Week 5 Discussion Section
Genome 540

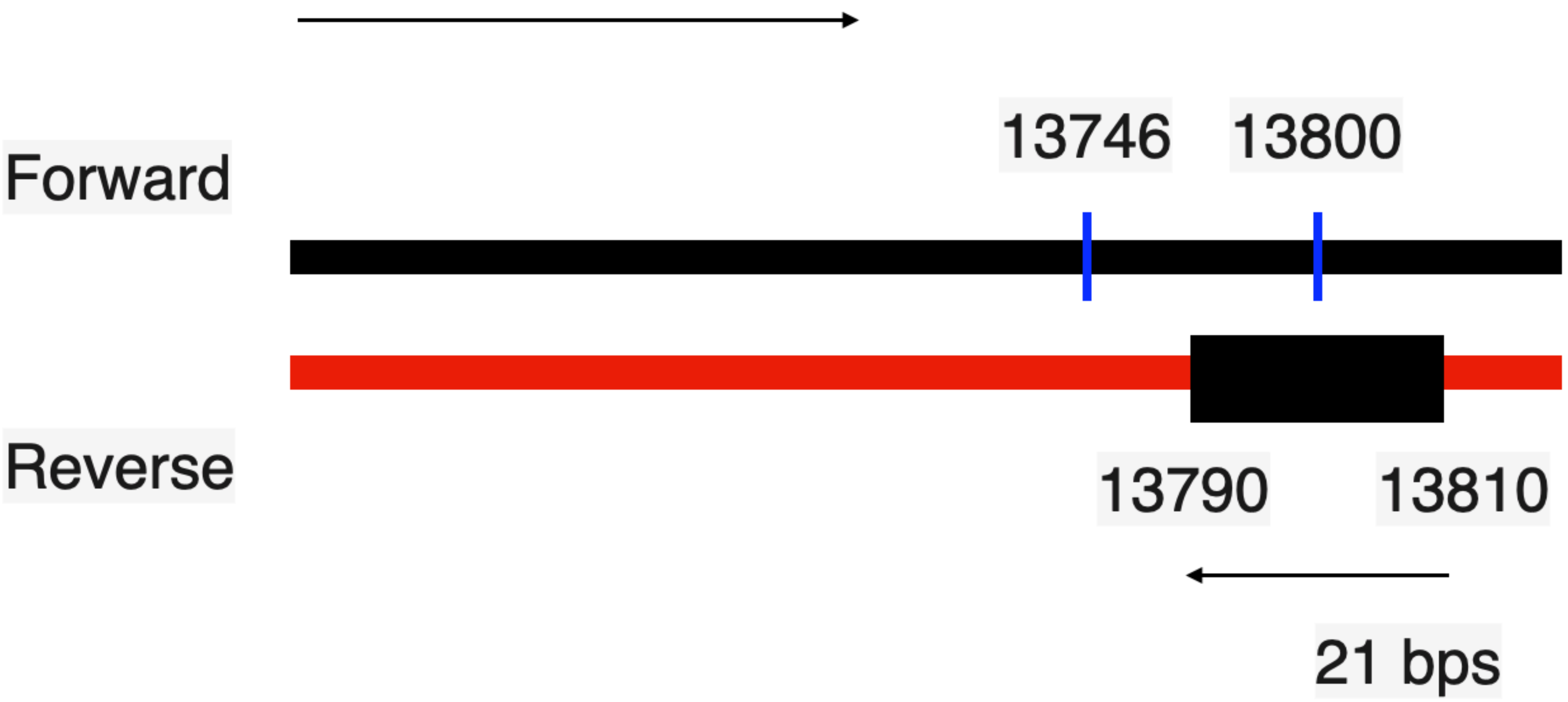
Chengxiang Qiu

HW3 feedbacks

Please at least go through the discussion slides before you start doing the homework.

complement(13746..13800)

Read the sequence from 1 to N

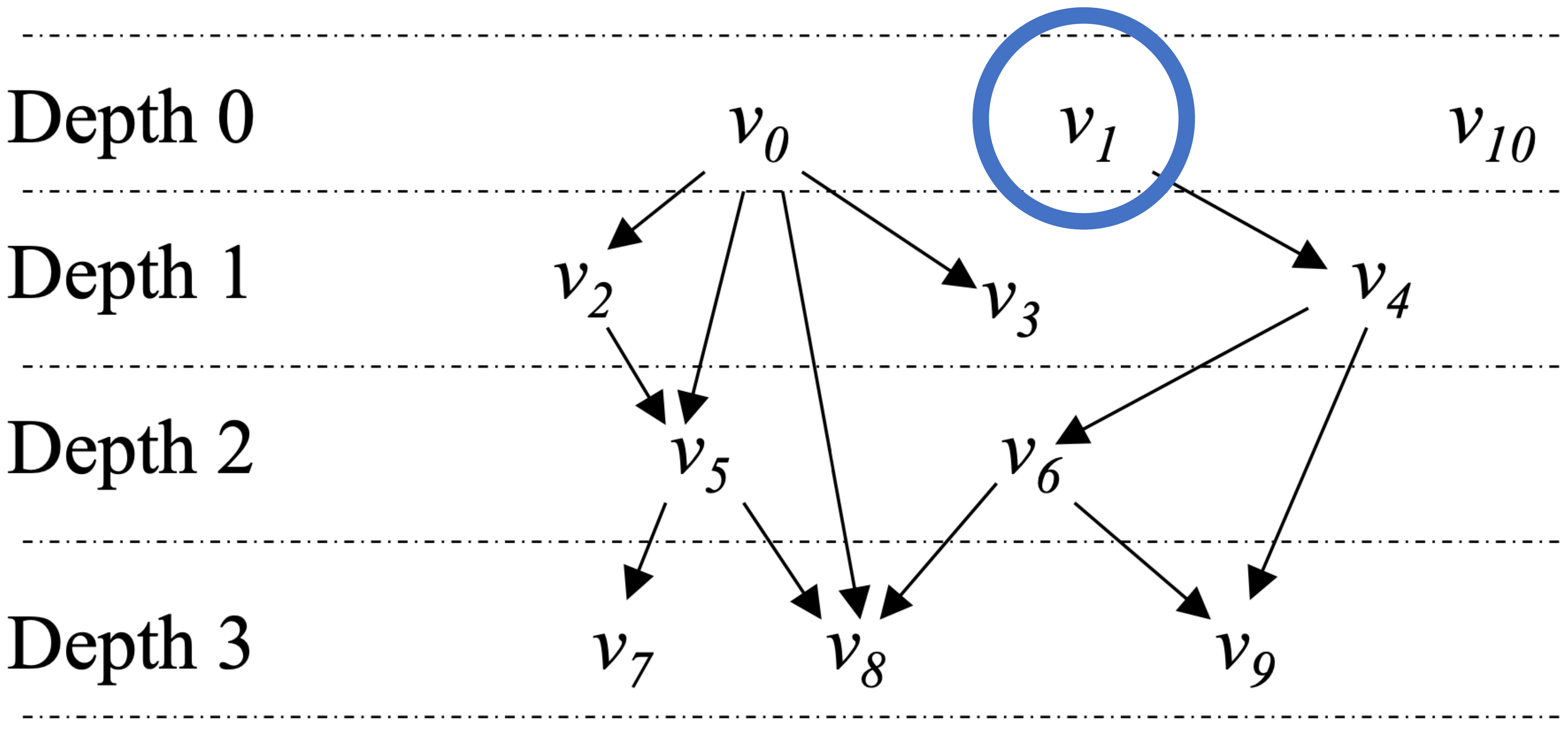


Postion list

111275	1	12.2057
120297	0	12.2057
122310	1	10.0736
122613	1	11.7212
124361	0	12.2057
128239	1	12.2057

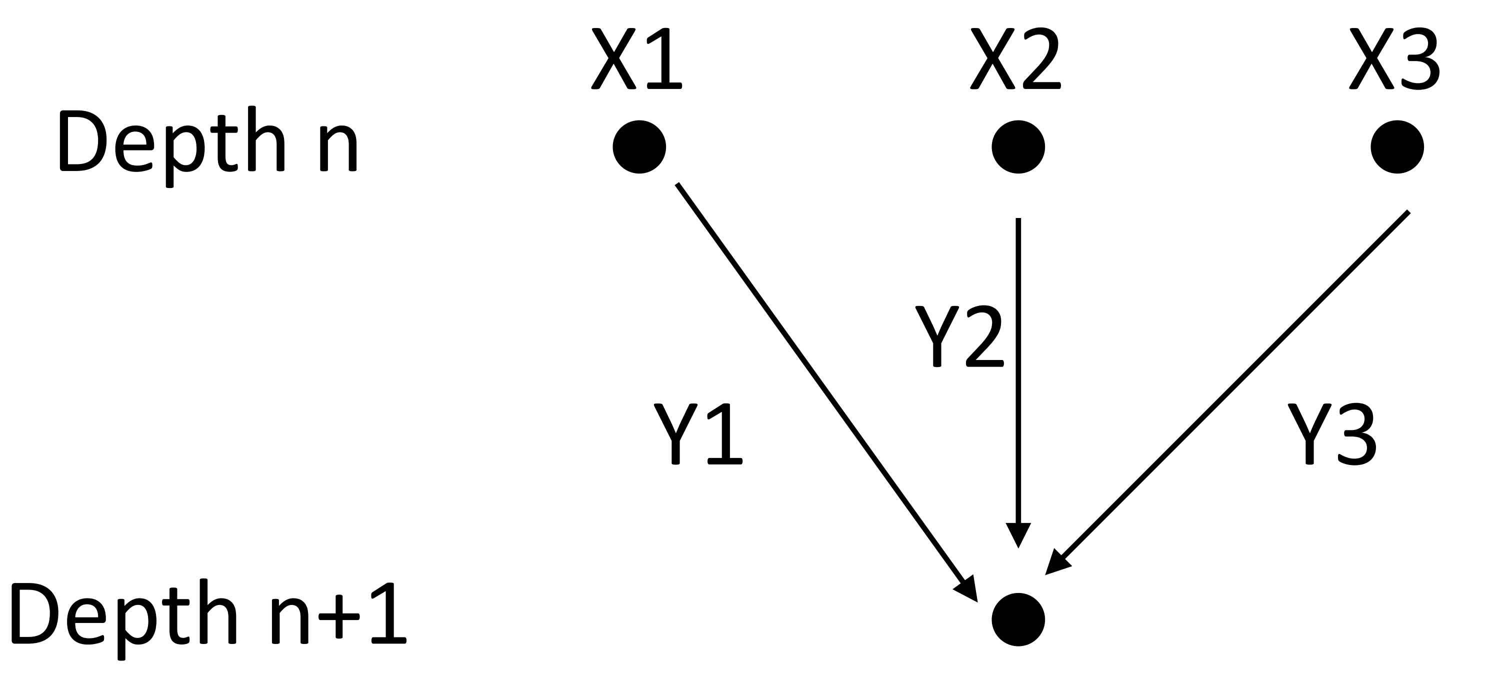
HW4 - Find a highest-weight path in a weighted directed acyclic graph

