JooHyun – Lee (comkiwer)

TS인플루언서

Hancom Education Co. Ltd.

Problem

Problem

인플루언서는 영향력 있는 사람을 가리키는 말이다.

SNS에서 인플루언서들을 찾아내고자 한다.

SNS에는 N명의 회원이 있으며, 각 회원의 번호는 0 이상 N-1 이하의 고유한 정수이다.

두 회원은 서로 친구일 수 있다.

친구 관계는 양방향이다.
만약 1번 회원이 2번 회원의 친구라면, 2번 회원도 반드시 1번 회원의 친구이다.

자기 자신을 다른 말로, 0촌이라고 한다.

친구를 다른 말로, 1촌이라고 한다.

친구의 친구이면서, 0촌이 아니고 1촌도 아닌 회원을 2촌이라고 한다.

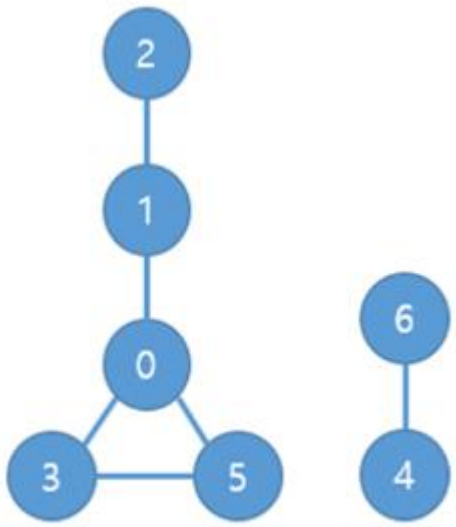
친구의 친구의 친구이면서, 0촌이 아니고 1촌도 아니고 2촌도 아닌 회원을 3촌이라고 한다.

0촌, 1촌, 2촌, 3촌을 모두 합해서 '3촌 이내의 관계' 라고 표현한다.

Problem

[Fig. 1]은 N이 7인 경우의 예시이다. 간선은 친구 관계를 의미한다.

[Table 1]은 [Fig. 1]을 분석하여 각 회원의 1촌, 2촌, 3촌을 적어놓은 표이다.



[Fig. 1]

	1촌	2촌	3촌
0	1, 3, 5	2	
1	0, 2	3, 5	
2	1	0	3, 5
3	0, 5	1	2
4	6		
5	0, 3	1	2
6	4		

[Table 1]

Problem

각 회원은 구매력을 가지고 있다. 구매력은 음이 아닌 정수이다.
각 회원의 영향력은 3촌 이내의 관계에 있는 회원들의 구매력을 통해 계산할 수 있는데, 그 식은 아래와 같다.
영향력 = (0촌의 구매력) * 10 + (1촌들의 구매력 총합) * 5 + (2촌들의 구매력 총합) * 3 + (3촌들의 구매력 총합) * 2

[Table 2]와 같이 3촌 이내의 관계와 구매력이 주어졌을 때, 각 회원의 영향력은 [Table 3]와 같다.

	1촌	2촌	3촌	구매력
0	1, 3, 5	2		2
1	0, 2	3, 5		1
2	1	0	3, 5	7
3	0, 5	1	2	12
4	6			15
5	0, 3	1	2	5
6	4			17

[Table 2]

	영향력 계산식	영향력
0	2 * 10 + (1 + 12 + 5) * 5 + 7 * 3	131
1	1 * 10 + (2 + 7) * 5 + (12 + 5) * 3	106
2	7 * 10 + 1 * 5 + 2 * 3 + (12 + 5) * 2	115
3	12 * 10 + (2 + 5) * 5 + 1 * 3 + 7 * 2	172
4	15 * 10 + 17 * 5	235
5	5 * 10 + (2 + 12) * 5 + 1 * 3 + 7 * 2	137
6	17 * 10 + 15 * 5	245

[Table 3]

Problem

친구 관계와 구매력이 수시로 추가되는 상황 속에서,
영향력이 높은 인플루언서들을 찾아내는 프로그램을 작성하고자 한다.

