

Algorithm template

lxz

November 27, 2021

Contents

1 基础算法	6
1.1 高精度	6
1.1.1 高精/低精	6
1.2 差分	6
1.2.1 差分矩阵	6
2 经典动态规划	8
2.1 数位 dp	8
2.2 最长上升子序列 LIS	10
2.3 背包	11
2.3.1 01 背包	11
2.3.2 完全背包	12
2.3.3 多重背包 (普通和二进制拆位)	12
2.3.4 分组背包	14
2.4 最长公共子序列	14
2.5 单调队列优化 dp	15
3 搜索	17
3.1 模拟退火	17
4 图论	19
4.1 树的重心 (一棵树的和每个子树的)	19
4.2 差分约束	22
4.3 欧拉路径和欧拉回路	25
4.3.1 欧拉回路和路径判定	25
4.3.2 欧拉回路输出路径	25
4.4 网络流	27
4.4.1 最大流	27
4.4.2 无源汇上下界可行流	30
4.4.3 费用流	32
4.5 最小生成树	35
4.5.1 Kruskal	35
4.6 2Sat	36
4.7 lca	38
4.8 拓扑排序	40
4.9 多源最短路	41
4.9.1 Floyd	41
4.9.2 Floyd 求最小环	42
4.10 单源最短路	43
4.10.1 SPFA	43
4.10.2 堆优化 dij	44
4.10.3 SPFA 判负环	45

4.11 有向图强连通分量	47
4.11.1 scc tarjan	47
4.12 无向图双连通分量	49
4.12.1 边双联通分量	49
4.12.2 点双联通分量	51
4.13 二分图	55
4.13.1 染色法判定二分图	55
4.13.2 最大匹配 = 最小点覆盖：选出最少的点，使每条边至少有一个点被选出来（在选出的点里面）= 点数-最大独立集：选出最多的点使得选出的点之间没有边。 = 点数-最小路径点覆盖（最小路径覆盖）：针对一个有向无环图（DAG），用最少条互不相交路径，覆盖所有点。（其中互不相交是指点不重复）（匈牙利算法 $O(n * m)$ ）	56
5 数据结构	58
5.1 链表	58
5.2 字典树	59
5.3 可撤销并查集	60
5.4 线段树	61
5.4.1 单点修改	61
5.4.2 区间修改	63
5.4.3 返回 node 的线段树（区间最大字段和）	65
5.4.4 括号序列线段树（区间括号序列是否合法）	67
5.5 可持久化数据结构（历史版本）	69
5.5.1 主席树第 K 小数	69
5.5.2 可持久化 trie 求 L-R 中某个后缀与 X 最大异或和	71
5.6 树套树	73
5.6.1 线段树套线段树求区间 max,sum,min	73
5.7 树状数组	77
5.8 RMQ (st 表)	77
5.9 单调队列	78
5.10 点分治	79
5.11 树链剖分	81
5.12 Splay	86
5.13 莫队	88
5.13.1 基础莫队	88
5.13.2 带修莫队	90
6 数论	93
6.1 阶乘分解	93
6.2 扩展中国剩余定理	94
6.3 min25 求 1e10 以内素数和	95
6.4 矩阵快速幂	97

6.5	筛质数	99
6.5.1	线性筛 (欧拉函数)	99
6.6	组合数	100
6.6.1	杨辉三角	100
6.6.2	lucas 定理 ($a, b \leq 10^{18}$) P 是质数	101
6.6.3	扩展 lucas ($a, b \leq 10^{18}$) P 不一定质数	102
6.7	卡特兰数的应用	105
6.7.1	卡特兰数 (P 可能是合数, 阶乘分解法)	106
6.8	约数	108
6.8.1	约数个数	108
6.8.2	约数的和	109
6.9	Exgcd 及同余性质	111
6.10	高斯消元 n^3	112
6.10.1	高斯消元解齐次线性方程组	112
6.10.2	高斯消元解异或线性方程组	114
6.10.3	高斯消元求行列式值 (取模,double)	117
6.11	容斥原理	119
6.12	FFT 与 NTT	120
6.13	BSGS (普通和扩展)	126
6.14	生成函数	130
6.15	线性基	131
6.16	欧拉降幂	132
6.17	康托展开求排列的字典序排名	133
7	计算几何	137
7.1	凸包	137
7.2	半平面交	139
7.3	最小圆覆盖	142
7.4	三维凸包	145
7.5	旋转卡壳	148
7.5.1	平面最远点对	148
7.5.2	最小矩形覆盖	150
7.6	三角剖分 (圆和多边形面积交)	154
7.7	扫描线	158
7.7.1	三角形面积并	158
7.7.2	矩形面积并	162
7.8	自适应辛普森积分	164
8	字符串	166
8.1	字符串哈希	166
8.2	KMP	167
8.3	EXKMP	168
8.4	manacher	170
8.5	AC 自动机	171

8.6 后缀自动机	173
8.7 后缀数组	175
9 博弈论	178
9.1 SG 函数	178
10 混合板子	180
10.1 杜教 BM (求线性递推)	180
10.2 complex 复数板子	182
10.3 快读	186
10.4 随机与 cin 解绑	186

1 基础算法

1.1 高精度

1.1.1 高精/低精

```
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
vector<int>A;
vector<int> div(vector<int>a,int b,int &r)
{
    vector<int>c;
    r=0;
    for(int i=a.size()-1;i>=0;i--)//从高位开始算
    {
        r=r*10+a[i];
        c.pb(r/b);
        r%=b;
    }
    reverse(c.begin(),c.end());
    while(c.size()>1&&c.back()==0)c.pop_back();
    return c;
}
int main()
{
    string a;
    int b;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.pb(a[i]-'0');
    int r;
    auto c=div(A,b,r);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    cout<<endl;
    cout<<r;
    return 0;
}
```

1.2 差分

1.2.1 差分矩阵

```
//2020/8/3
#include<bits/stdc++.h>
using namespace std;
```

```
const int N=1e3+10;
int n,m,q,a[N][N],b[N][N]; //b 数组为变化量
int main()
{
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            cin>>a[i][j];
        }
    }
    while(q--)
    {
        int x1,y1,x2,y2,c;
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2,&c);
        b[x1][y1]+=c; //x1, y1 右下角的矩阵全部加上 c
        b[x2+1][y1]-=c;
        b[x1][y2+1]-=c;
        b[x2+1][y2+1]+=c;
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            //如下操作 i, j 前面的 b 已经变成前缀和了
            b[i][j]+=b[i-1][j]+b[i][j-1]-b[i-1][j-1]; //差分矩阵
            // 复原为前缀和 (对 b 做前缀和 (1, 1) - (i, j) 的和
            printf("%d ",b[i][j]+a[i][j]); //原来的加上差值即可
        }
        puts("");
    }
}
```

2 经典动态规划

2.1 数位 dp

```

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=35;
int f[N][N],pos=0,a[N],x,y,k,b;//f[i][j] 表示 当前枚举到第 i 位,
→ 前面已经取了 j 个 1 的方案数
int dfs(int pos,int pre,int last,int limit)
//pos 当前枚举的位数, pre 前面取了多少个 1, last 上一个数是几,
→ limit 当前位取数有没有限制
{
    if(!pos) return pre==k;//枚举完成了, 判断边界这个数是否符合要
    → 求
    if(last>1) return 0;//如果前一位取了比 1 大的数, 直接不符合要
    → 求。
    if(!limit&&f[pos][pre]!=-1) return f[pos][pre];//记忆化
    int up=limit?a[pos]:b;
    int res=0;
    if(up==0) res+=dfs(pos-1,pre,0,1);//当前位最高只能取 0, 直接
    → 进入下一位看
    else if(up)//当前位最高可以取 >=1
    {
        res+=dfs(pos-1,pre,0,0);//则当前位取 0 一定合法
        if(up>1)
        {
            if(pre<k) res+=dfs(pos-1,pre+1,1,0);//当前位取 1 的
            → 话
        }
        else if(up==1)//当前位最高只能取 1, 那么只能取 1 了。(取
        → 0 在前面已经写了)
        {
            if(pre<k) res+=dfs(pos-1,pre+1,1,1);
        }
    }
    if(!limit)f[pos][pre]=res;
    return res;
}
int dp(int n)
{
    if(!n) return 0;
    int pos=0;

```

```
while(n)
{
    a[++pos]=n%b;
    n/=b;
}
return dfs(pos,0,0,1);
}
int main()
{
    memset(f,-1,sizeof f);
    cin>>x>>y>>k>>b;
    cout<<dp(y)-dp(x-1);
    return 0;
}

ll dfs(int pos,ll pre,int lead,int limit)
{
    if(!limit&&!lead&&f[pos].count(pre))return f[pos][pre];
    if(!pos)
    {
        return pre<=k&&!lead;
    }
    int up = limit? a[pos] : 9;
    ll res = 0;
    for(int k = 0; k <= up; k++){
        if(lead)
        {
            if(!k)res += dfs(pos-1,1,lead,limit && k == up );
            else res += dfs(pos-1,k,0,limit && k == up );
        }
        else res += dfs(pos-1,pre*k,0,limit && k == up );
    }
    if(!limit&&!lead)return f[pos][pre]=res;
    return res;
}
ll dp(ll n)
{
    int pos=0;
    while(n)
    {
        a[++pos]=n%10;
        n/=10;
    }
}
```

```
    return dfs(pos,1,1,1);
}
```

2.2 最长上升子序列 LIS

```
int a[N],q[N];
void solve()
{
    int n;
    n=read();
    for(int i=0;i<n;i++)a[i]=read(),q[i]=0;
    q[0]=-2e9;
    int len1=0;
    for(int i=0;i<n;i++)//非递减
    {
        int l=0,r=len1;
        while(l<r)
        {
            int mid=l+r+1>>1;
            if(q[mid]<=a[i])l=mid;
            else r=mid-1;
        }
        len1=max(len1,r+1);
        q[r+1]=a[i];
    }
    int len2=0;
    for(int i=0;i<n;i++)q[i]=0;
    q[0]=2e9;
    for(int i=0;i<n;i++)//非递增
    {
        int l=0,r=len2;
        while(l<r)
        {
            int mid=l+r+1>>1;
            if(q[mid]>=a[i])l=mid;
            else r=mid-1;
        }
        len2=max(len2,r+1);
        q[r+1]=a[i];
    }
    if(len1>=n-1||len2>=n-1)puts("YES");
    else puts("NO");
```

```

}
int main()
{
    int t;
    cin>>t;
    //t=1;
    for(int i=1;i<=t;i++)
    {
        //printf("Case #%d: ", i);
        solve();
    }
    return 0;
}

```

2.3 背包

2.3.1 01 背包

//有 n 件物品和一个容量是 m 的背包，每件物品只能用一次，第 i 件物品的体积是 vi ，价值是 wi 。
 //求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。
 //输出最大价值。

```

#include<bits/stdc++.h>
using namespace std;
const int N=1005;
int f[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        int v,w;
        cin>>v>>w;
        for(int j=m;j>=v;j--)
        {
            f[j]=max(f[j],f[j-v]+w);
        }
    }
    cout<<f[m];
    return 0;
}

```

2.3.2 完全背包

```
//完全背包，每件物品可以无限用
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e3+10;
int f[maxn],v[maxn],w[maxn];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
    for(int i=1;i<=n;i++)
    {
        for(int j=v[i];j<=m;j++)//从小到大，保证是 f[i]..(用
        //的是这层的来更新)
        {
            if(j>=v[i])f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    cout<<f[m];
    return 0;
}
```

2.3.3 多重背包（普通和二进制拆位）

```
//多重背包，第 i 件物品有 si 件。
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e2+10;
int f[maxn];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        int v,w,s;
        cin>>v>>w>>s;
        for(int j=m;j>=0;j--)
        {
            //f[i][j]=f[i-1][j];
            for(int k=0;k<=s&&k*v<=j;k++)
            {
```

```

        f[j]=max(f[j],f[j-k*v]+k*w);
    }

}

cout<<f[m];
return 0;
}

/*
多重背包问题 II。每个物品之只有 s 件。时间复杂度要求更高。 $2 \times 10^{17}$ 。
将背包用二进制拆开，变成 01 背包问题。如 10，拆成 1 2 4 3，用这
→ 4 个数可以表示 0-10 的数
*/
#include<bits/stdc++.h>
using namespace std;
const int N=2e3+10;
int f[N];
int main()
{
    int n,m;
    cin>>n>>m;
    vector<pair<int,int>>goods;//fi 体积 se 价值
    for(int i=1;i<=n;i++)
    {
        int v,w,s;
        cin>>v>>w>>s;
        for(int k=1;k<=s;k*=2)//把 s 拆分成几个物品，变成 01 背
        → 包
        {
            s-=k;
            goods.push_back({k*v,k*w});
        }
        if(s>0)goods.push_back({s*v,s*w});
    }
    for(auto i:goods)
    {
        for(int
        → j=m;j>=i.first;j--)f[j]=max(f[j],f[j-i.first]+i.second);
    }
    cout<<f[m];
}

```

2.3.4 分组背包

```
/*
有  $N$  组物品和一个容量是  $V$  的背包。
每组物品有若干个，同一组内的物品最多只能选一个。
每件物品的体积是  $v_{ij}$ ，价值是  $w_{ij}$ ，其中  $i$  是组号， $j$  是组内编号。
求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最
大。
输出最大价值。
*/
#include<bits/stdc++.h>
using namespace std;
const int N=105;
int f[N],v[N],w[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=0;i<n;i++)
    {
        int s;
        cin>>s;
        for(int j=0;j<s;j++)cin>>v[j]>>w[j];
        for(int j=m;j>=0;j--)
        {
            for(int k=0;k<s;k++)
            {
                if(j>=v[k])f[j]=max(f[j],f[j-v[k]]+w[k]);
            }
        }
    }
    cout<<f[m];
    return 0;
}
```

2.4 最长公共子序列

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int n,m,f[N][N];
char a[N],b[N];
int main()
{
```

```

cin>>n>>m>>a+1>>b+1;
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=m;j++)
    {
        f[i][j]=max(f[i-1][j],f[i][j-1]);
        if(a[i]==b[j])f[i][j]=max(f[i][j],f[i-1][j-1]+1);
    }
}
cout<<f[n][m];
return 0;
}

```

2.5 单调队列优化 dp

```

//输入一个长度为 n 的整数序列，从中找出一段长度不超过 m 的连续子
→ 序列，使得子序列中所有数的和最大。
//注意：子序列的长度至少是 1。
#include<bits/stdc++.h>
using namespace std;
const int N=3e5+10;
typedef long long ll;
ll s[N],q[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        cin>>s[i];
        s[i]+=s[i-1];
    }
    int tt=0,hh=0;//t 从 -1 开始，但因为 q 已经加了个 0 进去。所
    → 以 tt++ 变成 0
//q[0]=0;
ll res=INT_MIN;
for(int i=1;i<=n;i++)
{
    if(i-q[hh]>m)hh++;//j 在 [i-m, i-1]，因为是前缀和,
    → si-s(j-1); 即队列内队头元素下标 <i-m 就要弹出
    res=max(res,s[i]-s[q[hh]]);
    while(hh<=tt&&s[q[tt]]>=s[i])tt--;
    q[++tt]=i;
}

```

```
    }
    cout<<res;
    return 0;
}
```

3 搜索

3.1 模拟退火

```
/* 模拟退火 求最大值最小值 */
#include<bits/stdc++.h>
using namespace std;
const int N=105;
#define x first
#define y second
typedef pair<double,double> pdd;
int n;
pdd q[N];
double ans=1e8;

double rand(double l,double r)//得到 l-r 之间的随机数
{
    return (double)rand()/RAND_MAX*(r-l)+l;
}
double cal(pdd a,pdd b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
double calc(pdd x)
{
    double res=0;
    for(int i=0;i<n;i++)
    {
        res+=cal(q[i],x);
    }
    ans=min(ans,res);
    return res;
}
void simulate_anneal()
{
    pdd cur(rand(0,10000),rand(0,10000));
    for(double t=1e4;t>1e-4;t*=0.99)//初温 1e4 终止温 1e-4 衰减
        → 系数 0.99(可以看情况改，越大循环次数越多),t 步长
    {
        pdd np(rand(cur.x-t,cur.x+t),rand(cur.y-t,cur.y+t));
        double dt=calc(np)-calc(cur);
        if(exp(-dt/t)>rand(0,1))cur=np;
        /* 这样写恰好就是我们想走的
```

```
    1.  $dt < 0$  时, 说明新点距离变小, 结果更好,  
    ↳  $\exp(>0) > 1 > \text{rand}(0, 1)$  一定跳到新点  
    2.  $dt > 0$  时, 说明新点距离变大, 结果变差。 $0 < \exp(<0) < 1$  且  $dt$   
    ↳ 越大 (新点结果很差) 整个  $\exp$  越小, 跳的概率越低。  
    */
}  
}  
int main()  
{  
    cin >> n;  
    for(int i=0; i<n; i++) cin >> q[i].x >> q[i].y;  
    for(int i=0; i<n; i++) simulate_anneal(); // 迭代 100 次模拟退火  
    printf("%.0lf", ans);  
    return 0;  
}
```

4 图论

4.1 树的重心（一棵树的和每个子树的）

```

/*
1. 树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两
→ 个重心，他们的距离和一样。
2. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
3. 一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。
4. 一棵树最多有两个重心，且相邻。
*/
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
vector<int>g[N];
#define pb push_back
int n,ans=N,st[N];
int dfs(int u)//返回以 u 为根结点的树的点个数，包括 u
{
    int sum=1;//以 u 为根结点的树的点个数，包括 u
    int res=0;//以当前点为重心（删掉该点），（剩余）最大的联通块点
    → 的个数
    st[u]=1;
    for(auto i:g[u])
    {
        if(!st[i])
        {
            int s=dfs(i);
            sum+=s;
            res=max(res,s);
        }
    }
    res=max(res,n-sum);
    ans=min(ans,res);//所有最大中选最小的出来
    return sum;
}
int main()
{
    cin>>n;
    for(int i=0;i<n-1;i++)
    {
        int u,v;
        cin>>u>>v;
    }
}

```

```

        g[u].pb(v);
        g[v].pb(u);
    }
    dfs(1);
    cout<<ans;
}
/*
 * https://vjudge.net/contest/468038#problem/M
 * 询问每个子树的重心，核心思路是重心一直往上走，On。
 */
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int n,sz[N],f[N],son[N],d[N];
vector<int>g[N];
void find(int rt,int x,int y)
{
    while(d[x]>d[rt]&&sz[x]<sz[rt]-sz[x])x=f[x];
    while(d[y]>d[rt]&&sz[y]<sz[rt]-sz[y])y=f[y];
    if(d[y]>d[x])son[rt]=y;
    else son[rt]=x;
}
void dfs(int u,int fa)
{
    f[u]=fa;
    sz[u]=1;
    son[u]=u;
    for(auto i:g[u])
    {
        if(i==fa)continue;
        d[i]=d[u]+1;
        dfs(i,u);
        sz[u]+=sz[i];
        f[i]=u;
        find(u,son[u],son[i]);
    }
}
int main()
{
    cin>>n;
    for(int i=0;i<n-1;i++)
    {
        int u,v;

```

```
u=read(),v=read();
g[u].pb(v);
g[v].pb(u);
}
dfs(1,-1);
for(int i=1;i<=n;i++)
{
    bool flag=0;
    if(son[i]!=1&&sz[son[i]]==sz[i]-sz[son[i]])flag=1;
    if(flag)printf("%d
                   %d\n",min(son[i],f[son[i]]),max(son[i],f[son[i]]));
    else printf("%d\n",son[i]);
}
return 0;
}
```

4.2 差分约束

求最少，则要最长路（若有正环则不行），把每一个式子都化成 $b \geq a + C$ 的形式，然后从 a 连一条边权为 C 的边到 b ，因为最长路都有 $\text{dis}[b] \geq \text{dis}[a] + c$ 若出现 $b > a$ 则可以利用整数的性质。变成 $b \geq a + 1$ ，每个人至少一个糖果。则从0连一条长度为1的边到每一个点。这题用vector会超时，求环的时候可以用栈玄学优化

如求最大值，跑最短路的时候。 $a \leq b + C$ ，可以理解为 b 连向 a ，就像最终是1连到n，每条边都取到等号就是最大值了，

只要有环就一定不行。而不是从1开始找不到就不行，因为从1走不到环不代表没有环

建立超级原点以后，点数变成 $n+1$ 要判断 $\text{cnt}[\text{to}] >= n+1$ 才是有环

差分约束

(1) 求不等式组的可行解

源点需要满足的条件：从源点出发，一定可以走到所有的边。

步骤：

- [1] 先将每个不等式 $x_i \leq x_j + c_k$, 转化成一条从 x_j 走到 x_i ，长度为 c_k 的一条边
- [2] 找一个超级源点，使得该源点一定可以遍历到所有边
- [3] 从源点求一遍单源最短路

结果1：如果存在负环，则原不等式组一定无解

结果2：如果没有负环，则 $\text{dist}[i]$ 就是原不等式组的一个可行解

(2) 如何求最大值或者最小值，这里的最值指的是每个变量的最值

结论：如果求的是最小值，则应该求最长路；如果求的是最大值，则应该求最短路；

问题：如何转化 $x_i \leq c$ ，其中 c 是一个常数，这类的不等式

方法：建立一个超级源点，0，然后建立 $0 \rightarrow i$ ，长度是 c 的边即可。

以求 x_i 的最大值为例：求所有从 x_i 出发，构成的不等式链 $x_i \leq x_j + c_1 \leq x_k + c_2 + c_1 \leq \dots \leq c_1 + c_2 + \dots$

所计算出的上界，最终 x_i 的最大值等于所有上界的最小值。

$$\left\{
 \begin{array}{lll}
 \textcircled{1} A=B \Leftrightarrow & A \geq B & B \geq A \\
 \textcircled{2} A < B \Leftrightarrow & B \geq A+1 & \\
 \textcircled{3} A \geq B \Leftrightarrow & A \geq B \\
 \textcircled{4} A \geq B \Leftrightarrow & A \geq B+1 & \\
 \textcircled{5} A \leq B \Leftrightarrow & B \geq A & \\
 & \vdots & \vdots \quad \vdots \\
 X \geq 1 & X \geq X_0 + 1 & \\
 X_0 = 0 & & 0 \rightarrow 1
 \end{array}
 \right.$$

X_0 取 0

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10,M=3e5+10;
typedef long long ll;
bool st[N];
int n,k,stk[N],cnt[N],dis[N];
int h[N], e[M], w[M], ne[M], idx;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
bool spfa()
{
    int tt=0,hh=0;
    stk[tt]=0;
    st[0]=1;
    cnt[0]++;
    while(hh!=tt)
    {
        int t=stk[tt--];
        st[t]=0;
        for(int i=h[t];~i;i=ne[i])
        {
            int to=e[i],cost=w[i];
            if(dis[to]<dis[t]+cost)
            {
                dis[to]=dis[t]+cost;
                if(st[to]==0)
                    stk[++tt]=to;
                st[to]=1;
                cnt[to]++;
            }
        }
    }
}
    
```

```
dis[to]=dis[t]+cost;
if(!st[to])
{
    st[to]=1;
    cnt[to]++;
    if(cnt[to]>n) return 0;//有正环
    stk[++tt]=to;
}
}
}
}
return 1;
}
int main()
{
    memset(h,-1,sizeof h);
    cin>>n>>k;
    for(int i=1;i<=k;i++)
    {
        int x,a,b;
        scanf("%d %d %d",&x,&a,&b);
        if (x == 1) add(b, a, 0), add(a, b, 0);
        else if (x == 2) add(a, b, 1);
        else if (x == 3) add(b, a, 0);
        else if (x == 4) add(b, a, 1);
        else add(a, b, 0);
    }
    for(int i=1;i<=n;i++)add(0, i, 1);//xi>=1 <=> xi>=x0+1

    ll res=0;
    if(spfa())
    {
        for(int i=1;i<=n;i++)res+=dis[i];
        printf("%lld",res);
    }
    else puts("-1");
    return 0;
}
```

4.3 欧拉路径和欧拉回路

4.3.1 欧拉回路和路径判定

欧拉路径：该路径经过图的每一条边且仅经过一次
 欧拉回路：满足欧拉回路且终点就是起点

一、无向图

- 1 存在欧拉路径的充要条件：度数为奇数的点只能有0或2个
- 2 存在欧拉回路的充要条件：度数为奇数的点只能有0个

二、有向图

- 1 存在欧拉路径的充要条件：要么所有点的出度均==入度；
 要么除了两个点之外，其余所有点的出度==入度 剩余的两个点：一个满足出度-入度==1(起点) 一个满足入度-出度==1(终点)
- 2 存在欧拉回路的充要条件：所有点的出度均等于入度

