图论 3

网络流经典例题

方格取数问题

洛谷p2774

题意

给定一个 $n\times m$的方格图，每个方格中都有一个正整数。

现要从方格中取数，使任意两个数所在方格没有公共边，且取出的数的总和最大，请求出最大的和。

题解

由于每个方格限制较少，故考虑利用限制建边。

考虑先取所有方格，再删去权值和最小的一组方格，使得剩下的方格没有公共边。

问题转化为建立一个模型，支持以下操作：

- 支持删除元素，且删除代价为元素点权
- 保证操作最优或者操作可逆
- 最终状态为剩余元素没有冲突

发现可以将方格进行黑白二染色，显然同色方格间没有限制不连边，所以可以得到二分图。

从源点向每个黑格连一条边，容量为黑格点权。如果删去该边，表示删除黑格。

从白格向汇点连一条边，容量为白格点权。如果删去该边，表示删除白格。

最后从黑格向有冲突的白格连一条边，容量待定。

问题转化为最小割问题。

事实上网络流图不连通等价于没有从源点到黑格，再经过冲突边到白格，最后到汇点的路径。

这又等价于不会同时选择冲突的黑格和白格，即最终的目的。

然后需要保证最小割只选择源点到黑格的边和白格到汇点的边，所以把黑格向有冲突的白格连的边容量设为 $\inf$

最后求最小割，答案为点权总和减去最小割。

```
const int MAXS=105,MAXN=MAXS*MAXS,MAXM=6*MAXN,Inf=0x7fffffff;
const int dir[][2]={{0,1},{0,-1},{1,0},{-1,0}};
struct Edge{
```

```
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
```

```
        }
        return flow;
    }
    int Maxflow(int s,int t){
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
int n,m,a[MAXS][MAXS];
int Id(int r,int c){
    return (r-1)*m+c;
}
bool in(int r,int c){return r>0&&r<=n&&c>0&&c<=m;}
int main()
{
    n=read_int(),m=read_int();
    int sum=0,s=Id(n,m)+1,t=s+1;
    Clear();
    _rep(i,1,n)
    _rep(j,1,m){
        a[i][j]=read_int(),sum+=a[i][j];
        if((i+j)&1)
        Insert(Id(i,j),t,a[i][j]);
        else{
            Insert(s,Id(i,j),a[i][j]);
            _for(k,0,4){
                int ti=i+dir[k][0],tj=j+dir[k][1];
                if(in(ti,tj))
                Insert(Id(i,j),Id(ti,tj),Inf);
            }
        }
    }
    sum-=solver.Maxflow(s,t);
    enter(sum);
    return 0;
}
```

练习题

洛谷p2762

题意

给定 n 个可选择实验□m 个实验器材。完成第 i 个实验可以得到 p_i 元奖金，购买第 j 个实验

器材需要花费 c_j 元。

完成每个实验需要若干个器材(允许多个实验共用一个器材)，问最大利益及购买方案。

题解

先收取所有实验的奖金。然后考虑要么舍弃实验奖金，要么购买相应器材。

剩下思路类似方格取数。

```
const int MAXN=200,MAXM=3000,Inf=0x7fffffff;
struct Edge{
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
```

```
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        return flow;
    }
    int Maxflow(int s,int t){
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
char buf[MAXN];
bool vis[MAXN];
void dfs(int u){
    vis[u]=true;
    for(int i=head[u];~i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||edge[i].cap==0)
        continue;
        dfs(v);
    }
}
int main()
{
    Clear();
    int n=read_int(),m=read_int(),s=n+m+1,t=s+1;
    int c,pos,p,sum=0;
    _rep(i,1,n){
        sum+=c=read_int();
        Insert(s,i,c);
        gets(buf);
        pos=0;
        while(sscanf(buf+pos,"%d",&p)==1){
            Insert(i,p+n,Inf);
            if(p==0)
            pos++;
```

```
            else{
                while(p){
                    p/=10;
                    pos++;
                }
            }
            while(buf[pos]==' ')
            pos++;
        }
    }
    _rep(i,1,m)
    Insert(i+n,t,read_int());
    int flow=solver.Maxflow(s,t);
    dfs(s);
    _rep(i,1,n) if(vis[i])
    space(i);
    puts("");
    _rep(i,1,m) if(vis[i+n])
    space(i);
    puts("");
    enter(sum-flow);
    return 0;
}
```

