# [Thực hành buổi 1](https://else.ctu.edu.vn/course/view.php?id=54#section-5)

## Bài 1 - Ma trận đỉnh - cung (add\_edge)

Cho cấu trúc dữ liệu đồ thị được khai báo sử dụng ma trận đỉnh - cung như sau:

typedef struct {

int A[100][500];

int n, m;

} Graph;

Giả sử đồ thị vô hướng, không chứa khuyên.

Viết hàm add\_edge(Graph\* G, int e, int x, int y) để thêm cung e = (x, y) vào đồ thị G.

void add\_edge(Graph\* G, int e, int x, int y) {

}

**Answer**

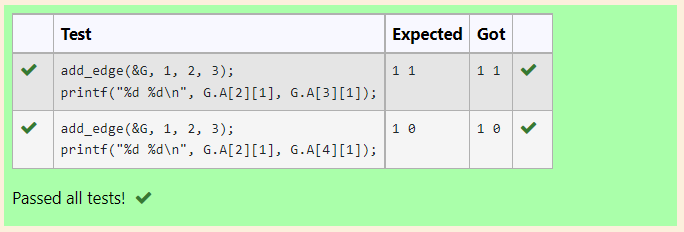
void add\_edge(Graph\* G, int e, int x, int y) {

//e la stt canh: x, y la dinh duoc noi boi dinh e

G->A[x][e]=1; //e co dinh la x ->; e ko la canh cua dinh x ->0

G->A[y][e]=1;

}



## Bài 2 - Ma trận đỉnh - cung (neighbors)

Cho cấu trúc dữ liệu đồ thị được khai báo sử dụng ma trận đỉnh - cung như sau:

typedef struct {

int A[100][500];

int n, m;

} Graph;

Giả sử đồ thị không chứa khuyên.

Viết hàm List neighbors(Graph\* G, int x) trả về danh sách các đỉnh kề của x.

Chú ý: các đỉnh kề của x được sắp xếp theo thứ tự tăng dần và không trùng nhau. Ví dụ: nếu các đỉnh kề của 1 là 4 và 2 thì danh sách trả về chứa: 2 và 4.

Cấu trúc dữ liệu List được định nghĩa như bên dưới:

#define MAX\_ELEMENTS 100

typedef int ElementType;

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

**Answer**

//A[dinh][cung]

//Khong lap dinh ke

List neighbors (Graph \*G, int x){ //dinh x

int y, e;

List list;

make\_null (&list);

for (y=1; y<= G->n; y++){ //Chay tu dinh 1 den dinh n

for (e=1; e<= G->m; e++){ //Chay tu canh 1 den canh n

if (x==y) continue;

if (G->A[x][e] !=0 && G->A[y][e] !=0){

push\_back(&list, y);

break;

}

}

}

return list; //in danh sach dinh ke

}

//Lap dinh ke

/\*List neighbors (Graph \*G, int x){ //dinh x

int y, e;

List list;

make\_null (&list);

for (y=1; y<= G->n; y++){ //Chay tu dinh 1 den dinh n

for (e=1; e<= G->m; e++){ //Chay tu canh 1 den canh n

if (x==y) continue;

if (G->A[x][e] !=0 && G->A[y][e] !=0){

push\_back(&list, y);

//break;

}

}

}

return list; //in danh sach dinh ke

}

\*/

#### Question author's solution (C):

List neighbors(Graph\* G, int x) {

List L;

make\_null(&L);

int e, y;

for (y = 1; y <= G->n; y++) {

if (x == y) continue;

for (e = 1; e <= G->m; e++)

if (G->A[x][e] > 0 && G->A[y][e] > 0) {

push\_back(&L, y);

break;

}

}

return L;

}



## Bài 3 - Ma trận đỉnh - đỉnh (degree)

Cho cấu trúc dữ liệu đồ thị được khai báo sử dụng ma trận đỉnh - đỉnh như sau:

typedef struct {

int A[100][100];

int n;

} Graph;

Giả sử đồ thị vô hướng, không chứa khuyên, không chứa đa cung

Viết hàm int deg(Graph\* G, int x) để tính bậc của đỉnh x.

int deg(Graph\* G, int x) {

}

**Answer**

int deg(Graph\* G, int x) {

//dinh x, dinh y

int y, t=0;

for (y=1; y<=G->n; y++){

if (x==y) continue;

if (G->A[x][y]!=0 || G->A[y][x]!=0) t++;

}

return t;

}



## Bài 4 - Bậc lớn nhất

Cho một đồ thị vô hướng, không khuyên, không đa cung có n đỉnh và m cung.

Viết chương trình tính và in ra màn hình đỉnh có bậc lớn nhất và bậc tương ứng của nó. Nếu có nhiều đỉnh có bật bằng nhau thì in ra đỉnh có số thứ tự nhỏ 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 màn hình đỉnh có bậc lớn nhất và bậc của nó.

Xem thêm ví dụ bên dưới.

Trong ví dụ đầu tiên ta có:

* Bậc của đỉnh 1 là 1,
* Bậc của đỉnh 2 là 3
* Bậc của đỉnh 3 là 2
* Bậc của đỉnh 4 là 2

Vậy đỉnh có bậc lớn nhất là đỉnh 2 và bậc của nó là 3. Vì thế ta in ra:

2 3

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);

}

**Answer**

//n dinh, m cung

//ma tran dinh-dinh

#include<stdio.h>

#define max\_vertices 100 // max dinh

#define max\_edges 500 // max canh

typedef struct{

int n; /\*n: so dinh\*/

int A[max\_vertices][max\_vertices];

}Graph;

//Khoi tao do th G co n dinh, m cung

void init\_graph(Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

//Ko da cung, ko khuyen, vo huong

void add\_edge(Graph\* G, int x, int y){

G->A[x][y] =1;

G->A[y][x] =1;

}

void maxDegree(Graph\* G){

//x dinh, y dinh

int dinhmax=0, bacmax=0;

int x;

//xet dinh x

for (x=1; x<= G->n; x++){

//xet qua tat ca dinh ke cua tung x de dem bac

int y, deg =0;

for (y=1; y<=G->n; y++){

deg += G->A[x][y];

}

//printf("%d %d\n", x, deg);

if (deg>bacmax) {

bacmax=deg;

dinhmax=x;

}

}

printf("%d %d", dinhmax, bacmax);

}

int main (){

//freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.

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);

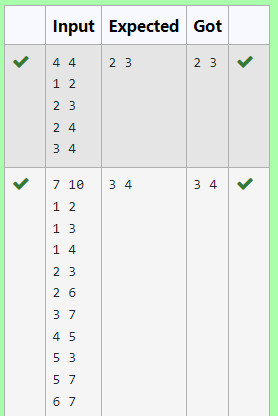
add\_edge(&G, u, v);

}

maxDegree(&G);

return 0;

}



## Bài 5 - Danh sách cung

Cho cấu trúc dữ liệu đồ thị được cài đặt bằng phương pháp "Danh sách cung" như sau:

typedef struct {

int x, y;

} Edge;

typedef struct {

int n, m;

Edge edges[MAX\_EDGES];

} Graph;

Các cung được lưu trong danh sách edges với chỉ số từ 0, 1, 2, ..., m-1

Hàm khởi tạo đồ thị:

void init\_graph(Graph\* G, int n){

G->n = n;

G->m = 0;

}

Viết hàm add\_edge(Graph\* G, int x, int y) để thêm cung (x, y) vào đồ thị G.

void add\_edge(Graph\* G, int x, int y) {

}

**Answer**

//dinh x, y

//n dinh, m cung

void add\_edge(Graph\* G, int x, int y) {

G->edges[G->m].x=x;

G->edges[G->m].y=y;

G->m = G->m +1;

}



Cho cấu trúc dữ liệu đồ thị được cài đặt bằng phương pháp "Danh sách cung" như sau:

typedef struct {

int x, y;

} Edge;

typedef struct {

int n, m;

Edge edges[MAX\_EDGES];

} Graph;

Các cung được lưu trong danh sách edges với chỉ số từ 0, 1, 2, ..., m-1

Hàm khởi tạo đồ thị:

void init\_graph(Graph\* G, int n){

G->n = n;

G->m = 0;

}

Viết hàm add\_edge(Graph\* G, int x, int y) để thêm cung (x, y) vào đồ thị G.

void add\_edge(Graph\* G, int x, int y) {

}

Chú ý:

- Nếu cung (x, y) không hợp lệ (vd: x < 1, y > n, ...) thì bỏ qua không làm gì cả.

**Answer**

//dinh x, y

//n dinh, m cung

void add\_edge(Graph\* G, int x, int y) {

if (x>=1 && x<=G->n && y>=1 && y<=G->n){

G->edges[G->m].x=x;

G->edges[G->m].y=y;

G->m = G->m +1;

}

}



## Bài 6 - Đọc đồ thị từ tập tin

Hãy viết chương trình đọc đồ thị từ tập tin và hiển thị ma trận kề của đồ thị này.

Giả sử đồ thị được cho là đồ thị vô hướng đơn.

Đầu vào (Input):

Dữ liệu đầu vào được nhập từ tập tin dt1.txt 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 Ma trận kề (0/1) của đồ thị

Ví dụ:

Nếu đầu vào là:

4 4

1 2

1 3

2 3

3 4

Kết quả là:

0 1 1 0

1 0 1 0

1 1 0 1

0 0 1 0

**Answer**

//ma tran dinh-dinh

#include<stdio.h>

#define max\_vertices 100 // max dinh

typedef struct{

int n; /\*n: so dinh\*/

int A[max\_vertices][max\_vertices];

}Graph;

//Khoi tao do th G co n dinh, m cung

void init\_graph(Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge(Graph\* G, int x, int y){

G->A[x][y] =1;

G->A[y][x] =1;

}

int main (){

Graph G;

int n, m, e, u, v; //n dinh, m cung //u dinh, m cung

FILE\* file =fopen("dt1.txt", "r");

fscanf(file, "%d%d", &n, &m);

init\_graph (&G, n);

for (e=1; e<=m; e++){

fscanf(file, "%d%d", &u, &v);

add\_edge (&G, u, v);

}

int i,j;

for (i=1; i<=G.n; i++){

for (j=1; j<=G.n; j++){

printf ("%d ", G.A[i][j]);

}

printf("\n");

}

return 0;

}



# [Thực hành buổi](https://else.ctu.edu.vn/course/view.php?id=54#section-5) 2

## Bài 1 - Duyệt đồ thị

Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.

Nếu đồ thị không liên thông, 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 nói rằng mô tả cung (u, v).

Đầu ra (Output):

In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng.

Xem thêm các ví dụ bên dưới.

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  2  3  4 |
| 4 2  1 2  3 4 | 1  2  3  4 |
| 4 2  1 4  2 3 | 1  4  2  3 |
| 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  2  4  12  6  7  10  11  5  13  3  8  9 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int data[max\_elements];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){

return Q->data[Q->front];

}

void pop(Queue\* Q){

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

void breath\_first\_search(Graph\* G, int mark[], Queue\* L, int n) {

int v;

push(L,n);

while (!empty(L)) {

int u = top(L);

pop(L);

if (mark[u] != 0) continue;

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

mark[u]=1;

for (v = 1; v <= G->n; v++) {

if (G->A[u][v]==1) {

push(L, v);

}

}

}

}

void BFS(Graph\* G) {

Queue L;

int mark[max\_vertices];

make\_null\_queue(&L);

int j;

for (j = 1; j <= G->n; j++) mark[j] = 0;

int t;

for (t=1; t<=G->n; t++){ //Duyet tu 1, duyet 1 truoc

if(mark[t]==0) breath\_first\_search (G, mark, &L, t);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

BFS(&G);

}



—----------------------------

Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều sâu (sử dụng ĐỆ QUY) bắt đầu từ đỉnh 1.

Nếu đồ thị không liên thông, 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 nói rằng mô tả cung (u, v).

Đầu ra (Output):

In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng.

Xem thêm các ví dụ bên dưới.

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  2  4  3 |
| 4 2  1 2  3 4 | 1  2  3  4 |
| 4 2  1 4  2 3 | 1  4  2  3 |
| 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  2  4  6  5  8  9  7  3  13  12  10  11 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

void traversal(Graph\* G, int x) {

int y;

if (mark[x] == 1) return;

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

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

traversal(G, y);

}

}

}

void DFS(Graph\* G) {

int j;

for (j = 1; j <= G->n; j++) mark[j] = 0;

int t;

for (t=1; t<=G->n; t++){

if(mark[t]==0) traversal(G, t);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

DFS(&G);

}



—---------------------------------

Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều sâu (sử dụng NGĂN XẾP) bắt đầu từ đỉnh 1.

Nếu đồ thị không liên thông, 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 nói rằng mô tả cung (u, v).

Đầu ra (Output):

In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng.

Xem thêm các ví dụ bên dưới.

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  3  2  4 |
| 4 2  1 2  3 4 | 1  2  3  4 |
| 4 2  1 4  2 3 | 1  4  2  3 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

void depth\_first\_search(Graph\* G, int mark[], Stack\* L, int n) {

int y;

push(L, n);

while (!empty(L)) {

int x = top(L); pop(L);

if (mark[x] != 0) continue;

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

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

push(L, y);

}

}

}

}

void DFS(Graph\* G) {

Stack L;

int mark[max\_vertices];

make\_null\_stack(&L);

int j;

for (j = 1; j <= G->n; j++) mark[j] = 0;

int t;

for (t=1; t<=G->n; t++){

if(mark[t]==0) depth\_first\_search(G, mark, &L, t);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

DFS(&G);

}



## Bài 2 - Duyệt đồ thị & Dựng cây duyệt đồ thị

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 đồ thị không liên thông, 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 nói rằng 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>

Quy ước: Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 0.