4.3.2 欧拉回路输出路径

```
//图中可能有重边也可能有自环。
//ver 是 1 是无向图, 2 是有向图
//如果 t=1 则表示 vi 到 ui 有一条无向边。
//如果 t=2 则表示 vi 到 ui 有一条有向边。

/*
如果 t=1
输出 m 个整数 p1,p2,⋯,pm。令 e=|pi|, 那么 e 表示经过的第 i 条边
→ 的编号。
如果 pi 为正数表示从 ve 走到 ue, 否则表示从 ue 走到 ve。
如果 t=2, 输出 m 个整数 p1,p2,⋯,pm。其中 pi 表示经过的第 i 条边
→ 的编号。
1 n 10^5,
0 m 2×10^5
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100100, M = 400100;

int h[N], e[M], ne[M], idx;
int ans[N*2], cnt;
bool used[M];
int din[N], dout[N];
int n, m, ver;

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
```

```

}

void dfs(int u){
    for(int &i = h[u]; ~i; ){
        if(used[i]) { //如果这条边用过了
            i = ne[i]; //删除这条边
            continue;
        }

        used[i] = true; //标记这条边已使用
        if(ver == 1) used[i^1] = true; //如果是无向图，那么这
        ↵ 条边的反向边也要标记使用过了

        int t;
        if(ver == 1){
            t = i/2 + 1;
            if(i&1) t = -t; // (0,1) (2,3) (4,5) 奇数编号是返回
            ↵ 的边
        }

        else t = i+1;

        int j = e[i];
        i = ne[i];
        dfs(j);
        ans[cnt++] = t;
    }
}

int main()
{
    scanf("%d%d%d", &ver, &n, &m);
    memset(h, -1, sizeof h);

    for(int i = 0; i<m; i++){
        int a,b;
        scanf("%d%d", &a, &b);
        add(a,b);
        if(ver == 1) add(b,a); //无向边
        din[b]++;
        dout[a]++;
    }

    if(ver == 1){
        for(int i = 1; i<=n; i++){
            if(din[i]+dout[i] &1){

```

```
//无向图含欧拉回路的充要条件是每个点的度都为偶数
puts("NO");
return 0;
}
}
}else{
    for(int i = 1; i<=n; i++){
        if(din[i] != dout[i]){
            //有向图含欧拉回路的充要条件是每个点的入度等于出
            //度
            puts("NO");
            return 0;
        }
    }
}

for(int i = 1; i<=n; i++){
    if(~h[i]) {
        dfs(i);
        break;
    }
}

if(cnt < m){
    puts("NO");
    return 0;
}

puts("YES");
for(int i = cnt-1; i>=0; --i){
    cout<<ans[i]<<" ";
}
return 0;
}
```

4.4 网络流

4.4.1 最大流

```
/*
最大流
dinic 求最大流 O(n^2mn^2m) 多路增广 可以处理 10000-100000 的网络
三个重要优化
1. 当前弧优化 cur, 直接从还没流满的边开始
```

2. $flow < limit$ 当前 u 点后面的总流量小于 $limit$
 3. 废点优化 该点 dfs 找不到可行的流，把那个点的层设为 -1

```

 $bfs()$  : 返回有无增广路, 有的话会顺便建立分层图 (避免后面搜的时候
 $\hookrightarrow$  遇到环)
注意表头初始化 -1
*/
#include<bits/stdc++.h>
using namespace std;
const int N=1e4+10,M=2e5+10,inf=0x3f3f3f3f;
int h[N],e[M],ne[M],f[M],idx;
int d[N],q[N],cur[N]; //d 表示点 i 的层数, cur 表示点 i 直接从
 $\hookrightarrow$  cur[i] 这条边开始
int n,m,S,T;
void add(int a,int b,int c)
{
    e[idx]=b,f[idx]=c,ne[idx]=h[a],h[a]=idx++;
    e[idx]=a,f[idx]=0,ne[idx]=h[b],h[b]=idx++;
}
bool bfs() //返回有无增广路, 有的话会顺便建立分层图 (避免后面搜的
 $\hookrightarrow$  时候遇到环)
{
    int hh=0,tt=0;
    memset(d,-1,sizeof d);
    q[0]=S;
    d[S]=0;
    cur[S]=h[S];
    while(hh<=tt)
    {
        int t=q[hh++];
        for(int i=h[t];~i;i=ne[i])
        {
            int j=e[i];
            if(d[j]==-1&&f[i])
            {
                d[j]=d[t]+1;
                cur[j]=h[j];
                if(j==T) return 1; //遇到层数比 T 大的点, 如果他有
                 $\hookrightarrow$  可行流到 T, 总会在某次 bfs 被搜出来
                q[++tt]=j;
            }
        }
    }
}

```

```

    return 0;
}
int find(int u,int limit)//limit 表示 u 前面的最大流量 (最小容
→ 量)
{
    if(u==T) return limit;
    int flow=0;//flow 表示从 u 开始的总流量
    //直接从当前弧开始, 且 u 后面的总流量必须小于前面的流量限制
    → limit, 流入流出守恒。
    for(int i=cur[u];~i&&flow<limit;i=ne[i])
    {
        cur[u]=i;//到 i 这条边, 说明前面的边都满了。
        int j=e[i];
        if(d[j]==d[u]+1&&f[i])//必须是下一层, 且有容量
        {
            //u 后面已经流了 flow, u 前面流入 limit。所以 u 后面
            → 还能流 limit-flow
            int t=find(j,min(limit-flow,f[i]));//返回 j 点开始
            → 最大流量
            if(!t)d[j]=-1;
            f[i]-=t,f[i^1]+=t,flow+=t;
        }
    }
    return flow;
}
int dinic()
{
    int r=0;
    while(bfs())r+=find(S,inf);//从 S 开始流量限制为 inf
    return r;
}
int main()
{
    memset(h,-1,sizeof h);
    scanf("%d%d%d%d",&n,&m,&S,&T);
    for(int i=1;i<=m;i++)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
    }
    printf("%d",dicnic());
    return 0;
}

```

4.4.2 无源汇上下界可行流

```

/*
无源汇上下界可行流
每条边的容量有上界和下界要求，求最大流。
让上界-下界为容量。流量-下界为新图流量。这样操作以后满足容量限制，
→ 但流量守恒不满足。
看入边减的多的话就从  $S$  连一条边过去。否则连到  $T$ 
最终从  $S$  连出去的流必须都是满的，因为原图中流量是守恒的。每条边的
→ 流量-下界以后。就不守恒了。

```

$f[i \sim 1]$ 则是新图中边 i 的流量。加上 $l[i]$ 就是实际的流量

```

*/
#include<bits/stdc++.h>
using namespace std;
const int N=205,M=50200,inf=0x3f3f3f3f;
int h[N],ne[M],e[M],idx,f[M],l[M];
int q[N],cur[N],d[N],A[N],S,T;
void add(int a,int b,int c,int d)
{
    e[idx]=b,f[idx]=d-c,l[idx]=c,ne[idx]=h[a],h[a]=idx++;
    e[idx]=a,f[idx]=0,ne[idx]=h[b],h[b]=idx++;
}
bool bfs()
{
    int hh=0,tt=0;
    memset(d,-1,sizeof d);
    q[0]=S,cur[S]=h[S],d[S]=0;
    while(hh<=tt)
    {
        int t=q[hh++];
        for(int i=h[t];~i;i=ne[i])
        {
            int j=e[i];
            if(d[j]==-1&&f[i])
            {
                d[j]=d[t]+1;
                cur[j]=h[j];
                if(j==T) return 1;
                q[++tt]=j;
            }
        }
    }
    return 0;
}

```

```
}

int find(int u,int limit)
{
    if(u==T) return limit;
    int flow=0;
    for(int i=cur[u];~i&&flow<limit;i=ne[i])
    {
        int j=e[i];
        cur[u]=i;
        if(d[j]==d[u]+1&&f[i])
        {
            int t=find(j,min(f[i],limit-flow));
            if(!t)d[j]=-1;
            flow+=t;
            f[i]-=t,f[i^1]+=t;
        }
    }
    return flow;
}
int dinic()
{
    int res=0;
    while(bfs())res+=find(S,inf);
    return res;
}
int main()
{
    memset(h,-1,sizeof h);
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int a,b,c,d;
        cin>>a>>b>>c>>d;
        add(a,b,c,d);
        A[a]-=c,A[b]+=c;
    }
    S=0,T=n+1;
    int tot=0;
    for(int i=1;i<=n;i++)
    {
        if(A[i]>0)add(S,i,0,A[i]),tot+=A[i];
        else add(i,T,0,-A[i]);
    }
}
```

```

if(dinic() !=tot)puts("NO");
else
{
    puts("YES");
    for(int i=0;i<m*2;i+=2)printf("%"d\n",f[i^1]+l[i]);
}
return 0;
}

```

4.4.3 费用流

```

/*
费用流， 最小费用最大流
*/
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
typedef long long ll;
const int N = 505, M = 100010, INF = 1e9;

int n, m, S, T;
ll h[N], e[M], f[M], w[M], ne[M], idx, num[N];
ll q[N], d[N], pre[N], incf[N];
bool st[N];

void add(int a, int b, int c, int d)
{
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] =
        ↳ idx ++ ;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b]
        ↳ = idx ++ ;
}

bool spfa()
{
    int hh = 0, tt = 1;
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    memset(pre, 0, sizeof pre);
    memset(st, 0, sizeof st);
    q[0] = S, d[S] = 0, incf[S] = INF;
    while (hh != tt)
    {

```

```

int t = q[hh ++ ];
if (hh == N) hh = 0;
st[t] = false;

for (int i = h[t]; ~i; i = ne[i])
{
    int ver = e[i];
    if (f[i] && d[ver] > d[t] + w[i])
    {
        d[ver] = d[t] + w[i];
        pre[ver] = i;
        incf[ver] = min(f[i], incf[t]);
        if (!st[ver])
        {
            q[tt ++ ] = ver;
            if (tt == N) tt = 0;
            st[ver] = true;
        }
    }
}

return incf[T] > 0;
}

void EK(ll& flow, ll& cost)
{
    flow = cost = 0;
    while (spfa())
    {
        ll t = incf[T];
        flow += t, cost += t * d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1])
        {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
}

int main()
{
    int t;
    cin>>t;
}

```

```
while(t--)
{
    idx=0;
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    ll sum=0;
    for(int i=1; i<=n; i++)
    {
        scanf("%d", &num[i]);
        sum+=num[i];
    }
    while (m -- )
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, INF, c);
        add(b, a, INF, c);
    }
    if(sum%n!=0)
    {
        puts("-1");
        continue;
    }
    S=0, T=n+1;
    sum/=n;
    ll totflow=0;
    for(int i=1; i<=n; i++)
    {

        if(num[i]>sum)add(S,i,num[i]-sum,0),totflow+=num[i]-sum;
        else if(num[i]<sum)add(i,T,sum-num[i],0);
    }
    ll flow, cost;
    EK(flow, cost);
    if(totflow==flow)printf("%lld\n",cost); //判断满流
    else puts("-1");
}

return 0;
}
```

4.5 最小生成树

4.5.1 Kruskal

```
//2020/8/17
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
struct edge
{
    int a,b,z;
    bool operator < (const edge &T) const {
        return z < T.z;
    }
}e[N];
int fa[N],cnt,res;
int find(int x)
{
    if(x!=fa[x])return fa[x]=find(fa[x]);
    return x;
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)fa[i]=i;
    for(int i=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        e[i]={a,b,c};
    }
    sort(e,e+m);
    for(int i=0;i<m;i++)
    {
        int a=e[i].a,b=e[i].b,c=e[i].z;
        a=find(a),b=find(b);
        if(a!=b)
        {
            fa[a]=b;
            //cout<<c<<endl;
            res+=c;
            cnt++;
        }
    }
}
```

```
    }
    if(cnt<n-1)puts("impossible");
    else cout<<res;
}
```

4.6 2Sat

```
/*
2-Sat
每个条件包含两个命题的 SAT 问题，且命题取值 0 或者 1
```

如：对于 x_1, x_2, x_3 使得 $x_1 \vee x_3, \neg x_2 \vee x_3$ 成立
把关系转化为 $\neg \neg$ 的关系。且两边都建边，即别忘了逆否命题。
1. x 和 $\neg x$ 在同一强连通分量，则无解。
2. 枚举所有 xi ，缩点找强连通分量，当 xi 所在的强连通分量的拓扑排序
 \rightarrow 序在 $\neg xi$ 所在的强连通分量的拓扑排序之后时，取 x 为 True
(让拓扑排序靠后的成立)

```
1. aVb 非 a->b 非 b->a
2. a=1 aVa
3. a=0 非 aV 非 a
*/
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cstdio>

using namespace std;

const int N = 2000010, M = 2000010;

int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], ts, stk[N], top;
int id[N], cnt;
bool ins[N];

void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void tarjan(int u)
{
```

```

dfn[u] = low[u] = ++ ts;
stk[ ++ top] = u, ins[u] = true;
for (int i = h[u]; ~i; i = ne[i])
{
    int j = e[i];
    if (!dfn[j])
    {
        tarjan(j);
        low[u] = min(low[u], low[j]);
    } else if (ins[j]) low[u] = min(low[u], dfn[j]);
}

if (low[u] == dfn[u])
{
    int y;
    cnt ++ ;
    do
    {
        y = stk[top -- ], ins[y] = false, id[y] = cnt;
    } while (y != u);
}
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);

    while (m -- )
    {
        int i, a, j, b;
        scanf("%d%d%d%d", &i, &a, &j, &b);
        i -- , j -- ;
        add(2 * i + !a, 2 * j + b);
        add(2 * j + !b, 2 * i + a);
    }

    for (int i = 0; i < n * 2; i ++ )
        if (!dfn[i])
            tarjan(i);
    for (int i = 0; i < n; i ++ )
        if (id[i * 2] == id[i * 2 + 1])
        {
            puts("IMPOSSIBLE");
        }
}

```

```

        return 0;
    }

    puts("POSSIBLE");
    for (int i = 0; i < n; i++) {
        if (id[i * 2] < id[i * 2 + 1]) printf("0
        "); //让拓扑排序靠前的满足
        else printf("1 ");
    }
    return 0;
}

```

4.7 lca

```

#include<bits/stdc++.h>
using namespace std;
const int N=4*1e4+10;
int f[N][16], depth[N], n, root, q[N];
vector<int> g[N];
void bfs(int root)
{
    memset(depth, 0x3f, sizeof depth);
    int hh=0, tt=0;
    q[hh]=root;
    depth[0]=0, depth[root]=1; //depth[0]=0, 设置为哨兵。当 f 跳
    → 出去了，就会跳到 0。
    while(hh<=tt)
    {
        int u=q[hh++];
        for(auto i:g[u])
        {
            if(depth[i]>depth[u]+1)
            {
                depth[i]=depth[u]+1;
                q[++tt]=i;
                f[i][0]=u; //i 是 u 的子节点, i 往上跳一步就到 u
                for(int k=1; k<=15; k++)
                    f[i][k]=f[f[i][k-1]][k-1]; //i 跳 2^k 步相当
                    → 于 i 先跳 2^(k-1) 再跳 2^(k-1)
            }
        }
    }
    int lca(int a, int b)
    {

```

```

if(depth[b]>depth[a])swap(a,b); //让 a 在下面，深度大
for(int k=15;k>=0;k--) //让 a, b 跳到同一层
{
    if(depth[f[a][k]]>=depth[b]) //哨兵的作用：如果跳出去了,
        → depth[0]=0 一定会小于右边，则不会跳
    {
        a=f[a][k];
    }
}
if(a==b) return a; //如果跳到同一个点。
for(int k=15;k>=0;k--) //在同一层一起往上跳，直到跳到祖先的下
→ 面
{
    if(f[a][k]!=f[b][k]) //每次跳同样的步数，不会出现跳过 lca
        → 的情况
    {
        a=f[a][k];
        b=f[b][k];
    }
}
return f[a][0];
}

int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        int u,v;
        cin>>u>>v;
        if(v==-1)root=u;
        else
        {
            g[u].push_back(v);
            g[v].push_back(u);
        }
    }
    bfs(root);
    int q;
    cin>>q;
    while(q--)
    {
        int a,b;
        cin>>a>>b;
    }
}

```

```
    int p=lca(a,b);
    if (p == a) puts("1");
    else if (p == b) puts("2");
    else puts("0");
}
return 0;
}
```

4.8 拓扑排序

```
#include<bits/stdc++.h>
using namespace std;
const int N=105;
vector<int>g[N];
int in[N],n;
void topos()
{
    int q[N],hh=0,tt=-1;
    for(int i=1;i<=n;i++)
        if(!in[i])q[++tt]=i;
    while(hh<=tt)
    {
        int t=q[hh++];
        for(auto i:g[t])
        {
            in[i]--;
            if(!in[i])
                q[++tt]=i;
        }
    }
    for(int i=0;i<hh;i++)printf("%d ",q[i]);
}
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        int x;
        while(cin>>x,x)
        {
            g[i].push_back(x);
            in[x]++;
        }
    }
}
```

```

    }
    topos();
}

return 0;
}

```

4.9 多源最短路

4.9.1 Floyd

```

#include<bits/stdc++.h>
using namespace std;
const int N=205;
int a[N][N];
#define inf 0x3f3f3f3f
int main()
{
    int n,m,q;
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(i==j)a[i][j]=0;
            else a[i][j]=inf;
        }
    }
    while(m--)
    {
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        if(z<a[x][y])a[x][y]=z;
    }
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                if(a[i][j]>a[i][k]+a[k][j])a[i][j]=a[i][k]+a[k][j];
            }
        }
    }
}

```

```

while(q--)
{
    int u,v;
    cin>>u>>v;
    if(a[u][v]>inf/2)puts("impossible");
    else cout<<a[u][v]<<endl;
}
}

```

4.9.2 Floyd 求最小环

```

/*
给定一张无向图，求图中一个至少包含 3 个点的环，环上的节点不重复，  

→ 并且环上的边的长度之和最小。  

该问题称为无向图的最小环问题。  

你需要输出最小环的方案，若最小环不唯一，输出任意一个均可。

```

floyd 找最小环。可以将环分类为环上最大的点的编号为 *k*。*floyd* 其实
 → 是插点法。当最外层循环到 *k* 时，
i,j 之间的最短路由 (*1~k-1*) 这些点更新，这时候就可以顺便枚举环上
 → 最大点为 *k* 了。在最内层循环记得记录 *pos[i][j]* 表示，
i 与 *j* 之间的最短路由 *k* 点转移来。

```

*/
#include<bits/stdc++.h>
using namespace std;
const int N=105,inf=0x3f3f3f3f;
int g[N][N],dis[N][N],path[N],n,m,cnt,pos[N][N];
void getpath(int i,int j)//递归输出 i, j 之间的路径，不包括 ij
{
    if(pos[i][j]==0)return ;//ij 之间没被其他点更新
    int k=pos[i][j];
    getpath(i,k);//左边
    path[cnt++]=k;
    getpath(k,j);//右边
}
int main()
{
    cin>>n>>m;
    memset(g,0x3f,sizeof g);
    for(int i=1;i<=n;i++)g[i][i]=0;
    for(int i=1;i<=m;i++)
    {
        int a,b,w;
        cin>>a>>b>>w;
    }
}

```

```

        g[a][b]=g[b][a]=min(g[b][a],w); //可能有重边
    }
    int res=inf;
    memcpy(dis,g,sizeof g);
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<k;i++) //枚举 ij 的点对就可以了。
            for(int j=i+1;j<k;j++)
            {
                if((long long)dis[i][j]+g[i][k]+g[k][j]<res)
                {
                    res=dis[i][j]+g[i][k]+g[k][j];
                    cnt=0;
                    path[cnt++]=k;
                    path[cnt++]=i;
                    getpath(i,j);
                    path[cnt++]=j;
                }
            }
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
            {
                if(dis[i][j]>dis[i][k]+dis[k][j])
                {
                    dis[i][j]=dis[i][k]+dis[k][j];
                    pos[i][j]=k;
                }
            }
    }
    if(res!=inf)
    {
        for(int i=0;i<cnt;i++)printf("%d ",path[i]);
    }
    else puts("No solution.");
    return 0;
}

```

4.10 单源最短路

4.10.1 SPFA

```

/* 中文注释测试 */
/*
 * Args:

```

```

*   g[]: graph, (u, v, w) = (u, g[u][i].first,
→   g[u][i].second)
*   st: source vertex
* Return:
*   dis[]: distance from source vertex to each other vertex
*/
vector<pair<int, int>> g[N];
int dis[N], vis[N];
void spfa(int st)
{
    memset(dis, -1, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    q.push(st);
    dis[st] = 0;
    vis[st] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (auto x : g[u]) {
            int v = x.first, w = x.second;
            if (dis[v] == -1 || dis[u] + w < dis[v]) {
                dis[v] = dis[u] + w;
                if (!vis[v]) {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
}
    
```

4.10.2 堆优化 dij

```

#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int d[N], n, m;;
#define pii pair<int, int>
vector<pii> g[N];
int dijkstra(int x)
{
    memset(d, 0x3f, sizeof(d));
    
```

```

priority_queue<pii,vector<pii>,greater<pii>>p;
p.push({0,1});
d[1]=0;//别漏了初始化
while(p.size())
{
    auto t=p.top();
    p.pop();
    int dis=t.first,ver=t.second;

    if(d[ver]<dis) continue;
    for(auto i:g[ver])
    {
        int to=i.first,val=i.second;
        if(d[to]>d[ver]+val)
        {
            d[to]=d[ver]+val;
            p.push({d[to],to});
        }
    }
}
if(d[n]==0x3f3f3f3f) return -1;
else return d[n];

}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        g[u].push_back({v,w});
    }
    cout<<dijkstra(1);
    return 0;
}

```

4.10.3 SPFA 判负环

```

#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int d[N],st[N],cnt[N],n,m;//st 表示 x 在不在队列中
#define pii pair<int,int>

```

```

vector<pii>g[N];
int spfa(int x)
{
    memset(d,0x3f,sizeof(d));
    queue<int>q;
    for(int i=1;i<=n;i++)
    {
        q.push(i);
        st[i]=1;
        cnt[i]++;
    }
    while(q.size())
    {
        auto t=q.front();
        q.pop();
        st[t]=0;//每个点可能进队多次
        for(auto i:g[t])
        {
            int a=i.first,b=i.second;
            if(d[a]>d[t]+b)//每次 a 更新， 肯定是因为 t 更新了,
            ↳ 下次 c 更新则是因为 a
            {
                d[a]=d[t]+b;
                if(!st[a])
                {
                    q.push(a);
                    st[a]=1;
                    cnt[a]++;//若某个点进队大于等于 n 次，则有负
                    ↳ 环。写大于好像也可以
                    if(cnt[a]>=n)
                    {
                        return 1;
                    }
                }
            }
        }
    }
    return 0;
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)

```

```
{
    int u,v,w;
    cin>>u>>v>>w;
    g[u].push_back({v,w});
}
int t=spfa(1);
if(t==1)puts("Yes");
else puts("No");
return 0;
}
```

4.11 有向图强连通分量

4.11.1 scc tarjan

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e4+10;
int
→ dfn[N],low[N],scc_cnt,id[N],Size[N],times,stk[N],top,dout[N];
bool in[N];
vector<int>g[N];
void tarjan(int u)
{
    dfn[u]=low[u]=++times;
    stk[++top]=u,in[u]=1;
    for(auto j:g[u])
    {
        if(!dfn[j])
        {
            tarjan(j);
            low[u]=min(low[u],low[j]);
        }
        else if(in[j])
            low[u]=min(low[u],dfn[j]);
    }
    if(dfn[u]==low[u])
    {
        int y;
        scc_cnt++;
        do
        {
            y=stk[top--];
            in[y]=0;
```

```

        id[y]=scc_cnt;
        Size[scc_cnt]++;
    }while(y!=u);
}
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        cin>>u>>v;
        g[u].push_back(v);
    }
    for(int i=1;i<=n;i++)
        if(!dfn[i])
            tarjan(i);
    for(int i=1;i<=n;i++)
        for(auto j:g[i])
        {
            int a=id[i],b=id[j];
            if(a!=b)dout[a]++;
        }
    int zeros=0,sum=0;
    for(int i=1;i<=scc_cnt;i++)
    {
        if(!dout[i])
        {
            zeros++;
            sum+=Size[i];
            if(zeros>1)
            {
                sum=0;
                break;
            }
        }
    }
    cout<<sum;
}

```

4.12 无向图双连通分量

4.12.1 边双联通分量

```
/*
边双连通分量
给一个无向连通图，求加几条边变成一个边双连通分量。
无向图缩点后为一棵树，只剩下桥和点
答案为树的  $(\text{叶子节点数} + 1)/2$  下取整
*/
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 5010, M = 20010;

int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
int id[N], dcc_cnt;
bool is_bridge[M];
int d[N];

void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void tarjan(int u, int from)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u;

    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!dfn[j])
        {
            tarjan(j, i);
            //sz[u]+=sz[j]+1; sz[j] 是 j 这个子树的边数
            low[u] = min(low[u], low[j]);
        }
    }
}
```

```

    if (dfn[u] < low[j])
        is_bridge[i] = is_bridge[i ^ 1] = true; //加边
        ↳ 的时候 01, 23, 45 是一对边
    }
    else if (i != (from ^ 1))
    {
        low[u] = min(low[u], dfn[j]);
        //sz[j]++;
    }
}

if (dfn[u] == low[u])
{
    ++ dcc_cnt;
    int y;
    do {
        y = stk[top --];
        id[y] = dcc_cnt;
    } while (y != u);
}
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m --)
    {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    tarjan(1, -1);
    for (int i = 0; i < idx; i++)
        if (is_bridge[i])
            d[id[e[i]]]++;

    int cnt = 0;
    for (int i = 1; i <= dcc_cnt; i++)
        if (d[i] == 1)
            cnt++;
}

printf("%d\n", (cnt + 1) / 2);

```

```

        return 0;
}

```

4.12.2 点双联通分量

```

//题意：删掉一个点后连通块数最多是多少
#include<bits/stdc++.h>
using namespace std;
const int N = 10010;
int n, m;
int dfn[N], low[N], timestamp;
int root, ans;
vector<int> g[N];
void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;

    int cnt = 0;
    for (auto j:g[u])
    {
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if (low[j] >= dfn[u]) cnt++; //说明 u 下面有一个子
            → 树， u 可能是割点，具体看 u 是不是根讨论
        }
        else low[u] = min(low[u], dfn[j]);
    }

    if (u != root) cnt++; //u 不是根的话，说明该点上面还有一条
    → 边。(dfs 搜索，如果这个点上面不止一条边，肯定被上面的某一条边搜掉了。)
    //if 中的 &&cnt 可以去掉 如果 cnt 为 0 (不是割点) 那么删掉这
    → 个点还会有 1 个连通块
    ans = max(ans, cnt);
    /* 判断割点
    if ((u != root && cnt) || (u==root && cnt>=2))
    {
        iscut[u]=1;
    }
    */
}

```