最大权闭合子图

定义

给定一个有向图，取图中的一些点构成点集 V□若对 V 中的任意一个节点其后继节点均属于 V□则称 V 及其相关边为闭合子图。

如果每个点拥有一个点权，则称 V 点权和最大的闭合子图为最大权闭合子图。

性质

按如下规则构图：

- 若某点点权为正，则从源点连一条容量等于点权的边到该点
- 若某点点权为负，则从该点连一条容量等于点权绝对值的边到汇点
- 如某条边属于原图，则将其容量改为 $\inf$

在该图中跑最大流，则最大权闭合子图点权和 $=$ 所有正点权之和 $-$ 最小割。

证明

简单割定义：割 (S,T) 中每一条割边都与 s 或者 t 相连。

任意闭合子图一定对应一个简单割，否则取闭合子图和源点构成 S其余点构成 T

如果有 $(u,v)\subset E,u\in S,v\in T$则表示选择 u但不选择 v与闭合子图定义矛盾。

任意一个简单割对应一个闭合子图，因为对 $(u,v)\subset E,u\in S$根据简单割定义，有 $v\in S$满足闭合子图定义。

记 T 中所有正权点的权值之和为 $T_1$$S$ 中所有正权点的权值之和为 $S_1$$T$ 中所有负权点的权值绝对值之和为 S_2

则割的容量 $C(S,T)=T_1+S_2$闭合子图的权值 $W=S_1-S_2$

则有 $C(S,T)+W=T_1+S_1$即 $W=T_1+S_1-C(S,T)$

T_1+S_1 即为所有正点权之和，是定值。剩下只需要考虑令 $C(S,T)$ 尽量小。

而按上述方法构建的网络流图的最小割一定为简单割，因为非简单割容量大于 $\inf$一定不是最小割。

所以直接求最小割即可，证毕。

最小路径覆盖问题

洛谷p2764

题意

给定一个 DAG求最小的不相交路径集，使得每个点恰好属于其中一条路径。(允许路径长度为 0，此时路径仅覆盖一个点)

题解

先用 n 条长度为 0 的路径覆盖 n 个点，然后考虑合并路径。

将每个点拆成入点和出点，易知每个入点/出点最多被合并一次，否则有两条路径相交。

从源点连一条容量为 1 的边到每个入点，从每个出点连一条容量为 1 的边到汇点，则可以满足上述的合并次数限制。

最后，对每条边，从入点连一条容量为 1 的边到出点，表示以入点结束的路径可与以出点开始的路径合并一次。

最后需要合并次数最多，跑最大流即可。最小路径覆盖为节点数减去最大流。

关于路径打印，只要在求完最大流后跑残量网络即可。

```
const int MAXN=305,MAXM=18005,Inf=0x7fffffff;
struct Edge{
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
```

```
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        return flow;
    }
```

```
    int Maxflow(int s,int t){
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
int Next[MAXN],Pre[MAXN];
int main()
{
    int n=read_int(),m=read_int(),u,v;
    int s=2*n+1,t=2*n+2;
    Clear();
    _rep(i,1,n){
        Insert(s,i,1);
        Insert(i+n,t,1);
    }
    while(m--){
        u=read_int(),v=read_int();
        Insert(u,v+n,1);
    }
    int flow=solver.Maxflow(s,t);
    _rep(u,1,n){
        for(int i=head[u];~i;i=edge[i].next){
            int v=edge[i].to;
            if(v!=s&&edge[i].cap==0){
                Next[u]=v-n;
                Pre[v-n]=u;
            }
        }
    }
    _rep(i,1,n){
        if(!Pre[i]){
            int pos=i;
            while(pos){
                space(pos);
                pos=Next[pos];
            }
            puts("");
        }
    }
    enter(n-flow);
    return 0;
}
```

