**1. 개요**

**# 설계 목적**

* **인하커뮤니티에서는 보다 나은 서비스를 위해 회원들 간의 친분관계(friendship)를 유향그래프(directed graph)로 모델링**
* **강결합요소(strongly connected components)를 사용하여 관리**
* **유향그래프는 인접리스트표현(adjacency-list representation)으로 구현한다.**

**# 요구 사항**

* **파일로부터 회원 목록 불러오기**
* **SCC를 계산하는 알고리즘은 Kosaraju's algorithm으로 구현**
* **회원의 리더 정보를 조회하는 기능 구현**
* **팔로우 하기 / 끊기 기능 구현**
* **프로그램 종료 기능 구현**
* **프로그램은 5초 이내 수행**

**# 개발 환경**

* **OS : Windows 10**
* **Language / Tool : C++ / Visual Studio 2017**

**2. 필요한 자료구조 및 기능**

**# 필요한 자료구조**

**+ class Member**

**: 회원을 표현할 클래스. 멤버변수로 id, name, phone, leader를 가지고 있다.**

**+ Member arrayByID[5001]**

**: 회원의 ID를 index를 삼는 Member 1차원 배열**

**+ vector<list<int>> adjacencyList(5001)**

**: 회원들 간의 친분관계(friendship)를 유향그래프(directed graph) 및 인접리스트표현**

**+ vector<list<int>> adjacencyList(5001)**

**: 회원들 간의 친분관계(friendship)를 역방향 유향그래프(directed graph) 및 인접리스트**

**표현, Kosaraju’s algorithm 구현시 필요**

**+ int visited[5001]**

**: DFS 구현시 필요한 배열. Graph Traversal시 각 Node의 방문여부를 저장함.**

**3. 기능별 알고리즘 명세- (N:현재 저장되 있는 멤버의 수)**

**# dfs(cur)**

**: cur(현재 노드)에서부터 dfs 진행. finish되는 멤버는 stack에 push**

* **시간복잡도는 인접리스트로 구현했으므로 현재 노드의 인접노드 개수 O(E)**
* **공간복잡도는 cur의 인접리스트와 visited의 배열이 필요하고 stack에 push하는 횟수 만큼 추가적인 메모리 요구O(3N)**

**# reverseDFS(cur, leader)**

**: cur(현재 노드)에서부터 역방향 dfs 진행. 진행 중 leader를 저장.**

* **시간복잡도는 인접리스트로 구현했으므로 현재 노드의 역방향 인접노드 개수 O(E)**
* **공간복잡도는 cur의 인접리스트와 visited의 배열이 필요O(2N)**

**# Kosaraju**

**: SCC의 개수와 각 회원의 leader를 설정해 주기 위한 알고리즘**

* **시간복잡도**

**+ DFS : 0번부터 N번까지 DFS실행. 모든 노드가 visited될 때 까지 수행되므로 O(N+E)**

**+ reverseDFS: stack에 저장된 원소를 pop하면서 reverseDFS 수행. Pop된 멤버가**

**visited되지 않은 멤버이면 SCC 개수를 증가시키고 해당 멤버를 leader로 삼음.**

**모든 노드가 visited될 때 까지 수행되므로 O(N+E).**

**Con. O(2(N+E))**

* **공간복잡도는 visited[N], stack(N), vector<list<int>> adjacencyList(N)**

**vector<list<int>> reverse\_adjacencyList(N)의 자료구조들이 요구되므로 O(2\*N^2+2N)**

**# accessLeaderInfo**

**: id를 가진 멤버가 속한 그룹의 리더 정보 출력. (1)번 기능 구현 메서드**

* **시간복잡도는 리더의 ID를 배열접근을 통해 얻어 올 수 있기 때문에 상수시간에 해결**

**가능. 공간복잡도는 arrayByID가 필요하기 때문에 해당 배열을 구성하기 위한 추가적**

**인 메모리 요구O(N)**

**# changeFollowing**

**: 팔로우 하기 / 끊기. (2)번 기능 구현 메서드**

**A\_ID의 인접리스트를 iterate하면서 B\_ID가 있는 지 확인한다. 일치하는 경우 following**

**되어있다는 의미이므로 B\_ID 노드를 지운다. List로 구현되어있으므로 erase는 상수시간에 해**

**결가능. 정방향이 지워졌으므로 역방향도 마찬가지방식으로 수행한다. 만약 following되어 있**

**지 않다면 A\_ID의 인접리스트에 B\_ID 노드를 추가하면 되고 B\_ID 역방향 인접리스트에 A\_ID**

**를 추가하면 된다. 리스트에 마지막 삽입 연산은 상수시간에 수행 가능. 멤버를 구성하는 그**

**래프가 변경되었으므로 Kosaraju 알고리즘 재실행한다.**

**- 시간복잡도는 두번의 인접리스트 iterate가 요구되는 최악의 경우 O(2N) + Kosaraju**

**O(2(N+E))**

**- 공간복잡도는 Kosaraju O(2\*N^2+2N)**