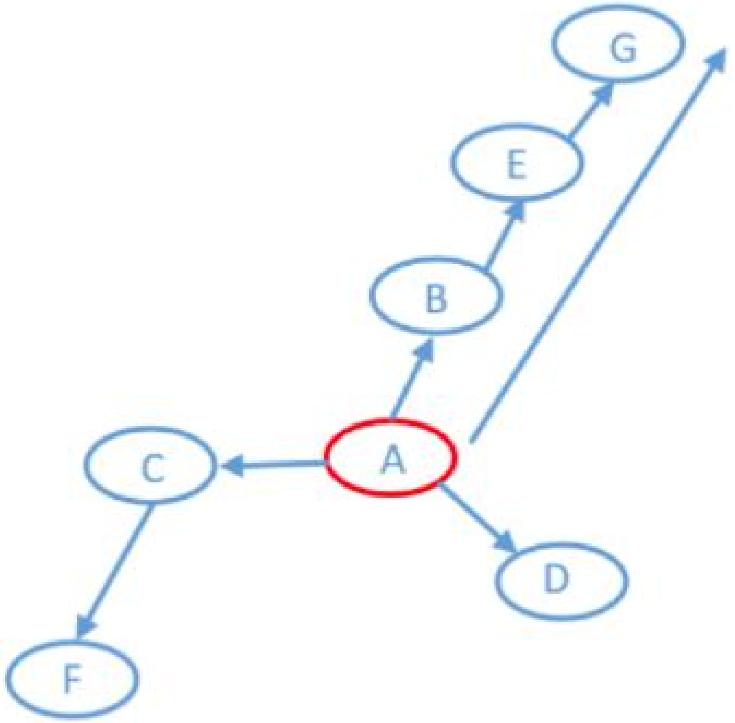
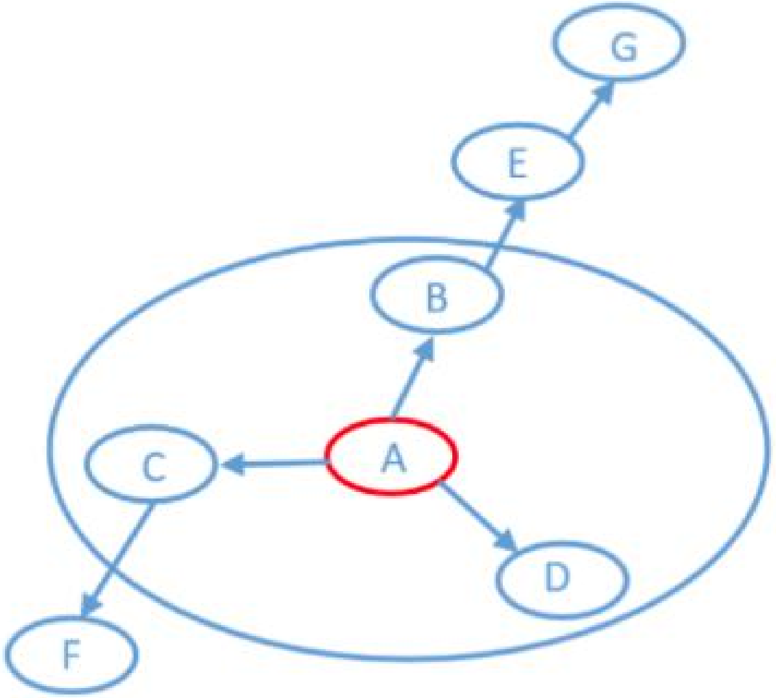
CSC3100 tutorial 9