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 3  2 4 | 1 0  2 1  3 1  4 2 |
| 4 2  1 2  3 4 | 1 0  2 1  3 0  4 3 |
| 4 2  1 4  2 3 | 1 0  2 0  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 0  2 1  3 7  4 1  5 6  6 4  7 4  8 5  9 5  10 12  11 12  12 1  13 6 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int data[max\_elements];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){

return Q->data[Q->front];

}

void pop(Queue\* Q){

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

void breath\_first\_search(Graph\* G, int mark[], int parent[], Queue\* L, int n) {

int v;

push(L,n);

parent[n]=0;

while (!empty(L)) {

int u = top(L);

pop(L);

if (mark[u] != 0) continue;

//printf ("%d\n", u);

mark[u]=1;

for (v = 1; v <= G->n; v++) {

if (G->A[u][v]==1) {

push(L, v);

if (parent[v] == -1) parent[v]=u;

}

}

}

}

void BFS(Graph\* G) {

Queue L;

int mark[max\_vertices];

int parent[max\_vertices];

make\_null\_queue(&L);

int j;

for (j = 1; j <= G->n; j++){

mark[j] = 0;

parent[j]=-1;

}

int t;

for (t=1; t<=G->n; t++){ //Duyet tu 1, duyet 1 truoc

if(mark[t]==0) breath\_first\_search (G, mark, parent, &L, t);

}

for (t=1; t<=G->n; t++){

printf ("%d %d\n", t, parent[t]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

BFS(&G);

}



—-------------------------------------

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

Nếu đồ thị không liên thông, 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 nói rằng 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>

Quy ước: Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 0.

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.
* Sử dụng 1 stack lưu các cặp <u, parent>. Khi xét đỉnh v (là kề của u) để đưa vào stack, ta đưa cả <v, u> vào stack. Lúc này u được xem như là parent của v.

typedef struct {

int u;

int parent;

} ELEMENT\_TYPE;

* 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 3  2 4 | 1 0  2 1  3 1  4 2 |
| 4 2  1 2  3 4 | 1 0  2 1  3 0  4 3 |
| 4 2  1 4  2 3 | 1 0  2 0  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 0  2 4  3 7  4 1  5 6  6 7  7 4  8 9  9 5  10 11  11 12  12 1  13 6 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

void depth\_first\_search(Graph\* G, int mark[], int parent[], Stack\* L, int n) {

int y;

push(L, n);

parent[n]=0;

while (!empty(L)) {

int x = top(L); pop(L);

if (mark[x] != 0) continue; //nut da danh dau bo qua

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

push(L, y);

if (mark[y] == 0) parent[y]=x; //parent chua xac dinh va luon thay doi den khi duoc danh dau

}

}

}

}

void DFS(Graph\* G) {

Stack L;

int mark[max\_vertices];

int parent[max\_vertices];

make\_null\_stack(&L);

int j;

for (j = 1; j <= G->n; j++){

mark[j] = 0;

parent[j]=-1;

}

int t;

for (t=1; t<=G->n; t++){

if(mark[t]==0) depth\_first\_search(G, mark, parent, &L, t);

}

for (t=1; t<=G->n; t++){

printf ("%d %d\n", t, parent[t]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

DFS(&G);

}



—----------------------

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

Nếu đồ thị không liên thông, 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 nói rằng 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>

Quy ước: Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 0.

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.
* Thêm 1 tham số p (đỉnh cha của đỉnh u) cho hàm visit(). Khi gọi đệ quy duyệt v ta truyền u như là đỉnh cha của v.

void visit(Graph\* G, int u, int p) {

...

for (các đỉnh kề v của u)

visit(G, v, u);

}

* 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 3  2 4 | 1 0  2 1  3 1  4 2 |
| 4 2  1 2  3 4 | 1 0  2 1  3 0  4 3 |
| 4 2  1 4  2 3 | 1 0  2 0  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 0  2 1  3 7  4 2  5 6  6 4  7 6  8 5  9 8  10 12  11 10  12 1  13 6 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

int parent[max\_vertices];

void traversal(Graph\* G, int x) {

int y;

if (mark[x] == 1) return;

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

if (mark[y] == 0) parent[y]=x; //parent chua xac dinh va luon thay doi den khi duoc danh dau

traversal(G, y);

//if (mark[y] != 0 ) parent[y]=x; // y da danh dau thi gan cha

//if (mark[y] == 0) parent[y]=x; //parent chua xac dinh va luon thay doi den khi duoc danh dau

//if (parent[y] == -1) parent[y]=x; vi kqua se gan vao luc cuoi nen lenh nay sai

}

}

}

void DFS(Graph\* G) {

int j;

for (j = 1; j <= G->n; j++){

mark[j] = 0;

parent[j]=-1;

}

int t;

for (t=1; t<=G->n; t++){

if(mark[t]==0) {

traversal(G, t);

parent[t]=0;

}

}

for (t=1; t<=G->n; t++){

printf ("%d %d\n", t, parent[t]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

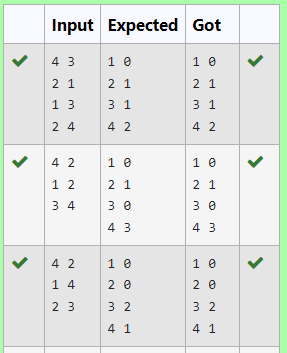
scanf("%d%d", &u, &v);

add\_edge(&G, u, v);

}

DFS(&G);

}



## Bài 3 - Đồ thị liên thông - Qua đảo

Có n hòn đảo và m cây cầu. Mỗi cây cầu bắt qua 2 hòn đảo. Một hôm chúa đảo tự hỏi là với các cây cầu hiện tại thì đứng ở một hòn đảo bất kỳ có thể nào đi đến được tất cả các hòn đảo khác hay không.

Hãy giúp chúa đảo viết chương trình kiểm tra.

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 đảo và số cây cầu.

- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng có 1 cây cầu bắt qua hai hòn đảo u và v.

Đầu ra (Output):

Nếu có thể đi được in ra màn hình YES, ngược lại in ra NO.

Xem thêm ví dụ bên dưới.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 3  1 2  2 3  3 4 | YES |
| 4 2  3 4  1 2 | NO |

**Answer:**(penalty regime: 0, 0, ... %)

### —--------------Sâu Ngăn xếp

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

void depth\_first\_search(Graph\* G, Stack\* L, int n) {

int y;

push(L, n);

while (!empty(L)) {

int x = top(L); pop(L);

if (mark[x] != 0) continue;

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

push(L, y);

}

}

}

}

//Ktra tinh lien thong

int KtraLT\_DFS(Graph\* G) {

Stack L;

make\_null\_stack(&L);

int j;

for (j = 1; j <= G->n; j++) mark[j] = 0;

depth\_first\_search(G, &L, 1);

for (j = 1; j <= G->n; j++){

if (mark[j] == 0) return 0;

}

return 1;

}

int main(){

//nhap--------

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

//-----------

if(KtraLT\_DFS(&G)==1) printf("YES");

else printf("NO");

return 0;

}

### —--------------Sâu Đệ quy

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

void traversal(Graph\* G, int x) {//Khong doi

int y;

if (mark[x] == 1) return;

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

traversal(G, y);

}

}

}

int DFS(Graph\* G) {

int t;

for (t = 1; t <= G->n; t++) mark[t] = 0;

traversal(G, 1);

for (t=1; t<=G->n; t++){

if(mark[t]==0) return 0;

}

return 1;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

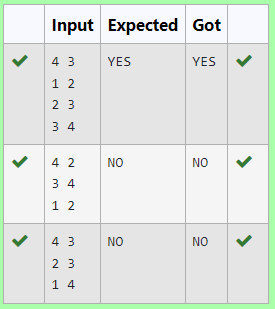
//-----------

if(DFS(&G)==1) printf("YES");

else printf("NO");

return 0;

}



## Bài 4 - Tôn Ngộ Không

Tôn Ngộ Không là một trong các nhân vật chính của truyện Tây du ký. Khi còn ở Hoa Quả Sơn, Tôn Ngộ Không là vua của loài khỉ. Hoa Quả Sơn có rất nhiều cây ăn trái, nên loài khỉ rất thích. Do đặc tính của mình, khỉ không thích đi bộ mà chỉ thích chuyền từ cây này sang một cây khác. Tuy nhiên, nếu khoảng cách giữa hai cây quá xa thi chúng không thể chuyền qua lại được.

Đường đường là vua của loài khỉ, Tôn Ngộ Không muốn vạch ra một kế hoạch hái trái cây trên tất cả các cây có trên Hoa Quả Sơn mà không cần phải nhảy xuống đất. Tôn Ngộ Không dự định sẽ bắt đầu leo lên một cây, hái trái của cây này, sau đó chuyền qua một cây kế tiếp hái trái của này và tiếp tục như thế cho đến khi tất cả các cây đều được hái trái. Một cây có thể được chuyền qua chuyền lại nhiều lần.

Hãy giúp Tôn Ngộ Không kiểm tra xem kế hoạch này có thể thực hiện được không.

Đầu vào (Input):

Giả sử số lượng cây ăn trái ở Hoa Quả Sơn là n cây và được đánh số từ 1 đến n.

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 cây và số cặp cây có thể chuyền qua lại.

- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng có thể chuyền từ cây u sang cây v hoặc chuyền từ cây v sang cây u.

Đầu ra (Output):

Nếu kế hoạch của Tôn Ngộ Không có thể thực hiện được DUOC, ngược lại in ra KHONG.

Xem thêm ví dụ bên dưới. Trong ví dụ đầu tiên, Tôn Ngộ Không bắt đầu từ cây 1, chuyền qua cây 2, sau đó chuyền ngược về 1, chuyền tiếp sang 3 và sau cùng là sang 4.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 3  2 1  1 3  3 4 | DUOC |
| 4 2  1 2  3 4 | KHONG |

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

### —--------------Sâu Ngăn xếp

//Cho phep qua nhieu diem

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

void depth\_first\_search(Graph\* G, Stack\* L, int n) {

int y;

push(L, n);

while (!empty(L)) {

int x = top(L); pop(L);

if (mark[x] != 0) continue;

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

push(L, y);

}

}

}

}

//Ktra tinh lien thong

int KtraLT\_DFS(Graph\* G) {

Stack L;

make\_null\_stack(&L);

int j;

for (j = 1; j <= G->n; j++) mark[j] = 0;

depth\_first\_search(G, &L, 1);

for (j = 1; j <= G->n; j++){

if (mark[j] == 0) return 0;

}

return 1;

}

int main(){

//nhap--------

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

//-----------

if(KtraLT\_DFS(&G)==1) printf("DUOC");

else printf("KHONG");

return 0;

}

### —--------------Sâu Đệ quy

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

int mark[max\_vertices];

void traversal(Graph\* G, int x) {//Khong doi

int y;

if (mark[x] == 1) return;

//printf("%d\n", x);

mark[x] = 1;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

traversal(G, y);

}

}

}

int DFS(Graph\* G) {

int t;

for (t = 1; t <= G->n; t++) mark[t] = 0;

traversal(G, 1);

for (t=1; t<=G->n; t++){

if(mark[t]==0) return 0;

}

return 1;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

//-----------

if(DFS(&G)==1) printf("DUOC");

else printf("KHONG");

return 0;

}



## Bài 5 - Kiểm tra chu trình - đồ thị vô hướng

Cho G=<V, E> là một đồ thị vô hướng đơn (không có khuyên, không có đa cung). Hãy viết chương trình kiểm tra xem có chứa chu trình hay không.

Chu trình là một đường đi đơn cung có đỉnh đầu trùng với đỉnh cuối.

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 YES nếu đồ thị có chứa chu trình, ngược lại in ra NO

Xem thêm ví dụ bên dưới.

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** |
| --- | --- |
| 3 2  1 3  2 3 | NO |
| 7 9  1 2  1 3  1 4  2 3  2 6  3 5  3 7  4 5  5 7 | YES |
| 7 9  1 2  1 3  2 4  2 5  2 6  3 5  3 6  4 7  5 7 | YES |
| 3 2  1 2  2 3 | NO |
| 4 3  1 2  2 3  2 4 | NO |

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

### —--------------Sâu Ngăn xếp

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

//Ktra chu trinh

#define white 0

#define gray 1

#define black 2

int color[max\_vertices]; //bo mark that bang color

int has\_circle, dem;

void depth\_first\_search(Graph\* G, int u) {

color[u] = gray;

for (int v = 1; v <= G->n; v++){

if (G->A[u][v]==1) {

if (color[v] == white){

//dem++;

depth\_first\_search(G, v);

dem--;

}

else if (color[v] == gray){

//printf("-%d", dem);

if (dem>1){

has\_circle = 1;

break;

}

dem++;

}

}

}

color[u] = black;

}

void KtraCT\_DFS(Graph\* G) {

int j;

for (j = 1; j <= G->n; j++){

color[j] = white;

}

has\_circle = 0;

for (j = 1; j <= G->n; j++){

if (color[j] == white){

dem =0;

depth\_first\_search(G, j);

}

if (has\_circle==1) break;

}

if (has\_circle==1) printf("YES");

else printf("NO");

}

int main(){

//nhap--------

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

KtraCT\_DFS(&G);

return 0;

}

### —--------------Sâu Đệ quy

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

//Ktra chu trinh

#define white 0

#define gray 1

#define black 2

int color[max\_vertices]; //bo mark that bang color

int parent[max\_vertices];

int has\_circle;

void traversal(Graph\* G, int x) {

int y;

if (color[x] == black) return;

//printf("%d\n", x);

color[x] = gray;

for (y = 1; y <= G->n; y++) {

if (y==parent[x]) continue;//printf("<<%d-%d>>", y, parent[y]);

else if (G->A[x][y]==1) {

if (color[y] == white){

parent[y]=x;

traversal(G, y);

}

else if (color[y] == gray){

//printf("<%d-%d>", y, parent[y]);

has\_circle = 1;

break;

}

}

}

color[x] = black;

}

void DFS(Graph\* G) {

int t;

for (t = 1; t <= G->n; t++) color[t] = white;

has\_circle = 0;

for (t=1; t<=G->n; t++){

if (color[t] == white){

parent[t]=0;

traversal(G, t);

}

if (has\_circle==1) break;

}

if (has\_circle==1) printf("YES");

else printf("NO");

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

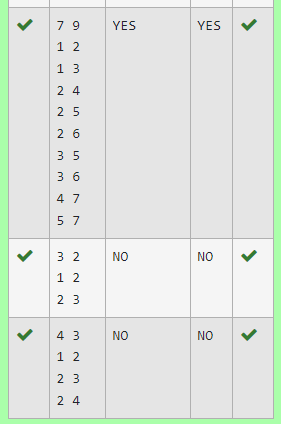
DFS(&G);

//int t;

//for (t = 1; t <= G.n; t++) printf("<%d-%d>", t, parent[t]);

return 0;

}



## Bài 6 - Thuyền trưởng Haddock

Thuyền trưởng Haddock (truyện Tintin) là một người luôn say xỉn. Vì thế đôi khi Tintin không biết ông ta đang say hay tỉnh. Một ngày nọ, Tintin hỏi ông ta về cách uống. Haddock nói như thế này: Có nhiều loại thức uống (soda, wine, water, …), tuy nhiên Haddock lại tuân theo quy tắc “*để uống một loại thức uống nào đó cần phải uống tất cả các loại thức uống tiên quyết của nó*”. Ví dụ: để uống rượu (wine), Haddock cần phải uống soda và nước (water) trước. Vì thế muốn say cũng không phải dễ !

