大学生程序设计竞赛算法模板
The ACM-ICPC Algorithm Library

Figure 1: The ACM-ICPC Icon

Figure 2: The BIT Icon

学校名称: 北京理工大学
University: Bejing Institute of Technology

队伍名称: 肥宅大哭.jpg
Owing to team: *fatty newbie*

队内成员: 龙水彬、周赫斌、程苗苗
Teamates: *strawberry*, *zhber*, *miamiao*

完成时间: 2018 年 10 月 15 日
Build at October 15, 2018

Contents

1 字符串 **6**
1.1 KMP . . . 6
1.2 扩展 KMP . . . 7
1.3 manacher . . . 8
1.4 AC 自动机 . . . 9
1.5 后缀数组 . . . 11
1.6 自动机 . . . 14
1.6.1 后缀自动机 . . . 14
1.6.2 回文自动机 . . . 16
1.6.3 前后加字符的回文自动机 . . . 17
1.7 可持久化 Trie . . . 19
1.8 最小表示法 . . . 21

2 数据结构 **24**
2.1 线段树（静态） . . . 24
2.2 线段树（动态开点） . . . 28
2.3 可持久化线段树 . . . 30
2.4 树状数组 . . . 41
2.5 ST 表 . . . 43
2.6 平衡树 . . . 43
2.6.1 Treap . . . 43
2.6.2 Splay . . . 50
2.7 Link-Cut-Tree . . . 57
2.8 树链剖分 . . . 59
2.9 KD 树 . . . 65
2.9.1 一维 KD-Tree . . . 65
2.9.2 高维 KD-Tree . . . 69
2.10 莫队算法 . . . 71
2.11 分块算法 . . . 74
2.12 可持久化并查集 . . . 86

3 图论 **89**
3.1 最小生成树 . . . 89
3.1.1 MST for Kruskal . . . 89
3.1.2 MST for Prim . . . 90
3.2 次小生成树 . . . 90
3.3 最短路 . . . 92
3.3.1 单源最短路 - Dijkstra . . . 92
3.3.2 单源最短路 - SPFA . . . 92
3.3.3 有向图判负环 . . . 93
3.3.4 同余最短路 . . . 94
3.3.5 Floyd 倍增法 . . . 95
3.4 差分约束 . . . 96
3.5 K 短路 . . . 99
3.6 图的割点、桥和双连通分支的基本概念 . . . 101

3.6.1 点连通度与边连通度 . . . 101
3.6.2 双连通图、割点与桥 . . . 101
3.6.3 双联通分支 . . . 101
3.6.4 求割点与桥 . . . 101
3.6.5 求双连通分支 . . . 102
3.6.6 构造双连通图 . . . 102
3.7 有向图强连通分量 . . . 102
3.8 割点、桥 . . . 106
3.9 二分图基础 . . . 107
3.10 有向无环图 (DAG) 基础 . . . 108
3.11 二分图匹配 . . . 108
3.11.1 匈牙利 . . . 108
3.11.2 Hopcroft-Karp . . . 112
3.11.3 多重匹配 . . . 115
3.11.4 最大权匹配 . . . 117
3.12 一般图匹配 . . . 118
3.12.1 带花树算法 . . . 118
3.12.2 最大加权匹配 . . . 121
3.13 2-SAT（二元可满足性问题） . . . 122
3.13.1 2-SAT 染色法（可以得到字典序最小的解） . . . 123
3.13.2 2-SAT 跑强连通分量缩点法（只能得到任意解） . . . 124
3.14 最大流、最小割 . . . 126
3.14.1 Dinic . . . 126
3.14.2 ISAP . . . 127
3.15 费用流 . . . 131
3.15.1 最小费用最大流 . . . 131
3.15.2 zkw 费用流 . . . 132
3.16 有上下界网络流 . . . 134
3.16.1 无源汇可行流 . . . 134
3.16.2 有源汇可行流 . . . 134
3.16.3 有源汇最大流 . . . 134
3.17 二分图匹配最大流算法 . . . 135
3.18 分数规划 . . . 135
3.18.1 二分法 . . . 136
3.18.2 迭代法 . . . 138
3.19 最大权闭合图 . . . 140
3.20 最大密度子图 . . . 141
3.21 最近公共祖先 . . . 142
3.21.1 DFS+ST 表在线算法 . . . 142
3.21.2 离线 +tarjan 算法 . . . 144
3.22 平面图 . . . 146
3.23 欧拉图 . . . 146
3.23.1 有向图 . . . 147
3.23.2 无向图 . . . 149
3.23.3 混合图 . . . 150
3.24 最小树形图 . . . 153

3.25 曼哈顿最小生成树 . . . 154
3.26 生成树计数 . . . 157
3.27 树分治 . . . 160
3.27.1 点分治 . . . 160
3.27.2 边分治 . . . 164

4 搜索 **170**
4.1 折半搜索（Meet in middle） . . . 170
4.2 跳链（Dancing links） . . . 170
4.3 反向搜索 . . . 173
4.4 模拟退火 . . . 177

5 动态规划 **180**
5.1 背包问题 . . . 180
5.2 最长上升子序列 $O(n \cdot log(n))$ . . . 180
5.3 数位 DP . . . 181

6 数学 **185**
6.1 线性筛 . . . 185
6.2 min_25 法筛质数 . . . 187
6.3 非互质逆元 . . . 188
6.4 拆分数 . . . 188
6.5 扩展欧几里得 . . . 189
6.6 随机素数测试 . . . 190
6.7 大整数分解 . . . 191
6.8 中国剩余定理 . . . 194
6.9 离散对数 . . . 195
6.10 原根 . . . 196
6.11 求 A^B 的约数之和对 MOD 取模 . . . 200
6.12 组合数学基础 . . . 200
6.12.1 组合数 . . . 200
6.12.2 排列数 . . . 201
6.12.3 错排数 . . . 201
6.12.4 鸽巢原理 . . . 201
6.12.5 斐波那契数列 . . . 202
6.12.6 特征根 . . . 202
6.12.7 卡特兰数 . . . 202
6.12.8 斯特林数 . . . 203
6.12.9 贝尔数 . . . 203
6.12.10 拆分数 . . . 203
6.12.11 其他结论 . . . 203
6.13 爆破线性递推第 n 项 . . . 203
6.14 矩阵快速幂 . . . 205
6.15 快速幂、快速乘 . . . 208
6.16 高斯消元 . . . 209
6.17 快速傅里叶变换 (FFT) . . . 213
6.18 多项式 . . . 216

6.18.1 函数操作 . . . 216
6.18.2 生成函数 . . . 218
6.18.3 拉格朗日插值 . . . 223
6.19 计算 N 阶线性齐次递推式第 K 项 . . . 224
6.20 单纯形（解决一类线性规划问题） . . . 226
6.21 快速数论变换 (FWT) . . . 230
6.22 线性基 . . . 233
6.23 Polya 定理 . . . 241
6.24 Lindstrom-Gessel-Viennot lemma 定理 . . . 242

7 计算几何 244
7.1 二维几何 . . . 244
7.1.1 通用函数 . . . 244
7.1.2 二维凸包 . . . 266
7.1.3 动态凸包 . . . 268
7.1.4 凸多边形面积并和半平面交 . . . 269
7.1.5 凸多边形面积交 . . . 271
7.1.6 极角排序 . . . 277
7.1.7 圆的内、外心 . . . 278
7.1.8 平面最近点对 . . . 279
7.1.9 最小圆覆盖 . . . 280
7.1.10 自适应辛普森积分 . . . 283
7.2 三维几何 . . . 284
7.2.1 通用函数 . . . 288
7.2.2 三维凸包 . . . 292
7.2.3 最小球覆盖 . . . 297

8 博弈论 299

9 其他 301
9.1 快速输入输出 . . . 301
9.2 程序对拍（Windows 环境、Linux 环境） . . . 302
9.3 快速排序 . . . 302
9.4 桶排序 . . . 303
9.5 分治查询第 K 大 . . . 304
9.6 最大全 0 子矩阵 $O(n^2)$ . . . 305
9.7 高精度 . . . 305
9.8 完全高精度 . . . 307
9.9 子集系列问题 . . . 313
9.9.1 枚举 S 的子集 . . . 313
9.9.2 枚举所有集合的子集 . . . 313
9.9.3 格雷码枚举 2 全集 . . . 313
9.9.4 枚举集合 S 中大小为 r 的子集 . . . 314
9.10 strtok 和 sscanf 结合输入 . . . 314
9.11 解决爆栈问题 . . . 314
9.12 STL 常用操作 . . . 314
9.12.1 优先队列 priority_queue . . . 314

9.12.2 set 和 multiset 315
9.13 Java 类 316
9.13.1 基础语法 316
9.13.2 大整数（BigInteger） 316
9.13.3 高精度（BigDecimal） 318

1 字符串

1.1 KMP

```
/* 例题：POJ - 3461(Oulipo), hdu1711(Number Sequence)
 * 这个模板 字符串是从 0 开始的
 * Next 数组是从 1 开始的
 */
#include <iostream>
#include <cstring>
using namespace std;
const int N = 1000002;
int next_[N];
void getNext(char *pattern, int plen) {
    int j, k;
    j = 0;
    k = -1;
    next_[0] = -1;
    while(j < plen)
        if(k == -1 || pattern[j] == pattern[k]) {
            next_[++j] = ++k; //表示 T[j-1] 和 T[k-1] 相匹配, 当 j 处失配
              ↪ 时, 直接用 next_[j] 处来匹配当前失配处
        } else k = next_[k];
}
/* 返回模式串 T 在主串 S 中首次出现的位置, 返回的位置是从 0 开始的。*/
int KMP_Index(char *str, char *pattern, int slen, int plen) {
    int i = 0, j = 0;
    getNext(pattern, plen);
    while(i < slen && j < plen) {
        if(j == -1 || str[i] == pattern[j]) {
            i++;
            j++;
        } else j = next_[j];
    }
    if(j == plen) return i - plen;
    else return -1;
}
/* 返回模式串在主串 S 中出现的次数 */
int KMP_Count(char *str, char *pattern, int slen, int plen) {
    int ans = 0;
    int i, j = 0;
    if(slen == 1 && plen == 1) {
        if(str[0] == pattern[0])  return 1;
        else  return 0;
    }
    getNext(pattern, plen);
```

```
    for(i = 0; i < slen; i++) {
        while(j > 0 && str[i] != pattern[j])j = next_[j];
        if(str[i] == pattern[j])
            j++;
        if(j == plen) {
            ans++;
            j = next_[j];
        }
    }
    return ans;
}
```

1.2 扩展 KMP

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100010;    //字符串长度最大值
int nxt[maxn], extend[maxn]; //ex 数组即为 extend 数组
//预处理计算 nxt 数组,nxt[i] 代表 i 位置和开始位置的最长公共前缀
void GETNEXT(char *str) {
    int i = 0, j, po, len = strlen(str);
    nxt[0] = len; //初始化 nxt[0]
    while(str[i] == str[i + 1] && i + 1 < len) //计算 nxt[1]
        i++;
    nxt[1] = i;
    po = 1; //初始化 po 的位置
    for(i = 2; i < len; i++) {
        if(nxt[i - po] + i < nxt[po] + po) //第一种情况，可以直接得到 nxt[i] 的值
            nxt[i] = nxt[i - po];
        else { //第二种情况，要继续匹配才能得到 nxt[i] 的值
            j = nxt[po] + po - i;
            if(j < 0) j = 0;    //如果 i>po+nxt[po]，则要从头开始匹配
            while(i + j < len && str[j] == str[j + i]) //计算 nxt[i]
                j++;
            nxt[i] = j;
            po = i; //更新 po 的位置
        }
    }
}
//计算 extend 数组
void EXKMP(char *s1, char *s2) {
    int i = 0, j, po, len = strlen(s1), l2 = strlen(s2);
    while(s1[i] == s2[i] && i < l2 && i < len) //计算 extend[0]
        i++;
    extend[0] = i;
```

```
    po = 0; //初始化 po 的位置
    for(i = 1; i < len; i++) {
        if(nxt[i - po] + i < extend[po] + po) //第一种情况，直接可以得到 extend[i] 的值
            extend[i] = nxt[i - po];
        else { //第二种情况，要继续匹配才能得到 extend[i] 的值
            j = extend[po] + po - i;
            if(j < 0)j = 0; //如果 i>extend[po]+po 则要从头开始匹配
            while(i + j < len && j < l2 && s1[j + i] == s2[j]) //计算 extend[i]
                j++;
            extend[i] = j;
            po = i; //更新 po 的位置
        }
    }
}
char s1[100010], s2[100010];
int main(int argc, char const *argv[]) {
    while(~scanf("%s%s", s1, s2)) {
        GETNEXT(s1);
        EXKMP(s2, s1);
        int len = strlen(s2);
        int ans = 0;
        for(int i = 0; i < len; i++) {
            if(extend[i] + i == len)ans = max(ans, extend[i]);
        }
        s1[ans] = '\0';
        if(ans)
            printf("%s %d\n", s1, ans );
        else printf("%d\n", ans );
    }
    return 0;
}
```

1.3 manacher

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2000005;
char str1[MAXN], str[MAXN];
int ls, p[MAXN];
// transfer string:abcd ==> string:$#a#b#c#d# 预处理
void pre() {
    int ls1 = strlen(str1);
    ls = 0;
    str[ls++] = '$';
```

```
    str[ls++] = '#';
    for (int i = 0; i < ls1; i++) {
        str[ls++] = str1[i];
        str[ls++] = '#';
    }
    str[ls] = 0;
}
// o(n)
int manacher() {
    pre();
    int mx = 0, id, ans = 0; // mx 表示最右的位置，id 表示取到 mx 的中心位置
    for (int i = 1; i < ls; i++) {
        if (mx > i) p[i] = min(p[2 * id - i], mx - i);
        else p[i] = 1;
        for (; str[i + p[i]] == str[i - p[i]]; p[i]++);
        if (p[i] + i > mx) {
            mx = p[i] + i;
            id = i;
        }
        ans = max(ans, p[i]);
    }
    /*
        输出回文串
        p[ans]-=2;
        for(int i = (ans-p[ans])/2-1;i<=(ans+p[ans])/2-1;i++)printf("%c",str1[i]);printf("\n");
    */
    return ans - 1;
}
int main(int argc, char const *argv[]) {
    while(~scanf("%s", str1)) {
        printf("%d\n", manacher());
    }
    return 0;
}
```

1.4 AC 自动机

```
#include <bits/stdc++.h>
using namespace std;
const int AC_NOD = 28, AC_N = 5e5 + 10;
struct Tire {
    int nxt[AC_N][AC_NOD], fail[AC_N], cnt[AC_N];
    int root, tot;
    int id[AC_NOD];
```

```
    int end[AC_N];
    int newnode() {
        for(int i = 0; i < AC_NOD ; i++)nxt[tot][i] = -1;
        fail[tot] = -1;
        cnt[tot] = 0;
        return tot++;
    }
    void init() {
        tot = 0;
        memset(end, false, sizeof end);
        root = newnode();
    }
    int getid(char x) {
        if(x <= 'z' && x >= 'a') return x - 'a' + 10;
        else if(x <= 'Z' && x >= 'A')return 36 + x - 'A';
        else return x - '0';
    }


    void insert(string s) {
        int p = root, len = s.length();
        for(int i = 0; i < len; i++) {
            int cur = getid(s[i]);
            if(nxt[p][cur] == -1)nxt[p][cur] = newnode();
            p = nxt[p][cur];
        }
        cnt[p]++;
        end[p] = true;
    }
    void build() {
        queue<int> que;
        que.push(root);
        int now;

        while(!que.empty()) {
            now = que.front();
            que.pop();
            if(end[fail[now]])end[now] = true;
            for(int i = 0; i < AC_NOD; i++) {
                if(~nxt[now][i]) {
                    if(now == root) fail[nxt[now][i]] = root;
                    else fail[nxt[now][i]] = nxt[fail[now]][i];
                    que.push(nxt[now][i]);
                } else {
                    if(now == root) nxt[now][i] = root;
                    else nxt[now][i] = nxt[fail[now]][i];
```

```
                }
            }
        }
    }
    int count(string t) {
        int now = root;
        int len = t.length(), p, ans = 0;
        for(int i = 0; i < len; i++) {
            p = getid(t[i]), now = nxt[now][p];
            for(int tmp = now; tmp != root; tmp = fail[tmp])ans +=
             ↪  cnt[tmp];
        }
        return ans;
    }
} T;
int main() {
    int t;
    cin.sync_with_stdio(0);
    cin >> t;
    while(t--) {
        T.init();
        int n;
        cin >> n;
        string q;
        for(int i = 1; i <= n; i++)cin >> q, T.insert(q);
        T.build();
        cin >> q;
        cout << T.count(q) << endl;
    }
    return 0;
}
```

1.5 后缀数组

```
/* n 为总字符串的长度
 * r 数组为被操作的字符串，索引从 0 开始，最后一位 (r[n-1]) 一定是'$'(char
↪  数组)||'0'(int 数组)
 * sa 为生成的后缀数组，从 0 开始到 n-1 结束，保证 sa[0] = n-1
 * m 代表字符集的大小或者说范围。
 * height[1] 的值无效因为其没有前驱字符串
 * 查询两个后缀的 lcp 应该查询 query(rnk[l]+1,rnk[r]);
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 300010;
```

```
char a[maxn];
int wa[maxn], wb[maxn], wv[maxn], wts[maxn];
int cmp(int *r, int a, int b, int l) {
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void init_sa(char *r, int *sa, int n, int m) {
    int i, j, *x = wa, *y = wb, *t;
    int p;
    for(i = 0; i < m; i ++) wts[i] = 0;
    for(i = 0; i < n; i ++) wts[ x[i] = r[i]]++;
    for(i = 0; i < m; i ++) wts[i] += wts[i - 1];
    for(i = n - 1; i >= 0; --i) sa[--wts[x[i]]] = i;
    for(j = 1, p = 1; p < n ; j <<= 1, m = p ) {
        for(p = 0, i = n - j; i < n; i ++) y[p++] = i;
        for(i = 0; i < n; i ++) if(sa[i] >= j) y[p++] = sa[i] - j;
        for(i = 0; i < n; i ++) wv[i] = x[y[i]];
        for(i = 0; i < m; i ++) wts[i] = 0;
        for(i = 0; i < n; i ++) wts[wv[i]]++;
        for(i = 1; i < m; i ++) wts[i] += wts[i - 1];
        for(i = n - 1; i >= 0; i--) sa[--wts[wv[i]]] = y[i];
        for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }

    return;
}
int rnk[maxn], height[maxn];
int d[maxn][20], inx[maxn];
void init_inx() {
    inx[0] = -1;
    for(int i = 1; i < maxn; i++)inx[i] = (((i) & (i - 1)) ? 0 : 1) + inx[i - 1];
}
void init_rmq(int *r, int n) {
    for(int i = 0; i < n; i++)d[i][0] = height[i];
    for(int j = 1; j < 20; ++j) {
        int k = 1 << (j - 1);
        for(int i = 0; i + k * 2 - 1 < n; ++i)d[i][j] = min(d[i][j - 1], d[i + k][j - 1]);
    }
}
int query(int l, int r) {
    int k = inx[r - l + 1];
    return min(d[l][k], d[r - (1 << k) + 1][k]);
}
void calheight(char *r, int *sa, int n) {
```

```
    int i, j, k = 0;
    for(i = 1; i < n; i++)rnk[sa[i]] = i;
    for(i = 0; i < n - 1; height[rnk[i++]] = k)
        for(k ? k-- : 0, j = sa[rnk[i] - 1]; r[i + k] == r[j + k]; k++);
}
int sa[maxn];
void solve() {
    scanf("%s", a);
    int len = strlen(a);
    a[len] = '$';
    len++;
    int n = len;
    scanf("%s", a + len);
    len = strlen(a);
    a[len] = '#';
    len++;
    init_sa(a, sa, len, 256);
    calheight(a, sa, len);
    init_rmq(height, len);
    int r = 2;
    int ans = 0;
    for(int i = 2; i < len; i++) {
        r = max(r, i);
        if(sa[i] >= n)continue;
        while(r < len && sa[r] < n)r++;
        if(r == len)break;
        ans = max(ans, query(i + 1, r));
    }
    r = 2;
    for(int i = 2; i < len; i++) {
        r = max(r, i);
        if(sa[i] < n)continue;
        while(r < len && sa[r] >= n)r++;
        if(r == len)break;
        ans = max(ans, query(i + 1, r));
    }
    printf("%d\n", ans );
}
int main(int argc, char const *argv[]) {
    init_inx();
    solve();
    return 0;
}
```

1.6 自动机

1.6.1 后缀自动机

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000010;
struct Sam {
    int n, tot, last, ans, l;
    int nxt[maxn << 1][26], pa[maxn << 1], dep[maxn << 1];
    int root;
    char s[maxn];
    inline void init() {
        memset(vis, 0, sizeof vis);
        root = tot = last = 1;
        memset(nxt, -1, sizeof nxt);
        memset(pa, -1, sizeof pa);
        memset(dep, 0, sizeof dep);
    }
    inline void extend(int x) {
        int c = s[x] - 'a', p = last, np = ++tot;
        last = np;
        dep[np] = x;
        for(; p != -1 && nxt[p][c] == -1; p = pa[p])nxt[p][c] = np;
        if(p == -1)pa[np] = 1;
        else {
            int q = nxt[p][c];
            if(dep[p] + 1 == dep[q])pa[np] = q;
            else {
                int nq = ++tot;
                dep[nq] = dep[p] + 1;
                memcpy(nxt[nq], nxt[q], sizeof nxt[q]);
                pa[nq] = pa[q];
                pa[np] = pa[q] = nq;
                for(; nxt[p][c] == q; p = pa[p])nxt[p][c] = nq;
            }
        }
    }
    int res[maxn << 1];
    int w[maxn << 1], b[maxn << 1], r[maxn << 1];
    void topo() {
        memset(w, 0, sizeof w);
        memset(r, 0, sizeof r);
        //拓扑排序
        for(int i = 1; i <= tot; i++)w[dep[i]]++;
        for(int i = 1; i <= l; i++) w[i] += w[i - 1];
        for(int i = 1; i <= tot; i++)b[w[dep[i]]--] = i;
```

```
        //拓扑排序

        //计算每个状态 right 的 size
        int p = root;
        for(int i = 1; i <= l; i++)p = nxt[p][s[i] - 'a'], r[p]++;
        for(int i = tot; i >= 1; i--) if( pa[b[i]] != -1 ) r[ pa[b[i]] ] += r[b[i]];
    }

    int vis[maxn << 1];
    /* CodeForces 235C */
    void query(char a[], int len, int now) {
        int p = root;
        int ans = 0;
        int t = 0;
        p = root;
        for(int i = 0; i < len; i++)if(nxt[p][a[i] - 'a'] == -1) {
                printf("0\n");
                return ;
            }
        for(int i = 0; i < len; i++) {
            while( p > 1 && nxt[p][a[i] - 'a'] == -1) {
                p = pa[p];
                t = dep[p];
            }
            p = nxt[p][a[i] - 'a'];
            t++;
            if(t >= len / 2) {
                while(pa[p] != -1 && dep[pa[p]] >= len / 2 )p = pa[p], t = dep[p];
                if(vis[p] != now)vis[p] = now, ans += r[p];
            }
        }
        printf("%d\n", ans);
    }
    //查询子串第 k 大
    inline void query(int k) {
        int p = root;
        while(k) {
            for(int i = 0; i < 26; i++) {
                if(nxt[p][i] == -1)continue;
                if(k > r[nxt[p][i]]) k -= r[nxt[p][i]];
                else if(k <= r[nxt[p][i]]) {
                    printf("%c", i + 'a');
                    p = nxt[p][i];
                    break;
```

```
                }
            }
        }
        printf("\n");
    }
} S;
int main(int argc, char const *argv[]) {
    scanf("%s", S.s + 1);
    int len = strlen(S.s + 1);
    S.l = len;
    S.init();
    for(int i = 1; i <= len; i++) S.extend(i);
    S.topo();
    int k;
    scanf("%d", &k), S.query(k);
    return 0;
}
```

1.6.2 回文自动机

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5e5 + 10;
typedef long long ll;
// 每个节点代表一个回文串
int len[maxn];// 此节点回文串长度
int nxt[maxn][26];
int fail[maxn];
int cnt[maxn];
int num[maxn];//border 个数
int s[maxn];
int last;
int p;// 节点个数
int n;// 已添加字符个数
int newnode() {
    for(int i = 0; i < 26; i++)nxt[p][i] = 0;
    cnt[p] = num[p] = 0;
    return p++;
}
void init() {
    p = 0;
    len[0] = 0;
    len[1] = -1;
    fail[0] = 1;
    newnode();
    newnode();
```

```
    n = 0;
    s[0] = -1;// 开头放一个字符集中没有的字符，减少特判
    last = 1;
}
int get_fail(int x) {
    while(s[n - len[x] - 1] != s[n])x = fail[x];
    return x;
}
void add(char c) {
    int x = c - 'a';
    s[++n] = x;
    last = get_fail(last);
    int cur = last;
    if(!nxt[cur][x]) {
        int now = newnode();
        len[now] = len[cur] + 2;
        //以下两行因为边界原因，顺序不可互换
        fail[now] = nxt[get_fail(fail[cur])][x];
        nxt[cur][x] = now;
        num[now] = num[fail[now]] + 1;
    }
    last = nxt[cur][x];
    cnt[last]++;
}
char ss[maxn];
ll solve() {
    ll ans = 0;
    for(int i = p - 1; i >= 0; i--)cnt[fail[i]] += cnt[i];
    for(int i = 0; i < p; i++)ans = max(ans, 1ll * cnt[i] * len[i]);
    return ans;
}
int main(int argc, char const *argv[]) {
    scanf("%s", ss + 1);
    int len = strlen(ss + 1);
    init();
    for(int i = 1; i <= len; i++)add(ss[i]);
    printf("%lld\n", solve());
    return 0;
}
```

1.6.3 前后加字符的回文自动机

```
#include <bits/stdc++.h>
using namespace std;
const int mod = 19930726;
typedef long long ll;
```

```
const int LEN = 2e5 + 10;
int len[LEN], fail[LEN], num[LEN];
const int mid = 100003;
int nxt[LEN][26];
int last[2], p, npre, nend;
char s[LEN];
ll ans;
int newnode() {
    memset(nxt[p], 0, sizeof nxt[p]);
    fail[p] = 0;
    return p++;
}
void init() {
    p = 0;
    ans = 0;
    len[0] = 0;
    len[1] = -1;
    newnode();
    newnode();
    fail[0] = 1;
    npre = nend = mid;
    npre++;
    s[npre] = s[nend] = -1;
    last[0] = 1;
    last[1] = 1;
}
int getfail(int x, int k) {
    if(k == 0)while(s[nend - (len[x] + 1)] != s[nend])x = fail[x];
    else while(s[npre + (len[x] + 1)] != s[npre])x = fail[x];
    return x;
}
void add(char c, int q) {
    int x = c - 'a';
    last[q] = getfail(last[q], q);
    int cur = last[q];
    if(!nxt[cur][x]) {
        int now = newnode();
        len[now] = len[cur] + 2;
        fail[now] = nxt[getfail(fail[cur], q)][x];
        nxt[cur][x] = now;
        num[now] = num[fail[now]] + 1;
    }
    last[q] = nxt[cur][x];
    if(len[last[q]] == nend - npre + 1)last[q ^ 1] = last[q];
    ans += num[last[q]];
}
```

```
void addpre(char c) {
    s[++nend] = c - 'a';
    s[nend + 1] = -1;
    add(c, 0);
}
void addend(char c) {
    s[--npre] = c - 'a';
    s[npre - 1] = -1;
    add(c, 1);
}
pair<int, int> a[LEN];
int main(int argc, char const *argv[]) {
    int m, op;
    while(~scanf("%d", &m)) {
        init();
        char a[10];
        while(m--) {
            scanf("%d", &op);
            if(op == 1) {
                scanf("%s", a);
                addpre(a[0]);
            } else if(op == 2) {
                scanf("%s", a);
                addend(a[0]);
            } else if(op == 3)printf("%d\n", p - 2);
            else printf("%lld\n", ans);

        }
    }
    return 0;
}
```

1.7 可持久化 Trie

```
/* 例题：BZOJ 3261 - 给定一个非负整数序列 {a}, 初始长度为 N。
 * 有 M 个操作，有以下两种操作类型：
     1. A x：添加操作，表示在序列末尾添加一个数 x，序列的长度 N+1。
     2. Q l r x：询问操作，你需要找到一个位置 p，满足 l<=p<=r，使得：a[p] xor a[p+1] xor ... xor a[N] xor x 最大，输出最大是多少。
 */
#include <bits/stdc++.h>
using namespace std;
struct Trie {
#define NODE 20000005
#define SET 2
    int nxt[NODE][SET];
```

```
    int cnt[NODE], tot;
    int root[600100];
    int nw;
    int newnode() {
        for(int i = 0; i < SET; i++)nxt[tot][i] = -1;
        cnt[tot] = 0;
        return tot++;
    }
    void init() {
        tot = 0;
        root[0] = newnode();
        nw = 0;
    }
    void insert(int x) {
        int dig[50] = {0};
        for(int i = 1; i <= 25; i++)dig[25 - i + 1] = x % 2, x /= 2;
        nw++;
        root[nw] = newnode();
        int p = root[nw], pre = root[nw - 1];
        for(int i = 0; i < SET; i++)nxt[p][i] = nxt[pre][i];
        for(int i = 1; i <= 25; i++) {
            int cur = dig[i];
            nxt[p][cur] = newnode();
            p = nxt[p][cur];
            if(~pre && ~nxt[pre][cur]) {
                pre = nxt[pre][cur];
                for(int j = 0; j < SET; j++)nxt[p][j] = nxt[pre][j];
                cnt[p] = cnt[pre] + 1;
            } else cnt[p] = 1, pre = -1;
        }
    }
    int query(int l, int r, int x) {
        int pl = root[l - 1], pr = root[r];
        int dig[50] = {0};
        for(int i = 1; i <= 25; i++)dig[25 - i + 1] = x % 2, x /= 2;
        int a, b, ans = 0;
        for(int i = 1; i <= 25; i++) {
            int cur = dig[i];
            if(~pl && ~nxt[pl][cur ^ 1])a = cnt[nxt[pl][cur ^ 1]];
            else a = 0;
            if(~nxt[pr][cur ^ 1])b = cnt[nxt[pr][cur ^ 1]];
            else b = 0;
            if(b - a) {
                if(~pl)pl = nxt[pl][cur ^ 1];
                pr = nxt[pr][cur ^ 1];
                ans <<= 1;
```

```
                ans++;
            } else {
                if(~pl)pl = nxt[pl][cur];
                pr = nxt[pr][cur];
                ans = (ans << 1);
            }
        }
        return ans;
    }
} T;
int main(int argc, char const *argv[]) {
    int n, m;
    scanf("%d%d", &n, &m);
    T.init();
    int x;
    int tot = 0, total = 0;
    T.insert(0);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &x);
        tot ^= x;
        T.insert(tot);
    }
    int lastans = 0, l, r;
    char s[10];
    for(int i = 1; i <= m; i++) {
        scanf("%s", s);
        if(s[0] == 'A') {
            scanf("%d", &x);
            tot ^= x;
            T.insert(tot);
        } else {
            scanf("%d%d%d", &l, &r, &x);
            printf("%d\n", T.query(l, r, tot ^ x));
        }
    }
    return 0;
}
```

1.8 最小表示法

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000010;
char a[maxn];
int Next[maxn], num, L;
void KMP() {
```

```
    for(int i = 2, j = 0; i <= L; i++) {
        while(j && a[i] != a[j + 1]) j = Next[j];
        if(a[i] == a[j + 1]) j++;
        Next[i] = j;
    }
    num = L % (L - Next[L]) ? 1 : L / (L - Next[L]);
}
int minexpress() {
    int i = 1, j = 2, k = 0;
    while(i <= L && j <= L && k < L) {
        int ti = (i + k > L ? i + k - L : i + k);
        int tj = j + k > L ? j + k - L : j + k;
        if(a[ti] == a[tj]) k++;
        else {
            if(a[ti] < a[tj]) j += k + 1;
            else i += k + 1;
            if(i == j) j++;
            k = 0;
        }
    }
    return min(i, j);
}
int maxexpress() {
    int i = 1, j = 2, k = 0;
    while(i <= L && j <= L && k < L) {
        int ti = i + k > L ? i + k - L : i + k;
        int tj = j + k > L ? j + k - L : j + k;
        if(a[ti] == a[tj]) k++;
        else {
            if(a[ti] > a[tj]) j += k + 1;
            else i += k + 1;
            if(i == j) j++;
            k = 0;
        }
    }
    return min(i, j);
}
int main() {
    int T, Min, Max;
    while(~scanf("%s", a + 1)) { // 字符串从 1 开始
        L = strlen(a + 1);
        KMP();
        Min = minexpress();
        Max = maxexpress();
        printf("%d %d %d %d\n", Min, num, Max, num);
    }
```

```
    return 0;
}
```

2 数据结构

2.1 线段树（静态）

```
// 这是一个区间加和 RMQ
class SegmentTree {
public:
#define lson (root << 1)
#define rson (root << 1 | 1)
#define lent (t[root].r - t[root].l + 1)
#define lenl (t[lson].r - t[lson].l + 1)
#define lenr (t[rson].r - t[rson].l + 1)
	struct Tree {
		int l, r, val, lazy;
	} t[maxn << 4];

	void pushup(int root) {
		t[root].val = t[lson].val + t[rson].val;
	}

	void pushdown(int root) {
		if (t[root].lazy) {
			t[lson].lazy += t[root].lazy;
			t[rson].lazy += t[root].lazy;
			t[lson].val += lenl * t[root].lazy;
			t[rson].val += lenr * t[root].lazy;
			t[root].lazy = 0;
		}
	}

	void build(int l, int r, int root) {
		t[root].l = l;
		t[root].r = r;
		t[root].lazy = 0;
		if (l == r) {
			t[root].val = 0;
			return;
		}
		int mid = l + r >> 1;
		build(l, mid, lson);
		build(mid + 1, r, rson);
		pushup(root);
	}

	void update(int l, int r, int val, int root) {
		if (l <= t[root].l && t[root].r <= r) {
```

```
                        t[root].val += lent * val;
                        t[root].lazy += val;
                        return;
                }
                pushdown(root);
                int mid = t[root].l + t[root].r >> 1;
                if (l <= mid) update(l, r, val, lson);
                if (r > mid) update(l, r, val, rson);
                pushup(root);
        }

        int query(int l, int r, int root) {
                if (l <= t[root].l && t[root].r <= r)
                        return t[root].val;
                pushdown(root);
                int mid = t[root].l + t[root].r >> 1;
                int ans = 0;
                if (l <= mid) ans += query(l, r, lson);
                if (r > mid) ans += query(l, r, rson);
                return ans;
        }
#undef lenr
#undef lenl
#undef lent
#undef rson
#undef lson
};
// new HDU-5828
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
int a[maxn];
class SegmentTree {
public:
#define ls (rt<<1)
#define rs (rt<<1|1)
    struct Tree {
        int l, r;
        ll mi, mx, tag, sum;
    } t[maxn << 2];

    inline int treelen(int rt) { return t[rt].r - t[rt].l + 1; }

    inline void push_up(int rt) {
        t[rt].mi = min(t[ls].mi, t[rs].mi);
```

```
        t[rt].mx = max(t[ls].mx, t[rs].mx);
        t[rt].sum = t[ls].sum + t[rs].sum;
    }

    inline void push_down(int rt) {
        if (t[rt].tag) {
            t[ls].tag += t[rt].tag;
            t[rs].tag += t[rt].tag;
            t[ls].mi += t[rt].tag;
            t[rs].mi += t[rt].tag;
            t[ls].mx += t[rt].tag;
            t[rs].mx += t[rt].tag;
            t[ls].sum += treelen(ls) * t[rt].tag;
            t[rs].sum += treelen(rs) * t[rt].tag;
            t[rt].tag = 0;
        }
    }

    void build(int l, int r, int rt) {
        t[rt].l = l,  t[rt].r = r, t[rt].tag = 0;
        if (l == r) {
            t[rt].mi = t[rt].mx = t[rt].sum = a[l];
            return;
        }
        int m = l + r >> 1;
        build(l, m, ls);
        build(m + 1, r, rs);
        push_up(rt);
    }

    void update(int l, int r, int rt) {
        if (l <= t[rt].l && t[rt].r <= r) {
            if (t[rt].mi == t[rt].mx) {
                ll tmp = sqrt(t[rt].mx + 0.5);
                ll sub = t[rt].mx - tmp;
                t[rt].sum -= sub * treelen(rt);
                t[rt].tag -= sub;
                t[rt].mi = t[rt].mx = tmp;
                return;
            } else if (t[rt].mi == t[rt].mx - 1) {
                ll tmp1 = sqrt(t[rt].mx + 0.5);
                ll tmp2 = sqrt(t[rt].mi + 0.5);
                if (tmp2 == tmp1 - 1) {
                    ll sub = t[rt].mx - tmp1;
                    t[rt].sum -= sub * treelen(rt);
                    t[rt].tag -= sub;
```

```
                    t[rt].mx = tmp1, t[rt].mi = tmp2;
                    return;
                }
            }
        }
        push_down(rt);
        int m = t[rt].l + t[rt].r >> 1;
        if (l <= m) update(l, r, ls);
        if (r > m) update(l, r, rs);
        push_up(rt);
    }

    void update(int l, int r, int x, int rt) {
        if (l <= t[rt].l && t[rt].r <= r) {
            t[rt].sum +=  1LL * treelen(rt) * x;
            t[rt].mi += x;
            t[rt].mx += x;
            t[rt].tag += x;
            return;
        }
        push_down(rt);
        int m = t[rt].l + t[rt].r >> 1;
        if (l <= m) update(l, r, x, ls);
        if (r > m) update(l, r, x, rs);
        push_up(rt);
    }

    ll query(int l, int r, int rt) {
        if (l <= t[rt].l && t[rt].r <= r) return t[rt].sum;
        push_down(rt);
        int m = t[rt].l + t[rt].r >> 1;
        ll ans = 0;
        if (l <= m) ans += query(l, r, ls);
        if (r > m) ans += query(l, r, rs);
        return ans;
    }
#undef ls
#undef rs
#undef lt
#undef ll
#undef lr
} T;
int main() {
    int t, n, m;
    scanf("%d", &t);
    while (t--) {
```

```
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        T.build(1, n, 1);
        while (m--) {
            int op, l, r, x;
            scanf("%d%d%d", &op, &l, &r);
            if (op == 1) {
                scanf("%d", &x);
                T.update(l, r, x, 1);
            } else if (op == 2) T.update(l, r, 1);
            else printf("%lld\n", T.query(l, r, 1));
        }
    }
    return 0;
}
```

2.2 线段树（动态开点）

```
/* 动态开点类似于主席树，每次开点需要记录每个点的儿子是谁
 * 例题：hdu6183 - 二维矩形，4 种操作，每次可以清除所有点、给点 (x,y) 添加
 ↪ 一种颜色 c、查询在 (0,y1) 与 (x,y2) 所围成的矩形里有多少种颜色
 * 解法：因为每次查询的矩阵都是从 x=0 开始到 x，因此只需要维护 1 颗关于 y
 ↪ 轴值域的 50 颗关于颜色的线段树即可，对于每个 y_i，存这个颜色下出现的最
 ↪ 小的 x ，查询是否 mi[rt] <= x 即可 ，但是因为数据范围较大，但实际上点数
 ↪ 不多，需要动态开点或者离线以后离散化
 */
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 5;
int up = 1e6;
int col[52], ls[maxn << 5], rs[maxn << 5], mi[maxn << 5], idx;
void insert(int &o, int x, int y, int l, int r) {
    if (o == 0) { o = ++idx; mi[o] = x; }
        if (mi[o] == 0 || mi[o] > x) mi[o] = x;
        if (l == r) return;
        int m = (l + r) / 2;
        if (y <= m) insert(ls[o], x, y, l, m);
        else insert(rs[o], x, y, m+1, r);
}
bool query(int o, int x, int y1, int y2, int l, int r) {
    if (!o) return 0;
    if (y1 <= l && r <= y2) return mi[o] <= x;
    int m = (l + r) / 2;
    bool ok = 0;
    if (y1 <= m) ok = query(ls[o], x, y1, y2, l, m);
    if (!ok && y2 > m) ok = query(rs[o], x, y1, y2, m+1, r);
```

```
    return ok;
}
int main() {
    int op;
    idx = 0;
    while (~scanf("%d", &op) && op != 3) {
        if (op == 0) {
            memset(col, 0, sizeof col);
            memset(ls, 0, sizeof ls);
            memset(rs, 0, sizeof rs);
            memset(mi, 0, sizeof mi);
        } else if (op == 1) {
            int x, y, c;
            scanf("%d%d%d", &x, &y, &c);
            insert(col[c], x, y, 1, up);
        } else if (op == 2) {
            int x, y1, y2;
            scanf("%d%d%d", &x, &y1, &y2);
            int ans = 0;
            for (int i = 0; i <= 50; i++) ans += query(col[i], x, y1, y2, 1, up);
            printf("%d\n", ans);
        }
    }
    return 0;
}
/* 例题 2：Codeforces 915E - 区间修改、区间查询
 * 解法：n 很大，但是 q 很小，用到的点很少，因此线段树动态开点，注意内存 10 倍以上？
 */
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e7 + 5e6 + 5;
int ls[(maxn], rs[maxn], sum[maxn], tag[maxn], idx, rt;
void init() {
    memset(ls, 0, sizeof ls);
    memset(rs, 0, sizeof rs);
    memset(sum, 0, sizeof sum);
    memset(tag, 0, sizeof tag);
    rt = idx = 0;
}
void push_up(int o) {
    sum[o] = sum[ls[o]] + sum[rs[o]];
}
void push_down(int l, int r, int o) {
    if (tag[o] != -1) {
```

```
        int m = (l + r) / 2;
        if (l != r) {
            if (!ls[o]) ls[o] = ++idx;
            if (!rs[o]) rs[o] = ++idx;
            tag[ls[o]] = tag[o];
            tag[rs[o]] = tag[o];
            sum[ls[o]] = (m - l + 1) * tag[o];
            sum[rs[o]] = (r - m) * tag[o];
        }
        tag[o] = -1;
    }
}
void update(int ql, int qr, int v, int l, int r, int &o) {
    if (!o) o = ++idx;
    if (ql <= l && r <= qr) {
        if (l != r) tag[o] = v;
        sum[o] = v * (r - l + 1);
        return;
    }
    push_down(l, r, o);
    int m = (l + r) / 2;
    if (ql <= m) update(ql, qr, v, l, m, ls[o]);
    if (qr > m) update(ql, qr, v, m + 1, r, rs[o]);
    push_up(o);
}
int main() {
    int n, q;
    scanf("%d%d", &n, &q);
    init();
    while (q--) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        if (k == 1) update(l, r, 1, 1, n, rt);
        else if (k == 2) update(l, r, 0, 1, n, rt);
        printf("%d\n", n - sum[rt]);
    }
    return 0;
}
```

2.3 可持久化线段树

```
/* 例题 1：SPOJ - DQUERY 查询区间有多少个不同的数
 */
const int MAXN = 30010;
const int M = MAXN * 100;
int n, q, tot;
```

```
int a[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
int build(int l, int r) {
    int root = tot++;
    c[root] = 0;
    if(l != r) {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}
int update(int root, int pos, int val) {
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = n;
    while(l < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        } else {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int root, int pos) {
    int ret = 0;
    int l = 1, r = n;
    while(pos < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            r = mid;
            root = lson[root];
        } else {
            ret += c[lson[root]];
            root = rson[root];
```

```
                l = mid + 1;
            }
        }
        return ret + c[root];
}
int main() {
    while(scanf("%d", &n) == 1) {
        tot = 0;
        for(int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        T[n + 1] = build(1, n);
        map<int, int>mp;
        for(int i = n; i >= 1; i--) {
            if(mp.find(a[i]) == mp.end()) {
                T[i] = update(T[i + 1], i, 1);
            } else {
                int tmp = update(T[i + 1], mp[a[i]], -1);
                T[i] = update(tmp, i, 1);
            }
            mp[a[i]] = i;
        }
        scanf("%d", &q);
        while(q--) {
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", query(T[l], r));
        }
    }
    return 0;
}
/* 例题 2：POJ2104 - 静态区间第 K 大
 */
const int MAXN = 100010;
const int M = MAXN * 30;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash() {
    for(int i = 1; i <= n; i++)
        t[i] = a[i];
    sort(t + 1, t + 1 + n);
    m = unique(t + 1, t + 1 + n) - t - 1;
}
int build(int l, int r) {
    int root = tot++;
    c[root] = 0;
```

```
    if(l != r) {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}
int hash(int x) {
    return lower_bound(t + 1, t + 1 + m, x) - t;
}
int update(int root, int pos, int val) {
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while(l < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        } else {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int left_root, int right_root, int k) {
    int l = 1, r = m;
    while( l < r) {
        int mid = (l + r) >> 1;
        if(c[lson[left_root]] - c[lson[right_root]] >= k ) {
            r = mid;
            left_root = lson[left_root];
            right_root = lson[right_root];
        } else {
            l = mid + 1;
            k -= c[lson[left_root]] - c[lson[right_root]];
            left_root = rson[left_root];
            right_root = rson[right_root];
```

```
        }
    }
    return l;
}
int main() {
    while(scanf("%d%d", &n, &q) == 2) {
        tot = 0;
        for(int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        Init_hash();
        T[n + 1] = build(1, m);
        for(int i = n; i ; i--) {
            int pos = hash(a[i]);
            T[i] = update(T[i + 1], pos, 1);
        }
        while(q--) {
            int l, r, k;
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", t[query(T[l], T[r + 1], k)]);
        }
    }
    return 0;
}
/* 例题 3：SPOJ - COT 树上路径点权第 K 大
 * 解法：LCA + 主席树
 */
// 主席树部分 *****************
const int MAXN = 200010;
const int M = MAXN * 40;
int n, q, m, TOT;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash() {
    for(int i = 1; i <= n; i++)
        t[i] = a[i];
    sort(t + 1, t + 1 + n);
    m = unique(t + 1, t + n + 1) - t - 1;
}
int build(int l, int r) {
    int root = TOT++;
    c[root] = 0;
    if(l != r) {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
```

```
    return root;
}
int hash(int x) {
    return lower_bound(t + 1, t + 1 + m, x) - t;
}
int update(int root, int pos, int val) {
    int newroot = TOT++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while( l < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            lson[newroot] = TOT++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        } else {
            rson[newroot] = TOT++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int left_root, int right_root, int LCA, int k) {
    int lca_root = T[LCA];
    int pos = hash(a[LCA]);
    int l = 1, r = m;
    while(l < r) {
        int mid = (l + r) >> 1;
        int tmp = c[lson[left_root]] + c[lson[right_root]] - 2 * c[
                        lson[lca_root]] + (pos >= l && pos <= mid);
        if(tmp >= k) {
            left_root = lson[left_root];
            right_root = lson[right_root];
            lca_root = lson[lca_root];
            r = mid;
        } else {
            k -= tmp;
            left_root = rson[left_root];
            right_root = rson[right_root];
            lca_root = rson[lca_root];
```

```
            l = mid + 1;
        }
    }
    return l;
}
// LCA 部分
int rmq[2 * MAXN]; // rmq 数组，就是欧拉序列对应的深度序列
struct ST {
    int mm[2 * MAXN];
    int dp[2 * MAXN][20]; // 最小值对应的下标
    void init(int n) {
        mm[0] = -1;
        for(int i = 1; i <= n; i++) {
            mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
            dp[i][0] = i;
        }
        for(int j = 1; j <= mm[n]; j++)
            for(int i = 1; i + (1 << j) - 1 <= n; i++)
                dp[i][j] = rmq[dp[i][j - 1]] < rmq[dp[i + (1 << (j - 1))][j - 1]] ? dp[i][j - 1] : dp[i + (1 << (j - 1))][j - 1];
    }
    // 查询 [a,b] 之间最小值的下标
    int query(int a, int b) {
        if(a > b)swap(a, b);
        int k = mm[b - a + 1];
        return rmq[dp[a][k]] <= rmq[dp[b - (1 << k) + 1][k]] ? dp[a][k] : dp[b - (1 << k) + 1][k];
    }
};
// 边的结构体定义
struct Edge {
    int to, next;
};
Edge edge[MAXN * 2];
int tot, head[MAXN];
int F[MAXN * 2]; // 欧拉序列，就是 dfs 遍历的顺序，长度为 2*n-1，下标从 1 开始
int P[MAXN];// P[i] 表示点 i 在 F 中第一次出现的位置
ST st;
void init() {
    tot = 0;
    memset(head, -1, sizeof(head));
}
// 加边，无向边需要加两次
void addedge(int u, int v) {
```

```
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void dfs(int u, int pre, int dep) {
    F[++cnt] = u;
    rmq[cnt] = dep;
    P[u] = cnt;
    for(int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].to;
        if(v == pre)continue;
        dfs(v, u, dep + 1);
        F[++cnt] = u;
        rmq[cnt] = dep;
    }
}
// 查询 LCA 前的初始化
void LCA_init(int root, int node_num) {
    cnt = 0;
    dfs(root, root, 0);
    st.init(2 * node_num - 1);
}
// 查询 u,v 的 lca 编号
int query_lca(int u, int v) {
    return F[st.query(P[u], P[v])];
}
void dfs_build(int u, int pre) {
    int pos = hash(a[u]);
    T[u] = update(T[pre], pos, 1);
    for(int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].to;
        if(v == pre)continue;
        dfs_build(v, u);
    }
}
int main() {
    while(scanf("%d%d", &n, &q) == 2) {
        for(int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        Init_hash();
        init();
        TOT = 0;
        int u, v;
        for(int i = 1; i < n; i++) {
            scanf("%d%d", &u, &v);
            addedge(u, v);
```

```
                addedge(v, u);
            }
            LCA_init(1, n);
            T[n + 1] = build(1, m);
            dfs_build(1, n + 1);
            int k;
            while(q--) {
                scanf("%d%d%d", &u, &v, &k);
                printf("%d\n", t[query(T[u], T[v], query_lca(u, v), k)]);
            }
            return 0;
        }
        return 0;
}
/* 例题 3：ZOJ2112 - 动态第 K 大（树状数组套主席树）
 */
const int MAXN = 60010;
const int M = 2500010;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
int S[MAXN];
struct Query {
    int kind;
    int l, r, k;
} query[10010];
void Init_hash(int k) {
    sort(t, t + k);
    m = unique(t, t + k) - t;
}
int hash(int x) {
    return lower_bound(t, t + m, x) - t;
}
int build(int l, int r) {
    int root = tot++;
    c[root] = 0;
    if(l != r) {
        int mid = (l + r) / 2;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}
int Insert(int root, int pos, int val) {
    int newroot = tot++, tmp = newroot;
    int l = 0, r = m - 1;
```

```
    c[newroot] = c[root] + val;
    while(l < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        } else {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int lowbit(int x) {
    return x & (-x);
}
int use[MAXN];
void add(int x, int pos, int val) {
    while(x <= n) {
        S[x] = Insert(S[x], pos, val);
        x += lowbit(x);
    }
}
int sum(int x) {
    int ret = 0;
    while(x > 0) {
        ret += c[lson[use[x]]];
        x -= lowbit(x);
    }
    return ret;
}
int Query(int left, int right, int k) {
    int left_root = T[left - 1];
    int right_root = T[right];
    int l = 0, r = m - 1;
    for(int i = left - 1; i; i -= lowbit(i)) use[i] = S[i];
    for(int i = right; i ; i -= lowbit(i)) use[i] = S[i];
    while(l < r) {
        int mid = (l + r) / 2;
```

```
        int tmp = sum(right) - sum(left - 1) + c[lson[right_root]] -
        ↪  c[lson[left_root]];
        if(tmp >= k) {
            r = mid;
            for(int i = left - 1; i ; i -= lowbit(i))
                use[i] = lson[use[i]];
            for(int i = right; i; i -= lowbit(i))
                use[i] = lson[use[i]];
            left_root = lson[left_root];
            right_root = lson[right_root];
        } else {
            l = mid + 1;
            k -= tmp;
            for(int i = left - 1; i; i -= lowbit(i))
                use[i] = rson[use[i]];
            for(int i = right; i ; i -= lowbit(i))
                use[i] = rson[use[i]];
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}
void Modify(int x, int p, int d) {
    while(x <= n) {
        S[x] = Insert(S[x], p, d);
        x += lowbit(x);
    }
}
int main() {
    int Tcase;
    scanf("%d", &Tcase);
    while(Tcase--) {
        scanf("%d%d", &n, &q);
        tot = 0;
        m = 0;
        for(int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            t[m++] = a[i];
        }
        char op[10];
        for(int i = 0; i < q; i++) {
            scanf("%s", op);
            if(op[0] == 'Q') {
                query[i].kind = 0;
                scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
```

```
            } else {
                query[i].kind = 1;
                scanf("%d%d", &query[i].l, &query[i].r);
                t[m++] = query[i].r;
            }
        }
        Init_hash(m);
        T[0] = build(0, m - 1);
        for(int i = 1; i <= n; i++)
            T[i] = Insert(T[i - 1], hash(a[i]), 1);
        for(int i = 1; i <= n; i++)
            S[i] = T[0];
        for(int i = 0; i < q; i++) {
            if(query[i].kind == 0)
                printf("%d\n", t[Query(query[i].l, query[i].r,
                 ↪  query[i].k)]);
            else {
                Modify(query[i].l, hash(a[query[i].l]), -1);
                Modify(query[i].l, hash(query[i].r), 1);
                a[query[i].l] = query[i].r;
            }
        }
    }
    return 0;
}
```

2.4 树状数组

```
/* 一维树状数组
 * add 单点修改, ask 区间前缀和查询
 * 如果改成单点查询区间修改 add(x,z), add(y+1,-z) 然后 ask(t)
 */
#define mx 1000010
ll c[mx];
int n, q;
inline void add(int x, int d) {
    for (int i = x; i < mx; i += i & (-i))c[i] += d;
}
inline ll ask(int x) {
    ll sum = 0;
    for (int i = x; i; i -= i & (-i))sum += c[i];
    return sum;
}
// 区间修改, 区间查询
#define mx 1000010
ll c1[mx], c2[mx];
```

```
void modify(int x, int y) {
    for(int i = x; i < mx; i += i & (-i))c1[i] += y, c2[i] += (ll)x * y;
}
// 差分数组中位置 x 加上 y
void add(int l, int r, int x) {
    modify(l, x);
    modify(r + 1, -x);
}
ll ask(int x) {
    ll ans(0);
    for(int i = x; i; i -= i & (-i)) ans += (ll)(x + 1) * c1[i] - c2[i];
    return ans;
}
// 二维树状数组，如果多维度，注意类比 n 维立方体进行容斥
#define y1 y1234
#define mx 2060
ll c1[mx][mx], c2[mx][mx], c3[mx][mx], c4[mx][mx];
void modify(int x, int y, ll z) { // 差分数组 (x,y) 位置加上 z
    for(int i = x; i < mx; i += i & -i) {
        for (int j = y; j < mx; j += j & -j) {
            c1[i][j] += z;
            c2[i][j] += z * x;
            c3[i][j] += z * y;
            c4[i][j] += z * x * y;
        }
    }
}
void add(int x1, int y1, int x2, int y2, ll x) {
    x2++;
    y2++;
    modify(x2, y2, x);
    modify(x1, y2, -x);
    modify(x2, y1, -x);
    modify(x1, y1, x);
}
ll query(int x, int y) {
    ll ans = 0;
    for (int i = x; i; i -= i & (-i)) {
        for (int j = y; j; j -= j & (-j)) {
            ans += c1[i][j] * (x + 1) * (y + 1) - c2[i][j] * (y + 1) -
             ↪ c3[i][j] * (x + 1) + c4[i][j];
        }
    }
    return ans;
}
ll ask(int x1, int y1, int x2, int y2) {
```

```
    x1--;
    y1--;
    return query(x2, y2) - query(x1, y2) - query(x2, y1) + query(x1, y1);
}
#undef y1
```

2.5 ST 表

```
#include <bits/stdc++.h>
using namespace std;
const int MX = 1000010;
int d[MX][21], inx[MX]; // 这里跳 2^21 层, 随着题目数据范围变化
int num[MX];
void init(int n) {
    inx[0] = -1;
    for (int i = 1; i <= n; ++i) { // 预处理 Log, O(1) 计算
        d[i][0] = num[i];
        inx[i] = (((i) & (i - 1)) ? 0 : 1) + inx[i - 1];
    }
    for (int j = 1; j < 21; ++j) {
        int k = 1 << (j - 1);
        for (int i = 1; i + k * 2 - 1 <= n; ++i) d[i][j] = min(d[i][j -
        ↪ 1], d[i + k][j - 1]);
    }
}
int query(int l, int r) {
    int k = inx[r - l + 1];
    return min(d[l][k], d[r - (1 << k) + 1][k]);
}
```

2.6 平衡树

2.6.1 Treap

- C++ Version

```
/* 不要在函数里或者 main 里开 Treap<int>T, 开不下 放全局的
 * T.clear() 清空
 * T.insert(x) 插入
 * T.del(x) 删除, 没有成功删除返回 false
 * T.pred(x) 返回第一个小于 x 的数, T.pred(x+1) 返回第一个小等于 x 的
 ↪ 数
 * T.succ(x) 返回第一个大于 x 的数, T.succ(x-1) 返回第一个大等于 x 的
 ↪ 数
 * T.ask_kth(x) 返回第 x 小的数
 * T.ask_rank(x) 返回 x 的排名是第几小
 */
```

```
template<class Tdata>
class Treap {
public:
#define Treap_size 200010 /* 记得改这个东西 */
#define NotFound (Tdata)-1
    struct treap_point {
        int l, r, rep, son, rnd;
        Tdata x;
    } t[Treap_size];
    int treesz, root;
    void update(int k) {
        t[k].son = t[t[k].l].son + t[t[k].r].son + t[k].rep;
    }
    void right_rotate(int &k) {
        int tt = t[k].l;
        t[k].l = t[tt].r;
        t[tt].r = k;
        update(k);
        update(tt);
        k = tt;
    }
    void left_rotate(int &k) {
        int tt = t[k].r;
        t[k].r = t[tt].l;
        t[tt].l = k;
        update(k);
        update(tt);
        k = tt;
    }
    void insert_to_tree(int &k, Tdata x) {
        if (!k) {
            k = ++treesz;
            t[k].x = x;
            t[k].rnd = rand();
            t[k].son = t[k].rep = 1;
            t[k].l = t[k].r = 0;
            return;
        }
        t[k].son++;
        if (x == t[k].x) {
            t[k].rep++;
            return;
        }
        if (x < t[k].x) {
            insert_to_tree(t[k].l, x);
            if (t[t[k].l].rnd < t[k].rnd)right_rotate(k);
```

```
        }
        if (x > t[k].x) {
            insert_to_tree(t[k].r, x);
            if (t[t[k].r].rnd < t[k].rnd)left_rotate(k);
        }
    }
    bool delete_from_tree(int &k, Tdata x) {
        if (!k)return false;
        if (x == t[k].x) {
            if (t[k].rep > 1) {
                t[k].rep--;
                t[k].son--;
                return true;
            }
            if (!t[k].l || !t[k].r) {
                k = t[k].l + t[k].r;
                return true;
            }
            if (t[t[k].l].rnd < t[t[k].r].rnd) {
                right_rotate(k);
                return delete_from_tree(k, x);
            }
            left_rotate(k);
            return delete_from_tree(k, x);
        }
        bool res;
        if (x < t[k].x) {
            res = delete_from_tree(t[k].l, x);
            if (res)t[k].son--;
            return res;
        }
        res = delete_from_tree(t[k].r, x);
        if (res)t[k].son--;
        return res;
    }
    Tdata get_succ_in_tree(int k, Tdata x) {
        if (!k) return NotFound;
        Tdata sv;
        if (x < t[k].x) {
            sv = get_succ_in_tree(t[k].l, x);
            return sv == NotFound ? t[k].x : sv;
        }
        return get_succ_in_tree(t[k].r, x);
    }
    Tdata get_pred_in_tree(int k, Tdata x) {
        if(!k) return NotFound;
```