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

void init(int N, int mPurchasingPower[], int M, int mFriend1[], int mFriend2[])

각 테스트 케이스의 처음에 호출된다. 회원은 N명이다.

i 번 회원의 구매력은 mPurchasingPower[i]이다. (0 ≤ i ≤ N-1)

친구 관계는 M개이다.

mFriend1[i]번 회원과 mFriend2[i]번 회원은 서로 친구이다. (0 ≤ i ≤ M-1)

mFriend1[i]과 mFriend2[i]은 서로 다름이 보장된다.

Problem

void init(int N, int mPurchasingPower[], int M, int mFriend1[], int mFriend2[])

(계속)

동일한 친구 관계가 두 번 이상 주어지는 경우는 없음이 보장된다.

'1번 회원과 2번 회원의 친구 관계'와
'2번 회원과 1번 회원의 친구 관계'는 동일한 친구 관계이다.

각 회원의 친구는 15명 이하임이 보장된다.

각 회원과 3촌 이내의 관계에 있는 회원은 100명 이하임이 보장된다.

Parameters

N : 회원의 수 (3 ≤ N ≤ 20,000)

mPurchasingPower[i] : 구매력 (0 ≤ mPurchasingPower[i] ≤ 300) (0 ≤ i ≤ N-1)

M : 친구 관계의 수 (0 ≤ M ≤ 20,000)

mFriend1[i] : 회원 번호 (0 ≤ mFreind1[i] ≤ N-1) (0 ≤ i ≤ M-1)

mFriend2[i] : 회원 번호 (0 ≤ mFreind2[i] ≤ N-1) (0 ≤ i ≤ M-1)

Problem

int influencer(int mRank)

영향력이 mRank번째로 높은 회원의 고유 번호를 반환한다.

영향력이 같은 경우엔, 고유 번호가 낮은 회원의 영향력을 더 높은 것으로 취급한다.

mRank는 100 이하임이 보장된다.

Parameters

mRank : 찾아야 하는 인플루언서의 랭킹 (1 ≤ mRank ≤ min(100, N))

Return

영향력이 mRank번째로 높은 사람의 고유 번호

Problem

int addPurchasingPower(int mID, int mPower)

mID번 회원의 구매력을 mPower만큼 상승시킨다. mPower는 음이 아닌 정수이다.

mID의 영향력을 반환한다.

Parameters

mID : 회원 번호 (0 ≤ mID ≤ N-1)

mPower : 구매력 변화량 (0 ≤ mPower ≤ 100)

Return

mID의 영향력

Problem

int addFriendship(int mID1, int mID2)

mID1번 회원과 mID2번 회원은 서로 친구가 아님이 보장된다.

mID1과 mID2는 서로 다름이 보장된다.

mID1번 회원과 mID2번 회원을 서로 친구로 만든다.

mID1번 회원의 영향력과 mID2번 회원의 영향력의 합을 반환한다.

각 회원의 친구는 15명 이하임이 계속 보장된다.

각 회원과 3촌 이내의 관계에 있는 회원은 100명 이하임이 계속 보장된다.

Parameters

mID1 : 회원 번호 (0 ≤ mID1 ≤ N-1)

mID2 : 회원 번호 (0 ≤ mID2 ≤ N-1)

Return

mID1번 회원의 영향력과 mID2번 회원의 영향력의 합

Problem

[제약사항]

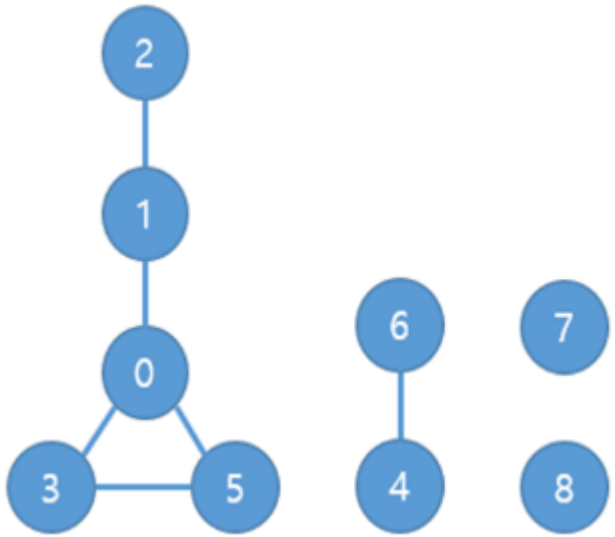
1. 각 테스트 케이스 시작 시 `init()` 함수가 호출된다.
2. 각 테스트 케이스에서 `influencer()` 함수의 호출 횟수는 `3,000` 이하이다.
3. 각 테스트 케이스에서 `addPurchasingPower()` 함수의 호출 횟수는 `3,000` 이하이다.
4. 각 테스트 케이스에서 `addFriendship()` 함수의 호출 횟수는 `100` 이하이다.

