Started on	Thursday, 12 June 2025, 9:30 PM
State	Finished
Completed on	Friday, 13 June 2025, 1:56 PM
Time taken	16 hours 26 mins
Marks	4.00/4.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Cho một đồ thị **vô hướng đơn**. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.

Nếu vẫn còn đỉnh chưa duyệt sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.

Quy ước:

- Các đỉnh kề của 1 đỉnh được liệt kê *theo thứ tự tăng dần*

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 cây duyệt đồ thị theo định dạng:

```
1 <đỉnh cha của 1>
2 <đỉnh cha của 2>
....
i <đỉnh cha của i>
...
n <đỉnh cha của n>
```

- Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là -1.
- Xem thêm các ví dụ bên dưới.

Gợi ý

- Sử dụng mảng parent[u] để lưu đỉnh cha của đỉnh u.
- Trong quá trình duyệt, thay vì in các đỉnh ra màn hình, ghi nhận lại đỉnh cha của các đỉnh.
- Khi duyệt xong lần lượt in ra u và parent[u] (u chạy từ 1 đến n).

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 4 2 4	1 -1 2 1 3 -1 4 1

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

```
#include <stdio.h>
#define max 100

int mark[max];
int parent[max];
typedef struct{
    int n,m;
    int A[max][max];
}Graph;
typedef struct{
    int u,p;
}ElementType;
typedef struct{
    ElementType data[max];
    int front,rear;
}Queue;

//khoi tao hang doi
void make_null_queue (Queue *pQ){
    pQ->front = 0;
    pQ->rear = -1;
}

//dua ptu vao cuoi
void enqueue (Queue *pQ, ElementType u){
```

```
    pQ->rear++;
    pQ->data[pQ->rear] = u;
}

ElementType front (Queue *pQ){
    return pQ->data[pQ->front];
}

void dequeue (Queue *pQ){
    pQ->front++;
}

int empty (Queue *pQ){
    return pQ->front > pQ->rear;
}

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;
}


void BFS(Graph *pG, int s){
    Queue Q;
    make_null_queue (&Q);

    ElementType pair;
    pair.u = s;
    pair.p = -1;
    enqueue(&Q,pair);
    while (!empty(&Q)){
        ElementType pair = front(&Q);
        int u = pair.u;
        int p = pair.p;
        dequeue(&Q);

        if (mark[u] != 0){
            continue;
        }
        // printf ("%d\n", u);
        mark[u] = 1;
        parent[u]= p;
        //duyệt các cạnh kề của uu
        for (int v = 1; v <= pG->n; v++){
            if (adjacent(pG,u,v)){
                ElementType pair;
                pair.u =v;
                pair.p =u;
                enqueue (&Q,pair);
            }
        }
    }
}

int main (){
    Graph G;
    int n, m, u, v;
    scanf("%d%d", &n, &m);

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

    for (int i = 1; i <= n; i++) {
        mark[i] = 0;
        parent[i] = -1;
    }

    for (int i = 1; i <= n; i++) {
        if (mark[i] == 0) {
            BFS(&G, i);
        }
    }

    for (int i = 1; i <= n; i++) {
        printf("%d %d\n", i, parent[i]);
```

```
}
```

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

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;
}

/* Khai báo CTDL Queue */

#define MAX_SIZE 100
typedef struct {
    int u, p;
} ElementType;

typedef struct {
    ElementType data[MAX_SIZE];
    int front, rear;
} Queue;

/* Khởi tạo hàng đợi rỗng */
void make_null_queue(Queue *pQ) {
    pQ->front = 0;
```

```c
    pQ->rear = -1;
}
/* Đưa phần tử u vào cuối hàng đợi */
void enqueue(Queue *pQ, ElementType u) {
    pQ->rear++;
    pQ->data[pQ->rear] = u;
}

/* Xem phần tử đầu hàng đợi */
ElementType front(Queue *pQ) {
    return pQ->data[pQ->front];
}

/* Xoá bỏ phần bỏ đầu hàng đợi */
void dequeue(Queue *pQ) {
    pQ->front++;
}

/* Kiểm tra hàng đợi rỗng */
int empty(Queue *pQ) {
    return pQ->front > pQ->rear;
}


