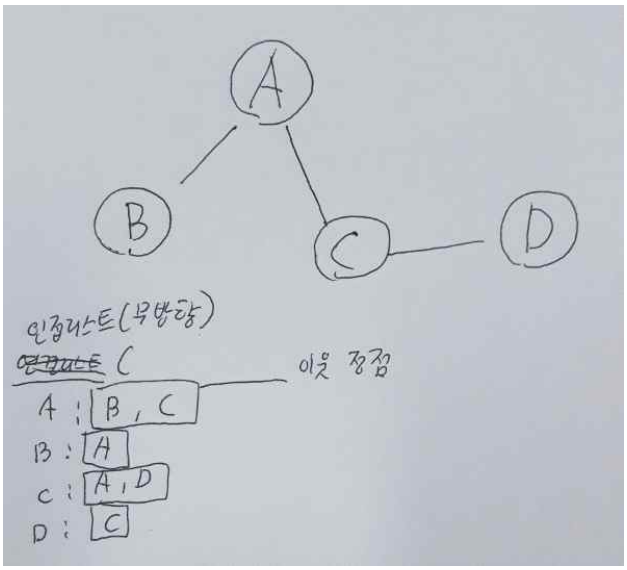
알고리즘-6주차-Graph-Traversal
이름 : 유형곤
학번 : 201902722
사용 언어 : Java

<점수표>

RANK	TEAM	SCORE	1-NEIGHBOR [2 POINTS]	2-BFS [2 POINTS]	3-DFS [2 POINTS]	4-P2P [4 POINTS]
1	201902722	10 110	12 2 tries	23 1 try	20 1 try	35 1 try

문제 1 : Neighbor
문제 : 그래프의 각 정점과 정점을 연결한 후, 특정 정점의 이웃을 출력하는 문제
해결 방법 :



그림처럼 인접 리스트로 구현할 경우, A와 연결된 리스트가 A의 이웃이 됩니다. 그런데, 문제에서는 A-B B-A와 같이 중복된 테스트케이스가 존재하므로 Set을 이용하여 구현했습니다. 결과적으로 정점의 이름과 Set을 연결하는 Map을 활용하여 구현하였습니다.

```
graph = new HashMap<String, Set<String>>();
```

<그래프 선언>

```
for(int i = 0; i < edges; i++) {
    st = new StringTokenizer(br.readLine());
    String s = st.nextToken();
    String e = st.nextToken();
    graph.get(s).add(e);
    graph.get(e).add(s);
}
```

<정점의 각 정점 연결하기>

```
String name = br.readLine();
System.out.println(graph.get(name).size());
br.close();
```

<그래프의 이름과 연결된 Set의 길이 구하기>

<실행 결과>

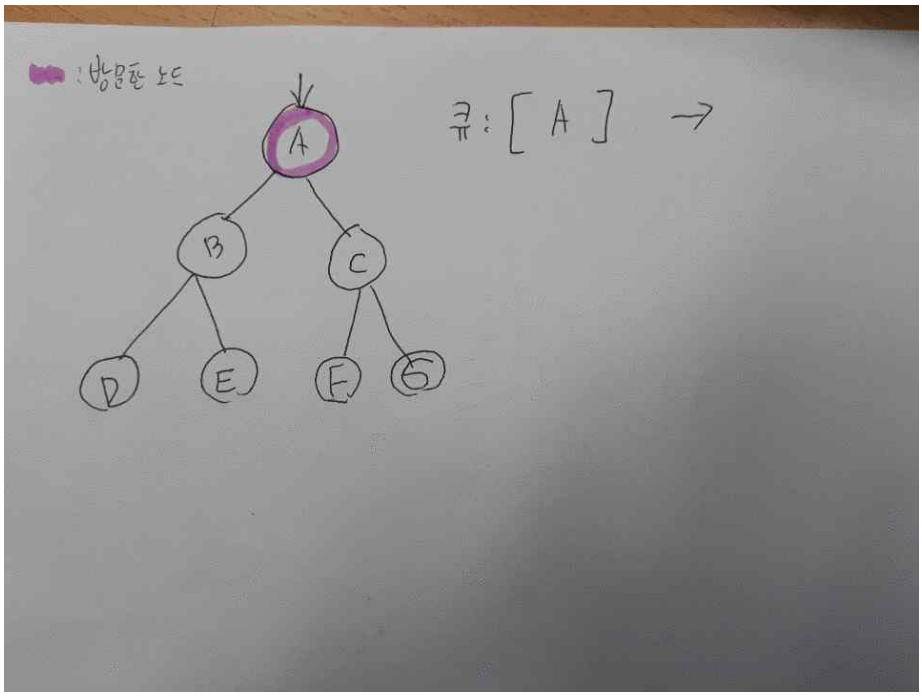
시간복잡도 : O(Unknown) * O(K * log E)

1. 그래프 생성 O(Unknown) * O(K * log E):
E : 간선의 수, V : 정점의 수, K = (E/V) : 정점 당 간선의 수
그래프에는 두 가지 자료구조가 사용되었는데, 바로 해시맵과 TreeSet입니다. 해시맵에는 해시가 사용되는데, 해시함수가 얼마나 충돌이 일어나는지에 따라서 탐색(접근)의 시간복잡도가 달라지므로 O(Unknown)으로 표기하였습니다.

