



ACM/ICPC Template

ZheJiang GongShang University

linbinwu

2020 年 12 月 7 日

目录

0 其他	1
0.1 BigInteger	1
0.2 bound	3
0.3 Cantor	4
0.4 Hash	5
0.5 RMQ	5
0.6 set	5
0.7 四维偏序	6
0.8 模拟退火	8
0.9 莫队	9
0.9.1 回滚莫队	9
0.9.2 带修莫队	11
0.9.3 普通莫队	12
0.9.4 树上莫队	13
1 动态规划	15
1.1 四边形优化	15
1.2 数位 DP	16
1.3 树上背包	16
1.4 环基树 DP	17
2 图论	18
2.1 BellmanFord	18
2.2 Dijkstra	18
2.3 Kruskal	18
2.4 Kruskal 重构树	19
2.5 Prim	20
2.6 SPFA	21
2.7 严格次小生成树	21
2.8 二分图判定	24
2.9 匹配问题	25
2.9.1 GaleShapley	25
2.9.2 Hungry	26
2.9.3 KM	26
2.9.4 一些结论	28
2.10 同余最短路	28
2.11 差分约束	30
2.12 拓扑排序	30
2.13 矩阵树	31
2.14 网络流	31
2.14.1 Dinic	31
2.14.2 ISAP	33
2.14.3 MCMF	34
2.15 连通分量	36
2.15.1 割点	36
2.15.2 桥	36
2.15.3 连通分量	36
3 字符串	37
3.1 KMP	37

4	数学	38
4.1	数论	38
4.1.1	BSGS	38
4.1.2	CRT	39
4.1.3	exBSGS	40
4.1.4	exCRT	41
4.1.5	exEuler	42
4.1.6	exGCD	43
4.1.7	exLucas	43
4.1.8	Lucas	44
4.1.9	二次剩余	45
4.1.10	反演相关	46
4.1.11	常见积性函数	47
4.1.12	整除分块	47
4.1.13	杜教筛	47
4.1.14	筛 mobius	48
4.1.15	筛 phi	48
4.1.16	筛积性函数	49
4.1.17	线性同余方程	49
4.1.18	质因子分解	50
4.1.19	逆元	50
4.1.20	龟速乘	51
4.2	线性代数	52
4.2.1	多项式	52
4.2.2	拉格朗日插值	71
4.2.3	矩阵	71
4.2.4	矩阵求逆	73
4.2.5	高斯消元	74
4.3	组合数学	74
4.3.1	Lucas	74
4.3.2	二项式相关	75
4.3.3	卡特兰数	75
4.3.4	容斥原理	75
4.3.5	第二类斯特林数	76
4.3.6	组合数	76
4.4	计算几何	77
4.4.1	凸包	77
5	数据结构	79
5.1	01Trie	79
5.2	BIT	79
5.3	HashTable	79
5.4	K-D_Tree	80
5.5	Scapegoat	82
5.6	Splay	84
5.7	Splay 区间翻转	87
5.8	Trie	89
5.9	主席树	89
5.9.1	动态主席树	89
5.9.2	区间不同数	91
5.9.3	区间第 K 小	92
5.9.4	树上建树	92
5.10	可持久化 01Trie	95
5.11	扫描线	96

5.12 线性基	97
5.13 线段树动态开点合并分裂	99
6 树和森林	100
6.1 LCT	100
6.2 带权并查集	102
6.3 按秩合并可撤回并查集	102
6.4 树上 K 祖先	103
6.5 树上启发式合并	104
6.6 树剖 LCA	105
6.7 树的直径	105
6.8 树的重心	107
6.9 树链剖分 (点权)	108
6.10 树链剖分 (边权)	109
6.11 点分树	112
6.12 点分治	113
6.13 虚树	114
6.14 轻重链划分	116

0 其他

0.1 BigInteger

```
1 struct BigInteger {
2     typedef unsigned long long LL;
3
4     static const int BASE = 100000000;
5     static const int WIDTH = 8;
6     vector<int> s;
7
8     BigInteger& clean() { while (!s.back() && s.size()>1)s.pop_back(); return *this; }
9     BigInteger(LL num = 0) { *this = num; }
10    BigInteger(string s) { *this = s; }
11    BigInteger& operator = (long long num) {
12        s.clear();
13        do {
14            s.push_back(num % BASE);
15            num /= BASE;
16        } while (num > 0);
17        return *this;
18    }
19    BigInteger& operator = (const string& str) {
20        s.clear();
21        int x, len = (str.length() - 1) / WIDTH + 1;
22        for (int i = 0; i < len; i++) {
23            int end = str.length() - i*WIDTH;
24            int start = max(0, end - WIDTH);
25            sscanf(str.substr(start, end - start).c_str(), "%d", &x);
26            s.push_back(x);
27        }
28        return (*this).clean();
29    }
30
31    BigInteger operator + (const BigInteger& b) const {
32        BigInteger c; c.s.clear();
33        for (int i = 0, g = 0; ; i++) {
34            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
35            int x = g;
36            if (i < s.size()) x += s[i];
37            if (i < b.s.size()) x += b.s[i];
38            c.s.push_back(x % BASE);
39            g = x / BASE;
40        }
41        return c;
42    }
43    BigInteger operator - (const BigInteger& b) const {
44        assert(b <= *this); // 减数不能大于被减数
45        BigInteger c; c.s.clear();
46        for (int i = 0, g = 0; ; i++) {
47            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
48            int x = s[i] + g;
49            if (i < b.s.size()) x -= b.s[i];
50            if (x < 0) { g = -1; x += BASE; }
```

```

51         else g = 0;
52         c.s.push_back(x);
53     }
54     return c.clean();
55 }
56 BigInteger operator * (const BigInteger& b) const {
57     int i, j; LL g;
58     vector<LL> v(s.size() + b.s.size(), 0);
59     BigInteger c; c.s.clear();
60     for (i = 0; i<s.size(); i++) for (j = 0; j<b.s.size(); j++) v[i + j] += LL(s[i])*b.
s[j];
61     for (i = 0, g = 0; ; i++) {
62         if (g == 0 && i >= v.size()) break;
63         LL x = v[i] + g;
64         c.s.push_back(x % BASE);
65         g = x / BASE;
66     }
67     return c.clean();
68 }
69 BigInteger operator / (const BigInteger& b) const {
70     assert(b > 0); // 除数必须大于0
71     BigInteger c = *this; // 商:主要是让c.s和(*this).s的vector一样大
72     BigInteger m; // 余数:初始化为0
73     for (int i = s.size() - 1; i >= 0; i--) {
74         m = m*BASE + s[i];
75         c.s[i] = bsearch(b, m);
76         m -= b*c.s[i];
77     }
78     return c.clean();
79 }
80 BigInteger operator % (const BigInteger& b) const { //方法与除法相同
81     BigInteger c = *this;
82     BigInteger m;
83     for (int i = s.size() - 1; i >= 0; i--) {
84         m = m*BASE + s[i];
85         c.s[i] = bsearch(b, m);
86         m -= b*c.s[i];
87     }
88     return m;
89 }
90 // 二分法找出满足bx<=m的最大的x
91 int bsearch(const BigInteger& b, const BigInteger& m) const {
92     int L = 0, R = BASE - 1, x;
93     while (1) {
94         x = (L + R) >> 1;
95         if (b*x <= m) { if (b*(x + 1)>m) return x; else L = x; }
96         else R = x;
97     }
98 }
99 BigInteger& operator += (const BigInteger& b) { *this = *this + b; return *this; }
100 BigInteger& operator -= (const BigInteger& b) { *this = *this - b; return *this; }
101 BigInteger& operator *= (const BigInteger& b) { *this = *this * b; return *this; }
102 BigInteger& operator /= (const BigInteger& b) { *this = *this / b; return *this; }
103 BigInteger& operator %= (const BigInteger& b) { *this = *this % b; return *this; }

```

```

104
105     bool operator < (const BigInteger& b) const {
106         if (s.size() != b.s.size()) return s.size() < b.s.size();
107         for (int i = s.size() - 1; i >= 0; i--)
108             if (s[i] != b.s[i]) return s[i] < b.s[i];
109         return false;
110     }
111     bool operator >(const BigInteger& b) const { return b < *this; }
112     bool operator<=(const BigInteger& b) const { return !(b < *this); }
113     bool operator>=(const BigInteger& b) const { return !(*this < b); }
114     bool operator!=(const BigInteger& b) const { return b < *this || *this < b; }
115     bool operator==(const BigInteger& b) const { return !(b < *this) && !(b > *this); }
116 };
117
118 ostream& operator << (ostream& out, const BigInteger& x) {
119     out << x.s.back();
120     for (int i = x.s.size() - 2; i >= 0; i--) {
121         char buf[20];
122         sprintf(buf, "%08d", x.s[i]);
123         for (int j = 0; j < strlen(buf); j++) out << buf[j];
124     }
125     return out;
126 }
127
128 istream& operator >> (istream& in, BigInteger& x) {
129     string s;
130     if (!(in >> s)) return in;
131     x = s;
132     return in;
133 }
134
135 int main()
136 {
137     int t;
138     scanf("%d", &t);
139     while (t--)
140     {
141         BigInteger a, b;
142         cin >> a >> b;
143         cout << a + b << endl;
144     }
145 }

```

0.2 bound

```

1
2 //get pos of first num >= x
3 int pos(int x) { return lower_bound(b + 1, b + 1 + N, x) - b; }
4
5 //get pos of first num > x
6 int pos(int x) { return upper_bound(b + 1, b + 1 + N, x) - b; }

```

0.3 Cantor

```
1 //主要应用为将N维的排列状态压缩成数字id
2 //然后需要知道具体状态时用逆Cantor得到
3
4 int N;
5 int a[MAX], c[MAX];
6
7 void upd(int p, int k) { for (; p <= N; p += lowbit(p)) c[p] += k; }
8 int query(int p) {
9     int res = 0;
10    for (; p; p -= lowbit(p)) res += c[p];
11    return res;
12 }
13
14 int cantor() {
15     //ans = 1 + \sum_{i=1}^N fac[N-i] * (\sum_{j=i+1}^N x[i] > x[j])
16     int res = 0, fac = 1;
17     for (int i = N; i >= 1; i--) {
18         upd(a[i], 1);
19         res = (res + 1ll * fac * query(a[i] - 1) % mod) % mod;
20         fac = 1ll * fac * (N - i + 1) % mod;
21     }
22     return res + 1;
23 }
24
25 //逆Cantor
26 #define lc u<<1
27 #define rc u<<1|1
28 #define mid (l+r)/2
29 int sum[MAX << 4];
30 void push_up(int u) { sum[u] = sum[lc] + sum[rc]; }
31 void build(int u, int l, int r) {
32     if (l == r) {
33         sum[u] = 1;
34         return;
35     }
36     build(lc, l, mid); build(rc, mid + 1, r);
37     push_up(u);
38 }
39 int query(int u, int l, int r, int k) { //查找第k大并且删除该数
40     sum[u]--;
41     if (l == r) return l;
42     if (k <= sum[lc]) return query(lc, l, mid, k);
43     else return query(rc, mid + 1, r, k - sum[lc]);
44 }
45
46 vector<int> inCantor(int x, int n) {
47     x--;
48     vector<int> res;
49     ll fac = 1;
50     build(1, 1, n);
51     for (int i = 1; i <= n; i++) fac = fac * i;
52     for (int i = 1; i <= n; i++) {
```



```

53         fac = fac / (n - i + 1);
54         int k = x / fac + 1; //比当前这位大的有  $x / fac$  位
55         res.push_back(query(1, 1, n, k)); //找到没被选的第  $k$  大
56         x %= fac;
57     }
58     return res;
59 }

```

0.4 Hash

```

1 struct Hash {
2     int b[N], tot;
3     void init() { tot = 0; }
4     void insert(int x) { b[++tot] = x; }
5     void build() {
6         sort(b + 1, b + 1 + tot);
7         tot = unique(b + 1, b + 1 + tot) - (b + 1);
8     }
9     int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
10 };

```

0.5 RMQ

```

1 const int MAX_BIT = 20;
2
3 int dp[N][MAX_BIT];
4
5 void build(int siz) {
6     for (int i = 1; i <= siz; i++) cin >> dp[i][0];
7     for (int j = 1; j < MAX_BIT; j++)
8         for (int i = 1; i + (1 << j) - 1 <= siz; i++)
9             dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
10 }
11
12 int query(int ql, int qr) {
13     int k = (int)log2(qr - ql + 1);
14     return min(dp[ql][k], dp[qr - (1 << k) + 1][k]);
15 }

```

0.6 set

```

1 set<int> st;
2 set<int>::iterator now;
3
4 int lower(int x) {
5     now = st.find(x);
6     if (now == st.begin()) return -1;
7     return *(--now);
8 }
9
10 int upper(int x) {
11     now = st.find(x); now++;

```

```

12     if (now != st.end()) return *now;
13     else return -1;
14 }

```

0.7 四维偏序

```

1 //四维偏序查最大值
2 int N;
3 struct node {
4     int a, b, c, d;
5     inline bool operator < (const node &rhs) const { //四维排序掉一维
6         return a < rhs.a || (a == rhs.a && (b < rhs.b || (b == rhs.b && (c < rhs.c || (c ==
7             rhs.c && d < rhs.d))));
8     }
9 } nod[MAX];
10
11 struct point {
12     int x, y, z, val;
13 } p[MAX];
14
15 inline point max(const point &a, const point &b) {
16     return point{max(a.x, b.x), max(a.y, b.y), max(a.z, b.z)};
17 }
18
19 inline point min(const point &a, const point &b) {
20     return point{min(a.x, b.x), min(a.y, b.y), min(a.z, b.z)};
21 }
22
23 int cmptype;
24
25 inline bool operator < (const point &a, const point &b) {
26     if (cmptype == 0) return a.x < b.x;
27     else if (cmptype == 1) return a.y < b.y;
28     else return a.z < b.z;
29 }
30
31 struct K_D_Tree{
32     point pos, lpos, rpos;
33     int mx, siz;
34     int ls, rs;
35 } t[MAX];
36
37 int root, tot, top, rub[MAX];
38
39 inline int newnode() {
40     if (top) return rub[top--];
41     else return ++tot;
42 }
43
44 inline void push_up(int u) {
45     t[u].lpos = t[u].rpos = t[u].pos;
46     t[u].mx = t[u].pos.val;

```

```

47     if (lc) {
48         t[u].siz += t[lc].siz;
49         t[u].lpos = min(t[u].lpos, t[lc].lpos);
50         t[u].rpos = max(t[u].rpos, t[lc].rpos);
51         t[u].mx = max(t[u].mx, t[lc].mx);
52     }
53     if (rc) {
54         t[u].siz += t[rc].siz;
55         t[u].lpos = min(t[u].lpos, t[rc].lpos);
56         t[u].rpos = max(t[u].rpos, t[rc].rpos);
57         t[u].mx = max(t[u].mx, t[rc].mx);
58     }
59 }
60
61 inline int build(int l, int r, int type) { //建树
62     if (l > r) return 0;
63     cmptype = type;
64     nth_element(p + l, p + m, p + r + 1);
65     int u = newnode();
66     t[u].pos = p[m];
67     t[u].ls = build(l, m - 1, (type + 1) % 3);
68     t[u].rs = build(m + 1, r, (type + 1) % 3);
69     push_up(u);
70     return u;
71 }
72
73 inline void pia(int u, int num) { //拍扁回炉重做
74     if (lc) pia(lc, num);
75     p[t[lc].siz + num + 1] = t[u].pos, rub[++top] = u;
76     if (rc) pia(rc, t[lc].siz + num + 1);
77 }
78
79 inline void check(int &u, int type) { //检查是否平衡, 不平衡则需要重建
80     if (t[u].siz * alpha < t[lc].siz || t[u].siz * alpha < t[rc].siz) pia(u, 0), u = build
        (1, t[u].siz, type);
81 }
82
83 inline void insert(int &u, int type, point tp) {
84     if (!u) { //新点
85         u = newnode();
86         lc = rc = 0;
87         t[u].pos = tp;
88         push_up(u);
89         return;
90     }
91     cmptype = type;
92     if (tp < t[u].pos) insert(lc, (type + 1) % 3, tp);
93     else insert(rc, (type + 1) % 3, tp);
94     push_up(u);
95     check(u, type);
96 }
97
98 inline bool out(const point &l, const point &r, const point &L, const point &R) { //完全在外
    面

```

```

99     return l.x > R.x || l.y > R.y || l.z > R.z || r.x < L.x || r.y < L.y || r.z < L.z;
100 }
101
102 inline bool in(const point &l, const point &r, const point &L, const point &R) { //完全在里面
103     return l.x <= L.x && R.x <= r.x && l.y <= L.y && R.y <= r.y && l.z <= L.z && R.z <= r.z
104     ;
105 }
106 inline int query(int u, point ql, point qr) { //查询最大值
107     if (!u || out(ql, qr, t[u].lpos, t[u].rpos)) return 0;
108     if (in(ql, qr, t[u].lpos, t[u].rpos)) return t[u].mx;
109     int res = 0;
110     if (in(ql, qr, t[u].pos, t[u].pos)) res = max(res, t[u].pos.val);
111     res = max(res, query(lc, ql, qr));
112     res = max(res, query(rc, ql, qr));
113     return res;
114 }
115
116 inline void init() {
117     root = tot = top = 0;
118 }
119
120 int main() {
121     init();
122     scanf("%d", &N);
123     for (int i = 1; i <= N; i++) scanf("%d%d%d", &nod[i].a, &nod[i].b, &nod[i].c, &nod[i].d);
124     sort(nod + 1, nod + 1 + N);
125     for (int i = 1; i <= N; i++) p[i] = point{nod[i].b, nod[i].c, nod[i].d, 0};
126     int ans = 0;
127     for (int i = 1; i <= N; i++) {
128         ans = max(ans, p[i].val = query(root, point{ -INF, -INF, -INF}, p[i]) + 1);
129         insert(root, 0, p[i]);
130     }
131     printf("%d\n", ans);
132     return 0;
133 }

```

0.8 模拟退火

```

1  const double DOWN = 0.996;
2  const double START_T = 5000;
3  double ansx, ansy, ansz, anse;
4
5  void initAns() {
6      //初始化一个答案点(可以选任意点)
7  }
8
9  double getEnergy(double x, double y, double z) {
10     //具体分析题目
11 }
12
13 void SA() {

```

```

14     double T = START_T;
15     while (T > 1e-15) {
16         double newx = ansx + (rand() * 2 - RAND_MAX) * T;
17         double newy = ansy + (rand() * 2 - RAND_MAX) * T;
18         double newz = ansz + (rand() * 2 - RAND_MAX) * T;
19         double newe = getEnergy(newx, newy, newz);
20         double delta = newe - anse;
21         if (delta < 0) ansx = newx, ansy = newy, ansz = newz, anse = newe;
22         else if (exp(-delta / T) * RAND_MAX > rand())
23             ansx = newx, ansy = newy, ansz = newz;
24         T *= DOWN;
25     }
26 }
27
28 void solve() {
29     initAns();
30     while ((double) clock() / CLOCKS_PER_SEC < 2.0) SA();
31 }

```

0.9 莫队

0.9.1 回滚莫队

```

1 //问题可以莫队（询问可以离线，不带修改）
2 //区间伸长的时候很好维护信息
3 //区间缩短的时候不太好维护信息（如最大值，删除以后不知道次大值是多少）
4
5 struct Hash {
6     int b[N], tot;
7     void init() { tot = 0; }
8     void insert(int x) { b[++tot] = x; }
9     void build() {
10         sort(b + 1, b + 1 + tot);
11         tot = unique(b + 1, b + 1 + tot) - (b + 1);
12     }
13     int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
14 } ha;
15
16 int n, m;
17 int c[N], pos[N], cnt[N], cntt[N];
18 int belong[N], sizz;
19 ll ans[N], res;
20
21 struct Query {
22     int l, r, id;
23     bool operator < (const Query &rhs) const {
24         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
25     }
26 } q[N];
27
28 ll bruteForce(int ql, int qr) {
29     ll result = 0;
30     for (int i = ql; i <= qr; i++) {

```

```

31         cntt[pos[i]]++;
32         result = max(result, 1ll * c[i] * cntt[pos[i]]);
33     }
34     for (int i = ql; i <= qr; i++) cntt[pos[i]]--;
35     return result;
36 }
37
38 void add(int x) {
39     cnt[pos[x]]++;
40     res = max(res, 1ll * c[x] * cnt[pos[x]]);
41 }
42
43 void del(int x) {
44     cnt[pos[x]]--;
45 }
46
47 int main() {
48
49
50     scanf("%d%d", &n, &m);
51     for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
52     ha.build();
53     for (int i = 1; i <= n; i++) pos[i] = ha.pos(c[i]);
54
55     sizz = sqrt(n); int num = ceil((long double)n / sizz);
56     for (int i = 1, j = 1; i <= num; i++)
57         while (j <= i * sizz && j <= n)
58             belong[j++] = i;
59
60     for (int i = 1; i <= m; i++) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
61     sort(q + 1, q + 1 + m);
62
63     for (int i = 1, j = 1; i <= num; i++) {
64         memset(cnt, 0, sizeof(cnt));
65         int right = min(i * sizz, n);
66         res = 0;
67         for (int l = right + 1, r = right; j <= m && belong[q[j].l] == i; j++, l = right +
1) {
68             int ql = q[j].l, qr = q[j].r;
69             if (qr - ql + 1 <= sizz) {
70                 ans[q[j].id] = bruteForce(ql, qr);
71                 continue;
72             }
73             while (r < qr) add(++r);
74             ll tmp = res;
75             while (l > ql) add(--l);
76             ans[q[j].id] = res;
77             res = tmp;
78             while (l < right + 1) del(l++);
79         }
80     }
81
82     for (int i = 1; i <= m; i++) printf("%lld\n", ans[i]);
83

```

```

84     return 0;
85 }

```

0.9.2 带修莫队

```

1
2 //带修莫队模板题
3 //查询 $[ql, qr]$ 间不同颜色数量, 带修改
4
5 int n, m;
6 int c[N], cnt[N];
7 int belong[N], size, totq, totm;
8 int ans[N], res;
9
10 struct Query {
11     int l, r, t, id;
12     bool operator < (const Query &rhs) const {
13         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
14             (belong[r] ^ belong[rhs.r] ? belong[r] < belong[rhs.r] : t < rhs.t);
15     }
16 } q[N];
17
18 struct Modify {
19     int pos, val;
20 } modify[N];
21
22 void add(int x) {
23     if (!cnt[c[x]]) res++;
24     cnt[c[x]]++;
25 }
26
27 void del(int x) {
28     cnt[c[x]]--;
29     if (!cnt[c[x]]) res--;
30 }
31
32 void upd(int x, int ql, int qr) {
33     int pos = modify[x].pos;
34     if (ql <= pos && pos <= qr) {
35         cnt[c[pos]]--; if (!cnt[c[pos]]) res--;
36         if (!cnt[modify[x].val]) res++; cnt[modify[x].val]++;
37     }
38     swap(modify[x].val, c[pos]); // 交换
39 }
40
41 int main() {
42 #ifdef ACM_LOCAL
43     freopen("input.in", "r", stdin);
44     freopen("output.out", "w", stdout);
45 #endif
46     scanf("%d%d", &n, &m);
47     for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
48     for (int i = 1; i <= m; i++) {

```

```

49     char op[10]; scanf("%s", op);
50     if (op[0] == 'Q') {
51         int ql, qr; scanf("%d%d", &ql, &qr); totq++;
52         q[totq] = Query{ql, qr, totm, totq};
53     }
54     else {
55         int pos, val; scanf("%d%d", &pos, &val); totm++;
56         modify[totm] = Modify{pos, val};
57     }
58 }
59
60 //size = N ^ (2 / 3), (N * totm) ^ (1 / 3)
61 size = ceil(pow(n, (long double)2.0 / 3)); int num = ceil((long double)n / size);
62 for (int i = 1, j = 1; i <= num; i++)
63     while (j <= i * size && j <= n)
64         belong[j++] = i;
65
66 sort(q + 1, q + 1 + totq);
67
68 int l = 1, r = 0, t = 0;
69 for (int i = 1; i <= totq; i++) {
70     int ql = q[i].l, qr = q[i].r, qt = q[i].t;
71     while (l < ql) del(l++);
72     while (l > ql) add(--l);
73     while (r < qr) add(++r);
74     while (r > qr) del(r--);
75     while (t < qt) upd(++t, ql, qr);
76     while (t > qt) upd(t--, ql, qr);
77     ans[q[i].id] = res;
78 }
79
80 for (int i = 1; i <= totq; i++) printf("%d\n", ans[i]);
81
82 return 0;
83 }

```

0.9.3 普通莫队

```

1
2 int n, m;
3 int belong[N], size;
4 int ans[N], res;
5
6 struct node {
7     int l, r, id;
8     bool operator < (const node &rhs) const { //奇偶优化
9         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
10             ((belong[l] & 1) ? r < rhs.r : r > rhs.r);
11     }
12 } q[N];
13
14 void add(int x) {
15

```



```

16 }
17
18 void del(int x) {
19
20 }
21
22 int main() {
23
24     size = sqrt(n); int num = ceil((long double)n / size);
25     for (int i = 1, j = 1; i <= num; i++)
26         while (j <= i * size && j <= n)
27             belong[j++] = i;
28
29
30     int l = 1, r = 0;
31     for (int i = 1; i <= m; i++) {
32         int ql = q[i].l, qr = q[i].r;
33         while (l < ql) del(l++);
34         while (l > ql) add(--l);
35         while (r < qr) add(++r);
36         while (r > qr) del(r--);
37         ans[q[i].id] = res;
38     }
39
40
41
42 }

```

0.9.4 树上莫队

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int N = 1e5 + 10;
5
6 struct Hash {
7     int b[N], tot;
8     void init() { tot = 0; }
9     void insert(int x) { b[++tot] = x; }
10    void build() {
11        sort(b + 1, b + 1 + tot);
12        tot = unique(b + 1, b + 1 + tot) - (b + 1);
13    }
14    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
15 } ha;
16
17 int n, m;
18 int c[N], cnt[N];
19 vector<int> g[N];
20 int st[N], ed[N], dfnt, nodeOf[N << 1], tag[N];
21 int belong[N], sizz;
22 int ans[N], res;
23

```

