网络流24题

Good Morning，这里是世界第一可爱的恶魔妹妹！

网络流24题里第24题是一道假题，所以这里只有23道。已经按照我认为合适的顺序排列过了，可以直接按照这个顺序刷题。

24题是谁总结出来的已经不太好考察了，但是24题基本上覆盖了网络流大多数的问题模型。刷完后可以掌握最基本的模型转化思维。

默认读者能够使用网络流的板子。在最后附上了恶魔妹妹自己使用的网络流板子。推荐最大流使用Dinic，费用流使用ZKW费用流。

24题里有很多题不只是网络流，也可以使用其他做法。还有一些题是最短路，如果你感觉能用最短路做那不用怀疑，就是。

RQY曾经告诉过我网络流本质是贪心，所以很多网络流问题最后都可以进行转化，转变为一些其他问题。如贪心，最短路，dp，模拟费用流等等。这里我们只考虑最基本的如何利用网络流来解题。

网络流的很多题解令人恼火，大量的博客互相抄袭，他们只告诉你怎么建图，却不告诉你为什么。我会尽量在我的题解里包含为什么这样建图。另外我认为如果按照我排列的顺序来作题的话已经积累足够的知识和经验，一些十分显然的地方不会长篇大论的讲解。如果你看不懂，你应该翻到前面确保自己理解了之前的内容。讲道理24题很多水题应该被删掉的，做水题并不是什么特别高兴的事情。

建议使用板子的时候计算好点数和边数，最好精确到个位数。具体可以看看我的代码实现。

最后，我是在LOJ做的题，很多题luogu上格式不太一样。

菜菜园子交流群：730602769；恶魔妹妹企鹅号：730602769。

~By ProtectEMmm，2022年11月16日。

最长递增子序列

题面

给定正整数序列 $x_1 \sim x_n$，以下递增子序列均为非严格递增。

1. 计算其最长递增子序列的长度 s。
2. 计算从给定的序列中最多可取出多少个长度为 s 的递增子序列。
3. 如果允许在取出的序列中多次使用 x_1 和 x_n，则从给定序列中最多可取出多少个长度为 s 的递增子序列。

题解

第一问：可以dp求解。

第二问：因为问的是取出，所以每个元素只能被取一次。考虑拆点，一个点分为 v, v' 并在他们之间连一条容量为 1 的边来限制住只能取一次这个条件。问的是最多的条数。因为跑完dp后是一张DAG，所以直接源点和 $dp_x = 1$ 的点连，汇点和 $dp_x = S$ 的点连。对于所有 $v < u, dp_u = dp_v + 1$ 的点连边。容量都是 1。然后求最大流就可以了。

第三问：因为 x_1, x_n 可以取出多次，所以 $S \to X_1 \to X_1', X_n \to X_n' \to T$ 这四条边容量都设为 INF。表示可以取用多次。

注意特判 $n = 1$ 的情况。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 5e2 + 10;
/*====================*/
int n, a[N];
/*====================*/
void Read(void)
{
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        cin >> a[i];
    }
}
/*====================*/
namespace _DP
{
    int dp[N], S = 0;
    vector<int> vec;
    void Init(void)
    {
        vec.clear();
        dp[0] = 0;
        for (int i = 1; i <= n; i++)
        {
            dp[i] = 1;
            for (int j = 1; j < i; j++)
            {
                if (a[j] <= a[i])
                {
                    dp[i] = max(dp[i], dp[j] + 1);
                }
            }
            S = max(S, dp[i]);
        }
    }
}
using namespace _DP;
/*====================*/
namespace _Dinic
{
    const int N = 1002 + 10;
    const int M = N * N + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
```

```
    int u, v, c, f;
    Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
    {
        u = _u, v = _v, c = _c, f = _f;
    }
};
/*====================*/
int cur[N];//当前弧优化
int n, m, s, t;//点，边，源，汇
vector<int>G[N];//邻接表
int d[N], vis[N];//图分层
Edge edge[2 * M]; int cnt;//边
/*====================*/
void AddEdge(int u, int v, int c)
{
    edge[cnt++] = Edge(u, v, c, 0);
    edge[cnt++] = Edge(v, u, 0, 0);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
```

```
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    /*====================*/
    int Init1(void)
    {
        cnt = 0;
        n = 2 * ::n + 2;
        s = 2 * ::n + 1, t = 2 * ::n + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        //拆点
        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(i, i + ::n, 1);
        }
        //连接源汇
        for (int i = 1; i <= ::n; ++i)
        {
            if (dp[i] == 1)
            {
                AddEdge(s, i, 1);
            }
            if (dp[i] == S)
            {
                AddEdge(i + ::n, t, 1);
            }
        }
        //与能匹配子序列的点连接
        for (int i = 1; i <= ::n; ++i)
        {
            for (int j = i + 1; j <= ::n; ++j)
            {
                if (a[i] <= a[j] && dp[j] == dp[i] + 1)
                {
                    AddEdge(i + ::n, j, 1);
                }
            }
        }
        int temp = MaxFlow();
```

```
        return temp != INF ? temp : 1;
    }
    int Init2(void)
    {
        for (int i = 0; i < cnt; i += 2)
        {
            Edge& e1 = edge[i]; e1.f = 0;
            Edge& e2 = edge[i ^ 1]; e2.f = 0;
            if (e1.u == s && e1.v == 1)e1.c = INF;
            if (e1.u == 1 && e1.v == 1 + ::n)e1.c = INF;
            if (e1.u == ::n + ::n && e1.v == t)e1.c = INF;
            if (e1.u == ::n && e1.v == ::n + ::n)e1.c = INF;
        }
        int temp = MaxFlow();
        return temp != INF ? temp : 1;
    }
}
using namespace _Dinic;
/*====================*/
int main()
{
    Read();
    _DP::Init();
    cout << _DP::S << endl;
    cout << _Dinic::Init1() << endl;
    cout << _Dinic::Init2() << endl;
    return 0;
}
```

航空路线

题面

给定一张航空图，图中顶点代表城市，边代表两个城市间的直通航线。现要求找出一条满足下述限制条件的且途经城市最多的旅行路线。

1. 从最西端城市出发，单向从西向东途经若干城市到达最东端城市，然后再单向从东向西飞回起点（可途经若干城市）。
2. 除起点城市外，任何城市只能访问一次。

对于给定的航空图，试设计一个算法找出一条满足要求的最佳航空旅行路线。

题解

首先先不考虑途经城市最多这条限制。

然后再不考虑每个城市只能经过一次这条限制。

要求从最西端的城市到最东端的城市再到最西端的城市，化成了一个圈。想象一个圈，固定两个端点，其实就是两点之间有两条路径。

从起点流到终点，流量最后如果为 2 就是有两条路径，否则就是没有。这里记得强行改变成有向边，只能从西边的城市连向东边的城市。

然后考虑每个城市只能经过一次。可以把每个点拆开成两个点，边的容量设为 1，来保证每个点只被流一次。其中注意起点和终点的容量是 2。可以想象从起点有两架飞机，每一架就是一个流，然后流到终点。那么起点有两架飞机流出，终点有两架飞机流入。

最后考虑途经城市最多。我们可以给每一个城市定一个权值，经过了权值加一。这样就是最大权值最大流。把权值设成相反数就变成了最小费用最大流。起点和终点的容量为 2 不影响最后的费用，因为无论如何起点和终点也要包含在答案之中。

对了这题好像可以 $O(n^3)$ 的dp做。

代码

```
#include<map>
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e2 + 10;
const int M = 5e3 + 10;
/*====================*/
int n, m;
string S, E;
map<string, int>f;
map<int, string>g;
struct Edge
{
    int u, v;
    Edge(int _u = 0, int _v = 0)
    {
        u = _u, v = _v;
        if (u > v)swap(u, v);
    }
}edge[M];
/*====================*/
void Read(void)
{
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
    {
        string str; cin >> str;
        f[str] = i; g[i] = str;
        if (i == 1) S = str;
        if (i == n) E = str;
    }
    for (int i = 1; i <= m; ++i)
    {
        string u, v; cin >> u >> v;
        edge[i] = Edge(f[u], f[v]);
    }
}
/*====================*/
namespace _ZKW
{
    const int N = 200 + 10;
```

```
const int M = 5200 + 10;
/*====================*/
const int INF = 0X7FFFFFFF;
/*====================*/
struct Edge
{
    int u, v, c, w;
    Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
    {
        u = _u, v = _v, c = _c, w = _w;
    }
};
/*====================*/
int n, m, s, t;
vector<int>G[N];
int mincost, maxflow;
int dep[N]; bool vis[N];
Edge edge[2 * M]; int cnt;
/*====================*/
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt++] = Edge(u, v, c, +w);
    edge[cnt++] = Edge(v, u, 0, -w);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool SPFA(void)
{
    for (int i = 1; i <= n; ++i)
    {
        vis[i] = false, dep[i] = INF;
    }
    vis[t] = true, dep[t] = 0;
    deque<int> q; q.push_back(t);
    while (!q.empty())
    {
        int x = q.front(); q.pop_front(), vis[x] = false;
        if (!q.empty() && dep[q.front()] > dep[q.back()])
        {
            swap(q.front(), q.back());
        }
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
            {
                dep[e1.v] = dep[x] - e1.w;
                if (!vis[e1.v])
                {
                    vis[e1.v] = true;
                    if (!q.empty() && dep[e1.v] < dep[q.front()])
                    {
                        q.push_front(e1.v);
```

```
                    }
                    else
                    {
                        q.push_back(e1.v);
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = ::n + ::n; m = 0;
        s = 1; t = ::n + ::n;
        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= ::n; ++i)
```

```
        {
            AddEdge(i, ::n + i, (i == 1 || i == ::n) ? 2 : 1, -1);
        }
        for (int i = 1; i <= ::m; ++i)
        {
            AddEdge(::n + ::edge[i].u, ::edge[i].v, INF, 0);
        }
        return MinCost();
    }
}
/*====================*/
bool vis[N];
vector<int>G[N];
void DFS(int cur)
{
    if (!vis[cur])
    {
        vis[cur] = true;
        cout << g[cur] << endl;
        for (auto nxt : G[cur])DFS(nxt);
    }
}
/*====================*/
int main()
{
    Read();
    _ZKW::Init();
    if (_ZKW::maxflow != 2)
    {
        cout << "No Solution!" << endl;
    }
    else
    {
        int cnt = 0;
        for (int i = 0; i < _ZKW::cnt; i += 2)
        {
            int u = _ZKW::edge[i].u;
            int v = _ZKW::edge[i].v;
            int f = _ZKW::edge[i].c;
            if (n + u == v)
            {
                if (f == 0)
                {
                    cnt++;
                }
            }
            else
            {
                if (f != _ZKW::INF)
                {
                    G[u - n].push_back(v);
                    G[v].push_back(u - n);
                }
            }
```

```
        }
        cout << cnt << endl; DFS(1); cout << S << endl;
    }
    return 0;
}
```

星际转移

题面

由于人类对自然资源的消耗，人们意识到大约在 2300 年之后，地球就不能再居住了。于是在月球上建立了新的绿地，以便在需要时移民。令人意想不到的是，2177 年冬由于未知的原因，地球环境发生了连锁崩溃，人类必须在最短的时间内迁往月球。

现有 n 个太空站位于地球与月球之间，且有 m 艘公共交通太空船在其间来回穿梭。每个太空站可容纳无限多的人，而每艘太空船 i 只可容纳 H_i 个人。每艘太空船将周期性地停靠一系列的太空站，例如：$\{1,3,4\}$ 表示该太空船将周期性地停靠太空站 $134134134\ldots$

每一艘太空船从一个太空站驶往任一太空站耗时均为 1。人们只能在太空船停靠太空站（或月球、地球）时上、下船。

初始时所有人全在地球上，太空船全在初始站。试设计一个算法，找出让所有人尽快地全部转移到月球上的运输方案。

题解

这道题极其不好做，恶魔妹妹调了四个小时。

首先这题很容易想到费用流，如果你这么想了那你就错了。因为这题求的是最少的时间，而同一时间可以有多个人在路线上。

最大流，流的是人。核心问题：怎么解决这个循环节。答案是暴力分层。具体到哪一层停下呢，这个就需要计算一下了。

这么考虑：最多 k 个人，最少 1 艘飞船，只能载 1 个人，每次都要经过 n 个空间站。也就是 650 天。

事实上这个计算方法并不是最坏情况，因为可以 m 艘飞船卡一下时间轴，每次要等 $n*m$ 天。不过这样就太麻烦了。考场上正确做法是设一个超大的数字如果TLE了就改小点。。。。

核心思路就是这样，这题细节较多，如果你WA在莫名其妙的地方上可以看看我的代码。

代码

```
#include<queue>
#include<vector>
#include<string>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
/*====================*/
const int N = 2e1 + 10;
const int M = 2e1 + 10;
const int K = 5e1 + 10;
/*====================*/
int n, m, k;
```

```
/*====================*/
struct Ship
{
    int h = 0;
    vector<int>id;
}ship[M];
/*====================*/
void Read(void)
{
    cin >> n >> m >> k; n += 2;
    for (int i = 1; i <= m; ++i)
    {
        cin >> ship[i].h;
        int cnt; cin >> cnt;
        for (int j = 1; j <= cnt; ++j)
        {
            int temp; cin >> temp;
            if (temp == +0)temp = n - 1;
            if (temp == -1)temp = n - 0;
            ship[i].id.push_back(temp);
        }
    }
}
/*====================*/
namespace _Dinic
{
    const int N = 12004 + 10;
    const int M = 60002 + 10;
    /*====================*/
    const int DAY = 800;
    const int INF = 0X3FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
```