```
            Tdata sv;
            if (x > t[k].x) {
                sv = get_pred_in_tree(t[k].r, x);
                return sv == NotFound ? t[k].x : sv;
            }
            return get_pred_in_tree(t[k].l, x);
        }
        Tdata ask_kth_in_tree(int k, int x) {
            if (!k) return NotFound;
            if (x <= t[t[k].l].son) return ask_kth_in_tree(t[k].l, x);
            if (x > t[t[k].l].son + t[k].rep) return
             ↪  ask_kth_in_tree(t[k].r, x - t[t[k].l].son - t[k].rep);
            return t[k].x;
        }
        int ask_rank_in_tree(int k, Tdata x) {
            if (!k) return -1;
            if (x == t[k].x) return t[t[k].l].son + 1;
            if (x < t[k].x) return ask_rank_in_tree(t[k].l, x);
            return t[t[k].l].son + t[k].rep + ask_rank_in_tree(t[k].r,
             ↪  x);
        }
        Treap() {
            treesz = root = 0;
        }
        void clear() {
            treesz = root = 0;
        }
        void insert(Tdata x) {
            insert_to_tree(root, x);
        }
        bool del(Tdata x) {
            return delete_from_tree(root, x);
        }
        Tdata pred(Tdata x) {
            return get_pred_in_tree(root, x);
        }
        Tdata succ(Tdata x) {
            return get_succ_in_tree(root, x);
        }
        Tdata ask_kth(int x) {
            return ask_kth_in_tree(root, x);
        }
        int ask_rank(Tdata x) {
            return ask_rank_in_tree(root, x);
        }
    #undef NotFound
```

```
#undef Treap_size
};
Treap<int>T;
```

- Java Version

```
class int2 {
    int x;
    int2(int _x) {
        x = _x;
    }
    public void set(int _x) {
        x = _x;
    }
}
class TreapNode {
    int x = 0, son = 0, rep = 0, rnd = 0;
    int2 l = new int2(0), r = new int2(0);
    TreapNode(int _x) {
        l.set(0);
        r.set(0);
        son = rep = 1;
        rnd = new Random().nextInt(2147483647);
        x = _x;
    }
}
class Treap {
    int treesize = 0;
    int2 root = new int2(0);
    TreapNode[] t = new TreapNode[2000001];
    Treap() {
        t[0] = new TreapNode(0);
        t[0].son = t[0].rep = t[0].rnd = 0;
    }
    public void insert(int x) {
        Insert(root, x);
    }
    public int delete(int x) {
        return Delete(root, x);
    }
    public int get_succ(int x) {
        return Get_Succ(root, x);
    }
    public int get_pred(int x) {
        return Get_Pred(root, x);
    }
    public int ask_kth(int x) {
```

```
            return Ask_Kth(root, x);
        }
        public int ask_rank(int x) {
            return Ask_Rank(root, x);
        }
        private void update(int k) {
            t[k].son = t[t[k].l.x].son + t[t[k].r.x].son + t[k].rep;
        }
        private void right_rotate(int2 k) {
            int tt = t[k.x].l.x;
            t[k.x].l.set(t[tt].r.x);
            t[tt].r.set(k.x);
            update(k.x);
            update(tt);
            k.set(tt);
        }
        private void left_rotate(int2 k) {
            int tt = t[k.x].r.x;
            t[k.x].r.set(t[tt].l.x);
            t[tt].l.set(k.x);
            update(k.x);
            update(tt);
            k.set(tt);
        }
        private void Insert(int2 k, int x) {
            if (k.x == 0) {
                k.set(++treesize);
                t[treesize] = new TreapNode(x);
                return;
            }
            t[k.x].son++;
            if (x == t[k.x].x) {
                t[k.x].rep++;
                return;
            }
            if (x < t[k.x].x) {
                Insert(t[k.x].l, x);
                if (t[t[k.x].l.x].rnd < t[k.x].rnd)right_rotate(k);
            } else if (x > t[k.x].x) {
                Insert(t[k.x].r, x);
                if (t[t[k.x].r.x].rnd < t[k.x].rnd)left_rotate(k);
            }
        }
        private int Delete(int2 k, int x) {
            if (k.x == 0)return 0;
            if (x == t[k.x].x) {
```

```
            if (t[k.x].rep > 1) {
                t[k.x].rep--;
                t[k.x].son--;
                return 1;
            }
            if (t[k.x].l.x == 0 || t[k.x].r.x == 0) {
                k.set(t[k.x].l.x + t[k.x].r.x);
                return 1;
            }
            if (t[t[k.x].l.x].rnd < t[t[k.x].r.x].rnd) {
                right_rotate(k);
                return Delete(k, x);
            }
            left_rotate(k);
            return Delete(k, x);
        }
        int res;
        if (x < t[k.x].x) {
            res = Delete(t[k.x].l, x);
            if (res != 0)t[k.x].son--;
            return res;
        }
        res = Delete(t[k.x].r, x);
        if (res != 0)t[k.x].son--;
        return res;
    }
    private int Get_Succ(int2 k, int x) {
        if (k.x == 0)return -1;
        int sv;
        if (x < t[k.x].x) {
            sv = Get_Succ(t[k.x].l, x);
            return sv == -1 ? t[k.x].x : sv;
        }
        return Get_Succ(t[k.x].r, x);
    }
    private int Get_Pred(int2 k, int x) {
        if (k.x == 0)return -1;
        int sv;
        if (x > t[k.x].x) {
            sv = Get_Pred(t[k.x].r, x);
            return sv == -1 ? t[k.x].x : sv;
        }
        return Get_Pred(t[k.x].l, x);
    }
    private int Ask_Kth(int2 k, int x) {
        if (k.x == 0)return -1;
```

```
            if (x <= t[t[k.x].l.x].son)return Ask_Kth(t[k.x].l, x);
            if (x > t[t[k.x].l.x].son + t[k.x].rep)
                return Ask_Kth(t[k.x].r, x - t[t[k.x].l.x].son - t[k.x].rep);
            return t[k.x].x;
        }
        private int Ask_Rank(int2 k, int x) {
            if (k.x == 0)return -1;
            if (x == t[k.x].x)return t[t[k.x].l.x].son + 1;
            if (x < t[k.x].x)return Ask_Rank(t[k.x].l, x);
            return t[t[k.x].l.x].son + t[k.x].rep + Ask_Rank(t[k.x].r, x);
        }
    }
```

2.6.2 Splay

- 结构体存储

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 10;
/* Gym-100796-J */
class Splay {
public:
        int rt, idx;
        int d[maxn];
        struct Tree { int l, r, fa, rev, sz; } t[maxn];

        Splay() { rt = 0, idx = 0; }

        void push_down(int x) {
                if (!x) return;
                int ls = t[x].l, rs = t[x].r;
                if (t[x].rev) {
                        if (ls) t[ls].rev ^= 1;
                        if (rs) t[rs].rev ^= 1;
                        swap(t[x].l, t[x].r);
                        t[x].rev = 0;
                }
        }

        void maintain(int x) {
                if (!x) return;
                int ls = t[x].l, rs = t[x].r;
                t[x].sz = t[ls].sz + t[rs].sz + 1;
        }
```

```
	void rotate(int x) {
		int y = t[x].fa, z = t[y].fa;
		int ls = t[x].l, rs = t[x].r;
		if (t[z].l == y) t[z].l = x;
		else t[z].r = x;
		t[x].fa = z;
		if (t[y].l == x) {
			t[y].l = t[x].r, t[rs].fa = y;
			t[x].r = y, t[y].fa = x;
		} else {
			t[y].r = t[x].l, t[ls].fa = y;
			t[x].l = y, t[y].fa = x;
		}
		if (rt == y) rt = x;
		maintain(y); maintain(x);
	}

	void splay(int x, int ed) {
		if (x == ed) return;
		d[0] = 0;
		while (x != ed) {
			d[++d[0]] = x;
			x = t[x].fa;
		}
		for (int i = d[0]; i > 0; i--) push_down(d[i]);
		x = d[1];
		while (t[x].fa != ed && t[t[x].fa].fa != ed) {
			int y = t[x].fa, z = t[y].fa;
			if (!((t[y].l == x) ^ (t[z].l == y))) rotate(y);
			else rotate(x);
			rotate(x);
		}
		if (t[x].fa != ed) rotate(x);
	}

	void insert_front(int x) {
		t[x].sz = 1;
		if (!rt) { rt = x; return; }
		int p = rt, last = p;
		for (push_down(p); p; push_down(p)) {
			t[p].sz++;
			last = p;
			p = t[p].l;
		}
```

```
        t[x].fa = last, t[last].l = x;
        splay(x, 0);
    }

    void insert_back(int x) {
        t[x].sz = 1;
        if (!rt) { rt = x; return; }
        int p = rt, last = p;
        for (push_down(p); p; push_down(p)) {
            t[p].sz++;
            last = p;
            p = t[p].r;
        }
        t[x].fa = last, t[last].r = x;
        splay(x, 0);
    }

    void push_front() {
        ++idx;
        t[idx].l = t[idx].r = t[idx].fa = t[idx].rev = t[idx].sz = 0;
        insert_front(idx);
    }

    void push_back() {
        ++idx;
        t[idx].l = t[idx].r = t[idx].fa = t[idx].rev = t[idx].sz = 0;
        insert_back(idx);
    }

    void erase(int x) {
        splay(x, 0);
        int ls = t[x].l, rs = t[x].r;
        if (t[ls].sz <= t[rs].sz) printf("%d\n", t[ls].sz);
        else printf("%d\n", t[rs].sz);
        swap(t[x].l, t[x].r);
        int pre = t[x].l, suf = t[x].r;
        while (pre && t[pre].r) {
            push_down(pre);
            pre = t[pre].r;
        }
        while (suf && t[suf].l) {
            push_down(suf);
            suf = t[suf].l;
        }
```

```
                if (pre) {
                        splay(pre, 0);
                        if (suf) {
                                splay(suf, pre);
                                t[suf].l = 0;
                                maintain(suf), maintain(pre);
                        } else {
                                t[pre].r = 0;
                                maintain(pre);
                        }
                } else {
                        if (!suf) rt = 0;
                        else {
                                splay(suf, 0);
                                t[suf].l = 0;
                                maintain(suf);
                        }
                }
        }
} S;
inline void solve() {
        int n;
        scanf("%d", &n);
        while (n--) {
                char op[3];
                scanf("%s", op);
                if (op[0] == 'F') S.push_front();
                else if (op[0] == 'B') S.push_back();
                else {
                        int x;
                        scanf("%d", &x);
                        S.erase(x);
                }
        }
}
int main() {
        solve();
        return 0;
}
```

- 数组 ch[rt][0/1] 存储

```
/* 例题：bzoj1251 序列终结者, 操作 1: 区间加 操作 2: 区间翻转 操作 3:
↪  区间最大值
 */
#include <bits/stdc++.h>
#define inf 0x7fffffff
```

```
using namespace std;
const int maxn = 100010;
int n, m, sz, rt;
int fa[maxn], c[maxn][2], id[maxn];
int tag[maxn], v[maxn], mx[maxn], size[maxn];
bool rev[maxn];
inline void pushup(int k) {
    int l = c[k][0], r = c[k][1];
    mx[k] = max(mx[l], mx[r]);
    mx[k] = max(mx[k], v[k]);
    size[k] = size[l] + size[r] + 1;
}
void pushdown(int k) {
    int l = c[k][0], r = c[k][1], t = tag[k];
    if(t) {
        tag[k] = 0;
        if(l) {
            tag[l] += t;
            mx[l] += t;
            v[l] += t;
        }
        if(r) {
            tag[r] += t;
            mx[r] += t;
            v[r] += t;
        }
    }
    if(rev[k]) {
        rev[k] = 0;
        rev[l] ^= 1;
        rev[r] ^= 1;
        swap(c[k][0], c[k][1]);
    }
}
void rotate(int x, int &k) {
    int y = fa[x], z = fa[y], l, r;
    if(c[y][0] == x)l = 0;
    else l = 1;
    r = l ^ 1;
    if(y == k)k = x;
    else {
        if(c[z][0] == y)c[z][0] = x;
        else c[z][1] = x;
    }
    fa[x] = z;
    fa[y] = x;
```

```
    fa[c[x][r]] = y;
    c[y][l] = c[x][r];
    c[x][r] = y;
    pushup(y);
    pushup(x);
}
void splay(int x, int &k) {
    while(x != k) {
        int y = fa[x], z = fa[y];
        if(y != k) {
            if(c[y][0] == x ^ c[z][0] == y)
                rotate(x, k);
            else rotate(y, k);
        }
        rotate(x, k);
    }
}
int find(int k, int rank) {
    if(tag[k] || rev[k])pushdown(k);
    int l = c[k][0], r = c[k][1];
    if(size[l] + 1 == rank)return k;
    else if(size[l] >= rank)return find(l, rank);
    else return find(r, rank - size[l] - 1);
}
inline void update(int l, int r, int val) {
    int x = find(rt, l), y = find(rt, r + 2);
    splay(x, rt);
    splay(y, c[x][1]);
    int z = c[y][0];
    tag[z] += val;
    v[z] += val;
    mx[z] += val;
}
inline void rever(int l, int r) {
    int x = find(rt, l), y = find(rt, r + 2);
    splay(x, rt);
    splay(y, c[x][1]);
    int z = c[y][0];
    rev[z] ^= 1;
}
inline void query(int l, int r) {
    int x = find(rt, l), y = find(rt, r + 2);
    splay(x, rt);
    splay(y, c[x][1]);
    int z = c[y][0];
    printf("%d\n", mx[z]);
```

```
}
inline void build(int l, int r, int f) {
    if(l > r)return;
    int now = id[l], last = id[f];
    if(l == r) {
        fa[now] = last;
        size[now] = 1;
        if(l < f)c[last][0] = now;
        else c[last][1] = now;
        return;
    }
    int mid = (l + r) >> 1;
    now = id[mid];
    build(l, mid - 1, mid);
    build(mid + 1, r, mid);
    fa[now] = last;
    pushup(now);
    if(mid < f)c[last][0] = now;
    else c[last][1] = now;
}
int main() {
    mx[0] = -inf;
    n = read();
    m = read();
    for(int i = 1; i <= n + 2; i++)
        id[i] = ++sz;
    build(1, n + 2, 0);
    rt = (n + 3) >> 1;
    for(int i = 1; i <= m; i++) {
        int f, l, r, d;
        scanf("%d", &f);
        switch(f) {
        case 1:
            scanf("%d%d%d", &l, &r, &d);
            update(l, r, d);
            break;
        case 2:
            scanf("%d%d", &l, &r);
            rever(l, r);
            break;
        case 3:
            scanf("%d%d", &l, &r);
            query(l, r);
            break;
        }
    }
```

```
    return 0;
}
```

2.7 Link-Cut-Tree

```
/* bzoj1180:
 * 'b': 询问 (x,y) 是否联通，如果不是，那么加一条边 (x,y)
 * 'p': 修改一个点的点权
 * 'e': 如果 (x,y) 联通，那么输出路径上权值和
 * 对于带边权的问题，给每一个边建一个点 P(id=[n+1,n+m])，然后连接
 ↪  (P,x)(P,y)，在 P 上面设置点权即可
 */
#include<bits/stdc++.h>
#define inf 1000000000
#define mod 1000000007
#define ll long long
using namespace std;
int n, q, top;
int c[30005][2], fa[30005], sum[30005], v[30005], st[30005];
bool rev[30005];
char ch[15];
void update(int x) {
    int l = c[x][0], r = c[x][1];
    sum[x] = sum[l] + sum[r] + v[x];
}
void pushdown(int x) {
    int l = c[x][0], r = c[x][1];
    if(rev[x]) {
        rev[x] ^= 1, rev[l] ^= 1, rev[r] ^= 1;
        swap(c[x][0], c[x][1]);
    }
}
bool isroot(int x) {
    return !(c[fa[x]][0] == x || c[fa[x]][1] == x);
}
void rotate(int &x) {
    int y = fa[x], z = fa[y], l, r;
    l = (c[y][1] == x);
    r = l ^ 1;
    if(!isroot(y))c[z][c[z][1] == y] = x;
    fa[x] = z;
    fa[y] = x;
    fa[c[x][r]] = y;
    c[y][l] = c[x][r];
    c[x][r] = y;
    update(y);
```

```
    update(x);
}
void splay(int x) {
    top = 0;
    st[++top] = x;
    for(int i = x; !isroot(i); i = fa[i])st[++top] = fa[i];
    while(top)pushdown(st[top--]);
    while(!isroot(x)) {
        int y = fa[x], z = fa[y];
        if(!isroot(y)) {
            if(c[y][0] == x ^ c[z][0] == y)rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
}
void access(int x) {
    for(int t = 0; x; t = x, x = fa[x])
        splay(x), c[x][1] = t, update(x);
}
void makeroot(int x) {
    access(x);
    splay(x);
    rev[x] ^= 1;
}
void link(int x, int y) {
    makeroot(x);
    fa[x] = y;
}
int getrt(int x) {
    access(x);
    splay(x);
    while(c[x][0])x = c[x][0];
    return x;
}
int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)scanf("%d%d", sum + i, v + i);
    scanf("%d", &q);
    while(q--) {
        int x, y;
        scanf("%s%d%d", ch + 1, &x, &y);
        if(ch[1] == 'b') {
            if(getrt(x) == getrt(y))puts("no");
            else puts("yes"), link(x, y);
        }
```

```
        if(ch[1] == 'p') {
            makeroot(x);
            v[x] = y;
            update(x);
        }
        if(ch[1] == 'e') {
            if(getrt(x) != getrt(y))puts("impossible");
            else {
                makeroot(x), access(y), splay(y);
                printf("%d\n", sum[y]);
            }
        }
    }
    return 0;
}
```

2.8 树链剖分

```
// 点权式 - 配合线段树的单点修改 + 区间查询
struct Edge {
        int to, next;
        Edge() {}
        Edge(int a, int b) { to = a; next = b; }
} E[maxn << 1];
int head[maxn], cnt, tot;
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
int a[maxn];
void init() {
        memset(head, -1, sizeof head);
        tot = cnt = 0;
}
void addedge(int u, int v) {
        E[cnt].to = v;
        E[cnt].next = head[u];
        head[u] = cnt++;
}
void dfs1(int u, int fa, int d) { // 处理出重链
        size[u] = 1; deep[u] = d; son[u] = 0;
        for (int i = head[u]; ~i; i = E[i].next) {
                int v = E[i].to;
                if (v != fa) {
                        dfs1(v, u, d + 1);
                        pa[v] = u;
                        size[u] += size[v];
                        if (size[v] > size[son[u]]) son[u] = v;
                }
```

```
        }
}
void dfs2(int u, int first) {
        top[u] = first;
        id[u] = ++tot;
        if (son[u]) dfs2(son[u], first);
        for (int i = head[u]; ~i; i = E[i].next) {
                int v = E[i].to;
                if (v != pa[u] && v != son[u]) dfs2(v, v);
        }
}
void solve(int u, int v) { // 跑 LCA 同时计算答案
        int x = top[u], y = top[v], res = 0;
        while (x != y) {
                if (deep[x] < deep[y]) {
                        swap(u, v);
                        swap(x, y);
                }
                res += T.query(id[x], id[u], 1);
                u = pa[x];
                x = top[u];
        }
        if (deep[u] > deep[v]) swap(u, v);
        res += T.query(id[u], id[v], 1);
        printf("%d\n", res);
}
// 边权式：把边权偏移到深度较大的点，化成点权式
struct Point {
        int from, to, val;
} p[maxn];
struct Edge {
        int to, next;
} E[maxn << 1];
int head[maxn], cnt, tot;
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
int a[maxn];
void init() {
        memset(head, -1, sizeof head);
        tot = cnt = 0;
}
void addedge(int u, int v) {
        E[cnt].to = v;
        E[cnt].next = head[u];
        head[u] = cnt++;
}
void dfs1(int u, int fa, int d) {
```

```
        size[u] = 1; deep[u] = d; son[u] = 0;
        for (int i = head[u]; ~i; i = E[i].next) {
                int v = E[i].to;
                if (v != fa) {
                        dfs1(v, u, d + 1);
                        pa[v] = u;
                        size[u] += size[v];
                        if (size[v] > size[son[u]]) son[u] = v;
                }
        }
}
void dfs2(int u, int first) {
        top[u] = first;
        id[u] = ++tot;
        if (son[u]) dfs2(son[u], first);
        for (int i = head[u]; ~i; i = E[i].next) {
                int v = E[i].to;
                if (v != pa[u] && v != son[u]) dfs2(v, v);
        }
}
int solve(int u, int v) {
        int x = top[u], y = top[v];
        int ans = 0;
        while (x != y) {
                if (deep[x] < deep[y]) {
                        swap(u, v);
                        swap(x, y);
                }
                ans += T.query(id[x], id[u], 1);
                u = pa[x];
                x = top[u];
        }
//              printf("%d : %d - %d : ", ans, u, v);
        if (deep[u] > deep[v])
                swap(u, v);
        if (u != v) ans += T.query(id[son[u]], id[v], 1);
        return ans;
}
/* 例题：POJ 2763 - 给一颗树，边之间有权值，两种操作
 *          第一种：求任意两点的权值和，
 *          第二种：修改树上两点的权值。
 */
#include <bits/stdc++.h>
using namespace std;
#define maxn 100010
/* head */
```

```
class SegmentTree {
public:
#define lson root << 1
#define rson root << 1 | 1
#define lent (t[root].r - t[root].l + 1)
#define lenl (t[lson].r - t[lson].l + 1)
#define lenr (t[rson].r - t[rson].l + 1)
    struct Tree {
        int l, r, val;
    } t[maxn << 4];

    void pushup(int root) {
        t[root].val = t[lson].val + t[rson].val;
    }

    void build(int l, int r, int root) {
        t[root].l = l;
        t[root].r = r;
        if (l == r) {
            t[root].val = 0;
            return;
        }
        int mid = l + r >> 1;
        build(l, mid, lson);
        build(mid + 1, r, rson);
        pushup(root);
    }

    void update(int l, int r, int val, int root) {
        if (l <= t[root].l && t[root].r <= r) {
            t[root].val = val;
            return;
        }
        int mid = t[root].l + t[root].r >> 1;
        if (l <= mid) update(l, r, val, lson);
        if (r > mid) update(l, r, val, rson);
        pushup(root);
    }

    int query(int l, int r, int root) {
        if (l <= t[root].l && t[root].r <= r) return t[root].val;
        int mid = t[root].l + t[root].r >> 1;
        int ans = 0;
        if (l <= mid) ans += query(l, r, lson);
        if (r > mid) ans += query(l, r, rson);
        return ans;
```

```
    }
#undef lenr
#undef lenl
#undef lent
#undef rson
#undef lson
} T;
struct Point {
    int from, to, val;
} p[maxn];
struct Edge {
    int to, next;
} E[maxn << 1];
int head[maxn], cnt, tot;
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
int a[maxn];
void init() {
    memset(head, -1, sizeof head);
    tot = cnt = 0;
}
void addedge(int u, int v) {
    E[cnt].to = v;
    E[cnt].next = head[u];
    head[u] = cnt++;
}
void dfs1(int u, int fa, int d) {
    size[u] = 1; deep[u] = d; son[u] = 0;
    for (int i = head[u]; ~i; i = E[i].next) {
        int v = E[i].to;
        if (v != fa) {
            dfs1(v, u, d + 1);
            pa[v] = u;
            size[u] += size[v];
            if (size[v] > size[son[u]]) son[u] = v;
        }
    }
}
void dfs2(int u, int first) {
    top[u] = first;
    id[u] = ++tot;
    if (son[u]) dfs2(son[u], first);
    for (int i = head[u]; ~i; i = E[i].next) {
        int v = E[i].to;
        if (v != pa[u] && v != son[u]) dfs2(v, v);
    }
}
```

```cpp
int solve(int u, int v) {
    int x = top[u], y = top[v];
    int ans = 0;
    while (x != y) {
        if (deep[x] < deep[y]) {
            swap(u, v);
            swap(x, y);
        }
        ans += T.query(id[x], id[u], 1);
        u = pa[x];
        x = top[u];
    }
    if (deep[u] > deep[v]) swap(u, v);
    if (u != v) ans += T.query(id[son[u]], id[v], 1);
    return ans;
}
int main() {
    int n, m, q;
    while (~scanf("%d%d%d", &n, &q, &m)) {
        init();
        for (int i = 1; i < n; i++) {
            scanf("%d%d%d", &p[i].from, &p[i].to, &p[i].val);
            addedge(p[i].from, p[i].to);
            addedge(p[i].to, p[i].from);
        }
        dfs1(1, 0, 1);
        dfs2(1, 1);
        T.build(1, tot, 1);
        for (int i = 1; i < n; i++) {
            if (deep[p[i].from] < deep[p[i].to]) swap(p[i].from, p[i].to);
            T.update(id[p[i].from], id[p[i].from], p[i].val, 1);
        }
        while (q--) {
            int op, x, y;
            scanf("%d%d", &op, &x);
            if (op) {
                scanf("%d", &y);
                T.update(id[p[x].from], id[p[x].from], y, 1);
            } else {
                printf("%d\n", solve(m, x));
                m = x;
            }
        }
    }
    return 0;
```

```
}
```

2.9 KD 树

2.9.1 一维 KD-Tree

```
#include <bits/stdc++.h>
using namespace std;
#define x first
#define y second
const int MAXN = 2e5 + 5;
typedef long long ll;
const ll inf = 4e18;
namespace KD_Tree {
    struct node {
        node *ch[2];
        int d[2], mx[2], my[2], size;
        // d 表示这个点的坐标 [0]->x,[1]->y,mx 表示这个平面的 x 的范围, my 表示这个平面的 y 的范围
        inline void push_up() {
            size = 1;
            for(int i = 0; i <= 1; i++) {
                if(ch[i]) {
                    mx[0] = min(mx[0], ch[i]->mx[0]);
                    mx[1] = max(mx[1], ch[i]->mx[1]);
                    my[0] = min(my[0], ch[i]->my[0]);
                    my[1] = max(my[1], ch[i]->my[1]);
                    size += ch[i]->size;
                }
            }
        }
    } pool_node[MAXN], *pool_top = pool_node;
    node *del_pool[MAXN], * *del_top = del_pool;
    inline node *newnode() {
        return del_top == del_pool ? ++pool_top : *(del_top--);
    }
    typedef pair<int, int> Point;
    inline bool cmp_x(const Point &a, const Point &b) {
        return a.x < b.x;
    }
    inline bool cmp_y(const Point &a, const Point &b) {
        return a.y < b.y;
    }
    node **rebuild_need;
    int rebuild_d;
    Point stk[MAXN];
    int point_cnt;
```

```
    //用于最开始建树，可以不用直接插点，方法为 build(1,point_cnt,0)，保证
    ↪  点在 stk 数组中
    node *build(int l, int r, bool f) {
        int mid = (l + r) >> 1;
        node *o = newnode();
        nth_element(stk + l, stk + mid, stk + r + 1, !f ? cmp_x : cmp_y);
        o->d[0] = o->mx[0] = o->mx[1] = stk[mid].x;
        o->d[1] = o->my[0] = o->my[1] = stk[mid].y;
        o->ch[0] = l < mid ? build(l, mid - 1, f ^ 1) : 0;
        o->ch[1] = mid < r ? build(mid + 1, r, f ^ 1) : 0;
        o->push_up();
        return o;
    }
    void remove(node *o) {
        if(o->ch[0])
            remove(o->ch[0]);
        if(o->ch[1])
            remove(o->ch[1]);
        stk[++point_cnt] = Point(o->d[0], o->d[1]);
        *(++del_top) = o;
    }
    /*
    插入以及重构代码如下
    insert(root,x,y,0);
    if(rebuild_need)
        rebuild();
    */
    void rebuild() {
        point_cnt = 0;
        remove(*rebuild_need);
        *rebuild_need = build(1, point_cnt, rebuild_d);
        rebuild_need = 0;
    }
    void insert(node *&o, int x, int y, bool f) {
        if(!o) {
            o = newnode();
            o->d[0] = o->mx[0] = o->mx[1] = x;
            o->d[1] = o->my[0] = o->my[1] = y;
        } else if(o->d[0] != x || o->d[1] != y) {
            int d = !f ? o->d[0] < x : o->d[1] < y;
            insert(o->ch[d], x, y, f ^ 1);
            o->push_up();
            if(o->ch[d]->size * 10 >= o->size * 7)
                rebuild_need = &o, rebuild_d = f;
        }
    }
```

```
    Point P;
    ll ans;
    int calc_mx(node *o) { //同下 calc_mn
        int ret = 0;
        ret += max(abs(o->mx[0] - P.x), abs(o->mx[1] - P.x));
        ret += max(abs(o->my[0] - P.y), abs(o->my[1] - P.y));
        return ret;
    }
    ll calc_mn(node *o) { //Manhattan 距离下进行计算，如果是欧氏距离，则为 max(P.x-o->mx[1],0)+max(o->mx[0]-P.x,0) 的平方（本处表示到这个子树平面的最短距离）
        ll ret = 0;
        ret += max(abs(o->mx[0] - P.x), abs(o->mx[1] - P.x));
        ret += max(abs(o->my[0] - P.y), abs(o->my[1] - P.y));
        // ret+=1ll*(max(P.x-o->mx[1],0ll)+max(o->mx[0]-P.x,0ll))*(max(P.x-o->mx[1]
        // ret+=1ll*(max(P.y-o->my[1],0ll)+max(o->my[0]-P.y,0ll))*(max(P.y-o->my[1]
        return ret;
    }
    void Query_Max(node *o) { //查询平面中到目标点最远的点
        ans = max(ans, 1ll * (o->d[0] - P.x) * (o->d[0] - P.x) + 1ll * (o->d[1] - P.y) * (o->d[1] - P.y)); //欧几里得距离
        // ans=max(ans,abs(o->d[0]-P.x)+abs(o->d[1]-P.y)); //曼哈顿距离
        int dl = o->ch[0] ? calc_mx(o->ch[0]) : -inf;
        int dr = o->ch[1] ? calc_mx(o->ch[1]) : -inf;
        if(dl > dr) {
            if(dl > ans)
                Query_Max(o->ch[0]);
            if(dr > ans)
                Query_Max(o->ch[1]);
        } else {
            if(dr > ans)
                Query_Max(o->ch[1]);
            if(dl > ans)
                Query_Max(o->ch[0]);
        }
    }
    void Query_Min(node *o) { // 查询平面中到目标点最近的点
        if(!o)return;
        ans = min(ans, 1ll * (o->d[0] - P.x) * (o->d[0] - P.x) + 1ll * (o->d[1] - P.y) * (o->d[1] - P.y)); //欧几里得距离
        // ans=max(ans,abs(o->d[0]-P.x)+abs(o->d[1]-P.y)); //曼哈顿距离
        ll dl = o->ch[0] ? calc_mn(o->ch[0]) : inf;
        ll dr = o->ch[1] ? calc_mn(o->ch[1]) : inf;
        if(dl < dr) {
```

```
                if(dl < ans)
                    Query_Min(o->ch[0]);
                if(dr < ans)
                    Query_Min(o->ch[1]);
            } else {
                if(dr < ans)
                    Query_Min(o->ch[1]);
                if(dl < ans)
                    Query_Min(o->ch[0]);
            }
        }
        node *root;
        ll Query(int x, int y, bool f) { // 传入 1 为查询最远距离, 传入 0 为查
        ↪  询最近距离
            P = Point(x, y);
            if(f) ans = -inf, Query_Max(root);
            else ans = inf, Query_Min(root);
            return ans;
        }
        // 以下三个函数为本人手写, 所以可靠性有待验证, 不过过过题....
        void debug(node *&a) {
            if(a->ch[0]) debug(a->ch[0]);
            printf("%lld %lld\n", a->d[0], a->d[1]);
            if(a->ch[1]) debug(a->ch[1]);
        }
        void del(node *&a) { //用于清空整颗树
            for(int i = 0; i <= 1; i++) {
                if(a->ch[i]) del(a->ch[i]);
            }
            a = NULL;
        }
        void init() { //用于初始化
            del_top = del_pool, pool_top = pool_node;
        }
}
int x[100010], y[100010];
void solve() {
    int n;
    scanf("%d", &n);
    KD_Tree::init();//初始化
    for(int i = 1; i <= n; i++) {
        scanf("%d%d", &x[i], &y[i]);
        KD_Tree::insert(KD_Tree::root, x[i], y[i], 1);
        if(KD_Tree::rebuild_need)KD_Tree::rebuild();
    }
    for(int i = 1; i <= n; i++) {
```

```
        ll s = KD_Tree::Query(x[i], y[i], 0) ;
        printf("%lld\n", s);
    }
    KD_Tree::del(KD_Tree:: root);//清空
}
int main() {
    int t;
    scanf("%d", &t);
    while(t--) solve();
    return 0;
}
```

2.9.2 高维 KD-Tree

```
/* 求取一个点的 m 维度的最近的点
 */
#include <bits/stdc++.h>
using namespace std;
const int maxn = 200010;
int cmpNo, K;
struct Node {
    int x[10], l, r, id;
    bool operator< (const Node &b) const {
        return x[cmpNo] < b.x[cmpNo];
    }
};
long long Dis(const Node &a, const Node &b) {
    long long ret = 0;
    for(int i = 0; i < K; i++) ret += (a.x[i] - b.x[i]) * (a.x[i] -
    ↪  b.x[i]);
    return ret;
}
Node p[maxn];
int Build(int l, int r, int d) {
    if(l > r) return 0;
    cmpNo = d;
    int mid = l + r >> 1;
    nth_element(p + l, p + mid, p + r + 1);
    p[mid].l = Build(l, mid - 1, (d + 1) % K);
    p[mid].r = Build(mid + 1, r, (d + 1) % K);
    return mid;
}
priority_queue<pair<long long, int> >q;
void Kth(int l, int r, Node tar, int k, int d) {
    if(l > r)return;
    int mid = l + r >> 1;
    pair<long long, int>v = make_pair(Dis(p[mid], tar), p[mid].id);
```

```
    if(q.size() == k && v < q.top())q.pop();
    if(q.size() < k)q.push(v);
    long long t = tar.x[d] - p[mid].x[d];
    if(t <= 0) {
        Kth(l, mid - 1, tar, k, (d + 1) % K);
        if(q.top().first > t * t)
            Kth(mid + 1, r, tar, k, (d + 1) % K);
    } else {
        Kth(mid + 1, r, tar, k, (d + 1) % K);
        if(q.top().first > t * t)
            Kth(l, mid - 1, tar, k, (d + 1) % K);
    }
}
int k, ans[20];
Node a[maxn];
int main() {
    int n;
    while(scanf("%d%d", &n, &K) != EOF) {
        for(int id = 1; id <= n; id++) {
            for(int i = 0; i < K; i++) scanf("%d", &p[id].x[i]);
            p[id].id = id;
            a[id] = p[id];
        }
        Build(1, n, 0);
        int Q, tot;
        scanf("%d", &Q);
        Node tar;
        while(Q--) {
            for(int i = 0; i < K; i++) scanf("%d", &tar.x[i]);
            scanf("%d", &k);
            printf("the closest %d points are:\n", k);
            for(int i = 1; i <= k; i++)q.push(make_pair(1e18, -1));
            Kth(1, n, tar, k, 0);
            tot = 0;
            while(!q.empty()) {
                int id = (q.top()).second;
                q.pop();
                ans[tot++] = id;
            }
            for(int i = tot - 1; i >= 0; i--) {
                for(int j = 0; j < K; j++) printf("%d%c", a[ans[i]].x[j],
                 ↪ j == K - 1 ? '\n' : ' ');
            }
        }
    }
    return 0;
```

```
}
```

2.10 莫队算法

```
/* 例题 1：Codeforces 617E - 对每个查询区间，计算有多少个子区间异或为 k
 * 解法：预处理异或前缀， 一个区间 [L,R] 的异或值等于 sum[R]^sum[L-1]
 *      接下来发现如果已知一个区间很容易推到相邻区间信息，因此可以用莫队维护
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
const int maxk = 2e6 + 5;
struct Query {
    int l, r, id;
    ll ans;
} q[maxn];
int pos[maxn], BLOCK;
inline bool cmp1(const Query& x, const Query& y) { return pos[x.l] <
    pos[y.l]|| (pos[x.l] == pos[y.l] && x.r < y.r); }
inline bool cmp2(const Query& x, const Query& y) { return x.id < y.id; }
int pre[maxn], vis[maxk];
ll res;
int n, m, k;
void modify(int p, bool op) {
    if (op == 0) {
        vis[pre[p]]--;
        res -= vis[pre[p] ^ k];
    } else if (op == 1) {
        res += vis[pre[p] ^ k];
        vis[pre[p]]++;
    }
}
int main() {
    scanf("%d%d%d", &n, &m, &k);
    BLOCK = sqrt(n + 0.5);
    pre[0] = 0;
    for (int i = 1; i <= n; i++) {
        int a;
        scanf("%d", &a);
        pre[i] = pre[i - 1] ^ a;
        pos[i] = (i - 1) / BLOCK + 1;
    }
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].id = i;
```

```
    }
    sort(q, q + m, cmp1);
    int L = 0, R = 0; // 维护左区间
    memset(vis, 0, sizeof vis);
    res = 0;
    for (int i = 0; i < m; i++) {
        int l = q[i].l, r = q[i].r;
        while (L < l - 1) modify(L, 0), L++;
        while (L >= l) modify(L - 1, 1), L--;
        while (R <= r) modify(R, 1), R++;
        while (R > r + 1) modify(R - 1, 0), R--;
        q[i].ans = res;
    }
    sort(q, q + m, cmp2);
    for (int i = 0; i < m; i++) printf("%lld\n", q[i].ans);
    return 0;
}
/* 例题 2：Codeforces 940F - n 个整数，m 次操作
 *      一种是查询 [l,r] 的 mex{c_1,...,c_{10^9}}，c_x 指 x 出现的次数，另
 ↪ 一种是修改某个整数的值
 * 解法：把查询分块，按顺序暴力修改，即带修改的莫队
 */
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
const int maxn = 1e5 + 10;
/* head */
struct Change { // origin y, now x
    int p, x, y;
} c[maxn];
struct Query {
    int l, r, t, ans, id;
} q[maxn];
int pos[maxn];
bool cmp1(const Query& x, const Query& y) {
    if (pos[x.l] == pos[y.l]) {
        if (pos[x.r] == pos[y.r]) return x.t < y.t;
        return pos[x.r] < pos[y.r];
    }
    return pos[x.l] < pos[y.l];
}
bool cmp2(const Query& x, const Query& y) {
    return x.id < y.id;
}
map<int, int> M;
int a[maxn], b[maxn];
```

```
int num[maxn << 1], cnt[maxn];
void update(int x, int v) {
    if (num[x] > 0) cnt[num[x]]--;
    num[x] += v;
    if (num[x] > 0) cnt[num[x]]++;
}
void modify(int p, int x, int l, int r) {
    if (l <= p && p <= r) {
        update(a[p], -1);
        update(x, 1);
    }
    a[p] = x;
}
int calc() {
    int i = 1;
    while (i) {
        if (!cnt[i]) return i;
        i++;
    }
}
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    int block = pow(n, 2.0 / 3.0); // 带修改莫队一般块大小取 n^{2/3}
    int idx = 0;
    rep(i, 1, n + 1) {
        scanf("%d", &a[i]);
        if (!M.count(a[i])) M[a[i]] = ++idx;
        int tmp = M[a[i]];
        b[i] = a[i] = tmp;
        pos[i] = i / block + 1;
    }
    int q_cnt = 0, c_cnt = 0;
    rep(i, 1, m + 1) {
        int op, x, y;
        scanf("%d%d%d", &op, &x, &y);
        if (op == 1) q[++q_cnt] = { x, y, c_cnt, 0, q_cnt };
        else if (op == 2) {
            if (!M.count(y)) M[y] = ++idx;
            int tmp = M[y];
            c[++c_cnt] = { x, tmp, b[x] };
            b[x] = tmp;
        }
    }
    sort(q + 1, q + q_cnt + 1, cmp1);
    int v = 0, l = 1, r = 0;
```

```
    rep(i, 1, q_cnt + 1) {
        int T = q[i].t, L = q[i].l, R = q[i].r;
        while (v < T) modify(c[v + 1].p, c[v + 1].x, l, r), v++;
        while (v > T) modify(c[v].p, c[v].y, l, r), v--;
        while (l < L) update(a[l], -1), l++;
        while (l > L) update(a[l - 1], 1), l--;
        while (r < R) update(a[r + 1], 1), r++;
        while (r > R) update(a[r], -1), r--;
        q[i].ans = calc();
    }
    sort(q + 1, q + q_cnt + 1, cmp2);
    rep(i, 1, q_cnt + 1) printf("%d\n", q[i].ans);
    return 0;
}
```

2.11 分块算法

```
// 区间加法，单点求值：
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5e4 + 5;
int a[maxn], b[maxn], pos[maxn], L[maxn], R[maxn];
int m;
void modify(int l, int r, int c) {
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) a[i] += c;
    }
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) b[i] += c;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) a[i] += c;
    }
}
int main() {
    int n;
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        pos[i] = (i - 1) / m + 1;
        b[i] = 0;
    }
    int tot = n / m;
    if (n % m) tot++;
    for (int i = 1; i <= tot; i++) {
        L[i] = (i - 1) * m + 1;
```

```
        R[i] = i * m;
    }
    R[tot] = n;
    for (int i = 1; i <= n; i++) {
        int op, l, r, c;
        scanf("%d%d%d%d", &op, &l, &r, &c);
        if (op == 0) modify(l, r, c);
        else if (op == 1) printf("%d\n", a[r] + b[pos[r]]);
    }
    return 0;
}
// 区间加法，区间小于某个值的元素之和：
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 5e4 + 5;
int pos[maxn], L[maxn], R[maxn];
pii a[maxn];
int b[maxn];
int m;
void modify(int l, int r, int c) {
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= a[i].second && a[i].second <= r) a[i].first += c;
    }
    sort(a + L[pos[l]], a + R[pos[l]] + 1);
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) b[i] += c;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= a[i].second && a[i].second <= r) a[i].first += c;
    }
    sort(a + L[pos[r]], a + R[pos[r]] + 1);
}
int calc(int left, int right, int c) {
    int l = left, r = right, res = 0;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m].first < c) {
            res = m - left + 1;
            l = m + 1;
        } else r = m - 1;
    }
    return res;
}
int query(int l, int r, int c) {
    int cnt = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
```

```cpp
            if (l <= a[i].second && a[i].second <= r && a[i].first < c - b[pos[l]]) cnt++;
        }
        if (pos[l] == pos[r]) return cnt;
        for (int i = pos[l] + 1; i < pos[r]; i++) cnt += calc(L[i], R[i], c - b[i]);
        for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
            if (l <= a[i].second && a[i].second <= r && a[i].first < c - b[pos[r]]) cnt++;
        }
        return cnt;
}
int main() {
    int n;
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i].first);
        a[i].second = i;
        pos[i] = (i - 1) / m + 1;
        b[i] = 0;
    }
    int tot = n / m;
    if (n % m) tot++;
    for (int i = 1; i <= tot; i++) {
        L[i] = (i - 1) * m + 1;
        R[i] = i * m;
    }
    R[tot] = n;
    for (int i = 1; i <= tot; i++) sort(a + L[i], a + R[i] + 1);
    for (int i = 1; i <= n; i++) {
        int op, l, r, c;
        scanf("%d%d%d%d", &op, &l, &r, &c);
        if (op == 0) modify(l, r, c);
        else if (op == 1) printf("%d\n", query(l, r, c * c));
    }
    return 0;
}
// 区间加法，区间小于某个值 x 的前驱：
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 1e5 + 5;
int pos[maxn], L[maxn], R[maxn];
pii a[maxn];
int b[maxn];
```

```
int m;
void modify(int l, int r, int c) {
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= a[i].second && a[i].second <= r) a[i].first += c;
    }
    sort(a + L[pos[l]], a + R[pos[l]] + 1);
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) b[i] += c;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= a[i].second && a[i].second <= r) a[i].first += c;
    }
    sort(a + L[pos[r]], a + R[pos[r]] + 1);
}
int calc(int left, int right, int c, int d) {
    int l = left, r = right, res = -1 - d;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m].first < c - d) {
            res = a[m].first;
            l = m + 1;
        } else r = m - 1;
    }
    return res;
}
int query(int l, int r, int c) {
    int mx = -1;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= a[i].second && a[i].second <= r && a[i].first + b[pos[l]] < c) mx = max(mx, a[i].first + b[pos[l]]);
    }
    if (pos[l] == pos[r]) return mx;
    for (int i = pos[l] + 1; i < pos[r]; i++) mx = max(mx, calc(L[i], R[i], c, b[i]) + b[i]);
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= a[i].second && a[i].second <= r && a[i].first + b[pos[r]] < c) mx = max(mx, a[i].first + b[pos[r]]);
    }
    return mx;
}
int main() {
    int n;
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i].first);
        a[i].second = i;
```

```
        pos[i] = (i - 1) / m + 1;
        b[i] = 0;
    }
    int tot = n / m;
    if (n % m) tot++;
    for (int i = 1; i <= tot; i++) {
        L[i] = (i - 1) * m + 1;
        R[i] = i * m;
    }
    R[tot] = n;
    for (int i = 1; i <= tot; i++) sort(a + L[i], a + R[i] + 1);
    for (int i = 1; i <= n; i++) {
        int op, l, r, c;
        scanf("%d%d%d%d", &op, &l, &r, &c);
        if (op == 0) modify(l, r, c);
        else if (op == 1) printf("%d\n", query(l, r, c));
    }
    return 0;
}
// 区间加法，区间求和：
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 5e4 + 5;
int pos[maxn], L[maxn], R[maxn];
ll a[maxn], b[maxn], c[maxn];
int n, m, k;
void modify(int l, int r, int d) {
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) {
            a[i] += d;
            c[pos[l]] += d;
        }
    }
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) {
        b[i] += d;
        c[i] += 1LL * (R[i] - L[i] + 1) * d;
    }
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) {
            a[i] += d;
            c[pos[r]] += d;
        }
    }
}
```

```
int query(int l, int r, int mod) {
    int res = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) res = (res + a[i] + b[pos[l]]) % mod;
    }
    if (pos[l] == pos[r]) return res;
    for (int i = pos[l] + 1; i < pos[r]; i++) res = (res + c[i]) % mod;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) res = (res + a[i] + b[pos[r]]) % mod;
    }
    return res;
}
int main() {
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        pos[i] = (i - 1) / m + 1;
        b[i] = 0;
        c[pos[i]] += a[i];
    }
    k = n / m;
    if (n % m) k++;
    for (int i = 1; i <= k; i++) {
        L[i] = (i - 1) * m + 1;
        R[i] = i * m;
    }
    R[k] = n;
    for (int i = 1; i <= n; i++) {
        int op, l, r, c;
        scanf("%d%d%d%d", &op, &l, &r, &c);
        if (op == 0) modify(l, r, c);
        else if (op == 1) printf("%d\n", query(l, r, c + 1));
    }
    return 0;
}
// 单点插入，单点询问（带块的分裂）：
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 2e5 + 5;
int pos[maxn];
vector<int> v[maxn];
int n, m, k;
int s[maxn], top;
void rebuild() {
```

```
    top = 0;
    for (int i = 1; i <= k; i++) {
        for (int j = 0; j < v[i].size(); j++) s[++top] = v[i][j];
        v[i].clear();
    }
    m = sqrt(top + 0.5);
    for (int i = 1; i <= top; i++) {
        pos[i] = (i - 1) / m + 1;
        v[pos[i]].push_back(s[i]);
    }
    k = top / m;
    if (top % m) k++;
}
pii query(int x) {
    int id = 1;
    while (x > v[id].size()) x -= v[id].size(), id++;
    return {id, x - 1};
}
void insert(int l, int r) {
    pii res = query(l);
    v[res.first].insert(v[res.first].begin() + res.second, r);
    if (v[res.first].size() > 20 * m) rebuild();
}
int main() {
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &s[i]);
        pos[i] = (i - 1) / m + 1;
        v[pos[i]].push_back(s[i]);
    }
    k = n / m;
    if (n % m) k++;
    for (int i = 1; i <= n; i++) {
        int op, l, r, c;
        scanf("%d%d%d%d", &op, &l, &r, &c);
        if (op == 0) insert(l, r);
        else {
            pii res = query(r);
            printf("%d\n", v[res.first][res.second]);
        }
    }
    return 0;
}
// 区间乘法, 区间加法, 单点询问:
#include <bits/stdc++.h>
```

```
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
const int mod = 10007;
int pos[maxn], L[maxn], R[maxn];
ll a[maxn], b[maxn], c[maxn];
int n, m, k;
void modify1(int l, int r, int x) {
    for (int i = L[pos[l]]; i <= R[pos[l]] && c[pos[l]] != 1; i++) a[i] =
    ↪ a[i] * c[pos[l]] % mod;
    c[pos[l]] = 1;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) a[i] = (a[i] + x) % mod;
    }
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) b[i] = (b[i] + x) % mod;
    for (int i = L[pos[r]]; i <= R[pos[r]] && c[pos[r]] != 1; i++) a[i] =
    ↪ a[i] * c[pos[r]]  % mod;
    c[pos[r]] = 1;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) a[i] = (a[i] + x) % mod;
    }
}
void modify2(int l, int r, int x) {
    for (int i = L[pos[l]]; i <= R[pos[l]] && (b[pos[l]] || c[pos[l]] !=
    ↪ 1); i++) a[i] = (a[i] * c[pos[l]] % mod + b[pos[l]]) % mod;
    b[pos[l]] = 0;
    c[pos[l]] = 1;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) a[i] = a[i] * x % mod;
    }
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) {
        c[i] = c[i] * x % mod;
        b[i] = b[i] * x % mod;
    }
    for (int i = L[pos[r]]; i <= R[pos[r]] && (b[pos[r]] || c[pos[r]] !=
    ↪ 1); i++) a[i] = (a[i] * c[pos[r]] % mod + b[pos[r]]) % mod;
    b[pos[r]] = 0;
    c[pos[r]] = 1;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) a[i] = a[i] * x % mod;
    }
}
int main() {
    scanf("%d", &n);
```

```
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        pos[i] = (i - 1) / m + 1;
    }
    k = n / m;
    if (n % m) k++;
    for (int i = 1; i <= k; i++) {
        b[i] = 0, c[i] = 1;
        L[i] = (i - 1) * m + 1;
        R[i] = i * m;
    }
    R[k] = n;
    for (int i = 1; i <= n; i++) {
        int op, l, r, x;
        scanf("%d%d%d%d", &op, &l, &r, &x);
        if (op == 0) modify1(l, r, x);
        else if (op == 1) modify2(l, r, x);
        else if (op == 2) printf("%lld\n", (a[r] * c[pos[r]] % mod +
        ↪  b[pos[r]]) % mod);
    }
    return 0;
}
// 区间询问等于 c 的个数，并修改成 c：
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
const int mod = 10007;
int pos[maxn], L[maxn], R[maxn];
int a[maxn], b[maxn], tag[maxn];
int n, m, k;
inline void modify(int l, int r, int c) {
    for (int i = L[pos[l]]; i <= R[pos[l]] && tag[pos[l]]; i++) a[i] =
    ↪  b[pos[l]];
    tag[pos[l]] = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) a[i] = c;
    }
    if (pos[l] == pos[r]) return;
    for (int i = pos[l] + 1; i < pos[r]; i++) b[i] = c, tag[i] = 1;
    for (int i = L[pos[r]]; i <= R[pos[r]] && tag[pos[r]]; i++) a[i] =
    ↪  b[pos[r]];
    tag[pos[r]] = 0;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r) a[i] = c;
```

```
    }
}
inline int query(int l, int r, int c) {
    int cnt = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]] && tag[pos[l]]; i++) a[i] =
     ↪  b[pos[l]];
    tag[pos[l]] = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r && a[i] == c) cnt++;
    }
    if (pos[l] == pos[r]) return cnt;
    for (int i = pos[l] + 1; i < pos[r]; i++) {
        if (tag[i]) {
            if (b[i] == c) cnt += R[i] - L[i] + 1;
        } else {
            for (int j = L[i]; j <= R[i]; j++) {
                if (a[j] == c) cnt++;
            }
        }
    }
    for (int i = L[pos[r]]; i <= R[pos[r]] && tag[pos[r]]; i++) a[i] =
     ↪  b[pos[r]];
    tag[pos[r]] = 0;
    for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
        if (l <= i && i <= r && a[i] == c) cnt++;
    }
    return cnt;
}
int main() {
    scanf("%d", &n);
    m = sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        pos[i] = (i - 1) / m + 1;
    }
    k = n / m;
    if (n % m) k++;
    for (int i = 1; i <= k; i++) {
        tag[i] = b[i] = 0;
        L[i] = (i - 1) * m + 1;
        R[i] = i * m;
    }
    R[k] = n;
    for (int i = 1; i <= n; i++) {
        int l, r, c;
        scanf("%d%d%d", &l, &r, &c);
```

```
        printf("%d\n", query(l, r, c));
        modify(l, r, c);
    }
    return 0;
}
// 区间众数：
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 5;
const int maxm = 505;
map<int, int> M;
vector<int> v[maxn];
int a[maxn], b[maxn], c[maxn];
int pos[maxn], L[maxm], R[maxm], f[maxm][maxm];
int n, m, k, id;
void init(int j) {
    memset(c, 0, sizeof c);
    int mx = 0, ans = 0;
    for (int i = L[j]; i <= n; i++) {
        c[a[i]]++;
        if (c[a[i]] > mx || (c[a[i]] == mx && b[a[i]] < b[ans])) {
            mx = c[a[i]];
            ans = a[i];
        }
        f[j][pos[i]] = ans;
    }
}
int query(int l, int r, int x) {
    return upper_bound(v[x].begin(), v[x].end(), r) - lower_bound(v[x].begin(), v[x].end(), l);
}
int query(int l, int r) {
    int mx = 0, ans = 0;
    for (int i = L[pos[l]]; i <= R[pos[l]]; i++) {
        if (l <= i && i <= r) {
            int tmp = query(l, r, a[i]);
            if (tmp > mx || (tmp == mx && (b[a[i]] < b[ans]))) {
                mx = tmp;
                ans = a[i];
            }
        }
    }
    if (pos[l] == pos[r]) return ans;
    int tmp = query(l, r, f[pos[l] + 1][pos[r] - 1]);
    if (tmp > mx || (tmp == mx && b[f[pos[l] + 1][pos[r] - 1]] < b[ans])) {
```

```
            mx = tmp;
            ans = f[pos[l] + 1][pos[r] - 1];
        }
        for (int i = L[pos[r]]; i <= R[pos[r]]; i++) {
            if (l <= i && i <= r) {
                int tmp = query(l, r, a[i]);
                if (tmp > mx || (tmp == mx && (b[a[i]] < b[ans]))) {
                    mx = tmp;
                    ans = a[i];
                }
            }
        }
        return ans;
    }
    int main() {
        scanf("%d", &n);
        m = sqrt(n + 0.5);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            if (!M.count(a[i])) {
                M[a[i]] = ++id;
                b[id] = a[i];
            }
            a[i] = M[a[i]];
            v[a[i]].push_back(i);
            pos[i] = (i - 1) / m + 1;
        }
        k = n / m;
        if (n % m) k++;
        for (int i = 1; i <= k; i++) {
            L[i] = (i - 1) * m + 1;
            R[i] = i * m;
        }
        R[k] = n;
        for (int i = 1; i <= k; i++) init(i);
        for (int i = 1; i <= n; i++) {
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", b[query(l, r)]);
        }
        return 0;
    }
```

2.12 可持久化并查集

```
/* 可持久化线段树维护 fa 数组即可，按秩合并必需，可以同时加上路径压缩，但是
↪ 效果未必更好
 * f=1 合并 a,b 所在集合
 * f=2 回到第 k 次操作之后状态（查询也算操作）
 * f=3 询问 (bool)ab 属于一个集合？
 * 注意内存，可能要开大点或者开小点
 */
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
struct segtree {
    int lson, rson, fa, dep; //左右儿子，并查集的 fa，并查集集合内深度上限
} t[200010 << 6];
int root[200010], n, m, treesize;
inline void modify(int &k, int old, int x, int y, int pos, int newfa) {
    if (!k) t[k = ++treesize] = (segtree) {0, 0, 0, 0};
    if (x == y) {
        t[k].fa = newfa;
        t[k].dep = t[old].dep;
        return;
    }
    int mid = (x + y) >> 1;
    if (pos <= mid) modify(t[k].lson, t[old].lson, x, mid, pos, newfa);
    else modify(t[k].rson, t[old].rson, mid + 1, y, pos, newfa);
    if (!t[k].lson) t[k].lson = t[old].lson;
    if (!t[k].rson) t[k].rson = t[old].rson;
}
inline void force_modify(int &k, int old, int x, int y, int pos, int
↪ newfa) { //区别在于强制新加一条链，没有路径压缩不用写
    t[k = ++treesize] = (segtree) {0, 0, 0, 0};//就这一行改了
    if (x == y) {
        t[k].fa = newfa;
        t[k].dep = t[old].dep;
        return;
    }
    int mid = (x + y) >> 1;
    if (pos <= mid) modify(t[k].lson, t[old].lson, x, mid, pos, newfa);
    else modify(t[k].rson, t[old].rson, mid + 1, y, pos, newfa);
    if (!t[k].lson) t[k].lson = t[old].lson;
    if (!t[k].rson) t[k].rson = t[old].rson;
}
inline int askpos(int &k, int x, int y, int pos) { //返回的是树上的点编号
↪ 而不是 fa 的值
    if (x == y) return k;
```

```
    int mid = (x + y) >> 1;
    if (pos <= mid) return askpos(t[k].lson, x, mid, pos);
    else return askpos(t[k].rson, mid + 1, y, pos);
}
inline void adddep(int &k, int x, int y, int pos) {
    if (x == y) {
        t[k].dep++;
        return;
    }
    int mid = (x + y) >> 1;
    if (pos <= mid) adddep(t[k].lson, x, mid, pos);
    else adddep(t[k].rson, mid + 1, y, pos);
}
inline int getfa(int &nowroot, int x) { //返回的还是树上的点编号而不是祖先
↪ 编号
    register int pos = askpos(nowroot, 1, n, x);
    if (t[pos].fa == x) return pos;
    /* 路径压缩
    register int newfa=getfa(nowroot,t[pos].fa);
    force_modify(nowroot,nowroot,1,n,t[pos].fa,t[newfa].fa);
    return newfa;
    */
    return getfa(nowroot, t[pos].fa);
}
inline void merge(int x, int y, int &newroot, int old) {
    int px = getfa(old, x);
    int py = getfa(old, y);
    if (t[px].fa == t[py].fa) {
        newroot = old;
        return;
    }
    if (t[px].dep > t[py].dep) swap(px, py); //按秩合并 px -> py
    modify(newroot, old, 1, n, t[px].fa, t[py].fa);
    if (t[px].dep == t[py].dep) adddep(newroot, 1, n, py);
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) modify(root[0], 0, 1, n, i, i);
    int lastans = 0;
    for (int i = 1; i <= m; i++) {
        int op;
        scanf("%d", &op);
        if (op == 1) {
            int x, y;
            scanf("%d%d", &x, &y);
            x ^= lastans, y ^= lastans;
```

```
            merge(x, y, root[i], root[i - 1]);
        } else if (op == 2) {
            int k;
            scanf("%d", &k);
            k ^= lastans;
            root[i] = root[k];
        } else if (op == 3) {
            root[i] = root[i - 1];
            int x, y;
            scanf("%d%d", &x, &y);
            x ^= lastans, y ^= lastans;
            printf("%d\n", lastans = t[getfa(root[i], x)].fa ==
             ↪  t[getfa(root[i], y)].fa);
        }
    }
    return 0;
}
```

