Started on	Friday, 13 June 2025, 11:07 PM
State	Finished
Completed on	Sunday, 15 June 2025, 5:26 PM
Time taken	1 day 18 hours
Marks	3.60/4.00
Grade	**9.00** out of 10.00 (**90**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Viết chương trình đọc một đồ thị **vô hướng** từ bàn phím và đếm số bộ phận liên thông của đồ thị.

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 số bộ phận liên thông của đồ thị.

Gợi ý

- Xem tài liệu thực hành

Hướng dẫn đọc dữ liệu và chạy thử chương trình

- Để chạy thử chương trình, bạn nên tạo một tập tin **dt.txt** chứa đồ thị cần kiểm tra.
- Thêm dòng **freopen("dt.txt", "r", stdin);** vào ngay sau hàm main(). Khi nộp bài, nhớ gỡ bỏ dòng này ra.
- Có thể sử dụng đoạn chương trình đọc dữ liệu mẫu sau đây:

```
freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.
Graph G;
int n, m, u, v, w, e;
scanf("%d%d", &n, &m);
init_graph(&G, n);

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

For example:

Input	Result
4 3 2 1 1 3 2 4	1

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
typedef struct{
    int n,m;
    int A[max][max];
}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;
    pG->A[v][u] = 1;
    pG->m++;
}

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

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

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

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

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

```c
}

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

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

void DFS(Graph *pG, int s){
    Stack S;
    make_null_stack(&S);

    push(&S, s);

    while (!empty(&S)){
        int u = top(&S);
        pop(&S);
        if(mark[u] != 0)
            continue;
        mark[u] = 1;

        for(int v=1; v <= pG->n; v++){
            if(adjacent(pG,u,v)&& mark[v] == 0)
                push(&S,v);
        }
    }
}
int main (){
    Graph G;
    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++)
    mark[u] = 0;
    int cnt = 0;
    for(int u = 1; u <= G.n; u++)
        if(mark[u]==0){
            DFS(&G,u);
            cnt++;
        }

    printf("%d",cnt);
    return 0;
}
```

	Input	Expected	Got	
✔	4 3 2 1 1 3 2 4	1	1	✔
✔	4 2 1 2 3 4	2	2	✔
✔	4 2 1 4 2 3	2	2	✔
✔	13 16 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 5 6	1	1	✔

	Input	Expected	Got	
✔	13 15 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	2	2	✔
✔	13 13 1 2 1 12 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	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;

    pG->m++;
}

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




//Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
int mark[MAX_N];

void DFS(Graph *pG, int u) {
    //1. Đánh dấu u đã duyệt
    //printf("Duyet %d\n", u);  //Làm gì đó trên u
    mark[u] = 1;            //Đánh dấu nó đã duyệt

    //2. Xét các đỉnh kề của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
            DFS(pG, v);                 //gọi đệ quy duyệt nó

}

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);
    }

    //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
    for (int u = 1; u <= G.n; u++) {
        mark[u] = 0;
    }

    //4. Duyệt toàn bộ đồ thị
    int cnt = 0;
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0) {
            DFS(&G, u);
            cnt++;
        }

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

    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 một đồ thị **vô hướng** từ bàn phím và đếm số đỉnh của bộ phận liên thông của đỉnh 1.

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 số đỉnh của bộ phận liên thông của đỉnh 1.

Gợi ý

- Xem tài liệu thực hành.

Hướng dẫn đọc dữ liệu và chạy thử chương trình

- Để chạy thử chương trình, bạn nên tạo một tập tin **dt.txt** chứa đồ thị cần kiểm tra.
- Thêm dòng **freopen("dt.txt", "r", stdin);** vào ngay sau hàm main(). Khi nộp bài, nhớ gỡ bỏ dòng này ra.
- Có thể sử dụng đoạn chương trình đọc dữ liệu mẫu sau đây:

```
freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.
Graph G;
int n, m, u, v, w, e;
scanf("%d%d", &n, &m);
init_graph(&G, n);

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

For example:

Input	Result
4 3 2 1 1 3 2 4	4

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
int nb_u;
typedef struct{
    int n,m;
    int A[max][max];
}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;
    pG->A[v][u] = 1;
    pG->m++;
}

