

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%)

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

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

```
1 #include <stdio.h>
2 #define max 100
3
4 int mark[max];
5 int parent[max];
6 typedef struct{
7     int n,m;
8     int A[max][max];
9 }Graph;
10 typedef struct{
11     int u,p;
12 }ElementType;
13 typedef struct{
14     ElementType data[max];
15     int front,rear;
16 }Queue;
17
18 //khởi tạo hàng đợi
19 void make_null_queue (Queue *pQ){
20     pQ->front = 0;
21     pQ->rear = -1;
22 }
23
24 //dưa ptu vào cuối
25 void enqueue (Queue *pQ, ElementType u){
```



```

26     pQ->rear++;
27     pQ->data[pQ->rear] = u;
28 }
29
30 ElementType front (Queue *pQ){
31     return pQ->data[pQ->front];
32 }
33
34 void dequeue (Queue *pQ){
35     pQ->front++;
36 }
37
38 int empty (Queue *pQ){
39     return pQ->front > pQ->rear;
40 }
41
42 void init_graph (Graph *pG, int n){
43     pG->n = n;
44     pG->m = 0;
45     for (int u = 1; u <= n; u++){
46         for (int v = 1; v <= n; v++){
47             pG->A[u][v] = 0;
48         }
49     }
50 }
51
52 void add_edge(Graph *pG, int u, int v){
53     pG->A[u][v] = 1;
54     pG->A[v][u] = 1;
55     pG->m++;
56 }
57 int adjacent(Graph *pG, int u, int v)
58 {
59     return pG->A[u][v] > 0;
60 }
61
62
63 void BFS(Graph *pG, int s){
64     Queue Q;
65     make_null_queue (&Q);
66
67     ElementType pair;
68     pair.u = s;
69     pair.p = -1;
70     enqueue(&Q,pair);
71     while (!empty(&Q)){
72         ElementType pair = front(&Q);
73         int u = pair.u;
74         int p = pair.p;
75         dequeue(&Q);
76
77         if (mark[u] != 0){
78             continue;
79         }
80         // printf ("%d\n", u);
81         mark[u] = 1;
82         parent[u]= p;
83         //duyet các cạnh kề của uu
84         for (int v = 1; v <= pG->n; v++){
85             if (adjacent(pG,u,v)){
86                 ElementType pair;
87                 pair.u =v;
88                 pair.p =u;
89                 enqueue (&Q,pair);
90             }
91         }
92     }
93 }
94
95 int main (){
96     Graph G;
97     int n, m, u, v;
98     scanf("%d%d", &n, &m);
99
100     init_graph(&G, n);
101     for (int i = 0; i < m; i++) {
102         scanf("%d%d", &u, &v);
103         add_edge(&G, u, v);
104     }
105
106     for (int i = 1; i <= n; i++) {
107         mark[i] = 0;
108         parent[i] = -1;
109     }
110
111     for (int i = 1; i <= n; i++) {
112         if (mark[i] == 0) {
113             BFS(&G, i);
114         }
115     }
116
117     for (int i = 1; i <= n; i++) {
118         printf("%d %d\n", i, parent[i]);
119     }

```

```

119 }
120 }
121 }
122 }

```

	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):

```

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 }
17
18 void add_edge(Graph *pG, int u, int v) {
19     pG->A[u][v] += 1;
20     if (u != v)
21         pG->A[v][u] += 1;
22
23     pG->m++;
24 }
25
26 int adjacent(Graph *pG, int u, int v) {
27     return pG->A[u][v] > 0;
28 }
29
30 /* Khai báo CTDL Queue */
31
32 #define MAX_SIZE 100
33 typedef struct {
34     int u, p;
35 } ElementType;
36
37 typedef struct {
38     ElementType data[MAX_SIZE];
39     int front, rear;
40 } Queue;
41
42 /* Khởi tạo hàng đợi rỗng */
43 void make_null_queue(Queue *pQ) {
44     pQ->front = 0;