3 图论

3.1 最小生成树

3.1.1 MST for Kruskal

```
/* Kruskal 算法求 MST
 */
const int MAXN = 110; //最大点数
const int MAXM = 10000; //最大边数
int F[MAXN];//并查集使用
struct Edge {
    int u, v, w;
} edge[MAXM]; //存储边的信息，包括起点/终点/权值
int tol;//边数，加边前赋值为 0
void addedge(int u, int v, int w) {
    edge[tol].u = u;
    edge[tol].v = v;
    edge[tol++].w = w;
}
//排序函数，讲边按照权值从小到大排序
bool cmp(Edge a, Edge b) {
    return a.w < b.w;
}
int find(int x) {
    if(F[x] == -1)return x;
    else return F[x] = find(F[x]);
}
//传入点数，返回最小生成树的权值，如果不连通返回-1
int Kruskal(int n) {
    memset(F, -1, sizeof(F));
    sort(edge, edge + tol, cmp);
    int cnt = 0; //计算加入的边数
    int ans = 0;
    for(int i = 0; i < tol; i++) {
        int u = edge[i].u;
        int v = edge[i].v;
        int w = edge[i].w;
        int t1 = find(u);
        int t2 = find(v);
        if(t1 != t2) {
            ans += w;
            F[t1] = t2;
            cnt++;
        }
        if(cnt == n-1)break;
    }
```

```
    if(cnt < n-1)return -1; //不连通
    else return ans;
}
```

3.1.2 MST for Prim

```
/* Prim 求 MST
 * 耗费矩阵 cost[][], 标号从 0 开始, 0 n-1
 * 返回最小生成树的权值, 返回-1 表示原图不连通
 */
const int INF = 0x3f3f3f3f;
const int MAXN = 110;
bool vis[MAXN];
int lowc[MAXN];
//点是 0...n-1
int Prim(int cost[][MAXN], int n) {
    int ans = 0;
    memset(vis, false, sizeof(vis));
    vis[0] = true;
    for(int i = 1; i < n; i++)lowc[i] = cost[0][i];
    for(int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for(int j = 0; j < n; j++)
            if(!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        if(minc == INF)return -1; // 原图不连通
        ans += minc;
        vis[p] = true;
        for(int j = 0; j < n; j++)
            if(!vis[j] && lowc[j] > cost[p][j])
                lowc[j] = cost[p][j];
    }
    return ans;
}
```

3.2 次小生成树

```
/* 次小生成树
 * 求最小生成树时, 用数组 Max[i][j] 来表示 MST 中 i 到 j 最大边权
 * 求完后, 直接枚举所有不在 MST 中的边, 替换掉最大边权的边, 更新答案
 * 点的编号从 0 开始
 */
const int MAXN = 110;
const int INF = 0x3f3f3f3f;
```

```
bool vis[MAXN];
int lowc[MAXN];
int pre[MAXN];
int Max[MAXN][MAXN];// Max[i][j] 表示在最小生成树中从 i 到 j 的路径中的最
↪ 大边权
bool used[MAXN][MAXN];
int Prim(int cost[][MAXN], int n) {
    int ans = 0;
    memset(vis, false, sizeof(vis));
    memset(Max, 0, sizeof(Max));
    memset(used, false, sizeof(used));
    vis[0] = true;
    pre[0] = -1;
    for(int i = 1; i < n; i++) {
        lowc[i] = cost[0][i];
        pre[i] = 0;
    }
    lowc[0] = 0;
    for(int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for(int j = 0; j < n; j++)
            if(!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        if(minc == INF)return -1;
        ans += minc;
        vis[p] = true;
        used[p][pre[p]] = used[pre[p]][p] = true;
        for(int j = 0; j < n; j++) {
            if(vis[j] && j != p)Max[j][p] = Max[p][j] =
            ↪ max(Max[j][pre[p]], lowc[p]);
            if(!vis[j] && lowc[j] > cost[p][j]) {
                lowc[j] = cost[p][j];
                pre[j] = p;
            }
        }
    }
    return ans;
}
```

3.3 最短路

3.3.1 单源最短路 - Dijkstra

```cpp
struct node {
        int to, val;
        friend bool operator< (const node& a, const node& b) {
                return a.val > b.val;
        }
};
vector<node> g[maxn];
int dis[maxn];
int dijkstra(int s, int t, int n) {
        fill(dis, dis + n + 1, INF);
        priority_queue<node> q;
        dis[s] = 0;
        q.push({s, 0});
        while (!q.empty()) {
                int u = q.top().to; q.pop();
                int sz = SZ(g[u]);
                for (int i = 0; i < sz; i++) {
                        int v = g[u][i].to, w = g[u][i].val;
                        if (dis[v] > dis[u] + w)
                                q.push({v, dis[v] = dis[u] + w});
                }
        }
        return dis[t];
}
```

3.3.2 单源最短路 - SPFA

```cpp
#define inf 0x3fffffff
#define M 1005 // 最大点数
struct edge {
    int v, w, next;
} e[10005]; // 估计好有多少条边
int pre[M], cnt, dist[M], n;
bool inq[M];
void init () { // 注意初始化
    cnt = 0;
    memset (pre, -1, sizeof(pre));
}
// 注意双向加边
void addedge (int u, int v, int w) {  // 加边函数, 慢慢模拟就会明白的
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = pre[u]; //接替已有边
```

```
    pre[u] = cnt++; //自己前插成为 u 派生的第一条边
}
void spfa (int u) {
    int v, w, i;
    for (i = 1; i <= n; i++) dist[i] = inf, inq[i] = false;
    dist[u] = 0;
    deque<int> q; // 双端队列优化
    q.push_back(u);
    inq[u] = true;
    while (!q.empty()) {
        u = q.front();
        q.pop_front();
        inq[u] = false;
        for (i = pre[u]; i != -1; i = e[i].next) {
            w = e[i].w;
            v = e[i].v;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                if (!inq[v]) { // 判断负环的方法是看进入队列是否超过 n 次
                    inq[v] = true;
                    if (!q.empty() || dist[v] < dist[q.front()])
                    ↪ q.push_front(v);
                    else q.push_back(v);
                }
            }
        }
    }
}
```

3.3.3 有向图判负环

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct edge {
    int to, next;
    double v;
} e[4010];
int head[810], cnt;
inline void ins(int u, int v, double w) {
    e[++cnt].to = v;
    e[cnt].next = head[u];
    e[cnt].v = w;
    head[u] = cnt;
}
inline void insert(int u, int v, double w) {
```

```
    ins(u, v, w);
    ins(v, u, w);
}
int n, m, ans;
double dis[810];
bool vis[810];
inline void dfs(int x) {
    vis[x] = 1;
    for(int i = head[x]; i; i = e[i].next) {
        if (dis[x] + e[i].v < dis[e[i].to]) {
            if (vis[e[i].to]) {
                ans = 1;
                return;
            }
            dis[e[i].to] = dis[x] + e[i].v;
            dfs(e[i].to);
        }
    }
    vis[x] = 0;
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int x, y;
        double z;
        scanf("%d%d%lf", &x, &y, &z);
        z = -log(z);
        ins(x, y, z);
    }
    for (int i = 1; i <= n; i++) {
        memset(vis, 0, sizeof vis);
        memset(dis, 0, sizeof dis);
        dfs(i);
        if (ans)break;
    }
    puts(ans ? "inadmissible" : "admissible");
    return 0;
}
```

3.3.4 同余最短路

```
/* 例题：一张无向图每条边可以多次经过问 s 到 t 的不小于 k 的最短路
 * 解法：对于终点的边取最短的边 w 然后对%2w 跑同余最短路
 */
typedef long long LL;
const int N=1000010;
```

```
const int maxn = 100005;
const LL INF=0x3f3f3f3f3f3f3f3f;
typedef pair<LL,int>seg;
priority_queue<seg,vector<seg>,greater<seg> >q;
int stu[N],u[N],vto[N],w[N],nxt[N],n,m,tot=0;
LL d[maxn][N],W;
void add(int a,int b,int c){
    u[++tot]=a,vto[tot]=b,w[tot]=c;
    nxt[tot]=stu[u[tot]];
    stu[u[tot]]=tot;
}
void SPFA(int src)
{
    for(int i = 0;i <= n;i ++ )for(int j = 0;j < W;j ++ ) d[i][j]=INF;
    q.push(make_pair(0,src));
    while(!q.empty())
    {
        seg now = q.top(); q.pop();
        LL wnow = now.first;
        int x = now.second;
        if(wnow > d[x][wnow%W])continue;
        for(int e = stu[x] ; e!=-1 ; e=nxt[e]){
            LL nw = wnow + w[e];
            int v = vto[e];
            if(d[v][nw % W] > nw){
                d[v][nw % W] = nw;
                q.push( make_pair (nw,v));
            }
        }
    }
}
```

3.3.5 Floyd 倍增法

```
/* 例题：给出一张图，求经过 k 条边的最短路。
 * 解法：魔改矩阵乘法，换成 floyd 传递形式，那么矩阵的 k 次方就是答案
 *      注意到 n 很大，因此需要快速幂加速矩阵乘法
 */
#include <cstdio>
#include <algorithm>
#include <map>
#include <cstring>
using namespace std;
const int MAXN = 210;
int ID[1010];
int n, k, q, z, tn;
```

```
struct Matrix {
    int a[MAXN][MAXN];
    Matrix operator * (const Matrix &x) const {
        Matrix c;
        memset(c.a, 0x3f, sizeof(c.a));
        for (int k = 1; k <= tn; k++)
            for (int i = 1; i <= tn; ++i)
                for (int j = 1; j <= tn; ++j)
                    c.a[i][j] = min(c.a[i][j], a[i][k] + x.a[k][j]);
        return c;
    }
} s, ans;
void quickpow() {
    ans = s;
    k--;
    for (; k; k >>= 1) {
        if (k & 1) ans = ans * s;
        s = s * s;
    }
}
int main() {
    scanf("%d%d%d%d", &k, &n, &q, &z);
    memset(s.a, 0x3f, sizeof(s.a));
    for (int i = 1; i <= n; ++i) {
        int x, y, w;
        scanf("%d%d%d", &w, &x, &y);
        if (!ID[x]) ID[x] = ++tn;
        if (!ID[y]) ID[y] = ++tn;
        s.a[ID[x]][ID[y]] = s.a[ID[y]][ID[x]] = w;
    }
    quickpow();
    printf("%d\n", ans.a[ID[q]][ID[z]]);
    return 0;
}
```

3.4 差分约束

- 什么是差分约束系统

对于一组不等式:

$$\begin{cases} x_1 - x_2 \le 0 \\ x_1 - x_5 \le 1 \\ x_2 - x_5 \le 1 \\ x_3 - x_1 \le 5 \\ x_4 - x_1 \le 4 \\ x_4 - x_3 \le -1 \\ x_5 - x_3 \le -3 \\ x_5 - x_4 \le -3 \end{cases}$$

特点是全都是两个未知数的差**小于等于**某个常数（大于等于也可以，因为左右乘 -1 就可以化成小于等于的形式)，这样的不等式组称作**差分约束系统**。

这个不等式组**要么无解，要么就有无限组解**。因为如果存在一组解 $\{x_1, x_2, \cdots, x_n\}$ 的话，那么对于任何一个常数 k 有 $\{x_1 + k, x_2 + k, \cdots, x_n + k\}$ 也肯定是一组解，因为任何两个数加上一个数以后，它们之间的关系（差）是不变的，这个差分约束系统中的所有不等式都不会被破坏。

- 与最短路的关系

差分约束系统的解法用到了**单源最短路径**问题中的三角形不等式。即对有向图中任意一条边 $<u, v>$ 都有:$dis[v] \le dis[u] + len[u][v]$，其中 $dis[u]$ 和 $dis[v]$ 是从源点分别到点 u 和点 v 的最短路径的长度，$len[u][v]$ 是边 $<u, v>$ 的长度值。

这是显然的: 如果存在顶点 u 到顶点 v 的有向边，那么**从源点到顶点 v 的最短路径长度**小于等于**从源点到顶点 u 的最短路径长度加上边 $<u, v>$ 的长度值**。

显然上述的不等式就是所描述的，也和差分约束系统中的不等式相同，因此就可以把差分约束系统转化成一张图。

- 构图求解

1. 基本构图
   每个未知数 x_i 对应图中的一个顶点 v_i，把所有的不等式都化成图中的一条边，对于不等式 $x_i - x_j \le y$ 化成三角形不等式 $x_i \le x_j + y$ 就可以化成边 $< v_j, v_i >$ 权值为 y。最后在这张图上求一遍**单源最短路**，这些三角不等式就全部满足了，因为它是最短路问题的基本性质。

2. 求解
   以 1 为起点，则起点到各个顶点的最短距离为 $dis[1] = 0, dis[2] = 0, dis[3] = 5, dis[4] = 4, dis[5] = 1$ 则得到解 $x_1 = 0, x_2 = 1, x_3 = 5, x_4 = 4, x_5 = 1$
   以 3 为起点,则起点到各个顶点的最短距离为 $dis[1] = -5, dis[2] = -5, dis[3] = 0, dis[4] = -1, dis[5] = -4$ 则得到解 $x_1 = -5, x_2 = -4, x_3 = 0, x_4 = -1, x_5 = -4$
   以不同顶点作为起点会得到不同的解，但这些解都一定合理。
   什么情况下无解？**存在负权回路！**

$$
\begin{aligned}
X1 - X2 &<= 0\\
X1 - X5 &<= -1\\
X2 - X5 &<= 1\\
X3 - X1 &<= 5\\
X4 - X1 &<= 4\\
X4 - X3 &<= -1\\
X5 - X3 &<= -3\\
X5 - X4 &<= -3
\end{aligned}
$$

Figure 3: 不等式组 (1)

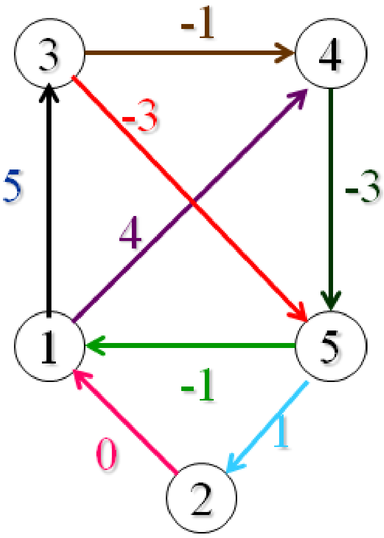

Figure 4: 转换成的顶点图

3. 增加源点

具体实现时，往往在原图上附加一个顶点，这个顶点与每个顶点都连接一条权值为 0 的边，以上述不等式为例，也就是新加入一个未知数 x_0，然后对每个未知数都对 x_0 加一个不等式，得到这样的不等式组 (2)，化成顶点图如下:

$$
\begin{aligned}
X1 - X0 &<= 0\\
X2 - X0 &<= 0\\
X3 - X0 &<= 0\\
X4 - X0 &<= 0\\
X5 - X0 &<= 0
\end{aligned}
$$

Figure 5: 不等式组 (2)

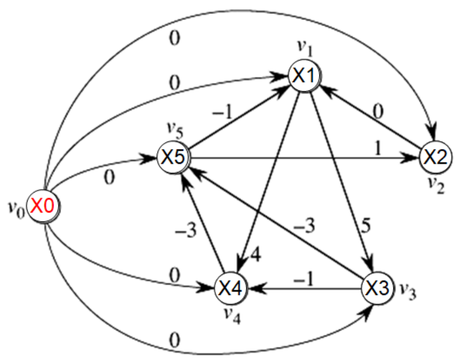

Figure 6: 转换成的顶点图

图中**每一条边**都代表差分约束系统的**一个不等式**。现在以 v_0 为源点，求单源最短路，**最终得到的 v_0 到 v_i 的最短路径长度就是 x_i 的一个解**。如图中 v_0 到其他各个顶点的最短距离分别是 $\{-5,-3,0,-1,-4\}$，因此满足上述不等式的一组解就是 $\{x_1,x_2,x_3,x_4,x_5\}=\{-5,-3,0,-1,-4\}$。当然把每个数都加上 10 也是一组解 $\{5,7,10,9,6\}$，但是这组解只满足不等式组 (1)，也就是原先的差分约束系统，而不满足不等式组 (2)，也是我们后来加上的那些不等式。当然这是无关紧要的，因为 x_0 本来就是个局外人，并不在乎它。

对于上面例子而言，它代表的解 x_0 值也在其中也就是 $x_0=0$。但是 x_0 的值是无可争议的，既然是以它作为源点求最短路径，那么源点到它的最短路径

当然是 0 了，因此，我们解这个差分约束系统无形中存在一个条件 $x_0 = 0$，那么它有什么用呢？**可以限制所有的未知数的解都不大于 0** 。

一个有趣的结论: 当我们一开始就把 x_0 的解死定为 A 的时候，所有未知数的解都不会大于 A（一开始把 $dis[0] = A$）

- 总结
  差分约束求解不等式组分为两种方法: **最短路**和**最长路**。其中最短路对应最大解，最长路对应最小解，特别地，若有 $x_i = x_j$ 的情况，建图时候，从 x_i 出发连一条权值为 0 的边到 x_j，同时也建一条同等权值的反向边。

3.5 K 短路

```
#include <bits/stdc++.h>
#include <ext/pb_ds/priority_queue.hpp>
#define pa pair<int,int>
#define inf 2000000000
using namespace std;
using namespace __gnu_pbds;
inline int read() {
    int x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9') {
        if(ch == '-')f = -1;
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
typedef __gnu_pbds::priority_queue<pa, greater<pa> > heap;
int n, m, K, cnt, cnt2;
int ans[105];
int d[10005], g[10005], tim[10005];
int last[10005], last2[10005];
heap q;
heap::point_iterator id[10005];
struct edge {
    int to, next, v;
} e[10005], ed[10005];
void insert(int u, int v, int w) {
    e[++cnt].to = v;
    e[cnt].next = last[u];
    last[u] = cnt;
    e[cnt].v = w;
```

```
    ed[++cnt2].to = u;
    ed[cnt].next = last2[v];
    last2[v] = cnt2;
    ed[cnt2].v = w;
}
void dijkstra() {
    for(int i = 1; i <= n; i++)d[i] = inf;
    d[1] = 0;
    id[1] = q.push(make_pair(0, 1));
    while(!q.empty()) {
        int now = q.top().second;
        q.pop();
        for(int i = last2[now]; i; i = ed[i].next)
            if(ed[i].v + d[now] < d[ed[i].to]) {
                d[ed[i].to] = d[now] + ed[i].v;
                if(id[ed[i].to] != 0)
                    q.modify(id[ed[i].to], make_pair(d[ed[i].to], ed[i].to));
                else id[ed[i].to] = q.push(make_pair(d[ed[i].to], ed[i].to));
            }
    }
}
void astar() {
    if(d[n] == inf) return;
    q.push(make_pair(d[n], n));
    while(!q.empty()) {
        int now = q.top().second, dis = q.top().first;
        q.pop();
        tim[now]++;
        if(now == 1)ans[tim[now]] = dis;
        if(tim[now] <= K)
            for(int i = last[now]; i; i = e[i].next)
                q.push(make_pair(dis - d[now] + d[e[i].to] + e[i].v, e[i].to));
    }
}
int main() {
    n = read();
    m = read();
    K = read();
    for(int i = 1; i <= m; i++) {
        int u = read(), v = read(), w = read();
        insert(u, v, w);
    }
    dijkstra();
```

```
    astar();
    for(int i = 1; i <= K; i++)
        if(ans[i])printf("%d\n", ans[i]);
        else puts("-1");
    return 0;
}
```

3.6 图的割点、桥和双连通分支的基本概念

3.6.1 点连通度与边连通度

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为割点集合。一个图的点连通度的定义为，最小割点集合中的顶点数。

3.6.2 双连通图、割点与桥

如果一个无向连通图的点连通度大于 1，则称该图是点双连通的 (point biconnected)，简称双连通或重连通。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点 (cut point)，又叫关节点 (articulation point)。

如果一个无向连通图的边连通度大于 1，则称该图是边双连通的 (edge biconnected)，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为桥 (bridge)，又叫关节边 (articulation edge)。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。

3.6.3 双联通分支

在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个双连通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为极大双连通子图。双连通分支 (biconnected component)，或重连通分支，就是图的极大双连通子图。特殊的，点双连通分支又叫做块。

3.6.4 求割点与桥

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 $DFS(u)$ 为 u 在搜索树 (以下简称为树) 中被遍历到的次序号。定义 $Low(u)$ 为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据定义，则有:$Low(u) = Min\{DFS(u), DFS(v)\}(u, v)$ 为后向边 (返祖边) 等价于 $DFS(v) < DFS(u)$ 且 v 不为 u 的父亲节点 $Low(v)(u, v)$ 为树枝边 (父子边) 一个顶点 u 是割点，当且仅当满足 (1) 或 (2)

1. u 为树根，且 u 有多于一个子树。
2. u 不为树根，且满足存在 (u, v) 为树枝边 (或称父子边，即 u 为 v 在搜索树中的父亲)，使得 $DFS(u) \leq Low(v)$。

一条无向边 (u, v) 是桥，当且仅当 (u, v) 为树枝边，且满足 $DFS(u) < Low(v)$。

3.6.5 求双连通分支

下面要分开讨论点双连通分支与边双连通分支的求法:

- 对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向边 (非横叉边)，就把这条边加入栈中。如果遇到某时满足 $DFS(u) \leq Low(v)$，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边 (u, v)，取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。
- 对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。

3.6.6 构造双连通图

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 $leaf$。则至少在树上添加 $\frac{leaf+1}{2}$ 条边，就能使树达到边二连通，所以至少添加的边数就是 $\frac{leaf+1}{2}$。具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是 $\frac{leaf+1}{2}$ 次，把所有点收缩到了一起。

3.7 有向图强连通分量

```
/* tarjan 缩点找出度为 0 点
 */
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
const int maxn = 1e4 + 10;
/* head */
int low[maxn], dfn[maxn];
bool vis[maxn];
int belong[maxn], son[maxn], scc, cnt;
vector<int> g[maxn];
stack<int> s;
int out[maxn];
inline void init() {
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(son, 0, sizeof son);
    memset(vis, 0, sizeof vis);
```

```
    cnt = scc = 0;
    memset(out, 0, sizeof out);
}
void tarjan(int u) {
    low[u] = dfn[u] = ++cnt;
    s.push(u);
    vis[u] = 1;
    for (int v: g[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v]) low[u] = min(low[u], dfn[v]);
    }
    if (low[u] == dfn[u]) {
        ++scc;
        int v;
        do {
            v = s.top();
            s.pop();
            vis[v] = 0;
            belong[v] = scc;
            son[scc]++;
        } while (u != v);
    }
}
inline void rebuild(int n) {
    rep(u, 1, n + 1) {
        for (int v: g[u]) {
            if (belong[v] != belong[u]) {
                ++out[belong[u]];
            }
        }
    }
}
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    init();
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        g[a].push_back(b);
    }
    rep(i, 1, n + 1) if (!dfn[i]) tarjan(i);
    rebuild(n);
    int ans = 0;
```

```
    rep(i, 1, scc + 1) {
        if (!out[i]) {
            ans = son[i];
            break;
        }
    }
    printf("%d\n", ans);
    return 0;
}
/* 例题：codeforces - 962F 找简单边双连通分量 并输出
 */
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 1e5 + 5;
struct Edge {
    int to, id, next;
} e[maxn << 1];
int head[maxn], cnt;
int low[maxn], dfn[maxn];
bool vis[maxn];
int belong[maxn], block, dep;
stack<int> s;
vector<int> ans;
inline void init() {
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(vis, 0, sizeof vis);
    memset(belong, 0, sizeof belong);
    block = cnt = dep = 0;
    while (!s.empty()) s.pop();
}
inline void addedge(int u, int v, int t) {
    e[cnt].to = v;
    e[cnt].id = t;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++dep;
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        if (!dfn[v]) {
            s.push(i);
```

```
                tarjan(v, u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    block++;
                    bool ok = 1;
                    vector<int> a;
                    int tmp;
                    do {
                        tmp = s.top(); s.pop();
                        int x = e[tmp].to;
                        a.push_back(e[tmp].id);
                        if (belong[x] != block) belong[x] = block;
                        else ok = 0;
                    } while (tmp != i);
                    if (ok && a.size() > 1) {
                        for (int j: a) ans.push_back(j);
                    }
                }
            } else if (dfn[v] < dfn[u]) {
                low[u] = min(low[u], dfn[v]);
                s.push(i);
            }
        }
}
int main() {
    int n, m;
    init();
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        addedge(u, v, i);
        addedge(v, u, i);
    }
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) tarjan(i, -1);
    }
//    for (int i = 1; i <= n; i++) printf("%d\n", belong[i]);
//    puts("");
    sort(ans.begin(), ans.end());
    unique(ans.begin(), ans.end());
    printf("%d\n", ans.size());
    for (int i = 0; i < ans.size(); i++) {
        if (i == 0) printf("%d", ans[i]);
        else printf(" %d", ans[i]);
    }
```

```
    puts("");
    return 0;
}
```

3.8 割点、桥

```
/* 求无向图的割点和桥
 * 可以找出割点和桥，求删掉每个点后增加的连通块。
 * 需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重
 */
const int MAXN = 10010;
const int MAXM = 100010;
struct Edge {
    int to, next;
    bool cut;//是否为桥的标记
} edge[MAXM];
int head[MAXN], tot;
int Low[MAXN], DFN[MAXN], Stack[MAXN];
int Index, top;
bool Instack[MAXN];
bool cut[MAXN];
int add_block[MAXN];//删除一个点后增加的连通块
int bridge;
void addedge(int u, int v) {
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].cut = false;
    head[u] = tot++;
}
void Tarjan(int u, int pre) {
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    int son = 0;
    int pre_cnt = 0; //处理重边，如果不需要可以去掉
    for(int i = head[u]; i != ?1; i = edge[i].next) {
        v = edge[i].to;
        if(v == pre && pre_cnt == 0) {
            pre_cnt++;
            continue;
        }
        if( !DFN[v] ) {
            son++;
            Tarjan(v, u);
            if(Low[u] > Low[v])Low[u] = Low[v];
```

```
            //桥
            //一条无向边 (u,v) 是桥，当且仅当 (u,v) 为树枝边，且满足
            ↪ DFS(u) < Low(v)。
            if(Low[v] > DFN[u]) {
                bridge++;
                edge[i].cut = true;
                edge[i ^ 1].cut = true;
            }
            //割点
            if(u != pre && Low[v] >= DFN[u]) { //不是树根
                cut[u] = true;
                add_block[u]++;
            }
        } else if( Low[u] > DFN[v])
            Low[u] = DFN[v];
    }
    //树根，分支数大于 1
    if(u == pre && son > 1)cut[u] = true;
    if(u == pre)add_block[u] = son ? 1;
    Instack[u] = false;
    top??;
}
```

3.9 二分图基础

- **定义:** 能把所有点划分成两个集合，使得每条边的两端点分属于不同集合的无向图。
- **二分图最大匹配:** 二分图中取最多的边，使得任意两条边不经过同一个点。——【最大流】或【匈牙利算法】。对于最大流做法，注意到每个点只被最多一条边取到，建图显然。
- **二分图最优匹配/最大权匹配:** 二分图中取边，使得任意两条边不经过同一个点，且边权和最大。——【KM 算法】
- **二分图最小点覆盖:** 取最少的点，使得二分图中的每条边都经过其中至少一个点。二分图最小点覆盖 = 二分图最大匹配。
- **二分图最小边覆盖/最小（不相交）路径覆盖:** 取最少的不相交简单路径覆盖二分图中的所有顶点。注意取的是**路径**不是边，取边没有意义，因为最大匹配就是算这个的，最大匹配也不一定都能覆盖。——二分图最小边覆盖 = n - 二分图最大匹配
- **二分图最大独立集:** 在二分图中找一个点集，使得集合中任意两点之间没有边。(就像有向图的“反链”) ——二分图最大独立集 = n - 二分图最大匹配

3.10 有向无环图 (DAG) 基础

- **链:** 一个点集，对其中任意两点 x, y，要么 x 能到达 y，要么 y 能到达 x。DAG 不存在双方互通的情况。
- **反链:** 一个点集，其中任意两点 x, y，满足 x, y 都互不可达。(反链不是反着建边，是一个集合一样的东西)
- **最长反链:** 最大的点集，其中任意两点 x, y，满足 x, y 都互不可达。(感觉 “最长” 用词不是很恰当？) ——最长反链 = 最小可相交路径覆盖（Dilworth 定理）
- **最小可相交路径覆盖:** 取最少的链，使得任意一个点都出现在至少一条链中，不同链允许经过同一点。
  - 最小链覆盖一般指的是最小可相交路径覆盖。
  - 最小可相交路径覆盖 = 对 DAG 做 $Floyd$ 处理 + 最小不相交路径覆盖
- **最小不相交路径覆盖:** 取最少的链，使得任意一个点都出现且仅出现在一条链中，不同链不允许经过同一点。
  - **做法:** 拆成二分图做最大匹配。
  - 设原图 n 个点，每个点拆成两点 x, x'，原图中一条单向边 (x, y) 对应新图中的双向边 (x, y')。答案是（$n-$ 新图的二分图最大匹配）。其中 n 指的是原图的点数。

3.11 二分图匹配

3.11.1 匈牙利

- DFS 版本

```
// 链表写法
// 实际上，在处理二分图匹配问题时，边可以加单向边，扫点也可以只扫两个区
↪ 间中的一个区间即可
#define rep(i,a,n) for (int i=a;i<n;i++)
vector<int> g[maxn];
int match[maxn];
bool vis[maxn];
bool dfs(int u) {
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (!vis[v]) {
            vis[v] = 1;
            if (match[v] == -1 || dfs(match[v])) {
                match[v] = u;
                return 1;
            }
        }
    }
```

```
    return 0;
}
int bipartite_matching() {
        int ans = 0;
        memset(match, -1, sizeof match);
        rep(u, 1, n + 1) {
                if (match[u] == -1) {
                        memset(vis, 0, sizeof vis);
                        if (dfs(u))
                                ans++;
                }
        }
        return ans;
}
// 邻接矩阵
int g[N][N], match[N];
bool vis[N];
int n, m; // n 个点 m 个边
bool dfs(int u) {
        rep(v, 1, n + 1) {
                if (g[u][v] && !vis[v]) {
                        vis[v] = 1;
                        if (match[v] == -1 || dfs(match[v])) {
                                match[v] = u;
                                return 1;
                        }
                }
        }
        return 0;
}
int hungary() {
        int ans = 0;
        memset(match, -1, sizeof match);
        rep(u, 1, m + 1) {
                memset(vis, 0, sizeof vis);
                if (dfs(u))
                        ans++;
        }
        return ans;
}
```

- BFS 版本

```
/* 算二分图最大匹配, bfs 版
 * 以下是二分图左右点已经分开的版本, 往下翻还有左右点没分开的版本
 * 编号 (1)~(n1) 的点是第一个集合, 编号 (n1+1)~(n2) 的点是第二个集合
*/
```

```
#define M 2010
#define N 1010
struct edge {
    int to, next;
} e[M];
int head[N], cnt;
int hungary_mrk[N];
int hungary_pre[N];
int match[N];
inline void ins(int u, int v) {
    e[++cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}
inline void insert(int u, int v) {
    ins(u, v);
    ins(v, u);
}
queue<int>Q;
int hungary(int n1, int n2) {
    int tot = 0;
    memset(match, 0, sizeof(match));
    memset(hungary_mrk, 0, sizeof(hungary_mrk));
    for(int i = 1; i <= n1; i++)
        if (!match[i]) {
            while(!Q.empty())Q.pop();
            Q.push(i);
            hungary_pre[i] = 0;
            bool flg = 0;
            while(!Q.empty() && !flg) {
                int now = Q.front();
                Q.pop();
                for (int j = head[now]; j && !flg; j = e[j].next) {
                    int nex = e[j].to;
                    if (hungary_mrk[nex] != i) {
                        hungary_mrk[nex] = i;
                        Q.push(match[nex]);
                        if (match[nex])hungary_pre[match[nex]] = now;
                        else {
                            flg = 1;
                            int uu = now, vv = nex;
                            while (uu) {
                                int tt = match[uu];
                                match[uu] = vv;
                                match[vv] = uu;
```

```
                                                        uu = hungary_pre[uu];
                                                        vv = tt;
                                                }
                                        }
                                }
                        }
                }
                if (match[i])tot++;
        }
    return tot;
}
//以上是二分图左右已经分开的版本
//以下是二分图左右编号打乱的版本
#define M 2010
#define N 1010
struct edge {
    int to, next;
} e[M];
int head[N], cnt;
int hungary_mrk[N];
int hungary_pre[N];
int hungary_left[N];
int match[N];
inline void ins(int u, int v) {
    e[++cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}
inline void insert(int u, int v) {
    ins(u, v);
    ins(v, u);
}
queue<int>Q2;
queue<pa >Q1;
int hungary(int n) {
    int tot = 0;
    memset(match, 0, sizeof(match));
    memset(hungary_mrk, 0, sizeof(hungary_mrk));
    memset(hungary_left, -1, sizeof(hungary_left));
    for (int i = 1; i <= n; i++)
        if (hungary_left[i] == -1) {
            while (!Q1.empty())Q1.pop();
            Q1.push(mkp(i, 1));
            while (!Q1.empty()) {
                int now = Q1.front().first, op = Q1.front().second;
                Q1.pop();
```

```cpp
                hungary_left[now] = op;
                for (int j = head[now]; j; j = e[j].next)
                    if (hungary_left[e[j].to] == -1)Q1.push(mkp(e[j].to, op ^ 1));
            }
        }
    for(int i = 1; i <= n; i++)
        if (hungary_left[i] && !match[i]) {
            while(!Q2.empty())Q2.pop();
            Q2.push(i);
            hungary_pre[i] = 0;
            bool flg = 0;
            while(!Q2.empty() && !flg) {
                int now = Q2.front();
                Q2.pop();
                for (int j = head[now]; j && !flg; j = e[j].next) {
                    int nex = e[j].to;
                    if (hungary_mrk[nex] != i) {
                        hungary_mrk[nex] = i;
                        Q2.push(match[nex]);
                        if (match[nex])hungary_pre[match[nex]] = now;
                        else {
                            flg = 1;
                            int uu = now, vv = nex;
                            while (uu) {
                                int tt = match[uu];
                                match[uu] = vv;
                                match[vv] = uu;
                                uu = hungary_pre[uu];
                                vv = tt;
                            }
                        }
                    }
                }
            }
            if (match[i])tot++;
        }
    return tot;
}
```

3.11.2 Hopcroft-Karp

```cpp
/* [LightOJ-1356] - 最大点独立集 + 素数筛
 * Hopcroft-Carp 算法适用于点集稠密的二分图，相较于匈牙利更有优势
 */
#include <bits/stdc++.h>
```

```
using namespace std;
const int maxn = 4e4 + 5;
const int maxm = 5e5 + 5;
vector<int> prime, g[maxn], v[maxn];
int a[maxn], p[2][maxm], mx[maxn], my[maxn], dx[maxn], dy[maxn];
bool vis[maxm], used[maxn];
int n, dis;
inline void init() {
    memset(vis, 0, sizeof vis);
    for (int i = 2; i <= maxm - 5; i++) {
        if (!vis[i]) {
            prime.push_back(i);
            for (int j = 2*i; j <= maxm - 5; j += i) vis[j] = 1;
        }
    }
}
bool find() {
    queue<int> q;
    dis = 1e9;
    memset(dx, -1, sizeof dx);
    memset(dy, -1, sizeof dy);
    for (int i = 0; i < n; i++) {
        if (mx[i] == -1) {
            q.push(i);
            dx[i] = 0;
        }
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        if (dx[u] > dis) break;
        for (int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            if (dy[v] == -1) {
                dy[v] = dx[u] + 1;
                if (my[v] == -1) dis = dy[v];
                else {
                    dx[my[v]] = dy[v] + 1;
                    q.push(my[v]);
                }
            }

        }
    }
    return dis != 1e9;
}
```

```
bool dfs(int u) {
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (!used[v] && dy[v] == dx[u] + 1) {
            used[v] = 1;
            if (my[v] != -1 && dy[v] == dis) continue;
            if (my[v] == -1 || dfs(my[v])) {
                my[v] = u;
                mx[u] = v;
                return 1;
            }
        }
    }
    return 0;
}
inline void solve(int kase) {
    scanf("%d", &n);
    for (int i = 0; i <= n; i++) {
        g[i].clear();
        v[i].clear();
    }
    memset(p, 0, sizeof p);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        int tmp = a[i], cnt= 0;;
        for (int j = 0; prime[j] * prime[j] <= tmp; j++) {
            if (tmp % prime[j] == 0) {
                v[i].push_back(prime[j]);
                while (tmp % prime[j] == 0) {
                    cnt++;
                    tmp /= prime[j];
                }
            }
        }
        if (tmp > 1) {
            v[i].push_back(tmp);
            cnt++;
        }
        p[cnt&1][a[i]] = i;
    }
    for (int i = 1; i<= n; i++) {
        if (p[0][a[i]]) {
            for (int j = 0; j < v[i].size(); j++) {
                int tmp = a[i] / v[i][j];
                if (!p[1][tmp]) continue;
                g[i-1].push_back(p[1][tmp]-1);
```

```
            }
        } else {
            for (int j = 0;  j < v[i].size(); j++) {
                int tmp = a[i] / v[i][j];
                if (!p[0][tmp]) continue;
                g[p[0][tmp]-1].push_back(i-1);
            }
        }
    }
    int cnt = 0;
    memset(mx, -1, sizeof mx);
    memset(my, -1, sizeof my);
    while (find()) {
        memset(used, 0, sizeof used);
        for (int i = 0; i < n; i++) {
            if (mx[i] == -1 && dfs(i)) cnt++;
        }
    }
    printf("Case %d: %d\n", kase, n-cnt);
}
int main() {
    init();
    int t;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++) solve(i);
    return 0;
}
```

3.11.3 多重匹配

```
/* 例题：POJ-2289 有一个通讯录，里面有 n 个人，现在要把他们分成 m 组，给出
 ↪ 每个人的名字，和想要把他们分在的组号. 求怎么分组，能使人数最多的那个组
 ↪ 的人最少.
 * 解法：二分这个人数，看最大匹配是否为 n，最后输出这个人数.
 */
#include <bits/srdc++.h>
using namespace std;
const int maxn = 1005;
const int maxm = 505;
vector<int> g[maxn];
int match[maxn][maxm], a[maxn];
bool vis[maxn];
bool dfs(int u) {
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (!vis[v]) {
```

```
                vis[v] = 1;
                if (match[v][0] < a[v]) {
                    match[v][++match[v][0]] = u;
                    return 1;
                }
                for (int j = 1; j <= match[v][0]; j++) {
                    if (dfs(match[v][j])) {
                        match[v][j] = u;
                        return 1;
                    }
                }
            }
        }
    return 0;
}
inline bool check(int x, int n) {
    memset(match, -1, sizeof match);
    for (int i = 0; i < n; i++) {
        a[i] = x;
        match[i][0] = 0;
    }
    int res = 0;
    for (int i = 0; i < n; i++) {
        memset(vis, 0, sizeof vis);
        if (dfs(i)) res++;
    }
    return res == n;
}
char s[maxn][maxm];
int main() {
    int n, m;
    while (~scanf("%d%d", &n, &m), n + m) {
        for (int i = 0; i < n; i++) g[i].clear();
        for (int i = 0; i < n; i++) {
            scanf("%s", s[i]);
            while (getchar() != '\n') {
                int v;
                scanf("%d", &v);
                g[i].push_back(v);
            }
        }
        int l = 0, r = n, ans = 0;
        while (l <= r) {
            int mid = l + r >> 1;
            if (check(mid, n)) {
                ans = mid;
```

```
                r = mid - 1;
            } else l = mid + 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.11.4 最大权匹配

```
/* 最大权完美匹配：一个两边各 n 个点的二分图，中间边的权值由 g[i][j] 给出，要求找出一个包含 n 条边的完美匹配，使得权值和最大。
 * 最小权完美匹配的做法是把边取反跑一遍，然后答案再取反
 */
#define N 310
#define inf 0x3f3f3f3f
int g[N][N];
int lx[N], ly[N];
int match[N];
bool vis_x[N], vis_y[N];
int slack[N];
int n;
bool dfs(int cur) {
    vis_x[cur] = 1;
    int t;
    for(int y = 1; y <= n; y++) {
        if(vis_y[y])continue;
        t = lx[cur] + ly[y] - g[cur][y];
        if(!t) {
            vis_y[y] = 1;
            if(match[y] == -1 || dfs(match[y])) {
                match[y] = cur;
                return 1;
            }
        } else if(slack[y] > t)slack[y] = t;
    }
    return 0;
}
int KM() {
    memset(match, -1, sizeof(match));
    memset(ly, 0, sizeof(ly));
    for(int i = 1; i <= n; i++) {
        lx[i] = -inf;
        for(int j = 1; j <= n; j++)
            if(g[i][j] > lx[i])lx[i] = g[i][j];
    }
```

```
    for(int x = 1; x <= n; x++) {
        for(int i = 1; i <= n; i++)slack[i] = inf;
        int d;
        while(1) {
            memset(vis_x, 0, sizeof(vis_x));
            memset(vis_y, 0, sizeof(vis_y));
            if(dfs(x))break;
            d = inf;
            for(int i = 1; i <= n; i++)
                if(!vis_y[i] && d > slack[i])d = slack[i];
            for(int i = 1; i <= n; i++)
                if(vis_x[i])lx[i] -= d;
            for(int i = 1; i <= n; i++)
                if(vis_y[i])ly[i] += d;
                else slack[i] -= d;
        }
    }
    int res = 0;
    for(int i = 1; i <= n; i++)
        if(match[i] > -1)res += g[match[i]][i];
    return res;
}
```

3.12 一般图匹配

3.12.1 带花树算法

```
// URAL 1099
#include <bits/stdc++.h>
using namespace std;
const int maxn = 230;
int nxt[maxn], spouse[maxn];
int pa[maxn];
int find(int x) {
    if (x == pa[x]) return x;
    return pa[x] = find(pa[x]);
}
inline void merge(int u, int v) {
    int fu = find(u);
    int fv = find(v);
    if (fu != fv) pa[fu] = fv;
}
vector<int> g[maxn];
int mark[maxn], vis[maxn];
int Q[maxn], l, r;
int n, t = 0; //点的个数, 点的编号从 1 到 n
inline int LCA(int x, int y) {
```

```
    ++t;
    while (1) {
        if (x != -1) {
            x = find(x);
            if (vis[x]== t) return x;
            vis[x] = t;
            if (spouse[x] != -1) x = nxt[spouse[x]];
            else x = -1;
        }
        swap(x, y);
    }
}
inline void group(int a, int p) {
    while (a != p) {
        int b = spouse[a], c = nxt[b];
        if (find(c) != p) nxt[c] = b;
        if (mark[b] == 2) mark[Q[r++] = b] = 1;
        if (mark[c] == 2) mark[Q[r++] = c] = 1;
        merge(a, b); merge(b, c);
        a = c;
    }
}
inline void Augment(int s) {
    for (int i = 0; i < n; i++) {
        nxt[i] = -1;
        vis[i] = -1;
        pa[i] = i;
        mark[i] = 0;
    }
    Q[l = 0] = s; r = 1; mark[s] = 1;
    while (l < r && spouse[s] == -1) {
        int u = Q[l++];
        for (int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            int fu = find(u);
            int fv = find(v);
            if (spouse[u] != v && fu != fv && mark[v] != 2) {
                if (mark[v] == 1) {
                    int p = LCA(u, v);
                    if (fu != p) nxt[u] = v;
                    if (fv != p) nxt[v] = u;
                    group(u, p); group(v, p);
                } else if (spouse[v] == -1) {
                    nxt[v] = u;
                    int now = v;
                    while (now != -1) {
```

```
                        int p = nxt[now];
                        int tmp = spouse[p];
                        spouse[now] = p; spouse[p] = now;
                        now = tmp;
                    }
                    break;
                } else {
                    nxt[v] = u;
                    mark[Q[r++] = spouse[v]] = 1;
                    mark[v] = 2;
                }
            }
        }
    }
}
bool m[maxn][maxn];
int main() {
    int x, y;
    scanf("%d", &n);
    memset(m, 0, sizeof m);
    while (~scanf("%d%d", &x, &y)) {
        x--; y--;
        if (x != y && !m[x][y]) {
            g[x].push_back(y);
            g[y].push_back(x);
            m[x][y] = m[y][x] = 1;
        }
    }
    for (int i = 0; i < n; i++) spouse[i] = -1;
    for (int i = 0; i < n; i++) {
        if (spouse[i] == -1) Augment(i);
    }
    int res = 0; //匹配数，匹配对数是 Count/2
    for (int i = 0; i < n; i++) {
        if (spouse[i] != -1) res++;
    }
    printf("%d\n", res);
    for (int i = 0; i < n; i++) {
        if (spouse[i] != -1 && spouse[i] > i) {
            printf("%d %d\n", i + 1, spouse[i] + 1);
        }
    }
    return 0;
}
```

3.12.2 最大加权匹配

```
// 一般图的最大加权匹配模板
// 注意 G 的初始化，需要偶数个点，刚好可以形成 n/2 个匹配
// 如果要求最小权匹配，可以取相反数，或者稍加修改就可以了
// 点的编号从 0 开始的
const int MAXN = 110;
const int INF = 0x3f3f3f3f;
int G[MAXN][MAXN];
int cnt_node;// 点的个数
int dis[MAXN];
int match[MAXN];
bool vis[MAXN];
int sta[MAXN], top; // 堆栈
bool dfs(int u) {
    sta[top++] = u;
    if(vis[u])return true;
    vis[u] = true;
    for(int i = 0; i < cnt_node; i++)
        if(i != u && i != match[u] && !vis[i]) {
            int t = match[i];
            if(dis[t] < dis[u] + G[u][i] - G[i][t]) {
                dis[t] = dis[u] + G[u][i] - G[i][t];
                if(dfs(t))return true;
            }
        }
    top--;
    vis[u] = false;
    return false;
}
int P[MAXN];
// 返回最大匹配权值
int get_Match(int N) {
    cnt_node = N;
    for(int i = 0; i < cnt_node; i++)P[i] = i;
    for(int i = 0; i < cnt_node; i += 2) {
        match[i] = i + 1;
        match[i + 1] = i;
    }
    int cnt = 0;
    while(1) {
        memset(dis, 0, sizeof(dis));
        memset(vis, false, sizeof(vis));
        top = 0;
        bool update = false;
        for(int i = 0; i < cnt_node; i++)
            if(dfs(P[i])) {
```

```
                    update = true;
                    int tmp = match[sta[top-1]];
                    int j = top-2;
                    while(sta[j] != sta[top-1]) {
                        match[tmp] = sta[j];
                        swap(tmp, match[sta[j]]);
                        j--;
                    }
                    match[tmp] = sta[j];
                    match[sta[j]] = tmp;
                    break;
                }
            if(!update) {
                cnt++;
                if(cnt >= 3)break;
                random_shuffle(P, P + cnt_node);
            }
        }
    int ans = 0;
    for(int i = 0; i < cnt_node; i++) {
        int v = match[i];
        if(i < v)ans += G[i][v];
    }
    return ans;
}
```

3.13 2-SAT（二元可满足性问题）

- **定义:** 有 n 个布尔变量，要么是 0 要么是 1（实际问题中可对应 n 个物品取或不取），给出限制关系，比如 $a=1$ 必须先有 $b=1$，或者 a 和 b 不能取 1，问有没有一组合法方案。

- **解法:** 考虑拆点，每个变量拆成 2 个点 i 和 i', 分别表示 1 和 0。在图中加单向边 (i,j) 表示若选 i 必须选 j（即选 i 不选 j 是不行的）。

对于不同的限制，建边如下，最后建出边只要 $tarjan$ 判一下是不是存在同时包括了 i 和 i' 的环，如果有就是无解：

- $x=1 \rightarrow (x',x)$
- $x=0 \rightarrow (x,x')$
- $x \ \& \ y=1 \rightarrow (x',x),(y',y)$
- $x \ \& \ y=0 \rightarrow (y,x'),(x,y')$ 表示 x 和 y 不能都选
- $x \mid y=1 \rightarrow (x',y),(y',x)$ 表示 x 和 y 不能都不选
- $x \mid y=0 \rightarrow (x,x'),(y,y')$

- $x \oplus y = 1 \to (x, y'), (y, x'), (y', x), (x', y)$ 表示 x 和 y 有且只有选一个
- $x \oplus y = 0 \to (x, y), (x', y'), (y, x), (y', x')$ 表示 x 和 y 要么都选要么都不选
- $x \,\&\, \bar{y} = 0 \to (x, y'), (y, x')$
- $\bar{x} \,\&\, y = 0 \to (x', y'), (y, x)$

3.13.1 2-SAT 染色法（可以得到字典序最小的解）

```
struct twoSAT {
#define max_V 505
    struct Edge {
        int to, next;
    } E[max_V * max_V];
    int n;
    bool vis[max_V << 1];
    int head[max_V << 1], cnt;
    int S[max_V << 1], c;

    void init(int n) {
        this->n = n;
        for (int i = 0; i < n * 2; i++) {
            head[i] = -1;
            vis[i] = 0;
        }
        cnt = 0;
    }

    void addedge(int u, int v) {
        E[cnt].to = v;
        E[cnt].next = head[u];
        head[u] = cnt++;
    }

    bool dfs(int u) {
        if (vis[u ^ 1]) return 0;
        if (vis[u]) return 1;
        vis[u] = 1;
        S[c++] = u;
        for (int i = head[u]; ~i; i = E[i].next) {
            int v = E[i].to;
            if (!dfs(v)) return 0;
        }
        return 1;
    }

    bool solve() {
```

```
        for (int i = 0; i < n * 2; i += 2) {
            if (!vis[i] && !vis[i ^ 1]) {
                c = 0;
                if (!dfs(i)) {
                    while (c) vis[S[--c]] = 0;
                    if (!dfs(i + 1)) return 0;
                }
            }
        }
        return 1;
    }
#undef max_V
};
```

3.13.2 2-SAT 跑强连通分量缩点法（只能得到任意解）

```
/* 强连通缩点法、
 * 连边、跑 tarjan
 * 判可行性，即同一集合中的两个点是否同属一个强连通块
 * 缩点建新图，连反边、拓扑序，若当前点没有被访问过，则选择该点，不选择其另
 ↪ 外的点
 * 连边：
    a->b 即选 a 必选 b
    a、b 不能同时选：选了 a 就要选 b'，选了 b 就要选 a'
    a、b 必须同时选：选了 a 就要选 b，选了 b 就要选 a，选了 a'就要选 b'，
 ↪ 选了 b'就要选 a'
    a、b 必须选一个：选了 a 就要选 b'，选了 b 就要选 a'，选了 a'就要选 b，
 ↪ 选了 b'就要选 a
    a 必须选：a'->a。
 */
#include <bits/stdc++.h>
#define ll long long
#define inf 0x3f3f3f3f
#define pa pair<int,int>
#define mkp make_pair
#define N 2010
#define M 200100
#define p(x) ((x)*2)
#define q(x) ((x)*2-1)
using namespace std;
//==============================================
struct edge {
    int to, next;
} e[M];
int head[N], cnt;
inline void ins(int u, int v) {
```

```
    e[++cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}
//==============================================
int n;
int dfn[N], low[N], belong[N], cnt2, scc;
int zhan[N], top;
bool instack[N];
void dfs(int x) {
    dfn[x] = low[x] = ++scc;
    zhan[++top] = x;
    instack[x] = 1;
    for (int i = head[x]; i; i = e[i].next) {
        if (!dfn[e[i].to]) {
            dfs(e[i].to);
            low[x] = min(low[x], low[e[i].to]);
        } else if (instack[e[i].to])low[x] = min(low[x], dfn[e[i].to]);
    }
    if (low[x] == dfn[x]) {
        cnt2++;
        int tmp;
        do {
            tmp = zhan[top--];
            belong[tmp] = cnt2;
            instack[tmp] = 0;
        } while (x != tmp);
    }
}
inline bool solve() {
    scc = cnt2 = top = 0;
    memset(belong, 0, sizeof belong);
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(instack, 0, sizeof instack);
    for (int i = 1; i <= 2 * n; i++)if(!dfn[i])dfs(i);
    for (int i = 1; i <= 2 * n; i += 2)if (belong[i] == belong[i +
    ↪ 1])return 0;
    return 1;
}
int l, r;
struct pnt {
    int x, y;
} p[1010];
inline void build() {
    memset(head, 0, sizeof head);
```

```
    cnt = 0;
    for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
            if (p[i].x == p[j].x && abs(p[i].y - p[j].y) < 2 * r - 1) {
                ins(p(j), q(i));
                ins(p(i), q(j));
            } else if (p[i].y == p[j].y && abs(p[i].x - p[j].x) < 2 * r - 1) {
                ins(q(i), p(j));
                ins(q(j), p(i));
            }
}
int main() {
    scanf("%d%d%d", &l, &r, &n);
    for (int i = 1; i <= n; i++)scanf("%d%d", &p[i].x, &p[i].y);
    build();
    if(solve())puts("YES");
    else puts("NO");
}
```

3.14 最大流、最小割

3.14.1 Dinic

```
/* 点数 N, 边数 maxn, INF=0x3f3f3f3f, 注意图是单向边还是双向边
 * 小数流量记得加 eps
 */
struct Edge {
        int to, cap, next;
} E[maxn];
int head[N], pa[N], vis[N], cnt;
void init() {
        memset(head, -1, sizeof head);
        cnt = 0;
}
void addedge(int u, int v, int w) {
        E[cnt].to = v; E[cnt].cap = w; E[cnt].next = head[u]; head[u] = cnt++;
        E[cnt].to = u; E[cnt].cap = 0; E[cnt].next = head[v]; head[v] = cnt++; // 0 or w
}
bool bfs(int s, int t) {
        memset(vis, -1, sizeof vis);
        queue<int> q;
        vis[s] = 0;
        q.push(s);
        while (!q.empty()) {
```

```
			int u = q.front();
			q.pop();
			for (int i = head[u]; i != -1; i = E[i].next) {
				int v = E[i].to;
				if (E[i].cap && vis[v] == -1) {
					vis[v] = vis[u] + 1;
					q.push(v);
				}
			}
		}
		return vis[t] != -1;
}
int dfs(int u, int t, int flow) {
		if (u == t) return flow;
		for (int &i = pa[u]; i != -1; i = E[i].next) {
			int v = E[i].to;
			if (E[i].cap && vis[v] == vis[u] + 1) {
				int res = dfs(v, t, min(flow, E[i].cap));
				if (res) {
					E[i].cap -= res;
					E[i ^ 1].cap += res;
					return res;
				}
			}
		}
		return 0;
}
int Dinic(int s, int t) {
		int max_flow = 0;
		while (bfs(s, t)) {
			memcpy(pa, head, sizeof head);
			int res;
			do {
				res = dfs(s, t, INF);
				max_flow += res;
			} while (res);
		}
		return max_flow;
}
```

3.14.2 ISAP

```
/* 记得初始化 T+1 个点 */
struct Edge {
		int from, to, cap, flow;
};
class ISAP {
```

```
public:
#define max_V 202
        int n, m, s, t;
        vector<Edge> edges;
        vector<int> g[max_V];
        bool vis[max_V];
        int d[max_V], cur[max_V], p[max_V], num[max_V];

        void Addedge(int u, int v, int w) {
                edges.pb((Edge) {
                        u, v, w, 0
                });
                edges.pb((Edge) {
                        v, u, 0, 0
                });
                m = SZ(edges);
                g[u].pb(m - 2);
                g[v].pb(m - 1);
        }

        bool Bfs() {
                memset(vis, 0, sizeof vis);
                queue<int> q;
                q.push(t);
                vis[t] = 1;
                d[t] = 0;
                while (!q.empty()) {
                        int u = q.front(); q.pop();
                        int sz = SZ(g[u]);
                        rep(i, 0, sz) {
                                Edge &E = edges[g[u][i] ^ 1];
                                if (!vis[E.from] && E.cap > E.flow) {
                                        vis[E.from] = 1;
                                        d[E.from] = d[u] + 1;
                                        q.push(E.from);
                                }
                        }
                }
                return vis[s];
        }

        void ClearAll(int n) {
                this->n = n;
                rep(i, 0, n) g[i].clear();
                edges.clear();
        }
```

```

	void ClearFlow() {
		int sz = SZ(edges);
		rep(i, 0, sz) edges[i].flow = 0;
	}

	int Augment() {
		int x = t, a = INF;
		while (x != s) {
			Edge &E = edges[p[x]];
			a = min(a, E.cap - E.flow);
			x = edges[p[x]].from;
		}
		x = t;
		while (x != s) {
			edges[p[x]].flow += a;
			edges[p[x] ^ 1].flow -= a;
			x = edges[p[x]].from;
		}
		return a;
	}

	int MaxFlow(int s, int t) {
		this->s = s; this->t = t;
		int flow = 0;
		Bfs();
		memset(num, 0, sizeof num);
		rep(i, 0, n) num[d[i]]++;
		int x = s;
		memset(cur, 0, sizeof cur);
		while (d[s] < n) {
			if (x == t) {
				flow += Augment();
				x = s;
			}
			bool ok = 0;
			int sz = SZ(g[x]);
			rep(i, cur[x], sz) {
				Edge &E = edges[g[x][i]];
				if (E.cap > E.flow && d[x] == d[E.to] + 1) {
					ok = 1;
					p[E.to] = g[x][i];
					cur[x] = i;
					x = E.to;
					break;
```

```
                        }
                }
                if (!ok) {
                        int m = n - 1;
                        rep(i, 0, sz) {
                                Edge &E = edges[g[x][i]];
                                if (E.cap > E.flow)
                                        m = min(m, d[E.to]);
                        }
                        if (--num[d[x]] == 0) break; // gap
                        num[d[x] = m + 1]++;
                        cur[x] = 0;
                        if (x != s) x = edges[p[x]].from;
                }
        }
        return flow;
    }

    vector<int> Mincut() {
        Bfs();
        vector<int> ans;
        int sz = SZ(edges);
        rep(i, 0, sz) {
                Edge &E = edges[i];
                if (!vis[E.from] && vis[E.to] && E.cap > 0)
                        ans.pb(i);
        }
        return ans;
    }

    void Reduce() {
        int sz = SZ(edges);
        rep(i, 0, sz) {
                edges[i].cap -= edges[i].flow;
        }
    }

    void Print() {
        puts("Graph:");
        int sz = SZ(edges);
        rep(i, 0, sz) {
                printf("%d->%d, %d, %d\n", edges[i].from,
                 ↪ edges[i].to, edges[i].cap, edges[i].flow);
        }
    }
#undef max_V
```

```
};
```

