Code Template for ACM-ICPC

forever97 @ zjutacm

2019 年 10 月 2 日

目录

1 字符串

1.1 Next 函数

```
/*
    nxt函数的应用
*/
int nxt[N];
void get_nxt(int n, char* a) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < n; nxt[i++] = j) {
        while (~j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) j++;
    }
}
/*
    求补上最少字母数量使得这是个循环串
*/
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%s", s);
        get_nxt(len = strlen(s), s);
        int L = len - (nxt[len - 1] + 1);
        if (L < len && len % L == 0)
            puts("0");
        else
            printf("%d\n", L - len % L);
    }
    return 0;
}
/*
    求循环节数量
*/
int main() {
    while (~scanf(" %s", &s)) {
        if (s[0] == '.') break;
        int n = strlen(s);
        get_nxt(n, s);
        printf("%d\n", n % (n - nxt[n - 1] - 1) ? 1 : n / (n - nxt[n - 1] - 1));
    }
    return 0;
}
/*
    求同时是前缀和后缀的串长
*/
```

```
int main() {
    while (~scanf(" %s", &s)) {
        int cnt = 0;
        get_nxt(s, nxt);
        for (t = nxt[(n = strlen(s)) - 1]; t != -1; t = nxt[t]) {
            if (s[t] == s[n - 1]) ans[cnt++] = t + 1;
        }
        for (int i = cnt - 1; i >= 0; i--) printf("%d ", ans[i]);
        printf("%d\n", n);
    }
    return 0;
}
/*
    求出每个循环节的数量和终点位置
*/
int main() {
    while (~scanf("%d", &n) && n) {
        printf("Test case #%d\n", ++cas);
        scanf("%s", s);
        get_nxt(strlen(s), s);
        for (int i = 2; i <= n; i++) {
            if (nxt[i - 1] != -1 && (i % (i - nxt[i - 1] - 1) == 0))
                printf("%d %d\n", i, i / (i - nxt[i - 1] - 1));
        }
        puts("");
    }
    return 0;
}
/*
    求第一个串的前缀和第二个串的后缀的最大匹配
*/
int main() {
    while (~scanf("%s", s)) {
        int L1 = strlen(s);
        scanf("%s", s + L1);
        int L = strlen(s), L2 = L - L1;
        get_nxt(L, s);
        for (k = nxt[L - 1]; k >= L1 || k >= L2; k = nxt[k])
            ;
        if (k == -1)
            puts("0");
        else {
            for (int i = 0; i <= k; i++) printf("%c", s[i]);
            printf(" %d\n", k + 1);
        }
    }
```

```
    return 0;
}
/*
    求字符串每个前缀和串匹配成功的次数和
*/
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%s", &len, s);
        get_nxt(len, s);
        ans = 0;
        for (int i = 0; i < len; i++) {
            if (nxt[i] < 0)
                dp[i] = 1;
            else
                dp[i] = (dp[nxt[i]] + 1) % mod;
            ans = (ans + dp[i]) % mod;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.2 KMP

```
/*
    KMP
    计算a串在b串中的出现次数
*/
const int N = 1000005;
int T, nxt[N], ans;
char a[N], b[N];
void kmp(int n, char* a, int m, char* b) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < n; nxt[i++] = j) {
        while (~j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) j++;
    }
    for (j = -1, i = 0; i < m; i++) {
        while (~j && a[j + 1] != b[i]) j = nxt[j];
        if (a[j + 1] == b[i]) j++;
        if (j == n - 1) ans++, j = nxt[j];
    }
}
int main() {
```

```
    scanf("%d", &T);
    while (T--) {
        ans = 0;
        scanf("%s %s", a, b);
        kmp(strlen(a), a, strlen(b), b);
        printf("%d\n", ans);
    }
    return 0;
}
```

1.3 矩阵加速 KMP

```
/*
    kmp+矩阵加速
    求长度为n的不包含长度为m的子串的串个数
*/
#define rep(i, n) for (int i = 0; i < n; i++)
using namespace std;
int t, n, m, mod, a[25][25], b[25][25], tmp[25][25], p[25];
char s[25];
void mul(int a[25][25], int b[25][25], int ans[25][25]) {
    rep(i, m) rep(j, m) {
        tmp[i][j] = 0;
        rep(k, m) tmp[i][j] = (tmp[i][j] + a[i][k] * b[k][j]) % mod;
    }
    rep(i, m) rep(j, m) ans[i][j] = tmp[i][j];
}
int main() {
    scanf("%d%d%d", &n, &m, &mod);
    scanf("%s", s + 1);
    int j = 0;
    for (int i = 2; i <= m; i++) {
        while (j > 0 && s[j + 1] != s[i]) j = p[j];
        if (s[j + 1] == s[i]) j++;
        p[i] = j;
    }
    rep(i, m) rep(j, 10) {
        for (t = i; t && s[t + 1] - '0' != j; t = p[t])
            ;
        if (s[t + 1] - '0' == j) t++;
        if (t != m) b[i][t] = (b[i][t] + 1) % mod;
    }
    rep(i, m) a[i][i] = 1;
    int sum = 0;
    while (n) {
```

```
        if (n & 1) mul(a, b, a);
        mul(b, b, b);
        n >>= 1;
    }
    for (int i = 0; i < m; i++) sum = (sum + a[0][i]) % mod;
    return printf("%d\n", sum), 0;
}
```

1.4 压缩串 KMP

```
/*
    压缩串KMP
    求最长匹配
*/
pair<char, long long> s[200005], t[200005], tmp1, tmp2;
int nxt[200005], n, m, cnt1, cnt2;
long long ans, num;
char c;
void kmp(int n, pair<char, long long>* a, int m, pair<char, long long>* b) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < n; nxt[i++] = j) {
        while (~j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) j++;
    }
    for (j = -1, i = 0; i < m; i++) {
        while (~j && a[j + 1] != b[i]) j = nxt[j];
        if (a[j + 1] == b[i]) j++;
        if (j == n - 1) {
            if (i >= n && b[i + 1].first == tmp2.first &&
                b[i + 1].second >= tmp2.second &&
                b[i - n].first == tmp1.first && b[i - n].second >= tmp1.second)
                ans++;
            j = nxt[j];
        }
    }
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf(" %lld-%c", &num, &c);
        if (s[cnt1].first == c)
            s[cnt1].second += num;
        else {
            s[++cnt1].first = c;
            s[cnt1].second = num;
```

```
        }
    }
    for (int i = 1; i <= m; i++) {
        scanf(" %lld-%c", &num, &c);
        if (t[cnt2].first == c)
            t[cnt2].second += num;
        else {
            t[++cnt2].first = c;
            t[cnt2].second = num;
        }
    }
    if (cnt2 >= 3) {
        tmp1 = t[1];
        tmp2 = t[cnt2];
        kmp(cnt2 - 2, t + 2, cnt1, s + 1);
    } else if (cnt2 == 1) {
        for (int i = 1; i <= cnt1; i++)
            if (t[cnt2].first == s[i].first && s[i].second >= t[cnt2].second) {
                ans += (long long)(s[i].second - t[cnt2].second + 1);
            }
    } else if (cnt2 == 2) {
        for (int i = 1; i < cnt1; i++)
            if (t[1].first == s[i].first && t[2].first == s[i + 1].first &&
                t[1].second <= s[i].second && t[2].second <= s[i + 1].second)
                ans++;
    }
    return printf("%lld\n", ans), 0;
}
```

1.5 Exkmp

```
/*
    扩展kmp
    LCP表示T第i位往后和T原串的最长公共前缀
    extend表示S的第i位往后和T的最长公共前缀
*/
namespace Exkmp {
const int N = 500010;
int len, LCP[N], ex1[N], ex2[N], a[N], s[N];
char S[N], T[N];
void getLCP(char *T) {
    int i, len = strlen(T);
    LCP[0] = len;
    for (i = 0; i < len - 1 && T[i] == T[i + 1]; i++)
        ;
```

```
    LCP[1] = i;
    int a = 1;
    for (int k = 2; k < len; k++) {
        int p = a + LCP[a] - 1, L = LCP[k - a];
        if ((k - 1) + L >= p) {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while (k + j < len && T[k + j] == T[j]) j++;
            LCP[k] = j, a = k;
        } else
            LCP[k] = L;
    }
}
void exkmp(char *S, char *T, int *extend) {
    memset(LCP, 0, sizeof(LCP));
    getLCP(T);
    int Slen = strlen(S), Tlen = strlen(T), a = 0;
    int MinLen = Slen > Tlen ? Tlen : Slen;
    while (a < MinLen && S[a] == T[a]) a++;
    extend[0] = a, a = 0;
    for (int k = 1; k < Slen; k++) {
        int p = a + extend[a] - 1, L = LCP[k - a];
        if ((k - 1) + L >= p) {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while (k + j < Slen && j < Tlen && S[k + j] == T[j]) j++;
            extend[k] = j;
            a = k;
        } else
            extend[k] = L;
    }
}
} // namespace Exkmp
/*
    Example1
    一个字符串的价值定义为，当它是一个回文串的时候，
    价值为每个字符的价值的和，如果不是回文串，价值为0，现在给出每种字符的价值。
    给出一个字符串，要求将其划分为两个子串，要求两个子串的价值和最大。
*/
void revcpy(char *S, char *T, int len) {
    memset(T, 0, sizeof(T));
    for (int i = 0, k = len - 1; i < len; ++i, --k) T[i] = S[k];
}
int Cas;
int main() {
    using namespace Exkmp;
    scanf("%d", &Cas);
    while (Cas--) {
```

```
        for (int i = 0; i < 26; i++) scanf("%d", &a[i]);
        scanf("%s", S);
        len = strlen(S);
        for (int i = 0; S[i]; i++) s[i + 1] = s[i] + a[S[i] - 'a'];
        revcpy(S, T, len);
        exkmp(S, T, ex2);
        exkmp(T, S, ex1);
        int ans = -1e9;
        for (int i = 0; i < len; i++) {
            if (i && ex1[i] + i == len) { // S前缀是回文串
                int j = ex1[i], tmp = s[j];
                if (ex2[j] + j == len) tmp += s[len] - s[j]; // S后缀是回文串
                if (tmp > ans) ans = tmp;
            } else {
                int j = i + 1, tmp = 0;
                if (ex2[j] + j == len) tmp += s[len] - s[j];
                if (tmp > ans) ans = tmp;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example2
    给出一个数字，每次将其最后一位提到最前面来，问产生的所有数字中，
    有多少比原数大，有多少比原数小，有多少和原数相等
*/
int main() {
    int Cas;
    scanf("%d", &Cas);
    for (int cas = 1; cas <= Cas; cas++) {
        int L = 0, E = 0, G = 0;
        scanf("%s", S);
        int len = strlen(S);
        for (int i = 0; i < len; i++) S[len + i] = S[i];
        S[len * 2] = '\0';
        getLCP(S);
        for (int i = 0; i < len; i++) {
            if (LCP[i] >= len)
                E++;
            else if (S[LCP[i]] < S[LCP[i] + i])
                G++;
            else
                L++;
        }
```

```
        printf("Case %d: %d %d %d\n", cas, L / E, E / E, G / E);
    }
    return 0;
}
```

1.6 K 串最长公共子串

```
/*
    求k个串的最长公共子串
    并输出字典序最小的一个
*/
const int N = 4050, M = 210;
using namespace std;
int nxt[M], n;
char dict[N][M];
void get_nxt(char *a, int n) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < n; nxt[i++] = j) {
        while (~j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) j++;
    }
}
int LongestPre(char *s, int len) {
    get_nxt(s, len);
    for (int i = 1; i < n; i++) {
        char *p = dict[i];
        int ans = 0;
        for (int j = -1; *p; p++) {
            while (~j && s[j + 1] != *p) j = nxt[j];
            if (s[j + 1] == *p) {
                j++;
                ans = max(ans, j + 1);
            }
            if (j == len - 1) j = nxt[j];
        }
        len = min(len, ans);
    }
    return len;
}
int main() {
    while (scanf("%d", &n) && n) {
        getchar();
        for (int i = 0; i < n; i++) gets(dict[i]);
        int len = strlen(dict[0]), ans = 0, pos = 0;
        for (int i = 0; i < len; i++) {
```

```
            int tmp = LongestPre(dict[0] + i, len - i);
            if (tmp >= ans) {
                if (tmp > ans)
                    ans = tmp, pos = i;
                else {
                    bool flag = 1;
                    for (int t = 0; t < ans; t++) {
                        if (dict[0][pos + t] > dict[0][i + t])
                            break;
                        else if (dict[0][pos + t] < dict[0][i + t]) {
                            flag = 0;
                            break;
                        }
                    }
                    if (flag) pos = i;
                }
            }
        }
        if (ans) {
            for (int i = 0; i < ans; i++) putchar(dict[0][pos + i]);
            puts("");
        } else
            puts("IDENTITY LOST");
    }
    return 0;
}
```

1.7 Trie

```
/*
    Trie
    Trick: dfs序为符合前缀的所有串或者符合后缀的所有串
*/
struct Trie {
    int tot, dfn, st[S], en[S], son[S][26], ansL, ansR;
    int Tr(char c) { return c - 'a'; }
    void Init() {
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j < 26; j++) son[i][j] = 0;
        dfn = tot = 0;
    }
    int insert(char *s) {
        int x = 0;
        for (int l = strlen(s), i = 0; i < l; i++) {
            if (!son[x][Tr(s[i])]) son[x][Tr(s[i])] = ++tot;
```

```
            x = son[x][Tr(s[i])];
        }
        return x;
    }
    void dfs(int x) {
        st[x] = ++dfn;
        for (int i = 0; i < 26; i++)
            if (son[x][i]) dfs(son[x][i]);
        en[x] = dfn;
    }
    void ask(char *s) {
        int x = 0;
        for (int l = strlen(s), i = 0; i < l; i++) {
            if (!son[x][Tr(s[i])]) {
                ansL = ansR = 0;
                return;
            }
            x = son[x][Tr(s[i])];
        }
        ansL = st[x], ansR = en[x];
    }
};
struct RevTrie {
    int tot, dfn, st[S], en[S], son[S][26], ansL, ansR;
    int Tr(char c) { return c - 'a'; }
    void Init() {
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j < 26; j++) son[i][j] = 0;
        dfn = tot = 0;
    }
    int insert(char *s) {
        int x = 0;
        for (int l = strlen(s), i = l - 1; i >= 0; i--) {
            if (!son[x][Tr(s[i])]) son[x][Tr(s[i])] = ++tot;
            x = son[x][Tr(s[i])];
        }
        return x;
    }
    void dfs(int x) {
        st[x] = ++dfn;
        for (int i = 0; i < 26; i++)
            if (son[x][i]) dfs(son[x][i]);
        en[x] = dfn;
    }
    void ask(char *s) {
        int x = 0;
```

```
        for (int l = strlen(s), i = l - 1; i >= 0; i--) {
            if (!son[x][Tr(s[i])]) {
                ansL = ansR = 0;
                return;
            }
            x = son[x][Tr(s[i])];
        }
        ansL = st[x], ansR = en[x];
    }
};
```

1.8 链表版 Trie

```
/*
    Trie树
    链表版本，节省内存
    选出一些字符串，使得字符串的最长公共前缀*字符串的总个数最大化
*/
const int S = 5000100;
struct Trie {
    int tot, cnt[S], ans, head[S], nxt[S], to[S], m;
    char c[S];
    void insert() {
        char C = getchar();
        for (int x = 0, i = 0; C != '\n'; i++, C = getchar()) {
            int y = -1;
            for (int e = head[x]; e; e = nxt[e])
                if (c[e] == C) {
                    y = to[e];
                    break;
                }
            if (y < 0)
                to[++m] = ++tot, c[m] = C, nxt[m] = head[x], head[x] = m,
                y = tot;
            cnt[x = y]++;
            ans = max(ans, cnt[x] * (i + 1));
        }
    }
} T;
int n;
int main() {
    scanf("%d", &n);
    getchar();
    for (int i = 1; i <= n; i++) T.insert();
    printf("%d\n", T.ans);
```

```
    return 0;
}
```

1.9 AC 自动机

```
/*
    AC自动机
*/
namespace AC_DFA {
const int Csize = 27; // 预留一位'z'+1作为分隔符
int tot, son[N][Csize], sum[N], fail[N], q[N], ans[N], dph[N], match[N];
void Initialize() {
    memset(sum, 0, sizeof(int) * (tot + 1));
    memset(dph, 0, sizeof(int) * (tot + 1));
    memset(ans, 0, sizeof(int) * (tot + 1));
    memset(match, 0, sizeof(int) * (tot + 1));
    memset(fail, 0, sizeof(int) * (tot + 1));
    for (int i = 0; i <= tot; i++)
        for (int j = 0; j < Csize; j++) son[i][j] = 0;
    tot = 0;
    fail[0] = -1;
}
inline int Tr(char ch) { return ch - 'a'; }
int Insert(char *s) {
    int x = 0;
    for (int l = strlen(s), i = 0, w; i < l; i++) {
        if (!son[x][w = Tr(s[i])]) {
            son[x][w] = ++tot;
            dph[tot] = i + 1;
        }
        x = son[x][w];
    }
    sum[x]++;
    return x;
}
void MakeFail() {
    int h = 1, t = 0, i, j, x = 0;
    // 注意单个字符的处理
    for (i = 0; i < Csize; i++)
        if (son[0][i]) {
            q[++t] = son[0][i];
            match[son[0][i]] =
                sum[son[0][i]] ? son[0][i] : match[fail[son[0][i]]];
        }
    while (h <= t)
```

```
        for (x = q[h++], i = 0; i < Csize; i++)
            if (son[x][i]) {
                fail[son[x][i]] = son[fail[x]][i], q[++t] = son[x][i];
                match[son[x][i]] =
                    sum[son[x][i]] ? son[x][i] : match[fail[son[x][i]]];
            } else
                son[x][i] = son[fail[x]][i];
}
/*
    查询有多少串在模式串中出现
    破坏性查询
    sum置0防止被重复统计
*/
int Query(char *s) {
    int res = 0;
    for (int l = strlen(s), x = 0, i = 0; i < l; i++) {
        x = son[x][Tr(s[i])];
        int t = x;
        while (t) {
            res += sum[t];
            sum[t] = 0;
            t = fail[t];
        }
    }
    return res;
}
/*
    在AC_DFA上统计每个串在多少个模式串中出现
    注意最后答案需要累加到fail链的前继
*/
void Search(char *s, int len) {
    for (int l = strlen(s), i = 0, x = 0, w; i < l; i++) {
        x = son[x][Tr(s[i])];
        while (dph[x] > len) x = fail[x];
        ans[match[x]]++;
    }
}
int d[N], st[N];
void Solve() {
    int k = 0;
    memset(d, 0, sizeof(int) * (tot + 1));
    for (int i = 1; i <= tot; i++) d[fail[i]]++;
    for (int i = 1; i <= tot; i++)
        if (!d[i]) st[k++] = i;
    for (int i = 0; i < k; i++) {
        int j = fail[st[i]];
```

```
        ans[j] += ans[st[i]];
        if (!--d[j]) st[k++] = j;
    }
}
for (int i = 0; i < q; i++) pos[i] = Insert(t[i]); // 插入询问串
MakeFail();
for (int i = 0; i < n; i++) Search(s[i], len[i]); // 处理模式串
Solve();
for (int i = 0; i < q; i++) printf("%d\n", ans[pos[i]]);
/*
    给出一些串，询问第x个串和第y个串的公共子串，
    同时要求该公共子串为某个串的前缀。求最长符合要求的答案
    我们对所有串构建AC自动机，将两个询问串之一在AC自动机上mark所有的匹配位置
    另一个串在mark的地方寻找最长匹配即可
*/
int Ans = 0;
void Cal(char *s) {
    memset(vis, 0, sizeof(vis));
    for (int l = strlen(s), i = 0, x = 0, w; i < l; i++) {
        while (!son[x][Tr(s[i])]) x = fail[x];
        x = son[x][Tr(s[i])];
        for (int j = x; j; j = fail[j]) vis[j] = 1;
    }
}
void Find(char *s) {
    for (int l = strlen(s), i = 0, x = 0, w; i < l; i++) {
        while (!son[x][Tr(s[i])]) x = fail[x];
        x = son[x][Tr(s[i])];
        for (int j = x; j; j = fail[j])
            if (vis[j]) Ans = max(Ans, dph[j]);
    }
}
void Solve() {
    int x, y;
    Ans = 0;
    scanf("%d%d", &x, &y);
    Cal(s[x]);
    Find(s[y]);
    printf("%d\n", Ans);
}
} // namespace AC_DFA
```

1.10 AC 自动机 Fail 树应用

```
/*
```

```
    题目大意：查询最长的是x串和y串的公共后缀，且至少是一个给定串集中串的前缀
    求给定串集中有多少串满足前缀是他们的公共后缀
    题解：对所有串建立AC自动机，那么若前缀i是前缀j的后缀，说明i是Fail树上j的祖先。
    所以对于询问(x,y)，答案就是两点在Fail树上的LCA在原Trie中子树内的字符串总数。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 500010;
namespace AC_DFA {
const int Csize = 27;
int tot, son[N][Csize], val[N], fail[N], q[N];
void Initialize() {
    memset(fail, 0, sizeof(int) * (tot + 1));
    memset(val, 0, sizeof(int) * (tot + 1));
    for (int i = 0; i <= tot; i++)
        for (int j = 0; j < Csize; j++) son[i][j] = 0;
    tot = 0;
    fail[0] = -1;
}
inline int Tr(char ch) { return ch - 'a'; }
int Insert(char *s) {
    int x = 0;
    for (int l = strlen(s), i = 0, w; i < l; i++) {
        if (!son[x][w = Tr(s[i])]) son[x][w] = ++tot;
        x = son[x][w];
        val[x]++;
    }
    return x;
}
void MakeFail() {
    int h = 1, t = 0, i, x = 0;
    for (i = 0; i < Csize; i++)
        if (son[0][i]) q[++t] = son[0][i];
    while (h <= t)
        for (x = q[h++], i = 0; i < Csize; i++)
            if (son[x][i])
                fail[son[x][i]] = son[fail[x]][i], q[++t] = son[x][i];
            else
                son[x][i] = son[fail[x]][i];
}
} // namespace AC_DFA
namespace Tree_Chain_Subdivision {
int ed, root, d[N], num[N], v[N << 1], vis[N], f[N], g[N << 1];
int nxt[N << 1], size[N], son[N], st[N], en[N], dfn, top[N];
```

```
void add_edge(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    size[x] = 1;
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != f[x]) {
            f[v[i]] = x, d[v[i]] = d[x] + 1;
            dfs(v[i]), size[x] += size[v[i]];
            if (size[v[i]] > size[son[x]]) son[x] = v[i];
        }
}
void dfs2(int x, int y) {
    if (x == -1) return;
    st[x] = ++dfn;
    top[x] = y;
    if (son[x]) dfs2(son[x], y);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != son[x] && v[i] != f[x]) dfs2(v[i], v[i]);
    en[x] = dfn;
}
int lca(int x, int y) {
    for (; top[x] != top[y]; x = f[top[x]])
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
    return d[x] < d[y] ? x : y;
}
void Initialize() {
    memset(g, dfn = ed = 0, sizeof(g));
    memset(v, 0, sizeof(v));
    memset(nxt, 0, sizeof(nxt));
    memset(son, -1, sizeof(son));
}
} // namespace Tree_Chain_Subdivision
int n, m, pos[N];
char s[N];
int ask(int x, int y) {
    int lca = Tree_Chain_Subdivision::lca(pos[x], pos[y]);
    return AC_DFA::val[lca];
}
int main() {
```

```
    while (~scanf("%d", &n)) {
        using namespace AC_DFA;
        Initialize();
        for (int i = 1; i <= n; i++) {
            scanf("%s", s);
            pos[i] = Insert(s);
        }
        MakeFail();
        Tree_Chain_Subdivision::Initialize();
        for (int i = 1; i <= tot; i++)
            Tree_Chain_Subdivision::add_edge(fail[i], i);
        Tree_Chain_Subdivision::dfs(0);
        Tree_Chain_Subdivision::dfs2(0, 0);
        scanf("%d", &m);
        while (m--) {
            int x, y;
            scanf("%d%d", &x, &y);
            int ans = ask(x, y);
            if (!ans)
                puts("N");
            else
                printf("%d\n", ans);
        }
    }
    return 0;
}
/*
    题目大意：给出一个打印的过程，'a'-'z'表示输入字母，P表示打印该字符串
    B表示删去一个字符。问第x个打印的字符串在第y个打印的字符串中出现的次数
    题解：我们根据打印的过程建立trie树，
    当x是y的子串当且仅当y到根的链上有fail指针指向x的结尾，
    而x在y中的出现次数则取决于有几个这样的指针，
    我们根据fail指针建立fail树，按照fail树的dfs序进行统计，
    在每个y处记录其要查询的x，在y点用树状数组对x点求dfs序区间和即可。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const int N = 100010;
int dfn, l[N], r[N], ans[N];
vector<int> v[N], Q[N], ID[N];
namespace BIT {
int c[N << 1]; // dfs
void Initialize() { memset(c, 0, sizeof(c)); }
```

```
void add(int x, int v) {
    while (x <= dfn) c[x] += v, x += x & -x;
}
int query(int x) {
    int res = 0;
    while (x) res += c[x], x -= x & -x;
    return res;
}
} // namespace BIT
namespace AC_DFA {
const int Csize = 26;
int id, tot, son[N][Csize], sum[N], f[N], fail[N], q[N], pos[N], match[N];
void Initialize() {
    memset(sum, 0, sizeof(int) * (tot + 1));
    memset(fail, 0, sizeof(int) * (tot + 1));
    for (int i = 0; i <= tot; i++)
        for (int j = 0; j < Csize; j++) son[i][j] = 0;
    tot = 0;
    id = 0;
    fail[0] = -1;
}
inline int Tr(char ch) { return ch - 'a'; }
void Build(char *s) {
    int x = 0;
    for (int l = strlen(s), i = 0, w; i < l; i++) {
        if (s[i] == 'P')
            pos[++id] = x;
        else if (s[i] == 'B')
            x = f[x];
        else {
            if (!son[x][w = Tr(s[i])]) {
                son[x][w] = ++tot;
                f[tot] = x;
            }
            x = son[x][w];
        }
    }
}
void MakeFail() {
    int h = 1, t = 0, i, j, x = 0;
    for (i = 0; i < Csize; i++)
        if (son[0][i]) q[++t] = son[0][i];
    while (h <= t)
        for (x = q[h++], i = 0; i < Csize; i++)
            if (son[x][i]) {
                fail[son[x][i]] = son[fail[x]][i], q[++t] = son[x][i];
```

```
            } else
                son[x][i] = son[fail[x]][i];
}
void Solve(char *s) {
    using namespace BIT;
    BIT::Initialize();
    int x = 0, id = 0;
    add(l[0], 1);
    for (int L = strlen(s), i = 0; i < L; i++) {
        if (s[i] == 'P') {
            id++;
            for (int k = 0; k < Q[id].size(); k++) {
                int u = pos[Q[id][k]];
                ans[ID[id][k]] = query(r[u]) - query(l[u] - 1);
            }
        } else if (s[i] == 'B')
            add(l[x], -1), x = f[x];
        else
            x = son[x][Tr(s[i])], add(l[x], 1);
    }
}
} // namespace AC_DFA
void Dfs(int x) {
    l[x] = ++dfn;
    for (int i = 0; i < v[x].size(); i++) Dfs(v[x][i]);
    r[x] = ++dfn;
}
char s[N];
int main() {
    using namespace AC_DFA;
    Initialize();
    scanf("%s", s);
    Build(s);
    MakeFail();
    for (int i = 1; i <= tot; i++) v[fail[i]].push_back(i);
    int m, x, y;
    scanf("%d", &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &x, &y);
        Q[y].push_back(x);
        ID[y].push_back(i);
    }
    Dfs(0);
    Solve(s);
    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    return 0;
```

```
}
```

1.11 AC 自动机 + 矩阵

```
/*
    求长度为n的不包含任何给出字符串的串数量
*/
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 110;
typedef long long LL;
LL P = 100000LL;
struct mat {
    int n;
    LL num[110][110];
    void init0(int t) {
        n = t;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) num[i][j] = 0;
    }
    void init1(int t) {
        n = t;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i != j)
                    num[i][j] = 0;
                else
                    num[i][j] = 1;
    }
    mat operator=(const struct mat p) {
        n = p.n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) num[i][j] = p.num[i][j];
    }
    mat operator*(const struct mat p) const {
        struct mat ans;
        ans.init0(n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                    ans.num[i][j] =
                        (ans.num[i][j] + num[i][k] * p.num[k][j]) % P;
        return ans;
    }
```

```
    mat operator^(int t) const {
        struct mat ans, now;
        ans.init1(n);
        now.n = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) now.num[i][j] = num[i][j];
        while (t > 0) {
            if (t & 1) ans = ans * now;
            now = now * now;
            t >>= 1;
        }
        return ans;
    }
} mat;
namespace AC_DFA {
const int Csize = 4;
int tot, son[N][Csize], sum[N], fail[N], q[N], ans[N], match[N];
void Initialize() {
    memset(sum, 0, sizeof(int) * (tot + 1));
    memset(ans, 0, sizeof(int) * (tot + 1));
    memset(match, 0, sizeof(int) * (tot + 1));
    memset(fail, 0, sizeof(int) * (tot + 1));
    for (int i = 0; i <= tot; i++)
        for (int j = 0; j < Csize; j++) son[i][j] = 0;
    tot = 0;
    fail[0] = -1;
}
inline int Tr(char ch) {
    if (ch == 'A') return 0;
    if (ch == 'T') return 1;
    if (ch == 'C') return 2;
    if (ch == 'G') return 3;
}
int Insert(char *s) {
    int x = 0;
    for (int l = strlen(s), i = 0, w; i < l; i++) {
        if (!son[x][w = Tr(s[i])]) {
            son[x][w] = ++tot;
        }
        x = son[x][w];
    }
    sum[x]++;
    return x;
}
void MakeFail() {
    int h = 1, t = 0, i, j, x = 0;
```

```
    for (i = 0; i < Csize; i++)
        if (son[0][i]) {
            q[++t] = son[0][i];
            match[son[0][i]] =
                sum[son[0][i]] ? son[0][i] : match[fail[son[0][i]]];
        }
    while (h <= t)
        for (x = q[h++], i = 0; i < Csize; i++)
            if (son[x][i]) {
                fail[son[x][i]] = son[fail[x]][i], q[++t] = son[x][i];
                match[son[x][i]] =
                    sum[son[x][i]] ? son[x][i] : match[fail[son[x][i]]];
            } else
                son[x][i] = son[fail[x]][i];
}
} // namespace AC_DFA
using namespace AC_DFA;
char s[20];
void BuildMat() {
    mat.init0(tot + 1);
    for (int i = 0; i <= tot; i++) {
        if (match[i]) continue;
        for (int j = 0; j < Csize; j++) {
            if (!match[son[i][j]]) mat.num[i][son[i][j]]++;
        }
    }
}
int n, m;
int main() {
    while (~scanf("%d%d", &m, &n)) {
        Initialize();
        while (m--) {
            scanf("%s", s);
            Insert(s);
        }
        MakeFail();
        BuildMat();
        mat = mat ^ n;
        int ans = 0;
        for (int i = 0; i < mat.n; i++) ans = (ans + mat.num[0][i]) % P;
        printf("%d\n", ans);
    }
    return 0;
}
```

1.12 后缀数组 SA

```
/*
    SA求后缀数组
    复杂度O(nlogn)
*/
#define inf 1000000000
using namespace std;
typedef long long ll;
const int N = 100010;
int n, m, Log[N];
char S[N];
struct SA {
    int p, q, k;
    int sa[2][N], rk[2][N], mn[17][N];
    int a[N], h[N], v[N];
    ll s[N];
    SA() { p = 0, q = 1; }
    void mul(int *sa, int *rk, int *SA, int *RK) {
        for (int i = 1; i <= n; i++) v[rk[sa[i]]] = i;
        for (int i = n; i; i--)
            if (sa[i] > k) SA[v[rk[sa[i] - k]]--] = sa[i] - k;
        for (int i = n - k + 1; i <= n; i++) SA[v[rk[i]]--] = i;
        for (int i = 1; i <= n; i++)
            RK[SA[i]] = RK[SA[i - 1]] + (rk[SA[i - 1]] != rk[SA[i]] ||
                                         rk[SA[i - 1] + k] != rk[SA[i] + k]);
    }
    void getsa() {
        for (int i = 1; i <= n; i++) v[a[i]]++;
        for (int i = 1; i <= 30; i++) v[i] += v[i - 1];
        for (int i = 1; i <= n; i++) sa[p][v[a[i]]--] = i;
        for (int i = 1; i <= n; i++)
            rk[p][sa[p][i]] =
                rk[p][sa[p][i - 1]] + (a[sa[p][i]] != a[sa[p][i - 1]]);
        for (k = 1; k < n; k <<= 1, swap(p, q)) mul(sa[p], rk[p], sa[q], rk[q]);
        for (int k = 0, i = 1; i <= n; i++) {
            int j = sa[p][rk[p][i] - 1];
            while (a[i + k] == a[j + k]) k++;
            h[rk[p][i]] = k;
            if (k) k--;
        }
    }
    void pre() {
        for (int i = 1; i <= n; i++) a[i] = S[i] - 'a' + 1;
        getsa();
        for (int i = 1; i <= n; i++) mn[0][i] = h[i];
```

```
        for (int i = 1; i <= Log[n]; i++)
            for (int j = 1; j + (1 << i) - 1 <= n; j++) {
                mn[i][j] = min(mn[i - 1][j], mn[i - 1][j + (1 << (i - 1))]);
            }
        for (int i = 1; i <= n; i++) s[i] = s[i - 1] + n - sa[p][i] + 1 - h[i];
    }
    int query(int a, int b) {
        a = rk[p][a], b = rk[p][b];
        if (a > b) swap(a, b);
        a++;
        int t = Log[b - a + 1];
        return min(mn[t][a], mn[t][b - (1 << t) + 1]);
    }
    void print() {
        for (int i = 2; i <= n; i++) printf("%d ", h[i]);
    }
} A, B;
/*
   Example
      将一个字符串的子串去重后排序
      给出一个字符串的两个子串的在其子串序列中的排名
      求这两个子串公共前后缀的长度平方和
*/
int main() {
    Log[0] = -1;
    for (int i = 1; i <= 100000; i++) Log[i] = Log[i >> 1] + 1;
    scanf("%d%d", &n, &m);
    scanf("%s", S + 1);
    A.pre();
    reverse(S + 1, S + n + 1);
    B.pre();
    for (int i = 1; i <= m; i++) {
        ll l, r, ans = 0, id, a1, a2, b1, b2;
        scanf("%lld%lld", &l, &r);
        if (l > A.s[n] || r > A.s[n]) {
            puts("-1");
            continue;
        }
        id = lower_bound(A.s + 1, A.s + n + 1, l) - A.s;
        a1 = A.sa[A.p][id];
        b1 = A.sa[A.p][id] + A.h[id] - 1 + l - A.s[id - 1];
        id = lower_bound(A.s + 1, A.s + n + 1, r) - A.s;
        a2 = A.sa[A.p][id];
        b2 = A.sa[A.p][id] + A.h[id] - 1 + r - A.s[id - 1];
        ll t = (a1 == a2) ? inf : A.query(a1, a2);
        t = min(t, min(b1 - a1 + 1, b2 - a2 + 1));
```

```
        ans += t * t;
        t = (n - b1 + 1 == n - b2 + 1) ? inf : B.query(n - b1 + 1, n - b2 + 1);
        t = min(t, min(b1 - a1 + 1, b2 - a2 + 1));
        ans += t * t;
        printf("%lld\n", ans);
    }
    return 0;
}
```

1.13 后缀数组 DC3

```
/*
    DC3
    复杂度O(n)
    rk: 该位置[0~len-1]的后缀排名
    sa: 对应排名在原串[0~len-1]中的位置
    ht: 排名相邻的后缀最长公共前缀[2~len]
*/
#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : ty))
#define G(x) ((x) < ty ? (x)*3 + 1 : ((x)-ty) * 3 + 2)
using namespace std;
const int N = 20100;
template <typename T = int> // 应用时直接将T赋值为需要的类型速度比较快
struct SA {
    int str[N * 3], sa[N * 3], rk[N], ht[N], sz;
    int wa[N], wb[N], wv[N], ws[N];
    int &operator[](int k) { return sa[k]; }
    int size() const { return sz; }
    bool eq(const int *r, int a, int b) const {
        return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2];
    }
    bool cmp(const int *r, int a, int b, int d) const {
        if (d == 1)
            return (r[a] < r[b]) || (r[a] == r[b] && wv[a + 1] < wv[b + 1]);
        return (r[a] < r[b]) || (r[a] == r[b] && cmp(r, a + 1, b + 1, 1));
    }
    void rsort(const int *r, const int *a, int *b, int n, int m) {
        int i;
        fill(ws, ws + m, 0);
        for (i = 0; i < n; i++) ++ws[wv[i] = r[a[i]]];
        for (i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (i = n - 1; ~i; i--) b[--ws[wv[i]]] = a[i];
    }
    void dc3(int *r, int *sa, int n, int m) {
        int i, j, k, *rn = r + n, *san = sa + n, tx = 0, ty = (n + 1) / 3,
```

```
                tz = 0;
        r[n] = r[n + 1] = 0;
        for (i = 0; i < n; i++) {
            if (i % 3) wa[tz++] = i;
        }
        rsort(r + 2, wa, wb, tz, m);
        rsort(r + 1, wb, wa, tz, m);
        rsort(r, wa, wb, tz, m);
        for (rn[F(wb[0])] = 0, k = i = 1; i < tz; i++) {
            rn[F(wb[i])] = eq(r, wb[i - 1], wb[i]) ? k - 1 : k++;
        }
        if (k < tz)
            dc3(rn, san, tz, k);
        else {
            for (i = 0; i < tz; i++) san[rn[i]] = i;
        }
        for (i = 0; i < tz; i++) {
            if (san[i] < ty) wb[tx++] = san[i] * 3;
        }
        if (n % 3 == 1) wb[tx++] = n - 1;
        rsort(r, wb, wa, tx, m);
        for (i = 0; i < tz; i++) wv[wb[i] = G(san[i])] = i;
        for (i = j = k = 0; i < tx && j < tz; k++) {
            sa[k] = cmp(r, wa[i], wb[j], wb[j] % 3) ? wa[i++] : wb[j++];
        }
        for (; i < tx; i++) sa[k++] = wa[i];
        for (; j < tz; j++) sa[k++] = wb[j];
    }
    void build(const T *s, int n, int m = 128) {
        int i;
        for (i = 0; i < n; i++) str[i] = (int)s[i];
        str[n] = 0;
        sz = n + 1;
        dc3(str, sa, sz, m);
    }
    void CalHeight() {
        int i, j, k = 0;
        for (i = 0; i < sz; i++) rk[sa[i]] = i;
        for (i = 0; i < sz; ht[rk[i++]] = k) {
            for (k ? --k : 0, j = sa[rk[i] - 1]; str[i + k] == str[j + k]; k++)
                ;
        }
    }
};
/*
    Example
```

```
    求最长可允许重叠的出现次数不小于k的子串
*/
SA<int> Sa;
int n, s[N];
int k;
bool check(int x) {
    int num = 0;
    for (int i = 1; i <= n; i++) {
        if (Sa.ht[i] >= x)
            num++;
        else
            num = 1;
        if (num >= k) return 1;
    }
    return 0;
}
int main() {
    while (~scanf("%d%d", &n, &k)) {
        for (int i = 0; i < n; i++) scanf("%d", &s[i]);
        s[n] = 0;
        Sa.build(s, n, 128);
        Sa.CalHeight();
        int l = 0, r = n, ans = 0;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.14 诱导排序 SA

```
/*
    诱导排序SA
    复杂度O(n)
    rk: 该位置[0~len-1]的后缀排名
    sa: 对应排名在原串[0~len-1]中的位置
    ht: 排名相邻的后缀最长公共前缀[2~len]
*/
const int N = 100010;
```

```
namespace SA {
int sa[N], rk[N], ht[N], s[N << 1], t[N << 1], p[N], cnt[N], cur[N];
#define pushS(x) sa[cur[s[x]]--] = x
#define pushL(x) sa[cur[s[x]]++] = x
#define inducedSort(v)                          \
    fill_n(sa, n, -1);                          \
    fill_n(cnt, m, 0);                          \
    for (int i = 0; i < n; i++) cnt[s[i]]++;    \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1]; \
    for (int i = 0; i < m; i++) cur[i] = cnt[i] - 1; \
    for (int i = n1 - 1; ~i; i--) pushS(v[i]);  \
    for (int i = 1; i < m; i++) cur[i] = cnt[i - 1]; \
    for (int i = 0; i < n; i++)                 \
        if (sa[i] > 0 && t[sa[i] - 1]) pushL(sa[i] - 1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i] - 1; \
    for (int i = n - 1; ~i; i--)                \
        if (sa[i] > 0 && !t[sa[i] - 1]) pushS(sa[i] - 1)
void sais(int n, int m, int *s, int *t, int *p) {
    int n1 = t[n - 1] = 0, ch = rk[0] = -1, *s1 = s + n;
    for (int i = n - 2; ~i; i--)
        t[i] = s[i] == s[i + 1] ? t[i + 1] : s[i] > s[i + 1];
    for (int i = 1; i < n; i++)
        rk[i] = t[i - 1] && !t[i] ? (p[n1] = i, n1++) : -1;
    inducedSort(p);
    for (int i = 0, x, y; i < n; i++)
        if (~(x = rk[sa[i]])) {
            if (ch < 1 || p[x + 1] - p[x] != p[y + 1] - p[y])
                ch++;
            else
                for (int j = p[x], k = p[y]; j <= p[x + 1]; j++, k++)
                    if ((s[j] << 1 | t[j]) != (s[k] << 1 | t[k])) {
                        ch++;
                        break;
                    }
            s1[y = x] = ch;
        }
    if (ch + 1 < n1)
        sais(n1, ch + 1, s1, t + n, p + n1);
    else
        for (int i = 0; i < n1; i++) sa[s1[i]] = i;
    for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
    inducedSort(s1);
}
template <typename T>
int mapCharToInt(int n, const T *str) {
    int m = *max_element(str, str + n);
```

```
    fill_n(rk, m + 1, 0);
    for (int i = 0; i < n; i++) rk[str[i]] = 1;
    for (int i = 0; i < m; i++) rk[i + 1] += rk[i];
    for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
    return rk[m];
}
// 最后一位添加符一定要保证是字典序最小的字符
template <typename T>
void suffixArray(int n, const T *str) {
    int m = mapCharToInt(++n, str);
    sais(n, m, s, t, p);
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = ht[0] = 0; i < n - 1; i++) {
        int j = sa[rk[i] - 1];
        while (i + h < n && j + h < n && s[i + h] == s[j + h]) h++;
        if (ht[rk[i]] = h) h--;
    }
}
}; // namespace SA
int n;
char s[100010];
// Test
int main() {
    scanf("%s", s);
    n = strlen(s);
    s[n] = 'a' - 1;
    SA::suffixArray(n, s);
    for (int i = 1; i <= n; i++)
        printf(i == n ? "%d\n" : "%d ", SA::sa[i] + 1); // 输出在原串中的位置
    for (int i = 2; i <= n; i++)
        printf(i == n ? "%d\n" : "%d ",
               SA::ht[i]); // 排名相邻的后缀的最长公共前缀
    return 0;
}
/*
    Example1
    求最长可允许重叠的出现次数不小于k的子串
*/
int n, s[N];
int k;
bool check(int x) {
    int num = 0;
    for (int i = 1; i <= n; i++) {
        if (SA::ht[i] >= x)
            num++;
        else
```

```
            num = 1;
        if (num >= k) return 1;
    }
    return 0;
}
int main() {
    while (~scanf("%d%d", &n, &k)) {
        for (int i = 0; i < n; i++) scanf("%d", &s[i]);
        s[n] = 0;
        SA::suffixArray(n, s);
        int l = 0, r = n, ans = 0;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example2
    求最长不允许重叠的出现次数大于1的子串
    示例为差分数组求解(升降调视为相同旋律)
*/
vector<int> v[N];
bool check(int x) {
    int cnt = -1;
    for (int i = 1; i <= n; i++) {
        if (SA::ht[i] < x) v[++cnt].clear();
        v[cnt].push_back(i);
    }
    for (int i = 0; i <= cnt; i++) {
        int L = N, R = -1;
        if (v[i].size() > 1) {
            for (int j = 0; j < v[i].size(); j++) {
                R = max(R, SA::sa[v[i][j]]);
                L = min(L, SA::sa[v[i][j]]);
            }
            if (R - L >= x) return 1;
        }
    }
    return 0;
}
```

```
int main() {
    while (scanf("%d", &n), n) {
        for (int i = 0; i < n; i++) scanf("%d", &s[i]);
        for (int i = 0; i < n - 1; i++) s[i] = s[i + 1] - s[i] + 90;
        s[--n] = 0;
        SA::suffixArray(n, s);
        int l = 0, r = n, ans = 0;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        printf("%d\n", ans >= 4 ? ans + 1 : 0);
    }
    return 0;
}
/*
    Example3
    求至少在k个字符串中出现的最长公共子串
*/
int first = 0, len[N], u, K;
vector<int> S[N];
bool vis[N];
bool check(int L) {
    int cur = -1;
    for (int i = 1; i <= u; i++) {
        if (h[i] < L) S[++cur].clear();
        S[cur].push_back(i);
    }
    for (int i = 0; i <= cur; i++) {
        if (S[i].size() > K) {
            memset(vis, 0, sizeof(vis));
            for (int j = 0; j < S[i].size(); j++) {
                int k = S[i][j];
                int x = upper_bound(a, a + n + 1, sa[S[i][j]]) - a - 1;
                vis[x] = 1;
            }
            int count = 0;
            for (int j = 0; j < n; j++)
                if (vis[j]) count++;
            if (count > K) return 1;
        }
    }
    return 0;
```

```
}
void Print(int L) {
    int cur = -1;
    for (int i = 1; i <= u; i++) {
        if (SA::ht[i] < L) S[++cur].clear();
        S[cur].push_back(i);
    }
    for (int i = 0; i <= cur; i++) {
        if (S[i].size() > K) {
            memset(vis, 0, sizeof(vis));
            for (int j = 0; j < S[i].size(); j++) {
                int k = S[i][j];
                int x = upper_bound(a, a + n + 1, SA::sa[S[i][j]]) - a - 1;
                vis[x] = true;
            }
            int count = 0;
            for (int j = 0; j < n; j++)
                if (vis[j]) count++;
            if (count > K) {
                for (int j = 0; j < L; j++)
                    printf("%c", char(s[SA::sa[S[i][0]] + j]));
                puts("");
            }
        }
    }
}
int main() {
    while (~scanf("%d", &n), n) {
        if (first++) puts("");
        int tmp = 200;
        u = 0;
        for (int i = 0; i < n; i++) {
            scanf("%s", str);
            len[i] = strlen(str);
            for (int j = 0; j < len[i]; j++) s[u++] = (int)str[j];
            s[u++] = tmp++;
        }
        tmp = 0;
        s[u] = 0;
        for (int i = 0; i <= n; i++) {
            a[i] = tmp;
            if (i < n) tmp = tmp + (i == 0 ? len[i] : len[i] + 1);
        }
        SA::suffixArray(u, s);
        int l = 1, r = 1000, ans = 0;
        K = n / 2; // 求至少在一半字符串中出现的最长公共子串
```

```
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        if (ans == 0)
            puts("?");
        else
            Print(ans);
    }
    return 0;
}
/*
    Example4
    求在每个字符串中出现至少两次的最长的子串
*/
int first = 0, len[N], u, K;
vector<int> S[N];
int Min[15], Max[15];
bool check(int L) {
    int cur = -1;
    for (int i = 1; i <= u; i++) {
        if (SA::ht[i] < L) S[++cur].clear();
        S[cur].push_back(i);
    }
    for (int i = 0; i <= cur; i++) {
        if (S[i].size() >= 2 * n) {
            memset(Min, -1, sizeof(Min));
            memset(Max, -1, sizeof(Max));
            for (int j = 0; j < S[i].size(); j++) {
                int k = S[i][j];
                int x = upper_bound(a, a + n + 1, SA::sa[k]) - a - 1;
                Min[x] = Min[x] == -1 ? sa[k] : min(Min[x], SA::sa[k]);
                Max[x] = Max[x] == -1 ? sa[k] : max(Max[x], SA::sa[k]);
            }
            bool flag = 1;
            for (int i = 0; i < n; i++) {
                if (Min[i] == -1 || Max[i] - Min[i] < L) {
                    flag = 0;
                    break;
                }
            }
            if (flag) return 1;
        }
```

```
    }
    return 0;
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        int tmp = 200;
        u = 0;
        for (int i = 0; i < n; i++) {
            scanf("%s", str);
            len[i] = strlen(str);
            for (int j = 0; j < len[i]; j++) s[u++] = (int)str[j];
            s[u++] = tmp++;
        }
        tmp = 0;
        s[u] = 0;
        for (int i = 0; i <= n; i++) {
            a[i] = tmp;
            if (i < n) tmp = tmp + (i == 0 ? len[i] : len[i] + 1);
        }
        SA::suffixArray(u, s);
        int l = 1, r = 10000, ans = 0;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.15 诱导排序 SA+ST 表

```
/*
    SAIS+ST_Table
    给定n个字符串，输出第x串跟第y串的最长公共前缀
    SA的ST判断的时候注意相同位置后缀判断会出错，要特判
*/
#include <algorithm>
#include <cstdio>
```

```
#include <cstring>
using namespace std;
const int N = 4000010;
namespace SA {
int sa[N], rk[N], ht[N], s[N << 1], t[N << 1], p[N], cnt[N], cur[N];
#define pushS(x) sa[cur[s[x]]--] = x
#define pushL(x) sa[cur[s[x]]++] = x
#define inducedSort(v)                          \
    fill_n(sa, n, -1);                          \
    fill_n(cnt, m, 0);                          \
    for (int i = 0; i < n; i++) cnt[s[i]]++;    \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1]; \
    for (int i = 0; i < m; i++) cur[i] = cnt[i] - 1; \
    for (int i = n1 - 1; ~i; i--) pushS(v[i]);  \
    for (int i = 1; i < m; i++) cur[i] = cnt[i - 1]; \
    for (int i = 0; i < n; i++)                 \
        if (sa[i] > 0 && t[sa[i] - 1]) pushL(sa[i] - 1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i] - 1; \
    for (int i = n - 1; ~i; i--)                \
        if (sa[i] > 0 && !t[sa[i] - 1]) pushS(sa[i] - 1)
void sais(int n, int m, int *s, int *t, int *p) {
    int n1 = t[n - 1] = 0, ch = rk[0] = -1, *s1 = s + n;
    for (int i = n - 2; ~i; i--)
        t[i] = s[i] == s[i + 1] ? t[i + 1] : s[i] > s[i + 1];
    for (int i = 1; i < n; i++)
        rk[i] = t[i - 1] && !t[i] ? (p[n1] = i, n1++) : -1;
    inducedSort(p);
    for (int i = 0, x, y; i < n; i++)
        if (~(x = rk[sa[i]])) {
            if (ch < 1 || p[x + 1] - p[x] != p[y + 1] - p[y])
                ch++;
            else
                for (int j = p[x], k = p[y]; j <= p[x + 1]; j++, k++)
                    if ((s[j] << 1 | t[j]) != (s[k] << 1 | t[k])) {
                        ch++;
                        break;
                    }
            s1[y = x] = ch;
        }
    if (ch + 1 < n1)
        sais(n1, ch + 1, s1, t + n, p + n1);
    else
        for (int i = 0; i < n1; i++) sa[s1[i]] = i;
    for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
    inducedSort(s1);
}
```

```
template <typename T>
int mapCharToInt(int n, const T *str) {
    int m = *max_element(str, str + n);
    fill_n(rk, m + 1, 0);
    for (int i = 0; i < n; i++) rk[str[i]] = 1;
    for (int i = 0; i < m; i++) rk[i + 1] += rk[i];
    for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
    return rk[m];
}
// 最后一位添加符一定要保证是字典序最小的字符
template <typename T>
void suffixArray(int n, const T *str) {
    int m = mapCharToInt(++n, str);
    sais(n, m, s, t, p);
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = ht[0] = 0; i < n - 1; i++) {
        int j = sa[rk[i] - 1];
        while (i + h < n && j + h < n && s[i + h] == s[j + h]) h++;
        if (ht[rk[i]] = h) h--;
    }
}
}; // namespace SA
namespace ST {
int f[N][30], lg2[N];
void Init(int n) {
    for (int i = 2; i <= n; i++) lg2[i] = lg2[i / 2] + 1;
    for (int i = 1; i <= n; i++) f[i][0] = SA::ht[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
}
int Min(int l, int r) {
    if (l > r) swap(l, r);
    l++;
    int k = lg2[r - l + 1];
    return min(f[l][k], f[r - (1 << k) + 1][k]);
}
}; // namespace ST
int T, n, len[N], pos[N];
char t[N], s[N];
int main() {
    scanf("%d", &T);
    for (int cas = 1; cas <= T; cas++) {
        printf("Case %d:\n", cas);
        scanf("%d", &n);
        int u = 0;
```

```
        for (int i = 1; i <= n; i++) {
            scanf("%s", &t);
            len[i] = strlen(t);
            pos[i] = u;
            for (int j = 0; j < len[i]; j++) s[u++] = t[j];
        }
        s[u] = 'a' - 1;
        SA::suffixArray(u, s);
        ST::Init(u);
        int q;
        scanf("%d", &q);
        while (q--) {
            int x, y;
            scanf("%d%d", &x, &y);
            int ans = min(len[x], len[y]);
            if (x == y)
                printf("%d\n", len[x]);
            else
                printf("%d\n",
                       min(ans, ST::Min(SA::rk[pos[x]], SA::rk[pos[y]])));
        }
    }
    return 0;
}
```

1.16 后缀自动机

```
/*
    后缀自动机
    时间复杂度O(n)
    cnt: 节点数量
    l[x]: 节点x的匹配长度
    f[x]: 失配指针, f[x]表示的状态为x表示的最小状态的后缀
        l[x]-l[f[x]]即x到f[x]之间非重子串的数量
        比如x:abc f[x]:c, 则l[x]-l[f[x]]就得到了串bc和串abc
    r[x]: right集合, x状态的串数量
    数组开串长两倍大小
*/
char s[N];
struct SAM {
    int p, q, np, nq, cnt, lst, a[N][26], l[N], f[N], tot;
    int Tr(char c) { return c - 'a'; }
    SAM() {
        cnt = 0;
        lst = ++cnt;
```

```
}
// 匹配长度与失配位置的匹配长度之差就是他们之间的子串数量
int val(int c) { return l[c] - l[f[c]]; }
void Initialize() {
    memset(l, 0, sizeof(int) * (cnt + 1));
    memset(f, 0, sizeof(int) * (cnt + 1));
    for (int i = 0; i <= cnt; i++)
        for (int j = 0; j < 26; j++) a[i][j] = 0;
    cnt = 0;
    lst = ++cnt;
}
/*
    每次extend之后tot表示非重子串的数量
    Trick:
        求解串区间非重子串数量的时候可考虑建立长度个数SAM
        用ans[l][r]保存l为起点extend到r位置时候的tot
*/
void extend(int c) {
    p = lst;
    np = lst = ++cnt;
    l[np] = l[p] + 1;
    while (!a[p][c] && p) a[p][c] = np, p = f[p];
    if (!p) {
        f[np] = 1;
        tot += val(np);
    } else {
        q = a[p][c];
        if (l[p] + 1 == l[q]) {
            f[np] = q;
            tot += val(np);
        } else {
            nq = ++cnt;
            l[nq] = l[p] + 1;
            memcpy(a[nq], a[q], sizeof(a[q]));
            tot -= val(p) + val(q);
            f[nq] = f[q];
            f[np] = f[q] = nq;
            tot += val(p) + val(q) + val(np) + val(nq);
            while (a[p][c] == q) a[p][c] = nq, p = f[p];
        }
    }
}
int b[N], x[N], r[N];
void build() {
    scanf("%s", s + 1);
    int len = strlen(s + 1);
```

```
    for (int i = 1; i <= len; i++) extend(Tr(s[i]));
    /*
        以下部分为基数排序求right集合r[]
        r[i]表示i状态的串数量
        x[i]表示节点编号
    */
    memset(r, 0, sizeof(int) * (cnt + 1));
    memset(b, 0, sizeof(int) * (cnt + 1));
    for (int i = 1; i <= cnt; i++) b[l[i]]++;
    for (int i = 1; i <= len; i++) b[i] += b[i - 1];
    for (int i = 1; i <= cnt; i++) x[b[l[i]]--] = i;
    for (int i = p = 1; i <= len; i++) {
        p = a[p][Tr(s[i])];
        r[p]++;
    }
    for (int i = cnt; i; i--) r[f[x[i]]] += r[x[i]];
}
/*
    计算母串中恰好出现k次的子串的数量
*/
void solve() {
    int ans = 0, k;
    scanf("%d", &k);
    build();
    for (int i = 1; i <= cnt; i++)
        if (r[x[i]] == k) ans += val(x[i]);
    printf("%d\n", ans);
}
/*
    求子串在母串中的最长匹配
*/
void LongestMatch() {
    scanf("%s", s + 1);
    int len = strlen(s + 1);
    int p = 1, ans = 0;
    for (int i = 1; i <= len; i++) {
        int c = Tr(s[i]);
        if (a[p][c])
            p = a[p][c], ans++;
        else
            break;
    }
    printf("%d\n", ans);
}
/*
    求子串在母串中圆环匹配成功次数
```

```
    (将该子串拆成两段再首尾交换相接的串和母串匹配)
*/
vector<int> ans;
bool flag[N];
void CircleMatch() {
    long long tot = 0;
    scanf("%s", s + 1);
    int len = strlen(s + 1), p = 1, tmp = 0;
    for (int i = 1; i < len; i++) s[i + len] = s[i];
    for (int i = 1; i <= len * 2 - 1; i++) {
        int c = Tr(s[i]);
        if (a[p][c])
            p = a[p][c], tmp++;
        else {
            while (p && !a[p][c]) p = f[p];
            if (!p)
                p = 1, tmp = 0;
            else
                tmp = l[p] + 1, p = a[p][c];
        }
        while (l[f[p]] >= len) p = f[p], tmp = l[p];
        if (!flag[p] && tmp >= len) ans.push_back(p), flag[p] = 1;
    }
    for (int i = 0; i < ans.size(); i++) flag[ans[i]] = 0;
    for (int i = 0; i < ans.size(); i++) tot += r[ans[i]];
    ans.clear();
    printf("%I64d\n", tot);
}
/*
    F函数: S的所有长度为x的子串中, 出现次数的最大值。
*/
int F[N];
void CalF() {
    int len = strlen(s + 1);
    for (int i = 1; i <= cnt; i++) F[l[i]] = max(F[l[i]], r[i]);
    for (int i = 1; i <= len; i++) printf("%d\n", F[i]);
}
/*
    最小表示法
    [不需要build]
*/
void MinExp() {
    Initialize();
    scanf("%s", s);
    int n = strlen(s), len = n;
    for (int k = 0; k < 2; k++)
```

```
        for (int i = 0; i < n; i++) extend(s[i] - 'a');
    int p = 1;
    while (n--) {
        for (int i = 0; i < 26; i++)
            if (a[p][i]) {
                p = a[p][i];
                break;
            }
    }
    printf("%d\n", l[p] - len + 1);
}
/*
    计算在母串中除给定字符串之外的非重子串数量
    注意多组数据时u[]的初始化
    u表示每个位置的最长匹配
    AddString: 往字符集中加入字符串
    Calu: 计算最终答案
*/
int u[N];
void AddString() {
    scanf("%s", s + 1);
    int p = 1, len = strlen(s + 1), tmp = 0;
    for (int i = 1; i <= len; i++) {
        int c = s[i] - 'a';
        if (a[p][c])
            p = a[p][c], tmp++, u[p] = max(u[p], tmp);
        else {
            while (p && !a[p][c]) p = f[p];
            if (!p)
                p = 1, tmp = 0;
            else
                tmp = l[p] + 1, p = a[p][c], u[p] = max(u[p], tmp);
        }
    }
}
void Calu() {
    long long ans = 0;
    for (int i = cnt; i; i--) {
        if (u[x[i]]) {
            u[f[x[i]]] = max(u[x[i]], u[f[x[i]]]);
            if (u[x[i]] < l[x[i]]) ans += l[x[i]] - u[x[i]];
        } else
            ans += val(x[i]);
    }
    printf("%lld\n", ans);
}
```

```
/*
    逆串SAM构建后缀树
    两点的LCA为两个后缀的最长前缀
    BuildTree+ShowResult可计算出sum(len(T[i])+len(T[j])-2*lcp(T[i],T[j]))
    T表示母串的后缀
*/
vector<int> v[N];
void BuildTree() {
    scanf("%s", s + 1);
    int len = strlen(s + 1);
    for (int i = len; i; i--) extend(Tr(s[i]));
    for (int i = 2; i <= cnt; i++) v[f[i]].push_back(i);
}
long long res;
void Dfs(int x, int fx) {
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i];
        Dfs(y, x);
        size[x] += size[y];
    }
    l[x] -= l[fx];
    res = res - (long long)size[x] * (size[x] - 1) * l[x];
}
void ShowResult() {
    int len = strlen(s + 1);
    res = (long long)(len - 1) * len * (len + 1) / 2;
    for (int i = 0; i < v[1].size(); i++) Dfs(v[1][i], 1);
    printf("%lld\n", res);
}
/*
    多串最长公共子串
    母串build
    mx数组初始化for(int i=1;i<=cnt;i++)mx[i]=l[i];
*/
void doit() {
    int len = strlen(s + 1), tmp = 0, p = 1;
    static int arr[N];
    for (int i = 1; i <= len; i++) {
        int c = s[i] - 'a';
        if (a[p][c])
            p = a[p][c], tmp++;
        else {
            while (p && !a[p][c]) p = f[p];
            if (!p)
                p = 1, tmp = 0;
            else
```

```
                tmp = l[p] + 1, p = a[p][c];
        }
        arr[p] = max(arr[p], tmp);
    }
    for (int i = cnt; i; i--) {
        int t = x[i];
        mx[t] = min(mx[t], arr[t]);
        if (arr[t] && f[t]) arr[f[t]] = l[f[t]];
        arr[t] = 0;
    }
}
void getans() {
    int ans = 0;
    for (int i = 1; i <= cnt; i++) ans = max(ans, mx[i]);
    printf("%d\n", ans);
}
/*
    查询字典序排名第k的子串
*/
void query(int k) {
    int p = 1;
    while (k) {
        for (int i = 0; i < 26; i++)
            if (a[p][i]) {
                if (r[a[p][i]] >= k) {
                    putchar('a' + i);
                    p = a[p][i];
                    --k;
                    break;
                } else
                    k -= r[a[p][i]];
            }
    }
    puts("");
}
/*
    两串最长公共子串
*/
int u[N];
void solve() {
    scanf("%s", s + 1);
    int p = 1, len = strlen(s + 1), now = 1, tmp = 0, ans = len + 1;
    for (int i = 1; i <= len; i++) {
        int c = s[i] - 'a';
        if (a[p][c])
            p = a[p][c], u[p]++;
```

```
            else {
                while (p && !a[p][c]) p = f[p];
                if (!p)
                    p = 1;
                else
                    p = a[p][c], u[p]++;
            }
        }
        for (int i = cnt; i; i--) u[f[x[i]]] += u[x[i]];
        for (int i = 1; i <= cnt; i++)
            if (u[i] == 1 && r[i] == 1) ans = min(ans, l[f[i]] + 1);
        if (ans == len + 1)
            puts("-1");
        else
            printf("%d\n", ans);
    }
}
```

1.17 广义后缀自动机

```
/*
    广义SAM
    题目大意：给出多个主串，和多个子串，问每个子串在多少个主串中出现过
    题解：对Trie建广义SAM，从根dfs保存lst，
    用SAM跑每个主串，状态维护cnt和cur分别为出现次数及上一次出现是哪个串，
    出现次数向父亲传递，所以要沿着Parent向上跑更新，遇到cur=当前串时停止
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <string>
using namespace std;
const int N = 200010;
typedef long long ll;
int n, Q;
string ss[N];
char s[N << 1];
struct node {
    int ch[130], par, val;
    int cnt, cur;
} t[N];
int sz = 1, root = 1, lst = 1;
void extend(int c) {
    int p = lst, np = ++sz;
    t[np].val = t[p].val + 1;
```

```
    for (; p && !t[p].ch[c]; p = t[p].par) t[p].ch[c] = np;
    if (!p)
        t[np].par = root;
    else {
        int q = t[p].ch[c];
        if (t[q].val == t[p].val + 1)
            t[np].par = q;
        else {
            int nq = ++sz;
            t[nq] = t[q];
            t[nq].val = t[p].val + 1;
            t[q].par = t[np].par = nq;
            for (; p && t[p].ch[c] == q; p = t[p].par) t[p].ch[c] = nq;
        }
    }
    lst = np;
}
void solve() {
    int u;
    for (int i = 1; i <= n; i++) {
        u = root;
        string &s = ss[i];
        for (int j = 0; j < s.size(); j++) {
            u = t[u].ch[s[j]];
            int p = u;
            for (; p && t[p].cur != i; p = t[p].par) t[p].cnt++, t[p].cur = i;
        }
    }
    while (Q--) {
        scanf("%s", s);
        int n = strlen(s), u = root, flag = 0;
        for (int i = 0; i < n; i++) {
            int c = s[i];
            if (t[u].ch[c])
                u = t[u].ch[c];
            else {
                flag = 1;
                break;
            }
        }
        if (flag)
            puts("0");
        else
            printf("%d\n", t[u].cnt);
    }
}
```

```
int main() {
    scanf("%d%d", &n, &Q);
    for (int i = 1; i <= n; i++) {
        scanf("%s", s);
        ss[i] = string(s);
        lst = root;
        int len = ss[i].size();
        for (int j = 0; j < len; j++) extend(s[j]);
    }
    solve();
    return 0;
}
```

1.18 Endpos 集维护

```
/*
    Endpos集维护
    Problem:
        我们定义一个字符串A比一个字符串B酷当B在A中作为子串出现至少两次
        现在给定一个字符串S，要求找到最长的子串序列S1,S2,S3……
        要求对于任意i，S[i+1]比S[i]酷
    Solution:
        我们发现，如果我们要找一个字符串b，使得字符串a作为子串在其中出现至少两次
        即a的endpos集合在[pos[b]-len[b]+len[a],pos[b]]中出现至少两次
        最短的b串一定是以a为前缀且以a为后缀的，可以得出结论a为b的border
        那么只要a的endpos集合在[pos[b]-len[b]+len[a],pos[b]-1]中出现即可
        用线段树维护endpos，将endpos按fail链线段树合并得到每个节点的endpos集合
        查询区间[pos[b]-len[b]+len[a],pos[b]-1]是否有值
        在fail链节点上保存最长子序列的最后一位，每次扩展时在这一位的线段树上查询
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 400000 + 10;
int Tr(char c) { return c - 'a'; }
int p, q, np, nq, cnt, lst, a[N][26], l[N], f[N], pos[N];
void extend(int c, int ps) {
    p = lst;
    np = lst = ++cnt;
    l[np] = l[p] + 1;
    pos[np] = ps;
    while (!a[p][c] && p) a[p][c] = np, p = f[p];
    if (!p)
        f[np] = 1;
    else {
        q = a[p][c];
```

```
        if (l[p] + 1 == l[q])
            f[np] = q;
        else {
            nq = ++cnt;
            l[nq] = l[p] + 1;
            pos[nq] = pos[q];
            memcpy(a[nq], a[q], sizeof(a[q]));
            f[nq] = f[q];
            f[np] = f[q] = nq;
            while (a[p][c] == q) a[p][c] = nq, p = f[p];
        }
    }
}
int ls[N * 30], rs[N * 30], rt[N], tot;
void upd(int &x, int l, int r, int p) {
    x = ++tot;
    if (l == r) return;
    int mid = l + r >> 1;
    p <= mid ? upd(ls[x], l, mid, p) : upd(rs[x], mid + 1, r, p);
}
int merge(int x, int y) {
    if (!x || !y) return x + y;
    int z = ++tot;
    ls[z] = merge(ls[x], ls[y]);
    rs[z] = merge(rs[x], rs[y]);
    return z;
}
int qry(int x, int l, int r, int ql, int qr) {
    if (!x) return 0;
    if (ql <= l && qr >= r) return 1;
    int mid = l + r >> 1, res = 0;
    if (ql <= mid) res |= qry(ls[x], l, mid, ql, qr);
    if (qr > mid) res |= qry(rs[x], mid + 1, r, ql, qr);
    return res;
}
char s[N];
int n, b[N], x[N], r[N], top[N], dp[N];
void solve() {
    int ans = 1;
    cnt = 0, lst = ++cnt;
    scanf("%d", &n);
    scanf("%s", s + 1);
    for (int i = 1; i <= n; i++) {
        extend(Tr(s[i]), i);
        upd(rt[lst], 1, n, i);
    }
```

```
    for (int i = 1; i <= cnt; i++) b[l[i]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = 1; i <= cnt; i++) x[b[l[i]]--] = i;
    for (int i = cnt; i > 1; i--) rt[f[x[i]]] = merge(rt[f[x[i]]], rt[x[i]]);
    for (int i = 2; i <= cnt; i++) {
        int u = x[i], fu = f[u];
        if (fu == 1) {
            dp[u] = 1, top[u] = u;
            continue;
        }
        int t = qry(rt[top[fu]], 1, n, pos[u] - l[u] + l[top[fu]], pos[u] - 1);
        dp[u] = dp[fu] + t;
        top[u] = t ? u : top[fu];
        ans = max(ans, dp[u]);
    }
    printf("%d\n", ans);
}
int main() {
    solve();
    return 0;
}
```

1.19 区间非重子串数量

```
/*
    Problem:
        求区间本质不同的子串数量
    Solution:
        我们用线段树维护每个节点为左端点的，到r为右端点为止的本质不同子串数量
        每个相同的子串只将值保留在最后一次出现的左端点
        当一个字符被新增到原串的末尾时，会在某些左端点增加新的本质不同的串
        同时部分子串最后一次出现的左端点将移动到串尾
        我们对[1,pos]前缀的所有左端点答案+1，考虑减去重复的串
        对于串[v,r]，如果出现在[l,r-v+l]，那么两者在后缀自动机fail树上的lca就是两者的重复串集
        在每个串位置记录最后一次出现的位置ps，我们从根到当前叶节点按照长度处理重复串，更新对应right集新的位置
        我们发现重复串和LCT的access操作一致，只会发生O(log(n))段ps的变动
        我们通过LCT的access操作保存在fail链上上一次串的出现位置，提取变动段，用线段树维护即可
        复杂度O(nlog^2n)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 100000 + 10;
struct Ask {
    int l, r, id;
```

```
    friend bool operator<(Ask &a, Ask &b) { return a.r < b.r; };
} ask[N];
ll ans[N];
char s[N];
// SAM
int Tr(char c) { return c - 'a'; }
int cnt, lst, a[N << 1][26], l[N << 1], f[N << 1], mrk[N << 1];
void extend(int c) {
    int p = lst;
    int np = lst = ++cnt;
    l[np] = l[p] + 1;
    mrk[np] = 1;
    while (!a[p][c] && p) a[p][c] = np, p = f[p];
    if (!p)
        f[np] = 1;
    else {
        int q = a[p][c];
        if (l[p] + 1 == l[q])
            f[np] = q;
        else {
            int nq = ++cnt;
            l[nq] = l[p] + 1;
            mrk[nq] = 0;
            memcpy(a[nq], a[q], sizeof(a[q]));
            f[nq] = f[q];
            f[np] = f[q] = nq;
            while (a[p][c] == q) a[p][c] = nq, p = f[p];
        }
    }
}
// Segment Tree
ll tag[N << 1], sum[N << 1];
int idx(int l, int r) { return l + r | l != r; }
void build(int l, int r) {
    int x = idx(l, r), mid = l + r >> 1;
    tag[x] = sum[x] = 0;
    if (l == r) return;
    build(l, mid);
    build(mid + 1, r);
}
void add_tag(int l, int r, ll v) {
    int x = idx(l, r);
    tag[x] += v;
    sum[x] += v * (r - l + 1);
}
void pd(int l, int r) {
```

```
    int x = idx(l, r), mid = l + r >> 1;
    if (tag[x]) {
        add_tag(l, mid, tag[x]);
        add_tag(mid + 1, r, tag[x]);
        tag[x] = 0;
    }
}
void up(int l, int r) {
    int mid = l + r >> 1;
    sum[idx(l, r)] = sum[idx(l, mid)] + sum[idx(mid + 1, r)];
}
void upd(int l, int r, int ul, int ur, ll v) {
    int x = idx(l, r), mid = l + r >> 1;
    if (ul <= l && r <= ur) {
        add_tag(l, r, v);
        return;
    }
    pd(l, r);
    if (ul <= mid) upd(l, mid, ul, ur, v);
    if (mid < ur) upd(mid + 1, r, ul, ur, v);
    up(l, r);
}
ll qry(int l, int r, int ql, int qr) {
    int x = idx(l, r), mid = l + r >> 1;
    if (ql <= l && r <= qr) return sum[x];
    pd(l, r);
    ll res = 0;
    if (ql <= mid) res += qry(l, mid, ql, qr);
    if (mid < qr) res += qry(mid + 1, r, ql, qr);
    return res;
}
// LCT
pair<int, int> mp[N];
int fa[N << 1], son[N << 1][2], tmp[N << 1];
int cov[N << 1], ps[N << 1], len[N << 1];
void cov1(int x, int v) {
    cov[x] = v;
    ps[x] = v;
}
void pb(int x) {
    if (cov[x]) {
        cov1(son[x][0], cov[x]);
        cov1(son[x][1], cov[x]);
        cov[x] = 0;
    }
}
```

```
void lct_up(int x) {
    if (son[x][0]) ps[x] = max(ps[x], ps[son[x][0]]);
    if (son[x][1]) ps[x] = max(ps[x], ps[son[x][1]]);
}
void rotate(int x) {
    int y = fa[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) fa[son[x][w ^ 1]] = y;
    if (fa[y]) {
        int z = fa[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    fa[x] = fa[y];
    fa[y] = x;
    son[x][w ^ 1] = y;
    lct_up(y);
}
bool isroot(int x) { return !fa[x] || son[fa[x]][0] != x && son[fa[x]][1] != x; }
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = fa[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = fa[x];
        if (!isroot(y)) {
            if ((son[fa[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    lct_up(x);
}
int mcnt;
void access(int x, int v) {
    mcnt = 0;
    cov1(x, v);
    for (int y = 0; x; y = x, x = fa[x]) {
        splay(x);
        son[x][1] = y;
        mp[++mcnt] = make_pair(len[x], ps[x]);
```

```
        lct_up(x);
    }
}
int n, q;
void lct_init() {
    for (int i = 0; i <= 2 * n; ++i) {
        son[i][0] = son[i][1] = fa[i] = 0;
        ps[i] = cov[i] = len[i] = 0;
    }
}
int b[N << 1], x[N << 1];
int ver[N << 1], id[N << 1];
void init() {
    scanf("%d%d", &n, &q);
    scanf("%s", s + 1);
    lct_init();
    build(1, n);
    memset(l, 0, sizeof(int) * (cnt + 1));
    memset(f, 0, sizeof(int) * (cnt + 1));
    memset(b, 0, sizeof(int) * (cnt + 1));
    cnt = 0, lst = ++cnt;
    for (int i = 1; i <= n; i++) extend(Tr(s[i]));
    for (int i = 1; i <= cnt; i++) b[l[i]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = 1; i <= cnt; i++) x[b[l[i]]--] = i;
    for (int i = 1; i <= cnt; ++i) {
        id[x[i]] = i;
        if (mrk[x[i]]) ver[l[x[i]]] = i;
    }
    for (int i = 1; i <= cnt; i++) {
        len[i] = l[x[i]];
        if (f[x[i]]) fa[i] = id[f[x[i]]];
    }
    int l, r;
    for (int i = 1; i <= q; i++) {
        scanf("%d%d", &l, &r);
        ask[i] = {l + 1, r + 1, i};
    }
    sort(ask + 1, ask + q + 1);
}
void solve() {
    int p = 1;
    for (int i = 1; i <= n; i++) {
        upd(1, n, 1, i, 1);
        access(ver[i], i);
        int pre = 0;
```

```
        for (int j = mcnt; j > 1; j--) {
            if (mp[j].first == 0) continue;
            if (mp[j].second != 0)
                upd(1, n, mp[j].second - mp[j].first + 1, mp[j].second - pre,
                    -1);
            pre = mp[j].first;
        }
        while (p <= q && ask[p].r == i) {
            ans[ask[p].id] = qry(1, n, ask[p].l, ask[p].r);
            p++;
        }
    }
    for (int i = 1; i <= q; i++) printf("%lld\n", ans[i]);
}
int main() {
    init();
    solve();
    return 0;
}
```

1.20 子序列自动机

```
/*
    子序列自动机
    复杂度O(n*SIGMA)
*/
#include <bits/stdc++.h>
using namespace std;
struct DASG {
    static const int SIZE = 100000 + 10, SIGMA = 26;
    struct Node {
        Node *ch[SIGMA];
        Node() { memset(ch, 0, sizeof(ch)); }
    } pool[SIZE], *sz, *rt, *last[SIGMA];
    void init() {
        sz = pool;
        rt = sz++;
        std::fill(last, last + SIGMA, rt);
    }
    void add(int x) {
        Node *np = sz++;
        for (Node *p = last[x]; p < np; ++p) {
            if (!p->ch[x]) p->ch[x] = np;
        }
        last[x] = np;
```

```
    }
} dasg;
char s[100010];
int m;
int main() {
    scanf("%s", s);
    int n = strlen(s);
    dasg.init();
    for (int i = 0; i < n; i++) dasg.add(s[i] - 'a');
    scanf("%d", &m);
    while (m--) {
        scanf("%s", s);
        int len = strlen(s);
        DASG::Node *x = dasg.rt;
        for (int i = 0; i < len; i++) {
            if (!x->ch[s[i] - 'a']) {
                puts("No");
                break;
            }
            x = x->ch[s[i] - 'a'];
            if (i == len - 1) puts("Yes");
        }
    }
    return 0;
}
// 求串s的最长的子序列使得其字典序大于串t
#include <bits/stdc++.h>
using namespace std;
const int N = 1000010;
int ps[N];
int ch[N][26];
int sz, rt, lst[26];
void init() {
    sz = 0;
    rt = sz++;
}
void add(int x, int pos) {
    ps[sz] = pos;
    int np = sz++;
    for (int p = lst[x]; p < np; ++p) {
        if (!ch[p][x]) ch[p][x] = np;
    }
    lst[x] = np;
}
char s[N], t[N];
int n, m;
```

```
int main() {
    scanf("%d%d", &n, &m);
    scanf("%s", s);
    scanf("%s", t);
    int ans = -1;
    init();
    for (int i = 0; i < n; i++) add(s[i] - 'a', i);
    int x = rt;
    for (int i = 0; i < m; i++) {
        for (int j = t[i] - 'a' + 1; j < 26; j++) {
            if (!ch[x][j]) continue;
            ans = max(ans, n - ps[ch[x][j]] + i);
        }
        if (!ch[x][t[i] - 'a']) break;
        x = ch[x][t[i] - 'a'];
        if (i == m - 1 && ps[x] != n - 1) ans = max(ans, n - ps[x] + i);
    }
    printf("%d\n", ans);
    return 0;
}
```

1.21 Manacher

```
/*
    Manacher算法
    r数组为奇回文半径
*/
int n, f[N << 1], r[N << 1];
char s[N], c[N << 1];
void manacher() {
    for (int i = 1; i <= n; i++) c[i << 1] = s[i], c[(i << 1) + 1] = '#';
    c[1] = '#';
    c[n << 1 | 1] = '#';
    c[0] = '&';
    c[(n + 1) << 1] = '$';
    int j = 0, k;
    n = n << 1 | 1;
    for (int i = 1; i <= n;) {
        while (c[i - j - 1] == c[i + j + 1]) j++;
        r[i] = j;
        for (k = 1; k <= j && r[i] - k != r[i - k]; k++)
            r[i + k] = min(r[i - k], r[i] - k);
        i += k;
        j = max(j - k, 0);
    }
```

```
}
/*
   求最长双倍回文
   （记Wr为W串的倒置，求最长的形如WWrWWr的串的长度）
*/
int ans;
int sf(int x) { return f[x] == x ? x : f[x] = sf(f[x]); }
int main() {
    while (~scanf("%d", &n)) {
        scanf("%s", s + 1);
        manacher();
        ans = 0;
        for (int i = 1; i <= n; i++) f[i] = (c[i] == '#') ? i : (i + 1);
        for (int i = 3; i < n; i += 2) {
            int j = sf(max(i - (r[i] >> 1), 1));
            for (; j < i && j + r[j] < i; f[j] = sf(j + 1), j = f[j])
                ;
            if (j < i)
                if ((i - j) << 1 > ans) ans = (i - j) << 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Problem:
        求每一种旋转串的最长回文子串
    Solution:
        倍长字符串，转化为区间回文子串，manacher预处理倍长后的回文串
        对于查询区间[L,R]最长回文子串
        我们可以二分答案，检验区间[L+x-1,R-x+1]中是否存在长度大于x的回文中心即可
        ST表预处理r数组的区间最大值，复杂度O(nlogn)
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000 + 10;
int n, f[N << 1], r[N << 1];
char s[N], c[N << 1];
int dp[N << 1][21], lg2[N << 1];
void Init(int n) {
    for (int i = 2; i <= n; i++) lg2[i] = lg2[i / 2] + 1;
    for (int i = 1; i <= n; i++) dp[i][0] = r[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
```

```
int Max(int l, int r) {
    if (l > r) swap(l, r);
    int k = lg2[r - l + 1];
    return max(dp[l][k], dp[r - (1 << k) + 1][k]);
}
void manacher() {
    for (int i = 1; i <= n; i++) c[i << 1] = s[i], c[(i << 1) + 1] = '#';
    c[1] = '#';
    c[n << 1 | 1] = '#';
    c[0] = '&';
    c[(n + 1) << 1] = '$';
    int j = 0, k;
    n = n << 1 | 1;
    for (int i = 1; i <= n;) {
        while (c[i - j - 1] == c[i + j + 1]) j++;
        r[i] = j;
        for (k = 1; k <= j && r[i] - k != r[i - k]; k++)
            r[i + k] = min(r[i - k], r[i] - k);
        i += k;
        j = max(j - k, 0);
    }
}
int main() {
    scanf("%d%s", &n, s + 1);
    int m = n;
    n = strlen(s + 1);
    for (int i = n + 1; i <= 2 * n; i++) s[i] = s[i - n];
    n = n * 2;
    manacher();
    Init(n);
    // 区间[i,i+n-1]的最长回文子串
    for (int i = 1; i <= m; i++) {
        int l = 1, r = m, ans = 1;
        int L = 2 * i, R = 2 * (i + m - 1);
        while (l <= r) {
            int mid = l + r >> 1;
            if (Max(L + mid - 1, R - mid + 1) >= mid)
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.22 Manacher+ 树状数组

```
/*
    Manacher+BIT
    问最少用几个回文串可以构成给出串，重叠部分可以合并
*/
char s[100010], a[50010];
const int INF = 0x3f3f3f3f;
int cnt, n, c[100010], f[100010];
struct seg {
    int l, r;
} l[100010];
void add_edge(int x, int y) {
    x = x / 2 + 1;
    y = y / 2 - 1;
    if (x > y) return;
    l[++cnt] = (seg){x, y};
}
bool operator<(seg a, seg b) { return a.r < b.r; }
void update(int x, int val) {
    while (x) c[x] = min(c[x], val), x -= x & -x;
}
int query(int x) {
    if (!x) return 0;
    int res = INF;
    while (x <= n) res = min(c[x], res), x += x & -x;
    return res;
}
void manacher(char *a) {
    int m, r, p, i;
    for (i = 1; i <= n; i++) s[i << 1] = a[i], s[i << 1 | 1] = '#';
    s[0] = '$', s[1] = '#', s[m = (n + 1) << 1] = '@';
    for (r = p = 0, f[1] = 1, i = 2; i < m; i++) {
        for (f[i] = r > i ? min(r - i, f[p * 2 - i]) : 1;
             s[i - f[i]] == s[i + f[i]]; f[i]++)
            ;
        add_edge(i - f[i], i + f[i]);
        if (i + f[i] > r) r = i + f[i], p = i;
    }
}
int main() {
    while (~scanf("%s", a + 1)) {
        n = strlen(a + 1);
        cnt = 0;
        for (int i = 1; i <= n; i++) c[i] = INF;
        manacher(a);
```

```
        sort(l + 1, l + cnt + 1);
        int ans = INF, tmp;
        for (int i = 1; i <= cnt; i++) {
            update(l[i].r, tmp = query(l[i].l - 1) + 1);
            if (l[i].r == n) ans = min(ans, tmp);
        }
        printf("%d\n", ans - 1);
    }
    return 0;
}
```

1.23 反对称子串

```
/*
    反对称子串
    将一个01串的01取反之后将串反过来跟原串一样
    求一个字符串的反对称子串的数量
*/
int n, m, i, r, p, f[N << 1];
long long ans;
char a[N], s[N << 1];
int min(int a, int b) { return a < b ? a : b; }
bool check(char x, char y) {
    if (x == '#' && y == '#') return 1;
    if ((x - '0') + (y - '0') == 1) return 1;
    return 0;
}
void manacher(char *a) {
    for (i = 1; i <= n; i++) s[i << 1] = a[i], s[i << 1 | 1] = '#';
    s[0] = '$', s[1] = '#', s[m = (n + 1) << 1] = '&';
    for (r = p = 0, f[1] = 1, i = 1; i<m; ans += f[i++]>> 1) {
        for (f[i] = r > i ? min(r - i, f[p * 2 - i]) : 0;
             check(s[i - f[i]], s[i + f[i]]); f[i]++)
            ;
        if (i + f[i] > r) r = i + f[i], p = i;
    }
}
int main() {
    scanf("%d", &n);
    scanf(" %s", a + 1);
    manacher(a);
    printf("%lld\n", ans);
    return 0;
}
```

1.24 回文自动机

```
/*
    回文自动机
    本质为回文后缀自动机，可处理回文后缀关系和回文统计
    attention: 使用时加入字符需要做char to int转化
    add(s[i] - 'a') 返回值为当前字符插入尾部形成的最长回文串后缀的id

    N: 串长。
    S: 字符集大小。
    text[1..all]: 字符串。
    son[x][y]: 第 x 个点所代表的回文串两边加上字符 y 后的回文串。
    fail[x]: 第 x 个点所代表的回文串的最长回文后缀。
    cnt[x]: 第 x 个点所代表的回文串的出现次数（需建完树后 count() 一遍才可以）。
    len[x]: 第 x 个点所代表的回文串的长度。
    num[x]: 第 x 个点所代表的回文串的后缀回文的数量
    l[x]: 第 x 个点所代表的回文串的回文后缀的长度和（取模）
    half[x]: 第 x 个点所代表的回文串的小于等于串长一半的最长回文后缀

    l数组和num数组的应用
    id = add(s[i] - 'a')
    R[i] = num[id] * (i + 1) - l[id]
    可以计算出插入一个字符后以当前位置为右端点的回文串左端点之和

    fail数组的应用
    可以fail[x]向x连边形成fail树，根为0
    一个点的祖先节点为它的回文后缀

*/
const int N = 1000010, S = 26;
int l[N], half[N];
int all, son[N][S], fail[N], len[N], text[N], num[N], cnt[N], lst, tot;
int newnode(int _l) {
    for (int i = 0; i < S; i++) son[tot][i] = 0;
    l[tot] = num[tot] = cnt[tot] = 0, len[tot] = _l;
    return tot++;
}
void init() {
    lst = tot = all = 0;
    newnode(0), newnode(-1);
    text[0] = -1, fail[0] = 1;
}
int getfail(int x) {
    while (text[all - len[x] - 1] != text[all]) x = fail[x];
    return x;
}
```

```
void gethalf(int y, int x, int w) {
    half[y] = son[getfail(half[x])][w];
    while (len[half[y]] * 2 > len[y]) half[y] = fail[half[y]];
}
int add(int w) {
    text[++all] = w;
    int x = getfail(lst);
    if (!son[x][w]) {
        int y = newnode(len[x] + 2);
        fail[y] = son[getfail(fail[x])][w];
        son[x][w] = y;
        l[y] = (1ll * l[fail[y]] + len[y]) % P;
        num[y] = num[fail[y]] + 1;
        gethalf(y, x, w);
    }
    lst = son[x][w];
    cnt[lst]++;
    return lst;
}
void count() {
    for (int i = tot - 1; ~i; i--) cnt[fail[i]] += cnt[i];
}
/*
    Example(回文串划分方案数)
    题目大意：给出一个串s，现在要求将其划分子串，并且划分结果呈回文
    求方案数，如abcdabcd划分为ab，cd，cd，ab，为回文。
    题解：我们将串后半部分倒序依次插入前半部分的后面，比如abcdcdab，
    将构成abbacddc，那么问题就转化为新串能拆分成回文子串的方案数，
    我们对新串边构建回文自动机，并在构建的同时计算答案，
    diff数组表示节点与其失配位置最长后缀回文的差值，
    记anc为将连续相同差值去除后的祖先节点，比如abbabbabba，
    在去除连续相同差值abb之后，得到祖先节点a，
    则对于一种差值长度的答案来说，若失配位置为其祖先节点，则答案等于失配节点的答案
    否则其值为拿掉祖先节点加一倍差值之后的位置的答案ans，加上其失配节点的答案g，
    对于不同差值长度的节点答案g累积到当前位置答案ans上去，
    最后输出结尾位置的答案即可。
*/
using namespace std;
const int N = 1000010;
const int mod = 1000000007;
int n, cnt, lst, s[N], l[N], f[N], diff[N], anc[N], g[N], ans[N], a[N][26];
char S[N];
void add(int &x, int y) {
    if ((x += y) >= mod) x -= mod;
}
void init() {
```

```
    lst = cnt = 1;
    f[0] = 1, f[1] = 0;
    l[1] = -1;
}
void extend(int np, int c) {
    int p = lst;
    while (s[np] != s[np - l[p] - 1]) p = f[p];
    if (!a[p][c]) {
        int x = ++cnt, fp = f[p];
        while (s[np] != s[np - l[fp] - 1]) fp = f[fp];
        l[x] = l[p] + 2;
        f[x] = a[fp][c];
        a[p][c] = x;
        diff[x] = l[x] - l[f[x]];
        anc[x] = diff[x] == diff[f[x]] ? anc[f[x]] : f[x];
    }
    lst = a[p][c];
}
int main() {
    scanf("%s", S + 1);
    n = strlen(S + 1);
    for (int i = 1; i * 2 <= n; i++) {
        s[i * 2 - 1] = S[i] - 'a';
        s[i * 2] = S[n - i + 1] - 'a';
    }
    s[0] = -1;
    ans[0] = 1;
    init();
    for (int i = 1; i <= n; i++) {
        extend(i, s[i]);
        for (int x = lst; x; x = anc[x]) {
            g[x] = ans[i - l[anc[x]] - diff[x]];
            if (anc[x] != f[x]) add(g[x], g[f[x]]);
            if (i % 2 == 0) add(ans[i], g[x]);
        }
    }
    printf("%d\n", ans[n]);
    return 0;
}
```

1.25 双端回文树

```
/*
    双端回文树
    Operation
```

```
        1. 在串首插入字符
        2. 在串尾插入字符
        3. 查询非重回文串数量
        4. 查询回文串数量
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 200010, S = 26;
struct Palindromic_Tree {
    int son[N][S], fail[N], len[N];
    int text[N], L, R, num[N], lst[2], p;
    LL tot;
    int newnode(int l) {
        for (int i = 0; i < S; i++) son[p][i] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }
    void init(int n) {
        p = 0;
        newnode(0);
        newnode(-1);
        memset(text, -1, sizeof(text));
        L = n;
        R = n - 1;
        lst[0] = lst[1] = 1;
        fail[0] = 1;
        tot = 0;
    }
    int get_fail(int v, bool d) {
        if (d)
            while (text[R - len[v] - 1] != text[R]) v = fail[v];
        else
            while (text[L + len[v] + 1] != text[L]) v = fail[v];
        return v;
    }
    void add(int c, bool d) {
        if (d)
            text[++R] = c;
        else
            text[--L] = c;
        int x = get_fail(lst[d], d);
        if (!son[x][c]) {
            int y = newnode(len[x] + 2);
            fail[y] = son[get_fail(fail[x], d)][c];
```

```
            son[x][c] = y;
            num[y] = num[fail[y]] + 1;
        }
        lst[d] = son[x][c];
        if (len[lst[d]] == R - L + 1) lst[d ^ 1] = lst[d];
        tot += num[lst[d]];
    }
};
Palindromic_Tree T;
int n;
void solve() {
    int op;
    char c;
    T.init(n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &op);
        if (op <= 2) {
            scanf(" %c", &c);
            T.add(c - 'a', op - 1);
        }
        if (op == 3) printf("%d\n", T.p - 2);
        if (op == 4) printf("%lld\n", T.tot);
    }
}
int main() {
    while (~scanf("%d", &n)) solve();
    return 0;
}
```

1.26 最小最大表示法

```
/*
    最小最大表示法
    flag=1: 表示最小表示法, flag=0: 最大表示法
*/
int Min_Max_Express(char *s, int len, bool flag) {
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len) {
        int t = s[(j + k) % len] - s[(i + k) % len];
        if (t == 0)
            k++;
        else {
            if (flag) {
                if (t > 0)
                    j += k + 1;
```

```
            else
                i += k + 1;
        } else {
            if (t > 0)
                i += k + 1;
            else
                j += k + 1;
        }
        if (i == j) j++;
        k = 0;
    }
  }
  return min(i, j);
}
```

1.27 Lyndon-Word 生成器

```
// 按字典序生成长度为n以内的Lyndonword
void lyndon_generate(int n, int m) {
    char z = 'a' + m - 1, s[1000];
    s[0] = 'a' - 1;
    for (int i = 1, x = 1;; ++i) {
        s[x - 1]++;
        s[x] = 0;
        puts(s);
        for (int j = x; j < n; ++j) s[j] = s[j - x];
        for (x = n; s[x - 1] == z; --x);
    }
}
```

1.28 Lyndon 划分

```
/*
    Duval算法
        求出一个字符串的Lyndon分解
    Lyndon串
        串本身小于所有的循环位移结果
*/
#include <bits/stdc++.h>
using namespace std;
std::vector<int> duval(char s[]) {
    std::vector<int> ret;
    int n = strlen(s) + 1; // zero used here
```

```
    int start = 0, mid = 1, cur = 0;
    ret.push_back(0);
    for (int i = 0; i < n; ++i) {
        if (s[i] == s[cur]) {
            if (++cur == mid) cur = start;
        } else if (s[i] > s[cur]) {
            mid = i + 1;
            cur = start;
        } else if (s[i] < s[cur]) {
            int temp = mid - start;
            while (start + temp <= i) {
                start += temp;
                ret.push_back(start);
                // 如果输出分界点直接在里面输出比较快
                // printf("%d ",start);
            }
            i = cur = start;
            mid = start + 1;
        }
    }
    return ret;
}
char s[2000100];
int main() {
    scanf("%s", s);
    auto res = duval(s);
//     for (auto v : res) {
//             if (v) printf("%d%c", v, " \n"[v==strlen(s)]);
//     }
    return 0;
}
```

1.29 ShiftAnd

```
/*
    ShiftAnd
    柔性字符串匹配
    给出一个字符串，找出其中所有的符合特定模式的子串位置，
    符合特定模式是指，该子串的长度为n，并且第i个字符需要在给定的字符集合Si中
*/
const int M = 510, N = 2000010, L = 65, U = 256;
bitset<M> dp[2], bt[U];
int n, m, id[U], cnt, l;
char s[N], t[L];
void init() {
```

```
    cnt = 0;
    for (int i = '0'; i <= '9'; i++) id[i] = ++cnt;
    for (int i = 'A'; i <= 'Z'; i++) id[i] = ++cnt;
    for (int i = 'a'; i <= 'z'; i++) id[i] = ++cnt;
}
void ShiftAnd(int n, int m) {
    int cur = 1, f = 0;
    dp[0].reset();
    dp[0].set(0);
    for (int i = 1; i <= n; i++, cur ^= 1) {
        dp[cur] = dp[cur ^ 1] << 1 & bt[id[s[i]]];
        dp[cur].set(0);
        if (dp[cur][m]) printf("%d\n", i - m + (f = 1));
    }
    if (!f) puts("NULL");
}
int main() {
    init();
    while (gets(s + 1)) {
        n = strlen(s + 1);
        scanf("%d", &m);
        for (int i = 1; i <= cnt; i++) bt[i].reset();
        for (int i = 1; i <= m; i++) {
            scanf("%d %s", &l, t);
            for (int j = 0; j < l; j++) bt[id[t[j]]].set(i);
        }
        ShiftAnd(n, m);
        gets(s);
    }
    return 0;
}
```