```

```

45     pQ->rear = -1;
46 }
47 /* Đưa phần tử u vào cuối hàng đợi */
48 void enqueue(Queue *pQ, ElementType u) {
49     pQ->rear++;
50     pQ->data[pQ->rear] = u;
51 }
52
53 /* Xem phần tử đầu hàng đợi */
54 ElementType front(Queue *pQ) {
55     return pQ->data[pQ->front];
56 }
57
58 /* Xóa bỏ phần tử đầu hàng đợi */
59 void dequeue(Queue *pQ) {
60     pQ->front++;
61 }
62
63 /* Kiểm tra hàng đợi rỗng */
64 int empty(Queue *pQ) {
65     return pQ->front > pQ->rear;
66 }
67
68 //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
69 int mark[MAX_N];
70 int parent[MAX_N];
71
72 void BFS(Graph *pG, int s) {
73     //1. Khai báo hàng đợi Q, khởi tạo rỗng
74     Queue Q;
75     make_null_queue(&Q);
76     //2. Đưa s vào Q, bắt đầu duyệt từ đỉnh s */
77     ElementType pair;
78     pair.u = s; pair.p = -1;
79     enqueue(&Q, pair);
80     //3. Vòng lặp chính dùng để duyệt
81     while (!empty(&Q)) {
82         //3a. Lấy phần tử ở đầu hàng đợi
83         ElementType pair = front(&Q); dequeue(&Q);
84         int u = pair.u, p = pair.p;
85         if (mark[u] != 0) //u đã duyệt rồi, bỏ qua
86             continue;
87
88         //printf("Duyet %d\n", u); //Làm gì đó trên u
89         mark[u] = 1; //Đánh dấu nó đã duyệt
90         parent[u] = p; //Đánh dấu nó đã duyệt
91
92         //3b. Xét các đỉnh kề của u, đưa vào hàng đợi Q
93         for (int v = 1; v <= pG->n; v++)
94             if (adjacent(pG, u, v)) {
95                 ElementType pair;
96                 pair.u = v; pair.p = u;
97                 enqueue(&Q, pair);
98             }
99     }
100 }
101
102
103
104
105 int main() {
106     //1. Khai báo đồ thị G
107     Graph G;
108     //2. Đọc dữ liệu và dựng đồ thị
109     int n, m, u, v;
110     scanf("%d%d", &n, &m);
111     init_graph(&G, n);
112     for (int e = 0; e < m; e++) {
113         scanf("%d%d", &u, &v);
114         add_edge(&G, u, v);
115     }
116
117     //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
118     for (int u = 1; u <= G.n; u++) {
119         mark[u] = 0;
120         parent[u] = -1;
121     }
122
123     //4. Gọi hàm DFS
124     for (int u = 1; u <= G.n; u++)
125         if (mark[u] == 0)
126             BFS(&G, u);
127
128     for (int u = 1; u <= G.n; u++)
129         printf("%d %d\n", u, parent[u]);
130
131     return 0;
132 }
133
134 }
135
136

```

Correct

Marks for this submission: 1.00/1.00.