练习题

洛谷p2765

题意

给定 n 个栈，要求依次放入 $1,2,3,\cdots$

要求同一个栈中每两个相邻元素之和是平方数。

输出最多可以放入的数的个数以及任意一个方案。

题解

把原图想像成一个 DAG□小数向可以与它相加构成平方数的大数连一条单向边，需要栈的个数等于最小路径覆盖。

依次向图中加入每个数，每次加入一个数时先用长度为 0 的路径将其覆盖，然后在残量网络中考虑路径合并。

残量网络流量为 0 表示路径合并失败，需要使用一个新的栈。如果最小路径覆盖不超过 n 即可继续加数，否则终止循环。

方案打印同最小路径覆盖。

```
const int MAXN=1e5+5,MAXM=2e5+5,Inf=0x7fffffff;
struct Edge{
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
```

```
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        return flow;
    }
    int Maxflow(int s,int t){
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
const int MAXS=60;
int Next[MAXN],Pre[MAXN],s=MAXN-2,t=MAXN-1;
int main()
{
    Clear();
    int n=read_int(),pos=0,num=0;
    while(pos<=n){
        ++num;
        Insert(s,num<<1,1);Insert(num<<1|1,t,1);
        for(int i=sqrt(num)+1;i*i<2*num;i++)
        Insert((i*i-num)<<1,num<<1|1,1);
        if(solver.Maxflow(s,t)==0)
        pos++;
    }
    enter(--num);
```

```
    _rep(u,1,num){
        for(int i=head[u<<1];~i;i=edge[i].next){
            int v=edge[i].to;
            if(v!=s&&edge[i].cap==0){
                Next[u]=v>>1;
                Pre[v>>1]=u;
            }
        }
    }
    _rep(i,1,num){
        if(!Pre[i]){
            int pos=i;
            while(pos){
                space(pos);
                pos=Next[pos];
            }
            puts("");
        }
    }
    return 0;
}
```

往返路线问题

洛谷p2770

题意

给定 n 个点□m 条边。规定 1 号点为起点□n 号点为终点。

要求输出一条从起点经过终点再回到起点的路径，要求路径上的节点编号先单调增，再单调减，且路径上无重复节点。

如果存在多条路径，输出任意一条最长的路径。

题解 1

首先问题可以转化为从起点开始走两条不相交的路径。

每个点可以走一次且记一次贡献，所以考虑拆点并连一条容量为 1 权值为 -1 的边。

注意起点终点会走两遍，但贡献只计算一次，所以需要额外连一条容量为 1 权值为 0 的边。

然后从汇点连一条容量为 2 的边到起点，从终点连一条容量为 2 的边到汇点，保证存在两条路径。

最后根据输入的边连接网络流的相应点即可，注意这些边不一定只会走一次(例如 $n=2$ 且起点终点连通的情况)，所以边的容量设为 2。

```
const int MAXN=205,MAXM=6000,Inf=0x7fffffff;
struct Edge{
    int to,cap,w,next;
    Edge(int to=0,int cap=0,int w=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->w=w;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int w,int c){
    edge[edge_cnt]=Edge(v,c,w,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,-w,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],dis[MAXN];
    bool inque[MAXN];
    bool spfa(){
        mem(dis,127/3);
        queue<int>q;
        q.push(s);
        inque[s]=true,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[u]+edge[i].w<dis[v]&&edge[i].cap){
                    dis[v]=dis[u]+edge[i].w,pos[v]=head[v];
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return dis[t]!=dis[0];
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        inque[u]=true;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
```

```
if(!inque[v]&&dis[u]+edge[i].w==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[
i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        inque[u]=false;
        return flow;
    }
    void MCMF(int s,int t,int &flow,LL &cost){
        this->s=s;this->t=t;
        cost=flow=0;
        int temp_flow;
        mem(inque,0);
        while(spfa()){
            temp_flow=dfs(s,Inf);
            flow+=temp_flow;
            cost+=1LL*temp_flow*dis[t];
        }
    }
}solver;
map<string,int>mp;
map<int,string>pt;
string a,b;
int n,m,s,t;
bool vis[MAXN];
void dfs1(int u){
    if(u==t)
    return;
    cout<<pt[u]<<endl;
    for(int i=head[u+n];~i;i=edge[i].next){
        int v=edge[i].to;
        if(u<v&&edge[i].cap<2){
            vis[v]=true;
            return dfs1(v);
        }
    }
}
void dfs2(int u){
    for(int i=head[u+n];~i;i=edge[i].next){
        int v=edge[i].to;
        if(!vis[v]&&u<v&&edge[i].cap<2){
            dfs2(v);
            break;
        }
    }
```

```
    cout<<pt[u]<<endl;
}
int main()
{
    n=read_int(),m=read_int(),s=2*n+1,t=2*n+2;
    Clear();
    _rep(i,1,n){
        cin>>a;
        mp[a]=i;pt[i]=a;
        Insert(i,i+n,-1,1);
    }
    Insert(s,1,0,2);Insert(1,1+n,0,1);
    Insert(n,2*n,0,1);Insert(2*n,t,0,2);
    int u,v;
    while(m--){
        cin>>a>>b;
        u=mp[a],v=mp[b];
        if(u>v)
        swap(u,v);
        Insert(u+n,v,0,2);
    }
    int flow;LL cost;
    solver.MCMF(s,t,flow,cost);
    if(flow<2)
    puts("No Solution!");
    else{
        enter(-cost);
        dfs1(1);
        dfs2(1);
    }
    return 0;
}
```

