2022 여름학기 동국대학교 SW역량강화캠프

10일차. 그래프 2

● 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

```
graph = new ArrayList<ArrayList<Integer>>();
for(int i=0; i<=N ; i++) {
    graph.add(new ArrayList<Integer>());
}

for(int i = 0; i < M; i++) {
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());

    graph.get(u).add(v);
    graph.get(v).add(u);
}
```

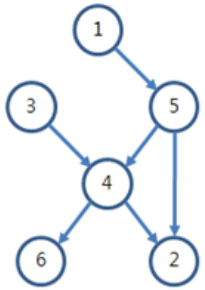
```
int N,M,V; cin>>N>>M>>V;
for(int i=0; i<M; i++)
{
    int x,y; cin>>x>>y;
    v[x].push_back(y);
    v[y].push_back(x);
}
```

● 키 순서 (4313)

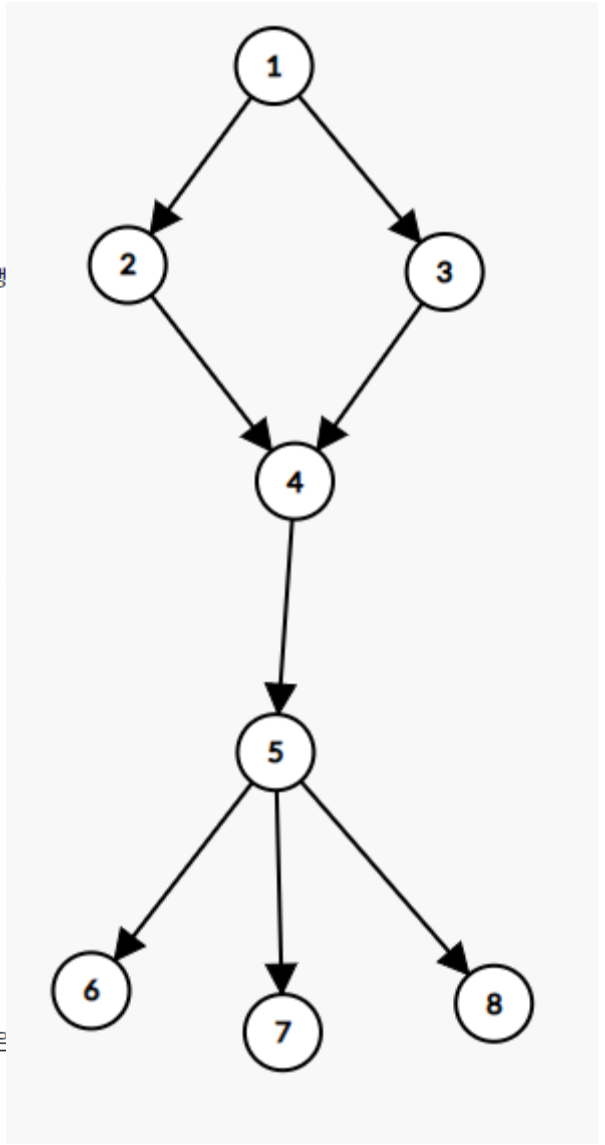
문제

1번부터 N번까지 번호가 붙여져 있는 학생들에 대하여 두 학생끼리 키를 비교한 결과의 일부가 주어져 있다. 단, N명의 학생들의 키는 모두 다르다고 가정한다. 예를 들어, 6명의 학생
만 키를 비교하였고, 그 결과가 다음과 같다고 하자.

1번 학생의 키 < 5번 학생의 키
3번 학생의 키 < 4번 학생의 키
5번 학생의 키 < 4번 학생의 키
4번 학생의 키 < 2번 학생의 키
4번 학생의 키 < 6번 학생의 키
5번 학생의 키 < 2번 학생의 키
이 비교 결과로부터 모든 학생 중에서 키가 가장 작은 학생부터 자신이 몇 번째인지 알 수 있는 학생들도 있고 그렇지 못한 학생들도 있다는 사실을 아래처럼 그림을 그려 쉽게 확인할
생의 키가 b번 학생의 키보다 작다면, a에서 b로 화살표를 그려서 표현하였다.



1번은 5번보다 키가 작고, 5번은 4번보다 작기 때문에, 1번은 4번보다 작게 된다. 그러면 1번, 3번, 5번은 모두 4번보다 작게 된다. 또한 4번은 2번과 6번보다 작기 때문에, 4번 학생은
학생이 3명이 있고, 자기보다 큰 학생이 2명이 있게 되어 자신의 키가 몇 번째인지 정확히 알 수 있다. 그러나 4번을 제외한 학생들은 자신의 키가 몇 번째인지 알 수 없다.



