Simple template

Sparken

September 19, 2018

目录

1 默背一万遍的注意事项

1.1 浮点

1. 浮点初始化 memset(d,0x7f,sizeof(d));
2. 浮点数比大小
   - 相等 if (fabs (a-b) <= eps)
   - 大于 if (a>b && fabs (a-b) > eps)
   - 小于 if (a<b && fabs (a-b) > eps)

1.2 整数类型范围

1. 255：1111 1111B
2. 65535：2^16-1，16bit 无符号整数
3. 2147483647：2^31-1，32bit 带符号整数的最大值
4. 4294967296：2^32，32bit 无符号整数的最大值
5. 9223372036854775807：2^63-1，64bit 带符号整数的最大值
6. 1061109567：0x3f3f3f3f, int inf, 略大于 1e9
7. 4557430888798830399：0x3f3f3f3f3f3f3f3f, ll inf

1.3 热身赛

1. 测 pbds
2. python3 计算器

1.4 计算器

1. 终端
   - 分解素因数 factor num
   - 逆串 rev+enter string
2. python3
   - from fractions import * [Fraction,gcd]
     - 最简分数 fraction(a,b)
     - gcd(a,b)
   - from math import *
     - 阶乘 factorial(num)

1.5 Attention

1. 审题
   - **读新题的优先级高于一切**
   - **注意限制条件**，不清楚的善用 Clarification
   - 读完题、交题前都要看一遍 clarification
   - 每题至少两人确认题意
2. 做题
   (a) **开题**
   - 构造不要开场做
   - 想不出优雅复杂度但过了很多队的**暴力**莽一莽，单车变摩托

   (b) **上机**
   - 和队友确认做法
   - 有猜想性质的后面写
   - 写了半小时以上的考虑是否弃题
   - **细节和公式纸上写好**，不要越码越乱
   - 中后期题考虑一人写一人辅助，及时发现手误
   - 多题要写时，容易码、码量小、想得无敌清楚的优先

   (c) **交题**
   - 检查初始化和清空
   - 取模的输出前再模一次
   - claris: 检查 solve(n,m)==solve(m,n)?
   - spj 的题目提交前也应尽量与样例完全一致
   - claris: 舍入输出若 abs 不超过 eps，需要强行设置 0 防止-0.000000 的出现
3. 打印
   - 交完题目马上打印并让机
   - 打表时想清楚打哪些量，代码乱改前注意备份。善用打印，保留代码。
4. **心态**：签到莫急，最后半小时不要慌。

1.6 Debug

1. 初始化，清空图，0 和 n 等边界，mem 里面 sizeof(int) 还是 ll
2. for 里是给本层循环变量 ++ 咩?
3. 区间 l,r 为防坑：if(l>r) swap(l,r);

4. 考虑小数据有没有发生突变的地方
5. 注意板子有没有哪里要改 ll
6. inf 的大小符不符合

1.7 打表找规律

1. 直接找规律
2. 差分后找规律
3. 找积性
4. 点阵打表
5. 相除
6. 循环节
7. 凑量纲
8. 猜想满足 P(n)f(n)=Q(n)f(n-2)+R(n)f(n-1)+C, 其中 P、Q、R 为关于 n 二次多项式

1.8 优化

1. 数论
   - 分块加速 O(sqrt(n))
   - 枚举除数、调和级数 O(log(n))
   - floor 函数求和、ceil 函数求和 (hdu6134)
   - getpre 里的取模，以及连续取模注意顺序，还有爆精度取模和式子 i 从 2 开始 (n/i)
   - **WA 太久或出不了，考虑公式是否错误**
2. cdq 分治
3. 树上点分治
4. 一般分块

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

inputminted[breaklines]c++"Gragh Theory/The shortest path/secdij.cpp"

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)    //dijstra 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*N 有向无环图，拆点，i->j ==> i1->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 ST 表

```
// 欧拉序列 +ST 表，时间复杂度 O(2nlog(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

```
stack<int> s;
void dfs(int u)
{
```

```
    for(auto &v:G[u])
    {
        if(!vis[u][v])
        {
            vis[u][v]=1;// 有向图
            dfs(v);
        }
    }
    s.push(u);
}
```

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]~[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]~[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;y<=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 表达式树

```
int lch[maxn],rch[maxn];
char op[maxn];
int nc=0;// 结点数
int build_tree(char*s,int x,int y)
{
     int i,c1=-1,c2=-1,p=0;
     int u;
     if(y-x==1)
     {
         u=++nc;
         lch[u]=rch[u]=0;op[u]=s[x];
         return u;
```