Problem analysis

Problem analysis : 예제

Order	Function	Return	ID	Purchasing power	Influence	Figure
1	init(9, {2, 1, 7, 12, 15, 5, 17, 6, 6}, 6, {0, 5, 3, 6, 1, 2}, {3, 0, 5, 4, 0, 1})		0	2	131	[Fig. 2]
			1	1	106	
			2	7	115	
			3	12	172	
			4	15	235	
			5	5	137	
			6	17	245	
			7	6	60	
			8	6	60	
2	influencer(8)	7				
3	influencer(9)	8				

[Table 4]

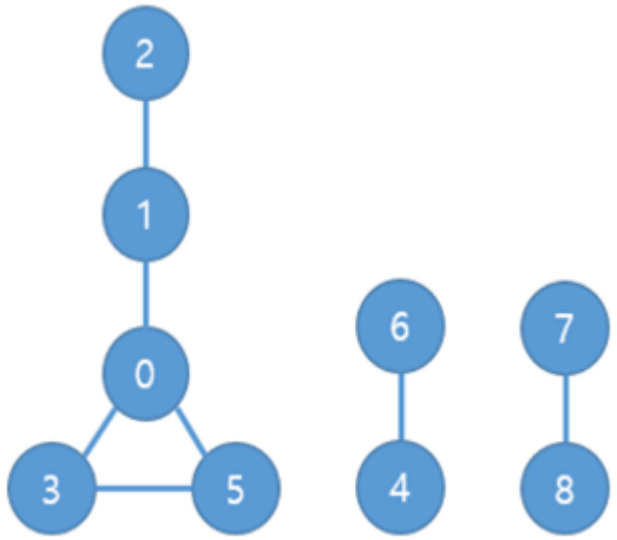


[Fig. 2]

Problem analysis : 예제

Order	Function	Return	ID	Purchasing power	Influence	Figure
4	addFriendship(7, 8)	180	0	2	131	[Fig. 3]
			1	1	106	
			2	7	115	
			3	12	172	
			4	15	235	
			5	5	137	
			6	17	245	
			7	6	90	
			8	6	90	
5	influencer(1)	6				
6	influencer(2)	4				
7	addPurchasingPower(2, 10)	215	0	2	161	
			1	1	156	
			2	17	215	
			3	12	192	
			4	15	235	
			5	5	157	
			6	17	245	
			7	6	90	
			8	6	90	
8	influencer(3)	2				
9	influencer(4)	3				

[Table 5]

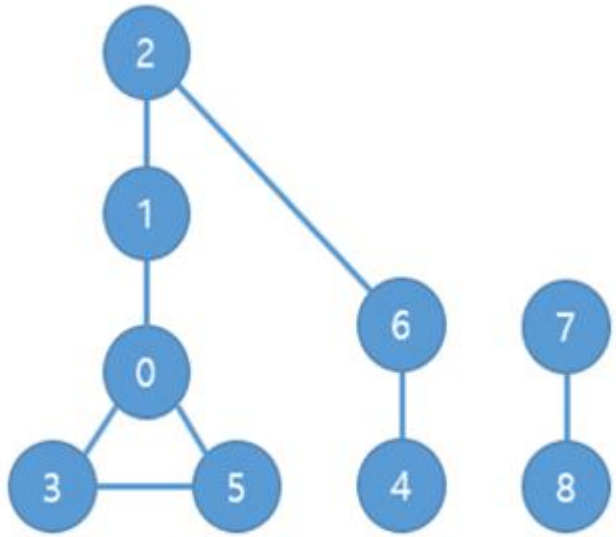


[Fig. 3]

