GRAPH TRAVERSAL ALGORITHM: DEPTH FIRST SEARCH (DFS)

WHY STUDY GRAPH TRAVERSAL ALGORITHMS?

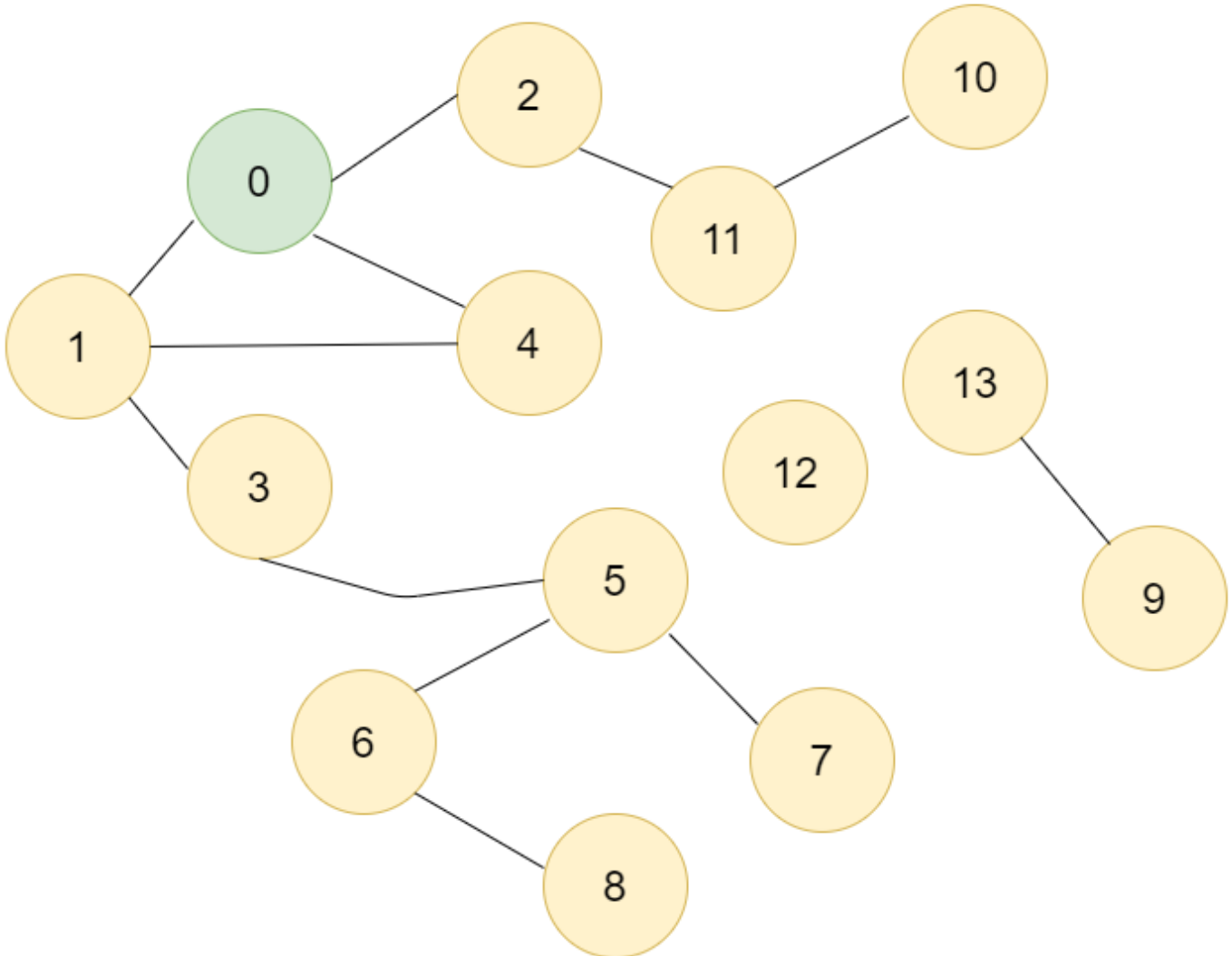
- Graph Reachability Problems
- Cycle Detection
- Topological Sort
- Connected Components
- Flood Fill Problems
- Shortest path in unweighted Graphs
- Strongly connected components (Tarjan's)
  - Only for directed graphs.
- Biconnected Components
- Finding path in an unweighted graph
- Bipartite graph Check
- Articulation Points / Bridges

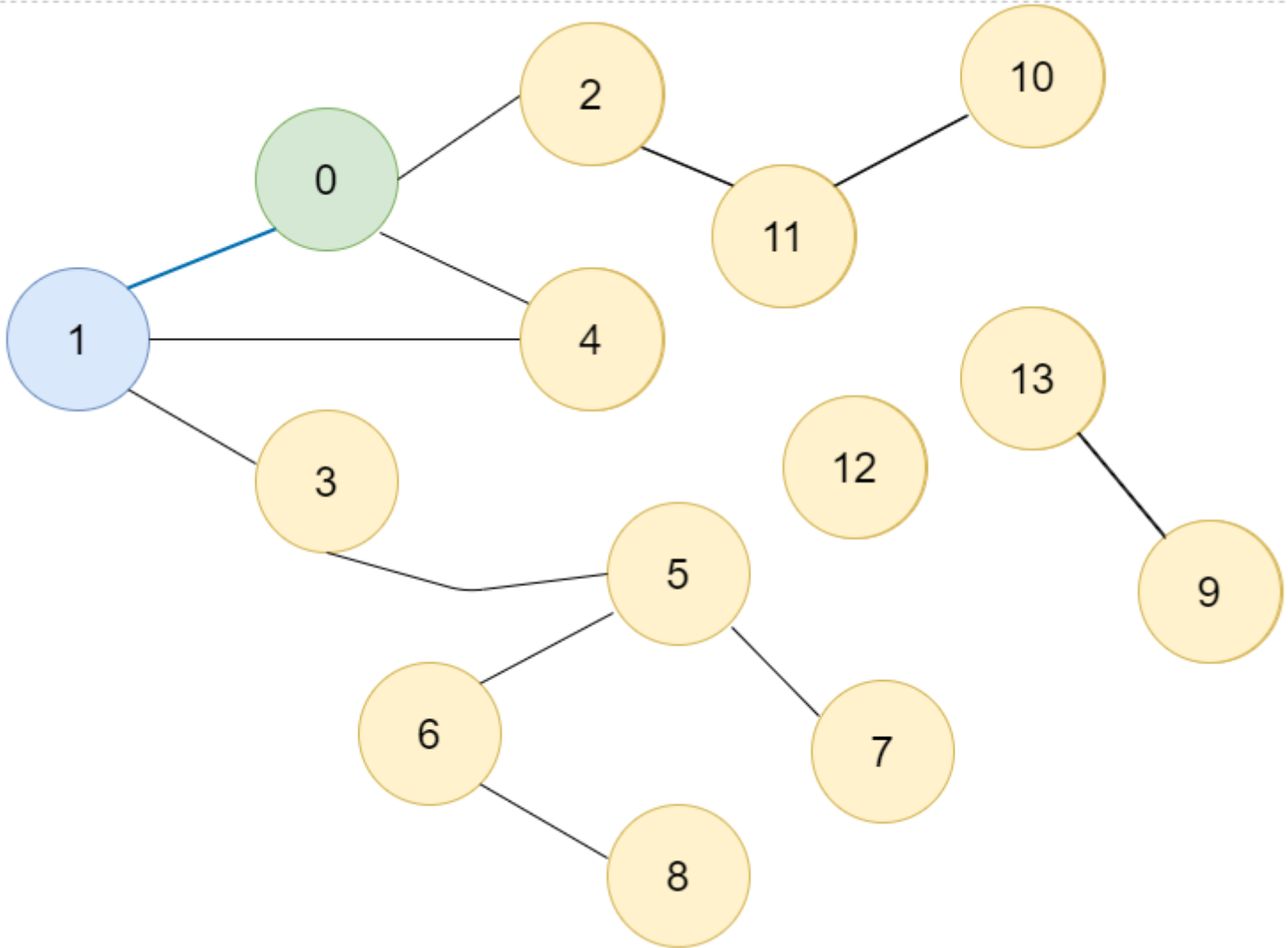
GRAPH TRAVERSAL ALGORITHMS

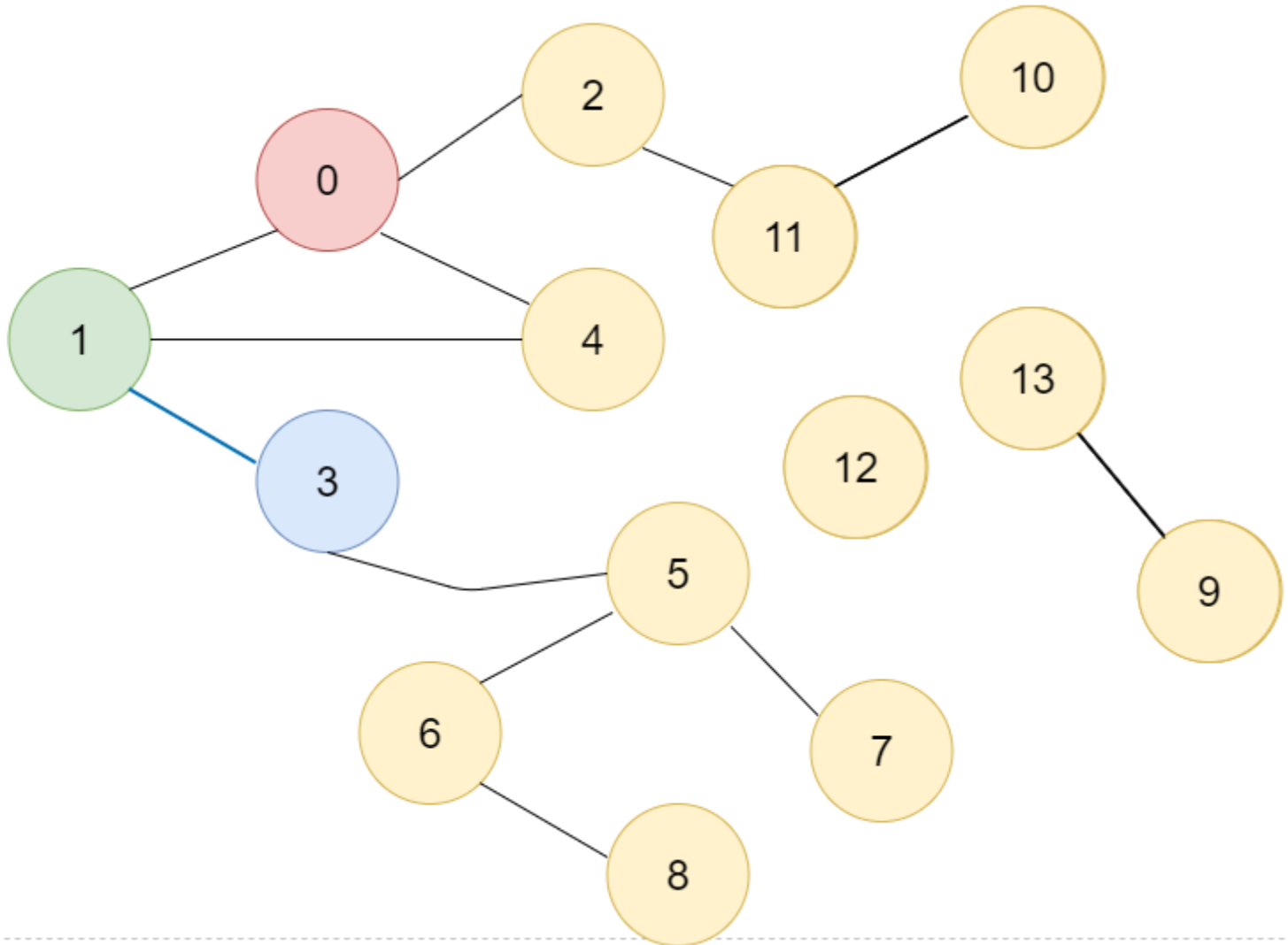
- Depth First Search (DFS)
- Breadth First Search (BFS)