Cho danh sách các thức uống và các thức uống tiên quyết của nó. Hãy xét xem Haddock có thể nào say không ? Để làm cho Haddock say, ông ta phải uống hết tất cả các thức uống.

Ví dụ 1:

soda wine

water wine

Thức uống tiên quyết được cho dưới dạng a b, có nghĩa là để uống b bạn phải uống a trước. Trong ví dụ trên để uống wine, Haddock phải uống soda và water trước. Soda và water không có thức uống tiên quyết nên Haddock sẽ SAY.

Ví dụ 2:

soda wine

water wine

wine water

Để uống wine, cần uống water. Tuy nhiên để uống water lại cần wine. Vì thế Haddock không thể uống hết được các thức uống nên ông ta KHÔNG SAY.

Để đơn giản ta có thể giả sử các thức uống được mã hoá thành các số nguyên từ 1, 2, … và dữ liệu đầu vào được cho có dạng như sau (ví dụ 1):

3 2

1 2

3 2

Có loại thức uống (soda: 1, wine: 2 và water: 3) và có 2 điều kiện tiên quyết

1 -> 2 và 3 -> 2.

Với ví dụ 2, ta có dữ liệu:

3 3

1 2

3 2

2 3

Viết chương trình đọc dữ liệu các thức uống và kiểm tra xem Haddock có thể say không. Nếu có in ra “YES”, ngược lại in ra “NO”.

Đầu vào (Input):

Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:

* Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số thức uống và số điều kiện tiên quyết .
* m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng thức uống u là tiên quyết của thức uống v.

Đầu ra (Output):

* Nếu Haddock có thể say in ra YES, ngược lại in ra NO.

Xem thêm các ví dụ bên dưới.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 2  1 2  3 2 | YES |
| 3 3  1 2  3 2  2 3 | NO |
| 5 5  1 2  2 3  3 4  4 2  5 4 | NO |

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

### —--------------Sâu Ngăn xếp

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

//-------------------------

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]+=1;

//G->A[y][x]=1;

}

//------------------------

//Ktra chu trinh

#define white 0

#define gray 1

#define black 2

int color[max\_vertices]; //bo mark that bang color

int has\_circle;

//, dem

void depth\_first\_search(Graph\* G, int u) {

color[u] = gray;

for (int v = 1; v <= G->n; v++){

if (G->A[u][v]!=0) {

if (color[v] == white){

depth\_first\_search(G, v);

//dem--;

}

else if (color[v] == gray){

//printf("-%d", dem);

//if (dem>1){

has\_circle = 1;

break;

//}

//dem++;

}

}

}

color[u] = black;

}

void KtraCT\_DFS(Graph\* G) {

int j;

for (j = 1; j <= G->n; j++){

color[j] = white;

}

has\_circle = 0;

for (j = 1; j <= G->n; j++){

if (color[j] == white){

//dem =0;

depth\_first\_search(G, j);

}

if (has\_circle==1) break;

}

if (has\_circle==1) printf("NO"); //Chu trinh nen ko say

else printf("YES");

//Neu co 1 vong lap uong lien tuc se ko say

//Vi day la do thi co huong nen khong so tao chu trinh 2 nut(?)

}

int main(){

//nhap--------

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

KtraCT\_DFS(&G);

return 0;

}

### —--------------Sâu Đệ quy

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1;

}

//------------------------

//Ktra chu trinh

#define white 0

#define gray 1

#define black 2

int color[max\_vertices]; //bo mark that bang color

//int parent[max\_vertices];

int has\_circle;

void traversal(Graph\* G, int x) {

int y;

if (color[x] == black) return;

//printf("%d\n", x);

color[x] = gray;

for (y = 1; y <= G->n; y++) {

//if (y==parent[x]) continue;//printf("<<%d-%d>>", y, parent[y]);

if (G->A[x][y]==1) {

if (color[y] == white){

//parent[y]=x;

traversal(G, y);

}

else if (color[y] == gray){

//printf("<%d-%d>", y, parent[y]);

has\_circle = 1;

break;

}

}

}

color[x] = black;

}

void DFS(Graph\* G) {

int t;

for (t = 1; t <= G->n; t++) color[t] = white;

has\_circle = 0;

for (t=1; t<=G->n; t++){

if (color[t] == white){

//parent[t]=0;

traversal(G, t);

}

if (has\_circle==1) break;

}

if (has\_circle==1) printf("NO"); //Chu trinh nen ko say

else printf("YES");

//Neu co 1 vong lap uong lien tuc se ko say

//Vi day la do thi co huong nen khong so tao chu trinh 2 nut(?)

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

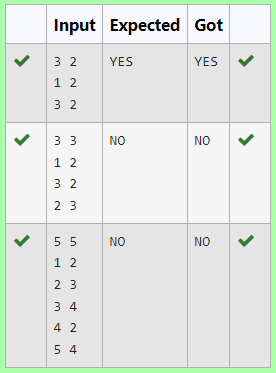
DFS(&G);

//int t;

//for (t = 1; t <= G.n; t++) printf("<%d-%d>", t, parent[t]);

return 0;

}



## Bài 7- Phân chia đội bóng

David là huấn luyện viên của một đội bóng gồm N thành viên. David muốn chia đội bóng thành hai nhóm. Để tăng tính đa dạng của các thành viên trong nhóm, David quyết định không xếp hai thành viên đã từng thi đấu với nhau vào chung một nhóm. Bạn hãy lập trình giúp David phân chia đội bóng.

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 thành viên và số cặp thành viên đã từng thi đấu với nhau.
* M dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng 2 thành viên u và v đã từng thi đấu chung với nhau.

Đầu ra (Output):

* Nếu phân chia được, hãy in ra các thành viên của mỗi nhóm. Nhóm của thành viên 1 sẽ được in trước, nhóm còn lại in ra sau. Các thành viên trong nhóm được in ra theo thứ tự tăng dần và in trên 1 dòng. Hai thành viên cách nhau 1 khoảng trắng.
* Nếu không thể phân chia, in ra IMPOSSIBLE

Xem thêm các ví dụ bên dưới.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 2  1 2  2 3 | 1 3  2 |
| 3 3  1 2  2 3  3 1 | IMPOSSIBLE |
| 9 8  1 2  1 3  1 4  1 5  1 6  1 7  1 8  1 9 | 1  2 3 4 5 6 7 8 9 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

G->A[y][x]=1;

}

//------------------------

//Do thi phan doi

int color[max\_vertices]; //bo mark that bang color

int fail;

//Khong can dfs

void colorize(Graph\* G, int x, int c) {

if (color[x] == -1){

color[x] = c;

int y;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) colorize(G, y, !c);

}

}

else if (color[x] != c) fail = 1;

}

int is\_bigraph(Graph\* G) {

int j;

for (j = 1; j <= G->n; j++) color[j] = -1;

fail = 0;

colorize(G, 1, 0); /\*To mau dinh 1 \*/

return !fail;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

if (!is\_bigraph(&G)) printf("IMPOSSIBLE");

else{

for (e = 1; e <= G.n; e++){

if (color[e]==0) printf("%d ", e);

}

printf("\n");

for (e = 1; e <= G.n; e++){

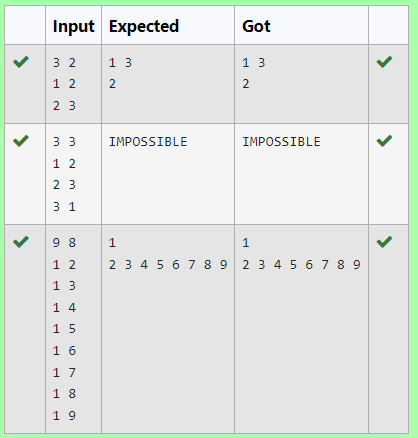
if (color[e]!=0) printf("%d ", e);

}

}

return 0;

}



## Bài 8 - Kiểm tra tính liên thông mạnh - đếm số BPLT mạnh

Viết chương trình kiểm tra xem một đồ thị có hướng có liên thông mạnh không.

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).

Chú ý: đồ thị không chứa đa cung.

Đầu ra (Output):

In ra màn hình strong connected nếu đồ thị đã cho liên thông mạnh, ngược lại in ra unconnected.

Xem thêm ví dụ bên dưới.

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** |
| --- | --- |
| 5 7  1 2  2 3  3 1  2 4  3 4  4 5  5 3 | strong connected |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1;

}

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//------------------------

int num[max\_vertices];

int min\_num[max\_vertices];

int on\_stack[max\_vertices];

int k, dinhkhop;

int min(int a, int b){

return a<b? a: b;

}

void strong\_connect(Graph\* G, Stack\* S, int x) {

num[x] = k;

min\_num[x] = k;

push(S, x);

on\_stack[x] = 1;

k++;

int y;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

if (num[y] < 0) {

strong\_connect(G, S, y);

min\_num[x] = min(min\_num[x], min\_num[y]);

}

else if (on\_stack[y])

min\_num[x] = min(min\_num[x], num[y]);

}

}

//printf("min\_num[%d] = %d\n", x, min\_num[x]);

//Di tim tat ca dinh khop

if (num[x] == min\_num[x]) {

dinhkhop++;

//printf("%d.%d la dinh khop.\n", dinhkhop, x);

int w;

do {

w = top(S);

pop(S);

on\_stack[w] = 0;

} while (w != x);

}

}

int str\_connect (Graph\* G){

Stack S;

make\_null\_stack(&S);

int j;

for (j = 1; j <= G->n; j++){

num[j] = -1;

min\_num[j] = -1;

on\_stack[j] = 0;

}

k=1, dinhkhop=0;

strong\_connect (G, &S, 1); //duyet tu dinh 1

if (dinhkhop ==1) return 1;

else return 0;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

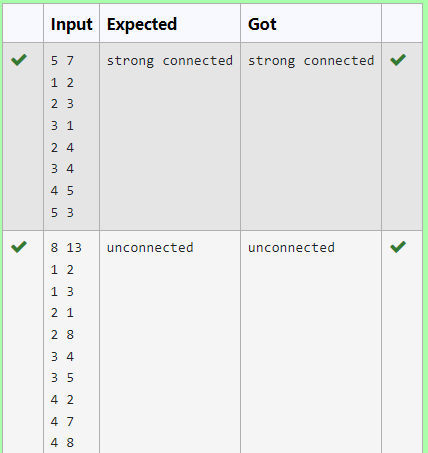
}

//---------------

if (str\_connect(&G)==1) printf("strong connected");

else printf("unconnected");

}



Viết chương trình đếm số bộ phận liên thông mạnh (BPLTM) của một đồ thị có hướng.

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).

Chú ý: đồ thị không chứa đa cung.

Đầu ra (Output):

In ra màn hình một con số nguyên duy nhất chỉ số BPTLM của đồ thị.

Xem thêm ví dụ bên dưới.

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** |
| --- | --- |
| 5 7  1 2  2 3  3 1  2 4  3 4  4 5  5 3 | 1 |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1;

}

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//------------------------

int num[max\_vertices];

int min\_num[max\_vertices];

int on\_stack[max\_vertices];

int k, dinhkhop;

int min(int a, int b){

return a<b? a: b;

}

void strong\_connect(Graph\* G, Stack\* S, int x) {

num[x] = k;

min\_num[x] = k;

push(S, x);

on\_stack[x] = 1;

k++;

int y;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

if (num[y] < 0) {

strong\_connect(G, S, y);

min\_num[x] = min(min\_num[x], min\_num[y]);

}

else if (on\_stack[y])

min\_num[x] = min(min\_num[x], num[y]);

}

}

//printf("min\_num[%d] = %d\n", x, min\_num[x]);

//Di tim tat ca dinh khop

if (num[x] == min\_num[x]) {

dinhkhop++;

//printf("%d.%d la dinh khop.\n", dinhkhop, x);

int w;

do {

w = top(S);

pop(S);

on\_stack[w] = 0;

} while (w != x);

}

}

void str\_connect (Graph\* G){

Stack S;

make\_null\_stack(&S);

int j;

for (j = 1; j <= G->n; j++){

num[j] = -1;

min\_num[j] = -1;

on\_stack[j] = 0;

}

k=1, dinhkhop=0;

strong\_connect (G, &S, 1); //duyet tu dinh 1

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

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

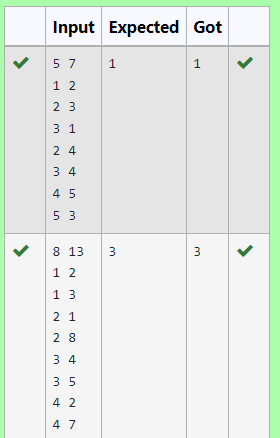
}

//---------------

str\_connect(&G);

return 0;

}



## Bài 9 - Come and Go

Trong một thành phố có N địa điểm được nối với nhau bằng M con đường 1 chiều và 2 chiều. Yêu cầu tối thiểu của một thành phố là từ địa điểm này bạn phải có thể đi đến một địa điểm khác bất kỳ.

Hãy viết chương trình kiểm tra xem các con đường của thành phố có thoả mãn yêu cầu tối thiểu này không. Nếu có in ra OKIE, ngược lại in ra NO.

