

ACM 参赛模板

[Henan U: Noob Threesome](#)

2019-12-16

目录

图论.....	5
1. 费用流.....	5
1. Dijkstra_优化版本.....	7
2. 网络流.....	9
3. 树链剖分.....	10
3.1. 树链剖分（线段树 边权）.....	14
3.2 动态维护树的直径.....	20
4. 启发式合并.....	23
5. 二分图.....	24
5.1 KM 算法.....	24
数据结构.....	26
1. 分块.....	26
2. 主席树.....	27
2.1. 基于边权 主席树 + LCA.....	29
2.2. 主席树查询区间不同数个数.....	33
2.3 树状数组套主席树.....	35
3. 伸展树.....	37
4. Treap.....	41
5. 动态树（LCT）.....	47
4.1.1 Lazy 标记.....	49
6. 可持续化字典树.....	53
7. CDQ 分治.....	54
8. 点分治.....	57
字符串处理.....	59
1. Hash.....	59
2. 二维 Hash.....	61
3. unordered_map.....	62
4. 回文树.....	63
5. AC 自动机.....	64
6. 后缀自动机.....	67
7. 后缀数组.....	70
8. 后缀数组 DC3.....	72
9. 扩展 KMP.....	74
10. 字符串最小表示法.....	75
数学.....	76
1. 逆元.....	76
上面已经列出一个整数解的方法，在找到 $p * a + q * b = \text{Gcd}(p, q)$ 的一组解 p_0, q_0 后， $p * a + p * b = \text{Gcd}(p, q)$ 的其他整数解满足：.....	77
2. 阶乘的逆元.....	77
3. 组合数的一些性质.....	78

4. 扩展 Lucas 定理.....	81
5. 扩展中国剩余定理.....	82
6. 欧拉降幂.....	83
7. 容斥求 x 的因子 k	85
8. 多项式乘法.....	85
8.1. FFT.....	85
8.2. NTT.....	87
8.3. FWT.....	89
8.4. 原根.....	90
11. 三分.....	91
12. 大数.....	92
13. Pell 方程.....	103
14. 拉格朗日插值法.....	107
15. 杜教 BM.....	109
16. SG 函数.....	115
17. 线性基.....	116
18. BSGS.....	121
19. 二次同余式.....	126
20. 平方和公式.....	128
21. 杜教筛.....	129
22. 勾股数.....	134
23. 高斯消元.....	134
24. 斯特林数.....	139
贝尔数.....	140
25. 威尔逊定理和 Miller_Rabin.....	140
26. 卡特兰数.....	142
27. RSA 算法.....	143
28. 高次剩余.....	144
动态规划.....	144
1. 数位 dp.....	144
2. 斜率优化 dp.....	145
3. RMQ.....	149
4. 斯坦纳树.....	150
计算几何.....	152
1. 最小覆盖圆.....	152
2. 两球相交体积.....	154
3. 最大半径圆覆盖.....	154
4. 计算圆和多边形面积交.....	156
5. 三维凸包的模板.....	159
其他.....	165
1. 读入优化.....	165
2. 技巧优化.....	172
3. 取模优化.....	172

4. Java 快速读入.....	172
5. 二进制的高效位运算 __builtin_系列函数.....	174
6. 基姆拉尔森计算公式.....	174
7. 随机生成数.....	174
8. 二维 vector 的定义	175
9. 随机数.....	175

图论

1. 费用流

```
#define INF 1e18
const int N = 1e5 + 5;
int sp, tp; //sp 是源点, tp 是汇点 最小费用流
struct Node{
    int v, next;
    ll cost, cap;
}edge[N * 2];

struct node{
    int id, v;
}pre[N*2];

int head[N], cnt, vis[N];
ll dis[N];
void init() {
    memset(head, -1, sizeof(head));
    cnt = 0;
}

void add(int u, int v, ll cost, ll cap) {
    edge[cnt].v = v;
    edge[cnt].cap = cap;
    edge[cnt].cost = cost;
    edge[cnt].next = head[u];
    head[u] = cnt++;

    edge[cnt].v = u;
    edge[cnt].cap = 0;
    edge[cnt].cost = -cost;
    edge[cnt].next = head[v];
    head[v] = cnt++;
}

int spfa()
{
    queue<int> q;
    fill(dis, dis + N, INF);
    memset(vis, 0, sizeof(vis));
```

```

dis[sp] = 0;
q.push(sp);
while(!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = 0;
    for(int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].v;
        ll cap = edge[i].cap, cost = edge[i].cost;
        if(cap > 0 && dis[v] > dis[u] + cost) {
            dis[v] = dis[u] + cost;
            pre[v].v = u;
            pre[v].id = i;
            if(!vis[v])
                vis[v] = 1, q.push(v);
        }
    }
}
if(dis[tp] == INF) return 0;
return 1;
}

ll MCMF() {
    int flow = 0;
    ll minCost = 0;
    while(spfa()) {
        ll minFlow = INF + 1;
        for(int i = tp; i != sp; i = pre[i].v)
            minFlow = min(minFlow, edge[pre[i].id].cap);

        for(int i = tp; i != sp; i = pre[i].v) {
            int j = pre[i].id;
            edge[j].cap -= minFlow;
            edge[j ^ 1].cap += minFlow;
        }
        minCost += minFlow * dis[tp];
    }
    return minCost;
}

```

1. Dijkstra_优化版本

```
const int INF = 0x3f3f3f3f;
const int maxn = 1e4;
struct edge {
    int to, capacity, cost, rev;
    edge() {}
    edge(int to, int _capacity, int _cost, int _rev):
        to(to), capacity(_capacity), cost(_cost), rev(_rev) {}
};

struct Min_Cost_Max_Flow {
    int V, H[maxn + 5], dis[maxn + 5], PreV[maxn + 5], PreE[maxn + 5];
    vector<edge> G[maxn + 5];
    void Init(int n) {
        V = n;
        for (int i = 0; i <= V; ++i) G[i].clear();
    }
    void Add_Edge(int from, int to, int cap, int cost) {
        G[from].push_back(edge(to, cap, cost, G[to].size()));
        G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
    }
    int Min_cost_max_flow(int s, int t, int f, int& flow) {
        int res = 0; fill(H, H + 1 + V, 0);
        while (f) {
            priority_queue<pii, vector<pii>, greater<pii>> > q;
            fill(dis, dis + 1 + V, INF);
            dis[s] = 0; q.push(pii(0, s));
            while (!q.empty()) {
                pii now = q.top(); q.pop();
                int v = now.second;
                if (dis[v] < now.first) continue;
                for (int i = 0; i < G[v].size(); ++i) {
                    edge& e = G[v][i];
                    if (e.capacity > 0 && dis[e.to] > dis[v] + e.cost +
H[v] - H[e.to]) {
                        dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
                        PreV[e.to] = v;
                        PreE[e.to] = i;
                        q.push(pii(dis[e.to], e.to));
                    }
                }
            }
        }
        return res;
    }
};
```

```

        if (dis[t] == INF)break;
        for (int i = 0; i <= V; ++i)H[i] += dis[i];
        int d = f;
        for (int v = t; v != s; v = PreV[v])d = min(d,
G[PreV[v]][PreE[v]].capacity);
        f -= d; flow += d; res += d*H[t];
        for (int v = t; v != s; v = PreV[v]) {
            edge& e = G[PreV[v]][PreE[v]];
            e.capacity -= d;
            G[v][e.rev].capacity += d;
        }
    }
    return res;
}
};

int a[maxn];

int main() {
    int t; scanf("%d",&t);
    while(t --){
        int n,k; scanf("%d%d",&n,&k);
        rep(i,1,n + 1) scanf("%d",&a[i]);
        Min_Cost_Max_Flow MCMF;
        MCMF.Init(2 * n + 2);
        for(int i = 1;i <= n;++ i){
            MCMF.Add_Edge(0,i,1,0);
            MCMF.Add_Edge(i,n + i,1,-a[i]);
            MCMF.Add_Edge(n + i,2 * n + 1,1,0);
            for(int j = i + 1;j <= n;++ j){
                if(a[i] <= a[j])
                    MCMF.Add_Edge(n + i,j,1,0);
            }
        }
        MCMF.Add_Edge(2 * n + 1,2 * n + 2,k,0);
        printf("%d\n",-MCMF.Min_cost_max_flow(0,2 * n + 2,INF,k));
    }
    return 0;
}

```


2. 网络流

```
//网络流最大流 POJ 1273
#define INF 0x3f3f3f3f
using namespace std;
const int N = 1e5 + 5;
int sp, tp;

struct node{
    int v, next;
    ll cap;
}edge[N * 2];
int head[N], deg[N], cur[N];
int cnt = 0;

void init() {
    cnt = 0;
    memset(head, -1, sizeof(head));
}

void add(int u, int v, ll cap) {
    edge[cnt].v = v;
    edge[cnt].cap = cap;
    edge[cnt].next = head[u];
    head[u] = cnt++;

    edge[cnt].v = u;
    edge[cnt].cap = 0;
    edge[cnt].next = head[v];
    head[v] = cnt++;
}

int bfs() {
    memset(deg, -1, sizeof(deg));
    deg[sp] = 0;
    queue<int> q;
    q.push(sp);
    while(!q.empty()) {
        int u = q.front();
        if(u == tp) return 1;
        q.pop();
        for(int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].v;
```

```

        ll cap = edge[i].cap;
        if(deg[v] == -1 && cap) {
            deg[v] = deg[u] + 1;
            q.push(v);
        }
    }
}
return 0;
}

ll dfs(int u, ll flow) {
    if(u == tp || flow == 0) return flow;
    ll res = 0, f;
    for(int &i = cur[u]; ~i; i = edge[i].next) {
        int v = edge[i].v;
        ll cap = edge[i].cap;
        if(deg[v] == deg[u] + 1 && (f = dfs(v, min(flow - res, cap))) >
0)
        {
            edge[i].cap -= f;
            edge[i ^ 1].cap += f;
            res += f;
            if(res == flow) return flow;
        }
    }
    if(!res) deg[u] = -1;
    return res;
}

ll dinic() {
    ll ans = 0;
    while(bfs()) {
        memcpy(cur, head, sizeof(head));
        ans += dfs(sp, INF);
    }
    return ans;
}
}

```

3. 树链剖分

```

// luogu P3384
const int N = 1e5 + 5;

```

```

// 开的数组是真的多~~~!
int dep[N], fa[N], sz[N], w[N], son[N];
int a[N], sum[N << 2], id[N], top[N], cnt, mod;
int lazy[N << 2];
vector<int> q[N];

struct node {
    int l, r;
}tree[N << 2];

void dfs1(int u, int f, int deep) { //u 当前节点, f 父亲, deep 深度
    dep[u] = deep; fa[u] = f; sz[u] = 1;
    int maxSon = -1;
    for(auto v : q[u]) {
        if(v == f) continue;
        dfs1(v, u, deep + 1);
        sz[u] += sz[v];
        if(sz[v] > maxSon) maxSon = sz[v], son[u] = v; //标记每个非叶子节
点的重儿子编号
    }
}

void dfs2(int u, int topf) { //u 当前节点, topf 当前链的最顶端的节点
    id[u] = ++cnt; //标记每个点的新编号
    w[cnt] = a[u]; //把每个点的初始值赋到新编号上来
    top[u] = topf; //这个点所在链的顶端
    if(son[u]) dfs2(son[u], topf); //按先处理重儿子, 再处理轻儿子的顺序递
归处理
    for(auto v : q[u]) {
        if(v == fa[u] || v == son[u]) continue;
        dfs2(v, v); //对于每一个轻儿子都有一条从它自己开始的链
    }
}

//----- 以下为线段树
void pushUp(int i) {
    sum[i] = (sum[i << 1] + sum[i << 1 | 1]) % mod;
}

void build(int i, int l, int r) {
    tree[i] = {l, r};
    if(l == r) {
        sum[i] = w[l] % mod;
        return ;
    }
}

```

```

    }
    int mid = (l + r) >> 1;
    build(i << 1, l, mid);
    build(i << 1 | 1, mid + 1, r);
    pushUp(i);
}

void sl(int i, int val, int m) {
    lazy[i] = (lazy[i] + val) % mod;
    sum[i] = (sum[i] + 1LL * m * val % mod) % mod;
}

void pushDown(int i, int m) {
    if(lazy[i] == 0) return ;
    int val = lazy[i];
    sl(i << 1, val, m - (m >> 1));
    sl(i << 1 | 1, val, m >> 1);
    lazy[i] = 0;
}

int query(int i, int l, int r) {
    if(tree[i].l == l && tree[i].r == r)
        return sum[i];
    pushDown(i, tree[i].r - tree[i].l + 1);
    int mid = (tree[i].l + tree[i].r) >> 1;
    if(r <= mid) return query(i << 1, l, r);
    else if(l > mid) return query(i << 1 | 1, l, r);
    else {
        return (query(i << 1, l, mid) + query(i << 1 | 1, mid + 1,
r)) % mod;
    }
}

void update(int i, int l, int r, int v) {
    if(tree[i].l == l && tree[i].r == r) {
        sl(i, v, r - l + 1);
        return ;
    }
    pushDown(i, tree[i].r - tree[i].l + 1);
    int mid = (tree[i].l + tree[i].r) >> 1;
    if(r <= mid) update(i << 1, l, r, v);
    else if(l > mid) update(i << 1 | 1, l, r, v);
    else {

```

```

        update(i << 1, l, mid, v); update(i << 1 | 1, mid + 1, r, v);
    }
    pushUp(i);
}
//-----以上为线段树
int qRange(int x, int y) {
    int ans = 0;
    while(top[x] != top[y]) { //当两个点不在同一条链上
        if(dep[top[x]] < dep[top[y]]) swap(x, y); //把 x 点改为所在链顶端的
        //深度更深的那个点
        ans += query(1, id[top[x]], id[x]); //ans 加上 x 点到 x 所在链顶端
        //这一段区间的点权和
        ans %= mod;
        x = fa[top[x]]; //把 x 跳到 x 所在链顶端的那个点的上面一个点
    }
    //直到两个点处于一条链上
    if(dep[x] > dep[y]) swap(x, y); //把 x 点深度更深的那个点
    ans += query(1, id[x], id[y]); //这时再加上此时两个点的区间和即可
    return ans % mod;
}

void updRange(int x, int y, int k) { //同上
    k %= mod;
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        update(1, id[top[x]], id[x], k);
        x = fa[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    update(1, id[x], id[y], k);
}

int qSon(int x) { //子树区间右端点为 id[x]+siz[x]-1
    return query(1, id[x], id[x] + sz[x] - 1) % mod;
}

void updSon(int x, int k) { //同上
    update(1, id[x], id[x] + sz[x] - 1, k%mod);
}

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

```

```

for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
for(int i = 1; i < n; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    q[u].push_back(v);
    q[v].push_back(u);
}
dfs1(rt, 0, 1);
dfs2(rt, 0);
build(1, 1, n);
while(m--) {
    int op, l, r, k;
    scanf("%d %d", &op, &l);
    if(op == 4) printf("%d\n", qSon(l));
    else if(op == 1){
        scanf("%d %d", &r, &k);
        updRange(l, r, k);
    }
    else if(op == 2) {
        scanf("%d", &r);
        printf("%d\n", qRange(l, r));
    }
    else {
        scanf("%d", &k);
        updSon(l, k);
    }
}
return 0;
}

```

3.1. 树链剖分（线段树 边权）

```

const int N = 1e5 + 5;
struct Edge {
    int v, c;
};

vector<Edge> q[N];
int son[N], sz[N], fa[N], dep[N], top[N], id[N], cnt = 0, p[N], cc[N];

void dfs1(int u, int f, int deep) {
    dep[u] = deep;

```

```

fa[u] = f;
sz[u] = 1;
int maxSon = -1;
for(auto te : q[u]) {
    int v = te.v;
    if(v == f) continue;
    dfs1(v, u, deep + 1);
    sz[u] += sz[v];
    if(sz[v] > maxSon) {
        maxSon = sz[v];
        son[u] = v;
        cc[u] = te.c;
    }
}
}

void dfs2(int u, int topf, int c) {
    id[u] = ++cnt;
    p[cnt] = c;
    top[u] = topf;
    if(!son[u]) return ;
    dfs2(son[u], topf, cc[u]);
    for(auto te : q[u]) {
        int v = te.v, c = te.c;
        if(v == fa[u] || v == son[u]) continue;
        dfs2(v, v, c);
    }
}

struct Seg {
    struct node {
        int l, r;
    }tree[N << 2];

    int mx[N << 2], lazy[N << 2], lazy1[N << 2];
    // lazy-> 区间加 lazy1 区间覆盖

    void pushUp(int i) {
        mx[i] = max(mx[i << 1], mx[i << 1 | 1]);
    }

    void build(int i, int l, int r) {
        tree[i] = {l, r};
    }
}

```

```

        lazy[i] = lazy1[i] = 0;
        if(l == r) {
            mx[i] = p[l];
            return ;
        }
        int mid = (l + r) >> 1;
        build(i << 1, l, mid);
        build(i << 1 | 1, mid + 1, r);
        pushUp(i);
    }

    void sl(int i, int val) {
        if(lazy1[i]) lazy1[i] += val;
        lazy[i] += val;
        mx[i] += val;
    }

    void sl2(int i, int val) {
        lazy[i] = 0;
        lazy1[i] = val;
        mx[i] = val;
    }

    void pushDown(int i) {
        if(lazy[i]) {
            int val = lazy[i];
            sl(i << 1, val);
            sl(i << 1 | 1, val);
            lazy[i] = 0;
        }

        if(lazy1[i]) {
            int val = lazy1[i];
            sl2(i << 1, val);
            sl2(i << 1 | 1, val);
            lazy1[i] = 0;
        }
    }

    int query_Max(int i, int l, int r) {
        if(tree[i].l == l && tree[i].r == r)
            return mx[i];
        pushDown(i);
        int mid = (tree[i].l + tree[i].r) >> 1;

```



```

        if(r <= mid) return query_Max(i << 1, l, r);
        else if(l > mid) return query_Max(i << 1 | 1, l, r);
        else {

                return max(query_Max(i << 1, l, mid),
query_Max(i << 1 | 1, mid + 1, r));
        }
}

void Add(int i, int l, int r, int v) {
    if(tree[i].l == l && tree[i].r == r) {
        sl(i, v);
        return ;
    }
    pushDown(i);
    int mid = (tree[i].l + tree[i].r) >> 1;
    if(r <= mid) Add(i << 1, l, r, v);
    else if(l > mid) Add(i << 1 | 1, l, r, v);
    else {
        Add(i << 1, l, mid, v); Add(i << 1 | 1, mid + 1,
r, v);
    }
    pushUp(i);
}

void Cover(int i, int l, int r, int v) {
    if(tree[i].l == l && tree[i].r == r) {
        sl2(i, v);
        return ;
    }
    pushDown(i);

    int mid = (tree[i].l + tree[i].r) >> 1;
    if(r <= mid) Cover(i << 1, l, r, v);
    else if(l > mid) Cover(i << 1 | 1, l, r, v);
    else {
        Cover(i << 1, l, mid, v); Cover(i << 1 | 1, mid
+ 1, r, v);
    }
    pushUp(i);
}

```

```

}seg;

pair<int, int> pa[N];

void Cover(int x, int y, int k) { //区间覆盖
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        seg.Cover(1, id[top[x]], id[x], k);
        x = fa[top[x]];
    }
    if(x == y) return ;
    if(dep[x] > dep[y]) swap(x, y);
    seg.Cover(1, id[x] + 1, id[y], k);
}

void Add(int x, int y, int k) { //区间加
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        seg.Add(1, id[top[x]], id[x], k);
        x = fa[top[x]];
    }
    if(x == y) return ;
    if(dep[x] > dep[y]) swap(x, y);
    seg.Add(1, id[x] + 1, id[y], k);
}

int Max(int x, int y) { //区间最大值
    int ans = -1;
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        ans = max(ans, seg.query_Max(1, id[top[x]], id[x]));
        x = fa[top[x]];
    }
    if(x == y) return ans;
    if(dep[x] > dep[y]) swap(x, y);
    ans = max(ans, seg.query_Max(1, id[x] + 1, id[y]));
    return ans;
}

int main()
{

```

```

int n;
scanf("%d", &n);
for(int i = 1; i < n; i++) {
    int u, v, c;
    scanf("%d %d %d", &u, &v, &c);
    pa[i] = {u, v};
    q[u].push_back({v, c});
    q[v].push_back({u, c});
}

dfs1(1, 0, 1);
dfs2(1, 1, 0);
//cout << "-----";
seg.build(1, 1, n);
char op[15];
// for(int i = 1; i <= n; i++) {
//     cout << i << " : " << id[i] << endl;
// }
while(~scanf(" %s", op)) {
    if(op[0] == 'S') break;
    int l, r, v;
    scanf("%d %d", &l, &r);
    if(op[1] == 'o') {
        scanf("%d", &v);
        Cover(l, r, v);
    }
    else if(op[0] == 'A') {
        scanf("%d", &v);
        Add(l, r, v);
    }
    else if(op[0] == 'C') {
        int u = pa[l].first, v = pa[l].second;
        Cover(u, v, r);
    }
    else {
        printf("%d\n", Max(l, r));
    }
}

return 0;
}

```

3.2 动态维护树的直径

```
struct node {
    int v; ll c;
};

vector<node> q[N];

struct Bit {
    int n; ll a[N];
    void init(int nn) { n = nn; for(int i = 0; i <= n; i++) a[i] = 0; }
    int lowbits(int x) { return x & -x; }
    void add(int x, int v) {
        for(; x <= n; x += lowbits(x)) a[x] += v;
    }
    void add(int x, int y, int v) {
        add(x, v); add(y+1, -v);
    }
    ll sum(int x) {
        ll ans = 0;
        if(x <= 0) return 0;
        for(; x > 0; x -= lowbits(x)) ans += a[x];
        return ans;
    }
}B;

int dep[N], in[N], out[N], top[N], cnt, fa[N], sz[N], son[N], w[N];

void dfs1(int u, int f, int deep) {
    dep[u] = deep; fa[u] = f; sz[u] = 1;
    int mxSz = -1;
    for(auto & te : q[u]) {
        int v = te.v;
        if(v == f) continue;
        dfs1(v, u, deep + 1);
        sz[u] += sz[v];
        if(sz[v] > mxSz) mxSz = sz[v], son[u] = v;
    }
}

void dfs2(int u, int topf) {
    in[u] = ++cnt;    w[cnt] = u; top[u] = topf;
```

```

    if(son[u]) dfs2(son[u], topf);
    for(auto & te : q[u]) {
        int v = te.v;
        if(v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
    out[u] = cnt;
}

int Lca(int x, int y) {
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    return x;
}

ll getDis(int x, int y) {
    return B.sum(in[x]) + B.sum(in[y]) - 2ll * B.sum(in[Lca(x, y)]);
}

struct Seg {
    struct node { int l, r; }a[N << 2];
    struct Node { int u, v; ll mx; }b[N << 2];

    void pushUp(int i) {
        int x = b[i << 1].u, y = b[i << 1].v;
        int u = b[i << 1 | 1].u, v = b[i << 1 | 1].v;
        if(b[i << 1].mx > b[i << 1 | 1].mx) b[i] = b[i << 1];
        else b[i] = b[i << 1 | 1];
        ll tmp;
        if((tmp = getDis(x, v)) > b[i].mx) b[i] = {x, v, tmp};
        if((tmp = getDis(x, u)) > b[i].mx) b[i] = {x, u, tmp};
        if((tmp = getDis(y, v)) > b[i].mx) b[i] = {y, v, tmp};
        if((tmp = getDis(y, u)) > b[i].mx) b[i] = {y, u, tmp};
    }

    void build(int i, int l, int r) {
        a[i] = {l, r};
        if(l == r) {
            b[i] = {w[l], w[l], 0};
            return ;
        }
    }
}

```

```

        int mid = (l + r) >> 1;
        build(i << 1, l, mid);
        build(i << 1 | 1, mid + 1, r);
        pushUp(i);
    }

    void update(int i, int l, int r) {
        if(a[i].l == l && a[i].r == r) return ;
        int mid = (a[i].l + a[i].r) >> 1;
        if(r <= mid) update(i << 1, l, r);
        else if(l > mid) update(i << 1 | 1, l, r);
        else {
            update(i << 1, l, mid);
            update(i << 1 | 1, mid + 1, r);
        }
        pushUp(i);
    }
}seg;

struct Edge {
    int u, v; ll c;
}e[N];

int main()
{
    int n, m, u, v, c;
    scanf("%d", &n);
    B.init(n);
    for(int i = 1; i < n; i++) {
        scanf("%d %d %d", &u, &v, &c);
        q[u].push_back({v, c});
        q[v].push_back({u, c});
        e[i] = {u, v, c};
    }
    dfs1(1, 0, 1);
    dfs2(1, 0);
    for(int i = 1; i < n; i++) {
        if(dep[e[i].u] < dep[e[i].v]) swap(e[i].u, e[i].v);
        B.add(in[e[i].u], out[e[i].u], e[i].c);
    }
    seg.build(1, 1, n);
    char op;
    scanf("%d", &m);
    while(m--) {

```

```

scanf(" %c %d", &op, &u);
if(op == 'C') {
    scanf("%d", &v);
    B.add(in[e[u].u], out[e[u].u], v - e[u].c);
    e[u].c = v;
    seg.update(1, in[e[u].u], out[e[u].u]);
}
else {
    int x = seg.b[1].u, y = seg.b[1].v;
    printf("%lld\n", max(getDis(x, u), getDis(y, u)));
}
}
return 0;
}

```

4. 启发式合并

```

const int MAXN = 1e5 + 10;
inline int read() {
    char c = getchar(); int x = 0, f = 1;
    while(c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}
    while(c >= '0' && c <= '9') x = x * 10 + c - '0', c = getchar();
    return x * f;
}
int N, col[MAXN], son[MAXN], siz[MAXN], cnt[MAXN], Mx, Son;
LL sum = 0, ans[MAXN];
vector<int> v[MAXN];
void dfs(int x, int fa) {
    siz[x] = 1;
    for(int i = 0; i < v[x].size(); i++) {
        int to = v[x][i];
        if(to == fa) continue;
        dfs(to, x);
        siz[x] += siz[to];
        if(siz[to] > siz[son[x]]) son[x] = to; //轻重链剖分
    }
}
void add(int x, int fa, int val) {
    cnt[col[x]] += val; //这里可能会因题目而异
    if(cnt[col[x]] > Mx) Mx = cnt[col[x]], sum = col[x];
}

```

```

else if(cnt[col[x]] == Mx) sum += (LL)col[x];
for(int i = 0; i < v[x].size(); i++) {
    int to = v[x][i];
    if(to == fa || to == Son) continue;
    add(to, x, val);
}
}
void dfs2(int x, int fa, int opt) {
    for(int i = 0; i < v[x].size(); i++) {
        int to = v[x][i];
        if(to == fa) continue;
        if(to != son[x]) dfs2(to, x, 0); //暴力统计轻边的贡献, opt = 0 表示递归完成后消除对该点的影响
    }
    if(son[x]) dfs2(son[x], x, 1), Son = son[x]; //统计重儿子的贡献, 不消除影响

    add(x, fa, 1); Son = 0; //暴力统计所有轻儿子的贡献
    ans[x] = sum; //更新答案
    if(!opt) add(x, fa, -1), sum = 0, Mx = 0; //如果需要删除贡献的话就删掉
}
int main() {
    N = read();
    for(int i = 1; i <= N; i++) col[i] = read();
    for(int i = 1; i <= N - 1; i++) {
        int x = read(), y = read();
        v[x].push_back(y); v[y].push_back(x);
    }
    dfs(1, 0);
    dfs2(1, 0, 0);
    for(int i = 1; i <= N; i++) printf("%I64d ", ans[i]);
    return 0;
}

