**Tổ chức thi công**

#include<stdio.h>

#define MAX\_ELEMENTS 100

#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 isEmpty(Queue \*Q){

return Q->front > Q->rear;

}

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

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;

}

#define MAX\_VERTICES 100

#define MIN\_VALUES -9999999

#define MAX\_VALUES 9999999

//\*\*\*\*\*\*\*\*\* DO THI VOI MA TRAN KE (DINH - DINH) \*\*\*\*\*\*\*\*\*\*//

// Do thi vo huong - don cung //

typedef struct{

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

//Khoi tao Graph

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

int i,j;

G->n = n;

G->m = 0;

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

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

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

}

}

}

//Them cung vao Graph

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

G->m++;

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

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

}

//Kiem tra xem 2 dinh co ke nhau hay khong

int adjacent(Graph \*G, int x, int y){

return G->A[x][y] != 0;

}

//Tinh so bac cua mot dinh bat ky

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

int count =0;

int e;

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

count += G->A[x][e];

}

return count;

}

//Tim danh sach cac dinh ke cua dinh yeu cau

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

List L;

make\_null(&L);

int y;

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

if(adjacent(G, x, y)){

push\_back(&L, y);

}

}

return L;

}

void topo\_sort(Graph \*G, List \*list){

int d[MAX\_VERTICES];

int i,u;

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

d[i] = 0;

}

//tinh gia tri d[] cho cac dinh

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

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

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

d[u]++;

}

}

//tao danh sach L rong

make\_null(list);

//khoi tao queue

Queue Q;

make\_null\_queue(&Q);

//them cac dinh co d[u] = 0 vao Q

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

if(d[i] == 0)

push(&Q, i);

}

// vong lap while

while(!isEmpty(&Q)){

int y = top(&Q); pop(&Q);

push\_back(list, y);

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

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

d[u]--;

if(d[u] == 0)

push(&Q, u);

}

}

}

}

int min(int a, int b){

if(a<b){

return a;

}else{

return b;

}

}

int max(int a, int b){

if(a>b){

return a;

}else{

return b;

}

}

int d[MAX\_VERTICES];

int deg\_neg[MAX\_VERTICES];

int deg\_pos[MAX\_VERTICES];

int main(){

List list;

//1.doc do thi

int n,v,j,i,x;

int t[MAX\_VERTICES];

int T[MAX\_VERTICES];

Graph G;

freopen("tochucthicong.txt", "r", stdin);

scanf("%d", &n);

init\_graph(&G, n+2);

d[n+1] = 0; // dinh alpha khoi tao 0

//khoi tao cac bac

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

deg\_neg[i] = 0;

deg\_pos[i] = 0;

}

//nhap cung

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

scanf("%d", &d[i]); // thoi gian hoan thanh cv i

do{

scanf("%d", &x);

if(x>0){

add\_edge(&G, x, i);

deg\_neg[i]++;

deg\_pos[x]++;

}

}while(x>0);

}

//2.Them cung noi voi alpha cac dinh co bac vao = 0

//3.Them cung noi voi beta cac dinh co bac ra = 0

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

if(deg\_neg[i]==0)

add\_edge(&G,n+1, i);

if(deg\_pos[i] ==0)

add\_edge(&G, i, n+2);

}

//4. Xap xep thu tu topo va luu vao list

topo\_sort(&G, &list);

//5. tinh t[u] thoi gian som nhat bat dau cv

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

t[i] = MIN\_VALUES;

}

t[n+1] = 0; // thoi gian bat dau dinh alpha la 0

for(j=1; j<=list.size; j++){

int u = element\_at(&list, j);

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

if(G.A[x][u]!=0){

t[u] = max(t[u], t[x] + d[x]);

}

}

}

//6. Tinh T[u] thoi gian tre nhat bat dau cv

T[n+2] = t[n+2];

for(j=list.size-1; j>=1; j--){

int u = element\_at(&list, j);

T[u] = MAX\_VALUES;

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

if(G.A[u][v] > 0){

T[u] = min(T[u], T[v] - d[u]);

}

}

}

// for(i=1; i<=list.size; i++){

// printf("%d ", element\_at(&list, i));

//

// }

// printf("\n");

// //in t[u]

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

// printf("%d ", i);

// }

// printf("\n");

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

// printf("%d ", t[i]);

// }

// printf("\n");

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

// printf("%d ", T[i]);

// }

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

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

printf("%d-%d\n", t[i], T[i]);

}

return 0;

}

**Prim**

#include<stdio.h>

#define MAX\_VALUES 9999999

#define MAX\_VERTICES 100

#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

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;

}

typedef struct {

int n, m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

//khoi tao tat ca cac cung co trong so la 0

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

G->n = n;

int i,j;

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

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

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

}

}

}

//them cung

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

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

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

}

//kiem tra 2 dinh co ke hay khong

int adjacent(Graph \*G, int x, int y){

return G->A[x][y];

}

//Tinh bac cua dinh bat ky

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

int deg = 0;

int i;

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

if(G->A[i][x] || G->A[x][i])

deg++;

}

return deg;

}

void printGraph(Graph G){

int i,j;

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

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

if(G.A[i][j] != 0 && i<j){

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

}

}

}

}

int distanceFrom(int u, List \*S, Graph \*G){

int min\_dist = MAX\_VALUES;

int min\_v = -1;

int i;

for(i=1; i<=S->size; i++){

int v = element\_at(S, i);

if(G->A[u][v] != 0 && min\_dist > G->A[u][v]){

min\_dist = G->A[u][v];

min\_v = v;

}

}

return min\_v;

}

int mark[MAX\_VERTICES];

int Prim(Graph \*G, Graph \*T){

init\_graph(T, G->n);

List S;

make\_null(&S);

int i,u;

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

mark[i] = 0;

}

//chon dinh 1 dua vao s

push\_back(&S, 1);

mark[1] = 1;

int sum\_w = 0;

// vong lap chinh

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

// Tim u gan nhat voi S

int min\_dist = MAX\_VALUES, min\_u, min\_v;

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

if(mark[u] == 0){

int v = distanceFrom(u, &S, G);

if(v != -1 && G->A[u][v] < min\_dist){

min\_dist = G->A[u][v];

min\_u = u;

min\_v = v;

}

}

}

//Dua min\_u vao S

push\_back(&S, min\_u);

mark[min\_u] = 1;

// Dua cung da chon dc vao T

add\_edge(T, min\_u, min\_v, min\_dist);

sum\_w += min\_dist;

}

return sum\_w;

}

int main(){

int i;

int n,m,x,y,w;

Graph G,T;

freopen("prim.txt", "r", stdin);

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

init\_graph(&G, n);

//nhap cung

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

scanf("%d%d%d", &x, &y, &w);

add\_edge(&G, x, y, w);

}

//printGraph(G);

int kq = Prim(&G, &T);

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

printGraph(T);

return 0;

}