(nguồn: UVA Online Judge, Problem 11838)

Dữ liệu đầu vào có dạng như sau:

4 5

1 2 1

1 3 2

2 4 1

3 4 1

4 1 2

Trong ví dụ này, có 4 địa điểm và 5 con đường, mỗi con đường có dạng a b p, trong đó a, b là các địa điểm; và nếu p = 1, con đường đang xét là đường 1 chiều, ngược lại nó là đường 2 chiều.

Đầu vào (Input):

Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:

- Dòng đầu tiên chứa 2 số nguyên N và M, tương ứng là số địa điểm và số con đường.

- M dòng tiếp theo mỗi dòng chứa 3 số nguyên a, b, p. Nếu p = 1, con đường (a, b) là con đường 1 chiều, ngược lại nếu p = 2, con đường (a, b) là con đường 2 chiều.

Đầu ra (Output):

In ra màn hình OKIE nếu các con đường của thành phố có thoả mãn yêu cầu, ngược lại in ra NO.

Xem thêm ví dụ bên dưới.

Gợi ý:

* Xây dựng đồ thị có hướng từ dữ liệu các con đường và các địa điểm
  + Địa điểm ~ đỉnh
  + Đường 1 chiều ~ cung
  + Đường 2 chiều ~ 2 cung
* Áp dụng giải thuật kiểm tra đồ thị có liên thông mạnh hay không.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5 7  1 2 1  2 3 1  3 1 1  2 4 1  3 4 1  4 5 1  5 3 1 | OKIE |
| 8 10  1 2 2  1 3 1  2 8 1  3 4 1  3 5 2  4 2 1  4 7 1  4 8 1  5 7 1  6 7 2 | NO |
| 3 2  1 2 2  1 3 2 | OKIE |

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

//n dinh, m cung

//ma tran dinh-dinh

//Chieurong

#include<stdio.h>

#define max\_vertices 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int n;

int A[max\_vertices][max\_vertices];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y, int p){

G->A[x][y]=1;

if (p==2) G->A[y][x]=1;

}

/\* Khai bao Stack\*/

typedef struct {

int data[max\_elements];

int size;

} Stack;

void make\_null\_stack(Stack\* S) {

S->size = 0;

}

void push(Stack\* S, int x) {

S->data[S->size] = x;

S->size++;

}

int top(Stack\* S) {

return S->data[S->size - 1];

}

void pop(Stack\* S) {

S->size--;

}

int empty(Stack\* S) {

return S->size == 0;

}

//------------------------

int num[max\_vertices];

int min\_num[max\_vertices];

int on\_stack[max\_vertices];

int k, dinhkhop;

int min(int a, int b){

return a<b? a: b;

}

void strong\_connect(Graph\* G, Stack\* S, int x) {

num[x] = k;

min\_num[x] = k;

push(S, x);

on\_stack[x] = 1;

k++;

int y;

for (y = 1; y <= G->n; y++) {

if (G->A[x][y]==1) {

if (num[y] < 0) {

strong\_connect(G, S, y);

min\_num[x] = min(min\_num[x], min\_num[y]);

}

else if (on\_stack[y])

min\_num[x] = min(min\_num[x], num[y]);

}

}

//printf("min\_num[%d] = %d\n", x, min\_num[x]);

//Di tim tat ca dinh khop

if (num[x] == min\_num[x]) {

dinhkhop++;

//printf("%d.%d la dinh khop.\n", dinhkhop, x);

int w;

do {

w = top(S);

pop(S);

on\_stack[w] = 0;

} while (w != x);

}

}

int str\_connect (Graph\* G){

Stack S;

make\_null\_stack(&S);

int j;

for (j = 1; j <= G->n; j++){

num[j] = -1;

min\_num[j] = -1;

on\_stack[j] = 0;

}

k=1, dinhkhop=0;

for (j = 1; j <= G->n; j++){

if (num[j]==-1) strong\_connect (G, &S, j); //duyet dinh

}

/\*strong\_connect (G, &S, 1);\*/

if (dinhkhop ==1) return 1;

else return 0;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, p, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &p);

add\_edge(&G, u, v, p);

}

//---------------

if (str\_connect(&G)==1) printf("OKIE");

else printf("NO");

}

# [Thực hành buổi 3](https://else.ctu.edu.vn/course/view.php?id=54#section-7)

## Bài 1 - Tìm đường đi ngắn nhất

Cho đồ thị có hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến n.

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.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n. Nếu không có đường đi từ 1 đến n, in ra -1.

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | 4 |
| 3 1  1 2 5 | -1 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | 20 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

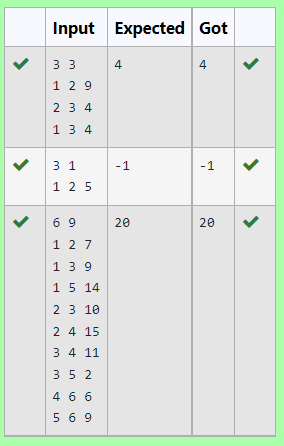
}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



—-----------------------

Cho đồ thị vô hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến n.

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.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n. Nếu không có đường đi từ 1 đến n, in ra -1.

Xem thêm ví dụ bên dưới.

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, &w);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | 4 |
| 3 1  1 2 5 | -1 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | 20 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

G->L[v][u] = w;

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

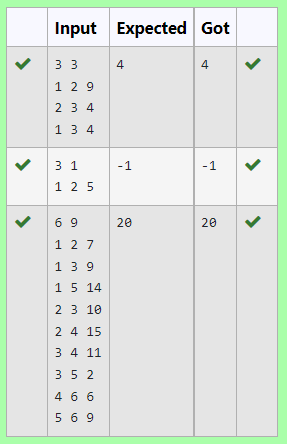
}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



## Bài 2 - Tìm đường đi ngắn nhất (Check được): Giống hệt BTH1

Cho đồ thị có hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến n.

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.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n. Nếu không có đường đi từ 1 đến n, in ra -1.

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | 4 |
| 3 1  1 2 5 | -1 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | 20 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



—----------------------

Cho đồ thị vô hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến n.

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.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n. Nếu không có đường đi từ 1 đến n, in ra -1.

Xem thêm ví dụ bên dưới.

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, &w);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | 4 |
| 3 1  1 2 5 | -1 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | 20 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

G->L[v][u] = w;

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



## Bài 3 - Kiểm tra chu trình âm và ứng dụng đường đi ngắn nhất

Viết chương trình kiểm tra một đồ thị có hướng (không có khuyên, không có đa cung) xem có chứa chu trình âm hay không.

Chu trình âm là chu trình có tổng trọng số các cung trong chu trình nhỏ hơn 0.

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, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình negative cycle nếu đồ thị có chứa chu trình âm, ngược lại in ra ok

Xem thêm ví dụ bên dưới.

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, &w);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 2  1 3 1  3 1 1 | ok |
| 7 10  1 2 1  1 3 1  1 4 1  2 3 1  2 6 1  3 7 1  4 5 1  5 3 1  5 7 1  3 1 1 | ok |
| 7 12  1 2 1  1 3 1  2 4 1  2 5 1  2 6 1  3 2 1  3 5 1  3 6 1  4 7 1  5 7 1  6 4 1  2 1 1 | ok |
| 3 3  1 2 4  2 3 -10  3 1 2 | negative cycle |
| 3 3  1 2 1  2 3 1  3 1 1 | ok |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

/\*for (i = 1; i <= G->n; i++)

printf("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);\*/

if (pi[G->n] <0) printf("negative cycle");

else printf("ok");

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

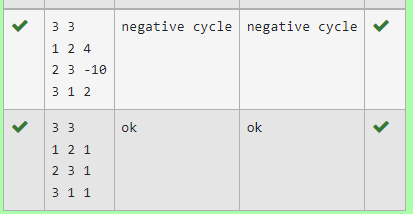
}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



—---------------------

"*Ngưu Lang là vị thần chăn trâu của Ngọc Hoàng Thượng đế, vì say mê một tiên nữ phụ trách việc dệt vải tên là Chức Nữ nên bỏ bễ việc chăn trâu, để trâu đi nghênh ngang vào điện Ngọc Hư. Chức Nữ cũng vì mê tiếng tiêu của Ngưu Lang nên trễ nải việc dệt vải. Ngọc Hoàng giận dữ, bắt cả hai phải ở cách xa nhau, người đầu sông Ngân, kẻ cuối sông*.

*Mỗi năm một lần, Sau đó, Ngọc Hoàng thương tình nên ra ơn cho hai người mỗi năm được gặp nhau vào ngày 7 tháng Bảy âm lịch. Khi tiễn biệt nhau, Ngưu Lang và Chức Nữ khóc sướt mướt. Nước mắt của họ rơi xuống trần hóa thành cơn mưa và được người dưới trần gian đặt tên là mưa ngâu.*" (Theo wikipedia.com)

Để gặp được nhau vào ngày 7/7, Ngưu Lang và Chức Nữ phải nhờ đến bầy quạ đen bắt cầu (gọi là Ô kiều) cho mình đi qua để gặp nhau.

Sông Ngân Hà có n ngôi sao, giả sử được đánh số từ 1 đến n. Ngưu Lang ở tại ngôi sao Ngưu (Altair), được đánh số 1, còn Chức Nữ ở tại ngôi sao Chức Nữ (Vega) được đánh số n. Để bắt được một cây cầu từ ngôi sao này sang ngôi sao kia cần một số lượng quạ nào đó. Một khi con quạ ở cây cầu nào thì phải ở đó cho đến khi Ngưu Lang và Chức Nữ gặp được nhau.

Quạ thì càng ngày càng hiếm, nên Ngưu Lang và Chức Nữ phải tính toán sao cho số lượng quạ ít nhất có thể.

Hãy giúp Ngưu Lang và Chức Nữ viết chương trình tính xem cần phải nhờ đến ít nhất bao nhiêu con quạ để bắt cầu cho họ gặp nhau.

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 ngôi sao và số cặp sao có thể bắt cầu.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u v q nói rằng để bắt 1 cây cầu bắt qua hai ngôi sao u và v cần phải tốn q con quạ.

Đầu ra (Output):

In ra màn hình số lượng quạ cần thiết.

Xem thêm ví dụ bên dưới.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 3  1 2 5  2 3 2  3 4 1 | 8 |
| 4 3  1 2 3  2 4 4  1 4 10 | 7 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

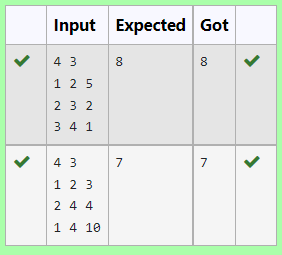
}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



## Bài 4 - Ứng dụng đường đi ngắn nhất

Đất nước CyberGraph có n thành phố và m con đường. Mỗi con đường nối 2 thành phố lại với nhau. tất cả các con đường đều là đường 2 chiều, mỗi con đường có một chiều dài nào đó. giữa hai thành phố có nhiều nhất là 1 con đường.

Tổng thổng của nước này dự định sẽ đi từ thành phố s đến thành phố t. Đương nhiên, ông ta sẽ chọn hành trình có tổng chiều dài các con đường đi qua ngắn nhất.

Hãy giúp Ngài tổng thổng tìm hành trình ngắn nhất đi từ s đến t.

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 thành phố và số con đường.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u v d mô tả con đường nối hai thành phố u và v có chiều dài d.

- Dòng cuối cùng chứa số nguyên s t là thành phố bắt đầu và kết thúc.

Đầu ra (Output):

In ra màn hình tổng chiều dài các con đường của hành trình ngắn nhất..

Xem thêm ví dụ bên dưới.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 3  1 2 5  2 3 2  3 4 5  1 4 | 12 |
| 4 3  1 2 3  2 4 4  1 4 10  1 4 | 7 |
| 4 4  1 2 2  2 4 4  1 3 6  1 4 10  2 3 | 8 |

**Answer:**(penalty regime: 0, 0, ... %)

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

G->L[v][u] = w;

}

void Dijkstra(Graph\* G, int s, int t) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[t] == INFINITY) printf("-1");

else printf("%d",pi[t]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

}

int s, t;

scanf("%d%d", &s, &t);

//---Het nhap---

Dijkstra(&G, s, t);

return 0;

}



## Bài 5 - Mê cung số (Number Maze)

Mê cung số (number maze)

Cho một mê cung số được biểu diễn bằng một mảng 2 chiều chứa các con số từ 0 đến 9 (xem

hình bên dưới).



Một con robot được đặt tại góc trên bên trái của mê cung và muốn đi đến góc dưới bên phải của mê cung. Con robot có thể đi lên, xuống, qua trái và qua phải 1 ô. Chi phí để đi đến một ô bằng với con số bên trong ô đó.

Hãy tìm cách giúp con robot đi đến ô góc dưới phải sao cho tổng chi phí thấp nhất.

Đường đi có chi phí thấp nhất cho ví dụ này là 24.

Đầu vào (Input):

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

* Dòng dầu chứa 2 số nguyên M N (M: số hàng, N: số cột)
* M dòng tiếp theo mô tả các số trong mê cung

Đầu ra (Output):

In ra màn hình chi phí thấp nhất để con robot đi từ góc trên bên trái về góc dưới bên phải. Ví dụ trên, cần in ra màn hình: 24.

Xem thêm các ví dụ bên dưới.

Gợi ý:

Mô hình hoá bài toán về đồ thị có hướng

* Đỉnh ~ ô
* Cung ~ hai ô cạnh nhau
* Trọng số cung (u, v) = giá trị của ô tương ứng với đỉnh v.