```
    }
    for(i=x;i<y;i++)
    {
        switch(s[i])
        {
            case'(':p++;break;
            case')':p--;break;
            case'+':case'-':if(!p)c1=i;break;
            case'*':case'/':if(!p)c2=i;break;
        }
    }
    if(c1<0) c1=c2;
    if(c1<0) return build_tree(s,x+1,y-1);
    u=++nc;
    lch[u]=build_tree(s,x,c1);
    rch[u]=build_tree(s,c1+1,y);
    op[u]=s[c1];
    return u;
}
```

3.6 线段树 Segment Tree

3.6.1 基础

```
#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;
}
```

3.6.2 维护线性变化

```
void pushdown(ull len,int rt)
{   // lz+,mul*
    if(mul[rt]!=1 || lz[rt])
    {
        lz[lson]=(lz[lson]*mul[rt])+lz[rt];
        lz[rson]=(lz[rson]*mul[rt])+lz[rt];
        mul[lson]=mul[lson]*mul[rt];
        mul[rson]=mul[rson]*mul[rt];
        sum[lson]=(sum[lson]*mul[rt]+lz[rt]*(len-(len>>1)));
        sum[rson]=(sum[rson]*mul[rt]+lz[rt]*(len>>1));
        mul[rt]=1;
        lz[rt]=0;
    }
}
```

3.7 可持久化数据结构

3.7.1 01 字典树

```
// 可持久化 01 字典树
const int maxn=1e5+5;
int ch[maxn*20][2],cnt[maxn*20],rt[maxn];
int sz,a[maxn];
void inittree()
{
    sz=0,cnt[0]=0;
    memset(ch[0],-1,sizeof(ch[0]));
    memset(cnt,0,sizeof(cnt));
}
// 维护每个点到根这条路径上所有点的权值构成的字典树
```

```
int insert(int old,int val)
{
    ++sz;
    int entry=sz,dad=sz;// 版本入口，父亲结点（旧版本继承来的）
    ch[sz][0]=ch[old][0],ch[sz][1]=ch[old][1];
    cnt[sz]=0;
    // cnt[sz]=cnt[old];
    for(int i=16;i>=0;--i)
    {
        int bit=(val>>i)&1;
        int newnode=++sz;
        // 创建新结点，先继承旧版本的对应结点过来（此时父亲还是旧版的没有更改过）
        // 继承的时候一定要注意继承对啊你是猪吗啊啊啊啊啊啊啊啊!!!!!!!!
        ch[newnode][0]=ch[ch[dad][bit]][0],ch[newnode][1]=ch[ch[dad][bit]][1];
        cnt[newnode]=cnt[ch[dad][bit]];
        ++cnt[newnode];// 更新这个结点记录的个数
        // printf("id=%d bit=%d cnt=%d\n",newnode,bit,cnt[newnode]);
        ch[dad][bit]=newnode;// 把本版本的父亲连到这个新结点
        dad=newnode;// 为向下更新做准备
    }
    return entry;
}
// 求 u->v 路径上除了 lca 的所有点权中和 z 异或值最大的结果值
int query(int u,int v,int lca,int z)
{
    int ans=0;
    for(int i=16;i>=0;--i)
    {
        int bit=(z>>i)&1;
        int num=cnt[ch[u][bit^1]]+cnt[ch[v][bit^1]]-2*cnt[ch[lca][bit^1]];
        // printf("digit=%d bit=%d
        ↪  num=%d+%d-%d=%d\n",i,bit,cnt[ch[u][bit^1]],cnt[ch[v][bit^1]],2*cnt[ch[lca][bit^1]],num
        if(num>0) ans|=(1<<i), bit^=1;
        // 这一位有跟它相反的就往那走（bit^=1），否则只能走另一个方向
        u=ch[u][bit],v=ch[v][bit],lca=ch[lca][bit];
    }
    // printf("query ans=%d\n",ans);
    return ans;
}
```

```
void build(int u)
{
    rt[u]=insert(rt[par[u]],a[u]);
    for(auto &v:G[u])
        if(v!=par[u]) build(v);
}
```

3.7.2 权值线段树

