MỘT SỐ DẠNG BÀI TẬP SỬ DỤNG LUỒNG CỰC ĐẠI

Bài toán luồng cực đại trên mạng (Max Flow) là một trong những bài toán cơ bản của lý thuyết Đồ thị và nó thường được sử dụng để giải quyết một lớp các bài toán tối ưu. Dạng bài tập sử dụng luồng cực đại trên mạng cũng là một trong những dạng xuất hiện trong các đề thị chọn học sinh giỏi tin học tuy rằng tần số xuất hiện của chúng ít và đôi khi những bài tập có thể sử dụng luông cực đại để giải quyết thì cũng có thể được giải quyết bằng phương pháp khác. Tuy vậy, việc giảng dạy luồng cực đại cho học sinh chuyên tin là một phần bắt buộc trong chương trình tin học. Chuyên đề này không phải là một chuyên đề giảng dạy lý thuyết luồng cực đại. Ở đây, tôi chỉ phân loại các dạng toán thông dụng khác nhau sử dụng luồng cực đại để giải quyết. Qua đó xây dựng cho học sinh được một phong cách chung về tư duy khi áp dụng giải bài tập cơ sử dụng luồng cực đại trên mạng.

Có nhiều thuật toán khác nhau để giải quyết bài toán luồng cực đại trên mạng. Tuy nhiên, thuật toán Ford-Fulkerson, theo tôi có lẽ là thuật toán đơn giản nhất và cái chính là tư duy của thuật toán cho phép chúng ta tổng kết được các hình thức khác nhau sử dụng luồng cực đại trên mạng trong thực tế.

Nội dung của thuật toán Ford-Fulkerson được tóm tắt theo mô tả dưới đây:

```
B1: Khởi đầu bằng một luồng hợp lệ nào đó (thường là luồng 0)
WHILE (Tìm được đường tăng luồng) DO
    B2: Tăng luồng dọc theo đường tăng luồng tìm được
```

Các chi tiết cụ thể có thể tìm thấy trong tài liệu tham khảo Sách giáo khoa chuyên TIN Tập 2 (Nhà xuất bản Giáo dục)

I. Bài toán cặp ghép cực đại

Bài toán: Cho đồ thị hai phía G(X, Y) trong đó các cạnh được nối từ tập X sang tập Y. Hãy tìm một bộ cặp ghép cực đại.

Có thể nói đây là dạng bài tập phổ biến và hay sử dụng nhất của việc áp dụng luồng cực đại.

Giả sử $X = (x_1, x_2, \dots, x_m)$, $Y = (y_1, y_2, \dots, y_n)$. Xây dựng một mạng với đỉnh phát s và đỉnh thu t với các cung:

- Cùng từ s nối đến các đỉnh thuộc tập X với độ thông qua bằng 1
- Cung nối từ tập X đến tập Y (là cạnh của đồ thị hai phía) với độ thông qua bằng $+\infty$
- Cung nối từ các đỉnh của tập Y đến t có độ thông qua bằng 1

Có thể thấy một bộ ghép cực đại trên đồ thị hai phía tương ứng với một luồng cực đại trên mạng này và bài toán tìm cặp ghép có thể được giải quyết bằng luồng cực đại.

Tuy nhiên do đặc điểm của mạng ta có thể cải tiến để giảm độ phức tạp của bài toán luồng cực đại trong trường hợp này. Cụ thể như sau:

Việc mô tả luồng có thể được thay thế bằng hai mảng

```
int Fx[maxm];// Fx[i]=k>0 có nghĩa là đỉnh x_i được ghép với đỉnh y_k
int Fy[maxn];// Fy[k]=i>0 có nghĩa là đỉnh y_k được ghép với x_i
```

Với mô tả luồng như trên thì một đường tăng luồng có thể được mô tả như là một dãy:

$$x_{i_1}, y_{i_1}, x_{i_2} \equiv Fy[y_{i_1}], y_{i_2}, x_{i_3} \equiv Fy[y_{i_2}], y_{i_3}, x_{i_4} \equiv F[y_{i_3}], \dots., x_{i_k} \equiv F[y_{i_{k-1}}], y_{i_k}, t$$

Trong đó $Fx[x_{i_1}] \neq y_{i_1}, Fx[x_{i_2}] \neq y_{i_2}, \dots., Fx[x_{i_k}] \neq y_{i_k}, Fy[y_k] = 0$

Bởi vì "cung xuôi" ứng với cạnh không nằm trong cặp ghép và cung ngược ứng với cạnh đã ghép rồi. Do vậy một đường tăng luồng có thể mô tả bằng một số "nhịp":

$$(x_{i_1}, y_{i_1}, x_{i_2}), (x_{i_2}, y_{i_2}, x_{i_3}), \dots., (x_{i_k}, y_{i_k}, t)$$

Dễ thấy không cần nhớ $y_{i_1}(= Fx[x_{i_2}]), y_{i_2}(= Fx(x_{i_3}), \dots, y_{i_k}$(Qui ước $= Fx[0]$)

Do đó "độ dài" của đường tăng luồng có thể được đo bằng số lượng "nhịp":

$$d[x_{i_1}] = 0, d[x_{i_2}] = 1, \ldots., d[0] = d[x_{i_k}] + 1 = k + 1$$

(Chú ý ta coi đỉnh thu t là đỉnh số hiệu 0 ở bên trái).

Thuật toán tìm đường tăng luồng có thể viết:

```
bool FindPath() {
    int L=1, R=0;                           // Khởi động hàng đợi
    for(int u=1;u<=n;u++)
    if (Fx[u]==0) {                         // Fx[u]=0 có nghĩa u chưa ghép
        d[u]=0;                             // và được dùng làm xuất phát của
        q[++R]=u;                           // nhịp đầu tiên
    } else d[u]=INF;                        // d[u]=+∞ là đỉnh u chưa thăm
    d[0]=INF;                               // Đỉnh 0 chưa thăm
    while (L<=R) {
        int u=q[L++];
        FOR v ∈ Ke(u) {                     // (u,v) là cạnh của đồ thị
            if (d[Fy[v]]==INF) {
                d[Fy[v]]=d[u]+1;
                if (Fy[v]) q[++R]=y[v];
            }
        }
    }
    return (d[0]!=INF);
}
```

Một cải tiến quan trọng là trong một lần duyệt đồ thị theo chiều rộng (BFS) để tìm đường tăng luồng ta có thể tìm được nhiều đường tăng luồng khác nhau. Chính điều này cho ta một cải tiến quan trọng là bằng việc sử dụng một lần BFS ta có thể tìm được nhiều đường tăng luồng khác nhau (tương đương với việc ta có thể tăng cặp ghép thêm không chỉ một mà nhiều cạnh). Có thể thực hiện điều này bằng một hàm đệ qui từ một đỉnh u thuộc bên trái (X):

```
bool Ghep(int u) {
    if (u==0) return true;          // Do qui ước t=0
    FOR v∈Ke(u) {                   // (u,v) là cạnh của đồ thị hai phía
        if (d[Fy[v]]==d[u]+1)       // (u,v,Fy[v]) là một "nhịp"
        if (Ghep(Fy[v])) {          // Gọi đệ qui ghép Fy[v]∈X
            Fx[u]=v; Fy[v]=u;       // Nếu ghép được đổi lại Fx[u]=v, Fy[v]=u
            return true;            // Trả về true vì đã ghép được Fy[v]
        }
    }
    d[u]=INF;                       // d[u]=+∞ để lần sau không sử dụng u nữa
    return false;
}
```