1) Arranging vertices by depth



2) dynamic programing

$$w(v) = \max(0, \max_{u \in parents(v)} (w(u) + w((u,v))))$$

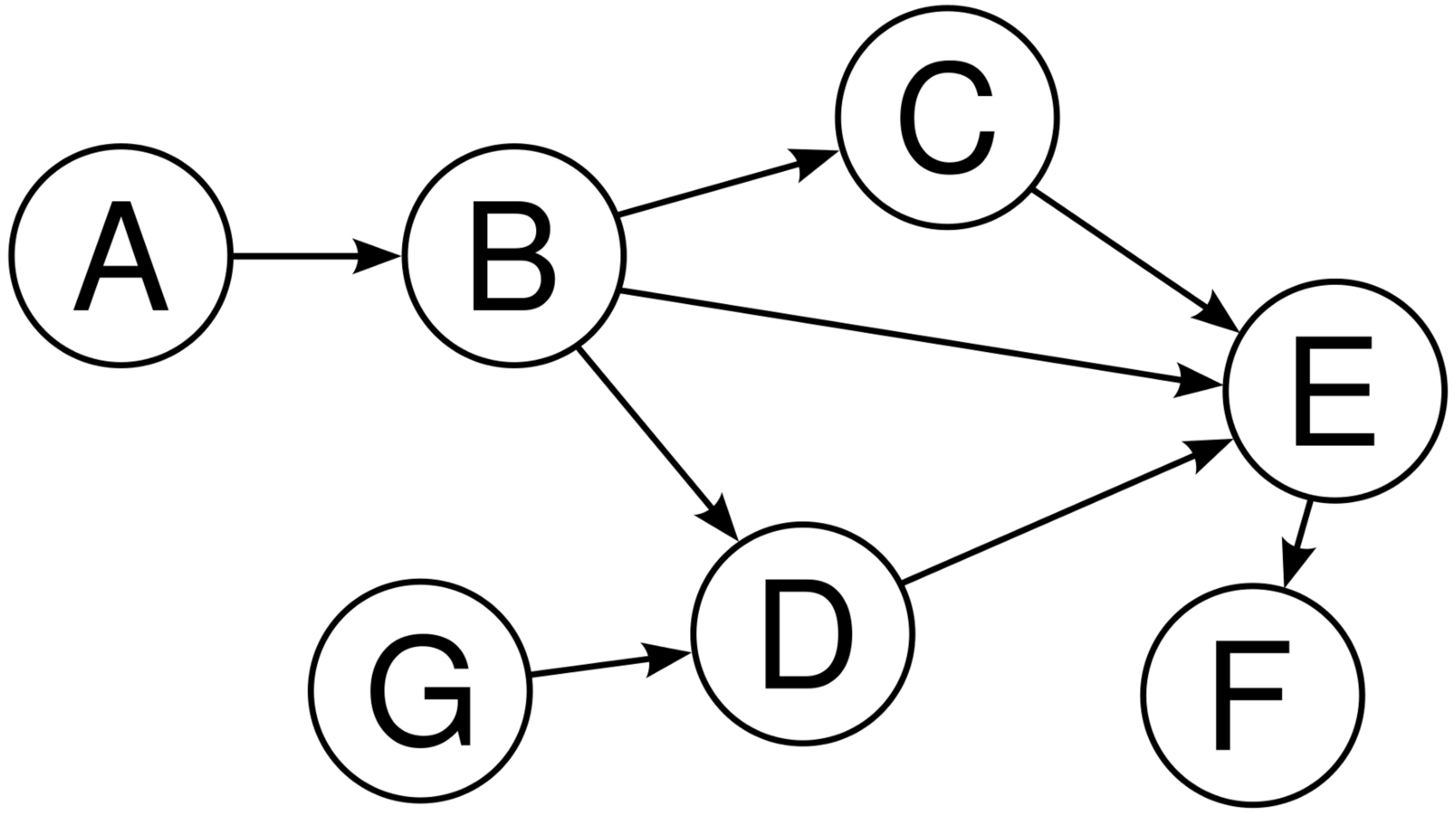


max(0, max(x1+y1, x2+y2, x3+y3))

3) "Constrained"

For example, requiring the path start at node v1

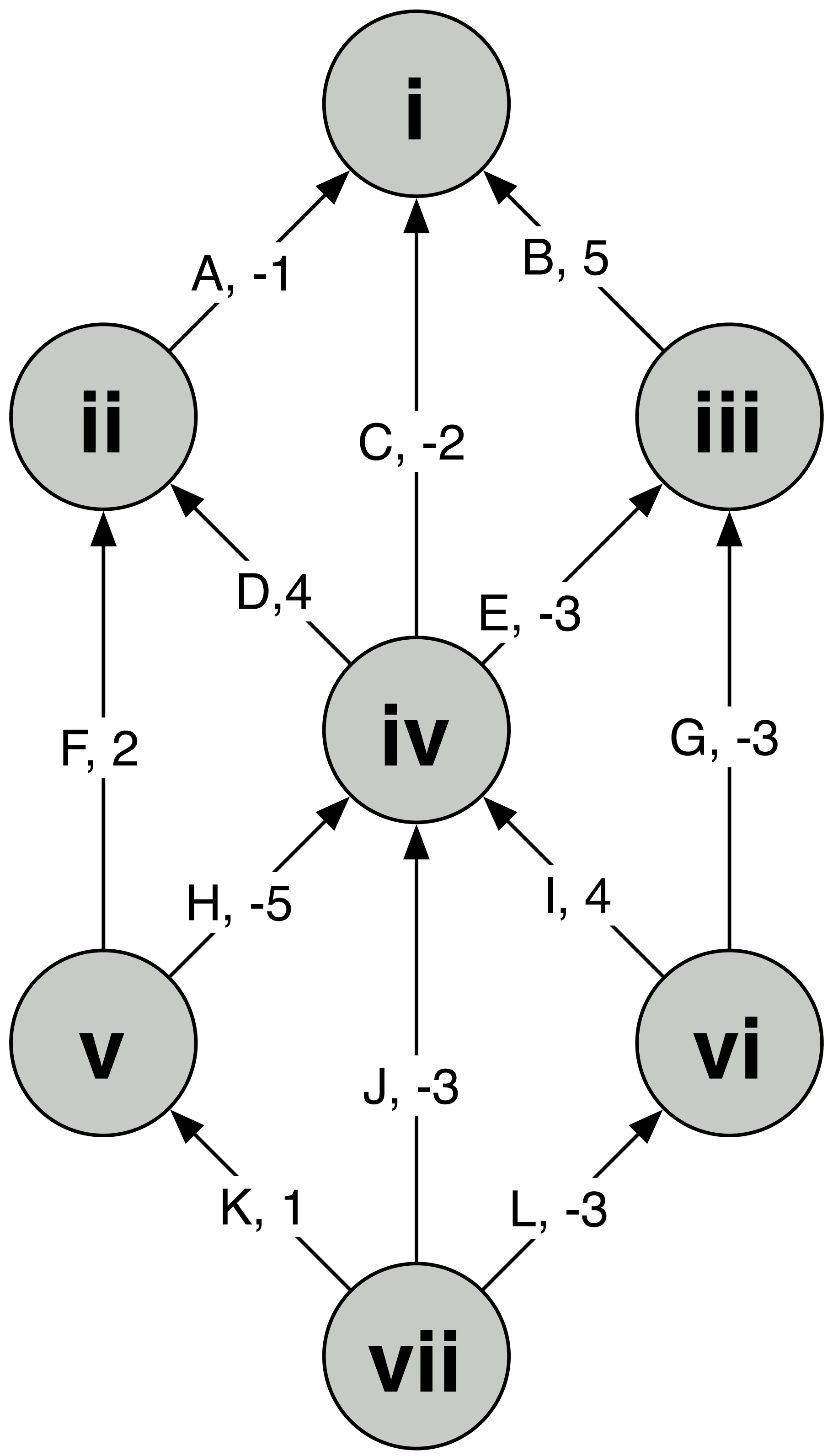
Other DAG Algorithms



- Every edge has the same weight
- The shortest path from A to any nodes (e.g. F)?

Other DAG Algorithms

Minimum-weight path on a DAG



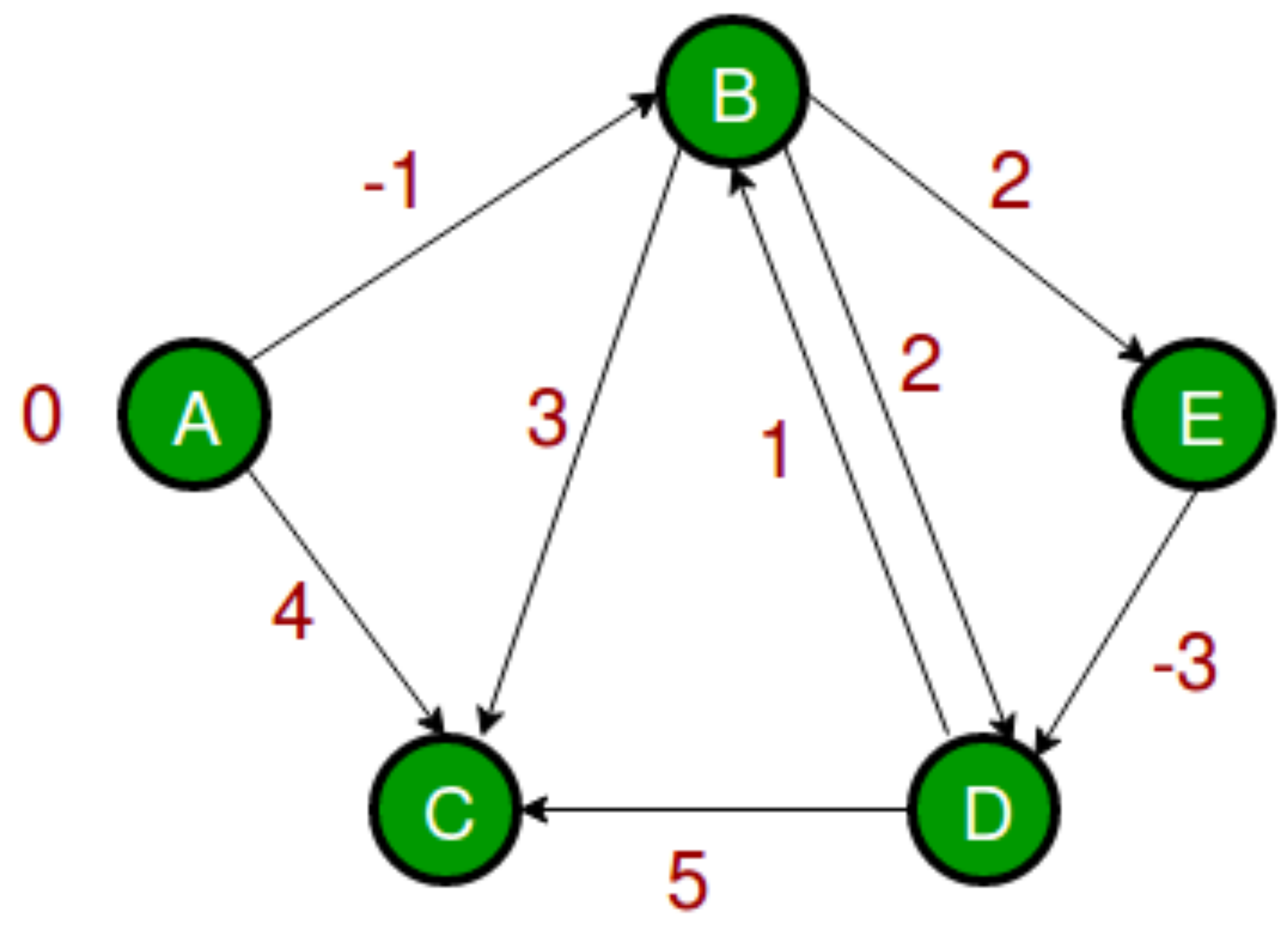
- Similar to the homework
  - Looking for the minimum instead of the maximum
  - If there are no negative weights, then the shortest path is technically weight 0 (single node)
  - Otherwise, update vertex weights in depth order as normal

$$w(v) = \min\left(0, \min_{u \in parents(v)} (w(u) + w((u,v)))\right)$$

What if graph has cycles?

- More difficult (can’t order nodes by depth)

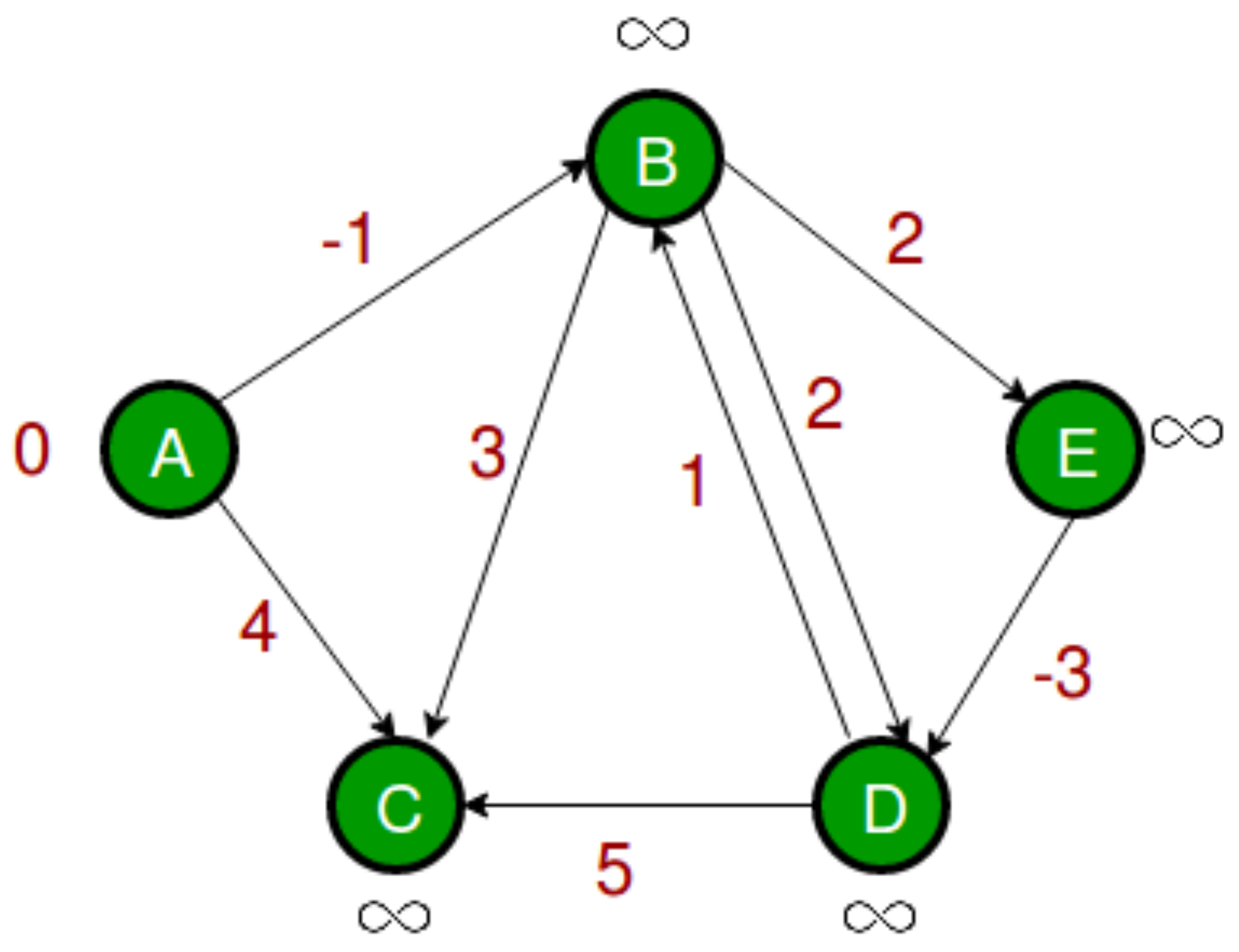
- From a given start vertex, the shortest path to each vertex