1.30 Hash

```
/*
    字符串哈希
    循环同构
    Problem:
        给出两个字符串，判断他们每一个前缀是否循环同构，
        循环同构的意思就是，字符串首位相接拼成一个环，两个环通过旋转可以相等。
    Solution:
        这道题用到了一个神奇的结论，如果S字符串和T字符串循环同构，
        那么必有S=u+v，T=v+u，而且u和v必有一个是最长匹配。
        那么根据这个结论，我们可以用KMP算法在T中找S的最长前缀，
        也就是每次匹配到i时候的j。那么对于T的前缀和S的后缀，
```

```
        我们直接用哈希来判断是否相等，这样子就可以计算出两个串的前缀是否循环同构了。
*/
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 10005, base = 233;
typedef long long ll;
int T, nxt[N], ans[N];
char a[N], b[N];
ll hash[2][N], p[N];
ll get_hash(int t, int L, int R) {
    return hash[t][R] - hash[t][L - 1] * p[R - L + 1];
}
int check(int t, int m, int n) {
    if (m == n) return 1;
    return get_hash(t, m + 1, n) == get_hash(t ^ 1, 1, n - m);
}
void kmp(int n, char* a, int m, char* b, int t) {
    int i, j;
    for (nxt[1] = j = 0, i = 2; i <= n; nxt[i++] = j) {
        while (j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) j++;
    }
    for (j = 0, i = 1; i <= m; i++) {
        while (j && a[j + 1] != b[i]) j = nxt[j];
        if (a[j + 1] == b[i]) {
            j++;
            if (!ans[i]) ans[i] = check(t, j, i);
        }
        if (j == n) j = nxt[j];
    }
}
int main() {
    for (int i = p[0] = 1; i < N; i++) p[i] = p[i - 1] * base;
    while (~scanf(" %s %s", a + 1, b + 1)) {
        int n = strlen(a + 1);
        for (int i = 1; i <= n; i++) {
            ans[i] = 0;
            hash[0][i] = hash[0][i - 1] * base + a[i];
            hash[1][i] = hash[1][i - 1] * base + b[i];
        }
        kmp(n, a, n, b, 0);
        kmp(n, b, n, a, 1);
        for (int i = 1; i <= n; i++) printf("%d", ans[i]);
        puts("");
    }
```

```
    return 0;
}
/*
    集合哈希
    Problem:
        规定一个字符串和另一个字符串匹配的条件为首尾两个字符相同，且所有字符的出现次数相同
        现在给定一个母串，询问多个子串，问子串在母串中的匹配次数
        子串长度和 \le 10^5，母串长度 \le 10^5
    Solution:
        我们将询问串按照长度分组，对于每种分组，我们计算母串中对应长度的哈希值
        然后统计对应组询问在该长度子串的哈希值中出现次数即可
        我们对除串首尾做字符的集合哈希，串首尾特殊处理累加到哈希值上去
        考虑子串长度和限制，复杂度均摊，最坏情况O(n\sqrt{n})
*/
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
const ull base = 131;
const int N = 100000 + 10;
int n, T, L[N], ans[N];
ull p[125], Hash[N], h[N];
char s[N], t[N];
vector<int> id[N];
int main() {
    scanf("%d", &T);
    for (int i = p[0] = 1; i < 125; i++) p[i] = p[i - 1] * base;
    while (T--) {
        scanf("%s", s);
        scanf("%d", &n);
        int len = strlen(s);
        for (int i = 1; i <= n; i++) {
            scanf("%s", t);
            Hash[i] = 0;
            L[i] = strlen(t);
            id[L[i]].push_back(i);
            for (int j = 1; j < L[i] - 1; j++) Hash[i] += p[t[j]];
            Hash[i] += p[123] * t[0];
            Hash[i] += p[124] * t[L[i] - 1];
        }
        sort(L + 1, L + n + 1);
        int cnt = unique(L + 1, L + n + 1) - (L + 1);
        for (int i = 1; i <= cnt; i++) {
            ull H = 0;
            int tot = 0;
            for (int j = 0; j < len; j++) {
                if (j >= L[i] - 1) {
```

```
                H -= p[s[j - L[i] + 1]];
                h[++tot] = H + p[123] * s[j - L[i] + 1] + p[124] * s[j];
            }
            H += p[s[j]];
        }
        sort(h + 1, h + tot + 1);
        for (auto x : id[L[i]]) {
            int p1 = upper_bound(h + 1, h + tot + 1, Hash[x]) - h;
            int p2 = lower_bound(h + 1, h + tot + 1, Hash[x]) - h;
            ans[x] = p1 - p2;
        }
        id[L[i]].clear();
    }
    for (int i = 1; i <= n; i++) printf("%d\n", ans[i]);
  }
  return 0;
}
/*
  相似串哈希
  Problem:
    两个字符串a和b相似当且仅当两个串长度相等
    且a[i]等于a[j]时b[i]等于b[j]
    a[i]不等于a[j]时b[i]也不等于b[j]
    给定一个字符串，多次询问其某个子串在原串中相似匹配的次数
  Solution:
    对于每个子串，我们根据每个字母第一次出现的位置对其标号
    用标号来表示这个子串，那么如果标号序列的哈希值相同则子串相等
    当子串起始位置不同时，我们会得到不同的标号，我们发现标号序列存在后缀递推关系
    因此我们从后往前预处理这个标号，得到每个位置作为子串起点时标号对字母的映射
    维护每个字符的位置哈希，对于两个子串的比较，只要比较子串区间内所有对应标号的字符的位置哈希是否相同即可
    我们对所有后缀按照相似匹配的定义进行排序，那么包含与查询子串相似的串的后缀一定是连续的段
    对于查询一个子串相似匹配的次数，我们只要在后缀排名中二分找到这样的相似段的左右端点即可
    即两个后缀的最长匹配大于等于子串长度的最远位置，求最长匹配的过程可以二分加速
*/
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
const int N = 100000 + 10, C = 10;
const int base = 131;
int n, m, a[N][C], id[N], rk[N];
ull p[N], b[N][C];
char s[N];
int getmax(int x, int y) {
  if (x > y) swap(x, y);
  int r = n - y + 1, l = 0, ans = 0, mid, h = 0;
  while (l <= r) {
```

```
        mid = (l + r) / 2;
        h = 0;
        for (int i = 0; h == i && i < C; i++)
            h += (((b[x + mid - 1][a[x][i]] - b[x - 1][a[x][i]]) * p[y - x] -
                  b[y + mid - 1][a[y][i]] + b[y - 1][a[y][i]]) == 0);
        if (h == C)
            l = mid + 1, ans = mid;
        else
            r = mid - 1;
    }
    return ans;
}
bool cmp(int x, int y) {
    int d = getmax(x, y);
    if (d == n - max(x, y) + 1) return x > y;
    int _A = 0, _B = 0;
    for (int i = 0; i < C; i++) {
        if (s[x + d] == a[x][i]) _A = i;
        if (s[y + d] == a[y][i]) _B = i;
    }
    return _A < _B;
}
void init() {
    for (int i = p[0] = 1; i < N; i++) p[i] = p[i - 1] * base;
    for (int i = 1; i <= n; i++) s[i] -= 'a';
    for (int i = 0; i < C; i++) a[n + 1][i] = i;
    for (int i = n; i > 0; i--) {
        int pos = 0;
        for (int j = 0; j < C; j++) {
            a[i][j] = a[i + 1][j];
            if (a[i][j] == s[i]) pos = j;
        }
        while (pos--) swap(a[i][pos], a[i][pos + 1]);
    }
    for (int i = 1; i <= n; i++) {
        b[i][s[i]] = p[i];
        for (int j = 0; j < C; j++) b[i][j] = b[i][j] + b[i - 1][j];
    }
    for (int i = 1; i <= n; i++) id[i] = i;
    stable_sort(id + 1, id + n + 1, cmp);
    for (int i = 1; i <= n; i++) rk[id[i]] = i;
}
int main() {
    scanf("%d%d", &n, &m);
    scanf("%s", s + 1);
    init();
```

```
    while (m--) {
        int l, r, ans;
        scanf("%d%d", &l, &r);
        int x = rk[l];
        int d = r - l + 1;
        l = ans = 1;
        r = x;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (getmax(id[x], id[mid]) >= d)
                ans = mid, r = mid - 1;
            else
                l = mid + 1;
        }
        int L = ans;
        l = x;
        r = n;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (getmax(id[x], id[mid]) >= d)
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        int R = ans;
        printf("%d\n", R - L + 1);
    }
    return 0;
}
```

1.31 二维 Hash

```
/*
    二维字符串Hash
    找到矩阵中出现至少两次的正方形，输出边长
    改变x和y可以查询矩形
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 1010;
typedef unsigned long long ll;
const ll D1 = 197, D2 = 131;
int n, m, x, y, i, j, ans, t, cnt;
```

```
char a[N][N];
ll pow1[N], pow2[N], h[N][N], tmp;
struct P {
    int x, y;
    ll z;
    P() {}
    P(int _x, int _y, ll _z) { x = _x, y = _y, z = _z; }
} q[N * N], fin[N * N];
bool cmp(const P& a, const P& b) { return a.z < b.z; }
bool cmp2(const P& a, const P& b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }
int main() {
    while (~scanf("%d%d", &n, &m)) {
        gets(a[0]);
        for (i = 1; i <= n; i++) gets(a[i] + 1);
        for (pow1[0] = pow2[0] = i = 1; i <= n || i <= m; i++) {
            pow1[i] = pow1[i - 1] * D1, pow2[i] = pow2[i - 1] * D2;
        }
        int Ans = 0;
        int l = 1, r = min(n, m);
        while (l <= r) {
            int mid = (l + r) >> 1;
            ans = cnt = 0;
            x = y = mid; // x和y为矩形边长
            for (i = 1; i <= n; i++) {
                for (tmp = 0, j = 1; j < y; j++)
                    tmp = tmp * D1 + a[i][j], h[i][j] = 0;
                for (j = y; j <= m; j++)
                    h[i][j] = tmp = tmp * D1 - pow1[y] * a[i][j - y] + a[i][j];
            }
            for (t = 0, i = y; i <= m; i++) {
                for (tmp = 0, j = 1; j < x; j++) tmp = tmp * D2 + h[j][i];
                for (j = x; j <= n; j++)
                    q[++t] =
                        P(j - x + 1, i - y + 1,
                          tmp = tmp * D2 - pow2[x] * h[j - x][i] + h[j][i]);
            }
            for (sort(q + 1, q + t + 1, cmp), j = 1, i = 2; i <= t; i++)
                if (q[i - 1].z != q[i].z) {
                    if (i - j > ans) ans = i - j, tmp = q[j].z;
                    j = i;
                }
            if (t - j + 1 > ans) ans = t - j + 1, tmp = q[t].z;
            for (i = 1; i <= t; i++)
                if (q[i].z == tmp) fin[++cnt] = P(q[i].x, q[i].y, 0);
            if (cnt > 1)
                Ans = mid, l = mid + 1;
```

```
            else
                r = mid - 1;
            /*
                注释部分可输出该大小下最大的矩阵和出现位置
            */
            // sort(fin+1,fin+cnt+1,cmp2);
            // printf("%d\n",cnt);
            // for(i=0;i<x;puts(""),i++)for(j=0;j<y;j++)putchar(a[fin[1].x+i][fin[1].y+j]);
            // for(printf("%d\n",cnt),i=1;i<=cnt;i++)printf(" %d
            // %d\n",fin[i].x,fin[i].y);
        }
        printf("%d\n", Ans);
    }
    return 0;
}
```

1.32 正则表达式

```
# 正则表达式
# 示例：求仅包含ab的正则表达式能匹配的长度不超过n的串的数量
# Sample Input
# 3
# ((ab)|(ba)) 2
# ((a|b)*) 5
# ((a*)(b(a*))) 100
# Sample Output
# 2
# 32
# 100
# Code
from enum import Enum
from collections import namedtuple

Edge = namedtuple('Edge', 'dest char')

class Alphabet(Enum):
    a = 'a'
    b = 'b'
    e = None


def empty_edge(dest):
    return Edge(dest, Alphabet.e)
```

```
class CountStrings:
    def __init__(self, regex_string):
        RegexGraphNFA.node_count = 0
        self.regex_string = regex_string
        nfa_graph = self.translate_regex()
        translate_graph = TranslateGraph(nfa_graph)
        self.dfa_graph = translate_graph.translate()

    def calculate(self, string_length):
        return self.dfa_graph.count_paths(string_length)

    def translate_regex(self, index=0):
        result_set = ResultSet()
        while index < len(self.regex_string):
            if self.regex_string[index] == '(':
                out_list, index = self.translate_regex(index + 1)
                result_set.insert(out_list)
            elif self.regex_string[index] == ')':
                result = result_set.calculate_result()
                return result, index
            elif self.regex_string[index] == '|':
                result_set.set_or()
            elif self.regex_string[index] == '*':
                result_set.set_repeat()
            else:
                result_set.insert(RegexGraphNFA(Alphabet[self.regex_string[index]]))
            index += 1
        return result_set.calculate_result()


class ResultSet:
    AND, OR, REPEAT = 1, 2, 3

    def __init__(self):
        self.r1 = None
        self.r2 = None
        self.op = self.AND

    def set_or(self):
        self.op = self.OR

    def set_repeat(self):
        self.op = self.REPEAT

    def calculate_result(self):
        if self.op == self.REPEAT:
```

```
            self.calculate_repeat()
        elif self.r2 is None:
            pass
        elif self.op == self.OR:
            self.calculate_or()
        else:
            self.calculate_and()
        return self.r1

    def calculate_repeat(self):
        self.r1.graph_repeat()

    def calculate_or(self):
        self.r1.graph_or(self.r2)

    def calculate_and(self):
        self.r1.graph_add(self.r2)

    def insert(self, value):
        if self.r1 is None:
            self.r1 = value
        else:
            self.r2 = value


class RegexGraphNFA:
    node_count = 0

    def __init__(self, in_char):
        self.edges = None
        self.head = None
        self.tail = None
        self.insert_char(in_char)

    @classmethod
    def get_next_node_id(cls):
        node_id = cls.node_count
        cls.node_count += 1
        return node_id

    def insert_char(self, value):
        self.head = self.get_next_node_id()
        self.tail = self.get_next_node_id()
        self.edges = {self.head: [Edge(self.tail, value)],
                      self.tail: []}
```

```
def graph_add(self, other):
    join_node = self.get_next_node_id()
    self.join(other)
    self.edges[self.tail].append(empty_edge(join_node))
    self.edges[join_node] = [empty_edge(other.head)]
    self.tail = other.tail

def graph_repeat(self):
    new_head = self.get_next_node_id()
    new_tail = self.get_next_node_id()
    self.edges[self.tail].extend([empty_edge(self.head), empty_edge(new_tail)])
    self.edges[new_head] = [empty_edge(self.head), empty_edge(new_tail)]
    self.edges[new_tail] = []
    self.head = new_head
    self.tail = new_tail

def graph_or(self, other):
    new_head = self.get_next_node_id()
    new_tail = self.get_next_node_id()
    self.join(other)
    self.edges[new_head] = [empty_edge(self.head), empty_edge(other.head)]
    self.edges[self.tail].append(empty_edge(new_tail))
    self.edges[other.tail].append(empty_edge(new_tail))
    self.edges[new_tail] = []
    self.head = new_head
    self.tail = new_tail

def join(self, other):
    for node, edge in other.edges.items():
        if node in self.edges:
            self.edges[node].extend(edge)
        else:
            self.edges[node] = edge

def get_dfa_char_node_set(self, origin, use_char):
    node_set = set()
    for my_node in origin:
        for edges in self.edges[my_node]:
            if edges.char == use_char:
                node_set.add(edges.dest)

    return self.get_dfa_zero_node_set(node_set)

def get_dfa_zero_node_set(self, origin):
    node_set = set(origin)
    processed = set()
```

```
        while len(node_set.difference(processed)) > 0:
            my_node = node_set.difference(processed).pop()
            for edges in self.edges[my_node]:
                if edges.char == Alphabet.e:
                    node_set.add(edges.dest)
            processed.add(my_node)
        return frozenset(node_set)


class TranslateGraph:
    language = (Alphabet.a, Alphabet.b)

    def __init__(self, nfa_graph: RegexGraphNFA):
        self.node_count = 0
        self.nfa_graph = nfa_graph
        self.trans_to = {}
        self.trans_from = {}
        self.table = {}

    def get_next_node_id(self):
        node_id = self.node_count
        self.node_count += 1
        return node_id

    def add_translate(self, nfa_ids):
        if len(nfa_ids) == 0:
            return None
        if nfa_ids not in self.trans_from:
            dfa_id = self.get_next_node_id()
            self.trans_to[dfa_id] = nfa_ids
            self.trans_from[nfa_ids] = dfa_id
            self.table[dfa_id] = dict(zip(self.language, [None] * len(self.language)))
        return self.trans_from[nfa_ids]

    def translate(self):
        self.create_translate_table()
        return self.build_dfa()

    def build_dfa(self):
        head = 0
        valid_ends = set()
        adjacency = {}
        for node, edges in self.table.items():
            adjacency[node] = []
            if self.nfa_graph.tail in self.trans_to[node]:
                valid_ends.add(node)
```

```
            for my_char, node_dest in edges.items():
                if node_dest is not None:
                    adjacency[node].append(Edge(node_dest, my_char))
        return RegexGraphDFA(head, valid_ends, adjacency)

    def create_translate_table(self):
        nfa_ids = self.nfa_graph.get_dfa_zero_node_set({self.nfa_graph.head})
        self.add_translate(nfa_ids)
        processed = set()

        while len(set(self.table).difference(processed)) > 0:
            my_node = set(self.table).difference(processed).pop()
            for char in self.language:
                next_nodes = self.nfa_graph.get_dfa_char_node_set(self.trans_to[my_node], char)
                dfa_id = self.add_translate(next_nodes)
                self.table[my_node][char] = dfa_id
            processed.add(my_node)


class RegexGraphDFA:
    def __init__(self, head, valid_ends, edges):
        self.edges = edges
        self.head = head
        self.valid_ends = valid_ends
        self.edge_matrix = Matrix.get_from_edges(len(self.edges), self.edges)

    def count_paths(self, length):
        modulo = 1000000007
        if length == 0:
            return 0 if 0 not in self.valid_ends else 1
        edge_walk = self.edge_matrix.pow(length, modulo)
        count = 0
        for end_node in self.valid_ends:
            count += edge_walk.matrix[self.head][end_node]
        return count % modulo


class Matrix:
    @staticmethod
    def get_from_edges(dimension, adj_list):
        my_matrix = Matrix.get_zeros(dimension)
        my_matrix.add_edges(adj_list)
        return my_matrix

    @staticmethod
    def get_zeros(dimension):
```

```
    my_matrix = Matrix(dimension)
    my_matrix.pad_zeros()
    return my_matrix

def copy(self):
    my_matrix = Matrix(self.dimension)
    my_matrix.matrix = []
    for i in range(self.dimension):
        my_matrix.matrix.append([])
        for j in range(self.dimension):
            my_matrix.matrix[i].append(self.matrix[i][j])
    return my_matrix

def __init__(self, dimension):
    self.matrix = None
    self.dimension = dimension

def __str__(self):
    my_str = ''
    for row in self.matrix:
        my_str += str(row) + "\n"
    return my_str

def pad_zeros(self):
    self.matrix = []
    for i in range(self.dimension):
        self.matrix.append([])
        for j in range(self.dimension):
            self.matrix[i].append(0)

def add_edges(self, adj_list):
    if adj_list is not None:
        for from_node, edge_list in adj_list.items():
            for to_node, my_char in edge_list:
                self.matrix[from_node][to_node] = 1

def pow(self, pow_val, mod_val=None):
    started = False
    current_pow = 1
    current_val = 0
    while pow_val > 0:
        if current_pow == 1:
            current_pow_matrix = self.copy()
        else:
            current_pow_matrix = current_pow_matrix.mat_square_mult(current_pow_matrix, mod_val)
        if pow_val % (2 * current_pow):
```

```
                current_val += current_pow
                if started:
                    result = result.mat_square_mult(current_pow_matrix, mod_val)
                else:
                    result = current_pow_matrix.copy()
                    started = True
                pow_val -= current_pow
            current_pow *= 2
        return result

    def mat_square_mult(self, other, mod_val=None):
        result = Matrix.get_zeros(self.dimension)
        for i in range(self.dimension):
            for j in range(self.dimension):
                val = 0
                for k in range(self.dimension):
                    val += self.matrix[i][k] * other.matrix[k][j]
                if mod_val is not None:
                    val %= mod_val
                result.matrix[i][j] = val

        return result


def main():
    cases = int(input().strip())
    for i in range(cases):
        in_line = input().strip().split()
        my_class = CountStrings(in_line[0])
        print(my_class.calculate(int(in_line[1])))


if __name__ == "__main__":
    main()
```

1.33 Rope

```
/*
    rope
    append(x):字符串末尾添加x
    insert(pos,sub):在pos位置插入字符串sub
    erase(pos,x):从pos位置开始删除x个字符
    replace(pos,x):将pos位置的字符替换成x串
    substr(pos,x):提取pos开始的x个字符，返回rope类型
    s.copy(pos,len,t):将从s第6位开始的len个字符插入到t开头
```

```
    at(x)/[x]:访问第x个元素
    翻转操作实现方法:同时维护一正一反两个rope,反转即交换两个子串
    区间循环位移:拆为多个子串拼接即可


*/
#include <bits/stdc++.h>
#include <ext/rope>
using namespace std;
using namespace __gnu_cxx;
int n, m;
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
int main() {
    read(n);
    rope<int> s;
    for (int i = 1; i <= n; i++) s.append(i);
    read(m);
    while (m--) {
        int st, len;
        read(st);
        read(len);
        auto sub = s.substr(st - 1, len);
        s.erase(st - 1, len);
        s.insert(0, sub);
    }
    for (int i = 0; i < n; i++) printf("%d%c", s[i], " \n"[i == n - 1]);
    return 0;
}
```

1.34 可持久化字符串

```
/*
    可持久化字符串
    his[i]=new rope<char>(*his[i-1]) 即可O(1)拷贝历史版本
*/
/*
    要求支持操作
    1.在pos位置插入一串字符
    2.从pos位置开始删除长度为l的字符串
    3.查询第v个历史版本中从pos位置开始的长度为l的字符串
    pos,l,v均需要减去d，d的定义：所有询问中输出的字符‘c’的数量和
*/
#include <bits/stdc++.h>
#include <ext/rope>
using namespace std;
using namespace __gnu_cxx;
const int N = 50010;
rope<char> *his[N];
char ch[210];
int n, d, op, cur, x, y, v;
int main() {
    scanf("%d", &n);
    his[0] = new rope<char>();
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%s", &x, ch);
            x -= d;
            cur++;
            his[cur] = new rope<char>(*his[cur - 1]);
            his[cur]->insert(x, ch);
        } else if (op == 2) {
            scanf("%d%d", &x, &y);
            x -= d;
            y -= d;
            cur++;
            his[cur] = new rope<char>(*his[cur - 1]);
            his[cur]->erase(x - 1, y);
        } else {
            scanf("%d%d%d", &v, &x, &y);
            v -= d;
            x -= d;
            y -= d;
            auto res = his[v]->substr(x - 1, y);
            d += count(res.begin(), res.end(), 'c');
```

```
            cout << res << endl;
        }
    }
    return 0;
}
/*
    要求支持操作
    T 字符串末尾加字符x
    U 撤销最后x次操作(不包含查询操作)
    Q 查询字符串的第x个字母并输出
*/
#include <bits/stdc++.h>
#include <ext/rope>
using namespace std;
using namespace __gnu_cxx;
const int N = 1e5 + 10;
rope<char> *his[N];
int n, d[N];
inline int lowbit(int x) { return x & -x; }
inline void update(int x) {
    while (x <= n) {
        d[x]++;
        x += lowbit(x);
    }
}
inline int get(int x) {
    int res = 0;
    while (x) {
        res += d[x];
        x -= lowbit(x);
    }
    return res;
}
inline char getC() {
    char ch = getchar();
    while (!isalpha(ch)) ch = getchar();
    return ch;
}
inline int getint() {
    int res = 0;
    char ch, ok = 0;
    while (ch = getchar()) {
        if (isdigit(ch)) {
            res *= 10;
            res += ch - '0';
            ok = 1;
```

```
        } else if (ok)
            break;
    }
    return res;
}
// debug
void deb(rope<char> s) {
    for (int i = 0; i < s.length(); i++) cout << s[i];
    puts("");
}
int main() {
    n = getint();
    his[0] = new rope<char>();
    for (int i = 1; i <= n; i++) {
        his[i] = new rope<char>(*his[i - 1]);
        char opt = getC();
        if (opt == 'T') {
            // 字符串末尾加字符x
            char x = getC();
            his[i]->push_back(x);
            update(i);
        } else if (opt == 'U') {
            // 撤销最后x次操作(不包含查询操作)
            update(i);
            int x = getint();
            int l = 1, r = i, mid, now = get(i);
            while (l < r) {
                mid = (l + r) >> 1;
                if (now - get(mid) > x)
                    l = mid + 1;
                else
                    r = mid;
            }
            his[i] = his[l - 1];

        } else if (opt == 'Q') {
            // 查询字符串的第x个字母并输出
            int x = getint() - 1;
            putchar(his[i]->at(x));
            putchar('\n');
        }
    }
    return 0;
}
```

1.35 CLCS

```
/*
    CLCS算法
    求解s串和t串的循环最长公共子序列
*/
#include <algorithm>
#include <cstring>
const int N = 4000 + 10;
int dp[N][N], from[N][N];
int clcs(char s[], char t[]) {
    int n = strlen(s), m = strlen(t);
    auto eq = [&](int a, int b) { return s[(a - 1) % n] == t[(b - 1) % m]; };
    dp[0][0] = from[0][0] = 0;
    for (int i = 0; i <= n * 2; ++i) {
        for (int j = 0; j <= m; ++j) {
            dp[i][j] = 0;
            if (j && dp[i][j - 1] > dp[i][j]) {
                dp[i][j] = dp[i][j - 1];
                from[i][j] = 0;
            }
            if (i && j && eq(i, j) && dp[i - 1][j - 1] + 1 > dp[i][j]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                from[i][j] = 1;
            }
            if (i && dp[i - 1][j] > dp[i][j]) {
                dp[i][j] = dp[i - 1][j];
                from[i][j] = 2;
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < n; ++i) {
        ret = std::max(ret, dp[i + n][m]);
        // re-root
        int x = i + 1, y = 0;
        while (y <= m && from[x][y] == 0) ++y;
        for (; y <= m && x <= n * 2; ++x) {
            from[x][y] = 0, --dp[x][m];
            if (x == n * 2) break;
            for (; y <= m; ++y) {
                if (from[x + 1][y] == 2) break;
                if (y + 1 <= m && from[x + 1][y + 1] == 1) {
                    ++y;
                    break;
                }
```

```
            }
        }
    }
    return ret;
}
```

1.36 字符串技巧

```
/*
    字符串的使用技巧示例：
    对于单串长度未知单总长一定的串
    我们将其保存在一个char数组中并记录每个串的起始位置
    每次保留换行符方便长度计算和输出
    示例为寻找给出字符串集合中是否存在一个串包含所有串
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <string>
using namespace std;
const int N = 200010;
char s[300010];
int T, n, id, len, L[N], nl;
struct SAM {
    int p, q, np, nq, cnt, lst, a[N][26], l[N], f[N];
    int Tr(char c) { return c - 'a'; }
    SAM() {
        cnt = 0;
        lst = ++cnt;
    }
    void Initialize() {
        memset(l, 0, sizeof(int) * (cnt + 1));
        memset(f, 0, sizeof(int) * (cnt + 1));
        for (int i = 0; i <= cnt; i++)
            for (int j = 0; j < 26; j++) a[i][j] = 0;
        cnt = 0;
        lst = ++cnt;
    }
    void extend(int c) {
        p = lst;
        np = lst = ++cnt;
        l[np] = l[p] + 1;
        while (!a[p][c] && p) a[p][c] = np, p = f[p];
        if (!p) {
```

```
            f[np] = 1;
        } else {
            q = a[p][c];
            if (l[p] + 1 == l[q]) {
                f[np] = q;
            } else {
                nq = ++cnt;
                l[nq] = l[p] + 1;
                memcpy(a[nq], a[q], sizeof(a[q]));
                f[nq] = f[q];
                f[np] = f[q] = nq;
                while (a[p][c] == q) a[p][c] = nq, p = f[p];
            }
        }
    }
    void build(int x) {
        int len = strlen(s + L[x]);
        for (int i = 0; i < len; i++) extend(Tr(s[L[x] + i]));
    }
    int Match(int x) {
        int len = strlen(s + L[x]);
        int p = 1;
        for (int i = 0; i < len; i++) {
            int c = Tr(s[L[x] + i]);
            if (!a[p][c]) return 0;
            p = a[p][c];
        }
        return 1;
    }
} sam;
int main() {
    scanf("%d", &T);
    while (T--) {
        int flag = 1;
        sam.Initialize();
        id = len = nl = 0;
        L[1] = nl;
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%s", s + nl);
            int l = strlen(s + nl);
            nl += l + 1;
            L[i + 1] = nl;
            if (l > len) len = l, id = i;
        }
        sam.build(id);
```

```
        for (int i = 1; i <= n; i++) {
            if (!sam.Match(i)) {
                puts("No");
                flag = 0;
                break;
            }
        }
        if (flag) puts(s + L[id]);
    }
    return 0;
}
```

2 数据结构

2.1 离散化

```
/*
    离散化
*/
void Discretization() {
    sort(b + 1, b + n + 1);
    int siz = unique(b + 1, b + n + 1) - (b + 1);
    for (int i = 1; i <= n; i++)
        w[i] = lower_bound(b + 1, b + siz + 1, w[i]) - b;
}
```

2.2 循环链表

```
/*
    循环链表
    示例：在圈上选择m个不相邻的点使得其和最大
*/
const int N = 200010;
namespace Circular_List {
bool del[N];
int n, nxt[N], pre[N], val[N];
typedef pair<int, int> P;
priority_queue<P, vector<P> > Q; // 大根堆
void Initialize() {
    while (Q.size()) Q.pop();
    for (int i = 1; i <= n; i++) pre[i] = i - 1;
    pre[1] = n;
    for (int i = 1; i <= n; i++) nxt[i] = i + 1;
    nxt[n] = 1;
    for (int i = 1; i <= n; i++) Q.push(make_pair(val[i], i)), del[i] = 0;
}
void Del(int x) {
    del[x] = 1;
    nxt[pre[x]] = nxt[x];
    pre[nxt[x]] = pre[x];
    nxt[x] = pre[x] = 0;
}
int Get() {
    while (del[Q.top().second]) Q.pop();
    int res = Q.top().second;
    Q.pop();
```

```
    return res;
}
} // namespace Circular_List
int m;
int main() {
    using namespace Circular_List;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
    if (m > n / 2) {
        puts("Error!");
        return 0;
    }
    Initialize();
    int ans = 0;
    for (int i = 1; i <= m; i++) {
        int x = Get();
        ans += val[x];
        val[x] = val[pre[x]] + val[nxt[x]] - val[x];
        Del(pre[x]);
        Del(nxt[x]);
        Q.push(make_pair(val[x], x));
    }
    printf("%d\n", ans);
    return 0;
}
```

2.3 双向链表

```
/*
    双向链表
*/
namespace Doubly_Linked_List {
bool del[N];
int n, nxt[N], pre[N], val[N];
void Initialize() {
    for (int i = 1; i = n; i++) pre[i] = i - 1;
    pre[1] = 0;
    for (int i = 1; i = n; i++) nxt[i] = i + 1;
    nxt[n] = 0;
    for (int i = 1; i = n; i++) del[i] = 0;
}
void Del(int x) {
    del[x] = 1;
    nxt[pre[x]] = nxt[x];
    pre[nxt[x]] = pre[x];
```

```
    nxt[x] = pre[x] = 0;
}
} // namespace Doubly_Linked_List
```

2.4 笛卡尔树

```
/*
    笛卡尔树
    任意节点值比子树的节点值要小
    中序遍历可得原序列
    构造O(n)
*/
#define CartesianTree_size 500001;
struct Cartesian_Tree {
    int num[CartesianTree_size], size[CartesianTree_size];
    int l[CartesianTree_size], r[CartesianTree_size], fa[CartesianTree_size];
    int st[CartesianTree_size];
    int S, root;
    void get_size(int now) {
        if (l[now]) get_size(l[now]);
        if (r[now]) get_size(r[now]);
        size[now] = size[l[now]] + size[r[now]] + 1;
    }
    inline void build() {
        int i, top = 0;
        bool dec;
        for (i = 1; i <= S; ++i) {
            dec = false;
            while (top && num[i] < num[st[top]]) {
                dec = true;
                --top;
            }
            r[st[top]] = i;
            fa[i] = st[top];
            if (dec) l[i] = st[top + 1], fa[st[top + 1]] = i;
            ++top;
            st[top] = i;
        }
        root = st[1];
        get_size(root);
    }
} tree;
void dfs(int now) {
    int L = tree.l[now], R = tree.r[now];
    if (L) dfs(L);
```

```
    printf("%d", tree.num[now]);
    if (R) dfs(R);
}
int n;
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &tree.num[i]);
    tree.build();
    dfs(tree.root);
    return 0;
}
```

2.5 哈希表

```
/*
    链式Hash
    ASL=1.4
*/
namespace Hash_Table {
const int M = 262143;
struct E {
    int v;
    E* nxt;
} * g[M + 1], pool[N], *cur = pool, *p;
int vis[M + 1];
void ins(int v) {
    int u = v & M;
    if (vis[u] < T) vis[u] = T, g[u] = NULL;
    for (p = g[u]; p; p = p->nxt)
        if (p->v == v) return;
    p = cur++;
    p->v = v;
    p->nxt = g[u];
    g[u] = p;
}
int ask(int v) {
    int u = v & M;
    if (vis[u] < T) return 0;
    for (p = g[u]; p; p = p->nxt)
        if (p->v == v) return 1;
    return 0;
}
void init() { T++, cur = pool; }
} // namespace Hash_Table
```

2.6 单调队列

```
/*
    单调队列
    求[i-k+1,i]中的最小最大值
*/
namespace Monotone_Queue {
int n, k;
int a[N], maxv[N], minv[N];
int deq[N];
void getMax() {
    int l = 0, r = 0;
    for (int i = 1; i <= n; i++) {
        while (l < r && a[deq[r - 1]] <= a[i]) r--;
        dep[r++] = i;
        if (i >= k) {
            maxv[i - k + 1] = a[deq[l]];
            if (deq[l] == i - k + 1) l++;
        }
    }
}
void getMin() {
    int l = 0, r = 0;
    for (int i = 1; i <= n; i++) {
        while (l < r && a[deq[r - 1]] >= a[i]) r--;
        deq[r++] = i;
        if (i >= k) {
            minv[i - k + 1] = a[deq[l]];
            if (deq[l] == i - k + 1) l++;
        }
    }
}
} // namespace Monotone_Queue
```

2.7 单调栈

```
/*
    单调栈
*/
namespace Monotone_Stack {
int n, a[N], l[N], r[N];
LL LargestArea() {
    stack<int> s;
    for (int i = 1; i <= n; i++) {
        while (s.size() && a[s.top()] >= a[i]) s.pop();
```

```
        l[i] = s.size() ? s.top() : 0;
        s.push(i);
    }
    while (s.size()) s.pop();
    for (int i = n; i >= 1; i--) {
        while (s.size() && a[s.top()] >= a[i]) s.pop();
        r[i] = s.size() ? s.top() : n + 1;
        s.push(i);
    }
    LL ans = 0;
    for (int i = 1; i <= n; i++) ans = max(ans, 1LL * a[i] * (r[i] - l[i] - 1));
    return ans;
}
} // namespace Monotone_Stack
```

2.8 二叉堆

```
/*
    二叉堆 (大根堆)
*/
namespace Heap {
int h[N], l;
void down(int x) {
    int ls = x << 1, rs = l | 1, idx = x;
    if (ls <= l && h[ls] > h[idx]) idx = ls;
    if (rs <= l && h[rs] > h[idx]) idx = rs;
    if (idx != x) {
        swap(h[idx], h[x]);
        down(idx);
    }
}
void build() { l = 0; }
/*
    离线O(n)建立二叉堆
*/
void build(int a[], int n) {
    l = n;
    for (int i = 1; i <= n; i++) h[i] = a[i];
    for (int i = n >> 1; i; i--) down(i);
}
void push(int x) {
    h[++l] = x;
    for (int i = l; i > 1 && h[i] > h[i >> 1]; i >>= 1) swap(h[i], h[i >> 1]);
}
int top() { return h[1]; }
```

```
void pop() {
    h[1] = h[l--];
    for (int i = 1;;) {
        int tmp = h[i];
        int j = 0;
        if ((i << 1) <= l && h[i << 1] > tmp) tmp = h[j = i << 1];
        if ((i << 1 | 1) <= l && h[i << 1 | 1] > tmp) j = i << 1 | 1;
        if (j)
            swap(h[i], h[j]), i = j;
        else
            return;
    }
}
} // namespace Heap
```

2.9 可并随机堆

```
/*
   Mergeable_Heap
   可并随机堆+set
*/
#define N 301000
#define inf 0x3f3f3f3f
#define ls son[x][0]
#define rs son[x][1]
#define LS son[y][0]
#define RS son[y][1]
using namespace std;
multiset<int> H;
struct Mergeable_Heap {
    int son[N][2], val[N], flag[N], FLAG;
    int root[N], fa[N];
    int find(int x) {
        if (root[x] == x) return x;
        return root[x] = find(root[x]);
    }
    int stk[N], top;
    void init(int n) {
        val[0] = -inf;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &val[i]);
            H.insert(val[i]);
            root[i] = i;
        }
    }
```

```
void pushdown(int x) {
    val[x] += flag[x];
    if (ls) flag[ls] += flag[x];
    if (rs) flag[rs] += flag[x];
    flag[x] = 0;
}
int merge(int x, int y) {
    pushdown(x), pushdown(y);
    if (val[x] < val[y]) swap(x, y);
    if ((x && y) == 0) return x;
    int d = rand() & 1;
    son[x][d] = merge(son[x][d], y);
    if (son[x][d]) fa[son[x][d]] = x;
    return x;
}
void U() {
    int x, y;
    scanf("%d%d", &x, &y);
    int fx = find(x), fy = find(y);
    if (fx == fy) return;
    root[fx] = root[fy] = merge(fx, fy);
    if (root[fx] == fx)
        H.erase(H.find(val[fy]));
    else
        H.erase(H.find(val[fx]));
}
void A1() {
    int x, y, A, B;
    scanf("%d%d", &x, &A);
    for (stk[top = 1] = x, y = fa[x]; y; y = fa[y]) stk[++top] = y;
    int head = stk[top];
    H.erase(H.find(val[head]));
    while (top) pushdown(stk[top--]);
    val[x] += A;
    y = fa[x];
    int i = (x == son[y][1]);
    son[y][i] = fa[x] = 0;
    A = ls, B = rs;
    fa[A] = fa[B] = 0;
    ls = rs = 0;
    A = merge(A, B);
    if (A) fa[A] = y;
    if (y) son[y][i] = A, A = head;
    int last = head;
    root[A] = root[x] = root[head] = merge(x, A);
    root[0] = 0;
```

```
        pushdown(root[x]);
        H.insert(val[root[x]]);
    }
    void A2() {
        int x, y;
        scanf("%d%d", &x, &y);
        H.erase(H.find(val[find(x)]));
        flag[find(x)] += y;
        pushdown(root[x]);
        H.insert(val[root[x]]);
    }
    void A3() {
        int x;
        scanf("%d", &x);
        FLAG += x;
    }
    void F1() {
        int x, y;
        scanf("%d", &x);
        pushdown(x);
        if (x == root[x]) {
            printf("%d\n", val[x] + FLAG);
            return;
        }
        for (stk[top = 1] = x, y = fa[x]; y; y = fa[y]) stk[++top] = y;
        while (top) pushdown(stk[top--]);
        printf("%d\n", val[x] + FLAG);
    }
    void F2() {
        int x;
        scanf("%d", &x);
        pushdown(find(x));
        printf("%d\n", val[root[x]] + FLAG);
    }
    void F3() {
        multiset<int>::iterator it = H.end();
        printf("%d\n", (*(--it)) + FLAG);
    }
} heap;
int n, m;
char s[5];
int main() {
    srand(252503);
    int i, j, k;
    scanf("%d", &n);
    heap.init(n);
```

```
    for (scanf("%d", &m); m--;) {
        scanf("%s", s);
        // 加一条边，连接第x个节点和第y个节点
        if (s[0] == 'U') heap.U();
        // 将第x个节点的权值增加v
        else if (s[0] == 'A' && s[1] == '1')
            heap.A1();
        // 将第x个节点所在的连通块的所有节点的权值都增加v
        else if (s[0] == 'A' && s[1] == '2')
            heap.A2();
        // 将所有节点的权值增加v
        else if (s[0] == 'A' && s[1] == '3')
            heap.A3();
        // 输出第x个节点当前的权值
        else if (s[0] == 'F' && s[1] == '1')
            heap.F1();
        // 输出第x个节点所在的连通块中，权值最大的节点的权值
        else if (s[0] == 'F' && s[1] == '2')
            heap.F2();
        // 输出所有节点中，权值最大的节点的权值
        else if (s[0] == 'F' && s[1] == '3')
            heap.F3();
    }
    return 0;
}
```

2.10 斜堆

```
/*
    斜堆
    可并堆
*/
struct SHeap {
    int v[N], l[N], r[N];
    int merge(int x, int y) {
        if (x == 0 || y == 0) return x + y;
        if (v[x] < v[y]) swap(x, y);
        r[x] = merge(r[x], y);
        swap(l[x], r[x]);
        return x;
    }
    void pop(int &x) { x = merge(l[x], r[x]); }
    int top(int x) { return v[x]; }
} heap;
```

2.11 左偏树

```
/*
    左偏树
*/
namespace Leftist_Tree {
typedef pair<int, int> P;
struct Node {
    int l, r, d;
    P v;
    Node() {}
    Node(int _l, int _r, int _d, P _v) { l = _l, r = _r, d = _d, v = _v; }
} T[N];
// 并查集判断两个点是否在同一棵左偏树上
int f[N];
int sf(int x) { return x == f[x] ? x : f[x] = sf(f[x]); }
// 返回值为合并后的根节点
int Merge(int a, int b) {
    if (!a) return b;
    if (!b) return a;
    if (T[a].v < T[b].v) swap(a, b);
    T[a].r = Merge(T[a].r, b);
    if (T[T[a].l].d < T[T[a].r].d) swap(T[a].l, T[a].r);
    T[a].d = T[a].r ? T[T[a].r].d + 1 : 0;
    return a;
}
// 返回值为删去了堆顶元素的新左偏树根节点编号
int Pop(int a) {
    int l = T[a].l, r = T[a].r;
    T[a].l = T[a].r = T[a].d = 0;
    return Merge(l, r);
}
// 新建一棵单节点的树，权值为y，下标为x
int Newnode(int x, int y) {
    T[x] = Node(0, 0, 0, y);
    f[x] = x;
}
} // namespace Leftist_Tree
```

2.12 杨氏图表

```
/*
    杨氏图表
    求k条不共享元素的单调不降子序列，使得子序列元素和最大
    对于每个前缀数组输出答案
```

```
    我们对于每个元素x拆分为x个x，则题目转化为
    求k条不共享元素的单调不降子序列，使得子序列长度和最大，
    等价于杨氏图表前k层长度之和
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int K = 5;
int Case, n, i, x;
ll ans;
map<int, ll> T[K];
void ins(int o, int x, ll p) {
    if (o >= K) return;
    T[o][x] += p;
    ans += p;
    while (p) {
        map<int, ll>::iterator it = T[o].lower_bound(x + 1);
        if (it == T[o].end()) return;
        ll t = min(p, it->second);
        ans -= t;
        p -= t;
        ins(o + 1, it->first, t);
        if (t == it->second)
            T[o].erase(it);
        else
            it->second -= t;
    }
}
int main() {
    scanf("%d", &Case);
    while (Case--) {
        scanf("%d", &n);
        ans = 0;
        for (i = 0; i < K; i++) T[i].clear();
        for (i = 1; i <= n; i++) {
            scanf("%d", &x);
            ins(0, x, x);
            printf("%lld%c", ans, " \n"[i==n]);
        }
    }
}
```

2.13 ST 表

```
/*
   ST表
   O(nlogn)预处理
   O(1)查询区间极值
*/
namespace ST {
int f[N][30], lg2[N];
void Init(int n) {
    for (int i = 2; i <= n; i++) lg2[i] = lg2[i / 2] + 1;
    for (int i = 1; i <= n; i++) f[i][0] = a[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
}
int Min(int l, int r) {
    if (l > r) swap(l, r);
    int k = lg2[r - l + 1];
    return min(f[l][k], f[r - (1 << k) + 1][k]);
}
} // namespace ST
/*
    Problem: 求区间最长的连续相同的数字长度
    Solution:
   ST表维护区间不考虑边界限制的最长长度，将左右端连续长度去掉即可直接查询区间内的答案，
    左右端点单独考虑即可。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100010;
int n, m, sumL[N], sumR[N], a[N], d[N][20], lg2[N];
int Max(int l, int r) {
    int x = min(sumR[l], r - l + 1);
    int y = min(sumL[r], r - l + 1);
    l += x;
    r -= y;
    if (r < l) return max(x, y);
    int k = lg2[r - l + 1];
    return max(max(x, y), max(d[l][k], d[r - (1 << k) + 1][k]));
}
int main() {
    while (scanf("%d", &n), n) {
        scanf("%d", &m);
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]), sumL[i] = sumR[i] = 1;
```

```
        for (int i = 2; i <= n; i++)
            if (a[i] == a[i - 1]) sumL[i] += sumL[i - 1];
        for (int i = n - 1; i; i--)
            if (a[i] == a[i + 1]) sumR[i] += sumR[i + 1];
        for (int i = 2; i <= n; i++) lg2[i] = lg2[i / 2] + 1;
        for (int i = 1; i <= n; i++) d[i][0] = sumL[i] + sumR[i] - 1;
        ;
        for (int j = 1; (1 << j) <= n; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++)
                d[i][j] = max(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
        int x, y;
        while (m--) {
            scanf("%d%d", &x, &y);
            printf("%d\n", Max(x, y));
        }
    }
    return 0;
}
```

2.14 树状数组 (点数据)

```
/*
    树状数组
    查询[1,n]第k大
    查询复杂度O(logn)
*/
namespace BIT {
int n, c[N];
void Initialize(int _n) {
    n = _n;
    memset(c, 0, (n + 1) * sizeof(int));
}
void add(int x, int val) {
    while (x <= n) c[x] += val, x += x & -x;
}
void query(int x) {
    int res = 0;
    while (x) res += c[x], x -= x & -x;
    return res;
}
int findkth(int k) {
    int m = 31 - _builtin_clz(n);
    int cnt = 0, res = 0;
    for (int i = m; i >= 0; i--) {
        res += 1 << i;
```

```
        if (res >= n || cnt + c[res] >= k)
            res -= 1 << i;
        else
            cnt += c[res];
    }
    return res + 1;
}
} // namespace BIT
/*
    1: 前缀最大值
    2: 后缀最大值
    更新操作均为add(x,val[x])
*/
void add1(int x, int val) {
    while (x <= n) c1[x] = max(val, c1[x]), x += x & -x;
}
int query1(int x) {
    int res = 0;
    while (x) res = max(res, c1[x]), x -= x & -x;
    return res;
}
void add2(int x, int val) {
    while (x) c2[x] = max(c2[x], val), x -= x & -x;
}
int query2(int x) {
    int res = 0;
    while (x <= n) res = max(res, c2[x]), x += x & -x;
    return res;
}
```

2.15 树状数组 (区间可加类数据)

```
/*
    BIT
    区间修改区间查询
    仅限可加类数据
*/
struct BIT {
    int n, s[N], a[N];
    ll b[N];
    void init(int x) {
        n = x;
        for (int i = 1; i <= n; i++) a[i] = b[i] = s[i] = 0;
    }
    void modify(int x, int p) {
```

```
        for (int i = x; i <= n; i += i & -i) a[i] += p, b[i] += p * s[x - 1];
    }
    ll ask(int x) {
        int t0 = 0;
        ll t1 = 0;
        for (int i = x; i; i -= i & -i) t0 += a[i], t1 += b[i];
        return 1LL * s[x] * t0 - t1;
    }
} T;
int n, m, op, x, y;
int main() {
    scanf("%d", &n);
    init(n);
    for (int i = 1; i <= n; i++) scanf("%d", &T.s[i]), T.s[i] += T.s[i - 1];
    scanf("%d", &m);
    while (m--) {
        scanf("%d%d%d", &op, &x, &y);
        if (op == 1)
            T.modify(x, 1), T.modify(y + 1, -1);
        else
            printf("%lld\n", T.ask(y) - T.ask(x - 1));
    }
    return 0;
}
```

2.16 二维树状数组

```
/*
    二维树状数组
*/
namespace Td_BIT {
int c[M][M];
void Initialize() { memset(c, 0, sizeof(c)); }
void add(int x, int y, int val) {
    for (int i = x; i < M; i += i & -i)
        for (int j = y; j < M; j += j & -j) c[i][j] += val;
}
int query(int x, int y) {
    int res = 0;
    for (int i = x; i; i -= i & -i)
        for (int j = y; j; j -= j & -j) res += c[i][j];
    return res;
}
} // namespace Td_BIT
```

2.17 线段树

```
/*
    线段树
    区间修改区间求和
*/
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b;
    LL tag, val;
} T[M];
void build(int, int);
void Initialize(int n) {
    tot = 0;
    build(1, n);
}
void addtag(int x, int tag) {
    T[x].tag += tag;
    T[x].val += ((LL)T[x].b - T[x].a + 1) * tag;
}
void pb(int x) {
    if (T[x].l) {
        addtag(T[x].l, T[x].tag);
        addtag(T[x].r, T[x].tag);
    }
    T[x].tag = 0;
}
void up(int x) { T[x].val = T[T[x].l].val + T[T[x].r].val; }
void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].tag = T[x].l = T[x].r = T[x].val = 0;
    if (l == r) {
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
    up(x);
}
void change(int x, int a, int b, int p) {
    if (T[x].a >= a && T[x].b <= b) {
```

```
        addtag(x, p);
        return;
    }
    if (T[x].tag) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    if (mid >= a && T[x].l) change(T[x].l, a, b, p);
    if (mid < b && T[x].r) change(T[x].r, a, b, p);
    up(x);
}
LL query(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].val;
    if (T[x].tag) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    LL res = 0;
    if (mid >= a && T[x].l) res += query(T[x].l, a, b);
    if (mid < b && T[x].r) res += query(T[x].r, a, b);
    return res;
}
} // namespace Segment_Tree
```

2.18 线段树除法

```
/*
    线段树-区间除法
    线段树维护区间极值，支持区间加法和区间整除
    当区间的最大值和最小值在除法之后得到的数等价于原数字减去相同数的时候，
    区间除法转化为区间减法，否则向下划分区间直到满足条件，以此方法解决区间除法问题
*/
using namespace std;
const int INF = 1e9;
const int M = 1000000;
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b;
    int tag, mx, mn;
} T[M];
void build(int, int);
void Initialize(int n) {
    tot = 0;
    build(1, n);
}
void addtag(int x, int tag) {
    T[x].tag += tag;
    T[x].mx += tag;
```

```
    T[x].mn += tag;
}
void pb(int x) {
    if (T[x].l) {
        addtag(T[x].l, T[x].tag);
        addtag(T[x].r, T[x].tag);
    }
    T[x].tag = 0;
}
void up(int x) {
    T[x].mx = max(T[T[x].l].mx, T[T[x].r].mx);
    T[x].mn = min(T[T[x].l].mn, T[T[x].r].mn);
}
void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].tag = T[x].l = T[x].r = T[x].mx = T[x].mn = 0;
    if (l == r) {
        scanf("%d", &T[x].mx);
        T[x].mn = T[x].mx;
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
    up(x);
}
void change(int x, int a, int b, int p) {
    if (T[x].a >= a && T[x].b <= b) {
        addtag(x, p);
        return;
    }
    if (T[x].tag) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    if (mid >= a && T[x].l) change(T[x].l, a, b, p);
    if (mid < b && T[x].r) change(T[x].r, a, b, p);
    up(x);
}
int querymx(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].mx;
    if (T[x].tag) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    int res = -INF;
```

```
    if (mid >= a && T[x].l) res = max(res, querymx(T[x].l, a, b));
    if (mid < b && T[x].r) res = max(res, querymx(T[x].r, a, b));
    ;
    return res;
}
void Divide(int x, int a, int b, int p) {
    if (T[x].a >= a && T[x].b <= b) {
        if (T[x].mn - T[x].mn / p == T[x].mx - T[x].mx / p) {
            addtag(x, T[x].mx / p - T[x].mx);
            return;
        }
    }
    if (T[x].tag) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    if (mid >= a && T[x].l) Divide(T[x].l, a, b, p);
    if (mid < b && T[x].r) Divide(T[x].r, a, b, p);
    up(x);
}
} // namespace Segment_Tree
int n, q, t, l, r, x;
int main() {
    scanf("%d%d", &n, &q);
    Segment_Tree::Initialize(n);
    while (q--) {
        scanf("%d%d%d%d", &t, &l, &r, &x);
        l++;
        r++;
        if (t == 0)
            Segment_Tree::change(1, l, r, x);
        else if (t == 1)
            Segment_Tree::Divide(1, l, r, x);
        else
            printf("%d\n", Segment_Tree::querymx(1, l, r));
    }
    return 0;
}
```

2.19 线段树求幂

```
/*
    线段树区间求幂
    操作
        1. 查询区间乘积
        2. 区间每个数变为原来的k次
        3. 区间每个数乘k
```

```
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int mod = 1e9 + 7;
const int M = 400010;
void read(int &u) {
    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();
    }
    u = x * f;
}
int tot;
struct node {
    int l, r, val, a, b, ta, tb, L;
} T[M];
void build(int, int);
inline void Initialize(int n) {
    tot = 0;
    build(1, n);
}
inline int pow(int a, int b, int mod) {
    int res = 1;
    while (b) {
        if (b & 1) res = 1LL * res * a % mod;
        b >>= 1;
        a = 1LL * a * a % mod;
    }
    return res;
}
inline void addtag(int x, int a, int b) {
    T[x].val = 1LL * pow(T[x].val, b, mod) * pow(a, T[x].L, mod) % mod;
    T[x].ta = 1LL * pow(T[x].ta, b, mod) * a % mod;
    T[x].tb = 1LL * T[x].tb * b % (mod - 1);
}
inline void pb(int x) {
    if (T[x].l) {
        addtag(T[x].l, T[x].ta, T[x].tb);
        addtag(T[x].r, T[x].ta, T[x].tb);
```

```
    }
    T[x].ta = T[x].tb = 1;
}
inline void up(int x) {
    T[x].val = (1LL * T[T[x].l].val * T[T[x].r].val) % mod;
}
inline void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].ta = T[x].tb = 1;
    T[x].l = T[x].r = 0;
    T[x].L = T[x].b - T[x].a + 1;
    if (l == r) {
        scanf("%d", &T[x].val);
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
    up(x);
}
inline void change(int x, int a, int b, int m, int k) {
    if (T[x].a >= a && T[x].b <= b) {
        addtag(x, m, k);
        return;
    }
    if (T[x].ta != 1 || T[x].tb != 1) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    if (mid >= a && T[x].l) change(T[x].l, a, b, m, k);
    if (mid < b && T[x].r) change(T[x].r, a, b, m, k);
    up(x);
}
inline int query(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].val;
    if (T[x].ta != 1 || T[x].tb != 1) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    int res = 1;
    if (mid >= a && T[x].l) res = 1LL * res * query(T[x].l, a, b) % mod;
    if (mid < b && T[x].r) res = 1LL * res * query(T[x].r, a, b) % mod;
    return res;
}
int cas, n, q, op, x, y, z;
int main() {
```

```
    read(cas);
    while (cas--) {
        read(n);
        read(q);
        Initialize(n);
        while (q--) {
            read(op);
            if (op == 1) {
                read(x);
                read(y);
                read(z);
                change(1, x, y, z, 1);
            } else if (op == 2) {
                read(x);
                read(y);
                read(z);
                change(1, x, y, 1, z);
            } else {
                read(x);
                read(y);
                printf("%d\n", query(1, x, y));
            }
        }
    }
    return 0;
}
/*
    指标法+线段树
    操作:
        1. 查询区间乘积
        2. 区间每个数变为原来的k次, k<=1e9
        3. 区间每个数乘k, k<=1e3
    所有初始数据小于等于1e3
    题解: 我们对1-1000预处理指标I(x),
    有I(x*y%P)=(I(x)+I(y))%(P-1), I((x^k)%P)=k*I(x)%(P-1)
    所以只要用线段树维护乘法和加法即可,
    求指标的时候, BSGS的步长=询问*P/(步长), 所以我们求得步长为1000000为佳,
    考虑常数情况, 调整至300000。
*/
#include <algorithm>
#include <cstdio>
#include <tr1/unordered_map>
using namespace std;
using namespace std::tr1;
const int P = 1000000007, Q = P - 1;
const int M = 400010;
```

```
void read(int &u) {
    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();
    }
    u = x * f;
}
inline int pow(int a, int b, int mod) {
    int res = 1;
    while (b) {
        if (b & 1) res = 1LL * res * a % mod;
        b >>= 1;
        a = 1LL * a * a % mod;
    }
    return res;
}
namespace NT {
unordered_map<int, int> T;
typedef pair<int, int> PI;
const int K = 300000, G = 5; // 1e9+7的原根为5
int ind[1010], base, i;
inline int ask(int x) {
    if (x == 1) return 0;
    int t = pow(x, P - 2, P);
    for (int i = K;; i += K) {
        t = 1LL * t * base % P;
        unordered_map<int, int>::iterator it = T.find(t);
        if (it != T.end()) return i - it->second;
    }
}
void init() {
    for (base = 1, i = 0; i < K; i++) {
        T.insert(PI(base, i));
        base = 1LL * base * G % P;
    }
    for (i = 1; i <= 1000; i++) ind[i] = ask(i);
}
} // namespace NT
int tot;
struct node {
```

```
    int l, r, val, a, b, ta, tb, L;
} T[M];
void build(int, int);
inline void Initialize(int n) {
    tot = 0;
    build(1, n);
}
inline void addtag(int x, int a, int b) {
    T[x].val = (1LL * T[x].val * b + 1LL * a * T[x].L) % Q;
    T[x].ta = (1LL * T[x].ta * b + a) % Q;
    T[x].tb = 1LL * T[x].tb * b % Q;
}
inline void pb(int x) {
    if (T[x].l) {
        addtag(T[x].l, T[x].ta, T[x].tb);
        addtag(T[x].r, T[x].ta, T[x].tb);
    }
    T[x].ta = 0;
    T[x].tb = 1;
}
inline void up(int x) { T[x].val = (T[T[x].l].val + T[T[x].r].val) % Q; }
inline void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].ta = 0;
    T[x].tb = 1;
    T[x].l = T[x].r = 0;
    T[x].L = T[x].b - T[x].a + 1;
    if (l == r) {
        read(T[x].val);
        T[x].val = NT::ind[T[x].val];
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
    up(x);
}
inline void change(int x, int a, int b, int m, int k) {
    if (T[x].a >= a && T[x].b <= b) {
        addtag(x, m, k);
        return;
    }
```

```
    if (T[x].ta != 0 || T[x].tb != 1) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    if (mid >= a && T[x].l) change(T[x].l, a, b, m, k);
    if (mid < b && T[x].r) change(T[x].r, a, b, m, k);
    up(x);
}
inline int query(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].val;
    if (T[x].ta != 0 || T[x].tb != 1) pb(x);
    int mid = (T[x].a + T[x].b) >> 1;
    int res = 0;
    if (mid >= a && T[x].l) res = (res + query(T[x].l, a, b)) % Q;
    if (mid < b && T[x].r) res = (res + query(T[x].r, a, b)) % Q;
    return res;
}
int cas, n, q, op, x, y, z;
int main() {
    NT::init();
    read(cas);
    while (cas--) {
        read(n);
        read(q);
        Initialize(n);
        while (q--) {
            read(op);
            if (op == 1) {
                read(x);
                read(y);
                read(z);
                change(1, x, y, NT::ind[z], 1);
            } else if (op == 2) {
                read(x);
                read(y);
                read(z);
                change(1, x, y, 0, z);
            } else {
                read(x);
                read(y);
                printf("%d\n", pow(NT::G, query(1, x, y), P));
            }
        }
    }
    return 0;
}
```

2.20 线段树合并

```
/*
    线段树合并
*/
int merge(int x, int y, int l, int r) {
    if (!x) return y;
    if (!y) return x;
    int z = ++tot;
    if (l == r) {
        T[z].v = T[x].v + T[y].v;
        return z;
    }
    int mid = (l + r) >> 1;
    T[z].l = merge(T[x].l, T[y].l, l, mid);
    T[z].r = merge(T[x].r, T[y].r, mid + 1, r);
    T[z].v = T[T[z].l].v + T[T[z].r].v;
    return z;
}
/*
    Example
        给出一棵树，对于每个节点，求其子树中比父节点大的点个数
    Solution
        我们考虑每个权值建立一棵线段树，边dfs边将子节点合并为一颗线段树，
        那么只要查询当前点的树上后缀和即可。
*/
const int N = 100010, M = N * 20;
int n, a[N], ans[N], root[N], disc[N];
vector<int> v[N];
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b, sum;
} T[M];
void up(int x) { T[x].sum = T[T[x].l].sum + T[T[x].r].sum; }
int build(int l, int r, int p) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].sum = 0;
    if (l == r) {
        T[x].sum = 1;
        return x;
    }
    int mid = (l + r) >> 1;
    if (p <= mid) {
```

```
        T[x].l = build(l, mid, p);
    } else {
        T[x].r = build(mid + 1, r, p);
    }
    return up(x), x;
}
int ask(int x, int l, int r) {
    if (!x) return 0;
    if (l <= T[x].a && T[x].b <= r) return T[x].sum;
    int mid = (T[x].a + T[x].b) >> 1, res = 0;
    if (l <= mid) res += ask(T[x].l, l, r);
    if (r > mid) res += ask(T[x].r, l, r);
    return res;
}
int merge(int x, int y) {
    if (!x || !y) return x ^ y;
    T[x].l = merge(T[x].l, T[y].l);
    T[x].r = merge(T[x].r, T[y].r);
    return up(x), x;
}
void dfs(int x, int fx) {
    int res = 0;
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i];
        if (y == fx) continue;
        dfs(y, x);
        res += ask(root[y], a[x] + 1, n);
        root[x] = merge(root[x], root[y]);
    }
    ans[x] = res;
}
} // namespace Segment_Tree
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]), disc[i] = a[i];
    sort(disc + 1, disc + n + 1);
    int m = unique(disc + 1, disc + n + 1) - disc - 1;
    for (int i = 1; i <= n; i++)
        a[i] = lower_bound(disc + 1, disc + m + 1, a[i]) - disc;
    for (int i = 2; i <= n; i++) {
        int x;
        scanf("%d", &x);
        v[x].push_back(i);
        v[i].push_back(x);
    }
    for (int i = 1; i <= n; i++) root[i] = Segment_Tree::build(1, n, a[i]);
```

```
    Segment_Tree::dfs(1, 1);
    for (int i = 1; i <= n; i++) printf("%d\n", ans[i]);
    return 0;
}
/*
    Example
        给出一棵二叉树，每个叶节点上有一个权值，现在可以任意交换左右儿子，
        使得逆序对最少，求最少的逆序对数量
    Solution
        我们发现对于每个非叶节点来说，其贡献值为左右两个儿子的权值树上，
        每个节点想反位置的数量和乘积，
        比如左儿子的权值树左节点和右儿子权值树的右节点相乘，
        那么我们对于每个节点建立一颗权值线段树，仅保留非0链，
        递归合并这些权值线段树，同时每次将相反位置数量乘积的最小值累加到答案即可
*/
const int N = 400010, M = N * 20;
typedef long long LL;
int n, a[N], cnt, Root, root[N];
int Tree[N][2];
LL Ans, Ans0, Ans1;
void Read_Tree(int &x) {
    x = ++cnt;
    scanf("%d", &a[x]);
    if (a[x]) return;
    Read_Tree(Tree[x][0]);
    Read_Tree(Tree[x][1]);
}
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b, sum;
} T[M];
void up(int x) { T[x].sum = T[T[x].l].sum + T[T[x].r].sum; }
int build(int l, int r, int p) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].sum = 0;
    if (l == r) {
        T[x].sum = 1;
        return x;
    }
    int mid = (l + r) >> 1;
    if (p <= mid) {
        T[x].l = build(l, mid, p);
    } else {
```

```
        T[x].r = build(mid + 1, r, p);
    }
    return up(x), x;
}
int merge(int x, int y) {
    if (!x || !y) return x ^ y;
    Ans0 += (LL)T[T[x].r].sum * (LL)T[T[y].l].sum;
    Ans1 += (LL)T[T[x].l].sum * (LL)T[T[y].r].sum;
    T[x].l = merge(T[x].l, T[y].l);
    T[x].r = merge(T[x].r, T[y].r);
    return up(x), x;
}
void dfs(int x) {
    if (a[x]) return;
    dfs(Tree[x][0]);
    dfs(Tree[x][1]);
    Ans0 = Ans1 = 0;
    root[x] = merge(root[Tree[x][0]], root[Tree[x][1]]);
    Ans += min(Ans0, Ans1);
}
} // namespace Segment_Tree
int main() {
    scanf("%d", &n);
    Read_Tree(Root);
    for (int i = 1; i <= cnt; i++)
        if (a[i] != 0) root[i] = Segment_Tree::build(1, n, a[i]);
    Segment_Tree::dfs(Root);
    printf("%lld\n", Ans);
    return 0;
}
```

2.21 线段树维护矩阵

```
/*
    线段树+矩阵变换
    Example
    1.Move l r x y 让编号在[l,r]内的所有石子堆向东移动x个单位，向北移动y个单位。
    2.To l r x y 让编号在[l,r]内的所有石子堆移动至(x,y) 。
    3.Rotate l r 让编号在[l,r]内的所有石子堆按原点逆时针旋转90°。
    4.Ask k 询问第k个石子堆的坐标。
*/
#define rep(i, n) for (int i = 0; i < n; i++)
const int N = 3;
struct mat {
    int a[N][N];
```

```
    mat() { rep(i, N) rep(j, N) a[i][j] = 0; }
};
mat loc(int x, int y) {
    mat c;
    c.a[0][0] = x;
    c.a[1][0] = y;
    c.a[2][0] = 1;
    return c;
}
mat rotate() {
    mat c;
    c.a[0][0] = 0;
    c.a[0][1] = -1;
    c.a[1][0] = 1;
    c.a[1][1] = 0;
    c.a[2][2] = 1;
    return c;
}
mat move(int x, int y) {
    mat c;
    c.a[0][0] = c.a[1][1] = c.a[2][2] = 1;
    c.a[0][2] = x;
    c.a[1][2] = y;
    return c;
}
mat moveto(int x, int y) {
    mat c;
    c.a[0][2] = x;
    c.a[1][2] = y;
    c.a[2][2] = 1;
    return c;
}
mat one() {
    mat c;
    c.a[0][0] = c.a[1][1] = c.a[2][2] = 1;
    return c;
}
mat mul(mat a, mat b) {
    mat c;
    rep(i, N) rep(j, N) rep(k, N) c.a[i][j] += a.a[i][k] * b.a[k][j];
    return c;
}
const int M = 100010;
int l[M], r[M], tot;
mat val[M], tmp;
bool tag[M];
```

```
int n, i, j, q;
int X[50010], Y[50010], XX, YY;
char op[10];
void build(int a, int b) {
    int x = ++tot;
    if (a == b) {
        val[x] = loc(X[a], Y[a]);
        return;
    }
    int mid = (a + b) >> 1;
    val[x] = one();
    l[x] = tot + 1;
    build(a, mid);
    r[x] = tot + 1;
    build(mid + 1, b);
}
void addtag(int x, mat p) { val[x] = mul(p, val[x]), tag[x] = 1; }
void pb(int x) {
    if (tag[x]) {
        if (l[x]) addtag(l[x], val[x]);
        if (r[x]) addtag(r[x], val[x]);
        tag[x] = 0;
        val[x] = one();
    }
}
void change(int x, int a, int b, int c, int d, mat p) {
    if (c <= a && b <= d) {
        addtag(x, p);
        return;
    }
    pb(x);
    int mid = (a + b) >> 1;
    if (c <= mid) change(l[x], a, mid, c, d, p);
    if (d > mid) change(r[x], mid + 1, b, c, d, p);
}
mat ask(int x, int a, int b, int c) {
    if (a == b) return val[x];
    int mid = (a + b) >> 1;
    if (c <= mid) return mul(val[x], ask(l[x], a, mid, c));
    return mul(val[x], ask(r[x], mid + 1, b, c));
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; i++) scanf("%d%d", X + i, Y + i);
        tot = 0;
        build(1, n);
```

```
        scanf("%d", &q);
        while (q--) {
            scanf("%s", op);
            if (op[0] == 'M') {
                scanf("%d%d%d%d", &i, &j, &XX, &YY);
                change(1, 1, n, i, j, move(XX, YY));
            }
            if (op[0] == 'T') {
                scanf("%d%d%d%d", &i, &j, &XX, &YY);
                change(1, 1, n, i, j, moveto(XX, YY));
            } else if (op[0] == 'R') {
                scanf("%d%d", &i, &j);
                change(1, 1, n, i, j, rotate());
            } else if (op[0] == 'A') {
                scanf("%d", &i);
                tmp = ask(1, 1, n, i);
                printf("%d %d\n", tmp.a[0][0], tmp.a[1][0]);
            }
        }
    }
    return 0;
}
```

2.22 线段树标记持久化

```
/*
    线段树-标记持久化
    题目大意：给出一棵树，要求维护两个操作：
    1.给一个树的节点加上x，他的子节点加上x-k，他的孙子节点加上x-2k，按层递推
    2.查询一个点的当前权值
    题解：对于一个节点来说，一次修改对节点u的影响为x+k*(d[changed]-d[u])
    x+k*d[changed]是一个子树修改值，通过维护dfs序可以转化为区间修改值，
    对于-k*d[u]来说，是一个查询时可以单独计算的值，
    我们维护x+k*d[changed]的区间加法，和k的区间加法，
    然后单点查询即可，因为只有区间可加操作和单点查询，因此我们可以对标记进行持久化
*/
const int N = 3000010;
const int mod = 1e9 + 7;
int T[N], K[N], st[N], en[N], dfn, d[N];
vector<int> v[N];
void dfs(int x, int fx) {
    st[x] = ++dfn;
    d[x] = d[fx] + 1;
    for (int i = 0; i < (int)v[x].size(); i++) dfs(v[x][i], x);
    en[x] = dfn;
```

```
}
void modify(int x, int l, int r, int L, int R, int root, int p, int k) {
    if (L <= l && r <= R) {
        K[x] = (K[x] + k) % mod;
        T[x] = ((T[x] + p) % mod + 1LL * d[root] * k % mod) % mod;
        return;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) modify(x << 1, l, mid, L, R, root, p, k);
    if (mid < R) modify(x << 1 | 1, mid + 1, r, L, R, root, p, k);
}
int query(int x, int l, int r, int pos, int p) {
    if (l == r) return (T[x] - 1LL * K[x] * d[p] % mod + mod) % mod;
    int mid = (l + r) >> 1;
    if (pos <= mid)
        return ((query(x << 1, l, mid, pos, p) + T[x]) % mod -
                1LL * K[x] * d[p] % mod + mod) %
               mod;
    else
        return ((query(x << 1 | 1, mid + 1, r, pos, p) + T[x]) % mod -
                1LL * K[x] * d[p] % mod + mod) %
               mod;
}
int cas, n, q, op, u, x, k;
int main() {
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) v[i].clear();
        memset(d, 0, sizeof(d));
        memset(T, 0, sizeof(T));
        memset(K, 0, sizeof(K));
        for (int i = 2; i <= n; i++) {
            scanf("%d", &x);
            v[x].push_back(i);
        }
        dfn = 0;
        dfs(1, 0);
        scanf("%d", &q);
        while (q--) {
            scanf("%d", &op);
            if (op == 1) {
                scanf("%d%d%d", &u, &x, &k);
                modify(1, 1, n, st[u], en[u], u, x, k);
            } else {
                scanf("%d", &u);
```

```
            printf("%d\n", query(1, 1, n, st[u], u));
        }
    }
}
return 0;
}
```

2.23 ZKW 线段树

```
/*
    ZKW线段树
    T:区间和 C:区间数字个数
*/
namespace ZKW_Segment_Tree {
int M, T[MAX_M], C[MAX_M], t, c;
void Initialize(int n) {
    for (M = 1; M < (n + 2); M <<= 1)
        ;
    memset(T, 0, sizeof(T));
    memset(C, 0, sizeof(C));
}
void Add(int x, int y) {
    T[x += M] += y;
    C[x]++;
    for (x /= 2; x; x /= 2) {
        T[x] = T[x << 1] + T[(x << 1) ^ 1];
        C[x] = C[x << 1] + C[(x << 1) ^ 1];
    }
}
void Cal(int x, int y) {
    t = c = 0;
    x += M - 1;
    y += M + 1;
    while (x ^ y ^ 1 > 0) {
        if (~x & 1) t += T[x + 1], c += C[x + 1];
        if (y & 1) t += T[y - 1], c += C[y - 1];
        x >>= 1;
        y >>= 1;
    }
}
} // namespace ZKW_Segment_Tree
```

2.24 TD 树

```
/*
    TD树
    划分为四个平面维护
    Example: 给出一个颜色方格图，每个格子都有一个颜色，
    现在往一些矩形内的方格涂新的颜色，当一个格子涂色和原来不一样时就不再是一个纯色格子，
    问有多少个格子不再是纯色格子
*/
#include <bits/stdc++.h>
#define rep(i, n) for (int i = 1; i <= n; i++)
typedef long long ll;
using namespace std;
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
const int N = 1000010;
int tot = 0, T[N << 2][4], col[N << 2];
int a[N];
int *mp[N];
void build(int &x, int l1, int r1, int l2, int r2) {
    if (l1 > r1 || l2 > r2) return; // Notice
    x = ++tot;
    if (l1 == r1 && l2 == r2) {
        col[x] = mp[l1 - 1][l2 - 1];
        return;
    }
    int mid1 = (l1 + r1) >> 1, mid2 = (l2 + r2) >> 1;
    build(T[x][0], l1, mid1, l2, mid2);
    build(T[x][1], l1, mid1, mid2 + 1, r2);
    build(T[x][2], mid1 + 1, r1, l2, mid2);
    build(T[x][3], mid1 + 1, r1, mid2 + 1, r2);
}
void pb(int x) {
```

```
    if (!col[x]) return;
    for (int i = 0; i < 4; i++)
        if (T[x][i]) {
            if (!col[T[x][i]] || col[T[x][i]] == col[x])
                col[T[x][i]] = col[x];
            else
                col[T[x][i]] = -1;
        }
    col[x] = 0;
}
int cx1, cx2, cy1, cy2, cc;
void update(int x, int l1, int r1, int l2, int r2) {
    if (cx1 <= l1 && r1 <= cx2 && cy1 <= l2 && r2 <= cy2) {
        if (!col[x] || col[x] == cc)
            col[x] = cc;
        else
            col[x] = -1;
        return;
    }
    if (col[x] < 0) return;
    pb(x);
    int mid1 = (l1 + r1) >> 1, mid2 = (l2 + r2) >> 1;
    if (cx1 <= mid1 && cy1 <= mid2) update(T[x][0], l1, mid1, l2, mid2);
    if (cx1 <= mid1 && cy2 > mid2) update(T[x][1], l1, mid1, mid2 + 1, r2);
    if (cx2 > mid1 && cy1 <= mid2) update(T[x][2], mid1 + 1, r1, l2, mid2);
    if (cx2 > mid1 && cy2 > mid2) update(T[x][3], mid1 + 1, r1, mid2 + 1, r2);
}
int qx, qy;
int query(int x, int l1, int r1, int l2, int r2) {
    if (l1 == r1 && l2 == r2 || col[x] < 0) return col[x];
    pb(x);
    int mid1 = (l1 + r1) >> 1, mid2 = (l2 + r2) >> 1;
    if (qx <= mid1 && qy <= mid2) return query(T[x][0], l1, mid1, l2, mid2);
    if (qx <= mid1 && qy > mid2) return query(T[x][1], l1, mid1, mid2 + 1, r2);
    if (qx > mid1 && qy <= mid2) return query(T[x][2], mid1 + 1, r1, l2, mid2);
    return query(T[x][3], mid1 + 1, r1, mid2 + 1, r2);
}
int main() {
    int n, m, k, root, lft = 0;
    read(n);
    read(m);
    read(k);
    for (int i = 0; i < n * m; i++) read(a[i]);
    for (int i = 0; i < n; i++) mp[i] = a + i * m;
    build(root, 1, n, 1, m);
    for (int i = 1; i <= k; i++) {
```

```
        read(cx1);
        read(cy1);
        read(cx2);
        read(cy2);
        read(cc);
        update(root, 1, n, 1, m);
    }
    for (qx = 1; qx <= n; qx++) {
        for (qy = 1; qy <= m; qy++) lft += query(root, 1, n, 1, m) >= 0;
    }
    printf("%d\n", n * m - lft);
    return 0;
}
```

2.25 LC 线段树

```
/*
    LC线段树
    支持插入一条线段，查询横坐标为某个值时最上面的线段。
    插入 O(logn*logn)，查询O(logn)
*/
struct Seg {
    double k, b;
    Seg() {}
    Seg(int x0, int y0, int x1, int y1) {
        if (x0 == x1)
            k = 0, b = max(y0, y1);
        else
            k = 1.0 * (y0 - y1) / (x0 - x1), b = -k * x0 + y0;
    }
    double gety(int x) { return k * x + b; }
} s[100010];
int m, op, cnt, X0, Y0, X1, Y1, ans, v[131000];
inline int sig(double x) { return fabs(x) < 1e-8 ? 0 : (x > 0 ? 1 : -1); }
void ins(int x, int a, int b, int c, int d, int p) {
    if (c <= a && b <= d) {
        if (sig(s[p].gety(a) - s[v[x]].gety(a)) > 0 &&
            sig(s[p].gety(b) - s[v[x]].gety(b)) > 0) {
            v[x] = p;
            return;
        }
        if (sig(s[p].gety(a) - s[v[x]].gety(a)) <= 0 &&
            sig(s[p].gety(b) - s[v[x]].gety(b)) <= 0)
            return;
        if (a == b) return;
```

```
    }
    int mid = (a + b) >> 1;
    if (c <= mid) ins(x << 1, a, mid, c, d, p);
    if (d > mid) ins(x << 1 | 1, mid + 1, b, c, d, p);
}
void ask(int x, int a, int b, int c) {
    if (sig(s[ans].gety(c) - s[v[x]].gety(c)) < 0)
        ans = v[x];
    else if (!sig(s[ans].gety(c) - s[v[x]].gety(c)) && ans > v[x])
        ans = v[x];
    if (a == b) return;
    int mid = (a + b) >> 1;
    c <= mid ? ask(x << 1, a, mid, c) : ask(x << 1 | 1, mid + 1, b, c);
}
void read(int& a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
int main() {
    s[0].b = -1;
    read(m);
    while (m--) {
        read(op);
        if (!op) {
            read(X0);
            ans = 0, ask(1, 1, N, X0);
            printf("%d\n", ans);
        } else {
            read(X0), read(Y0), read(X1), read(Y1);
            s[++cnt] = Seg(X0, Y0, X1, Y1);
            ins(1, 1, N, X0, X1, cnt);
        }
    }
    return 0;
}
```

2.26 主席树 (点修改)

```
/*
    可持久化线段树
    每次从root[0]版本开始更新即可
*/
```

```
namespace Persistent_Segment_Tree {
int l[N * 40], r[N * 40], v[N * 40], tot, root[S];
void Initialize() {
    root[0] = 0;
    tot = 0;
}
int change(int x, int a, int b, int c, int p) {
    int y = ++tot;
    v[y] = v[x] + p;
    if (a == b) return y;
    int mid = (a + b) >> 1;
    if (c <= mid)
        l[y] = change(l[x], a, mid, c, p), r[y] = r[x];
    else
        l[y] = l[x], r[y] = change(r[x], mid + 1, b, c, p);
    return y;
}
int query(int x, int a, int b, int c, int d) {
    if (!x) return 0;
    if (c <= a && b <= d) return v[x];
    int mid = (a + b) >> 1, res = 0;
    if (c <= mid) res += query(l[x], a, mid, c, d);
    if (d > mid) res += query(r[x], mid + 1, b, c, d);
    return res;
}
// 区间第k大
int kth(int x, int y, int a, int b, int k) {
    if (a == b) return a;
    int mid = (a + b) >> 1;
    if (v[l[y]] - v[l[x]] >= k)
        return kth(l[x], l[y], a, mid, k);
    else
        return kth(r[x], r[y], mid + 1, b, k - v[l[y]] + v[l[x]]);
}
// 第x个版本全局第k大
int getkth(int x, int a, int b, int k) {
    if (a == b) return a;
    int mid = (a + b) >> 1;
    if (v[l[x]] >= k)
        return getkth(l[x], a, mid, k);
    else
        return getkth(r[x], mid + 1, b, k - v[l[x]]);
}
} // namespace Persistent_Segment_Tree
/*
    离散求静态区间kth
```

```
*/
using namespace Persistent_Segment_Tree;
int cmp(int i, int j) { return a[i] < a[j]; }
int n, m, rk[N], sa[N], a[N];
int main() {
    while (~scanf("%d%d", &n, &m)) {
        Initialize();
        for (int i = 1; i <= n; i++) scanf("%d", a + i);
        for (int i = 1; i <= n; i++) sa[i] = i;
        sort(sa + 1, sa + n + 1, cmp);
        for (int i = 1; i <= n; i++) rk[sa[i]] = i;
        for (int i = 1; i <= n; i++)
            root[i] = change(root[i - 1], 1, n, rk[i], 1);
        for (int i = 1; i <= m; i++) {
            scanf("%d%d%d", &x, &y, &z);
            int id = kth(root[x - 1], root[y], 1, n, z);
            printf("%d\n", a[sa[id]]);
        }
    }
    return 0;
}
/*
    求区间非重元素的个数
*/
int T, n, m, a[N], pre[N];
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        Initialize();
        memset(pre, -1, sizeof(pre));
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        root[n + 1] = 0;
        for (int i = n; i; i--) {
            if (pre[a[i]] == -1)
                root[i] = change(root[i + 1], 1, n, i, 1);
            else {
                int tmp = change(root[i + 1], 1, n, pre[a[i]], -1);
                root[i] = change(tmp, 1, n, i, 1);
            }
            pre[a[i]] = i;
        }
        while (m--) {
            scanf("%d%d", &x, &y);
            printf("%d\n", query(root[x], 1, n, x, y));
        }
```

```
    }
    return 0;
}
```

2.27 主席树 (区间修改)

```
/*
    可持久化线段树
    区间修改lazy标记
*/
typedef long long LL;
const int N = 100010;
namespace Persistent_Segment_Tree {
int l[N * 100], r[N * 100], tot, root[N], num[N], lazy[N * 100], mark[N * 100],
    V;
LL v[N * 100];
char op[10];
void up(int x) { v[x] = v[l[x]] + v[r[x]]; }
void pb(int x, int a, int b) {
    if (mark[x]) {
        v[++tot] = v[l[x]];
        lazy[tot] = lazy[l[x]];
        l[tot] = l[l[x]];
        r[tot] = r[l[x]];
        l[x] = tot;
        v[++tot] = v[r[x]];
        lazy[tot] = lazy[r[x]];
        l[tot] = l[r[x]];
        r[tot] = r[r[x]];
        r[x] = tot;
        mark[l[x]] = mark[r[x]] = 1;
        mark[x] = 0;
    }
    if (lazy[x]) {
        int mid = (a + b) >> 1;
        lazy[l[x]] += lazy[x];
        lazy[r[x]] += lazy[x];
        v[l[x]] += (LL)lazy[x] * (mid - a + 1);
        v[r[x]] += (LL)lazy[x] * (b - mid);
        lazy[x] = 0;
    }
}
int build(int a, int b) {
    int x = ++tot;
    mark[x] = lazy[x] = 0;
```

```
    if (a == b) {
        v[x] = num[a];
        l[x] = r[x] = 0;
        return x;
    }
    int mid = (a + b) >> 1;
    return l[x] = build(a, mid), r[x] = build(mid + 1, b), up(x), x;
}
int change(int x, int a, int b, int L, int R, int val) {
    int y = ++tot;
    if (L == a && b == R) {
        lazy[y] = lazy[x] + val;
        v[y] = v[x] + LL(val) * (R - L + 1);
        if (L != R) {
            l[y] = l[x];
            r[y] = r[x];
            mark[y] = 1;
        }
        return y;
    }
    lazy[y] = v[y] = mark[y] = 0;
    pb(x, a, b);
    int mid = (a + b) >> 1;
    if (R <= mid)
        l[y] = change(l[x], a, mid, L, R, val), r[y] = r[x];
    else if (L > mid)
        l[y] = l[x], r[y] = change(r[x], mid + 1, b, L, R, val);
    else {
        l[y] = change(l[x], a, mid, L, mid, val);
        r[y] = change(r[x], mid + 1, b, mid + 1, R, val);
    }
    up(y);
    return y;
}
LL query(int x, int a, int b, int L, int R) {
    if (L == a && R == b) return v[x];
    pb(x, a, b);
    int mid = (a + b) >> 1;
    if (R <= mid)
        return query(l[x], a, mid, L, R);
    else if (L > mid)
        return query(r[x], mid + 1, b, L, R);
    else
        return query(l[x], a, mid, L, mid) +
               query(r[x], mid + 1, b, mid + 1, R);
}
```

```
} // namespace Persistent_Segment_Tree
/*
    区间修改
    历史版本区间查询
*/
int n, Q, x, y, z, ans;
int main() {
    while (~scanf("%d%d", &n, &Q)) {
        for (int i = 1; i <= n; i++) scanf("%d", &num[i]);
        V = tot = 0;
        root[0] = build(1, n);
        while (Q--) {
            scanf("%s", op);
            if (op[0] == 'Q') {
                scanf("%d%d", &x, &y);
                printf("%lld\n", query(root[V], 1, n, x, y));
            } else if (op[0] == 'H') {
                scanf("%d%d%d", &x, &y, &z);
                printf("%lld\n", query(root[z], 1, n, x, y));
            } else if (op[0] == 'C') {
                scanf("%d%d%d", &x, &y, &z);
                root[V + 1] = change(root[V], 1, n, x, y, z);
                V++;
            } else {
                scanf("%d", &x);
                V = x;
            }
        }
    }
    return 0;
}
```

2.28 线段树分治

```
/*
    Problem:
        有1e9只小怪兽，第i只编号为i
        有一张n个点，m条边的无向图n,m<=1e5
        每条边只允许编号在[l,r]之间的小怪兽通过
        问有多少小怪兽可以从1点抵达n点
    Solution:
        我们对边标号区间[l,r]进行离散，最多产生2*m-1个区间，
        对区间进行时间分治，在线段树节点上保存覆盖子树所有区间的边id，
        对线段树DFS遍历，当1和n连通时将节点包含的编号统计入答案即可
        连通性判断用可回溯并查集，复杂度O(mlogm)
```

```
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 200000 + 10;
// Union Find Set
int st[N << 1], top, f[N], d[N];
void init(int n) {
    for (int i = 1; i <= n; i++) f[i] = i, d[i] = 1;
    top = 0;
}
int sf(int x) { return f[x] == x ? x : sf(f[x]); }
void back(int tag) {
    for (; top != tag; top--) {
        if (st[top] < 0)
            d[-st[top]]--;
        else
            f[st[top]] = st[top];
    }
}
void Union(int x, int y) {
    if (d[x] > d[y]) swap(x, y);
    if (d[x] == d[y]) d[y]++, st[++top] = -y;
    f[x] = y;
    st[++top] = x;
}
// Edge
struct Data {
    int x, y, l, r;
} E[N];
// Segment Tree
vector<int> v[N << 2];
void upd(int x, int l, int r, int y) {
    int mid = (l + r) >> 1;
    if (E[y].l <= l && r <= E[y].r) {
        v[x].push_back(y);
        return;
    }
    if (E[y].l < mid) upd(x << 1, l, mid, y);
    if (E[y].r > mid) upd(x << 1 | 1, mid, r, y);
}
int n, m, ans = 0;
int b[N << 2], x, y, l, r;
void dfs(int x, int l, int r) {
    int mid = (l + r) >> 1, t = top;
    for (auto y : v[x]) {
        int fx = sf(E[y].x), fy = sf(E[y].y);
```

```
        if (fx != fy) Union(fx, fy);
    }
    if (sf(1) == sf(n)) {
        ans += b[r] - b[l];
        back(t);
        return;
    }
    if (l + 1 == r) {
        back(t);
        return;
    }
    dfs(x << 1, l, mid);
    dfs(x << 1 | 1, mid, r);
    back(t);
}
int main() {
    scanf("%d%d", &n, &m);
    init(n);
    int cnt = 0;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        E[i] = {x, y, l, r + 1};
        b[++cnt] = l;
        b[++cnt] = r + 1;
    }
    sort(b + 1, b + cnt + 1);
    int siz = unique(b + 1, b + cnt + 1) - (b + 1);
    for (int i = 1; i <= m; i++) {
        E[i].l = lower_bound(b + 1, b + siz + 1, E[i].l) - b;
        E[i].r = lower_bound(b + 1, b + siz + 1, E[i].r) - b;
        upd(1, 1, siz, i);
    }
    dfs(1, 1, siz);
    printf("%d\n", ans);
    return 0;
}

/*
    Problem:
        给定一张无向边权图，要求维护三个操作
        OP1.[x,y,z]:在点x和点y之间加一条边权为z的边，保证之前没有边
        OP2.[x,y]:将点x和y之间的边删除，保证之前有边
        OP3.[x,y]:查询x到y的路径的异或最小值，可以是非简单路
    Solution:
        图上两点异或路径的最小值为生成树上异或距离和树上环的组合
        我们以OP3为时间线建线段树，将覆盖操作时间点的边保存在线段树节点
```

```
        对OP3的线段树进行DFS遍历，用并查集维护两点间的xor距离
        当成环时将环加入xor线性基，在叶节点查询xor线性基和xor距离组合的最小值即可
        线性基空间O(30)可以选择直接传参，并查集空间O(n)需回溯
        时间复杂度O(mlogt)，t为OP3的数量，m为总边数
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 400000 + 10;
using PII = pair<int, int>;
// Xor Base
struct Base {
    int p[31];
    void ins(int x) {
        for (int j = 30; ~j; --j)
            if ((x >> j) & 1) {
                if (!p[j]) {
                    p[j] = x;
                    return;
                }
                x ^= p[j];
            }
    }
    int ask(int x) {
        for (int j = 30; ~j; --j) x = min(x, x ^ p[j]);
        return x;
    }
} S;
// Union Find Set
int st[N << 1], top, f[N], val[N], d[N];
void init(int n) {
    for (int i = 1; i <= n; i++) f[i] = i, val[i] = 0, d[i] = 1;
    top = 0;
}
int sf(int x) { return f[x] == x ? x : sf(f[x]); }
int ask(int x) {
    int res = 0;
    for (; x != f[x]; x = f[x]) res ^= val[x];
    return res;
}
void back(int tag) {
    for (; top != tag; top--) {
        if (st[top] < 0)
            d[-st[top]]--;
        else {
            f[st[top]] = st[top];
            val[st[top]] = 0;
```

```
        }
    }
}
void Union(int x, int y, int _val) {
    if (d[x] > d[y]) swap(x, y);
    if (d[x] == d[y]) d[y]++, st[++top] = -y;
    f[x] = y;
    val[x] = _val;
    st[++top] = x;
}
// Edge
struct data1 {
    int x, y, w, l, r;
} E[N];
// Query
struct data2 {
    int x, y;
} Q[N];
// Segment Tree
vector<int> v[N << 1];
int idx(int l, int r) { return l + r | l != r; }
void upd(int l, int r, int y) {
    int x = idx(l, r), mid = (l + r) >> 1;
    if (E[y].l <= l && r <= E[y].r) {
        v[x].push_back(y);
        return;
    }
    if (E[y].l <= mid) upd(l, mid, y);
    if (E[y].r > mid) upd(mid + 1, r, y);
}
void dfs(int l, int r, Base S) {
    int x = idx(l, r), t = top, mid = (l + r) >> 1;
    for (auto y : v[x]) {
        int fx = sf(E[y].x), fy = sf(E[y].y);
        int dx = ask(E[y].x), dy = ask(E[y].y);
        if (fx == fy) {
            S.ins(dx ^ dy ^ E[y].w);
            continue;
        }
        Union(fx, fy, dx ^ dy ^ E[y].w);
    }
    if (l == r) {
        int dx = ask(Q[l].x), dy = ask(Q[l].y);
        printf("%d\n", S.ask(dx ^ dy));
        back(t);
        return;
```

```
    }
    dfs(l, mid, S);
    dfs(mid + 1, r, S);
    back(t);
}
map<PII, int> id;
int n, m, x, y, z, q, op, tot;
int main() {
    scanf("%d%d", &n, &m);
    init(n);
    tot = 0;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &x, &y, &z);
        if (x > y) swap(x, y);
        E[i] = {x, y, z, tot + 1, -1};
        id[{x, y}] = i;
    }
    scanf("%d", &q);
    while (q--) {
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%d%d", &x, &y, &z);
            if (x > y) swap(x, y);
            E[++m] = {x, y, z, tot + 1, -1};
            id[{x, y}] = m;
        } else if (op == 2) {
            scanf("%d%d", &x, &y);
            if (x > y) swap(x, y);
            E[id[{x, y}]].r = tot;
        } else {
            scanf("%d%d", &x, &y);
            Q[++tot] = {x, y};
        }
    }
    for (int i = 1; i <= m; i++) {
        if (E[i].r == -1) E[i].r = tot;
        if (E[i].l <= E[i].r) upd(1, tot, i);
    }
    dfs(1, tot, S);
    return 0;
}
/*
    Problem:
        给出一张图，有些边只存在一段时间，问在一个每个时间段，
        这张图是否是二分图
    Solution:
```

```
        判断是否是二分图只要判断是否存在奇环即可，
        我们对时间进行分治，在操作树上加删边，
        保留涵盖时间区间的有效操作，将剩余操作按时间划分到两端的子树，
        退出子树的时候撤销加边操作。
        对于判断奇环，我们用并查集维护每个点与标兵的相对距离的奇偶性即可，
        由于需要撤销操作，我们放弃对并查集的压缩操作，
        采用按秩合并，保证查询的logn复杂度，同时保存每次合并过程即可。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 300010;
namespace Union_Find_Set {
int st[N], top, f[N], val[N], d[N];
void Initialize(int n) {
    for (int i = 1; i <= n; i++) f[i] = i, val[i] = 0, d[i] = 1;
    top = 0;
}
int sf(int x) { return f[x] == x ? x : sf(f[x]); }
int ask(int x) {
    int res = 0;
    for (; x != f[x]; x = f[x]) res ^= val[x];
    return res;
}
void back(int tag) {
    for (; top != tag; top--) {
        if (st[top] < 0)
            d[-st[top]]--;
        else {
            f[st[top]] = st[top];
            val[st[top]] = 0;
        }
    }
}
void Union(int x, int y, int _val) {
    if (d[x] > d[y]) swap(x, y);
    if (d[x] == d[y]) d[y]++, st[++top] = -y;
    f[x] = y;
    val[x] = _val;
    st[++top] = x;
}
} // namespace Union_Find_Set
using namespace Union_Find_Set;
struct data {
    int x, y, l, r;
} E[N];
```

```
void dfs(int l, int r, int pos) {
    int t = top;
    for (int i = 1; i <= pos; i++) {
        int x = E[i].x, y = E[i].y;
        if (E[i].l <= l && E[i].r >= r) {
            int fx = sf(x), fy = sf(y);
            int val = ask(x) ^ ask(y) ^ 1;
            if (fx == fy) {
                if (val) {
                    for (int j = l; j <= r; j++) puts("No");
                    back(t);
                    return;
                }
            }
            Union(fx, fy, val);
            swap(E[i--], E[pos--]);
        }
    }
    if (l == r) {
        puts("Yes");
        back(t);
        return;
    }
    int mid = (l + r) >> 1, ppos = pos;
    for (int i = 1; i <= ppos; i++) {
        if (E[i].l > mid) swap(E[i--], E[ppos--]);
    }
    dfs(l, mid, ppos);
    ppos = pos;
    for (int i = 1; i <= ppos; i++) {
        if (E[i].r <= mid) swap(E[i--], E[ppos--]);
    }
    dfs(mid + 1, r, ppos);
    back(t);
}
int n, m, op, x, y, l, r, T;
int main() {
    scanf("%d%d%d", &n, &m, &T);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        E[i] = (data){x, y, ++l, r};
    }
    Initialize(n);
    for (int i = 1; i <= m; i++)
        if (E[i].l > E[i].r) swap(E[i], E[m--]);
    dfs(1, T, m);
```

```
    return 0;
}
```