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

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

```
1 #include <stdio.h>
2 #define max 100
3
4 int mark[max];
5 int parent[max];
6 typedef struct{
7     int n,m;
8     int A[max][max];
9 }Graph;
10 typedef struct{
11     int u,p;
12 }ElementType;
13 typedef struct{
14     ElementType data[max];
15     int front,rear;
16 }Queue;
17
18 //khởi tạo hàng đợi
19 void make_null_queue (Queue *pQ){
20     pQ->front = 0;
21     pQ->rear = -1;
22 }
23
24 //dưa ptu vào cuối
25 void enqueue (Queue *pQ, ElementType u){
```



```

26     pQ->rear++;
27     pQ->data[pQ->rear] = u;
28 }
29
30 ElementType front (Queue *pQ){
31     return pQ->data[pQ->front];
32 }
33
34 void dequeue (Queue *pQ){
35     pQ->front++;
36 }
37
38 int empty (Queue *pQ){
39     return pQ->front > pQ->rear;
40 }
41
42 void init_graph (Graph *pG, int n){
43     pG->n = n;
44     pG->m = 0;
45     for (int u = 1; u <= n; u++){
46         for (int v = 1; v <= n; v++){
47             pG->A[u][v] = 0;
48         }
49     }
50 }
51
52 void add_edge(Graph *pG, int u, int v){
53     pG->A[u][v] = 1;
54     // pG->A[v][u] = 1;
55     pG->m++;
56 }
57 int adjacent(Graph *pG, int u, int v)
58 {
59     return pG->A[u][v] > 0;
60 }
61
62
63 void BFS(Graph *pG, int s){
64     Queue Q;
65     make_null_queue (&Q);
66
67     ElementType pair;
68     pair.u = s;
69     pair.p = -1;
70     enqueue(&Q,pair);
71     while (!empty(&Q)){
72         ElementType pair = front(&Q);
73         int u = pair.u;
74         int p = pair.p;
75         dequeue(&Q);
76
77         if (mark[u] != 0){
78             continue;
79         }
80         // printf ("%d\n", u);
81         mark[u] = 1;
82         parent[u]= p;
83         //duyet các cạnh kề của uu
84         for (int v = 1; v <= pG->n; v++){
85             if (adjacent(pG,u,v)){
86                 ElementType pair;
87                 pair.u =v;
88                 pair.p =u;
89                 enqueue (&Q,pair);
90             }
91         }
92     }
93 }
94
95 int main (){
96     Graph G;
97     int n, m, u, v;
98     scanf("%d%d", &n, &m);
99
100     init_graph(&G, n);
101     for (int i = 0; i < m; i++) {
102         scanf("%d%d", &u, &v);
103         add_edge(&G, u, v);
104     }
105
106     for (int i = 1; i <= n; i++) {
107         mark[i] = 0;
108         parent[i] = -1;
109     }
110
111     for (int i = 1; i <= n; i++) {
112         if (mark[i] == 0) {
113             BFS(&G, i);
114         }
115     }
116
117     for (int i = 1; i <= n; i++) {
118         printf("%d %d\n", i, parent[i]);
119     }

```



```

119 }
120 }
121 }
122 }

```

	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):

```

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 }
17
18 void add_edge(Graph *pG, int u, int v) {
19     pG->A[u][v] += 1;
20     //if (u != v)
21     //    pG->A[v][u] += 1;
22
23     pG->m++;
24 }
25
26 int adjacent(Graph *pG, int u, int v) {
27     return pG->A[u][v] > 0;
28 }
29
30 /* Khai báo CTDL Queue */
31
32 #define MAX_SIZE 100
33 typedef struct {
34     int u, p;
35 } ElementType;
36
37 typedef struct {
38     ElementType data[MAX_SIZE];
39     int front, rear;
40 } Queue;
41
42 /* Khởi tạo hàng đợi rỗng */
43