Xem tài liệu thực hành để biết cách đặt tên cho các ô.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 5  0 3 1 2 9  7 3 4 9 9  1 7 5 5 3  2 3 4 2 5 | 24 |
| 3 3  1 2 3  2 4 4  1 4 10 | 18 |
| 4 3  1 2 2  2 1 4  4 2 1  8 4 10 | 17 |

**Answer:**(penalty regime: 0, 0, ... %)

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s, int gtbandau) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = gtbandau;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int hang, cot, u, v;

scanf("%d%d", &hang, &cot);

int n = ((hang-1)\*cot + (cot-1)) + 1; //n dinh, moi dinh add max 4 canh +1, -1, +cot, -cot

init\_graph(&G, n);

int A[hang][cot];

for (u = 0; u < hang; u++) { //hang

for (v = 0; v < cot; v++) { //cot

scanf ("%d", &A[u][v]);

}

}

for (u = 0; u < hang; u++) { //hang

for (v = 0; v < cot; v++) { //cot

//xet dinh (u,v)

int dinhxet=(u\*cot + v) + 1;

if (dinhxet+1>=1 && dinhxet+1<=n && dinhxet%cot!=0) add\_edge(&G, dinhxet, dinhxet+1, A[u][v+1]); //(u,v+1)

if (dinhxet-1>=1 && dinhxet-1<=n && (dinhxet-1)%cot!=0) add\_edge(&G, dinhxet, dinhxet-1, A[u][v-1]); //(u,v-1)

if (dinhxet+cot>=1 && dinhxet+cot<=n) add\_edge(&G, dinhxet, dinhxet+cot, A[u+1][v]); //(u+1,v)

if (dinhxet-cot>=1 && dinhxet-cot<=n) add\_edge(&G, dinhxet, dinhxet-cot, A[u-1][v]); //(u-1,v)

}

}

//---Het nhap---

//Dijkstra(&G, 1, A[0][0]);

Dijkstra(&G, 1, A[(1-1)/cot][(1-1)%cot]);

return 0;

}

—----------------Hoặc—-----------------------------

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

}

void Dijkstra(Graph\* G, int s, int gtbandau) {

int i, j, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = gtbandau;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

if (pi[G->n] == INFINITY) printf("-1");

else printf("%d",pi[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int hang, cot, u, v;

scanf("%d%d", &hang, &cot);

int n = ((hang-1)\*cot + (cot-1)) + 1; //n dinh, moi dinh add max 4 canh +1, -1, +cot, -cot

init\_graph(&G, n);

int A[hang][cot];

for (u = 0; u < hang; u++) { //hang

for (v = 0; v < cot; v++) { //cot

scanf ("%d", &A[u][v]);

}

}

int di[] = {-1, 1, 0, 0};

int dj[] = { 0, 0, -1, 1};

int i,j,k;

for (u = 1; u < n; u++) {

//Đổi đỉnh u thành ô (i, j)

i = (u - 1) /cot;

j = (u - 1) %cot;

//Duyệt qua các ô kề của ô (i, j)

for (k = 0; k < 4; k++) {

int ii = i + di[k];

int jj = j + dj[k];

//Kiểm tra ô (ii,jj) có nằm trong mê cung không

if (ii >= 0 && ii < hang && jj >= 0 && jj < cot) {

v = (ii\*cot + jj)+1; //đổi ô (ii,jj) thành đỉnh v

//v là đỉnh kề của đỉnh u

add\_edge(&G, u, v, A[ii][jj]);

//printf("%d %d\n", u, v);

}

}

}

//---Het nhap---

//Dijkstra(&G, 1, A[0][0]);

Dijkstra(&G, 1, A[(1-1)/cot][(1-1)%cot]);

return 0;

}



## Bài 6 - Bellman - Ford pi và pi

Cho đồ thị có hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (-100 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại.

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.
* m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Dữ liệu được đảm bảo không tồn tại chu trình âm.

Đầu ra (Output):

In ra màn hình các giá trị pi và p của các đỉnh theo thứ tự 1, 2, ..., n.

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | pi[1] = 0, p[1] = -1  pi[2] = 9, p[2] = 1  pi[3] = 4, p[3] = 1 |
| 8 13  1 2 4  1 3 4  3 5 4  3 6 2  4 1 3  4 3 2  5 4 1  5 7 5  6 2 3  6 5 -3  7 6 2  7 8 2  8 5 -2 | pi[1] = 0, p[1] = -1  pi[2] = 4, p[2] = 1  pi[3] = 4, p[3] = 1  pi[4] = 4, p[4] = 5  pi[5] = 3, p[5] = 6  pi[6] = 6, p[6] = 3  pi[7] = 8, p[7] = 5  pi[8] = 10, p[8] = 7 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | pi[1] = 0, p[1] = -1  pi[2] = 7, p[2] = 1  pi[3] = 9, p[3] = 1  pi[4] = 20, p[4] = 3  pi[5] = 11, p[5] = 3  pi[6] = 20, p[6] = 5 |

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

#include<stdio.h>

//Ma tran cung

//Chuong trinh chi tim duong ngan nhat, ko loại chu trinh am

#define INFINITY 9999999

#define MAXN 1000

int pi[MAXN];

int p[MAXN];

typedef struct {

int u, v; // đỉnh đầu v, đỉnh cuối v

int w; // trọng số w

} Edge;

typedef struct {

int n, m; // n: đỉnh, m: cung

Edge edges[1000]; // lưu các cung của đồ thị

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

G->m = 0;

}

void add\_edge(Graph\* G, int u, int v, int w) {

G->edges[G->m].u = u;

G->edges[G->m].v = v;

G->edges[G->m].w = w;

G->m++;

}

void BellmanFord(Graph\* G, int s) {

int i, k, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

//Duyet dinh, lặp n hoặc n-1 lần đều được

for (it = 1; it < G->n; it++) {

// Duyệt qua các cung và cập nhật (nếu thoả)

for (k = 0; k < G->m; k++) {

int u = G->edges[k].u;

int v = G->edges[k].v;

int w = G->edges[k].w;

if (pi[u] + w < pi[v]) {

pi[v] = pi[u] + w;

p[v] = u;

}

}

}

for (it = 1; it <= G->n; it++) {

printf("pi[%d] = %d, p[%d] = %d\n", it, pi[it], it, p[it]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

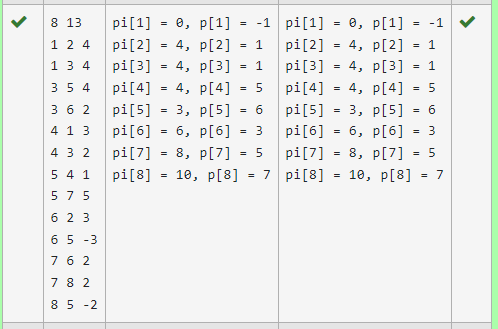
}

//---Het nhap---

BellmanFord(&G, 1);

return 0;

}



## Bài 7 - Bellman - Ford

Áp dụng giải thuật Bellman – Ford kiểm tra xem một đồ thị có hướng có chứa chu trình âm hay không, nếu ta tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại. In kết quả YES (nếu đồ thị có chu trình âm) hoặc NO (trường hợp ngược lại).

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 3 số nguyên u, v, w nói rằng cung (u, v) có trọng số w.

- Dòng cuối cùng chứa đỉnh s.

Đầu ra (Output):

In ra màn hình YES nếu phát hiện chu trình âm, ngược lại in ra NO.

Xem thêm ví dụ bên dưới.

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, &w);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 4  1 2 1  2 3 -1  3 4 -1  4 1 -1  2 | YES |
| 8 13  1 2 4  1 3 4  3 5 4  3 6 -2  4 1 3  4 3 2  5 4 1  5 7 -2  6 2 3  6 5 -3  7 6 2  7 8 2  8 5 -2  1 | YES |
| 8 13  1 2 4  1 3 4  3 5 4  3 6 2  4 1 3  4 3 2  5 4 1  5 7 5  6 2 3  6 5 -3  7 6 2  7 8 2  8 5 -2  1 | NO |

**Answer:**(penalty regime: 0, 0, ... %)

#include<stdio.h>

//Ma tran cung

//Chuong trinh chi tim duong ngan nhat, ko loại chu trinh am

#define INFINITY 9999999

#define MAXN 1000

int pi[MAXN];

int p[MAXN];

typedef struct {

int u, v; // đỉnh đầu v, đỉnh cuối v

int w; // trọng số w

} Edge;

typedef struct {

int n, m; // n: đỉnh, m: cung

Edge edges[1000]; // lưu các cung của đồ thị

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

G->m = 0;

}

void add\_edge(Graph\* G, int u, int v, int w) {

G->edges[G->m].u = u;

G->edges[G->m].v = v;

G->edges[G->m].w = w;

G->m++;

}

void BellmanFord(Graph\* G, int s) {

int i, k, it;

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

//Duyet dinh, lặp n hoặc n-1 lần đều được

for (it = 1; it < G->n; it++) {

// Duyệt qua các cung và cập nhật (nếu thoả)

for (k = 0; k < G->m; k++) {

int u = G->edges[k].u;

int v = G->edges[k].v;

int w = G->edges[k].w;

if (pi[u] + w < pi[v]) {

pi[v] = pi[u] + w;

p[v] = u;

}

}

}

int chutrinham=0;

// Duyệt qua các cung một lần nữa

for (k = 0; k < G->m; k++) {

int u = G->edges[k].u;

int v = G->edges[k].v;

int w = G->edges[k].w;

if (pi[u] + w < pi[v]) {

// Có chu trình âm

chutrinham=1;

}

}

if (chutrinham) printf("YES");

else printf("NO");

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

}

int s;

scanf("%d", &s);

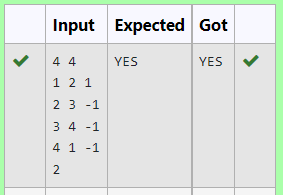
//---Het nhap---

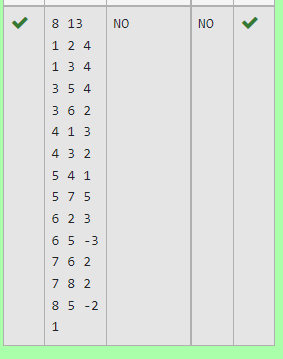
//printf("%d",s);

BellmanFord(&G, s);

return 0;

}





## Bài 9 - Tìm số đường đi ngắn nhất (nâng cao)

Cho đồ thị đơn, vô hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đếm xem có bao nhiêu đường đi ngắn nhất từ đỉnh 1 đến n.

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.

- m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n và số đường đi có chiều dài bằng với chiều dài dường đi ngắn nhất. Nếu không có đường đi từ 1 đến n, in ra -1 0.

Trong ví dụ đầu tiên có 2 đường đi từ 1 đến 3 là: 1 - 2 - 3 và 1 - 3. Cả hai đường đi này đều ngắn nhất và có chiều dài bằng 6.

Xem thêm ví dụ bên dưới.

Gợi ý:

Ngoài pi[v], ta định nghĩa thêm cnt[v]: số đường đi ngắn nhất từ s đến v.

Mỗi lần cập nhật pi[v] ta chú ý cập nhật lại cnt[v]. Nếu đường đi mới qua u tốt hơn thì cnt[v] = cnt[u]. Nếu bằng nhau thì cộng thêm: cnt[v] += cnt[u].

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, &w);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 2  2 3 4  1 3 6 | 6 2 |
| 3 1  1 2 5 | -1 0 |
| 6 9  1 2 7  1 3 9  1 5 14  2 3 10  2 4 15  3 4 11  3 5 2  4 6 6  5 6 9 | 20 1 |

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

#include<stdio.h>

//Ma tran dinh-dinh

#define MAXN 1000

#define NO\_EDGE 0

//hoac gia tri dac biet nao do

#define INFINITY 9999999

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

typedef struct {

int n;

int L[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->L[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->L[u][v] = w; //thay vi dung A them canh ta dùng L luu luon trong so

//Có trong so là có cạnh

G->L[v][u] = w;

}

void Dijkstra(Graph\* G, int s) {

int i, j, it;

int cnt[G->n]; //so duong di ngan nhat cua moi dinh

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

mark[i] = 0;

cnt[i]=0; //Tat ca deu chua co duong di ngan nhat

}

pi[s] = 0;

p[s] = -1; //trước đỉnh s không có đỉnh nào cả

// lặp n hoặc n-1 lần đều được

cnt[1]=1; //Vi dinh 1 xuat phat nen mac dinh co 1 duong ngan nhat

for (it = 1; it <= G->n; it++) {

//1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

i = j;

}

//Đánh dấu i đã xét

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[i] = 1;

//2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)

for (j = 1; j <= G->n; j++)

if (G->L[i][j] != NO\_EDGE && mark[j] == 0) {

if (pi[i] + G->L[i][j] < pi[j]) {

pi[j] = pi[i] + G->L[i][j];

p[j] = i;

cnt[j] = cnt[i]; //cap nhat duong di ngan nhat theo i

}

else if (pi[i] + G->L[i][j] == pi[j]) cnt[j] += cnt[i]; //neu bang lay so luong tong duong di ngan nhat cua 2 ben

//printf("Sua pi[%d] = %d, p[%d] = %d, cnt[%d] = %d\n", j, pi[j], j, p[j], j, cnt[j]);

}

}

//for (i = 1; i <= G->n; i++) printf("pi[%d] = %d, p[%d] = %d, cnt[%d] = %d\n", i, pi[i], i, p[i], i, cnt[i]);

if (pi[G->n] == INFINITY) printf("-1 0");

else printf("%d %d",pi[G->n], cnt[G->n]);

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, w, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &w);

add\_edge(&G, u, v, w);

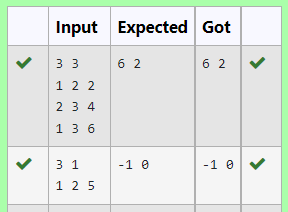
}

//---Het nhap---

Dijkstra(&G, 1);

return 0;

}



—-------------------------------------

# [Thực hành buổi 4](https://else.ctu.edu.vn/course/view.php?id=54#section-8)

## Bài 1- Xếp hạng đồ thị

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, ... %)

### Chiểu rộng

//Chieurong

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int data[max\_n];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){ //enQ

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){ //Front

return Q->data[Q->front];

}

void pop(Queue\* Q){ //deQ

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

//------------------------

void topo\_sort (Graph \*G){

int d[max\_n], hang[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

Queue Q;

make\_null\_queue(&Q);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push (&Q, u);

hang[u]=0;

}

while (!empty(&Q)){

u = top(&Q); //dinh co d[u]==0

pop(&Q);

//printf ("%d\n", u);

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0){

push(&Q, v);

hang[v]=hang[u]+1;

//printf("hang[%d]=%d\n", v, hang[v]);

}

}

}

}

for (u=1; u<=G->n; u++){

printf("%d\n", hang[u]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

topo\_sort(&G);

return 0;

}

### List

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

//------------------------

int hang[max\_n];

void rank(Graph \*G){

int d[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=0; //Tùy bai toán mà có thể là 0 hoặc 1

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

hang[u]=k;

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0) push\_back(&L2, v);

}

}

}

copy\_list (&L1,&L2);

k++;

}

for (u=1; u<=G->n; u++){

printf("%d\n", hang[u]);

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

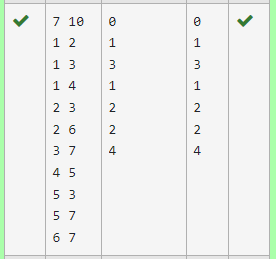
add\_edge(&G, u, v);

}

rank(&G);

return 0;

}



## Bài 2 - Cân đá

Peter rất thích chơi đá. Anh ta thường dùng đá để trang trí sân nhà của mình. Hiện tại Peter có n hòn đá. Dĩ nhiên mỗi hòn đá có một khối lượng nào đó. Peter muốn đặt các hòn đá này dọc theo lối đi từ cổng vào nhà của mình. Peter lại muốn sắp xếp như thế này: hòn đá nặng nhất sẽ đặt ở cạnh cổng rào, kế tiếp là hòn đá nặng thứ 2, ... hòn đá nhẹ nhất sẽ được đặt cạnh nhà. Như vậy nếu đi từ trong nhà ra cổng, ta sẽ gặp các hòn đá có khối lượng tăng dần.