```

5. 二分图

5.1 KM 算法

```

// w[i][j] 代表二分图 i -> j 的价值 调用 km()
#define ll long long
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int N = 605;

```



```

int n, linker[N];
ll lx[N], ly[N], slack[N], pre[N];
ll w[N][N], a[N], b[N], c[N], d[N];
bool visy[N];

void bfs(int k) {
    int x, y = 0, yy = 0; ll delta;
    memset(pre, 0, sizeof(pre));
    for(int i = 1; i <= n; i++) slack[i] = INF;
    linker[y] = k;
    while(1) {
        x = linker[y]; delta = INF; visy[y] = true;
        for(int i = 1; i <= n; i++) {
            if(!visy[i]) {
                if(slack[i] > lx[x] + ly[i] - w[x][i]) {
                    slack[i] = lx[x] + ly[i] - w[x][i];
                    pre[i] = y;
                }
                if(slack[i] < delta) delta = slack[i], yy = i;
            }
        }
        for(int i = 0; i <= n; i++) {
            if(visy[i]) lx[linker[i]] -= delta, ly[i] += delta;
            else slack[i] -= delta;
        }
        y = yy;
        if(linker[y] == -1) break;
    }
    while(y) linker[y] = linker[pre[y]], y = pre[y];
}

ll KM() {
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    memset(linker, -1, sizeof(linker));
    for(int i = 1; i <= n; i++) {
        memset(visy, false, sizeof(visy));
        bfs(i);
    }
    ll res = 0;
    for(int i = 1; i <= n; i++) if(linker[i] != -1)
        res += w[linker[i]][i];
}

```

```
    return res;
}
```

数据结构

1. 分块

```
const int N = 5e4 + 5;
int l[N], r[N], bl[N];
int block, n, m, num;
// num 分块个数
// block 每块大小
// bl[i] 第i个数属于那个分块
void build() {
    block = sqrt(n);
    num = n / block; if(n % block) num++;
    for(int i = 1; i <= num; i++)
        l[i] = (i - 1) * block + 1, r[i] = i * block;
    r[num] = n;
    for(int i = 1; i <= n; i++)
        bl[i] = (i - 1) / block + 1;
}

ll a[N], b[N];

void update(int L, int R, int v) {
    if(bl[L] == bl[R]) {
        for(int i = L; i <= R; i++) a[i] += v;
        return ;
    }
    for(int i = L; i <= r[bl[L]]; i++) a[i] += v;
    for(int i = bl[L] + 1; i < bl[R]; i++) b[i] += v;
    for(int i = l[bl[R]]; i <= R; i++) a[i] += v;
}

ll ask(int idx) {
    return a[idx] + b[bl[idx]];
}
```

```

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    build();
    for(int i = 1; i <= n; i++) {
        int op, l, r, v;
        scanf("%d %d %d %d", &op, &l, &r, &v);
        if(op == 0) update(l, r, v);
        else {
            printf("%lld\n", ask(r));
        }
    }
    return 0;
}

```

2. 主席树

```

// 主席树，求区间第k小
const int N = 1e5 + 5;
struct Tree {
    struct node {
        int l, r;
    }a[N * 36];
    int ls[N * 36], rs[N * 36], sum[N * 36], rt[N * 36];
    int tot;
    void init() {
        tot = 0;
    }
    void build(int &o, int i, int l, int r) {
        o = ++tot;
        sum[o] = 0;
        a[i] = {l, r};
        if(l == r) return ;
        int mid = (l + r) >> 1;
        build(ls[o], i << 1, l, mid);
        build(rs[o], i << 1 | 1, mid + 1, r);
    }
    void update(int &o, int i, int last, int p) {
        o = ++tot;
        ls[o] = ls[last];

```

```

        rs[o] = rs[last];
        sum[o] = sum[last] + 1;
        if(a[i].l == a[i].r) return ;
        int mid = (a[i].l + a[i].r) >> 1;
        if(p <= mid) update(ls[o], i << 1, ls[last], p);
        else update(rs[o], i << 1 | 1, rs[last], p);
    }
    int query(int ss, int tt, int i, int k) {
        if(a[i].l == a[i].r) return a[i].l;
        int cnt = sum[ls[tt]] - sum[ls[ss]];
        if(k <= cnt) return query(ls[ss], ls[tt], i << 1, k);
        else return query(rs[ss], rs[tt], i << 1 | 1, k - cnt);
    }
    int Query(int ss, int tt, int i, int l, int r) {
        if(a[i].l == l && a[i].r == r) return sum[tt] - sum[ss];
        int mid = (a[i].l + a[i].r) >> 1;
        if(r <= mid) return Query(ls[ss], ls[tt], i << 1, l, r);
        else if(l > mid) return Query(rs[ss], rs[tt], i << 1 | 1, l, r);
        else {
            return Query(ls[ss], ls[tt], i << 1, l, mid) +
                   Query(rs[ss], rs[tt], i << 1 | 1, mid + 1, r);
        }
    }
}tree;

int a[N], b[N];

void work() {
    int l, r, k;
    scanf("%d %d %d", &l, &r, &k);
    int idx = tree.query(tree.rt[l - 1], tree.rt[r], 1, k);
    printf("%d\n", b[idx]);
}

int main()
{
    int T;
    scanf("%d", &T);
    while(T--) {
        tree.init();
        int n, q;
        scanf("%d %d", &n, &q);
        for(int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);

```

```

        b[i] = a[i];
    }
    sort(b + 1, b + n + 1);
    int cnt = unique(b + 1, b + n + 1) - b - 1;
    for(int i = 1; i <= n; i++) a[i] = lower_bound(b + 1, b + cnt +
1, a[i]) - b;
    tree.build(tree.rt[0], 1, 1, cnt);
    for(int i = 1; i <= n; i++) tree.update(tree.rt[i], 1, tree.rt[i
- 1], a[i]);
    while(q-- > 0) work();

}
return 0;
}
/*
1
10 10
1 4 2 3 5 6 7 8 9 0
1 3 2

*/

```

2.1. 基于边权 主席树 + LCA

```

const int N = 2e5 + 5;
struct Tree {
    struct node {
        int l, r;
    }a[N * 36];
    int rt[N * 36], ls[N * 36], rs[N * 36], sum[N * 36], tot;

    void init() { tot = 0; }

    void build(int &o, int i, int l, int r) {
        o = ++tot;
        sum[o] = 0;
        a[i] = {l, r};
        if(l == r) return ;
        int mid = (l + r) >> 1;
        build(ls[o], i << 1, l, mid);
    }
}

```

```

        build(rs[o], i << 1 | 1, mid + 1, r);
    }

    void update(int &o, int i, int last, int p) {
        o = ++tot;
        ls[o] = ls[last];
        rs[o] = rs[last];
        sum[o] = sum[last] + 1;
        if(a[i].l == a[i].r) return ;
        int mid = (a[i].l + a[i].r) >> 1;
        if(p <= mid) update(ls[o], i << 1, ls[last], p);
        else update(rs[o], i << 1 | 1, rs[last], p);
    }

    int query(int ss, int tt, int i, int k) {
        if(a[i].l == a[i].r) return a[i].l;
        int cnt = sum[ls[tt]] - sum[ls[ss]];
        if(k <= cnt) return query(ss, tt, i << 1, k);
        else return query(ss, tt, i << 1 | 1, k - cnt);
    }

    int Query(int ss, int tt, int i, int k) {
        if(a[i].r <= k) return sum[tt] - sum[ss];
        int mid = (a[i].l + a[i].r) >> 1;
        if(k <= mid) return Query(ls[ss], ls[tt], i << 1, k);
        else return sum[ls[tt]] - sum[ls[ss]] + Query(rs[ss], rs[tt], i
<< 1 | 1, k);
    }

    int Query2(int x, int y, int fa, int i, int k) { // 主席树 求 (x,
y) 路径上的边权 [1, k] 的个数
        if(a[i].r <= k) return sum[x] + sum[y] - 2*sum[fa];
        int mid = (a[i].l + a[i].r) >> 1;
        int ans = Query2(ls[x], ls[y], ls[fa], i << 1, k);
        if(k > mid) ans += Query2(rs[x], rs[y], rs[fa], i << 1 | 1, k);
        return ans;
    }
}tree;

inline int read() {
    char c = getchar(); int x = 0, f = 1;
    while(!isdigit(c)) { if(c == '-') f = -1; c = getchar();}
    while(isdigit(c)) { x = x * 10 + (c - '0'); c = getchar();}
    return x * f;
}

```

```

}

struct node {
    int v, w;
};

struct Edge {
    int u, v, w;
}edge[N], qu[N];

vector<node> q[N];

int a[N], dep[N], dp[N][25];

void dfs(int u, int f, int w, int deep) {
    if(f) tree.update(tree.rt[u], 1, tree.rt[f], w);
    dep[u] = deep;
    for(auto te : q[u]) {
        int v = te.v, w = te.w;
        if(v == f) continue;
        dfs(v, u, w, deep + 1);
        dp[v][0] = u;
    }
}

int n, m, cnt = 0;
void init() {
    for(int j = 1; j <= 20; j++) {
        for(int i = 1; i <= n; i++) {
            dp[i][j] = dp[dp[i][j-1]][j-1];
        }
    }
}

int Lca(int x, int y) {
    if(dep[x] < dep[y]) swap(x, y);
    int dis = dep[x] - dep[y];
    for(int i = 20; i >= 0; i--) {
        if(dis & (1 << i)) x = dp[x][i];
    }
    if(x == y) return x;
    for(int i = 20; i >= 0; i--) {
        if(dp[x][i] != dp[y][i]) {
            x = dp[x][i];
            y = dp[y][i];
        }
    }
    return dp[x][0];
}

```

```

    }
}
return dp[x][0];
}

int get(int u, int v, int k) {
    if(u == v) return 0;
    return tree.Query(tree.rt[u], tree.rt[v], 1, k);
}

int main()
{
    n = read(), m = read();
    for(int i = 1; i < n; i++) {
        int u, v, w;
        u = read(); v = read(); w = read();
        edge[i] = {u, v, w};
        a[++cnt] = w;
    }

    for(int i = 1; i <= m; i++){
        int u, v, w;
        u = read(); v = read(); w = read();
        qu[i] = {u, v, w};
        a[++cnt] = w;
    }
    sort(a + 1, a + cnt + 1);
    cnt = unique(a + 1, a + cnt + 1) - a - 1;
    tree.init();
    tree.build(tree.rt[1], 1, 1, cnt);

    for(int i = 1; i < n; i++) {
        int w = lower_bound(a + 1, a + cnt + 1, edge[i].w) - a;
        int u = edge[i].u, v = edge[i].v;
        q[u].push_back({v, w});
        q[v].push_back({u, w});
    }
    dfs(1, 0, 0, 1);
    dp[1][0] = 1;
    init();
    for(int i = 1; i <= m; i++) {
        int w = lower_bound(a + 1, a + cnt + 1, qu[i].w) - a;
        int u = qu[i].u, v = qu[i].v;

```



```

        int lca = Lca(u, v);
        //int ans = get(lca, u, w) + get(lca, v, w);
        printf("%d\n", tree.Query2(tree.rt[u], tree.rt[v], tree.rt[lca],
1, w));
    }

    return 0;
}

/*
3 3
1 3 2
2 3 4
1 3 1
1 2 3
1 2 4

*/

```

2.2. 主席树查询区间不同数个数

```

const int N = 1e5 + 5;
struct Tree {
    struct node {
        int l, r;
    }a[N * 36];
    int sum[N * 36], ls[N * 36], rs[N * 36], rt[N * 36], tot;

    void init() { tot = 0; }
    void build(int &o, int i, int l, int r) {
        o = ++tot;
        sum[o] = 0;
        a[i] = {l, r};
        if(l == r) return ;
        int mid = (l + r) >> 1;
        build(ls[o], i << 1, l, mid);
        build(rs[o], i << 1 | 1, mid + 1, r);
    }
}

```

```

void update(int &o, int i, int last, int pos, int v) {
    o = ++tot;
    ls[o] = ls[last];
    rs[o] = rs[last];
    sum[o] = sum[last] + v;
    if(a[i].l == a[i].r) return ;
    int mid = (a[i].l + a[i].r) >> 1;
    if(pos <= mid) update(ls[o], i << 1, ls[last], pos, v);
    else update(rs[o], i << 1 | 1, rs[last], pos, v);
}

int query(int i, int pos, int t) {
    if(a[i].l == a[i].r) return sum[t];
    int mid = (a[i].l + a[i].r) >> 1;
    if(pos <= mid) return sum[rs[t]] + query(i << 1, pos, ls[t]);
    else return query(i << 1 | 1, pos, rs[t]);
}
}tree;

int f[N * 10 + 100];

int main()
{
    int n, m;
    scanf("%d", &n);
    tree.init();
    tree.build(tree.rt[0], 1, 1, n);
    for(int i = 1; i <= n; i++) {
        int v;
        scanf("%d", &v);
        if(!f[v]) {
            tree.update(tree.rt[i], 1, tree.rt[i-1], i, 1);
        }
        else {
            int tmp;
            tree.update(tmp, 1, tree.rt[i-1], f[v], -1);
            tree.update(tree.rt[i], 1, tmp, i, 1);
        }
        f[v] = i;
    }
    scanf("%d", &m);
    for(int i = 1; i <= m; i++) {
        int l, r;
        scanf("%d %d", &l, &r);
    }
}

```

```

        printf("%d\n", tree.query(1, l, tree.rt[r]));
    }
    return 0;
}

```

2.3 树状数组套主席树

```

// luogu P2617
const int N = 2e5 + 5;

struct Tree {
    struct node {
        int l, r;
    }a[N << 2];
    int ls[N * 400], rs[N * 400], rt[N], sum[N * 400], tot;
    int R[105], L[105], c_l, c_r, n;

    int lowbits(int x) { return x & -x;}

    void init(int nn) { tot = 0; n = nn;}

    void build(int i, int l, int r) {
        a[i] = {l, r};
        if(l == r) return ;
        int mid = (l + r) >> 1;
        build(i << 1, l, mid);
        build(i << 1 | 1, mid + 1, r);
    }

    void update(int &o, int i, int pos, int v) {
        if(!o) o = ++tot;
        sum[o] += v;
        if(a[i].l == a[i].r) return ;
        int mid = (a[i].l + a[i].r) >> 1;
        if(pos <= mid) update(ls[o], i << 1, pos, v);
        else update(rs[o], i << 1 | 1, pos, v);
    }

    void p_update(int pos, int val, int v) {
        for(int i = pos; i <= n; i += lowbits(i)) update(rt[i], 1,
val, v);
    }
}

```

```

int query(int i, int k) {
    if(a[i].l == a[i].r) return a[i].l;
    int n_sum = 0;
    for(int j = 1; j <= c_r; j++) n_sum += sum[ls[R[j]]];
    for(int j = 1; j <= c_l; j++) n_sum -= sum[ls[L[j]]];
    if(n_sum < k) {
        k -= n_sum;
        for(int j = 1; j <= c_r; j++) R[j] = rs[R[j]];
        for(int j = 1; j <= c_l; j++) L[j] = rs[L[j]];
        return query(i << 1 | 1, k);
    }
    else {
        for(int j = 1; j <= c_r; j++) R[j] = ls[R[j]];
        for(int j = 1; j <= c_l; j++) L[j] = ls[L[j]];
        return query(i << 1, k);
    }
}

int p_query(int l, int r, int k) {
    c_l = c_r = 0;
    for(int i = r; i; i -= lowbits(i)) R[++c_r] = rt[i];
    for(int i = l - 1; i; i -= lowbits(i)) L[++c_l] = rt[i];
    return query(1, k);
}

}tree;
int a[N], b[N];
int n, m, cnt, num;

void _hash() {
    sort(b + 1, b + num + 1);
    cnt = unique(b + 1, b + num + 1) - b - 1;
    for(int i = 1; i <= n; i++) a[i] = lower_bound(b + 1, b + cnt + 1,
a[i]) - b;
}

struct Lx {
    char op;
    int l, r, k;
}p[N];

int main()
{

```

```

scanf("%d %d", &n, &m);
for(int i = 1; i <= n; i++) scanf("%d", &a[i]), b[i] = a[i];
num = n;
char op; int l, r, k;
for(int i = 1; i <= m; i++) {
    scanf(" %c %d %d", &op, &l, &r);
    if(op == 'Q') {
        scanf("%d", &k);
        p[i] = {op, l, r, k};
    }
    else {
        p[i] = {op, l, r};
        b[++num] = r;
    }
}
_hash();
tree.init(cnt);
tree.build(1, 1, cnt);
for(int i = 1; i <= n; i++) tree.p_update(i, a[i], 1);
for(int i = 1; i <= m; i++) {
    op = p[i].op, l = p[i].l, r = p[i].r, k = p[i].k;
    if(op == 'Q') {
        printf("%d\n", b[tree.p_query(l, r, k)]);
    }
    else {
        tree.p_update(l, a[l], -1);
        a[l] = lower_bound(b + 1, b + cnt + 1, r) - b;
        tree.p_update(l, a[l], 1);
    }
}

return 0;
}

```

3. 伸展树

```

#define lc c[x][0]
#define rc c[x][1]
const int N = 2e5 + 5;

```

```

int n, m, x, y;
namespace Splay_Tree{
    int c[N][2], f[N], val[N], cnt[N], sz[N], rev[N], ncnt, root;

    bool ck(int x) {
        return c[f[x]][1] == x;
    }

    void pushUp(int x) {
        sz[x] = sz[lc] + sz[rc] + cnt[x];
    }

    void pushDown(int x) {
        if(rev[x]) {
            swap(lc, rc);
            rev[lc] ^= 1;
            rev[rc] ^= 1;
            rev[x] = 0;
        }
    }

    void rotate(int x) {
        int y = f[x], z = f[y], k = ck(x), w = c[x][k^1];
        c[y][k] = w; f[w] = y;
        c[z][ck(y)] = x; f[x] = z;
        c[x][k^1] = y; f[y] = x;
        pushUp(y); pushUp(x);
    }

    void splay(int x, int goal = 0) {
        while(f[x] != goal) {
            int y = f[x], z = f[y];
            if(z != goal) {
                if(ck(x) == ck(y)) rotate(y);
                else rotate(x);
            }
            rotate(x);
        }
        if(!goal) root = x;
    }

    void insert(int x) {
        int cur = root, p = 0;
        while(cur && val[cur] != x) {

```

```

        p = cur;
        cur = c[cur][x > val[cur]];
    }
    if(cur) {
        cnt[cur]++;
    } else {
        cur = ++ncnt;
        if(p) c[p][x > val[p]] = cur;
        c[cur][0] = c[cur][1] = 0;
        f[cur] = p; val[cur] = x;
        cnt[cur] = sz[cur] = 1;
    }
    splay(cur);
}

void find(int x) {
    int cur = root;
    while(c[cur][x > val[cur]] && x != val[cur]) {
        cur = c[cur][x > val[cur]];
    }
    splay(cur);
}

int kth(int k) {
    int cur = root;
    while(1) {
        pushDown(cur);
        if(c[cur][0] && k <= sz[c[cur][0]]) {
            cur = c[cur][0];
        } else if(k > sz[c[cur][0]] + cnt[cur]) {
            k -= sz[c[cur][0]] + cnt[cur];
            cur = c[cur][1];
        } else {
            return cur;
        }
    }
}

void _reverse(int l, int r) {
    int x = kth(l), y = kth(r + 2);
    splay(x); splay(y, x);
    rev[c[y][0]] ^= 1;
}

```

```

int pre(int x) {
    find(x);
    if(val[root] < x) return root;
    int cur = c[root][0];
    while(c[cur][1]) cur = c[cur][1];
    return cur;
}

int succ(int x) {
    find(x);
    if(val[root] > x) return root;
    int cur = c[root][1];
    while(c[cur][0]) cur = c[cur][0];
    return cur;
}

void remove(int x) {
    int last = pre(x), next = succ(x);
    splay(last); splay(next, last);
    int del = c[next][0];
    if(cnt[del] > 1) {
        cnt[del]--;
        splay(del);
    }
    else c[next][0] = 0;
}

void output(int x) {
    pushDown(x);
    if(lc) output(lc);
    if(val[x] && val[x] <= n) printf("%d ", val[x]);
    if(rc) output(rc);
}

};
using namespace Splay_Tree;

int main()
{
    /*
    int n, op, x;
    scanf("%d", &n);
    insert(0x3f3f3f3f);
    insert(0xcfcfcfcf);
    while(n--) {

```



```

scanf("%d %d", &op, &x);
switch(op) {
    case 1: insert(x); break;
    case 2: remove(x); break;
    case 3: find(x); printf("%d\n", sz[c[root][0]]); break;
    case 4: printf("%d\n", val[kth(x+1)]); break;
    case 5: printf("%d\n", val[pre(x)]); break;
    case 6: printf("%d\n", val[succ(x)]); break;
}
}
*/
// 反转 //
scanf("%d %d", &n, &m);
for(int i = 0; i <= n + 1; i++) insert(i);
while(m--) {
    scanf("%d %d", &x, &y);
    _reverse(x, y);
}
output(root);
return 0;
}
/*
1.插入 xx 数
2.删除 xx 数(若有多个相同的数, 因只删除一个)
3.查询 xx 数的排名(排名定义为比当前数小的数的个数+1+1。若有多个相同的数, 因输出最小的排名)
4.查询排名为 xx 的数
5.求 xx 的前驱(前驱定义为小于 xx, 且最大的数)
6.求 xx 的后继(后继定义为大于 xx, 且最小的数)
*/

```

4. Treap

普通平衡树

```

const int N = 3e6 + 5;

mt19937 rnd(time(0));

struct Treap {
    struct node {
        int l, r, val, key, sz;
    }a[N];
}

```

```

int rt, tot;
int new_node(int val) {
    a[++tot] = {0, 0, val, rnd(), 1};
    return tot;
}
void init() { tot = rt = 0; }

void update(int now) {
    a[now].sz = a[a[now].l].sz + a[a[now].r].sz + 1;
}

void split(int now, int &x, int &y, int val) {
    if(!now) {
        x = y = 0; return ;
    }
    if(a[now].val <= val) {
        x = now;
        split(a[now].r, a[now].r, y, val);
    }
    else {
        y = now;
        split(a[now].l, x, a[now].l, val);
    }
    update(now);
}

void Merge(int &now, int x, int y) {
    if(!x || !y) {
        now = x + y; return ;
    }
    if(a[x].key >= a[y].key) {
        now = x;
        Merge(a[now].r, a[now].r, y);
    }
    else {
        now = y;
        Merge(a[now].l, x, a[now].l);
    }
    update(now);
}
int x, y, z;
void Insert(int val) {
    split(rt, x, y, val);
    Merge(x, x, new_node(val));
}

```

```

    Merge(rt, x, y);
}

void Delete(int val) {
    split(rt, x, y, val);
    split(x, x, z, val - 1);
    Merge(z, a[z].l, a[z].r);
    Merge(x, x, z);
    Merge(rt, x, y);
}

int Rank(int val) {
    split(rt, x, y, val - 1);
    int sz = a[x].sz + 1;
    Merge(rt, x, y);
    return sz;
}

int get_val(int rk) {
    int now = rt;
    while(1) {
        if(a[a[now].l].sz + 1 == rk) return a[now].val;
        else if(a[a[now].l].sz >= rk) now = a[a[now].l];
        else {
            rk -= a[a[now].l].sz + 1;
            now = a[a[now].r];
        }
    }
    return 0;
}

int pre(int val) {
    split(rt, x, y, val-1);
    int now = x;
    while(a[now].r)
        now = a[now].r;
    Merge(rt, x, y);
    return a[now].val;
}

int suf(int val) {
    split(rt, x, y, val);
    int now = y;
    while(a[now].l)

```

```

        now = a[now].l;
        Merge(rt, x, y);
        return a[now].val;
    }

}T;

void pr(int v) {
    printf("%d\n", v);
}

int main() {
    int n, op, v; T.init();
    scanf("%d", &n);
    while(n--) {
        scanf("%d %d", &op, &v);
        switch(op) {
            case 1 : T.Insert(v); break;
            case 2 : T.Delete(v); break;
            case 3 : pr(T.Rank(v)); break;
            case 4 : pr(T.get_val(v)); break;
            case 5 : pr(T.pre(v)); break;
            case 6 : pr(T.suf(v)); break;
        }
    }
    return 0;
}

```

文艺平衡树

您需要写一种数据结构（可参考题目标题），来维护一个有序数列，其中需要提供以下操作：翻转一个区间，例如原有序序列是5 4 3 2 1，翻转区间是[2,4]的话，结果是5 2 3 4 1

```

const int N = 3e6 + 5;

mt19937 rnd(time(0));

vector<int> q;
struct Treap {
    struct node {
        int l, r, val, key, sz, rev;
    }a[N];
    int rt, tot;
    int new_node(int val) {
        a[++tot] = {0, 0, val, rnd(), 1, 0};
    }
}

```

```

        return tot;
    }
    void init() { tot = rt = 0; }

    void up(int i) {
        swap(a[i].l, a[i].r);
        a[i].rev ^= 1;
    }

    void push_r(int i) {
        if(a[i].rev) {
            up(a[i].l);
            up(a[i].r);
            a[i].rev = 0;
        }
    }

    void push_up(int i) {
        a[i].sz = a[a[i].l].sz + a[a[i].r].sz + 1;
    }

    int x, y, z;
    void split(int now, int &x, int &y, int sz) {
        if(!now) {
            x = y = 0; return ;
        }
        push_r(now);
        if(a[a[now].l].sz < sz) {
            x = now;
            split(a[now].r, a[now].r, y, sz - a[a[now].l].sz - 1);
        }
        else {
            y = now;
            split(a[now].l, x, a[now].l, sz);
        }
        push_up(now);
    }

    void Merge(int &now, int x, int y) {
        if(!x || !y) { now = x + y; return ;}
        push_r(x); push_r(y);
        if(a[x].key > a[y].key) {
            now = x;
            Merge(a[now].r, a[now].r, y);
        }
        else {
            now = y;
            Merge(a[now].l, x, a[now].l);
        }
        push_up(now);
    }

```

```

    }
    else {
        now = y;
        Merge(a[now].l, x, a[now].l);
    }
    push_up(now);
}

void Insert(int pos, int v) {
    split(rt, x, y, pos - 1);
    Merge(x, x, new_node(v));
    Merge(rt, x, y);
}

void rev(int l, int r) {
    split(rt, x, y, l - 1);
    split(y, y, z, r - l + 1);
    up(y);
    Merge(y, y, z);
    Merge(rt, x, y);
}

void dfs(int now) {
    if(!now) return ;
    push_r(now);
    dfs(a[now].l);
    q.push_back(a[now].val);
    dfs(a[now].r);
}

}T;

//void pr(int v) {
//    printf("%d\n", v);
//}

int main() {
    T.init();
    int n, m, l, r;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++)
        T.Insert(i, i);
    while(m--) {
        scanf("%d %d", &l, &r);

```

```

        T.rev(l, r);
    }
    T.dfs(T.rt);
    for(int i = 0; i < n; i++)
        printf("%d%c", q[i], (i == n - 1 ? '\n' : ' '));

    return 0;
}

```

5. 动态树（LCT）

```

#define lc c[x][0]
#define rc c[x][1]
const int N = 3e5 + 5;

namespace Lct {
    int f[N], c[N][2], v[N], s[N], st[N];
    bool r[N];
    bool nroot(int x) { //判断节点是否为一个 Splay 的根（与普通 Splay 的区别 1）
        return c[f[x]][0] == x || c[f[x]][1] == x;
    }
    //原理很简单，如果连的是轻边，他的父亲的儿子没有它
    void pushUp(int x) { //上传信息
        s[x] = s[lc] ^ s[rc] ^ v[x];
    }
    void push_r(int x) { //翻转操作
        swap(lc, rc); r[x] ^= 1;
    }
    void pushDown(int x) { //判断并释放懒标记
        if(r[x]) {
            if(lc) push_r(lc);
            if(rc) push_r(rc);
            r[x] = 0;
        }
    }
    void rotate(int x) { //一次旋转
        int y=f[x],z=f[y],k=c[y][1]==x,w=c[x][!k];
        if(nroot(y))c[z][c[z][1]==y]=x;c[x][!k]=y;c[y][k]=w;
        if(w)f[w]=y;f[y]=x;f[x]=z;
        pushUp(y);
    }
}

