

Simple template

Sparken

August 14, 2018

目录

1	Pretreatment	3
1.1	头文件 Headers and constants	3
1.2	配置 Vim setting	4
2	图论 Graph Theory	5
2.1	最短路 The shortest path	5
2.1.1	Dijkstra	5
2.1.2	Spfa	6
2.1.3	次短路	7
2.1.4	第 K 短路	8
2.2	生成树 Spanning tree	10
2.2.1	最小生成树 Minimum spanning tree	10
2.2.2	次小生成树	13
2.3	网络流 Network flow	15
2.3.1	最大流-Dinic	15
2.3.2	最大流-ISAP	17
2.3.3	最小费用最大流-EdmondsKarp	20
2.3.4	建图-有上下界的可行流	22
2.4	二分图	28
2.4.1	概念公式	28
2.4.2	最大匹配-匈牙利	28
2.5	强连通缩点 tarjan	30
2.6	最近公共祖先 LCA	32
2.6.1	tarjan	32
2.6.2	dfs+ST	32
2.7	欧拉回路	34
2.7.1	判定	34
2.7.2	求解	34
2.8	哈密顿回路	34
3	数据结构 Data Structure	35
3.1	并查集 Union-Find Set	35
3.2	拓扑排序 Topological Sorting	36
3.3	树状数组	37
3.4	RMQ	38
3.5	线段树 Segment Tree	38
3.6	树链剖分 HeavyLightDecomposition	40
3.7	伸展树 splay	41
3.7.1	维护序列	41

3.7.2	平衡树	44
4	数学 Math	45
4.1	定理公式与结论 Conclusions	45
4.2	快速乘-快速幂	45
4.3	矩阵快速幂	45
4.4	扩展欧几里得	46
4.5	欧拉函数	47
4.6	中国剩余定理求同余方程组	48
4.6.1	素数	48
4.6.2	非素数	49
5	字符串 String	50
5.1	字典树 Trie	50
5.2	KMP	51
5.3	扩展 KMP	52
5.4	最长回文子串 Manacher	54
5.5	AC 自动机	55
6	动态规划	57
6.1	背包	57
7	其它 Other	59
7.1	莫队	59
7.2	离散化	60
7.3	STL	60
7.4	pbds	61
8	输入输出 IO	62
8.1	输入输出外挂	62
8.2	简单大数	62
8.3	Python 输入输出	64

1 Pretreatment

1.1 头文件 Headers and constants

```
#include <bits/stdc++.h>
using namespace std;
#define X first
#define Y second
const int INF=0x3f3f3f3f;
const double eps=1e-6;
/*
    整型初始化 memset(d,0x3f,sizeof(d));
    浮点初始化 memset(d,0x7f,sizeof(d));
    浮点数比较大小:    相等 if(fabs(a-b)<=eps)
                        大于 if(a>b ES fabs(a-b)>eps)
                        小于 if(a<b ES fabs(a-b)>eps)
*/
```

1.2 配置 Vim setting

=====

14 行基本设置

syntax on

set cindent

set nu

set shortmess=atI

set tabstop=4

set shiftwidth=4

set confirm

set mouse=a

map<C-A> ggVG"+y

map <F5> :call Run()<CR>

func! Run()

 exec "w"

 exec "!g++ -Wall % -o %<"

 exec "!time ./%<"

endfunc

=====

括号补全

inoremap { {<CR>}<ESC>kA<CR>

inoremap ((<ESC>i

inoremap [[<ESC>i

inoremap " "<ESC>i

跳转行末

inoremap <C- > <End>

=====

【快捷键】

:nu 行跳转 /text 查找 text,n 查找下一个, N 查找前一个

u 撤销, U 撤销对整行的操作

Ctrl+r 撤销的撤销

2 图论 Graph Theory

2.1 最短路 The shortest path

2.1.1 Dijkstra

```
typedef pair<int,int> P;
struct Dijkstra
{
    vector<P> G[MAXN];
    bool vis[MAXN];
    int d[MAXN];
    void init(int N)
    {
        for(int i=0;i<=N;++i) G[i].clear();
        memset(vis,false,sizeof(vis));
        memset(d,0x3f,sizeof(d));
    }
    void addEdge(int u,int v,int cost)
    {
        G[u].push_back(make_pair(cost,v));
    }
    void dij(int s)
    {
        priority_queue<P,vector<P>,greater<P> > q;
        d[s]=0;
        q.push(make_pair(d[s],s));
        while(!q.empty())
        {
            P temp=q.top();q.pop();
            int v=temp.Y;
            if(vis[v]) continue;
            vis[v]=true;
            for(int i=0;i<G[v].size();++i)
            {
                int u=G[v][i].Y,cost=G[v][i].X;
                if(!vis[u] && d[u]>d[v]+cost)
                {
                    d[u]=d[v]+cost;
                    q.push(make_pair(d[u],u));
                }
            }
        }
    }
};
```

```
        }
    }
}
};
```

2.1.2 Spfa

```
typedef pair<int,int> P;
struct Spfa
{
    vector<pair<int,int> > G[MAXN];
    bool vis[MAXN];
    int inq[MAXN],d[MAXN];
    void init()
    {
        for(int i=0;i<=MAXN;++i) G[i].clear();
        memset(vis,false,sizeof(vis));
        memset(inq,0,sizeof(inq));
        memset(d,0x3f,sizeof(d));
    }
    void add_edge(int u,int v,int cost)
    {
        G[u].push_back(make_pair(cost,v));
    }
    int spfa(int s)
    {
        queue<int> q;
        d[s]=0;
        q.push(s);
        ++inq[s];
        vis[s]=true;
        while(!q.empty())
        {
            int v=q.front();q.pop();
            vis[v]=false;
            for(int i=0;i<G[v].size();++i)
            {
                int u=G[v][i].Y,cost=G[v][i].X;
                if(d[u]>d[v]+cost)
                {
```

```
        d[u]=d[v]+cost;
        if(!vis[u])
        {
            q.push(u);
            ++inq[u];
            vis[u]=true;
        }
    }
    if(inq[v]>N) return -1;    //有负圈
}
if(d[N]==INF) return -2;    //不可达
return d[N];
}
};
```

2.1.3 次短路

```
typedef pair<int,int> P;
struct Dijkstra
{
    vector<P> G[MAXN];
    int d[MAXN],d2[MAXN];
    void init(int N)
    {
        for(int i=0;i<=N;++i) G[i].clear();
        memset(d,0x3f,sizeof(d));
        memset(d2,0x3f,sizeof(d2));
    }
    void addEdge(int u,int v,int cost)
    {
        G[u].push_back(make_pair(cost,v));
    }
    void dij(int s)
    {
        priority_queue<P,vector<P>,greater<P> > q;
        d[s]=0;
        q.push(make_pair(d[s],s));
        while(!q.empty())
        {
```



```

    P temp=q.top();q.pop();
    int v=temp.Y;
    if(d2[v]<temp.X) continue;
    for(int i=0;i<G[v].size();++i)
    {
        int u=G[v][i].Y,cost=G[v][i].X;
        int dist=temp.X+cost;
        if(d[u]>dist)//该点当前最短路比新距离大
        {
            swap(d[u],dist);
            q.push(P(d[u],u));
        }
        if(d2[u]>dist && d[u]<dist)
        {
            d2[u]=dist;
            q.push(P(d2[u],u));
        }
    }
}
};
//另一种做法：正反跑最短路，枚举每边是否在最短路，非则记此时路长

```

2.1.4 第 K 短路

```

struct Edge
{
    int from;
    int d,f;
    Edge(int u,int d,int f):from(u),d(d),f(f){}
    bool operator <(const Edge &a)const //从大到小排序，避免用反 pq
    {
        if(f==a.f) return a.d<d;
        return a.f<f;
    }
};
struct Kpath
{
    vector<pair<int,int> > G[MAXN];
    vector<pair<int,int> > GB[MAXN];