```

24 struct Query {
25     int l, r, id, k;
26     bool operator < (const Query &rhs) const {
27         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
28     }
29 } q[N];
30
31 int son[N], siz[N], top[N], fa[N], dep[N];
32 void dfs(int u, int par) {
33     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
34     int max_son = -1; nodeOf[st[u] = ++dfnt] = u;
35     for (auto &v: g[u])
36         if (v != par) {
37             dfs(v, u);
38             siz[u] += siz[v];
39             if (max_son < siz[v])
40                 son[u] = v, max_son = siz[v];
41         }
42     nodeOf[ed[u] = ++dfnt] = u;
43 }
44 void dfs2(int u, int topf) {
45     top[u] = topf;
46     if (!son[u]) return;
47     dfs2(son[u], topf);
48     for (auto &v: g[u])
49         if (v != fa[u] && v != son[u]) dfs2(v, v);
50 }
51 int lca(int x, int y) {
52     while (top[x] != top[y]) {
53         if (dep[top[x]] < dep[top[y]]) swap(x, y);
54         x = fa[top[x]];
55     }
56     return dep[x] < dep[y] ? x : y;
57 }
58
59 void upd(int x) {
60     x = nodeOf[x];
61     if (tag[x]) {
62         cnt[c[x]]--;
63         if (!cnt[c[x]]) res--;
64     }
65     else {
66         if (!cnt[c[x]]) res++;
67         cnt[c[x]]++;
68     }
69     tag[x] ^= 1;
70 }
71
72 int main() {
73 #ifdef ACM_LOCAL
74     freopen("input.in", "r", stdin);
75     freopen("output.out", "w", stdout);
76 #endif
77     scanf("%d%d", &n, &m);

```

```

78     for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
79     ha.build();
80     for (int i = 1; i <= n; i++) c[i] = ha.pos(c[i]);
81
82     for (int i = 1; i < n; i++) {
83         int u, v; scanf("%d%d", &u, &v);
84         g[u].push_back(v); g[v].push_back(u);
85     }
86     int rt = 1; dfs(rt, 0); dfs2(rt, rt);
87
88     sizz = sqrt(dfnt); int num = ceil((long double)dfnt / sizz);
89     for (int i = 1, j = 1; i <= num; i++)
90         while (j <= i * sizz && j <= dfnt)
91             belong[j++] = i;
92     for (int i = 1; i <= m; i++) {
93         int u, v; scanf("%d%d", &u, &v);
94         int tlca = lca(u, v);
95         if (st[u] > st[v]) swap(u, v);
96         if (u == tlca) q[i] = Query{st[u], st[v], i, 0};
97         else q[i] = Query{ed[u], st[v], i, tlca};
98     }
99
100    sort(q + 1, q + 1 + m);
101
102    int l = 1, r = 0;
103    for (int i = 1; i <= m; i++) {
104        int ql = q[i].l, qr = q[i].r;
105        while (l < ql) upd(l++);
106        while (l > ql) upd(--l);
107        while (r < qr) upd(++r);
108        while (r > qr) upd(r--);
109        ans[q[i].id] = res + (q[i].k ? (cnt[c[q[i].k]] == 0) : 0);
110    }
111
112    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
113
114    return 0;
115 }

```

1 动态规划

1.1 四边形优化

```

1
2 //四边形优化区间dp( $n^3 \rightarrow n^2$ )
3 // $a < b < c < d$ ,  $f[l][r] = \min(f[l][k] + f[k+1][r] + cost(l, r))$ 
4 //1.  $cost(b, c) \leq cost(a, d)$ 
5 //2.  $cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c)$ , 即交叉小于包含
6
7 void solve() {
8     for (int len = 2; len <= n; len++) {
9         for (int l = 1, r; l + len - 1 <= n; l++) {
10             r = l + len - 1;

```

```

11         mn[l][r] = 0x3f3f3f3f;
12         for (int k = m[l][r - 1]; k <= m[l + 1][r]; k++)
13             if (mn[l][k] + mn[k + 1][r] + cost(l, r) < mn[l][r]) {
14                 mn[l][r] = mn[l][k] + mn[k + 1][r] + cost(l, r);
15                 m[l][r] = k;
16             }
17     }
18 }
19 }

```

1.2 数位 DP

```

1  ll dfs(int pos, ... , bool lead, bool isMax) { //当前位pos, ...为省略条件, lead判前导零, isMax
    判前几位数是否选的都是最大值
2      if (!pos) return 1; //此处为越过一个数的最后一位(最小的一位), 如123, 越过3这里说明当前已经是
    123了, 所以只有一个数
3      //有时候不一定是返回1, 看条件
4      if (!isMax && !lead && dp[pos][...][...] != -1) return dp[pos][...][...]; //记忆化, 如果
    有直接返回
5      int up = isMax ? a[pos] : 9; //如果一直是最大, 当前位最多也就是a[pos], 超过了就大于这个数了
6      ll res = 0;
7      for (int i = 0; i <= up; i++) {
8          //按限制条件来
9          //如, 判是否有前导0
10         if (lead) res += dfs(pos - 1, ... , !i, isMax && i == a[pos]);
11         else res += dfs(pos - 1, ... , 0, isMax && i == a[pos]);
12     }
13     return isMax || lead ? res : dp[pos][...][...] = res; //如果有前导0或者是前几位都是最大, 直
    接返回
14     //否则dp[pos][...][...]记录值再返回
15 }
16
17 ll calc(ll x) {
18     int pos = 0;
19     while (x) a[++pos] = x % 10, x /= 10;
20     memset(dp, -1, sz(dp)); //约束是不确定的 跟数有关
21     //约束确定才放外面, 比如不要62, 对于每一个数都是不要62
22     return dfs(pos, ... , 1, 1);
23 }

```

1.3 树上背包

```

1  //加了剪枝后复杂度为O(NM)
2
3  void dfs(int u, int fa) {
4      siz[u] = 1;
5      for (auto &v: g[u])
6          if (v != fa) {
7              dfs(v, u);
8
9              int now = min(siz[u] + siz[v] + 1, M);
10             int t[MAX_M]; for (int i = 0; i <= M; i++) t[i] = INF/-INF; //初始化

```

```

11         for (int i = 0; i <= siz[u]; i++)
12             for (int j = 0; j <= siz[v] && i + j <= M; j++) {
13                 //...转移方程
14             }
15         for (int i = 0; i <= now; i++) f[u][i] = min/max(f[u][i], t[i]);
16         siz[u] = now;
17     }
18 }

```

1.4 环基树 DP

```

1  int flag, S, E; //flag是否找到环, SE为环上两个点
2
3  void findCircle(int u, int fa) {
4      vis[u] = 1;
5      for (int i = head[u], v; i; i = e[i].nxt)
6          if ((v = e[i].to) != fa) {
7              if (vis[v]) flag = 1, S = u, E = v;
8              else findCircle(v, u);
9          }
10 }
11
12 void dp(int u, int fa) {
13     //dp过程
14
15     for (int i = head[u], v; i; i = e[i].nxt)
16         if ((v = e[i].to) != fa && v) {
17             dp(v, u);
18
19         }
20 }
21
22 ll calc(int u) {
23     flag = 0;
24     findCircle(u, 0);
25     if (flag) {
26         for (int i = head[S], v; i; i = e[i].nxt)
27             if ((v = e[i].to) == E) {
28                 e[i].to = e[i ^ 1].to = 0; //删边操作, 注意e[tot]中tot从2开始
29                 break;
30             }
31         ll res = 0;
32         dp(S, 0); res = max(res, ...);
33         dp(E, 0); res = max(res, ...);
34         return res;
35     }
36     else {
37         dp(u, 0);
38         return ...;
39     }
40 }

```

2 图论

2.1 BellmanFord

```
1 struct edge {
2     int u, v, w;
3 } e[MAX];
4
5 bool BellmanFord(int s) { //判负环
6     for (int i = 1; i <= n; i++) dis[i] = (i == s ? 0 : INF);
7     for (int i = 1; i < n; i++)
8         for (int j = 1; j <= m; j++)
9             if (dis[e[j].u] + e[j].w < dis[e[j].v])
10                 dis[e[j].v] = dis[e[j].u] + e[j].w;
11     for (int i = 1; i <= m; i++)
12         if (dis[e[i].u] + e[i].w < dis[e[i].v])
13             return false;
14     return true;
15 }
```

2.2 Dijkstra

```
1 //O((N+M)LogM)
2 struct Node {
3     int now, d;
4     bool operator < (const Node &rhs) const {
5         return d > rhs.d;
6     }
7 };
8
9 priority_queue<Node> q;
10
11 void dijkstra(int s) {
12     for (int i = 1; i <= n; i++) dis[i] = INF, vis[i] = 0;
13     dis[s] = 0;
14     q.push(Node{s, dis[s]});
15     while (!q.empty()) {
16         Node p = q.top(); q.pop();
17         int u = p.now;
18         if (vis[u]) continue;
19         vis[u] = 1;
20         for (int i = head[u], v; i; i = e[i].nxt)
21             if (dis[u] + e[i].w < dis[v = e[i].to]) {
22                 dis[v] = dis[u] + e[i].w;
23                 if (!vis[v]) q.push(Node{v, dis[v]});
24             }
25     }
26 }
```

2.3 Kruskal

```

2  int pre[N], m;
3
4  struct edge {
5      int u, v, w;
6      bool operator < (const edge &rhs) const {
7          return w < rhs.w;
8      }
9  } e[M * M];
10
11 int find(int x) { return x == pre[x] ? x : pre[x] = find(pre[x]); }
12
13 int kruskal() {
14     int cnt = 0, ans = 0;
15     for (int i = 1; i <= m; i++) {
16         int u = find(e[i].u), v = find(e[i].v);
17         if (u == v) continue;
18         ans += e[i].w;
19         pre[v] = u;
20         cnt++;
21         if (cnt == n - 1) break;
22     }
23     return ans;
24 }

```

2.4 Kruskal 重构树

```

1  //用于解决图中两点间多条路中最大/小边权最小/大值问题
2  //u, v为原图上的点, 则val[lca(u, v)]为u->v间多条路中...
3  //由于u, v可能在原图中不连通, 所以需要find(u)和find(v)判断一下是不是在一棵树中
4  //经典问题, 如P4768 [NOI2018]归程, 求v->1路径中前半段路所有路径海拔大于给定h, 后半段路不管海拔高
   度的最短路
5  //一条路中最大边权小于等于给定值, 求....
6
7  struct Edge {
8      int u, v, w;
9      bool operator < (const Edge &rhs) const {
10         return w > rhs.w;
11         //升序为(u, v)间多条路中最大边权最小值
12         //降序为(u, v)间多条路中最小边权最大值
13     }
14 } E[M];
15 vector<int> g[N];
16 int pre[N], val[N], cnt; //cnt为kruskal重构树的点数, 点数最多为2N - 1
17 int son[N], siz[N], top[N], fa[N], dep[N];
18 void dfs(int u, int par) {
19     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
20     int max_son = -1;
21     for (auto &v: g[u])
22         if (v != par) {
23             dfs(v, u);
24             siz[u] += siz[v];
25             if (max_son < siz[v])
26                 son[u] = v, max_son = siz[v];

```

```

27     }
28 }
29 void dfs2(int u, int topf) {
30     top[u] = topf;
31     if (!son[u]) return;
32     dfs2(son[u], topf);
33     for (auto &v: g[u])
34         if (v != fa[u] && v != son[u]) dfs2(v, v);
35 }
36 int lca(int x, int y) {
37     while (top[x] != top[y]) {
38         if (dep[top[x]] < dep[top[y]]) swap(x, y);
39         x = fa[top[x]];
40     }
41     return dep[x] < dep[y] ? x : y;
42 }
43 int find(int x) { return x == pre[x] ? x : pre[x] = find(pre[x]); }
44 void exKruskal() {
45     cnt = N; for (int i = 1; i < (N << 1); i++) pre[i] = i;
46     sort(E + 1, E + 1 + M);
47     for (int i = 1; i <= M; i++) {
48         int u = find(E[i].u), v = find(E[i].v);
49         if (u == v) continue;
50         val[++cnt] = E[i].w;
51         pre[u] = pre[v] = cnt;
52         g[u].push_back(cnt), g[cnt].push_back(u);
53         g[v].push_back(cnt), g[cnt].push_back(v);
54         if (cnt == (N << 1) - 1) break;
55     }
56     //原图不连通的情况，那形成的就是森林
57     for (int i = 1; i <= cnt; i++)
58         if (!siz[i]) { //未访问过
59             int rt = find(i); //下树剖lca
60             dfs(rt, 0); dfs2(rt, rt);
61         }
62 }

```

2.5 Prim

```

1 struct Node {
2     int to, d;
3     bool operator < (const Node &rhs) const {
4         return d > rhs.d;
5     }
6 };
7
8 ll prim(int n) {
9     vector<int> vis(n + 1);
10    priority_queue<Node> q; q.push({1, 0});
11    ll ans = 0, cnt = 0;
12    while (!q.empty() && cnt <= n) {
13        Node now = q.top(); q.pop();
14        int u = now.to, dis = now.d;

```



```

15         if (vis[u]) continue;
16         vis[u] = 1, cnt++, ans += dis;
17         for (int i = head[u], v; i; i = e[i].nxt)
18             if (!vis[v = e[i].to]) q.push({v, e[i].w});
19     }
20     return ans;
21 }

```

2.6 SPFA

```

1 bool spfa(int s) {
2     for (int i = 1; i <= n; i++) dis[i] = (i == s ? 0 : INF), vis[i] = (i == s), cnt[i] = 0;
3     queue<int> q; q.push(s);
4     while (!q.empty()) {
5         int u = q.front();
6         q.pop();
7         vis[u] = 0, cnt[u]++;
8         if (cnt[u] >= n) return false;
9         for (int i = head[u], v; i; i = e[i].nxt)
10             if (dis[u] + e[i].w < dis[v = e[i].to]) {
11                 dis[v] = dis[u] + e[i].w;
12                 if (!vis[v]) q.push(v), vis[v] = 1;
13             }
14     }
15     return true;
16 }
17
18 bool spfa(int u) { //dfs
19     vis[u] = 1;
20     for (int i = head[u]; i; i = e[i].nxt)
21         if (dis[u] + e[i].w < dis[v = e[i].to]) {
22             if (vis[v]) return false;
23             else {
24                 dis[v] = dis[u] + e[i].w;
25                 if (!spfa(v)) return false;
26             }
27         }
28     vis[u] = 0;
29     return true;
30 }

```

2.7 严格次小生成树

```

1 const int N = 1e5 + 10;
2 const int M = 3e5 + 10;
3 const int INF = 0x3fffffff;
4 const ll INF64 = 0x3fffffffffffffffffLL;
5
6 struct Edge {
7     int u, v, w;
8     bool operator < (const Edge &rhs) const { return w < rhs.w; }

```

```

9  };
10
11  Edge e[M];
12  bool used[M];
13
14  int n, m;
15
16  class Tr {
17  private:
18      struct Edge {
19          int to, nxt, val;
20      } e[M << 1];
21      int tot, head[N];
22
23      int par[N][22], dep[N];
24      // 到祖先的路径上边权最大的边
25      int maxx[N][22];
26      // 到祖先的路径上边权次大的边, 若不存在则为 -INF
27      int minn[N][22];
28
29  public:
30      void add(int u, int v, int val) {
31          e[++tot] = (Edge){v, head[u], val};
32          head[u] = tot;
33      }
34
35      void insedge(int u, int v, int val) {
36          add(u, v, val);
37          add(v, u, val);
38      }
39
40      void dfs(int u, int fa) {
41          dep[u] = dep[fa] + 1;
42          par[u][0] = fa;
43          minn[u][0] = -INF;
44          for (int i = 1; (1 << i) <= dep[u]; i++) {
45              par[u][i] = par[par[u][i - 1]][i - 1];
46              int kk[4] = {maxx[u][i - 1], maxx[par[u][i - 1]][i - 1],
47                          minn[u][i - 1], minn[par[u][i - 1]][i - 1]};
48              // 从四个值中取得最大值
49              sort(kk, kk + 4);
50              maxx[u][i] = kk[3];
51              // 取得严格次大值
52              int ptr = 2;
53              while (ptr >= 0 && kk[ptr] == kk[3]) ptr--;
54              minn[u][i] = (ptr == -1 ? -INF : kk[ptr]);
55          }
56
57          for (int i = head[u]; i; i = e[i].nxt)
58              if (e[i].to != fa) {
59                  maxx[e[i].to][0] = e[i].val;
60                  dfs(e[i].to, u);
61              }
62      }

```

```

63
64     int lca(int u, int v) {
65         if (dep[u] < dep[v]) swap(u, v);
66         for (int i = 21; i >= 0; i--)
67             if (dep[par[u][i]] >= dep[v]) u = par[u][i];
68         if (u == v) return u;
69         for (int i = 21; i >= 0; i--)
70             if (par[u][i] != par[v][i]) {
71                 u = par[u][i];
72                 v = par[v][i];
73             }
74         return par[u][0];
75     }
76
77     int query(int u, int v, int val) {
78         int res = -INF;
79         for (int i = 21; i >= 0; i--)
80             if (dep[par[u][i]] >= dep[v]) {
81                 if (val != maxx[u][i]) res = max(res, maxx[u][i]);
82                 else res = max(res, minn[u][i]);
83                 u = par[u][i];
84             }
85         return res;
86     }
87 } tr;
88
89 int pre[N];
90 int find(int x) { return pre[x] == x ? x : pre[x] = find(pre[x]); }
91
92 ll Kruskal() {
93     int tot = 0; ll sum = 0;
94     sort(e + 1, e + m + 1);
95     for (int i = 1; i <= n; i++) pre[i] = i;
96
97     for (int i = 1; i <= m; i++) {
98         int a = find(e[i].u);
99         int b = find(e[i].v);
100         if (a != b) {
101             pre[a] = b;
102             tot++;
103             tr.insedge(e[i].u, e[i].v, e[i].w);
104             sum += e[i].w;
105             used[i] = 1;
106         }
107         if (tot == n - 1) break;
108     }
109     return sum;
110 }
111
112 int main() {
113
114
115     scanf("%d%d", &n, &m);
116     for (int i = 1; i <= m; i++) {

```

```

117     int u, v, w; scanf("%d%d%d", &u, &v, &w);
118     e[i] = {u, v, w};
119 }
120
121 ll sum = Kruskal();
122 ll ans = INF64;
123 tr.dfs(1, 0);
124
125 for (int i = 1; i <= m; i++) {
126     if (!used[i]) {
127         int _lca = tr.lca(e[i].u, e[i].v);
128         // 找到路径上不等于 e[i].val 的最大边权
129         ll tmpa = tr.query(e[i].u, _lca, e[i].w);
130         ll tmpb = tr.query(e[i].v, _lca, e[i].w);
131         // 这样的边可能不存在, 只在这样的边存在时更新答案
132         if (max(tmpa, tmpb) > -INF)
133             ans = min(ans, sum - max(tmpa, tmpb) + e[i].w);
134     }
135 }
136 // 次小生成树不存在时输出 -1
137 printf("%lld\n", ans == INF64 ? -1 : ans);
138 return 0;
139 }

```

2.8 二分图判定

```

1 //不存在奇环即为二分图
2 int n, m;
3 int pre[N << 1], rk[N << 1];
4
5 int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }
6
7 void merge(int x, int y) {
8     x = find(x), y = find(y);
9     if (x == y) return;
10    if (rk[x] < rk[y]) swap(x, y);
11    rk[x] += rk[x] == rk[y];
12    pre[y] = x;
13 }
14
15 int main() {
16
17     for (int i = 1; i <= n << 1; i++) pre[i] = i, rk[i] = 0;
18
19     int u, v; scanf("%d%d", &u, &v);
20     if (find(u) == find(v)) cnt++; //flag = 0
21     else {
22         merge(u + n, v);
23         merge(v + n, u);
24     }
25
26
27

```

28 }

2.9 匹配问题

2.9.1 GaleShapley

```
1  #include<iostream>
2  using namespace std;
3
4  const int N=4;
5
6  void GaleShapley(const int (&man)[MAX][MAX], const int (&woman)[MAX][MAX], int (&match)[MAX
    ]) {
7      int wm[MAX][MAX];    // wm[i][j]: rank from girl i to boy j
8      int choose[MAX];    // choose[i]: current boyfriend of girl i
9      int manIndex[MAX]; //   manIndex[i]: how many girls that have rejected boy i
10     int i, j;
11     int w, m;
12     for (i = 0; i < N; i++) {
13         match[i] = -1;
14         choose[i] = -1;
15         manIndex[i] = 0;
16         for (j = 0; j < N; j++)
17             wm[i][woman[i][j]] = j;
18     }
19
20     bool bSingle = false;
21     while (!bSingle) {
22         bSingle = true;
23         for (i = 0; i < N; i++) {
24             if (match[i] != -1) // boy i already have a girlfriend
25                 continue;
26             bSingle = false;
27             j = manIndex[i]++; // the jth girl that boy i like most
28             w = man[i][j];
29             m = choose[w];    // current girl w's boyfriend
30             if (m == -1 || wm[w][i] < wm[w][m]) { // if girl w prefer boy i
31                 match[i] = w;
32                 choose[w] = i;
33                 if (m != -1)
34                     match[m] = -1;
35             }
36         }
37     }
38 }
39
40
41 void Print(const int(&match)[MAX], int N) {
42     for (int i = 0; i < N; i++)
43         cout << i << " " << match[i] << endl;
44 }
45
46
```

```

47 int main(){
48     int man[N][N]={
49         {2,3,1,0},
50         {2,1,3,0},
51         {0,2,3,1},
52         {1,3,2,0},
53     };
54     int woman[N][N]={
55         {0,3,2,1},
56         {0,1,2,3},
57         {0,2,3,1},
58         {1,0,3,2},
59     };
60
61     int match[N];
62     GaleShapley(man,woman,match);
63     Print(match,N);
64
65     return 0;
66 }

```

2.9.2 Hungry

```

1 //不带权重的最大匹配, 复杂度 $O(nm)$ 
2 int used[N], match[N];
3 vector<int> g[N];
4
5 bool find(int u) {
6     for (auto &v: g[u])
7         if (!used[v]) {
8             used[v] = 1;
9             if (!match[v] || find(match[v])) {
10                 match[v] = u;
11                 return true;
12             }
13         }
14     return false;
15 }
16
17 int hungry() {
18     int res = 0;
19     for (int i = 1; i <= n; i++) match[i] = 0;
20     for (int i = 1; i <= n; i++) {
21         for (int j = 1; j <= n; j++) used[j] = 0;
22         if (find(i)) res++;
23     }
24     return res;
25 }

```

2.9.3 KM

```

1 //https://ac.nowcoder.com/acm/contest/view-submission?submissionId=44654655
2

```

```

3 struct KM {
4 #define type int
5     ///define inf 0x3f3f3f3f
6     static const int N = 505;
7     static const int INF = 0x3f3f3f3f;
8     int n, mx[N], my[N], prv[N];
9     type slk[N], lx[N], ly[N], w[N][N];
10    bool vx[N], vy[N];
11
12    void init(int siz) {
13        n = siz;
14        for (int i = 1; i <= n; i++) {
15            for (int j = 1; j <= n; j++) {
16                w[i][j] = -505;
17            }
18        }
19    }
20
21    void add_edge(int x, int y, type val) { w[x][y] = val; }
22
23    void match(int y) { while (y) swap(y, mx[my[y] = prv[y]]); }
24
25    void bfs(int x) {
26        int i, y;
27        type d;
28        for (i = 1; i <= n; i++) {
29            vx[i] = vy[i] = 0;
30            slk[i] = INF;
31        }
32        queue<int> q;
33        q.push(x);
34        vx[x] = 1;
35        while (1) {
36            while (!q.empty()) {
37                x = q.front();
38                q.pop();
39                for (y = 1; y <= n; y++) {
40                    d = lx[x] + ly[y] - w[x][y];
41                    if (!vy[y] && d <= slk[y]) {
42                        prv[y] = x;
43                        if (!d) {
44                            if (!my[y]) return match(y);
45                            q.push(my[y]);
46                            vx[my[y]] = 1;
47                            vy[y] = 1;
48                        } else slk[y] = d;
49                    }
50                }
51            }
52            d = INF + 1;
53            for (i = 1; i <= n; i++) {
54                if (!vy[i] && slk[i] < d) {
55                    d = slk[i];
56                    y = i;

```

```

57         }
58     }
59     for (i = 1; i <= n; i++) {
60         if (vx[i]) lx[i] -= d;
61         if (vy[i]) ly[i] += d;
62         else slk[i] -= d;
63     }
64     if (!my[y]) return match(y);
65     q.push(my[y]);
66     vx[my[y]] = 1;
67     vy[y] = 1;
68 }
69 }
70
71 type max_match() {
72     int i;
73     type res;
74     for (i = 1; i <= n; i++) {
75         mx[i] = my[i] = ly[i] = 0;
76         lx[i] = *max_element(w[i] + 1, w[i] + n + 1);
77     }
78     for (i = 1; i <= n; i++) bfs(i);
79     res = 0;
80     for (i = 1; i <= n; i++) res += lx[i] + ly[i];
81     return res;
82 }
83
84 #undef type
85 };

```

2.9.4 一些结论

```

1  /*
2  最大匹配数：最大匹配的匹配边的数目
3
4  最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择
5
6  最大独立数：选取最多的点，使任意所选两点均不相连
7
8  最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长
   可以为 0（即单个点）。
9
10 定理1：最大匹配数 = 最小点覆盖数（这是 Konig 定理）
11
12 定理2：最大匹配数 = 最大独立数
13
14 定理3：最小路径覆盖数 = 顶点数 - 最大匹配数
15  */

```

2.10 同余最短路

```

1  /*

```



```

2 洛谷P3403: 给定 $x[0], x[1], x[2], \dots, x[n-1]$ , 对于 $k \leq h$ , 求有多少个 $k$ 满足 $a[0]x[0] + a[1]x[1] + \dots + a[n-1]x[n-1] = k$ 
3 洛谷P2662: 最大的不能被 $x[0], x[1], \dots, x[n-1]$ 表示的数(从小到大), 显然如果 $\gcd(x[i]) = x[0]$ , 无解, 否则跑同余最短路求出 $\max(dis) - x[0]$ 即为答案
4 解决形如上述类型的题目
5  */
6
7 struct Node {
8     ll now, d;
9     bool operator < (const Node &rhs) const {
10         return d > rhs.d;
11     }
12 };
13
14 priority_queue<Node> q;
15
16 ll dis[N];
17
18 void dijkstra(int s, int n) {
19     vector<int> vis(n);
20     for (int i = 0; i < n; i++) dis[i] = INF;
21     dis[s] = 0;
22     q.push(Node{s, dis[s]});
23     while (!q.empty()) {
24         Node p = q.top(); q.pop();
25         int u = p.now;
26         if (vis[u]) continue;
27         vis[u] = 1;
28         for (int i = head[u], v; i; i = e[i].nxt)
29             if (dis[u] + e[i].w < dis[v = e[i].to]) {
30                 dis[v] = dis[u] + e[i].w;
31                 if (!vis[v]) q.push(Node{v, dis[v]});
32             }
33     }
34 }
35
36 ll solve(ll *x, int n, ll h) {
37     sort(x, x + n);
38     if (x[0] == 1) return h;
39     for (int i = 0; i < x[0]; i++)
40         for (int j = 1; j < n; j++)
41             add(i, (i + x[j]) % x[0], x[j]);
42     dijkstra(1, x[0]);
43     ll res = 0;
44     for (int i = 0; i < x[0]; i++)
45         if (dis[i] <= h) res += (h - dis[i] + x[0] - 1) / x[0];
46     return res;
47 }
48
49 ll x[N];
50
51 int main() {
52 #ifdef ACM_LOCAL
53     freopen("input.in", "r", stdin);

```

