Started on	Tuesday, 1 July 2025, 10:52 PM
State	Finished
Completed on	Tuesday, 1 July 2025, 10:53 PM
Time taken	9 secs
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
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

//__________LIST___________
typedef struct
{
    int data[MAX_N]; // mảng bao gồm các phần tử của danh sách
    int size;        // độ dài của danh sách
} List;

void make_null_list(List *pL)
{
    pL->size = 0;
    // (*L).size=0;
}

// Thêm một phần tử mới 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

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

//__________LIST__________
typedef struct
{
    int data[MAX_N]; // mảng bao gồm các phần tử của danh sách
    int size;        // độ dài của danh sách
} List;

void make_null_list(List *pL)
{
    pL->size = 0;
    // (*L).size=0;
}

// Thêm một phần tử mới vào cuối ds
void push_back(List *pL, int x)
{

    // pL->data[pL->size] = x;
    // pL->size++;
    pL->data[pL->size++] = x;
}

// Trả về phần tử ở vị trí p
int element_at(List *pL, int i)
{
    return pL->data[i - 1];
}

void copy_list(List *pS1, List *pS2)
{
    make_null_list(pS1);
    for (int i = 1; i <= pS2->size; i++)
        push_back(pS1, element_at(pS2, 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->A[v][u] = 1;
    pG->m++;
}

//_____________________xếp hạng_____________________________
int r[MAX_N]; // hàm xếp hạng
void rank(Graph *pG)
{
    int d[MAX_N]; // luu bac vao cua dinh u

    // tính bậc vào của đỉnh d[u]
    for (int u = 1; u <= pG->n; u++)
    {
        d[u] = 0;
        for (int x = 1; x <= pG->n; x++)
            if (pG->A[x][u] != 0) // bậc vào của u
                d[u]++;
    }

    // Sử dụng 2 danh sách S1, S2
    List S1, S2;
```

```
    // Tìm gốc, đưa vào S1
    make_null_list(&S1); // khởi tạo rỗng cho s1
    for (int u = 1; u < pG->n; u++)
        if (d[u] == 0)
            push_back(&S1, u);

    int k = 0; // hạng tính từ 0. Tùy theo bài toán có thể k = 1

    // vòng lặp chính, lặp đến khi S1 rỗng thì dừng
    while (S1.size > 0)
    {
        make_null_list(&S2); // khởi tạo rỗng cho s2
        for (int i = 1; i <= S1.size; i++)
        {
            int u = element_at(&S1, i); // lấy các gốc u trong S1 ra
            r[u] = k;                   // xếp hạng cho u

            // xóa đỉnh u <=> giảm bậc vào của các đỉnh kề v của u
            for (int v = 1; v <= pG->n; v++)
                if (pG->A[u][v] != 0)
                {
                    d[v]--;
                    if (d[v] == 0)
                        push_back(&S2, v);
                }
        }
        copy_list(&S1, &S2); // copy s2 vào s1
        k++;                 // tăng hạng kế tiếp cho các gốc mới
    }
}