```
// 可持久化权值线段树
const int maxn=1e5+5;
int lson[maxn*20],rson[maxn*20],sum[maxn*20];
// lson[i],rson[i] 为结点 i 左右子树编号,sum 维护的信息, 本题中为所管辖数字区间内的数出
↪  现多少次
int a[maxn],rt[maxn],cnt;
int n,m;
int id(int x){return lower_bound(temp.begin(),temp.end(),x)-temp.begin()+1;}
void update(int l,int r,int value,int pre,int &cur)
/**
 * 当前维护的区间 l,r 和要去更新的 value,
 * 要移植的前置版本在该处的编号 pre, 现正创的新版本的当前结点编号 cur (更新时给它打编号所
↪  以引用)
**/
{
    // 复制前置版本的信息
    ++cnt;
    lson[cnt]=lson[pre],rson[cnt]=rson[pre],sum[cnt]=sum[pre];
    // 因为要从这条分岔下去更新, 所以这里的信息要变化
    ++sum[cnt];
    // 记录这个新结点的编号
    cur=cnt;
    if(l==r) return ;// 到底, 更新完毕
    int mid=(l+r)>>1;// 否则继续向下
    if(value<=mid) update(l,mid,value,lson[pre],lson[cur]);
    else update(mid+1,r,value,rson[pre],rson[cur]);
}
int query(int l,int r,int k,int pre,int last)
/**
 * 当前查询的区间范围 l,r 和要查的第 k 大
 * 查询的版本左右两端编号
```

```
**/
{
    if(l==r) return l;// 到底，找到所查第 k 大的值（离散化后）
    int mid=(l+r)>>1;
    // 否则看右端版本小的一半数有多少个，减去左端版本的个数，即查询版本内小的那一半数有多
    ↪ 少个
    int s=sum[lson[last]]-sum[lson[pre]];
    // 如果个数比 k 大，说明第 k 大个在小的那半数里，也就是左子树中
    if(s>=k) return query(l,mid,k,lson[pre],lson[last]);
    else return query(mid+1,r,k-s,rson[pre],rson[last]);
    // 个数比 k 小，转化为在大的那半数里求第 k-sum 大的数
}
```

3.8 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;
    }
};
```

3.9 伸展树 splay

3.9.1 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 要动的那个子结点
    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;
```

3.9.2 平衡树

4 数学 Math

4.1 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;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;
}
```

4.2 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;
}
```

4.3 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;
```

```
    }
}
```

4.4 筛法求素数

4.4.1 埃式筛

```
// O(nloglogn) 筛出 maxn 内所有素数
// notprime[i] = 0/1 0 为素数 1 为非素数
bool notprime[maxn] = {1, 1};   // 0 && 1 为非素数
void GetPrime()
{
    for (int i = 2; i < maxn; i++)
        if (!notprime[i] && i <= maxn / i)  // 筛到 √n 为止
            for (int j = i * i; j < maxn; j += i)
                notprime[j] = 1;
}
```

4.5 欧拉筛线性筛

```
/* 求一个数的 phi, 时间复杂度 sqrt(n) */
inline ll Phi(ll num)
{
    int ans = num;
    for (int i = 2; i * i <= num; ++i)
        if (num % i == 0)
        {
            ans -= ans / i;
            while (num % i == 0)
                num /= i;
        }
    if (num > 1)
        ans -= ans / num;
    return phi[num] = ans;
}

/*
 * 线性筛 O(n) 得 1e7 内所有数的欧拉函数 phi[]、素数表 prime[]、素数个数 tot
 * 不要 phi 时可以把所有关于 phi 的表达式去掉 (别去 break), 传入的 n 为函数定义域上界。
 */
bool vis[maxn];
```

```
int tot, phi[maxn], prime[maxn];
void CalPhi(int n)
{
    memset(vis, 0, sizeof(vis));
    phi[1] = 1;
    tot = 0;
    for (int i = 2; i < n; i++)
    {
        if (!vis[i]) prime[tot++] = i, phi[i] = i - 1;
        for (int j = 0; j < tot; j++)
        {
            if (i * prime[j] > n) break;
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
}
```

4.5.1 区间筛

```
/*
 * [a,b) 区间筛（长度 <1e6，a、b 范围 1e12），函数返回区间内素数个数
 * is_prime[i-a]=true 表示 i 是素数，把 1 当素数了，记得特判。
 */
bool is_prime_small[maxn], is_prime[maxn];
int prime[maxn];
int segment_sieve(ll a, ll b)
{
    int tot = 0;
    for (ll i = 0; i * i < b; ++i)
        is_prime_small[i] = true;
    for (ll i = 0; i < b - a; ++i)
        is_prime[i] = true;
    for (ll i = 2; i * i < b; ++i)
        if (is_prime_small[i])
```

```
        {
            for (ll j = 2 * i; j * j < b; j += i)
                is_prime_small[j] = false;
            for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
                is_prime[j - a] = false;
        }
    for (ll i = 0; i < b - a; ++i)
        if (is_prime[i]) prime[tot++] = i + a;
    return tot;
}
```