```



```

43 void make_null_queue(Queue *pQ) {
44     pQ->front = 0;
45     pQ->rear = -1;
46 }
47 /* Đưa phần tử u vào cuối hàng đợi */
48 void enqueue(Queue *pQ, ElementType u) {
49     pQ->rear++;
50     pQ->data[pQ->rear] = u;
51 }
52
53 /* Xem phần tử đầu hàng đợi */
54 ElementType front(Queue *pQ) {
55     return pQ->data[pQ->front];
56 }
57
58 /* Xóa bỏ phần tử đầu hàng đợi */
59 void dequeue(Queue *pQ) {
60     pQ->front++;
61 }
62
63 /* Kiểm tra hàng đợi rỗng */
64 int empty(Queue *pQ) {
65     return pQ->front > pQ->rear;
66 }
67
68
69 //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
70 int mark[MAX_N];
71 int parent[MAX_N];
72
73 void BFS(Graph *pG, int s) {
74     //1. Khai báo hàng đợi Q, khởi tạo rỗng
75     Queue Q;
76     make_null_queue(&Q);
77     //2. Đưa s vào Q, bắt đầu duyệt từ đỉnh s */
78     ElementType pair;
79     pair.u = s; pair.p = -1;
80     enqueue(&Q, pair);
81     //3. Vòng lặp chính dùng để duyệt
82     while (!empty(&Q)) {
83         //3a. Lấy phần tử ở đầu hàng đợi
84         ElementType pair = front(&Q); dequeue(&Q);
85         int u = pair.u, p = pair.p;
86         if (mark[u] != 0) //u đã duyệt rồi, bỏ qua
87             continue;
88
89         //printf("Duyet %d\n", u); //Làm gì đó trên u
90         mark[u] = 1; //Đánh dấu nó đã duyệt
91         parent[u] = p; //Đánh dấu nó đã duyệt
92
93         //3b. Xét các đỉnh kề của u, đưa vào hàng đợi Q
94         for (int v = 1; v <= pG->n; v++)
95             if (adjacent(pG, u, v)) {
96                 ElementType pair;
97                 pair.u = v; pair.p = u;
98                 enqueue(&Q, pair);
99             }
100     }
101 }
102
103
104
105 int main() {
106     //1. Khai báo đồ thị G
107     Graph G;
108     //2. Đọc dữ liệu và dựng đồ thị
109     int n, m, u, v;
110     scanf("%d%d", &n, &m);
111     init_graph(&G, n);
112     for (int e = 0; e < m; e++) {
113         scanf("%d%d", &u, &v);
114         add_edge(&G, u, v);
115     }
116
117     //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
118     for (int u = 1; u <= G.n; u++) {
119         mark[u] = 0;
120         parent[u] = -1;
121     }
122
123     //4. Gọi hàm DFS
124     for (int u = 1; u <= G.n; u++)
125         if (mark[u] == 0)
126             BFS(&G, u);
127
128
129     for (int u = 1; u <= G.n; u++)
130         printf("%d %d\n", u, parent[u]);
131
132
133     return 0;
134 }
135
136

```

Correct

Marks for this submission: 1.00/1.00.



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

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

```
1 #include <stdio.h>
2 #define max 100
3 typedef int ElementType;
4 int mark[max];
5 int parent[max];
6 typedef struct
7 {
8     int n, m;
9     int A[max][max];
10 } Graph;
11
12 void init_graph(Graph *pG, int n)
13 {
14     pG->n = n;
15     pG->m = 0;
16     for (int u = 1; u <= n; u++)
17     {
18         for (int v = 1; v <= n; v++)
19         {
20             pG->A[u][v] = 0;
21         }
22     }
23 }
24
```



```

25 void add_edge(Graph *pG, int u, int v)
26 {
27     pG->A[u][v] = 1;
28     pG->A[v][u] = 1;
29     pG->m++;
30 }
31
32 int adjacent(Graph *pG, int u, int v)
33 {
34     return pG->A[u][v] > 0;
35 }
36
37 typedef struct
38 {
39     ElementType data[max];
40     int top_idx;
41 } Stack;
42
43 // khởi tạo hàm rỗng
44 void make_null_stack(Stack *pS)
45 {
46     pS->top_idx = -1;
47 }
48
49 // thêm phần tử u vào đỉnh ngăn xếp
50 void push(Stack *pS, ElementType u)
51 {
52     pS->top_idx++;
53     pS->data[pS->top_idx] = u;
54 }
55
56 // xem phần tử trên đầu đỉnh ngăn xếp
57 ElementType top(Stack *pS)
58 {
59     return pS->data[pS->top_idx];
60 }
61
62 // xóa phần tử trên đỉnh ngăn xếp
63 void pop(Stack *pS)
64 {
65     pS->top_idx--;
66 }
67
68 // kiểm tra ngăn xếp rỗng
69 int empty(Stack *pS)
70 {
71     return pS->top_idx == -1;
72 }
73
74 // Duyệt cây duyệt đồ thị theo chiều sâu dùng đệ quy
75 void DFS(Graph *pG, int u, int p)
76 {
77     // 1. đánh dấu u đã duyệt
78     mark[u] = 1;
79     parent[u] = p;
80
81     // 2. xét các đỉnh kề của u
82     for (int v = 1; v <= pG->n; v++)
83         if (adjacent(pG, u, v) && mark[v] == 0)
84             DFS(pG, v, u);
85 }
86
87 int main()
88 {
89     Graph G;
90     int n, m, u, v;
91     scanf("%d%d", &n, &m);
92     init_graph(&G, n);
93
94     for (int e = 0; e < m; e++)
95     {
96         scanf("%d%d", &u, &v);
97         add_edge(&G, u, v);
98     }
99     for (int i = 1; i <= G.n; i++)
100         mark[i] = 0;
101
102     for (int i = 1; i <= G.n; i++)
103     {
104         if (mark[i] == 0)
105             DFS(&G, i, -1);
106     }
107
108     for (int i = 1; i <= n; i++)
109     {
110         printf("%d %d\n", i, parent[i]);
111     }
112     return 0;
113 }
114