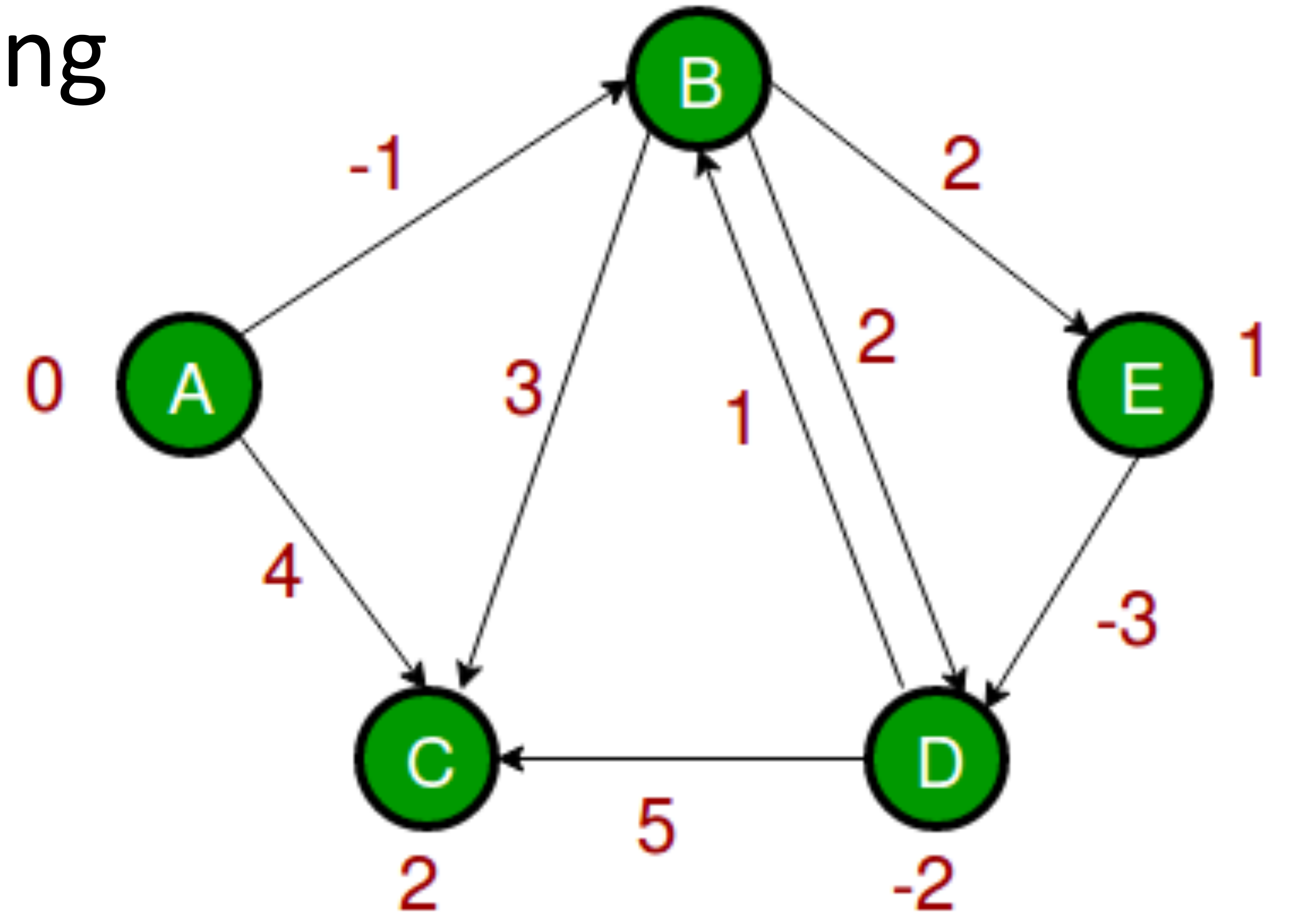


What if graph has cycles?

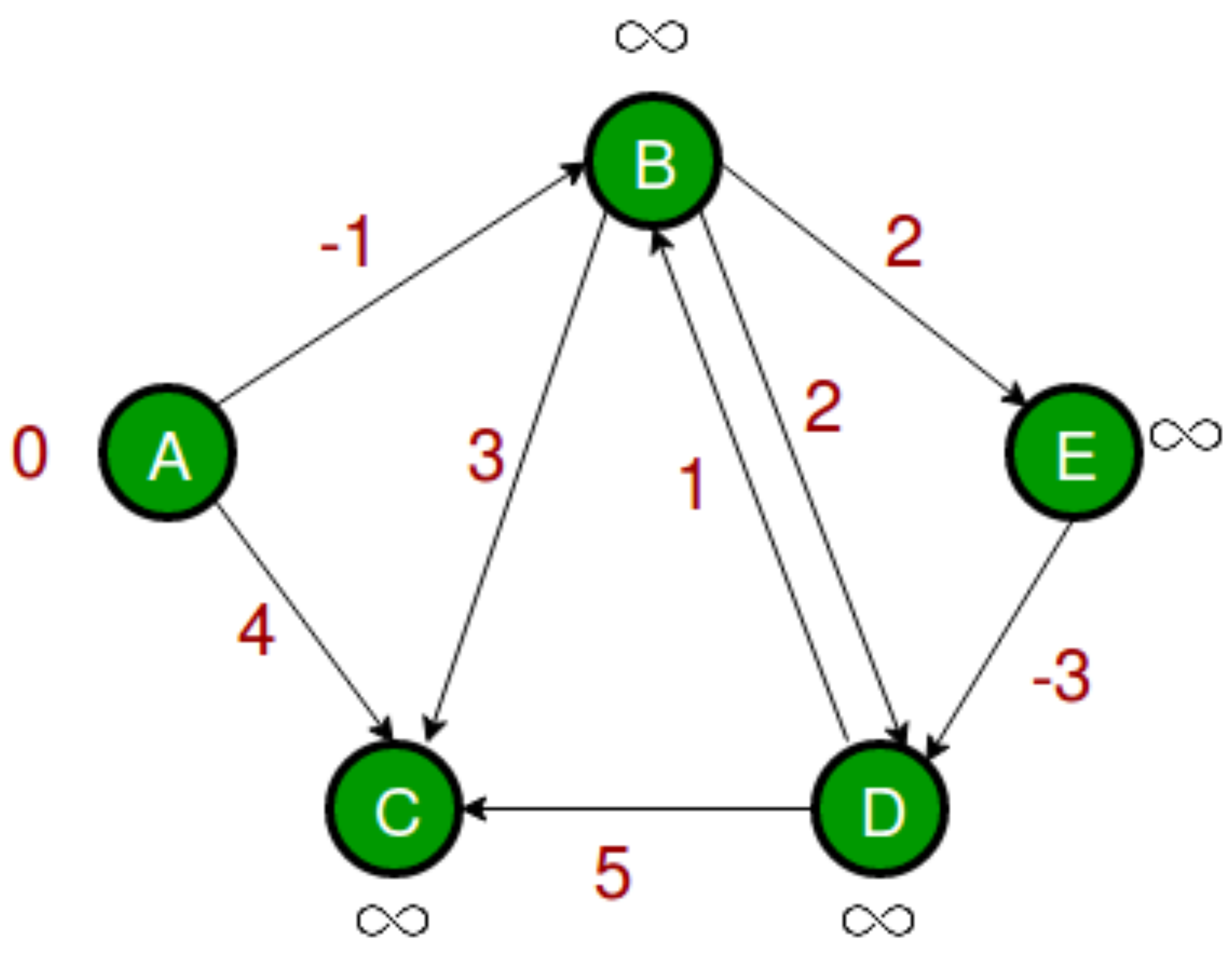
- More difficult (can’t order nodes by depth)

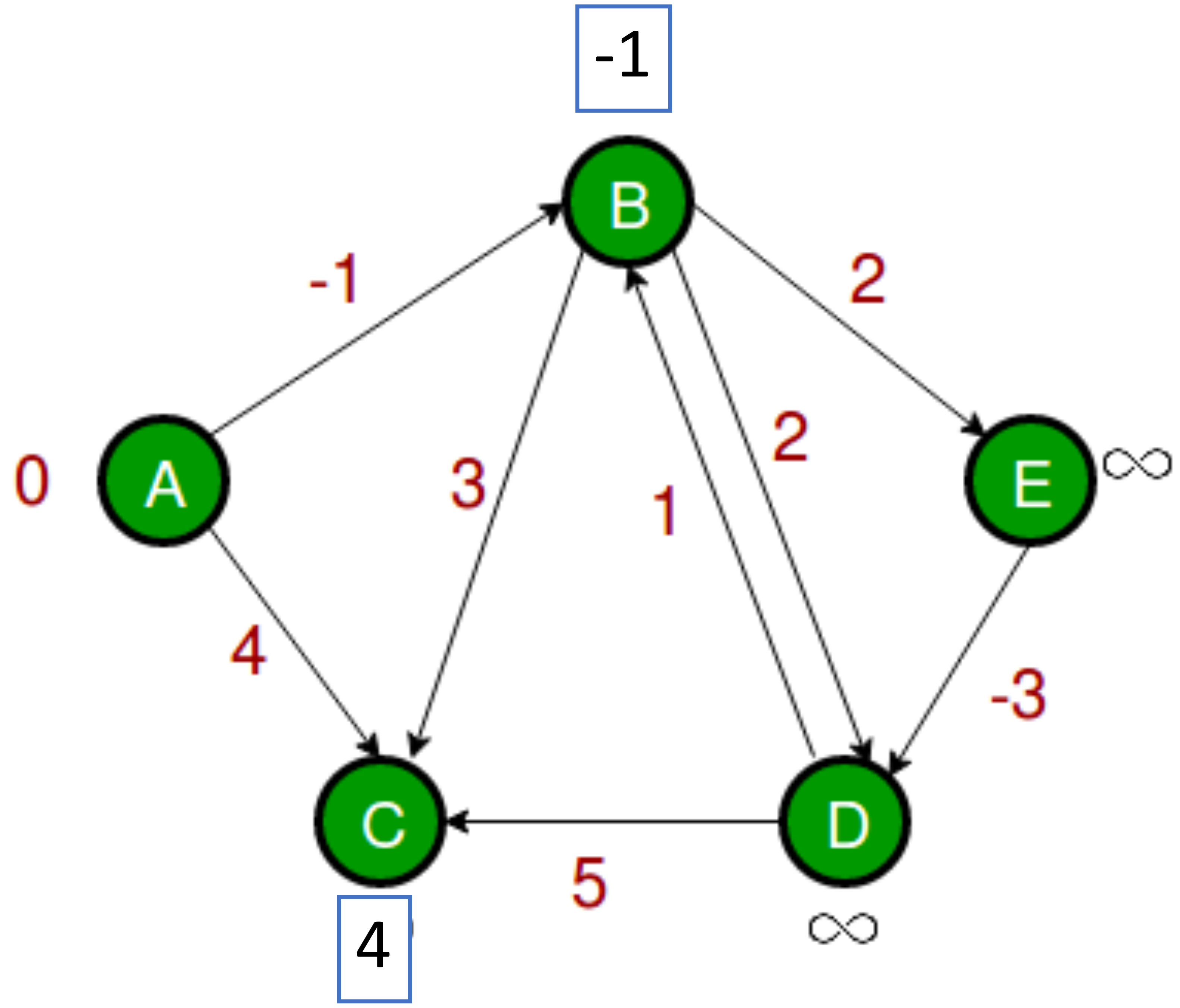
- Bellman-Ford algorithm (for given start vertex)
  - Choose source node and set distance to 0
  - Set distance to all other nodes to infinity
  - For **each** edge (u,v), if v’s distance can be reduced by taking that edge, update v’s distance
  - Cycle through all edges in this way |V|-1 times
  - (can also check for negative-weight cycle with one extra iteration)