3.15 费用流

3.15.1 最小费用最大流

```
struct Edge {
        int from, to, cap, flow, cost;
        Edge(int a, int b, int c, int d, int E) { from = a; to = b; cap =
        ↪ c; flow = d; cost = E; }
};
class MCMF {
public:
#define max_V 202
        int n, m, s, t;
        vector<Edge> edges;
        vector<int> g[max_V];
        bool vis[max_V];
        int d[max_V], p[max_V], a[max_V];

        void Init(int n) {
                this->n = n;
                rep(i, 0, n) g[i].clear();
                edges.clear();
        }

        void Addedge(int u, int v, int c, int w) {
                edges.pb(Edge(u, v, c, 0, w));
                edges.pb(Edge(v, u, 0, 0, -w));
                int m = SZ(edges);
                g[u].pb(m - 2);
                g[v].pb(m - 1);
        }

        bool Spfa(int s, int t, int &flow, int &cost) {
                fill(d, d + n, INF);
                memset(vis, 0, sizeof vis);
                d[s] = 0; vis[s] = 1; p[s] = 0; a[s] = INF;
                queue<int> q;
                q.push(s);
                while (!q.empty()) {
                        int u = q.front(); q.pop();
                        vis[u] = 0;
                        int sz = SZ(g[u]);
                        rep(i, 0, sz) {
                                Edge& E = edges[g[u][i]];
```

```
                if (E.cap > E.flow && d[E.to] > d[u] + E.cost) {
                    d[E.to] = d[u] + E.cost;
                    p[E.to] = g[u][i];
                    a[E.to] = min(a[u], E.cap - E.flow);
                    if (!vis[E.to]) {
                        vis[E.to] = 1;
                        q.push(E.to);
                    }
                }
            }
        }
        if (d[t] == INF) return 0;
        flow += a[t];
        cost += d[t] * a[t];
        int u = t;
        while (u != s) {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
            u = edges[p[u]].from;
        }
        return 1;
    }

    int MincostMaxflow(int s, int t) {
        int flow = 0, cost = 0;
        while (Spfa(s, t, flow, cost));
        return cost;
    }
#undef max_V
};
```

3.15.2 zkw 费用流

```
// 对于二分图类型的比较高效
const int MAXN = 100;
const int MAXM = 20000;
const int INF = 0x3f3f3f3f;
struct Edge {
    int to, next, cap, flow, cost;
    Edge(int _to = 0, int _next = 0, int _cap = 0, int _flow = 0, int _cost = 0): to(_to), next(_next), cap(_cap), flow(_flow), cost(_cost) {}
} edge[MAXM];
struct ZKW_MinCostMaxFlow {
    int head[MAXN], tot;
```

```
    int cur[MAXN];
    int dis[MAXN];
    bool vis[MAXN];
    int ss, tt, N; // 源点、汇点和点的总个数（编号是 0...N-1），不需要额外赋值，调用会直接赋值
    int min_cost, max_flow;
    void init() {
        tot = 0;
        memset(head, -1, sizeof(head));
    }
    void addedge(int u, int v, int cap, int cost) {
        edge[tot] = Edge(v, head[u], cap, 0, cost);
        head[u] = tot++;
        edge[tot] = Edge(u, head[v], 0, 0, -cost);
        head[v] = tot++;
    }
    int aug(int u, int flow) {
        if(u == tt)return flow;
        vis[u] = true;
        for(int i = cur[u]; i != -1; i = edge[i].next) {
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && !vis[v] && dis[u] == dis[v] + edge[i].cost) {
                int tmp = aug(v, min(flow, edge[i].cap-edge[i].flow));
                edge[i].flow += tmp;
                edge[i ^ 1].flow -= tmp;
                cur[u] = i;
                if(tmp)return tmp;
            }
        }
        return 0;
    }
    bool modify_label() {
        int d = INF;
        for(int u = 0; u < N; u++)
            if(vis[u])
                for(int i = head[u]; i != -1; i = edge[i].next) {
                    int v = edge[i].to;
                    if(edge[i].cap > edge[i].flow && !vis[v])
                        d = min(d, dis[v] + edge[i].cost-dis[u]);
                }
        if(d == INF)return false;
        for(int i = 0; i < N; i++)
            if(vis[i]) {
                vis[i] = false;
                dis[i] += d;
```

```
            }
        return true;
    }
    /* 直接调用获取最小费用和最大流
     * 输入: start-源点, end-汇点, n-点的总个数 (编号从 0 开始)
     * 返回值: pair<int,int> 第一个是最小费用, 第二个是最大流
     */
    pair<int, int> mincostmaxflow(int start, int end, int n) {
        ss = start, tt = end, N = n;
        min_cost = max_flow = 0;
        for(int i = 0; i < n; i++)dis[i] = 0;
        while(1) {
            for(int i = 0; i < n; i++)cur[i] = head[i];
            while(1) {
                for(int i = 0; i < n; i++)vis[i] = false;
                int tmp = aug(ss, INF);
                if(tmp == 0)break;
                max_flow += tmp;
                min_cost += tmp * dis[ss];
            }
            if(!modify_label())break;
        }
        return make_pair(min_cost, max_flow);
    }
} solve;
```

3.16 有上下界网络流

- **问题提出:** 给出由 n 点 m 边的有向网络图，每条边有容量下限 L 和上限 R，即流量必须在区间 $[L, R]$ 内。

3.16.1 无源汇可行流

没有源点和汇点，问是否存在满足限制的一道流（判定问题）

- **解法:** 将原来容量 $[L, R]$ 边拆成两条没有下限的边容量分别为 L 和 $R - L$，前者为必要弧。增加 x 和 y，并且令 $c[x, y] = \infty$，每个必要弧改成统一由 y 进入，由 x 统一流出，有可行流的充要条件是删掉边 (x, y) 后从 x 到 y 跑最大流能够满流。

3.16.2 有源汇可行流

- **解法:** 从汇点到源点建一条容量为 ∞ 边，问题转换为无源汇

3.16.3 有源汇最大流

- **解法:** 从汇点 t 到源点 s 建边成为无源汇的网络，二分最大流的值，作为从 t 到 s 的边的容量，判断如果存在可行流，说明合法，反之不合法。

3.17 二分图匹配最大流算法

在原图点集基础上增加源 s 和汇 t:

- 将每条二分图的边 $<u,v>\in E$ 替换为容量为 $c[u,v]=\infty$ 的有向边 $<u,v>\in E_N$
- 增加源 s 到二分图 X 部分的点 u 的有向边 $<s,u>\in E_N$，容量为该点权值（如果权为 1 则 $w=1$）$c[s,u]=w_u$
- 增加二分图 Y 部分的点 v 到汇 t 的有向边 $<v,t>\in E_N$，同样 $c(v,t)=w_v$

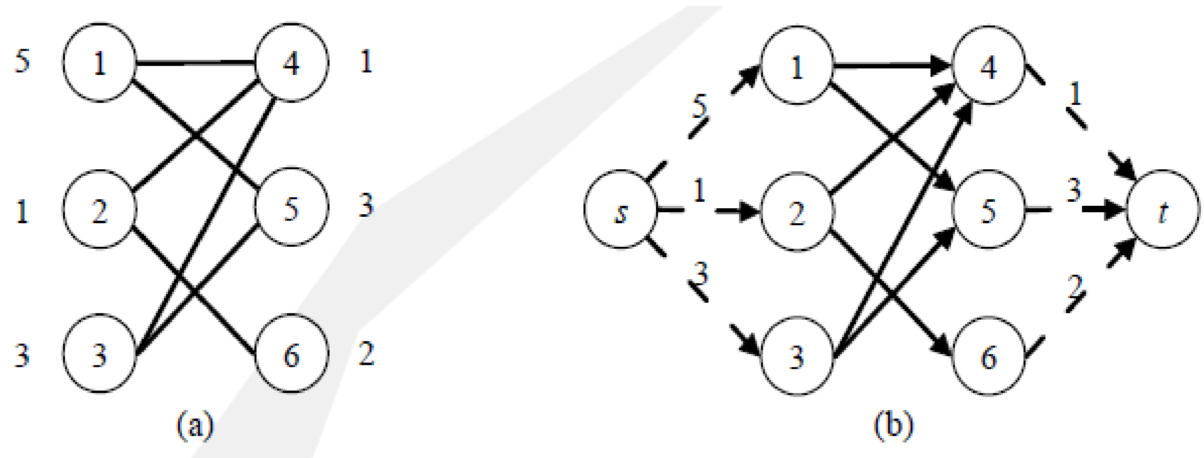

Figure 7: 二分图的模型转换

3.18 分数规划

一般形式: `Minimize` $\lambda=f(x)=\frac{a(x)}{b(x)}$ $(x\in S)$ $st.\ \forall x\in S,\ b(x)>0$，其中解向量 x 在解空间 S 内，$a(x)$ 与 $b(x)$ 都是**连续的实值函数**。

解法: 参数搜索法，对答案进行猜测，再验证其最优性，将优化问题转化为判定问题或其他的优化问题。假设 $\lambda^*=f(x^*)$ 为该规划的最优解，有:

$$\lambda^*=f(x^*)=\frac{a(x^*)}{b(x^*)}\Rightarrow\lambda^*\cdot b(x^*)=a(x^*)\Rightarrow a(x^*)-\lambda^*\cdot b(x^*)=0$$

由上面的形式构造一个新函数 $g(\lambda)$: $g(\lambda)=\texttt{min}\{a(x)-\lambda\cdot b(x)\}$, $x\in S$，那么有以下结论:

- $g(\lambda)$ 是一个严格递减函数，即对于 $\lambda_1<\lambda_2$，一定有 $g(\lambda_1)>g(\lambda_2)$
- $g(\lambda)=0$ 当且仅当 $\lambda=\lambda^*$

可以推出如下结论，发现可以对参数进行二分搜索。

$$\begin{cases}g(\lambda)=0\Leftrightarrow\lambda=\lambda^*\\g(\lambda)<0\Leftrightarrow\lambda>\lambda^*\\g(\lambda)>0\Leftrightarrow\lambda<\lambda^*\end{cases}$$

3.18.1 二分法

```
/* 例题：HDU - 6070：给你长度为 n 的序列，求某个区间 [l,r] 使得区间内的数
   字种类/区间长度最小，输出这个最小值
 * 解法：对于这种区间最优比率问题（分数规划问题）我们常规的解法是二分答案来
   求
 *     根据题目意思我们二分答案后可以转化成 size(l,r)/(r-l+1)<=mid   ->
   size(l,r)+l*mid <=(r+1)*mid
 *     这里我们可以枚举右边的 r ，然后用最值线段树来维护左边的最小值，
 *     我们 build 的时候将线段树初始化为 l*mid，然后对于枚举 i,a[i] 的贡献
   区间为 [pre[a[i]]+1,i]+1
 *     因为如果前面已经存在了 a[i]，该数就对之前的区间没有贡献了
 */
const int MX = 60005
int n;
struct segtree {
    int l, r, tag;
    double mn;
} t[MX * 4];

int num[MX];
int pre[MX];
int pos[MX];
inline void update(int k) {
    t[k].mn = min(t[k << 1].mn, t[k << 1 | 1].mn);
}
inline void pushdown(int k) {
    if(t[k].l == t[k].r || !t[k].tag)return;
    int tt = t[k].tag;
    t[k].tag = 0;
    t[k << 1].tag += tt;
    t[k << 1 | 1].tag += tt;
    t[k << 1].mn += tt;
    t[k << 1 | 1].mn += tt;
}
inline void buildtree(int now, int l, int r, double x) {
    t[now].l = l;
    t[now].r = r;
    t[now].tag = 0;
    if(l == r) {
        t[now].mn = x * l;
        return;
    }
    int mid = (l + r) >> 1;
    buildtree(now << 1, l, mid, x);
    buildtree(now << 1 | 1, mid + 1, r, x);
```

```
    update(now);
}
void change(int now, int x, int y) {
    pushdown(now);
    int l = t[now].l, r = t[now].r;
    if(l >= x && r <= y ) {
        t[now].mn++;
        t[now].tag++;
        return;
    }
    int mid = (l + r) >> 1;
    if(y <= mid) change(now << 1, x, y);
    else if(x > mid)change(now << 1 | 1, x, y);
    else change(now << 1, x, mid), change(now << 1 | 1, mid + 1, y);
    update(now);
}
inline double askmn(int now, int x, int y) {
    pushdown(now);
    int l = t[now].l, r = t[now].r;
    if(l >= x && r <= y)return t[now].mn;
    int mid = (l + r) >> 1;
    if(y <= mid) return askmn(now << 1, x, y);
    else if(x > mid) return askmn(now << 1 | 1, x, y);
    else return min(askmn(now << 1, x, mid), askmn(now << 1 | 1, mid + 1,
    ↪  y));
}
inline bool jud(double x) {
    buildtree(1, 1, n, x);
    for (int i = 1; i <= n ; i++) {
        change(1, pre[i] + 1, i);
        if(askmn(1, 1, i) <= x * (i + 1)) return true;
    }
    return false;
}
int main(int argc, char const *argv[]) {
    int t;
    scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i)pos[i] = 0;
        for (int i = 1; i <= n ; ++i) {
            scanf("%d", &num[i]);
            pre[i] = pos[num[i]];
            pos[num[i]] = i;
        }
        double l = 0, r = 1;
```

```
        double ans = 1;
        for (int i = 0; i < 15; ++i) {
            double mid = (l + r) / 2;
            if(jud(mid))r = mid, ans = mid;
            else l = mid;
        }
        printf("%f\n", ans );
    }
    return 0;
}
```

3.18.2 迭代法

```
#include <bits/stdc++.h>
using namespace std;
#define eps 1e-3
#define inf 1e17
const int maxn = 100100;
int tot = 0;
int stu[maxn], nxt[maxn << 1], vto[maxn << 1];
double w[maxn << 1];
int n, m;
int l[maxn], r[maxn];
double a[maxn], b[maxn];
double ll[maxn << 1], rr[maxn << 1];
void init() {
    memset(stu, -1, sizeof stu);
    tot = 0;
}
void added(int u, int v, double ww, double aa, double bb) {
    ll[tot] = aa, rr[tot] = bb;
    vto[tot] = v, w[tot] = ww, nxt[tot] = stu[u];
    stu[u] = tot++;
}
struct node {
    double u, d;
};
double dis[maxn];
bool vis[maxn];
int ped[maxn];
int pn[maxn];
node spfa() {
    for(int i = 1; i <= m + 1; i++) {
        dis[i] = inf;
        vis[i] = false;
    }
    deque<int> que;
```

```
    que.push_back(1);
    dis[1] = 0;
    while(!que.empty()) {
        int u = que.front();
        que.pop_front();
        vis[u] = false;
        for(int i = stu[u]; ~i; i = nxt[i]) {
            int v = vto[i];
            double ww = w[i];
            if(dis[u] + ww < dis[v]) {
                dis[v] = dis[u] + ww;
                ped[v] = i;
                pn[v] = u;
                if(vis[v])continue;
                vis[v] = true;
                if(!que.empty() && dis[v] >= dis[que.front()])
                 ↪  que.push_back(v);
                else que.push_front(v);
            }
        }
    }
    int u = m + 1;
    node ret;
    ret.u = ret.d = 0;
    while(u != 1) {
        ret.u += ll[ped[u]];
        ret.d += rr[ped[u]];
        u = pn[u];
    }
    return ret;
}
double check(double t) {
    init();
    for(int i = 1; i <= m; i++)added(i + 1, i, 0, 0, 0);
    double up = 0, dow = 0;
    for(int i = 1; i <= n; i++) {
        if(a[i] - b[i]*t < 0) {
            added(l[i], r[i] + 1, 0, 0, 0 );
            up += a[i];
            dow += b[i];
        } else {
            added(l[i], r[i] + 1, a[i] - b[i]*t, a[i], b[i]);
        }
    }
    node ans = spfa();
    return (ans.u + up) / (ans.d + dow);
```

```
}
void solve() {
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++)scanf("%d%d%lf%lf", &l[i], &r[i], &a[i],
    ↪   &b[i]);
    double pre = 0, t = 1e5;
    while(fabs(pre - t) > eps) {
        pre = t;
        t = check(t);
    }
    printf("%.3f\n", t);
}
int main(int argc, char const *argv[]) {
    int t;
    scanf("%d", &t);
    while(t--) solve();
    return 0;
}
```

3.19 最大权闭合图

- **问题提出:** 有向图中（不一定非得是 DAG）有边 (u, v)，如果选择 u 必须选择 v，这样选择出来的图叫做**闭合图**。闭合图某点有权值，则悬着出来的闭合图权值最大的问题叫最大权闭合图问题。

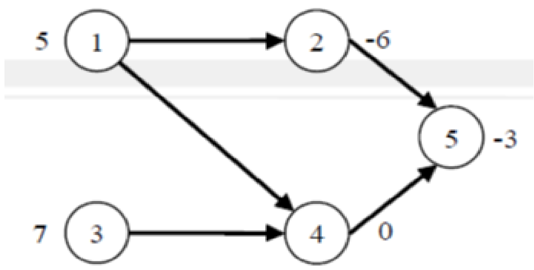

Figure 8: 闭合图的例子

- **解法:** 转化为网络流模型，构图如下，答案就是 $\sum_{v\in V^+} w[v] - c[s,t]$，其中 $c[s,t]$ 是新图的最小割。

  1. 添加超级源点 s 和汇点 t
  2. 原图中 (u, v) 容量设为 $c[u, v] = \infty$
  3. 如果某个点权 v 为正，则向源点 s 连边，容量为该点权值 $c[s, v] = w[v]$
  4. 如果某个点权 v 为负，则向汇点 t 连边，容量为该店权值相反数 $c[v, t] = -w[v]$

- **性质:**

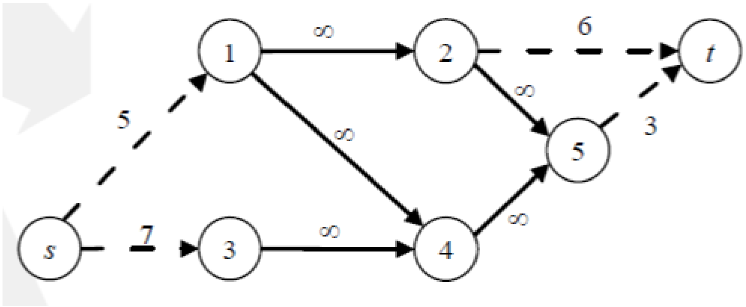

Figure 9: 闭合图的例子

1. 此网络流的割中的每条边，包含了 s 或者 t（称为简单割）
2. 以割为界，靠近 s 的一侧的点属于最大权闭合图的点，靠近 t 的点不属于 (因此找最大权闭合图可以从 s 按照残余流量进行 dfs)
3. 最大权闭合图的权 = 原图中权值为正的点的和 - 最大流（最小割）
4. 如果求最小权闭合图，则将原图中正负对换即可

3.20 最大密度子图

- **问题提出:** 无向图 $G =< V, E >$，n 个点 m 条边，选择一个子图 $G' =< V', E' >$ 使得 $\frac{|E'|}{|V'|}$ 最大。
- **分析:**
  1. 对于无向图 G 中，有一个密度为 D 的子图 G'，且在无向图 G 中不存在一个密度超过 $D + \frac{1}{n^2}$（$\frac{1}{n^2}$ 是精度）的子图，那么 G' 就是最大密度子图
  2. 设 $g = \frac{|E'|}{|V'|}$，要让 g 最大，就要让 $h(g) = |E'| - g \cdot |V'|$ 最大，因此参数搜索 g:
     - ∗ 当 $h(g) > 0$ 时，密度 g 偏小，存在更大密度的子图;
     - ∗ 当 $h(g) < 0$ 时，密度 g 偏大，不存在子图;
     - ∗ 当 $h(g) = 0$ 时，不考虑
- **解法:** 01 分数规划 + 网络流判定
  - 主代码

```
double l=0, r=U, eps = 1/(n*n);
while (r-l > eps) {
    g = (l+r)/2.0;
    h = search(g); // 参数搜索
    if (h > 0) l = g;
    else r = g;
} // 结束后 l 是密度
```

  - 构图过程
    1. 新图中增加超级源点 S 和超级汇点 T
    2. 原图中的边 $< u, v >$ 在新图中分别增加 $< u, v >$ 容量 $c[u, v] = 1.0$，和 $< v, u >$ 容量 $c[v, u] = 1$

   3. 对原图中每个点 i，在新图中增加边 $< S, i >$ 容量 $c[S,i] = U$，和边 $< i, T >$ 容量 $c[i,T] = U + 2 \cdot g - d_i$，其中 d_i 是原图中节点 i 的度数

 – 假设新图最大流 (最小割) $c[S,T] = flow$，那么本次搜索答案 $h(g) = \frac{U \cdot n - flow}{2}$

- **注意:**
  1. search 中其实没有返回 $h(g) < 0$ 的情况，但不影响运算
  2. 找最大密度子图 G' 结束搜索以后应该先跑一遍 search(l)，然后在残余流量中 dfs，靠近 S 一侧的点就是
  3. 若最后 $l = 0$，说明所有的点都是孤立的，任何一个点都是最大密度子图
- **扩展:**
  1. 边带权 w（非负）。这种情况实质上是两个点之间增加或减少一些边（可能是分数）

     解法: 令 $U = \sum w_i$，把精度变小一点搜索，其中每个边 $< u, v >$ 原来的的权（1.0）变成 $w[u,v]$，每个点度数 d_i 变成与点 i 相连的边权和。
  2. 边带权 w（非负），点带权 p（实数），使得 $\frac{\sum p_i + \sum w_i}{|V'|}$ 最大

     解法: 令 $U = 2 \cdot \sum\limits_{v \in V} |p_i| + \sum\limits_{e \in E} w_e$，把精度变小一点搜索，$< u, v >$ 原来的权（1.0）变成 $w[u,v]$，度数 d_i 变成与点 i 相连的所有边权和 $+2 \cdot p_i$。

3.21 最近公共祖先

3.21.1 DFS+ST 表在线算法

```
void RMQ() {
        for (int j = 1; (1 << j) <= n; j++)
                for (int i = 1; i <= n; i++)
                        if (~pa[i][j - 1])
                                pa[i][j] = pa[pa[i][j - 1]][j - 1];
}
int LCA(int x, int y) {
        if (deep[x] < deep[y]) swap(x, y);
        int i, j;
        for (i = 0; (1 << i) <= deep[x]; i++); i--;
        for (j = i; j >= 0; j--) // 把深度统一
                if (deep[x] - (1 << j) >= deep[y])
                         x = pa[x][j];
        if (x == y) return x;
        for (j = i; j >= 0; j--) //去找 LCA
                if (pa[x][j] != -1 && pa[x][j] != pa[y][j]) {
                        x = pa[x][j];
                        y = pa[y][j];
                }
        return pa[x][0];
}
```

```
/* 例题：Codeforces - 832D - 给 3 个点任选一个作为终点其他两个为起始点，求
↪  两条路径相交的最大值。
 */
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 5;
vector<int> g[maxn];
bool vis[maxn];
int pa[maxn][21], d[maxn];
void dfs(int u, int fa) {
    pa[u][0] = fa; // dfs 先预处理出该点向上爬的倍增结果
    for (int i = 1; i <= 20; i++) pa[u][i] = pa[pa[u][i-1]][i-1];
    for (auto v: g[u]) {
        if (v != fa) {
            d[v] = d[u] + 1;
            dfs(v, u);
        }
    }
}
int LCA(int u, int v) {
    if (d[u] < d[v]) swap(u, v);
    for (int i = 20; i >= 0; i--) {
        if (d[pa[u][i]] >= d[v]) u = pa[u][i];
    }
    if (u == v) return u;
    for (int i = 20; i >= 0; i--) {
        if (pa[u][i] != pa[v][i]) u = pa[u][i], v = pa[v][i];
    }
    return pa[u][0];
}
int dis(int u, int v) {
    int lca = LCA(u, v);
    return d[u] + d[v] - 2 * d[lca];
}

int main() {
    int n, q;
    scanf("%d%d", &n, &q);
    for (int i = 2; i <= n; i++) {
        int to;
        scanf("%d", &to);
        g[i].push_back(to);
        g[to].push_back(i);
    }
    memset(pa, 0, sizeof pa);
    memset(d, 0, sizeof d);
```

```
    dfs(1, 1);
    while (q--) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        int ab = dis(a, b), bc = dis(b, c), ac = dis(a, c);
        int len1 = (ab + bc - ac) / 2 + 1, len2 = (ab + ac - bc) / 2 + 1,
        ↪  len3 = (ac + bc - ab) / 2 + 1;
        printf("%d\n", max(len1, max(len2, len3)));
    }
    return 0;
}
```

3.21.2 离线 +tarjan 算法

```
// codeforces - 406D
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define rep(i,a,n) for (int i=a;i<n;i++)
typedef long long ll;
const int maxn = 1e5 + 10;
/* head */
vector<int> g[maxn];
struct Query {
        int to, next, id;
} q[maxn << 1];
int head[maxn], fa[maxn], ans[maxn], cnt;
bool vis[maxn];
void addedge(int u, int v, int id) {
        q[cnt].id = id, q[cnt].to = v, q[cnt].next = head[u], head[u] =
        ↪  cnt++;
        q[cnt].id = id, q[cnt].to = u, q[cnt].next = head[v], head[v] =
        ↪  cnt++;
}
int pa[maxn];
inline void init(int n) {
        cnt = 0;
        memset(head, -1, sizeof head);
        memset(vis, 0, sizeof vis);
        rep(i, 1, n + 1) pa[i] = i;
}
int find(int x) {
        if (pa[x] == x) return x;
        return pa[x] = find(pa[x]);
}
inline void merge(int u, int v) {
```

```
        int fu = find(u);
        int fv = find(v);
        if (fu != fv) pa[fu] = fv;
}
void dfs(int u) {
        fa[u] = u;
        vis[u] = 1;
        for (auto v: g[u]) {
                if (!vis[v]) {
                        dfs(v);
                        merge(u, v);
                        fa[find(v)] = u;
                }
        }
        for (int i = head[u]; ~i; i = q[i].next) {
                int v = q[i].to;
                if (vis[v]) ans[q[i].id] = fa[find(v)];
        }
}
struct Point {
        ll x, y;
        Point operator- (const Point& rhs) const {
                return { x - rhs.x, y - rhs.y };
        }
} p[maxn];
inline bool cross(const Point& A, const Point& B) { return A.x * B.y -
↪   A.y * B.x > 0; }
int s[maxn], top;
inline void solve() {
        int n, m;
        scanf("%d", &n);
        rep(i, 1, n + 1) scanf("%lld%lld", &p[i].x, &p[i].y);
        init(n);
        top = 0;
        for (int i = n; i >= 1; i--) {
                while (top > 1 && cross(p[s[top]] - p[i], p[s[top - 1]] -
                ↪   p[i])) top--;
                s[++top] = i;
                if (top > 1) g[s[top - 1]].pb(s[top]);
        }
        scanf("%d", &m);
        rep(i, 1, m + 1) {
                int u, v;
                scanf("%d%d", &u, &v);
                addedge(u, v, i);
        }
```

```
        dfs(n); // 应该要找入度为 0 的点作为根 dfs 下去，这题建树特点可以看
        ↪ 出 n 是根，所以省去了找根的步骤
        rep(i, 1, m + 1) {
                if (i == 1) printf("%d", ans[i]);
                else printf(" %d", ans[i]);
        }
        puts("");
}
int main() {
        solve();
        return 0;
}
```

3.22 平面图

- **定义:** 能平面嵌入 (能画在平面上并且边不交) 的无向图。(平面图 $G =< V, E, F >$ 同无向图 $G =< V, E >$)
- **面** F**:** 由 G 的一条或多条边所框出来的区域（内外都可以算）如果不含其它 E 和 V 的元素，就是一个面。
- **平面图判定:** 不能存在 5 个点能两两联通，或者两组各 3 个点（一共 6 个）能不同组两两联通（就是不存在完全图 K_5 或完全二分图 $K_{3,3}$，其中“边”应该认为“联通”或“传递闭包后有边”）。
- **平面图欧拉定理:** $v - e + f = c + 1$，其中 v 是点数，e 是边数，f 是平面数（外面那个无限大的也算一个），c 是联通块个数（图中不一定只有一个联通块），如果只有一块则 $c = 1, v - e + f = 2$。

3.23 欧拉图

- **欧拉回路:** 每条边只经过一次，而且回到起点
- **欧拉路径:** 每条边只经过一次，不要求回到起点
- **欧拉回路判断:**
  - **无向图:** 连通（不考虑度为 0 的点），每个顶点度数都为偶数。
  - **有向图:** 基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度。
  - **混合图（有无向边和有向边）:** 首先是基图连通（不考虑度为 0 的点），然后需要借助网络流判定。

首先给原图中的每条无向边随便指定一个方向（称为初始定向），将原图改为有向图 G’，然后的任务就是改变 G’ 中某些边的方向（当然是无向边转化来的，原混合图中的有向边不能动）使其满足每个点的入度等于出度。

设 $D[i]$ 为 G’ 中 (点 i 的出度 – 点 i 的入度）。可以发现，在改变 G’ 中边的方向的过程中，任何点的 D 值的奇偶性都不会发生改变（设将边 $< i, j >$ 改为

$< j, i >$，则 i 入度加 1 出度减 1，j 入度减 1 出度加 1，两者之差加 2 或减 2，奇偶性不变）而最终要求的是每个点的入度等于出度，即每个点的 D 值都为 0，是偶数，故可得：若初始定向得到的 G' 中任意一个点的 D 值是奇数，那么原图中一定不存在欧拉环！

- **欧拉路径的判断:**
  - **无向图:** 连通（不考虑度为 0 的点），每个顶点度数都为偶数或者仅有两个点的度数为偶数。
  - **有向图:** 基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度或者有且仅有一个点的出度比入度多 1，有且仅有一个点的出度比入度少 1，其余出度等于入度。
  - **混合图（有无向边和有向边）:** 如果存在欧拉回路，一点存在欧拉路径了。否则如果有且仅有两个点的（出度 - 入度）是奇数，那么给这个两个点加边，判断是否存在欧拉回路。

3.23.1 有向图

```
/* 例题：POJ 2337 - 给出 n 个小写字母组成的单词，要求将 n 个单词连接起来，
↪ 使得前一个单词的最后一个字母
和后一个单词的第一个字母相同。输出字典序最小的解。
 */
struct Edge {
    int to, next;
    int index;
    bool flag;
} edge[2010];
int head[30], tot;
void init() {
    tot = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int index) {
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].index = index;
    edge[tot].flag = false;
    head[u] = tot++;
}
string str[1010];
int in[30], out[30];
int cnt;
int ans[1010];
void dfs(int u) {
    for(int i = head[u] ; i != -1; i = edge[i].next)
        if(!edge[i].flag) {
            edge[i].flag = true;
```

```
            dfs(edge[i].to);
            ans[cnt++] = edge[i].index;
        }
}
int main() {
    int T, n;
    scanf("%d", &T);
    while(T--) {
        scanf("%d", &n);
        for(int i = 0; i < n; i++) cin >> str[i];
        sort(str, str + n); //要输出字典序最小的解, 先按照字典序排序
        init();
        memset(in, 0, sizeof(in));
        memset(out, 0, sizeof(out));
        int start = 100;
        for(int i = n-1; i >= 0; i--) { //字典序大的先加入
            int u = str[i][0] - 'a';
            int v = str[i][str[i].length() - 1] - 'a';
            addedge(u, v, i);
            out[u]++;
            in[v]++;
            if(u < start)start = u;
            if(v < start)start = v;
        }
        int cc1 = 0, cc2 = 0;
        for(int i = 0; i < 26; i++) {
            if(out[i] - in[i] == 1) {
                cc1++;
                start = i;//如果有一个出度比入度大 1 的点, 就从这个点出发,
                ↪ 否则从最小的点出发
            } else if(out[i] - in[i] == -1) cc2++;
            else if(out[i] - in[i] != 0) cc1 = 3;
        }
        if(! ( (cc1 == 0 && cc2 == 0) || (cc1 == 1 && cc2 == 1) )) {
            printf("***\n");
            continue;
        }
        cnt = 0;
        dfs(start);
        if(cnt != n) { //判断是否连通
            printf("***\n");
            continue;
        }
        for(int i = cnt-1; i >= 0; i--) {
            cout << str[ans[i]];
            if(i > 0)printf(".");
```

```
            else printf("\n");
        }
    }
    return 0;
}
```

3.23.2 无向图

```
// SGU 101
struct Edge {
    int to, next;
    int index;
    int dir;
    bool flag;
} edge[220];
int head[10], tot;
void init() {
    memset(head, -1, sizeof(head));
    tot = 0;
}
void addedge(int u, int v, int index) {
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].index = index;
    edge[tot].dir = 0;
    edge[tot].flag = false;
    head[u] = tot++;
    edge[tot].to = u;
    edge[tot].next = head[v];
    edge[tot].index = index;
    edge[tot].dir = 1;
    edge[tot].flag = false;
    head[v] = tot++;
}
int du[10];
vector<int>ans;
void dfs(int u) {
    for(int i = head[u]; i != -1; i = edge[i].next)
        if(!edge[i].flag ) {
            edge[i].flag = true;
            edge[i ^ 1].flag = true;
            dfs(edge[i].to);
            ans.push_back(i);
        }
}
int main() {
    int n;
```

```
    while(scanf("%d", &n) == 1) {
        init();
        int u, v;
        memset(du, 0, sizeof(du));
        for(int i = 1; i <= n; i++) {
            scanf("%d%d", &u, &v);
            addedge(u, v, i);
            du[u]++;
            du[v]++;
        }
        int s = -1;
        int cnt = 0;
        for(int i = 0; i <= 6; i++) {
            if(du[i] & 1) {
                cnt++;
                s = i;
            }
            if(du[i] > 0 && s == -1)
                s = i;
        }
        bool ff = true;
        if(cnt != 0 && cnt != 2) {
            printf("No solution\n");
            continue;
        }
        ans.clear();
        dfs(s);
        if(ans.size() != n) {
            printf("No solution\n");
            continue;
        }
        for(int i = 0; i < ans.size(); i++) {
            printf("%d ", edge[ans[i]].index);
            if(edge[ans[i]].dir == 0)printf("-\n");
            else printf("+\n");
        }
    }
    return 0;
}
```

3.23.3 混合图

```
// 最大流部分 SAP
const int MAXN = 210;
const int MAXM = 20100;
const int INF = 0x3f3f3f3f;
struct Edge {
```

```
    int to, next, cap, flow;
} edge[MAXM];
int tol;
int head[MAXN];
int gap[MAXN], dep[MAXN], pre[MAXN], cur[MAXN];
void init() {
    tol = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int w, int rw = 0) {
    edge[tol].to = v;
    edge[tol].cap = w;
    edge[tol].next = head[u];
    edge[tol].flow = 0;
    head[u] = tol++;
    edge[tol].to = u;
    edge[tol].cap = rw;
    edge[tol].next = head[v];
    edge[tol].flow = 0;
    head[v] = tol++;
}
int sap(int start, int end, int N) {
    memset(gap, 0, sizeof(gap));
    memset(dep, 0, sizeof(dep));
    memcpy(cur, head, sizeof(head));
    int u = start;
    pre[u] = -1;
    gap[0] = N;
    int ans = 0;
    while(dep[start] < N) {
        if(u == end) {
            int Min = INF;
            for(int i = pre[u]; i != -1; i = pre[edge[i ^ 1].to])
                if(Min > edge[i].cap - edge[i].flow)
                    Min = edge[i].cap - edge[i].flow;
            for(int i = pre[u]; i != -1; i = pre[edge[i ^ 1].to]) {
                edge[i].flow += Min;
                edge[i ^ 1].flow -= Min;
            }
            u = start;
            ans += Min;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u]; i != -1; i = edge[i].next) {
```

```
                v = edge[i].to;
                if(edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u]) {
                    flag = true;
                    cur[u] = pre[v] = i;
                    break;
                }
            }
            if(flag) {
                u = v;
                continue;
            }
            int Min = N;
            for(int i = head[u]; i != -1; i = edge[i].next)
                if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min) {
                    Min = dep[edge[i].to];
                    cur[u] = i;
                }
            gap[dep[u]]--;
            if(!gap[dep[u]])return ans;
            dep[u] = Min + 1;
            gap[dep[u]]++;
            if(u != start)u = edge[pre[u] ^ 1].to;
        }
        return ans;
}
//the end of 最大流部分
int in[MAXN], out[MAXN]; //每个点的出度和入度
int main() {
    int T;
    int n, m;
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &m);
        init();
        int u, v, w;
        memset(in, 0, sizeof(in));
        memset(out, 0, sizeof(out));
        while(m--) {
            scanf("%d%d%d", &u, &v, &w);
            out[u]++;
            in[v]++;
            if(w == 0)//双向
                addedge(u, v, 1);
        }
        bool flag = true;
        for(int i = 1; i <= n; i++) {
```

```
            if(out[i] - in[i] > 0) addedge(0, i, (out[i] - in[i]) / 2);
            else if(in[i] - out[i] > 0) addedge(i, n + 1, (in[i] - out[i]) / 2);
            if((out[i] - in[i]) & 1) flag = false;
        }
        if(!flag) {
            printf("impossible\n");
            continue;
        }
        sap(0, n + 1, n + 2);
        for(int i = head[0]; i != -1; i = edge[i].next)
            if(edge[i].cap > 0 && edge[i].cap > edge[i].flow) {
                flag = false;
                break;
            }
        if(flag) printf("possible\n");
        else printf("impossible\n");
    }
    return 0;
}
```

3.24 最小树形图

```
//求具有 V 个点, 以 root 为根节点的图 map 的最小树形图
int zhuliu(int root, int V, int map[MAXV + 7][MAXV + 7]) {
    bool visited[MAXV + 7];
    bool flag[MAXV + 7];//缩点标记为 true, 否则仍然存在
    int pre[MAXV + 7];//点 i 的父节点为 pre[i]
    int sum = 0;//最小树形图的权值
    int i, j, k;
    for(i = 0; i <= V; i++) flag[i] = false, map[i][i] = INF;
    pre[root] = root;
    while(true) {
        for(i = 1; i <= V; i++) { //求最短弧集合 E0
            if(flag[i] || i == root) continue;
            pre[i] = i;
            for(j = 1; j <= V; j++)
                if(!flag[j] && map[j][i] < map[pre[i]][i])
                    pre[i] = j;
            if(pre[i] == i) return -1;
        }
        for(i = 1; i <= V; i++) { //检查 E0
            if(flag[i] || i == root) continue;
            for(j = 1; j <= V; j++) visited[j] = false;
            visited[root] = true;
            a
```

```
            j = i;//从当前点开始找环
            do {
                visited[j] = true;
                j = pre[j];
            } while(!visited[j]);
            if(j == root)continue;//没找到环
            i = j;//收缩 G 中的有向环
            do { //将整个环的取值保存，累计计入原图的最小树形图
                sum += map[pre[j]][j];
                j = pre[j];
            } while(j != i);
            j = i;
            do { //对于环上的点有关的边，修改其权值
                for(k = 1; k <= V; k++)
                    if(!flag[k] && map[k][j] < INF && k != pre[j])
                        map[k][j] -= map[pre[j]][j];
                j = pre[j];
            } while(j != i);
            for(j = 1; j <= V; j++) { //缩点，将整个环缩成 i 号点，所有与环上的点有关的边转移到点 i
                if(j == i) continue;
                for(k = pre[i]; k != i; k = pre[k]) {
                    if(map[k][j] < map[i][j]) map[i][j] = map[k][j];
                    if(map[j][k] < map[j][i]) map[j][i] = map[j][k];
                }
            }
            for(j = pre[i]; j != i; j = pre[j]) flag[j] = true;//标记环上其他点为被缩掉
            break;//当前环缩点结束，形成新的图 G'，跳出继续求 G'的最小树形图
        }
        if(i > V) { //如果所有的点都被检查且没有环存在，现在的最短弧集合 E0 就是最小树形图. 累计计入 sum，算法结束
            for(i = 1; i <= V; i++)
                if(!flag[i] && i != root) sum += map[pre[i]][i];
            break;
        }
    }
    return sum;
}
```

3.25 曼哈顿最小生成树

```
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Point {
```

```
    int x, y, id;
} p[MAXN];
bool cmp(Point a, Point b) {
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}
// 树状数组, 找 y-x 大于当前的, 但是 y+x 最小的
struct BIT {
    int min_val, pos;
    void init() {
        min_val = INF;
        pos = -1;
    }
} bit[MAXN];
// 所有有效边
struct Edge {
    int u, v, d;
} edge[MAXN << 2];
bool cmpedge(Edge a, Edge b) {
    return a.d < b.d;
}
int tot;
int n;
int F[MAXN];
int find(int x) {
    if(F[x] == -1) return x;
    else return F[x] = find(F[x]);
}
void addedge(int u, int v, int d) {
    edge[tot].u = u;
    edge[tot].v = v;
    edge[tot++].d = d;
}
int lowbit(int x) {
    return x & (-x);
}
void update(int i, int val, int pos) {
    while(i > 0) {
        if(val < bit[i].min_val) {
            bit[i].min_val = val;
            bit[i].pos = pos;
        }
        i -= lowbit(i);
    }
}
// 查询 [i,m] 的最小值位置
```

```
int ask(int i, int m) {
    int min_val = INF, pos = -1;
    while(i <= m) {
        if(bit[i].min_val < min_val) {
            min_val = bit[i].min_val;
            pos = bit[i].pos;
        }
        i += lowbit(i);
    }
    return pos;
}
int dist(Point a, Point b) {
    return abs(a.x - b.x) + abs(a.y - b.y);
}
void Manhattan_minimum_spanning_tree(int n, Point p[]) {
    int a[MAXN], b[MAXN];
    tot = 0;
    for(int dir = 0; dir < 4; dir++) {
        // 4 种坐标变换
        if(dir == 1 || dir == 3) {
            for(int i = 0; i < n; i++)
                swap(p[i].x, p[i].y);
        } else if(dir == 2) {
            for(int i = 0; i < n; i++)
                p[i].x = -p[i].x;
        }
        sort(p, p + n, cmp);
        for(int i = 0; i < n; i++)
            a[i] = b[i] = p[i].y - p[i].x;
        sort(b, b + n);
        int m = unique(b, b + n) - b;
        for(int i = 1; i <= m; i++)
            bit[i].init();
        for(int i = n-1 ; i >= 0; i--) {
            int pos = lower_bound(b, b + m, a[i]) - b + 1;
            int ans = ask(pos, m);
            if(ans != -1)
                addedge(p[i].id, p[ans].id, dist(p[i], p[ans]));
            update(pos, p[i].x + p[i].y, i);
        }
    }
}
int solve(int k) {
    Manhattan_minimum_spanning_tree(n, p);
    memset(F, -1, sizeof(F));
    sort(edge, edge + tot, cmpedge);
```

```
    for(int i = 0; i < tot; i++) {
        int u = edge[i].u;
        int v = edge[i].v;
        int t1 = find(u), t2 = find(v);
        if(t1 != t2) {
            F[t1] = t2;
            k--;
            if(k == 0)return edge[i].d;
        }
    }
}
int main() {
    int k;
    while(scanf("%d%d", &n, &k) == 2 && n) {
        for(int i = 0; i < n; i++) {
            scanf("%d%d", &p[i].x, &p[i].y);
            p[i].id = i;
        }
        printf("%d\n", solve(n-k));
    }
    return 0;
}
```

3.26 生成树计数

矩阵树定理 (Matrix-Tree) - 基尔霍夫矩阵（Kirchhoff）定理：

- G 的度数矩阵 $D[G]$ 是一个 $n \times n$ 的矩阵
  - 当 $i \neq j$ 时，$d[i,j] = 0$
  - 当 $i = j$ 时，$d[i,i] = degree[i]$
- G 的邻接矩阵 $A[G]$ 也是一个 $n \times n$ 的矩阵
  - 如果 u 和 v 有直接边相连，$a[u,v] = 1$
  - 如果 u 和 v 不直接相连，$a[u,v] = 0$

定义 G 的基尔霍夫矩阵 $C[G] = D[G] - A[G]$，那么矩阵树定理可以描述为：G 的所有不同的生成树的个数等于其基尔霍夫矩阵 $C[G]$ 的任何一个 $(n-1)$ 阶主子式的行列式的绝对值。（**矩阵的** $(n-1)$ **阶主子式:** 矩阵 M 删除第 x 行和列后的新的 $(n-1)$ 阶矩阵）

```
/* 例题 1：HDU 4305 - 求生成树计数部分代码，计数对 10007 取模
 */
const int MOD = 10007;
int INV[MOD];
//求 ax = 1( mod m) 的 x 值，就是逆元 (0<a<m)
long long inv(long long a, long long m) {
```

```
    if(a == 1)return 1;
    return inv(m % a, m) * (m-m / a) % m;
}
struct Matrix {
    int mat[330][330];
    void init() {
        memset(mat, 0, sizeof(mat));
    }
    //求行列式的值模上，需要使用逆元 MOD
    int det(int n) {
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                mat[i][j] = (mat[i][j] % MOD + MOD) % MOD;
        int res = 1;
        for(int i = 0; i < n; i++) {
            for(int j = i; j < n; j++)
                if(mat[j][i] != 0) {
                    for(int k = i; k < n; k++)
                        swap(mat[i][k], mat[j][k]);
                    if(i != j)
                        res = (-res + MOD) % MOD;
                    break;
                }
            if(mat[i][i] == 0) {
                res = -1;//不存在也就是行列式值为 (0)
                break;
            }
            for(int j = i + 1; j < n; j++) {
                //int mut = (mat[j][i]*INV[mat[i][i]])%MOD 打表逆元;
                int mut = (mat[j][i] * inv(mat[i][i], MOD)) % MOD;
                for(int k = i; k < n; k++)
                    mat[j][k] = (mat[j][k]-(mat[i][k] * mut) % MOD + MOD)
                     ↪ % MOD;
            }
            res = (res * mat[i][i]) % MOD;
        }
        return res;
    }
};
// 主函数里面
Matrix ret;
ret.init();
for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        if(i != j && g[i][j]) {
            ret.mat[i][j] = -1;
```

```
                ret.mat[i][i]++;
        }
printf("%d\n", ret.det(n-1));

/* 例题 2：SPOJ 104 - 求生成树个数，不取模
 */
 const double eps = 1e-8;
const int MAXN = 110;
int sgn(double x) {
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
double b[MAXN][MAXN];
double det(double a[][MAXN], int n) {
    int i, j, k, sign = 0;
    double ret = 1;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            b[i][j] = a[i][j];
    for(i = 0; i < n; i++) {
        if(sgn(b[i][i]) == 0) {
            for(j = i + 1; j < n; j++)
                if(sgn(b[j][i]) != 0)
                    break;
            if(j == n)return 0;
            for(k = i; k < n; k++)
                swap(b[i][k], b[j][k]);
            sign++;
        }
        ret *= b[i][i];
        for(k = i + 1; k < n; k++)
            b[i][k] /= b[i][i];
        for(j = i + 1; j < n; j++)
            for(k = i + 1; k < n; k++)
                b[j][k] - = b[j][i] * b[i][k];
    }
    if(sign & 1)ret = -ret;
    return ret;
}
double a[MAXN][MAXN];
int g[MAXN][MAXN];
int main() {
    int T;
    int n, m;
    int u, v;
```

```
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &m);
        memset(g, 0, sizeof(g));
        while(m--) {
            scanf("%d%d", &u, &v);
            u--;
            v--;
            g[u][v] = g[v][u] = 1;
        }
        memset(a, 0, sizeof(a));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(i != j && g[i][j]) {
                    a[i][i]++;
                    a[i][j] = -1;
                }
        double ans = det(a, n-1);
        printf("%.0lf\n", ans);
    }
    return 0;
}
```

3.27 树分治

3.27.1 点分治

```
// HDU 5314 点权路径极差小于 D 的对数
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
struct Edge { int to, next; } e[maxn << 1];
int head[maxn], cnt;
inline void init() { memset(head, -1, sizeof head); cnt = 0; }
inline void addedge(int u, int v) {
    e[cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
int son[maxn], f[maxn], vis[maxn], sum, rt;
int p[maxn], n, d, sz;
ll ans;
pair<int, int> v[maxn];
void dfs1(int u, int fa) { // for son and root
    son[u] = 1, f[u] = 0;
    for (int i = head[u]; ~i; i = e[i].next) {
```

```
        int v = e[i].to;
        if (v != fa && !vis[v]) {
            dfs1(v, u);
            son[u] += son[v];
            f[u] = max(f[u], son[v]);
        }
    }
    f[u] = max(f[u], sum - son[u]);
    if (f[u] < f[rt]) rt = u;
}
void dfs2(int u, int fa, int mi, int mx) { // calculate
    mi = min(mi, p[u]);
    mx = max(mx, p[u]);
    if (mi + d >= mx) v[++sz] = {mi, mx};
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to;
        if (!vis[v] && v != fa) dfs2(v, u, mi, mx);
    }
}
ll calc(int u, int mi, int mx) {
    sz = 0;
    dfs2(u, 0, mi, mx);
    sort(v +1, v + sz + 1);
    ll res = 0;
    for (int i = sz; i >= 1; i--) {
        int id = lower_bound(v + 1, v + i, make_pair(v[i].second - d, 0))
        ↪  - v;
        res += (i - id);
    }
    return res;
}
void dfs(int u) {
    ans += calc(u, p[u], p[u]);
    vis[u] = 1;
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to;
        if (!vis[v]) {
            ans -= calc(v, p[u], p[u]);
            rt = 0;
            f[0] = sum = son[v];
            dfs1(v, 0);
            dfs(rt);
        }
    }
}
int main() {
```

```
    int t;
    scanf("%d", &t);
    while (t--) {
        init();
        scanf("%d%d", &n, &d);
        for (int i = 1; i <= n; i++) scanf("%d", &p[i]);
        for (int i = 1; i < n; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            addedge(u, v);
            addedge(v, u);
        }
        ans = 0; rt = 0;
        memset(vis, 0, sizeof vis);
        f[0] = sum = n;
        dfs1(1, 0);
        dfs(rt);
        printf("%lld\n", ans * 2);
    }
    return 0;
}
// BZOJ 2152 mod 3 意义下路径点对计数
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e4 + 5;
struct Edge { int to, val, next; } e[maxn << 1];
int head[maxn], cnt;
inline void init() { memset(head, -1, sizeof head); cnt = 0; }
inline void addedge(int u, int v, int w) { e[cnt].to = v; e[cnt].next =
    head[u]; e[cnt].val = w; head[u] = cnt++; }
int son[maxn], f[maxn], d[maxn], t[5], sum, rt, ans;
bool vis[maxn];
void dfs1(int u, int fa) {  // root
    son[u] = 1, f[u] = 0;
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to;
        if (!vis[v] && v != fa) {
            dfs1(v, u);
            son[u] += son[v];
            f[u] = max(f[u], son[v]);
        }
    }
    f[u] = max(f[u], sum - son[u]);
    if (f[u] < f[rt]) rt = u;
}
void dfs2(int u, int fa) {  // deep
```

```
    t[d[u]]++;
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to, w = e[i].val;
        if (!vis[v] && v != fa) {
            d[v] = (d[u] + w) % 3;
            dfs2(v, u);
        }
    }
}
int calc(int u, int w) {
    t[0] = t[1] = t[2] = 0;
    d[u] = w;
    dfs2(u, 0);
    return t[1] * t[2] * 2 + t[0] * t[0];
}
void dfs3(int u) {
    ans += calc(u, 0);
    vis[u] = 1;
    for (int i = head[u]; ~i; i = e[i].next) {
        int  v = e[i].to, w = e[i].val;
        if (!vis[v]) {
            ans -= calc(v, w);
            rt = 0;
            sum = son[v];
            dfs1(v, 0);
            dfs3(rt);
        }
    }
}
int main() {
    int n;
    scanf("%d", &n);
    init();
    for (int i = 1; i < n; i++) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        addedge(u, v, w % 3);
        addedge(v, u, w % 3);
    }
    f[0] = sum = n;
    dfs1(1, 0);
    ans = 0;
    dfs3(rt);
    int g = __gcd(ans, n * n);
    printf("%d/%d\n", ans / g, n * n / g);
    return 0;
```

```
}
```

3.27.2 边分治

```
/* 例题：HDU 5039 一颗树, 每条边的属性为 0 或 1, 求有多少条路径经过奇数条属
 ↪ 性为 1 的边。一种是查询操作, 一种是修改边的属性。
 */
const int MAXN = 30010;
const int INF = 0x3f3f3f3f;
struct Edge {
    int to, next;
    int f;
} edge[MAXN * 2];
int head[MAXN], tot;
void init() {
    tot = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int f) {
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].f = f;
    head[u] = tot++;
}
long long ans;
int num0[MAXN], num1[MAXN];
long long tnum[MAXN];
struct Node {
    int l0, l1;
    int r0, r1;
    int cc;
    long long sum;
    Node gao(int u) {
        l0 = r0 = num0[u];
        l1 = r1 = num1[u];
        sum = tnum[u];
        cc = 0;
        return *this;
    }
};
int pos[MAXN];
int val[MAXN];
int fa[MAXN];
int cnt[MAXN];
int col[MAXN];
int link[MAXN];
```

```
int CHANGEU;
struct chain {
    vector<int>uu;
    vector<Node>nde;
    int n;
    void init() {
        n = uu.size();
        nde.resize(n << 2);
        for(int i = 0; i < n; i++)pos[uu[i]] = i;
        build(0, n-1, 1);
    }
    void up(int l, int r, int p) {
        int mid = (l + r) >> 1;
        nde[p].cc = nde[p << 1].cc ^ nde[(p << 1) | 1].cc ^ val[uu[mid]];
        nde[p].l0 = nde[p << 1].l0;
        nde[p].l1 = nde[p << 1].l1;
        if(nde[p << 1].cc ^ val[uu[mid]]) {
            nde[p].l0 += nde[(p << 1) | 1].l1;
            nde[p].l1 += nde[(p << 1) | 1].l0;
        } else {
            nde[p].l0 += nde[(p << 1) | 1].l0;
            nde[p].l1 += nde[(p << 1) | 1].l1;
        }
        nde[p].r0 = nde[(p << 1) | 1].r0;
        nde[p].r1 = nde[(p << 1) | 1].r1;
        if(nde[(p << 1) | 1].cc ^ val[uu[mid]]) {
            nde[p].r0 += nde[p << 1].r1;
            nde[p].r1 += nde[p << 1].r0;
        } else {
            nde[p].r0 += nde[p << 1].r0;
            nde[p].r1 += nde[p << 1].r1;
        }
        if(val[uu[mid]] == 0) {
            nde[p].sum = nde[p << 1].sum + nde[(p << 1) | 1].sum +
                         (long long)nde[p << 1].r0 * nde[(p << 1) | 1].l1 +
                         (long long)nde[p << 1].r1 * nde[(p << 1) | 1].l0;
        } else {
            nde[p].sum = nde[p << 1].sum + nde[(p << 1) | 1].sum +
                         (long long)nde[p << 1].r0 * nde[(p << 1) | 1].l0 +
                         (long long)nde[p << 1].r1 * nde[(p << 1) | 1].l1;
        }
    }
```

```
    void build(int l, int r, int p) {
        if(l == r) {
            nde[p].gao(uu[l]);
            return;
        }
        int mid = (l + r) / 2;
        build(l, mid, p << 1);
        build(mid + 1, r, (p << 1) | 1);
        up(l, r, p);
    }
    void update(int k, int l, int r, int p) {
        if(l == r) {
            nde[p].gao(uu[k]);
            return;
        }
        int mid = (l + r) / 2;
        if(k <= mid)update(k, l, mid, p << 1);
        else update(k, mid + 1, r, (p << 1) | 1);
        up(l, r, p);
    }
    int change(int y) {
        int x = uu.back();
        int p = fa[x];
        if(p) {
            if(x == CHANGEU)val[x] ^= 1;
            if(val[x]) {
                tnum[p] -= (long long)nde[1].r0 * (num0[p]-nde[1].r1);
                tnum[p] -= (long long)nde[1].r1 * (num1[p]-nde[1].r0);
                num0[p] -= nde[1].r1;
                num1[p] -= nde[1].r0;
            } else {
                tnum[p] -= (long long)nde[1].r1 * (num0[p]-nde[1].r0);
                tnum[p] -= (long long)nde[1].r0 * (num1[p]-nde[1].r1);
                num0[p] -= nde[1].r0;
                num1[p] -= nde[1].r1;
            }
            if(x == CHANGEU)val[x] ^= 1;
        }
        ans -= nde[1].sum;
        update(pos[y], 0, n-1, 1);
        if(p) {
            if(val[x]) {
                tnum[p] += (long long)nde[1].r0 * num0[p];
                tnum[p] += (long long)nde[1].r1 * num1[p];
                num0[p] += nde[1].r1;
                num1[p] += nde[1].r0;
```

```
            } else {
                tnum[p] += (long long)nde[1].r0 * num1[p];
                tnum[p] += (long long)nde[1].r1 * num0[p];
                num0[p] += nde[1].r0;
                num1[p] += nde[1].r1;
            }
        }
        ans += nde[1].sum;
        return p;
    }
} ch[MAXN];
void dfs1(int u, int pre) {
    chain &c = ch[u];
    c.uu.clear();
    int v, x = 0;
    cnt[u] = 1;
    for(int i = head[u]; i != -1; i = edge[i].next) {
        v = edge[i].to;
        if(v == pre)continue;
        dfs1(v, u);
        link[i / 2] = v;
        val[v] = edge[i].f;
        cnt[u] += cnt[v];
        fa[v] = u;
        if(cnt[v] > cnt[x]) x = v;
    }
    if(!x)col[u] = u;
    else col[u] = col[x];
    ch[col[u]].uu.push_back(u);
    num0[u] = 1;
    num1[u] = 0;
    tnum[u] = 0;

}
void dfs2(int x) {
    x = col[x];
    chain &c = ch[x];
    int n = c.uu.size();
    int u, v;
    for(int i = 1; i < n; i++) {
        u = c.uu[i];
        for(int j = head[u]; j != -1; j = edge[j].next) {
            v = edge[j].to;
            if(v == c.uu[i-1] || fa[u] == v)continue;
            dfs2(v);
            if(val[v]) {
```

```
                tnum[u] += (long long)num0[u] * ch[col[v]].nde[1].r0 +
                 ↪  (long long)num1[u] * ch[col[v]].nde[1].r1;
                num0[u] += ch[col[v]].nde[1].r1;
                num1[u] += ch[col[v]].nde[1].r0;
            } else {
                tnum[u] += (long long)num1[u] * ch[col[v]].nde[1].r0 +
                 ↪  (long long)num0[u] * ch[col[v]].nde[1].r1;
                num0[u] += ch[col[v]].nde[1].r0;
                num1[u] += ch[col[v]].nde[1].r1;
            }
        }
    }
    c.init();
    ans += c.nde[1].sum;
}
char str[100];
char str1[100], str2[100];
int main() {
    int T;
    int iCase = 0;
    scanf("%d", &T);
    int n;
    while(T--) {
        ans = 0;
        iCase++;
        scanf("%d", &n);
        map<string, int>mp;
        init();
        for(int i = 1; i <= n; i++) {
            scanf("%s", str);
            mp[str] = i;
        }
        int u, v, f;
        for(int i = 1; i < n; i++) {
            scanf("%s%s%d", str1, str2, &f);
            addedge(mp[str1], mp[str2], f);
            addedge(mp[str2], mp[str1], f);
        }
        int Q;
        char op[10];
        scanf("%d", &Q);
        printf("Case␣#%d:\n", iCase);
        val[1] = 0;
        fa[1] = 0;
        dfs1(1, 1);
        dfs2(1);
```

```
            while(Q--) {
                scanf("%s", op);
                if(op[0] == 'Q') {
                    printf("%I64d\n", ans * 2);
                } else {
                    int id ;
                    scanf("%d", &id);
                    id--;
                    u = link[id];
                    val[u] ^= 1;
                    CHANGEU = u;
                    while(u)
                        u = ch[col[u]].change(u);
                }
            }
        }
        return 0;
}
```