문제 해설

▶ 우선 그래프 문제이니 인접 리스트 형태로 입력을 받는다.

▶ 방향이 있는 간선이므로 간선을 추가할 때 역방향 간선을 넣지 않도록 유의해주자

```
Graphlist = new ArrayList<ArrayList<Integer>>();
// Graph 인접리스트 만들기
for (int i = 0; i <= N; i++)
    //Graphlist -> get(0) ~ get(N)
    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); 단방향 간선이므로 q -> p 는 넣어주지 않는다.
}
```

문제 해설

▶ i번 정점에서 DFS를 돌리면, i번 정점에서 방문 가능한 정점, 즉 자신보다 키가 큰 학생의 수를 알 수 있다.

▶ 자신의 등수를 정확히 알 수 있는 사람은
(자신보다 키가 큰 학생의 수) + (자신보다 키가 작은 학생의 수) = N−1
인 사람이다.

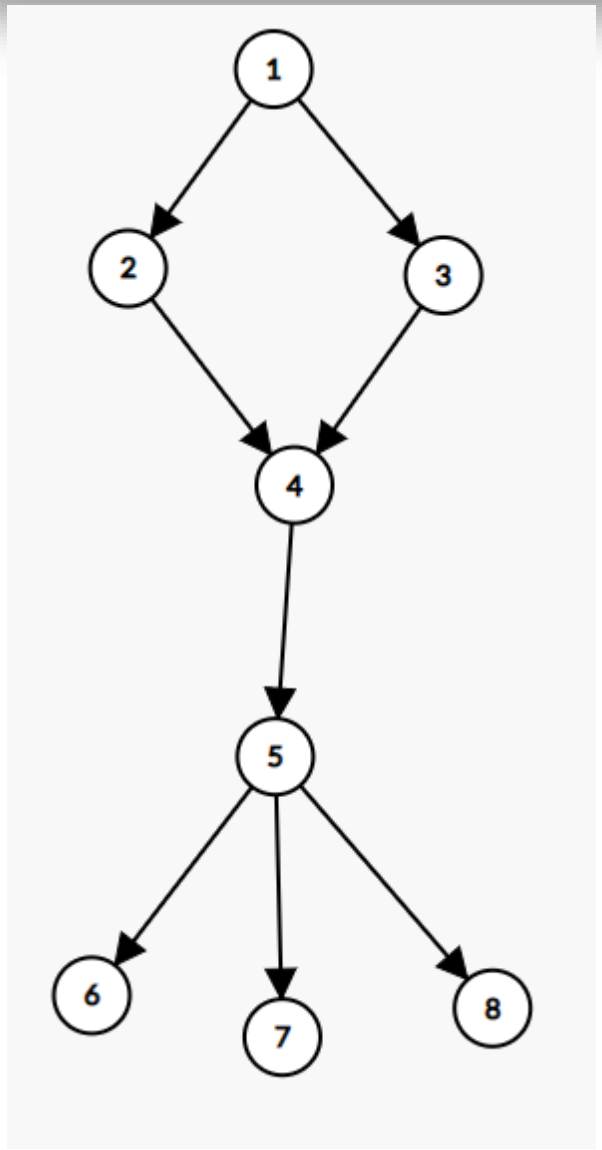
▶ 자신보다 키가 작은 학생의 수, 즉 i번 정점을 방문 가능한 정점은 어떻게 알 수 있을까?

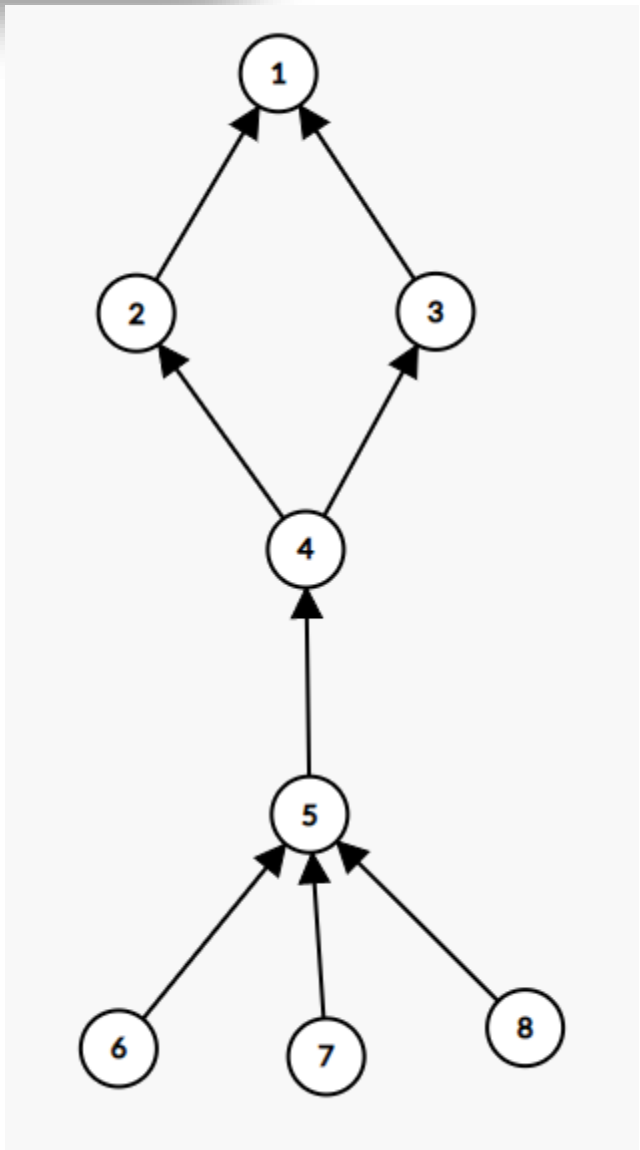
▶ 역방향 그래프를 이용하자.

