Praktikum 1 prob wdyt

DESKRIPSI

Dayat adalah seorang hengker dari kerajaan mbetoyo. Dia diminta oleh raja mbetoyo untuk ngehack facebook milik mark zuckerberg. Namun raja mbetoyo tidak lepas tangan begitu saja, dia juga menyediakan beberapa anak buah untuk menjalankan misi ini. Selain jago ngehek Dayat juga bisa membaca pikiran orang lain. Karena hal tersebut Dayat dapat memperkirakan skill anak buah yang disediakan oleh raja dengan sebuah bilangan bulat. Untuk menyelesaikan permintaan dari raja, Dayat membagi anak buah tersebut ke beberapa grup dengan ketentuan:

1. Semua tingkat kemampuan dari anak buah dalam satu grup pasti unik
2. Setiap anak buah memiliki kemampuan yang bukan kemampuan terkecil dalam grup, pasti memiliki satu tingkat kemampuan dibawahnya dalam satu grup.

Bantulah dayat untuk menentukan banyaknya jumlah anggota grup terkecil yang dibentuk, namun terbesar dari semua kemungkinan.

FORMAT MASUKAN

Baris pertama berisi bilangan Q yang menyatakan banyaknya N, dimana N merupakan banyak anak buah dan diikuti sebaris bilangan N yang menyatakan kemampuan dari setiap anak buah.

FORMAT KELUARAN

Menampilkan banyaknya jumlah anggota grup terkecil yang dibentuk, namun terbesar dari semua kemungkinan pada setiap bilangan N yang diberikan dan dipisahkan dengan enter.

BATASAN

$1 < Q < 100$

$1 \leq N \leq 10^5$

$-10^5 \leq kemampuan\ setiap\ anak\ buah \leq 10^5$

CONTOH MASUKAN DAN KELUARAN

Contoh Masukan 1

```
2
9 -5 -4 -3 -3 -2 2 3 4 5
5 5 4 3 2 1
```

Contoh Keluaran 1

```
2

5
```

Penjelasan:

Dari test case pertama dapat dibentuk grup sebagai berikut:

{-5,-4,-3}, {-3,-2}, {2, 3, 4 ,5}

Jadi banyak jumlah anggota grup yang terkecil namun terbesar dari semua kemungkinan adalah 2.

kode dan penjelasannya

```cpp
#include <iostream>
#include <queue>
#include <algorithm>

using namespace std;

int queryNya, num;
//Mendeklarasi banyak num (queryNya) dan banyak anak buat (num)
int vp [10005];
//Mendeklarasi kemampuan setiap anak buah
queue<int> team;
//Mendeklarasi team dalam bentuk queue

int main(void) {
    cin >> queryNya;
    while (queryNya--) {
    //Melakukan perulangan sebanyak queryNya
        int hasil = 0;
        //Menginisialisasi hasil = 0 yang akan menyimpan ukuran minimum tim
        cin >> num;
        for (int i = 0; i < num; i++) {
            cin >> vp[i];
        }

        sort(vp, vp+num);
        //Mengurutkan elemen-elemen array vp dalam urutan menaik menggunakan fungsi sort dari library <algorithm>

        int temp = 0;
        while (temp < num) {
        //Mengiterasi sampai temp = atau > dari num
```

```
int start = temp;
//Start mewakili indeks awal dari sekelompok anak buah dengan tingkat kemampuan yang sama
int skill = vp[temp];
//Menyimpan tingkat kemampuan tim tersebut

while (temp < num && vp[temp] == skill) {
    temp++;
}
//Perulangan ini mengincrement temp hingga akhir array atau menemukan elemen dalam vp yang tidak sama dengan skill
//Digunakan untuk melewati sekelompok anak buah dengan tingkat kemampuan yang sama

int anggotaTim = temp - start;
//Menghitung jumlah anak buah dalam tim saat ini dengan mengurangkan indeks awal (start) dari indeks saat ini (temp)
if (anggotaTim >= team.size()) {
    int diff = anggotaTim - team.size();
    for (int i = 0; i < diff; i++) {
        team.push(skill);
    }
}
//Jika anggota tim >= ukuran queue team, jumlah nilai skill yang berbeda akan dimasukkan ke dalam queue team
//Memastikan bahwa ukuran queue team sesuai dengan jumlah pemain dalam tim saat ini

if (anggotaTim < team.size()) {
    int diff = team.size() - anggotaTim;
    for (int i = 0; i < diff; i++) {
        int skillstart = team.front();
        team.pop();
        int sizeofteam = skill - skillstart;
        if (!hasil || sizeofteam < hasil){
            hasil = sizeofteam;

                }
            }
        }
        //Jika anggota tim < ukuran queue team, jumlah elemen yang berbeda dari depan queue tim akan dimunculkan
        //Menghitung perbedaan level skill antara skill dan elemen yang muncul (skillstart)
        //Memperbarui hasil jika hasilnya nol atau lebih kecil dari perbedaan saat ini

        if (vp[temp] - skill > 1 || temp == num) {
            while (!team.empty()) {
                int skillstart = team.front();
                team.pop();
                int sizeofteam = skill - skillstart + 1;
                if (!hasil || sizeofteam < hasil){
                    hasil = sizeofteam;
                }cout << hasil << endl;
            }
        }
        //Jika perbedaan antara tingkat kemampuan anggota berikutnya (vp[temp]) dan anggota saat ini lebih besar dari 1
        //Atau jika indeks saat ini (temp) berada di akhir array, semua elemen dari antrean tim akan dimunculkan
        //Menghitung selisih level skill antara skill dengan elemen yang dimunculkan (skillstart) ditambah 1
        //Memperbarui hasil jika hasilnya nol atau lebih kecil dari selisih yang ada
        //Lalu mengoutputkan ukuran tim minimum (hasil) untuk query saat ini
    }
}
```