2.29 ODT

```
/*
    ODT
    擅长区间推平操作
    区间推平:
        在set里找到对应区间两端点，删除中间所有元素，插入一个新线段
        当出现端点在线段中时对线段进行拆分
    其余操作暴力
*/
#define IT set<node>::iterator
struct node {
    int ll, rr;
    mutable int val;
    node(int L, int R = -1, int V = 0) : ll(L), rr(R), val(V) {}
    bool operator<(const node& tt) const { return ll < tt.ll; }
};
set<node> st;
IT split(int pos) {
    IT it = st.lower_bound(node(pos));
    if (it != st.end() && it->ll == pos) return it;
    --it;
    int ll = it->ll, rr = it->rr, val = it->val;
    st.erase(it);
    st.insert(node(ll, pos - 1, val));
    return st.insert(node(pos, rr, val)).first;
}
void assign(int ll, int rr, int val) {
    // attention: first rr+1, then ll
    IT itr = split(rr + 1), itl = split(ll);
    st.erase(itl, itr);
    st.insert(node(ll, rr, val));
}
void other(int ll, int rr, int val) {
    // attention: first rr+1, then ll
    IT itr = split(rr + 1), itl = split(ll);
    for (; itl != itr; ++itl) {
        // doit
    }
}
/*
    Example:
```

```
        给定一棵树，每条边都有颜色，每次操作u,c,m
        要求将从u到根的所有边变成颜色c，然后统计对于这棵树，
        有多少种颜色恰好染了m条路
*/
#include <bits/stdc++.h>
using namespace std;
#define IT set<node>::iterator
const int N = 200010;
int c[N], cnt[N];
struct node {
    int l, r;
    mutable int val;
    node(int L, int R = -1, int V = 0) : l(L), r(R), val(V) {}
    bool operator<(const node& tt) const { return l < tt.l; }
};
set<node> st;
IT split(int pos) {
    IT it = st.lower_bound(node(pos));
    if (it != st.end() && it->l == pos) return it;
    --it;
    int l = it->l, r = it->r, val = it->val;
    st.erase(it);
    st.insert(node(l, pos - 1, val));
    return st.insert(node(pos, r, val)).first;
}
void assign(int l, int r, int val) {
    IT itr = split(r + 1), itl = split(l);
    st.erase(itl, itr);
    st.insert(node(l, r, val));
}
void update(int l, int r, int val) {
    // attention: first rr+1, then ll
    IT itr = split(r + 1), itl = split(l);
    for (; itl != itr; ++itl) {
        if (itl->val) {
            --cnt[c[itl->val]];
            c[itl->val] -= itl->r - itl->l + 1;
            ++cnt[c[itl->val]];
        }
    }
    --cnt[c[val]];
    c[val] += r - l + 1;
    ++cnt[c[val]];
    assign(l, r, val);
}
vector<int> v[N];
```

```
int id[N], dfn, top[N], fa[N], sz[N], son[N];
void dfs1(int x, int fx) {
    sz[x] = 1;
    fa[x] = fx;
    for (int y : v[x])
        if (y != fa[x]) {
            dfs1(y, x);
            sz[x] += sz[y];
            if (sz[son[x]] < sz[y]) son[x] = y;
        }
}
void dfs2(int x, int fx) {
    top[x] = fx;
    id[x] = ++dfn;
    if (son[x]) dfs2(son[x], fx);
    for (int y : v[x])
        if (y != fa[x] && y != son[x]) dfs2(y, y);
}
void path(int x, int val) {
        while (x > 1) {
                update(max(id[top[x]], 2), id[x], val);
                x = fa[top[x]];
        }
}
int n, m, q;
int main() {
    scanf("%d%d%d", &n, &m, &q);
    for(int i = 1; i < n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        v[x].push_back(y);
        v[y].push_back(x);
    }
    cnt[0] = m;
    dfs1(1, 1);
    dfs2(1, 1);
    st.insert(node(2, n));
    while (q--) {
        int u, c, m;
        scanf("%d%d%d", &u, &c, &m);
        path(u, c);
        printf("%d\n", cnt[m]);
    }
    return 0;
}
```

2.30 伸展树 (维护数列)

```
/*
    Splay维护数列
*/
namespace Splay {
int a[N]; //原数列
int val[N], mn[N], tag[N], size[N], son[N][2], f[N], tot, root;
bool rev[N];
int build(int, int, int);
void Initialize(int n) {
    tot = 0;
    root = build(0, n + 1, 0);
}
void rev1(int x) {
    if (!x) return;
    swap(son[x][0], son[x][1]);
    rev[x] ^= 1;
}
void add1(int x, int p) {
    if (!x) return;
    val[x] += p;
    mn[x] += p;
    tag[x] += p;
}
void pb(int x) {
    if (rev[x]) {
        rev1(son[x][0]);
        rev1(son[x][1]);
        rev[x] = 0;
    }
    if (tag[x]) {
        add1(son[x][0], tag[x]);
        add1(son[x][1], tag[x]);
        tag[x] = 0;
    }
}
void up(int x) {
    size[x] = 1, mn[x] = val[x];
    if (son[x][0]) {
        size[x] += size[son[x][0]];
        if (mn[x] > mn[son[x][0]]) mn[x] = mn[son[x][0]];
        //区间最大值 if(mn[x]<mn[son[x][0]])mn[x]=mn[son[x][0]];
    }
    if (son[x][1]) {
        size[x] += size[son[x][1]];
```

```
        if (mn[x] > mn[son[x][1]]) mn[x] = mn[son[x][1]];
        //区间最大值 if(mn[x]<mn[son[x][1]])mn[x]=mn[son[x][1]];
    }
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y) son[z][0] = x;
        if (son[z][1] == y) son[z][1] = x;
    }
    f[x] = f[y];
    son[x][w ^ 1] = y;
    f[y] = x;
    up(y);
}
void splay(int x, int w) {
    int s = 1, i = x, y;
    a[1] = x;
    while (f[i]) a[++s] = i = f[i];
    while (s) pb(a[s--]);
    while (f[x] != w) {
        y = f[x];
        if (f[y] != w) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    if (!w) root = x;
    up(x);
}
// root=build(0,n+1,0);
int build(int l, int r, int fa) {
    int x = ++tot, mid = (l + r) >> 1;
    f[x] = fa;
    val[x] = a[mid];
    if (l < mid) son[x][0] = build(l, mid - 1, x);
    if (r > mid) son[x][1] = build(mid + 1, r, x);
    up(x);
    return x;
}
```

```
int kth(int k) {
    int x = root, tmp;
    while (1) {
        pb(x);
        tmp = size[son[x][0]] + 1;
        if (k == tmp) return x;
        if (k < tmp)
            x = son[x][0];
        else
            k -= tmp, x = son[x][1];
    }
}
// [x,y]区间增加z
void ADD(int x, int y, int z) {
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), add1(son[y][0], z);
}
// 翻转区间[x,y]
void RESERVE(int x, int y) {
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), rev1(son[y][0]);
}
// [x,y]向右旋转T个单位
void REVOLVE(int x, int y, int z) {
    z %= y - x + 1;
    if (z) {
        int u = x, t;
        x = kth(y - z + 1), y = kth(y + 2);
        splay(x, 0), splay(y, x), t = son[y][0];
        son[y][0] = 0, up(y), up(x);
        x = kth(u), y = kth(u + 1);
        splay(x, 0), splay(y, x), son[y][0] = t, f[t] = y;
        up(y), up(x);
    }
}
// 删去第x个数字
void DELETE(int x) {
    int y = x;
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), son[y][0] = 0;
    up(y), up(x);
}
// 输出[x,y]中的最小值
void PrintMIN(int x, int y) {
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), printf("%d\n", mn[son[y][0]]);
```

```
}
// 在第x个数字后面插入y
void INSERT(int x, int y) {
    x = kth(x + 1);
    splay(x, 0);
    f[++tot] = x, val[tot] = y;
    son[tot][1] = son[x][1], f[son[x][1]] = tot, son[x][1] = tot;
    up(tot), up(x);
}
// 输出第x个位置上的数字
int A(int x) {
    int y = kth(x + 1);
    return val[y];
}
} // namespace Splay
```

2.31 伸展树 (树上问题)

```
/*
    双旋Splay
*/
const int N = 200010, inf = ~0U >> 1;
namespace Splay {
int sum[N], key[N], son[N][2], f[N], root, tot, sumx;
void Initialize(int n) {
    tot = 0;
    root = 0;
}
int up(int x) { sum[x] = sum[son[x][0]] + sum[son[x][1]] + 1; }
int rotate(int x) {
    int y = f[x];
    int w = (son[y][0] == x); // w为x与其父亲关系的相反值
    son[y][!w] = son[x][w];
    if (son[x][w]) f[son[x][w]] = y;
    if (f[y]) {
        int z = f[y];
        son[z][son[z][1] == y] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w] = y;
    up(y);
    up(x);
}
int splay(int x) {
```

```
    while (f[x]) {
        int y = f[x];
        if (f[y]) {
            if ((son[f[y]][1] == y) ^ (son[y][1] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    root = x;
}
int newnode(int k) {
    sum[++tot] = 1;
    key[tot] = k;
    return tot;
}
int add(int x, int k) {
    if (!x) return splay(newnode(k));
    int w = (key[x] < k);
    if (!son[x][w])
        son[x][w] = newnode(k), f[son[x][w]] = x, splay(son[x][w]);
    else
        add(son[x][w], k);
    up(x);
}
int find(int x, int k) {
    int ans = 0;
    sumx = 0;
    while (x) {
        if (key[x] == k) ans = x;
        if (key[x] >= k)
            x = son[x][0];
        else
            sumx += sum[son[x][0]] + 1, x = son[x][1];
    }
    return ans;
}
int del(int x) {
    splay(x);
    int w = son[x][0];
    f[son[x][0]] = f[son[x][1]] = 0;
    if (!w) return root = son[x][1];
    while (son[w][1]) w = son[w][1];
    splay(w);
    son[w][1] = son[x][1];
```

```
    if (son[x][1]) f[son[x][1]] = w;
}
int rank(int x, int k) {
    if (k == sum[son[x][0]] + 1) return key[x];
    if (k > sum[son[x][0]] + 1)
        return rank(son[x][1], k - sum[son[x][0]] - 1);
    else
        return rank(son[x][0], k);
}
int pred(int x, int k) {
    int ans = 0;
    while (x) {
        if (key[x] >= k)
            x = son[x][0];
        else
            ans = max(ans, key[x]), x = son[x][1];
    }
    return ans;
}
int succ(int x, int k) {
    int ans = inf;
    while (x) {
        if (key[x] <= k)
            x = son[x][1];
        else
            ans = min(ans, key[x]), x = son[x][0];
    }
    return ans;
}
} // namespace Splay
// Test
int main() {
    using namespace Splay;
    int n, x, y;
    scanf("%d", &n);
    while (n--) {
        scanf("%d%d", &x, &y);
        // 插入数字y
        if (x == 1) add(root, y);
        // 删除y数(若有多个相同的数, 因只删除一个)
        if (x == 2 && find(root, y)) del(find(root, y));
        // 查询x数的排名(若有多个相同的数, 输出最小的排名)
        if (x == 3) find(root, y), printf("%d\n", sumx + 1);
        // 查询排名为x的数
        if (x == 4) printf("%d\n", rank(root, y));
        // 求x的前驱(前驱定义为小于x, 且最大的数)
```

```
        if (x == 5) printf("%d\n", pred(root, y));
        // 求x的后继(后继定义为大于x, 且最小的数)
        if (x == 6) printf("%d\n", succ(root, y));
    }
    return 0;
}
```

2.32 树点分治

```
/*
    树点分治
*/
int ans, sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
void solve(int x) {
    cal(x);
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) {
            root = 0;
            dp[0] = sum = size[v[i]];
            getroot(v[i], 0);
            solve(root);
        }
    }
}
void doit() {
    memset(vis, 0, sizeof(vis));
    dp[root = 0] = sum = n;
    getroot(1, 0);
    solve(root);
}
/*
```

```
    Example
    查询树上距离不超过K的点对数
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
typedef long long LL;
const int N = 100100;
// Treap
struct node {
    LL val;
    int cnt, sum, p;
    node *l, *r;
    node() {
        val = cnt = sum = p = 0;
        l = r = NULL;
    }
    inline void up() { sum = cnt + l->sum + r->sum; }
} *blank = new (node), *T, pool[N], *cur = pool;
inline void Rotatel(node *&x) {
    node *y = x->r;
    x->r = y->l;
    x->up();
    y->l = x;
    y->up();
    x = y;
}
inline void Rotater(node *&x) {
    node *y = x->l;
    x->l = y->r;
    x->up();
    y->r = x;
    y->up();
    x = y;
}
inline void Ins(node *&x, int p) {
    if (x == blank) {
        x = cur++;
        x->val = p;
        x->l = x->r = blank;
        x->cnt = x->sum = 1;
        x->p = std::rand();
        return;
    }
```

```
    x->sum++;
    if (p == x->val) {
        x->cnt++;
        return;
    }
    if (p < x->val) {
        Ins(x->l, p);
        if (x->l->p > x->p) Rotater(x);
    } else {
        Ins(x->r, p);
        if (x->r->p > x->p) Rotatel(x);
    }
}
inline int Ask(node *x, LL p) {
    if (x == blank) return 0;
    if (p == x->val) return x->l->sum + x->cnt;
    if (p > x->val) return x->cnt + x->l->sum + Ask(x->r, p);
    return Ask(x->l, p);
}
// 邻接表
LL w[N << 1];
int tot, g[N], nxt[N << 1], v[N << 1];
void add_edge(int x, int y, int z) {
    v[tot] = y, w[tot] = z, nxt[tot] = g[x], g[x] = tot++;
}
// 树分治
LL K;
int ans, sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
inline void dfscal(int x, LL l, int pre) {
    if (l > K) return;
    ans += Ask(T, K - l);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != pre) dfscal(v[i], l + w[i], x);
```

```
    }
}
inline void dfsadd(int x, LL l, int pre) {
    if (l > K) return;
    Ins(T, l);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != pre) dfsadd(v[i], l + w[i], x);
    }
}
void cal(int x) {
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) dfscal(v[i], w[i], x), dfsadd(v[i], w[i], x);
    }
}
void solve(int x) {
    cur = pool;
    Ins(T = blank, 0);
    cal(x);
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) {
            root = 0;
            dp[0] = sum = size[v[i]];
            getroot(v[i], 0);
            solve(root);
        }
    }
}
int n, m;
int main() {
    blank->l = blank->r = blank;
    while (~scanf("%d%d", &n, &m)) {
        memset(g, -1, sizeof(g));
        memset(vis, 0, sizeof(vis));
        ans = tot = 0;
        char str[10];
        int a, b;
        LL c;
        for (int i = 0; i < m; i++) {
            scanf("%d%d%lld%s", &a, &b, &c, str);
            add_edge(a, b, c);
            add_edge(b, a, c);
        }
        scanf("%lld", &K);
        ans = K ? 0 : n;
        dp[root = 0] = sum = n;
```

```
        getroot(1, 0);
        solve(root);
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example
    在点权树上查询点权相乘对P取模为K的链，要求输出起点终点字典序最小的链(不能是单点)
    维护hash保存某个链权值的最小端点，沿dfs查询即可，
    注意点权的特殊处理方式，存值时起始点要包含重心，查询时不包含
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int P = 1000003;
const int N = 1000010;
const int INF = 0x3f3f3f3f;
int inv[N], mp[N];
void pre_inv() {
    inv[1] = 1;
    for (int i = 2; i < P; i++) {
        int a = P / i, b = P % i;
        inv[i] = (1LL * inv[b] * (-a) % P + P) % P;
    }
}
// 邻接表
int tot, g[N], nxt[N << 1], v[N << 1];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
}
void add_edge(int x, int y) { v[tot] = y, nxt[tot] = g[x], g[x] = tot++; }
// 树分治
int K;
int sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
```

```
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
int ans1, ans2, val[N];
void Insert(int x, int L) {
    int now = mp[L];
    if (!now || x < now) mp[L] = x;
}
void Query(int x, int L) {
    L = 1LL * inv[L] * K % P;
    int y = mp[L];
    if (y == 0) return;
    if (y < x) swap(y, x);
    if (x < ans1 || (x == ans1 && y < ans2)) ans1 = x, ans2 = y;
}
// 三个相似的函数注意不要写串
void dfsadd(int x, int w, int pre) {
    Insert(x, w);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != pre)
            dfsadd(v[i], (1LL * w * val[v[i]]) % P, x);
    }
}
void dfscal(int x, int w, int pre) {
    Query(x, w);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != pre)
            dfscal(v[i], (1LL * w * val[v[i]]) % P, x);
    }
}
void dfsdel(int x, int w, int pre) {
    mp[w] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != pre)
            dfsdel(v[i], (1LL * w * val[v[i]]) % P, x);
    }
}
void cal(int x) {
    mp[val[x]] = x; // 将重心加入查询
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]])
            dfscal(v[i], val[v[i]], x),
                dfsadd(v[i], 1LL * val[v[i]] * val[x] % P, x);
    }
    for (int i = g[x]; ~i; i = nxt[i])
```

```
        if (!vis[v[i]]) dfsdel(v[i], 1LL * val[v[i]] * val[x] % P, x);
    mp[val[x]] = 0; // 注意重心的删除
}
void solve(int x) {
    cal(x);
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) {
            root = 0;
            dp[0] = sum = size[v[i]];
            getroot(v[i], 0);
            solve(root);
        }
    }
}
void read(int &x) {
    x = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') ch = getchar();
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
}
int n;
int main() {
    pre_inv();
    while (~scanf("%d%d", &n, &K)) {
        init();
        for (int i = 1; i <= n; i++) read(val[i]);
        for (int i = 1; i < n; i++) {
            int x, y;
            read(x);
            read(y);
            add_edge(x, y);
            add_edge(y, x);
        }
        ans1 = ans2 = INF;
        memset(vis, 0, sizeof(vis));
        dp[root = 0] = sum = n;
        getroot(1, 0);
        solve(root);
        if (ans1 == INF)
            puts("No solution");
        else
            printf("%d %d\n", ans1, ans2);
```

```
    }
    return 0;
}
```

2.33 树点分治 + 容斥

```
/*
    树分治+容斥
    题目大意：给出一棵边权树，求链长小于等于W，链上边数小于等于L的链数量
    题解：我们求出树上重心，进行分治处理，对于当前分治层，
    计算出所有包含重心的链，记录链的长度和链上边数，我们将链按照长度排序，
    双指针逆序根据长度顺序将满足长度相加条件的链上边数加入树状数组维护，
    查询长度满足条件位置的统计和，因为这种计算多算了子树内部不过重心的答案，
    所以我们减去对于每个子树的以重心为起点的链计算出的答案，
    最后答案减去单点的链并去除偏序性
*/
const int N = 300010;
typedef pair<int, int> P;
long long ans;
int L, W;
// 树状数组 [注意0位置的处理]
int n, c[N];
void add(int x, int v) {
    if (!x)
        c[0] += v;
    else
        while (x <= n) c[x] += v, x += x & -x;
}
int query(int x) {
    int res = c[0];
    while (x) res += c[x], x -= x & -x;
    return res;
}
// 邻接表
int tot, g[N], nxt[N << 1], v[N << 1], w[N << 1];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
}
void add_edge(int x, int y, int z) {
    v[tot] = y, w[tot] = z, nxt[tot] = g[x], g[x] = tot++;
}
// 树分治
int K;
int sum, top, root, size[N], dp[N], vis[N], len[N], dis[N];
```

```
P st[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
void getdis(int x, int fx) {
    st[++top] = P(dis[x], len[x]);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            len[v[i]] = len[x] + 1;
            dis[v[i]] = dis[x] + w[i];
            getdis(v[i], x);
        }
    }
}
void cal(int x, int _len, int _dis, int f) {
    len[x] = _len;
    dis[x] = _dis;
    getdis(x, top = 0);
    sort(st + 1, st + top + 1);
    int p = 1;
    for (int i = top; i; i--) {
        while (p <= top && st[p].first + st[i].first <= W)
            add(st[p++].second, 1);
        if (L >= st[i].second) ans += f * query(L - st[i].second);
    }
    for (int i = 1; i < p; i++) add(st[i].second, -1);
}
void solve(int x) {
    cal(x, 0, 0, 1);
    vis[x] = 1;
    // 删去不过重心的分支内部组合
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) cal(v[i], 1, w[i], -1);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) {
            root = 0;
```

```
            dp[0] = sum = size[v[i]];
            getroot(v[i], 0);
            solve(root);
        }
    }
}
int main() {
    while (~scanf("%d%d%d", &n, &L, &W)) {
        init();
        for (int i = 2; i <= n; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            add_edge(i, x, y);
            add_edge(x, i, y);
        }
        ans = 0;
        memset(vis, 0, sizeof(vis));
        dp[root = 0] = sum = n;
        getroot(1, 0);
        solve(root);
        // 去除自己到自己的组合并消除答案的偏序性
        printf("%lld\n", (ans - n) / 2);
    }
    return 0;
}
/*
    题目大意：给出一个城市的道路连通情况，城镇之间连接呈树状，
    有些城镇上有超市有些没有，居民想去超市会去离其最近的有超市的城市，
    如果一样近则会去编号小的，现在要开一家新超市，问开在哪里会有最高的人流量
    题解：我们先通过两遍dfs计算出每个城镇和最近的超市的距离d[x]和最小id，
    考虑树分治，对于每个分治层，计算点到重心的距离，
    统计一点过重心的链长是否小于另一点的d值，即dis[x]+dis[y]<d[y]
    我们在获取分治层信息时保存d[y]-dis[y]以及id[y]，
    那么计算对于x的贡献时只要在排序后的分治层信息中二分找到其能吸引的城镇数即可，
    由于计算的时候会计算同分支子树的信息组合，所以要分别计算分支信息并从答案中减去
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100010;
int d[N], id[N], c[N];
int tot, g[N], nxt[N << 1], v[N << 1], w[N << 1];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
```

```
}
void add_edge(int x, int y, int z) {
    v[tot] = y, w[tot] = z, nxt[tot] = g[x], g[x] = tot++;
}
void dfs(int x, int fx) {
    if (c[x]) {
        id[x] = x;
        d[x] = 0;
    }
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx) continue;
        dfs(v[i], x);
        if (d[v[i]] + w[i] < d[x])
            d[x] = d[v[i]] + w[i], id[x] = id[v[i]];
        else if (d[v[i]] + w[i] == d[x])
            id[x] = min(id[x], id[v[i]]);
    }
}
void dfs2(int x, int fx) {
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx) continue;
        if (d[x] + w[i] < d[v[i]])
            d[v[i]] = d[x] + w[i], id[v[i]] = id[x];
        else if (d[x] + w[i] == d[v[i]])
            id[v[i]] = min(id[x], id[v[i]]);
        dfs2(v[i], x);
    }
}
int sum, root, size[N], dp[N], vis[N], ans[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
void getsize(int x, int fx) {
    size[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
```

```
            getsize(v[i], x);
            size[x] += size[v[i]];
        }
    }
}
int top, dis[N];
struct data {
    int d, id;
} st[N];
bool cmp(data x, data y) { return x.d < y.d || (x.d == y.d && x.id < y.id); }
void getdis(int x, int fx, int l) {
    dis[x] = l;
    st[++top] = data{d[x] - l, id[x]};
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]] && v[i] != fx) getdis(v[i], x, l + w[i]);
}
int BinarySearch(int D, int ID) {
    int l = 1, r = top, ans = 0;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (st[mid].d < D || (st[mid].d == D && st[mid].id <= ID))
            l = mid + 1, ans = mid;
        else
            r = mid - 1;
    }
    return ans;
}
void getans(int x, int fx, int f, int s) {
    ans[x] += f * (s - BinarySearch(dis[x], x));
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]] && v[i] != fx) getans(v[i], x, f, s);
}
void cal(int x, int l, int f) {
    top = 0;
    getdis(x, x, l);
    sort(st + 1, st + top + 1, cmp);
    getans(x, x, f, size[x]);
}
void solve(int x) {
    getsize(x, x);
    cal(x, 0, 1);
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]]) {
            cal(v[i], w[i], -1);
            root = 0;
```

```
            dp[0] = sum = size[v[i]];
            getroot(v[i], 0);
            solve(root);
        }
    }
}
int n;
int main() {
    while (~scanf("%d", &n)) {
        init();
        memset(d, 0x3f, sizeof(d));
        memset(id, 0, sizeof(id));
        for (int i = 1; i < n; i++) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            add_edge(x, y, z);
            add_edge(y, x, z);
        }
        for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
        dfs(1, 1);
        dfs2(1, 1);
        memset(ans, 0, sizeof(ans));
        memset(vis, 0, sizeof(vis));
        root = 0;
        dp[0] = sum = n;
        getroot(1, 0);
        solve(root);
        int res = 0;
        for (int i = 1; i <= n; i++)
            if (!c[i]) res = max(res, ans[i]);
        printf("%d\n", res);
    }
    return 0;
}
```

2.34 重心树

```
/*
    重心树
    将分治层的重心和父重心之间连边构树
    题目大意：给出一棵点权树，允许修改点权，询问与点x距离差小于等于d的点的点权和
    题解：对于原树建立重心树，对于每个点维护两棵权值线段树，
    分别表示以该点为重心的分治层中到这个点的距离的点集以及到这个点重心树父亲距离的点集
    在统计x节点的时候只要统计其到重心树根的链上每个重心的贡献值累加即可，
    每个重心的贡献要计算折算后的距离在第一棵线段树中的答案，
```

```
    减去子重心的父重心在第二棵线段树的统计答案，即删去Fx和x之间的不该被统计的点权。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 200010, LEV = 20;
int dph[N];
int fa[N][LEV];
vector<int> G[N];
inline void add(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
}
void dfs(int rt, int f) {
    for (int i = 1; i < LEV; i++) {
        if (dph[rt] - 1 < (1 << i)) break;
        fa[rt][i] = fa[fa[rt][i - 1]][i - 1];
    }
    for (int v : G[rt]) {
        if (v == f) continue;
        dph[v] = dph[rt] + 1;
        fa[v][0] = rt;
        dfs(v, rt);
    }
}
inline int lca(int a, int b) {
    if (dph[a] < dph[b]) swap(a, b);
    int t = dph[a] - dph[b];
    for (int i = 0; i < LEV; i++)
        if ((1 << i) & t) a = fa[a][i];
    for (int i = LEV - 1; i >= 0; i--) {
        if (fa[a][i] != fa[b][i]) a = fa[a][i], b = fa[b][i];
    }
    if (a != b) return fa[a][0];
    return a;
}
inline int dis(int x, int y) { return dph[x] + dph[y] - 2 * dph[lca(x, y)]; }
void Initialize(int n) {
    for (int i = 1; i <= n; i++) G[i].clear();
    memset(fa, 0, sizeof(fa));
    dph[1] = 1;
}
int K;
int sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
```

```
    size[x] = 1;
    dp[x] = 0;
    for (int v : G[x]) {
        if (!vis[v] && v != fx) {
            getroot(v, x);
            size[x] += size[v];
            dp[x] = max(dp[x], size[v]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
// 建立重心树
int belong[N];
void build(int x, int fx) {
    belong[x] = fx;
    vis[x] = 1;
    for (int v : G[x])
        if (!vis[v]) {
            root = 0;
            sum = size[v];
            getroot(v, x);
            build(root, x);
        }
}
int n, m, ans, tot, rt[N], rtc[N];
struct data {
    int l, r, sum;
} T[N * 60];
inline void up(int x) { T[x].sum = T[T[x].l].sum + T[T[x].r].sum; }
void insert(int &rt, int l, int r, int x, int val) {
    if (!rt) {
        rt = ++tot;
        T[rt].l = T[rt].r = T[rt].sum = 0;
    }
    if (l == r) {
        T[rt].sum += val;
        return;
    }
    int mid = (l + r) >> 1;
    if (x <= mid)
        insert(T[rt].l, l, mid, x, val);
    else
        insert(T[rt].r, mid + 1, r, x, val);
    up(rt);
}
```

```
int query(int x, int l, int r, int L, int R) {
    if (L > R) return 0;
    if (L <= l && r <= R) return T[x].sum;
    int mid = (l + r) >> 1, res = 0;
    if (L <= mid) res += query(T[x].l, l, mid, L, R);
    if (R > mid) res += query(T[x].r, mid + 1, r, L, R);
    return res;
}
void addval(int fx, int x, int val) {
    int D = dis(fx, x);
    insert(rt[fx], 0, n, D, val);
    if (!belong[fx]) return;
    int Fx = belong[fx];
    D = dis(Fx, x);
    insert(rtc[fx], 0, n, D, val);
    addval(Fx, x, val);
}
void ask(int x, int son, int v, int d) {
    if (!x) return;
    if (x == son)
        ans += query(rt[x], 0, n, 0, d);
    else {
        int D = dis(x, v);
        ans += query(rt[x], 0, n, 0, d - D);
        ans -= query(rtc[son], 0, n, 0, d - D);
    }
    ask(belong[x], x, v, d);
}
int a[N];
int main() {
    while (~scanf("%d%d", &n, &m)) {
        tot = 0;
        Initialize(n);
        memset(rt, 0, sizeof(rt));
        memset(rtc, 0, sizeof(rtc));
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        for (int i = 1; i < n; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            add(x, y);
        }
        dfs(1, 0);
        memset(vis, 0, sizeof(vis));
        dp[root = 0] = sum = n;
        getroot(1, 0);
        build(root, 0);
```

```
        for (int i = 1; i <= n; i++) addval(i, i, a[i]);
        while (m--) {
            char op[10];
            int x, y;
            ans = 0;
            scanf("%s%d%d", op, &x, &y);
            if (op[0] == '!')
                addval(x, x, y - a[x]), a[x] = y;
            else {
                ask(x, x, x, y);
                printf("%d\n", ans);
            }
        }
    }
    return 0;
}
```

2.35 重链剖分

```
/*
    树链剖分
*/
int seq[N]; // seq记录的是dfs序，方便用线段树维护data[seq[i]]
namespace Tree_Chain_Subdivision {
int ed, root, d[N], num[N], v[N << 1], vis[N], f[N], g[N << 1];
int nxt[N << 1], size[N], son[N], st[N], en[N], dfn, top[N];
void add_edge(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    size[x] = 1;
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != f[x]) {
            f[v[i]] = x, d[v[i]] = d[x] + 1;
            dfs(v[i]), size[x] += size[v[i]];
            if (size[v[i]] > size[son[x]]) son[x] = v[i];
        }
}
void dfs2(int x, int y) {
    if (x == -1) return;
    st[x] = ++dfn;
    seq[dfn] = x;
    top[x] = y;
```

```
    if (son[x]) dfs2(son[x], y);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != son[x] && v[i] != f[x]) dfs2(v[i], v[i]);
    en[x] = dfn;
}
//查询x,y两点的lca
int lca(int x, int y) {
    for (; top[x] != top[y]; x = f[top[x]])
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
    return d[x] < d[y] ? x : y;
}
// x是y的祖先，查询x到y方向的第一个点
int lca2(int x, int y) {
    int t;
    while (top[x] != top[y]) t = top[y], y = f[top[y]];
    return x == y ? t : son[x];
}
//以root为根对x的子树操作
int subtree(int x, int n) {
    if (x == root) {
        return Segment_Tree::query(1, 1, n);
    }
    if (st[x] > st[root] || en[x] < en[root]) {
        return Segment_Tree::query(1, st[x], en[x]);
    }
    int y = lca2(x, root);
    return min(Segment_Tree::query(1, 1, st[y] - 1),
               Segment_Tree::query(1, en[y] + 1, n));
}
//对x到y路径上的点进行操作
void chain(int x, int y, int p) {
    for (; top[x] != top[y]; x = f[top[x]]) {
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
        Segment_Tree::change(1, st[top[x]], st[x], p);
    }
    if (d[x] < d[y]) {
        int z = x;
        x = y;
```

```
        y = z;
    }
    Segment_Tree::change(1, st[y], st[x], p);
    // 如果是边权转点权，则为change(st[y]+1,st[x])
}
void Initialize() {
    memset(g, dfn = ed = 0, sizeof(g));
    memset(v, 0, sizeof(v));
    memset(nxt, 0, sizeof(nxt));
    memset(son, -1, sizeof(son));
}
} // namespace Tree_Chain_Subdivision
```

2.36 长链剖分

```
/*
    Problem:
        给定一棵根为1的树，对于每个节点，求子树中，哪个距离下的节点数量最多，
        当数量相同时，取较小的那个距离值
    Solution:
        我们对树进行长链剖分，记录sum[x]数组表示节点x不同距离的点的数量，
        同一条长链中sum[fx][i+1]=sum[x][i]，直接继承长链，暴力轻儿子即可
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 1000010;
vector<int> G[N];
int n, x, y, len[N], son[N], ans[N], pos[N], dfn;
void dfs(int x, int fx) {
    len[x] = 0;
    for (int y : G[x]) {
        if (y == fx) continue;
        dfs(y, x);
        if (len[y] > len[son[x]]) {
            len[x] = len[y] + 1;
            son[x] = y;
        }
    }
}
int poor[N], *p = poor, *sum[N];
void dfs2(int x, int fx) {
    sum[x][0] = 1;
    if (son[x]) {
        sum[son[x]] = sum[x] + 1;
        dfs2(son[x], x);
```

```
        ans[x] = ans[son[x]] + 1;
    }
    for (int y : G[x]) {
        if (y == fx || y == son[x]) continue;
        sum[y] = p;
        p += len[y] + 1;
        dfs2(y, x);
        for (int i = 0; i <= len[y]; i++) {
            sum[x][i + 1] += sum[y][i];
            if ((sum[x][i + 1] > sum[x][ans[x]]) ||
                (sum[x][i + 1] == sum[x][ans[x]] && i + 1 < ans[x]))
                ans[x] = i + 1;
        }
    }
    if (sum[x][ans[x]] == 1) ans[x] = 0;
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        scanf("%d%d", &x, &y);
        G[x].push_back(y);
        G[y].push_back(x);
    }
    len[0] = -1;
    dfs(1, 0);
    sum[1] = p;
    p += len[1] + 1;
    dfs2(1, 0);
    for (int i = 1; i <= n; i++) printf("%d\n", ans[i]);
    return 0;
}
/*
    Problem:
        求一棵树上三点距离两两相等的三元组对数
        树的边权均为1
    Solution:
        设f[i][j]表示i子树距离i为j的点数量
        设g[i][j]表示i子树两点lca距离彼此为d，且该lca距离i点为d-j的点对数
        当y子树并入父节点x时有
        g[x][j+1]+=f[x][j+1]*f[y][j]
        g[x][j-1]+=g[y][j]
        f[x][j+1]+=f[y][j]
        ans=f[x][j]*g[y][j+1]+g[x][j]*f[y][j-1]
        我们发现状态转移只跟节点深度有关，因此可以长链剖分优化
        同一条长链上对于f数组有f[fx][i+1]=f[x][i],
        对于g数组有g[fx][i-1]=g[x][i],
```

```
        因此g数组前后均要预留len[x]+1，需要开两倍空间，
        继承重儿子的g和f函数，暴力统计轻链，同时计算ans即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 100010;
vector<int> G[N];
int n, x, y, len[N], son[N];
void dfs(int x, int fx) {
    len[x] = 0;
    for (int i = 0; i < G[x].size(); i++) {
        int y = G[x][i];
        if (y == fx) continue;
        dfs(y, x);
        if (len[y] > len[son[x]]) {
            len[x] = len[y] + 1;
            son[x] = y;
        }
    }
}
ll poorf[N], poorg[N << 1], *pf = poorf, *pg = poorg, *f[N], *g[N], ans = 0;
void dfs2(int x, int fx) {
    f[x][0] = 1;
    if (son[x]) {
        f[son[x]] = f[x] + 1;
        g[son[x]] = g[x] - 1;
        dfs2(son[x], x);
    }
    ans += g[x][0];
    for (int i = 0; i < G[x].size(); i++) {
        int y = G[x][i];
        if (y == fx || y == son[x]) continue;
        f[y] = pf, pf += len[y] + 1;
        pg += len[y] + 1, g[y] = pg, pg += len[y] + 1;
        dfs2(y, x);
        for (int j = 0; j <= len[y]; j++) {
            if (j) ans += f[x][j - 1] * g[y][j];
            ans += g[x][j + 1] * f[y][j];
        }
        for (int j = 0; j <= len[y]; j++) {
            g[x][j + 1] += f[x][j + 1] * f[y][j];
            if (j) g[x][j - 1] += g[y][j];
            f[x][j + 1] += f[y][j];
        }
    }
```

```
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        scanf("%d%d", &x, &y);
        G[x].push_back(y);
        G[y].push_back(x);
    }
    len[0] = -1;
    dfs(1, 0);
    f[1] = pf, pf += len[1] + 1;
    pg += len[1] + 1, g[1] = pg, pg += len[1] + 1;
    dfs2(1, 0);
    printf("%lld\n", ans);
    return 0;
}
/*
    Problem:
        给出一棵树，每个点上有一个点权，点权<=30000
        现在求有多少条链，链上点权的gcd>1
    Solution:
        考虑枚举i的倍数的点权，在i的倍数的点生成的诱导子图上求等于D的链长，
        因为点均为i的倍数，因此不需要考虑gcd的问题，
        枚举的i值包含倍数关系，需要容斥，有效的i值为miu值不为0的值，
        i对应的诱导子图计算出的长度等于D的链长对答案的贡献为-miu*sum
        诱导子图求链长为D的链数量可以用长链剖分来求，短链并入长链边合并边计算答案，
        则复杂度为每个诱导子图的点数和，考虑数字a[i]，对诱导子图的点数贡献为2^(loglog(a[i]))
        因此复杂度为O(n*2^(log(log(n))))
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 500010;
const int M = 30010;
typedef long long ll;
vector<int> E[M], P[M];
int v[N], cv;
int g[N], to[N << 1], nxt[N << 1], ed;
int m, mrk, n, D, a[N], x[N], y[N], len[N], son[N], vis[N];
ll ANS, Ans;
inline void add(int x, int y) {
    to[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x, int fx) {
    len[x] = 0;
```

```
    for (int o = g[x]; o; o = nxt[o]) {
        int y = to[o];
        if (y == fx) continue;
        dfs(y, x);
        if (len[y] > len[son[x]]) {
            len[x] = len[y] + 1;
            son[x] = y;
        }
    }
}
int poor[N], *p = poor, *sum[N];
void dfs2(int x, int fx) {
    vis[x] = mrk;
    sum[x][0] = 1;
    if (son[x]) {
        sum[son[x]] = sum[x] + 1;
        dfs2(son[x], x);
    }
    for (int o = g[x]; o; o = nxt[o]) {
        int y = to[o];
        if (y == fx || y == son[x]) continue;
        sum[y] = p;
        p += len[y] + 1;
        dfs2(y, x);
        for (int i = 0; i <= len[y] && i < D - 1; i++)
            if (len[x] >= D - i - 1) Ans += 1ll * sum[y][i] * sum[x][D - i - 1];
        for (int i = 0; i <= len[y]; i++) sum[x][i + 1] += sum[y][i];
    }
    if (len[x] >= D) Ans += sum[x][D];
}
bool notp[M];
int prime[M], pnum, mu[M];
void sieve() {
    notp[0] = notp[1] = mu[1] = 1;
    for (int i = 2; i < M; i++) {
        if (!notp[i]) {
            prime[++pnum] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= pnum && prime[j] * i < M; j++) {
            notp[prime[j] * i] = 1;
            if (i % prime[j] == 0) {
                mu[prime[j] * i] = 0;
                break;
            }
            mu[prime[j] * i] = -mu[i];
```

```
        }
    }
}
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
int main() {
    int T;
    read(T);
    len[0] = -1;
    sieve();
    for (int cas = 1; cas <= T; cas++) {
        ANS = 0;
        for (int i = 1; i <= m; i++) {
            E[i].clear();
            P[i].clear();
        }
        m = 0;
        read(n);
        read(D);
        for (int i = 1; i <= n; i++) {
            vis[i] = 0;
            read(a[i]);
            m = max(m, a[i]);
            P[a[i]].push_back(i);
        }
        for (int i = 1; i < n; i++) {
            read(x[i]);
            read(y[i]);
            int gcd = __gcd(a[x[i]], a[y[i]]);
            E[gcd].push_back(i);
        }
        for (int i = 2; i <= m; i++) {
```

```
            if (!mu[i]) continue;
            mrk = i;
            cv = 0; // i的倍数点数组
            for (int j = i; j <= m; j += i) {
                for (auto u : P[j]) {
                    v[++cv] = u;
                    son[u] = 0; // 清空长儿子
                    g[u] = 0;  // 清空u的邻接表
                }
            }
            ed = 0;
            for (int j = i; j <= m; j += i) {
                for (auto id : E[j]) {
                    add(x[id], y[id]);
                    add(y[id], x[id]);
                }
            }
            for (int j = 1; j <= cv; j++) {
                int u = v[j];
                if (vis[u] == mrk) continue;
                Ans = 0;
                p = poor;
                dfs(u, 0);
                sum[u] = p;
                p += len[u] + 1;
                dfs2(u, 0);
                ANS -= mu[i] * Ans;
            }
        }
        printf("Case #%d: %lld\n", cas, ANS * 2);
    }
    return 0;
}
```

2.37 动态树 (链修改)

```
/*
    动态树(链修改)
    初始 for(int i=1;i<=n;i++)sum[i]=val[i],mt[i]=size[i]=1
*/
namespace Link_Cut_Tree {
int f[N], son[N][2], tmp[N], size[N];
bool rev[N];
LL val[N], sum[N], at[N], mt[N];
void Initialize() {
```

```
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    memset(val, 0, sizeof(val));
    memset(rev, 0, sizeof(rev));
    memset(sum, 0, sizeof(sum));
    memset(at, 0, sizeof(at));
    memset(mt, 0, sizeof(mt));
}
void modify(int x, int m, int a) {
    if (!x) return;
    val[x] = (val[x] * m + a) % mod;
    sum[x] = (sum[x] * m + a * size[x]) % mod;
    at[x] = (at[x] * m + a) % mod;
    mt[x] = (mt[x] * m) % mod;
}
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void rev1(int x) {
    if (!x) return;
    swap(son[x][0], son[x][1]);
    rev[x] ^= 1;
}
void pb(int x) {
    if (rev[x]) rev1(son[x][0]), rev1(son[x][1]), rev[x] = 0;
    int m = mt[x], a = at[x];
    mt[x] = 1;
    at[x] = 0;
    if (m != 1 || a != 0) {
        modify(son[x][0], m, a);
        modify(son[x][1], m, a);
    }
}
void up(int x) {
    sum[x] = (val[x] + sum[son[x][0]] + sum[son[x][1]]) % mod;
    size[x] = (size[son[x][0]] + size[son[x][1]] + 1) % mod;
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
```

```
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
// 查询x所在的树的根
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
// 使x成为根
void makeroot(int x) {
    access(x);
    splay(x);
    rev1(x);
}
// 将x和y所属树合并
void link(int x, int y) {
    makeroot(x);
    f[x] = y;
    access(x);
}
// 将x和其父节点分开
void cutf(int x) {
```

```
    access(x);
    splay(x);
    f[son[x][0]] = 0;
    son[x][0] = 0;
    up(x);
}
// 将边x-y切断
void cut(int x, int y) {
    makeroot(x);
    cutf(y);
}
// 提取链
void split(int x, int y) {
    makeroot(y);
    access(x);
    splay(x);
}
// 查询x到y的链和，注意考虑清楚是int还是LL
int ask(int x, int y) {
    split(x, y);
    return sum[x];
}
// 查询节点到根的距离，注意考虑清楚是int还是LL
int query(int x) {
    access(x);
    splay(x);
    return sum[x];
}
// 将x为下标的值改为y
int change(int x, int y) {
    makeroot(x);
    val[x] = y;
    up(x);
}
// 将x的父亲改为y
int changef(int x, int y) {
    cutf(x);
    f[x] = y;
}
// 链上点权加法
void add(int x, int y, int z) {
    split(x, y);
    modify(x, 1, z);
}
// 链上点权乘法
void mul(int x, int y, int z) {
```

```
    split(x, y);
    modify(x, z, 0);
}
} // namespace Link_Cut_Tree
```

2.38 动态树 (点修改))

```
/*
    动态树(点修改)
    初始化的时候，读入val[i]，sum[i]=xor[i]=val[i]
*/
namespace Link_Cut_Tree {
int f[N], son[N][2], val[N], sum[N], tmp[N], Xor[N];
bool rev[N];
void Initialize() {
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    memset(val, 0, sizeof(val));
    memset(rev, 0, sizeof(rev));
    memset(sum, 0, sizeof(sum));
    memset(Xor, 0, sizeof(Xor));
}
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void rev1(int x) {
    if (!x) return;
    swap(son[x][0], son[x][1]);
    rev[x] ^= 1;
}
void pb(int x) {
    if (rev[x]) rev1(son[x][0]), rev1(son[x][1]), rev[x] = 0;
}
void up(int x) {
    sum[x] = Xor[x] = val[x];
    if (son[x][0]) sum[x] += sum[son[x][0]];
    if (son[x][1]) sum[x] += sum[son[x][1]];
    if (son[x][0]) Xor[x] ^= Xor[son[x][0]];
    if (son[x][1]) Xor[x] ^= Xor[son[x][1]];
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
```

```
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
// 查询x所在的树的根
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
// 使x成为根
void makeroot(int x) {
    access(x);
    splay(x);
    rev1(x);
}
// 将x和y所属树合并
void link(int x, int y) {
    makeroot(x);
    f[x] = y;
```

```
    access(x);
}
// 将x和其父节点分开
void cutf(int x) {
    access(x);
    splay(x);
    f[son[x][0]] = 0;
    son[x][0] = 0;
    up(x);
}
// 将边x-y切断
void cut(int x, int y) {
    makeroot(x);
    cutf(y);
}
// 查询x到y的链和
int ask(int x, int y) {
    makeroot(x);
    access(y);
    splay(y);
    return sum[y];
}
// 计算x到y的xor和
int xorsum(int x, int y) {
    makeroot(x);
    access(y);
    splay(y);
    return Xor[y];
}
// 查询节点到根的距离
int query(int x) {
    access(x);
    splay(x);
    return sum[x];
}
// 将x为下标的值改为y
int change(int x, int y) {
    makeroot(x);
    val[x] = y;
    up(x);
}
// 将x的父亲改为y
int changef(int x, int y) {
    cutf(x);
    f[x] = y;
}
```

```
} // namespace Link_Cut_Tree
```

2.39 启发式合并动态树

```
/*
    启发式合并LCT
    加边，查询每个连通块的重心到其它点的距离和的和
*/
namespace Heuristic_LCT {
int ans, g[N], v[N << 1], nxt[N << 1], ed;
int f[N], son[N][2], val[N], tag[N], sum[N], ts[N], td[N], size[N], tmp[N];
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void add1(int x, int p) {
    if (!x) return;
    val[x] += p;
    tag[x] += p;
}
void add2(int x, int s, int d) {
    if (!x) return;
    sum[x] += s + size[son[x][1]] * d;
    ts[x] += s;
    td[x] += d;
}
void pb(int x) {
    if (tag[x]) {
        add1(son[x][0], tag[x]);
        add1(son[x][1], tag[x]);
        tag[x] = 0;
    }
    if (td[x]) {
        add2(son[x][0], ts[x] + (size[son[x][1]] + 1) * td[x], td[x]);
        add2(son[x][1], ts[x], td[x]);
        ts[x] = td[x] = 0;
    }
}
void up(int x) { size[x] = size[son[x][0]] + size[son[x][1]] + 1; }
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
```

```
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void addleaf(int x, int y) {
    f[y] = x,
    son[y][0] = son[y][1] = val[y] = tag[y] = sum[y] = ts[y] = td[y] = 0,
    size[y] = 1;
    x = root(x), access(y), splay(x), add1(x, 1), add2(x, 0, 1);
    for (y = son[x][1]; son[y][0]; y = son[y][0])
        ;
    splay(y);
    int vx = val[x], vy = val[y];
    if (vy * 2 > vx) {
        val[y] = vx, val[x] -= vy;
        sum[x] -= sum[y] + vy, sum[y] += sum[x] + vx - vy;
        access(y), splay(x), son[x][0] = y, son[x][1] = 0;
```

```
    }
}
void dfs(int x, int y) {
    addleaf(y, x);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != y) dfs(v[i], x);
}
void addedge(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void link(int x, int y) {
    int X = root(x), Y = root(y);
    ans -= sum[X] + sum[Y];
    if (val[X] < val[Y]) swap(x, y);
    dfs(y, x), addedge(x, y), addedge(y, x);
    ans += sum[root(x)];
}
void Initialize() {
    ans = 0;
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    memset(ts, 0, sizeof(ts));
    memset(td, 0, sizeof(td));
    memset(tag, 0, sizeof(tag));
}
} // namespace Heuristic_LCT
int n, m, x, y;
char op[5];
int main() {
    using namespace Heuristic_LCT;
    Initialize();
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) val[i] = size[i] = 1;
    while (m--) {
        scanf("%s", op);
        if (op[0] == 'A') scanf("%d%d", &x, &y), link(x, y);
        if (op[0] == 'Q') printf("%d\n", ans);
    }
    return 0;
}
```

2.40 欧拉游览树

```
/*
    ETT
    欧拉游览树(欧拉序+Splay维护)
    Q 查询子树和
    C 改变x的父节点为y
    F 增加子树的权值
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
typedef long long ll;
template <class T>
void rd(T &a) {
    a = 0;
    char c;
    while (c = getchar(), c < 48)
        ;
    do
        a = a * 10 + (c ^ 48);
    while (c = getchar(), c > 47);
}
inline void rdc(char &c) {
    c = 0;
    char s;
    while (s = getchar(), s < 65)
        ;
    c = s;
}
template <class T>
void nt(T x) {
    if (!x) return;
    nt(x / 10);
    putchar(48 + x % 10);
}
template <class T>
void pt(T x) {
    if (!x)
        putchar('0');
    else
        nt(x);
}
const int M = 2e5 + 10;
const int N = 1e5 + 2;
```

```
int f[N], w[N];
struct Edge {
    int to, nxt;
} G[N];
int tot_edge, head[N];
inline void add_edge(int from, int to) {
    G[tot_edge] = (Edge){to, head[from]};
    head[from] = tot_edge++;
}
struct Splay_Tree {
    struct node {
        int tr[2], fa, sz, pos;
        ll sum, add, val;
        inline node() { tr[0] = tr[1] = sum = add = val = fa = sz = pos = 0; }
    } T[M];
    int que[M], allc, rt;
    inline void up(int &x) {
        node &l = T[T[x].tr[0]], &r = T[T[x].tr[1]];
        T[x].sz = l.sz + r.sz + 1;
        T[x].sum = l.sum + r.sum + T[x].val;
        T[x].pos = l.pos + r.pos + (x <= N);
    }
    inline void down(int &x) {
        if (!T[x].add) return;
        ll &ad = T[x].add;
        T[x].val += x >= N ? -ad : ad;
        node &l = T[T[x].tr[0]], &r = T[T[x].tr[1]];
        l.add += T[x].add, r.add += T[x].add;
        int nagl = l.sz - l.pos, nagr = r.sz - r.pos;
        l.sum += 1ll * l.pos * ad - 1ll * nagl * ad;
        r.sum += 1ll * r.pos * ad - 1ll * nagr * ad;
        ad = 0;
    }
    inline void rotate(int x, int &k) {
        int y = T[x].fa, z = T[y].fa, r = T[y].tr[0] == x, l = r ^ 1;
        if (y == k)
            k = x;
        else {
            if (T[z].tr[0] == y)
                T[z].tr[0] = x;
            else
                T[z].tr[1] = x;
        }
        T[x].fa = z, T[y].fa = x, T[T[x].tr[r]].fa = y;
        T[y].tr[l] = T[x].tr[r], T[x].tr[r] = y;
        up(y);
```

```
}
inline void Splay(int x, int &k) {
    for (int v = x; v != rt; v = T[v].fa) que[++allc] = v;
    que[++allc] = rt;
    for (; allc;) down(que[allc--]);
    for (; x != k;) {
        int y = T[x].fa, z = T[y].fa;
        if (y != k) {
            if (T[z].tr[0] == y ^ T[y].tr[0] == x)
                rotate(x, k);
            else
                rotate(y, k);
        }
        rotate(x, k);
    }
    up(x);
}
inline void Insert(int x, int w) {
    T[x].val = w;
    if (!rt)
        rt = x;
    else
        T[rt].tr[1] = x, T[x].fa = rt, up(x), Splay(x, rt);
}
inline ll query(int &x) {
    Splay(x, rt);
    return T[T[x].tr[0]].sum + T[x].val;
}
inline int search(int x, const bool &dir) {
    if (!T[x].tr[dir]) return x;
    return search(T[x].tr[dir], dir);
}
inline void Change(int x, int y) {
    Splay(x, rt);
    int k1 = search(T[rt].tr[0], 1);
    Splay(x + N, rt);
    int k2 = search(T[rt].tr[1], 0);
    Splay(k1, rt), Splay(k2, T[rt].tr[1]);
    int w = T[k2].tr[0];
    T[k2].tr[0] = 0, T[w].fa = 0;
    Splay(y, rt);
    int k = search(T[y].tr[1], 0);
    Splay(k, T[y].tr[1]);
    T[w].fa = k, T[k].tr[0] = w;
}
inline void Add(int x, int w) {
```

```
        Splay(x, rt);
        int k1 = search(T[rt].tr[0], 1);
        Splay(x + N, rt);
        int k2 = search(T[rt].tr[1], 0);
        Splay(k1, rt), Splay(k2, T[rt].tr[1]);
        int s = T[k2].tr[0];
        T[s].add += w,
            T[s].sum += 1ll * w * T[s].pos - 1ll * (T[s].sz - T[s].pos) * w;
    }
} spt;
inline void dfs(int v) {
    spt.Insert(v + 1, w[v]);
    for (int i = head[v]; ~i; i = G[i].nxt) dfs(G[i].to);
    spt.Insert(v + N + 1, -w[v]);
}
int main() {
    int n, q, u, v;
    cin >> n;
    memset(head, -1, sizeof(head));
    tot_edge = 0;
    for (int i = 2; i <= n; ++i) rd(f[i]), add_edge(f[i], i);
    for (int i = 1; i <= n; ++i) rd(w[i]);
    spt.Insert(1, 0);
    dfs(1);
    spt.Insert(N << 1 | 1, 0);
    char c;
    for (cin >> q; q--;) {
        rdc(c);
        switch (c) {
            case ('Q'): // 查询子树和
                rd(v), ++v;
                pt(spt.query(v)), putchar('\n');
                break;
            case ('C'): // 把u的父亲改为v
                rd(u), rd(v), ++u, ++v;
                spt.Change(u, v);
                break;
            case ('F'): // 增加子树权值
                rd(u), rd(v);
                ++u;
                spt.Add(u, v);
                break;
        }
    }
    return 0;
}
```

2.41 TopTree

```
/*
    TopTree
    c[0],c[1]: LCT的左右儿子(维护实边)
    C[2],c[3]: Splay的左右儿子(维护虚边)
    链上信息: LCT维护
    子树信息: 以点x为根的子树的信息
    =x在链上的所有后代的信息+x在链上的所有后代连出去的虚边的信息+它本身的信息
    (在实际操作中其实只有后两个的信息)
    cha: 在Splay维护的实边形成的链中x的所有子节点(含本身)信息和
    sub: x的所有子节点(含本身)连出去的虚边的信息和(不包括链的信息)
    all: x的所有信息和,即cha+sub
    All: 子树修改标记(不包括链)
    Cha: 传统的链标记
    linkcut:
    cut(u): 把u的父亲access,然后del(u)
    link(u,v): 把v的父亲access,然后add(u,v->f)
    makert: 同LCT
*/
#include <bits/stdc++.h>
#define rep(i, a, n) for (int i = a; i < n; i++)
using namespace std;
const int maxn = 1e5 + 1000;
int getint() {
    int res = 0, f = 1;
    char c = getchar();
    while (!isdigit(c)) f = f == -1 || c == '-' ? -1 : 1, c = getchar();
    while (isdigit(c)) res = res * 10 + c - '0', c = getchar();
    return res * f;
}
int n, m;
struct info {
    int mx, mn, sum, sz;
    info() {}
    info(int mx, int mn, int sum, int sz) : mx(mx), mn(mn), sum(sum), sz(sz) {}
    void deb() { printf("sum:%d size:%d", (int)sum, sz); }
};
struct flag {
    int mul, add;
    flag() { mul = 1; }
    flag(int mul, int add) : mul(mul), add(add) {}
    bool empty() { return mul == 1 && add == 0; }
};
info operator+(const info &a, const flag &b) {
    return a.sz ? info(a.mx * b.mul + b.add, a.mn * b.mul + b.add,
```

```
                    a.sum * b.mul + b.add * a.sz, a.sz)
            : a;
}
info operator+(const info &a, const info &b) {
    return info(max(a.mx, b.mx), min(a.mn, b.mn), a.sum + b.sum, a.sz + b.sz);
}
flag operator+(const flag &a, const flag &b) {
    return flag(a.mul * b.mul, a.add * b.mul + b.add);
}
struct node {
    node *c[4], *f;
    flag Cha, All;
    info cha, sub, all;
    bool rev, inr;
    int val;
    void makerev() {
        rev ^= 1;
        swap(c[0], c[1]);
    }
    void makec(const flag &a) {
        Cha = Cha + a;
        cha = cha + a;
        val = val * a.mul + a.add;
        all = cha + sub;
    }
    void makes(const flag &a, bool _ = 1) {
        All = All + a;
        all = all + a;
        sub = sub + a;
        if (_) makec(a);
    }
    void rz() {
        cha = all = sub = info(-(1 << 30), 1 << 30, 0, 0);
        if (!inr) all = cha = info(val, val, val, 1);
        rep(i, 0, 2) if (c[i]) cha = cha + c[i]->cha, sub = sub + c[i]->sub;
        rep(i, 0, 4) if (c[i]) all = all + c[i]->all;
        rep(i, 2, 4) if (c[i]) sub = sub + c[i]->all;
    }
    void init(int _val) {
        rep(i, 0, 4) c[i] = 0;
        f = 0;
        All = Cha = flag(1, 0);
        sub = info(-(1 << 30), 1 << 30, 0, 0);
        inr = rev = 0;
        val = _val;
        all = cha = info(val, val, val, 1);
```

```
    }
    void pd() {
        if (rev) {
            if (c[0]) c[0]->makerev();
            if (c[1]) c[1]->makerev();
            rev = 0;
        }
        if (!All.empty()) {
            rep(i, 0, 4) if (c[i]) c[i]->makes(All, i >= 2);
            All = flag(1, 0);
        }
        if (!Cha.empty()) {
            rep(i, 0, 2) if (c[i]) c[i]->makec(Cha);
            Cha = flag(1, 0);
        }
    }
    node *C(int i) {
        if (c[i]) c[i]->pd();
        return c[i];
    }
    bool d(int ty) { return f->c[ty + 1] == this; }
    int D() { rep(i, 0, 4) if (f->c[i] == this) return i; }
    void sets(node *x, int d) {
        if (x) x->f = this;
        c[d] = x;
    }
    bool rt(int ty) {
        if (ty == 0)
            return !f || (f->c[0] != this && f->c[1] != this);
        else
            return !f || !f->inr || !inr;
    }
} nd[maxn * 2], *cur = nd + maxn, *pool[maxn], **Cur = pool;
int _cnt;
node *newnode() {
    _cnt++;
    node *x = (Cur == pool) ? cur++ : *(--Cur);
    rep(i, 0, 4) x->c[i] = 0;
    x->f = 0;
    x->All = x->Cha = flag(1, 0);
    x->all = x->cha = info(-(1 << 30), (1 << 30), 0, 0);
    x->inr = 1;
    x->rev = 0;
    x->val = 0;
    return x;
}
```

```
void dele(node *x) { *(Cur++) = x; }
void rot(node *x, int ty) {
    node *p = x->f;
    int d = x->d(ty);
    if (!p->f)
        x->f = 0;
    else
        p->f->sets(x, p->D());
    p->sets(x->c[!d + ty], d + ty);
    x->sets(p, !d + ty);
    p->rz();
}
node *splay(node *x, int ty = 0) {
    while (!x->rt(ty)) {
        if (x->f->rt(ty))
            rot(x, ty);
        else if (x->d(ty) == x->f->d(ty))
            rot(x->f, ty), rot(x, ty);
        else
            rot(x, ty), rot(x, ty);
    }
    x->rz();
    return x;
}
void add(node *u, node *w) {
    w->pd();
    rep(i, 2, 4) if (!w->c[i]) {
        w->sets(u, i);
        return;
    }
    node *x = newnode(), *v;
    for (v = w; v->c[2]->inr; v = v->C(2))
        ;
    x->sets(v->c[2], 2);
    x->sets(u, 3);
    v->sets(x, 2);
    splay(x, 2);
}
void del(node *w) {
    if (w->f->inr) {
        w->f->f->sets(w->f->c[5 - w->D()], w->f->D());
        dele(w->f);
        splay(w->f->f, 2);
    } else
        w->f->sets(0, w->D());
    w->f = 0;
```

```
}
void access(node *w) {
    static node *sta[maxn];
    static int top = 0;
    node *v = w, *u;
    for (u = w; u; u = u->f) sta[top++] = u;
    while (top) sta[--top]->pd();
    splay(w);
    if (w->c[1]) u = w->c[1], w->c[1] = 0, add(u, w), w->rz();
    while (w->f) {
        for (u = w->f; u->inr; u = u->f)
            ;
        splay(u);
        if (u->c[1])
            w->f->sets(u->c[1], w->D()), splay(w->f, 2);
        else
            del(w);
        u->sets(w, 1);
        (w = u)->rz();
    }
    splay(v);
}
void makert(node *x) {
    access(x);
    x->makerev();
}
node *findp(node *u) { // 找到原树上的父亲
    access(u);
    u = u->C(0);
    while (u && u->c[1]) u = u->C(1);
    return u;
}
node *findr(node *u) {
    for (; u->f; u = u->f)
        ;
    return u;
}
node *cut(node *u) {
    node *v = findp(u);
    if (v) access(v), del(u), v->rz();
    return v;
}
void link(node *u, node *v) {
    node *p = cut(u);
    if (findr(u) != findr(v)) p = v;
    if (p) access(p), add(u, p), p->rz();
```

```
}
int main() {
    n = getint();
    m = getint();
    static int _u[maxn], _v[maxn];
    rep(i, 1, n) _u[i] = getint(), _v[i] = getint();
    rep(i, 1, n + 1) nd[i].init(getint());
    rep(i, 1, n) makert(nd + _u[i]), link(nd + _u[i], nd + _v[i]);
    int root = getint();
    makert(nd + root);
    int x, y;
    node *u;
    while (m--) {
        int k = getint();
        x = getint();
        u = nd + x;
        if (k == 0 || k == 3 || k == 4 || k == 5 || k == 11) {
            access(u);
            if (k == 3 || k == 4 || k == 11) { // 子树查询
                int ans = u->val;
                rep(i, 2, 4) if (u->c[i]) {
                    info res = u->c[i]->all;
                    if (k == 3)
                        ans = min(ans, res.mn);
                    else if (k == 4)
                        ans = max(ans, res.mx);
                    else if (k == 11)
                        ans += res.sum;
                }
                printf("%d\n", ans);
            } else {
                y = getint();
                flag fg(k == 5, y); // 子树加法/子树赋值
                u->val = u->val * fg.mul + fg.add;
                rep(i, 2, 4) if (u->c[i]) u->c[i]->makes(fg);
                u->rz();
            }
        } else if (k == 2 || k == 6 || k == 7 || k == 8 || k == 10) {
            y = getint();
            makert(u), access(nd + y), splay(u);
            if (k == 7 || k == 8 || k == 10) { // 链查询
                info ans = u->cha;
                if (k == 7)
                    printf("%d\n", ans.mn);
                else if (k == 8)
                    printf("%d\n", ans.mx);
```

```
            else
                printf("%d\n", ans.sum);
        } else
            u->makec(flag(k == 6, getint())); // 链加法/链查询
        makert(nd + root);
    } else if (k == 9)
        link(u, nd + getint()); // x的父节点变成y
    else if (k == 1)
        makert(u), root = x; // 换根
  }
  return 0;
}
```

2.42 Treap

```
/*
   Treap
*/
namespace Treap {
struct node {
   int val, cnt, size, p, id;
   node *l, *r;
   node() {}
   node(int val, int id) : val(val), id(id) {
      p = rand();
      cnt = size = 1;
      l = r = NULL;
   }
   void up() {
      size = cnt;
      if (l) size += l->size;
      if (r) size += r->size;
   }
} * root[N], *pool[N];
int top;
node* new_node() { return pool[top++]; }
void Initialize() {
   top = 0;
   for (int i = 0; i < N; i++) pool[i] = new node();
}
void Rotatel(node*& x) {
   node* y = x->r;
   x->r = y->l;
   x->up();
   y->l = x;
```

```
    y->up();
    x = y;
}
void Rotater(node*& x) {
    node* y = x->l;
    x->l = y->r;
    x->up();
    y->r = x;
    y->up();
    x = y;
}
//往treap上插入点
void Insert(node*& x, node y) {
    if (!x) {
        x = new_node();
        (*x) = y;
        return;
    }
    x->size++;
    if (y.val == x->val) {
        x->cnt++;
        return;
    }
    if (y.val < x->val) {
        Insert(x->l, y);
        if (x->l->p > x->p) Rotater(x);
    } else {
        Insert(x->r, y);
        if (x->r->p > x->p) Rotatel(x);
    }
}
// 查找第k小的元素
int kth(node* x, int rnk) {
    while (x) {
        int d = x->l ? x->l->size : 0;
        if (rnk <= d)
            x = x->l;
        else if (rnk > d + x->cnt)
            rnk -= d + x->cnt, x = x->r;
        else
            return x->id;
    }
    return -1;
}
// 删除一个p
void Delete(node*& x, int p) {
```

```
    x->size--;
    if (p == x->val) {
        x->cnt--;
        return;
    }
    if (p < x->val)
        Delete(x->l, p);
    else
        Delete(x->r, p);
}
// 查询大于p的数字的个数
int Ask(node* x, int p) {
    if (!x) return 0;
    if (p == x->val) return x->r->size;
    if (p < x->val) return x->cnt + x->r->size + Ask(x->l, p);
    return Ask(x->r, p);
}
// 查询在[c,d]范围内的数字的个数
int Ask(node* x, int a, int b, int c, int d) {
    if (!x) return 0;
    if (c <= a && b <= d) return x->size;
    int t = c <= x->val && x->val <= d ? x->cnt : 0;
    if (c < x->val) t += Ask(x->l, a, x->val - 1, c, d);
    if (d > x->val) t += Ask(x->r, x->val + 1, b, c, d);
}
void Del_node(node*& u) {
    pool[--top] = u;
    u = NULL;
}
} // namespace Treap
```

2.43 启发式合并 Treap

```
/*
    启发式合并Treap
*/
namespace Treap {
struct node {
    int val, cnt, size, p, id;
    node *l, *r;
    node() {}
    node(int val, int id) : val(val), id(id) {
        p = rand();
        cnt = size = 1;
        l = r = NULL;
```

```
    }
    void up() {
        size = cnt;
        if (l) size += l->size;
        if (r) size += r->size;
    }
} * root[N], *pool[N];
int top;
node* new_node() { return pool[top++]; }
void Initialize() {
    top = 0;
    for (int i = 0; i < N; i++) pool[i] = new node();
}
void Rotatel(node*& x) {
    node* y = x->r;
    x->r = y->l;
    x->up();
    y->l = x;
    y->up();
    x = y;
}
void Rotater(node*& x) {
    node* y = x->l;
    x->l = y->r;
    x->up();
    y->r = x;
    y->up();
    x = y;
}
//往treap上插入点
void Insert(node*& x, node y) {
    if (!x) {
        x = new_node();
        (*x) = y;
        return;
    }
    x->size++;
    if (y.val == x->val) {
        x->cnt++;
        return;
    }
    if (y.val < x->val) {
        Insert(x->l, y);
        if (x->l->p > x->p) Rotater(x);
    } else {
        Insert(x->r, y);
```

```
        if (x->r->p > x->p) Rotatel(x);
    }
}
// 查找第k小的元素
int kth(node* x, int rnk) {
    while (x) {
        int d = x->l ? x->l->size : 0;
        if (rnk <= d)
            x = x->l;
        else if (rnk > d + x->cnt)
            rnk -= d + x->cnt, x = x->r;
        else
            return x->id;
    }
    return -1;
}
// 删除一个p
void Delete(node*& x, int p) {
    x->size--;
    if (p == x->val) {
        x->cnt--;
        return;
    }
    if (p < x->val)
        Delete(x->l, p);
    else
        Delete(x->r, p);
}
// 查询大于p的数字的个数
int Ask(node* x, int p) {
    if (!x) return 0;
    if (p == x->val) return x->r->size;
    if (p < x->val) return x->cnt + x->r->size + Ask(x->l, p);
    return Ask(x->r, p);
}
// 查询在[c,d]范围内的数字的个数
int Ask(node* x, int a, int b, int c, int d) {
    if (!x) return 0;
    if (c <= a && b <= d) return x->size;
    int t = c <= x->val && x->val <= d ? x->cnt : 0;
    if (c < x->val) t += Ask(x->l, a, x->val - 1, c, d);
    if (d > x->val) t += Ask(x->r, x->val + 1, b, c, d);
}
void Del_node(node*& u) {
    pool[--top] = u;
    u = NULL;
```

```
}
// 将B树并入A树
void merge(node*& A, node*& B) {
    if (!B) return;
    if (B->l) merge(A, B->l);
    if (B->r) merge(A, B->r);
    Insert(A, *B);
    Del_node(B);
}
} // namespace Treap
int n, m, k, u, v, q;
namespace Union_Find_Set {
int f[M], size[M], block;
void Initialize() {
    memset(f, 0, sizeof(f));
    block = n;
}
int Find(int x) {
    if (!f[x]) f[x] = x, size[x] = 1;
    if (f[x] == x) return x;
    return f[x] = Find(f[x]);
}
void Union(int x, int y) {
    x = Find(x);
    y = Find(y);
    if (x == y) return;
    if (size[x] > size[y]) swap(x, y);
    f[x] = y;
    size[y] += size[x];
    block--;
}
} // namespace Union_Find_Set
void Heuristic_merge(int x, int y) {
    int fx = Union_Find_Set::Find(x);
    int fy = Union_Find_Set::Find(y);
    if (fx == fy) return;
    if (Treap::root[fx]->size < Treap::root[fy]->size) swap(x, y), swap(fx, fy);
    Treap::merge(Treap::root[fx], Treap::root[fy]);
    Union_Find_Set::f[fy] = fx;
}
/*
    Test
    有连边操作和查询操作，每个点有其重要度
    查询要求找出连通块中重要度第k大的点的id
*/
int main() {
```

```
    using namespace Treap;
    Initialize();
    Union_Find_Set::Initialize();
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &k);
        Union_Find_Set::f[i] = i;
        Insert(root[i], node(k, i));
    }
    while (m--) {
        scanf("%d%d", &u, &v);
        Heuristic_merge(u, v);
    }
    scanf("%d", &q);
    char op[5];
    while (q--) {
        scanf("%s", op);
        if (op[0] == 'B') {
            scanf("%d%d", &u, &v);
            Heuristic_merge(u, v);
        } else {
            scanf("%d%d", &u, &v);
            printf("%d\n", kth(root[Union_Find_Set::Find(u)], v));
        }
    }
    return 0;
}
```

2.44 KD 树

```
/*
    KD树
    支持暴力重构
    初始化只需要把dcnt置0
    KD_Tree::Insert(root,0,KD_Tree::Dot(x0,y0,c))
    if(KD_Tree::dcnt%5000==0)root=KD_Tree::Rebuild(1,KD_Tree::dcnt,0)
    KD_Tree::query(root,x0,y0,x1,y1)
    Attention: 有些数据情况，直接插入点不重构效率更高
*/
namespace KD_Tree {
struct Dot {
    int d[2], mn[2], mx[2], l, r, sz, sum, val;
    Dot() { l = r = 0; }
    Dot(int x, int y, int c) {
        d[0] = x;
```

```
        d[1] = y;
        val = c;
        l = r = sz = sum = 0;
    }
    int& operator[](int x) { return d[x]; }
};
int D, dcnt = 0, pt[N];
Dot T[N];
bool operator<(Dot a, Dot b) { return a[D] < b[D]; }
inline void umax(int& a, int b) {
    if (a < b) a = b;
}
inline void umin(int& a, int b) {
    if (a > b) a = b;
}
inline bool cmp(int x, int y) { return T[x][D] < T[y][D]; }
inline void up(int x) {
    T[x].sz = T[T[x].l].sz + T[T[x].r].sz + 1;
    T[x].sum = T[T[x].l].sum + T[T[x].r].sum + T[x].val;
    T[x].mn[0] = T[x].mx[0] = T[x][0];
    T[x].mn[1] = T[x].mx[1] = T[x][1];
    if (T[x].l) {
        umax(T[x].mx[0], T[T[x].l].mx[0]);
        umin(T[x].mn[0], T[T[x].l].mn[0]);
        umax(T[x].mx[1], T[T[x].l].mx[1]);
        umin(T[x].mn[1], T[T[x].l].mn[1]);
    }
    if (T[x].r) {
        umax(T[x].mx[0], T[T[x].r].mx[0]);
        umin(T[x].mn[0], T[T[x].r].mn[0]);
        umax(T[x].mx[1], T[T[x].r].mx[1]);
        umin(T[x].mn[1], T[T[x].r].mn[1]);
    }
}
// NewDot中所有用到的参数一定要初始化
inline int NewDot(int x, int y, int c) {
    ++dcnt;
    pt[dcnt] = dcnt;
    T[dcnt][0] = x;
    T[dcnt][1] = y;
    T[dcnt].val = c;
    T[dcnt].l = T[dcnt].r = T[dcnt].sz = 0;
    return up(dcnt), dcnt;
}
// 离线建树时，只加点，不构树，最后build
inline void AddDot(int x, int y, int c) {
```

```
    ++dcnt;
    pt[dcnt] = dcnt;
    p[dcnt][0] = x;
    p[dcnt][1] = y;
    p[dcnt].val = c;
}
// 查询矩阵内数字和
int query(int x, int x0, int y0, int x1, int y1) {
    if (!x || T[x].mn[0] > x1 || T[x].mx[0] < x0 || T[x].mn[1] > y1 ||
        T[x].mx[1] < y0)
        return 0;
    if (T[x].mn[0] >= x0 && T[x].mx[0] <= x1 && T[x].mn[1] >= y0 &&
        T[x].mx[1] <= y1)
        return T[x].sum;
    int res = 0;
    if (T[x][0] >= x0 && T[x][0] <= x1 && T[x][1] >= y0 && T[x][1] <= y1)
        res += T[x].val;
    return res + query(T[x].l, x0, y0, x1, y1) + query(T[x].r, x0, y0, x1, y1);
}
LL sqr(LL x) { return x * x; }
// 欧几里得上界估价函数（K远点对）
inline LL dist(int x, int px, int py) {
    LL dis = 0;
    if (!x) return 0;
    dis += max(sqr(T[x].mn[0] - px), sqr(T[x].mx[0] - px));
    dis += max(sqr(T[x].mn[1] - py), sqr(T[x].mx[1] - py));
    return dis;
}
} // namespace KD_Tree
// 曼哈顿距离估价函数
inline int dist(int p1, int px, int py) {
    int dis = 0;
    if (px < T[p1].mn[0]) dis += T[p1].mn[0] - px;
    if (px > T[p1].mx[0]) dis += px - T[p1].mx[0];
    if (py < T[p1].mn[1]) dis += T[p1].mn[1] - py;
    if (py > T[p1].mx[1]) dis += py - T[p1].mx[1];
    return dis;
}
// 欧几里得距离估价函数
inline LL dist(int p1, int px, int py) {
    LL dis = 0;
    if (px < T[p1].mn[0]) dis += sqr(T[p1].mn[0] - px);
    if (px > T[p1].mx[0]) dis += sqr(px - T[p1].mx[0]);
    if (py < T[p1].mn[1]) dis += sqr(T[p1].mn[1] - py);
    if (py > T[p1].mx[1]) dis += sqr(py - T[p1].mx[1]);
    return dis;
```

```
}
// 查询(px,py)最近点距离（曼哈顿距离为例）
int ans = 0;
inline void ask(int x, int px, int py) {
    int dl, dr, d0 = abs(T[x][0] - px) + abs(T[x][1] - py);
    if (d0 < ans) ans = d0;
    dl = T[x].l ? dist(T[x].l, px, py) : INF;
    dr = T[x].r ? dist(T[x].r, px, py) : INF;
    if (dl < dr) {
        if (dl < ans) ask(T[x].l, px, py);
        if (dr < ans) ask(T[x].r, px, py);
    } else {
        if (dr < ans) ask(T[x].r, px, py);
        if (dl < ans) ask(T[x].l, px, py);
    }
}
int Querymin(int x, int px, int py) {
    ans = INF;
    ask(x, px, py);
    return ans;
}
// 查询离树上的某点距离最近的点的id（欧式距离为例）
LL ans, ans2;
Dot ansP, ansP2;
inline void ask(int x, int px, int py) {
    LL dl, dr, d0 = sqr(T[x][0] - px) + sqr(T[x][1] - py);
    if (d0 < ans || (d0 == ans && T[x] < ansP))
        ans2 = ans, ansP2 = ansP, ans = d0, ansP = T[x];
    else if (d0 < ans2 || (d0 == ans2 && T[x] < ansP2))
        ans2 = d0, ansP2 = T[x];
    dl = T[x].l ? dist(T[x].l, px, py) : INF;
    dr = T[x].r ? dist(T[x].r, px, py) : INF;
    if (dl < dr) {
        if (dl <= ans2) ask(T[x].l, px, py);
        if (dr <= ans2) ask(T[x].r, px, py);
    } else {
        if (dr <= ans2) ask(T[x].r, px, py);
        if (dl <= ans2) ask(T[x].l, px, py);
    }
}
int getP(int x, int px, int py) {
    ans = ans2 = INF;
    ask(root);
    return ansP2.id;
}
// 在线插入元素
```

```
void Insert(int& x, int D, const Dot& p) {
    if (!x) {
        x = NewDot(p.d[0], p.d[1], p.val);
        return;
    }
    if (p.d[D] < T[x][D])
        Insert(T[x].l, D ^ 1, p);
    else
        Insert(T[x].r, D ^ 1, p);
    up(x);
}
// 暴力重构
int Rebuild(int l, int r, int now) {
    if (l > r) return 0;
    int mid = (l + r) >> 1, x;
    D = now;
    nth_element(pt + l, pt + mid, pt + r + 1, cmp);
    x = pt[mid];
    T[x].l = Rebuild(l, mid - 1, now ^ 1);
    T[x].r = Rebuild(mid + 1, r, now ^ 1);
    return up(x), x;
}
// 建树
int build(int l, int r, int now) {
    int mid = (l + r) >> 1;
    D = now;
    nth_element(p + l, p + mid, p + r + 1);
    T[mid] = p[mid];
    for (int i = 0; i < 2; i++) T[mid].mn[i] = T[mid].mx[i] = T[mid][i];
    if (l < mid) T[mid].l = build(l, mid - 1, now ^ 1);
    if (r > mid) T[mid].r = build(mid + 1, r, now ^ 1);
    return up(mid), mid;
}
}
/*
    估价函数:
    欧几里得距离下界:
    sqr(max(max(X-x.Max[0],x.Min[0]-X),0))+sqr(max(max(Y-x.Max[1],x.Min[1]-Y),0))
    曼哈顿距离下界:
    max(x.Min[0]-X,0)+max(X-x.Max[0],0)+max(x.Min[1]-Y,0)+max(Y-x.Max[1],0)
    欧几里得距离上界:
    max(sqr(X-x.Min[0]),sqr(X-x.Max[0]))+max(sqr(Y-x.Min[1]),sqr(Y-x.Max[1])
    曼哈顿距离上界:
    max(abs(X-x.Max[0]),abs(x.Min[0]-X))+max(abs(Y-x.Max[1]),abs(x.Min[1]-Y))
*/
/*
```

```
    Example:
    给出一些点，给出与x轴交点为LA，与y轴交点为LB的直线，问有多少点在这条直线上
    因为点随机生成，所以我们对于给出的点建立KD树，
    每次查询直线时，若查询直线与当前点子树的最小包围矩形不相交，则剪枝
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int ans;
namespace KD_Tree {
struct Dot {
    int d[2], mn[2], mx[2], l, r, sz, sum, val;
    Dot() { l = r = 0; }
    Dot(int x, int y, int c) {
        d[0] = x;
        d[1] = y;
        val = c;
        l = r = sz = sum = 0;
    }
    int& operator[](int x) { return d[x]; }
};
int D, dcnt = 0, pt[N];
Dot T[N], p[N];
bool operator<(Dot a, Dot b) { return a[D] < b[D]; }
inline void umax(int& a, int b) {
    if (a < b) a = b;
}
inline void umin(int& a, int b) {
    if (a > b) a = b;
}
inline bool cmp(int x, int y) { return T[x][D] < T[y][D]; }
inline void up(int x) {
    T[x].sz = T[T[x].l].sz + T[T[x].r].sz + 1;
    T[x].sum = T[T[x].l].sum + T[T[x].r].sum + T[x].val;
    T[x].mn[0] = T[x].mx[0] = T[x][0];
    T[x].mn[1] = T[x].mx[1] = T[x][1];
    if (T[x].l) {
        umax(T[x].mx[0], T[T[x].l].mx[0]);
        umin(T[x].mn[0], T[T[x].l].mn[0]);
        umax(T[x].mx[1], T[T[x].l].mx[1]);
        umin(T[x].mn[1], T[T[x].l].mn[1]);
    }
    if (T[x].r) {
        umax(T[x].mx[0], T[T[x].r].mx[0]);
        umin(T[x].mn[0], T[T[x].r].mn[0]);
```

```
        umax(T[x].mx[1], T[T[x].r].mx[1]);
        umin(T[x].mn[1], T[T[x].r].mn[1]);
    }
}
inline int NewDot(int x, int y, int c) {
    ++dcnt;
    pt[dcnt] = dcnt;
    T[dcnt][0] = x;
    T[dcnt][1] = y;
    T[dcnt].val = c;
    return up(dcnt), dcnt;
}
inline void AddDot(int x, int y, int c) {
    ++dcnt;
    pt[dcnt] = dcnt;
    p[dcnt][0] = x;
    p[dcnt][1] = y;
    p[dcnt].val = c;
}
int build(int l, int r, int now) {
    int mid = (l + r) >> 1;
    D = now;
    nth_element(p + l, p + mid, p + r + 1);
    T[mid] = p[mid];
    for (int i = 0; i < 2; i++) T[mid].mn[i] = T[mid].mx[i] = T[mid][i];
    if (l < mid) T[mid].l = build(l, mid - 1, now ^ 1);
    if (r > mid) T[mid].r = build(mid + 1, r, now ^ 1);
    return up(mid), mid;
}
long long LA, LB, LC;
bool check(int xl, int xr, int yl, int yr) {
    long long t = -LB * xl + LC;
    if (LA * yl <= t && t <= LA * yr) return 1;
    t = -LB * xr + LC;
    if (LA * yl <= t && t <= LA * yr) return 1;
    t = -LA * yl + LC;
    if (LB * xl <= t && t <= LB * xr) return 1;
    t = -LA * yr + LC;
    if (LB * xl <= t && t <= LB * xr) return 1;
    return 0;
}
void ask(int x) {
    if (!check(T[x].mn[0], T[x].mx[0], T[x].mn[1], T[x].mx[1])) return;
    if (LB * T[x].d[0] + LA * T[x].d[1] == LC) ans++;
    if (T[x].l) ask(T[x].l);
    if (T[x].r) ask(T[x].r);
```

```
}
} // namespace KD_Tree
int T, root, n, m;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        KD_Tree::dcnt = 0;
        for (int i = 1; i <= n; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            KD_Tree::AddDot(x, y, 1);
        }
        int root = KD_Tree::build(1, n, 0);
        while (m--) {
            ans = 0;
            scanf("%lld%lld", &KD_Tree::LA, &KD_Tree::LB);
            KD_Tree::LC = KD_Tree::LA * KD_Tree::LB;
            KD_Tree::ask(root);
            printf("%d\n", ans);
        }
    }
    return 0;
}
/*
    Example:
    求K远点对距离
    修改估价函数为欧式上界估价，对每个点进行dfs，
    因为是无向点对，在小根堆中保留前2k个距离，
    不断更新堆顶元素即可。
*/
#include <algorithm>
#include <cstdio>
#include <queue>
using namespace std;
typedef long long LL;
const int N = 200000;
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;
}
struct data {
    LL dis;
    data(){};
    data(const LL& x) { dis = x; }
    bool operator<(const data& x) const { return dis > x.dis; }
} tmp;
priority_queue<data> q; // 小根堆
namespace KD_Tree {
struct Dot {
    int d[2], mn[2], mx[2], l, r;
    Dot() { l = r = 0; }
    Dot(int x, int y) {
        d[0] = x;
        d[1] = y;
        l = r = 0;
    }
    int& operator[](int x) { return d[x]; }
};
int D, dcnt = 0, pt[N];
Dot T[N], p[N];
bool operator<(Dot a, Dot b) { return a[D] < b[D]; }
inline void umax(int& a, int b) {
    if (a < b) a = b;
}
inline void umin(int& a, int b) {
    if (a > b) a = b;
}
inline bool cmp(int x, int y) { return T[x][D] < T[y][D]; }
inline void up(int x) {
    T[x].mn[0] = T[x].mx[0] = T[x][0];
    T[x].mn[1] = T[x].mx[1] = T[x][1];
    if (T[x].l) {
        umax(T[x].mx[0], T[T[x].l].mx[0]);
        umin(T[x].mn[0], T[T[x].l].mn[0]);
        umax(T[x].mx[1], T[T[x].l].mx[1]);
        umin(T[x].mn[1], T[T[x].l].mn[1]);
    }
    if (T[x].r) {
        umax(T[x].mx[0], T[T[x].r].mx[0]);
        umin(T[x].mn[0], T[T[x].r].mn[0]);
        umax(T[x].mx[1], T[T[x].r].mx[1]);
        umin(T[x].mn[1], T[T[x].r].mn[1]);
```

```
    }
}
void AddDot(int x, int y) {
    ++dcnt;
    pt[dcnt] = dcnt;
    p[dcnt][0] = x;
    p[dcnt][1] = y;
}
// 建树
int build(int l, int r, int now) {
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    D = now;
    nth_element(p + l, p + mid, p + r + 1);
    T[mid] = p[mid];
    for (int i = 0; i < 2; i++) T[mid].mn[i] = T[mid].mx[i] = T[mid][i];
    if (l < mid) T[mid].l = build(l, mid - 1, now ^ 1);
    if (r > mid) T[mid].r = build(mid + 1, r, now ^ 1);
    return up(mid), mid;
}
LL sqr(LL x) { return x * x; }
//欧几里得上界估价函数
inline LL dist(int x, int px, int py) {
    LL dis = 0;
    if (!x) return 0;
    dis += max(sqr(T[x].mn[0] - px), sqr(T[x].mx[0] - px));
    dis += max(sqr(T[x].mn[1] - py), sqr(T[x].mx[1] - py));
    return dis;
}
} // namespace KD_Tree
void query(int x, int px, int py) {
    if (!x) return;
    LL dl, dr, d0 = sqr(T[x][0] - px) + sqr(T[x][1] - py);
    dl = T[x].l ? dist(T[x].l, px, py) : 0;
    dr = T[x].r ? dist(T[x].r, px, py) : 0;
    if (d0 > q.top().dis) q.pop(), q.push(data(d0));
    if (dl > dr) {
        if (dl > q.top().dis) query(T[x].l, px, py);
        if (dr > q.top().dis) query(T[x].r, px, py);
    } else {
        if (dr > q.top().dis) query(T[x].r, px, py);
        if (dl > q.top().dis) query(T[x].l, px, py);
    }
}
}
int n, k;
```

```
int main() {
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++) KD_Tree::AddDot(read(), read());
    int root = KD_Tree::build(1, n, 0);
    for (int i = 0; i < k + k; i++) q.push(tmp);
    for (int i = 1; i <= n; i++)
        KD_Tree::query(root, KD_Tree::p[i][0], KD_Tree::p[i][1]);
    printf("%lld\n", q.top().dis);
    return 0;
}
```

2.45 替罪羊式 KD 树

```
/*
    替罪羊式KD树
    初始化只需要把dcnt置0
    支持在线动态插入查询矩阵中数字和，矩阵中点的个数，最近点距离
    Insert(root[i],p) 向第i棵KD树中插入节点p，支持多棵
    一开始将root[i]置0传入即可
*/
namespace KD_Tree {
struct Dot {
    int d[2], mn[2], mx[2], l, r, sz, sum, val;
    Dot() { l = r = 0; }
    Dot(int x, int y, int c) {
        d[0] = x;
        d[1] = y;
        val = c;
        l = r = sz = sum = 0;
    }
    int& operator[](int x) { return d[x]; }
};
int D, dcnt = 0; // 维度,点数
Dot T[N];
inline void umax(int& a, int b) {
    if (a < b) a = b;
}
inline void umin(int& a, int b) {
    if (a > b) a = b;
}
inline bool cmp(int x, int y) { return T[x][D] < T[y][D]; }
inline void up(int x) {
    T[x].sz = T[T[x].l].sz + T[T[x].r].sz + 1;
    T[x].sum = T[T[x].l].sum + T[T[x].r].sum + T[x].val;
    T[x].mn[0] = T[x].mx[0] = T[x][0];
```

```
    T[x].mn[1] = T[x].mx[1] = T[x][1];
    if (T[x].l) {
        umax(T[x].mx[0], T[T[x].l].mx[0]);
        umin(T[x].mn[0], T[T[x].l].mn[0]);
        umax(T[x].mx[1], T[T[x].l].mx[1]);
        umin(T[x].mn[1], T[T[x].l].mn[1]);
    }
    if (T[x].r) {
        umax(T[x].mx[0], T[T[x].r].mx[0]);
        umin(T[x].mn[0], T[T[x].r].mn[0]);
        umax(T[x].mx[1], T[T[x].r].mx[1]);
        umin(T[x].mn[1], T[T[x].r].mn[1]);
    }
}
// NewDot中所有用到的参数一定要初始化
inline int NewDot(int x, int y, int c) {
    ++dcnt;
    T[dcnt][0] = x;
    T[dcnt][1] = y;
    T[dcnt].val = c;
    T[dcnt].l = T[dcnt].r = T[dcnt].sz = 0;
    return up(dcnt), dcnt;
}
// 查询矩阵内数字和
int Query(int x, int x0, int y0, int x1, int y1) {
    if (!x || T[x].mn[0] > x1 || T[x].mx[0] < x0 || T[x].mn[1] > y1 ||
        T[x].mx[1] < y0)
        return 0;
    if (T[x].mn[0] >= x0 && T[x].mx[0] <= x1 && T[x].mn[1] >= y0 &&
        T[x].mx[1] <= y1)
        return T[x].sum;
    int res = 0;
    if (T[x][0] >= x0 && T[x][0] <= x1 && T[x][1] >= y0 && T[x][1] <= y1)
        res += T[x].val;
    return res + Query(T[x].l, x0, y0, x1, y1) + Query(T[x].r, x0, y0, x1, y1);
}
// 查询矩阵内点的个数
int query(int x, int x0, int y0, int x1, int y1) {
    if (!x || T[x].mn[0] > x1 || T[x].mx[0] < x0 || T[x].mn[1] > y1 ||
        T[x].mx[1] < y0)
        return 0;
    if (T[x].mn[0] >= x0 && T[x].mx[0] <= x1 && T[x].mn[1] >= y0 &&
        T[x].mx[1] <= y1)
        return T[x].sz;
    int res = 0;
    if (T[x][0] >= x0 && T[x][0] <= x1 && T[x][1] >= y0 && T[x][1] <= y1) res++;
```

```
    return res + query(T[x].l, x0, y0, x1, y1) + query(T[x].r, x0, y0, x1, y1);
}
// 曼哈顿距离估价函数
inline int dist(int p1, int px, int py) {
    int dis = 0;
    if (px < T[p1].mn[0]) dis += T[p1].mn[0] - px;
    if (px > T[p1].mx[0]) dis += px - T[p1].mx[0];
    if (py < T[p1].mn[1]) dis += T[p1].mn[1] - py;
    if (py > T[p1].mx[1]) dis += py - T[p1].mx[1];
    return dis;
}
//欧几里得距离估价函数
inline LL dist(int p1, int px, int py) {
    LL dis = 0;
    if (px < T[p1].mn[0]) dis += sqr(T[p1].mn[0] - px);
    if (px > T[p1].mx[0]) dis += sqr(px - T[p1].mx[0]);
    if (py < T[p1].mn[1]) dis += sqr(T[p1].mn[1] - py);
    if (py > T[p1].mx[1]) dis += sqr(py - T[p1].mx[1]);
    return dis;
}
// 查询(px,py)最近点距离
int ans = 0;
inline void ask(int x, int px, int py) {
    int dl, dr, d0 = abs(T[x][0] - px) + abs(T[x][1] - py);
    if (d0 < ans) ans = d0;
    dl = T[x].l ? dist(T[x].l, px, py) : INF;
    dr = T[x].r ? dist(T[x].r, px, py) : INF;
    if (dl < dr) {
        if (dl < ans) ask(T[x].l, px, py);
        if (dr < ans) ask(T[x].r, px, py);
    } else {
        if (dr < ans) ask(T[x].r, px, py);
        if (dl < ans) ask(T[x].l, px, py);
    }
}
int Querymin(int x, int px, int py) {
    ans = INF;
    ask(x, px, py);
    return ans;
}
// 替罪羊式重构
int tot = 0, pt[N];
int gt, gtd, gtf;
const double alp = 0.8;
inline bool isbad(int x) {
    return max(T[T[x].l].sz, T[T[x].r].sz) > T[x].sz * alp + 5;
```

```
}
void ins(int& x, int D, const Dot& p) {
    if (!x) {
        x = NewDot(p.d[0], p.d[1], p.val);
        return;
    }
    if (p.d[D] < T[x][D])
        ins(T[x].l, D ^ 1, p);
    else
        ins(T[x].r, D ^ 1, p);
    up(x);
    if (isbad(x))
        gt = x, gtd = D, gtf = 0;
    else if (gt == T[x].l || gt == T[x].r)
        gtf = x;
}
void treavel(int& x) {
    if (!x) return;
    pt[++tot] = x;
    treavel(T[x].l), treavel(T[x].r);
}
int build(int l, int r, int now) {
    if (l > r) return 0;
    int mid = (l + r) >> 1, x;
    D = now;
    nth_element(pt + l, pt + mid, pt + r + 1, cmp);
    x = pt[mid];
    T[x].l = build(l, mid - 1, now ^ 1);
    T[x].r = build(mid + 1, r, now ^ 1);
    return up(x), x;
}
void Insert(int& x, const Dot& p) {
    gt = gtf = 0, ins(x, 0, p);
    if (!gt) return;
    tot = 0, treavel(gt);
    if (!gtf) {
        x = build(1, tot, gtd);
        return;
    }
    if (gt == T[gtf].l)
        T[gtf].l = build(1, tot, gtd);
    else
        T[gtf].r = build(1, tot, gtd);
}
} // namespace KD_Tree
```