```

```
bool vis[MAXN];
int h[MAXN];
int t;
void init()
{
    for(int i=0;i<=N;++i)
    {
        G[i].clear();
        GB[i].clear();
    }
    t=0;
    memset(h,0x3f,sizeof(h));
    memset(vis,false,sizeof(vis));
}
void addEdge(int u,int v,int cost)
{
    G[u].push_back(mp(cost,v));
    GB[v].push_back(mp(cost,u));
}
void spfa(int s)    //dijkstra 可能效率更高,另注意题目会不会有负圈
{
    queue<int> q;
    h[s]=0;
    q.push(s);
    vis[s]=true;
    while(!q.empty())
    {
        int u=q.front();q.pop();
        vis[u]=false;
        for(int i=0;i<GB[u].size();++i)
        {
            int v=GB[u][i].Y,cost=GB[u][i].X;
            if(h[v]>h[u]+cost)
            {
                h[v]=h[u]+cost;
                if(!vis[v])
                {
                    q.push(v);
                    vis[v]=true;
                }
            }
        }
    }
}
```

```
        }
    }
}

int Astar(int S,int T,int K)
{
    if(S==T) ++K;          //如果 S==T, d=0 不算一条路
    if(h[S]==INF) return -1;
    priority_queue<Edge> q;
    q.push(Edge(S,0,h[S]));
    while(!q.empty())
    {
        Edge temp=q.top();q.pop();
        int u=temp.from,d=temp.d;
        if(u==T) ++t;
        if(t==K) return d;
        for(int i=0;i<G[u].size();++i)
        {
            int v=G[u][i].Y,cost=G[u][i].X;
            q.push(Edge(v,d+cost,d+cost+h[v]));
        }
    }
    return -1;
};
```

2.2 生成树 Spanning tree

2.2.1 最小生成树 Minimum spanning tree

```
struct Edge
{
    int u,v,d;
    Edge(int from,int to,int cost):u(from),v(to),d(cost){}
    bool operator < (const Edge &a)const
    {
        return d<a.d;
    }
};

struct Kruskal
```

```
{
    vector<Edge> edges;
    int par[MAXN];
    int n;
    void init(int n)
    {
        this->n=n;
        edges.clear();
        for(int i=1;i<=n;++i) par[i]=i;
    }
    void add_edge(int u,int v,int d)
    {
        edges.push_back(Edge(u,v,d));
        edges.push_back(Edge(v,u,d));
    }
    int Find(int x)
    {
        if(par[x]==x) return x;
        return par[x]=Find(par[x]);
    }
    void uni(int A,int B)
    {
        int x=Find(A),y=Find(B);
        if(x==y) return ;
        par[x]=y;
    }
    bool same(int A,int B)
    {
        return Find(A)==Find(B);
    }
    int kruskal()
    {
        sort(edges.begin(),edges.end());
        int ans=0;
        for(int i=0;i<edges.size();++i)
        {
            Edge &e=edges[i];
            if(!same(e.v,e.u))
            {
```

```
        ans+=e.d;
        uni(e.v,e.u);
    }
}
return ans;
}
};
```

```
struct Prim
{
    bool vis[MAXN];
    int d[MAXN],cost[MAXN][MAXN];
    int n;
    void init(int n)
    {
        this->n=n;
        memset(d,0x3f,sizeof(d));
        memset(cost,0x3f,sizeof(cost));
        memset(vis,false,sizeof(vis));
    }
    int prim(int s)
    {
        d[s]=0;
        int ans=0;
        while(1)
        {
            int v=-1;
            for(int u=1;u<=n;++u)
                if(!vis[u] && (v==-1 || d[u]<d[v])) v=u;
            if(v==-1) break;
            vis[v]=true;
            ans+=d[v];
            for(int u=1;u<=n;++u)
                d[u]=min(d[u],cost[v][u]);
        }
        return ans;
    }
};
```

2.2.2 次小生成树

```
struct Edge
{
    int u,v,cost;
    bool use;
    Edge(int u,int v,int c,bool use):u(u),v(v),cost(c),use(use){}
};

struct SecMST
{
    vector<Edge> es;
    int par[MAXN],length[MAXN][MAXN];
    void init(int n)
    {
        for(int i=0;i<=n;++i) par[i]=i;
        memset(length,0,sizeof(length));
        es.clear();
    }
    int Find(int x)
    {
        if(par[x]==x) return x;
        return par[x]=Find(par[x]);
    }
    void uni(int A,int B)
    {
        int x=Find(A),y=Find(B);
        if(x==y) return ;
        par[x]=y;
    }
    bool same(int A,int B){return Find(A)==Find(B);}
    bool cmp(Edge a,Edge b){return a.cost<b.cost;}
    void update(int u,int v,int cost)
    {
        for(int i=1;i<=N;++i)
            for(int j=1;j<=N;++j)
            {
                if(i!=j && same(a,u) && same(b,v))
                {
                    length[a][b]=length[b][a]=cost;
                }
            }
    }
};
```

```
    }
}

int kruskal()
{
    sort(es.begin(), es.end(), cmp);
    int ans=0;
    for(int i=0; i<es.size(); ++i)
    {
        Edge &e=es[i];
        int u=e.u, v=e.v, cost=e.cost;
        if(!same(u, v))
        {
            ans+=cost;
            e.use=true;
            update(u, v, cost); //若 MST 结束 DFS 遍历树得 length 效率更高
            uni(u, v);
        }
    }
    return ans;
}

int secmst()
{
    int MST=kruskal();
    int SECMST=INF;
    bool flag=false;
    for(int i=0; i<es.size(); ++i)
    {
        Edge &e=es[i];
        if(!e.use)
        {
            //枚举非 MST 的边 (u, v), 加入 MST 形成环
            //则 SECMST=MST+ 该边 w-所成环中 uv 间最长边
            SECMST=min(SECMST, MST+e.cost-length[e.u][e.v]);
            if(SECMST==MST)
            {
                flag=true;
                break;
            }
        }
    }
}
```

```
    }  
    return SECMST;  
}  
};
```

2.3 网络流 Network flow

2.3.1 最大流-Dinic

```
struct Edge  
{  
    int from,to,cap,flow;  
    Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f){}  
};  
  
struct Dinic  
{  
    int n,m,s,t;  
    vector<Edge> edges;  
    vector<int> G[MAXN];  
    bool vis[MAXN];  
    int d[MAXN],cur[MAXN];  
    void init(int n)  
    {  
        for(int i=0;i<=n;++i) G[i].clear();  
        edges.clear();  
    }  
    void addEdge(int from,int to,int cap)  
    {  
        edges.push_back(Edge(from,to,cap,0));  
        edges.push_back(Edge(to,from,0,0));  
        m=edges.size();  
        G[from].push_back(m-2);  
        G[to].push_back(m-1);  
    }  
    bool BFS()  
    {  
        memset(vis,false,sizeof(vis));  
        queue<int> q;  
        q.push(s);  
        d[s]=0;  
        vis[s]=1;
```



```
while(!q.empty())
{
    int v=q.front();q.pop();
    for(int i=0;i<G[v].size();++i)
    {
        int ecode=G[v][i];
        Edge &e=edges[ecode];
        if(!vis[e.to] && e.cap>e.flow)
        {
            vis[e.to]=true;
            d[e.to]=d[v]+1;
            q.push(e.to);
        }
    }
}
return vis[t];
}

int DFS(int v,int a)
{
    if(v==t || a==0) return a;
    int flow=0,f;
    for(int &i=cur[v];i<G[v].size();++i)
    {
        int ecode=G[v][i];
        Edge &e=edges[ecode];
        if(d[v]+1==d[e.to] && (f=DFS(e.to,min(a,e.cap-e.flow)))>0)
        {
            e.flow+=f;
            edges[ecode^1].flow-=f;
            flow+=f;
            a-=f;
            if(a==0) break;
        }
    }
    return flow;
}

int maxFlow(int s,int t)
{
    this->s=s;this->t=t;
```

```
    int flow=0;
    while(BFS())
    {
        memset(cur,0,sizeof(cur));
        flow+=DFS(s,INF);
    }
    return flow;
}
};
```