```
int main()
{
    while (scanf("%d%d", &n, &m), n || m)
    {
        memset(dfn, 0, sizeof dfn);
        timestamp = 0;
        for(int i=0;i<n;i++)g[i].clear();
        while (m -- )
        {
            int a, b;
            scanf("%d%d", &a, &b);
            g[a].push_back(b);
            g[b].push_back(a);
        }

        ans = 0;
        int cnt = 0;
        for (root = 0; root < n; root ++ )//以该点为根搜索
        {
            if (!dfn[root])
            {
                cnt++;
                tarjan(root);
            }
        }

        printf("%d\n", ans + cnt - 1);
    }
    return 0;
}
/*
 * 题意：最少设置几个安全点，使任意删除一个点后，所有点均可以到达
 * 安全区。
tarjan 无向图点双连通分量缩点，割点最少会属于两个双连通分量。
*/
typedef unsigned long long ULL;

const int N = 1010;

int n, m;
int dfn[N], low[N], timestamp;
int stk[N], top;
int dcc_cnt;
vector<int> dcc[N], g[N];//dcc 存这个块包含哪些点
bool cut[N];
int root;
```

```

void tarjan(int u)
{
    dfn[u] = low[u] = ++ timestamp;
    stk[ ++ top] = u;

    if (u == root && !g[u].size())
    {
        dcc_cnt ++ ;
        dcc[dcc_cnt].push_back(u);
        return;
    }

    int cnt = 0;
    for (auto j:g[u])
    {
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if (dfn[u] <= low[j])
            {
                cnt ++ ;
                if (u != root || cnt > 1) cut[u] = true;
                ++ dcc_cnt;
                int y;
                do {
                    y = stk[top -- ];
                    dcc[dcc_cnt].push_back(y);
                } while (y != j);
                dcc[dcc_cnt].push_back(u);
            }
        }
        else low[u] = min(low[u], dfn[j]);
    }
}

int main()
{
    int T = 1;
    while (cin >> m, m)
    {
        for (int i = 1; i <= dcc_cnt; i ++ ) dcc[i].clear();
        n = timestamp = top = dcc_cnt = 0;
        memset(dfn, 0, sizeof dfn);
    }
}

```

```

        memset(cut, 0, sizeof cut);

        while (m -- )
        {
            int a, b;
            cin >> a >> b;
            n = max(n, a), n = max(n, b);
            g[a].push_back(b);
            g[b].push_back(a);
        }

        for (root = 1; root <= n; root ++ )
        if (!dfn[root])
            tarjan(root);
        for(int i=1;i<=n;i++)g[i].clear();
        int res = 0;
        ULL num = 1;
        for (int i = 1; i <= dcc_cnt; i ++ )
        {
            int cnt = 0;
            for (int j = 0; j < dcc[i].size(); j ++ )
                if (cut[dcc[i][j]])
                    cnt ++ ;

            if (cnt == 0)
            {
                if (dcc[i].size() > 1) res += 2, num
                ← *= dcc[i].size() * (dcc[i].size()
                ← - 1) / 2;
                else res ++ ;
            }
            else if (cnt == 1) res ++, num *=
                ← dcc[i].size() - 1;
        }

        printf("Case %d: %d %llu\n", T ++, res, num);
    }

    return 0;
}

```

4.13 二分图

4.13.1 染色法判定二分图

```
//2020/8/18
//有奇数环（边数为奇数）一定不是二分图
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
vector<int>g[N];
int n,m,color[N];
bool dfs(int u,int c)
{
    color[u]=c;
    for(auto i:g[u])
    {
        if(!color[i])
        {
            if(!dfs(i,3-c))return 0;
        }
        else if(color[i]==c) return 0;
    }
    return 1;
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    bool flag= 1;
    for(int i=1;i<=n;i++) //这个图可能有几个块
    {
        if(!color[i])
        {
            if(!dfs(i,1))
            {
                flag=0;
                break;
            }
        }
    }
}
```

```

        }
    }
    if(flag)puts("Yes");
    else puts("No");
    return 0;
}

```

4.13.2 最大匹配 = 最小点覆盖：选出最少的点，使每条边至少有一个点被选出来（在选出的点里面） = 点数-最大独立集：选出最多的点使得选出的点之间没有边。 = 点数-最小路径点覆盖（最小路径覆盖）：针对一个有向无环图（DAG），用最少条互不相交路径，覆盖所有点。（其中互不相交是指点不重复）（匈牙利算法 $O(n * m)$ ）

```

#include<bits/stdc++.h>
using namespace std;
const int N=505;
vector<int>g[N];
int n1,n2,m,match[N]; //match 右边女孩匹配的左边男孩
bool st[N]; //每次循环右边点是否被选,
bool find(int x)
{
    for(auto i:g[x])
    {
        if(!st[i])
        {
            st[i]=1;
            if(match[i]==0||find(match[i]))
            {
                match[i]=x;
                return 1;
            }
        }
    }
    return 0;
}
int main()
{
    cin>>n1>>n2>>m;
    for(int i=0;i<m;i++)
    {
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
    }
}

```

```
}

int res=0;
for(int i=1;i<=n1;i++)
{
    memset(st,0,sizeof(st));
    if(find(i))res++;
}
cout<<res;
return 0;
}
```

5 数据结构

5.1 链表

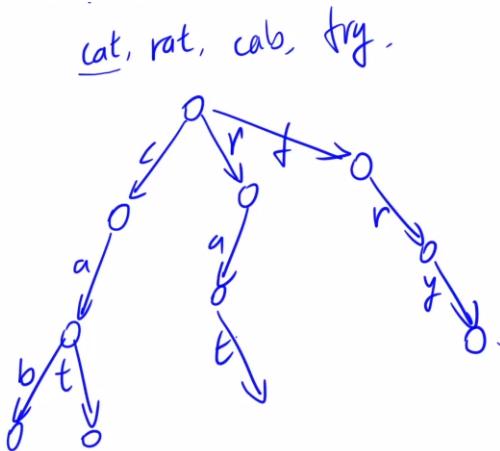
```

void ini()
{
    head=-1, idx=1;
}
void add_to_head(int x)
{
    e[idx]=x, ne[idx]=head, head=idx++;
}
void add(int k, int x) //第 k 个数后插入
{
    e[idx]=x, ne[idx]=ne[k], ne[k]=idx++;
}
void remove(int k)
{
    ne[k]=ne[ne[k]];
}
//双链表
// 在节点 a 的右边插入一个数 x
void insert(int a, int x)
{
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx++;
}

// 删除节点 a
void remove(int a)
{
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
//遍历
for(int i=head; i!=-1; i=ne[i]) cout<<e[i]<< ' ';

```

5.2 字典树



```

//结点存的是编号，边上是字母，代表方向
#include<bits/stdc++.h>
using namespace std;
const int N=2e4+10;
int son[N][26], cnt[N], idx; //cnt 表示以。。结尾的个数
char str[N];
void insert(char s[])
{
    int p=0;
    for(int i=0;s[i];i++)
    {
        int u=s[i]-'a';
        if(!son[p][u])son[p][u]=++idx;
        p=son[p][u];
    }
    cnt[p]++;
}
int query(char s[])
{
    int p=0;
    for(int i=0;s[i];i++)
    {
        int u=s[i]-'a';
        if(!son[p][u])return 0;
        p=son[p][u];
    }
    return cnt[p];
}
  
```

```

int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        char x;
        cin>>x>>str;
        if(x=='I')insert(str);
        else printf("%d\n",query(str));
    }
}

```

5.3 可撤销并查集

```

struct DSU
{
    struct Stack
    {
        int u, v, rk;
    } st[maxn];
    int fa[maxn], rk[maxn], top;
    inline void init()
    {
        top = 0;
        for(int i=1; i<=n; i++)
            fa[i] = i, rk[i] = 0;
    }
    inline int find(int x)
    {
        return x == fa[x] ? x : find(fa[x]);
    }
    inline bool unite(int u, int v)
    {
        int fu = find(u), fv = find(v);
        if(fu == fv)
            return 0;
        if(rk[fu] > rk[fv])
            swap(u, v), swap(fu, fv);
        st[++top] = {fu, fv, rk[fv]};
        fa[fu] = fv, rk[fv] += rk[fu] == rk[fv];
        return 1;
    }
    inline void undo(int k)//撤回 k 次

```

```
{
    while(k--)
    {
        Stack &now = st[top--];
        int u = now.u, v = now.v;
        fa[u] = u, rk[v] = now.rk;
    }
}
}dd;
```

5.4 线段树

5.4.1 单点修改

```
//2020/9/2
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
struct node{
    int l,r;//该节点代表的区间 [l~r]
    int v;
}tr[4*N];//某个结点
int m,p;
inline void pushup(int u)
{
    tr[u].v=max(tr[u<<1].v,tr[u<<1|1].v);
}
void build(int u,int l,int r)
{
    tr[u]={l,r};
    if(l==r) return ;
    int mid=l+r>>1;
    build(u<<1,l,mid),build(u<<1|1,mid+1,r);
}
int query(int u,int l,int r)
{
    if(tr[u].l>=l&&tr[u].r<=r) return tr[u].v;//当前区间包含于查
    ↳ 询区间 [l,r], 这样最后肯定是多段区间 return
    int mid=tr[u].l+tr[u].r>>1;
    int v=0;
    //与左区间有交集
    if(l<=mid)v=query(u<<1,l,r);//递归区间, 查询区间始终是 [l,r]
    //与右区间有交集
    if(r>mid)v=max(query(u<<1|1,l,r),v);
}
```

```
    return v;
}
void update(int u,int x,int v)
{
    if(tr[u].l==x&&tr[u].r==x)
    {
        tr[u].v=v;
    }
    else
    {
        int mid=tr[u].l+tr[u].r>>1;
        if(x<=mid)update(u<<1,x,v);
        else update(u<<1|1,x,v);
        pushup(u);
    }
}

int main()
{
    cin>>m>>p;
    build(1,1,m);
    int last=0,n=0;
    while(m--)
    {
        char op[2];
        int x;
        scanf("%s%d",&op,&x); //不能 scanf("%c%d")%c 连空格回车
        // 都会读入，把回车读进去了，除非后面加个 getchar
        if(*op=='A')
        {
            update(1,n+1,(x+last)%p);
            n++;
        }
        else
        {
            last=query(1,n-x+1,n);
            printf("%d\n",last);
        }
    }
    return 0;
}
```

5.4.2 区间修改

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 100010;

int n, m;
int w[N];
struct Node
{
    int l, r;
    LL sum, add;
} tr[N * 4];

void pushup(int u)
{
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

void pushdown(int u)
{
    auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1
        | 1];
    if (root.add)
    {
        left.add += root.add, left.sum += (LL)(left.r - left.l
            + 1) * root.add;
        right.add += root.add, right.sum += (LL)(right.r -
            right.l + 1) * root.add;
        root.add = 0;
    }
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = {l, r, w[r], 0};
    else
```

```

{
    tr[u] = {l, r};
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    pushup(u);
}
}

void modify(int u, int l, int r, int d)
{
    if (tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].sum += (LL)(tr[u].r - tr[u].l + 1) * d;
        tr[u].add += d;
    }
    else // 一定要分裂
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, d);
        if (r > mid) modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}

LL query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum += query(u << 1 | 1, l, r);
    return sum;
}

int main()
{
    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
}

```

```

build(1, 1, n);

char op[2];
int l, r, d;

while (m --)
{
    scanf("%s%d%d", op, &l, &r);
    if (*op == 'C')
    {
        scanf("%d", &d);
        modify(1, l, r, d);
    }
    else printf("%lld\n", query(1, l, r));
}

return 0;
}

```

5.4.3 返回 node 的线段树（区间最大字段和）

```

/*
跨区间写的时候，即 pushup, 切记
u.l=L.l, u.r=R.r;
u.pre=L.pre, u.suf=R.suf; 之类的，因为 u 可能啥都没有，要先继承
*/
#include<bits/stdc++.h>
using namespace std;
const int N=500005;
int a[N];
struct node
{
    int l,r;
    int tmax,lmax,rmax,sum;//最大子段和，最大前缀和，最大后缀和,
    → 区间和
    }tr[4*N];
inline void pushup(node &u,node &l,node &r)
{
    u.sum=l.sum+r.sum;
    u.lmax=max(l.lmax,l.sum+r.lmax);
    u.rmax=max(r.rmax,r.sum+l.rmax);
    u.tmax=max(l.tmax,max(r.tmax,l.rmax+r.lmax));
}
inline void pushup(int u)

```

```

{
    pushup(tr[u], tr[u<<1], tr[u<<1|1]);
}
void build(int u, int l, int r)
{
    tr[u]={l,r};
    if(l==r)
    {
        tr[u]={l,r,a[l],a[l],a[l],a[l]};
        return ;
    }
    int mid=l+r>>1;
    build(u<<1,l,mid),build(u<<1|1,mid+1,r);
    pushup(u);
}
void update(int u, int x, int v)
{
    if(tr[u].l==x&&tr[u].r==x)
    {
        tr[u]={x,x,v,v,v,v};
        return ;
    }
    int mid=tr[u].l+tr[u].r>>1;
    if(x<=mid)update(u<<1,x,v);
    else update(u<<1|1,x,v);
    pushup(u);
}
node query(int u, int l, int r)
{
    if(tr[u].l>=l&&tr[u].r<=r)
    {
        return tr[u];
    }
    int mid=tr[u].l+tr[u].r>>1;
    if(r<=mid)return query(u<<1,l,r); //查询区间在当前区间 mid
    ↳ 左侧
    else if(l>mid)return query(u<<1|1,l,r); //查询区间在当前区间
    ↳ mid 右侧
    else//查询区间跨过 mid, 则要 pushup
    {
        auto left=query(u<<1,l,r);
        auto right=query(u<<1|1,l,r);
        node tree;
    }
}

```

```

        pushup(tree, left, right);
        return tree;
    }
}
int main()
{
    int n, m;
    cin >> n >> m;
    for(int i=1; i<=n; i++) scanf("%d", &a[i]);
    build(1, 1, n);
    while(m--)
    {
        int op, x, y;
        scanf("%d%d%d", &op, &x, &y);
        if(op==1)
        {
            if(x>y) swap(x, y);
            printf("%d\n", query(1, x, y).tmax);
        }
        else
        {
            update(1, x, y);
        }
    }
    return 0;
}

```

5.4.4 括号序列线段树（区间括号序列是否合法）

```

#include<bits/stdc++.h>

using namespace std;
const int maxn=1e6+10;
char c[maxn];
struct in
{
    int a, b, sum;
    in() { a=0; b=0; sum=0; }
}s[maxn<<2];
void read(int &res)
{
    char c=getchar(); res=0; for(; c>'9' || c<'0'; c=getchar());
    for(; c<='9' && c>='0'; c=getchar())
        res=(res<<3)+(res<<1)+c-'0';
}

```

```

}

struct segment_tree
{
    void pushup(int Now)
    {
        int tmp=min(s[Now<<1].a,s[Now<<1|1].b);
        s[Now].sum=s[Now<<1].sum+s[Now<<1|1].sum+tmp;
        s[Now].a=s[Now<<1].a+s[Now<<1|1].a-tmp;
        s[Now].b=s[Now<<1].b+s[Now<<1|1].b-tmp;
    }
    void build(int Now,int L,int R)
    {
        if(L==R) {
            s[Now].a=(c[L]=='?'?1:0);
            s[Now].b=(c[L]=='!'?1:0);
            s[Now].sum=0; return ;
        }
        int Mid=(L+R)>>1;
        build(Now<<1,L,Mid);
        build(Now<<1|1,Mid+1,R);
        pushup(Now);
    }
    in query(int Now,int L,int R,int l,int r)
    {
        if(l<=L&&r>=R) return s[Now];
        int Mid=(L+R)>>1; in w,e,res;
        if(l<=Mid) w=query(Now<<1,L,Mid,l,r);
        if(r>Mid) e=query(Now<<1|1,Mid+1,R,l,r);
        res.sum=min(w.a,e.b);
        res.a=w.a+e.a-res.sum;
        res.b=w.b+e.b-res.sum;
        res.sum+=w.sum+e.sum;
        return res;
    }
    void update(int Now,int L,int R,int v,char x)
    {
        if(L==v&&R==v)
        {
            s[Now].a=(x=='('?1:0);
            s[Now].b=(x=='!'?1:0);
            s[Now].sum=0;
            return ;
        }
    }
}

```

```
    int mid=L+R>>1;
    if(v<=mid) update(Now<<1,L,mid,v,x);
    if(v>mid) update(Now<<1|1,mid+1,R,v,x);
    pushup(Now);
}
}Tree;

int main()
{
    int N,Q,x,y;
    read(N);
    read(Q);
    scanf("%s",c+1);
    Tree.build(1,1,N);

    while(Q--){
        int op;
        read(op),read(x); read(y);
        if(op==2)
        {
            if(2*Tree.query(1,1,N,x,y).sum==y-x+1)puts("Yes");
            else puts("No");
        }
        else
        {
            Tree.update(1,1,N,x,c[y]);
            Tree.update(1,1,N,y,c[x]);
            swap(c[x],c[y]);
        }
    }
    return 0;
}
```

5.5 可持久化数据结构（历史版本）

5.5.1 主席树第 K 小数

```
/*
主席树。可持久化权值线段树。权值即比如权值在 1-4 区间的数个数。6 4
→ 2 3 4 则权值在 1-4 中有 4 个数。
1-R 版本中权值在某区间的个数。用类似前缀和可以求任意一段区间
*/
#include<bits/stdc++.h>
```

```

using namespace std;
const int N = 1e5 + 10, M = 1e4 + 10;
int n, m;
int a[N], t[N];
struct node {
    int l, r; //存左子树和右子树的结点 idx 下标。
    int cnt; //区间中有几个数出现了。
} tr[N * 4 + 17 * N]; //初始就要 N*4, 后续插入每个数最多 logn, 插
→ 入 n 次。
int root[N], idx; //每个版本的根节点 idx
int build(int l, int r) {
    int p = ++idx;
    if (l == r) return p;
    int mid = l + r >> 1;
    tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
    return p;
}

int insert(int p, int l, int r, int x) {
    int q = ++idx;
    tr[q] = tr[p];
    if (l == r) {
        tr[q].cnt++;
        return q;
    }
    int mid = l + r >> 1;
    if (x <= mid) tr[q].l = insert(tr[p].l, l, mid, x); //新的东
    → 西要新开结点, 其他的全部复制过来
    else tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
    return q;
}

int query(int q, int p, int l, int r, int k) //返回离散化后下标
{
    if (l == r) return r;
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt; //两个版本之间
    → 的区间。在左 (1~mid) 区间范围内数的个数
    int mid = l + r >> 1;
    if (k <= cnt) return query(tr[q].l, tr[p].l, l, mid, k);
    else return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
}

```

```

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        t[i] = a[i];
    }
    sort(t + 1, t + 1 + n);
    int len = unique(t + 1, t + 1 + n) - (t + 1);
    for (int i = 1; i <= n; i++)
        a[i] = lower_bound(t + 1, t + 1 + n, a[i]) - t;
    root[0] = build(1, len); //似乎不要也可以
    for (int i = 1; i <= n; i++)
        root[i] = insert(root[i - 1], 1, len, a[i]);
    while (m--) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", t[query(root[r], root[l - 1], 1, len,
                               k)]);
    }
    return 0;
}

```

5.5.2 可持久化 trie 求 L-R 中某个后缀与 X 最大异或和

```

/*
01 可持久化 trie, 求区间 L-R 与 C 的最大异或
你需要找到一个位置 p, 满足 l p r, 使得: a[p] xor a[p+1] xor ...
→ xor a[N] xor x 最大, 输出这个最大值。
第 R 个版本存的就是 1-R 所有数的信息
*/
#include<bits/stdc++.h>
using namespace std;
const int N=6e5+10,M=25*N;
int s[N];
int tr[M][2],max_id[M];//maxid 代表结点 idx 的子树 (的数中) 对应
→ 最大的下标 i
int root[N],idx;//root 代表第 i 个数的结点 idx
void insert(int k,int p,int q)//第 i 个数, 上一个版本, 当前版本
{
    for(int i=23;i>=0;i--)
    {
        int v=s[k]>>i&1;
        if(p)tr[q][v^1]=tr[p][v^1];//复制除了当前 v 以外的信息,
        → 因为当前的都是新开结点
    }
}

```

```

//上面都一样，复制过来以后就可以从新版本把前面全部都走一遍
tr[q][v]=++idx;
max_id[idx]=k;
p=tr[p][v],q=tr[q][v];
}
}
int query(int p,int l,int c)
{
    for(int i=23;i>=0;i--)
    {
        int v=c>>i&1;
        if(max_id[tr[p][v^1]]>=l)p=tr[p][v^1];//如果相反的方向的子树最大下标 >=l 就走过去（因为可以取到那个数）
        else p=tr[p][v];
    }
    return c^s[max_id[p]];
}
int main()
{
    max_id[0]=-1;//假如不存在 tr[p][v^1] 这个结点。则结点为 0.
    //而 l 可能取到 0, 所以设为 -1
    root[0]=++idx;//在 trie 树中 0 既是根节点，又是空节点，意义非凡，得先空出来。
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        int x;
        scanf("%d",&x);
        s[i]=s[i-1]^x;
        root[i]=++idx;
        insert(i,root[i-1],root[i]);
    }
    char op[2];
    while(m--)
    {
        scanf("%s",&op);
        if(*op=='A')//序列末尾添加一个元素
        {
            int x;
            scanf("%d",&x);
            n++;
        }
    }
}

```

```

        root[n]=++idx;
        s[n]=s[n-1]^x;
        insert(n,root[n-1],root[n]);
    }
    else//你需要找到一个位置 p, 满足 l p r, 使得: a[p] xor
    → a[p+1] xor ... xor a[N] xor x 最大, 输出这个最大值。
    {
        int l,r,x;
        scanf("%d%d%d",&l,&r,&x);
        printf("%d\n",query(root[r-1],l-1,s[n]^x));
    }
}
return 0;
}

```

5.6 树套树

5.6.1 线段树套线段树求区间 max,sum,min

```

/*
n,m 在 3000 以内可以用。
线段树套线段树, 求二维区间最大值, 最小值, 和。注意每次 query 都要
→ 初始化 maxV, minV, sum。
向下是 x 轴 (n), 向右是 y 轴 (m)
*/
#define ll long long
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 1024 + 5;
int MAX[N << 2][N << 2], minV, maxV, MIN[N<<2][N<<2]; //维护最值
int a[N<<2][N<<2]; //初始矩阵
int SUM[N<<2][N<<2], sumV; //维护求和
int n,m;
void pushupX(int deep, int rt)
{
    MAX[deep][rt] = max(MAX[deep << 1][rt], MAX[deep << 1 |
    → 1][rt]);
    MIN[deep][rt] = min(MIN[deep << 1][rt], MIN[deep << 1 |
    → 1][rt]);
    SUM[deep][rt] = SUM[deep<<1][rt]+SUM[deep<<1|1][rt];
}
void pushupY(int deep, int rt)
{

```

```

MAX[deep][rt] = max(MAX[deep][rt << 1], MAX[deep][rt << 1
→ | 1]);
MIN[deep][rt] = min(MIN[deep][rt << 1], MIN[deep][rt << 1
→ | 1]);
SUM[deep][rt]=SUM[deep][rt<<1]+SUM[deep][rt<<1|1];
}

void buildY(int ly, int ry, int deep, int rt, int flag)
{
    //y 轴范围 ly,ry;deep,rt; 标记 flag
    if (ly == ry){
        if (flag!=-1)
            MAX[deep][rt] = MIN[deep][rt] = SUM[deep][rt] =
            → a[flag][ly];
        else
            pushupX(deep, rt);
        return;
    }
    int mid = (ly + ry) >> 1;
    buildY(ly, mid, deep, rt << 1, flag);
    buildY(mid + 1, ry, deep, rt << 1 | 1, flag);
    pushupY(deep, rt);
}
void buildX(int lx, int rx, int deep)
{
    //建树 x 轴范围 lx,rx;deep
    if (lx == rx){
        buildY(1, m, deep, 1, lx);
        return;
    }
    int mid = (lx + rx) >> 1;
    buildX(lx, mid, deep << 1);
    buildX(mid + 1, rx, deep << 1 | 1);
    buildY(1, m, deep, 1, -1);
}
void updateY(int Y, int val, int ly, int ry, int deep, int rt,
→ int flag)
{
    //单点更新 y 坐标; 更新值 val; 当前操作 y 的范围
    → ly,ry;deep,rt; 标记 flag
    if (ly == ry){
        if (flag) //注意读清楚题意, 看是单点修改值还是单点加值
            MAX[deep][rt] = MIN[deep][rt] = SUM[deep][rt] =
            → val;
        else pushupX(deep, rt);
    }
}