```

	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):

```

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 }
17
18 void add_edge(Graph *pG, int u, int v) {
19     pG->A[u][v] += 1;
20     if (u != v)
21         pG->A[v][u] += 1;
22
23     pG->m++;
24 }
25
26 int adjacent(Graph *pG, int u, int v) {
27     return pG->A[u][v] > 0;
28 }
29
30
31
32
33 //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
34 int mark[MAX_N];
35 int parent[MAX_N];
36
37 void DFS(Graph *pG, int u, int p) {
38     //1. Đánh dấu u đã duyệt
39     //printf("Duyet %d\n", u); //Làm gì đó trên u
40     mark[u] = 1; //Đánh dấu nó đã duyệt
41     parent[u] = p; //Cho cha của u là p
42
43     //2. Xét các đỉnh kề của u
44     for (int v = 1; v <= pG->n; v++)
45         if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
46             DFS(pG, v, u); //gọi đệ quy duyệt nó
47
48 }
49
50

```

```

50
51
52
53 int main() {
54     //1. Khai báo đồ thị G
55     Graph G;
56     //2. Đọc dữ liệu và dựng đồ thị
57     int n, m, u, v;
58     scanf("%d%d", &n, &m);
59     init_graph(&G, n);
60     for (int e = 0; e < m; e++) {
61         scanf("%d%d", &u, &v);
62         add_edge(&G, u, v);
63     }
64
65     //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
66     for (int u = 1; u <= G.n; u++) {
67         mark[u] = 0;
68         parent[u] = -1;
69     }
70
71     //4. Gọi hàm DFS
72     for (int u = 1; u <= G.n; u++)
73         if (mark[u] == 0)
74             DFS(&G, u, -1);
75
76
77     for (int u = 1; u <= G.n; u++)
78         printf("%d %d\n", u, parent[u]);
79
80
81     return 0;
82 }
83

```

**Correct**

Marks for this submission: 1.00/1.00.