Yueyao Yu

DFS and BFS



DFS



BFS

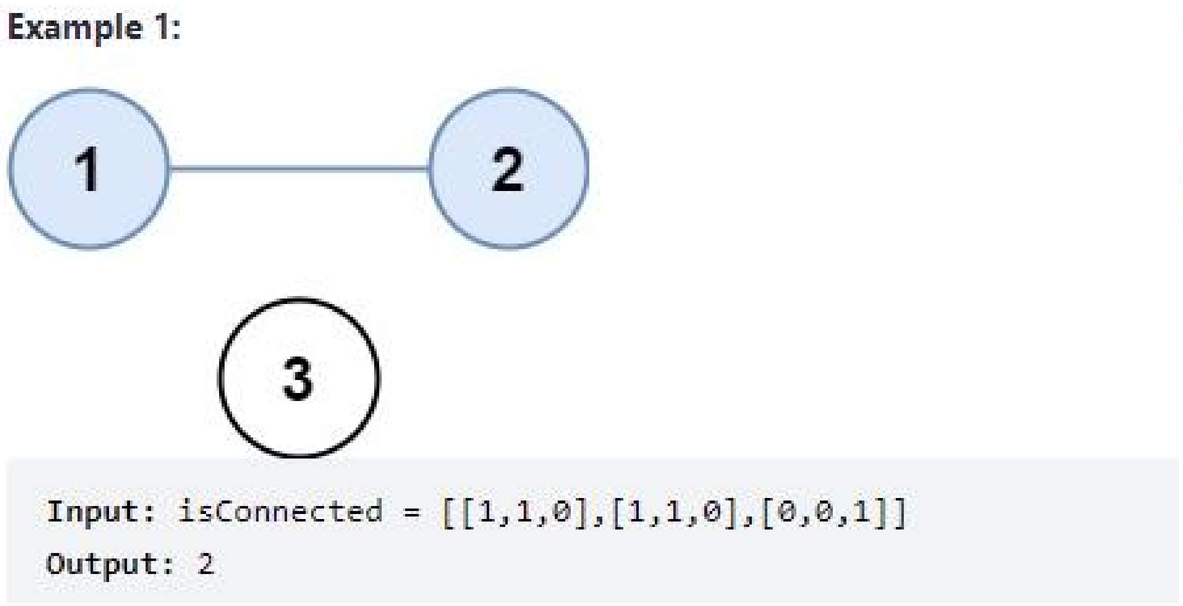
DFS: starts with the initial node, and then goes to deeper and deeper until we find the goal node or the node which has no children.

BFS: starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes.

Problem: Number of Provinces

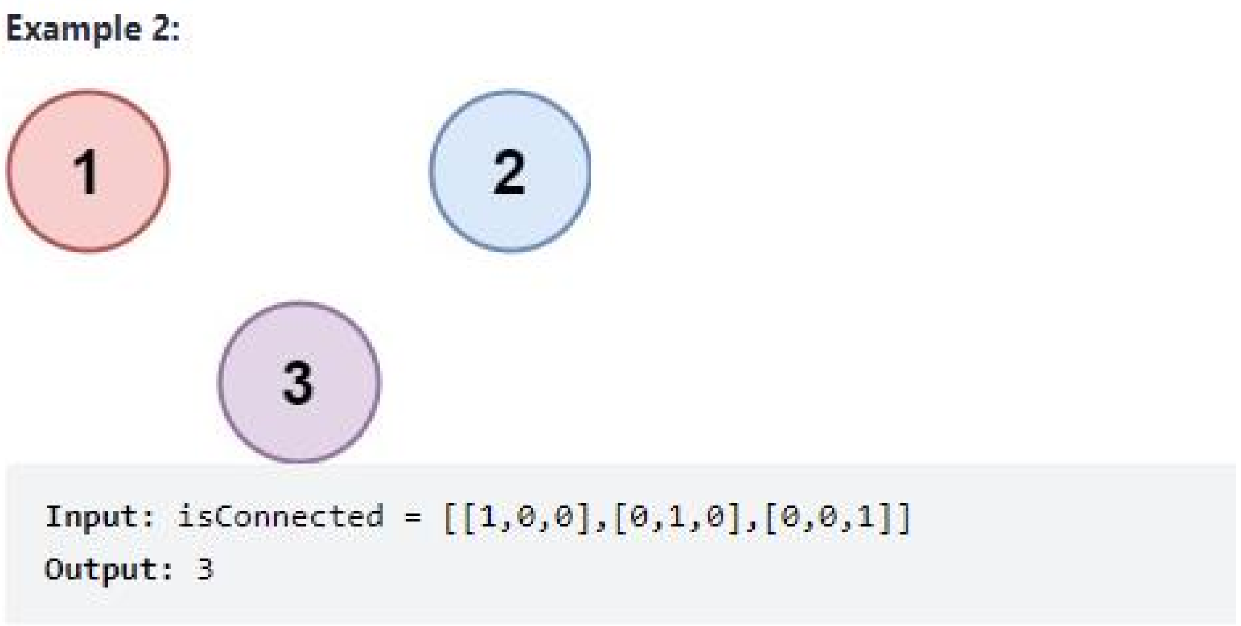
- There are ***n*** cities. Some of them are connected, while some are not. If city ***a*** is connected directly with city ***b***, and city ***b*** is connected directly with city ***c***, then city ***a*** is connected indirectly with city ***c***.
- A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.
- You are given an ***n* x *n*** matrix **isConnected** where **isConnected**[i][j] = 1 if the i-th city and the j-th city are directly connected, and **isConnected**[i][j] = 0 otherwise.
- Return **the total number of provinces**.