```

```
        return;
    }
    int mid = (ly + ry) >> 1;
    if (Y <= mid)
        updateY(Y, val, ly, mid, deep, rt << 1, flag);
    else
        updateY(Y, val, mid + 1, ry, deep, rt << 1 | 1, flag);
    pushupY(deep, rt);
}
void updateX(int X, int Y, int val, int lx, int rx, int deep)
{
    //单点更新范围 x,y; 更新值 val; 当前操作 x 的范围 lx,rx;deep
    if (lx == rx){
        updateY(Y, val, 1, m, deep, 1, 1);
        return;
    }
    int mid = (lx + rx) >> 1;
    if (X <= mid)
        updateX(X, Y, val, lx, mid, deep << 1);
    else
        updateX(X, Y, val, mid + 1, rx, deep << 1 | 1);
    updateY(Y, val, 1, m, deep, 1, 0);
}
void queryY(int Yl, int Yr, int ly, int ry, int deep, int rt)
{
    //询问区间 y 轴范围 y1,y2; 当前操作 y 的范围 ly,ry;deep,rt
    if (Yl <= ly && ry <= Yr)
    {
        minV = min(MIN[deep][rt], minV);
        maxV = max(MAX[deep][rt], maxV);
        sumV += SUM[deep][rt];
        return;
    }
    int mid = (ly + ry) >> 1;
    if (Yl <= mid)
        queryY(Yl, Yr, ly, mid, deep, rt << 1);
    if (mid < Yr)
        queryY(Yl, Yr, mid + 1, ry, deep, rt << 1 | 1);
}
void queryX(int Xl, int Xr, int Yl, int Yr, int lx, int rx,
           int rt)
{
    //询问区间范围 x1,x2,y1,y2; 当前操作 x 的范围 lx,rx;rt
    if (Xl <= lx && rx <= Xr){
```

```

        queryY(Yl, Yr, 1, m, rt, 1);
        return;
    }
    int mid = (lx + rx) >> 1;
    if (Xl <= mid)
        queryX(Xl, Xr, Yl, Yr, lx, mid, rt << 1);
    if (mid < Xr)
        queryX(Xl, Xr, Yl, Yr, mid + 1, rx, rt << 1 | 1);
}
ll w[N][N], sum[N][N];
int main()
{
    scanf("%d%d", &n, &m);
    int h1, w1, h2, w2;
    cin >> h1 >> w1 >> h2 >> w2;
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
    {
        scanf("%lld", &w[i][j]);
        → sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + w[i][j];
    }
    for(int i=min(h2, h1); i<=n; i++)
        for(int j=min(w2, w1); j<=m; j++)
    {
        → a[i][j] = sum[i][j] - sum[i-min(h2, h1)][j] - sum[i][j-min(w2, w1)] + sum[i-min(h2, h1)][j-min(w2, w1)];
    }
    buildX(1, n, 1);
    ll ans = 0;
    for(int i=h1; i<=n; i++)
        for(int j=w1; j<=m; j++)
    {
        maxV = 0;
        int x = i - h1 + min(h1, h2), y = j - w1 + min(w1, w2);
        queryX(x, i, y, j, 1, n, 1); // 填入 x1, x2, y1, y2 注意
        → x1 <= x2 y1 <= y2
        ll
        → now = sum[i][j] - sum[i-h1][j] - sum[i][j-w1] + sum[i-h1][j-w1];
        ans = max(ans, now - maxV);
    }
    cout << ans;
    return 0;
}

```

5.7 树状数组

```

int lowbit(int x)
{
    return x& -x;
}
int ask(int x) // x 前缀和
{
    int res=0;
    for(int i=x;i;i-=lowbit(i))res+=tr[i];
    return res;
}
void add(int x,int c)
{
    for(int i=x;i<=n;i+=lowbit(i))tr[i]+=c;
}

```

5.8 RMQ (st 表)

```

//2020/10/18
//RMQ 也叫 st 表，解决区间最值
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10,M=18;
int w[N],f[N][M],n,q; // f 代表从 i 开始 2^j 长度（点）的区间
void ini()
{
    for(int j=0;j<M;j++) // 类似区间 dp，要从小的长度开始
        for(int i=1;i+(1<<j)-1<=n;i++)
    {
        if(!j)f[i][j]=w[i]; // j=0, 即长度为 1, 只有一个点
        else
            f[i][j]=max(f[i][j-1],f[i+(1<<j-1)][j-1]); // 区间平分。
    }
}
int query(int l,int r)
{
    int t=r-l+1;
    k=log2(t); // 2^k 再乘 2 则必定大于 l-r 长度
    return max(f[l][k],f[r-(1<<k)+1][k]); // 分为 l 前缀, r 后缀
    ↳ 两个区间 [l, l-2^k], [r-2^k+1, r]
}

```

```
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)cin>>w[i];
    cin>>q;
    ini();
    while(q--)
    {
        int l,r;
        cin>>l>>r;
        cout<<query(l,r)<<endl;
    }
    return 0;
}
```

5.9 单调队列

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,k,q[N],a[N];
int main()
{
    cin>>n>>k;
    for(int i=0;i<n;i++)cin>>a[i];
    int hh=0,tt=-1;
    for(int i=0;i<n;i++)//求窗口最小值
    {
        if(hh<=tt&&q[hh]<i-k+1)hh++;
        while(hh<=tt&&a[q[tt]]>=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    puts("");
    hh=0,tt=-1;
    for(int i=0;i<n;i++)//求窗口最大值
    {
        if(hh<=tt&&q[hh]<i-k+1)hh++;
        while(hh<=tt&&a[q[tt]]<=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    return 0;
}
```

5.10 点分治

```

/*
点分治
1. 连通块内的答案
2. 重心连接两个连通块的点的答案 (get 所有连通块的点-sum: get (某
→ 个连通块的点)) 就是答案。容斥的思想，因为一个联通块内肯定是自
→ 己块内选两个点，而不是自己跑到重心又跑回来自己联通块。
3. 路径有一端是重心
*/
#include<bits/stdc++.h>
using namespace std;
const int N=1e4+10;
#define pii pair<int,int>
int n,m;
int p[N],q[N];//p 存的是所有连通块中的点到重心的距离, q 存的是某
→ 连通块中的点到重心的距离
vector<pii>g[N];
bool st[N];
int get_wc(int u,int fa,int tot,int &wc)//求重心（不一定是重心,
→ 但满足时间复杂度要求），返回值 sz 是包含 u 的子树大小
{
    if(st[u])return 0;
    int ms=0,sz=1;
    for(auto i:g[u])
    {
        if(i.first==fa)continue;
        int t=get_wc(i.first,u,tot,wc);
        sz+=t;
        ms=max(ms,t);
    }
    ms=max(ms,tot-sz);//ms 是最大的连通块大小, tot-sz 表示父连通
→ 块大小。
    if(ms<=tot/2)wc=u;//满足时间复杂度的要求，每个连通块必须是原
→ 大小/2,
    return sz;
}
int get_size(int u,int fa)//u 为子树的大小
{
    if(st[u])return 0;
    int res=1;
    for(auto i:g[u])
    {
        if(i.first==fa)continue;

```

```

        res+=get_size(i.first,u);
    }
    return res;
}
void get_dis(int u,int fa,int dis,int &qt)//求 u 子树到重心的距离
{
    if(st[u])return ;
    q[qt++]=dis;
    for(auto i:g[u])
    {
        if(i.first==fa)continue;
        get_dis(i.first,u,dis+i.second,qt);
    }
}
int get(int a[],int k)//求 a 中任取两点和 <=m 的方案数
{
    sort(a,a+k);
    int ans=0;
    for(int i=0;i<k;i++)
    {
        int r=upper_bound(a,a+k,m-a[i])-a;
        r--;
        if(r<=i)break;
        ans+=r-i;
    }
    return ans;
}
int cal(int u)
{
    if(st[u])return 0;
    get_wc(u,-1,get_size(u,-1),u);//求重心
    st[u]=1;
    int pt=0,res=0;//pt 是 p 数组大小
    for(auto i:g[u])//遍历被重心划分的每一个连通块
    {
        int qt=0;
        get_dis(i.first,u,i.second,qt); //计算被重心划分的每个连通块中的点到重心的距离，存在 q 中
        for(int j=0;j<qt;j++)
        {
            if(q[j]<=m)res++; //同一连通块中任意点到重心距离 <=m
            // 也是合法方案
        }
    }
}

```

```

        p[pt++]=q[j];
    }
    res-=get(q,qt); //减掉同一连通块中任意两点到重心距离和
    → <=m 的方案数。
}
res+=get(p,pt); //pt 存所有的点
for(auto i:g[u])res+=cal(i.first); //递归的答案
return res;
}
int main()
{
    while(cin>>n>>m)
    {
        if(n==0&&m==0)break;
        for(int i=0;i<=n;i++)g[i].clear(),st[i]=0;
        for(int i=1;i<=n-1;i++)
        {
            int a,b,c;
            scanf("%d%d%d",&a,&b,&c);
            g[a].push_back({b,c});
            g[b].push_back({a,c});
        }
        printf("%d\n",cal(0));
    }
    return 0;
}

```

5.11 树链剖分

```

/*
树链剖分，把树中某条路径转化为 logn 段区间。配合线段树等数据结构
→ 使用
因为 dfs 序先搜重儿子，所以重链节点编号是连续的。
*/
#include<bits/stdc++.h>

using namespace std;

typedef long long LL;
const int N = 100010, M = N * 2;

int n, m;
int w[N], h[N], e[M], ne[M], idx;
int id[N], nw[N], cnt; //dfs 序, nw 表示 dfs 序中, 点的权值

```

```

int dep[N], sz[N], top[N], fa[N], son[N]; //节点深度, 子树大小,
→ 当前重链的顶点, 当前点的父亲, 当前点的重儿子
struct Tree
{
    int l, r;
    LL add, sum;
}tr[N * 4];

void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void dfs1(int u, int father, int depth)//求重儿子
{
    dep[u] = depth, fa[u] = father, sz[u] = 1;
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (j == father) continue;
        dfs1(j, u, depth + 1);
        sz[u] += sz[j];
        if (sz[son[u]] < sz[j]) son[u] = j; //重儿子子树节点更多
    }
}

void dfs2(int u, int t)//求 dfs 序, 先遍历重儿子。t 是顶点
{
    id[u] = ++ cnt, nw[cnt] = w[u], top[u] = t;
    if (!son[u]) return; //叶子结点
    dfs2(son[u], t); //先遍历重儿子
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (j == fa[u] || j == son[u]) continue;
        dfs2(j, j); //轻儿子的顶点就是自己
    }
}

void pushup(int u)
{
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

```

```

void pushdown(int u)
{
    auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1
        ↵ | 1];
    if (root.add)
    {
        left.add += root.add, left.sum += root.add * (left.r -
            ↵ left.l + 1);
        right.add += root.add, right.sum += root.add *
            ↵ (right.r - right.l + 1);
        root.add = 0;
    }
}

void build(int u, int l, int r)
{
    tr[u] = {l, r, 0, nw[r]};
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void update(int u, int l, int r, int k)
{
    if (l <= tr[u].l && r >= tr[u].r)
    {
        tr[u].add += k;
        tr[u].sum += k * (tr[u].r - tr[u].l + 1);
        return;
    }
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if (l <= mid) update(u << 1, l, r, k);
    if (r > mid) update(u << 1 | 1, l, r, k);
    pushup(u);
}

LL query(int u, int l, int r)
{
    if (l <= tr[u].l && r >= tr[u].r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL res = 0;
}

```

```

if (l <= mid) res += query(u << 1, l, r);
if (r > mid) res += query(u << 1 | 1, l, r);
return res;
}

void update_path(int u, int v, int k)//将 u 到 v 的路径上的点加
→ 上 k
{
    while (top[u] != top[v])//向上爬找到相同重链；同时维护所有经
    → 过的重链
    {
        if (dep[top[u]] < dep[top[v]]) swap(u, v); //u 比 v 更
        → 深
        update(1, id[top[u]], id[u], k); //重链顶点编号比当前点
        → 小。
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    update(1, id[v], id[u], k);
}

LL query_path(int u, int v)//求 u 到 v 路径的权值和
{
    LL res = 0;
    while (top[u] != top[v])
    {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        res += query(1, id[top[u]], id[u]);
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    res += query(1, id[v], id[u]);
    return res;
}

void update_tree(int u, int k)//将子树都修改
{
    update(1, id[u], id[u] + sz[u] - 1, k);
}

LL query_tree(int u)//查询子树
{
    return query(1, id[u], id[u] + sz[u] - 1);
}

```

```
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    memset(h, -1, sizeof h);
    for (int i = 0; i < n - 1; i++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }
    dfs1(1, -1, 1);
    dfs2(1, 1);
    build(1, 1, n);

    scanf("%d", &m);
    while (m--)
    {
        int t, u, v, k;
        scanf("%d%d", &t, &u);
        if (t == 1)//u 到 v 的路径上的点加上 k
        {
            scanf("%d%d", &v, &k);
            update_path(u, v, k);
        }
        else if (t == 2)//u 子树加上 k
        {
            scanf("%d", &k);
            update_tree(u, k);
        }
        else if (t == 3)//查询 u 到 v 路径和
        {
            scanf("%d", &v);
            printf("%lld\n", query_path(u, v));
        }
        else printf("%lld\n", query_tree(u));//查询 u 的子树和
    }

    return 0;
}
```

5.12 Splay

```
//给定一个长度为 n 的整数序列，初始时序列为 {1,2,⋯,n-1,n}。
//序列中的位置从左到右依次标号为 1 n。
//我们用 [l,r] 来表示从位置 l 到位置 r 之间（包括两端点）的所有数
→ 字构成的子序列。
//现在要对该序列进行 m 次操作，每次操作选定一个子序列 [l,r]，并将
→ 该子序列中的所有数字进行翻转。
//例如，对于现有序列 1 3 2 4 6 5 7，如果某次操作选定翻转子序列为
→ [3,6]，那么经过这次操作后序列变为 1 3 5 6 4 2 7。
//请你求出经过 m 次操作后的序列。
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n, m;
struct Node
{
    int s[2], p, v;
    int size, flag;

    void init(int _v, int _p)
    {
        v = _v, p = _p;
        size = 1;
    }
} tr[N];
int root, idx;

void pushup(int x)
{
    tr[x].size = tr[tr[x].s[0]].size + tr[tr[x].s[1]].size +
→ 1;
}

void pushdown(int x)
{
    if (tr[x].flag)
    {
```

```

        swap(tr[x].s[0], tr[x].s[1]);
        tr[tr[x].s[0]].flag ^= 1;
        tr[tr[x].s[1]].flag ^= 1;
        tr[x].flag = 0;
    }
}

void rotate(int x)
{
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x; // k=0 表示 x 是 y 的左儿子；k=1
    // 表示 x 是 y 的右儿子
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}

void splay(int x, int k)
{
    while (tr[x].p != k)
    {
        int y = tr[x].p, z = tr[y].p;
        if (z != k)
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y))
                // rotate(x);
            else rotate(y);
        rotate(x);
    }
    if (!k) root = x;
}

void insert(int v)
{
    int u = root, p = 0;
    while (u) p = u, u = tr[u].s[v > tr[u].v];
    u = ++ idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(u, 0);
}

int get_k(int k)
{

```

```
int u = root;
while (true)
{
    pushdown(u);
    if (tr[tr[u].s[0]].size >= k) u = tr[u].s[0];
    else if (tr[tr[u].s[0]].size + 1 == k) return u;
    else k -= tr[tr[u].s[0]].size + 1, u = tr[u].s[1];
}
return -1;

void output(int u)
{
    pushdown(u);
    if (tr[u].s[0]) output(tr[u].s[0]);
    if (tr[u].v >= 1 && tr[u].v <= n) printf("%d ", tr[u].v);
    if (tr[u].s[1]) output(tr[u].s[1]);
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n + 1; i++) insert(i);
    while (m--)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        l = get_k(l), r = get_k(r + 2);
        splay(l, 0), splay(r, 1);
        tr[tr[r].s[0]].flag ^= 1;
    }
    output(root);
    return 0;
}
```

5.13 莫队

5.13.1 基础莫队

```
/*
分块大小  $n/\sqrt{m}n/\sqrt{m}$ , 复杂度  $n*\sqrt{m}n*\sqrt{m}$ 
可以处理  $n=1e5, m=1e6n=1e5, m=1e6$  的问题
玄学优化：奇数块右端点小到大，偶数块右端点大到小排序
*/
```

```
using namespace std;

const int N = 500010, M = 200010, S = 1000010;

int n, m, len;
int w[N], ans[M];
struct Query
{
    int id, l, r;
}q[M];
int cnt[S];

int get(int x)
{
    return x / len;
}

bool cmp(const Query& a, const Query& b)
{
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    return a.r < b.r;
}

void add(int x, int& res)
{
    if (!cnt[x]) res += 1;
    cnt[x] += 1;
}

void del(int x, int& res)
{
    cnt[x] -= 1;
    if (!cnt[x]) res -= 1;
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    scanf("%d", &m);
    len = max(1, (int)sqrt((double)n * n / m));
```

```

for (int i = 0; i < m; i ++ )
{
    int l, r;
    scanf("%d%d", &l, &r);
    q[i] = {i, l, r};
}
sort(q, q + m, cmp);

for (int k = 0, i = 0, j = 1, res = 0; k < m; k ++ )
{
    int id = q[k].id, l = q[k].l, r = q[k].r;
    while (i < r) add(w[ ++ i], res);
    while (i > r) del(w[i -- ], res);
    while (j < l) del(w[j ++ ], res);
    while (j > l) add(w[ -- j], res);
    ans[id] = res;
}

for (int i = 0; i < m; i ++ ) printf("%d\n", ans[i]);
return 0;
}

```

5.13.2 带修莫队

```

#include <iostream>
#include <cstring>
#include <cstdio>
#include <algorithm>
#include <cmath>

using namespace std;

const int N = 10010, S = 1000010;

int n, m, mq, mc, len;
int w[N], cnt[S], ans[N];
struct Query
{
    int id, l, r, t;
    }q[N];
struct Modify
{
    int p, c;
    }c[N];

```

```
int get(int x)
{
    return x / len;
}

bool cmp(const Query& a, const Query& b)
{
    int al = get(a.l), ar = get(a.r);
    int bl = get(b.l), br = get(b.r);
    if (al != bl) return al < bl;
    if (ar != br) return ar < br;
    return a.t < b.t;
}

void add(int x, int& res)
{
    if (!cnt[x]) res ++ ;
    cnt[x] ++ ;
}

void del(int x, int& res)
{
    cnt[x] -- ;
    if (!cnt[x]) res -- ;
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i]);
    for (int i = 0; i < m; i ++ )
    {
        char op[2];
        int a, b;
        scanf("%s%d%d", op, &a, &b);
        if (*op == 'Q') mq ++, q[mq] = {mq, a, b, mc};
        else c[ ++ mc] = {a, b};
    }

    len = cbrt((double)n * mc) + 1;
    sort(q + 1, q + mq + 1, cmp);
```

```
for (int i = 0, j = 1, t = 0, k = 1, res = 0; k <= mq; k
    ↪ ++ )
{
    int id = q[k].id, l = q[k].l, r = q[k].r, tm = q[k].t;
    while (i < r) add(w[ ++ i], res);
    while (i > r) del(w[i -- ], res);
    while (j < l) del(w[j ++ ], res);
    while (j > l) add(w[ -- j], res);
    while (t < tm)
    {
        t ++ ;
        if (c[t].p >= j && c[t].p <= i)
        {
            del(w[c[t].p], res); //修改一个数可以当作删掉原
            ↪ 数再加回去
            add(c[t].c, res);
        }
        swap(w[c[t].p], c[t].c);
    }
    while (t > tm)
    {
        if (c[t].p >= j && c[t].p <= i)
        {
            del(w[c[t].p], res);
            add(c[t].c, res);
        }
        swap(w[c[t].p], c[t].c);
        t -- ;
    }
    ans[id] = res;
}

for (int i = 1; i <= mq; i ++ ) printf("%d\n", ans[i]);
return 0;
}
```

6 数论

6.1 阶乘分解

```
//On 求阶乘的质因数分解
//首先把 n 以内质数全部打出来，然后枚举每个质数，计算次幂。
//比如 p^n，在 p^1 被 +1，在 p^2 被 +1，最终会加 n 次。比如
→ 2345678 2^1 贡献 4, 2^2 贡献 2 2^4 贡献 1
//复杂度 n/ln(n)*ln(n)=On
// 枚举倍数的时候记得开 ll

#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1000010;

int primes[N], cnt;
bool st[N];

void init(int n)
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] * i <= n; j++)
        {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int main()
{
    int n;
    cin >> n;
    init(n);

    for (int i = 0; i < cnt; i++) //枚举每个质因数
    {
```

```

int p = primes[i];
int s = 0;
//for (int j = n; j; j /= p) s += j / p;
for(long long i=p;i<=n;i*=p)s+=n/i;//计算 p 的倍数个数
    → +p 2 个数倍数 +.....
printf("%d %d\n", p, s);
}

return 0;
}

```

6.2 扩展中国剩余定理

```

//求解 n 个。 x=ai%mi 其中 mi 不一定两两互质
#include <cstdio>
#include <iostream>
using namespace std;
typedef long long ll;
int n;
ll exgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){
        x = 1, y = 0;
        return a;
    }

    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
ll inline mod(ll a, ll b){
    return ((a % b) + b) % b;
}
int main(){
    while(~scanf("%lld", &n))
    {
        ll a1, m1;
        bool f=1;
        scanf("%lld%lld", &a1, &m1);
        for(int i = 1; i < n; i++){
            ll a2, m2, k1, k2;
            scanf("%lld%lld", &a2, &m2);
            ll d = exgcd(a1, -a2, k1, k2);
            if((m2 - m1) % d)
            {

```

```

        f=0;
    }
    k1 = mod(k1 * (m2 - m1) / d, abs(a2 / d));
    m1 = k1 * a1 + m1;
    a1 = abs(a1 / d * a2);
}
if(f)printf("%lld\n", m1);
else puts("-1");
}
return 0;
}

```

6.3 min25 求 $1e10$ 以内素数和

```

//注意 n 在调用了 solve 以后就可能会改变。 $1e10$  的时候复杂度接近
//  $3e7$ .
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1000000 + 10;
int prime[N], id1[N], id2[N], flag[N], ncnt, m;
ll g[N], sum[N], a[N], T;
ll n;

int ID(ll x) {
    return x <= T ? id1[x] : id2[n / x];
}

ll calc(ll x) {
    return x * (x + 1) / 2 - 1;
}

ll init(ll x) {
    T = sqrt(x + 0.5);
    for (int i = 2; i <= T; i++) {
        if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt
            - 1] + i;
        for (int j = 1; j <= ncnt && i * prime[j] <= T; j++) {
            flag[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    for (ll l = 1; l <= x; l = x / (x / l) + 1) {
        a[++m] = x / l;
    }
}

```

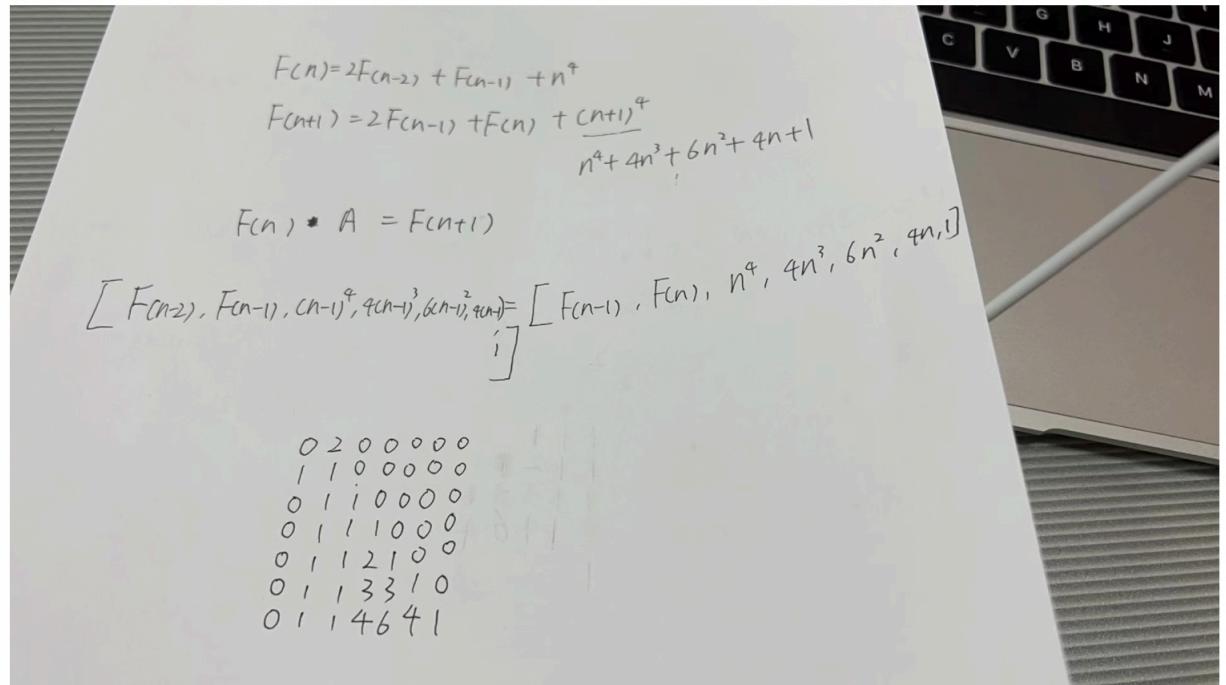
```
    if (a[m] <= T) id1[a[m]] = m; else id2[x / a[m]] = m;
    g[m] = calc(a[m]);
}
for (int i = 1; i <= ncnt; i++)
    for (int j = 1; j <= m && (ll) prime[i] * prime[i] <=
        → a[j]; j++)
        g[j] = g[j] - (ll) prime[i] * (g[ID(a[j] /
        → prime[i])] - sum[i - 1]));
}

ll solve(ll x) {
    if (x <= 1) return x;
    return n = x, init(n), g[ID(n)];
}

int main() {
    int t;
    cin>>t;
    while(t--)
    {
        memset(g, 0, sizeof(g));
        memset(a, 0, sizeof(a));
        memset(sum, 0, sizeof(sum));
        memset(prime, 0, sizeof(prime));
        memset(id1, 0, sizeof(id1));
        memset(id2, 0, sizeof(id2));
        memset(flag, 0, sizeof(flag));
        ncnt = m = 0;
        scanf("%lld", &n);
        cout<<solve(n)<<endl;
    }
}
```

6.4 矩阵快速幂

矩阵快速幂，注意一开始先弄个对角线为1的矩阵，1.先乘出 X^{n-1} 再乘初值矩阵（右边那个）注意次序！！！左乘和右乘答案是不一样的