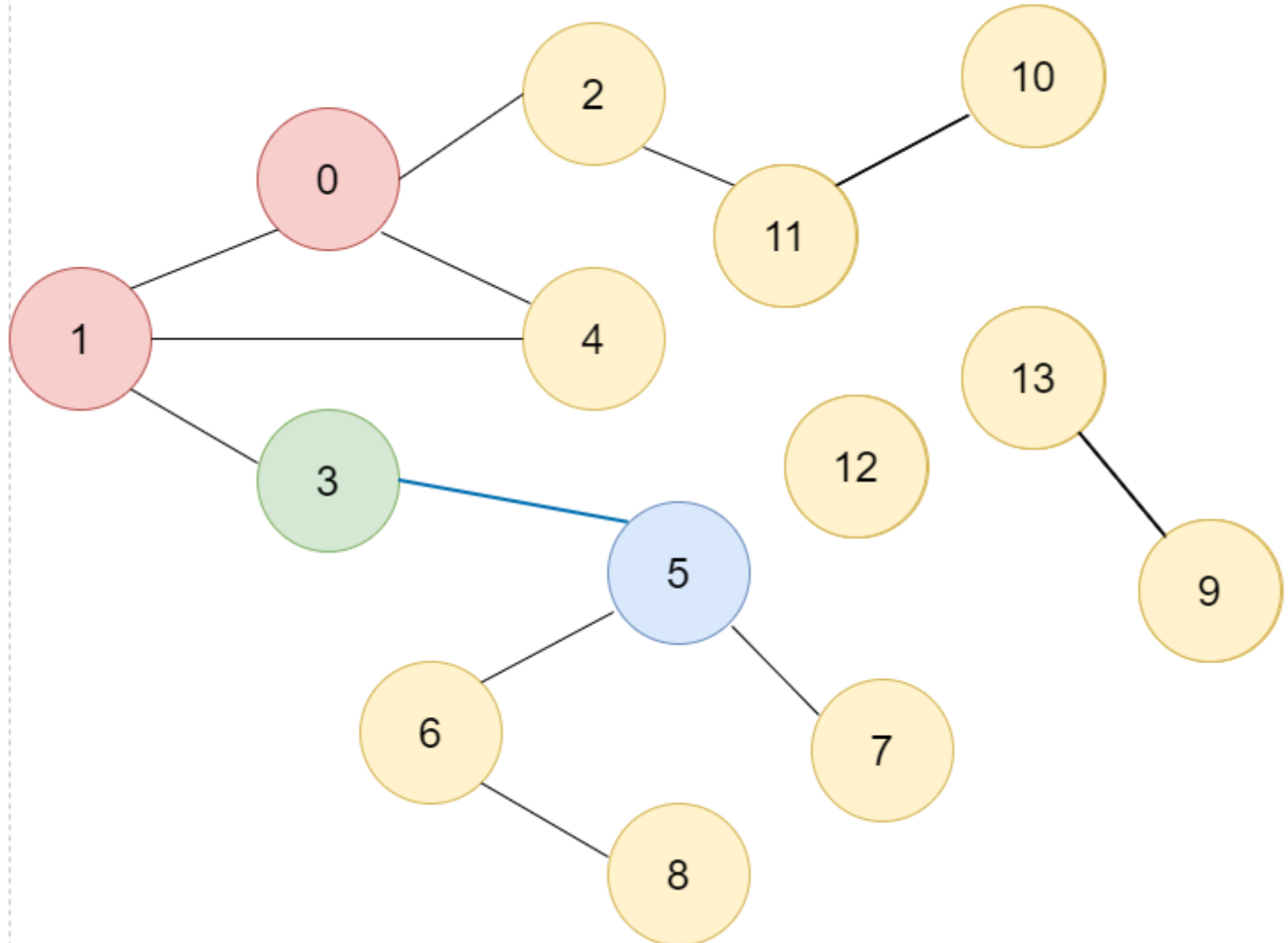
HOW DEPTH FIRST SEARCH (DFS) WORK?

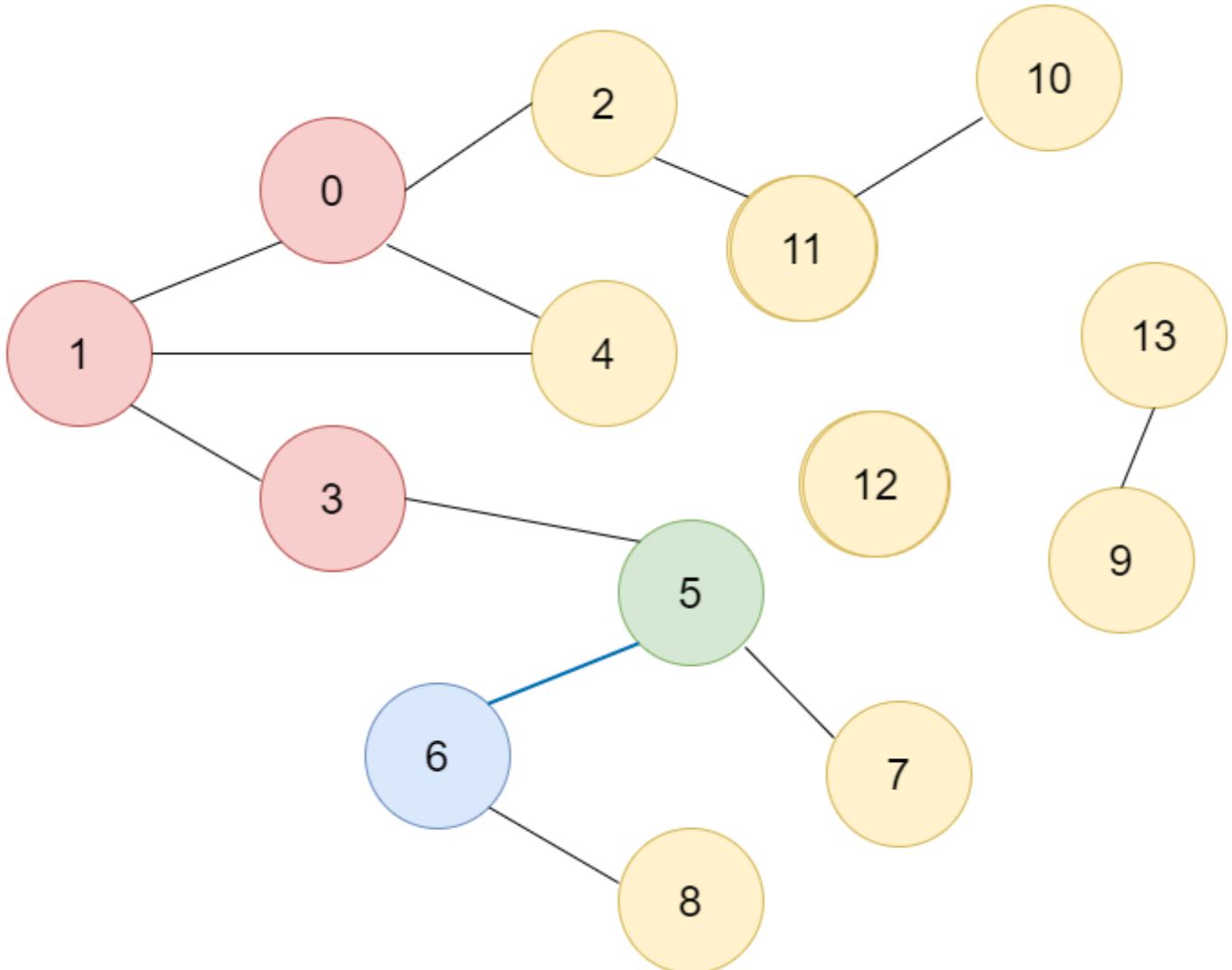
- Start from a single node and traverse the graph in a depth first fashion.
  - Do depth first search at the adjacent nodes.
  - Keep going deeper until reaches a vertex which does not have any unvisited vertex.
  - Then backtrack (return) to the previous node and explore other branches/paths from that node.
- Follows a single path as long as it finds a new node.
  - Explores a branch completely before moving on to a new branch of the graph
  - It goes deep first and branches later.
- Keep tracks of the visited nodes so that no nodes get processed more than once.