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

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

```
1 #include <stdio.h>
2 #define max 100
3 typedef int ElementType;
4 int mark[max];
5 int parent[max];
6 typedef struct
7 {
8     int n, m;
9     int A[max][max];
10 } Graph;
11
12 void init_graph(Graph *pG, int n)
13 {
14     pG->n = n;
15     pG->m = 0;
16     for (int u = 1; u <= n; u++)
17     {
18         for (int v = 1; v <= n; v++)
19         {
20             pG->A[u][v] = 0;
21         }
22     }
23 }
24
```





```

25 void add_edge(Graph *pG, int u, int v)
26 {
27     pG->A[u][v] = 1;
28     // pG->A[v][u] = 1;
29     pG->m++;
30 }
31
32 int adjacent(Graph *pG, int u, int v)
33 {
34     return pG->A[u][v] > 0;
35 }
36
37 typedef struct
38 {
39     ElementType data[max];
40     int top_idx;
41 } Stack;
42
43 // kkhởi tạo hàm rỗng
44 void make_null_stack(Stack *pS)
45 {
46     pS->top_idx = -1;
47 }
48
49 // thêm phần tử u vào đỉnh ngăn xếp
50 void push(Stack *pS, ElementType u)
51 {
52     pS->top_idx++;
53     pS->data[pS->top_idx] = u;
54 }
55
56 // xem phần tử trên đầu đỉnh ngăn xếp
57 ElementType top(Stack *pS)
58 {
59     return pS->data[pS->top_idx];
60 }
61
62 // xóa phần tử trên đỉnh ngăn xếp
63 void pop(Stack *pS)
64 {
65     pS->top_idx--;
66 }
67
68 // kiểm tra ngăn xếp rỗng
69 int empty(Stack *pS)
70 {
71     return pS->top_idx == -1;
72 }
73
74 // Duyệt cây duyệt đồ thị theo chiều sâu dùng đệ quy
75 void DFS(Graph *pG, int u, int p)
76 {
77     // 1. đánh dấu u đã duyệt
78     mark[u] = 1;
79     parent[u] = p;
80
81     // 2. xét các đỉnh kề của u
82     for (int v = 1; v <= pG->n; v++)
83         if (adjacent(pG, u, v) && mark[v] == 0)
84             DFS(pG, v, u);
85 }
86
87 int main()
88 {
89     Graph G;
90     int n, m, u, v;
91     scanf("%d%d", &n, &m);
92     init_graph(&G, n);
93
94     for (int e = 0; e < m; e++)
95     {
96         scanf("%d%d", &u, &v);
97         add_edge(&G, u, v);
98     }
99     for (int i = 1; i <= G.n; i++)
100         mark[i] = 0;
101
102     for (int i = 1; i <= G.n; i++)
103     {
104         if (mark[i] == 0)
105             DFS(&G, i, -1);
106     }
107
108     for (int i = 1; i <= n; i++)
109     {
110         printf("%d %d\n", i, parent[i]);
111     }
112     return 0;
113 }
114

```

	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):

```

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 }
17
18 void add_edge(Graph *pG, int u, int v) {
19     pG->A[u][v] += 1;
20     //if (u != v)
21     //    pG->A[v][u] += 1;
22
23     pG->m++;
24 }
25
26 int adjacent(Graph *pG, int u, int v) {
27     return pG->A[u][v] > 0;
28 }
29
30
31
32
33 //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt
34 int mark[MAX_N];
35 int parent[MAX_N];
36
37 void DFS(Graph *pG, int u, int p) {
38     //1. Đánh dấu u đã duyệt
39     //printf("Duyet %d\n", u); //Làm gì đó trên u
40     mark[u] = 1; //Đánh dấu nó đã duyệt
41     parent[u] = p; //Cho cha của u là p
42
43     //2. Xét các đỉnh kề của u
44     for (int v = 1; v <= pG->n; v++)
45         if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt
46             DFS(pG, v, u); //gọi đệ quy duyệt nó
47 }
48
49
50
51
52

```

```

52
53 int main() {
54     //1. Khai báo đồ thị G
55     Graph G;
56     //2. Đọc dữ liệu và dựng đồ thị
57     int n, m, u, v;
58     scanf("%d%d", &n, &m);
59     init_graph(&G, n);
60     for (int e = 0; e < m; e++) {
61         scanf("%d%d", &u, &v);
62         add_edge(&G, u, v);
63     }
64
65     //3. Khởi tạo mảng mark[u] = 0, với mọi u = 1, 2, ..., n
66     for (int u = 1; u <= G.n; u++) {
67         mark[u] = 0;
68         parent[u] = -1;
69     }
70
71     //4. Gọi hàm DFS
72     for (int u = 1; u <= G.n; u++)
73         if (mark[u] == 0)
74             DFS(&G, u, -1);
75
76
77     for (int u = 1; u <= G.n; u++)
78         printf("%d %d\n", u, parent[u]);
79
80
81     return 0;
82 }
83
84

```

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 ▶