2.3.2 最大流-ISAP

```
struct Edge
{
    int from,to,cap,flow;
    Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f){}
};

struct ISAP
{
    int n,m,s,t;
    vector<Edge> edges;
    vector<int> G[MAXN];
    bool vis[MAXN];
    int d[MAXN],cur[MAXN];
    int p[MAXN],num[MAXN];
    void init(int n)
    {
        this->n=n;
        for(int i=0;i<n;++i) G[i].clear();
        edges.clear();
        memset(d,0x3f,sizeof(d));
    }
    void add_edge(int from,int to,int cap)
    {
        edges.push_back(Edge(from,to,cap,0));
        edges.push_back(Edge(to,from,0,0));
        m=edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }
};
```

```
bool bfs()
{
    memset(vis,false,sizeof(vis));
    queue<int> q;
    q.push(t);
    d[t]=0;
    vis[t]=true;
    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(int i=0;i<G[u].size();++i)
        {
            Edge &e=edges[G[u][i]^1];
            if(!vis[e.from] && e.cap>e.flow)
            {
                vis[e.from]=true;
                d[e.from]=d[u]+1;
                q.push(e.from);
            }
        }
    }
    return vis[s];
}

int Augment()
{
    int flow=INF;
    for(int u=t;u!=s;u=edges[p[u]].from)
    {
        Edge &e=edges[p[u]];
        flow=min(flow,e.cap-e.flow);
    }
    for(int u=t;u!=s;u=edges[p[u]].from)
    {
        edges[p[u]].flow+=flow;
        edges[p[u]^1].flow-=flow;
    }
    return flow;
}

int Maxflow(int s,int t)
```

```
{
    this->s=s;this->t=t;
    int flow=0;
    bfs();
    if(d[s]>=n) return 0;
    memset(num,0,sizeof(num));
    for(int i=0;i<n;++i)
        if(d[i]<INF) ++num[d[i]];
    int u=s;
    memset(cur,0,sizeof(cur));
    while(d[s]<n)
    {
        if(u==t)
        {
            flow+=Augment();
            u=s;
        }
        int ok=0;
        for(int i=cur[u];i<G[u].size();++i)
        {
            Edge &e=edges[G[u][i]];
            if(e.cap>e.flow && d[u] == d[e.to]+1)
            {
                ok=1;
                p[e.to]=G[u][i];
                cur[u]=i;
                u=e.to;
                break;
            }
        }
        if(!ok)
        {
            int m=n-1;
            for(int i=0;i<G[u].size();++i)
            {
                Edge &e=edges[G[u][i]];
                if(e.cap>e.flow) m=min(m,d[e.to]);
            }
            if(--num[d[u]]==0) break;
        }
    }
}
```

```
        ++num[d[u]=m+1];
        cur[u]=0;
        if(u!=s) u=edges[p[u]].from;
    }
}
return flow;
}
};
```

2.3.3 最小费用最大流-EdmondsKarp

//最大费用最大流则费用取反

```
struct Edge
{
    int from,to,cap,flow,cost;
    Edge(int u,int v,int c,int f,int w):from(u),to(v),cap(c),flow(f),cost(w){}
};

struct MCMF
{
    int n,m;
    vector<Edge> edges;
    vector<int> G[MAXN];
    int inq[MAXN],d[MAXN],p[MAXN],a[MAXN];

    void init(int n)
    {
        this->n=n;
        for(int i=0;i<n;++i) G[i].clear();
        edges.clear();
    }

    void add_edge(int from,int to,int cap,int cost)
    {
        edges.push_back(Edge(from,to,cap,0,cost));
        edges.push_back(Edge(to,from,0,0,-cost));
        m=edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }
};
```

```
bool spfa(int s,int t,int &flow,long long &cost)
{
    for(int i=0;i<n;++i) d[i]=INF;
    memset(inq,0,sizeof(inq));
    d[s]=0;inq[s]=1;p[s]=0;a[s]=INF;

    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int u=q.front();q.pop();
        inq[u]=0;
        for(int i=0;i<G[u].size();++i)
        {
            Edge &e=edges[G[u][i]];
            if(e.cap>e.flow && d[e.to]>d[u]+e.cost)
            {
                d[e.to]=d[u]+e.cost;
                p[e.to]=G[u][i];
                a[e.to]=min(a[u],e.cap-e.flow);
                if(!inq[e.to])
                {
                    q.push(e.to);
                    inq[e.to]=1;
                }
            }
        }
    }
    if(d[t]==INF) return false;
    flow+=a[t];
    cost+=(long long)d[t]*(long long)a[t];
    for(int u=t;u!=s;u=edges[p[u]].from)
    {
        edges[p[u]].flow+=a[t];
        edges[p[u]^1].flow-=a[t];
    }
    return true;
}
```

```
int MincostMaxflow(int s,int t,long long &cost)
{
    int flow=0;cost=0;
    while(spfa(s,t,flow,cost)) ;
    return flow;
}
};
```

2.3.4 建图-有上下界的可行流

```
//poj2396
struct Edge
{
    int from,to,cap,flow;
    Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f){}
};
int low[MAXN][MAXN],up[MAXN][MAXN];
int rowsum[MAXN],colsum[MAXN];
int in[MAXN],out[MAXN];
int n,m,source;
struct ISAP
{
    int n,m,s,t;
    vector<Edge> edges;
    vector<int> G[MAXN];
    bool vis[MAXN];
    int d[MAXN],cur[MAXN];
    int p[MAXN],num[MAXN];

    void init(int n)
    {
        this->n=n;
        for(int i=0;i<n;++i) G[i].clear();
        edges.clear();
        memset(d,0x3f,sizeof(d));
    }

    void addedge(int from,int to,int cap)
    {
        edges.push_back(Edge(from,to,cap,0));
```

```
edges.push_back(Edge(to,from,0,0));
m=edges.size();
G[from].push_back(m-2);
G[to].push_back(m-1);
}

bool bfs()
{
    memset(vis,false,sizeof(vis));
    queue<int> q;
    q.push(t);
    d[t]=0;
    vis[t]=true;
    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(int i=0;i<G[u].size();++i)
        {
            Edge &e=edges[G[u][i]^1];
            if(!vis[e.from] && e.cap>e.flow)
            {
                vis[e.from]=true;
                d[e.from]=d[u]+1;
                q.push(e.from);
            }
        }
    }
    return vis[s];
}

int Augment()
{
    int flow=INF;
    for(int u=t;u!=s;u=edges[p[u]].from)
    {
        Edge &e=edges[p[u]];
        flow=min(flow,e.cap-e.flow);
    }
    for(int u=t;u!=s;u=edges[p[u]].from)
```



```
{
    edges[p[u]].flow+=flow;
    edges[p[u]^1].flow-=flow;
}
return flow;
}

int Maxflow(int s,int t)
{
    this->s=s;this->t=t;
    int flow=0;
    bfs();
    if(d[s]>=n) return 0;
    memset(num,0,sizeof(num));
    for(int i=0;i<n;++i)
        if(d[i]<INF) ++num[d[i]];
    int u=s;
    memset(cur,0,sizeof(cur));
    while(d[s]<n)
    {
        if(u==t)
        {
            flow+=Augment();
            u=s;
        }
        int ok=0;
        for(int i=cur[u];i<G[u].size();++i)
        {
            Edge &e=edges[G[u][i]];
            if(e.cap>e.flow && d[u]==d[e.to]+1)
            {
                ok=1;
                p[e.to]=G[u][i];
                cur[u]=i;
                u=e.to;
                break;
            }
        }
    }
    if(!ok)
```

```
{
    int m=n-1;
    for(int i=0;i<G[u].size();++i)
    {
        Edge &e=edges[G[u][i]];
        if(e.cap>e.flow) m=min(m,d[e.to]);
    }
    if(--num[d[u]]==0) break;
    ++num[d[u]=m+1];
    cur[u]=0;
    if(u!=s) u=edges[p[u]].from;
}
}
return flow;
}

bool build(int m,int n,int s,int t)
{
    for(int i=1;i<=m;++i)
        for(int j=1;j<=n;++j)
            if(low[i][j]<=up[i][j])
                addedge(i,j+m,up[i][j]-low[i][j]);
            else return false;

    for(int i=1;i<=m;++i)
    {
        addedge(s,i,rowsum[i]-in[i]);
        source+=rowsum[i]-in[i];
    }
    for(int j=1;j<=n;++j)
    {
        addedge(j+m,t,colsum[j]-out[j]);
    }
    return true;
}

void print(int m,int n)
{
    for(int i=1;i<=m;++i)
```

```
        for(int j=1;j<=n;++j)
        {
            Edge &e=edges[(i-1)*n*2+(j-1)*2];
            if(j>1) putchar(' ');
            printf("%d",e.flow+low[i][j]);
            if(j==n) putchar('\n');
        }
    }
}ans;
void init(int m,int n)
{
    source=0;
    for(int i=1;i<=m;++i)
        for(int j=1;j<=n;++j)
            low[i][j]=0,up[i][j]=INF;
}
int main()
{
    int N;
    scanf("%d",&N);
    while(N--)
    {
        scanf("%d%d",&m,&n);
        ans.init(MAXN);
        init(m,n);
        int row=0,col=0;
        for(int i=1;i<=m;++i)
        {
            scanf("%d",&rowsum[i]);
            row+=rowsum[i];
        }
        for(int i=1;i<=n;++i)
        {
            scanf("%d",&colsum[i]);
            col+=colsum[i];
        }
        int C;
        scanf("%d",&C);
        while(C--)
```

```
{
    int r,c,val;
    char ope[5];
    scanf("%d%d%s%d",&r,&c,ope,&val);
    int rstart=r,rend=r,cstart=c,cend=c;
    if(c==0) cstart=1,cend=n;
    if(r==0) rstart=1,rend=m;
    for(int i=rstart;i<=rend;++i)
        for(int j=cstart;j<=cend;++j)
        {
            if(ope[0]=='=')
                low[i][j]=up[i][j]=val;
            else if(ope[0]=='>')
                low[i][j]=max(low[i][j],val+1);
            else if(ope[0]=='<')
                up[i][j]=min(up[i][j],val-1);
        }
    }
    memset(in,0,sizeof(in));
    memset(out,0,sizeof(out));
    for(int i=1;i<=m;++i)
        for(int j=1;j<=n;++j)
            in[i]+=low[i][j];
    for(int j=1;j<=n;++j)
        for(int i=1;i<=m;++i)
            out[j]+=low[i][j];
    int S=0,T=m+n+1;
    bool flag=false;
    if(row!=col) flag=true;
    if(!flag && !ans.build(m,n,S,T)) flag=true;
    int flow=ans.Maxflow(S,T);
    if(!flag && flow!=source) flag=true;
    if(flag) printf("IMPOSSIBLE\n");
    else ans.print(m,n);
    printf("\n");
}
return 0;
}
```