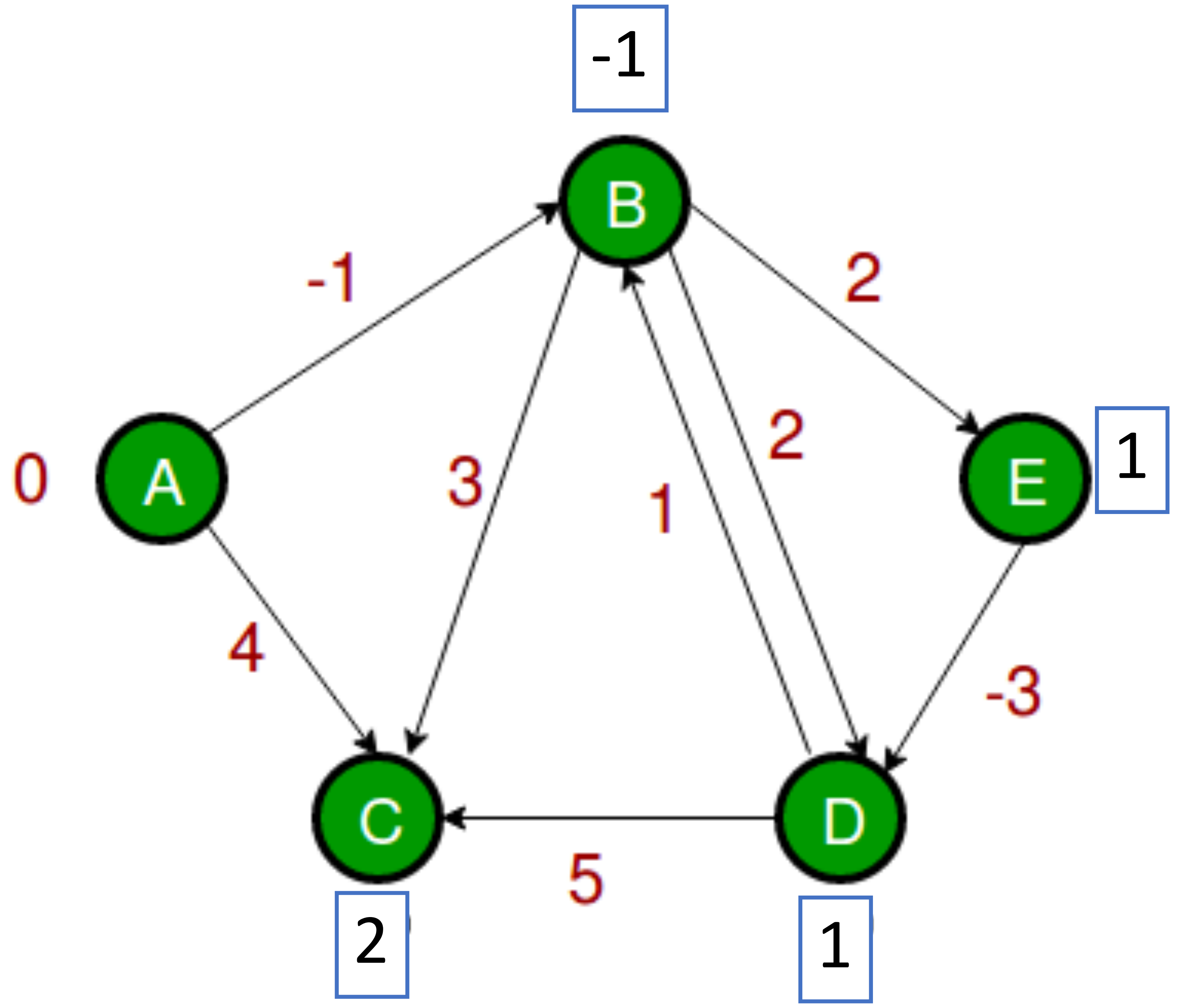




Tutorial and dynamic programming implementation here.