Example :



Example 1:

```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```



Example 2:

```
Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3
```

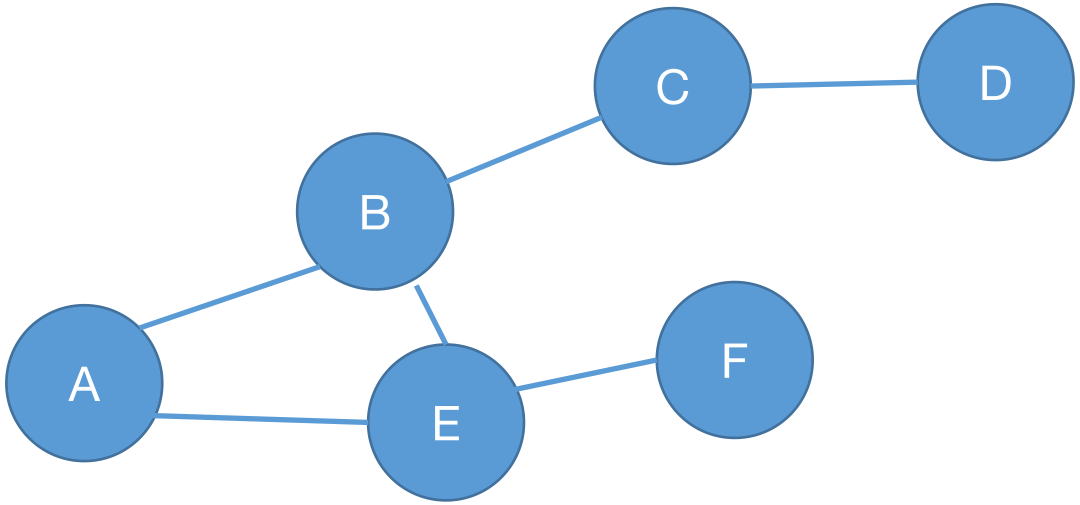
Analysis:

- ***n*** cities and the connection relationship between them can be regarded as a graph. The city is the node in the graph, and the connection relationship is the edge in the graph.
- The given matrix ***isconnected*** is the adjacency matrix of the graph, and the province is the connected component in the graph.

Analysis: DFS