```

54     freopen("output.out", "w", stdout);
55 #endif
56     ll h, n; scanf("%lld", &h);
57     for (int i = 0; i < n; i++) scanf("%lld", &x[i]);
58     printf("%lld\n", solve(x, n, h));
59
60
61     return 0;
62 }

```

2.11 差分约束

```

1  /*
2  差分约束是解决这样一类问题
3  给出n个形如 $x[j] - x[i] \leq k$ 的式子, 求 $x[n] - x[1]$ 的最大/最小值
4  最大值—>把所有式子整理为 $x[j] - x[i] \leq k$ , 从i向j连一条边权为k的边, 跑最短路
5  最小值—>把所有式子整理为 $x[j] - x[i] \geq k$ , 从i向j连一条边权为k的边, 跑最长路
6  注意初始化 有时候需要超级源点0
7  */
8
9  bool spfa(int u) { //dfs跑差分约束最短路
10     vis[u] = 1;
11     for (int i = head[u], v; i; i = e[i].nxt)
12         if (dis[u] + e[i].w < dis[v = e[i].to]) {
13             if (vis[v]) return false;
14             else {
15                 dis[v] = dis[u] + e[i].w;
16                 if (!spfa(v)) return false;
17             }
18         }
19     vis[u] = 0;
20     return true;
21 }

```

2.12 拓扑排序

```

1  void topo() {
2      queue<int> q;
3      for (int i = 1; i <= N; i++)
4          if (!degree[i]) q.push(i);
5      while (!q.empty()) {
6          int u = q.front(); q.pop();
7          //
8          for (auto &v: g[u]) {
9              degree[v]--;
10             if (!degree[v]) q.push(v);
11         }
12     }
13     //
14 }

```

2.13 矩阵树

```
1  /*
2  计算生成树个数
3  即求  $\sum_{Tree} \prod_{e \in Tree} num(e)$ 
4  */
5
6  ll gauss(int n, ll K[][N]) { //求矩阵K的n-1阶顺序主子式
7      ll res = 1;
8      for (int i = 1; i <= n - 1; i++) { //枚举主对角线上第i个元素
9          for (int j = i + 1; j <= n - 1; j++) { //枚举剩下的行
10             while (K[j][i]) { //辗转相除
11                 int t = K[i][i] / K[j][i];
12                 for (int k = i; k <= n - 1; k++) //转为倒三角
13                     K[i][k] = (K[i][k] - t * K[j][k] + mod) % mod;
14                 swap(K[i], K[j]); //交换i、j两行
15                 res = -res; //取负
16             }
17         }
18         res = (res * K[i][i]) % mod;
19     }
20     return (res + mod) % mod;
21 }
22
23 int n, m;
24 int K[N][N];
25
26 int main() {
27
28     for (int i = 1; i <= m; i++) {
29         int u, v; ll w; scanf("%d%d%lld", &u, &v, &w);
30         K[u][u]++, K[v][v]++, K[u][v]--, K[v][u]--;
31     }
32
33     ll ans = gauss(n, K);
34
35     return 0;
36 }
```

2.14 网络流

2.14.1 Dinic

```
1  //理论复杂度 $O(n^2m)$ , 求解二分图匹配问题时, 时间复杂度为 $O(m \sqrt{n})$ 
2  struct Dinic {
3      static const int N = ...; //size
4      struct Edge { int from, to, cap, flow; };
5      int n, m, s, t;
6      vector<Edge> edges;
7      vector<int> G[N];
8      int dep[N], cur[N];
9      bool vis[N];
10 }
```

```

11 void init(int siz) { n = siz; for (int i = 0; i < siz; i++) G[i].clear(); edges.clear()
    ; }
12
13 void addEdge(int from, int to, int cap) {
14     edges.push_back(Edge{from, to, cap, 0});
15     edges.push_back(Edge{to, from, 0, 0});
16     m = edges.size();
17     G[from].push_back(m - 2);
18     G[to].push_back(m - 1);
19 }
20
21 bool bfs() {
22     memset(vis, 0, sizeof(vis));
23     queue<int> q; q.push(s); dep[s] = 0, vis[s] = 1;
24     while (!q.empty()) {
25         int x = q.front();
26         q.pop();
27         for (auto &v: G[x]) {
28             Edge& e = edges[v];
29             if (!vis[e.to] && e.cap > e.flow) {
30                 dep[e.to] = dep[x] + (vis[e.to] = 1);
31                 q.push(e.to);
32             }
33         }
34     }
35     return vis[t];
36 }
37
38 int dfs(int u, int a) {
39     if (u == t || a == 0) return a;
40     int flow = 0;
41     for (int& i = cur[u], f; i < G[u].size(); i++) {
42         Edge& e = edges[G[u][i]];
43         if (dep[u] + 1 == dep[e.to] && (f = dfs(e.to, min(a, e.cap - e.flow))) > 0) {
44             e.flow += f;
45             edges[G[u][i] ^ 1].flow -= f;
46             flow += f;
47             a -= f;
48             if (a == 0) break;
49         }
50     }
51     return flow;
52 }
53
54 int maxFlow(int S, int T) {
55     s = S, t = T;
56     int flow = 0;
57     while (bfs()) {
58         memset(cur, 0, sizeof(cur));
59         flow += dfs(S, INF);
60     }
61     return flow;
62 }
63 } flow;

```

2.14.2 ISAP

```
1 struct ISAP {
2     const static int N = ...; // node size
3     struct Edge {
4         int from, to, cap, flow;
5         bool operator < (const Edge &rhs) const {
6             return from < rhs.from || (from == rhs.from && to < rhs.to);
7         }
8     };
9     int n, m, s, t;
10    vector<Edge> edges;
11    vector<int> g[N];
12    bool vis[N];
13    int dep[N], cur[N], p[N], num[N];
14
15    void addEdge(int from, int to, int cap) {
16        edges.push_back(Edge{from, to, cap, 0});
17        edges.push_back(Edge{to, from, 0, 0});
18        m = edges.size();
19        g[from].push_back(m - 2);
20        g[to].push_back(m - 1);
21    }
22
23    bool bfs() {
24        memset(vis, 0, sizeof(vis));
25        queue<int> q; q.push(t); vis[t] = 1, dep[t] = 0;
26        while (!q.empty()) {
27            int u = q.front(); q.pop();
28            for (auto &v: g[u]) {
29                Edge &e = edges[v ^ 1];
30                if (!vis[e.from] && e.cap > e.flow) {
31                    dep[e.from] = dep[u] + (vis[e.from] = 1);
32                    q.push(e.from);
33                }
34            }
35        }
36        return vis[s];
37    }
38
39    void init(int siz) {
40        n = siz;
41        for (int i = 0; i < siz; i++) g[i].clear();
42        edges.clear();
43    }
44
45    int augment() {
46        int u = t, a = INF;
47        while (u != s) {
48            Edge &e = edges[p[u]];
49            a = min(a, e.cap - e.flow);
50            u = edges[p[u]].from;
51        }
52        u = t;
```

```

53     while (u != s) {
54         edges[p[u]].flow += a;
55         edges[p[u] ^ 1].flow -= a;
56         u = edges[p[u]].from;
57     }
58     return a;
59 }
60
61 int maxFlow(int S, int T) {
62     s = S, t = T;
63     int flow = 0; bfs();
64     memset(num, 0, sizeof(num));
65     for (int i = 0; i < n; i++) num[dep[i]]++;
66     int u = S;
67     memset(cur, 0, sizeof(cur));
68     while (dep[S] < n) {
69         if (u == T) {
70             flow += augment();
71             u = S;
72         }
73         int ok = 0;
74         for (int i = cur[u]; i < g[u].size(); i++) {
75             Edge &e = edges[g[u][i]];
76             if (e.cap > e.flow && dep[u] == dep[e.to] + 1) {
77                 ok = 1;
78                 p[e.to] = g[u][i];
79                 cur[u] = i;
80                 u = e.to;
81                 break;
82             }
83         }
84         if (!ok) {
85             int mn = n - 1;
86             for (int i = 0; i < g[u].size(); i++) {
87                 Edge &e = edges[g[u][i]];
88                 if (e.cap > e.flow) mn = min(mn, dep[e.to]);
89             }
90             if (--num[dep[u]] == 0) break;
91             num[dep[u] = mn + 1]++;
92             cur[u] = 0;
93             if (u != S) u = edges[p[u]].from;
94         }
95     }
96     return flow;
97 }
98
99 } flow;

```

2.14.3 MCMF

```

1 //spfa费用流
2 struct MCMF {
3     static const int N = ...;

```

```

4     static const int M = ...;
5     struct Edge { int nxt, to, cap, cost; } e[M << 1];
6     int head[N], tot;
7     int n, m;
8     int cur[N], dis[N], minCost;
9     bool vis[N];
10
11     void init(int siz) { n = siz, tot = 1; for (int i = 0; i < siz; i++) head[i] = 0; }
12
13     void add(int u, int v, int w, int c) { e[++tot] = Edge{head[u], v, w, c}; head[u] = tot
14     ; }
15
16     void addEdge(int u, int v, int w, int c) { add(u, v, w, c), add(v, u, 0, -c); }
17
18     bool spfa(int s, int t) {
19         for (int i = 0; i < n; i++) dis[i] = INF, cur[i] = head[i];
20         queue<int> q; q.push(s), dis[s] = 0, vis[s] = 1;
21         while (!q.empty()) {
22             int u = q.front();
23             q.pop(), vis[u] = 0;
24             for (int i = head[u], v; i; i = e[i].nxt) {
25                 if (e[i].cap && dis[v = e[i].to] > dis[u] + e[i].cost) {
26                     dis[v] = dis[u] + e[i].cost;
27                     if (!vis[v]) q.push(v), vis[v] = 1;
28                 }
29             }
30             return dis[t] != INF;
31         }
32
33     int dfs(int u, int t, int flow) {
34         if (u == t) return flow;
35         vis[u] = 1;
36         int ans = 0;
37         for (int &i = cur[u], v; i && ans < flow; i = e[i].nxt) {
38             if (!vis[v = e[i].to] && e[i].cap && dis[v] == dis[u] + e[i].cost) {
39                 int x = dfs(v, t, min(e[i].cap, flow - ans));
40                 if (x) minCost += x * e[i].cost, e[i].cap -= x, e[i ^ 1].cap += x, ans += x
41                 ;
42             }
43             vis[u] = 0;
44             return ans;
45         }
46
47     pair<int, int> mcmf(int s, int t) {
48         int ans = 0; minCost = 0;
49         while (spfa(s, t)) {
50             int x;
51             while ((x = dfs(s, t, INF))) ans += x;
52         }
53         return make_pair(ans, minCost);
54     }
55 } flow;

```

2.15 连通分量

2.15.1 割点

```
1
2 int dfn[N], low[N], cnt, tot;
3 bool cut[N];
4
5 void tarjan(int u, int topf) { //无向图割点
6     dfn[u] = low[u] = ++cnt;
7     int child = 0;
8     for (auto &v: g[u]) {
9         if (!dfn[v]) {
10             tarjan(v, topf);
11             low[u] = min(low[u], low[v]);
12             if (low[v] >= dfn[u] && u != topf) cut[u] = 1;
13             if (u == topf) child++;
14         }
15         low[u] = min(low[u], dfn[v]);
16     }
17     if (child >= 2 && u == topf) cut[u] = 1;
18 }
```

2.15.2 桥

```
1 vector<int> g[N];
2 int low[N], dfn[N], fa[N], dfnt, cnt_bridge;
3 bool isbridge[N]; // (x, fa[x]) 为桥
4
5 void tarjan(int u, int par) {
6     fa[u] = par;
7     low[u] = dfn[u] = ++dfnt;
8     for (auto &v: g[u]) {
9         if (!dfn[v]) {
10             tarjan(v, u);
11             low[u] = min(low[u], low[v]);
12             if (low[v] > dfn[u]) {
13                 isbridge[v] = true;
14                 cnt_bridge++;
15             }
16         }
17         else if (dfn[v] < dfn[u] && v != par) {
18             low[u] = min(low[u], dfn[v]);
19         }
20     }
21 }
```

2.15.3 连通分量

```
1 vector<int> g[N];
2 vector<pii> edge;
3 int dfn[N], low[N], vis[N], dfnt;
4 int color[N], siz[N], col;
```



```

5  int st[N], top;
6
7  void tarjan(int u, int fa) {
8      dfn[u] = low[u] = ++dfnt;
9      st[++top] = u;
10     vis[u] = 1;
11     for (auto &v: g[u]) {
12         if (v == fa) continue;//有向图去掉
13         if (!dfn[v]) {
14             tarjan(v, u);
15             low[u] = min(low[u], low[v]);
16         }
17         else if (vis[v]) low[u] = min(low[u], dfn[v]);
18     }
19     if (dfn[u] == low[u]) {
20         color[u] = ++col;
21         vis[u] = 0;
22         ++siz[col];
23         while (st[top] != u) {
24             ++siz[col];
25             color[st[top]] = col;
26             vis[st[top--]] = 0;
27         }
28         --top;
29     }
30 }
31
32 void init() {
33     dfnt = top = col = 0;
34     for (int i = 1; i <= n; i++) g[i].clear();
35 }
36
37 int main() {
38
39
40     for (int i = 1; i <= n; i++) if (!dfn[i]) tarjan(i, 0);
41     for (int i = 1; i <= n; i++) g[i].clear();
42     for (auto &i: edge) {
43         int u = color[i.first], v = color[i.second];
44         if (u != v) {
45             g[u].push_back(v);
46             g[v].push_back(u);
47         }
48     }
49
50 }

```

3 字符串

3.1 KMP

```

1  vector<int> getNext(string s) {
2      int n = s.length();

```

```

3     vector<int> nxt(n);
4     for (int i = 1; i < n; i++) {
5         int j = nxt[i - 1];
6         while (j > 0 && s[i] != s[j]) j = nxt[j - 1];
7         if (s[i] == s[j]) j++;
8         nxt[i] = j;
9     }
10    return nxt;
11 }
12
13 int nxt[MAX];
14 void getNext(string str) {
15     nxt[1] = 0;
16     int j = 0, len = str.length();
17     for (int i = 2; i <= len; i++) {
18         while (j && str[j + 1] != str[i]) j = nxt[j];
19         if (str[j + 1] == str[i]) j++;
20         nxt[i] = j;
21     }
22 }
23
24 int main() {
25     int N = strlen(s + 1), M = strlen(t + 1);
26     int j = 0;
27     for (int i = 1; i <= N; i++) {
28         while (j && t[j + 1] != s[i]) j = nxt[j];
29         if (t[j + 1] == s[i]) j++;
30         if (j == M) {
31             ans.push_back(i - M + 1);
32             j = nxt[j];
33         }
34     }
35 }

```

4 数学

4.1 数论

4.1.1 BSGS

```

1
2 struct HashTable {
3     static const int MOD = 1e7 + 10; // 此处 sqrt(p) 即可
4     struct edge {
5         int nxt;
6         ll num, val;
7     } e[MOD];
8     int head[MOD], tot;
9     void clear() { tot = 0; memset(head, 0, sizeof(head)); }
10    void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w}, head[u % MOD] = tot; }
11    int find(ll u) {
12        for (int i = head[u % MOD]; i; i = e[i].nxt)
13            if (e[i].num == u) return e[i].val;

```

```

14         return -1;
15     }
16 } hs;
17
18 ll qpow(ll a, ll b, ll mod) {
19     ll res = 1;
20     while (b) {
21         if (b & 1) res = res * a % mod;
22         a = a * a % mod;
23         b >>= 1;
24     }
25     return res;
26 }
27
28 ll BSGS(ll a, ll b, ll p) {//a ^ x = b (mod p)
29     //令x = i * m - j
30     //a ^ {i * m} = b * a ^ j (mod p) , j ∈ [0, m - 1]
31     b %= p;
32     if (a % p == 0 && b) return -1;
33     if (b == 1) return 0;
34     ll m = sqrt(p) + 1, base = qpow(a, m, p);
35     hs.clear();
36     //insert t = b * a ^ j to HashTable
37     for (ll i = 0, t = b; i < m; i++, t = t * a % p) hs.insert(t, i);
38     //t = a ^ {i * m}
39     for (ll i = 1, t = base; i <= m + 1; i++, t = t * base % p) {
40         ll j = hs.find(t);
41         if (j != -1) return i * m - j;
42     }
43     return -1;
44 }

```

4.1.2 CRT

```

1 ll exgcd(ll a, ll b, ll &x, ll &y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     ll res = exgcd(b, a % b, x, y);
7     ll t = y;
8     y = x - a / b * y;
9     x = t;
10    return res;
11 }
12
13 ll inv(ll a, ll b) {
14     ll x = 0, y = 0;
15     exgcd(a, b, x, y);
16     return x = (x % b + b) % b;
17 }
18
19 //r[]为余数, m为模数, 其中模数互质

```

```

20 //M = pi(mi), Mi = M / mi, invMi = Mi % mi
21 //ni满足是除了mi之外的倍数, 且模mi为ri
22 //利用逆元性质, 即ri * Mi * invMi = ri (mod mi)
23 //res = (sigma(ri * Mi * invMi)) % M
24
25 ll china(ll r[], ll m[], int n) {
26     ll M = 1, res = 0;
27     for (int i = 1; i <= n; i++) M *= m[i];
28     for (int i = 1; i <= n; i++) {
29         ll Mi = M / m[i], invMi = inv(Mi, m[i]);
30         res = (res + r[i] * Mi % M * invMi % M) % M;
31         //res = (res + mul(mul(r[i], Mi, M), invMi, M)) % M;按位乘
32     }
33     return (res % M + M) % M;
34 }

```

4.1.3 exBSGS

```

1 ll qpow(ll a, ll b, ll mod) {
2     ll res = 1;
3     while (b) {
4         if (b & 1) res = res * a % mod;
5         a = a * a % mod;
6         b >>= 1;
7     }
8     return res;
9 }
10
11 ll exgcd(ll a, ll b, ll &x, ll &y) {
12     if (!b) {
13         x = 1, y = 0;
14         return a;
15     }
16     ll res = exgcd(b, a % b, x, y);
17     ll t = y;
18     y = x - a / b * y;
19     x = t;
20     return res;
21 }
22
23 ll inv(ll a, ll b) {
24     ll x = 0, y = 0;
25     exgcd(a, b, x, y);
26     return x = (x % b + b) % b;
27 }
28
29 struct HashTable {
30     static const int MOD = 1e5 + 10;
31     struct edge {
32         int nxt;
33         ll num, val;
34     } e[MOD];
35     int head[MOD], tot;

```

```

36 void clear() { tot = 0; memset(head, 0, sizeof(head)); }
37 void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w }, head[u % MOD] = tot; }
38 int find(ll u) {
39     for (int i = head[u % MOD]; i; i = e[i].nxt)
40         if (e[i].num == u) return e[i].val;
41     return -1;
42 }
43 } hs;
44
45 ll BSGS(ll a, ll b, ll p) { //  $a^x = b \pmod p$ 
46     if (a % p == 0 && b) return -1;
47     if (b == 1) return 0;
48     ll m = sqrt(p) + 1, base = qpow(a, m, p);
49     hs.clear();
50     for (ll i = 0, t = b; i < m; i++, t = t * a % p) hs.insert(t, i);
51     for (ll i = 1, t = base; i <= m + 1; i++, t = t * base % p) {
52         ll j = hs.find(t);
53         if (j != -1) return i * m - j;
54     }
55     return -1;
56 }
57
58 ll exBSGS(ll a, ll b, ll p) {
59     if (b == 1 || p == 1) return 0; //  $b = 1$  ||  $(b = 0 \text{ \&\& } p = 1)$  的特殊情况
60     //  $b \% \gcd(a, p) \neq 0 \text{ \&\& } b \neq 1$  时方程无解
61     //  $a^x = b \pmod p$ 
62     //  $\Rightarrow a^{(x-1)} * (a / G) = (b / G) \pmod{(p / G)}$ 
63     //  $\Rightarrow a^{(x-1)} = (b / G * invg) \pmod{(p / G)}$ 
64     //  $\Rightarrow a^{x'} = b' \pmod{p'}$ 
65     ll G = __gcd(a, p), k = 0, g = 1;
66     while (G != 1) {
67         if (b % G) return -1;
68         k++, b /= G, p /= G, g = g * (a / G) % p;
69         if (g == b) return k; // 即  $a^{x'} = 1 \pmod{p'}$  时, 返回  $k$  即可
70         G = __gcd(a, p);
71     }
72     ll res = BSGS(a, b * inv(g, p) % p, p);
73     return res == -1 ? -1 : res + k;
74 }

```

4.1.4 exCRT

```

1 ll mul(ll a, ll b, ll mod) {
2     ll res = 0;
3     while (b) {
4         if (b & 1)
5             res = (res + a) % mod;
6         a = (a << 1) % mod;
7         b >>= 1;
8     }
9     return res;
10 }
11

```

```

12 ll exgcd(ll a, ll b, ll &x, ll &y) {
13     if (!b) {
14         x = 1, y = 0;
15         return a;
16     }
17     ll res = exgcd(b, a % b, x, y);
18     ll t = y;
19     y = x - a / b * y;
20     x = t;
21     return res;
22 }
23
24 ll excrt(ll r[], ll m[], int n) {
25     //模数m[i]不互质时用excrt
26     ll M = m[1], res = r[1];
27     for (int i = 2; i <= n; i++) {
28         ll a = M, b = m[i], c = (r[i] - res % m[i] + m[i]) % m[i], x = 0, y = 0;
29         ll g = exgcd(a, b, x, y);
30         if (c % g != 0) return -1; //c不能整除g那就无正整数解
31         x = mul(x, c / g, b / g);
32         res += x * M;
33         M *= b / g;
34         res = (res % M + M) % M;
35     }
36     return (res % M + M) % M;
37 }

```

4.1.5 exEuler

```

1 ll qpow(ll a, ll b, ll mod) {
2     ll res = 1;
3     while (b) {
4         if (b & 1) res = res * a % mod;
5         a = a * a % mod;
6         b >>= 1;
7     }
8     return res;
9 }
10
11 ll getPhi(ll x) {
12     ll res = 1;
13     for (ll i = 2; i * i <= x; i++)
14         if (x % i == 0) {
15             x = x / i;
16             res *= i - 1;
17             while (x % i == 0) x = x / i, res *= i;
18         }
19     if (x > 1) res *= x - 1;
20     return res;
21 }
22
23 ll exEuler(char *sa, char *sb, ll p) { //欧拉降幂求  $a^b \pmod p$ 
24     //gcd(a, p) = 1  $\Rightarrow a^b = a^{(b \% \phi(p))}$ 

```

```

25 //gcd(a, p) ≠ 1, b < p => a ^ b = a ^ b
26 //gcd(a, p) ≠ 1, b ≥ p => a ^ b = a ^ (b % phi(p) + phi(p))
27 int N = strlen(sa), M = strlen(sb), flag = 0;
28 ll phi = getPhi(p), a = 0, b = 0;
29 for (int i = 0; i < N; i++) a = (a * 10 + sa[i] - '0') % p;
30 for (int i = 0; i < M; i++) {
31     b = b * 10 + sb[i] - '0';
32     if (b >= phi) flag = 1;
33     b %= phi;
34 }
35 if (flag) b += phi;
36 return qpow(a, b, p);
37 }

```

4.1.6 exGCD

```

1 int exgcd(int a, int b, int &x, int &y) {
2     //算gcd的同时, 得到ax + by = gcd(a, b)的解(x, y)
3     if (!b) {
4         x = 1, y = 0;
5         return a;
6     }
7     int res = exgcd(b, a % b, x, y);
8     int t = y;
9     y = x - a / b * y;
10    x = t;
11    return res;
12 }

```

4.1.7 exLucas

```

1 ll qpow(ll a, ll b, ll mod) {
2     ll res = 1;
3     while (b) {
4         if (b & 1) res = res * a % mod;
5         a = a * a % mod;
6         b >>= 1;
7     }
8     return res;
9 }
10 ll exgcd(ll a, ll b, ll &x, ll &y) {
11     if (!b) {
12         x = 1, y = 0;
13         return a;
14     }
15     ll res = exgcd(b, a % b, x, y);
16     ll t = y;
17     y = x - a / b * y;
18     x = t;
19     return res;
20 }
21 ll inv(ll a, ll b) {
22     ll x = 0, y = 0;

```

```

23     exgcd(a, b, x, y);
24     return x = (x % b + b) % b;
25 }
26
27 ll f(ll n, ll p, ll pk) {
28     if (n == 0) return 1;
29     ll s = 1;
30     for (ll i = 1; i <= pk; i++)
31         if (i % p) s = s * i % pk;
32     s = qpow(s, n / pk, pk);
33     for (ll i = pk * (n / pk); i <= n; i++)
34         if (i % p) s = i % pk * s % pk;
35     return f(n / p, p, pk) * s % pk;
36 }
37
38 ll g(ll n, ll p) {
39     if (n < p) return 0;
40     return g(n / p, p) + (n / p);
41 }
42
43 ll c(ll n, ll m, ll p, ll pk) {
44     ll frac1 = f(n, p, pk), frac2 = inv(f(m, p, pk) * f(n - m, p, pk) % pk, pk);
45     ll s = qpow(p, g(n, p) - g(m, p) - g(n - m, p), pk);
46     return frac1 * frac2 % pk * s % pk;
47 }
48
49 ll calc(ll r, ll m, ll M) {
50     //CRT, M = p质因数分解后再全部相乘仍然是p
51     ll Mi = M / m, invMi = inv(Mi, m);
52     return r * Mi % M * invMi % M;
53 }
54
55 ll exLucas(ll n, ll m, ll p) { //O(pLogp), 不要求p是质数
56     ll t = p, res = 0;
57     for (ll i = 2; i * i <= p; i++)
58         if (t % i == 0) { //i为当前的质数
59             ll pk = 1; //p^k
60             while (t % i == 0) {
61                 t /= i, pk *= i;
62             }
63             res = (res + calc(c(n, m, i, pk), pk, p)) % p;
64         }
65     if (t > 1)
66         res = (res + calc(c(n, m, t, t), t, p)) % p;
67     return res;
68 }

```

4.1.8 Lucas