2.4 二分图

2.4.1 概念公式

/*

- 1、二分图等价条件：不存在奇环的图。
- 2、概念：最小点覆盖：选最少点使每边至少和一点关联
 最小边覆盖：选最少边使每点和且仅和一条边关联
 最大独立集：[无向图] 选最多点使它们互不相邻
 最大团：[无向图] 选最多点使构成完全子图
- 3、公式：最大匹配数 + 最小边覆盖 = V
 最大独立集 + 最小点覆盖 = V
 最大匹配数 = 最小点覆盖（二分图中）
 最大团：补图最大独立集
 最小路径覆盖： $N \times N$ 有向无环图，拆点， $i \rightarrow j \implies i1 \rightarrow j2$ 构成二分图
 最小路径覆盖 = $n - m$ (n 原图点数， m 新图最大匹配数)
- 4、常用建图法：行列、奇偶（坐标和）、反向（所给条件相反的两点间建图）、
 拆点、一行变多行 一列变多列

*/

2.4.2 最大匹配-匈牙利

```
struct Hungary
{
    vector<int> G[MAXN];
    int match[MAXN];
    bool used[MAXN];
    int V;

    void init(int N)
    {
        this->V=N;
        for(int i=0;i<=V;++i) G[i].clear();
    }
    void add_edge(int u,int v)
    {
        G[u].push_back(v);
    }
    bool dfs(int v)
    {
        for(int i=0;i<G[v].size();++i)
        {
```

```
        int u=G[v][i];
        if(!used[u])
        {
            used[u]=true;
            if(match[u]==-1 || dfs(match[u]))
            {
                match[u]=v;
                return true;
            }
        }
    }
    return false;
}

int hungary()
{
    int ans=0;
    memset(match,-1,sizeof(match));
    for(int v=1;v<=V;++v)
    {
        memset(used,false,sizeof(used));
        if(dfs(v)) ++ans;
    }
    return ans;
}

};
```

```
/*
 * 邻接矩阵
 * 初始化 G[][] 两边顶点划分
 * G[i][j] 表示 i->j 有向边 (左向右)
 * G 无边相连则初始化为 0
 * 复杂度 O(VE)
 * 编号从 0 开始
 */
```

```
struct HungaryM
{
    bool used[MAXN];
    int G[MAXN][MAXN],match[MAXN];
    int uN,vN;//左点数, 右点数
    bool dfs(int u)
```

```
{
    for(int v=0;v<vN;++v)
        if(G[u][v] && !used[v])
        {
            used[v]=true;
            if(match[v]==-1 || dfs(match[v]))
            {
                match[v]=u;
                return true;
            }
        }
    return false;
}
int hungary()
{
    int ans=0;
    memset(match,-1,sizeof(match));
    for(int u=0;u<uN;++u)
    {
        memset(used,false,sizeof(used));
        if(dfs(u)) ++ans;
    }
    return ans;
}
};
```

2.5 强连通缩点 tarjan

```
struct SCC
{
    vector<int> G[MAXN];
    int pre[MAXN],lowlink[MAXN],sccno[MAXN],dfs_clock,scc;
    //scc: 强连通分量个数, sccno[i]: 缩点后 i 所在点编号
    stack<int> s;
    void init()
    {
        for(int i=0;i<=N;++i) G[i].clear();
        memset(sccno,0,sizeof(sccno));
        memset(pre,0,sizeof(pre));
        while(!s.empty()) s.pop();
    }
```

```
    dfs_clock=scc=0;
}
void add_edge(int u,int v)
{
    G[u].push_back(v);
}
void tarjan(int u)
{
    pre[u]=lowlink[u]=++dfs_clock;
    s.push(u);
    for(int i=0;i<G[u].size();++i)
    {
        int v=G[u][i];
        if(!pre[v])
        {
            tarjan(v);
            lowlink[u]=min(lowlink[u],lowlink[v]);
        }
        else if(!sccno[v])
        {
            lowlink[u]=min(lowlink[u],pre[v]);
        }
    }
    if(lowlink[u]==pre[u])
    {
        ++scc;
        for(;;)
        {
            int v=s.top();s.pop();
            sccno[v]=scc;
            if(v==u) break;
        }
    }
}
/* 全图缩点
   for(int i=1;i<=N;++i)
       if(!pre[i]) tarjan(i);
*/
};
```


2.6 最近公共祖先 LCA

2.6.1 tarjan

```
// 离线 Tarjan, 时间复杂度:  $O(n+q)$ 
vector<int> G[maxn];
int par[maxn], vis[maxn], ans[maxn];
vector<PII> query[maxn]; // 存储查询信息
inline void init(int n)
{
    for (int i = 1; i <= n; i++)
    {
        G[i].clear(), query[i].clear();
        par[i] = i, vis[i] = 0;
    }
}
inline void add_edge(int u, int v) { G[u].pb(v); }
inline void add_query(int u, int v, int id) { query[u].pb({v, id}); query[v].pb({u,
↪ id}); }
void tarjan(int u)
{
    vis[u] = 1;
    for (auto& v : G[u])
    {
        if (vis[v]) continue;
        tarjan(v);
        unite(u, v);
    }
    for (auto& q : query[u])
    {
        int &v = q.X, &id = q.Y;
        if (!vis[v]) continue;
        ans[id] = find(v);
    }
}
```

2.6.2 dfs+ST

```
// 欧拉序列 +ST 表, 时间复杂度  $O(2n\log(2n)+q)$ 
vector<int> G[maxn];
vector<int> seq; // 欧拉序列 (但叶子处只放了一个)
int dep[maxn], in[maxn]; // 深度和各点进栈时间 (从 0 开始)
```

```
pair<int,int> dp[21][maxn << 1]; // dp[log(maxn)][maxn<<1], .X 为深度, .Y 为位置
void init(int n)
{
    for (int i = 0; i <= n; i++) G[i].clear();
    seq.clear();
}
void addedge(int u,int v) { G[u].emplace_back(v); G[v].emplace_back(u); }
void dfs(int u, int fa)
{
    dep[u] = dep[fa] + 1;
    in[u] = seq.size(); // 进栈时间
    seq.push_back(u);
    for (auto &v : G[u])
    {
        if (v == fa) continue;
        dfs(v, u);
        seq.push_back(u);
    }
    // 出栈时间 seq.size()+1;
}
void initrmq()
{
    int n = seq.size();
    for (int i = 0; i < n; i++) dp[0][i] = {in[seq[i]], seq[i]};
    for (int i = 1; (1 << i) <= n; i++)
        for (int j = 0; j + (1 << i) - 1 < n; j++)
            dp[i][j] = min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
}
int lca(int u, int v)
{
    int l = in[u], r = in[v];
    if (l > r) swap(l, r);
    int k = 31 - __builtin_clz(r - l + 1);
    return min(dp[k][l], dp[k][r - (1 << k) + 1]).Y;
}
```

