**알고리즘 설계과제1**

**12161581 박진성**

**qkrwlstjd2@naver.com**

**1. 개요**

이번 과제에서는, adjacency list rep 로 digraph 를 구현하여,

해당 digraph 를 입력 받아 DFS 알고리즘을 통해, SCC problem 을 해결하고,

그 결과를 출력하는 내용이었다.

개발환경은 windows visual studio 를 이용하였으며, 사용언어는 C++

사용 STL 은 다음과 같다

#include <iostream>

#include <vector>;

#include <algorithm>;

#include <stack>;

주안점은 어떻게 digraph를 인접리스트로 잘 구현하는가,

그리고, DFS 알고리즘을 통해 어떻게 SCC를 구하는가

이 두가지였다고 생각한다

**2. 필요한 자료구조 및 기능**

**1.** vector<int> user[50001] / vector<int> userT[50001]

원래 인접리스트를 정석대로 구현하게 되면, Doublylinkedlist 를 구현해야 되지만,

나는 세세하게 Doublylinkedlist 를 구현하는 것 보다, DFS 알고리즘 구현이 우선이라고 생각하였기 때문에,

오직 Vector 만 이용해, 인접리스트로 Digraph를 구현하였다.

정방향 그래프 user 와 역방향 그래프 userT 두가지를 구현하였다

**2.** stack<int> finished / stack<int> for\_print

Finish 된 vector 를 담아줄 stack을 사용하였다.

두번째 DFS 의 수행순서를 결정해주는 stack 이다

그리고, stack 에 들어가는 순서대로 출력에 필요한

For\_print 도 stack 으로 구현하였다

**3.** vector<int> priority\_v

첫번째 DFS 를 수행할 때, 수행순서를 Vector에 담았다.

팔로워수 내림차순 및, 동률일경우 id 오름차순 으로 정렬하였다

**4.** int explored[50001] / int SCC\_leader[50001]

해당 vertex 를 visit 했는지 안했는지를 저장할 explored 배열과

각 vertex 별 SCC leader 를 저장할 SCC\_leader 배열 두가지를

배열로 구현하였다

**3. 기능별 알고리즘 명세**

1. DFS **알고리즘**

1. 1 phase (DFS)

첫번째 DFS는, priority\_v 라는 이름의 vector 에 왼->오 순서대로 수행하게 된다.

DFS 내부에서도, 원하는 우선순위로 재귀적으로 진행하게 되는데,

이는, 미리 adjacency list 에서 각 행의 우선순위를 sort를 통해 정렬시켜놨기 때문이다.

bool compare(int a, int b) {

return (userT[a].size() == userT[b].size()) ? a < b : userT[a].size() > userT[b].size();

}

다음 조건으로 adjacency list 의 각 행을 미리 정렬 시켜놓았다.

explored[id] 를 1로바꿔, 방문했음을 기록한뒤

해당 id가 팔로우한 id들을 우선순위대로 재귀적 방문하게된다.

더 이상 방문할 next\_id 가 없는경우, finished 라는 이름의 stack 에 저장되게된다

1. 2 phase (DFS2)

두번째 DFS 에서는, 앞서 저장된 finished 라는 이름의 stack 의 pop 순서대로, 수행되게 된다. 첫번째 DFS 와 마찬가지로, explored[id] 를 1로 바꿔, 방문했음을 기록한뒤

해당 id 가 포함되는 SCC 의 leader 을 저장한뒤, 루프를 통해 역방향 그래프 userT 를 finished stack 의 우선순위에 의거하여 수행하게 된다.

1. **수행시간**

DFS 알고리즘은 최악의 경우, 각각 O(N+M) time 을 가지게 된다

추가적으로, 역방향 그래프인 userT 생성에도 O(N+M) time 을 가지게 된다.

따라서, 총 시간복잡도는 O(n+m) time 이 된다.

**2. sorting 알고리즘**

1. **user / userT sorting**

원활한 DFS 진행을 위하여 그래프를 인접행렬로 각각 정방향 역방향 구현한

user, userT의 각 행을 미리 sorting 해야 했다.

STL algorithm 의 sort 메소드를 이용하였으며, 조건은 compare1, compare2 함수를 각각

작성하여, 넣어주었다.

for (int i = 1; i <= N; i++) { // 각 팔로잉 user 들을 sorting.

sort(user[i].begin(), user[i].end(), compare);

sort(userT[i].begin(), userT[i].end(), compare);

}

다음과 같이, for문을 통하여 sorting 시켜주었는데, sort 메소드는 quick sort 기반으로 만들어

졌으므로, 시간복잡도는 O(NlogN) 이 된다

1. **priority\_v sorting**

첫번째 DFS 를 수행하기 위해서는, 특별한 우선순위가 존재한다.

바로, **“팔로워수가 많은 순서대로. 만약 동률이면 id 작은순서”**

이를 DFS를 수행하며 만족시키기 위해서는, 너무 복잡해지는걸 느껴서, 미리

Sorting 하는 방향으로 선회하였다.

for (int i = 1; i <= N; i++) { // DFS 실행시킬 순서 담은 vector 생성

priority\_v.push\_back(i);

}

sort(priority\_v.begin(), priority\_v.end(), compare);

첫번째 sorting 과 마찬가지로, STL 의 sort를 이용하였으며, 조건에 넣은 함수도 동일하다.

단순히, 1번부터 N번까지의 vertex가 존재하므로, vector에 1부터 N까지 삽입한뒤,

Sorting 하였다.