```

1 //求解C(n, m) (mod p), 其中p为素数且较小
2 //O(p+Logp)O(Logn)
3
4 ll qpow(ll a, ll b, ll mod) {

```



```

5     ll res = 1;
6     while (b) {
7         if (b & 1)
8             res = res * a % mod;
9         a = a * a % mod;
10        b >>= 1;
11    }
12    return res;
13 }
14
15 ll C(ll n, ll m, ll mod) {
16     if (n < m) return 0;
17     m = min(m, n - m);
18     ll a = 1, b = 1;
19     for (int i = 0; i < m; i++)
20         a = a * (n - i) % mod, b = b * (i + 1) % mod;
21     return a * qpow(b, mod - 2) % mod;
22 }
23
24 ll Lucas(ll n, ll m, ll mod) {
25     if (m == 0) return 1;
26     return Lucas(n / mod, m / mod, mod) * C(n % mod, m % mod, mod) % mod;
27 }

```

4.1.9 二次剩余

```

1 struct Complex {
2     ll x, y;
3 };
4 ll w;
5
6 Complex mul(Complex a, Complex b, ll mod) { //复数乘法
7     Complex ans = {0, 0};
8     ans.x = ((a.x * b.x % mod + a.y * b.y % mod * w % mod) % mod + mod) % mod;
9     ans.y = ((a.x * b.y % mod + a.y * b.x % mod) % mod + mod) % mod;
10    return ans;
11 }
12
13 ll binpow_real(ll a, ll b, ll mod) { //实部快速幂
14     ll ans = 1;
15     while (b) {
16         if (b & 1) ans = ans * a % mod;
17         a = a * a % mod;
18         b >>= 1;
19     }
20    return ans % mod;
21 }
22
23 ll binpow_imag(Complex a, ll b, ll mod) { //虚部快速幂
24     Complex ans = {1, 0};
25     while (b) {
26         if (b & 1) ans = mul(ans, a, mod);
27         a = mul(a, a, mod);

```

```

28     b >>= 1;
29 }
30 return ans.x % mod;
31 }
32
33 ll cipolla(ll n, ll mod) { // n = 0 外面特判
34     n %= mod;
35     if (mod == 2) return n;
36     if (binpow_real(n, (mod - 1) / 2, mod) == mod - 1) return -1;
37     ll a;
38     while (1) { // 生成随机数再检验找到满足非二次剩余的 a
39         a = rand() % mod;
40         w = ((a * a % mod - n) % mod + mod) % mod;
41         if (binpow_real(w, (mod - 1) / 2, mod) == mod - 1) break;
42     }
43     Complex x = {a, 1};
44     return binpow_imag(x, (mod + 1) / 2, mod);
45 }
46
47 int main() {
48
49     ll n, mod; scanf("%lld%lld", &n, &mod);
50     if (n == 0) {
51         printf("0\n");
52         continue;
53     }
54     ll ans1 = cipolla(n, mod);
55     if (ans1 == -1) printf("-1\n");
56     else {
57         ll ans2 = mod - ans1;
58         if (ans1 == ans2) printf("%lld\n", ans1);
59         else {
60             if (ans1 > ans2) swap(ans1, ans2);
61             printf("%lld %lld\n", ans1, ans2);
62         }
63     }
64
65     return 0;
66 }

```

4.1.10 反演相关

```

1  /*
2  莫比乌斯反演
3   $g[n] = \sum_{d|n} f[d]$ 
4   $f[d] = \sum_{d|n} g[d] * \mu[n/d]$ 
5
6  二项式反演
7   $g[n] = \sum_{i=1}^n C(n, i) * f[i]$ 
8   $f[n] = \sum_{i=1}^n C(n, i) * g[i] * (-1)^{n-i}$ 
9
10 子集反演
11  $f(S) = \sum_{T \in S} g(T)$ 

```

```

12  $g(S) = \sum_{T \in S} f(T) * (-1)^{|S| - |T|}$ 
13 */

```

4.1.11 常见积性函数

```

1 //phi
2 //phi[i * j] = phi[i] * phi[j] * gcd(i, j) / phi[gcd(i, j)]

```

4.1.12 整除分块

```

1
2 int calc(int n, int m) {
3     //sum_{i=1}^m n / i
4     //向下取整
5     for (int l = 1, r; l <= m; l = r + 1) {
6         if (n / l) r = min(m, n / (n / l));
7         else r = m;
8         //[l, r]之间的 n / l 都相等
9     }
10
11     //向上取整
12     for (int l = 1, r; l <= m; l = r + 1) {
13         int t = (n + l - 1) / l;
14         if (t == 1) r = m;
15         else r = min(m, (n - 1) / (t - 1));
16         //[l, r]之间的 (n + l - 1) / l 都相等
17     }
18
19 }

```

4.1.13 杜教筛

```

1
2
3
4 unordered_map<int, ll> smu, sphi;
5 bool isPrime[MAX];
6 int prime[MAX], num;
7 ll mu[MAX], phi[MAX];
8
9 void makeMobiusAndEuler(int siz) {
10     mu[1] = phi[1] = 1;
11     for (int i = 2; i <= siz; i++) {
12         if (!isPrime[i]) prime[++num] = i, mu[i] = -1, phi[i] = i - 1;
13         for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
14             isPrime[i * prime[j]] = 1;
15             if (i % prime[j] == 0) {
16                 mu[i * prime[j]] = 0;
17                 phi[i * prime[j]] = phi[i] * prime[j];
18                 break;
19             }
20             else {

```

```

21         phi[i * prime[j]] = phi[prime[j]] * phi[i];
22         mu[i * prime[j]] = -mu[i];
23     }
24 }
25 }
26 for (int i = 1; i <= siz; i++) mu[i] += mu[i - 1], phi[i] += phi[i - 1];
27 }
28
29 ll getSmu(int n) {
30     if (n <= N) return mu[n];
31     if (smu[n]) return smu[n];
32     ll res = 1;
33     for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
34         r = n / (n / l);
35         res -= 1ll * (r - l + 1) * getSmu(n / l);
36     }
37     return smu[n] = res;
38 }
39
40 ll getSphi(int n) {
41     if (n <= N) return phi[n];
42     if (sphi[n]) return sphi[n];
43     ll res = 1ll * n * (n + 1) / 2;
44     for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
45         r = n / (n / l);
46         res -= 1ll * (r - l + 1) * getSphi(n / l);
47     }
48     return sphi[n] = res;
49 }

```

4.1.14 筛 mobius

```

1 int vis[N], prime[N], num, mu[N];
2 void makeMobius(int siz) {
3     mu[1] = 1, num = 0;
4     for (int i = 2; i <= siz; i++) {
5         if (!vis[i]) prime[++num] = i, mu[i] = -1;
6         for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
7             vis[i * prime[j]] = 1;
8             if (i % prime[j] == 0) {
9                 mu[i * prime[j]] = 0;
10                break;
11            }
12            else mu[i * prime[j]] = mu[i] * mu[prime[j]];
13        }
14    }
15 }

```

4.1.15 筛 phi

```

1 int vis[N], prime[N], num, phi[N];
2 void makePhi(int siz) {
3     phi[1] = 1, num = 0;

```

```

4     for (int i = 2; i <= siz; i++) {
5         if (!vis[i]) prime[++num] = i, phi[i] = i - 1;
6         for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
7             vis[i * prime[j]] = 1;
8             if (i % prime[j] == 0) {
9                 phi[i * prime[j]] = phi[i] * prime[j];
10                break;
11            }
12            else phi[i * prime[j]] = phi[i] * phi[prime[j]];
13        }
14    }
15 }

```

4.1.16 筛积性函数

```

1 //只需要计算 $f(p^k)$ 即可
2 //其余的都可以通过积性函数的性质来计算
3
4 int vis[N], prime[N], num;
5 int f[N], low[N];
6
7 void makeF(int siz) { //f为积性函数
8     num = 0, low[1] = f[1] = 1;
9     for (int i = 2; i <= siz; i++) {
10        if (!vis[i]) prime[++num] = i, low[i] = i, f[i] = ...; //这里是 $f(p)$ 的答案
11        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
12            vis[i * prime[j]] = 1;
13            if (i % prime[j] == 0) {
14                low[i * prime[j]] = low[i] * prime[j];
15                if (low[i] == i) { //  $i = \text{prime}[j]^k$ 
16                    //只需要这里算一下
17                    //考虑  $p^1, p^2, p^3 \dots$ 
18                }
19                else f[i * prime[j]] = 111 * f[i / low[i]] * f[prime[j] * low[i]] % mod;
20                break;
21            }
22            low[i * prime[j]] = prime[j];
23            f[i * prime[j]] = 111 * f[i] * f[prime[j]] % mod;
24        }
25    }
26
27 }

```

4.1.17 线性同余方程

```

1 int exgcd(int a, int b, int &x, int &y) {
2     //算gcd的同时, 得到 $ax + by = \text{gcd}(a, b)$ 的解(x, y)
3     if (!b) {
4         x = 1, y = 0;
5         return a;
6     }
7     int res = exgcd(b, a % b, x, y);
8     int t = y;

```

```

9     y = x - a / b * y;
10    x = t;
11    return res;
12 }
13
14 bool solve(int a, int b, int c, int &x, int &y) {
15     //求  $ax + by = c$  的解( $x, y$ ), 有解的条件为  $c \mid (gcd(a, b))$ !!!!!!
16     //等价于求  $ax = b \pmod{c}$  的整数解 $x$ 
17     int d = exgcd(a, b, x, y);
18     if (c % d != 0) return false;
19     int k = c / d;
20     x *= k, y *= k;
21     /*求 $ax = b \pmod{c}$ 的最小正整数解
22     int t = b / __gcd(a, b);
23     x = (x % t + t) % t;*/
24     return true;
25 }

```

4.1.18 质因子分解

```

1  int vis[N], prime[N], num;
2  void makePrime(int siz) {
3      num = 0;
4      for (int i = 2; i <= siz; i++) {
5          if (!vis[i]) prime[++num] = i;
6          for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
7              vis[i * prime[j]] = 1;
8              if (i % prime[j] == 0) break;
9          }
10     }
11 }
12
13 void divide(ll x) {
14     for (int i = 1; i <= num && 1ll * prime[i] * prime[i] <= x; i++)
15         if (x % prime[i] == 0) {
16             int cnt = 0;
17             while (x % prime[i] == 0) x /= prime[i], cnt++;
18             store.push_back({prime[i], cnt});
19         }
20     if (x > 1) store.push_back({x, 1});
21 }

```

4.1.19 逆元

```

1  //存在逆元的充要条件为 $gcd(a, mod) = 1$ 
2  //小心那种 $mod = 2^k$ , 然后求 $inv(2^i)$ , 这种只能迭代求(可能为0)
3  //mod为质数, 费马小定理求解
4  #define inv(x,y) qpow(x,y-2,y)
5
6  ll qpow(ll a, ll b, ll mod) {
7      ll res = 1;
8      while (b) {
9          if (b & 1)

```

```

10         res = res * a % mod;
11         a = a * a % mod;
12         b >>= 1;
13     }
14     return res;
15 }
16
17 //mod不是质数, exgcd求解
18 //exgcd(a, b, x, y) -> b=mod, x为逆元
19 ll exgcd(ll a, ll b, ll &x, ll &y) {
20     if (!b) {
21         x = 1, y = 0;
22         return a;
23     }
24     ll res = exgcd(b, a % b, x, y);
25     ll t = y;
26     y = x - a / b * y;
27     x = t;
28     return res;
29 }
30
31 ll inv(ll a, ll b = mod) {
32     ll x = 0, y = 0;
33     exgcd(a, b, x, y);
34     return x = (x % b + b) % b;
35 }

```

4.1.20 龟速乘

```

1 //在a * a > LL, a * a % mod < LL下使用
2 ll mul(ll a, ll b, ll mod) {
3     ll res = 0;
4     while (b) {
5         if (b & 1)
6             res = (res + a) % mod;
7         a = (a << 1) % mod;
8         b >>= 1;
9     }
10    return res;
11 }
12
13 ll qpow(ll a, ll b, ll mod) {
14     ll res = 1;
15     while (b) {
16         if (b & 1)
17             res = mul(res, a, mod);
18         a = mul(a, a, mod);
19         b >>= 1;
20     }
21     return res;
22 }

```

4.2 线性代数

4.2.1 多项式

FFT

```
1 //F(x)=FL(x^2)+x*FR(x^2)
2 //F(W^k)=FL(w^k)+W^k*FR(w^k)
3 //F(W^{k+n/2})=FL(w^k)-W^k*FR(w^k)
4
5
6 const int N = 4e5 + 10;//4倍空间
7 const double PI = acos(-1);
8
9 struct Complex {
10     double a, b;
11     Complex(double a = 0, double b = 0): a(a), b(b) {}
12     Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs
        .b + b * rhs.a); }
13     Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
14     Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
15 };
16
17 int tr[N];
18
19 void FFT(Complex *A, int len, int type) {
20     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
21     for (int i = 2; i <= len; i <= 1) {//区间长度
22         int mid = i / 2;
23         Complex Wn(cos(2 * PI / i), type * sin(2 * PI / i));//单位根
24         for (int k = 0; k < len; k += i) {//每个子问题的起始点
25             Complex w(1, 0);//omega
26             for (int l = k; l < k + mid; l++) {
27                 Complex t = w * A[l + mid];
28                 A[l + mid] = A[l] - t;
29                 A[l] = A[l] + t;
30                 w = w * Wn;
31             }
32         }
33     }
34 }
35
36 void mul(Complex *a, Complex *b, int n) {
37     int len = 1; while (len <= n) len <= 1;
38     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
39     FFT(a, len, 1), FFT(b, len, 1);
40     for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
41     FFT(a, len, -1);
42     for (int i = 0; i < len; i++) a[i].a /= len;
43 }
44
45 Complex a[N], b[N];
46
47 int main() {
48
```



```

49     int n, m;
50     scanf("%d%d", &n, &m);
51     for (int i = 0; i <= n; i++) scanf("%lf", &a[i].a);
52     for (int i = 0; i <= m; i++) scanf("%lf", &b[i].a);
53
54     mul(a, b, n + m);
55     for (int i = 0; i <= n + m; i++)
56         printf("%d ", (int)(a[i].a + 0.5));
57
58
59     return 0;
60 }

```

FWT

```

1  const int mod = ...;
2  const int inv2 = ...;
3
4  int n;
5  int a[MAX], b[MAX];
6
7  inline void get() {
8      for (int i = 0; i < n; i++) a[i] = 1ll * a[i] * b[i] % mod;
9  }
10
11 inline void OR(int *f, int x = 1) {
12     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
13         for (int i = 0; i < n; i += o)
14             for (int j = 0; j < k; j++)
15                 f[i + j + k] = (f[i + j + k] + 1ll * f[i + j] * x % mod + mod) % mod;
16 }
17
18 inline void AND(int *f, int x = 1) {
19     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
20         for (int i = 0; i < n; i += o)
21             for (int j = 0; j < k; j++)
22                 f[i + j] = (f[i + j] + 1ll * f[i + j + k] * x % mod + mod) % mod;
23 }
24
25 inline void XOR(int *f, int x = 1) {
26     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
27         for (int i = 0; i < n; i += o)
28             for (int j = 0; j < k; j++) {
29                 ll a0 = f[i + j], a1 = f[i + j + k];
30                 f[i + j] = (a0 + a1) % mod * x % mod;
31                 f[i + j + k] = (a0 - a1 + mod) % mod * x % mod;
32             }
33 }
34
35 int main() {
36
37     FWT(a, 1); FWT(b, 1); get(); FWT(a, (OR + AND: -1, XOR: inv2));
38
39 }

```

NTT

```
1 //mod比最终答案要大
2 //如果中间乘法计算会爆LL, 换成按位乘
3 //精度高, 比3次FFT要快(复数运算=>整数运算)
4 //4倍空间
5
6 ll qpow(ll a, ll b, ll mod) {
7     ll res = 1;
8     while (b) {
9         if (b & 1) res = res * a % mod;
10        a = a * a % mod;
11        b >>= 1;
12    }
13    return res;
14 }
15
16 const ll mod = 998244353;
17 const ll G = 3;
18 const ll invG = qpow(G, mod - 2, mod);
19 int tr[N];
20
21 void NTT(ll *A, int len, int type) {
22     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
23     for (int i = 2; i <= len; i <= 1) {
24         int mid = i / 2;
25         ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
26         for (int k = 0; k < len; k += i) {
27             ll w = 1;
28             for (int l = k; l < k + mid; l++) {
29                 ll t = w * A[l + mid] % mod;
30                 A[l + mid] = (A[l] - t + mod) % mod;
31                 A[l] = (A[l] + t) % mod;
32                 w = w * Wn % mod;
33             }
34         }
35     }
36     if (type == -1) {
37         ll invn = qpow(len, mod - 2, mod);
38         for (int i = 0; i < len; i++)
39             A[i] = A[i] * invn % mod;
40     }
41 }
42
43 void mul(ll *a, ll *b, int n) {
44     int len = 1; while (len <= n) len <= 1;
45     for (int i = 0; i < len; i++) tr[i] = (tr[i] >> 1) >> 1 | (i & 1 ? len >> 1 : 0);
46     NTT(a, len, 1), NTT(b, len, 1);
47     for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
48     NTT(a, len, -1);
49 }
50
51 int n, m;
52 ll a[N], b[N];
53
```

```

54 int main() {
55
56     scanf("%d%d", &n, &m);
57     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
58     for (int i = 0; i <= m; i++) scanf("%lld", &b[i]);
59
60     mul(a, b, n + m);
61     for (int i = 0; i <= n + m; i++)
62         printf("%lld ", a[i]);
63
64     return 0;
65 }

```

任意模数 MTT

```

1 //将多项式拆成 $(a1 * mod + a2) * (b1 * mod + b2)$ 的形式
2 //=> $a1 * b1 * mod^2 + (a2 * b1 + a1 * b2) * mod + a2 * b2$ 
3 //在利用DFT合并、IDFT合并，最终只需要4次DFT即可
4 //精度 $10^{14}$ 
5 //4倍空间
6
7
8 const double PI = acos(-1);
9
10 struct Complex {
11     double x, y;
12     Complex(double a = 0, double b = 0): x(a), y(b) {}
13     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
14     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
15     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs
        .y + y * rhs.x); }
16     Complex conj() { return Complex(x, -y); }
17 } w[N];
18
19 ll mod;
20 int tr[N];
21 int a[N], b[N], ans[N];
22
23 void FFT(Complex *A, int len) {
24     for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
25     for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
26         for (int j = 0; j < len; j += i) {
27             Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
28             for (int k = 0; k < i >> 1; k++) {
29                 Complex tmp = *r * *p;
30                 *r = *l - tmp, *l = *l + tmp;
31                 ++l, ++r, p += lyc;
32             }
33         }
34 }
35
36 inline void MTT(int *x, int *y, int *z, int n) {
37     int len = 1; while (len <= n) len <<= 1;
38     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);

```

```

39     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI *
40         i / len));
41
42     for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
43     static Complex a[N], b[N];
44     static Complex dfta[N], dftb[N], dftc[N], dftd[N];
45
46     for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
47     for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
48     FFT(a, len), FFT(b, len);
49     for (int i = 0; i < len; i++) {
50         int j = (len - i) & (len - 1);
51         static Complex da, db, dc, dd;
52         da = (a[i] + a[j].conj()) * Complex(0.5, 0);
53         db = (a[i] - a[j].conj()) * Complex(0, -0.5);
54         dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
55         dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
56         dfta[j] = da * dc;
57         dftb[j] = da * dd;
58         dftc[j] = db * dc;
59         dftd[j] = db * dd;
60     }
61     for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
62     for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
63     FFT(a, len), FFT(b, len);
64     for (int i = 0; i < len; i++) {
65         int da = (11)(a[i].x / len + 0.5) % mod;
66         int db = (11)(a[i].y / len + 0.5) % mod;
67         int dc = (11)(b[i].x / len + 0.5) % mod;
68         int dd = (11)(b[i].y / len + 0.5) % mod;
69         z[i] = (da + ((11)(db + dc) << 15) + ((11)dd << 30)) % mod;
70     }
71 }
72
73
74 int main() {
75
76     int n, m;
77     scanf("%d%d%lld", &n, &m, &mod);
78     for (int i = 0; i <= n; i++) scanf("%d", &a[i]);
79     for (int i = 0; i <= m; i++) scanf("%d", &b[i]);
80
81     MTT(a, b, ans, n + m);
82     for (int i = 0; i <= n + m; i++)
83         printf("%s%d", i == 0 ? "" : " ", (ans[i] + mod) % mod);
84
85     return 0;
86 }

```

任意模数 NTT

- 1 //要求选取的三个模数 $mod1 * mod2 * mod3 \geq p^2 * n$
- 2 //优点是精度高, 可达 10^{26}

```

3 //缺点是常数大(9次NTT), 并且还使用了龟速乘
4 //4倍空间
5
6 ll qmul(ll a, ll b, ll mod) {
7     ll res = 0;
8     while (b) {
9         if (b & 1)
10             res = (res + a) % mod;
11         a = (a << 1) % mod;
12         b >>= 1;
13     }
14     return res;
15 }
16
17 ll qpow(ll a, ll b, ll mod) {
18     ll res = 1;
19     while (b) {
20         if (b & 1) res = qmul(res, a, mod);
21         a = qmul(a, a, mod);
22         b >>= 1;
23     }
24     return res;
25 }
26
27 const ll mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, mod4 = mod1 * mod2;
28 const ll G = 3;
29 ll a[3][MAX], b[3][MAX], ans[MAX], p;
30 int tr[MAX];
31
32 void NTT(ll *A, int len, int type, ll mod) {
33     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
34     for (int i = 2; i <= len; i <= 1) {
35         int mid = i / 2;
36         ll Wn = qpow(type == 1 ? G : qpow(G, mod - 2, mod), (mod - 1) / i, mod);
37         for (int k = 0; k < len; k += i) {
38             ll w = 1;
39             for (int l = k; l < k + mid; l++) {
40                 ll t = w * A[l + mid] % mod;
41                 A[l + mid] = (A[l] - t + mod) % mod;
42                 A[l] = (A[l] + t) % mod;
43                 w = w * Wn % mod;
44             }
45         }
46     }
47     if (type != 1) {
48         ll invn = qpow(len, mod - 2, mod);
49         for (int i = 0; i < len; i++) A[i] = A[i] * invn % mod;
50     }
51 }
52
53 void mul(int i, int len, ll mod) {
54     NTT(a[i], len, 1, mod), NTT(b[i], len, 1, mod);
55     for (int j = 0; j < len; j++) a[i][j] = a[i][j] * b[i][j] % mod;
56     NTT(a[i], len, -1, mod);

```

```

57 }
58
59 void CRT(int len) {
60     ll inv1 = qpow(mod2, mod1 - 2, mod1);
61     ll inv2 = qpow(mod1, mod2 - 2, mod2);
62     ll inv3 = qpow(mod4 % mod3, mod3 - 2, mod3);
63     for (int i = 0; i < len; i++) {
64         ll t = 0;
65         t = (t + qmul(a[0][i] * mod2 % mod4, inv1, mod4)) % mod4;
66         t = (t + qmul(a[1][i] * mod1 % mod4, inv2, mod4)) % mod4;
67         a[1][i] = t;
68         t = (a[2][i] - a[1][i] % mod3 + mod3) % mod3 * inv3 % mod3;
69         ans[i] = (mod4 % p * t % p + a[1][i] % p) % p;
70     }
71 }
72
73 void doNTT(int n) {
74     int len = 1; while (len <= n) len <<= 1;
75     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
76     mul(0, len, mod1), mul(1, len, mod2), mul(2, len, mod3);
77     CRT(len);
78 }
79
80 int main() {
81
82     int n, m;
83     scanf("%d%d%lld", &n, &m, &p);
84     for (int i = 0; i <= n; i++) {
85         ll x; scanf("%lld", &x);
86         a[0][i] = a[1][i] = a[2][i] = x % p;
87     }
88     for (int i = 0; i <= m; i++) {
89         ll x; scanf("%lld", &x);
90         b[0][i] = b[1][i] = b[2][i] = x % p;
91     }
92     doNTT(n + m);
93     for (int i = 0; i <= n + m; i++) printf("%lld ", ans[i]);
94
95     return 0;
96 }

```

任意模数多项式

```

1  const double PI = acos(-1);
2
3  ll qpow(ll a, ll b, ll mod) {
4      ll res = 1;
5      while (b) {
6          if (b & 1) res = res * a % mod;
7          a = a * a % mod;
8          b >>= 1;
9      }
10     return res;
11 }

```

```

12
13 struct Complex {
14     double x, y;
15     Complex(double a = 0, double b = 0): x(a), y(b) {}
16     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
17     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
18     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs
        .y + y * rhs.x); }
19     Complex conj() { return Complex(x, -y); }
20 } w[N];
21
22 ll mod;
23 int tr[N];
24
25 int getLen(int n) {
26     int len = 1; while (len < (n << 1)) len <<= 1;
27     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
28     for (int i = 0; i < len; i++) w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI *
        i / len));
29     return len;
30 }
31
32 void FFT(Complex *A, int len) {
33     for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
34     for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
35         for (int j = 0; j < len; j += i) {
36             Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
37             for (int k = 0; k < i >> 1; k++) {
38                 Complex tmp = *r * *p;
39                 *r = *l - tmp, *l = *l + tmp;
40                 ++l, ++r, p += lyc;
41             }
42         }
43 }
44
45 inline void MTT(ll *x, ll *y, ll *z, int len) {
46
47     for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
48     static Complex a[N], b[N];
49     static Complex dfta[N], dftb[N], dftc[N], dftd[N];
50
51     for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
52     for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
53     FFT(a, len), FFT(b, len);
54     for (int i = 0; i < len; i++) {
55         int j = (len - i) & (len - 1);
56         static Complex da, db, dc, dd;
57         da = (a[i] + a[j].conj()) * Complex(0.5, 0);
58         db = (a[i] - a[j].conj()) * Complex(0, -0.5);
59         dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
60         dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
61         dfta[j] = da * dc;
62         dftb[j] = da * dd;
63         dftc[j] = db * dc;

```