Như vậy thuật toán luông cực đại sửa thành:

```
int MaxFlow() {
    int mf=0;                               // Khởi đầu luồng 0
    memset(Fx,0,sizeof(Fx));
    memset(Fy,0,sizeof(Fy));
    while (FindPath()) {                    // Nếu BFS đi đến được đỉnh 0
        for(int u=1;u<=n;u++)               // Thử qua các đỉnh Fx[u]=0 để tìm
            if (Fx[u]==0)                   // một đường tăng luồng từ u
                if (Ghep(u)) mf++;
    }
    return mf;
}
```

Dưới đây là toàn văn chương trình code bằng C++ 4.9.2 giải bài

http://vn.spoj.com/problems/FMATCH/:

```
#include <bits/stdc++.h>

#define MAXN 50005
#define INF 1000000007
#define tr(i,c) for(typeof((c).begin()) i=(c).begin();i!=(c).end();i++)

using namespace std;

int m,n,res=0;
vector<int> g[MAXN];
int x[MAXN],y[MAXN],d[MAXN], q[MAXN];

bool FindPath() {
    int L=1, R=0;
    for(int u=1;u<=n;u++)
    if (x[u]==0) {
        d[u]=0;
        q[++R]=u;
    } else d[u]=INF;
    d[0]=INF;
    while (L<=R) {
        int u=q[L++];
        tr(i,g[u]) {
            int v=*i;
            if (d[y[v]]==INF) {
                d[y[v]]=d[u]+1;
                if (y[v]) q[++R]=y[v];
            }
        }
    }
    return (d[0]!=INF);
}

bool dfs(int u) {
    if (u==0) return(true);
    tr(i,g[u]) {
        int v=*i;
        if (d[y[v]]==d[u]+1)
        if (dfs(y[v])) {
            x[u]=v; y[v]=u;
            return true;
        }
    }
    d[u]=INF;
    return false;
}

void GhepMax() {
    while (FindPath()) {
        for(int u=1;u<=n;u++)
            if (x[u]==0)
                if (dfs(u)) res++;
    }
    printf("%d",res);
}

int main() {
    int p;
    scanf("%d%d%d",&n,&m,&p);
    for(int i=1;i<=p;i++) {
        int u,v; scanf("%d%d",&u,&v);
        g[u].push_back(v);
```

```
    }
    GhepMax();
}
```

Thuật toán trên còn gọi là thuật toán **Hopkroft Karp** với cách đặt vấn đề có khác một chút nhưng về mặt bản chất là tương tự. Các thầy (cô) giáo có thể tham khảo theo đường link dưới đây:
https://en.wikipedia.org/wiki/Hopcroft%E2%80%93Karp_algorithm

Bài tập tương tự:
http://vn.spoj.com/problems/VOIEXAM/

II. Bài toán vận tải

Bài toán: Cho mạng $G(V, E, s, t)$ (s- đỉnh phát, t- đỉnh thu,) với các cung ngoài độ thông qua còn có thêm "chi phí" là một số nguyên không âm. Hãy tìm một luồng có giá trị bằng K với tổng chi phí nhỏ nhất. Ở đây tổng chi phí được tính bằng biểu thức:

$$\sum_{(u,v)\in E} f_{uv} \times a_{uv}$$

Trong đó f_{uv} là giá trị luồng trên cung (u,v) còn a_{uv} là "chi phí" vận chuyển một đơn vị luồng trên cung (u,v). Bài toán trên còn gọi là bài toán vận tải.
Để giải bài toán này chúng ta thêm một đỉnh phát giả S, nối S với đỉnh phát s bằng một cung có độ thông qua là K và chi phí bằng 0. thêm một đỉnh thu giả T, nối đỉnh thu t với T bằng một cung có độ thông qua K và chi phí 0. Bài toán qui về tìm luồng cực đại (có giá trị K) của mạng trên và tổng chi phí là bé nhất.
Có thể giải bài toán trên bằng thuật toán tìm luồng cực đại với một cải tiến: Thay vì sử dụng BFS để tìm đường ít cạnh nhất từ đỉnh phát đến đỉnh thu chúng ta tìm đường đi có "tổng chi phí" bé nhất từ đỉnh phát đến đỉnh thu với qui ước cung xuôi chi phí mang giá trị dương và cung ngược chi phí mang giá trị âm. Bài toán tìm đường tăng luồng qui về bài toán tìm đường đi ngắn nhất. Do các cung có thể có giá trị âm nên thuật toán được áp dụng ở đây là thuật toán Ford - Bellman.

Bài 1: Chọn vận động viên (ATHLETE)

Kết quả thi đấu quốc gia của n vận động viên (đánh số từ 1 đến n) trên m môn (đánh số từ 1 đến m) được đánh giá bằng điểm (giá trị nguyên không âm). Với mỗi vận động viên ta biết điểm đánh giá trên từng môn của vận động viên ấy.
Cần chọn ra k vận động viên và k môn ($1 \le k \le \min(m, n)$) để thành lập đội tuyển thi đấu Olimpic quốc tế, trong đó mỗi vận động viên chỉ được thi đấu đúng một môn sao cho tổng số điểm của các vận động viên trên các môn đã chọn là lớn nhất.
Dữ liệu: Vào từ file văn bản ATHLETE.INP:

- Dòng đầu ghi hai số n, m ($1 \le n, m \le 200$)
- n dòng tiếp theo, mỗi dòng ghi các điểm đánh giá trên tất cả các môn theo thứ tự môn thi 1, 2, ..., m. Các dòng này được ghi theo thứ tự vận động viên 1, 2, ..., n. Các số trên cùng dòng ghi cách nhau dấu cách. Điểm nằm trong phạm vi từ 0 đến 1000
- Dòng cuối cùng ghi số k

Kết quả: Ghi ra file văn bản ATHLETE.OUT:

- Dòng đầu ghi tổng số điểm của đội tuyển Olimpic
- Trong k dòng tiếp theo mỗi dòng ghi hai số u, v thể hiện vận động viên u sẽ được cử thi đấu môn v.

Example:

ATHLETE.INP	ATHLETE.OUT
3 3 1 5 0 5 7 4 3 6 3 2	11 2 1 3 2

Thuật toán:

Xây dựng một mạng :

- Đỉnh phát s nối đến n vận động viên bằng các cung có độ thông qua bằng 1,chi phí bằng 0.
- Vận động viên i nối với môn thi j bởi cung có độ thông qua $+\infty$ và chi phí bằng $-a[i,j]$ với $a[i,j]$ là điểm của vận động viên i thi đấu môn j
- Các môn học được nối với đỉnh thu t bằng cung có độ thông qua 1, chi phí 0

Bài toán qui về tìm luồng với giá trị k mà có tổng chi phí nhỏ nhất.

