

2021 여름학기 동국대학교 SW역량강화캠프

8일차. DFS 1

● DFS (깊이 우선 탐색)

- ▶ 재귀함수를 통한 그래프 탐색(인접한 정점, 인접한 상태)
- ▶ 모든 경우의 수를 탐색하는 완전 탐색
- ▶ 중복 방문을 제한하는 경우 방문 여부를 저장하는 배열을 사용한다.

● 그래프 저장 방법 (인접 리스트)

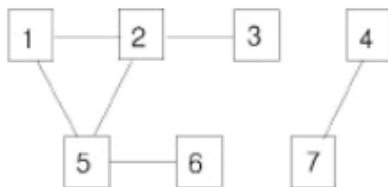
- ▶ 그래프 탐색 문제에서 “x와 연결된 정점 ” 을 빠르게 찾을 수 있는 그래프 저장 형태
- ▶ x번 List에 x와 연결된 정점들을 저장하는 형태

● 바이러스 (877)

문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크 상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



< 그림 1 >

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

- ▶ 재귀함수를 이용한 풀이
- ▶ dfs 함수를 $\text{dfs}(x)$: x 에서 갈 수 있는 모든 정점의 `visited`를 `true`로 바꾸는 함수
- ▶ dfs를 통해서 모든 갈 수 있는 정점들을 탐색

```
public class main {  
    static ArrayList<ArrayList<Integer>> Graphlist;  
    static boolean visited[];  
  
    public static void main(String[] args) throws IOException {  
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
        StringTokenizer st;  
        int N = Integer.parseInt(br.readLine());  
        int M = Integer.parseInt(br.readLine());  
  
        Graphlist = new ArrayList<ArrayList<Integer>>();  
        visited = new boolean[N + 1];  
  
        // Graph 인접리스트 만들기  
        for (int i = 0; i <= N; i++)  
            Graphlist.add(new ArrayList<Integer>());  
        for (int i = 0; i < M; i++) {  
            st = new StringTokenizer(br.readLine());  
            int p = Integer.parseInt(st.nextToken());  
            int q = Integer.parseInt(st.nextToken());  
            Graphlist.get(p).add(q);  
            Graphlist.get(q).add(p);  
        }  
    }  
}
```

```
dfs(1);

int ans = 0;
for (int i = 2; i <= N; i++) {
    if (visited[i]) {
        ans++;
    }
}
System.out.println(ans);
}

public static void dfs(int x) {
    visited[x] = true;
    for (int y : Graphlist.get(x)) {
        if (visited[y] == false) {
            dfs(y);
        }
    }
}
```

● DFS를 이용한 Flood Fill

- ▶ 배열에서 인접한 칸끼리 이동 가능할 때, 연결된 그룹을 판별하는 형태
- ▶ $dx[4] = \{-1, 0, 1, 0\}$, $dy[4] = \{0, 1, 0, -1\}$
- ▶ $newx = x + dx[i]$, $newy = y + dy[i]$ 로 선언하면 (x, y) 와 $(newx, newy)$ 는 인접

● 단지번호 붙이기(4284)

문제

<그림 1>과 같이 정사각형 모양의 지도가 있다. 1은 집이 있는 곳을, 0은 집이 없는 곳을 나타낸다. 철수는 이 지도를 가지고 연결된 집의 모임인 단지를 정의하고, 단지에 번호를 붙이려 한다. 여기서 연결되었다는 것은 어떤 집이 좌우, 혹은 아래위로 다른 집이 있는 경우를 말한다. 대각선상에 집이 있는 경우는 연결된 것이 아니다. <그림 2>는 <그림 1>을 단지별로 번호를 붙인 것이다. 지도를 입력하여 단지수를 출력하고, 각 단지에 속하는 집의 수를 오름차순으로 정렬하여 출력하는 프로그램을 작성하시오.

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

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

<그림 2>

- ▶ Flood Fill을 이용하자
- ▶ 배열을 전체탐색하면서, dfs에서 방문하지 않은 1이 있다면, 해당 점을 기준으로 dfs를 하여 연결된 1들을 모두 검사한다.
- ▶ dfs에서는 연결된 1의 총 개수를 count할 수 있어야 한다.
- ▶ 전역변수 cnt를 visited 배열을 갱신할 때마다 ++ 하여, 몇 개의 1을 방문하였는지 count

```
for (int i = 1; i <= N; i++) {  
    for (int j = 1; j <= N; j++) {  
        if (arr[i][j] == 1 && visited[i][j] == false) {  
            cnt = 0;  
            dfs(i, j);  
            ans.add(cnt);  
        }  
    }  
}
```

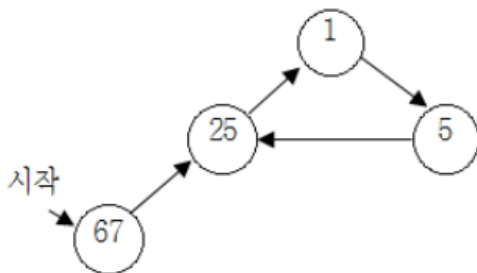
```
public static void dfs(int x, int y) {  
    visited[x][y] = true;  
    cnt++;  
    for (int i = 0; i < 4; i++) {  
        int newx = x + dx[i];  
        int newy = y + dy[i];  
        if (newx < 1 || newx > N || newy < 1 || newy > N)  
            continue;  
        if (visited[newx][newy] == false && arr[newx][newy] == 1)  
            dfs(newx, newy);  
    }  
}
```

