Started on	Sunday, 15 June 2025, 11:08 PM
State	Finished
Completed on	Sunday, 15 June 2025, 11:10 PM
Time taken	2 mins 46 secs
Marks	2.00/2.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Come and Go (nguồn: UVA Online Judge, Problem 11838)

Trong một thành phố có **N** địa điểm được nối với nhau bằng **M** con đường 1 chiều và 2 chiều. Yêu cầu tối thiểu của một thành phố là **từ địa điểm này bạn phải có thể đi đến một địa điểm khác bất kỳ**.

Hãy viết chương trình kiểm tra xem các con đường của thành phố có thoả mãn yêu cầu tối thiểu này không.

Dữ liệu đầu vào có dạng như sau:

```
4 5
1 2 1
1 3 2
2 4 1
3 4 1
4 1 2
```

Trong ví dụ này, có 4 địa điểm và 5 con đường, mỗi con đường có dạng a b p, trong đó a, b là các địa điểm; và nếu p = 1, con đường đang xét là đường 1 chiều, ngược lại nó là đường 2 chiều.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:

- Dòng đầu tiên chứa 2 số nguyên N và M, tương ứng là số địa điểm và số con đường.
- M dòng tiếp theo mỗi dòng chứa 3 số nguyên a, b, p. Nếu p = 1, con đường (a, b) là con đường 1 chiều, ngược lại nếu p = 2, con đường (a, b) là con đường 2 chiều.

Đầu ra (Output)

- In ra màn hình OKIE nếu các con đường của thành phố có thoả mãn yêu cầu, ngược lại in ra NO.
- Xem thêm ví dụ bên dưới.

Gợi ý

- Xây dựng đồ thị có hướng từ dữ liệu các con đường và các địa điểm
  - Địa điểm ~ đỉnh
  - Đường 1 chiều ~ cung
  - Đường 2 chiều ~ 2 cung
- Áp dụng giải thuật kiểm tra đồ thị có liên thông mạnh hay không.

For example:

Input	Result
5 7 1 2 1 2 3 1 3 1 1 2 4 1 3 4 1 4 5 1 5 3 1	OKIE
8 10 1 2 2 1 3 1 2 8 1 3 4 1 3 5 2 4 2 1 4 7 1 4 8 1 5 7 1 6 7 2	NO

Answer: (penalty regime: 10, 20, ... %)

```
#include <stdio.h>

#define MAX_SIZE 100

//----------------- Stack--------------------------
typedef int ElementType;
```

```c
typedef struct {
    ElementType data[MAX_SIZE];
    int top_idx;
} Stack;

void init_stack(Stack* pS) {
    pS->top_idx = -1;
}

int is_empty(Stack* pS) {
    return pS->top_idx == -1;
}

void push(Stack* pS, int x) {
    pS->top_idx++;
    pS->data[pS->top_idx] = x;
}

void pop(Stack* pS) {
    pS->top_idx--;
}

ElementType top(Stack* pS) {
    return pS->data[pS->top_idx];
}

//------------------Graph-------------------------
typedef struct {
    int n, m;
    int A[MAX_SIZE][MAX_SIZE];
} Graph;

void init_graph(Graph* pG, int n) {
    pG->n = n;
    pG->m = 0;

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            pG->A[i][j] = 0;
        }
    }
}

void add_edge(Graph* pG, int u, int v) {
    pG->A[u][v]++;
    // if (u != v) {
    //     pG->A[v][u]++;
    // }

    pG->m++;
}

int adj(Graph* pG, int u, int v) {
    return pG->A[u][v] > 0;
}

//------------------Strong connection-----------------
int num[MAX_SIZE];
int min_num[MAX_SIZE];
int k;
Stack S;
int on_stack[MAX_SIZE];
int cnt;

void SCC(Graph* pG, int u) {
    // 1. Đánh số và đưa vào stack
    num[u] = k;
    min_num[u] = k;
    k++;
    push(&S, u);
    on_stack[u] = 1;

    // 2.Duyệt các đỉnh kề
    for (int v = 1; v <= pG->n; v++) {
        if (adj(pG, u, v)) {
            // chưa duyệt
            if (num[v] < 0) {
```

```
                SCC(pG, v);
                min_num[u] = min_num[v] < min_num[u] ? min_num[v] : min_num[u];
            } else if (on_stack[v]) {
                min_num[u] = num[v] < min_num[u] ? num[v] : min_num[u];
            }
        }
    }

    // 3. so sánh num và min num
    if (num[u] == min_num[u]) {
        cnt++;
        int w;
        do {
            w = top(&S);
            pop(&S);
            on_stack[w] = 0;
        } while (w != u);
    }
}
int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    Graph G;
    init_graph(&G, n);

    for (int i = 0; i < m; i++) {
        int u, v, t;
        scanf("%d%d%d", &u, &v, &t);
        add_edge(&G, u, v);
        if (t == 2) {
            add_edge(&G, v, u);
        }
    }

    init_stack(&S);
    k = 1;

    // chưa duyet
    for (int i = 0; i <= n; i++) {
        num[i] = -1;
    }
    cnt = 0;
    for (int i = 1; i <= G.n; i++) {
        if (num[i]< 0) {
            SCC(&G, i);
        }
    }

    if (cnt > 1) {
        printf("NO");
    }else{
        printf("OKIE");
    }
}
```