Dưới đây là code tham khảo C++:

```
#include <bits/stdc++.h>
#define maxn 500
#define INF 2000001

using namespace std;

int n, m, k;
int a[maxn][maxn], fl[maxn][maxn], c[maxn][maxn];
int cl[maxn], kc[maxn], prv[maxn];
deque<int> q;
int s, t;
int cp;

bool FindPath() {
    q.clear();
    for(int u=1;u<=n+m+2;++u) cl[u]=0, kc[u]=INF;
    q.push_back(s); cl[s]=1; kc[s]=0;
    while (!q.empty()) {
        int u=q.front(); q.pop_front(); cl[u]=0;
        // cung xuoi
        for(int v=1;v<=n+m+2;++v)
        if (fl[u][v]<c[u][v] && kc[v]>kc[u]+a[u][v]) {
            kc[v]=kc[u]+a[u][v]; prv[v]=u;
            if (cl[v]==0) q.push_back(v), cl[v]=1;
        }
        // Cung nguoc
        for(int v=1;v<=n+m+2;++v)
        if (fl[v][u]>0 && kc[v]>kc[u]-a[v][u]) {
            kc[v]=kc[u]-a[v][u]; prv[v]=-u;
            if (cl[v]==0) q.push_back(v), cl[v]=1;
        }
    }
    return (kc[t]!=INF);
}

void IncFlow() {
```

```
    int v=t;
    while (v!=s) {
        int u=prv[v];
        if (u>0) fl[u][v]=1, cp+=a[u][v]; else {
            u=-u;
            cp-=a[v][u];
            fl[v][u]=0;
        }
        v=u;
    }
}

int main() {
    freopen("athlete.inp","r",stdin);
    freopen("athlete.out","w",stdout);
    scanf("%d%d",&n,&m);
    s=n+m+1; t=n+m+2;
    memset(a,0,sizeof(a));
    memset(c,0,sizeof(c));
    for(int i=1;i<=n;++i)
        for(int j=1;j<=m;++j) {
            c[i][j+n]=1;
            scanf("%d",&a[i][j+n]);
            a[i][j+n]=-a[i][j+n];
    }
    for(int i=1;i<=n;++i) c[s][i]=1;
    for(int j=1;j<=m;++j) c[j+n][t]=1;
    scanf("%d",&k);
    memset(fl,0,sizeof(fl));
    cp=0;
    for(int i=1;i<=k;++i) {
        FindPath();
        IncFlow();
    }
    printf("%d\n",-cp);
    for(int u=1;u<=n;++u)
    for(int v=1;v<=m;++v) if (fl[u][v+n]) {
        printf("%d %d\n",u,v);
        break;
    }
}
```

Một nhận xét rằng bài toán trên là mở rộng của bài toán tìm cặp ghép đầy đủ đạt max. Thuật toán Min Cost cũng có thể dùng giải bài toán này.

Bài 2: K đường đi không chung cạnh (KWAY)

(Bài toán trên SPOJ: http://vn.spoj.com/problems/KWAY/)

Cho đồ thị vô hướng có trọng số không âm $G = (V, E)$ và hai đỉnh $s, t \in V$ hãy tìm k đường đi khác nhau không chung đỉnh (ngoại trừ đỉnh xuất phát s và đỉnh kết thúc t) từ s đến t sao cho tổng trọng số của k đường đi này là nhỏ nhất?.

Thuật toán:

Ta xây dựng mạng với đỉnh phát s, đỉnh thu t độ thông qua của tất cả các cung đều bằng 1 và chi phí chính là trọng số của các cung. Dễ nhận thấy một phương án chọn k đường đi hợp lệ sẽ tương đương với một luồng có giá trị k. Do độ thông qua của các cung đều bằng 1 nên đây chính là bài toán tìm luồng có giá trị k với tổng chi phí nhỏ nhất.

Tất nhiên thuật toán được sử dụng là luồng Min Cost.

Dưới đây là chương trình thực hiện với C++4.9.2:

```
#include <bits/stdc++.h>
#define maxn 101
#define maxm 10001
#define INF 2000000000
#define tr(i,c) for(typeof((c).begin()) i=(c).begin();i!=(c).end();i++)

using namespace std;
struct edge{
    int x, y;
    int c, f;
    int cost;
};

int n, m, k, xp, kt;
edge e[2*maxm];
vector<int> g[maxn];
deque<int> q;
int cl[maxn], kc[maxn], prev[maxn];

void InitFlow() {
    for(int i=1;i<=2*m;i++) e[i].f=0;
}

bool FindPath() {
    q.clear();
    for(int i=1;i<=n;i++) cl[i]=0;
    for(int i=1;i<=n;i++) kc[i]=INF;
    cl[xp]=1, kc[xp]=0, prev[xp]=0;
    q.push_back(xp);
    while (!q.empty()) {
        int u=q.front(); q.pop_front();
        cl[u]=0;
        tr(i,g[u]) {
            int id=*i;
            if (e[id].f<e[id].c) {
                int v=e[id].y;
                int L=(e[id].f>=0) ? e[id].cost : -e[id].cost;
                if (kc[v]>kc[u]+L) {
                    kc[v]=kc[u]+L, prev[v]=id;
                    if (cl[v]==0) cl[v]=1, q.push_back(v);
                }
            }
        }
    }
    return (kc[kt]<INF);
```

```
}

void IncFlow() {
    int v=kt;
    while (prev[v]!=0) {
        int id=prev[v];
        e[id].f++;
        e[2*m+1-id].f--;
        v=e[id].x;
    }
}

bool MaxFlow() {
    InitFlow();
    for(int i=1;i<=k;i++) {
        if (!FindPath()) return false;
        IncFlow();
    }
    return true;
}

int x[maxn], slx;

void write_out() {
    int ans=0;
    for(int i=1;i<=2*m;i++) if (e[i].f>=0) ans+=e[i].f*e[i].cost;
    cout << ans << endl;
    for(int i=1;i<=k;i++) {
        slx=0;
        int u, v=kt;
        while (v!=xp) {
            x[++slx]=v;
            tr(i,g[v]) {
                int id=*i;
                if (e[id].f<0) {
                    u=e[id].y;
                    e[id].f=0;
                    break;
                }
            }
            v=u;
        }
        x[++slx]=xp;
        cout << slx;
        for(int i=slx;i>=1;i--) cout <<" "<<x[i];
        cout << endl;
    }
}

int main() {
    #ifndef ONLINE_JUDGE
    freopen("inp.txt","r",stdin);
    freopen("out.txt","w",stdout);
    #endif // ONLINE_JUDGE
```

```
    ios::sync_with_stdio(false);
    cin >> n >> m >> k >> xp >> kt;
    for(int i=1;i<=m;i++) {
        int u, v, w;
        cin >> u >> v >> w;
        e[i].x=u, e[i].y=v, e[i].c=1, e[i].cost=w;
        e[2*m+1-i].x=v, e[2*m+1-i].y=u, e[2*m+1-i].c=1, e[2*m+1-i].cost=w;
    }
    for(int i=1;i<=2*m;i++) g[e[i].x].push_back(i);
    if (!MaxFlow()) cout <<"-1"; else write_out();
}
```

Khi $k = 2$ ta có bài toán tìm 2 đường đi từ s đến t không chung đỉnh có tổng trọng số bé nhất. Có thể thấy trong trường hợp này có 2 lần tăng luồng:

- Lần thứ nhất do trọng số các cạnh là không âm (không có cung ngược) nên có thể sử dụng thuật toán Dijkstra để tìm đường tăng luồng chi phí bé nhất.
- Lần thứ hai do có các cạnh trọng số âm nên thuật toán phải sử dụng tìm đường tăng luồng là Ford-Bellman.