4 搜索

4.1 折半搜索（Meet in middle）

```cpp
/* 例题：codeforces - 888E - 有 n(n<36) 个数，求从中选任意个数加和对
   m(m<10^9) 取模后的最大值
 * 解法：先对序列的前一半进行枚举，之后再对后一半序列进行枚举并维护和前一半
   情况相结合的最优结果。
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
set<ll> s[2];
int a[36], n, m;
void dfs(ll sum, int l, int r) {
    if (l > r) {
        s[r == n].insert(sum);
        return;
    }
    dfs(sum, l + 1, r);
    dfs((sum + a[l]) % m, l + 1, r);
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    dfs(0, 1, n / 2);
    dfs(0, n / 2 + 1, n);
//    printf("%d %d\n", s[0].size(), s[1].size());
    ll ans = 0;
    for (auto left: s[0]) {
        if (left < m) {
            ll res = m - left - 1;
            auto it = s[1].upper_bound(res);
            it--;
            ans = max(ans, left + *it);
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

4.2 跳链（Dancing links）

```cpp
/* 精确覆盖问题：
 *     给个 01 矩阵，选出一些行的集合，使得集合中每一列 ** 恰好 ** 一个 1
 *     用法 bool ans=dlx.dfs(0) 参数是 dfs 层数不用管
```

```
 *     如果要求取最少行，需要在 dfs 修改
 * 重复覆盖问题：
 *     给个 01 矩阵，选出一些行的集合，使得集合中每一列 ** 至少 ** 一个 1
 *     用法 bool ans=dlx.dfs2(0) 参数是 dfs2 层数不用管
 *     如果要求取最少行，需要在 dfs2 修改
 *     多组数据要 dlx.init(n,m)
 * 矩阵单点 0 改 1：link(x,y)
 * 输出方案：dlx.ans[k] 是取的行,0<=k<dlx.cnt
 * maxn 和 maxm 和 maxnode 的值太小可能会 T
 */
struct DLX {
    const static int maxn = 2010;
    const static int maxm = 2010;
    const static int maxnode = 1000010; //矩阵中最多'1'的个数
#define FF(i,A,s) for(int i=A[s];i!=s;i=A[i])
    int L[maxnode], R[maxnode], U[maxnode], D[maxnode];
    int size, col[maxnode], row[maxnode], s[maxm], H[maxn];
    bool vis[maxm];
    int ans[maxn], cnt;
    void init(int n, int m) {
        for(int i = 0; i <= m; i++) {
            L[i] = i - 1;
            R[i] = i + 1;
            U[i] = D[i] = i;
            s[i] = 0;
        }
        for (int i = 1; i <= n; i++) H[i] = -1;
        L[0] = m;
        R[m] = 0;
        size = m + 1;
        cnt = -1;
    }
    void link(int r, int c) {
        U[size] = c;
        D[size] = D[c];
        U[D[c]] = size;
        D[c] = size;
        if(H[r] < 0) H[r] = L[size] = R[size] = size;
        else {
            L[size] = H[r];
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            R[H[r]] = size;
        }
        s[c]++;
        col[size] = c;
```

```
        row[size] = r;
        size++;
    }
    // ==================== 精确覆盖 ===================
    void del(int c) {
        L[R[c]] = L[c];
        R[L[c]] = R[c];
        FF(i, D, c) FF(j, R, i) U[D[j]] = U[j], D[U[j]] = D[j], --s[col[j]];
    }
    void add(int c) {
        R[L[c]] = L[R[c]] = c;
        FF(i, U, c) FF(j, L, i) ++s[col[U[D[j]] = D[U[j]] = j]];
    }
    bool dfs(int k) {
        if(!R[0]) {
            cnt = ~cnt ? min(k, cnt) : k;
            return true;
        }
        if(~cnt && k >= cnt)return false;
        int c = R[0];
        bool mk = 0;
        FF(i, R, 0)if(s[c] > s[i])c = i;
        del(c);
        FF(i, D, c) {
            FF(j, R, i)del(col[j]);
            ans[k] = row[i];
            if(dfs(k + 1)) return true; /* 此为只要输出一解, 较快 */
            // mk|=dfs(k+1); /* 此为输出取行数最少 */
            FF(j, L, i)add(col[j]);
        }
        add(c);
        return mk;
    }
    // ==================== 重复覆盖 ===================
    void remove(int c) {
        FF(i, D, c)L[R[i]] = L[i], R[L[i]] = R[i];
    }
    void resume(int c) {
        FF(i, U, c)L[R[i]] = R[L[i]] = i;
    }
    int A() {
        int res = 0;
        memset(vis, 0, sizeof(vis));
        FF(i, R, 0)if(!vis[i]) {
            res++;
```

```
            vis[i] = 1;
            FF(j, D, i)FF(k, R, j)vis[col[k]] = 1;
        }
        return res;
    }
    bool dfs2(int k) {
        if(!R[0]) {
            cnt = ~cnt ? min(k, cnt) : k;
            return true;
        }
        if(~cnt && k >= cnt)return false;
        bool mk = 0;
        if(cnt == -1 || k + A() < cnt) {
            int temp = inf, c;
            FF(i, R, 0)if(temp > s[i])temp = s[i], c = i;
            FF(i, D, c) {
                remove(i);
                ans[k] = row[i];
                FF(j, R, i)remove(j);
                if (dfs2(k + 1))return true; /* 此为只要输出一解，较快 */
                //mk|=dfs2(k+1); /* 此为输出取行数最少 */
                FF(j, L, i)resume(j);
                resume(i);
            }
        }
        return mk;
    }
#undef FF
} dlx;
```

4.3 反向搜索

```
/* HDU - 1043 - 八数码，输出路径
 * 思路：反向搜索，从目标状态找回状态对应的路径，用康托展开判重
 */
const int MAXN = 1000000; //最多是 9!/2
int fac[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880}; //康拖展开判重
// 0!1!2!3! 4! 5! 6! 7! 8! 9!
bool vis[MAXN];//标记
string path[MAXN];//记录路径
//康拖展开求该序列的 hash 值
int cantor(int s[]) {
    int sum = 0;
    for(int i = 0; i < 9; i++) {
        int num = 0;
```

```
        for(int j = i + 1; j < 9; j++)
            if(s[j] < s[i])num++;
        sum += (num * fac[9-i-1]);
    }
    return sum + 1;
}
struct Node {
    int s[9];
    int loc;//'0' 的位置
    int status;//康拖展开的 hash 值
    string path;//路径
};
int move[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; //u,d,l,r
char indexs[5] = "durl"; //和上面的要相反, 因为是反向搜索
int aim = 46234; //123456780 对应的康拖展开的 hash 值
void bfs() {
    memset(vis, false, sizeof(vis));
    Node cur, next;
    for(int i = 0; i < 8; i++)cur.s[i] = i + 1;
    cur.s[8] = 0;
    cur.loc = 8;
    cur.status = aim;
    cur.path = "";
    queue<Node>q;
    q.push(cur);
    path[aim] = "";
    while(!q.empty()) {
        cur = q.front();
        q.pop();
        int x = cur.loc / 3;
        int y = cur.loc % 3;
        for(int i = 0; i < 4; i++) {
            int tx = x + move[i][0];
            int ty = y + move[i][1];
            if(tx < 0 || tx > 2 || ty < 0 || ty > 2)continue;
            next = cur;
            next.loc = tx * 3 + ty;
            next.s[cur.loc] = next.s[next.loc];
            next.s[next.loc] = 0;
            next.status = cantor(next.s);
            if(!vis[next.status]) {
                vis[next.status] = true;
                next.path = indexs[i] + next.path;
                q.push(next);
                path[next.status] = next.path;
            }
```

```
        }
    }

}
int main() {
    char ch;
    Node cur;
    bfs();
    while(cin >> ch) {
        if(ch == 'x') {
            cur.s[0] = 0;
            cur.loc = 0;
        } else cur.s[0] = ch-'0';
        for(int i = 1; i < 9; i++) {
            cin >> ch;
            if(ch == 'x') {
                cur.s[i] = 0;
                cur.loc = i;
            } else cur.s[i] = ch-'0';
        }
        cur.status = cantor(cur.s);
        if(vis[cur.status]) {
            cout << path[cur.status] << endl;
        } else cout << "unsolvable" << endl;
    }
    return 0;
}/* HDU - 1043 - 八数码，输出路径
 * 思路：反向搜索，从目标状态找回状态对应的路径，用康托展开判重
 */
const int MAXN = 1000000; //最多是 9!/2
int fac[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880}; //康拖展开判
↪ 重
// 0!1!2!3! 4! 5! 6! 7! 8! 9!
bool vis[MAXN];//标记
string path[MAXN];//记录路径
//康拖展开求该序列的 hash 值
int cantor(int s[]) {
    int sum = 0;
    for(int i = 0; i < 9; i++) {
        int num = 0;
        for(int j = i + 1; j < 9; j++)
            if(s[j] < s[i])num++;
        sum += (num * fac[9-i-1]);
    }
    return sum + 1;
}
```

```
struct Node {
    int s[9];
    int loc;//'0' 的位置
    int status;//康拖展开的 hash 值
    string path;//路径
};
int move[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; //u,d,l,r
char indexs[5] = "durl"; //和上面的要相反, 因为是反向搜索
int aim = 46234; //123456780 对应的康拖展开的 hash 值
void bfs() {
    memset(vis, false, sizeof(vis));
    Node cur, next;
    for(int i = 0; i < 8; i++)cur.s[i] = i + 1;
    cur.s[8] = 0;
    cur.loc = 8;
    cur.status = aim;
    cur.path = "";
    queue<Node>q;
    q.push(cur);
    path[aim] = "";
    while(!q.empty()) {
        cur = q.front();
        q.pop();
        int x = cur.loc / 3;
        int y = cur.loc % 3;
        for(int i = 0; i < 4; i++) {
            int tx = x + move[i][0];
            int ty = y + move[i][1];
            if(tx < 0 || tx > 2 || ty < 0 || ty > 2)continue;
            next = cur;
            next.loc = tx * 3 + ty;
            next.s[cur.loc] = next.s[next.loc];
            next.s[next.loc] = 0;
            next.status = cantor(next.s);
            if(!vis[next.status]) {
                vis[next.status] = true;
                next.path = indexs[i] + next.path;
                q.push(next);
                path[next.status] = next.path;
            }
        }
    }

}
int main() {
    char ch;
```

```
    Node cur;
    bfs();
    while(cin >> ch) {
        if(ch == 'x') {
            cur.s[0] = 0;
            cur.loc = 0;
        } else cur.s[0] = ch-'0';
        for(int i = 1; i < 9; i++) {
            cin >> ch;
            if(ch == 'x') {
                cur.s[i] = 0;
                cur.loc = i;
            } else cur.s[i] = ch-'0';
        }
        cur.status = cantor(cur.s);
        if(vis[cur.status]) {
            cout << path[cur.status] << endl;
        } else cout << "unsolvable" << endl;
    }
    return 0;
}
```

4.4 模拟退火

```
/* 问题：单机调度 flowshop 问题
 * 解法：模拟退火算法
 */
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1005;
const double eps = 1e-8;
const double delta = 0.98;
const int T = 3000;
const int I_LOOP = 100; // Internal loop counts
const int O_LOOP = 20;  // Outside loop counts
const int LIMIT = 1000; // bound for P_L
struct node {  // Part info
    int a, p, d;
} J[maxn];
struct sol {   // One permutation and all delays
    int p[maxn], delay;
} best;
int n;
void calc(sol& tmp) { // for each legal permutation, caculate the
    results
    tmp.delay = 0;
```

```
    int now = 0;
    for (int i=1; i<=n; i++) {
        if (now < J[tmp.p[i]].a) now = J[tmp.p[i]].a;
        now += J[tmp.p[i]].p;
        tmp.delay += max(0, now-J[tmp.p[i]].d);
    }
}
sol getNext(sol tmp) {  // for each random change
    sol res = tmp;
    int x = (int)(n*(rand()/(RAND_MAX+1.0)));
    int y = (int)(n*(rand()/(RAND_MAX+1.0)));
    while (x == y) {
        x = (int)(n*(rand()/(RAND_MAX+1.0)));
        y = (int)(n*(rand()/(RAND_MAX+1.0)));
    }
    x++, y++;
    swap(res.p[x], res.p[y]);
    calc(res);
    return res;
}
void SA() { // main SA algorithm
    double t = T;
    sol pre = best;
    int P_L = 0, P_F = 0;
    while (1) { // to change t
        for (int i=0; i<I_LOOP; i++) { // to find optimism
            sol cur = getNext(pre);
            double dE = cur.delay - pre.delay;
            if (dE < 0) { // better
                pre = cur;
                P_L = 0, P_F = 0;
            } else { // decide to accept
                double rd = rand()/(RAND_MAX+1.0);
                if (exp(dE/t)>rd && exp(dE/t)<1) pre = cur;
                P_L++;
            }
            if (P_L > LIMIT) {
                P_F++;
                break;
            }
        }
        if (pre.delay < best.delay) best = pre;
        if (P_F > O_LOOP || t < eps) break;
        t *= delta;
    }
}
```

```
int main() {
    srand(time(NULL));
    scanf("%d", &n);
    for (int i=1; i<=n; i++) scanf("%d%d%d", &J[i].a, &J[i].p, &J[i].d);
    /* initial answer */
    for (int i=1; i<=n; i++) best.p[i] = i;
    calc(best);
    /* caculations */
    SA();
    printf("Final paths:");
    for (int i=1; i<=n; i++) printf(" %d", best.p[i]);
    printf("\nDelays: %d\n", best.delay);
    return 0;
}
```

5 动态规划

5.1 背包问题

```
int dp[maxn];
//0-1 背包，代价为 cost，获得的价值为 weight
void ZeroOnePack(int cost, int weight, int nValue) {
    for (int i = nValue; i >= cost; i--)
        dp[i] = max(dp[i], dp[i - cost] + weight);
}
//完全背包，代价为 cost，获得的价值为 weight
void CompletePack(int cost, int weight, int nValue) {
    for (int i = cost; i <= nValue; i++)
        dp[i] = max(dp[i], dp[i - cost] + weight);
}
//多重背包
void MultiplePack(int cost, int weight, int amount, int nValue) {
    if (cost * amount >= nValue)
        CompletePack(cost, weight, nValue);
    else {
        int k = 1;
        while (k < amount) {
            ZeroOnePack(k * cost, k * weight, nValue);
            amount -= k;
            k <<= 1;
        }
        ZeroOnePack(amount * cost, amount * weight, nValue);
    }
}
```

5.2 最长上升子序列 $O(n \cdot log(n))$

```
const int MAXN = 500010;
int a[MAXN], b[MAXN];

//用二分查找的方法找到一个位置，使得 num>b[i-1] 并且 num<b[i]，并用 num 代
↪ 替 b[i]
int Search(int num, int low, int high) {
    int mid;
    while(low <= high) {
        mid = (low + high) / 2;
        if(num >= b[mid]) low = mid + 1;
        else high = mid - 1;
    }
    return low;
}
```

```
int DP(int n) {
    int i, len, pos;
    b[1] = a[1];
    len = 1;
    for(i = 2; i <= n; i++) {
        if(a[i] >= b[len]) { //如果 a[i] 比 b[] 数组中最大还大直接插入到后面即可
            len = len + 1;
            b[len] = a[i];
        } else { //用二分的方法在 b[] 数组中找出第一个比 a[i] 大的位置并且让 a[i] 替代这个位置
            pos = Search(a[i], 1, len);
            b[pos] = a[i];
        }
    }
    return len;
}
```

5.3 数位 DP

```
/* 例题 1：原题是求 x,y 之间有多少个 0
 * 状态：digit, real 代表是否为前导 0
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll dp[20][20][3][20];
int digit[35];
ll dfs(int len, int hasl, int vto, int real, int fp) {
    if(vto < hasl) return 0;
    if(!len) {
        if(vto == hasl) return 1;
        else return 0;
    }
    if((!fp) && dp[len][hasl][real][vto] != -1) return dp[len][hasl][real][vto];
    ll ret = 0;
    int fpmax = fp ? digit[len] : 9;
    for (int i = 0; i <= fpmax ; ++i) {
        if(real) ret += dfs(len - 1, hasl + (i == 0), vto, real, fp && i == digit[len]);
        else ret += dfs(len - 1, 0, vto, real || i != 0, fp && i == digit[len]);
    }
    if(!fp) dp[len][hasl][real][vto] = ret;
    return ret;
```

```
}
ll calc(ll n) {
    int len = 0;
    while(n > 0) {
        digit[++len] = n % 10;
        n /= 10;
    }
    ll ans = 0;
    for (int i = 1; i <= len ; ++i) ans += 1ll * i * dfs(len, 0, i, 0, 1);
    return ans;
}
int main() {
    int t;
    ll n, m;
    int kase = 1;
    memset(dp, -1, sizeof(dp));
    scanf("%d", &t);
    while(t--) {
        ll ans = 0;
        scanf("%lld%lld", &n, &m);
        if(n == 0) ans = calc(m) + 1;
        else ans = calc(m) - calc(n - 1);
        printf("Case %d: %lld\n", kase++, ans);
    }
    return 0;
}
/* 例题 2：定义重心是数的某一位，这个数平衡当且仅当左右两侧权乘以重心相等，询问区间平衡数个数
 * 解法：前 i 位数，支点是 j，力矩是 k，然后枚举支点计算即可
 */
#define rep(i,a,n) for (int i=a;i<n;i++)
typedef long long ll;
ll dp[20][20][2020];
int digit[20];
ll dfs(int len, int mid, int M, bool limit) {
        if (!len) return M == 0;
        if (M < 0) return 0;
        if (!limit && dp[len][mid][M] != -1) return dp[len][mid][M];
        ll res = 0;
        int End = limit ? digit[len] : 9;
        rep(i, 0, End + 1) res += dfs(len - 1, mid, M + i * (len - mid), limit && i == End);
        if (!limit) dp[len][mid][M] = res;
        return res;
}
```

```
ll calc(ll x) {
	int len = 0;
	ll ans = 0;
	while (x) {
		digit[++len] = x % 10;
		x /= 10;
	}
	rep(i, 1, len + 1) ans += dfs(len, i, 0, 1);
	return ans - len + 1;
}
/* 例题 3：求在 [l,r] 中能够整除自己每个数位上的数字的数的个数
 * 解法：sum%(x*n)%x == sum%x，对于每一位数字如果他们的最小公倍数 (lcm) 能够整除这个数字，那么这个数字就符合要求，计算出 1 到 9 的最小公倍数是 2520，因此 dp[i]][j][k]，i 表示第 i 位，j 表示最高位到第 i 位的 lcm，k 表示数字对 2520 的模，但是 k 的值仍然较大，但是 1 到 9 的最小公倍数只有 48 种，所以需要离散化处理
 */
#define rep(i,a,n) for (int i=a;i<n;i++)
typedef long long ll;
ll dp[20][2520][50];
int LCM[2530], digit[20];
int gcd(int a, int b) {
	if (b == 0) return a;
	return gcd(b, a % b);
}
int lcm(int a, int b) {
	return a / gcd(a, b) * b;
}
ll dfs(int pos, int preSum, int preLcm, bool limit) {
	if (!pos) return preSum % preLcm == 0;
	if (!limit && dp[pos][preSum][LCM[preLcm]] != -1) return dp[pos][preSum][LCM[preLcm]];
	int End = limit ? digit[pos] : 9;
	ll res = 0;
	rep(i, 0, End + 1) {
		int nowSum = (preSum * 10 + i) % 2520;
		int nowLcm = preLcm;
		if (i) nowLcm = lcm(preLcm, i);
		res += dfs(pos - 1, nowSum, nowLcm, limit && i == End);
	}
	if (!limit) dp[pos][preSum][LCM[preLcm]] = res;
	return res;
}
ll calc(ll x) {
	int len = 0;
	while (x) {
```

```
                digit[++len] = x % 10;
                x /= 10;
        }
        return dfs(len, 0, 1, 1);
}
int main() {
        int t;
        ll l, r;
        memset(dp, -1, sizeof dp);
        int cnt = 0;
        rep(i, 1, 2531) {
                if (2520 % i == 0) LCM[i] = cnt++;
        }
        scanf("%d", &t);
        while (t--) {
                scanf("%I64d%I64d", &l, &r);
                printf("%I64d\n", calc(r) - calc(l - 1));
        }
        return 0;
}
```

6 数学

6.1 线性筛

```
/* 线性筛逆元/阶乘逆元, 上限多大才开多大, 别 MLE 了 */
const static int mx = 2e7 + 10;
int inv[mx];// [0,mx) 逆元
int jc[mx], invjc[mx]; // 0!~(mx-1)! 逆元
void init() {
    // [0,mx) 逆元
    inv[0] = inv[1] = 1;
    int p, q;
    for (int i = 2; i < mx; i++) {
        p = mod / i, q = mod - p * i;
        inv[i] = 1ll * (mod - p) * inv[q] % mod;
    }
    // 0!~(mx-1)! 及逆元
    jc[0] = 1;
    for (int i = 1; i < mx; i++)jc[i] = 1ll * jc[i - 1] * i % mod;
    invjc[mx - 1] = quickpow(jc[mx - 1], mod - 2, mod);
    for (int i = mx - 2; i >= 0; i--)invjc[i] = 1ll * invjc[i + 1] * (i +
     ↪ 1) % mod;
}
/* 线性筛素数 */
const static int mx = 2e7 + 10;
int prime[mx], len;
bool isprime[mx];
void init() {
    len = 0;
    memset(isprime, 0, sizeof isprime);
    for (int i = 2; i < mx; i++) {
        if (!isprime[i])isprime[i] = 1, prime[++len] = i;
        else isprime[i] = 0;
        for (int j = 1; j <= len && prime[j]*i < mx; j++) {
            isprime[prime[j]*i] = 1;
            if (i % prime[j] == 0)break;
        }
    }
}
/* 线性筛欧拉函数 phi */
const static int mx = 2e7 + 10;
int prime[mx], phi[mx], len;
void init() {
    len = 0;
    memset(phi, 0, sizeof phi);
    phi[1] = 1;
```

```
    for (int i = 2; i < mx; i++) {
        if (!phi[i])phi[i] = i - 1, prime[++len] = i;
        for (int j = 1; j <= len && prime[j]*i < mx; j++) {
            if (i % prime[j] == 0) {
                phi[prime[j]*i] = phi[i] * prime[j];
                break;
            } else phi[prime[j]*i] = phi[i] * (prime[j] - 1);
        }
    }
}
/* 线性筛莫比乌斯函数 mu */
const static int mx = 2e7 + 10;
int prime[mx], mu[mx], len;
bool mrk[mx];
void init() {
    len = 0;
    memset(mu, 127 / 3, sizeof mu);
    mu[1] = 1;
    for (int i = 2; i < mx; i++) {
        if (mu[i] > 1)mu[i] = -1, prime[++len] = i;
        for (int j = 1; j <= len && prime[j]*i < mx; j++) {
            if (i % prime[j] == 0) {
                mu[prime[j]*i] = 0;
                break;
            } else mu[prime[j]*i] = -mu[i];
        }
    }
}
/* 线性筛任意积性函数（需要能够快速求出素数幂的函数值 f[p^k]）*/
const static int mx = 2e7 + 10;
int prime[mx], dprime[mx], f[mx], len;
// dprime[i] 是 i 的最小质因子部分，可能不是最小质因子 p，而是 p^k
void init() {
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i < mx; i++) {
        if(!dprime[i])dprime[i] = prime[++len] = i, f[i] =;
        // f 取值自己算，这里 i 是质数
        for (int j = 1; j <= len && i * prime[j] < mx; j++) {
            if (i % prime[j] == 0) {
                dprime[i * prime[j]] = dprime[i] * prime[j];
                if(dprime[i] == i)f[i * prime[j]] =;
                // 这里 i*prime[j] 是 prime[j]^k,i=prime[j]^(k-1)
                else f[i * prime[j]] = f[i / dprime[i]] * f[dprime[i] * prime[j]];
                break;
```

```
                }
                dprime[i * prime[j]] = prime[j];
                f[i * prime[j]] = f[i] * f[prime[j]];
            }
        }
}
```

6.2 min_25 法筛质数

```
/* 必须先调用 init(), 注意 mx 要开够
 * G(x,plen) 返回 1~x 的所有质数之和
 */
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
const int mx = 1e6 + 5; // 这里 mx=sqrt(n),1e6 不够的话还要开大
int prime[mx / 10], plen;
bool isprime[mx];
int psum[mx], pcount[mx], psum2[mx];
const static int mod = 1e9 + 7;
inline ll quickpow(ll a, int b, int md = mod) {
    ll s = 1;
    while (b) {
        if (b & 1)s = s * a % md;
        a = a * a % md;
        b >>= 1;
    }
    return s;
}
inline void init() {
    for (int i = 2; i < mx; i++) {
        if (!isprime[i])prime[++plen] = i, psum[i] = i, pcount[i] = 1, psum2[plen] = (psum2[plen - 1] + i) % mod;
        psum[i] += psum[i - 1];
        if (psum[i] >= mod)psum[i] -= mod;
        pcount[i] += pcount[i - 1];
        for (int j = 1, p = prime[j] * i; j <= plen && p < mx; j++, p = prime[j] * i) {
            isprime[p] = 1;
            if (i % prime[j] == 0)break;
        }
    }
}
ll inv2 = quickpow(2, mod - 2, mod);
ll Sum(ll t) {
    return (t + 1) % mod * t % mod * inv2 % mod;   // (t+1)*t/2
```

```
}
ll G(ll n, int m) {
    if(n < mx && n < (ll)prime[m]*prime[m])return psum[n];
    if(!m)return Sum(n);
    for(; n < (ll)prime[m]*prime[m]; --m);
    ll ans = 0;
    for(; m; --m)ans = ((ans - (G(n / prime[m], m - 1) - psum[m - 1]) *
    ↪ prime[m]) % mod + mod) % mod;
    return (ans + Sum(n)) % mod;
}
int main() {
    init();
    ll n;
    while(~scanf("%lld", &n))printf("%lld\n", G(n, plen));
    return 0;
}
```

6.3 非互质逆元

```
// a 对模 MOD 的逆元，无解返回-1
ll exgcd(ll a, ll b, ll &x, ll &y) {
    ll d = a;
    if(b != 0)d = exgcd(b, a % b, y, x), y -= (a / b) * x;
    else x = 1, y = 0;
    return d;
}
ll inv(ll a, ll MOD) {
    ll x, y, d = exgcd(a, MOD, x, y);
    return d == 1 ? (x + MOD) % MOD : -1;
}
```

6.4 拆分数

```
/*  n*sqrt(n) 筛拆分数 P(x) P(x) 表示把 x 表示成多个数相加的方案数（不计顺
↪ 序），如 7=3+2+2
 *
 *  n^2 筛拆分数 p(n,m) p(n,m) 表示把 x 表示成恰好 m 个数相加的方案数（不
↪ 计顺序）
 *      p(n,m) 也表示把 x 表示成最大值恰好为 m 的多个数相加的方案数（不计顺
↪ 序） [可以证明这两个相等]
 *      v(n,m) 表示把 x 表示成最多 m 个数相加的方案数（不计顺序），即
↪ p(n,i) 的前缀和
 */
int p[2010][2010];
int v[2010][2010];
int P[100010];
```

```
const static int mod = 1e9 + 7;
void inc(int &x, int y) {
    x = x + y >= mod ? x + y - mod : x + y;
}
void dec(int &x, int y) {
    x = x - y < 0 ? x - y + mod : x - y;
}
inline void initP() {
    P[0] = 1;
    for(int i = 1; i <= 60000; i++) {
        for(int j = 1, w = 1; w <= i; w += 3 * j + 1, j++)
            if(j & 1)inc(P[i], P[i - w]);
            else dec(P[i], P[i - w]);
        for(int j = 1, w = 2; w <= i; w += 3 * j + 2, j++)
            if(j & 1)inc(P[i], P[i - w]);
            else dec(P[i], P[i - w]);
    }
}
inline void initp() {
    p[0][0] = 1;
    for (int i = 0; i <= 1000; i++)v[0][i] = 1;
    for (int i = 1; i <= 1000; i++) {
        for (int j = 1; j <= i; j++) {
            p[i][j] = v[i - j][j];
            v[i][j] = v[i][j - 1] + p[i][j];
            if (v[i][j] >= mod)v[i][j] -= mod;
        }
        for (int j = i + 1; j <= 1000; j++) {
            p[i][j] = 0;
            v[i][j] = v[i][i];
        }
    }
}
```

6.5 扩展欧几里得

```
/* 找出一组 ax+by==gcd(a,b) 的方案, 返回值即 gcd(a,b)
 * ax==1(mod b) 等价于 ax-by==1
 * 可能需要调整解 x 到最小正数
 */
ll exgcd(ll a, ll b, ll &x, ll &y) {
    ll d = a;
    if(b != 0)d = exgcd(b, a % b, y, x), y -= (a / b) * x;
    else x = 1, y = 0;
    return d;
}
```

```cpp
// ax = 1(mod n) 求逆元
ll mod_reverse(ll a, ll n) {
    ll x, y;
    ll d = exgcd(a, n, x, y);
    if(d == 1) return (x % n + n) % n;
    return -1;
}
```

6.6 随机素数测试

```cpp
// 使用 is_prime(x), 切记：一定要用快速乘不能用普通乘，基本上是一定会爆 ll
ll mul(ll a, ll b, ll p) {
    if(b < 0)b = (b % p + p) % p;
    a %= p;
    if(a < b)swap(a, b);
    ll ans = 0;
    while(b) {
        if(b & 1)ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
}
ll quickpow(ll a, ll b, ll p) {
    ll s = 1;
    while (b) {
        if(b & 1)s = mul(s, a, p);
        a = mul(a, a, p);
        b >>= 1;
    }
    return s;
}
bool witness(ll a, ll b) {
    if(a == b)return 1;
    ll s = b - 1;
    int t = 0;
    while(!(s & 1))s >>= 1, t++;
    ll x = quickpow(a, s, b);
    if(x == 1)return 1;
    while(t--) {
        if(x == b - 1)return 1;
        x = mul(x, x, b);
        if(x == 1)return 0;
    }
    return 0;
}
```

```
bool is_prime(ll x) {
    if(x < 2)return 0;
    static int p[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31};
    for(int i = 0; i <= 10; i++)if(!witness(p[i], x))return 0;
    return 1;
}
```

6.7 大整数分解

```
/* 质因子分解，小数据用筛法直接判，大数据用 pollard_rho
 * factor[i] 是含有多少个质因子 i
 * map<ll,int>::iterator c,c->first 表示质因子，c->second 表示次方
 * wrk.out(p+1,q+1,len) 可以导出分解结果
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn_for_division 1000001
#define rep_times 20
bool is_prime[maxn_for_division];
vector<int>prime;
inline void get_prime() {
    for(int i = 0; i < maxn_for_division; i++)is_prime[i] = 1;
    is_prime[0] = is_prime[1] = 0;
    for(int i = 2; i < maxn_for_division; i++)
        if (is_prime[i]) {
            prime.push_back(i);
            for (int j = i; j < maxn_for_division; j += i)is_prime[j] =
            ↪ 0;
        }
}

inline ll gcd(ll a, ll b) {
    if (!b)return a;
    return gcd(b, a % b);
}
inline ll mod_mul(ll a, ll b, ll p) {
    ll ans = 0ll;
    a %= p, b %= p;
    if (a > b)swap(a, b);
    while (b) {
        if (b & 1)ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
```

```
}
ll mod_pow(ll a, ll b, ll p) {
    ll ans = 1ll;
    a %= p;
    while (b) {
        if (b & 1)ans = mod_mul(ans, a, p);
        a = mod_mul(a, a, p);
        b >>= 1;
    }
    return ans;
}
struct get_factor {
    ll n;
    map<ll, int>factor;
    bool witness(ll a, ll n) {
        ll m = n - 1;
        int j = 0;
        while(!(m & 1))j++, m >>= 1;
        ll x = mod_pow(a, m, n);
        if (x == 1 || x == n - 1)return 0;
        while(j--) {
            x = mod_mul(x, x, n);
            if(x == n - 1)return 0;
        }
        return 1;
    }
    bool Miller_Rabin(ll n) { //判断 n 是否为素数
        srand(time(0));
        if(n < 2)return 0;
        if(n == 2)return 1;
        if (!(n & 1))return 0;
        for(int i = 0; i < rep_times; i++) {
            ll a = rand() % (n - 1) + 1;
            if (witness(a, n))return 0;
        }
        return 1;
    }
#undef rep_times
    ll Pollard_Rho(ll n, int c) {
        ll x = 2, y = 2, d = 1;
        while (d == 1) {
            x = mod_mul(x, x, n) + c;
            y = mod_mul(y, y, n) + c;
            y = mod_mul(y, y, n) + c;
            d = gcd((x - y >= 0 ? x - y : y - x), n);
        }
```

```
        if (d == n)return Pollard_Rho(n, c + 1);
        return d;
    }
    bool Is_Prime(ll n) {
        return n < maxn_for_division && is_prime[n] || n >= maxn_for_division && Miller_Rabin(n);
    }
    void Find_Factor(ll n) {
        if (n == 1ll) {
            factor[1ll] = 1;
            return;
        }
        if (Is_Prime(n)) {
            factor[n]++;
            return;
        }
        for (int i = 0; i < prime.size() && prime[i] <= n; i++)
            if (n % prime[i] == 0) {
                while (n % prime[i] == 0) {
                    factor[prime[i]]++;
                    n /= prime[i];
                }
            }
        if (n != 1) {
            if (Is_Prime(n))factor[n]++;
            else {
                ll p = Pollard_Rho(n, 1);
                Find_Factor(p);
                Find_Factor(n / p);
            }
        }
    }
    inline void calc() {
        factor.clear();
        Find_Factor(n);
    }
    inline void out(ll *p, int *q, int &len) { //导出质因子分解结果, p 数组放质因子, q 数组放次数, len 从 1 开始
        len = 0;
        for(map<ll, int>::iterator c = factor.begin(); c != factor.end();) {
            p[++len] = c->first;
            q[len] = c->second;
            c++;
        }
    }
```

```
    inline void put() { //for debug
        printf("%lld = ", n);
        for(map<ll, int>::iterator c = factor.begin(); c != factor.end();) {
            printf("%lld^%d", c->first, c->second);
            if((++c) != factor.end())printf("*");
        }
        puts("");
    }
} wrk;
int main() {
    get_prime();
    while(~scanf("%lld", &wrk.n)) {
        wrk.calc();
        wrk.put();
    }
    return 0;
}
```

6.8 中国剩余定理

- 除数无限制

```
//r: 余数,d: 模数,n: 方程数, 下标 0 开始, 无解返回-1
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    ll d=a;
    if(b!=0)d=exgcd(b,a%b,y,x),y-=(a/b)*x;
    else x=1,y=0;
    return d;
}
ll CRT(ll* d,ll* r,int n)
{
    ll M=d[0],R=r[0],x,y,g,c,t;
    for(int i=1;i<n;i++)
    {
        g=exgcd(M,d[i],x,y);
        c=r[i]-R;t=d[i]/g;
        if(c%g!=0)return -1;
        x=(x*(c/g)%t+t)%t;
        R+=x*M;
        M=M/g*d[i];
    }
    R=(R%M+M)%M;
    return R;
}
```

- 除数必须小于 long long

```
//r: 余数,d: 模数,n: 方程数, 下标 0 开始
ll mul(ll a,ll b,ll p)
{
    if(b<0)b=(b%p+p)%p;
    a%=p;
    if(a<b)swap(a,b);
    ll ans=0;
    while(b)
    {
        if(b&1)ans=(ans+a)%p;
        a=(a+a)%p;
        b>>=1;
    }
    return ans;
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    ll d=a;
    if(b!=0)d=exgcd(b,a%b,y,x),y-=(a/b)*x;
    else x=1,y=0;
    return d;
}
ll CRT(ll d[],ll r[],int n)
{
    ll M=1,Mi,ans=0;
    ll x,y,t;
    for(int i=0;i<n;i++)M*=d[i];
    for(int i=0;i<n;i++)
    {
        Mi=M/d[i];
        t=exgcd(Mi,d[i],x,y);
        ans=(ans+mul(mul(Mi,x,M),d[i],M))%M;
        //ans=(ans+Mi*x%M*d[i]%M)%M;
    }
    if(ans<0)ans+=M;
    return ans;
}
```

6.9 离散对数

```
// a^x==b(mod MOD), 无解 ==-1
ll exgcd(ll a, ll b, ll &x, ll &y) {
    ll d = a;
    if(b != 0)d = exgcd(b, a % b, y, x), y -= (a / b) * x;
    else x = 1, y = 0;
```

```
    return d;
}
ll inv(ll a, ll MOD) {
    ll x, y, d = exgcd(a, MOD, x, y);
    return d == 1 ? (x + MOD) % MOD : -1;
}
ll dclog(ll a, ll b, ll MOD) {
    map<ll, ll>base;
    ll m = (ll)ceil(sqrt(MOD) + 1), e = 1, i, v;
    base[1] = 0;
    for(i = 0; i < m; i++, e = (e * a) % MOD) {
        if(!base.count(e))base[e] = i;
        if (b == e)return i;
    }
    v = inv(e, MOD);
    if (v == -1)return -1;
    for(i = 0; i <= MOD / m; i++) {
        if(base[b])return (i * m + base[b]) % (MOD - 1);
        b = (b * v) % MOD;
    }
    return -1;
}
```

6.10 原根

- **定义:** 设 $m>1$，$gcd(a,m)=1$，使得 $a^d \equiv 1(mod\ m)$ 成立的最小的正整数 d 为 a 对模 m 的阶，记为 $\delta_m(a)$，如果 $\delta_m(a)=\varphi(m)$ 则称 a 是模 m 的原根。
- **定理 1:** 如果 $m>1$，$gcd(a,m)=1$，正整数 d 满足 $a^d \equiv 1(mod\ m)$ 则 $\delta_m(a)$ 整除 d。
- **定理 2:** 模 m 有原根的充要条件是 $m=2,4,\cdots,p^n,2\cdot p^n$，其中 p 是奇质数，且 n 是任意正整数。
- **定理 3:** 如果模 m 有原根，那么它一定有 $\varphi(\varphi(m))$ 个原根。
- **定理 4:** 如果 p 是素数，那么一定有原根，并且个数为 $\varphi(p-1)$。
- **求模 p 素数原根方法:** 对 $p-1$ 素因子分解，即 $p-1=p_1^{a_1}p_2^{a_2}\cdots p_k^{a_k}$，如果恒有 $g^{\frac{p-1}{p_i}} \neq 1(mod\ p)$ 那么 g 就是原根（合数则把 $p-1$ 换成 $\varphi(p)$ 即可）

```
//r: 余数,d: 模数,n: 方程数, 下标 0 开始, 无解返回-1
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    ll d=a;
    if(b!=0)d=exgcd(b,a%b,y,x),y-=(a/b)*x;
    else x=1,y=0;
    return d;
```

```
}
ll CRT(ll* d,ll* r,int n)
{
    ll M=d[0],R=r[0],x,y,g,c,t;
    for(int i=1;i<n;i++)
    {
        g=exgcd(M,d[i],x,y);
        c=r[i]-R;t=d[i]/g;
        if(c%g!=0)return -1;
        x=(x*(c/g)%t+t)%t;
        R+=x*M;
        M=M/g*d[i];
    }
    R=(R%M+M)%M;
    return R;
}
```

- 除数必须小于 long long

```
//r: 余数,d: 模数,n: 方程数, 下标 0 开始
ll mul(ll a,ll b,ll p)
{
    if(b<0)b=(b%p+p)%p;
    a%=p;
    if(a<b)swap(a,b);
    ll ans=0;
    while(b)
    {
        if(b&1)ans=(ans+a)%p;
        a=(a+a)%p;
        b>>=1;
    }
    return ans;
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    ll d=a;
    if(b!=0)d=exgcd(b,a%b,y,x),y-=(a/b)*x;
    else x=1,y=0;
    return d;
}
ll CRT(ll d[],ll r[],int n)
{
    ll M=1,Mi,ans=0;
    ll x,y,t;
    for(int i=0;i<n;i++)M*=d[i];
    for(int i=0;i<n;i++)
```

```
    {
        Mi=M/d[i];
        t=exgcd(Mi,d[i],x,y);
        ans=(ans+mul(mul(Mi,x,M),d[i],M))%M;
        //ans=(ans+Mi*x%M*d[i]%M)%M;
    }
    if(ans<0)ans+=M;
    return ans;
}
```

```
#include <bits/stdc++.h>
const int MAXN = 100000;
int prime[MAXN + 1];
void getPrime() { //素数筛选和合数分解
    memset(prime, 0, sizeof(prime));
    for(int i = 2; i <= MAXN; i++) {
        if(!prime[i])prime[++prime[0]] = i;
        for(int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++) {
            prime[prime[j]*i] = 1;
            if(i % prime[j] == 0) break;
        }
    }
}
long long factor[100][2];
int fatCnt;
int getFactors(long long x) {
    fatCnt = 0;
    long long tmp = x;
    for(int i = 1; prime[i] <= tmp / prime[i]; i++) {
        factor[fatCnt][1] = 0;
        if(tmp % prime[i] == 0) {
            factor[fatCnt][0] = prime[i];
            while(tmp % prime[i] == 0) {
                factor[fatCnt][1]++;
                tmp /= prime[i];
            }
            fatCnt++;
        }
    }
    if(tmp != 1) {
        factor[fatCnt][0] = tmp;
        factor[fatCnt++][1] = 1;
    }
    return fatCnt;
}
long long pow_m(long long a, long long n, long long mod) {
    long long ret = 1;
```

```
    long long tmp = a % mod;
    while(n) {
        if(n & 1)ret = ret * tmp % mod;
        tmp = tmp * tmp % mod;
        n >>= 1;
    }
    return ret;
}
//求素数 P 的最小的原根
void solve(int P) {
    if(P == 2) {
        printf("1\n");
        return;
    }
    getFactors(P-
        1);
    for(int g = 2; g < P; g++) {
        bool flag = true;
        for(int i = 0; i < fatCnt; i++) {
            int t = (P-1) / factor[i][0];
            if(pow_m(g, t, P) == 1) {
                flag = false;
                break;
            }
        }
        if(flag) {
            printf("%d\n", g);
            return;
        }
    }
}
int main() {
    getPrime();
    int T;
    int P;
    scanf("%d", &T);
    while(T--) {
        scanf("%d", &P);
        solve(P);
    }
    return 0;
}
```

6.11 求 A^B 的约数之和对 MOD 取模

```
// 参考 POJ1845
// 里面有一种求 1+p+p^2+p··· ^3+p^n 的方法。
// 需要素数筛选和合数分解的程序，需要先调用 getPrime();
ll pow_m(ll a, ll n) {
    ll ret = 1;
    ll tmp = a % MOD;
    while(n) {
        if(n & 1)ret = (ret * tmp) % MOD;
        tmp = tmp * tmp % MOD;
        n >>= 1;
    }
    return ret;
}
// 计算 1+p+p^2+...+p^n
ll sum(ll p, ll n) {
    if(p == 0)return 0;
    if(n == 0)return 1;
    if(n & 1) {
        return ((1 + pow_m(p, n / 2 + 1)) % MOD * sum(p, n / 2) % MOD) %
         ↪ MOD;
    } else return ((1 + pow_m(p, n / 2 + 1)) % MOD * sum(p, n / 2-1) +
     ↪ pow_m(p, n / 2) % MOD) % MOD;
}
// 返回 A^B 的约数之和%MOD
ll solve(ll A, ll B) {
    getFactors(A);
    ll ans = 1;
    for(int i = 0; i < fatCnt; i++) {
        ans *= sum(factor[i][0], B * factor[i][1]) % MOD;
        ans %= MOD;
    }
    return ans;
}
```

6.12 组合数学基础

6.12.1 组合数

n 个物品中不记顺序取 m 个的方案：C_n^m

- 显式解 $C(n,m) = \frac{n!}{m!(n-m)!}$
- 递推解 $C(n,m) = C(n-1,m) + C(n-1,m-1), C(i,i) = C(i,0) = 1$
- **Lucas 定理:** $C(n,m)\%p = C(n/p,m/p) \times C(n\%p,m\%p)\%p$ 这里好处是可以递归

- **范德蒙卷积公式:** $\sum_{k=0}^{m} C(m_1,k)\times C(m_2,n-k)=C(m_1+m_2,n)$
  - 令 $m_1=m_2=n, C(2n,n)=C(n,0)^2+C(n,1)^2+\cdots+C(n,n)^2$
- 二项式系数具有单峰性、对称性

n 个球放进 m 个盒，根据球是否相同，盒是否相同，盒是否非空，共 8 种可能

- 球相同，盒相同，盒非空：$v(n,m)$ 即总和为 n，恰好 m 个数相加的拆分数，见拆分数
- 球相同，盒相同，盒可空：$p(n)$ - 拆分数
- 球相同，盒不同，盒非空：$C(n-1,m-1)$
- 球相同，盒不同，盒可空：$C(n+m-1,m-1)$
- 球不同，盒相同，盒非空：$S(n,m)$ - 第二类斯特林数
- 球不同，盒相同，盒可空：$S(n,1)+\cdots+S(n,m)$
- 球不同，盒不同，盒非空：$S(n,m)\cdot m!$
- 球不同，盒不同，盒可空：m^n
- n 种元素（每种足够多）选 m 个：$C(n+m-1,m)$

6.12.2 排列数

n 个物品中记顺序取 m 个的方案：$P(n,m)=\frac{n!}{(n-m)!}=C(n,m)\times m!$

6.12.3 错排数

全排列中满足 $a[i]\neq i$ 的方案数：$D(1)=0, D(2)=1, D(n)=(n-1)\cdot[D(n-2)+D(n-1)]$

6.12.4 鸽巢原理

$n+1$ 个物体放进 n 个盒子，至少有一个盒子里有两个物体

- 变形 1：m 个正整数 $a[1],\cdots,a[m]$，存在 $0\leq i<j\leq m$，使得 $(a[i+1]+a[i+2]+\cdots+a[j])\mid m$
- 变形 2：n^2+1 个数，要么存在长度是 $n+1$ 的最长上升子序列，要么存在长度是 $n+1$ 的最长下降子序列。

6.12.5 斐波那契数列

$f_0=0,f_1=1,f_n=f_{n-1}+f_{n-2}(n\geq 2)$

- 显式解 $f_n=\frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n-(\frac{1-\sqrt{5}}{2})^n]$
- 前缀和 $S_n=f_1+f_2+\cdots+f_n=f_{n+2}-1$
  - 奇数项前缀和 $f_1+f_3+\cdots+f_{2n-1}=f_{2n}-f_2+f_1$
  - 偶数项前缀和 $f_2+f_4+\cdots+f_{2n}=f_{2n+1}-f_1$
  - 平方和 $f_1^2+f_2^2+\cdots+f_n^2=f_nf_{n+1}$
- 一些性质:
  - 对 $\forall n>0,\frac{f_{2n}}{f_n}=f_{n-1}+f_{n+1}$
  - 相邻三项的关系：对 $\forall n>0,f_{n-1}f_{n+1}=f_n^2+(-1)^n$
  - 和组合数的关系：$f_n=C_{n-1}^0+C_{n-2}^1+\cdots+C_{n-\lfloor\frac{n-1}{2}\rfloor}^{\lfloor\frac{n-1}{2}\rfloor-1}=\sum\limits_{i=1}^{\lfloor\frac{n-1}{2}\rfloor}C_{n-i}^{i-1}$

6.12.6 特征根

如果有递推关系 $f_n=a_1f_{n-1}+a_2f_{n-2}+\cdots+a_kf_{n-k}$，解特征根方程：$x^k-a_1x^{k-1}-a_2x^{k-2}+\cdots+(-a_k)=0$

6.12.7 卡特兰数

- $f_n=\frac{C(2n,n)}{n+1}$
- $f_n=\sum\limits_{k=0}^{n-1}f_k\cdot f_{n-1-k}$
- $f_n=f_{n-1}\cdot\frac{4n-2}{n+1}$

直接算用第一个，打表用第三个，奇特的时间空间复杂度第二个（基本是不可能的），关于卡特兰数的应用如下:

- 火车进出站问题
- n 对括号构成的合法序列
- 含 n 个叶节点的满二叉树个数
- $n+1$ 条边的三角剖分方案数
- 格点中从 $(0,0)$ 走到 (n,n) 的方案数（只走主对角线下方）

拟卡特兰数: $g_n=n!\cdot f_n$（为了去掉分母）$g_n=(4n-6)\cdot g_{n-1}$（应用：n 个数的无序乘法方案）

6.12.8 斯特林数

- **第一类斯特林数:** 把 p 个对象排成 k 个非空循环环排列的方案（p 个分成 k 个环，每个环非空）$s_{p,0}=0, s_{p,p}=1, s_{p,k}=(p-1)\cdot s_{p-1,k}+s_{p-1,k-1}$
- **第二类斯特林数:** 把 p 个对象排成 k 个不可区分的盒子并且没有空盒子的方案（n 个分成 k 堆，每堆非空）$S_{0,0}=1, S_{p,0}=0, S_{p,p}=1, S_{p,k}=k\cdot S_{p-1,k}+S_{p-1,k-1}$

6.12.9 贝尔数

把 n 个元素划分到 $1\cdots n$ 个集合内的方案数之和，即 n 个元素的集合划分方案数。根据定义贝尔数是第二类斯特林数之和：$B_0=1, B_1=1, B_x=\sum\limits_{k=1}^{x} S_{x,k}$

- $O(n^2)$ 递推：$B_n=\sum\limits_{k=0}^{n-1} C(n-1,k)\cdot B_k$
- 取模 p 意义下，$B_{n+p}=B_n+B_{n+1}\ (mod\ p)$，可以做 $O(p^3)$ 的矩阵快速幂，一次转移 p 个数 $B_{ip}\cdots B_{ip+p-1}\rightarrow B_{(i+1)p}\cdots B_{(i+1)p+p-1}$

6.12.10 拆分数

把一个数拆成一个或多个不增数字之和的方案数。如 $7=3+2+2$，做法是 $O(n\sqrt{n})$ 筛正五边形数然后递推 $p(0)=1, p(x)=p(x-1)+p(x-2)-p(x-5)-p(x-7)$，令 $v(n,m)$ 表示 p 第二维前缀和即 $\sum\limits_{i=1}^{m} p(n,i)$，则 $v(n,m)$ 表示 n 拆成最大数不超过 m 的拆分方案数。则有 $p(n,m)=v(n-m,m)$，即必取一个 m，剩下只要最大值不超过 m 即可。

6.12.11 其他结论

- $gcd(a^{b_i}-c^{d_i})=a^{gcd(b_i)}-c^{gcd(d_i)}$
- $gcd(f_x,f_y)=f_{gcd(x,y)}$，其中 f 为满足斐波那契性质的数列

6.13 爆破线性递推第 n 项

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i,a,n) for (ll i=a;i<n;i++)
#define per(i,a,n) for (ll i=n-1;i>=a;i--)
#define SZ(x) ((long long)(x).size())
#define pb push_back
typedef vector<long long> VI;
const static int mod = 1e9 + 7;
inline ll quickpow(ll a, ll b, ll mod) {
    ll s = 1;
    while (b) {
```

```
        if (b & 1)s = s * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return s;
}
namespace linear_seq {
    const ll N = 10010;
    ll res[N], base[N], _c[N], _md[N];

    vector<ll> Md;
    void mul(ll *a, ll *b, ll k) {
        rep(i, 0, k + k) _c[i] = 0;
        rep(i, 0, k) if (a[i]) rep(j, 0, k)
            _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
        per(i, k, k + k) if (_c[i]) rep(j, 0, SZ(Md))
            _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]])
             ↪ % mod;
        rep(i, 0, k) a[i] = _c[i];
    }
    ll solve(ll n, VI a, VI b) {
        // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
        ll ans = 0, pnt = 0, k = SZ(a);
        assert(SZ(a) == SZ(b));
        rep(i, 0, k) _md[k - 1 - i] = -a[i];
        _md[k] = 1;
        Md.clear();
        rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
        rep(i, 0, k) res[i] = base[i] = 0;
        res[0] = 1;
        while ((1ll << pnt) <= n) pnt++;
        per(p, 0, pnt + 1) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                per(i, 0, k) res[i + 1] = res[i];
                res[0] = 0;
                rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] *
                 ↪ _md[Md[j]]) % mod;
            }
        }
        rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
        if (ans < 0) ans += mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1, 1), B(1, 1);
```

```
        ll L = 0, m = 1, b = 1;
        rep(n, 0, SZ(s)) {
            ll d = 0;
            rep(i, 0, L + 1) d = (d + (ll)C[i] * s[n - i]) % mod;
            if (d == 0) ++m;
            else if (2 * L <= n) {
                VI T = C;
                ll c = mod - d * quickpow(b, mod - 2, mod) % mod;
                while (SZ(C) < SZ(B) + m) C.pb(0);
                rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
                L = n + 1 - L;
                B = T;
                b = d;
                m = 1;
            } else {
                ll c = mod - d * quickpow(b, mod - 2, mod) % mod;
                while (SZ(C) < SZ(B) + m) C.pb(0);
                rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
                ++m;
            }
        }
        return C;
    }
    ll gao(VI a, ll n) {
        VI c = BM(a);
        c.erase(c.begin());
        rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
        return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
    }
};
int main() {
    ll n;
    while (~scanf("%lld", &n)) printf("%lld\n", linear_seq::gao(VI{1, 2,
     3, 4, 5}, n - 1));
    //输入前几项输出第 n 项，前几项越多越好
}
```

6.14 矩阵快速幂

```
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
class Matrix {
#define N 2
public:
    int a[N][N];
    int n;
```

```

    void Init(int key) {
        memset(a, 0, sizeof a);
        if (key) rep(i, 0, n) a[i][i] = 1;
    }

    Matrix operator+ (const Matrix &b) {
        Matrix c;
        c.n = n;
        rep(i, 0, n) rep(j, 0, n) c.a[i][j] = (a[i][j] + b.a[i][j]) %
         mod;
        return c;
    }

    Matrix operator+ (int x) {
        Matrix p = *this;
        rep(i, 0, n) p.a[i][i] = (p.a[i][i] + x % mod) % mod;
        return p;
    }

    Matrix operator- (const Matrix &b) {
        Matrix c;
        c.n = n;
        rep(i, 0, n) rep(j, 0, n) c.a[i][j] = (a[i][j] - b.a[i][j] + mod)
         % mod;
        return c;
    }

    Matrix operator* (const Matrix &b) {
        Matrix c;
        c.n = n;
        c.Init(0);
        rep(i, 0, n) rep(j, 0, n) rep(k, 0, n)
            c.a[i][j] = (c.a[i][j] + a[i][k] * b.a[k][j] % mod) % mod;
        return c;
    }

    Matrix Power(int t) {
        Matrix ans, p = *this;
        ans.n = p.n;
        ans.Init(1);
        while (t) {
            if (t & 1) ans = ans * p;
            p = p * p;
            t >>= 1;
        }
```

```
        return ans;
    }

    void Print() {
        rep(i, 0, n) {
            rep(j, 0, n) {
                if (j == 0) printf("%d", a[i][j]);
                else printf(" %d", a[i][j]);
            }
            puts("");
        }
    }
#undef N
};
/* 带 R、C 版，但是其中快速幂必须是 n*n 方阵 */
struct Matrix {
#define N 155
    int r, c;
    int a[N][N];

    Matrix(int r = 0, int c = 0) {
        this->r = r; this->c = c;
        memset(a, 0, sizeof a);
    }

    Matrix operator* (Matrix& b) {
        Matrix res(r, b.c);
        for (int k = 0; k < c; k++) {
            for (int i = 0; i < r; i++) {
                if (!a[i][k]) continue;
                for (int j = 0; j < b.c; j++) {
                    res.a[i][j] += a[i][k] * b.a[k][j];
                }
            }
        }
        return res;
    }

    Matrix Power(int t) {
        Matrix res(r, r), p = *this;
        for (int i = 0; i < r; i++) res.a[i][i] = 1;
        while (t > 0) {
            if (t & 1) res = res * p;
            p = p * p;
            t >>= 1;
        }
```

```
            return res;
        }

        inline void Print() {
            printf("<%d,%d>\n", r, c);
            for (int i = 0; i < r; i++) {
                for (int j = 0; j < c; j++) {
                    if (j == 0) printf("%d", a[i][j]);
                    else printf(" %d", a[i][j]);
                }
                puts("");
            }
        }
#undef N
};
```

6.15 快速幂、快速乘

```
ll quickpow(ll a, ll b, ll p) {
    if (a > p) a %= p;
    ll s = 1;
    while (b) {
        if(b & 1)s = mul(s, a, p);
        a = mul(a, a, p);
        //if(b&1)s=s*a%p;a=a*a%p;
        b >>= 1;
    }
    return s;
}
ll mul_1(ll a, ll b, ll p) {
    if(b < 0) b = (b % p + p) % p;
    a %= p;
    if(a < b)swap(a, b);
    ll ans = 0;
    while(b) {
        if(b & 1)ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
}
ll mul_2(ll x, ll n, ll MOD) {
    ll res = x * n - (ll)((long double)x * n / MOD + 0.5) * MOD;
    while(res < 0)res += MOD;
    while(res >= MOD) res -= MOD;
    return res;
```

```
}
```

6.16 高斯消元

- 浮点数方程

```
#define eps 1e-9
const int MAXN = 220;
double a[MAXN][MAXN], x[MAXN];
// 方程的左边的矩阵和等式右边的值，求解之后 x 存的就是结果
int equ, var; // 方程数和未知数个数
int Gauss() { // 返回 0 表示无解,1 表示有解
    int i, j, k, col, max_r;
    for(k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for(i = k + 1; i < equ; i++)
            if(fabs(a[i][col]) > fabs(a[max_r][col]))
                max_r = i;
        if(fabs(a[max_r][col]) < eps)return 0;
        if(k != max_r) {
            for(j = col; j < var; j++) swap(a[k][j], a[max_r][j]);
            swap(x[k], x[max_r]);
        }
        x[k] /= a[k][col];
        for(j = col + 1; j < var; j++)a[k][j] /= a[k][col];
        a[k][col] = 1;
        for(i = 0; i < equ; i++)
            if(i != k) {
                x[i] -= x[k] * a[i][col];
                for(j = col + 1; j < var; j++)a[i][j] -= a[k][j] * a[i][col];
                a[i][col] = 0;
            }
    }
    return 1;
}
```