```

/*
套路
题目给出 Fn=...
自己写出 Fn+1=...
然后有 Fn * A = Fn+1
注意 Fn 和 Fn+1 写出矩阵的形式必须是对应的 (n 对应 n-1 等等)
然后根据 Fn 的行列和 Fn+1 的行列，就可以知道 A 的行列。
*/
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 10;
const ll mod = 2147493647;

#define pid pair<int,double>
#define pii pair<int,int>

struct Mat {
    ll m[N][N];

```

```
Mat() //记得写，不然超时
{
    memset(m, 0, sizeof m);
}

void build() {
    for (int i = 0; i < N; i++) m[i][i] = 1;
}

} a, ans, org; //a 是常数矩阵, ans 是快速幂中存答案用的, org 是原
// 矩阵 (A 左边那个)。
int n = 7; //矩阵的行列数
Mat operator*(Mat &a, Mat &b) {
    Mat tmp;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            for (int k = 1; k <= n; k++) tmp.m[i][j] =
                (tmp.m[i][j] + a.m[i][k] * b.m[k][j]) % mod;
    return tmp;
}

void qp(ll b) {
    ans = Mat();
    ans.build();
    while (b) {
        if (b & 1) ans = ans * a;
        b >>= 1;
        a = a * a;
    }
}

int x[N][N] = { //构造的常数矩阵
    {0, 2, 0, 0, 0, 0, 0},
    {1, 1, 0, 0, 0, 0, 0},
    {0, 1, 1, 0, 0, 0, 0},
    {0, 1, 1, 1, 0, 0, 0},
    {0, 1, 1, 2, 1, 0, 0},
    {0, 1, 1, 3, 3, 1, 0},
    {0, 1, 1, 4, 6, 4, 1}
};

int main() {
```

```
int t;
cin >> t;
while (t--) {
    ll n, A, B;
    scanf("%lld%lld%lld", &n, &A, &B);
    if (n == 1)printf("%lld\n", A);
    else if (n == 2)printf("%lld\n", B);
    else {
        for (int i = 0; i < 7; i++)
            for (int j = 0; j < 7; j++)a.m[i + 1][j + 1] =
                x[i][j];
        qp(n - 2);
        org = Mat();
        org.m[1][1] = A, org.m[1][2] = B, org.m[1][3] = 2
        ↪ * 2 * 2 * 2, org.m[1][4] = 4 * 2 * 2 * 2,
        ↪ org.m[1][5] =
            6 * 2 * 2, org.m[1][6] = 4 * 2,
            ↪ org.m[1][7] = 1;
        org = org * ans;
        printf("%lld\n", org.m[1][2]);
    }
}

return 0;
}
```

6.5 筛质数

6.5.1 线性筛 (欧拉函数)

```
int phi[N], primes[N], cnt;
bool st[N];
void init(int n)
{
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])
        {
            primes[cnt ++ ] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; primes[j] * i <= n; j++)
            if (primes[j] * i != i)
                phi[i] *= primes[j];
    }
}
```

```
{  
    st[i * primes[j]] = true;  
    if (i % primes[j] == 0)  
    {  
        phi[i * primes[j]] = phi[i] * primes[j];  
        break;  
    }  
    phi[i * primes[j]] = phi[i] * (primes[j] - 1);  
}  
}  
}
```

6.6 组合数

6.6.1 杨辉三角

```
//2020/8/18  
#include<bits/stdc++.h>  
using namespace std;  
const int N=2e3+10;  
const int mod=1e9+7;  
typedef long long ll;  
ll c[N][N];  
void ini()  
{  
    for(int i=0;i<N;i++)  
    {  
        for(int j=0;j<=i;j++)  
        {  
            if(!j)c[i][j]=1;//避免 j==0 时越界, 同时 i==0 的越  
            //界的情况也处理了  
            else c[i][j]=(c[i-1][j]+c[i-1][j-1])%mod;  
        }  
    }  
}  
int main()  
{  
    int n;  
    cin>>n;  
    ini();  
    while(n--)  
    {  
        int a,b;  
        cin>>a>>b;
```

```
    printf("%d\n", c[a][b]);
}
return 0;
}
```

6.6.2 lucas 定理 ($a, b \leq 10^{18}$) P 是质数

```
//2020/8/28
/*
lucas 定理，用于求解组合数 a, b 非常大的时候，p 小于等于 1e5。把
→ a, b 处理到小于 1e5 然后用定义法 nlogn 解决
*/
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
typedef long long ll;
int n,p;
ll a,b;
ll qp(ll a,ll b)
{
    ll res=1;
    while(b)
    {
        if(b&1)res=res*a%p;
        a=a*a%p;
        b>>=1;
    }
    return res%p;
}
ll C(ll a,ll b)
{
    ll res=1;
    for(int i=a,j=b;j>=1;j--,i--)
    {
        res=res*i%p;
        res=res*qp(j,p-2)%p;
    }
    return res;
}
ll lucas(ll a,ll b)
{
    if(a<p&&b<p) return C(a,b);
    else return C(a%p,b%p)*lucas(a/p,b/p)%p;
}
```

```
int main()
{
    cin>>n;
    while(n--)
    {
        cin>>a>>b>>p;
        cout<<luca(a,b)<<endl;
    }
    return 0;
}
```

6.6.3 扩展 lucas ($a, b \leq 10^{18}$) P 不一定质数

```
//多组询问，给出 N,K,M，要求回答 C(N,K)%M,
// 1<=N<=10^18, 1<=K<=N, 2<=M<=10^6
using namespace std;
typedef long long LL;
const int maxn=1000005;

LL N,K; int M;
int pri[maxn],tot; bool ntp[maxn];

void get_pri(){
    ntp[0]=ntp[1]=1;
    for(int i=2;i<=1000000;i++){
        if(!ntp[i]) pri[++tot]=i;
        for(int j=1;j<=tot&&1ll*pri[j]*i<=1000000;j++){
            ntp[pri[j]*i]=1;
            if(i%pri[j]==0) break;
        }
    }
}
void exgcd(int a,int b,LL &d,LL &x,LL &y){
    if(!b) d=a,x=1,y=0;
    else exgcd(b,a%b,d,y,x),y-=a/b*x;
}
int inv(int a,int p){
    LL d,x,y; exgcd(a,p,d,x,y);
    return (x%p+p)%p;
}
int qkpow(int x,LL y,int p){
    int re=1,t=x;
    for(int i=0;(1ll<<i)<=y;i++){
        if((1ll<<i)&y) re=1ll*re*t%p;
```

```
t=1ll*t*t%p;
}
return re;
}
int J(LL n,int p,int pt,int mul){
if(n<=1) return 1;
int re=1;
if(n>=pt) re=qkpow(mul,n/pt,pt);
for(int i=1;i<=n%pt;i++)
    if(i%p) re=1ll*re*i%pt;
return 1ll*re*J(n/p,p,pt,mul)%pt;
}
int C(LL n,LL m,int p,int pt){
LL k=0,t;
for(t=n;t;t/=p) k+=t/p;
for(t=m;t;t/=p) k-=t/p;
for(t=n-m;t;t/=p) k-=t/p;
int pw=qkpow(p,k,pt); if(!pw) return 0;
int mul=1;
for(int i=1;i<pt;i++)
    if(i%p) mul=1ll*mul*i%pt;
int
    ↪ a=J(n,p,pt,mul),b=inv(J(m,p,pt,mul),pt),c=inv(J(n-m,p,pt,mul),pt);
return 1ll*pw*a%pt*b%pt*c%pt;
}
int exLucas(LL n,LL m,int p){
int re=0,t=p;
for(int i=1;i<=tot&&pri[i]<=t;i++){
    if(t%pri[i]) continue;
    int pt=1,c;
    while(t%pri[i]==0) pt*=pri[i],t/=pri[i];
    c=C(n,m,pri[i],pt);
    re=(re+1ll*p/pt*c%p*inv(p/pt,pt)%p)%p;
}
return re;
}
int main()
{
get_pri();
while(cin>>N>>K>>M) cout<<exLucas(N,K,M)<<'\n';
return 0;
}
```


6.7 卡特兰数的应用

-
- Handwritten notes on Catalan numbers:
- $2x \leq y + \alpha_1 x \leq y$
 - $2x - y \geq 0$
 - $2x - y \geq 0$
- 2021/5/8
- Catalan 数 详
- 卡特兰数的应用都可以归结到一种情况：操作次数相同，都为 N ，且在进行第一，问有多少中情况？结果就 $\text{Catalan}(N)$
- Catalan数的典型应用：**
- 由 n 个 $+1$ 和 n 个 -1 组成的排列中，满足前 n 项为正的有 $\text{Catalan}(n)$ 种。
 - 括号化问题。左括号和右括号各有 n 个时，有 $\text{Catalan}(n)$ 种。
 - 有 $n+1$ 个数连乘，乘法顺序有 $\text{Catalan}(n)$ 种。
 - n 个数按照特定顺序入栈，出栈顺序随意，有 $\text{Catalan}(n)$ 种。
 - 给定 N 个节点，能构成 $\text{Catalan}(N)$ 种形态。
 - n 个非叶节点的满二叉树的形态数为 $\text{Catalan}(n)$ 。

6.7.1 卡特兰数 (P 可能是合数, 阶乘分解法)

```
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 2000010;

int n, p;
int primes[N], cnt;
bool st[N];

void init(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!st[i]) primes[cnt++]=i;
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[primes[j]*i]=true;
            if(i%primes[j]==0) break;
        }
    }
}

int qmi(int a,int k)
{
    int res = 1;
    while(k)
    {
        if(k&1) res = (LL)res*a%p;
        a = (LL)a*a%p;
        k>>=1;
    }
    return res;
}

int get(int n,int p)
{
    int s = 0;
```

```
while(n)
{
    s+=n/p;
    n/=p;
}
return s;
}

int C(int a,int b)
{
    int res = 1;
    for(int i=0;i<cnt;i++)
    {
        int prime = primes[i];
        //  $C_{\{a\}}^{\{b\}} = a!/b!(a-b)!$ 
        int s = get(a,prime)-get(b,prime)-get(a-b,prime);
        res = (LL)res*qmi(prime,s)%p;
    }
    return res;
}

int main()
{
    scanf("%d%d", &n, &p);
    init(n * 2);

    cout << (C(n * 2, n) - C(n * 2, n - 1) + p) % p << endl;

    return 0;
}
```

6.8 约数

6.8.1 约数个数

约数个数

(int)类型内的某个数，约数个数最多为1500左右

$$\text{某个数 } N = p_1^{a_1} * p_2^{a_2} * \dots * p_k^{a_k}$$

$$\text{约数个数 } ans = (a_1 + 1) * (a_2 + 1) * \dots * (a_k + 1)$$

证明： $p_1^{a_1}$ 有约数 $p_1^0, p_1^1 \dots p_1^{a_1}$ 共 $(a_1 + 1)$ 个由乘法原理可得

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
typedef long long ll;
unordered_map<ll,ll>ms;
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        for(int i=2;i<=x/i;i++)//对每一个数质因数分解，把底数相
        → 同的指数加起来
        {
            while(x%i==0)
            {
                x/=i;
                ms[i]++;
            }
        }
        if(x>1)ms[x]++;
    }
    ll ans=1;
    for(auto m:ms)
    {
        ans=ans*(m.second+1)%mod;
    }
    cout<<ans;
}
```

```

    return 0;
}

```

6.8.2 约数的和

(3) 约数之和 $(p_1^0 + p_1^1 + \dots + p_1^{d_1}) \cdots (p_k^0 + p_k^1 + \dots + p_k^{d_k})$

```

const int mod=1e9+7;
typedef long long ll;
unordered_map<ll,ll>ms;
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        for(int i=2;i<=x/i;i++)//对每一个数质因数分解，把底数相
        → 同的指数加起来
        {
            while(x%i==0)
            {
                x/=i;
                ms[i]++;
            }
        }
        if(x>1)ms[x]++;
    }
    ll res=1;
    for(auto m:ms)
    {
        int p=m.first,a=m.second;
        ll t=1;
        while(a--)t=(t*p+1)%mod;//p+1,p^2+p+1
        res=res*t%mod;
    }
    cout<<res;
}

```


6.9 Exgcd 及同余性质

解 $ax+by=m$ 当 $\gcd(a,b)$ 整除 m 时有解。

求 $ax+by=d$ ($d=\gcd(a,b)$) 求 x 的最小整数解。令 $\text{mod}=(b/d)$ 解为 $(x\% \text{mod} + \text{mod}) \% \text{mod}$;

exgcd解出来的是 $ax+by=d$ 的一组特解，如果求 $ax+by=m$ 则 x 和 y 乘 $m/\gcd(a,b)$ 即可。求这个的最小整数解也要先 x 乘 $m/\gcd(a,b)$ 再令 $\text{mod}=(b/d)$ 解为 $(x\% \text{mod} + \text{mod}) \% \text{mod}$;

$$\gcd(a, b) = d$$

$$ax_0 + by_0 = d$$

$$x = x_0 + k\left(\frac{b}{d}\right)$$

$$y = y_0 - k\left(\frac{a}{d}\right)$$

特别的

$ax+by=0$ 的一般解

是 $x = rb/(a,b), y = -ra/(a,b)$

(a,b) 是 a,b 的最大公约数, r 是一任意整数.

1.同余是一种等价关系，即具有自反性、对称性和传递性。

2.同余有四个与等式相类似的性质:

(1)如果 a_1, b_1, a_2, b_2 都是整数，而 m 是正整数，则当 $a_1 \equiv b_1 \pmod{m}, a_2 \equiv b_2 \pmod{m}$ 都成立时，有 $a_1 \pm a_2 \equiv b_1 \pm b_2 \pmod{m}$.

(2)如果 a_1, b_1, a_2, b_2 都是整数，而 m 是正整数，则当 $a_1 \equiv b_1 \pmod{m}, a_2 \equiv b_2 \pmod{m}$ 都成立时，有 $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{m}$.

(3)如果 a, b, c 都是整数，而 m 是正整数，则当 $a + b \equiv c \pmod{m}$ 时，有 $a \equiv c - b \pmod{m}$

3.同余有五个与等式不相似的性质:

(1)如果 a, b 都是整数，而 k, m 是正整数，则当 $a \equiv b \pmod{m}$ 成立时，有 $ak \equiv bk \pmod{mk}$.

(2)如果 a, b 都是整数，而 d, m 是正整数， d 是 a, b 及 m 的任一公因数，则当 $a \equiv b \pmod{m}$ 成立时，有 $\frac{a}{d} \equiv \frac{b}{d} \pmod{\frac{m}{d}}$.

(3)如果 a, b 都是整数， d, m 是正整数，且 $d | m$,则当 $a \equiv b \pmod{m}$ 成立时，有 $a \equiv b \pmod{d}$

(5)如果 a, b 都是整数，而 d, m 是正整数，则当 $a \equiv b \pmod{m}$ 成立时，有 $(a, m) = (b, m)$ ，若 d 能整除 m 及 a, b 中的一个，则 d 必能整除 a, b 中的另一个. [1]

$ab \bmod c$ 若 b 和 c 互质，则对于不同的 a ，该式取值只有 $[0, c)$ ，并且这里面每个数字都能取到即 $0, 1, 2, \dots$ 。
 $c-1$
 $ab \bmod c$ 的解个数 $g = \gcd(b, c)$ 等价于 $a^*(b/g) \bmod (c/g)$ 的解个数 此时就符合上面的式子了，算出解的个数乘 g ，就是原式解的个数。似乎也叫完全剩余系

```
//2020/7/25
#include<bits/stdc++.h>
using namespace std;
int exgcd(int a,int b,int &x,int &y)
{
    if(!b)
    {
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,y,x); //带入得可得 ax+b (y-a/bx)=d, a 的系
    → 数 x 不变
    y-=a/b*x;
    return d;
}

int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int a,b,x,y;
        scanf("%d %d",&a,&b);
        exgcd(a,b,x,y);
        printf("%d %d\n",x,y);

    }
    return 0;
}
```

6.10 高斯消元 n^3

6.10.1 高斯消元解齐次线性方程组

```
#include<bits/stdc++.h>
using namespace std;
const int N=105;
double a[N][N];
```

```

const double eps=1e-6;
int n;
int gauss()
{
    int r,c;//row 行。 column 列
    for(int r=0,c=0;c<n;c++)//一列一列处理
    {
        int t=r;//找到当前列， 绝对值最大的一个数所在的行
        for(int i=r;i<n;i++)
        {
            if(fabs(a[i][c])>fabs(a[t][c]))t=i;
        }
        if (fabs(a[t][c]) < eps) continue;// 如果当前这一列的最
→ 大数都是 0， 那么所有数都是 0， 就没必要去算了， 因为它
→ 的约束方程， 可能在上面几行
        for(int i=c;i<=n;i++)swap(a[t][i],a[r][i]);//把当前这行
→ 换到最上面去 (r 行) 不是第 0 行
        for(int i=n;i>=c;i--)a[r][i]/=a[r][c];//从后往前
//交换好后， 这行到 r 行去了， 把 r 行第一个数变成 1。 从后
→ 往前刚好 (避免第一个数先变成 1)。 把这行每个数都除第一
→ 个数
        for(int i=r+1;i<n;i++)
        {
            /*
             *      c      j
             * r:      1 x x x x
             * r+1:    k s x t s
             */
            for(int j=n;j>=c;j--)//为了把 r 行下面每一行第一个
→ 数变成 1， 每一行从后往前，
            {
                a[i][j]-=a[r][j]*a[i][c];
            }
        }
        r++;
    }
    if(r<n)
    {
        /* r~n-1 的 = 左边都是 0
         * 1 x x s
         *   1 x s
         *     0 s
         */
    }
}

```

```

for(int i=r;i<n;i++)
{
    if(fabs(a[i][n]>eps))return 2;//无解。因为出现了
    ↪ 0==一个数
}
return 1;//出现 0==0 无限解
}
//有唯一解，有 n 个方程组 n 个未知数
/*
1 0 0 0 x
1 0 0 x
1 0 x
1 x
处理成这种效果
*/
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];
return 0;
}
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
        for(int j=0;j<n+1;j++)cin>>a[i][j];
    int t=gauss();
    if(t==0)for(int i=0;i<n;i++)printf("%.2lf\n",a[i][n]);
    else if(t==1)puts("Infinite group solutions");
    else puts("No solution");
    return 0;
}

```

6.10.2 高斯消元解异或线性方程组

```

//方程组中的系数和常数为 0 或 1，每个未知数的取值也为 0 或 1，等
↪ 号右边也是 0 或 1。
//多组输入切记清空 a 数组
//普通版本
//n 行 m 列的高斯消元板子，此时 m-r 就是自由元的个数（自由元可以
↪ 随意取值）
int n,m;//n 行 m 列 (m 个未知数)，如果等号右边要设置值的话则是
↪ m+1 列，默认 0.
int a[310][310];
int gauss()//返回矩阵的秩
{

```

```

int c, r;
for (c = 0, r = 0; c < n; c++)
{
    int t = r;
    for (int i = r; i < n; i++)
        if (a[i][c])
            t = i;

    if (!a[t][c]) continue;

    for (int i = c; i <= m; i++) swap(a[r][i],
        a[t][i]);
    for (int i = r + 1; i < n; i++)
        if (a[i][c])
            for (int j = m; j >= c; j--)
                a[i][j] ^= a[r][j];

    r++;
}
return r;
}

//此时 m-r 就是自由元的个数（自由元可以随意取值）
//bitset 优化版本，复杂度/64，下标从 0 开始
bitset<N>a[N];
ll gauss(int n,int m)
{
    int row = 0,col = 0,maxx;
    for(; col < m && row < n ;++ col){
        for(maxx = row ;maxx < n;++maxx){
            if(a[maxx][col])
                break;
        }
        if(a[maxx][col] == 0)
            continue;

        swap(a[maxx],a[row]);
        for(int i = row + 1;i < n; ++i){
            if(a[i][col])
                a[i] ^= a[row];
        }
        row++;
    }
    return m-row;//m-row 就是自由元个数
}

```

```
//解出每个变量答案
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 110;

int n,m;
int a[N][N];

int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c++)
    {
        int t = r;
        for (int i = r; i < n; i++)
            if (a[i][c])
                t = i;

        if (!a[t][c]) continue;

        for (int i = c; i <= m; i++) swap(a[r][i],
                                         a[t][i]);
        for (int i = r + 1; i < n; i++)
            if (a[i][c])
                for (int j = m; j >= c; j--)
                    a[i][j] -= a[r][j];

        r++;
    }

    if (r < n)
    {
        for (int i = r; i < n; i++)
            if (a[i][n])
                return 2;
        return 1;
    }
}
```

```

for (int i = n - 1; i >= 0; i -- )
    for (int j = i + 1; j < n; j ++ )
        a[i][n] ^= a[i][j] * a[j][n];

return 0;
}

int main()
{
    cin >> n;
    m=n;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < m + 1; j ++ )
            cin >> a[i][j];

    int t = gauss();

    if (t == 0)
    {
        for (int i = 0; i < n; i ++ ) cout << a[i][n] <<
        endl;
    }
    else if (t == 1) puts("Multiple sets of solutions");
    else puts("No solution");

    return 0;
}

```

6.10.3 高斯消元求行列式值（取模,double）

```

/*
 * 取模
算出来的都是非负的
*/
typedef long long ll;
ll A[110][110];

const int mod=1e9+7;

ll quick_pow(ll a, ll n, ll mod) {
    ll ans = 1;
    while(n) {
        if(n & 1) ans = ans * a % mod;

```

```
a = a * a % mod;
n >= 1;
}
return ans;
}

ll inv(ll a) {
    return quick_pow(a, mod - 2, mod);
}

ll gauss(int n){
    ll ans = 1;
    for(int i = 1; i <= n; i++){
        for(int j = i; j <= n; j++) {
            if(A[j][i]){
                for(int k = i; k <= n; k++) swap(A[i][k],
                                              A[j][k]);
                if(i != j) ans = -ans;
                break;
            }
        }
        if(!A[i][i]) return 0;
        for(ll j = i + 1, iv = inv(A[i][i]); j <= n; j++) {
            ll t = A[j][i] * iv % mod;
            for(int k = i; k <= n; k++)
                A[j][k] = (A[j][k] - t * A[i][k] % mod + mod)
                           % mod;
        }
        ans = (ans * A[i][i] % mod + mod) % mod;
    }
    return ans;
}
//double
int Gauss1()
{
    for(int i=1;i<=n;i++){
        int p=i;
        while(!f[p][i] and p<=n)p++;
        if(p>=n+1) return 0;
        if(p!=i)swap(f[i],f[p]);
        for(int j=i+1;j<=n;j++)
        {
            double tmp=f[j][i]/f[i][i];
            for(int k=i;k<=n;k++)

```

```

        f[j][k] -= f[i][k] * tmp;
    }
}
double ans=1;
for(int i=1;i<=n;i++)
    ans *= f[i][i];
return (int)(ans+0.5);
}

```

6.11 容斥原理

```

//容斥原理 复杂度 O(m*2^m)
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int p[20];
int main()
{
    ll n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++)cin>>p[i];
    ll res=0;
    for(int i=1;i<1<<m;i++) //枚举每一种情况，选一个，选两个.....
    {
        ll t=1,cnt=0;
        for(int j=0;j<m;j++)
        {
            if(i>>j&1)
            {
                if(p[j]*t>n) //如果选的数乘起来比 n 大
                {
                    t=-1;
                    break;
                }
                t*=p[j];
                cnt++;
            }
        }
        if(t!=-1)
        {
            if(cnt%2)res+=n/t;//奇数时是加，偶数是-
            else res-=n/t;
        }
    }
}