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

    rank(&G);
    for (int u = 1; u <= n; u++)
        printf("%d\n",r[u]);
    return 0;
}
```

Run 2

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

//__________LIST__________
typedef struct
{
    int data[MAX_N]; // mảng bao gồm các phần tử của danh sách
    int size;        // độ dài của danh sách
} List;

void make_null_list(List *pL)
{
    pL->size = 0;
    // (*L).size=0;
}

// Thêm một phần tử mới vào cuối ds
void push_back(List *pL, int x)
{

    // pL->data[pL->size] = x;
    // pL->size++;
    pL->data[pL->size++] = x;
}

// Trả về phần tử ở vị trí p
int element_at(List *pL, int i)
{
    return pL->data[i - 1];
}

void copy_list(List *pS1, List *pS2)
{
    make_null_list(pS1);
    for (int i = 1; i <= pS2->size; i++)
        push_back(pS1, element_at(pS2, 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->A[v][u] = 1;
    pG->m++;
}

//_____________________xếp hạng_____________________________
int r[MAX_N]; // hàm xếp hạng
void rank(Graph *pG)
{
    int d[MAX_N]; // luu bac vao cua dinh u

    // tính bậc vào của đỉnh d[u]
    for (int u = 1; u <= pG->n; u++)
    {
        d[u] = 0;
        for (int x = 1; x <= pG->n; x++)
            if (pG->A[x][u] != 0) // bậc vào của u
                d[u]++;
    }

    // Sử dụng 2 danh sách S1, S2
    List S1, S2;
```

```
    // Tìm gốc, đưa vào S1
    make_null_list(&S1); // khởi tạo rỗng cho s1
    for (int u = 1; u < pG->n; u++)
        if (d[u] == 0)
            push_back(&S1, u);

    int k = 0; // hạng tính từ 0. Tùy theo bài toán có thể k = 1

    // vòng lặp chính, lặp đến khi S1 rỗng thì dừng
    while (S1.size > 0)
    {
        make_null_list(&S2); // khởi tạo rỗng cho s2
        for (int i = 1; i <= S1.size; i++)
        {
            int u = element_at(&S1, i); // lấy các gốc u trong S1 ra
            r[u] = k;                   // xếp hạng cho u

            // xóa đỉnh u <=> giảm bậc vào của các đỉnh kề v của u
            for (int v = 1; v <= pG->n; v++)
                if (pG->A[u][v] != 0)
                {
                    d[v]--;
                    if (d[v] == 0)
                        push_back(&S2, v);
                }
        }
        copy_list(&S1, &S2); // copy s2 vào s1
        k++;                 // tăng hạng kế tiếp cho các gốc mới
    }
}

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

    rank(&G);
    for (int u = 1; u <= n; u++)
        printf("%d\n",r[u]);
    return 0;
}
```

Run 3

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

//__________LIST__________
typedef struct
{
    int data[MAX_N]; // mảng bao gồm các phần tử của danh sách
    int size;        // độ dài của danh sách
} List;

void make_null_list(List *pL)
{
    pL->size = 0;
    // (*L).size=0;
}

// Thêm một phần tử mới vào cuối ds
void push_back(List *pL, int x)
{

    // pL->data[pL->size] = x;
    // pL->size++;
    pL->data[pL->size++] = x;
}

// Trả về phần tử ở vị trí p
int element_at(List *pL, int i)
{
    return pL->data[i - 1];
}

void copy_list(List *pS1, List *pS2)
{
    make_null_list(pS1);
    for (int i = 1; i <= pS2->size; i++)
        push_back(pS1, element_at(pS2, 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->A[v][u] = 1;
    pG->m++;
}

//_____________________xếp hạng_____________________________
int r[MAX_N]; // hàm xếp hạng
void rank(Graph *pG)
{
    int d[MAX_N]; // luu bac vao cua dinh u

    // tính bậc vào của đỉnh d[u]
    for (int u = 1; u <= pG->n; u++)
    {
        d[u] = 0;
        for (int x = 1; x <= pG->n; x++)
            if (pG->A[x][u] != 0) // bậc vào của u
                d[u]++;
    }

    // Sử dụng 2 danh sách S1, S2
    List S1, S2;
```

```
    // Tìm gốc, đưa vào S1
    make_null_list(&S1); // khởi tạo rỗng cho s1
    for (int u = 1; u < pG->n; u++)
        if (d[u] == 0)
            push_back(&S1, u);

    int k = 0; // hạng tính từ 0. Tùy theo bài toán có thể k = 1

    // vòng lặp chính, lặp đến khi S1 rỗng thì dừng
    while (S1.size > 0)
    {
        make_null_list(&S2); // khởi tạo rỗng cho s2
        for (int i = 1; i <= S1.size; i++)
        {
            int u = element_at(&S1, i); // lấy các gốc u trong S1 ra
            r[u] = k;                   // xếp hạng cho u

            // xóa đỉnh u <=> giảm bậc vào của các đỉnh kề v của u
            for (int v = 1; v <= pG->n; v++)
                if (pG->A[u][v] != 0)
                {
                    d[v]--;
                    if (d[v] == 0)
                        push_back(&S2, v);
                }
        }
        copy_list(&S1, &S2); // copy s2 vào s1
        k++;                 // tăng hạng kế tiếp cho các gốc mới
    }
}

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

    rank(&G);
    for (int u = 1; u <= n; u++)
        printf("%d\n",r[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	✔

	Input	Expected	Got	
✔	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 ►