Started on	Friday, 4 July 2025, 11:33 PM
State	Finished
Completed on	Saturday, 5 July 2025, 11:29 PM
Time taken	23 hours 56 mins
Marks	1.00/1.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Viết chương trình xếp hạng cho đồ thị có hướng không chu trình.

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 hạng của các đỉnh theo thứ tự của đỉnh, mỗi đỉnh trên 1 dòng:

Hạng đỉnh 1

Hạng đỉnh 2

...

Hạng đỉnh n

Xem thêm ví dụ bên dưới. Trong ví dụ đầu tiên ta có: hạng của 1 = 0, hạng của 2 = 2 và hạng của 3 = 1.

Chú ý:

- 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%d", &u, &v);
        add_edge(&G, u, v);
}
```

For example:

Input	Result
3 2 1 3 3 2	0 2 1
7 10 1 2 1 3 1 4 2 3 2 6 3 7 4 5 5 3 5 7 6 7	0 1 3 1 2 2 4

Input	Result
7 12 1 2 1 3 2 4 2 5 2 6 3 2 3 5 3 6 4 7 5 7 6 4 6 5	0 2 1 4 4 3 5

Answer: (penalty regime: 33.3, 66.7, ... %)

```c
#include <stdio.h>
#define MAX_N 100
int rank[MAX_N];

//.....................List.............................
typedef struct{
    int size;
    int data[MAX_N];
}List;

//khoi tao list rong
void init_list (List *pL){
    pL->size = 0;
}

//thêm ptu x vào cuối ds
void push_back (List *pL, int x){
    pL->data[pL->size] = x;
    pL->size++;
}

```

Debug: source code from all test runs

Run 1

```c
#include <stdio.h>
#define MAX_N 100
int rank[MAX_N];

//.....................List.............................
typedef struct{
    int size;
    int data[MAX_N];
}List;

//khoi tao list rong
void init_list (List *pL){
    pL->size = 0;
}

//thêm ptu x vào cuối ds
void push_back (List *pL, int x){
    pL->data[pL->size] = x;
    pL->size++;
}

//số ptu hiện có
int size (List *pL){
    return pL->size;
}

//lấy gtri ptu tai chi so i
int element_at (List *pL, int i){
    if (i >= 0 && i < pL->size){
        return pL->data[i];
    }
    else{
        return -1;
    }
}

//hàm copy các ptu từ pL2 vào pL1
void copy_list (List *pL1, List *pL2){
    pL1->size = pL2->size;
    for (int i = 0; i < pL2->size; i++){
        pL1->data[i] = pL2->data[i];
    }
}

//.................Graph................
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;
    pG->m++;
}
```

```
//.................Rank.....................
void topo_rank (Graph *pG, List *pL){
    //khởi tạo
    int deg[MAX_N];
    List S1, S2;
    //s1 chứa đỉnh hiện tại có indeg = 0
    //s2 chứa mới có indeg = 0 sau khi đã xử lý các đỉnh trong s1
    init_list(&S1);
    init_list(&S2);

    //tính bậc cho tất cả đỉnh (đây là cách tính bậc vào in_deg) = cách đếm số cạnh đi vào u
    for (int u = 1; u <= pG->n; u++){
        deg[u] = 0;
        for (int x = 1; x <= pG->n; x++){
            if (pG->A[x][u] != 0){  //nếu tồn tại cạnh đi từ x -> u thì tăng bậc lên 1
                deg[u]++;
            }
        }
    }

    //thêm các đỉnh u có deg[u] == 0 vào S1
    for (int u = 1; u <= pG->n; u++){
        if (deg[u] == 0){
            push_back(&S1, u);
        }
    }

    int k = 0;
    while (S1.size > 0){
        init_list(&S2);
        //init lần 2 là vì trong mỗi lần lặp, S2 phải làm mới để chứa các đỉnh thuộc tầng kế tiếp, thay
thế S1 ở vòng sau.
        for (int i = 0; i < S1.size; i++){
            int u = element_at(&S1, i);
            rank[u] = k;