```

```

}
void splay(int x) { //只传了一个参数，因为所有操作的目标都是该 Splay 的
根（与普通 Splay 的区别 3）
    int y = x, z = 0;
    st[++z] = y; //st 为栈，暂存当前点到根的整条路径，pushdown 时一定要
    从上往下放标记（与普通 Splay 的区别 4）
    while(nroot(y)) st[++z] = y = f[y];
    while(z) pushDown(st[z--]);
    while(nroot(x)) {
        y = f[x]; z = f[y];
        if(nroot(y))
            rotate((c[y][0]==x)^(c[z][0]==y)?x:y);
        rotate(x);
    }
    pushUp(x);
}
void access(int x) { //访问
    for(int y = 0; x; x = f[y=x])
        splay(x), c[x][1] = y, pushUp(x);
}
void makeRoot(int x) { //换根
    access(x); splay(x);
    push_r(x);
}
int findRoot(int x) { //找根（在真实的树中的）
    access(x); splay(x);
    while(lc) pushDown(x), x = lc;
    splay(x);
    return x;
}
void split(int x, int y) { //提取路径
    makeRoot(x);
    access(y); splay(y);
}
void link(int x, int y) { //连边
    makeRoot(x);
    if(findRoot(y) != x) f[x] = y;
}
void cut(int x, int y) { //断边
    makeRoot(x);
    if(findRoot(y) == x && f[y] == x && !c[y][0]) {
        f[y] = c[x][1] = 0;
        pushDown(x);
    }
}

```



```

    }
};
using namespace Lct;

int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++) scanf("%d", &v[i]);
    while(m--) {
        int op, x, y;
        scanf("%d %d %d", &op, &x, &y);
        switch(op) {
            case 0 : split(x, y); printf("%d\n", s[y]); break;
            case 1 : link(x, y); break;
            case 2 : cut(x, y); break;
            case 3 : splay(x); v[x] = y; //先把 x 转上去再改，不然会影响
Splay 信息的正确性
        }
    }
    return 0;
}

```

4.1.1 Lazy 标记

题目描述

[展开](#)

一棵 n 个点的树，每个点的初始权值为 1。对于这棵树有 q 个操作，每个操作为以下四种操作之一：

- `+ u v c` : 将 u 到 v 的路径上的点的权值都加上自然数 c ;
- `- u1 v1 u2 v2` : 将树中原有的边 $(u1, v1)$ 删除，加入一条新边 $(u2, v2)$ ，保证操作完之后仍然是一棵树;
- `* u v c` : 将 u 到 v 的路径上的点的权值都乘上自然数 c ;
- `/ u v` : 询问 u 到 v 的路径上的点的权值和，求出答案对于 51061 的余数。

```

#define lc c[x][0]
#define rc c[x][1]
#define ll long long
const int N = 3e5 + 5;
const int mod = 51061;
namespace Lct {
    ll f[N], c[N][2], v[N], s[N], st[N], sz[N], lm[N], la[N];

```

```

bool r[N];
bool nroot(int x) { //判断节点是否为一个 Splay 的根（与普通 Splay 的区别 1）
    return c[f[x]][0] == x || c[f[x]][1] == x;
} //原理很简单，如果连的是轻边，他的父亲的儿子没有它

void pushUp(int x) { //上传信息
    s[x] = (s[lc] + s[rc] + v[x]) % mod;
    sz[x] = sz[lc] + sz[rc] + 1;
}

void push_a(int x, int c) {
    s[x] = (s[x] + sz[x] * c % mod) % mod;
    v[x] = (v[x] + c) % mod;
    la[x] = (la[x] + c) % mod;
}

void push_m(int x, int c) {
    s[x] = s[x] * c % mod;
    v[x] = v[x] * c % mod;
    la[x] = la[x] * c % mod;
    lm[x] = lm[x] * c % mod;
}

void push_r(int x) { //翻转操作
    swap(lc, rc); r[x] ^= 1;
}

void pushDown(int x) { //判断并释放懒标记
    if(r[x]) {
        if(lc) push_r(lc);
        if(rc) push_r(rc);
        r[x] = 0;
    }
    if(lm[x] != 1) {
        if(lc) push_m(lc, lm[x]);
        if(rc) push_m(rc, lm[x]);
        lm[x] = 1;
    }
    if(la[x]) {
        if(lc) push_a(lc, la[x]);
        if(rc) push_a(rc, la[x]);
        la[x] = 0;
    }
}

void rotate(int x) { //一次旋转
    int y=f[x],z=f[y],k=c[y][1]==x,w=c[x][!k];
    if(nroot(y))c[z][c[z][1]==y]=x;c[x][!k]=y;c[y][k]=w;
}

```

```

        if(w)f[w]=y;f[y]=x;f[x]=z;
        pushUp(y);
    }
    void splay(int x) {//只传了一个参数，因为所有操作的目标都是该 Splay 的
    根（与普通 Splay 的区别 3）
        int y = x, z = 0;
        st[++z] = y;//st 为栈，暂存当前点到根的整条路径，pushdown 时一定要
        从上往下放标记（与普通 Splay 的区别 4）
        while(nroot(y)) st[++z] = y = f[y];
        while(z) pushDown(st[z--]);
        while(nroot(x)) {
            y = f[x]; z = f[y];
            if(nroot(y))
                rotate((c[y][0]==x)^(c[z][0]==y)?x:y);
            rotate(x);
        }
        pushUp(x);
    }
    void access(int x) {//访问
        for(int y = 0; x; x = f[y=x])
            splay(x), c[x][1] = y, pushUp(x);
    }
    void makeRoot(int x) {//换根
        access(x); splay(x);
        push_r(x);
    }
    int findRoot(int x) {//找根（在真实的树中的）
        access(x); splay(x);
        while(lc) pushDown(x), x = lc;
        splay(x);
        return x;
    }
    void split(int x, int y) {//提取路径
        makeRoot(x);
        access(y); splay(y);
    }
    void link(int x, int y) {//连边
        makeRoot(x);
        if(findRoot(y) != x) f[x] = y;
    }
    void cut(int x, int y) {//断边
        makeRoot(x);
        if(findRoot(y) == x && f[y] == x && !c[y][0]) {
            f[y] = c[x][1] = 0;
        }
    }

```

```

        pushDown(x);
    }
}
bool isConnect(int x, int y) {
    makeRoot(x);
    if(findRoot(y) == x) return true;
    return false;
}
};
using namespace Lct;

int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++) v[i] = lm[i] = sz[i] = 1;
    for(int i = 1; i < n; i++) {
        int x, y; scanf("%d %d", &x, &y);
        link(x, y);
    }
    // for(int i = 1; i <= n; i++) scanf("%d", &v[i]);
    while(m--) {
        int x, y, u, v;
        char op;
        scanf(" %c %d %d", &op, &x, &y);
        switch(op) {
            case '*' : scanf("%d", &u); split(x, y); push_m(y, u);
break;

            case '/' : split(x, y); printf("%lld\n", s[y]); break;
            case '+' : scanf("%d", &u); split(x, y); push_a(y, u);
break;

            case '-' : scanf("%d %d", &u, &v); cut(x, y); link(u, v);
break;

        }
    }
    return 0;
}

```

6. 可持续化字典树

```
const int N = 1e5 + 5;
int tot = 0, rt[N];
struct Trie{
    int a[N * 33][2], sum[N * 33];
    int update(int root, int v)
    {
        int now = ++tot;
        int tmp = now;
        sum[now] = sum[root] + 1;
        for(int i = 31; i >= 0; i--)
        {
            int t = (v >> i) & 1;
            a[now][t] = ++tot;
            a[now][t^1] = a[root][t^1];
            root = a[root][t];
            now = tot;
            sum[now] = sum[root] + 1;
        }
        return tmp;
    }
    int query(int lx, int rx, int val)
    {
        int tmp = 0;
        for(int i = 31; i >= 0; i--)
        {
            int t = (val >> i) & 1;
            if(sum[a[rx][t^1]] - sum[a[lx][t^1]])
            {
                tmp += (1 << i);
                lx = a[lx][t^1];
                rx = a[rx][t^1];
            }
            else
            {
                lx = a[lx][t];
                rx = a[rx][t];
            }
        }
        return tmp;
    }
}trie;
```

```

int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
    {
        int v;
        scanf("%d", &v);
        rt[i] = trie.update(rt[i - 1], v);
    }
    int q;
    scanf("%d", &q);
    while(q--)
    {
        int l, r, x;
        scanf("%d %d %d",&x ,&l, &r);
        printf("%d\n",trie.query(rt[l - 1], rt[r], x));
    }
    return 0;
}

```

7. CDQ 分治

```

/*
P1393 动态逆序对
对于给定的一段正整数序列，我们定义它的逆序对的个数为序列中  $a_i > a_j$  且  $i < j$  的有序对  $(i, j)$  的个数。你需要计算出一个序列的逆序对组数及其删去其中的某个数的逆序对组数。
*/
const int N = 1e5 + 5;
struct node {
    int t, x, y;
}q1[N], q2[N], tmp[N];

bool cmp1 (node a, node b) {
    if(a.t == b.t) return a.x < b.x;
    return a.t < b.t;
}
bool cmp2 (node a, node b) {

```

```

    if(a.t == b.t) return a.x > b.x;
    return a.t < b.t;
}

int b[N], id[N], c[N], f[N], n, m;

struct BIT {
    int c[N];
    void init() { memset(c, 0, sizeof(c));}
    int lowbits(int x) { return x & -x;}
    void add(int i, int v) {
        for(; i <= n; i += lowbits(i)) c[i] += v;
    }
    int sum(int i) {
        int ans = 0;
        for(; i > 0; i -= lowbits(i)) ans += c[i];
        return ans;
    }
}B;

long long ans[N];

void cdq(int l, int r) {
    if(l >= r) return ;
    int mid = (l + r) >> 1;
    cdq(l, mid); cdq(mid + 1, r);
    int i = l, j = mid + 1, o = l;
    while(i <= mid || j <= r) {
        if(j > r || (i <= mid && q1[i].x < q1[j].x)) B.add(q1[i].y, 1),
tmp[o++] = q1[i++];
        else ans[q1[j].t] += B.sum(n) - B.sum(q1[j].y), tmp[o++] =
q1[j++];
    }
    for(int i = l; i <= mid; i++) B.add(q1[i].y, -1);
    for(int i = l; i <= r; i++) q1[i] = tmp[i];

    i = l, j = mid + 1, o = l;
    while(i <= mid || j <= r) {
        if(j > r || (i <= mid && q2[i].x > q2[j].x)) B.add(q2[i].y, 1),
tmp[o++] = q2[i++];
        else ans[q2[j].t] += B.sum(q2[j].y), tmp[o++] = q2[j++];
    }
    for(int i = l; i <= mid; i++) B.add(q2[i].y, -1);
    for(int i = l; i <= r; i++) q2[i] = tmp[i];
}

```

```

}

int x[N];

int main()
{
    B.init();
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &b[i]);
        x[i] = b[i];
    }
    sort(x + 1, x + n + 1);
    for(int i = 1; i <= n; i++) {
        b[i] = lower_bound(x + 1, x + n + 1, b[i]) - x;
        id[b[i]] = i;
    }
    int tim = 1, cur = 0;
    for(int i = 1; i <= m; i++) {
        scanf("%d", &c[i]);
        c[i] = b[c[i]];
    }
    for(int i = m; i >= 1; i--) {
        q1[++cur] = {++tim, id[c[i]], c[i]};
        q2[cur] = q1[cur];
        f[c[i]] = 1;
    }
    for(int i = 1; i <= n; i++) {
        if(!f[i]) {
            q1[++cur] = {1, id[i], i};
            q2[cur] = q1[cur];
        }
    }
    // 每个点算前后一次
    sort(q1 + 1, q1 + cur + 1, cmp1);
    sort(q2 + 1, q2 + cur + 1, cmp2);
    cdq(1, n);
    //没被删除的点多算了一次
    ans[1] /= 2;
    for(int i = 1; i <= tim; i++) {
        ans[i] += ans[i - 1];
    }
    for(int i = tim; i >= 1; i--) {
        printf("%lld%c", ans[i], i == 1 ? '\n' : ' ');
    }
}

```



```

    }
    return 0;
}
/*

6 3
5 4 2 6 3 1
4 5 6

3 0
2 3 1

*/

```

8. 点分治

```

const int INF = 0x3f3f3f3f;
const int N = 1e5 + 5;
struct node {
    int v, c;
};

vector<node> q[N];

int sum, rt, sz[N], mx[N], pm[N * 100], qu[N], ans[N];
int vis[N], rem[N], dis[N], cl[N];

void getrt(int u, int fa) {
    sz[u] = 1;
    mx[u] = 0;
    for(auto te : q[u]) {
        int v = te.v;
        if(vis[v] || v == fa) continue;
        getrt(v, u);
        sz[u] += sz[v];
        mx[u] = max(mx[u], sz[v]);
    }
    mx[u] = max(mx[u], sum - sz[u]);
    if(mx[u] < mx[rt]) rt = u;
}

void getdis(int u, int dep, int fa) {

```

```

//    dis[u] = dep;
    rem[++rem[0]] = dep;
    for(auto te : q[u]) {
        int v = te.v;
        if(v == fa || vis[v]) continue;
        getdis(v, dep + te.c, u);
    }

}

int n, m;
void calc(int u) {
    int num = 0;
    pm[0] = 1;
    for(auto te : q[u]) {
        int v = te.v;
        if(vis[v]) continue;
        rem[0] = 1;
        rem[1] = 0;
        getdis(v, te.c, u);
        for(int i = 1; i <= rem[0]; i++)
            for(int j = 1; j <= m; j++) {
                if(qu[j] >= rem[i])
                    ans[j] |= pm[qu[j] - rem[i]];
            }
        for(int i = 1; i <= rem[0]; i++) {
            if(rem[i] <= 10000000)
                pm[rem[i]] = 1, cl[++num] = rem[i];
        }
        //    cout << rem[i] << endl;
    }
}

for(int i = 1; i <= num; i++) pm[cl[i]] = 0;
}

void dfs(int u) {
    mx[rt = 0] = INF;
    getrt(u, -1); u = rt;
    vis[u] = 1;
    calc(u);
    //    cout << rt << endl;
    for(auto te : q[u]) {
        int v = te.v;
        if(vis[v]) continue;
        sum = sz[v];
        dfs(v);
    }
}

```

```

    }
}

int main()
{
    scanf("%d %d", &n, &m);
    for(int i = 1; i < n; i++) {
        int u, v, c;
        scanf("%d %d %d", &u, &v, &c);
        q[u].push_back({v, c});
        q[v].push_back({u, c});
    }
    for(int i = 1; i <= m; i++) {
        scanf("%d", &qu[i]);
    }
    sum = n;
    dfs(1);
    for(int i = 1; i <= m; i++) {
        if(ans[i]) puts("AYE");
        else puts("NAY");
    }
    return 0;
}

```

字符串处理

1. Hash

```

void Hash(char *s) {
    p[0] = 1; h[0] = 0;
    int len = strlen(s + 1);
    for(int i = 1; i <= len; i++) {
        p[i] = p[i - 1] * seed;
        h[i] = h[i - 1] * seed + s[i];
    }
}

ull get_hash(int l, int r) {
    return h[r] - h[l - 1] * p[r - l + 1];
}

```

```

// 判断回文串的话，在 hash 一个从后往前的，比较两个 hash 一样，
// unordered_map 的复杂度可能也很高

struct Hash {
    static const int MOD = 1999997; // 2908361
    static const int N = 1e6;
    int head[MOD + 10], nx[N], top;
    int hs[N], id[N];
    void init() {
        memset(head, -1, sizeof head);
        top = 0;
    }
    void insert(int x, int y) {
        int k = x % MOD;
        hs[top] = x; id[top] = y; nx[top] = head[k]; head[k] = top++;
    }
    int find(int x) {
        int k = x % MOD;
        for (int i = head[k]; i != -1; i = nx[i]) {
            if (hs[i] == x) {
                return id[i];
            }
        }
        return -1;
    }
}hs;

struct Hash {
    ll base[3] = {43, 47, 41}, mod[3] = {1000000007, 1000000009,
998244353};
    ll h[3][N], p[3][N], tmp[3];
    char str[N];
    int n;
    void init() {
        n = 0;
        for(int i = 0; i < 3; i++) {
            p[i][0] = 1; h[i][0] = 0;
            for(int j = 1; j <= N; j++) p[i][j] = p[i][j-1] *
base[i] % mod[i];
        }
    }

    ll get_hash(int i, int l, int r) {

```

```

        return (h[i][r] - h[i][l-1] * p[i][r-l+1] % mod[i] + mod[i]) %
mod[i];
    }

    void hash_(char *s) {
        int len = strlen(s);
        int mi = min(len, n), idx = 0;
        memset(tmp, 0, sizeof(tmp));

        for(int j = 1; j <= mi; j++) {
            int flag = 0;
            for(int i = 0; i < 3; i++) {
                tmp[i] = (tmp[i] * base[i] + s[j-1]) % mod[i];
                if(tmp[i] != get_hash(i, n-j+1, n)) flag = 1;
            }
            if(!flag) idx = j;
        }

        for(int j = idx; j < len; j++) {
            for(int i = 0; i < 3; i++) {
                h[i][++n] = h[i][n-1] * base[i] + s[j];
                h[i][n] %= mod[i];
            }
            str[n] = s[j];
        }
    }

    void pr() {
        for(int i = 1; i <= n; i++) printf("%c", str[i]); puts("");
    }

}hs;

```

2. 二维 Hash

```

#define maxn 1001
const int base1=13;
const int base2=131;
inline int read(){
    int x=0,f=0;char ch=getchar();
    while(ch>'9' || ch<'0')f|=ch=='-',ch=getchar();
    while(ch>='0' && ch<='9')x=(x<<3)+(x<<1)+(ch^48),ch=getchar();
    return f?-x:x;
}

```

```

}
int n,m,a,b;
unordered_map<ull,bool>hs;
ull mp[maxn][maxn],b1[maxn],b2[maxn];
int main(){
    n=read(),m=read(),a=read(),b=read(),b1[0]=b2[0]=1;
    for(int i=1;i<=maxn;++i)b1[i]=b1[i-1]*base1,b2[i]=b2[i-1]*base2;
    for(int i=1;i<=n;++i)for(int
j=1;j<=m;++j)scanf("%1d",&mp[i][j]);
    for(int i=1;i<=n;++i)for(int j=1;j<=m;++j)mp[i][j]+=mp[i][j-
1]*base1;
    for(int i=1;i<=n;++i)for(int j=1;j<=m;++j)mp[i][j]+=mp[i-
1][j]*base2;
    for(int i=a;i<=n;++i)
        for(int j=b;j<=m;++j){
            ull hss=mp[i][j]-mp[i][j-b]*b1[b]-mp[i-
a][j]*b2[a]
            +mp[i-a][j-b]*b1[b]*b2[a];
            hs[hss]=1;
        }
    int q=read();
    while(q--){
        for(int i=1;i<=a;++i)for(int
j=1;j<=b;++j)scanf("%1d",&mp[i][j]);
        for(int i=1;i<=a;++i)for(int
j=1;j<=b;++j)mp[i][j]+=mp[i][j-1]*base1;
        for(int i=1;i<=a;++i)for(int
j=1;j<=b;++j)mp[i][j]+=mp[i-1][j]*base2;
        if(hs.count(mp[a][b]))puts("YES");
        else puts("NO");
    }
    return 0;
}

```

3. unordered_map

```

#define eps 1e-10
struct Point{
    double x, y;
    bool friend operator == (Point a, Point b){
        return fabs(a.x - b.x) < eps && fabs(a.y - b.y) < eps;
    }
};

```

```

    }
};
struct hash_cmp{
    ll operator()(const Point &p1) const{
        return (ll)(p1.x * p1.y);
    }
};
unordered_set<Point, hash_cmp> mp;

```

4. 回文树

```

struct Palindromic_Tree {
    int next[N][27] ;//next 指针，next 指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[N] ;//fail 指针，失配后跳转到 fail 指针指向的节点
    int cnt[N] ; //表示节点 i 表示的本质不同的串的个数（建树时求出的不是完全的，最后 count()函数跑一遍以后才是正确的）
    int num[N] ; //表示以节点 i 表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[N] ;//len[i]表示节点 i 表示的回文串的长度
    int S[N] ;//存放添加的字符
    int last ;//指向上一个字符所在的节点，方便下一次 add
    int n ;//字符数组指针
    int p ;//节点指针

    int newnode ( int l ) { //新建节点
        for ( int i = 0 ; i < 27 ; ++ i ) next[p][i] = 0 ;
        cnt[p] = 0 ;
        num[p] = 0 ;
        len[p] = l ;
        return p ++ ;
    }

    void init () { //初始化
        p = 0 ;
        newnode ( 0 ) ;
        newnode ( -1 ) ;
        last = 0 ;
        n = 0 ;
        S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1 ;
    }
}

```

```

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

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

void count () {
    for ( int i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;
    //父亲累加儿子的 cnt，因为如果 fail[v]=u，则 u 一定是 v 的子回文串！
}
}pam;

```

5. AC 自动机

```

const int N = 500005;
struct Trie {
    int next[N][27], fail[N], end[N];
    int root;
    int tot;
    int newnode() {
        for (int i = 0; i < 27; i++) next[tot][i] = -1;
        end[tot++] = 0;
        return tot - 1;
    }
    void init() {
        tot = 0;
        root = newnode();
    }
};

```



```

}

void insert(char buf[]) {
    int len = strlen(buf);
    int now = root;
    for (int i = 0; i < len; i++) {
        int k = buf[i] - 'a';
        if (next[now][k] == -1)
            next[now][k] = newnode();
        now = next[now][k];
    }
    end[now]++;
}

void build() {
    queue<int> que;
    fail[root] = root;
    for (int i = 0; i < 27; i++)
        if (next[root][i] == -1)
            next[root][i] = root;
        else {
            fail[next[root][i]] = root;
            que.push(next[root][i]);
        }

    while (!que.empty()) {
        int now = que.front();
        que.pop();
        for (int i = 0; i < 27; i++)
            if (next[now][i] == -1)
                next[now][i] = next[fail[now]][i];
            else {
                fail[next[now][i]] = next[fail[now]][i];
                que.push(next[now][i]);
            }
    }
}

int query(char buf[]) {
    int len = strlen(buf);
    int now = root;
    int res = 0;
    for (int i = 0; i < len; i++) {
        now = next[now][buf[i] - 'a'];
    }
}

```

```

        int temp = now;
        while (temp != root) {
            res += end[temp];
            end[temp] = 0; //模式串只在主串中匹配一次就可以了
            temp = fail[temp];
        } // 这边可能超时用记忆化记录下
    }
    return res;
}

/** 记忆化
int num[N], dp[N];
int dfs(int cur) {
    if(cur == root) return 0;
    if(dp[cur] != -1) return dp[cur];
    return dp[cur] = end[cur] + dfs(fail[cur]);
}

void query(char buf[]) {
    int len = strlen(buf);
    int now = root;
    memset(dp, -1, sizeof(dp));
    for (int i = 0; i < len; i++) {
        now = next[now][buf[i] - 'a'];
        num[i] = dfs(now);
    }
}

**/
};

Trie ac;
char buf[N + N];
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n;
        scanf("%d", &n);
        ac.init();
        for (int i = 0; i < n; i++) {
            scanf("%s", buf);
            ac.insert(buf);
        }
        ac.build();
        scanf("%s", buf);
        printf("%d\n", ac.query(buf));
    }
}

```

```
    return 0;
}
```

6. 后缀自动机

```
const int N = 4e5 + 5;
struct SAM {
    int nex[N][28], fail[N], len[N];
    int sz, last, res;
    int newnode() {
        sz++;
        fail[sz] = 0, len[sz] = 0;
        memset(nex[sz], 0, sizeof(nex[sz]));
        return sz;
    }

    void init() {
        res = 0, sz = -1, last = 0;
        newnode();
        fail[0] = -1;
    }

    void add(int c) {
        int cur = newnode();
        len[cur] = len[last] + 1;
        int p;
        for(p = last; p != -1 && !nex[p][c]; p = fail[p]) nex[p][c] =
cur;

        if(p == -1) fail[cur] = 0;
        else {
            int q = nex[p][c];
            if(len[p] + 1 == len[q]) fail[cur] = q;
            else {
                int nq = newnode(); //新节点 nq
                fail[nq] = fail[q]; //复制 q 节点
                len[nq] = len[p] + 1; //当前节点最长后缀长度
                memcpy(nex[nq], nex[q], sizeof(nex[q]));
            }
        }
    }
};
```

```

        for(; p != -1 && nex[p][c] == q; p = fail[p]) nex[p][c]
= nq;
        fail[q] = fail[cur] = nq;
    }
}
last = cur;
res += len[cur] - len[fail[cur]];
}
}sam;

```

// 计算本质不同的子串个数

```

for(int i = 1; i <= sam.sz; i++)
    ans += sam.len[i] - sam.len[sam.fail[i]];

```

// 计算长度为 k 的子串出现次数最多, cnt[k]

```

for(int i = len-1; i >= 1; i--)
    sam.cnt[i] = max(sam.cnt[i], sam.cnt[i+1]);

```

//字典序第 k 小子串--这一问题的基础思路 and 上两题类似。字典序第 k 小子串——自动机中字典序第 k 小的路径。因此，考虑从每个状态出

发的不同路径数，我们将得以轻松地确定第 k 小路径，从初始状态开始逐位确定答案。

//最小循环移位 --我们将字符串 S+S 建立后缀自动机。该自动机将包含和 S 循环同构的所有字符串。从而，问题就简化成了在自动机中找出字典序最小的，长度为 length(S) 的路径，这很简单：从初始状态开始，每一步都贪心地走，经过最小的转移。

// 10 个 串的最长公共子串 ->hzwer

```

#define N 200005

```

```

#define inf 1000000000

```

```

int tot,cnt,n;

```

```

int s[N],l[N],cl[N],mn[N],fa[N],to[N][26];

```

```

int v[100005];

```

```

char ch[100005];

```

```

struct sam

```

```

{

```

```

    int p,q,np,nq;

```

```

    int last;

```

```

    sam(){

```

```

        cnt=++last;

```

```

    }

```

```

    void extend(int c){

```

```

        p=last;last=np=++cnt;l[np]=l[p]+1;

```

```

        for(;p&&!to[p][c];p=fa[p])to[p][c]=np;
        if(!p)fa[np]=1;
        else
        {
            q=to[p][c];
            if(l[p]+1==l[q])fa[np]=q;
            else
            {
                nq=++cnt;l[nq]=l[p]+1;
                memcpy(to[nq],to[q],sizeof(to[q]));
                fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                for(;to[p][c]==q;p=fa[p])to[p][c]=nq;
            }
        }
    }
}

void build(){
    scanf("%s",ch);
    n=strlen(ch);
    for(int i=0;i<n;i++)
        extend(ch[i]-'a');
}

bool pre()
{
    if(scanf("%s",ch)==EOF)return 0;
    memset(cl,0,sizeof(cl));
    n=strlen(ch);int tmp=0;
    for(int p=1,i=0;i<n;i++)
    {
        int c=ch[i]-'a';
        if(to[p][c])p=to[p][c],tmp++;
        else
        {
            while(p&&!to[p][c])p=fa[p];
            if(!p)p=1,tmp=0;
            else tmp=l[p]+1,p=to[p][c];
        }
        cl[p]=max(cl[p],tmp);
    }
    for(int i=cnt;i-->0)
    {
        int t=s[i];
        mn[t]=min(mn[t],cl[t]);
        if(cl[t]&&fa[t])cl[fa[t]]=l[fa[t]];
    }
}