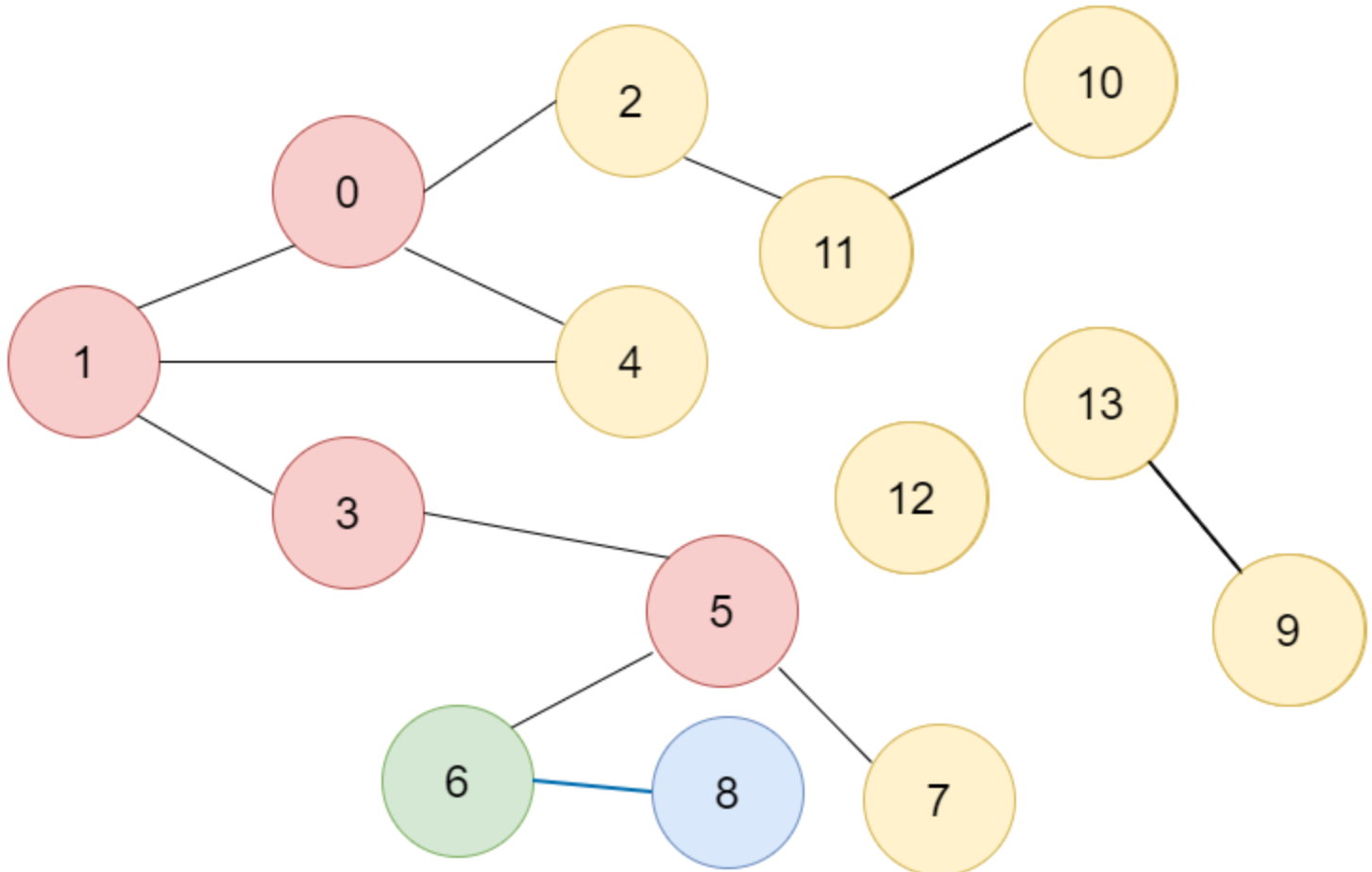

0
1
2
3
4
5
6
7
8
9
10
11
12
13

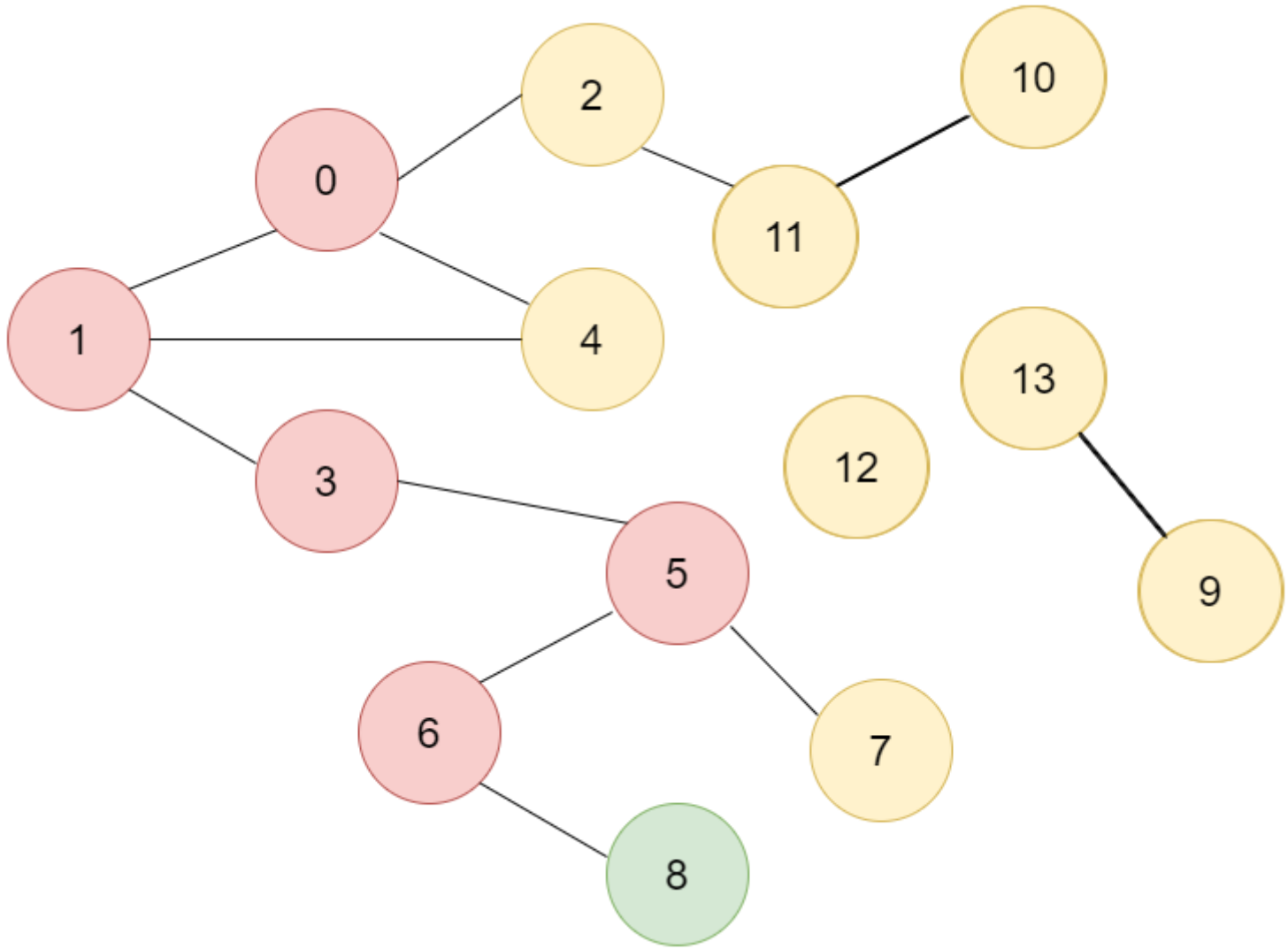

2
10
0
11
1
4
13
12
3
9
5
6
7
8

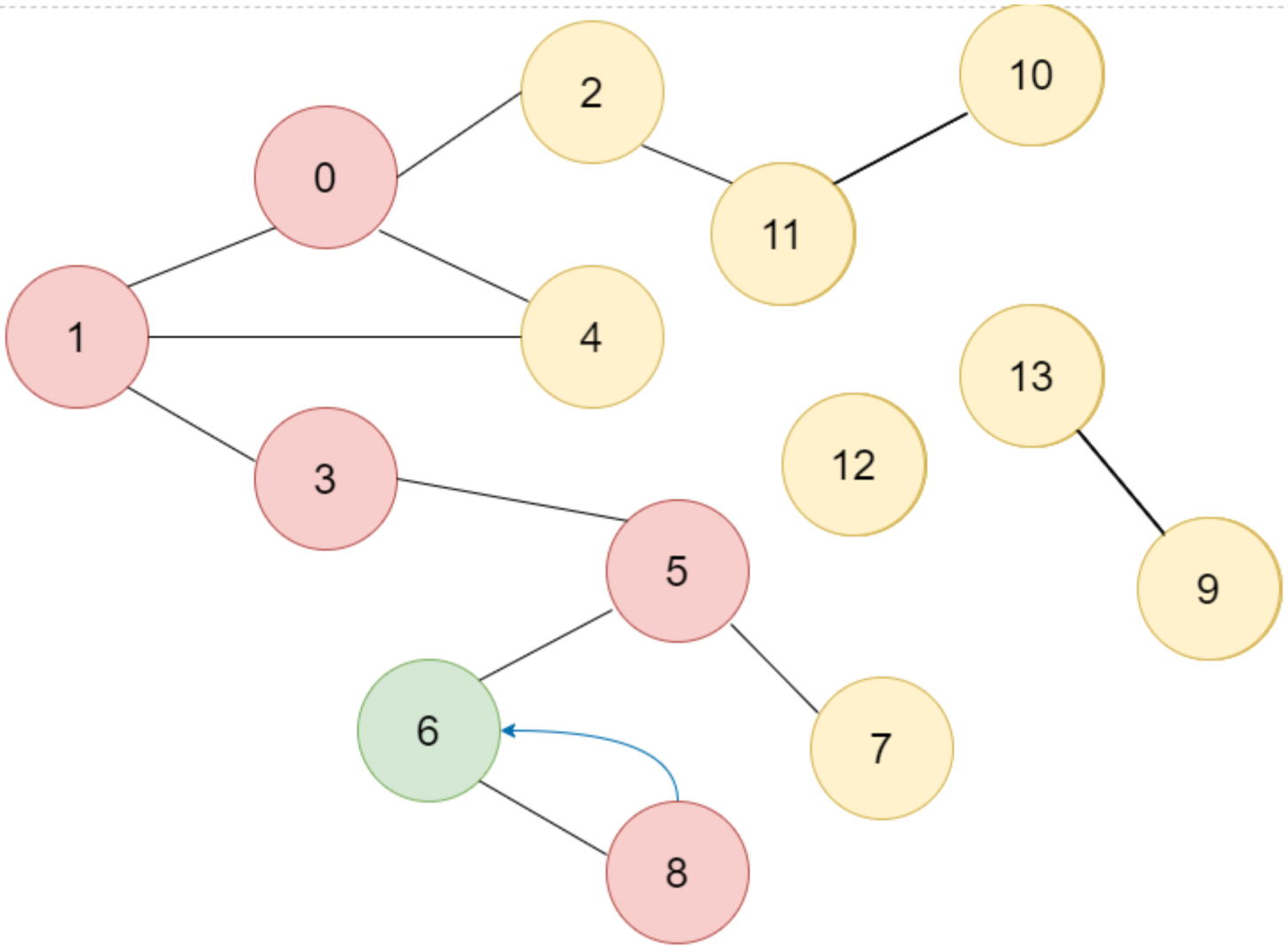

0
1
2
3
4
5
6
7
8
9
10
11
12
13

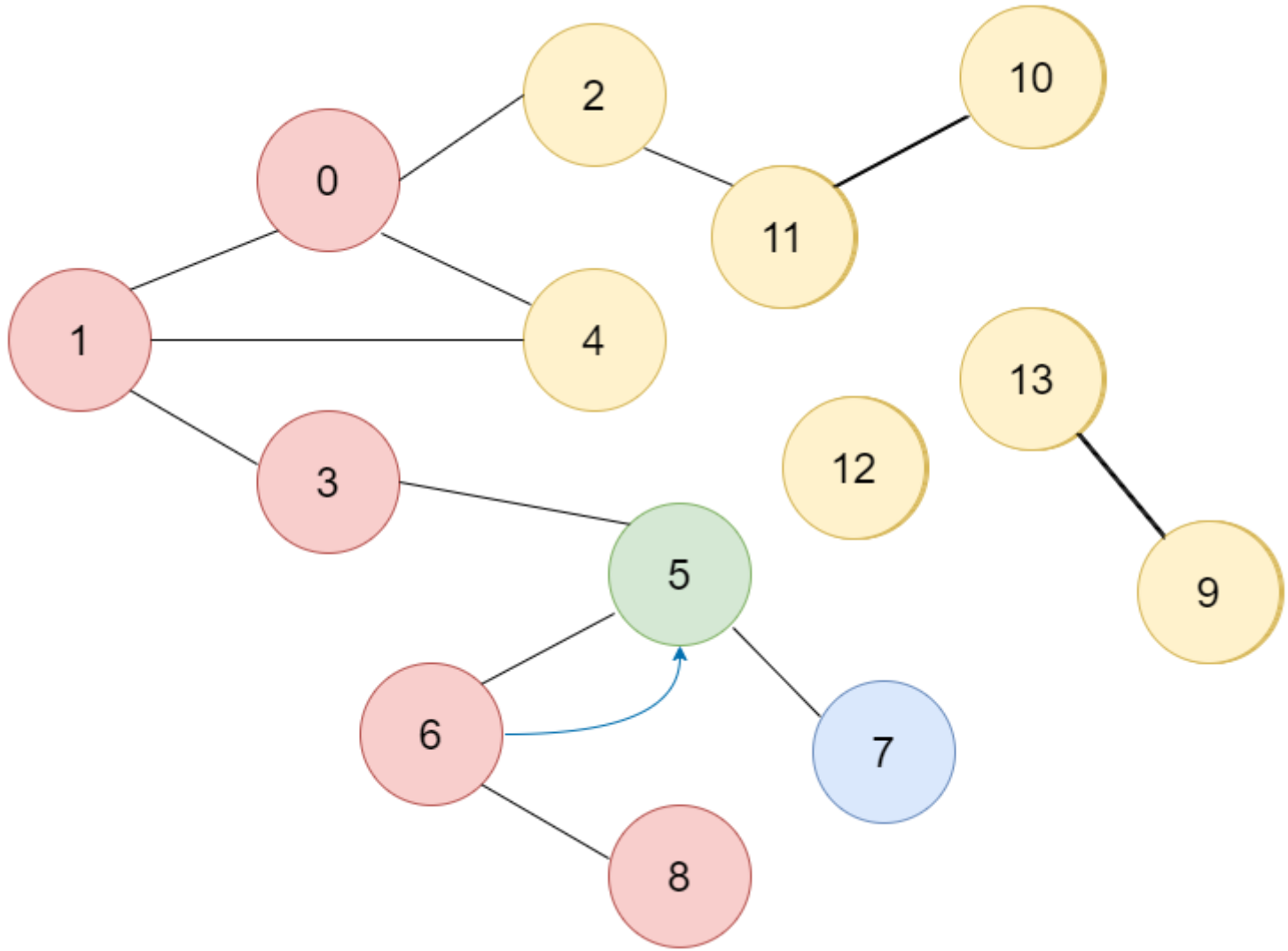

0
1
2
3
4
5
6
7
8
9
10
11
12
13

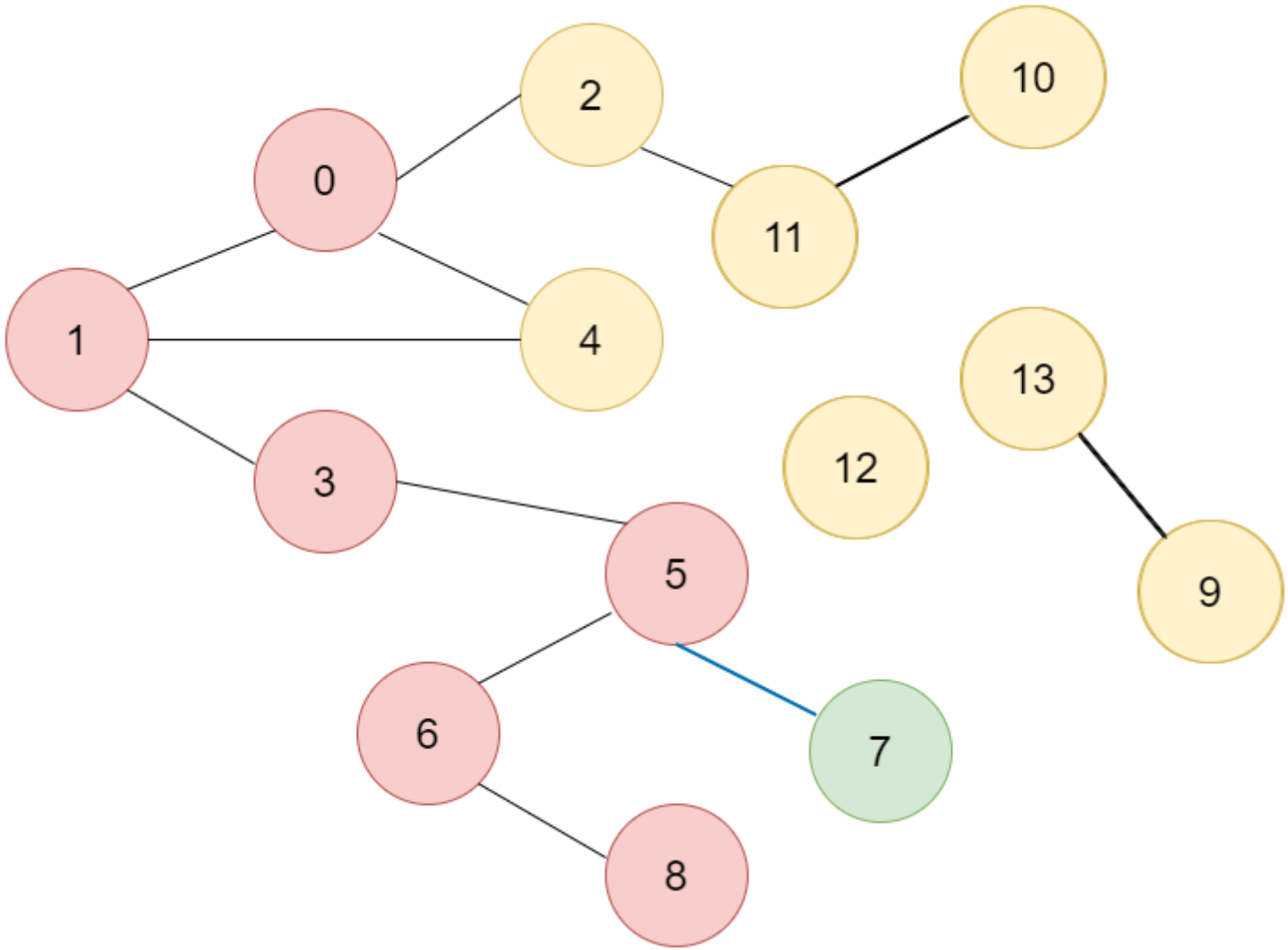

2
10
0
11
13
1
4
12
3
9
5
6
7
8

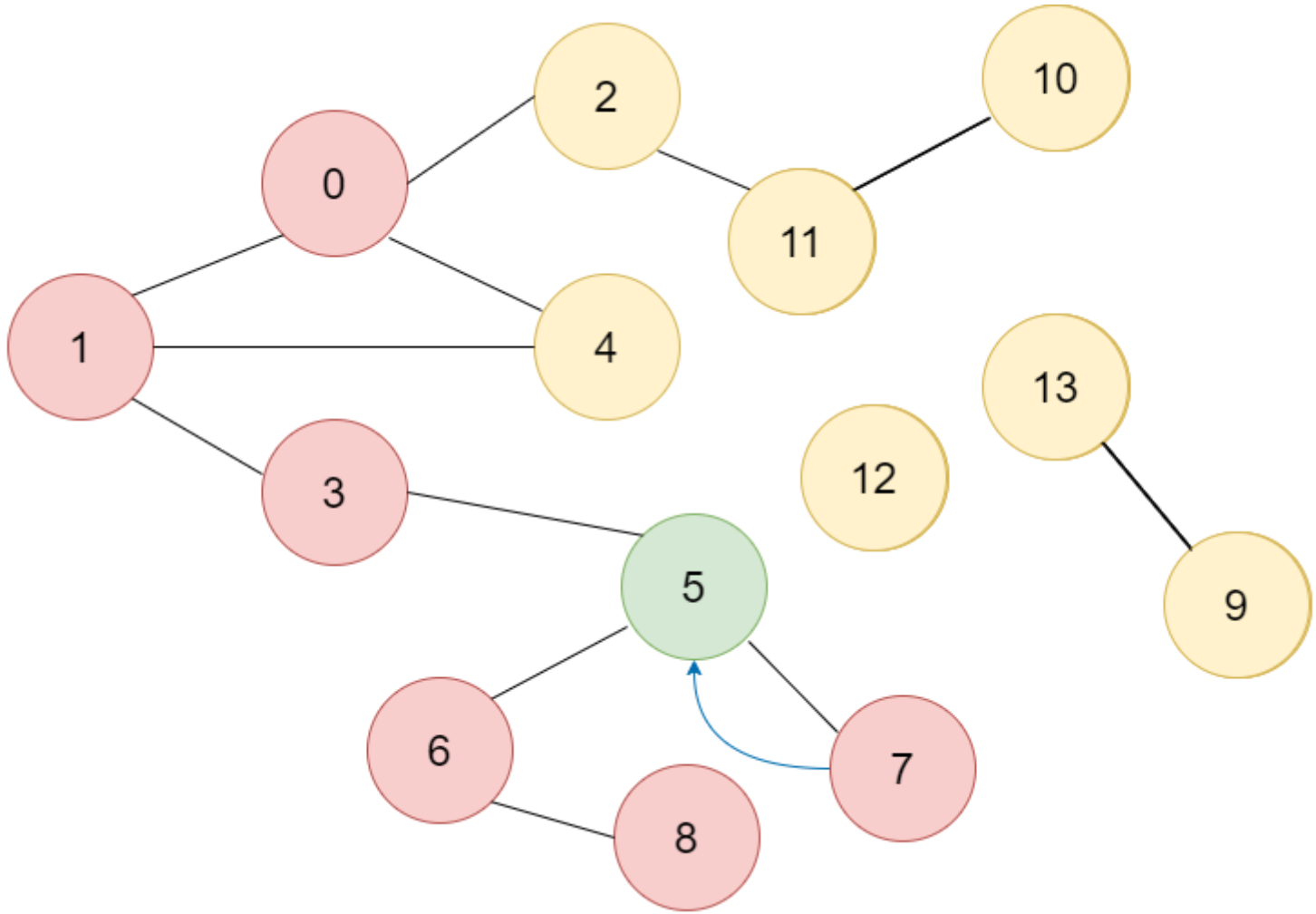

2
10
0
11
1
4
13
12
3
9