AB: -1
AC: 4

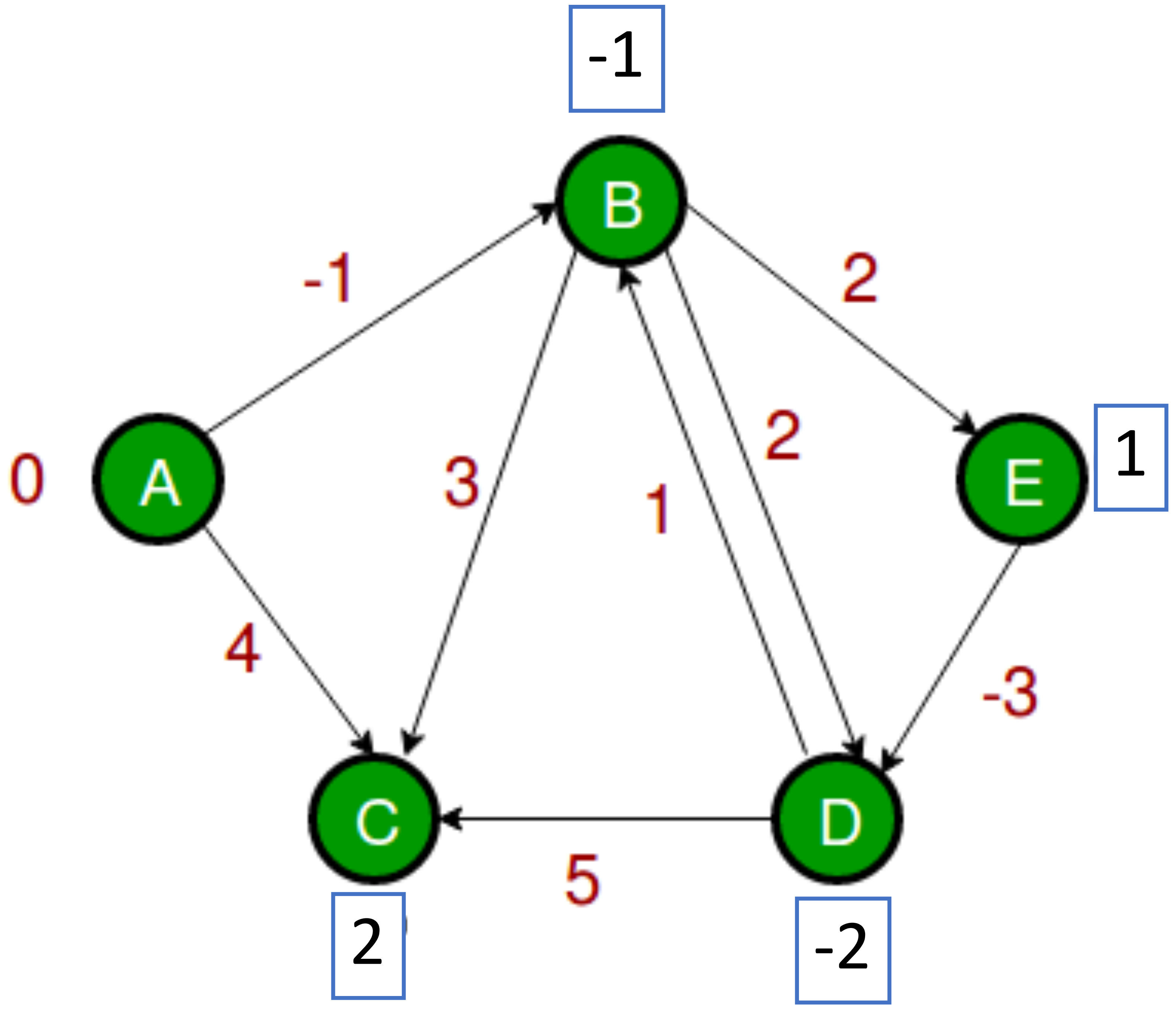
B: -1
C: 4

BC: 3
BE: 2
BD: 2

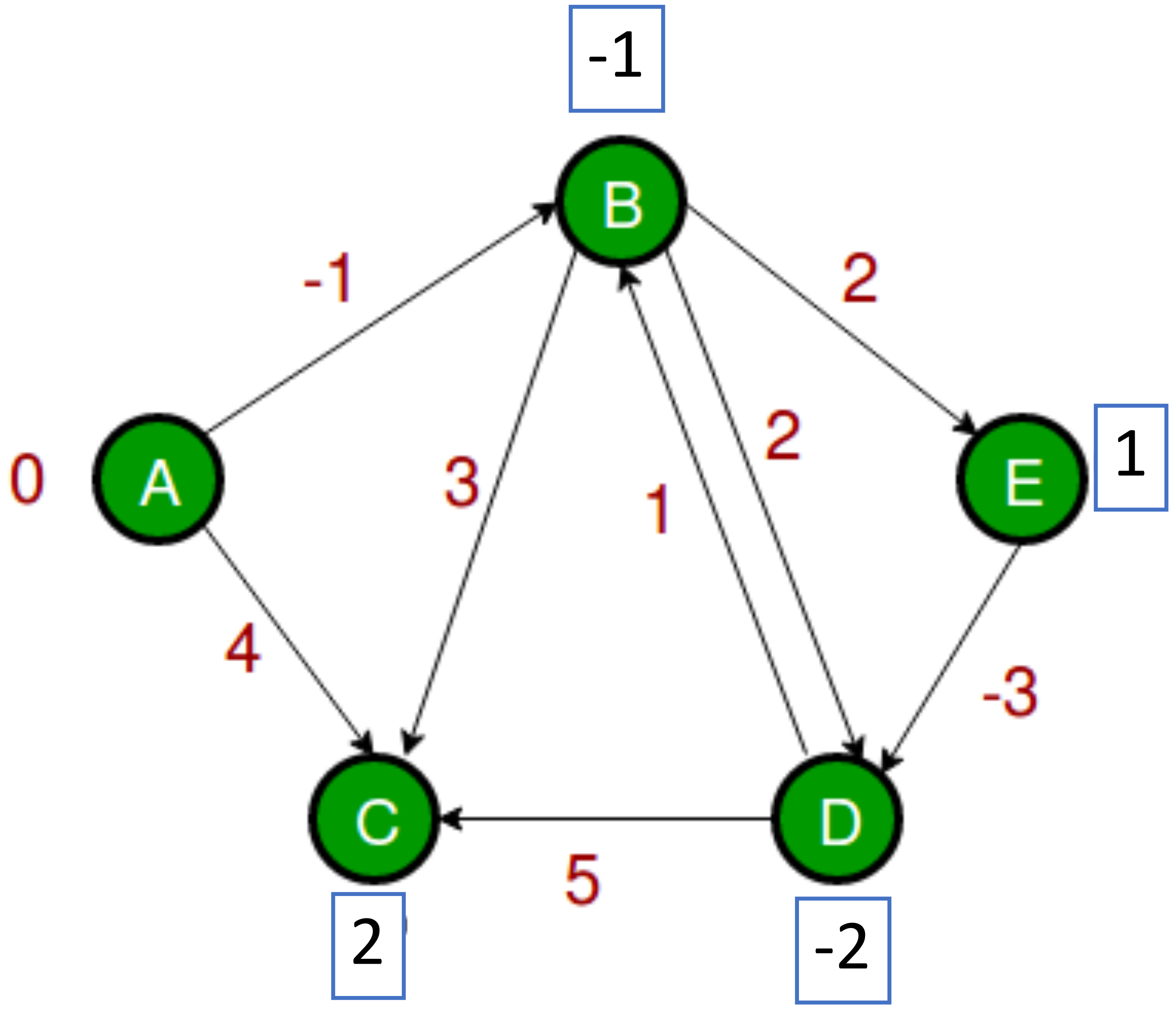
C: 2
E: 1
D: 1

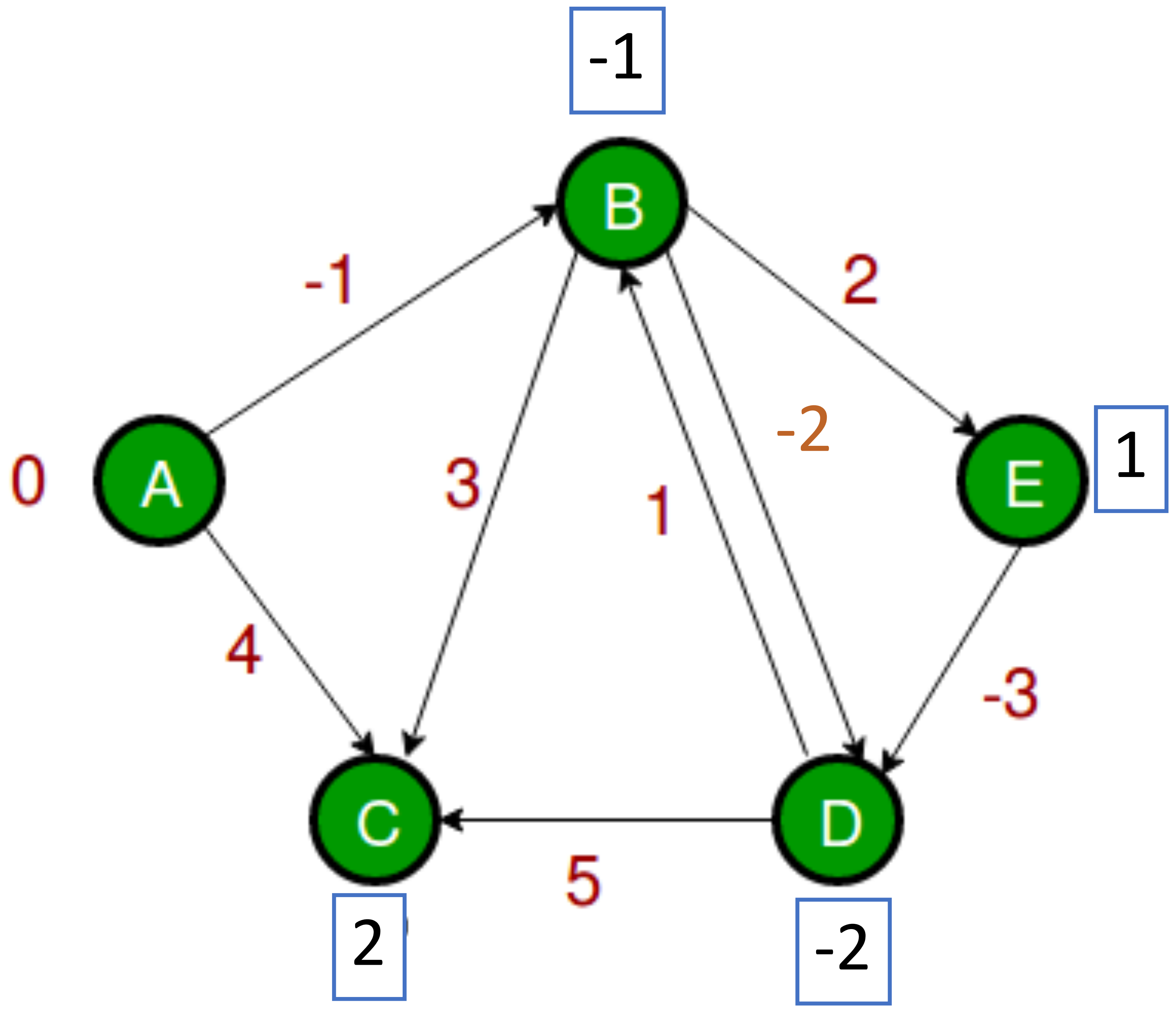
ED: -3
DB: 1
DC: 5

D: -2




```
function BellmanFord(list vertices, list edges, vertex source) is

    // This implementation takes in a graph, represented as
    // lists of vertices (represented as integers [0..n-1]) and edges,
    // and fills two arrays (distance and predecessor) holding
    // the shortest path from the source to each vertex

    distance := list of size n
    predecessor := list of size n

    // Step 1: initialize graph
    for each vertex v in vertices do

        distance[v] := inf              // Initialize the distance to all vertices to infinity
        predecessor[v] := null          // And having a null predecessor

    distance[source] := 0               // The distance from the source to itself is, of course, zero

    // Step 2: relax edges repeatedly

    repeat |V|−1 times:
         for each edge (u, v) with weight w in edges do
             if distance[u] + w < distance[v] then
                 distance[v] := distance[u] + w
                 predecessor[v] := u

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            error "Graph contains a negative-weight cycle"

    return distance, predecessor
```



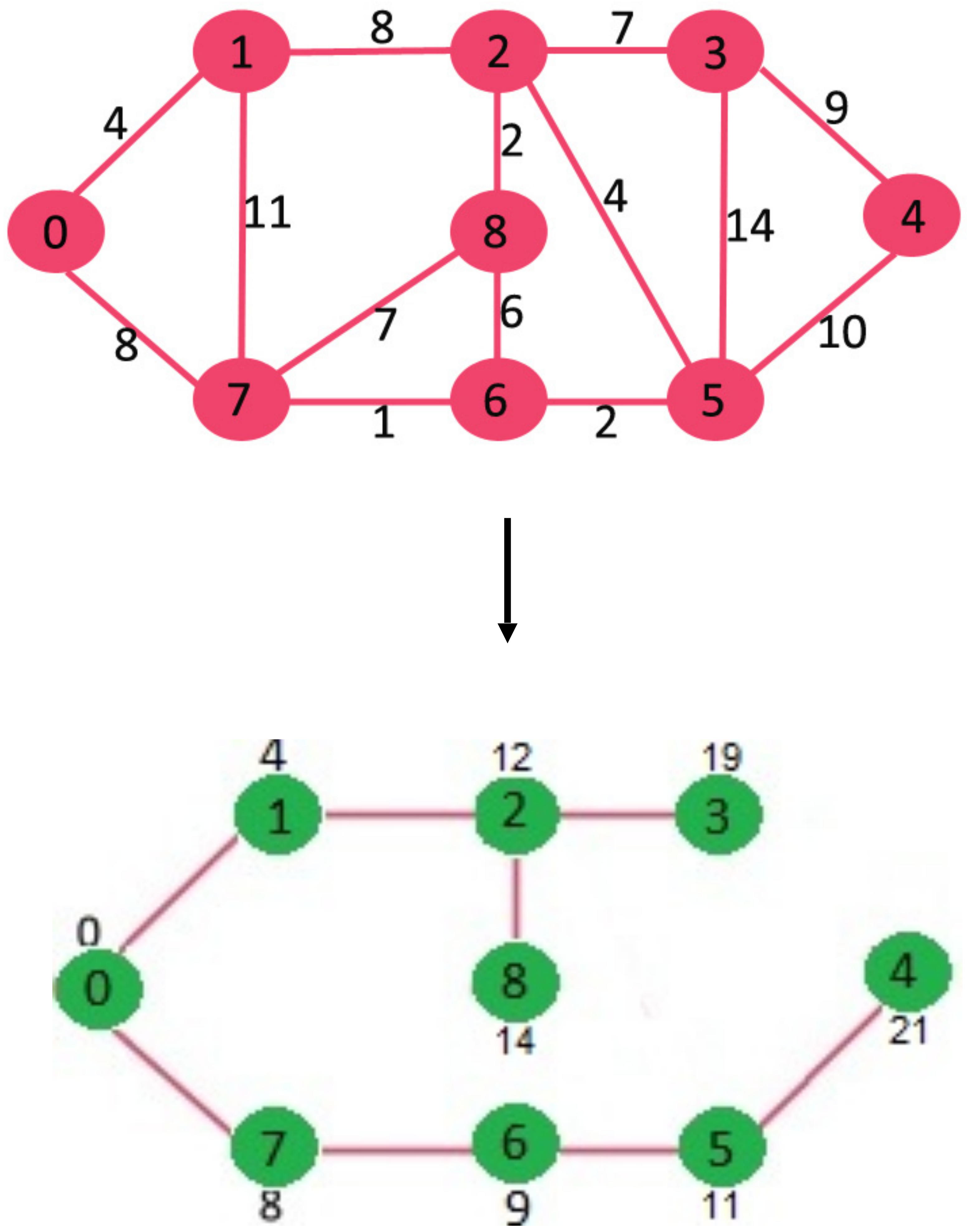


Negative-weight cycle

- Question 1: what's the time complexity?
- Question 2: if it's an "unconstrained" question (we are not starting from A), what's the shortest path?

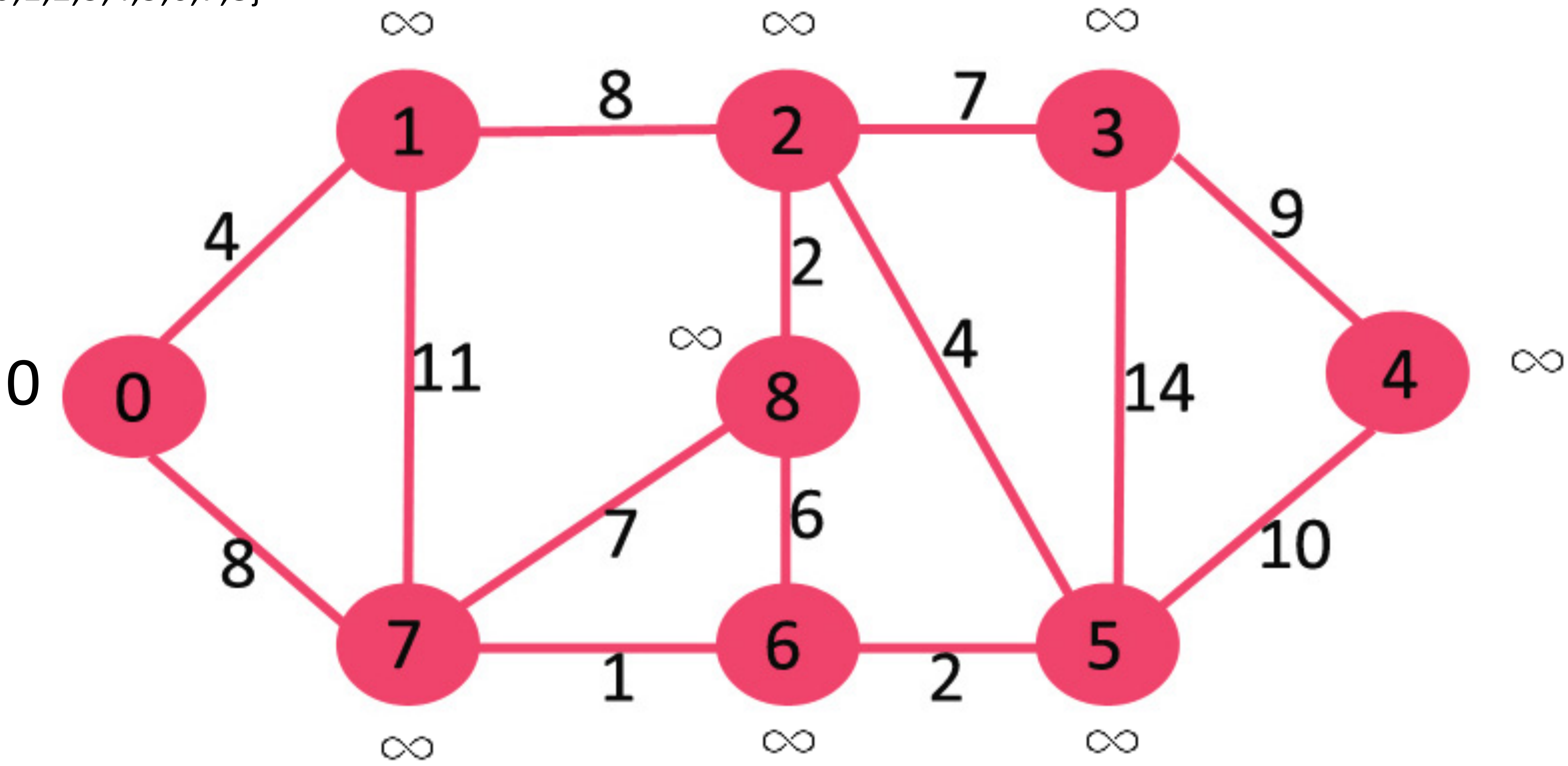
What if graph has cycles? (no negative edges)

- Dijkstra's algorithm (for given start vertex)
  - Choose source node and set distance to 0
  - Set distance to all other nodes to infinity
  - Set source node to **current**
  - Make distance offers to all **unvisited** neighbors, which are accepted if they're less than the previous best offer
  - Mark current as **visited** (it will never be updated again)
  - Select unvisited neighbor with smallest distance, set it to current, and repeat
  - (When destination node has been marked visited, stop)