```

64     dftd[j] = db * dd;
65 }
66 for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
67 for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
68 FFT(a, len), FFT(b, len);
69 for (int i = 0; i < len; i++) {
70     ll da = (ll)(a[i].x / len + 0.5) % mod;
71     ll db = (ll)(a[i].y / len + 0.5) % mod;
72     ll dc = (ll)(b[i].x / len + 0.5) % mod;
73     ll dd = (ll)(b[i].y / len + 0.5) % mod;
74     z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
75 }
76 }
77
78 void getInv(ll *f, ll *g, int n) {
79     if (n == 1) { g[0] = qpow(f[0], mod - 2, mod); return; }
80     getInv(f, g, (n + 1) >> 1);
81     int len = getLen(n);
82     static ll c[N];
83     for (int i = 0; i < len; ++i) c[i] = i < n ? f[i] : 0;
84     MTT(c, g, c, len), MTT(c, g, c, len);
85     for (int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
86     for (int i = n; i < len; ++i) g[i] = 0;
87     for (int i = 0; i < len; i++) c[i] = 0;
88 }
89
90 void getDer(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i - 1] = f[i] * i %
    mod; g[len - 1] = 0; }
91
92 void getInt(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i] = f[i - 1] * qpow(i
    , mod - 2, mod) % mod; g[0] = 0; }
93
94 void getLn(ll *f, ll *g, int n) {
95     static ll a[N], b[N];
96     getDer(f, a, n);
97     getInv(f, b, n);
98     int len = getLen(n);
99     MTT(a, b, a, len);
100    getInt(a, g, len);
101    for (int i = n; i < len; i++) g[i] = 0;
102    for (int i = 0; i < len; i++) a[i] = b[i] = 0;
103 }
104
105
106 void getExp(ll *f, ll *g, int n) {
107     if (n == 1) return (void) (g[0] = 1);
108     getExp(f, g, (n + 1) >> 1);
109     static ll a[N];
110     getLn(g, a, n);
111     a[0] = (f[0] + 1 - a[0] + mod) % mod;
112     for (int i = 1; i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
113     int len = getLen(n);
114     MTT(a, g, g, len);
115     for (int i = n; i < len; i++) g[i] = 0;

```



```

116     for (int i = 0; i < len; i++) a[i] = 0;
117 }
118
119 void getPow(ll *f, ll *g, int n, ll k) {
120     static ll a[N];
121     getLn(f, a, n);
122     for (int i = 0; i < n; i++) a[i] = a[i] * k % mod;
123     getExp(a, g, n);
124     for (int i = 0, len = getLen(n); i < len; i++) a[i] = 0;
125 }
126
127 void fenZhiFFT(ll *f, ll *g, int n) {
128     //计算 $g[i] = \sum_{j=1}^i g[i-j] * f[j]$ 
129     static ll a[N];
130     for (int i = 1; i < n; i++) a[i] = (mod - f[i]) % mod;
131     a[0] = 1;
132     ll g0 = g[0];
133     getInv(a, g, n);
134     for (int i = 0; i < n; i++) g[i] = g[i] * g0 % mod, a[i] = 0;
135 }

```

其他

两次 FFT

```

1 //原理
2 //记 $P(x) = F(x) + G(x)i$ 
3 // $P(x)^2 = F(x)^2 - G(x)^2 + 2F(x)G(x)i$ 
4 //故虚部/2就可得到 $F(x)G(x)$ 
5
6 //快但是会丢失精度
7 //比如说 $F(x)$ 的系数均在 $[1e6, 1e5]$ 之间,  $G(x)$ 的系数均在 $[1e5, 1e6]$ 之间
8 //直接做FFT, 涉及的精度跨度上限是 $1e12$ 
9 //假如使用三次变两次优化, 由于平方项的存在, 涉及的精度跨度上限是 $1e24$ , 严重掉精度
10
11 const int MAX = 4e6 + 10;
12 const double PI = acos(-1);
13
14 struct Complex {
15     double a, b;
16     Complex(double a = 0, double b = 0): a(a), b(b) {}
17     Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs
        .b + b * rhs.a); }
18     Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
19     Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
20 };
21
22 int N, M;
23 int n, tr[MAX];
24 Complex a[MAX];
25
26 void FFT(Complex *A, int type) {
27     for (int i = 0; i < n; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
28     for (int len = 2; len <= n; len <<= 1) { //区间长度

```

```

29     int mid = len / 2;
30     Complex Wn(cos(2 * PI / len), type * sin(2 * PI / len)); //单位根
31     for (int k = 0; k < n; k += len) { //每个子问题的起始点
32         Complex w(1, 0); //omega
33         for (int l = k; l < k + mid; l++) {
34             Complex t = w * A[l + mid];
35             A[l + mid] = A[l] - t;
36             A[l] = A[l] + t;
37             w = w * Wn;
38         }
39     }
40 }
41
42 }
43
44 int main() {
45     scanf("%d%d", &N, &M);
46     for (int i = 0; i <= N; i++) scanf("%lf", &a[i].a);
47     for (int i = 0; i <= M; i++) scanf("%lf", &a[i].b); //虚部
48
49     n = 1; while (n <= N + M) n <<= 1;
50     for (int i = 0; i < n; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? n >> 1 : 0);
51
52     FFT(a, 1);
53     for (int i = 0; i < n; i++) a[i] = a[i] * a[i];
54
55     FFT(a, -1);
56
57     for (int i = 0; i <= N + M; i++)
58         printf("%d ", (int)(a[i].b / n / 2 + 0.5));
59
60
61     return 0;
62 }

```

递归无优化 FFT

```

1 struct Complex {
2     double a, b;
3     Complex(double a = 0, double b = 0): a(a), b(b) {}
4     Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs
        .b + b * rhs.a); }
5     Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
6     Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
7 } store[MAX << 1];
8
9 //递归版
10 void FFT(Complex *A, int len, int type) { //type(1, -1) = (DFT, IDFT)
11     if (len == 1) return;
12     Complex *fl = A, *fr = A + len / 2;
13     for (int k = 0; k < len; k++) store[k] = A[k];
14     for (int k = 0; k < len / 2; k++) fl[k] = store[k << 1], fr[k] = store[k << 1 | 1];
15     FFT(fl, len / 2, type), FFT(fr, len / 2, type);
16     Complex Wn = Complex(cos(2 * PI / len), type * sin(2 * PI / len));

```

```

17     Complex w = Complex(1, 0);
18     //Wn -> 单位根, w * I -> 下一个单位根
19     for (int k = 0; k < len / 2; k++) {
20         store[k] = fl[k] + w * fr[k];
21         store[k + len / 2] = fl[k] - w * fr[k];
22         w = w * Wn;
23     }
24     for (int k = 0; k < len; k++) A[k] = store[k];
25 }

```

原根表

1	mod				原根
2	r^{2^k+1}	r	k	g	
3	3	1	1	2	
4	5	1	2	2	
5	17	1	4	3	
6	97	3	5	5	
7	193	3	6	5	
8	257	1	8	3	
9	7681	15	9	17	
10	12289	3	12	11	
11	40961	5	13	3	
12	65537	1	16	3	
13	786433	3	18	10	
14	5767169	11	19	3	
15	7340033	7	20	3	
16	23068673	11	21	3	
17	104857601	25	22	3	
18	167772161	5	25	3	
19	469762049	7	26	3	
20	998244353	119	23	3	这个数常用
21	1004535809	479	21	3	加起来不会爆int
22	2013265921	15	27	31	
23	2281701377	17	27	3	这个数平方刚好不会爆11
24	3221225473	3	30	5	
25	75161927681	35	31	3	
26	77309411329	9	33	7	
27	206158430209	3	36	22	
28	2061584302081	15	37	7	
29	2748779069441	5	39	3	
30	6597069766657	3	41	5	
31	39582418599937	9	42	5	
32	79164837199873	9	43	5	
33	263882790666241	15	44	7	
34	1231453023109121	35	45	3	
35	1337006139375617	19	46	3	
36	3799912185593857	27	47	5	
37	4222124650659841	15	48	19	
38	7881299347898369	7	50	6	
39	31525197391593473	7	52	3	
40	180143985094819841	5	55	6	
41	1945555039024054273	27	56	5	
42	4179340454199820289	29	57	3	

多项式

```
1
2 typedef long long ll;
3 const int N = 1e6 + 10;
4
5 ll qpow(ll a, ll b, ll mod) {
6     ll res = 1;
7     while (b) {
8         if (b & 1) res = res * a % mod;
9         a = a * a % mod;
10        b >>= 1;
11    }
12    return res;
13 }
14
15 const ll mod = 998244353;
16 const ll G = 3;
17 const ll invG = qpow(G, mod - 2, mod);
18 const ll inv2 = qpow(2ll, mod - 2, mod);
19 int tr[N];
20
21 int getLen(int n) {
22     int len = 1;
23     while (len < (n << 1)) len <<= 1;
24     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
25     return len;
26 }
27
28 void NTT(ll *A, int len, int type) {
29     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
30     for (int i = 2; i <= len; i <<= 1) {
31         int mid = i / 2;
32         ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
33         for (int k = 0; k < len; k += i) {
34             ll w = 1;
35             for (int l = k; l < k + mid; l++) {
36                 ll t = w * A[l + mid] % mod;
37                 A[l + mid] = (A[l] - t + mod) % mod;
38                 A[l] = (A[l] + t) % mod;
39                 w = w * Wn % mod;
40             }
41         }
42     }
43     if (type == -1) {
44         ll invn = qpow(len, mod - 2, mod);
45         for (int i = 0; i < len; i++)
46             A[i] = A[i] * invn % mod;
47     }
48 }
49
50 void getDer(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i - 1] = f[i] * i %
    mod; g[len - 1] = 0; }
51
```

```

52 void getInt(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i] = f[i - 1] * qpow(i
    , mod - 2, mod) % mod; g[0] = 0; }
53
54 void getInv(ll *f, ll *g, int n) {
55     if (n == 1) return (void) (g[0] = qpow(f[0], mod - 2, mod));
56     getInv(f, g, (n + 1) >> 1);
57     int len = getLen(n);
58     static ll a[N];
59     for (int i = 0; i < len; ++i) a[i] = i < n ? f[i] : 0;
60     NTT(a, len, 1), NTT(g, len, 1);
61     for (int i = 0; i < len; i++)
62         g[i] = 1LL * (2 - 1LL * a[i] * g[i] % mod + mod) % mod * g[i] % mod;
63     NTT(g, len, -1);
64     for (int i = n; i < len; i++) g[i] = 0;
65 }
66
67 void getLn(ll *f, ll *g, int n) {
68     static ll a[N], b[N];
69     getDer(f, a, n);
70     getInv(f, b, n);
71     int len = getLen(n);
72     NTT(a, len, 1), NTT(b, len, 1);
73     for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
74     NTT(a, len, -1);
75     getInt(a, g, len);
76     for (int i = n; i < len; i++) g[i] = 0;
77     for (int i = 0; i < len; i++) a[i] = b[i] = 0;
78 }
79
80 void getExp(ll *f, ll *g, int n) {
81     if (n == 1) return (void) (g[0] = 1);
82     getExp(f, g, (n + 1) >> 1);
83     static ll a[N];
84     getLn(g, a, n);
85     a[0] = (f[0] + 1 - a[0] + mod) % mod;
86     for (int i = 1; i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
87     int len = getLen(n);
88     NTT(a, len, 1), NTT(g, len, 1);
89     for (int i = 0; i < len; i++) g[i] = g[i] * a[i] % mod;
90     NTT(g, len, -1);
91     for (int i = n; i < len; i++) g[i] = 0;
92     for (int i = 0; i < len; i++) a[i] = 0;
93 }
94
95 void getPow(ll *f, ll *g, int n, ll k) {
96     static ll a[N];
97     getLn(f, a, n);
98     for (int i = 0; i < n; i++) a[i] = a[i] * k % mod;
99     getExp(a, g, n);
100     for (int i = 0, len = getLen(n); i < len; i++) a[i] = 0;
101 }
102
103 void getPower(ll *f, ll *g, int n, ll k1, ll k2) { //k1为原始模数, k2为模phi(mod - 1)
104     int pos = 0; while (pos < n && !f[pos]) pos++;

```

```

105     if (k1 * pos >= n) { for (int i = 0; i < n; i++) g[i] = 0; return; }
106     static ll a[N], b[N];
107     int m = n - pos, inv = qpow(f[pos], mod - 2, mod), t = qpow(f[pos], k2, mod);
108     for (int i = 0; i < m; i++) a[i] = f[i + pos] * inv % mod;
109     getLn(a, b, m);
110     for (int i = 0; i < m; i++) b[i] = b[i] * k1 % mod;
111     getExp(b, g, m);
112     for (int i = 0; i < m; i++) g[i] = g[i] * t % mod;
113     pos = min(1ll * pos * k1, 1ll * n);
114     for (int i = n - 1; i >= pos; i--) g[i] = g[i - pos];
115     for (int i = pos - 1; i >= 0; i--) g[i] = 0;
116     for (int i = 0, len = getLen(m); i < len; i++) a[i] = b[i] = 0;
117 }
118
119 void fenziFFT(ll *f, ll *g, int n) {
120     //计算 $g[i] = \sum_{j=1}^i g[i-j] * f[j]$ 
121     static ll a[N];
122     for (int i = 1; i < n; i++) a[i] = (mod - f[i]) % mod;
123     a[0] = 1;
124     ll g0 = g[0];
125     getInv(a, g, n);
126     for (int i = 0; i < n; i++) g[i] = g[i] * g0 % mod, a[i] = 0;
127 }
128
129 struct Complex { ll x, y; }; ll w;
130 Complex mul(Complex a, Complex b, ll mod) {
131     Complex ans = {0, 0};
132     ans.x = ((a.x * b.x % mod + a.y * b.y % mod * w % mod) % mod + mod) % mod;
133     ans.y = ((a.x * b.y % mod + a.y * b.x % mod) % mod + mod) % mod;
134     return ans;
135 }
136 ll binpow_imag(Complex a, ll b, ll mod) {
137     Complex ans = {1, 0};
138     while (b) {
139         if (b & 1) ans = mul(ans, a, mod);
140         a = mul(a, a, mod);
141         b >>= 1;
142     }
143     return ans.x % mod;
144 }
145 ll cipolla(ll n, ll mod) {
146     srand(time(0));
147     n %= mod;
148     if (mod == 2) return n;
149     if (qpow(n, (mod - 1) / 2, mod) == mod - 1) return -1;
150     ll a;
151     while (1) {
152         a = rand() % mod;
153         w = ((a * a % mod - n) % mod + mod) % mod;
154         if (qpow(w, (mod - 1) / 2, mod) == mod - 1) break;
155     }
156     Complex x = {a, 1};
157     return binpow_imag(x, (mod + 1) / 2, mod);
158 }

```

```

159 void getSqrt(ll *f, ll *g, int n) {
160     if (n == 1) {
161         if (f[0] == 0) g[0] = f[0];
162         else {
163             ll t = cipolla(f[0], mod);
164             g[0] = min(t, mod - t);
165         }
166         return;
167     }
168     getSqrt(f, g, (n + 1) >> 1);
169     int len = getLen(n);
170     static ll a[N], b[N];
171     for (int i = 0; i < len; ++i) a[i] = i < n ? f[i] : 0;
172     getInv(g, b, n);
173     NTT(a, len, 1), NTT(b, len, 1);
174     for (int i = 0; i < len; ++i) a[i] = 1ll * a[i] * b[i] % mod;
175     NTT(a, len, -1);
176     for (int i = 0; i < n; ++i) g[i] = (g[i] + a[i]) % mod * inv2 % mod;
177     for (int i = n; i < len; ++i) g[i] = 0;
178     for (int i = 0; i < len; ++i) a[i] = b[i] = 0;
179 }
180
181 //分治乘法
182 void solve(ll *f1, ll *f2, ll *g, int l, int r) {
183     if (l == r) return (void) (g[(l - 1) * 2] = f1[l], g[(l - 1) * 2 + 1] = f2[l]);
184     int mid = (l + r) / 2;
185     solve(f1, f2, g, l, mid);
186     solve(f1, f2, g, mid + 1, r);
187     static ll a[N], b[N];
188     int len1 = mid - l + 2, len2 = r - mid + 1;
189     for (int i = 0; i < len1; ++i) a[i] = g[(l - 1) * 2 + i];
190     for (int i = 0; i < len2; ++i) b[i] = g[mid * 2 + i];
191     int n = r - l + 2, len = getLen(n);
192     NTT(a, len, 1), NTT(b, len, 1);
193     for (int i = 0; i < len; ++i) a[i] = a[i] * b[i] % mod;
194     NTT(a, len, -1);
195     for (int i = 0; i < n; ++i) g[(l - 1) * 2 + i] = a[i];
196     for (int i = n; i < len; ++i) g[(l - 1) * 2 + i] = 0;
197     for (int i = 0; i < len; ++i) a[i] = b[i] = 0;
198
199 }

```

多项式 2

```

1 namespace FFT {
2     #define mod 998244353
3     #define G 3
4     #define maxn 501000
5     typedef long long ll;
6     int bh[maxn], tmpA[maxn], tmpB[maxn], tmpC[maxn], tmp[maxn], lim = maxn, w[2][maxn];
7
8     int qpow(int a, int b) {
9         int ans = 1, tmp = a;
10        for (; b; b >>= 1, tmp = 1ll * tmp * tmp % mod)

```

```

11         if (b & 1)ans = 11l * ans * tmp % mod;
12     return ans;
13 }
14
15 int qpow(int a, int b, int p) {
16     int ans = 1, tmp = a;
17     for (; b; b >>= 1, tmp = 11l * tmp * tmp % p)
18         if (b & 1)ans = 11l * ans * tmp % p;
19     return ans;
20 }
21
22 void init(int b) {
23     for (int i = 1; i < (1 << b); i <= 1) {
24         int wn = qpow(G, (mod - 1) / (i << 1));
25         for (int j = 0; j < i; ++j)w[1][i + j] = (j ? 11l * wn * w[1][i + j - 1] % mod
: 1);
26         wn = qpow(G, mod - 1 - (mod - 1) / (i << 1));
27         for (int j = 0; j < i; ++j)w[0][i + j] = (j ? 11l * wn * w[0][i + j - 1] % mod
: 1);
28     }
29 }
30
31 ll gmod(ll x) {
32     return x >= mod ? x - mod : x;
33 }
34
35 void fft(int h[], int len, int flag) {
36     if (flag == -1)flag = 0;
37     for (int i = 0, j = 0; i < len; ++i) {
38         if (i < j)swap(h[i], h[j]);
39         for (int k = len >> 1; (j ^= k) < k; k >>= 1);
40     }
41     for (int i = 1; i < len; i <= 1)
42         for (int j = 0; j < len; j += (i << 1))
43             for (int k = 0; k < i; ++k) {
44                 int x = h[j + k], y = (ll) h[j + k + i] * w[flag][i + k] % mod;
45                 h[j + k] = gmod((ll) x + y);
46                 h[j + k + i] = gmod((ll) x - y + mod);
47             }
48     if (flag == 0) {
49         int x = qpow(len, mod - 2);
50         for (int i = 0; i < len; ++i)
51             h[i] = 11l * h[i] * x % mod;
52     }
53 }
54
55 void poly_mul(const int A[], int lenA, const int B[], int lenB, int C[], int lenc) {
56     int l = 1, k = 0, L = min(lenA, lenc) + min(lenB, lenc) + 1;
57     if (11l * lenA * lenB <= 400) {
58         for (int i = 0; i < L; ++i)tmpA[i] = 0;
59         for (int i = 0; i < lenA; ++i)
60             if (A[i])
61                 for (int j = 0; j < lenB; ++j)
62                     if (B[j])

```



```

63         tmpA[i + j] = (tmpA[i + j] + 111 * A[i] * B[j]) % mod;
64     for (int i = 0; i < lenc && i < L; ++i)C[i] = tmpA[i];
65     for (int i = L; i < lenc; ++i)C[i] = 0;
66     return;
67 }
68 while (l < L)l <= 1, k++;
69 memset(tmpA, 0, l << 2);
70 memset(tmpB, 0, l << 2);
71 memcpy(tmpA, A, min(lenA, lenc) << 2);
72 memcpy(tmpB, B, min(lenB, lenc) << 2);
73 fft(tmpA, l, 1), fft(tmpB, l, 1);
74 for (int i = 0; i < l; ++i)tmpA[i] = 111 * tmpA[i] * tmpB[i] % mod;
75 fft(tmpA, l, -1);
76 for (int i = 0; i < lenc && i < L; ++i)C[i] = tmpA[i];
77 for (int i = L; i < lenc; ++i)C[i] = 0;
78 }
79
80 void poly_inv(int n, const int a[], int C[]) {
81     if (n == 1) {
82         C[0] = qpow(a[0], mod - 2);
83         return;
84     }
85     static int A[maxn];
86     A[0] = qpow(a[0], mod - 2);
87     int m = (n + 1) / 2, len = 1, k = 0;
88     poly_inv(m, a, C);
89     while (len < n + n)len <= 1, k++;
90     memcpy(A, a, n << 2);
91     memset(A + n, 0, (len - n) << 2);
92     memset(C + m, 0, (len - m) << 2);
93     fft(A, len, 1), fft(C, len, 1);
94     for (int i = 0; i < len; ++i)C[i] = 111 * (211 - 111 * A[i] * C[i] % mod + mod) * C
[i] % mod;
95     fft(C, len, -1);
96 }
97
98 void poly_div(const int a[], int lena, const int b[], int lenb, int C[], int &plen) {
99     static int A[maxn], B[maxn];
100    while (!a[lena - 1] && lena >= 1)lena--;
101    while (!b[lenb - 1] && lenb >= 1)lenb--;
102    if (lena < lenb) {
103        plen = 1, C[0] = 0;
104        return;
105    }
106    plen = lena - lenb + 1;
107    memcpy(A, a, lena << 2);
108    memcpy(B, b, lenb << 2);
109    reverse(A, A + lena);
110    reverse(B, B + lenb);
111    poly_inv(plen, B, tmp);
112    poly_mul(tmp, plen, A, lena, C, plen);
113    reverse(C, C + plen);
114 }
115

```

```

116     typedef vector<int> Poly;
117
118     int SZ(const Poly &v) { return v.size(); }
119
120     void upd(Poly &v) { while (v.size() > 1 && !v.back())v.pop_back(); }
121
122     Poly operator*(const Poly &a, const Poly &b) {
123         Poly ret(SZ(a) + SZ(b) - 1);
124         poly_mul(a.data(), SZ(a), b.data(), SZ(b), ret.data(), SZ(ret));
125         upd(ret);
126         return ret;
127     }
128
129     Poly operator/(const Poly &a, const Poly &b) {
130         int len;
131         poly_div(a.data(), SZ(a), b.data(), SZ(b), tmp, len);
132         Poly ret = Poly(tmp, tmp + len);
133         upd(ret);
134         return ret;
135     }
136
137     Poly operator-(const Poly &a, const Poly &b) {
138         Poly
139         ret(max(SZ(a), SZ(b)));
140         for (int i = 0; i < SZ(ret); ++i)
141             ret[i] = (1ll * (i < SZ(a) ? a[i] : 0) - (i < SZ(b) ? b[i] : 0) + mod) % mod;
142         upd(ret);
143         return ret;
144     }
145
146     Poly operator+(const Poly &a, const Poly &b) {
147         Poly
148         ret(max(SZ(a), SZ(b)));
149         for (int i = 0; i < SZ(ret); ++i)
150             ret[i] = (1ll * (i < SZ(a) ? a[i] : 0) + (i < SZ(b) ? b[i] : 0)) % mod;
151         upd(ret);
152         return ret;
153     }
154
155     Poly operator%(const Poly &a, const Poly &b) {
156         int len;
157         poly_div(a.data(), SZ(a), b.data(), SZ(b), tmp, len);
158         poly_mul(b.data(), SZ(b), tmp, len, tmp, SZ(a));
159         for (int i = 0; i < SZ(a); ++i)
160             tmp[i] = (1ll * a[i] - tmp[i] + mod) % mod;
161         for (len = SZ(a); len > 1 && !tmp[len - 1]; len--);
162         return Poly(tmp, tmp + len);
163     }
164
165     void print(const Poly &x) {
166         printf("\n[len=%d]", SZ(x));
167         for (int i = 0; i < SZ(x); ++i)
168             printf("%d ", x[i]);
169         puts("");

```

```

170     }
171
172     Poly getinv(Poly g, int n) {
173         for (int i = 0; i < n; ++i) tmpC[i] = 0;
174         for (int i = 0; i < g.size() && i < n; ++i) tmpC[i] = g[i];
175         poly_inv(n, tmpC, tmp);
176         return Poly(tmp, tmp + n);
177     }
178
179     int getrt(int p) {
180         if (p == 2) return 1;
181         for (int i = 2; i < p; i++) {
182             int flg = 1, s = p - 1;
183             for (int j = 1; 1ll * j * j <= s && flg; j++)
184                 if (s % j == 0) {
185                     flg &= qpow(i, j, p) != 1;
186                     if (j != 1) flg &= qpow(i, s / j, p) != 1;
187                 }
188             if (flg) return i;
189         }
190         exit(1);
191     }
192
193 #undef maxn
194 #undef G
195 #undef mod
196 };

```

4.2.2 拉格朗日插值

```

1 //拉格朗日插值, 求点值转化为的多项式 $f(x)$ 
2 //得到 $f(k) = \sum_{i=1}^n y[i] \prod_{i \neq j} \frac{k - x[j]}{x[i] - x[j]}$ 
3
4 int lagrange(int n, int *x, int *y, int k) {
5     ll res = 0;
6     for (int i = 0; i < n; i++) {
7         ll s1 = 1, s2 = 1;
8         for (int j = 0; j < n; j++)
9             if (i != j) {
10                 s1 = s1 * (k - x[j] + mod) % mod;
11                 s2 = s2 * (x[i] - x[j] + mod) % mod;
12             }
13         res = (res + 1ll * y[i] * s1 % mod * inv(s2) % mod) % mod;
14     }
15     return (res + mod) % mod;
16 }

```

4.2.3 矩阵

循环矩阵快速幂优化

```

1 const int N = 5e2 + 10;
2 const ll mod = 998244353;
3

```

```

4  ll qpow(ll a, ll b) {
5      ll res = 1;
6      while (b) {
7          if (b & 1) res = res * a % mod;
8          a = a * a % mod;
9          b >>= 1;
10     }
11     return res;
12 }
13
14 void mul(ll a[], ll b[], int n) { // a = a * b
15     static ll res[N];
16     for (int i = 0; i < n; i++) res[i] = 0;
17     for (int i = 0; i < n; i++)
18         for (int j = 0; j < n; j++)
19             res[(i + j) % n] = (res[(i + j) % n] + 1ll * a[i] * b[j] % mod) % mod;
20     memcpy(a, res, sizeof(res));
21 }
22
23 void qpow(ll a[], int n, ll k) { // a = a^k
24     static ll res[N];
25     for (int i = 0; i < n; i++) res[i] = (i == 0);
26     while (k) {
27         if (k & 1) mul(res, a, n);
28         mul(a, a, n);
29         k >>= 1;
30     }
31     memcpy(a, res, sizeof(res));
32 }
33
34 ll x[N], A[N], ans[N];
35
36 int main() {
37
38     // x[i] = a * x[i - 1] + b * x[i] + c * x[i + 1]
39     A[n - 1] = a, A[0] = b, A[1] = c;
40     qpow(A, n, k);
41     // X = A^{k} * X
42     for (int i = 0; i < n; i++)
43         for (int j = 0; j < n; j++)
44             ans[j] = (ans[j] + 1ll * A[(j - i + n) % n] * x[i] % mod) % mod;
45     for (int i = 0; i < n; i++)
46         printf("%lld%s", ans[i], i == n - 1 ? "\n" : " ");
47
48     return 0;
49 }

```

矩阵光速幂