2.46 字典树合并

```
/*
    Problem:
        给出一棵点权树，每次查询一棵子树中选择一个数字与给定数异或的最大值
    Solution:
        用字典树维护每个子树的数集，按dfs序合并子树字典树，在根处查询每个询问
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
const int D = 31;
int T[(D + 2) * N][2], root[N], tot, Q[N], ans[N], a[N];
vector<int> v[N], id[N];
void init() { tot = 0; }
int newnode() {
    ++tot;
    T[tot][0] = T[tot][1] = 0;
    return tot;
}
void insert(int rt, int x) {
    for (int i = D; i >= 0; i--) {
        int idx = (x >> i) & 1;
        if (!T[rt][idx]) T[rt][idx] = newnode();
        rt = T[rt][idx];
    }
}
int query(int rt, int x) {
    int res = 0;
    for (int i = D; i >= 0; i--) {
        int idx = (x >> i) & 1;
        if (T[rt][idx ^ 1]) {
            rt = T[rt][idx ^ 1];
            res |= (1 << i);
        } else
            rt = T[rt][idx];
    }
    return res;
}
int merge(int x, int y) {
    if (!x || !y) return x ^ y;
    T[x][0] = merge(T[x][0], T[y][0]);
    T[x][1] = merge(T[x][1], T[y][1]);
    return x;
}
void dfs(int x, int fx) {
```

```
    root[x] = newnode();
    insert(root[x], a[x]);
    for (auto &y : v[x]) {
        if (y == fx) continue;
        dfs(y, x);
        root[x] = merge(root[x], root[y]);
    }
    for (auto &idx : id[x]) ans[idx] = query(root[x], Q[idx]);
}
int n, q, x;
int main() {
    while (~scanf("%d%d", &n, &q)) {
        init();
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            v[i].clear();
            id[i].clear();
        }
        for (int i = 2; i <= n; i++) {
            scanf("%d", &x);
            v[x].push_back(i);
        }
        for (int i = 1; i <= q; i++) {
            scanf("%d%d", &x, &Q[i]);
            id[x].push_back(i);
        }
        dfs(1, 1);
        for (int i = 1; i <= q; i++) printf("%d\n", ans[i]);
    }
    return 0;
}
/*
    Problem:
        给出一棵n个点的边权树，边权值从0到n-1各不相同，定义A值为一条链上的最大值，B值为一条链的异或值
        问不同的A值对应的最大B值为多少
    Solution:
        我们从小到大枚举每条边，问题就转化为将这条边两边连通，计算从这条边两边的连通块中各选出一个点，
        连起来得到的链的异或值的最大值，我们通过dfs计算每个点到定根的xor值，记为a，
        那么a[x]^a[y]就表示x到y之间的异或值，所以我们只要求两个集合中各选出一个数字异或和最大即可，
        我们将小的集合中的元素在大的集合中查询，启发式合并字典树即可
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
const int D = 31;
int T[(D + 2) * N][2], root[N], tot;
```

```
void init() { tot = 0; }
int newnode() {
    ++tot;
    T[tot][0] = T[tot][1] = 0;
    return tot;
}
void insert(int rt, int x) {
    for (int i = D; i >= 0; i--) {
        int idx = (x >> i) & 1;
        if (!T[rt][idx]) T[rt][idx] = newnode();
        rt = T[rt][idx];
    }
}
int query(int rt, int x) {
    int res = 0;
    for (int i = D; i >= 0; i--) {
        int idx = (x >> i) & 1;
        if (T[rt][idx ^ 1]) {
            rt = T[rt][idx ^ 1];
            res |= (1 << i);
        } else
            rt = T[rt][idx];
    }
    return res;
}
int merge(int x, int y) {
    if (!x || !y) return x ^ y;
    T[x][0] = merge(T[x][0], T[y][0]);
    T[x][1] = merge(T[x][1], T[y][1]);
    return x;
}
struct Edge {
    int x, y, z;
} p[N];
bool cmp(Edge a, Edge b) { return a.z < b.z; }
vector<int> w[N], v[N], s[N];
int n, f[N], ans[N], a[N];
int sf(int x) { return x == f[x] ? x : f[x] = sf(f[x]); }
void Union(int x, int y, int id) {
    int fx = sf(x), fy = sf(y);
    if (s[fx].size() < s[fy].size()) swap(fx, fy);
    for (auto &u : s[fy]) {
        s[fx].push_back(u);
        ans[id] = max(ans[id], query(root[fx], a[u]));
    }
    f[fy] = fx;
```

```
    root[fx] = merge(root[fx], root[fy]);
}
void dfs(int x, int fx) {
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i], z = w[x][i];
        if (y == fx) continue;
        a[y] = a[x] ^ z;
        dfs(y, x);
    }
}
int main() {
    while (~scanf("%d", &n)) {
        init();
        for (int i = 1; i <= n; i++)
            f[i] = i, root[i] = newnode(), s[i].clear();
        for (int i = 1; i < n; i++) {
            scanf("%d%d%d", &p[i].x, &p[i].y, &p[i].z);
            int x = p[i].x, y = p[i].y, z = p[i].z;
            v[x].push_back(y);
            w[x].push_back(z);
            v[y].push_back(x);
            w[y].push_back(z);
        }
        a[1] = 0;
        dfs(1, 1);
        for (int i = 1; i <= n; i++) insert(root[i], a[i]), s[i].push_back(i);
        sort(p + 1, p + n, cmp);
        for (int i = 1; i < n; i++) {
            Union(p[i].x, p[i].y, i);
            printf("%d%c", ans[i], " \n"[i == n - 1]);
        }
    }
    return 0;
}
```

2.47 可持久化字典树

```
/*
    可持久化Trie
    支持查询区间内与某个数异或最大值
    trick: 当求区间和异或最大的时候，我们可以求前缀异或和，
    这样子又转化回选择一个已插入的数使得其与当前值异或值最大
    Presum[R]^Presum[L-1]=Xorsum[L~R]
    那么插入R的同时查询与之前的Xor最大值即可，复杂度O(len*n)
*/
```

```
const int N = 600010;
int bin[32], root[N];
bin[0] = 1;
struct trie {
    int cnt, ch[N * 32][2], sum[N * 32];
    void Initialize() {
        memset(ch, 0, sizeof(ch));
        memset(sum, 0, sizeof(sum));
        cnt = 0;
    }
    int insert(int x, int val) {
        int tmp, y;
        tmp = y = ++cnt;
        for (int i = 31; i >= 0; i--) {
            int t = val & bin[i];
            t >>= i;
            ch[y][0] = ch[x][0];
            ch[y][1] = ch[x][1];
            x = ch[x][t];
            y = ch[y][t] = ++cnt;
            sum[y] = sum[x] + 1;
        }
        return tmp;
    }
    int query(int l, int r, int val) {
        int tmp = 0;
        for (int i = 31; i >= 0; i--) {
            int t = val & bin[i];
            t >>= i;
            if (sum[ch[r][t ^ 1]] - sum[ch[l][t ^ 1]])
                tmp += bin[i], r = ch[r][t ^ 1], l = ch[l][t ^ 1];
            else
                r = ch[r][t], l = ch[l][t];
        }
        return tmp;
    }
} Trie;
// Test
int main() {
    for (int i = 1; i < 32; i++) bin[i] = bin[i - 1] << 1;
    for (int Cas = 1; Cas <= T; Cas++) {
        Trie.Initialize();
        // 插入数据
        for (int i = 1; i <= n; i++) root[i] = trie.insert(root[i - 1], a[i]);
        // 查询 [L,R] 中与x异或值最大的数
        int ans=trie.query(root[L-1],root[R],x));
```

```
    }
}
```

2.48 并查集

```
/*
    按秩合并并查集
    连通块数量为block，大小为size。
*/
namespace Union_Find_Set {
int f[M], size[M], block;
void Initialize() {
    memset(f, 0, sizeof(f));
    block = n;
}
int Find(int x) {
    if (!f[x]) f[x] = x, size[x] = 1;
    if (f[x] == x) return x;
    return f[x] = Find(f[x]);
}
void Union(int x, int y) {
    x = Find(x);
    y = Find(y);
    if (x == y) return;
    if (size[x] > size[y]) swap(x, y);
    f[x] = y;
    size[y] += size[x];
    block--;
}
} // namespace Union_Find_Set
/*
    加权并查集
*/
int sf(int x) {
    if (f[x] == x)
        return x;
    else {
        int fx = sf(f[x]);
        r[x] = (r[x] + r[f[x]]) % 2;
        return f[x] = fx;
    }
}
void U(int x, int y, int fx, int fy, int d) {
    f[fy] = fx;
    r[fy] = (2 - r[y] + d + r[x]) % 2;
```

```
}
/*
    食物链(拆点)
    草原上有三种物种，分别为A,B,C
    A吃B，B吃C，C吃A。
    1 x y表示x和y是同类，2 x y表示x吃y
    问给出的信息有几条是和前面相违背的
*/
#include <cstdio>
const int N = 50010 * 3;
int f[N], n, k, ans = 0, op, x, y;
void init(int n) {
    for (int i = 0; i <= n; i++) f[i] = i;
}
int sf(int x) { return f[x] == x ? x : f[x] = sf(f[x]); }
bool Same(int x, int y) { return sf(x) == sf(y); }
bool Union(int x, int y) { f[sf(x)] = sf(y); }
int main() {
    scanf("%d%d", &n, &k);
    init(3 * n);
    for (int i = 1; i <= k; i++) {
        scanf("%d%d%d", &op, &x, &y);
        if (--x < 0 || x >= n || --y < 0 || y >= n) {
            ans++;
            continue;
        }
        if (op == 1) {
            if (Same(x, y + n) || Same(x, y + 2 * n))
                ans++;
            else {
                Union(x, y);
                Union(x + n, y + n);
                Union(x + 2 * n, y + 2 * n);
            }
        } else {
            if (Same(x, y) || Same(x, y + 2 * n))
                ans++;
            else {
                Union(x, y + n);
                Union(x + n, y + 2 * n);
                Union(x + 2 * n, y);
            }
        }
    }
    printf("%d\n", ans);
    return 0;
```

```
}
```

2.49 可持久化并查集

```
/*
    可持久化并查集
    1.当前版本集合合并
    2.回退版本y
    3.当前版本集合判定
*/
#include <cstdio>
const int N = 2000005;
int n, m, i, x, y, z, ans;
int l[N << 2], r[N << 2], v[N << 2], tot, root[N];
int build(int a, int b) {
    int x = ++tot;
    if (a == b) return v[x] = a, x;
    int mid = (a + b) >> 1;
    return l[x] = build(a, mid), r[x] = build(mid + 1, b), x;
}
int change(int x, int a, int b, int c, int p) {
    int y = ++tot;
    if (a == b) return v[y] = p, y;
    int mid = (a + b) >> 1;
    if (c <= mid)
        l[y] = change(l[x], a, mid, c, p), r[y] = r[x];
    else
        l[y] = l[x], r[y] = change(r[x], mid + 1, b, c, p);
    return y;
}
int ask(int x, int a, int b, int c) {
    if (a == b) return v[x];
    int mid = (a + b) >> 1;
    return c <= mid ? ask(l[x], a, mid, c) : ask(r[x], mid + 1, b, c);
}
int f(int x) {
    int t = ask(root[i], 1, n, x);
    if (t == x) return x;
    return root[i] = change(root[i], 1, n, x, t = f(t)), t;
}
int main() {
    scanf("%d%d", &n, &m);
    for (root[0] = build(i = 1, n); i <= m; i++) {
        root[i] = root[i - 1];
        scanf("%d%d", &x, &y);
```

```
        if (x != 2) scanf("%d", &z);
        if (x == 1) root[i] = change(root[i], 1, n, f(y), f(z));
        if (x == 2) root[i] = root[y];
        if (x == 3) printf("%d\n", f(y) == f(z));
    }
    return 0;
}
```

2.50 可回溯并查集

```
/*
    ask(x): 查询与集合标识的距离奇偶性
    合并操作: Union(fx,fy,ask(x)^ask(y)^1)
    撤回操作: back(Tim), 回退到时间Tim
    查询复杂度 O(logn)
*/
namespace Union_Find_Set {
int st[N], top, f[N], val[N], d[N];
void Initialize(int n) {
    for (int i = 1; i <= n; i++) f[i] = i, val[i] = 0, d[i] = 1;
    top = 0;
}
int sf(int x) { return f[x] == x ? x : sf(f[x]); }
int ask(int x) {
    int res = 0;
    for (; x != f[x]; x = f[x]) res ^= val[x];
    return res;
}
void back(int tag) {
    for (; top != tag; top--) {
        if (st[top] < 0)
            d[-st[top]]--;
        else {
            f[st[top]] = st[top];
            val[st[top]] = 0;
        }
    }
}
void Union(int x, int y, int _val) {
    if (d[x] > d[y]) swap(x, y);
    if (d[x] == d[y]) d[y]++, st[++top] = -y;
    f[x] = y;
    val[x] = _val;
    st[++top] = x;
}
```

```
} // namespace Union_Find_Set
/*
    Test
    给出一张图，有些边只存在一段时间，
    问在一个每个时间段，这张图是否是二分图
*/
using namespace Union_Find_Set;
struct data {
    int x, y, l, r;
} E[N];
void dfs(int l, int r, int pos) {
    int t = top;
    for (int i = 1; i <= pos; i++) {
        int x = E[i].x, y = E[i].y;
        if (E[i].l <= l && E[i].r >= r) {
            int fx = sf(x), fy = sf(y);
            int val = ask(x) ^ ask(y) ^ 1;
            if (fx == fy) {
                if (val) {
                    for (int j = l; j <= r; j++) puts("No");
                    back(t);
                    return;
                }
            }
            Union(fx, fy, val);
            swap(E[i--], E[pos--]);
        }
    }
    if (l == r) {
        puts("Yes");
        back(t);
        return;
    }
    int mid = (l + r) >> 1, ppos = pos;
    for (int i = 1; i <= ppos; i++) {
        if (E[i].l > mid) swap(E[i--], E[ppos--]);
    }
    dfs(l, mid, ppos);
    ppos = pos;
    for (int i = 1; i <= ppos; i++) {
        if (E[i].r <= mid) swap(E[i--], E[ppos--]);
    }
    dfs(mid + 1, r, ppos);
    back(t);
}
/*
```

```
    对时间进行分治，在操作树上加删边，
    保留涵盖时间区间的有效操作，将剩余操作按时间划分到两端的子树，
    退出子树的时候撤销加边操作。
    对于判断奇环，我们用并查集维护每个点与标兵的相对距离的奇偶性即可
*/
int n, m, op, x, y, l, r, T;
int main() {
    scanf("%d%d%d", &n, &m, &T);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        E[i] = (data){x, y, ++l, r};
    }
    Initialize(n);
    for (int i = 1; i <= m; i++)
        if (E[i].l > E[i].r) swap(E[i], E[m--]);
    dfs(1, T, m);
    return 0;
}
```

2.51 树套树

```
/*
    线段树套Treap
    离散动态区间第k小
*/
const int N = 200010, M = 524289;
int n, m, cnt, i, a[N], op[N][4], b[N];
inline void read(int& a) {
    char c;
    while (!((c = getchar()) >= '0') && (c <= '9')))
        ;
    a = c - '0';
    while (((c = getchar()) >= '0') && (c <= '9')) (a *= 10) += c - '0';
}
inline int lower(int x) {
    int l = 1, r = cnt, t, mid;
    while (l <= r)
        if (b[mid = (l + r) >> 1] <= x)
            l = (t = mid) + 1;
        else
            r = mid - 1;
    return t;
}
struct node {
    int val, cnt, sum, p;
```

```
    node *l, *r;
    node() {
        val = sum = cnt = p = 0;
        l = r = NULL;
    }
    inline void up() { sum = l->sum + r->sum + cnt; }
}* blank = new (node), *T[M], pool[2000000], *cur;
inline void Rotatel(node*& x) {
    node* y = x->r;
    x->r = y->l;
    x->up();
    y->l = x;
    y->up();
    x = y;
}
inline void Rotater(node*& x) {
    node* y = x->l;
    x->l = y->r;
    x->up();
    y->r = x;
    y->up();
    x = y;
}
void Ins(node*& x, int y, int p) {
    if (x == blank) {
        x = cur++;
        x->val = y;
        x->l = x->r = blank;
        x->sum = x->cnt = 1;
        x->p = std::rand();
        return;
    }
    x->sum += p;
    if (y == x->val) {
        x->cnt += p;
        return;
    }
    if (y < x->val) {
        Ins(x->l, y, p);
        if (x->l->p > x->p) Rotater(x);
    } else {
        Ins(x->r, y, p);
        if (x->r->p > x->p) Rotatel(x);
    }
}
inline int Ask(node* x, int y) { // ask how many <= y
```

```
    int t = 0;
    while (x != blank)
        if (y < x->val)
            x = x->l;
        else
            t += x->l->sum + x->cnt, x = x->r;
    return t;
}
inline void add(int v, int i, int p) {
    int a = 1, b = cnt, mid, f = 1, x = 1;
    while (a < b) {
        if (f) Ins(T[x], i, p);
        mid = (a + b) >> 1;
        x <<= 1;
        if (f = v <= mid)
            b = mid;
        else
            a = mid + 1, x |= 1;
    }
    Ins(T[x], i, p);
}
inline int kth(int l, int r, int k) {
    int x = 1, a = 1, b = cnt, mid;
    while (a < b) {
        mid = (a + b) >> 1;
        x <<= 1;
        int t = Ask(T[x], r) - Ask(T[x], l - 1);
        if (k <= t)
            b = mid;
        else
            k -= t, a = mid + 1, x |= 1;
    }
    return a;
}
void build(int x, int a, int b) {
    T[x] = blank;
    if (a == b) return;
    int mid = (a + b) >> 1;
    build(x << 1, a, mid), build(x << 1 | 1, mid + 1, b);
}
int main() {
    blank->l = blank->r = blank;
    while (~scanf("%d", &n)) {
        cur = pool;
        for (i = 1; i <= n; i++) read(a[i]), b[i] = a[i];
        cnt = n;
```

```
        read(m);
        for (i = 1; i <= m; i++) {
            read(op[i][0]), read(op[i][1]), read(op[i][2]);
            if (op[i][0] == 1)
                b[++cnt] = op[i][2];
            else
                read(op[i][3]);
        }
        sort(b + 1, b + cnt + 1);
        for (i = 1; i <= n; i++) a[i] = lower(a[i]);
        for (i = 1; i <= m; i++)
            if (op[i][0] == 1) op[i][2] = lower(op[i][2]);
        build(1, 1, cnt);
        for (i = 1; i <= n; i++) add(a[i], i, 1);
        for (i = 1; i <= m; i++) {
            if (op[i][0] == 1)
                add(a[op[i][1]], op[i][1], -1),
                    add(a[op[i][1]] = op[i][2], op[i][1], 1);
            else
                printf("%d\n", b[kth(op[i][1], op[i][2], op[i][3])]);
        }
    }
    return 0;
}
/*
    权值线段树套替罪羊式KD树
    操作 1 x y k表示在点(x,y)上放置k个物品,
    操作 2 x0 y0 x1 y1 k表示查询矩形内放置物品第k多的格子有几个物品
    同一个格子不会被同时放置物品一次以上
*/
const int N = 3000000, INF = 1e9;
namespace KD_Tree {
struct Dot {
    int d[2], mn[2], mx[2], l, r, sz;
    Dot() { l = r = 0; }
    Dot(int x, int y) {
        d[0] = x;
        d[1] = y;
        l = r = sz = 0;
    }
    int& operator[](int x) { return d[x]; }
};
int D, dcnt = 0; // 维度,点数
Dot T[N];
inline void umax(int& a, int b) {
    if (a < b) a = b;
```

```
}
inline void umin(int& a, int b) {
    if (a > b) a = b;
}
inline bool cmp(int x, int y) { return T[x][D] < T[y][D]; }
inline void up(int x) {
    T[x].sz = T[T[x].l].sz + T[T[x].r].sz + 1;
    T[x].mn[0] = T[x].mx[0] = T[x][0];
    T[x].mn[1] = T[x].mx[1] = T[x][1];
    if (T[x].l) {
        umax(T[x].mx[0], T[T[x].l].mx[0]);
        umin(T[x].mn[0], T[T[x].l].mn[0]);
        umax(T[x].mx[1], T[T[x].l].mx[1]);
        umin(T[x].mn[1], T[T[x].l].mn[1]);
    }
    if (T[x].r) {
        umax(T[x].mx[0], T[T[x].r].mx[0]);
        umin(T[x].mn[0], T[T[x].r].mn[0]);
        umax(T[x].mx[1], T[T[x].r].mx[1]);
        umin(T[x].mn[1], T[T[x].r].mn[1]);
    }
}
inline int NewDot(int x, int y) {
    ++dcnt;
    T[dcnt][0] = x;
    T[dcnt][1] = y;
    return up(dcnt), dcnt;
}
// 查询矩阵内点的个数
int query(int x, int x0, int y0, int x1, int y1) {
    if (!x || T[x].mn[0] > x1 || T[x].mx[0] < x0 || T[x].mn[1] > y1 ||
        T[x].mx[1] < y0)
        return 0;
    if (T[x].mn[0] >= x0 && T[x].mx[0] <= x1 && T[x].mn[1] >= y0 &&
        T[x].mx[1] <= y1)
        return T[x].sz;
    int res = 0;
    if (T[x][0] >= x0 && T[x][0] <= x1 && T[x][1] >= y0 && T[x][1] <= y1) res++;
    return res + query(T[x].l, x0, y0, x1, y1) + query(T[x].r, x0, y0, x1, y1);
}
// 替罪羊式重构
int tot = 0, pt[N];
int gt, gtd, gtf;
const double alp = 0.8;
inline bool isbad(int x) {
    return max(T[T[x].l].sz, T[T[x].r].sz) > T[x].sz * alp + 5;
```

```
}
void ins(int& x, int D, const Dot& p) {
    if (!x) {
        x = NewDot(p.d[0], p.d[1]);
        return;
    }
    if (p.d[D] < T[x][D])
        ins(T[x].l, D ^ 1, p);
    else
        ins(T[x].r, D ^ 1, p);
    up(x);
    if (isbad(x))
        gt = x, gtd = D, gtf = 0;
    else if (gt == T[x].l || gt == T[x].r)
        gtf = x;
}
void treavel(int& x) {
    if (!x) return;
    pt[++tot] = x;
    treavel(T[x].l), treavel(T[x].r);
}
int build(int l, int r, int now) {
    if (l > r) return 0;
    int mid = (l + r) >> 1, x;
    D = now;
    nth_element(pt + l, pt + mid, pt + r + 1, cmp);
    x = pt[mid];
    T[x].l = build(l, mid - 1, now ^ 1);
    T[x].r = build(mid + 1, r, now ^ 1);
    return up(x), x;
}
void Insert(int& x, const Dot& p) {
    gt = gtf = 0, ins(x, 0, p);
    if (!gt) return;
    tot = 0, treavel(gt);
    if (!gtf) {
        x = build(1, tot, gtd);
        return;
    }
    if (gt == T[gtf].l)
        T[gtf].l = build(1, tot, gtd);
    else
        T[gtf].r = build(1, tot, gtd);
}
} // namespace KD_Tree
int tot = 0;
```

```
struct data {
    int rt, l, r;
} T[N];
void Insert(int& x, const KD_Tree::Dot& p, int val, int l = 1, int r = INF) {
    if (!x) x = ++tot;
    KD_Tree::Insert(T[x].rt, p);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (val <= mid)
        Insert(T[x].l, p, val, l, mid);
    else
        Insert(T[x].r, p, val, mid + 1, r);
}
int query(int x, int x0, int y0, int x1, int y1, int k, int l = 1,
          int r = INF) {
    if (l == r) return l;
    int rcnt = KD_Tree::query(T[T[x].r].rt, x0, y0, x1, y1);
    int mid = (l + r) >> 1;
    if (k <= rcnt) return query(T[x].r, x0, y0, x1, y1, k, mid + 1, r);
    return query(T[x].l, x0, y0, x1, y1, k - rcnt, l, mid);
}
int n, q, op, x0, y0, x1, y1, k, root = 0, ans = 0;
int main() {
    scanf("%d%d", &n, &q);
    while (q--) {
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%d%d", &x0, &y0, &k);
            x0 ^= ans, y0 ^= ans, k ^= ans;
            KD_Tree::Dot p = KD_Tree::Dot(x0, y0);
            Insert(root, p, k);
        } else {
            scanf("%d%d%d%d%d", &x0, &y0, &x1, &y1, &k);
            x0 ^= ans, y0 ^= ans, x1 ^= ans, y1 ^= ans, k ^= ans;
            int res = KD_Tree::query(T[root].rt, x0, y0, x1, y1);
            if (res < k)
                puts("NAIVE!ORZzyz."), ans = 0;
            else
                printf("%d\n", ans = query(root, x0, y0, x1, y1, k));
        }
    }
    return 0;
}
```

3 图论

3.1 SPFA

```
/*
    SPFA
    SLF优化
*/
const int N = 66000;
int v[M << 1], w[M << 1], g[N], d[N], nxt[M << 1], ed;
int q[N];
unsigned short h, t;
void add_edge(int x, int y, int z) {
    v[++ed] = y;
    w[ed] = z;
    nxt[ed] = g[x];
    g[x] = ed;
}
void add(int x, int y) {
    if (y >= d[x]) return;
    d[x] = y;
    if (!in[x]) {
        in[x] = 1;
        if (y < d[q[h]])
            q[--h] = x;
        else
            q[++t] = x; // SLF优化
    }
}
void spfa(int S) { // S为源点
    int i, x;
    for (i = h = 1; i <= n; i++) d[i] = inf, in[i] = 0;
    add(S, t = 0);
    while (h != t + 1)
        for (i = g[x = q[h++]], in [x] = 0; i; i = nxt[i])
            add(v[i], d[x] + w[i]);
}
```

3.2 DFS 版 SPFA

```
/*
    SPFA-DFS
    题目大意：给出n个字符串，当两个字符串首尾有两个字母相同的时候就能可以相接，
    问能否接成一个串环，使得这个环上串的平均长度最长
```

```
    题解：我们将串的连接转化为首尾点的连接，边长为串长，那么问题转化为
    求一个平均长度最长的环，我们二分这个平均长度，用所有的边减去这个二分值，
    那么只要完成是否存在正环的判定即可，环的判定用SPFA-DFS效率比较高
*/
using namespace std;
typedef long long LL;
const double eps = 1e-3;
const int N = 100010;
int ed, head[N], u[N], v[N], nxt[N], cnt[N];
double d[N], w[N];
void init() {
    ed = 0;
    memset(head, -1, sizeof(head));

    void add(int a, int b, double c) {
        u[++ed] = a, v[ed] = b, w[ed] = c;
        nxt[ed] = head[u[ed]];
        head[u[ed]] = ed;
    }
    char s[N];
    const int M = 25 * 26 + 25 + 1;
    double mx;
    int f[M], tot, n;
    int vis[N], PC;
    void SPFA(int x, double ave) {
        if (PC) return;
        vis[x] = 1;
        for (int i = head[x]; i != -1; i = nxt[i]) {
            if (d[x] + w[i] - ave > d[v[i]] + eps) {
                d[v[i]] = d[x] + w[i] - ave;
                if (!vis[v[i]])
                    SPFA(v[i], ave);
                else {
                    PC = 1;
                    return;
                }
            }
        }
        vis[x] = 0;
    }
    bool Check(double x) {
        PC = 0;
        memset(vis, 0, sizeof(vis));
        memset(d, 0, sizeof(d));
        for (int i = 1; i <= tot; i++) {
            SPFA(i, x);
```

```
            if (PC) break;
        }
        return PC;
    }
    int main() {
        while (~scanf("%d", &n) && n) {
            init();
            tot = 0;
            memset(f, 0, sizeof(f));
            for (int i = 1; i <= n; i++) {
                scanf("%s", s + 1);
                int l = strlen(s + 1);
                int a = (s[1] - 'a') * 26 + s[2] - 'a';
                int b = (s[l - 1] - 'a') * 26 + s[l] - 'a';
                if (!f[a]) f[a] = ++tot;
                if (!f[b]) f[b] = ++tot;
                if (l > 1) add(f[a], f[b], l);
            }
            mx *= n;
            double ans = -1, l = 0, r = 1000;
            while (l + eps < r) {
                double mid = (l + r) * 0.5;
                if (Check(mid))
                    l = mid, ans = mid;
                else
                    r = mid;
            }
            if (ans == -1)
                puts("No solution.");
            else
                printf("%.3f\n", ans);
        }
        return 0;
    }
```

3.3 差分约束

```
/* 差分约束:
 *  按照A-B<=C建图，或者转化B-A>C为A-B<=-C建图
 *  用最短路来松弛条件，固定起点值可从无穷向下约束得到其余点最大值，
 *  按照A-B>=C建图，用最长路来松弛条件，从负无穷往上约束可得最小值
 *  若图存在负环则表明无解，d为正负INF表明解无穷大或者无穷小
 *  对于非连通图的可行性判断利用加入超级源点的方法可以实现
 *  特别注意dijkstra算法不能处理有负权边的情况
 * Example
```

```
 *  题意：求是否存在一个数列，满足给定m个区间和的取值范围，
 *  题解：对于区间和，我们可以转化为前缀和的点差关系，
 *  根据小于等于建图，若图中不存在负环，说明在该约束条件下有解，
 *  由于图未必连通，所以需要建立超级源点，存在负环的情况下，
 *  用stack代替queue在实际运作中会快很多
*/
using namespace std;
const int N = 210;
int ed, d[N], head[N], u[N], v[N], w[N], nxt[N], cnt[N];
const int INF = ~0U >> 2;
void init() {
    ed = 0;
    memset(head, -1, sizeof(head));
}
void add(int a, int b, int c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
bool SPFA(int s, int n) {
    for (int i = 0; i <= n; i++) d[i] = INF;
    d[s] = 0;
    memset(cnt, 0, sizeof(cnt));
    bool inq[N] = {0};
    stack<int> Q;
    Q.push(s);
    while (!Q.empty()) {
        int x = Q.top();
        Q.pop();
        inq[x] = 0;
        for (int i = head[x]; i != -1; i = nxt[i]) {
            if (d[v[i]] <= d[x] + w[i]) continue;
            d[v[i]] = d[x] + w[i];
            if (!inq[v[i]]) {
                inq[v[i]] = 1;
                Q.push(v[i]);
                if (++cnt[v[i]] > n) return 0;
            }
        }
    }
    return 1;
}
int n, m;
int main() {
    while (scanf("%d", &n) && n) {
        scanf("%d", &m);
```

```
        init();
        char op[10];
        int x, y, z;
        while (m--) {
            scanf("%d %d %s %d", &x, &y, op, &z);
            if (op[0] == 'g')
                add(x + y + 1, x, -z - 1);
            else
                add(x, x + y + 1, z - 1);
        }
        for (int i = 1; i <= n + 1; i++) add(0, i, 0);
        if (SPFA(0, n + 1))
            puts("lamentable kingdom");
        else
            puts("successful conspiracy");
    }
    return 0;
}
```

3.4 Dijkstra

```
/*
    Dijkstra+优先队列
    O(nlogn)
*/
using namespace std;
const int N = 150100, M = 300010;
const LL INF = 1LL << 60;
typedef long long LL;
typedef pair<LL, int> seg;
priority_queue<seg, vector<seg>, greater<seg> > q;
int head[N], u[M], v[M], nxt[M]; // 双向边记得开[M<<1]
LL w[M], d[N];
bool vis[N];
int n, m, ed;
void init() {
    ed = 0;
    memset(head, -1, sizeof(head));
}
void add(int a, int b, LL c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
void Dijkstra(int src, LL d[]) {
```

```
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i <= n; i++) d[i] = INF;
    d[src] = 0;
    q.push(make_pair(d[src], src));
    while (!q.empty()) {
        seg now = q.top();
        q.pop();
        int x = now.second;
        if (vis[x]) continue;
        vis[x] = true;
        for (int e = head[x]; e != -1; e = nxt[e])
            if (d[v[e]] > d[x] + w[e]) {
                d[v[e]] = d[x] + w[e];
                q.push(make_pair(d[v[e]], v[e]));
            }
    }
}
/*
    Test
    给出m个条件A-B<=C,求Xn-X1的最大值
    我们从无穷开始向小约束一定能得到最大值，所以我们从A向B连C的边，求一遍最短路即可
*/
int main() {
    while (~scanf("%d%d", &n, &m)) {
        int A, B, C;
        init();
        while (m--) {
            scanf("%d%d%lld", &A, &B, &C);
            add(A, B, C);
        }
        Dijkstra(1, d);
        printf("%lld\n", d[n]);
    }
    return 0;
}
/*
    Notice
    在多源Dijkstra中，不要建立超级源点，而是直接把点放入队列中，
    可以减少queue开支防止段错误，同时节省时间
*/
inline void Dijkstra() {
    while (!q.empty()) {
        seg t = q.top();
        q.pop();
        if (d[t.second] < t.first) continue;
        for (int i = head[t.second]; ~i; i = nxt[i]) {
```

```
            if (d[v[i]] > t.first + w[i])
                q.push(seg(d[v[i]] = t.first + w[i], v[i]));
        }
    }
}
int main() {
    scanf("%d%d", &n, &m);
    init();
    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        if (x == 1)
            q.push(seg(0, i)), d[i] = 0;
        else
            d[i] = INF;
    }
    /*[some other code]*/
}
```

3.5 同余最短路

```
/* 同余最短路
 * 给定一张n个点m条边的无向带权图,
 * 问是否存在一条从st号点到en号的长度为T的路径, 不需要是简单路径。
 * 取一条与st相连的权值最小的边w
 * 设d[i][j]表示从st到i, 路径长度模2w为j时, 路径长度的最小值。
 * 用最短路算法求出所有d[i][j],然后检查d[en][T%2w]是否不超过T即可
 */
typedef long long LL;
const int N = 1000010;
const LL INF = 0x3f3f3f3f3f3f3f3f;
typedef pair<LL, int> seg;
priority_queue<seg, vector<seg>, greater<seg> > q;
int head[N], u[N], v[N], w[N], nxt[N], n, m, ed = 0;
LL d[5][N], W;
void add(int a, int b, int c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
void Dijkstra(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 _w = now.first;
        int x = now.second;
        if (_w > d[x][_w % W]) continue;
        for (int e = head[x]; e != -1; e = nxt[e]) {
            LL nw = _w + w[e];
            if (d[v[e]][nw % W] > nw) {
                d[v[e]][nw % W] = nw;
                q.push(make_pair(nw, v[e]));
            }
        }
    }
}
/* 例题
 * 给出四个点1,2,3,4，1和2，2和3，3和4，4和1之间有路相连，
 * 现在从2点出发，最后回到2点，要求路径大于等于K，问路径长度最短是多少
 */
int T;
LL k, d1, d2, d3, d4;
int main() {
    scanf("%d", &T);
    n = 4;
    while (T--) {
        LL ans = INF;
        memset(head, -1, sizeof(head));
        ed = 0;
        scanf("%lld%lld%lld%lld%lld", &k, &d1, &d2, &d3, &d4);
        if (d2 < d1)
            W = 2 * d2;
        else
            W = 2 * d1;
        add(3, 4, d3);
        add(4, 3, d3);
        add(2, 1, d1);
        add(1, 2, d1);
        add(2, 3, d2);
        add(3, 2, d2);
        add(1, 4, d4);
        add(4, 1, d4);
        Dijkstra(2);
        for (int i = 0; i < W; i++) {
            if (k <= d[2][i])
                ans = min(ans, d[2][i]);
            else
                ans = min(ans, d[2][i] + ((k - d[2][i] + W - 1) / W) * W);
```

```
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

3.6 分层图最短路

```
/*
    分层图最短路
    (有k条路可以无代价通过)
*/
const int N = 200100;
const int INF = ~0U >> 2;
typedef pair<int, int> seg;
priority_queue<seg, vector<seg>, greater<seg> > q;
int p[N], d[N][25], head[N], u[N], v[N], w[N], nxt[N], n, m, a, b, c, k,
    ed = 0, H, x[N], y[N];
bool vis[N][25];
void add(int a, int b, int c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
void Dijkstra(int src) {
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= k; j++) d[i][j] = INF;
    d[src][0] = 0;
    q.push(make_pair(d[src][0], src));
    while (!q.empty()) {
        seg now = q.top();
        q.pop();
        int deep = now.second / (n + 1), x = now.second % (n + 1);
        if (vis[x][deep]) continue;
        vis[x][deep] = true;
        for (int e = head[x]; e != -1; e = nxt[e]) {
            if (d[v[e]][deep] > d[x][deep] + w[e]) {
                d[v[e]][deep] = d[x][deep] + w[e];
                q.push(make_pair(d[v[e]][deep], deep * (n + 1) + v[e]));
            }
            if (deep == k) continue;
            if (d[x][deep] < d[v[e]][deep + 1]) {
                d[v[e]][deep + 1] = d[x][deep];
                q.push(
```

```
                make_pair(d[v[e]][deep + 1], (deep + 1) * (n + 1) + v[e]));
            }
        }
    }
}
void solve() {
    scanf("%d%d%d", &n, &m, &k);
    memset(head, -1, sizeof(head));
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
        add(b, a, c);
    }
    Dijkstra(1);
    printf("%d\n", d[n][k]);
}
```

3.7 次短路

```
/*
    次短路
*/
typedef long long LL;
const int MAX_V = 100010;
struct edge {
    int to;
    LL cost;
    edge() {}
    edge(int to, LL cost) : to(to), cost(cost) {}
};
typedef pair<LL, int> P;
vector<edge> G[MAX_V];
LL d1[MAX_V], d2[MAX_V];
void dijkstra(int s) {
    priority_queue<P, vector<P>, greater<P> > que;
    memset(d1, 0x3f, sizeof(d1));
    memset(d2, 0x3f, sizeof(d2));
    que.push(P(d1[s] = 0, s));
    while (!que.empty()) {
        P p = que.top();
        que.pop();
        int v = p.second, u;
        LL d;
        if (d2[v] < p.first) continue;
        for (int i = 0; i < G[v].size(); i++) {
```

```
            u = G[v][i].to;
            d = p.first + G[v][i].cost;
            if (d1[u] > d) {
                d2[u] = d1[u];
                d1[u] = d;
                que.push(P(d, u));
            } else if (d2[u] > d) {
                d2[u] = d;
                que.push(P(d, u));
            }
        }
    }
}
int T, n, m;
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) G[i].clear();
        while (m--) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            G[x].push_back(edge(y, z));
            G[y].push_back(edge(x, z));
        }
        dijkstra(1);
        printf("%lld\n", d2[n]);
    }
    return 0;
}
```

3.8 严格次短路

```
/*
    严格次短路
*/
const int N = 200100;
const int INF = ~0U >> 2;
typedef pair<int, int> seg;
priority_queue<seg, vector<seg>, greater<seg> > q;
int p[N], d[2][N], head[N], u[N], v[N], w[N], nxt[N], n, m, a[N], b[N], c[N],
    ed = 0, H, x[N], y[N];
bool vis[N];
void add(int a, int b, int c) {
```

```
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
void Dijkstra(int src, int t) {
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i <= n + 1; i++) d[t][i] = INF;
    d[t][src] = 0;
    q.push(make_pair(d[t][src], src));
    while (!q.empty()) {
        seg now = q.top();
        q.pop();
        int x = now.second;
        if (vis[x]) continue;
        vis[x] = true;
        for (int e = head[x]; e != -1; e = nxt[e])
            if (d[t][v[e]] > d[t][x] + w[e]) {
                d[t][v[e]] = d[t][x] + w[e];
                q.push(make_pair(d[t][v[e]], v[e]));
            }
    }
}
void Solve() {
    scanf("%d%d", &n, &m);
    memset(head, -1, sizeof(head));
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &a[i], &b[i], &c[i]);
        add(a[i], b[i], c[i]);
        add(b[i], a[i], c[i]);
    }
    Dijkstra(1, 0);
    Dijkstra(n, 1);
    for (int i = 1; i <= 2 * m; i++) {
        p[2 * i - 1] = c[i] + d[0][a[i]] + d[1][b[i]];
        p[2 * i] = c[i] + d[1][a[i]] + d[0][b[i]];
    }
    sort(p + 1, p + 2 * m + 1);
    for (int i = 2; i <= m; i++)
        if (p[i] != p[i - 1]) {
            printf("%d\n", p[i]);
            return;
        }
}
```

3.9 K 短路

```
/*
    求k短路
    Astar算法
*/
int n, m, s, t, k, dis[MAXN];
struct node {
    int v, c;
    node(int v, int c) : v(v), c(c) {}
    //用于优先队列先出的条件
    inline bool operator<(const node &b) const {
        return c + dis[v] > b.c + dis[b.v];
    }
};
vector<node> map1[MAXN]; //用于dijkstra算法
vector<node> map2[MAXN]; //用于A_star算法
void dijkstra() {
    int i, find[MAXN], v;
    for (i = 1; i <= n; i++) dis[i] = INF;
    memset(find, 0, sizeof(find));
    priority_queue<node> heap;
    dis[t] = 0;
    heap.push(node(t, 0));
    while (!heap.empty()) {
        v = heap.top().v;
        heap.pop();
        if (find[v]) continue;
        find[v] = 1;
        for (i = 0; i < map1[v].size(); i++)
            if (!find[map1[v][i].v] &&
                dis[v] + map1[v][i].c < dis[map1[v][i].v]) {
                dis[map1[v][i].v] = dis[v] + map1[v][i].c;
                heap.push(node(map1[v][i].v, dis[map1[v][i].v]));
            }
    }
}
int A_star() {
    int i, cnt[MAXN], v, g;
    if (dis[s] == INF) return -1;
    priority_queue<node> heap;
    memset(cnt, 0, sizeof(cnt));
    heap.push(node(s, 0)); // 0是g(x)
    while (!heap.empty()) {
        v = heap.top().v;
        g = heap.top().c;
```

```
        heap.pop();
        cnt[v]++;
        if (cnt[t] == k) return g;
        if (cnt[v] > k) continue;
        for (i = 0; i < map2[v].size(); i++)
            heap.push(node(map2[v][i].v, g + map2[v][i].c));
    }
    return -1;
}
int main() {
    int i, u, v, c;
    scanf("%d%d", &n, &m);
    for (i = 0; i < m; i++) {
        scanf("%d%d%d", &u, &v, &c);
        map2[u].push_back(node(v, c));
        map1[v].push_back(node(u, c)); //反向储存求各节点到目标节点的最短距离
    }
    scanf("%d%d%d", &s, &t, &k);
    if (s == t) k++;
    dijkstra();
    int ans = A_star();
    printf("%d\n", ans);
    return 0;
}
```

3.10 K 短路输出路径

```
/* YenAlgorithm
 * O(kn(m+nlogn))
 * 求出s到t的k短路并输出路径
 */
typedef vector<int> Path;
struct Cell {
    int len, pos;
    Path path, revPath;
    vector<vector<int> > forbiddenList;
    Cell() {}
    Cell(int _len, int _pos, Path _path) : len(_len), pos(_pos), path(_path) {
        revPath = path;
        reverse(path.begin(), path.end());
    }
    bool operator<(const Cell &t) const {
        return len > t.len || len == t.len && revPath > t.revPath;
    }
};
```

```
#define RESET()                                         \
    memset(pre, 0, sizeof(pre)), memset(d, 0x3f, sizeof(d)), \
        memset(flag, 0, sizeof(flag)), d[S] = 0, flag[S] = 1
const int INF = 0x3f3f3f3f;
int n, m, K, s, t;
int g[55][55];
int pre[55], d[55];
int flag[55];
int filter[55][55];
priority_queue<Cell> q;
Cell dijkstra(int S, int T) {
    int now = S;
    while (1) {
        flag[now] = 1;
        for (int i = 1; i <= n; i++)
            if (!filter[now][i] && g[now][i] < INF) {
                if (g[now][i] + d[now] < d[i])
                    d[i] = d[now] + g[now][i], pre[i] = now;
                else if (g[now][i] + d[now] == d[i] && now < pre[i])
                    pre[i] = now;
            }
        now = 0;
        for (int i = 1; i <= n; i++)
            if (!flag[i] && d[i] < d[now]) now = i;
        if (!now) break;
    }
    if (d[T] == INF) return Cell(0, 0, Path());
    int fork_p, cnt = 0;
    Path tmp;
    for (int p = T; p; p = pre[p], cnt++)
        tmp.push_back(p), pre[p] == S && (fork_p = cnt);
    return Cell(d[T], tmp.size() - 1 - fork_p, tmp);
}
void modify(const Cell &pre, Cell &now) {
    for (int i = 0; i < now.path.size(); i++)
        now.forbiddenList.push_back(vector<int>());
    for (int i = 0; i < now.path.size() - 1; i++)
        now.forbiddenList[i].push_back(now.path[i + 1]);
    if (pre.path.empty()) return;
    now.forbiddenList[now.pos - 1] = pre.forbiddenList[now.pos - 1];
    now.forbiddenList[now.pos - 1].push_back(now.path[now.pos]);
}
void printPath(Path &path) {
    if (!path.size())
        puts("No");
    else {
```

```
        printf("%d", path[0]);
        for (int i = 1; i < path.size(); i++) printf("-%d", path[i]);
        puts("");
    }
}
Path yenAlgorithm(int S, int T, int K) {
    RESET();
    Cell now = dijkstra(S, T);
    if (now.path.empty()) return Path();
    modify(Cell(), now);
    for (int i = 1; i < K; i++) {
        Path nowP = now.path;
        int pos = now.pos;
        for (int j = pos - 1; j < nowP.size() - 1; j++) {
            RESET();
            for (int k = 1; k <= j; k++)
                d[nowP[k]] =
                    d[pre[nowP[k]] = nowP[k - 1]] + g[nowP[k - 1]][nowP[k]],
                flag[nowP[k]] = 1;
            memset(filter, 0, sizeof(filter));
            for (int k = 0; k < now.forbiddenList[j].size(); k++)
                filter[nowP[j]][now.forbiddenList[j][k]] = 1;
            Cell newOne = dijkstra(nowP[j], T);
            if (newOne.path.empty()) continue;
            modify(now, newOne);
            q.push(newOne);
        }
        if (q.empty()) return Path();
        now = q.top();
        q.pop();
    }
    return now.revPath;
}
int main() {
    scanf("%d%d%d%d%d", &n, &m, &K, &s, &t);
    memset(g, 0x3f, sizeof(g));
    while (m--) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        g[v][u] = w;
    }
    Path path = yenAlgorithm(t, s, K);
    printPath(path);
    return 0;
}
```

3.11 前 K 短路

```
/*
    前k短路
    Astar算法
    允许被重复走
*/
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> P;
const int N = 10100, M = 10010, inf = 1000000010;
int n, m, k, S, T, i, x, y, z;
int g[N], h[N], v[M << 1], w[M << 1], nxt[M << 1], ed, d[N], vis[N], ans[N];
void add(int x, int y, int z) {
    v[++ed] = x;
    w[ed] = z;
    nxt[ed] = g[y];
    g[y] = ed;
    v[++ed] = y;
    w[ed] = z;
    nxt[ed] = h[x];
    h[x] = ed;
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        scanf("%d%d%d", &S, &T, &k);
        memset(g, ed = 0, sizeof(g));
        memset(h, 0, sizeof(h));
        memset(vis, 0, sizeof(vis));
        priority_queue<P, vector<P>, greater<P> > Q;
        for (i = 1; i <= k; i++) ans[i] = -1;
        while (m--) scanf("%d%d%d", &x, &y, &z), add(x, y, z);
        for (i = 1; i <= n; i++) d[i] = inf;
        Q.push(P(d[T] = 0, T));
        while (!Q.empty()) {
            P t = Q.top();
            Q.pop();
            if (d[t.second] < t.first) continue;
            for (i = g[x = t.second]; i; i = nxt[i])
                if (d[x] + w[i] < d[v[i]])
                    Q.push(P(d[v[i]] = d[x] + w[i], v[i]));
        }
        if (d[S] < inf) Q.push(P(d[S], S));
        while (!Q.empty()) {
            P t = Q.top();
            Q.pop();
```

```
            vis[x = t.second]++;
            if (x == T && vis[T] <= k) ans[vis[T]] = t.first;
            if (vis[T] > k) break;
            if (vis[x] <= k)
                for (i = h[x]; i; i = nxt[i])
                    Q.push(P(t.first - d[x] + d[v[i]] + w[i], v[i]));
        }
        for (int i = 1; i <= k; i++) printf("%d\n", ans[k]);
    }
    return 0;
}
```

3.12 稳定 k 短路

```
/*
    稳定k短路
    左偏树优化
    允许重复走
*/
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> P;
const int N = 10100, M = 100010, inf = ~0U >> 1;
int n, m, i, S, T, K, x, y, z;
int g[N], v[M], u[M], w[M], nxt[M], d[N], f[N], h[N], tot;
bool is[M], vis[N];
struct Node {
    int l, r, d;
    P v;
    Node() {}
    Node(int _l, int _r, int _d, P _v) { l = _l, r = _r, d = _d, v = _v; }
} pool[2000010];
int build(P v) {
    pool[++tot] = Node(0, 0, 0, v);
    return tot;
}
int merge(int a, int b) {
    if (!a || !b) return a + b;
    if (pool[a].v > pool[b].v) swap(a, b);
    int x = ++tot;
    pool[x] = pool[a];
    pool[x].r = merge(pool[a].r, b);
    if (pool[pool[x].l].d < pool[pool[x].r].d) swap(pool[x].l, pool[x].r);
    pool[x].d = pool[x].r ? pool[pool[x].r].d + 1 : 0;
    return x;
```

```
}
void getdis() {
    int i, x;
    priority_queue<P, vector<P>, greater<P> > q;
    for (i = 1; i <= n; i++) d[i] = inf, f[i] = 0;
    q.push(P(d[T] = 0, T));
    while (!q.empty()) {
        P t = q.top();
        q.pop();
        if (t.first > d[x = t.second]) continue;
        for (i = g[x]; i; i = nxt[i])
            if (d[v[i]] > d[x] + w[i]) {
                f[v[i]] = i;
                q.push(P(d[v[i]] = d[x] + w[i], v[i]));
            }
    }
}
void dfs(int x) {
    if (!f[x] || vis[x]) return;
    vis[x] = 1;
    dfs(u[f[x]]);
    h[x] = merge(h[x], h[u[f[x]]]);
}
void add(int x, int y, int z) {
    v[++m] = x;
    u[m] = y;
    w[m] = z;
    nxt[m] = g[y];
    g[y] = m;
}
int tim;
int solve() {
    int mm = m;
    scanf("%d%d%d%d", &S, &T, &K, &tim);
    for (m = 0, i = 1; i <= n; i++) g[i] = 0;
    while (mm--) scanf("%d%d%d", &x, &y, &z), add(x, y, z);
    if (S == T) K++;
    getdis();
    if (d[S] == inf) return -1;
    if (K == 1) return d[S];
    K--;
    for (i = 1; i <= m; i++) is[i] = 0;
    for (tot = 0, i = 1; i <= n; i++) is[f[i]] = 1, h[i] = vis[i] = 0;
    for (i = 1; i <= m; i++)
        if (!is[i] && d[u[i]] < inf)
            h[v[i]] = merge(h[v[i]], build(P(w[i] - d[v[i]] + d[u[i]], u[i])));
```

```
    for (i = 1; i <= n; i++) dfs(i);
    priority_queue<P, vector<P>, greater<P> > q;
    int ans, x, y;
    y = h[S];
    if (y) q.push(P(d[S] + pool[y].v.first, y));
    while (!q.empty() && K) {
        K--;
        P t = q.top();
        q.pop();
        ans = t.first;
        x = t.second, y = pool[x].l;
        if (y) q.push(P(ans - pool[x].v.first + pool[y].v.first, y));
        y = pool[x].r;
        if (y) q.push(P(ans - pool[x].v.first + pool[y].v.first, y));
        y = h[pool[x].v.second];
        if (y) q.push(P(ans + pool[y].v.first, y));
    }
    return K ? -1 : ans;
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        int res = solve();
        if (res != -1 && res <= tim)
            puts("yareyaredawa");
        else
            puts("Whitesnake!");
    }
    return 0;
}
```

3.13 线段树优化建图

```
/*
    线段树优化建图+最短路
    给出一些星球，现在有一些传送枪，可以从一个星球到另一个星球，
    从一个星球到另一些星球，或者从一些星球到某个星球，
    每种传送枪使用一次要花费不同的价格
    地球是其中一个星球，问从地球到其它星球的最少花费是多少
*/
const int N = 100010;
const int V = N * 5;
int n, m, s;
vector<pair<int, int> > G[V];
void addedge(int u, int v, int c) { G[u].push_back(make_pair(v, c)); }
int id[2][N << 2], idx;
```

```
void build(int x, int l, int r, int k) {
    id[k][x] = ++idx;
    if (l == r) {
        if (k == 0)
            addedge(id[k][x], l, 0);
        else
            addedge(l, id[k][x], 0);
        return;
    }
    int mid = (l + r) >> 1;
    build(x << 1, l, mid, k);
    build(x << 1 | 1, mid + 1, r, k);
    if (k == 0) {
        addedge(id[k][x], id[k][x << 1], 0);
        addedge(id[k][x], id[k][x << 1 | 1], 0);
    } else {
        addedge(id[k][x << 1], id[k][x], 0);
        addedge(id[k][x << 1 | 1], id[k][x], 0);
    }
}
vector<int> vs;
void get(int x, int l, int r, int L, int R, int k) {
    if (L <= l && r <= R) {
        vs.push_back(id[k][x]);
        return;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) get(x << 1, l, mid, L, R, k);
    if (R > mid) get(x << 1 | 1, mid + 1, r, L, R, k);
}
typedef long long LL;
const LL LLINF = 0x3f3f3f3f3f3f3f3fLL;
LL dis[V];
bool vis[V];
void Dijkstra(int s) {
    for (int i = 1; i <= 5 * n; i++) vis[i] = 0, dis[i] = LLINF;
    priority_queue<pair<LL, int> > q;
    q.push(make_pair(-0, s));
    dis[s] = 0;
    while (!q.empty()) {
        int u = q.top().second;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = 0; i < G[u].size(); i++) {
            int v = G[u][i].first, c = G[u][i].second;
```

```
            if (dis[v] > dis[u] + c) {
                dis[v] = dis[u] + c;
                q.push(make_pair(-dis[v], v));
            }
        }
    }
}
int main() {
    while (~scanf("%d%d%d", &n, &m, &s)) {
        for (int i = 0; i <= 5 * n; i++) G[i].clear();
        idx = n;
        build(1, 1, n, 0);
        build(1, 1, n, 1);
        while (m--) {
            int t, u;
            scanf("%d%d", &t, &u);
            if (t == 1) {
                int v, c;
                scanf("%d%d", &v, &c);
                addedge(u, v, c);
            } else if (t == 2) {
                vs.clear();
                int l, r, c;
                scanf("%d%d%d", &l, &r, &c);
                get(1, 1, n, l, r, 0);
                for (int i = 0; i < vs.size(); i++) addedge(u, vs[i], c);
            } else {
                vs.clear();
                int l, r, c;
                scanf("%d%d%d", &l, &r, &c);
                get(1, 1, n, l, r, 1);
                for (int i = 0; i < vs.size(); i++) addedge(vs[i], u, c);
            }
        }
        Dijkstra(s);
        for (int i = 1; i <= n; i++) {
            if (dis[i] == LLINF) dis[i] = -1;
            printf("%lld%c", dis[i], i == n ? '\n' : ' ');
        }
    }
    return 0;
}
```

3.14 欧拉回路

```
/*
    无向图:
        欧拉回路: 每个顶点的度数都是偶数, 则存在欧拉回路。
        欧拉路: 当且仅当该图所有顶点的度数为偶数,
                或者除了两个度数为奇数外其余的全是偶数。
    有向图:
        欧拉回路: 每个顶点的入度=出度, 则存在欧拉回路。
        欧拉路: 当且仅当该图所有顶点出度=入度
                或者一个顶点出度=入度+1, 另一个顶点入度=出度+1,
                其他顶点出度=入度
     O(n+m)求欧拉回路
     n为点数, m为边数,
     若有解则依次输出经过的边的编号,
     若是无向图, 则正数表示x到y, 负数表示y到x
*/
namespace UndirectedGraph {
int n, m, i, x, y, d[N], g[N], v[M << 1], w[M << 1], vis[M << 1], nxt[M << 1],
    ed;
int ans[M], cnt;
void add(int x, int y, int z) {
    d[x]++;
    v[++ed] = y;
    w[ed] = z;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    for (int& i = g[x]; i;) {
        if (vis[i]) {
            i = nxt[i];
            continue;
        }
        vis[i] = vis[i ^ 1] = 1;
        int j = w[i];
        dfs(v[i]);
        ans[++cnt] = j;
    }
}
void solve() {
    scanf("%d%d", &n, &m);
    for (i = ed = 1; i <= m; i++)
        scanf("%d%d", &x, &y), add(x, y, i), add(y, x, -i);
    for (i = 1; i <= n; i++)
        if (d[i] & 1) {
            puts("NO");
```

```
            return;
        }
    for (i = 1; i <= n; i++)
        if (g[i]) {
            dfs(i);
            break;
        }
    for (i = 1; i <= n; i++)
        if (g[i]) {
            puts("NO");
            return;
        }
    puts("YES");
    for (i = m; i; i--) printf("%d ", ans[i]);
}
} // namespace UndirectedGraph
namespace DirectedGraph {
int n, m, i, x, y, d[N], g[N], v[M], vis[M], nxt[M], ed;
int ans[M], cnt;
void add(int x, int y) {
    d[x]++;
    d[y]--;
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    for (int& i = g[x]; i;) {
        if (vis[i]) {
            i = nxt[i];
            continue;
        }
        vis[i] = 1;
        int j = i;
        dfs(v[i]);
        ans[++cnt] = j;
    }
}
void solve() {
    scanf("%d%d", &n, &m);
    for (i = 1; i <= m; i++) scanf("%d%d", &x, &y), add(x, y);
    for (i = 1; i <= n; i++)
        if (d[i]) {
            puts("NO");
            return;
        }
```

```
    for (i = 1; i <= n; i++)
        if (g[i]) {
            dfs(i);
            break;
        }
    for (i = 1; i <= n; i++)
        if (g[i]) {
            puts("NO");
            return;
        }
    puts("YES");
    for (i = m; i; i--) printf("%d ", ans[i]);
}
} // namespace DirectedGraph
```

3.15 强连通分量

```
/*
    强连通分量
    点编号 0~n-1
    cmp记录强连通分量拓扑序
    即缩后的点标号
*/
namespace SCC {
const int MAX_V = 10000;
int V;               //顶点数
vector<int> G[MAX_V]; //图的邻接表表示
vector<int> rG[MAX_V]; //反向图
vector<int> vs;      //后序遍历
bool used[MAX_V];
int cmp[MAX_V]; //所属强连通分量的拓扑序
void Initialize() {
    for (int i = 0; i < V; i++) G[i].clear();
    for (int i = 0; i < V; i++) rG[i].clear();
}
void add_edge(int from, int to) {
    G[from].push_back(to);
    rG[to].push_back(from);
}
void dfs(int v) {
    used[v] = 1;
    for (int i = 0; i < G[v].size(); i++) {
        if (!used[G[v][i]]) dfs(G[v][i]);
    }
    vs.push_back(v);
```

```
}
void rdfs(int v, int k) {
    used[v] = 1;
    cmp[v] = k;
    for (int i = 0; i < rG[v].size(); i++) {
        if (!used[rG[v][i]]) rdfs(rG[v][i], k);
    }
}
int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for (int v = 0; v < V; v++) {
        if (!used[v]) dfs(v);
    }
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = vs.size() - 1; i >= 0; i--) {
        if (!used[vs[i]]) rdfs(vs[i], k++);
    }
    return k;
}
} // namespace SCC
/*
    Example1
    询问一张图需要加入的最小有向边数量使得其构成强连通
    Ans=max(NO_IN,NO_OUT)
*/
const int MAX_M = 50000;
int N, M, x;
int A[MAX_M], B[MAX_M];
int in[MAX_V], out[MAX_V], NO_IN, NO_OUT;
void solve() {
    using namespace SCC;
    scanf("%d%d", &N, &M);
    V = N;
    Initialize();
    for (int i = 0; i < M; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        add_edge(x - 1, y - 1);
    }
    int n = scc();
    if (n == 1) {
        puts("0");
        return;
    }
```

```
    memset(in, 0, sizeof(in));
    memset(out, 0, sizeof(out));
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < G[i].size(); j++) {
            if (cmp[i] != cmp[G[i][j]]) {
                ++out[cmp[i]];
                ++in[cmp[G[i][j]]];
            }
        }
    }
    NO_IN = NO_OUT = 0;
    for (int i = 0; i < n; i++) {
        if (!in[i]) NO_IN++;
        if (!out[i]) NO_OUT++;
    }
    printf("%d\n", max(NO_IN, NO_OUT));
}
/*
    Example2
    求大小大于1的强连通数量
*/
const int MAX_M = 50000;
int N, M;
int A[MAX_M], B[MAX_M];
int cnt[MAX_M];
void solve() {
    using namespace SCC;
    scanf("%d%d", &N, &M);
    V = N;
    for (int i = 0; i < M; i++) scanf("%d%d", &A[i], &B[i]);
    for (int i = 0; i < M; i++) {
        add_edge(A[i] - 1, B[i] - 1);
    }
    int n = scc();
    int num = 0;
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < V; i++) ++cnt[cmp[i]];
    for (int i = 0; i < n; i++)
        if (cnt[i] > 1) num++;
    printf("%d\n", num);
}
/*
    Example3
    求能被所有点到达的点的数量
*/
const int MAX_M = 50000;
```

```
int N, M;
int A[MAX_M], B[MAX_M];
void solve() {
    using namespace SCC;
    scanf("%d%d", &N, &M);
    V = N;
    for (int i = 0; i < M; i++) scanf("%d%d", &A[i], &B[i]);
    for (int i = 0; i < M; i++) {
        add_edge(A[i] - 1, B[i] - 1);
    }
    int n = scc();
    int u = 0, num = 0;
    for (int v = 0; v < V; v++) {
        if (cmp[v] == n - 1) {
            u = v;
            num++;
        }
    }
    memset(used, 0, sizeof(used));
    rdfs(u, 0);
    for (int v = 0; v < V; v++) {
        if (!used[v]) {
            num = 0;
            break;
        }
    }
    printf("%d\n", num);
}
```

3.16 2-SAT

```
/*
    2-SAT
*/
namespace SCC {
const int MAX_V = 10000;
int V;               //顶点数
vector<int> G[MAX_V]; //图的邻接表表示
vector<int> rG[MAX_V]; //反向图
vector<int> vs;      //后序遍历
bool used[MAX_V];
int cmp[MAX_V]; //所属强连通分量的拓扑序
void Initialize() {
    for (int i = 0; i < V; i++) G[i].clear();
    for (int i = 0; i < V; i++) rG[i].clear();
```

```
}
void add_edge(int from, int to) {
    G[from].push_back(to);
    rG[to].push_back(from);
}
void dfs(int v) {
    used[v] = 1;
    for (int i = 0; i < G[v].size(); i++) {
        if (!used[G[v][i]]) dfs(G[v][i]);
    }
    vs.push_back(v);
}
void rdfs(int v, int k) {
    used[v] = 1;
    cmp[v] = k;
    for (int i = 0; i < rG[v].size(); i++) {
        if (!used[rG[v][i]]) rdfs(rG[v][i], k);
    }
}
int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for (int v = 0; v < V; v++) {
        if (!used[v]) dfs(v);
    }
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = vs.size() - 1; i >= 0; i--) {
        if (!used[vs[i]]) rdfs(vs[i], k++);
    }
    return k;
}
} // namespace SCC
using namespace SCC;
int N, M;
int x, y, c;
char op[10];
int solve() {
    V = N * 2;
    // 0~N-1 表示 x取0
    // N~N+N-1 表示 x取1
    for (int i = 0; i < M; i++) {
        scanf("%d%d%d%s", &x, &y, &c, op);
        if (op[0] == 'A') {
            // x and y = 0
            if (c == 0) {
```

```
            add_edge(y + N, x);
            add_edge(x + N, y);
        }
        // x and y = 1
        else {
            add_edge(x, x + N);
            add_edge(y, y + N);
        }
    } else if (op[0] == 'O') {
        // x or y = 0
        if (c == 0) {
            add_edge(x + N, x);
            add_edge(y + N, y);
        }
        // x or y = 1
        else {
            add_edge(x, y + N);
            add_edge(y, x + N);
        }
    } else if (op[0] == 'X') {
        // x xor y = 0
        if (c == 0) {
            add_edge(x, y);
            add_edge(y, x);
            add_edge(x + N, y + N);
            add_edge(y + N, x + N);
        }
        // x xor y = 1
        else {
            add_edge(x, y + N);
            add_edge(y, x + N);
            add_edge(x + N, y);
            add_edge(y + N, x);
        }
    }
}
// 判断满足条件下解集是否存在
int n = scc();
int flag = 1;
for (int i = 0; i < N; i++)
    if (cmp[i] == cmp[i + N]) flag = 0;
puts(flag ? "YES" : "NO");
}
```

3.17 边双连通分量

```
/*
    边双连通分量
    cut[i]表示输入的第i条边是否是桥
    cnt表示边双连通分量的个数
    from[i]表示i所属的边双连通分量编号
*/
using namespace std;
const int N = 5010, M = 10010;
int e[M][2], cut[M], g[N], v[M << 1], nxt[M << 1], ed = 1;
int f[N], dfn[N], low[N], num, cnt, from[N], d[N];
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++num;
    for (int i = g[x]; i; i = nxt[i])
        if (!dfn[v[i]]) {
            f[v[i]] = i >> 1, tarjan(v[i]);
            if (low[x] > low[v[i]]) low[x] = low[v[i]];
        } else if (f[x] != (i >> 1) && low[x] > dfn[v[i]])
            low[x] = dfn[v[i]];
    if (f[x] && low[x] == dfn[x]) cut[f[x]] = 1;
}
void dfs(int x, int y) {
    from[x] = y;
    for (int i = g[x]; i; i = nxt[i])
        if (!from[v[i]] && !cut[i >> 1]) dfs(v[i], y);
}
/*
    Example
    给出一张图，问增加几条边，
    使得整张图构成双连通分量
*/
int n, m;
int main() {
    while (~scanf("%d%d", &n, &m)) {
        memset(g, 0, sizeof(g));
        memset(d, 0, sizeof(d));
        memset(from, 0, sizeof(from));
        memset(f, 0, sizeof(f));
        memset(cut, 0, sizeof(cut));
        memset(dfn, 0, sizeof(dfn));
```

```
        num = 0;
        ed = 1; // 求边双连通分量时，ed一定要为1
        for (int i = 1; i <= m; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            e[i][0] = u;
            e[i][1] = v;
            add(u, v);
            add(v, u);
        }
        tarjan(1);
        cnt = 0;
        for (int i = 1; i <= n; i++)
            if (!from[i]) dfs(i, ++cnt);
        for (int i = 1; i <= m; i++) {
            if (from[e[i][0]] != from[e[i][1]]) {
                d[from[e[i][0]]]++;
                d[from[e[i][1]]]++;
            }
        }
        int res = 0;
        for (int i = 1; i <= n; i++)
            if (d[i] == 1) res++;
        printf("%d\n", (res + 1) / 2);
    }
    return 0;
}
```

3.18 点双连通分量

```
/*
    点双连通分量
*/
int dfn[N], low[N], num, cut[N], q[N], t;
void tarjan(int x) {
    dfn[x] = low[x] = ++num, q[++t] = x;
    for (int i = g[x]; i; i = nxt[i])
        if (!dfn[v[i]]) {
            int y = v[i];
            tarjan(y);
            if (low[x] > low[y]) low[x] = low[y];
            if (dfn[x] <=
                low[y]) { // x是割点，接下来一行输出所有该点双连通分量内的点
                cut[x] = 1;
                while (q[t] != x) printf("%d ", q[t--]);
```

```
            printf("%d\n", x);
        }
    } else if (low[x] > dfn[v[i]])
        low[x] = dfn[v[i]];
}
```

3.19 支配树

```
/* Dominator Tree
 * dfn[x]: x的DFS序。
 * id[x]: DFS序第x个点的id。
 * gd[x]: DFS序第x个点在Dominator Tree上的孩子列表。
 * idom[x]: DFS序第x个点在Dominator Tree上的父亲。
 * sd[x]: DFS序第x个点的半必经点。
 * id[idom[dfn[x]]]: x的最近必经点
 */
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100100, M = 200010;
typedef long long LL;
int n, m, i, x, y;
int g1[N], g2[N], gd[N], v[M * 3 + N], nxt[M * 3 + N], ed;
int cnt, dfn[N], id[N], fa[N], f[N], mn[N], sd[N], idom[N];
void add(int* g, int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
int F(int x) {
    if (f[x] == x) return x;
    int y = F(f[x]);
    if (sd[mn[x]] > sd[mn[f[x]]]) mn[x] = mn[f[x]];
    return f[x] = y;
}
void dfs(int x) {
    id[dfn[x] = ++cnt] = x;
    for (int i = g1[x]; i; i = nxt[i])
        if (!dfn[v[i]]) dfs(v[i]), fa[dfn[v[i]]] = dfn[x];
}
void tarjan(int S) {
    int i, j, k, x;
    for (cnt = 0, i = 1; i <= n; i++)
        gd[i] = dfn[i] = id[i] = fa[i] = idom[i] = 0, f[i] = sd[i] = mn[i] = i;
    dfs(S);
```

```
    for (i = cnt; i > 1; i--) {
        for (j = g2[id[i]]; j; j = nxt[j])
            if (dfn[v[j]])
                F(k = dfn[v[j]]), sd[i] = sd[i] < sd[mn[k]] ? sd[i] : sd[mn[k]];
        add(gd, sd[i], i);
        for (j = gd[f[i] = x = fa[i]]; j; j = nxt[j])
            F(k = v[j]), idom[k] = sd[mn[k]] < x ? mn[k] : x;
        gd[x] = 0;
    }
    for (i = 2; i <= cnt; add(gd, idom[i], i), i++)
        if (idom[i] != sd[i]) idom[i] = idom[idom[i]];
}
/*
 * 给出一张有向图，n点为消息源，求每个点i如果要接受n点的信息的必经点权id和
 * 在Dominator Tree上做dfs即可，计算id和
 */
LL ans[N];
void Get_ans(int x) {
    ans[id[x]] += id[x];
    for (int i = gd[x]; i; i = nxt[i]) {
        ans[id[v[i]]] += ans[id[x]];
        Get_ans(v[i]);
    }
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (ed = 0, i = 1; i <= n; i++) g1[i] = g2[i] = 0;
        while (m--) scanf("%d%d", &x, &y), add(g1, x, y), add(g2, y, x);
        tarjan(n);
        memset(ans, 0, sizeof(ans));
        Get_ans(1);
        for (int i = 1; i <= n; i++) printf("%lld%c", ans[i], " \n"[i == n]);
    }
    return 0;
}
```

3.20 最近公共祖先

```
/*
    最近公共祖先
    要保证一整棵树连通
    以1为根节点(注意dph[1]=1，否则深度判定会出错)
*/
namespace LCA {
const int N = 50010, LEV = 20;
```

```
int dph[N];
int fa[N][LEV];
vector<int> G[N];
void add(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
}
void dfs(int rt, int f) {
    for (int i = 1; i < LEV; i++) {
        if (dph[rt] - 1 < (1 << i)) break;
        fa[rt][i] = fa[fa[rt][i - 1]][i - 1];
    }
    for (int v : G[rt]) {
        if (v == f) continue;
        dph[v] = dph[rt] + 1;
        fa[v][0] = rt;
        dfs(v, rt);
    }
}
int lca(int a, int b) {
    if (dph[a] < dph[b]) swap(a, b);
    int t = dph[a] - dph[b];
    for (int i = 0; i < LEV; i++)
        if ((1 << i) & t) a = fa[a][i];
    for (int i = LEV - 1; i >= 0; i--) {
        if (fa[a][i] != fa[b][i]) a = fa[a][i], b = fa[b][i];
    }
    if (a != b) return fa[a][0];
    return a;
}
// 从节点x往上走l个节点的位置
int GoUp(int x, int l) {
    for (int i = 0; i < LEV; i++) {
        if ((1 << i) & l) x = fa[x][i];
    }
    return x;
}
void Initialize(int n) {
    for (int i = 1; i <= n; i++) G[i].clear();
    memset(fa, 0, sizeof(fa));
    dph[1] = 1;
}
} // namespace LCA
/*
    查询路径最大值
    (以下模板不用保证dph[1]=1)
```

```
*/
int head[M << 1], u[M << 1], v[M << 1], nxt[M << 1];
inline void init() {
    ed = 0;
    memset(head, -1, sizeof(head));
}
inline void add(int a, int b, LL c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
const int LEV = 20;
LL fw[N][LEV];
int fa[N][LEV], dph[N];
void dfs(int rt, int fx) {
    vis[rt] = 1;
    for (int i = 1; i < LEV; i++) {
        fa[rt][i] = fa[fa[rt][i - 1]][i - 1];
        fw[rt][i] = max(fw[rt][i - 1], fw[fa[rt][i - 1]][i - 1]);
    }
    for (int i = head[rt]; ~i; i = nxt[i]) {
        if (v[i] == fx) continue;
        fa[v[i]][0] = rt;
        fw[v[i]][0] = w[i];
        dph[v[i]] = dph[rt] + 1;
        dfs(v[i], rt);
    }
}
LL Cal(int a, int b) {
    LL res = 0;
    if (dph[a] < dph[b]) swap(a, b);
    for (int i = LEV - 1; i >= 0; i--) {
        if (dph[fa[a][i]] >= dph[b]) {
            res = max(res, fw[a][i]);
            a = fa[a][i];
        }
    }
    for (int i = LEV - 1; i >= 0; i--) {
        if (fa[a][i] != fa[b][i]) {
            res = max(res, max(fw[a][i], fw[b][i]));
            a = fa[a][i];
            b = fa[b][i];
        }
    }
    if (a != b) res = max(res, max(fw[a][0], fw[b][0]));
    return res;
```

```
}
/*
    对于森林的处理方式
*/
void build() {
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) dfs(i, 0);
    }
}
```

3.21 稳定婚姻系统

```
/* 稳定婚姻系统
 * 给定n对夫妻关系，再给定m对情人关系。
 * 如果在第i对夫妻离婚的前提下，
 * 这些人仍然能组成n对夫妻，就称i这个婚姻是不稳定的。
 * 要求判断所有的婚姻稳不稳定
 * O(n+m)
 */
const int N = 4010, M = 20010;
bool v[N];
int idx, cnt, dfn[N], low[N], c[N], s[N], sz[N], h[N], p[M], nxt[M], n, m, i, x;
map<string, int> g, b;
string name;
inline void _min(int& x, int y) {
    if (y < x) x = y;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++idx;
    v[s[++s[0]] = x] = 1;
    for (int k = h[x]; k; k = nxt[k]) {
        if (!dfn[p[k]])
            tarjan(p[k]), _min(low[x], low[p[k]]);
        else if (v[p[k]])
            _min(low[x], dfn[p[k]]);
    }
    if (dfn[x] == low[x])
        for (++sz[c[s[s[0]]] = ++cnt], v[s[s[0]]] = 0; s[s[0]--] != x;
             ++sz[c[s[s[0]]] = cnt], v[s[s[0]]] = 0)
            ;
}
int main() {
    cin >> n;
    for (i = 1; i <= n; i++) {
```

```
        cin >> name;
        g[name] = i;
        cin >> name;
        b[name] = i;
    }
    cin >> m;
    for (i = 1; i <= m; i++) {
        cin >> name;
        x = g[name];
        cin >> name;
        p[i] = b[name];
        nxt[i] = h[x];
        h[x] = i;
    }
    for (i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i);
    for (i = 1; i <= n; i++) puts(sz[c[i]] == 1 ? "Safe" : "Unsafe");
    return 0;
}
```

3.22 最大团算法

```
/* 极大团
 * 定义：团是G的一个完全子图，不能被更大的团所包含的团称为极大团
 * 用法：Initialize(n)
 *       Add_Edge()
 *       Dfs(1,0,n,0)
 */
namespace BronKerbosch {
const int N = 200;
int G[N][N], Allow[N][N], Forbid[N][N], Num[N][N], Ans;
void Initialize(int n) {
    Ans = 0;
    for (int i = 1; i <= n; i++) Allow[1][i] = i;
    memset(G, 0, sizeof(G));
}
void Add_Edge(int x, int y) { G[x][y] = G[y][x] = 1; }
void Dfs(int x, int Nn, int An, int Fn) {
    if (!An && !Fn) {
        // Nn为极大团的大小，当两个集合同时为0时表示找到了一个极大团
        Ans = max(Ans, Nn);
        return;
        // 可以用 Ans++ 统计极大团的数量
    }
    if (!An) return;
```

```
    int key = Allow[x][1];
    for (int j = 1; j <= An; j++) {
        int v = Allow[x][j], tAn = 0, tFn = 0;
        if (G[key][v]) continue;
        for (int i = 1; i <= Nn; i++) Num[x + 1][i] = Num[x][1];
        Num[x + 1][Nn + 1] = v;
        for (int i = 1; i <= An; i++)
            if (G[v][Allow[x][i]]) Allow[x + 1][++tAn] = Allow[x][i];
        for (int i = 1; i <= Fn; i++)
            if (G[v][Forbid[x][i]]) Forbid[x + 1][++tFn] = Forbid[x][i];
        Dfs(x + 1, Nn + 1, tAn, tFn);
        Allow[x][j] = 0;
        Forbid[x][++Fn] = v;
    }
}
} // namespace BronKerbosch
```

3.23 最小环算法

```
// 有向图最小环
void solve() {
#define rep(i, n) for (int i = 1; i <= n; i++)
    rep(i, n) rep(j, n) d[i][j] = INF;
    while (m--) {
        scanf("%d%d%d", &x, &y, &z);
        if (z < d[x][y]) d[x][y] = z;
    }
    rep(k, n) rep(i, n) rep(j, n) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    rep(i, n) ans = min(ans, d[i][i]);
}
// 无向图最小环
void solve() {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) g[i][j] = d[i][j] = inf;
    while (m--) {
        scanf("%d%d%d", &x, &y, &z);
        if (z < g[x][y]) g[x][y] = g[y][x] = d[x][y] = d[y][x] = z;
    }
    for (ans = inf, k = 1; k <= n; k++) {
        for (i = 1; i < k; i++)
            for (j = i + 1; j < k; j++)
                ans = min(ans, d[i][j] + g[i][k] + g[k][j]);
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    }
```

```
}
```

3.24 最小点覆盖

```
/*
    最小点覆盖
    邻接矩阵给出覆盖关系
*/
#include <bitset>
#include <cstdio>
using namespace std;
const int N = 80;
bitset<N> a[N];
char s[N];
int n, limit, ans[N], cnt = 1;
bool dfs(int d, int pos, bitset<N> mp) {
    if (d >= limit) return mp.count() == n;
    for (int i = pos; i < n; i++)
        if (dfs(d + 1, (ans[d] = i + 1), mp | a[i])) return 1;
    return 0;
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; i++) {
            a[i].reset();
            scanf("%s", s);
            for (int j = 0; j < n; j++)
                if (s[j] == '1' || i == j) a[i].set(j);
        }
        for (limit = 1; limit <= 80; limit++)
            if (dfs(0, 0, bitset<N>())) break;
        printf("Case %d: %d", cnt++, limit);
        for (int i = 0; i < limit; i++) printf(" %d", ans[i]);
        puts("");
    }
    return 0;
}
```

3.25 补图 BFS

```
/*
    补图BFS
    给定一张n个点m条边的图
```

```
    求对于完全图的补图的连通块
*/
using namespace std;
const int N = 100010;
int f[N], n, m, vis[N], ans, cnt[N];
vector<int> v[N];
int sf(int x) { return f[x] == x ? x : f[x] = sf(f[x]); }
void bfs(int st) {
    queue<int> q;
    q.push(st);
    cnt[++ans] = 1;
    while (q.size()) {
        int x = q.front();
        q.pop();
        f[x] = sf(x + 1);
        for (int i = 0; i < v[x].size(); i++) vis[v[x][i]] = x;
        for (int i = sf(1); i <= n; i = sf(i + 1))
            if (vis[i] != x) {
                cnt[ans]++;
                f[i] = sf(i + 1);
                q.push(i);
            }
    }
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        ans = 0;
        while (m--) {
            int x, y;
            scanf("%d%d", &x, &y);
            v[x].push_back(y);
            v[y].push_back(x);
        }
        for (int i = 1; i <= n + 1; i++) f[i] = i;
        for (int i = 1; i <= n; i = sf(i + 1)) bfs(i);
        printf("%d\n", ans);
        sort(cnt + 1, cnt + ans + 1);
        for (int i = 1; i <= ans; i++) printf("%d ", cnt[i]);
        puts("");
    }
    return 0;
}
```

3.26 完全图计数

```
// 统计每个连通分量都是完全图的方案数
#include <cstdio>
const int N = 731, P = 999999599;
int n, m, i, j, f[731], g[200005];
int main() {
    for (f[1] = 1, f[2] = 2, f[3] = 5, f[4] = 7, i = 5; i < N; i++)
        f[i] = 3 + 2 * f[i - 2] - f[i - 4];
    for (scanf("%d%d", &n, &m), g[0] = i = 1; i <= n; i++) {
        for (j = 1; f[j] <= i; j++) {
            if ((j + 1) >> 1 & 1)
                g[i] = (g[i] + g[i - f[j]]) % (P - 1);
            else
                g[i] = (g[i] - g[i - f[j]]) % (P - 1);
        }
    }
    for (i = (g[n] + P - 1) % (P - 1), j = 1; i; i >>= 1, m = 1LL * m * m % P)
        if (i & 1) j = 1LL * j * m % P;
    return printf("%d", j), 0;
}
```

3.27 次小生成树

```
/*
    次小生成树
    复杂度O(nm)
    Example: 判定是否存在唯一最小生成树
*/
const int N = 10005;
int cnt, n, m, f[N], mst[N], T, tot, ans;
struct data {
    int l, r, c;
} e[N];
int sf(int x) { return x == f[x] ? x : f[x] = sf(f[x]); }
bool cmp(data a, data b) { return a.c < b.c; }
int init() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) scanf("%d%d%d", &e[i].l, &e[i].r, &e[i].c);
}
void kruskal() {
    sort(e + 1, e + m + 1, cmp);
    for (int i = 0; i <= n; i++) f[i] = i;
    cnt = ans = 0;
    for (int i = 1; i <= m && cnt < n - 1; i++) {
        if (sf(e[i].l) != sf(e[i].r)) {
```

```
            f[f[e[i].l]] = f[e[i].r];
            ans += e[i].c;
            mst[++cnt] = i;
        }
    }
    for (int i = 1; i < n; i++) {
        tot = cnt = 0;
        for (int j = 0; j <= n; j++) f[j] = j;
        for (int j = 1; j <= m; j++) {
            if (j == mst[i]) continue;
            if (sf(e[j].l) != sf(e[j].r)) {
                f[f[e[j].l]] = f[e[j].r];
                tot += e[j].c;
                cnt++;
            }
        }
        if (cnt != n - 1) continue;
        if (tot == ans) {
            puts("Not Unique!");
            return;
        }
    }
    printf("%d\n", ans);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        init();
        kruskal();
    }
    return 0;
}
```

3.28 斯坦纳树

```
// 斯坦纳树：只用包含关键点的最小生成树
/*
 * Problem: 给出一张图，要求i属于1~d号点与n-i+1号点对应连通，求最小边权和
 * Solution: f[i][msk]表示i为根，关键点连通状态为msk时候的最小代价
 * Ans[msk]表示连通态为msk时最小代价，如果对应点同时连通或不连通则可以更新
 */
// Demo1
#include <bits/stdc++.h>
using namespace std;
const int N = 10005;
```

```
const int MOD = 1e9 + 7;
int n, m, d, x, y, z;
int nxt[N << 1], g[N], v[N << 1], w[N << 1], tot;
int a[N], All, f[N][(1 << 8) + 10], INF, vis[N];
int Ans[(1 << 8) + 10];
void add_edge(int a, int b, int c) {
    nxt[++tot] = g[a];
    g[a] = tot;
    v[tot] = b;
    w[tot] = c;
}
void Add(int a, int b, int c) {
    add_edge(a, b, c);
    add_edge(b, a, c);
}
typedef pair<int, int> seg;
priority_queue<seg, vector<seg>, greater<seg>> q;
namespace Steiner {
void init() {
    memset(f, 63, sizeof(f));
    INF = f[0][0];
    int num = 0;
    for (int i = 1; i <= d; i++) f[i][1 << num] = 0, num++;
    for (int i = n - d + 1; i <= n; i++) f[i][1 << num] = 0, num++;
    All = (1 << num) - 1;
}
void Dijkstra(int msk) {
    while (q.size()) {
        int x = q.top().second;
        q.pop();
        for (int e = g[x]; e; e = nxt[e]) {
            int y = v[e];
            if (f[y][msk] > f[x][msk] + w[e]) {
                f[y][msk] = f[x][msk] + w[e];
                if (!vis[y]) {
                    vis[y] = 1;
                    q.push(make_pair(f[y][msk], y));
                }
            }
        }
        vis[x] = 0;
    }
}
void Deal() {
    for (int msk = 0; msk <= All; msk++) {
        for (int i = 1; i <= n; i++) {
```

```
            for (int sub = msk; sub; sub = (sub - 1) & msk)
                f[i][msk] = min(f[i][msk], f[i][sub] + f[i][msk ^ sub]);
            if (f[i][msk] != INF) {
                q.push(make_pair(f[i][msk], i));
                vis[i] = 1;
            }
        }
        Dijkstra(msk);
    }
}
} // namespace Steiner
bool Check(int msk) {
    for (int i = 0, j = (d << 1) - 1; i < d; i++, j--)
        if (((msk & (1 << i)) == 0) != ((msk & (1 << j)) == 0)) return 0;
    return 1;
}
int main() {
    scanf("%d%d%d", &n, &m, &d);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &x, &y, &z);
        Add(x, y, z);
    }
    Steiner::init();
    Steiner::Deal();
    memset(Ans, 63, sizeof(Ans));
    for (int msk = 0; msk <= All; msk++)
        if (Check(msk)) {
            for (int i = 1; i <= n; i++) Ans[msk] = min(Ans[msk], f[i][msk]);
        }
    for (int msk = 0; msk <= All; msk++)
        for (int sub = msk; sub; sub = (sub - 1) & msk)
            Ans[msk] = min(Ans[msk], Ans[sub] + Ans[msk ^ sub]);
    if (Ans[All] == INF)
        printf("-1");
    else
        printf("%d", Ans[All]);
    return 0;
}
// Demo2
#include <bits/stdc++.h>
const int MOD = 1e9 + 7;
using Info = std::pair<int, int>;
template <typename T>
using PQ = std::priority_queue<T, std::vector<T>, std::greater<T>>;
static const Info ONE{1, 1};
int lowbit(int msk) { return msk & -msk; }
```

```
Info add(const Info& a, const Info& b) {
    return {a.first + b.first, 1LL * a.second * b.second % MOD};
}
void update(Info& x, const Info& a) {
    if (a.first < x.first) {
        x = {a.first, 0};
    }
    if (a.first == x.first) {
        x.second += a.second;
        if (x.second >= MOD) {
            x.second -= MOD;
        }
    }
}
int n, m, l;
int Ans[(1 << 8) + 10];
bool Check(int msk) {
    for (int i = 0, j = (l << 1) - 1; i < l; i++, j--)
        if (((msk & (1 << i)) == 0) != ((msk & (1 << j)) == 0)) return 0;
    return 1;
}
int main() {
    while (~scanf("%d%d%d", &n, &m, &l)) {
        memset(Ans, 63, sizeof(Ans));
        int INF = Ans[0];
        std::vector<std::vector<int>> graph(n);
        std::vector<std::vector<int>> W(n);
        for (int i = 0, a, b, c; i < m; i++) {
            scanf("%d%d%d", &a, &b, &c);
            a--, b--;
            graph[a].push_back(b);
            W[a].push_back(c);
            graph[b].push_back(a);
            W[b].push_back(c);
        }
        int All = (1 << (l << 1)) - 1;
        std::vector<std::vector<Info>> dp(All + 1,
                                          std::vector<Info>(n, Info{INF, 0})),
            merged(n, std::vector<Info>(All + 1, Info{INF, 0}));
        int num = 0;
        for (int i = 0; i < l; i++) {
            dp[1 << num][i] = {0, 1};
            num++;
        }
        for (int i = n - l; i < n; i++) {
            dp[1 << num][i] = {0, 1};
```

```
        num++;
    }
    for (int msk = 0; msk < 1 << (l << 1); msk++) {
        for (int u = 0; u < n; u++) {
            auto& ref = merged.at(u);
            for (int subset = msk; subset > 0; subset = subset - 1 & msk) {
                if (lowbit(subset) == lowbit(msk)) {
                    update(ref.at(msk),
                           add(dp.at(subset).at(u), ref.at(msk ^ subset)));
                }
            }
        }
        for (int u = 0; u < n; u++) {
            for (int i = 0; i < graph[u].size(); i++) {
                int v = graph[u][i];
                int w = W[u][i];
                update(dp.at(msk).at(v),
                       add(merged.at(u).at(msk), Info{w, 1}));
            }
        }
        auto& ref = dp.at(msk);
        PQ<std::pair<int, int>> pq;
        for (int u = 0; u < n; u++) {
            pq.emplace(ref.at(u).first, u);
        }
        while (!pq.empty()) {
            auto top = pq.top();
            pq.pop();
            int u = top.second;
            if (top.first == ref.at(u).first) {
                for (int i = 0; i < graph[u].size(); i++) {
                    int v = graph[u][i];
                    int w = W[u][i];
                    Info todo = add(ref.at(u), Info{w, 1});
                    if (todo.first < ref.at(v).first) {
                        pq.emplace(todo.first, v);
                    }
                    update(ref.at(v), todo);
                }
            }
        }
        for (int u = 0; u < n; u++) {
            update(merged.at(u).at(msk), dp.at(msk).at(u));
        }
    }
    for (int msk = 0; msk <= All; msk++)
```

```
            if (Check(msk)) {
                for (int i = 0; i < n; i++) {
                    int tmp = merged.at(i).at(msk).first;
                    Ans[msk] = std::min(Ans[msk], tmp);
                }
            }
        for (int msk = 0; msk <= All; msk++)
            for (int sub = msk; sub; sub = (sub - 1) & msk)
                Ans[msk] = std::min(Ans[msk], Ans[sub] + Ans[msk ^ sub]);
        if (Ans[All] == INF)
            printf("-1");
        else
            printf("%d", Ans[All]);
    }
    return 0;
}
/*
 * 超级源点题
 * 给出一些庙宇的位置，以及另一些位置，每个位置都可以打井，打井的费用不同，
 * 现在这些位置不连通，让两个位置连通需要修路，给出修每条路的代价，
 * 现在问这些庙宇都能喝上水(该处是水井或者能到有水井的地方)需要的最小总代价
 *
 * 建立一个超级源点表示水源，打井的代价转化为该地到水源的路的代价
 * 那么题目转化为将水源与所有庙宇连在一起需要的最小代价，斯坦纳树求解即可
 */
#include <bits/stdc++.h>
const int MOD = 1e9 + 7;
using Info = std::pair<int, int>;
template <typename T>
using PQ = std::priority_queue<T, std::vector<T>, std::greater<T>>;
static const Info ONE{1, 1};
int lowbit(int msk) { return msk & -msk; }
Info add(const Info& a, const Info& b) {
    return {a.first + b.first, 1LL * a.second * b.second % MOD};
}
void update(Info& x, const Info& a) {
    if (a.first < x.first) {
        x = {a.first, 0};
    }
    if (a.first == x.first) {
        x.second += a.second;
        if (x.second >= MOD) {
            x.second -= MOD;
        }
    }
}
```

```
int n, m, l;
int main() {
    while (~scanf("%d%d%d", &n, &m, &l)) {
        std::vector<std::vector<int>> graph(n + m + 1);
        std::vector<std::vector<int>> W(n + m + 1);
        for (int i = 1, c; i <= n + m; i++) {
            scanf("%d", &c);
            graph[i].push_back(0);
            W[i].push_back(c);
            graph[0].push_back(i);
            W[0].push_back(c);
        }
        int INF = 1 << 28;
        for (int i = 0, a, b, c; i < l; i++) {
            scanf("%d%d%d", &a, &b, &c);
            graph[a].push_back(b);
            W[a].push_back(c);
            graph[b].push_back(a);
            W[b].push_back(c);
        }
        int All = (1 << (n + 1)) - 1;
        std::vector<std::vector<Info>> dp(
            All + 1, std::vector<Info>(n + m + 1, Info{INF, 0})),
            merged(n + m + 1, std::vector<Info>(All + 1, Info{INF, 0}));
        int num = 0;
        for (int i = 0; i <= n; i++) {
            dp[1 << num][i] = {0, 1};
            num++;
        }
        for (int msk = 0; msk < 1 << (n + 1); msk++) {
            for (int u = 0; u <= n + m; u++) {
                auto& ref = merged.at(u);
                for (int subset = msk; subset > 0; subset = subset - 1 & msk) {
                    if (lowbit(subset) == lowbit(msk)) {
                        update(ref.at(msk),
                               add(dp.at(subset).at(u), ref.at(msk ^ subset)));
                    }
                }
            }
            for (int u = 0; u <= n + m; u++) {
                for (int i = 0; i < graph[u].size(); i++) {
                    int v = graph[u][i];
                    int w = W[u][i];
                    update(dp.at(msk).at(v),
                           add(merged.at(u).at(msk), Info{w, 1}));
                }
```

```
            }
            auto& ref = dp.at(msk);
            PQ<std::pair<int, int>> pq;
            for (int u = 0; u <= n + m; u++) {
                pq.emplace(ref.at(u).first, u);
            }
            while (!pq.empty()) {
                auto top = pq.top();
                pq.pop();
                int u = top.second;
                if (top.first == ref.at(u).first) {
                    for (int i = 0; i < graph[u].size(); i++) {
                        int v = graph[u][i];
                        int w = W[u][i];
                        Info todo = add(ref.at(u), Info{w, 1});
                        if (todo.first < ref.at(v).first) {
                            pq.emplace(todo.first, v);
                        }
                        update(ref.at(v), todo);
                    }
                }
            }
            for (int u = 0; u <= n + m; u++) {
                update(merged.at(u).at(msk), dp.at(msk).at(u));
            }
        }
        int ans = INF;
        for (int i = 0; i <= n + m; i++) {
            int tmp = merged.at(i).at(All).first;
            ans = std::min(ans, tmp);
        }
        printf("%d\n", ans);
    }
    return 0;
}
// 给出一张图，求斯坦纳树的方案数
#include <bits/stdc++.h>
const int MOD = 1e9 + 7;
using Info = std::pair<int, int>;
template <typename T>
using PQ = std::priority_queue<T, std::vector<T>, std::greater<T>>;
static const Info ONE{1, 1};
int lowbit(int msk) { return msk & -msk; }
Info add(const Info& a, const Info& b) {
    return {a.first + b.first, 1LL * a.second * b.second % MOD};
}
```

```
void update(Info& x, const Info& a) {
    if (a.first < x.first) {
        x = {a.first, 0};
    }
    if (a.first == x.first) {
        x.second += a.second;
        if (x.second >= MOD) {
            x.second -= MOD;
        }
    }
}
int main() {
    int n, m, l;
    while (~scanf("%d%d%d", &n, &m, &l)) {
        std::vector<std::vector<int>> graph(n);
        for (int i = 0, a, b; i < m; i++) {
            scanf("%d%d", &a, &b);
            a--, b--;
            graph[a].push_back(b);
            graph[b].push_back(a);
        }
        if (l == 1) {
            puts("1");
            continue;
        }
        l--;
        std::vector<std::vector<Info>> dp(1 << l,
                                    std::vector<Info>(n, Info{m + 1, 0})),
            merged(n, std::vector<Info>(1 << l, Info{m + 1, 0}));
        int root = l;
        for (int i = 0; i < l; i++) {
            dp[1 << i][i] = {0, 1};
        }
        for (int msk = 0; msk < 1 << l; msk++) {
            for (int u = 0; u < n; u++) {
                auto& ref = merged.at(u);
                for (int subset = msk; subset > 0; subset = subset - 1 & msk) {
                    if (lowbit(subset) == lowbit(msk)) {
                        update(ref.at(msk),
                               add(dp.at(subset).at(u), ref.at(msk ^ subset)));
                    }
                }
            }
            for (int u = 0; u < n; u++) {
                for (int v : graph[u]) {
                    update(dp.at(msk).at(v), add(merged.at(u).at(msk), ONE));
```

```
            }
        }
        auto& ref = dp.at(msk);
        PQ<std::pair<int, int>> pq;
        for (int u = 0; u < n; u++) {
            pq.emplace(ref.at(u).first, u);
        }
        while (!pq.empty()) {
            auto top = pq.top();
            pq.pop();
            int u = top.second;
            if (top.first == ref.at(u).first) {
                for (int v : graph.at(u)) {
                    Info todo = add(ref.at(u), ONE);
                    if (todo.first < ref.at(v).first) {
                        pq.emplace(todo.first, v);
                    }
                    update(ref.at(v), todo);
                }
            }
        }
        for (int u = 0; u < n; u++) {
            update(merged.at(u).at(msk), dp.at(msk).at(u));
        }
    }
    printf("%d\n", merged.at(root).at((1 << l) - 1).second);
  }
  return 0;
}
```

3.29 黑白树最小生成树