sourcecode:
https://ideone.com/YVOq27

Praktikum 4 prob mk

Description

Pada suatu zaman, di sebuah kerajaan yang damai, hiduplah seorang raja bijaksana yang sangat mencintai rakyatnya. Namun, takdir tidak berpihak pada mereka ketika gunung berapi berkobar di pulau tempat mereka tinggal. Lava yang menghancurkan pulau itu membuat mereka terpaksa mencari tempat baru untuk dihuni.

Raja dan rakyatnya melakukan perjalanan panjang melintasi lautan dalam mencari pulau baru yang layak. Setelah beberapa minggu berlayar, mereka menemukan pulau yang indah dengan alam yang subur. Pulau ini memberi mereka harapan baru dan peluang untuk membangun kehidupan yang lebih baik.

Setelah sampai di pulau baru, raja dan rakyatnya dengan giat membangun tempat tinggal mereka. Rakyat kerajaan ini memiliki keistimewaan, yaitu kemampuan untuk mengendalikan elemen seperti air, api, angin, dan lainnya. Kemampuan ini membuat mereka terhubung erat dengan alam dan memberi mereka kekuatan unik.

Namun, raja memiliki visi yang lebih besar. Dia ingin membentuk kerajaan yang terorganisir dengan baik di pulau baru ini. Untuk itu, dia memutuskan untuk memetakan kota-kota mereka seperti sebuah grafik. Setiap kota yang bersebelahan di dalam grafik ini tidak boleh memiliki elemen yang sama.

Raja berkumpul dengan penasihatnya dan para ahli untuk merancang rencana pemetaan. Mereka mengidentifikasi elemen-elemen yang dimiliki oleh setiap kota dan menempatkannya dalam grafik dengan hati-hati. Sebagai penasihat raja, beritahu raja apakah rancangan kota yang telah dibuat raja bisa membentuk harmoni atau tidak.

Input Format

- Baris pertama berisi K, E yang merepresentasikan jumlah Kota yang akan dibangun dan jumlah Elemen yang ada secara berturut-turut.
- Untuk K baris berikutnya, akan berisi k yang merepresentasikan kota dan kota-kota tetangga setelah tanda panah yang dipisah menggunakan koma.
- kota selalu direpresentasikan dengan 1 karakter dengan pengecualian :
  - `{space} ' '`
  - `{comma} ','`
  - `{minus} '-'`
  - `{greater than} '>'`

Constraints

$1 \leq N < 100$; $1 \leq E \leq 5$

Output Format

outputkan `YES!` bila mungkin, dan `NO!` bila tidak

Sample Input 1

```
5 3
A -> B, C
B -> A, C, D
C -> A, B, D, E
D -> B, C, E
E -> C, D, A
```

