ACM板子

对拍

```
#!/bin/bash
g++ std.cpp -o std -Wall
# g++ data.cpp -o data -Wall
g++ brute_force.cpp -o brute_force -Wall
while true;do
    # ./data>data.in
    python3 data.py
    ./std<data.in>std.out
    ./brute_force<data.in>brute.out
    if diff -b -B -q brute.out std.out;then
        echo "accept"
    else
        echo "Fake!Wrong Answer!"
        echo -n "std:"
        cat std.out
        echo -n "brute:"
        cat brute.out
        break
    fi
done


//双斜线后面为注解，不是check.sh的内容
#!/bin/bash                          //假装这是个头文件(我也不
知道啥意思
g++ a.cpp -o a -Wall
g++ data.cpp -o data -Wall
g++ brute_force.cpp -o brute_force -Wall   //编译你的三个代码
while true;do                            //一直做，直到拍出错
误
    ./data>data.in
    ./a<data.in>a.out
    ./brute_force<data.in>brute.out       //造输入输出，然后用
你自己的程序再去造一组输出
    if diff -b -B -q test.out std.out;then         //diff -b
 -B -q后解俩文件是比较这俩文件的意思，-b -B -q是忽略制表符行末空
格以及空行的影响。如果没差别则会返回逻辑真
        echo "Wonderful!"                           //没差
别输出Wonderful!
    else
        echo "Fake!Wrong Answer!"
        break                                       //有差别
输出WA，并且跳出循环
    fi                                          //if的终止标
志
done                                            //while的终止
标志
```

```
mt19937用法
unsigned seed = std::chrono::system_clock::now().time_since_
epoch().count();
        mt19937 rand_num(seed); // 大随机数
```

造好check.sh后，在终端输入chmod 777 check.sh（相当于取得使用权限）然后就可以输入./check.sh来愉快的对拍了。

另外，在终端输入time a<data.in>data.out可以查看你的程序跑了多少时间。

高精度

```
struct bigint
{
    static const ll p = (ll)1e8;
    static const ll width = 8;
    vector<ll> s;
    bool flag = true;//flase负
    bigint(ll num = 0)
    {
        *this = num;
    }
    bigint(vector<ll> num)
    {
        s = num;
    }
    bigint operator=(ll num)
    {
        s.clear();
        if (num <
        0)
        {
            flag = false;
```

```cpp
            num = -num;
        }
        while (num)
        {
            s.push_back(num%p);
            num /= p;
        }
        return *this;
    }
    bigint(string str)
    {
        *this = str;
    }
    bigint operator=(string & str)
    {
        s.clear();
        int pos = 0;
        if (str[0] == '-')
        {
            pos = 1;
            flag = false;
        }
        ll x, len = (str.length() - 1) / width + 1;
        for (int i = 0; i < len; i++) {
            ll end = str.length() - i * width;
            ll start = max((ll)pos, end - width);
            sscanf(str.substr(start, end - start).c_str(),
"%lld", &x);
            s.push_back(x);
        }
        return *this;
    }
    bigint operator=(bigint ans)
    {
        s = ans.s;
        return *this;
    }
    bigint operator+(bigint a)
    {
        vector<ll> C;
        ll t = 0;
        vector<ll> & B = a.s;
        vector<ll> & A = s;
        for (int i = 0; i < s.size() || i < a.s.size(); i++)
        {
            if (i < A.size()) t += A[i];
            if (i < B.size()) t += B[i];
            C.push_back(t % p);
            t /= p;
        }
        if (t) C.push_back(1);
        return bigint(C);
    }
    bool judge(bigint b)
    {
        if (s.size() > b.s.size())
            return true;
        if (b.s.size() > s.size())
            return false;
        for (int i = s.size() - 1; i >= 0; i--)
        {
            if (s[i] > b.s[i])
                return true;
            if (b.s[i] > s[i])
                return false;
        }
        return true;
    }
    friend bigint operator-(bigint a,bigint b)
    {
        vector<ll> A, B;
        bigint ans;
        if (!a.judge(b))
            A = b.s, B = a.s, ans.flag = false;
        else
            A = a.s,B = b.s;
        for (int i = 0; i < A.size(); i++)
        {
            ll val1=A[i], val2=0;
            if (i < B.size())
                val2 = B[i];
            if (val1 < val2)
            {
                val1 += p;
                A[i + 1]--;
            }
            ans.s.push_back(val1 - val2);
        }
        while (!ans.s.empty() && ans.s.back() == 0)
            ans.s.pop_back();
        return ans;
    }
    vector<ll> Plus(vector<ll>a, ll b)
    {
        vector<ll>c;
        c.clear();
        a[0] += b;
        ll t = 0;
        for (int i = 0; i < a.size() || t; i++)
        {
            t += a[i];
            c.push_back(t % p);
            t = t / p;
        }
        return c;
    }
    bigint Mul(vector<ll>a, ll b)
    {
        vector<ll> c;
        if (b == 0)
        {
            c.push_back(0);
            return bigint(c);
        }
        c.clear();
        ll t = 0;
        for (int i = 0; i < a.size() || t; i++)
        {
            if (i < a.size()) t += b * a[i];
            c.push_back(t % p);
            t = t / p;
        }
        return bigint(c);
    }
    bigint operator*(bigint A)
    {
        vector<ll>c;
        c.clear();
        bigint d;
        vector<ll>& a = s;
        vector<ll>& b = A.s;
        for (int i = 0; i < a.size() || d.s.size(); i++)
        {
            if (i < a.size())
                d = d + Mul(b, a[i]);
            c.push_back(d.s[0]);
            d.s.erase(d.s.begin(), d.s.begin() + 1);
        }
        return bigint(c);
    }
    void show()
    {
        if (s.empty())
        {
            printf("0\n");
            return;
        }
        if (!flag)
            putchar('-');
        printf("%lld", s.back());
        for (int i = s.size() - 2; i >= 0; i--)
        {
            printf("%08lld", s[i]);
        }
        printf("\n");
    }
```

科技

```
struct ios {
    inline char read(){
        static const int IN_LEN=4e6+10;
        static char buf[IN_LEN],*s,*t;
        return (s==t)&&(t=
(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t?-1:*s++;
    }
    template <typename _Tp> inline ios & operator >> (_Tp&x)
{
        static char c11,boo;
        for(c11=read(),boo=0;!isdigit(c11);c11=read()){
            if(c11==-1)return *this;
            boo|=c11=='-';
        }
        for(x=0;isdigit(c11);c11=read())x=x*10+(c11^'0');
        boo&&(x=-x);
        return *this;
    }
} io;
namespace IO
{
    template <typename T>
    inline void w(T x)
    {
        if (x > 9)
            w(x / 10);
        putchar(x % 10 + 48);
    }
    template <typename T>
    inline void write(T x, char c)
    {
        if (x < 0)
            putchar('-'), x = -x;
        w(x);
        putchar(c);
    }
    template <typename T>
    inline void read(T &x)
    {
        x = 0;
        T f = 1;
        char c = getchar();
        for (; !isdigit(c); c = getchar())
            if (c == '-')
                f = -1;
        for (; isdigit(c); c = getchar())
            x = (x << 1) + (x << 3) + (c ^ 48);
        x *= f;
    }
}; // namespace IO
#pragma GCC optimize(3, "Ofast", "inline")

#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
```

```
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

1.数据结构

dfs序性质

1.维护当前节点的子树，即添加的时候对每个点直接$+1$，(整个子树dfs序互通)
查询$L[u]-R[u]$即可
2.维护当前链的祖先中的所有值，添加利用差分,$L[u]+1,(R[u]+1)-1,$查询时
查询前缀和即可

1.树链剖分

如果给的是将边权转化为点权，一般是将父亲到儿子的边的边权赋给儿子，因为每个儿子都只有一个父亲，然后在查询时把路径的两个端点的LCA去掉就好了。

```cpp
ll mod = 998244353;
const ll inf = 1e18;
#define fir(i, a, b) for (int i = a; i <= b; i++)
const int maxn = 1e5 + 20, N = 1e5+20, M = 3000 + 10;
struct node
{
    int v;
    int next;
};
int dep[maxn],fa[maxn],son[maxn],siz[maxn],head[maxn];
ll dis[maxn],w[maxn];
int dfn[maxn],idx=0,top[maxn];
node e[maxn<<1];
int tot=0;
void add(int x,int y)
{
    e[++tot] = {y, head[x]}, head[x] = tot;
}
ll add(ll x, ll t,ll mod) {
x %= mod;t %= mod;return (x + t) % mod;
}
void dfs1(int u,int f)
{
    dep[u]=dep[f]+1;
    fa[u]=f;
    siz[u]=1;
    int maxson=-1;
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].v;
        if(v!=f)
        {
            dfs1(v,u);
            siz[u]+=siz[v];
            if(siz[v]>maxson)
            son[u]=v,maxson=siz[v];
        }
    }
}
void dfs2(int u,int t)
{
    dfn[u]=++idx;
    top[u]=t;
    w[idx]=dis[u];//w[i]为转换为序列后的值
    if(!son[u])
    return;
    dfs2(son[u],t);
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].v;
        if(v!=fa[u]&&v!=son[u])
        dfs2(v,v);
    }
```

```
}
#define left (i<<1)
#define right (i<<1|1)
struct Segnode
{
    int l;
    int r;
    ll sum;
    ll add;
};
Segnode sgt[maxn<<2];
inline void push_up(int i)
{
    sgt[i].sum=(sgt[left].sum+sgt[right].sum)%mod;
}
void build(int l,int r,int i)
{
    sgt[i].l=l;
    sgt[i].r=r;
    if(l==r)
    {
        sgt[i].sum=w[l];
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,left);
    build(mid+1,r,right);
    push_up(i);
}
inline void push_down(int i)
{
    if(sgt[i].add)
    {
        sgt[left].sum=add(sgt[left].sum,sgt[i].add*(sgt[lef
t].r-sgt[left].l+1),mod);
        sgt[left].add+=sgt[i].add;
        sgt[right].sum=add(sgt[right].sum,sgt[i].add*(sgt[ri
ght].r-sgt[right].l+1),mod);
        sgt[right].add+=sgt[i].add;
        sgt[i].add=0;
    }
}
ll query(int l,int r,int i)
{
    if(l<=sgt[i].l&&sgt[i].r<=r)
    {
        return sgt[i].sum;
    }
    push_down(i);
    int mid=(sgt[i].l+sgt[i].r)>>1;
    ll ans=0;
    if(l<=mid)
    ans=add(ans,query(l,r,left),mod);
    if(r>mid)
    ans=add(ans,query(l,r,right),mod);
    push_up(i);
    return ans;
}
void change(int l,int r,int i,ll x)
{
    if(l<=sgt[i].l&&sgt[i].r<=r)
    {
        sgt[i].sum=add(sgt[i].sum,(sgt[i].r-sgt[i].l+1)*x,mo
d);
        sgt[i].add=add(sgt[i].add,x,mod);
        return;
    }
    push_down(i);
    int mid=(sgt[i].l+sgt[i].r)>>1;
    if(l<=mid)
    change(l,r,left,x);
    if(r>mid)
    change(l,r,right,x);
    push_up(i);
}
void mofidy(int x,int y,ll z)
{//路径维护
    while(top[x]!=top[y])
    {
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        change(dfn[top[x]],dfn[x],1,z);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])
    swap(x,y);
    change(dfn[x],dfn[y],1,z);
}
ll qrange(int x,int y)
{//路径修改
    ll ans=0;
    while(top[x]!=top[y])
    {//一直找，找到交点
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        ans=add(ans,query(dfn[top[x]],dfn[x],1),mod);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])
    swap(x,y);
    ans=add(ans,query(dfn[x],dfn[y],1),mod);
    return ans;
}
int main()
{
    int n,Q,root;
    cin>>n>>Q>>root>>mod;
    fir(i,1,n)
    scanf("%lld",&dis[i]);
    fir(i,1,n-1)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y),add(y,x);
    }
    dfs1(root,0);
    dfs2(root,0);
    build(1,idx,1);
    while(Q--)
    {
        int ch;
        scanf("%d",&ch);
        int x,y;ll z;
        if(ch==1)
        {
            scanf("%d%d%lld",&x,&y,&z);
            mofidy(x,y,z);
        }
        else if(ch==2)
        {
            scanf("%d%d",&x,&y);
            ll ans=qrange(x,y);
            printf("%lld\n",ans);
        }
        else if(ch==3)
        {
            scanf("%d%lld",&x,&z);
            change(dfn[x],dfn[x]+siz[x]-1,1,z);
        }
        else
        {
            scanf("%d",&x);

printf("%lld\n",query(dfn[x],dfn[x]+siz[x]-1,1));
        }
    }
}
```

2.Splay-区间翻转

```
struct Node
{
    int ch[2];
    int val;
    int size;
    int fa;
    int cnt;
    int lazy;
    Node()
    {
        ch[2] = 0;
        val = size = fa = cnt = lazy = 0;
    }
};
```

```
int n, m;
namespace Splay
{
    Node spl[maxn];
    int cnt = 0, root = 0;
    int st[maxn];
    inline void newnode(int &now, int fa, int &val)
    {
        spl[now = ++cnt].val = val;
        spl[cnt].size = 1;
        spl[cnt].cnt = 1;
        spl[cnt].lazy = 0;
        spl[cnt].fa = fa;
        spl[cnt].ch[0] = spl[cnt].ch[1] = 0;
        if (!fa)
            spl[fa].ch[1] = now, root = now;
    }
    inline void update(int now)
    {
        int l = spl[now].ch[0];
        int x = spl[now].ch[1];
        spl[now].size = spl[l].size + spl[x].size + spl[no
w].cnt;
    }
    inline bool ident(int x, int f) { return spl[f].ch[1] ==
 x; } //right:1,left:0
    inline void connect(int x, int f, int s)
    {
        spl[f].ch[s] = x;
        spl[x].fa = f;
    }
    inline void change_rev(int now)
    {
        if (!now)
            return;
        swap(spl[now].ch[0], spl[now].ch[1]);
        spl[now].lazy ^= 1;
    }
    inline void push_down(int now)
    {
        if (spl[now].lazy)
        {
            int &l = spl[now].ch[0], &r = spl[now].ch[1];
            change_rev(l);
            change_rev(r);
            spl[now].lazy = 0;
        }
    }
    inline void rotate(int x)
    {
        int f = spl[x].fa, ff = spl[f].fa, k = ident(x, f);
        connect(spl[x].ch[k ^ 1], f, k);
        connect(x, ff, ident(f, ff));
        connect(f, x, k ^ 1);
        update(f), update(x);
    }
    inline void splaying(int x, int goal)
    {
        int y = x, top = 0;
        st[++top] = y;
        while (spl[y].fa)
            st[++top] = spl[y].fa, y = spl[y].fa;
        while (top)
            push_down(st[top--]);
        while (spl[x].fa != goal)
        {
            int y = spl[x].fa, z = spl[y].fa;
            if (spl[y].fa != goal)
                rotate((spl[z].ch[1] == y) ^ (spl[y].ch[1] =
= x) ? x : y);
            rotate(x);
        }
        if (!goal)
            root = x;
        update(x);
    }
    inline int find(int val)
    {
        int u = root;
        if (u == 0)
            return -1;
        while (spl[u].ch[spl[u].val < val] && val != spl[u].
val)
            u = spl[u].ch[spl[u].val < val];
        if (spl[u].val != val)
            return -1;
        splaying(u, 0);
        return u;
    }
    void ins(int val)
    {
        int fa = 0, now = root;
        while (now && val != spl[now].val)
        {
            fa = now;
            now = spl[now].ch[spl[now].val < val];
        }
        if (now)
            spl[now].cnt++;
        else
        {
            newnode(now, fa, val);
            connect(now, fa, val > spl[fa].val);
        }
        splaying(now, 0);
    }
    void del(int now)
    {
        splaying(now, 0);
        push_down(now);
        if (spl[now].cnt > 1)
            spl[now].cnt--, spl[now].size--;
        else if (!spl[now].ch[1])
            root = spl[root].ch[0], spl[root].fa = 0;
        else
        {
            int p = spl[now].ch[1];
            push_down(p);
            while (spl[p].ch[0])
                p = spl[p].ch[0], push_down(p);
            splaying(p, now);
            connect(spl[now].ch[0], p, 0);
            root = p;
            spl[p].fa = 0;
            update(root);
        }
    }
    int build(int l, int x, int fa)
    {
        if (l > x)
            return 0;
        int mid = (l + x) >> 1;
        int now;
        newnode(now, fa, mid);
        spl[now].ch[0] = build(l, mid - 1, now);
        spl[now].ch[1] = build(mid + 1, x, now);
        update(now);
        return now;
    }
    int getnum(int rank)
    {
        rank++;
        int now = root;
        while (now)
        {
            push_down(now);
            int lsize = spl[spl[now].ch[0]].size;
            if (lsize + 1 <= rank && rank <= lsize + spl[no
w].cnt)
            {
                splaying(now, 0);
                break;
            }
            if (rank <= lsize)
                now = spl[now].ch[0];
            else
            {
                rank -= lsize + spl[now].cnt;
                now = spl[now].ch[1];
            }
        }
        return now;
    }
    void reverse(int x, int y)
    {
        int l = getnum(x-1), r = getnum(y + 1);
        splaying(l, 0);
        splaying(r, l);
        int pos = spl[r].ch[0];
        change_rev(pos);
    }
    vector<int> ans;
    void dfs(int now)
    {
        push_down(now);
        if (spl[now].ch[0])
            dfs(spl[now].ch[0]);
        if (spl[now].val >= 1 && spl[now].val <= n)
            ans.push_back(spl[now].val);
        if (spl[now].ch[1])
            dfs(spl[now].ch[1]);
    }
    void show()
    {
        ans.clear();
        dfs(root);
        for (int i = 0; i < ans.size() - 1; i++)
            printf("%d ", ans[i]);
        printf("%d\n", ans.back());
    }
};
int ans[maxn];
int main()
{
    //int n, m;
    cin >> n >> m;
    Splay::build(0, n + 1, 0);
    fir(i, 1, m)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        Splay::reverse(l, r);
    }
    Splay::show();
    return 0;
}
```

Splay-普通操作

```
    struct Node
    {
        int ch[2];
        int val;
        int size;
        int fa;
        int cnt;
    };
```

```
struct Splay
{
    Node spl[maxn];
    int cnt = 0, root = 0;
    inline void newnode(int &now, int fa, int &val)
    {
        spl[now = ++cnt].val = val;
        spl[cnt].size++;
        spl[cnt].cnt++;
    }
    inline void update(int now)
    {
        int l = spl[now].ch[0];
        int r = spl[now].ch[1];
        spl[now].size = spl[l].size + spl[r].size + spl[no
w].cnt;
    }
    inline bool ident(int x, int f) { return spl[f].ch[1] ==
 x; } //right:1,left:0
    inline void connect(int x, int f, int s)
    {
        spl[f].ch[s] = x;
        spl[x].fa = f;
    }
    void rotate(int x) //合二为一的旋转
    {
        int f = spl[x].fa, ff = spl[f].fa, k = ident(x, f);
        connect(spl[x].ch[k ^ 1], f, k); //三次建立父子关系
        connect(x, ff, ident(f, ff));
        connect(f, x, k ^ 1);
        update(f), update(x); //别忘了更新大小信息
    }
    void splaying(int x, int top) //代表把x转到top的儿子，top
为0则转到根结点
    {
        if (!top)
            root = x;
        while (spl[x].fa != top)
        {
            int f = spl[x].fa, ff = spl[f].fa;
            if (ff != top)
                ident(f, ff) ^ ident(x, f) ? rotate(x) : rot
ate(f);
            rotate(x); //最后一次都是旋转x
        }
    }
    inline int find(int val)
    {
        int u = root;
        if (u == 0)
            return -1;
        while (spl[u].ch[spl[u].val < val] && val != spl[u].
val)
            u = spl[u].ch[spl[u].val < val];
        if (spl[u].val != val)
```