```
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
        }
    }
    return flow;
}
/*====================*/
int Init(void)
{
    cnt = 0;
    n = DAY * ::n + 4; m = 0;
```

```
        s = DAY * ::n + 1, t = DAY * ::n + 4;
        int _s = DAY * ::n + 2, _t = DAY * ::n + 3;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        AddEdge(s, _s, ::k);
        AddEdge(_t, t, ::k);
        int maxflow = 0;
        for (int day = 1; day < DAY; ++day)
        {
            for (int i = 1; i <= ::m; ++i)
            {
                if (ship[i].id.size() == 1)continue;//搁浅了

                int u = ship[i].id[(day - 1) % ship[i].id.size()];
                int v = ship[i].id[(day - 0) % ship[i].id.size()];

                if (u == ::n - 1)AddEdge(_s, (day - 1) * ::n + u, INF);
                if (v == ::n - 0)AddEdge((day - 0) * ::n + v, _t, INF);

                u += (day - 1) * ::n;
                v += (day - 0) * ::n;

                AddEdge(u, v, ship[i].h);
            }
            for (int i = 1; i <= ::n; ++i)
            {
                int u = (day - 1) * ::n + i;
                int v = (day - 0) * ::n + i;
                AddEdge(u, v, INF);
            }
            maxflow += MaxFlow();
            if (maxflow == ::k)
            {
                return day;
            }
        }
        return 0;
    }
}
/*====================*/
int main()
{
    Read();
    cout << _Dinic::Init() << endl;
    return 0;
}
```

太空飞行计划

题面

W 教授正在为国家航天中心计划一系列的太空飞行。每次太空飞行可进行一系列商业性实验而获取利润。现已确定了一个可供选择的实验集合 $E = \{E_1, E_2, \cdots, E_m\}$，和进行这些实验需要使用的全部仪器的集合 $I = \{I_1, I_2, \cdots, I_n\}$。实验 E_j 需要用到的仪器是 I 的子集 $R_j \subseteq I$。

配置仪器 I_k 的费用为 c_k 美元。实验 E_j 的赞助商已同意为该实验结果支付 p_j 美元。W 教授的任务是找出一个有效算法，确定在一次太空飞行中要进行哪些实验并因此而配置哪些仪器才能使太空飞行的净收益最大。这里净收益是指进行实验所获得的全部收入与配置仪器的全部费用的差额。

对于给定的实验和仪器配置情况，编程找出净收益最大的试验计划。

题解

考虑到选中一个实验后必须选中该实验所需的仪器，是最大权闭合子图问题模型。运用最大流等于最小割的原理，思考怎么把题目转化成最小割。这里先考虑普通的ST割。

先建立源点 S 和所有的实验相连，容量为每个实验能赚到的钱。然后建立汇点 T 和所有仪器相连，容量为每个仪器需要花的钱。因为选中一个实验后必须选中该实验所需的仪器，所以实验与仪器之间的边容量设为 INF 防止被割断。

在纸上画一张草图随意假设一种选择方法。我们发现，割边是：未选中的实验赚的钱+选中的仪器花的钱。而我们要求的是：选中的实验赚的钱-选中的仪器花的钱。两式相加后为：所有实验赚的钱。

所以所求即为：所有实验赚的钱-割。因为我们希望赚的钱最多，所以在所有实验赚的钱固定的情况下，就只能让割的流量最小。而最小割等于最大流。所以最后就是：所有实验赚的钱-最大流。

考虑输出方案。有一个易错的地方是，并不是一条边满流就代表这条边是割边。可以举一个例子 $S \to X \to T$ 其中两条边的容量都为 1，可以发现无论如何都是满流。具体的来说，使用Dinic的时候有一个给图分层的过程。如果一个结点能被分层，那么一定与 S 相连。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 5e1 + 10;
const int M = 5e1 + 10;
/*====================*/
int m, n;//实验数，仪器数
int p[M];//一个实验能赚的钱
int c[N];//一个仪器要花的钱
vector<int>I[M];//一个实验要用的仪器
/*====================*/
void Read(void)
{
    cin >> m >> n;
    for (int i = 1; i <= m; ++i)
    {
        cin >> p[i]; char c = getchar();
        while (c == ' ')
        {
            int x; cin >> x;
```

```
            I[i].push_back(x);
            c = getchar();
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        cin >> c[i];
    }
}
/*====================*/
namespace _Dinic
{
    const int N = 102 + 10;
    const int M = 2600 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = 0;
        }
        queue<int>q; q.push(s);
        d[s] = 0; vis[s] = 1;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.v] && e.c > e.f)
```

```
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
        }
    }
    return flow;
}
/*====================*/
int Init(void)
{
    cnt = 0;
    n = ::m + ::n + 2; m = 0;
    s = ::m + ::n + 1; t = ::m + ::n + 2;
    for (int i = 1; i <= n; ++i)
    {
        G[i].clear();
    }
    for (int i = 1; i <= ::m; ++i)
    {
        int u, v, c;
        u = s, v = i, c = p[i];
        AddEdge(u, v, c); m += 2;
    }
    for (int i = 1; i <= ::m; ++i)
    {
        for (int j = 0; j < I[i].size(); ++j)
        {
```

```
                int u, v, c;
                u = i, v = ::m + I[i][j], c = INF;
                AddEdge(u, v, c); m += 2;
            }
        }
        for (int i = 1; i <= ::n; ++i)
        {
            int u, v, c;
            u = ::m + i, v = t, c = ::c[i];
            AddEdge(u, v, c); m += 2;
        }
        return MaxFlow();
    }
}
/*====================*/
int main()
{
    Read();
    /*====================*/
    int money = -_Dinic::Init();
    for (int i = 1; i <= m; ++i)
    {
        money += p[i];
    }
    /*====================*/
    vector<int>ansm;
    for (int i = 1; i <= m; ++i)
    {
        if (_Dinic::d[i] != 0)
        {
            ansm.push_back(i);
        }
    }
    /*====================*/
    vector<int>ansn;
    for (int i = 1; i <= n; ++i)
    {
        if (_Dinic::d[m + i] != 0)
        {
            ansn.push_back(i);
        }
    }
    /*====================*/
    for (int i = 0; i < ansm.size(); ++i)
    {
        cout << ansm[i] << " ";
    }cout << endl;
    for (int i = 0; i < ansn.size(); ++i)
    {
        cout << ansn[i] << " ";
    }cout << endl;
    cout << money << endl;
    return 0;
}
```

搭配飞行员

题面

飞行大队有若干个来自各地的驾驶员，专门驾驶一种型号的飞机，这种飞机每架有两个驾驶员，需一个正驾驶员和一个副驾驶员。由于种种原因，例如相互配合的问题，有些驾驶员不能在同一架飞机上飞行，问如何搭配驾驶员才能使出航的飞机最多。

因为驾驶工作分工严格，两个正驾驶员或两个副驾驶员都不能同机飞行。

题解

考虑到只有两种飞行员，每个飞行员只能出现在一架飞机上，是二分图最大匹配模型。

使用匈牙利的话复杂度是 $O(mn)$，但使用Dinic的话复杂度是 $O(m\sqrt{n})$。所以使用Dinic来跑二分图匹配。

建图的话所有边容量都为 1，源汇分别与正副飞行员相连就可以。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
namespace _Dinic
{
    const int N = 102 + 10;
    const int M = 2600 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
```

```
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
        }
    }
    return flow;
}
/*====================*/
int Init(void)
{
    cnt = 0;
```

```
        cin >> n >> m; s = ++n; t = ++n;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= m; ++i)
        {
            AddEdge(s, i, 1);
        }
        for (int i = m + 1; i + 2 <= n; ++i)
        {
            AddEdge(i, t, 1);
        }
        int u, v;
        while (cin >> u >> v)
        {
            AddEdge(u, v, 1);
        }
        return MaxFlow();
    }
}
/*====================*/
int main()
{
    cout << _Dinic::Init() << endl;
    return 0;
}
```

最小路径覆盖

题面

给定有向图 $G=(V,E)$。设 P 是 G 的一个简单路（顶点不相交）的集合。如果 V 中每个顶点恰好在 P 的一条路上，则称 P 是 G 的一个路径覆盖。P 中路径可以从 V 的任何一个顶点开始，长度也是任意的，特别地，可以为 0。G 的最小路径覆盖是 G 的所含路径条数最少的路径覆盖。

设计一个有效算法求一个有向无环图 G 的最小路径覆盖。

题解

不考虑最少这个条件，可以让每个点单独为一个简单路径。这样答案是 n。

因为不好表示一个点是否被覆盖，所以将点拆点后连边就是一张二分图，每个点变成 $x \to x'$。这些边为虚边，并不需要真的连边。

假设原图中存在边 $u \to v$，则二分图中存在边 $u \to v'$。这些边为实边，即在图中需要真的连边。

现在对这张二分图进行思考。逻辑上存在三条边，两条虚边一条实边即 $u \to u', u \to v', v \to v'$。

发现中间那条实边将两条虚边融合在了一起。所以我们想要更多的融合，这样每融合两条虚边答案就可以减少一个。

考虑到路径覆盖，即一条链式结构，不能存在一个点连向多个点或者多个点连向一个点。问题转化为二分图最大匹配。

这样最少路径条数就是：点数-最大匹配数。

考虑怎么输出方案。如果两个点在一条路径里，那这条路径一定在流网络里满流。把流网络里所有满流的边取出来。这些边的两个端点都在同一条路径里。只需要维护每条路径覆盖的点就可以。这里我使用并查集来处理。具体细节可以看代码。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 2e2 + 10;
const int M = 6e3 + 10;
/*====================*/
int n, m;
bool vis[N];
/*====================*/
namespace _Dinic
{
    const int N = 402 + 10;
    const int M = 6400 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = 0;
        }
        queue<int>q; q.push(s);
        d[s] = 0; vis[s] = 1;
```

```
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.v] && e.c > e.f)
                {
                    vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int& i = cur[x]; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
            {
                e.f += f; edge[G[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        n = 2 * ::n + 2; m = 0;
        s = 2 * ::n + 1, t = 2 * ::n + 2;
        for (int i = 1; i <= ::m; ++i)
        {
            int u, v;
            cin >> u >> v;
            AddEdge(u, v + ::n, 1);
        }
        for (int i = 1; i <= ::n; ++i)
        {
```

```
            AddEdge(s, i, 1);
            AddEdge(i + ::n, t, 1);
        }
        return MaxFlow();
    }
}
/*====================*/
class _DSU
{
public:
    int find(int cur)
    {
        return cur == pre[cur] ? cur : pre[cur] = find(pre[cur]);
    }
    void clear(void)
    {
        delete[] pre; pre = NULL;
        delete[] siz; siz = NULL;
    }
    void init(int n)
    {
        pre = new int[n + 1];
        siz = new int[n + 1];
        for (int i = 0; i <= n; ++i)
        {
            pre[i] = i, siz[i] = 1;
        }
    }
    int operator[](int cur)
    {
        return find(cur);
    }
    void merge(int u, int v)
    {
        u = find(u), v = find(v);
        if (siz[u] < siz[v])
        {
            pre[u] = v, siz[v] += siz[u];
        }
        else
        {
            pre[v] = u, siz[u] += siz[v];
        }
    }
    void operator()(int u, int v)
    {
        merge(u, v);
    }
private:
    int* pre = NULL;
    int* siz = NULL;
};
/*====================*/
int main()
{
```

```
    cin >> n >> m;
    _DSU dsu; dsu.init(n);
    int ans = n - _Dinic::Init();
    for (int i = 0; i < 2 * m; i += 2)
    {
        if (_Dinic::edge[i].f == 1)
        {
            int u = _Dinic::edge[i].u;
            int v = _Dinic::edge[i].v - n;
            if (dsu[u] != dsu[v])dsu(u, v);
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        vis[dsu[i]] = true;
    }
    for (int i = 1; i <= n; ++i)
    {
        if (vis[i])
        {
            for (int j = 1; j <= n; ++j)
            {
                if (dsu[j] == i)
                {
                    cout << j << " ";
                }
            }
            cout << endl;
        }
    }
    cout << ans << endl;
    return 0;
}
```