Tuy nhiên, điều khó khăn đối với Peter là anh chỉ có một cây cân đĩa mà không có quả cân nào. Nói cách khác, mỗi lần cân Peter chỉ có thể biết được hòn đá nào nhẹ hơn hòn đá nào chứ không biết nó nặng bao nhiêu kg.

Sau m lần cân, Peter biết được sự khác nhau về cân nặng của m cặp. Với các thông tin này, hãy giúp Peter sắp xếp các viên đá theo thứ tự anh mong muốn.

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 hòn đó và số lần cân

- m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng hòn đá u nhẹ hơn hòn đá v.

Đầu ra (Output):

In ra màn hình thứ tự của các hòn đá theo khối lượng tăng dần. In các số thứ tự trên cùng một dòng, mỗi số cách nhau một khoảng trắng.

Bạn có thể yên tâm là dữ liệu đầu được giả sử rằng chỉ có một kết quả quả duy nhất.

Xem thêm ví dụ bên dưới. Trong ví dụ đầu tiên ta có: hòn đá 1 nhẹ nhất, kế đến là hòn đá 3 và sau cùng là hòn đá 2.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 2  1 3  3 2 | 1 3 2 |
| 7 13  1 2  1 3  1 4  2 3  2 6  3 7  4 5  5 3  5 7  6 7  2 4  6 5  4 6 | 1 2 4 6 5 3 7 |

**Answer:**(penalty regime: 0, 0, ... %)

### Chiểu rộng

//Chieurong

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int data[max\_n];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){ //enQ

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){ //Front

return Q->data[Q->front];

}

void pop(Queue\* Q){ //deQ

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

//------------------------

void topo\_sort (Graph \*G){

int d[max\_n], hang[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

Queue Q;

make\_null\_queue(&Q);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push (&Q, u);

hang[u]=0;

printf("%d ", u);

}

while (!empty(&Q)){

u = top(&Q); //dinh co d[u]==0

pop(&Q);

//printf ("%d\n", u);

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0){

push(&Q, v);

hang[v]=hang[u]+1;

//printf("hang[%d]=%d\n", v, hang[v]);

printf("%d ", v);

}

}

}

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

topo\_sort(&G);

return 0;

}

### List

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

//------------------------

int hang[max\_n];

void rank(Graph \*G){

int d[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

printf("%d ", u);

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=0; //Tùy bai toán mà có thể là 0 hoặc 1

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

hang[u]=k;

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0){

printf("%d ", v);

push\_back(&L2, v);

}

}

}

}

copy\_list (&L1,&L2);

k++;

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

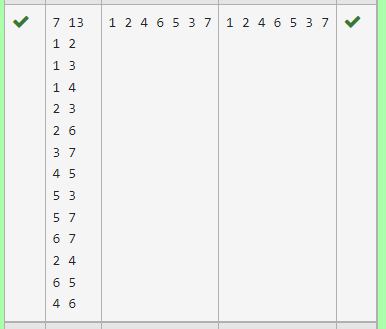
add\_edge(&G, u, v);

}

rank(&G);

return 0;

}



## Bài 3 - Chia kẹo

Cô giáo Trang chuẩn bị kẹo để phát cho các bé mà cô đang giữ. Dĩ nhiên môi bé đều có một tên gọi rất dễ thương ví dụ: Mạnh Phát, Diễm Quỳnh, Đăng Khoa, ... Tuy nhiên, để đơn giản vấn đề ta có thể giả sử các em được đánh số từ 1 đến n.

Cô giáo muốn rằng tất cả các em đều phải có kẹo. Cô lại biết thêm rằng có một số bé có ý muốn hơn bạn mình một chút vì thế các em ấy muốn kẹo của mình nhiều hơn của bạn.

Hãy viết chương trình giúp cô tính xem mỗi em cần được chia ít nhất bao nhiêu kẹo và tổng số kẹo ít nhất mà cô phải chuẩn bị là bao nhiêu.

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 bé và số cặp bé mà trong đó có 1 bé muốn có kẹo hơn bạn mình..

- m dòng tiếp theo mỗi dòng chứa 2 số nguyên a, b nói rằng bé a muốn có kẹo nhiều hơn bé b.

Đầu ra (Output):

In ra màn hình số kẹo ít nhất của từng em, mỗi em trên một dòng.

Dòng cuối cùng in tổng số kẹo ít nhất mà cô giáo Trang cần phải chuẩn bị

Chú ý:

Xem thêm các ví dụ để hiểu thêm về đầu vào và đầu ra.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 2  1 3  3 2 | 3  1  2  6 |
| 7 10  2 1  3 1  4 1  3 2  6 2  7 3  5 4  3 5  7 5  7 6 | 1  2  4  2  3  3  5  20 |

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

### Chiểu rộng

//Chieurong

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

#define max\_elements 100

typedef struct{

int data[max\_n];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){ //enQ

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){ //Front

return Q->data[Q->front];

}

void pop(Queue\* Q){ //deQ

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[y][x]=1; //Du lieu dau vao nguoc nen dao chieu

//G->A[y][x]=1; Co huong nen an

}

//------------------------

void topo\_sort (Graph \*G){

int d[max\_n], hang[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

Queue Q;

make\_null\_queue(&Q);

int tong=0;

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push (&Q, u);

hang[u]=1;

//printf("%d\n", u);

tong=tong+hang[u];

}

while (!empty(&Q)){

u = top(&Q); //dinh co d[u]==0

pop(&Q);

//printf ("%d\n", u);

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0){

push(&Q, v);

hang[v]=hang[u]+1;

//printf("hang[%d]=%d\n", v, hang[v]);

//printf("%d\n", v);

tong=tong+hang[v];

}

}

}

}

for (u=1; u<=G->n; u++){

printf("%d\n", hang[u]);

}

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

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

}

topo\_sort(&G);

return 0;

}

### List

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[y][x]=1; //Du lieu dau vao nguoc nen dao chieu

//G->A[y][x]=1; Co huong nen an

}

//------------------------

int hang[max\_n];

void rank(Graph \*G){

int d[max\_n];

int u, x, v, tong=0;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=1; //Tùy bai toán mà có thể là 0 hoặc 1

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

hang[u]=k;

tong=tong+hang[u];

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0) push\_back(&L2, v);

}

}

}

copy\_list (&L1,&L2);

k++;

}

for (u=1; u<=G->n; u++){

printf("%d\n", hang[u]);

}

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

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

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);

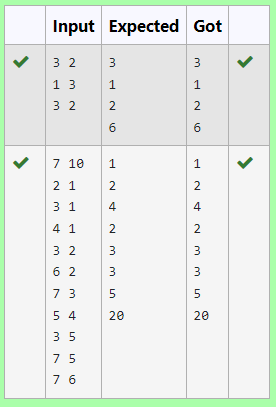
add\_edge(&G, u, v);

}

rank(&G);

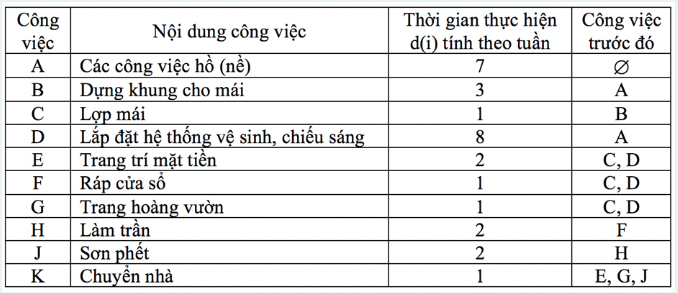
return 0;

}



## Bài 4 - Tổ chức thi công - Dự án xây nhà

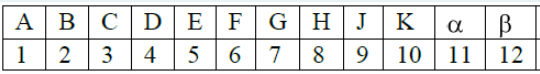
Có một dự án xây nhà với 10 công việc được cho như bảng sau:



Người ta cần:

- Xác định thời điểm sớm nhất và trể nhất để bắt đầu cho mỗi công việc mà không ảnh hưởng đến tiến độ của dự án

Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 ...thay vì A, B, C....như sau:



và lưu vào tập tin theo định dạng như bảng giá trị đầu vào

Hãy viết chương trình tìm thời gian sớm nhất hoàn thành dự án và Thời điểm sớm nhất và trể nhất để bắt đầy cho mỗi công việc của dự án mà không ảnh hưởng đến tiến độ của dự án.

##### Đầu vào:

10

7 0

3 1 0

1 2 0

8 1 0

2 3 4 0

1 3 4 0

1 3 4 0

2 6 0

2 8 0

1 5 7 9 0

Dòng đầu tiên là số công việc (10), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 7 và danh sách các công việc trước đó rỗng.

Công việc 2 (công việc B) có d[2] = 3 và danh sách công việc trước đó là {1}.

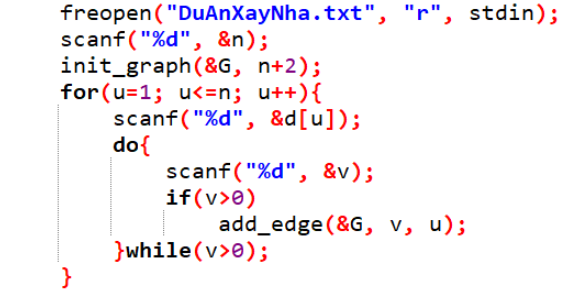
##### Đầu ra:

Dòng đầu tiên: Thời gian sớm nhất hoàn thành dự án

Mỗi dòng tiếp theo: In ra thời gian sớm nhất và thời trể nhất để bắt đầu cho mỗi công việc (1 => n+2, gồm cả công việc alpha và beta)

t(u)-T(u)

##### Chú ý đọc dữ liệu:



**For example:**

| **Input** | **Result** |
| --- | --- |
| 10  7 0  3 1 0  1 2 0  8 1 0  2 3 4 0  1 3 4 0  1 3 4 0  2 6 0  2 8 0  1 5 7 9 0 | 21  0-0  7-11  10-14  7-7  15-18  15-15  15-19  16-16  18-18  20-20  0-0  21-21 |

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

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

int min (int a, int b){

return a<b?a:b;

}

int max (int a, int b){

return a<b?b:a;

}

//------------------------

int hang[max\_n];

void rank(Graph \*G, List \*L){

int d[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=0; //Tùy bai toán mà có thể là 0 hoặc 1

make\_null\_list(L);

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

push\_back(L, u);

hang[u]=k;

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0) push\_back(&L2, v);

}

}

}

copy\_list (&L1,&L2);

k++;

}

}

int d[max\_n]; //tgian hoàn thành

int main(){

Graph G;

int n, u, x, v, j;

//1. Đọc đồ thị

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

scanf("%d", &n);

//1a. Tạo đồ thị có n+2 đỉnh (alpha = n+1 và beta=n+2)

init\_graph(&G, n+2);

int alpha = n+1, beta = n+2;

d[alpha]=0;

//1b. Đọc danh sách các công việc

for (u = 1; u <= n; u++) {

scanf("%d", &d[u]);

do{

scanf("%d", &x);

if (x>0) add\_edge(&G, x, u);

} while (x>0);

}

//2. Thêm cung nối alpha với các đỉnh có bậc vào =0

for (u = 1; u <= n; u++) {

int deg\_neg =0;

for (x=1; x<=n; x++)

if(G.A[x][u]>0) deg\_neg++;

if (deg\_neg==0) add\_edge(&G, alpha, u);

}

//3. Thêm cung nối các đỉnh có bậc ra =0 vào beta

for (u = 1; u <= n; u++) {

int deg\_pos =0;

for (v=1; v<=n; v++)

if(G.A[u][v]>0) deg\_pos++;

if (deg\_pos==0) add\_edge(&G, u, beta);

}

//4. Xếp thứ tự các đỉnh theo giải thuật sắp xếp topo

List L;

rank(&G, &L);

//5. Tính t[u]

int t[max\_n];

t[alpha]=0;

for (j=2; j<=L.size; j++){

int u = element\_at(&L, j);

t[u]= -999999;

for (x=1; x<=G.n; x++)

if (G.A[x][u]>0) t[u]= max(t[u], t[x]+d[x]);

}

//6. Tính T[u]

int T[max\_n];

T[beta]=t[beta];

for (j=L.size-1; j>=1; j--){

int u = element\_at(&L, j);

T[u]= 999999;

for (v=1; v<=G.n; v++)

if (G.A[u][v]>0) T[u]= min(T[u], T[v]-d[u]);

}

//7. In kết quả: in t[u] và T[u] ra màn hình

printf ("%d\n", t[beta]);

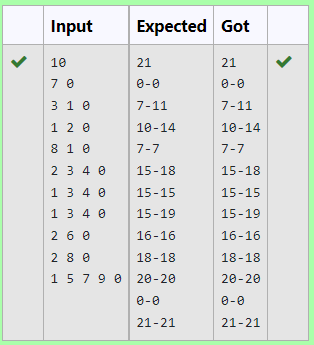
for (u = 1; u <= n+2; u++){

printf ("%d-%d\n", t[u], T[u]);

}

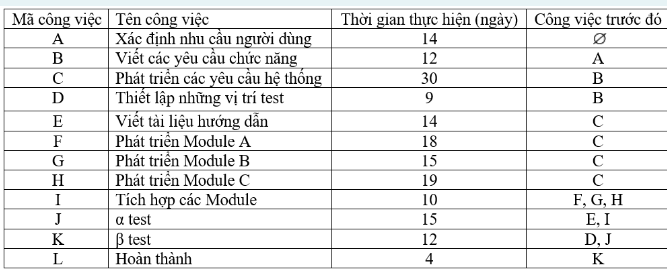
return 0;

}



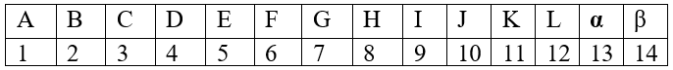
## Bài 5 - Tổ chức thi công - Dự án phần mềm

Việc thực hiện một dự án phát triển phần mềm được bố trí thành các công việc và thời gian thực hiện như sau:



- Anh Tuấn là một thành viên trong nhóm phát triển phần mềm. Anh ta thường hay hỏi mọi thành viên trong nhóm các câu hỏi tương tự như sau: "Nếu công việc E mình bắt đầu làm vào ngày thứ 60 thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?" "Nếu công việc H mình bắt đầu làm vào ngày thứ 50 thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?". Anh ta hỏi mọi người hoài những câu hỏi tương tự như thế làm cho các thành viên trong nhóm bực bội. Biết rằng dựa vào bảng công việc người ta có thể xác định thời điểm sớm nhất và trể nhất để bắt đầu cho mỗi công việc mà không ảnh hưởng đến tiến độ của dự án phần mềm. Hãy viết chương trình để giúp anh Tuấn tự trả lời câu hỏi của mình.

Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 thay vì A, B, C và lưu vào tập tin theo định dạng như sau:



### **Đầu vào:**

12

14 0

12 1 0

30 2 0

9 2 0

14 3 0

18 3 0

15 3 0

19 3 0

10 6 7 8 0

15 5 9 0

12 4 10 0

4 11 0

5 60

Dòng đầu tiên là số công việc (12), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 14 và danh sách các công việc trước đó rỗng.

Công việc 2 (công việc B) có d[2] = 12 và danh sách công việc trước đó là {1}.

Dòng cuối cùng: công việc u và thời gian bắt đầu t, hai giá trị u và t tương ứng với câu hỏi của anh Tuấn: "Nếu công việc u mình bắt đầu làm vào ngày thứ t thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?"

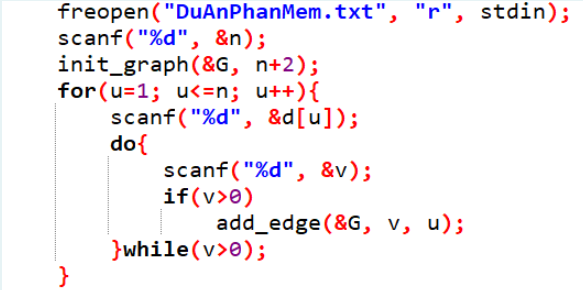
##### Đầu ra:

Yes: Nếu ngày bắt đầu thực hiên công việc nằm trong thời điểm sớm nhất và trể nhất để bắt đầu công việc tương ứng.

No: Nếu ngày bắt đầu thực hiện công việc KHÔNG nằm trong thời điểm sớm nhất và trể nhất để bắt đầu công việc tương ứng.

Ví dụ: Công việc 5, mình có thể bắt đầu làm vào ngày thứ 60 được hay không? => YES (Vì Thời gian sớm nhất và thời gian trể nhất thực hiện công việc 5 là: 56-71, 60 nằm trong khoảng thời gian cho phép)

##### Chú ý đọc dữ liệu:



**For example:**

| **Input** | **Result** |
| --- | --- |
| 12  14 0  12 1 0  30 2 0  9 2 0  14 3 0  18 3 0  15 3 0  19 3 0  10 6 7 8 0  15 5 9 0  12 4 10 0  4 11 0  4 60 | YES |
| 12  14 0  12 1 0  30 2 0  9 2 0  14 3 0  18 3 0  15 3 0  19 3 0  10 6 7 8 0  15 5 9 0  12 4 10 0  4 11 0  5 60 | YES |
| 12  14 0  12 1 0  30 2 0  9 2 0  14 3 0  18 3 0  15 3 0  19 3 0  10 6 7 8 0  15 5 9 0  12 4 10 0  4 11 0  7 70 | NO |

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

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

int min (int a, int b){

return a<b?a:b;

}

int max (int a, int b){

return a<b?b:a;

}

//------------------------

int hang[max\_n];

void rank(Graph \*G, List \*L){

int d[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=0; //Tùy bai toán mà có thể là 0 hoặc 1

make\_null\_list(L);

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

push\_back(L, u);

hang[u]=k;

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0) push\_back(&L2, v);

}

}

}

copy\_list (&L1,&L2);

k++;

}

}

int d[max\_n]; //tgian hoàn thành

int main(){

Graph G;

int n, u, x, v, j;

//1. Đọc đồ thị

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

scanf("%d", &n);

//1a. Tạo đồ thị có n+2 đỉnh (alpha = n+1 và beta=n+2)

init\_graph(&G, n+2);

int alpha = n+1, beta = n+2;

d[alpha]=0;

//1b. Đọc danh sách các công việc

for (u = 1; u <= n; u++) {

scanf("%d", &d[u]);

do{

scanf("%d", &x);

if (x>0) add\_edge(&G, x, u);

} while (x>0);

}

int duan, tgianbd;

scanf("%d%d", &duan, &tgianbd);

//2. Thêm cung nối alpha với các đỉnh có bậc vào =0

for (u = 1; u <= n; u++) {

int deg\_neg =0;

for (x=1; x<=n; x++)

if(G.A[x][u]>0) deg\_neg++;

if (deg\_neg==0) add\_edge(&G, alpha, u);

}

//3. Thêm cung nối các đỉnh có bậc ra =0 vào beta

for (u = 1; u <= n; u++) {

int deg\_pos =0;

for (v=1; v<=n; v++)

if(G.A[u][v]>0) deg\_pos++;

if (deg\_pos==0) add\_edge(&G, u, beta);

}

//4. Xếp thứ tự các đỉnh theo giải thuật sắp xếp topo

List L;

rank(&G, &L);

//5. Tính t[u]

int t[max\_n];

t[alpha]=0;

for (j=2; j<=L.size; j++){

int u = element\_at(&L, j);

t[u]= -999999;

for (x=1; x<=G.n; x++)

if (G.A[x][u]>0) t[u]= max(t[u], t[x]+d[x]);

}

//6. Tính T[u]

int T[max\_n];

T[beta]=t[beta];

for (j=L.size-1; j>=1; j--){

int u = element\_at(&L, j);

T[u]= 999999;

for (v=1; v<=G.n; v++)

if (G.A[u][v]>0) T[u]= min(T[u], T[v]-d[u]);

}

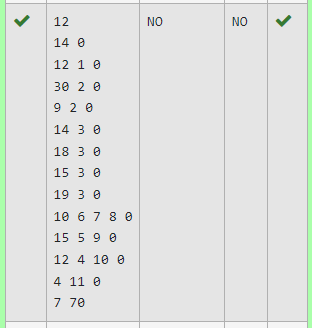
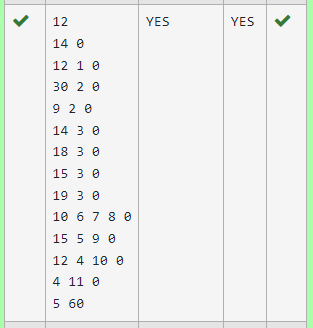
//7. In kết quả: in t[u] và T[u] ra màn hình

if (tgianbd>=t[duan] && tgianbd<=T[duan]) printf("YES");

else printf("NO");

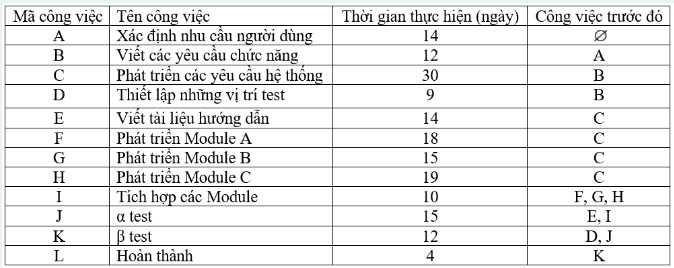
return 0;

}



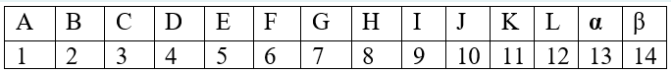
—------------------------------------

Việc thực hiện một dự án phát triển phần mềm được bố trí thành các công việc và thời gian thực hiện như sau:



- Biết rằng dựa vào bảng công việc người ta có thể xác định thời điểm sớm nhất và trể nhất để bắt đầu cho mỗi công việc mà không ảnh hưởng đến tiến độ của dự án phần mềm. Trong danh sách các công việc thì Trưởng dự án cực kỳ quan tâm đến các công việc then chốt. Công việc then chốt là những công việc có thời gian sớm nhất và thời gian trể nhất bắt đầu công việc bằng nhau. Hãy viết chương trình cho trưởng dự án biết những công việc nào là công việc then chốt?

Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 thay vì A, B, C và lưu vào tập tin theo định dạng như sau:



### **Đầu vào:**

12

14 0

12 1 0

30 2 0

9 2 0

14 3 0

18 3 0

15 3 0

19 3 0

10 6 7 8 0

15 5 9 0

12 4 10 0

4 11 0

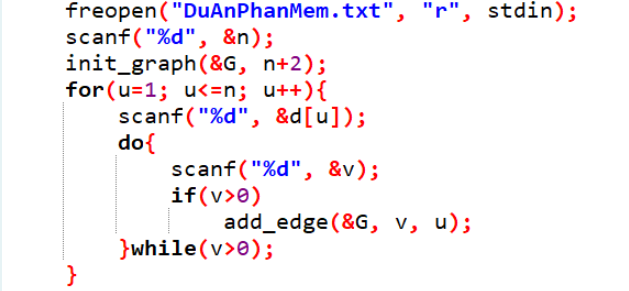
Dòng đầu tiên là số công việc (12), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 14 và danh sách các công việc trước đó rỗng.

Công việc 2 (công việc B) có d[2] = 12 và danh sách công việc trước đó là {1}.

##### **Đầu ra:**

Danh sách các công việc then chốt. Mỗi công việc trên một dòng.

##### Chú ý đọc dữ liệu:



**For example:**

| **Input** | **Result** |
| --- | --- |
| 12  14 0  12 1 0  30 2 0  9 2 0  14 3 0  18 3 0  15 3 0  19 3 0  10 6 7 8 0  15 5 9 0  12 4 10 0  4 11 0 | 1  2  3  8  9  10  11  12  13  14 |

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

//List

#include<stdio.h>

#define max\_n 100

#define max\_edges 500

//Khai báo CTDL List và các phép toán cơ bản

typedef int ElementType;

typedef struct {

ElementType data[max\_n];

int size;

} List;

//Tạo danh sách rỗng

void make\_null\_list(List \*L) {

L->size = 0;

}

//Thêm một phần tử vào cuối danh sách

void push\_back(List \*L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

//Lấy phần tử thứ i, phần tử bắt đầu có vị trí 1

ElementType element\_at(List \*L, int i) {

return L->data[i-1];

}

//Trả về số phần tử của danh sách

int count\_list(List \*L) {

return L->size;

}

void copy\_list(List \*L1, List \*L2){

int i, x;

make\_null\_list(L1);

for(i=1;i<=L2->size;i++){

x=element\_at(L2,i);

push\_back(L1,x);

}

}

//-------------------------

typedef struct{

int n;

int A[max\_n][max\_n];

}Graph;

void init\_graph (Graph\* G, int n){

int i, j;

G->n=n;

for (i=1; i<=n; i++){

for (j=1; j<=n; j++)

G->A[i][j]=0;

}

}

void add\_edge (Graph\* G, int x, int y){

G->A[x][y]=1;

//G->A[y][x]=1; Co huong nen an

}

int min (int a, int b){

return a<b?a:b;

}

int max (int a, int b){

return a<b?b:a;

}

//------------------------

int hang[max\_n];

void rank(Graph \*G, List \*L){

int d[max\_n];

int u, x, v;

//Tinh bac vao cua y

for (u=1; u<=G->n; u++){

d[u]=0;

for (x=1; x<=G->n; x++){

if (G->A[x][u]!=0) d[u]++;

}

//printf ("d[%d]=%d\n", u, d[u]);

}

List L1, L2;

make\_null\_list(&L1);

//Dua cac dinh co d[u]=0 vao hang doi

for (u=1; u<=G->n; u++)

if(d[u]==0){

push\_back (&L1, u); //Hàng đợi cb duyệt

}

int k=0; //Tùy bai toán mà có thể là 0 hoặc 1

make\_null\_list(L);

while (L1.size>0){

make\_null\_list(&L2);

for (x=1; x<=L1.size; x++){

int u = element\_at(&L1, x);

push\_back(L, u);

hang[u]=k;

for (v=1; v<=G->n; v++){

if (G->A[u][v] !=0){

d[v]--;

if (d[v]==0) push\_back(&L2, v);

}

}

}

copy\_list (&L1,&L2);

k++;

}

}

int d[max\_n]; //tgian hoàn thành

int main(){

Graph G;

int n, u, x, v, j;

//1. Đọc đồ thị

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

scanf("%d", &n);

//1a. Tạo đồ thị có n+2 đỉnh (alpha = n+1 và beta=n+2)

init\_graph(&G, n+2);

int alpha = n+1, beta = n+2;

d[alpha]=0;