```

1  struct Matrix {
2      static const int N = 510;
3      int a[N][N], n;
4      Matrix(int siz) {
5          n = siz;

```

```

6         for (int i = 0; i < n; i++)
7             for (int j = 0; j < n; j++)
8                 a[i][j] = 0;
9     }
10    Matrix operator * (const Matrix& A) const { //重载矩阵乘法
11        Matrix res = Matrix(n);
12        for (int i = 0; i < n; i++)
13            for (int j = 0; j < n; j++)
14                for (int k = 0; k < n; k++)
15                    res.a[i][j] = (res.a[i][j] + 1ll * A.a[i][k] * a[k][j] % mod) % mod;
16        return res;
17    }
18 };
19
20 Matrix power(Matrix A, ll k) { //普通的快速幂
21     Matrix res = Matrix(A.n);
22     for (int i = 0; i < res.n; i++) res.a[i][i] = 1;
23     while (k) {
24         if (k & 1) res = res * A;
25         A = A * A;
26         k >>= 1;
27     }
28     return res;
29 }
30
31 Matrix power(Matrix A, string s) { //优化的快速幂
32     Matrix res = Matrix(A.n);
33     for (int i = 0; i < res.n; i++) res.a[i][i] = 1;
34     for (int i = s.length() - 1; i >= 0; --i) {
35         if (s[i] != '0') res = res * power(A, s[i] - '0');
36         A = power(A, 10);
37     }
38     return res;
39 }

```

4.2.4 矩阵求逆

```

1 //原始矩阵A[0, n - 1][0, n - 1]
2 //右边一个单位阵I, 在a[0, n - 1][n, (n << 1) - 1]
3 //将左边A变成单位阵时, 右边的I变为A-1
4
5 bool Gauss(ll a[][MAX << 1], int n) {
6     for (int i = 0, r; i < n; i++) {
7         r = i;
8         for (int j = i + 1; j < n; j++)
9             if (a[j][i] > a[r][i]) r = j;
10        if (r != i) swap(a[i], a[r]);
11        if (!a[i][i]) return false; //无解
12
13        ll inv = qpow(a[i][i], mod - 2);
14        for (int k = 0; k < n; k++) {
15            if (k == i) continue;
16            ll t = a[k][i] * inv % mod;

```

```

17         for (int j = i; j < (n << 1); j++)
18             a[k][j] = (a[k][j] - t * a[i][j] % mod + mod) % mod;
19     }
20     for (int j = 0; j < (n << 1); j++) a[i][j] = a[i][j] * inv % mod;
21 }
22 return true;
23 }
24
25 int main() {
26
27
28     scanf("%d", &n);
29     for (int i = 0; i < n; i++) {
30         a[i][i + n] = 1;
31         for (int j = 0; j < n; j++)
32             scanf("%lld", &a[i][j]);
33     }
34
35 }

```

4.2.5 高斯消元

```

1 bool gauss(double a[][MAX], int n) {
2     //a[][]为增广矩阵[0, n) x [0, n)
3     //最后解在a[][n]中
4     int i, j, k, r;
5     for (i = 0; i < n; i++) {
6         r = i;
7         for (j = i + 1; j < n; j++)
8             if (abs(a[j][i]) > abs(a[r][i])) r = j;
9         if (abs(a[r][i]) < eps) return 0; //无穷多解
10        if (r != i) {
11            for (j = 0; j <= n; j++)
12                swap(a[r][j], a[i][j]);
13        }
14        for (j = n; j >= i; j--)
15            for (k = i + 1; k < n; k++)
16                a[k][j] -= a[k][i] / a[i][i] * a[i][j];
17    }
18    for (i = n - 1; i >= 0; i--) {
19        for (j = i + 1; j < n; j++)
20            a[i][n] -= a[j][n] * a[i][j];
21        a[i][n] /= a[i][i];
22    }
23    return 1; //唯一解
24 }

```

4.3 组合数学

4.3.1 Lucas

```

1 ll qpow(ll a, ll b) {
2     ll res = 1;

```

```

3     while (b) {
4         if (b & 1)
5             res = res * a % mod;
6         a = a * a % mod;
7         b >>= 1;
8     }
9     return res;
10 }
11
12 ll fac[N];
13 void init(int siz) {
14     fac[0] = 1;
15     for (int i = 1; i <= siz; i++) fac[i] = i * fac[i - 1] % mod;
16 }
17
18 ll C(ll n, ll m) {
19     if (n < m) return 0;
20     return fac[n] * qpow(fac[m], mod - 2) % mod * qpow(fac[n - m], mod - 2) % mod;
21 }
22
23 ll Lucas(ll n, ll m) {
24     if (m == 0) return 1;
25     return Lucas(n / mod, m / mod) * C(n % mod, m % mod) % mod;
26 }

```

4.3.2 二项式相关

```

1  /*
2  广义二项式:
3   $(1 + x)^{-k} = \sum_{i=0}^{\infty} C(k + i - 1, i) * x^i * (-1)^i$ 
4   $(1 - x)^{-k} = \sum_{i=0}^{\infty} C(k + i - 1, i) * x^i$ 
5  */

```

4.3.3 卡特兰数

```

1  //卡特兰数有以下几种形式:
2  //
3  //
4  //
5
6  int catalan[N];
7  void getCatalan(int siz) {
8     catalan[0] = catalan[1] = 1;
9     for (int i = 2; i <= siz; i++) {
10         //catalan[i] = catalan[i - 1] * (4 * i - 2) % mod * qpow(i + 1, mod - 2) % mod;
11         catalan[i] = (C(2 * i, i) - C(2 * i, i - 1) + mod) % mod;
12     }
13 }

```

4.3.4 容斥原理

```

1  /*
2  集合容斥
3   $f(S) = \sum_{T \in S} g(T)$ 
4   $g(S) = \sum_{T \in S} f(T) * (-1)^{|S| - |T|}$ 
5
6  Min-Max容斥
7   $max(S) = \sum_{T \in S} min(T) * (-1)^{|T| - 1}$ 
8   $min(S) = \sum_{T \in S} max(T) * (-1)^{|T| - 1}$ 
9  第k大, 同理有第k小
10  $max\_k(S) = \sum_{T \in S} min(T) * (-1)^{|T| - k} * C(|T| - 1, k - 1)$ 
11 对期望也适用
12  $E[max(S)] = \sum_{T \in S} E[min(T)] * (-1)^{|T| - 1}$ 
13 */

```

4.3.5 第二类斯特林数

```

1 //S(n, m) = ifac[m] * \sum{k = 0}^m (-1)^k * C(m, k) * (m - k)^n
2
3 ll S(ll n, ll m) {
4     ll res = 0;
5     for (int k = 0, sign = 1; k <= m; k++, sign *= -1)
6         res = (res + sign * C(m, k) * qpow(m - k, n) % mod) % mod;
7     res = (res + mod) % mod;
8     return res * ifac[m] % mod;
9 }

```

4.3.6 组合数

```

1 ll C(ll n, ll m) {
2     if (m == 0 || n == m) return 1;
3     if (m > n) return 0;
4     if (m == 1) return n;
5     if (c[n][m]) return c[n][m];
6     else return c[n][m] = C(n - 1, m) + C(n - 1, m - 1);
7 }
8
9
10 //预处理版本
11 ll qpow(ll a, ll b) {
12     ll res = 1;
13     while (b) {
14         if (b & 1) res = res * a % mod;
15         a = a * a % mod;
16         b >>= 1;
17     }
18     return res;
19 }
20
21 ll fac[N], ifac[N];
22 void init(int siz) {
23     fac[0] = 1;
24     for (int i = 1; i <= siz; i++)
25         fac[i] = i * fac[i - 1] % mod;

```



```

26     ifac[siz] = qpow(fac[siz], mod - 2);
27     for (int i = siz; i >= 1; i--) ifac[i - 1] = ifac[i] * i % mod;
28 }
29
30 ll C(ll n, ll m) {
31     if (m == 0 || n == m) return 1;
32     if (m > n) return 0;
33     if (m == 1) return n;
34     return fac[n] * ifac[m] % mod * ifac[n - m] % mod;
35 }

```

4.4 计算几何

4.4.1 凸包

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int maxn = 1005;
8
9  struct point
10 {
11     double x, y;
12 }plist[maxn];
13 int pstack[maxn], top;
14
15 double cross(const point &p0, const point &p1, const point&p2)
16 {
17     return (p1.x - p0.x)*(p2.y - p0.y) - (p2.x - p0.x)*(p1.y - p0.y);
18 }
19
20 double dis(const point &p1, const point &p2)
21 {
22     return sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
23 }
24
25 bool cmp(const point &p1, const point &p2)
26 {
27     double tmp = cross(plist[0], p1, p2);
28     if (tmp > 0.0)return true;
29     else if (tmp == 0.0 && dis(plist[0], p1) < dis(plist[0], p2))return true;
30     else return false;
31 }
32
33
34 void Graham(int n)
35 {
36
37     top = 1;
38     pstack[0] = 0;

```

```

39     pstack[1] = 1;
40     for (int i = 2; i < n; i++) {
41         while (top > 0 && cross(plist[pstack[top - 1]], plist[pstack[top]], plist[i]) <= 0)
            top--;
42         pstack[++top] = i;
43     }
44 }
45
46
47 int main()
48 {
49     int n;
50     double res;
51     while (scanf("%d", &n) != EOF && n) {
52         res = 0;
53         int k = 0;
54         scanf("%lf %lf", &plist[0].x, &plist[0].y);
55         point p0 = plist[0];
56         for (int i = 1; i < n; i++) {
57             scanf("%lf %lf", &plist[i].x, &plist[i].y);
58             if (p0.y > plist[i].y || (p0.y == plist[i].y && p0.x > plist[i].x)) {
59                 p0.x = plist[i].x;
60                 p0.y = plist[i].y;
61                 k = i;
62             }
63         }
64
65         plist[k] = plist[0];
66         plist[0] = p0;
67
68         if (n == 1)
69         {
70             printf("0.00\n");
71             continue;
72         }
73         else if (n == 2)
74         {
75             printf("%.2lf\n", dis(plist[0], plist[1]));
76             continue;
77         }
78
79         sort(plist + 1, plist + n, cmp);
80         Graham(n);
81         for (int i = 1; i <= top; i++) {
82             res += dis(plist[pstack[i - 1]], plist[pstack[i]]);
83         }
84         res += dis(plist[pstack[top]], plist[0]);
85         printf("%.2lf\n", res);
86     }
87     return 0;
88 }

```

5 数据结构

5.1 01Trie

```
1 struct Trie {
2     static const int MAX_N = N * 35, rt = 0;
3     static const int MAX_BIT = 31;
4     int ch[MAX_N][2], sum[MAX_N], tot;
5     void init() { tot = 0; ch[tot][0] = ch[tot][1] = sum[tot] = 0; }
6     void insert(ll val) {
7         int u = rt;
8         for (int bit = MAX_BIT; bit >= 0; bit--) {
9             int s = (val >> bit) & 1; sum[u]++;
10            if (!ch[u][s]) {
11                ch[u][s] = ++tot;
12                ch[tot][0] = ch[tot][1] = sum[tot] = 0;
13            }
14            u = ch[u][s];
15        }
16        sum[u]++;
17    }
18    ll queryKth(ll val, int k = 1) { //第k大
19        int u = rt; ll res = 0;
20        for (int bit = MAX_BIT; bit >= 0; bit--) {
21            int s = (val >> bit) & 1;
22            if (!ch[u][s] && !ch[u][s ^ 1]) return res;
23            if (!ch[u][s ^ 1]) u = ch[u][s];
24            else if (sum[ch[u][s ^ 1]] >= k) u = ch[u][s ^ 1], res += (1ll << bit);
25            else k -= sum[ch[u][s ^ 1]], u = ch[u][s];
26        }
27        return res;
28    }
29 } trie;
```

5.2 BIT

```
1 #define lowbit(x) x&-x
2 int BIT[N];
3 void upd(int p, int k) { for (; p < N; p += lowbit(p)) BIT[p] += k; }
4 int ask(int p) { int res = 0; for (; p; p -= lowbit(p)) res += BIT[p]; return res; }
```

5.3 HashTable

```
1 struct HashTable {
2     static const int MOD = 1e7 + 10;
3     struct edge {
4         int nxt;
5         ll num, val;
6     } e[MOD];
7     int head[MOD], tot;
8     void clear() { tot = 0; memset(head, 0, sizeof(head)); }
9     void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w }, head[u % MOD] = tot; }
```

```

10     int find(ll u) {
11         for (int i = head[u % MOD]; i; i = e[i].nxt)
12             if (e[i].num == u) return e[i].val;
13         return -1;
14     }
15 } hs;

```

5.4 K-D_Tree

```

1  //查包含在x1,y1,x2,y2为左下角和右上角的矩形里面权值之和
2  //K-D Tree 二维划分树
3  int N, ans, rt, WD, tot, top, rub[MAX];
4
5  struct node {
6      int x[2], w;
7  } p[MAX];
8
9  struct K_D_tree {
10     int ls, rs, siz, mn[2], mx[2], sum;
11     //mn[0], mx[0] -> x的取值范围
12     //mn[1], mx[1] -> y的取值范围
13     node tmp;
14 } t[MAX];
15
16 int operator < (const node &a, const node &b) { return a.x[WD] < b.x[WD]; }
17
18 int newnode() {
19     if (top) return rub[top--];
20     else return ++tot;
21 }
22
23 void push_up(int u) {
24     for (int i = 0; i <= 1; i++) { //更新x, y的取值范围
25         t[u].mn[i] = t[u].mx[i] = t[u].tmp.x[i];
26         if (lc) { //左子树的最大最小值
27             t[u].mn[i] = min(t[u].mn[i], t[lc].mn[i]);
28             t[u].mx[i] = max(t[u].mx[i], t[lc].mx[i]);
29         }
30         if (rc) { //右子树的最大最小值
31             t[u].mn[i] = min(t[u].mn[i], t[rc].mn[i]);
32             t[u].mx[i] = max(t[u].mx[i], t[rc].mx[i]);
33         }
34     }
35     t[u].sum = t[lc].sum + t[rc].sum + t[u].tmp.w;
36     t[u].siz = t[lc].siz + t[rc].siz + 1;
37 }
38
39 int build(int l, int r, int wd) {
40     if (l > r) return 0;
41     int u = newnode();
42     WD = wd; nth_element(p + l, p + m, p + r + 1);
43     t[u].tmp = p[m];
44     t[u].ls = build(l, m - 1, wd ^ 1);

```

```

45     t[u].rs = build(m + 1, r, wd ^ 1);
46     push_up(u);
47     return u;
48 }
49
50 void pia(int u, int num) { //拍扁回炉重做
51     if (lc) pia(lc, num);
52     p[t[lc].siz + num + 1] = t[u].tmp, rub[++top] = u;
53     if (rc) pia(rc, t[lc].siz + num + 1);
54 }
55
56 void check(int &u, int wd) { //检查是否平衡, 不平衡则需要重建
57     if (t[u].siz * alpha < t[lc].siz || t[u].siz * alpha < t[rc].siz) pia(u, 0), u = build
        (1, t[u].siz, wd);
58 }
59
60 void insert(int &u, node tp, int wd) { //插入点
61     if (!u) {
62         u = newnode();
63         lc = rc = 0, t[u].tmp = tp;
64         push_up(u);
65         return;
66     }
67     if (tp.x[wd] < t[u].tmp.x[wd]) insert(lc, tp, wd ^ 1);
68     else insert(rc, tp, wd ^ 1);
69     push_up(u);
70     check(u, wd);
71 }
72
73 bool in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) { //完全被包含
74     return (x1 <= X1 && X2 <= x2 && y1 <= Y1 && Y2 <= y2);
75 }
76
77 bool out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) { //完全无交集
78     return (x1 > X2 || x2 < X1 || y1 > Y2 || y2 < Y1);
79 }
80
81 int query(int u, int x1, int y1, int x2, int y2) {
82     if (!u) return 0;
83     int res = 0;
84     if (in(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return t[u].sum
        ;
85     if (out(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return 0;
86     if (in(x1, y1, x2, y2, t[u].tmp.x[0], t[u].tmp.x[1], t[u].tmp.x[0], t[u].tmp.x[1])) res
        += t[u].tmp.w;
87     res += query(lc, x1, y1, x2, y2) + query(rc, x1, y1, x2, y2);
88     return res;
89 }
90
91 void init() {
92     ans = rt = top = tot = 0;
93 }
94
95 int main() {

```

```

96
97     init();
98     scanf("%d", &N);
99     while (1) {
100         int op; scanf("%d", &op);
101         if (op == 3) break;
102         if (op == 2) {
103             int x1, y1, x2, y2; scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
104             printf("%d\n", ans = query(rt, x1 ^ ans, y1 ^ ans, x2 ^ ans, y2 ^ ans));
105         }
106         else {
107             int x, y, w; scanf("%d%d%d", &x, &y, &w);
108             insert(rt, node{x ^ ans, y ^ ans, w ^ ans}, 0);
109         }
110     }
111
112
113     return 0;
114 }

```

5.5 Scapegoat

```

1 namespace Scapegoat_Tree {
2 #define MAXN (100000 + 10)
3     const double alpha = 0.75;
4     struct Node {
5         Node * ch[2];
6         int key, size, cover; // size为有效节点的数量, cover为节点总数量
7         bool exist; // 是否存在 (即是否被删除)
8         void PushUp(void) {
9             size = ch[0]—>size + ch[1]—>size + (int)exist;
10            cover = ch[0]—>cover + ch[1]—>cover + 1;
11        }
12        bool isBad(void) { // 判断是否需要重构
13            return ((ch[0]—>cover > cover * alpha + 5) ||
14                (ch[1]—>cover > cover * alpha + 5));
15        }
16    };
17    struct STree {
18    protected:
19        Node mem_pool[MAXN]; //内存池, 直接分配好避免动态分配内存占用时间
20        Node *tail, *root, *null; // 用null表示NULL的指针更方便, tail为内存分配指针, root为根
21        Node *bc[MAXN]; int bc_top; // 储存被删除的节点的内存地址, 分配时可以再利用这些地址
22
23        Node * NewNode(int key) {
24            Node * p = bc_top ? bc[--bc_top] : tail++;
25            p—>ch[0] = p—>ch[1] = null;
26            p—>size = p—>cover = 1; p—>exist = true;
27            p—>key = key;
28            return p;
29        }
30        void Travel(Node * p, vector<Node *>&v) {
31            if (p == null) return;

```

```

32         Travel(p->ch[0], v);
33         if (p->exist) v.push_back(p); // 构建序列
34         else bc[bc_top++] = p; // 回收
35         Travel(p->ch[1], v);
36     }
37     Node * Divide(vector<Node *>&v, int l, int r) {
38         if (l >= r) return null;
39         int mid = (l + r) >> 1;
40         Node * p = v[mid];
41         p->ch[0] = Divide(v, l, mid);
42         p->ch[1] = Divide(v, mid + 1, r);
43         p->PushUp(); // 自底向上维护, 先维护子树
44         return p;
45     }
46     void Rebuild(Node * &p) {
47         static vector<Node *>v; v.clear();
48         Travel(p, v); p = Divide(v, 0, v.size());
49     }
50     Node ** Insert(Node *p, int val) {
51         if (p == null) {
52             p = NewNode(val);
53             return &null;
54         }
55         else {
56             p->size++; p->cover++;
57
58             // 返回值储存需要重构的位置, 若子树也需要重构, 本节点开始也需要重构, 以本节点为根
59             Node ** res = Insert(p->ch[val >= p->key], val);
60             if (p->isBad()) res = &p;
61             return res;
62         }
63     }
64     void Erase(Node *p, int id) {
65         p->size--;
66         int offset = p->ch[0]->size + p->exist;
67         if (p->exist && id == offset) {
68             p->exist = false;
69             return;
70         }
71         else {
72             if (id <= offset) Erase(p->ch[0], id);
73             else Erase(p->ch[1], id - offset);
74         }
75     }
76     public:
77         void Init(void) {
78             tail = mem_poor;
79             null = tail++;
80             null->ch[0] = null->ch[1] = null;
81             null->cover = null->size = null->key = 0;
82             root = null; bc_top = 0;
83         }
84     STree(void) { Init(); }

```

重构

```

85
86     void Insert(int val) {
87         Node ** p = Insert(root, val);
88         if (*p != null) Rebuild(*p);
89     }
90     int Rank(int val) {
91         Node * now = root;
92         int ans = 1;
93         while (now != null) { // 非递归求排名
94             if (now->key >= val) now = now->ch[0];
95             else {
96                 ans += now->ch[0]->size + now->exist;
97                 now = now->ch[1];
98             }
99         }
100         return ans;
101     }
102     int Kth(int k) {
103         Node * now = root;
104         while (now != null) { // 非递归求第K大
105             if (now->ch[0]->size + 1 == k && now->exist) return now->key;
106             else if (now->ch[0]->size >= k) now = now->ch[0];
107             else k -= now->ch[0]->size + now->exist, now = now->ch[1];
108         }
109     }
110     void Erase(int k) {
111         Erase(root, Rank(k));
112         if (root->size < alpha * root->cover) Rebuild(root);
113     }
114     void Erase_kth(int k) {
115         Erase(root, k);
116         if (root->size < alpha * root->cover) Rebuild(root);
117     }
118 };
119 #undef MAXN
120
121 }

```

5.6 Splay

```

1 struct Splay {
2     #define rt ch[0][1]
3     #define lc ch[u][0]
4     #define rc ch[u][1]
5     #define identify(u) (ch[fa[u]][1] == u)
6     int tot, totElement;
7     int val[N], cnt[N], fa[N], sum[N];
8     int ch[N][2];
9     Splay() { tot = totElement = 0; }
10    void push_up(int u) { sum[u] = sum[lc] + sum[rc] + cnt[u]; }
11    void connect(int u, int par, int son) { ch[par][son] = u, fa[u] = par; }
12    void rotate(int u) {
13        int fc = identify(u), f = fa[u];

```



```

14     int gc = identify(f), g = fa[f];
15     int uc = fc ^ 1, son = ch[u][uc];
16     connect(son, f, fc);
17     connect(f, u, uc);
18     connect(u, g, gc);
19     push_up(f);
20     push_up(u);
21 }
22 void splay(int u, int v) {
23     v = fa[v];
24     while (fa[u] != v) {
25         int f = fa[u];
26         if (fa[f] != v)
27             rotate(identify(u) ^ identify(f) ? u : f);
28         rotate(u);
29     }
30 }
31 int creat(int v, int par) {
32     val[++tot] = v;
33     fa[tot] = par;
34     sum[tot] = cnt[tot] = 1;
35     return tot;
36 }
37 void destory(int u) {
38     val[u] = cnt[u] = fa[u] = sum[u] = lc = rc = 0;
39     tot -= u == tot;
40 }
41 int find(int v) {
42     int u = rt;
43     while (1) {
44         if (val[u] == v) {
45             splay(u, rt);
46             return u;
47         }
48         int nxt = v > val[u];
49         if (!ch[u][nxt]) return -1;
50         u = ch[u][nxt];
51     }
52 }
53 int insert(int v) {
54     totElement++;
55     if (totElement == 1) {
56         creat(v, 0);
57         return rt = tot;
58     }
59     int u = rt;
60     while (1) {
61         sum[u]++;
62         if (v == val[u]) {
63             cnt[u]++;
64             return u;
65         }
66         int nxt = v > val[u];
67         if (!ch[u][nxt]) {

```

```

68         creat(v, u);
69         splay(ch[u][nxt] = tot, rt);
70         return tot;
71     }
72     u = ch[u][nxt];
73 }
74 }
75 void remove(int v) {
76     int u = find(v);
77     if (!u) return;
78     totElement--;
79     if (cnt[u] > 1) {
80         cnt[u]--, sum[u]--;
81         return;
82     }
83     if (!lc) fa[rt = rc] = 0;
84     else {
85         int now = lc;
86         while (ch[now][1]) now = ch[now][1];
87         splay(now, lc);
88         connect(rc, now, 1); connect(now, 0, 1);
89         push_up(now);
90     }
91     destory(u);
92 }
93 int getRank(int v) {
94     int k = 0, u = rt;
95     while (1) {
96         if (v == val[u]) {
97             k += sum[lc] + 1;
98             splay(u, rt);
99             return k;
100         }
101         else if (v < val[u]) u = lc;
102         else {
103             k += sum[lc] + cnt[u];
104             u = rc;
105         }
106         if (!u) return 0;
107     }
108 }
109 int atRank(int k) {
110     if (k > totElement) return -1;
111     int u = rt;
112     while (1) {
113         if (k > sum[lc] && k <= sum[lc] + cnt[u]) {
114             splay(u, rt);
115             break;
116         }
117         if (k <= sum[lc]) u = lc;
118         else {
119             k -= sum[lc] + cnt[u];
120             u = rc;
121         }

```

```

122     }
123     return val[u];
124 }
125 int upper(int v) {
126     int u = rt, minn = 0x3f3f3f3f, p = 0;
127     while (u) {
128         if (val[u] > v && val[u] < minn) minn = val[u], p = u;
129         if (v >= val[u]) u = rc;
130         else u = lc;
131     }
132     if (!p) splay(p, rt);
133     return minn;
134 }
135 int lower(int v) {
136     int u = rt, maxx = -0x3f3f3f3f, p = 0;
137     while (u) {
138         if (val[u] < v && val[u] > maxx) maxx = val[u], p = u;
139         if (v <= val[u]) u = lc;
140         else u = rc;
141     }
142     if (!p) splay(p, rt);
143     return maxx;
144 }
145 };

```

5.7 Splay 区间翻转

```

1 struct Splay {
2     #define rt ch[0][1]
3     #define lc ch[u][0]
4     #define rc ch[u][1]
5     int tot, totElement;
6     int val[MAX], fa[MAX], sum[MAX], tag[MAX];
7     int ch[MAX][2];
8     Splay() { tot = totElement = 0; }
9     void push_up(int u) { sum[u] = sum[lc] + sum[rc] + 1; }
10    void push_down(int u) {
11        if (tag[u]) {
12            tag[lc] ^= 1;
13            tag[rc] ^= 1;
14            tag[u] = 0;
15            swap(lc, rc);
16        }
17    }
18    int identify(int u) { return ch[fa[u]][1] == u; }
19    void connect(int u, int par, int son) { ch[par][son] = u, fa[u] = par; }
20    void rotate(int u) {
21        int fc = identify(u), f = fa[u];
22        int gc = identify(f), g = fa[f];
23        int uc = fc ^ 1, son = ch[u][uc];
24        connect(son, f, fc);
25        connect(f, u, uc);
26        connect(u, g, gc);