- 同余方程

```
#include <bits/stdc++.h>
//#include <algorithm>
using namespace std;
const int MAXN = 310;
const int MAXM = 110000;
const double eps = 1e-14;
const double PI = 4.0 * atan(1.0);
char str[7][10] = {"MON", "TUE", "WED", "THU", "FRI", "SAT", "SUN"};
```

```
int findNum(char *s) {
    for(int i = 0; i < 7; i ++ ) {
        if(strcmp(str[i], s) == 0) return i;
    }
    return 0;
}
int LCM(int a, int b) {
    return a / __gcd(a, b) * b;
}
int n, m, k, cnt[MAXN];
int var, equ, a[MAXN][MAXN], ans[MAXN];
void init() {
    int x;
    char st[10], ed[10];
    memset(a, 0, sizeof a);
    var = n;
    equ = m;
    for(int i = 0; i < m; i ++ ) {
        memset(cnt, 0, sizeof cnt);
        scanf("%d %s %s", &k, st, ed);
        for(int j = 0; j < k; j ++ ) {
            scanf("%d", &x);
            cnt[x - 1] ++;
        }
        a[i][var] = findNum(ed) - findNum(st) + 1;
        for(int j = 0; j < var; j ++ )
            a[i][j] = cnt[j] % 7;
    }
}
void exgcd(int a, int b, int &x, int &y) {
    if(!b) {
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b, y, x);
        y -= x * (a / b);
    }
}
int gauss() {
    int x, y;
    int k, col = 0;
    for(k = 0; k < equ && col < var; k ++, col ++) {
        int Max = 0, row = -1;
        for(int r = k ; r < equ; r ++) {
            if( Max < abs(a[r][col]) )
                Max = abs(a[r][col]), row = r;
```

```
        }
        if(row == -1) {
            k--;
            continue;
        }
        for(int c = col; c <= var; c ++)
            swap(a[k][c], a[row][c]);
        for(int r = k + 1; r < equ; r ++) {
            if(a[r][col]) {
                int lcm = LCM(abs(a[k][col]), abs(a[r][col]));
                int ta = lcm / a[r][col];
                int tb = lcm / a[k][col];
                if(a[r][col] * a[k][col] < 0) tb = -tb;
                for(int c = col; c <= var; c ++ ) {
                    a[r][c] = a[r][c] * ta - a[k][c] * tb;
                    a[r][c] = (a[r][c] % 7 + 7) % 7;
                }
            }
        }
    }
    for(int r = k; r < equ; r ++) {
        if(a[r][var]) return -1;
    }
    if(k < var) return 1;
    for(int r = var - 1; r >= 0; r --) {
        int tmp = a[r][var];
        for(int c = var - 1; c > r; c -- ) {
            tmp -= ans[c] * a[r][c] % 7;
        }
        tmp = (tmp % 7 + 7) % 7;
        exgcd(a[r][r], 7, x, y);
        ans[r] = (tmp * x % 7 + 7) % 7;
        if(ans[r] < 3) ans[r] += 7;
    }
    return 0;
}
int main() {
    while(~scanf("%d %d", &n, &m) && (n || m)) {
        init();
        int free_num = gauss();
        if(free_num == -1) {
            printf("Inconsistent data.\n");
        } else if(free_num == 1) {
            printf("Multiple solutions.\n");
        } else {
            for(int i = 0; i < var; i ++ ) {
```

```
                    printf("%d%c", ans[i], i == var - 1 ? '\n' : ' ');
                }
            }
        }
        return 0;
    }
```

- 异或方程

```
// POJ 1681 需要枚举自由变元，找解中 1 个数最少的对 2 取模的 01 方程组
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1010;
// 有 equ 个方程，var 个变元。增广矩阵行数为 equ，列数为 var+1，分别为 0 到 var
int equ, var;
int a[52][MAXN]; // 增广矩阵
int x[MAXN]; // 解集
int free_x[MAXN];// 用来存储自由变元（多解枚举自由变元可以使用）
int free_num;// 自由变元的个数

// 返回值为-1 表示无解，为 0 是唯一解，否则返回自由变元个数
int Gauss() {
    int max_r, col, k;
    free_num = 0;
    for(k = 0, col = 0 ; k < equ && col < var ; k++, col++) {
        max_r = k;
        for(int i = k + 1; i < equ; i++) {
            if(abs(a[i][col]) > abs(a[max_r][col]))max_r = i;
        }
        if(a[max_r][col] == 0) {
            k--;free_x[free_num++] = col;// 这个是自由变元
            continue;
        }
        if(max_r != k) {
            for(int j = col; j < var ; j++)
                swap(a[k][j], a[max_r][j]);
        }
        for(int i = k + 1; i < equ; i++) {
            if(a[i][col] != 0) {
                for(int j = col; j < var ; j++)
                    a[i][j] ^= a[k][j];
            }
        }
    }
    if(col<var)
```

```
        for(int i = k; i < equ; i++)
            if(a[i][col] != 0)return -1;// 无解
    if(k < var) return var-k;// 自由变元个数
    // 唯一解，回代
    for(int i = var-1; i >= 0; i--) {
        x[i] = a[i][var];
        for(int j = i + 1; j < var; j++)
            x[i] ^= (a[i][j] && x[j]);
    }
    return 0;
}
int n;
void solve() {
    int n;
    ll s;scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%lld",&s);
        for(int j=0;j<50;j++)a[j][i] = s%2,s/=2;
    }
    for(int i=0;i<50;i++)x[i] = 0;
    equ = 50;var = n;
    int t = Gauss();
    if(t == -1||t == 0||n == 1)printf("No\n");
    else printf("Yes\n");
}
int main() {
    int T;
    scanf("%d", &T);
    while(T--) solve();
    return 0;
}
```

6.17 快速傅里叶变换 (FFT)

```
#define data_type int // 记得改类型！记得改类型！记得改类型！记得改类型！
    记得改类型！
#define maxfft 131072+5
const double pi = acos(-1.0);
struct cp {
    double a, b;
    cp operator +(const cp &o)const {
        return (cp) {
            a + o.a, b + o.b
        };
    }
    cp operator -(const cp &o)const {
```

```
        return (cp) {
            a - o.a, b - o.b
        };
    }
    cp operator *(const cp &o)const {
        return (cp) {
            a *o.a - b *o.b, b *o.a + a *o.b
        };
    }
    cp operator *(const double &o)const {
        return (cp) {
            a *o, b *o
        };
    }
    cp operator !()const {
        return (cp) {
            a, -b
        };
    }
} w[maxfft];
int pos[maxfft];
void fft_init(int len) {
    int j = 0;
    while((1 << j) < len)j++;
    j--;
    for(int i = 0; i < len; i++)pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
}
void fft(cp *x, int len, int sta) {
    for(int i = 0; i < len; i++)if(i < pos[i])swap(x[i], x[pos[i]]);
    w[0] = (cp) {
        1, 0
    };
    for(unsigned i = 2; i <= len; i <<= 1) {
        cp g = (cp) {
            cos(2 * pi / i), sin(2 * pi / i)*sta
        };
        for(int j = i >> 1; j >= 0; j -= 2)w[j] = w[j >> 1];
        for(int j = 1; j<i >> 1; j += 2)w[j] = w[j - 1] * g;
        for(int j = 0; j < len; j += i) {
            cp *a = x + j, *b = a + (i >> 1);
            for(int l = 0; l<i >> 1; l++) {
                cp o = b[l] * w[l];
                b[l] = a[l] - o;
                a[l] = a[l] + o;
            }
```

```
        }
    }
    if(sta == -1)for(int i = 0; i < len; i++)x[i].a /= len, x[i].b /= len;
}
/* 普通 FFT - 不带取模 */
cp x[maxfft], y[maxfft], z[maxfft];
void FFT(data_type *a, data_type *b, int n, int m, data_type *c) {
    // 直接调用, n,m 是 a,b 的长度, 下标 0 开始,c 是卷出来的东西
    int i, j, len = 1;
    while(len < (n + m + 1) >> 1)len <<= 1;
    fft_init(len);
    for(i = n / 2; i < len; i++)x[i].a = x[i].b = 0;
    for(i = m / 2; i < len; i++)y[i].a = y[i].b = 0;
    for(i = 0; i < n; i++)(i & 1 ? x[i >> 1].b : x[i >> 1].a) = a[i];
    for(i = 0; i < m; i++)(i & 1 ? y[i >> 1].b : y[i >> 1].a) = b[i];
    fft(x, len, 1), fft(y, len, 1);
    for(i = 0; i < len / 2; i++) {
        j = len - 1 & len - i;
        z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * (w[i] + (cp) {1, 0}) * 0.25;
    }
    for(i = len / 2; i < len; i++) {
        j = len - 1 & len - i;
        z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * ((cp) {
            1, 0
        }
        -w[i ^ len >> 1]) * 0.25;
    }
    fft(z, len, -1);
    for(i = 0; i < n + m; i++)
        if(i & 1)c[i] = (data_type)(z[i >> 1].b + 0.5);
        else c[i] = (data_type)(z[i >> 1].a + 0.5);
}
/* 取模 FFT - 模数较大时, 注意 data_type=int || long long */
cp x[maxfft], y[maxfft], z[maxfft];
int temp[maxfft];
void FFT(data_type *a, data_type *b, int n, int m, data_type *c) {
    // 直接调用, n,m 是 a,b 的长度, 下标 0 开始,c 是卷出来的东西
    if(n <= 100 && m <= 100 || min(n, m) <= 5) {
        for(int i = 0; i < n + m - 1; i++)temp[i] = 0;
        for(int i = 0; i < n; i++)for(int j = 0; j < m; j++) {
                temp[i + j] += (ll)a[i] * b[j] % mod;
                if(temp[i + j] >= mod)temp[i + j] -= mod;
            }
        for(int i = 0; i < n + m - 1; i++)c[i] = temp[i];
```

```
            return;
        }
        int len = 1;
        while(len < n + m + 1)len <<= 1;
        fft_init(len);
        for(int i = 0; i < len; i++) {
            int aa = i < n ? a[i] : 0, bb = i < m ? b[i] : 0;
            x[i] = (cp) {(aa >> 15), (aa & 32767)}, y[i] = (cp) {(bb >> 15),
             ↪  (bb & 32767)};
        }
        fft(x, len, 1), fft(y, len, 1);
        for(int i = 0; i < len; i++) {
            int j = len - 1 & len - i;
            z[i] = ((x[i] + !x[j]) * (y[i] - !y[j]) + (x[i] - !x[j]) * (y[i]
             ↪  + !y[j])) * (cp) {0, -0.25};
        }
        fft(z, len, -1);
        for(int i = 0; i < n + m - 1; i++) {
            ll ta = (ll)(z[i].a + 0.5) % mod;
            ta = (ta << 15) % mod;
            c[i] = ta;
        }
        for(int i = 0; i < len; i++) {
            int j = len - 1 & len - i;
            z[i] = (x[i] - !x[j]) * (y[i] - !y[j]) * (cp) {-0.25, 0}
                   +(x[i] + !x[j])*(y[i] + !y[j])*(cp) {0, 0.25};
        }
        fft(z, len, -1);
        for(int i = 0; i < n + m - 1; i++) {
            ll ta = (ll)(z[i].a + 0.5) % mod, tb = (ll)(z[i].b + 0.5) % mod;
            ta = (ta + (tb << 30)) % mod;
            c[i] = (c[i] + ta) % mod;
        }
    }
```

6.18 多项式

6.18.1 函数操作

```
#define mod 998244353
int inv[maxn];
void init(int n = 100001) {
    inv[1] = 1;
    for(int i = 2; i <= n; i++)inv[i] = mod - (ll)(mod / i) * inv[mod %
     ↪  i] % mod;
}
int temp1[maxfft], temp2[maxfft], temp3[maxfft], temp4[maxfft];
```

```
void Poly_Inv(int *poly, int n, int *ans) { // 多项式求逆元
    ans[0] = inv[poly[0]];
    for(int i = 2; i <= n; i <<= 1) {
        FFT(poly, ans, i, i / 2, temp1);
        FFT(ans, temp1 + i / 2, i / 2, i / 2, temp1);
        for(int j = 0; j < i / 2; j++)ans[j + i / 2] = temp1[j] == 0 ? 0 : mod - temp1[j];
    }
}
void Poly_Log(int *poly, int n, int *ans) { // 多项式取对数
    Poly_Inv(poly, n, temp2);
    for(int i = 0; i < n - 1; i++)ans[i] = (ll)poly[i + 1] * (i + 1) % mod;
    FFT(ans, temp2, n - 1, n, ans);
    for(int i = n - 1; i > 0; i--)ans[i] = (ll)ans[i - 1] * inv[i] % mod;
    ans[0] = 0;
}
void Poly_Exp(int *poly, int n, int *ans) { // 多项式求指数
    if(n == 1) {
        ans[0] = 1;
        return;
    }
    Poly_Exp(poly, n / 2, ans);
    Poly_Log(ans, n, temp3);
    for(int i = 0; i < n; i++) {
        temp3[i] = poly[i] - temp3[i];
        if(temp3[i] < 0)temp3[i] += mod;
    }
    temp3[0]++;
    if(temp3[0] == mod)temp3[0] = 0;
    FFT(ans, temp3, n, n, ans);
    for(int i = n; i < 2 * n; i++)ans[i] = 0;
}
void Poly_Root(int *poly, int n, int *ans) { // 多项式开根号
    ans[0] = 1;
    for(int i = 2; i <= n; i <<= 1) {
        Poly_Inv(ans, i, temp4);
        FFT(ans, ans, i / 2, i / 2, ans);
        for(int j = 0; j < i; j++)ans[j] = (ll)(ans[j] + poly[j]) * inv[2] % mod;
        FFT(ans, temp4, i, i, ans);
        for(int j = i; j < 2 * i; j++)ans[j] = 0;
    }
}
```

6.18.2 生成函数

生成函数: 又称母函数，多用于解决多重集的组合问题，对于序列 $a_0, \cdots, a_n$，令 $G(x) = \sum\limits_{i=0}^{n} a_i x^i$，称 $G(x)$ 为该序列的生成函数。

例 1：有 n 种物品，第 i 种有 a_i 个，问所有物品中取出 m 个物品的方案数，两种方案如果每种物品所取数量相同则视为同一种。

解：令 $P_j(x) = \sum\limits_{i=0}^{a_j} x^i$ 为第 j 种物品所取数量的生成函数，则问题转化为求 $F(x) = \prod\limits_{j=1}^{n} P_j(x)$ 的 x^m 的系数。特别的，如果每件物品只有一个，那么 $P_j(x) = 1 + x, F(x) = (x+1)^n$，答案即为 $C(n, m)$。

总结：把每个集合所取元素个数看作一个序列并求出其生成函数，那么多重集的组合问题变成了多项式乘积，对所取元素个数的限制变成了幂指数的和为定值，进而所求答案即为这些生成函数乘积的某项系数。

例 2：多重背包，有 n 种物品，第 i 种有 a_i 个，第 i 种物品价值为 v_i，问从这些物品中取出价值和为 m 的方案数，两种方案如果每种物品所取数量相同则视为同一种。

解：令 $P_j(x) = \sum\limits_{i=0}^{a_j} x^{i \cdot v_j}$ 为第 j 种物品所取价值的生成函数，则问题转化为求 $F(x) = \prod\limits_{j=1}^{n} P_j(x)$ 的 x^m 的系数。

总结：把每个集合中取若干元素是否可以达到某个价值看作一个序列并求其生成函数，那么多重背包问题转化为求这些生成函数乘积的某项系数，完全背包问题由于价值和有限，故每个物品总数量有限，进而也是多重背包问题。

指数型生成函数: 又称指数型母函数，多用于解决多重集的排列问题。在考虑多重集组合问题时，如果两种方案在每个集合中所选元素个数相同则视为一种方案，但是如果考虑所选元素在排列中的位置（或者说这些元素的顺序）时，之前的方法需要做些变化，假设第 i 种物品选 b_i 个，$\sum\limits_{i=1}^{n} b_i = m$，那么这 m 个物品的全排列为 $\frac{m!}{\prod\limits_{i=1}^{n} b_i!}$，进而构造第 i 种物品的生成函数为 $P_j = \sum\limits_{i=0}^{a_i} \frac{1}{i!} x^i$，那么 $F(x) = \prod\limits_{j=1}^{n} P_j(x)$ 的 x^m 的系数再乘上 $m!$ 即为从这些物品中选出 m 个的排列数。

注意：对于数量无限的情况需要考虑幂级数收敛性，由于生成函数在解决问题中的关键在于每一项的系数和幂指数，和 x 本身取值无关，在绝大部分情况下都可以取到适当的 x 使得幂级数收敛：在第一个例子里让第 j 件物品数量无限，那么其生成函数 $P_j(x) = \sum\limits_{i \geq 0} x^i = \frac{1}{1-x}, |x| < 1$，在第二个例子里加入价值的考量，同样的有 $P_j(x) = \sum\limits_{i \geq 0} x^{i \cdot v_j} = \frac{1}{1-x^{v_j}}, |x| < 1$，在指数型生成函数的例子里，同样的有 $P_j(x) = \sum\limits_{i \geq 0} \frac{1}{i!} x^i = e^x, |x| < 1$。

- **例 1:** 只有 A, B, C, D 四种字母，要求用这四种字母构造一个长度为 n 的序列使得该序列中 A, C 出现次数均为偶数，问方案数 $(1 \leq n \leq 2^{64})$。

  解: 考虑 A, B, C, D 四个字母数量的指数型生成函数 $P_A(x) = P_C(x) = \sum\limits_{i \geq 0} \frac{1}{(2i)!} x^{2i} =$

$\frac{e^x+e^{-x}}{2}$，$P_B(x)=P_D(x)=\sum\limits_{i\geq 0}\frac{1}{i!}x^i=e^x$，故四种字母取若干构成一个排列的生成函数 $F(x)=P_A^2(x)\cdot P_B^2(x)=(\frac{e^x+e^{-x}}{2})^2\cdot e^{2x}=\frac{e^{4x}+2\cdot e^{2x}+1}{4}$ 答案即为 $n!\cdot[x^n]F(x)=\frac{n!}{4}\cdot(\frac{4^n}{n!}+2\cdot\frac{2^n}{n!})=4^{n-1}+2^{n-1}$，其中 $[x^n]F(x)$ 表示 $F(x)$ 的 x^n 项的系数。

- **例 2:** T 组用例，每组用例给出正整数 n,k，问 n 的所有本质不同划分中，每种数字出现次数均小于 k 的方案数 $(1\leq T,n,k\leq 10^5)$。

  解: 考虑划分中 j 的数量的生成函数 $P_j(x)=\sum\limits_{i=0}^{k-1}x^{ij}=\frac{1-x^{kj}}{1-x^j}$。设 $p_k(n)$ 为 n 的本质不同划分中每种数字出现次数小于 k 的方案数，固定 k 后考虑 $p_k(n)$ 的生成函数：$F_k(x)=\prod\limits_{j\geq 1}P_j(x)=\frac{\prod\limits_{j\geq 1}(1-x^{kj})}{\prod\limits_{j\geq 1}(1-x^j)}=\frac{Q(x^k)}{Q(x)}=Q(x^k)P(x)$。即 $F_k(x)=(1-x^k-x^{2k}+x^{5k}+x^{7k}-\cdots)(1+p(1)\cdot x+p(2)\cdot x^2+\cdots)$，考虑等式两边 x^n 的系数有 $p_k(n)=p(n)-p(n-k)-p(n-2k)+p(n-5k)+p(n-7k)-\cdots$，预处理出 $p(1)$ $p(n)$ 后枚举广义五边形数即可求出 $p_k(n)$，时间复杂度 $O(n\sqrt{n}+T\sqrt{n})$。

- **例 3:** 给出一长度为 n 的序列 $a_1,\cdots,a_n$，每次操作等概率的从 1 n 中选一个数 x，把这 n 个数去掉 a_x 后的乘积加到答案里，然后把 a_x 减一，问 k 次操作后答案的期望值，结果模 10^9+7，$(1\leq n\leq 5000, 1\leq k\leq 10^9, 0\leq a_i\leq 10^9)$。

  解: 假设某次操作前这 n 个数字变成 a_i-b_i，b_i 为 i 被选中的次数。当前操作选中了 x，那么该次操作的贡献为 $\prod\limits_{i\neq x}(a_i-b_i)=\prod\limits_{i=1}^{n}(a_i-b_i)-(a_i-(b_i+1))\prod\limits_{i\neq x}(a_i-b_i)$。不妨设 $c_i=b_i, i\neq x, c_x=b_x+1$ 为该次操作结束后 i 被选中的次数，那么本次操作的贡献可以写成：$\prod\limits_{i=1}^{n}(a_i-b_i)-\prod\limits_{i=1}^{n}(a_i-c_i)$，计算该交错和得总贡献为 $\prod\limits_{i=1}^{n}a_i-\prod\limits_{i=1}^{n}(a_i-b_i)$，其中 b_i 为 k 次操作后 i 被选中的次数，$\sum\limits_{i=1}^{n}b_i=k$。故答案的期望值为 $\frac{1}{n^k}\sum_{\sum\limits_{i=1}^{n}b_i=k}\frac{k!}{\prod\limits_{i=1}^{n}b_i!}(\prod\limits_{i=1}^{n}a_i-\prod\limits_{i=1}^{n}(a_i-b_i))=\prod\limits_{i=1}^{n}a_i-\frac{k!}{n^k}\sum_{\sum\limits_{i=1}^{n}b_i=k}\prod\limits_{i=1}^{n}\frac{a_i-b_i}{b_i!}$，令 $P_j(x)$ 为上述求和式中第 j 项对答案的贡献，则有 $P_j(x)=\sum\limits_{i\geq 0}\frac{a_j-i}{i!}x^i, 1\leq j\leq n$，化简得 $P_j(x)=a_j\sum\limits_{i\geq 0}\frac{1}{i!}x^i-x\sum\limits_{i\geq 1}\frac{1}{(i-1)!}x^{i-1}=(a_j-x)\cdot e^x$。记 $f(k)=\sum_{\sum\limits_{i=1}^{n}b_i=k}\prod\limits_{i=1}^{n}\frac{a_i-b_i}{b_i!}$，考虑其生成函数 $F(x)=\sum\limits_{i\geq 0}f(i)\cdot x^i$，有 $F(x)=\prod\limits_{j=1}^{n}P_j(x)=e^{nx}\cdot\prod\limits_{j=1}^{n}(a_j-x)=e^{nx}\cdot G(x)$。$O(n^2)$ 直接计算得 $G(x)=\sum\limits_{i=0}^{n}g_i\cdot x^i$，进而有：$f(k)=[x^k]F(x)=\sum\limits_{i=0}^{min(n,k)}[x^i]G(x)\cdot[x^{k-i}]e^{nx}=\sum\limits_{i=0}^{min(n,k)}g_i\cdot\frac{n^{k-i}}{(k-i)!}$，故答案为 $\prod\limits_{i=1}^{n}a_i-\frac{k!}{n^k}f(k)=\prod\limits_{i=1}^{n}a_i-\sum\limits_{i=0}^{min(n,k)}g_i\cdot\frac{\prod\limits_{j=k-i+1}^{k}j}{n^i}$。

- **例 4:** 给出一个体积为 $2n$ 的背包，有 n 种食物，第 i 种食物体积为 i，数量为 a_i，有 m 种装备，只能带一件，第 i 件体积为 b_i，问装满背包的方案数，$(1\leq n\leq 5\cdot 10^4, 1\leq m\leq 2n, 0\leq a_1<\cdots<a_n\leq 2n, 1\leq b_i\leq 2n)$。

  解: 设 $f(k)$ 为所带食物总体积为 k 的方案数，考虑 $f(k)$ 的生成函数 $F(x)=\sum\limits_{k\geq 0}f(k)\cdot x^k(mod\ x^{2n})$，其中模 x^{2n} 表示只取该多项式幂指数小于 $2n$ 的项，考虑

第 j 种食物对体积贡献的生成函数 $P_j(x)=\sum\limits_{i=0}^{a_i}x^{ij}=\frac{1-x^{(a_j+1)j}}{1-x^j}$。进而有 $F(x)=\prod\limits_{j=1}^{n}P_j(x)=\prod\limits_{i=1}^{n}(1-x^{(a_i+1)i})\prod\limits_{i=1}^{n}\frac{1}{1-x^i}(mod\ x^{2n})$，$\prod\limits_{i=1}^{n}\frac{1}{1-x^i}=\prod\limits_{i=1}^{2n-1}\frac{1}{1-x^i}\prod\limits_{i=n+1}^{2n-1}(1-x^i)=P(x)\prod\limits_{i=n+1}^{2n-1}(1-x^i)(mod\ x^{2n})$，$O(n\sqrt{n})$ 复杂度可以求出 $p(1),\cdots,p(2n-1)$ 进而得到 $P(x)$。对于多项式 $P(x)=\sum\limits_{i=0}^{2n-1}p_i\cdot x^i$，对其乘上 $1-x^{n+j},1\le j\le n-1$ 得: $\sum\limits_{i=0}^{2n-1}p_i\cdot x^i-\sum\limits_{i=0}^{n-j}p_i\cdot x^{n+i+j}=\sum\limits_{i=0}^{n+j-1}p_i\cdot x^i+\sum\limits_{i=n+j}^{2n-1}(p_i-p_{i-(n+j)})\cdot x^i(mod\ x^{2n})$。即从 p_{n+j} 中减掉 p_j，那么 $P(x)$ 乘上 $\prod\limits_{i=n+1}^{2n-1}(1-x^i)$ 后，$p_0,\cdots,p_n$ 不变，p_{n+i} 中减掉了 $\sum\limits_{j=0}^{i}p_j,1\le i\le n-1$，维护前 n 项 p_i 的前缀和依次从后 n 项中减去即可，时间复杂度 $O(n)$。注意到 $0\le a_1<\cdots<a_n\le 2n$，故 $a_i+1\ge i$，进而 $(a_i+1)i\ge i^2$。这表明 $\prod\limits_{i=1}^{n}(1-x^{(a_i+1)i})$ 在模 x^{2n} 意义下至多只有前 $\sqrt{n}$ 项有意义，这部分 $O(n\sqrt{n})$ 直接计算即可。求出 $f(0)\ f(2n-1)$ 后，枚举带第 i 件装备可得总方案数为 $\sum\limits_{i=1}^{m}f(2n-b_i)$。

- **例 5:** 给出两个长度不超过 10^5 的数字串 A,B，计算 $A\cdot B$。

  解: 记 $A=a_{n-1}\cdots a_0,B=b_{m-1}\cdots b_0$，考虑每一位贡献的生成函数 $A(x)=\sum\limits_{i=0}^{n-1}a_i\cdot x^i,B(x)=\sum\limits_{i=0}^{m-1}b_i\cdot x^i$，则有 $A=A(10),B=B(10)$，令 $C(x)=A(x)\cdot B(x)=\sum\limits_{i=0}^{n+m-2}c_i\cdot x^i$，其中 $c_i=\sum\limits_{j=0}^{i}a_j\cdot b_{i-j}$，用 FFT 得到 c 序列后，$A\cdot B=C(10)=c_{n+m-2}\cdots c_0$。

- **例 6:** 给出一长度为 n 的串 S 和一长度为 m 的串 T，两个串均只由 $'A','G','C','T'$ 组成，定义 T 和 S 在 i 位置匹配为对于任意 $1\le j\le m$，存在 $p(1\le p\le n)$ 使得 $|(i+j-1)-p|\le k$ 且 $S[p]=T[j]$，问 T 可以在多少位置和 S 匹配，$(1\le m\le n\le 2\times 10^5,0\le k\le 2\times 10^5)$。

  解: 对四种字符单独计算固定起点的匹配长度，如果四个匹配长度加一起是 m 说明可以匹配，以下只考虑一种字符。用一个 01 序列 a 表示 S 串这个字符的存在情况，$a[i]=1$ 表示 S 串的 $[i-k,i+k]$ 段出现过该字符，统计该字符出现次数的前缀和，做差即可得到一个区间该字符数量，继而可以 $O(n)$ 得到 a 序列。再用一个 01 序列 b 表示 T 串的反串中该字符的存在情况，$b[i]=1$ 表示 $T[m-i]$ 是该字符。那么 T 从 S 的第 i 个位置开始往前匹配的该种字符数量为 $c[i]=\sum\limits_{j=0}^{i}a[j]\cdot b[i-j]$，用 FFT 加速该卷积即可。为减少 DFT 次数，可以先把处理四种字符时的 a,b 序列 DFT 后结果乘积先加一起，最后做一次 $IDFT$ 即可。

- **例 7:** 给出一个只由 $'V','K','?'$ 组成的字符串，$'?'$ 可以变成 $'V'$ 或 $'K'$，问该字符串可能的循环节，一个长度为 n 的字符串 s 以 d 为循环节的定义是 $s[i]=s[i+d],0\le i<n-d$，$(1\le n\le 5\cdot 10^5)$。

解: 首先考虑没有问号的字符串，如果 s 以 d 为循环节，则对于 $0 \le i < n-d$ 都有 $s[i]=s[i+d]$。令 ss 为 s 的反串，则有 $s[i]=ss[n-1-i]$，故之前的等式可以变成 $s[i]=ss[n-1-i-d]$。用一个序列 f 记录 s 中 V 的位置，用另一个序列 g 记录 s 的反串 ss 中 V 的位置。那么 $f*g(n-1-d)$ 的值即表示满足 $s[i]=ss[n-1-d-i]='V'$ 的 i 个数，同理再记录 K 的位置可以得到满足 $s[i]=ss[n-1-d-i]='K'$ 的 i 个数。这俩加起来等于 $n-d$ 说明该字符串以 d 为循环节，现在考虑有问号的字符串，由于问号位置不确定，我们反向考虑字符串 s 不以 d 为循环节的情况，用 f 记录 s 中 V 的位置，g 记录 ss 中 K 的位置，则 $f*g(n-1-d)$ 表示 $s[i]\neq ss[n-1-i-d]$ 的个数，该值非零说明该字符串不可能以 d 为循环节。注意到该值为 0 不等价于 s 可以以 d 为循环节，可能会出现 $s[i]\neq s[i+2d]$ 但 $s[i+d]$ 是问号的情况，但是这种情况会在判断 s 是否以 $2d$ 为循环节的时候判出来。即如果 s 不以 d 为循环节，那么 s 不以 d 的所有正因子为循环节。故在卷积得到该序列后，对于不成立的循环节，给其因子也标为不成立即可。

- **例 8:** 用 $[1,2^k-1]$ 之间的整数构造一个长度为 n 的序列 $a_1,\cdots,a_n$，令 $b_i = a_1|a_2|\cdots|a_i, 1\le i\le n$，问使得 b 序列严格递增的方案数，$(1\le n\le 10^{18}, 1\le k\le 30000)$。

  解: 为保证 b 序列严格递增，每次 a_i 需要至少在一个 a_1 a_{i-1} 都是 0 的位上是 1，故序列长度至多为 k。即如 $n>k$ 则无解，下面只考虑 $n\le k$ 的情况。设 $dp[i][j]$ 为前 i 个数占据 j 个二进制位的方案数，不考虑这些位在 k 个位中的位置，则答案应为 $\sum\limits_{i=1}^{k} C_k^i\cdot dp[n][i]$。假设求 $dp[x+y][i]$，前 x 个数已经占据了 i 位中的 j 位，方案数 $C_i^j\cdot dp[x][j]$，而后 y 个数要把剩下的 $i-j$ 位占据，而在前 x 个数占据的 j 位上可以随意取值，方案数 $dp[y][i-j]\cdot 2^{yj}$。故有 $dp[x+y][i]=\sum\limits_{j=0}^{i} dp[x][j]\cdot dp[y][i-j]\cdot 2^{yj}\cdot C_i^j$。在求 $dp[n][i]$ 时，如果 n 是偶数，那么有 $\frac{dp[n][i]}{i!}=\sum\limits_{j=0}^{i}\frac{dp[\frac{n}{2}][j]\cdot 2^{\frac{n}{2}j}}{j!}\cdot\frac{dp[\frac{n}{2}][i-j]}{(i-j)!}$。如果 n 是奇数，可以先用上面的方法求出 $dp[n-1]$ 的值，然后 $\frac{dp[n][i]}{i!}=\sum\limits_{j=0}^{i}\frac{dp[n-1][j]\cdot 2^j}{j!}\cdot\frac{1}{(i-j)!}$，故可以快速幂求 $dp[n]$，每次两个序列卷积得到一个新的序列，FFT 即可，时间复杂度 $O(klog_2^2k)$。

- **例 9:** n 种物品，第 i 种物品价值为 a_i，每种物品数量无限，拿 k 件物品，问可能的价值和，$(1\le n,k,a_i\le 1000)$。

  解: 拿一件物品的价值的生成函数为 $f(x)=\sum\limits_{i=1}^{n}x^{a_i}$，拿 k 件物品的价值和的生成函数即为 $g(x)=f^k(x)$，快速幂套 FFT 即可，注意由于只需要问某个价值和是否存在，故在卷积过程中将非零值全部存为 1 即可。

- **例 10:** 给出 n 个不同的数 $a_1,\cdots,a_n$，现在要求从这 n 个数中选出最少的数字，使得其满足每一个 a_i 都可以通过从中选取任意数字（每种数字可以选任意个）组成，且从中取任意数字，只要其和不超过 m，那么其和必然在之前的 n 个数里出现过，$(1\le n,m\le 10^6, 1\le a_1<a_2<\cdots<a_n\le m)$。

  解: 显然如果有解那么 a_i 的任意整数倍（值不超过 m）都在 $A=\{a_1,a_2,\cdots,a_n\}$ 中，这个不满足则无解，如果一个不超过 m 的数字可以被 A 中一些数字线性组合

得到，即 $c \le m, c = \sum\limits_{i=1}^{n} x_i a_i, x_i \ge 0$。由于 $x_i > 0, x_i a_i \le m$ 时有 $x_i a_i \in A$，故 c 必然可以是 A 中一些数字的和。且由条件，几个数字的和也在 A 中，故 c 必然可以可以表示成 A 中两个数的和，进而判断是否有解只需要知道 A 是否对加法封闭即可。令 $f(x) = 1 + \sum\limits_{i=1}^{n} x^{a_i}$，令 $g(x) = f^2(x)$，对于 $1 \le c \le m$，则 $[x^c]g(x)$ 含义即为从 $A \cup \{0\}$ 选出两个数字 a, b 满足 $a + b = c$ 的方案数 $\cdot 2$。如果 $[x^c]g(x) \ne 0$ 且 $[x^c]f(x) = 0$，说明运算不封闭，无解；如果 $[x^c]g(x) = 0$ 则不用考虑 c。故只需考虑 $[x^c]g(x) \ne 0$ 且 $[x^c]f(x) \ne 0$ 的情况来找出所选数字最少个数。如果 $[x^c]g(x) = 2$，在尚有解的情况下，c 的表示只能是 $0 + c$，故 c 必须选，否则 c 无法被表示。如果 $[x^c]g(x) > 2$，说明存在 $a, b \in A$ 满足 $a + b = c$，先前所选数字集已经可以表示出 a, b，c 不用选。

- **例 11:** 给出一个含有 n 个互异正整数的序列 $c[1], c[2], \cdots, c[n]$。一棵带点权的有根二叉树满足其所有顶点的权值都在集合 $\{c[1], c[2], \cdots, c[n]\}$ 中就是合法的，其中一棵带点权树的权值是其所有顶点权值的总和。给出一个整数 m，对于任意的 $s(1 \le s \le m)$ 计算出权值为 s 的合法二叉树个数，结果模 998244353，$(1 \le n \le 10^5, 1 \le m \le 10^5, 1 \le c[i] \le 10^5)$。

  解: 设每个点权值的生成函数 $A(x) = \sum\limits_{i=1}^{n} x^{c[i]}$，树的生成函数为 $F(x)$，加上常数项 1 表示空树的情况。考虑是否为空树有方程 $F(x) = A(x)F^2(x) + 1$，解此二次方程得 $F(x) = \frac{1 \pm \sqrt{1-4A(x)}}{2A(x)}$。注意到 $1 - 4A(x)$ 开根常数项为 1，而 $A(x)$ 中没有常数项，故只有将常数项消掉才可以做除法。即解为 $F(x) = \frac{1-\sqrt{1-4A(x)}}{2A(x)}$，做多项式开根和逆元即可。

- **例 12:** 给出一个含有 n 个互异正整数的序列 $c[1], c[2], \cdots, c[n]$，记集合 $C = \{c[1], \cdots, c[n]\}$。对于一棵带权多叉树，定义叶子节点点权为 1，非叶子节点点权为其儿子节点点权之和，如果对于任一点权大于 1 的结点 u，u 的孩子数目属于集合 C，则称该多叉树是合法的。给出一个整数 s，求根节点权值为 s 的合法多叉树的个数，结果模 950009857，$(1 \le n, s, c[i] \le 10^5)$。

  解: 设满足条件的树的生成函数为 $F(x)$，则有 $F(x) = x + \sum\limits_{i=1}^{n} (F(x))^{c[i]}$，不考虑空树的情况。设 $A(x) = x - \sum\limits_{i=1}^{n} x^{c[i]}$，则 $A(F(x)) = x$，由拉格朗日反演，$[x^n]F(x) = \frac{1}{n}[x^{n-1}](\frac{x}{A(x)})^n$。而 $(\frac{x}{A(x)})^n = e^{-nln(\frac{A(x)}{x})}$，故只需多项式 $\frac{A(x)}{x}$ 取对数后乘上 $-n$ 然后取指数即可。

- **例 13:** 求 $f(n) = \sum\limits_{i=0}^{n} \sum\limits_{j=0}^{n} S(i,j) \cdot 2^j \cdot j!$，其中 $S(i,j)$ 表示第二类斯特林数，递推公式为:$S(i,j) = j \cdot S(i-1,j) + S(i-1,j-1), 1 \le j \le i-1$，$S(i,i) = 1(i \ge 0), S(i,0) = 0(i \ge 1)$，$(1 \le n \le 100000)$。

  解: 令 $g(i) = \sum\limits_{j=0}^{n} S(i,j) \cdot 2^j \cdot j!$，则答案为 $ans = \sum\limits_{i=0}^{n} g(i)$。由第二类斯特林数定义知 $g(i)$ 的意义是将 i 个数分到 j 个不同的非空集合，且每多一集合对答案的贡献就乘 2。考虑第一个集合中元素个数为 k，则有 $g(i) = \sum\limits_{k=1}^{i} 2C_i^k g(i-k)$，

化简得到卷积形式 $\frac{g(i)}{i!} = \sum\limits_{j=1}^{i} \frac{g(i-j)}{(i-j)!}\frac{2}{j!}$。令 $F(x) = \sum\limits_{i=0}^{\infty} \frac{g(i)}{i!}x^i, G(x) = \sum\limits_{i=1}^{\infty} \frac{2}{i!}x^i$，则有 $F(x) = F(x)G(x) + 1$，故有 $F(x) = \frac{1}{1-G(x)}$，多项式求逆得到 $F(x)$，进而得到 $g(0), \cdots, g(n)$。

- **例 14:** 找出 n 个点的二元关系中满足对称性和传递性但不满足自反性的个数，$(1 \le n \le 4000)$。

  解: 将 n 个点及它们之间的二元关系看作一张无向图。问题转化为求没有自环且满足传递性的无向图个数，考虑每个连通块：规模为 1 的连通块有两种：单点集和自环，规模超过 1 的连通块只有一种，即完全图。先不考虑点的编号，则一个连通块的生成函数为 $A(x) = 2 \cdot \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = e^x + x - 1$，若干连通块组成的图的指数型生成函数即为 $e^{A(x)} = e^{e^x+x-1}$。若干不是自环的连通块组成的图的生成函数即为 $A(x) - x$，故其指数型生成函数为 e^{e^x-1}。进而答案的生成函数为 $B(x) = e^{e^x+x-1} - e^{e^x-1} = (e^x - 1) \cdot e^{e^x-1} = f(x) \cdot e^{f(x)}$。故只要对 $f(x)$ 求一下多项式 exp 然后和自身乘一下即可，此时考虑节点编号，答案即为 $n! \cdot [x^n]B(x)$。

6.18.3 拉格朗日插值

```cpp
#include <bits/stdc++.h>
using namespace std;
struct DATA {
    double x;
    double y;
} ;
vector <DATA> v;
double func() { //把数据 push_back 到 v 里
    return;
}
double ans[6];
void lagrang() {
    for(int i = 1; i <= 5; i++)ans[i] = 0;
    for(int i = 0; i < v.size(); i++) {
        double now = v[i].y;
        for(int j = 0; j < v.size(); j++) {
            if(j == i)continue;
            else now = now / (v[i].x - v[j].x);
        }
        double tmp[6] = {0};
        tmp[0] = 1;
        for(int j = 0; j < v.size(); j++) {
            if(j == i)continue;
            for(int k = 5; k >= 1; k--)tmp[k] = tmp[k - 1] - tmp[k] *
            ↪ v[j].x;
            tmp[0] = - tmp[0] * v[j].x;
        }
        for(int k = 0; k <= 5; k++)ans[k] += tmp[k] * now;
```

```
    }
    int mx = 5;
    while(fabs(ans[mx]) <= 1e-7)mx--;
    for(int i = mx; i >= 0; i--) {
        if(i)printf("%.2lfx^%d+", ans[i], i );
        else printf("%.2lf\n", ans[i]);
    }
}
int main() {
    func(); // 获得数据
    lagrang(); // 插值输出多项式
    return 0;
}
```

6.19 计算 N 阶线性齐次递推式第 K 项

```
/* 例题：(徐州邀请赛) 计算 N 阶线性齐次递推式第 K 项
 * 解法：BM 算法
 */
#include <bits/stdc++.h>
using namespace std;
const int LOG = 31, MOD = 1000000007;
typedef long long LL;
int myrand() {
    return ((LL)rand() << 32 ^ (LL)rand() << 16 ^ rand()) % 1000000000LL;
}
void data_maker() {
    srand(time(0));
    freopen("H.in", "w", stdout);
    //printf("1 1000000000\n1000000000\n1000000000\n");
    int n = 3, m;
    for (int Case = 1; Case <= 4; Case++) {
        m = 5;
        printf("%d %d\n", n, m);
        for (int i = 1; i <= n; i++)
            if (i < n) printf("%d ", myrand() % 5 + 1);
            else printf("%d\n", myrand() % 5 + 1);
        for (int i = 1; i <= n; i++)
            if (i < n) printf("%d ", myrand() % 5 + 1);
            else printf("%d\n", myrand() % 5 + 1);
    }
    fclose(stdout);
}
struct LinearRec {
    int n;
    vector<int> first, trans;
```

```
    vector<vector<int> > bin;

    vector<int> add(vector<int> &a, vector<int> &b) {
        vector<int> result(n * 2 + 1, 0);
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= n; ++j) {
                if ((result[i + j] += (long long)a[i] * b[j] % MOD) >= MOD) {
                    result[i + j] -= MOD;
                }
            }
        }
        for (int i = 2 * n; i > n; --i) {
            for (int j = 0; j < n; ++j) {
                if ((result[i - 1 - j] += (long long)result[i] * trans[j] % MOD) >= MOD) {
                    result[i - 1 - j] -= MOD;
                }
            }
            result[i] = 0;
        }
        result.erase(result.begin() + n + 1, result.end());
        return result;
    }

    LinearRec(vector<int> &first, vector<int> &trans): first(first), trans(trans) {
        n = first.size();
        vector<int> a(n + 1, 0);
        a[1] = 1;
        bin.push_back(a);
        for (int i = 1; i < LOG; ++i) {
            bin.push_back(add(bin[i - 1], bin[i - 1]));
        }
    }

    int calc(int k) {
        vector<int> a(n + 1, 0);
        a[0] = 1;
        for (int i = 0; i < LOG; ++i) {
            if (k >> i & 1) {
                a = add(a, bin[i]);
            }
        }
        int ret = 0;
        for (int i = 0; i < n; ++i) {
```

```
                if ((ret += (long long)a[i + 1] * first[i] % MOD) >= MOD) {
                    ret -= MOD;
                }
            }
            return ret;
        }
};
int main() {
    int n, k;
    while (scanf("%d%d", &n, &k) != EOF) {
        vector<int> a(n), b(n);
        for (int i = 0; i < n; i++) scanf("%d", &a[i]); //,c[i+1]=a[i];
        for (int i = 0; i < n; i++) scanf("%d", &b[i]);
        LinearRec f(a, b);
        printf("%d\n", f.calc(k));
    }
    return 0;
}
```

6.20 单纯形（解决一类线性规划问题）

```
/* 功能：
 * 接受有 n 个约束，m 个基本变量的方程组 a[0~n][0~m]
 * a[0][] 存放需要最大化的目标函数，a[][0] 存放常数
 * Base[] 存放基本变量的 id，初始为 1~m
 * Rest[] 存放松弛变量的 id，初始为 m+1~m+n
 * 返回此线性规划的最小值 ans
 * 要求方案的话，Base[] 中的变量值为 0,Rest[] 中的变量值为相应行的 [0]
 * 如果 solve
    + 返回 1，说明运行正常 ans 是它的最大值
    + 返回 0，说明无可行解
    + 返回-1，说明解没有最大值
 * 测试：
 *      m=2,n=3
 *      double a[4][3]={
 *          {0,1,3},
 *          {8,-1,1},
 *          {-3,1,1},
 *          {2,1,-4}
 *      };
 *      solve=1,ans=64/3;
 *  注意 ac 不了可能是 eps 的问题
 */
const int Maxn = 110, Maxm = 59;
class Simplex {
public:
```

```
    static const double Inf;
    static const double eps;
    int n, m;
    double a[Maxn][Maxm];
    int Base[Maxm], Rest[Maxn];
    double val[Maxm];
    double ans;
    void pt() {
        for(int i = 0; i <= n; i++) {
            for(int j = 0; j <= m; j++)printf("%.2f ", a[i][j]);
            puts("");
        }
    }
    void pivot(int x, int y) { //将第 x 个非基本变量和第 y 个基本变量调换
        swap(Rest[x], Base[y]);
        double tmp = -1. / a[x][y];
        a[x][y] = -1.;
        for(int j = 0; j <= m; j++)a[x][j] *= tmp;
        for(int i = 0; i <= n; i++) {
            if(i == x || fabs(a[i][y]) < eps)continue;
            tmp = a[i][y];
            a[i][y] = 0;
            for(int j = 0; j <= m; j++)a[i][j] += tmp * a[x][j];
        }
    }
    bool opt() {
        while(1) {
            int csi = 0;
            for(int i = 1; i <= m; i++)if(a[0][i] > eps && (!csi || Base[i] < Base[csi]))csi = i;
            if(!csi)break;
            int csj = 0;
            double cur;
            for(int j = 1; j <= n; j++) {
                if(a[j][csi] > -eps)continue;
                double tmp = -a[j][0] / a[j][csi];
                if(!csj || tmp + eps < cur || (fabs(tmp - cur) < eps && Rest[j] < Rest[csj]))csj = j, cur = tmp;
            }
            if(!csj)return 0;
            pivot(csj, csi);
        }
        ans = a[0][0];
        return 1;
    }
    bool init() {
```

```
    ans = 0;
    for(int i = 1; i <= m; i++)Base[i] = i;
    for(int i = 1; i <= n; i++)Rest[i] = m + i;
    int cs = 1;
    for(int i = 2; i <= n; i++)if(a[i][0] < a[cs][0])cs = i;
    if(a[cs][0] >= -eps)return 1;
    static double tmp[Maxm];
    for(int i = 0; i <= m; i++)tmp[i] = a[0][i], a[0][i] = 0;
    for(int i = 1; i <= n; i++)a[i][m + 1] = 1.;
    a[0][m + 1] = -1.;
    Base[m + 1] = m + n + 1;
    pivot(cs, ++m);
    opt();
    m--;
    if(a[0][0] < -eps)return 0;
    cs = -1;
    for(int i = 1; i <= n; i++) {
        if(Rest[i] > m + n) {
            cs = i;
            break;
        }
    }
    if(cs >= 1) {
        int nxt = -1;
        m++;
        for(int i = 1; i <= m; i++)if(a[cs][i] > eps || a[cs][i] <
         ↪  -eps) {
                nxt = i;
                break;
            }
        pivot(cs, nxt);
        m--;
    }
    for(int i = 1; i <= m; i++) {
        if(Base[i] > m + n) {
            swap(Base[i], Base[m + 1]);
            for(int j = 0; j <= n; j++)a[j][i] = a[j][m + 1];
            break;
        }
    }
    for(int i = 1; i <= m; i++)a[0][i] = 0;
    a[0][0] = tmp[0];
    for(int i = 1; i <= m; i++)if(Base[i] <= m)a[0][i] =
 ↪  tmp[Base[i]];
    for(int i = 1; i <= n; i++) {
        if(Rest[i] <= m) {
```

```
                for(int j = 0; j <= m; j++)a[0][j] += tmp[Rest[i]] *
                 ↪ a[i][j];
            }
        }
        return 1;
    }
    void getval() {
        for(int i = 1; i <= m; i++)val[i] = 0;
        for(int i = 1; i <= n; i++)if(Rest[i] <= m)val[Rest[i]] =
         ↪ a[i][0];
        //for(int i=1;i<=m;i++)printf("%.2f ",val[i]);puts("");
    }
    int solve() {
        if(!init())return 0;
        if(!opt())return -1;
        getval();
        return 1;
    }
} solver;
const double Simplex:: Inf = 1e80;
const double Simplex:: eps = 1e-8;
int main() {
    int m, n, type;
    scanf("%d%d%d", &m, &n, &type);
    solver.a[0][0] = 0;
    for(int i = 1; i <= m; i++) scanf("%lf", &solver.a[0][i]);
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m + 1; j++) {
            if(j == m + 1) scanf("%lf", &solver.a[i][0]);
            else {
                scanf("%lf", &solver.a[i][j]);
                solver.a[i][j] = -solver.a[i][j];
            }
        }
    }
    solver.m = m, solver.n = n;
    int rep = solver.solve();
    if(rep == 0) puts("Infeasible");
    else if(rep == -1) puts("Unbounded");
    else {
        printf("%.12f\n", solver.ans);
        if(type == 1) {
            for(int i = 1; i <= m; i++)printf("%.12f%c", solver.val[i], i
             ↪ == m ? '\n' : ' ');
        }
    }
```

```
    return 0;
}
```

6.21 快速数论变换 (FWT)

```
#define ll long long
#define mod 1000000007

ll mod_pow(ll a, ll n, ll mo) {
    ll ret = 1;
    while (n) {
        if (n & 1) ret = ret * a % mo;
        a = a * a % mo;
        n >>= 1;
    }
    return ret;
}

ll inv2 = mod_pow(2, mod - 2, mod);

void fwt_xor(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_xor(a, l, mid);
    fwt_xor(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        ll x1 = a[i];
        ll x2 = a[i + len];
        a[i] = (x1 + x2) % mod;
        a[i + len] = (x1 - x2 + mod) % mod;
    }
}

void ifwt_xor(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        // y1=x1+x2
        // y2=x1-x2
        ll y1 = a[i];
        ll y2 = a[i + len];
        a[i] = (y1 + y2) * inv2 % mod;
        a[i + len] = ((y1 - y2 + mod) % mod * inv2) % mod;
    }
```

```
    ifwt_xor(a, l, mid);
    ifwt_xor(a, mid + 1, r);
}

void fwt_and(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_and(a, l, mid);
    fwt_and(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        ll x1 = a[i];
        ll x2 = a[i + len];
        a[i] = (x1 + x2) % mod;
        a[i + len] = x2 % mod;
    }
}

void ifwt_and(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        // y1=x1+x2
        // y2=x2
        ll y1 = a[i];
        ll y2 = a[i + len];
        a[i] = (y1 - y2 + mod ) % mod;
        a[i + len] = y2 % mod;
    }
    ifwt_and(a, l, mid);
    ifwt_and(a, mid + 1, r);
}

void fwt_or(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_or(a, l, mid);
    fwt_or(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        ll x1 = a[i];
        ll x2 = a[i + len];
        a[i] = x1 % mod;
        a[i + len] = (x2 + x1) % mod;
    }
```

```
}

void ifwt_or(ll a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        // y1=x1
        // y2=x2+x1
        ll y1 = a[i];
        ll y2 = a[i + len];
        a[i] = y1 % mod;
        a[i + len] = (y2 - y1 + mod) % mod;
    }
    ifwt_or(a, l, mid);
    ifwt_or(a, mid + 1, r);
}

const int maxn = 1024;
bool test_xor() {
    ll a1[maxn], a2[maxn];
    ll b1[maxn], b2[maxn];
    for (int i = 0; i < maxn; ++i) {
        a1[i] = a2[i] = rand();
        b1[i] = b2[i] = rand();
    }
    ll c1[maxn];
    ll c2[maxn];
    memset(c1, 0, sizeof(c1));
    memset(c2, 0, sizeof(c2));
    for (int i = 0; i < maxn; ++i) {
        for (int j = 0; j < maxn; ++j) {
            c1[i ^ j] += (a1[i] * b1[j]) % mod;
            c1[i ^ j] %= mod;
        }
    }
    fwt_xor(a2, 0, maxn - 1);
    fwt_xor(b2, 0, maxn - 1);
    for (int i = 0; i < maxn; ++i) {
        c2[i] = a2[i] * b2[i] % mod;
    }
    ifwt_xor(c2, 0, maxn - 1);
    for (int i = 0; i < maxn; ++i) {
        if (c1[i] != c2[i]) {
            return false;
        }
```

```
    }
    return true;
}
```

6.22 线性基

```
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
class XORbase {
#define HIGH 62
public:
    ll b[HIGH], nb[HIGH]
    int tot;

    void Init() {
        memset(b, 0, sizeof b);
        memset(nb, 0, sizeof nb);
    }

    void Insert(ll x) {
        per(i, 0, HIGH) {
            if (x & (1LL << i)) {
                if (!b[i]) {
                    b[i] = x;
                    break;
                } else x ^= b[i];
            }
        }
    }

    void Init(ll x) {
        memset(b, 0, sizeof b);
        Insert(x);
        tot = 0;
    }

    ll Query_max(ll x = 0) {
        ll res = x;
        per(i, 0, HIGH) res = max(res, res ^ b[i]);
        return res;
    }

    ll Query_min(ll x = 0) {
        ll res = x;
        rep(i, 0, HIGH) {
            if (b[i]) res ^= b[i];
```

```
        }
        return res;
    }

    void Rebuild() {
        tot = 0;
        per(i, 0, HIGH) {
            per(j, 0, i) {
                if (b[i] & (1LL << j)) b[i] ^= b[j];
            }
        }
        rep(i, 0, HIGH) {
            if (b[i]) nb[tot++] = b[i];
        }
    }

    ll Query_Kth(ll k) { // 需要先 rebuild
        ll res = 0;
        per(i, 0, HIGH) {
            if (k & (1LL << i)) res ^= nb[i];
        }
        return res;
    }

    friend XORbase operator+ (const XORbase &x, const XORbase &y) { // 两
    ↪   个线性向量合并
        XORbase res = x;
        per(i, 0, HIGH) if (y.b[i]) res.Insert(y.b[i]);
        return res;
    }
#undef HIGH
};

/*
 * 例题 1：K 大异或和
 * 解法：那么就从高位到低位贪心，每碰到一位，看看需不需要异或上它就好了。
 *      其中若这 n 个数都是线性无关的，那么 k 要加 1，因为不能表示出 0。
 */
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define Close() ios::sync_with_stdio(0),cin.tie(0)
typedef long long ll;
/* head */
class XORbase {
```

```
#define HIGH 64
public:
    ll b[HIGH];
    int sz, n;

    void Init() {
        memset(b, 0, sizeof b);
    }

    void Insert(ll x) {
        per(i, 0, HIGH) {
            if (x & (1LL << i)) {
                if (!b[i]) {
                    b[i] = x;
                    break;
                } else x ^= b[i];
            }
        }
    }

    void Init(ll x) {
        memset(b, 0, sizeof b);
        Insert(x);
    }

    void Init(int _n) {
        n = _n;
        sz = 0;
        rep(i, 0, HIGH) {
            rep(j, i + 1, HIGH) {
                if ((b[j] >> i) & 1) b[j] ^= b[i];
            }
        }
        rep(i, 0, HIGH) if (b[i]) b[sz++] = b[i];
    }

    ll Calc(ll k) {
        if (sz != n) k--;
        if (k > (1LL << sz) - 1) return -1;
        ll res = 0;
        rep(i, 0, HIGH) if (k & (1LL << i)) res ^= b[i];
        return res;
    }

    friend XORbase operator+ (const XORbase &x, const XORbase &y) {
        XORbase res = x;
```

```
        per(i, 0, HIGH) if (y.b[i]) res.Insert(y.b[i]);
        return res;
    }
#undef HIGH
} e;
int main() {
    Close();
    int n, q;
    ll k;
    cin >> n;
    e.Init();
    rep(i, 1, n + 1) {
        ll x;
        cin >> x;
        e.Insert(x);
    }
    e.Init(n);
    cin >> q;
    while (q--) {
        cin >> k;
        cout << e.Calc(k) << endl;
    }
    return 0;
}

/*
 * 例题 2:「SCOI2016」幸运数字 - 树上查询 u,v 路径上选一些点异或起来最大
 * 解法:树链剖分维护一段路径的线性基, 暴力合并
 */
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
typedef long long ll;
const int maxn = 4e4 + 10;
/* head */
class XORbase {
public:
    ll b[62];

    void Init() {
        memset(b, 0, sizeof b);
    }

    void Insert(ll x) {
        per(i, 0, 61) {
```

```
                if (x & (1LL << i)) {
                    if (!b[i]) {
                        b[i] = x;
                        break;
                    } else x ^= b[i];
                }
            }
        }

        void Init(ll x) {
            memset(b, 0, sizeof b);
            Insert(x);
        }

        ll Calc() {
            ll res = 0;
            per(i, 0, 61) res = max(res, res ^ b[i]);
            return res;
        }

        friend XORbase operator+ (const XORbase &x, const XORbase &y) {
            XORbase res = x;
            per(i, 0, 61) if (y.b[i]) res.Insert(y.b[i]);
            return res;
        }

};
ll a[maxn];
class SegmentTree {
public:
#define lson (root << 1)
#define rson (root << 1 | 1)
#define lent (t[root].r - t[root].l + 1)
#define lenl (t[lson].r - t[lson].l + 1)
#define lenr (t[rson].r - t[rson].l + 1)
    struct Tree {
        int l, r;
        XORbase v;
    } t[maxn << 2];

    void pushup(int root) {
        t[root].v = t[lson].v + t[rson].v;
    }

    void build(int l, int r, int root) {
        t[root].l = l;
```

```
        t[root].r = r;
        t[root].v.Init();
        if (l == r) {
            return;
        }
        int mid = l + r >> 1;
        build(l, mid, lson);
        build(mid + 1, r, rson);
        pushup(root);
    }

    void update(int l, int r, ll x, int root) {
        if (l <= t[root].l && t[root].r <= r) {
            t[root].v.Insert(x);
            return;
        }
        int mid = t[root].l + t[root].r >> 1;
        if (l <= mid) update(l, r, x, lson);
        if (r > mid) update(l, r, x, rson);
        pushup(root);
    }

    XORbase query(int l, int r, int root) {
        if (l <= t[root].l && t[root].r <= r)
            return t[root].v;
        int mid = t[root].l + t[root].r >> 1;
        XORbase ans;
        ans.Init();
        if (l <= mid) ans = ans + query(l, r, lson);
        if (r > mid) ans = ans + query(l, r, rson);
        return ans;
    }
#undef lenr
#undef lenl
#undef lent
#undef rson
#undef lson
} T;
struct Edge {
    int to, next;
    Edge() {}
    Edge(int a, int b) {
        to = a;
        next = b;
    }
} E[maxn << 1];
```

```
int head[maxn], cnt, tot;
int top[maxn], son[maxn], size[maxn], deep[maxn], pa[maxn], id[maxn];
inline void init() {
    memset(head, -1, sizeof head);
    tot = cnt = 0;
}
inline void addedge(int u, int v) {
    E[cnt].to = v;
    E[cnt].next = head[u];
    head[u] = cnt++;
}
inline void dfs1(int u, int fa, int d) {
    size[u] = 1;
    deep[u] = d;
    son[u] = 0;
    for (int i = head[u]; ~i; i = E[i].next) {
        int v = E[i].to;
        if (v != fa) {
            dfs1(v, u, d + 1);
            pa[v] = u;
            size[u] += size[v];
            if (size[v] > size[son[u]]) son[u] = v;
        }
    }
}
void dfs2(int u, int first) {
    top[u] = first;
    id[u] = ++tot;
    if (son[u]) dfs2(son[u], first);
    for (int i = head[u]; ~i; i = E[i].next) {
        int v = E[i].to;
        if (v != pa[u] && v != son[u]) dfs2(v, v);
    }
}
XORbase calc(int u, int v) {
    int x = top[u], y = top[v];
    XORbase res;
    res.Init();
    while (x != y) {
        if (deep[x] < deep[y]) {
            swap(u, v);
            swap(x, y);
        }
        res = res + T.query(id[x], id[u], 1);
        u = pa[x];
        x = top[u];
```

```
    }
    if (deep[u] > deep[v]) swap(u, v);
    res = res + T.query(id[u], id[v], 1);
    return res;
}
int main() {
    int n, q;
    init();
    scanf("%d%d", &n, &q);
    rep(i, 1, n + 1) scanf("%lld", &a[i]);
    rep(i, 1, n) {
        int u, v;
        scanf("%d%d", &u, &v);
        addedge(u, v);
        addedge(v, u);
    }
    dfs1(1, 0, 1);
    dfs2(1, 1);
    T.build(1, tot, 1);
    rep(i, 1, n + 1) T.update(id[i], id[i], a[i], 1);
    while (q--) {
        int x, y;
        ll ans = 0;
        scanf("%d%d", &x, &y);
        XORbase res = calc(x, y);
        per(i, 0, 61) ans = max(ans, ans ^ res.b[i]);
        printf("%lld\n", ans);
    }
    return 0;
}
/*
 * 例题 3：BZOJ - 2115 - 边权图求 1-n 的最大 Xor 和路径，存在重边，允许经
   过重复点、重复边。
 * 解法：路径一定是由许多的环和一条从 1 到 n 的路径组成。因此预处理所有环的
   Xor，维护线性基取 Max 即可
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int>P;
const int INF = 0x3f3f3f3f, maxn = 50005;
int n, m, vis[maxn], tot, head[maxn];
ll dis[maxn], base[66];
vector<ll>a;
struct node {
    int v, next;
```

```
    ll w;
} g[200005];
void add(int u, int v, ll w) {
    g[tot].v = v, g[tot].next = head[u], g[tot].w = w, head[u] = tot++;
}
void dfs(int u, ll d) {
    dis[u] = d, vis[u] = 1;
    for(int i = head[u]; ~i; i = g[i].next) {
        int v = g[i].v;
        ll w = g[i].w;
        if(vis[v])a.push_back(dis[u]^dis[v]^w);
        else dfs(v, dis[u]^w);
    }
}
int main() {
    while(~scanf("%d%d", &n, &m)) {
        a.clear();
        tot = 0;
        for(int i = 1; i <= n; i++)vis[i] = 0, head[i] = -1;
        while(m--) {
            int u, v;
            ll w;
            scanf("%d%d%lld", &u, &v, &w);
            add(u, v, w), add(v, u, w);
        }
        dfs(1, 0);
        ll ans = dis[n];
        memset(base, 0, sizeof(base));
        for(int i = 0; i < a.size(); i++)
            for(int j = 62; j >= 0; j--)
                if((a[i] >> j) & 1) {
                    if(!base[j]) {
                        base[j] = a[i];
                        break;
                    } else a[i] ^= base[j];
                }
        for(int i = 62; i >= 0; i--)ans = max(ans, ans ^ base[i]);
        printf("%lld\n", ans);
    }
    return 0;
}
```