Sample Output 1

```
YES!
```

Sample Input 2

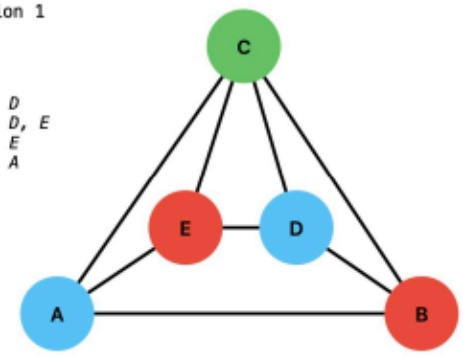
```
6 2
A -> B, C
B -> A, C, D
C -> A, B, E
D -> B, E
E -> C, D
F ->
```

Sample Output 2

```
NO!
```

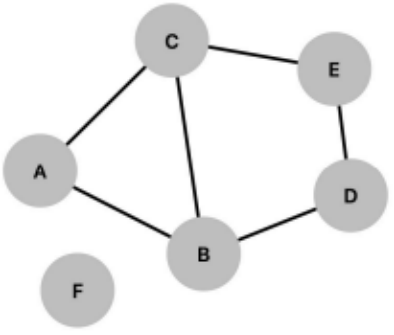
Explanation 1

```
5 3
A -> B, C
B -> A, C, D
C -> A, B, D, E
D -> B, C, E
E -> C, D, A
```



Explanation 2

```
6 2
A -> B, C
B -> A, C, D
C -> A, B, E
D -> B, E
E -> C, D
F ->
```



※ elemen dimisalkan dengan menggunakan warna yang berbeda

※ Explanation 1: peletakan elemen tidak diperhatikan, selama memenuhi keinginan raja yaitu setiap kota tidak boleh memiliki elemen yang sama