```
            return -1;
        splaying(u, 0);
        return u;
    }
    void ins(int val)
    {
        int fa = 0, now = root;
        while (now && val != spl[now].val)
        {
            fa = now;
            now = spl[now].ch[spl[now].val < val];
        }
        if (now)
            spl[now].cnt++;
        else
        {
            newnode(now, fa, val);
            connect(now, fa, val > spl[fa].val);
        }
        splaying(now, 0);
    }
    void del(int val)
    {
        int now = find(val);
        splaying(now, 0);
        if (spl[now].cnt > 1)
            spl[now].cnt--, spl[now].size--;
        else if (!spl[now].ch[1])
            root = spl[root].ch[0], spl[root].fa = 0;
        else
        {
            int p = spl[now].ch[1];
            while (spl[p].ch[0])
                p = spl[p].ch[0];
            splaying(p, now);
            connect(spl[now].ch[0], p, 0);
            root = p;
            spl[p].fa = 0;
            update(root);
        }
    }
    int getrank(int val)
    {
        int rank = 1, now = root;
        while (now)
        {
            if (spl[now].val == val)
            {
                rank += spl[spl[now].ch[0]].size;
                splaying(now, 0);
                break;
            }
            if (spl[now].val > val)
                now = spl[now].ch[0];
            else if (spl[now].val < val)
            {
                rank += spl[spl[now].ch[0]].size + spl[now].
cnt;
                now = spl[now].ch[1];
            }
        }
        return rank;
    }
    int getnum(int rank)
    {
        int now = root;
        while (now)
        {
            int lsize = spl[spl[now].ch[0]].size;
            if (lsize + 1 <= rank && rank <= lsize + spl[no
w].cnt)
            {
                splaying(now, 0);
                break;
            }
            if (rank <= lsize)
                now = spl[now].ch[0];
            else
            {
                rank -= lsize + spl[now].cnt;
                now = spl[now].ch[1];
            }
        }
        return spl[now].val;
    }
    void clear()
    {
        while(cnt>=0)
        {
            spl[cnt].ch[1]=spl[cnt].ch[0]=0;
            spl[cnt].val=spl[cnt].size=spl[cnt].fa=spl[cnt].
cnt=0;
            cnt--;
        }
        cnt=0,root=0;
    }
};
Splay a;
int main()
{
    //  freopen("ans.txt", "r", stdin);
    int n, m;
    scanf("%d%d", &n, &m);
    a.clear();
    fir(i, 1, n)
    {
        int x;
        read(x);
        a.ins(x);
    }
    int last = 0;
    int ans = 0;
    fir(i, 1, m)
    {
        int opt, x;
        read(opt), read(x);
        x = last ^ x;
        //  cout<<opt<<' '<<x<<endl;
        if (opt == 1)
            a.ins(x);//插入
        else if (opt == 2)
            a.del(x);//删除
        if (opt <= 2)
            continue;
        else if (opt == 3)
            last = a.getrank(x);//查询x排名
        else if (opt == 4)
            last = a.getnum(x);//查询第x个数
        else if (opt == 5)
            last = a.getnum(a.getrank(x) - 1);//前驱
        else
            last = a.getnum(a.getrank(x + 1));//后继
        ans ^= last;
    }
    printf("%d\n", ans);
}
```

ST表

```
struct ST{
    int st1[N][20], st2[N][20];
    int lg[N];
    int querymin(int l, int r){ // 查询区间[l,r]的最值
        int k = lg[r - l + 1];
        return min(st1[l][k], st1[r - (1 << k) + 1][k]);
    }
    int querygcd(int l, int r){
        int k = lg[r - l + 1];
        return gcd(st2[l][k], st2[r - (1 << k) + 1][k]);
    }
    void init(int *a, int n){ // 初始化
        for(int i = 1; i <= n; i++) st1[i][0] = st2[i][0] =
 a[i];
        int k = log2(n / 2) + 1;
        lg[1] = 0;
        for(int i = 2;i <= n; i++) lg[i] = lg[i >> 1] + 1;
        for(int j = 1; j <= k; j++)
            for(int i = 1; i + (1 << j) - 1 <= n; i++)
                st1[i][j] = min(st1[i][j - 1], st1[i + (1 <<
 (j - 1))][j - 1]);
        for(int j = 1; j <= k; j++)
            for(int i = 1; i + (1 << j) - 1 <= n; i++)
```

```
                    st2[i][j] = gcd(st2[i][j - 1], st2[i + (1 <<
 (j - 1))][j - 1]);
    }
} st;

namespace ST{
    int st[N][20];
    int lg[N];
    int query(int l, int r){ // 查询区间[l,r]的最值
        int k = lg[r - l + 1];
        return max(st[l][k], st[r - (1 << k) + 1][k]);
    }
    void init(int *a, int n){ // 初始化
        for(int i = 1; i <= n; i++) st[i][0] = a[i];
        int k = log2(n / 2) + 1;
        lg[1] = 0;
        for(int i = 2;i <= n; i++) lg[i] = lg[i >> 1] + 1;
        for(int j = 1; j <= k; j++)
            for(int i = 1; i + (1 << j) - 1 <= N; i++)
                st[i][j] = max(st[i][j - 1], st[i + (1 << (j
 - 1))][j - 1]);
    }
};
//不修改区间GCD:
void init_st() {
    int mt = log(n) / log(2) + 1;
    for(int i=1; i<=n; i++)
    st[i][0] = a[i];
    for(int k=1; k<=mt; k++) {
        for(int i=1; i+(1<<k)-1<=n; i++) {
            st[i][k] = gcd(st[i][k-1], st[i+(1<<k-1)][k-1]);
        }
    }
}
int query(int l, int r) {
    int kt = log(r-l+1) / log(2);
    return gcd(st[l][kt], st[r-(1<<kt)+1][kt]);
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i=1; i<=n; i++) {
        scanf("%d", &a[i]);
    }
    init_st();
    for(int i=1; i<=m; i++) {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", query(l, r));
    }
    return 0;
}
```

线段树

```
namespace Segtree
{
#define left (i << 1)
#define right (i << 1 | 1)
    struct node
    {
        int l;
        int r;
        ll sum;
        ll lazy;
    } seg[maxn << 2];
    inline void push_up(int i)
    {
        seg[i].sum = seg[left].sum + seg[right].sum;
    }
    inline void change_node(int i, int val)
    {
        seg[i].sum += val * (seg[i].r - seg[i].l + 1);
        seg[i].lazy += val;
    }
    inline void push_down(int i)
    {
        if (seg[i].lazy)
        {
            change_node(left, seg[i].lazy);
            change_node(right, seg[i].lazy);
            seg[i].lazy = 0;
        }
    }
    void build(int l, int r, int i)
    {
        seg[i].l = l, seg[i].r=r;
        seg[i].lazy = 0;
        if (l == r)
        {
            seg[i].sum = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, left);
        build(mid + 1, r, right);
        push_up(i);
    }
    void change(int l, int r, int i, int val)
    {
        //cout<<seg[i].l<<' '<<seg[i].r<<endl;
        if (l <= seg[i].l && seg[i].r <= r)
        {
            change_node(i, val);
            return;
        }
        push_down(i);
        int mid = (seg[i].l + seg[i].r) >> 1;
        if (l <= mid)
            change(l, r, left, val);
        if (r > mid)
            change(l, r, right, val);
        push_up(i);
    }
    ll query(int l, int r, int i)
    {
        if (l <= seg[i].l && seg[i].r <= r)
        {
            return seg[i].sum;
        }
        push_down(i);
        int mid = (seg[i].l + seg[i].r) >> 1;
        ll ans = 0;
        if (l <= mid)
            ans += query(l, r, left);
        if (r > mid)
            ans += query(l, r, right);
        push_up(i);
        return ans;
    }
};
```

线段树区间加乘

```
int mod;
const int maxn = 1e5 + 20, N = 2e6 + 20, M = 200000 + 10;
namespace Segtree{
    #define left (i<<1)
    #define right (i<<1|1)
    int a[maxn];
    struct node{
        int l;int r;ll sum;
        ll add;ll mul;
    }t[maxn<<2];
    inline ll adc(ll a,ll b){
        return (a+b)%mod;
    }
     ll muc(ll a,ll b){
        return (a*b)%mod;
    }
    void add_val(int i,ll add){
        t[i].sum=adc(t[i].sum,add*(t[i].r-t[i].l+1));
        t[i].add=adc(t[i].add,add);
    }
    void add_mul(int i,ll mul){
        t[i].sum=muc(t[i].sum,mul);
        t[i].mul=muc(t[i].mul,mul);
        t[i].add=muc(t[i].add,mul);
    }
    void push_down(int i){
        if(t[i].mul!=1){
            add_mul(left,t[i].mul);
            add_mul(right,t[i].mul);
            t[i].mul=1;
        }
        if(t[i].add){
            add_val(left,t[i].add);
            add_val(right,t[i].add);
            t[i].add=0;
        }
    }
    void push_up(int i){
        t[i].sum=adc(t[left].sum,t[right].sum);
    }
    void build(int l,int r,int i){
        t[i].l=l,t[i].r=r;
        t[i].add=0;
        t[i].mul=1;
        if(l==r){
            t[i].sum=a[l];
            return;
        }
        int mid=(l+r)>>1;
        build(l,mid,left);
        build(mid+1,r,right);
        push_up(i);
    }
    void change(int l,int r,int i,int add,int mul){
        if(l<=t[i].l&&t[i].r<=r){
            if(add)add_val(i,add);
            if(mul!=1)add_mul(i,mul);
            return;
        }
        push_down(i);
        int mid=(t[i].l+t[i].r)>>1;
        if(l<=mid)change(l,r,left,add,mul);
        if(r>mid)change(l,r,right,add,mul);
        push_up(i);
    }
    ll query(int l,int r,int i){
        if(l<=t[i].l&&t[i].r<=r){
            return t[i].sum;
        }
        push_down(i);
        ll ans=0;
        int mid=(t[i].l+t[i].r)>>1;
        if(l<=mid)ans=adc(ans,query(l,r,left));
        if(r>mid)ans=adc(ans,query(l,r,right));
        push_up(i);
        return ans;
    }
};
int main()
{
    int debug = 0;
    if (debug)
    {
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
    }
    int n, m;
    scanf("%d%d%d",&n,&m,&mod);
    fir(i,1,n)
    scanf("%d",&Segtree::a[i]);
    Segtree::build(1,n,1);
    while(m--){
        int ch,x,y;
        scanf("%d%d%d",&ch,&x,&y);
        if(ch==3)
        printf("%lld\n",Segtree::query(x,y,1));
        else
        {
            int k;scanf("%d",&k);
            if(ch==1)Segtree::change(x,y,1,0,k);
            else Segtree::change(x,y,1,k,1);
        }
    }
    return 0;
}
```

树套树(区间第K大)

```
struct SplNode
{
    int ch[2];
    int val;
    int size;
    int fa;
    int cnt;
    SplNode()
    {
        ch[2] = 0;
        val = size = fa = cnt = 0;
    }
};
SplNode spl[maxn * 50];
int cnt = 0;
int a[maxn];
int n, m;
struct Splay
{

    int root = 0;
    inline void newnode(int &now, int fa, int &val)
    {
        spl[now = ++cnt].val = val;
        spl[cnt].size = 1;
        spl[cnt].cnt = 1;
        spl[cnt].fa = fa;
        spl[cnt].ch[0] = spl[cnt].ch[1] = 0;
        if (!fa)
            spl[fa].ch[1] = now, root = now;
    }
    inline void update(int now)
    {
        int l = spl[now].ch[0];
        int x = spl[now].ch[1];
        spl[now].size = spl[l].size + spl[x].size + spl[no
w].cnt;
    }
    inline bool ident(int x, int f) { return spl[f].ch[1] ==
 x; } //right:1,left:0
    inline void connect(int x, int f, int s)
    {
        spl[f].ch[s] = x;
        spl[x].fa = f;
    }
    inline void rotate(int x)
    {
        int f = spl[x].fa, ff = spl[f].fa, k = ident(x, f);
        connect(spl[x].ch[k ^ 1], f, k);
        connect(x, ff, ident(f, ff));
        connect(f, x, k ^ 1);
        update(f), update(x);
    }
    inline void splaying(int x, int goal)
    {
        while (spl[x].fa != goal)
        {
            int y = spl[x].fa, z = spl[y].fa;
            if (spl[y].fa != goal)
                rotate((spl[z].ch[1] == y) ^ (spl[y].ch[1] =
= x) ? x : y);
            rotate(x);
        }
        if (!goal)
            root = x;
        update(x);
    }
    inline int find(int val)
    {
        int u = root;
        if (u == 0)
            return -1;
        while (spl[u].ch[spl[u].val < val] && val != spl[u].
val)
            u = spl[u].ch[spl[u].val < val];
        if (spl[u].val != val)
            return -1;
        splaying(u, 0);
        return u;
    }
    void ins(int val)
    {
        int fa = 0, now = root;
        while (now && val != spl[now].val)
        {
            fa = now;
            now = spl[now].ch[spl[now].val < val];
        }
        if (now)
            spl[now].cnt++;
        else
        {
            newnode(now, fa, val);
            connect(now, fa, val > spl[fa].val);
        }
        splaying(now, 0);
    }
    void del(int now)
    {
        splaying(now, 0);
        if (spl[now].cnt > 1)
            spl[now].cnt--, spl[now].size--;
        else if (!spl[now].ch[1])
            root = spl[root].ch[0], spl[root].fa = 0;
        else
        {
            int p = spl[now].ch[1];
            while (spl[p].ch[0])
                p = spl[p].ch[0];
            splaying(p, now);
            connect(spl[now].ch[0], p, 0);
            root = p;
            spl[p].fa = 0;
            update(root);
        }
    }
    void change(int pos, int val)
    {
        del(find(pos));
        ins(val);
    }
    void build(int l, int r, int fa)
    {
        ins(-2147483647);
        ins(2147483647);
        fir(i, l, r)
            ins(a[i]);
    }
    int getnum(int rank)
    {
        int now = root;//rank从1开始
        while (now)
        {
            int lsize = spl[spl[now].ch[0]].size;
            if (lsize + 1 <= rank && rank <= lsize + spl[no
w].cnt)
            {
                //  splaying(now, 0);
                break;
            }
            if (rank <= lsize)
                now = spl[now].ch[0];
            else
            {
                rank -= lsize + spl[now].cnt;
                now = spl[now].ch[1];
            }
        }
        return now;
    }
    int getrank(int val)
    {
        if (val == 2)
            int x = 1;
        int now = root;
        int rank = 0;
        while (now)
        {
            int& l = spl[now].ch[0], &r = spl[now].ch[1];
            if (val < spl[now].val)
                now = spl[now].ch[0];
            else if (spl[now].val < val)
                rank += spl[l].size + spl[now].cnt, now = r;
            else
            {
                rank += spl[l].size;
                break;
            }
        }
        return rank - 1;
    }
    vector<int> ans;
    void dfs(int now)
    {
        //  push_down(now);
        if (spl[now].ch[0])
            dfs(spl[now].ch[0]);
        //  if (spl[now].val >= 1 && spl[now].val <= n)
        ans.push_back(spl[now].val);
        if (spl[now].ch[1])
            dfs(spl[now].ch[1]);
    }
    void show()
    {
        ans.clear();
        dfs(root);
        for (int i = 0; i < ans.size() - 1; i++)
            printf("%d ", ans[i]);
        printf("%d\n", ans.back());
    }
};
#define left (i<<1)
#define right (i<<1|1)
struct node
{
    int l;
    int r;
    Splay s;
}seg[maxn * 4];
void build(int l, int r, int i)
{
    seg[i].l = l;
    seg[i].r = r;
    seg[i].s.build(l, r, 0);
    // seg[i].s.show();
    // cout<<l<<' '<<r<<endl;
    if (l == r)return;
    int mid = (l + r) >> 1;
    build(l, mid, left);
    build(mid + 1, r, right);
}
int query_rank(int l, int r, int i, int val)
{
    int ans = 0;
    if (l <= seg[i].l&&seg[i].r <= r)
    {
        return seg[i].s.getrank(val);
    }
    int mid = (seg[i].l + seg[i].r) >> 1;
    if (l <= mid)ans += query_rank(l, r, left, val);
    if (r > mid)ans += query_rank(l, r, right, val);
    return ans;
}
int query_val(int L, int R, int k)
{
    if (k > R - L + 1)return 2147483647;
    if (k <= 0)return -2147483647;
    int l = 0, r = 1e8;
    while (l < r)
    {
        int mid = (l + r + 1) >> 1;
        if (query_rank(L, R, 1, mid) + 1 <= k)l = mid;
        else r = mid - 1;
        //      cout<<mid<<':'<<query_rank(L,R,1,mid)+1<<end
l;
    }
    return l;
}
void change(int l, int i, int val)
{
    seg[i].s.change(a[l], val);
    if (seg[i].l == seg[i].r)return;
    int mid = (seg[i].l + seg[i].r) >> 1;
    if (l <= mid)change(l, left, val);
    else change(l, right, val);
}
int getpre(int l, int r, int val)
{
    int pos = query_rank(l, r, 1, val) + 1;
    return query_val(l, r, pos - 1);
}
int getnext(int l, int r, int val)
{
    int pos = query_rank(l, r, 1, val + 1) + 1;
    // cout<<val<<':'<<pos<<endl;
    return query_val(l, r, pos);
}
int main()
{
    int debug = 0;
    if (debug)
    {
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
    }
    scanf("%d%d", &n, &m);
    fir(i, 1, n)scanf("%d", &a[i]);
    build(1, n, 1);
    while (m--)
    {
        int ch;
        scanf("%d", &ch);
        int l, r, pos, k;
        if (ch != 3)scanf("%d%d%d", &l, &r, &k);
        else scanf("%d%d", &pos, &k);//修改
        if (ch == 1)printf("%d\n", query_rank(l, r, 1, k) +
1);//求区间数的排名
        else if (ch == 2)printf("%d\n", query_val(l, r,
k));//求区间第k个数
        else if (ch == 3)change(pos, 1, k), a[pos] = k;
        else if (ch == 4)printf("%d\n", getpre(l, r, k));//
求前驱
        else if (ch == 5)printf("%d\n", getnext(l, r, k));//
求后继
    }
}
```

扫描线求周长

```
vector<int> y;
namespace s
{
#define left (i<<1)
#define right (i<<1|1)
struct node
{
    int l;
    int r;
    ll sum;
    int num;
    int leftc,rightc;
    int add;
} seg[maxn<<2];
void build(int l,int r,int i)
{
    seg[i].l=l,seg[i].r=r;
    seg[i].sum=seg[i].add=seg[i].num=0;
    seg[i].leftc=seg[i].rightc=0;
    if(l==r)return;
    int mid=(l+r)>>1;
    build(l,mid,left);
    build(mid+1,r,right);
}
void push_up(int i)
{
    //扫描线中对应的y1,y2相等，不会出现减的一段区间内无值情况
    if(seg[i].add)
    {
        seg[i].sum=y[seg[i].r]-y[seg[i].l-1];
        seg[i].num=1;
        seg[i].leftc=seg[i].rightc=1;
    }
    else
    {
        seg[i].sum=seg[left].sum+seg[right].sum;
        seg[i].leftc=seg[left].leftc;
        seg[i].rightc=seg[right].rightc;
        seg[i].num=seg[left].num+seg[right].num-(seg[left].r
ightc&&seg[right].leftc);
    }
}
void query(int l,int r,int i,int add)
{
    //维护区间覆盖
    if(l<=seg[i].l&&seg[i].r<=r)
    {
        seg[i].add+=add;
        push_up(i);
        return;
    }
    int mid = (seg[i].l + seg[i].r) >> 1;
    if (l <= mid)
        query(l, r, left,add);
    if (r > mid)
        query(l, r, right, add);
    push_up(i);
}
};
struct node
{
    int x;
    int y1;
    int y2;
    int flag;
    bool operator<(const node & A)
    {
        return x<A.x||x==A.x&&y1<A.y1||x==A.x&&y1==A.y1&&y2<
A.y2;
    }
};
int get_id(int x)
{
    return lower_bound(y.begin(), y.end(), x)-y.begin() + 1;
}
vector<node> a;
int32_t main()
{
 //   freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);
    int n;
    cin>>n;
    fir(i,1,n)
    {
        int x1,x2,y1,y2;
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        a.push_back({x1,y1,y2,1});
        a.push_back({x2,y1,y2,-1});
        y.push_back(y1),y.push_back(y2);
    }
    sort(y.begin(),y.end());
    y.erase(unique(y.begin(),y.end()),y.end());
    sort(a.begin(),a.end());
    s::build(1,y.size(),1);
    ll ans=0;
    for(int i=0; i<a.size(); i++)
    {
        int l = get_id(a[i].y1);
        int r = get_id(a[i].y2)-1;//覆盖问题细节
     //   cout<<a[i].x<<' '<<a[i].y1<<' '<<a[i].y2<<endl;
        ll last=s::seg[1].sum;
        //cout<<l<<' '<<r<<endl;
        s::query(l, r, 1, a[i].flag);
        ll now =s::seg[1].sum;
        ans+=abs(now-last);
    //    cout<<abs(now-last)<<endl;
       // cout<<s::seg[1].num<<endl;
       if(i+1<a.size())
        ans+=s::seg[1].num*2*(a[i+1].x-a[i].x);
    }
    cout<<ans<<endl;
}
```

扫描线求面积