```

```
    cout<<res;
    return 0;
}

6.12 FFT 与 NTT

//FFT
//
//nlogn 求两个多项式乘（卷积）
//数组开 4 倍
//使用分治时，将其他无关的位置置为 0，记得循环到 limit
//p 一般取 998244353, 1004535809, 469762049
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

using namespace std;

const int N = 300010;
const double PI = acos(-1);

int n, m;
struct Complex
{
    double x, y;
    Complex operator+ (const Complex& t) const
    {
        return {x + t.x, y + t.y};
    }
    Complex operator- (const Complex& t) const
    {
        return {x - t.x, y - t.y};
    }
    Complex operator* (const Complex& t) const
    {
        return {x * t.x - y * t.y, x * t.y + y * t.x};
    }
}a[N], b[N];
int rev[N], bit, tot;

void fft(Complex a[], int inv)
{
    for (int i = 0; i < tot; i++)

```

```
if (i < rev[i])
    swap(a[i], a[rev[i]]);
for (int mid = 1; mid < tot; mid <= 1)
{
    auto w1 = Complex({cos(PI / mid), inv * sin(PI /
        → mid)});
    for (int i = 0; i < tot; i += mid * 2)
    {
        auto wk = Complex({1, 0});
        for (int j = 0; j < mid; j ++, wk = wk * w1)
        {
            auto x = a[i + j], y = wk * a[i + j +
                → mid];
            a[i + j] = x + y, a[i + j + mid] = x - y;
        }
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n; i ++ ) scanf("%lf", &a[i].x);
    for (int i = 0; i <= m; i ++ ) scanf("%lf", &b[i].x);
    while ((1 << bit) < n + m + 1) bit++;
    tot = 1 << bit;
    for (int i = 0; i < tot; i ++ )
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    fft(a, 1), fft(b, 1);
    for (int i = 0; i < tot; i ++ ) a[i] = a[i] * b[i];
    fft(a, -1);
    for (int i = 0; i <= n + m; i ++ )
        printf("%d ", (int)(a[i].x / tot + 0.5)); //+0.5 因为是
        → 浮点, 怕误差。

    return 0;
}
//NTT
#include<bits/stdc++.h>
using namespace std;
#define getchar() (p1 == p2 && (p2 = (p1 = buf) + fread(buf,
    → 1, 1<<21, stdin), p1 == p2) ? EOF : *p1++)
#define swap(x, y) x ^= y, y ^= x, x ^= y
#define LL long long
```

```

const int MAXN = 3 * 1e6 + 10, P = 998244353, G = 3, Gi =
    ↵ 332748118;
char buf[1 << 21], *p1 = buf, *p2 = buf;

inline int read()
{
    char c = getchar();
    int x = 0, f = 1;
    while (c < '0' || c > '9')
    {
        if (c == '-') f = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9') x = x * 10 + c - '0', c =
        ↵ getchar();
    return x * f;
}

int N, M, limit = 1, L, r[MAXN];
LL a[MAXN], b[MAXN];

inline LL fastpow(LL a, LL k)
{
    LL base = 1;
    while (k)
    {
        if (k & 1) base = (base * a) % P;
        a = (a * a) % P;
        k >= 1;
    }
    return base % P;
}

inline void NTT(LL *A, int type)
{
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <= 1)
    {
        LL Wn = fastpow(type == 1 ? G : Gi, (P - 1) / (mid
            ↵ << 1));
        for (int j = 0; j < limit; j += (mid << 1))
        {
            LL w = 1;

```

```

        for (int k = 0; k < mid; k++, w = (w * Wn) %
            ↪ P)
        {
            int x = A[j + k], y = w * A[j + k + mid] %
                ↪ P;
            A[j + k] = (x + y) % P,
            A[j + k + mid] = (x - y + P) % P;
        }
    }
}

int main()
{
    N = read();
    M = read();
    for (int i = 0; i <= N; i++) a[i] = (read() + P) % P;
    for (int i = 0; i <= M; i++) b[i] = (read() + P) % P;
    while (limit <= N + M) limit <= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) |
        ((i & 1) << (L - 1));
    NTT(a, 1);
    NTT(b, 1);
    for (int i = 0; i < limit; i++) a[i] = (a[i] * b[i]) % P;
    NTT(a, -1);
    LL inv = fastpow(limit, P - 2);
    for (int i = 0; i <= N + M; i++)
        printf("%d ", (a[i] * inv) % P);
    return 0;
}

// 分治 fft, n 个多项式相乘，每个多项式都是  $1+aix$  这种
#include<bits/stdc++.h>
using namespace std;
#define getchar() (p1 == p2 && (p2 = (p1 = buf) + fread(buf,
    ↪ 1, 1<<21, stdin), p1 == p2) ? EOF : *p1++)
#define swap(x, y) x ^= y, y ^= x, x ^= y
#define ll long long
const int MAXN = 3 * 1e6 + 10, P = 998244353, G = 3, Gi =
    ↪ 332748118;
char buf[1 << 21], *p1 = buf, *p2 = buf;

inline int read()
{

```

```

char c = getchar();
int x = 0, f = 1;
while (c < '0' || c > '9')
{
    if (c == '-') f = -1;
    c = getchar();
}
while (c >= '0' && c <= '9') x = x * 10 + c - '0', c =
→ getchar();
return x * f;
}

int N, M, limit = 1, L, r[MAXN];
ll a[MAXN], b[MAXN];

inline ll fastpow(ll a, ll k)
{
    ll base = 1;
    while (k)
    {
        if (k & 1) base = (base * a) % P;
        a = (a * a) % P;
        k >>= 1;
    }
    return base % P;
}

void ini(int len)
{
    limit = 1, L = 0;
    while (limit <= len) limit <= 1, L++;
    for (int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) |
→ ((i & 1) << (L - 1));
}

inline void NTT(ll *A, int type)
{
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <= 1)
    {
        ll Wn = fastpow(type == 1 ? G : Gi, (P - 1) / (mid
→ << 1));
        for (int j = 0; j < limit; j += (mid << 1))

```

```

{
    ll w = 1;
    for (int k = 0; k < mid; k++, w = (w * Wn) %
        → P)
    {
        int x = A[j + k], y = w * A[j + k + mid] %
            → P;
        A[j + k] = (x + y) % P,
        A[j + k + mid] = (x - y + P) % P;
    }
}
}

ll w[MAXN], g[MAXN];

void solve(int l, int r, vector<ll> &f)
{
    if (l == r)
    {
        f[0]=1;
        f[1]=a[l];
        return;
    }
    int mid = l + r >> 1;
    int len1=mid-l+1, len2=r-mid;
    vector<ll> f1(len1+5), f2(len2+5);
    solve(l,mid,f1);
    solve(mid+1,r,f2);
    for(int i=0;i<=len1;i++) w[i]=f1[i];
    for(int i=0;i<=len2;i++) g[i]=f2[i];
    int len=len1+len2;
    ini(len);
    for(int i=len1+1;i<limit;i++) w[i]=0;
    for(int i=len2+1;i<limit;i++) g[i]=0;
    NTT(w, 1);
    NTT(g, 1);
    for (int i = 0; i < limit; i++) w[i] = (w[i] * g[i]) % P;
    NTT(w, -1);
    ll inv = fastpow(limit, P - 2);
    for (int i=0;i<=len;i++) f[i] = w[i] * inv % P;
}
ll in[MAXN];
int main()

```

```

{
    int t;
    t = read();
    in[0]=1;
    for(ll i=1;i<MAXN;i++) in[i]=in[i-1]*i%P;
    while(t--)
    {
        N = read();
        for (int i = 1; i <= N ; i++) a[i] = (read() + P) % P;
        vector<ll>ans(N+5);
        solve(1,N,ans);
        ll res=0;
        for(int i=1;i<=N;i++)
        {
            //cout<<ans[i]<<endl;
            res=(res+ans[i]*in[i]%P*in[N-i]%P)%P;
        }
        printf("%lld\n",res*fastpow(in[N],P-2)%P);
    }
    return 0;
}

```

6.13 BSGS (普通和扩展)

```

//普通 bsgs
//复杂度  $O(\sqrt{p})$ 。求  $a^x \equiv b \pmod{p}$  的最小非负整数解  $x$ 。要求  $a$  和  $p$ 
//互质。扩展的 bsgs 则不要求互质，可以直接用扩展的
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <unordered_map>

using namespace std;

typedef long long LL;

int bsgs(int a, int b, int p)
{
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i++)
    {

```

```
        hash[j] = i;
        j = (LL)j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i++) ak = (LL)ak * a % p;

    for (int i = 1, j = ak; i <= k; i++)
    {
        if (hash.count(j)) return (LL)i * k - hash[j];
        j = (LL)j * ak % p;
    }
    return -1;
}

int main()
{
    int a, p, b;
    while (cin >> a >> p >> b, a || p || b)
    {
        int res = bsqs(a, b, p);
        if (res == -1) puts("No Solution");
        else cout << res << endl;
    }
    return 0;
}

//扩展 bsqs
//给定整数 a,p,b。
//求满足  $a^x \equiv b \pmod{p}$  的最小非负整数 x。
//1 a,p 2~31-1,
//0 b 2~31-1
//如果有 x 满足该要求，输出最小的非负整数 x，否则输出 No
// Solution。
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <unordered_map>

using namespace std;

typedef long long LL;
const int INF = 1e8;
```

```

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

int bsgs(int a, int b, int p)
{
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i++)
    {
        hash[j] = i;
        j = (LL)j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i++) ak = (LL)ak * a % p;
    for (int i = 1, j = ak; i <= k; i++)
    {
        if (hash.count(j)) return i * k - hash[j];
        j = (LL)j * ak % p;
    }
    return -INF;
}

int exbsgs(int a, int b, int p)
{
    b = (b % p + p) % p;
    if (1 % p == b % p) return 0;
    int x, y;
    int d = exgcd(a, p, x, y);
    if (d > 1)
    {
        if (b % d) return -INF;
        exgcd(a / d, p / d, x, y);
        return exbsgs(a, (LL)b / d * x % (p / d), p / d) + 1;
    }
}

```

```
    return bsgs(a, b, p);
}

int main()
{
    int a, p, b;
    while (cin >> a >> p >> b, a || p || b)
    {
        int res = exbsgs(a, b, p);
        if (res < 0) puts("No Solution");
        else cout << res << endl;
    }
    return 0;
}
```

6.14 生成函数

$$f(x) = (1 - x)^{-k} = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n$$

指类型母函数适用于解决多重集排列问题。

这里给出指类型生成函数常见的化简公式：

$$\sum_{i=0}^{\infty} \frac{x^i}{i!} = e^x;$$

$$1 - \frac{x}{1!} + \frac{x^2}{2!} - \dots + (-1)^n \frac{x^n}{n!} + \dots = e^{-x}.$$

所以只取偶数项的指类型生成函数为 $\frac{e^x + e^{-x}}{2}$, 只取奇数项的指类型生成函数为 $\frac{e^x - e^{-x}}{2}$ 。

①数列 r^n 的指类型生成函数为 $e^{rx} = \sum_{i=0}^{\infty} \frac{(rx)^i}{i!}$ 。

②数列 {0, 1, 0, -1, 0, 1, 0, -1, ...} 的指类型生成函数为 $\sin(x)$ 。

③数列 {1, 0, -1, 0, 1, 0, -1, 0, ...} 的指类型生成函数为 $\cos(x)$ 。

$$e^{4x} = 1 + (4x)/1! + (4x)^2/2! + (4x)^3/3! + \dots + (4x)^n/n!;$$

$$e^{2x} = 1 + (2x)/1! + (2x)^2/2! + (2x)^3/3! + \dots + (2x)^n/n!;$$

指类型生成函数（多重集，即考虑排列）的方案是 $x^n/n!$ 前的系数，分母必须是对应的阶乘。系数就是答案

x当成-1~1之间的数，所以比如 $1 + x + x^2 + \dots + x^n = (1 - x^{n+1})/(1 - x) = 1/(1 - x)$ 因为次幂趋近于0

6.15 线性基

- 1.原序列里面的任意一个数都可以由线性基里面的一些数异或或得到
- 2.线性基里面的任意一些数异或起来都不能得到 0 (P中任意数都不能被P中其他数异或出来)
- 3.线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的
- 4.线性基的最高位不重复

能插入x，说明线性基里面不能异或出x，否则说明能

线性基的大小跟数的位数有关，比如这题每个数最多63位，线性基的大小就小于等于63位

```
//给定 n 个整数 (可能重复)，现在需要从中挑选任意个整数，使得选出整
→ 数的异或和最大。
//请问，这个异或和的最大可能值是多少。
//从高位到低位枚举，如果异或上线性基的数结果变大就异或他，否则不异
→ 或他
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
typedef long long ll;
ll d[N];
void add(ll x)
{
    for(int i=62;i>=0;i--)
    {
        if(x&(1ll<<i)) //注意，如果 i 大于 31，前面的 1 的后面一
        → 定要加 ll
        {
            if(d[i])x^=d[i];
            else
            {
                d[i]=x;
                break; //插入成功就退出
            }
        }
    }
}
int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
```

```

    ll x;
    scanf("%lld", &x);
    add(x);
}
ll ans=0;
for(int i=62; i>=0; i--) ans=max(ans, ans^d[i]); //注意从大到小
cout<<ans;
return 0;
}

```

6.16 欧拉降幂

内容

在数论中，**欧拉定理**（也称**费马-欧拉定理**）是一个关于同余的性质。欧拉定理表明，若n,a为正整数，且n,a互质，则：

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad \text{https://blog.csdn.net/qq_37632935}$$

由此可以得到降幂公式

$$a^b \equiv \begin{cases} a^{b\% \phi(p)} & \gcd(a, p) = 1 \\ a^b & \gcd(a, p) \neq 1, b < \phi(p) \\ a^{b\% \phi(p)+\phi(p)} & \gcd(a, p) \neq 1, b \geq \phi(p) \end{cases} \pmod{p}$$

https://blog.csdn.net/qq_37632935

第一个要求a和p互质，第二个和第三个是广义欧拉降幂，不要求a和p互质，但要求b和 $\phi(p)$ 的大小关系。

```

#include <bits/stdc++.h>
#define ll __int64
#define mod 10000000007
using namespace std;
char a[1000006];
ll x, z;
ll quickpow(ll x, ll y, ll z)
{
    ll ans=1;
    while(y)
    {
        if(y&1)
            ans=ans*x%z;
        x=x*x%z;
        y>>=1;
    }
}

```

```
    return ans;
}
ll phi(ll n)
{
    ll i, rea=n;
    for(i=2; i*i<=n; i++)
    {
        if(n%i==0)
        {
            rea=re-a/i;
            while(n%i==0)
                n/=i;
        }
    }
    if(n>1)
        rea=re-a/n;
    return rea;
}
int main()
{
    while(scanf("%lld %s %lld", &x, a, &z) !=EOF)
    {
        ll len=strlen(a);
        ll p=phi(z);
        ll ans=0;
        for(ll i=0; i<len; i++)
            ans=(ans*10+a[i]-'0')%p;
        ans+=p;
        printf("%lld\n", quickpow(x, ans, z));
    }
    return 0;
}
```

6.17 康托展开求排列的字典序排名



首发于
算法学习笔记

算法学习笔记(56): 康托展开



Pecco
可能算ACMer

关注

致知计划 科学季收录 >

149 人赞同了该文章

康托展开用于求一个排列在所有 $1 \sim n$ 的排列间的字典序排名。

其实康托展开的原理很简单。设有排列 $p = a_1 a_2 \dots a_n$ ，那么对任意字典序比 p 小的排列，一定存在 i ，使得其前 $i-1$ ($1 \leq i < n$) 位与 p 对应位相同，第 i 位比 p_i 小，后续位随意。于是对于任意 i ，满足条件的排列数就是从后 $n-i+1$ 位中选一个比 a_i 小的数、并将剩下 $n-i$ 个数任意排列的方案数，即为 $A_i \cdot (n-i)!$ (A_i 表示 a_i 后面比 a_i 小的数的个数)。

遍历 i 即得总方案数 $\sum_{i=1}^{n-1} A_i \cdot (n-i)!$ ，再加1即为排名。

例如若 $p = 4 1 3 2$ ，可以求得 $A = [3, 0, 1, 0]$ ，第一位比 p_1 小的排列数为 $3 \times 3! = 18$ ；第一位与 p_1 相等，第二位比 p_2 小的排列没有；第一、二位分别等于 p_1 、 p_2 ，第三位比 p_3 小的排列数为 $1 \times 1! = 1$ 。所以 p 的排名是 $18 + 1 + 1 = 20$ 。

那么问题就转换成了如何求 A_i ，显然可以 $O(n^2)$ 地求：

```
ll fact[MAXN] = {1}, P[MAXN], A[MAXN]; // fact需要在外部初始化
ll cantor(int P[], int n) // 这里传入的P是1-index数组
{
    ll ans = 1;
    for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
            if (P[j] < P[i])
                A[i]++;
    for (int i = 1; i < n; i++)
        ans += A[i] * fact[n - i];
    return ans;
}
```

这个复杂度其实很够了，毕竟 $21!$ 就已经超过 long long 范围了，很多时候康托展开也就是在这个框架下进行 加里笛卡 就重压上高精度或者重压取模了 当然 优化也不难 很容易想到用树

▲ 赞同 149 ▾ 4 条评论 分享 喜欢 收藏 申请转载 ...



```

ll query(ll x)
{
    ll ans = 0;
    for (int i = x; i >= 1; i -= lowbit(i))
        ans += tree[i];
    return ans;
}
void update(ll x, ll d)
{
    for (int i = x; i < MAXN; i += lowbit(i))
        tree[i] += d;
}
ll cantor(int P[], int n)
{
    ll ans = 1;
    for (int i = n; i >= 1; i--)
    {
        A[i] = query(P[i]);
        update(P[i], 1);
    }
    for (int i = 1; i < n; i++)
        ans = (ans + A[i] * fact[n - i]) % MOD;
    return ans;
}

```

与康托展开相对应的是逆康托展开，即求指定排名的排列。原理也很简单，注意到

$$n! = n(n-1)! = (n-1) \cdot (n-1)! + (n-1)! , \text{ 继续展开得 } n! = \sum_{i=1}^{n-1} i \cdot i! , \text{ 而}$$

$$A_i \leq n-i , \text{ 所以 } \sum_{i=j}^{n-1} A_i \cdot (n-i)! \leq \sum_{i=j}^{n-1} (n-i) \cdot (n-i)! = \sum_{i=1}^{n-j} i \cdot i! = (n-j+1)! .$$

这意味着对于这个和式而言，每一项的 $(n-i)!$ 都比后面所有项的总和还大。于是可以用类似进制转换的方法，不断地模、除，来得到 A 的每一项。

得到 A 后，我们已经知道每一项之后有多少个比该项小的数，也就是说 p_i 就是剩余未用的数中第 $A_i + 1$ 小的。可以朴素地实现：



首发于
算法学习笔记

```
{
    x--;
    vector<int> rest(n, 0);
    iota(rest.begin(), rest.end(), 1); // 将rest初始化为1, 2, ..., n
    for (int i = 1; i <= n; ++i)
    {
        A[i] = x / fact[n - i];
        x %= fact[n - i];
    }
    for (int i = 1; i <= n; ++i)
    {
        P[i] = rest[A[i]];
        rest.erase(lower_bound(rest.begin(), rest.end(), P[i]));
    }
}
```

当然，也可以使用各种平衡树来优化到 $O(n \log n)$ ：

```
ll fact[MAXN] = {1}, P[MAXN], A[MAXN]; // fact需要在外部初始化
void decanter(ll x, int n) // x为排列的排名, n为排列的长度
{
    x--;
    for (int i = 1; i <= n; ++i)
        insert(i);
    for (int i = 1; i <= n; ++i)
    {
        A[i] = x / fact[n - i];
        x %= fact[n - i];
    }
    for (int i = 1; i <= n; ++i)
    {
        P[i] = kth(A[i] + 1);
        remove(P[i]);
    }
}
```

其中 `insert`、`kth`、`remove` 函数分别表示插入、查找第k小、删除元素。（参加二叉搜索树的笔记）

7 计算几何

7.1 凸包

```
/* 凸包 这里求的是围住的长度，有三点共线会忽略掉里面的一个点。 */
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;
const int N = 10010;

int n;
PDD q[N];
int stk[N];
bool used[N];

double get_dist(PDD a, PDD b)
{
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

PDD operator-(PDD a, PDD b)
{
    return {a.x - b.x, a.y - b.y};
}

double cross(PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

double area(PDD a, PDD b, PDD c)
{
    return cross(b - a, c - a);
}
```

```
double andrew()
{
    sort(q, q + n);
    int top = 0;
    for (int i = 0; i < n; i++)
    {
        while (top >= 2 && area(q[stk[top - 1]], q[stk[top]], 
        ↪ q[i]) <= 0)
        {
            // 凸包边界上的点即使被从栈中删掉，也不能删掉 used
            ↪ 上的标记
            if (area(q[stk[top - 1]], q[stk[top]], q[i]) < 0)
                used[stk[top --]] = false;
            else top--;
        }
        stk[++top] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; i--)
    {
        if (used[i]) continue;
        while (top >= 2 && area(q[stk[top - 1]], q[stk[top]], 
        ↪ q[i]) <= 0)
            top--;
        stk[++top] = i;
    }

    double res = 0;
    for (int i = 2; i <= top; i++)
        res += get_dist(q[stk[i - 1]], q[stk[i]]);
    return res;
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &q[i].x,
    ↪ &q[i].y);
    double res = andrew();
    printf("%.2lf\n", res);
```

```
    return 0;
}
```

7.2 半平面交

```
/* 二维平面  
* 逆时针给出 n 个凸多边形的顶点坐标，求它们交的面积。
```

例如 $n=2$ 时，两个凸多边形如下图：

```
/*
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double,double> PDD;
const int N = 510;
const double eps = 1e-8;

int cnt;
struct Line
{
    PDD st, ed;
}line[N];
PDD pg[N], ans[N];
int q[N];

int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}
```

```
}

double get_angle(const Line& a)
{
    return atan2(a.ed.y - a.st.y, a.ed.x - a.st.x);
}

PDD operator-(PDD a, PDD b)
{
    return {a.x - b.x, a.y - b.y};
}

double cross(PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

double area(PDD a, PDD b, PDD c)
{
    return cross(b - a, c - a);
}

bool cmp(const Line& a, const Line& b)
{
    double A = get_angle(a), B = get_angle(b);
    if (!dcmp(A, B)) return area(a.st, a.ed, b.ed) < 0;
    return A < B;
}

PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w)
{
    auto u = p - q;
    double t = cross(w, u) / cross(v, w);
    return {p.x + v.x * t, p.y + v.y * t};
}

PDD get_line_intersection(Line a, Line b)
{
    return get_line_intersection(a.st, a.ed - a.st, b.st, b.ed
        - b.st);
}

// bc 的交点是否在 a 的右侧
```

```
bool on_right(Line& a, Line& b, Line& c)
{
    auto o = get_line_intersection(b, c);
    return sign(area(a.st, a.ed, o)) <= 0;
}

double half_plane_intersection()
{
    sort(line, line + cnt, cmp);
    int hh = 0, tt = -1;
    for (int i = 0; i < cnt; i++)
    {
        if (i && !dcmp(get_angle(line[i]), get_angle(line[i - 1]))) continue;
        while (hh + 1 <= tt && on_right(line[i], line[q[tt - 1]], line[q[tt]])) tt--;
        while (hh + 1 <= tt && on_right(line[i], line[q[hh]], line[q[hh + 1]])) hh++;
        q[++tt] = i;
    }
    while (hh + 1 <= tt && on_right(line[q[hh]], line[q[tt - 1]], line[q[tt]])) tt--;
    while (hh + 1 <= tt && on_right(line[q[tt]], line[q[hh]], line[q[hh + 1]])) hh++;
    q[++tt] = q[hh];
    int k = 0;
    for (int i = hh; i < tt; i++)
        ans[k++] = get_line_intersection(line[q[i]], line[q[i + 1]]);
    double res = 0;
    for (int i = 1; i + 1 < k; i++)
        res += area(ans[0], ans[i], ans[i + 1]);
    return res / 2;
}

int main()
{
    int n, m;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%d", &m);
```

```
    for (int i = 0; i < m; i++) scanf("%lf%lf",
        → &pg[i].x, &pg[i].y);
    for (int i = 0; i < m; i++)
        line[cnt++] = {pg[i], pg[(i + 1) % m]};
}
double res = half_plane_intersection();
printf("%.3lf\n", res);

return 0;
}
```

7.3 最小圆覆盖

/* 最小圆覆盖 最小的能够包含所有点的圆。
输出圆的半径和圆心的坐标。 */

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;
const int N = 100010;
const double eps = 1e-12;
const double PI = acos(-1);

int n;
PDD q[N];
struct Circle
{
    PDD p;
    double r;
};

int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
```

```
}

int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

PDD operator- (PDD a, PDD b)
{
    return {a.x - b.x, a.y - b.y};
}

PDD operator+ (PDD a, PDD b)
{
    return {a.x + b.x, a.y + b.y};
}

PDD operator* (PDD a, double t)
{
    return {a.x * t, a.y * t};
}

PDD operator/ (PDD a, double t)
{
    return {a.x / t, a.y / t};
}

double operator* (PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

PDD rotate(PDD a, double b)
{
    return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y *
           cos(b)};
}

double get_dist(PDD a, PDD b)
{
    double dx = a.x - b.x;
    double dy = a.y - b.y;
```

