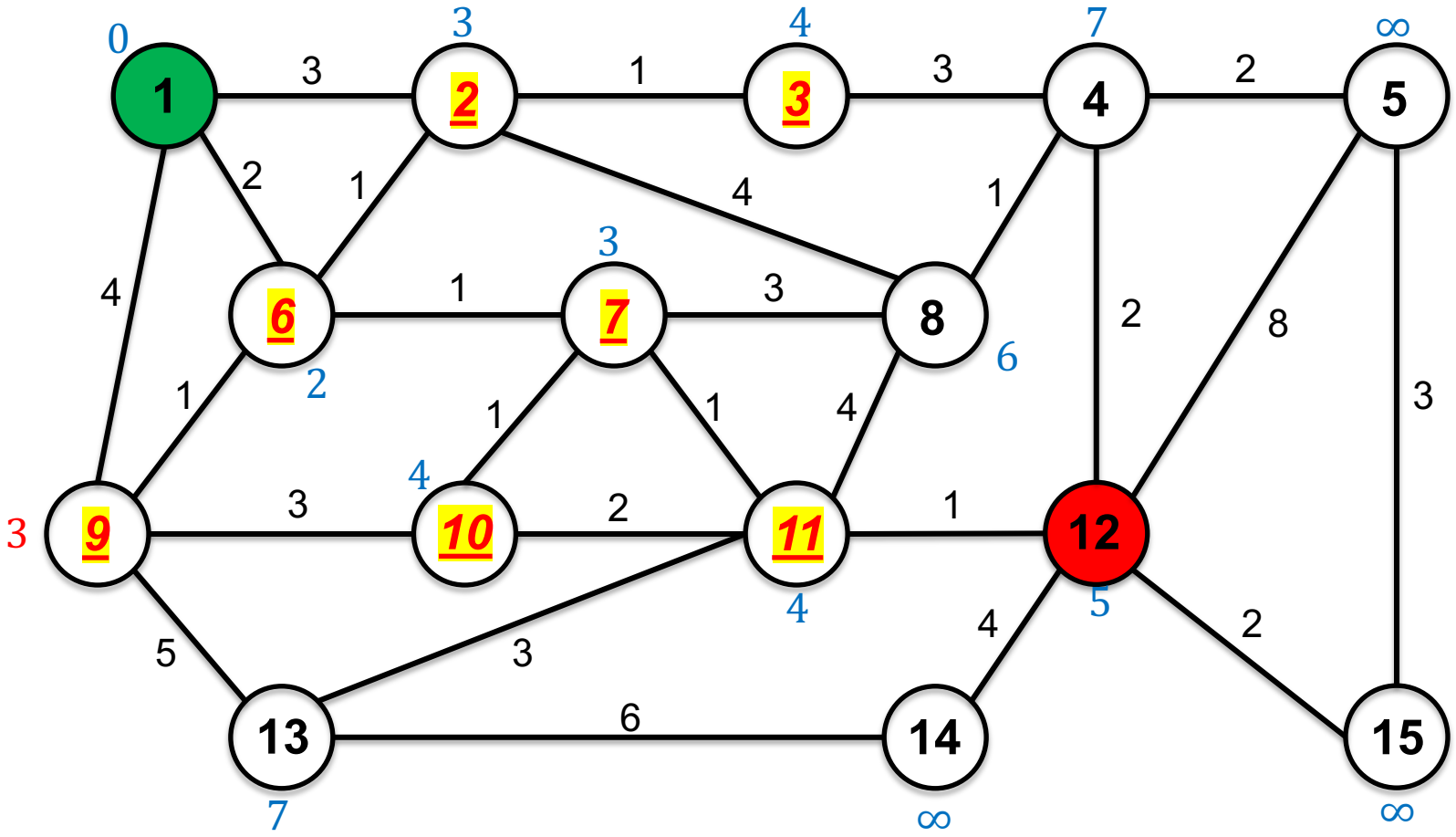
Update distance

prev[12] = 11, prev[13] = 11



Example (Undirected Graph)