```cpp
vector<int> y;
namespace s
{
#define left (i<<1)
#define right (i<<1|1)
struct node
{
    int l;
    int r;
    ll sum;
    int add;
} seg[maxn<<2];
void build(int l,int r,int i)
{
    seg[i].l=l,seg[i].r=r;
    seg[i].sum=seg[i].add=0;
    if(l==r)return;
    int mid=(l+r)>>1;
    build(l,mid,left);
    build(mid+1,r,right);
}
void push_up(int i)
{
    //扫描线中对应的y1,y2相等，不会出现减的一段区间内无值情况
    int l=seg[i].l,r=seg[i].r;
    if(seg[i].add)
        seg[i].sum=y[r]-y[l-1];
    else
        seg[i].sum=seg[left].sum+seg[right].sum;
}
void query(int l,int r,int i,int add)
{//维护区间覆盖
    if(l<=seg[i].l&&seg[i].r<=r)
    {
        seg[i].add+=add;
        push_up(i);
        return;
    }
    int mid = (seg[i].l + seg[i].r) >> 1;
    if (l <= mid)
        query(l, r, left,add);
    if (r > mid)
        query(l, r, right, add);
    push_up(i);
}
};
struct node
{
    int x;
    int y1;
    int y2;
    int flag;
    bool operator<(const node & A)
    {
        return x<A.x;
    }
};
int get_id(int x)
{
    return lower_bound(y.begin(), y.end(), x)-y.begin() + 1;
}
vector<node> a;
int32_t main()
{
    int n;
    cin>>n;
    int cnt=0;
    fir(i,1,n)
    {
        int x1,x2,y1,y2;
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        a.push_back({x1,y1,y2,1});
        a.push_back({x2,y1,y2,-1});
        y.push_back(y1),y.push_back(y2);
    }
    sort(y.begin(),y.end());
    y.erase(unique(y.begin(),y.end()),y.end());
    sort(a.begin(),a.end());
    s::build(1,y.size(),1);
    ll ans=0;
    for(int i=0;i<a.size()-1;i++)
    {
        int l = get_id(a[i].y1);
        int r = get_id(a[i].y2)-1;//覆盖问题细节
        s::query(l, r, 1, a[i].flag);
        ans += s::seg[1].sum*(a[i + 1].x - a[i].x);;
    }
    cout<<ans<<endl;
}
```

可持久化线段树

```
struct node
{
    int l;
    int r;
    int sum;
};
vector<int> ans;
const int maxn = 2e5 + 50;
int a[maxn];
inline int getid(int x)
{
    return lower_bound(ans.begin(), ans.end(),x) - ans.begin
() + 1;
}
node t[maxn * 40];
int root[maxn + 50];
int cnt = 0;
void insert(int l, int r,int pre,int &now,int x)
{
    t[++cnt] = t[pre];
    now = cnt;
    t[now].sum++;
    if (l == r)
        return;
    int mid = (l + r) >> 1;
    if (x <= mid) insert(l, mid, t[pre].l, t[now].l, x);
    else insert(mid+1, r, t[pre].r, t[now].r, x);
}
int query(int l, int r, int L, int R, int k)
{
    if (l == r)
        return l;
    int mid = (l + r) >> 1;
    int num = t[t[R].l].sum - t[t[L].l].sum;
    if (k <= num)
        return query(l, mid, t[L].l, t[R].l, k);
    else
        return query(mid + 1, r, t[L].r, t[R].r, k - num);
}
int main()
{
    int n, m;
    while (cin >> n >> m)
    {
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            ans.push_back(a[i]);
        }
        sort(ans.begin(), ans.end());
        ans.erase(unique(ans.begin(), ans.end()), ans.end
());
        int size = ans.size();
        for (int i = 1; i <= n; i++)
            insert(1,size,root[i-1],root[i],getid(a[i]));
        for (int i = 1; i <= m; i++)
        {
            int L,R,k;
            scanf("%d%d%d",&L,&R,&k);
            printf("%d\n",ans[query(1, size, root[L - 1], ro
ot[R], k)-1]);
        }
    }
}
```

主席树区间修改(无up和down)

```
struct node
{
    int l;
    int r;
    ll sum;
    ll lazy;
};
node t[maxn * 30];
int cnt = 0;
int a[maxn];
int root[maxn];
void build(int l, int r, int &now)
{
    now = ++cnt;
    t[now].lazy = 0;
    if (l == r)
    {
        t[now].sum = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, t[now].l);
    build(mid + 1, r, t[now].r);
    t[now].sum = t[t[now].l].sum + t[t[now].r].sum;
}
void insert(int &now, int l, int r, int pre, int L, int R, i
nt val)
{
    t[++cnt] = t[pre];
    now = cnt;
    t[now].sum += 1ll * val * (R - L + 1);
    if (l == L && r == R)
    {
        t[now].lazy += val;
        return;
    }
    int mid = (l + r) >> 1;
    if (R <= mid)
        insert(t[now].l, l, mid, t[pre].l, L, R, val);
    else if (L > mid)
        insert(t[now].r, mid + 1, r, t[pre].r, L, R, val);
    else
    {
        insert(t[now].l, l, mid, t[pre].l, L, mid, val);
        insert(t[now].r, mid + 1, r, t[pre].r, mid + 1, R, v
al);
    }
}
ll query(int now, int l, int r, int L, int R)
{
    if (l == L && r == R)
        return t[now].sum;
    ll ans = 1ll * (R - L + 1) * t[now].lazy;
    int mid = (l + r) >> 1;
    if (R <= mid)
        return ans + query(t[now].l, l, mid, L, R);
    else if (L > mid)
        return ans + query(t[now].r, mid + 1, r, L, R);
    else
        return ans + query(t[now].l, l, mid, L, mid) + query
(t[now].r, mid + 1, r, mid + 1, R);
}
int main()
{
    //input();
    int n, m;
    while (cin >> n >> m)
    {
        fir(i, 1, n) scanf("%d", &a[i]);
        int rootnum = 0;
        build(1, n, root[0]);
        while (m--)
        {
            char ch[5];
            scanf("%s", ch);
            int l, r;
            if (ch[0] == 'B')
                scanf("%d", &rootnum);
            else
            {
                scanf("%d%d", &l, &r);
                if (ch[0] == 'Q')
                    printf("%lld\n", query(root[rootnum], 1,
 n, l, r));
                else
                {
                    int d;
                    scanf("%d", &d);
                    if (ch[0] == 'C')
                        insert(root[rootnum + 1], 1, n, root
[rootnum], l, r, d), rootnum++;
                    else
                        printf("%lld\n", query(root[d], 1,
 n, l, r));
                }
            }
        }
    }
}
```

树状数组

```
int n;
int tree[maxn];
int lowbit(int n) { return n & (-n); }
```

```
void add(int x, int now)
{
    while (x <= n)
    {
        tree[x] += now;
        x += lowbit(x);
    }
}
int ask(int x)
{
    int ans = 0;
    while (x)
    {
        ans += tree[x];
        x -= lowbit(x);
    }
    return ans;
}
```

求区间出现的数字个数(树状数组)

```
int n;
int tree[maxn];
int lowbit(int n) { return n & (-n); }
void add(int x, int now)
{
    while (x <= n)
    {
        tree[x] += now;
        x += lowbit(x);
    }
}
int ask(int x)
{
    int ans = 0;
    while (x)
    {
        ans += tree[x];
        x -= lowbit(x);
    }
    return ans;
}
int a[maxn];
struct node
{
    int l;
    int r;
    int id;
    bool operator<(const node &A) const
    {
        return r < A.r || r == A.r && l < A.l;
    }
};
node b[maxn];
map<int, int> mp;
int ans[maxn];
int main()
{
    cin >> n;
    fir(i, 1, n) scanf("%d", &a[i]);
    int m;
    cin >> m;
    fir(i, 1, m) scanf("%d%d", &b[i].l, &b[i].r), b[i].id =
 i;
    sort(b + 1, b + m + 1);
    int pos = 1;
    fir(i, 1, m)
    {
        for (int j = pos; j <= b[i].r; j++)
        {
            int x = a[j];
            if(mp.count(x))
            add(mp[x],-1);
            mp[x]=j;
            add(j,1);
        }
        pos=b[i].r+1;
    /*  cout<<b[i].r<<':'<<ask(b[i].r)<<endl;
        cout<<b[i].l-1<<':'<<ask(b[i].l-1)<<endl; */
        ans[b[i].id]=ask(b[i].r)-ask(b[i].l-1);
    }
    fir(i,1,m)
    printf("%d\n",ans[i]);
}
```

普通莫队

```
namespace IO
{
    template <typename T>
    inline void w(T x)
    {
        if (x > 9)
            w(x / 10);
        putchar(x % 10 + 48);
    }
    template <typename T>
    inline void write(T x, char c)
    {
        if (x < 0)
            putchar('-'), x = -x;
        w(x);
        putchar(c);
    }
    template <typename T>
    inline void read(T &x)
    {
        x = 0;
        T f = 1;
        char c = getchar();
        for (; !isdigit(c); c = getchar())
            if (c == '-')
                f = -1;
        for (; isdigit(c); c = getchar())
            x = (x << 1) + (x << 3) + (c ^ 48);
        x *= f;
    }
}; // namespace IO
const int maxn=1e6+10;
struct node{
    int l,r;
    int id;
    int l1;
}q[maxn];
int a[maxn];
bool cmp(node a,node b){//奇偶排序
    return a.l1==b.l1?(a.l1&1)?a.r<b.r:a.r>b.r:a.l1<b.l1;
}
int cnt[maxn];
int now=0;
void add(int i){//加操作
    int x=a[i];
    if(!cnt[x])now++;
    cnt[x]++;
}
void del(int i){//减操作
    int x=a[i];
    cnt[x]--;
    if(!cnt[x])now--;
}
int ans[maxn];
void file_iuput(bool flag){
    if(flag){
        freopen("in.txt","r",stdin);
        freopen("out.txt","w",stdout);
    }
}
int main(){
    file_iuput(false);
    int n,m;
    scanf("%d",&n);
    fir(i,1,n)scanf("%d",&a[i]);
    int s=sqrt(n);//莫队分块
    scanf("%d",&m);
    fir(i,1,m)
    {
        IO::read(q[i].l),IO::read(q[i].r);
        q[i].l1=q[i].l/s,q[i].id=i;
    }
    sort(q+1,q+m+1,cmp);
    int l=1,r=0;
    for(int i=1;i<=m;i++){
        while(r<q[i].r)add(++r);
        while(r>q[i].r)del(r--);
        while(l>q[i].l)add(--l);
        while(l<q[i].l)del(l++);
        ans[q[i].id]=now;
    }
    fir(i,1,m)
    IO::write(ans[i],'\n');
}
```

日期

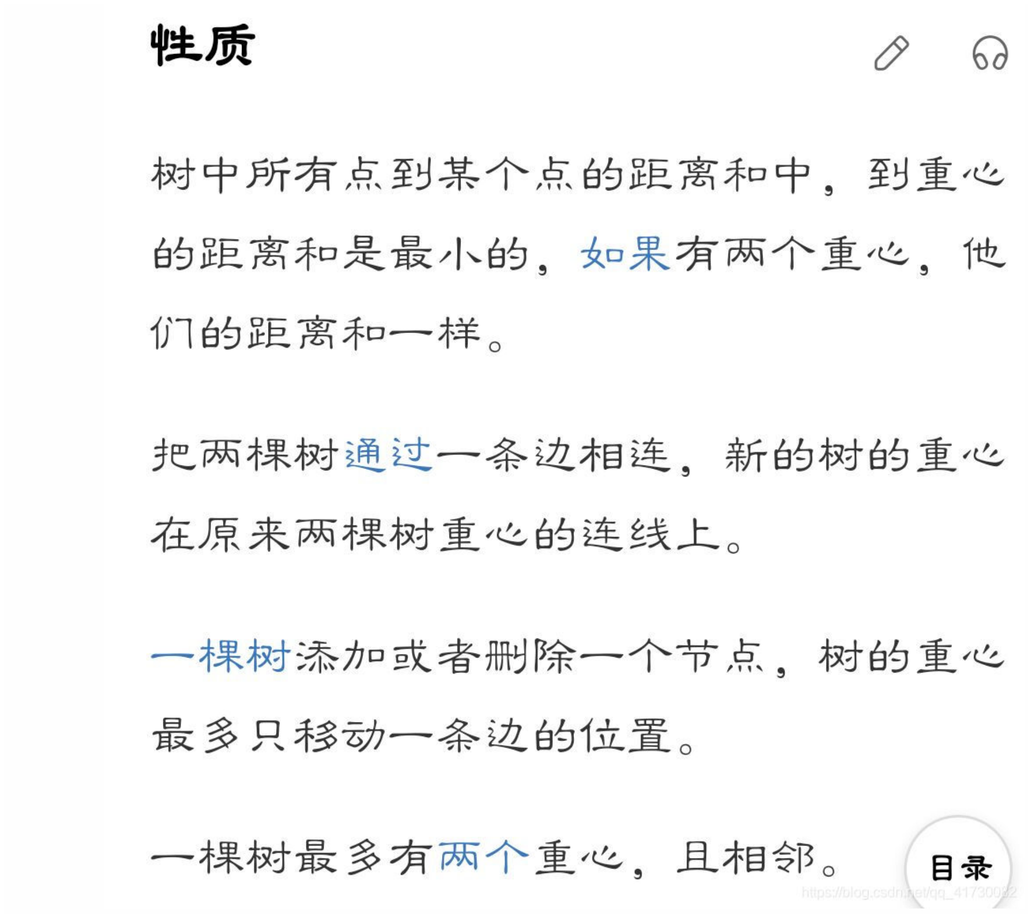
```
struct Data
{
    int a[2][20]={{0,31,28,31,30,31,30,31,31,30,31,30,31}
,{0,31,29,31,30,31,30,31,31,30,31,30,31}};
    int year;
    int month;
    int day;
    bool flag;
    bool isrun()
    {
        if(year%400==0||((year%100!=0)&&(year%4==0)))
           return true;
           return false;
    }
    Data(int _year,int _month,int _day)
    {
        year=_year,month=_month,day=_day;
        flag=isrun();
    }
    bool operator<(const Data& A)const
    {
        if(year<A.year)return true;
        if(year==A.year&&month<A.month)return true;
        if(year==A.year&&month==A.month&&day<A.day)return tr
ue;
           return false;
    }
    void add()
    {
        day++;
        if(day>a[flag][month])
            day=1,month++;
        if(month>12)
        month=1,year++,flag=isrun();;
    }
    bool operator==(const Data A)const
    {
        return year==A.year&&month==A.month&&day==A.day;
    }
};
```

带修莫队

```
const int maxn=1e6+10;
int belong[maxn];
struct node{
    int l,r;
    int id;
    int t;
}q[maxn];
bool cmp(node & a, node & b) {
    return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belon
g[b.l] : ((belong[a.r] ^ belong[b.r]) ? belong[a.r] < belong
[b.r] : a.t < b.t);
}
int cnt[maxn];
int now=0;
int k,cntq=0,cntr=0;
struct cnode{
    int pos;
    int val;
}c[maxn];
int a[maxn];
void add(int i){
    int x=a[i];
    if(++cnt[x]==1)now++;
}
void del(int i){
    int x=a[i];
    if(--cnt[x]==0)now--;
}
void work(int ct,int i)
{
    if(c[ct].pos>=q[i].l&&c[ct].pos<=q[i].r)
    {
        int x=a[c[ct].pos];
        if(--cnt[x]==0)now--;
        x=c[ct].val;
        if(++cnt[x]==1)now++;
    }
    swap(c[ct].val,a[c[ct].pos]);
}
int ans[maxn];
void file_iuput(){
    freopen("in.txt","r",stdin);
    freopen("out.txt","w",stdout);
}
int main(){
  //  file_iuput();
```

```
    int n,m;
    scanf("%d%d",&n,&m);
    int s=pow(n,2.0/3.0);
    fir(i,1,n)scanf("%d",&a[i]),belong[i]=(i-1)/s+1;
  //  getchar();
    fir(i,1,m)
    {
        char ch[10];
        scanf("%s",ch);
      //  cout<<ch<<' ';
        if(ch[0]=='Q'){
           IO::read(q[++cntq].l);
           IO::read(q[cntq].r);
           q[cntq].id=cntq;
           q[cntq].t=cntr;
       //     cout<<q[cntq].l<<' '<<q[cntq].r<<endl;
        }
        else
        {
            cntr++;
            IO::read(c[cntr].pos);
            IO::read(c[cntr].val);
       //      cout<<c[cntr].pos<<' '<<c[cntr].val<<endl;
        }
       // getchar();
    }
    sort(q+1,q+cntq+1,cmp);
    int l=1,r=0,t=0;
    fir(i,1,cntq)
    {
        while(r<q[i].r)add(++r);
        while(r>q[i].r)del(r--);
        while(l>q[i].l)add(--l);
        while(l<q[i].l)del(l++);
        while(t<q[i].t)work(++t,i);
        while(t>q[i].t)work(t--,i);
        ans[q[i].id]=now;
    }
    fir(i,1,cntq)
    IO::write(ans[i],'\n');
}
```

树的重心的性质



性质

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。

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。

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。

一棵树最多有两个重心，且相邻。

DSU On Tree

```
int c[maxn];
vector<int> a[maxn];
int son[maxn],siz[maxn],currentson;
void dfs1(int u,int fa)
{
    siz[u]=1;son[u]=0;
    for(auto v:a[u])
    {
        if(v!=fa)
        {
            dfs1(v,u);
            siz[u]+=siz[v];
            if(siz[son[u]]<siz[v])
            son[u]=v;
        }
    }
}
ll ans[maxn];
ll cnt[maxn];
ll sum,maxnum;
void updata(int u,int fa,int val)
{
    cnt[c[u]]+=val;
    if(maxnum<cnt[c[u]])
    sum=c[u],maxnum=cnt[c[u]];
    else if(maxnum==cnt[c[u]])
```

```
        sum+=c[u];
        for(auto v:a[u])
        {
            if(v!=fa&&v!=currentson)
            updata(v,u,val);
        }
    }
    void dfs2(int u,int fa,int flag)
    {
        for(auto v:a[u])
        {
            if(v!=fa&&v!=son[u])
            {
                dfs2(v,u,0);//跑轻儿子
            }
        }
        if(son[u])dfs2(son[u],u,1),currentson=son[u];//重儿子统
    计，不删除贡献
        updata(u,fa,1);//统计轻儿子
        currentson=0;
        ans[u]=sum;
        if(!flag)updata(u,fa,-1),sum=0,maxnum=0;//轻儿子删除贡献
    }
    int main()
    {
        //input();
        int n;
        cin>>n;
        fir(i,1,n)scanf("%d",&c[i]);
        fir(i,1,n-1)
        {
            int u,v;
            scanf("%d%d",&u,&v);
            a[u].push_back(v);
            a[v].push_back(u);
        }
        dfs1(1,0);
       // fir(i,1,n)cout<<i<<':'<<son[i]<<endl;
        dfs2(1,0,0);
        for(int i=1;i<n;i++)printf("%lld ",ans[i]);
        printf("%lld\n",ans[n]);
        return 0;
    }
```

点分治

```
    int n,k;
    struct node
    {
        int v;
        int w;
    };
```

```
vector<node> a[maxn];
int siz[maxn],maxp[maxn];//子树大小和当前点为重心是最多的子树
int vis[maxn];
int rt=0;
void getsize(int u,int fa)//子树大小
{
    siz[u]=1;
    for(auto it:a[u]){
        int v=it.v;
        if(v==fa||!vis[v])
        continue;
        getsize(v,u),siz[u]+=siz[v];
    }
}
//找重心
void getroot(int u,int fa,int sum)
{
    siz[u]=1,maxp[u]=0;
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v]||v==fa)continue;
        getroot(v,u,sum);
        siz[u]+=siz[v];
        maxp[u]=max(maxp[u],siz[v]);
    }
    maxp[u]=max(maxp[u],sum-siz[u]);
    if(maxp[u]<maxp[rt])rt=u;
}
//求各点到重心的距离
int q[maxn],qt,pt,p[maxn];
void getdist(int u,int fa,int d)
{
    p[++pt]=d;
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v]||v==fa)continue;
        getdist(v,u,d+it.w);
    }
}
//处理
int cal()
{
}
int ans;
void solve(int u)
{
    vis[u]=false;
    /* 处理 */
    getsize(u,0);
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v])continue;
```