4.6 逆元

4.6.1 模 n 下 a 的逆元

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
void exgcd(ll a,ll b,ll &d,ll &x,ll &y)
{
    if(!b)
    {
        x=1;y=0;d=a;
    }
    else
    {
        exgcd(b,a%b,d,y,x);
        y-=a/b*x;
    }
}
ll inv(ll a,ll n)
{
    ll d,x,y;
    exgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}
int main()
{
    ll a,b;
    while(cin>>a>>b)
    {
```

```
        cout<<inv(a,b)<<endl;
    }
}
```

4.6.2 线性求逆元

```
// 线性求 1-(p-1)modp 的逆元
int main()
{
    A[i]=-(p/i)*A[p%i];
    inv[i]=(p-(p/i))*inv[p%i]%p;
}
```

4.7 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;
        }
}
```

4.8 中国剩余定理求同余方程组

4.8.1 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;
```

4.8.2 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;
}
```

4.9 数值计算

4.9.1 FFT

```
#include<bits/stdc++.h>
//    j
using namespace std;
const double PI = acos(-1);
struct A{
    double r,i;
    A(double r = 0, double i = 0):r(r), i(i){}
}a[100],b[100];
A operator + (const A& x, const A& y){
    return A(x.r + y.r, x.i + y.i);
}
A operator - (const A& x, const A&y){
    return A(x.r - y.r, x.i - y.i);
}
A operator * (const A& x, const A&y){
    return A(x.r * y.r - x.i *y.i, x.r * y.i + x.i * y.r);
```

```
}

void FFT(A x[], int n,int p)
{
    for(int i = 0, t = 0; i < n; ++i){
       if(i > t)swap(x[i], x[t]);
       for(int j = n >> 1; (t ^= j) < j; j >>= 1);
       //¶½ ´
    }
    for(int h = 2; h <= n; h <<= 1){
       A wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
       //±n¸      2µ
       for(int i = 0; i < n; i += h){
          A w(1,0),u;
          for(int j = i, k = h>>1; j < i + k; ++j){
             u = x[j + k] * w;
             x[j + k] = x[j] - u;
             x[j] = x[j] + u;
             w = w * wn;
          }
       //      °±³½      ;
       }
    }
    if(p == -1)
        for(int i = 0; i < n; ++i)
           x[i].r /= n;
}
void conv(A a[], A b[], int n){
    FFT(a, n, 1);
    FFT(b, n, 1);
    for(int i = 0; i < n; ++i)
       a[i] = a[i] * b[i];
    FFT(a, n, -1);
}
int main()
{
     int n,m;
     scanf("%d%d",&n,&m);
     int N = 1;
```

```
    while(N < n + m - 1)N <<= 1;
    conv(a, b, N);
}
```

4.9.2 FFT 二进制反转问题

```
void get_rev(int bit)//bit± º¶½
{
    for(int i=0;i<(1<<bit);i++)// ô¶1~2^bit-1      IbitµÄ½
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));//?!! SMG ?!!
}
//  õdpµ
```

4.9.3 NTT

```
const ll MOD = 998244353;
const ll G = 3;
const int N = 15;
ll wn[N << 2],rev[N << 2];
ll fmod(ll x, ll y, ll z)
{
    ll ans = 1;
    while(y)
    {
        if(y & 1)ans = ans * x % z;
        y >>= 1;
        x = x * x % z;
    }
    return ans;
}
int NTT_init(int n_)
{
    int step = 0; int n = 1;
    for( ; n < n_; n <<= 1) ++step;
    for(int i = 1; i < n; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
    // 蝴蝶操作，二进制反转
    int g = fmod(G, (MOD - 1) / n, MOD);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % MOD;
```

```
    // 求出 n 次单位根
    return n;
}
void NTT(ll a[], int n, int f)
{
    for(int i = 0; i < n; ++i)
        if(i < rev[i])swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);
            for(int j = 0; j < k; ++j){
                ll w = f == 1 ? wn[t * j] : wn[n - t * j];
                ll x = a[i + j];
                ll y = a[i + j + k] * w % MOD;
                a[i + j] = (x + y) % MOD;
                a[i + j + k] = (x - y + MOD) % MOD;
            }
        }
    }
    if(f == -1)
    {
      ll ninv = fmod(n, MOD - 2, MOD);
      for(int i = 0; i < n; ++i)
         a[i] = a[i] * ninv % MOD;
    }
}
```

4.10 卢卡斯 Lucas

4.10.1 Lucas

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9+7;
ll fac[100005];
ll fmod(ll x, ll y){ll res = 1;while(y){if(y&1)res=res*x%mod;y>>=1;x=x*x%mod;}return
↪  res;}
ll facinit()
{
    fac[0] = fac[1] = 1;
```