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

void BFS(Graph *pG, int s) {
    //1. Khai báo hàng đợi Q, khởi tạo rỗng
    Queue Q;
    make_null_queue(&Q);
    //2. Đưa s vào Q, bắt đầu duyệt từ đỉnh s */
    ElementType pair;
    pair.u = s; pair.p = -1;
    enqueue(&Q, pair);
    //3. Vòng lặp chính dùng để duyệt
    while (!empty(&Q)) {
        //3a. Lấy phần tử ở đầu hàng đợi
        ElementType pair = front(&Q); dequeue(&Q);
        int u = pair.u, p = pair.p;
        if (mark[u] != 0)       //u đã duyệt rồi, bỏ qua
            continue;

        //printf("Duyet %d\n", u);  //Làm gì đó trên u
        mark[u] = 1;            //Đánh dấu nó đã duyệt
        parent[u] = p;          //Đánh dấu nó đã duyệt

        //3b. Xét các đỉnh kề của u, đưa vào hàng đợi Q
        for (int v = 1; v <= pG->n; v++)
            if (adjacent(pG, u, v)) {
                ElementType pair;
                pair.u = v; pair.p = u;
                enqueue(&Q, pair);
            }
    }
}



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;
        parent[u] = -1;
    }

    //4. Gọi hàm DFS
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0)
            BFS(&G, u);


    for (int u = 1; u <= G.n; u++)
        printf("%d %d\n", u, parent[u]);


    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Cho một đồ thị **có hướng đơn**. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.

Nếu vẫn còn đỉnh chưa duyệt sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.

Quy ước:

- Các đỉnh kề của 1 đỉnh được liệt kê *theo thứ tự tăng dần*

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 cây duyệt đồ thị theo định dạng:

```
1 <đỉnh cha của 1>
2 <đỉnh cha của 2>
...
i <đỉnh cha của i>
..
n <đỉnh cha của n>
```

- Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là -1.
- Xem thêm các ví dụ bên dưới.

Gợi ý

- Sử dụng mảng parent[u] để lưu đỉnh cha của đỉnh u.
- Trong quá trình duyệt, thay vì in các đỉnh ra màn hình, ghi nhận lại đỉnh cha của các đỉnh.
- Khi duyệt xong lần lượt in ra u và parent[u] (u chạy từ 1 đến n).

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 4 2 4	1 -1 2 -1 3 -1 4 1

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

```
#include <stdio.h>
#define max 100

int mark[max];
int parent[max];
typedef struct{
    int n,m;
    int A[max][max];
}Graph;
typedef struct{
    int u,p;
}ElementType;
typedef struct{
    ElementType data[max];
    int front,rear;
}Queue;

//khoi tao hang doi
void make_null_queue (Queue *pQ){
    pQ->front = 0;
    pQ->rear = -1;
}

//dua ptu vao cuoi
void enqueue (Queue *pQ, ElementType u){
```

```
    pQ->rear++;
    pQ->data[pQ->rear] = u;
}

ElementType front (Queue *pQ){
    return pQ->data[pQ->front];
}

void dequeue (Queue *pQ){
    pQ->front++;
}

int empty (Queue *pQ){
    return pQ->front > pQ->rear;
}

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;
}


void BFS(Graph *pG, int s){
    Queue Q;
    make_null_queue (&Q);

    ElementType pair;
    pair.u = s;
    pair.p = -1;
    enqueue(&Q,pair);
    while (!empty(&Q)){
        ElementType pair = front(&Q);
        int u = pair.u;
        int p = pair.p;
        dequeue(&Q);

        if (mark[u] != 0){
            continue;
        }
        // printf ("%d\n", u);
        mark[u] = 1;
        parent[u]= p;
        //duyệt các cạnh kề của uu
        for (int v = 1; v <= pG->n; v++){
            if (adjacent(pG,u,v)){
                ElementType pair;
                pair.u =v;
                pair.p =u;
                enqueue (&Q,pair);
            }
        }
    }
}

int main (){
    Graph G;
    int n, m, u, v;
    scanf("%d%d", &n, &m);

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

    for (int i = 1; i <= n; i++) {
        mark[i] = 0;
        parent[i] = -1;
    }

    for (int i = 1; i <= n; i++) {
        if (mark[i] == 0) {
            BFS(&G, i);
        }
    }

    for (int i = 1; i <= n; i++) {
        printf("%d %d\n", i, parent[i]);
```

```
}

}
```