```
            rt=0;
            getroot(v,u,siz[v]);
            solve(rt);
        }
}
int main()
{
   // input();
    while(cin>>n>>k)
    {
        if(!n&&!k)return 0;
        fir(i,1,n)a[i].clear();
        fir(i,1,n)vis[i]=true;
        ans=0;
        fir(i,2,n)
        {
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            u++,v++;
            a[u].push_back({v,w});
            a[v].push_back({u,w});
        }
        maxp[0]=MAX_INF;//为0，重心点初值
        rt=0;
        getroot(1,0,n);
        solve(rt);
        printf("%d\n",ans);
    }
}
```

求多少个路径数量不超过K

```
const int maxn = 1e4+ 10;
int n,k;
struct node
{
    int v;
    int w;
};
vector<node> a[maxn];
int siz[maxn],maxp[maxn];//子树大小和当前点为重心是最多的子树
int vis[maxn];
int rt=0;
void getsize(int u,int fa)//子树大小
{
    siz[u]=1;
    for(auto it:a[u]){
        int v=it.v;
        if(v==fa||!vis[v])
        continue;
        getsize(v,u),siz[u]+=siz[v];
    }
}
```

```
//找重心
void getroot(int u,int fa,int sum)
{
    siz[u]=1,maxp[u]=0;
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v]||v==fa)continue;
        getroot(v,u,sum);
        siz[u]+=siz[v];
        maxp[u]=max(maxp[u],siz[v]);
    }
    maxp[u]=max(maxp[u],sum-siz[u]);
    if(maxp[u]<maxp[rt])rt=u;
}
//求各点到重心的距离
int q[maxn],qt,pt,p[maxn];
void getdist(int u,int fa,int d)
{
    p[++pt]=d;
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v]||v==fa)continue;
        getdist(v,u,d+it.w);
    }
}
//处理
int cal(int a[],int cnt)
{
    sort(a+1,a+cnt+1);
    int sum=0;
    int r=cnt;
    fir(i,1,cnt)
    {
        while(r>i&&a[i]+a[r]>k)r--;
        if(r<=i)break;
        sum+=r-i;
    }
    return sum;
}
int ans;
void solve(int u)
{
    vis[u]=false;
    getsize(u,0);
    qt=0;
    for(auto it:a[u])
    {
        int v=it.v;
        if(!vis[v])continue;
        pt=0;
        getdist(v,u,it.w);
        //对每个块里，减去块里选两个的
        ans-=cal(p,pt);
```

```
            fir(i,1,pt)q[++qt]=p[i];
        }
        //整体选2个
        ans+=cal(q,qt);
        //以重心为根的点
        fir(i,1,qt)if(q[i]<=k)ans++;
        //cout<<u<<':'<<ans<<endl;
        for(auto it:a[u])
        {
            int v=it.v;
            if(!vis[v])continue;
            rt=0;
            getroot(v,u,siz[v]);
            solve(rt);
        }
    }
    int main()
    {
       // input();
        while(cin>>n>>k)
        {
            if(!n&&!k)return 0;
            fir(i,1,n)a[i].clear();
            fir(i,1,n)vis[i]=true;
            ans=0;
            fir(i,2,n)
            {
                int u,v,w;
                scanf("%d%d%d",&u,&v,&w);
                u++,v++;
                a[u].push_back({v,w});
                a[v].push_back({u,w});
            }
            maxp[0]=MAX_INF;//为0，重心点初值
            rt=0;
            getroot(1,0,n);
            solve(rt);
            printf("%d\n",ans);
        }
    }
```

2.字符串

双HASH

```
    namespace stringHash{
    #define Hash pair<int,int>
        const Hash MOD = {998244353,1004535809};
        const Hash BASE = {233,137};
        Hash operator + (const Hash A, const Hash B){ return mak
    e_pair((A.first+B.first)%MOD.first,(A.second+B.second)%MOD.s
    econd); }
        Hash operator - (const Hash A, const Hash B){ return mak
    e_pair((A.first-B.first+MOD.first)%MOD.first,(A.second-B.sec
    ond+MOD.second)%MOD.second); }
        Hash operator * (const Hash A, const Hash B){ return mak
    e_pair(1ll*A.first*B.first%MOD.first,1ll*A.second*B.second%M
    OD.second); }
        Hash operator * (const Hash A, const int x){ return make
    _pair(1ll*A.first*x%MOD.first, 1ll*A.second*x%MOD.second); }
    }
    using namespace stringHash;
```

Trie

```
int t[maxn][26],cnt[maxn],idx=0;
char str[maxn];
void insert()
{
    int p=0;
    int len=strlen(str);
    for(int i=0;i<len;i++)
    {
        int &s=t[p][str[i]-'a'];
        if(!s) s=++idx;
        p=s;
    }
    cnt[p]++;
}
int query()
{
    int res=0,p=0;
    int len=strlen(str);
    for(int i=0;i<len;i++)
    {
        p=t[p][str[i]-'a'];
        if(p==0)
        break;
        res+=cnt[p];
    }
    return res;
}
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    while(n--)
    {
        scanf("%s",str);
        insert();
    }
    while(m--)
    {
        scanf("%s",str);
        printf("%d\n",query());
    }
}
```

AC自动机

```
namespace AC
{
const int maxn = 1e6+ 20, N = 1e5+20, M = 4e6 + 10;
//const int P=1000000007;
int trie[N][26];
int cnt=0;
int cntwork[N];
string getstr[N];
int fail[N];
void init()
{
    memset(trie,0,sizeof(trie));
    memset(cntwork,0,sizeof(cntwork));
    memset(fail,0,sizeof(fail));
    cnt=0;
}
void insert(string str)
{
    int now=0;
    for(auto ch:str)
    {
        int id=ch-'a';
        if(!trie[now][id])trie[now][id]=++cnt;
        now=trie[now][id];
    }
    cntwork[now]++;
    getstr[now]=str;
}
void getfail()
{
    queue<int> p;
    for(int i=0;i<26;i++)
    {
        if(trie[0][i])
        {
            p.push(trie[0][i]);
            fail[trie[0][i]]=0;
        }
    }
    while(!p.empty())
    {
        int now=p.front();
        p.pop();
        for(int i=0;i<26;i++)
        {
            if(trie[now][i])
            {
                fail[trie[now][i]]=trie[fail[now]][i];
                p.push(trie[now][i]);
            }
            else
            trie[now][i]=trie[fail[now]][i];
        }
    }
}
char s[maxn];
int match()
{
    int n=strlen(s+1);
    int now=0;
    int ans=0;
    fir(i,1,n)
    {
        now=trie[now][s[i]-'a'];
        for(int j=now;j;j=fail[j])
        {
            if(cntwork[j])
            ans++;
        }
    }
}
};
int main()
{
    int debug=0;
    if(debug)
    {
        freopen("in.txt","r",stdin);
        freopen("out.txt","w",stdout);
    }
    int n;
    while(scanf("%d",&n)&&n)
    {
        AC::init();
        fir(i,1,n)
        {
            string str;
            cin>>str;
            AC::insert(str);
        }
        AC::getfail();
        scanf("%s",AC::s+1);
        AC::match();
    }
}
```

AC自动机优化

```
namespace AC
{
    //const int P=1000000007;
    int sum[N];
    int trie[N][26];
    int cnt = 0;
    int cntwork[N];
    vector<int> a[N];
    int get[N];
    int fail[N];
    void init()
    {
        cnt = 0;
    }
    char str[maxn];
    void insert(int num)
    {
        int now = 0;
        int n=strlen(str+1);
        for (int i=1;i<=n;i++)
        {
            int id = str[i]-'a';
            if (!trie[now][id])
                trie[now][id] = ++cnt;
            now = trie[now][id];
        }
        cntwork[now]++;
        get[num] = now;
    }
    void getfail()
    {
        queue<int> p;
        for (int i = 0; i < 26; i++)
        {
            if (trie[0][i])
            {
                p.push(trie[0][i]);
                fail[trie[0][i]] = 0;
            }
        }
        while (!p.empty())
        {
            int now = p.front();
            p.pop();
            for (int i = 0; i < 26; i++)
            {
                if (trie[now][i])
                {
                    fail[trie[now][i]] = trie[fail[now]][i];
                    p.push(trie[now][i]);
                }
                else
                    trie[now][i] = trie[fail[now]][i];
            }
        }
    }
    char s[maxn];
    void match()
    {
        int n = strlen(s + 1);
        int now = 0;
        int ans = 0;
        fir(i, 1, n){
            now = trie[now][s[i] - 'a'];
            sum[now]++;
        }
        for(int i=1;i<=cnt;i++){
            a[fail[i]].push_back(i);
        }
    }
    void dfs(int u){
        for(auto v:a[u]){
            dfs(v);
            sum[u]+=sum[v];
        }
    }
}; // namespace A
```

KMP

```
char str[maxn];
int next_str[maxn];
int n;
void kmp()
{
    next_str[1] = 0;
    for (int i = 2, j = 0; i <= n; i++)
    {
        while (j > 0 && str[i] != str[j + 1])
            j = next_str[j];
        if (str[i] == str[j + 1])
            j++;
        next_str[i] = j;
    }
}
```

马拉车

```
namespace Manacher
{
    char s[maxn], str[maxn];
    int p[maxn];
    void build()
    {
        scanf("%s", str + 1);
        int n = strlen(str + 1);
        int cnt = 0;
        s[++cnt] = '+';
        s[++cnt] = '#';
        fir(i, 1, n) s[++cnt] = str[i], s[++cnt] = '#';
        s[++cnt] = '!';
    }
    void solve()
    {
        int n=strlen(s+1),id=1,mx=1;
        for(int i=1;i<n;i++)
        {
            if(i<mx)
            p[i]=min(mx-i,p[id*2-i]);
            else p[i]=1;
            while(s[i-p[i]]==s[i+p[i]])
            p[i]++;
            if(mx<i+p[i])
            id=i,mx=i+p[i];
        }
    }
    void show()
    {
        int n=strlen(s+1),ans=0;
        fir(i,1,n-1)
        ans=max(ans,p[i]-1);
        cout<<ans<<endl;
    }
} // namespace Manacher
int main()
{
   // input();
    Manacher::build();
    Manacher::solve();
    Manacher::show();
}
```

3.数论

中国剩余定理

```
const int N = 1e5 + 5;
void ex_gcd(ll a, ll b, ll &d, ll &x, ll &y){
    if(b == 0){
        d = a,x = 1,y = 0;
        return;
    }
    ex_gcd(b, a % b, d, y, x);
    y -= x * (a / b);
}
inline ll mull(ll a, ll b, ll p) {
    return (a * b - (ll)((long double)a / p * b) * p + p) %
 p;
}
ll ex_china(ll a[], ll m[], int n){//x = a(mod m)
    ll M = m[1];
    ll ans = a[1];
    ll d, x, y;
    for(int i = 2; i <= n; i++){
        ll c = ((a[i] - ans) % m[i] + m[i]) % m[i];
        ex_gcd(M, m[i], d, x, y);
        if(c % d)return -1;//不存在解的情况
        ll mod = m[i] / d;
        x = mull(x, c / d, mod);
        ans += x * M;
        M *= mod;
        ans = (ans % M + M) % M;
    }
    return ans > 0? ans : ans + M;//注意ans是M倍数时输出M
}
int main(){
    int n;
    ll a[N], m[N];
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%lld%lld", &m[i], &a[i]); // 线模再余数
    }
    printf("%lld\n", ex_china(a, m, n));
    return 0;
}
```

n个直线分割平面

一 首先考虑 n条直线最多把平面分成an部分
于是a0=1 a1=2 a2=4
对于已经有n条直线 将平面分成了最多的an块
那么加一条直线 他最多与前n条直线有n个交点 于是被它穿过的区域都被一分为二 那么增加的区域数就是穿过的区域数 也就是这条直线自身被分成的段数 就是n+1 故a(n+1)=an+n+1
an=n+(n-1)+...+2+a1=n(n+1)/2 +1
二 再考虑n个平面最多把空间分成bn个部分

n个平面分割空间

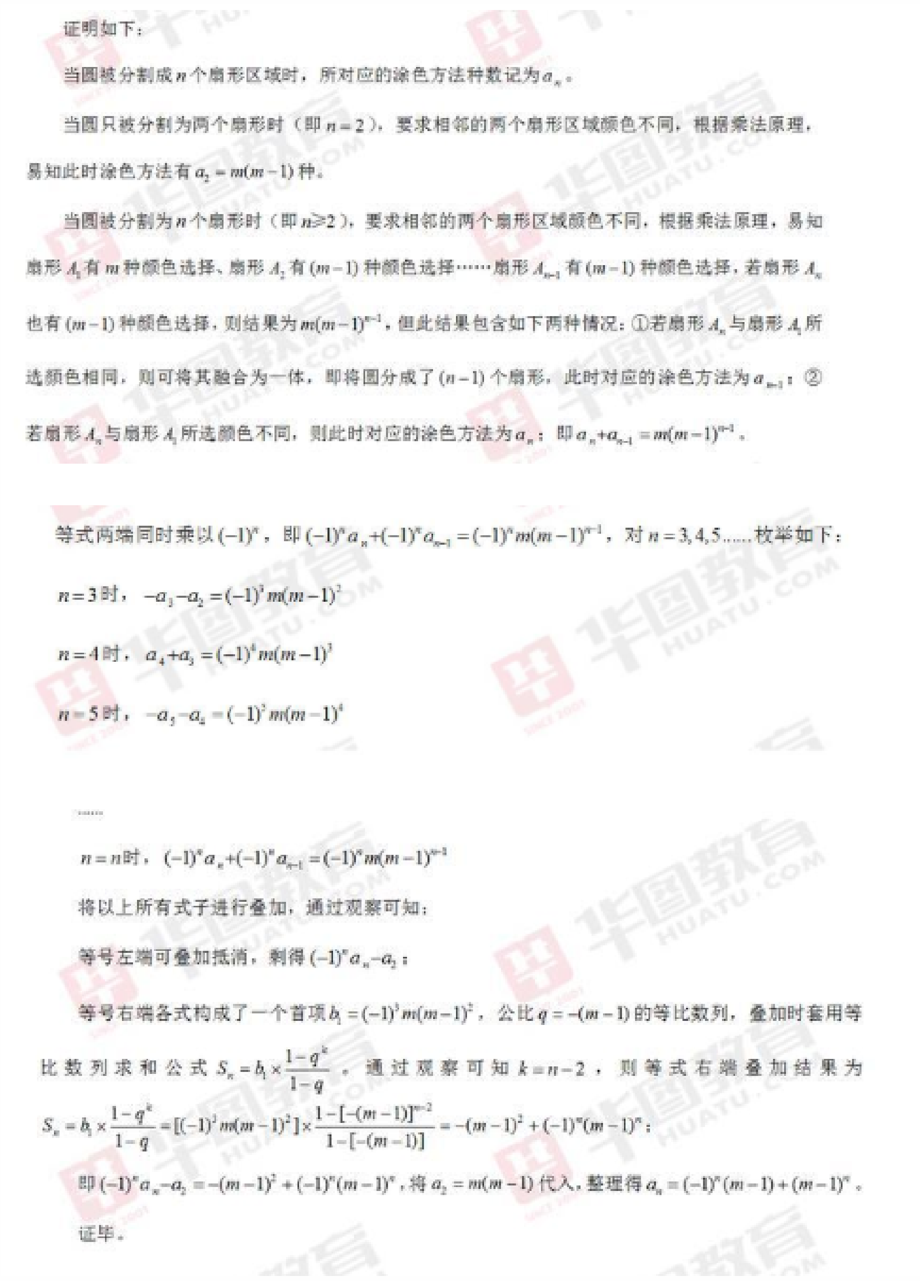
于是b0=1 b1=2 b2=4
对于已经有n个平面 将空间分成了最多的bn块
那么加入一个平面 它最多与每个平面相交 在它的上面就会得到至多n条交线
同时被它穿过的空间区域也被它一分为二 那么增加的区域数仍旧是它穿过的区域数 也就是这个平面自身被直线分割成的块数 就是an

于是b(n+1)=bn+an
bn=a(n-1)+b(n-1)=...=a(n-1)+a(n-2)+...+a1+b1
=(n-1)n/2 +(n-2)(n-1)/2+...+1*(1+1)/2+n+2
=求和[1方到(n-1)方]/2 + 求和[1到(n-1)]/2 +n+1
=n(n-1)(2n-1)/12 +n(n-1)/4 +n+1
=n(n+1)(n-1)/6 +n+1
=(n^3+5n+6)/6

3.1 环形涂色问题:

一个环形的花坛（不包括中心，中心可能建喷泉 - - ! ）分成n块

然后有m种颜色的花可供选取，要求相邻区域颜色不能相同，共有多少种方法？



证明如下：

当圆被分割成 n 个扇形区域时，所对应的涂色方法种数记为 a_n。

当圆只被分割为两个扇形时（即 $n=2$），要求相邻的两个扇形区域颜色不同，根据乘法原理，易知此时涂色方法有 $a_2=m(m-1)$ 种。

当圆被分割为 n 个扇形时（即 $n\geq2$），要求相邻的两个扇形区域颜色不同，根据乘法原理，易知扇形 A_1 有 m 种颜色选择、扇形 A_2 有 $(m-1)$ 种颜色选择……扇形 A_{n-1} 有 $(m-1)$ 种颜色选择，若扇形 A_n 也有 $(m-1)$ 种颜色选择，则结果为 $m(m-1)^{n-1}$，但此结果包含如下两种情况：①若扇形 A_n 与扇形 A_1 所选颜色相同，则可将其融合为一体，即将圆分成了 $(n-1)$ 个扇形，此时对应的涂色方法为 a_{n-1}；②若扇形 A_n 与扇形 A_1 所选颜色不同，则此时对应的涂色方法为 a_n；即 $a_n+a_{n-1}=m(m-1)^{n-1}$。

等式两端同时乘以 $(-1)^n$，即 $(-1)^n a_n+(-1)^n a_{n-1}=(-1)^n m(m-1)^{n-1}$，对 $n=3,4,5......$ 枚举如下：

$n=3$ 时，$-a_3-a_2=(-1)^3 m(m-1)^2$

$n=4$ 时，$a_4+a_3=(-1)^4 m(m-1)^3$

$n=5$ 时，$-a_5-a_4=(-1)^5 m(m-1)^4$

......

$n=n$ 时，$(-1)^n a_n+(-1)^n a_{n-1}=(-1)^n m(m-1)^{n-1}$

将以上所有式子进行叠加，通过观察可知：

等号左端可叠加抵消，剩得 $(-1)^n a_n-a_2$；

等号右端各式构成了一个首项 $b_1=(-1)^3 m(m-1)^2$，公比 $q=-(m-1)$ 的等比数列，叠加时套用等比数列求和公式 $S_n=b_1\times\frac{1-q^k}{1-q}$，通过观察可知 $k=n-2$，则等式右端叠加结果为 $S_n=b_1\times\frac{1-q^k}{1-q}=[(-1)^3 m(m-1)^2]\times\frac{1-[-(m-1)]^{n-2}}{1-[-(m-1)]}=-(m-1)^2+(-1)^n(m-1)^n$；

即 $(-1)^n a_n-a_2=-(m-1)^2+(-1)^n(m-1)^n$，将 $a_2=m(m-1)$ 代入，整理得 $a_n=(-1)^n(m-1)+(m-1)^n$。

证毕。

公式为:

$$An=(m-1)^n+(-1)^n*(m-1)$$

3.2线性基

3.2.1异或最大值

```
 namespace LinearBasis
{
    ll a[maxn];
    ll p[100];
    int n;
    void insert(ll x)
    {
        for(int i=62;i>=0;i--)
        {
            if(!((x>>i)&1))
            continue;
            if(!p[i])
            {
                p[i]=x;
                break;
            }
            x^=p[i];
        }
    }
    void build()
    {
        cin >> n;
       // memset(p,0,sizeof(p));
        fir(i, 1, n) scanf("%lld", &a[i]),insert(a[i]);
        ll ans=0;
        for(int i=62;i>=0;i--)
        if(!(ans>>i&1))
        ans^=p[i];
        cout<<ans<<endl;
    }
}; // namespace LinearBasis
```