```
    for(ll i = 2; i <= 100000; ++i)
        fac[i] = fac[i - 1]*i%mod;
}
ll C(ll a, ll b)
{
    if(b > a)return 0;
    return fac[a]*fmod(fac[b]*fac[a-b], mod - 2LL)%mod;
}
ll lucas(ll a,ll b)
{
    if(!b)return 1;
    return C(a%mod, b%mod)*lucas(a/mod, b/mod)%mod;
}
int main()
{
    ll x,y;
    facinit();
    while(cin>>x>>y)
        cout<<lucas(x,y)<<endl;
}
```

4.10.2 扩展卢卡斯 ExLucas

```
const int maxn = 1e5+5;
const ll mod = 1e9+7;
ll fmod(ll x,ll y, ll M){ll res=1;while(y){if(y&1)res=res*x%M;y>>=1;x=x*x%M;}return
↪  res;}
void exgcd(ll a,ll b,ll &d,ll &x,ll &y)
{
    if(!b){
        x=1;y=0;d=a;}
    else{
        exgcd(b,a%b,d,y,x);
        y-=a/b*x;}
}
ll inv(ll a,ll M)
{
    if(!a)return 0LL;
    ll d,x = 0LL,y = 0LL;
    exgcd(a,M,d,x,y);
```

```
    x = ((x%M) + M)%M;
    if(!x)x += M;
    return x;
}
ll mul(ll n, ll pi, ll pk)
{
    if(!n)return 1LL;
    ll ans = 1LL;
    if(n/pk)
    {
        for(ll i = 2; i <= pk; ++i)
            if(i%pi)ans = ans*i%pk;
        ans = fmod(ans, n/pk, pk);
    }
    for(ll i = 2; i <= n%pk; ++i)
        if(i%pi)ans = ans*i%pk;
    return ans * mul(n/pi,pi,pk)%pk;
}
ll C(ll a, ll b, ll M, ll pi, ll pk)
{
    if(b > a)return 0LL;
    ll x = mul(a,pi,pk),y = mul(b,pi,pk),z = mul(a - b,pi,pk);
    ll cnt = 0LL,ans;
    for(ll i = a; i; i /= pi)cnt += i/pi;
    for(ll i = b; i; i /= pi)cnt -= i/pi;
    for(ll i = a - b; i; i /= pi)cnt -= i/pi;

    ans = x * inv(y,pk)%pk * inv(z,pk)%pk *fmod(pi,cnt,pk)%pk;
    return ans * (M/pk)%M * inv(M/pk, pk)%M;
}
ll ex_lucas(ll a, ll b, ll M)
{
    ll ans = 0;
    for(ll x = M,i = 2; i <= M; ++i)
        if(x%i == 0)
        {
            ll pk = 1LL;
            while(x%i == 0)pk *= i,x /= i;
            ans = (ans + C(a,b,M,i,pk))%M;
```

```
        }
        return ans;
}
int main()
{
   ll a,b,m;
   while(cin>>a>>b>>m)
    cout<<ex_lucas(a,b,m)<<endl;
}
```

4.11 线性基

```
struct l_B
{
    ll d[61],b[61]; int cnt = 0;
    void init() { clr(d,0), clr(b,0), cnt = 0; }
    bool insrt(ll val)
    {
        for(int i = 60; i >= 0; i--)
        {
            if(!d[i])
            {
                d[i] = val;
                break;
            }
            val ^= d[i];
        }
        return val > 0;
    }
    ll get_max()
    {
        ll ans = 0;
        for(int i = 60; i >= 0; --i)
            if((ans^d[i] > ans))
                ans ^= d[i];
        return ans;
    }
    ll get_min()
    {
        for(int i = 0; i <= 60; ++i)
```

```
            if(d[i]) return d[i];
        return 0;
    }
    void rebuild()
    {
        for(int i = 60; i >= 0; --i)
            for(int j = i - 1; j >= 0; --j)
                if(d[i]&(1LL<<j))d[i] ^= d[j];
        for(int i = 0; i <= 60; ++i)
            if(d[i])b[cnt++] = d[i];
    }
    ll get_kth(ll k)
    {
        ll ans = 0;
        if( k >= (1LL<<cnt) ) return -1;
        for(int i = 60; i >= 0; --i)
            if(k&(1LL<<i))ans ^= b[i];
        return ans;
    }
};
l_B merg(const l_B &x,const l_B &y)
{
    l_B ans = x;
    for(int i = 60; i >= 0; --i)
        if(y.d[i])ans.insrt(y.d[i]);
    return ans;
}
//保持上三角性质的求线性基
void cal()
{
    for(int i = 0; i < n; ++i)
       for(int j = max_size; j >= 0; --j)
            if((a[i] >> j)&1)
            {
                if(b[j]) a[i] ^= b[j];
                else
                {
                    b[j] = a[i];
                    for(int k = j - 1; k >= 0; --k)
```