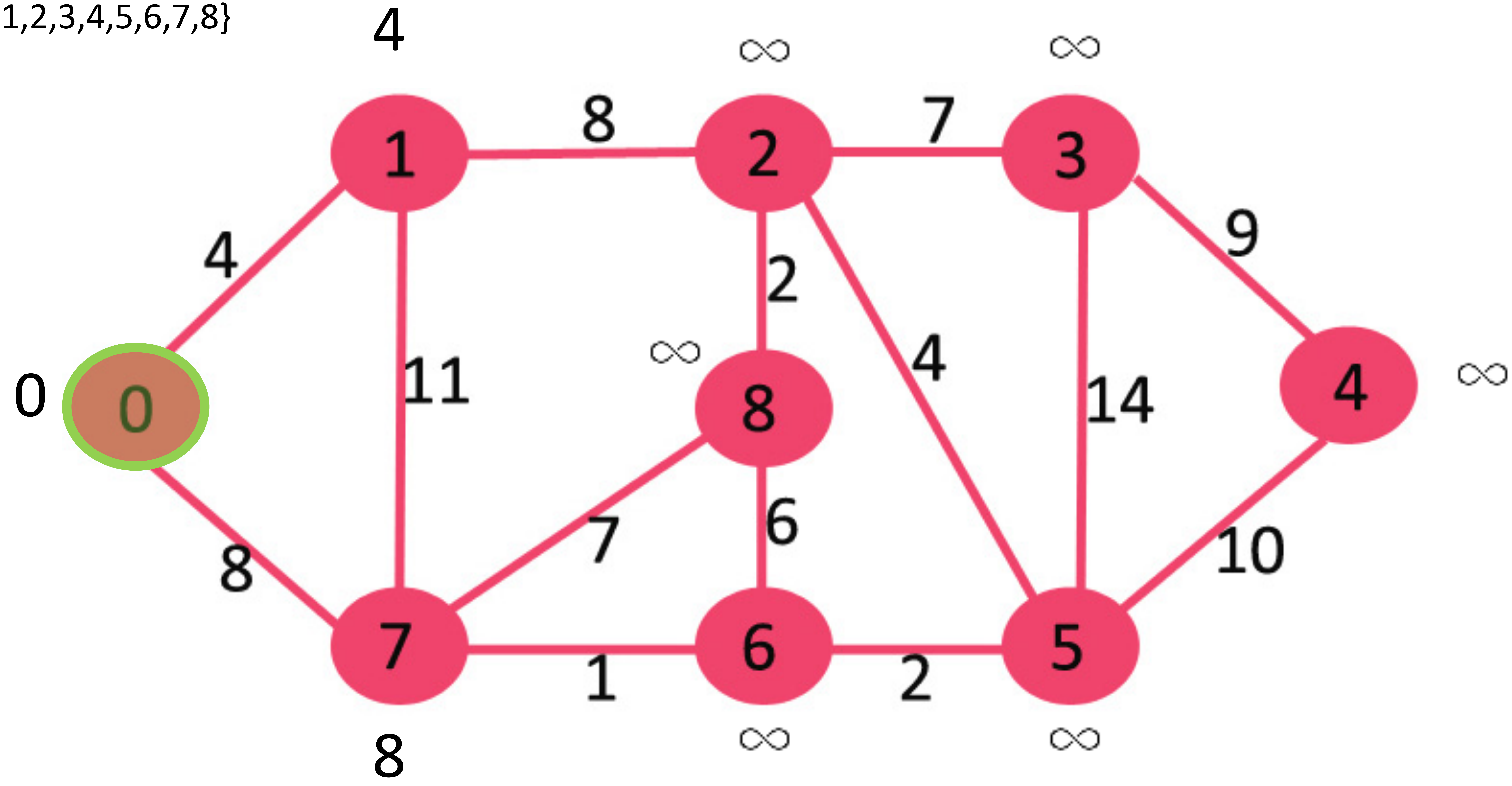


1) Taking less time;
2) only working on graph with no negative edges

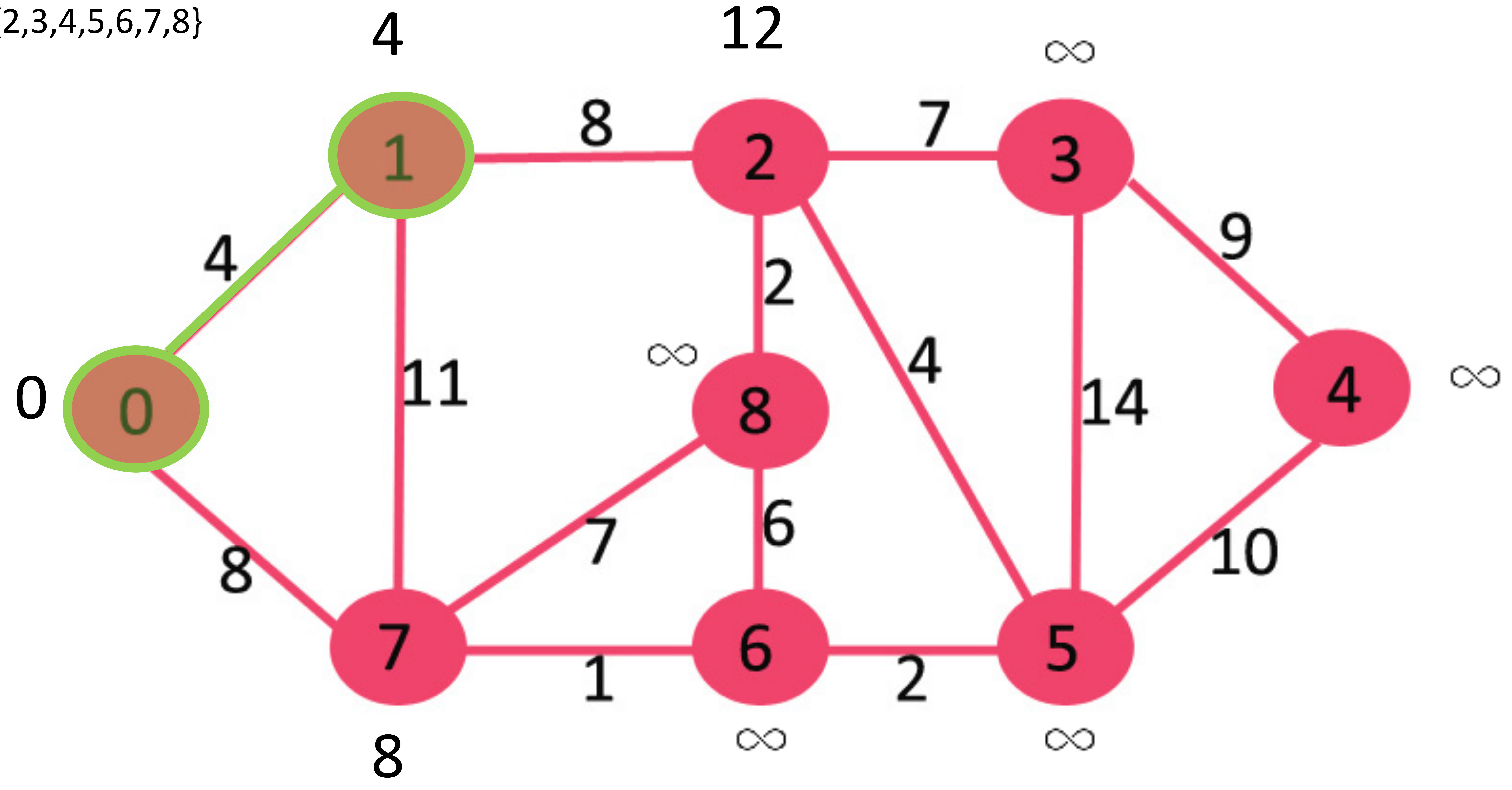
Current: {}
Visited: {}
Unvisited: {0,1,2,3,4,5,6,7,8}



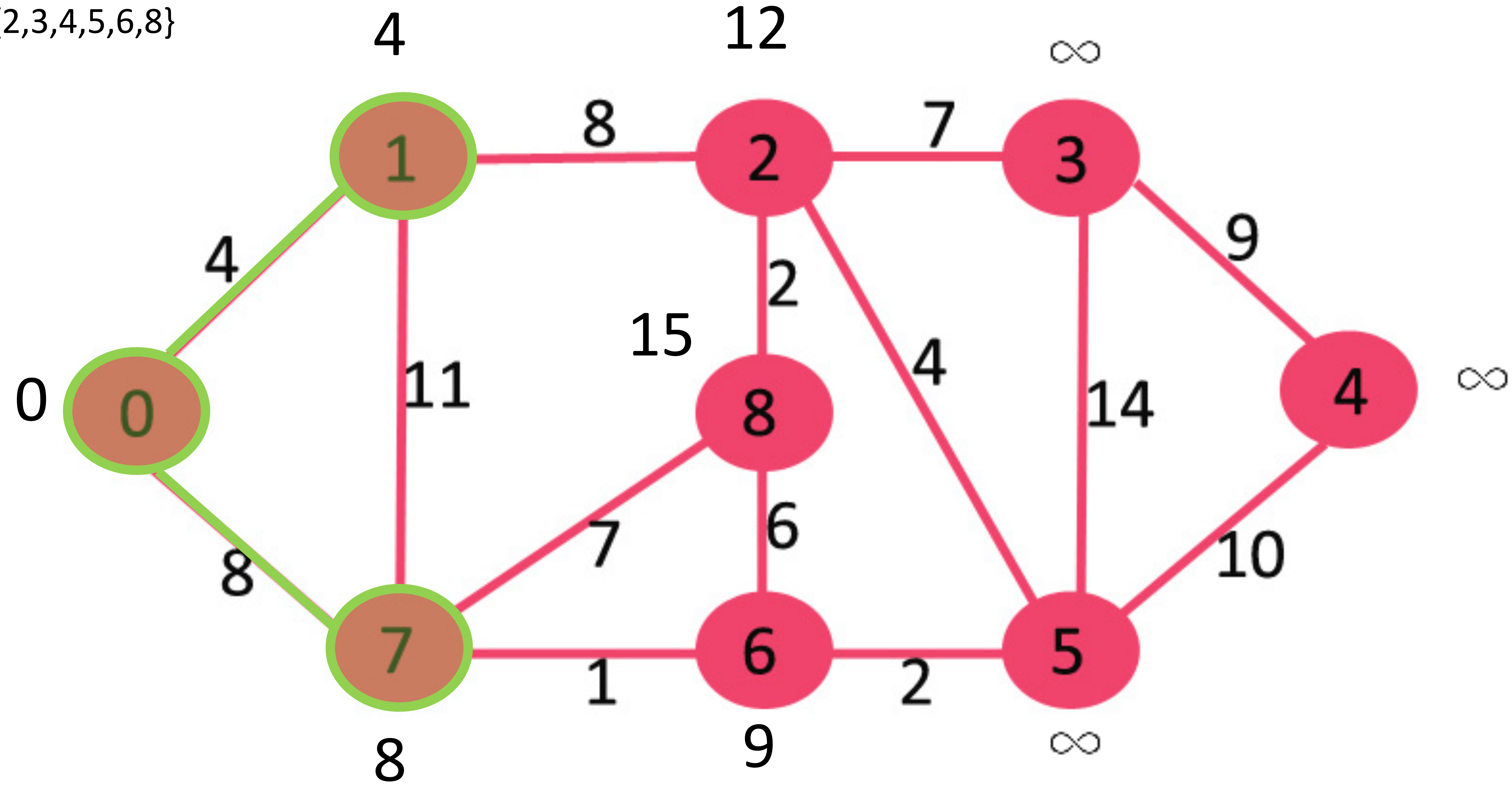
Current: {0}
Visited: {}
Unvisited: {1,2,3,4,5,6,7,8}