```
namespace Linerabasis
{
    ll a[maxn];
    int n;
    int k = 1;
    void build()
    {
        k=1;
        for (int i = 62; i >= 0; i--)
        {
            for (int j = k; j <= n; j++)
            {
                if (a[j] >> i & 1)
                {
                    swap(a[j], a[k]);
                    break;
                }
            }
            if (!(a[k] >> i & 1))
                continue;
            for (int j = 1; j <= n; j++)
                if (j != k && a[j] >> i & 1)
                    a[j] ^= a[k];
            k++;
            if (k-1 == n)break;
        }
    }
    ll getmax()
    {
        ll ans=0;
        fir(i,1,k-1)
        ans^=a[i];
        return ans;
    }
    ll getkth(ll rank)
    {
        int cnt=k-1;
        if(cnt<n)
        rank--;
        if(!rank)return 0;
        if(rank>=mov(cnt))return -1;
        ll ans=0;
        for(int i=1;i<=cnt;i++)
        {
            if(rank>>(cnt-i)&1)
            ans^=a[i];
        }
        return ans;
    }
} // namespace Linerabasis
```

RHO

```
namespace Pollard_Rho{
        std::vector<ll> fac;
        std::vector<pair<ll, int> > res;
        inline ll llabs(ll x){
            return x > 0 ? x : -x;
        }
        inline ll mul(ll a, ll b, ll mod) {
            return (a * b - (ll)((long double)a * b / mod) * 
mod + mod) % mod;
        }
        inline ll pow(ll a, ll b, ll mod) {
            ll ans = 1 % mod; a = a % mod;
            while (b) {
                if (b & 1) ans = mul(ans, a, mod);
                a = mul(a, a, mod), b >>= 1;
            }
            return ans;
        }
        bool miller_rabin(ll n){
            if(n == 2)return 1;
            if(n < 2 || !(n & 1))return 0;
            ll u,t;
            for(t = 0,u = n - 1; !(u & 1); t++,u>>=1);//n-1
=u*2^t
            for(int i = 0; i < 10; i++){//10次试探
                ll a = rand() % (n - 1) + 1;//a∈[1,n)
                ll x = pow(a,u,n);
                for(int j = 0; j < t; j++){//二次试探
                    ll y = mul(x,x,n);
                    if(y == 1 && x != 1 && x != n - 1)
                        return 0;
                    x = y;//相当于让幂重新变回p - 1
                }
                if(x != 1)return 0;
            }
            return 1;
        }
        inline ll f(ll x, ll c, ll mod) {
            return (mul(x, x, mod) + c) % mod;
        }
        inline ll gcd(ll a, ll b) {
            return b == 0 ? a : gcd(b, a % b);
        }
        inline ll pollard_rho(ll x) {
            ll c = rand() % x, s = 0, t = 0, val = 1;
            for (int goal = 1; ; goal *= 2, s = t, val = 1)
{
                for (int step = 1; step <= goal; ++step) {
                    t = f(t, c, x);
                    val = mul(val, llabs(s - t), x);
                    if (step % 127 == 0) {
                        ll d = gcd(val, x);
                        if (d > 1) return d;
                    }
                }
                ll d = gcd(val, x);
                if (d > 1) return d;
            }
        }
        void solve(ll x) { //dfs获取质因子
            if(x == 1) return;
            if (miller_rabin(x)) {
                fac.push_back(x);
                return;
            }
            ll p = x;
            while (p == x) p = pollard_rho(x);
            while (x % p == 0) x /= p;
            solve(x), solve(p);
        }
        std::vector<ll> getFac(ll n){ // 获得质数的所有质因子
```

```
            fac.clear(); solve(n);
            sort(fac.begin(), fac.end());
            fac.erase(unique(fac.begin(), fac.end()), fac.en
d());
            return fac;
        }
        void getcnt(ll n){
            for(int i = 0; i < fac.size(); i++) {
                int cnt = 0;
                while(n % fac[i] == 0) cnt++, n /= fac[i];
                res.push_back(make_pair(fac[i], cnt));
            }
        }
        std::vector<pair<ll, int> > getPrimeFac(ll n){ // 获
取所有的质因子极其幂次
            fac.clear(); solve(n);
            sort(fac.begin(), fac.end());
            fac.erase(unique(fac.begin(), fac.end()), fac.en
d());
            res.clear(), getcnt(n);
            sort(res.begin(), res.end());
            return res;
        }
    };
```

欧拉函数

```
//单个数的欧拉函数
int eular(int n){
    int ans = n;
    for(int i = 2; i * i <= n; i++){
        if(n % i == 0){
            ans -= ans/i;
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1)ans -= ans / n;
    return ans;
}
//求1-n的
bool vis[N];
int pri[N], phi[N], tot;
void euler_table(int N){
    phi[1] = 1; vis[1] = vis[0] = 1;
    for(int i = 2 ;i <= N; i++){
        if(!vis[i]){
            pri[++tot] = i;
            phi[i] = i - 1;//i是素数时<i的所有大于0的数都与i互
质
        }
        for(int j = 1; j <= tot; j++){
            if(i * pri[j] > N)break;
```

```
                vis[i * pri[j]] = 1;//筛选掉i的倍数
                if(i % pri[j] == 0){//如果有一个质数是i的因子
                    phi[i * pri[j]] = phi[i] * pri[j];
                    break;
                }else phi[i * pri[j]] = phi[i] * phi[pri[j]];
            }
        }
}


int phi[N];
void phi_table(int N){
    for(int i = 0; i < N; i++) phi[i] = i;
    for(int i = 2; i < N; i++) {
        if(phi[i] == i) {
            for(int j = i; j < N; j += i) {
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}
```

欧拉降幂

$$b < \varphi(m), \quad a^b \equiv a^b (\bmod m)$$
$$b \geq \varphi(m), \quad a^b \equiv a^{b \bmod \varphi(m) + \varphi(m)} (\bmod m)$$

$b >= phi(m), a^b \equiv a^{(b \bmod phi(m)) + phi(m)} \pmod{phi(m)}$

②欧拉降幂

p不一定为质数且a和p不一定互质

$$b < \varphi(m), \quad a^b \equiv a^b (\bmod m)$$
$$b \geq \varphi(m), \quad a^b \equiv a^{b \bmod \varphi(m) + \varphi(m)} (\bmod m)$$

$a^{a^{a^{a}}}$,b次幂，最后膜p

线性筛

```
namespace Prime
{
    int v[maxn], prime[maxn];
    int m;
    void primes(int n)
    {
        memset(v, 0, sizeof(v));
        m = 0;
        for (int i = 2; i <= n; i++)
        {
            if (v[i] == 0)
            {
                v[i] = i;
                prime[++m] = i;
            }
            for (int j = 1; j <= m; j++)
            {
                if (prime[j] > v[i] || prime[j] > n / i)
                    break;
                v[i * prime[j]] = prime[j];
            }
        }
    }
};
```

线性求逆元+组合数

```
inv[1]=1;
for(int i=2;i<=n;++i)
  inv[i]=MOD-(long long)MOD/i*inv[MOD%i]%MOD;


ll fac[maxn], invfac[maxn], t[maxn];
void init(int n, ll MOD)
{ // 线性求[1, n]的组合数和逆元
    fac[0] = 1;
    mod = MOD;
    for (int i = 1; i <= n; i++)
        fac[i] = fac[i - 1] * i % mod;
    invfac[n] = qpow(fac[n], mod - 2, mod);
    for (int i = n; i >= 1; i--)
        invfac[i - 1] = invfac[i] * i % mod;
}
ll C(ll n, ll m)
{
    return n >= m ? fac[n] * invfac[n - m] % mod * invfac[m]
 % mod : 0;
}
ll lucas(ll n,ll m)
{
    if(n<mod&&m<mod)return C(n,m);
    return C(n%mod,m%mod)*lucas(n/mod,m/mod)%mod;
}
```

求组合数所有奇数项

$(a+b)^n+(-a+b)^n=2(C_n^0a^0b^n+C_n^2a^2b^{n-2}...C_n^na^nb^0)$

$(a+b)^n-(-a+b)^n=2(C_n^1a^1b^{n-1}+C_n^3a^3b^{n-3}...C_n^na^nb^0)$

求$\sum_{i=0}^{N}\binom{N}{i}$中所有i为奇数的项

组合数公式

$$\binom{N}{i} = \binom{N-1}{i} + \binom{N-1}{i-1}$$

$$C_n^m = C_n^{n-m},\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$$

令 $r-l=m$，则:

$$\boxed{\begin{aligned}\text{原式} &= \sum_{k=1}^{n} C_{m+k}^{k} \\ &= C_{m+1}^{1} + C_{m+2}^{2} + \cdots + C_{m+n}^{n} \\ &= C_{m+1}^{m} + C_{m+2}^{m} + \cdots + C_{m+n}^{m} \\ &= ({\color{green}C_{m+1}^{m+1}} + C_{m+1}^{m}) + \cdots + C_{m+n}^{m} - {\color{red}C_{m+1}^{m+1}} \\ &= (C_{m+2}^{m+1} + C_{m+2}^{m}) + \cdots + C_{m+n}^{m} - 1 \\ &= \cdots \\ &= C_{m+n+1}^{m+1} - 1\end{aligned}}$$

exgcd求逆元

```
ll exgcd(ll a,ll b,ll &x,ll &y)//扩展欧几里得算法
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    ll ret=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return ret;
}
ll inv(int a,int mod)//求a在mod下的逆元，不存在逆元返回-1
{
    ll x,y;
    ll d=exgcd(a,mod,x,y);
    return d==1?(x%mod+mod)%mod:-1;
}
```

区间筛

```
int v[maxn], prime[maxn];
int cnt;
void primes(int n)
{
    memset(v, 0, sizeof(v));
    cnt = 0;
    for (int i = 2; i <= n; i++)
    {
        if (v[i] == 0)
        {
            v[i] = i;
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt; j++)
        {
            if (prime[j] > v[i] || prime[j] > n / i)
                break;
            v[i * prime[j]] = prime[j];
        }
    }
}
bool flag[1010];
ll work(ll l,ll r)
{
//    cout<<l<<' '<<r<<endl;
    memset(flag,true,sizeof(flag));
    if(l==1)flag[0]=false;//为1要特判
    for(ll i=1;i<=cnt;i++)
    {
        for(ll j=(l/prime[i])+(l%prime[i]?1:0);j<=(r/prime
[i]);j++)//celi为向上取整,floor为向下取整.
            if (j!=1)
            {
                cout<<prime[i]*j-l<<endl;
                flag[prime[i]*j-l]=false;//统一减去l
            }
    }
    ll ans=0;
    for(int i=0;i<r-l+1;i++)
        if(flag[i])ans++;
        cout<<ans<<endl;
    return ans;
}
```

快速幂

```
ll qpow(ll a,ll b,ll p){
    ll ans=1;
    while(b){
        if(b&1) ans=(ans*a)%p;
        a=a*a%p;
        b>>=1;
    }
    return ans%p;
}
```

矩阵快速幂

```
struct Matrix
{
    int a[maxn][maxn], n;
    Matrix(){};
    Matrix(int _n)
    {
        n = _n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = 0;
    }
    Matrix operator*(Matrix b)
    {
        Matrix res = Matrix(n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                {
                    int teget = (ll)a[i][k] * b.a[k][j] % mo
d;
                    res.a[i][j] = (res.a[i][j] + teget) % mo
d;
                }
        return res;
    }
};
Matrix qpow(Matrix a, int b)
{
    Matrix ans = Matrix(a.n);
    fir(i, 0, a.n - 1) ans.a[i][i] = 1;
    while (b)
    {
        if (b & 1)
            ans = ans * a;
        a = a * a;
        b >>= 1;
    }
    return ans;
}
```

高斯消元

```
namespace Guess
{
    double a[maxn][maxn];
    double eps=1e-10;
    int n;
    void solve()
    {
        for (int r = 1; r <= n; r++)
        {
            int t = r;
            for (int i = r + 1; i <= n; i++)
            {
                if (fabs(a[i][r]) > fabs(a[t][r]))
                    t = i;
            }
            for (int j = r; j <= n + 1; j++)
                swap(a[t][j], a[r][j]);
            for (int j = n + 1; j >= r; j--)
                a[r][j] /= a[r][r];
            for (int i = 1; i <= n; i++)
            {
                if (i != r)
                {
                    for (int j = n + 1; j >= r; j--)
                        a[i][j] -= a[i][r] * a[r][j];
                }
            }
        }
/*      fir(i,1,n)
        {
            fir(j,1,n+1)
            printf("%.2f ",a[i][j]);
            puts("");
        } */
    }
} // namespace Guess
```

异或高斯消元

```
bitset<maxn> b[maxn];
namespace Guess
{
    bitset<maxn> a[maxn];
    //double eps=1e-10;
    int n;
    bool solve()
    {
        for (int r = 1; r <= n; r++)
        {
            int t = r;
            fir(i, r, n) if (a[i][r])
            {
                t = i;//选出当前项
                break;
            }
            if (!a[t][r])//如果都为0跳过本行
                continue;
            //for (int j = r; j <= n + 1; j++)
            swap(a[t], a[r]);//交换
```

```
                swap(id[t], id[r]);
                for (int i = 1; i <= n; i++)
                {//相消
                    if (i != r && a[i][r])
                    {
                        a[i] ^= a[r];
                        b[id[i]] ^= b[id[r]];
                    }
                }
    /*             fir(i, 1, n)
                {
                    fir(j, 1, n)
                            cout
                        << a[i][j] << ' ';
                    cout<<id[i]<<' ';
                    fir(j,1,n)cout<<b[id[i]][j];
                    cout<<endl;
                }
                puts(""); */
            }
            return true;
        }
    } // namespace Guess
```

3.图论

欧拉回路

定义

　　欧拉路径（欧拉通路）：通过图中所有边的简单路。（换句话说，每条边都通过且仅通过一次）也叫"一笔画"问题。

　　欧拉回路：闭合的欧拉路径。（即一个环，保证每条边都通过且仅通过一次）

　　欧拉图：包含欧拉回路的图。

起源历史

　　在一个图中求解一条欧拉回路的问题，起源于欧拉提出的、著名的"七桥问题"。详见百度百科。

判定（充要条件）

　　下列是判定一个图中是否有欧拉回路/欧拉路径的充要条件。

欧拉回路

1.图G是连通的，无孤立点。

2.无向图奇点数为0；有向图每个点的入度必须等于出度。

欧拉路径

1.图G是连通的，无孤立点。

2.无向图奇点数为0或2，并且这两个奇点其中一个为起点另外一个为终点。有向图，可以存在两个点，其入度不等于出度，其中一个入度比出度大1，为路径的起点；另外一个出度比入度大1，为路径的终点。

```
/*
 欧拉回路: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

1 无向图
    1.1 所有点的度数必须是奇数
    1.2 所有边连通
2 有向图
    2.1 所有点的入度等于出度
    2.2 所有边连通

   _O_ 环可以合并进边  或者环可以和环合并 OO

     o
    _|_
   | |_|_
   |   |_|
   o
   对于除了起点与终点外中间的点
   由于它们的度数是偶数 则只要它从某一个过环的出度出发 则必然会走
完这个环回到这个点

   合并环的方法:有环先走环
dfs(u)
{
 for 从n出发的所有边
    dfs() 扩展
 把u加入栈  stk[]←u
}
最终 stk[]中存下的就是欧拉路径的倒序

有向图:
      每用一条边 删掉
无向图:
      每用一条边标记对应的反向边(XOR技巧)
      a→b
      b→a
*/
vector<pair<int,int> > a[maxn];
bool use[maxn];
int n,m;
int t;
int dr[maxn],dc[maxn],d[maxn];
vector<int> ans;
void dfs(int u)
```

```
{
    cout<<u<<endl;
    while(!a[u].empty())
    {
        int v=a[u].back().first,w=a[u].back().second;
        if(!use[w])
        {
            a[u].pop_back();
            continue;
        }
        use[w]=false;
        if(t==1)use[w^1]=false;
        a[u].pop_back();
        dfs(v);
        if(w&1)
        ans.push_back(w/2*-1);
        else
        ans.push_back(w/2);
    }
}
int main()
{
   // input();
    cin>>t;
    cin>>n>>m;
    memset(use,true,sizeof(use));
    fir(i,1,m)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        a[u].push_back({v,2*i});
        dr[v]++,dc[u]++;
        if(t==1)
        a[v].push_back({u,2*i+1}),d[u]++,d[v]++;
    }
    if(t==1)
    {
        fir(i,1,n)
        if(d[i]&1)
        {
            puts("NO");
            return 0;
        }
    }
    else
    {
        fir(i,1,n)
        if(dr[i]!=dc[i])
        {
            puts("NO");
            return 0;
        }
    }
    fir(i,1,n)
    if(a[i].size())
```

```
        {
            dfs(i);
            break;
        }
      //  for(auto it:ans)printf("%d ",it);
        if(ans.size()!=m)
        puts("NO");
        else
        {
            puts("YES");
            for(auto it:ans)
            printf("%d ",it);
        }
    }
```

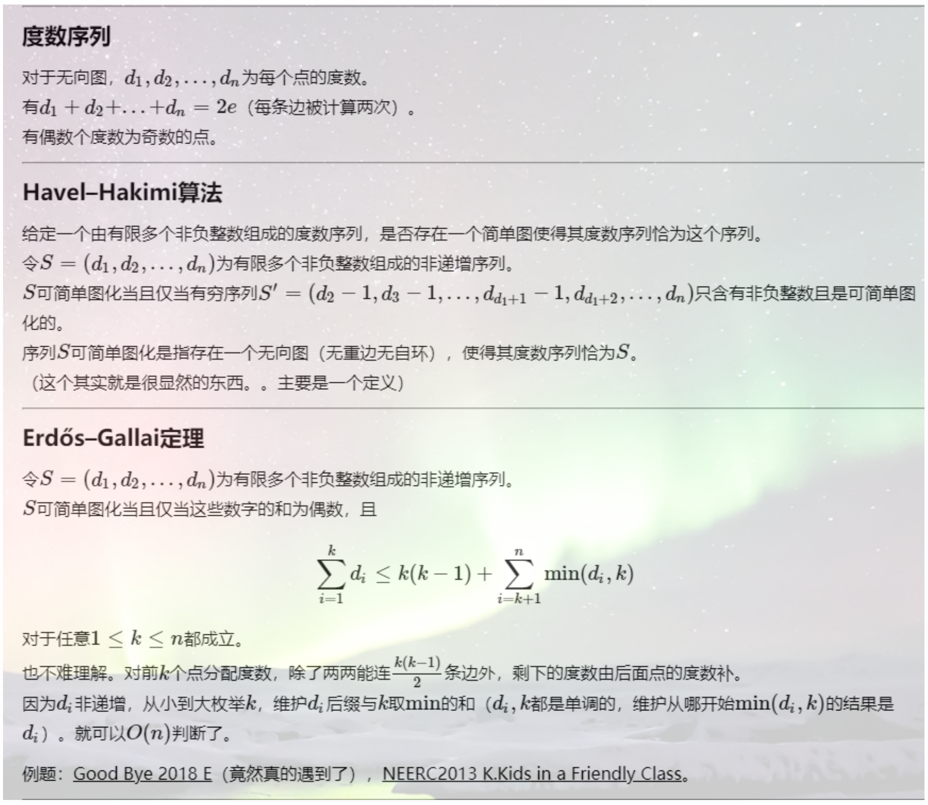
欧拉通路

找出起点，存在欧拉通路的判定：至多有2个奇数点(无奇数度点也满足)，记得判连通