魔术球

题面

假设有 n 根柱子，现要按下述规则在这 n 根柱子中依次放入编号为 $1, 2, 3, 4, \cdots$ 的球。

1. 每次只能在某根柱子的最上面放球。
2. 在同一根柱子中，任何 2 个相邻球的编号之和为完全平方数。

试设计一个算法，计算出在 n 根柱子上最多能放多少个球。

题解

假设有 x 个球，那么如果两个球之和是完全平方数，就连一条边。

每根柱子其实就是一条路径。一个球只能在一根柱子上，所以符合最小路径覆盖模型。

首先打一个表，可以发现在 2000 个球的时候需要的柱子数是 63 。

因为 n 的范围在 55 以内，所以答案肯定在 2000 以下。然后二分一下球数。令最小路径等于 n 的时候就是答案。

值得一提的是，这题纯贪心也可以做。

代码

```
#include<cmath>
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 2e3 + 10;
/*====================*/
int n; bool vis[N];
/*====================*/
bool Check(int x)
{
    int d = double(sqrt(x) + 0.5);
    return (d * d) == x;
}
/*====================*/
namespace _Dinic
{
    const int N = 4002 + 10;
    const int M = 8016006 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = 0;
        }
        queue<int>q; q.push(s);
```

```
        d[s] = 0; vis[s] = 1;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.v] && e.c > e.f)
                {
                    vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int& i = cur[x]; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
            {
                e.f += f; edge[G[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        n = 2 * ::n + 2; m = 0;
        s = 2 * ::n + 1, t = 2 * ::n + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= ::n; ++i)
        {
```

```
            for (int j = i + 1; j <= ::n; ++j)
            {
                int u = i, v = j;
                if (Check(u + v))
                {
                    AddEdge(u, v + ::n, 1);
                }
            }
        }
        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(s, i, 1);
            AddEdge(i + ::n, t, 1);
        }
        return MaxFlow();
    }
}
/*====================*/
class _DSU
{
public:
    int find(int cur)
    {
        return cur == pre[cur] ? cur : pre[cur] = find(pre[cur]);
    }
    void clear(void)
    {
        delete[] pre; pre = NULL;
        delete[] siz; siz = NULL;
    }
    void init(int n)
    {
        pre = new int[n + 1];
        siz = new int[n + 1];
        for (int i = 0; i <= n; ++i)
        {
            pre[i] = i, siz[i] = 1;
        }
    }
    int operator[](int cur)
    {
        return find(cur);
    }
    void merge(int u, int v)
    {
        u = find(u), v = find(v);
        if (siz[u] < siz[v])
        {
            pre[u] = v, siz[v] += siz[u];
        }
        else
        {
            pre[v] = u, siz[u] += siz[v];
        }
    }
```

```
    void operator()(int u, int v)
    {
        merge(u, v);
    }
private:
    int* pre = NULL;
    int* siz = NULL;
};
/*====================*/
int main()
{
    int x; cin >> x;
    int l = 1, r = 2000;
    int _n = 0;
    while (l <= r)
    {
        n = (l + r) >> 1;
        int temp = n - _Dinic::Init();
        if (temp <= x)
        {
            l = n + 1; _n = n;
        }
        else
        {
            r = n - 1;
        }
    }
    n = _n;
    _DSU dsu; dsu.init(n);
    _Dinic::Init(); cout << n << endl;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 0; j < _Dinic::G[i].size(); ++j)
        {
            if (_Dinic::edge[_Dinic::G[i][j]].f == 1)
            {
                int u = _Dinic::edge[_Dinic::G[i][j]].u + 0;
                int v = _Dinic::edge[_Dinic::G[i][j]].v - n;
                if (v <= n && dsu[u] != dsu[v])dsu(u, v);
            }
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        vis[dsu[i]] = true;
    }
    for (int i = 1; i <= n; ++i)
    {
        if (vis[i])
        {
            for (int j = 1; j <= n; ++j)
            {
                if (dsu[j] == i)
                {
                    cout << j << " ";
```

```
                }
            }
            cout << endl;
        }
    }
    return 0;
}
```

方格取数

题面

在一个有 $m \times n$ 个方格的棋盘中，每个方格中有一个正整数。

现要从方格中取数，使任意 2 个数所在方格没有公共边，且取出的数的总和最大。试设计一个满足要求的取数算法。

题解

注意这题是 n 行 m 列。

首先在纸上随便画一些方格，然后按照国际象棋棋盘那样涂一下色。我们发现相邻的不同颜色的格子是不能同时取的。

有一个很明显错误的贪心，选所有黑色格子和白色格子中最大的那一个。hack样例：1 行 4 列 2112。

先这样建图

1. 如果格子 x 为黑色 $S \to X$。
2. 如果格子 x 为白色 $X \to T$。
3. 每个格子 x 向他四周的四个格子连一条 INF 的边。

考虑集合划分，n 个数划分成**选中**并且**能被选中**的格子集合 S 和**没选中**或者**不能选中**的格子集合 T。这里注意集合里存的是割边，而不是点。因为我们ST割是用边来表示一个点是否被选中。

因为我们给不能同时选的格子之间都连了 INF 的边，所以必然不会被割断，也就是说选中一个格子，所有不能被选中的格子也被包含了，其实就是一个闭合子图。那么不能选的格子到 T 的边就变成了割边。我们发现这些割边是属于 T 集合里的。

还剩下 S 到没选中的点之间的边，这些边也是割边。

现在来看一下割边集合都有哪些种类的割边：不能选中或没选中的格子=获得不到的格子权值。

我们要求的是：选中且能被选中=所有格子-不能选中或没选中的格子=所有格子-ST割。

因为要令选中的格子最大，所以就要ST割最小。最大流等于最小割。然后就套板子就可以了。

这题的关键在于理解ST割和闭合子图。一个点被选中不取决于这个点在闭合子图里，而是取决 $S \to X$ 这条割边在闭合子图里。

顺便提一下，如果这题要求输出方案的话，那参考***太空飞行计划***。

1. 跟 S 直接相连的点 x，如果BFS完 $vis[x] = true$，则被选中。
2. 跟 T 直接相连的点 x，如果BFS完 $vis[x] \neq true$，则被选中。

代码

```
#include<queue>
#include<vector>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int N = 3e1 + 10;
const int M = 3e1 + 10;
/*====================*/
int n, m;
int num[N][M];
/*====================*/
void Read(void)
{
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= m; ++j)
        {
            cin >> num[i][j];
        }
    }
}
/*====================*/
int f(int x, int y)
{
    return (x - 1) * m + y;
}
/*====================*/
const int dx[] = { -1,1,0,0 };
const int dy[] = { 0,0,-1,1 };
/*====================*/
namespace _Dinic
{
    const int N = 902 + 10;
    const int M = 2704 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
```

```
/*====================*/
void AddEdge(int u, int v, int c)
{
    edge[cnt++] = Edge(u, v, c, 0);
    edge[cnt++] = Edge(v, u, 0, 0);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
```

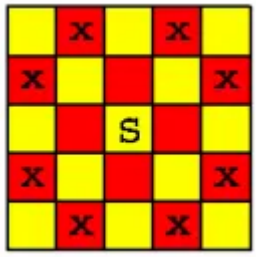
```
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        n = ::n * ::m + 2; m = 0;
        s = ::n * ::m + 1; t = ::n * ::m + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= ::n; ++i)
        {
            for (int j = 1; j <= ::m; ++j)
            {
                int curx = i, cury = j, cur = f(curx, cury);
                if ((i + j) & 1)
                {
                    AddEdge(s, cur, num[curx][cury]);
                    for (int k = 0; k < 4; ++k)
                    {
                        int nxtx = i + dx[k], nxty = j + dy[k], nxt = f(nxtx,
nxty);
                        if (nxtx<1 || nxtx>::n)continue;
                        if (nxty<1 || nxty>::m)continue;
                        AddEdge(cur, nxt, INF);
                    }
                }
                else
                {
                    AddEdge(cur, t, num[curx][cury]);
                }
            }
        }
        return MaxFlow();
    }
}
/*====================*/
int main()
{
    Read();
    int ans = 0;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= m; ++j)
        {
            ans += num[i][j];
        }
    }
    ans -= _Dinic::Init();
    cout << ans << endl;
    /*这段代码对应输出方案数的情况。把上面删了直接交下面也对。
```

```
    Read();
    int ans = 0;
    _Dinic::Init();
    for (int i = 0; i < _Dinic::cnt; i += 2)
    {
        int u = _Dinic::edge[i].u;
        int v = _Dinic::edge[i].v;
        if (u == _Dinic::s)
        {
            if (_Dinic::vis[v] == true)
            {
                ans += _Dinic::edge[i].c;
                //printf("%d ", _Dinic::edge[i].c);
            }
        }
        if (v == _Dinic::t)
        {
            if (_Dinic::vis[u] != true)
            {
                ans += _Dinic::edge[i].c;
                //printf("%d ", _Dinic::edge[i].c);
            }
        }
    }
    printf("%d\n", ans);
    */
    return 0;
}
```

骑士共存

题面

在一个 $n \times n$ 个方格的国际象棋棋盘上，马（骑士）可以攻击的棋盘方格如图所示。棋盘上某些方格设置了障碍，骑士不得进入。



对于给定的 $n \times n$ 个方格的国际象棋棋盘和障碍标志，计算棋盘上最多可以放置多少个骑士，使得它们彼此互不攻击。

题解

这题和方格取数基本上是一道题。

类比一下：

1. 方格取数不能取周围四个格子，骑士共存不能放攻击点八个格子。
2. 方格取数每个格子取走的是格子上的数据，其实共存放一个点提供 1 的贡献。

唯一的区别就是多了一些障碍点。棋盘里假设没有这些点就可以了。说白了这题就是方格取数。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 2e2 + 10;
const int M = 4e4 + 10;
/*====================*/
const int dx[] = { 1,1,-1,-1,2,2,-2,-2 };
const int dy[] = { 2,-2,2,-2,1,-1,1,-1 };
/*====================*/
int n, m; bool stone[N][N];
/*====================*/
int f(int x, int y)
{
    return (x - 1) * n + y;
}
/*====================*/
void Read(void)
{
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        int x, y;
        cin >> x >> y;
        stone[x][y] = true;
    }
}
/*====================*/
namespace _Dinic
{
    const int N = 40002 + 10;
    const int M = 200000 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
```

```
void AddEdge(int u, int v, int c)
{
    edge[cnt++] = Edge(u, v, c, 0);
    edge[cnt++] = Edge(v, u, 0, 0);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
        }
```

```
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        n = ::n * ::n + 2; m = 0;
        s = ::n * ::n + 1; t = ::n * ::n + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int x = 1; x <= ::n; ++x)
        {
            for (int y = 1; y <= ::n; ++y)
            {
                if (stone[x][y])continue;
                if ((x + y) & 1)
                {
                    AddEdge(s, f(x, y), 1);
                    for (int k = 0; k < 8; ++k)
                    {
                        int tx = x + dx[k];
                        int ty = y + dy[k];
                        if (tx<1 || tx>::n)continue;
                        if (ty<1 || ty>::n)continue;
                        if (stone[tx][ty])continue;
                        AddEdge(f(x, y), f(tx, ty), INF);
                    }
                }
                else
                {
                    AddEdge(f(x, y), t, 1);
                }
            }
        }

        return MaxFlow();
    }
}
/*====================*/
int main()
{
    Read();
    int ans = n * n - m;
    ans -= _Dinic::Init();
    cout << ans << endl;
    return 0;
}
```

试题库

题面

假设一个试题库中有 n 道试题。每道试题都标明了所属类别。同一道题可能有多个类别属性。现要从题库中抽取 m 道题组成试卷。并要求试卷包含指定类型的试题。试设计一个满足要求的组卷算法。

题解

这题应该是一个很简单的题，我不知道为什么LOJ上过题人数这么少。

这题题意有些不清楚的地方，最关键的一条是，一道题只能用在一个知识点上面。

因为一道题可以用在多个知识点上面，我们不知道用在哪一个上面，所以，流。

源点到所有题连一条容量为 1 的边，限制一道题只能选一次，并且只能用在一个知识点上面。

每道题到他对应的知识点连一条容量为 1 的边，流了这条边说明这道题用在这个知识点上面。

每个知识点到汇点连一个容量为 k_i 的边，限制住每个类型的题需要多少个。

然后流。如果最大流不是 $\sum_{i=1}^{k} k_i$，那就说明无论如何也组不出来。那就直接输出。

然后枚举所有题到知识点直接的边，如果满流，说明这道题用在这个知识点上。存一下，最后输出。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int K = 2e1 + 10;
const int N = 1e3 + 10;
/*====================*/
int k, n;
int need[K];
vector<int>ans[K];
vector<int>kind[N];
/*====================*/
void Read(void)
{
    cin >> k >> n;
    for (int i = 1; i <= k; ++i)
    {
        cin >> need[i]; need[0] += need[i];
    }
    for (int i = 1; i <= n; ++i)
    {
        int p; cin >> p;
        while (p--)
        {
            int temp; cin >> temp; kind[i].push_back(temp);
        }
    }
}
/*====================*/
namespace _Dinic
```

```
{
    const int N = 1022 + 10;
    const int M = 21020 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = 0;
        }
        queue<int>q; q.push(s);
        d[s] = 0; vis[s] = 1;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.v] && e.c > e.f)
                {
                    vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
```

```
        for (int& i = cur[x]; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
            {
                e.f += f; edge[G[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        n = ::n + ::k + 2; m = 0;
        s = ::n + ::k + 1; t = ::n + ::k + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(s, i, 1);
        }
        for (int i = 1; i <= ::n; ++i)
        {
            int u = i;
            for (int j = 0; j < kind[i].size(); ++j)
            {
                int v = ::n + kind[i][j];
                AddEdge(u, v, 1);
            }
        }
        for (int i = 1; i <= ::k; ++i)
        {
            AddEdge(::n + i, t, need[i]);
        }

        return MaxFlow();
    }
```

```
}
/*====================*/
int main()
{
    Read();
    int maxflow = _Dinic::Init();
    if (maxflow == need[0])
    {
        for (int i = 0; i < _Dinic::cnt; i += 2)
        {
            int f = _Dinic::edge[i].f;
            int u = _Dinic::edge[i].u;
            int v = _Dinic::edge[i].v - n;
            if (1 <= u && u <= n && f == 1)
            {
                ans[v].push_back(u);
            }
        }
        for (int i = 1; i <= k; ++i)
        {
            cout << i << ": ";
            for (auto unit : ans[i])
            {
                cout << unit << " ";
            }
            cout << endl;
        }
    }
    else
    {
        cout << "No Solution!" << endl;
    }
    return 0;
}
```

圆桌聚餐

题面

假设有来自 m 个不同单位的代表参加一次国际会议。每个单位的代表数分别为 r_i。会议餐厅共有 n 张餐桌，每张餐桌可容纳 c_i 个代表就餐。
为了使代表们充分交流，希望从同一个单位来的代表不在同一个餐桌就餐。