题解 2

发现可以 dp 解决，时间复杂度 $O(nm)$

```
const int MAXN=105,MAXM=6000;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int dp[MAXN][MAXN];
pair<int,int> pre[MAXN][MAXN];
map<string,int>mp;
```

```
map<int,string>pt;
string a,b;
int n,m;
void dfs1(int pos1,int pos2){
    if(pos1==1){
        cout<<pt[pos1]<<endl;
        return;
    }
    while(pre[pos1][pos2].first==pos1)
    pos2=pre[pos1][pos2].second;
    dfs1(pre[pos1][pos2].first,pos2);
    cout<<pt[pos1]<<endl;
}
void dfs2(int pos1,int pos2){
    if(pos2==1){
        cout<<pt[pos2]<<endl;
        return;
    }
    if(pos2!=n)
    cout<<pt[pos2]<<endl;
    while(pre[pos1][pos2].second==pos2)
    pos1=pre[pos1][pos2].first;
    dfs2(pos1,pre[pos1][pos2].second);
}
int main()
{
    n=read_int(),m=read_int();
    _rep(i,1,n){
        cin>>a;
        mp[a]=i;pt[i]=a;
    }
    int u,v;
    while(m--){
        cin>>a>>b;
        u=mp[a],v=mp[b];
        if(u>v)
        swap(u,v);
        Insert(u,v);
    }
    mem(dp,-1);
    dp[1][1]=0;
    _rep(i,1,n)
    _rep(j,1,n){
        if(dp[i][j]==-1)
        continue;
        for(int k=head[i];k;k=edge[k].next){
            int v=edge[k].to;
            if(v<=j&&v!=n)
            continue;
            if(dp[v][j]<dp[i][j]+1){
```

```
                dp[v][j]=dp[i][j]+1;
                pre[v][j]=make_pair(i,j);
            }
        }
        for(int k=head[j];k;k=edge[k].next){
            int v=edge[k].to;
            if(v<=i&&v!=n)
            continue;
            if(dp[i][v]<dp[i][j]+1){
                dp[i][v]=dp[i][j]+1;
                pre[i][v]=make_pair(i,j);
            }
        }
    }
    if(dp[n][n]==dp[0][0])
    puts("No Solution!");
    else{
        enter(dp[n][n]);
        dfs1(n,n);
        dfs2(n,n);
    }
    return 0;
}
```