Problem analysis : 예제

Order	Function	Return	ID	Purchasing power	Influence	Figure
10	addFriendship(2, 6)	682	0	2	195	[Fig. 4]
			1	1	237	
			2	17	345	
			3	12	192	
			4	15	288	
			5	5	157	
			6	17	337	
			7	6	90	
			8	6	90	
11	influencer(3)	4				
12	influencer(4)	1				
13	addPurchasingPower(7, 100)	1090	0	2	195	
			1	1	237	
			2	17	345	
			3	12	192	
			4	15	288	
			5	5	157	
			6	17	337	
			7	106	1090	
			8	6	590	
14	influencer(3)	2				
15	influencer(9)	5				

[Table 6]



[Fig. 4]

Problem analysis : 제약조건 및 API

- 그래프 문제이다.
- 그래프 표현을 위하여 인접 배열을 사용할 수 있다.
- 그래프 탐색을 위하여 BFS탐색을 사용할 수 있다.
  출발지로부터 최단거리로 방문해야 하기 때문이다.
  DFS를 사용하는 경우 최단거리 탐색이 간단하지 않거나 효율이 떨어질 수 있다.

- 회원의 등수를 관리해야 한다.
  - ✓ 랭크를 관리하면서 회원의 영향력이 바뀔 때마다 랭킹을 실시간으로 업데이트 할 수 있다.
  - ✓ 또는 회원의 영향력만을 업데이트 하고 랭킹을 묻는 쿼리가 주어질 때,
    구하는 방법을 사용할 수 있다.

Problem analysis : 제약조건 및 API

- 회원수는 최대 20,000이다.
- 간선 정보수는 최대 20,100이다.
  init():20,000번 + addFriendship():100번

- influencer(int mRank)의 경우
  1.영향력 내림차순, 2.id오름차순으로
  mRank번째 회원 id를 반환한다. => 최대 3,000 이하 호출

  - ✓ 영향력 점수만 업데이트하여 관리하는 경우
    nth_element를 이용한다면 O(20,000 * 3000 = 60,000,000)에 답할 수 있다.
  - ✓ 랭킹을 실시간 관리한다면 O(100:mRank * 3000 = 300,000)에 답할 수 있다.
  - ✓ 어느 방법이든 시간안에 실행이 가능할 것으로 생각된다.

Problem analysis

- addPurchasingPower(int mID, int mPower)의 경우
  mID회원의 구매력을 mPower만큼 상승시키는데
  mID와 3레벨 이하친구들의 영향력에 영향을 미친다.
  따라서 이들도 업데이트가 필요하다. => 최대 3,000 이하 호출

  - ✓ 영향력만 업데이트 하는 경우 mID와 관련된 모든 회원 id에 대하여 업데이트하는데
    O(100:친구수 * 100:인접노드수 *3000:함수호출 = 30,000,000) 걸린다.
  - ✓ 랭킹을 실시간 관리한다면 mID와 관련된 모든 회원 id에 대하여 랭킹을 업데이트하는데
    O(100 * 100 * log(20000)*3000 =
    100:친구수 * 100:인접노드수 *15:트리높이*3000:함수호출 = 450,000,000) 걸린다.
  - ✓ 어느 방법이든 시간안에 실행이 가능할 것으로 생각된다.

Problem analysis

- addFriendship(int mID1, int mID2)의 경우
  mID1, mID2와 친구레벨 3이하 친구들의 영향력에 영향을 미친다.
  따라서 이들도 업데이트가 필요하다. => 100이하 호출

  - ✓ 영향력만 업데이트 하는 경우 mID1, mID2와 관련된 모든 회원 id에 대하여 업데이트하는데
    O(100:친구수 * 100:인접노드수 * 100:함수호출수 = 1,000,000) 걸린다.
  - ✓ 랭킹을 실시간 관리한다면 mID1, mID2 와 관련된 모든 회원 id에 대하여
    랭킹을 업데이트하는데
    O(100:친구수 *15:트리높이* 100:인접노드수 * 100:함수호출수 = 15,000,000) 걸린다.
  - ✓ 어느 방법이든 시간안에 실행이 가능할 것으로 생각된다.