```

```

        }
        return 1;
    }
}sam;
int main()
{
    sam.build();
    for(int i=1;i<=cnt;i++)
        mn[i]=l[i];
    for(int i=1;i<=cnt;i++)
        v[l[i]]++;
    for(int i=1;i<=n;i++)v[i]+=v[i-1];
    for(int i=1;i<=cnt;i++)
        s[v[l[i]]--]=i;
    while(sam.pre());
    int ans=0;
    for(int i=1;i<=cnt;i++)
        ans=max(ans,mn[i]);
    printf("%d\n",ans);
    return 0;
}

```

7. 后缀数组

```

const int N = 1e5 + 5; // 开二倍吧
int sa[N], rk[N], Height[N], tax[N], tp[N], a[N], n, m;
//rk[i] 第 i 个后缀的排名; SA[i] 排名为 i 的后缀位置; Height[i] 排名为 i 的后缀与排名为(i-1)的后缀的 LCP
//tax[i] 计数排序辅助数组; tp[i] rk 的辅助数组(计数排序中的第二关键字),与 SA 意义一样。
//a 为原串
void Rsort() { //基数排序 要弄懂
    //rk 第一关键字,tp 第二关键字。
    for(int i = 0; i <= m; i++) tax[i] = 0;
    for(int i = 1; i <= n; i++) tax[rk[tp[i]]]++;
    for(int i = 1; i <= m; i++) tax[i] += tax[i - 1];
    for(int i = n; i >= 1; i--) sa[tax[rk[tp[i]]]--] = tp[i]; //确保满足第一关键字的同时,再满足第二关键字的要求
}
int cmp(int *f, int x, int y, int w) { return f[x] == f[y] && f[x + w] == f[y + w]; }

```

```

//通过二元组两个下标的比较，确定两个子串是否相同
void init() {
    memset(tp, 0, sizeof(tp));
    memset(rk, 0, sizeof(rk));
    memset(a, 0, sizeof(a));
}
void Suffix() {
    for(int i = 1; i <= n; i++) rk[i] = a[i], tp[i] = i;
    m = 128; Rsort();
    for(int w = 1, p = 1, i; p < n; w += w, m = p) {
        //w 当前一个子串的长度; m 当前离散后的排名种类数
        //当前的 tp(第二关键字)可直接由上一次的 SA 的得到
        for(p = 0, i = n - w + 1; i <= n; i++) tp[++p] = i;
        for(int i = 1; i <= n; i++) if(sa[i] > w) tp[++p] = sa[i] - w;
        //更新 SA 值,并用 tp 暂时存下上一轮的 rank(用于 cmp 比较)
        Rsort(); swap(rk, tp), rk[sa[1]] = p = 1;
        //用已经完成的 SA 来更新与它互逆的 rank,并离散 rank
        for(int i = 2; i <= n; i++) rk[sa[i]] = cmp(tp, sa[i], sa[i -
1], w) ? p : ++p;
    }
}
void cal_height() { //这个知道原理后就比较好理解程序
    int j, k = 0;
    for(int i = 1; i <= n; Height[rk[i++]] = k)
        for(k = k ? k - 1 : k, j = sa[rk[i] - 1]; a[i + k] == a[j + k];
++k);
}
int dp[N][20];
void ST() {
    for(int i = 1; i <= n; i++) dp[i][0] = Height[i];
    for(int j = 1; j <= 20; j++)
        for(int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
int RMQ(int l, int r) {
    int k = log2(r - l + 1);
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}
}

```

经典应用

两个后缀的最大公共前缀

$lcp(x, y) = \min(height[x - y])$, 用rmq维护, $O(1)$ 查询

可重叠最长重复子串

Height数组里的最大值

不可重叠最长重复子串 POJ1743

首先二分答案 x , 对height数组进行分组, 保证每一组的 $minheight$ 都 $\geq x$

依次枚举每一组, 记录下最大和最小长度, 多 $sa[mx] - sa[mi] \geq x$ 那么可以更新答案

本质不同的子串的数量

枚举每一个后缀, 第 i 个后缀对答案的贡献为 $len - sa[i] + 1 - height[i]$

[回到顶部](#)

8. 后缀数组 DC3

```
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
const int MAXN = 2e6 + 5; // n*10
int sa[MAXN*3];
int Rank[MAXN];
int Height[MAXN];
int n;
char s[MAXN*3];
int r[MAXN*3];
int wa[MAXN*3], wb[MAXN*3], wv[MAXN*3];
int wws[MAXN*3];

void sort(int *r, int *a, int *b, int n, int m)
{
    int i;
    for(i=0; i<n; i++) wv[i]=r[a[i]];
    for(i=0; i<m; i++) wws[i]=0;
    for(i=0; i<n; i++) wws[wv[i]]++;
    for(i=1; i<m; i++) wws[i]+=wws[i-1];
    for(i=n-1; i>=0; i--) b[--wws[wv[i]]]=a[i];
}
```



```

        return;
    }

    int c0(int *r,int a,int b)
    {return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}
    int c12(int k,int *r,int a,int b)
    {if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}

    void dc3(int *r,int *sa,int n,int m)
    {
        int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
        r[n]=r[n+1]=0;
        for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
        sort(r+2,wa,wb,tbc,m);
        sort(r+1,wb,wa,tbc,m);
        sort(r,wa,wb,tbc,m);
        for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
            rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
        if(p<tbc) dc3(rn,san,tbc,p);
        else for(i=0;i<tbc;i++) san[rn[i]]=i;
        for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
        if(n%3==1) wb[ta++]=n-1;
        sort(r,wb,wa,ta,m);
        for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
        for(i=0,j=0,p=0;i<ta && j<tbc;p++)
            sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
        for(;i<ta;p++) sa[p]=wa[i++];
        for(;j<tbc;p++) sa[p]=wb[j++];
        return;
    }

    int K;
    void calHeight(int *r, int *sa, int n)
    {
        int i, j, k = 0;
        for (i = 1; i <= n; ++i) Rank[sa[i]] = i;
        for (i = 0; i < n; Height[Rank[i++]] = k)
            for (k ? k-- : 0, j = sa[Rank[i] - 1]; r[i + k] == r[j + k];
++k);
        return;
    }

    int main()
    {

```

```

scanf("%s",s);
int Max=-1;
n=strlen(s);
K = n;
for(int i=0;i<n;i++){
    r[i]=s[i];
    if(r[i]>Max)Max=r[i];
}
r[n]=0;
dc3(r,sa,n+1,Max+1);
calHeight(r,sa,n);
return 0;
}

```

9. 扩展 KMP

```

void EKMP(char s[],char t[],int next[],int extend[])//s 为主串，t 为模板串
{
    int i,j,p,L;
    int lens=strlen(s);
    int lent=strlen(t);
    next[0]=lent;
    j=0;
    while(j+1<lent && t[j]==t[j+1])j++;
    next[1]=j;

    int a=1;
    for(i=2;i<lent;i++)
    {
        p=next[a]+a-1;
        L=next[i-a];
        if(i+L<p+1)next[i]=L;
        else
        {
            j=max(0,p-i+1);
            while(i+j<lent && t[i+j]==t[j])j++;
            next[i]=j;
            a=i;
        }
    }
}

j=0;

```

```

while(j<lens && j<lent && s[j]==t[j])j++;
extend[0]=j;
a=0;
for(i=1;i<lens;i++)
{
    p=extend[a]+a-1;
    L=next[i-a];
    if(L+i<p+1)extend[i]=L;
    else
    {
        j=max(0,p-i+1);
        while(i+j<lens && j<lent && s[i+j]==t[j])j++;
        extend[i]=j;
        a=i;
    }
}
}

```

10. 字符串最小表示法

字符串最小表示法：以某个小标开始字典序最小的串。

假设有两个下标 i, j ，表示如果从 i 和从 j 出发的字符串，有一个 k 表示字符串的长度，如果长度达到 len ，就表示找到最小的串。

$s[i+k] == s[j+k]$: $k++$

$s[i+k] > s[j+k]$: $i=i+k+1$ 表示以 i ，到 $i+k$ 为起点的字符串，都不是最小字符串的前缀。

$s[i+k] < s[j+k]$: $j=j+k+1$ 同理

注意：1. i 和 j 一定是不同的。

2.每次不等时，需要设置 k 为0。

```

int get_min(char x[], int n) {
    int i = 0, j = 1;
    while(i < n && j < n) {
        int k = 0;
        while(k < n && x[i + k] == x[j + k]) k++;
        if(k == n) break;
        if(x[i + k] > x[j + k])
            i = max(i + k + 1, j + 1);
    }
}

```

```

        else j = max(j + k + 1, i + 1);
    }
    return min(i, j);
}

int get_max(char x[], int n) {
    int i = 0, j = 1;
    while(i < n && j < n) {
        int k = 0;
        while(x[i + k] == x[j + k] && k < n) k++;
        if(k == n) break;
        if(x[i + k] < x[j + k])
            i = max(i + k + 1, j + 1);
        else j = max(j + k + 1, i + 1);
    }
    return min(i, j);
}

```

数学

1. 逆元

线性递推

```

inv[1]=inv[0]=1;
for(int i=2;i<maxn;++i) inv[i]=(mod-mod/i)*inv[mod%i]%mod;

```

扩展欧几里得

```

typedef long long ll;
void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
    if(!b){ d=a; x=1; y=0;}
    else{ extgcd(b,a%b,d,y,x); y-=x*(a/b); }
}
ll inverse(ll a,ll n){
    ll d,x,y;
    extgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}

ax + by = 1
ax + by + kab - kab = 1
a(x + kb) + b(y - ka) = 1 (这里需要 ab / gcd(a,b) )

```

令 $\text{gcd} = \text{gcd}(a, b)$

特解: $x = x_0 + kb/\text{gcd}$, $y = y_0 - ka/\text{gcd}$

至于 $pa+qb=c$ 的整数解, 只需将 $p * a + q * b = \text{Gcd}(p, q)$ 的每个解乘上 $c/\text{Gcd}(p, q)$ 即可

最小解: $b = b/\text{gcd}$, $x = (x_0 \% b + b) \% b$

上面已经列出找一个整数解的方法, 在找到 $p * a + q * b = \text{Gcd}(p, q)$ 的一组解 p_0, q_0

后, $p * a + q * b = \text{Gcd}(p, q)$ 的其他整数解满足:

$$p = p_0 + b/\text{Gcd}(p, q) * t$$

$$q = q_0 - a/\text{Gcd}(p, q) * t \text{ (其中 } t \text{ 为任意整数)}$$

2. 阶乘的逆元

```
const ll Mod = 998244353;
const int MAX = 1e6+10;
ll fac_inv[MAX], fac[MAX];
ll Quick_pow(ll x, ll N){
    ll res = 1;
    while(N > 0){
        if(N&1) res = res*x%Mod;
        x = x%Mod*x%Mod;
        N >>= 1;
    }
    return res;
}
void Init_inv(){
    int N = 1e6+3;
    fac[0] = fac[1] = 1;
    for(int i=2; i<=N; ++i) fac[i] = fac[i-1]*i%Mod;
    fac_inv[N] = Quick_pow(fac[N], Mod-2);
    for(int i=N-1; i>=0; --i) fac_inv[i] = fac_inv[i+1]*(i+1)%Mod;
}
```

3. 组合数的一些性质

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

$$mC_n^m = nC_{n-1}^{m-1}$$

$$C_n^0 + C_n^1 + C_n^2 + \dots + C_n^n = 2^n$$

$$1C_n^1 + 2C_n^2 + 3C_n^3 + \dots + nC_n^n = n2^{n-1}$$

$$1^2C_n^1 + 2^2C_n^2 + 3^2C_n^3 + \dots + n^2C_n^n = n(n+1)2^{n-2}$$

$$\frac{C_n^1}{1} - \frac{C_n^2}{2} + \frac{C_n^3}{3} + \dots + (-1)^{n-1} \frac{C_n^n}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

$$(C_n^0)^2 + (C_n^1)^2 + (C_n^2)^2 + \dots + (C_n^n)^2 = C_{2n}^n$$

定义 $S(n, m) = \sum_{i=0}^m \binom{n}{i}$, 不难发现

$S(n, m) = S(n, m-1) + \binom{n}{m}$, $S(n, m) = 2S(n-1, m) - \binom{n-1}{m}$ 。也就是说, 如果我们知道 $S(n, m)$, 就能以 $O(1)$ 的代价计算出

$S(n-1, m), S(n, m-1), S(n+1, m), S(n, m+1)$, 可以采用莫队算法。

求 $C(n, m)$, $(n, m < 1e5)$ 查询次数 $1e5$ 。

```
ll fac[maxn+50], inv[maxn+50];
```

```
int block;
```

```
const ll mod=1e9+7;
```

```
struct nod
```

```
{
```

```
    ll l, r, pos;
```

```
}query[maxn+50];
```

```
ll pow(ll a, ll b)
```

```
{
```

```
    ll ans=1;
```

```
    while(b)
```

```
    {
```

```
        if(b&1) ans=ans*a%mod;
```

```

        a=a*a%mod;
        b>>=1;
    }
    return ans;
}

void ini()
{
    fac[0]=1;
    for(int i=1;i<=maxn;i++)
        fac[i]=fac[i-1]*i%mod;
    inv[maxn-1]=pow(fac[maxn-1],mod-2);
    for(int i=maxn-2;i>=0;i--)
    {
        inv[i]=inv[i+1]*(i+1)%mod; //乘起来等于 fac[maxn]^(mod-1),模 mod
也是 1
    }
}

bool cmp(nod a,nod b)
{
    if((a.l/block)==(b.l/block))
        return a.r<b.r;
    else
        return ((a.l/block)<(b.l/block));
}

ll C(ll n,ll m)
{
    if(n==m||m==0)
        return 1;
    if(m>n) return 0;
    return ((long long)fac[n]*inv[m]%mod)*inv[n-m]%mod;
}

int main()
{
    ll ans[maxn+50]={0};
    int T,x,y,maxnn=0;
    ini();
    //cout<<C(6,4)<<endl;
    scanf("%I64d",&T);
    for(int i=1;i<=T;i++)
    {

```

```

        scanf("%I64d%I64d",&x,&y);
        query[i].l=x;
        query[i].r=y;
        query[i].pos=i;
        maxnn=max(maxnn,x);

    }
    block=sqrt(maxnn);
    sort(query+1,query+T+1,cmp);
    int le=1,ri=1;
    ll now=2;
    for(int i=1;i<=T;i++)
    {
        while(le<query[i].l)
        {
            now=(2*now-C(le,ri)+mod)%mod;
            //printf("%d\n",C(le,ri));
            le++;
        }
        while(le>query[i].l)
        {
            le--;
            now=(now+C(le,ri))%mod*inv[2]%mod;
        }
        while(ri<query[i].r)
        {
            now=(now+C(le,ri+1))%mod;
            ri++;
        }
        while(ri>query[i].r)
        {
            now=(now-C(le,ri)+5*mod)%mod;
            ri--;
        }
        ans[query[i].pos]=now;
    }
    for(int i=1;i<=T;i++)
    {
        printf("%I64d\n",ans[i]);
    }
    return 0;
}

```


4. 扩展 Lucas 定理

```
//模数可能不是质数的情况下
LL n,m,MOD,ans;
LL fast_pow(LL a,LL p,LL Mod)
{
    LL ans=1LL;
    for (; p>=1,a=a%Mod)
        if (p&1)
            ans=ans*a%Mod;
    return ans;
}
void exgcd(LL a,LL b,LL &x,LL &y)
{
    if (!b) x=1LL,y=0LL;
    else exgcd(b,a%b,y,x),y-=a/b*x;
}
LL inv(LL A,LL Mod)
{
    if (!A) return 0LL;
    LL a=A,b=Mod,x=0LL,y=0LL;
    exgcd(a,b,x,y);
    x=((x%b)+b)%b;
    if (!x) x+=b;
    return x;
}
LL Mul(LL n,LL pi,LL pk)
{
    if (!n) return 1LL;
    LL ans=1LL;
    if (n/pk)
    {
        for (LL i=2; i<=pk; ++i)
            if (i%pi) ans=ans*i%pk;
        ans=fast_pow(ans,n/pk,pk);
    }
    for (LL i=2; i<=n%pk; ++i)
        if (i%pi) ans=ans*i%pk;
    return ans*Mul(n/pi,pi,pk)%pk;
}
LL C(LL n,LL m,LL Mod,LL pi,LL pk)
{
    if (m>n) return 0LL;
```

```

        LL a=Mul(n,pi,pk),b=Mul(m,pi,pk),c=Mul(n-m,pi,pk);
        LL k=0LL,ans;
        for (LL i=n; i; i/=pi) k+=i/pi;
        for (LL i=m; i; i/=pi) k-=i/pi;
        for (LL i=n-m; i; i/=pi) k-=i/pi;
        ans=a*inv(b,pk)%pk*inv(c,pk)%pk*fast_pow(pi,k,pk)%pk;
        return ans*(Mod/pk)%Mod*inv(Mod/pk,pk)%Mod;
    }
    int main()
    {
        scanf("%I64d%I64d%I64d",&n,&m,&MOD);
        for (LL x=MOD,i=2; i<=MOD; ++i)
            if (x%i==0)
            {
                LL pk=1LL;
                while (x%i==0) pk*=i,x/=i;
                ans=(ans+C(n,m,MOD,i,pk))%MOD;
            }
        printf("%I64d\n",ans);
    }

```

5. 扩展中国剩余定理

题目

poj2891

解同于方程组：

$$x \equiv r_1 \pmod{a_1}$$

$$x \equiv r_2 \pmod{a_2}$$

.....

$$x \equiv r_n \pmod{a_n}$$

其中模数不一定互质。

(结果可能不存在)

```

void ex_gcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (!b) {
        d = a, x = 1, y = 0;
    }
    else {
        ex_gcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}

```

```

    }
}
LL ex_crt(LL *m, LL *r, int n)
{
    LL M = m[1], R = r[1], x, y, d;
    for (int i = 2; i <= n; ++i) {
        ex_gcd(M, m[i], d, x, y);
        if ((r[i] - R) % d)
            return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R > 0 ? R : R + M;
}

```

6. 欧拉降幂

公式

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

递归处理 ϕ ，一个数取 ϕ 约 \log 次就成1了，暴力搞就行了

题意： $A^B \bmod C$ 其中B为 $10^{1000000}$ 。

题解：

$$A^B \bmod C = A^{B \bmod \phi(C) + \phi(C)} \bmod C$$

降幂公式 $\phi()$ 为欧拉函数

第一个是欧拉降幂互质情况，第二三个是广义欧拉降幂，不要求互质，但要求大小。

```

#define ll __int64
#define mod 1000000007
char a[1000006];
ll x, z;
ll quickpow(ll x, ll y, ll z)
{
    ll ans = 1;
    while(y)

```

```

    {
        if(y&1)
            ans=ans*x%z;
        x=x*x%z;
        y>>=1;
    }
    return ans;
}
ll phi(ll n)
{
    ll i,rea=n;
    for(i=2;i*i<=n;i++)
    {
        if(n%i==0)
        {
            rea=rea-rea/i;
            while(n%i==0)
                n/=i;
        }
    }
    if(n>1)
        rea=rea-rea/n;
    return rea;
}
int main()
{
    while(scanf("%I64d %s %I64d",&x,a,&z)!=EOF)
    {
        ll len=strlen(a);
        ll p=phi(z);
        ll ans=0;
        for(ll i=0;i<len;i++)
            ans=(ans*10+a[i]-'0')%p;
        ans+=p;
        printf("%I64d\n",quickpow(x,ans,z));
    }
    return 0;
}

```

7. 容斥求 x 的因子 k

```
/*容斥求 x 的因子 k gcd(a,x) = k 的个数 a 属于 1~m*/
vector<ll> cnt[N], q[N];
int m;
void init() {
    for(int i = 1; i <= m; i++) {
        for(int j = i; j <= m; j += i)
            q[j].push_back(i);
    }
    for(int i = 1; i <= m; i++) {
        cnt[i].resize(q[i].size() + 1); //动态申请空间
        for(int j = q[i].size() - 1; j >= 0; j--) {
            cnt[i][j] = m / q[i][j];
            for(int k = j + 1; k < q[i].size(); k++) {
                if(q[i][k] % q[i][j] == 0) //减去他的倍数
                    cnt[i][j] -= cnt[i][k];
            }
        }
    }
}
```

8. 多项式乘法

8.1. FFT

```
#define PI acos(-1.0)
const int N = 3e6 + 10;
int r[N]; // 1 -> 二进制的位数-1
struct Complex {
    double x, y;
    Complex (double xx = 0, double yy = 0) {x = xx, y = yy;}
    Complex friend operator + (Complex a, Complex b) { return
Complex(a.x + b.x , a.y + b.y);}
    Complex friend operator - (Complex a, Complex b) { return
Complex(a.x - b.x , a.y - b.y);}
    Complex friend operator * (Complex a, Complex b) { return
Complex(a.x * b.x - a.y * b.y , a.x * b.y + a.y * b.x);} //不懂的看复数
的运算那部分
};
typedef Complex cp;
```

```

cp a[N], b[N];

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

void fft(Complex *p, int op, int n) {
    for(int i = 1; i < n; i++) if(i > r[i]) swap(p[i], p[r[i]]);

    for(int i = 1; i < n; i <= 1) { //表示操作区间集的每个区间的长度
        cp Wn(cos(PI/i), op*sin(PI/i)); // 单元根
        for(int r = i << 1, j = 0; j < n; j += r) {//表示每个区间集的最
右边端位置
            cp w(1, 0); // 幂
            for(int k = 0; k < i; k++, w = w * Wn) { //只遍历左区间, 右
区间 0(1)得到
                cp X= p[j+k], Y = w * p[j+k+i];
                p[j+k] = X + Y; p[j+k+i] = X - Y;
            }
        }
    }
}

void mult(Complex *a, Complex *b, int m) {
    int n = 0, l = 0;
    for(n = 1; n <= m; n <= 1) l++; l--;
    for(int i = 0; i < n; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
1);
    fft(a, 1, n); fft(b, 1, n);
    for(int i = 0; i < n; i++) a[i] = a[i] * b[i];
    fft(a, -1, n);
}

int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 0; i <= n; i++) a[i].x = read();

```

```

for(int i = 0; i <= m; i++) b[i].x = read();
//1 表示从系数变为点值
//-1 表示从点值变为系数
mult(a, b, 2 * n);
for(int i = 0; i <= 2*n; i++)
    printf("%d%c", (int)(a[i].x/n + 0.5), i == m ? '\n' : ' ');
return 0;
}

```

8.2. NTT

```

#define PI acos(-1.0)
const int N = 3e6 + 10;
const int mod = 998244353;
int r[N]; // 1 -> 二进制的位数-1
ll a[N], b[N];

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

ll poww(ll a, ll n) {
    ll ans = 1;
    while(n) {
        if(n & 1) ans = ans * a % mod;
        n >>= 1;
        a = a * a % mod;
    }
    return ans;
}

void ntt(ll *p, int n, int op) {
    for(int i = 1; i < n; i++) if(i > r[i]) swap(p[i], p[r[i]]);

    for(int i = 1; i < n; i <= 1) { //表示操作区间集的每个区间的长度
        ll Wn = poww(3, (mod - 1) / (i < 1)); //原根
        if(op == -1) Wn = poww(Wn, mod - 2); // 求逆元
    }
}

```

```

        for(int r = i << 1, j = 0; j < n; j += r) { //表示每个区间集的最
        右边端位置
            ll w = 1; //幂
            for(int k = 0; k < i; k++, w = w * Wn % mod) { //只遍历左区
            间, 右区间 O(1)得到
                ll X= p[j+k], Y = w * p[j+k+i] % mod;
                p[j+k] = (X + Y) % mod; p[j+k+i] = ((X - Y) % mod +
                mod) % mod;
            }
        }
    }

    if (op==-1) for (int i=0,Inv=poww(n,mod-2);i<n;++i)
    p[i]=1ll*p[i]*Inv%mod;
}

void multi(ll *a, ll *b, int m) {
    int n, l = 0;
    for(n = 1; n <= m; n <= 1) l++; l--;
    for(int i = 0; i < n; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) <<
    1);
    ntt(a, n, 1); ntt(b, n, 1);
    //1 表示从系数变为点值
    //-1 表示从点值变为系数
    for(int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
    ntt(a, n, -1);
}

int main()
{
    int n, m, l;
    scanf("%d %d", &n, &m);
    for(int i = 0; i <= n; i++) a[i] = read();
    for(int i = 0; i <= m; i++) b[i] = read();

    multi(a, b, n + m);

    ll inv = poww(n, mod - 2);
    for(int i = 0; i <= m; i++)
        printf("%lld%c",a[i] * inv % mod, i == m ? '\n' : ' ');

    return 0;
}

```


8.3. FWT

```

void FWT_or(int *a,int opt)
{
    for(int i=1;i<N;i<=1)
        for(int p=i<<1,j=0;j<N;j+=p)
            for(int k=0;k<i;++k)
                if(opt==1)a[i+j+k]=(a[j+k]+a[i+j+k])%MOD;
                else a[i+j+k]=(a[i+j+k]+MOD-a[j+k])%MOD;
}
void FWT_and(int *a,int opt)
{
    for(int i=1;i<N;i<=1)
        for(int p=i<<1,j=0;j<N;j+=p)
            for(int k=0;k<i;++k)
                if(opt==1)a[j+k]=(a[j+k]+a[i+j+k])%MOD;
                else a[j+k]=(a[j+k]+MOD-a[i+j+k])%MOD;
}
void FWT_xor(int *a,int opt)
{
    for(int i=1;i<N;i<=1)
        for(int p=i<<1,j=0;j<N;j+=p)
            for(int k=0;k<i;++k)
            {
                int X=a[j+k],Y=a[i+j+k];
                a[j+k]=(X+Y)%MOD;a[i+j+k]=(X+MOD-Y)%MOD;
                if(opt==1)
                {
                    a[j+k]=111*a[j+k]*inv2%MOD,a[i+j+k]=111*a[i+j+k]*inv2%MOD;
                }
            }
}

```

xor

$$tf(A) = (tf(A_0) + tf(A_1), tf(A_0) - tf(A_1))$$

$$utf(A) = (utf(\frac{A_0+A_1}{2}), (\frac{A_0-A_1}{2}))$$

and

$$tf(A) = (tf(A_0) + tf(A_1), tf(A_1))$$

$$utf(A) = (utf(A_0) - utf(A_1), utf(A_1))$$

or

$$tf(A) = (tf(A_0), tf(A_1) + tf(A_0))$$

$$utf(A) = (utf(A_0), utf(A_1) - utf(A_0))$$

8.4. 原根

```
struct get_Omg {
    ll prime[25];
    int tot = 0;
    void get_prime_factor(ll x) {
        for(ll i = 2; i * i <= x; i++) {
            if(x % i == 0) {
                prime[++tot] = i;
                while(x%i == 0) x /= i;
            }
        }
        if(x > 1) prime[++tot] = x;
    }

    ll poww(ll a, ll n, ll mod) {
        ll ans = 1;
        while(n) {
            if(n & 1) ans = ans * a % mod;
            a = a * a % mod;
            n >>= 1;
        }
        return ans;
    }

    ll get(ll p) {
        get_prime_factor(p - 1);
        for(ll a = 2; a <= p; a++) {
            int flag = 0;
            for(int i = 1; i <= tot; i++) {
                ll x = (p - 1) / prime[i];
                if(poww(a, x, p) == 1) {
                    flag = 1; break;
                }
            }
            if(!flag) return a;
        }
    }
}Omg;
```