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

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

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

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

ElementType top(Stack *pS){
```

```
    return pS->data[pS->top_idx];
}

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

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

void DFS(Graph *pG, int s){
    Stack S;
    make_null_stack(&S);

    push(&S, s);

    while (!empty(&S)){
        int u = top(&S);
        pop(&S);
        if(mark[u] != 0)
            continue;

        mark[u] = 1;
        nb_u ++;

        for(int v=1; v <= pG->n; v++){
            if(adjacent(pG,u,v)&& mark[v] == 0)
                push(&S,v);
        }
    }
}
int main (){
    Graph G;
    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++)
        mark[u] = 0;

    nb_u =0;
    DFS(&G,1);
    printf("%d",nb_u);
    return 0;
}
```

	Input	Expected	Got	
✔	4 3 2 1 1 3 2 4	4	4	✔
✔	4 2 1 2 3 4	2	2	✔
✔	4 2 1 4 2 3	2	2	✔
✔	13 16 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 5 6	13	13	✔

	Input	Expected	Got	
✔	13 15 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	10	10	✔
✔	13 13 1 2 1 12 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	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;

    pG->m++;
}

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




//Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
int mark[MAX_N];
int nb_u;

void DFS(Graph *pG, int u) {
    //1. Đánh dấu u đã duyệt
    //printf("Duyet %d\n", u);  //Làm gì đó trên u
    mark[u] = 1;            //Đánh dấu nó đã duyệt
    nb_u++;

    //2. Xét các đỉnh kề của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
            DFS(pG, v);                 //gọi đệ quy duyệt nó

}

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);
    }

    //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
    for (int u = 1; u <= G.n; u++) {
        mark[u] = 0;
    }

    //4. Duyệt đồ thị từ đỉnh 1
    nb_u = 0;
    DFS(&G, 1);

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

    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 một đồ thị **vô hướng** từ bàn phím và đếm số đỉnh của bộ phận liên thông của đỉnh **s**. Đỉnh s được đọc từ bàn phím.

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).
- Dòng cuối cùng chứa đỉnh s.

Đầu ra (Output)

- In ra số đỉnh của bộ phận liên thông của đỉnh s.

Gợi ý

- Xem tài liệu thực hành.

Hướng dẫn đọc dữ liệu và chạy thử chương trình

- Để chạy thử chương trình, bạn nên tạo một tập tin **dt.txt** chứa đồ thị cần kiểm tra.
- Thêm dòng **freopen("dt.txt", "r", stdin);** vào ngay sau hàm main(). Khi nộp bài, nhớ gỡ bỏ dòng này ra.
- Có thể sử dụng đoạn chương trình đọc dữ liệu mẫu sau đây:

```
freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.
Graph G;
int n, m, u, v, w, e;
scanf("%d%d", &n, &m);
init_graph(&G, n);

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

For example:

Input	Result
4 3 2 1 1 3 2 4 3	4

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
int nb_u;
typedef struct{
    int n,m;
    int A[max][max];
}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;
    pG->A[v][u] = 1;
    pG->m++;
}

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

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

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

void push(Stack *pS, ElementType u){
    pS->top_idx++;
    pS->data[pS->top_idx] = u;
}
```

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

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

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

void DFS(Graph *pG, int s){
    Stack S;
    make_null_stack(&S);

    push(&S, s);

    while (!empty(&S)){
        int u = top(&S);
        pop(&S);
        if(mark[u] != 0)
            continue;

        mark[u] = 1;
        nb_u ++;

        for(int v=1; v <= pG->n; v++){
            if(adjacent(pG,u,v)&& mark[v] == 0)
                push(&S,v);
        }
    }
}
int main (){
    Graph G;
    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++)
        mark[u] = 0;

    nb_u =0;
    int s;
    scanf("%d",&s);
    DFS(&G,s);
    printf("%d",nb_u);
    return 0;
}
```

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

	Input	Expected	Got	
✔	13 16 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 5 6 5	13	13	✔
✔	13 15 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 6	10	10	✔
✔	13 13 1 2 1 12 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 5	3	3	✔
✔	13 14 1 2 1 12 3 7 3 12 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 2	10	10	✔

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;

    pG->m++;
}

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




//Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
int mark[MAX_N];
int nb_u;

void DFS(Graph *pG, int u) {
    //1. Đánh dấu u đã duyệt
    //printf("Duyet %d\n", u);  //Làm gì đó trên u
    mark[u] = 1;            //Đánh dấu nó đã duyệt
    nb_u++;