//1b. Đọc danh sách các công việc

for (u = 1; u <= n; u++) {

scanf("%d", &d[u]);

do{

scanf("%d", &x);

if (x>0) add\_edge(&G, x, u);

} while (x>0);

}

//2. Thêm cung nối alpha với các đỉnh có bậc vào =0

for (u = 1; u <= n; u++) {

int deg\_neg =0;

for (x=1; x<=n; x++)

if(G.A[x][u]>0) deg\_neg++;

if (deg\_neg==0) add\_edge(&G, alpha, u);

}

//3. Thêm cung nối các đỉnh có bậc ra =0 vào beta

for (u = 1; u <= n; u++) {

int deg\_pos =0;

for (v=1; v<=n; v++)

if(G.A[u][v]>0) deg\_pos++;

if (deg\_pos==0) add\_edge(&G, u, beta);

}

//4. Xếp thứ tự các đỉnh theo giải thuật sắp xếp topo

List L;

rank(&G, &L);

//5. Tính t[u]

int t[max\_n];

t[alpha]=0;

for (j=2; j<=L.size; j++){

int u = element\_at(&L, j);

t[u]= -999999;

for (x=1; x<=G.n; x++)

if (G.A[x][u]>0) t[u]= max(t[u], t[x]+d[x]);

}

//6. Tính T[u]

int T[max\_n];

T[beta]=t[beta];

for (j=L.size-1; j>=1; j--){

int u = element\_at(&L, j);

T[u]= 999999;

for (v=1; v<=G.n; v++)

if (G.A[u][v]>0) T[u]= min(T[u], T[v]-d[u]);

}

//7. In kết quả: in t[u] và T[u] ra màn hình

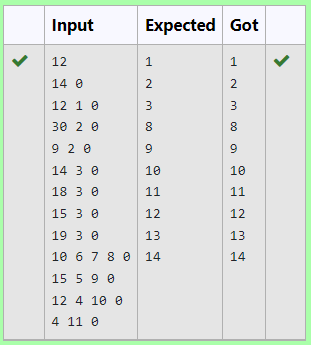
for (u = 1; u <= n+2; u++) {

if (t[u]==T[u]) printf("%d\n", u);

}

return 0;

}



# [Thực hành buổi 5](https://else.ctu.edu.vn/course/view.php?id=54#section-5)

## Bài 5.1. Tìm cây khung bằng giải thuật Kruskal

Viết chương trình tìm cây khung có trọng số nhỏ nhất bằng giải thuật Kruskal.

Đầu vào:

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 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra:

In ra màn hình trọng số của cây khung tìm được và danh sách các cung theo thứ tự tăng dần của trọng số.

Quy ước: các cung được in ra theo định dạng:

u v w

với (u < v), mỗi cung trên 1 dòng. Nếu hai cung có trọng số bằng nhau thì cung nào có u nhỏ hơn sẽ được in trước. Nếu có trọng số bằng nhau và u giống nhau thì cung nào có v nhỏ hơn sẽ in trước.

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 4  1 2 3  2 3 6  4 3 1  1 4 9 | 10  3 4 1  1 2 3  2 3 6 |
| 4 4  1 2 3  3 2 5  4 2 5  3 4 9 | 13  1 2 3  2 3 5  2 4 5 |

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

#include<stdio.h>

#define max\_n 100

typedef struct {

int u,v;

int w;

}Edge;

typedef struct {

int n, m;

Edge edges[max\_n];

} Graph;

void init\_graph(Graph \*G, int n) {

G->n =n;

G->m=0;

}

void add\_edge(Graph \*G, int u, int v, int w) {

G->edges[G->m].u=u;

G->edges[G->m].v=v;

G->edges[G->m].w=w;

G->m++;

}

int parent[max\_n];

//Tìm gốc đệ quy

/\*int findRoot(int u) {

if (parent[u] = u)

return u;

return findRoot(parent[u]);

}\*/

//Tìm gốc lặp

int findRoot(int u) {

while (parent[u] != u) u = parent[u];

return u;

}

int Kruskal(Graph\* G, Graph\* T) {

int i, j;

Edge temp;

//Sắp xếp các cung của G theo thứ tự trọng số tăng dần

for (i=0; i < G->m; i++) {

for(j=i+1; j < G->m; j++){

if(G->edges[i].w > G->edges[j].w){

temp = G->edges[i];

G->edges[i] = G->edges[j];

G->edges[j] = temp;

}

}

}

//for (i = 0; i < G->m; i++)

//printf("%d %d %d\n", G->edges[i].u, G->edges[i].v, G->edges[i].w);

//printf("---\n");

int u, e;

// Khởi tạo T rỗng.

init\_graph(T, G->n);

for (u = 1; u <= G->n; u++)

parent[u] = u; //Mỗi đỉnh u là một bộ phận liên thông

int sum\_w = 0;

//Duyệt qua các cung của G (đã sắp xếp)

for (e = 0; e <= G->m; e++) {

int u = G->edges[e].u;

int v = G->edges[e].v;

int w = G->edges[e].w;

int root\_u = findRoot(u);

int root\_v = findRoot(v);

if (root\_u != root\_v) {

add\_edge(T, u, v, w);

//Gộp 2 BPLT root\_u và root\_v lại

parent[root\_v] = root\_u;

sum\_w += w;

}

}

return sum\_w;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.

Graph G, T;

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, &w);

add\_edge(&G, u, v, w);

}

int sum\_w = Kruskal(&G, &T);

printf("%d\n", sum\_w);

for (e = 0; e < T.m; e++){

if(T.edges[e].u < T.edges[e].v)

printf("%d %d %d\n", T.edges[e].u, T.edges[e].v, T.edges[e].w);

else printf("%d %d %d\n", T.edges[e].v, T.edges[e].u, T.edges[e].w);

}

return 0;

}



## Bài 5.2. tìm cây khung có trọng lượng nhỏ nhất bằng giải thuật Prim

Viết chương trình tìm cây khung có trọng số nhỏ nhất bằng giải thuật Prim.

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 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình theo định dạng sau:

- Dòng đầu tiên in trọng số của cây khung tìm được

- n - 1 dòng kế tiếp in ra các cung của cây tìm tìm được theo định dạng: u v w. Cung (u1, v1) sẽ được in ra trước cung (u2, v2) nếu (u1 < u2) hoặc (u1 = u2 và v1 < v2).

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4 4  1 2 4  2 3 6  3 4 1  1 4 8 | 11  1 2 4  2 3 6  3 4 1 |

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

#include<stdio.h>

#define MAXN 1000

#define NO\_EDGE 0

#define INFINITY 99999;

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

int chon[MAXN];

int sochon=0;

typedef struct {

int n;

int W[MAXN][MAXN];

} Graph;

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->W[i][j] = NO\_EDGE;

}

void add\_edge(Graph \*G, int u, int v, int w){

G->W[u][v] = w; //thay vi dung A them canh ta dùng W luu luon trong so

//Có trong so là có cạnh

G->W[v][u] = w;

}

int Prim(Graph\* G, Graph\* T, int s) {

int i, j, u, v;

//1. Khởi tạo

for (i = 1; i <= G->n; i++) {

pi[i] = INFINITY;

p[i]=-1;

mark[i] = 0;

}

pi[s] = 0;

p[s] = 0; //chỉ p[s] bang 0

//2. Lặp n-1 lần

for (i = 1; i <= G->n; i++) {

//2a. Tìm u gần S nhất (có pi[u] nhỏ nhất)

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi) {

min\_pi = pi[j];

u = j;

}

//2b. Đánh dấu u

//printf("Danh dau%d[%d] = %d, p[%d] = %d\n", it, i, pi[i], i, p[i]);

mark[u] = 1;

//2c. Cập nhật pi và p của các đỉnh kề của u (nếu thoả)

for (v = 1; v <= G->n; v++)

if (G->W[u][v] != NO\_EDGE) {

if ( pi[v] > G->W[u][v] && mark[v] == 0) {

pi[v] = G->W[u][v];

p[v] = u;

}

//printf("Sua pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);

}

}

//3. Dựng cây dựa vào p[i]: thêm cung (p[i],i) vào cây T

init\_graph(T, G->n);

int sum\_w=0;

for (i = 1; i <= G->n; i++) {

if(p[i] != -1){

int w = G->W[p[i]][i];

add\_edge(T, p[i], i, w);

chon[sochon]=i;

sochon++;

sum\_w += w;

}

}

return sum\_w;

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.

Graph G, T;

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, &w);

add\_edge(&G, u, v, w);

}

int sum\_w = Prim(&G, &T, 1);

printf("%d\n", sum\_w);

for (e = 1; e < sochon; e++){

u = chon[e];

v = p[u];

w = G.W[p[u]][u];

if(u<v) printf("%d %d %d\n", u, v, w);

else printf("%d %d %d\n", v, u, w);

}

return 0;

}



## Bài 5.3. Tìm luồng cực đại trong mạng

Viết chương trình tìm luồng cực đại trên mạng bằng thuật toán Ford - Fullkerson (duyệt theo chiều rộng).

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 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

Đầu ra (Output):

In ra màn hình theo định dạng sau:

- Dòng đầu tiên in luồng cực đại theo dạng: Max flow: s

- Dòng thứ hai in các đỉnh của X0 theo dạng: X0: x1 x2 ..., mỗi đỉnh cách nhau 1 khoảng trắng

- Dòng thứ ba in các đỉnh của Y0 theo dạng: Y0: y1 y2 ..., mỗi đỉnh cách nhau 1 khoảng trắng

Xem thêm ví dụ bên dưới.

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, &c);

add\_edge(&G, u, v, w);

}

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 3  1 2 9  2 3 4  1 3 4 | Max flow: 8  X0: 1 2  Y0: 3 |
| 7 10  1 2 9  1 3 4  1 4 8  2 3 4  2 6 3  3 7 7  4 5 5  5 3 3  5 7 2  6 7 6 | Max flow: 12  X0: 1 2 3 4 5  Y0: 6 7 |

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

#include<stdio.h>

#define MAXN 100

#define NO\_EDGE 0

#define INF 9999999

//------------Queue--------------

typedef struct{

int data[MAXN];

int front, rear;

} Queue;

void make\_null\_queue(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void push (Queue\* Q, int x){ //enqueue

Q->rear++;

Q->data[Q->rear]=x;

}

int top (Queue\* Q){ //front

return Q->data[Q->front];

}

void pop(Queue\* Q){ //dequeue

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//------------Graph--------------

typedef struct {

int C[MAXN][MAXN]; //khả năng thông qua của cung

int F[MAXN][MAXN]; //luồng trên cung

int n;

} Graph;

typedef struct {

int dir; // >0: +1: +, -1: -, 0: chưa có nhãn

int p; //đỉnh trước

int sigma; //lượng tăng luồng

} Label;

Label labels[MAXN]; //nhãn các đỉnh

void init\_graph(Graph\* G, int n) {

G->n = n;

int i, j;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++){

G->C[i][j] = 0;

}

}

int min(int a, int b){

return a<b? a:b;

}

int FordFullkerson(Graph\* G, int s, int t) {

//I. Khởi tạo luồng = 0, gán F[u][v] = 0 với mọi u, v.

int u, v, x;

for (u=1; u<=G->n; u++){

for (v=1; v<=G->n; v++) G->F[u][v]=0;

}

int max\_flow=0;

//II. Lặp

Queue Q;

do {

int found=0;

//Bước 1 – xoá nhãn các đỉnh và gán nhãn cho s

//1.1 Xoá tất cả các nhãn cũ

for (u=1; u<=G->n; u++) labels[u].dir=0;

//1.2 Gán nhãn s: (+, s, oo)

labels[s].dir=1;

labels[s].p=s;

labels[s].sigma=INF;

//1.3 Khởi tạo Q rỗng và đưa s vào Q

make\_null\_queue(&Q);

push(&Q,s);

//Bước 2 Lặp gán nhãn cho các đỉnh tìm đường tăng luông

while (!empty(&Q)) {

//2.1 Lấy 1 đỉnh trong Q ra, gọi là u

u = top(&Q);

pop(&Q);

//2.2 Xét gán nhãn cho các đỉnh kề với u: cung thuận

for(v=1; v<=G->n; v++){

if (G->C[u][v]!=NO\_EDGE && labels[v].dir==0

&& G->F[u][v] < G->C[u][v]){

labels[v].dir=1;

labels[v].p=u;

labels[v].sigma=min(labels[u].sigma,

G->C[u][v] - G->F[u][v]);

push(&Q, v);

}

}

//2.3 Xét gán nhãn cho các đỉnh đi đến u: cung nghịch

for(x=1; x<=G->n; x++){

if (G->C[x][u]!=NO\_EDGE && labels[x].dir==0

&& G->F[x][u] >0){

labels[x].dir=-1;

labels[x].p=u;

labels[x].sigma=min(labels[u].sigma, G->F[x][u]);

push(&Q, x);

}

}

//2.4 Nếu t được gán nhãn => tìm được đường tăng luồng, thoát vòng lặp.

if (labels[t].dir!=0){

found =1;

break;

}

}

//Bước 3 Tăng luồng

if (found==1) {

int sigma=labels[t].sigma; //Lượng luồng tăng thêm

//3.1 Cập nhật luồng của các cung trên đường tăng luông

int u=t;

while(u!=s){

int p = labels[u].p;

if (labels[t].dir>0) G->F[p][u] += sigma; //Tăng luồng

else G->F[u][p] -= sigma; //Giảm luồng

u=p;

}

//3.2 Tăng giá trị luồng trên mạng

max\_flow += sigma;

}

else break; //thoát vòng lặp

} while (1);

return max\_flow; //trả về luồng cực đại

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi nộp bài nhớ bỏ dòng này.

Graph G;

int n, m, u, v, c, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n);

//Đọc cung trực tiếp

for (e = 0; e < m; e++) {

scanf("%d%d%d", &u, &v, &c);

G.C[u][v]=c;

}

int max\_flow =FordFullkerson(&G, 1, n);

printf("Max flow: %d", max\_flow);

printf("\nX0: ");

for (e = 1; e <= n; e++) {

if(labels[e].dir==1) printf("%d ", e);

}

printf("\nY0: ");

for (e = 1; e <= n; e++) {

if(labels[e].dir==0) printf("%d ", e);

}

return 0;

}