```
                    if(b[k]&&((b[j] >> k)&1)) b[j] ^= b[k];
                for(int k = j + 1; k <= max_size; ++k)
                    if((b[k] >> j)&1) b[k] ^= b[j];
            }
        }
}
```

4.12 自适应辛普森

```
const double exps=1e-5;
double F(double x)
{
    return 0;//写函数
}
double simpson(double a,double b,double A,double B)
{
    double c=a+(b-a)/2;
    return (A+4*F(c)+B)*(b-a)/6;
}
double asr(double a,double b,double eps,double A,double l,double r)// 必要时加一个
↪  deep
{
    double c=a+(b-a)/2;
    double m=F(c);
    double L=simpson(a,c,l,m),R=simpson(c,b,m,r);
    if(fabs(L+R-A)<=15*eps)return L+R+(L+R-A)/15.0;
    return asr(a,c,eps/2,L,l,m)+asr(c,b,eps/2,R,m,r);
}
double asr(double a,double b,double eps)
{
    double x=F(a),y=F(b);
    return asr(a,b,eps,simpson(a,b,x,y),x,y);
}
```

4.13 高斯消元 GauseElimination

```
const double eps=1e-8;
const int maxn=1e3+5;
typedef double Matrix[maxn][maxn];
//保证可逆
void gauss_emilination(Matrix A,int n)
```

```
{
    //求解的增广矩阵，最后的 A[i][n] 是第 i 个未知数的值
    for(int i=0;i<n;i++)
    {
        //选一行 r 并和 i 行交换
        int r=i;
        for(int j=i+1;j<n;j++)
            if( fabs(A[j][i]) > fabs(A[r][i]) )
                r=j;
        if(r!=i)
            for(int j=0;j<=n;j++)swap(A[r][j],A[i][j]);
        //与 i+1~n 行消元
        for(int k=i+1;k<n;k++)
        {
            double f=A[k][i]/A[i][i];
            for(int j=i;j<=n;j++)A[k][j]-=f*A[i][j];
        }
    }
    //回代求值
    for(int i=n-1;i>=0;i--)
    {
        for(int j=i+1;j<n;j++)
            A[i][n]-=A[j][n]*A[i][j];
        A[i][n]/=A[i][i];
    }
}
```

4.14 对角阵 GaussJordan

```
const double eps=1e-8;
const int maxn=1e3+5;
typedef double Matrix[maxn][maxn];
// 得到对角阵
void gauss_jordan(Matrix A,int n)
{
    // 求解的增广矩阵，最后的 A[i][n] 是第 i 个未知数的值
    for(int i=0;i<n;i++)
    {
        // 选一行 r 并和 i 行交换
        int r=i;
```

```
        for(int j=i+1;j<n;j++)
            if(fabs(A[j][i])>fabs(A[r][i]))r=j;

        if(fabs(A[r][i])<eps) continue;
        if(r!=i)
            for(int j=0;j<=n;j++) swap(A[r][j],A[i][j]);
        // 与 i+1~n 行消元
        for(int k=0;k<n;k++)
        {
            if(k!=i)
            for(int j=n;j>=i;j--) A[k][j]-=A[k][i]/A[i][i]*A[i][j];
        }
    }
}
```

4.15 米勒罗宾素数测试 MillerRabin

```
const int T=8;
ll random(ll n) { return (ll)((double)rand()/RAND_MAX*n+0.5); }
ll fmod(ll a,ll b,ll c)//a^b%c;
{
    ll ans=1;
    while(b)
    {
        if(b&1) ans=ans*a%c;
        a=a*a%c;
        b>>=1;
    }
    return ans;
}
bool Witness(ll a,ll b)//a^b;
{
    ll m=b-1; int j=0;
    while(!(m&1)) //分解 b-1=m*2^j;
        j++, m>>=1;
    ll x=fmod(a,m,b);
    if( x==1 || x==b-1 ) return true;
    while(j--)      //二次探测
    {
        x=x*x%b;
```