6.23 Polya 定理

Burnside 引理: 等价类的个数为 $= \frac{c_1(a_1)+c_1(a_2)+\cdots c_1(a_n)}{|G|}$，竞赛中使用没有 Polya 定理普遍。

Polya 定理定义: 设 G 是 p 个对象的一个置换群，用 k 种颜色去染这 p 个对象，若一种染色方案在群 G 的作用下变为另一种方案，则这两个方案当作是同一种方案，这样的不同染色方案数为：$L = \frac{1}{|G|} \times \sum k^{C(f)}$，$f \in G$，其中 $C(f)$ 为循环节，$|G|$ 表示群的置换方法数。

- 对于有 n 个位置的手镯，有 n 种旋转置换和 n 种翻转置换，那么：
  - 对于旋转置换，$C(f_i) = gcd(n, i)$，其中 i 表示一次转过 i 颗宝石，$i = 0$ 时 $c = n$
  - 对于翻转置换，如果 n 为偶数，则有 $\frac{n}{2}$ 个置换 $C(f_1) = \frac{n}{2}$，有 $\frac{n}{2}$ 个置换 $C(f_2) = \frac{n}{2} + 1$；如果 n 为奇数，那么 $C(f) = \frac{n}{2} + 1$

一般循环置换的代码是:

```
int ans = 0;
for(int i = 0; i < n; i++) {
    ans += a[__gcd(n, i)];
}
ans /= n;
```

6.24 Lindstrom-Gessel-Viennot lemma 定理

对于一张无边权的 DAG 图，给定 n 个起点对应的 n 个终点，这 n 条不相交路径的方案数为下列矩阵的行列式:

$$\begin{bmatrix} f(a_1, b_1) & f(a_1, b_2) & \cdots & f(a_1, b_n) \\ f(a_2, b_1) & f(a_2, b_2) & \cdots & f(a_2, b_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(a_n, b_1) & f(a_n, b_2) & \cdots & f(a_n, b_n) \end{bmatrix}$$

其中 $f(a_i, b_i)$ 为图上点 a_i 到点 b_i 的方案数。

- **例题 1:** Codeforces348D - 给定一张 $n \times m$ $(2 \le n, m \le 3000)$ 带障碍的图，求从左上角到右下角不相交两条路径的方案。
  - **解法:** $a_1 = (1, 2), a_2 = (2, 1), b_1 = (n - 1, m), b_2 = (n, m - 1)$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9 + 7;
char s[3003][3003];
ll f[3003][3003];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%s", s[i] + 1);
    memset(f, 0, sizeof f);
```

```
    f[1][2] = s[1][2] == '.';
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s[i][j] == '.') {
                f[i][j] = (f[i][j] + f[i - 1][j] + f[i][j - 1]) % mod;
            }
        }
    }
    ll x1 = f[n - 1][m];
    ll x2 = f[n][m - 1];
    memset(f, 0, sizeof f);
    f[2][1] = s[2][1] == '.';
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s[i][j] == '.') {
                f[i][j] = (f[i][j] + f[i - 1][j] + f[i][j - 1]) % mod;
            }
        }
    }
    ll y1 = f[n - 1][m];
    ll y2 = f[n][m - 1];
    printf("%I64d\n", (x1 * y2 % mod - x2 * y1 % mod + mod) % mod);
    return 0;
}
```

- **例题 2:** HDU5852 - 给一张 $n \times n$ 的图，第一行 m 个点对应第 n 行的 m 个点，求路径不相交的方案数
  - **解法:** 计算对应的行列式，注意高斯消元不要 TLE
- **例题 3:** 问你有多少个满足条件的 $n \cdot m$ $(1 \le n, m \le 10^3)$ 的矩阵 A，满足矩阵每个元素 $A_{i,j} \in \{0, 1, 2\}$ 并且 $A_{i,j} \le A_{i+1,j}$ 而且 $A_{i,j} \le A_{i,j+1}$，答案取模 $10^9 + 7$。
  - **解法:** 考虑 01 和 12 的分界线，用 $(n, 0)$ 到 $(0, m)$ 的两条不相交（可重合）路径，平移其中一条变成 $(n-1, -1)$ 到 $(-1, m-1)$ 变成起点 $(n, 0)$ 和 $(n-1, -1)$，终点 $(0, m)$ 和 $(-1, m-1)$ 的严格不相交路径，套 Lindstrom-Gessel-Viennot lemma 定理，答案就是 $(C_{n+m}^{n})^2 - C_{n+m}^{m-1} \cdot C_{n+m}^{n-1}$！

7 计算几何

7.1 二维几何

7.1.1 通用函数

```
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
const int maxp = 1010;
// Compares a double to zero
int sgn(double x) {
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
// square of a double
inline double sqr(double x) {
    return x * x;
}
/*
 * Point
 * Point() -Empty constructor
 * Point(double _x,double _y) -constructor
 * input() -double input
 * output() -%.2f output
 * operator == -compares x and y
 * operator < -compares first by x, then by y
 * operator --return new Point after subtracting curresponging x and y
 * operator ^ -cross product of 2d points
 * operator * -dot product
 * len() -gives length from origin
 * len2() -gives square of length from origin
 * distance(Point p) -gives distance from p
 * operator + Point b -returns new Point after adding curresponging x
 ↪ and y
 * operator * double k -returns new Point after multiplieing x and y by
 ↪ k
 * operator / double k -returns new Point after divideing x and y by k
 * rad(Point a,Point b)-returns the angle of Point a and Point b from
 ↪ this Point
 * trunc(double r) -return Point that if truncated the distance from
 ↪ center to r
 * rotleft() -returns 90 degree ccw rotated point
 * rotright() -returns 90 degree cw rotated point
 * rotate(Point p,double angle) -returns Point after rotateing the Point
 ↪ centering at p by angle radian ccw
```

```
 */
struct Point {
    double x, y;
    Point() {}
    Point(double _x, double _y) {
        x = _x;
        y = _y;
    }
    void input() {
        scanf("%lf%lf", &x, &y);
    }
    void output() {
        printf("%.2f %.2f\n", x, y);
    }
    bool operator == (Point b)const {
        return sgn(x - b.x) == 0 && sgn(y - b.y) == 0;
    }
    bool operator < (Point b)const {
        return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0 : x < b.x;
    }
    Point operator -(const Point &b)const {
        return Point(x - b.x, y - b.y);
    }
    // 叉积
    double operator ^(const Point &b)const {
        return x * b.y - y * b.x;
    }
    // 点积
    double operator *(const Point &b)const {
        return x * b.x + y * b.y;
    }
    // 返回长度
    double len() {
        return hypot(x, y); // 库函数
    }
    // 返回长度的平方
    double len2() {
        return x * x + y * y;
    }
    // 返回两点的距离
    double distance(Point p) {
        return hypot(x - p.x, y - p.y);
    }
    Point operator +(const Point &b)const {
        return Point(x + b.x, y + b.y);
    }
```

```
    Point operator *(const double &k)const {
        return Point(x * k, y * k);
    }
    Point operator /(const double &k)const {
        return Point(x / k, y / k);
    }
    // 计算 pa 和 pb 的夹角
    // 就是求这个点看 a,b 所成的夹角
    // 测试 LightOJ1203
    double rad(Point a, Point b) {
        Point p = *this;
        return fabs(atan2( fabs((a - p) ^ (b - p)), (a - p) * (b - p) ));
    }
    // 化为长度为 r 的向量
    Point trunc(double r) {
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x * r, y * r);
    }
    // 逆时针旋转 90 度
    Point rotleft() {
        return Point(-y, x);
    }
    // 顺时针旋转 90 度
    Point rotright() {
        return Point(y, -x);
    }
    // 绕着 p 点逆时针旋转 angle
    Point rotate(Point p, double angle) {
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
    }
};
/*
 * Stores two points
 * Line() -Empty constructor
 * Line(Point _s,Point _e) -Line through _s and _e
 * operator == -checks if two points are same
 * Line(Point p,double angle) -one end p , another end at angle degree
 * Line(double a,double b,double c) -Line of equation ax + by + c = 0
 * input() -inputs s and e
 * adjust() -orders in such a way that s < e
 * length() -distance of se
 * angle() -return 0 <= angle < pi
```

```
 * relation(Point p) -3 if point is on line
 * 1 if point on the left of line
 * 2 if point on the right of line
 * pointonseg(double p) -return true if point on segment
 * parallel(Line v) -return true if they are parallel
 * segcrossseg(Line v) -returns 0 if does not intersect
 * returns 1 if non-standard intersection
 * returns 2 if intersects
 * linecrossseg(Line v) -line and seg
 * linecrossline(Line v) -0 if parallel
 * 1 if coincides
 * 2 if intersects
 * crosspoint(Line v) -returns intersection point
 * dispointtoline(Point p) -distance from point p to the line
 * dispointtoseg(Point p) -distance from p to the segment
 * dissegtoseg(Line v) -distance of two segment
 * lineprog(Point p) -returns projected point p on se line
 * symmetrypoint(Point p) -returns reflection point of p over se
 *
 */
struct Line {
    Point s, e;
    Line() {}
    Line(Point _s, Point _e) {
        s = _s;
        e = _e;
    }
    bool operator ==(Line v) {
        return (s == v.s) && (e == v.e);
    }
    // 根据一个点和倾斜角 angle 确定直线,0<=angle<pi
    Line(Point p, double angle) {
        s = p;
        if(sgn(angle - pi / 2) == 0) {
            e = (s + Point(0, 1));
        } else {
            e = (s + Point(1, tan(angle)));
        }
    }
    // ax+by+c=0
    Line(double a, double b, double c) {
        if(sgn(a) == 0) {
            s = Point(0, -c / b);
            e = Point(1, -c / b);
        } else if(sgn(b) == 0) {
            s = Point(-c / a, 0);
```

```
            e = Point(-c / a, 1);
        } else {
            s = Point(0, -c / b);
            e = Point(1, (-c - a) / b);
        }
    }
    void input() {
        s.input();
        e.input();
    }
    void adjust() {
        if(e < s)swap(s, e);
    }
    // 求线段长度
    double length() {
        return s.distance(e);
    }
    // 返回直线倾斜角 0<=angle<pi
    double angle() {
        double k = atan2(e.y - s.y, e.x - s.x);
        if(sgn(k) < 0)k += pi;
        if(sgn(k - pi) == 0)k -= pi;
        return k;
    }
    // 点和直线关系
    // 1 在左侧
    // 2 在右侧
    // 3 在直线上
    int relation(Point p) {
        int c = sgn((p - s) ^ (e - s));
        if(c < 0)return 1;
        else if(c > 0)return 2;
        else return 3;
    }
    //  点在线段上的判断
    bool pointonseg(Point p) {
        return sgn((p - s) ^ (e - s)) == 0 && sgn((p - s) * (p - e)) <=
         ↪  0;
    }
    // 两向量平行 (对应直线平行或重合)
    bool parallel(Line v) {
        return sgn((e - s) ^ (v.e-v.s)) == 0;
    }
    // 两线段相交判断
    // 2 规范相交
    // 1 非规范相交
```

```
    // 0 不相交
    int segcrossseg(Line v) {
        int d1 = sgn((e - s) ^ (v.s - s));
        int d2 = sgn((e - s) ^ (v.e-s));
        int d3 = sgn((v.e-v.s) ^ (s - v.s));
        int d4 = sgn((v.e-v.s) ^ (e - v.s));
        if( (d1 ^ d2) == -2 && (d3 ^ d4) == -2 )return 2;
        return (d1 == 0 && sgn((v.s - s) * (v.s - e)) <= 0) ||
               (d2 == 0 && sgn((v.e-s) * (v.e-e)) <= 0) ||
               (d3 == 0 && sgn((s - v.s) * (s - v.e)) <= 0) ||
               (d4 == 0 && sgn((e - v.s) * (e - v.e)) <= 0);
    }
    // 直线和线段相交判断
    // *this line -v seg
    // 2 规范相交
    // 1 非规范相交
    // 0 不相交
    int linecrossseg(Line v) {
        int d1 = sgn((e - s) ^ (v.s - s));
        int d2 = sgn((e - s) ^ (v.e-s));
        if((d1 ^ d2) == -2) return 2;
        return (d1 == 0 || d2 == 0);
    }
    // 两直线关系
    // 0 平行
    // 1 重合
    // 2 相交
    int linecrossline(Line v) {
        if((*this).parallel(v))
            return v.relation(s) == 3;
        return 2;
    }
    // 求两直线的交点
    // 要保证两直线不平行或重合
    Point crosspoint(Line v) {
        double a1 = (v.e-v.s) ^ (s - v.s);
        double a2 = (v.e-v.s) ^ (e - v.s);
        return Point((s.x * a2 - e.x * a1) / (a2 - a1), (s.y * a2 - e.y *
         ↪  a1) / (a2 - a1));
    }
    // 点到直线的距离
    double dispointtoline(Point p) {
        return fabs((p - s) ^ (e - s)) / length();
    }
    // 点到线段的距离
    double dispointtoseg(Point p) {
```

```
        if(sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.distance(s), p.distance(e));
        return dispointtoline(p);
    }
    // 返回线段到线段的距离
    // 前提是两线段不相交，相交距离就是 0 了
    double dissegtoseg(Line v) {
        return min(min(dispointtoseg(v.s), dispointtoseg(v.e)),
        ↪  min(v.dispointtoseg(s), v.dispointtoseg(e)));
    }
    // 返回点 p 在直线上的投影
    Point lineprog(Point p) {
        return s + ( ((e - s) * ((e - s) * (p - s))) / ((e - s).len2())
        ↪  );
    }
    // 返回点 p 关于直线的对称点
    Point symmetrypoint(Point p) {
        Point q = lineprog(p);
        return Point(2 * q.x - p.x, 2 * q.y - p.y);
    }
};
// 圆
struct circle {
    Point p;// 圆心
    double r;// 半径
    circle() {}
    circle(Point _p, double _r) {
        p = _p;
        r = _r;
    }
    circle(double x, double y, double _r) {
        p = Point(x, y);
        r = _r;
    }
    // 三角形的外接圆
    // 需要 Point 的 + / rotate() 以及 Line 的 crosspoint()
    // 利用两条边的中垂线得到圆心
    // 测试：UVA12304
    circle(Point a, Point b, Point c) {
        Line u = Line((a + b) / 2, ((a + b) / 2) + ((b - a).rotleft()));
        Line v = Line((b + c) / 2, ((b + c) / 2) + ((c - b).rotleft()));
        p = u.crosspoint(v);
        r = p.distance(a);
    }
    // 三角形的内切圆
    // 参数 bool t 没有作用，只是为了和上面外接圆函数区别
```

```
    // 测试：UVA12304
    circle(Point a, Point b, Point c, bool t) {
        Line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x
         ↪  - a.x);
        u.s = a;
        u.e = u.s + Point(cos((n + m) / 2), sin((n + m) / 2));
        v.s = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
        v.e = v.s + Point(cos((n + m) / 2), sin((n + m) / 2));
        p = u.crosspoint(v);
        r = Line(a, b).dispointtoseg(p);
    }
    // 输入
    void input() {
        p.input();
        scanf("%lf", &r);
    }
    // 输出
    void output() {
        printf("%.2lf %.2lf %.2lf\n", p.x, p.y, r);
    }
    bool operator == (circle v) {
        return (p == v.p) && sgn(r - v.r) == 0;
    }
    bool operator < (circle v)const {
        return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
    }
    // 面积
    double area() {
        return pi * r * r;
    }
    // 周长
    double circumference() {
        return 2 * pi * r;
    }
    // 点和圆的关系
    // 0 圆外
    // 1 圆上
    // 2 圆内
    int relation(Point b) {
        double dst = b.distance(p);
        if(sgn(dst - r) < 0)return 2;
        else if(sgn(dst - r) == 0)return 1;
        return 0;
    }
```

```
    // 线段和圆的关系
    // 比较的是圆心到线段的距离和半径的关系
    int relationseg(Line v) {
        double dst = v.dispointtoseg(p);
        if(sgn(dst - r) < 0)return 2;
        else if(sgn(dst - r) == 0)return 1;
        return 0;
    }
    // 直线和圆的关系
    // 比较的是圆心到直线的距离和半径的关系
    int relationline(Line v) {
        double dst = v.dispointtoline(p);
        if(sgn(dst - r) < 0)return 2;
        else if(sgn(dst - r) == 0)return 1;
        return 0;
    }
    // 两圆的关系
    // 5 相离
    // 4 外切
    // 3 相交
    // 2 内切
    // 1 内含
    // 需要 Point 的 distance
    // 测试：UVA12304
    int relationcircle(circle v) {
        double d = p.distance(v.p);
        if(sgn(d - r - v.r) > 0)return 5;
        if(sgn(d - r - v.r) == 0)return 4;
        double l = fabs(r - v.r);
        if(sgn(d - r - v.r) < 0 && sgn(d - l) > 0)return 3;
        if(sgn(d - l) == 0)return 2;
        if(sgn(d - l) < 0)return 1;
    }
    // 求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点
    // 需要 relationcircle
    // 测试：UVA12304
    int pointcrosscircle(circle v, Point &p1, Point &p2) {
        int rel = relationcircle(v);
        if(rel == 1 || rel == 5)return 0;
        double d = p.distance(v.p);
        double l = (d * d + r * r - v.r * v.r) / (2 * d);
        double h = sqrt(r * r - l * l);
        Point tmp = p + (v.p - p).trunc(l);
        p1 = tmp + ((v.p - p).rotleft().trunc(h));
        p2 = tmp + ((v.p - p).rotright().trunc(h));
        if(rel == 2 || rel == 4)
```

```
            return 1;
        return 2;
    }
    // 求直线和圆的交点，返回交点个数
    int pointcrossline(Line v, Point &p1, Point &p2) {
        if(!(*this).relationline(v))return 0;
        Point a = v.lineprog(p);
        double d = v.dispointtoline(p);
        d = sqrt(r * r - d * d);
        if(sgn(d) == 0) {
            p1 = a;
            p2 = a;
            return 1;
        }
        p1 = a + (v.e-v.s).trunc(d);
        p2 = a - (v.e-v.s).trunc(d);
        return 2;
    }
    // 得到过 a,b 两点，半径为 r1 的两个圆
    int gercircle(Point a, Point b, double r1, circle &c1, circle &c2) {
        circle x(a, r1), y(b, r1);
        int t = x.pointcrosscircle(y, c1.p, c2.p);
        if(!t)return 0;
        c1.r = c2.r = r;
        return t;
    }
    // 得到与直线 u 相切，过点 q，半径为 r1 的圆
    // 测试：UVA12304
    int getcircle(Line u, Point q, double r1, circle &c1, circle &c2) {
        double dis = u.dispointtoline(q);
        if(sgn(dis - r1 * 2) > 0)return 0;
        if(sgn(dis) == 0) {
            c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
            c2.p = q + ((u.e-u.s).rotright().trunc(r1));
            c1.r = c2.r = r1;
            return 2;
        }
        Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)), (u.e +
         ↪  (u.e-u.s).rotleft().trunc(r1)));
        Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)), (u.e +
         ↪  (u.e-u.s).rotright().trunc(r1)));
        circle cc = circle(q, r1);
        Point p1, p2;
        if(!cc.pointcrossline(u1, p1, p2))cc.pointcrossline(u2, p1, p2);
        c1 = circle(p1, r1);
        if(p1 == p2) {
```

```
            c2 = c1;
            return 1;
        }
        c2 = circle(p2, r1);
        return 2;
    }
    // 同时与直线 u,v 相切，半径为 r1 的圆
    // 测试：UVA12304
    int getcircle(Line u, Line v, double r1, circle &c1, circle &c2, circle &c3, circle &c4) {
        if(u.parallel(v))return 0;// 两直线平行
        Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1), u.e + (u.e-u.s).rotleft().trunc(r1));
        Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1), u.e + (u.e-u.s).rotright().trunc(r1));
        Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1), v.e + (v.e-v.s).rotleft().trunc(r1));
        Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1), v.e + (v.e-v.s).rotright().trunc(r1));
        c1.r = c2.r = c3.r = c4.r = r1;
        c1.p = u1.crosspoint(v1);
        c2.p = u1.crosspoint(v2);
        c3.p = u2.crosspoint(v1);
        c4.p = u2.crosspoint(v2);
        return 4;
    }
    // 同时与不相交圆 cx,cy 相切，半径为 r1 的圆
    // 测试：UVA12304
    int getcircle(circle cx, circle cy, double r1, circle &c1, circle &c2) {
        circle x(cx.p, r1 + cx.r), y(cy.p, r1 + cy.r);
        int t = x.pointcrosscircle(y, c1.p, c2.p);
        if(!t)return 0;
        c1.r = c2.r = r1;
        return t;
    }
    // 过一点作圆的切线 (先判断点和圆的关系)
    // 测试：UVA12304
    int tangentline(Point q, Line &u, Line &v) {
        int x = relation(q);
        if(x == 2)return 0;
        if(x == 1) {
            u = Line(q, q + (q - p).rotleft());
            v = u;
            return 1;
        }
```

```
        double d = p.distance(q);
        double l = r * r / d;
        double h = sqrt(r * r - l * l);
        u = Line(q, p + ((q - p).trunc(l) + (q - p).rotleft().trunc(h)));
        v = Line(q, p + ((q - p).trunc(l) + (q -
        ↪  p).rotright().trunc(h)));
        return 2;
    }
    // 求两圆相交的面积
    double areacircle(circle v) {
        int rel = relationcircle(v);
        if(rel >= 4)return 0.0;
        if(rel <= 2)return min(area(), v.area());
        double d = p.distance(v.p);
        double hf = (r + v.r + d) / 2.0;
        double ss = 2 * sqrt(hf * (hf - r) * (hf - v.r) * (hf - d));
        double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 * r * d));
        a1 = a1 * r * r;
        double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
        a2 = a2 * v.r * v.r;
        return a1 + a2 - ss;
    }
    // 求圆和三角形 pab 的相交面积
    // 测试：POJ3675 HDU3982 HDU2892
    double areatriangle(Point a, Point b) {
        if(sgn((p - a) ^ (p - b)) == 0)return 0.0;
        Point q[5];
        int len = 0;
        q[len++] = a;
        Line l(a, b);
        Point p1, p2;
        if(pointcrossline(l, q[1], q[2]) == 2) {
            if(sgn((a - q[1]) * (b - q[1])) < 0)q[len++] = q[1];
            if(sgn((a - q[2]) * (b - q[2])) < 0)q[len++] = q[2];
        }
        q[len++] = b;
        if(len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0)swap(q[1],
        ↪  q[2]);
        double res = 0;
        for(int i = 0; i < len - 1; i++) {
            if(relation(q[i]) == 0 || relation(q[i + 1]) == 0) {
                double arg = p.rad(q[i], q[i + 1]);
                res += r * r * arg / 2.0;
            } else {
                res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
            }
```

```
        }
        return res;
    }
};
/*
 * n,p Line l for each side
 * input(int _n) -inputs _n size polygon
 * add(Point q) -adds a point at end of the list
 * getline() -populates line array
 * cmp -comparision in convex_hull order
 * norm() -sorting in convex_hull order
 * getconvex(polygon &convex) -returns convex hull in convex
 * Graham(polygon &convex) -returns convex hull in convex
 * isconvex() -checks if convex
 * relationpoint(Point q) -returns 3 if q is a vertex
 * 2 if on a side
 * 1 if inside
 * 0 if outside
 * convexcut(Line u,polygon &po) -left side of u in po
 * gercircumference() -returns side length
 * getarea() -returns area
 * getdir() -returns 0 for cw, 1 for ccw
 * getbarycentre() -returns barycenter
 *
 */
struct polygon {
    int n;
    Point p[maxp];
    Line l[maxp];
    void input(int _n) {
        n = _n;
        for(int i = 0; i < n; i++)
            p[i].input();
    }
    void add(Point q) {
        p[n++] = q;
    }
    void getline() {
        for(int i = 0; i < n; i++) {
            l[i] = Line(p[i], p[(i + 1) % n]);
        }
    }
    struct cmp {
        Point p;
        cmp(const Point &p0) {
            p = p0;
```

```
        }
        bool operator()(const Point &aa, const Point &bb) {
            Point a = aa, b = bb;
            int d = sgn((a - p) ^ (b - p));
            if(d == 0) {
                return sgn(a.distance(p) - b.distance(p)) < 0;
            }
            return d > 0;
        }
    };
    // 进行极角排序
    // 首先需要找到最左下角的点
    // 需要重载号好 Point 的 < 操作符 (min 函数要用)
    void norm() {
        Point mi = p[0];
        for(int i = 1; i < n; i++)mi = min(mi, p[i]);
        sort(p, p + n, cmp(mi));
    }
    // 得到凸包
    // 得到的凸包里面的点编号是 0 n-1 的
    // 两种凸包的方法
    // 注意如果有影响，要特判下所有点共点，或者共线的特殊情况
    // 测试 LightOJ1203 LightOJ1239
    void getconvex(polygon &convex) {
        sort(p, p + n);
        convex.n = n;
        for(int i = 0; i < min(n, 2); i++) {
            convex.p[i] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;// 特
        ↪ 判
        if(n <= 2)return;
        int &top = convex.n;
        top = 1;
        for(int i = 2; i < n; i++) {
            while(top && sgn((convex.p[top] - p[i]) ^ (convex.p[top - 1]
            ↪ - p[i])) <= 0)
                top--;
            convex.p[++top] = p[i];
        }
        int temp = top;
        convex.p[++top] = p[n - 2];
        for(int i = n - 3; i >= 0; i--) {
            while(top != temp && sgn((convex.p[top] - p[i]) ^
            ↪ (convex.p[top - 1] - p[i])) <= 0)
                top--;
```

```
            convex.p[++top] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;// 特
        ↪ 判
        convex.norm();// 原来得到的是顺时针的点，排序后逆时针
    }
    // 得到凸包的另外一种方法
    // 测试 LightOJ1203 LightOJ1239
    void Graham(polygon &convex) {
        norm();
        int &top = convex.n;
        top = 0;
        if(n == 1) {
            top = 1;
            convex.p[0] = p[0];
            return;
        }
        if(n == 2) {
            top = 2;
            convex.p[0] = p[0];
            convex.p[1] = p[1];
            if(convex.p[0] == convex.p[1])top--;
            return;
        }
        convex.p[0] = p[0];
        convex.p[1] = p[1];
        top = 2;
        for(int i = 2; i < n; i++) {
            while( top > 1 && sgn((convex.p[top - 1] - convex.p[top - 2])
            ↪ ^ (p[i] - convex.p[top - 2])) <= 0 )
                top--;
            convex.p[top++] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;// 特
        ↪ 判
    }
    // 判断是不是凸的
    bool isconvex() {
        bool s[2];
        memset(s, false, sizeof(s));
        for(int i = 0; i < n; i++) {
            int j = (i + 1) % n;
            int k = (j + 1) % n;
            s[sgn((p[j] - p[i]) ^ (p[k] - p[i])) + 1] = true;
            if(s[0] && s[2])return false;
        }
```

```
            return true;
        }
        // 判断点和任意多边形的关系
        // 3 点上
        // 2 边上
        // 1 内部
        // 0 外部
        int relationpoint(Point q) {
            for(int i = 0; i < n; i++) {
                if(p[i] == q)return 3;
            }
            getline();
            for(int i = 0; i < n; i++) {
                if(l[i].pointonseg(q))return 2;
            }
            int cnt = 0;
            for(int i = 0; i < n; i++) {
                int j = (i + 1) % n;
                int k = sgn((q - p[j]) ^ (p[i] - p[j]));
                int u = sgn(p[i].y - q.y);
                int v = sgn(p[j].y - q.y);
                if(k > 0 && u < 0 && v >= 0)cnt++;
                if(k < 0 && v < 0 && u >= 0)cnt--;
            }
            return cnt != 0;
        }
        // 直线 u 切割凸多边形左侧
        // 注意直线方向
        // 测试：HDU3982
        void convexcut(Line u, polygon &po) {
            int &top = po.n;// 注意引用
            top = 0;
            for(int i = 0; i < n; i++) {
                int d1 = sgn((u.e-u.s) ^ (p[i] - u.s));
                int d2 = sgn((u.e-u.s) ^ (p[(i + 1) % n] - u.s));
                if(d1 >= 0)po.p[top++] = p[i];
                if(d1 * d2 < 0)po.p[top++] = u.crosspoint(Line(p[i], p[(i +
                 ↪  1) % n]));
            }
        }
        // 得到周长
        // 测试 LightOJ1239
        double getcircumference() {
            double sum = 0;
            for(int i = 0; i < n; i++) {
                sum += p[i].distance(p[(i + 1) % n]);
```

```
        }
        return sum;
    }
    //  得到面积
    double getarea() {
        double sum = 0;
        for(int i = 0; i < n; i++) {
            sum += (p[i] ^ p[(i + 1) % n]);
        }
        return fabs(sum) / 2;
    }
    //  得到方向
    //  1 表示逆时针, 0 表示顺时针
    bool getdir() {
        double sum = 0;
        for(int i = 0; i < n; i++)
            sum += (p[i] ^ p[(i + 1) % n]);
        if(sgn(sum) > 0)return 1;
        return 0;
    }
    // 得到重心
    Point getbarycentre() {
        Point ret(0, 0);
        double area = 0;
        for(int i = 1; i < n - 1; i++) {
            double tmp = (p[i] - p[0]) ^ (p[i + 1] - p[0]);
            if(sgn(tmp) == 0)continue;
            area += tmp;
            ret.x += (p[0].x + p[i].x + p[i + 1].x) / 3 * tmp;
            ret.y += (p[0].y + p[i].y + p[i + 1].y) / 3 * tmp;
        }
        if(sgn(area)) ret = ret / area;
        return ret;
    }
    // 多边形和圆交的面积
    // 测试：POJ3675 HDU3982 HDU2892
    double areacircle(circle c) {
        double ans = 0;
        for(int i = 0; i < n; i++) {
            int j = (i + 1) % n;
            if(sgn( (p[j] - c.p) ^ (p[i] - c.p) ) >= 0)
                ans += c.areatriangle(p[i], p[j]);
            else ans -= c.areatriangle(p[i], p[j]);
        }
        return fabs(ans);
    }
```

```
    //  多边形和圆关系
    //  2 圆完全在多边形内
    //  1 圆在多边形里面, 碰到了多边形边界
    //  0 其它
    int relationcircle(circle c) {
        getline();
        int x = 2;
        if(relationpoint(c.p) != 1)return 0;// 圆心不在内部
        for(int i = 0; i < n; i++) {
            if(c.relationseg(l[i]) == 2)return 0;
            if(c.relationseg(l[i]) == 1)x = 1;
        }
        return x;
    }
};
// AB X AC
double cross(Point A, Point B, Point C) {
    return (B - A) ^ (C - A);
}
// AB*AC
double dot(Point A, Point B, Point C) {
    return (B - A) * (C - A);
}
//  最小矩形面积覆盖
//  A 必须是凸包 (而且是逆时针顺序)
//  测试 UVA 10173
double minRectangleCover(polygon A) {
    // 要特判 A.n < 3 的情况
    if(A.n < 3)return 0.0;
    A.p[A.n] = A.p[0];
    double ans = -1;
    int r = 1, p = 1, q;
    for(int i = 0; i < A.n; i++) {
        // 卡出离边 A.p[i] - A.p[i+1] 最远的点
        while( sgn( cross(A.p[i], A.p[i + 1], A.p[r + 1]) - cross(A.p[i],
        ↪  A.p[i + 1], A.p[r]) ) >= 0 )
            r = (r + 1) % A.n;
        // 卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点
        while(sgn( dot(A.p[i], A.p[i + 1], A.p[p + 1]) - dot(A.p[i],
        ↪  A.p[i + 1], A.p[p]) ) >= 0 )
            p = (p + 1) % A.n;
        if(i == 0)q = p;
        // 卡出 A.p[i] - A.p[i+1] 方向上负向最远的点
        while(sgn(dot(A.p[i], A.p[i + 1], A.p[q + 1]) - dot(A.p[i], A.p[i
        ↪  + 1], A.p[q])) <= 0)
            q = (q + 1) % A.n;
```

```
        double d = (A.p[i] - A.p[i + 1]).len2();
        double tmp = cross(A.p[i], A.p[i + 1], A.p[r]) * (dot(A.p[i],
        ↪  A.p[i + 1], A.p[p]) - dot(A.p[i], A.p[i + 1], A.p[q])) / d;
        if(ans < 0 || ans > tmp)ans = tmp;
    }
    return ans;
}
// 直线切凸多边形
// 多边形是逆时针的, 在 q1q2 的左侧
// 测试:HDU3982
vector<Point> convexCut(const vector<Point> &ps, Point q1, Point q2) {
    vector<Point>qs;
    int n = ps.size();
    for(int i = 0; i < n; i++) {
        Point p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = sgn((q2 - q1) ^ (p1 - q1)), d2 = sgn((q2 - q1) ^ (p2 -
        ↪  q1));
        if(d1 >= 0)
            qs.push_back(p1);
        if(d1 * d2 < 0)
            qs.push_back(Line(p1, p2).crosspoint(Line(q1, q2)));
    }
    return qs;
}
// 半平面交
// 测试 POJ3335 POJ1474 POJ1279
// ***************************
struct halfplane: public Line {
    double angle;
    halfplane() {}
    // 表示向量 s->e 逆时针 (左侧) 的半平面
    halfplane(Point _s, Point _e) {
        s = _s;
        e = _e;
    }
    halfplane(Line v) {
        s = v.s;
        e = v.e;
    }
    void calcangle() {
        angle = atan2(e.y - s.y, e.x - s.x);
    }
    bool operator <(const halfplane &b)const {
        return angle < b.angle;
    }
};
```

```
struct halfplanes {
    int n;
    halfplane hp[maxp];
    Point p[maxp];
    int que[maxp];
    int st, ed;
    void push(halfplane tmp) {
        hp[n++] = tmp;
    }
    // 去重
    void unique() {
        int m = 1;
        for(int i = 1; i < n; i++) {
            if(sgn(hp[i].angle - hp[i - 1].angle) != 0)
                hp[m++] = hp[i];
            else if(sgn( (hp[m - 1].e-hp[m - 1].s) ^ (hp[i].s - hp[m - 1].s) ) > 0)
                hp[m - 1] = hp[i];
        }
        n = m;
    }
    bool halfplaneinsert() {
        for(int i = 0; i < n; i++)hp[i].calcangle();
        sort(hp, hp + n);
        unique();
        que[st = 0] = 0;
        que[ed = 1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for(int i = 2; i < n; i++) {
            while(st < ed && sgn((hp[i].e-hp[i].s) ^ (p[ed] - hp[i].s)) < 0)ed--;
            while(st < ed && sgn((hp[i].e-hp[i].s) ^ (p[st + 1] - hp[i].s)) < 0)st++;
            que[++ed] = i;
            if(hp[i].parallel(hp[que[ed - 1]]))return false;
            p[ed] = hp[i].crosspoint(hp[que[ed - 1]]);
        }
        while(st < ed && sgn((hp[que[st]].e-hp[que[st]].s) ^ (p[ed] - hp[que[st]].s)) < 0)ed--;
        while(st < ed && sgn((hp[que[ed]].e-hp[que[ed]].s) ^ (p[st + 1] - hp[que[ed]].s)) < 0)st++;
        if(st + 1 >= ed)return false;
        return true;
    }
    // 得到最后半平面交得到的凸多边形
    // 需要先调用 halfplaneinsert() 且返回 true
```

```
    void getconvex(polygon &con) {
        p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
        con.n = ed - st + 1;
        for(int j = st, i = 0; j <= ed; i++, j++)
            con.p[i] = p[j];
    }
};
// ***************************
const int maxn = 1010;
struct circles {
    circle c[maxn];
    double ans[maxn];// ans[i] 表示被覆盖了 i 次的面积
    double pre[maxn];
    int n;
    circles() {}
    void add(circle cc) {
        c[n++] = cc;
    }
    // x 包含在 y 中
    bool inner(circle x, circle y) {
        if(x.relationcircle(y) != 1)return 0;
        return sgn(x.r - y.r) <= 0 ? 1 : 0;
    }
    // 圆的面积并去掉内含的圆
    void init_or() {
        bool mark[maxn] = {0};
        int i, j, k = 0;
        for(i = 0; i < n; i++) {
            for(j = 0; j < n; j++)
                if(i != j && !mark[j]) {
                    if( (c[i] == c[j]) || inner(c[i], c[j]) )break;
                }
            if(j < n)mark[i] = 1;
        }
        for(i = 0; i < n; i++)
            if(!mark[i])
                c[k++] = c[i];
        n = k;
    }
    // 圆的面积交去掉内含的圆
    void init_add() {
        int i, j, k;
        bool mark[maxn] = {0};
        for(i = 0; i < n; i++) {
            for(j = 0; j < n; j++)
                if(i != j && !mark[j]) {
```

```
                if( (c[i] == c[j]) || inner(c[j], c[i]) )break;
            }
            if(j < n)mark[i] = 1;
        }
        for(i = 0; i < n; i++)
            if(!mark[i])
                c[k++] = c[i];
        n = k;
    }
    // 半径为 r 的圆，弧度为 th 对应的弓形的面积
    double areaarc(double th, double r) {
        return 0.5 * r * r * (th - sin(th));
    }
    // 测试 SPOJVCIRCLES SPOJCIRUT
    // SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init_or() 去掉重复圆（否则
    ↪  WA）
    // SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init_or()
    // 对于求覆盖多少次面积的问题，不能解决相同圆，而且不能 init_or()
    // 求多圆面积并，需要 init_or，其中一个目的就是去掉相同圆
    void getarea() {
        memset(ans, 0, sizeof(ans));
        vector<pair<double, int> >v;
        for(int i = 0; i < n; i++) {
            v.clear();
            v.push_back(make_pair(-pi, 1));
            v.push_back(make_pair(pi, -1));
            for(int j = 0; j < n; j++)
                if(i != j) {
                    Point q = (c[j].p - c[i].p);
                    double ab = q.len(), ac = c[i].r, bc = c[j].r;
                    if(sgn(ab + ac - bc) <= 0) {
                        v.push_back(make_pair(-pi, 1));
                        v.push_back(make_pair(pi, -1));
                        continue;
                    }
                    if(sgn(ab + bc - ac) <= 0)continue;
                    if(sgn(ab - ac - bc) > 0)continue;
                    double th = atan2(q.y, q.x), fai = acos((ac * ac + ab
                    ↪  * ab - bc * bc) / (2.0 * ac * ab));
                    double a0 = th - fai;
                    if(sgn(a0 + pi) < 0)a0 += 2 * pi;
                    double a1 = th + fai;
                    if(sgn(a1 - pi) > 0)a1 -= 2 * pi;
                    if(sgn(a0 - a1) > 0) {
                        v.push_back(make_pair(a0, 1));
                        v.push_back(make_pair(pi, -1));
```

```
                        v.push_back(make_pair(-pi, 1));
                        v.push_back(make_pair(a1, -1));
                    } else {
                        v.push_back(make_pair(a0, 1));
                        v.push_back(make_pair(a1, -1));
                    }
                }
            sort(v.begin(), v.end());
            int cur = 0;
            for(int j = 0; j < v.size(); j++) {
                if(cur && sgn(v[j].first - pre[cur])) {
                    ans[cur] += areaarc(v[j].first - pre[cur], c[i].r);
                    ans[cur] += 0.5 * (Point(c[i].p.x + c[i].r *
                     ↪  cos(pre[cur]), c[i].p.y + c[i].r *
                     ↪  sin(pre[cur]))^Point(c[i].p.x + c[i].r *
                     ↪  cos(v[j].first), c[i].p.y + c[i].r *
                     ↪  sin(v[j].first)));
                }
                cur += v[j].second;
                pre[cur] = v[j].first;
            }
        }
        for(int i = 1; i < n; i++)
            ans[i] -= ans[i + 1];
    }
};
```

7.1.2 二维凸包

```
/* zhan 按顺时针保存了凸包上的点编号, 有 top 个
 * 所有点的下标 1 开始, 栈中也是下标 1 开始
 * 注意重复点！会影响凸形
 */
#include <bits/stdc++.h>
#define maxn 50010
#define eps 1e-8
using namespace std;
typedef int data_type;
struct point {
    data_type x, y;
    int id;
    point() {}
    inline point(data_type _x, data_type _y, int _id) {
        x = _x;
        y = _y;
        id = _id;
    }
```

```
    inline point(data_type _x, data_type _y) {
        x = _x;
        y = _y;
    }
    inline point operator+(const point &b)const {
        return point(x + b.x, y + b.y);
    }
    inline point operator-(const point &b)const {
        return point(x - b.x, y - b.y);
    }
    inline data_type operator^(const point &b)const {
        return x * b.y - y * b.x;    // 叉乘
    }
    inline data_type operator*(const point &b)const {
        return x * b.x - y * b.y;    // 点乘
    }
    inline bool operator<(const point &b)const {
        return x < b.x || x == b.x && y < b.y;
    }
} p[maxn];
inline data_type sqr_dist(const point a, point b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
inline double dist(const point a, point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
inline bool jijiao_cmp(const point &a, const point &b) { // 顺时针
    data_type tmp = (a - p[1]) ^ (b - p[1]);
    if (tmp < -eps)return 0;
    if (tmp > eps)return 1;
    return sqr_dist(p[1], a) < sqr_dist(p[1], b);
}
int zhan[maxn], top;
inline void graham(int n) {
    top = 0;
    for (int i = 2; i <= n; i++)if (p[i] < p[1])swap(p[i], p[1]);
    sort(p + 2, p + n + 1, jijiao_cmp);
    for(int i = 1; i <= n; i++) { // 默认凸包边界直线上的点在凸包上，如果
     ↪ 不要算，< 改 <= 即可
        while (top > 1 && ((p[zhan[top]] - p[zhan[top - 1]]) ^ (p[i] -
         ↪ p[zhan[top - 1]])) < 0)top--;
        zhan[++top] = i;
    }
    int now = n - 1;
    // 默认凸包边界直线上的点在凸包上，要加这一句
```

```cpp
    while (now >= 1 && ((p[now] - p[1]) ^ (p[zhan[top]] - p[1])) == 0)zhan[++top] = now--;
}
int n;
inline void work() {
    for (int i = 1; i <= n; i++)scanf("%d%d", &p[i].x, &p[i].y);
    graham(n);
    puts(top == n ? "convex" : "concave");
}
int main() {
    while (~scanf("%d", &n) && n) work();
    return 0;
}
```

7.1.3 动态凸包

```cpp
/* 例题：codeforces 70D
 *          操作一：插入一个点
 *          操作二：查询一个点是否在凸包内
 * 解法：平衡树维护凸包
 */
#include <bits/stdc++.h>
#define inf 0x3f3f3f3f
typedef long long ll;
using namespace std;
map<int, int> lower, upper;
ll det(ll x1, ll y1, ll x2, ll y2, ll x3, ll y3) {
    return x1 * y2 - x2 * y1 + x2 * y3 - x3 * y2 + x3 * y1 - x1 * y3;
}
bool check(map<int, int> &A, int x, int y) { //检查 (x,y) 是否在 A 下凸壳内
    auto it = A.lower_bound(x);
    if (it == A.end())return false;
    if (it->first == x)return it->second >= y;
    if (it == A.begin())return false;
    auto itp = it--;
    return det(it->first, it->second, x, y, itp->first, itp->second) >= 0;
}
void add(map<int, int> &A, int x, int y) {
    if (check(A, x, y)) return;
    A[x] = y;
    auto it = A.find(x), ita = it++, itb = it;
    while (it != A.end()) {
        itb = it++;
        if (it == A.end())break;
```

```
        if (det(ita->first, ita->second, itb->first, itb->second,
        ↪  it->first, it->second) < 0) break;
        A.erase(itb);
    }
    while (1) {
        it = ita;
        if (it == A.begin()) break;
        ita = it--;
        if (it == A.begin()) break;
        itb = it--;
        if (det(ita->first, ita->second, itb->first, itb->second,
        ↪  it->first, it->second) > 0) break;
        A.erase(itb);
    }
}
int main() {
    int T;
    scanf("%d", &T);
    for (int i = 1; i <= T; i++) {
        int t, x, y;
        scanf("%d%d%d", &t, &x, &y);
        if (t == 1)add(lower, x, y), add(upper, x, -y);
        else puts((check(lower, x, y) && check(upper, x, -y)) ? "YES" :
        ↪  "NO");
    }
    return 0;
}
```

7.1.4 凸多边形面积并和半平面交

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100010;
const double eps = 1e-16;
int T, n, cnt, front, rear;
double area;
int dcmp(double x) {
    if(fabs(x) < eps) return 0;
    return x > eps ? 1 : -1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double x, double y): x(x), y(y) {}
} p[MAXN];
typedef Point Vector;
```

```
Vector operator+(const Vector &p, const Vector &q) {
    return Vector(p.x + q.x, p.y + q.y);
}
Vector operator-(const Vector &p, const Vector &q) {
    return Vector(p.x - q.x, p.y - q.y);
}
Vector operator*(const Vector &p, double k) {
    return Vector(p.x * k, p.y * k);
}
double Cross(const Vector &p, const Vector &q) {
    return p.x * q.y - p.y * q.x;
}
struct Line {
    Point p;
    Vector v;
    double ang;
    Line() {}
    Line(const Point &p, const Vector &v): p(p), v(v) {
        ang = atan2(v.y, v.x);
    }
    friend bool operator<(const Line &p, const Line &q) {
        return p.ang < q.ang;
    }
} L[MAXN], q[MAXN];
Point LineIntersection(const Line &a, const Line &b) {
    Vector p = a.p - b.p;
    double t = Cross(b.v, p) / Cross(a.v, b.v);
    return Point(a.p + a.v * t);
}
bool Left(const Point &a, const Line &b) {
    return dcmp(Cross(a - b.p, b.v)) < 0;
}
bool Halfplane() {
    sort(L + 1, L + n + 1);
    q[front = rear = 1] = L[1];
    for(int i = 2; i <= n; i++) {
        while(front < rear && !Left(p[rear - 1], L[i])) rear--;
        while(front < rear && !Left(p[front], L[i])) front++;
        q[++rear] = L[i];
        if(!dcmp(q[rear].ang - q[rear - 1].ang) && Left(L[i].p,
         ↪  q[--rear])) q[rear] = L[i];
        if(front < rear) p[rear - 1] = LineIntersection(q[rear - 1],
         ↪  q[rear]);
    }
    while(front < rear && !Left(p[rear - 1], q[front])) rear--;
    if(rear - front <= 1) return false;
```

```
    p[rear] = LineIntersection(q[rear], q[front]);
    return true;
}
int len[10];
inline void work() {
    int T = 2;
    scanf("%d%d", len + 1, len + 2);
    n = cnt = front = rear = 0;
    area = 0;
    for(int cas = 1; cas <= T; cas++) {
        n = len[cas];
        for(int i = 1; i <= n; i++)scanf("%lf%lf", &p[i].x, &p[i].y);
        for(int i = 1; i < n; i++) L[++cnt] = Line(p[i], p[i + 1] - p[i]);
        L[++cnt] = Line(p[n], p[1] - p[n]);
    }
    n = cnt;
    if(!Halfplane()) {
        puts("0.0000");
        return;
    } else {
        for(int i = front; i < rear; i++) area += Cross(p[i], p[i + 1]);
        area += Cross(p[rear], p[front]);
    }
    printf("%.4f\n", fabs(area / 2));
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) work();
    return 0;
}
```

7.1.5 凸多边形面积交

- C++ Version

```
#include <bits/stdc++.h>
#define EPS 1e-8
using namespace std;
const int MAXN = 105;
inline int sgn(double x) {
    if (x < -EPS) return -1;
    else if (x > EPS) return 1;
    else return 0;
}
struct Point {
```

```
    double x, y;
    Point() {}
    Point(double x, double y) : x(x), y(y) {}
};
typedef Point Vector;
Vector operator + (Vector a, Vector b) {
    return Vector(a.x + b.x, a.y + b.y);
}
Vector operator - (Vector a, Vector b) {
    return Vector(a.x - b.x, a.y - b.y);
}
Vector operator * (Vector v, double mul) {
    return Vector(v.x * mul, v.y * mul);
}
double cross(Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}
struct Segment {
    Point s, e;
    Segment() {}
    Segment(Point s, Point e) : s(s), e(e) {}
    double y(double x) {
        return s.y + (e.y - s.y) * (x - s.x) / (e.x - s.x);
    }
};
bool Segment_intersect(Segment a, Segment b) {
    if (sgn(min(a.s.x, a.e.x) - max(b.s.x, b.e.x)) > 0 ||
            sgn(max(a.s.x, a.e.x) - min(b.s.x, b.e.x)) < 0 ||
            sgn(min(a.s.y, a.e.y) - max(b.s.y, b.e.y)) > 0 ||
            sgn(max(a.s.y, a.e.y) - min(b.s.y, b.e.y)) < 0) return
             ↪ false;
    double t1 = cross(a.e - a.s, b.s - a.s),
           t2 = cross(a.e - a.s, b.e - a.s),
           t3 = cross(b.e - b.s, a.s - b.s),
           t4 = cross(b.e - b.s, a.e - b.s);
    return sgn(t1) * sgn(t2) < 0 && sgn(t3) * sgn(t4) < 0;
}
struct Line {
    Point p;
    Vector v;
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) {}
};
Point Line_intersect(Line a, Line b) {
    double t = cross(b.v, a.p - b.p) / cross(a.v, b.v);
    return a.p + a.v * t;
```

```
}
typedef vector<Point> Polygon;
double Polygon_area(Polygon &poly) {
    double ret = 0;
    for (int i = 1; i < (int)poly.size() - 1; i ++)
        ret += cross(poly[i] - poly[0], poly[i + 1] - poly[0]);
    return fabs(ret);
}
void Polygon_intersect(Polygon &a, Polygon &b, vector<Point> &res) {
    int n = a.size(), m = b.size();
    for (int i = 0; i < n; i ++)
        for (int j = 0; j < m; j ++) {
            if (!Segment_intersect(Segment(a[i], a[(i + 1) % n]),
                                   Segment(b[j], b[(j + 1) % m])))
                                   ↪ continue;
            Point p = Line_intersect(Line(a[i], a[(i + 1) % n] -
            ↪ a[i]),
                                     Line(b[j], b[(j + 1) % m] -
                                     ↪ b[j]));
            res.push_back(p);
        }
}
typedef pair<double, double> Interval;
int merge_Interval(vector<Interval> &itv) {
    sort(itv.begin(), itv.end());
    int ret = 1;
    for (int i = 1; i < (int)itv.size(); i ++) {
        if (sgn(itv[ret - 1].second - itv[i].first) >= 0)
            itv[ret - 1].second = max(itv[ret - 1].second,
            ↪ itv[i].second);
        else itv[ret++] = itv[i];
    }
    itv.resize(ret);
    return ret;
}
double Polygons_area_union(Polygon *poly, int n) {
    vector<double> scan;
    for (int i = 0; i < n; i ++)
        for (auto j : poly[i])
            scan.push_back(j.x);
    for (int i = 0; i < n; i ++) {
        vector<Point> p;
        for (int j = i + 1; j < n; j ++)
            Polygon_intersect(poly[i], poly[j], p);
        for (auto &j : p)
            scan.push_back(j.x);
```

```
    }
    sort(scan.begin(), scan.end());
    scan.resize(unique(scan.begin(), scan.end()) - scan.begin());
    double ret = 0;
    for (int i = 1; i < (int)scan.size(); i ++) {
        double x = (scan[i - 1] + scan[i]) / 2;
        vector<Interval> itv;
        for (int j = 0; j < n; j ++) {
            vector<double> y;
            int m = poly[j].size();
            for (int k = 0; k < m; k ++) {
                Point s = poly[j][k], e = poly[j][(k + 1) % m];
                if (sgn(min(s.x, e.x) - scan[i - 1]) > 0 ||
                        sgn(max(s.x, e.x) - scan[i]) < 0) continue;
                y.push_back(Segment(s, e).y(x));
            }
            if (y.size() < 2) continue;
            sort(y.begin(), y.end());
            for (int k = 0; k < (int)y.size(); k += 2)
                itv.emplace_back(y[k], y[k + 1]);
        }
        if (itv.empty()) continue;
        merge_Interval(itv);
        double sum = 0;
        for (auto &j : itv)
            sum += j.second - j.first;
        ret += sum * (scan[i] - scan[i - 1]);
    }
    return ret;
}
int n;
Polygon poly[MAXN];
int main() {
    scanf("%d", &n);
    double tot = 0;
    for (int i = 0; i < n; i ++) {
        int m;
        scanf("%d", &m);
        while (m--) {
            Point p;
            scanf("%lf%lf", &p.x, &p.y);
            poly[i].push_back(p);
        }
        tot += Polygon_area(poly[i]);
    }
    tot /= 2;
```

```
    double cov = Polygons_area_union(poly, n);
    printf("%.08lf %.08lf\n", tot, cov);
    return 0;
}
```

- Java Version

```
import java.io.*;
import java.awt.geom.Area;
import java.awt.geom.Path2D;
import java.awt.geom.PathIterator;
import java.util.*;
import java.math.*;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        int T = 1;
        for (int i = 0; i < T; i++) solve(in, out);
        out.close();
    }
    public static void solve(InputReader in, PrintWriter out) {
        int n = in.nextInt();// 多边形个数
        Area totalArea = new Area();
        double total = 0.0;
        for(int i = 0; i < n; i++) {
            Path2D.Double pd = new Path2D.Double();
            int m = in.nextInt();
            double a = in.nextDouble(), b = in.nextDouble();
            pd.moveTo(a, b);
            for(int j = 1; j < m; j++) { // 读入第 i 个多边形，这里是
            ↪ 四边形
                a = in.nextDouble();
                b = in.nextDouble();
                pd.lineTo(a, b);
            }
            pd.closePath();
            Area tmpArea = new Area(pd);
            totalArea.add(tmpArea);// 加入总图形
            total += calcArea(tmpArea);// 获得第 i 个多边形的面积
        }
        out.println(total * 0.5 + " " + calcArea(totalArea) * 0.5);
    }
    private static double calcArea(Area area) {
```

```
        double ret = 0;
        PathIterator it = area.getPathIterator(null);// 获得该图形的边界
        double[] buffer = new double[6];
        double last = new double[2], first = null;
        while (!it.isDone()) {
            switch (it.currentSegment(buffer)) {
            case PathIterator.SEG_MOVETO:
            case PathIterator.SEG_LINETO:
                if (first == null) {
                    first = new double[2];
                    first[0] = buffer[0];
                    first[1] = buffer[1];
                } else {
                    ret += last[0] * buffer[1] - last[1] * buffer[0];
                }
                last[0] = buffer[0];
                last[1] = buffer[1];
                break;
            case PathIterator.SEG_CLOSE:
                ret += last[0] * first[1] - last[1] * first[0];
                first = null;
                break;
            }
            it.next();
        }
        if(ret < 0) ret = -ret;
        return ret;
    }

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

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

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens())
            {
                try {
```

```
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

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

        public double nextDouble() {
            return Double.parseDouble(next());
        }

    }
}
```

7.1.6 极角排序

```
typedef ll data_type;
struct point {
    data_type x, y, h;
    point() {}
    inline point(data_type _x, data_type _y) {
        x = _x;
        y = _y;
    }
    inline point operator+(const point &b)const {
        return point(x - b.x, y - b.y);
    }
    inline point operator-(const point &b)const {
        return point(x - b.x, y - b.y);
    }
    inline data_type operator^(const point &b)const {
        return x * b.y - y * b.x;    // 叉乘
    }
    inline data_type operator*(const point &b)const {
        return x * b.x - y * b.y;    // 点乘
    }
    inline bool operator<(const point &b)const {
        return x < b.x || x == b.x && y < b.y;
    }
} p[1001000];
```

```
int n, ans;
inline data_type sqr_dist(const point a, point b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
inline double dist(const point a, point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
inline int Quad(point a) {
    if (a.x > 0 && a.y >= 0)return 1;
    if (a.x <= 0 && a.y > 0)return 2;
    if (a.x < 0 && a.y <= 0)return 3;
    if (a.x >= 0 && a.y < 0)return 4;
}
inline data_type cmp(const point &a, const point &b, const point &c) {
    return (b - a) ^ (c - a);
}
inline bool jijiao_cmp(const point &a, const point &b) { // 顺时针
    if (Quad(a - p[1]) != Quad(b - p[1]))
        return Quad(a - p[1]) < Quad(b - p[1]);
    data_type mmp = cmp(p[1], a, b);
    if (mmp == 0) return sqr_dist(p[1], a) < sqr_dist(p[1], b);;
    return mmp > 0;
}
```