Solution sketch

Solution sketch

- 랭킹을 실시간으로 관리하는 버전을 생각해보자.
- 각회원은 다음과 같은 속성을 갖는다.

```
int influ[LM];          // id별 영향력
int power[LM];          // id별 힘(구매력)
vector<int> adj[LM]; // 인접 배열
```

- 등수를 실시간으로 관리하기 위하여 set을 사용할 수 있다.
  우선 1순위를 저장할 때, 음수로 저장한다면 별도의 functor를 선언하지 않아도 된다.
- 따라서 pair<-int, int> 형태로 저장하여 사용한다.

```
set<pii> ranking;  // <int: 1. influ[i]의 내림차순, int: 2. id의 오름차순>
```

Solution sketch

- mID와 3레벨 이하인 친구를 방문하기 위하여 BFS함수를 사용할 수 있다.
- 업데이트 시 유의 해야 할 것은 set에 대하여 제거->변경->추가 한다는 것이다.

```
struct Data {
    int node, lev;  // node: 회원 id, lev: 친구 레벨
}que[1005];
int fr, re;

void bfs(int src, int delta = 0) {        // 영향력 업데이트 BFS
    fr = re = 0, ++vn;
    ranking.erase({ influ[src], src });  // 1. 제거하고

    que[re++] = { src, 0 };
    visited[src] = vn;
    int infSum = 0;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        infSum += wei[lev] * power[node];// 레벨별 영향력 점수 구하기
        if (lev > 2) continue;
        for (int child : adj[node]) if (visited[child] < vn) {
            visited[child] = vn;
            que[re++] = { child, lev + 1 };
        }
    }
    influ[src] = -infSum;                 // 2. 변경하기
    ranking.insert({ influ[src], src }); // 3. 추가하기
}
```

Solution sketch

- mID회원의 친구레벨 3이하인 모든 친구를 구하기 위하여
  다음과 같은 getID() 함수를 사용할 수 있다. 또 다른 BFS함수이다.

```
unordered_set<int> A;     // src의 친구레벨 3이하 친구들의 집합
void getID(int src) {     // src 의 친구레벨 3이하인 모든 친구 구하기 BFS
    fr = re = 0;
    ++vn;
    que[re++] = { src, 0 }, visited[src] = vn;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        A.insert(node);   // src의 친구목록에 추가
        if (lev > 2) continue;
        for (int child : adj[node]) if (visited[child] < vn) {
            visited[child] = vn;
            que[re++] = { child, lev + 1 };
        }
    }
}
```

Solution sketch

- **이제 각 함수별 할 일을 정리해 보자.**

void init(int N, int mPurchasingPower[], int M, int mFriend1[], int mFriend2[])

✓ **랭킹등 전역변수를 초기화한다.**

```
::N = N;
ranking.clear();
```

✓ **각 회원 id별 인접배열을 구성한다.**

```
for (int i = 0; i < M; ++i) {          // 인접 배열 구성하기
    int a = mFriend1[i], b = mFriend2[i];
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

✓ **각 회원별로 bfs()함수를 이용하여 회원별 영향력을 구한다.**

```
for (int i = 0; i < N; ++i)          // 각 회원별 영향력 구하기
    bfs(i);
```

Solution sketch

int influencer(int mRank)

- ✓ **이미 정렬되어 있으므로 다음과 같이 간단히 작성할 수 있다.**

```
return next(ranking.begin(), mRank-1)->second;
```

int addPurchasingPower(int mID, int mPower)

- ✓ **mID와 레벨 3이하인 친구들을 구한다.**
- ✓ **mID의 구매력을 mPower만큼 증가시키고 자신과 친구를 순회하며 bfs()함수를 이용하여 영향력과 랭킹을 업데이트 한다.**

```
A.clear();              // 친구 목록 초기화
getID(mID);             // 친구 구하기
power[mID] += mPower;
for (int a : A)
    bfs(a);             // 자신과 친구들의 영향력과 랭킹 업데이트하기
return - influ[mID]; // mID의 영향력 반환하기. 음수로 저장하였으므로 양수로 변환하여 반환한다.
```

Solution sketch

int addFriendship(int mID1, int mID2)

- ✓ mID1과 mID2에 영향을 받는 회원 id 구하기
- ✓ 인접배열 업데이트
- ✓ 자신과 친구들에 대하여
  bfs()함수를 이용하여 영향력과 랭킹을 업데이트 한다.

