3기 / 1주차_BFS, DFS

🌟 BFS, DFS

1-1. 그래프

DFS/BFS에 대해 이해하려면 먼저 그래프의 기본 구조를 알아야한다.

그래프는 노드(Node)와 간선(Edge)으로 표현되며 이 때 노드를 정점(Vertex)라고 말한다.

그래프 탐색이란 하나의 노드를 시작으로 다수의 노드를 방문하는 것을 말하며 두 노드가 간선으로 연결되어 있을 때 '인접하다'고 표현한다.

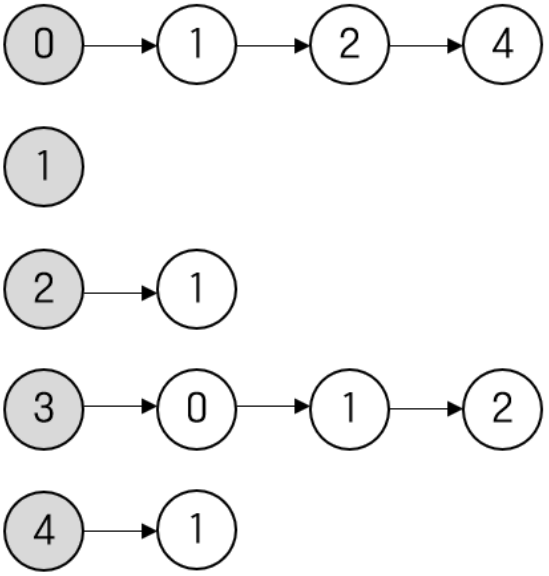
그래프는 크게 2가지 방식으로 표현할 수 있다.

- 인접 행렬(Adjacency Matrix)
- 인접 리스트(Adjacency List)

먼저 인접 행렬 방식은 2차원 배열에 각 노드가 연결된 형태를 기록하는 방식이다.

	0	1	2	3	4
0	0	1	1	0	1
1	0	0	0	0	0
2	0	1	1	0	0
3	1	1	1	1	0
4	0	1	0	0	0

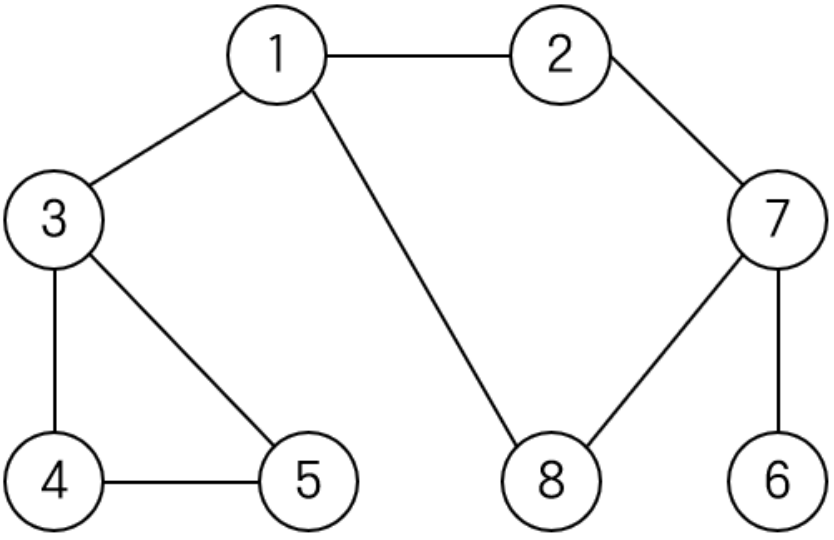
인접 리스트 방식은 모든 노드에 연결된 노드에 대한 정보를 차례대로 연결해서 저장한다.



인접 행렬 방식은 노드 개수가 많을수록 메모리가 불필요하게 낭비되는 반면 인접 리스트 방식은 메모리를 효율적으로 사용한다. 하지만 이와 같은 속성 때문에 인접 리스트 방식은 두 노드가 연결되어 있는지에 대한 정보를 얻는 속도가 느리다.

1-2. DFS

DFS는 Depth First Search, 깊이 우선 탐색이라고 부르며 그래프에서 깊은 부분을 우선적으로 탐색하는 알고리즘이다.