5
6
8
7

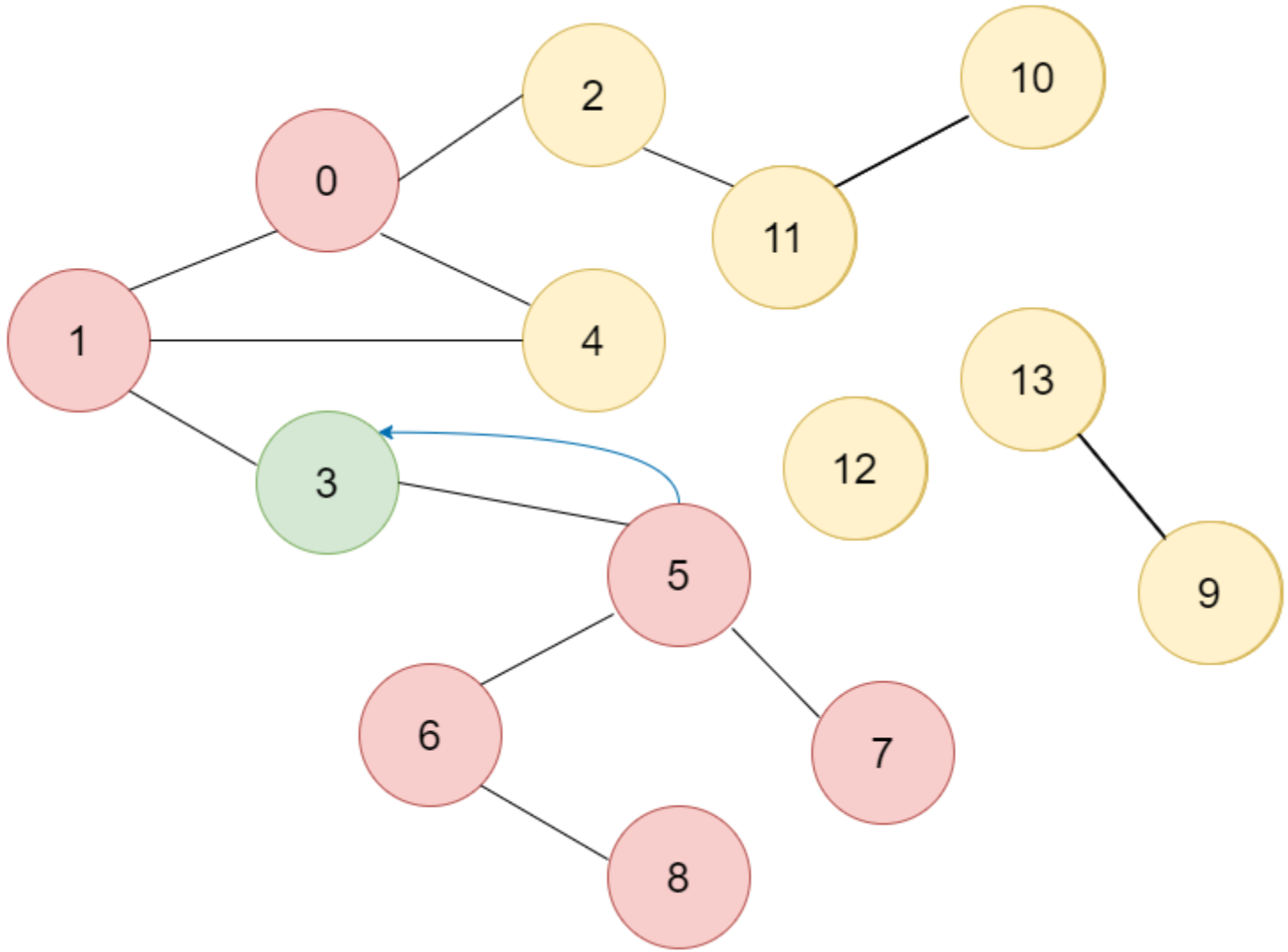

2
10
0
11
1
4
13
12
3
9
5
6
7
8

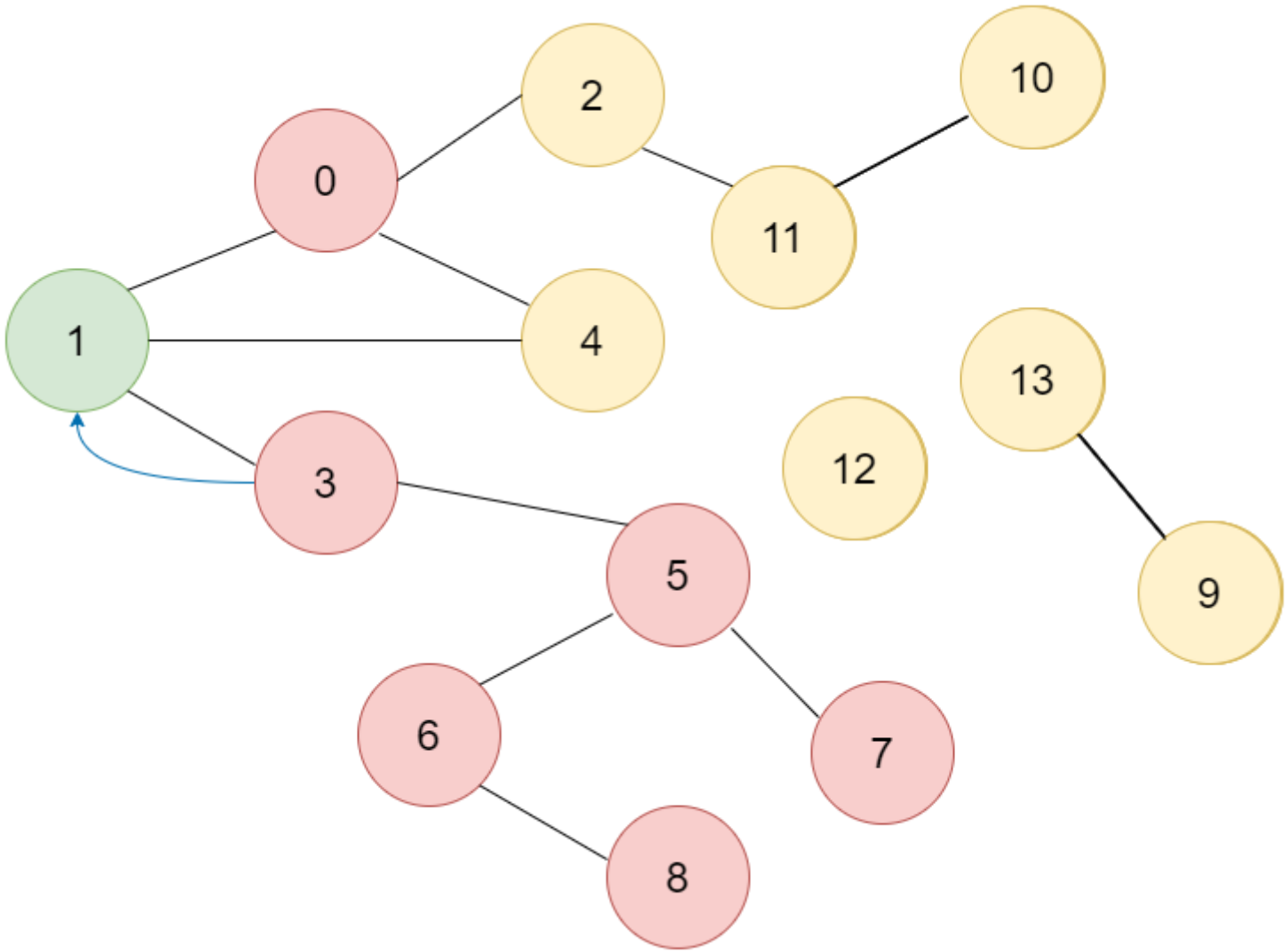

2
10
0
11
1
4
13
12
3
9
5
6
7
8

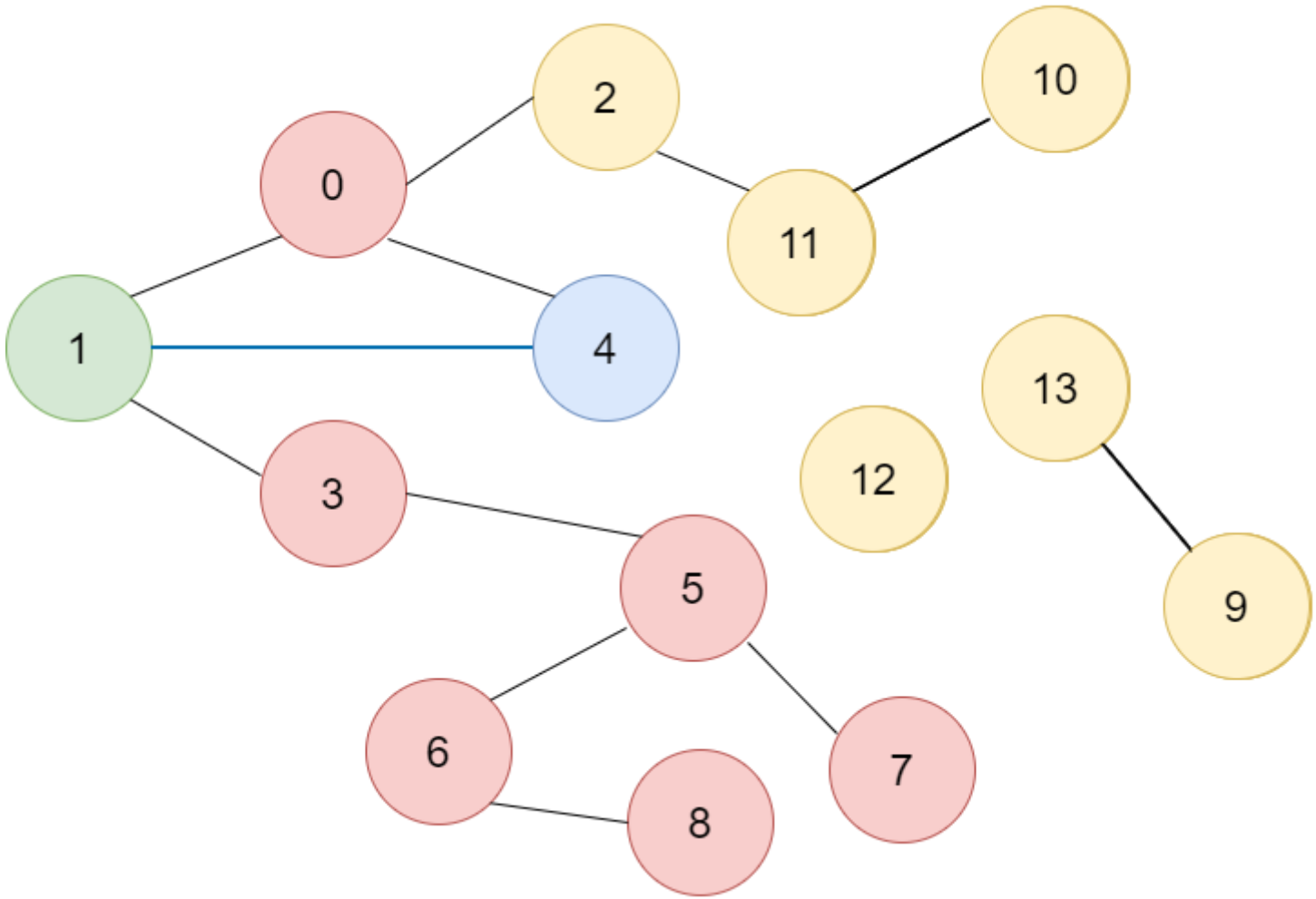

2
10
0
11
1
4
13
12
3
9
5
6
7
8

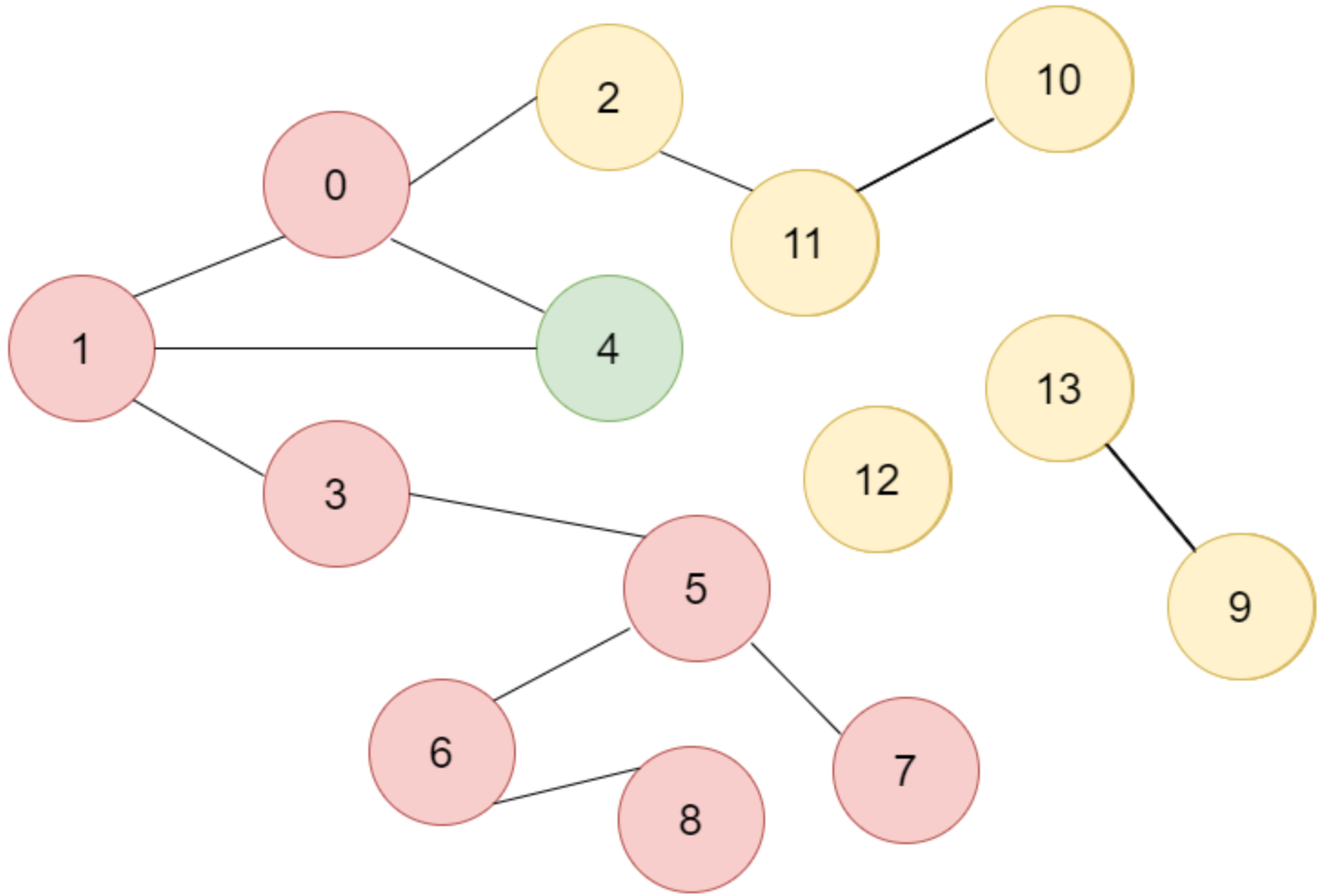

2
10
0
11
1
4
13
12
3
9
5
6
7
8

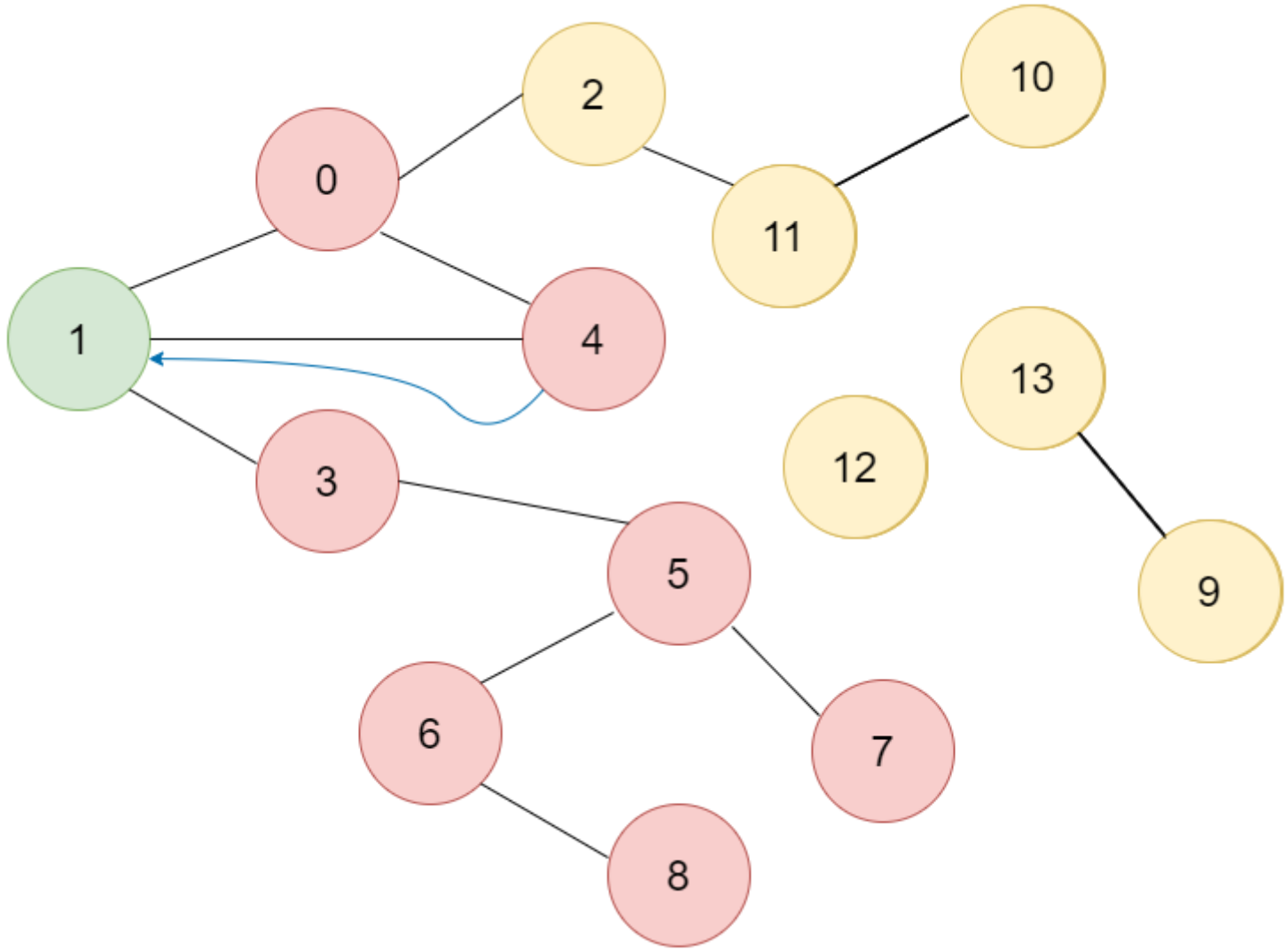

2
10
0
11
1
4
13
12
3
9
5
6
7
8

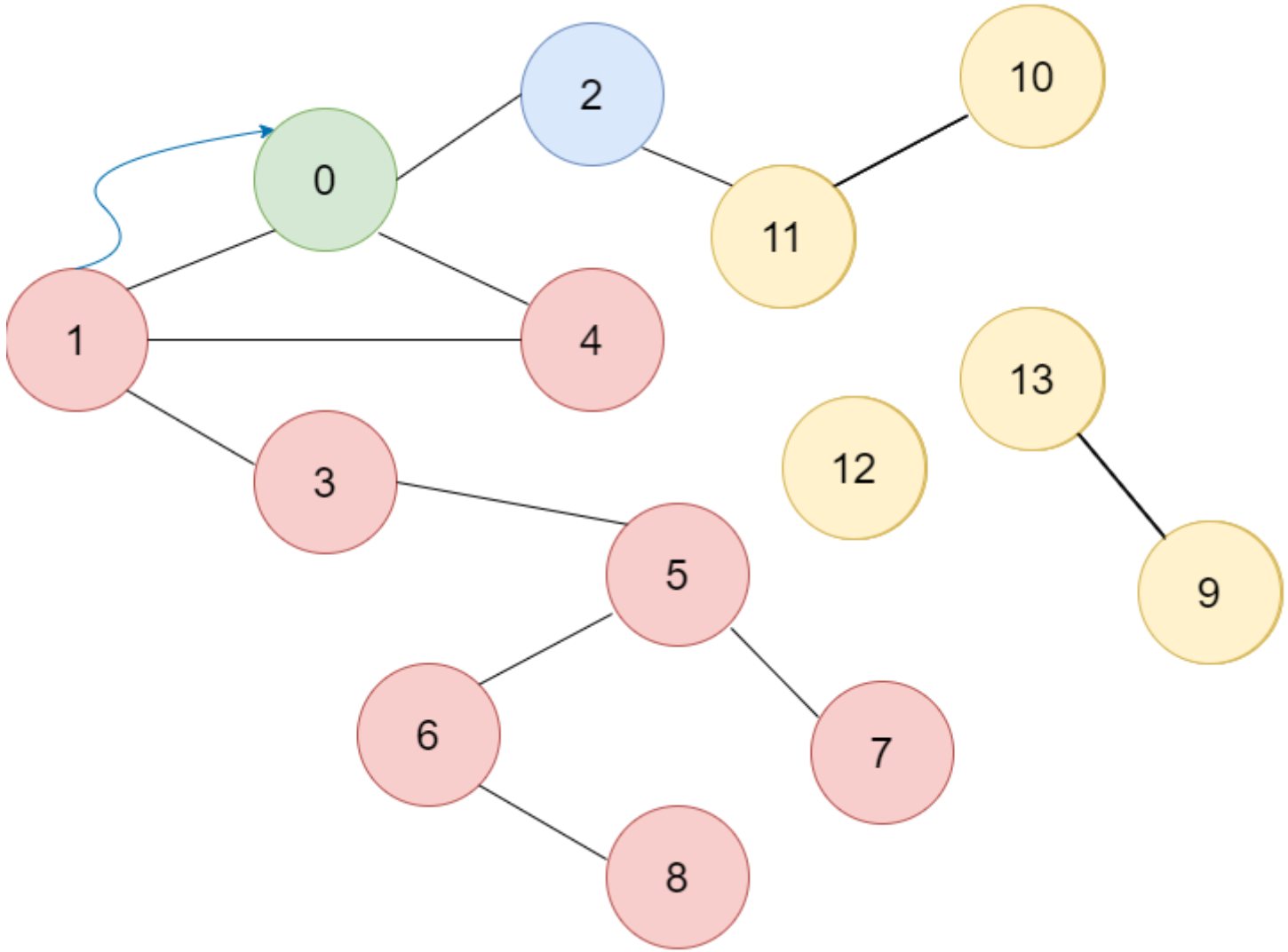

2
10
0
11
1
4
13