	Input	Expected	Got	
✔	5 7 1 2 1 2 3 1 3 1 1 2 4 1 3 4 1 4 5 1 5 3 1	OKIE	OKIE	✔

	Input	Expected	Got	
✔	8 10 1 2 2 1 3 1 2 8 1 3 4 1 3 5 2 4 2 1 4 7 1 4 8 1 5 7 1 6 7 2	NO	NO	✔
✔	3 2 1 2 2 1 3 2	OKIE	OKIE	✔
✔	3 2 1 2 2 1 3 1	NO	NO	✔
✔	4 2 1 2 2 3 4 2	NO	NO	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Trust group (nguồn: UVA Online Judge, Problem 11709)

Phòng nhân sự của tổ chức *Association of Cookie Monsters* (ACM) nhận thấy rằng gần đây hiệu quả làm việc của các nhân viên có chiều hướng giảm sút. Vì thế họ đã lấy ý kiến các nhân viên và phát hiện ra nguyên nhân của vấn đề này, đó là: sự tin cậy. Một số nhân viên không tin cậy vào các nhân viên khác trong nhóm làm việc của mình. Điều này làm giảm động lực và niềm vui trong công việc của các nhân viên.

Phòng nhân sự muốn giải quyết triệt để vấn đề này nên họ quyết định tổ chức lại các nhóm làm việc sao cho ổn định. Một nhóm làm việc sẽ ổn định khi mà những người trong nhóm tin cậy lẫn nhau. Họ đã hỏi các nhân viên và biết được những người mà một nhân viên tin cậy trực tiếp. Ngoài ra, sự tin cậy có tính bắt cầu: *nếu A tin cậy B và B tin cậy C thì A cũng sẽ tin cậy C*. Lẽ dĩ nhiên, một nhân viên sẽ tự tin cậy chính bản thân mình. Tuy nhiên, cần chú ý là sự tin cậy lại không có tính đối xứng: *A tin cậy B thì không nhất thiết B phải tin cậy A*.

Phòng nhân sự muốn tổ chức thành ít nhóm nhất có thể. Hãy lập trình để giúp họ.

Giả sử các nhân viên được đánh số là 1, 2, ..., n.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím theo định dạng:

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số nhân viên và số cặp tin cậy.
- m dòng tiếp theo, mỗi dòng chứa 2 số nguyên a b, nói rằng người a tin cậy vào người b.