▶ 원래의 그래프가 (키가 작은 사람) → (키가 큰 사람) 의 간선을 담고 있다면
그래프를 하나 더 만들어 (키가 큰 사람) → (키가 작은 사람) 의 간선을 담도록 하자

▶ 원래 그래프에서 i번 정점을 방문 가능한 정점 =
역방향 그래프에서 i번 정점에서 방문 가능한 정점

문제 해설





핵심 코드

```
fw = new ArrayList<ArrayList<Integer>>();
bk = new ArrayList<ArrayList<Integer>>();
for(int i=0; i<=N ; i++) {
    fw.add(new ArrayList<Integer>());
    bk.add(new ArrayList<Integer>());
}

for(int i = 0; i < M; i++) {
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());

    fw.get(u).add(v);
    bk.get(v).add(u);
}
```

```
int ans = 0;
for(int i = 1; i <= N; i++) {
    visit = new boolean[N + 1];
    cnt = 0;
    dfs(i, fw);
    dfs(i, bk);
    if(cnt == N) ans ++;
}
System.out.println(ans);
```

핵심 코드

```
public static void dfs(int x, ArrayList<ArrayList<Integer>> list) {
    if(visit[x]==false) {
        cnt++;
    }
    visit[x] = true;
    for(int y : list.get(x)) {
        if(!visit[y]) {
            dfs(y, list);
        }
    }
}
```

● 이분 그래프 (4693)

문제

그래프의 정점의 집합을 둘로 분할하여, 각 집합에 속한 정점끼리는 서로 인접하지 않도록 분할할 수 있을 때, 그러한 그래프를 특별히 이분 그래프 (Bipartite Graph) 라 부른다.

그래프가 입력으로 주어졌을 때, 이 그래프가 이분 그래프인지 아닌지 판별하는 프로그램을 작성하시오.