※ Explanation 2: tidak mungkin dibuat dengan 2 elemen yang ada

kode dan penjelasannya

```
#include <iostream>
#include <map>
#include <vector>
#include <string>

using namespace std;

bool element_cond(vector<vector<int>> &adjList, int elemen_arr[], int kota)
{
    for(int i = 0; i < kota; i++)
    {
        if(!adjList.empty())
        {
            for (auto k : adjList[i])
                if (elemen_arr[i] == elemen_arr[k]) return false;
        }
   }
   return true;
}
//Memeriksa apakah sebuah kondisi terpenuhi untuk elemen-elemen di elemen_arr berdasarkan daftar kedekatan adjList
/*Dibutuhkan tiga parameter, yaitu adjList: mereference ke sebuah vektor berisi vektor-vektor int yang merepresentasikan daftar kedekatan,
   elemen_arr: sebuah array int, dan kota: sebuah int yang merepresentasikan jumlah elemen dalam elemen_arr */
//Fungsi ini mengulang setiap elemen dalam elemen_arr dan memeriksa apakah elemen-elemen yang bersebelahan dalam adjList memiliki nilai yang sama
//Jika ditemukan kecocokan, fungsi akan mengembalikan nilai false, yang mengindikasikan bahwa kondisi tersebut tidak terpenuhi
//Jika tidak ada kecocokan yang ditemukan, fungsi akan mengembalikan nilai true, yang mengindikasikan bahwa kondisi terpenuhi.

bool check_element(vector<vector<int>> &adjList, int elemen, int index, int elemen_arr[], int kota)
{
    if (index == kota)
    {
        if (element_cond(adjList, elemen_arr, kota)) return true;
        return false;
    }
    for (int j = 1; j <= elemen; j++){
        elemen_arr[index] = j;
        if (check_element(adjList, elemen, index + 1, elemen_arr, kota))
            return true;
        elemen_arr[index] = 0;
    }
    return false;
}
//Fungsi ini secara rekursif memeriksa semua kemungkinan kombinasi elemen di elemen_arr untuk menemukan penugasan yang valid yang memenuhi kondisi yang diberikan
/*Dibutuhkan lima parameter, yaitu adjList: mereference ke sebuah vektor vektor int, yang mewakili daftar kedekatan,
    elemen: sebuah int yang merepresentasikan jumlah nilai yang mungkin untuk setiap elemen dalam elemen_arr,
    index: sebuah int yang merepresentasikan indeks saat ini yang sedang diproses dalam elemen_arr, elemen_arr: sebuah array int,
    dan kota: sebuah int yang merepresentasikan jumlah elemen dalam elemen_arr.*/
//Fungsi ini menggunakan pelacakan mundur untuk mengeksplorasi semua nilai yang mungkin untuk setiap indeks dalam elemen_arr
//Memberikan nilai pada elemen_arr[indeks], secara rekursif memanggil dirinya sendiri untuk indeks berikutnya, dan mengulangi proses ini hingga semua indeks diberikan nilai
//Jika indeks mencapai elemen terakhir (indeks == kota), akan dimeriksa apakah penugasan saat ini memenuhi kondisi dengan memanggil elemen_cond
//Jika ya, mengembalikan nilai true; jika tidak, mengembalikan nilai false
//Dalam panggilan rekursif, untuk setiap indeks, panggilan ini mengulang semua nilai yang mungkin (1 sampai elemen) dan memberikan nilai tersebut ke elemen_arr[indeks]
/*Kemudian, dilakukan panggilan rekursif untuk memproses indeks berikutnya (indeks + 1)
    Jika penugasan yang valid ditemukan dalam panggilan rekursif, akan mengembalikan nilai true
    Jika tidak, akan mereset nilai elemen_arr[index] ke 0 dan melanjutkan iterasi berikutnya*/
//Jika tidak ada panggilan rekursif yang menghasilkan nilai true, fungsi ini akan menghasilkan nilai false

int main()
{
    map<char, int> id;
    char capital = 'A';

    for(int i=0; i<26; i++)
    {
        id[capital] = i;
        capital++;
    }
    //Menginisialisasi id dengan bentuk map yang memetakan char ke int (A sampai Z),
    //di mana setiap char dikaitkan dengan indeks int yang unik

    int kota, elemen;
    cin >> kota >> elemen;
    int elemen_arr[kota];
    //Mendeklarasikan array elemen_arr dengan ukuran kota

    vector<vector<int>> adjList(kota);
    //Membuat sebuah daftar kedekatan adjList sebagai sebuah vektor dari vektor-vektor int,
    //di mana setiap vektor merepresentasikan elemen-elemen yang berdekatan untuk sebuah elemen tertentu

    for(int i=0; i<=kota; i++)
    {
        string input;
        getline(cin, input);
        int start = id[input[0]];
        vector<int> temp;
        auto greaterThan = input.find('>');

        for (int j = greaterThan + 2; j < input.size(); j++)
        {
            if (input[j] != ' ' && input[j] != ',')
                temp.push_back(id[input[j]]);
        }
```

```
        if (temp.size() != 0)
            for (auto i : temp) adjList[start].push_back(i);
    }
    /*Perulangan untuk menginput setiap elemen dan mengisi daftar kedekatan yang sesuai
        Membaca sebuah baris input menggunakan getline(cin, input) dan memprosesnya sebagai berikut:
        Menentukan elemen start dengan mencari indeks dari karakter pertama dalam peta id
        Membuat vektor sementara temp untuk menyimpan elemen-elemen yang berdekatan
        Mencari posisi karakter > di baris masukan
        Mengulang karakter setelah karakter > dan memeriksa apakah karakter tersebut valid (bukan spasi atau koma)
        Jika valid, ia akan mencari indeksnya di peta id dan menambahkannya ke vektor temp
        Terakhir, jika temp tidak kosong, program akan menambahkan elemen-elemen dari temp ke daftar kedekatan untuk elemen start saat ini */

    if (!check_element(adjList, elemen, 0, elemen_arr, kota)) cout << "NO!" << endl;
    else cout << "YES!" << endl;
    /*Setelah memproses input, program memanggil fungsi cek_elemen dengan argumen yang sesuai (adjList, elemen, 0, elemen_arr, kota)
        Jika fungsi tersebut menghasilkan nilai true, akan mengoutputkan "YA!" untuk mengindikasikan bahwa penugasan yang valid dapat dilakukan
        Jika tidak, fungsi akan mencetak "TIDAK!" untuk mengindikasikan bahwa tidak ada penugasan yang valid */

    return 0;
}
```

sourcecode:
https://ideone.com/MhKLOu

Praktikum 4 prob p

Peta

Diberikan peta berupa bit berbentuk persegi panjang dengan size n * m. Setiap pixel pada peta tersebut berwarna putih atau hitam. Setidaknya ada satu pixel yang berwarna putih. Piksel pada baris ke-I dan kolom ke-j disebut piksel (i, j). Jarak antara dua pixel $p_1 = (i_1, j_1)$ dan $p_2 = (i_2, j_2)$ yang didefinisikan sebagai

$$d(p_1, p_2) = |i_1 - i_2| + ¿ j_1 - j_2 \vee ¿$$

Job kita adalah membuat program yang dapat membaca deskripsi dari peta berupa bit berdasarkan inputan. Kemudian, untuk setiap pixel, hitung jarak ke pixel putih yang terdekat. Lalu, berikan hasilnya di standard output.