```
A.clear();
getID(mID1);
getID(mID2);                          // mID1과 mID2에 영향을 받는 회원 id 구하기
adj[mID1].push_back(mID2);
adj[mID2].push_back(mID1);
for (int a : A) {                     // A.size()의 최대 크기는 200
    bfs(a);                           // 각 친구의 영향력 구하기
}
return - influ[mID1] - influ[mID2];   // 영향력의 합 반환하기
```

Solution sketch

[Summary]

- 그래프를 표현할 수 있다.

- 그래프를 탐색할 수 있다.
  BFS를 문제해결에 사용할 수 있다.

- set을 이용하여 랭킹을 실시간으로 관리할 수 있다.

- unordered_set을 이용하여
  순서가 필요 없는 중복되지 않는 데이터를 관리할 수 있다.

Code example1

: vector, unordered_set

: nth_element

Code example1

```cpp
#include <vector>
#include <unordered_set>
#include <algorithm>
using namespace std;

const int LM = 20005;
int influ[LM];                  // id별 영향력
int power[LM];                  // id별 힘
vector<int> adj[LM];            // 인접 배열
int N, ids[LM];                 // 회원수, 회원 id
int visited[LM], vn;            // 방문체크
int wei[] = {10, 5, 3, 2};      // 친구 레벨별 가중치

struct Comp {
    bool operator()(const int&a, const int&b)const {
        return influ[a] == influ[b]? a<b: influ[a]>influ[b];
    }
};
```

Code example1

```
struct Data {
    int node, lev;  // node: 회원 id, lev: 친구 레벨
}que[1005];
int fr, re;

// 두 가지 형태로 영향력을 업데이트 한다.
void bfs(int src, int delta = -1) {
    fr = re = 0,  ++vn;
    que[re++] = { src, 0 };
    visited[src] = vn;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        if(delta < 0) influ[src] += wei[lev] * power[node]; // 가져오기 : init(), addFriendship()
        else influ[node] += wei[lev] * delta;               // 뿌려주기 : addPurchasingPower()
        if (lev > 2) continue;
        for (int child : adj[node]) if(visited[child] < vn) {
            visited[child] = vn;
            que[re++] = {child, lev + 1};
        }
    }
}
```

Code example1

```
void init(int N, int mPurchasingPower[20000], int M, int mFriend1[20000], int mFriend2[20000]) {
    ::N = N;
    for (int i = 0; i < N; ++i) {       // 초기화
        influ[i] = 0, ids[i] = i;
        adj[i].clear();
        power[i] = mPurchasingPower[i];
    }

    for (int i = 0; i < M; ++i) {       // 인접 배열 구성하기
        int a = mFriend1[i], b = mFriend2[i];
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    for (int i = 0; i < N; ++i)         // 각 회원별 영향력 구하기
        bfs(i);
}
```

Code example1

```
int influencer(int mRank) {                 // O(N)
    nth_element(ids, ids + mRank - 1, ids + N, Comp());  // mRank번째 회원 번호 구하기
    return ids[mRank - 1];
}

int addPurchasingPower(int mID, int mPower) {
    if (mPower == 0) return influ[mID]; // 증가시킬 구매력이 없는경우
    bfs(mID, mPower);                   // 구매력을 mPower만큼 증가시키기
    power[mID] += mPower;
    return influ[mID];
}

unordered_set<int> A;
void getID(int node) {      // node1과 node2 의 친구레벨 3이하인 모든 친구 구하기
    fr = re = 0, ++vn;
    que[re++] = { node, 0 }, visited[node] = vn;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        A.insert(node);
        if (lev > 2) continue;
        for (int child : adj[node]) if (visited[child] < vn) {
            visited[child] = vn;
            que[re++] = { child, lev + 1 };
        }
    }
}
```

Code example1