11. 三分

```
double solve(double MIN,double MAX)
{
    double Left, Right;
    double mid, midmid;
    double mid_value, midmid_value;
    Left = MIN;
    Right = MAX;
    while (Left +eps < Right)
    {
        mid = (Left + Right) / 2;
        midmid = (mid + Right) / 2;
        mid_value = Calc(mid);
        midmid_value = Calc(midmid);
        ///求最大值改成>= 最小值改成<=
        if (mid_value >= midmid_value) Right = midmid;
        else Left = mid;
    }
    return Left;
}
```

先递减在递增

```
int l = 0, r = m + 1;
while(l < r - 1 )
{
    int left = (r + l) / 2;
    int right = (left + r) / 2;
    if(f(left) <= f(right))
    {
        r = right;
    }
    else l = left;
}
```

12. 大数

```
constexpr int base = 1000000000;
constexpr int base_digits = 9;

struct bigint
{
    // value == 0 is represented by empty z
    vector<int> z; // digits

    // sign == 1 <==> value >= 0
    // sign == -1 <==> value < 0
    int sign;

    bigint() : sign(1) {}

    bigint(long long v) { *this = v; }

    bigint& operator=(long long v)
    {
        sign = v < 0 ? -1 : 1;
        v *= sign;
        z.clear();
        for (; v > 0; v = v / base)
            z.push_back((int)(v % base));
        return *this;
    }

    bigint(const string& s) { read(s); }

    bigint& operator+=(const bigint& other)
    {
        if (sign == other.sign)
        {
            for (int i = 0, carry = 0; i < other.z.size() || carry; ++i)
            {
                if (i == z.size())
                    z.push_back(0);
                z[i] += carry + (i < other.z.size() ? other.z[i] : 0);
                carry = z[i] >= base;
                if (carry)
                    z[i] -= base;
            }
        }
    }
}
```

```

    }
    else if (other != 0 /* prevent infinite loop */)
    {
        *this -= -other;
    }
    return *this;
}

friend bigint operator+(bigint a, const bigint& b)
{
    return a += b;
}

bigint& operator--(const bigint& other)
{
    if (sign == other.sign)
    {
        if (sign == 1 && *this >= other || sign == -1 && *this <=
other)
        {
            for (int i = 0, carry = 0; i < other.z.size() || carry;
++i)
            {
                z[i] -= carry + (i < other.z.size() ? other.z[i] :
0);

                carry = z[i] < 0;
                if (carry)
                    z[i] += base;
            }
            trim();
        }
        else
        {
            *this = other - *this;
            this->sign = -this->sign;
        }
    }
    else
    {
        *this += -other;
    }
    return *this;
}

```

```

friend bigint operator-(bigint a, const bigint& b)
{
    return a -= b;
}

bigint& operator*=(int v)
{
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < z.size() || carry; ++i)
    {
        if (i == z.size())
            z.push_back(0);
        long long cur = (long long)z[i] * v + carry;
        carry = (int)(cur / base);
        z[i] = (int)(cur % base);
    }
    trim();
    return *this;
}

bigint operator*(int v) const
{
    return bigint(*this) *= v;
}

friend pair<bigint, bigint> divmod(const bigint& a1, const bigint&
b1)
{
    int norm = base / (b1.z.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.z.resize(a.z.size());

    for (int i = (int)a.z.size() - 1; i >= 0; i--)
    {
        r *= base;
        r += a.z[i];
        int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()] : 0;
        int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size() - 1] :
0;

        int d = (int)((((long long)s1 * base + s2) / b.z.back()));
        r -= b * d;
    }
}

```

```

        while (r < 0)
            r += b, --d;
        q.z[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return {q, r / norm};
}

friend bigint sqrt(const bigint& a1)
{
    bigint a = a1;
    while (a.z.empty() || a.z.size() % 2 == 1)
        a.z.push_back(0);

    int n = a.z.size();

    int firstDigit = (int)::sqrt((double)a.z[n - 1] * base + a.z[n -
2]);
    int norm = base / (firstDigit + 1);
    a *= norm;
    a *= norm;
    while (a.z.empty() || a.z.size() % 2 == 1)
        a.z.push_back(0);

    bigint r = (long long)a.z[n - 1] * base + a.z[n - 2];
    firstDigit = (int)::sqrt((double)a.z[n - 1] * base + a.z[n -
2]);
    int q = firstDigit;
    bigint res;

    for (int j = n / 2 - 1; j >= 0; j--)
    {
        for (;;) --q
        {
            bigint r1 = (r - (res * 2 * base + q) * q) * base * base
+ (j > 0 ? (long long)a.z[2 * j - 1] * base + a.z[2 * j - 2] : 0);
            if (r1 >= 0)
            {
                r = r1;
                break;
            }
        }
    }
}

```

```

        }
    }
    res *= base;
    res += q;

    if (j > 0)
    {
        int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size()
+ 2] : 0;
        int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size()
+ 1] : 0;
        int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] :
0;
        q = (int)((((long long)d1 * base * base + (long long)d2 *
base + d3) / (firstDigit * 2));
    }
}

res.trim();
return res / norm;
}

bigint operator/(const bigint& v) const
{
    return divmod(*this, v).first;
}

bigint operator%(const bigint& v) const
{
    return divmod(*this, v).second;
}

bigint& operator/=(int v)
{
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int)z.size() - 1, rem = 0; i >= 0; --i)
    {
        long long cur = z[i] + rem * (long long)base;
        z[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
    return *this;
}

```



```

}

bigint operator/(int v) const
{
    return bigint(*this) /= v;
}

int operator%(int v) const
{
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = (int)z.size() - 1; i >= 0; --i)
        m = (int)((z[i] + m * (long long)base) % v);
    return m * sign;
}

bigint& operator*=(const bigint& v)
{
    *this = *this * v;
    return *this;
}

bigint& operator/=(const bigint& v)
{
    *this = *this / v;
    return *this;
}

bool operator<(const bigint& v) const
{
    if (sign != v.sign)
        return sign < v.sign;
    if (z.size() != v.z.size())
        return z.size() * sign < v.z.size() * v.sign;
    for (int i = (int)z.size() - 1; i >= 0; i--)
        if (z[i] != v.z[i])
            return z[i] * sign < v.z[i] * sign;
    return false;
}

bool operator>(const bigint& v) const
{
    return v < *this;
}

```

```

}

bool operator<=(const bigint& v) const
{
    return !(v < *this);
}

bool operator>=(const bigint& v) const
{
    return !(*this < v);
}

bool operator==(const bigint& v) const
{
    return !(*this < v) && !(v < *this);
}

bool operator!=(const bigint& v) const
{
    return *this < v || v < *this;
}

void trim()
{
    while (!z.empty() && z.back() == 0)
        z.pop_back();
    if (z.empty())
        sign = 1;
}

bool isZero() const
{
    return z.empty();
}

friend bigint operator-(bigint v)
{
    if (!v.z.empty())
        v.sign = -v.sign;
    return v;
}

bigint abs() const
{

```

```

        return sign == 1 ? *this : -*this;
    }

    long long longValue() const
    {
        long long res = 0;
        for (int i = (int)z.size() - 1; i >= 0; i--)
            res = res * base + z[i];
        return res * sign;
    }

    friend bigint gcd(const bigint& a, const bigint& b)
    {
        return b.isZero() ? a : gcd(b, a % b);
    }

    friend bigint lcm(const bigint& a, const bigint& b)
    {
        return a / gcd(a, b) * b;
    }

    void read(const string& s)
    {
        sign = 1;
        z.clear();
        int pos = 0;
        while (pos < s.size() && (s[pos] == '-' || s[pos] == '+'))
        {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = (int)s.size() - 1; i >= pos; i -= base_digits)
        {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            z.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream& stream, bigint& v)
    {

```

```

        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream& stream, const bigint& v)
    {
        if (v.sign == -1)
            stream << '-';
        stream << (v.z.empty() ? 0 : v.z.back());
        for (int i = (int)v.z.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.z[i];
        return stream;
    }

    static vector<int> convert_base(const vector<int>& a, int
old_digits, int new_digits)
    {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < p.size(); i++)
            p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int v : a)
        {
            cur += v * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits)
            {
                res.push_back(int(cur % p[new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int)cur);
        while (!res.empty() && res.back() == 0)
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

```

```

static vll karatsubaMultiply(const vll& a, const vll& b)
{
    int n = a.size();
    vll res(n + n);
    if (n <= 32)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint& v) const
{

```

```

        vector<int> a6 = convert_base(this->z, base_digits, 6);
        vector<int> b6 = convert_base(v.z, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < c.size(); i++)
        {
            long long cur = c[i] + carry;
            res.z.push_back((int)(cur % 1000000));
            carry = (int)(cur / 1000000);
        }
        res.z = convert_base(res.z, 6, base_digits);
        res.trim();
        return res;
    }
};

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    bigint a, b, sa, sb;
    cin >> a >> b;
    cout << a * b << endl;
    return 0;
}

```

13. Pell 方程

Pell方程是一类二元二次不定方程，其形如

$$x^2 - dy^2 = 1$$

其中 d 为一非完全平方数的正数。

对于Pell方程解的存在性和求法以及超出了本文的范畴，在此仅讨论在已知Pell方程最小解的情况下的一些操作。

假设我们已知了Pell方程的最小解 (x_0, y_0) ，那么其他的解可以由最小解的幂次得到，即

$$x_n + \sqrt{d}y_n = (x_0 + \sqrt{d}y_0)^n$$

或

$$x_n - \sqrt{d}y_n = (x_0 - \sqrt{d}y_0)^n$$

由此可以推知其通解公式的形式为

$$x_n = \frac{1}{2}[(x_0 + \sqrt{d}y_0)^n + (x_0 - \sqrt{d}y_0)^n]$$
$$y_n = \frac{1}{2\sqrt{d}}[(x_0 + \sqrt{d}y_0)^n - (x_0 - \sqrt{d}y_0)^n]$$

推得

$$x_n = x_0x_{n-1} + dy_0y_{n-1}$$

$$y_n = y_0x_{n-1} + x_0y_{n-1}$$

由此可以推知其递推公式的形式为

$$x_n = 2x_0x_{n-1} - x_{n-2}$$

$$y_n = 2x_0y_{n-1} - y_{n-2}$$

题目大意：求方程 $x^2 - n \cdot y^2 = 1$ 的最小正整数解。

分析：连分数求佩尔方程特解。由于要用高精度，用 java。

```
import java.math.BigInteger;
```

```
import java.util.Scanner;
```

```
public class Main
```

```
{
```

```
    public static void solve(int n)
```

```
    {
```

```
        BigInteger N, p1, p2, q1, q2, a0, a1, a2, g1, g2, h1, h2, p, q;
```

```
        g1 = q2 = p1 = BigInteger.ZERO;
```

```
        h1 = q1 = p2 = BigInteger.ONE;
```

```
        a0 = a1 = BigInteger.valueOf((int)Math.sqrt(1.0*n));
```

```
        BigInteger ans=a0.multiply(a0);
```

```
        if(ans.equals(BigInteger.valueOf(n)))
```

```
        {
```

```

        System.out.println("No solution!");
        return;
    }
    N = BigInteger.valueOf(n);
    while (true)
    {
        g2 = a1.multiply(h1).subtract(g1);
        h2 = N.subtract(g2.pow(2)).divide(h1);
        a2 = g2.add(a0).divide(h2);
        p = a1.multiply(p2).add(p1);
        q = a1.multiply(q2).add(q1);
        if
        (p.pow(2).subtract(N.multiply(q.pow(2))).compareTo(BigInteger.ONE) ==
        0) break;
        g1 = g2;h1 = h2;a1 = a2;
        p1 = p2;p2 = p;
        q1 = q2;q2 = q;
    }
    System.out.println(p+" "+q);
}

public static void main(String[] args)
{
    Scanner cin = new Scanner(System.in);
    while(cin.hasNextInt())
    {
        solve(cin.nextInt());
    }
}
}

```

pell 方程 $X^2 - nY^2 = 1$ 的第 k 项解 $\%mod$ $2 \leq n < 29, k < 1e9$ 。

如果我们求出Pell方程的最小正整数解后，就可以根据递推式求出所有的解。

$$\begin{aligned}x_n &= x_{n-1}x_1 + dy_{n-1}y_1 \\ y_n &= x_{n-1}y_1 + y_{n-1}x_1\end{aligned}$$

则根据上式我们可以构造矩阵，然后就可以快速幂了。

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_1 & dy_1 \\ y_1 & x_1 \end{bmatrix}^{k-1} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

这样就可以求出第k大的解。

```
int n, x, y;
const int maxn = 4;
const int mod = 8191;
typedef struct
{
    int m[maxn][maxn];
}Matrax;
Matrax a,per;
void sovle() //暴力法求特解 ,x,y 为特解
{
    y = 1;
    while(1)
    {
        x = (ll)sqrt(n * y * y + 1);
        if(x * x - n * y * y == 1)
        {
            break;
        }
        y++;
    }
}
```

```

}
void init() //矩阵快速幂初始化
{
    int i,j;
    a.m[0][0] = x % mod;
    a.m[0][1] = n * y % mod;
    a.m[1][0] = y % mod;
    a.m[1][1] = x % mod;
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            per.m[i][j] = (i == j);
        }
    }
}
Matrax multi(Matrax a,Matrax b) //矩阵乘法
{
    Matrax c;
    int k,i,j;
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            c.m[i][j] = 0; //初始化
            for(k=0;k<2;k++)
            {
                c.m[i][j] += a.m[i][k] * b.m[k][j];
            }
            c.m[i][j] %= mod;
        }
    }
    return c;
}
Matrax power(int k) //矩阵乘方
{
    Matrax p,ans = per;
    p = a;
    while(k)
    {
        if(k & 1)
        {
            ans = multi(ans,p);
            k--;
        }
    }
}

```

```

    }
    k /= 2;
    p = multi(p,p);
}
return ans;
}
//pell 方程  $X^2 - nY^2 = 1$  的第 k 项解
// $2 \leq n < 29, k < 1e9$ 。
int main()
{
    int k;
    while(scanf("%d%d",&n,&k) != EOF)
    {
        int yi = sqrt(n + 0.0);
        if(yi * yi == n) //完全平方数不满足使用佩尔方程
        {
            printf("No answers can meet such conditions\n");
            continue;
        }
        sovle(); //求一组特解 x,y
        init(); //初始化矩阵快速幂
        a = power(k-1); //k - 1 次方
        x = (a.m[0][0] * x % mod + a.m[0][1] * y % mod) % mod; //求 x
        y = sqrt(x*x-1);
        printf("%d %d\n",x,y);
    }
    return 0;
}

```

14. 拉格朗日插值法

```

const int mod=1e9+7;
const int N=1000105;
ll f[N],g[N],p1[N],p2[N],b[N];
/***** ?head *****/
*****/
namespace polysum
{
    #define rep(i,a,n) for (int i=a;i<n;i++)
    #define per(i,a,n) for (int i=n-1;i>=a;i--)
    ll powmod(ll a,ll b)
    {

```

```

    ll res=1;
    a%=mod;
    assert(b>=0);
    for(; b; b>>=1)
    {
        if(b&1)res=res*a%mod;
        a=a*a%mod;
    }
    return res;
}
ll calcn(int d,ll *a,ll n)    // a[0].. a[d] ?a[n]?
{
    if (n<=d) return a[n];
    p1[0]=p2[0]=1;
    rep(i,0,d+1)
    {
        ll t=(n-i+mod)%mod;
        p1[i+1]=p1[i]*t%mod;
    }
    rep(i,0,d+1)
    {
        ll t=(n-d+i+mod)%mod;
        p2[i+1]=p2[i]*t%mod;
    }
    ll ans=0;
    rep(i,0,d+1)
    {
        ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
        if ((d-i)&1) ans=(ans-t+mod)%mod;
        else ans=(ans+t)%mod;
    }
    return ans;
}
void init(int M)
{
    f[0]=f[1]=g[0]=g[1]=1;
    rep(i,2,M+5) f[i]=f[i-1]*i%mod;
    g[M+4]=powmod(f[M+4],mod-2);
    per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
}
ll polysum(ll m,ll *a,ll n)    // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
{
    for(int i=0; i<=m; i++) b[i]=a[i];
    b[m+1]=calcn(m,b,m+1);
}

```

```

        rep(i,1,m+2) b[i]=(b[i-1]+b[i])%mod;
        return calcn(m+1,b,n-1);
    }
} // polysum::init();

ll a[1000005];
int main()
{
    ll n,k;
    polysum::init(1000005);
    cin>>n>>k;
    if(k==0) {
        cout<<n<<endl;
        return 0;
    }
    a[0]=0;
    for(int i=1;i<=k+2;i++) //算出前 k+2 项，调用拉格朗日插值。
    {
        a[i]=(polysum::powmod(i,k))%mod;
    }
    ll ans=polysum::polysum(k+2,a,n+1)%mod;//求第 n 项要传 n+1
    cout<<ans%mod<<endl;
    return 0;
}

```

15. 杜教 BM

```

//BM: 解决递推式.请保证模数的平方不会爆 long long!!!
using VI = vector<int64_t> ;
class Linear_Seq
{
public:
    static const int N = 50010;///多项式系数最大值
    int64_t res[N],base[N],c[N],md[N],COEF[N]**COEF 是多项式系数*/,Mod;
    vector<int> Md;

    inline static int64_t gcdEx(int64_t a, int64_t b, int64_t&x,
int64_t& y)
    {
        if(!b) {x=1;y=0;return a;}
        int64_t d = gcdEx(b,a%b,y,x);
        y -= (a/b)*x;
    }
}

```

```

        return d;
    }

    static int64_t Inv(int64_t a, int64_t Mod) {
        int64_t x, y;
        return gcdEx(a, Mod, x, y)==1?(x%Mod+Mod)%Mod:-1;
    };

    inline void mul(int64_t *a,int64_t *b,int k) {
        fill(c,c+2*k,0) ;
        for(int i(0);i<k;++i)if(a[i])for(int j(0);j<k;++j)
            c[i+j]=(c[i+j]+a[i]*b[j])%Mod;
        for (int i(2*k-1);i>=k;--i) if (c[i])for(size_t
j(0);j<Md.size();++j)
            c[i-k+Md[j]]=(c[i-k+Md[j]]-c[i]*md[Md[j]])%Mod;
        copy(c,c+k,a) ;
    }

    int solve(int64_t n,VI a,VI b) { // a 系数 b 初值
b[n+1]=a[0]*b[n]+...
        int64_t ans(0),cnt(0);
        int k(a.size());
        for(int i(0);i<k;++i) md[k-i-1]=-a[i];
        md[k]=1 ; Md.clear() ;
        for(int i(0);i<k;++i) {
            res[i] = base[i] = 0;
            if (md[i]) Md.push_back(i);
        }
        res[0]=1;
        while ((1LL<<cnt)<=n) ++ cnt;
        for (int p(cnt);~p;-- p) {
            mul(res,res,k);
            if ((n>>p)&1) {
                copy(res,res+k,res+1) ;
                res[0]=0;
                for(size_t j(0);j<Md.size();++j)
                    res[Md[j]]=(res[Md[j]]-res[k]*md[Md[j]])%Mod;
            }
        }
        for(int i(0);i<k;++i) ans=(ans+res[i]*b[i])%Mod;
        return ans+(ans<0?Mod:0);
    }

    ///1-st*****模数是质数用这里*****/
    VI BM(VI s) {

```

```

VI C(1,1),B(1,1);
int L(0),m(1),b(1);
for(size_t n(0);n<s.size();++n) {
    int64_t d(0);
    for(int i(0);i<=L;++i) d=(d+(int64_t)C[i]*s[n-i])%Mod;
    if (!d) ++m;
    else {
        VI T(C);
        int64_t c(Mod-d*Inv(b,Mod)%Mod);
        while (C.size()<B.size()+m) C.push_back(0);
        for (size_t i(0);i<B.size();++i)
            C[i+m]=(C[i+m]+c*B[i])%Mod;
        if (2*L<=(int)n) {L=n+1-L; B=T; b=d; m=1;}
        else ++m ;
    }
}
/** //下边这样写能够输出递推式的系数.
printf("F[n] = ") ;
for(size_t i(0);i<C.size();++i) {
    COEF[i+1] = min(C[i],Mod-C[i]) ;
    if(i>0) {
        if(i != 1) printf(" + ") ;
        printf("%lld*F[n-%d]",COEF[i+1],i+1) ;
        putchar(i+1==C.size()?'\n':' ') ;
    }
}
*/
return C;
}
///1-ed*****模数是质数用这里*****

///2-st*****模数非质数用这里*****
inline static void extand(VI &a, size_t d, int64_t value = 0) {
    if (d <= a.size()) return; a.resize(d, value);
}
static int64_t CRT(const VI &c, const VI &m) {
    int n(c.size());
    int64_t M(1), ans(0);
    for (int i = 0; i < n; ++i) M *= m[i];
    for (int i = 0; i < n; ++i) {
        int64_t x,y,tM(M / m[i]);
        gcdEx(tM, m[i], x, y);
        ans = (ans + tM * x * c[i] % M) % M;
    }
}

```

```

        return (ans + M) % M;
    }

    static VI ReedsSloane(const VI &s, int64_t Mod) {
        auto L = [](const VI &a, const VI &b) {
            int da = (a.size()>1||((a.size()== 1&&a[0]))?a.size()-1:-
1000;
            int db = (b.size()>1||((b.size()== 1&&b[0]))?b.size()-1:-
1000;
            return max(da, db + 1);
        };
        auto prime_power = [&](const VI &s, int64_t Mod, int64_t p,
int64_t e) {
            // linear feedback shift register Mod p^e, p is prime
            vector<VI> a(e), b(e), an(e), bn(e), ao(e), bo(e);
            VI t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
            pw[0] = 1;
            for (int i(pw[0] = 1); i <= e; ++i) pw[i] = pw[i - 1] * p;
            for (int64_t i(0); i < e; ++i) {
                a[i] = {pw[i]}; an[i] = {pw[i]};
                b[i] = {0}; bn[i] = {s[0] * pw[i] % Mod};
                t[i] = s[0] * pw[i] % Mod;
                if (!t[i]) {t[i] = 1; u[i] = e;}
                else for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i]);
            }
            for (size_t k(1); k < s.size(); ++k) {
                for (int g(0); g < e; ++g) {
                    if (L(an[g], bn[g]) > L(a[g], b[g])) {
                        int id (e-1-u[g]);
                        ao[g] = a[id]; bo[g] = b[id];
                        to[g] = t[id]; uo[g] = u[id];
                        r[g] = k - 1;
                    }
                }
            }
            a = an; b = bn;
            for (int o(0); o < e; ++o) {
                int64_t d(0);
                for (size_t i(0); i < a[o].size() && i <= k; ++i)
                    d = (d + a[o][i] * s[k - i]) % Mod;
                if (d == 0) {t[o] = 1; u[o] = e;}
                else {
                    for (u[o]=0,t[o]=d;!(t[o]%p);t[o]/=p,++u[o]);
                    int g (e-1-u[o]);
                    if (!L(a[g], b[g])) {

```



```

        extend(bn[o], k + 1);
        bn[o][k] = (bn[o][k] + d) % Mod;
    } else {
        int64_t coef =
t[o]*Inv(to[g],Mod)%Mod*pw[u[o]-uo[g]]%Mod;
        int m(k-r[g]);
        extend(an[o],ao[g].size()+m);
    extend(bn[o],bo[g].size()+m);
        auto fun = [&](vector<VI> &vn,vector<VI>
&vo,bool f) {
            for (size_t i(0);i < vo[g].size(); ++i)
{
                vn[o][i+m] -= coef*vo[g][i]%Mod;
                if (vn[o][i + m]<0) vn[o][i+m] +=
Mod*(f?1:-1);
            }
            while (vn[o].size() && !vn[o].back())
vn[o].pop_back();
        } ;
        fun(an,ao,1) ;fun(bn,bo,-1) ;
    }
}
}
}
return make_pair(an[0], bn[0]);
};
vector<tuple<int64_t, int64_t, int> > fac;
for (int64_t i(2); i*i <= Mod; ++i)
    if (!(Mod % i)) {
        int64_t cnt(0),pw(1);
        while (!(Mod % i)) {Mod /= i; ++cnt; pw *= i;}
        fac.emplace_back(pw, i, cnt);
    }
if (Mod > 1) fac.emplace_back(Mod, Mod, 1);
vector<VI> as;
size_t n = 0;
for (auto &&x: fac) {
    int64_t Mod, p, e;
    VI a, b;
    std::tie(Mod, p, e) = x;
    auto ss = s;
    for (auto &&x: ss) x %= Mod;
    std::tie(a, b) = prime_power(ss, Mod, p, e);
    as.emplace_back(a);

```

```

        n = max(n, a.size());
    }
    VI a(n),c(as.size()),m(as.size());
    for (size_t i(0); i < n; ++i) {
        for (size_t j(0); j < as.size(); ++j) {
            m[j] = std::get<0>(fac[j]);
            c[j] = i < as[j].size() ? as[j][i] : 0;
        }
        a[i] = CRT(c, m);
    }
    return a;
}
//2-ed*****模数非质数用这里*****/

int64_t solve(VI a,int64_t n,int64_t Mod,bool prime=true) {
    VI c; this->Mod = Mod ;
    if(prime) c = BM(a);///如果已经知道系数了,直接输入到 c 就行了,不用调用 BM().
    else c = ReedsSloane(a,Mod);
    c.erase(c.begin()) ;
    for(size_t i(0);i<c.size();++i) c[i] = (Mod-c[i])%Mod;
    return solve(n,c,VI(a.begin(),a.begin()+c.size()));
}
}BMEX;
///BMEX.slove(初始值 vector[从 0 开始],要得到的项数,模数,模数是不是质数)

int64_t Mod (1000000000) ;

int64_t powEx(int64_t base,int64_t n,int64_t Mod>::Mod)
{
    int64_t ret(1);
    while(n){
        if(n&1) ret = ret*base%Mod ;
        base = base * base %Mod ;
        n >>= 1 ;
    }
    return ret % Mod ;
}

int main()
{
    ios::sync_with_stdio(false) ;
    cin.tie(0);

```

```

cout.tie(0);
int64_t n, m;
cin >> n >> m;
VI f({0, 1});
for (int i = 2; i < 1000; i++)
    f.push_back((f[i - 1] + f[i - 2]) % Mod);
for (auto& t : f) t = powEx(t, m);
for (int i = 1; i < 1000; i++)
    f[i] = (f[i - 1] + f[i]) % Mod;
printf("%lld\n", BMEX.solve(f, n, Mod, false));
return 0;
}

```

16. SG 函数

```

//f[N]:可改变当前状态的方式, N 为方式的种类, f[N]要在 getSG 之前先预处理
//SG[]:0~n 的 SG 函数值
//S[]:为 x 后继状态的集合
int f[N], SG[MAXN], S[MAXN];
void getSG(int n){
    int i, j;
    memset(SG, 0, sizeof(SG));
    //因为 SG[0]始终等于 0, 所以 i 从 1 开始
    for(i = 1; i <= n; i++){
        //每一次都要将上一状态 的 后继集合 重置
        memset(S, 0, sizeof(S));
        for(j = 0; f[j] <= i && j <= N; j++){
            S[SG[i-f[j]]] = 1; //将后继状态的 SG 函数值进行标记
        }
        for(j = 0; j <= N; j++) if(!S[j]){ //查询当前后继状态 SG 值中最小的非零值
            SG[i] = j;
            break;
        }
    }
}
}