Bài toán hai đường đi chính là bài: http://vn.spoj.com/problems/HIWAY/

III- Lát cắt hẹp nhất

Một trong những ứng dụng quan trọng nhất của thuật toán Ford-Fulkerson chính là định lý Fulkerson: ***Giá trị của luồng cực đại chính bằng giá trị của lát cắt hẹp nhất***. Một cách tự nhiên, việc sử dụng định lý này là một trong những ứng dụng phổ biến của luồng cực đại.

Nhắc lại: Cho $G = (V, E, s, t)$ là một mạng với đỉnh phát s, đỉnh thu t. Một lát cắt hẹp nhất là một phân hoạch $V = X \cup Y,\ X \cap Y = \phi$ sao cho $s \in X, t \in Y$. Khi đó giá trị của lát cắt được định nghĩa:

$$f(X,Y) = \sum_{(u,v)\in E, u\in X, v\in Y} c(u,v)$$

Để có thể áp dụng định lý trên, khi giải một bài toán tối ưu chúng ta thực hiện theo ba bước:

Bước 1: Phát biểu lại bài toán về dạng cực tiểu (bài toán tìm min)

Bước 2: Xây dựng mạng thích hợp sao cho giá trị cần tìm cực tiểu tương ứng với một lát cắt

Bước 3: Tìm luồng cực đại trên mạng này

Ngoài ra chú ý rằng để tìm lát cắt hẹp nhất thì trong lần tìm đường tăng luồng cuối cùng (không tìm được đường) thì tất cả những đỉnh thăm được đều thuộc tập X và những đỉnh không thăm được đều thuộc tập Y.

Để minh họa chúng ta xét bài tập sau:

Bài 3: Các cuộc thám hiểm không gian (NASA)

Giáo sư Spock đang cố vấn cho NASA, cơ quan này đang hoạch định một loạt các chuyến bay con thoi không gian và phải quyết định các cuộc thí nghiệm thương mại để thực hiện và các dụng cụ phải có trên tàu cho mỗi chuyến bay. Với mỗi chuyến bay, NASA xét một tập hợp các thí nghiệm có thể thực hiện, các thí nghiệm này đánh số từ 1 đến N, với thí nghiệm j, các nhà bảo trợ thương mại đã đồng ý thanh toán cho NASA p_j đôla (j=1,2,..,N). Các cuộc thí nghiệm sử dụng M dụng các dụng cụ, đánh số từ 1 đến M. Mỗi thí nhgiệm sẽ sử dụng một số trong số M dụng cụ này. Múc hao phí để mang dụng cụ thứ k vào không gian là c_k đôla (k=1,2,..,M). Công việc của giáo sư Spock là xác định các cuộc thí nghiệm để thực hiện và các dụng cụ để mang theo cho một chuyến bay đã cho sao cho tối đa

hoá lợi tức ròng, tức là tổng thu nhập từ các cuộc thí nghiệm được thực hiện trừ đi mức hao phí của tất cả các dụng cụ mang theo.

Yêu cầu: Viết chương trình giúp giáo sư Spock thực hiện yêu cầu trên.

Input: Vào từ file văn bản NASA.INP:

- Dòng đầu tiên chứa hai số nguyên N, M (1≤N,M≤100)
- Dòng thứ hai chứa N số nguyên $p_1, p_2, ..., p_n$
- Dòng thứ ba chứa M số nguyên $c_1, c_2, ..., c_m$
- Tiếp theo là N dòng, dòng thứ i chứa các số $a_{i1}, a_{i2}, ..., a_{im}$ với a_{ij} bằng 1nếu thí nghiệm i sử dụng dụng cụ j và bằng 0 nếu thí nghiệm i không sử dụng dụng cụ j.

Output: Ghi ra file văn bản NASA.OUT:

- Dòng đầu tiên ghi tổng lợi tức ròng thu được.
- Dòng 2 liệt kê danh sách các thí nghiệm cần thực hiện (nếu không có thí nghiệm nào thực hiện thì ghi số 0)
- Dòng 3 liệt kê danh sách các dụng cụ mà chuyến bay phải mang theo (nếu không có dụng cụ nào phải mang theo thì ghi 0)

Example:

Input	Output
3 4 4 10 11 6 2 3 7 1 0 0 1 0 1 1 0 0 1 0 0	16 2 3 2 3

Thuật toán:

Đặt X là tập các thí nghiệm, X_1 là tập các thí nghiệm được thực hiện. Đặt Y là tập các dụng cụ, Y_1 là tập các dụng cụ mang theo. Nhiệm vụ của chúng ta là cần phải cực đại giá trị:

$$T = \sum_{i \in X_1} p_i - \sum_{j \in Y_1} c_j$$

Bước 1: Trước tiên chúng ta đưa bài toán trên về bài toán cực tiểu. Ta có:

$$S = \sum_{i \in X} p_i - T = \sum_{i \in X} p_i - \left(\sum_{i \in X_1} p_i - \sum_{j \in Y_1} c_j \right) = \sum_{i \in X_2} p_i + \sum_{j \in Y_1} c_j$$

Ở đây $X_2 = X \backslash X_1$

Bởi vì $\sum_{i \in X} p_i$ là hằng số nên muốn T cực đại thì S cực tiểu và thay vì tìm T cực đại chúng ta tìm S cực tiểu. Một điều quan trọng cần chú ý là trong biểu thức của S chỉ có phép toán cộng.

Bước 2: Xây dựng mạng.

Ta xây dựng một mạng như sau:

- Đỉnh phát s nối với tập $x_1, x_2, \dots, x_n$ (mỗi đỉnh đại diện cho một thí nghiệm) bằng các cung có độ thông qua p_i $(i = 1,2, \dots, n)$.
- Tập $X = \{x_1, x_2, \dots, x_n\}$ nối đến tập $Y = \{y_1, y_2, \dots, y_m\}$ (đại diện cho các thí nghiệm) với cung (x_i, y_j) khi thí nghiệm i sử dụng dụng cụ j. Độ thông qua là $+\infty$
- Tập $Y = \{y_1, y_2, \dots, y_m\}$ nối đến đỉnh thu t bằng các cung có độ thông qua c_j $(j = 1,2, \dots, m)$

Như vậy tập đỉnh của mạng sẽ là $V = \{s\} \cup X \cup Y \cup \{t\}$.

Đặt $V = (V_1, V_2)$ là một lát cắt của V với

$$V_1 = \{s\} \cup X_a \cup Y_a; \quad V_2 = X_b \cup Y_b \cup \{t\}$$

Ở đây $X = X_a \cup X_b, Y = Y_a \cup Y_b, X_a \cap X_b = Y_a \cap Y_b = \phi$

Nếu (V_1, V_2) là một lát cắt hẹp nhất ta có một số nhận xét:

- Sẽ không tồn tại cung nối từ X_a đến Y_b bởi vì độ thông qua của tất cả các cung (x_i, y_j) được gán là $+\infty$ nên nếu có một cung như vậy thì đỉnh y_j phải được thăm (mẫu thuẫn với giả thiết là lát cắt hẹp nhất).
- Chỉ tồn tại hai loại cung nối từ V_1 đến V_2 là loại cung $(s, x_i): x_i \in X_b$ và $(y_j, t): y_j \in Y_a$
- Giá trị lát cắt (V_1, V_2) bằng $\sum_{i \in X_b} p_i + \sum_{j \in Y_a} c_j$

Do vậy giá trị cực tiểu của S đạt được khi chọn $X_2 \equiv X_b, Y_1 \equiv Y_a$. Dễ thấy điều kiện tất cả các dụng cụ đều được mang đi bởi vì không có cung đi từ $X_1 = X_a$ đến $Y_2 = Y_b$

Bài toán đơn giản là tìm luồng cực đại trên mạng vừa mô tả.