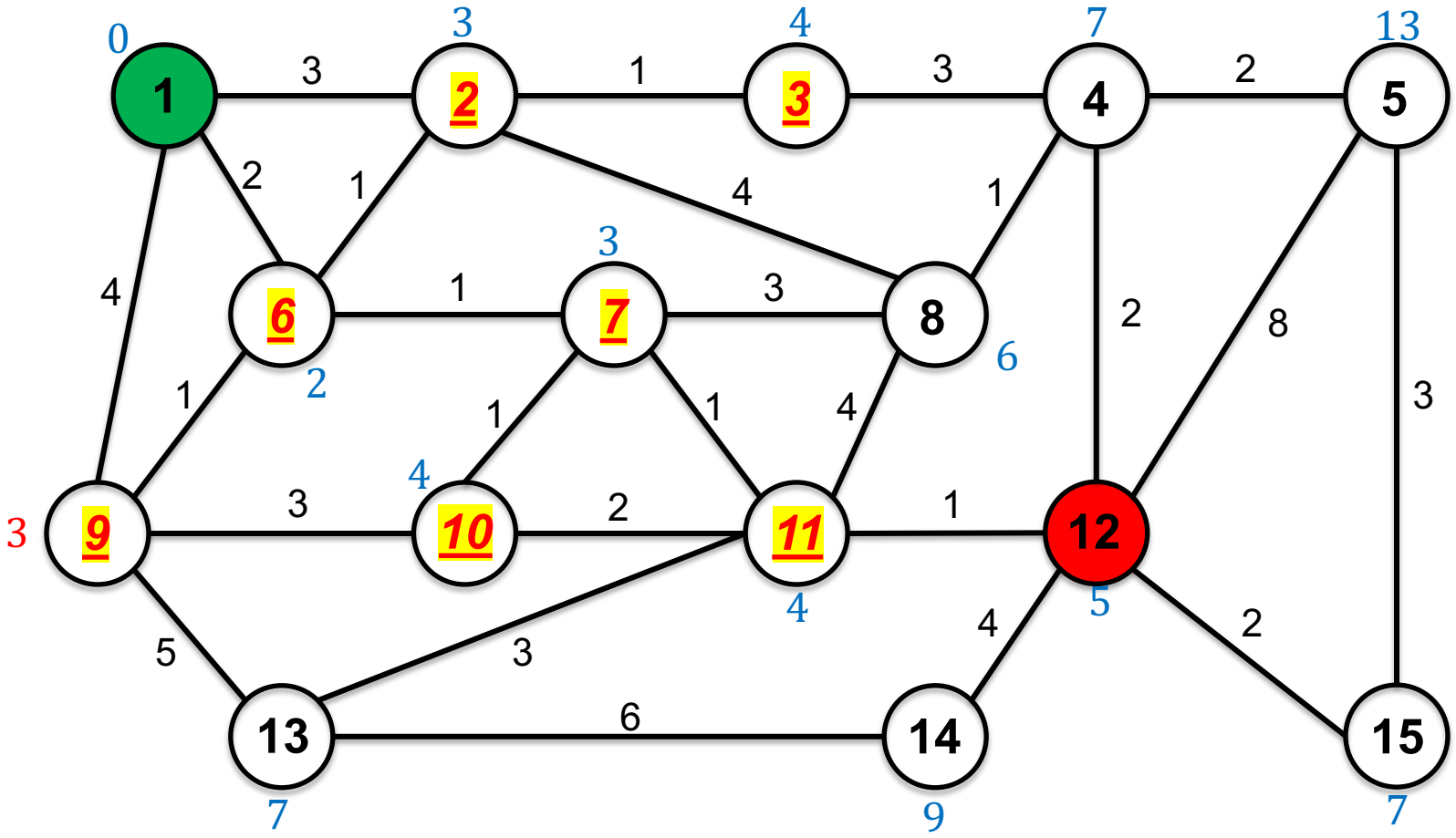
C = 12, U = {4,5,8,13,14,15}



Example (Undirected Graph)

C = 12, U = {4,5,8,13,14,15}
Update distance

prev[14] = 12, prev[15] = 12



Example (Undirected Graph)

C = 12, U = {4,5,8,13,14,15}

Stop (because 12 is not in U) and backward search through *prev* to find the shortest path