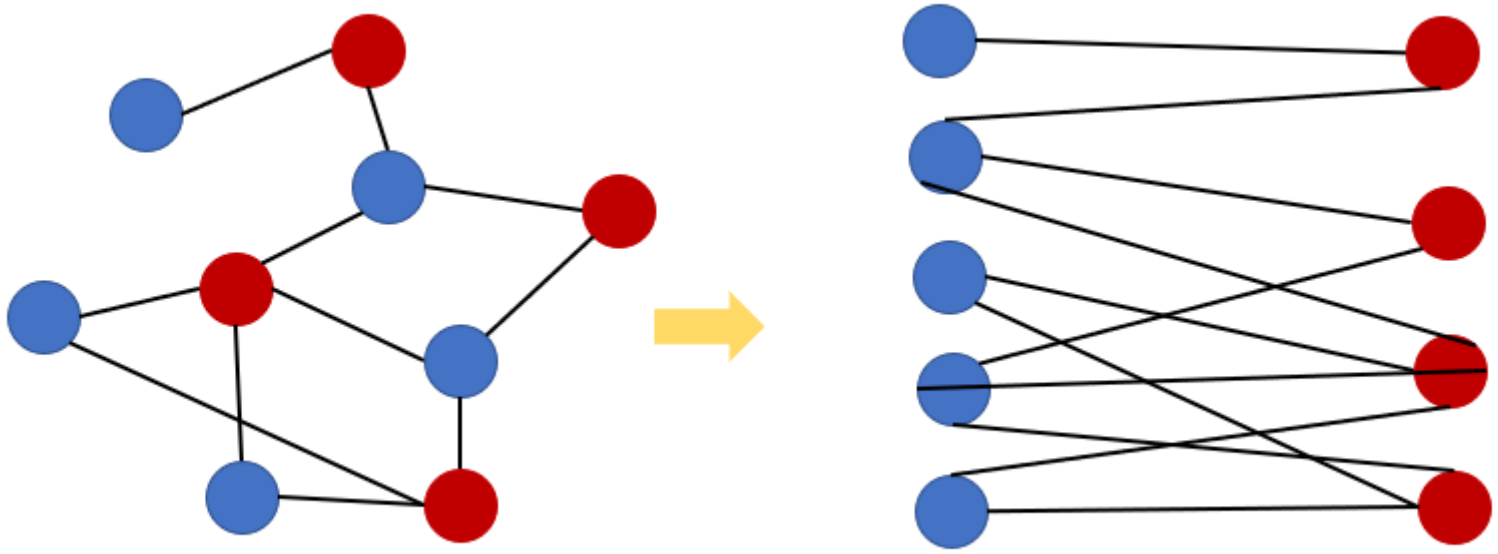
입력

첫 줄에 그래프의 정점의 개수 V(1 ≤ V ≤ 20,000)와 간선의 개수 E(1 ≤ E ≤ 200,000)가 빈 칸을 사이에 두고 순서대로 주어진다.
각 정점에는 1부터 V까지 차례로 번호가 붙어 있다.
둘째 줄부터 E개의 줄에 걸쳐 간선에 대한 정보가 주어지는데, 각 줄에 인접한 두 정점의 번호가 빈 칸을 사이에 두고 주어진다.

출력

입력으로 주어진 그래프가 이분 그래프이면 YES, 아니면 NO를 출력한다.

● 이분 그래프 (4693)



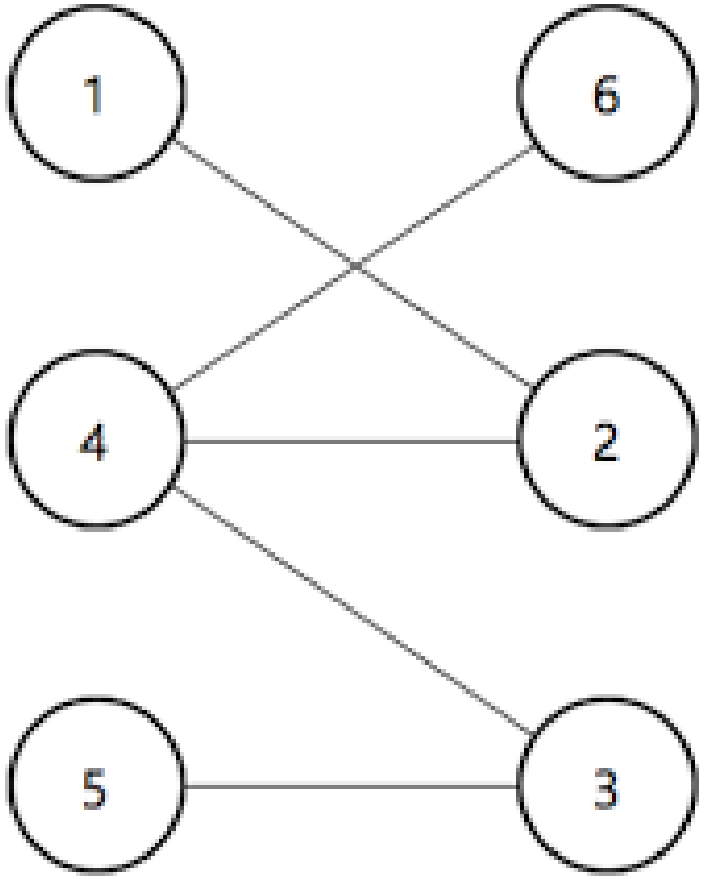
▶ DFS를 이용해서 정점에 색을 칠하자

▶ visited 배열 대신 color 배열을 사용, 0(방문하지 않음) , 1(빨강), 2(파랑)

▶ color가 1인 정점에서 인접한 정점들은 전부 color를 2로 칠해야 한다.

0인 정점을 발견하면, 2로 칠하고 그 정점에서 dfs
1인 정점을 발견하면 불가능한 경우이므로 불가능 체크
2인 정점은 pass

▶ 모든 정점을 색칠하고 불가능 여부를 확인하면 된다.


1
6
4
2
5
3

핵심 코드

```
public static void dfs(int x, int c) {
    color[x] = c;
    for (int y : graph.get(x)) {
        if (color[y] == color[x]) pos = false;
        if (color[y] == 0) {
            dfs(y, 3 - c);
        }
    }
}
```

```
color = new int[N+1];
pos = true;

for(int i = 1; i <= N; i++) {
    if(color[i] == 0) dfs(i, 1);
}
if(pos) System.out.println("YES");
else System.out.println("NO");
```