12
3
9
5
6
7
8

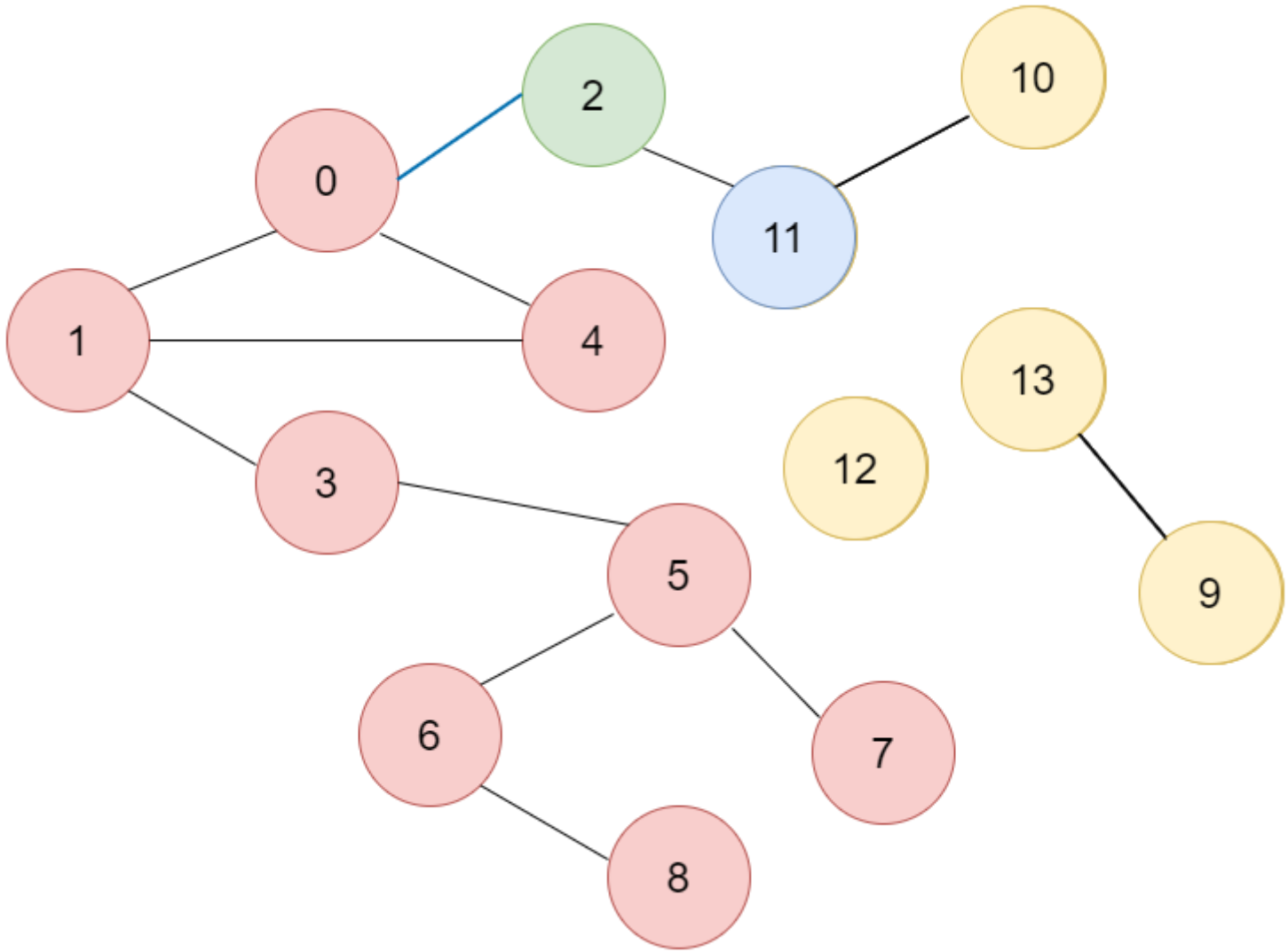

2
10
0
11
1
4
13
12
3
5
9
6
7
8

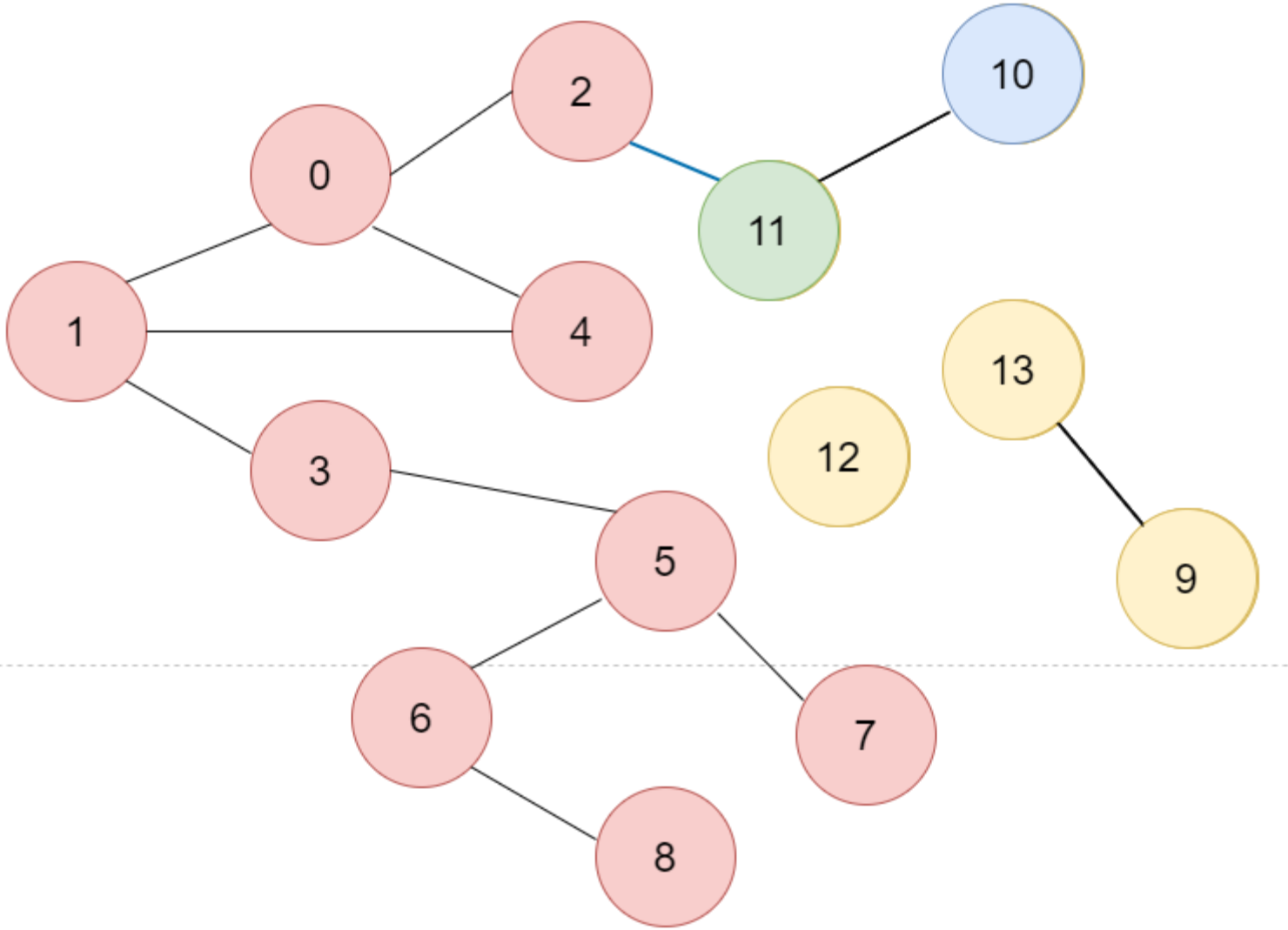

2
10
0
11
1
4
13
12
3
9
5
6
7
8

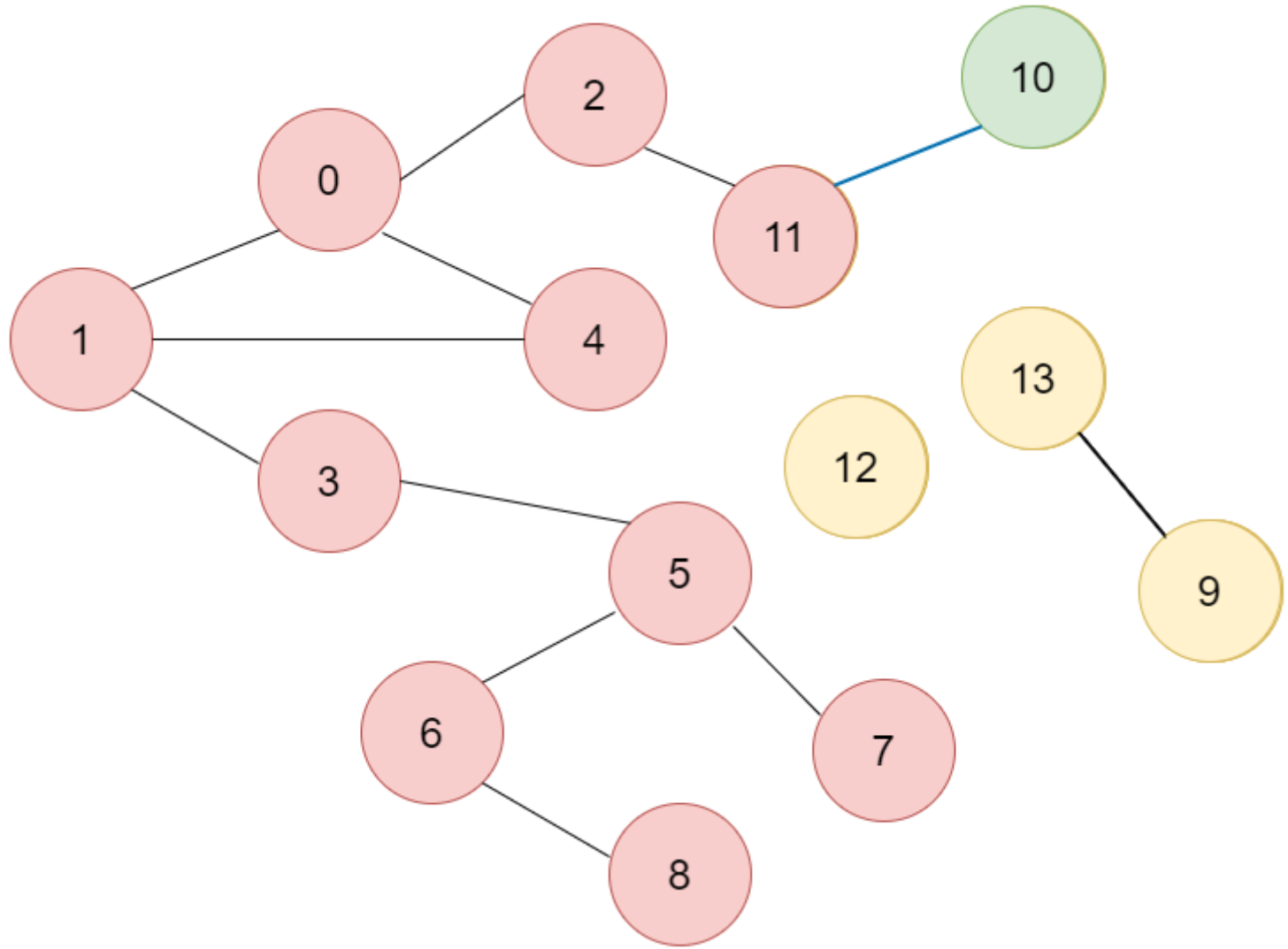

2
10
0
11
1
4
13
12
3
9
5
6
7
8

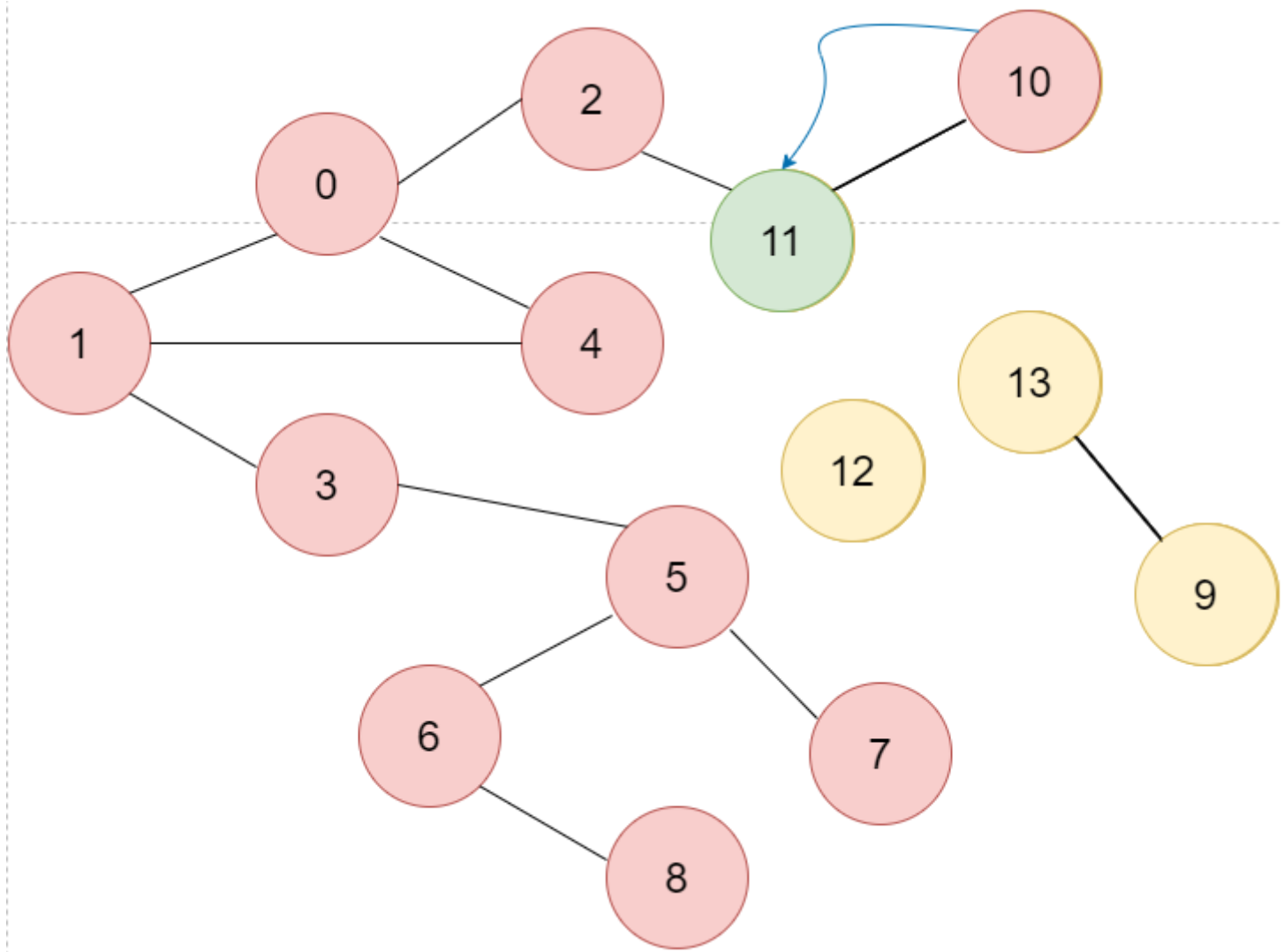

2
10
0
11
1
4
13
12
3
9
5
6
7
8

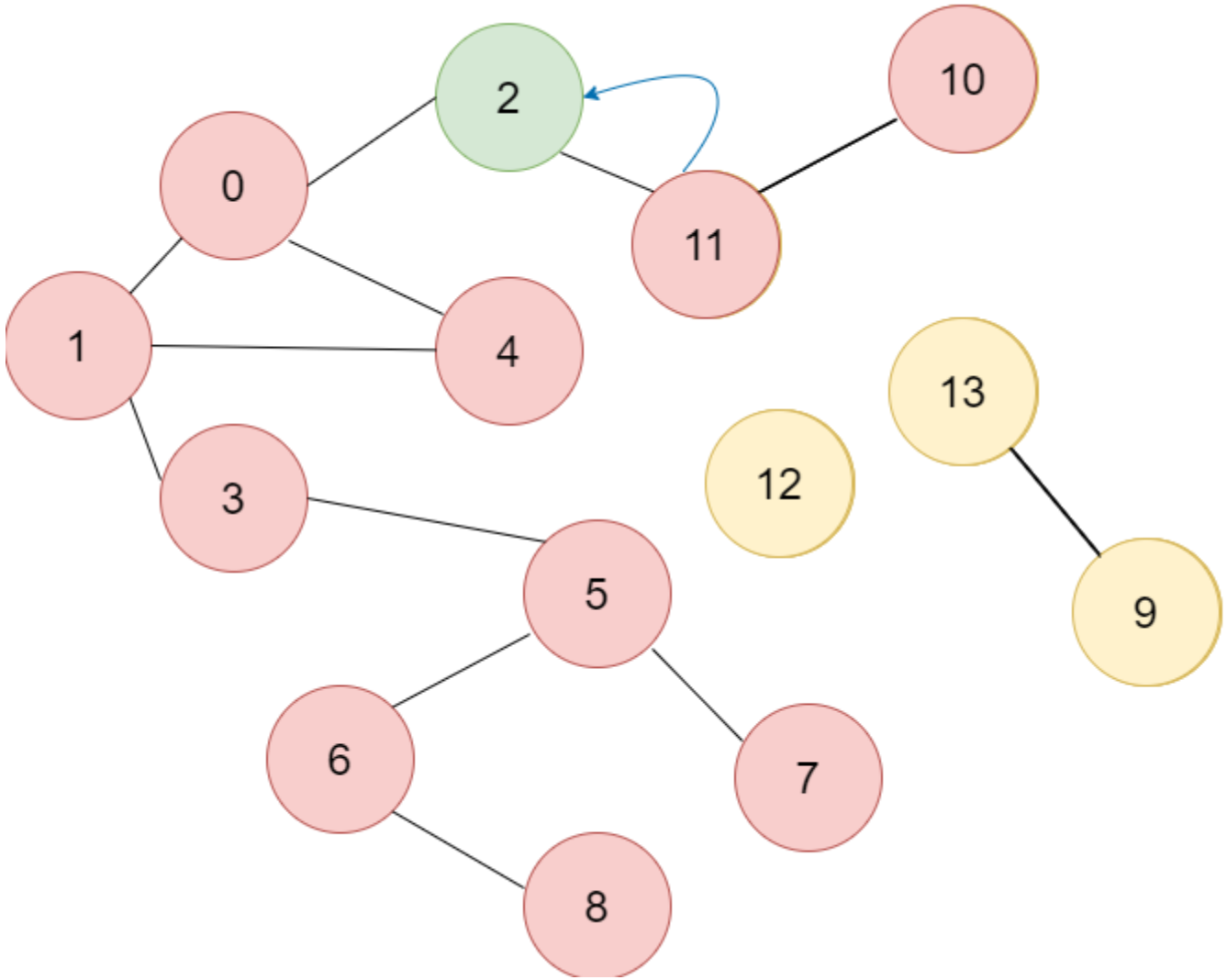

2
10
0
11
1
4
13
12
3
9
5
6
7
8

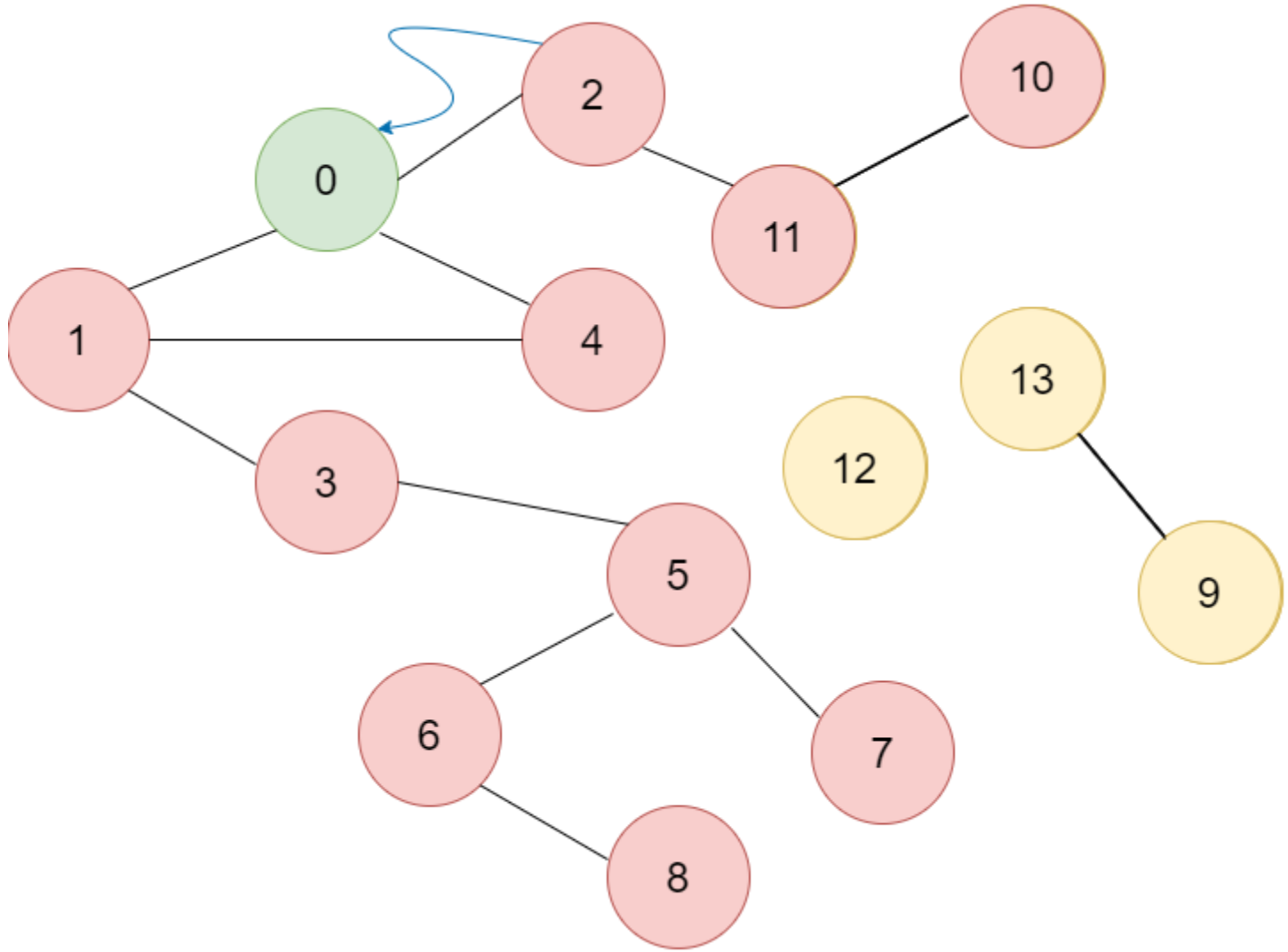

2
10
0