            //cứ tính hạng 1 đỉnh thì sẽ xóa bỏ đỉnh đó
            for (int v = 1; v <= pG->n; v++){
                if (pG->A[u][v] != 0){
                    deg[v]--;  //xóa canh u->v bằng cách giảm bậc
                    if (deg[v] == 0){
                        push_back(&S2,v); //nếu hiện tại v có bậc = 0 thì thêm vào s2
                    }
                }
            }
        }
    k++;
    copy_list (&S1, &S2);
    }
}

int main (){
    int n,m,u,v;
    Graph G;
    List L;
    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);
    }

    topo_rank(&G, &L);
    for (int u = 1; u <= n; u++){
```

```
        printf ("%d\n", rank[u]);
    }
    return 0;
}
```

Run 2

```c
#include <stdio.h>
#define MAX_N 100
int rank[MAX_N];

//.....................List.............................
typedef struct{
    int size;
    int data[MAX_N];
}List;

//khoi tao list rong
void init_list (List *pL){
    pL->size = 0;
}

//thêm ptu x vào cuối ds
void push_back (List *pL, int x){
    pL->data[pL->size] = x;
    pL->size++;
}

//số ptu hiện có
int size (List *pL){
    return pL->size;
}

//lấy gtri ptu tai chi so i
int element_at (List *pL, int i){
    if (i >= 0 && i < pL->size){
        return pL->data[i];
    }
    else{
        return -1;
    }
}

//hàm copy các ptu từ pL2 vào pL1
void copy_list (List *pL1, List *pL2){
    pL1->size = pL2->size;
    for (int i = 0; i < pL2->size; i++){
        pL1->data[i] = pL2->data[i];
    }
}

//.................Graph................
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;
    pG->m++;
}
```

```
//.................Rank.....................
void topo_rank (Graph *pG, List *pL){
    //khởi tạo
    int deg[MAX_N];
    List S1, S2;
    //s1 chứa đỉnh hiện tại có indeg = 0
    //s2 chứa mới có indeg = 0 sau khi đã xử lý các đỉnh trong s1
    init_list(&S1);
    init_list(&S2);

    //tính bậc cho tất cả đỉnh (đây là cách tính bậc vào in_deg) = cách đếm số cạnh đi vào u
    for (int u = 1; u <= pG->n; u++){
        deg[u] = 0;
        for (int x = 1; x <= pG->n; x++){
            if (pG->A[x][u] != 0){  //nếu tồn tại cạnh đi từ x -> u thì tăng bậc lên 1
                deg[u]++;
            }
        }
    }

    //thêm các đỉnh u có deg[u] == 0 vào S1
    for (int u = 1; u <= pG->n; u++){
        if (deg[u] == 0){
            push_back(&S1, u);
        }
    }

    int k = 0;
    while (S1.size > 0){
        init_list(&S2);
        //init lần 2 là vì trong mỗi lần lặp, S2 phải làm mới để chứa các đỉnh thuộc tầng kế tiếp, thay
thế S1 ở vòng sau.
        for (int i = 0; i < S1.size; i++){
            int u = element_at(&S1, i);
            rank[u] = k;

            //cứ tính hạng 1 đỉnh thì sẽ xóa bỏ đỉnh đó
            for (int v = 1; v <= pG->n; v++){
                if (pG->A[u][v] != 0){
                    deg[v]--;  //xóa canh u->v bằng cách giảm bậc
                    if (deg[v] == 0){
                        push_back(&S2,v); //nếu hiện tại v có bậc = 0 thì thêm vào s2
                    }
                }
            }
        }
    k++;
    copy_list (&S1, &S2);
    }
}

int main (){
    int n,m,u,v;
    Graph G;
    List L;
    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);
    }

    topo_rank(&G, &L);
    for (int u = 1; u <= n; u++){
```

```
        printf ("%d\n", rank[u]);
    }
    return 0;
}
```

Run 3

```c
#include <stdio.h>
#define MAX_N 100
int rank[MAX_N];