最长 k 可重区间集问题

洛谷p3358

题意

给定 n 个开区间，要求选取若干个区间，使得 X 轴上每个点最多被覆盖 k 次。

问在所有可能方案中，选取区间长度和最大值为多少。

题解

很自然想到每个点最多覆盖 k 次可以理解为 X 轴上流量最大为 k而长度和最大可以理解为费用流。

首先考虑到费用流没有点权，考虑将一个区间拆成左端点和右端点，两端点间连一条容量为 1，费用为区间长度相反数的边。

没有相交的区间不存在约束，考虑在靠左区间的右端点向靠右区间的左端点间连一条容量无穷大且费用为 0 的边，表示可以同时选择。

而有相加区间的之间则不连边，使得两个区间竞争来自源点的流量，相互约束。

然而按上述方案连边，边的数量可能达到 $O(n^2)$难以接受。

考虑将点离散化，分配到 X 轴，在 X 轴所有相邻点间连一条容量为无穷大(其实 k 就够了)，费用为 0 边。

对每个区间，仍然按上述方案连边。最后从源点连一条容量为 k 的边到最左端点，从最右端点连一条容量为 k 的边到汇点，跑费用流即可。

如果题目把开区间改成闭区间，拆点的时候当作 $[lef,rig+1)$ 处理就行。

```
const int MAXN=1005,MAXM=2005,Inf=0x7fffffff;
struct Edge{
    int to,cap,w,next;
    Edge(int to=0,int cap=0,int w=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->w=w;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int w,int c){
    edge[edge_cnt]=Edge(v,c,w,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,-w,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],dis[MAXN];
    bool inque[MAXN];
    bool spfa(){
        mem(dis,127/3);
        queue<int>q;
        q.push(s);
        inque[s]=true,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[u]+edge[i].w<dis[v]&&edge[i].cap){
                    dis[v]=dis[u]+edge[i].w,pos[v]=head[v];
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return dis[t]!=dis[0];
    }
    int dfs(int u,int max_flow){
```

```
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        inque[u]=true;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(!inque[v]&&dis[u]+edge[i].w==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[
i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        inque[u]=false;
        return flow;
    }
    void MCMF(int s,int t,int &flow,LL &cost){
        this->s=s;this->t=t;
        cost=flow=0;
        int temp_flow;
        mem(inque,0);
        while(spfa()){
            temp_flow=dfs(s,Inf);
            flow+=temp_flow;
            cost+=1LL*temp_flow*dis[t];
        }
    }
}solver;
int x[MAXN],lef[MAXN],rig[MAXN],len[MAXN];
int main()
{
    Clear();
    int n=read_int(),k=read_int(),m;
    _for(i,0,n){
        lef[i]=read_int(),rig[i]=read_int(),len[i]=rig[i]-lef[i];
        x[m++]=lef[i],x[m++]=rig[i];
    }
    sort(x,x+m);
    m=unique(x,x+m)-x;
    int s=m+1,t=m+2;
    _for(i,1,m)
    Insert(i,i+1,0,k);
    Insert(s,1,0,k);Insert(m,t,0,k);
    _for(i,0,n){
        lef[i]=lower_bound(x,x+m,lef[i])-x+1;
        rig[i]=lower_bound(x,x+m,rig[i])-x+1;
        Insert(lef[i],rig[i],-len[i],1);
    }
```

```
    int flow;LL cost;
    solver.MCMF(s,t,flow,cost);
    enter(-cost);
    return 0;
}
```

最长不下降子序列问题

洛谷p2766

题意

给定一个正整数序列 $x_1,x_2,\cdots x_n$。

1. 计算其最长不下降子序列的长度 s
2. 如果每个元素只允许使用一次，计算从给定的序列中最多可取出多少个长度为 s 的不下降子序列
3. 如果允许在取出的序列中多次使用 x_1,x_n则从给定序列中最多可取出多少个不同的长度为的 s 不下降子序列