7.1.7 圆的内、外心

```
typedef double datatype;
struct point {
    datatype x, y;
    int id;
    point() {}
    point(datatype _x, datatype _y) {
        x = _x;
        y = _y;
    }
    point(datatype _x, datatype _y, int _id) {
        x = _x;
        y = _y;
        id = _id;
    }
    point operator +(const point &b)const {
        return point(x + b.x, y + b.y);
    }
    point operator -(const point &b)const {
        return point(x - b.x, y - b.y);
    }
    point operator *(const datatype &b)const {
```

```
        return point(b * x, b * y);
    }
    datatype operator *(const point &b)const {
        return x * b.x + y * b.y;
    }
    datatype operator ^(const point &b)const {
        return x * b.y - y * b.x;
    }
};
inline double dist(const point &a, const point &b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
inline point get_neixin(const point &A, const point &B, const point &C) {
    datatype a = dist(B, C), b = dist(A, C), c = dist(A, B);
    return (A * a + B * b + C * c) * (1.0 / (a + b + c));
}
inline point get_waixin(const point &A, const point &B, const point &C) {
    datatype a1 = B.x-A.x, b1 = B.y-A.y, c1 = (a1*a1+b1*b1)/2;
    datatype a2 = C.x-A.x, b2 = C.y-A.y, c2 = (a2*a2+b2*b2)/2;
    datatype d = a1 * b2 - a2 * b1;
    return point(A.x+(c1*b2-c2*b1)/d, A.y+(a1*c2-a2*c1)/d);
}
```

7.1.8 平面最近点对

```
// HDU1007/ZOJ2107
const int MAXN = 100010;
const double eps = 1e-8;
const double INF = 1e20;
struct Point {
    double x, y;
    void input() {
        scanf("%lf%lf", &x, &y);
    }
};
double dist(Point a, Point b) {
    return sqrt((a.x-b.x) * (a.x-b.x) + (a.y-b.y) * (a.y-b.y));
}
Point p[MAXN];
Point tmpt[MAXN];
bool cmpx(Point a, Point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cmpy(Point a, Point b) {
    return a.y < b.y || (a.y == b.y && a.x < b.x);
}
double Closest_Pair(int left, int right) {
```

```
    double d = INF;
    if(left == right)return d;
    if(left + 1 == right)return dist(p[left], p[right]);
    int mid = (left + right) / 2;
    double d1 = Closest_Pair(left, mid);
    double d2 = Closest_Pair(mid + 1, right);
    d = min(d1, d2);
    int cnt = 0;
    for(int i = left; i <= right; i++) {
        if(fabs(p[mid].x - p[i].x) <= d)
            tmpt[cnt++] = p[i];
    }
    sort(tmpt, tmpt + cnt, cmpy);
    for(int i = 0; i < cnt; i++) {
        for(int j = i + 1; j < cnt && tmpt[j].y - tmpt[i].y < d; j++)
            d = min(d, dist(tmpt[i], tmpt[j]));
    }
    return d;
}
int main() {
    int n;
    while(scanf("%d", &n) == 1 && n) {
        for(int i = 0; i < n; i++)p[i].input();
        sort(p, p + n, cmpx);
        printf("%.2lf\n", Closest_Pair(0, n-1));
    }
    return 0;
}
```

7.1.9 最小圆覆盖

```
// 随机增量法
double X, Y;
struct Point {
    double x, y;
} p[M];
struct Triangle {
    Point v[3];
};
struct Circle {
    Point center;
    double r;
};
double pow(double x) {
    return x * x;
}
```

```
double Len(Point a, Point b) {
    return sqrt(pow(a.x - b.x) + pow(a.y - b.y));
}
double TriangleArea(Triangle a) { //求三角形的面积
    double px1 = a.v[1].x - a.v[0].x;
    double py1 = a.v[1].y - a.v[0].y;
    double px2 = a.v[2].x - a.v[0].x;
    double py2 = a.v[2].y - a.v[0].y;
    return fabs(px1 * py2 - px2 * py1) / 2;
}
Circle CircleOfTriangle(Triangle t) { //就三角形外接圆
    Circle tmp;
    double a = Len(t.v[0], t.v[1]);
    double b = Len(t.v[0], t.v[2]);
    double c = Len(t.v[1], t.v[2]);
    tmp.r = a * b * c / 4 / TriangleArea(t);
    double a1 = t.v[1].x - t.v[0].x;
    double b1 = t.v[1].y - t.v[0].y;
    double c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = t.v[2].x - t.v[0].x;
    double b2 = t.v[2].y - t.v[0].y;
    double c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    tmp.center.x = t.v[0].x + (c1 * b2 - c2 * b1) / d;
    tmp.center.y = t.v[0].y + (a1 * c2 - a2 * c1) / d;
    return tmp;
}
void Run(int n) {
    random_shuffle(p + 1, p + n + 1); //随机排序取点
    int i, j, k;
    Circle tep;
    tep.center = p[1];
    tep.r = 0;
    for(i = 2; i <= n; i++) {
        if(Len(p[i], tep.center) > tep.r + eps) {
            tep.center = p[i];
            tep.r = 0;
            for(j = 1; j < i; j++) {
                if(Len(p[j], tep.center) > tep.r + eps) {
                    tep.center.x = (p[i].x + p[j].x) / 2;
                    tep.center.y = (p[i].y + p[j].y) / 2;
                    tep.r = Len(p[i], p[j]) / 2;
                    for(k = 1; k < j; k++) {
                        if(Len(p[k], tep.center) > tep.r + eps) {
                            Triangle t;
                            t.v[0] = p[i];
```

```
                                t.v[1] = p[j];
                                t.v[2] = p[k];
                                tep = CircleOfTriangle(t);
                            }
                        }
                    }
                }
            }
        }
    printf("%.2lf %.2lf %.2lf\n", tep.center.x, tep.center.y, tep.r);
}
int main() {
    int n, i;
    while(scanf("%d", &n), n) {
        for(i = 1; i <= n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        Run(n);
    }
}
// 模拟退火法
double X, Y;
int n;
struct Point {
    double x, y, dis;
    Point() {}
    Point(double xx, double yy) {
        x = xx;
        y = yy;
    }
    bool check() {
        if(x > 0 && y > 0 && x < X && y < Y)return true;
        return false;
    }
} p[M], q[50];
double pow(double x) {
    return x * x;
}
double Len(Point a, Point b) {
    return sqrt(pow(a.x - b.x) + pow(a.y - b.y));
}
double fun(Point a) {
    double maxi = 0;
    for(int i = 1; i <= n; i++) {
        double L = Len(a, p[i]);
        if(maxi < L)
            maxi = L;
```

```
    }
    return maxi;
}
int main() {
    int i;
    while(scanf("%lf%lf%d", &X, &Y, &n) != EOF) {
        for(i = 1; i <= n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        int po = 15, est = 15;
        for(int i = 1; i <= po; i++) {
            q[i].x = (rand() % 1000 + 1) / 1000.0 * X;
            q[i].y = (rand() % 1000 + 1) / 1000.0 * Y;
            q[i].dis = fun(q[i]);
        }
        double temp = max(X, Y);
        while(temp > 0.001) {
            for(int i = 1; i <= po; i++) {
                for(int j = 1; j <= est; j++) {
                    double rad = (rand() % 1000 + 1) / 1000.0 * PI * 2;
                    Point cur;
                    cur.x = q[i].x + temp * cos(rad);
                    cur.y = q[i].y + temp * sin(rad);
                    if(!cur.check())continue;
                    cur.dis = fun(cur);
                    if(cur.dis < q[i].dis)
                        q[i] = cur;
                }
            }
            temp *= 0.8;
        }
        int id = 1;
        for(int i = 1; i <= po; i++)
            if(q[id].dis > q[i].dis)
                id = i;
        printf("(%.1lf,%.1lf).\n%.1lf\n", q[id].x, q[id].y, q[id].dis);
    }
    return 0;
}
```

7.1.10 自适应辛普森积分

```
double f(double x) {
    // Coding this
}
double simpson(double a, double b, double fl, double fr, double fm) {
    return (b - a) / 6 * (fl + 4 * fm + fr);
```

```
}
double integral(double l, double r, double fl, double fr, double fm, double EPS) {
    double m = (l + r) * 0.5;
    double flm = f((l + m) * 0.5), frm = f((m + r) * 0.5);
    double itg, itg_l, itg_r;
    itg = simpson(l, r, fl, fr, fm);
    itg_l = simpson(l, m, fl, fm, flm);
    itg_r = simpson(m, r, fm, fr, frm);
    if (fabs(itg_l + itg_r - itg) <= EPS)return itg;
    return integral(l, m, fl, fm, flm) + integral(m, r, fm, fr, frm);
}
double calc(double l, double r, double EPS) {
    return integral(l, r, f(l), f(r), f((l + r) * 0.5), EPS);
}
```

7.2 三维几何

```
const double eps = 1e-8;
int sgn(double x) {
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point3 {
    double x, y, z;
    Point3(double _x = 0, double _y = 0, double _z = 0) {
        x = _x;
        y = _y;
        z = _z;
        z = _z;
    }
    void input() {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    void output() {
        scanf("%.2lf %.2lf %.2lf\n", x, y, z);
    }
    bool operator ==(const Point3 &b)const {
        return sgn(x - b.x) == 0 && sgn(y - b.y) == 0 && sgn(z - b.z) == 0;
    }
    bool operator <(const Point3 &b)const {
        return sgn(x - b.x) == 0 ? (sgn(y - b.y) == 0 ? sgn(z - b.z) < 0 : y < b.y) : x < b.x;
    }
```

```
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
    double len2() {
        return x * x + y * y + z * z;
    }
    double distance(const Point3 &b)const {
        return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y - b.y) + (z -
         ↪ b.z) * (z - b.z));
    }
    Point3 operator -(const Point3 &b)const {
        return Point3(x - b.x, y - b.y, z - b.z);
    }
    Point3 operator +(const Point3 &b)const {
        return Point3(x + b.x, y + b.y, z + b.z);
    }
    Point3 operator *(const double &k)const {
        return Point3(x * k, y * k, z * k);
    }
    Point3 operator /(const double &k)const {
        return Point3(x / k, y / k, z / k);
    }
    // 点乘
    double operator *(const Point3 &b)const {
        return x * b.x + y * b.y + z * b.z;
    }
    // 叉乘
    Point3 operator ^(const Point3 &b)const {
        return Point3(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y *
         ↪ b.x);
    }
    double rad(Point3 a, Point3 b) {
        Point3 p = (*this);
        return acos( ( (a - p) * (b - p) ) / (a.distance(p) *
         ↪ b.distance(p)) );
    }
    // 变换长度
    Point3 trunc(double r) {
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point3(x * r, y * r, z * r);
    }
};
struct Line3 {
    Point3 s, e;
```

```
    Line3() {}
    Line3(Point3 _s, Point3 _e) {
        s = _s;
        e = _e;
    }
    bool operator ==(const Line3 v) {
        return (s == v.s) && (e == v.e);
    }
    void input() {
        s.input();
        e.input();
    }
    double length() {
        return s.distance(e);
    }
    // 点到直线距离
    double dispointtoline(Point3 p) {
        return ((e - s) ^ (p - s)).len() / s.distance(e);
    }
    // 点到线段距离
    double dispointtoseg(Point3 p) {
        if(sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.distance(s), e.distance(p));
        return dispointtoline(p);
    }
    // 返回点 p 在直线上的投影
    Point3 lineprog(Point3 p) {
        return s + ( ((e - s) * ((e - s) * (p - s))) / ((e - s).len2()) );
    }
    // p 绕此向量逆时针 arg 角度
    Point3 rotate(Point3 p, double ang) {
        if(sgn(((s - p) ^ (e - p)).len()) == 0)return p;
        Point3 f1 = (e - s) ^ (p - s);
        Point3 f2 = (e - s) ^ (f1);
        double len = ((s - p) ^ (e - p)).len() / s.distance(e);
        f1 = f1.trunc(len);
        f2 = f2.trunc(len);
        Point3 h = p + f2;
        Point3 pp = h + f1;
        return h + ((p - h) * cos(ang)) + ((pp - h) * sin(ang));
    }
    // 点在直线上
    bool pointonseg(Point3 p) {
        return sgn(((s - p) ^ (e - p)).len()) == 0 && sgn((s - p) * (e - p)) == 0;
```

```
    }
};
struct Plane {
    Point3 a, b, c, o; // 平面上的三个点，以及法向量
    Plane() {}
    Plane(Point3 _a, Point3 _b, Point3 _c) {
        a = _a;
        b = _b;
        c = _c;
        o = pvec();
    }
    Point3 pvec() {
        return (b - a) ^ (c - a);
    }
    // ax+by+cz+d = 0
    Plane(double _a, double _b, double _c, double _d) {
        o = Point3(_a, _b, _c);
        if(sgn(_a) != 0)
            a = Point3((-_d - _c - _b) / _a, 1, 1);
        else if(sgn(_b) != 0)
            a = Point3(1, (-_d - _c - _a) / _b, 1);
        else if(sgn(_c) != 0)
            a = Point3(1, 1, (-_d - _a - _b) / _c);
    }
    // 点在平面上的判断
    bool pointonplane(Point3 p) {
        return sgn((p - a) * o) == 0;
    }
    // 两平面夹角
    double angleplane(Plane f) {
        return acos(o * f.o) / (o.len() * f.o.len());
    }
    // 平面和直线的交点，返回值是交点个数
    int crossline(Line3 u, Point3 &p) {
        double x = o * (u.e-a);
        double y = o * (u.s - a);
        double d = x - y;
        if(sgn(d) == 0)return 0;
        p = ((u.s * x) - (u.e * y)) / d;
        return 1;
    }
    // 点到平面最近点 (也就是投影)
    Point3 pointtoplane(Point3 p) {
        Line3 u = Line3(p, p + o);
        crossline(u, p);
        return p;
```

```
    }
    // 平面和平面的交线
    int crossplane(Plane f, Line3 &u) {
        Point3 oo = o ^ f.o;
        Point3 v = o ^ oo;
        double d = fabs(f.o * v);
        if(sgn(d) == 0)return 0;
        Point3 q = a + (v * (f.o * (f.a - a)) / d);
        u = Line3(q, q + oo);
        return 1;
    }
};
```

7.2.1 通用函数

```
const double eps = 1e-8;
int sgn(double x) {
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point3 {
    double x, y, z;
    Point3(double _x = 0, double _y = 0, double _z = 0) {
        x = _x;
        y = _y;
        z = _z;
        z = _z;
    }
    void input() {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    void output() {
        scanf("%.2lf %.2lf %.2lf\n", x, y, z);
    }
    bool operator ==(const Point3 &b)const {
        return sgn(x - b.x) == 0 && sgn(y - b.y) == 0 && sgn(z - b.z) ==
        ↪ 0;
    }
    bool operator <(const Point3 &b)const {
        return sgn(x - b.x) == 0 ? (sgn(y - b.y) == 0 ? sgn(z - b.z) < 0
        ↪ : y < b.y) : x < b.x;
    }
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
    double len2() {
```

```
        return x * x + y * y + z * z;
    }
    double distance(const Point3 &b)const {
        return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y - b.y) + (z -
        ↪  b.z) * (z - b.z));
    }
    Point3 operator -(const Point3 &b)const {
        return Point3(x - b.x, y - b.y, z - b.z);
    }
    Point3 operator +(const Point3 &b)const {
        return Point3(x + b.x, y + b.y, z + b.z);
    }
    Point3 operator *(const double &k)const {
        return Point3(x * k, y * k, z * k);
    }
    Point3 operator /(const double &k)const {
        return Point3(x / k, y / k, z / k);
    }
    // 点乘
    double operator *(const Point3 &b)const {
        return x * b.x + y * b.y + z * b.z;
    }
    // 叉乘
    Point3 operator ^(const Point3 &b)const {
        return Point3(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y *
        ↪  b.x);
    }
    double rad(Point3 a, Point3 b) {
        Point3 p = (*this);
        return acos( ( (a - p) * (b - p) ) / (a.distance(p) *
        ↪  b.distance(p)) );
    }
    // 变换长度
    Point3 trunc(double r) {
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point3(x * r, y * r, z * r);
    }
};
struct Line3 {
    Point3 s, e;
    Line3() {}
    Line3(Point3 _s, Point3 _e) {
        s = _s;
        e = _e;
```

```
    }
    bool operator ==(const Line3 v) {
        return (s == v.s) && (e == v.e);
    }
    void input() {
        s.input();
        e.input();
    }
    double length() {
        return s.distance(e);
    }
    // 点到直线距离
    double dispointtoline(Point3 p) {
        return ((e - s) ^ (p - s)).len() / s.distance(e);
    }
    // 点到线段距离
    double dispointtoseg(Point3 p) {
        if(sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.distance(s), e.distance(p));
        return dispointtoline(p);
    }
    // 返回点 p 在直线上的投影
    Point3 lineprog(Point3 p) {
        return s + ( ((e - s) * ((e - s) * (p - s))) / ((e - s).len2())
         ↪ );
    }
    // p 绕此向量逆时针 arg 角度
    Point3 rotate(Point3 p, double ang) {
        if(sgn(((s - p) ^ (e - p)).len()) == 0)return p;
        Point3 f1 = (e - s) ^ (p - s);
        Point3 f2 = (e - s) ^ (f1);
        double len = ((s - p) ^ (e - p)).len() / s.distance(e);
        f1 = f1.trunc(len);
        f2 = f2.trunc(len);
        Point3 h = p + f2;
        Point3 pp = h + f1;
        return h + ((p - h) * cos(ang)) + ((pp - h) * sin(ang));
    }
    // 点在直线上
    bool pointonseg(Point3 p) {
        return sgn(((s - p) ^ (e - p)).len()) == 0 && sgn((s - p) * (e -
         ↪ p)) == 0;
    }
};
struct Plane {
    Point3 a, b, c, o; // 平面上的三个点，以及法向量
```

```
    Plane() {}
    Plane(Point3 _a, Point3 _b, Point3 _c) {
        a = _a;
        b = _b;
        c = _c;
        o = pvec();
    }
    Point3 pvec() {
        return (b - a) ^ (c - a);
    }
    // ax+by+cz+d = 0
    Plane(double _a, double _b, double _c, double _d) {
        o = Point3(_a, _b, _c);
        if(sgn(_a) != 0)
            a = Point3((-_d - _c - _b) / _a, 1, 1);
        else if(sgn(_b) != 0)
            a = Point3(1, (-_d - _c - _a) / _b, 1);
        else if(sgn(_c) != 0)
            a = Point3(1, 1, (-_d - _a - _b) / _c);
    }
    // 点在平面上的判断
    bool pointonplane(Point3 p) {
        return sgn((p - a) * o) == 0;
    }
    // 两平面夹角
    double angleplane(Plane f) {
        return acos(o * f.o) / (o.len() * f.o.len());
    }
    // 平面和直线的交点，返回值是交点个数
    int crossline(Line3 u, Point3 &p) {
        double x = o * (u.e-a);
        double y = o * (u.s - a);
        double d = x - y;
        if(sgn(d) == 0)return 0;
        p = ((u.s * x) - (u.e * y)) / d;
        return 1;
    }
    // 点到平面最近点（也就是投影）
    Point3 pointtoplane(Point3 p) {
        Line3 u = Line3(p, p + o);
        crossline(u, p);
        return p;
    }
    // 平面和平面的交线
    int crossplane(Plane f, Line3 &u) {
        Point3 oo = o ^ f.o;
```

```
        Point3 v = o ^ oo;
        double d = fabs(f.o * v);
        if(sgn(d) == 0)return 0;
        Point3 q = a + (v * (f.o * (f.a - a)) / d);
        u = Line3(q, q + oo);
        return 1;
    }
};
```

7.2.2 三维凸包

```
// HDU 4273 给一个三维凸包，求重心到表面的最短距离。
const double eps = 1e-8;
const int MAXN = 550;
int sgn(double x) {
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point3 {
    double x, y, z;
    Point3(double _x = 0, double _y = 0, double _z = 0) {
        x = _x;
        y = _y;
        z = _z;
    }
    void input() {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    bool operator ==(const Point3 &b)const {
        return sgn(x - b.x) == 0 && sgn(y - b.y) == 0 && sgn(z - b.z) ==
         ↪  0;
    }
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
    double len2() {
        return x * x + y * y + z * z;
    }
    double distance(const Point3 &b)const {
        return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y - b.y) + (z -
         ↪  b.z) * (z - b.z));
    }
    Point3 operator -(const Point3 &b)const {
        return Point3(x - b.x, y - b.y, z - b.z);
    }
    Point3 operator +(const Point3 &b)const {
```

```
        return Point3(x + b.x, y + b.y, z + b.z);
    }
    Point3 operator *(const double &k)const {
        return Point3(x * k, y * k, z * k);
    }
    Point3 operator /(const double &k)const {
        return Point3(x / k, y / k, z / k);
    }
    // 点乘
    double operator *(const Point3 &b)const {
        return x * b.x + y * b.y + z * b.z;
    }
    // 叉乘
    Point3 operator ^(const Point3 &b)const {
        return Point3(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y *
        ↪  b.x);
    }
};
struct CH3D {
    struct face {
        // 表示凸包一个面上的三个点的编号
        int a, b, c;
        // 表示该面是否属于最终的凸包上的面
        bool ok;
    };
    // 初始顶点数
    int n;
    Point3 P[MAXN];
    // 凸包表面的三角形数
    int num;
    // 凸包表面的三角形
    face F[8 * MAXN];
    int g[MAXN][MAXN];
    // 叉乘
    Point3 cross(const Point3 &a, const Point3 &b, const Point3 &c) {
        return (b - a) ^ (c - a);
    }
    // 三角形面积 *2
    double area(Point3 a, Point3 b, Point3 c) {
        return ((b - a) ^ (c - a)).len();
    }
    // 四面体有向面积 *6
    double volume(Point3 a, Point3 b, Point3 c, Point3 d) {
        return ((b - a) ^ (c - a)) * (d - a);
    }
    // 正：点在面同向
```

```
    double dblcmp(Point3 &p, face &f) {
        Point3 p1 = P[f.b] - P[f.a];
        Point3 p2 = P[f.c] - P[f.a];
        Point3 p3 = p - P[f.a];
        return (p1 ^ p2) * p3;
    }
    void deal(int p, int a, int b) {
        int f = g[a][b];
        face add;
        if(F[f].ok) {
            if(dblcmp(P[p], F[f]) > eps)
                dfs(p, f);
            else {
                add.a = b;
                add.b = a;
                add.c = p;
                add.ok = true;
                g[p][b] = g[a][p] = g[b][a] = num;
                F[num++] = add;
            }
        }
    }
    // 递归搜索所有应该从凸包内删除的面
    void dfs(int p, int now) {
        F[now].ok = false;
        deal(p, F[now].b, F[now].a);
        deal(p, F[now].c, F[now].b);
        deal(p, F[now].a, F[now].c);
    }
    bool same(int s, int t) {
        Point3 &a = P[F[s].a];
        Point3 &b = P[F[s].b];
        Point3 &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps &&
               fabs(volume(a, b, c, P[F[t].b])) < eps &&
               fabs(volume(a, b, c, P[F[t].c])) < eps;
    }
    // 构建三维凸包
    void create() {
        num = 0;
        face add;
        // ***********************************
        // 此段是为了保证前四个点不共面
        bool flag = true;
        for(int i = 1; i < n; i++) {
            if(!(P[0] == P[i])) {
```

```
                swap(P[1], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        flag = true;
        for(int i = 2; i < n; i++) {
            if( ((P[1] - P[0]) ^ (P[i] - P[0])).len() > eps ) {
                swap(P[2], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        flag = true;
        for(int i = 3; i < n; i++) {
            if(fabs( ((P[1] - P[0]) ^ (P[2] - P[0])) * (P[i] - P[0]) ) > eps) {
                swap(P[3], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        // *********************************
        for(int i = 0; i < 4; i++) {
            add.a = (i + 1) % 4;
            add.b = (i + 2) % 4;
            add.c = (i + 3) % 4;
            add.ok = true;
            if(dblcmp(P[i], add) > 0)swap(add.b, add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] =
                                                    num;
            F[num++] = add;
            for(int i = 4; i < n; i++)
                for(int j = 0; j < num; j++)
                    if(F[j].ok && dblcmp(P[i], F[j]) > eps) {
                        dfs(i, j);
                        break;
                    }
            int tmp = num;
            num = 0;
            for(int i = 0; i < tmp; i++)
                if(F[i].ok)
                    F[num++] = F[i];
```

```
        }
    }
    // 表面积
    // 测试：HDU3528
    double area() {
        double res = 0;
        if(n == 3) {
            Point3 p = cross(P[0], P[1], P[2]);
            return p.len() / 2;
        }
        for(int i = 0; i < num; i++)
            res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res / 2.0;
    }
    double volume() {
        double res = 0;
        Point3 tmp = Point3(0, 0, 0);
        for(int i = 0; i < num; i++)
            res += volume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
        return fabs(res / 6);
    }
    // 表面三角形个数
    int triangle() {
        return num;
    }
    // 表面多边形个数
    // 测试：HDU3662
    int polygon() {
        int res = 0;
        for(int i = 0; i < num; i++) {
            bool flag = true;
            for(int j = 0; j < i; j++)
                if(same(i, j)) {
                    flag = 0;
                    break;
                }
            res += flag;
        }
        return res;
    }
    // 重心
    // 测试：HDU4273
    Point3 barycenter() {
        Point3 ans = Point3(0, 0, 0);
        Point3 o = Point3(0, 0, 0);
        double all = 0;
```

```
        for(int i = 0; i < num; i++) {
            double vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
            ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4.0)
             ↪  * vol);
            all += vol;
        }
        ans = ans / all;
        return ans;
    }
    // 点到面的距离
    // 测试：HDU4273
    double ptoface(Point3 p, int i) {
        double tmp1 = fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p));
        double tmp2 = ((P[F[i].b] - P[F[i].a]) ^ (P[F[i].c] -
         ↪  P[F[i].a])).len();
        return tmp1 / tmp2;
    }
};
CH3D hull;
int main() {
    while(scanf("%d", &hull.n) == 1) {
        for(int i = 0; i < hull.n; i++)hull.P[i].input();
        hull.create();
        Point3 p = hull.barycenter();
        double ans = 1e20;
        for(int i = 0; i < hull.num; i++)
            ans = min(ans, hull.ptoface(p, i));
        printf("%.3lf\n", ans);
    }
    return 0;
}
```

7.2.3 最小球覆盖

```
/* 最小球的球心，它必然处于一个稳定态，
 * 也就是与它距离最远的点最多有 4 个且等距离。
 * 于是，我们首先任选一个点作为球心，
 * 并找到点集中与它距离最远的点，我们让球心靠近最远的点，
 * 不断重复此过程，就可以让球心达到稳定态了！此时我们就找到了最小球。
 */
const double eps = 1e-7;
struct point3D {
    double x, y, z;
} data[35];
int n;
double dis(point3D a, point3D b) {
```

```
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) +
     ↪  (a.z - b.z) * (a.z - b.z));
}
double solve() {
    double step = 100, ans = 1e30, mt;
    point3D z;
    z.x = z.y = z.z = 0;
    int s = 0;
    while(step > eps) {
        for(int i = 0; i < n; i++)
            if(dis(z, data[s]) < dis(z, data[i])) s = i;
        mt = dis(z, data[s]);
        ans = min(ans, mt);
        z.x += (data[s].x - z.x) / mt * step;
        z.y += (data[s].y - z.y) / mt * step;
        z.z += (data[s].z - z.z) / mt * step;
        step *= 0.98;
    }
    return ans;
}
int main() {
    double ans;
    while(~scanf("%d", &n), n) {
        for(int i = 0; i < n; i++)
            scanf("%lf%lf%lf", &data[i].x, &data[i].y, &data[i].z);
        ans = solve();
        printf("%.5f\n", ans);
    }
    return 0;
}
```

8 博弈论

- **游戏图:** 把每一个状态抽象成图中一个点, 状态 u 能走到 v 就连一条有向边 $u \to v$
  - 出度为 0 的点是先手必败态
  - 每一次对弈都是图上一条走到底的路径
- **博弈:** 两人参与游戏，轮流做出决策，每次决策都是对自己最有利的，无法决策时输
  - 游戏会在有限步内结束
  - 同一个局面不可能多次出现，不会有平局
  - 某一个确定状态下当前决策者能走的步只和当前局面有关，和游戏者无关
- SG **函数:** $sg(v) = mex\{sg(u) | \exists\ edge\ v \to u\}$ ，$mex\{A\}$ 表示不在集合 A 中出现的最小非负整数
  - $sg = 0$ 的状态先手必败。否则先手必胜。
  - 出度为 0 的点是先手必败态 $sg = 0$，然后倒推即可
  - $O(max\{sg\}\cdot$ 状态数$)$ 可处理每个点的 sg，对于游戏的组合太慢。多个游戏组合的状态数是笛卡儿积相乘
  - 对于游戏图非 DAG 的不能计算 sg 函数，因为有环，可能永不结束。不用 sg 函数直接计算会结束的情况的胜负即可
- **游戏的组合:** 多个游戏同时存在，双方每次选择在其中一个游戏里进行一次操作
  - 会互相干扰的多个游戏不能组合，应当用笛卡儿积表示每个状态
  - 游戏的组合的 sg 值等于各个游戏的 sg 值的异或
- $Anti-SG$ **游戏:** 如果定义无法决策的人赢，即走了最后一步的人输，其他完全一样，判断先手必胜或必败
  - SJ 定理：定义 $mark_1 = (sg_{[all\ games]} = 0), mark_2 = (\exists sg_{[one\ of\ game]} > 1)$，则 $Anti-SG$ 先手必胜 $= mark_1\ \oplus\ mark_2$
- $Multi-SG$ **游戏:** 一个单一游戏的后继可以是多个单一游戏的并（就是现在可以把一堆石子分成多堆了）
- $Multi-SG$ **游戏:** 一个单一游戏的后继可以是多个单一游戏的并（就是现在可以把一堆石子分成多堆了）

  没有做法，打表找规律。对于特例。一次分两堆的 nim 游戏:

$$sg[x] = \begin{cases} x, & x\%4 = 1,2 \\ x+1, & x\%4 = 3 \\ x-1 & x\%4 = 0 \end{cases}$$

- **斐波那契博弈:** 如果定义无法决策的人赢，即走了最后一步的人输，其他完全一样，判断先手必胜或必败

  - 先手必胜当且仅当石子数是斐波那契数

- **威佐夫博弈:** 两堆石子，每次可以在一堆取任意多或者在两堆取同样多
  - 先手必败当且仅当状态 (x,y) 或 (y,x) 属于 $(\lfloor kp\rfloor, \lfloor kp^2\rfloor)$，其中 $p=\frac{1+\sqrt{5}}{2}$
- **巴什博弈:** 一堆石子，每次最多取 k 个
  - 先手必胜当且仅当答案不是 $k+1$ 倍数

9 其他

9.1 快速输入输出

```cpp
#include <bits/stdc++.h>
using namespace std;
namespace fastIO {
#define BUF_SIZE 10000
#define OUT_SIZE 10000
//fread -> read
bool IOerror = 0;
inline char nc() {static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf + BUF_SIZE;if(p1 == pend) {p1 = buf;pend = buf + fread(buf, 1, BUF_SIZE, stdin);if(pend == p1) {IOerror = 1;return -1;}} return *p1++;}
inline bool blank(char ch) {return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';}
inline void read(int &x) {char ch;int f = 1;while(blank(ch = nc()));while(ch == '-')f = -f, ch = nc();if(IOerror)return;for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');x *= f;}
inline void read(char *s) {char ch;while(blank(ch = nc()));if(IOerror)return;int i = 0;for (; !blank(ch); s[i++] = ch, ch = nc());s[i] = '\0';}
//fwrite
struct Ostream_fwrite{
        char *buf,*p1,*pend;
        Ostream_fwrite(){buf=new char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
        void out(char ch){if (p1==pend){fwrite(buf,1,BUF_SIZE,stdout);p1=buf;}*p1++=ch;}
        void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
    ~Ostream_fwrite(){flush();}
}Ostream;
inline void print(char x){Ostream.out(x);}
inline void println(){Ostream.out('\n');}
inline void flush(){Ostream.flush();}
char Out[OUT_SIZE],*o=Out;
inline void print1(char c){*o++=c;}
inline void println1(){*o++='\n';}
inline void flush1(){if (o!=Out){if (*(o-1)=='\n')*--o=0;puts(Out);}}
struct puts_write{~puts_write(){flush1();}}_puts;
};
using namespace fastIO;
```

9.2 程序对拍（Windows 环境、Linux 环境）

```
/* 此文件 和 makedata.cpp test.cpp baoli.cpp 放在同一文件夹下
 * makedata.cpp: 输出一组样例，不要开文件输入输出
 * test.cpp: 用来测试正确性的代码，不要开文件输入输出
 * baoli.cpp: 暴力代码，不要开文件输入输出
 * 1.in 自动生成的输入数据
 * 1.out 待测试代码跑出来的结果
 * 2.out 暴力跑出来的结果
 * 注意输出结果会全文比较
 */
#include <bits/stdc++.h>
using namespace std;
int main() {
    system("g++ -std=c++11 makedata.cpp -o makedata");
    system("g++ -std=c++11 test.cpp -o test");
    system("g++ -std=c++11 baoli.cpp -o baoli");
    while (1) {
        system("makedata > 1.in");
        system("test < 1.in > 1.out");
        system("baoli < 1.in > 2.out");
        // if (system("diff 1.out 2.out")) /* for linux */
        if (system("fc 1.out 2.out")) { /* for windows */
            getchar();
        }
    }
}
```

9.3 快速排序

```
#include<bits/stdc++.h>
using namespace std;
void quickSort(int a[],int left,int right) {
        int i=left;
        int j=right;
        if(left>=right)return;
        swap(a[left],a[left+rand()%(right-left+1)]); //随机选择基准防止原数据有序
        int temp=a[left]; //temp 代表选择的基准
        while(i!=j) {
                while(i<j&&a[j]>=temp)j--; //一定要先从后到前找第一个比基准要大的
                if(j>i)a[i]=a[j];
                //a[i] 已经赋值给 temp，所以直接将 a[j] 赋值给 a[i]，赋值完之后 a[j]，有空位
                while(i<j&&a[i]<=temp)i++; //从前到后找第一个比基准要小的
```

```
                if(i<j)a[j]=a[i];
        }
        a[i]=temp;  //把基准插入, 此时 i 与 j 已经相等
        // R[low..pivotpos-1].keys <=
        ↪  R[pivotpos].key R[pivotpos+1..high].keys
        quickSort(a,left,i-1); //递归左边
        quickSort(a,i+1,right); //递归右边
}
int a[1000010];
int main() {
        srand(time(0));
        int n,m;
        while(~scanf("%d%d",&n,&m)){
                for(int i=1;i<=n;i++)scanf("%d",&a[i]);
                quickSort(a,1,n);
                printf("%d\n",a[m]);
        }
        return 0;
}
```

9.4 桶排序

```
// 十进制按位桶排 O(nk), k 是 log_10_max{a[i]}
#define inf 0x7fffffff
#define N 100010
#define max(a,b) ((a)>(b)?(a):(b))
int rk[N][2], sa[N][2];
int a[N], b[N];
void bucket_sort(int l, int r) {
    if (l >= r)return;
    int sz[10] = {0}, tot = r - l + 1, i;
    int bin = 1, mx = -inf;
    for (i = 1; i <= tot; i++)b[i] = a[i + l - 1], mx = max(mx, b[i]);
    int pre = 0, cur = 1;
    while (bin <= mx) {
        memset(sz, 0, sizeof(sz));
        for (i = 1; i <= tot; i++)sz[(b[i] / bin) % 10]++;
        for (i = 1; i <= 9; i++)sz[i] += sz[i - 1];
        if (bin == 1) {
            for (i = tot; i >= 1; i--) {
                rk[i][cur] = sz[(b[i] / bin) % 10]--;
                sa[rk[i][cur]][cur] = i;
            }
        } else for (i = tot; i >= 1; i--) {
                int now = sa[i][pre];
                rk[now][cur] = sz[(b[now] / bin) % 10]--;
```

```
                sa[rk[now][cur]][cur] = now;
            }
        bin *= 10;
        pre ^= 1;
        cur ^= 1;
    }
    for (i = l; i <= r; i++)a[i] = b[sa[i - l + 1][pre]];
}
```

9.5 分治查询第 K 大

```
#include<bits/stdc++.h>
using namespace std;
int quickSort(int a[],int left,int right,int k) {
        int i=left;
        int j=right;
        swap(a[left],a[left+rand()%(right-left+1)]); //随机选择基准防止原数据有序
        int temp=a[left]; //temp 代表选择的基准
        while(i!=j) {
                while(i<j&&a[j]>=temp)j--; //一定要先从后到前找第一个比基准要大的
                if(j>i)a[i]=a[j];
                //a[i] 已经赋值给 temp，所以直接将 a[j] 赋值给 a[i]，赋值完之后 a[j]，有空位
                while(i<j&&a[i]<=temp)i++; //从前到后找第一个比基准要小的
                if(i<j)a[j]=a[i];
        }
        a[i]=temp; //把基准插入，此时 i 与 j 已经相等
        //R[low..pivotpos-1].keys R[pivotpos].key<=R[pivotpos+1..high].keys
        if(k+left-1<i) return quickSort(a,left,i-1,k); //查询左边
        else if(k+left-1>i)return quickSort(a,i+1,right,k-(i-left+1)); //查询右边
        else return temp;
}
int a[1000010];
int main() {
        srand(time(0));
        int n,m;
        while(~scanf("%d%d",&n,&m)){
                for(int i=1;i<=n;i++)scanf("%d",&a[i]);
                printf("%d\n", quickSort(a,1,n,m));
        }
        return 0;
}
```

9.6 最大全 0 子矩阵 $O(n^2)$

```
int a[2010][2010];
int h[2010][2010];
int n, m, ans;
int zhan[2010], l[2010], r[2010], top;
int main() {
    n = read();
    for (int i = 1; i <= n; i++)for (int j = 1; j <= n; j++)a[i][j] =
     ↪ read();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            h[i][j] = a[i][j] == 0 ? 1 + h[i - 1][j] : 0;
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            while (top && h[i][zhan[top]] > h[i][j]) {
                ans = max(ans, h[i][zhan[top]] * (j - r[top - 1] - 1));
                top--;
            }
            if (top && h[i][zhan[top]] == h[i][j]) {
                r[top] = j;
                continue;
            }
            zhan[++top] = j;
            l[top] = r[top] = j;
        }
        while (top) {
            ans = max(ans, h[i][zhan[top]] * (n - r[top - 1]));
            top--;
        }
    }
    printf("%d\n", ans);
}
```

9.7 高精度

```
struct BigInt {
    const static int mod = 10000;
    const static int LEN = 4;
    int a[maxn], len;
    BigInt() {
        memset(a, 0, sizeof(a));
        len = 1;
    }
    void init(int x) {
        memset(a, 0, sizeof(a));
```

```
        len = 0;
        do {
            a[len++] = x % mod;
            x /= mod;
        } while(x);
    }
    int Compare(const BigInt &b) {
        if(len < b.len)return -1;
        if(len > b.len)return 1;
        for(int i = len - 1; i >= 0; i--)
            if(a[i] < b.a[i])return -1;
            else if(a[i] > b.a[i])return 1;
        return 0;
    }
    BigInt operator +(const BigInt &b)const {
        BigInt ans;
        ans.len = max(len, b.len);
        for(int i = 0; i <= ans.len; i++)ans.a[i] = 0;
        for(int i = 0; i < ans.len; i++) {
            ans.a[i] += ((i < len) ? a[i] : 0) + ((i < b.len) ? b.a[i] :
             ↪  0);
            ans.a[i + 1] += ans.a[i] / mod;
            ans.a[i] %= mod;
        }
        if(ans.a[ans.len] > 0)ans.len++;
        return ans;
    }
    BigInt operator -(const BigInt &b)const {
        BigInt ans;
        ans.len = len;
        int k = 0;
        for(int i = 0; i < ans.len; i++) {
            ans.a[i] = a[i] + k - b.a[i];
            if(ans.a[i] < 0)ans.a[i] += mod, k = -1;
            else k = 0;
        }
        while(ans.a[ans.len - 1] == 0 && ans.len > 1)ans.len--;
        return ans;
    }
    BigInt operator *(const BigInt &b)const {
        BigInt ans;
        for(int i = 0; i < len; i++) {
            int k = 0;
            for(int j = 0; j < b.len; j++) {
                int temp = a[i] * b.a[j] + ans.a[i + j] + k;
                ans.a[i + j] = temp % mod;
```

```
                k = temp / mod;
            }
            if(k != 0)ans.a[i + b.len] = k;
        }
        ans.len = len + b.len;
        while(ans.a[ans.len - 1] == 0 && ans.len > 1)ans.len--;
        return ans;
    }
    void output() {
        printf("%d", a[len - 1]);
        for(int i = len - 2; i >= 0; i--)
            printf("%04d", a[i]);
        printf("\n");
    }
} ans[121];
```

9.8 完全高精度

```
/* 完全大数模板
 * 输入 cin>>a
 * 输出 a.print();
 * 注意这个输入不能自动去掉前导 0 的，可以先读入到 char 数组，去掉前导 0，
 ↪ 再用构造函数。
 */
#define MAXN 9999
#define MAXSIZE 1010
#define DLEN 4
class BigNum {
private:
    int a[500]; //可以控制大数的位数
    int len;
public:
    BigNum() {
        len = 1;     //构造函数
        memset(a, 0, sizeof(a));
    }
    BigNum(const int); //将一个 int 类型的变量转化成大数
    BigNum(const char *); //将一个字符串类型的变量转化为大数
    BigNum(const BigNum &); //拷贝构造函数
    BigNum &operator=(const BigNum &); //重载赋值运算符，大数之间进行赋值
    ↪ 运算
    friend istream &operator>>(istream &, BigNum &); //重载输入运算符
    friend ostream &operator<<(ostream &, BigNum &); //重载输出运算符
    BigNum operator+(const BigNum &)const; //重载加法运算符，两个大数之间
    ↪ 的相加运算
```

```
    BigNum operator-(const BigNum &)const; //重载减法运算符，两个大数之间的相减运算
    BigNum operator*(const BigNum &)const; //重载乘法运算符，两个大数之间的相乘运算
    BigNum operator/(const int &)const; //重载除法运算符，大数对一个整数进行相除运算

    BigNum operator^(const int &)const; //大数的 n 次方运算
    int operator%(const int &)const; //大数对一个类型的变量进行取模运算int
    bool operator>(const BigNum &T)const; //大数和另一个大数的大小比较
    bool operator>(const int &t)const; //大数和一个 int 类型的变量的大小比较

    void print(); //输出大数
};
//将一个 int 类型的变量转化为大数
BigNum::BigNum(const int b) {
    int c, d = b;
    len = 0;
    memset(a, 0, sizeof(a));
    while(d > MAXN) {
        c = d-(d / (MAXN + 1)) * (MAXN + 1);
        d = d / (MAXN + 1);
        a[len++] = c;
    }
    a[len++] = d;
}
//将一个字符串类型的变量转化为大数
BigNum::BigNum(const char *s) {
    int t, k, index, L, i;
    memset(a, 0, sizeof(a));
    L = strlen(s);
    len = L / DLEN;
    if(L % DLEN)len++;
    index = 0;
    for(i = L-1; i >= 0; i -= DLEN) {
        t = 0;
        k = i-DLEN + 1;
        if(k < 0)k = 0;
        for(int j = k; j <= i; j++)
            t = t * 10 + s[j]-'0';
        a[index++] = t;
    }
}
//拷贝构造函数
```

```
BigNum::BigNum(const BigNum &T): len(T.len) {
    int i;
    memset(a, 0, sizeof(a));
    for(i = 0; i < len; i++)
        a[i] = T.a[i];
}
//重载赋值运算符，大数之间赋值运算
BigNum &BigNum::operator=(const BigNum &n) {
    int i;
    len = n.len;
    memset(a, 0, sizeof(a));
    for(i = 0; i < len; i++)
        a[i] = n.a[i];
    return *this;
}
istream &operator>>(istream &in, BigNum &b) {
    char ch[MAXSIZE * 4];
    int i = -1;
    in >> ch;
    int L = strlen(ch);
    int count = 0, sum = 0;
    for(i = L-1; i >= 0;) {
        sum = 0;
        int t = 1;
        for(int j = 0; j < 4 && i >= 0; j++, i--, t *= 10) {
            sum += (ch[i]-'0') * t;
        }
        b.a[count] = sum;
        count++;
    }
    b.len = count++;
    return in;
}
//重载输出运算符
ostream &operator<<(ostream &out, BigNum &b) {
    int i;
    cout << b.a[b.len-1];
    for(i = b.len-2; i >= 0; i--) {
        printf("%04d", b.a[i]);
    }
    return out;
}
//两个大数之间的相加运算
BigNum BigNum::operator+(const BigNum &T)const {
    BigNum t(*this);
    int i, big;
```

```
    big = T.len > len ? T.len : len;
    for(i = 0; i < big; i++) {
        t.a[i] += T.a[i];
        if(t.a[i] > MAXN) {
            t.a[i + 1]++;
            t.a[i] -= MAXN + 1;
        }
    }
    if(t.a[big] != 0)
        t.len = big + 1;
    else t.len = big;
    return t;
}
//两个大数之间的相减运算
BigNum BigNum::operator-(const BigNum &T)const {
    int i, j, big;
    bool flag;
    BigNum t1, t2;
    if(*this > T) {
        t1 = *this;
        t2 = T;
        flag = 0;
    } else {
        t1 = T;
        t2 = *this;
        flag = 1;
    }
    big = t1.len;
    for(i = 0; i < big; i++) {
        if(t1.a[i] < t2.a[i]) {
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1-t2.a[i];
        } else t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[t1.len-1] == 0 && t1.len > 1) {
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1] = 0-t1.a[big-1];
```

```
    return t1;
}
//两个大数之间的相乘
BigNum BigNum::operator*(const BigNum &T)const {
    BigNum ret;
    int i, j, up;
    int temp, temp1;
    for(i = 0; i < len; i++) {
        up = 0;
        for(j = 0; j < T.len; j++) {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN) {
                temp1 = temp-temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            } else {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len-1] == 0 && ret.len > 1)ret.len--;
    return ret;
}
//大数对一个整数进行相除运算
BigNum BigNum::operator/(const int &b)const {
    BigNum ret;
    int i, down = 0;
    for(i = len-1; i >= 0; i--) {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1)-ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len-1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
//大数对一个 int 类型的变量进行取模
int BigNum::operator%(const int &b)const {
    int i, d = 0;
    for(i = len-1; i >= 0; i--)
        d = ((d * (MAXN + 1)) % b + a[i]) % b;
    return d;
```

```
}
//大数的 n 次方运算
BigNum BigNum::operator^(const int &n)const {
    BigNum t, ret(1);
    int i;
    if(n < 0)exit(-1);
    if(n == 0)return 1;
    if(n == 1)return *this;
    int m = n;
    while(m > 1) {
        t = *this;
        for(i = 1; (i << 1) <= m; i <<= 1)
            t = t * t;
        m -= i;
        ret = ret * t;
        if(m == 1)ret = ret * (*this);
    }
    return ret;
}
//大数和另一个大数的大小比较
bool BigNum::operator>(const BigNum &T)const {
    int ln;
    if(len > T.len)return true;
    else if(len == T.len) {
        ln = len-1;
        while(a[ln] == T.a[ln] && ln >= 0)
            ln--;
        if(ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    } else
        return false;
}
//大数和一个 int 类型的变量的大小比较
bool BigNum::operator>(const int &t)const {
    BigNum b(t);
    return *this > b;
}
//输出大数
void BigNum::print() {
    int i;
    printf("%d", a[len-1]);
    for(i = len-2; i >= 0; i--)
        printf("%04d", a[i]);
    printf("\n");
```

```
}
BigNum f[110];//卡特兰数
int main() {
    f[0] = 1;
    for(int i = 1; i <= 100; i++)
        f[i] = f[i-1] * (4 * i-2) / (i + 1); //卡特兰数递推式
    int n;
    while(scanf("%d", &n) == 1) {
        if(n == -1)break;
        f[n].print();
    }
    return 0;
}
```

9.9 子集系列问题

9.9.1 枚举 S 的子集

```
for(int i=s;i;i=(i-1)&s){
  //具体操作
}
```

9.9.2 枚举所有集合的子集

```
for (int S=1; S<(1<<n); ++S){
    for (int S0=S; S0; S0=(S0-1)&S)
        //do something.
}
```

9.9.3 格雷码枚举 2 全集

```
// 每次变换状态, 修改一个位置即可。
const static int sz = 20;
int P[(1 << sz) | 1] = {0};
for (int i = 0; i <= n; i++) P[1 << i] = i; // P 数组表示一个 (2^i -> i) 的映射
int S = 0, sum = 0; // 以元素求和为例
jud();
for (int i = 1; i < 1 << n; i++) {
    int state = (i ^ i << 1) >> 1; // 格雷码第 i 个是 (i^(i*2)/2)
    int pos = P[state ^ S];
    if (state & 1 << pos)sum += a[pos];
    else sum -= a[pos];
    jud();
    S = state;
}
```

9.9.4 枚举集合 S 中大小为 r 的子集

```
for(int s=(1<<r)-1;s<1<<n;){
    // 具体操作
    int x=s&-s;
    int y=s+x;
    s=((s&~y)/x>>1)|y;
}
```

9.10 strtok 和 sscanf 结合输入

```
gets(buf);
int v;
char *p = strtok(buf, " ");
while(p) {
    sscanf(p, "%d", &v);
    p = strtok(NULL, " ");
}
```

9.11 解决爆栈问题

- C++

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

- G++

```
int __size__ = 256<<20;
char *__p__ = (char *)malloc(__size__)+__size__;
__asm__("movl %0,%%esp\n"::"r"(__p__));
```

9.12 STL 常用操作

9.12.1 优先队列 priority_queue

- empty() 如果队列为空返回真
- pop() 删除堆顶元素
- push() / emplace() 加入一个元素
- size() 返回堆中元素个数
- top() 返回堆顶元素

```
priority_queue<int> q_1; // 大的先出队
priority_queue<int,vector<int>,greater<int>> q_2; //小的先出队
```

- 自定义 struct 扔进堆里，需要这样重载：

```
struct node{int x,y;};
bool operator<(node a,node b){return a.x<b.x;}
priority_queue<node> q;//这是大根堆
// 如果用小根堆，那么应当重载大于号，因为用到了 greater<node>
// 最好不要特意把小于号写成大于号的逻辑，而是该写小根堆就写小根堆
```

或者是

```
struct node {
    int x, y;
};
struct cmp {
    bool operator()(node a, node b) {
        return a.x > b.x;
    }
};
priority_queue<node, vector<node>, cmp> q;
// 对比小根堆，这里 cmp 就是重载的大于号
```

9.12.2 set 和 multiset

set 和 multiset 用法一样，就是 multiset 允许重复元素。元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<> 排序规则来排序。不能修改容器里面的元素值，只能插入和删除。

- 自定义 int 排序函数：(默认的是从小到大的，下面这个从大到小)

```
struct classcomp {
    bool operator() (const int &lhs, const int &rhs) const {
       return lhs > rhs;
    }
};// 这里有个逗号的，注意
multiset<int, classcomp> fifth; // class as Compare
```

- begin() 返回指向第一个元素的迭代器
- clear() 清除所有元素
- count() 返回某个值元素的个数，常用作判断元素是否存在
- empty() 如果集合为空，返回 true
- end() 返回指向最后一个元素的迭代器
- erase() 删除集合中的元素 (参数是一个元素值，或者迭代器)
- find() 返回一个指向被查找到元素的迭代器
- insert() 在集合中插入元素
- size() 集合中元素的数目

- lower_bound() 返回指向大于（或等于）某值的第一个元素的迭代器
- upper_bound() 返回大于某个值元素的迭代器
- equal_range() 返回集合中与给定值相等的上下限的两个迭代器
- iterator 迭代器的–和 ++ 操作能返回前驱后继，而且似乎是 $O(1)$ 的

9.13 Java 类

9.13.1 基础语法

```
/* 头文件 */
import java.io.*;
import java.util.*;
import java.math.*;
/* 读入 */
Scanner cin = Scanner (System.in);
while(cin.hasNext()) // 等价于!=EOF
n = cin.nextInt(); // 读入一个 int 型的数
n = cin.nextBigInteger();// 读入一个大整数
/* 输出 */
System.out.print(n); // 打印 n
System.out.println(); // 换行
System.out.printf("%d\n",n); // 也可以类似 c++ 里的输出方式
/* 定义变量 */
int i,j,k,a[];
a = new int[100];
BigInteger n,m;
String s;
/* 数据类型 */
boolean 1 true,false false // 布尔型
byte 8 -128-127 0 // 字节型
char 16 // 字符型
short 16 -32768-32767 0 // 短整型
int 32 -2147483648,2147483647 0 // 整型
long 64 -9.22E18,9.22E18 0 // 长整型
float 32 1.4E-45-3.4028E+38 0.0 // 浮点型
double 64 4.9E-324,1.7977E+308 0.0 // 双精度型
```

9.13.2 大整数（BigInteger）

```
/* 基本函数 */
valueOf(parament); // 将参数转换为制定的类型
BigInteger b = BigInteger.valueOf(a);
String s = "12345";
BigInteger c = BigInteger.valueOf(s);
add(); // 大整数相加
```

```
BigInteger a = new BigInteger("23");
BigInteger b = new BigInteger("34");
a.add(b);
subtract(); // 相减
multiply(); // 相乘
divide();  // 相除取整
remainder(); // 取余
pow();
a.pow(b) = a ^ b
gcd();  // 最大公约数
abs(); // 绝对值
negate(); // 取反数
mod();
a.mod(b) = a % b = a.remainder(b);
max(); min();
int comareTo();
boolean equals(); // 是否相等
/* 构造函数 */
BigInteger(String val); // 将指定字符串转换为十进制表示形式；
// 将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger
BigInteger(String val, int radix);
/* 基本变量 */
A = BigInteger.ONE
B = BigInteger.TEN
C = BigInteger.ZERO
/* 基本操作 */
Scanner cin = new Scanner(System.in); // 读入
while(cin.hasNext())  { // 等同于!=EOF
    int n;
    BigInteger m;
    n = cin.nextInt(); // 读入一个 int;
    m = cin.BigInteger(); // 读入一个 BigInteger;
    System.out.print(m.toString());
}
if( a.compareTo(b) == 0 ) System.out.println("a == b"); // 大整数 a==b
else if( a.compareTo(b) > 0 ) System.out.println("a > b"); // 大整数 a>b
else if( a.compareTo(b) < 0 ) System.out.println("a < b"); // 大整数 a<b
/* 大整数绝对值 */
System.out.println(a.abs()); // 大整数 a 的绝对值
/* 大整数的幂 */
int exponent = 10;
System.out.println(a.pow(exponent)); // 大整数 a 的 exponent 次幂
/* 返回大整数十进制的字符串表示 */
System.out.println(a.toString());
/* 返回大整数 p 进制的字符串表示 */
int p = 8;
```

```
System.out.println(a.toString(p));
```

9.13.3 高精度（BigDecimal）

```
/* 基本介绍 */
BigDecimal BigDecimal(String s); //常用，推荐使用
static BigDecimal valueOf(double d); //常用，推荐使用
BigDecimal BigDecimal(double d); //不允许使用
/* 常用函数 */
BigDecimal remainder(BigDecimal divisor); //求余数
BigDecimal negate(); //求相反数
int compareTo(BigDecimal val);
/* compareTo:
 * 将此 BigDecimal 与指定的 BigDecimal 比较
 * 根据此方法，值相等但具有不同标度的两个 BigDecimal 对象
 * (如，2.0 和 2.00) 被认为是相等的;
 * 相对六个 boolean 比较运算符 (<, ==, >, >=, !=, <=) 中
 * 每一个运算符的各个方法，优先提供此方法;
 * 建议使用以下语句执行上述比较：(x.compareTo(y) <op> 0),
 * 其中 <op> 是六个比较运算符之一;
 * 指定者：接口 Comparable<BigDecimal> 中的 compareTo
 * 返回：当此 BigDecimal 在数字上小于、等于或大于 val 时，返回 -1、0 或 1
 */
public class ArithUtil {
    // 除法运算默认精度
    private static final int DEF_DIV_SCALE = 10;
    private ArithUtil() {}

    /**
     * 精确加法
     */
    public static double add(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.add(b2).doubleValue();
    }

    /**
     * 精确减法
     */
    public static double sub(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.subtract(b2).doubleValue();
    }

    /**
```

```java
     * 精确乘法
     */
    public static double mul(double value1, double value2) {
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        return b1.multiply(b2).doubleValue();
    }

    /**
     * 精确除法 使用默认精度
     */
    public static double div(double value1, double value2)
      throws IllegalAccessException {
        return div(value1, value2, DEF_DIV_SCALE);
    }

    /**
     * 精确除法
     * @param scale 精度
     */
    public static double div(double value1, double value2, int scale)
      throws IllegalAccessException {
        if(scale < 0) {
            throw new IllegalAccessException(" 精确度不能小于 0");
        }
        BigDecimal b1 = BigDecimal.valueOf(value1);
        BigDecimal b2 = BigDecimal.valueOf(value2);
        // return b1.divide(b2, scale).doubleValue();
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

    /**
     * 四舍五入
     * @param scale 小数点后保留几位
     */
    public static double round(double v, int scale) throws IllegalAccessException {
        return div(v, 1, scale);
    }

    /**
     * 比较大小
     */
    public static boolean equalTo(BigDecimal b1, BigDecimal b2) {
        if(b1 == null || b2 == null) {
            return false;
        }
```

```
        return 0 == b1.compareTo(b2);
    }
}
```

某道例题，防止基础 java 忘了怎么写：

```
import java.io.*;
import java.util.*;
import java.math.BigDecimal;

public class Main {
    public static void main(String [] args) {
        Scanner cin = new Scanner(System.in);
        int t = cin.nextInt();
        for (int kase = 1; kase <= t; kase++) {
            int n = cin.nextInt();
            BigDecimal [] a = new BigDecimal[105];
            BigDecimal [] b = new BigDecimal[105];
            BigDecimal [] c = new BigDecimal[105];
            BigDecimal res = new BigDecimal("0.0");
            BigDecimal one = new BigDecimal("1.0");
            cin.nextLine();
            for (int i = 1; i <= n; i++) {
                String str = cin.nextLine();
                String [] da = str.split(":");
                double x = Double.parseDouble(da[0]);
                double y = Double.parseDouble(da[1]);
                a[i] = BigDecimal.valueOf(x);
                b[i] = BigDecimal.valueOf(y);
                c[i] = a[i].divide(a[i].add(b[i]), 50, BigDecimal.ROUND_HALF_UP);
            }
            int cnt = 0;
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j < i; j++) {
                    if (c[j].compareTo(c[i]) == 1) {
                        BigDecimal tmp = new BigDecimal("0");
                        tmp = c[j];
                        c[j] = c[i];
                        c[i] = tmp;
                    }
                }
            }
            for (int i = 1; i <= n; i++) {
                if (res.add(c[i]).compareTo(one) == -1) {
                    cnt++;
                    res = res.add(c[i]);
                } else break;
```

```
            }
            System.out.println("Case #" + kase + ": " + cnt);
        }
        cin.close();
    }
}
```