Started on	Wednesday, 11 June 2025, 10:29 AM
State	Finished
Completed on	Sunday, 15 June 2025, 11:04 PM
Time taken	4 days 12 hours
Marks	4.00/4.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Viết chương trình đọc vào một đồ thị có hướng và áp dụng thuật toán Tarjan lên nó.

Đầu vào (Input)

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

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

Đầu ra (Output)

- In ra màn hình n dòng. Ở dòng thứ u, in ra 2 số nguyên num[u] và min_num[u] cách nhau 1 khoảng trắng.
- Xem thêm ví dụ bên dưới.

For example:

Input	Result
5 7 1 2 2 3 3 1 2 4 3 4 4 5 5 3	1 1 2 1 3 1 4 3 5 3
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	1 1 2 1 4 2 5 2 8 4 7 6 6 6 3 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;
}

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];
```

```c
}

//------------------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];

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]) {
        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++) {
```

```
110         int u, v;
111         scanf("%d%d", &u, &v);
112         add_edge(&G, u, v);
113     }
114
115     init_stack(&S);
116     k = 1;
117
118     // chưa duyet
119     for (int i = 0; i <= n; i++) {
120         num[i] = -1;
121     }
122
123     for (int i = 1; i <= G.n; i++) {
124         if (num[i] < 0) {
125             SCC(&G, i);
126         }
127     }
128
129     for (int i = 1; i <= G.n; i++) {
130         printf("%d %d\n", num[i], min_num[i]);
131     }
132 }
```

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

Passed all tests! ✔

Question author's solution (C):

```
1  #include <stdio.h>
2
3  /* Khai báo CTDL Graph*/
4  #define MAX_N 100
5  typedef struct {
6      int n, m;
7      int A[MAX_N][MAX_N];
8  } Graph;
9
10 void init_graph(Graph *pG, int n) {
11     pG->n = n;
12     pG->m = 0;
13     for (int u = 1 ; u <= n; u++)
14         for (int v = 1 ; v <= n; v++)
15             pG->A[u][v] = 0;
16 }
```

```c
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
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. Kiem tra u co phai la dinh khop
    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


    for (int u = 1; u <= n; u++)
        printf("%d %d\n", num[u], min_num[u]);

    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Viết chương trình đọc vào một đồ thị có hướng và kiểm tra xem nó có liên thông mạnh không.

Đầu vào (Input)

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

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

Đầu ra (Output)

- In ra màn hình **STRONG CONNECTED** nếu đồ thị liên thông mạnh, ngược lại in ra **DISCONNECTED.**
- Xem thêm ví dụ bên dưới.

For example:

Input	Result
5 7 1 2 2 3 3 1 2 4 3 4 4 5 5 3	STRONG CONNECTED
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	DISCONNECTED

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;
}

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];
```

```c
}

//------------------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);
        }
    }

    if (cnt > 1) {
        printf("DISCONNECTED");
    } else {
        printf("STRONG CONNECTED");
    }
}
```

	Input	Expected	Got	
✔	5 7 1 2 2 3 3 1 2 4 3 4 4 5 5 3	STRONG CONNECTED	STRONG CONNECTED	✔
✔	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	DISCONNECTED	DISCONNECTED	✔

Passed all tests! ✔

Question author's solution (C):

```c
#include <stdio.h>

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

```c
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
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


    if (nb_cnt == 1)
        printf("STRONG CONNECTED\n");
    else
        printf("DISCONNECTED\n");

    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Viết chương trình đọc vào một đồ thị có hướng và đếm số bộ phận liên thông mạnh của nó.

Đầu vào (Input)

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

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

Đầu ra (Output)

- In ra màn hình số BPLT mạnh của đồ thị.
- Xem thêm ví dụ bên dưới.

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;
}

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];
```

```c
}

//------------------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):

```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
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.

Question **4**

Correct

Mark 1.00 out of 1.00

Viết chương trình đọc vào một đồ thị có hướng và tìm bộ phận liên thông mạnh có nhiều đỉnh nhất.

Đầu vào (Input)

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

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.
- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

Đầu ra (Output)

- In ra màn hình số đỉnh của BPLT mạnh có nhiều đỉnh nhất.
- Xem thêm ví dụ bên dưới.

For example:

Input	Result
5 6 1 2 2 3 3 1 2 4 3 4 5 3	3
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	5

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

```c
#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;
}

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];
}
```

```c
//------------------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, max;

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 = 0;
        int w;
        do {
            cnt++;
            w = top(&S);
            pop(&S);
            on_stack[w] = 0;
        } while (w != u);
        max = cnt > max ? cnt : max;
    }
}
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;
    max = cnt;
    for (int i = 1; i <= G.n; i++) {
        if (num[i] < 0) {
            SCC(&G, i);
        }
    }

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

	Input	Expected	Got	
✔	5 6 1 2 2 3 3 1 2 4 3 4 5 3	3	3	✔
✔	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	5	5	✔

Passed all tests! ✔

Question author's solution (C):

```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 max_cnt;


//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]);
```

```
92              min_num[u] = min(min_num[u], num[v]);
93          }
94      }
95      //3. Kiểm tra u có phải là đỉnh khớp
96      if (num[u] == min_num[u]) {
97          //printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);
98          int nb_cnt = 0;
99          int w;
100         do {    //Lấy các đỉnh trong S ra cho đến khi gặp u
101             w = top(&S);
102             pop(&S);
103             on_stack[w] = 0;
104             //printf("Lay %d.\n", w);
105             nb_cnt++;
106
107         } while (w != u);
108         if (nb_cnt > max_cnt)
109             max_cnt = nb_cnt;
110     }
111 }
112
113
114
115
116 int main() {
117     //1. Khai báo đồ thị G
118     Graph G;
119     //2. Đọc dữ liệu và dựng đồ thị
120     int n, m, u, v;
121     scanf("%d%d", &n, &m);
122     init_graph(&G, n);
123     for (int e = 0; e < m; e++) {
124         scanf("%d%d", &u, &v);
125         add_edge(&G, u, v);
126     }
127
128     for (int u = 1; u <= G.n; u++)
129         num[u] = -1;
130
131
132     //3. Duyệt toàn bộ đồ thị để kiểm tra chu trình
133     k = 1;              //1b. Tất cả đều chưa duyệt
134     make_null_stack(&S);        //1c. Làm rỗng ngăn xếp
135     //2. Duyệt toàn bộ đồ thị để tìm BPLT mạnh
136     max_cnt = 0;
137     for (int u = 1; u <= G.n; u++)
138         if (num[u] == -1)       //u chưa duyệt
139             SCC(&G, u);     //duyệt nó
140
141
142     printf("%d\n", max_cnt);
143
144     return 0;
145 }
146
147
```

Correct

Marks for this submission: 1.00/1.00.

◄ * Bài tập 12 - Kiểm tra đồ thị phân đôi

Jump to...

* Bài tập 14 - Ứng dụng liên thông mạnh ►