//.....................List.............................
typedef struct{
    int size;
    int data[MAX_N];
}List;

//khoi tao list rong
void init_list (List *pL){
    pL->size = 0;
}

//thêm ptu x vào cuối ds
void push_back (List *pL, int x){
    pL->data[pL->size] = x;
    pL->size++;
}

//số ptu hiện có
int size (List *pL){
    return pL->size;
}

//lấy gtri ptu tai chi so i
int element_at (List *pL, int i){
    if (i >= 0 && i < pL->size){
        return pL->data[i];
    }
    else{
        return -1;
    }
}

//hàm copy các ptu từ pL2 vào pL1
void copy_list (List *pL1, List *pL2){
    pL1->size = pL2->size;
    for (int i = 0; i < pL2->size; i++){
        pL1->data[i] = pL2->data[i];
    }
}

//.................Graph................
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;
    pG->m++;
}
```

```
//.................Rank.....................
void topo_rank (Graph *pG, List *pL){
    //khởi tạo
    int deg[MAX_N];
    List S1, S2;
    //s1 chứa đỉnh hiện tại có indeg = 0
    //s2 chứa mới có indeg = 0 sau khi đã xử lý các đỉnh trong s1
    init_list(&S1);
    init_list(&S2);

    //tính bậc cho tất cả đỉnh (đây là cách tính bậc vào in_deg) = cách đếm số cạnh đi vào u
    for (int u = 1; u <= pG->n; u++){
        deg[u] = 0;
        for (int x = 1; x <= pG->n; x++){
            if (pG->A[x][u] != 0){  //nếu tồn tại cạnh đi từ x -> u thì tăng bậc lên 1
                deg[u]++;
            }
        }
    }

    //thêm các đỉnh u có deg[u] == 0 vào S1
    for (int u = 1; u <= pG->n; u++){
        if (deg[u] == 0){
            push_back(&S1, u);
        }
    }

    int k = 0;
    while (S1.size > 0){
        init_list(&S2);
        //init lần 2 là vì trong mỗi lần lặp, S2 phải làm mới để chứa các đỉnh thuộc tầng kế tiếp, thay
thế S1 ở vòng sau.
        for (int i = 0; i < S1.size; i++){
            int u = element_at(&S1, i);
            rank[u] = k;

            //cứ tính hạng 1 đỉnh thì sẽ xóa bỏ đỉnh đó
            for (int v = 1; v <= pG->n; v++){
                if (pG->A[u][v] != 0){
                    deg[v]--;  //xóa canh u->v bằng cách giảm bậc
                    if (deg[v] == 0){
                        push_back(&S2,v); //nếu hiện tại v có bậc = 0 thì thêm vào s2
                    }
                }
            }
        }
    k++;
    copy_list (&S1, &S2);
    }
}

int main (){
    int n,m,u,v;
    Graph G;
    List L;
    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);
    }

    topo_rank(&G, &L);
    for (int u = 1; u <= n; u++){
```

```
        printf ("%d\n", rank[u]);
    }
    return 0;
}
```

	Input	Expected	Got	
✔	3 2 1 3 3 2	0 2 1	0 2 1	✔
✔	7 10 1 2 1 3 1 4 2 3 2 6 3 7 4 5 5 3 5 7 6 7	0 1 3 1 2 2 4	0 1 3 1 2 2 4	✔
✔	7 12 1 2 1 3 2 4 2 5 2 6 3 2 3 5 3 6 4 7 5 7 6 4 6 5	0 2 1 4 4 3 5	0 2 1 4 4 3 5	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

◄ Chia kẹo

Jump to...

Thuật toán xếp hạng đồ thị ►