**4. 인터페이스 및 사용법**

**- 간단한 사용법 설명**

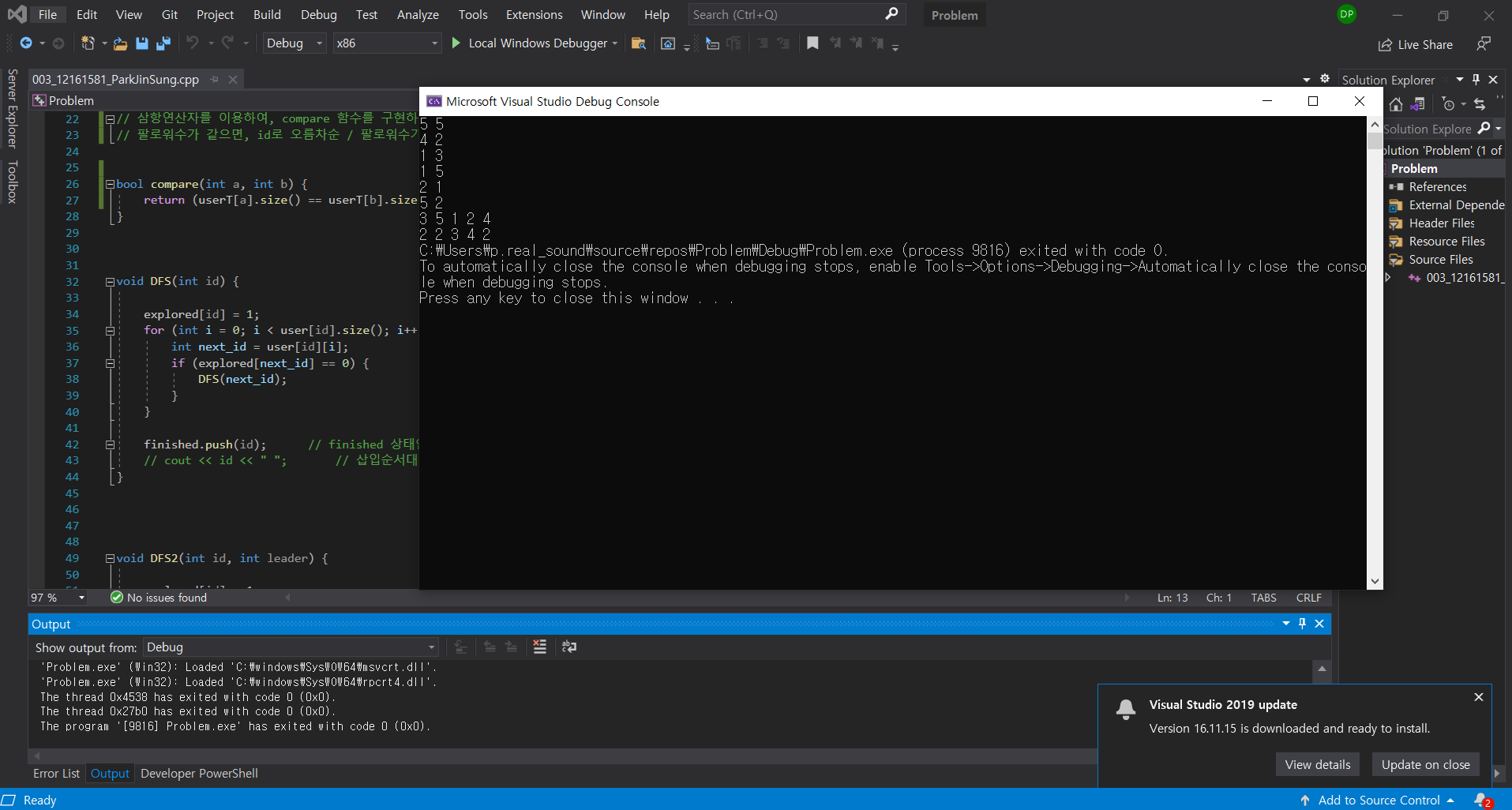
처음 입력은, 사용자수(vertex 수), 총팔로워수(edge 수) 를 입력한다

그후, A B 순으로 입력하는데, A 가 B 를 팔로우한다는 뜻이다

이런식으로 총 팔로워수 만큼 입력을 끝마치게 되면,

Finished stack 에 들어간 순서대로 출력이 한줄로 되고

그다음줄에 각 vertex의 SCC leader 가 누구인지 한줄로 출력이 된다



**5. 평가 및 개선 방향**

**- 구현한 알고리즘의 장점**

우선, DoublyLinkedList 가 아닌, vector 로 간결하게 인접리스트를 구현하였기에, 가독성 이 매우 좋고, 메소드를 구현할 때 또한 수월하게 구현할 수 있었다.

또한, 다른 구현방법보다, graph 를 만드는것에 있어서 직관적으로 생각할 수 있었다.

**- 구현한 알고리즘의 단점**

하지만, vector 로 구현하였기 때문에, edge 에 대한 정보를 담을 수가 없다.

문제의 요구사항이 더 추가된다면, 곤란할 수도 있겠다는 생각이 들었다.

또한, edge 가 없기때문에, 가중그래프를 구현 못하여서

DFS 알고리즘 수행시, 그때그떄 우선순위를 판단하지 못하고, 어쩔수 없이 모두 사전에 sort 시켜야 한다는 단점도 존재하는 것 같다

**- 향후 개선 방향**

해당 문제에서는 edge 정보가 없어도 해결이 가능하였지만, edge 정보를 추가하는 방향 으로 개선된다면, 더 효율을 높일 수 있다고 생각한다