试设计一个算法，给出满足要求的代表就餐方案。

题解

这题跟试题库那道题其实一模一样。

源点到每个公司连边 r_i，限制每个公司有 r_i个人。

每个公司到每个桌子连边 1，限制每个桌子每个公司只能坐一个人。

每个桌子到汇点连边 c_i，限制每个公司最多坐 c_i 个人。

还不明白，看试题库那题的题解。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int M = 150 + 10;
const int N = 270 + 10;
/*====================*/
int m, n;
int r[M], c[N];
vector<int>ans[M];
/*====================*/
void Read(void)
{
    cin >> m >> n;
    for (int i = 1; i <= m; ++i)
    {
        cin >> r[i]; r[0] += r[i];
    }
    for (int i = 1; i <= n; ++i)
    {
        cin >> c[i];
    }
}
/*====================*/
namespace _Dinic
{
    const int N = 422 + 10;
    const int M = 40920 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
```

```
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int MaxFlow(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
        for (int i = 1; i <= n; ++i)
        {
            cur[i] = 0;
        }
    }
    return flow;
}
/*====================*/
int Init(void)
{
```

```
        cnt = 0;
        n = ::m + ::n + 2; m = 0;
        s = ::m + ::n + 1; t = ::m + ::n + 2;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::m; ++i)
        {
            AddEdge(s, i, r[i]);
        }
        for (int i = 1; i <= ::m; ++i)
        {
            for (int j = 1; j <= ::n; ++j)
            {
                AddEdge(i, ::m + j, 1);
            }
        }
        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(::m + i, t, c[i]);
        }

        return MaxFlow();
    }
}
/*====================*/
int main()
{
    Read();
    int maxflow = _Dinic::Init();
    if (maxflow == r[0])
    {
        cout << "1" << endl;
        for (int i = 0; i < _Dinic::cnt; i += 2)
        {
            int f = _Dinic::edge[i].f;
            int u = _Dinic::edge[i].u;
            int v = _Dinic::edge[i].v - m;
            if (1 <= u && u <= m && f == 1)
            {
                ans[u].push_back(v);
            }
        }
        for (int i = 1; i <= m; ++i)
        {
            for (auto unit : ans[i])
            {
                cout << unit << " ";
            }
            cout << endl;
        }
    }
    else
```

```
    {
        cout << "0" << endl;
    }
    return 0;
}
```

分配问题

题面

有 n 件工作要分配给 n 个人做。第 i 个人做第 j 件工作产生的效益为 c_{ij}。试设计一个将 n 件工作分配给 n 个人做的分配方案，使产生的总效益最大。

题解

看到这类分配问题，匹配问题，应该下意识想到网络流了。

很简单的费用流，源点连向每个人，每个工作连向汇点，容量 1 费用 0。

问最小收益，那设费用为正数，因为是最小费用最大流。每个人到每个工作连边，费用为 $+c_{i,j}$。

问最大收益，那设费用为负数，这样越大的费用越小，结果取负就行。每个人到每个工作连边，费用为 $-c_{i,j}$。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*======================*/
const int N = 1e2 + 10;
/*======================*/
int n;
int c[N][N];
/*======================*/
void Read(void)
{
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
        {
            cin >> c[i][j];
        }
    }
}
/*======================*/
namespace _ZKW
{
    const int N = 202 + 10;
    const int M = 10200 + 10;
    /*======================*/
    const int INF = 0X7FFFFFFF;
    /*======================*/
```

```cpp
struct Edge
{
    int u, v, c, w;
    Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
    {
        u = _u, v = _v, c = _c, w = _w;
    }
};
/*====================*/
int n, m, s, t;
vector<int>G[N];
int mincost, maxflow;
int dep[N]; bool vis[N];
Edge edge[2 * M]; int cnt;
/*====================*/
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt++] = Edge(u, v, c, +w);
    edge[cnt++] = Edge(v, u, 0, -w);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool SPFA(void)
{
    for (int i = 1; i <= n; ++i)
    {
        vis[i] = false, dep[i] = INF;
    }
    vis[t] = true, dep[t] = 0;
    deque<int> q; q.push_back(t);
    while (!q.empty())
    {
        int x = q.front(); q.pop_front(), vis[x] = false;
        if (!q.empty() && dep[q.front()] > dep[q.back()])
        {
            swap(q.front(), q.back());
        }
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
            {
                dep[e1.v] = dep[x] - e1.w;
                if (!vis[e1.v])
                {
                    vis[e1.v] = true;
                    if (!q.empty() && dep[e1.v] < dep[q.front()])
                    {
                        q.push_front(e1.v);
                    }
                    else
                    {
                        q.push_back(e1.v);
```

```
                    }
                }
            }
        }
    }
    return dep[s] < INF;
}
int DFS(int x, int k)
{
    vis[x] = true; int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int i = 0; i < G[x].size(); ++i)
    {
        Edge& e1 = edge[G[x][i] ^ 0];
        Edge& e2 = edge[G[x][i] ^ 1];
        if (vis[e1.v] || e1.c == 0)continue;
        if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
        {
            e1.c -= f, e2.c += f; flow += f, k -= f;
            mincost += f * e1.w; if (k == 0) break;
        }
    }
    return flow;
}
int MinCost(void)
{
    while (SPFA())
    {
        vis[t] = true;
        while (vis[t])
        {
            for (int i = 1; i <= n; ++i)
            {
                vis[i] = false;
            }
            maxflow += DFS(s, INF);
        }
    }
    return mincost;
}
/*====================*/
int Init1(void)
{
    n = ::n + ::n + 2; m = 0;
    s = ::n + ::n + 1, t = ::n + ::n + 2;

    mincost = maxflow = cnt = 0;
    for (int i = 1; i <= n; ++i)
    {
        G[i].clear();
    }

    for (int i = 1; i <= ::n; ++i)
    {
        AddEdge(s, i, 1, 0);
```

```
            for (int j = 1; j <= ::n; ++j)
            {
                AddEdge(i, ::n + j, 1, +c[i][j]);
            }
            AddEdge(::n + i, t, 1, 0);
        }

        return MinCost();
    }
    int Init2(void)
    {
        n = ::n + ::n + 2; m = 0;
        s = ::n + ::n + 1, t = ::n + ::n + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(s, i, 1, 0);
            for (int j = 1; j <= ::n; ++j)
            {
                AddEdge(i, ::n + j, 1, -c[i][j]);
            }
            AddEdge(::n + i, t, 1, 0);
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    cout << +_ZKW::Init1() << endl;
    cout << -_ZKW::Init2() << endl;
    return 0;
}
```

运输问题

题面

W 公司有 m 个仓库和 n 个零售商店。第 i 个仓库有 a_i 个单位的货物；第 j 个零售商店需要 b_j 个单位的货物。货物供需平衡，即 $\sum\limits_{i=1}^{m} a_i = \sum\limits_{j=1}^{n} b_j$。从第 i 个仓库运送每单位货物到第 j 个零售商店的费用为 c_{ij}。试设计一个将仓库中所有货物运送到零售商店的运输方案，使总运输费用最少。

题解

如果做过分配问题应该会这道题，如果不会那就去看分配问题。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e2 + 10;
const int M = 1e2 + 10;
/*====================*/
int m, n;
int c[M][N];
int a[M], b[N];
/*====================*/
void Read(void)
{
    cin >> m >> n;
    for (int i = 1; i <= m; ++i)
    {
        cin >> a[i];
    }
    for (int i = 1; i <= n; ++i)
    {
        cin >> b[i];
    }
    for (int i = 1; i <= m; ++i)
    {
        for (int j = 1; j <= n; ++j)
        {
            cin >> c[i][j];
        }
    }
}
/*====================*/
namespace _ZKW
{
    const int N = 202 + 10;
    const int M = 10200 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
```

```
vector<int>G[N];
int mincost, maxflow;
int dep[N]; bool vis[N];
Edge edge[2 * M]; int cnt;
/*====================*/
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt++] = Edge(u, v, c, +w);
    edge[cnt++] = Edge(v, u, 0, -w);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool SPFA(void)
{
    for (int i = 1; i <= n; ++i)
    {
        vis[i] = false, dep[i] = INF;
    }
    vis[t] = true, dep[t] = 0;
    deque<int> q; q.push_back(t);
    while (!q.empty())
    {
        int x = q.front(); q.pop_front(), vis[x] = false;
        if (!q.empty() && dep[q.front()] > dep[q.back()])
        {
            swap(q.front(), q.back());
        }
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
            {
                dep[e1.v] = dep[x] - e1.w;
                if (!vis[e1.v])
                {
                    vis[e1.v] = true;
                    if (!q.empty() && dep[e1.v] < dep[q.front()])
                    {
                        q.push_front(e1.v);
                    }
                    else
                    {
                        q.push_back(e1.v);
                    }
                }
            }
        }
    }
    return dep[s] < INF;
}
int DFS(int x, int k)
{
    vis[x] = true; int flow = 0, f;
```

```
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init1(void)
    {
        n = ::m + ::n + 2; m = 0;
        s = ::m + ::n + 1, t = ::m + ::n + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::m; ++i)
        {
            AddEdge(s, i, a[i], 0);
        }
        for (int i = 1; i <= ::m; ++i)
        {
            for (int j = 1; j <= ::n; ++j)
            {
                AddEdge(i, ::m + j, INF, +c[i][j]);
            }
        }
        for (int i = 1; i <= ::n; ++i)
        {
```

```
            AddEdge(::m + i, t, b[i], 0);
        }

        return MinCost();
    }
    int Init2(void)
    {
        n = ::m + ::n + 2; m = 0;
        s = ::m + ::n + 1, t = ::m + ::n + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::m; ++i)
        {
            AddEdge(s, i, a[i], 0);
        }
        for (int i = 1; i <= ::m; ++i)
        {
            for (int j = 1; j <= ::n; ++j)
            {
                AddEdge(i, ::m + j, INF, -c[i][j]);
            }
        }
        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(::m + i, t, b[i], 0);
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    cout << +_ZKW::Init1() << endl;
    cout << -_ZKW::Init2() << endl;
    return 0;
}
```

负载平衡

题面

G 公司有 n 个沿铁路运输线环形排列的仓库，每个仓库存储的货物数量不等。如何用最少搬运量可以使 n 个仓库的库存数量相同。搬运货物时，只能在相邻的仓库之间搬运。

题解

一道非常简单的费用流。设一单位货物为一单位流。因为搬运货物只能在相邻仓库之间搬运，先把这个环建出来，注意是双向边。因为搬运一单位就是一单位的搬运量，所以费用为 1。

设最后每个仓库库存的数量为 x。如果当前仓库大于 x，和源点相连，容量为 $a_i - x$，因为他最多只能流出这么多货物。如果当前仓库小于 x，和汇点相连，容量为 $x - a_i$，因为他最多只能接收这么多货物。

这题跟别的题有点不一样在，除开源汇，他的关系不是一张二分图。但是网络流不管你有环还是无环，都能跑出来结果。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e2 + 10;
/*====================*/
int n, a[N];
/*====================*/
void Read(void)
{
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        cin >> a[i]; a[0] += a[i];
    }
    a[0] /= n;
}
/*====================*/
namespace _ZKW
{
    const int N = 102 + 10;
    const int M = 300 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
```

```
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt++] = Edge(u, v, c, +w);
    edge[cnt++] = Edge(v, u, 0, -w);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool SPFA(void)
{
    for (int i = 1; i <= n; ++i)
    {
        vis[i] = false, dep[i] = INF;
    }
    vis[t] = true, dep[t] = 0;
    deque<int> q; q.push_back(t);
    while (!q.empty())
    {
        int x = q.front(); q.pop_front(), vis[x] = false;
        if (!q.empty() && dep[q.front()] > dep[q.back()])
        {
            swap(q.front(), q.back());
        }
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
            {
                dep[e1.v] = dep[x] - e1.w;
                if (!vis[e1.v])
                {
                    vis[e1.v] = true;
                    if (!q.empty() && dep[e1.v] < dep[q.front()])
                    {
                        q.push_front(e1.v);
                    }
                    else
                    {
                        q.push_back(e1.v);
                    }
                }
            }
        }
    }
    return dep[s] < INF;
}
int DFS(int x, int k)
{
    vis[x] = true; int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int i = 0; i < G[x].size(); ++i)
    {
        Edge& e1 = edge[G[x][i] ^ 0];
        Edge& e2 = edge[G[x][i] ^ 1];
```

```
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = ::n + 2; m = 0;
        s = ::n + 1, t = ::n + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::n; ++i)
        {
            if (a[i] > a[0])
            {
                AddEdge(s, i, a[i] - a[0], 0);
            }
            if (a[i] < a[0])
            {
                AddEdge(i, t, a[0] - a[i], 0);
            }
            AddEdge(i, i + 1 > ::n ? 1 : i + 1, INF, 1);
            AddEdge(i + 1 > ::n ? 1 : i + 1, i, INF, 1);
        }

        return MinCost();
    }
}
/*====================*/
```

```
int main()
{
    Read();
    cout << _ZKW::Init() << endl;
    return 0;
}
```

数字梯形

题面

给定一个由 n 行数字组成的数字梯形如下方所示。梯形的第一行有 m 个数字。从梯形的顶部的 m 个数字开始，在每个数字处可以沿左下或右下方向移动，形成一条从梯形的顶至底的路径。

```
    2 3
   3 4 5
  9 10 9 1
 1 1 10 1 1
1 1 10 12 1 1
```