```

17. 线性基

如何求最大值

完整的说，是如何求在一个序列中，取若干个数，使得它们的异或和最大。

首先构造出这个序列的线性基，然后从线性基的最高位开始，假如当前的答案异或线性基的这个元素可以变得更大，那么就异或它，答案的初值为0。

代码如下：

```
1 ll ans()
2 {
3     ll anss=0;
4     for(int i=50;i>=0;i--)//记得从线性基的最高位开始
5         if((anss^d[i])>anss)anss^=d[i];
6     return anss;
7 }
```

```
/*
    维护 【1, R】的线性基
    询问 【L, R】挑选一些数最大异或和
*/
void add(int x, int value) {
    int t = x;
    for (int i = 30; i >= 0; i--) {
        if (value & (1 << i)) {
            if (!d[i]) {
                d[i] = value;
                pos[i] = x;
                break;
            }
            else if (pos[i] < x) {
                swap(d[i], value);
                swap(pos[i], x);
            }
            value ^= d[i];
        }
    }
    for (int i = 0; i <= 30; i++)
        v[t][i] = d[i], p[t][i] = pos[i];
}

int ask(int l, int r) {
    int res = 0;
    for (int i = 30; i >= 0; i--)
        if (p[r][i] >= l && (res ^ v[r][i]) > res)
            res ^= v[r][i];
    return res;
}
```

```
}
```

```
#define maxr 100000
#define maxn 100005
#define rep(i, l, r) for (register int i = l; i <= r; i++)
#define per(i, r, l) for (register int i = r; i >= l; i--)
#define srep(i, l, r) for (register int i = l; i < r; i++)
#define sper(i, r, l) for (register int i = r; i > l; i--)
#define gc() ((p1 == p2 && (p2 = (p1 = buffer) + fread(buffer, 1,
maxr, stdin), p1 == p2)) ? EOF : *p1++)
#define ms(a, b) memset(a, b, sizeof(a))
#define uint unsigned int
#define Author Renatus
using namespace std;

/*
** 这是一个 线性基 (LinearBase) 的模板
** 有关信息维护的标准以 LOJ#113. 最大异或和, LOJ#114. k 大异或和 的题目要求为准
** 在实际使用中, 可以对有关部分作出适当改变以适应不同题目要求
** 需要说明的部分已经给出较为详尽的注释
** 作者: Renatus
*/

char buffer[maxr], *p1, *p2;
template <class T> void read(T& x){
    char ch = gc(); x = 0; bool f = 1;
    while (ch != '-' && !('0' <= ch && ch <= '9')) ch = gc();
    if (ch == '-') f = 0, ch = gc();
    while ('0' <= ch && ch <= '9') x = (x << 1) + (x << 3) + ch - '0',
ch = gc();
    x = (f) ? x : -x;
}
// 有的时候, 数组开小不会产生问题, 但是当打开 -O2 优化后则极易产生问题
struct LinearBase{
    uint a[32]; //非常重要! 数组一定不要开小
    LinearBase() { ms(a, 0); }
    int get_size(){
        int siz = 0;
        per(i, 31, 0) if (a[i]) siz++;
        return siz;
    }
    bool insert(uint v){
```

```

uint now = (1 << 31);
per(i, 31, 0) {
    if (v & now){
        if (!a[i]) { a[i] = v; return true; }
        else v ^= a[i];
    }
    if (!v) return false;
    now >>= 1;
}
//assert(false);
return true;
}

void process(){ //将 01 矩阵化为行简化阶梯型（高斯消元法）
    per(i, 31, 0) if (a[i]) rep(j, i + 1, 31) a[j] = min(a[j],
a[j] ^ a[i]);
}

uint get_kth_min(uint k){ //注意此处的 k 可以为 0，即空集（最小的那个），必须在行简化阶梯型下使用
    uint ans = 0;
    rep(i, 0, 31){
        if (a[i]) {
            if (k & 1) ans ^= a[i];
            k >>= 1;
            if (!k) return ans;
        }
    }
    return (k) ? -1 : ans;
} // 注意，此函数只能用于 不同的 异或和中求第 k 小

bool express(uint v){
    uint now = (1 << 31);
    per(i, 31, 0) {
        if (v & now){
            if (!a[i]) return false;
            else v ^= a[i];
        }
        if (!v) return true;
        now >>= 1;
    }
    //assert(false);
    return false;
}

void print(){
    per(i, 31, 0){
        uint x = a[i], now = ((uint)1 << 31);

```

```

        per(i, 31, 0) printf("%d", (x & now) ? 1 : 0), now >>= 1;
        printf("\n");
    }
}
void print(int bit){
    per(i, bit, 0){
        uint x = a[i], now = ((uint)1 << bit);
        per(i, bit, 0) printf("%d", (x & now) ? 1 : 0), now >>= 1;
        printf("\n");
    }
}
}; //在线性基中, int 一定要谨慎使用

LinearBase merge(LinearBase& A, LinearBase& B){
    LinearBase C;
    per(i, 31, 0) if (A.a[i]) C.insert(A.a[i]);
    per(i, 31, 0) if (B.a[i]) C.insert(B.a[i]);
    return C;
} //注意, 合并操作的复杂度为  $O(\log^2)$ 

LinearBase intersect(LinearBase& A, LinearBase& B){
    LinearBase C, D, E;
    per(i, 31, 0) if (A.a[i]) C.insert(A.a[i]), D.a[i] = (uint)1 << i;
    rep(i, 0, 31){
        if (!B.a[i]) continue;
        uint v = 0, x = B.a[i], now = (1 << 31);
        per(i, 31, 0) {
            if (x & now){
                if (C.a[i]) x ^= C.a[i], v ^= D.a[i];
                else { C.a[i] = x, D.a[i] = v; break; }
            }
            now >>= 1;
        }
        if (!x) {
            rep(i, 0, 31){
                if (v & 1) x ^= C.a[i];
                v >>= 1;
            }
            E.insert(x);
        }
    }
    return E;
} //注意, 求交操作的复杂度为  $O(\log^2)$ 

```

```

LinearBase A[maxn << 1];
int qx, qy;
uint qd;
void pu(int o){
    A[o] = intersect(A[o << 1], A[o << 1 | 1]);
}
void init(int l, int r, int o){
    if (l == r){
        int sz; uint x;
        read(sz); rep(i, 1, sz) read(x), A[o].insert(x);
        return;
    }
    int m = ((r - l) >> 1) + 1;
    init(l, m, o << 1), init(m + 1, r, o << 1 | 1);
    pu(o);
}

bool query(int l, int r, int o){
    if (qx <= l && r <= qy) return A[o].express(qd);
    int m = ((r - l) >> 1) + 1;
    if (qx <= m) if (!query(l, m, o << 1)) return false;
    if (qy > m) if (!query(m + 1, r, o << 1 | 1)) return false;
    return true;
}

int main(){
    //注意，示例部分也非常重要，请仔细阅读

    /*
        //LOJ#113. 最大异或和
        ll x;
        int n;
        read(n);
        rep(i, 1, n) read(x), A.insert(x);
        A.process(); //要先化为行简化阶梯型
        printf("%lld", A.get_kth_min((1ll << A.get_size()) - 1));
    */

    /*
        //LOJ#114. k 大异或和
        ll x;
        int n, m;
        read(n);
        rep(i, 1, n) read(x), A.insert(x);
    */

```



```

        A.process();
        read(m);
        int d = (A.get_size() < n) ? 1 : 0; //重要: 修正 0 的非空集表示
        rep(i, 1, m) read(x), printf("%lld\n", A.get_kth_min(x - d));
    */

    int n, m;
    read(n), read(m);
    init(1, n, 1);
    rep(i, 1, m){
        read(qx), read(qy), read(qd);
        printf(query(1, n, 1) ? "YES\n" : "NO\n");
    }
    return 0;
}

```

18. BSGS

```

const ll INF = 1e18
ll p;

ll poww(ll a, ll n) {
    ll ans = 1;
    while(n) {
        if(n & 1) ans = a * ans % p;
        a = a * a % p;
        n >>= 1;
    }
    return ans;
}

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

struct BSGS {
    ll V, m, lim, n;

```

```

inline ll getsqrt(ll n) {
    //适当更改块的大小和 Hash 的块的大小
    ll x = (ll)pow(n, 2.0 / 3);    //需要预处理的大小
    return x;
}

inline void init(ll a, ll _n) {    //预处理
    n = _n;
    hs.init();    //可以换成 map
    m = getsqrt(n);
    lim = n / m + 1;    //剩余查询处理范围, 查询的  $a^m$  的上界
    V = qpow(qpow(a, m, n), n-2, n);    //  $1/a^m$ 
    //预处理  $a^d \quad d \in [0, m-1]$ ;
    hs.insert(1, 0);
    ll e = 1;
    for (int i = 1; i < m; ++i) {    //预处理 1-m 范围
        e = e * a % n;
        if (hs.find(e) == -1) {
            hs.insert(e, i);
        }
    }
}

// $a^x = b \% n$ 
//需要保证  $\gcd(a, n) = 1$ , 即存在逆元
//如果有解输出的是最小的正整数解, 否则输出 -1
inline ll cal(ll b) {
    if (b == 1) return 0;
    for (int i = 0; i < lim; ++i) {
        int j = hs.find(b);
        if (j != -1) return 1ll * i * m + j;
        b = V * b % n;
    }
    return -1;
}

}bsgs;

void extgcd(ll a, ll b, ll& d, ll& x, ll& y) {
    if (!b) { d=a; x=1; y=0; }
    else { extgcd(b, a%b, d, y, x); y-=x*(a/b); }
}

int main()
{

```

```

int n, k;
scanf("%d %d", &n, &k);
while(n--) {
    ll y, z;
    scanf("%lld %lld %lld", &y, &z, &p);
    if(k == 1) printf("%lld\n", (y%p, z));
    else if(k == 2) {
        ll d, xx, yy;
        extgcd(y, p, d, xx, yy);
        if(z % d) puts("Orz, I cannot find x!");
        else {
            z = z/d;
            p = p/d;
            xx = (xx*z % p + p) % p;
            printf("%lld\n", xx);
        }
    }
    else if(k == 3){
        ll v = BSGS(y%p, z%p);
        if(v == -1) puts("Orz, I cannot find x!");
        else {
            printf("%lld\n", v);
        }
    }
}

return 0;
}

```

扩展BSGS

对于 $\gcd(y, p) \neq 1$ 怎么办？

我们把它写成 $y * y^{x-1} + k * p = z, k \in \mathbb{Z}$ 的形式

根据 $exgcd$ 的理论

那么如果 y, p 的 \gcd 不是 z 的约数就不会有解

设 $d = \gcd(y, p)$

那么

$$\frac{y}{d} * y^{x-1} + k * \frac{p}{d} = \frac{z}{d}$$

递归到 $d = 1$

设之间的所有的 d 的乘积为 g ，递归 c 次

令 $x' = x - c, p' = \frac{p}{g}, z' = \frac{z}{g}$

那么

$$y^{x'} * \frac{y^c}{g} = z' \pmod{p'}$$

那么BSGS求解就好了

```
# include <bits/stdc++.h>
# define RG register
# define IL inline
# define Fill(a, b) memset(a, b, sizeof(a))
# define File(a) freopen(a".in", "r", stdin), freopen(a".out", "w",
stdout)
using namespace std;
typedef long long ll;

template <class Int>
IL void Input(RG Int &x){
    RG int z = 1; RG char c = getchar(); x = 0;
    for(; c < '0' || c > '9'; c = getchar()) z = c == '-' ? -1 : 1;
    for(; c >= '0' && c <= '9'; c = getchar()) x = (x << 1) + (x << 3)
+ (c ^ 48);
    x *= z;
}

map <int, int> pw;
```

```

IL int Gcd(RG int x, RG int y){
    return !y ? x : Gcd(y, x % y);
}

IL int Pow(RG ll x, RG ll y, RG int p){
    RG ll ret = 1;
    for(; y; x = x * x % p, y >>= 1)
        if(y & 1) ret = ret * x % p;
    return ret;
}

int a, b, p;

IL int EX_BSGS(){
    if(b == 1) return 0;
    pw.clear();
    RG int cnt = 0, t = 1, s, x, m;
    for(RG int d = Gcd(a, p); d != 1; d = Gcd(a, p)){
        if(b % d) return -1;
        ++cnt, b /= d, p /= d, t = 1LL * t * a / d % p;
        if(b == t) return cnt;
    }
    s = b, m = sqrt(p) + 1;
    for(RG int i = 0; i < m; ++i){
        pw[s] = i;
        s = 1LL * s * a % p;
    }
    x = Pow(a, m, p), s = t;
    for(RG int i = 1; i <= m; ++i){
        s = 1LL * s * x % p;
        if(pw.count(s)) return i * m - pw[s] + cnt;
    }
    return -1;
}

int ans;

int main(RG int argc, RG char* argv[]){
    for(Input(a), Input(p), Input(b); a + b + p;){
        a %= p, b %= p, ans = EX_BSGS();
        if(ans < 0) puts("No Solution");
        else printf("%d\n", ans);
        Input(a), Input(p), Input(b);
    }
}

```

```
}  
return 0;  
}
```

19. 二次同余式

```
struct NumSolve {  
    struct T {  
        ll x, y;  
    };  
    ll w;  
  
    T mul_two(T a, T b, ll p) {  
        T ans;  
        ans.x = (a.x * b.x % p + a.y * b.y % p * w % p) % p;  
        ans.y = (a.x * b.y % p + a.y * b.x % p) % p;  
        return ans;  
    }  
  
    T qpow_two(T a, ll n, ll p) {  
        T ans;  
        ans.x = 1;  
        ans.y = 0;  
        while (n) {  
            if (n & 1)  
                ans = mul_two(ans, a, p);  
            n >>= 1;  
            a = mul_two(a, a, p);  
        }  
        return ans;  
    }  
  
    ll qpow(ll a, ll n, ll p) {  
        ll ans = 1;  
        a %= p;  
        while (n) {  
            if (n & 1)  
                ans = ans * a % p;  
            n >>= 1;  
            a = a * a % p;  
        }  
        return ans % p;  
    }  
};
```

```

}

ll Legendre(ll a, ll p) { return qpow(a, (p - 1) >> 1, p); }

ll solve(ll n, ll p) {
    if (p == 2)
        return 1;
    if (Legendre(n, p) + 1 == p)
        return -1;
    ll a, t;
    while (1) {
        a = rand() % p;
        t = a * a - n;
        w = (t % p + p) % p;
        if (Legendre(w, p) + 1 == p)
            break;
    }
    T tmp;
    tmp.x = a;
    tmp.y = 1;
    T ans = qpow_two(tmp, (p + 1) >> 1, p);
    return ans.x;
}

pair<ll, ll> get_ans(ll a, ll b, ll c, ll p) {
    ll A = b * b - a * c * 4, d;
    A %= p;
    if (A <= 0)
        A += p;
    A %= p;
    if (A != 0)
        d = solve(A, p);
    else
        d = 0;
    if (d == -1)
        return pair<ll, ll>(-1, -1);
    ll f = p - d, x1, x2;
    x1 = ((d - b + p) % p) * qpow(a * 2ll % p, p - 2, p) % p;
    x2 = ((f - b + p) % p) * qpow(a * 2ll % p, p - 2, p) % p;
    if (x1 > x2)
        swap(x1, x2);
    return pair<ll, ll>(x1, x2);
}
} Robot;

```

```

ll b, c, p = 1e9 + 7, x, y;
pair<ll, ll> pi;

int main()
{
    int T;
    scanf("%d", &T);
    while(T--) {
        scanf("%lld %lld", &b, &c);
        pi = Robot.get_ans(1, -b, c, p);
        if(pi.first == -1) { puts("-1 -1"); continue; }
        x = pi.first;
        y = (b - x + p) % p;
        if(x > y) swap(x, y);
        printf("%lld %lld\n", x, y);
    }

    return 0;
}

```

20. 平方和公式

平方和公式

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^n i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$ans = \sum_{i=1}^n ig\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$\mu(lcm(i, j)) = \mu(i)\mu(j)\mu(gcd(i, j))$$

$$\begin{aligned}\varphi(i)\varphi(j) &= i \prod_{p|i} \frac{p-1}{p} j \prod_{p|j} \frac{p-1}{p} \\ &= ij \prod_{p|ij} \frac{p-1}{p} \prod_{q|\gcd(i,j)} \frac{p-1}{p}\end{aligned}$$

$$\therefore \varphi(i)\varphi(j)\gcd(i,j) = \varphi(ij)\varphi(\gcd(i,j))$$

solution:

$a > b$ 且 $\gcd(a,b)=1$, 有 $\gcd(a^n - b^n, a^m - b^m) = a^{\gcd(n,m)} - b^{\gcd(n,m)}$

21. 杜教筛

前置知识

积性函数

积性函数：对于任意互质的整数 a, b 有 $f(ab) = f(a)f(b)$ 则称 $f(x)$ 为数论函数。

完全积性函数：对于任意整数 a, b 有 $f(ab) = f(a)f(b)$ 为数论函数。

- 常见的积性函数： φ, μ, σ, d
- 常见的完全积性函数： ϵ, I, id

这里特殊解释一下 ϵ, I, id 分别是什么意思： $\epsilon(n) = [n = 1], I(n) = 1, id(n) = n$

狄利克雷卷积

设 f, g 是两个数论函数，它们的狄利克雷卷积是：
$$(f * g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$$

性质：满足交换律，结合律

单位元： ϵ （即 $f * \epsilon = f$ ）

结合狄利克雷卷积得到的几个性质：

1. $\mu * I = \epsilon$
2. $\varphi * I = id$
3. $\mu * id = \varphi$

莫比乌斯反演

若

$$g(n) = \sum_{d|n} f(d)$$

则

$$f(n) = \sum_{d|n} \mu(d)g\left(\frac{n}{d}\right)$$

证明：这里需要用到前面提到的性质： $\mu * I = \epsilon$

给出的条件等价于 $g = f * I$

所以 $g * \mu = f * I * \mu = f * \epsilon = f$ 即 $g * \mu = f$ 即 结论。

杜教筛

设现在要求积性函数 f 的前缀和，设 $\sum_{i=1}^n f(i) = S(n)$ 。

再找一个积性函数 g ，则考虑它们的狄利克雷卷积的前缀和

$$\begin{aligned} & \sum_{i=1}^n (f * g)(i) \\ &= \sum_{i=1}^n \sum_{d|i} f(d) g\left(\frac{i}{d}\right) \\ &= \sum_{d=1}^n g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i) \\ &= \sum_{d=1}^n g(d) S\left(\left\lfloor \frac{n}{d} \right\rfloor\right) \end{aligned}$$

其中得到第一行是根据狄利克雷卷积的定义。

得到第二行则是先枚举 d 提出 g 。

得到第三行则是把 $\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i)$ 替换为 $S(\lfloor \frac{n}{d} \rfloor)$

接着考虑 $g(1)S(n)$ 等于什么。

可以发现，他就等于

$$\sum_{i=1}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

(可以理解成从1开始的前缀和减去从2开始的前缀和就是第一项)

前面这个式子 $\sum_{i=1}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$ ，根据刚才的推导，他就等于 $\sum_{i=1}^n (f * g)(i)$

所以得到杜教筛的核心式子：

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

积性函数的性质与例子

1. 常见的积性函数有

1. 除数函数 $\sigma_k(n) = \sum_{d|n} d^k$, 表示 n 的约数的 k 次幂和, 注意 $\sigma_k(n)$ 与 $\sigma^k(n)$ 是不同的。
2. 约数个数函数 $\tau(n) = \sigma_0(n) = \sum_{d|n} 1$, 表示 n 的约数个数, 一般也写为 $d(n)$ 。
3. 约数和函数 $\sigma(n) = \sigma_1(n) = \sum_{d|n} d$, 表示 n 的约数之和。
4. 欧拉函数 $\varphi(n) = \sum_{i=1}^n [(n, i) = 1] \cdot 1$, 表示不大于 n 且与 n 互质的正整数个数, 另外 $\sum_{i=1}^n [(n, i) = 1] \cdot i = \frac{n \cdot \varphi(n) + [n=1]}{2}$, 且对于正整数 $n > 2$ 来说 $\varphi(n)$ 是偶数。
5. 莫比乌斯函数 $\mu(n)$, 在狄利克雷卷积的乘法中与恒等函数互为逆元, $\mu(1) = 1$, 对于无平方因子数 $n = \prod_{i=1}^t p_i$ 有 $\mu(n) = (-1)^t$, 对于有平方因子数 n 有 $\mu(n) = 0$ 。
6. 元函数 $e(n) = [n = 1]$, 狄利克雷卷积的乘法单位元, 完全积性。
7. 恒等函数 $I(n) = 1$, 完全积性。
8. 单位函数 $id(n) = n$, 完全积性。
9. 幂函数 $id^k(n) = n^k$, 完全积性。

2. 关于莫比乌斯函数和欧拉函数有两个经典的公式

1. $[n = 1] = \sum_{d|n} \mu(d)$, 将 $\mu(d)$ 看作是容斥的系数即可证明。
2. $n = \sum_{d|n} \varphi(d)$, 将 $\frac{i}{n} (1 \leq i \leq n)$ 化为最简分数统计个数即可证明。
3. 若 $f(n)$ 为积性函数, 则对于正整数 $n = \prod_{i=1}^t p_i^{k_i}$ 有 $f(n) = \prod_{i=1}^t f(p_i^{k_i})$; 若 $f(n)$ 为完全积性函数, 则对于正整数 $n = \prod_{i=1}^t p_i^{k_i}$ 有 $f(n) = \prod_{i=1}^t f(p_i)^{k_i}$ 。

狄利克雷卷积与莫比乌斯反演

1. 数论函数 f 和 g 狄利克雷卷积定义为 $(f * g)(n) = \sum_{d|n} f(d) \cdot g(\frac{n}{d})$, 狄利克雷卷积满足交换律、结合律, 对加法满足分配律, 存在单位元函数 $e(n) = [n = 1]$ 使得 $f * e = f = e * f$, 若 f 和 g 为积性函数则 $f * g$ 也为积性函数。
2. 狄利克雷卷积的一个常用技巧是对于积性函数 f 与恒等函数 I 的卷积的处理, 例如 $n = \prod_{i=1}^t p_i^{k_i}, g(n) = \sum_{d|n} f(d)$, 则有 $g(n) = \prod_{i=1}^t \sum_{j=0}^{k_i} f(p_i^j)$ 。
3. 莫比乌斯反演也是对于 $g(n) = \sum_{d|n} f(d)$ 的讨论, 但是不要求 f 是积性函数, 适用于已知 $g(n)$ 求 $f(n)$ 的情况, 由于 $I * \mu = e$, 则 $g * \mu = f * I * \mu = f * e = f$, 即 $f(n) = \sum_{d|n} g(d) \cdot \mu(\frac{n}{d})$, 类似地有 $g(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} g(d) \cdot \mu(\frac{d}{n})$, 二项式反演也是类似的技巧。有一个例子可以看出欧拉函数和莫比乌斯函数之间的关系, 由于 $\sum_{d|n} \varphi(d) = id(n)$, 所以 $\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$, 也即 $\frac{\varphi(n)}{n} = \sum_{d|n} \frac{\mu(d)}{d}$ 。

```
#define ll long long
const int N = 7e6 + 5;
unordered_map<int, ll> Sphi;
unordered_map<int, int> Smu;

int prime[N], mu[N];
ll phi[N];
bool check[N];

void init() {
    int tot = 0;
    phi[1] = mu[1] = 1;
    for(int i = 2; i < N; i++) {
```

```

        if(!check[i]) {
            prime[++tot] = i;
            mu[i] = -1;
            phi[i] = i - 1;
        }
        for(int j = 1; j <= tot; j++) {
            if(i * prime[j] >= N) break;
            check[i * prime[j]] = true;
            if(i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            mu[i * prime[j]] = -mu[i];
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    for(int i = 1; i < N; i++)
        mu[i] += mu[i - 1], phi[i] += phi[i - 1];
}

int cal_Smu(int n) {
    if(n < N-1) return mu[n];
    if(Smu[n]) return Smu[n];
    int last, ret = 1;
    for(int i = 2; i <= n; i = last + 1) {
        last = n / (n / i);
        ret -= cal_Smu(n / i) * (last - i + 1);
    }
    return Smu[n] = ret;
}

ll cal_Sphi(int n) {
    if(n < N-1) return phi[n];
    if(Sphi[n]) return Sphi[n];
    ll ret = 1LL*(1 + n) * n / 2;
    int last;
    for(int i = 2; i <= n; i = last + 1) {
        last = n / (n / i);
        ret -= cal_Sphi(n / i) * (last - i + 1);
    }
    return Sphi[n] = ret;
}

```

```

int main()
{
    init(); int T; scanf("%d", &T);
    while(T--) {
        int n; scanf("%d", &n);
        printf("%lld %d\n", cal_Sphi(n), cal_Smu(n));
    }
    return 0;
}

```

22. 勾股数

已知 a , 且 $a^2+b^2 = c^2$ 求 b, c ,

解:

已知 a , 其中一组解为:

若 a 为奇数 则 : $b = a*(a/2)+a/2, c = b+1$;

若 a 为偶数 则: $b = (a*(a/2))/2 - 1, c = b + 2$;

23. 高斯消元

```

const int MAXN = 50;
int a[MAXN][MAXN]; //增广矩阵
int x[MAXN]; //解集
bool free_x[MAXN]; //标记是否是不确定的变元

inline int lcm(int a, int b) {
    return a / __gcd(a, b) * b; //先除后乘防溢出
}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，
// -1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)
// 有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为
var+1,分别为 0 到 var.