```

    return sqrt(dx * dx + dy * dy);
}

PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w)
{
    auto u = p - q;
    double t = w * u / (v * w);
    return p + v * t;
}

pair<PDD, PDD> get_line(PDD a, PDD b)
{
    return {(a + b) / 2, rotate(b - a, PI / 2)};
}

Circle get_circle(PDD a, PDD b, PDD c)
{
    auto u = get_line(a, b), v = get_line(a, c);
    auto p = get_line_intersection(u.x, u.y, v.x, v.y);
    return {p, get_dist(p, a)};
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &q[i].x,
        &q[i].y);
    random_shuffle(q, q + n);

    Circle c({q[0], 0});
    for (int i = 1; i < n; i++)
        if (dcmp(c.r, get_dist(c.p, q[i])) < 0)
    {
        c = {q[i], 0};
        for (int j = 0; j < i; j++)
            if (dcmp(c.r, get_dist(c.p, q[j])) < 0)
        {
            c = {(q[i] + q[j]) / 2, get_dist(q[i],
                q[j]) / 2};
            for (int k = 0; k < j; k++)
                if (dcmp(c.r, get_dist(c.p, q[k])) <
                    0)
                    c = get_circle(q[i], q[j], q[k]);
        }
    }
}

```

```

    }

    printf("%.10lf\n", c.r);
    printf("%.10lf %.10lf\n", c.p.x, c.p.y);
    return 0;
}

```

7.4 三维凸包

```

/*
三维凸包
1. 三维向量表示  $(x, y, z)$ 
2. 向量加减法、数乘运算，与二维相同
3. 模长  $|A| = \sqrt{x * x + y * y + z * z}$ 
4. 点积
   (1) 几何意义： $A \cdot B = |A| * |B| * \cos(C)$ 
   (2) 代数求解： $(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = (x_1x_2, y_1y_2,$ 
    $\rightarrow z_1z_2);$ 
5. 叉积
   (1) 几何意义： $AxB = |A| * |B| * \sin(C)$ , 方向：右手定则
   (2) 代数求解： $AxB = (y_1z_2 - z_1y_2, z_1x_2 - x_1z_2, x_1y_2 - y_1x_1)$ 
6. 如何求平面法向量
任取平面上两个不共线的向量  $A, B : AxB$ 
7. 判断点  $D$  是否在平面里
任取平面上两个不共线的向量  $A, B$ : 先求法向量  $C = AxB$ , 然后求平面上
 $\rightarrow$  任意一点到  $D$  的向量  $E$  与  $C$  的点积, 判断点积是否为  $0$ 。
8. 求点  $D$  到平面的距离
任取平面上两个不共线的向量  $A, B$ : 先求法向量  $C = AxB$ 。然后求平面上
 $\rightarrow$  任意一点到  $D$  的向量  $E$  在  $C$  上的投影长度即可。即： $E \cdot C / |C|$ 
9. 多面体欧拉定理
顶点数 - 棱长数 + 表面数 = 2
10. 三维凸包
*/
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

using namespace std;

const int N = 210;
const double eps = 1e-10;

int n, m;

```

```
bool g[N][N];

double rand_eps()
{
    return ((double)rand() / RAND_MAX - 0.5) * eps;
}

struct Point
{
    double x, y, z;
    void shake()
    {
        x += rand_eps(), y += rand_eps(), z += rand_eps();
    }
    Point operator- (Point t)
    {
        return {x - t.x, y - t.y, z - t.z};
    }
    double operator& (Point t)
    {
        return x * t.x + y * t.y + z * t.z;
    }
    Point operator* (Point t)
    {
        return {y * t.z - t.y * z, z * t.x - x * t.z, x * t.y
            - y * t.x};
    }
    double len()
    {
        return sqrt(x * x + y * y + z * z);
    }
}q[N];
struct Plane
{
    int v[3];
    Point norm() // 法向量
    {
        return (q[v[1]] - q[v[0]]) * (q[v[2]] - q[v[0]]);
    }
    double area() // 求面积
    {
        return norm().len() / 2;
    }
    bool above(Point a)
```

```

{
    return ((a - q[v[0]])) & norm()) >= 0;
}
}plane[N], np[N];

void get_convex_3d()
{
    plane[m ++ ] = {0, 1, 2};
    plane[m ++ ] = {2, 1, 0};
    for (int i = 3; i < n; i ++ )
    {
        int cnt = 0;
        for (int j = 0; j < m; j ++ )
        {
            bool t = plane[j].above(q[i]);
            if (!t) np[cnt ++ ] = plane[j];
            for (int k = 0; k < 3; k ++ )
                g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
        }
        for (int j = 0; j < m; j ++ )
            for (int k = 0; k < 3; k ++ )
            {
                int a = plane[j].v[k], b = plane[j].v[(k + 1)
                    % 3];
                if (g[a][b] && !g[b][a])
                    np[cnt ++ ] = {a, b, i};
            }
        m = cnt;
        for (int j = 0; j < m; j ++ ) plane[j] = np[j];
    }
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i ++ )
    {
        scanf("%lf%lf%lf", &q[i].x, &q[i].y, &q[i].z);
        q[i].shake();
    }
    get_convex_3d();

    double res = 0;
    for (int i = 0; i < m; i ++ )

```

```
    res += plane[i].area();
    printf("%lf\n", res);
    return 0;
}
```

7.5 旋转卡壳

7.5.1 平面最远点对

```
/*
旋转卡壳 求平面最远点对 输出距离的平方
注意，当所有点共线时，求得的凸包只包含起点，但并不影响求最远点对，
→ 因为代码第 70 行特判了所有点共线的情况。
另外如果打算求所有点共线的凸包，可以将代码第 60 行的小于等于号变
→ 成小于号。
*/
#include <iostream>
#include <cstring>
#include <algorithm>

#define x first
#define y second

using namespace std;

typedef pair<int, int> PII;
const int N = 50010;

int n;
PII q[N];
int stk[N], top;
bool used[N];

PII operator- (PII a, PII b)
{
    return {a.x - b.x, a.y - b.y};
}

int operator* (PII a, PII b)
{
    return a.x * b.y - a.y * b.x;
}
```

```
int area(PII a, PII b, PII c)
{
    return (b - a) * (c - a);
}

int get_dist(PII a, PII b)
{
    int dx = a.x - b.x;
    int dy = a.y - b.y;
    return dx * dx + dy * dy;
}

void get_convex()
{
    sort(q, q + n);
    for (int i = 0; i < n; i++)
    {
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top -
        ↪ 1]], q[i]) <= 0)
        {
            if (area(q[stk[top - 2]], q[stk[top - 1]], q[i]) <
                ↪ 0)
                used[stk[-- top]] = false;
            else top--;
        }
        stk[top ++] = i;
        used[i] = true;
    }

    used[0] = false;
    for (int i = n - 1; i >= 0; i--)
    {
        if (used[i]) continue;
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top -
        ↪ 1]], q[i]) <= 0)
            top--;
        stk[top ++] = i;
    }
    top --;
}

int rotating_calipers()
{
    if (top <= 2) return get_dist(q[0], q[n - 1]);
}
```

```
int res = 0;
for (int i = 0, j = 2; i < top; i++)
{
    auto d = q[stk[i]], e = q[stk[i + 1]];
    while (area(d, e, q[stk[j]]) < area(d, e, q[stk[j +
        1]])) j = (j + 1) % top;
    res = max(res, max(get_dist(d, q[stk[j]]), get_dist(e,
        q[stk[j]])));
}
return res;
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d%d", &q[i].x,
        &q[i].y);
    get_convex();
    printf("%d\n", rotating_calipers());

    return 0;
}
```

7.5.2 最小矩形覆盖

```
/*
 *
 * 已知平面上不共线的一组点的坐标，求覆盖这组点的面积最小的矩形。
输出矩形的面积和四个顶点的坐标。
输入格式
第一行包含一个整数 n，表示点的数量。
接下来 n 行，每行包含两个用空格隔开的浮点数，表示一个点的 x 坐标
    和 y 坐标。
不用科学计数法，但如果小数部分为 0，则可以写成整数。
输出格式
共 5 行，第 1 行输出一个浮点数，表示所求得的覆盖输入点集的最小矩
    形的面积。
接下来 4 行，每行包含两个用空格隔开的浮点数，表示所求矩形的一个顶
    点的 x 坐标和 y 坐标。
先输出 y 坐标最小的顶点的 x,y 坐标，如果有两个点的 y 坐标同时达到
    最小，则先输出 x 坐标较小者的 x,y 坐标。
然后，按照逆时针的顺序输出其他三个顶点的坐标。
不用科学计数法，精确到小数点后 5 位，后面的 0 不可省略。
```

```
/*
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;
const int N = 50010;
const double eps = 1e-12, INF = 1e20;
const double PI = acos(-1);

int n;
PDD q[N];
PDD ans[N];
double min_area = INF;
int stk[N], top;
bool used[N];

int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

PDD operator+ (PDD a, PDD b)
{
    return {a.x + b.x, a.y + b.y};
}

PDD operator- (PDD a, PDD b)
{
```

```
    return {a.x - b.x, a.y - b.y};
}

PDD operator* (PDD a, double t)
{
    return {a.x * t, a.y * t};
}

PDD operator/ (PDD a, double t)
{
    return {a.x / t, a.y / t};
}

double operator* (PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

double operator& (PDD a, PDD b)
{
    return a.x * b.x + a.y * b.y;
}

double area(PDD a, PDD b, PDD c)
{
    return (b - a) * (c - a);
}

double get_len(PDD a)
{
    return sqrt(a & a);
}

double project(PDD a, PDD b, PDD c)
{
    return ((b - a) & (c - a)) / get_len(b - a);
}

PDD norm(PDD a)
{
    return a / get_len(a);
}

PDD rotate(PDD a, double b)
```

```

{
    return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y *
         $\rightarrow$  cos(b)};
}

void get_convex()
{
    sort(q, q + n);
    for (int i = 0; i < n; i++)
    {
        while (top >= 2 && sign(area(q[stk[top - 2]], 
             $\rightarrow$  q[stk[top - 1]], q[i])) >= 0)
            used[stk[ -- top]] = false;
        stk[top ++ ] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; i--)
    {
        if (used[i]) continue;
        while (top >= 2 && sign(area(q[stk[top - 2]], 
             $\rightarrow$  q[stk[top - 1]], q[i])) >= 0)
            top -- ;
        stk[top ++ ] = i;
    }
    reverse(stk, stk + top);
    top -- ;
}

void rotating_calipers()
{
    for (int i = 0, a = 2, b = 1, c = 2; i < top; i++)
    {
        auto d = q[stk[i]], e = q[stk[i + 1]];
        while (dcmp(area(d, e, q[stk[a]]), area(d, e, q[stk[a
             $\rightarrow$  + 1]])) < 0) a = (a + 1) % top;
        while (dcmp(project(d, e, q[stk[b]]), project(d, e,
             $\rightarrow$  q[stk[b + 1]])) < 0) b = (b + 1) % top;
        if (!i) c = a;
        while (dcmp(project(d, e, q[stk[c]]), project(d, e,
             $\rightarrow$  q[stk[c + 1]])) > 0) c = (c + 1) % top;
        auto x = q[stk[a]], y = q[stk[b]], z = q[stk[c]];
        auto h = area(d, e, x) / get_len(e - d);
        auto w = ((y - z) & (e - d)) / get_len(e - d);
    }
}

```

```

if (h * w < min_area)
{
    min_area = h * w;
    ans[0] = d + norm(e - d) * project(d, e, y);
    ans[3] = d + norm(e - d) * project(d, e, z);
    auto u = norm(rotate(e - d, -PI / 2));
    ans[1] = ans[0] + u * h;
    ans[2] = ans[3] + u * h;
}
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &q[i].x,
        &q[i].y);
    get_convex();
    rotating_calipers();

    int k = 0;
    for (int i = 1; i < 4; i++)
        if (dcmp(ans[i].y, ans[k].y) < 0 || !dcmp(ans[i].y,
            ans[k].y) && dcmp(ans[i].x, ans[k].x))
            k = i;

    printf("%.5lf\n", min_area);
    for (int i = 0; i < 4; i++, k++)
    {
        auto x = ans[k % 4].x, y = ans[k % 4].y;
        if (!sign(x)) x = 0;
        if (!sign(y)) y = 0;
        printf("%.5lf %.5lf\n", x, y);
    }

    return 0;
}

```

7.6 三角剖分（圆和多边形面积交）

```

/*
三角剖分
求圆与多边形面积的交
其圆心位于原点，半径为 R。

```

飞行物可视作一个 N 个顶点的简单多边形。
Updog 希望知道飞行物处于望远镜视野之内的部分的面积。
*/

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;
const int N = 55;
const double eps = 1e-8;
const double PI = acos(-1);

double R;
int n;
PDD q[N], r;

int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

PDD operator+ (PDD a, PDD b)
{
    return {a.x + b.x, a.y + b.y};
}

PDD operator- (PDD a, PDD b)
{
```

```
    return {a.x - b.x, a.y - b.y};
}

PDD operator* (PDD a, double t)
{
    return {a.x * t, a.y * t};
}

PDD operator/ (PDD a, double t)
{
    return {a.x / t, a.y / t};
}

double operator* (PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

double operator& (PDD a, PDD b)
{
    return a.x * b.x + a.y * b.y;
}

double area(PDD a, PDD b, PDD c)
{
    return (b - a) * (c - a);
}

double get_len(PDD a)
{
    return sqrt(a & a);
}

double get_dist(PDD a, PDD b)
{
    return get_len(b - a);
}

double project(PDD a, PDD b, PDD c)
{
    return ((c - a) & (b - a)) / get_len(b - a);
}

PDD rotate(PDD a, double b)
```

```

{
    return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y *
        ↳  cos(b)};
}

PDD norm(PDD a)
{
    return a / get_len(a);
}

bool on_segment(PDD p, PDD a, PDD b)
{
    return !sign((p - a) * (p - b)) && sign((p - a) & (p - b))
        ↳  <= 0;
}

PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w)
{
    auto u = p - q;
    auto t = w * u / (v * w);
    return p + v * t;
}

double get_circle_line_intersection(PDD a, PDD b, PDD& pa,
    ↳  PDD& pb)
{
    auto e = get_line_intersection(a, b - a, r, rotate(b - a,
        ↳  PI / 2));
    auto mind = get_dist(r, e);
    if (!on_segment(e, a, b)) mind = min(get_dist(r, a),
        ↳  get_dist(r, b));
    if (dcmp(R, mind) <= 0) return mind;
    auto len = sqrt(R * R - get_dist(r, e) * get_dist(r, e));
    pa = e + norm(a - b) * len;
    pb = e + norm(b - a) * len;
    return mind;
}

double get_sector(PDD a, PDD b)
{
    auto angle = acos((a & b) / get_len(a) / get_len(b));
    if (sign(a * b) < 0) angle = -angle;
    return R * R * angle / 2;
}

```

```
double get_circle_triangle_area(PDD a, PDD b)
{
    auto da = get_dist(r, a), db = get_dist(r, b);
    if (dcmp(R, da) >= 0 && dcmp(R, db) >= 0) return a * b /
        2;
    if (!sign(a * b)) return 0;
    PDD pa, pb;
    auto mind = get_circle_line_intersection(a, b, pa, pb);
    if (dcmp(R, mind) <= 0) return get_sector(a, b);
    if (dcmp(R, da) >= 0) return a * pb / 2 + get_sector(pb,
        b);
    if (dcmp(R, db) >= 0) return get_sector(a, pa) + pa * b / 2;
    return get_sector(a, pa) + pa * pb / 2 + get_sector(pb,
        b);
}

double work()
{
    double res = 0;
    for (int i = 0; i < n; i++)
        res += get_circle_triangle_area(q[i], q[(i + 1) % n]);
    return fabs(res);
}

int main()
{
    while (scanf("%lf%d", &R, &n) != -1)
    {
        for (int i = 0; i < n; i++) scanf("%lf%lf", &q[i].x,
            &q[i].y);
        printf("%.2lf\n", work());
    }

    return 0;
}
```

7.7 扫描线

7.7.1 三角形面积并

```
/*
 * 给定 n, n 行给出三角形顶点
```

```
/*
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <vector>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;
const int N = 110;
const double eps = 1e-8, INF = 1e6;

int n;
PDD tr[N][3];
PDD q[N];

int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

PDD operator+ (PDD a, PDD b)
{
    return {a.x + b.x, a.y + b.y};
}

PDD operator- (PDD a, PDD b)
{
    return {a.x - b.x, a.y - b.y};
}
```

```
PDD operator* (PDD a, double t)
{
    return {a.x * t, a.y * t};
}

double operator* (PDD a, PDD b)
{
    return a.x * b.y - a.y * b.x;
}

double operator& (PDD a, PDD b)
{
    return a.x * b.x + a.y * b.y;
}

bool on_segment(PDD p, PDD a, PDD b)
{
    return sign((p - a) & (p - b)) <= 0;
}

PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w)
{
    if (!sign(v * w)) return {INF, INF};
    auto u = p - q;
    auto t = w * u / (v * w);
    auto o = p + v * t;
    if (!on_segment(o, p, p + v) || !on_segment(o, q, q + w))
        return {INF, INF};
    return o;
}

double line_area(double a, int side)
{
    int cnt = 0;
    for (int i = 0; i < n; i++)
    {
        auto t = tr[i];
        if (dcmp(t[0].x, a) > 0 || dcmp(t[2].x, a) < 0)
            continue;
        if (!dcmp(t[0].x, a) && !dcmp(t[1].x, a))
        {
            if (side) q[cnt++] = {t[0].y, t[1].y};
        }
        else if (!dcmp(t[2].x, a) && !dcmp(t[1].x, a))
    }
}
```

```
{  
    if (!side) q[cnt ++ ] = {t[2].y, t[1].y};  
}  
else  
{  
    double d[8];  
    int u = 0;  
    for (int j = 0; j < 3; j ++ )  
    {  
        auto o = get_line_intersection(t[j], t[(j + 1)  
        → % 3] - t[j], {a, -INF}, {0, INF * 2});  
        if (dcmp(o.x, INF))  
            d[u ++ ] = o.y;  
    }  
    if (u)  
    {  
        sort(d, d + u);  
        q[cnt ++ ] = {d[0], d[u - 1]};  
    }  
}  
}  
if (!cnt) return 0;  
for (int i = 0; i < cnt; i ++ )  
    if (q[i].x > q[i].y)  
        swap(q[i].x, q[i].y);  
    sort(q, q + cnt);  
    double res = 0, st = q[0].x, ed = q[0].y;  
    for (int i = 1; i < cnt; i ++ )  
        if (q[i].x <= ed) ed = max(ed, q[i].y);  
        else  
        {  
            res += ed - st;  
            st = q[i].x, ed = q[i].y;  
        }  
    res += ed - st;  
    return res;  
}  
  
double range_area(double a, double b)  
{  
    return (line_area(a, 1) + line_area(b, 0)) * (b - a) / 2;  
}  
  
int main()
```

```

{
    scanf("%d", &n);
    vector<double> xs;
    for (int i = 0; i < n; i ++ )
    {
        for (int j = 0; j < 3; j ++ )
        {
            scanf("%lf%lf", &tr[i][j].x, &tr[i][j].y);
            xs.push_back(tr[i][j].x);
        }
        sort(tr[i], tr[i] + 3);
    }
    for (int i = 0; i < n; i ++ )
        for (int j = i + 1; j < n; j ++ )
            for (int x = 0; x < 3; x ++ )
                for (int y = 0; y < 3; y ++ )
                {
                    auto o = get_line_intersection(tr[i][x],
                        tr[i][(x + 1) % 3] - tr[i][x],
                        tr[j][y],
                        tr[j][(y + 1) % 3] - tr[j][y]);
                    if (dcmp(o.x, INF))
                        xs.push_back(o.x);
                }
    sort(xs.begin(), xs.end());
    double res = 0;
    for (int i = 0; i + 1 < xs.size(); i ++ )
        if (dcmp(xs[i], xs[i + 1]))
            res += range_area(xs[i], xs[i + 1]);
    printf("%.2lf\n", res);
    return 0;
}

```

7.7.2 矩形面积并

```

/*
 * 给定 n, n 行给出矩形左下顶点和右上顶点
 */
#include <iostream>
#include <cstring>
#include <algorithm>

```

```
#include <vector>

#define x first
#define y second

using namespace std;

typedef long long LL;
typedef pair<int, int> PII;
const int N = 1010;

int n;
PII l[N], r[N];
PII q[N];

LL range_area(int a, int b)
{
    int cnt = 0;
    for (int i = 0; i < n; i++)
        if (l[i].x <= a && r[i].x >= b)
            q[cnt++] = {l[i].y, r[i].y};
    if (!cnt) return 0;
    sort(q, q + cnt);
    LL res = 0;
    int st = q[0].x, ed = q[0].y;
    for (int i = 1; i < cnt; i++)
        if (q[i].x <= ed) ed = max(ed, q[i].y);
        else
    {
        res += ed - st;
        st = q[i].x, ed = q[i].y;
    }
    res += ed - st;
    return res * (b - a);
}

int main()
{
    scanf("%d", &n);
    vector<int> xs;
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d%d%d", &l[i].x, &l[i].y, &r[i].x, &r[i].y);
        xs.push_back(l[i].x), xs.push_back(r[i].x);
    }
}
```

```
    }
    sort(xs.begin(), xs.end());
    LL res = 0;
    for (int i = 0; i + 1 < xs.size(); i++)
        if (xs[i] != xs[i + 1])
            res += range_area(xs[i], xs[i + 1]);
    printf("%lld\n", res);
    return 0;
}
```

7.8 自适应辛普森积分

```
/*
 * 求  $\sin x/x$  从  $a$  到  $b$  的积分
 */
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;

const double eps = 1e-12;

double f(double x)
{
    return sin(x) / x;
}

double simpson(double l, double r)
{
    auto mid = (l + r) / 2;
    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
}

double asr(double l, double r, double s)
{
    auto mid = (l + r) / 2;
    auto left = simpson(l, mid), right = simpson(mid, r);
    if (fabs(left + right - s) < eps) return left + right;
    return asr(l, mid, left) + asr(mid, r, right);
}

int main()
```

```
{  
    double l, r;  
    scanf("%lf%lf", &l, &r);  
    printf("%lf\n", asr(l, r, simpson(l, r)));  
    return 0;  
}
```

8 字符串

8.1 字符串哈希

```
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull; //相当于模 2^64
const int N=1e5+10,P=131; //P 是经验值，一般取 131 或 13331，然后
→ 模 2^64
ull p[N],h[N]; //p 存储 P 的 i 次方，h 是前缀哈希
char s[N];
ull get(int l,int r)
{
    return h[r]-h[l-1]*p[r-l+1];
/*
h[r] 最高位的次幂比 h[l-1] 高
所以要把 h[l-1] 的次幂乘上去
然后做差，把高位的部分抵掉
比如
h(abcd)=a*p^3+b*p^2+c*p^1+d*p^0
h(ab)=a*p^1+b*p^0
l=3, r=4
h(cd)=c*p^1+d*p^0=h(abcd)-h(ab)*P^(2)
*/
}
int main()
{
    int m,n;
    cin>>m>>n;
    cin>>s+1;
    p[0]=1;
    for(int i=1;i<=m;i++)
    {
        p[i]=p[i-1]*P;
        h[i]=h[i-1]*P+s[i];
        /*
        例子
        abcd
        H(a)=a*P^(0)
        H(ab)=a*P^1+b*P^0=H(a)*P+b
        */
    }
    while(n--)
    {
```

```

int a,b,c,d;
scanf("%d%d%d%d",&a,&b,&c,&d);
if(get(a,b)==get(c,d))puts("Yes");
else puts("No");
}
}
//判断 l-r 是否是回文
ull getl(int l,int r)//从左往右的哈希
{
    return hl[r]-hl[l-1]*p[r-l+1];
}
ull getr(int l,int r)//从右往左的哈希
{
    return hr[l]-hr[r+1]*p[r-l+1];
}
bool check(int l,int r)
{
    return getl(l,r)==getr(l,r);
}

for(int i=1;i<=n;i++)
{
    hl[i]=hl[i-1]*P+s[i];
    p[i]=p[i-1]*P;
}
for(int i=n;i>=1;i--)
{
    hr[i]=hr[i+1]*P+s[i];
}

```

8.2 KMP

```

//2020/7/27
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
char p[N],s[N];
int ne[N];
int main()
{
    int lp,ls;
    scanf("%d%s",&lp,p+1);
    scanf("%d%s",&ls,s+1);
    for(int i=1,j=0;i<=lp;)//不需要 i++

```

```

{
    if(j==0 || p[i]==p[j]) ne[++i]=++j;
    else j=ne[j];
}
int i=1,j=1;
while(i<=ls)
{
    if(p[j]==s[i] || j==0)
    {
        i++;
        j++;
        if(j>lp)
        {
            cout<<i-j<<' ';
            j=ne[j];
        }
    }
    else j=ne[j];
}

return 0;
}

```

8.3 EXKMP

```

const int maxn=100010; //字符串长度最大值
int next[maxn], ex[maxn]; //ex 数组即为 extend 数组
//预处理计算 next 数组
/*
    拓展 kmp 是对 KMP 算法的扩展，它解决如下问题：

```

定义母串 S 和字串 T , 设 S 的长度为 n , T 的长度为 m , 求 T 与 S 的
 ↳ 每一个后缀的最长公共前缀, 也就是说, 设 $extend$ 数组, $extend[i]$
 ↳ 表示 T 与 $S[i, n-1]$ 的最长公共前缀, 要求出所有
 ↳ $extend[i]$ ($0 \leq i < n$)。