```
/*
    黑白边最小生成树
    给你一个无向带权连通图，每条边是黑色或白色。
    让你求一棵最小权的恰好有need条白色边的生成树，题目保证一定有解
    我们对所有的白色边加一个权值，那么排序后做mst选取的白色边数量增减性单调，
    对于增加的权值进行二分，验证能否满足要求即可。
    单点度限制最小生成树可以将其点相连边染色后转化为黑白边最小生成树
*/
const int N = 100010;
struct data {
    int a, b, u, c;
} p[N];
bool cmp(data a, data b) { return a.u < b.u; }
```

```
int ans, res, v, e, need, f[N];
int sf(int x) { return f[x] == x ? x : f[x] = sf(f[x]); }
bool check(int x) {
    int cnt = 0;
    res = 0;
    for (int i = 0; i < v; i++) f[i] = i;
    for (int i = 1; i <= e; i++) {
        if (!p[i].c) p[i].u += x;
    }
    sort(p + 1, p + e + 1, cmp);
    for (int i = 1; i <= e; i++) {
        if (!p[i].c) p[i].u -= x;
    }
    for (int i = 1; i <= e; i++) {
        if (p[i].u < x) continue;
        if (sf(p[i].a) != sf(p[i].b)) {
            res += p[i].u;
            f[sf(p[i].a)] = sf(p[i].b);
            cnt += (p[i].c == 0);
        }
    }
    return cnt >= need;
}
int main() {
    while (~scanf("%d%d%d", &v, &e, &need)) {
        for (int i = 1; i <= e; i++) {
            scanf("%d%d%d%d", &p[i].a, &p[i].b, &p[i].u, &p[i].c);
        }
        int l = -100, r = 100;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(mid))
                l = mid + 1, ans = res;
            else
                r = mid - 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.30 最小树形图

```
/* 最小树形图
 * 节点: 0~n-1
```

```
 * variable用于计算不定根的最小树形图
 * 使用方法: init()
 *         add(from,to,cost)
 *         dmst(root)
 * 复杂度 O(VE)
 */
const int N = 10050, M = 50050, inf = 0x7fffffff;
struct DMST {
   int n, size, pre[N], id[N], vis[N], in[N];
   struct EDGE {
      int u, v, cost;
      EDGE() {}
      EDGE(int a, int b, int c) : u(a), v(b), cost(c) {}
   } E[M];
   void init(int _n) { n = _n, size = 0; }
   void add(int u, int v, int w) { E[size++] = EDGE(u, v, w); }
   int dmst(int root) {
      int u, v, cnt, ret = 0;
      while (1) {
         for (int i = 0; i < n; i++) in[i] = inf;
         for (int i = 0; i < size; i++) {
            u = E[i].u, v = E[i].v;
            if (E[i].cost < in[v] && u != v) {
               pre[v] = u, in[v] = E[i].cost;
               if (u == root) ROOT = i;
            }
         }
         for (int i = 0; i < n; i++)
            if (i != root && in[i] == inf) return -1;
         cnt = in[root] = 0;
         for (int i = 0; i < n; i++) id[i] = vis[i] = -1;
         for (int i = 0; i < n; i++) {
            ret += in[i], v = i;
            while (vis[v] != i && id[v] == -1 && v != root)
               vis[v] = i, v = pre[v];
            if (v != root && id[v] == -1) {
               for (u = pre[v]; u != v; u = pre[u]) id[u] = cnt;
               id[v] = cnt++;
            }
         }
         if (!cnt) break;
         for (int i = 0; i < n; i++)
            if (id[i] == -1) id[i] = cnt++;
         for (int i = 0; v = E[i].v, i < size; i++) {
            E[i].u = id[E[i].u], E[i].v = id[E[i].v];
            if (E[i].u != E[i].v) E[i].cost -= in[v];
```

```
            }
            n = cnt, root = id[root];
        }
        return ret;
    }
} U;
void variable(int &cost, int &root) {
    for (int i = 0; i < n; i++)
        G.add(st, i, tot); // st=n,tot=sum of Edge Wight+1
    int ans = G.dmst(st);
    if (ans == -1 || ans - tot > tot) return;
    cost = ans - tot, root = ROOT - m;
}
/* 限制n号点父亲编号
 * 给出一个有向图，求1点为根的最大树形图
 * 使得第n个点的直接父亲编号最小
 */
int T, n, m;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        U.init(n);
        while (m--) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            z *= 1000;
            if (y == n) z += n - x;
            U.add(x - 1, y - 1, -z);
        }
        LL ans = -U.dmst(0);
        int fa = n - ans % 1000;
        ans /= 1000;
        printf("%lld %d\n", ans, fa);
    }
    return 0;
}
```

3.31 最小乘积生成树

```
/*
 * 最小乘积生成树
 * 给出一张无向图，每条边上有a,b两个值，求生成树，
 * 使得suma*sumb最小，在满足这个前提下保证suma最小
 */
```

```
const int N = 210, M = 10010;
typedef long long LL;
struct P {
    int x, y;
    P() { x = y = 0; }
    P(int _x, int _y) {
        x = _x;
        y = _y;
    }
    P operator-(const P& rhs) { return P(x - rhs.x, y - rhs.y); }
} l, r;
LL cross(P a, P b) { return (LL)a.x * b.y - (LL)a.y * b.x; }
struct E {
    int x, y, a, b, c;
} e[M];
bool cmp(E a, E b) { return a.c < b.c; }
int n, m, f[N];
LL ans = 0x3f3f3f3f3f3f3f3f, ansx, ansy;
int sf(int x) { return f[x] == x ? x : f[x] = sf(f[x]); }
P kruskal() {
    P p;
    int i;
    sort(e + 1, e + m + 1, cmp);
    for (int i = 1; i <= n; i++) f[i] = i;
    for (int i = 1; i <= m; i++) {
        if (sf(e[i].x) != sf(e[i].y)) {
            f[f[e[i].x]] = f[e[i].y];
            p.x += e[i].a, p.y += e[i].b;
        }
    }
    if ((LL)p.x * p.y <= ans) {
        if (ans == (LL)p.x * p.y) {
            if (p.x < ansx) ansx = p.x, ansy = p.y;
        } else {
            ans = (LL)p.x * p.y;
            ansx = p.x;
            ansy = p.y;
        }
    }
    return p;
}
void work(P l, P r) {
    P t = l - r;
    for (int i = 1; i <= m; i++) e[i].c = cross(P(e[i].a, e[i].b), t);
    P mid = kruskal();
    if (cross(mid - l, r - mid) > 0) work(l, mid), work(mid, r);
```

```
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &e[i].x, &e[i].y, &e[i].a, &e[i].b);
        e[i].x++;
        e[i].y++;
        e[i].c = e[i].a;
    }
    l = kruskal();
    for (int i = 1; i <= m; i++) e[i].c = e[i].b;
    r = kruskal();
    work(l, r);
    printf("%lld %lld\n", ansx, ansy);
    return 0;
}
```

3.32 Kruskal 重构树

```
/*
    Kruskal重构树
    在用kruskal算法做最小生成树的过程中，将边权作为点权，
    当做父节点将两个需要连接的点连起来，
    那么树上每个点的权值都小于父节点的权值，构成一棵单调可二分的树
*/
void ReKruskal(int n, int m) {
    sort(e + 1, e + m + 1, cmp);
    for (int i = 1; i <= n; i++) {
        f[i] = i;
        w[i] = 0;
    }
    R = n;
    for (int i = 1; i <= m; i++) {
        int x = sf(e[i].x), y = sf(e[i].y);
        if (x == y) continue;
        w[++R] = e[i].z;
        f[x] = f[y] = f[R] = R;
        v[R].push_back(x);
        v[R].push_back(y);
    }
    w[0] = N;
}
/*
    Problem:
        给出一个无向连通图，有n个顶点，m条边。
```

```
        有q次询问，每次给出x,y,z，
        最小化从x和y开始，总计访问z个不重复顶点，经过的边的编号的最大值
    Solution:
        二分答案+Kruskal重构树
        对原图建立Kruskal重构树，对重构树建立父节点倍增表，
        对于每个询问，二分最大边编号，倍增找到x和y节点在重构树上对应的小于等于二分值的深度最小的祖先，
        其子树和即为能够访问到的不重复点数量。
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 500010;
vector<int> v[N];
int f[N], w[N], R;
int sf(int x) { return x == f[x] ? x : f[x] = sf(f[x]); }
struct data {
    int x, y, z;
} e[N];
bool cmp(data a,data b){
       return a.z < b.z;
}
void ReKruskal(int n, int m) {
    sort(e + 1, e + m + 1, cmp);
    for (int i = 1; i <= n; i++) {
        f[i] = i;
        w[i] = 0;
    }
    R = n;
    for (int i = 1; i <= m; i++) {
        int x = sf(e[i].x), y = sf(e[i].y);
        if (x == y) continue;
        w[++R] = e[i].z;
        f[x] = f[y] = f[R] = R;
        v[R].push_back(x);
        v[R].push_back(y);
    }
    w[0] = N;
}
int n, m;
const int LOG = 20;
int s[N], fa[N][LOG+1];
void dfs(int x, int fx) {
    fa[x][0] = fx;
    for (int i = 1; i < LOG; i++) fa[x][i] = fa[fa[x][i - 1]][i - 1];
    for (auto y : v[x]) {
        dfs(y, x);
        s[x] += s[y];
```

```
    }
    if (x <= n) s[x] = 1;
}
int find(int x, int W) {
    for (int i = LOG; ~i; i--)
        if (w[fa[x][i]] <= W) x = fa[x][i];
    return x;
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &e[i].x, &e[i].y);
        e[i].z = i;
    }
    ReKruskal(n, m);
    dfs(R, 0);
    int q, x, y, z;
    scanf("%d", &q);
    while (q--) {
        scanf("%d%d%d", &x, &y, &z);
        int ans = m, l = 0, r = m;
        while (l <= r) {
            int mid = (l + r) >> 1;
            int fx = find(x, mid), fy = find(y, mid), Size = s[fx];
            if (fy != fx) Size += s[fy];
            if (Size >= z)
                ans = mid, r = mid - 1;
            else
                l = mid + 1;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.33 完美消除序列 MCS 算法

```
/* MCS求完美消除序列
 * 一个无向图称为弦图当图中任意长度大于3的环都至少有一个弦。
 * 弦图里有团数=最小染色数
 * 最大独立点集数=最小团覆盖数
 * 弦图必存在完美消除序列，弦图的染色数等于完美消除序列点数
 * Example: 用最少的颜色给每个点染色使得弦图中相邻的点染的颜色不同
 * Solution: 通过完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色
 * Example: 最大独立集问题：选择最多的点使得任意两个点不相邻
```

```
 * solution: 通过完美消除序列从前往后能选就选。
 */
bool vis[N];
int n, m, g[N], nxt[M], v[M], ed, i, j, k, col[N], ans;
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void MCS() {
    scanf("%d%d", &n, &m);
    while (m--) scanf("%d%d", &i, &j), add(i, j), add(j, i);
    for (i = n; i; i--) {
        for (k = 0, j = 1; j <= n; j++)
            if (!vis[j] && col[j] >= col[k]) k = j;
        for (vis[k] = 1, j = g[k]; j; j = nxt[j])
            if (!vis[v[j]])
                if (++col[v[j]] > ans) ans = col[v[j]];
    }
    printf("%d", ans + 1);
}
// 优先队列优化MCS
const int N = 1010;
int n, m;
vector<int> g[N];
bool mp[N][N];
int label[N], seq[N], id[N];
struct Stat {
    int lab, u;
    Stat(int lab, int u) : lab(lab), u(u) {}
    bool operator<(const Stat &b) const { return lab < b.lab; }
};
void MCS() {
    priority_queue<Stat> q;
    memset(label, 0, sizeof(label));
    memset(id, 0, sizeof(id));
    for (int u = 1; u <= n; u++) q.push(Stat(0, u));
    for (int i = n; i; i--) {
        while (id[q.top().u]) q.pop();
        int u = q.top().u;
        q.pop();
        id[u] = i;
        for (int t = 0; t < g[u].size(); t++) {
            int v = g[u][t];
            if (!id[v]) {
                label[v]++;
```

```
                q.push(Stat(label[v], v));
            }
        }
    }
    for (int u = 1; u <= n; u++) seq[id[u]] = u;
}
// 弦图判定
bool check() {
    vector<int> c;
    for (int i = 1; i <= n; i++) {
        int u = seq[i];
        c.clear();
        for (int t = 0; t < g[u].size(); t++) {
            int v = g[u][t];
            if (id[v] > id[u]) c.push_back(v);
        }
        if (c.empty()) continue;
        int sc = c[0];
        for (int t = 1; t < c.size(); t++)
            if (id[c[t]] < id[sc]) sc = c[t];
        for (int t = 0; t < c.size(); t++) {
            int v = c[t];
            if (v == sc) continue;
            if (!mp[sc][v]) return false;
        }
    }
    return true;
}
void init(int n) {
    memset(mp, false, sizeof(mp));
    for (int i = 1; i <= n; i++) g[i].clear();
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        init(n);
        for (int i = 1, u, v; i <= m; i++) {
            scanf("%d%d", &u, &v);
            mp[u][v] = mp[v][u] = true;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        MCS();
        printf("%s\n", check() ? "Perfect" : "Imperfect");
    }
    return 0;
}
```

3.34 优先队列优化 MCS 算法

```
/*
    MCS+优先队列
    一个无向图称为弦图当图中任意长度大于3的环都至少有一个弦。
    Example: 用最少的颜色给每个点染色使得相邻的点染的颜色不同:
    Solution: 通过完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色
    Example: 最大独立集问题：选择最多的点使得任意两个点不相邻
    solution: 通过完美消除序列从前往后能选就选。
*/
const int N = 1010;
int n, m;
vector<int> g[N];
bool mp[N][N];
int label[N], seq[N], id[N];
struct Stat {
    int lab, u;
    Stat(int lab, int u) : lab(lab), u(u) {}
    bool operator<(const Stat &b) const { return lab < b.lab; }
};
void MCS() {
    priority_queue<Stat> q;
    memset(label, 0, sizeof(label));
    memset(id, 0, sizeof(id));
    for (int u = 1; u <= n; u++) q.push(Stat(0, u));
    for (int i = n; i; i--) {
        while (id[q.top().u]) q.pop();
        int u = q.top().u;
        q.pop();
        id[u] = i;
        for (int t = 0; t < g[u].size(); t++) {
            int v = g[u][t];
            if (!id[v]) {
                label[v]++;
                q.push(Stat(label[v], v));
            }
        }
    }
    for (int u = 1; u <= n; u++) seq[id[u]] = u;
}
/*
    Example
    弦图判定
*/
bool check() {
    vector<int> c;
```

```
    for (int i = 1; i <= n; i++) {
        int u = seq[i];
        c.clear();
        for (int t = 0; t < g[u].size(); t++) {
            int v = g[u][t];
            if (id[v] > id[u]) c.push_back(v);
        }
        if (c.empty()) continue;
        int sc = c[0];
        for (int t = 1; t < c.size(); t++)
            if (id[c[t]] < id[sc]) sc = c[t];
        for (int t = 0; t < c.size(); t++) {
            int v = c[t];
            if (v == sc) continue;
            if (!mp[sc][v]) return false;
        }
    }
    return true;
}
void init(int n) {
    memset(mp, false, sizeof(mp));
    for (int i = 1; i <= n; i++) g[i].clear();
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        init(n);
        for (int i = 1, u, v; i <= m; i++) {
            scanf("%d%d", &u, &v);
            mp[u][v] = mp[v][u] = true;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        MCS();
        printf("%s\n", check() ? "Perfect" : "Imperfect");
    }
    return 0;
}
```

3.35 Prufer 数列

```
/*
    任意一棵n节点的树都可唯一的用长度为n-2的prufer编码表示,
    度数为m的节点的序号在prufer编码中出现的次数为m-1
    因此无限制生成树的种类有n^(n-2)种,
    Prufer数列构造方法:
```

```
        选取编号最小的叶子节点删掉
        并将它的父亲加入到prufer数列中
        直到树上还有两个节点。
    Prufer数列转化成树方法:
        设{a1,a2,..an-2}为一棵有n个节点的树的Prufer序列
        另建一个集合G含有元素{1..n},
        找出集合中最小的未在Prufer序列中出现过的数,
        将该点与Prufer序列中首项连一条边,
        并将该点和Prufer序列首项删除, 重复操作n-2次,
        将集合中剩余的两个点之间连边即可。
    示例: 给出树上n个点的度数, 求满足条件的树的数目
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int M = 200;
int p[200], pNum;
bool f[M];
void Prime() {
    int i, j;
    for (i = 2; i < M; i++) {
        if (!f[i]) {
            p[pNum++] = i;
        }
        for (j = 0; j < pNum && p[j] * i < M; j++) {
            f[p[j] * i] = 1;
            if (!(i % p[j])) break;
        }
    }
}
int n, d[200], u[200];
void Calc(int x, int r) {
    if (!x) return;
    for (int i = 0; i < pNum; i++)
        for (int j = p[i]; x / j != 0; j *= p[i]) u[i] += x / j * r;
}
void Solve() {
    scanf("%d", &n);
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        scanf("%d", &d[i]), sum += d[i];
        if (d[i] <= 0 && n > 1) {
            puts("0");
            return;
        }
    }
```

```
    if (sum != (n - 1) * 2) {
        puts("0");
        return;
    }
    if (n == 1) {
        puts("1");
        return;
    }
    Calc(n - 2, 1);
    for (int i = 1; i <= n; i++) Calc(d[i] - 1, -1);
    long long ans = 1;
    for (int i = 0; i < pNum; i++)
        for (int j = 1; j <= u[i]; j++) ans *= p[i];
    printf("%lld\n", ans);
}
int main() {
    Prime();
    Solve();
    return 0;
}
```

3.36 带花树

```
/* 一般图最大匹配
 * 用带花树求编号从0开始的n个点的图的最大匹配，时间复杂度 O(n^3)
 * mate[]为配偶结点的编号，没有匹配上的点为-1。
 * 传入结点个数n及各结点的出边表G[]，返回匹配点对的数量ret。
 */
using namespace std;
const int N = 510;
int n, m, x, y;
vector<int> g[N];
namespace Blossom {
int mate[N], n, ret, nxt[N], f[N], mark[N], vis[N];
queue<int> Q;
int F(int x) { return x == f[x] ? x : f[x] = F(f[x]); }
void merge(int a, int b) { f[F(a)] = F(b); }
int lca(int x, int y) {
    static int t = 0;
    t++;
    for (;; swap(x, y))
        if (~x) {
            if (vis[x = F(x)] == t) return x;
            vis[x] = t;
            x = mate[x] != -1 ? nxt[mate[x]] : -1;
```

```
        }
}
void group(int a, int p) {
    for (int b, c; a != p; merge(a, b), merge(b, c), a = c) {
        b = mate[a], c = nxt[b];
        if (F(c) != p) nxt[c] = b;
        if (F(c) != p) nxt[c] = b;
        if (mark[b] == 2) mark[b] = 1, Q.push(b);
        if (mark[c] == 2) mark[c] = 1, Q.push(c);
    }
}
void aug(int s, const vector<int> G[]) {
    for (int i = 0; i < n; ++i) nxt[i] = vis[i] = -1, f[i] = i, mark[i] = 0;
    while (!Q.empty()) Q.pop();
    Q.push(s);
    mark[s] = 1;
    while (mate[s] == -1 && !Q.empty()) {
        int x = Q.front();
        Q.pop();
        for (int i = 0, y; i < (int)G[x].size(); ++i) {
            if ((y = G[x][i]) != mate[x] && F(x) != F(y) && mark[y] != 2) {
                if (mark[y] == 1) {
                    int p = lca(x, y);
                    if (F(x) != p) nxt[x] = y;
                    if (F(y) != p) nxt[y] = x;
                    group(x, p), group(y, p);
                } else if (mate[y] == -1) {
                    nxt[y] = x;
                    for (int j = y, k, l; ~j; j = l)
                        k = nxt[j], l = mate[k], mate[j] = k, mate[k] = j;
                    break;
                } else
                    nxt[y] = x, Q.push(mate[y]), mark[mate[y]] = 1, mark[y] = 2;
            }
        }
    }
}
int solve(int _n, const vector<int> G[]) {
    n = _n;
    memset(mate, -1, sizeof(mate));
    for (int i = 0; i < n; ++i)
        if (mate[i] == -1) aug(i, G);
    for (int i = ret = 0; i < n; ++i) ret += mate[i] > i;
    printf("%d\n", ret);
    for (int i = 0; i < n; i++) printf("%d ", mate[i] + 1);
    return ret;
```

```
}
} // namespace Blossom
int main() {
    scanf("%d%d", &n, &m);
    while (m--)
        scanf("%d%d", &x, &y), x--, y--, g[x].push_back(y), g[y].push_back(x);
    Blossom::solve(n, g);
    return 0;
}
```

3.37 权值带花树

```
/* 一般图最大权匹配
 * 输出匹配方案
 */
#include <bits/stdc++.h>
using namespace std;
#define INF INT_MAX
#define MAXN 400
struct edge {
    int u, v, w;
    edge() {}
    edge(int u, int v, int w) : u(u), v(v), w(w) {}
};
int n, n_x;
edge g[MAXN * 2 + 1][MAXN * 2 + 1];
int lab[MAXN * 2 + 1];
int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1],
    pa[MAXN * 2 + 1];
int flower_from[MAXN * 2 + 1][MAXN + 1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
vector<int> flower[MAXN * 2 + 1];
queue<int> q;
inline int e_delta(const edge &e) { // does not work inside blossoms
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
}
inline void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])) slack[x] = u;
}
inline void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
        if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0) update_slack(u, x);
}
void q_push(int x) {
    if (x <= n)
```

```
        q.push(x);
    else
        for (size_t i = 0; i < flower[x].size(); i++) q_push(flower[x][i]);
}
inline void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
        for (size_t i = 0; i < flower[x].size(); ++i) set_st(flower[x][i], b);
}
inline int get_pr(int b, int xr) {
    int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
    if (pr % 2 == 1) { //檢查他在前一層圖是奇點還是偶點
        reverse(flower[b].begin() + 1, flower[b].end());
        return (int)flower[b].size() - pr;
    } else
        return pr;
}
inline void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u > n) {
        edge e = g[u][v];
        int xr = flower_from[u][e.u], pr = get_pr(u, xr);
        for (int i = 0; i < pr; ++i) set_match(flower[u][i], flower[u][i ^ 1]);
        set_match(xr, v);
        rotate(flower[u].begin(), flower[u].begin() + pr, flower[u].end());
    }
}
inline void augment(int u, int v) {
    for (;;) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        u = st[pa[xnv]], v = xnv;
    }
}
inline int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
        if (u == 0) continue;
        if (vis[u] == t) return u;
        vis[u] = t; //這種方法可以不用清空v陣列
        u = st[match[u]];
        if (u) u = st[pa[u]];
    }
    return 0;
```

```
}
inline void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
        flower[b].push_back(x), flower[b].push_back(y = st[match[x]]),
            q_push(y);
    reverse(flower[b].begin() + 1, flower[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
        flower[b].push_back(x), flower[b].push_back(y = st[match[x]]),
            q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flower_from[b][x] = 0;
    for (size_t i = 0; i < flower[b].size(); ++i) {
        int xs = flower[b][i];
        for (int x = 1; x <= n_x; ++x)
            if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                g[b][x] = g[xs][x], g[x][b] = g[x][xs];
        for (int x = 1; x <= n; ++x)
            if (flower_from[xs][x]) flower_from[b][x] = xs;
    }
    set_slack(b);
}
inline void expand_blossom(int b) { // S[b] == 1
    for (size_t i = 0; i < flower[b].size(); ++i)
        set_st(flower[b][i], flower[b][i]);
    int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flower[b][i], xns = flower[b][i + 1];
        pa[xs] = g[xns][xs].u;
        S[xs] = 1, S[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flower[b].size(); ++i) {
        int xs = flower[b][i];
        S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
```

```
}
inline bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
        pa[v] = e.u, S[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
        int lca = get_lca(u, v);
        if (!lca)
            return augment(u, v), augment(v, u), true;
        else
            add_blossom(u, lca, v);
    }
    return false;
}
inline bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(x);
    if (q.empty()) return false;
    for (;;) {
        while (q.size()) {
            int u = q.front();
            q.pop();
            if (S[st[u]] == 1) continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v])) return true;
                    } else
                        update_slack(u, st[v]);
                }
        }
        int d = INF;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x]) {
                if (S[x] == -1)
                    d = min(d, e_delta(g[slack[x]][x]));
                else if (S[x] == 0)
                    d = min(d, e_delta(g[slack[x]][x]) / 2);
```

```
            }
        for (int u = 1; u <= n; ++u) {
            if (S[st[u]] == 0) {
                if (lab[u] <= d) return 0;
                lab[u] -= d;
            } else if (S[st[u]] == 1)
                lab[u] += d;
        }
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b) {
                if (S[st[b]] == 0)
                    lab[b] += d * 2;
                else if (S[st[b]] == 1)
                    lab[b] -= d * 2;
            }
        q = queue<int>();
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x] && st[slack[x]] != x &&
                e_delta(g[slack[x]][x]) == 0)
                if (on_found_edge(g[slack[x]][x])) return true;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1 && lab[b] == 0) expand_blossom(b);
    }
    return false;
}
inline pair<long long, int> weight_blossom() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) {
            flower_from[u][v] = (u == v ? u : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
        if (match[u] && match[u] < u) tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
}
inline void init_weight_graph() {
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) g[u][v] = edge(u, v, 0);
```

```
}
int main() {
    int m;
    scanf("%d%d", &n, &m);
    init_weight_graph();
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        g[u][v].w = g[v][u].w = w;
    }
    printf("%lld\n", weight_blossom().first);
    for (int u = 1; u <= n; ++u) printf("%d ", match[u]);
    puts("");
    return 0;
}
```

3.38 虚树

```
/* 虚树
 * 每次询问只与树上关键点有关
 * dp部分计算出最小边割集使得1和所有关键点不连通
 */
namespace Virtual_Tree {
const LL INF = 1LL << 60;
LL mn[N];
int ed, d[N], vis[N], f[N], size[N], son[N], w[N << 1];
int nxt[N << 1], g[N], v[N << 1], st[N], en[N], dfn, top[N];
void add_edge(int x, int y, int z) {
    v[++ed] = y;
    w[ed] = z;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    size[x] = 1;
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != f[x]) {
            mn[v[i]] = min(mn[x], (LL)w[i]);
            f[v[i]] = x, d[v[i]] = d[x] + 1;
            dfs(v[i]), size[x] += size[v[i]];
            if (size[v[i]] > size[son[x]]) son[x] = v[i];
        }
}
void dfs2(int x, int y) {
    if (x == -1) return;
```

```
    st[x] = ++dfn;
    top[x] = y;
    if (son[x]) dfs2(son[x], y);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != son[x] && v[i] != f[x]) dfs2(v[i], v[i]);
    en[x] = dfn;
}
int lca(int x, int y) {
    for (; top[x] != top[y]; x = f[top[x]])
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
    return d[x] < d[y] ? x : y;
}
void Initialize() {
    mn[1] = INF;
    memset(g, dfn = ed = 0, sizeof(g));
    memset(v, 0, sizeof(v));
    memset(nxt, 0, sizeof(nxt));
    memset(son, -1, sizeof(son));
}
int V[N << 1], NXT[N << 1], G[N], ED, a[N], q[N], mark[N];
void ADD_edge(int x, int y) {
    V[++ED] = y;
    NXT[ED] = G[x];
    G[x] = ED;
}
bool cmp(int a, int b) { return st[a] < st[b]; }
LL dp[N];
void DP(int x) {
    dp[x] = mn[x];
    LL tmp = 0;
    for (int i = G[x]; i; i = NXT[i]) {
        DP(V[i]);
        if (mark[V[i]])
            tmp += mn[V[i]];
        else
            tmp += dp[V[i]];
    }
    G[x] = 0;
    if (tmp == 0)
        dp[x] = mn[x];
    else if (tmp <= dp[x])
        dp[x] = tmp;
```

```
}
void build(int m) {
    int tot, t, i, x;
    ED = 0;
    vis[1] = a[1] = 1;
    for (tot = ++m, i = 2; i <= m; i++)
        a[i] = read(), mark[a[i]] = vis[a[i]] = 1;
    // 读入一组关键点a, mark用于标记关键点
    sort(a + 1, a + m + 1, cmp);
    for (i = 1; i < m; i++)
        if (!vis[x = lca(a[i], a[i + 1])]) vis[a[++tot] = x] = 1;
    m = tot, sort(a + 1, a + m + 1, cmp);
    for (q[t = 1] = 1, i = 2; i <= m; q[++t] = a[i++]) {
        while (st[a[i]] < st[q[t]] || en[a[i]] > en[q[t]]) t--;
        ADD_edge(q[t], a[i]);
    }
    DP(1);
    for (i = 1; i <= m; i++) mark[a[i]] = vis[a[i]] = 0;
}
} // namespace Virtual_Tree
```

4 网络流

4.1 二分图匹配

```
/*
    二分图匹配
*/
namespace Hungary {
const int MAX_V = 2000;
const int INF = 0x3f3f3f3f;
int V, match[MAX_V];
vector<int> G[MAX_V];
bool used[MAX_V];
void add_edge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
}
bool dfs(int v) {
    used[v] = 1;
    for (int i = 0; i < G[v].size(); i++) {
        int u = G[v][i], w = match[u];
        if (w < 0 || !used[w] && dfs(w)) {
            match[v] = u;
            match[u] = v;
            return 1;
        }
    }
    return 0;
}
int bipartite_matching() {
    int res = 0;
    memset(match, -1, sizeof(match));
    for (int v = 0; v < V; v++) {
        if (match[v] < 0) {
            memset(used, 0, sizeof(used));
            if (dfs(v)) res++;
        }
    }
    return res;
}
void Initialize(int n) {
    V = n;
    for (int i = 0; i < V; i++) G[i].clear();
}
} // namespace Hungary
```

4.2 KM 算法

```
/*
    KM算法
    复杂度O(nx*nx*ny)
    求最大权匹配
    若求最小权匹配，可将权值取相反数，结果取相反数
    点的编号从0开始
*/
const int N = 310, INF = 0x3f3f3f3f;
namespace KM {
int nx, ny;                 //两边的点数
int g[N][N];                //二分图描述
int linker[N], lx[N], ly[N]; // y中各点匹配状态，x,y中的点标号
int n, slack[N];
bool visx[N], visy[N];
bool DFS(int x) {
    visx[x] = 1;
    for (int y = 0; y < ny; y++) {
        if (visy[y]) continue;
        int tmp = lx[x] + ly[y] - g[x][y];
        if (tmp == 0) {
            visy[y] = true;
            if (linker[y] == -1 || DFS(linker[y])) {
                linker[y] = x;
                return 1;
            }
        } else if (slack[y] > tmp)
            slack[y] = tmp;
    }
    return 0;
}
int KM() {
    memset(linker, -1, sizeof(linker));
    memset(ly, 0, sizeof(ly));
    for (int i = 0; i < nx; i++) {
        lx[i] = -INF;
        for (int j = 0; j < ny; j++)
            if (g[i][j] > lx[i]) lx[i] = g[i][j];
    }
    for (int x = 0; x < nx; x++) {
        for (int i = 0; i < ny; i++) slack[i] = INF;
        for (;;) {
            memset(visx, false, sizeof(visx));
            memset(visy, false, sizeof(visy));
            if (DFS(x)) break;
```

```
            int d = INF;
            for (int i = 0; i < ny; i++)
                if (!visy[i] && d > slack[i]) d = slack[i];
            for (int i = 0; i < nx; i++)
                if (visx[i]) lx[i] -= d;
            for (int i = 0; i < ny; i++) {
                if (visy[i])
                    ly[i] += d;
                else
                    slack[i] -= d;
            }
        }
    }
    int res = 0;
    for (int i = 0; i < ny; i++)
        if (linker[i] != -1) res += g[linker[i]][i];
    return res;
}
} // namespace KM
// Test
int main() {
    using namespace KM;
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) scanf("%d", &g[i][j]);
        nx = ny = n;
        printf("%d\n", KM());
    }
    return 0;
}
```

4.3 最小乘积完美匹配

```
/*
    最小乘积匹配
    给出一张二分图，每条边上有a,b两个值，
    求完美匹配，使得suma*sumb最小
*/
typedef long long LL;
const int N = 310;
const LL INF = 0x3f3f3f3f3f3f3f3f;
int nx, ny;     //两边的点数
LL g[N][N];     //二分图描述
int linker[N]; // y中各点匹配状态
LL lx[N], ly[N]; // x,y中的点标号
```

```
int n;
LL slack[N];
bool visx[N], visy[N];
LL ans = INF;
int T, a[N][N], b[N][N];
struct P {
    int x, y;
    P() { x = y = 0; }
    P(int _x, int _y) {
        x = _x;
        y = _y;
    }
    P operator-(const P& rhs) { return P(x - rhs.x, y - rhs.y); }
} l, r;
LL cross(P a, P b) { return (LL)a.x * b.y - (LL)a.y * b.x; }
bool DFS(int x) {
    visx[x] = 1;
    for (int y = 0; y < ny; y++) {
        if (visy[y]) continue;
        int tmp = lx[x] + ly[y] - g[x][y];
        if (tmp == 0) {
            visy[y] = true;
            if (linker[y] == -1 || DFS(linker[y])) {
                linker[y] = x;
                return 1;
            }
        } else if (slack[y] > tmp)
            slack[y] = tmp;
    }
    return 0;
}
P KM() {
    P p;
    memset(linker, -1, sizeof(linker));
    memset(ly, 0, sizeof(ly));
    for (int i = 0; i < nx; i++) {
        lx[i] = -INF;
        for (int j = 0; j < ny; j++)
            if (g[i][j] > lx[i]) lx[i] = g[i][j];
    }
    for (int x = 0; x < nx; x++) {
        for (int i = 0; i < ny; i++) slack[i] = INF;
        for (;;) {
            memset(visx, false, sizeof(visx));
            memset(visy, false, sizeof(visy));
            if (DFS(x)) break;
```

```
            LL d = INF;
            for (int i = 0; i < ny; i++)
                if (!visy[i] && d > slack[i]) d = slack[i];
            for (int i = 0; i < nx; i++)
                if (visx[i]) lx[i] -= d;
            for (int i = 0; i < ny; i++) {
                if (visy[i])
                    ly[i] += d;
                else
                    slack[i] -= d;
            }
        }
    }
    LL res = 0;
    for (int i = 0; i < ny; i++)
        if (linker[i] != -1) {
            p.x += a[linker[i]][i];
            p.y += b[linker[i]][i];
        }
    res = (LL)p.x * p.y;
    if (res < ans) ans = res;
    return p;
}
void work(P l, P r) {
    P t = l - r;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) g[i][j] = -cross(P(a[i][j], b[i][j]), t);
    P mid = KM();
    if (cross(mid - l, r - mid) > 0) work(l, mid), work(mid, r);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        ans = INF;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) scanf("%d", &a[i][j]);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) scanf("%d", &b[i][j]);
        nx = ny = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) g[i][j] = -a[i][j];
        l = KM();
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) g[i][j] = -b[i][j];
        r = KM();
```

```
        work(l, r);
        printf("%lld\n", ans);
    }
    return 0;
}
```

4.4 Dinic 算法

```
/*
    Dinic算法
*/
namespace Dinic {
const int INF = 0x3f3f3f3f;
const int MAX_V = 500;
struct edge {
    int to, cap, rev;
};
vector<edge> G[MAX_V];
int V, level[MAX_V], iter[MAX_V];
void add_edge(int from, int to, int cap) {
    G[from].push_back((edge){to, cap, G[to].size()});
    G[to].push_back((edge){from, 0, G[from].size() - 1});
}
void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while (!que.empty()) {
        int v = que.front();
        que.pop();
        for (int i = 0; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}
int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int &i = iter[v]; i < G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
```

```
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}
int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        int f;
        while ((f = dfs(s, t, INF)) > 0) {
            flow += f;
        }
    }
}
void Initialize(int n) {
    V = n;
    for (int i = 0; i <= V; i++) G[i].clear();
}
} // namespace Dinic
/*
    Example
    求最小割的最小边数
*/
int T, n, m, a, b, c, s, t;
int main() {
    using namespace Dinic;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        scanf("%d%d", &s, &t);
        Initialize(n);
        for (int i = 1; i <= m; i++) {
            scanf("%d%d%d", &a, &b, &c);
            add_edge(a, b, c * (m + 1) + 1);
        }
        printf("%d\n", max_flow(s, t) % (m + 1));
    }
    return 0;
```

```
}
/*
    Example
    方格取数
    相邻的数字不能选，要求点权和最大
    二分图最大独立集=总和-最小割集
*/
int n, a[30][30];
void solve() {
    using namespace Dinic;
    int sum = 0, s = n * n, t = n * n + 1;
    for (int i = 0; i <= t; i++) G[i].clear();
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            if ((i + j) % 2 == 0) {
                if (i + 1 < n) add_edge(i * n + j, (i + 1) * n + j, INF);
                if (j + 1 < n) add_edge(i * n + j, i * n + j + 1, INF);
                if (i > 0) add_edge(i * n + j, (i - 1) * n + j, INF);
                if (j > 0) add_edge(i * n + j, i * n + j - 1, INF);
                add_edge(s, i * n + j, a[i][j]);
            } else
                add_edge(i * n + j, t, a[i][j]);
            sum += a[i][j];
        }
    printf("%d\n", sum - max_flow(s, t));
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) scanf("%d", &a[i][j]);
        }
        solve();
    }
    return 0;
}
```

4.5 ISAP 算法

```
/*
    ISAP
*/
namespace ISAP {
const int N = 30010, inf = ~0U >> 2;
struct edge {
    int t, f;
```

```
    edge *nxt, *pair;
} * g[N], *d[N], pool[4 * N], *cur = pool;
int v[N], x, cnt, i, S, T, h[N], gap[N], maxflow;
void Initialize() {
    for (cur = pool, i = 0; i <= sum; i++)
        g[i] = d[i] = NULL, h[i] = gap[i] = 0;
    maxflow = 0;
}
void add(int s, int t, int w) {
    edge* p = cur++;
    p->t = t;
    p->f = w;
    p->nxt = g[s];
    g[s] = p;
    p = cur++;
    p->t = s;
    p->f = 0;
    p->nxt = g[t];
    g[t] = p;
    g[s]->pair = g[t];
    g[t]->pair = g[s];
}
int sap(int v, int flow) {
    if (v == T) return flow;
    int rec = 0;
    for (edge* p = d[v]; p; p = p->nxt)
        if (h[v] == h[p->t] + 1 && p->f) {
            int ret = sap(p->t, min(flow - rec, p->f));
            p->f -= ret;
            p->pair->f += ret;
            d[v] = p;
            if ((rec += ret) == flow) return flow;
        }
    if (!(--gap[h[v]])) h[S] = T;
    gap[++h[v]]++;
    d[v] = g[v];
    return rec;
}
/*
    残量网络
*/
void dfs(int x) {
    v[x] = 1;
    for (edge* p = g[x]; p; p = p->nxt)
        if (p->f && !v[p->t]) dfs(p->t);
}
```

```
void doit() {
    for (gap[0] = T, i = 1; i <= T; i++) d[i] = g[i];
    while (h[S] < T) maxflow += sap(S, inf);
}
} // namespace ISAP
/*
    给出一个二分图
    左边点和右边点存在矛盾关系的不可共存
    选择一些点，使得其点权和最大
    不能共存的连INF，其余点权分别连源汇
    选择点权和最大=总权值-建图后边权最小割=总权值-最大流
    残量网络=最小割后的点=选择的点
*/
using namespace ISAP;
int sum, ss, t1, t2, n, m, s, tmp1[N], tmp2[N], u, e, cost;
int main() {
    while (~scanf("%d%d%d", &n, &m, &s)) {
        memset(v, 0, sizeof(v));
        ss = 0;
        sum = n + m + 2;
        S = sum - 1;
        T = sum;
        for (i = 1; i <= n; i++) {
            scanf("%d", &x);
            add(S, i, x);
            ss += x;
        }
        for (i = 1; i <= m; i++) {
            scanf("%d", &x);
            add(i + n, T, x);
            ss += x;
        }
        for (i = 1; i <= s; i++) {
            scanf("%d%d", &u, &e);
            add(u, e + n, inf);
        }
        doit();
        printf("%d ", ss - maxflow);
        dfs(S);
        for (t1 = 0, i = 1; i <= n; i++)
            if (v[i]) tmp1[t1++] = i;
        for (t2 = 0, i = n + 1; i <= n + m; i++)
            if (!v[i]) tmp2[t2++] = i - n;
        printf("%d %d\n", t1, t2);
        for (i = 0; i < t1 - 1; i++) printf("%d ", tmp1[i]);
        if (t1) printf("%d\n", tmp1[t1 - 1]);
```

```
        for (i = 0; i < t2 - 1; i++) printf("%d ", tmp2[i]);
        if (t2) printf("%d\n", tmp2[t2 - 1]);
    }
    return 0;
}
```

4.6 最大密度子图

```
/*
    最大密度子图
    要求[边权和/点数]最大
    输出选定点集
*/
using namespace std;
const double INF = 0x3fffffff;
const double eps = 1e-8;
const int MAX_V = 110;
typedef double cap_type;
struct edge {
    int to, rev;
    cap_type cap;
    edge(int to, cap_type cap, int rev) : to(to), cap(cap), rev(rev) {}
};
vector<edge> G[MAX_V];
int V, level[MAX_V], iter[MAX_V];
void add_edge(int from, int to, cap_type cap) {
    G[from].push_back((edge){to, cap, G[to].size()});
    G[to].push_back((edge){from, 0, G[from].size() - 1});
}
void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while (!que.empty()) {
        int v = que.front();
        que.pop();
        for (int i = 0; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
```

```
}
cap_type dfs(int v, int t, cap_type f) {
    if (v == t) return f;
    for (int &i = iter[v]; i < G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
            cap_type d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}
cap_type max_flow(int s, int t) {
    cap_type flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        cap_type f;
        while ((f = dfs(s, t, INF)) > 0) {
            flow += f;
        }
    }
}
const int MAX_M = 1010;
const int MAX_N = 100;
int N, M, x[MAX_M], y[MAX_M], D[MAX_N];
void construct_graph(int s, int t, cap_type g) {
    for (int i = 0; i < MAX_V; i++) G[i].clear();
    for (int i = 0; i < N; i++) {
        add_edge(s, i, M);
        add_edge(i, t, M + 2 * g - D[i]);
    }
    for (int i = 0; i < M; i++) {
        add_edge(x[i] - 1, y[i] - 1, 1.0);
        add_edge(y[i] - 1, x[i] - 1, 1.0);
    }
}
int leftv, vis[MAX_V];
void cal_res_net(int v) {
    ++leftv;
    vis[v] = 1;
```

```
    for (int i = 0; i < G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > eps && !vis[e.to]) cal_res_net(e.to);
    }
}
void init() {
    memset(D, 0, sizeof(D));
    for (int i = 0; i < M; i++) {
        scanf("%d%d", &x[i], &y[i]);
        D[x[i] - 1]++;
        D[y[i] - 1]++;
    }
}
void solve() {
    if (M == 0) {
        printf("%d\n%d\n", 1, 1);
        return;
    }
    int s = N, t = N + 1;
    double l = 0, r = M, mid, tmp;
    const double Limit = 1.0 / N / N;
    while (r - l >= Limit) {
        mid = (l + r) / 2;
        construct_graph(s, t, mid);
        tmp = (N * M - max_flow(s, t)) / 2;
        (tmp > eps ? l : r) = mid;
    }
    construct_graph(s, t, l);
    max_flow(s, t);
    leftv = 0;
    cal_res_net(s);
    printf("%d\n", leftv - 1);
    for (int i = 0; i < N; i++)
        if (vis[i]) printf("%d\n", i + 1);
}
int main() {
    while (~scanf("%d%d", &N, &M)) {
        init();
        solve();
    }
    return 0;
}
```

4.7 最大权闭合图

```
/*
    最大权闭合图
    有向图点权可正可负。
    对于任意一条有向边i和j，选择了点i就必须选择点j，
    选择一些点使得得到权值最大，同时删去的点最少
*/
using namespace Dinic;
const int MAX_N = 5000;
const int MAX_M = 60000;
int N, M, w[MAX_N], a[MAX_M], b[MAX_M];
LL max_weight_closure(int s, int t) {
    LL W = 0;
    V = t;
    for (int i = 0; i <= V; i++) G[i].clear();
    for (int i = 0; i < N; i++) {
        if (w[i] > 0) W += w[i], add_edge(s, i, w[i]);
        if (w[i] < 0) add_edge(i, t, -w[i]);
    }
    for (int i = 0; i < M; i++) {
        add_edge(a[i] - 1, b[i] - 1, INF);
    }
    return W - max_flow(s, t);
}
int leftv, vis[MAX_V];
// 遍历残余网络
void cal_res_net(int v) {
    ++leftv;
    vis[v] = 1;
    for (int i = 0; i < G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > 0 && !vis[e.to]) cal_res_net(e.to);
    }
}
void init() {
    for (int i = 0; i < N; i++) scanf("%d", &w[i]);
    for (int i = 0; i < M; i++) scanf("%d%d", &a[i], &b[i]);
}
void solve() {
    int s = N, t = N + 1;
    LL max_profit = max_weight_closure(s, t);
    memset(vis, 0, sizeof(vis));
    leftv = 0;
    cal_res_net(s);
    printf("%d %lld\n", --leftv, max_profit);
}
```

```
int main() {
    while (~scanf("%d%d", &N, &M)) {
        init();
        solve();
    }
    return 0;
}
```

4.8 混合图欧拉回路

```
/*
    混合图欧拉回路判定
    Example:
        给定一张图，每条边的两个方向有两个不同的权值，
        现在要求从1号节点出发遍历每条边一次且仅一次，最后回到1号节点，
        求最大边权的最小值
        二分答案+混合图欧拉回路判定
*/
const int M = 2020, S = 0, T = 2019, INF = 0x3f3f3f3f;
using namespace std;
struct edge {
    int x, y, z1, z2;
} edges[M];
int n, m, max_num;
namespace Euler_Circuit {
namespace Union_Find_Set {
int fa[M], size[M], cnt;
void Initialize() {
    memset(fa, 0, sizeof fa);
    cnt = n;
}
int Find(int x) {
    if (!fa[x]) fa[x] = x, size[x] = 1;
    if (fa[x] == x) return x;
    return fa[x] = Find(fa[x]);
}
void Union(int x, int y) {
    x = Find(x);
    y = Find(y);
    if (x == y) return;
    if (size[x] > size[y]) swap(x, y);
    fa[x] = y;
    size[y] += size[x];
    cnt--;
}
```

```
} // namespace Union_Find_Set
namespace Max_Flow {
struct abcd {
    int to, f, next;
} table[1001001];
int head[M], tot = 1;
int dpt[M];
void Initialize() {
    memset(head, 0, sizeof head);
    tot = 1;
}
void Add(int x, int y, int z) {
    table[++tot].to = y;
    table[tot].f = z;
    table[tot].next = head[x];
    head[x] = tot;
}
void Link(int x, int y, int z) {
    Add(x, y, z);
    Add(y, x, 0);
}
bool BFS() {
    static int q[M];
    int i, r = 0, h = 0;
    memset(dpt, -1, sizeof dpt);
    q[++r] = S;
    dpt[S] = 1;
    while (r != h) {
        int x = q[++h];
        for (i = head[x]; i; i = table[i].next)
            if (table[i].f && !~dpt[table[i].to]) {
                dpt[table[i].to] = dpt[x] + 1;
                q[++r] = table[i].to;
                if (table[i].to == T) return true;
            }
    }
    return false;
}
int Dinic(int x, int flow) {
    int i, left = flow;
    if (x == T) return flow;
    for (i = head[x]; i && left; i = table[i].next)
        if (table[i].f && dpt[table[i].to] == dpt[x] + 1) {
            int temp = Dinic(table[i].to, min(left, table[i].f));
            left -= temp;
            table[i].f -= temp;
```

```
            table[i ^ 1].f += temp;
        }
    if (left) dpt[x] = -1;
    return flow - left;
}
} // namespace Max_Flow
bool Judge(int x) {
    using namespace Union_Find_Set;
    using namespace Max_Flow;
    Union_Find_Set::Initialize();
    Max_Flow::Initialize();
    static int degree[M];
    //入度+1 出度-1
    int i;
    memset(degree, 0, sizeof degree);
    for (i = 1; i <= m; i++) {
        if (edges[i].z2 <= x) {
            Union(edges[i].x, edges[i].y);
            degree[edges[i].x]++;
            degree[edges[i].y]--;
            Link(edges[i].x, edges[i].y, 1);
        } else if (edges[i].z1 <= x) {
            Union(edges[i].x, edges[i].y);
            degree[edges[i].x]--;
            degree[edges[i].y]++;
        }
    }
    if (cnt >= 2) return false;
    for (i = 1; i <= n; i++) {
        if (degree[i] & 1) return false;
        if (degree[i] > 0)
            Link(S, i, degree[i] >> 1);
        else
            Link(i, T, -degree[i] >> 1);
    }
    while (BFS()) Dinic(S, INF);
    for (i = head[S]; i; i = table[i].next)
        if (table[i].f) return false;
    return true;
}
} // namespace Euler_Circuit
int Bisection(int l, int r) {
    using namespace Euler_Circuit;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (Judge(mid))
```

```
            r = mid;
        else
            l = mid;
    }
    return Judge(l) ? l : r;
}
int main() {
    int i, x, y, z1, z2;
    cin >> n >> m;
    for (i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &x, &y, &z1, &z2);
        if (z1 > z2) swap(x, y), swap(z1, z2);
        edges[i].x = x;
        edges[i].y = y;
        edges[i].z1 = z1;
        edges[i].z2 = z2;
        max_num = max(max_num, z1);
        max_num = max(max_num, z2);
    }
    int ans = Bisection(1, max_num + 1);
    if (ans == max_num + 1)
        cout << "NIE" << endl;
    else
        cout << ans << endl;
    return 0;
}
```

4.9 最小费用流

```
/*
    V=点数+1
    add_edge(from,to,cap,cost)
    min_cost_flow(s,t,f): 从s到t的流量为f的流的最小费用
*/
namespace Min_Cost_flow {
const int INF = 0x3f3f3f3f;
struct edge {
    int to, cap, cost, rev;
    edge(int to, int cap, int cost, int rev)
        : to(to), cap(cap), cost(cost), rev(rev) {}
};
const int MAX_V = 10010;
int V, dist[MAX_V], prevv[MAX_V], preve[MAX_V];
vector<edge> G[MAX_V];
void add_edge(int from, int to, int cap, int cost) {
```

```
    G[from].push_back(edge(to, cap, cost, G[to].size()));
    G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
}
int min_cost_flow(int s, int t, int f) {
    int res = 0;
    while (f > 0) {
        fill(dist, dist + V, INF);
        dist[s] = 0;
        bool update = 1;
        while (update) {
            update = 0;
            for (int v = 0; v < V; v++) {
                if (dist[v] == INF) continue;
                for (int i = 0; i < G[v].size(); i++) {
                    edge &e = G[v][i];
                    if (e.cap > 0 && dist[e.to] > dist[v] + e.cost) {
                        dist[e.to] = dist[v] + e.cost;
                        prevv[e.to] = v;
                        preve[e.to] = i;
                        update = 1;
                    }
                }
            }
        }
        if (dist[t] == INF) return -1;
        int d = f;
        for (int v = t; v != s; v = prevv[v]) {
            d = min(d, G[prevv[v]][preve[v]].cap);
        }
        f -= d;
        res += d * dist[t];
        for (int v = t; v != s; v = prevv[v]) {
            edge &e = G[prevv[v]][preve[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
    }
    return res;
}
void Initialize() {
    for (int i = 0; i <= V; i++) G[i].clear();
}
} // namespace Min_Cost_flow
```

4.10 最小费用最大流

```
/*
    最小费用最大流
    限制: S<T
*/
namespace Min_Cost_Max_Flow {
const int INF = 0x3f3f3f3f;
int S, T, cnt, ans, d[N], from[N], g[N], flow;
struct edge {
    int from, to, nxt, c, v;
} e[M];
void add(int u, int v, int w, int c) {
    e[++cnt].from = u;
    e[cnt].to = v;
    e[cnt].nxt = g[u];
    g[u] = cnt;
    e[cnt].c = c;
    e[cnt].v = w;
}
void add_edge(int u, int v, int w, int c) {
    add(u, v, w, c);
    add(v, u, 0, -c);
}
bool spfa() {
    memset(d, INF, sizeof(d));
    d[S] = 0;
    memset(from, 0, sizeof(from));
    queue<int> q;
    q.push(S);
    while (!q.empty()) {
        int now = q.front();
        q.pop();
        for (int i = g[now]; i; i = e[i].nxt) {
            if (e[i].v && d[e[i].to] > d[now] + e[i].c) {
                d[e[i].to] = d[now] + e[i].c;
                from[e[i].to] = i;
                q.push(e[i].to);
            }
        }
    }
    return (d[T] != INF);
    // 改为return(d[T]<=0)即为最小费用可行流
}
void mcf() {
    int x = INF;
```

```
    for (int i = from[T]; i; i = from[e[i].from]) x = min(x, e[i].v);
    flow += x;
    for (int i = from[T]; i; i = from[e[i].from]) {
        e[i].v -= x;
        e[i ^ 1].v += x;
        ans += e[i].c * x;
    }
}
void Initialize() {
    memset(g, 0, sizeof(g));
    memset(e, 0, sizeof(e));
    ans = flow = 0;
    cnt = 1;
}
void doit() {
    while (spfa()) mcf();
}
} // namespace Min_Cost_Max_Flow
/*
    纸巾模型:
        每天对毛巾都有一定的需求ni，每天可以花f价值每条购买毛巾，
        当天用完的毛巾可以花费fA价值每条通过快消毒在A天之后得到一条可用的，
        也可以通过花费fB价值每条，通过慢消毒在B天之后获得可用的
        问满足每天需求所用的最小花费。
    Solution:
        首先，我们建立X集合，表示每天用完之后需要消毒的毛巾，显然第i个点值为ni，
        建立Y集合表示每天需要的毛巾数量，第i个点为ni，
        对于X中每个点，向A天后对应Y中的每个点连流量为INF，费用为fA的边
        同时向B题后对应Y中的每个点连流量为INF，费用为fB的边
        源点向X中第i个点连ni流量0费用的边，Y中第i个点向汇点连ni流量0费用的边，
        对于购买新毛巾的操作，我们从源点向Y中每个点连INF流量f费用的边
        那么求该图的最小费用最大流就是答案。
        但是我们发现按照以上方式建图边数量非常的庞大。
        考虑建图优化，我们对于X中每个点i向i+1连边，
        而对于X向Y的连边，我们只要连到有效区间的起点即可，
        这样就等价于X中每个点，向A(B)天后对应Y中的每个点连流量为INF，费用为fA(fB)的边
*/
void solve() {
    namespace Min_Cost_Max_Flow;
    scanf("%d%d%d%d%d%d", &n, &A, &B, &f, &fA, &fB) Initialize();
    S = 0;
    T = 2 * n + 1;
    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        add_edge(S, i, x, 0);
```

```
        add_edge(i + n, T, x, 0);
    }
    for (int i = 1; i < n; i++) add_edge(i, i + 1, INF, 0);
    for (int i = 1; i <= n; i++) add_edge(S, i + n, INF, f);
    for (int i = 1; i + A + 1 <= n; i++) add_edge(i, i + A + 1 + n, INF, fA);
    for (int i = 1; i + B + 1 <= n; i++) add_edge(i, i + B + 1 + n, INF, fB);
    doit();
    printf("%d\n", ans);
}
```

4.11 动态费用流

```
/*
    动态加边网络流
    Example
        修车问题
        n种型号的车由m个工人修理需要不同的时间t[i][j],
        第i种型号的车有c[i]辆。
        问最少的总等待时间
    Solution
        对工人进行拆点,
        点[(i-1)*tot+j]表示第i个工人修理自己负责的倒数第j辆车
        型号为k的车代价为t[k][i]*(j-1)
        考虑数据规模过大, 进行动态加边
        某位工人在修理倒数第j辆车时
        再把他修理倒数第j+1辆车的边加上
*/
const int INF = 0x7fffffff, N = 100010, M = 3000010;
namespace Dynamic_Min_Cost_Max_Flow {
int n, m, S = 0, T = 100001, flow, tot, cnt, ans, t[45][105];
int c[45], d[N], q[N], from[N], g[N];
bool inq[N];
struct edge {
    int from, to, nxt, c, v;
} e[M];
void Initialize() {
    memset(g, 0, sizeof(g));
    memset(e, 0, sizeof(e));
    ans = flow = 0;
    cnt = 1;
}
void add(int u, int v, int w, int c) {
    cnt++;
    e[cnt].from = u;
    e[cnt].to = v;
```

```
    e[cnt].nxt = g[u];
    g[u] = cnt;
    e[cnt].c = c;
    e[cnt].v = w;
}
void add_edge(int u, int v, int w, int c) {
    add(u, v, w, c);
    add(v, u, 0, -c);
}
bool spfa() {
    for (int i = 0; i <= T; i++) d[i] = INF;
    int t = 0, w = 1;
    d[S] = 0;
    inq[0] = 1;
    q[S] = 0;
    while (t != w) {
        int now = q[t];
        t++;
        if (t == T) t = 0;
        for (int i = g[now]; i; i = e[i].nxt)
            if (e[i].v && d[e[i].to] > d[now] + e[i].c) {
                d[e[i].to] = d[now] + e[i].c;
                from[e[i].to] = i;
                if (!inq[e[i].to]) {
                    inq[e[i].to] = 1;
                    q[w++] = e[i].to;
                    if (w == T) w = 0;
                }
            }
        inq[now] = 0;
    }
    if (d[T] == INF) return 0;
    return 1;
}
void mcf() {
    int x = INF, a, b, y;
    for (int i = from[T]; i; i = from[e[i].from]) {
        x = min(x, e[i].v);
        flow += x;
        if (e[i].from == 0) {
            y = e[i].to;
            a = (y - 1) / tot + 1;
            b = y % tot + 1;
        }
    }
    for (int i = from[T]; i; i = from[e[i].from]) {
```

```
            e[i].v -= x;
            e[i ^ 1].v += x;
            ans += e[i].c * x;
        }
        for (int i = 1; i <= n; i++)
            add_edge((a - 1) * tot + b, m * tot + i, 1, b * t[i][a]);
    }
} // namespace Dynamic_Min_Cost_Max_Flow
using namespace Dynamic_Min_Cost_Max_Flow;
int main() {
    Initialize();
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &c[i]);
        tot += c[i];
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) scanf("%d", &t[i][j]);
    for (int i = 1; i <= m * tot; i++) add_edge(S, i, 1, 0);
    for (int i = 1; i <= n; i++) add_edge(m * tot + i, T, c[i], 0);
    for (int i = 1; i <= m; i++)
        for (int k = 1; k <= n; k++)
            add_edge((i - 1) * tot + 1, m * tot + k, 1, t[k][i]);
    while (spfa()) mcf();
    printf("%d", ans);
    return 0;
}
```

4.12 上下界费用流

```
/*
    上下界费用流
    Initialize()
    add_edge(x,y,下界,上界,cost)
    Rebuild()
    Run()
    ans=-1表示无解
*/
namespace ULB_Min_Cost_Max_Flow {
const int INF = 0x3f3f3f3f;
int V, SS, TT, S, T, cnt, ans, D[N], d[N], from[N], g[N], flow;
bool in[N];
struct edge {
    int from, to, nxt, c, v;
} e[M];
```

```
void add(int u, int v, int w, int c) {
    e[++cnt].from = u;
    e[cnt].to = v;
    e[cnt].nxt = g[u];
    g[u] = cnt;
    e[cnt].c = c;
    e[cnt].v = w;
}
void add_edge(int u, int v, int wl, int wr, int c) {
    D[u] -= wl;
    D[v] += wl;
    add(u, v, wr - wl, c);
    add(v, u, 0, -c);
}
void Rebuild() {
    SS = V + 1;
    TT = V + 2;
    for (int i = 1; i <= V; i++) {
        if (D[i] < 0) add(i, TT, -D[i], 0), add(TT, i, 0, 0);
        if (D[i] > 0) add(SS, i, D[i], 0), add(i, SS, 0, 0);
    }
}
bool spfa() {
    memset(d, INF, sizeof(d));
    d[SS] = 0;
    memset(from, 0, sizeof(from));
    queue<int> q;
    q.push(SS);
    while (!q.empty()) {
        int now = q.front();
        q.pop();
        for (int i = g[now]; i; i = e[i].nxt) {
            if (e[i].v && d[e[i].to] > d[now] + e[i].c) {
                d[e[i].to] = d[now] + e[i].c;
                from[e[i].to] = i;
                q.push(e[i].to);
            }
        }
    }
    return (d[TT] != INF);
}
void mcf() {
    int x = INF;
    for (int i = from[TT]; i; i = from[e[i].from]) x = min(x, e[i].v);
    flow += x;
    for (int i = from[TT]; i; i = from[e[i].from]) {
```

```
            e[i].v -= x;
            e[i ^ 1].v += x;
            ans += e[i].c * x;
        }
    }
    void Initialize() {
        memset(g, 0, sizeof(g));
        memset(e, 0, sizeof(e));
        memset(D, 0, sizeof(D));
        ans = flow = 0;
        cnt = 1;
        V = 0;
        S = ++V;
        T = ++V;
        add_edge(T, S, 0, INF, 0);
    }
    void Run() {
        while (spfa()) mcf();
        for (int i = g[SS]; i; i = e[i].nxt)
            if (e[i].v) ans = -1;
    }
} // namespace ULB_Min_Cost_Max_Flow
```

4.13 ZKW 费用流

```
/*
    ZKW费用流
*/
const int maxint = ~0U >> 1;
int n, m, pi1, cost = 0;
bool v[550];
struct etype {
    int t, c, u;
    etype *next, *pair;
    etype() {}
    etype(int t_, int c_, int u_, etype *next_)
        : t(t_), c(c_), u(u_), next(next_) {}
    void *operator new(unsigned, void *p) { return p; }
} * e[550];
int aug(int no, int m) {
    if (no == n) return cost += pi1 * m, m;
    v[no] = true;
    int l = m;
    for (etype *i = e[no]; i; i = i->next)
        if (i->u && !i->c && !v[i->t]) {
```

```
            int d = aug(i->t, l < i->u ? l : i->u);
            i->u -= d, i->pair->u += d, l -= d;
            if (!l) return m;
        }
    return m - l;
}
bool modlabel() {
    int d = maxint;
    for (int i = 1; i <= n; ++i)
        if (v[i])
            for (etype *j = e[i]; j; j = j->next)
                if (j->u && !v[j->t] && j->c < d) d = j->c;
    if (d == maxint) return false;
    for (int i = 1; i <= n; ++i)
        if (v[i])
            for (etype *j = e[i]; j; j = j->next) j->c -= d, j->pair->c += d;
    pi1 += d;
    return true;
}
int main() {
    scanf("%d %d", &n, &m);
    etype *Pe = new etype[m + m];
    while (m--) {
        int s, t, c, u;
        scanf("%d%d%d%d", &s, &t, &u, &c);
        e[s] = new (Pe++) etype(t, c, u, e[s]);
        e[t] = new (Pe++) etype(s, -c, 0, e[t]);
        e[s]->pair = e[t];
        e[t]->pair = e[s];
    }
    do
        do
            memset(v, 0, sizeof(v));
        while (aug(1, maxint));
    while (modlabel());
    printf("%d\n", cost);
    return 0;
}
```

4.14 最小割树

```
/*
    最小割树
    GomoryHu Tree
    计算出任意两点间的最小割(最大流)
```

```
   Example: 给出一张无向图，求任意两点间最小割minCut(s,t)<=x的点对数
*/
#include <bits/stdc++.h>
using namespace std;
#define REP(i, a) for (int i = 0, _a = (a); i < _a; i++)
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
const int INF = 1000000000;
struct Edge {
   int a, b, cap, flow;
};
struct MaxFlow {
   // MaxFlow flow(n)
   // For each edge: flow.addEdge(u, v, c)
   // Index from 0
   int n, s, t;
   vector<int> d, ptr, q;
   vector<Edge> e;
   vector<vector<int> > g;
   MaxFlow(int n) : n(n), d(n), ptr(n), q(n), g(n) {
      e.clear();
      REP(i, n) {
         g[i].clear();
         ptr[i] = 0;
      }
   }
   void addEdge(int a, int b, int cap) {
      Edge e1 = {a, b, cap, 0};
      Edge e2 = {b, a, 0, 0};
      g[a].push_back((int)e.size());
      e.push_back(e1);
      g[b].push_back((int)e.size());
      e.push_back(e2);
   }
   int getMaxFlow(int _s, int _t) {
      s = _s;
      t = _t;
      int flow = 0;
      for (;;) {
         if (!bfs()) break;
         REP(i, n) ptr[i] = 0;
         while (int pushed = dfs(s, INF)) flow += pushed;
      }
      return flow;
   }

  private:
```

```
    bool bfs() {
        int qh = 0, qt = 0;
        q[qt++] = s;
        REP(i, n) d[i] = -1;
        d[s] = 0;
        while (qh < qt && d[t] == -1) {
            int v = q[qh++];
            REP(i, g[v].size()) {
                int id = g[v][i], to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;
                    d[to] = d[v] + 1;
                }
            }
        }
        return d[t] != -1;
    }
    int dfs(int v, int flow) {
        if (!flow) return 0;
        if (v == t) return flow;
        for (; ptr[v] < (int)g[v].size(); ++ptr[v]) {
            int id = g[v][ptr[v]], to = e[id].b;
            if (d[to] != d[v] + 1) continue;
            int pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
            if (pushed) {
                e[id].flow += pushed;
                e[id ^ 1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
};
const int MN = 211;
const int oo = 1000111000;
struct GomoryHu {
    int ok[MN], cap[MN][MN];
    int answer[MN][MN], parent[MN];
    int n;
    MaxFlow flow;
    GomoryHu(int n) : n(n), flow(n) {
        for (int i = 0; i < n; ++i) ok[i] = parent[i] = 0;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) cap[i][j] = 0, answer[i][j] = INF;
    }
    void addEdge(int u, int v, int c) { cap[u][v] += c; }
```

```
    void calc() {
        for (int i = 0; i < n; ++i) parent[i] = 0;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) answer[i][j] = 2000111000;
        for (int i = 1; i <= n - 1; ++i) {
            flow = MaxFlow(n);
            REP(u, n) REP(v, n) if (cap[u][v]) flow.addEdge(u, v, cap[u][v]);
            int f = flow.getMaxFlow(i, parent[i]);
            bfs(i);
            for (int j = i + 1; j < n; ++j)
                if (ok[j] && parent[j] == parent[i]) parent[j] = i;
            answer[i][parent[i]] = answer[parent[i]][i] = f;
            for (int j = 0; j < i; ++j)
                answer[i][j] = answer[j][i] = min(f, answer[parent[i]][j]);
        }
    }
    void bfs(int start) {
        memset(ok, 0, sizeof ok);
        queue<int> qu;
        qu.push(start);
        while (!qu.empty()) {
            int u = qu.front();
            qu.pop();
            for (int xid = 0; xid < flow.g[u].size(); ++xid) {
                int id = flow.g[u][xid];
                int v = flow.e[id].b, fl = flow.e[id].flow,
                    cap = flow.e[id].cap;
                if (!ok[v] && fl < cap) {
                    ok[v] = 1;
                    qu.push(v);
                }
            }
        }
    }
};
int main() {
    ios::sync_with_stdio(false);
    int ntest;
    cin >> ntest;
    while (ntest--) {
        int n, m;
        cin >> n >> m;
        GomoryHu gh(n);
        while (m--) {
            int u, v, c;
            cin >> u >> v >> c;
```

```
            --u;
            --v;
            gh.addEdge(u, v, c);
            gh.addEdge(v, u, c);
        }
        gh.calc();
        int q;
        cin >> q;
        while (q--) {
            int t;
            cin >> t;
            int res = 0;
            REP(i, n) FOR(j, i + 1, n - 1) if (gh.answer[i][j] <= t) { ++res; }
            cout << res << endl;
        }
        cout << endl;
    }
}
```

4.15 全局最小割

```
/*
    全局最小割
    -邻接矩阵-
*/
namespace Stoer_Wagner() {
    const int INF = 0x3f3f3f3f;
    const int MAX_N = 510;
    int N, M;
    int G[MAX_N][MAX_N], v[MAX_N], f[MAX_N], vis[MAX_N];
    int Stoer_Wagner() {
        int ret = INF;
        for (int i = 1; i <= N; i++) v[i] = i;
        while (N > 1) {
            int k, lst = v[1];
            memset(vis, 0, sizeof(vis));
            memset(f, 0, sizeof(f));
            vis[v[1]] = 1;
            for (int i = 2; i <= N; i++) {
                k = 1;
                for (int j = 2; j <= N; j++)
                    if (!vis[v[j]] && (f[v[j]] += G[lst][v[j]]) > f[v[k]])
                        k = j;
                vis[v[k]] = 1;
                if (i < N) lst = v[k];
```

```
            }
            ret = min(ret, f[v[k]]);
            for (int j = 1; j <= N; j++)
                G[v[j]][lst] = G[lst][v[j]] = G[lst][v[j]] + G[v[k]][v[j]];
            v[k] = v[N--];
        }
        return ret;
    }
    void Initialize() {
        memset(G, 0, sizeof(G));
        for (int i = 1; i <= M; i++) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            ++x;
            ++y;
            G[x][y] += z;
            G[y][x] += z;
        }
    }
} // namespace Stoer_Wagner()
```

5 分块

5.1 分块算法

```
/*
    分块
    区间修改+区间查询大于等于c的数字个数
    O(qsqrt(n)log(sqrt(n))
*/
int n, m, q, block;
const int N = 1000010;
int a[N], b[N], pos[N], add[N];
void Sortblock(int x) {
    int l = (x - 1) * block + 1, r = min(x * block, n);
    for (int i = l; i <= r; i++) b[i] = a[i];
    sort(b + l, b + r + 1);
}
int find(int x, int v) {
    int l = (x - 1) * block + 1, r = min(x * block, n);
    int R = r;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (b[mid] < v)
            l = mid + 1;
        else
            r = mid - 1;
    }
    return R - l + 1;
}
void update(int x, int y, int v) {
    if (pos[x] == pos[y]) {
        for (int i = x; i <= y; i++) a[i] += v;
    } else {
        for (int i = x; i <= pos[x] * block; i++) a[i] += v;
        for (int i = (pos[y] - 1) * block + 1; i <= y; i++) a[i] += v;
    }
    Sortblock(pos[x]);
    Sortblock(pos[y]);
    for (int i = pos[x] + 1; i < pos[y]; i++) add[i] += v;
}
int query(int x, int y, int v) {
    int res = 0;
    if (pos[x] == pos[y]) {
        for (int i = x; i <= y; i++)
            if (a[i] + add[pos[i]] >= v) res++;
    } else {
```

```
        for (int i = x; i <= pos[x] * block; i++)
            if (a[i] + add[pos[i]] >= v) res++;
        for (int i = (pos[y] - 1) * block + 1; i <= y; i++)
            if (a[i] + add[pos[i]] >= v) res++;
    }
    for (int i = pos[x] + 1; i < pos[y]; i++) res += find(i, v - add[i]);
    return res;
}
int main() {
    while (~scanf("%d%d", &n, &q)) {
        block = int(sqrt(n));
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            pos[i] = (i - 1) / block + 1;
            b[i] = a[i];
        }
        if (n % block)
            m = n / block + 1;
        else
            m = n / block;
        for (int i = 1; i <= m; i++) Sortblock(i);
        for (int i = 1; i <= q; i++) {
            char op[5];
            int x, y, v;
            scanf("%s%d%d%d", op, &x, &y, &v);
            if (op[0] == 'M')
                update(x, y, v);
            else
                printf("%d\n", query(x, y, v));
        }
    }
    return 0;
}
```

5.2 块重构

```
/*
    区间查询区间求和
    用栈存储修改操作
    当数量大于限定时重构前缀和
    复杂度O(nsqrt(n))
*/
typedef long long LL;
char op[3];
struct data {
```

```
    int x, y;
    LL c;
} st[100005];
int N, Q, top = 0, size, x, y;
LL s[100010], a[100010];
int main() {
    scanf("%d%d", &N, &Q);
    size = (int)sqrt(N * 1.0);
    for (int i = 1; i <= N; i++) {
        scanf("%lld", &s[i]);
        s[i] += s[i - 1];
    }
    while (Q--) {
        scanf(" %s", op);
        if (op[0] == 'C') {
            ++top;
            scanf("%d%d%lld", &st[top].x, &st[top].y, &st[top].c);
            a[st[top].x] += st[top].c;
            a[st[top].y + 1] -= st[top].c;
            if (top > size) {
                LL tmp = 0;
                for (int i = 1; i <= N; i++) {
                    a[i] += a[i - 1];
                    tmp += a[i];
                    s[i] += tmp;
                }
                for (int i = 1; i <= N; i++) a[i] = 0;
                top = 0;
            }
        } else {
            scanf("%d%d", &x, &y);
            LL ans = s[y] - s[x - 1];
            for (int i = 1; i <= top; i++) {
                if (y < st[i].x || x > st[i].y) continue;
                ans +=
                    st[i].c * (LL)(st[i].y - st[i].x + 1 - max(st[i].y - y, 0) -
                                   max(x - st[i].x, 0));
            }
            printf("%lld\n", ans);
        }
    }
    return 0;
}
```

5.3 第 K 大

```
/*
    求静态区间第k大
*/
const int B = 1000, N = 100010, M = 5010;
int n, m, A[N], num[N], l, r, k;
vector<int> block[N / B];
int main() {
    while (~scanf("%d%d", &n, &m)) {
        memset(block, 0, sizeof(block));
        for (int i = 0; i < n; i++) {
            scanf("%d", &A[i]);
            block[i / B].push_back(A[i]);
            num[i] = A[i];
        }
        sort(num, num + n);
        for (int i = 0; i < n / B; i++) sort(block[i].begin(), block[i].end());
        for (int i = 0; i < m; i++) {
            scanf("%d%d%d", &l, &r, &k);
            l--;
            int L = 0, R = n - 1, ans = 0;
            while (L <= R) {
                int mid = (L + R) >> 1, x = num[mid];
                int tl = l, tr = r, c = 0;
                while (tl < tr && tl % B != 0)
                    if (A[tl++] <= x) c++;
                while (tl < tr && tr % B != 0)
                    if (A[--tr] <= x) c++;
                while (tl < tr) {
                    int b = tl / B;
                    tl += B;
                    c += upper_bound(block[b].begin(), block[b].end(), x) -
                         block[b].begin();
                }
                if (c >= k)
                    ans = mid, R = mid - 1;
                else
                    L = mid + 1;
            }
            printf("%d\n", num[ans]);
        }
    }
    return 0;
}
```

5.4 动态第 K 大

```
/*
    区间修改区间第k大
    二分答案+分块
    Example:
        给出一棵树，可以增加一条边的边长（询问给出一个点，修改的边为其和父节点之间的边）
        或者查询一个子树中深度第k小的点
*/
typedef pair<int, int> P;
const int N = 100010, K = 12, M = (N >> K) + 5;
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
int g[N], w[N], nxt[N], stq[N], enq[N], dfn;
P a[N << 1];
void add(int x, int y, int z) {
    w[y] = z;
    nxt[y] = g[x];
    g[x] = y;
}
void dfs(int x, int y) {
    stq[x] = ++dfn;
    a[dfn] = P(y, dfn);
    for (int i = g[x]; i; i = nxt[i]) dfs(i, y + w[i]);
    enq[x] = dfn;
}
int n, m, lim, op, x, y;
int block, st[M], en[M], tag[M];
inline void change(int o, int l, int r, int p) {
    static P A[N], B[N];
    int ca = 0, cb = 0, i, j, k = st[o];
    for (i = st[o]; i <= en[o]; i++) {
```

```
        if (a[i].second < l || a[i].second > r)
            A[++ca] = a[i];
        else
            B[++cb] = a[i], B[cb].first += p;
    }
    i = j = 1;
    while (i <= ca && j <= cb) a[k++] = A[i] < B[j] ? A[i++] : B[j++];
    while (i <= ca) a[k++] = A[i++];
    while (j <= cb) a[k++] = B[j++];
}
inline void modify(int x, int y, int p) {
    int X = x >> K, Y = y >> K, i;
    for (i = X + 1; i < Y; i++) tag[i] += p;
    change(X, x, y, p);
    if (X < Y) change(Y, x, y, p);
}
inline int ask(int o, int p) {
    int l = st[o], r = en[o], mid, t = l - 1;
    if (l > r) return 0;
    p -= tag[o];
    while (l <= r) {
        if (a[mid = (l + r) >> 1].first <= p)
            l = (t = mid) + 1;
        else
            r = mid - 1;
    }
    return t - st[o] + 1;
}
inline int kth(int x, int y, int k) {
    if (k > y - x + 1) return -1;
    int X = x >> K, Y = y >> K, i, s = n + 1, e = n;
    tag[block + 1] = tag[X];
    for (i = st[X]; i <= en[X]; i++)
        if (a[i].second >= x && a[i].second <= y) a[++e] = a[i];
    st[block + 1] = s, en[block + 1] = e;
    s = e + 1;
    if (X < Y) {
        tag[block + 2] = tag[Y];
        for (i = st[Y]; i <= en[Y]; i++)
            if (a[i].second <= y) a[++e] = a[i];
    }
    st[block + 2] = s, en[block + 2] = e;
    int l = 0, r = lim, mid, t, ans;
    while (l <= r) {
        mid = (l + r) >> 1;
        t = ask(block + 1, mid) + ask(block + 2, mid);
```

```
        for (i = X + 1; i < Y && t < k; i++) t += ask(i, mid);
        if (t >= k)
            r = (ans = mid) - 1;
        else
            l = mid + 1;
    }
    return ans;
}
int main() {
    read(n), read(m), read(x);
    for (int i = 2; i <= n; i++) read(x), read(y), add(x, i, y), lim += y;
    dfs(1, 0);
    block = n >> K;
    for (int i = 1; i <= n; i++) en[i >> K] = i;
    for (int i = n; i; i--) st[i >> K] = i;
    for (int i = 0; i <= block; i++) sort(a + st[i], a + en[i] + 1);
    while (m--) {
        read(op), read(x), read(y);
        if (op == 1)
            printf("%d\n", kth(stq[x], enq[x], y));
        else
            modify(stq[x], enq[x], y), lim += y;
    }
    return 0;
}
```

5.5 动态逆序对

```
/*
    动态逆序对
    分块+权值树状数组
    一个1到n顺序排列的数列，每次选择两个位置的数进行交换，
    求交换后的数列的逆序对数
*/
const int N = 200010, M = 650;
int n, q, l, r, x, y, arr[N], st[M], en[M], c[M][N], block[N];
void update(int k, int x, int val) {
    while (x < N) c[k][x] += val, x += x & -x;
}
int query(int k, int x) {
    int s = 0;
    while (x) s += c[k][x], x -= x & -x;
    return s;
}
long long ans = 0;
```

```
int query(int l, int r, int x) {
    int res = 0;
    for (int i = block[l]; i <= block[r]; i++) {
        if (st[i] >= l && en[i] <= r)
            res += query(i, x);
        else
            for (int j = max(st[i], l); j <= min(en[i], r); j++)
                res += (arr[j] <= x);
    }
    return res;
}
void cal(int x, int y) {
    ans -= x - 1 - query(1, x - 1, arr[x]);
    ans -= query(x + 1, n, arr[x] - 1);
    update(block[x], y, 1);
    update(block[x], arr[x], -1);
    arr[x] = y;
    ans += x - 1 - query(1, x - 1, arr[x]);
    ans += query(x + 1, n, arr[x] - 1);
}
int main() {
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++) arr[i] = i;
    for (int i = 1, cur = 0; i <= n;) {
        int j = i;
        st[++cur] = i;
        while (j <= n && j < i + M) {
            block[j] = cur;
            update(cur, arr[j++], 1);
        }
        en[cur] = j - 1;
        i = j;
    }
    while (q--) {
        scanf("%d%d", &l, &r);
        if (l != r) {
            int t = arr[l];
            x = l;
            y = arr[r];
            cal(x, y);
            x = r;
            y = t;
            cal(x, y);
        }
        printf("%lld\n", ans);
    }
```

```
    return 0;
}
```

5.6 莫队算法

```
/*
    莫队算法
    求区间内所有数出现次数的平方和
*/
const int N = 50005;
using namespace std;
typedef long long LL;
int pos[N], num[N], n, m, k, limit, i, l, r, S[N];
LL ans;
struct Q {
    int l, r, id;
    LL ans;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[N];
inline bool cmp(const Q &a, const Q &b) { return a.id < b.id; }
inline void read(int &a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'));
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
inline void modify(int p, LL &ans, int add) {
    ans = ans - (LL)S[num[p]] * S[num[p]];
    S[num[p]] += add;
    ans = ans + (LL)S[num[p]] * S[num[p]];
}
int main() {
    read(n);
    read(m);
    read(k);
    limit = (int)sqrt(n + 0.5);
    for (i = 1; i <= n; i++) {
        read(num[i]);
        pos[i] = (i - 1) / limit + 1;
    }
    for (i = 1; i <= m; i++) {
        read(ask[i].l);
        read(ask[i].r);
```

```
        ask[i].id = i;
    }
    ans = 0;
    sort(ask + 1, ask + m + 1);
    for (i = 1, l = 1, r = 0; i <= m; i++) {
        while (r < ask[i].r) modify(++r, ans, 1);
        while (r > ask[i].r) modify(r--, ans, -1);
        while (l < ask[i].l) modify(l++, ans, -1);
        while (l > ask[i].l) modify(--l, ans, 1);
        ask[i].ans = ans;
    }
    sort(ask + 1, ask + m + 1, cmp);
    for (i = 1; i <= m; i++) printf("%lld\n", ask[i].ans);
    return 0;
}
```

5.7 单增莫队算法

```
/*
    单增莫队
    对于左端点在同一个块内的询问，每次还原左端点到块的最右端
    然后移动左端点和右端点到对应位置
    这样就只有区间扩增操作，没有区间缩减操作
    用于非可加类答案的统计
    Example: 求区间内数字*出现次数的最大值
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 100000 + 10;
int limit, block, pos[N], a[N], cnt[N];
ll lst_ans, ans, Ans[N];
int T, n, m;
struct Q {
    int l, r, id;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[N];
int b[N], sa[N];
void Discretization() {
    sort(b + 1, b + n + 1);
    int siz = unique(b + 1, b + n + 1) - (b + 1);
    for (int i = 1; i <= n; i++)
        sa[i] = lower_bound(b + 1, b + siz + 1, a[i]) - b;
```

```
}
void modify(int x) {
    cnt[sa[x]]++;
    if (1ll * cnt[sa[x]] * a[x] > ans) {
        ans = 1ll * cnt[sa[x]] * a[x];
    }
}
int main() {
    scanf("%d%d", &n, &m);
    block = (int)sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        b[i] = a[i];
        pos[i] = (i - 1) / block + 1;
    }
    Discretization();
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &ask[i].l, &ask[i].r);
        ask[i].id = i;
    }
    sort(ask + 1, ask + m + 1);
    int L = 1, R = 0;
    lst_ans = ans = 0;
    ask[0].l = 0;
    pos[0] = 0;
    for (int i = 1; i <= m; ++i) {
        if (pos[ask[i - 1].l] != pos[ask[i].l]) {
            memset(cnt, 0, sizeof(cnt));
            limit = pos[ask[i].l] * block;
            R = limit;
            ans = lst_ans = 0;
        }
        L = min(limit + 1, ask[i].r + 1);
        while (R < ask[i].r) modify(++R);
        lst_ans = ans;
        while (L > ask[i].l) modify(--L);
        Ans[ask[i].id] = ans;
        for (int j = L; j <= ask[i].r && j <= limit; ++j) {
            cnt[sa[j]]--;
        }
        ans = lst_ans;
    }
    for (int i = 1; i <= m; ++i) printf("%lld\n", Ans[i]);
    return 0;
}
```

5.8 莫队算法 + 分块

```
/*
    莫队+分块
    查询[l,r]中[a,b]范围内的数字种类数
*/
const int N = 100001, M = 1000001;
using namespace std;
typedef long long LL;
int size[N], ans[M], pos[N], color[N], c[400], n, m, limit, bl[400], br[400];
struct Q {
    int l, r, id, a, b;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[M];
bool cmp(const Q &a, const Q &b) { return a.id < b.id; }
void read(int &a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
int query(int x, int y) {
    int res = 0;
    int L = pos[x], R = pos[y];
    for (int i = L + 1; i < R; i++) res += c[i];
    if (L == R) {
        for (int i = x; i <= y; i++)
            if (size[i]) res++;
    } else {
        for (int i = x; i <= br[L]; i++)
            if (size[i]) res++;
        for (int i = bl[R]; i <= y; i++)
            if (size[i]) res++;
    }
    return res;
}
void modify(int u, int x) {
    if (size[color[u]] == 1 && x == -1) c[pos[color[u]]]--;
    if (size[color[u]] == 0 && x == 1) c[pos[color[u]]]++;
    size[color[u]] += x;
}
int main() {
    read(n);
```

```
    read(m);
    limit = (int)sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) {
        read(color[i]);
        pos[i] = (i - 1) / limit + 1;
    }
    for (int i = 1; i <= n; i++) {
        br[pos[i]] = i;
        if (!bl[pos[i]]) bl[pos[i]] = i;
    }
    for (int i = 1; i <= m; i++) {
        read(ask[i].l);
        read(ask[i].r);
        read(ask[i].a);
        read(ask[i].b);
        ask[i].id = i;
    }
    sort(ask + 1, ask + m + 1);
    for (int i = 1, l = 1, r = 0; i <= m; i++) {
        while (r < ask[i].r) modify(++r, 1);
        while (r > ask[i].r) modify(r--, -1);
        while (l < ask[i].l) modify(l++, -1);
        while (l > ask[i].l) modify(--l, 1);
        ans[ask[i].id] = query(ask[i].a, ask[i].b);
    }
    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

5.9 动态莫队算法

```
/*
    带修改莫队
    给出一颜色序列，每次可以修改一个位置的颜色或者询问一个区间不同颜色的数目
*/
const int N = 1000010;
using namespace std;
int Ans, ans[N], pos[N], c[N], pre[N], sz, cnt, n, m, limit;
struct Q {
    int l, r, id, t;
    friend bool operator<(const Q &a, const Q &b) {
        if (pos[a.l] != pos[b.l]) return pos[a.l] < pos[b.l];
        if (a.r != b.r) return a.r < b.r;
        return a.t < b.t;
    }
```

```
} ask[N];
struct update {
    int pos, c, pre;
} w[N];
int in[N], num[N];
void cal(int x) {
    if (in[x]) {
        if (!--num[c[x]]) Ans--;
    } else {
        if (++num[c[x]] == 1) Ans++;
    }
    in[x] ^= 1;
}
void update(int x, int y) {
    if (in[x]) {
        cal(x);
        c[x] = y;
        cal(x);
    } else
        c[x] = y;
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        int limit = sqrt(n);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &c[i]);
            pre[i] = c[i], pos[i] = (i - 1) / limit + 1, in[i] = 0;
        }
        cnt = sz = Ans = 0;
        for (int i = 1; i <= m; i++) {
            char op[10];
            scanf("%s", op);
            int x, y;
            scanf("%d%d", &x, &y);
            if (op[0] == 'R') {
                w[++cnt].pos = x;
                w[cnt].c = y;
                w[cnt].pre = pre[x];
                pre[x] = y;
            } else {
                ask[++sz].l = x;
                ask[sz].r = y;
                ask[sz].id = sz;
                ask[sz].t = cnt;
            }
        }
```

```
        sort(ask + 1, ask + sz + 1);
        int l, r;
        cal(l = r = 1);
        for (int i = 1; i <= sz; i++) {
            for (int j = ask[i - 1].t + 1; j <= ask[i].t; j++)
                update(w[j].pos, w[j].c);
            for (int j = ask[i - 1].t; j > ask[i].t; j--)
                update(w[j].pos, w[j].pre);
            while (l < ask[i].l) cal(l++);
            while (l > ask[i].l) cal(--l);
            while (r < ask[i].r) cal(++r);
            while (r > ask[i].r) cal(r--);
            ans[ask[i].id] = Ans;
        }
        for (int i = 1; i <= sz; i++) printf("%d\n", ans[i]);
    }
    return 0;
}
```

5.10 区间逆序对

```
/*
    区间逆序对
    莫队+树状数组
*/
const int N = 50100;
using namespace std;
typedef long long LL;
int pos[N], a[N], disc[N], n, m, limit, i, l, r, c[N];
struct Q {
    int l, r, id;
    LL ans;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[N];
bool cmp(const Q &a, const Q &b) { return a.id < b.id; }
void read(int &a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
void update(int x, int val) {
```

```
    while (x <= n) c[x] += val, x += x & -x;
}
LL query(int x) {
    LL res = 0;
    while (x) res += c[x], x -= x & -x;
    return res;
}
int main() {
    read(n);
    limit = (int)sqrt(n + 0.5);
    for (i = 1; i <= n; i++) {
        read(a[i]);
        pos[i] = (i - 1) / limit + 1;
    }
    for (int i = 1; i <= n; i++) disc[i] = a[i];
    sort(disc + 1, disc + n + 1);
    for (int i = 1; i <= n; i++)
        a[i] = lower_bound(disc + 1, disc + n + 1, a[i]) - disc;
    read(m);
    memset(c, 0, sizeof(c));
    for (int i = 1; i <= m; i++) {
        read(ask[i].l);
        read(ask[i].r);
        ask[i].id = i;
    }
    sort(ask + 1, ask + m + 1);
    LL ans = 0;
    for (int i = 1, l = 1, r = 0; i <= m; i++) {
        while (r < ask[i].r)
            r++, update(a[r], 1), ans += r - l + 1 - query(a[r]);
        while (r > ask[i].r) update(a[r], -1), ans -= r - l - query(a[r]), r--;
        while (l < ask[i].l) update(a[l], -1), ans -= query(a[l] - 1), l++;
        while (l > ask[i].l) l--, update(a[l], 1), ans += query(a[l] - 1);
        ask[i].ans = ans;
    }
    sort(ask + 1, ask + m + 1, cmp);
    for (int i = 1; i <= m; i++) printf("%lld\n", ask[i].ans);
    return 0;
}
```

5.11 二维莫队算法

```
/*
    二维莫队
    给出一个矩阵
```

```
    询问查询给出子矩阵中每个种类数字出现次数的平方和
*/
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
struct W {
    int x, y, v;
} t[40010];
struct Q {
    int x1, x2, y1, y2, p;
} q[100010];
bool vis[210][210];
int a[210][210], bl[210][210], Ans[100010], id[210][210], T[40010];
int n, m, tot, mx, bn, bm, ans;
bool cmp(W a, W b) { return a.v < b.v; }
bool cmp1(Q a, Q b) {
    return bl[a.x1][a.y1] == bl[b.x1][b.y1] ? id[a.x2][a.y2] < id[b.x2][b.y2]
                                        : bl[a.x1][a.y1] < bl[b.x1][b.y1];
}
void insert(int x, int y) {
    if (!vis[x][y]) {
        vis[x][y] = 1;
        ans += 2 * (T[a[x][y]]++) + 1;
    }
}
void del(int x, int y) {
    if (vis[x][y]) {
        vis[x][y] = 0;
        ans -= 2 * (--T[a[x][y]]) + 1;
    }
}
void ADD(int x1, int x2, int y1, int y2) {
    for (int i = x1; i <= x2; i++)
```

```
        for (int j = y1; j <= y2; j++) insert(i, j);
}
void DEL(int x1, int x2, int y1, int y2) {
    for (int i = x1; i <= x2; i++)
        for (int j = y1; j <= y2; j++) del(i, j);
}
void change(Q a, Q b) {
    int t;
    t = min(a.x1 - 1, b.x2);
    ADD(b.x1, t, b.y1, b.y2);
    t = max(a.x2 + 1, b.x1);
    ADD(t, b.x2, b.y1, b.y2);
    t = min(a.y1 - 1, b.y2);
    ADD(b.x1, b.x2, b.y1, t);
    t = max(a.y2 + 1, b.y1);
    ADD(b.x1, b.x2, t, b.y2);
    t = min(b.x1 - 1, a.x2);
    DEL(a.x1, t, a.y1, a.y2);
    t = max(b.x2 + 1, a.x1);
    DEL(t, a.x2, a.y1, a.y2);
    t = min(b.y1 - 1, a.y2);
    DEL(a.x1, a.x2, a.y1, t);
    t = max(b.y2 + 1, a.y1);
    DEL(a.x1, a.x2, t, a.y2);
}
int main() {
    read(n);
    read(m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            id[i][j] = ++tot, t[tot].x = i, t[tot].y = j, read(t[tot].v);
    sort(t + 1, t + tot + 1, cmp);
    for (int i = 1; i <= tot; i++) {
        if (t[i].v != t[i - 1].v)
            a[t[i].x][t[i].y] = ++mx;
        else
            a[t[i].x][t[i].y] = mx;
    }
    bn = sqrt(n), bm = sqrt(m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            bl[i][j] = (i - 1) / bn * bm + (j - 1) / bm;
    int Q;
    read(Q);
    for (int i = 1; i <= Q; i++) {
        q[i].p = i;
```

```
        read(q[i].x1), read(q[i].y1);
        read(q[i].x2), read(q[i].y2);
        if (q[i].x1 > q[i].x2) swap(q[i].x1, q[i].x2);
        if (q[i].y1 > q[i].y2) swap(q[i].y1, q[i].y2);
    }
    sort(q + 1, q + Q + 1, cmp1);
    for (int i = q[1].x1; i <= q[1].x2; i++)
        for (int j = q[1].y1; j <= q[1].y2; j++) insert(i, j);
    Ans[q[1].p] = ans;
    for (int i = 2; i <= Q; i++) {
        change(q[i - 1], q[i]);
        Ans[q[i].p] = ans;
    }
    for (int i = 1; i <= Q; i++) printf("%d\n", Ans[i]);
    return 0;
}
```

5.12 树上莫队算法

```
/*
    树上莫队，注意查询时额外考虑lca位置
    查询树链上不同权值的数量
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
const int N = 100010;
const int K = 17;
using namespace std;
struct P {
    int l, r, z, id;
} Q[N];
int lim, pos[N << 1], l, r, c[N], g[N], v[N << 1], nxt[N << 1], ed;
int n, m, x, y, z, loc[N << 1], dfn, st[N], en[N], d[N], f[N][18];
int ans[N], cnt[N], sum;
bool vis[N];
bool cmp(const P& a, const P& b) {
    return pos[a.l] == pos[b.l] ? a.r < b.r : pos[a.l] < pos[b.l];
}
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
```

```
    for (int i = vis[loc[st[x] = ++dfn] = x] = 1; i <= K; i++)
        f[x][i] = f[f[x][i - 1]][i - 1];
    for (int i = g[x]; i; i = nxt[i])
        if (!vis[v[i]]) d[v[i]] = d[f[v[i]][0] = x] + 1, dfs(v[i]);
    loc[en[x] = ++dfn] = x;
}
int lca(int x, int y) {
    if (x == y) return x;
    if (d[x] < d[y]) swap(x, y);
    for (int i = K; ~i; i--)
        if (d[f[x][i]] >= d[y]) x = f[x][i];
    if (x == y) return x;
    for (int i = K; ~i; i--)
        if (f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
    return f[x][0];
}
void deal(int x) {
    if (!vis[x]) {
        if (!(--cnt[c[x]])) sum--;
    } else if (!(cnt[c[x]]++))
        sum++;
    vis[x] ^= 1;
}
void read(int& x) {
    int f = 1;
    x = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if ('-' == ch) f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + ch - '0';
        ch = getchar();
    }
}
int b[N];
int main() {
    read(n), read(m);
    for (int i = 1; i <= n; i++) read(c[i]), b[i] = c[i];
    sort(b + 1, b + n + 1);
    int siz = unique(b + 1, b + n + 1) - (b + 1);
    for (int i = 1; i <= n; i++)
        c[i] = lower_bound(b + 1, b + siz + 1, c[i]) - b;
    for (int i = 1; i < n; i++) read(x), read(y), add(x, y), add(y, x);
    dfs(d[1] = 1), lim = (int)sqrt(n * 2 + 0.5);
```

```
    for (int i = 1; i <= dfn; i++) pos[i] = (i - 1) / lim + 1;
    for (int i = 1; i <= m; i++) {
        read(x), read(y);
        Q[i].id = i;
        if (st[x] > st[y]) swap(x, y);
        z = lca(x, y);
        if (z == x)
            Q[i].l = st[x], Q[i].r = st[y];
        else
            Q[i].l = en[x], Q[i].r = st[y], Q[i].z = z;
    }
    sort(Q + 1, Q + m + 1, cmp);
    for (int i = 1, l = 1, r = 0; i <= m; i++) {
        if (r < Q[i].r) {
            for (r++; r <= Q[i].r; r++) deal(loc[r]);
            r--;
        }
        if (r > Q[i].r)
            for (; r > Q[i].r; r--) deal(loc[r]);
        if (l < Q[i].l)
            for (; l < Q[i].l; l++) deal(loc[l]);
        if (l > Q[i].l) {
            for (l--; l >= Q[i].l; l--) deal(loc[l]);
            l++;
        }
        if (Q[i].z) deal(Q[i].z);
        ans[Q[i].id] = sum;
        if (Q[i].z) deal(Q[i].z);
    }
    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

5.13 基于位运算的 LCS