11
1
4
13
12
3
9
5
6
7
8

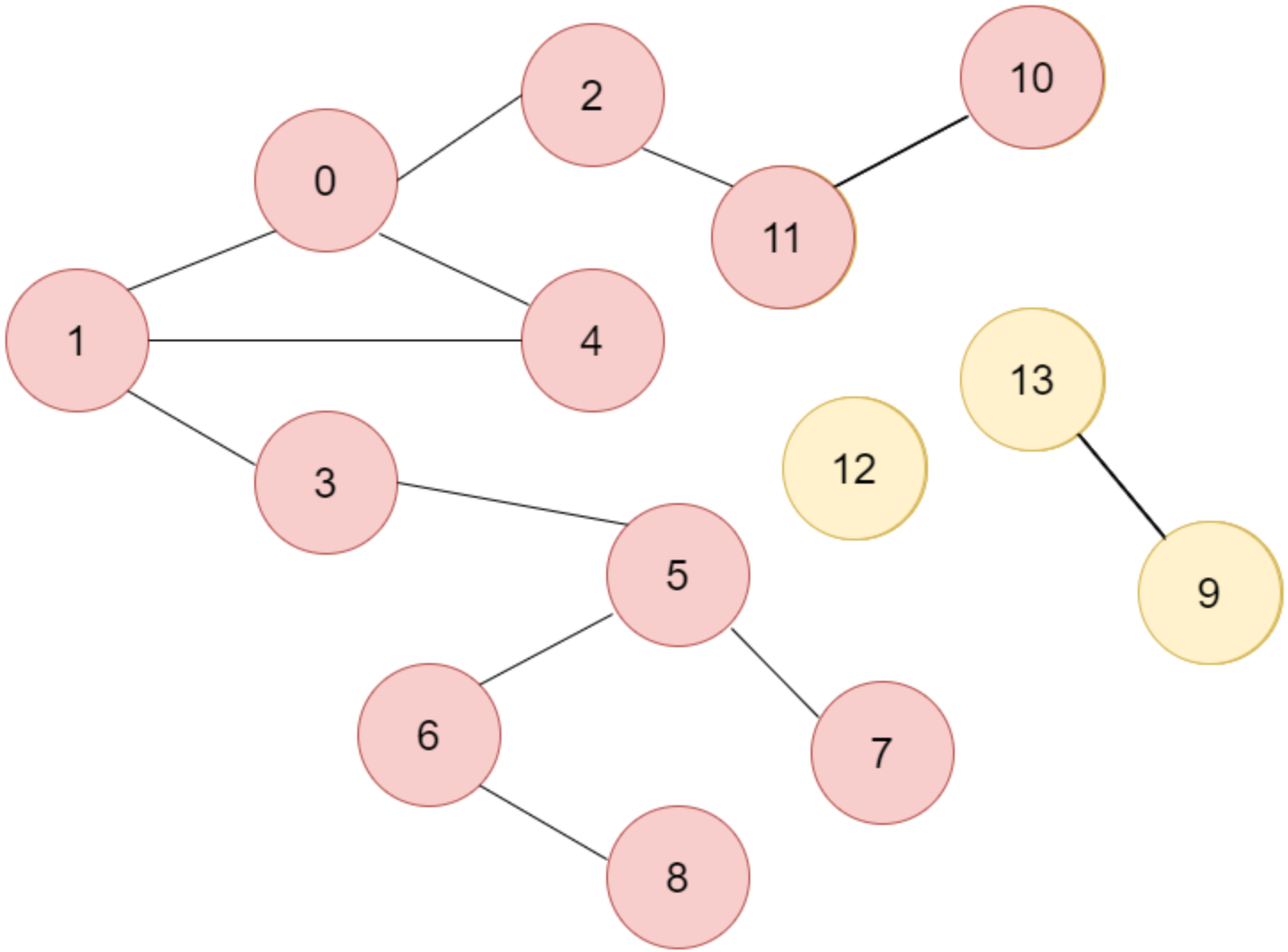

2
10
0
11
1
4
13
12
3
9
5
6
7
8


2
10
0
11
1
4
13
12
3
9
5
6
7
8


2
10
0
11
1
4
13
12
3
9
5
6
7
8


2
10
0
11
1
4
13
12
3
9
5
6
7
8


2
10
0
11
1
4
13
12
3
9
5
6
7
8


2
10
0
11
1
4
13
12
3
9
5
6
7
8

PSEUDO CODE

DFS (to visit a vertex v)

Mark v as visited.

Recursively visit all unmarked vertices w adjacent to v.