Dưới đây là code tham khảo bằng C++:

```
#include <bits/stdc++.h>
#define INF 2000000000
#define maxn 222

using namespace std;

int n, m, c[maxn][maxn];
int s,t,nn;
int f[maxn][maxn], mf;

int cl[maxn], q[maxn], prev[maxn];
void initflow() {
    for(int i=1;i<=nn;i++)
        for(int j=1;j<=nn;j++) f[i][j]=0;
    mf=0;
}
bool findpath() {
    for(int i=1;i<=nn;i++) cl[i]=0;
    int L=1, R=0;
    cl[s]=1; q[++R]=s; prev[s]=0;
    while (L<=R) {
        int u=q[L++];
        for(int v=1;v<=nn;v++) if (cl[v]==0 && f[u][v]<c[u][v]) {
            cl[v]=1; q[++R]=v; prev[v]=u;
            if (v==t) return true;
        }
        for(int v=1;v<=nn;v++) if (cl[v]==0 && f[v][u]>0) {
            cl[v]=1; q[++R]=v; prev[v]=-u;
        }
    }
    return false;
}
void incflow() {
    int v=nn, delta=INF;
    while (prev[v]!=0) {
        int u=prev[v];
        if (u>0) delta=min(delta,(c[u][v]==INF) ? INF : c[u][v]-f[u][v]);
```

```
        else u=-u, delta=min(delta,f[v][u]);
        v=u;
    }
    v=nn;
    while (prev[v]!=0) {
        int u=prev[v];
        if (u>0) f[u][v]+=delta;
        else u=-u, f[v][u]-=delta;
        v=u;
    }
    mf+=delta;
}

int main() {
    freopen("nasa.inp","r",stdin);
    freopen("nasa.out","w",stdout);
    scanf("%d%d",&n,&m);
    s=n+m+1, t=n+m+2;
    nn=n+m+2;
    for(int i=1;i<=nn;i++)
        for(int j=1;j<=nn;j++) c[i][j]=0;
    int ans=0;
    for(int i=1;i<=n;i++) {
        int u; scanf("%d",&u);
        c[s][i]=u;
        ans+=u;
    }

    for(int i=1;i<=m;i++) {
        int u; scanf("%d",&u);
        c[i+n][t]=u;
    }
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++) {
        int u; scanf("%d",&u);
        c[i][j+n]=(u) ? INF : 0;
    }
    initflow();
    while (findpath()) incflow();
    ans-=mf;
    printf("%d\n",ans);
    int sl=0;
    for(int i=1;i<=n;i++) if (cl[i]) printf("%d ",i),++sl;
    if (sl==0) printf("0\n"); else printf("\n");
    sl=0;
    for(int i=1;i<=m;i++) if (cl[i+n]) printf("%d ",i),++sl;
    if (sl==0) printf("0\n"); else printf("\n");
}
```

Bài 4. Phủ sóng đô thị (COVERAGE)

Tại một thành phố có N hộ dân cư và 2 nhà cung cấp dịch vụ viễn thông. Với mỗi hộ dân cư i, nếu họ chọn nhà cung cấp thứ nhất sẽ mất chi phí là a_i, nếu họ chọn nhà cung cấp thứ hai sẽ mất chi phí

là b_i. Ngoài ra, nếu 2 hộ dân cư i và j lắp dịch vụ của 2 nhà cung cấp khác nhau, họ phải tốn thêm một chi phí là c_{ij} cho việc lắp đặt hệ thống chống nhiễu sóng.
Yêu cầu: Hãy tính chi phí nhỏ nhất để phủ sóng toàn bộ N hộ dân cư.
Input:

- Dòng đầu ghi số N. ($1 \le N \le 200$)
- Dòng thứ 2 ghi N số a_i. ($1 \le a_i \le 1000$)
- Dòng thứ 3 ghi N số b_i. ($1 \le b_i \le 1000$)
- N dòng cuối thể hiện c_{ij}. ($0 \le c_{ij} = c_{ji} \le 100$)

Output: Một số duy nhất thể hiện chi phí nhỏ nhất.
Example:

Input	Output
3 1 1 10 10 10 1 0 0 1 0 0 1 1 1 0	5

Thuật toán:
Đặt X là tập các hộ dân cư sử dụng nhà cung cấp thứ nhất và Y là tập các hộ dân cư sử dụng nhà cung cấp thứ hai. Bài toán yêu cầu tìm cực tiểu của biểu thức:

$$S = \sum_{i \in X} a_i + \sum_{j \in Y} b_j + \sum_{i \in X, j \in Y} c_{ij}$$

Trong trường hợp này ta chỉ việc xây dựng một mạng phù hợp (không cần phát biểu lại bài toan). Đặt V là tập hợp toàn bộ các hộ dân cư. Xây dựng mạng $G = (V, E, s, t)$ với đỉnh phát s và đỉnh thu t cùng với các cung:

- Cung $(s, i): i \in V$ với độ thông qua là a_i
- Cung $(j, t): j \in V$ với độ thông qua là b_j
- Cung $(i, j): i, j \in V$ với độ thông qua là c_{ij}

Gọi (V_1, V_2) là một lát cắt. Ta có $V_1 = \{s\} \cup X_1; V_2 = Y_1 \cup \{t\}$ ở đây $X_1 \cup Y_1 = V, X_1 \cap Y_1 = \phi$. Các cạnh nối từ V_1 đến V_2 được chia thành ba loại:

- $(s, i): i \in Y_1$ và tổng độ thông qua các cạnh này bằng $\sum_{i \in Y_1} a_i$
- $(j, t): j \in X_1$và tổng độ thông qua của các cạnh này bằng $\sum_{j \in X_1} b_j$
- $(i, j): i \in X_1, j \in Y_1$ và tổng độ thông qua của các cạnh này bằng $\sum_{i \in X_1, j \in Y_1} c_{ij}$

Do vậy giá trị lát cắt sẽ là:

$$S_1 = \sum_{i \in Y_1} a_i + \sum_{j \in X_1} b_j + \sum_{i \in Y_1, j \in X_1} c_{ij}$$

Và để phù hợp với S ta chỉ cần chọn Y_1 là tập các hộ sử dụng nhà cung cấp 1 và X_1 là tập các hộ sử dụng nhà cung cấp 2.
Bài toán qui về bài toán tìm lát cắt cực tiểu và phương pháp là tìm luồng cực đại:

Chương trình minh họa bằng C++:

```
#include <bits/stdc++.h>
#define maxn 222
#define INF 2000000000

using namespace std;
```

```
int n, c[maxn][maxn];
int f[maxn][maxn], mf;

void initflow() {
    for(int i=1;i<=n+2;i++)
        for(int j=1;j<=n+2;j++) f[i][j]=0;
    mf=0;
}

int cl[maxn], q[maxn], prev[maxn];
bool findpath() {
    for(int i=1;i<=n+2;i++) cl[i]=0;
    int L=1, R=0;
    cl[n+1]=1; q[++R]=n+1;
    while (L<=R) {
        int u=q[L++];
        // Di xuoi
        for(int v=1;v<=n+2;v++) if (cl[v]==0 && f[u][v]<c[u][v]) {
            cl[v]=1; q[++R]=v; prev[v]=u;
            if (v==n+2) return true;
        }
        // Di nguoc
        for(int v=1;v<=n+2;v++) if (cl[v]==0 && f[v][u]>0) {
            cl[v]=1; q[++R]=v; prev[v]=-u;
        }
    }
    return false;
}
void incflow() {
    int v=n+2, delta=INF;
    while (prev[v]!=0) {
        int u=prev[v];
        if (u>0) delta=min(delta,c[u][v]-f[u][v]);
        else u=-u,delta=min(delta,f[v][u]);
        v=u;
    }
    v=n+2;
    while (prev[v]!=0) {
        int u=prev[v];
        if (u>0) f[u][v]+=delta;
        else u=-u,f[v][u]-=delta;
        v=u;
    }
    mf+=delta;
}

int main() {
    freopen("coverage.inp","r",stdin);
    freopen("coverage.out","w",stdout);
    scanf("%d",&n);
    for(int i=1;i<=n;i++) {
        int u; scanf("%d",&u);
        c[n+1][i]=u;
```

```
        c[i][n+1]=0;
    }
    for(int i=1;i<=n;i++) {
        int u; scanf("%d",&u);
        c[i][n+2]=u;
        c[n+2][i]=0;
    }
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++) scanf("%d",&c[i][j]);
    initflow();
    while (findpath()) incflow();
    printf("%d",mf);
}
```

Bài tập tương tự: http://vn.spoj.com/problems/LIGHT/

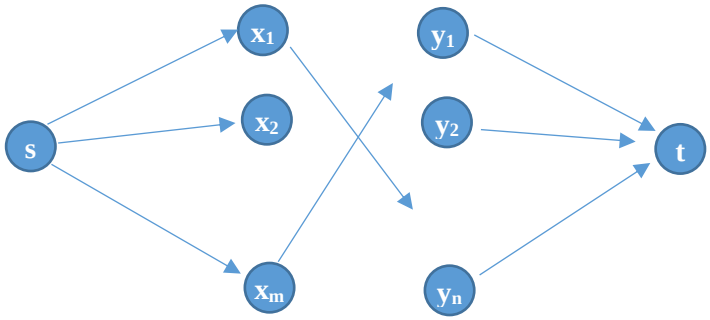
IV. Phủ đỉnh trên đồ thị hai phía

Bài toán: Cho đồ thị hai phía $G = (X \cup Y, E)$. Hãy chọn một tập đỉnh $C \subset X \cup Y$ có lực lượng bé nhất sao cho:

$$\forall\ (u,v) \in E \text{ thì hoặc } u \in C, \text{ hoặc } v \in C$$

Tập C được gọi là tập phủ đỉnh. Bài toán tìm tập phủ đỉnh trên đồ thị tổng quát hiện tại chưa có thuật toán trong thời gian đa thức để giải quyết. Tuy nhiên khi đồ thị là hai phía, phủ đỉnh có thể được xây dựng bằng luồng cực đại hoặc cặp ghép cực đại.

Xây dựng mô hình mạng như hình vẽ dưới đây:



Đỉnh phát s được nối với tất cả các đỉnh $x_i \in X$ (bên trái) cung với độ thông qua là 1. Tất cả các đỉnh $y_j \in Y$ (bên phải) nối với đỉnh thu t cung với độ thông qua 1. Các cạnh $(x_i, y_j) \in E$ sẽ là cung nối đỉnh $x_i \in X$ với $y_j \in Y$ (hai chiều) với độ thông qua $+\infty$.

Gọi (S,T) là lát cắt hẹp nhất của mạng. Đặt $A = \{x \in T\}, B = \{y \in S\}$ ta có $A \subset X$ là các đỉnh chưa thăm trong lần tìm đường cuối cùng còn $B \subset Y$ là các đỉnh đã thăm trong lần tìm đường cuối cùng. Vì $X - A$ đã thăm và $Y - B$ chưa thăm nên không có cạnh $(x,y) \in E$ với $x \in X - A, y \in Y - B$ do vậy $C = A \cup B$ là một phủ đỉnh.

Giả sử C là một phủ đỉnh, $A = C \cap X, B = C \cap Y$. Khi đó hiển nhiên không có cạnh nối giữa $X - A$ và $Y - B$. Đặt $S = \{s\} \cup (X - A) \cup B,\ \ T = A \cup (Y - B) \cup \{t\}$ có thể thấy (S,T) là một lát cắt có giá trị đúng bằng số đỉnh của phủ đỉnh.

Vậy bài toán tìm C có lực lượng nhỏ nhất qui về bài toán tìm luồng cực đại trên mạng đã dựng

Luồng trên mạng này chính là cặp ghép cực đại trên đồ thị hai phía $G = (X \cup Y, E)$ và những đỉnh thuộc C gồm:

$B = \{v \in Y : \exists\ u \in X, x[u] = 0, (u,v) \in E\}$ - Những đỉnh đã thăm bên Y

$A = \{u \in X : x[u] \neq 0, x[u] \notin B\}$ - Những đỉnh chưa thăm bên X

Bài 5. Chiếu sáng (LIGHTING)

Bản đồ công viên DisneyLand là một hình chữ nhật kích thước $m \times n$ được chia thành các lưới ô vuông đơn vị m hàng, n cột. Các hàng của lưới được đánh số từ 1 đến m, các cột của lưới được đánh số từ 1 đến n. Ô nằm ở hàng i, cột j được gọi là (i, j). Người ta đặt k khu vui chơi vào một số ô của lưới, mỗi khu vui chơi nằm hoàn toàn trong một ô và có thể có nhiều khu vui chơi nằm trong cùng một ô.

Hệ thống chiếu sáng của công viên gồm m đèn loại A: $a_1, a_2, \dots, a_m$ và n đèn loại B: $b_1, b_2, \dots, b_n$. Đèn a_i có thể chiếu sáng tất cả các ô trên hàng i và đèn b_j có thể chiếu sáng tất cả các ô nằm trên cột j. $(1 \le i \le m; 1 \le j \le n)$

Yêu cầu: Hãy bật sáng một số ít nhất các đèn để chiếu sáng toàn bộ các ô có khu vui chơi.

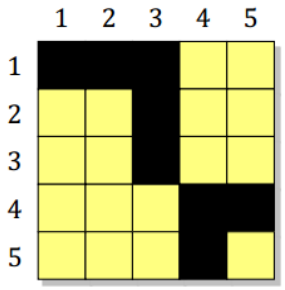
Input:

- Dòng 1 chứa ba số nguyên dương $m, n, k \le 10^5$
- k dòng tiếp theo, dòng thứ i ghi chỉ số hàng và chỉ số cột của khu vui chơi thứ i.

Output: Một số nguyên duy nhất là tổng lượng đèn ít nhất cần bật.