注意到, 如果有一个位置 $extend[i]=m$, 则表示 T 在 S 中出现, 而且是
 ↳ 在位置 i 出现, 这就是标准的 KMP 问题, 所以说拓展 kmp 是对 KMP
 ↳ 算法的扩展, 所以一般将它称为扩展 KMP 算法。

```

*/
void GETNEXT(char *str)
{
    int i=0,j,po,len=strlen(str);
    next[0]=len;//初始化 next[0]

```

```

while(str[i]==str[i+1]&&i+1<len) //计算 next[1]
    i++;
    next[1]=i;
    po=1; //初始化 po 的位置
    for(i=2;i<len;i++)
    {
        if(next[i-po]+i<next[po]+po) //第一种情况，可以直接得到
            ← next[i] 的值
            next[i]=next[i-po];
        else//第二种情况，要继续匹配才能得到 next[i] 的值
        {
            j=next[po]+po-i;
            if(j<0)j=0;//如果 i>po+next[po]，则要从头开始匹配
            while(i+j<len&&str[j]==str[j+i]) //计算 next[i]
                j++;
            next[i]=j;
            po=i; //更新 po 的位置
        }
    }
}

//计算 extend 数组
void EXKMP(char *s1,char *s2)
{
    int i=0,j,po,len=strlen(s1),l2=strlen(s2);
    GETNEXT(s2); //计算子串的 next 数组
    while(s1[i]==s2[i]&&i<l2&&i<len) //计算 ex[0]
        i++;
    ex[0]=i;
    po=0; //初始化 po 的位置
    for(i=1;i<len;i++)
    {
        if(next[i-po]+i<ex[po]+po) //第一种情况，直接可以得到
            ← ex[i] 的值
            ex[i]=next[i-po];
        else//第二种情况，要继续匹配才能得到 ex[i] 的值
        {
            j=ex[po]+po-i;
            if(j<0)j=0;//如果 i>ex[po]+po 则要从头开始匹配
            while(i+j<len&&j<l2&&s1[j+i]==s2[j]) //计算 ex[i]
                j++;
            ex[i]=j;
            po=i; //更新 po 的位置
        }
    }
}

```

```

    }
}

```

8.4 manacher

```

#include<iostream>
#include<cstdio>
using namespace std;
const int maxn = 100010;
char ma[maxn << 1], s[maxn];
int mp[maxn << 1];
void Manacher(char s[], int len)
{
    int l = 0;
    ma[l++] = '$';
    ma[l++] = '#';
    for(int i = 0; i < len; i++)
    {
        ma[l++] = s[i];
        ma[l++] = '#';
    }
    ma[l] = 0;
    /* i 代表此刻字符串的位置
     * id 代表此时最大回文子串的中心
     * mx 代表当前得到的最大回文子串的右边界 */
    int mx = 0;
    int id = 0;
    for(int i = 0; i < l; i++)
    {
        mp[i] = mx > i ? min(mp[id * 2 - i], mx - i) : 1;
        while(ma[i + mp[i]] == ma[i - mp[i]])
            mp[i]++;
        if(i + mp[i] > mx)
        {
            mx = i + mp[i];
            id = i;
        }
    }
}
int main()
{
    while(scanf("%s", s) == 1)
    {
        int len = strlen(s);

```

```
    Manacher(s,len);
    int ans = 0;
    for(int i = 0;i < 2 * len + 2; i++)
        ans = max(ans,mp[i] - 1);
    printf("%d\n",ans);
}
}

/*
* a b a a b a
* i:      0 1 2 3 4 5 6 7 8 9 10 11 12 13
* Ma[i]: $ # a # b # a # a # b # a #
* Mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1
*/

```

8.5 AC 自动机

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 10010, S = 55, M = 1000010;

int n;
int tr[N * S][26], cnt[N * S], idx;
char str[M];
int q[N * S], ne[N * S];

void insert()
{
    int p = 0;
    for (int i = 0; str[i]; i++)
    {
        int t = str[i] - 'a';
        if (!tr[p][t]) tr[p][t] = ++idx;
        p = tr[p][t];
    }
    cnt[p]++;
}

void build()
```

```
{  
    int hh = 0, tt = -1;  
    for (int i = 0; i < 26; i++)  
        if (tr[0][i])  
            q[++tt] = tr[0][i];  
  
    while (hh <= tt)  
    {  
        int t = q[hh++];  
        for (int i = 0; i < 26; i++)  
        {  
            int p = tr[t][i];  
            if (!p) tr[t][i] = tr[ne[t]][i];  
            else  
            {  
                ne[p] = tr[ne[t]][i];  
                q[++tt] = p;  
            }  
        }  
    }  
}  
  
int main()  
{  
    int T;  
    scanf("%d", &T);  
    while (T--)  
    {  
        memset(tr, 0, sizeof tr);  
        memset(cnt, 0, sizeof cnt);  
        memset(ne, 0, sizeof ne);  
        idx = 0;  
  
        scanf("%d", &n);  
        for (int i = 0; i < n; i++)  
        {  
            scanf("%s", str);  
            insert();  
        }  
  
        build();  
  
        scanf("%s", str);
```

```

int res = 0;
for (int i = 0, j = 0; str[i]; i++)
{
    int t = str[i] - 'a';
    j = tr[j][t];

    int p = j;
    while (p)
    {
        res += cnt[p];
        cnt[p] = 0;
        p = ne[p];
    }
}

printf("%d\n", res);
}

return 0;
}

```

8.6 后缀自动机

```

/*
后缀自动机
图中每个点就是一个等价类，这些点形成的串（从原点到这个点的路径）都
→ 有相同的 endpos 集合。任意两个点的串没有任何交集。每个点对应
→ 的串一定都是最长串的连续后缀。每个点的最短串删掉头部的一个字
→ 符后，连条绿边到短串剩下的对应串的点。

```

当前等价类是 p ，则该等价类对应串的公共后缀是 $fa[p]$ 的 len ，该等价
 \hookrightarrow 类最短串是 $fa[p].len+1$
如下是枚举每个等价类，每次相当于枚举一堆子串，每次枚举知道这堆子串
 \hookrightarrow 出现了 fi 次，且其中最长的是 $node[u].len$ ，最短的是
 $\hookrightarrow node[node[u].fa].len+1$ 。相当于枚举出所有子串

```

多组
idx=ans=0;
tot=1, last=1;
memset(h, -1, sizeof h);
memset(node, 0, sizeof node);
memset(f, 0, sizeof f); //!!!!
*/
#include <iostream>

```

```

#include <cstring>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 2000010; //节点数量是长度的两倍 !!

int tot = 1, last = 1;
struct Node
{
    int len, fa; //len 表示当前状态最大串的长度, fa 是绿边。
    int ch[26]; //蓝边
} node[N];
char str[N];

LL f[N], ans; //f[i] 表示 i 这类 (endpos 集合) 内的元素数。所以枚举所有集合就是枚举所有子串, 如果 f[i]>1 说明这个集合的串出现次数都大于 1。然后用 f[i]*len
int h[N], e[N], ne[N], idx;

void extend(int c)
{
    int p = last, np = last = ++ tot;
    f[tot] = 1; //这个位置前的前缀记录一下
    node[np].len = node[p].len + 1;
    for (; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else
    {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else
        {
            int nq = ++ tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
            for (; p && node[p].ch[c] == q; p = node[p].fa)
                node[p].ch[c] = nq;
        }
    }
}

```

```
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void dfs(int u)
{
    for (int i = h[u]; ~i; i = ne[i])
    {
        dfs(e[i]);
        f[u] += f[e[i]];
    }
    if (f[u] > 1) ans = max(ans, f[u] * node[u].len);
}

int main()
{
    scanf("%s", str);
    for (int i = 0; str[i]; i++) extend(str[i] - 'a');
    memset(h, -1, sizeof h);
    for (int i = 2; i <= tot; i++) add(node[i].fa, i);
    dfs(1);
    printf("%lld\n", ans);

    return 0;
}
```

8.7 后缀数组

```
/*
后缀数组
多组，且可以求  $lcp(i, j)$ ，即  $querymin(i+1, j)$ ，这里  $i, j$  都是排名
 $Sa[i]$  排名第  $i$  的是哪个后缀
 $Rank[i]$  第  $i$  个后缀的排名
 $Height[i]$   $sa[i]$  和  $sa[i-1]$  的最长公共前缀
 $Lcp(i, j)=\min(lcp(i, i+1), \dots, lcp(j-1, j))$ .
*/
#include<bits/stdc++.h>

using namespace std;

const int N = 5005, M=15;
```

```

int n, m;
char s[N];
int sa[N], x[N], y[N], c[N], rk[N],
→ height[N], dp[N], lg[N], f2[N][M];

void get_sa()
{
    for (int i = 1; i <= n; i++) c[x[i]]++;
    for (int i = 2; i <= m; i++) c[i] += c[i - 1];
    for (int i = n; i; i--) sa[c[x[i]]--] = i;
    for (int k = 1; k <= n; k <= 1)
    {
        int num = 0;
        for (int i = n - k + 1; i <= n; i++) y[++num] = i;
        for (int i = 1; i <= n; i++)
            if (sa[i] > k)
                y[++num] = sa[i] - k;
        for (int i = 1; i <= m; i++) c[i] = 0;
        for (int i = 1; i <= n; i++) c[x[i]]++;
        for (int i = 2; i <= m; i++) c[i] += c[i - 1];
        for (int i = n; i; i--) sa[c[x[y[i]]]]--;
        → y[i], y[i] = 0;
        swap(x, y);
        x[sa[1]] = 1, num = 1;
        for (int i = 2; i <= n; i++)
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] &&
            → y[sa[i] + k] == y[sa[i - 1] + k]) ? num :
            → ++num;
        if (num == n) break;
        m = num;
    }
}

void get_height()
{
    for (int i = 1; i <= n; i++) rk[sa[i]] = i;
    for (int i = 1, k = 0; i <= n; i++)
    {
        if (rk[i] == 1) continue;
        if (k) k--;
        int j = sa[rk[i] - 1];
        while (i + k <= n && j + k <= n && s[i + k] == s[j +
        → k]) k++;
        height[rk[i]] = k;
    }
}

```

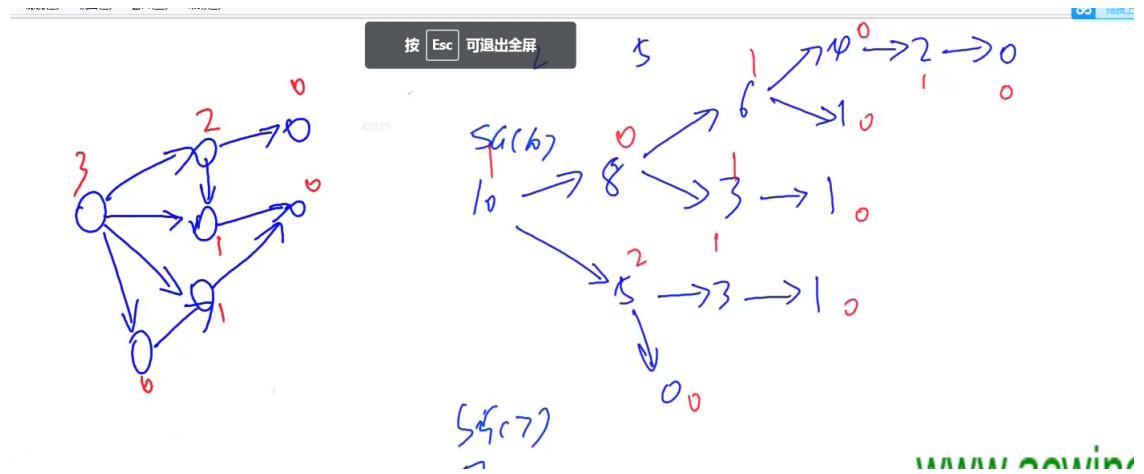
```
    }
}

void inimin()
{
    for(int j=0;j<M;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
    {
        if(!j)f2[i][j]=height[i];
        else f2[i][j]=min(f2[i][j-1],f2[i+(1<<j-1)][j-1]);
    }

}
int querymin(int l,int r)
{
    // if(l>r)swap(l,r);
    // if(max(l,r)>n)return 0;
    // 求 lcp 时 (i+1, j) 因为左边加了 1, 可能超过 n。
    int t=r-l+1,k;
    k=lg[t];
    return min(f2[l][k],f2[r-(1<<k)+1][k]);
}
int main()
{
    for(int i=2;i<N;i++)lg[i]=lg[i/2]+1;
    int t;
    cin>>t;
    while(t--)
    {
        cin>>n;
        scanf("%s", s + 1);
        m = 122;
        for(int i=0;i<=max(n,122);i++)
        {
            sa[i]=x[i]=y[i]=c[i]=rk[i]=height[i]=0;
        }
        get_sa();
        get_height();
        inimin();
    }
    return 0;
}
```

9 博弈论

9.1 SG 函数



```

/*
给定 n 堆石子以及一个由 k 个不同正整数构成的数字集合 S。
现在有两位玩家轮流操作，每次操作可以从任意一堆石子中拿取石子，每次
→ 拿取的石子数量必须包含于集合 S，最后无法进行操作的人视为失败。
问如果两人都采用最优策略，先手是否必胜。
*/
//对每一堆石子求个 sg 函数，用记忆化，最后将每一堆石子的 sg 异或。
// → 为 0 则先手必败，否则先手必胜
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int k,n;
int s[N],h[N],f[N];
int sg(int x)
{
    if(f[x]>=0) return f[x];
    unordered_set<int>S;
    for(int i=0;i<k;i++)
    {
        if(x>=s[i]) S.insert(sg(x-s[i]));
    }
    for(int i=0;;i++) //当是最后一个点的时候，刚好保证是 0
    {
        if(!S.count(i)) return f[x]=i; //求 mex
    }
}
int main()

```

```
{  
    cin>>k;  
    for(int i=0;i<k;i++)cin>>s[i];  
    cin>>n;  
    for(int i=0;i<n;i++)cin>>h[i];  
    int res=0;  
    memset(f,-1,sizeof f);  
    for(int i=0;i<n;i++)  
    {  
        res^=sg(h[i]);  
    }  
    if(res)puts("Yes");  
    else puts("No");  
    return 0;  
}
```

10 混合板子

10.1 杜教 BM (求线性递推)

```
//杜教 BM 数据大时都改为 long long t 组可能会超时
//对于线性递推式，给出前 8 项，可以快速算出后面的项，1e18 都可以
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b)
{
    ll res=1;
    a%=mod;
    assert(b>=0);
    for(; b; b>>=1)
    {
        if(b&1)res=res*a%mod;
        a=a*a%mod;
    }
    return res;
}
ll _,n;
namespace linear_seq
{
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];
    vector<ll> Md;
    void mul(ll *a,ll *b,int k)
    {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k)
            → _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1; i>=k; i--) if (_c[i])
```

```

rep(j,0,SZ(Md))
    ↪ _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
rep(i,0,k) a[i]=_c[i];
}
int solve(ll n,VI a,VI b)
{
    ll ans=0,pnt=0;
    int k=SZ(a);
    assert(SZ(a)==SZ(b));
    rep(i,0,k) _md[k-1-i]=-a[i];
    _md[k]=1;
    Md.clear();
    rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
    rep(i,0,k) res[i]=base[i]=0;
    res[0]=1;
    while ((1ll<<pnt)<=n) pnt++;
    for (int p=pnt; p>=0; p--)
    {
        mul(res,res,k);
        if ((n>>p)&1)
        {
            for (int i=k-1; i>=0; i--) res[i+1]=res[i];
            res[0]=0;
            rep(j,0,SZ(Md))
                ↪ res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
        }
    }
    rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
    if (ans<0) ans+=mod;
    return ans;
}
VI BM(VI s)
{
    VI C(1,1),B(1,1);
    int L=0,m=1,b=1;
    rep(n,0,SZ(s))
    {
        ll d=0;
        rep(i,0,L+1) d=(d+(1ll)C[i]*s[n-i])%mod;
        if (d==0) ++m;
        else if (2*L<=n)
        {
            VI T=C;
            ll c=mod-d*powmod(b,mod-2)%mod;

```

```

while (SZ(C)<SZ(B)+m) C.pb(0);
rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
L=n+1-L;
B=T;
b=d;
m=1;
}
else
{
    ll c=mod-d*powmod(b,mod-2)%mod;
    while (SZ(C)<SZ(B)+m) C.pb(0);
    rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
    ++m;
}
}
return C;
}
int gao(VI a,ll n)
{
    VI c=BM(a);
    c.erase(c.begin());
    rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
    return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
}
};

ll f[205];
int main()
{
    ll n,m;
    scanf("%lld%lld",&n,&m);
    for(int i=1;i<=m;i++) f[i]=1;
    for(int i=m;i<=200;i++)
        f[i]=(f[i-1]+f[i-m])%mod;
    cout<<f[m];
    vector<int>v;
    n++;//实际加不加 1 要看情况
    for(int i=1;i<=200;i++)
        v.push_back(f[i]); //至少 8 项，越多越好。
    ←
    printf("%lld\n",linear_seq::gao(v,n-1)%mod);
}

```

10.2 complex 复数板子



C++ complex复数类用法详解

复数是 $a+bi$ 形式的数，其中 a 和 b 是实数，在 C++ 代码中是浮点值， i 是根号 -1 。 a 被称作复数的 **实数部分**， b 乘以 i 被称作**虚数部分**。

使用复数的程序一般都很专业，例如，复数可以用于电气和电磁理论、数字信号处理，当然也可以用于数学。复数可以用来生成非常复杂的 Mandelbrot 集合和 Julia 集合的分形图。

complex 头文件定义了用于处理复数的功能。complex<T> 模板类型的实例表示的是复数，这里定义了 3 个特化类型：complex<float>、complex<double>、complex<long double>。在这一节中，全部使用 complex<double>，但其他特化类型的操作是基本相同的。

生成表示复数的对象

complex<double> 类型的构造函数接受两个参数，第一个参数是实部的值，第二个参数是虚部的值。例如：

```
01. std::complex<double> z1 {2, 5}; // 2 + 5i
02. std::complex<double> z; // Default parameter values, are 0 so 0 + 0i
```

它也有拷贝构造函数，因此可以按如下方式复制 z1：

```
01. std::complex<double> z2 {z1}; // 2 + 5i
```

显然，我们需要复数常量以及复数对象，命名空间 std::literals::complex_literals 中定义了 3 个运算符函数，在这个命名空间中，命名空间 literals 和 complex_literals 都是内联定义的。在对 std::literals::complex_literals、std::literals 或 std::complex_literals 使用 using 指令之后，就可以访问用于复数常量的运算符函数。假设使用了一个或多个这种指令，并且 using std::complex 对这一节余下的代码都有效。

运算符 ""i() 函数定义了实部为 0 的 complex<double> 类型的常量。因此，3i 是一个等同于 complex<double>{0, 3} 的常量。当然，可以用实部和虚部表示复数。例如：

```
01. z = 5.0 + 3i; // z is now complex<double>{5, 3}
```

这展示了如何定义两部分都是非零值的复数，并顺便说明已经为复数对象实现了赋值运算符。可对 complex<float> 常量使用后缀if，对 complex<long double> 常量使用后缀il，例如 22if 或 3.

这些后缀是由函数 `operator""if()` 和 `operator""il()` 定义的。注意，不能写成 `1.0+i` 或 `2.0+il`，因为这里的 `i` 和 `il` 会被解释为变量名，必须写成 `1.0 +li` 和 `2.0+1.0il`。

所有的复数类型都定义了成员函数 `real()` 和 `imag()`，它们可以用来访问对象的实部或虚部，或者用提供的参数设置这些部分。例如：

```
01. complex<double> z{1.5, -2.5}; // z: 1.5 - 2.5i
02. z.imag(99); // z: 1.5 + 99.0i
03. z.real(-4.5); // z: -4.5 + 99.0i
04. std::cout << "Real part: " << z.real() << " Imaginary part: " << z.imag() <<
   std::endl;
05. // Real part: -4.5 Imaginary part: 99
```

`real()` 和 `imag()` 接受参数的版本什么都不会返回。

有为复数对象实现流的插入和提取的非成员函数模板。当从流中读取一个复数时，它可能只有实部，例如 `55`，或者括号中只有实部，例如 `(2.6)`，或者实部和虚部在由一个逗号隔开的括号中，例如 `(3,-2)`。如果只提供了实部，虚部会为 `0`。下面是一个示例：

```
01. complex<double> z1, z2, z3; // 3 default objects 0+0i
02. std::cout << "Enter 3 complex numbers: ";
03. std::cin >> z1 >> z2 >> z3; // Read 3 complex numbers
04. std::cout << " z1 = " << z1 << " z2 = " << z2 << " z3 = " << z3 <<
   std::endl;
```

下面是示例的输入和输出结果：

```
Enter 3 complex numbers: -4 (6) (-3, 7)
z1 = (-4,0) z2 = (6,0) z3 = (-3,7)
```

如果输入的一个复数没有括号，就不会有虚部。但是，在括号中可以省略虚部。复数的输出周围总是有括号，虚部即使为 `0` 也会被输出。

复数的运算

`complex` 类模板为有复数操作数的二元运算符 `+`、`-`、`*`、`/` 及一元 `+` 和 `-` 运算符定义了非成员函数。成员函数定义了 `+=`、`-=`、`*=` 和 `/=`。下面是使用它们的一些示例：

```
01. complex<double> z {1,2}; // 1+2i
02. auto z1 = z + 3.0; // 4+2i
03. auto z2 = z*z + (2.0 + 4i); // -1+8i
04. auto z3 = z1 - z2; // 5-6i
05. z3 /= z2; // 815385-0.523077i
```

注意，复数对象和数值常量之间的运算需要数值常量是正确的类型。不能将整数常量加到 `complex<double>` 对象上；为了能够进行这个运算，必须写成 2.0。

复数上的比较和其他运算

一些非成员函数模板可以用来比较两个复数对象相等或不相等。也有 `==` 和 `!=` 运算可以用来比较复数对象和数值，这里数值会被看作虚部为 0 的复数。为了相等，所有的部分都必须相等，如果操作数的实部或虚部不同，它们就不相等。例如：

```
01. complex<double> z1 {3,4}; // 3+4i
02. complex<double> z2 {4,-3}; // 4-3i
03. std::cout << std::boolalpha << (z1 == z2) << " " // false
04. << (z1 != (3.0 + 4i)) << " " // false
05. << (z2 == 4.0 - 3i) << '\n'; // true
```

注释中的结果很清楚。注意在最后一个比较中，编译器会将 `4.0-3i` 看作复数。

另一种比较复数的方法是比较它们的量。各部分值和复数的实部及虚部都相同的向量的量和复数相同，是两部分平方和的平方根。非成员函数模板 `abs()` 接受 `complex<T>` 类型的参数，并返回一个 `T` 类型的量。下面是一个将 `abs()` 函数应用到前面的代码段中定义的 `z1` 和 `z2` 上的示例：

```
01. std::cout << std::boolalpha
02. << (std::abs(z1) == std::abs(z2)) // true
03. << " " << std::abs(z2 + 4.0 + 9i); // 10
```

最后的输出值是 10，因为作为 `abs()` 的参数的表达式的计算结果是 $(8.0+6i)$ ；82 和 62 是 100，平方根是 10。

- `norm()` 函数模板会返回复数的量的平方。
- `arg()` 模板会返回以弧度为单位的相角，是复数 z 对应的 `std::atan(z.imag()/z.real())`。
- `conj()` 函数模板会返回共轭复数，是 $a+bi$ 和 $a-bi$ 。
- `polar()` 函数模板接受量和相角作为参数，并返回和它们对应的复数对象。
- `proj()` 函数模板返回的复数是复数参数在黎曼球上的投影。

一些非成员函数模板提供了一整套的三角函数，并为复数参数提供了双曲函数。也有用于复数参数的 `cmath` 版本的函数 `exp()`、`pow()`、`log()`、`log10()` 和 `sqrt()`。下面是一个有趣的示例：

```
01. complex<double> zc {0.0, std::acos(-1)};
02. std::cout << (std::exp(zc) + 1.0) << '\n'; // (0, 1.22465e-16) or zero
near enough
```

`acos(-1)` 是 π ，所以这揭示了欧拉方程令人震惊的真相， π 和欧拉数 e 是有关联的： $e^{i\pi}+1=0$ 。



10.3 快读

```
//普通快读
inline ll read()
{
    ll x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
//高速快读，本地无法使用，在oj上才可以。会增加空间开销
inline char nc() {
    static char buf[1000000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1,
        1000000, stdin), p1 == p2) ? EOF : *p1++;
}
template <typename _Tp> inline void read(_Tp&sum) {
    char ch = nc(); sum = 0;
    while (!(ch >= '0'&&ch <= '9')) ch = nc();
    while (ch >= '0'&&ch <= '9') sum = (sum << 3) + (sum << 1)
        + (ch - 48), ch = nc();
}
```

10.4 随机与 cin 解绑

```
#define IO ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)

//得到 l-r 之间的整数随机数
#define RAND mt19937
→ rng(chrono::steady_clock::now().time_since_epoch().count())
#define grand(l,r) uniform_int_distribution<int>(l,r)(rng)
```

使用方法
RAND;

```
int x=rand(l,r);

double rand(double l,double r)//得到 l-r 之间的浮点随机数
{
    return (double)rand()/RAND_MAX*(r-l)+l;
}
```