题解

第一问考虑 dp 求出以第 i 个数结尾的最长序列长度。

第二问先考虑只允许使用一次这个限制条件，只需要把每个点拆成入点和出点，然后连一条容量为 1 的边即可。

接下若满足 $j\lt i,a_j\le a_i,\text{dp}_j+1==\text{dp}_i$则连一条 $j\to i$ 的容量为 1 的边。

最后考虑从源点向 $\text{dp}_i==1$ 的点连一条容量为 1 的边，从 $\text{dp}_i==s$ 的点向汇点连一条容量为 1 的边。

最后跑最大流即可求出第二问答案。

对第三问，考虑把第 $1,n$ 个点的入点和出点的连边容量改为 $\inf$同时把从源点到第 1 个的连边的容量改为 $\inf$。

如果 $\text{dp}_n==s$还需要把从第 n 个点向汇点的连边的容量改为 $\inf$然后重新跑一遍最大流。

但实际编程中不需要修改边，额外添加边即可。同时也不需要重新跑最大流，在残量网络上跑最大流然后累加上第二问答案即可。

需要注意 $s=1$ 的特例，题目需要求不同的不下降子序列就是为了防止此时答案为 $\inf$ 的情况。

```
const int MAXN=1005,MAXM=5e5+5,Inf=0x7fffffff;
struct Edge{
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
```

```
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        return flow;
    }
    int Maxflow(int s,int t){
```

```
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
int a[MAXN],dp[MAXN],f[MAXN],pos;
int main()
{
    int n=read_int();
    _rep(i,1,n)
    a[i]=read_int();
    f[pos=0]=-Inf;
    _rep(i,1,n){
        if(a[i]>=f[pos]){
            f[++pos]=a[i];
            dp[i]=pos;
        }
        else{
            int t=upper_bound(f,f+pos+1,a[i])-f;
            f[t]=a[i];
            dp[i]=t;
        }
    }
    enter(pos);
    if(pos==1){
        sort(a+1,a+n+1);
        int ans=unique(a+1,a+n+1)-a-1;
        enter(ans);
        enter(ans);
        return 0;
    }
    int s=2*n+1,t=2*n+2;
    Clear();
    _rep(i,1,n){
        Insert(i,i+n,1);
        if(dp[i]==1)
        Insert(s,i,1);
        if(dp[i]==pos)
        Insert(i+n,t,1);
        _for(j,1,i){
            if(a[j]<=a[i]&&dp[j]+1==dp[i])
            Insert(j+n,i,1);
        }
    }
    int ans=solver.Maxflow(s,t);
    enter(ans);
    Insert(s,1,Inf);Insert(1,1+n,Inf);
```

```
    if(dp[n]==pos){
        Insert(n,2*n,Inf);
        Insert(2*n,t,Inf);
    }
    ans+=solver.Maxflow(s,t);
    enter(ans);
    return 0;
}
```

星际转移问题

洛谷p2754

题意

一共 n 个点□m 条船。每条船有一定容量，并且存在周期性环形航线。

设船从航线中的一个点移动到另一个点花费 1 个单位的时间，问最少要花多少时间才能把 k 个人从起点运到终点。

题解

考虑按时间拆点，起点连接源点，终点连接汇点。

每个位置的相邻时间点连接一条容量无限大的边，表示停留在该位置一个单位时间。

然后枚举时间，根据时间和飞船航线加边，每次在残量网络上跑最大流并累加到总流量，总流量不小于 k 时结束枚举。

至于无解的判定，可以一开始用并查集维护一下连通性。

```
const int MAXN=1e5+5,MAXM=2e5+5,Inf=0x7fffffff;
struct Edge{
    int to,cap,next;
    Edge(int to=0,int cap=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int c){
    edge[edge_cnt]=Edge(v,c,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,head[v]);
    head[v]=edge_cnt++;
}
```

```
struct Dinic{
    int s,t;
    int pos[MAXN],vis[MAXN],dis[MAXN];
    bool bfs(int k){
        queue<int>q;
        q.push(s);
        vis[s]=k,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(vis[v]!=k&&edge[i].cap){
                    vis[v]=k,dis[v]=dis[u]+1,pos[v]=head[v];
                    q.push(v);
                    if(v==t)
                    return true;
                }
            }
        }
        return false;
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(dis[u]+1==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        return flow;
    }
    int Maxflow(int s,int t){
        this->s=s;this->t=t;
        int ans=0,k=0;
        mem(vis,0);
        while(bfs(++k))
        ans+=dfs(s,Inf);
        return ans;
    }
}solver;
const int MAXS=25;
vector<int> g[MAXS];
int p[MAXS],f[MAXS],T[MAXS];
```

```
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
int main()
{
    Clear();
    int n=read_int()+2,m=read_int(),k=read_int(),s=MAXN-2,t=MAXN-1,x,y;
    _rep(i,1,n)
    p[i]=i;
    _for(i,0,m){
        f[i]=read_int(),T[i]=read_int();
        _for(j,0,T[i]){
            x=read_int();
            if(x==0)
            x=n-1;
            else if(x==-1)
            x=n;
            g[i].push_back(x);
        }
        _for(j,1,T[i]){
            x=Find(g[i][j-1]),y=Find(g[i][j]);
            if(x!=y)
            p[x]=y;
        }
    }
    if(Find(n-1)!=Find(n)){
        puts("0");
        return 0;
    }
    Insert(s,n-1,Inf);Insert(n,t,Inf);
    int ans=0;
    for(int flow=0;flow<k;ans++){
        _for(i,1,n)
        Insert(i+ans*n,i+(ans+1)*n,Inf);
        Insert((ans+2)*n,(ans+1)*n,Inf);
        _for(i,0,m)
        Insert(g[i][ans%T[i]]+ans*n,g[i][(ans+1)%T[i]]+(ans+1)*n,f[i]);
        flow+=solver.Maxflow(s,t);
    }
    enter(ans);
    return 0;
}
```