```
        if(x==b-1) return true;
    }
    return false;
}
bool miller_rabin(ll x)
{
    if(x<2)return false;
    if(x==2)return true;
    if(!(x&1))return false;
    for(int i=1;i<=T;i++)
    {
        ll a=random(x-2)+1;
        if(!Witness(a,x))
            return false;
    }
    return true;
}
int main()
{
    ll x;
    while(cin>>x)
    {
        if(miller_rabin(x)) cout<<" 素数"<<endl;
        else cout<<" 合数"<<endl;
    }
}
```

4.16 模方程 (可非素数)

```
map<ll, ll> dic;
ll fmod(ll x, ll y, ll p)
{
    ll ans = 1;
    while(y)
    {
        if(y&1) ans = ans*x%p;;
        y>>=1;
        x = x*x%p;
    }
    return ans;
```

```
}
ll exbsgs(ll a, ll b, ll p)
{
    if(b == 1LL) return 0;
    ll t, d = 1, k = 0;
    while((t = gcd(a,p)) != 1)
    {
        if(b % t) return -1;
        ++k; b /= t; p /= t; d = d*(a/t)%p;
        if(b == d) return k;
    }
    dic.clear();
    ll m = ceil(sqrt(p)), a_m = fmod(a,m,p), mul = b;
    for(ll j = 1; j <= m; ++j)
    {
        mul = mul *a%p;
        dic[mul] = j;
    }
    for(ll i = 1; i <= m; ++i)
    {
        d = d *a_m%p;
        if(dic[d]) return i*m-dic[d]+k;
    }
    return -1;
}
```

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 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]);
        }
    }
};
```

5.4 回文树

```
// 空间：O(n*size)，时间：O(nlog(size))
// 数组版
const int maxn = 100005;
const int SIZE = 26;

struct Palindromic_Tree
{
    int next[maxn][SIZE]; // next 指针，next 指针和字典树类似，指向的串为当前串两端加上
    ↪ 同一个字符构成
    int fail[maxn];       // fail 指针，失配后跳转到 fail 指针指向的节点
    int cnt[maxn];        // i 表示的本质不同的串个数（count() 一遍后正确）
    int num[maxn];        // i 表示的最长回文串最右端为回文串结尾的回文串个数
    int len[maxn];        // len[i] 表示节点 i 表示的回文串的长度
    int S[maxn];          // 存放添加的字符
    int last;             // 指向上一个字符所在的节点，方便下一次 add
    int n;                // 字符数组指针
    int p;                // 节点指针

    int newnode(int l)
    { //新建节点
        for (int i = 0; i < SIZE; ++i)
            next[p][i] = 0;
        cnt[p] = 0, num[p] = 0;
        len[p] = l;
        return p++;
    }

    void init()//初始化
    {
        p = 0, n = 0, last = 0;
```

```
        newnode(0),newnode(-1);
        S[n] = -1;fail[0] = 1; //开头放个字符集中没有的字符减少特判
    }

    int get_fail(int x)
    { //和 KMP 一样，失配后找一个尽量最长的
        while (S[n - len[x] - 1] != S[n])
            x = fail[x];
        return x;
    }

    void add(int c)
    {
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last); //通过上一个回文串找这个回文串的匹配位置
        if (!next[cur][c])
        {                                                    //如果这个回文串没有出现过，说明
        ↪ 出现了一个新的本质不同的回文串
            int now = newnode(len[cur] + 2);                 //新建节点
            fail[now] = next[get_fail(fail[cur])][c]; //和 AC 自动机一样建立 fail 指
            ↪ 针，以便失配后跳转
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = next[cur][c];
        cnt[last]++;
    }

    void count()
    {
        for (int i = p - 1; i >= 0; --i)
            cnt[fail[i]] += cnt[i];
        //父亲累加儿子的 cnt，因为如果 fail[v]=u，则 u 一定是 v 的子回文串!
    }
};
// 邻接表，空间稍优，时间略慢
struct PAM
{
```

```
vector<pair<int, int>> next[maxn];
int fail[maxn], num[maxn], len[maxn];
ll cnt[maxn];
int s[maxn], n, p, last, sum;

void init()
{
    p = 0,sum = 0, last = 0, n = 0;
    newnode(0), newnode(-1);
    s[n] = -1;
    fail[0] = 1;
}
int newnode(int w)
{
    next[p].clear();
    cnt[p] = num[p] = 0;
    len[p] = w;
    return p++;
}
int get_fail(int x)
{
    while (s[n - len[x] - 1] != s[n])
        x = fail[x];
    return x;
}
int add(int c)
{
    c -= 'a';
    s[++n] = c;
    int cur = get_fail(last);
    int flag = 0;
    for (int i = 0; i < next[cur].size(); i++)
        if (next[cur][i].first == c)
        {
            last = next[cur][i].second;
            cnt[last]++;
            return num[last];
        }
    int now = newnode(len[cur] + 2);
```

```
        int fi = get_fail(fail[cur]);
        flag = 0;
        for (int i = 0; i < next[fi].size(); i++)
            if (next[fi][i].first == c)
            {
                flag = next[fi][i].second;
                break;
            }
        fail[now] = flag;
        next[cur].push_back(make_pair(c, now));
        num[now] = num[flag] + 1;
        last = now;
        cnt[now]++;
        return num[now];
    }
    void count()
    {
        for (int i = p - 1; i > 1; i--)
            cnt[fail[i]] += cnt[i];
    }
};
```