```

```

int Gauss(int equ, int var) {
    int i, j, k;
    int max_r; // 当前这列绝对值最大的行.
    int col;    //当前处理的列
    int ta, tb, LCM, temp, free_x_num, free_index;

    for (int i = 0; i <= var; i++) {
        x[i] = 0;
        free_x[i] = true;
    }

    //转换为阶梯阵.
    col = 0; // 当前处理的列
    for (k = 0; k < equ && col < var; k++, col++) { // 枚举当前处理的
行.
        // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r = k;
        for (i = k + 1; i < equ; i++) {
            if (abs(a[i][col]) > abs(a[max_r][col]))
                max_r = i;
        }
        if (max_r != k) { // 与第 k 行交换.
            for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
        }
        if (a[k][col] == 0) { // 说明该 col 列第 k 行以下全是 0 了, 则处理当前行的下一列.
            k--;
            continue;
        }
        for (i = k + 1; i < equ; i++) { // 枚举要删去的行.
            if (a[i][col] != 0) {
                LCM = lcm(abs(a[i][col]), abs(a[k][col]));
                ta = LCM / abs(a[i][col]);
                tb = LCM / abs(a[k][col]);
                if (a[i][col] * a[k][col] < 0)
                    tb = -tb; //异号的情况是相加
                for (j = col; j < var + 1; j++) {
                    a[i][j] = a[i][j] * ta - a[k][j] * tb;
                }
            }
        }
    }
}

```

```

// Debug();

// 1. 无解的情况：化简的增广阵中存在 $(0, 0, \dots, a)$ 这样的行( $a \neq 0$ )。
for (i = k; i < equ;
    i++) { // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换
中的交换就会影响，则要记录交换。
    if (a[i][col] != 0)
        return -1;
}

// 2. 无穷解的情况：在  $\text{var} * (\text{var} + 1)$  的增广阵中出现 $(0, 0, \dots, 0)$ 这
样的行，即说明没有形成严格的上三角阵。
// 且出现的行数即为自由变元的个数。
if (k < var) {
    // 首先，自由变元有  $\text{var} - k$  个，即不确定的变元至少有  $\text{var} - k$  个。
    for (i = k - 1; i >= 0; i--) {
        // 第 i 行一定不会是 $(0, 0, \dots, 0)$ 的情况，因为这样的行是在第 k 行
到第 equ 行。
        // 同样，第 i 行一定不会是 $(0, 0, \dots, a)$ ， $a \neq 0$  的情况，这样的
无解的。

        free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超
过 1 个，则无法求解，它们仍然为不确定的变元。
        for (j = 0; j < var; j++) {
            if (a[i][j] != 0 && free_x[j])
                free_x_num++, free_index = j;
        }
        if (free_x_num > 1)
            continue; // 无法求解出确定的变元。
        // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，
且该变元是确定的。
        temp = a[i][var];
        for (j = 0; j < var; j++) {
            if (a[i][j] != 0 && j != free_index)
                temp -= a[i][j] * x[j];
        }
        x[free_index] = temp / a[i][free_index]; // 求出该变元。
        free_x[free_index] = 0; // 该变元是确定的。
    }
    return var - k; // 自由变元有  $\text{var} - k$  个。
}

// 3. 唯一解的情况：在  $\text{var} * (\text{var} + 1)$  的增广阵中形成严格的上三角阵。
// 计算出  $x_{n-1}, x_{n-2} \dots x_0$ 。
for (i = var - 1; i >= 0; i--) {
    temp = a[i][var];
    for (j = i + 1; j < var; j++) {

```



```

        if (a[i][j] != 0)
            temp -= a[i][j] * x[j];
    }
    if (temp % a[i][i] != 0)
        return -2; // 说明有浮点数解，但无整数解。
    x[i] = temp / a[i][i];
}
return 0;
}

int main(void) {
    int i, j, equ, var;
    while (scanf("%d %d", &equ, &var) != EOF) {
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++) {
            for (j = 0; j < var + 1; j++) {
                scanf("%d", &a[i][j]);
            }
        }
        //      Debug();
        int free_num = Gauss(equ, var);
        if (free_num == -1)
            printf("无解!\n");
        else if (free_num == -2)
            printf("有浮点数解，无整数解!\n");
        else if (free_num > 0) {
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++) {
                if (free_x[i])
                    printf("x%d 是不确定的\n", i + 1);
                else
                    printf("x%d: %d\n", i + 1, x[i]);
            }
        } else {
            for (i = 0; i < var; i++) {
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}

```

```

const int maxn = 500 + 10;
const ll mod = 1000000000 + 7;

```

```

ll quick(ll a, ll n) {
    ll ans = 1;
    for (; n >>= 1, a = a * a % mod)
        if (n & 1) ans = ans * a % mod;
    return ans;
}
ll a[maxn][maxn], b[maxn][maxn];

void gauss(int n) {
    memset(b, 0, sizeof(b));
    for (int i = 1; i <= n; i++) b[i][i] = 1;
    ll det = 1;
    for (int i = 1; i <= n; i++) {
        int t;
        for (t = i; t <= n; t++) {
            if (a[t][i]) break;
        }
        if (t != i) det *= -1;
        for (int j = 1; j <= n; j++) {
            swap(a[i][j], a[t][j]);
            swap(b[i][j], b[t][j]);
        }
        det = (det * a[i][i] % mod + mod) % mod;
        ll inv = quick(a[i][i], mod - 2);
        for (int j = 1; j <= n; j++) {
            a[i][j] = inv * a[i][j] % mod;
            b[i][j] = inv * b[i][j] % mod;
        }
        for (int k = 1; k <= n; k++) {
            if (k == i) continue;
            ll tmp = a[k][i];
            for (int j = 1; j <= n; j++) {
                a[k][j] = (a[k][j] - a[i][j] * tmp % mod + mod) %
mod;
                b[k][j] = (b[k][j] - b[i][j] * tmp % mod + mod) %
mod;
            }
        }
        det = (det + mod) % mod;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {

```

```

        if ((i + j) & 1) cout << (mod - b[j][1] * det % mod) %
mod;

        else cout << b[j][1] * det % mod;
        if (j != n) cout << " ";
        else cout << "\n";
    }
}
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n;
    while (cin >> n) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cin >> a[i][j];
            }
        }
        for (int i = 1; i <= n; i++) a[1][i] = 1;
        gauss(n);
    }
    return 0;
}

```

24.斯特林数

1.第一类斯特林数

定理

第一类斯特林数 $S_1(n, m)$ 表示的是将 n 个不同元素构成 m 个圆排列的数目。

```

const int mod = 1e9 + 7; //取模
LL s[N][N];              //存放要求的第一类 Stirling 数
void init() {
    memset(s, 0, sizeof(s));
    s[1][1] = 1;
    for (int i = 2; i <= N - 1; i++) {
        for (int j = 1; j <= i; j++) {
            s[i][j] = s[i - 1][j - 1] + (i - 1) * s[i - 1][j];
            if (s[i][j] >= mod)
                s[i][j] %= mod;
        }
    }
}

```

```

    }
}
}

```

1.第二类斯特林数

定理

第二类斯特林数 $S_2(n,m)$ 表示的是把 n 个不同元素划分到 m 个集合的方案数。

```

const int mod = 1e9 + 7; //取模
LL s[N][N];              //存放要求的 Stirling 数
void init() {
    memset(s, 0, sizeof(s));
    s[1][1] = 1;
    for (int i = 2; i <= N - 1; i++) {
        for (int j = 1; j <= i; j++) {
            s[i][j] = s[i - 1][j - 1] + j * s[i - 1][j];
            if (s[i][j] >= mod)
                s[i][j] %= mod;
        }
    }
}

```

贝尔数

每个贝尔数都是"第二类Stirling数"的和

$$B_n = \sum_{k=1}^n S(n, k).$$

25.威尔逊定理和 Miller_Rabin

```

long long gcd(long long a, long long b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

long long mul(long long a, long long b, long long mod) {
    long long ret = 0;

```

```

    while(b) {
        if(b & 1) ret=(ret+a)%mod;
        a=(a+a)%mod;
        b >>= 1;
    }
    return ret;
}

long long pow(long long a,long long b,long long mod) {
    long long ret = 1;
    while(b) {
        if(b & 1) ret = mul(ret,a,mod);
        a = mul(a,a,mod);
        b >>= 1;
    }
    return ret;
}

bool check(long long a,long long n){
    long long x = n - 1;
    int t = 0;
    while((x & 1) == 0) {
        x >>= 1;
        t ++;
    }
    x = pow(a,x,n);
    long long y;
    for(int i=1;i<=t;i++) {
        y = mul(x,x,n);
        if(y == 1 && x != 1 && x != n - 1) return true;
        x = y;
    }
    if(y != 1) return true;
    return false;
}

bool Miller_Rabin(long long n) {
    if(n == 2) return true;
    if(n == 1 || !(n & 1)) return false;
    const int arr[12] = {2,3,5,7,11,13,17,19,23,29,31,37};
    for(int i = 0; i < 12; i++) {
        if (arr[i] >= n) break;
        if(check(arr[i], n)) return false;
    }
    return true;
}

```

```

int main() {
    int t; long long n; scanf("%d",&t);
    while(t--) {
        scanf("%lld",&n); long long p=n-1;
        while(!Miller_Rabin(p)) p--;
        long long ans=1;
        for(long long i=p+1; i+1<n; i++) ans=mul(ans,pow(i,n-2,n),n);
        printf("%lld\n",ans);
    }
}

```

26.卡特兰数

原理:

令 $h(0)=1, h(1)=1$, catalan 数满足递归式:

$$h(n) = h(0) * h(n-1) + h(1) * h(n-2) + \dots + h(n-1)h(0) \quad (\text{其中 } n \geq 2)$$

另类递推公式:

$$h(n) = h(n-1) * (4 * n - 2) / (n + 1)$$

该递推关系的解为:

$$h(n) = C(2n, n) / (n + 1) \quad (n = 1, 2, 3, \dots)$$

卡特兰数的应用实质上都是递归等式的应用

前几项为: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

应用:

27.RSA 算法

【问题描述】

RSA 是一种经典的加密算法。它的基本加密过程如下。

首先生成两个质数 p, q , 令 $n = p \cdot q$, 设 d 与 $(p-1) \cdot (q-1)$ 互质, 则可找到 e 使得 $d \cdot e$ 除 $(p-1) \cdot (q-1)$ 的余数为 1。

n, d, e 组成了私钥, n, d 组成了公钥。

当使用公钥加密一个整数 X 时 (小于 n), 计算 $C = X^d \bmod n$, 则 C 是加密后的密文。

当收到密文 C 时, 可使用私钥解开, 计算公式为 $X = C^e \bmod n$ 。

例如, 当 $p = 5, q = 11, d = 3$ 时, $n = 55, e = 27$ 。

若加密数字 24, 得 $24^3 \bmod 55 = 19$ 。

解密数字 19, 得 $19^{27} \bmod 55 = 24$ 。

现在你知道公钥中 $n = 1001733993063167141, d = 212353$, 同时你截获了别人发送的密文 $C = 20190324$, 请问, 原文是多少?

题解

RSA解密, 令 $e = 2^{30} + 3$ 。令 $r = (p-1)(q-1)$

$$d * e \bmod r \equiv 1$$

$$c = f^e \bmod n$$

$$f = c^d \bmod n$$

可以暴力在 \sqrt{n} 附近寻找 p, q , 因为两个素数差距不会很大。

已知 e 和 r , 可以解同余方程

$$ex + ry = 1$$

得出 d 的值。

即可以算出 f 的值。

此题需要用 $O(1)$ 快速乘才能通过。

28. 高次剩余

【51nod 1038】 $X^A \bmod P$

题目描述

$X^A \bmod P = B$, 其中 P 为质数。给出 P 和 A, B , 求 $< P$ 的所有 X 。
例如: $P = 11, A = 3, B = 5$ 。
 $3^3 \bmod 11 = 5$
所有数据中, 解的数量不超过 $\text{Sqrt}(P)$ 。

分析

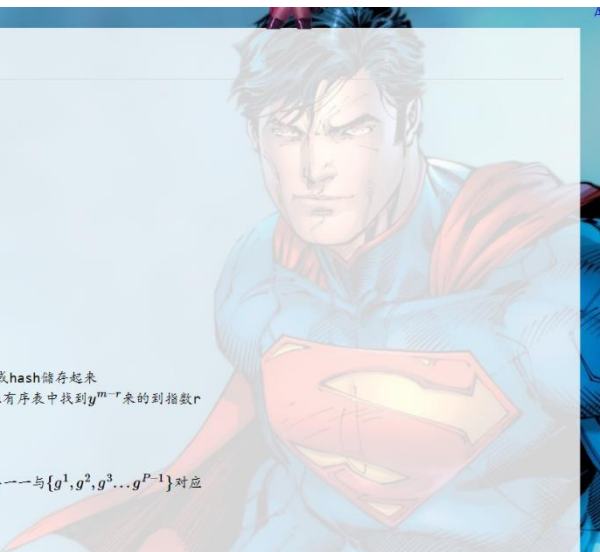
这道题包括几个知识点

离散对数 (大步小步BSGS算法)

求关于 x 的同余方程 $y^x \equiv n \pmod{P}$ (P 为质数) 的解,
设 $m = \lceil \sqrt{n} \rceil, x = bm + r$, 我们预处理出 $y^i (i \in [0, P-1])$, 用map或hash储存起来
我们从小到大枚举 b , 就可以根据 $y^{(b+1)m} \cdot n^{-1} \equiv y^{m-r} \pmod{P}$, 在有序表中找到 y^{m-r} 来的到指数 r

N次剩余

求关于 x 的同余方程 $x^n \equiv n \pmod{P}$ (P 为质数) 的解
设 P 的原根为 g , 因为 $\varphi(P) = P-1$, 根据原根的性质 $\{1, 2, 3 \dots P-1\}$ 一一与 $\{g^1, g^2, g^3 \dots g^{P-1}\}$ 对应
令 $x = g^a, n = g^b$
通过BSGS, 我们可以求出 t ,
于是 $s \cdot y \equiv t \pmod{P-1}$, 用扩展欧几里得解方程。



动态规划

1. 数位 dp

```
//数位 dp 不含 49 的数。
ll dp[100][2];
int a[100];
//len:当前统计到第几位
//if4:上一位是否是 4
//limit:当前的数是否有上限
ll dfs(int len, bool if4, bool limit)
{
    if(len==0) return 1;
    if(!limit&&dp[len][if4]) return dp[len][if4]; //数位 dp
    //的本质就是记忆化减少重复计算。
    //只有没有限制时才能返回, 如果有限制, 不同的限制的结果也会不一样。
    int maxi=limit?a[len]:9;
    ll cnt=0;
    for(int i=0; i<=maxi; i++)
    {
        if(if4&&i==9) continue;
        if(i==4) cnt+=dfs(len-1, 1, limit&&i==maxi);
        else cnt+=dfs(len-1, 0, limit&&i==maxi);
    }
    //只有之前有限制现在的达到了上限才能构成限制
```


//什么是上限，比如数字 756，当统计到第一位且 $i=7$ 时，那么此时 dfs 统计下一位就有了上限，因为第二位最多只能到 5，因为再多就超过了 756。

//再比如当统计到第一位且 $i=6$ 时，那么此时 dfs 统计下一位就没有上限，因为第二位任意取总体都不会超过 699。

```
    }
    if(!limit) //如果有限制，那么就不能记忆化
        dp[len][if4]=cnt;
    return cnt;
}
ll solve(ll n)
{
    int k=0;while(n)
    {
        a[++k]=n%10;n/=10;
    }
    return dfs(k,0,1);
}
int main()
{
    ll l,r;cin>>l>>r;
    cout<<solve(r)-solve(l-1)<<endl;
    return 0;
}
```

2. 斜率优化 dp

```
//  $dp[i] = \min\{dp[j] + a[i]^2 + a[j]^2 - 2*a[i]*a[j] + m\}$  求最小的
#define ll long long
const int N = 5e5 + 5;

ll a[N], dp[N], x[N];
int n, m;

ll getDp(int i, int j) {
    return dp[j] + (x[i] - x[j]) * (x[i] - x[j]) + m;
}

ll getUp(int i, int j) {
    return (dp[i] + x[i] * x[i]) - (dp[j] + x[j] * x[j]);
}

ll getDown(int i, int j) {
    return x[i] - x[j];
}
```

```

}

int q[N], tail, head;

int main()
{
    while(~scanf("%d %d", &n, &m)) {
        for(int i = 1; i <= n; i++) {
            scanf("%lld", &a[i]);
            x[i] = x[i - 1] + a[i];
        }
        head = tail = 0;
        q[tail++] = 0;
        for(int i = 1; i <= n; i++) {
            while(head + 1 < tail && getUp(q[head + 1], q[head]) <= 2LL
* x[i] * getDown(q[head + 1], q[head]))
                head++;
            dp[i] = getDp(i, q[head]);
            while(head + 1 < tail && getUp(q[tail-1], q[tail-2]) *
getDown(i, q[tail-1]) >= getUp(i, q[tail-1]) * getDown(q[tail-1],
q[tail-2]))
                tail--;
            q[tail++] = i;
        }
        cout << dp[n] << endl;
    }
    return 0;
}

/* 求 min
j > k
说明 1.如果  $g[j, k] < 2*sum[i]$  表示对于  $dp[i]$ , 从  $j$  转移过来比  $k$  更优, 反之  $k$ 
更优
说明 2.下面我们来考虑着怎么从解集去掉多余的元素, 可以证明可能存在某些元素, 无
论怎样都不会是最优的, 可以去掉这些多余的元素
假设  $k < j < i$ 
结论: 如果  $g[i, j] < g[j, k]$ , 那么  $j$  可以去掉
证明: 对于某个  $i$ , 如果  $g[i, j] < 2*sum[i]$ , 那么  $i$  比  $j$  更优, 结论成立;
如果  $g[i, j] >= 2*sum[i]$ , 那么  $g[j, k] > g[i, j] >= 2*sum[i]$ , 那么  $k$  比  $j$ 
更优, 结论成立.
所以如果把所有  $g[i, j] < g[j, k]$  的情况中(后面斜率比前面斜率小的情况)的  $j$  都
去掉, 那么我们就得到相邻两个元素的斜率递增的状况

*/

```

```

二维 斜率 dp[i][k] = min{ dp[j][k-1] + ..}

for(int i = 2; i <= m; i++) solve(i);

void solve(int k) {
    l = r = 0;
    q[r++] = 0;
    __int128 flag=1; // 防止超 long long
    for(int i = 1; i <= n; i++) {
        while(l + 1 < r && flag*up(q[l], q[l+1], k) >= flag*len[i] *
down(q[l], q[l+1])) l++;

        dp[i][k] = Dp(i, q[l], k);
        while(l + 1 < r && flag*up(q[r-2], q[r-1], k) * down(q[r-1],
i) >=
            flag*up(q[r-1],i, k) * down(q[r-2], q[r-1]) ) r--;
        q[r++] = i;
    }
}

```

斜率优化DP

有一类DP问题有特殊的性质，转移方程的形式如下：

$$f_i = \max_{j < i} (a_i * b_j + c_i + d_j)$$

其中， f 是DP数组， a 和 c 是在开始求解前就能确定的常数， b 和 d 是算出 f 后就能确定的常数。



只看这个式子不好判断最优转移所具有的性质。考虑两个转移 j 和 k ，假如 j 不比 k 更差，就有 $a_i * b_j + c_i + d_j \geq a_i * b_k + c_i + d_k$ 成立。

为了整理方便，我们暂且假设 $b_j < b_k$ 。于是得到

$$-a_i \geq \frac{d_k - d_j}{b_k - b_j}$$

我们发现，不等式的右侧实际上是一个斜率的形式，斜率优化由此得名。

说明1.如果 $g[j, k] < 2 * sum[i]$ 表示对于 $dp[i]$, 从j转移过来比k更优, 反之k更优

说明2.下面我们来考虑着怎么从解集去掉多余的元素, 可以证明可能存在某些元素, 无论怎样都不会是最优的, 可以去掉这些多余的元素

假设 $k < j < i$

结论:如果 $g[i, j] < g[j, k]$, 那么j可以去掉

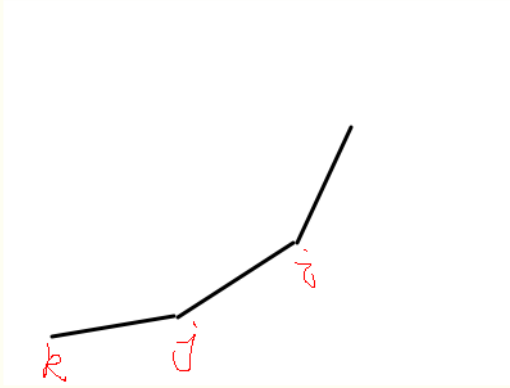
证明:对于某个i, 如果 $g[i, j] < 2 * sum[i]$, 那么i比j更优, 结论成立;

如果 $g[i, j] \geq 2 * sum[i]$, 那么 $g[j, k] > g[i, j] \geq 2 * sum[i]$, 那么k比j更优, 结论成立.

证毕.

所以如果把所有 $g[i, j] < g[j, k]$ 的情况中(后面斜率比前面斜率小的情况)的j都去掉, 那么我们就得到相邻两个元素的斜率递增的状况

如下图



这样, 从左到右, 斜率之间就是单调递增的了。当我们的最优解取得在j点的时候, 那么k点不可能再取得比j点更优的解了, 于是k点也可以排除。换句话说, j点之前的点全部不可能再比j点更优了, 可以全部从解集中排除。

于是对于这题我们对于斜率优化做法可以总结如下:

- 1, 用一个单调队列来维护解集。
- 2, 假设队列中从头到尾已经有元素 a b c。那么当d要入队的时候, 我们维护队列的上凸性质, 即如果 $g[d, c] < g[c, b]$, 那么就将c点删除。直到找到 $g[d, x] \geq g[x, y]$ 为止, 并将d点加入在该位置中。
- 3, 求解时候, 从队头开始, 如果已有元素 a b c, 当i点要求解时, 如果 $g[b, a] < sum[i]$, 那么说明b点比a点更优, a点可以排除, 于是a出队。最后 $dp[i] = getDp(q[head])$ 。

3. RMQ

```
void ST(int n) {
    for (int i = 1; i <= n; i++)
        dp[i][0] = A[i];
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int RMQ(int l, int r) {
    int k = log2(r - l + 1);
    return max(dp[l][k], dp[r - (1 << k) + 1][k]);
}
```

4. 斯坦纳树

```
const int limit = 1050;
const int INF = 1e9;
inline int read() {
    char c = getchar(); int x = 0, f = 1;
    while(c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}
    while(c >= '0' && c <= '9') {x = x * 10 + c - '0'; c = getchar();}
    return x * f;
}
#define MP(i,j) make_pair(i,j)
#define se second
#define fi first
#define Pair pair<int,int>
int N, M, tot = 0;
int a[12][12], f[12][12][limit];
int xx[5] = {-1, +1, 0, 0};
int yy[5] = {0, 0, -1, +1};
int vis[12][12];
struct PRE {
    int x, y, S;
}Pre[12][12][limit];
queue<Pair>q;
void SPFA(int cur) {
    while(q.size() != 0) {
        Pair p = q.front();q.pop();
        vis[p.fi][p.se] = 0;
        for(int i = 0; i <4; i++) {
            int wx = p.fi + xx[i], wy = p.se + yy[i];
            if(wx < 1 || wx > N || wy < 1 || wy > M) continue;
            if(f[wx][wy][cur] > f[p.fi][p.se][cur] + a[wx][wy]) {
                f[wx][wy][cur] = f[p.fi][p.se][cur] + a[wx][wy];
                Pre[wx][wy][cur] = (PRE){p.fi, p.se, cur};
                if(!vis[wx][wy])
                    vis[wx][wy] = 1, q.push(MP(wx,wy));
            }
        }
    }
}
void dfs(int x, int y, int now) {
    vis[x][y] = 1;
    PRE tmp = Pre[x][y][now];
    if(tmp.x == 0 && tmp.y == 0) return;
```

```

    dfs(tmp.x, tmp.y, tmp.S);
    if(tmp.x == x && tmp.y == y) dfs(tmp.x, tmp.y, now - tmp.S);
}
int main() {
    //freopen("a.in", "r", stdin);
    N = read(); M = read();
    memset(f, 0x3f, sizeof(f));
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= M; j++) {
            a[i][j] = read();
            if(a[i][j] == 0)
                f[i][j][1 << tot] = 0, tot++;
        }
    int limit = (1 << tot) - 1;
    for(int sta = 0; sta <= limit; sta++) {
        for(int i = 1; i <= N; i++)
            for(int j = 1; j <= M; j++) {
                for(int s = sta; s; s = (s - 1) & sta) {
                    if(f[i][j][s] + f[i][j][sta - s] - a[i][j] <
f[i][j][sta])
                        f[i][j][sta] = f[i][j][s] + f[i][j][sta - s] -
a[i][j],
                        Pre[i][j][sta] = (PRE){i,j,s};
                }
                if(f[i][j][sta] < INF) q.push(MP(i,j)), vis[i][j] = 1;
            }
        SPFA(sta);
    }
    int ansx, ansy, flag = 0;
    for(int i = 1; i <= N && !flag; i++)
        for(int j = 1; j <= M; j++)
            if(!a[i][j])
                {ansx = i, ansy = j; flag = 1; break;}
    printf("%d\n", f[ansx][ansy][limit]);
    memset(vis, 0, sizeof(vis));
    dfs(ansx, ansy, limit);
    for(int i = 1; i <= N; i++, puts("")) {
        for(int j = 1; j <= M; j++) {
            if(a[i][j] == 0) putchar('x');
            else if(vis[i][j]) putchar('o');
            else putchar('_');
        }
    }
    return 0;
}

```

```
}
```

计算几何

1. 最小覆盖圆

```
/*最小圆覆盖*/
/*给定 n 个点，让求半径最小的圆将 n 个点全部包围,可以在圆上*/
#define EPS 1e-8
const int maxn = 550;
struct point{
    double x, y;
};
int sgn(double x)
{
    if (fabs(x) < EPS)
        return 0;
    return x < 0 ? -1 : 1;
}
double get_distance(const point a, const point b)//两点之间的距离
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
point get_circle_center(const point a, const point b, const point c)//得到三角形外接圆的圆心
{
    point center;
    double a1 = b.x - a.x;
    double b1 = b.y - a.y;
    double c1 = (a1 * a1 + b1 * b1) / 2.0;
    double a2 = c.x - a.x;
    double b2 = c.y - a.y;
    double c2 = (a2 * a2 + b2 * b2) / 2.0;
    double d = a1 * b2 - a2 * b1;
    center.x = a.x + (c1 * b2 - c2 * b1) / d;
    center.y = a.y + (a1 * c2 - a2 * c1) / d;
    return center;
}
//p 表示定点，n 表示顶点的个数，c 代表最小覆盖圆圆心，r 是半径
```



```

void min_cover_circle(point *p, int n, point &c, double &r)//找最小覆盖圆
(这里没有用全局变量 p[], 因为是为了封装一个函数便于调用)
{
    random_shuffle(p, p + n);//随机函数,使用了之后使程序更快点,也可以不用
    c = p[0];
    r = 0;
    for (int i = 1; i < n; i++)
    {
        if (sgn(get_distance(p[i], c) - r) > 0)//如果 p[i]在当前圆的外面,
        那么以当前点为圆心开始找
        {
            c = p[i];//圆心为当前点
            r = 0;//这时候这个圆只包括他自己.所以半径为 0
            for (int j = 0; j < i; j++)//找它之前的所有点
            {
                if (sgn(get_distance(p[j], c) - r) > 0)//如果之前的点有不满足的,
                那么就是以这两点为直径的圆
                {
                    c.x = (p[i].x + p[j].x) / 2.0;
                    c.y = (p[i].y + p[j].y) / 2.0;
                    r = get_distance(p[j], c);
                    for (int k = 0; k < j; k++)
                    {
                        if (sgn(get_distance(p[k], c) - r) > 0)//找新作出来的
                        圆之前的点是否还有不满足的, 如果不满足一定就是三个点都在圆上了
                        {
                            c = get_circle_center(p[i], p[j], p[k]);
                            r = get_distance(p[i], c);
                        }
                    }
                }
            }
        }
    }
}

int main()
{
    int n;
    point p[maxn];
    point c; double r;
    while (~scanf("%d", &n) && n)
    {
        for (int i = 0; i < n; i++)
            scanf("%lf %lf", &p[i].x, &p[i].y);
    }
}

```

```

        min_cover_circle(p, n, c, r);
        printf("%.2lf %.2lf %.2lf\n", c.x, c.y, r);
    }
    return 0;
}

```

2. 两球相交体积

只能相交时候用（要判断相离和包含）

```

double dist(point p1, point p2) {          //返回平面上两点距离
    return sqrt((p1 - p2)*(p1 - p2));
}
typedef struct sphere { //球
    double r;
    point centre;
}sphere;
void SphereInterVS(sphere a, sphere b, double &v, double &s) {
    double d = dist(a.centre, b.centre); //球心距
    double t = (d*d + a.r*a.r - b.r*b.r) / (2.0 * d); //
    double h = sqrt((a.r*a.r) - (t*t)) * 2; //h1=h2, 球冠的高
    double angle_a = 2 * acos((a.r*a.r + d*d - b.r*b.r) / (2.0 *
a.r*d)); //余弦公式计算 r1 对应圆心角，弧度
    double angle_b = 2 * acos((b.r*b.r + d*d - a.r*a.r) / (2.0 *
b.r*d)); //余弦公式计算 r2 对应圆心角，弧度
    double l1 = ((a.r*a.r - 0.5*d*d - b.r*b.r) / d + d) / 2;
    double l2 = d - l1;
    double x1 = a.r - l1, x2 = b.r - l2; //分别为两个球缺的高度
    double v1 = PI*x1*x1*(a.r - x1 / 3); //相交部分 r1 圆所对应的球缺部分体积
    double v2 = PI*x2*x2*(b.r - x2 / 3); //相交部分 r2 圆所对应的球缺部分体积
    v = v1 + v2; //相交部分体积
    double s1 = PI*a.r*x1; //r1 对应球冠表面积
    double s2 = PI*b.r*x2; //r2 对应球冠表面积
    s = 4 * PI*(a.r*a.r + b.r*b.r) - s1 - s2; //剩余部分表面积
}

```

3. 最大半径圆覆盖

```

#define Mn 300
const double eps = 1e-9;