分别遵守以下规则：

1. 从梯形的顶至底的 m 条路径互不相交；
2. 从梯形的顶至底的 m 条路径仅在数字结点处相交；
3. 从梯形的顶至底的 m 条路径允许在数字结点相交或边相交。

题解

很标准的费用流。每个数字当作费用（记得取负）。

注意一下起点一定是第一行的每一个数字。

第一问：把每个数字拆点限制只用一次。所有边容量都为 1。

第二问：每个数字拆点之间的容量改为 INF。记得最后一行到汇点的容量也改成 INF。因为可能多条路径最后在同一个地方流出。

第三问：路径也可以重复了，那就把三角形之间的连边容量也改为 INF。但是起点到第一层的边容量不变，因为要限制他们作为起点。

代码

```
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
/*======================*/
const int M = 2e1 + 10;
const int N = 2e1 + 10;
/*======================*/
int m, n;
int idx = 0;
vector<int>idx1[N];
vector<int>idx2[N];
vector<int>line[N];
```

```
/*====================*/
void Read(void)
{
    cin >> m >> n;
    for (int i = 1; i <= n; ++i)
    {
        line[i].push_back(0);
        for (int j = 1; j <= m + i - 1; ++j)
        {
            int temp; cin >> temp;
            line[i].push_back(temp);
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        idx1[i].push_back(0);
        for (int j = 1; j <= m + i - 1; ++j)
        {
            idx1[i].push_back(++idx);
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        idx2[i].push_back(0);
        for (int j = 1; j <= m + i - 1; ++j)
        {
            idx2[i].push_back(++idx);
        }
    }
}
/*====================*/
namespace _ZKW
{
    const int N = 1182 + 10;
    const int M = 2950 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
```

```
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
```

```
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init1(void)
    {
        n = ::idx + 2; m = 0;
        s = ::idx + 1; t = ::idx + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        for (int i = 1; i <= ::n; ++i)
        {
            for (int j = 1; j <= ::m + i - 1; ++j)
            {
                AddEdge(idx1[i][j], idx2[i][j], 1, -line[i][j]);
                if (i == 1)
                {
                    AddEdge(s, idx1[i][j], 1, 0);
                }
                if (i == ::n)
                {
                    AddEdge(idx2[i][j], t, 1, 0);
                }
                if (i != ::n)
                {
                    AddEdge(idx2[i][j], idx1[i + 1][j + 0], 1, 0);
                    AddEdge(idx2[i][j], idx1[i + 1][j + 1], 1, 0);
                }
            }
        }
```

```
    return MinCost();
}
int Init2(void)
{
    n = ::idx + 2; m = 0;
    s = ::idx + 1; t = ::idx + 2;

    mincost = maxflow = cnt = 0;
    for (int i = 1; i <= n; ++i)
    {
        G[i].clear();
    }

    for (int i = 1; i <= ::n; ++i)
    {
        for (int j = 1; j <= ::m + i - 1; ++j)
        {
            AddEdge(idx1[i][j], idx2[i][j], INF, -line[i][j]);
            if (i == 1)
            {
                AddEdge(s, idx1[i][j], 1, 0);
            }
            if (i == ::n)
            {
                AddEdge(idx2[i][j], t, INF, 0);
            }
            if (i != ::n)
            {
                AddEdge(idx2[i][j], idx1[i + 1][j + 0], 1, 0);
                AddEdge(idx2[i][j], idx1[i + 1][j + 1], 1, 0);
            }
        }
    }

    return MinCost();
}
int Init3(void)
{
    n = ::idx + 2; m = 0;
    s = ::idx + 1; t = ::idx + 2;

    mincost = maxflow = cnt = 0;
    for (int i = 1; i <= n; ++i)
    {
        G[i].clear();
    }

    for (int i = 1; i <= ::n; ++i)
    {
        for (int j = 1; j <= ::m + i - 1; ++j)
        {
            AddEdge(idx1[i][j], idx2[i][j], INF, -line[i][j]);
            if (i == 1)
            {
```

```
                    AddEdge(s, idx1[i][j], 1, 0);
                }
                if (i == ::n)
                {
                    AddEdge(idx2[i][j], t, INF, 0);
                }
                if (i != ::n)
                {
                    AddEdge(idx2[i][j], idx1[i + 1][j + 0], INF, 0);
                    AddEdge(idx2[i][j], idx1[i + 1][j + 1], INF, 0);
                }
            }
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    cout << -_ZKW::Init1() << endl;
    cout << -_ZKW::Init2() << endl;
    cout << -_ZKW::Init3() << endl;
    return 0;
}
```

餐巾计划

题面

一个餐厅在相继的 n 天里，每天需用的餐巾数不尽相同。假设第 i 天需要 r_i 块餐巾。餐厅可以购买新的餐巾，每块餐巾的费用为 P 分；或者把旧餐巾送到快洗部，洗一块需 M 天，其费用为 F 分；或者送到慢洗部，洗一块需 N 天，其费用为 S 分（$S < F$）。

每天结束时，餐厅必须决定将多少块脏的餐巾送到快洗部，多少块餐巾送到慢洗部，以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和，要满足当天的需求量。

试设计一个算法为餐厅合理地安排好 n 天中餐巾使用计划,使总的花费最小。

题解

首先有一个很直观的错误做法。

把每天拆成两个点 x_s, x_t，然后连边 $S \to x_s \to x_t \to T$。送去送洗和保存起来延期送洗，则是 $x_t \to (x+d)_s, x_t \to (x+1)_t$。

为什么是错误的呢，因为最小费用最大流，首先要保证是最大流。那么每天必然流满 r_i。根本原因是因为拿去送洗的餐巾多占用了一条餐巾的需求，但在流量中依旧为 1。

既然如此，只需要让拿去送洗的餐巾多占用一次流量即可。思路是将送洗的餐巾独立出来。具体措施是，先假设每天用的都是买来的，断开 $x_s \to x_t \to T$，连上 $x_s \to T$。然后考虑送洗的餐巾，让送洗的餐巾直接从源点获得，而不是继续使用 $S \to x_s \to x_t$ 获得的流量。

错误做法与正确做法的本质区别是：错误做法流的是餐巾，正确做法流的是需求。

代码

```
#include<queue>
#include<vector>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e3 + 10;
/*====================*/
int n, p, d1, w1, d2, w2, r[N];
/*====================*/
namespace _ZKW
{
    const int N = 2002 + 10;
    const int M = 5996 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
```

```
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
```

```
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = 2 * ::n + 2; m = 0;
        s = 2 * ::n + 1; t = 2 * ::n + 2;
        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= ::n; ++i)
        {
            AddEdge(s, i, r[i], p);
            AddEdge(i, t, r[i], 0);
            AddEdge(s, i + ::n, r[i], 0);
            if (i != ::n)AddEdge(i + ::n, i + 1 + ::n, INF, 0);
            if (i + d1 <= ::n)AddEdge(i + ::n, i + d1, INF, w1);
            if (i + d2 <= ::n)AddEdge(i + ::n, i + d2, INF, w2);
        }
        return MinCost();
    }
}
/*====================*/
int main()
{
    cin >> n;
    cin >> p;
    cin >> d1 >> w1;
    cin >> d2 >> w2;
    for (int i = 1; i <= n; ++i)
    {
        cin >> r[i];
    }
    cout << _ZKW::Init() << endl;
    return 0;
}
```

深海机器人

题面

深海资源考察探险队的潜艇将到达深海的海底进行科学考察。

潜艇内有多个深海机器人。潜艇到达深海海底后，深海机器人将离开潜艇向预定目标移动。

深海机器人在移动中还必须沿途采集海底生物标本。沿途生物标本由最先遇到它的深海机器人完成采集。

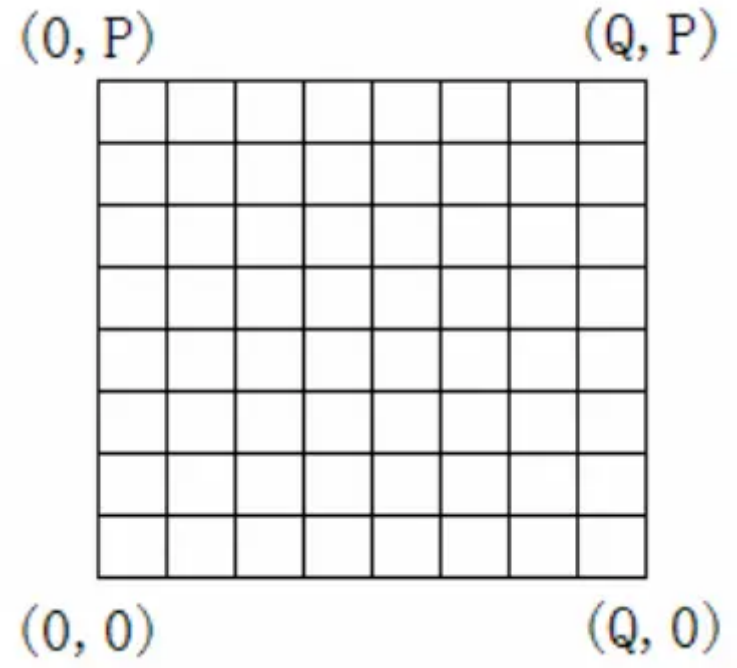
每条预定路径上的生物标本的价值是已知的，而且生物标本只能被采集一次。

本题限定深海机器人只能从其出发位置沿着向北或向东的方向移动，而且多个深海机器人可以在同一时间占据同一位置。

用一个 $P \times Q$ 网格表示深海机器人的可移动位置。西南角的坐标为 $(0, 0)$，东北角的坐标为 (Q, P)。

给定每个深海机器人的出发位置和目标位置，以及每条网格边上生物标本的价值。

计算深海机器人的最优移动方案， 使深海机器人到达目的地后，采集到的生物标本的总价值最高。



题解

费用流裸题。

这题的特点在于每个路径只有经过第一次的时候可以收获价值。

所以对每条路径建两条边，第一条容量只有 1，第二条容量 INF 但价值为 0。

代码

```
#include<queue>
#include<vector>
#include<string>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
/*====================*/
const int A = 4 + 10;
const int B = 6 + 10;
const int P = 15 + 10;
const int Q = 15 + 10;
/*====================*/
int a, b;
int p, q;
int G[P][Q][P][Q];
/*====================*/
struct S
{
```

```
    int k, x, y;
    S(int _k = 0, int _x = 0, int _y = 0)
    {
        k = _k, x = _x, y = _y;
    }
}s[A];
struct T
{
    int r, x, y;
    T(int _r = 0, int _x = 0, int _y = 0)
    {
        r = _r, x = _x, y = _y;
    }
}t[B];
/*====================*/
void Read(void)
{
    cin >> a >> b;
    cin >> p >> q;
    for (int i = 0; i <= p; ++i)
    {
        for (int j = 0; j + 1 <= q; ++j)
        {
            cin >> G[i][j][i][j + 1];
        }
    }
    for (int i = 0; i <= q; ++i)
    {
        for (int j = 0; j + 1 <= p; ++j)
        {
            cin >> G[j][i][j + 1][i];
        }
    }
    for (int i = 1; i <= a; ++i)
    {
        int k, x, y;
        cin >> k >> x >> y;
        s[i] = S(k, x, y);
    }
    for (int i = 1; i <= b; ++i)
    {
        int r, x, y;
        cin >> r >> x >> y;
        t[i] = T(r, x, y);
    }
}
/*====================*/
int Pos(int x, int y)
{
    return x * (q + 1) + y + 1;
}
/*====================*/
namespace _ZKW
{
    const int N = 258 + 10;
```

```
const int M = 970 + 10;
/*====================*/
const int INF = 0X7FFFFFFF;
/*====================*/
struct Edge
{
    int u, v, c, w;
    Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
    {
        u = _u, v = _v, c = _c, w = _w;
    }
};
/*====================*/
int n, m, s, t;
vector<int>G[N];
int mincost, maxflow;
int dep[N]; bool vis[N];
Edge edge[2 * M]; int cnt;
/*====================*/
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt++] = Edge(u, v, c, +w);
    edge[cnt++] = Edge(v, u, 0, -w);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool SPFA(void)
{
    for (int i = 1; i <= n; ++i)
    {
        vis[i] = false, dep[i] = INF;
    }
    vis[t] = true, dep[t] = 0;
    deque<int> q; q.push_back(t);
    while (!q.empty())
    {
        int x = q.front(); q.pop_front(), vis[x] = false;
        if (!q.empty() && dep[q.front()] > dep[q.back()])
        {
            swap(q.front(), q.back());
        }
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
            {
                dep[e1.v] = dep[x] - e1.w;
                if (!vis[e1.v])
                {
                    vis[e1.v] = true;
                    if (!q.empty() && dep[e1.v] < dep[q.front()])
                    {
                        q.push_front(e1.v);
```

```
                    }
                    else
                    {
                        q.push_back(e1.v);
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = (::p + 1) * (::q + 1) + 2; m = 0;
        s = (::p + 1) * (::q + 1) + 1; t = (::p + 1) * (::q + 1) + 2;
        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
```

```
        for (int i = 0; i <= ::p; ++i)
        {
            for (int j = 0; j + 1 <= ::q; ++j)
            {
                AddEdge(Pos(i, j), Pos(i, j + 1), 1, -::G[i][j][i][j + 1]);
                AddEdge(Pos(i, j), Pos(i, j + 1), INF, 0);
            }
        }
        for (int i = 0; i <= ::q; ++i)
        {
            for (int j = 0; j + 1 <= ::p; ++j)
            {
                AddEdge(Pos(j, i), Pos(j + 1, i), 1, -::G[j][i][j + 1][i]);
                AddEdge(Pos(j, i), Pos(j + 1, i), INF, 0);
            }
        }
        for (int i = 1; i <= ::a; ++i)
        {
            AddEdge(s, Pos(::s[i].x, ::s[i].y), ::s[i].k, 0);
        }
        for (int i = 1; i <= ::b; ++i)
        {
            AddEdge(Pos(::t[i].x, ::t[i].y), t, ::t[i].r, 0);
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    cout << -_ZKW::Init() << endl;
    return 0;
}
```