6 动态规划 Dynamic Programme

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)
{
    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);
}
```

6.2 旅行商 TSP

```
void TSP()
{
    memset(dp,0x3f,sizeof(dp));
    for(int i=0;i<m;++i)
        dp[1<<i][i]=0;

    for(int i=1;i<(1<<m);++i)// 枚举状态：经过哪些点
        for(int j=0;j<m;++j)// 经过这些点时到达的最后一个点
            if(dp[i][j]!=INF)// 这个状态是合法的
            /** 我觉得其实 i 里存在 j 判断合法不太对？存在 j 也有可能到不了 j 状态是
            ↪ INF？
            但 i&(1<<j) 也可以过，是因为取小操作避免了从 INF 转移过去？
            dp[i][j]!=INF 这个判断比 i&(1<<j) 快了 15ms，可能的确存在符合后者但 INF 的
↪ 情况。
            好吧，再交次没区别了，甚至更慢了点，可能是看机器心情吧 orz**/
                for(int k=0;k<m;++k)// 枚举新走到的点，那么它一定是新的终点
                    if(((i&(1<<k))==0) && cost[j][k]!=INF)
                    /** 注意如果 i 中已有 k 点，那么如果发生转移，有可能发生：
                    经过 i 这个集合，终点在 x，但是又转移到了 y，也就是这个状态同时有两
↪ 个终点了，而这个转移又修改了状态，就 GG 了!!!
                    **/
                        dp[i|(1<<k)][k]=min(dp[i|(1<<k)][k],dp[i][j]+cost[j][k]);
}
```

6.3 数位 dp

```
#include<bits/stdc++.h>//sum%(x*n)%x=sum%x;
using namespace std;
typedef long long ll;
int num[20];
ll dp[20][state];
ll dfs(int pos,/*state*/,int lead/*¾0*/,int limit/*  */)
{
    if(pos==-1)return 1;//¿´
    if(!limit&&dp[pos][state]!=-1)return dp[pos][state];
    int up=limit?a[pos]:9;
    ll ans=0;
    for(int i=0;i<=up;i++)
    {
        if()...
        }
        if(!limit&&!lead)dp[pos][state]=ans;
        //±  F    62 ô   ½ ¼ » 6    10±
        //   º        ´¾  ½
        return ans;
    }
    ll solve(ll x)
    {
        int pos=0;
        while(x)
        {
            a[pos++]=x%10;
            x/=10;
        }
        return dfs(pos-1,/* */,1,1);
        //  ª   ¶¼     ¢   ¾µ¼  ,
        // ±   »¹  ,    IO
    }
    int main()
    {
    }
            else if()...
```

7 计算几何 Computation Geometry

7.1 圆 Circle

```
struct Circle
{
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a)
    {
        return Point(c.x + cos(a) * r, c.y + sin(a) * r);
```

```
    }
};

// 两圆交点个数及坐标
int getCircleIntersection(Circle C1, Circle C2, vector<Point> &sol)
{
    double d = Length(C1.c - C2.c);
    if (dcmp(d) == 0) // 首先圆心重合
    {
        if (dcmp(C1.r - C2.r) == 0)
            return -1; // 其次半径相同，然后就可以推出两圆重合
        return 0;
    }
    if (dcmp(C1.r + C2.r - d) < 0)
        return 0; // 相离没交点
    if (dcmp(fabs(C1.r - C2.r) - d) > 0)
        return 0; // 内含，没有交点

    double a = angle(C2.c - C1.c);                                          //向量
    ↪ C1C2 的极角
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d)); //C1C2 到
    ↪ C1P1 的角
    Point p1 = C1.point(a - da), p2 = C1.point(a + da);

    sol.push_back(p1);
    if (p1 == p2) return 1; // 相切
    sol.push_back(p2);
    return 2; // 相交
}
```

8 其它 Other

8.1 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;
    }
}
```

8.2 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;
```

8.3 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);
```

8.4 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)   >=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
*/
```

9 输入输出 IO