```

```

const double pi = acos(-1.0);
#define sqr(x) ((x) * (x))
struct point
{
    double x,y;
    void read()
    {
        scanf("%lf%lf",&x,&y);
    }
    void print()
    {
        printf("%lf%lf\n",x,y);
    }
    double friend dis(const point &a,const point &b)
    {
        return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
    }
} p[Mn + 5];
struct alpha
{
    double v;
    bool flag;
    bool friend operator <(const alpha &a,const alpha &b)
    {
        return a.v < b.v;
    }
} alp[Mn * Mn + 5];
int solve(int n,double R)//半径为 R 的圆最多能覆盖多少个点
{
    int MAX = 0;
    for(int i = 0; i < n; i++)
    {
        int t = 0;
        for(int j = 0; j < n; j++)
        {
            if(i == j)
                continue;
            double theta,phi,D;
            D = dis(p[i],p[j]);
            if(D > 2.0 * R)
                continue;
            theta = atan2(p[j].y - p[i].y,p[j].x - p[i].x);
            if(theta < 0)
                theta += 2 * pi;

```

```

        phi = acos(D / (2.0 * R));
        alp[t].v = theta - phi + 2 * pi;
        alp[t].flag = true;
        alp[t + 1].v = theta + phi + 2 * pi;
        alp[t + 1].flag = false;
        t += 2;
    }
    sort(alp, alp + t);
    int sum = 0;
    for(int j = 0; j < t; j++)
    {
        if(alp[j].flag)
            sum ++;
        else
            sum --;
        if(sum > MAX)
            MAX = sum;
    }
}
return MAX+1;
}

```

4. 计算圆和多边形面积交

```

const int MAX=1e6+10;
const int MOD=998244353;
const double PI=acos(-1.0);
typedef long long ll;
struct Point
{
    double x,y;
    Point(double x=0,double y=0):x(x),y(y){}
    void show(){cout<<"("<<x<<","<<y<<")"<<endl;}
}a[300],b[300];

//向量 A+B
Point operator+(Point A,Point B){return (Point){A.x+B.x,A.y+B.y};}
//向量 A-B
Point operator-(Point A,Point B){return (Point){A.x-B.x,A.y-B.y};}
//向量 A*B
Point operator*(Point A,double B){return (Point){A.x*B,A.y*B};}
//向量 A/B
Point operator/(Point A,double B){return (Point){A.x/B,A.y/B};}

```

```

//向量 A B 的点积
double operator*(Point A,Point B){return A.x*B.x+A.y*B.y;}
//向量 A B 的叉积
double operator^(Point A,Point B){return A.x*B.y-A.y*B.x;}
//向量 A B 的叉积
double cross(Point A,Point B){return A.x*B.y-A.y*B.x;}

//A B 两点的距离
Double dis(Point A,Point B)
{
return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
}

//A B 两点的距离的平方

double dis2(Point A,Point B)
{
return (A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y);
}
double Polyarea(Point *p,int n)    //求多边形面积(任意形状都可)
{
double area=0;
for(int i=1;i<n-1;i++)area+=(p[i]-p[0])^(p[i+1]-p[0]);
return fabs(area/2);
}
//点 P 到直线 AB 距离

double Distoline(Point P,Point A,Point B)
{
return fabs(cross(B-A,P-A))/dis(A,B);
}

//求点 P 在直线 AB 上的投影点

Point pverti(Point P,Point A,Point B)    {
double AA=A.y-B.y;
double BB=-(A.x-B.x);
double CC=A.y*(A.x-B.x)-A.x*(A.y-B.y);

Point PP =(Point) { P.x-2*AA*(AA*P.x+BB*P.y+CC)/(AA*AA+BB*BB) ,
P.y-2*BB*(AA*P.x+BB*P.y+CC)/(AA*AA+BB*BB)};
return (PP+P)/2;
}

```

```

double slove(Point P,Point A,Point B,double R)
{
double h=Distoline(P,A,B);
//A B 两点均在圆内
if(dis2(P,A)<=R*R&&dis2(P,B)<=R*R) return fabs(cross(A-P,B-P))/2;
if(dis2(P,A)>R*R)swap(A,B);
    //A B 两点有一点在圆内
    if(dis2(P,A)<=R*R&&dis2(P,B)>R*R) {
        double angle=(dis2(P,A)+dis2(A,B)-
dis2(P,B))/(2*dis(P,A)*dis(A,B));
        angle=acos(angle);
        double C=angle+asin(sin(angle)*dis(P,A)/R);
        double len=sin(C)*R/sin(angle);
        angle=(dis2(P,A)+dis2(P,B)-dis2(A,B))/(2*dis(P,A)*dis(P,B));
        angle=acos(angle)-(PI-C);
        return len*h/2+angle*R*R/2;
    }

//A B 两点均在圆外

double angle=(dis2(P,A)+dis2(P,B)-dis2(A,B))/
(2*dis(P,A)*dis(P,B));
double area=acos(angle)*R*R/2;
Point PP=pverti(P,A,B);
    //线段 AB 与圆不相交

if(h>=R||cross(PP-P,A-P)*cross(PP-P,B-P)>0)
    return area;
//线段 AB 与圆相交
return area-(2*acos(h/R)*R*R/2-h*sqrt(R*R-h*h));
}
//相交求面积
double cal(int n,Point p,double r)
{
    double tot=0;
    for(int i=0;i<n;i++)
    {
        if(cross(a[i]-p,a[(i+1)%n]-p)==0)continue;
        double area=slove(p,a[i],a[(i+1)%n],r);
        if(cross(a[i]-p,a[(i+1)%n]-p)>0)tot+=area;
        else tot-=area;
    }
    return fabs(tot);
}

```

```

}
int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++)scanf("%lf%lf",&a[i].x,&a[i].y);
    double sum=Polyarea(a,n);
    int m;
    cin>>m;
    for(int i=1;i<=m;i++)
    {
        int x,y,p,q;
        scanf("%d%d%d%d",&x,&y,&p,&q);
        p=q-p;
        double l=0,r=1e9;
//二分 这是另一个题
        for(int i=1;i<=100;i++)
        {
            double mid=(l+r)/2;
            if(cal(n,(Point){x*1.0,y*1.0},mid)<=sum*p/q)l=mid;
            else r=mid;
        }
        printf("%.12lf\n",(l+r)/2);
    }
    return 0;
}

```

5. 三维凸包的模板

```

//三维凸包的模板
/*给出三维空间中的 n 个顶点,求解由这 n 个顶点构成的凸包表面的多边形个数.
增量法求解:首先任选 4 个点形成的一个四面体,然后每次新加一个点,分两种情况:
    1> 在凸包内,则可以跳过
    2> 在凸包外,找到从这个点可以"看见"的面,删除这些面,
然后对于一边没有面的线段,和新加的这个点新建一个面,至于这个点可以看见的面,
就是求出这个面的方程(可以直接求法向量).*/
const int MAXN = 505;
const double EPS = 1e-8;
struct Point {
    double x, y, z;
    Point() {}

```

```

Point(double xx, double yy, double zz) : x(xx), y(yy), z(zz) {}

Point operator-(const Point p1) { //两向量之差
    return Point(x - p1.x, y - p1.y, z - p1.z);
}

Point operator*(Point p) { //叉乘
    return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y *
p.x);
}

double operator^(Point p) //点乘
{
    return (x * p.x + y * p.y + z * p.z);
}
};
struct CH3D {
    struct face {
        int a, b, c; //表示凸包一个面上三个点的编号
        bool ok;      //表示该面是否属于最终凸包中的面
    };

    int n;          //初始顶点数
    Point P[MAXN]; //初始顶点

    int num; //凸包表面的三角形数
    face F[8 * MAXN];

    int g[MAXN][MAXN]; //凸包表面的三角形

    double vlen(Point a) //向量长度
    {
        return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
    }

    Point cross(const Point &a, const Point &b, const Point &c) //叉乘
    {
        return Point((b.y - a.y) * (c.z - a.z) - (b.z - a.z) * (c.y -
a.y),
                    -((b.x - a.x) * (c.z - a.z) - (b.z - a.z) * (c.x -
a.x)),
                    (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x -
a.x));
    }
}

```



```

double area(Point a, Point b, Point c) //三角形面积*2
{
    return vlen((b - a) * (c - a));
}

double volume(Point a, Point b, Point c, Point d) //四面体有向体积*6
{
    return (b - a) * (c - a) ^ (d - a);
}

double dblcmp(Point &p, face &f) //正:点在面同向
{
    Point m = P[f.b] - P[f.a];
    Point n = P[f.c] - P[f.a];
    Point t = p - P[f.a];
    return (m * n) ^ t;
}

void deal(int p, int a, int b) {
    int f = g[a][b];
    face add;
    if (F[f].ok) {
        if (dblcmp(P[p], F[f]) > EPS)
            dfs(p, f);
        else {
            add.a = b;
            add.b = a;
            add.c = p;
            add.ok = 1;
            g[p][b] = g[a][p] = g[b][a] = num;
            F[num++] = add;
        }
    }
}

void dfs(int p, int now) {
    F[now].ok = 0;
    deal(p, F[now].b, F[now].a);
    deal(p, F[now].c, F[now].b);
    deal(p, F[now].a, F[now].c);
}

bool same(int s, int t) {
    Point &a = P[F[s].a];

```

```

        Point &b = P[F[s].b];
        Point &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < EPS && fabs(volume(a,
b, c, P[F[t].b])) < EPS &&
            fabs(volume(a, b, c, P[F[t].c])) < EPS;
    }

void solve() //构建三维凸包
{
    int i, j, tmp;
    face add;
    bool flag = true;
    num = 0;
    if (n < 4)
        return;
    for (i = 1; i < n; i++) //此段是为了保证前四个点不共面,若以保证,则
可去掉
    {
        if (vlen(P[0] - P[i]) > EPS) {
            swap(P[1], P[i]);
            flag = false;
            break;
        }
    }
    if (flag)
        return;
    flag = true;
    for (i = 2; i < n; i++) //使前三点不共线
    {
        if (vlen((P[0] - P[1]) * (P[1] - P[i])) > EPS) {
            swap(P[2], P[i]);
            flag = false;
            break;
        }
    }
    if (flag)
        return;
    flag = true;
    for (i = 3; i < n; i++) //使前四点不共面
    {
        if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) >
EPS) {
            swap(P[3], P[i]);
            flag = false;

```

```

        break;
    }
}
if (flag)
    return;
for (i = 0; i < 4; i++) {
    add.a = (i + 1) % 4;
    add.b = (i + 2) % 4;
    add.c = (i + 3) % 4;
    add.ok = true;
    if (dblcmp(P[i], add) > 0)
        swap(add.b, add.c);
    g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
    F[num++] = add;
}
for (i = 4; i < n; i++) {
    for (j = 0; j < num; j++) {
        if (F[j].ok && dblcmp(P[i], F[j]) > EPS) {
            dfs(i, j);
            break;
        }
    }
}
tmp = num;
for (i = num = 0; i < tmp; i++)
    if (F[i].ok) {
        F[num++] = F[i];
    }
}

double area() //表面积
{
    double res = 0.0;
    if (n == 3) {
        Point p = cross(P[0], P[1], P[2]);
        res = vlen(p) / 2.0;
        return res;
    }
    for (int i = 0; i < num; i++) res += area(P[F[i].a], P[F[i].b],
P[F[i].c]);
    return res / 2.0;
}

double volume() //体积

```

```

{
    double res = 0.0;
    Point tmp(0, 0, 0);
    for (int i = 0; i < num; i++) res += volume(tmp, P[F[i].a],
P[F[i].b], P[F[i].c]);
    return fabs(res / 6.0);
}

int triangle() //表面三角形个数
{
    return num;
}

int polygon() //表面多边形个数
{
    int i, j, res, flag;
    for (i = res = 0; i < num; i++) {
        flag = 1;
        for (j = 0; j < i; j++)
            if (same(i, j)) {
                flag = 0;
                break;
            }
        res += flag;
    }
    return res;
}

Point getcent() //求凸包质心
{
    Point ans(0, 0, 0), temp = P[F[0].a];
    double v = 0.0, t2;
    for (int i = 0; i < num; i++) {
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
            t2 = volume(temp, p1, p2, p3) / 6.0; //体积大于 0, 也就是
说, 点 temp 不在这个面上
            if (t2 > 0) {
                ans.x += (p1.x + p2.x + p3.x + temp.x) * t2;
                ans.y += (p1.y + p2.y + p3.y + temp.y) * t2;
                ans.z += (p1.z + p2.z + p3.z + temp.z) * t2;
                v += t2;
            }
        }
    }
}

```

```

        ans.x /= (4 * v);
        ans.y /= (4 * v);
        ans.z /= (4 * v);
        return ans;
    }
    double function(Point fuck) //点到凸包上的最近距离（枚举每个面到这个点的距离）
    {
        double min = 999999999;
        for (int i = 0; i < num; i++) {
            if (F[i].ok == true) {
                Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
                double a = ((p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z)
* (p3.y - p1.y));
                double b = ((p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1.x)
* (p3.z - p1.z));
                double c = ((p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y)
* (p3.x - p1.x));
                double d = (0 - (a * p1.x + b * p1.y + c * p1.z));
                double temp = fabs(a * fuck.x + b * fuck.y + c * fuck.z +
d) / sqrt(a * a + b * b + c * c);
                if (temp < min)
                    min = temp;
            }
        }
        return min;
    }
};
CH3D hull;
}

```

其他

1. 读入优化

```

//普通读入优化
void read(int &x)
{
    int f=1;x=0;char s=getchar();

```

```

while(s<'0' || s>'9'){if(s=='-')f=-1;s=getchar();}
while(s>='0'&&s<='9'){x=x*10+s-'0';s=getchar();}
x*=f;
}

```

```

inline bool read(int &num) //整数输入
{
    char in;
    in=getchar();

    if(in==EOF)
        return false;
    while(in<'0' || in>'9')
        in=getchar();

    num=in-'0';
    while(in=getchar(),in>='0'&&in<='9') //处理大于等于两位数情况
    {
        num*=10,num+=in-'0';
    }

    return true;
}

```

```

//需要自己写文件读入 scanf 和 getchar()都不能用。
//freopen("1.txt","r",stdin);

```

```

namespace io {
    const int SIZE = 1e7 + 10;
    char inbuff[SIZE];
    char *l, *r;
    inline void init() {
        l = inbuff;
        r = inbuff + fread(inbuff, 1, SIZE, stdin);
    }
    inline char gc() {
        if (l == r) init();
        return (l != r) ? *(l++) : EOF;
    }
    template<typename T>

```

```

void read(T &x) {
    x = 0; char ch = gc();
    while(!isdigit(ch)) ch = gc();
    while(isdigit(ch)) x = x * 10 + ch - '0', ch = gc();
}
template<typename T>
void write(T x)
{
    if(x<0) putchar('-'),x=-x;
    if(x>9) write(x/10);
    putchar(x%10+'0');
}
};
using namespace io;

```

```

namespace fastIO{
    #define BUF_SIZE 100000
    #define OUT_SIZE 100000
    #define ll long long
    //fread->read
    bool IOerror=0;
    inline char nc(){
        static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
        if (p1==pend){
            p1=buf; pend=buf+fread(buf,1,BUF_SIZE,stdin);
            if (pend==p1){IOerror=1;return -1;}
            //{printf("IO error!\n");system("pause");for (;;);exit(0);}
        }
        return *p1++;
    }
    inline bool blank(char ch){return ch=='
' || ch=='\n' || ch=='\r' || ch=='\t';}
    inline void read(int &x){
        bool sign=0; char ch=nc(); x=0;
        for (;blank(ch);ch=nc());
        if (IOerror)return;
        if (ch=='-')sign=1,ch=nc();
        for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
        if (sign)x=-x;
    }
    inline void read(ll &x){
        bool sign=0; char ch=nc(); x=0;
        for (;blank(ch);ch=nc());
        if (IOerror)return;
    }
}

```

```

        if (ch=='-')sign=1,ch=nc();
        for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
        if (sign)x=-x;
    }
    inline void read(double &x){
        bool sign=0; char ch=nc(); x=0;
        for (;blank(ch);ch=nc());
        if (IOerror)return;
        if (ch=='-')sign=1,ch=nc();
        for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
        if (ch=='.'){
            double tmp=1; ch=nc();
            for (;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp*(ch-'0');
        }
        if (sign)x=-x;
    }
    inline void read(char *s){
        char ch=nc();
        for (;blank(ch);ch=nc());
        if (IOerror)return;
        for (;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
        *s=0;
    }
    inline void read(char &c){
        for (c=nc();blank(c);c=nc());
        if (IOerror){c=-1;return;}
    }
    //getchar->read
    inline void read1(int &x){
        char ch;int bo=0;x=0;
        for (ch=getchar();ch<'0' || ch>'9';ch=getchar())if (ch=='-')bo=1;
        for (;ch>='0'&&ch<='9';x=x*10+ch-'0',ch=getchar());
        if (bo)x=-x;
    }
    inline void read1(ll &x){
        char ch;int bo=0;x=0;
        for (ch=getchar();ch<'0' || ch>'9';ch=getchar())if (ch=='-')bo=1;
        for (;ch>='0'&&ch<='9';x=x*10+ch-'0',ch=getchar());
        if (bo)x=-x;
    }
    inline void read1(double &x){
        char ch;int bo=0;x=0;
        for (ch=getchar();ch<'0' || ch>'9';ch=getchar())if (ch=='-')bo=1;
        for (;ch>='0'&&ch<='9';x=x*10+ch-'0',ch=getchar());
    }

```



```

        if (ch=='.'){
            double tmp=1;
            for (ch=getchar();ch>='0'&&ch<='9';tmp/=10.0,x+=tmp*(ch-
'0'),ch=getchar());
        }
        if (bo)x=-x;
    }
    inline void read1(char *s){
        char ch=getchar();
        for (;blank(ch);ch=getchar());
        for (;!blank(ch);ch=getchar())*s++=ch;
        *s=0;
    }
    inline void read1(char &c){for (c=getchar();blank(c);c=getchar());}
    //scanf->read
    inline void read2(int &x){scanf("%d",&x);}
    inline void read2(ll &x){
        #ifdef _WIN32
            scanf("%I64d",&x);
        #else
            #ifdef __linux
                scanf("%lld",&x);
            #else
                puts("error:can't recognize the system!");
            #endif
        #endif
    }
    inline void read2(double &x){scanf("%lf",&x);}
    inline void read2(char *s){scanf("%s",s);}
    inline void read2(char &c){scanf(" %c",&c);}
    // inline void readln2(char *s){gets(s);}
    //fwrite->write
    struct Ostream_fwrite{
        char *buf,*p1,*pend;
        Ostream_fwrite(){buf=new
char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
        void out(char ch){
            if (p1==pend){
                fwrite(buf,1,BUF_SIZE,stdout);p1=buf;
            }
            *p1++=ch;
        }
        void print(int x){
            static char s[15],*s1;s1=s;

```

```

        if (!x)*s1++='0';if (x<0)out('-'),x=-x;
        while(x)*s1++=x%10+'0',x/=10;
        while(s1--!=s)out(*s1);
    }
    void println(int x){
        static char s[15],*s1;s1=s;
        if (!x)*s1++='0';if (x<0)out('-'),x=-x;
        while(x)*s1++=x%10+'0',x/=10;
        while(s1--!=s)out(*s1); out('\n');
    }
    void print(ll x){
        static char s[25],*s1;s1=s;
        if (!x)*s1++='0';if (x<0)out('-'),x=-x;
        while(x)*s1++=x%10+'0',x/=10;
        while(s1--!=s)out(*s1);
    }
    void println(ll x){
        static char s[25],*s1;s1=s;
        if (!x)*s1++='0';if (x<0)out('-'),x=-x;
        while(x)*s1++=x%10+'0',x/=10;
        while(s1--!=s)out(*s1); out('\n');
    }
    void print(double x,int y){
        static ll
mul[]={1,10,100,1000,10000,100000,1000000,10000000,100000000,
1000000000,10000000000LL,100000000000LL,1000000000000LL,10000000000000LL,
L,
100000000000000LL,1000000000000000LL,10000000000000000LL,100000000000000000LL};
        if (x<-1e-12)out('-'),x=-x;x*=mul[y];
        ll x1=(ll)floor(x); if (x-floor(x)>=0.5)++x1;
        ll x2=x1/mul[y],x3=x1-x2*mul[y]; print(x2);
        if (y>0){out('.'); for (size_t
i=1;i<y&& x3*mul[i]<mul[y];out('0'),++i); print(x3);}
    }
    void println(double x,int y){print(x,y);out('\n');}
    void print(char *s){while (*s)out(*s++);}
    void println(char *s){while (*s)out(*s++);out('\n');}
    void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
    ~Ostream_{fwrite(){flush();}}
}Ostream;
inline void print(int x){Ostream.print(x);}

```

```

inline void println(int x){Ostream.println(x);}
inline void print(char x){Ostream.out(x);}
inline void println(char x){Ostream.out(x);Ostream.out('\n');}
inline void print(ll x){Ostream.print(x);}
inline void println(ll x){Ostream.println(x);}
inline void print(double x,int y){Ostream.print(x,y);}
inline void println(double x,int y){Ostream.println(x,y);}
inline void print(char *s){Ostream.print(s);}
inline void println(char *s){Ostream.println(s);}
inline void println(){Ostream.out('\n');}
inline void flush(){Ostream.flush();}
//puts->write
char Out[OUT_SIZE],*o=Out;
inline void print1(int x){
    static char buf[15];
    char *p1=buf;if (!x)*p1++='0';if (x<0)*o++='-',x=-x;
    while(x)*p1++=x%10+'0',x/=10;
    while(p1--!=buf)*o++=*p1;
}
inline void println1(int x){print1(x);*o++='\n';}
inline void print1(ll x){
    static char buf[25];
    char *p1=buf;if (!x)*p1++='0';if (x<0)*o++='-',x=-x;
    while(x)*p1++=x%10+'0',x/=10;
    while(p1--!=buf)*o++=*p1;
}
inline void println1(ll x){print1(x);*o++='\n';}
inline void print1(char c){*o++=c;}
inline void println1(char c){*o++=c;*o++='\n';}
inline void print1(char *s){while (*s)*o++=*s++;}
inline void println1(char *s){print1(s);*o++='\n';}
inline void println1(){*o++='\n';}
inline void flush1(){if (o!=Out){if (*(o-1)=='\n')*--
o=0;puts(Out);}}
struct puts_write{
    ~puts_write(){flush1();}
}_puts;
inline void print2(int x){printf("%d",x);}
inline void println2(int x){printf("%d\n",x);}
inline void print2(char x){printf("%c",x);}
inline void println2(char x){printf("%c\n",x);}
inline void print2(ll x){
    #ifdef _WIN32
        printf("%I64d",x);
    #endif
}

```

```

        #else
        #ifdef __linux
            printf("%lld",x);
        #else
            puts("error:can't recognize the system!");
        #endif
        #endif
    }
    inline void println2(ll x){print2(x);printf("\n");}
    inline void println2(){printf("\n");}
    #undef ll
    #undef OUT_SIZE
    #undef BUF_SIZE
};
using namespace fastIO;

```

2. 技巧优化

局部变量快一点。unordered_map<ll,int> 比 unordered_map<string, int> 快一点

3. 取模优化

```

inline ll mul(ll x, ll y)//mod long long
{
    if(y < 0) y += mod;
    ll tmp=(x*y-(ll)((long double)x/mod*y+0.5)*mod);
    return tmp<0 ? tmp+mod : tmp;
}

```

4. Java 快速读入

```

static class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream),
32768);
    }
}

```

```

        tokenizer = null;
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            try {
                tokenizer = new StringTokenizer(reader.readLine());
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }
}

```

Codeforces著名世界级选手Petr大爷写的Java输入内部类：

```

1  static class InputReader {
2      public BufferedReader reader;
3      public StringTokenizer tokenizer;
4
5      public InputReader(InputStream stream) {
6          reader = new BufferedReader(new InputStreamReader(stream), 32768);
7          tokenizer = null;
8      }
9
10     public String next() {
11         while (tokenizer == null || !tokenizer.hasMoreTokens()) {
12             try {
13                 tokenizer = new StringTokenizer(reader.readLine());
14             } catch (IOException e) {
15                 throw new RuntimeException(e);
16             }
17         }
18         return tokenizer.nextToken();
19     }
20
21     public int nextInt() {
22         return Integer.parseInt(next());
23     }
24
25 }

```

5. 二进制的高效位运算 __builtin_系列函数

•int __builtin_ffs (unsigned int x)

返回 x 的最后一位 1 的是从后向前第几位，比如 7368 (1110011001000) 返回 4。

•int __builtin_clz (unsigned int x)

返回前导的 0 的个数。

•int __builtin_ctz (unsigned int x)

返回后面的 0 的个数，和 __builtin_clz 相对。

•int __builtin_popcount (unsigned int x)

返回二进制表示中 1 的个数。

•int __builtin_parity (unsigned int x)

返回 x 的奇偶校验位，也就是 x 的 1 的个数模 2 的结果。

此外，这些函数都有相应的 unsigned long 和 unsigned long long 版本，只需要在函数名后面加上 l 或 ll 就可以了，比如 int __builtin_clzll。

6. 基姆拉尔森计算公式

$W = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400 + 1) \% 7$ //C++计算公式

在公式中 d 表示日期中的日数，m 表示月份数，y 表示年数。

注意：在公式中有个与其他公式不同的地方：

把一月和二月看成是上一年的十三月和十四月，例：如果是 2004-1-10 则换算成：2003-13-10 来代入公式计算。

7. 随机生成数

```
int ran(){
    static int seed=23333;
    return seed=(((((11)seed^20030927)%p+330802)%p*9410)%p-
19750115+p)%p^730903)%p;
} // p 是 mod

// 这个好一点
mt19937 rnd(time(0))
```

8. 二维 vector 的定义

```
typedef vector<int> VI;
typedef vector<VI > VVI;
typedef vector<VVI > WVVI;
int n,m,h,q;
inline int lowbit(int x){return x&(-x);}
struct BIT {
    int n, m, h;
    vector<vector<vector<int> > > bit;
    void init(int _n, int _m, int _h) {
        n = _n, m = _m, h = _h;
        bit = WVVI(n+1,VVI(m+1, VI(h+1, inf)));
    }
    void update(int _x, int _y, int _z, int v) {
        for(int x = _x; x <= n; x += lowbit(x)) {
            for(int y = _y; y <= m; y += lowbit(y)) {
                for(int z = _z; z <= h; z += lowbit(z)) {
                    bit[x][y][z] =min(bit[x][y][z],v);
                }
            }
        }
    }
    int query(int _x, int _y, int _z) {
        int res = inf;
        for(int x = _x; x > 0; x -= lowbit(x)) {
            for(int y = _y; y > 0; y -= lowbit(y)) {
                for(int z = _z; z > 0; z -= lowbit(z)) {
                    res=min(res,bit[x][y][z]);
                }
            }
        }
        return res;
    }
}bit[8];
```

9. 随机数

```
#define ll long long
int main()
{
    srand((unsigned )time(NULL));
```

```

// 生成 20 组数据
for (int test = 1; test <= 1; test++) {
    char name[100];
    sprintf(name, "%d.in", test);// 注意文件名称必须以 in 作为后缀
    FILE * fp = fopen(name, "w");
//    fprintf(fp, "1\n");// 输出到文件中
    int T = 1;
    while(T--) {
        int n = rand()%200+2;
        fprintf(fp, "%d\n", n);// 输出到文件中
        for(int i = 1; i <= n; i++) {
            fprintf(fp, "%d%c", rand()%n+1, i == n ? '\n' : ' ');//
输出到文件中
        }
//        fprintf(fp, "%d %d %d\n", a, b, c);// 输出到文件中

    }
    fclose(fp);
}
return 0;
}

```

10. 代码时间

```

auto start = std::chrono::system_clock::now();
std::chrono::duration<double, std::milli> duration
    = std::chrono::system_clock::now() - start;
std::cout << "time elapsed: " << duration.count() << "ms" <<
std::endl;

```