火星探险

题面

火星探险队的登陆舱将在火星表面着陆，登陆舱内有多部障碍物探测车。

登陆舱着陆后，探测车将离开登陆舱向先期到达的传送器方向移动。

探测车在移动中还必须采集岩石标本。

每一块岩石标本由最先遇到它的探测车完成采集。

每块岩石标本只能被采集一次。

岩石标本被采集后，其他探测车可以从原来岩石标本所在处通过。

探测车不能通过有障碍的地面。

本题限定探测车只能从登陆处沿着向南或向东的方向朝传送器移动，而且多个探测车可以在同一时间占据同一位置。

如果某个探测车在到达传送器以前不能继续前进，则该车所采集的岩石标本将全部损失。

用一个 $\mathrm{P} \times \mathrm{Q}$ 网格表示登陆舱与传送器之间的位置。登陆舱的位置在 (X_1, Y_1) 处，传送器 的位置在 (X_P, Y_Q) 处。 给定每个位置的状态，计算探测车的最优移动方案，使到达传送器的探测车的数量最多， 而且探测车采集到的岩石标本的数量最多。

X_1, Y_1	X_2, Y_1	X_3, Y_1	...	X_{P-1}, Y_1	X_P, Y_1
X_1, Y_2	X_2, Y_2	X_3, Y_2	...	X_{P-1}, Y_2	X_P, Y_2
X_1, Y_3	X_2, Y_3	X_3, Y_3	...	X_{P-1}, Y_3	X_P, Y_3
...			...		
X_1, Y_{Q-1}	X_2, Y_{Q-1}	X_3, Y_{Q-1}	...	X_{P-1}, Y_{Q-1}	X_P, Y_{Q-1}
X_1, Y_Q	X_2, Y_Q	X_3, Y_Q	...	X_{P-1}, Y_Q	X_P, Y_Q

题解

这题挺无聊的。

是深海机器人的加强版。有个坑点，这题要拆点。如果你是这么写的 $if(G[x_2][y_2] == 2)AddEdge((x_1, y_1), (x_2, y_2), 1, -1)$，那恭喜你写错了。因为 (x_2, y_2) 有两种抵达的方法，而点权放到边权上会导致二次使用。然后你就错了。

最后这个输出方案很麻烦，具体来说就是在残量网络上跑搜索。细节比较多，实现很麻烦，又臭又长。

代码

```
#include<queue>
#include<vector>
#include<string>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int P = 4e1 + 10;
const int Q = 4e1 + 10;
/*====================*/
const int dx[] = { 0,1 };
const int dy[] = { 1,0 };
/*====================*/
int car;
int p, q;
int G[P][Q];
/*====================*/
void Read(void)
{
    cin >> car;
    cin >> p >> q;
    for (int j = 1; j <= q; ++j)
    {
        for (int i = 1; i <= p; ++i)
        {
            cin >> G[i][j];
```

```
        }
    }
}
/*====================*/
int IPos(int x, int y)
{
    return (y - 1) * p + x;
}
int OPos(int x, int y)
{
    return p * q + (y - 1) * p + x;
}
/*====================*/
int id[P][Q][P][Q];
int Step[P][Q][P][Q];
/*====================*/
namespace _ZKW
{
    const int N = 2452 + 10;
    const int M = 4902 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w; int C;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w; C = _c;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    bool flag[2 * M];
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
        deque<int> q; q.push_back(t);
```

```
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
```

```
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = 2 * p * q + 2; m = 0;
        s = 2 * p * q + 1; t = 2 * p * q + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        AddEdge(s, IPos(1, 1), car, 0);
        for (int i = 1; i <= p; ++i)
        {
            for (int j = 1; j <= q; ++j)
            {
                if (::G[i][j] == 1)continue;
                AddEdge(IPos(i, j), OPos(i, j), INF, 0);
                if (::G[i][j] == 2)
                {
                    AddEdge(IPos(i, j), OPos(i, j), 1, -1);
                }
                for (int k = 0; k < 2; ++k)
                {
                    int _x = i + dx[k];
                    int _y = j + dy[k];
                    if (_x <= p && _y <= q)
                    {
                        if (::G[_x][_y] == 1)continue;
                        AddEdge(OPos(i, j), IPos(_x, _y), INF, 0);
                        id[i][j][_x][_y] = cnt - 2;
                    }
                }
            }
        }
        AddEdge(OPos(p, q), t, car, 0);

        return MinCost();
    }
}
using _ZKW::edge;
/*====================*/
int main()
{
    Read();
```

```
    _ZKW::Init();
    for (int x1 = 1; x1 <= p; ++x1)
    {
        for (int y1 = 1; y1 <= q; ++y1)
        {
            for (int x2 = 1; x2 <= p; ++x2)
            {
                for (int y2 = 1; y2 <= q; ++y2)
                {
                    if (id[x1][y1][x2][y2] != 0)
                    {
                        int idx = id[x1][y1][x2][y2];
                        Step[x1][y1][x2][y2] = edge[idx].c - edge[idx].c;
                    }
                }
            }
        }
    }
    for (int i = 1; i <= car; ++i)
    {
        int x = 1, y = 1;
        while (!(x == p && y == q))
        {
            if (Step[x][y][x + 1][y])
            {
                cout << i << " " << 1 << endl;
                Step[x][y][x + 1][y]--; x = x + 1;
            }
            else if (Step[x][y][x][y + 1])
            {
                cout << i << " " << 0 << endl;
                Step[x][y][x][y + 1]--; y = y + 1;
            }
        }
    }
    return 0;
}
```

最长 k 可重区间集

题面

给定实直线 L 上 n 个开区间组成的集合 I，和一个正整数 k，试设计一个算法，从开区间集合 I 中选取出开区间集合 $S \subseteq I$，使得在实直线 L 的任何一点 x，S 中包含点 x 的开区间个数不超过 k。且 $\sum_{z \in S} |z|$ 达到最大。这样的集合 S 称为开区间集合 I 的最长 k 可重区间集。$\sum_{z \in S} |z|$ 称为最长 k 可重区间集的长度。

对于给定的开区间集合 I 和正整数 k，计算开区间集合 I 的最长 k 可重区间集的长度。

题解

首先注意一下这是一条实数直线，也就是说存在浮点数。因为这题是开区间，所以每个端点只能被更大的区间覆盖掉。区间本身不会覆盖区间的端点。

很明显：一个区间可以覆盖若干个点，一个点可以被若干个区间覆盖。

有一个很直观的错误做法：源点连向每个区间，容量为 1；每个区间连向覆盖的点，容量为 1；每个点连向汇点，容量为 k。错误理由很简单，源点连向区间的容量为 1，那么这个流量该流到哪个覆盖点上?

改进一下这个错误做法，让它更错一点：把源点连向区间的容量设为区间长度。错误理由也很简单，不能保证源点到区间的边流满，也就是说不能限制住选择一个区间后覆盖整个区间。

接着往错里改：把每个数串起来，每个区间到该区间左端点的数连一条容量为 1 的边，每个区间右端点到汇点连一条容量为 1 的边。现在选择一个区间，可以通过串联的方式流过所有覆盖的点了，但是假如两个区间 $(1,4),(2,3)$，从 1 流入后会从 3 流出。流提前跑路了。

最后说正确做法，先把所有点串联起来，源点汇点先给连上容量 k 的边。表示每个点还可以被覆盖 k 次。如果存在一个区间 (l,r)，那就在 l,r 之间连一条容量为 1 的边，如果流经这条边了，就分走了一个流，那么可以被覆盖的次数就减一了。

记得离散化。

代码

```
#include<map>
#include<queue>
#include<vector>
#include<string>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
/*====================*/
const int N = 5e2 + 10;
/*====================*/
int n, k;
/*====================*/
struct Segment
{
    int l, r;
    Segment(int _l = 0, int _r = 0)
    {
        l = _l, r = _r;
        if (l > r)swap(l, r);
    }
}segment[N];
/*====================*/
void Read(void)
{
    cin >> n >> k;
    for (int i = 1; i <= n; ++i)
    {
        int l, r; cin >> l >> r;
        segment[i] = Segment(l, r);
```

```
    }
}
/*====================*/
vector<int>pos;
map<int, int>HASH;
/*====================*/
void GetHASH(void)
{
    for (int i = 1; i <= n; ++i)
    {
        pos.push_back(segment[i].l);
        pos.push_back(segment[i].r);
    }
    sort(pos.begin(), pos.end());
    pos.erase(unique(pos.begin(), pos.end()), pos.end());
    for (int i = 0; i < pos.size(); ++i)HASH[pos[i]] = i + 1;
}
/*====================*/
namespace _ZKW
{
    const int N = 1002 + 10;
    const int M = 1501 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
```

```
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
```

```
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = pos.size() + 2; m = 0;
        s = pos.size() + 1; t = pos.size() + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        AddEdge(s, 1, k, 0);
        for (int i = 2; i <= pos.size(); ++i)
        {
            AddEdge(i - 1, i, k, 0);
        }
        AddEdge(pos.size(), t, k, 0);

        for (int i = 1; i <= ::n; ++i)
        {
            int l = HASH[segment[i].l];
            int r = HASH[segment[i].r];
            AddEdge(l, r, 1, -(segment[i].r - segment[i].l));
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    GetHASH();
    int ans = -_ZKW::Init();
    cout << ans << endl;
    return 0;
}
```

最长 k 可重线段集

题面

给定平面 xoy上 n 个开线段组成的集合 I，和一个正整数 k,试设计一个算法。

从开线段集合 I 中选取出开线段集合 $\mathrm{S} \in \mathrm{I}$,

使得在x轴上的任何一点 p， S 中与直线 $\mathrm{x}=\mathrm{p}$ 相交的开线段个数不超过 k，且 $\sum_{z \in S}|z|$ 达到最大。

这样的集合 S 称为开线段集合 I 的最长 k 可重线段集的长度。

对于任何开线段 z，设其端点坐标为 (x_0, y_0) 和 (x_1, y_1)，

则开线段 z 的长度 $|\mathrm{z}|$ 定义为： $|z|=\lfloor\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}\rfloor$

对于给定的开线段集合 I 和正整数 k，计算开线段集合 I 的最长 k 可重线段集的长度。

题解

这题其实是最长 k 可重区间集的加强版。先把平面上的线段映射到x轴上。唯一有问题的就是如果一条线段与y轴平行，会被映射成一个点。最简单的方法就是对每个点拆点来表示。但是拆点太麻烦了，可以直接把坐标乘二。这样所有奇数位都空了下来。

$(x_i, x_i), (x_i, x_j), (x_j, x_j)$ 映射后就是 $(2*i, 2*i+1), (2*i, 2*j+1), (2*j, 2*j+1)$。我们发现原本不应该相交的三个区间映射后相交了，处理办法区分入和出。对两点不同的线段来说入加一，出减一。来抵消掉单点区间的影响。

即：$(2*i, 2*i+1), (2*i+1, 2*j), (2*j, 2*j+1)$。

代码

```
#include<map>
#include<cmath>
#include<queue>
#include<vector>
#include<string>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
/*====================*/
const int N = 5e2 + 10;
/*====================*/
int n, k;
/*====================*/
struct Segment
{
    int len;
    int l, r;
    int x1, y1;
    int x2, y2;
    Segment(int _x1 = 0, int _y1 = 0, int _x2 = 0, int _y2 = 0)
    {
        x1 = _x1, y1 = _y1;
        x2 = _x2, y2 = _y2;
        l = x1, r = x2; if (l > r)swap(l, r);
        len = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    }
```

```
}segment[N];
/*====================*/
void Read(void)
{
    cin >> n >> k;
    for (int i = 1; i <= n; ++i)
    {
        int x1, y1; cin >> x1 >> y1;
        int x2, y2; cin >> x2 >> y2;
        segment[i] = Segment(x1, y1, x2, y2);
    }
}
/*====================*/
vector<int>pos;
map<int, int>HASH;
/*====================*/
void GetHASH(void)
{
    for (int i = 1; i <= n; ++i)
    {
        pos.push_back(segment[i].l);
        pos.push_back(segment[i].r);
    }
    sort(pos.begin(), pos.end());
    pos.erase(unique(pos.begin(), pos.end()), pos.end());
    for (int i = 0; i < pos.size(); ++i)HASH[pos[i]] = i + 1;
}
/*====================*/
namespace _ZKW
{
    const int N = 2002 + 10;
    const int M = 2501 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
```

```
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
```

```
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        n = 2 * pos.size() + 2; m = 0;
        s = 2 * pos.size() + 1; t = 2 * pos.size() + 2;

        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }

        AddEdge(s, 1, k, 0);
        for (int i = 1; i <= pos.size(); ++i)
        {
            AddEdge(2 * i - 1, 2 * i, k, 0);
        }
        for (int i = 2; i <= pos.size(); ++i)
        {
            AddEdge(2 * i - 2, 2 * i - 1, k, 0);
        }
        AddEdge(2 * pos.size(), t, k, 0);

        for (int i = 1; i <= ::n; ++i)
        {
            int l = HASH[segment[i].l];
            int r = HASH[segment[i].r];
            if (l == r)
            {
                AddEdge(2 * l - 1, 2 * r, 1, -segment[i].len);
            }
            else
            {
                AddEdge(2 * l, 2 * r - 1, 1, -segment[i].len);
            }
```

```
        }

        return MinCost();
    }
}
/*====================*/
int main()
{
    Read();
    GetHASH();
    int ans = -_ZKW::Init();
    cout << ans << endl;
    return 0;
}
```