- Traverse all cities. For each city, if the city A has not been visited, start the DFS from A,
- Get cities directly connected to A through the matrix isconnected, and these cities belong to the same Province (connected component),
- and then continue the DFS for these cities, until all cities of the same connected component are visited, a province can be obtained.
- After traversing all cities, you can get the total number of provinces.

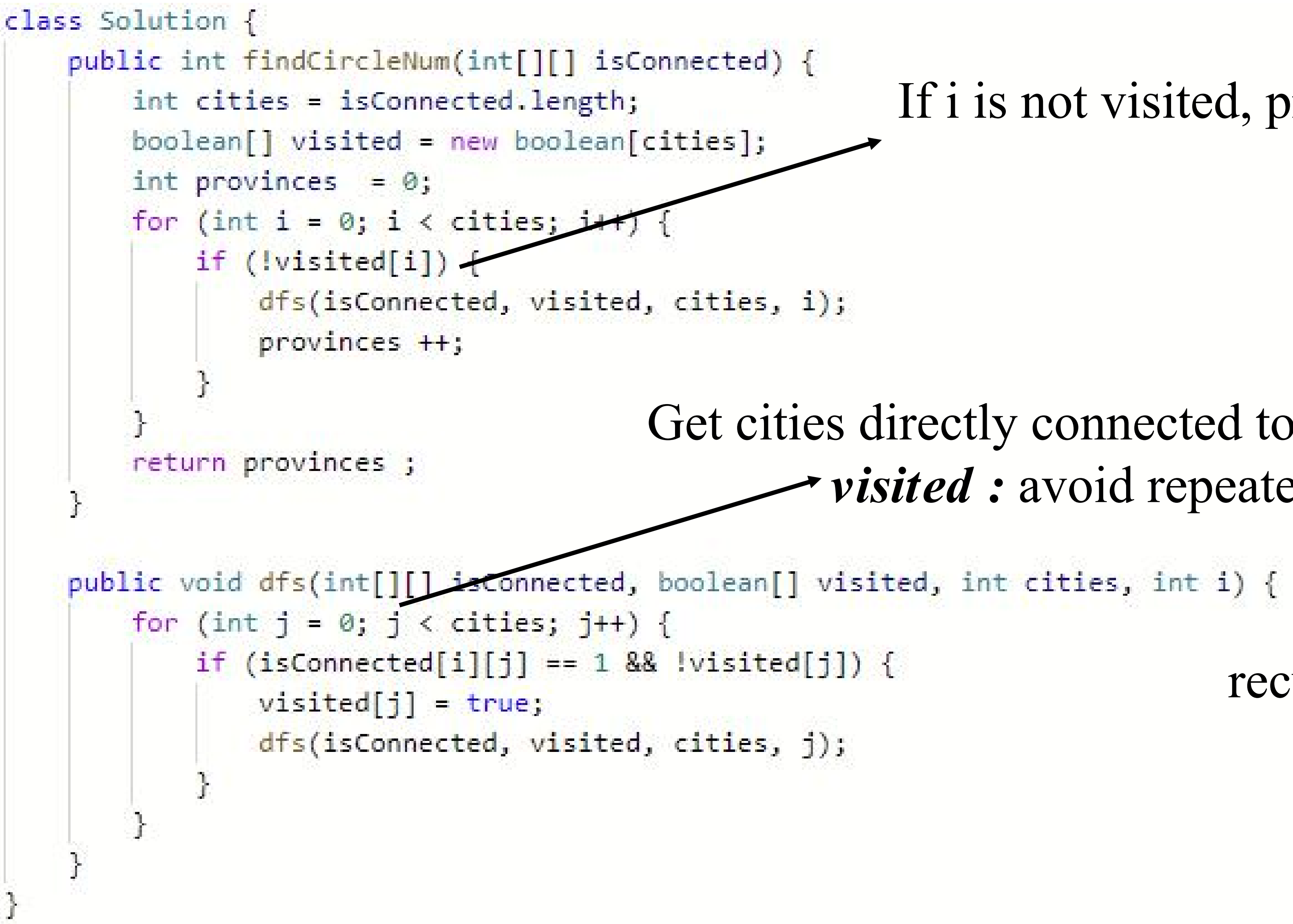
Analysis: DFS



Province example

- 1 A
- 2 visit A to B
- 3 visit B to C
- 4 visit C to D
- 5 visit B to E
- 6 visit E to F

Code : DFS