Input Format

Banyaknya testcase disebutkan pada baris pertama. Baris kedua berisi 2 angka yang dipisahkan dengan spasi. Kita sebut sebagai n dan m dengan ketentuan $1 \le n, m \le 182$. Di setiap n baris pada tiap testcase, terdapat one zero- one word dari panjang m (jadi 1 kalimat m bit, banyaknya bit adalah m). Pada posisi j di baris i, $1 \le i \le n, 1 \le j \le m$ dengan ketentuan bahwa dapat bernilai 1 jika dan hanya jika pixel (i, j) adalah putih.

Output Format

Di setiap baris ke-i untuk masing-masing testcase, $1 \le i \le n$, harus terdapat m integers $f(i,1), \ldots, f(i,m)$ yang dipisahkan oleh single spaces, dimana $f(i,j)$ adalah jarak dari pixel (i, j) ke white pixel terdekat.

Sample input:

```
1
3 4
0001
0011
0110
```

Sample output:

```
3 2 1 0
2 1 0 0
1 0 0 1
```

kode dan penjelasannya

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

int main(){
    int i, j;
    //Mendeklarasikan dua variabel int i dan j yang akan digunakan untuk perulangan loop

    int test_case;
    cin >> test_case;
    //Menerima input dari user dan menyimpannya dalam variabel test_case, merepresentasikan jumlah kasus uji yang akan dieksekusi

    while(test_case--){
        //Perulangan sebanyak jumlah test_case
        int n, m;
        cin >> n >> m;
        //Menerima input n dan m dari user, kedua int tersebut merepresentasikan ukuran dari peta

        vector<vector<int>> map(n, vector<int>(m));
        //Mendeklarasikan vektor 2D bernama map dengan dimensi n x m
        //Merepresentasikan peta di mana setiap elemennya bisa bernilai 0 atau 1

        for(i = 0; i < n; i++){
            string baris;
            cin >> baris;

            for(j = 0; j < m; j++){
                map[i][j] = baris[j] - '0';
    }
}
//Membaca n baris input dari user, di mana setiap baris mewakili satu baris peta
//Setiap karakter dari string input dikonversi menjadi bilangan bulat (0 atau 1) dan disimpan dalam elemen yang sesuai dari vektor map

vector<vector<int>> jarak(n, vector<int>(m, INT_MAX));
//Mendeklarasikan vektor 2D lain bernama jarak dengan dimensi n x m
//Vektor ini diinisialisasi dengan INT_MAX, yang merepresentasikan jarak awal yang tak terbatas

vector<pair<int, int>> pixel_putih;
//Mendeklarasikan sebuah vektor pasangan int bernama pixel_putih, akan digunakan untuk menyimpan posisi piksel putih pada peta

for(i = 0; i < n; i++){
    for(j = 0; j < m; j++){
        if(map[i][j] == 1){
            jarak[i][j] = 0;
            pixel_putih.push_back({i, j});
        }
    }
}
//Perulangan bersarang ini mengulang setiap elemen vektor peta
//Jika piksel putih (1) ditemukan, jarak yang sesuai dalam vektor jarak diatur ke 0, dan posisi piksel putih ditambahkan ke vektor piksel_putih

for(i = 0; i < n; i++){
    for(j = 0; j < m; j++){

        if(map[i][j] == 0){
            for(const auto& pixel : pixel_putih){

                int distance = abs(i - pixel.first) + abs(j - pixel.second);
                    jarak[i][j] = min(jarak[i][j], distance);
                }
            }
        }
        //Perulangan bersarang ini mengulang setiap elemen vektor peta
        //Jika piksel hitam (0) ditemukan, akan dihitung jarak antara piksel hitam tersebut dengan semua piksel putih yang tersimpan dalam vektor piksel_putih
        //Jarak minimum kemudian disimpan dalam elemen yang sesuai dari vektor jarak