孤岛营救

题面

1944 年，特种兵麦克接到国防部的命令，要求立即赶赴太平洋上的一个孤岛，营救被敌军俘虏的大兵瑞恩。瑞恩被关押在一个迷宫里，迷宫地形复杂，但幸好麦克得到了迷宫的地形图。迷宫的外形是一个长方形，其南北方向被划分为 n 行，东西方向被划分为 m 列， 于是整个迷宫被划分为 $n \times m$ 个单元。每一个单元的位置可用一个有序数对 (单元的行号, 单元的列号) 来表示。南北或东西方向相邻的 2 个单元之间可能互通，也可能有一扇锁着的门，或者是一堵不可逾越的墙。迷宫中有一些单元存放着钥匙，并且所有的门被分成 p 类， 打开同一类的门的钥匙相同，不同类门的钥匙不同。

大兵瑞恩被关押在迷宫的东南角，即 (n, m) 单元里，并已经昏迷。迷宫只有一个入口， 在西北角。也就是说，麦克可以直接进入 $(1, 1)$ 单元。另外，麦克从一个单元移动到另一个 相邻单元的时间为 1，拿取所在单元的钥匙的时间以及用钥匙开门的时间可忽略不计。

试设计一个算法，帮助麦克以最快的方式到达瑞恩所在单元，营救大兵瑞恩。

题解

首先这不是一道网络流的题。

其次因为一把钥匙可以使用多次，最多有 10 种钥匙。所以可以直接状压跑分层图最短路。

需要注意的是图是无向图。为了方便处理可以把通路也当作一种特殊的门处理，只不过从起点开始就带着钥匙。

很裸的分层图最短路。因为数据范围不大所以SPFA应该也能过。具体实现细节看代码。

代码

```
#include<queue>
#include<vector>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e1 + 10;
const int M = 1e1 + 10;
/*====================*/
const int dx[] = { -1,1,0,0 };
```

```
const int dy[] = { 0,0,-1,1 };
/*====================*/
int n, m, p;//n行m列,p种钥匙
int key[N][M];//每个房间存的钥匙
int G[N][M][N][M];//存图
/*====================*/
int KEY(int k)
{
    return 1 << k;
}
/*====================*/
void Read(void)
{
    cin >> n >> m >> p;
    /*====================*/
    int k; cin >> k;
    for (int i = 1; i <= k; ++i)
    {
        int x1, y1; cin >> x1 >> y1;
        int x2, y2; cin >> x2 >> y2;
        int g; cin >> g;
        G[x1][y1][x2][y2] = (g == 0 ? -1 : g);
        G[x2][y2][x1][y1] = (g == 0 ? -1 : g);
    }
    /*====================*/
    int s; cin >> s;
    for (int i = 1; i <= s; ++i)
    {
        int x, y, q;
        cin >> x >> y >> q;
        key[x][y] |= KEY(q);
    }
    /*====================*/
    key[1][1] |= KEY(0);
}
/*====================*/
namespace _Dijkstra
{
    struct Pos
    {
        int k, x, y;
        Pos(int _k = 0, int _x = 0, int _y = 0)
        {
            k = _k, x = _x, y = _y;
        }
    };
    /*====================*/
    struct Unit
    {
        Pos v; int w;
        Unit(Pos _v = 0, int _w = 0)
        {
            v = _v, w = _w;
        }
        friend bool operator<(const Unit& a, const Unit& b)
```

```
        {
            return a.w > b.w;
        }
    };
    /*====================*/
    int dis[1 << 11][N][M];
    bool vis[1 << 11][N][M];
    /*====================*/
    int Init(void)
    {
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        priority_queue<Unit>q;
        q.push(Unit(Pos(key[1][1],1,1), dis[key[1][1]][1][1] = 0));
        while (!q.empty())
        {
            Pos cur = q.top().v; q.pop();
            int x = cur.x, y = cur.y, k = cur.k;
            if (vis[k][x][y])continue; vis[k][x][y] = true;
            for (int i = 0; i < 4; ++i)
            {
                int _x = x + dx[i], _y = y + dy[i], _k = k | key[_x][_y];
                if (_x<1 || _x>n || _y<1 || _y>m)continue;
                if ((G[x][y][_x][_y] != -1) && (k & KEY(G[x][y][_x][_y])))
                {
                    if (dis[_k][_x][_y] > dis[k][x][y] + 1)
                    {
                        q.push(Unit(Pos(_k, _x, _y), dis[_k][_x][_y] = dis[k][x]
[y] + 1));
                    }
                }
            }
        }
        int ans = 0X3F3F3F3F;
        for (int i = 0; i < (1 << (p + 1)); ++i)
        {
            ans = min(ans, dis[i][n][m]);
        }
        return ans == 0X3F3F3F3F ? -1 : ans;
    }
}
/*====================*/
int main()
{
    Read();
    cout << _Dijkstra::Init() << endl;
    return 0;
}
```

汽车加油行驶问题

题面

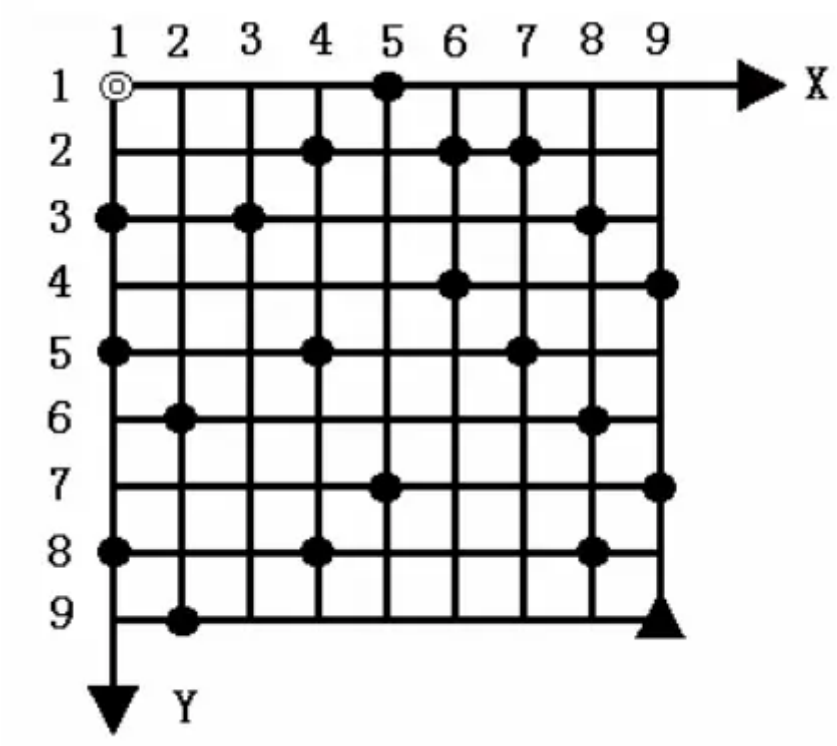
给定一个 $\mathrm{N} \times \mathrm{N}$ 的方形网格，设其左上角为起点◎，坐标为 $(1,1)$，X 轴向右为正，Y 轴向下为正，每个方格边长为 `1`，如图所示。

一辆汽车从起点◎出发驶向右下角终点▲，其坐标为 (N, N)。

在若干个网格交叉点处，设置了油库，可供汽车在行驶途中加油。汽车在行驶过程中应遵守如下规则:

- 汽车只能沿网格边行驶，装满油后能行驶 K 条网格边。出发时汽车已装满油，在起 点与终点处不设油库。
- 汽车经过一条网格边时，若其 X 坐标或 Y 坐标减小，则应付费用 B，否则免付费用。
- 汽车在行驶过程中遇油库则应加满油并付加油费用 A。
- 在需要时可在网格点处增设油库，并付增设油库费用 C (不含加油费用 A)。
- N , K , A , B , C 均为正整数， 且满足约束: $2 \leq \mathrm{N} \leq 100, 2 \leq \mathrm{K} \leq 10$。

设计一个算法，求出汽车从起点出发到达终点的一条所付费用最少的行驶路线。



题解

首先这题是分层最短路。其次这题可以费用流做。最后这题建议用分层最短路写。

有一个坑点，题目里说的可以在需要的时候设一个油库。这个油库是临时的，不是永久存在的。

你可以简单理解为一个空投要花你 $C + A$ 元。

然后跑一个裸的分层最短路，$dis_{k,x,y}$ 表示目前在坐标 (x, y) 上，还能走 k 条边。

注意一下如果遇到加油站强制消费这一点。

代码

```
#include<queue>
#include<vector>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int N = 1e2 + 10;
const int K = 1e1 + 10;
/*====================*/
const int dx[] = { -1,1,0,0 };
const int dy[] = { 0,0,-1,1 };
/*====================*/
int n, k, a, b, c;
/*====================*/
bool oil[N][N];
/*====================*/
void Read(void)
{
    cin >> n >> k >> a >> b >> c;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
        {
            cin >> oil[i][j];
        }
    }
}
/*====================*/
namespace _Dijkstra
{
    struct Unit
    {
        int k, x, y; int w;
        Unit(int _k = 0, int _x = 0, int _y = 0, int _w = 0)
        {
            k = _k, x = _x, y = _y, w = _w;
        }
        friend bool operator<(const Unit& a, const Unit& b)
        {
            return a.w > b.w;
        }
    };
    /*====================*/
    int dis[K][N][N]; bool vis[K][N][N];
    /*====================*/
    int Init(void)
    {
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        priority_queue<Unit>q;
        q.push(Unit(k, 1, 1, dis[k][1][1] = 0));
        while (!q.empty())
        {
```

```
            int curk = q.top().k;
            int curx = q.top().x;
            int cury = q.top().y; q.pop();
            if (!vis[curk][curx][cury])
            {
                vis[curk][curx][cury] = true;
                for (int i = 0; i < 4; ++i)
                {
                    int nxtx = curx + dx[i];
                    int nxty = cury + dy[i];
                    if (nxtx<1 || nxtx>n)continue;
                    if (nxty<1 || nxty>n)continue;
                    if (curk != 0)
                    {
                        int valu = 0;
                        int nxtk = curk - 1;
                        if (oil[nxtx][nxty])nxtk = k, valu += a;
                        if (nxtx < curx || nxty < cury)valu += b;
                        if (dis[nxtk][nxtx][nxty] > dis[curk][curx][cury] +
valu)
                        {
                            q.push(Unit(nxtk, nxtx, nxty, dis[nxtk][nxtx][nxty]
= dis[curk][curx][cury] + valu));
                        }
                    }
                    int valu = c + a;
                    int nxtk = k - 1;
                    if (oil[nxtx][nxty])nxtk = k, valu += a;
                    if (nxtx < curx || nxty < cury)valu += b;
                    if (dis[nxtk][nxtx][nxty] > dis[curk][curx][cury] + valu)
                    {
                        q.push(Unit(nxtk, nxtx, nxty, dis[nxtk][nxtx][nxty] =
dis[curk][curx][cury] + valu));
                    }
                }
            }
        }
        int res = 0X3F3F3F3F;
        for (int i = 0; i <= k; ++i)
        {
            res = min(res, dis[i][n][n]);
        }
        return res;
    }
}
/*====================*/
int main()
{
    Read();
    cout << _Dijkstra::Init() << endl;
    return 0;
}
```

软件补丁

题面

某公司发现其研制的一个软件中有 n 个错误，随即为该软件发放了一批共 m 个补丁程序。每一个补丁程序都有其特定的适用环境，某个补丁只有在软件中包含某些错误而同时又不包含另一些错误时才可以使用。一个补丁在排除某些错误的同时，往往会加入另一些错误。

换句话说，对于每一个补丁 i，都有 2 个与之相应的错误集合 $B_1(i)$ 和 $B_2(i)$，使得仅当软件包含 $B_1(i)$ 中的所有错误，而不包含 $B_2(i)$ 中的任何错误时，才可以使用补丁 i。补丁 i 将修复软件中的某些错误 $F_1(i)$，而同时加入另一些错误 $F_2(i)$。另外，每个补丁都耗费一定的时间。

试设计一个算法，利用公司提供的 m 个补丁程序将原软件修复成一个没有错误的软件，并使修复后的软件耗时最少。

题解

首先这不是网络流的题，其次我们注意到 n 在 20，所以我们状压一下错误的情况，然后跑SPFA就行了。至于为什么不跑Dijkstra，因为状态之间的转移可能会形成一个环。