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

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;
}

/* Khai báo CTDL Queue */

#define MAX_SIZE 100
typedef struct {
    int u, p;
} ElementType;

typedef struct {
    ElementType data[MAX_SIZE];
    int front, rear;
} Queue;

/* Khởi tạo hàng đợi rỗng */
```

```c
void make_null_queue(Queue *pQ) {
    pQ->front = 0;
    pQ->rear = -1;
}
/* Đưa phần tử u vào cuối hàng đợi */
void enqueue(Queue *pQ, ElementType u) {
    pQ->rear++;
    pQ->data[pQ->rear] = u;
}

/* Xem phần tử đầu hàng đợi */
ElementType front(Queue *pQ) {
    return pQ->data[pQ->front];
}

/* Xoá bỏ phần bỏ đầu hàng đợi */
void dequeue(Queue *pQ) {
    pQ->front++;
}

/* Kiểm tra hàng đợi rỗng */
int empty(Queue *pQ) {
    return pQ->front > pQ->rear;
}


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

void BFS(Graph *pG, int s) {
    //1. Khai báo hàng đợi Q, khởi tạo rỗng
    Queue Q;
    make_null_queue(&Q);
    //2. Đưa s vào Q, bắt đầu duyệt từ đỉnh s */
    ElementType pair;
    pair.u = s; pair.p = -1;
    enqueue(&Q, pair);
    //3. Vòng lặp chính dùng để duyệt
    while (!empty(&Q)) {
        //3a. Lấy phần tử ở đầu hàng đợi
        ElementType pair = front(&Q); dequeue(&Q);
        int u = pair.u, p = pair.p;
        if (mark[u] != 0)       //u đã duyệt rồi, bỏ qua
            continue;

        //printf("Duyet %d\n", u);  //Làm gì đó trên u
        mark[u] = 1;            //Đánh dấu nó đã duyệt
        parent[u] = p;          //Đánh dấu nó đã duyệt

        //3b. Xét các đỉnh kề của u, đưa vào hàng đợi Q
        for (int v = 1; v <= pG->n; v++)
            if (adjacent(pG, u, v)) {
                ElementType pair;
                pair.u = v; pair.p = u;
                enqueue(&Q, pair);
            }
    }
}



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;
        parent[u] = -1;
    }

    //4. Gọi hàm DFS
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0)
            BFS(&G, u);


    for (int u = 1; u <= G.n; u++)
        printf("%d %d\n", u, parent[u]);


    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Cho một đồ thị **vô hướng đơn**. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh 1.

Nếu vẫn còn đỉnh chưa duyệt sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.

Quy ước:

- Các đỉnh kề của 1 đỉnh được liệt kê *theo thứ tự tăng dần*
- Nếu cài đặt dùng ngăn xếp hãy đưa các đỉnh kề của đỉnh u vào ngăn xếp theo **thứ tự từ lớn đến nhỏ**

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 cây duyệt đồ thị theo định dạng:

```
1 <đỉnh cha của 1>
2 <đỉnh cha của 2>
....
i <đỉnh cha của i>
...
n <đỉnh cha của n>
```

- Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là -1.
- Xem thêm các ví dụ bên dưới.

Gợi ý

- Sử dụng mảng parent[u] để lưu đỉnh cha của đỉnh u.
- Trong quá trình duyệt, thay vì in các đỉnh ra màn hình, ghi nhận lại đỉnh cha của các đỉnh.
- Khi duyệt xong lần lượt in ra u và parent[u] (u chạy từ 1 đến n).

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 4 2 4	1 -1 2 1 3 -1 4 2

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
int parent[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;
        }
    }
}

```

```c
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;

// kkhởi tạo hàm rỗng
void make_null_stack(Stack *pS)
{
    pS->top_idx = -1;
}

// thêm phần tử u vào đỉnh ngăn xếp
void push(Stack *pS, ElementType u)
{
    pS->top_idx++;
    pS->data[pS->top_idx] = u;
}

// xem phần từ trên đầu đỉnh ngăn xếp
ElementType top(Stack *pS)
{
    return pS->data[pS->top_idx];
}

// xóa phần tử trên đỉnh ngăn xếp
void pop(Stack *pS)
{
    pS->top_idx--;
}

// kiểm tra ngăn xếp rỗng
int empty(Stack *pS)
{
    return pS->top_idx == -1;
}

// Dựng cây duyệt đồ thị theo chiều sâu dùng đệ quy
void DFS(Graph *pG, int u, int p)
{
    // 1. đánh dấu u đã duyệt
    mark[u] = 1;
    parent[u] = p;