        for(i = 0; i < n; i++){
            for(j = 0; j < m; j++){
                cout << jarak[i][j] << " ";
            }
            cout << endl;
        }
    }
    //Perulangan bersarang ini mengulang setiap elemen vektor jarak dan mencetak nilainya
    //Nilai dipisahkan oleh spasi dan dicetak baris baru untuk setiap baris
    return 0;
}
```

sourcecode:
https://ideone.com/tOjqXJ

Praktikum 4 prob sr

DESKRIPSI SOAL

Ray, seorang mahasiswa yang bersemangat dalam menjalankan riset skripsinya, memusatkan perhatiannya pada kelompok-kelompok pertemanan di kampus. Setelah refleksi yang panjang, ia memeluk hipotesis yang berani, menggugah pikirannya dengan teori yang menarik.

Hipotesis Ray menyatakan bahwa setiap individu hanya memiliki kemungkinan menjadi **introvert** atau **extrovert**, dan pertemanan yang baik hanya terjadi jika pasangan pertemanan memiliki **tipe kepribadian yang berbeda**. Menurutnya, perbedaan tersebut saling melengkapi satu sama lain dan menciptakan sinergi yang menakjubkan. Lebih lanjut, ia juga berpendapat bahwa pertemanan sejati hanya terbentuk melalui interaksi **dua individu**, tidak lebih. Ray percaya bahwa melalui interaksi tersebut, kedua individu dapat mengenal satu sama lain dengan lebih dalam.

Setiap mahasiswa di kampus Ray memiliki Nomor Pokok (**NRP**), yang merupakan nomor identitas unik bagi setiap mahasiswa. Sebagai asisten pribadi Ray, tugas kamu adalah membantunya menjalankan penelitian ini. Kamu akan diberikan daftar mahasiswa dalam suatu kelompok, dan tugasmu adalah menentukan apakah hasil penelitian ini mendukung hipotesis Ray atau tidak.

INPUT

Pada baris pertama terdapat bilangan **x**, yang merupakan **jumlah penelitian** yang akan dilakukan. Setiap penelitian terdiri dari bilangan **y**, yang merupakan **jumlah mahasiswa** dalam kelompok, dan bilangan **z**, yang merupakan **jumlah hubungan** pertemanan dalam kelompok tersebut. Setiap baris berikutnya berisi daftar hubungan pertemanan yang ditandai dengan **NRP** mahasiswa, dipisahkan dengan spasi.

OUTPUT

Output untuk setiap penelitian dimulai dengan "**Penelitian #i**", dengan i sebagai nomor urut penelitian yang dimulai dari 1. Selanjutnya, keluarkan output "**Tidak ada yang aneh, kak!**" jika hasil penelitian mendukung hipotesis Ray. Namun, jika hasil penelitian tidak sesuai dengan hipotesis Ray, keluarkan output "**Kak, aku menemukan ketidaksesuaian dengan hipotesis!**". Output setiap penelitian dipisahkan dengan newline.

CONSTRAINTS

```
1 ≤ x ≤ 20
1 ≤ y ≤ 50
1 ≤ z ≤ 10
1 ≤ NRP ≤ 100
```

SAMPLE INPUT 1

```
1
3 3
1 2
2 3
1 3
```

SAMPLE OUTPUT 1

```
Penelitian #1:
Kak, aku menemukan ketidaksesuaian dengan hipotesis!
```

kode dan penjelasannya

```
#include <iostream>
#include <vector>
#include <queue>
#define MAX_VERTEX 101

using namespace std;

void addEdge(vector<int> *adjList, int totalEdge, int *start)
{
    for(int i=0; i<totalEdge; i++)
    {
        int nrp1, nrp2;
        cin >> nrp1 >> nrp2;
        if(*start == 0)
            *start = nrp1;
        adjList[nrp1].push_back(nrp2);
    }
}
//Fungsi ini mengambil tiga parameter: sebuah pointer ke array vektor adjList, sebuah int totalEdge, dan sebuah pointer ke sebuah int start
//Lalu mengiterasi totalEdge beberapa kali dan menginput int nrp1 dan nrp2 dari user
//Jika nilai yang ditunjuk oleh start adalah 0, fungsi ini akan menetapkan start ke nilai nrp1
//Kemudian menambahkan nrp2 ke vektor pada indeks nrp1 dari adjList, yang merepresentasikan sebuah sisi antara nrp1 dan nrp2