Example:

input	output
5 5 8 1 1 1 2 1 3 2 3 3 3 4 4 4 5 5 4	4



Thuật toán:

Xây dựng đồ thị hai phía với tập các đỉnh bên trái là các hàng $X = \{1,2, \dots, m\}$ và tập các đỉnh bên phải là các cột $Y = \{1,2, \dots, n\}$. Đỉnh $i \in X$ có cạnh nối tới đỉnh $j \in Y$ khi ô (i, j) là khu vui chơi.

Bài toán chính là bài toán tìm phủ đỉnh tối thiểu trên đồ thị hai phía vừa lập. Thuật toán được áp dụng là luồng cực đại và trong trường hợp này là cặp ghép cực đại.

Để chương trình đạt tốc độ cần thiết ta cần code dạng cải tiến của luồng khi tìm cặp ghép cực đại (thuật toán Hopkroft Karp).

Dưới đây là code minh họa bằng C++:

```
#include <bits/stdc++.h>
#define maxn 100005
#define oo 2000000001
using namespace std;
    int m,n,k,deg[maxn],mf;
    vector<int> g[maxn];
    int x[maxn],y[maxn],d[maxn],q[maxn];
bool findp()
{
    for(int u=0; u<=m; u++) d[u]=oo;
    int L=1,R=0;
    for(int u=1; u<=m; u++)
    if(x[u]==0)
    {
        d[u]=0;
```

```
            q[++R]=u;
        }
        while(L<=R)
        {
            int u=q[L++];
            for(int i=0; i<deg[u]; i++)
            {
                int v=g[u][i];
                int w=y[v];
                if(d[w]==oo)
                {
                    d[w]=d[u]+1;
                    q[++R]=w;
                }
            }
        }
        return (d[0]!=oo);
}
bool incf(int u)
{
    if(u==0) return 1;
    for(int i=0; i<deg[u]; i++)
    {
        int v=g[u][i];
        int w=y[v];
        if(d[w]==d[u]+1)
        if(incf(w))
        {
            x[u]=v; y[v]=u;
            return 1;
        }
    }
    d[u]=oo;
    return 0;
}
int main()
{
    freopen("lighting.inp","r",stdin);
    freopen("lighting.out","w",stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>m>>n>>k;
    while(k--)
    {
        int u,v;
        cin>>u>>v;
        deg[u]++; g[u].push_back(v);
    }
    while(findp())
    {
        for(int u=1; u<=m; u++)
        if(x[u]==0)
            if(incf(u))
```

```
            mf++;
    }
    cout<<mf;
}
```

V. Phủ đường tối thiểu trên DAG

Cho $G = (V, E)$ là đồ thị có hướng không có chu trình (DAG). Một *phủ đường* là tập hợp P các đường đi trên G thỏa mãn: với mọi đỉnh $v \in V$ tồn tại duy nhất một đường đi trong P chứa v. Đường đi có thể bắt đầu và kết thúc ở bất cứ đâu. Tính cả đường đi có độ dài 0. Bài toán đặt ra là tìm *phủ đường tối thiểu*: phủ đường gồm ít đường đi nhất.

Gọi n là số đỉnh của đồ thị. Đánh số các đỉnh của V từ 1 đến n. Xây dựng đồ thị hai phía $G' = (X \cup Y, E')$ trong đó:

$$X = \{x_1, x_2, \dots, x_n\}$$
$$Y = \{y_1, y_2, \dots, y_n\}$$

Tập E' được dựng như sau: Với mỗi cung $(i, j) \in E$, thêm vào cạnh (x_i, y_j) trên E'.

Gọi M là bộ ghép trên G'. Khởi tạo P là tập n đường đi, mỗi đường đi chỉ gồm một đỉnh trong G. Khi đó P là phủ đường. Mỗi khi xét tới cạnh $(x_i, y_j) \in M$ thì (i, j) luôn là một đầu kết thúc và một đầu xuất phát của hai đường trong phủ đường P. Do vậy ta có thể nhập hai đường này làm một và $|P|$ giảm đi 1. Do đó khi thuật toán kết thúc $|P| = n - |M|$ do đó muốn $|P|$ cực tiểu thì $|M|$ cực đại

Dưới đây là thuật toán viết bằng C++:

```
#include <bits/stdc++.h>
#define maxn 50005
#define INF 1000000000
#define tr(i,c) for(typeof((c).begin()) i=(c).begin();i!=(c).end();i++)

using namespace std;

int n, m;
vector<int> g[maxn];

void read_in() {
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++) {
        int u, v; scanf("%d%d",&u,&v);
        g[u].push_back(v);
    }
}

int x[maxn], y[maxn], res;
int d[maxn], q[maxn];
bool FindPath() {
    int L=1,R=0;
    for(int i=1;i<=n;i++) if (x[i]==0) {
        d[i]=0; q[++R]=i;
    } else d[i]=INF;
    d[0]=INF;
    while (L<=R) {
        int u=q[L++];
        tr(i,g[u]) {
            int v=*i;
            if (d[y[v]]>d[u]+1) {
                d[y[v]]=d[u]+1;
                if (y[v]) q[++R]=y[v];
            }
        }
```

```
    }
    return (d[0]!=INF);
}
bool dfs(int u) {
    tr(i,g[u]) {
        int v=*i;
        if (y[v]==0) {
            x[u]=v, y[v]=u;
            d[u]=INF;
            return true;
        } else if (d[y[v]]==d[u]+1)
        if (dfs(y[v])) {
            x[u]=v, y[v]=u;
            d[u]=INF;
            return true;
        }
    }
    return false;
}
void MaxMatch() {
    for(int i=1;i<=n;i++) x[i]=y[i]=0;
    res=0;
    while (FindPath())
        for(int i=1;i<=n;i++) if (x[i]==0)
        if (dfs(i)) res++;
}
int sol, kq[maxn];
void write_out() {
    printf("%d\n",n-res);
    for(int i=1;i<=n;i++) if (x[i]==0) {
        int sol=0;
        int v=i;
        while (v) {kq[++sol]=v; v=y[v];}
        for(int i=sol;i>=1;i--) printf("%d ",kq[i]);
        printf("\n");
    }
}

int main() {
    freopen("inp.txt","r",stdin);
    freopen("out.txt","w",stdout);
    read_in();
    MaxMatch();
    write_out();
}
```

Xét bài tập dưới đây:

Bài 6. Điều tra (INQUIR.*)

Hãng zZz có hệ thống N siêu thị, đánh số 1,2,…,N. Gần đây hãng phát hiện K vụ trộm đồ ở các siêu thị. Hãng nghi vấn các vụ trộm này có cùng hung thủ nên muốn nhờ bạn dựa vào thông tin địa điểm và thời gian xảy ra các vụ trộm, thời gian di chuyển trên các đường nối trực tiếp giữa các siêu thị, xác định xem có ít nhất bao nhiêu hung thủ mới có thể thực hiện tất cả các vụ trộm.