餐巾计划问题

洛谷p1251

题意

一个餐厅第 i 天需要 r_i 条新餐巾，每条新餐巾用完后得到旧餐巾，一开始餐厅没有餐巾。

买一条新餐巾费用为 p□快洗一条旧餐巾时间为 m 天，费用为 f□慢洗一条旧餐巾时间为 n 天，费用为 s□

数据保证 $f\gt s,m\lt n$□问餐厅营业 N 天的最小花费。

题解

把一天分成一天的开始和一天的结束，一天的开始需要提供 r_i 条新餐巾，于是连一条容量为 r_i 费用为 0 的边到汇点。

接下来考虑三种获取新餐巾的操作。首先可以买新餐巾，对应源点连一条容量为 $\inf$ 费用为 p 的边到当天的开始。

另外也可以通过清洗之前的旧餐巾获得。不妨假设如果旧餐巾洗完就会被立即使用，不然可以留到以后再洗。这样假设的目的是减少连边数量。

于是第 i 天的结束向第 $i+m$ 天的开始连一条容量为 $\inf$ 费用为 f 的边，向第 $i+n$ 天的开始连一条容量为 $\inf$ 费用为 s 的边。

再考虑维护旧餐巾，旧餐巾可以通过两种方式获取。

首先一天的结束将有 r_i 条新餐巾变成旧餐巾，于是从源点连一条容量为 r_i 费用为 0 的边到一天的结束。

另外旧餐巾也可以继承前一天剩下的未被清洗的旧餐巾，对应每天结束向第二天结束连一条容量为 $\inf$ 费用为 0 的边。

最后跑最小费用最大流即可。

```
const int MAXN=5000,MAXM=15000,Inf=0x7fffffff;
struct Edge{
    int to,cap,w,next;
    Edge(int to=0,int cap=0,int w=0,int next=0){
        this->to=to;
        this->cap=cap;
        this->w=w;
        this->next=next;
    }
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Clear(){mem(head,-1);edge_cnt=0;}//边从0开始编号
void Insert(int u,int v,int w,int c){
    edge[edge_cnt]=Edge(v,c,w,head[u]);
    head[u]=edge_cnt++;
    edge[edge_cnt]=Edge(u,0,-w,head[v]);
    head[v]=edge_cnt++;
}
struct Dinic{
    int s,t;
    int pos[MAXN],dis[MAXN];
```

```
    bool inque[MAXN];
    bool spfa(){
        mem(dis,127/3);
        queue<int>q;
        q.push(s);
        inque[s]=true,dis[s]=0,pos[s]=head[s];
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];~i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[u]+edge[i].w<dis[v]&&edge[i].cap){
                    dis[v]=dis[u]+edge[i].w,pos[v]=head[v];
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return dis[t]!=dis[0];
    }
    int dfs(int u,int max_flow){
        if(u==t||!max_flow)
        return max_flow;
        int flow=0,temp_flow;
        inque[u]=true;
        for(int &i=pos[u];~i;i=edge[i].next){
            int v=edge[i].to;
if(!inque[v]&&dis[u]+edge[i].w==dis[v]&&(temp_flow=dfs(v,min(max_flow,edge[
i].cap)))){
                edge[i].cap-=temp_flow;
                edge[i^1].cap+=temp_flow;
                flow+=temp_flow;
                max_flow-=temp_flow;
                if(!max_flow)
                break;
            }
        }
        inque[u]=false;
        return flow;
    }
    void MCMF(int s,int t,int &flow,LL &cost){
        this->s=s;this->t=t;
        cost=flow=0;
        int temp_flow;
        mem(inque,0);
        while(spfa()){
            temp_flow=dfs(s,Inf);
            flow+=temp_flow;
            cost+=1LL*temp_flow*dis[t];
```

```
        }
    }
}solver;
int a[MAXN];
int main()
{
    Clear();
    int n=read_int(),s=n<<1|1,t=s+1;
    _rep(i,1,n)
    a[i]=read_int();
    int
c1=read_int(),t2=read_int(),c2=read_int(),t3=read_int(),c3=read_int();
    _rep(i,1,n){
        Insert(s,i+n,0,a[i]);
        Insert(i,t,0,a[i]);
        Insert(s,i,c1,Inf);
        if(i+t2<=n){
            Insert(i+n,i+t2,c2,Inf);
            if(i+t3<=n)
            Insert(i+n,i+t3,c3,Inf);
        }
    }
    _for(i,1,n)
    Insert(i+n,i+n+1,0,Inf);
    int flow;LL cost;
    solver.MCMF(s,t,flow,cost);
    enter(cost);
    return 0;
}
```