2.7 欧拉回路

2.7.1 判定

2.7.2 求解

2.8 哈密顿回路

3 数据结构 Data Structure

3.1 并查集 Union-Find Set

```
int par[MAXN];
void init(int N)
{
    for(int i=0;i<=N;++i) par[i]=i;
}
int find(int x)
{
    if(par[x]==x) return x;
    return par[x]=find(par[x]);
}
void uni(int A,int B)
{
    int x=find(A),y=find(B);
    if(x==y) return ;
    par[x]=y;
}
bool same(int A,int B)
{
    return find(A)==find(B);
}
//按秩合并
void unite(int x,int y)
{
    x=find(x),y=find(y);
    if(x==y) return ;
    if(rank[x]<rank[y])
        parent[x]=y; // 从 rank 小的向 rank 大的连边
    else
    {
        parent[y]=x;
        if(rank[x]==rank[y]) rank[x]++;
    }
}
//非递归路径压缩 (避免栈溢出 RE)
int find(int x)
{
```

```
int k, j, r;
r = x;
while(r != parent[r])    //查找跟节点
    r = parent[r];        //找到跟节点, 用 r 记录下
k = x;
while(k != r)            //非递归路径压缩操作
{
    j = parent[k];        //用 j 暂存 parent[k] 的父节点
    parent[k] = r;        //parent[x] 指向跟节点
    k = j;                //k 移到父节点
}
return r;                //返回根节点的值
}
```

3.2 拓扑排序 Topological Sorting

```
struct Topo
{
    vector<int> G[MAXN];
    int in[MAXN], ans[MAXN]; //ans 得到拓扑排序后点编号顺序
    int tot;
    void init(int N)
    {
        for(int i=0; i<=N; ++i) G[i].clear();
        memset(in, 0, sizeof(in));
        memset(ans, 0, sizeof(ans));
        tot=0;
    }
    void addEdge(int u, int v)
    {
        G[u].push_back(v);
        ++in[v];
    }
    void topo()
    {
        priority_queue<int, vector<int>, greater<int>> > q;
        for(int i=1; i<=N; ++i)
            if(!in[i]) q.push(i);
        while(!q.empty())
        {

```

```

        int u=q.top();q.pop();
        ans[total++]=u;
        for(int i=0;i<G[u].size();i++)
        {
            int v=G[u][i];
            if((--in[v])==0)
                q.push(v);
        }
    }
};

```

3.3 树状数组

`int bit[maxn],n,m;` // n 下界, m 右界, `bit` 信息, `getsum` 求前缀和。

`inline int lowbit(int x) { return x&(-x); }`

// 一维, 区间段求和 $[1] \sim [pos]$ 解决逆序对、连线交叉点等问题。非线性排列可通过 *dfs* 序、树链剖分等转化为线性排列。

```

void add(int pos,int val)
{
    for(int i=pos;i<=n;i+=lowbit(i))
        bit[i]+=val;
}

```

```

int getsum(int pos)
{
    int sum=0;
    for(int i=pos;i>0;i-=lowbit(i))
        sum+=bit[i];
    return sum;
}

```

// 二维, 矩阵块求和 $[1,1] \sim [x,y]$ 。解决矩形图点更新, 区域求和等二维问题。

```

void add(int x,int y,int val)
{
    for(int i=x;i<=n;i+=lowbit(i))//i,x 为行方向
        for(int j=y;j<=m;j+=lowbit(j))//j,y 为列方向
            bit[i][j]+=val;
}

int getsum(int x,int y)
{
    int sum=0;

```

```
    for(int i=x;i>0;i-=lowbit(i))
        for(int j=y;j>0;j-=lowbit(j))
            sum+=bit[i][j];
    return sum;
}
```

3.4 RMQ

```
int dp[maxn][maxn],s[maxn];
//储存区间段的最值信息等
void RMQ_init()
{
    //注意编号起始位置
    //依据题进行取值
    for(int i=0;i<n;i++) dp[i][0]=s[i];

    for(int j=1;(1<<j)<=n;j++)
        for(int i=0;i+(1<<j)-1<n;i++)
            dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}

int RMQ(int l,int r)
{
    int k=0;
    while((1<<(k+1))<=r-l+1)k++;
    return min(dp[l][k],dp[r-(1<<k)+1][k]);
}
```

3.5 线段树 Segment Tree

```
#define lson rt<<1
#define rson rt<<1|1
#define Lson l,mid,lson
#define Rson mid+1,r,rson
int sum[maxn<<2],lz[maxn<<2];
// 注意四倍空间，延迟更新标记的意义，~=1 还是用作其它用处
void pushUp(int rt) { sum[rt]=sum[lson]+sum[rson]; }
void build(int l,int r,int rt)
{
    lz[rt]=0; // 延迟更新标记初始化
    if(l==r)
    {
```

```
        scanf("%d",&sum[rt]);
        return ;
    }
    int mid=(l+r)>>1;
    build(Lson);
    build(Rson);
    pushUp(rt);
}

void update_point(int p,int val,int l,int r,int rt)
{
    if(r==l)
    {
        sum[rt]+=val;
        return;
    }
    int mid=(l+r)>>1;
    if(p<=mid) update(p,val,Lson);
    else update(p,val,Rson);
    pushUp(rt);
}

void pushDown(int rt,int len)
{
    if(lz[rt]==0) return ;
    sum[lson]=lz[rt]*(len-(len>>1));
    sum[rson]=lz[rt]*(len>>1);
    lz[lson]=lz[rson]=lz[rt];
    lz[rt]=0;
}

void update_range(int L,int R,int val,int l,int r,int rt)
{
    if(L<=l && r<=R)
    {
        lz[rt]=val;
        sum[rt]=val*(r-l+1); //注意所做操作
        return ;
    }
    pushDown(rt,r-l+1);
    int mid=(l+r)>>1;
    if(L<=mid) update(L,R,val,Lson);
```



```

    if(mid<R) update(L,R,val,Rson);
    pushUp(rt);
}
int query_range(int L,int R,int l,int r,int rt)
{
    if(L<=l && r<=R) return sum[rt];
    pushDown(rt,r-l+1);
    int mid=(l+r)>>1;
    int s=0;
    if(L<=mid) s+=query(L,R,Lson);
    if(mid<r) s+=query(L,R,Rson);
    return s;
}

```

3.6 树链剖分 HeavyLightDecomposition

```

struct HLD
{
    vector<int> G[MAXN];
    // 对 i: sz 以 i 为根子树大小, dep 深度, par 父亲, son 重儿子, top 所在链顶, id 入栈
    // 序 ([题目编号]= 树链编号)
    int sz[MAXN],dep[MAXN],par[MAXN],son[MAXN],top[MAXN],id[MAXN];
    /* 题目相关信息自己设, ### 注意输入编号、映射编号、数据结构编号间的映射和转换。### */
    int n,clk;
    void init(int n)
    {
        for(int i=0;i<=n;++i) G[i].clear();
        memset(son,0,sizeof(son)); // 如果某结点没有儿子会被之前的数据影响
        this->n=n,clk=0;
    }
    void addedge(int u,int v) { G[u].push_back(v);G[v].push_back(u); }
    void getson(int u,int pre) // 标记深度、父亲、子树大小和重儿子, 调用根getson(1,0);
    {
        dep[u]=dep[pre]+1,par[u]=pre,sz[u]=1;
        int fat=0;
        for(auto &v:G[u])
        {
            if(v==pre) continue;
            getson(v,u);
            sz[u]+=sz[v];
        }
    }
}

```

```

        if(sz[v]>fat) fat=sz[v],son[u]=v;
    }
}

void dfs(int u,int up)// 标记链顶、入栈序, dfs(1,1);
{
    top[u]=up,id[u]=++clk,reflect[clk]=u;
    if(son[u]==0) return ;// 已经到达叶子
    dfs(son[u],up);// 每次先走重儿子, 重儿子同样在该重链上, 链顶相同
    for(auto &v:G[u])// 其它轻儿子的链顶为其本身
        if(v!=son[u] && v!=par[u]) dfs(v,v);
}

// 数据结构相关操作, 一般线段树或树状数组 (维护一段连续区间)
// 注意: 更新和查找操作要用对应的 dfs 序号 id[pos], 可另外写个接口用于在外面调用, 和
↪ DS 分开
// 接口里面二次调用一定要 id[pos] 啊啊啊啊啊啊啊啊猪!!!

int query(int u,int v)
{
    int ans=0;
    while(top[u]!=top[v])// 先努力跳到同一根链上
    {
        if(dep[top[u]]<dep[top[v]]) swap(u,v);// 让链顶深的往上跳
        // 因为同一根链上是一段连续区间, 所以可以直接调维护的数据结构的查询操作了
        ans=max(ans,dsquery(id[top[u]],id[u],1,n,1));// 查要上跳的点所在链的信息
        u=par[top[u]);// 然后跳出这条链, 上跳到该链链顶的父亲
    }
    // 此时 u,v 已经在同一条链上, 又可以直接调用维护信息的数据结构的查询操作了 qaq
    if(dep[u]>dep[v]) swap(u,v);// 记深度小的点为 u (令其 dfs 入栈序小)
    //此时的 u 应该是原来 u,v 的 LCA 了, 因此注意若边权下放点权要去掉 LCA
    ans=max(ans,dsquery(id[u],id[v],1,n,1));// 即应变为 dsqmax(id[son[u]],id[v])
    return ans;
}
};