```
/*
    基于位运算的LCS
    复杂度O(n1*n2/E)
*/
typedef long long ll;
const int BIT=808,E=62;
int m,h;
struct Num{
    ll x[BIT];
    Num(){for(int i=0;i<BIT;i++)x[i]=0;}
```

```
    inline void set(int p){x[p/E]|=1LL<<(p%E);}
    inline Num operator&(Num b){
        Num c;
        for(int i=0;i<=m;i++)c.x[i]=x[i]&b.x[i];
        return c;
    }
    inline Num operator|(Num b){
        Num c;
        for(int i=0;i<=m;i++)c.x[i]=x[i]|b.x[i];
        return c;
    }
    inline Num operator^(Num b){
        Num c;
        for(int i=0;i<=m;i++)c.x[i]=x[i]^b.x[i];
        return c;
    }
    inline Num operator-(Num b){
        Num c;
        for(int i=0;i<=m;i++)c.x[i]=x[i]-b.x[i];
        for(int i=0;i<m;i++)if(c.x[i]<0)c.x[i]+=(1LL<<E),c.x[i+1]--;
        return c;
    }
    inline void shl(){
        ll o=1,p;
        for(int i=0;i<=m;o=p,i++){
            p=x[i]&(1LL<<h),(x[i]<<=1)&=~(1LL<<(h+1));
            if(o)x[i]|=1;
        }
    }
}ap[4],x,row[2];
inline int hash(int x){
    if(x=='A')return 0;
    if(x=='C')return 1;
    if(x=='G')return 2;
    return 3;
}
int n1,n2,p,ans;
char s[50000],t[50000];
int main(){
    scanf("%d%d%s%s",&n1,&n2,s,t);
    m=(n1-1)/E;
        h=(m?E:n1)-1;
    for(int i=0;i<n1;i++)ap[hash(s[i])].set(i);
    for(int i=0;i<n2;i++){
        p^=1;
        x=ap[hash(t[i])]|row[p^1];
```

```
        row[p^1].shl();
        row[p]=x&((x-row[p^1])^x);
    }
    for(int i=m;~i;i--){
        for(int j=h;~j;j--){
            if(row[p].x[i]&(1LL<<j))ans++;
        }
    }
    printf("%d",ans);
    return 0;
}
```

5.14 Bitset

```
/*
    题目大意：给出一个数列，有三种区间查询，
    分别查询区间是否存在两个数乘积为x，是否存在两个数和为x，以及是否存在两个数差为x，
    题解：我们对于询问进行莫队处理，保存当前区间的权值数组，记为F，
    同时保存权值数组的反向数组G
    那么存在差为x的情况只要存在一组F[i]&F[i-x]=1即可
    存在和为x的情况只要存在一组F[i]&G[M-x+i]即可。
    对于乘积为x的情况，我们枚举x的约数，判断F[i]&F[x/i]是否存在。
    复杂度O(nsqrt(n)+nm/w)
*/
#include <algorithm>
#include <bitset>
#include <cmath>
#include <cstdio>
using namespace std;
const int N = 100010, M = 100000;
int limit, n, m, pos[N], a[N], ans[N], cnt[N];
bitset<N> F, G;
struct Q {
    int l, r, x, id, op;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[M];
int read(int &x) {
    int f = 1;
    char ch = getchar();
    x = 0;
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
```

```
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    x *= f;
}
int main() {
    read(n);
    read(m);
    limit = (int)sqrt(n + 0.5);
    for (int i = 1; i <= n; i++) read(a[i]), pos[i] = (i - 1) / limit + 1;
    for (int i = 1; i <= m; i++)
        read(ask[i].op), read(ask[i].l), read(ask[i].r), read(ask[i].x),
            ask[i].id = i;
    sort(ask + 1, ask + m + 1);
    for (int i = 1, l = 1, r = 0; i <= m; i++) {
        for (; r < ask[i].r; r++)
            cnt[a[r + 1]]++, F.set(a[r + 1]), G.set(M - a[r + 1]);
        for (; l > ask[i].l; l--)
            cnt[a[l - 1]]++, F.set(a[l - 1]), G.set(M - a[l - 1]);
        for (; l < ask[i].l; l++) {
            cnt[a[l]]--;
            if (!cnt[a[l]]) F.reset(a[l]), G.reset(M - a[l]);
        }
        for (; r > ask[i].r; r--) {
            cnt[a[r]]--;
            if (!cnt[a[r]]) F.reset(a[r]), G.reset(M - a[r]);
        }
        if (ask[i].op == 1) {
            if ((F & (F >> ask[i].x)).any()) ans[ask[i].id] = 1;
        } else if (ask[i].op == 2) {
            if ((F & (G >> (M - ask[i].x))).any()) ans[ask[i].id] = 1;
        } else {
            for (int j = 1; j * j <= ask[i].x; j++)
                if (ask[i].x % j == 0) {
                    if (F[j] & F[ask[i].x / j]) {
                        ans[ask[i].id] = 1;
                        break;
                    }
                }
        }
    }
    for (int i = 1; i <= m; i++) puts(ans[i] ? "yuno" : "yumi");
    return 0;
}
```

```
/*
    题目大意：给出一个数组a一个数组b，以及询问数组c，
    问对于每个c有多少对a%b=c，答案对2取模
    题解：考虑对2取模我们发现答案等价于数字的xor，01状态可以用bitset保存，
    在bitset上存a的权值数组，那么每次只要将b*i~b*(i+1)-1的数值xor到答案数组的0~b-1上去即可，
    鉴于bitset没有截取区间的功能，我们手写压位，
    考虑压32位之后不是32倍数的部分截取起来非常尴尬，
    因此我们保存其偏移量为0~31的数组，这样取区间就比较方便了。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 10010;
struct Bitset {
    unsigned int u[N];
    void reset() { memset(u, 0, sizeof(u)); }
    void set(int x) { u[x >> 5] |= 1 << (x & 31); }
    void flip(int x) { u[x >> 5] ^= 1 << (x & 31); }
    bool test(int x) { return u[x >> 5] & (1 << (x & 31)); }
    void reset(int x) {
        if (test(x)) flip(x);
    }
} a[32], ans;
void Solve(int l, int r) {
    while ((r - l) & 31) {
        r--;
        if (a[0].test(r)) ans.flip(r - l);
    }
    int m = 0;
    while (l & 31) l++, r++, m++;
    l >>= 5;
    r >>= 5;
    for (int i = l; i < r; i++) ans.u[i - l] ^= a[m].u[i];
}
int T, n, m, q, x, mx;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d", &n, &m, &q);
        for (int i = 0; i < 32; i++) a[i].reset();
        ans.reset();
        mx = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            mx = max(x, mx);
```

```
            for (int j = 0; j < 32; j++) a[j].set(x + j);
        }
        for (int i = 1; i <= m; i++) {
            scanf("%d", &x);
            for (int j = 0; j <= mx; j += x) Solve(j, min(mx + 1, j + x));
        }
        while (q--) {
            scanf("%d", &x);
            if (ans.test(x))
                puts("1");
            else
                puts("0");
        }
    }
    return 0;
}
/*
    题目大意:
        给出一个数列，每个询问给出三个区间，问除去三个区间共有的数字外，
        还剩下几个数字，注意删去的是共有的数字个数，不是数字种类，统计时候也一样
    题解:
        首先，答案为区间长度和减去区间并数字个数和的三倍。
        所以题目转化为求区间并。很显然在开始对数据可以进行离散化。
        考虑每个数字只出现一次的情况，我们可以用bitset来统计区间某个数字是否存在，
        莫队处理查询每个区间，保存其bitset的值，最后求交即可，
        现在考虑每个数字出现多次的情况，
        我们发现经过离散的数据之间空位数量恰好可以用来标出现多次的数据，
        比如1 4 4 9 9，离散后为 1 2 2 4 4，
        我们可以将多出来的2标在3位置，4标在5位置，那么就可以用bitset统计了。
        - Me : 询问区间存不下怎么办?
        - Claris : 将询问分批进行处理，单次处理25000个询问
        - Me : 超时了欸……
        - Claris : 这题卡常数，要手写bitset.
        - Me : 你的代码为什么有6.7k?
        - Claris : 我分出现一次，两次和跟多次讨论
        - Me : 我……还是咸鱼吧……
*/
#include <algorithm>
#include <bitset>
#include <cmath>
#include <cstdio>
using namespace std;
typedef unsigned long long ULL;
const int N = 100010, M = N << 2;
int limit, n, m, pos[N], a[N], cnt[N], Ans[N], mark[N];
struct Q {
```

```
    int l, r, id;
    friend bool operator<(const Q &a, const Q &b) {
        return pos[a.l] < pos[b.l] || (pos[a.l] == pos[b.l] && a.r < b.r);
    }
} ask[M];
int read(int &x) {
    int f = 1;
    char ch = getchar();
    x = 0;
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    x *= f;
}
int disc[N];
int remark(int x) {
    int l = 1, r = n, res = 0;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (disc[mid] < x)
            l = mid + 1;
        else
            res = mid, r = mid - 1;
    }
    return res;
}
const int B = 1567, K = 25000;
ULL v[B], f[K + 3][B];
int u[65537], tmp, U;
void flip(int x) { v[x >> 6] ^= 1ULL << (x & 63); }
void Copy(ULL *a) {
    int i = 0;
    for (; i + 13 <= U; i += 14) {
        for (int j = 0; j < 14; j++) a[i + j] = v[i + j];
    }
    for (; i <= U; i++) a[i] = v[i];
}
void And(ULL *a) {
    int i = 0;
    for (; i + 13 <= U; i += 14) {
        for (int j = 0; j < 14; j++) a[i + j] &= v[i + j];
```

```
    }
    for (; i <= U; i++) a[i] &= v[i];
}
void popcount(ULL x) {
    tmp += u[x & 65535] + u[x >> 16 & 65535] + u[x >> 32 & 65535] + u[x >> 48];
}
int count(ULL *a) {
    int i = tmp = 0;
    for (; i + 13 <= U; i += 14) {
        for (int j = 0; j < 14; j++) popcount(a[i + j]);
    }
    for (; i <= U; i++) popcount(a[i]);
    return tmp;
}
void init() {
    for (int i = 1; i < 65536; i++) u[i] = u[i >> 1] + (i & 1);
}
int main() {
    read(n);
    read(m);
    U = n >> 6;
    init();
    limit = (int)sqrt(n + 0.5);
    for (int i = 1; i <= n; i++)
        read(a[i]), disc[i] = a[i], pos[i] = (i - 1) / limit + 1;
    sort(disc + 1, disc + n + 1);
    for (int i = 1; i <= n; i++) a[i] = remark(a[i]);
    int pos = 0, l = 1, r = 0;
    while (pos < m) {
        int tot = 0;
        for (int i = 1; i <= 25000 && i + pos <= m; i++) {
            tot += 3;
            Ans[i] = 0;
            mark[i] = 0;
            read(ask[i * 3 - 2].l);
            read(ask[i * 3 - 2].r);
            ask[i * 3 - 2].id = i;
            read(ask[i * 3 - 1].l);
            read(ask[i * 3 - 1].r);
            ask[i * 3 - 1].id = i;
            read(ask[i * 3].l);
            read(ask[i * 3].r);
            ask[i * 3].id = i;
            Ans[i] += ask[i * 3 - 2].r - ask[i * 3 - 2].l + 1;
            Ans[i] += ask[i * 3 - 1].r - ask[i * 3 - 1].l + 1;
            Ans[i] += ask[i * 3].r - ask[i * 3].l + 1;
```

```
        }
        sort(ask + 1, ask + tot + 1);
        for (int i = 1; i <= tot; i++) {
            for (; r < ask[i].r; r++) {
                flip(a[r + 1] + cnt[a[r + 1]]);
                cnt[a[r + 1]]++;
            }
            for (; l > ask[i].l; l--) {
                flip(a[l - 1] + cnt[a[l - 1]]);
                cnt[a[l - 1]]++;
            }
            for (; l < ask[i].l; l++) {
                cnt[a[l]]--;
                flip(a[l] + cnt[a[l]]);
            }
            for (; r > ask[i].r; r--) {
                cnt[a[r]]--;
                flip(a[r] + cnt[a[r]]);
            }
            if (mark[ask[i].id])
                And(f[ask[i].id]);
            else
                Copy(f[ask[i].id]), mark[ask[i].id] = 1;
        }
        tot /= 3;
        for (int i = 1; i <= tot; i++) printf("%d\n", Ans[i] - 3 * count(f[i]));
        pos += tot;
    }
    return 0;
}
```

6 数学

6.1 快速乘法

```
/*
    快速乘法
*/
// O(1)
LL mul(LL x, LL y, LL mod) {
    return (x * y - (LL)(x / (long double)mod * y + 1e-3) * mod + mod) % mod;
}
// O(logn)
LL mul(LL x, LL y, LL mod) {
    LL ret = 0;
    for (; y; y >>= 1) {
        if (y & 1) ret = (ret + x) % mod;
        x = (x + x) % mod;
    }
    return ret;
}
```

6.2 乘法逆元

```
/*
    预处理的对于模P的逆元
*/
int inv[N], mp[N];
void pre_inv() {
    inv[1] = 1;
    for (int i = 2; i < P; i++) {
        int a = P / i, b = P % i;
        inv[i] = (1LL * inv[b] * (-a) % P + P) % P;
    }
}
```

6.3 组合数

```
/*
    组合数
*/
namespace Comb {
const int U = 200000;
int f[U + 3], rf[U + 3];
```

```
LL inv(LL a, LL m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
void init() {
    f[0] = 1;
    for (int i = 1; i <= U; i++) f[i] = (LL)f[i - 1] * i % P;
    rf[U] = inv(f[U], P);
    for (int i = U; i; i--) rf[i - 1] = (LL)rf[i] * i % P;
}
LL C(int n, int m) {
    if (m < 0 || m > n) return 0;
    return (LL)f[n] * rf[m] % P * rf[n - m] % P;
}
} // namespace Comb
/*
    Problem:
        求解 _(1,n) _(i+1,n)C(Ai+Aj+Bi+Bj,Ai+Aj), A,B<=2000, n<=200000
    Solution:
        C(Ai+Aj+Bi+Bj,Ai+Aj)在方格上表示即从(-Ai,-Bi)开始
        只能向右走或者向下走走到(Aj,Bj)的方案数,
        有设dp[(起点坐标)]=1, dp[i][j]=dp[i-1][j]+dp[i][j-1], Ans=dp[(终点坐标)]
        对于每一个组我们可以建立起点(-A[i],-B[i])与终点(A[i],B[i]), 起点初始化为1
        直接dp计算即可, 考虑到多算了(-A[i],-B[i])到(A[i],B[i])的方案数,
        所以我们要容斥掉C(A[i]+A[i]+B[i]+B[i],A[i]+A[i]), 最后考虑偏序, 答案除2即可
*/
const int base = 2000, P = 1e9 + 7, N = 200010;
int dp[4010][4010], a[N], b[N];
using namespace Comb;
inline void up(int &x) {
    while (x < 0) x += P;
    while (x >= P) x -= P;
}
int main() {
    int n;
    init();
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d%d", &a[i], &b[i]);
        dp[base - a[i]][base - b[i]]++;
    }
    for (int i = 0; i <= base + base; i++) {
        for (int j = 0; j <= base + base; j++) {
            if (i) dp[i][j] += dp[i - 1][j], up(dp[i][j]);
            if (j) dp[i][j] += dp[i][j - 1], up(dp[i][j]);
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
```

```
        ans += dp[a[i] + base][b[i] + base];
        up(ans);
        ans -= C(a[i] + a[i] + b[i] + b[i], a[i] + a[i]);
        up(ans);
    }
    printf("%lld\n", 1ll * ans * rf[2] % P);
    return 0;
}
```

6.4 Lucas 定理

```
/*
    Lucas定理
    lucas(n,m,P)返回C(n,m)对P取摸的结果
*/
namespace Lucas {
LL f[N], rf[N];
LL mul(LL x, LL y, LL P) {
    return (x * y - (LL)(x / (long double)P * y + 1e-3) * P + P) % P;
}
LL pow(LL a, LL b, LL P) {
    LL t = 1;
    for (; b; b >>= 1, a = mul(a, a, P))
        if (b & 1) t = mul(t, a, P);
    return t;
}
void Initialize(int n) {
    f[0] = 1;
    for (int i = 1; i < n; i++) f[i] = f[i - 1] * i % n;
    rf[n - 1] = pow(f[n - 1], n - 2, n);
    for (int i = n - 1; i; i--) rf[i - 1] = rf[i] * i % n;
}
LL C(int n, int m, int mod) {
    if (m > n || m < 0 || n < 0) return 0;
    return f[n] * rf[m] % mod * rf[n - m] % mod;
}
LL lucas(LL n, LL m, LL P) {
    if (n < m) return 0;
    if (!m || n == m) return 1;
    return C(n % P, m % P, P) * lucas(n / P, m / P, P) % P;
}
} // namespace Lucas
```

6.5 卡特兰数

```
/*
    卡特兰数&&组合数
    打表计算Fact
    n,m,MOD数量级1e9
*/
/*--------------------Make Fact Table----------------------*/
typedef long long LL;
const LL MOD = 1000000007;
LL res = 1;
int main() {
    freopen("1.txt", "w", stdout);
    for (int i = 1; i <= 1000000000; i++) {
        res = res * i % MOD;
        if (i % 500000 == 0) printf(",%d", res);
    }
    return 0;
}
/*-------------------Calculate Catalan---------------------*/
typedef long long LL;
const LL MOD = 1000000007, blk = 500000;
const LL lst[2001] = {……}; //填入打出的Fact表
int T, n, s;
char ss[1000010];
LL Fact(LL x) {
    if (x < 0) return 0;
    LL res = lst[x / blk];
    for (LL i = (x / blk) * blk + 1; i <= x; i++) res = (res * i) % MOD;
    return res;
}
LL power(LL a, LL b) {
    if (b == 0) return 1;
    if (b & 1) return (power(a, b - 1) * a) % MOD;
    LL t = power(a, b / 2);
    return (t * t) % MOD;
}
LL C(LL a, LL b) {
    if (a < 0 || b < 0 || a < b) return 0;
    return (Fact(a) * power(Fact(a - b), MOD - 2) % MOD *
            power(Fact(b), MOD - 2)) %
           MOD;
}
LL Catalan(int x) { return (C(2 * x, x) - C(2 * x, x - 1) + MOD) % MOD; }
```

6.6 错排公式

```
/*
    错位排列
    求长度为n序列m位恰好在原来位置的方案数
*/
typedef long long LL;
const int N = 1000010;
const LL mod = 1e9 + 7;
LL fac[N], inv[N], f[N];
int n, m, T;
int main() {
    fac[0] = inv[0] = inv[1] = 1;
    for (int i = 1; i < N; i++) fac[i] = fac[i - 1] * i % mod;
    for (int i = 2; i < N; i++) inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    for (int i = 2; i < N; i++) inv[i] = inv[i - 1] * inv[i] % mod;
    f[f[0] = f[2] = 1] = 0;
    for (int i = 3; i < N; i++) f[i] = (f[i - 1] + f[i - 2]) * (i - 1) % mod;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        printf("%lld\n",
               fac[n] * inv[m] % mod * inv[n - m] % mod * f[n - m] % mod);
    }
    return 0;
}
```

6.7 二项式展开

```
/*
    二项式展开
*/
/*
    Example1:
        m个位置每个位置可以填上[0,n-1]，求m个数和为k的方案数
    Solution:
        答案为生成函数(1+x^1+x^2+x^3+……)^m中x^k的系数
        1+x^1+x^2+x^3+……等比数列求和为(1-x^n)/(1-x)
        那么生成函数为(1-x^n)^m*(1-x)^(-m)
        (1-x^n)^m展开第i项为C(m,i)*(-1)^i*x^i
        (1-x)^(-m)展开第i项为C(m+i-1,i)*x^i
        我们找到对应幂次乘积为k的系数乘积即可
*/
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long long LL;
const int P = 998244353;
const int U = 200000;
int f[U + 3], rf[U + 3];
LL inv(LL a, LL m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
void init() {
    f[0] = 1;
    for (int i = 1; i <= U; i++) f[i] = (LL)f[i - 1] * i % P;
    rf[U] = inv(f[U], P);
    for (int i = U; i; i--) rf[i - 1] = (LL)rf[i] * i % P;
}
LL C(int n, int m) {
    if (m < 0 || m > n) return 0;
    return (LL)f[n] * rf[m] % P * rf[n - m] % P;
}
int T, n, m, k;
int main() {
    init();
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d", &n, &m, &k);
        int ans = 0;
        if (1ll * m * (n - 1) < k) {
            puts("0");
            continue;
        }
        for (int i = 0; i <= k / n; i++) {
            int j = k - n * i;
            int upd = 1LL * C(m, i) * C(m + j - 1, j) % P;
            if (i % 2)
                ans = (ans - upd + P) % P;
            else
                ans = (ans + upd) % P;
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example2:
        求出Ans[n]=(a[1]+a[2]+……+a[n])^k+(a[2]+……+a[n])^k+(a[3]+……+a[n])^k,
        并顺序输出Ans[1~n]。
    Solution:
        记s[n]= _{i=1}^{n}a[i],
        则Ans[n]= _{i=0}^{n}(s[n]-s[i])^k
        我们将式子展开，得到Ans[n]= _C(k,i)s[n]^(k-i)*( _{i=0}^{n-1}(-s[j])^i)
```

```
        我们预处理S[i][j]=s[i]^j 以及 R[i][j]= _{t=0}^{i}S[t][j]
        那么就可以O(nk)计算答案了。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010, M = 110;
typedef long long LL;
const LL P = 1000000007LL;
LL C[M][M], a[N], ans[N], s[N], S[N][M], T[N][M], R[N][M];
char ch[N];
int Cas, n, k;
void up(LL &x, LL y) {
    x += y;
    if (x >= P) x -= P;
    if (x < 0) x += P;
}
int main() {
    scanf("%d", &Cas);
    while (Cas--) {
        scanf("%d%d", &n, &k);
        for (int i = 0; i <= k; i++) C[i][0] = C[i][i] = 1LL;
        for (int i = 1; i <= k; i++)
            for (int j = 1; j <= i; j++)
                C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % P;
        scanf("%s", ch + 1);
        for (int i = 1; i <= n; i++) a[i] = ch[i] - '0';
        for (int i = 1; i <= n; i++) s[i] = (s[i - 1] + a[i]) % P;
        // S[i][j]=s[i]^j
        for (int i = 0; i <= n; i++) S[i][0] = 1;
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= k; j++) S[i][j] = S[i][j - 1] * s[i] % P;
        // R[i][j]= S[t][j](t<=i)
        for (int i = 0; i <= k; i++) R[0][i] = S[0][i];
        for (int i = 1; i <= n; i++)
            for (int j = 0; j <= k; j++) R[i][j] = (R[i - 1][j] + S[i][j]) % P;
        for (int i = 1; i <= n; i++) {
            ans[i] = 0;
            for (int p = 0; p <= k; p++) {
                if (p % 2 == 0)
                    up(ans[i], C[k][p] * S[i][k - p] % P * R[i - 1][p] % P);
                else
                    up(ans[i], -C[k][p] * S[i][k - p] % P * R[i - 1][p] % P);
            }
        }
        for (int i = 1; i <= n; i++)
```

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

6.8 牛顿迭代法

```
# 牛顿迭代法求开方下取整
T=int(raw_input())
for i in range(0,T):
  n=int(raw_input())
  if n<2:
    print n
    continue
  m=2
  tmpn,len=n,0
  while tmpn>0:
    tmpn/=10
    len+=1
  base,digit,cur=300,len/m,len%m
  while(cur+m<= base)and(digit > 0):
    cur+=m
    digit-=1
  div=10**(digit*m)
  tmpn=n/div
  x =int(float(tmpn)**(1.0/m))
  x*=(10**digit)
  while True:
    x0=x
    x=x+x*(n-x**m)/(n*m)
    if x==x0:break
  while (x+1)**m<=n:
    x=x+1
  print x
```

6.9 GCD

```
// GCD(Fib(n),Fib(m))=GCD(Fib(n-m),Fib(m))
// GCD(a^m-b^m,a^n-b^n) = a ^ GCD(m,n) - b ^ GCD(m,n) [GCD(a,b)==1]
// 区间GCD
/*
    Problem:
        求区间GCD和与其区间GCD值相同的区间数量
```

```
    Solution:
        固定右端点，求出区间GCD值的左端点分界线，每个右端点只有logn个分界线，
        查询对应分界线和GCD值即可，用map保存GCD值的数量
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <map>
using namespace std;
map<int, long long> M;
const int N = 100005;
int Gcd, n, l[N], v[N], len[N], T, j, q, L, R, Cas, a[N];
struct data {
    int l, v;
} p[N][50];
int gcd(int x, int y) { return __gcd(x, y); }
int main() {
    scanf("%d", &T);
    while (T--) {
        printf("Case #%d:\n", ++Cas);
        M.clear();
        scanf("%d", &n);
        memset(len, 0, sizeof(len));
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        for (int i = 1; i <= n; i++)
            for (v[i] = a[i], j = l[i] = i; j; j = l[j] - 1) {
                v[j] = gcd(v[j], a[i]);
                while (l[j] > 1 && gcd(a[i], v[l[j] - 1]) == gcd(a[i], v[j]))
                    l[j] = l[l[j] - 1];
                p[i][len[i]].l = l[j];
                p[i][len[i]++].v = v[j];
                M[v[j]] += (j - l[j] + 1);
            }
        scanf("%d", &q);
        while (q--) {
            scanf("%d%d", &L, &R);
            for (int i = 0; i < len[R]; i++) {
                if (L >= p[R][i].l) {
                    Gcd = p[R][i].v;
                    break;
                }
            }
            printf("%d %lld\n", Gcd, M[Gcd]);
        }
    }
    return 0;
```

```
}
/*
    Problem:
        给定一个数列，多次查询区间[l,r]，询问其中不同的子区间gcd的数量
    Solution:
        将所有的查询按右区间从小到大排序，在树状数组上顺序更新每个右端点对应的GCD分界线，
        注意当一个 GCD值第二次出现的时候，需要将上一次出现位置消除，更新现在所在的位置，
        对树状数组区间求和即可得到答案
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 1000010;
int n, q, v[N], l[N], j, a[N], ans[N], pre[N], c[N];
struct Data {
    int l, r, id;
} p[N];
bool cmp(Data a, Data b) { return a.r < b.r; }
void upd(int x, int y) {
    for (; x <= n; x += x & -x) c[x] += y;
}
int query(int x) {
    int s = 0;
    while (x) {
        s += c[x];
        x -= x & -x;
    }
    return s;
}
int main() {
    while (~scanf("%d%d", &n, &q)) {
        memset(pre, -1, sizeof(pre));
        memset(c, 0, sizeof(c));
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        for (int i = 1; i <= q; i++) {
            scanf("%d%d", &p[i].l, &p[i].r);
            p[i].id = i;
        }
        sort(p + 1, p + q + 1, cmp);
        int pos = 1;
        for (int i = 1; i <= n; i++) {
            for (v[i] = a[i], j = l[i] = i; j; j = l[j] - 1) {
                v[j] = __gcd(v[j], a[i]);
                while (l[j] > 1 && __gcd(a[i], v[l[j] - 1]) == __gcd(a[i], v[j]))
                    l[j] = l[l[j] - 1];
                if (~pre[v[j]]) upd(pre[v[j]], -1);
                upd(j, 1);
```

```
            pre[v[j]] = j;
        }
        while (p[pos].r == i) {
            ans[p[pos].id] = query(p[pos].r) - query(p[pos].l - 1);
            pos++;
        }
    }
    for (int i = 1; i <= q; i++) printf("%d\n", ans[i]);
  }
  return 0;
}
```

6.10 扩展欧几里得算法

```
/*
   扩展欧几里得定理
*/
typedef long long ll;
ll exgcd(ll a, ll b, ll& x, ll& y) {
  if (!b) return x = 1, y = 0, a;
  ll d = exgcd(b, a % b, x, y), t = x;
  return x = y, y = t - a / b * y, d;
}
ll A, B, C, k;
int main() {
  while (scanf("%lld%lld%lld%lld", &A, &B, &C, &k), A + B + C + k) {
    ll a = C, b = B - A, n = 1LL << k, x, y, d = exgcd(a, n, x, y);
    if (b % d)
      puts("FOREVER");
    else {
      x = (x * (b / d)) % n; //方程ax=b(mod n)的最小解
      x = (x % (n / d) + n / d) % (n / d); //方程ax=b(mod n)的最小整数解
      printf("%I64d\n", x);
    }
  }
  return 0;
}
```

6.11 类欧几里得算法

```
/*
`  类欧几里得算法
   Ti=floor((a*i+b)/c)
```

```
    f(a,b,c,n)=sum(Ti)
    g(a,b,c,n)=sum(i*Ti)
    h(a,b,c,n)=sum(Ti^2)
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
using namespace std;
const int mo = 1e9 + 7, inv2 = 500000004, inv6 = 166666668;
typedef long long LL;
int a, b, c, l, r;
struct data {
    int f, g, h;
};
data calc(int a, int b, int c, LL n) {
    data tmp;
    if (!a) {
        tmp.f = tmp.g = tmp.h = 0;
        return tmp;
    }
    if (a >= c || b >= c) {
        tmp = calc(a % c, b % c, c, n);
        n %= mo;
        tmp.h = (tmp.h +
                n * (n + 1) % mo * (2 * n + 1) % mo * inv6 % mo * (a / c) %
                    mo * (a / c) % mo +
                (n + 1) * (b / c) % mo * (b / c) % mo +
                (LL)2 * (a / c) * tmp.g % mo + (LL)2 * (b / c) * tmp.f % mo +
                n * (n + 1) % mo * (a / c) % mo * (b / c)) %
               mo;
        tmp.f =
            (tmp.f + n * (n + 1) / 2 % mo * (a / c) + (n + 1) * (b / c)) % mo;
        tmp.g =
            (tmp.g + n * (n + 1) % mo * (2 * n + 1) % mo * inv6 % mo * (a / c) +
             n * (n + 1) / 2 % mo * (b / c)) %
            mo;
        return tmp;
    }
    LL m = ((LL)a * n + b) / c;
    data nxt = calc(c, c - b - 1, a, m - 1);
    n %= mo;
    m %= mo;
    tmp.f = ((n * m - nxt.f) % mo + mo) % mo;
    tmp.g = (LL)((n * (n + 1) % mo * m - nxt.f - nxt.h) % mo + mo) * inv2 % mo;
    tmp.h =
```

```
        ((m * (m + 1) % mo * n - (LL)2 * (nxt.g + nxt.f) % mo - tmp.f) % mo +
         mo) %
        mo;
    return tmp;
}
// 求l到r的g函数
int main() {
    scanf("%d%d%d%d%d", &a, &c, &b, &l, &r);
    printf("%d\n", (calc(a, b, c, r).g - calc(a, b, c, l - 1).g + mo) % mo);
    return 0;
}
/*
    题目大意：求sum(-1^floor(sqrt(d*r*d))) [d from 1 to n]
    题解，我们设x=sqrt(r)，则题目为求sum(-1^floor(dx)) [d from 1 to n],
    -1^floor(dx)=1-2(floor(dx)%2)=1-2(floor(dx)-floor(dx/2)*2)
    =1-2*floor(dx)+4*floor(dx/2)
    对floor(dx)和floor(dx/2)类欧求解
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
int T, n, r;
double t;
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
int likegcd(int n, int a, int b, int c) {
    if (!n) return 0;
    int f = gcd(gcd(a, b), c);
    a /= f, b /= f, c /= f;
    f = (t * b + c) / a;
    int res = 1LL * n * (n + 1) * f >> 1;
    c -= f * a;
    f = (b * t + c) * n / a;
    return res + n * f - likegcd(f, b * b * r - c * c, a * b, -a * c);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &r);
        t = sqrt(r);
        if ((int)t == t) /*完全平方数*/
            printf("%d\n", (r & 1) ? ((n & 1) ? -1 : 0) : n);
        else
            printf("%d\n",
                   n + 4 * likegcd(n, 2, 1, 0) - 2 * likegcd(n, 1, 1, 0));
    }
```

```
    return 0;
}
```

6.12 中国剩余定理

```
/*
    中国剩余定理
    n个同余方程，第i个为x b[i](mod a[i])，要求a[i]两两互质
*/
LL CRT(LL* a, LL* b, int n) {
    LL ans, P = 1;
    for (int i = 0; i < n; i++) P *= a[i];
    for (int i = 0; i < n; i++)
        ans =
            (ans + (P / a[i]) * pow(P / a[i], a[i] - 2, a[i]) % P * b[i] % P) %
            P;
    while (ans < 0) ans += P;
    return ans;
}
```

6.13 扩展 Lucas 定理

```
/*
    CRT+Lucas
    计算模为多个素数乘积的组合数
    模以m个素数形式给出
*/
namespace CRT_Lucas {
LL f[N], rf[N];
LL mul(LL x, LL y, LL P) {
    return (x * y - (LL)(x / (long double)P * y + 1e-3) * P + P) % P;
}
LL pow(LL a, LL b, LL P) {
    LL t = 1;
    for (; b; b >>= 1, a = mul(a, a, P))
        if (b & 1) t = mul(t, a, P);
    return t;
}
void Initialize(int n) {
    f[0] = 1;
    for (int i = 1; i < n; i++) f[i] = f[i - 1] * i % n;
    rf[n - 1] = pow(f[n - 1], n - 2, n);
    for (int i = n - 1; i; i--) rf[i - 1] = rf[i] * i % n;
```

```
}
LL C(int n, int m, int mod) {
    if (m > n || m < 0 || n < 0) return 0;
    return f[n] * rf[m] % mod * rf[n - m] % mod;
}
LL lucas(LL n, LL m, LL P) {
    if (n < m) return 0;
    if (!m || n == m) return 1;
    return C(n % P, m % P, P) * lucas(n / P, m / P, P) % P;
}
LL exgcd(LL a, LL b, LL& x, LL& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= x * (a / b);
    return d;
}
LL CRT(LL* a, LL* b, int n) {
    LL P = 1, d, y, x = 0;
    for (int i = 0; i < n; i++) P *= a[i];
    for (int i = 0; i < n; i++) {
        LL w = P / a[i];
        exgcd(a[i], w, d, y);
        y = (y % P + P) % P;
        x = (x + mul(mul(y, w, P), b[i], P));
    }
    return (x + P) % P;
}
LL a[K], p[K];
LL Cal(LL n, LL m, int k) {
    for (int i = 0; i < k; i++) {
        scanf("%lld", &p[i]);
        Initialize(p[i]);
        a[i] = lucas(n, m, p[i]);
    }
    return CRT(p, a, k);
}
} // namespace CRT_Lucas
```

6.14 BSGS

```
/*
```

```
    BSGS
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <map>
#include <tr1/unordered_map>
using namespace std::tr1;
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
int phi(int n) {
    int t = 1, i;
    for (i = 2; i * i <= n; i++)
        if (n % i == 0)
            for (n /= i, t *= i - 1; n % i == 0; n /= i, t *= i)
                ;
    if (n > 1) t *= n - 1;
    return t;
}
int pow(ll a, int b, int m) {
    ll t = 1;
    for (; b; b >>= 1, a = a * a % m)
        if (b & 1) t = t * a % m;
    return t;
}
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
int exgcd(int a, int b, int& x, int& y) {
    if (!b) return x = 1, y = 0, a;
    int d = exgcd(b, a % b, x, y), t = x;
    return x = y, y = t - a / b * y, d;
}
int bsgs(int a, int r, int m) {
    if (r >= m) return -1;
    int i, g, x, c = 0, at = int(2 + sqrt(m));
    for (i = 0, x = 1 % m; i < 50; i++, x = ll(x) * a % m)
        if (x == r) return i;
    for (g = x = 1; __gcd(int(ll(x) * a % m), m) != g; c++)
        g = __gcd(x = ll(x) * a % m, m);
    if (r % g) return -1;
    if (x == r) return c;
    unordered_map<int, int> u;
    g = phi(m / g), u[x] = 0;
    g = pow(a, g - at % g, m);
    for (i = 1; i < at; i++) {
        u.insert(P(x = ll(x) * a % m, i));
```

```
        if (x == r) return c + i;
    }
    for (i = 1; i < at; i++) {
        unordered_map<int, int>::iterator t = u.find(r = ll(r) * g % m);
        if (t != u.end()) return c + i * at + t->second;
    }
    return -1;
}
// 计算 Y^Z Mod P 的值
void solve1(ll y, int z, int p) { printf("%d\n", pow(y, z, p)); }
// 计算满足 xy  Z ( mod P ) 的最小非负整数
void solve2(int y, int z, int p) {
    p = -p;
    int t = gcd(y, p);
    if (z % t) {
        puts("Orz, I cannot find x!");
        return;
    }
    y /= t;
    z /= t;
    p /= t;
    int a, b;
    exgcd(y, p, a, b);
    a = (ll)a * z % p;
    while (a < 0) a += p;
    printf("%d\n", a);
}
// 计算满足 Y^x  Z ( mod P) 的最小非负整数
void solve3(int y, int z, int p) {
    y %= p;
    z %= p;
    int t = bsgs(y, z, p);
    if (t == -1) {
        puts("Orz, I cannot find x!");
        return;
    } else
        printf("%d\n", t);
}
int main() {
    int T, k, y, z, p;
    while (~scanf("%d%d", &T, &k)) {
        while (T--) {
            scanf("%d%d%d", &y, &z, &p);
            if (k == 1) solve1(y, z, p);
            if (k == 2) solve2(y, z, p);
            if (k == 3) solve3(y, z, p);
```

```
        }
    }
    return 0;
}
/*
    Problem:
        x_i=ax_{i-1}+b
        给定一个数字，问其最早在x序列的哪一位出现
    Solution:
        Ax+B=a(a(a……(ax+b)……+b)+b)+b
        我们记录k阶式子为Ax+B
        那么答案可以被分为y=A(A(A……(Ax+B)……+B)+B)+B
        和ans=a(a(a……(ay+b)……+b)+b)+b
        保存一个逆运算，BSGS即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> PI;
const int K = 1000000, mod = 1 << 22;
ll n;
int head[mod], cnt;
int x, a, b, P, A[K + 1], B[K + 1];
struct data {
    int x, nxt, s;
} g[mod];
inline int Hash(int x) { return (x + mod) & (mod - 1); }
void insert(int x, int y) {
    int key = Hash(x);
    for (int i = head[key]; i != -1; i = g[i].nxt) {
        if (g[i].x == x) return;
    }
    g[cnt].s = y;
    g[cnt].x = x;
    g[cnt].nxt = head[key];
    head[key] = cnt++;
}
int search(int x) {
    int key = Hash(x);
    for (int i = head[key]; i != -1; i = g[i].nxt) {
        if (g[i].x == x) return g[i].s;
    }
    return -1;
}
inline ll ask(int x) {
    for (int i = 0; i * K <= n && i * K <= P; i++) {
```

```
        int t = 1LL * (x - B[i] + P) * A[i] % P;
        auto it = search(t);
        if (it != -1) return (i * K + it) >= n ? -1 : i * K + it;
    }
    return -1;
}
ll pwmod(ll a, ll b, ll P) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return res;
}
inline void init() {
    cnt = 0;
    memset(head, -1, sizeof(head));
    A[0] = 1, B[0] = 0;
    insert(x, 0);
    for (int i = 1; i <= K; i++) {
        x = (1ll * x * a + b) % P;
        insert(x, i);
        A[i] = 1ll * A[i - 1] * a % P;
        B[i] = (1ll * B[i - 1] * a + b) % P;
    }
    ll inv = pwmod(A[K], P - 2, P);
    for (int i = 1; 1ll * i * K <= P; i++) {
        A[i] = 1ll * A[i - 1] * inv % P;
        B[i] = (1ll * B[i - 1] * A[K] + B[K]) % P;
    }
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%lld%d%d%d%d", &n, &x, &a, &b, &P);
        init();
        int q;
        scanf("%d", &q);
        while (q--) {
            scanf("%d", &x);
            printf("%lld\n", ask(x));
        }
    }
    return 0;
```

```
}
```

6.15 BSGS 求指标

```
/*
    BSGS-NT
    求给定模数的原根指标
    BSGS的步长=查询次数*P/步长时为佳
    考虑常数微下调步长
*/
const int P = 1e9 + 7;
namespace NT {
unordered_map<int, int> T;
typedef pair<int, int> PI;
const int K = 300000, G = 5; // 1e9+7的原根为5
int ind[1010], base, i;
inline int ask(int x) {
    if (x == 1) return 0;
    int t = pow(x, P - 2, P);
    for (int i = K;; i += K) {
        t = 1LL * t * base % P;
        unordered_map<int, int>::iterator it = T.find(t);
        if (it != T.end()) return i - it->second;
    }
}
void init() {
    for (base = 1, i = 0; i < K; i++) {
        T.insert(PI(base, i));
        base = 1LL * base * G % P;
    }
    for (i = 1; i <= 1000; i++) ind[i] = ask(i);
}
} // namespace NT
```

6.16 二次剩余

```
/*
    二次剩余
    sqrt(n)%P
    Example
        (x+y)%P=b
        (x*y)%P=c
        求x和y
```

```
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const LL P = 1e9 + 7;
const LL inv2 = (P + 1) / 2;
int T;
LL b, c;
LL mul(LL x, LL y, LL P) {
    return (x * y - (LL)(x / (long double)P * y + 1e-3) * P + P) % P;
}
LL pow_mod(LL a, LL b, LL P) {
    LL t = 1;
    for (; b; b >>= 1, a = mul(a, a, P))
        if (b & 1) t = mul(t, a, P);
    return t;
}
// 求sqrt(n)%P
LL ToneLLi_Shanks(LL n, LL p) {
    if (p == 2) return (n & 1) ? 1 : -1;
    if (pow_mod(n, p >> 1, p) + 1 == p) return -1;
    if (p & 2) return pow_mod(n, p + 1 >> 2, p);
    int s = __builtin_ctzll(p ^ 1);
    LL q = p >> s, z = 2;
    for (; pow_mod(z, p >> 1, p) == 1; ++z)
        ;
    LL c = pow_mod(z, q, p);
    LL r = pow_mod(n, q + 1 >> 1, p);
    LL t = pow_mod(n, q, p), tmp;
    for (int m = s, i; t != 1;) {
        for (i = 0, tmp = t; tmp != 1; ++i) tmp = tmp * tmp % p;
        for (; i < --m;) c = c * c % p;
        r = r * c % p;
        c = c * c % p;
        t = t * c % p;
    }
    return r;
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%lld%lld", &b, &c);
        LL x = ToneLLi_Shanks(((b * b - 4 * c) % P + P) % P, P);
        if (x == -1)
            puts("-1 -1");
        else {
```

```
            LL u1 = (b - x + P) * inv2 % P;
            LL u2 = (b + x) * inv2 % P;
            if (u1 > u2) swap(u1, u2);
            printf("%lld %lld\n", u1, u2);
        }
    }
    return 0;
}
```

6.17 Pell 方程

```
# Pell方程
# T组数据，给出n，求x*x-n*y*y=1的最小正整数解集(x,y)
T=int(raw_input())
while T>0:
    n=int(raw_input())
    j=1
    while j*j<n:
        j+=1
    if j*j==n:
        print j,1
    if j*j>n:
        p=[0 for i in range(0,1001)]
        q=[0 for i in range(0,1001)]
        a=[0 for i in range(0,1001)]
        g=[0 for i in range(0,1001)]
        h=[0 for i in range(0,1001)]
        p[1]=q[0]=h[1]=1
        p[0]=q[1]=g[1]=0
        a[2]=j-1
        i=2
        while 1:
            g[i]=-g[i-1]+a[i]*h[i-1]
            h[i]=(n-g[i]*g[i])/h[i-1]
            a[i+1]=(g[i]+a[2])/h[i]
            p[i]=a[i]*p[i-1]+p[i-2]
            q[i]=a[i]*q[i-1]+q[i-2]
            if(p[i]*p[i]-n*q[i]*q[i]==1):
                print p[i],q[i]
                break
            i+=1
    T-=1
```

6.18 DIJKSTRA 求解丢番图方程

```
/*
    利用最短路求解丢番图方程
    Check: 判断num是否能被表示为a[1]*x[1]+a[2]*x[2]+……+a[n]*x[n], x为非负整数
    Cal: 计算大于等于k的最小的能被表示的数
    Query: 求解x以内解的数量
*/
const int N = [the value of a[0]];
namespace DIJKSTRA {
typedef long long LL;
const LL INF = 0x3f3f3f3f3f3f3f3f;
typedef pair<LL, int> P;
priority_queue<P, vector<P>, greater<P> > Q;
int a[N], n, m;
LL d[N];
void Initialize() {
    int x, i;
    sort(a, a + n);
    m = a[0];
    d[0] = 0;
    for (i = 1; i < m; i++) d[i] = INF;
    Q.push(P(0, 0));
    while (!Q.empty()) {
        P t = Q.top();
        Q.pop();
        if (d[t.second] < t.first) continue;
        for (x = t.second, i = 1; i < n; i++) {
            if (d[x] + a[i] < d[(x + a[i]) % m])
                Q.push(P(d[(x + a[i]) % m] = d[x] + a[i], (x + a[i]) % m));
        }
    }
}
bool Check(LL x) { return d[x % m] <= x; }
int Cal(LL k) {
    for (int i = 0; i < m; i++)
        if (d[(k + i) % m] <= k + i) return k + i;
}
LL Query(LL x) {
    LL res = 0;
    for (int i = 0; i < m; i++)
        if (d[i] <= x) res += (x - d[i]) / m + 1;
    return res;
}
} // namespace DIJKSTRA
/*
```

```
    Test
    求解L到R之间的非负整数解集个数
    Input : 2 5 10 3 5
    Output : 5
*/
long long L, R;
int main() {
    using namespace DIJKSTRA;
    scanf("%d%lld%lld", &n, &L, &R);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    Initialize();
    printf("%lld\n", Query(R) - Query(L - 1));
    return 0;
}
```

6.19 模意义高斯消元

```
/*
    模意义下的高斯消元
*/
namespace Gauss {
const int N = 110, MOD = 2, INF = 1e9;
int a[N][N], ans[N];
bool isFreeX[N];
int inv(int a, int m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
int getAns(int n, int m, int r) {
    int res = 0;
    for (int i = r - 1; ~i; i--) {
        for (int j = 0; j < m; j++) {
            if (!a[i][j]) continue;
            ans[j] = a[i][m];
            for (int k = j + 1; k < m; k++) {
                ans[j] -= a[i][k] * ans[k];
                ans[j] %= MOD;
                if (ans[j] < 0) ans[j] += MOD;
            }
            ans[j] = ans[j] * inv(a[i][j], MOD) % MOD;
            break;
        }
    }
    for (int i = 0; i < m; i++) res += ans[i];
    return res;
}
/*
    返回值为方程的阶，即非自由变元数量
```

```
    无解返回-1
*/
int gauss(int n, int m) {
    for (int i = 0; i < m; i++) isFreeX[i] = 0;
    int r = 0, c = 0;
    for (; r < n && c < m; r++, c++) {
        int maxR = r;
        for (int i = r + 1; i < n; i++)
            if (abs(a[i][c]) > abs(a[maxR][c])) maxR = i;
        if (maxR != r) swap(a[maxR], a[r]);
        if (!a[r][c]) {
            r--;
            isFreeX[c] = 1;
            continue;
        }
        for (int i = r + 1; i < n; i++) {
            if (a[i][c]) {
                int delta = a[i][c] * inv(a[r][c], MOD);
                for (int j = c; j <= m; j++) {
                    a[i][j] -= delta * a[r][j];
                    a[i][j] %= MOD;
                    if (a[i][j] < 0) a[i][j] += MOD;
                }
            }
        }
    }
    for (int i = r; i < n; i++)
        if (a[i][m]) return -1;
    return r;
}
// 模2枚举自由变元
int getMinAns(int n, int m, int r) {
    int res = INF, freeX = m - r;
    for (int s = 0; s < 1 << freeX; s++) {
        if (__builtin_popcount(s) >= res) continue;
        int cnt = 0;
        for (int j = 0; j < m; j++) {
            if (isFreeX[j]) {
                ans[j] = s >> cnt & 1;
                ++cnt;
            }
        }
        res = min(res, getAns(n, m, r));
    }
    return res;
}
```

```
} // namespace Gauss
/*
    Test
    给出灯的相连关系，相邻的灯开关会相互影响
    求把灯全部打开的最少开关数
*/
int n, x, y;
int main() {
    while (~scanf("%d", &n), n) {
        using namespace Gauss;
        memset(a, 0, sizeof(a));
        for (int i = 1; i < n; i++) {
            scanf("%d%d", &x, &y);
            a[x - 1][y - 1] = 1;
            a[y - 1][x - 1] = 1;
        }
        for (int i = 0; i < n; i++) a[i][i] = a[i][n] = 1;
        int r = gauss(n, n);
        printf("%d\n", getMinAns(n, n, r));
    }
    return 0;
}
```

6.20 异或线性基

```
/*
    异或线性基
*/
typedef long long LL;
struct L_B {
    LL d[61], p[61];
    int cnt, zero;
    L_B() {
        memset(d, 0, sizeof(d));
        memset(p, 0, sizeof(p));
        cnt = zero = 0;
    }
    void ins(LL val) {
        for (int i = 60; i >= 0; i--)
            if (val & (1LL << i)) {
                if (!d[i]) {
                    d[i] = val;
                    cnt++;
                    return;
                }
```

```
            val ^= d[i];
        }
    zero = 1;
}
LL query_max(LL x) {
    LL ret = x;
    for (int i = 60; i >= 0; i--)
        if ((ret ^ d[i]) > ret) ret ^= d[i];
    return ret;
}
LL query_min() {
    for (int i = 0; i <= 60; i++)
        if (d[i]) return d[i];
    return 0;
}
void rebuild() {
    cnt = 0;
    for (int i = 60; i >= 0; i--)
        for (int j = i - 1; j >= 0; j--)
            if (d[i] & (1LL << j)) d[i] ^= d[j];
    for (int i = 0; i <= 60; i++)
        if (d[i]) p[cnt++] = d[i];
}
LL kth(LL k) { //求第k小之前需要rebuild
    int ret = 0;
    if (k >= (1LL << cnt)) return -1;
    for (int i = 60; i >= 0; i--)
        if (k & (1LL << i)) ret ^= p[i];
    return ret;
}
int rnk(LL x) {
    int rnk = 0, bit = 0;
    for (int i = 0; i <= 60; i++)
        if (d[i]) {
            if (x >> i & 1) rnk += 1 << bit;
            bit++;
        }
    return rnk;
}
};
L_B merge(const L_B &n1, const L_B &n2) {
    L_B ret = n1;
    for (int i = 60; i >= 0; i--)
        if (n2.d[i]) ret.ins(n1.d[i]);
    return ret;
}
```

```
/*
    Example: 求第k小的子集异或值
*/
int main() {
    int T, n, q;
    LL x;
    scanf("%d", &T);
    for (int cas = 1; cas <= T; cas++) {
        L_B b;
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%lld", &x);
            b.ins(x);
        }
        printf("Case #%d:\n", cas);
        b.rebuild();
        scanf("%d", &q);
        while (q--) {
            scanf("%lld", &x);
            if (b.zero)
                x--; //线性基包含线性相关向量组，能异或得到0，向量个数为2^cnt
            printf("%lld\n", b.kth(x));
        }
    }
    return 0;
}
/*
    Example: 给出一张无向边权图，有重边和自环，求一条从1到n的路径，使得异或和最大
    Solution: 等价于任意找一条1到n的路径，然后用图中的环去异或这个值，找出最大值，
    因为可以到达环并原路返回消掉。我们将图中所有环的异或值求出来，加入异或线性基，
    用1到n的路径值去线性基中求异或最大值即可
*/
L_B b;
bool vis[N];
void dfs(int x, int fx) {
    vis[x] = 1;
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i];
        LL z = w[x][i];
        if (y != fx) {
            if (!vis[y]) {
                d[y] = d[x] ^ z;
                dfs(y, x);
            } else
                b.ins(d[x] ^ d[y] ^ z);
        }
```

```
    }
}
int n, m;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int x, y;
        LL z;
        scanf("%d%d%lld", &x, &y, &z);
        v[x].push_back(y);
        w[x].push_back(z);
        v[y].push_back(x);
        w[y].push_back(z);
    }
    dfs(1, -1);
    printf("%lld\n", b.query_max(d[n]));
    return 0;
}
/*
    Example: 给定n个数，将其所有子集（可以为空）的异或值从小到大排序得到序列B，
    请问某数Q在B中第一次出现的下标是多少？保证Q在B中出现。
    Solution: 通过二进制拆分可以查询非重情况下的rank，
    而子集xor所有值中每种数出现次数都是一样的，可以直接用快速幂处理
*/
int n;
LL x;
LL pow(int a, int b) {
    int t = 1;
    while (b) {
        if (b & 1) t = t * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return t;
}
int main() {
    L_B b;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &x);
        b.ins(x);
    }
    scanf("%lld", &x);
    printf("%d\n", (b.rnk(x) % P * pow(2, n - b.cnt) % P + 1) % P);
    return 0;
}
```

6.21 欧拉函数

```
/*
    欧拉函数
    对正整数n，欧拉函数是少于或等于n的数中与n互质的数的数目。
    例如Euler(8)=4，因为1,3,5,7均和8互质。
    通式：
        对于一个正整数N的素数幂分解
            N=(P1^q1)*(P2^q2)*...*(Pn^qn).
              phi(1)=1.
              phi(N)=N*(1-1/P1)*(1-1/P2)*...*(1-1/Pn).
    定理：
        1.欧拉函数是非完全积性函数：phi(m*n)=phi(n)*phi(m)
        当且仅当GCD(n,m)=1.
        2.一个数的所有质因子之和是Euler(n)*n/2.
        3.若n是素数p的k次幂，phi(n)=p^k-p^(k-1)=(p-1)p^(k-1)，
        因为除了p的倍数外，其他数都跟n互质.
    特殊性质：
        1.当n为奇数时，phi(2n)=phi(n).
        2.对于质数p，phi(p)=p-1
        3.除了N=2,phi(N)都是偶数.
    指数循环节：
        A^x=A^(x%phi(C)+phi(C))(mod C)(x>=phi(C))
*/
// 筛法求欧拉函数
void Get_Euler() {
    for (int i = 1; i <= n; i++) phi[i] = i;
    for (int i = 2; i <= n; i++)
        if (phi[i] == i)
            for (int j = i; j <= n; j += i) phi[j] = phi[j] / i * (i - 1);
}
int Euler(int n) {
    int t = 1, i;
    if (!(n & 1))
        for (n >>= 1; !(n & 1); n >>= 1, t <<= 1)
            ;
    for (i = 3; i * i <= n; i += 2)
        if (n % i == 0)
            for (n /= i, t *= i - 1; n % i == 0; n /= i, t *= i)
                ;
    if (n > 1) t *= n - 1;
    return t;
}
/*
    求出 gcd(i,N)(1<=i<=N)
     _{i=1}^{N}gcd(i,N)
```

```
    = _{i=1}^{N} _{d|gcd(i,N)}\phi(d)
    = \phi(d) _{1=<i<=N&&d|i&&d|N}1
    = _{d|N}\phi(d)\floorl\frac{i}{d}\floorr
*/
int main() {
    int n;
    long long ans = 0;
    while (~scanf("%d", &n)) {
        for (int i = 1; i * i <= n; i++) {
            if (n % i == 0) {
                ans += 1LL * i * Euler(n / i);
                if (i * i < n) ans += 1LL * (n / i) * Euler(i);
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
/*
    f(0) = 1
    f(n) = (n%10)^f(n/10)
    求f(n) % m
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int phi(int n) {
    int ans = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
int pow(ll x, ll y, ll p) {
    ll cnt = 1, f1 = 0, f2 = 0;
    for (; y; y >>= 1) {
        if (y & 1) f1 |= (cnt * x >= p || f2), cnt = cnt * x % p;
        f2 |= (x * x >= p);
        x = x * x % p;
    }
    return cnt + f1 * p;
}
```

```
int T, n, m;
int Cal(int x, int p) {
    if (x == 0) return 1;
    return pow(x % 10, Cal(x / 10, phi(p)), p);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        printf("%d\n", Cal(n, m) % m);
    }
    return 0;
}
/*
    Problem:
        求a1^a2^a3^……^an%m（^表示幂）
    Solution:
        b>=phi(p)时有a^b%p=a^(b%phi(p)+phi(p1))%p
        递归处理，每次取phi(p)，试乘判断是否超过phi(p)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int read() {
    int x = 0, z = 1;
    char c = getchar();
    for (; c < '0' || c > '9'; c = getchar()) {
        if (c == '#') exit(0);
        z = c == '-' ? -1 : 1;
    }
    for (; c >= '0' && c <= '9'; c = getchar())
        x = (x << 1) + (x << 3) + (c ^ 48);
    return x * z;
}
int phi(int n) {
    int ans = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
int pow(ll x, ll y, ll p) {
```

```
    ll cnt = 1, f1 = 0, f2 = 0;
    for (; y; y >>= 1) {
        if (y & 1) f1 |= (cnt * x >= p || f2), cnt = cnt * x % p;
        f2 |= (x * x >= p);
        x = x * x % p;
    }
    return cnt + f1 * p;
}
const int N = 15;
int n, m, a[N];
int Cal(int x, int p) {
    if (x == n || p == 2) return pow(a[x], 1, p);
    return pow(a[x], Cal(x + 1, phi(p)), p);
}
int main() {
    for (int T = 1;; T++) {
        m = read(), n = read();
        for (int i = 1; i <= n; i++) a[i] = read();
        printf("Case #%d: %d\n", T, Cal(1, m) % m);
    }
    return 0;
}
```

6.22 反欧拉函数

```
/*
    反欧拉函数
    给定n(n 10^12)求最小的满足 (x)=n的x
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
typedef long long LL;
using namespace std;
int const MxCnt = 1e4;
LL Y, Ans, Cnt, Pri[MxCnt + 10];
bool CPri(LL x) {
    int Mx = sqrt(x);
    for (int i = 2; i <= Mx; i++)
        if (x % i == 0) return 0;
    return 1;
}
void Prep(LL x) {
    Cnt = 0;
```

```
    int Mx = sqrt(x);
    for (int i = 1; i <= Mx; i++)
        if (x % i == 0) {
            if (CPri(i + 1)) Pri[++Cnt] = i + 1;
            if (CPri(x / i + 1)) Pri[++Cnt] = x / i + 1;
        }
}
bool Cmp(LL X, LL Y) { return X > Y; }
void Dfs(int Pos, LL Y, LL X) {
    if (Y == 1) Ans = min(Ans, X);
    if ((Pos > Cnt) || (Y == 1) || (X > Ans)) return;
    for (LL i = Pri[Pos] - 1; Y % i == 0; i *= Pri[Pos])
        Dfs(Pos + 1, Y / i, X / (Pri[Pos] - 1) * Pri[Pos]);
    Dfs(Pos + 1, Y, X);
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%lld", &Y);
        Prep(Y);
        sort(Pri + 1, Pri + Cnt + 1, Cmp);
        Ans = (Y == 1) ? Y : Y * 10;
        Dfs(1, Y, Y);
        printf("%lld\n", Ans);
    }
    return 0;
}
/*
    给定n(n 10^10)求所有满足 (x)=n的x
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long ll;
const int N = 1000000, M = 5000;
int T, lim, m, d, cnt, i, j, p[N / 10], tot, small[N], big[N], g[M][700];
bool v[N];
ll n, a[M], b[M], q[N];
bool check(ll n) {
    if (n < N) return !v[n];
    for (int i = 2; 1LL * i * i <= n; i++)
        if (n % i == 0) return 0;
    return 1;
}
void dfs(int x, ll S, ll p) {
```

```
    if (p == 1) {
        q[cnt++] = S;
        return;
    }
    x = g[p <= lim ? small[p] : big[n / p]][x];
    if (x == m) return;
    dfs(x + 1, S, p);
    for (dfs(x + 1, S *= a[x], p /= a[x] - 1); p % a[x] == 0;
         dfs(x + 1, S *= a[x], p /= a[x]))
        ;
}
int main() {
    for (i = 2; i < N; i++) {
        if (!v[i]) p[tot++] = i;
        for (j = 0; j < tot && i * p[j] < N; j++) {
            v[i * p[j]] = 1;
            if (i % p[j] == 0) break;
        }
    }
    scanf("%d", &T);
    while (T--) {
        scanf("%lld", &n);
        if (n == 1) {
            puts("2\n1 2");
            continue;
        }
        for (lim = 1; 1LL * (lim + 1) * (lim + 1) <= n; lim++)
            ;
        for (cnt = m = d = 0, i = 1; i <= lim; i++)
            if (n % i == 0) {
                if (check(i + 1)) a[m++] = i + 1;
                b[d++] = i;
                if (1LL * i * i != n) {
                    if (check(n / i + 1)) a[m++] = n / i + 1;
                    b[d++] = n / i;
                }
            }
        std::sort(a, a + m), std::sort(b, b + d);
        for (i = 0; i < d; i++) {
            if (b[i] <= lim)
                small[b[i]] = i;
            else
                big[n / b[i]] = i;
            for (g[i][m] = m, j = m - 1; ~j; j--)
                g[i][j] = b[i] % (a[j] - 1) ? g[i][j + 1] : j;
        }
```

```
        dfs(0, 1, n);
        std::sort(q, q + cnt);
        printf("%d\n", cnt);
        if (cnt)
            for (printf("%lld", q[0]), i = 1; i < cnt; i++)
                printf(" %lld", q[i]);
        puts("");
    }
    return 0;
}
```

6.23 大欧拉函数

```
/*
    欧拉函数
    已知N, 求phi(N), N<=10^18
    Pollard_Rho
*/
#include <bits/stdc++.h>
#define C 2730
#define S 3
using namespace std;
typedef long long ll;
ll n;
vector<ll> v;
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll mul(ll a, ll b, ll n) {
    return (a * b - (ll)(a / (long double)n * b + 1e-3) * n + n) % n;
}
ll pow(ll a, ll b, ll n) {
    ll d = 1;
    a %= n;
    while (b) {
        if (b & 1) d = mul(d, a, n);
        a = mul(a, a, n);
        b >>= 1;
    }
    return d;
}
bool check(ll a, ll n) {
    ll m = n - 1, x, y;
    int i, j = 0;
    while (!(m & 1)) m >>= 1, j++;
    x = pow(a, m, n);
    for (i = 1; i <= j; x = y, i++) {
```

```
        y = pow(x, 2, n);
        if ((y == 1) && (x != 1) && (x != n - 1)) return 1;
    }
    return y != 1;
}
bool miller_rabin(int times, ll n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    if (!(n & 1)) return 0;
    while (times--)
        if (check(rand() % (n - 1) + 1, n)) return 0;
    return 1;
}
ll pollard_rho(ll n, int c) {
    ll i = 1, k = 2, x = rand() % n, y = x, d;
    while (1) {
        i++, x = (mul(x, x, n) + c) % n, d = gcd(y - x, n);
        if (d > 1 && d < n) return d;
        if (y == x) return n;
        if (i == k) y = x, k <<= 1;
    }
}
void findfac(ll n, int c) {
    if (n == 1) return;
    if (miller_rabin(S, n)) {
        v.push_back(n);
        return;
    }
    ll m = n;
    while (m == n) m = pollard_rho(n, c--);
    findfac(m, c), findfac(n / m, c);
}
int main() {
    while (~scanf("%lld", &n)) {
        findfac(n, C);
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        ll phi = n;
        for (int i = 0; i < v.size(); i++) {
            ll y = v[i];
            phi = (phi / y) * (y - 1);
        }
        printf("%lld\n", phi);
    }
    return 0;
}
```

6.24 Pollard-Rho

```
/*
    Pollard_Rho
    题目大意:
        给出最大公约数和最小公倍数，满足要求的x和y，且x+y最小
    题解:
        我们发现，(x/gcd)*(y/gcd)=lcm/gcd，并且x/gcd和y/gcd互质
        那么我们先利用把所有的质数求出来Pollard_Rho，将相同的质数合并
        现在的问题转变成把合并后的质数分为两堆，使得x+y最小
        我们考虑不等式a+b>=2sqrt(ab),在a趋向于sqrt(ab)的时候a+b越小
        所以我们通过搜索求出最逼近sqrt(ab)的值即可。
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#define C 2730
#define S 3
using namespace std;
typedef long long ll;
ll n, m, s[1000], cnt, f[1000], cnf, ans;
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll mul(ll a, ll b, ll n) {
    return (a * b - (ll)(a / (long double)n * b + 1e-3) * n + n) % n;
}
ll pow(ll a, ll b, ll n) {
    ll d = 1;
    a %= n;
    while (b) {
        if (b & 1) d = mul(d, a, n);
        a = mul(a, a, n);
        b >>= 1;
    }
    return d;
}
bool check(ll a, ll n) {
    ll m = n - 1, x, y;
    int i, j = 0;
    while (!(m & 1)) m >>= 1, j++;
    x = pow(a, m, n);
    for (i = 1; i <= j; x = y, i++) {
        y = pow(x, 2, n);
        if ((y == 1) && (x != 1) && (x != n - 1)) return 1;
    }
    return y != 1;
}
```

```
bool miller_rabin(int times, ll n) {
    ll a;
    if (n == 1) return 0;
    if (n == 2) return 1;
    if (!(n & 1)) return 0;
    while (times--)
        if (check(rand() % (n - 1) + 1, n)) return 0;
    return 1;
}
ll pollard_rho(ll n, int c) {
    ll i = 1, k = 2, x = rand() % n, y = x, d;
    while (1) {
        i++, x = (mul(x, x, n) + c) % n, d = gcd(y - x, n);
        if (d > 1 && d < n) return d;
        if (y == x) return n;
        if (i == k) y = x, k <<= 1;
    }
}
void findfac(ll n, int c) {
    if (n == 1) return;
    if (miller_rabin(S, n)) {
        s[cnt++] = n;
        return;
    }
    ll m = n;
    while (m == n) m = pollard_rho(n, c--);
    findfac(m, c), findfac(n / m, c);
}
void dfs(int pos, long long x, long long k) {
    if (pos > cnf) return;
    if (x > ans && x <= k) ans = x;
    dfs(pos + 1, x, k);
    x *= f[pos];
    if (x > ans && x <= k) ans = x;
    dfs(pos + 1, x, k);
}
int main() {
    while (~scanf("%lld%lld", &m, &n)) {
        if (n == m) {
            printf("%lld %lld\n", n, n);
            continue;
        }
        cnt = 0;
        long long k = n / m;
        findfac(k, C);
        sort(s, s + cnt);
```

```
        f[0] = s[0];
        cnf = 0;
        for (int i = 1; i < cnt; i++) {
            if (s[i] == s[i - 1])
                f[cnf] *= s[i];
            else
                f[++cnf] = s[i];
        }
        long long tmp = (long long)sqrt(1.0 * k);
        ans = 1;
        dfs(0, 1, tmp);
        printf("%lld %lld\n", m * ans, k / ans * m);
    }
    return 0;
}
```

6.25 FFT

```
/*
    FFT
    int N=1; while(N<len)N<<=1;
    不带模 FFT::mul(A,B,C,N);
    带模 FFT::mulmod(A,B,C,N);
*/
typedef long long LL;
const int N = 524300;
int n, pos[N];
namespace FFT {
const int P = MOD;
struct comp {
    double r, i;
    comp(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    comp operator+(const comp &x) { return comp(r + x.r, i + x.i); }
    comp operator-(const comp &x) { return comp(r - x.r, i - x.i); }
    comp operator*(const comp &x) {
        return comp(r * x.r - i * x.i, i * x.r + r * x.i);
    }
    comp conj() { return comp(r, -i); }
} A[N], B[N];
const double pi = acos(-1.0);
void FFT(comp a[], int n, int t) {
    for (int i = 1; i < n; i++)
        if (pos[i] > i) swap(a[i], a[pos[i]]);
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1;
```

```
        double o = pi * 2 / m2 * t;
        comp _w(cos(o), sin(o));
        for (int i = 0; i < n; i += m2) {
            comp w(1, 0);
            for (int j = 0; j < m; j++) {
                comp &A = a[i + j + m], &B = a[i + j], t = w * A;
                A = B - t;
                B = B + t;
                w = w * _w;
            }
        }
    }
    if (t == -1)
        for (int i = 0; i < n; i++) a[i].r /= n;
}
void mul(long long *a, long long *b, long long *c, int k) {
    int i, j;
    for (i = 0; i < k; i++) A[i] = comp(a[i], b[i]);
    j = __builtin_ctz(k) - 1;
    for (int i = 0; i < k; i++) {
        pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
    }
    FFT(A, k, 1);
    for (int i = 0; i < k; i++) {
        j = (k - i) & (k - 1);
        B[i] = (A[i] * A[i] - (A[j] * A[j]).conj()) * comp(0, -0.25);
    }
    FFT(B, k, -1);
    for (int i = 0; i < k; i++) c[i] = (long long)(B[i].r + 0.5);
}
void mulmod(int *a, int *b, int *c, int k) {
    int i, j;
    for (i = 0; i < k; i++) a0[i] = a[i] / M, b0[i] = b[i] / M;
    j = __builtin_ctz(k) - 1;
    for (int i = 0; i < k; i++) {
        pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
    }
    for (mul(a0, b0, a0), i = 0; i < k; i++) {
        c[i] = 1LL * a0[i] * M * M % P;
        a1[i] = a[i] % M, b1[i] = b[i] % M;
    }
    for (mul(a1, b1, a1), i = 0; i < k; i++) {
        c[i] = (a1[i] + c[i]) % P, a0[i] = (a0[i] + a1[i]) % P;
        a1[i] = a[i] / M + a[i] % M, b1[i] = b[i] / M + b[i] % M;
    }
    for (mul(a1, b1, a1), i = 0; i < k; i++)
```

```
        c[i] = (1LL * M * (a1[i] - a0[i] + P) + c[i]) % P;
}
} // namespace FFT
```

6.26 任意模 FFT

```
/*
    任意模FFT
    mod(1e8~1e9级别)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef double db;
#define upmo(a, b) \
    (((a) = ((a) + (b)) % mod) < 0 ? (a) += mod : (a)) // 相加后取模
int n, mod;
//前面需要有 mod(1e8~1e9级别),upmo(a,b) 的定义
namespace FFT_MO {
const int FFT_MAXN = 1 << 18;
const db pi = 3.14159265358979323846264338327950288L;
struct cp {
    db a, b;
    cp(double a_ = 0, double b_ = 0) { a = a_, b = b_; }
    cp operator+(const cp &rhs) const { return cp(a + rhs.a, b + rhs.b); }
    cp operator-(const cp &rhs) const { return cp(a - rhs.a, b - rhs.b); }
    cp operator*(const cp &rhs) const {
        return cp(a * rhs.a - b * rhs.b, a * rhs.b + b * rhs.a);
    }
    cp operator!() const { return cp(a, -b); }
} nw[FFT_MAXN + 1], f[FFT_MAXN], g[FFT_MAXN], t[FFT_MAXN]; // a<->f,b<->g,t<~>c
int bitrev[FFT_MAXN];
//初始化 nw[],bitrev[]
void fft_init() {
    int L = 0;
    while ((1 << L) != FFT_MAXN) L++;
    for (int i = 1; i < FFT_MAXN; i++)
        bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (L - 1));
    for (int i = 0; i <= FFT_MAXN; i++)
        nw[i] = cp((db)cosl(2 * pi / FFT_MAXN * i),
                   (db)sinl(2 * pi / FFT_MAXN * i));
}
// n已保证是2的整数次幂
// flag=1:DFT | flag=-1: IDFT
void dft(cp *a, int n, int flag = 1) {
```

```
    int d = 0;
    while ((1 << d) * n != FFT_MAXN) d++;
    for (int i = 0; i < n; i++)
        if (i < (bitrev[i] >> d)) swap(a[i], a[bitrev[i] >> d]); // NOTICE!
    for (int l = 2; l <= n; l <<= 1) {
        int del =
            FFT_MAXN / l *
            flag; // 决定 wn是在复平面是顺时针还是逆时针变化, 以及变化间距
        for (int i = 0; i < n; i += l) {
            cp *le = a + i, *ri = a + i + (l >> 1);
            cp *w = flag == 1 ? nw : nw + FFT_MAXN; // 确定wn的起点
            for (int k = 0; k < (l >> 1); k++) {
                cp ne = *ri * *w;
                *ri = *le - ne, *le = *le + ne;
                le++, ri++, w += del;
            }
        }
    }
    if (flag != 1)
        for (int i = 0; i < n; i++) a[i].a /= n, a[i].b /= n;
}
// convo(a,n,b,m,c) a[0..n]*b[0..m] -> c[0..n+m]
void convo(LL *a, int n, LL *b, int m, LL *c) {
    for (int i = 0; i <= n + m; i++) c[i] = 0;
    int N = 2;
    while (N <= n + m) N <<= 1; // N+1是c扩展后的长度
    //扩展 a[],b[],存入f[],g[],注意取模
    for (int i = 0; i < N; i++) {
        LL aa = i <= n ? a[i] : 0, bb = i <= m ? b[i] : 0;
        aa %= mod, bb %= mod;
        f[i] = cp(db(aa >> 15), db(aa & 32767));
        g[i] = cp(db(bb >> 15), db(bb & 32767));
    }
    dft(f, N), dft(g, N);
    // 频域求积
    for (int i = 0; i < N; i++) {
        int j = i ? N - i : 0;
        t[i] = ((f[i] + !f[j]) * (!g[j] - g[i]) +
                (!f[j] - f[i]) * (g[i] + !g[j])) *
               cp(0, 0.25);
    }
    dft(t, N, -1);
    for (int i = 0; i <= n + m; i++) upmo(c[i], (LL(t[i].a + 0.5)) % mod << 15);
    // 频域求积
    for (int i = 0; i < N; i++) {
        int j = i ? N - i : 0;
```

```
        t[i] = (!f[j] - f[i]) * (!g[j] - g[i]) * cp(-0.25, 0) +
               cp(0, 0.25) * (f[i] + !f[j]) * (g[i] + !g[j]);
    }
    dft(t, N, -1);
    for (int i = 0; i <= n + m; i++)
        upmo(c[i], LL(t[i].a + 0.5) + (LL(t[i].b + 0.5) % mod << 30));
}
} // namespace FFT_MO
```

6.27 原根

```
/*
    求原根
*/
int q[10000];
int pow(ll a, int b, int P) {
    ll t = 1;
    for (; b; b >>= 1, a = (ll)a * a % P)
        if (b & 1) t = t * a % P;
    return t;
}
int getG(int n) {
    int i, j, t = 0;
    for (i = 2; (ll)i * i < n - 1; i++)
        if ((n - 1) % i == 0) q[t++] = i, q[t++] = (n - 1) / i;
    for (i = 2;; i++) {
        for (j = 0; j < t; j++)
            if (pow(i, q[j], n) == 1) break;
        if (j == t) return i;
    }
}
/*
    对于给定的p, n次询问给出数x是否是其原根
*/
using namespace std;
typedef long long ll;
ll modPow(ll a, int n, int p) {
    ll res = 1;
    while (n) {
        if (n & 1) res = res * a % p;
        n >>= 1;
        a = a * a % p;
    }
    return res;
}
```

```
int fac[100000], cnt;
int main() {
    int p, n;
    while (scanf("%d%d", &p, &n), n || p) {
        cnt = 0;
        p--;
        int lim = sqrt(p + 1);
        for (int i = 2; i <= lim; i++)
            if (p % i == 0) fac[cnt++] = i, fac[cnt++] = p / i;
        while (n--) {
            int x;
            scanf("%d", &x);
            bool fg = 1;
            for (int i = 0; i < cnt && fg; i++)
                if (modPow(x, fac[i], p + 1) == 1) fg = 0;
            puts(fg ? "YES" : "NO");
        }
    }
    return 0;
}
```

6.28 NTT

```
/*
    NTT
    总初始化: NTT::GetWn();
    while(len<n<<1)len<<=1;
    for(i=n;i<len;i++)A[i]=B[i]=0;
    NTT::NTT(A,len,1);
    NTT::NTT(B,len,1);
    for(i=0;i<len;i++)A[i]=NTT::mul(A[i],B[i],P);
    NTT::NTT(A,len,-1);
    要求P为费马素数
    root[998244353]=3
    root[1e9+7]=5
*/
namespace NTT {
const int G = 3;
const int NUM = 20;
int wn[20];
int mul(int x, int y, int P) { return (LL)x * y % P; }
int PowMod(int a, int b) {
    int res = 1;
    a %= P;
    while (b) {
```

```
        if (b & 1) res = mul(res, a, P);
        a = mul(a, a, P);
        b >>= 1;
    }
    return res;
}
void GetWn() {
    for (int i = 0; i < NUM; i++) {
        int t = 1 << i;
        wn[i] = PowMod(G, (P - 1) / t);
    }
}
void Change(int a[], int len) {
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(a[i], a[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}
void NTT(int a[], int len, int on) {
    Change(a, len);
    int id = 0;
    for (int h = 2; h <= len; h <<= 1) {
        id++;
        for (int j = 0; j < len; j += h) {
            int w = 1;
            for (int k = j; k < j + h / 2; k++) {
                int u = a[k] % P;
                int t = mul(a[k + h / 2], w, P);
                a[k] = (u + t) % P;
                a[k + h / 2] = ((u - t) % P + P) % P;
                w = mul(w, wn[id], P);
            }
        }
    }
    if (on == -1) {
        for (int i = 1; i < len / 2; i++) swap(a[i], a[len - i]);
        int inv = PowMod(len, P - 2);
        for (int i = 0; i < len; i++) a[i] = mul(a[i], inv, P);
    }
}
```

```
} // namespace NTT
```

6.29 FWT

```
/*
    快速沃尔什变化
*/
void FWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                // xor:a[i+j]=x+y,a[i+j+d]=x-y;
                // and:a[i+j]=x+y;
                // or:a[i+j+d]=x+y;
            }
}
void UFWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                // xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
                // and:a[i+j]=x-y;
                // or:a[i+j+d]=y-x;
            }
}
/*
    Problem: 有n堆石子，每堆都是m以内的质数，请问后手必胜的局面有几种
    Solution: 后手必胜，则sg为0，那么就是求n个m以内的数xor为0的情况有几种，
    首先筛出素数，保存素数的个数数组，利用FWT计算c[i^j]=a[i]*b[j]，
    计算n次的结果逆向变化回来就是最终的sg个数数组，
    在计算n次c[i]=a[i]*b[i]的过程中，等价于计算c[i]=a[i]^n，
    这里我们可以用快速幂优化一个log。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long LL;
const int N = 100000;
const LL mod = 1e9 + 7;
LL a[N], u;
int p[N], n, m;
void FWT(LL *a, int n) {
    for (int d = 1; d < n; d <<= 1)
```

```
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                LL x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % mod, a[i + j + d] = (x - y + mod) % mod;
            }
}
void UFWT(LL *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                LL x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % mod * u % mod,
                     a[i + j + d] = (x - y + mod) % mod * u % mod;
            }
}
LL pow(LL a, LL b, LL p) {
    LL t = 1;
    for (a %= p; b; b >>= 1LL, a = a * a % p)
        if (b & 1LL) t = t * a % p;
    return t;
}
int main() {
    for (int i = 2; i <= 50000; i++) p[i] = 1;
    for (int i = 2; i <= 50000; i++)
        if (p[i]) {
            for (int j = 2; i * j <= 50000; j++) p[i * j] = 0;
        }
    u = pow(2, mod - 2, mod);
    while (~scanf("%d%d", &n, &m)) {
        int len = 1;
        while (len <= m) len <<= 1;
        for (int i = 0; i < len; i++) a[i] = p[i] & (i <= m);
        FWT(a, len);
        for (int i = 0; i < len; i++) a[i] = pow(a[i], n, mod);
        UFWT(a, len);
        printf("%lld\n", a[0]);
    }
    return 0;
}
/*
    Problem: 给出一棵树，其每棵连通子树的价值为其点权的xor和，
    问有多少连通子树的价值为1~m
    Solution: 首先定1为根，转有根树，我们在树的每个节点保存一个权值数组，
    表示与其连通的子树的权值，当一个子树并入其父节点时，
    dp[x][i]=dp[x][i]+dp[x][j]*dp[y][k](j^k==i)，我们发现这是一个位运算卷积式子，
    所以树上转移可以用fwt加速。
```

```
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
typedef long long LL;
const int N = 1030;
const LL mod = 1e9 + 7;
const LL inv2 = (mod + 1) / 2;
int val[N];
LL dp[N][N], tmp[N];
vector<int> v[N];
void FWT(LL *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                LL x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % mod, a[i + j + d] = (x - y + mod) % mod;
            }
}
void UFWT(LL *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                LL x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % mod * inv2 % mod,
                      a[i + j + d] = (x - y + mod) % mod * inv2 % mod;
            }
}
int T, n, m, x, y;
LL ans[N];
void Cal(LL *a, LL *b) {
    for (int i = 0; i < m; i++) tmp[i] = a[i];
    FWT(a, m);
    FWT(b, m);
    for (int i = 0; i < m; i++) a[i] = (1ll * a[i] * b[i]) % mod;
    UFWT(a, m);
    UFWT(b, m);
    for (int i = 0; i < m; i++) a[i] = (a[i] + tmp[i]) % mod;
}
void DP(int x, int fx) {
    dp[x][val[x]] = 1;
    for (auto y : v[x]) {
        if (y == fx) continue;
        DP(y, x);
```

```
        Cal(dp[x], dp[y]);
    }
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for (int i = 0; i <= n; i++) v[i].clear();
        for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
        for (int i = 1; i < n; i++) {
            scanf("%d%d", &x, &y);
            v[x].push_back(y);
            v[y].push_back(x);
        }
        memset(dp, 0, sizeof(dp));
        memset(ans, 0, sizeof(ans));
        DP(1, 1);
        for (int i = 0; i < m; i++) {
            for (int j = 1; j <= n; j++) ans[i] = (ans[i] + dp[j][i]) % mod;
        }
        for (int i = 0; i < m; i++) printf(i < m - 1 ? "%d " : "%d\n", ans[i]);
    }
    return 0;
}
/*
    Problem: 定义一张无向图的价值:
        给每个节点染色使得每条边连接的两个节点颜色不相同的最少颜色数。
        对于给定的一张由n个点组成的无向图，求该图的2^n-1张非空子图的价值。
    Solution: 设f[i][S]表示i种颜色覆盖S这个集合的方案数，
    我们只要得到最小的i，f[i][S]大于0，那么i就是S集合的答案。
    显然有f[i][S]= f[1][u]×f[i-1][v](u|v==S)，
    处理位运算形式的卷积，所以我们现在只要求n次FWT，就可以得到答案。
*/
#include <cstdio>
const int N = 18, M = 1 << N;
char s[N + 2];
int T, n, i, j, g[N], f[N + 1][M], h[M];
unsigned int pow[M], ans;
void FWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                a[i + j + d] = a[i + j] + a[i + j + d];
            }
}
void UFWT(int *a, int n) {
```

```
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                a[i + j + d] = a[i + j + d] - a[i + j];
            }
}
void mul() {
    FWT(h, 1 << n);
    for (i = 2; i <= n; i++) {
        for (j = 0; j < 1 << n; j++) f[i][j] = f[i - 1][j];
        FWT(f[i], 1 << n);
        for (j = 0; j < 1 << n; j++) f[i][j] *= h[j];
        UFWT(f[i], 1 << n);
        for (j = 0; j < 1 << n; j++) f[i][j] = !!f[i][j];
    }
}
int main() {
    scanf("%d", &T);
    for (pow[0] = i = 1; i < M; i++) pow[i] = pow[i - 1] * 233;
    while (T--) {
        scanf("%d", &n);
        for (i = 0; i < n; i++) {
            scanf("%s", s);
            g[i] = 0;
            for (j = 0; j < n; j++)
                if (s[j] == '1') g[i] |= 1 << j;
        }
        for (i = 0; i < 1 << n; i++) f[1][i] = 0;
        for (i = f[1][0] = 1; i < (1 << n); i++) {
            j = i & -i;
            if (!f[1][i - j]) continue;
            if (g[__builtin_ctz(j)] & i) continue;
            f[1][i] = 1;
        }
        for (j = 0; j < 1 << n; j++) h[j] = f[1][j];
        mul();
        ans = 0;
        for (i = 1; i < 1 << n; i++) {
            for (j = 1; !f[j][i]; j++)
                ;
            ans += j * pow[i];
        }
        printf("%u\n", ans);
    }
    return 0;
}
```

```
/*
    Problem: 给出一个数集，A从中选择一些数，B从中选择一些数，不能同时不选
    要求两者选择的数异或和为0，问方案数
    Solution: 题目等价于选取一个非空且xor为0的集合并将其拆分为两个子集的方案数
    用dp表示xor为j的方案数，易得dp方程dp[i][j]=dp[i-1][j]+2*dp[i-1][j^a[i]]
    该式等价于dp数组与只有两个元素有值的g[0]=1,g[a[i]]=2的数组做卷积运算
    对g数组进行反演可以发现每次卷积的数组只包含3和-1，
    那么我们只要知道对一个下标来说，做的n次卷积中有几个3和-1，
    就能够直接乘算出答案，根据FWT的和等于和的FWT，我们将多次需要做卷积的数组相加，
    一并做FWT，得到他们和的反演值，在每个位置解关于3和-1的二元一次方程组，
    再将其替换为正确值，最后FWT求逆之后下标为0的答案减去1就是答案，
    减一是因为两个人取数不能同时为空。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int P = 998244353;
const int inv2 = (P + 1) >> 1;
const int N = 2000000;
void FWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % P, a[i + j + d] = (x - y + P) % P;
            }
}
void UFWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = 1LL * (x + y) * inv2 % P,
                     a[i + j + d] = 1LL * (x - y) * inv2 % P;
            }
}
int n, x, mx, pw[N], a[N];
int main() {
    pw[0] = 1;
    for (int i = 1; i < N; i++) pw[i] = 3LL * pw[i - 1] % P;
    while (~scanf("%d", &n)) {
        memset(a, 0, sizeof(a));
        for (int i = mx = 1; i <= n; i++) {
            scanf("%d", &x);
            a[0]++;
```

```
            a[x] += 2;
            mx = max(mx, x);
        }
        int m = 1;
        while (m <= mx) m <<= 1;
        FWT(a, m);
        for (int i = 0; i < m; i++) {
            x = (3ll * n + P - a[i]) * inv2 % P * inv2 % P;
            a[i] = (x & 1) ? (P - pw[n - x]) % P : pw[n - x];
        }
        UFWT(a, m);
        printf("%d\n", (a[0] + P - 1) % P);
    }
    return 0;
}
```

6.30 FFT+CDQ 分治

```
/*
    FFT+CDQ分治
*/
/*
    dp[i]=sum_{j=0}^{i-1} dp[j]w[i-j]
    c[i]=sum_{j=0}^i a[j]b[i-j]
    题目大意:
        给出一个数组w，表示不同长度的字段的权值，
        比如w[3]=5表示如果字段长度为3，则其权值为5，现在有长度为n的字段，
        求通过不同拆分得到的字段权值乘积和。
    题解:
        记DP[i]表示长度为i时候的答案，DP[i]=sum_{j=0}^{i-1}DP[j]w[i-j]，
        发现是一个卷积的式子，因此运算过程可以用FFT优化，
        但是由于在计算过程中DP[j]是未知值，顺次计算复杂度是O(n2logn)，
        考虑到加法运算对乘法运算可分配，因此可以采取CDQ分治，
        利用递归统计每个区间内左边DP值对右边DP值的贡献，
        对于每次贡献值的计算则利用FFT进行优化，优化时间复杂度至O(nlognlogn)。
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
typedef long long LL;
const int N = 524300, P = 313;
int n, pos[N];
namespace FFT {
struct comp {
```

```
    double r, i;
    comp(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    comp operator+(const comp &x) { return comp(r + x.r, i + x.i); }
    comp operator-(const comp &x) { return comp(r - x.r, i - x.i); }
    comp operator*(const comp &x) {
        return comp(r * x.r - i * x.i, i * x.r + r * x.i);
    }
    comp conj() { return comp(r, -i); }
} A[N], B[N];
const double pi = acos(-1.0);
void FFT(comp a[], int n, int t) {
    for (int i = 1; i < n; i++)
        if (pos[i] > i) swap(a[i], a[pos[i]]);
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1;
        double o = pi * 2 / m2 * t;
        comp _w(cos(o), sin(o));
        for (int i = 0; i < n; i += m2) {
            comp w(1, 0);
            for (int j = 0; j < m; j++) {
                comp &A = a[i + j + m], &B = a[i + j], t = w * A;
                A = B - t;
                B = B + t;
                w = w * _w;
            }
        }
    }
    if (t == -1)
        for (int i = 0; i < n; i++) a[i].r /= n;
}
void mul(int *a, int *b, int *c, int k) {
    int i, j;
    for (i = 0; i < k; i++) A[i] = comp(a[i], b[i]);
    j = __builtin_ctz(k) - 1;
    for (int i = 0; i < k; i++) {
        pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
    }
    FFT(A, k, 1);
    for (int i = 0; i < k; i++) {
        j = (k - i) & (k - 1);
        B[i] = (A[i] * A[i] - (A[j] * A[j]).conj()) * comp(0, -0.25);
    }
    FFT(B, k, -1);
    for (int i = 0; i < k; i++) c[i] = (long long)(B[i].r + 0.5) % P;
}
} // namespace FFT
```

```
int w[N], a[N], b[N], c[N], F[N];
void CDQ(int l, int r) {
    if (l == r) {
        F[l] += w[l];
        F[l] %= P;
        return;
    }
    int mid = (l + r) >> 1;
    CDQ(l, mid);
    int N = 1;
    while (N < r - l) N <<= 1;
    for (int i = 0; i <= mid - l; i++) a[i] = F[i + l];
    for (int i = mid - l + 1; i < N; i++) a[i] = 0;
    for (int i = 0; i < r - l; i++) b[i] = w[i + 1];
    for (int i = r - l; i < N; i++) b[i] = 0;
    FFT::mul(a, b, c, N);
    for (int i = mid + 1; i <= r; i++) {
        F[i] += c[i - l - 1];
        F[i] %= P;
    }
    CDQ(mid + 1, r);
}
int main() {
    while (~scanf("%d", &n) && n) {
        F[0] = 1;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &w[i]);
            w[i] %= P;
            F[i] = 0;
        }
        CDQ(1, n);
        printf("%d\n", F[n]);
    }
    return 0;
}
```

6.31 NTT+CDQ 分治

```
/*
    NTT+CDQ分治
*/
/*
    dp[i]=i!-sum_{j=0}^{i-1} dp[i-j]*j!
    题目大意:
        给出一个排列，每对逆序对之间连一条边，现在给出每个连通块中的元素，
```

```
        问满足要求的排列的种类
    题解:
        我们发现一个连通块中序号一定是连续的，否则方案为0，那么每个连通块方案的计算是独立的，
        而方案的数量只与连通块的大小有关，记大小为i的连通块方案为dp[i]，
        考虑不合法的情况，不妨设最后一个块为j个连续的数，则需要减去的方案为dp[i-j]*j!
        有dp[i]=i!-sum_{j=0}^{i-1} dp[i-j]*j!，这是一个卷积运算，
        但是由于在计算过程中dp[i-j]是未知值，顺次计算复杂度是O(n^2logn)，
        考虑到加法运算对乘法运算可分配，因此可以采取CDQ分治，
        利用递归统计每个区间内左边DP值对右边DP值的贡献，
        对于每次贡献值的计算则利用NTT进行优化，优化时间复杂度至O(nlognlogn)。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long LL;
const int P = 786433;
namespace NTT {
const int G = 10;
const int NUM = 20;
int wn[20];
int mul(int x, int y, int P) { return (LL)x * y % P; }
int PowMod(int a, int b) {
    int res = 1;
    a %= P;
    while (b) {
        if (b & 1) res = mul(res, a, P);
        a = mul(a, a, P);
        b >>= 1;
    }
    return res;
}
void GetWn() {
    for (int i = 0; i < NUM; i++) {
        int t = 1 << i;
        wn[i] = PowMod(G, (P - 1) / t);
    }
}
void Change(int a[], int len) {
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(a[i], a[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
```

```
        if (j < k) j += k;
    }
}
void NTT(int a[], int len, int on) {
    Change(a, len);
    int id = 0;
    for (int h = 2; h <= len; h <<= 1) {
        id++;
        for (int j = 0; j < len; j += h) {
            int w = 1;
            for (int k = j; k < j + h / 2; k++) {
                int u = a[k] % P;
                int t = mul(a[k + h / 2], w, P);
                a[k] = (u + t) % P;
                a[k + h / 2] = ((u - t) % P + P) % P;
                w = mul(w, wn[id], P);
            }
        }
    }
    if (on == -1) {
        for (int i = 1; i < len / 2; i++) swap(a[i], a[len - i]);
        int inv = PowMod(len, P - 2);
        for (int i = 0; i < len; i++) a[i] = mul(a[i], inv, P);
    }
}
} // namespace NTT
const int N = 1 << 20;
int f[N], dp[N], a[N], b[N];
void CDQ(int l, int r) {
    if (l == r) {
        dp[l] = (-dp[l] + f[l] + P) % P;
        return;
    }
    int mid = (l + r) >> 1;
    CDQ(l, mid);
    int len = 1, n = mid - l + 1;
    while (len <= n << 1) len <<= 1;
    a[0] = b[0] = 0;
    for (int i = 1, j = l; i < len; i++, j++) a[i] = (j <= mid ? dp[j] : 0);
    for (int i = 1; i < len; i++) b[i] = f[i];
    NTT::NTT(a, len, 1);
    NTT::NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = 1LL * a[i] * b[i] % P;
    NTT::NTT(a, len, -1);
    for (int i = mid + 1, j = mid - l + 2; i <= r; i++, j++)
        (dp[i] += a[j]) %= P;
```

```
    CDQ(mid + 1, r);
}
int T, n, m, l, r, cnt, x;
int main() {
    NTT::GetWn();
    for (int i = f[0] = 1; i <= 100000; i++) f[i] = 1LL * f[i - 1] * i % P;
    CDQ(1, 100000);
    scanf("%d", &T);
    while (T--) {
        int ans = 1;
        scanf("%d%d", &n, &m);
        while (m--) {
            l = n + 1, r = -1;
            scanf("%d", &cnt);
            for (int i = 1; i <= cnt; i++) {
                scanf("%d", &x);
                l = min(l, x);
                r = max(r, x);
            }
            ans = 1LL * ans * dp[cnt] % P;
            if (r - l + 1 != cnt) ans = 0;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