Đầu ra (Output)

- In ra màn hình số lượng nhóm ít nhất mà những người trong nhóm đều tin cậy lẫn nhau
- Xem thêm trong phần ví dụ

For example:

Input	Result
5 7 1 2 2 3 3 1 2 4 3 4 4 5 5 3	1
8 13 1 2 1 3 2 1 2 8 3 4 3 5 4 2 4 7 4 8 5 3 5 7 6 7 7 6	3

Answer: (penalty regime: 10, 20, ... %)

```
#include <stdio.h>

#define MAX_SIZE 100

//----------------- Stack--------------------------
typedef int ElementType;

typedef struct {
    ElementType data[MAX_SIZE];
    int top_idx;
} Stack;

void init_stack(Stack* pS) {
    pS->top_idx = -1;
}

```

```c
int is_empty(Stack* pS) {
    return pS->top_idx == -1;
}

void push(Stack* pS, int x) {
    pS->top_idx++;
    pS->data[pS->top_idx] = x;
}

void pop(Stack* pS) {
    pS->top_idx--;
}

ElementType top(Stack* pS) {
    return pS->data[pS->top_idx];
}

//------------------Graph--------------------------
typedef struct {
    int n, m;
    int A[MAX_SIZE][MAX_SIZE];
} Graph;

void init_graph(Graph* pG, int n) {
    pG->n = n;
    pG->m = 0;

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            pG->A[i][j] = 0;
        }
    }
}

void add_edge(Graph* pG, int u, int v) {
    pG->A[u][v]++;
    // if (u != v) {
    //     pG->A[v][u]++;
    // }

    pG->m++;
}

int adj(Graph* pG, int u, int v) {
    return pG->A[u][v] > 0;
}

//------------------Strong connection-----------------
int num[MAX_SIZE];
int min_num[MAX_SIZE];
int k;
Stack S;
int on_stack[MAX_SIZE];
int cnt;

void SCC(Graph* pG, int u) {
    // 1. Đánh số và đưa vào stack
    num[u] = k;
    min_num[u] = k;
    k++;
    push(&S, u);
    on_stack[u] = 1;

    // 2.Duyệt các đỉnh kề
    for (int v = 1; v <= pG->n; v++) {
        if (adj(pG, u, v)) {
            // chưa duyệt
            if (num[v] < 0) {
                SCC(pG, v);
                min_num[u] = min_num[v] < min_num[u] ? min_num[v] : min_num[u];
            } else if (on_stack[v]) {
                min_num[u] = num[v] < min_num[u] ? num[v] : min_num[u];
            }
        }
    }

    // 3. so sánh num và min num
    if (num[u] == min_num[u]) {
```

```
            cnt++;
            int w;
            do {
                w = top(&S);
                pop(&S);
                on_stack[w] = 0;
            } while (w != u);
        }
    }
    int main() {
        int n, m;
        scanf("%d %d", &n, &m);

        Graph G;
        init_graph(&G, n);

        for (int i = 0; i < m; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            add_edge(&G, u, v);

        }

        init_stack(&S);
        k = 1;

        // chưa duyet
        for (int i = 0; i <= n; i++) {
            num[i] = -1;
        }
        cnt = 0;
        for (int i = 1; i <= G.n; i++) {
            if (num[i]< 0) {
                SCC(&G, i);
            }
        }

        printf("%d",cnt);
    }
```

	Input	Expected	Got	
✔	5 7 1 2 2 3 3 1 2 4 3 4 4 5 5 3	1	1	✔
✔	8 13 1 2 1 3 2 1 2 8 3 4 3 5 4 2 4 7 4 8 5 3 5 7 6 7 7 6	3	3	✔