```

3.7 伸展树 splay

3.7.1 维护序列

```

#define aim ch[ch[rt][1]][0]
// 维护序列的 splay, splay 上编号为序列下标

```

```
struct Splay
{
    int val[maxn],mx[maxn],lz[maxn],rev[maxn],sz[maxn],ch[maxn][2];
    // 结点值,最大值,标记区间加、翻转,子树大小,左右子结点编号
    int par[maxn],rt;// 各结点父亲编号, splay 树的根结点编号
    void newNode(int id,int v)
    {
        val[id]=mx[id]=v,sz[id]=1;
        lz[id]=rev[id]=ch[id][0]=ch[id][1]=0;
    }
    void init(int n)
    {
        newNode(0,-inf),newNode(1,-inf),newNode(n+2,-inf);
        for(int i=2;i<=n+1;++i) newNode(i,0);
        rt=build(1,n+2),par[rt]=0;
        par[0]=0,sz[0]=0,ch[0][1]=rt;
    }

    void pushup(int pos)
    {
        mx[pos]=val[pos],sz[pos]=1;
        int &l=ch[pos][0], &r=ch[pos][1];
        if(l) mx[pos]=max(mx[pos],mx[l]),sz[pos]+=sz[l];
        if(r) mx[pos]=max(mx[pos],mx[r]),sz[pos]+=sz[r];
    }

    int build(int l,int r)
    {
        if(l>r) return 0;
        if(l==r) return l;
        int mid=(l+r)>>1,ls,rs;
        ch[mid][0]=ls=build(l,mid-1);
        ch[mid][1]=rs=build(mid+1,r);
        par[ls]=par[rs]=mid;
        pushup(mid);
        return mid;
    }

    void pushdown(int pos)
```

```
{
    if(pos==0) return ;
    int &l=ch[pos][0], &r=ch[pos][1];
    if(lz[pos])
    {
        int &w=lz[pos];
        if(l) val[l]+=w,mx[l]+=w,lz[l]+=w;
        if(r) val[r]+=w,mx[r]+=w,lz[r]+=w;
        w=0;
    }
    if(rev[pos])
    {
        if(l) rev[l]^=1;
        if(r) rev[r]^=1;
        swap(l,r);
        rev[pos]=0;
    }
}

int find(int index)// 找到序列里 index 在 splay 树中对应的编号
{
    int u=rt;
    pushdown(u);
    while(sz[ch[u][0]]!=index)
    {
        int lsz=sz[ch[u][0]];
        if(index<lsz) u=ch[u][0];
        else index-=lsz+1,u=ch[u][1];
        pushdown(u);
    }
    return u;
}

void rotate(int pos,int type)// type=1 右旋,type=0 左旋
{
    int p=par[pos],gp=par[p];// pos 的父亲爷爷
    int &son=ch[pos][type];// pos 要动的那个结点
    ch[p][!type]=son,par[son]=p;
    son=p, par[p]=pos;
    ch[gp][ch[gp][1]==p]=pos,par[pos]=gp;
```

```
    pushup(p);
}
void splay(int pos,int goal)// pos 转到 goal 的右儿子
{
    if(pos==goal) return ;
    while(par[pos]!=goal)
    {
        int p=par[pos],gp=par[p];
        pushdown(gp),pushdown(p),pushdown(pos);
        int typepos=ch[p][0]==pos,typep=ch[gp][0]==p;
        // 左儿子右旋, 右儿子左旋
        if(gp==goal) rotate(pos,typepos);
        else
        {
            if(typepos==typep) rotate(p,typep);
            else rotate(pos,typepos);
            rotate(pos,typep);
        }
    }
    pushup(pos);
    if(goal==0) rt=pos;
}

void select(int l,int r)// 此时 r+1 的左儿子就是操作区间 [l,r]
{
    int u=find(l-1),v=find(r+1);
    splay(u,0),splay(v,u);
}

void update(int l,int r,int value)
{
    select(l,r);
    mx[aim]+=value,val[aim]+=value,lz[aim]+=value;
}

void reverse(int l,int r) { select(l,r); rev[aim]^=1; }
int query(int l,int r) { select(l,r); return mx[aim]; }
}t;
```

3.7.2 平衡树

4 数学 Math

4.1 定理公式与结论 Conclusions

- 1、费马小定理: $a^{p-1} \equiv 1 \pmod{p}$ 当 $(a, p) = 1$
- 2、欧拉降幂公式:

$$a^x \pmod{p} = \begin{cases} a^{x \% \varphi(p)}, & (a, p) = 1. \\ a^{x \% \varphi(p) + \varphi(p)}, & (a, p) \neq 1, x \geq \varphi(p). \\ a^x, & (a, p) \neq 1, x < \varphi(p). \end{cases} \quad (1)$$

4.2 快速乘-快速幂

```
//防止数太大 ll*ll 爆 ll
ll Mul(ll a,ll b,ll mod)
{
    ll t=0;
    for(;b>=>1,a=(a<<1)%mod)
        if(b&1) t=(t+a)%mod;
    return t;
}

typedef long long ll;
//边乘边模
ll fast(ll base,ll exp)
{
    ll ans=1;
    while(exp)
    {
        if(exp&1) ans=ans*base%mod;
        base=base*base%mod;
        exp>>=1;
    }
    return ans%mod;
}
```

4.3 矩阵快速幂

```
const int N;
struct matrix
{
    long long mat[N][N];
};
```

```
matrix operator *(matrix a,matrix b)
{
    matrix c;
    memset(c.mat,0,sizeof(c.mat));
    for(int k=0;k<N;k++)
        for(int i=0;i<N;i++)
        {
            if(a.mat[i][k]==0)
                continue;
            for(int j=0;j<N;j++)
            {
                if(b.mat[k][j]==0)
                    continue;
                c.mat[i][j]=(c.mat[i][j]+(a.mat[i][k]*b.mat[k][j])%mod)%mod;
            }
        }
    return c;
}

matrix operator ^(matrix a,int n)
{
    matrix c;
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            c.mat[i][j]= (i==j);
    while(n)
    {
        if(n&1)
            c=c*a;
        a=a*a;
        n>>=1;
    }
    return c;
}
```

4.4 扩展欧几里得

```
//d 最小公倍数, 解方程  $ax+by=gcd(a,b)$ 
//对于方程  $ax+by=c$ ; 要求  $c$  能被  $gcd(a,b)$  整除
void exgcd(int a,int b,int &d,int &x,int &y)
{
```

```
    if(!b)
    {
        x=1;y=0;d=a;
    }
    else
    {
        exgcd(b,a%b,d,y,x);
        y-=a/b*x;
    }
}
```

4.5 欧拉函数

```
const int maxn=1e5+5;
struct Num
{
    int count;//每个数的质因数个数
    int prime[16];//每个数的质因数数组
}N[maxn];
int Elur[maxn]; //欧拉函数值
void ELUR()//欧拉函数
{
    Elur[1]=1;
    for(int i=0;i<=1e5;i++)
        N[i].count=0;
    for(int i=2;i<=1e5;i++)
    {
        if(!Elur[i])
        {
            for(int j=i;j<=1e5;j+=i)
            {
                if(!Elur[j])Elur[j]=j;

                Elur[j]=Elur[j]*(i-1)/i;
                N[j].prime[N[j].count]=i;
                N[j].count++;
            }
        }
    }
}
```



```
int main()
{
    ELUR();
    for(int i=1;i<=20;i++)
    {
        cout<<N[i].count<<endl;
        int c=0;
        while(N[i].prime[c])
            cout<<N[i].prime[c++]<<" ";
        cout<<endl;
    }
}
```