bool bfs(vector<int> *adjList, int start){
    vector<bool> visited(MAX_VERTEX, false);
    vector<int> is_introvert(MAX_VERTEX, 0);
    queue<int> q;
    //Fungsi bfs ini melakukan BFS pada graf yang diwakili oleh adjList mulai dari simpul awal
    //Menginisialisasi sebuah vektor visited dengan elemen MAX_VERTEX, semuanya diset ke false
    //Lalu menginisialisasi sebuah vektor is_introvert dengan elemen-elemen MAX_VERTEX, semuanya diset ke 0
    //Membuat sebuah queue q untuk menyimpan simpul-simpul yang akan dikunjungi

    int mark = 1;
    q.push(start);
    visited[start] = true;
    is_introvert[start] = mark;
    //Menetapkan mark ke 1, mewakili kelompok simpul pertama
    //Algoritma ini mempush start ke queue q dan menandainya sebagai dikunjungi dan menjadi bagian dari kelompok pertama

    while(!q.empty()){
    //Algoritma BFS terus berlanjut sampai queue q menjadi kosong
        if(q.size() > 1) return false;
        int temp = q.front();
        q.pop();

        for(auto it:adjList[temp]){
            if (!visited[it]){
                q.push(it);
                visited[it] = true;
                if(mark==1)
                    mark=2;
                else
                    mark=1;
                is_introvert[it] = mark;
                //result->push_back(it);
            }
            else if(is_introvert[it] == is_introvert[temp])
                return false;

        }
```

```
    }
    return true;
}
//Di dalam perulangan, algoritma ini memeriksa apakah ukuran queue lebih besar dari 1, yang mengindikasikan sebuah ketidaksesuaian
//Lalu mengambil elemen terdepan dari queue dan menyimpannya di temp
/*Ia mengiterasi simpul-simpul yang berdekatan dari temp dan melakukan hal berikut:
    Jika simpul yang berdekatan belum dikunjungi, dipush ke queue q, menandainya sebagai telah dikunjungi,
    dan menugaskannya ke kelompok yang berlawanan (1 atau 2) dari simpul induknya
    Jika simpul yang berdekatan telah dikunjungi dan berada dalam kelompok yang sama dengan simpul induknya,
    ini mengindikasikan sebuah ketidaksesuaian, dan fungsi mengembalikan nilai salah
    Setelah BFS selesai tanpa ketidaksesuaian, fungsi ini akan mengembalikan nilai true. */

int main()
{
    int testcase;
    cin >> testcase;
    //Menerima input untuk kasus uji

    for(int i=1; i<=testcase; i++)
    //Perulangan sebanyak jumlah kasus uji
    {
        int vertex, totalEdge;
        cin >> vertex >> totalEdge;
        //Untuk setiap kasus uji, menerima input jumlah vertex dan totalEdge dari user

        vector<int> adjList[MAX_VERTEX];
        //Membuat sebuah array vektor adjList dengan elemen-elemen MAX_VERTEX, masing-masing merepresentasikan daftar ketetanggaan untuk vertex

        int start = 0;
        addEdge(adjList, totalEdge, &start);
        //Menginisialisasi start ke 0
        //Lalu memanggil fungsi addEdge untuk menambahkan sisi-sisi ke graf dengan mengoper adjList, totalEdge, dan alamat start

        cout << "Penelitian #" << i << ":\n";
        if(!bfs(adjList, start)) cout << "Kak, aku menemukan ketidaksesuaian dengan hipotesis!\n" << endl;
        else cout << "Tidak ada yang aneh, kak!\n" << endl;
        //Dipanggil juga fungsi bfs untuk melakukan BFS pada graf mulai dari start dan memeriksa apakah ada ketidaksesuaian
        //Berdasarkan hasil BFS, akan dioutputkan "Kak, saya menemukan ketidaksesuaian dengan hipotesis!" jika terdapat ketidaksesuaian
        //atau akan dioutputkan "Tidak ada yang aneh, kak!" jika tidak terdapat ketidaksesuaian
    }
    return 0;
}
```

sourcecode:
https://ideone.com/mzVb5D

Nama : Lathiifah Nabiila
NRP : 5025221130