```

```

27     push_up(f);
28     push_up(u);
29 }
30 void splay(int u, int v) {
31     v = fa[v];
32     while (fa[u] != v) {
33         int f = fa[u];
34         if (fa[f] != v)
35             rotate(identify(u) ^ identify(f) ? u : f);
36         rotate(u);
37     }
38 }
39 int creat(int v, int par) {
40     val[++tot] = v;
41     fa[tot] = par;
42     sum[tot] = 1;
43     return tot;
44 }
45 int insert(int v) {
46     totElement++;
47     if (totElement == 1) {
48         creat(v, 0);
49         return rt = tot;
50     }
51     int u = rt;
52     while (1) {
53         sum[u]++;
54         if (v == val[u]) return u;
55         int nxt = v > val[u];
56         if (!ch[u][nxt]) {
57             creat(v, u);
58             splay(ch[u][nxt] = tot, rt);
59             return tot;
60         }
61         u = ch[u][nxt];
62     }
63 }
64 int queryKth(int k) {
65     if (k > totElement) return -1;
66     int u = rt;
67     while (1) {
68         push_down(u);
69         if (k > sum[lc] && k <= sum[lc] + 1) {
70             splay(u, rt);
71             return val[u];
72         }
73         if (k <= sum[lc]) u = lc;
74         else {
75             k -= sum[lc] + 1;
76             u = rc;
77         }
78     }
79 }
80 void reverse(int ql, int qr) {

```

```

81         ql = queryKth(ql);
82         qr = queryKth(qr + 2);
83         splay(ql, rt);
84         splay(qr, ch[ql][1]);
85         tag[ch[qr][0]] ^= 1;
86     }
87 } splay;

```

5.8 Trie

```

1 struct Trie {
2     static const int MAX_N = 1e6 + 10, rt = 0;
3     int ch[MAX_N][26], sum[MAX_N], tot;
4     void init() { tot = 0; memset(ch[tot], 0, sizeof(ch[tot])); sum[tot] = 0; }
5     void insert(char *s, int val) {
6         int len = strlen(s), u = rt;
7         for (int i = 0; i < len; i++) {
8             int nxt = s[i] - 'a';
9             if (!ch[u][nxt]) {
10                 ch[u][nxt] = ++tot;
11                 memset(ch[tot], 0, sizeof(ch[tot]));
12                 sum[tot] = 0;
13             }
14             u = ch[u][nxt];
15         }
16         sum[u] += val;
17     }
18     int query(char *s) {
19         int len = strlen(s), u = rt;
20         for (int i = 0; i < len; i++) {
21             int nxt = s[i] - 'a';
22             if (!ch[u][nxt]) return -1;
23             u = ch[u][nxt];
24         }
25         if (!sum[u]) return -1;
26         return sum[u];
27     }
28 };

```

5.9 主席树

5.9.1 动态主席树

```

1 //树状数组套主席树，动态维护区间第k小
2 //当前点p作用于[p, N]的主席树（普通主席树是在rt[p - 1]的基础上得到）
3 //查询[1, p]，即可得到rt[p]时的状态
4 int N, M;
5 int a[MAX], b[MAX << 1], n;
6
7 struct node {
8     int x, y, z;
9 } q[MAX];
10 -----动态主席树-----

```

```

11 int rt[MAX], ru[MAX], rv[MAX], tot;
12 int lc[MAX_N], rc[MAX_N], sum[MAX_N];
13
14 void update(int &now, int l, int r, int p, int v) {
15     if (!now) now = ++tot;
16     sum[now] += v;
17     if (l < r) {
18         if (p <= m) update(lc[now], l, m, p, v);
19         else update(rc[now], m + 1, r, p, v);
20     }
21 }
22
23 void change(int p, int v) {
24     int pos = lower_bound(b + 1, b + 1 + n, a[p]) - b;
25     for (int i = p; i <= n; i += lowbit(i))
26         update(rt[i], 1, n, pos, v);
27 }
28
29 int query(int ql, int qr, int k) {
30     ql--; //前缀和相减, [ql, qr]的状态为pre[qr] - pre[ql - 1]
31     int cnt1 = 0, cnt2 = 0;
32     for (int i = qr; i; i -= lowbit(i)) ru[++cnt1] = rt[i];
33     for (int i = ql; i; i -= lowbit(i)) rv[++cnt2] = rt[i];
34     int l = 1, r = n;
35     while (l < r) {
36         int num = 0;
37         for (int i = 1; i <= cnt1; i++) num += sum[lc[ru[i]]];
38         for (int i = 1; i <= cnt2; i++) num -= sum[lc[rv[i]]];
39         if (k <= num) {
40             for (int i = 1; i <= cnt1; i++) ru[i] = lc[ru[i]];
41             for (int i = 1; i <= cnt2; i++) rv[i] = lc[rv[i]];
42             r = m;
43         }
44         else {
45             for (int i = 1; i <= cnt1; i++) ru[i] = rc[ru[i]];
46             for (int i = 1; i <= cnt2; i++) rv[i] = rc[rv[i]];
47             l = m + 1;
48             k -= num;
49         }
50     }
51     return b[l];
52 }
53
54 int main() {
55     scanf("%d%d", &N, &M);
56     for (int i = 1; i <= N; i++) scanf("%d", &a[i]), b[++n] = a[i];
57     for (int i = 1; i <= M; i++) {
58         char ch = getchar();
59         while (ch != 'Q' && ch != 'C') ch = getchar();
60         scanf("%d%d", &q[i].x, &q[i].y);
61         if (ch == 'Q') scanf("%d", &q[i].z);
62         else b[++n] = q[i].y;
63     }
64     sort(b + 1, b + 1 + n);

```

```

65     n = unique(b + 1, b + 1 + n) - b - 1;
66
67     for (int i = 1; i <= N; i++)
68         change(i, 1);
69
70     for (int i = 1; i <= M; i++) {
71         if (!q[i].z) {
72             change(q[i].x, -1);
73             a[q[i].x] = q[i].y;
74             change(q[i].x, 1);
75         }
76         else {
77             printf("%d\n", query(q[i].x, q[i].y, q[i].z));
78         }
79     }
80
81     return 0;
82 }

```

5.9.2 区间不同数

```

1  int last[N]; //上一种i出现位置
2  int rt[N], tot;
3  int lc[MAX_N], rc[MAX_N], sum[MAX_N];
4
5  void update(int &now, int pre, int l, int r, int p, int v) {
6      now = ++tot;
7      sum[now] = sum[pre] + v, lc[now] = lc[pre], rc[now] = rc[pre];
8      if (l < r) {
9          if (p <= mid) update(lc[now], lc[pre], l, mid, p, v);
10         else update(rc[now], rc[pre], mid + 1, r, p, v);
11     }
12 }
13
14 int query(int now, int l, int r, int ql, int qr) {
15     if (ql <= l && r <= qr) return sum[now];
16     int res = 0;
17     if (ql <= mid) res += query(lc[now], l, mid, ql, qr);
18     if (qr > mid) res += query(rc[now], mid + 1, r, ql, qr);
19     return res;
20 }
21
22 int main() {
23     for (int i = 1; i <= n; i++) {
24         int x; scanf("%d", &x);
25         update(rt[i], rt[i - 1], 1, n, i, 1);
26         if (last[x]) update(rt[i], rt[i], 1, n, last[x], -1); //上一种出现就删除掉
27         last[x] = i;
28     }
29     while (m--) {
30         int ql, qr; scanf("%d%d", &ql, &qr);
31         //rt[qr]包含[1, qr]的信息, 这里只查询[ql, qr]部分
32         printf("%d\n", query(rt[qr], 1, n, ql, n));

```

```

33     }
34     return 0;
35 }

```

5.9.3 区间第 K 小

```

1  struct Hash {
2      int b[N], tot;
3      void init() { tot = 0; }
4      void insert(int x) { b[++tot] = x; }
5      void build() {
6          sort(b + 1, b + 1 + tot);
7          tot = unique(b + 1, b + 1 + tot) - (b + 1);
8      }
9      int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
10 };
11
12 const int MAX_N = N * 25;
13 int rt[N], tot;
14 int lc[MAX_N], rc[MAX_N], num[MAX_N];
15
16 void update(int &now, int pre, int l, int r, int p) {
17     now = ++tot;
18     num[now] = num[pre] + 1, lc[now] = lc[pre], rc[now] = rc[pre];
19     if (l == r) return;
20     if (p <= mid) update(lc[now], lc[pre], l, mid, p);
21     else update(rc[now], rc[pre], mid + 1, r, p);
22 }
23
24 int query(int now, int pre, int k) {
25     int l = 1, r = ha.tot;
26     while (l < r) {
27         int sum = num[lc[now]] - num[lc[pre]];
28         if (k <= sum) {
29             now = lc[now], pre = lc[pre];
30             r = mid;
31         }
32         else {
33             now = rc[now], pre = rc[pre];
34             l = mid + 1;
35             k -= sum;
36         }
37     }
38     return ha.b[l];
39 }

```

5.9.4 树上建树

```

1  //树上建主席树，支持动态修改边
2  int N, M, Q;
3  int a[MAX], b[MAX], n;
4
5  int pos(int x) { return lower_bound(b + 1, b + 1 + n, x) - b; }

```



```

6
7 struct edge {
8     int nxt, to;
9 }e[MAX << 2];
10
11 int cnt, head[MAX];
12
13 void add(int u, int v) {
14     e[++cnt].nxt = head[u];
15     e[cnt].to = v;
16     head[u] = cnt;
17 }
18
19 -----主席树
20
21 int tot, rt[MAX];
22 int lc[MAX * 400], rc[MAX * 400], num[MAX * 400];
23
24 int update(int pre, int l, int r, int p) {
25     int now = ++tot;
26     num[now] = num[pre] + 1;
27     lc[now] = lc[pre], rc[now] = rc[pre];
28     if (l < r) {
29         if (p <= m)lc[now] = update(lc[pre], l, m, p);
30         else rc[now] = update(rc[pre], m + 1, r, p);
31     }
32     return now;
33 }
34
35 int query(int u, int v, int x, int y, int k) {
36     int l = 1, r = n;
37     while (l < r) {
38         int ls = num[lc[u]] + num[lc[v]] - num[lc[x]] - num[lc[y]];
39         if (k <= ls) {
40             u = lc[u], v = lc[v], x = lc[x], y = lc[y];
41             r = m;
42         }
43         else {
44             u = rc[u], v = rc[v], x = rc[x], y = rc[y];
45             k -= ls, l = m + 1;
46         }
47     }
48     return l;
49 }
50 -----LCA
51
52 int dep[MAX], fa[MAX][20], lg[MAX], siz[MAX], pre[MAX], fff, vis[MAX];
53
54 void init(int N) {
55     for (int i = 1; i <= N; i++)
56         lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
57 }
58
59 void dfs(int u, int f) {

```

```

58     vis[u] = 1;
59     //建树, 在fa的基础上建树
60     rt[u] = update(rt[f], 1, n, pos(a[u]));
61     siz[u] = 1;
62     dep[u] = dep[f] + 1;
63     fa[u][0] = f, pre[u] = fff;
64     //此处本来应该是 (1 << i) <= dep[u]
65     //但是这里是动态的树, 有边的修改, 所以祖先节点也需要更新
66     for (int i = 1; i <= 18; i++)
67         fa[u][i] = fa[fa[u][i - 1]][i - 1];
68     for (int i = head[u]; i; i = e[i].nxt)
69         if (e[i].to != f) {
70             dfs(e[i].to, u);
71             siz[u] += siz[e[i].to];
72         }
73 }
74
75 int lca(int x, int y) {
76     if (dep[x] < dep[y])
77         swap(x, y);
78     while (dep[x] > dep[y])
79         x = fa[x][lg[dep[x]] - dep[y] - 1];
80     if (x == y) return x;
81     for (int k = lg[dep[x]] - 1; k >= 0; k--)
82         if (fa[x][k] != fa[y][k])
83             x = fa[x][k], y = fa[y][k];
84     return fa[x][0];
85 }
86
87 int main() {
88     int T = read();
89     N = read(); M = read(); Q = read();
90     init(N);
91
92     for (int i = 1; i <= N; i++) a[i] = read(), b[i] = a[i], pre[i] = i;
93     sort(b + 1, b + 1 + N);
94     n = unique(b + 1, b + 1 + N) - (b + 1);
95
96     for (int i = 1; i <= M; i++) {
97         int u, v;
98         u = read(); v = read();
99         add(u, v); add(v, u);
100     }
101
102     for (int i = 1; i <= N; i++)
103         if (!vis[i]) {
104             fff = i;
105             dfs(i, 0);
106         }
107
108     char op[10];
109     int last_ans = 0;
110     while (Q--) {
111         scanf("%s", op);

```

```

112     int x, y, k;
113     x = read(); y = read();
114     x ^= last_ans; y ^= last_ans;
115     if (op[0] == 'Q') {
116         k = read(); k ^= last_ans;
117         int xy = lca(x, y);
118         last_ans = b[query(rt[x], rt[y], rt[xy], rt[fa[xy][0]], k)];
119         printf("%d\n", last_ans);
120     }
121     else {
122         add(x, y); add(y, x);
123         int fx = pre[x], fy = pre[y];
124         if (siz[fx] < siz[fy]) {
125             fff = fy;
126             dfs(x, y);
127         }
128         else {
129             fff = fx;
130             dfs(y, x);
131         }
132     }
133 }
134 return 0;
135 }

```

5.10 可持久化 01Trie

```

1 struct Trie {
2     static const int MAX_N = N * 35;
3     static const int MAX_BIT = 28;
4     int rt[N];
5     int ch[MAX_N][2], sum[MAX_N], tot;
6     void insert(int now, int pre, ll val) {
7         rt[now] = ++tot;
8         now = rt[now], pre = rt[pre];
9         for (int bit = MAX_BIT; bit >= 0; bit--) {
10             int s = (val >> bit) & 1;
11             sum[now] = sum[pre] + 1;
12             if (!ch[now][s]) ch[now][s] = ++tot;
13             ch[now][s ^ 1] = ch[pre][s ^ 1];
14             now = ch[now][s];
15             pre = ch[pre][s];
16         }
17         sum[now] = sum[pre] + 1;
18     }
19     ll queryKth(int now, int pre, ll val, int k = 1) {
20         now = rt[now], pre = rt[pre];
21         ll res = 0;
22         for (int bit = MAX_BIT; bit >= 0; bit--) {
23             int s = (val >> bit) & 1;
24             int lsum = sum[ch[now][s ^ 1]] - sum[ch[pre][s ^ 1]];
25             if (lsum >= k) res += (1 << bit), now = ch[now][s ^ 1], pre = ch[pre][s ^ 1];
26             else k -= lsum, now = ch[now][s], pre = ch[pre][s];

```

```

27     }
28     return res;
29 }
30 } trie;

```

5.11 扫描线

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define INF 0x3f3f3f3f
4  #define sz  sizeof
5  #define eps 1e-9
6  #define lowbit(x) x&-x
7  #define lc  u<<1
8  #define rc  u<<1|1
9  #define mid (l+r)/2
10 typedef long long ll;
11 const int MAX = 1e5 + 10;
12
13 int N;
14 int X[MAX << 1];
15 int tot, cnt;
16
17 int pos(int x) { return lower_bound(X + 1, X + 1 + cnt, x) - X; }
18
19 struct Scanline {
20     int l, r, h, f;
21     bool operator < (const Scanline &rhs) const {
22         return h < rhs.h;
23     }
24 } line[MAX << 1];
25
26 struct SegmentTree {
27     int l, r, sum, len;
28 } t[MAX << 2];
29
30 void push_up(int u) {
31     if (t[u].sum) t[u].len = X[t[u].r + 1] - X[t[u].l];
32     else t[u].len = t[lc].len + t[rc].len;
33 }
34
35 void build(int u, int l, int r) {
36     t[u] = SegmentTree{ l, r, 0, 0 };
37     if (l == r) return;
38     build(lc, l, mid); build(rc, mid + 1, r);
39 }
40
41 void update(int u, int ql, int qr, int k) {
42     if (ql <= t[u].l && t[u].r <= qr) {
43         t[u].sum += k;
44         return;
45     }
46

```

```

47 }
48
49 int main() {
50 #ifdef ACM_LOCAL
51     freopen("input.in", "r", stdin);
52     freopen("output.out", "w", stdout);
53     auto start_clock = clock();
54 #endif
55     scanf("%d", &N);
56     for (int i = 1; i <= N; i++) {
57         int x1, x2, y1, y2; scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
58         line[++tot] = Scanline{ x1, x2, y1, 1 };
59         line[++tot] = Scanline{ x1, x2, y2, -1 };
60         X[++cnt] = x1, X[++cnt] = x2;
61     }
62     sort(line + 1, line + 1 + tot);
63     sort(X + 1, X + 1 + cnt);
64     cnt = unique(X + 1, X + 1 + cnt) - (X + 1);
65
66
67
68 #ifdef ACM_LOCAL
69     cerr << (double)(clock() - start_clock) / CLOCKS_PER_SEC << "s" << endl;
70 #endif
71     return 0;
72 }

```

5.12 线性基

```

1 struct LinerBase {
2     static const int MAX_BIT = 60;
3     ll num[65], tmp[65];
4     bool flag; // 能否表示0
5     LinerBase() {
6         flag = 0;
7         memset(num, 0, sizeof(num));
8         memset(tmp, 0, sizeof(tmp));
9     }
10    void insert(ll x) {
11        for (int bit = MAX_BIT; ~bit; bit--)
12            if (x & (1ll << bit)) {
13                if (num[bit] x ^= num[bit];
14                else {
15                    num[bit] = x;
16                    return;
17                }
18            }
19        flag = 1;
20    }
21    ll queryMax() {
22        ll res = 0;
23        for (int bit = MAX_BIT; ~bit; bit--) res = max(res, res ^ num[bit]);
24        return res;

```

```

25     }
26     ll queryMin() {
27         if (flag) return 0;
28         for (int bit = 0; bit <= MAX_BIT; bit++)
29             if (num[bit]) return num[bit];
30     }
31     ll queryKth(ll k) {
32         ll res = 0, cnt = 0;
33         k -= flag; if (!k) return 0;
34         for (int i = 0; i < MAX_BIT; i++) {
35             for (int j = i - 1; ~j; j--)
36                 if (num[i] & (1ll << j)) num[i] ^= num[j];
37             if (num[i]) tmp[cnt++] = num[i];
38         }
39         if (k >= (1ll << cnt)) return -1;
40         for (int i = 0; i < cnt; i++)
41             if (k & (1ll << i)) res ^= tmp[i];
42         return res;
43     }
44 } linerBase;
45
46 struct LinerBase {
47     //区间查询
48     //构造的时候, lb[i] = lb[i - 1], lb[i].insert(x, i)
49     //ans = lb[qr].queryMax(ql)
50     static const int MAX_BIT = 30;
51     int num[32], pos[32];
52     LinerBase() {
53         memset(num, 0, sizeof(num));
54         memset(pos, 0, sizeof(pos));
55     }
56     void insert(int x, int p) {
57         for (int bit = MAX_BIT; ~bit; bit--)
58             if (x & (1ll << bit)) {
59                 if (!num[bit]) {
60                     num[bit] = x, pos[bit] = p;
61                     return;
62                 }
63                 else if (p > pos[bit]) {
64                     swap(num[bit], x);
65                     swap(pos[bit], p);
66                 }
67                 x ^= num[bit];
68             }
69     }
70     int queryMax(int ql) {
71         int res = 0;
72         for (int bit = MAX_BIT; ~bit; bit--)
73             if (pos[bit] >= ql)
74                 res = max(res, res ^ num[bit]);
75         return res;
76     }
77 } lb[N];
78

```

```

79 LinerBase merge(const LinerBase &a, const LinerBase &b) {
80     LinerBase res;
81     for (int bit = res.MAX_BIT; ~bit; bit--) res.num[bit] = a.num[bit];
82     for (int bit = res.MAX_BIT; ~bit; bit--) if (b.num[bit]) res.insert(b.num[bit]);
83     return res;
84 }

```

5.13 线段树动态开点合并分裂

```

1
2 #define mid (l+r)/2
3 static const int MAX_N = N * 40;
4 int rt[N], now;
5 int lc[MAX_N], rc[MAX_N];
6 ll sum[MAX_N];
7 int tot, rub[MAX_N];
8 int newNode() { return rub[0] ? rub[rub[0]--] : ++tot; }
9 void remove(int &u) {
10     lc[u] = rc[u] = sum[u] = 0;
11     rub[++rub[0]] = u;
12     u = 0;
13 }
14 void push_up(int u) { sum[u] = sum[lc[u]] + sum[rc[u]]; }
15 void build(int &u, int l, int r) {
16     u = newNode();
17     if (l == r) {
18         sum[u] = cnt[l];
19         return;
20     }
21     build(lc[u], l, mid); build(rc[u], mid + 1, r);
22     push_up(u);
23 }
24 void update(int &u, int l, int r, int p, ll k) {
25     if (!u) u = newNode();
26     if (l == r) {
27         sum[u] += k;
28         return;
29     }
30     if (p <= mid) update(lc[u], l, mid, p, k);
31     else update(rc[u], mid + 1, r, p, k);
32     push_up(u);
33 }
34 ll querySum(int u, int l, int r, int ql, int qr) {
35     if (!u) return 0;
36     if (ql <= l && r <= qr) return sum[u];
37     ll res = 0;
38     if (ql <= mid) res += querySum(lc[u], l, mid, ql, qr);
39     if (qr > mid) res += querySum(rc[u], mid + 1, r, ql, qr);
40     return res;
41 }
42 int queryKth(int u, int l, int r, ll k) {
43     if (l == r) return l;
44     if (k <= sum[lc[u]]) return queryKth(lc[u], l, mid, k);

```

```

45     else return queryKth(rc[u], mid + 1, r, k - sum[lc[u]]);
46 }
47 void merge(int u, int v, int l, int r) {
48     if (l == r) {
49         sum[u] += sum[v];
50         return;
51     }
52     if (lc[u] && lc[v]) merge(lc[u], lc[v], l, mid), remove(lc[v]);
53     else if (lc[v]) lc[u] = lc[v], lc[v] = 0;
54     if (rc[u] && rc[v]) merge(rc[u], rc[v], mid + 1, r), remove(rc[v]);
55     else if (rc[v]) rc[u] = rc[v], rc[v] = 0;
56     push_up(u);
57 }
58 void split(int &newp, int &u, int l, int r, int ql, int qr) { //分裂出[ql, qr]间的点
59     if (!u) return;
60     if (ql <= l && r <= qr) {
61         newp = u;
62         u = 0;
63         return;
64     }
65     if (!newp) newp = newNode();
66     if (ql <= mid) split(lc[newp], lc[u], l, mid, ql, qr);
67     if (qr > mid) split(rc[newp], rc[u], mid + 1, r, ql, qr);
68     push_up(u);
69     push_up(newp);
70 }
71 }
72 #undef mid

```

6 树和森林

6.1 LCT

```

1  int ch[N][2], fa[N], rev[N], siz[N]; //基本内容
2  int sum[N], val[N], tag[N]; //另外要维护的
3  #define lc ch[u][0]
4  #define rc ch[u][1]
5  #define identify(u) (ch[fa[u]][1] == u)
6  #define isRoot(u) (u != ch[fa[u]][0] && u != ch[fa[u]][1])
7  void flip(int u) { swap(lc, rc); rev[u] ^= 1; }
8  void push_up(int u) {
9      siz[u] = siz[lc] + siz[rc] + 1;
10     //...
11 }
12 void push_down(int u) {
13     if (rev[u]) {
14         if (lc) flip(lc);
15         if (rc) flip(rc);
16         rev[u] = 0;
17     }
18     //...
19 }
20 void update(int u) { //当前点之上的所有点都push_down

```



```

21     if (!isRoot(u)) update(fa[u]);
22     push_down(u);
23 }
24 void rotate(int u) {
25     int f = fa[u], fc = identify(u);
26     int g = fa[f], gc = identify(f);
27     int uc = fc ^ 1, c = ch[u][uc];
28     if (!isRoot(f))
29         ch[g][gc] = u; fa[u] = g;
30     ch[f][fc] = c, fa[c] = f;
31     ch[u][uc] = f, fa[f] = u;
32     push_up(f); push_up(u);
33 }
34 void splay(int u) { //将u变为u所在的Splay的根
35     update(u);
36     for (int f; f = fa[u], !isRoot(u); rotate(u))
37         if (!isRoot(f)) rotate(identify(f) ^ identify(u) ? u : f);
38 }
39 int access(int u) { //将(rt, u)之间的路径变为实链
40     int pre = 0;
41     for (; u; u = fa[pre = u])
42         splay(u), rc = pre, push_up(u);
43     return pre;
44 }
45 void makeRoot(int u) { //将u变为整棵树的根(注意:不一定是当前splay的根)
46     u = access(u);
47     flip(u);
48 }
49 int findRoot(int u) {
50     access(u), splay(u);
51     while (lc) push_down(u), u = lc;
52     splay(u);
53     return u;
54 }
55 void link(int u, int v) {
56     makeRoot(u); splay(u);
57     if (findRoot(v) != u) fa[u] = v;
58 }
59 void split(int u, int v) {
60     makeRoot(u);
61     access(v); splay(v); //加了这个就将v变为splay的根
62 }
63 void cut(int u, int v) {
64     makeRoot(u); splay(u);
65     if (findRoot(v) == u && fa[v] == u && !ch[v][0]) {
66         fa[v] = ch[u][1] = 0;
67         push_up(u);
68     }
69 }
70 void fix(int u, int k) {
71     splay(u); val[u] = k;
72 }

```

6.2 带权并查集

```
1  int pre[N], num[N], dis[N]; // num[i] 表示以 i 为队头时队内元素数量, dis[i] 表示到达 i 的祖先节点的距离
2
3  void merge(int x, int y) {
4      dis[x] = num[y];
5      num[y] += num[x];
6      num[x] = 0;
7      pre[x] = y;
8  }
9
10 int find(int x) { // 路径压缩并查集
11     if (x == pre[x]) return x;
12     int t = find(pre[x]);
13     dis[x] += dis[pre[x]];
14     return pre[x] = t;
15 }
```

6.3 按秩合并可撤回并查集