```
int addFriendship(int mID1, int mID2) {  // O(노드수*인접노드수*함수호출수 = (100+100)*100*100)
    A.clear();
    getID(mID1);                         // mID1에 영향을 받는 회원 id 구하기
    getID(mID2);                         // mID2에 영향을 받는 회원 id 구하기
    adj[mID1].push_back(mID2);
    adj[mID2].push_back(mID1);
    for (int a : A) {                    // A.size()의 최대 크기는 200
        influ[a] = 0;                    // a의 친구 수는 최대 100
        bfs(a);                          // 각 친구의 영향력 구하기
    }
    return influ[mID1] + influ[mID2];
}
```

Code example2

: vector, set,
unordered_set

Code example2

```
#include <set>
#include <vector>
#include <unordered_set>
#include <algorithm>
using namespace std;

using pii = pair<int, int>;
const int LM = 20005;
int influ[LM];                  // id별 영향력
int power[LM];                  // id별 힘
vector<int> adj[LM];            // 인접 배열
int N ;                         // 회원수
int visited[LM], vn;            // 방문체크
int wei[] = { 10, 5, 3, 2 };    // 친구 레벨별 가중치
set<pii> ranking;               // <int: 1. influ[i]의 내림차순, int: 2. id의 오름차순>

struct Data {
    int node, lev;  // node: 회원 id, lev: 친구 레벨
}que[1005];
int fr, re;
```

Code example2

```
void bfs(int src) {                          // 영향력 업데이트 BFS
    fr = re = 0, ++vn;
    ranking.erase({ influ[src], src });  // 1. 제거하고

    que[re++] = { src, 0 };
    visited[src] = vn;
    int infSum = 0;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        infSum += wei[lev] * power[node];// 레벨별 영향력 점수 구하기
        if (lev > 2) continue;
        for (int child : adj[node]) if (visited[child] < vn) {
            visited[child] = vn;
            que[re++] = { child, lev + 1 };
        }
    }
    influ[src] = -infSum;
    ranking.insert({ influ[src], src }); // 3. 추가하기
}
```

Code example2

```
unordered_set<int> A;
void getID(int src) {                    // src 의 친구레벨 3이하인 모든 친구 구하기 BFS
    fr = re = 0;
    ++vn;
    que[re++] = { src, 0 }, visited[src] = vn;
    while (fr < re) {
        int node = que[fr].node, lev = que[fr++].lev;
        A.insert(node);
        if (lev > 2) continue;
        for (int child : adj[node]) if (visited[child] < vn) {
            visited[child] = vn;
            que[re++] = { child, lev + 1 };
        }
    }
}
```

Code example2

```
void init(int N, int mPurchasingPower[20000], int M, int mFriend1[20000], int mFriend2[20000]) {
    ::N = N;
    ranking.clear();
    for (int i = 0; i < N; ++i) {          // 초기화
        influ[i] = 0;
        adj[i].clear();
        power[i] = mPurchasingPower[i];
    }

    for (int i = 0; i < M; ++i) {          // 인접 배열 구성하기
        int a = mFriend1[i], b = mFriend2[i];
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    for (int i = 0; i < N; ++i)            // 각 회원별 영향력 구하기
        bfs(i);
}

int influencer(int mRank) {                // O(mRank - 1)
    return next(ranking.begin(), mRank-1)->second;
}
```

Code example2

```
int addPurchasingPower(int mID, int mPower) {  // O( 100 * 100 * 3000)
    if (mPower == 0) return - influ[mID]; // 증가시킬 구매력이 없는경우
    A.clear();
    getID(mID);
    power[mID] += mPower;
    for (int a : A)
        bfs(a);
    return - influ[mID];
}

int addFriendship(int mID1, int mID2) {  // O(노드수*인접노드수*함수호출수 = (100+100)*100*100)
    A.clear();
    getID(mID1);
    getID(mID2);                         // mID1과 mID2에 영향을 받는 회원 id 구하기
    adj[mID1].push_back(mID2);
    adj[mID2].push_back(mID1);
    for (int a : A) {                    // A.size()의 최대 크기는 200
        bfs(a);                          // 각 친구의 영향력 구하기
    }
    return - influ[mID1] - influ[mID2];
}
```

Thank you.