Passed all tests! ✔

Question author's solution (C):

```
#include <stdio.h>
```

```c
#include <stdio.h>

/* Khai báo CTDL Graph*/
#define MAX_N 100
typedef struct {
    int n, m;
    int A[MAX_N][MAX_N];
} Graph;

void init_graph(Graph *pG, int n) {
    pG->n = n;
    pG->m = 0;
    for (int u = 1 ; u <= n; u++)
        for (int v = 1 ; v <= n; v++)
            pG->A[u][v] = 0;
}

void add_edge(Graph *pG, int u, int v) {
    pG->A[u][v] += 1;
    //if (u != v)
    //    pG->A[v][u] += 1;

    if (pG->A[u][v] > 1)
        printf("da cung (%d, %d)\n", u, v);
    if (u == v)
        printf("khuyen %d\n", u);


    pG->m++;
}

int adjacent(Graph *pG, int u, int v) {
    return pG->A[u][v] > 0;
}


#define MAX_SIZE 100
typedef int ElementType;
typedef struct {
    ElementType data[MAX_SIZE];
    int top_idx;
} Stack;
/* Hàm khởi tạo ngăn xếp rỗng */
void make_null_stack(Stack *pS) {
    pS->top_idx = -1;
}
/* Hàm thêm phần tử u vào đỉnh ngăn xếp */
void push(Stack *pS, ElementType u) {
    pS->top_idx++;
    pS->data[pS->top_idx] = u;
}
/* Hàm xem phần tử trên đỉnh ngăn xếp */
ElementType top(Stack *pS) {
    return pS->data[pS->top_idx];
}
/* Hàm xoá bỏ phần tử trên đỉnh ngăn xếp */
void pop(Stack *pS) {
    pS->top_idx--;
}
/* Hàm kiểm tra ngăn xếp rỗng */
int empty(Stack *pS) {
    return pS->top_idx == -1;
}



int min(int a, int b) {
    return a < b ? a : b;
}


int num[MAX_N], min_num[MAX_N];
int k;
Stack S;
int on_stack[MAX_N];
int nb_cnt;


//Duyệt đồ thị bắt đầu từ đỉnh u
```

```c
//Duyệt đồ thị bắt đầu từ đỉnh u
void SCC(Graph *pG, int u) {
    //1. Đánh số u, đưa u vào ngăn xếp S
    num[u] = min_num[u] = k; k++;
    push(&S, u);
    on_stack[u] = 1;
    //2. Xét các đỉnh kề của u
    for (int v = 1; v <= pG->n; v++) {
        if (adjacent(pG, u, v)) {
            if (num[v] < 0) {
                SCC(pG, v);
                min_num[u] = min(min_num[u], min_num[v]);
            } else if (on_stack[v])
                min_num[u] = min(min_num[u], num[v]);
        }
    }
    //3. Kiểm tra u có phải là đỉnh khớp
    if (num[u] == min_num[u]) {
        //printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);
        nb_cnt++;
        int w;
        do {    //Lấy các đỉnh trong S ra cho đến khi gặp u
            w = top(&S);
            pop(&S);
            on_stack[w] = 0;
            //printf("Lay %d.\n", w);

        } while (w != u);
    }
}




int main() {
    //1. Khai báo đồ thị G
    Graph G;
    //2. Đọc dữ liệu và dựng đồ thị
    int n, m, u, v;
    scanf("%d%d", &n, &m);
    init_graph(&G, n);
    for (int e = 0; e < m; e++) {
        scanf("%d%d", &u, &v);
        add_edge(&G, u, v);
    }

    for (int u = 1; u <= G.n; u++)
        num[u] = -1;


    //3. Duyệt toàn bộ đồ thị để kiểm tra chu trình
    k = 1;              //1b. Tất cả đều chưa duyệt
    make_null_stack(&S);        //1c. Làm rỗng ngăn xếp
    //2. Duyệt toàn bộ đồ thị để tìm BPLT mạnh
    nb_cnt = 0;
    for (int u = 1; u <= G.n; u++)
        if (num[u] == -1)       //u chưa duyệt
            SCC(&G, u);     //duyệt nó


    printf("%d\n", nb_cnt);

    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

◄ * Bài tập 13 - Tính liên thông mạnh

Jump to...

Bổ sung - Kiểm tra chu trình - đồ thị vô hướng ►