Input: Vào từ file văn bản INQUIR.INP

- Dòng 1: số nguyên T ($1 \le T \le 100$) là số tests
- Mỗi test được cho trên một nhóm dòng, bao gồm:
- Dòng 1: ba số nguyên N, K, M ($1 \le N, K \le 500; 1 \le M \le 10^5$) tương ứng là số siêu thị, số vụ trộm, số đường nối trực tiếp các cặp siêu thị

- Dòng 2…K+1: mỗi dòng ghi thông tin của một vụ trộm, gồm hai số nguyên t, v ($t \leq 10^6$) chỉ vụ trộm xảy ra ở thời điểm t tại siêu thị v
- Dòng K+2…K+M+1: mỗi dòng ba số nguyên a, b, c ($1 \leq c \leq 10^6$) chỉ một đường hai chiều nối trực tiếp hai siêu thị a, b tốn thời gian di chuyển c.

Output: Ghi ra file văn bản INQUIR.OUT gồm T dòng, dòng i ghi số nguyên kết quả của test thứ i.

Example:

Input	Output
2 2 3 1 10 1 18 2 20 1 1 2 5 2 3 2 10 1 18 2 20 1 1 2 5 1 2 1	2 1

Thuật toán:

Gọi các vụ trộm liệt kê theo thứ tự tăng dần của thời điểm mất trộm là 1, 2, ..., K. Các thời điểm mất trộm lần lượt là $t_1 \leq t_2 \leq \cdots \leq t_K$. Ta có thể lập ma trận khoảng cách $d[i, j]$ giữa địa điểm xảy ra vụ trộm i và địa điểm xảy ra vụ trộm j bằng cách dử dụng thuật toán Floy-Bellman hoặc sử dụng K lần thuật toán Dijkstra.

Dựng đồ thị với tập đỉnh là K địa điểm mất trộm. Đỉnh i kề với đỉnh j ($i < j$) khi vụ trộm i và vụ trộm j có khả năng do cùng một toán trộm. Điều này xảy ra khi $d[i, j] \leq t_j - t_i$.

Nhận xét rằng đồ thị vừa dựng là DAG và bài toán qui về bài toán tìm phủ đường tối thiểu trên DAG

Chương trình minh họa viết bằng C++:

```
#include <bits/stdc++.h>
#define maxn 505
#define fs first
#define sc second
#define oo 2000000001
using namespace std;
typedef pair<int,int> II;
    int T,n,k,m,deg[maxn],mf;
    vector<II> g[maxn];
    II a[maxn];
    int cl[maxn][maxn],kc[maxn][maxn];
    set<II> q;

void dj(int xp)
{
    cl[xp][xp]=1;
    kc[xp][xp]=0;
    q.insert(II(0,xp));
    while(!q.empty())
    {
        II t=*q.begin();
        q.erase(t);
        int u=t.second;
        for(int i=0; i<deg[u]; i++)
        {
```

```
            int v=g[u][i].fs,l=g[u][i].sc;
            if(cl[xp][v]==0)
            {
                cl[xp][v]=1;
                kc[xp][v]=kc[xp][u]+l;
                q.insert(II(kc[xp][v],v));
            } else
            if(kc[xp][v]>kc[xp][u]+l)
            {
                q.erase(II(kc[xp][v],v));
                kc[xp][v]=kc[xp][u]+l;
                q.insert(II(kc[xp][v],v));
            }
        }
    }
}
    int deg1[maxn];
    vector<int> g1[maxn];
    int d[maxn],q1[maxn];
    int x[maxn],y[maxn];
bool findp()
{
    for(int u=0; u<=k; u++) d[u]=oo;
    int L=1,R=0;
    for(int u=1; u<=k; u++)
    if(x[u]==0)
    {
        d[u]=0;
        q1[++R]=u;
    }
    while(L<=R)
    {
        int u=q1[L++];
        for(int i=0; i<deg1[u]; i++)
        {
            int v=g1[u][i];
            int w=y[v];
            if(d[w]==oo)
            {
                d[w]=d[u]+1;
                q1[++R]=w;
            }
        }
    }
    return (d[0]!=oo);
}
bool dfs(int u)
{
    if(u==0) return 1;
    for(int i=0; i<deg1[u]; i++)
    {
        int v=g1[u][i];
        int w=y[v];
        if(d[w]==d[u]+1)
```

```
        if(dfs(w))
        {
            x[u]=v; y[v]=u;
            return 1;
        }
    }
    d[u]=oo;
    return 0;
}
void khoitao()
{
    memset(cl,0,sizeof(cl));
    memset(kc,0,sizeof(kc));
    for(int i=1; i<=n; i++)
    {
        deg[i]=0;
        g[i].clear();
    }
    for(int i=1; i<=k; i++)
    {
        deg1[i]=0;
        x[i]=0;
        y[i]=0;
        g1[i].clear();
    }
    mf=0;
}
void lam()
{
    khoitao();
    cin>>n>>k>>m;
    for(int i=1; i<=k; i++)
        cin>>a[i].fs>>a[i].sc;

    for(int i=1; i<=m; i++)
    {
        int u,v,c;
        cin>>u>>v>>c;
        deg[u]++; g[u].push_back(II(v,c));
        deg[v]++; g[v].push_back(II(u,c));
    }
    for(int i=1; i<=n; i++)
        dj(i);

    sort(a+1,a+k+1);

    for(int i=1; i<=k; i++)
    for(int j=i+1; j<=k; j++)
    if(cl[a[i].sc][a[j].sc]&&kc[a[i].sc][a[j].sc]<a[j].fs-a[i].fs)
    {
        deg1[i]++;
        g1[i].push_back(j);
    }
```

```
    while(findp())
    {
        for(int u=1; u<=k; u++)
        if(x[u]==0)
            if(dfs(u)) mf++;
    }
    cout<<k-mf<<"\n";
}
int main()
{
    freopen("inquir.inp","r",stdin);
    freopen("inquir.out","w",stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>T;
    while(T--)
        lam();
}
```

VI-Một số bài tập áp dụng

http://vn.spoj.com/problems/NKTRAFIC/
http://vn.spoj.com/problems/NKNET/
http://vn.spoj.com/problems/PBCWAYS/
http://vn.spoj.com/problems/JOBSET/
http://vn.spoj.com/problems/BAOVE/
http://vn.spoj.com/problems/MATCH2/
http://vn.spoj.com/problems/DIVREL/

KẾT LUẬN

Các thuật toán về luồng cực đại, đặc biệt thuật toán Ford - Fulkerson là các thuật toán đẹp và đơn giản. Tuy nhiên việc nhận ra một bài toán có thể dùng luồng cực đại để giải quyết là một trong những vấn đề khó trong các kỳ thi.

Chuyên đề này không tổng kết được hết các dạng khác nhau của việc áp dụng luồng cực đại (vì điều đó là không thể) mà chri dừng lại ở việc phân loại một số dạng toán cơ bản của việc sử dụng luồng cùng với các bài toán minh họa cho sự phân loại đó. Các bài tập được sưu tầm qua các bài giảng của thầy Lê Minh Hoàng (ĐHSP Hà Nội), thầy Nguyễn Mạnh Hà (Chuyên Vĩnh Phúc) cũng như từ một số kỳ thi Quốc gia và Quốc tế. Nó không phải là những bài tập khó. Mục đích chính để tôi lựa chọn là vì phương pháp giải nó mang tính chất điển hình có thể áp dụng cho nhiều bài toán khác.

Do thời gian và khuôn khổ, chuyên đề chắc chắn sẽ còn có một số nhậm lẫn và thiếu sót. Kính mong các thầy (cô) chỉ giáo!. Trân trọng!