● 사이클 (783)

문제

두 자연수 N 과 P 를 가지고 다음 과정을 거쳐서 나오는 숫자들을 차례대로 출력해보자. 처음 출력하는 숫자는 N 이고, 두 번째 이후 출력하는 숫자들은 N 을 곱하고 P 로 나눈 나머지를 구하는 과정을 반복하여 구한다. 즉, 먼저 N 에 N 을 곱하고, 이 수를 P 로 나눈 나머지를 두 번째에 출력한다. 다음에는 이 나머지에 N 을 곱하고 P 로 나눈 나머지를 출력한다. 다음에는 이 나머지에 N 을 곱한 후 P 로 나눈 나머지를 출력한다. 이 과정을 계속 반복해보면 출력되는 숫자들에는 반복되는 부분이 있다.

예를 들어서, $N=67$, $P=31$ 인 경우를 생각해보자. 처음 출력되는 숫자는 67이고, 두 번째로 출력되는 숫자는 $67*67 = 4489$ 를 31로 나눈 나머지 25이다. 다음에는 $25*67 = 1675$ 를 31로 나눈 나머지 1, 다음에는 $1*67 = 67$ 을 31로 나눈 나머지 5가 차례대로 출력된다. 다음에는 $5*67 = 335$ 를 31로 나눈 나머지 25가 출력되는데, 이 수는 이미 이전에 출력된 수이다. 이 과정을 그림으로 보이면 다음과 같다.



즉 이 과정을 반복하면, 처음 67을 제외하면 3개의 숫자 25, 1, 5가 계속 무한히 반복되게 된다. 또 다른 예로, $N=9$, $P=3$ 을 가지고 시작하면, $9*9 = 81$ 이고 3으로 나눈 나머지는 0이며, $0*3 = 0$ 이고 3으로 나눈 나머지도 0이기 때문에 처음 9를 제외하면 0이 무한히 반복되게 된다.

N 과 P 를 입력받아 위와 같이 정의된 연산을 수행하였을 때, 반복되는 부분에 포함된 서로 다른 숫자의 개수를 구하는 프로그램을 작성하시오.

▶ DFS를 이용한 풀이

인접한 상태로의 전이를 DFS를 이용해 해결해 보자

$$N = 67, P = 31$$

$$67 \rightarrow 25 \rightarrow 1 \rightarrow 5 \rightarrow 25$$

$$(67, 1) \rightarrow (25, 2) \rightarrow (1, 3) \rightarrow (5, 4) \rightarrow (25, 5)$$

(25,2)와 (25,5) 의 두 상태에서부터 답이 $(5-2) = 3$ 임을 알 수 있다.

```
public static void dfs(int x, int round){  
    if(visited[x] == 0) {  
        visited[x] = round;  
        dfs((x*N)%P, round+1);  
    }  
    else {  
        ans = round - visited[x];  
    }  
}
```

● 둘레 (4283)

| 문제

농부 존은 N개의 건초더미를 농장에 배치했다. 농장은 100x100 형태의 정사각형으로 생각할 수 있고, 각 건초더미는 하나의 1x1 정사각형을 차지한다. (물론 건초더미가 같은 영역을 차지하지는 않는다.)

존은 모든 건초 더미가 모두 연결되었다는 사실을 깨달았다! 즉, 하나의 건초 더미에서 시작하여 변으로 연결된 건초 더미들을 통해 임의의 다른 건초 더미로 이동할 수 있다. 이 건초 더미들은 구멍을 포함할 수도 있다. (건초로 감싸진 구역을 의미한다.)

존은 건초 더미의 둘레 길이를 알고 싶다. 단, 구멍의 둘레는 세지 않는다. 존에게 둘레의 길이를 알려주자.

- ▶ DFS를 이용한 풀이
- ▶ 둘레의 길이를 알 수 있는 방법
- ▶ 건초더미의 상하좌우의 “외부 빈칸” 의 개수의 총 합
- ▶ 내부 빈칸과 외부 빈칸을 어떻게 구분할 것인가?

- ▶ DFS를 이용한 풀이
- ▶ 배열을 $(1,1) \sim (100,100)$ 에 입력받았다면 $(0,0)$ 은 반드시 외부
- ▶ 외부와 연결된 모든 빈칸은 외부, 외부와 연결되지 않은 빈칸은 내부

```
public static void dfs(int x, int y) {  
    visited[x][y] = true;  
    for (int i = 0; i < 4; i++) {  
        int newx = x + dx[i];  
        int newy = y + dy[i];  
        if (newx < 0 || newx > 101 || newy < 0 || newy > 101)  
            continue;  
        if (visited[newx][newy] == false && arr[newx][newy] == 0)  
            dfs(newx, newy);  
    }  
}
```

```
dfs(0,0);
int ans = 0;
for (int i = 1; i <= 100; i++) {
    for (int j = 1; j <= 100; j++) {
        if (arr[i][j] == 1) {
            for(int k = 0; k < 4; k++)
            {
                int newi = i + dx[k];
                int newj = j + dy[k];
                if(arr[newi][newj] == 0 && visited[newi][newj] == true) ans++;
            }
        }
    }
}
System.out.println(ans);
```