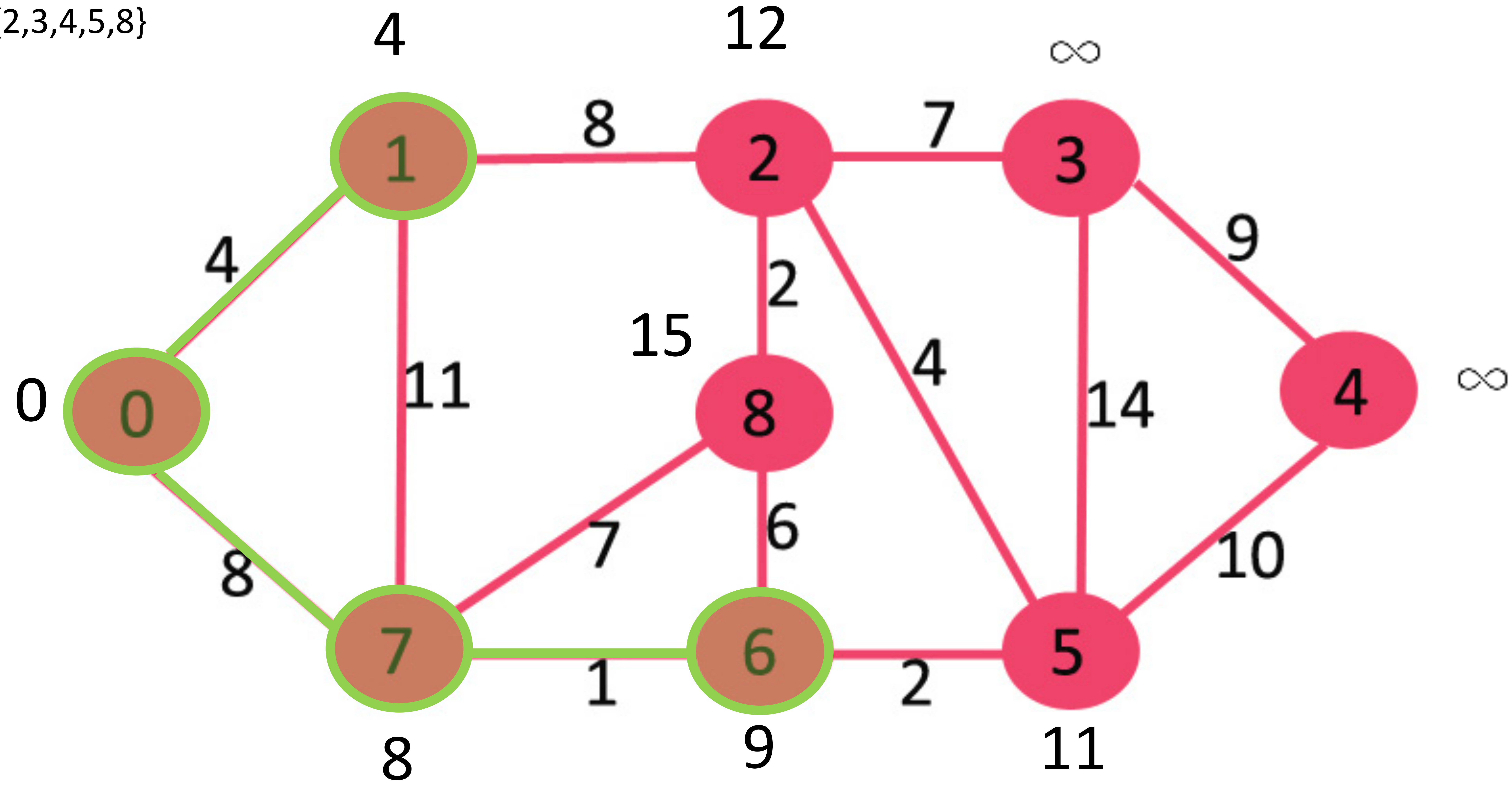
Current: {1}
Visited: {0}
Unvisited: {2,3,4,5,6,7,8}



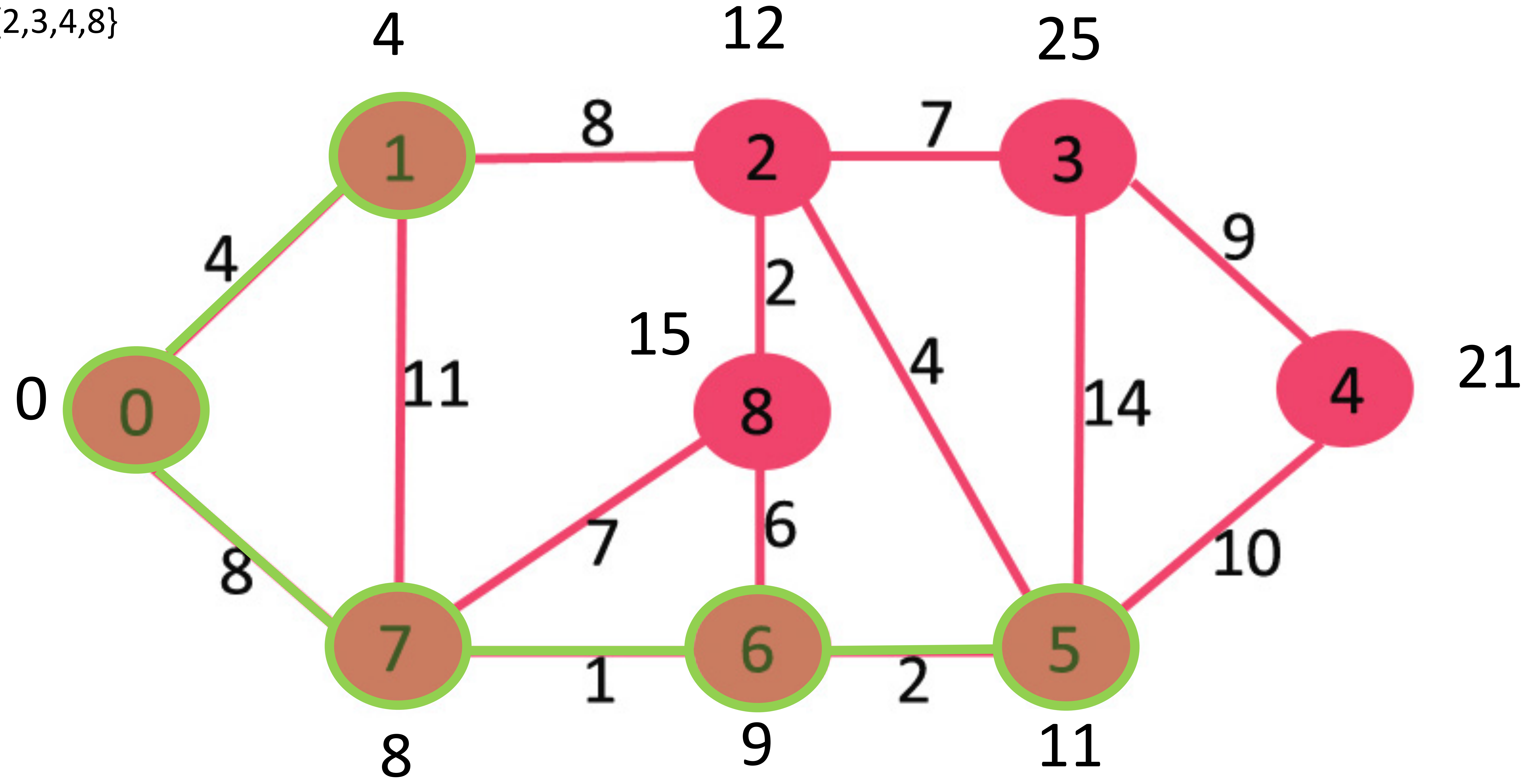
Current: {7}
Visited: {0,1}
Unvisited: {2,3,4,5,6,8}



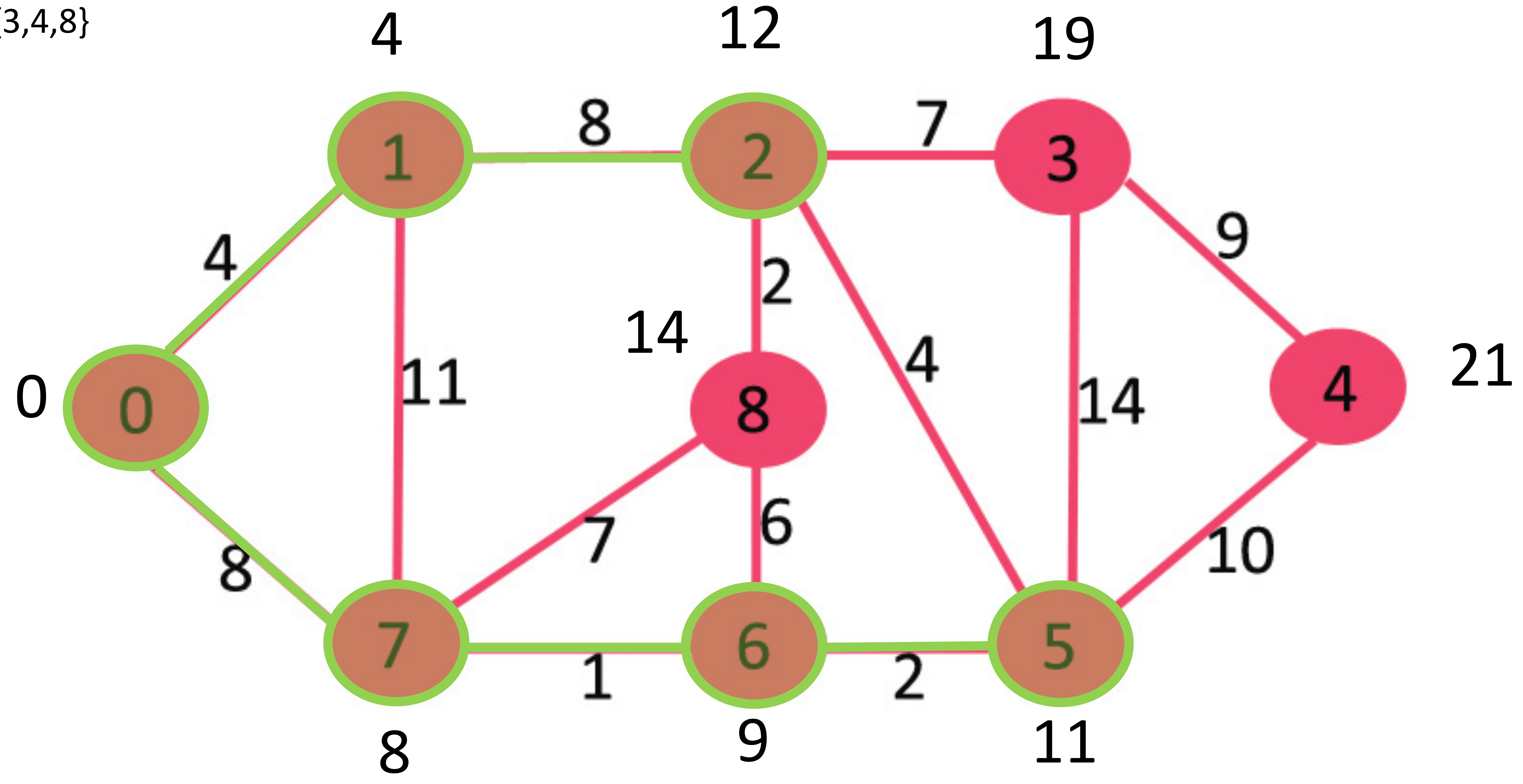
Current: {6}
Visited: {0,1,7}
Unvisited: {2,3,4,5,8}



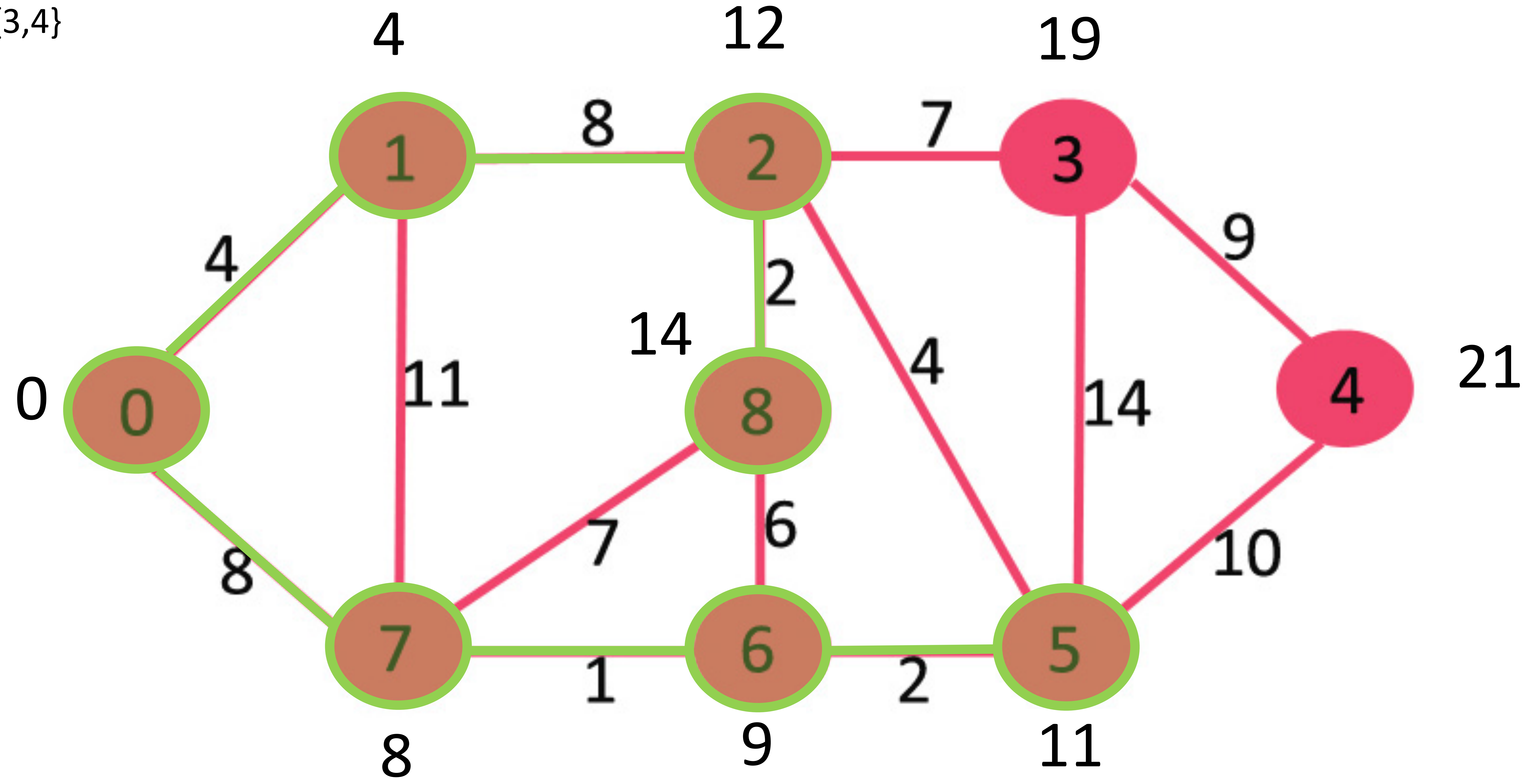
Current: {5}
Visited: {0,1,7,6}
Unvisited: {2,3,4,8}