```
    int n;
    int a[maxn][maxn];
    int d[maxn];
    int sta[2000],top=0;
    void dfs(int u)
    {
        for(int i=1;i<=n;i++)
        {
            if(a[u][i])
            {
                a[u][i]--,a[i][u]--;
                dfs(i);
            }
        }
        sta[++top]=u;
    }
    int main()
    {
        cin>>n;
        fir(i,1,n)
        {
            int u,v;
            scanf("%d%d",&u,&v);
            a[u][v]++,a[v][u]++;
            d[u]++,d[v]++;
        }
        n=500;
        int start=1;
        fir(i,1,n)
        if(d[i])
        {
            start=i;
            break;
        }
        for(int i=1;i<=n;i++)
        if(d[i]&1)
        {
            start=i;
            break;
        }
      //  cout<<start<<endl;
        dfs(start);
        while(top)
        {
            printf("%d\n",sta[top--]);
        }
    }
```

Tarjan缩点

定理



度数序列

对于无向图，$d_1, d_2, \dots, d_n$为每个点的度数。
有$d_1 + d_2 + \dots + d_n = 2e$（每条边被计算两次）。
有偶数个度数为奇数的点。

Havel–Hakimi算法

给定一个由有限多个非负整数组成的度数序列，是否存在一个简单图使得其度数序列恰为这个序列。
令$S = (d_1, d_2, \dots, d_n)$为有限多个非负整数组成的非递增序列。
S可简单图化当且仅当有穷序列$S' = (d_2 - 1, d_3 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n)$只含有非负整数且是可简单图化的。
序列S可简单图化是指存在一个无向图（无重边无自环），使得其度数序列恰为S。
（这个其实就是很显然的东西。。主要是一个定义）

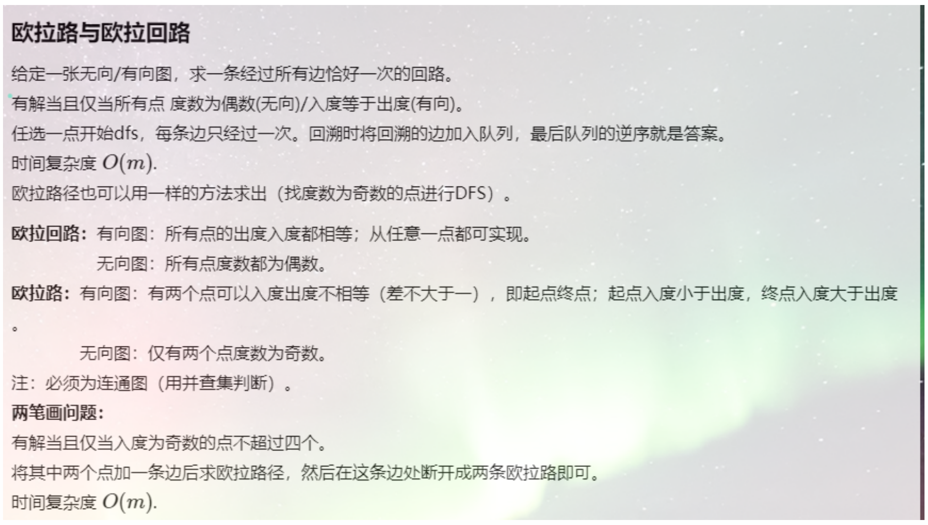
Erdős–Gallai定理

令$S = (d_1, d_2, \dots, d_n)$为有限多个非负整数组成的非递增序列。
S可简单图化当且仅当这些数字的和为偶数，且

$$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

对于任意$1 \le k \le n$都成立。
也不难理解。对前k个点分配度数，除了两两能连$\frac{k(k-1)}{2}$条边外，剩下的度数由后面点的度数补。
因为d_i非递增，从小到大枚举k，维护d_i后缀与k取$\min$的和（d_i, k都是单调的，维护从哪开始$\min(d_i, k)$的结果是d_i）。就可以$O(n)$判断了。

例题：Good Bye 2018 E（竟然真的遇到了），NEERC2013 K.Kids in a Friendly Class。



欧拉路与欧拉回路

给定一张无向/有向图，求一条经过所有边恰好一次的回路。

有解当且仅当所有点 度数为偶数(无向)/入度等于出度(有向)。

任选一点开始dfs，每条边只经过一次。回溯时将回溯的边加入队列，最后队列的逆序就是答案。

时间复杂度 $O(m)$.

欧拉路径也可以用一样的方法求出（找度数为奇数的点进行DFS）。

欧拉回路： 有向图：所有点的出度入度都相等；从任意一点都可实现。

无向图：所有点度数都为偶数。

欧拉路： 有向图：有两个点可以入度出度不相等（差不大于一），即起点终点；起点入度小于出度，终点入度大于出度。

无向图：仅有两个点度数为奇数。

注：必须为连通图（用并查集判断）。

两笔画问题：

有解当且仅当入度为奇数的点不超过四个。

将其中两个点加一条边后求欧拉路径，然后在这条边处断开成两条欧拉路即可。

时间复杂度 $O(m)$.

```cpp
namespace suodian
{
    vector<int> g[maxn];
    int n, m;
    int sta[maxn], top = 0, cnt = 0;
    int dfn[maxn], low[maxn];
    bool use[maxn];
    int siz[maxn];
    int scc_cnt = 0;
    int id[maxn];
    int d[maxn];
    void dfs(int u)
    {
        dfn[u] = low[u] = ++cnt;
        sta[++top] = u;
        use[u] = true;
        for (auto v : g[u])
        {
            if (!dfn[v])
            {
                dfs(v);
                low[u] = min(low[u], low[v]);
            }
            else if (use[v])
                low[u] = min(low[u], dfn[v]);
        }
        if (low[u] == dfn[u])
        {
            ++scc_cnt;
            int v;
            do
            {
                v = sta[top--];
                id[v] = scc_cnt;
                use[v] = false;
                siz[scc_cnt]++;
            } while (v != u);
        }
    }
    void tarjan()
    {
        fir(i, 1, 2 * n) if (!dfn[i]) dfs(i);
    }
    void init()
    {
        fir(i, 1, n)
        {
            sta[i]=0, top = 0, cnt = 0;
            dfn[i]=low[i]=0;
            use[i]=false;
            siz[i]=0;
            scc_cnt = 0;
            id[i]=0;
            d[i]=0;
            g[i].clear();
        }
    }
} // namespace suodian
```

2-sat

对于两个或的关系
比如 x || y,可转化成!x->y,即为若x==0，y就必须为1，
以此为根据建出一个有向图。
通过缩点之后
无解的情况：x和！ x可互相到达(在同一个强连通分量重)，即必须二者同时满足，矛盾。
找解的情况，缩点后的拓扑序，在后面的去满足，(因缩点后的scc_cnt顺序就是拓扑序的逆序)
为什么要选后面的点：
考虑，$!A||B, !A||!B$
存在有解为，A=0,B任取，连出来图为
$A->B，A->!B$
$B->!A$
$!B->!A!$
A可到达!A，若先选A比如选！ A前后矛盾
在2SAT中，若要强制 xx 只能取 xx 而不取 x′x′ ,令 x′x′ 向 xx 连边即可。
例题：
给定 n 个还未赋值的布尔变量 x1~xn。

现在有 m 个条件，每个条件的形式为“xi 为 0/1 或 xj 为 0/1 至少有一项成立”，例如 “x1 为 1 或 x3 为 0”、“x8 为 0 或 x4 为 0” 等。

现在，请你对这 n 个布尔变量进行赋值（0 或 1），使得所有 m 个条件能够成立。

输入格式
第一行包含两个整数 n,m。

接下来 m 行，每行包含四个整数 i,a,j,b，用来描述一个条件，表示 “xi 为 a 或 xj 为 b”。

输出格式
如果问题有解，则第一行输出 POSSIBLE，第二行输出 n 个整数表示赋值后的 n 个变量 x1~xn 的值（0 或 1），整数之间用单个空格隔开。

如果问题无解，则输出一行 IMPOSSIBLE 即可。

如果答案不唯一，则输出任意一种正确答案即可。

```
vector<int> a[maxn];
int n,m;
int sta[maxn],top=0,cnt=0;
int dfn[maxn],low[maxn];
bool use[maxn];
int id[maxn];
int scc_cnt=0;
void dfs(int u)
{
    dfn[u]=low[u]=++cnt;
    sta[++top]=u;
    use[u]=true;
    for(auto v:a[u])
    {
        if(!dfn[v])
        {
            dfs(v);
            low[u]=min(low[u],low[v]);
        }
        else if(use[v])
        low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        ++scc_cnt;
        int v;
        do{
            v=sta[top--];
            id[v]=scc_cnt;
            use[v]=false;
        }while(v!=u);
    }
}
int main()
{
    io>>n>>m;
    fir(i,1,m)
    {
        int u,v,cura,curb;
        io>>u>>cura>>v>>curb;
        a[u+(cura?n:0)].push_back(v+(curb?0:n));
        a[v+(curb?n:0)].push_back(u+(cura?0:n));
    }
    //puts("fuck");
    fir(i,1,2*n)
    if(!dfn[i])
    dfs(i);
    //puts("fuck");
    fir(i,1,n)
    if(id[i]==id[i+n]){
        puts("IMPOSSIBLE");
        return 0;
    }
    puts("POSSIBLE");
    fir(i,1,n)
    {
        printf("%d",id[i]<id[i+n]);
        if(i==n)puts("");
        else putchar(' ');
    }
}

//dfs方法
 O(nm)
namespace Two_sat
{
    vector<int> g[maxn];
    int n, m;
    bool vis[maxn];
    int sta[maxn], top = 0;
    void init(int _n)
    {
        n = _n;
        fir(i, 2, 2 * n + 1)
        {
            vis[i] = false,g[i].clear();
        }
    }
    bool dfs(int u)
    {
        if (vis[u ^ 1])
            return false;
        if (vis[u])
            return true;
        vis[u] = true;
        sta[++top]=u;
        for (auto v : g[u])
            if (!dfs(v))
                return false;
        return true;
    }
    bool solve()
    {
        for (int u = 2; u <= 2 * n; u += 2)
        {
            if (vis[u ^ 1] || vis[u])
                continue;
            top=0;
```

```
                if (!dfs(u))
                {
                    while (top)
                        vis[sta[top--]] = false;
                    if (!dfs(u ^ 1))
                        return false;
                }
            }
            return true;
        }
    }; // namespace Two_sat
```

LCA

```
struct node
{
    int v;
    int next;
    int val;
};
int head[maxn];
node e[maxn << 1];
int d[maxn], f[maxn][30];
int tot = 0;
int t;
void add(int x, int y, int z)
{
    e[++tot] = {y, head[x], z}, head[x] = tot;
}
void bfs(int s)
{
    t = (int)(log(n) / log(2)) + 1;
    queue<int> p;
    p.push(s);
    d[s] = 1;
    while (!p.empty())
    {
        int x = p.front();
        p.pop();
        for (int i = head[x]; i; i = e[i].next)
        {
            int y = e[i].v;
            if (!d[y])
            {
                d[y] = d[x] + 1;
                f[y][0] = x;
                for (int j = 1; j <= t; j++)
                    f[y][j] = f[f[y][j - 1]][j - 1];
                p.push(y);
            }
        }
    }
}
int lca(int x, int y)
{
    if (d[x] > d[y])
        swap(x, y);
    for (int i = t; i >= 0; i--)
        if (d[f[y][i]] >= d[x])
            y = f[y][i];
    if (x == y)
        return x;
    for (int i = t; i >= 0; i--)
        if (f[x][i] != f[y][i])
            x = f[x][i], y = f[y][i];
    return f[x][0];
}
```

对偶图费用流

```
struct Edge {
    int y, c, f, nxt;
    Edge() {}
    Edge(int _y, int _c, int _f, int _nxt) {
        y = _y, c = _c, f = _f, nxt = _nxt;
    }
}e[MAXM];
int lst[MAXN];
int nn, mm;
int E;
int _phi[MAXN], _q[MAXN];
int _s, _t, _cost, _tot;
bool _v[MAXN];
inline bool relable() {
    int x, to, tail = 1, front = 0;
    for(int i = 1; i <= nn; i++) _v[i] = 0, _phi[i] = 0x3fff
ffff;
    _phi[_t] = 0; _q[0] = _t;
    while(tail != front) {
        x = _q[front++]; _v[x] = 0;
        if(front > nn) front = 0;
        for(int i = lst[x]; ~i; i = e[i].nxt) {
            if(e[i ^ 1].c && _phi[to = e[i].y] > _phi[x] - e
[i].f) {
                _phi[to] = _phi[x] - e[i].f;
                if(_v[to] == 0) {
                    _v[to] = 1;
                    if(_phi[to] < _phi[_q[front]])
                        front == 0 ? front = nn : front--, _
q[front] = to;
                    else
                    _q[tail++] = to,tail > nn ? tail = 0 :
0;
                }
            }
        }
    }
    for(int i = 1; i <= nn; i++)
    for(int j = lst[i]; ~j; j = e[j].nxt)
    e[j].f += _phi[e[j].y] - _phi[i];
    _tot += _phi[_s];
    return _phi[_s] < 0x3fffffff;
}
inline int aug(int s, int flow) {
    if(s == _t) return _cost += _tot * flow, flow;
    int res = flow, te; _v[s] = 1;
    for(int i = lst[s]; ~i; i = e[i].nxt) {
        if(!_v[e[i].y] && !e[i].f && e[i].c) {
            res -= (te = aug(e[i].y, min(res, e[i].c)));
            e[i].c -= te; e[i ^ 1].c += te;
            if(!res) return flow;
        }
    }
    return flow - res;
}
pair<int, int> cost_flow(int s, int t) {
    _cost = _tot = 0; _s = s; _t = t; relable();
    int flow = 0, te = 0;
    do memset(_v,0,sizeof(_v)), flow += te;
    while((te = aug(s, 0x3fffffff)) || relable());
    return make_pair(_cost, flow);
}
inline void add_edge(int x, int y, int c, int f) {
    e[mm] = Edge(y, c, f, lst[x]); lst[x] = mm++;
    e[mm] = Edge(x, 0, -f, lst[y]); lst[y] = mm++;
}
signed main() {
    memset(lst, -1, sizeof(lst));
    read(nn), read(E);
    read(_s), read(_t);
    for(int i = 1, a, b, c, d; i <= E; i++) {
        read(a), read(b);read(c), read(d);
        add_edge(a, b, c, d);
    }
    pair<int, int> alpha;
    alpha = cost_flow(_s, _t);
    printf("%d %d\n", alpha.second, alpha.first);
    return 0;
}
```

最小费用最大流

```
    struct node
    {
        int v;
        int next;
        ll cost;
        int val;
    };
    struct MCMF
    {
        int head[N], flow[N], pre[N];
        bool flag[N];
        ll dis[N];
        int n, m, s, t, tot = 1;
        ll maxflow = 0, mincost = 0;
        node e[M];
        void add(int u, int v, int val, ll c)
        {
            e[++tot] = { v, head[u], c, val };
            head[u] = tot;
            e[++tot] = { u, head[v], -c, 0 };
            head[v] = tot;
        }
        bool spfa()
        {
            fir(i, 0, N - 1)
                dis[i] = 1e18;//最大费用为-1e18,也可边权取负跑最短
    路
            memset(flow, MAX_INF, sizeof(flow));
            memset(flag, true, sizeof(flag));
            queue<int> p;
            p.push(s);
            dis[s] = 0;
            pre[t] = -1;
            while (!p.empty())
            {
                int x = p.front();
                p.pop();
                flag[x] = true;
                for (int i = head[x]; i; i = e[i].next)
                {
                    int y = e[i].v;
                    if (e[i].val && dis[y] > dis[x] +
    e[i].cost)//最大费用为<
                    {
                        dis[y] = dis[x] + e[i].cost;
                        pre[y] = i;
                        flow[y] = min(flow[x], e[i].val);
                        if (flag[y])
                        {
                            p.push(y);
                            flag[y] = false;
                        }
                    }
                }
            }
            return pre[t] != -1;
        }
        void updata()
        {
            int x = t;
            while (x != s)
            {
                int i = pre[x];
                e[i].val -= flow[t];
                e[i ^ 1].val += flow[t];
                x = e[i ^ 1].v;
            }
            mincost += flow[t] * dis[t];
            maxflow += flow[t];
                }
        void work()
        {
            mincost=0;
            maxflow=0;
            while(spfa())
            updata();
            printf("%lld %lld\n",maxflow,mincost);
        }
        void init(int _s, int _t)
        {
            memset(head, 0, sizeof(head));
            s = _s;
            t = _t;
        }
    };
    int main()
    {
        int n,m,s,t;
        scanf("%d%d%d%d",&n,&m,&s,&t);
        MCMF mc;
        mc.init(s,t);
        fir(i,1,m)
        {
            int u,v;
            int c,z;
            scanf("%d%d%d%d",&u,&v,&c,&z);
            mc.add(u,v,c,z);
        }
        mc.work();
        //cout<<maxflow<<' '<<mincost<<endl;
        return 0;
    }
```

最大流

```
struct node
{
    int v;
    int next;
    ll val;
};
```

```
struct MCMF
{
    node e[M];
    int head[N], d[N], now[M];
    int s, t, tot = 1;
    void init(int _s,int _t)
    {
        tot=1;
        s=_s;
        t=_t;
        memset(head,0,sizeof(head));
    }
    inline void add(int x, int y, ll c)
    {
        e[++tot] ={ y, head[x], c };
        head[x] = tot;
        e[++tot] ={ x, head[y], 0 };
        head[y] = tot;
    }
    inline bool bfs()
    {
        memset(d, 0, sizeof(d));
        queue<int> p;
        p.push(s);
        d[s] = 1;
        now[s] = head[s];
        while (!p.empty())
        {
            int x = p.front();
            p.pop();
            for (int i = head[x]; i; i = e[i].next)
            {
                int v = e[i].v;
                if (e[i].val && !d[v])
                {
                    p.push(v);
                    d[v] = d[x] + 1;
                    now[v] = head[v];
                    if (v == t)
                        return true;
                }
            }
        }
        return false;
    }
    inline ll dinic(int x, ll flow)
    {
        if (x == t)
            return flow;
        ll rest = flow, k;
        int i;
        for (i = now[x]; i && rest; i = e[i].next)
        {
            int v = e[i].v;
            if (e[i].val > 0 && d[v] == d[x] + 1)
```

```
                {
                    now[x] = i;
                    k = dinic(v, min(rest, e[i].val));
                    if (!k)
                        d[v] = 0;
                    e[i].val -= k;
                    e[i ^ 1].val += k;
                    rest -= k;
                }
            }
            return flow - rest;
        }
        void work()
        {
            ll ans = 0;
            ll flow = 0;
            while (bfs())
                ans += dinic(s, (ll)20050020600);
            printf("%lld\n", ans);
        }
};
int main()
{
    // freopen("b1.in", "r", stdin);
    // auto start = chrono::steady_clock::now();
    int n,m,s,t;
    //scanf("%d%d%d", &n, &m, &e);
    cin>>n>>m>>s>>t;
    MCMF mcmf;
//  s = 0, t = 1001;
    mcmf.init(s,t);
    fir(i, 1, m)
    {
        int x, y;
        ll w;
        scanf("%d%d%lld", &x, &y,&w);
        mcmf.add(x, y, w);
    }
    mcmf.work();
    //  auto end = chrono::steady_clock::now();
    //  cout << chrono::duration_cast<chrono::microseconds>
(end - start).count() << "us\n";
    //二分图匹配输出方案: 如果无流(正向边则输出)
    /*for(int i=2+n*2;i<=tot;i+=2)
        {
            if(!e[i].val)
            printf("%d %d\n",e[i].u,e[i].v);
        }
        */
    return 0;
}
```

树同构

```
set<ull> s[100];
vector<int> a[100];
const ull p=131;
ull dfs(int u, int head)
{
    ull ans=1;
    vector<ull> cnt;
    for(int i=0;i<a[u].size();i++)
    {
        int v=a[u][i];
        if(v!=head)
        {
            cnt.push_back(dfs(v,u));
        }
    }
    sort(cnt.begin(),cnt.end());
    for(auto it:cnt)
    ans+=p*it;
    return ans*ans;
}
```

二分图最大匹配

匈牙利

```
struct xiongyali
{
    vector<int> a[maxn];
    int match[maxn];
    int visit[maxn];
    int n, m;
    void init(int _n, int _m)
    {
        n = _n;
        m = _m;
        fir(i,1,n)a[i].clear();
    }
    bool dfs(int x)
    {
        for (auto j : a[x])
        {
            if (visit[j])
                continue;
            visit[j] = 1;
            if (!match[j] || dfs(match[j]))
            {
                match[j] = x;
                return true;
            }
        }
```