이 그래프에서 노드 1부터 시작해서 DFS를 이용한 탐색을 진행해보자.

```
방문 순서 : 1 → 2 → 7 → 6 → 8 → 3 → 4 → 5
```

1-3. BFS

BFS는 Breadth First Search, 너비 우선 탐색이라고 부르며 가까운 노드부터 탐색하는 알고리즘이다.

```
방문 순서 : 1 → 2 → 3 → 8 → 7 → 4 → 5 → 6
```

1-4. 또간복잡도

- DFS (인접 행렬로 구현) : O(n^2)
  - DFS 하나당 N번의 루프를 돌기 때문에 O(n)이지만 N개의 정점을 모두 방문하기 때문
- DFS (인접 리스트로 구현) : O(n+e)
  - 각 정점에 연결되어 있는 간선의 개수만큼 탐색을 한다. 즉, 모든 정점과 모든 간선을 한번씩 다 확인한다고 생각
- BFS (인접 행렬로 구현) : O(n^2)
  - 정점 한 개당 N번의 for 루프를 돌기 때문에 O(n)이지만 N개의 정점을 모두 방문하기 때문
- BFS (인접 리스트로 구현) : O(n+e)

1. 프로그래머스 43162_네트워크

🔑 풀이

BFS의 가장 기초적인 풀이라고 생각한다.

정점을 큐에 담고 하나씩 빼면서 연결되어 있는 점이 있는지 확인한다.

만약 없다면 새로운 탐색을 시작하는 동시에 네트워크 개수(정답)에 1을 더한다.

💻 코드

```
import java.util.LinkedList;
import java.util.Queue;

public class Programmers_43162 {
    private static int n; // 컴퓨터의 개수
    private static int[][] computers;
    private static boolean[] isVisited;
    private static int answer = 0;

    public static void main(String[] args) {
        isVisited = new boolean[computers.length];

        for (int i = 0; i < computers.length; i++) {
            if (!isVisited[i]) {
                bfs(i);
                answer++;
            }
        }
    }

    static void bfs(int start) {
        isVisited[start] = true;
        Queue<Integer> qu = new LinkedList<>();
        qu.add(start);

        while (!qu.isEmpty()) {
            start = qu.poll();
            for (int i = 0; i < computers.length; i++) {
                if (!isVisited[i] && computers[start][i] == 1
                    isVisited[i] = true;
                    qu.add(i);
                }
            }
        }
    }
}
```

2. 프로그래머스 1844_게임 맵 최단거리

🔑 풀이

{0, 0} 부터 탐색을 시작한다.

dist 배열에 {0, 0}에서의 거리를 입력하고 만약 dist 배열의 최종점 (도착지) 값이 0이라면 상대방 진영에 도달할 수 없다는 얘기가 된다.

따라서 -1을 출력하고, 아니라면 해당 거리를 출력한다.

💻 코드

816 ms, 298108 KB

```
import java.util.LinkedList;
import java.util.Queue;

public class Programmers_1844 {
    private static int[][] maps;
    private static int answer = 0;
    private static int[][] dist;

    public static void main(String[] args) {
        dist = new int[maps.length][maps[0].length];
        bfs(new int[]{0, 0});

        System.out.println(answer);
    }

    static void bfs(int[] start) {
        Queue<int[]> qu = new LinkedList<>();
        qu.add(start);
        int[] dx = {0, -1, 0, 1};
        int[] dy = {1, 0, -1, 0};

        while(!qu.isEmpty()) {
            int[] current = qu.poll();
            int x = current[0];
            int y = current[1];
```

```
            for (int i = 0; i < 4; i++) {
                int nextX = x + dx[i];
                int nextY = y + dy[i];
                if (nextX >= 0 && nextY >= 0 && nextX < maps.
                    if (maps[nextX][nextY] == 1 && dist[nextX
                        dist[nextX][nextY] = dist[x][y] + 1;
                        qu.add(new int[]{nextX, nextY});
                    }
                }
            }
        }

        if (dist[maps.length - 1][maps[0].length - 1] == 0) {
            answer = -1;
        } else {
            answer = dist[maps.length - 1][maps[0].length - 1
        }
    }
}
```

3. 프로그래머스 43163_단어 변환

🔑 풀이

begin과 철자 하나만 다른 경우는 변환이 가능하다.

따라서 철자 하나만 다른 경우 해당 단어부터 또 다른 단어로 변환이 가능한지 찾는다.

💻 코드

```
public class Programmers_43163 {
    private static String[] words;
    private static String target;
    private static String begin;
    private static int answer;
    private static boolean[] isVisited;
    private static final int IMPOSSIBILITY = 100;
```

```
    public static void main(String[] args) {
        answer = 100;
        isVisited = new boolean[words.length];
        dfs(begin, 0);

        if (answer == IMPOSSIBILITY) answer = 0;
        System.out.println(answer);
    }

    static void dfs(String begin, int count) {
        if (begin.equals(target)) {
            answer = Math.min(answer, count);

            return;
        }

        for (int i = 0; i < words.length; i++) {
            int len = 0;
            if (!isVisited[i]) {
                for (int k = 0; k < words[i].length(); k++) {
                    if (begin.charAt(k) == words[i].charAt(k)
                }

                if (len == begin.length() - 1) {
                    isVisited[i] = true;
                    dfs(words[i], count + 1);
                    isVisited[i] = false;
                }
            }
        }
    }
}
```