4.6 中国剩余定理求同余方程组

4.6.1 素数

```
const int maxn=1e5+5;
int prime[maxn],r[maxn];
//中国剩余定理（除数两两互质）
//r[i]=x%prime[i],r[i] 存余数,a[i] 存被除数
void exgcd(int a,int b,int &d,int &x,int &y)
{
    if(!b)
    {
        x=1;y=0;d=a;
    }
    else
    {
        exgcd(b,a%b,d,y,x);
        y-=a/b*x;
    }
}
int Chinese_Remainder()
{
    int M=1;
    for(int i=1;i<=n;i++)
        M*=prime[i]; //所有除数最小公倍数
    int d,x,y,answer=0;
    for(int i=1;i<=n;i++)
    {
```

```
    int m=M/prime[i];
    exgcd(prime[i],m,d,x,y);
    answer=(answer+y*m*r[i])%M;
}
return (M+answer%M)%M;
```

4.6.2 非素数

```
const int maxn=1e5+5;
int c[maxn],r[maxn];
int n;
//模线性同余方程组 (CRT 非素数)
//两两方程结合法
//r[i]=x%chu[i],r[i] 存余数,chu[i] 存除数
void exgcd(int a,int b,int &d,int &x,int &y)
{
    if(!b)
    {
        x=1;y=0;d=a;
    }
    else
    {
        exgcd(b,a%b,d,y,x);
        y-=a/b*x;
    }
}

int Chinese_Remainder()
{
    int c1=c[1],r1=r[1];
    //a1,r1 为合并项
    for(int i=2;i<=n;i++)
    {
        int c2=c[i],r2=r[i];
        //a2,r2 为当前项
        int d,x,y,p=r2-r1;

        exgcd(c1,c2,d,x,y);

        if(p%d) return -1;
```

```
        int z=c2/d;
        x=(x*(p/d)%z+z)%z;
        r1=x*c1+r1;
        c1=c1*(c2/d);
        r1=(r1%c1+c1)%c1;
    }
    return (r1%c1+c1)%c1;
}
//队长后面是我测试的
int main()
{
    cin>>n;

    for(int i=1;i<=n;i++)
    {
        cin>>c[i]>>r[i];
    }
    cout<<Chinese_Remainder()<<endl;
}
```

5 字符串 String

5.1 字典树 Trie

```
struct Trie{
    int ch[maxnode][sigma_size];
    int val[maxnode];
    int sz;
    void clear()
    {
        sz=1;
        memset(ch[0],0,sizeof(ch[0]));
        memset(val,0,sizeof(val));
    }
    int idx(char c)
    {
        return c-'a';
    }
    void insert(const char*s)
    {

```

```
    int u=0,n=strlen(s);
    for(int i=0;i<n;i++)
    {
        int c=idx(s[i]);
        if(!ch[u][c])
        {
            memset(ch[sz],0,sizeof(sz));
            val[sz]=0;
            ch[u][c]=sz++;
        }
        u=ch[u][c];
        val[u]++;
    }
}

int search(const char *s)
{
    int u=0,n=strlen(s);
    for(int i=0;i<n;i++)
    {
        int c=idx(s[i]);
        if(!ch[u][c])
        {
            return 0;
        }
        u=ch[u][c];
    }
    return val[u];
}
}ans;
```

5.2 KMP

```
struct kmp{
    int s[maxN];
    int p[maxM];
    int f[maxM];
    void getfail(int *p,int *f)
    {
        int m=M;
        f[0]=0;
```

```
    f[1]=0;
    for(int i=1;i<m;i++)
    {
        int j=f[i];
        while(j&& p[i]!=p[j])
            j=f[j];
        f[i+1]=p[i]==p[j]?j+1:0;
    }
}

int find(int *t,int *p,int *f)
{
    int n=N;
    int m=M;
    getfail(p,f);
    int j=0;
    for(int i=0;i<n;i++)
    {
        while(j&& p[j]!=t[i])
            j=f[j];
        if(p[j]==t[i])
            j++;
        if(j==m)
            return i-m+2;
    }
    return -1;
}

}ans;
```

5.3 扩展 KMP

```
struct exKMP
{
    char t[maxn];
    char p[maxn];
    int f[maxn];
    int extend[maxn];
    void getfail(char *p,int *f)
    {
        int m=strlen(p);
        f[0]=m;
```

```
    int i=0;
    while(i<m-1&& p[i]==p[i+1])
        i++;
    f[1]=i;
    int po=1;
    for(i=2;i<m;i++)
    {
        if(f[i-po]+i<po+f[po])
            f[i]=f[i-po];
        else
        {
            int j=po+f[po]-i;
            if(j<0)
                j=0;
            while((i+j<m)&& p[i+j]==p[j])
                j++;
            f[i]=j;
            po=i;
        }
    }
}

void gettextend(char *t, char *p, int *f, int *extend)
{
    int n=strlen(t);
    int m=strlen(p);
    getfail(p,f);
    int i=0;
    while(t[i]==p[i]&& i<n&& i<m)
        i++;
    extend[0]=i;
    int po=0;
    for(int i=1;i<n;i++)
    {
        if(f[i-po]+i<extend[po]+po)
            extend[i]=f[i-po];
        else
        {
            int j=extend[po]+po-i;
            if(j<0)
```

```
        j=0;
        while(i+j<n&& j<m&&t[i+j]==p[j])
            j++;
        extend[i]=j;
        po=i;
    }
}
}ans;
```

5.4 最长回文子串 Manacher

```
struct Manacher
{
    char p[maxn];
    char temp[maxn<<1];
    int f[maxn<<1];
    void init(char *p, char *temp)
    {
        int n=strlen(p);
        temp[0]='*';
        for(int i=0; i<=n; i++)
        {
            temp[i*2+1]='#';
            temp[i*2+2]=p[i];
        }
        temp[2*n+2]='\0';
    }
    void getlen(char *p, int *f)
    {
        int mx=0, po=0, ans=0;
        int n=strlen(p);
        f[0]=0;
        for(int i=2; i<n; i++)
        {
            if(mx>i)
                f[i]=min(mx-i, f[2*po-i]);
            else
                f[i]=1;
            while(p[i-f[i]]==p[i+f[i]])
                ;
        }
    }
};
```

```
        f[i]++;
        if(f[i]+i>mx)
        {
            po=i;
            mx=f[i]+i;
        }
    }
}
}ans;
```

5.5 AC 自动机

```
struct AC
{
    int ch[maxnode][sigma_size];
    int val[maxnode],f[maxnode],last[maxnode];
    int sz;
    void init()
    {
        sz=1;
        memset(ch[0],0,sizeof(ch[0]));
        memset(val,0,sizeof(val));
        memset(f,0,sizeof(f));
        memset(last,0,sizeof(last));
    }
    int idx(char c){return c-'A';}
    void insert(char *s,int id)
    {
        int n=strlen(s),u=0;
        for(int i=0;i<n;i++)
        {
            int c=idx(s[i]);
            if(!ch[u][c])
            {
                memset(ch[sz],0,sizeof(ch[sz]));
                val[sz]=0;
                ch[u][c]=sz++;
            }
            u=ch[u][c];
        }
    }
}
```



```
    val[u]=id;
}
void getfail(int *f)
{
    queue<int> q;
    f[0]=0;
    for(int c=0;c<sigma_size;c++)
    {
        int u=ch[0][c];
        if(u)
        {
            f[u]=0;
            q.push(u);
            last[u]=0;
        }
    }
    while(!q.empty())
    {
        int r=q.front();q.pop();
        for(int c=0;c<sigma_size;c++)
        {
            int u=ch[r][c];
            if(!u)
            {
                ch[r][c]=ch[f[r]][c];
                continue;
            }
            q.push(u);
            int v=f[r];
            while(v &&!ch[v][c]) v=f[v];
            f[u]=ch[v][c];
            last[u]= val[f[u]]? f[u]:last[f[u]];
        }
    }
}
void find(char *t,int *f)
{
    int n=strlen(t);
    int j=0;
```

```
    for(int i=0;i<n;i++)
    {
        if(t[i]>'Z' || t[i]<'A')
        {
            j=0;
            continue;
        }
        int c=idx(t[i]);
        j=ch[j][c];
        if(val[j]) vis[val[j]]++;
        if(last[j]) bfind(last[j]);
    }
}
void bfind(int j)
{
    if(j)
    {
        vis[j]++;
        bfind(last[j]);
    }
}
};
```