```
            return false;
        }
        /*//用数组做
         bool dfs(int x)
        {
            for (int j = 1; j <= m; j++)
            {
                if (a[x][j])
                {
                    if (visit[j])
                        continue;
                    visit[j] = 1;
                    if (!match[j] || dfs(match[j]))
                    {
                        match[j] = x;
                        return true;
                    }
                }
            }
            return false;
        }
        */
        int solve()
        {
            int ans = 0;
            memset(match, 0, sizeof(match)); //此处初始化match和v
    isit,每次更新a即可
            for (int i = 1; i <= n; i++)
            {
                memset(visit, 0, sizeof(visit));
                if (dfs(i))
                    ans++;
            }
            return ans;
            //输出匹配方案:
            /*
            在m的那边
             fir(i,m+1,n)
            if(match[i])
            printf("%d %d\n",match[i],i);
            */
        }
    } t;
    int main()
    {
        int debug = 0;
        if (debug)
        {
            freopen("in.txt", "r", stdin);
            freopen("out.txt", "w", stdout);
        }
        int n, m, e;
        cin >> n >> m >> e;
        t.init(n, m);
        while (e--)
        {
            int x, y;
            scanf("%d%d", &x, &y);
            t.a[x].push_back(y);
        }
        cout << t.solve() << endl;
    }
```

最小生成树

```
struct node
{
    int x,y,z;
    bool operator<(const node & A)const
    {
        return z<A.z;
    }
};
node edge[maxn];
int fa[maxn],s[maxn];
int get(int x)
{
    if(fa[x]==x)
    return x;
    return fa[x]=get(fa[x]);
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        for(int i=0;i<n-1;i++)
            scanf("%d%d%d",&edge[i].x,&edge[i].y,&edge[i].
z);
        ll ans=0;
        sort(edge,edge+n-1);
        for(int i=1;i<maxn;i++)
        fa[i]=i,s[i]=1;
        for(int i=0;i<n-1;i++)
        {
            int x=get(edge[i].x);
            int y=get(edge[i].y);
            if(x==y)
            continue;
            ans+=(ll)(edge[i].z+1)*(s[x]*s[y]-1);
            fa[y]=fa[x];
            s[x]+=s[y];
        }
        cout<<ans<<endl;
    }
}
```

floyd最短路

```
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
```

spfa

```
struct node
{
    int v;
    int val;
};
vector<node> a[maxn];
int n, r, p, s;
int d[maxn];
bool flag[maxn];
void spfa()
{
    deque<int> p;
    p.push_back(s);
    memset(flag, true, sizeof(flag));
    d[s] = 0;
    while (!p.empty())
    {
        int temp = p.front();
        p.pop_front();
        flag[temp] = true;
        for (int i = 0; i < a[temp].size(); i++)
        {
            int v = a[temp][i].v;
            int val = a[temp][i].val;
            if (d[v] > d[temp] + val)
            {
                d[v] = d[temp] + val;
                if (flag[v])
                {
                    flag[v] = false;
                    if (!p.empty()&&d[v] <d[p.front()])
                        p.push_front(v);
                    else
                        p.push_back(v);
                }
            }
        }
    }
}
```

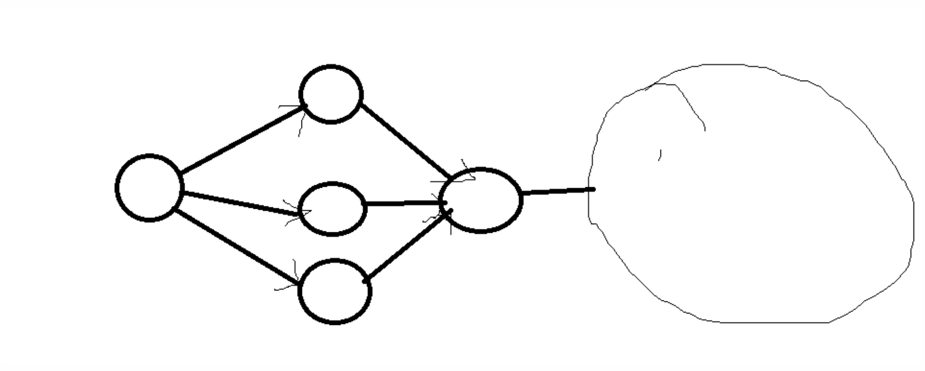
4.基础

```
补0输出位数
printf("%03d",a);
```

4.dp

图dp和树dp的区别

树dp每个点只会走一次，故不用记录是否走过
图上的dp，可能出现情况有带环，和一个点多个入度，故需要记录点是否走过



LIS结论

一串序列$\{a_i\}$要单调递增需要改变的数值个数，那么我们所求的代价相当于序列$\{a_i-i\}$中最长不下降子序列的长度。

二分贪心LIS

//单调不下降(可相等)

```
int a[maxn];
int dp[maxn];
int pos[maxn];
int main()
{
    //input();
    int T;
    cin>>T;
    while(T--)
    {
        int n;
        scanf("%d",&n);
        fir(i,1,n)scanf("%d",&a[i]);
```

```
            reverse(a+1,a+n+1);
            int cnt=1;
            pos[1]=a[1];
            dp[1]=1;
            fir(i,2,n)
            {
                int &x=a[i];
                if(x>pos[cnt])
                {
                    pos[++cnt]=x;
                    dp[i]=cnt;
                }
                else
                {
                    int now=upper_bound(pos+1,pos+cnt+1,x)-pos;
                    //单调上升此处为lower_bound
                    dp[i]=now;
                    pos[now]=x;
                }
            }
            printf("%d\n",cnt);
            for(int i=n;i>=1;i--)
            printf("%d ",dp[i]);
            puts("");
        }
    }
```

求n个数，m个逆序对得排列数

$dp[i][j]$表示i个数时，存在j个逆序对的排列种数。

2.分析状态转移方程：打个比方我们要求$dp[4][2]$，它的总
数排列是$B[]=1,4,2,3,3,1,2,4,2,1,4,3,2,3,1,4,1,3,4,2.$
在四个数中存在2个逆序对

无非是考虑第i个如何插入$i-1$个数之中，当前$i-1$个位正排序是（1,2,3），
那就把4插入2的前面1的后面，
即（1,4,2,3）$--dp[3][0].$假设已经存在一个逆序对了。（1,3,2）

，（2,1,3）。那我们只需要插入的位置只要大于一个
数就好了（1,3,4,2），（2,1,4,3）$--dp[3][1]$，
最后假设已经存在两个逆序对了$(3,1,2),(2,3,1)$，只需放到最后即
可（3,1,2,4），（2,3,1,4）.转移方程相信
已经很浅显了：$d[i][j]=d[i-1][j]+d[i-1][j-1]+......+d[i-1][0];$
3.初始化：$dp[i][0]=1;$正序为1.

垂线法

$O(n*m)$

```
    //O(n*m)
    cin >> n >> m;
    fir(i, 1, n)
        fir(j, 1, m)
            scanf("%d", &a[i][j]),l[i][j]=r[i][j]=j;
    fir(i, 1, n)
        fir(j, 1, m)
        {
        if (j == 1)
        {
            l[i][j] = 1;
            continue;
        }
        if (a[i][j] != a[i][j - 1])
            l[i][j] = l[i][j - 1] + 1;
        else
            l[i][j] = 1;
    }
    fir(j, 1, m)
        fir(i, 1, n)
    {
        if (i == 1)
        {
            up[i][j] = 1;
            continue;
        }
        if (a[i][j] != a[i - 1][j])
            up[i][j] = up[i - 1][j] + 1;
        else
            up[i][j] = 1;
    }
    for (int i = n; i >= 1; i--)
        for (int j = m; j >= 1; j--)
        {
            if (j == m)
            {
                r[i][j] = 1;
                continue;
            }
            if (a[i][j] != a[i][j + 1])
                r[i][j] = r[i][j + 1] + 1;
            else
                r[i][j] = 1;
        }
    int ans1 = 0, ans2 = 0;
    fir(i, 1, n)
    {
        fir(j, 1, m)
        {
            if (i > 1&&a[i][j]!=a[i-1][j])//上方能拓展的时候才
更新左右
                l[i][j] = min(l[i][j],l[i - 1][j]),r[i][j] =
 min(r[i][j], r[i - 1][j]);
            int len = l[i][j] + r[i][j] - 1;
            int h = up[i][j];
            int c = min(len, h);
            ans1 = max(ans1, c * c);
            ans2 = max(ans2, len * h);
        }
    }
//   puts("left");show(l);
//   puts("right");show(r);
//   puts("up");show(up);
    cout << ans1 << endl << ans2 << endl;
```

用途:
解决给定矩阵中满足条件的最大子矩阵

做法:
用一条线(横竖貌似都行)左右移动直到不满足约束条件或者到达边界

定义几个东西:
$left[i][j]left[i][j]$：代表从$(i,j)(i,j)$能到达的最左位置

$right[i][j]right[i][j]$：代表从$(i,j)(i,j)$能到达的最右位置

$up[i][j]up[i][j]$：代表从$(i,j)(i,j)$向上扩展最长长度.

递推公式:
$left[i][j] = max(left[i][j], left[i-1][j]$
$left[i][j] = max(left[i][j], left[i-1][j]$

$right[i][j] = min(right[i][j], right[i-1][j]$
$right[i][j] = min(right[i][j], right[i-1][j]$

至于为什么递推公式中考虑上一层的情况?
是因为up数组的定义,up数组代表向上扩展最长长度, 所以需要考虑上一层的情况.

TSP问题

$O(2^n * n * n)$

```
 memset(dp, MAX_INF, sizeof(dp));
    fir(i, 0, n)
        dp[mov(i)][0] = a[0][i];//初始化路径
    for (int j = 0; j < (1 << (n + 1)); j++)
    {//枚举状态
        fir(i, 0, n)
        {
            if (j & mov(i))
            {
                int c = j - mov(i);
                fir(k, 0, n)
                {
                    if (c & mov(k))//从哪里走过来
                        dp[j][i] = min(dp[j][i], dp[c][k] +
a[k][i]);
```

```
                }
            }
        }
    }
    ll ans = 1e18;
    fir(i, 1, n)
        ans = min(ans,(ll)dp[mov(n + 1) - 1][i] + a[i][0] +
 num * n);
```

7 计算几何

板子

```
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;
const double eps = 1e-6;
const double PI = acos(-1);
struct Point{//角
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){};
};
typedef Point Vector;
int sgn(double d){
    if(fabs(d) < eps)return 0;//0
    if(d > 0)return 1;//正数
    return -1;//负数
}
int dcmp(double x, double y){
    if(fabs(x - y) < eps)return 0;//相等
    if(x > y)return 1;//x大
    return -1;//y大
}
Vector operator + (Vector A, Vector B){return Vector(A.x +
 B.x, A.y + B.y);}//加
Vector operator - (Vector A, Vector B){return Vector(A.x -
 B.x, A.y - B.y);}//减
Vector operator * (Vector A, double p){return Vector(A.x *
 p, A.y * p);}//乘
Vector operator / (Vector A, double p){return Vector(A.x /
 p, A.y / p);}//除
bool operator == (const Point &a, const Point &b) {//两点相等
    if(sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0)
        return 1;
    return 0;
}

double Dot(Vector A, Vector B){return A.x * B.x + A.y * B.
y;}//向量内积
double Length(Vector A){return sqrt(Dot(A,A));}//向量长度
double Angle(Vector A, Vector B){return acos(Dot(A,B) / Leng
th(A) / Length(B));}//向量夹角，返回弧度
double Cross(Vector A, Vector B){return A.x * B.y - A.y * B.
x;}//向量外积
Vector Rotate(Vector A, double rad){return Vector(A.x *
cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(ra
d));}//向量旋转，正角为逆时针，负角为顺时针,弧度制，先把角度转化
为弧度

double torad(double angle){//角度转换为弧度
    return angle / 180 * PI;
}
double toangle(double rad){//弧度转换为角度
    return 180 / PI * rad;
}

Point GetLinersection(Point P, Vector v, Point Q, Vector w)
{//先调用Corss(P, Q)判断两点不平行
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}

bool isparallel(Vector A, Vector B){//平行
    return sgn(Cross(A, B)) == 0;
}
bool isvertical(Vector A, Vector B){//垂直
    return sgn(Dot(A, B)) == 0;
}
double Cross(Point a1, Point a2, Point b1, Point b2){
    return (a2.x - a1.x) * (b2.y - b1.y) - (b2.x - b1.x) *
 (a2.y - a1.y);
}
double calArea(Point *p, int cnt){
    double ans = 0;
    for(int i = 3; i <= cnt; i++){
        ans += Cross(p[1], p[i - 1], p[1], p[i]) / 2;
    }
    return ans;
}
int main(){

    return 0;
}
```

7.1 double精度问题

7.1.1 判断x和y的关系

```
const double eps = 1e-6;
int dcmp(double x, double y){
    if(fabs(x - y) < eps)//相等
```

```
            return 0;
        if(x > y)//x大
            return 1;
        return -1;//y大
    }
```

7.1.2 判断x的符号

```
int sgn(double d){
    if(fabs(d) < eps)//0
        return 0;
    if(d > 0)//正数
        return 1;
    return -1;//负数
}
```

7.2 点和向量

7.2.1 点的表示

```
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
```

7.2.2 向量基本运算

```
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
Vector operator + (Vector A, Vector B){//点 + 点
    return Vector(A.x + B.x, A.y + B.y);
}
Vector operator - (Vector A, Vector B){// 点 - 点
    return Vector(A.x - B.x, A.y - B.y);
}
Vector operator * (Vector A, double p){// 点 * p
    return Vector(A.x * p, A.y * p);
}
Vector operator / (Vector A, double p){// 点 / p
    return Vector(A.x / p, A.y / p);
}
bool operator < (const Point& a, const Point& b){// 点a是否在
点b左下角
```

```
    if(a.x == b.x)
        return a.y < b.y;
    return a.x < b.x;
}
bool operator == (const Point& a, const Point& b){// 点a是否和点b重合
    if(dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0)
        return true;
    return false;
}
```

7.2.3 向量的内积运算

$\alpha \cdot \beta = |\alpha||\beta|cos\theta = A.x * B.x + A.y * B.y$

- 若α与β的夹角为锐角，则其内积为正
- 若α与β的夹角为钝角，则其内积为负
- 若α与β的夹角为直角，则其内积为0

```
double Dot(Vector A, Vector B){
    return A.x * B.x + A.y * B.y;
}
```

7.2.4 向量的外积运算

$\alpha \times \beta = |\alpha||\beta|sin\theta = A.x * B.y - A.y * B.x$
三角形面积是 $\frac{\alpha\times\beta}{2}$

- 若α在β顺时针方向为正
- 若α在β逆时针方向为负

```
double Cross(Vector A, Vector B){
    return A.x * B.y - A.y * B.x;
}
```

常用函数

向量取模

```
double Length(Vector A){
    return sqrt(Dot(A, A));
}
```

计算两向量夹角

```
double Angle(Vector A, Vector B){
    return acos(Dot(A, B) / Length(A) / Length(B));
}
```

计算两向量构成的平行四边形有向面积

```
double Area2(Point A, Point B, Point C){
    return Cross(B - A, C - A);
}
```

计算向量逆时针旋转后的向量
旋转前$x_0 = |R| * cosA, y_0 = |R| * sinA$
旋转后$x_1 = |R|cos(A + b), y_1 = |R| * sin(A + B)$
旋转后的点进行展开，然后利用旋转前带入有
$x_1 = x_0 cosB - y_0 sinB, y_1 = x_0 * sinB + y_0 * cosB$

```
Vector Rotate(Vector A, double rad){//rad为弧度 且为逆时针旋转
的角
    return Vector(A.x * cos(rad) - A.y * sin(rad), A.x *
sin(rad) + A.y * cos(rad));
}
```

平面上一点x_1, y_1绕平面上另一点x_2, y_2顺时针旋转θ角度，求出旋转后x_1, y_1坐标

$$x = (x1 - x2)cos\theta - (y1 - y2)sin\theta + x2$$
$$y = (y1 - y2)cos\theta + (x1 - x2)sin\theta + y2$$

博客证明
计算向量逆时针旋转九十度的单位法向量

```
Vector Normal(Vector A){//向量A左转90°的单位法向量
    double L = Length(A);
    return Vector(-A.y / L, A. x / L);
}
```

角度与弧度转换

```
double torad(double angle){//角度转换为弧度
    return angle / 180 * PI;
}
double toangle(double rad){//弧度转换为角度
    return 180 / PI * rad;
}
```

三点共线
对于任意三点，要么三点共线，要么两点共线，一点不共线

```
bool isLine(Point a, Point b, Point c){
    return Cross(a - b, b - c) == 0;
}
```

7.3 线段表示

- 一般式$ax + by + c = 0$
- 点向式$x_0 + y_0 + v_x t + v_y t = 0$
- 斜截式$y = kx + b$
- 向量表示(x_1, y_1)

已知两个点$a(x_1, y_1), b(x_2, y_2)$
利用两点式 $\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1}$ 有
$(y_2 - y_1)x - (x_2 - x_1)x = x_1y_2 - x_2y_1$
$Ax + By = C$
$A = (y_2 - y_1), B = (x_1 - x_2), C = x_1y_2 - x_2y_1$

7.4 两条直线判断

两向量A和B

7.4.1 直线平行

即$cross(A, B) = 0$

```
bool isparallel(Vector A, Vector B){//平行
    return sgn(Cross(A, B)) == 0;
}
```

7.4.2 直线垂直

即$Dot(A, B) = 0$

```
bool isvertical(Vector A, Vector B){//垂直
    return sgn(Dot(A, B)) == 0;
}
```

7.4.3 直线相交

利用直线构造公式

```
bool check(Point a, Point b, Point c, Point d){
    int A1 = b.y - a.y, B1 = a.x - b.x, C1 = a.x * b.y - b.x
 * a.y;
    int A2 = d.y - c.y, B2 = c.x - d.x, C2 = c.x * d.y - d.x
 * c.y;
    int d1 = gcd(gcd(A1, B1), C1), d2 = gcd(gcd(A2, B2),
C2);
    A1 /= d1, B1 /= d1, C1 /= d1;
    A2 /= d2, B2 /= d2, C2 /= d2;
    if(A1 == A2 && B1 == B2){//平行
        if(C1 == C2) return 1;// 重合
        else return 0;//平行
    }
    return 1;
}
```

n条直线求相交直线个数，配合map来降时间复杂度

```
std::map<pair<ll,ll>, int> mp1;
std::map<pair<pair<ll,ll>,ll>, int> mp2;
void solve(){
    mp1.clear(), mp2.clear();
    int n;
    ll ans = 0;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        ll x1, x2, y1, y2;
        scanf("%lld%lld%lld%lld", &x1, &y1, &x2, &y2);
        ll A = y2 - y1;
        ll B = x1 - x2;
        ll C = x1 * y2 - x2 * y1;
        ll d = gcd(gcd(A, B), C);
        A /= d, B /= d, C /= d;
        mp1[{A, B}]++;
        mp2[{{A,B},C}]++;
        ans += i - 1 - (mp1[{A, B}] - mp2[{{A, B} ,C}]);
    }
    printf("%lld\n", ans);
}
```

7.4.4 直线求交点

当两个向量不平行时，即$cross(A, B)$不为0时有交点
求以经过P的向量v和经过Q的向量w的交点
就是三角形，知道两条边，和两个点的坐标，求第三个点，做个垂直，利用两边长度乘sin是垂线长度列等式即可

```
Point GetLinersection(Point P, Vector v, Point Q, Vector w){
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}
```

7.5 线段

7.5.1 线段求交点

```

```

7.5.2 线段平移

把线段ab向其垂直方向平移R长度，左边或右边

```
struct Point{
    double x, y;
    Point (double x = 0, double y = 0):x(x),y(y){};
    double len(){return sqrt(x * x + y * y);}
    Point trunc(double r) {
        double le = len();
        if(sgn(le) == 0) return *this;
        return Point(x * r / le, y * r / le);
    }
    Point rotleft(){return Point(-y, x);}
    Point rotright(){return Point(y, -x);}
};
void solve(Point a, Point b, double R){
    Point aa = a + (b - a).rotleft().trunc(R); // ab沿着左边
垂直方向平移
    Point bb = b + (b - a).rotleft().trunc(R); //
}
```

7.6 等分点

求线段AB上的二等分点为$(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$

求线段AB上靠近A的k等分点$(x_1+\frac{x_2-x_1}{k}, y_1+\frac{y_2-y_1}{k}) = (\frac{x_2+(k-1)x_1}{k}, \frac{y_2+(k-1)y_1}{k})$

7.7 扇形

7.7.1 弧长

$L = \alpha \times r$，r是边长，α是扇形角的弧度制

7.8 Pick定理

如果一个平面直角坐标系内，以整点为顶点的简单多边形(任意两边不交叉)，它内部整点数位a，它的边上(包括顶点)的整点数为b，那么对于面积S满足

$2S = 2a + b - 2$，即$S = a + \frac{b}{2} - 1$

求边上的点：对于一条边a, b求其上面的格点数