代码

```
#define DEBUG
#include<queue>
#include<string>
#include<cstring>
#include<iostream>
using namespace std;
/*====================*/
const int N = 2e1 + 10;
const int M = 1e2 + 10;
/*====================*/
struct Patch
{
    int cost;
    int B1, B2;
    int F1, F2;
    Patch(int _cost = 0, int _B1 = 0, int _B2 = 0, int _F1 = 0, int _F2 = 0)
    {
        cost = _cost;
        B1 = _B1, B2 = _B2;
        F1 = _F1, F2 = _F2;
    }
};
/*====================*/
int n, m;
Patch patch[M];
/*====================*/
#ifdef DEBUG
void Put(int x)
{
    cout << x << ":";
    for (int i = n - 1; i >= 0; --i)
    {
        if (x & (1 << i))cout << 1;
        else cout << 0;
    }
```

```
    cout << endl;
}
#endif // DEBUG
/*====================*/
bool CheckI(int s, int d)
{
    return d == (s & d);
}
bool CheckO(int s, int d)
{
    return 0 == (s & d);
}
int Add(int s, int d)
{
    int res = 0;
    for (int i = 0; i < n; ++i)
    {
        if ((s & 1) || (d & 1))
        {
            res |= (1 << i);
        }
        s >>= 1, d >>= 1;
    }
    return res;
}
int Del(int s, int d)
{
    int res = 0;
    for (int i = 0; i < n; ++i)
    {
        if ((s & 1) && !(d & 1))
        {
            res |= (1 << i);
        }
        s >>= 1, d >>= 1;
    }
    return res;
}
/*====================*/
void Read(void)
{
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        int cost; cin >> cost;
        patch[i] = Patch(cost);
        string B, F; cin >> B >> F;
        for (int j = 0; j < n; ++j)
        {
            if (B[j] == '+')//B1
            {
                patch[i].B1 |= (1 << (n - j - 1));
            }
            if (B[j] == '-')//B2
            {
```

```
                patch[i].B2 |= (1 << (n - j - 1));
            }
            if (F[j] == '+')//F2
            {
                patch[i].F2 |= (1 << (n - j - 1));
            }
            if (F[j] == '-')//F1
            {
                patch[i].F1 |= (1 << (n - j - 1));
            }
        }
    }
}
/*====================*/
namespace _SPFA
{
    const int N = 1 << 20;
    /*====================*/
    const int INF = 0X3F3F3F3F;
    /*====================*/
    int dis[N]; bool vis[N];
    /*====================*/
    int Init(void)
    {
        int s = (1 << n) - 1, t = 0;
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        queue<int>q; dis[s] = 0; q.push(s);
        while (!q.empty())
        {
            int cur = q.front(); q.pop(); vis[cur] = false;
            for (int i = 1; i <= m; ++i)
            {
                if (CheckI(cur, patch[i].B1) && CheckO(cur, patch[i].B2))
                {
                    int val = patch[i].cost;
                    int nxt = cur; nxt = Del(nxt, patch[i].F1); nxt = Add(nxt,
patch[i].F2);
                    if (dis[nxt] > dis[cur] + val)
                    {
                        dis[nxt] = dis[cur] + val;
                        if (!vis[nxt])
                        {
                            q.push(nxt); vis[nxt] = true;
                        }
                    }
                }
            }
        }
        return dis[t] == INF ? 0 : dis[t];
    }
}
/*====================*/
int main()
{
```

```
    Read();
    cout << _SPFA::Init();
    return 0;
}
```

附录：网络流相关模板

最大流

ISAP

```
namespace _ISAP
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*======================*/
    const int INF = 0X7FFFFFFF;
    /*======================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*======================*/
    int pre[N];//路径前驱
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N], num[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*======================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*======================*/
    void BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = num[i] = 0;
        }
        queue<int>q; q.push(t);
        d[t] = 0; vis[t] = 1;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
```

```
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.u] && e.c > e.f)
                {
                    vis[e.u] = 1; d[e.u] = d[x] + 1; q.push(e.u);
                }
            }
        }
        for (int i = 0; i < n; ++i)num[d[i]]++;
    }
    int Augumemt(void)
    {
        int x, k = INF;
        x = t; while (x != s)
        {
            Edge& e = edge[pre[x]];
            k = min(k, e.c - e.f);
            x = edge[pre[x]].u;
        }
        x = t; while (x != s)
        {
            edge[pre[x]].f += k;
            edge[pre[x] ^ 1].f -= k;
            x = edge[pre[x]].u;
        }
        return k;
    }
    int MaxFlow(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            pre[i] = cur[i] = 0;
        }

        BFS(); int x = s, flow = 0;

        while (d[s] < n)
        {
            if (x == t)
            {
                flow += Augumemt(); x = s;
            }
            int flag = 0;
            for (int& i = cur[x]; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (e.c > e.f && d[x] == d[e.v] + 1)
                {
                    flag = 1; pre[e.v] = G[x][i]; x = e.v; break;
                }
            }
            if (!flag)
            {
                int l = n - 1;
                for (int i = 0; i < G[x].size(); ++i)
```

```
                {
                    Edge& e = edge[G[x][i]];
                    if (e.c > e.f)l = min(l, d[e.v]);
                }
                if (--num[d[x]] == 0)break;
                num[d[x] = l + 1]++; cur[x] = 0;
                if (x != s)x = edge[pre[x]].u;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            AddEdge(u, v, c);
        }
        return MaxFlow();
    }
};
```

HLPP

```
namespace _HLPP
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int next, v, c;
        Edge(int _next = 0, int _v = 0, int _c = 0)
        {
            next = _next, v = _v, c = _c;
        }
    };
    /*====================*/
    int n, m, s, t;
    int d[N], num[N];
    stack<int> lib[N];
    int ex[N], level = 0;
    Edge edge[2 * M]; int head[N], cnt;
    /*====================*/
    void AddEdge(int u, int v, int c)
```

```
    {
        edge[cnt] = Edge(head[u], v, c), head[u] = cnt++;
        edge[cnt] = Edge(head[v], u, 0), head[v] = cnt++;
    }
    /*====================*/
    int Push(int u)
    {
        bool init = u == s;
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            const int& v = edge[i].v, & c = edge[i].c;
            if (!c || init == false && d[u] != d[v] + 1)continue;
            int k = init ? c : min(c, ex[u]);
            if (v != s && v != t && !ex[v]) lib[d[v]].push(v), level =
max(level, d[v]);
            ex[u] -= k, ex[v] += k, edge[i].c -= k, edge[i ^ 1].c += k;
            if (!ex[u]) return 0;
        }
        return 1;
    }
    void Relabel(int x)
    {
        d[x] = INF;
        for (int i = head[x]; i != -1; i = edge[i].next)
        {
            if (edge[i].c) d[x] = min(d[x], d[edge[i].v]);
        }
        if (++d[x] < n)
        {
            lib[d[x]].push(x); level = max(level, d[x]); ++num[d[x]];
        }
    }
    bool BFS(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            d[i] = INF; num[i] = 0;
        }
        queue<int>q; q.push(t), d[t] = 0;
        while (!q.empty())
        {
            int u = q.front(); q.pop(); num[d[u]]++;
            for (int i = head[u]; i!=-1; i = edge[i].next)
            {
                const int& v = edge[i].v;
                if (edge[i ^ 1].c && d[v] > d[u] + 1) d[v] = d[u] + 1,
q.push(v);
            }
        }
        return d[s] != INF;
    }
    int Select(void)
    {
        while (lib[level].size() == 0 && level > -1) level--;
        return level == -1 ? 0 : lib[level].top();
```

```
    }
    int MaxFlow(void)
    {
        if (!BFS()) return 0;
        d[s] = n; Push(s); int x;
        while (x = Select())
        {
            lib[level].pop();
            if (Push(x))
            {
                if (!--num[d[x]])
                {
                    for (int i = 1; i <= n; ++i)
                    {
                        if (i != s && i != t && d[i] > d[x] && d[i] < n + 1)
                        {
                            d[i] = n + 1;
                        }
                    }
                }
                Relabel(x);
            }
        }
        return ex[t];
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        memset(head, -1, sizeof(head));
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            AddEdge(u, v, c);
        }
        return MaxFlow();
    }
}
```

Dinic

```
namespace _Dinic
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
```

```
        u = _u, v = _v, c = _c, f = _f;
    }
};
/*====================*/
int cur[N];//当前弧优化
int n, m, s, t;//点，边，源，汇
vector<int>G[N];//邻接表
int d[N], vis[N];//图分层
Edge edge[2 * M]; int cnt;//边
/*====================*/
void AddEdge(int u, int v, int c)
{
    edge[cnt++] = Edge(u, v, c, 0);
    edge[cnt++] = Edge(v, u, 0, 0);
    G[u].push_back(cnt - 2);
    G[v].push_back(cnt - 1);
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = vis[i] = 0;
    }
    queue<int>q; q.push(s);
    d[s] = 0; vis[s] = 1;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (!vis[e.v] && e.c > e.f)
            {
                vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return vis[t];
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
```

```
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            AddEdge(u, v, c);
        }
        return MaxFlow();
    }
}
```

Dinic-Scaling

```
namespace _Dinic
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
        friend bool operator<(const Edge& a, const Edge& b)
        {
            return a.c > b.c;
        }
    };
    /*====================*/
    int d[N];//图分层
```

```
int cur[N];//当前弧优化
Edge _edge[M];//即将加入流网络的边
int n, m, s, t;//点，边，源，汇
vector<int>G[N];//邻接表
Edge edge[2 * M]; int cnt;//边
/*====================*/
void AddEdge(int u, int v, int c)
{
    edge[cnt++] = Edge(u, v, c, 0);
    edge[cnt++] = Edge(v, u, 0, 0);
    G[u].push_back(cnt - 2);
}
/*====================*/
bool BFS(void)
{
    for (int i = 0; i <= n; ++i)
    {
        d[i] = INF;
    }
    queue<int>q; q.push(s); d[s] = 0;
    while (!q.empty())
    {
        int x = q.front(); q.pop();
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[e.v] >= INF && e.c > e.f)
            {
                d[e.v] = d[x] + 1; q.push(e.v);
            }
        }
    }
    return d[t] < INF;
}
int DFS(int x, int k)
{
    int flow = 0, f;
    if (x == t || k == 0) return k;
    for (int& i = cur[x]; i < G[x].size(); ++i)
    {
        Edge& e = edge[G[x][i]];
        if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) > 0)
        {
            e.f += f; edge[G[x][i] ^ 1].f -= f;
            flow += f; k -= f; if (k == 0) break;
        }
    }
    return flow;
}
int Dinic(void)
{
    int flow = 0;
    while (BFS())
    {
        flow += DFS(s, INF);
```

```
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        sort(_edge, _edge + m);
        for (int type : {0, 1})
        {
            for (int p = 1 << 30, i = 0; p; p /= 2)
            {
                while (i < m && _edge[i].c >= p)
                {
                    if (type == 0)AddEdge(_edge[i].u, _edge[i].v, _edge[i].c);
                    if (type == 1)G[_edge[i].v].push_back(i * 2 + 1); i++;
                }
                flow += Dinic();
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 0; i < m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            _edge[i] = Edge(u, v, c);
        }
        return MaxFlow();
    }
}
```

费用流

EK

```
namespace _EK
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X3F3F3F3F;
    /*====================*/
```

```
struct Edge
{
    int next, v, c, w;
    Edge(int _next = 0, int _v = 0, int _c = 0, int _w = 0)
    {
        next = _next, v = _v, c = _c, w = _w;
    }
};
/*====================*/
int n, m, s, t;
int maxflow, mincost;
Edge edge[2 * M]; int head[N], cnt;
int dis[N], pre[N], incf[N]; bool vis[N];
/*====================*/
void AddEdge(int u, int v, int c, int w)
{
    edge[cnt] = Edge(head[u], v, c, +w); head[u] = cnt++;
    edge[cnt] = Edge(head[v], u, 0, -w); head[v] = cnt++;
}
/*====================*/
bool SPFA(void)
{
    memset(dis, 0X3F, sizeof(dis));
    queue<int> q; q.push(s);
    dis[s] = 0, incf[s] = INF, incf[t] = 0;
    while (!q.empty())
    {
        int u = q.front(); q.pop(); vis[u] = false;
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].v, c = edge[i].c, w = edge[i].w;
            if (!c || dis[v] <= dis[u] + w) continue;
            dis[v] = dis[u] + w, incf[v] = min(c, incf[u]), pre[v] = i;
            if (!vis[v])q.push(v), vis[v] = true;
        }
    }
    return incf[t];
}
int MinCost(void)
{
    while (SPFA())
    {
        maxflow += incf[t];
        for (int u = t; u != s; u = edge[pre[u] ^ 1].v)
        {
            edge[pre[u]].c -= incf[t];
            edge[pre[u] ^ 1].c += incf[t];
            mincost += incf[t] * edge[pre[u]].w;
        }
    }
    return mincost;
}
/*====================*/
int Init(void)
{
```

```
        cin >> n >> m >> s >> t;
        mincost = maxflow = cnt = 0;
        memset(head, -1, sizeof(head));
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c, w;
            cin >> u >> v >> c >> w;
            AddEdge(u, v, c, w);
        }
        return MinCost();
    }
}
```

ZKW费用流

```
namespace _ZKW
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    vector<int>G[N];
    int mincost, maxflow;
    int dep[N]; bool vis[N];
    Edge edge[2 * M]; int cnt;
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt++] = Edge(u, v, c, +w);
        edge[cnt++] = Edge(v, u, 0, -w);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    bool SPFA(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            vis[i] = false, dep[i] = INF;
        }
        vis[t] = true, dep[t] = 0;
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
```

```
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dep[q.front()] > dep[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dep[e1.v] > dep[x] - e1.w)
                {
                    dep[e1.v] = dep[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dep[e1.v] < dep[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dep[s] < INF;
    }
    int DFS(int x, int k)
    {
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dep[x] - e1.w == dep[e1.v] && (f = DFS(e1.v, min(k, e1.c))) > 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                for (int i = 1; i <= n; ++i)
                {
```

```
                    vis[i] = false;
                }
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
    /*======================*/
    int Init(void)
    {
        cin >> n >> m >> s >> t;
        mincost = maxflow = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c, w;
            cin >> u >> v >> c >> w;
            AddEdge(u, v, c, w);
        }
        return MinCost();
    }
}
```