6 动态规划

6.1 背包

```
int dp[maxn];
void zeropack(int c,int v)
{
    for(int i=m;i>=c;i--)
        dp[i]=max(dp[i],dp[i-c]+v);
}
void completepack(int c,int v)
{
    for(int i=c;i<=m;i++)
        dp[i]=max(dp[i],dp[i-c]+v);
}
void multipack(int c,int v,int shu)
{
    for(int i=0;i<shu;i++)
        zeropack(c,v);
}
```

```
    if (shu*c>=m)
    {
        completepack(c,v);
        return;
    }
    int k=1;
    while(k<shu)
    {
        zeropack(k*c,k*v);
        shu-=k;
        k*2;
    }
    zeropack(shu*c,shu*v);
}
```

7 其它 Other

7.1 莫队

```
int block[MAXN], cnt[MAXN], a[MAXN];
int n, q, Ans, ans[MAXN];
struct Node
{
    int l, r, id;
    bool operator<(const Node &b) const
    {
        if(block[l]==block[b.l]) return (block[l]&1)?(r<b.r):(b.r<r);
        return block[l]<block[b.l];
    }
}ask[MAXN];
inline void add(int pos){}
inline void del(int pos){}
void Mos()
{
    // read data
    int sz=ceil(sqrt(1.0*n));
    for(int i=1;i<=q;++i)
    {
        // read l,r
        ask[i].id=i;
        block[i]=i/sz;
    }
    sort(ask+1,ask+q+1);
    // init assistant space
    int L=1,R=1;Ans=0;add(1);
    for(int i=1;i<=q;++i)
    {
        while(L<ask[i].l) del(L++);
        while(L>ask[i].l) add(--L);
        while(R<ask[i].r) add(++R);
        while(R>ask[i].r) del(R--);
        ans[ask[i].id]=Ans;
    }
}
```

7.2 离散化

```
// a 原序列, v 暂存离散化
vector<int> v=a;
sort(v.begin(),v.end());
v.resize(unique(v.begin(),v.end())-v.begin());//
for(int i=0;i<n;++i)
    a[i]=lower_bound(v.begin(),v.begin()+size,oldData)-v.begin()+1;
```

7.3 STL

```
//一、set
//set 和 multiset 用法一样, multiset 允许重复元素
//利用 set 从大到小排序 (自定义排序函数)
struct classcmp
{
    bool operator()(const int &lhs,const int &rhs)const
    {return lhs>rhs;}
};
multiset<int,classcmp> s;
//结构体自定义排序函数
struct Node
{
    int x,y;
};
struct classcmp
{
    bool operator()(const Node &a,const Node &b)const
    {
        if(a.x!=b.x) return a.x<b.x;
        else return a.y>b.y;
    }//按 x 从小到大, 按 y 从大到小
};
multiset<Node,classcmp> s;
multiset<Node,classcmp>::iterator it;//若定义迭代器也要带排序函数
//函数
count()//某个值元素的个数
erase()//删除元素 (参数为元素值或迭代器, multi 会删光值的每一个)
find()//返回元素迭代器
size()//元素数目
lower_bound()//返回指向大于 (或等于) 某值的第一个元素的迭代器
```

`upper_bound()` // 返回大于某个值元素的迭代器
`equal_range()` // 返回集合中与给定值相等的上下限两个迭代器

// 二、string

```
s1.assign(s2);
s1.assign(s2,lenth);
s1.assign(s2,start,lenth);
s1.assign(times,char1);
s1.assign(start,end);
s1.at(pos);
```

7.4 pbds

// 红黑树，不能有重复元素，有重复就 *pair* 加个捣乱值

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
↪ tree_order_statistics_node_update> rbtree;
```

/*

定义一颗红黑树

int 关键字类型

null_type 无映射 (低版本 *g++* 为 *null_mapped_type*)

less<int> 从小到大排序 *greater<int>* 从大到小

rb_tree_tag 红黑树 (*splay_tree_tag*)

tree_order_statistics_node_update 结点更新

插入 *t.insert()*;

删除 *t.erase()*;

Rank:t.order_of_key(key);

第 *K* 小值:*t.find_by_order(K-1)*; 从 0 开始

// 前驱:*t.lower_bound()*;

// 后继 *t.upper_bound()*;

a.join(b) *b* 并入 *a* 前提是两棵树的 *key* 的取值范围不相交

a.split(v,b) *key* 小于等于 *v* 的元素属于 *a*, 其余的属于 *b*

T.lower_bound(x) $\geq x$ 的 *min* 的迭代器

T.upper_bound((x) $> x$ 的 *min* 的迭代器

T.find_by_order(k) 有 *k* 个数比它小的数

*/

8 输入输出 IO

8.1 输入输出外挂

`\Large{关同步}`

```
ios_base::sync_with_stdio(0);
cin.tie(0);
```

`\Large{kuangbin的IO挂}`

```
template <class T>
inline bool scan_d(T &ret)
{
    char c;
    int sgn;
    if(c=getchar(),c==EOF) return 0;
    while(c!='-'&&(c<'0' || c>'9')) c=getchar();
    sgn=(c=='-')?-1:1;
    ret=(c=='-')?0:(c-'0');
    while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}

inline void out(int x)
{
    if(x>9) out(x/10);
    putchar(x%10+'0');
}
```

8.2 简单大数

```
struct BigInt
{
    const static int mod=10000;
    const static int DLEN=4;
    int a[600],len;
    BigInt()
    {
        memset(a,0,sizeof(a));
        len=1;
    }
    BigInt(int v)
```

```
{
    memset(a,0,sizeof(a));
    len=0;
    do
    {
        a[len++]=v%mod;
        v/=mod;
    }while(v);
}

BigInt(const char s[])
{
    memset(a,0,sizeof(a));
    int L=strlen(s);
    len=L/DLEN;
    if(L%DLEN) len++;
    int index=0;
    for(int i=L-1;i>=0;i-=DLEN)
    {
        int t=0;
        int k=i-DLEN+1;
        if(k<0) k=0;
        for(int j=k;j<=i;j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}

BigInt operator +(const BigInt &b)const
{
    BigInt res;
    res.len=max(len,b.len);
    for(int i=0;i<=res.len;i++)
        res.a[i]=0;
    for(int i=0;i<res.len;i++)
    {
        res.a[i]+=((i<len?a[i]:0))+((i<b.len)?b.a[i]:0);
        res.a[i+1]+=res.a[i]/mod;
        res.a[i]%=mod;
    }
    if(res.a[res.len]>0) res.len++;
}
```



```
        return res;
    }
    BigInt operator *(const BigInt &b) const
    {
        BigInt res;
        for(int i=0; i<len; i++)
        {
            int up=0;
            for(int j=0; j<b.len; j++)
            {
                int temp=a[i]*b.a[j]+res.a[i+j]+up;
                res.a[i+j]=temp%mod;
                up=temp/mod;
            }
            if(up!=0)
                res.a[i+b.len]=up;
        }
        res.len=len+b.len;
        while(res.a[res.len-1]==0 && res.len>1) res.len--;
        return res;
    }
    void output()
    {
        printf("%d", a[len-1]);
        for(int i=len-2; i>=0; i--)
            printf("%04d", a[i]);
        printf("\n");
    }
};
```

8.3 Python 输入输出

```
# while(scanf("%d",&n)==1)
while True:
    try:
        n=int(input())
        print(n)
    except EOFError:
        break
```

```
# while(scanf("%d%d",&a,&b)==2)
while True:
    try:
        a, b = map(int, input().strip().split()) # strip 去掉前导和末尾的空格, split
        ↪ 切片
        print (a + b, end = ' ')
        # py3 的 print 是函数一定要括号, py2 不要, 最后括号外加逗号不自动换行, end 不加
        ↪ 参数默认\n
    except EOFError:
        break

# int T;scanf("%d",&T);while(T--){scanf("%d%d",&n,&m)==2}
T = int(input().strip())
for case in range(T):
    n, m = map(int, input().strip().split())
    print(n,m)

# 条件结束
while True:
    n, m = map(int, input().strip().split())
    print(n,m)
    if n == 0 and m == 0:
        break

# n a1 a2 a3 ... an
T = int(input().strip())
for case in range(T):
    line = map(int, input().strip().split())
    n, a = line[0], line[1:]

    sum = 0
    for i in range(n):
        sum += a[i]
    print (sum)
```