```
ll cal(Point a, Point b){ // 求除了两端点外线段上的格点数
    ll m = ABS(a.x - b.x);
    ll n = ABS(a.y - b.y);
    if(m == 0 && n == 0) return 0;
    if(m == 0) return n - 1;
    if(n == 0) return m - 1;
    return gcd(n, m) - 1;
}
```

7.9 三角剖分

7.10 凸包

给定平面上的数量为n二维点集，求解其凸包。

> 凸包指在平面上能包含所有给定点的最小凸多边形。

Graham算法
时间复杂度$O(nlogn)$

- 按照一个点进行极角排序
- 从该点开始逆时针扫描，找到合适的边

选择一个y值最小(如果相同就选x最小)的点，记为P_1
剩下的点集中按照极角的大小逆时针排序，编号为$P_2 \sim P_m$

特判：n=1 时为0，n = 2时，为两点距离
选择尽可能少的点构成凸包

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const int N = 1e5 + 5;
struct Point{
    double x, y;
}p[N], s[N];
double Cross(Point a1, Point a2, Point b1, Point b2){//
    return (a2.x - a1.x) * (b2.y - b1.y) - (b2.x - b1.x) *
 (a2.y - a1.y);
}
double dis(Point p1, Point p2){
    return sqrt((p2.y - p1.y) * (p2.y - p1.y) + (p2.x - p1.
x) * (p2.x - p1.x));
}
bool cmp(Point p1, Point p2){
    double tmp = Cross(p[1], p1, p[1], p2);
    if(tmp > 0) return 1;
    if(tmp == 0 && dis(p[1], p1) < dis(p[1], p2)) return 1;
    return 0;
}
int n, cnt;
void hull(){//求二维凸包
    sort(p + 2, p + n + 1, cmp);
    s[++cnt] = p[1];
    for(int i = 2; i <= n; i++){
        while(cnt > 1 && Cross(s[cnt - 1], s[cnt], s[cnt], p
[i]) <= 0)// = 0表示三点直线，不选择
            cnt--;
        s[++cnt] = p[i];
    }
    s[cnt + 1] = p[1];
}
int main(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%lf%lf", &p[i].x, &p[i].y);
        if(p[1].y > p[i].y || (p[1].y == p[i].y && p[1].x >
 p[i].x))//找最小值
            swap(p[i], p[1]);
    }

    hull();

    double ans = 0;
    for(int i = 1; i <= cnt; i++)
        ans += dis(s[i], s[i + 1]);
    if(n == 2) printf("%.2f\n", ans / 2); //看题目n=2时是周长
还是线段
    else printf("%.2f\n", ans);
    return 0;
}
```

7.11 旋转卡壳

> 旋转卡壳是一种求出凸包所有对踵点对的算法。

求凸包的直径
凸包直径是在凸包上的最远距
时间复杂度$O(nlogn)$

- 求出二维凸包

- 逆时针遍历每个点
- 以相邻点组成的边为底，找到最大三角形面积
- 答案就是相邻点到对顶点的距离的最大值

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
using namespace std;
const int N = 1e5 + 5;
struct Point{
    double x, y;
}p[N], s[N];
int n, cnt;
double Cross(Point a1, Point a2, Point b1, Point b2){//外积
    return (a2.x - a1.x) * (b2.y - b1.y) - (a2.y - a1.y) *
 (b2.x - b1.x);
}
double dis(Point p1, Point p2){
    return sqrt((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.
y) * (p2.y - p1.y));
}
bool cmp(Point p1, Point p2){
    double tmp = Cross(p[1], p1, p[1], p2);
    if(tmp > 0) return 1;
    if(tmp == 0 && dis(p[1], p1) < dis(p[1], p2)) return 1;
    return 0;
}
void hull(){//求二维凸包
    sort(p + 2, p + n + 1, cmp);
    s[++cnt] = p[1];
    for(int i = 2; i <= n; i++){
        while(cnt > 1 && Cross(s[cnt - 1], s[cnt], s[cnt], p
[i]) <= 0)
            cnt--;
        s[++cnt] = p[i];
    }
    s[cnt + 1] = p[1];
}
double getdis(){
    if(cnt == 1) return 0;
    if(cnt == 2) return dis(s[1], s[2]);
    int v = 2;
    double ans = 0;
    for(int i = 1; i <= cnt; i++){
        while(Cross(s[v], s[i], s[v], s[i + 1]) <=
            Cross(s[v + 1], s[i], s[v + 1], s[i + 1])){//面
积比较
            v = v == cnt ? 1 : v + 1;//p[i]和p[i + 1]的对顶点
        }
        ans = max(ans, max(dis(s[i], s[v]), dis(s[i + 1], s
[v])));
```

```
    }
    return ans;
}
int main(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%lf%lf", &p[i].x, &p[i].y);
        if(p[1].y > p[i].y || (p[1].y == p[i].y && p[1].x > 
p[i].x))//找最小值
            swap(p[1], p[i]);
    }
    hull();
    printf("%.2lf\n", getdis());
    return 0;
}
```

7.12 扫描线

7.13 半平面交

> 多个半平面的交集称之为半平面交。

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const int N = 505;
const double eps = 1e-6;
int sgn(double x){
    if(fabs(x) < eps) return 0;
    return x > 0 ? 1 : -1;
}
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){};
    double k() const{
        return atan2(y, x);
    }
    Point operator + (const Point &b) const{
        return Point(x + b.x, y + b.y);
    }
    Point operator - (const Point &b) const{
        return Point(x - b.x, y - b.y);
    }
    double operator ^ (const Point &b) const{ // 大于0时，b在
点的逆时针方向
        return x * b.y - y * b.x;
    }
    double operator * (const Point &b) const{
        return x * b.x + y * b.y;
    }
    Point operator * (const double &b) const{
        return Point(x * b, y * b);
    }
}p[N];
typedef Point Vector;
struct Line{
    Point from, to;
    Line(Point from = Point(), Point to = Point()):from(fro
m), to(to){};
    double k() const{
        return (to - from).k();
    }
    bool operator < (const Line &b) const{ // 平行就先考虑右下
角的，不平行就考虑按照斜率角逆时针排序
        return sgn(k() - b.k()) == 0 ? sgn((to - from) ^ (b.
to - from)) > 0 : sgn(k() - b.k()) < 0;
    }
}l[N], q[N];
int cnt = 0, n, head, tail;
Point GetIntersection(Line x, Line y){ // 求x和y的交点
    Vector a = x.to - x.from, b = y.to - y.from, c = y.from
 - x.from;
    return y.from + b * ((c ^ a) / (a ^ b));
}
bool OnLeft(Line x, Line y, Line z){ // 判断x和y的交点是否在z
左侧
    Point p = GetIntersection(x, y);
    return ((z.to - z.from) ^ (p - z.from)) < 0;
}
void HALF(){ // 求半平面, 输入一写直线即可, 直线方向朝逆时针,最
后p是交点集合, 逆时针
    sort(l + 1, l + cnt + 1); // 极角排序
    int tot = 0;
    for(int i = 1; i <= cnt; i++){ // 去重
        if(sgn(l[i].k() - l[i - 1].k())) tot++;
        l[tot] = l[i];
    }
    head = 1, tail = 0;
    for(int i = 1; i <= tot; i++){
        while(head < tail && OnLeft(q[tail - 1], q[tail], l
[i])) tail--;
        while(head < tail && OnLeft(q[head + 1], q[head], l
[i])) head++;
        q[++tail] = l[i];
    }
    while(head < tail && OnLeft(q[tail - 1], q[tail], q[hea
d])) tail--;
    while(head < tail && OnLeft(q[head + 1], q[head], q[tai
l])) head++;
    //if(tail - head + 1 <= 2); 无法构成半平面交
    q[tail + 1] = q[head]; // 半平面是q[head, tail]
    cnt = 0;
    for(int i = head; i <= tail; i++) p[++cnt] = GetIntersec
tion(q[i], q[i + 1]);
}
double area(Point * p, int cnt){ // 求凸包面积
    p[cnt + 1] = p[1];
    double ans = 0;
    for(int i = 2; i <= cnt; i++)
        ans += (p[i] - p[1]) ^ (p[i + 1] - p[1]);
    return fabs(ans) / 2;
}
int main(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){ // 输入一些直线
        int m; scanf("%d", &m);
        for(int j = 1; j <= m; j++)
            scanf("%lf%lf", &p[j].x, &p[j].y);
        p[m + 1] = p[1];
        for(int j = 1; j <= m; j++) l[++cnt] = Line(p[j], p
[j + 1]);
    }
    HALF();
    printf("%.3lf\n", area(p, cnt));
    return 0;
}
```

7.14 平面最近点对

见

7.15 点与直线

7.15.1 点关于直线对称

点(x, y)关于直线$Ax + By + C = 0$的对称点为

$$\begin{cases} X = x - 2A\dfrac{Ax+By+C}{A^2+B^2} \\ Y = y - 2B\dfrac{Ax+By+C}{A^2+B^2} \end{cases}$$

7.15.2 点到直线距离

求$p(x_0, y_0)$到直线AB的距离

$d = \frac{Ax_0+By_0+C}{\sqrt{A^2+B^2}}$

其中A，B，C用7.3表示即可

或者用$\frac{PA \times AB}{|AB|}$

```
double DistanceToLine(Point P, Point A, Point B){
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2)/Length(v1));
}
```

7.15.3 直线到线段距离

- 点在线段的投影在线段上，距离就是到直线的距离
- 点在线段的投影在线段外，距离就是到线段两端点近的距离

```
double DistanceToSegment(Point P, Point A, Point B){
    if(A == B) return Length(P - A);
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if(dcmp(Dot(v1, v2)) < 0)//离A近
        return Length(v2);
    if(dcmp(Dot(v1, v3)) > 0)//离B近
        return Length(v3);
    return DistanceToLine(P, A, B);
}
```

7.15.4 判断点在直线/线段/射线上

点在直线上， 构造$Ax + By = C$

```
bool check(Point a, Point b, Point c){// c是否在直线ab上
    double A = b.y - a.y, B = a.x - b.x, C = a.x * b.y - b.x
 * a.y;
    return sgn(A * c.x + B * b.y - C) == 0;
}
```

射线， 利用三点共线定理

```
bool check(Point a, Point b, Point c){// c是否在射线ab上
    if(a == c) return 1;
    Vector v = b - a;
    Vector p = c - a;
    if(sgn(p.x) == 0){
        double t1 = v.y / p.y;
        if(sgn(v.x) == 0 && sgn(t1) > 0) return 1;
    }
    if(sgn(p.y) == 0){
        double t2 = v.x / p.x;
        if(sgn(v.y) == 0 && sgn(t2) > 0) return 1;
    }
    double t1 = v.y / p.y;
    double t2 = v.x / p.x;
    if(sgn(t1 - t2) == 0 && sgn(t1) > 0) return 1;
    return 0;
}
```

在线段上
向量P1Q 和P2Q的叉积为0，点积小于等于0

```
bool OnSegment(Point P1, Point P2, Point Q){//Q是否在线段P1P2上
    return dcmp((P1 - Q) ^ (P2 - Q)) == 0 && dcmp((P1 - Q) * (P2 - Q)) <= 0;
}
```

7.16 圆

6.16.1 圆的表示

圆方程$(x-O_x)^2+(y-O_y)^2=r^2$

```
struct Circle {//圆
    Point o;
    double r;
    Circle(Point o, double r):o(o),r(r){}
    void read() { scanf("%lf%lf%lf", &o.x, &o.y, &r); }
};
```

三点确定一个圆
$Ax^2+Ay^2+Bx+Cy+D=0$

$$A=x_1\left(y_2-y_3\right)-y_1\left(x_2-x_3\right)+x_2y_3-x_3y_2$$
$$B=\left(x_1^2+y_1^2\right)\left(y_3-y_2\right)+\left(x_2^2+y_2^2\right)\left(y_1-y_3\right)+\left(x_3^2+y_3^2\right)\left(y_2-y_1\right)$$
$$C=\left(x_1^2+y_1^2\right)\left(x_2-x_3\right)+\left(x_2^2+y_2^2\right)\left(x_3-x_1\right)+\left(x_3^2+y_3^2\right)\left(x_1-x_2\right)$$
$$D=\left(x_1^2+y_1^2\right)\left(x_3y_2-x_2y_3\right)+\left(x_2^2+y_2^2\right)\left(x_1y_3-x_3y_1\right)+\left(x_3^2+y_3^2\right)\left(x_2y_1-x_1y_2\right)$$
$$x=\frac{\left(x_1^2+y_1^2\right)\left(y_2-y_3\right)+\left(x_2^2+y_2^2\right)\left(y_3-y_1\right)+\left(x_3^2+y_3^2\right)\left(y_1-y_2\right)}{2\left(x_1\left(y_2-y_3\right)-y_1\left(x_2-x_3\right)+x_2y_3-x_3y_2\right)}=-\frac{B}{2A}$$
$$y=\frac{\left(x_1^2+y_1^2\right)\left(x_3-x_2\right)+\left(x_2^2+y_2^2\right)\left(x_1-x_3\right)+\left(x_3^2+y_3^2\right)\left(x_2-x_1\right)}{2\left(x_1\left(y_2-y_3\right)-y_1\left(x_2-x_3\right)+x_2y_3-x_3y_2\right)}=-\frac{c}{2A}$$
$$r=\sqrt{(x-x_1)^2+\left(y-y_1\right)^2}=\sqrt{\frac{B^2+C^2-4AD}{4A^2}}$$

```
void getCircle(Point a, Point b, Point c) {
    double B = (a.x * a.x + a.y * a.y) * (b.y - c.y) + (b.x * b.x + b.y * b.y) * (c.y - a.y) + (c.x * c.x + c.y * c.y) * (a.y - b.y);
    double A = a.x * (b.y - c.y) - a.y * (b.x - c.x) + b.x * c.y - c.x * b.y;
    double C = (a.x * a.x + a.y * a.y) * (c.x - b.x) + (b.x * b.x + b.y * b.y) * (a.x - c.x) + (c.x * c.x + c.y * c.y) * (b.x - a.x);
    double x = - B / (2 * A);
```

```
        double y = - C / (2 * A);
    }
```

7.16.2 圆与直线交点

已知直线起始于(a_x, a_y)方向向量为(v_x, v_y)
那么直线坐标为

$$\begin{cases} x = v_x t + a_x \\ y = v_y t + a_y \end{cases}$$

圆方程$(x - O_x)^2 + (y - O_y)^2 = r^2$
联立方程得到关于t的一元二次方程
$(v_x^2 + v_y^2)t^2 + (2v_x a_x - 2v_x O_x + 2v_y a_y - 2v_y O_y)t + (a_x - O_x)^2 + (a_y - O_y)^2 - r^2 = 0$
根据方程$x = \frac{-b \pm \sqrt{\Delta}}{2a}$

如果有交点，那么根据直线坐标求即可

```
int getLineCircleIntersection (Point p, Vector v, Circle O,
double& t1, double& t2) {
    double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.
y;
    double e = a * a + c * c, f = 2 * (a * b + c * d), g = b
 * b + d * d - O.r * O.r;
    double delta = f * f - 4 * e * g;
    if (dcmp(delta) < 0) return 0;
    if (dcmp(delta) == 0) {
        t1 = t2 = -f / (2 * e);
        return 1;
    }

    t1 = (-f - sqrt(delta)) / (2 * e);
    t2 = (-f + sqrt(delta)) / (2 * e);
    return 2;
}
```

7.17 求面积

7.17.1 多边形求面积

把多边形看成cnt - 2个三角形，用外积求，三角形面积是$\frac{\alpha \times \beta}{2}$
先进行极角排序，然后求面积

```
double Cross(Point a1, Point a2, Point b1, Point b2){
    return (a2.x - a1.x) * (b2.y - b1.y) - (b2.x - b1.x) *
 (a2.y - a1.y);
}
double calArea(Point *p, int n){
    p[n + 1] = p[1];
    double ans = 0;
    for(int i = 1; i <= n; i++) {
        ans += (1ll * p[i].x * p[i + 1].y - 1ll * p[i + 1].x
 * p[i].y);
    }
    return fabs(ans) / 2;
}
```

```
double cal(){
    double ans = 0;
    for(int i = 1; i <= n; i++) {
        int j = i % n + 1;
        ans += p[i].x * p[j].y - p[i].y * p[j].x;
    }
    return fabs(ans / 2.0);
}
```

7.18 极角排序

7.19 点与多边形

判断一个点是否在任意多边形内部或者在边上
时间复杂度$O(n)$

```
bool OnSegment(Point P1, Point P2, Point Q){//点Q是否在线段p1
p2上
    return dcmp((P1 - Q) ^ (P2 - Q)) == 0 && dcmp((P1 - Q) *
 (P2 - Q)) <= 0;//叉积保持三点共线，点积保持p1,p2于Q两侧
}
bool InPolygon(Point Q, point *p){ // p是任意多边形，数组是按
照顺序的，顺逆都可
    bool flag = 0;
    Point P1, P2;
    for(int i = 1, j = n; i <= n; j = i++) {
        P1 = p[i];
        P2 = p[j];
        if(OnSegment(P1, P2, Q)) return 1; //点在多边形一条边
上
        if((dcmp(P1.y - Q.y) > 0 != dcmp(P2.y - Q.y) > 0) &&
 dcmp(Q.x - (Q.y - P1.y) * (P1.x - P2.x) / (P1.y - P2.y) - P
1.x) < 0)//p1,p2在射线上下侧，p1A斜率大于p1p2
            flag = !flag;
    }
    return flag;
}
```

判断一个点是否在凸多边形内

7.20 最小圆覆盖

给出n个点，画出一个最小的包含所有点的圆，求出圆的半径和圆心坐标
时间复杂度$O(n)$

```cpp
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const int N = 1e5 + 5;
const double eps = 1e-12;
struct Point{
    double x, y;
    Point (double x = 0, double y = 0):x(x),y(y){};
}p[N];
double dis(Point p1, Point p2){
    return sqrt((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.
y) * (p2.y - p1.y));
}
Point o; //圆心
double r; // 半径
void ToCircle(Point p1, Point p2, Point p3){ //三点定圆
    double a, b, c, d, e, f;
    a = p1.x - p2.x; b = p1.y - p2.y;
    c = p1.x - p3.x; d = p1.y - p3.y;
    e = (p1.x * p1.x - p2.x * p2.x) - (p2.y * p2.y - p1.y *
 p1.y);
    f = (p1.x * p1.x - p3.x * p3.x) - (p3.y * p3.y - p1.y *
 p1.y);
    o.x = (b * f - d * e) / (2 * b * c - 2 * a * d);
    o.y = (c * e - a * f) / (2 * b * c - 2 * a * d);
    r = dis(o, p1);//半径
}
int main(){
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) scanf("%lf%lf", &p[i].x, &p
[i].y);
    random_shuffle(p + 1, p + n + 1);
    o = p[1]; r = 0;
    for(int i = 2; i <= n; i++){
        if(dis(o, p[i]) > r + eps){
            o = p[i]; r = 0;
            for(int j = 1; j <= i - 1; j++){
                if(dis(o, p[j]) > r + eps){
                    o.x = (p[i].x + p[j].x) / 2;
                    o.y = (p[i].y + p[j].y) / 2;
                    r = dis(o, p[j]);
                    for(int k = 1; k <= j - 1; k++){
                        if(dis(o, p[k]) > r + eps){
                            ToCircle(p[i], p[j], p[k]);
                        }
                    }
                }
            }
        }
    }
    printf("%.2lf %.2lf %.2lf\n", o.x, o.y, r);
    return 0;
}
```

7.21 解析几何

7.21.1 圆锥

圆心在原点的圆锥，半径为r，高为h

$$\begin{cases} \dfrac{h-z}{h} = \dfrac{R}{r} \\ x^2 + y^2 = R^2 \end{cases}$$

7.21.2 余弦定理

$$\cos\alpha = \frac{b^2 + c^2 - a^2}{2bc}$$

$$\cos\alpha = \frac{\sin^2\beta + \sin^2\gamma - \sin^2\alpha}{2\sin\beta\sin\gamma}$$

7.22 正多边形

7.22.1 面积

边长为a，中心到顶点的距离是r
$cotx = \frac{1}{tanx}$

1. 用a表示$S = \frac{n}{4}\left(\frac{L}{n}\right)^2 \cot\frac{\pi}{n}$
2. 用r表示$S = \frac{1}{2}nr^2\sin\frac{2\pi}{n}$