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u,v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

Image source: KT Book

DFS (Recursive Version)

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

```
// all vertices unexplored before outer call
mark s as explored
for each edge (s,v) in s's adjacency list do
    if v is unexplored then
        DFS (G,v)
```

Image source: T. Roughgarden

Complexity: $O\ (V + E\ )$

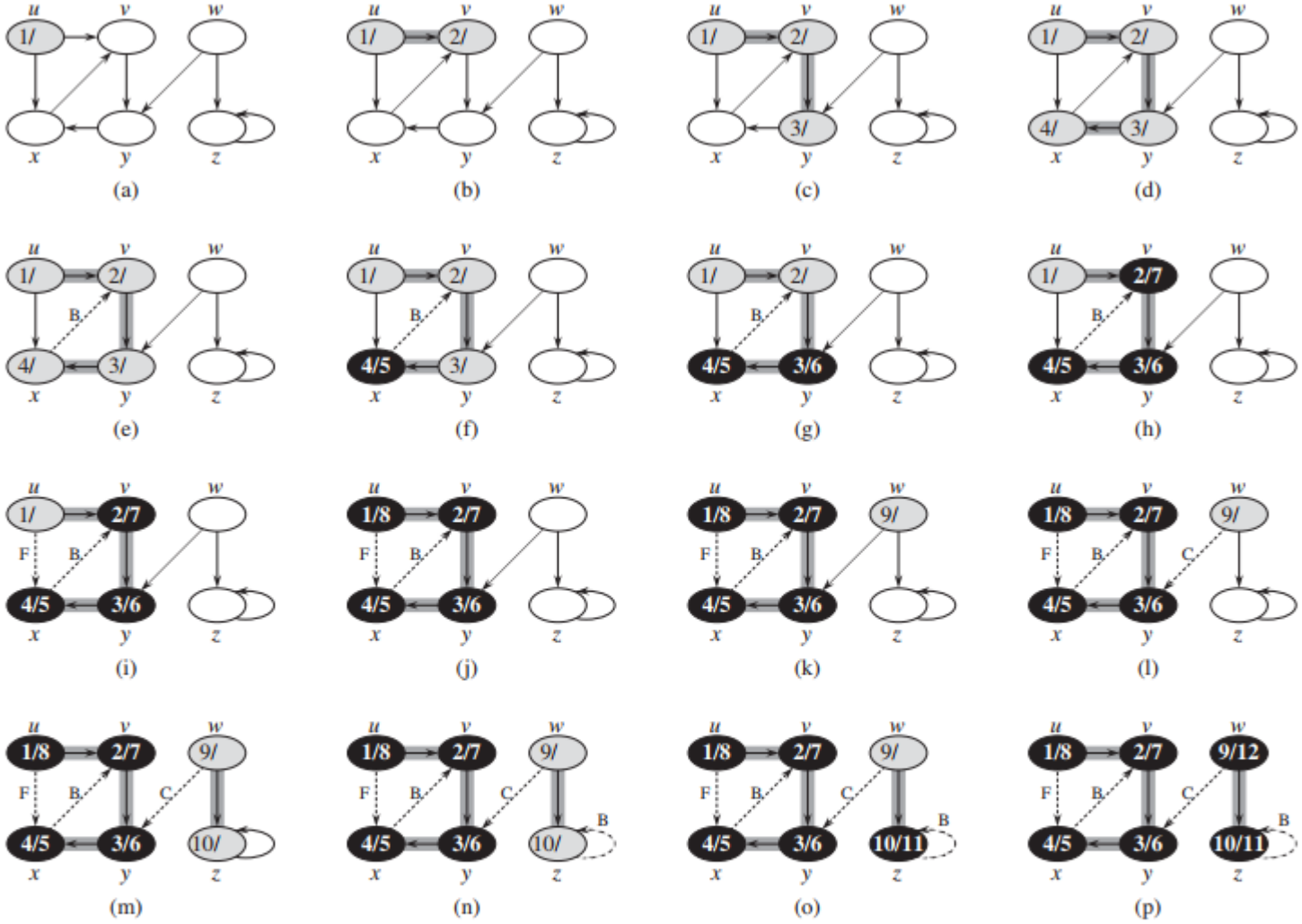


Figure 22.4 The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Timestamps within vertices indicate discovery time/finishing times.

Image Source: CLRS

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1            // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]      // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK            // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

NEXT TOPIC?

Depth First Search (DFS) with Stack