과제 #8 islands tour - 201921438 조현태

1) 소스코드

```
#include<iostream>
#include<algorithm>
#include<vector>
#include<stack>
using namespace std;

vector<int> v; // 그래프
vector<bool> check; // dfs 함수의 방문여부
vector<bool> visit; // dfs_for_cycle 함수의 방문여부
vector<int> indegree; // 그래프에서 indegree값 저장 벡터
vector<int> outdegree; // 그래프에서 outdegree값 저장 벡터
vector<int> dp; // 경우의 수

// 함수 선언
void dfs_for_cycle(int i);
void dfs(int i);

// 정답
int answer = 0;
// n = 점의 개수, m = 짚라인의 개수
int m, n;

int main(void)
{
	cin >> m >> n;

	// 벡터 크기 할당
	v.clear(), v.resize(n+1);
	check.clear(), check.resize(n+1,false);
	visit.clear(), visit.resize(n+1,false);
	indegree.clear(), indegree.resize(n+1);
	outdegree.clear(), outdegree.resize(n+1,true);
	dp.clear(), dp.resize(n+1,1);

	// 그래프 입력 (간선의 개수만큼 반복)
	for (int i = 0; i < m; i++)
	{
		int a, b;
		cin >> a >> b;
		// bool 배열을 사용하므로 1을 더해서 0을 없앰.
```

```
		v[a+1] = b+1;
		// indegree(들어오는 방향)의 수를 체크함.
		indegree[b+1]++;
		// outdegree(나가는 방향)이 없는 섬을 체크함.
		outdegree[a+1] = false;
}

// 간선(징라인이 없을 경우)
if (m == 0)
{
		// 정점(섬)이 없을 경우
		if (n == 0)
		{
			cout << 0 << endl;
			return 0;
		}
		// 정점(섬)이 있을 경우
		else
		{
			cout << 1 << endl;
			return 0;
		}
}

// 모든 정점(섬)에 대해서 cycle 탐색 실행
for (int i = 1; i <= n; i++)
{
		// 단, 한번이라도 방문기록이 있다면 예외처리
		if (!visit[i])
			dfs_for_cycle(i);
}

// answer = 사이클의 최대 크기
answer = *max_element(dp.begin()+1, dp.end());
for (int i = 1; i <= n; i++)
		// indegree = 0 -> 가장 끝쪽의 정점 + outdegree가 존재하는 경우
		if (indegree[i] == 0 && !outdegree[i])
			dfs(i);

// 정답 출력
cout << answer << endl;

return 0;
```

```
}

// 사이클 찾기
void dfs_for_cycle(int i)
{
	vector<int> cycle; // cycle 정점의 벡터
	stack<int> s; // stack

	s.push(i);
	// 방문 처리
	visit[i] = true;
	// stack의 크기가 없을 때까지
	while (!s.empty())
	{
		// 현재 노드(섬) = 스택의 가장 위쪽 원소
		int cur_node = s.top();
		// 스택의 가장 위쪽 원소 제거
		s.pop();
		// 다음 노드(섬) 설정
		int next_node = v[cur_node];

		// 다음 정점을 방문하지않았다면
		if (!visit[next_node])
		{
			// 방문 처리
			visit[next_node] = true;
			// 스택 처리
			s.push(cur_node);
			s.push(next_node);
			// cycle에 추가
			cycle.push_back(cur_node);
		}
		// 다음 정점을 방문했다면
		else
		{
			// cycle에 추가
			cycle.push_back(cur_node);
			cycle.push_back(next_node);

			// 중복되는 원소
			int overlap = next_node;
			// 사이클의 크기, 시작 인덱스
			int size = 0;
```

```
                        int start = 0;
                        for (int i = 0; i < cycle.size(); i++)
                                // 중복되는 원소와 같을 때
                                if (cycle[i] == overlap)
                                {
                                        // 사이클의 크기
                                        size = int(cycle.size()) - (i+1);
                                        // 사이클의 시작 인덱스
                                        start = i;
                                        break;
                                }
                        // 사이클이 없을 경우
                        if (size == 0 && start == 0)
                                return;

                        // 사이클의 정점에 크기를 입력하고, 방문처리를 한다.
                        for (int i = start; i < cycle.size()-1; i++)
                        {
                                dp[cycle[i]] = size;
                                check[cycle[i]] = true;
                        }
                        return;
                }
        }
}

// indegree = 0인 정점에서 다른 정점까지의 거리
void dfs(int i)
{
        while (1)
        {
                int next = v[i];
                // 다음 정점이 사이클의 정점인 경우 + outdegree가 없는 경우
                if (check[next] || outdegree[next])
                {
                        // 정답 = "indegree = 0"인 점부터 바로 직전의 점의 경우의 수 + 사
이클 or 마지막 점의 경우의 수
                        answer = max(answer, dp[next] + dp[i]);
                        return; // 종료
                }

                // 다음 정점에 방문기록이 있을때
                if (dp[next] > 0)
```

```
                {
                        // 이번 경로가 원래 경로보다 작을때
                        if (dp[next] == max(dp[i] + 1, dp[next]))
                                return; // 종료
                        dp[next] = dp[i] + 1;
                }
                // 다음 정점에 처음 도착했을 때
                else
                        // 다음 정점의 경우의 수 = 이전 정점의 경우의 수  + 1
                        dp[next] = dp[i] + 1;

                // 다음 단계로 가기위한 변수값 설정
                i = next;
        }
}
```

2) 문제 설명

각 섬에서 나갈 수 있는 방법은 1가지이고 단방향인 그래프이므로
무조건 사이클이 형성된다는 것을 알 수 있습니다.

이 경우, 2가지 형태로 형성이 됩니다.
첫 번째로는 사이클만 생기는 경우입니다.
이 경우는 사이클을 찾은 후, 사이클의 크기가 가장 큰 사이클을 출력하면 됩니다.

두 번째로는 사이클에 가지가 뻗어나가는 모양입니다.
이 경우는 사이클을 찾은 후, 사이클에 도달하는 모든 경우의 수에 사이클의 크기를
더해서 이 경우의 수 중 최댓값을 출력하면 됩니다.

이때, dfs를 이용해 사이클을 찾아야하지만,
n, m이 100만이므로 재귀함수를 이용한 dfs를 이용할 경우, 스택 오버플로우가 발생합니다.
따라서 stack을 이용한 dfs로 사이클을 찾은 후, 사이클의 최댓값에 answer를 입력하고
indegree = 0이고 outdegree = 1인 점에서부터 dfs를 이용하여
사이클 또는 outdegree = 0인 점에 도달할때까지 dp를 이용해 값을 계산한 후,
answer값에 max를 사용하여 최댓값으로 갱신하면 정답을 구할 수 있습니다.