    // 2. xét các đỉnh kè của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0)
            DFS(pG, v, u);
}
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 i = 1; i <= G.n; i++)
        mark[i] = 0;

    for (int i = 1; i <= G.n; i++)
    {
        if (mark[i] == 0)
        {
            DFS(&G, i, -1);
        }
    }

    for (int i = 1; i <= n; i++)
    {
        printf("%d %d\n", i, parent[i]);
    }
    return 0;
}
```

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

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 parent[MAX_N];

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

    //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, u);              //gọi đệ quy duyệt nó

}
```

```c


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;
        parent[u] = -1;
    }

    //4. Gọi hàm DFS
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0)
            DFS(&G, u, -1);

    for (int u = 1; u <= G.n; u++)
        printf("%d %d\n", u, parent[u]);


    return 0;
}

```

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Cho một đồ thị **có hướng đơn**. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh 1.

Nếu vẫn còn đỉnh chưa duyệt sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.

Quy ước:

- Các đỉnh kề của 1 đỉnh được liệt kê *theo thứ tự tăng dần*
- Nếu cài đặt dùng ngăn xếp hãy đưa các đỉnh kề của đỉnh u vào ngăn xếp theo **thứ tự từ lớn đến nhỏ**

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 cây duyệt đồ thị theo định dạng:

```
1 <đỉnh cha của 1>
2 <đỉnh cha của 2>
....
i <đỉnh cha của i>
...
n <đỉnh cha của n>
```

- Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là -1.
- Xem thêm các ví dụ bên dưới.

Gợi ý

- Sử dụng mảng parent[u] để lưu đỉnh cha của đỉnh u.
- Trong quá trình duyệt, thay vì in các đỉnh ra màn hình, ghi nhận lại đỉnh cha của các đỉnh.
- Khi duyệt xong lần lượt in ra u và parent[u] (u chạy từ 1 đến n).

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 4 2 4	1 -1 2 -1 3 -1 4 1

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

```
#include <stdio.h>
#define max 100
typedef int ElementType;
int mark[max];
int parent[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;
        }
    }
}

```

```c
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;

// kkhởi tạo hàm rỗng
void make_null_stack(Stack *pS)
{
    pS->top_idx = -1;
}

// thêm phần tử u vào đỉnh ngăn xếp
void push(Stack *pS, ElementType u)
{
    pS->top_idx++;
    pS->data[pS->top_idx] = u;
}

// xem phần từ trên đầu đỉnh ngăn xếp
ElementType top(Stack *pS)
{
    return pS->data[pS->top_idx];
}

// xóa phần tử trên đỉnh ngăn xếp
void pop(Stack *pS)
{
    pS->top_idx--;
}

// kiểm tra ngăn xếp rỗng
int empty(Stack *pS)
{
    return pS->top_idx == -1;
}

// Dựng cây duyệt đồ thị theo chiều sâu dùng đệ quy
void DFS(Graph *pG, int u, int p)
{
    // 1. đánh dấu u đã duyệt
    mark[u] = 1;
    parent[u] = p;

    // 2. xét các đỉnh kè của u
    for (int v = 1; v <= pG->n; v++)
        if (adjacent(pG, u, v) && mark[v] == 0)
            DFS(pG, v, u);
}
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 i = 1; i <= G.n; i++)
        mark[i] = 0;

    for (int i = 1; i <= G.n; i++)
    {
        if (mark[i] == 0)
        {
            DFS(&G, i, -1);
        }
    }

    for (int i = 1; i <= n; i++)
    {
        printf("%d %d\n", i, parent[i]);
    }
    return 0;
}
```

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

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 parent[MAX_N];

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

    //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, u);              //gọi đệ quy duyệt nó

}
```

```c
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;
        parent[u] = -1;
    }

    //4. Gọi hàm DFS
    for (int u = 1; u <= G.n; u++)
        if (mark[u] == 0)
            DFS(&G, u, -1);


    for (int u = 1; u <= G.n; u++)
        printf("%d %d\n", u, parent[u]);


    return 0;
}
```

Correct

Marks for this submission: 1.00/1.00.

◄ Bài 6 - Đọc đồ thị từ tập tin

Jump to...

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