6.32 多项式理论

```
/*
    多项式
    乘法 NTT之后相乘
    开方 getroot
    求逆(倒数) getinv
    求对数函数 getln
    求指数函数 getexp
    求幂 getpow
    初始化 init
    拉格朗日反演 [x^n]F(x)=[x^(n-1)]*(1/n)*(x/G(x))^n
*/
#include <cstdio>
typedef long long ll;
const int N = 262144, K = 17; // N = 1 << (K + 1)
int n, m, i, k;
int a[N + 10], b[N + 10], tmp[N + 10], tmp2[N + 10];
```

```
int P = 998244353, G = 3, g[K + 1], ng[K + 10], inv[N + 10], inv2;
int power(int a, int b) {
    int t = 1;
    for (; b; b >>= 1, a = (ll)a * a % P)
        if (b & 1) t = (ll)t * a % P;
    return t;
}
void NTT(int* a, int n, int t) {
    for (int i = 1, j = 0; i < n - 1; i++) {
        for (int s = n; j ^= s >>= 1, ~j & s;)
            ;
        if (i < j) {
            int k = a[i];
            a[i] = a[j];
            a[j] = k;
        }
    }
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1, _w = t == 1 ? g[d] : ng[d];
        for (int i = 0; i < n; i += m2)
            for (int w = 1, j = 0; j < m; j++) {
                int &A = a[i + j + m], &B = a[i + j], t = (ll)w * A % P;
                A = B - t;
                if (A < 0) A += P;
                B = B + t;
                if (B >= P) B -= P;
                w = (ll)w * _w % P;
            }
    }
    if (t == -1)
        for (int i = 0, j = inv[n]; i < n; i++) a[i] = (ll)a[i] * j % P;
}
// 给定a, 求a的逆元b
void getinv(int* a, int* b, int n) {
    if (n == 1) {
        b[0] = power(a[0], P - 2);
        return;
    }
    getinv(a, b, n >> 1);
    int k = n << 1, i;
    for (i = 0; i < n; i++) tmp[i] = a[i];
    for (i = n; i < k; i++) tmp[i] = b[i] = 0;
    NTT(tmp, k, 1), NTT(b, k, 1);
    for (i = 0; i < k; i++) {
        b[i] = (ll)b[i] * (2 - (ll)tmp[i] * b[i] % P) % P;
        if (b[i] < 0) b[i] += P;
```

```
    }
    NTT(b, k, -1);
    for (i = n; i < k; i++) b[i] = 0;
}
// 给定a，求a的对数函数，且a[0] = 1
void getln(int* a, int* b, int n) {
    getinv(a, tmp2, n);
    int k = n << 1, i;
    for (i = 0; i < n - 1; i++) b[i] = (ll)a[i + 1] * (i + 1) % P;
    for (i = n - 1; i < k; i++) b[i] = 0;
    NTT(b, k, 1), NTT(tmp2, k, 1);
    for (i = 0; i < k; i++) b[i] = (ll)b[i] * tmp2[i] % P;
    NTT(b, k, -1);
    for (i = n - 1; i; i--) b[i] = (ll)b[i - 1] * inv[i] % P;
    b[0] = 0;
}
// 给定a，求a的指数函数，且a[0]=0
void getexp(int* a, int* b, int n) {
    if (n == 1) {
        b[0] = 1;
        return;
    }
    getexp(a, b, n >> 1);
    getln(b, tmp, n);
    int k = n << 1, i;
    for (i = 0; i < n; i++) {
        tmp[i] = a[i] - tmp[i];
        if (tmp[i] < 0) tmp[i] += P;
    }
    if ((++tmp[0]) == P) tmp[0] = 0;
    for (i = n; i < k; i++) tmp[i] = b[i] = 0;
    NTT(tmp, k, 1), NTT(b, k, 1);
    for (i = 0; i < k; i++) b[i] = (ll)b[i] * tmp[i] % P;
    NTT(b, k, -1);
    for (i = n; i < k; i++) b[i] = 0;
}
// 给定a，求a的平方根b，且a[0]=1
void getroot(int* a, int* b, int n) {
    if (n == 1) {
        b[0] = 1;
        return;
    }
    getroot(a, b, n >> 1);
    getinv(b, tmp2, n);
    int k = n << 1, i;
    for (i = 0; i < n; i++) tmp[i] = a[i];
```

```
    for (i = n; i < k; i++) tmp[i] = b[i] = 0;
    NTT(tmp, k, 1), NTT(b, k, 1), NTT(tmp2, k, 1);
    for (i = 0; i < k; i++)
        b[i] = ((ll)b[i] * b[i] + tmp[i]) % P * inv2 % P * tmp2[i] % P;
    NTT(b, k, -1);
    for (i = n; i < k; i++) b[i] = 0;
}
// 给定a，求a的pw次幂，且a[0] = 1
void getpow(int* a, int n, int pw) {
    getln(a, b, n);
    for (i = 0; i < n; i++) b[i] = (ll)b[i] * pw % P;
    getexp(b, a, n);
}
void init() {
    g[K] = power(G, (P - 1) / N), ng[K] = power(g[K], P - 2);
    for (i = K - 1; ~i; i--)
        g[i] = (ll)g[i + 1] * g[i + 1] % P,
        ng[i] = (ll)ng[i + 1] * ng[i + 1] % P;
    for (inv[1] = 1, i = 2; i <= N; i++)
        inv[i] = (ll)(P - inv[P % i]) * (P / i) % P;
    inv2 = inv[2];
}
/*
    Example1:
        给定一个包含n个数字的非重数集C和正整数m，
        现在要求构造一棵点权二叉树，要求每个节点的权值在集合C中，
        定义二叉树的权值为所有点权之和，现在希望对于s [1,m]，
        求出本质不同的权值为s的二叉树数量
        两棵二叉树相同当且仅当树结构相同且对应位置点权相同，
        左右子树交换当做不相同处理
    Solution:
        我们记包含n的数字的非重集合C的生成函数为G(x)
        G(x)=1+x^{C_1}+x^{C_2}+……
        我们记满足条件的二叉树的生成函数为F(x)
        x^i前的系数为权值为i的二叉树的数量
        则有方程F(x)=G(x)*F(x)^2+1
        F(x)^2表示左右子树情况，G(x)表示子树根节点选值情况，+1表示子树为空
        移项得到G(x)*F(x)^2-F(x)+1=0，方程有解为(1±\sqrt(1-4G(x))/(2G(x))
        上下同乘(1 \sqrt(1-4G(x))得到2/(1 \sqrt(1-4G(x))
        因为分母常数项不能为0，所以F(x)=2/(1+\sqrt(1-4G(x))
        多项式开方并求逆即可
*/
int main() {
    init();
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++) {
```

```
        scanf("%d", &k);
        if (k <= m) a[k]++;
    }
    int len;
    for (len = 1; len <= m; len <<= 1);
    for (i = 1; i < len; i++) if (a[i]) a[i] = P - 4;
    a[0]++;
    getroot(a, b, len);
    b[0]++;
    getinv(b, a, len);
        for (int i = 1; i <= m; i++)
        printf("%d\n", (a[i] + a[i]) % P);
    return 0;
}
/*
    Example2:
        给定一个含有m个正整数的集合D和一个正整数s,
        一棵树是合法的当且仅当每个非叶子节点的孩子数量均在集合D内
        问有多少棵不同的含有s个叶节点的树，孩子有序
    Solution:
        我们记包含n的数字的非重集合C的生成函数为G(x)
        G(x)=1+x^{C_1}+x^{C_2}+……
        我们记满足条件的树的生成函数为F(x)
        则F(x)=G(F(x))+x
        表示子树的生成函数的合法幂次之和以及没有子树的情况
        F(x)-G(F(x))=x，F(x)是x-G(x)构成复合逆(反函数)
        根据拉格朗日反演[x^n]G(x)=(1/n)[x^(n-1)](x/F(x))^n
        我们可以得到[x^n]F(x)=(1/n)[x^(n-1)](x/(x-G(x)))^n
        即[x^n]F(x)=(1/n)[x^(n-1)](1-G(x)/x)^{-n}
        计算后面这个多项式的n-1项即可
*/
int s, m;
int main() {
    init();
    scanf("%d%d", &s, &m);
    a[0] = 1;
    for (int i = 1; i <= m; i++) {
        scanf("%d", &k);
        a[k - 1] = (a[k - 1] + P - 1) % P;
    }
    for (k = 1; k <= s; k <<= 1);
    getpow(a, k, P - s);
    printf("%d", 1ll * a[s - 1] * power(s, P - 2) % P);
    return 0;
}
```

6.33 生成函数

```
/*
    Problem:
        构造一个长度为n(n \le 10^18)数列，每个数字属于[1,m](m \le 2*10^5)
        要求每个偶数出现次数均为偶数次
    Solution:
        多重集排列问题，用指数型生成函数处理
        (1+(x^1)/(1!)+(x^2)/(2!)+(x^3)/(3!)+……)
        本题的指数型生成函数为
        (1+(x^1)/(1!)+(x^2)/(2!)+(x^3)/(3!)+……)^((m+1)/2)*(1+(x^2)/(2!)+(x^4)/(4!)+(x^6)/(6!)+……)^(m/2)
        根据泰勒公式
        e^x=(1+(x^1)/(1!)+(x^2)/(2!)+(x^3)/(3!)+……)
        e^-x=(1-(x^1)/(1!)+(x^2)/(2!)-(x^3)/(3!)+……)
        得G(x)=(e^x)^((m+1)/2)*((e^x+e^-x)/2)^(m/2)
        第n项系数为\sum_{i=1}^{m/2}C(m/2, i)*(m-2*i)^n
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int P = 1e9 + 7;
const int U = 200000;
int f[U + 3], rf[U + 3];
ll inv(ll a, ll m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
void init() {
    f[0] = 1;
    for (int i = 1; i <= U; i++) f[i] = (ll)f[i - 1] * i % P;
    rf[U] = inv(f[U], P);
    for (int i = U; i; i--) rf[i - 1] = (ll)rf[i] * i % P;
}
ll C(int n, int m) {
    if (m < 0 || m > n) return 0;
    return (ll)f[n] * rf[m] % P * rf[n - m] % P;
}
ll powmod(ll a, ll b, ll P) {
    ll t = 1;
    for (; b; b >>= 1, a = a * a % P)
        if (b & 1) t = t * a % P;
    return t;
}
int T, m;
ll n;
int main() {
    init();
    scanf("%d", &T);
    while (T--) {
```

```
        scanf("%lld%d", &n, &m);
        ll ans = 0;
        int t = m >> 1;
        for (int i = 0; i <= t; i++) {
            ans = (ans + C(t, i) * powmod(m - 2 * i, n, P) % P) % P;
        }
        ans = ans * powmod(powmod(2, t, P), P - 2, P) % P;
        printf("%lld\n", ans);
    }
    return 0;
}
/*
    Problem:
        给定F_1=1, F_n=a*F_{n-1}+1
        有一个有n个元素的多重集合S
        value(s)=F_{\sum_{s_i}}
        定义ans(k)=\sum_{s \in S \and |s|=k}value(s)
        求ans(1),ans(2),……,ans(n) mod 100003
        n \le 10^5, 2 \le a \le 1000, 1 \le s_i \le 1e9
    Solution:
        F_n=\frac{a^n-1}{a-1}
        对于给定的k，我们计算出不同方案下的a^sum之和，sum由k个s相加得到
        减去方案数C_n^k，然后除以(a-1)即可
        前者为母函数(x+a^{s_1})(x+a^{s_2})……(x+a^{s_n})的x^{n-k}前的系数
        所以我们计算这个母函数前的系数，fft计算即可
        我们可以通过分治区间优化多次等长数组的卷积，复杂度O(nlognlogn)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 524300;
const int P = 100003;
#define double long double
int pos[N];
namespace FFT {
struct comp {
    double r, i;
    comp(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    comp operator+(const comp &x) { return comp(r + x.r, i + x.i); }
    comp operator-(const comp &x) { return comp(r - x.r, i - x.i); }
    comp operator*(const comp &x) {
        return comp(r * x.r - i * x.i, i * x.r + r * x.i);
    }
    comp conj() { return comp(r, -i); }
} A[N], B[N];
const double pi = acos(-1.0);
```

```
void FFT(comp a[], int n, int t) {
    for (int i = 1; i < n; i++)
        if (pos[i] > i) swap(a[i], a[pos[i]]);
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1;
        double o = pi * 2 / m2 * t;
        comp _w(cos(o), sin(o));
        for (int i = 0; i < n; i += m2) {
            comp w(1, 0);
            for (int j = 0; j < m; j++) {
                comp &A = a[i + j + m], &B = a[i + j], t = w * A;
                A = B - t;
                B = B + t;
                w = w * _w;
            }
        }
    }
    if (t == -1)
        for (int i = 0; i < n; i++) a[i].r /= n;
}
void mul(long long *a, long long *b, long long *c, int k) {
    int i, j;
    for (i = 0; i < k; i++) A[i] = comp(a[i], b[i]);
    j = __builtin_ctz(k) - 1;
    for (int i = 0; i < k; i++) {
        pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
    }
    FFT(A, k, 1);
    for (int i = 0; i < k; i++) {
        j = (k - i) & (k - 1);
        B[i] = (A[i] * A[i] - (A[j] * A[j]).conj()) * comp(0, -0.25);
    }
    FFT(B, k, -1);
    for (int i = 0; i < k; i++) c[i] = (long long)(B[i].r + 0.5);
}
} // namespace FFT
ll a[N], b[N], c[N];
int pw[N], s[N];
vector<int> v[N];
void solve(int x, int l, int r) {
    if (l == r) {
        v[x].push_back(1);
        v[x].push_back(pw[l]);
        return;
    }
    int mid = (l + r) >> 1;
```

```
    solve(x << 1, l, mid);
    solve(x << 1 | 1, mid + 1, r);
    int N = 1;
    while (N <= r - l + 1) N <<= 1;
    for (int i = 0; i < N; i++) a[i] = b[i] = 0;
    for (int i = 0; i <= mid - l + 1; i++) a[i] = v[x << 1][i];
    for (int i = 0; i <= r - mid; i++) b[i] = v[x << 1 | 1][i];
    FFT::mul(a, b, c, N);
    for (int i = 0; i <= r - l + 1; i++) v[x].push_back(c[i] % P);
}
namespace Comb {
const int U = P - 1;
int f[U + 3], rf[U + 3];
ll inv(ll a, ll m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
void init() {
    f[0] = 1;
    for (int i = 1; i <= U; i++) f[i] = (ll)f[i - 1] * i % P;
    rf[U] = inv(f[U], P);
    for (int i = U; i; i--) rf[i - 1] = (ll)rf[i] * i % P;
}
ll C(int n, int m) {
    if (m < 0 || m > n) return 0;
    return (ll)f[n] * rf[m] % P * rf[n - m] % P;
}
} // namespace Comb
ll powmod(ll a, ll b, ll P) {
    ll t = 1;
    for (; b; b >>= 1, a = a * a % P)
        if (b & 1) t = t * a % P;
    return t;
}
int n, A, k, q, ans[N];
int main() {
    Comb::init();
    scanf("%d%d%d", &n, &A, &q);
    for (int i = 1; i <= n; i++) scanf("%d", &s[i]);
    for (int i = 1; i <= n; i++) pw[i] = powmod(A, s[i], P);
    solve(1, 1, n);
    int t = powmod((A - 1 + P) % P, P - 2, P);
    for (int i = 1; i <= n; i++)
        ans[i] = 1ll * (v[1][i] - Comb::C(n, i) + P) % P * t % P;
    while (q--) {
        scanf("%d", &k);
        printf("%d\n", ans[k]);
    }
    return 0;
```

```
}
/*
    Problem:
        给定数组a，b_i=\sum_{j=i-kx}a_j，k等于1到3
        对于给定一个k做一次操作，用得到的数组b去替换a，
        问m次操作后得到的a数组是什么
    Solution:
        a数组的生成函数为a_0+a_1*x+a_2*x^2+……+a_n*x^n
        对于操作k相当于乘上生成函数1+x^k+x^(2k)+……x^(nk)
        多次操作k就相当于(1+x^k+x^(2k)+……x^(nk))^m
        x^(ik)前的系数是(1+x+x^2+……+x^n)^m展开的第i项的系数，
        x取0~1之间时(1+x+x^2+……+x^n)等于1/(1-x)，
        (1-x)^-m的第i项为C(m+i-1,i)*x^i
        则x^y的系数为C(m-1+y/k,m-1)，
        我们将每种操作分别考虑，得到系数，计算与a的卷积即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 1 << 19 | 7;
const int P = 998244353;
namespace NTT {
const int G = 3;
const int NUM = 20;
int wn[20];
int mul(int a, int b, int P) {
    return 1ll * a * b % P;
}
int PowMod(int a, int b) {
    int res = 1;
    a %= P;
    while (b) {
        if (b & 1) res = mul(res, a, P);
        a = mul(a, a, P);
        b >>= 1;
    }
    return res;
}
void GetWn() {
    for (int i = 0; i < NUM; i++) {
        int t = 1 << i;
        wn[i] = PowMod(G, (P - 1) / t);
    }
}
void Change(int a[], int len) {
    int i, j, k;
```

```
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(a[i], a[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}
void NTT(int a[], int len, int on) {
    Change(a, len);
    int id = 0;
    for (int h = 2; h <= len; h <<= 1) {
        id++;
        for (int j = 0; j < len; j += h) {
            int w = 1;
            for (int k = j; k < j + h / 2; k++) {
                int u = a[k] % P;
                int t = mul(a[k + h / 2], w, P);
                a[k] = (u + t) % P;
                a[k + h / 2] = ((u - t) % P + P) % P;
                w = mul(w, wn[id], P);
            }
        }
    }
    if (on == -1) {
        for (int i = 1; i < len / 2; i++) swap(a[i], a[len - i]);
        int inv = PowMod(len, P - 2);
        for (int i = 0; i < len; i++) a[i] = mul(a[i], inv, P);
    }
}
void MUL(int *a, int *b, int n) {
    int len = 1;
    while (len < n * 2) len <<= 1;
    fill(a + n, a + len, 0);
    NTT::NTT(a, len, 1);
    fill(b + n, b + len, 0);
    NTT::NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = 1ll * a[i] * b[i] % P;
    NTT::NTT(a, len, -1);
}
} // namespace NTT
namespace Comb {
const int U = 2000000;
int f[U + 3], rf[U + 3];
```

```
LL inv(LL a, LL m) { return (a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m); }
void init() {
    f[0] = 1;
    for (int i = 1; i <= U; i++) f[i] = (LL)f[i - 1] * i % P;
    rf[U] = inv(f[U], P);
    for (int i = U; i; i--) rf[i - 1] = (LL)rf[i] * i % P;
}
LL C(int n, int m) {
    if (m < 0 || m > n) return 0;
    return (LL)f[n] * rf[m] % P * rf[n - m] % P;
}
} // namespace Comb
int a[N], b[N], c[4];
void getc(int *a, int k, int t, int n) {
    fill(a, a + n, 0);
    for (int i = 0; i < n; i += k) a[i] = Comb::C(i / k + t - 1, t - 1);
}
int T, n, m;
int main() {
    scanf("%d", &T);
    Comb::init();
    NTT::GetWn();
    while (T--) {
        scanf("%d%d", &n, &m);
        memset(c, 0, sizeof(c));
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
            a[i] %= P;
        }
        for (int i = 1; i <= m; i++) {
            int k;
            scanf("%d", &k);
            c[k]++;
        }
        for (int i = 1; i <= 3; i++) {
            if (!c[i]) continue;
            getc(b, i, c[i], n);
            NTT::MUL(a, b, n);
        }
        long long ans = 0;
        for (int i = 0; i < n; i++) ans ^= ((i + 1ll) * a[i]);
        printf("%lld\n", ans);
    }
    return 0;
}
```

6.34 素数筛

```
/*
    筛法求素数
*/
const int M = 3000500;
int p[400010], pNum;
bool f[M];
void Prime() {
    int i, j;
    for (i = 2; i < M; i++) {
        if (!f[i]) {
            p[pNum++] = i;
        }
        for (j = 0; j < pNum && p[j] * i < M; j++) {
            f[p[j] * i] = 1;
            if (!(i % p[j])) break;
        }
    }
}
```

6.35 素数测试

```
/*
    素数测试
*/
LL mulmod(LL a, LL b, LL p) {
    LL t = 0;
    while (b) {
        if (b & 1) t = (t + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return t;
}
LL powmod(LL a, LL b, LL p) {
    LL t = 1;
    while (b) {
        if (b & 1) t = mulmod(t, a, p);
        a = mulmod(a, a, p);
        b >>= 1;
    }
    return t;
}
int strong_pseudo_primetest(LL n, int base) {
```

```
    LL n2 = n - 1, res;
    int s = 0;
    while (n2 % 2 == 0) n2 >>= 1, s++;
    res = powmod(base, n2, n);
    if ((res == 1) || (res == n - 1)) return 1;
    s--;
    while (s >= 0) {
        res = mulmod(res, res, n);
        if (res == n - 1) return 1;
        s--;
    }
    return 0; // n is not a strong pseudo prime
}
int isprime(LL n) {
    static LL testNum[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    static LL lim[] = {4, 0, 1373653LL, 25326001LL,
                    25000000000LL, 2152302898747LL, 3474749660383LL, 341550071728321LL,
                    0, 0, 0, 0};
    if (n < 2 || n == 3215031751LL) return 0;
    for (int i = 0; i < 12; ++i) {
        if (n < lim[i]) return 1;
        if (strong_pseudo_primetest(n, testNum[i]) == 0) return 0;
    }
    return 1;
}
```

6.36 快速素数个数统计

```
/*
    快速素数个数统计
    1s内计算1e11以内素数的个数
*/
#define ll long long
using namespace std;
ll f[340000], g[340000], n;
void init() {
    ll i, j, m;
    for (m = 1; m * m <= n; ++m) f[m] = n / m - 1;
    for (i = 1; i <= m; ++i) g[i] = i - 1;
    for (i = 2; i <= m; ++i) {
        if (g[i] == g[i - 1]) continue;
        for (j = 1; j <= min(m - 1, n / i / i); ++j) {
            if (i * j < m)
                f[j] -= f[i * j] - g[i - 1];
            else
```

```
            f[j] -= g[n / i / j] - g[i - 1];
        }
        for (j = m; j >= i * i; --j) g[j] -= g[j / i] - g[i - 1];
    }
}
int main() {
    while (~scanf("%lld", &n)) {
        init();
        printf("%lld\n", f[1]);
    }
    return 0;
}
```

6.37 Power 定理

```
/*
    素数唯一分解定理
*/
const int M = 50010;
const int mod = 1e9 + 7;
int a[M], p[M], pNum, Num[M];
bool f[M];
void Prime() {
    int i, j;
    for (i = 2; i < M; i++) {
        if (!f[i]) {
            p[pNum++] = i;
        }
        for (j = 0; j < pNum && p[j] * i < M; j++) {
            f[p[j] * i] = 1;
            if (!(i % p[j])) break;
        }
    }
}
/*
    用于计算对于每个素数，在n的阶乘里的幂次
*/
void Power(int n, int a[]) {
    for (int i = 0; i < pNum; i++) {
        if (p[i] > n) break;
        int t = p[i];
        while (t <= n) {
            a[i] += n / t;
            t = t * p[i];
        }
```

```
    }
}
/*
    求一个数的阶乘有几个约数
*/
int T, n;
int main() {
    Prime();
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        memset(a, 0, sizeof(a));
        Power(n, a);
        int ans = 1;
        for (int i = 0; i < pNum; i++) {
            if (p[i] > n) break;
            ans = 1ll * ans * (a[i] + 1) % mod;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

6.38 线性筛

```
/*
    mu[i]: 莫比乌斯函数
    phi[i]: 欧拉函数
    d[i]: i的质因子个数
*/
bool notp[N];
int prime[N], pnum, mu[N], c[N], d[N], phi[N];
void sieve() {
    memset(notp, 0, sizeof(notp));
    notp[0] = notp[1] = mu[1] = phi[1] = 1;
    pnum = 0;
    for (int i = 2; i < N; i++) {
        if (!notp[i]) {
            prime[++pnum] = i;
            mu[i] = -1;
            phi[i] = i - 1;
            d[i] = 1;
        }
        for (int j = 1; j <= pnum && prime[j] * i < N; j++) {
            notp[prime[j] * i] = 1;
```

```
            d[prime[j] * i] = d[i] + 1;
            if (i % prime[j] == 0) {
                mu[prime[j] * i] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            mu[prime[j] * i] = -mu[i];
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
}
```

6.39 分块求和

```
/*
    分块求和
*/
ll solve(int n, int m) {
    if (n > m) swap(n, m);
    ll res = 0;
    for (int i = 1, last; i <= n; i = last + 1) {
        last = min(n / (n / i), m / (m / i));
        res += (f[last] - f[i - 1]) * (n / i) * (m / i);
    }
    return res;
}
/*
    Example1:
        求 ((n%i)*(m%j))其中1<=i<=n,1<=j<=m,i j。
    Solution:
        sum_{i=1}^{n}sum_{j=1}^{m}((n%i)*(m%j))(i j)
        =sum_{i=1}^{n}sum_{j=1}^{m}(n-(n/i)*i)*(m-(m/j)*j)
            -sum_{i=1}^{min(n,m)}(n-(n/i)*i)*(m-(m/i)*i)
        =sum_{i=1}^{n}(n-n/i)*sum_{i=1}^{m}(m-m/i)
            -sum_{i=1}^{min(n,m)}n*m-n*(m/i)*i
            -m*(n/i)*i+(n/i)(m/i)*i*i
        我们对于n/i分段统计即可。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long LL;
const LL inv6 = 3323403;
const LL mod = 19940417;
LL n, m, ans;
```

```
LL sum(LL a, LL b) { return (b - a + 1) * (a + b) / 2 % mod; }
LL sum2(LL x) { return x * (x + 1) % mod * (2 * x + 1) % mod * inv6 % mod; }
LL cal(LL n) {
    LL res = 0;
    for (LL l = 1, r; l <= n; l = r + 1) {
        r = n / (n / l);
        res = (res + n * (r - l + 1) % mod - sum(l, r) * (n / l)) % mod;
    }
    return (res + mod) % mod;
}
int main() {
    while (~scanf("%lld%lld", &n, &m)) {
        ans = cal(n) * cal(m) % mod;
        if (n > m) swap(n, m);
        for (int l = 1, r; l <= n; l = r + 1) {
            r = min(n / (n / l), m / (m / l));
            LL s1 = n * m % mod * (r - l + 1) % mod;
            LL s2 =
                (n / l) * (m / l) % mod * (sum2(r) - sum2(l - 1) + mod) % mod;
            LL s3 = (n / l * m + m / l * n) % mod * sum(l, r) % mod;
            ans = (ans - (s1 + s2 - s3) % mod + mod) % mod;
        }
        printf("%lld\n", ans);
    }
    return 0;
}
/*
    Example2:
        求 gcd(x^a-1,x^b-1) (1 a,b n)
    Solution:
         gcd(x^a-1,x^b-1)= (x^gcd(a,b)-1)
        枚举gcd，题目变为求 ((x^k-1)*(2* phi(n/k)-1))
        n/k的取值个数有限，我们对n/k数值分块即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1000000 + 10;
const int P = 1e9 + 7;
bool notp[N];
int phi[N], prime[N], pnum;
ll F[N];
void sieve() {
    memset(notp, 0, sizeof(notp));
    F[1] = phi[1] = notp[0] = notp[1] = 1;
    pnum = 0;
```

```
    for (int i = 2; i < N; i++) {
        if (!notp[i]) prime[++pnum] = i, phi[i] = i - 1;
        for (int j = 1; j <= pnum && prime[j] * i < N; j++) {
            notp[prime[j] * i] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
        F[i] = (F[i - 1] + phi[i]) % P;
    }
}
ll pow(ll a, ll b, ll p) {
    ll t = 1;
    for (a %= p; b; b >>= 1ll, a = a * a % p)
        if (b & 1ll) t = t * a % p;
    return t;
}
ll solve(int x, int n) {
    ll res = 0;
    for (int i = 1, last; i <= n; i = last + 1) {
        last = n / (n / i);
        ll t = pow(x, last + 1, P) - pow(x, i, P);
        t = t * pow(x - 1, P - 2, P) % P - last + i - 1;
        res += (t + P) % P * ((F[n / i] << 1) - 1) % P;
        res %= P;
    }
    return res;
}
int T, n, x;
int main() {
    scanf("%d", &T);
    sieve();
    while (T--) {
        scanf("%d%d", &x, &n);
        if (x == 1) {
            puts("0");
            continue;
        }
        printf("%lld\n", (solve(x, n) + P) % P);
    }
    return 0;
}
/*
    Example3:
```

```
        给出一个数列每个位置可以取到的最大值，
        问这个可以构造多少个数列，使得他们的最大公约数大于1
    Solution:
        我们可以枚举最大公约数k，对于k来说，
        他对答案的贡献为 [ai/k]，我们将数列中的数字转化为权值数组
         _{i=1}^{100000}[i/k]，对于求解i/k的部分我们可以进行数值分块，
        j*k-1~j*k+k-1的数值除k得到的结果都是相同的，因此可以直接求这个结果的幂次，
        这时候只要再加一个权值数组的前缀和，问题就迎刃而解了。
        数值分块计算的复杂度为n+n/2+n/3+n/4+n/5+……+n/n=nlogn。
        对于计算结果，我们需要进行容斥，奇数次素数乘的系数为1，偶数次素数乘的系数为-1，
        对于出现素数幂的合数其系数为0，
        我们发现这个容斥恰好是莫比乌斯函数的相反数，因此我们取反即可。
        这有个小小的优化，对于系数为0的情况，我们可以直接跳过，不进行计算。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 200010;
typedef long long LL;
const LL mod = 1000000007;
int T, n, a[N], b[N], cnt[N], cas = 1, p[N];
LL ans = 0;
int tot, miu[N], sum[N], v[N];
void read(int& a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
void mobius(int n) {
    int i, j;
    for (miu[1] = 1, i = 2; i <= n; i++) {
        if (!v[i]) p[tot++] = i, miu[i] = -1;
        for (j = 0; j < tot && i * p[j] <= n; j++) {
            v[i * p[j]] = 1;
            if (i % p[j])
                miu[i * p[j]] = -miu[i];
            else
                break;
        }
    }
    for (i = 1; i < n; i++) sum[i] = sum[i - 1] + miu[i];
}
LL pow(LL a, LL b, LL p) {
```

```
    if (b == 0) return 1;
    LL t = 1;
    for (a %= p; b; b >>= 1LL, a = a * a % p)
        if (b & 1LL) t = t * a % p;
    return t;
}
int main() {
    read(T);
    mobius(100000);
    while (T--) {
        read(n);
        ans = 0;
        int mn = ~0U >> 1, mx = 0;
        memset(cnt, 0, sizeof(cnt));
        for (int i = 1; i <= n; i++)
            read(a[i]), mn = min(a[i], mn), mx = max(a[i], mx), cnt[a[i]]++;
        for (int i = 1; i <= 200000; i++) cnt[i] += cnt[i - 1];
        for (int i = 2; i <= mn; i++) {
            if (!miu[i]) continue;
            LL tmp = 1;
            for (int j = 1; i * j <= 100000; j++)
                tmp = tmp * pow(j, cnt[i * j + i - 1] - cnt[i * j - 1], mod) %
                      mod;
            // j<=100000/i -> i*j<=100000 : TLE -> AC
            ans = (ans - tmp * miu[i] + mod) % mod;
        }
        printf("Case #%d: %lld\n", cas++, ans);
    }
    return 0;
}
```

6.40 基础反演

```
/*
    f(n)= [n|d]g(d)
    g(n)= [n|d]mu[d/n]f(d)
*/
//已知 g ,求 f
for (int i = 1; i <= n; ++i)
    for (int j = i + i; j <= n; j += i) f[i] += f[j];
//已知 f ,求 g
for (int i = n; i >= 1; --i)
    for (int j = i + i; j <= n; j += i) f[i] -= f[j];
/*
    f(n)= [d|n]g(d)
```

```
    g(n)= [d|n]mu[n/d]f(d)= [d|n]mu[d]f(n/d)
*/
//已知 g ,求 f
for (int i = n; i >= 1; --i)
    for (int j = i + i; j <= n; j += i) f[j] += f[i];
//已知 f ,求 g
for (int i = 1; i <= n; ++i)
    for (int j = i + i; j <= n; j += i) f[j] -= f[i];
/*
    Example1:
        gcd(a[i],a[j],a[k],……)
        求子集gcd之和
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 1010, P = 1e8 + 7;
int T, n, x, f[N], pw[N];
int main() {
    for (int i = pw[0] = 1; i < N; i++) pw[i] = pw[i - 1] * 2 % P;
    scanf("%d", &T);
    while (T--) {
        memset(f, 0, sizeof(f));
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            f[x]++;
        }
        n = 1000;
        for (int i = 1; i <= n; i++) {
            for (int j = i + i; j <= n; j += i) f[i] += f[j];
            f[i] = pw[f[i]] - 1;
        }
        for (int i = n; i; i--) {
            for (int j = i + i; j <= n; j += i) f[i] = (f[i] - f[j] + P) % P;
        }
        int ans = 0;
        for (int i = 1; i <= n; i++) ans = (ans + 1ll * i * f[i]) % P;
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example2:
         gcd(a[i],a[j])*(gcd(a[i],a[j])-1)
*/
#include <bits/stdc++.h>
```

```
using namespace std;
const int N = 10010, P = 10007;
int n, x, c[N], f[N];
int main() {
    while (~scanf("%d", &n)) {
        memset(c, 0, sizeof(c));
        memset(f, 0, sizeof(f));
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            c[x]++;
        }
        n = 10000;
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j += i) f[i] += c[j];
            f[i] = f[i] * f[i] % P;
        }
        for (int i = n; i >= 1; --i)
            for (int j = i + i; j <= n; j += i) f[i] -= f[j];
        int ans = 0;
        for (int i = 2; i <= n; i++)
            ans = (ans + f[i] * i % P * (i - 1) % P) % P;
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example3:
            gcd(a[i],a[j],a[k],a[l])
*/
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 10010;
int n, x, c[N];
long long f[N];
int main() {
    while (~scanf("%d", &n)) {
        memset(c, 0, sizeof(c));
        memset(f, 0, sizeof(f));
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            c[x]++;
        }
        n = 10000;
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j += i) f[i] += c[j];
```

```
            if (f[i] > 3)
                f[i] = f[i] * (f[i] - 1) * (f[i] - 2) * (f[i] - 3) / 24;
            else
                f[i] = 0;
        }
        for (int i = n; i >= 1; --i)
            for (int j = i + i; j <= n; j += i) f[i] -= f[j];
        printf("%lld\n", f[1]);
    }
    return 0;
}
```

6.41 莫比乌斯反演

```
/*
    莫比乌斯反演
    f(n)= [n|d]g(d)
    g(n)= [n|d]mu[d/n]f(d)
    ----------------------
    f(n)= [d|n]g(d)
    g(n)= [d|n]mu[n/d]f(d)= [d|n]mu[d]f(n/d)
*/
/*
    Example1:
        (i,a,b) (j,c,d)[gcd(i,j)==k]
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 100010;
bool notp[N];
int prime[N], pnum, mu[N];
void sieve() {
    memset(notp, 0, sizeof(notp));
    notp[0] = notp[1] = mu[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!notp[i]) {
            prime[++pnum] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= pnum && prime[j] * i < N; j++) {
            notp[prime[j] * i] = 1;
            if (i % prime[j] == 0) {
                mu[prime[j] * i] = 0;
                break;
```

```
            }
            mu[prime[j] * i] = -mu[i];
        }
    }
}
ll f[N];
ll solve(int n, int m) {
    if (n > m) swap(n, m);
    ll res = 0;
    for (int i = 1, last; i <= n; i = last + 1) {
        last = min(n / (n / i), m / (m / i));
        res += (f[last] - f[i - 1]) * (n / i) * (m / i);
    }
    return res;
}
int T, a, b, c, d, k;
int main() {
    sieve();
    for (int i = 1; i < N; i++) f[i] = f[i - 1] + mu[i];
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d%d%d", &a, &b, &c, &d, &k);
        printf("%lld\n", solve(b / k, d / k) - solve(b / k, (c - 1) / k) -
                         solve((a - 1) / k, d / k) +
                         solve((a - 1) / k, (c - 1) / k));
    }
    return 0;
}
/*
    Example2:
         [gcd(i,j)素因子个数<=P]
*/
#include <bits/stdc++.h>
using namespace std;
using namespace std;
typedef long long ll;
const int N = 500010;
bool notp[N];
int prime[N], pnum, mu[N], d[N];
void sieve() {
    memset(notp, 0, sizeof(notp));
    notp[0] = notp[1] = mu[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!notp[i]) {
            prime[++pnum] = i;
            mu[i] = -1;
```

```
            d[i] = 1;
        }
        for (int j = 1; j <= pnum && prime[j] * i < N; j++) {
            notp[prime[j] * i] = 1;
            d[prime[j] * i] = d[i] + 1;
            if (i % prime[j] == 0) {
                mu[prime[j] * i] = 0;
                break;
            }
            mu[prime[j] * i] = -mu[i];
        }
    }
}
int f[N][21];
void calF() {
    for (int i = 1; i < N; i++)
        for (int j = i; j < N; j += i) f[j][d[i]] += mu[j / i];
    for (int i = 1; i < N; i++)
        for (int j = 1; j <= 20; j++) f[i][j] += f[i][j - 1];
    for (int i = 1; i < N; i++)
        for (int j = 0; j <= 20; j++) f[i][j] += f[i - 1][j];
}
ll solve(int n, int m, int p) {
    if (n > m) swap(n, m);
    ll res = 0;
    for (int i = 1, last; i <= n; i = last + 1) {
        last = min(n / (n / i), m / (m / i));
        res += (ll)(f[last][p] - f[i - 1][p]) * (n / i) * (m / i);
    }
    return res;
}
int T, n, m, p;
int main() {
    sieve();
    calF();
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d", &n, &m, &p);
        p = min(p, 20);
        printf("%lld\n", solve(n, m, p));
    }
    return 0;
}
```

6.42 容斥原理

```
/*
    容斥原理
*/
/*
    Example
       给出一些数，求取出一些数，当他们的GCD大于0时，将数量乘GCD累加到答案上，求累加和。
    Solution
        我们枚举GCD，统计为其倍数的数字数量，先假定其能组成的集合数为贡献，
        但是我们发现在统计的过程中有多余统计的部分，比如4和8是2的倍数，
        然而它们的GCD等于4，所以我们对于每个集合数的贡献要减去所有其倍数的集合数的贡献，
        倒着容斥即可。
*/
const int N = 1000010, MOD = 1e9 + 7;
int n, ans = 0, w[N], dp[N], pw[N], mx;
int main() {
    scanf("%d", &n);
    for (int i = pw[0] = 1; i <= n; i++) pw[i] = 2 * pw[i - 1] % MOD;
    for (int i = 1, x; i <= n; i++) scanf("%d", &x), mx = max(x, mx), w[x]++;
    for (int i = mx; i > 1; i--) {
        int t = 0;
        for (int j = i; j <= mx; j += i) t += w[j];
        if (!t) continue;
        dp[i] = 1LL * t * pw[t - 1] % MOD;
        for (int j = i + i; j <= mx; j += i)
            dp[i] = (dp[i] - dp[j] + MOD) % MOD;
        ans = (1LL * dp[i] * i + ans) % MOD;
    }
    printf("%d\n", ans);
    return 0;
}
/*
    Example
        m个石子围成一圈，标号为0~m-1，现在有n只青蛙，每只每次跳a[i]个石子，
        问能被青蛙跳到的石子一共有几个
    Solution
        我们发现k*gcd(m,a[i])的位置均可以被跳到，那么我们首先筛出m的约数，
        判断其是否被覆盖到，不考虑重复的情况下，
        每个被覆盖到的约数的贡献为x*((m-1)/x)*((m-1)/x+1)/2,
        但是约数的倍数也为约数的情况被重复计算，因此我们按约数从大到小容斥计算答案。
*/
typedef long long LL;
int T, n, m, p[20010], mark[20010], x, tot;
LL dp[20010];
int main() {
```

```
    scanf("%d", &T);
    for (int Cas = 1; Cas <= T; Cas++) {
        memset(dp, 0, sizeof(dp));
        memset(mark, 0, sizeof(mark));
        scanf("%d%d", &n, &m);
        tot = 0;
        for (int i = 1; i * i <= m; i++) {
            if (m % i == 0) {
                p[++tot] = i;
                if (i * i != m) p[++tot] = m / i;
            }
        }
        sort(p + 1, p + tot + 1);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x);
            int GCD = __gcd(x, m);
            for (int j = 1; j <= tot; j++)
                if (p[j] % GCD == 0) mark[j] = 1;
        }
        LL ans = 0;
        for (int i = tot; i; i--)
            if (mark[i]) {
                int t = (m - 1) / p[i];
                dp[i] = 1LL * t * (t + 1) / 2 * p[i];
                for (int j = i + 1; j <= tot; j++)
                    if (mark[j] && p[j] % p[i] == 0) dp[i] -= dp[j];
                ans = ans + dp[i];
            }
        printf("Case #%d: %lld\n", Cas, ans);
    }
    return 0;
}
/*
    题目大意:
        给出一个数列每个位置可以取到的最大值,
        问这个可以构造多少个数列, 使得他们的最大公约数大于1
    题解:
        我们可以枚举最大公约数k, 对于k来说,
        他对答案的贡献为 [ai/k], 我们将数列中的数字转化为权值数组
        _{i=1}^{100000}[i/k], 对于求解i/k的部分我们可以进行数值分块,
        j*k-1~j*k+k-1的数值除k得到的结果都是相同的, 因此可以直接求这个结果的幂次,
        这时候只要再加一个权值数组的前缀和, 问题就迎刃而解了。
        数值分块计算的复杂度为n+n/2+n/3+n/4+n/5+……+n/n=nlogn。
        对于计算结果, 我们需要进行容斥, 奇数次素数乘的系数为1, 偶数次素数乘的系数为-1,
        对于出现素数幂的合数其系数为0,
        我们发现这个容斥恰好是莫比乌斯函数的相反数, 因此我们取反即可。
```

```
        这有个小小的优化，对于系数为0的情况，我们可以直接跳过，不进行计算。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 200010;
typedef long long LL;
const LL mod = 1000000007;
int T, n, a[N], b[N], cnt[N], cas = 1, p[N];
LL ans = 0;
int tot, miu[N], sum[N], v[N];
void read(int& a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
void mobius(int n) {
    int i, j;
    for (miu[1] = 1, i = 2; i <= n; i++) {
        if (!v[i]) p[tot++] = i, miu[i] = -1;
        for (j = 0; j < tot && i * p[j] <= n; j++) {
            v[i * p[j]] = 1;
            if (i % p[j])
                miu[i * p[j]] = -miu[i];
            else
                break;
        }
    }
    for (i = 1; i < n; i++) sum[i] = sum[i - 1] + miu[i];
}
LL pow(LL a, LL b, LL p) {
    if (b == 0) return 1;
    LL t = 1;
    for (a %= p; b; b >>= 1LL, a = a * a % p)
        if (b & 1LL) t = t * a % p;
    return t;
}
int main() {
    read(T);
    mobius(100000);
    while (T--) {
        read(n);
        ans = 0;
```

```
        int mn = ~0U >> 1, mx = 0;
        memset(cnt, 0, sizeof(cnt));
        for (int i = 1; i <= n; i++)
            read(a[i]), mn = min(a[i], mn), mx = max(a[i], mx), cnt[a[i]]++;
        for (int i = 1; i <= 200000; i++) cnt[i] += cnt[i - 1];
        for (int i = 2; i <= mn; i++) {
            if (!miu[i]) continue;
            LL tmp = 1;
            for (int j = 1; i * j <= 100000; j++)
                tmp = tmp * pow(j, cnt[i * j + i - 1] - cnt[i * j - 1], mod) %
                      mod;
            // j<=100000/i -> i*j<=100000 : TLE -> AC
            ans = (ans - tmp * miu[i] + mod) % mod;
        }
        printf("Case #%d: %lld\n", cas++, ans);
    }
    return 0;
}
```

6.43 杜教筛

```
/*
    杜教筛
    g(1)S(n)=\sum_{i=1}^{n}h(i)-\sum_{d=2}^{n}g(d)S(n/d)
*/
/*
    Example1:
        莫比乌斯函数区段和
*/
const int mod = 1333331;
typedef long long LL;
LL a, b, miu[5000010];
int p[500010], cnt = 0;
bool vis[5000010];
struct HASHMAP {
    int h[mod + 10], cnt, nxt[100010];
    LL st[100010], S[100010];
    void push(LL k, LL v) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return;
        }
        ++cnt;
        nxt[cnt] = h[key];
        h[key] = cnt;
```

```
        S[cnt] = k;
        st[cnt] = v;
    }
    LL ask(LL k) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return st[i];
        }
        return -1;
    }
} H;
void Get_Prime() {
    miu[1] = 1;
    for (int i = 2; i <= 5000000; ++i) {
        if (!vis[i]) {
            p[++cnt] = i;
            miu[i] = -1;
        }
        for (int j = 1; j <= cnt; ++j) {
            if (1LL * p[j] * i > 5000000) break;
            int ip = i * p[j];
            vis[ip] = true;
            if (i % p[j] == 0) break;
            miu[ip] = -miu[i];
        }
    }
    for (int i = 2; i <= 5000000; ++i) miu[i] += miu[i - 1];
}
LL miu_sum(LL n) {
    if (n <= 5000000) return miu[n];
    LL tmp = H.ask(n), la, A = 1;
    if (tmp != -1) return tmp;
    for (LL i = 2; i <= n; i = la + 1) {
        LL now = n / i;
        la = n / now;
        A = A - (la - i + 1) * miu_sum(n / i);
    }
    H.push(n, A);
    return A;
}
int main() {
    scanf("%lld%lld", &a, &b);
    Get_Prime();
    printf("%lld\n", miu_sum(b) - miu_sum(a - 1));
    return 0;
}
```

```
/*
    Example2:
        欧拉函数前缀和
*/
typedef long long LL;
const int mod = 1333331, inv2 = 500000004;
const LL MOD = 1e9 + 7;
LL a, b, miu[5000010], phi[5000010];
int p[500010], cnt = 0, i, tot;
bool v[5000010];
struct HASHMAP {
    int h[mod + 10], cnt, nxt[100010];
    LL st[100010], S[100010];
    void push(LL k, LL v) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return;
        }
        ++cnt;
        nxt[cnt] = h[key];
        h[key] = cnt;
        S[cnt] = k;
        st[cnt] = v;
    }
    LL ask(LL k) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return st[i];
        }
        return -1;
    }
} H;
void Get_Prime() {
    for (miu[1] = phi[1] = 1, i = 2; i <= 5000000; i++) {
        if (!v[i]) p[tot++] = i, miu[i] = -1, phi[i] = i - 1;
        for (int j = 0; i * p[j] <= 5000000 && j < tot; j++) {
            v[i * p[j]] = 1;
            if (i % p[j]) {
                miu[i * p[j]] = -miu[i];
                phi[i * p[j]] = phi[i] * (p[j] - 1);
            } else {
                miu[i * p[j]] = 0;
                phi[i * p[j]] = phi[i] * p[j];
                break;
            }
        }
```

```
    }
    for (int i = 2; i <= 5000000; ++i) phi[i] = (phi[i - 1] + phi[i]) % MOD;
}
LL phi_sum(LL n) {
    if (n <= 5000000) return phi[n];
    LL tmp = H.ask(n), la, A = 0;
    if (tmp != -1) return tmp;
    for (LL i = 2; i <= n; i = la + 1) {
        LL now = n / i;
        la = n / now;
        (A += (la - i + 1) % MOD * phi_sum(n / i) % MOD) %= MOD;
    }
    A = ((n % MOD) * (n % MOD + 1) % MOD * inv2 % MOD - A + MOD) % MOD;
    H.push(n, A);
    return A;
}
int main() {
    scanf("%lld", &a);
    Get_Prime();
    printf("%lld\n", phi_sum(a));
    return 0;
}
/*
    Example3:
        求[1,n][1,n]最大公约数之和
        枚举最大公约数k, 答案为2*(k*phi_sum(n/k))-n*(n+1)/2
        phi_sum利用杜教筛实现
*/
typedef long long LL;
const int mod = 1333331, inv2 = 500000004;
const LL MOD = 1e9 + 7;
LL a, b, n, miu[5000010], phi[5000010];
int p[500010], cnt = 0, i, tot;
bool v[5000010];
struct HASHMAP {
    int h[mod + 10], cnt, nxt[100010];
    LL st[100010], S[100010];
    void push(LL k, LL v) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return;
        }
        ++cnt;
        nxt[cnt] = h[key];
        h[key] = cnt;
        S[cnt] = k;
```

```
        st[cnt] = v;
    }
    LL ask(LL k) {
        int key = k % mod;
        for (int i = h[key]; i; i = nxt[i]) {
            if (S[i] == k) return st[i];
        }
        return -1;
    }
} H;
void Get_Prime() {
    for (miu[1] = phi[1] = 1, i = 2; i <= 5000000; i++) {
        if (!v[i]) p[tot++] = i, miu[i] = -1, phi[i] = i - 1;
        for (int j = 0; i * p[j] <= 5000000 && j < tot; j++) {
            v[i * p[j]] = 1;
            if (i % p[j]) {
                miu[i * p[j]] = -miu[i];
                phi[i * p[j]] = phi[i] * (p[j] - 1);
            } else {
                miu[i * p[j]] = 0;
                phi[i * p[j]] = phi[i] * p[j];
                break;
            }
        }
    }
    for (int i = 2; i <= 5000000; ++i) phi[i] = (phi[i - 1] + phi[i]) % MOD;
}
LL phi_sum(LL n) {
    if (n <= 5000000) return phi[n];
    LL tmp = H.ask(n), la, A = 0;
    if (tmp != -1) return tmp;
    for (LL i = 2; i <= n; i = la + 1) {
        LL now = n / i;
        la = n / now;
        (A += (la - i + 1) % MOD * phi_sum(n / i) % MOD) %= MOD;
    }
    A = ((n % MOD) * (n % MOD + 1) % MOD * inv2 % MOD - A + MOD) % MOD;
    H.push(n, A);
    return A;
}
int main() {
    scanf("%lld", &n);
    Get_Prime();
    LL la, ans = 0;
    for (LL i = 1; i <= n; i = la + 1) {
        LL now = n / i;
```

```
        la = n / now;
        ans = (ans + (i + la) % MOD * (la - i + 1) % MOD * inv2 % MOD *
                     phi_sum(now) % MOD) %
              MOD;
    }
    ans = ans * 2 % MOD;
    LL k = n % MOD;
    ans = (ans - k * (k + 1) % MOD * inv2 % MOD + MOD) % MOD;
    printf("%lld\n", ans);
    return 0;
}
/*
    Example4:
        求[1,n][1,m]两两LCM的和
*/
namespace Sum_LCM {
int tot, p[N], miu[N], v[N];
LL sum[N];
void mobius(int n) {
    int i, j;
    for (miu[1] = 1, i = 2; i <= n; i++) {
        if (!v[i]) p[tot++] = i, miu[i] = -1;
        for (j = 0; j < tot && i * p[j] <= n; j++) {
            v[i * p[j]] = 1;
            if (i % p[j])
                miu[i * p[j]] = -miu[i];
            else
                break;
        }
    }
    for (i = 1; i <= n; i++)
        sum[i] = (sum[i - 1] + (1LL * miu[i] * i % mod * i % mod)) % mod;
}
LL Sum(LL x, LL y) {
    return (((x * (x + 1) / 2) % mod) * ((y * (y + 1) / 2) % mod) % mod);
}
LL F(int n, int m) {
    LL t = 0;
    if (n > m) swap(n, m);
    for (int i = 1, j = 0; i <= n; i = j + 1) {
        j = min(n / (n / i), m / (m / i));
        t = (t + (sum[j] - sum[i - 1] + mod) % mod * Sum((LL)n / i, (LL)m / i) %
                    mod) %
            mod;
    }
    return t;
```

```
}
LL Cal_Sum(int n, int m) {
    if (n > m) swap(n, m);
    mobius(n);
    LL ans = 0;
    for (int i = 1, j = 0; i <= n; i = j + 1) {
        j = min(n / (n / i), m / (m / i));
        ans = (ans +
              1LL * (i + j) * (j - i + 1) / 2 % mod * F(n / i, m / i) % mod) %
             mod;
    }
    return ans;
}
} // namespace Sum_LCM
```

6.44 Min25 筛

```
/*
    Min25筛
    S(x,y): F(i)[i<=x&&i的最小质因子不小于P_y]
    pi>n时S(n,i)=0
    pi<=n时S(n,i)=\sum_{j>=i}^{(p_j)^2<=n}\sum_{c=1}^{(p_j)^{c+1}}S(n/p_i,j+1)*F(p_i^c)+F(p_i^{c+1})
    +\sum_{x是质数且x<=n}F(x)-\sum_{j=1}^{i-1}F(p_i)
    F(i)的前缀和答案为S(n,1)+F(1)
*/
/*
    Example1:
        欧拉函数前缀和&&莫比乌斯函数前缀和
*/
#include <bits/stdc++.h>
typedef long long ll;
const int N = 500000 << 1; // sqrt(n)*2
int n, m, Sqr, cnt, P[N >> 2], h[N], w[N];
ll sp[N], g[N];
inline int ID(int x) { return x <= Sqr ? x : m - n / x + 1; }
// g:前缀质数和
// h:前缀质数个数
void Init(int n) {
    Sqr = sqrt(n);
    cnt = m = 0;
    for (ll i = 1; i <= n; i = w[m] + 1ll)
        w[++m] = n / (n / i), g[m] = (1ll * w[m] * ((ll)w[m] + 1) >> 1) - 1,
        h[m] = w[m] - 1;
    for (int i = 2; i <= Sqr; ++i)
        if (h[i] != h[i - 1]) {
```

```
            P[++cnt] = i, sp[cnt] = sp[cnt - 1] + i;
            int lim = i * i;
            for (int j = m; lim <= w[j]; --j) {
                int k = ID(w[j] / i);
                g[j] -= 1ll * i * (g[k] - sp[cnt - 1]);
                h[j] -= h[k] - cnt + 1;
            }
        }
}
// 欧拉函数前缀和
ll S_Phi(int x, int y) {
    if (x <= 1 || P[y] > x) return 0;
    ll res = g[ID(x)] - h[ID(x)] - sp[y - 1] + y - 1; // g-h
    for (int i = y; i <= cnt && P[i] * P[i] <= x; ++i)
        for (ll p = P[i], p1 = p, t = p - 1; 1ll * p1 * p <= x; p1 *= p, t *= p)
            res += 1ll * (S_Phi(x / p1, i + 1) + p) * t;
    return res;
}
// 莫比乌斯函数前缀和
int S_Mu(int x, int y) {
    if (x <= 1 || P[y] > x) return 0;
    int res = -h[ID(x)] + y - 1; // h
    for (int i = y; i <= cnt && P[i] * P[i] <= x; ++i)
        res -= S_Mu(x / P[i], i + 1);
    return res;
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        if (!n)
            puts("0 0");
        else {
            Init(n);
            printf("%lld %d\n", S_Phi(n, 1) + 1, S_Mu(n, 1) + 1);
        }
    }
    return 0;
}
/*
    Example2:
        F(1)=1
        F(p^c)=p xor c(p的c次的函数值为p和c的异或值)
        F(ab)=F(a)F(b) gcd(a,b)==1
        求F前缀和(n<=10^10)
```

```
    Solution:
        除了2之外, p xor 1等于p-1,
        因此我们可以用质数常函数前缀g和质数前缀个数函数h
        来计算 _{x是质数且x<=n}F(x)
*/
#include <bits/stdc++.h>
typedef long long ll;
const int N = 100000 << 1 | 1; // sqrt(n)*2
const int P = 1e9 + 7;
int m, cnt, pri[N >> 2], h[N];
ll n, Sqr, sp[N], g[N], w[N];
inline int ID(ll x) { return x <= Sqr ? x : m - n / x + 1; }
// g:前缀质数和
// h:前缀质数个数
void Init(ll n) {
    Sqr = sqrt(n);
    cnt = m = 0;
    for (ll i = 1; i <= n; i = w[m] + 1) {
        w[++m] = n / (n / i);
        ll t = w[m] % P;
        g[m] = (t * (t + 1) >> 1) % P - 1;
        h[m] = w[m] - 1;
    }
    for (int i = 2; i <= Sqr; ++i){
        if (h[i] != h[i - 1]) {
            pri[++cnt] = i, sp[cnt] = sp[cnt - 1] + i;
            ll lim = 1ll * i * i;
            for (int j = m; lim <= w[j]; --j) {
                int k = ID(w[j] / i);
                g[j] -= i * (g[k] - sp[cnt - 1]);
                h[j] -= h[k] - cnt + 1;
            }
        }
        g[i] %= P;
        }
}
ll S(ll x, int y) {
    if (x <= 1 || pri[y] > x) return 0;
    ll res = g[ID(x)] - g[ID(pri[y - 1])] + P;
    for (int i = y; i <= cnt && 1ll * pri[i] * pri[i] <= x; ++i) {
        for (ll p = pri[i], p1 = p, j = 1; p1 * p <= x; p1 *= p, ++j)
            res += S(x / p1, i + 1) * (pri[i] ^ j) + (pri[i] ^ (j + 1));
    }
    return res % P;
}
int main() {
```

```
    scanf("%lld", &n);
    Init(n);
    for (int i = 1; i <= m; ++i) g[i] = (g[i] - h[i] + (i > 1) * 2 + P) % P;
    printf("%lld\n", (S(n, 1) + 1) % P);
    return 0;
}
/*
    Example3:
        对任意p为素数,
            p%4==1时有f(p^c)=3*c+1
            p%4!=1时f(p^c)=1
        对于非质数有
            f(a*b)=f(a)*f(b) gcd(a,b)==1
    Solution:
        处理g[n][r]表示前缀%4==r的质数个数, 可以通过Min25预处理
        设G[n]等于4*g[n][1]+g[n][3], 对于唯一素数2特殊处理, n>1时G[n]++
        G[n]= F(x)_x为小于n的质数, G[pri[i-1]]= _{j=1}^{i-1}F(p_i)
        套Min25得到f前缀和即可
*/
#include <bits/stdc++.h>
typedef long long ll;
const int N = 50000 << 1 | 1; // sqrt(n)*2
int n, m, Sqr, pnum, pri[N >> 2], w[N];
ll g[N][4], sp[N][4], G[N];
inline int ID(int x) { return x <= Sqr ? x : m - n / x + 1; }
int F(int p, int k) {
    if (p % 4 == 1) return 3 * k + 1;
    return 1;
}
bool notp[N];
void sieve() {
    memset(notp, 0, sizeof(notp));
    notp[0] = notp[1] = 1;
    pnum = 0;
    for (int i = 2; i < N; i++) {
        if (!notp[i]) {
            pri[++pnum] = i;
            for (int r = 0; r < 4; ++r)
                sp[pnum][r] = sp[pnum - 1][r] + (i % 4 == r);
        }
        for (int j = 1; j <= pnum && pri[j] * i < N; j++) {
            notp[pri[j] * i] = 1;
            if (i % pri[j] == 0) break;
        }
    }
}
```

```
void Init(int n) {
    Sqr = sqrt(n);
    m = 0;
    ll j;
    for (ll i = 1; i <= n; i = w[m] + 1ll) {
        w[++m] = n / (n / i);
        for (int r = 0; r < 4; r++) {
            if (r == 0)
                g[m][r] = w[m] / 4;
            else
                g[m][r] = w[m] / 4 + (w[m] % 4 >= r);
            if (r == 1) g[m][r]--;
        }
    }
    for (int i = 1; i <= pnum; ++i) {
        ll lim = 1ll * pri[i] * pri[i];
        for (int j = m; lim <= w[j]; --j) {
            int k = ID(w[j] / pri[i]);
            for (int r = 0; r < 4; ++r) {
                g[j][r * pri[i] % 4] -= g[k][r] - sp[i - 1][r];
            }
        }
    }
    for (int i = 1; i <= m; i++) G[i] = g[i][1] * 4 + g[i][3] + (i > 1);
}
ll S(ll x, int y) {
    if (x <= 1 || pri[y] > x) return 0;
    ll res = G[ID(x)] - G[pri[y - 1]];
    for (int i = y; i <= pnum && 1ll * pri[i] * pri[i] <= x; ++i) {
        for (ll p = pri[i], p1 = p, j = 1; p1 * p <= x; p1 *= p, ++j)
            res += S(x / p1, i + 1) * F(pri[i], j) + F(pri[i], j + 1);
    }
    return res;
}
int T;
int main() {
    sieve();
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        Init(n);
        printf("%lld\n", S(n, 1) + 1);
    }
    return 0;
}
```

6.45 BM 算法

```
/*
    BM算法计算线性递推
    放入数列前2*k项(如果确定为k阶递推的话, 不确定尽量多放)
    返回值为数列第n项
    Linear_seq::gao(Linear_seq::VI{1,5,11,36,95,281,781,2245,6336,18061},n-1)
*/
namespace Linear_seq {
typedef long long LL;
const LL mod = 1000000007;
#define SZ(x) ((int)(x).size())
#define rep(i, a, n) for (int i = a; i < n; i++)
#define pb push_back
typedef vector<int> VI;
typedef pair<int, int> PII;
const int N = 10010;
LL res[N], base[N], _c[N], _md[N];
vector<int> Md;
LL powmod(LL a, LL b) {
    LL res = 1;
    a %= mod;
    for (; b; b >>= 1) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
    }
    return res;
}
void mul(LL *a, LL *b, int k) {
    rep(i, 0, k + k) _c[i] = 0;
    rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] =
        (_c[i + j] + a[i] * b[j]) % mod;
    for (int i = k + k - 1; i >= k; i--)
        if (_c[i])
            rep(j, 0, SZ(Md)) _c[i - k + Md[j]] =
                (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) % mod;
    rep(i, 0, k) a[i] = _c[i];
}
int solve(LL n, VI a, VI b) {
    LL ans = 0, pnt = 0;
    int k = SZ(a);
    rep(i, 0, k) _md[k - 1 - i] = -a[i];
    _md[k] = 1;
    Md.clear();
    rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
    rep(i, 0, k) res[i] = base[i] = 0;
```

```
    res[0] = 1;
    while ((1LL << pnt) <= n) pnt++;
    for (int p = pnt; p >= 0; p--) {
        mul(res, res, k);
        if ((n >> p) & 1) {
            for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i];
            res[0] = 0;
            rep(j, 0, SZ(Md)) res[Md[j]] =
                (res[Md[j]] - res[k] * _md[Md[j]]) % mod;
        }
    }
    rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
    if (ans < 0) ans += mod;
    return ans;
}
VI BM(VI s) {
    VI C(1, 1), B(1, 1);
    int L = 0, m = 1, b = 1;
    rep(n, 0, SZ(s)) {
        LL d = 0;
        rep(i, 0, L + 1) d = (d + (LL)C[i] * s[n - i]) % mod;
        if (d == 0)
            ++m;
        else if (2 * L <= n) {
            VI T = C;
            LL c = mod - d * powmod(b, mod - 2) % mod;
            while (SZ(C) < SZ(B) + m) C.pb(0);
            rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
            L = n + 1 - L;
            B = T;
            b = d;
            m = 1;
        } else {
            LL c = mod - d * powmod(b, mod - 2) % mod;
            while (SZ(C) < SZ(B) + m) C.pb(0);
            rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
            ++m;
        }
    }
    return C;
}
int gao(VI a, LL n) {
    VI c = BM(a);
    c.erase(c.begin());
    rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
    return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
```

```
}
}; // namespace Linear_seq
```

6.46 ReedsSloane 算法

```
/*
    ReedsSloane算法
    求解线性递推，模非质数
    Example: 求 (fib[i]^m)%P, P=1e9
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ul;
#define mp make_pair
#define pb push_back
#define sqr(x) ((x) * (x))
#define sz(x) (int)x.size()
#define all(x) x.begin(), x.end()
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
const int mn = 105;
const int mod = 1000000000;
int n, m;
// given first m items init[0..m-1] and coefficents trans[0..m-1] or
// given first 2 *m items init[0..2m-1], it will compute trans[0..m-1]
// for you. trans[0..m] should be given as that
//     init[m] = sum_{i=0}^{m-1} init[i] * trans[i]
struct LinearRecurrence {
    using int64 = long long;
    using vec = std::vector<int64>;

    static void extand(vec &a, size_t d, int64 value = 0) {
        if (d <= a.size()) return;
        a.resize(d, value);
    }

    static vec BerlekampMassey(const vec &s, int64 mod) {
        std::function<int64(int64)> inverse = [&](int64 a) {
            return a == 1 ? 1 : (int64)(mod - mod / a) * inverse(mod % a) % mod;
        };
        vec A = {1}, B = {1};
        int64 b = s[0];
        for (size_t i = 1, m = 1; i < s.size(); ++i, m++) {
            int64 d = 0;
```

```
        for (size_t j = 0; j < A.size(); ++j) {
            d += A[j] * s[i - j] % mod;
        }
        if (!(d %= mod)) continue;
        if (2 * (A.size() - 1) <= i) {
            auto temp = A;
            extand(A, B.size() + m);
            int64 coef = d * inverse(b) % mod;
            for (size_t j = 0; j < B.size(); ++j) {
                A[j + m] -= coef * B[j] % mod;
                if (A[j + m] < 0) A[j + m] += mod;
            }
            B = temp, b = d, m = 0;
        } else {
            extand(A, B.size() + m);
            int64 coef = d * inverse(b) % mod;
            for (size_t j = 0; j < B.size(); ++j) {
                A[j + m] -= coef * B[j] % mod;
                if (A[j + m] < 0) A[j + m] += mod;
            }
        }
    }
    return A;
}

static void exgcd(int64 a, int64 b, int64 &g, int64 &x, int64 &y) {
    if (!b)
        x = 1, y = 0, g = a;
    else {
        exgcd(b, a % b, g, y, x);
        y -= x * (a / b);
    }
}

static int64 crt(const vec &c, const vec &m) {
    int n = c.size();
    int64 M = 1, ans = 0;
    for (int i = 0; i < n; ++i) M *= m[i];
    for (int i = 0; i < n; ++i) {
        int64 x, y, g, tm = M / m[i];
        exgcd(tm, m[i], g, x, y);
        ans = (ans + tm * x * c[i] % M) % M;
    }
    return (ans + M) % M;
}
```

```
static vec ReedsSloane(const vec &s, int64 mod) {
    auto inverse = [](int64 a, int64 m) {
        int64 d, x, y;
        exgcd(a, m, d, x, y);
        return d == 1 ? (x % m + m) % m : -1;
    };
    auto L = [](const vec &a, const vec &b) {
        int da = (a.size() > 1 || (a.size() == 1 && a[0])) ? a.size() - 1
                                                           : -1000;
        int db = (b.size() > 1 || (b.size() == 1 && b[0])) ? b.size() - 1
                                                           : -1000;
        return std::max(da, db + 1);
    };
    auto prime_power = [&](const vec &s, int64 mod, int64 p, int64 e) {
        // linear feedback shift register mod p^e, p is prime
        std::vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
        vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
        ;
        pw[0] = 1;
        for (int i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
        for (int64 i = 0; i < e; ++i) {
            a[i] = {pw[i]}, an[i] = {pw[i]};
            b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
            t[i] = s[0] * pw[i] % mod;
            if (t[i] == 0) {
                t[i] = 1, u[i] = e;
            } else {
                for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i])
                    ;
            }
        }
        for (size_t k = 1; k < s.size(); ++k) {
            for (int g = 0; g < e; ++g) {
                if (L(an[g], bn[g]) > L(a[g], b[g])) {
                    ao[g] = a[e - 1 - u[g]];
                    bo[g] = b[e - 1 - u[g]];
                    to[g] = t[e - 1 - u[g]];
                    uo[g] = u[e - 1 - u[g]];
                    r[g] = k - 1;
                }
            }
            a = an, b = bn;
            for (int o = 0; o < e; ++o) {
                int64 d = 0;
                for (size_t i = 0; i < a[o].size() && i <= k; ++i) {
                    d = (d + a[o][i] * s[k - i]) % mod;
```

```
            }
            if (d == 0) {
                t[o] = 1, u[o] = e;
            } else {
                for (u[o] = 0, t[o] = d; t[o] % p == 0;
                     t[o] /= p, ++u[o])
                    ;
                int g = e - 1 - u[o];
                if (L(a[g], b[g]) == 0) {
                    extand(bn[o], k + 1);
                    bn[o][k] = (bn[o][k] + d) % mod;
                } else {
                    int64 coef = t[o] * inverse(to[g], mod) % mod *
                                 pw[u[o] - uo[g]] % mod;
                    int m = k - r[g];
                    extand(an[o], ao[g].size() + m);
                    extand(bn[o], bo[g].size() + m);
                    for (size_t i = 0; i < ao[g].size(); ++i) {
                        an[o][i + m] -= coef * ao[g][i] % mod;
                        if (an[o][i + m] < 0) an[o][i + m] += mod;
                    }
                    while (an[o].size() && an[o].back() == 0)
                        an[o].pop_back();
                    for (size_t i = 0; i < bo[g].size(); ++i) {
                        bn[o][i + m] -= coef * bo[g][i] % mod;
                        if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
                    }
                    while (bn[o].size() && bn[o].back() == 0)
                        bn[o].pop_back();
                }
            }
        }
    }
    return std::make_pair(an[0], bn[0]);
};
std::vector<std::tuple<int64, int64, int>> fac;
for (int64 i = 2; i * i <= mod; ++i)
    if (mod % i == 0) {
        int64 cnt = 0, pw = 1;
        while (mod % i == 0) mod /= i, ++cnt, pw *= i;
        fac.emplace_back(pw, i, cnt);
    }
if (mod > 1) fac.emplace_back(mod, mod, 1);
std::vector<vec> as;
size_t n = 0;
for (auto &&x : fac) {
```

```
            int64 mod, p, e;
            vec a, b;
            std::tie(mod, p, e) = x;
            auto ss = s;
            for (auto &&x : ss) x %= mod;
            std::tie(a, b) = prime_power(ss, mod, p, e);
            as.emplace_back(a);
            n = std::max(n, a.size());
        }
        vec a(n), c(as.size()), m(as.size());
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < as.size(); ++j) {
                m[j] = std::get<0>(fac[j]);
                c[j] = i < as[j].size() ? as[j][i] : 0;
            }
            a[i] = crt(c, m);
        }
        return a;
    }
    LinearRecurrence(const vec &s, const vec &c, int64 mod)
        : init(s), trans(c), mod(mod), m(s.size()) {}
    LinearRecurrence(const vec &s, int64 mod, bool is_prime = true) : mod(mod) {
        vec A;
        if (is_prime)
            A = BerlekampMassey(s, mod);
        else
            A = ReedsSloane(s, mod);
        if (A.empty()) A = {0};
        m = A.size() - 1;
        trans.resize(m);
        for (int i = 0; i < m; ++i) {
            trans[i] = (mod - A[i + 1]) % mod;
        }
        std::reverse(trans.begin(), trans.end());
        init = {s.begin(), s.begin() + m};
    }
    int64 calc(int64 n) {
        if (mod == 1) return 0;
        if (n < m) return init[n];
        vec v(m), u(m << 1);
        int msk = !!n;
        for (int64 m = n; m > 1; m >>= 1) msk <<= 1;
        v[0] = 1 % mod;
        for (int x = 0; msk; msk >>= 1, x <<= 1) {
            std::fill_n(u.begin(), m * 2, 0);
            x |= !!(n & msk);
```

```
            if (x < m)
                u[x] = 1 % mod;
            else { // can be optimized by fft/ntt
                for (int i = 0; i < m; ++i) {
                    for (int j = 0, t = i + (x & 1); j < m; ++j, ++t) {
                        u[t] = (u[t] + v[i] * v[j]) % mod;
                    }
                }
                for (int i = m * 2 - 1; i >= m; --i) {
                    for (int j = 0, t = i - m; j < m; ++j, ++t) {
                        u[t] = (u[t] + trans[j] * u[i]) % mod;
                    }
                }
            }
            v = {u.begin(), u.begin() + m};
        }
        int64 ret = 0;
        for (int i = 0; i < m; ++i) {
            ret = (ret + v[i] * init[i]) % mod;
        }
        return ret;
    }

    vec init, trans;
    int64 mod;
    int m;
};
ll f[25005];
ll powmod(ll a, ll b, ll P) {
    ll t = 1;
    for (; b; b >>= 1, a = a * a % P)
        if (b & 1) t = t * a % P;
    return t;
}
int main() {
    scanf("%d%d", &n, &m);
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i <= 25000; i++) {
        f[i] = (f[i - 1] + f[i - 2]) % mod;
    }
    vector<ll> v;
    ll S = 0;
    for (int i = 0; i < 2000; i++) {
        S = (S + powmod(f[i], m, mod)) % mod;
        v.pb(S);
```

```
    }
    LinearRecurrence solver(v, mod, false);
    printf("%lld\n", solver.calc(n));

    return 0;
}
```

6.47 Linear-Rec

```
/*
    Linear_Rec
    n为矩阵的阶,k为需要计算的数列第k项
    比如An=2*An-1+An-2阶为2
    参数:
        以An=2*An-1+An-2为例
        基础数列first{1,1},转移数列trans{2,1}
    测试正确性: Calc(3)=3,Calc(10007)=71480733 (P=1e9+7)
    O(n*n*log2(k))
*/
struct LinearRec {
    int n;
    const int LOG = 31, P = 1000000007;
    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] += 1LL * a[i] * b[j] % P) >= P)
                    result[i + j] -= P;
            }
        }
        for (int i = 2 * n; i > n; i--) {
            for (int j = 0; j < n; j++) {
                if ((result[i - 1 - j] += 1LL * result[i] * trans[j] % P) >= P)
                    result[i - 1 - j] -= P;
            }
            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 += 1LL * a[i + 1] * first[i] % P) >= P) ret -= P;
        }
        return ret;
    }
};
/*
    Example:
        a[1]=a[2]=⋯=a[n]=1
        a[i]=a[i-1]+...+a[i-n](i>n)
        求第k项
*/
int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    vector<int> a(n, 1);
    LinearRec Rec(a, a);
    printf("%d\n", Rec.Calc(k));
    return 0;
}
```

6.48 Linear-Rec-Spc

```
/*
    Linear_Rec
    a[n+k]=(b[1]&a[n+k-1])^(b[2]&a[n+k-2])^⋯^(b[k]&a[n])
    求给定长度为n的a和b，求第k项a
    测试正确性:
    --------------
    Input
        5 100
        2345678901 1001001001 3333333333 3141592653 1234567890
```

```
        2147483648 2147483647 4294967295 4294967294 3434343434
    Output
        1067078691
    ---------------
    O(n*n*log2(k))
*/
struct LinearRec {
    int n;
    const unsigned LOG = 31;
    vector<unsigned> first, trans;
    vector<vector<unsigned> > bin;
    vector<unsigned> add(vector<unsigned> &a, vector<unsigned> &b) {
        vector<unsigned> result(n * 2 + 1, 0);
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= n; j++) result[i + j] ^= a[i] & b[j];
        }
        for (int i = 2 * n; i > n; i--) {
            for (int j = 0; j < n; j++)
                result[i - 1 - j] ^= result[i] & trans[j];
            result[i] = 0;
        }
        result.erase(result.begin() + n + 1, result.end());
        return result;
    }
    LinearRec(vector<unsigned> &first, vector<unsigned> &trans)
        : first(first), trans(trans) {
        n = first.size();
        vector<unsigned> a(n + 1, 0);
        a[1] = ~0u;
        bin.push_back(a);
        for (int i = 1; i < LOG; i++) {
            bin.push_back(add(bin[i - 1], bin[i - 1]));
        }
    }
    unsigned calc(int k) {
        vector<unsigned> a(n + 1, 0);
        a[0] = ~0u;
        for (int i = 0; i < LOG; i++) {
            if (k >> i & 1) a = add(a, bin[i]);
        }
        unsigned ret = 0;
        for (int i = 0; i < n; i++) ret = ret ^ (a[i + 1] & first[i]);
        return ret;
    }
};
int main() {
```

```
    int n, k;
    scanf("%d%d", &n, &k);
    vector<unsigned> a(n), b(n);
    for (int i = 0; i < n; i++) scanf("%u", &a[i]);
    for (int i = 0; i < n; i++) scanf("%u", &b[i]);
    LinearRec f(a, b);
    printf("%u\n", f.calc(k));
    return 0;
}
```

6.49 特征多项式加速递推

```
/*
    特征多项式+快速幂加速递推
    a[n]= {1,m}_c[m-i]a[n-i],给定a[0],a[1],...,a[m-1],求a[n]
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long ll;
const int M = 2000, P = 1000000007;
int n, m, i, j, x, w, b, t, a[M], c[M], v[M], u[M << 1], ans;
int main() {
    scanf("%d%d", &n, &m);
    for (i = m - 1; ~i; i--) scanf("%d", &c[i]), c[i] = (c[i] % P + P) % P;
    for (i = 0; i < m; i++) scanf("%d", &a[i]), a[i] = (a[i] % P + P) % P;
    for (i = 0; i < m; i++) v[i] = 1;
    for (w = !!n, i = n; i > 1; i >>= 1) w <<= 1;
    for (x = 0; w; copy(u, u + m, v), w >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0), b = !!(n & w), x |= b;
        if (x < m)
            u[x] = 1;
        else {
            for (i = 0; i < m; i++)
                for (j = 0, t = i + b; j < m; j++, t++)
                    u[t] = ((ll)v[i] * v[j] + u[t]) % P;
            for (i = (m << 1) - 1; i >= m; i--)
                for (j = 0, t = i - m; j < m; j++, t++)
                    u[t] = ((ll)c[j] * u[i] + u[t]) % P;
        }
    }
    for (i = 0; i < m; i++) ans = ((ll)v[i] * a[i] + ans) % P;
    printf("%d", (ans + P) % P);
    return 0;
}
```

```
/*
    Example
    f[i]=f[i-1]-f[i-2]
    给出前两项，求n项
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long ll;
const int M = 2000, P = 1000000007;
int n, m, i, j, x, w, b, t, a[M], c[M], v[M], u[M << 1], ans;
int main() {
    c[1] = 1;
    c[0] = P - 1;
    m = 2; //自己填写参数的时候注意也要对P取模
    for (i = 0; i < m; i++) scanf("%d", &a[i]), a[i] = (a[i] % P + P) % P;
    for (i = 0; i < m; i++) v[i] = 1;
    scanf("%d", &n);
    n--;
    if (n < 2) { //小于给出项数的特判
        printf("%d\n", a[n]);
        return 0;
    }
    for (w = !!n, i = n; i > 1; i >>= 1) w <<= 1;
    for (x = 0; w; copy(u, u + m, v), w >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0), b = !!(n & w), x |= b;
        if (x < m)
            u[x] = 1;
        else {
            for (i = 0; i < m; i++)
                for (j = 0, t = i + b; j < m; j++, t++)
                    u[t] = ((ll)v[i] * v[j] + u[t]) % P;
            for (i = (m << 1) - 1; i >= m; i--)
                for (j = 0, t = i - m; j < m; j++, t++)
                    u[t] = ((ll)c[j] * u[i] + u[t]) % P;
        }
    }
    for (i = 0; i < m; i++) ans = ((ll)v[i] * a[i] + ans) % P;
    printf("%d\n", (ans + P) % P);
    return 0;
}
```

6.50 矩阵乘法

```
/*
```

```
    矩阵乘法
*/
typedef long long LL;
LL P = 100000LL;
struct mat {
    int n;
    LL num[110][110];
    void init0(int t) {
        n = t;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) num[i][j] = 0;
    }
    void init1(int t) {
        n = t;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i != j)
                    num[i][j] = 0;
                else
                    num[i][j] = 1;
    }
    mat operator=(const struct mat p) {
        n = p.n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) num[i][j] = p.num[i][j];
    }
    mat operator*(const struct mat p) const {
        struct mat ans;
        ans.init0(n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                    ans.num[i][j] =
                        (ans.num[i][j] + num[i][k] * p.num[k][j]) % P;
        return ans;
    }
    mat operator^(int t) const {
        struct mat ans, now;
        ans.init1(n);
        now.n = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) now.num[i][j] = num[i][j];
        while (t > 0) {
            if (t & 1) ans = ans * now;
            now = now * now;
            t >>= 1;
```

```
        }
        return ans;
    }
}
```

6.51 Matrix-Tree 定理

```
/*
    Matrix-Tree定理
    1、G的度数矩阵D[G]是一个n*n的矩阵，并且满足：当i j时,dij=0;
        当i=j时，dij等于vi的度数。
    2、G的邻接矩阵A[G]也是一个n*n的矩阵，
        并且满足：如果vi、vj之间有边直接相连，则aij=1，否则为0。
        我们定义G的Kirchhoff矩阵(也称为拉普拉斯算子)C[G]为C[G]=D[G]-A[G]，
        则Matrix-Tree定理可以描述为：
            G的所有不同的生成树的个数等于
            其Kirchhoff矩阵C[G]任何一个n-1阶余子式的行列式的绝对值。
            所谓n-1阶余子式，就是对于r(1 r n)，
            将C[G]的第r行、第r列同时去掉后得到的新矩阵，用Cr[G]表示。
    Example:
        给出一张格子图，.表示房间，一开始房间之间都有墙壁相连
        求打通一些墙壁使得所有房间唯一路径，求方案数
*/
typedef long long LL;
#define rep(i, n, m) for (int i = n; i <= m; i++)
const LL mod = 1000000000;
int dx[] = {0, 0, 1, -1}, dy[] = {1, -1, 0, 0};
int n, m, id;
char mp[110][110];
LL a[110][110], p[110][110];
int det(int n) {
    LL ans = 1, f = 1;
    rep(i, 1, n) rep(j, 1, n) a[i][j] = (a[i][j] + mod) % mod;
    rep(i, 1, n) {
        rep(j, i + 1, n) {
            LL A = a[i][i], B = a[j][i];
            while (B != 0) {
                LL t = A / B;
                A %= B;
                swap(A, B);
                for (int k = 1; k <= n; k++)
                    a[i][k] = (a[i][k] - t * a[j][k] % mod + mod) % mod;
                for (int k = 1; k <= n; k++) swap(a[i][k], a[j][k]);
                f = -f;
            }
```

```
        }
        if (!a[i][i]) return 0;
        ans = ans * a[i][i] % mod;
    }
    if (f == -1) return (mod - ans) % mod;
    return ans;
}
bool check(int x, int y) {
    if (x < 1 || y < 1 || x > n || y > m || mp[x][y] != '.') return 0;
    return 1;
}
int main() {
    scanf("%d%d", &n, &m);
    rep(i, 1, n) scanf("%s", mp[i] + 1);
    rep(i, 1, n) rep(j, 1, m) if (mp[i][j] == '.') p[i][j] = ++id;
    rep(i, 1, n) rep(j, 1, m) if (mp[i][j] == '.') {
        rep(k, 0, 4) {
            int x = i + dx[k], y = j + dy[k];
            if (check(x, y)) {
                int u = p[i][j], v = p[x][y];
                a[u][u]++;
                a[u][v]--;
            }
        }
    }
    printf("%d\n", det(id - 1)); // n-1阶余子式
    return 0;
}
```

6.52 约瑟夫问题

```
/*
    约瑟夫问题
*/
int Josephus(int n, int k) { // Number(0-n-1)
    if (k == 1) return n - 1;
    if (n == 1) return 0;
    int ret;
    if (n < k) {
        ret = 0;
        for (int i = 2; i <= n; i++) ret = (ret + k) % i;
        return ret;
    }
    ret = Josephus(n - n / k, k);
    if (ret < n % k)
```

```
        ret = ret - n % k + n;
    else
        ret = ret - n % k + (ret - n % k) / (k - 1);
    return ret;
}
```

6.53 斐波那契数列

```
/*
    斐波那契数列
    Fib(n)=Fib(n-1)+Fib(n-2)
    S(n)=SumFib
    S(n)+1=Fib(n+2)
    gcd(Fib(n),Fib(m))=Fib(gcd(n,m))
*/
/*
    log求fib
    x为第n位fib
*/
void fib(LL n, LL& x, LL& y) {
    if (n == 0) {
        x = 0;
        y = 1;
        return;
    }
    if (n & 1) {
        fib(n - 1, y, x);
        y = (y + x) % mod;
    } else {
        LL a, b;
        fib(n >> 1, a, b);
        y = (a * a + b * b) % mod;
        x = (a * b + a * (b - a + mod)) % mod;
    }
}
/*
    log求fib
    记忆化版本
*/
typedef long long LL;
const LL M = 1e9 + 7;
map<LL, LL> Fib;
LL fib(LL n) {
    LL k = n / 2;
    if (Fib.count(n)) return Fib[n];
```

```
    if (n % 2 == 0)
        return Fib[n] = (fib(k) * fib(k) + fib(k - 1) * fib(k - 1)) % M;
    else
        return Fib[n] = (fib(k) * fib(k + 1) + fib(k - 1) * fib(k)) % M;
}
int main() {
    LL n;
    Fib[0] = Fib[1] = 1;
    scanf("%lld", &n);
    printf("%lld\n", fib(n));
    return 0;
}
/*
    判断一个数是否是斐波那契数
*/
bool isSquare(long long n) { /* */
}
bool isFibonacci(int n) {
    return n >= 0 && isSquare(5 * n * n + 4) || isSquare(5 * n * n - 4);
}
```

6.54 广义斐波那契数列循环节

```
/*
    Problem:
        已知x_0, x_1
        x_i=a*x_{i-1}+b*x_{i-2}
        求x_n, n<=10^(10^6)
    Solution:
        广义斐波那契数列存在模数循环节
        为(p+1)*(p-1)*p^(c-1)的乘积，其中p为模数素因子
        将n对循环节取模，然后特征根加速递推即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int M = 3, N = 1000000 + 10;
ll P, m, i, j, x, w, b, t, a[M], c[M], v[M], u[M << 1], ans;
char s[N];
bool notp[N];
int prime[N], pnum;
void sieve() {
    memset(notp, 0, sizeof(notp));
    notp[0] = notp[1] = 1;
    pnum = 0;
```

```
    for (int i = 2; i < N; i++) {
        if (!notp[i]) prime[++pnum] = i;
        for (int j = 1; j <= pnum && prime[j] * i < N; j++) {
            notp[prime[j] * i] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
ll getmod(ll x) {
    ll ans1 = 1, ans2 = 1, y = x;
    for (int i = 1; i <= pnum; i++) {
        if (1ll * prime[i] * prime[i] > x) break;
        if (x % prime[i] == 0) {
            ans1 *= (prime[i] - 1) * (prime[i] + 1);
            ans2 *= prime[i];
            while (x % prime[i] == 0) x /= prime[i];
        }
    }
    if (x > 1) {
        ans1 *= (x - 1) * (x + 1);
        ans2 *= x;
    }
    return y / ans2 * ans1;
}
ll mul(ll x, ll y, ll P) {
    ll tmp = (x * y - (ll)((long double)x / P * y + 1.0e-8) * P);
    return tmp < 0 ? tmp + P : tmp;
}
int main() {
    sieve();
    m = 2;
    for (i = 0; i < m; i++) scanf("%lld", &a[i]);
    for (i = m - 1; ~i; i--) scanf("%lld", &c[i]);
    for (i = 0; i < m; i++) v[i] = 1;
    scanf("%s", s + 1);
    scanf("%lld", &P);
    for (i = 0; i < m; i++) a[i] = a[i] % P;
    for (i = 0; i < m; i++) c[i] = c[i] % P;
    ll mod = getmod(P);
    int len = strlen(s + 1);
    ll n = 0;
    for (int i = 1; i <= len; i++) {
        n = mul(n, 10, mod);
        n = (n + s[i] - '0') % mod;
    }
    for (w = !!n, i = n; i > 1; i >>= 1) w <<= 1;
```

```
    for (x = 0; w; copy(u, u + m, v), w >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0), b = !!(n & w), x |= b;
        if (x < m)
            u[x] = 1;
        else {
            for (i = 0; i < m; i++)
                for (j = 0, t = i + b; j < m; j++, t++)
                    u[t] = ((ll)v[i] * v[j] + u[t]) % P;
            for (i = (m << 1) - 1; i >= m; i--)
                for (j = 0, t = i - m; j < m; j++, t++)
                    u[t] = ((ll)c[j] * u[i] + u[t]) % P;
        }
    }
    for (i = 0; i < m; i++) ans = ((ll)v[i] * a[i] + ans) % P;
    printf("%lld\n", (ans + P) % P);
    return 0;
}
```

6.55 五边形数

```
/*
    五边形数
    整数划分方案数
*/
namespace Pentagon_Number {
const int N = 731, P = mod;
int i, j, f[N + 1], g[200010];
void init(int n) {
    for (f[1] = 1, f[2] = 2, f[3] = 5, f[4] = 7, i = 5; i < N; i++)
        f[i] = 3 + 2 * f[i - 2] - f[i - 4];
    for (g[0] = i = 1; i <= n; i++)
        for (j = 1; f[j] <= i; j++)
            if ((j + 1) >> 1 & 1)
                g[i] = (g[i] + g[i - f[j]]) % P;
            else
                g[i] = (g[i] - g[i - f[j]] + P) % P;
}
} // namespace Pentagon_Number
```

6.56 蔡勒公式

```
/*
    蔡勒公式
```

```
    输入年月日，输出星期几
*/
int Zeller(int y, int m, int d) {
    if (m == 1 || m == 2) m += 12, y--;
    return (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7;
}
```

6.57 法雷序列

```
/*
    法雷序列
    求a/b<p/q<c/d的q最小的分数
*/
using namespace std;
typedef long long ll;
typedef pair<ll, ll> P;
ll a, b, c, d, t;
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
P cal(ll a, ll b, ll c, ll d) {
    ll x = a / b + 1;
    if (x * d < c) return P(x, 1);
    if (!a) return P(1, d / c + 1);
    if (a <= b && c <= d) {
        P t = cal(d, c, b, a);
        swap(t.first, t.second);
        return t;
    }
    x = a / b;
    P t = cal(a - b * x, b, c - d * x, d);
    t.first += t.second * x;
    return t;
}
int main() {
    while (~scanf("%lld%lld%lld%lld", &a, &b, &c, &d)) {
        t = gcd(a, b), a /= t, b /= t;
        t = gcd(c, d), c /= t, d /= t;
        P p = cal(a, b, c, d);
        printf("%lld/%lld\n", p.first, p.second);
    }
    return 0;
}
```

6.58 小数转化为分数

```
/*
    小数转化为分数
    0.2... 2/9
    0.20... 1/5
    0.474612399... 1186531/2500000
*/
string s;
int pow(int n) {
    int ret = 1;
    for (int i = 1; i <= n; i++) ret *= 10;
    return ret;
}
int main() {
    while (cin >> s) {
        if (s == "0") break;
        s = s.substr(2, s.size() - 5);
        int n = atoi(s.c_str()), sz = s.size();
        if (n == 0)
            cout << "0/1" << endl;
        else {
            int ansx, ansy = 1000000000;
            for (int i = 1; i <= sz; i++) {
                string xx = s.substr(0, sz - i);
                int x = n - atoi(xx.c_str());
                int y = pow(sz) - pow(sz - i);
                int gcd = __gcd(x, y);
                x /= gcd;
                y /= gcd;
                if (y < ansy) ansx = x, ansy = y;
            }
            cout << ansx << '/' << ansy << endl;
        }
    }
    return 0;
}
```

6.59 分数规划

```
/*
    分数规划
    Example
        有一个数列，要求选出M个区间，每个区间长度在[L,R]内，
        使得选出的那些数字的平均值尽可能大
*/
```

```
#include <deque>
typedef long long LL;
const LL INF = 10000000000000000071l;
typedef pair<int, LL> PIL;
int n, m, l, r, a[100005];
LL dp[15][100005], sum[100005], pfx[100005];
PIL pt[15][100005];
bool check(LL& S, LL& N) {
    for (int i = 0; i < n; i++) sum[i + 1] = sum[i] + a[i] * N - S;
    for (int x = 1; x <= m; x++) {
        dp[x][0] = -INF;
        deque<PIL> u;
        for (int i = 1; i <= n; i++) {
            while (u.size() && i - u.front().first > r) u.pop_front();
            int k = i - l;
            if (k >= 0 && dp[x - 1][k] != -INF) {
                LL tmp = dp[x - 1][k] - sum[k];
                while (u.size() && tmp > u.back().second) u.pop_back();
                u.push_back(PIL(k, tmp));
            }
            dp[x][i] = dp[x][i - 1], pt[x][i] = pt[x][i - 1];
            if (u.size()) {
                int pos = u.front().first;
                LL tmp = u.front().second + sum[i];
                if (tmp > dp[x][i]) {
                    dp[x][i] = tmp;
                    pt[x][i] = pt[x - 1][pos];
                    pt[x][i].first += i - pos;
                    pt[x][i].second += pfx[i] - pfx[pos];
                }
            }
        }
    }
    if (!dp[m][n]) return 1;
    return N = pt[m][n].first, S = pt[m][n].second, 0;
}
LL GetAns(LL S, LL N) {
    if (S >= 0) return S / N;
    return -((-S + N - 1) / N);
}
int main() {
    while (~scanf("%d%d%d%d", &n, &m, &l, &r)) {
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
            pfx[i + 1] = pfx[i] + a[i];
        }
```

```
        LL S = 0, N = 1;
        while (N && !check(S, N))
            ;
        if (!N) {
            puts("-1");
        } else {
            LL ans = GetAns(S * 100, N);
            printf("%lld.%02lld\n", ans / 100, abs(ans) % 100);
        }
    }
    return 0;
}
```

6.60 线性规划

```
/*
    线性规划
    最大化a[0][i]*xi
    线性约束a[k][i]*xi<=a[k][0]
    type=1时输出x此刻的取值
*/
using namespace std;
typedef long long ll;
const int N = 25;
const double eps = 1e-8, INF = 1e15;
inline int read() {
    char c = getchar();
    int x = 0, f = 1;
    while (c < '0' || c > '9') {
        if (c == '-') f = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * f;
}
int n, m, type;
double a[N][N], ans[N];
int id[N << 1];
int q[N];
void Pivot(int l, int e) {
    swap(id[n + l], id[e]);
    double t = a[l][e];
```

```
    a[l][e] = 1;
    for (int j = 0; j <= n; j++) a[l][j] /= t;
    for (int i = 0; i <= m; i++)
        if (i != l && abs(a[i][e]) > eps) {
            t = a[i][e];
            a[i][e] = 0;
            for (int j = 0; j <= n; j++) a[i][j] -= a[l][j] * t;
        }
}
bool init() {
    while (true) {
        int e = 0, l = 0;
        for (int i = 1; i <= m; i++)
            if (a[i][0] < -eps && (!l || (rand() & 1))) l = i;
        if (!l) break;
        for (int j = 1; j <= n; j++)
            if (a[l][j] < -eps && (!e || (rand() & 1))) e = j;
        if (!e) {
            puts("Infeasible");
            return false;
        } //不存在满足约束的解
        Pivot(l, e);
    }
    return true;
}
bool simplex() {
    while (true) {
        int l = 0, e = 0;
        double mn = INF;
        for (int j = 1; j <= n; j++)
            if (a[0][j] > eps) {
                e = j;
                break;
            }
        if (!e) break;
        for (int i = 1; i <= m; i++)
            if (a[i][e] > eps && a[i][0] / a[i][e] < mn)
                mn = a[i][0] / a[i][e], l = i;
        if (!l) {
            puts("Unbounded");
            return false;
        } //目标值趋于无穷
        Pivot(l, e);
    }
    return true;
}
```

```
int main() {
    srand(317);
    n = read();
    m = read();
    type = read();
    for (int i = 1; i <= n; i++) a[0][i] = read(); //目标系数
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) a[i][j] = read(); //线性约束系数
        a[i][0] = read();                          //小于等于约束
    }
    for (int i = 1; i <= n; i++) id[i] = i;
    if (init() && simplex()) {
        printf("%.8lf\n", -a[0][0]);
        if (type) {
            for (int i = 1; i <= m; i++) ans[id[n + i]] = a[i][0];
            for (int i = 1; i <= n; i++) printf("%.8lf ", ans[i]);
        }
    }
    return 0;
}
```

6.61 拉格朗日插值法

```
/*
    拉格朗日插值法
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int P = 1e9 + 7;
ll Pow(ll x, ll y) {
    ll ret = 1;
    while (y) {
        if (y & 1) ret = ret * x % P;
        y /= 2;
        x = x * x % P;
    }
    return ret;
}
ll inv(ll x) { return Pow(x, P - 2); }
struct polynomial {
    static const int mxN = 1000005;
    ll f[mxN];
    ll fac[mxN];
    ll ul[mxN];
```

```
    ll ur[mxN];
    int n;
    //丢进n个数产生最高次为n-1的多项式
    //丢进的必须是f(0)到f(n-1)
    void init(const vector<ll> &V) {
        n = V.size();
        fac[0] = 1;
        for (int i = 1; i < n; i++) fac[i] = fac[i - 1] * i % P;
        for (int i = 0; i < n; i++) f[i] = V[i];
    }
    //获取f(x)
    ll get(ll x) {
        ul[0] = 1;
        for (int i = 1; i < n; i++) ul[i] = ul[i - 1] * (x - (i - 1)) % P;
        ur[n - 1] = 1;
        for (int i = n - 2; i >= 0; i--) ur[i] = ur[i + 1] * (x - (i + 1)) % P;
        ll ret = 0;
        for (int i = 0; i < n; i++) {
            ll tmp1 = ul[i] * ur[i] % P;
            ll tmp2 = inv(fac[i] * fac[n - 1 - i] % P);
            ll tmp3 = f[i];
            if ((n - 1 - i) & 1) tmp3 *= -1;
            ret = (ret + tmp1 * tmp2 % P * tmp3 % P) % P;
        }
        return (ret + P) % P;
    }
} lgr;
int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    vector<ll> f(k + 2);
    f[0] = 0;
    for (int i = 1; i < k + 2; i++) f[i] = (f[i - 1] + Pow(i, k)) % P;
    lgr.init(f);
    printf("%lld\n", lgr.get(n));
}
/*
    拉格朗日插值法
    1^k+2^k+……+n^k
*/
typedef long long LL;
const int MOD = 1e9 + 7;
const int MX = 1e6 + 10;
struct Lagrange {
    static const int M = 1e6 + 10;
    short factor[M];
```