    //2. Xét các đỉnh kề của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
            DFS(pG, v);                 //gọi đệ quy duyệt nó

}

int main() {
    //1. Khai báo đồ thị G
    Graph G;
    //2. Đọc dữ liệu và dựng đồ thị
    int n, m, u, v, s;
    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);
    }

    scanf("%d", &s);
    //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
    for (int u = 1; u <= G.n; u++) {
        mark[u] = 0;
    }

    //4. Duyệt đồ thị từ đỉnh 1
    nb_u = 0;
    DFS(&G, s);

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

    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 0.60 out of 1.00

Viết chương trình đọc một đồ thị **vô hướng** từ bàn phím, tìm bộ phận liên thông 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 số đỉnh của bộ phận liên thông có nhiều đỉnh nhất.

Gợi ý

- Xem tài liệu thực hành.

Hướng dẫn đọc dữ liệu và chạy thử chương trình

- Để chạy thử chương trình, bạn nên tạo một tập tin **dt.txt** chứa đồ thị cần kiểm tra.
- Thêm dòng **freopen("dt.txt", "r", stdin);** vào ngay sau hàm main(). Khi nộp bài, nhớ gỡ bỏ dòng này ra.
- Có thể sử dụng đoạn chương trình đọc dữ liệu mẫu sau đây:

```
freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.
Graph G;
int n, m, u, v, w, e;
scanf("%d%d", &n, &m);
init_graph(&G, n);

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

For example:

Input	Result
4 3 2 1 1 3 2 4	4

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
int nb_u;
typedef struct{
    int n,m;
    int A[max][max];
}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;
    pG->A[v][u] = 1;
    pG->m++;
}

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

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

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

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

ElementType top(Stack *pS){
```

```c
    return pS->data[pS->top_idx];
}

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

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

void DFS(Graph *pG, int s){
    Stack S;
    make_null_stack(&S);

    push(&S, s);

    while (!empty(&S)){
        int u = top(&S);
        pop(&S);
        if(mark[u] != 0)
            continue;
        mark[u] = 1;
        nb_u ++;
        for(int v=1; v <= pG->n; v++){
            if(adjacent(pG,u,v)&& mark[v] == 0)
                push(&S,v);
        }
    }
}
int main (){
    Graph G;
    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++)
    mark[u] = 0;

    int max_cnt = 0;
    for(int u = 1; u <= G.n; u++)
        if(mark[u]==0){
            nb_u = 0;
            DFS(&G,u);
            if(nb_u > max_cnt)
                max_cnt = nb_u;
        }

    printf("%d",max_cnt);
    return 0;
}
```

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

	Input	Expected	Got	
✔	13 16 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12 5 6	13	13	✔
✔	13 15 1 4 1 2 1 12 2 4 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	10	10	✔
✔	13 13 1 2 1 12 3 7 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	5	5	✔
✔	13 14 1 2 1 12 3 7 3 12 4 6 4 7 5 8 5 9 6 7 6 13 8 9 10 11 10 12 11 12	10	10	✔

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;
```

```c
}

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

    pG->m++;
}

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



//Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
int mark[MAX_N];
int nb_u;

void DFS(Graph *pG, int u) {
    //1. Đánh dấu u đã duyệt
    //printf("Duyet %d\n", u);  //Làm gì đó trên u
    mark[u] = 1;            //Đánh dấu nó đã duyệt
    nb_u++;

    //2. Xét các đỉnh kề của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
            DFS(pG, v);                 //gọi đệ quy duyệt nó

}

int main() {
    //1. Khai báo đồ thị G
    Graph G;
    //2. Đọc dữ liệu và dựng đồ thị
    int n, m, u, v, s;
    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);
    }

    scanf("%d", &s);
    //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
    for (int u = 1; u <= G.n; u++) {
        mark[u] = 0;
    }

    //4. Duyệt đồ thị từ đỉnh 1
    int max_cnt = 0;
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0) {
            nb_u = 0;
            DFS(&G, u);
            if (nb_u > max_cnt)
                max_cnt = nb_u;
        }

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

    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.60/1.00**.

◄ * Bài tập 7. Kiểm tra đồ thị vô hướng liên thông

Jump to...

* Bài tập 9. Ứng dụng liên thông ►