```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int cities = isConnected.length;
        boolean[] visited = new boolean[cities];
        int provinces  = 0;
        for (int i = 0; i < cities; i++) {
            if (!visited[i]) {
                dfs(isConnected, visited, cities, i);
                provinces ++;
            }
        }
        return provinces ;
    }

    public void dfs(int[][] isConnected, boolean[] visited, int cities, int i) {
        for (int j = 0; j < cities; j++) {
            if (isConnected[i][j] == 1 && !visited[j]) {
                visited[j] = true;
                dfs(isConnected, visited, cities, j);
            }
        }
    }
}
```

If i is not visited, province+1

Get cities directly connected to the city i**.**
visited : avoid repeated visit

recursive algorithm

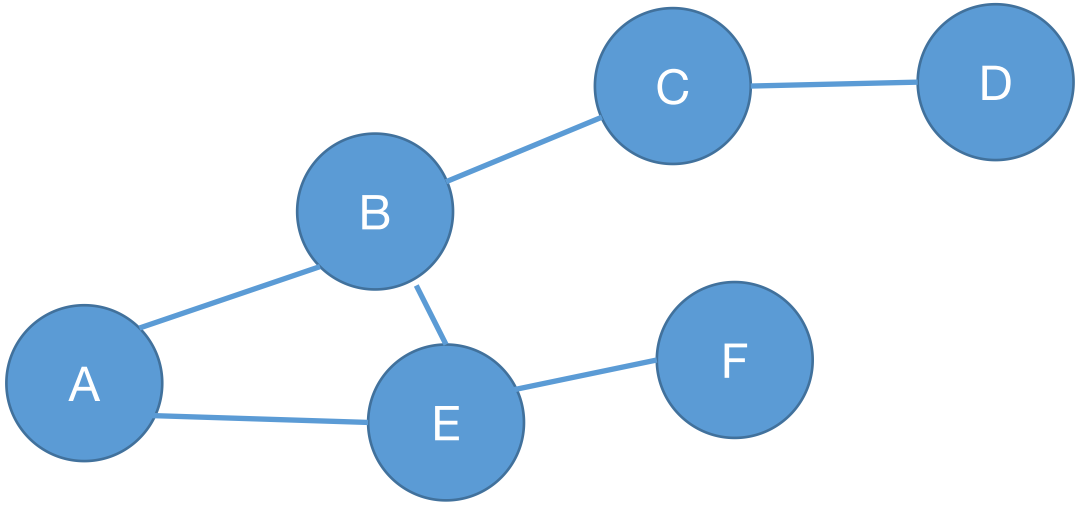
Complexity

- Time complexity: $O(n^2)$
- Space complexity: $O(n)$

Solution: BFS

- The total number of provinces can also be obtained by BFS.
- For each city, if the city has not been visited, start BFS from the city until all cities in the same connected component are visited, to get a province.

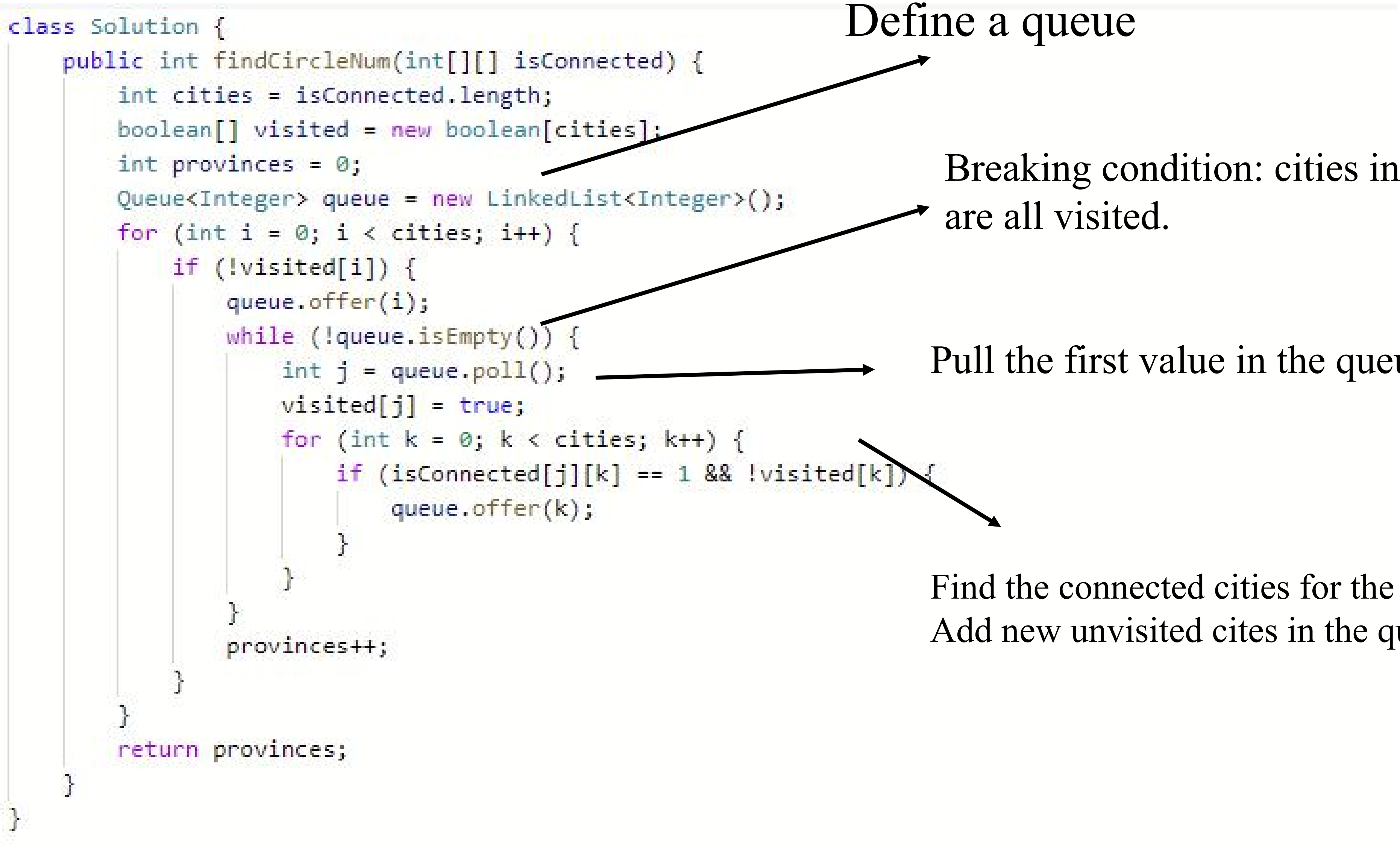
Analysis: BFS



Province example

• 1 A	queue: A
• 2 visit A to B	queue: B
• 3 visit A to E	queue: BE
• 4 visit B to C	queue: EC
• 5 visit E to F	queue: CF
• 6 visit C to D	queue: FD
• Remove F	queue: D
• Remove D	queue: none

Code : BFS



```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int cities = isConnected.length;
        boolean[] visited = new boolean[cities];
        int provinces = 0;
        Queue<Integer> queue = new LinkedList<Integer>();
        for (int i = 0; i < cities; i++) {
            if (!visited[i]) {
                queue.offer(i);
                while (!queue.isEmpty()) {
                    int j = queue.poll();
                    visited[j] = true;
                    for (int k = 0; k < cities; k++) {
                        if (isConnected[j][k] == 1 && !visited[k]) {
                            queue.offer(k);
                        }
                    }
                }
                provinces++;
            }
        }
        return provinces;
    }
}
```

Define a queue

Breaking condition: cities in this province are all visited.

Pull the first value in the queue.

Find the connected cities for the first city ***j*** in the queue. Add new unvisited cites in the queue.

Complexity

- Time complexity: $O(n^2)$
- Space complexity: $O(n)$