그리고 TreeSet은 Red-Black Tree로 구현되어있으므로, 삽입의 시간복잡도가 O(log E)입니다. 그런데 한 정점 당 연결된 정점의 개수는 정해져있지 않으므로 K = (E/V), 즉 각 정점에 균등하게 간선이 존재한다고 가정하였습니다.

2. 정점의 인접 정점 개수 구하기 O(Unknown) * O(1):
임의의 정점에 대한 Set을 HashMap으로 탐색하는 건 O(Unknown)이고,
Set의 사이즈를 구하는 것은 O(1)이므로 시간복잡도는 O(Unknown)입니다.

문제 2 : BFS
문제 : 너비 우선 탐색(BFS)으로 그래프를 탐색할 때, 탐색 순서를 출력하는 문제
해결 방법 :



<BFS.GIF>

너비우선 탐색을 구현하여 모든 정점을 탐색하려면
1. 처음 시작할 정점을 큐에 넣습니다. 그리고 방문했음을 기록합니다.
2. 큐가 빌 때 까지 큐에 있는 정점을 순회하는데,
큐에 있는 정점의 인접한 정점을 모두 다시 큐에 넣습니다.
3. 큐가 빈다면 모든 정점을 순회한 것입니다.

위와 같이 구현하면 파도가 퍼지는 것처럼 가까운 정점부터 순회하는 것을 알 수 있습니다.
단, 이때 인접한 정점이 여러 개가 있을 경우 정점을 방문하는 순서가 있어야합니다. 위의 사진에서는 알파벳 순서대로 방문했습니다.

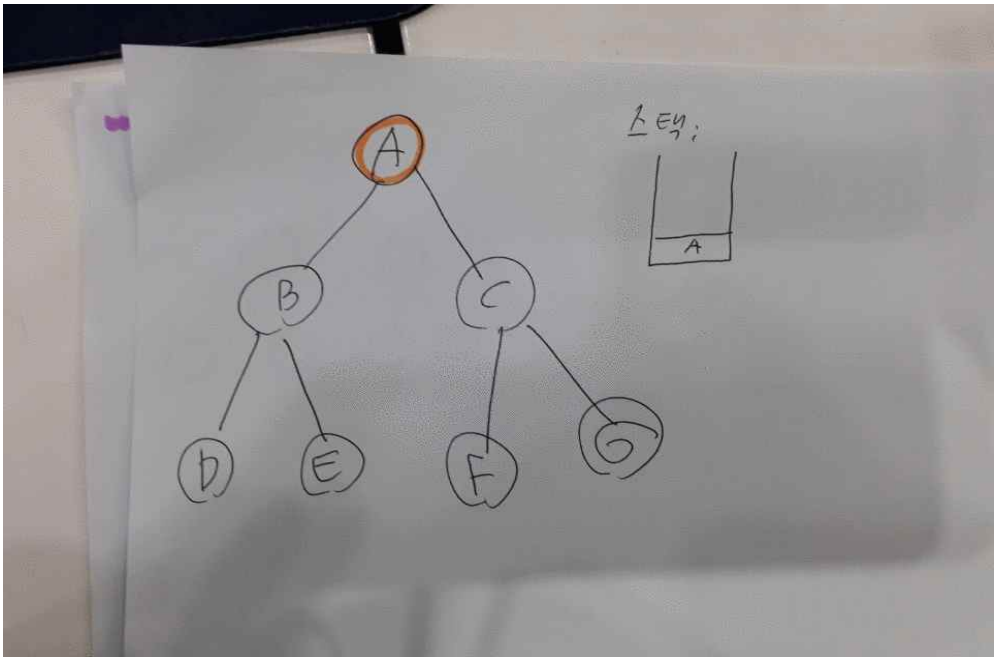
```java
public static void bfs(String v) {
    Queue<String> q = new LinkedList<String>();
    visited.put(v, true);
    sb.append(v).append(" ");
    q.add(v);
    while(!q.isEmpty()){
        v = q.poll();
        for(String v_ : graph.get(v)) {
            if(!visited.get(v_)) {
                visited.put(v_, true);
                q.add(v_);
                sb.append(v_).append(" ");
            }
        }
    }
}
```

<BFS 구현 코드>

시간복잡도 : O(V + E)
BFS 알고리즘의 시간복잡도는 V 또는 E인데, 저의 경우는 인접 리스트(여기선 인접 셋?))으로 구현했기 때문입니다. 인접 행렬로 구현할 경우에는 V^2이 됩니다. 무튼 간에 인접 행렬로 구현하면 모든 정점을 방문하므로 정점의 수, 또는 간선의 수 중에 큰 것이 시간복잡되가 됩니다. 이를 시간복잡도로 표현하면 O(V + E)라고 할 수 있습니다.