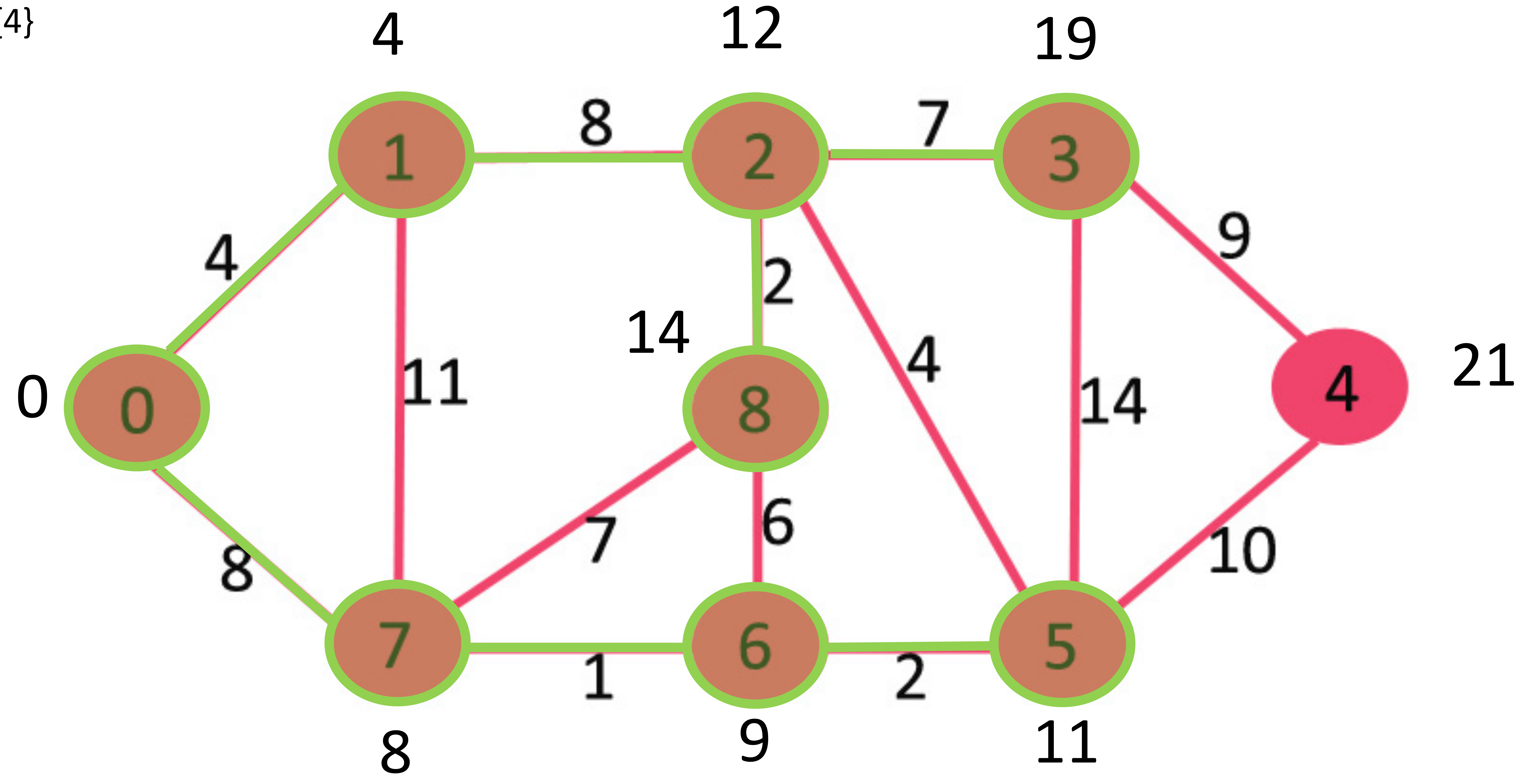
Current: {2}
Visited: {0,1,7,6,5}
Unvisited: {3,4,8}



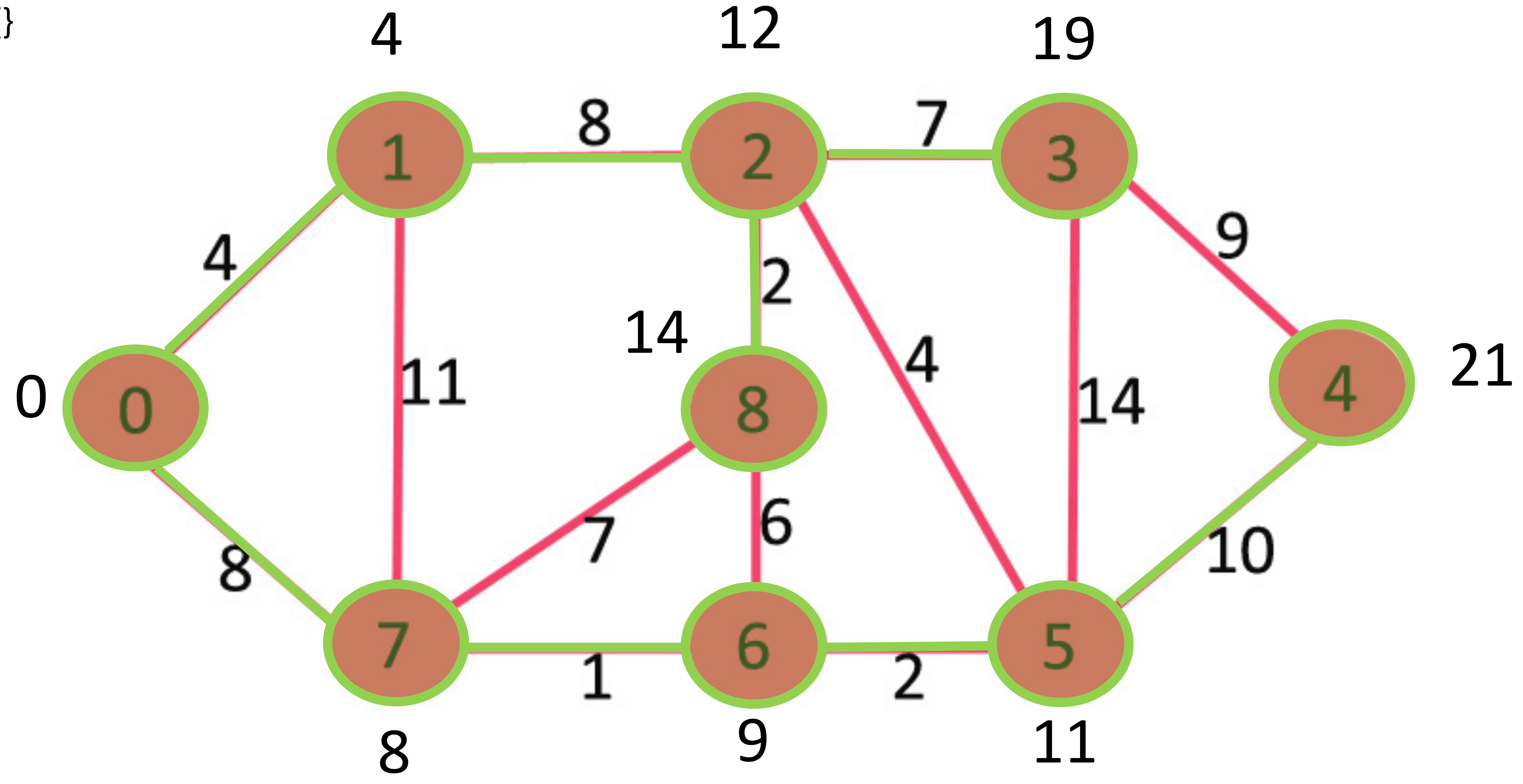
Current: {8}
Visited: {0,1,7,6,5,2}
Unvisited: {3,4}

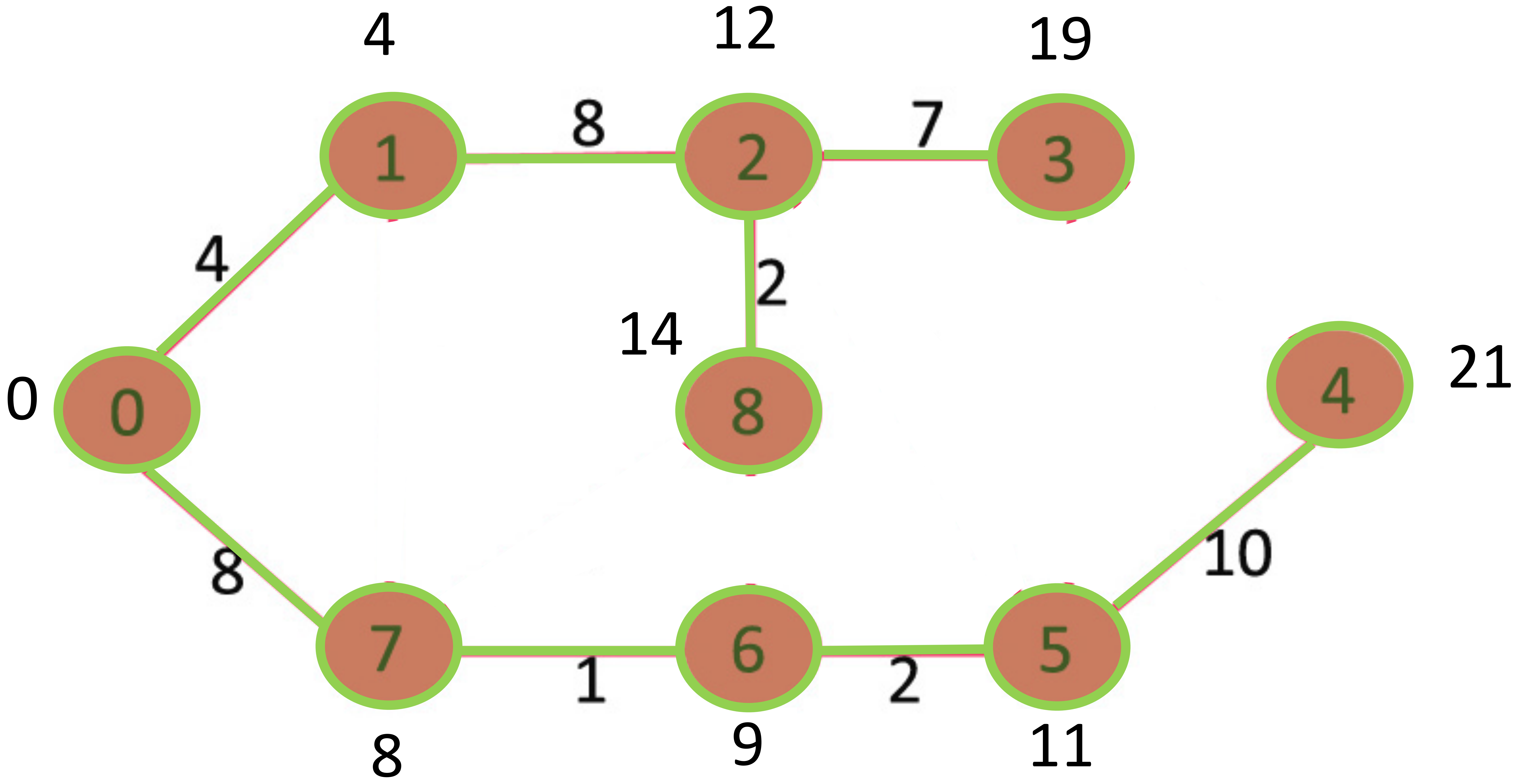


Current: {3}
Visited: {0,1,7,6,5,2,8}
Unvisited: {4}



Current: {4}
Visited: {0,1,7,6,5,2,8,3}
Unvisited: {}





4
12
19
1
8
2
7
3
4
2
14
0
0
8
4
21
8
10
7
1
6
2
5
8
9
11