```
1  struct node {
2      int x, y, z;
3  };
4
5  struct UnionFind {
6  private:
7      int rk[N], pre[N], siz[N], totNode; // N 为最大点数
8      stack<node> st; // node 记录上次修改的内容
9  public:
10     void init(int tot) {
11         totNode = tot;
12         for (int i = 1; i <= totNode; i++)
13             pre[i] = i, siz[i] = rk[i] = 1;
14     }
15     int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }
16     void merge(int x, int y) { // 按秩合并
17         x = find(x), y = find(y);
18         if (x == y) return;
19         if (rk[x] < rk[y]) swap(x, y);
20         st.push(node{ y, rk[x], siz[y] });
21         pre[y] = x, rk[x] += rk[y], siz[x] += siz[y];
22     }
23     int start() { return st.size(); }
24     void end(int last) { // 撤回 merge 操作
25         while (st.size() > last) {
26             node tp = st.top();
27             rk[pre[tp.x]] -= tp.y, siz[pre[tp.x]] -= tp.z;
28             pre[tp.x] = tp.x;
29             st.pop();
30         }
31     }
32     bool judge() { return siz[find(1)] == totNode; }
33 } uf;
```

6.4 树上 K 祖先

```
1 //倍增KFA, 空间大点, 但是好写
2 vector<int> g[N];
3
4 int anc[N][20];
5 void dfs(int u, int fa) {
6     anc[u][0] = fa;
7     for (int i = 1; i <= 19; i++) anc[u][i] = anc[anc[u][i - 1]][i - 1];
8     for (auto &v: g[u])
9         if (v != fa) dfs(v, u);
10 }
11
12 int kthFa(int u, int k) {
13     int bit = 0;
14     while (k) {
15         if (k & 1) u = anc[u][bit];
16         k >>= 1;
17         bit++;
18     }
19     return u;
20 }
21
22
23 //树剖KFA
24 int siz[N], son[N], dep[N], fa[N], top[N];
25 int id[N], nodeOf[N], cnt;
26 void dfs(int u, int par) {
27     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
28     for (auto &v: g[u])
29         if (v != par) {
30             dfs(v, u);
31             siz[u] += siz[v];
32             if (!son[u] || siz[v] > siz[son[u]])
33                 son[u] = v;
34         }
35 }
36
37 void dfs2(int u, int topf) {
38     nodeOf[id[u] = ++cnt] = u, top[u] = topf;
39     if (!son[u]) return;
40     dfs2(son[u], topf);
41     for (auto &v: g[u])
42         if (v != fa[u] && v != son[u]) dfs2(v, v);
43 }
44
45 int kthFa(int u, int k) {
46     while (k >= id[u] - id[top[u]] + 1 && u) {
47         k -= id[u] - id[top[u]] + 1;
48         u = fa[top[u]];
49     }
50     return nodeOf[id[u] - k];
51 }
```

6.5 树上启发式合并

```
1
2 //树上启发式合并和点分治都能解决树上边、点信息问题，一般来说
3 //如果已经确定树根，就不能用点分治，因为点分治需要找树的重心
4 //而树上启发式合并更侧重于处理——子树信息
5
6 int siz[N], son[N];
7 void dfs(int u, int fa) {
8     siz[u] = 1;
9     for (auto &v: g[u])
10         if (v != fa) {
11             dfs(v, u);
12             siz[u] += siz[v];
13             if (!son[u] || siz[v] > siz[son[u]])
14                 son[u] = v;
15         }
16 }
17 int vis[N];
18
19 void upd(int u, int fa, int k) {
20
21     for (auto &v: g[u])
22         if (v != fa && !vis[v]) upd(v, u, k);
23 }
24
25 void dsu(int u, int fa, int keep) { //点信息
26     for (auto &v: g[u])
27         if (v != fa && v != son[u]) dsu(v, u, 0);
28     if (son[u]) dsu(son[u], u, 1), vis[son[u]] = 1;
29     //更新自己+子树信息的同时统计答案
30
31     if (son[u]) vis[son[u]] = 0;
32     if (!keep) upd(u, fa, -1);
33 }
34
35
36 int id[N], nodeOf[N], cnt;
37 void dsu(int u, int fa, int keep) { //边信息
38     for (int i = head[u], v; i; i = e[i].nxt)
39         if ((v = e[i].to) != fa && v != son[u]) dsu(v, u, 0);
40     if (son[u]) dsu(son[u], u, 1);
41     for (int i = head[u], v; i; i = e[i].nxt)
42         if ((v = e[i].to) != fa && v != son[u]) {
43             for (int j = 0; j < siz[v]; j++)
44                 upd(nodeOf[id[v] + j]);
45             for (int j = 0; j < siz[v]; j++)
46                 update(nodeOf[id[v] + j], 1);
47         }
48     upd(u);
49     update(u, 1);
50     if (!keep) {
51         for (int j = 0; j < siz[u]; j++)
52             update(nodeOf[id[u] + j], -1);
```

```

53     }
54 }
55
56 int main() {
57
58     int rt = 1; dfs(rt, 0);
59     dsu(rt, 0, 0);
60
61     return 0;
62 }

```

6.6 树剖 LCA

```

1
2 int son[N], siz[N], top[N], fa[N], dep[N];
3 void dfs(int u, int par) {
4     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
5     int max_son = -1;
6     for (auto &v: g[u])
7         if (v != par) {
8             dfs(v, u);
9             siz[u] += siz[v];
10            if (max_son < siz[v])
11                son[u] = v, max_son = siz[v];
12        }
13 }
14 void dfs2(int u, int topf) {
15     top[u] = topf;
16     if (!son[u]) return;
17     dfs2(son[u], topf);
18     for (auto &v: g[u])
19         if (v != fa[u] && v != son[u]) dfs2(v, v);
20 }
21 int LCA(int x, int y) {
22     while (top[x] != top[y]) {
23         if (dep[top[x]] < dep[top[y]]) swap(x, y);
24         x = fa[top[x]];
25     }
26     return dep[x] < dep[y] ? x : y;
27 }

```

6.7 树的直径

```

1 //树的直径性质:
2 //1、直径两 endpoints 一定是两个叶子节点
3 //2、距离任意点最远的点一定是直径的一个端点, 这个基于贪心求直径方法的正确性可以得出
4 //3、对于两棵树, 如果第一棵树直径两 endpoints 为  $(u, v)$ , 第二棵树直径两 endpoints 为  $(x, y)$ , 用一条边将两
   棵树连接, 那么新树的直径一定是  $u, v, x, y, u, v, x, y$  中的两个点
5 //4、对于一棵树, 如果在一个点的上接一个叶子节点, 那么最多会改变直径的一个端点
6 //5、若一棵树存在多条直径, 那么这些直径交于一点且交点是这些直径的中点
7
8 //P3761 [TJOI2017]城市

```

```

9 //求去掉一条高速公路，并且重新修建一条一样的高速公路(即交通费用一样)，使得这个地区的两个城市之间的
   最大交通费用最小
10 //只需每次去掉一条路，查询两边连通块树的直径，树的半径
11 //最后答案为 两个连通块的树的直径 和 树的半径之和 的最小值
12
13 const int MAX = 5e3;
14 int N;
15 int dis[MAX], tag[MAX], ans, half;
16 //tag标记两个点是否要分开
17
18
19 struct edge {
20     int nxt, to, w;
21 } e[MAX << 1];
22
23 int head[MAX], tot;
24
25 void add(int u, int v, int w) {
26     e[++tot].to = v;
27     e[tot].w = w;
28     e[tot].nxt = head[u];
29     head[u] = tot;
30 }
31
32 void init() {
33     tot = 0;
34     memset(head, 0, sz(head));
35     memset(tag, 0, sz(tag));
36 }
37
38 pii dfs(int u, int fa) { //查找树的直径
39     pii mx = make_pair(dis[u], u);
40     for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
41         if (v != fa && !tag[v]) {
42             dis[v] = dis[u] + e[i].w;
43             mx = max(mx, dfs(v, u));
44         }
45     return mx;
46 }
47
48 pii getDiameter(int start) { //从任意出发点开始，查找树的直径的两个端点，长度为dis[second]
49     dis[start] = 0;
50     start = dfs(start, 0).second;
51     dis[start] = 0;
52     return make_pair(start, dfs(start, 0).second);
53 }
54
55 bool findHalf(int u, int fa, int d, int totlen, int end) { //查找树的半径
56     if (u == end) {
57         half = min(half, max(totlen - d, d));
58         return true;
59     }
60     for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
61         if (v != fa && !tag[v]) {

```

```

62         if (findHalf(v, u, d + e[i].w, totLen, end)) {
63             half = min(half, max(totLen - d, d));
64             return true;
65         }
66     }
67     return false;
68 }
69
70 pii getHalf(int s) { //得到直径和半径
71     pii t = getStartEnd(s); int totLen = dis[t.second];
72     half = INF; findHalf(t.first, 0, 0, totLen, t.second);
73     return make_pair(totLen, half);
74 }
75
76 bool solve(int u, int fa, int end) {
77     if (u == end) return true;
78     for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
79         if (v != fa) {
80             if (solve(v, u, end)) {
81                 //查找v块中树的直径的一半, 查找u块中树的直径的一半
82                 //答案是所有..中最小的
83                 tag[v] = 1; pii h1 = getHalf(u); tag[v] = 0;
84                 tag[u] = 1; pii h2 = getHalf(v); tag[u] = 0;
85                 int n1 = h1.first, n2 = h2.first, n3 = h1.second + h2.second + e[i].w;
86                 ans = min(ans, max(max(n1, n2), n3));
87                 tag[v] = tag[u] = 0;
88                 return true;
89             }
90         }
91     return false;
92 }
93
94 int main() {
95     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
96     cin >> N;
97     init();
98     for (int i = 1; i < N; i++) {
99         int u, v, w; cin >> u >> v >> w;
100         add(u, v, w); add(v, u, w);
101     }
102     ans = INF;
103     pii t = getStartEnd(1);
104     int start = t.first, end = t.second;
105     solve(start, 0, end);
106     cout << ans << endl;
107
108     return 0;
109 }

```

6.8 树的重心

- 1 //性质
- 2 //以重心为根, 所有的子树的大小都不超过整个树大小的一半

```

3 //树的重心最多有两个
4 //树的重心到其他节点的距离是最小的
5 //把两个树通过一条边相连得到一个新的树，那么新的树的重心在连接原来两个树的重心的路径上
6 //把一个树添加或删除一个叶子，那么它的重心最多只移动一条边的距离
7
8 int maxp[N], siz[N], rt;
9
10 void getRt(int u, int fa, int all) { //树的重心
11     siz[u] = 1, maxp[u] = 0;
12     for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
13         if (v != fa) {
14             getRt(v, u, all);
15             siz[u] += siz[v];
16             maxp[u] = max(maxp[u], siz[v]);
17         }
18     maxp[u] = max(maxp[u], all - siz[u]); //fa子树为all - siz[u]
19     if (maxp[u] < maxp[rt]) rt = u;
20 }
21
22 int main() {
23
24     maxp[rt = 0] = n; getRt(1, 0, n);
25
26     return 0;
27 }

```

6.9 树链剖分 (点权)

```

1 int siz[N], son[N], dep[N], fa[N], top[N];
2 int nodeOf[N], id[N], cnt;
3 void dfs(int u, int par) {
4     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
5     for (int i = head[u], v; i; i = e[i].nxt)
6         if ((v = e[i].to) != par) {
7             dfs(v, u);
8             siz[u] += siz[v];
9             if (!son[u] || siz[v] > siz[son[u]])
10                 son[u] = v;
11         }
12 }
13 void dfs2(int u, int topf) {
14     nodeOf[id[u] = ++cnt] = u, top[u] = topf;
15     if (!son[u]) return;
16     dfs2(son[u], topf);
17     for (int i = head[u], v; i; i = e[i].nxt)
18         if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
19 }
20
21 int ask(int x, int y) {
22     int res = 0;
23     while (top[x] != top[y]) {
24         if (dep[top[x]] < dep[top[y]]) swap(x, y);
25         res += query(1, id[top[x]], id[x]);

```



```

26         x = fa[top[x]];
27     }
28     if (dep[x] > dep[y]) swap(x, y);
29     return res += query(1, id[x], id[y]);
30 }
31
32 int main() {
33
34     int rt = 1; dfs(rt, 0); dfs2(rt, rt);
35     build(1, 1, cnt);
36
37     //x对应线段树上的点id[x]
38     //线段树上的点x对应树上点nodeOf[x]
39
40     return;
41 }

```

6.10 树链剖分 (边权)

```

1 //维护边权的树剖
2 //化边权为点权用线段树维护
3 //此处3种操作:
4 //1. 单边修改(修改第k条加入的边)
5 //2. 将(u, v)间的边反转符号
6 //3. 查询(u, v)间边的sum, max, min
7
8 #include <bits/stdc++.h>
9 #define INF 0x3f3f3f3f
10 #define lc u<<1
11 #define rc u<<1|1
12 #define mid (t[u].l+t[u].r)/2
13 using namespace std;
14 typedef long long ll;
15 const int MAX = 2e5 + 10;
16
17 int N, M;
18
19 struct edge {
20     int nxt, to, w, from; //需要额外记录from
21 } e[MAX << 1];
22 int head[MAX], tot; //tot为边数*2
23 void add(int u, int v, int w) { e[++tot] = edge{ head[u], v, w, u }; head[u] = tot; }
24
25 //重链剖分
26 int siz[MAX], son[MAX], dep[MAX], fa[MAX], top[MAX];
27 int id[MAX], cnt; //cnt为点对应在树链上的位置
28 void dfs(int u, int par) {
29     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
30     for (int i = head[u], v; i; i = e[i].nxt)
31         if ((v = e[i].to) != par) {
32             dfs(v, u);
33             siz[u] += siz[v];
34             if (!son[u] || siz[v] > siz[son[u]])

```

```

35         son[u] = v;
36     }
37 }
38 void dfs2(int u, int topf) {
39     id[u] = ++cnt, top[u] = topf;
40     if (!son[u]) return;
41     dfs2(son[u], topf);
42     for (int i = head[u], v; i; i = e[i].nxt)
43         if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
44 }
45 int lca(int x, int y) {
46     while (top[x] != top[y]) {
47         if (dep[top[x]] < dep[top[y]]) swap(x, y);
48         x = fa[top[x]];
49     }
50     return dep[x] < dep[y] ? x : y;
51 }
52
53 struct node { //维护最大/小值, 和
54     int mx, mn, sum;
55     node(int mx = -INF, int mn = INF, int sum = 0): mx(mx), mn(mn), sum(sum) {}
56     node merge(const node &rhs) { return node(max(mx, rhs.mx), min(mn, rhs.mn), sum + rhs.
57         sum); }
58     void upd() { //符号反转
59         sum = -sum;
60         mx = -mx, mn = -mn;
61         swap(mx, mn);
62     };
63 struct SegmentTree {
64     int l, r, tag;
65     node v;
66 } t[MAX << 2];
67 void build(int u, int l, int r) {
68     t[u] = SegmentTree{ l, r, 0, node() };
69     if (l == r) return;
70     build(lc, l, mid); build(rc, mid + 1, r);
71 }
72 void push_up(int u) { t[u].v = t[lc].v.merge(t[rc].v); }
73 void push_down(int u) {
74     if (t[u].tag) {
75         t[lc].v.upd(), t[lc].tag ^= 1;
76         t[rc].v.upd(), t[rc].tag ^= 1;
77         t[u].tag = 0;
78     }
79 }
80 void modify(int u, int p, int k) { //单点修改值
81     if (t[u].l == t[u].r) {
82         t[u].v = node{ k, k, k };
83         return;
84     }
85     push_down(u);
86     if (p <= mid) modify1(lc, p, k);
87     else modify1(rc, p, k);

```

```

88     push_up(u);
89 }
90 void modify2(int u, int ql, int qr) { //反转区间符号
91     if (ql <= t[u].l && t[u].r <= qr) {
92         t[u].v.upd(), t[u].tag ^= 1;
93         return;
94     }
95     push_down(u);
96     if (ql <= mid) modify2(lc, ql, qr);
97     if (qr > mid) modify2(rc, ql, qr);
98     push_up(u);
99
100 }
101 node query(int u, int ql, int qr) { //区间查询
102     if (ql <= t[u].l && t[u].r <= qr) return t[u].v;
103     push_down(u);
104     if (ql <= mid && qr > mid) return query(lc, ql, qr).merge(query(rc, ql, qr));
105     else if (ql <= mid) return query(lc, ql, qr);
106     else return query(rc, ql, qr);
107 }
108
109
110 void update(int x, int y) { //反转(x, y)间所有边的符号
111     while (top[x] != top[y]) {
112         if (dep[top[x]] < dep[top[y]]) swap(x, y);
113         modify2(1, id[top[x]], id[x]);
114         x = fa[top[x]];
115     }
116     if (x != y) {
117         if (dep[x] > dep[y]) swap(x, y);
118         modify2(1, id[son[x]], id[y]);
119     }
120 }
121
122 node ask(int x, int y) { //查询(x, y)间的信息
123     node res;
124     while (top[x] != top[y]) {
125         if (dep[top[x]] < dep[top[y]]) swap(x, y);
126         res = res.merge(query(1, id[top[x]], id[x]));
127         x = fa[top[x]];
128     }
129     if (x != y) {
130         if (dep[x] > dep[y]) swap(x, y);
131         res = res.merge(query(1, id[son[x]], id[y]));
132     }
133     return res;
134 }
135
136 void init() { //多组样例初始化
137     tot = cnt = 0; //tot为边数, cnt为树上的点对应线段树的位置
138     memset(head, 0, sizeof(head));
139     memset(son, 0, sizeof(son));
140 }
141

```

```

142 int main() {
143
144     scanf("%d", &N);
145     for (int i = 1; i < N; i++) {
146         int u, v, w; scanf("%d%d%d", &u, &v, &w);
147         add(u, v, w); add(v, u, w);
148     }
149
150     //预处理
151     int rt = 1; dfs(rt, 0); dfs2(rt, rt); //重链剖分
152     build(1, 1, cnt);
153     for (int i = 1; i <= tot; i += 2) {
154         int u = e[i].from, v = e[i].to;
155         if (dep[u] < dep[v]) swap(u, v); //边对应的点为以rt为根时(u, v)中的儿子节点
156         modify1(1, id[u], e[i].w); //化边权为点权
157     }
158
159     scanf("%d", &M);
160     while (M--) {
161         char op[10]; int x, y; scanf("%s%d%d", op, &x, &y);
162         if (op[0] == 'C') { //单点修改值
163             int u = e[(x << 1) - 1].from, v = e[(x << 1) - 1].to;
164             if (dep[u] < dep[v]) swap(u, v); //边对应的点为以rt为根时(u, v)中的儿子节点
165             modify1(1, id[u], y); //再找到线段树上对应的点id[u]
166         }
167         else if (op[0] == 'N') update(x, y); //反转(x, y)间所有边的符号
168         else { //查询(x, y)间的信息
169             node tmp = ask(x, y);
170             int ans = 0;
171             if (op[0] == 'S') ans = tmp.sum;
172             else if (op[1] == 'A') ans = tmp.mx;
173             else ans = tmp.mn;
174             printf("%d\n", ans);
175         }
176     }
177
178     return 0;
179 }

```

6.11 点分树

```

1 //建立点分树
2
3 struct Grap {
4     struct edge {
5         int nxt, to;
6         ll w;
7     } e[MAX << 1];
8     int head[MAX], tot;
9     void add(int u, int v, int w) { e[++tot] = edge{ head[u], v, w }; head[u] = tot; }
10    //G1为原树, G2点分树
11    //G2中v为当前点到下一个重心w靠近u的点
12 } G1, G2;

```

```

13
14
15 int root, rt;
16 int maxp[MAX], siz[MAX], vis[MAX], fa[MAX];
17 void getRt(int u, int par, int all) { //求树的重心, 将分治复杂度降为  $\log N$ 
18     siz[u] = 1, maxp[u] = 0;
19     for (int i = G1.head[u], v = G1.e[i].to; i; i = G1.e[i].nxt, v = G1.e[i].to)
20         if (v != par && !vis[v]) {
21             getRt(v, u, all);
22             siz[u] += siz[v];
23             maxp[u] = max(maxp[u], siz[v]);
24         }
25     maxp[u] = max(maxp[u], all - siz[u]);
26     if (maxp[u] < maxp[rt]) rt = u;
27 }
28 void rebuild(int u) { //建点分树
29     vis[u] = 1;
30     for (int i = G1.head[u], v = G1.e[i].to; i; i = G1.e[i].nxt, v = G1.e[i].to)
31         if (!vis[v]) {
32             maxp[rt = 0] = N; getRt(v, 0, siz[v]);
33             G2.add(u, v, rt); fa[rt] = u;
34             rebuild(rt);
35         }
36 }
37
38
39 int main() {
40
41     maxp[rt = 0] = N; getRt(1, 0, N); root = rt;
42     rebuild(root);
43
44
45     return 0;
46 }

```

6.12 点分治

```

1 //用于解决树上静态路径统计问题
2 //只需要考虑经过当前树根的两边组成的路径
3 //经过的树根若是不同一定能在当前树的子树中解决
4 //如距离为K/小于K的路径有多少....
5
6 int maxp[N], siz[N], vis[N], rt;
7
8 void getRt(int u, int fa, int all) { //求树的重心
9     siz[u] = 1, maxp[u] = 0;
10    for (int i = head[u], v; i; i = e[i].nxt)
11        if ((v = e[i].to) != fa && !vis[v]) {
12            getRt(v, u, all);
13            siz[u] += siz[v];
14            maxp[u] = max(maxp[u], siz[v]);
15        }
16    maxp[u] = max(maxp[u], all - siz[u]);

```

```

17     if (maxp[u] < maxp[rt]) rt = u;
18 }
19
20 void calc(int u) { //具体题目具体分析
21 //可以用store记录一下当前树所加的路径
22
23 //若路径是有向的, 有时候要扫两边(从前往后, 从后往前)
24 //如某括号题[USACO12NOV]Balanced Trees G(https://www.luogu.com.cn/problem/P3060)
25
26
27 //最后结束的时候清空即可
28 }
29
30 void dfs(int u) {
31     vis[u] = 1;
32     calc(u);
33     for (int i = head[u], v; i; i = e[i].nxt)
34         if (!vis[v = e[i].to]) {
35             maxp[rt = 0] = N; getRt(v, 0, siz[v]);
36             dfs(rt);
37         }
38     vis[u] = 0; //多组清空
39 }
40
41 int main() {
42
43     maxp[rt = 0] = N; getRt(1, 0, N);
44     dfs(rt);
45
46     return 0;
47 }

```

6.13 虚树

```

1 //虚树可以处理多次询问, 并且每次询问只需要树上的K个关键点
2 //建立的虚树能保证点数 < 2 * K
3 //如果对虚树做dp, 总体复杂度和ΣK有关
4 //考虑dp的时候, 需要同时考虑非关键点对答案的影响
5
6 int n;
7
8 struct edge {
9     int nxt, to;
10 } e[N << 1];
11 int head[N], tot;
12 void add(int u, int v) { e[++tot] = edge{ head[u], v }, head[u] = tot; }
13
14 int dep[N], fa[N], topfa[N], siz[N], son[N], dfn[N], cnt;
15 void dfs(int u, int par) {
16     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
17     int max_son = -1;
18     for (int i = head[u], v; i; i = e[i].nxt)
19         if ((v = e[i].to) != par) {

```

```

20         dfs(v, u);
21         siz[u] += siz[v];
22         if (max_son < siz[v]) son[u] = v, max_son = siz[v];
23     }
24 }
25 void dfs2(int u, int topf) {
26     topfa[u] = topf, dfn[u] = ++cnt;
27     if (!son[u]) return;
28     dfs2(son[u], topf);
29     for (int i = head[u], v; i; i = e[i].nxt)
30         if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
31 }
32 int LCA(int x, int y) {
33     while (topfa[x] != topfa[y]) {
34         if (dep[topfa[x]] < dep[topfa[y]]) swap(x, y);
35         x = fa[topfa[x]];
36     }
37     return dep[x] < dep[y] ? x : y;
38 }
39 int getDis(int x, int y) { return dep[x] + dep[y] - 2 * dep[LCA(x, y)]; }
40
41 //建立虚树
42 int tag[N]; //tag[u] = 1 <=> 关键点
43 vector<int> g[N]; //虚树边
44 void add_edge(int u, int v) { g[u].push_back(v); }
45 int st[N], top, rt; //rt为虚树根
46 void insert(int u) {
47     if (top == 1) {
48         st[++top] = u;
49         return;
50     }
51     int lca = LCA(u, st[top]);
52     if (lca != st[top]) {
53         while (top > 1 && dfn[st[top - 1]] >= dfn[lca])
54             add_edge(st[top - 1], st[top]), top--;
55         if (lca != st[top]) add_edge(lca, st[top]), st[top] = lca;
56     }
57     st[++top] = u;
58 }
59 bool cmp(const int &x, const int &y) { return dfn[x] < dfn[y]; }
60 void build(vector<int> &v) {
61     st[top = 1] = rt;
62     sort(v.begin(), v.end(), cmp);
63     for (auto &i: v) {
64         tag[i] = 1;
65         if (i != rt) insert(i);
66     }
67     while (top > 1) add_edge(st[top - 1], st[top]), top--;
68 }
69
70
71 void dp(int u) {
72     //...
73 }

```

```

74 void clear(int u) { //清空虚树边和标记, 也可以和dp合并
75     for (auto &v: g[u]) clear(v);
76     g[u].clear(); tag[u] = 0;
77 }
78 void solve() {
79     //...
80     dp(rt); clear(rt);
81     //...
82 }
83
84 int main() {
85     scanf("%d", &n);
86     for (int i = 1; i < n; i++) {
87         int u, v; scanf("%d%d", &u, &v);
88         add(u, v); add(v, u);
89     }
90     //此处距离为1, 所以用dep替代dis, dis[fa[rt] = 0] = -1
91     dep[0] = -1, rt = 1;
92     dfs(rt, 0); dfs2(rt, rt);
93
94
95     int Q; scanf("%d", &Q);
96     while (Q--) {
97         int K; scanf("%d", &K); //读取关键点
98         for (int i = 1; i <= K; i++) scanf("%d", &a[i]);
99         //构建虚树
100         build(a);
101         solve();
102     }
103
104     return 0;
105 }

```

6.14 轻重链划分

```

1 int siz[N], son[N], dep[N], fa[N], top[N];
2 int id[N], nodeOf[N], cnt; //划分点
3 void dfs(int u, int par) {
4     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
5     for (auto &v: g[u])
6         if (v != par) {
7             dfs(v, u);
8             siz[u] += siz[v];
9             if (!son[u] || siz[v] > siz[son[u]])
10                 son[u] = v;
11         }
12 }
13
14 void dfs2(int u, int topf) {
15     nodeOf[id[u] = ++cnt] = u, top[u] = topf;
16     if (!son[u]) return;
17     dfs2(son[u], topf);
18     for (auto &v: g[u])

```



```
19     if (v != fa[u] && v != son[u]) dfs2(v, v);  
20 }
```