문제 3 : DFS
문제 : 깊이 우선 탐색(DFS)으로 그래프를 탐색할 때, 탐색 순서를 출력하는 문제
해결 방법 :
DFS 알고리즘은 그래프의 가장 깊은 곳까지 탐색하고 다시 돌아온 후, 다시 깊은 곳으로 탐색하는 방법입니다. 스택을 활용하여 구현할 수 있는데, 따로 스택 컬렉션을 사용하지는 않고, 함수의 작동 원리에 스택이 활용되므로, 함수를 활용하면 됩니다.

DFS로 탐색을 하려면
방문하지 않은 모든 정점에 대하여 dfs(v)를 호출하면 됩니다.
DFS의 특징은 가장 깊은 곳을 먼저 탐색한다는 특징이 있습니다.



<DFS.GIF>

```java
public static void dfs(String v) {
    visited.put(v, true);
    sb.append(v).append(" ");
    for(String v_ : graph.get(v)) {
        if(!visited.get(v_)) {
            dfs(v_);
        }
    }
}
```

<DFS 소스코드>

```
Console
<terminated> DFS [Java Application] C:₩Program Files₩Java₩jre1.8.0_211₩b
12 11
A B C D E F G H I J K L
A B
A C
A D
B E
B F
D G
D H
E I
E J
G K
G L
A
A B E I J F C D G K L H
```

<DFS 실행결과>

시간복잡도 : O(V + E)
DFS도 BFS와 같이 단순히 모든 정점을 방문하는 알고리즘이므로 크게 시간복잡도가 차이나지는 않습니다. 다만 탐색을 하는 방식이 다르기 때문에, 주어진 문제 상황에 따라 어떤 것을 활용할지 선택해야 합니다.

문제 4 : P2P
문제 : 너비우선탐색으로 그래프의 가장 긴 거리를 구하는 문제
해결 방법 : 사실 처음 문제를 봤을 땐 DFS로 풀려고 했습니다. 그래서 그래프의 깊이가 가장 깊은 부분을 찾을려고 했는데, 생각해보니 그래프는 트리와 다르게 싸이클이 존재해서 단순히 깊이를 재는 방법으로는 풀 수 없겠다는 걸 깨달았습니다. 그래서 제 생각엔 DFS로 풀려면 정점과 정점사이의 거리 중에서 가장 긴 것을 찾아야겠다고 생각했습니다. 그런데 너무 복잡하게 풀기는 싫어서 BFS로 구현했습니다.

그래서 BFS로 이 문제를 해결한 방법은, BFS는 거리가 가까운 순서대로 정점을 방문하기 때문에 출발 정점에서 시작해서 가장 마지막에 도달하는 정점까지의 거리가 필요한 시간이라고 생각했습니다. 그래서 A->B정점로 갈 때, 출발점에서 A정점까지로 올 때 걸린 시간을 timeVal에 불러온 후, B정점에 timeVal+1을 저장했습니다.

그래서 timeVal의 최댓값을 구해서 출력했습니다.

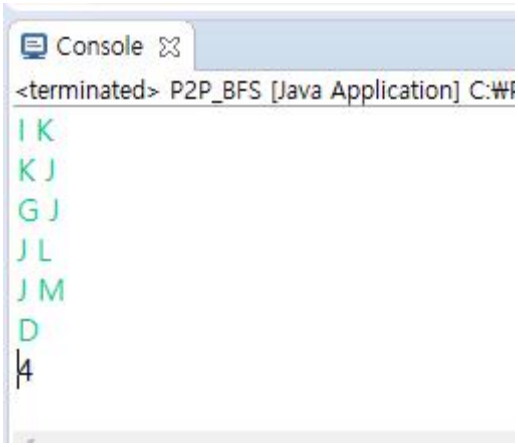
```
public static void bfs(String v) {
    Queue<String> q = new LinkedList<String>();
    visited.put(v, true);
    q.add(v);
    while(!q.isEmpty()){
        v = q.poll();
        int timeVal = time.getOrDefault(v, 0);
        for(String v_ : graph.get(v)) {
            if(!visited.get(v_)) {
                visited.put(v_, true);
                q.add(v_);
                time.put(v_, timeVal+1);
            }
        }

    }
}
```

<P2P 소스 코드 : BFS 구현 부분>

```
String name = br.readLine();
bfs(name);
int maxTime = 0;
for(String key : time.keySet()) {
    maxTime = Math.max(maxTime, time.get(key));
}
System.out.println(maxTime);
br.close();
```

<P2P 소스코드 : maxTime을 찾는 부분>



<실행 결과>

시간복잡도 : O(V + E + Unknown)

1. BFS로 그래프를 탐색할 때의 시간복잡도 : O(V + E)

2. HashMap으로 탐색할 때의 시간복잡도 : O(Unknown)

느낀점 :
아직 그래프 문제를 많이 풀어본 건 아니지만 그래프에 이름을 부여하는 문제는 처음 봐서 약간 당황했는데, 잘 해결해서 뿌듯했습니다.