```
int P[M], S[M], ar[M], inv[M];
inline LL power(LL a, LL b) {
    LL res = 1;
    while (b) {
        if (b & 1) res = res * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}
int lagrange(LL n, int k) {
    if (!k) return n % MOD;
    int i, j, x, res = 0;
    if (!inv[0]) {
        for (i = 2, x = 1; i < MX; i++) x = (long long)x * i % MOD;
        inv[MX - 1] = power(x, MOD - 2);
        for (i = MX - 2; i >= 0; i--)
            inv[i] = ((long long)inv[i + 1] * (i + 1)) % MOD;
    }
    k++;
    for (i = 0; i <= k; i++) factor[i] = 0;
    for (i = 4; i <= k; i += 2) factor[i] = 2;
    for (i = 3; i * i <= k; i += 2) {
        if (!factor[i]) {
            for (j = (i * i), x = i << 1; j <= k; j += x) factor[j] = i;
        }
    }
    for (ar[1] = 1, ar[0] = 0, i = 2; i <= k; i++) {
        if (!factor[i])
            ar[i] = power(i, k - 1);
        else
            ar[i] = ((LL)ar[factor[i]] * ar[i / factor[i]]) % MOD;
    }
    for (i = 1; i <= k; i++) {
        ar[i] += ar[i - 1];
        if (ar[i] >= MOD) ar[i] -= MOD;
    }
    if (n <= k) return ar[n];
    P[0] = 1, S[k] = 1;
    for (i = 1; i <= k; i++)
        P[i] = ((LL)P[i - 1] * ((n - i + 1) % MOD)) % MOD;
    for (i = k - 1; i >= 0; i--)
        S[i] = ((LL)S[i + 1] * ((n - i - 1) % MOD)) % MOD;
    for (i = 0; i <= k; i++) {
        x = (LL)ar[i] * P[i] % MOD * S[i] % MOD * inv[k - i] % MOD *
            inv[i] % MOD;
```

```
            if ((k - i) & 1) {
                res -= x;
                if (res < 0) res += MOD;
            } else {
                res += x;
                if (res >= MOD) res -= MOD;
            }
        }
        return res % MOD;
    }
} lgr;
```

6.62 拉格朗日插值在线

```
/*
    拉格朗日插值
    在线插值&求经过所有插入点的阶小于插入集的f(x)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll P = 998244353;
const int N = 1 << 15 | 1;
int f[N], g[N];
int num = 0;
int query(int x) {
    int ans = 0;
    for (int i = num - 1; i >= 0; i--) {
        ans = (ll)ans * x % P;
        ans += f[i];
        if (ans >= P) ans -= P;
    }
    return ans;
}
int query2(int x) {
    int ans = 0;
    for (int i = num; i >= 0; i--) {
        ans = (ll)ans * x % P;
        ans += g[i];
        if (ans >= P) ans -= P;
    }
    return ans;
}
int mt(int a, int b) {
    if (b == 0) return 1;
```

```
    int c = mt(a, b >> 1);
    c = (ll)c * c % P;
    if (b & 1) c = (ll)c * a % P;
    return c;
}
bool addp(int x, int y) {
    y -= query(x);
    if (y < 0) y += P;
    int d = query2(x);
    if (d == 0) return 0;
    y = (ll)y * mt(d, P - 2) % P;
    for (int i = 0; i <= num; i++) {
        f[i] += (ll)g[i] * y % P;
        if (f[i] >= P) f[i] -= P;
    }
    x = -x;
    if (x < 0) x += P;
    for (int i = num; i >= 0; i--) {
        g[i + 1] += g[i];
        if (g[i + 1] >= P) g[i + 1] -= P;
        g[i] = (ll)g[i] * x % P;
    }
    num++;
    return 1;
}
int main() {
    f[0] = 0, g[0] = 1;
    int n;
    scanf("%d", &n);
    while (n--) {
        int opt, x;
        scanf("%d%d", &opt, &x);
        if (opt == 2) {
            printf("%d\n", query(x));
        } else {
            int y;
            scanf("%d", &y);
            addp(x, y);
        }
    }
    return 0;
}
```

6.63 拉格朗日插值求值段

```
/*
    拉格朗日插值+FFT
    已知f(0)~f(n)求解f(m)~f(m+n)
*/
#include <bits/stdc++.h>
typedef unsigned long long ull;
const int maxN = 100005;
const int maxL = 1 << 18 | 1;
const int Mod = 998244353;
int fexp(int base, int exp) {
    int res = 1;
    for (; exp; exp >>= 1, base = (ull)base * base % Mod)
        if (exp & 1) res = (ull)res * base % Mod;
    return res;
}
int Rev[maxL], Wn[maxL];
void DFT_pre(int l, int bitcnt) {
    for (int i = 1; i != l; ++i)
        Rev[i] = (Rev[i >> 1] >> 1) | ((i & 1) << (bitcnt - 1));
}
void DFT(int *a, int n) {
    for (int i = 1; i != n; ++i)
        if (i < Rev[i]) std::swap(a[i], a[Rev[i]]);
    for (int l = 1; l != n; l <<= 1) {
        int w = fexp(3, (Mod - 1) / (l << 1));
        Wn[0] = 1;
        for (int i = 1; i != l; ++i) Wn[i] = (ull)Wn[i - 1] * w % Mod;
        for (int i = 0; i != n; i += l << 1)
            for (int j = 0; j != l; ++j) {
                int x = a[i + j], y = (ull)Wn[j] * a[i + l + j] % Mod;
                a[i + j] = (x + y) % Mod;
                a[i + l + j] = (x + Mod - y) % Mod;
            }
    }
}
void iDFT(int *a, int n) {
    for (int i = 1; i != n; ++i)
        if (i < Rev[i]) std::swap(a[i], a[Rev[i]]);
    for (int l = 1; l != n; l <<= 1) {
        int w = fexp((Mod + 1) / 3, (Mod - 1) / (l << 1));
        Wn[0] = 1;
        for (int i = 1; i != l; ++i) Wn[i] = (ull)Wn[i - 1] * w % Mod;
        for (int i = 0; i != n; i += l << 1)
            for (int j = 0; j != l; ++j) {
                int x = a[i + j], y = (ull)Wn[j] * a[i + l + j] % Mod;
```

```
                a[i + j] = (x + y) % Mod;
                a[i + l + j] = (x + Mod - y) % Mod;
            }
    }
    for (int i = 0, inv = fexp(n, Mod - 2); i != n; ++i)
        a[i] = (ull)a[i] * inv % Mod;
}
int N, M, F[maxN];
int Fac[maxN], invFac[maxN];
int A[maxL], B[maxL];
int main() {
    scanf("%d%d", &N, &M);
    for (int i = 0; i <= N; ++i) scanf("%d", F + i);
    Fac[0] = 1;
    for (int i = 1; i <= N; ++i) Fac[i] = (ull)Fac[i - 1] * i % Mod;
    invFac[N] = fexp(Fac[N], Mod - 2);
    for (int i = N; i; --i) invFac[i - 1] = (ull)invFac[i] * i % Mod;
    for (int i = 0; i <= N; ++i) {
        A[i] = (ull)invFac[i] * invFac[N - i] % Mod;
        if ((N - i) & 1) A[i] = Mod - A[i];
        A[i] = (ull)A[i] * F[i] % Mod;
    }
    int l = 1, bitcnt = 0;
    while (l <= N + N) l <<= 1, ++bitcnt;
    for (int i = -N; i <= N; ++i) B[(l + i) & (l - 1)] = fexp(M + i, Mod - 2);
    DFT_pre(l, bitcnt);
    DFT(A, l);
    DFT(B, l);
    for (int i = 0; i != l; ++i) A[i] = (ull)A[i] * B[i] % Mod;
    iDFT(A, l);
    int t = 1;
    for (int i = 0; i <= N; ++i) t = (ull)t * (M - i) % Mod;
    for (int i = 0; i <= N; ++i) {
        printf("%llu%c", (ull)t * A[i] % Mod, " \n"[i == N]);
        t = (ull)t * (M + i + 1) % Mod;
        t = (ull)t * fexp(M + i - N, Mod - 2) % Mod;
    }
    return 0;
}
```

6.64 曼哈顿最小距离

```
/*
    曼哈顿最大距离
    D表示维度
```

```
    a[n][D]为D维点坐标
*/
#define inf 1e100
const int D = 2;
int n;
double a[N][D];
void solve() {
    double ans = 0, mi, mx, t;
    for (int s = 0; s < (1 << D); s++) {
        mi = inf, mx = -inf;
        for (int i = 0; i < n; i++) {
            t = 0;
            for (int j = 0; j < D; j++)
                if ((1 << j) & s)
                    t += a[i][j];
                else
                    t -= a[i][j];
            mi = min(mi, t);
            mx = max(mx, t);
        }
        ans = max(ans, mx - mi);
    }
    printf("%.0lf\n", ans);
}
/*
    Problem:
        有n个主武器和m个副武器，分别有个S值和k个x值，x1,x2……，xk。
        请选取一个主武器和一个副武器，使得Sz+Sf+ |Xz-Xf|最大
    Solution:
        我们将其改为求|Sz-0|+|Sf-0|+ |Xz-Xf|,
        那么问题就转化为从两个集合中各选取一个k+2维点，使其k+2维曼哈顿距离最大
        我们加入1e15作为集合识别，将其转化为k+3维曼哈顿问题
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll inf = 1e18, base = 1e15;
const int N = 300010;
const int D = 8;
int T, n, m, k;
ll a[N][D];
void solve() {
    ll ans = 0, mi, mx, t;
    for (int s = 0; s < (1 << k); s++) {
        mi = inf, mx = -inf;
        for (int i = 0; i < n + m; i++) {
```

```
            t = 0;
            for (int j = 0; j < k; j++)
                if ((1 << j) & s)
                    t += a[i][j];
                else
                    t -= a[i][j];
            mi = min(mi, t);
            mx = max(mx, t);
        }
        ans = max(ans, mx - mi);
    }
    printf("%lld\n", ans - base);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d", &n, &m, &k);
        k += 3;
        for (int i = 0; i < n + m; i++) {
            if (i < n) {
                a[i][0] = base;
                scanf("%lld", &a[i][1]);
                a[i][2] = 0;
            } else {
                a[i][0] = 0;
                scanf("%lld", &a[i][2]);
                a[i][1] = 0;
            }
            for (int j = 3; j < k; j++) scanf("%lld", &a[i][j]);
        }
        solve();
    }
    return 0;
}
```

6.65 Simpson

```
/*
    自适应Simpson
    给定一个函数f(x),求[a,b]区间内f(x)到x轴所形成区域的面积。
*/
double simpson(double l, double r) {
    return (f(l) + f(r) + 4 * f((l + r) / 2.0)) * (r - l) / 6.0;
}
double rsimpson(double l, double r) {
```

```
    double mid = (l + r) / 2.0;
    if (fabs(simpson(l, r) - simpson(l, mid) - simpson(mid, r)) < eps)
        return simpson(l, mid) + simpson(mid, r);
    return rsimpson(l, mid) + rsimpson(mid, r);
}
```

6.66 群论

```
/*
    群论
    polya定理:  一个置换的不动置换方案数=k^(这个置换的循环个数)
    burnside定理 : 不等价方案数=每个置换的不动置换方案数的和 / 置换个数
*/
/*
    Example:
        给出一个序列
        交换值为x和y的两个数代价为x+y
        求从小排到大的代价
    Solution:
        1.找出初始状态和目标状态。明显，目标状态就是排序后的状态。
        2.画出置换群，在里面找循环。例如，数字是8 4 5 3 2 7
        明显，目标状态是2 3 4 5 7 8，能写为两个循环:
        (8 2 7)(4 3 5)。
        3.观察其中一个循环，明显地，要使交换代价最小，
        应该用循环里面最小的数字2，去与另外的两个数字，7与8交换。这样交换的代价是:
            sum - min + (len - 1) * min
        化简后为:
            sum + (len - 2) * min
        其中，sum为这个循环所有数字的和，len为长度，min为这个环里面最小的数字。
        4.考虑到另外一种情况，我们可以从别的循环里面调一个数字，
        进入这个循环之中，使交换代价更小。例如初始状态:
        1 8 9 7 6
        可分解为两个循环:
        (1)(8 6 9 7)，明显，第二个循环为(8 6 9 7)，最小的数字为6。、
        我们可以抽调整个数列最小的数字1进入这个循环。
        使第二个循环变为:(8 1 9 7)。让这个1完成任务后，
        再和6交换，让6重新回到循环之后。这样做的代价明显是:
        sum + min + (len + 1) * smallest
        其中，sum为这个循环所有数字的和，len为长度，
        min为这个环里面最小的数字，smallest是整个数列最小的数字。
        5.因此，对一个循环的排序，其代价是sum - min + (len - 1) * min
        和sum + min + (len + 1) * smallest之中小的那个数字。
        6.我们在计算循环的时候，不需要记录这个循环的所有元素，
        只需要记录这个循环的最小的数及其和。
        7.在储存数目的时候，我们可以使用一个hash结构，
```

```
        将元素及其位置对应起来，以达到知道元素，
        可以快速反查元素位置的目的。这样就不必要一个个去搜索。
*/
const int N = 10005, inf = 1000000000;
int n, gmn, cnt, ans, v[N], id[N], disc[N], mn[N], sum[N], len[N];
bool vis[N];
int find(int x) {
    int l = 1, r = n;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (disc[mid] > x)
            r = mid - 1;
        else if (disc[mid] == x)
            return mid;
        else
            l = mid + 1;
    }
}
void solve(int x) {
    vis[x] = 1;
    cnt++;
    len[cnt] = 1;
    sum[cnt] += v[x];
    mn[cnt] = min(mn[cnt], v[x]);
    int now = x;
    while (v[id[now]] != v[x]) {
        now = id[now];
        vis[now] = 1;
        len[cnt]++;
        sum[cnt] += v[now];
        mn[cnt] = min(mn[cnt], v[now]);
    }
}
int main() {
    memset(mn, 127 / 3, sizeof(mn));
    gmn = inf;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &v[i]), disc[i] = v[i], gmn = min(gmn, v[i]);
    sort(disc + 1, disc + n + 1);
    for (int i = 1; i <= n; i++) id[i] = find(v[i]);
    for (int i = 1; i <= n; i++)
        if (!vis[i] && i != id[i]) solve(i);
    for (int i = 1; i <= cnt; i++) {
        int t1 = (len[i] - 2) * mn[i];
        int t2 = mn[i] + (len[i] + 1) * gmn;
```

```
        ans += sum[i] + min(t1, t2);
    }
    return printf("%d", ans), 0;
}
/*
    Example:
        给定M种颜色的珠子，每种颜色珠子的个数均不限，将这些珠子做成长度为N的项链。
        问能做成多少种不重复的项链，最后的结果不会超过int类型数据的表示范围。
        并且两条项链相同，当且仅当两条项链通过旋转或是翻转后能重合在一起，
        且对应珠子的颜色相同。
*/
int N, M;
int main() {
    while (~scanf("%d%d", &M, &N) && (M || N)) {
        int ans = 0;
        for (int i = 1; i <= N; i++) {
            int tmp = __gcd(N, i);
            ans += (int)(pow(M * 1.0, tmp * 1.0));
        }
        if (N & 1)
            ans += (int)(N * pow(M * 1.0, (N + 1) / 2.0));
        else {
            ans += (int)((N / 2) * pow(M * 1.0, (N + 2) / 2.0));
            ans += (int)((N / 2) * pow(M * 1.0, N / 2.0));
        }
        ans = ans / (2 * N);
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example:
        一个手镯，用三种颜色上色，可以旋转和翻转，求有多少方案。
*/
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
int main() {
    int c = 3, s;
    while (scanf("%d", &s) && s != -1) {
        if (s == 0) {
            puts("0");
            continue;
        }
        long long p[64];
        p[0] = 1;
        for (int k = 0; k < s; k++) p[k + 1] = p[k] * c;
        long long cnt =
```

```
            s & 1 ? p[s / 2 + 1] * s : (p[s / 2] + p[s / 2 + 1]) * (s / 2);
        for (int k = 1; k <= s; k++) cnt += p[gcd(k, s)];
        printf("%lld\n", cnt /= 2 * s);
    }
    return 0;
}
```

6.67 置换开 m 次根

```
/*
    置换开m次方
    长度为n的置换开m次方，solve返回值表示能否表示为m次方，
    ans数组保存开方后的结果
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 1000010;
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
int cal(int x, int m) {
    int i, k = m, t = 1;
    for (i = 2; i * i <= x; i++)
        if (x % i == 0) {
            while (x % i == 0) x /= i, t *= i;
            while (k % i == 0) k /= i, t *= i;
        }
    if (x > 1)
        for (t *= x; k % x == 0; k /= x, t *= x)
            ;
    return t;
}
bool v[N];
int a[N], g[N], nxt[N], q[N], b[N], ans[N];
bool solve(int n, int m) {
    memset(v, 0, sizeof(v));
    memset(g, 0, sizeof(g));
    int x, i, j, k, o, l, t;
    for (i = 1; i <= n; i++)
        if (!v[i]) {
            t = v[i] = 1;
            for (int j = a[i]; j != i; j = a[j]) v[j] = 1, t++;
            nxt[i] = g[t], g[t] = i;
        }
    for (i = 1; i <= n; i++)
```

```
        if (g[i]) {
            for (t = 0, j = g[i]; j; j = nxt[j]) q[++t] = j;
            int d = gcd(l = cal(i, m), m);
            if (t % d) return 0;
            for (x = 1; x <= t; x += d) {
                for (j = 0; j < d; j++)
                    for (k = 0, o = q[x + j]; k < i; k++, o = a[o])
                        b[(j + 1LL * k * m) % l] = o;
                for (j = 0; j < l; j++) ans[b[j]] = b[(j + 1) % l];
            }
        }
    return 1;
}
/*
    判断给出置换能否开方
*/
int T, n, m;
char s[30];
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%s", s + 1);
        for (int i = 1; i <= 26; i++) a[i] = s[i] - 'A' + 1;
        int ans = solve(26, 2);
        if (ans)
            puts("Yes");
        else
            puts("No");
    }
    return 0;
}
```

6.68 构造

```
/*
    构造
*/
/*
    题目大意:
        构造出2^n个由1和-1组成的串使得其两两点积为0
    题解:
        我们可以构造这样一个矩阵，右下角四分之一和其余三个分块相反，
        不断扩展这个矩阵即可。
*/
#include <cstdio>
```

```
#define rep(i, n) for (int i = 1; i <= n; i++)
using namespace std;
const int N = 1000;
int k, p[N][N];
int main() {
    for (int n = 1; n <= 9; n++) {
        int m = 1 << (n - 1);
        rep(i, m) rep(j, m) p[i + m][j] = p[i][j + m] = p[i][j],
                                p[i + m][j + m] = !p[i][j];
    }
    while (~scanf("%d", &k)) {
        rep(i, 1 << k) {
            rep(j, 1 << k) putchar(p[i][j] ? '+' : '*');
            puts("");
        }
    }
    return 0;
}
/*
    题目大意:
        给出一个1到n的排列，问每两个位置都进行一次交换最终排列不变是否可能，
        如果可能输出交换顺序。
    题解:
        我们发现对于四个一组组内进行六次交换之后可以保证四个数的位置不变，
        而对于每组相互之间可以一共进行十六次交换使得两组均不会发生变化
        所以如果n能被4整除，那么我们可以4个分组达到目的，
        当不能被4整除的时候，我们发现余数为2和3的时候都不能构造出可行解。
        在余数为1的时候有一种特殊的基于4分组的构造法，
        我们在相邻两个数字交换的时候，我们将多出来的那个数字作为第三参数，
        这样多出来那个数字就能够和剩余所有位置交换过一次而不改变位置。
*/
#include <algorithm>
#include <cstdio>
#include <utility>
#include <vector>
using namespace std;
#define rep(i, n) for (int i = 0; i < n; i++)
typedef pair<int, int> P;
typedef vector<P> V;
void add(V &ans, int pos1, int pos2) {
    pos1 <<= 2;
    pos2 <<= 2;
    rep(k, 4) rep(i, 4) ans.push_back(P(pos1 + i, pos2 + (i ^ k)));
}
int dx[] = {0, 0, 1, 0, 1, 2}, dy[] = {1, 2, 3, 3, 2, 3};
void add4(V &ans, int pos) {
```

```
    pos <<= 2;
    rep(i, 6) ans.push_back(P(pos + dx[i], pos + dy[i]));
}
V make4(int n) {
    V ans;
    rep(i, n / 4) add4(ans, i);
    rep(i, n / 4) rep(j, i) add(ans, j, i);
    return ans;
}
int n;
void solve() {
    V ans;
    if (n % 4 == 0)
        ans = make4(n);
    else if (n % 4 == 1) {
        V tmp = make4(n - 1);
        for (int i = 0; i < tmp.size(); i++) {
            P p = tmp[i];
            int x = p.first, y = p.second;
            if (x > y) swap(x, y);
            if (y == x + 1 && y % 2) {
                ans.push_back(P(n - 1, x));
                ans.push_back(P(x, y));
                ans.push_back(P(n - 1, y));
            } else
                ans.push_back(p);
        }
    } else {
        puts("NO");
        return;
    }
    puts("YES");
    for (int i = 0; i < ans.size(); i++) {
        P p = ans[i];
        if (p.first > p.second) swap(p.first, p.second);
        printf("%d %d\n", p.first + 1, p.second + 1);
    }
}
int main() {
    while (~scanf("%d", &n)) solve();
    return 0;
}
```

6.69 分数类

```
/*
    分数类
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 500;
struct fraction {
    long long numerator; // 分子
    long long denominator; // 分母
    fraction() {
        numerator = 0;
        denominator = 1;
    }
    fraction(long long num) {
        numerator = num;
        denominator = 1;
    }
    fraction(long long a, long long b) {
        numerator = a;
        denominator = b;
        this->reduction();
    }
    void operator=(const long long num) {
        numerator = num;
        denominator = 1;
        this->reduction();
    }
    void operator=(const fraction &b) {
        numerator = b.numerator;
        denominator = b.denominator;
        this->reduction();
    }
    fraction operator+(const fraction &b) const {
        long long gcdnum = __gcd(denominator, b.denominator);
        fraction tmp = fraction(numerator * (b.denominator / gcdnum) +
                                b.numerator * (denominator / gcdnum),
                            denominator / gcdnum * b.denominator);
        tmp.reduction();
        return tmp;
    }
    fraction operator+(const int &b) const { return ((*this) + fraction(b)); }
    fraction operator-(const fraction &b) const {
        return ((*this) + fraction(-b.numerator, b.denominator));
    }
    fraction operator-(const int &b) const { return ((*this) - fraction(b)); }
```

```
    fraction operator*(const fraction &b) const {
        fraction tmp =
            fraction(numerator * b.numerator, denominator * b.denominator);
        tmp.reduction();
        return tmp;
    }
    fraction operator*(const int &b) const { return ((*this) * fraction(b)); }
    fraction operator/(const fraction &b) const {
        return ((*this) * fraction(b.denominator, b.numerator));
    }
    void reduction() {
        if (numerator == 0) {
            denominator = 1;
            return;
        }
        long long gcdnum = __gcd(numerator, denominator);
        numerator /= gcdnum;
        denominator /= gcdnum;
    }
    void print() {
        if (denominator == 1)
            printf("%lld\n", numerator);
        else {
            long long num = numerator / denominator;
            long long tmp = num;
            int len = 0;
            while (tmp) {
                len++;
                tmp /= 10;
            }
            for (int i = 0; i < len; i++) printf(" ");
            if (len != 0) printf(" ");
            printf("%lld\n", numerator % denominator);
            if (num != 0) printf("%lld ", num);
            tmp = denominator;
            while (tmp) {
                printf("-");
                tmp /= 10;
            }
            puts("");
            for (int i = 0; i < len; i++) printf(" ");
            if (len != 0) printf(" ");
            printf("%lld\n", denominator);
        }
    }
} f[N];
```

```
int n;
/*
    要抽齐n种卡片期望要买的卡片数
*/
int main() {
    while (~scanf("%d", &n)) {
        f[0] = 0;
        for (int i = 1; i <= n; i++) f[i] = f[i - 1] + fraction(1, i);
        f[n] = f[n] * n;
        f[n].print();
    }
    return 0;
}
```

6.70 连续随机均匀分布联合概率

```
/*
    连续随机均匀分布联合概率
    S=x1+x2+x3+……
    已知x[i]的取值范围xl[i]~xr[i]，求S取值范围为[l,r]的概率
    一般做法，将x的取值范围变为左端点为0，xr[i]=xr[i]-xl[i]，同时l,r也减去xl[i]，
    转化为x取值范围为[0.X]来计算
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 20;
typedef long double LD;
namespace Continuous_Random_Uniform_Distribution {
int n;
double x[N];
LD cal(double x) {
    if (x < 0) return 0;
    LD res = 1;
    for (int i = 1; i <= n; i++) res = res * x / i;
    return res;
}
LD solve(double d) {
    LD res = 0;
    for (int i = 0; i < 1 << n; i++) {
        int cnt = 1;
        double cur = d;
        for (int j = 0; j < n; j++) {
            if ((1 << j) & i) {
                cnt *= -1;
                cur -= x[j];
```

```
            }
        }
        res += cnt * cal(cur);
    }
    return res;
}
LD Cal(double l, double r) {
    LD base = 1;
    for (int i = 0; i < n; i++) base *= x[i];
    return fabs(solve(r) - solve(l)) / base;
}
} // namespace Continuous_Random_Uniform_Distribution
/*
    Example: Xi的取值范围为[-xi,xi], S=X1+X2+X3+……, 求S取值在[l,r]范围的概率
*/
using namespace Continuous_Random_Uniform_Distribution;
int main() {
    int T;
    double l, r;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%lf%lf", &n, &l, &r);
        for (int i = 0; i < n; i++) {
            scanf("%lf", &x[i]);
            l += x[i];
            r += x[i];
            x[i] *= 2;
        }
        printf("%.9f\n", (double)Cal(l, r));
    }
    return 0;
}
```

6.71 概率网络

```
/*
    概率网络
    Example:
        有一张无向图, 两个人在起点s1和s2, 在每个点i, 一个人在下一分钟有p[i]的概率留在原地,
        (1-p[i])的概率随机到相连的点, 问两个人每个点相遇的概率
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 1010;
int n, m, s1, s2, a[N], b[N], d[N];
```

```
int ed, nxt[N << 1], u[N << 1], v[N << 1], head[N];
double w[N << 1];
void add_edge(int a, int b, double c) {
    u[++ed] = a, v[ed] = b, w[ed] = c;
    nxt[ed] = head[u[ed]];
    head[u[ed]] = ed;
}
int sz;
double F[N][N], p[N];
int mp(int x, int y) { return (x - 1) * n + y; }
void BuildEquations() {
    F[mp(s1, s2)][0] = -1.0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) continue;
            for (int ii = head[i]; ii; ii = nxt[ii]) {
                for (int jj = head[j]; jj; jj = nxt[jj]) {
                    F[mp(v[ii], v[jj])][mp(i, j)] += w[ii] * w[jj];
                }
            }
        }
    }
    for (int i = 1; i <= sz; i++) F[i][i] -= 1.0;
}
void Swap(int x, int y) {
    for (int i = 0; i <= sz; i++) swap(F[x][i], F[y][i]);
}
void multplus(int x, int y, double c) {
    for (int i = 0; i <= sz; i++) F[x][i] += F[y][i] * c;
}
void Gauss() {
    for (int i = 1; i <= sz; i++) {
        if (F[i][i] == 0) {
            for (int j = i + 1; j <= sz; j++) {
                if (F[j][i] != 0) {
                    Swap(i, j);
                    break;
                }
            }
        }
        for (int j = i + 1; j <= sz; j++) multplus(j, i, -F[j][i] / F[i][i]);
        for (int j = i - 1; j; j--) multplus(j, i, -F[j][i] / F[i][i]);
    }
    for (int i = 1; i <= sz; i++) F[i][0] /= F[i][i];
}
int main() {
```

```
    scanf("%d%d%d%d", &n, &m, &s1, &s2);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &a[i], &b[i]);
        d[a[i]]++;
        d[b[i]]++;
    }
    for (int i = 1; i <= n; i++) {
        scanf("%lf", &p[i]);
        add_edge(i, i, p[i]);
    }
    for (int i = 1; i <= m; i++) {
        add_edge(a[i], b[i], (1 - p[a[i]]) / d[a[i]]);
        add_edge(b[i], a[i], (1 - p[b[i]]) / d[b[i]]);
    }
    sz = n * n;
    BuildEquations();
    Gauss();
    for (int i = 1; i <= n; i++)
        printf("%.7f%c", F[mp(i, i)][0], " \n"[i == n]);
    return 0;
}
```

6.72 期望-势能函数

```
/*
    期望-势能函数
*/
/*
    Problem:
        有n个节点，每个节点有两种状态：选中和未选中。
        每个选中的点后面都跟着若干个（可能是0个）未选中的点。每个未选中的点都一定跟在某个选中的点后面。
        每次操作随机选择两个被选中的点，随机将其中一个变成未选中，且跟在另一个后面，
        同时将跟在他后面的节点全部改为选中。求这样操作下去，直到最后只剩一个选中的点的期望步数。
    Solution:
        设一个点后面跟着k个被选中的点，设其的势能函数为：2^k-1
        那么目标状态的势能函数值就是2^(n-1)-1
        对每一次操作，计算它的势能函数期望变化量，
        Δ=1/2*((2^(p+1)-1)-(2^p-1)-(2^q-1))+1/2*((2^(q+1)-1)-(2^p-1)-(2^q-1))=1
        势能函数期望变化量为1，所以期望步数即势能函数差值
        整个的期望步数就是目标状态的势能函数，减去当前状态的势能函数
*/
#include <bits/stdc++.h>
using namespace std;
const int N = 510, P = 1e9 + 7;
int n, x, pw[N], cnt[N];
```

```
int main() {
    scanf("%d", &n);
    pw[0] = 1;
    for (int i = 1; i <= n; i++) pw[i] = pw[i - 1] * 2 % P;
    for (int i = 1; i <= n; i++) {
        scanf("%d", &x);
        if (~x) cnt[x]++;
    }
    int ans = pw[n - 1] - 1;
    for (int i = 1; i <= n; i++) ans = (ans - (pw[cnt[i]] - 1)) % P;
    printf("%d\n", (ans + P) % P);
    return 0;
}
```

6.73 期望-组合项精度保护

```
/*
    期望-组合项精度保护
    Problem:
        有两个盒子，每个盒子里有n颗糖，每次打开第一个盒子概率为p，第二个盒子概率为1-p，
        每次打开盒子之后就拿走一颗糖，问当某次打开盒子没有糖时，另一个盒子拥有的糖数量的期望
    Solution:
        假设最后打开第1个盒子为空，第2个盒子里还有i颗糖，其概率为C(2n-1,n)*p^(n+1)*(1-p)^(n-i)
        同理最后打开第2个盒子为空，第1个盒子里还有i颗糖的概率为C(2n-1,n)(1-p)^(n+1)p^(n-i)
        所以总期望就为 i*(C(2n-1,n)*p^(n+1)*(1-p)^(n-i)+C(2n-1,n)(1-p)^(n+1)p^(n-i))
        考虑到n接近20万时组合数极大而p的(n+1)次幂极小容易导致精度缺失，故两者均取ln后相加，
        最后求exp即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long double LD;
const int N = 400010;
LD logF[N];
void Init() {
    for (int i = 1; i < N; i++) logF[i] = logF[i - 1] + log(i);
}
LD logC(int n, int m) { return logF[n] - logF[m] - logF[n - m]; }
int n, cas = 0;
double p;
int main() {
    Init();
    while (~scanf("%d%lf", &n, &p)) {
        double ans = 0.0;
        for (int i = 1; i <= n; i++) {
            LD c = logC(2 * n - i, n);
```

```
            LD v1 = c + (n + 1) * log(p) + (n - i) * log(1 - p);
            LD v2 = c + (n + 1) * log(1 - p) + (n - i) * log(p);
            ans += i * (exp(v1) + exp(v2));
        }
        printf("Case %d: %.6f\n", ++cas, ans);
    }
    return 0;
}
```

7 博弈论

7.1 尼姆博弈

```
/*
    Nim游戏:
        n堆石子，双方轮流从任意一堆石子中取出至少一个，不能取的人输.
    策略:
        对于一堆x个石子的情况，sg(x)=x.
        所以所有石子个数的异或和为0是必败态，否则为必胜态.
*/
/*
    Example:
        每个人每次可以从一堆石子中取出若干个石子，
        每次取石子的个数有限制，谁不能取石子时就会输掉游戏。
        小H先进行操作，他想问你他是否有必胜策略，
        如果有，第一步如何取石
*/
const int N=1005;
int SG[N],b[N],hash[N],a[N],sum,tmp,n,m;
void FSG(int s){
    SG[0]=0;
    for(int i=1;i<=s;i++){
        for(int j=1;b[j]<=i&&j<=m;j++)hash[SG[i-b[j]]]=i;
        for(int j=0;j<=s;j++)if(hash[j]!=i){SG[i]=j;break;}
    }
}
int main(){
    // n个数表示每堆石子的个数
    scanf("%d",&n); for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    // m个数表示每次可以取的石子个数，输入保证这m个数按照递增顺序排列
    scanf("%d",&m); for(int i=1;i<=m;i++)scanf("%d",&b[i]);
    FSG(N); for(int i=1;i<=n;i++)sum^=SG[a[i]];
    if(!sum){printf("NO");return 0;}
    for(int i=1;i<=n;i++){
        tmp=sum^SG[a[i]];
        for(int j=1;b[j]<=a[i]&&j<=m;j++)
        if(!(tmp^SG[a[i]-b[j]])){
            printf("YES\n%d %d",i,b[j]);
            return 0;
        }
    }
}
/*
    Example:
        取石子游戏新手必胜策略数
```

```
*/
void solve(){
    while(~scanf("%d",&n),n){
        int sg=0,ans=0;
        for(int i=0;i<n;i++)scanf("%d",&a[i]),sg^=a[i];
        for(int i=0;i<n;i++)if((sg^a[i])<a[i])ans++;
        printf("%d\n",ans);
    }
}
```

7.2 巴什博奕

```
/*
    巴什博弈:
        有一堆石子，每人最多一次只能取m个石子，谁没有石子可以取谁输.
    策略:
        SG(x)=x%(m+1)
    Example:
        对于巴什博奕给定石子数，请给出每次取的上限值，使后手必胜
*/
int k,i,ans;
using namespace std;
int main(){
    while(~scanf("%d",&k)){
        if(k%3==0)puts("2");
        else if(k%4==0)puts("3");
        else{
            while(k%2==0)k/=2; ans=k-1;
            for(i=5;i<=int(sqrt((double)k))+1;i++)if(k%i==0){ans=i-1;break;}
            printf("%d\n",ans);
        }
    }return 0;
}
```

7.3 Multi-Nim

```
/*
    Multi-Nim游戏
    多个游戏的sg值为sg值的异或值
    Example:
        在一个1*n的方格纸上下棋，谁先连三子谁就赢了，问必胜的是谁。
    Solution:
        我们发现对于一个n规模的游戏。在i位置下棋就能将游戏拆分为i-3和n-i-2两个游戏
```

```
        对于可拆分的游戏，其sg函数为拆分后游戏sg值的异或和，
        因此我们用记忆化搜索来记录sg值。
*/
int n,sg[2010];
int SG(int n){
    if(n<0)return 0;
    if(sg[n]>=0)return sg[n];
    bool tmp[2010]={0};
    for(int i=1;i<=n;i++){
        int t=SG(i-3)^SG(n-i-2);
        tmp[t]=1;
    }for(int i=0;;i++){
        if(!tmp[i])return sg[n]=i;
    }
}
int main(){
    memset(sg,-1,sizeof(sg));
    while(~scanf("%d",&n)){
        if(SG(n))puts("1");
        else puts("2");
    }return 0;
}
```

7.4 反尼姆博弈

```
/*
    Anti-Nim博弈
        n堆石子，双方轮流从任意一堆石子中取出至少一个，不能取的一方获胜
    先手必胜:
        1.所有堆的石子数都为1且游戏的SG值为0;
        2.有些堆的石子数大于1且游戏的SG值不为0。
*/
while(T--){
    scanf("%d",&n);
    for(s=tmp=0;n--;)scanf("%d",&x),s^=x,tmp|=(x>1);
    puts((s>0)^tmp?"Win":"Lose");
}
```

7.5 阶梯博弈

```
/*
    阶梯博弈
    Example:
```

```
        有N堆石子，除了第一堆外，每堆石子个数都不少于前一堆的石子个数。
        两人轮流操作每次操作可以从一堆石子中移走任意多石子，
        但是要保证操作后仍然满足初始时的条件谁没有石子可移时输掉游戏。问先手是否必胜。
    Solution:
        我们定义最后一堆以及下标与其相差为偶数的堆为偶堆石子，其余的为奇堆石子，
        我们发现不管奇堆石子做什么操作，只要在偶堆石子中做相应的操作就可以抵消，
        因此决定这个游戏胜负的是奇堆石子和后一个偶堆石子之间的差值，
        其差值序列等同于NIM游戏，因此求出其sg值即可。
*/
const int N=1010;
int n,a[N],T;
int main(){
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        int sg=0;
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        for(int i=n;i;i--)if(!((n-i)&1))sg^=(a[i]-a[i-1]);
        puts(sg?"TAK":"NIE");
    }return 0;
}
```

7.6 Nimk-Game

```
/*
    Nimk游戏:
        n堆石子，每人一次可以从最多m堆石子中取出每堆至多k个，不能取的人输.
    策略:
        在二进制下各堆石子的sg函数均为(m+1)的倍数的话则为必败态，否则为必胜态.
*/
using namespace std;
const int N=100010;
int n,m,k,a[N],sg[N];
bool SG(){
    memset(sg,0,sizeof(sg));
    for(int i=0;i<n;i++)for(int j=0,g=a[i]%(k+1);sg[j]+=g&1,g;j++,g>>=1);
    for(int i=0;i<30;i++)if(sg[i]%(m+1))return 1;
    return 0;
}
int main(){
    while(~scanf("%d%d%d",&n,&m,&k)){
        for(int i=0;i<n;i++)scanf("%d",&a[i]);
        puts(SG()?"Alice":"Bob");
    }return 0;
}
```

7.7 威佐夫博弈

```
/*
    威佐夫博弈
        有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，
        规定每次至少取一个，多者不限，最后取光者得胜。
    策略:
        任给一个局势(a,b)，如下公式判断它是不是奇异局势(输):
        ak=[k(1+√5)/2]，bk=ak+k(k=0,1,2,…,n方括号表示取整函数）。
*/
bool Wythoff(int n,int m){
    if(n<m) swap(n,m);
    int k=n-m;
    n=(int)(k*(1+sqrt(5))/2.0);
    if(n==m)return 0;
    else return 1;
}
// JAVA
import java.math.BigDecimal;
import java.util.Scanner;
public class Main{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        BigDecimal two=new BigDecimal(2);
        BigDecimal three= new BigDecimal(3);
        BigDecimal five=new BigDecimal(5);
        BigDecimal l=two,r=three;
        for(int i=0;i<500;i++){
                BigDecimal mid=l.add(r).divide(two);
                if(mid.multiply(mid).compareTo(five)<0)
                        l=mid;
                else
                        r=mid;
        }
        BigDecimal gold=l.add(BigDecimal.ONE).divide(two);//黄金分割数
        BigDecimal a,b;
        while(sc.hasNext()){
                a=sc.nextBigDecimal();
                b=sc.nextBigDecimal();
                if(a.compareTo(b)>0){
                        BigDecimal tmp=a;
                        a=b;b=tmp;
                }
```

```
                b=b.subtract(a).multiply(gold);
                b=b.setScale(0,BigDecimal.ROUND_DOWN);
                if(a.compareTo(b)==0)
                        System.out.println("0");
                else
                        System.out.println("1");
        }
    }
}
```

7.8 Take-Break

```
/*
    Take&Break:
        有n堆石子，双方轮流取出一堆石子，
        然后新增两堆规模更小的石子堆（可以没有石子），无法操作者输.
    Solution:
        唯一解法：暴力求SG
*/
```

7.9 对称性博弈

```
/*
    对称性博弈
    Example:
        两个人玩游戏. 每次任选一堆，首先拿掉至少一个石头，
        然后移动任意个石子到任意堆中. 谁不能移动了，谁就输了
    Solution:
        首先如果对于奇数堆，那么先手必胜，因为可以构建必败态
        对于偶数的情况，如果是石子堆两两相同的对称局面，则为必败态，反之必胜
*/
string s[100010];
char tmp[10010];
int n;
int main(){
    scanf("%d",&n);
    if(n&1){puts("first player");return 0;}
    for(int i=1;i<=n;i++){scanf("%s",tmp);s[i]=tmp;}
    sort(s+1,s+n+1);
    for(int i=1;i<=n;i+=2)if(s[i]!=s[i+1]){puts("first player");return 0;}
    puts("second player");
    return 0;
}
```

7.10 斐波那契博弈

```
/*
    Fibonacci Game
        有一堆个数为 n 的石子，游戏双方轮流取石子，满足：
                （1）先手不能在第一次把所有的石子取完；
                （2）之后每次可以取的石子数介于1到对手刚取的石子数的2倍之间
                （包含1和对手刚取的石子数的2倍）。
        约定取走最后一个石子的人为赢家
        先手必胜：
        石头个数不为Fibonacci数
        即先手必败当且仅当石头个数为Fibonacci数。
*/
```

7.11 树上删边游戏

```
/*
    树上删边游戏
        给出一个有 n 个结点的树，有一个点作为树的根节点，
        双方轮流从树中删去一条边边，之后不与根节点相连的部分将被移走，无法操作者输.
    策略：
        叶子结点的SG值为0，其他结点SG值为其每个儿子结点SG值加1后的异或和
*/
int SG(int x){
    int sg=0;
    for(int i=0;i<v[x].size();i++)sg^=SG(v[x][i]);
        val[x]+=sg;
    return val[x];
}
int solve(){
    val[0]=0;
    for(int i=1;i<=n;i++)val[i]=1;
    puts(SG(0)?"Alice":"Bob");
}
```

7.12 先手必败局面数

```
/*
    求构造k个石子堆，每堆石子不超过n个，且先手必败的方案数(xor值为0)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
```

```
const LL mod=998244353;
LL pw[1010];
LL qpow(LL a,int b){
	LL t=1;
	while(b){
		if(b&1)t=t*a%mod;
		a=a*a%mod;
		b=b>>1;
	}
	return t;
}
int b,c,n,k;
LL inv(LL a,LL m){return(a==1?1:inv(m%a,m)*(m-m/a)%m);}
LL Cal(int n){
	if(n==0)return 1;
	b=floor(log(n)/log(2));
	c=n+1-pw[b];
	if(k&1)return (qpow(pw[b]+c,k)+qpow(pw[b]-c,k))%mod*inv(qpow(2,b+1),mod)%mod;
	else return
	((qpow(pw[b]+c,k)+qpow(pw[b]-c,k)-qpow(c,k)*2%mod+mod)%mod*inv(qpow(2,b+1),mod)%mod+Cal(c-1))%mod;
}
int main(){
	pw[0]=1;
	for(int i=1;i<=1000;i++)pw[i]=2LL*pw[i-1]%mod;
	scanf("%d%d",&k,&n);
	printf("%lld\n",Cal(n));
	return 0;
}
```

7.13 不平等博弈

```
/*
   Problem:
      游戏开始有四座塔，每座均由正方体叠成，所有正方体是黑色或者白色
      玩家L和R轮流操作，每次选定一个正方体，将正方体及其以上正方体全部拿走
      L玩家只能选择白色正方体，R玩家只能选择黑色正方体，不能操作者输
      如果L玩家无论先手或者后手都能赢，则称局面为W-configuration，
      定义子局面为三塔局面即为C，对于完整局面(C,T)，
      如果对于任意塔T，(C2,T)为W-configuration时，(C1,T)均为W-configuration
      则称C1不劣于C2，给定C1和C2，判断C1是否不劣于C2
   Solution:
      考虑一座塔的SN值，当塔为空时SN={|}=0
      如果塔包含一个白色正方体，则玩家L拥有可转移到0的决策，SN={0|}=1
      如果塔包含n个白色正方体，则SN={0,1,⋯,n-1|}=n
      同理塔包含n个黑色正方体时SN=-n
```

```
        当塔包含n个白色正方体和顶端一个黑色正方体时，SN={0,1,⋯,n-1|n}={n-1|n}=n-1/2
        如果包含n个白色正方体和顶端两个黑色正方体时，SN={n-1|n-1/2}=n-1/2-1/4
        在以上情况下在顶端再堆叠一个白色正方体，SN={n-1/2-1/4|n-1/2}=n-1/2-1/4+1/8
        结论就比较显然了，除去最底端的连续块，黑色方块-1/2^i，白色方块+1/2^i
*/
#include <cstdio>
#include <cstring>
using namespace std;
double getSN(int T[], int n) {
    double SN = 0;
    int i = 0;
    for (; i < n && T[i] == T[0]; i++) SN += T[i];
    for (double k = 2; i < n; i++, k = k * 2) SN += T[i] / k;
    return SN;
}
const int N = 60;
char s[N];
int T[N], d[3], Cas;
int main() {
    scanf("%d", &Cas);
    for (int cas = 1; cas <= Cas; cas++) {
        scanf("%s%s", s, s);
        for (int i = 0; i < 3; i++) scanf("%d", &d[i]);
        double SN1 = 0;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < d[i]; j++) {
                scanf("%s", s);
                T[j] = 2 * (s[0] == 'W') - 1;
            }
            SN1 += getSN(T, d[i]);
        }
        for (int i = 0; i < 3; i++) scanf("%d", &d[i]);
        double SN2 = 0;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < d[i]; j++) {
                scanf("%s", s);
                T[j] = 2 * (s[0] == 'W') - 1;
            }
            SN2 += getSN(T, d[i]);
        }
        if (SN1 >= SN2)
            printf("Test %d: Yes\n", cas);
        else
            printf("Test %d: No\n", cas);
    }
    return 0;
```

```
}
/*
    Problem:
        给定一个多个3*3的棋盘格，包含#,.,O和X
        Alice和Bob轮流对这些棋盘格做操作，当一方不能操作了算输
        Alice每次可以选择一个O，将其变成#
        同时，对以下三种事件挑选一种发生：[不能不选]
            1.选择的格子上下格子均变成#
            2.选择的格子左右格子均变成#
            3.上述两种事件同时发生
        Bob每次可以选择一个X，将其变成#，没有任何伴随事件
        如果Alice或者Bob始终能赢则对应输出Alice和Bob
        否则如果先手或者后手始终能赢则对应输出First和Second
        否则直接输出Others
    Solution:
        SN={Alice can get|Bob can get}
        非平等博弈只存在三种情况：
            1.SN>0：Alice总能够取胜
            2.SN<0：Bob总能取胜
            3.SN==0：后手总能取胜
        我们对$3*3$棋盘进行三进制状压
        暴力预处理出每种状态下的SN，Multi-SN情况将SN相加判断即可
*/
#include <bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < n; i++)
#define red(i, n) for (int i = n - 1; ~i; i--)
const int N = 27 * 27 * 27;
using ll = long long;
struct S {
    int x, y;
    S() {}
    S(int x, int y) : x(x), y(y) {}
    bool operator==(const S &b) const { return x == b.x && y == b.y; }
    bool operator<(const S &b) const { return x * b.y < y * b.x; }
    bool operator<=(const S &b) const { return *this < b || *this == b; }
};
S INF(0, -1), SN[N];
int st[9], nst[9];
auto msk = [&]() {
    int t = 0;
    red(i, 9) t = t * 3 + nst[i];
    return t;
};
auto getmid = [&](S &a, S &b) {
    ll down = max(a.y, b.y);
```

```
    ll up = a.x * (down / a.y) + b.x * (down / b.y);
    down <<= 1;
    while (up % 2 == 0) up /= 2, down /= 2;
    return S(up, down);
};
int Decode(int x) { rep(i, 9) st[i] = x % 3, x /= 3; }
void Init() {
    rep(i, N) {
        S L = INF, R = INF;
        Decode(i);
        auto updL = [&]() {
            S s = SN[msk()];
            if (L == INF || L < s) L = s;
        };
        auto updR = [&]() {
            S s = SN[msk()];
            if (R == INF || s < R) R = s;
        };
        rep(j, 9) if (st[j] == 1) {
            int x = j / 3, y = j % 3;
            rep(k, 9) nst[k] = st[k];
            nst[j] = 0;
            if (x) nst[j - 3] = 0;
            if (x != 2) nst[j + 3] = 0;
            updL();
            if (y) nst[j - 1] = 0;
            if (y != 2) nst[j + 1] = 0;
            updL();
            if (x) nst[j - 3] = st[j - 3];
            if (x != 2) nst[j + 3] = st[j + 3];
            updL();
        }
        rep(j, 9) if (st[j] == 2) {
            int x = j / 3, y = j % 3;
            rep(k, 9) nst[k] = st[k];
            nst[j] = 0;
            updR();
        }
        if (L == INF && R == INF)
            SN[i] = S(0, 1);
        else if (L == INF)
            SN[i] = S(R.x - 1, 1);
        else if (R == INF)
            SN[i] = S(L.x + 1, 1);
        else {
            assert(L < R);
```

```
            S l = INF, r = INF, x(0, 1);
            while (R <= x || x <= L) {
                if (R <= x) {
                    r = x;
                    if (l == INF)
                        x.x--;
                    else
                        x = getmid(l, r);
                } else {
                    l = x;
                    if (r == INF)
                        x.x++;
                    else
                        x = getmid(l, r);
                }
            }
            SN[i] = x;
        }
    }
}
int Tr(char c) {
    if (c == 'O') return 1;
    if (c == 'X') return 2;
    return 0;
}
int T, n;
char s[10];
int main() {
    Init();
    scanf("%d", &T);
    while (T--) {
        int sn = 0;
        scanf("%d", &n);
        while (n--) {
            string b;
            rep(i, 3) {
                scanf("%s", s);
                rep(j, 3) b += s[j << 1];
            }
            int msk = 0;
            red(i, 9) msk = msk * 3 + Tr(b[i]);
            sn += SN[msk].x * (64 / SN[msk].y);
        }
        if (sn == 0)
            puts("Second");
        else if (sn > 0)
```

```
            puts("Alice");
        else
            puts("Bob");
    }
    return 0;
}
```

8 算法

8.1 DancingLinks

```
/*
    精确覆盖问题：给定一个由0-1组成的矩阵，是否能找到一个行的集合，使得集合中每一列都恰好包含一个1
    In_Test:
        4 4
        2 1 3 （行中1的个数和位置）
        1 2
        1 1
        2 3 4
    Out_Test:
        3 2 4 3 （找到的那个集合大小和集合输出）
*/
#include <cstdio>
const int N = 100100;
struct DLX {
    int L[N], R[N], U[N], D[N], row[N], col[N];
    int H[1010];
    int ansd, ans[1010];
    int s[1010];
    int tot, n, m;
    void init(int _n, int _m) {
        n = _n;
        m = _m;
        tot = m;
        for (int i = 0; i <= m; i++)
            L[i] = i - 1, R[i] = i + 1, U[i] = i, D[i] = i, s[i] = 0;
        L[0] = m;
        R[m] = 0;
        for (int i = 1; i <= n; i++) H[i] = -1;
    }
    void link(int i, int j) {
        s[j]++;
        ++tot;
        row[tot] = i;
        col[tot] = j;
        U[tot] = U[j];
        D[tot] = j;
        D[U[j]] = tot;
        U[j] = tot;
        if (H[i] < 0) {
            H[i] = tot;
            L[tot] = tot;
            R[tot] = tot;
```

```
        } else {
            L[tot] = L[H[i]];
            R[tot] = H[i];
            R[L[H[i]]] = tot;
            L[H[i]] = tot;
        }
    }
    void remove(int c) {
        R[L[c]] = R[c];
        L[R[c]] = L[c];
        for (int i = D[c]; i != c; i = D[i])
            for (int j = R[i]; j != i; j = R[j]) {
                s[col[j]]--;
                D[U[j]] = D[j];
                U[D[j]] = U[j];
            }
    }
    void resume(int c) {
        R[L[c]] = c;
        L[R[c]] = c;
        for (int i = U[c]; i != c; i = U[i])
            for (int j = L[i]; j != i; j = L[j]) {
                s[col[j]]++;
                D[U[j]] = j;
                U[D[j]] = j;
            }
    }
    bool dfs(int d) {
        if (R[0] == 0) {
            ansd = d - 1;
            return 1;
        }
        int c = R[0];
        for (int i = R[0]; i != 0; i = R[i])
            if (s[i] < s[c]) c = i;
        remove(c);
        for (int i = D[c]; i != c; i = D[i]) {
            ans[d] = row[i];
            for (int j = R[i]; j != i; j = R[j]) remove(col[j]);
            if (dfs(d + 1)) return 1;
            for (int j = L[i]; j != i; j = L[j]) resume(col[j]);
        }
        resume(c);
        return 0;
    }
} g;
```

```
int main() {
    int n, m, k, x;
    while (~scanf("%d%d", &n, &m)) {
        g.init(n, m);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &k);
            for (int j = 1; j <= k; j++) scanf("%d", &x), g.link(i, x);
        }
        if (g.dfs(1)) {
            printf("%d", g.ansd);
            for (int i = 1; i <= g.ansd; i++) printf(" %d", g.ans[i]);
            printf("\n");
        } else
            puts("NO\n");
    }
    return 0;
}
/*
    重复覆盖问题：给定一个由0-1组成的矩阵，是否能找到k行，使得集合中每一列都至少包含一个1
    Problem: n城市，建k机场使得，使得每个城市最近机场的距离的最大值最小化，输出距离
    Solution: 二分+DLX检测
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
typedef long long ll;
const int N = 65, M = 65, V = 65 * 65;
int n, k;
struct DLX {
    int L[V], R[V], U[V], D[V], row[V], col[V];
    int H[N];
    int s[M];
    int tot, n, m;
    bool vis[M];
    void init(int _n, int _m) {
        n = _n;
        m = _m;
        tot = m;
        for (int i = 0; i <= m; i++)
            L[i] = i - 1, R[i] = i + 1, U[i] = i, D[i] = i, s[i] = 0;
        L[0] = m;
        R[m] = 0;
        for (int i = 1; i <= n; i++) H[i] = -1;
    }
    void link(int i, int j) {
```

```
    s[j]++;
    ++tot;
    row[tot] = i;
    col[tot] = j;
    U[tot] = U[j];
    D[tot] = j;
    D[U[j]] = tot;
    U[j] = tot;
    if (H[i] < 0) {
        H[i] = tot;
        L[tot] = tot;
        R[tot] = tot;
    } else {
        L[tot] = L[H[i]];
        R[tot] = H[i];
        R[L[H[i]]] = tot;
        L[H[i]] = tot;
    }
}
void remove(int c) {
    for (int i = D[c]; i != c; i = D[i]) R[L[i]] = R[i], L[R[i]] = L[i];
}
void resume(int c) {
    for (int i = U[c]; i != c; i = U[i]) R[L[i]] = L[R[i]] = i;
}
int h() {
    int ret = 0;
    for (int c = R[0]; c != 0; c = R[c]) {
        if (D[c] == c) return n + 1;
        vis[c] = 0;
    }
    for (int c = R[0]; c != 0; c = R[c])
        if (!vis[c]) {
            ret++;
            vis[c] = 1;
            for (int i = D[c]; i != c; i = D[i])
                for (int j = R[i]; j != i; j = R[j]) vis[col[j]] = 1;
        }
    return ret;
}
bool dfs(int d) {
    if (d + h() > k) return 0;
    if (R[0] == 0) return 1;
    int c = R[0];
    for (int i = R[0]; i != 0; i = R[i])
        if (s[i] < s[c]) c = i;
```

```
        for (int i = D[c]; i != c; i = D[i]) {
            remove(i);
            for (int j = R[i]; j != i; j = R[j]) remove(j);
            if (dfs(d + 1)) return 1;
            for (int j = L[i]; j != i; j = L[j]) resume(j);
            resume(i);
        }
        return 0;
    }
} g;
struct P {
    int x, y;
    ll operator-(const P &t) { return 0ll + abs(x - t.x) + abs(y - t.y); }
} a[N];
bool check(ll len) {
    g.init(n, n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (a[i] - a[j] <= len) g.link(i, j);
    return g.dfs(0);
}
ll dis[N * N];
int main() {
    int t;
    scanf("%d", &t);
    for (int cas = 1; cas <= t; cas++) {
        scanf("%d%d", &n, &k);
        for (int i = 1; i <= n; i++) scanf("%d%d", &a[i].x, &a[i].y);
        int cnt = 0;
        for (int i = 1; i < n; i++)
            for (int j = i + 1; j <= n; j++) dis[++cnt] = a[i] - a[j];
        dis[++cnt] = 0;
        sort(dis + 1, dis + cnt + 1);
        int l = 1, r = cnt + 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (!check(dis[mid]))
                l = mid + 1;
            else
                r = mid;
        }
        printf("Case #%d: %lld\n", cas, dis[l]);
    }
    return 0;
}
```

8.2 整体二分

```
/*
    整体二分
    Example:
        有N个成员国。现在它发现了一颗新的星球，
        这颗星球的轨道被分为M份（第M份和第1份相邻），第i份上有第Ai个国家的太空站。
        这个星球经常会下陨石雨。BIU已经预测了接下来K场陨石雨的情况。
        BIU的第i个成员国希望能够收集Pi单位的陨石样本。
        你的任务是判断对于每个国家，它需要在第几次陨石雨之后，才能收集足够的陨石。
    Solution:
        如果枚举每场陨石雨，树状数组查询每个国家是否完成收集，
        复杂度O(Kmlogm),难以接受
        因此我们对答案整体二分，不同位置的答案不断划分到对应的陨石时段计算。
        将完成任务的国家放到左边区间，未达成的放到右边区间。复杂度O(mlogmlogK).
*/
const int N = 300010;
typedef long long LL;
int x, n, m, T, K, l[N], r[N], val[N], id[N], p[N], tmp[N], ans[N], mark[N];
LL c[N];
vector<int> v[N];
LL query(int x) {
    LL s = 0;
    while (x) s += c[x], x -= x & -x;
    return s;
}
void add(int x, int val) {
    while (x <= m) c[x] += val, x += x & -x;
}
void update(int x, int k) {
    if (l[x] <= r[x])
        add(l[x], k * val[x]), add(r[x] + 1, -k * val[x]);
    else {
        add(1, k * val[x]), add(r[x] + 1, -k * val[x]);
        add(l[x], k * val[x]);
    }
}
void solve(int l, int r, int L, int R) {
    if (l > r) return;
    if (L == R) {
        for (int i = l; i <= r; i++) ans[id[i]] = L;
        return;
    }
    int mid = (L + R) >> 1;
    while (T <= mid) update(++T, 1);
    while (T > mid) update(T--, -1);
```

```
    int cnt = 0, x;
    LL tot;
    for (int i = l; i <= r; i++) {
        tot = 0;
        x = id[i];
        for (int j = 0; j < v[x].size(); j++) {
            tot += query(v[x][j]);
            if (tot >= p[x]) break;
        }
        if (tot >= p[x])
            mark[x] = 1, cnt++;
        else
            mark[x] = 0;
    }
    int l1 = l, l2 = l + cnt;
    for (int i = l; i <= r; i++) {
        if (mark[id[i]])
            tmp[l1++] = id[i];
        else
            tmp[l2++] = id[i];
    }
    for (int i = l; i <= r; i++) id[i] = tmp[i];
    solve(l, l1 - 1, L, mid);
    solve(l1, l2 - 1, mid + 1, R);
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        // 记录每个国家都有哪些太空站
        for (int i = 1; i <= m; i++) scanf("%d", &x), v[x].push_back(i);
        for (int i = 1; i <= n; i++) scanf("%d", &p[i]);
        scanf("%d", &K);
        for (int i = 1; i <= K; i++) scanf("%d%d%d", &l[i], &r[i], &val[i]);
        l[++K] = 1;
        r[K] = m;
        val[K] = 0x3f3f3f3f;
        for (int i = 1; i <= n; i++) id[i] = i;
        solve(1, n, 1, K);
        for (int i = 1; i <= n; i++) {
            if (ans[i] == K)
                puts("NIE");
            else
                printf("%d\n", ans[i]);
        }
    }
    return 0;
}
```

```
/*
    Example:
        单点修改，区间求第k大
    Solution:
        我们对询问和修改进行整体二分，分治第k小的数值，在当前层完成分治数值以下的修改和查询，
        我们用下标权值树状数组维护数值，当一个数字满足要求时在其下标位置进行修改操作，
        查询只需查询区间的数字数量是否满足k个即可判定询问的分治区间
        我们按照数值划分修改操作，按照是否满足区间存在至少k个值来划分询问，
        最后分到各个叶节点的询问答案即分治层的数值
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 200010, INF = 0x3f3f3f3f;
int c[N], a[N], ans[N];
int T, n, m;
void add(int x, int val) {
    while (x <= n) c[x] += val, x += x & -x;
}
int query(int x) {
    int s = 0;
    while (x) s += c[x], x -= x & -x;
    return s;
}
struct data {
    int x, y, k, id, isq;
} q[N], ql[N], qr[N];
void solve(int l, int r, int L, int R) {
    if (l > r) return;
    if (L == R) {
        for (int i = l; i <= r; i++) {
            if (q[i].isq) ans[q[i].id] = L;
        }
        return;
    }
    int mid = (L + R) >> 1;
    int p1 = 0, p2 = 0;
    for (int i = l; i <= r; i++) {
        if (!q[i].isq) {
            if (q[i].x <= mid) {
                add(q[i].id, q[i].y);
                ql[p1++] = q[i];
            } else
                qr[p2++] = q[i];
        } else {
```

```
            int cnt = query(q[i].y) - query(q[i].x - 1);
            if (cnt >= q[i].k)
                ql[p1++] = q[i];
            else {
                q[i].k -= cnt;
                qr[p2++] = q[i];
            }
        }
    }
    for (int i = 0; i < p1; i++) {
        if (!ql[i].isq) add(ql[i].id, -ql[i].y);
        q[l + i] = ql[i];
    }
    for (int i = 0; i < p2; i++) q[l + p1 + i] = qr[i];
    solve(l, l + p1 - 1, L, mid);
    solve(l + p1, r, mid + 1, R);
}
int main() {
    scanf("%d", &T);
    while (T--) {
        memset(c, 0, sizeof(c));
        scanf("%d%d", &n, &m);
        int cnt = 0, cntq = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            q[++cnt] = (data){a[i], 1, INF, i, 0};
        }
        for (int i = 1; i <= m; i++) {
            int x, y, k;
            char op[10];
            scanf("%s%d%d", op, &x, &y);
            if (op[0] == 'Q') {
                scanf("%d", &k);
                q[++cnt] = (data){x, y, k, ++cntq, 1};
            } else {
                q[++cnt] = (data){a[x], -1, INF, x, 0};
                a[x] = y;
                q[++cnt] = (data){y, 1, INF, x, 0};
            }
        }
        solve(1, cnt, -INF, INF);
        for (int i = 1; i <= cntq; i++) printf("%d\n", ans[i]);
    }
    return 0;
}
```

8.3 CDQ 分治

```
/*
    CDQ分治
    三维数点
    将区间整体按x排序,
    在每个区间完成操作将区间按照y排序,
    那么两个合并的区间可以保证每个区间y是单调递增的
    而显然右区间x要大于左区间,
    所以只要将y小于右区间当前统计点的左区间的点的z坐标全部加入数据结构
    然后在数据结构上进行相应的统计就可以了。
*/
const int N = 100005, M = 100001;
struct node {
    int x, y, z, t;
} q[N];
bool cmpx(node a, node b) { return a.x < b.x; }
bool cmpy(node a, node b) { return a.y < b.y; }
void read(int& a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
int n, ans[N], T, s[N], c[N];
int add(int x) {
    while (x < M) {
        if (c[x] != T) c[x] = T, s[x] = 0;
        s[x]++, x += x & -x;
    }
}
int query(int x) {
    int sum = 0;
    while (x) {
        if (c[x] != T) c[x] = T, s[x] = 0;
        sum += s[x], x -= x & -x;
    }
    return sum;
}
void CDQ(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1, tmp = q[mid].x;
    CDQ(l, mid);
    CDQ(mid + 1, r);
    T++;
```

```
    int L = l, R = mid + 1;
    while (L <= mid || R <= r) {
        if (L > mid)
            ans[q[R].t] += query(q[R].z), R++;
        else if (R > r)
            add(q[L].z), L++;
        else if (q[L].y <= q[R].y)
            add(q[L].z), L++;
        else
            ans[q[R].t] += query(q[R].z), R++;
    }
    T++;
    L = l, R = mid + 1;
    while (L <= mid || R <= r) {
        if (L > mid) {
            if (q[R].x == tmp) add(q[R].z);
            R++;
        } else if (R > r) {
            if (q[L].x == tmp) ans[q[L].t] += query(q[L].z);
            L++;
        } else if (q[L].y >= q[R].y) {
            if (q[R].x == tmp) add(q[R].z);
            R++;
        } else {
            if (q[L].x == tmp) ans[q[L].t] += query(q[L].z);
            L++;
        }
    }
    sort(q + l, q + r + 1, cmpy);
}
void init() {
    T = 0;
    memset(c, 0, sizeof(c));
    memset(ans, 0, sizeof(ans));
}
int main(int cas) {
    read(cas);
    while (cas--) {
        read(n);
        init();
        for (int i = 1; i <= n; i++) {
            read(q[i].x), read(q[i].y), read(q[i].z);
            q[i].t = i;
        }
        sort(q + 1, q + n + 1, cmpx);
        CDQ(1, n);
```

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

8.4 动态 CDQ 分治

```
/*
    动态CDQ分治
    Example:
        给一张二维表，要求支持两种操作，查询矩阵和以及增加单个格子的数值
    Solution:
        对于区间询问拆点
        1.读入询问，按x排序
        2.将[L,R]中的数分为前部分操作，后部分操作（各部分仍保持X升序）
        3.将前面对后面的影响记录ans
        4.复原影响
        5.递归[L,M],[M+1,R]
*/
int m, x1, x2, y1, y2, u, opt, ans[200010], c[500010], M;
struct data {
    int x, y, val, pos, id, opt;
} q[800010], tmp[800010];
bool operator<(data a, data b) {
    if (a.x == b.x && a.y == b.y)
        return a.opt < b.opt;
    else
        return (a.x == b.x ? a.y < b.y : a.x < b.x);
}
void addquery() {
    scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
    int pos = ++ans[0];
    ans[pos] = (x2 - x1 + 1) * (y2 - y1 + 1) * u;
    q[++m].pos = pos;
    q[m].id = m;
    q[m].x = x1 - 1;
    q[m].y = y1 - 1;
    q[m].val = 1;
    q[m].opt = 1;
    q[++m].pos = pos;
    q[m].id = m;
    q[m].x = x2;
    q[m].y = y2;
    q[m].val = 1;
    q[m].opt = 1;
```

```
    q[++m].pos = pos;
    q[m].id = m;
    q[m].x = x1 - 1;
    q[m].y = y2;
    q[m].val = -1;
    q[m].opt = 1;
    q[++m].pos = pos;
    q[m].id = m;
    q[m].x = x2;
    q[m].y = y1 - 1;
    q[m].val = -1;
    q[m].opt = 1;
}
void addmodify() {
    ++m;
    scanf("%d%d%d", &q[m].x, &q[m].y, &q[m].val);
    q[m].id = m;
}
void update(int x, int val) {
    while (x <= M) c[x] += val, x += x & -x;
}
int query(int x) {
    int s = 0;
    while (x) s += c[x], x -= x & -x;
    return s;
}
void CDQ(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1, l1 = l, l2 = mid + 1;
    for (int i = l; i <= r; i++) {
        if (q[i].id <= mid && !q[i].opt) update(q[i].y, q[i].val);
        if (q[i].id > mid && q[i].opt)
            ans[q[i].pos] += q[i].val * query(q[i].y);
    }
    for (int i = l; i <= r; i++)
        if (q[i].id <= mid && !q[i].opt) update(q[i].y, -q[i].val);
    for (int i = l; i <= r; i++)
        if (q[i].id <= mid)
            tmp[l1++] = q[i];
        else
            tmp[l2++] = q[i];
    for (int i = l; i <= r; i++) q[i] = tmp[i];
    CDQ(l, mid);
    CDQ(mid + 1, r);
}
int main() {
```

```
    scanf("%d", &M);
    while (1) {
        scanf("%d", &opt);
        if (opt == 1)
            addmodify();
        else if (opt == 2)
            addquery();
        else
            break;
    }
    sort(q + 1, q + m + 1);
    CDQ(1, m);
    for (int i = 1; i <= ans[0]; i++) printf("%d\n", ans[i]);
    return 0;
}
```

8.5 二维 LIS

```
/*
    二维最长上升子序列（严格单调）
        一维排序（区间按照a进行排序，a相同情况下标大的在前）
        二维CDQ（下标划分左右区间，左边更新右边统计）
        三维BIT（以b为下标，用统计值dp对BIT更新，b值域离散化）
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int n, a[N], b[N], c[N], d[N], dp[N], p[N], q[N];
bool cmp(int x, int y) {
    if (a[x] == a[y]) return x > y;
    return a[x] < a[y];
}
void CDQ(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    CDQ(l, mid);
    for (int i = l; i <= r; i++) q[i] = i;
    sort(q + l, q + r + 1, cmp);
    for (int i = l; i <= r; i++) {
        if (q[i] <= mid)
            for (int j = b[q[i]]; j <= p[0]; j += j & -j)
                c[j] = max(c[j], dp[q[i]]);
        else
            for (int j = b[q[i]] - 1; j; j -= j & -j)
```

```
            dp[q[i]] = max(dp[q[i]], c[j] + 1);
    }
    for (int i = l; i <= r; i++)
        if (q[i] <= mid)
            for (int j = b[q[i]]; j <= p[0]; j += j & -j) c[j] = 0;
    CDQ(mid + 1, r);
}
int main() {
    while (~scanf("%d", &n)) {
        p[0] = 0;
        for (int i = 1; i <= n; i++)
            scanf("%d%d", &a[i], &b[i]), p[++p[0]] = b[i], dp[i] = 1;
        sort(p + 1, p + p[0] + 1);
        p[0] = unique(p + 1, p + p[0] + 1) - p - 1;
        for (int i = 1; i <= n; i++)
            b[i] = lower_bound(p + 1, p + p[0] + 1, b[i]) - p;
        CDQ(1, n);
        int ans = 0;
        for (int i = 1; i <= n; i++) ans = max(ans, dp[i]);
        printf("%d\n", ans);
    }
    return 0;
}
```

8.6 枚举子集

```
/*
    枚举子集
*/
// ====================================================
for (int i = 1; i < all; i++)
    for (int j = i; j; j = i & (j - 1))
// ====================================================
```

8.7 斜率优化 DP

```
/*
    Example:
        n个牧场排成一行，需要在某些牧场上面建立控制站，
        每个牧场上只能建立一个控制站，每个控制站控制的牧场
        是它所在的牧场一直到它西边第一个控制站的所有牧场
        它西边第一个控制站所在的牧场不被控制，如果它西边不存在控制站，
        那么它控制西边所有的牧场，每个牧场被控制都需要一定的花费，
```

```
        而且该花费等于它到控制它的控制站之间的牧场数目乘上该牧场的放养量，
        在第i个牧场建立控制站的花费是ai，每个牧场i的放养量是bi，
        求最小总花费。
    Solution:
        考虑只在n建立控制站的情况，答案为 b[i]*(n-i)
        记dp[i]为考虑了i到n的情况，并且在i点建立了控制站的最优情况，
        有dp[i]=max{dp[j]+sum[i]*(j-i)}-a[i]
        =-a[i]-sum[i]*i+max(dp[j]+sum[i]*j)
        为关于sum[i]的线性函数，那么对f(y)=x*y+dp[x]进行斜率优化
*/
using namespace std;
typedef long long LL;
const int N = 1000010;
int deq[N], n;
LL tot, ans, dp[N], S[N], a[N], b[N];
LL f(int x, LL y) { return y * x + dp[x]; }
bool check(int f1, int f2, int f3) {
    LL a1 = f1, b1 = dp[f1];
    LL a2 = f2, b2 = dp[f2];
    LL a3 = f3, b3 = dp[f3];
    return (a2 - a1) * (b3 - b2) <= (b2 - b1) * (a3 - a2);
}
void solve() {
    for (int i = 1; i <= n; i++) S[i] = S[i - 1] + b[i];
    int s = 0, t = 1;
    deq[0] = n;
    for (int i = n - 1; i; i--) {
        while (s + 1 < t && f(deq[s], S[i]) <= f(deq[s + 1], S[i])) s++;
        dp[i] = -a[i] - S[i] * i + f(deq[s], S[i]);
        ans = max(ans, dp[i]);
        while (s + 1 < t && check(deq[t - 2], deq[t - 1], i)) t--;
        deq[t++] = i;
    }
    printf("%lld\n", tot - ans);
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    for (int i = 1; i <= n; i++) scanf("%lld", &b[i]);
    for (int i = 1; i < n; i++) tot += b[i] * (n - i);
    tot += a[n];
    solve();
    return 0;
}
/*
    Example:
```

```
        N个任务排成一个序列在一台机器上等待完成（顺序不得改变），
        这N个任务被分成若干批，每批包含相邻的若干任务。
        从时刻0开始，这些任务被分批加工，第i个任务单独完成所需的时间是Ti。
        在每批任务开始前，机器需要启动时间S，
        而完成这批任务所需的时间是各个任务需要时间的总和（同一批任务将在同一时刻完成）。
        每个任务的费用是它的完成时刻乘以一个费用系数Fi。
        请确定一个分组方案，使得总费用最小
    Solution:
        我们可以得到dp方程dp[i]=min{dp[j]+(sumT[i]-sumT[j]+S)*(sumF[n]-sumF[j])}
        dp[i]=(sumF[n]-sumF[j])*sumT[i]+dp[j]+(S-sumT[j])*(sumF[n]-sumF[j])
        是关于sumT[i]的一元一次方程，我们对此做斜率优化。
*/
typedef long long LL;
const int MAX_N = 10010;
int n;
LL k, a[MAX_N], b[MAX_N], dp[MAX_N], S[MAX_N], F[MAX_N], deq[MAX_N];
LL f(int x, int y) {
    return (F[n] - F[x]) * S[y] + dp[x] + (k - S[x]) * (F[n] - F[x]);
}
bool check(int f1, int f2, int f3) {
    LL a1 = F[n] - F[f1], b1 = dp[f1] + (k - S[f1]) * (F[n] - F[f1]);
    LL a2 = F[n] - F[f2], b2 = dp[f2] + (k - S[f2]) * (F[n] - F[f2]);
    LL a3 = F[n] - F[f3], b3 = dp[f3] + (k - S[f3]) * (F[n] - F[f3]);
    return (a2 - a1) * (b3 - b2) >= (b2 - b1) * (a3 - a2);
}
void solve() {
    for (int i = 0; i < n; i++) S[i + 1] = S[i] + a[i], F[i + 1] = F[i] + b[i];
    int s = 0, t = 1;
    deq[0] = 0;
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        while (s + 1 < t && f(deq[s], i) >= f(deq[s + 1], i)) s++;
        dp[i] = f(deq[s], i);
        while (s + 1 < t && check(deq[t - 2], deq[t - 1], i)) t--;
        deq[t++] = i;
    }
    printf("%lld\n", dp[n]);
}
int main() {
    while (~scanf("%d%lld", &n, &k)) {
        for (int i = 0; i < n; i++) scanf("%lld%lld", &a[i], &b[i]);
        solve();
    }
    return 0;
}
/*
```

```
    Example:
        给出一个长度为n个非严格单调递增数列，每次操作可以使得其中任意一项减一，
        问现在使得数列中每项数相同的数的数量都大于等于k-1，问最少进行几次操作
    Solution:
        我们设dp[i]为前i项答案，得到方程dp[i]=min(dp[j]+S[i]-S[j]-aj*(i-j)),
        dp[i]=S[i]+min(dp[j]-S[j]+aj*j-aj*i)，为关于i的线性函数，
        所以我们对f(y)=-ax*y+dp[x]-S[x]+ax*x进行斜率优化。
*/
typedef long long LL;
const int MAX_N = 500010;
int n, k;
LL a[MAX_N], dp[MAX_N], S[MAX_N], deq[MAX_N];
LL f(int x, int y) { return -a[x] * y + dp[x] - S[x] + a[x] * x; }
bool check(int f1, int f2, int f3) {
    LL a1 = -a[f1], b1 = dp[f1] - S[f1] + a[f1] * f1;
    LL a2 = -a[f2], b2 = dp[f2] - S[f2] + a[f2] * f2;
    LL a3 = -a[f3], b3 = dp[f3] - S[f3] + a[f3] * f3;
    return (a2 - a1) * (b3 - b2) >= (b2 - b1) * (a3 - a2);
}
void solve() {
    for (int i = 0; i < n; i++) S[i + 1] = S[i] + a[i];
    int s = 0, t = 1;
    deq[0] = 0;
    dp[0] = 0;
    for (int i = k; i <= n; i++) {
        if (i - k >= k) {
            while (s + 1 < t && check(deq[t - 2], deq[t - 1], i - k)) t--;
            deq[t++] = i - k;
        }
        while (s + 1 < t && f(deq[s], i) >= f(deq[s + 1], i)) s++;
        dp[i] = S[i] + f(deq[s], i);
    }
    printf("%lld\n", dp[n]);
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &k);
        for (int i = 0; i < n; i++) scanf("%lld", &a[i]);
        solve();
    }
    return 0;
}
```

8.8 DP 套 DP

```
/*
    Dp套Dp
    给出一个矩阵求至少删去一个元素之后的最大子矩阵
*/
const int INF = 0x3f3f3f3f, N = 310;
int n, m, P;
int dp[N][2], s[N][2], a[N][N];
int DP(int x, int y) {
    int res = -INF, st = 0;
    dp[0][1] = dp[0][0] = -INF;
    for (int i = 1; i <= m; i++) {
        dp[i][1] =
            max(max(dp[i - 1][1] + s[i][0], s[i][1]), dp[i - 1][0] + s[i][1]);
        if (dp[i - 1][0] < 0) {
            st = i;
            dp[i][0] = s[i][0];
        } else
            dp[i][0] = dp[i - 1][0] + s[i][0];
        if (x == 1 && y == n && i == m && st == 1)
            res = max(res, dp[i][1]);
        else
            res = max(res, max(dp[i][0], dp[i][1]));
    }
    return res;
}
int Dp(int n, int m) {
    int ans = -INF;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) s[j][1] = -INF, s[j][0] = 0;
        for (int j = i; j <= n; j++) {
            for (int k = 1; k <= m; k++) {
                s[k][1] = max(s[k][1] + a[j][k], s[k][0] + P);
                s[k][0] += a[j][k];
            }
            ans = max(ans, DP(i, j));
        }
    }
    return ans;
}
int main() {
    while (~scanf("%d%d%d", &n, &m, &P)) {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) scanf("%d", &a[i][j]);
        printf("%d\n", Dp(n, m));
```

```
    }
    return 0;
}
```

8.9 数位 DP

```
/*
    数位DP
    Problem:
        一个数被称之为平衡数当且仅当其为奇数的数位每个数字出现次数为偶数，
        偶数的数位每个数字出现次数为奇数，询问[L,R]之间有多少平衡数
    Solution:
        用三进制表示每个数位的状态，0为该数字没有出现过，1为出现奇数次，2为出现偶数次，
        用dp[i][j]保存剩下未填充的位为i位，当前得到状态为j且接下来没有数位限制时的方案数，
        记忆化数位dp即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long unsigned LL;
LL dp[20][60000], L, R;
int n, bit[20], pw[20], b[60000];
bool Check(int s) {
    for (int i = 0; i <= 9; i++, s /= 3) {
        if (i & 1 && s % 3 == 1)
            return 0;
        else if (!(i & 1) && s % 3 == 2)
            return 0;
    }
    return 1;
}
int Cal(int i, int s) {
    int p = s / pw[i] % 3;
    if (p < 2)
        s += pw[i];
    else
        s -= pw[i];
    return s;
}
LL DP(int i, int j, int lim, int z) { // lim: 数位大小限制 | z: 是否有前导0
    if (!i) return b[j];
    if (!lim && ~dp[i][j]) return dp[i][j];
    LL ans = 0;
    int ed = lim ? bit[i] : 9;
    for (int k = 0; k <= ed; k++)
        ans += DP(i - 1, (z && !k) ? 0 : Cal(k, j), lim && k == ed, z && !k);
```

```
    if (!lim) dp[i][j] = ans;
    return ans;
}
LL Solve(LL s, int len = 0) {
    for (; s; s /= 10) bit[++len] = s % 10;
    return DP(len, 0, 1, 1);
}
int main() {
    memset(dp, -1, sizeof(dp));
    pw[0] = 1;
    for (int i = 1; i <= 10; i++) pw[i] = pw[i - 1] * 3;
    for (int i = 0; i < pw[10]; i++) b[i] = Check(i);
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%llu%llu", &L, &R);
        printf("%llu\n", Solve(R) - Solve(L - 1));
    }
    return 0;
}
```

8.10 悬线法

```
/*
    悬线法
    求出最大全1矩阵的面积
    在处理每个点的时候，继承上一个点等高度下的左右最大扩展，
    计算在该层的左右最大扩展，然后对于每个点更新答案即可。
*/
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 2010;
int i, j, n, m, ans, l[N], r[N], h[N], lmax, rmax, a[N][N];
int main() {
    while (~scanf("%d%d", &n, &m)) {
        ans = 0;
        memset(h, 0, sizeof(h));
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) scanf("%d", &a[i][j]);
        for (int i = 1; i <= m; i++) l[i] = 1, r[i] = m;
        for (int i = 1; i <= n; i++) {
            for (lmax = j = 1; j <= m; j++)
                if (a[i][j]) {
                    h[j]++;
```

```
                if (lmax > l[j]) l[j] = lmax;
            } else
                h[j] = 0, l[j] = 1, r[j] = m, lmax = j + 1;
        for (rmax = j = m; j; j--)
            if (a[i][j]) {
                if (rmax < r[j]) r[j] = rmax;
                if ((r[j] - l[j] + 1) * h[j] > ans)
                    ans = (r[j] - l[j] + 1) * h[j];
            } else
                rmax = j - 1;
      }
      printf("%d\n", ans);
   }
   return 0;
}
/*
   Example: 给出一个数字矩阵，求子矩阵期望数字种数
   Solution: 我们统计[x,y]为其所表示的数字的最左上方的矩形数量，即该格子的贡献值，
   我们用悬线法将上边界上移，调整左右边界保证[x,y]为数字左上边界，
   那么x及其以下的部分都可以作为下边界，顺序统计即可，
   由于障碍点的顺序设置保证了二维的偏序性，不会出现重复统计贡献。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int N = 110;
int n, m, c, l[N], r[N], row[N * N][N];
LL Cal(int x, int y, int c) {
   LL res = 0;
   for (int i = 1; i <= x; i++) l[i] = 0, r[i] = m + 1;
   for (int i = 1; i < y; i++) l[row[c][i]] = i;
   for (int i = m; i > y; i--) r[row[c][i]] = i;
   int h = row[c][y];
   for (int i = x - 1; i > h; i--)
      l[i] = max(l[i], l[i + 1]), r[i] = min(r[i], r[i + 1]);
   for (int i = x; i > h; i--)
      res += (LL)(r[i] - y) * (y - l[i]) * (n - x + 1);
   return res;
}
int T;
int main() {
   scanf("%d", &T);
   while (T--) {
      memset(row, 0, sizeof(row));
```

```
        scanf("%d%d", &n, &m);
        LL ans = 0, tot = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                scanf("%d", &c);
                ans += Cal(i, j, c);
                row[c][j] = i;
                tot += 1LL * i * j;
            }
        }
        printf("%.9f\n", 1.0 * ans / tot);
    }
    return 0;
}
```

8.11 IMOS 法

```
/*
    imos法
    Example: 给出一个nxm的矩形方格图，给出每个矩形障碍块的左上角和右下角，
    多次询问，每次给出一个矩形，问放入这个有障碍的方格图有几种方案
    Solution: 通过二维差分计算出可以放的区域，枚举每一行作为底边，维护高度的单调栈，
    当每个元素出栈的时候维护包含位置的最大矩形，这些最大矩形对于长宽不超过自己的询问贡献均为1，
    因为最大矩形是二维imos差分后位置的区间修改(一段-1和一段+1)，所以可以继续差分为单点修改，
    最后计算二维差分的前缀和就可以得到答案的预处理数组。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 2010;
int n, m, u, q;
int top, h[N], st[N], mp[N][N], ans[N][N];
void doit(int x, int y, int z, int l) {
    ans[l][0]--;
    ans[l][y - x + 1]++;
    ans[l][z - y + 1]++;
    ans[l][z - x + 1 + 1]--;
}
int main() {
    scanf("%d%d%d%d", &n, &m, &u, &q);
    for (int i = 1; i <= u; i++) {
        int x1, y1, x2, y2;
        scanf("%d%d%d%d", &x1, &x2, &y1, &y2);
        mp[x1][y1]++;
        mp[x2][y2]++;
```

```
        mp[x1][y2]--;
        mp[x2][y1]--;
    }
    for (int i = n; i; i--)
        for (int j = m; j; j--)
            mp[i][j] += mp[i + 1][j] + mp[i][j + 1] - mp[i + 1][j + 1];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) mp[i][j] = mp[i][j] == 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) h[j] = mp[i][j] ? h[j] + 1 : 0;
        st[top = 0] = 0;
        for (int j = 1; j <= m; j++) {
            while (top && h[st[top]] > h[j]) {
                doit(st[top - 1] + 1, st[top], j - 1, h[st[top]]);
                top--;
            }
            st[++top] = j;
        }
        for (int j = 1; j <= top; j++) doit(st[j - 1] + 1, st[j], m, h[st[j]]);
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m + 1; j++) ans[i][j] += ans[i][j - 1];
    for (int i = n; i; i--)
        for (int j = m; j; j--)
            ans[i][j] += ans[i][j + 1] + ans[i + 1][j] - ans[i + 1][j + 1];
    while (q--) {
        int x, y;
        scanf("%d%d", &x, &y);
        printf("%d\n", ans[x][y]);
    }
    return 0;
}
```

8.12 插头 DP

```
/*
    插头DP
    维护连通块
    状态用最小表示法
    逐格递推清空哈希表
    当相同状态在hash中出现根据题目要求求极值或者累加
*/
/*
    Example
        左上角到右下角经过所有非障碍格的方案数
```

```
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int bit = 7, dig = 3;
const int N = 200010;
const int SIZE = 4001;
struct Hash_Table {
    int size, g[SIZE], nxt[N];
    LL state[N], value[N];
    void init() {
        size = 0;
        memset(g, -1, sizeof(g));
    }
    void push(LL S, LL V) {
        int id = S % SIZE;
        for (int i = g[id]; ~i; i = nxt[i]) {
            if (state[i] == S) {
                value[i] += V;
                return;
            }
        }
        state[size] = S, value[size] = V;
        nxt[size] = g[id], g[id] = size++;
    }
} dp[2];
int n, m;
int code[20], vis[20];
int cur, pre;
int mp[20][20];
int edx, edy;
LL ans;
inline void Decode(LL S, int m) {
    for (int i = 0; i <= m; i++) code[i] = S & bit, S >>= dig;
}
inline LL Encode(int m) {
    LL res = 0;
    int x = 1;
    memset(vis, -1, sizeof(vis));
    vis[0] = 0;
    for (int i = m; i >= 0; i--) {
        if (-1 == vis[code[i]]) vis[code[i]] = x++;
        code[i] = vis[code[i]];
        res <<= dig;
```

```
        res |= code[i];
    }
    return res;
}
inline bool check(int m) {
    for (int i = 0; i <= m; i++)
        if (code[i]) return 0;
    return 1;
}
inline void DP(int x, int y, int k) {
    Decode(dp[pre].state[k], m);
    int lft = code[y - 1], up = code[y];
    LL V = dp[pre].value[k], S = dp[pre].state[k];
    code[y] = code[y - 1] = 0;
    if ((x == n && y == 1) || (x == n && y == m)) {
        if ((lft && !up) || (!lft && up))
            dp[cur].push(Encode(m), V);
        else if (!lft && !up) {
            if (x < n && mp[x + 1][y]) {
                code[y - 1] = bit;
                dp[cur].push(Encode(m), V), code[y - 1] = 0;
            }
            if (y < m && mp[x][y + 1]) {
                code[y] = bit;
                dp[cur].push(Encode(m), V), code[y] = 0;
            }
        }
    } else if (!lft && !up) {
        if (!mp[x][y])
            dp[cur].push(Encode(m), V);
        else if (x < n && mp[x + 1][y] && y < m && mp[x][y + 1]) {
            code[y] = code[y - 1] = bit;
            dp[cur].push(Encode(m), V);
        }
    } else if (!lft || !up) {
        if (x < n && mp[x + 1][y]) {
            code[y - 1] = lft + up;
            dp[cur].push(Encode(m), V), code[y - 1] = 0;
        }
        if (y < m && mp[x][y + 1]) {
            code[y] = lft + up;
            dp[cur].push(Encode(m), V), code[y] = 0;
        }
    } else if (lft != up) {
        for (int i = 0; i <= m; i++)
            if (code[i] == lft) code[i] = up;
```

```
            dp[cur].push(Encode(m), V);
        }
    }
    LL PlugDp() {
        cur = 0;
        dp[0].init();
        dp[0].push(0, 1);
        for (int i = 1; i <= n; i++) {
            pre = cur, cur ^= 1;
            dp[cur].init();
            for (int k = 0; k < dp[pre].size; k++)
                dp[cur].push(dp[pre].state[k] << dig, dp[pre].value[k]);
            for (int j = 1; j <= m; j++) {
                pre = cur, cur ^= 1, dp[cur].init();
                for (int k = 0; k < dp[pre].size; k++) DP(i, j, k);
            }
        }
        ans = 0;
        for (int k = 0; k < dp[cur].size; k++) ans += dp[cur].value[k];
        return ans;
    }
    char s[30];
    int main() {
        while (scanf("%d%d", &n, &m), n + m) {
            memset(mp, 0, sizeof(mp));
            for (int i = 1; i <= n; i++) {
                scanf("%s", s + 1);
                for (int j = 1; j <= m; j++) {
                    if (s[j] == '.') mp[i][j] = 1;
                }
            }
            printf("%lld\n", PlugDp());
        }
        return 0;
    }
    /*
        Example
            给出一个格子图，'O'表示必须通过，'X'表示不能通过，'*'表示可通过
            问画一条经过所有'O'的哈密顿回路的方案数
        Solution
            对于可走可不走的格子我们可以将其分必须通过和不能通过状态讨论加入，
            我们对于连通块标号采用最小表示法，当发现不同标号的两个插头合并时，
            需要将其中一种的左右插头换成另一种的标号，
            相同标号的两个插头合并时必须在结束状态。
            由于可走可不走格子的存在导致终结位置不唯一，
            我们只要在最后一个O后面的格子判断和累加答案即可
```

```
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int bit = 7, dig = 3;
const int N = 200010;
const int SIZE = 4001;
struct Hash_Table {
    int size, g[SIZE], nxt[N];
    LL state[N], value[N];
    void init() {
        size = 0;
        memset(g, -1, sizeof(g));
    }
    void push(LL S, LL V) {
        int id = S % SIZE;
        for (int i = g[id]; ~i; i = nxt[i]) {
            if (state[i] == S) {
                value[i] += V;
                return;
            }
        }
        state[size] = S, value[size] = V;
        nxt[size] = g[id], g[id] = size++;
    }
} dp[2];
int n, m;
int code[20], vis[20];
int cur, pre;
char mp[20][20];
int edx, edy;
LL ans;
inline void Decode(LL S, int m) {
    for (int i = 0; i <= m; i++) code[i] = S & bit, S >>= dig;
}
inline LL Encode(int m) {
    LL res = 0;
    int x = 1;
    memset(vis, -1, sizeof(vis));
    vis[0] = 0;
    for (int i = m; i >= 0; i--) {
        if (-1 == vis[code[i]]) vis[code[i]] = x++;
        code[i] = vis[code[i]];
        res <<= dig;
```

```
        res |= code[i];
    }
    return res;
}
inline bool check(int m) {
    for (int i = 0; i <= m; i++)
        if (code[i]) return 0;
    return 1;
}
inline void DP(int x, int y, int k) {
    Decode(dp[pre].state[k], m);
    int lft = code[y - 1], up = code[y];
    LL V = dp[pre].value[k], S = dp[pre].state[k];
    code[y] = code[y - 1] = 0;
    if (mp[x][y] == 'X' || mp[x][y] == '*') { // 不能走|可走可不走
        if (!lft && !up) dp[cur].push(S, V);
    }
    if (mp[x][y] == 'O' || mp[x][y] == '*') { // 必经|可走可不走
        if (!lft && !up) {
            if (x < n && mp[x + 1][y] != 'X' && y < m && mp[x][y + 1] != 'X') {
                code[y] = code[y - 1] = bit;
                dp[cur].push(Encode(m), V);
            }
        } else if (!lft || !up) {
            if (x < n && mp[x + 1][y] != 'X') {
                code[y - 1] = lft + up;
                dp[cur].push(Encode(m), V), code[y - 1] = 0;
            }
            if (y < m && mp[x][y + 1] != 'X') {
                code[y] = lft + up;
                dp[cur].push(Encode(m), V), code[y] = 0;
            }
        } else if (lft != up) {
            for (int i = 0; i <= m; i++)
                if (code[i] == lft) code[i] = up;
            dp[cur].push(Encode(m), V);
        } else if ((x > edx || (x == edx && y >= edy)) && lft == up && check(m))
            ans += V;
    }
}
/*
    Example
        给出一张格子图，每个格子上有个权值，问从左上角到达右下角，
        可以获得的简单路最大路径权值和是多少
    Solution
        开始状态建立单插头，终结状态只接受单插头，因为每个格子不是必经，
```

```
        因此需要加入其成为障碍格和成为必经格两个状态，对于连通块的合并，
        必须标号不同，这样才能保证答案是简单路。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int bit = 7, dig = 3;
const int N = 200010;
const int SIZE = 4001;
struct Hash_Table {
    int size, g[SIZE], nxt[N];
    LL state[N], value[N];
    void init() {
        size = 0;
        memset(g, -1, sizeof(g));
    }
    void push(LL S, LL V) {
        int id = S % SIZE;
        for (int i = g[id]; ~i; i = nxt[i]) {
            if (state[i] == S) {
                value[i] = max(value[i], V);
                return;
            }
        }
        state[size] = S, value[size] = V;
        nxt[size] = g[id], g[id] = size++;
    }
} dp[2];
int n, m;
int code[20], vis[20];
int cur, pre;
int mp[20][20];
int edx, edy;
LL ans;
inline void Decode(LL S, int m) {
    for (int i = 0; i <= m; i++) code[i] = S & bit, S >>= dig;
}
inline LL Encode(int m) {
    LL res = 0;
    int x = 1;
    memset(vis, -1, sizeof(vis));
    vis[0] = 0;
    for (int i = m; i >= 0; i--) {
        if (-1 == vis[code[i]]) vis[code[i]] = x++;
```

```
        code[i] = vis[code[i]];
        res <<= dig;
        res |= code[i];
    }
    return res;
}
inline bool check(int m) {
    for (int i = 0; i <= m; i++)
        if (code[i]) return 0;
    return 1;
}
inline void DP(int x, int y, int k) {
    Decode(dp[pre].state[k], m);
    int lft = code[y - 1], up = code[y];
    LL V = dp[pre].value[k], S = dp[pre].state[k];
    code[y] = code[y - 1] = 0;
    if ((x == 1 && y == 1) || (x == n && y == m)) {
        if ((lft && !up) || (!lft && up)) ans = max(ans, V + mp[x][y]);
        if (!lft && !up) {
            if (x < n) {
                code[y - 1] = bit;
                dp[cur].push(Encode(m), V + mp[x][y]), code[y - 1] = 0;
            }
            if (y < m) {
                code[y] = bit;
                dp[cur].push(Encode(m), V + mp[x][y]), code[y] = 0;
            }
        }
    } else if (!lft && !up) {
        if (x < n && y < m) {
            code[y] = code[y - 1] = bit;
            dp[cur].push(Encode(m), V + mp[x][y]);
        }
        code[y] = code[y - 1] = 0;
        dp[cur].push(Encode(m), V);
    } else if (!lft || !up) {
        if (x < n) {
            code[y - 1] = lft + up;
            dp[cur].push(Encode(m), V + mp[x][y]), code[y - 1] = 0;
        }
        if (y < m) {
            code[y] = lft + up;
            dp[cur].push(Encode(m), V + mp[x][y]), code[y] = 0;
        }
    } else if (lft != up) {
        for (int i = 0; i <= m; i++)
```

```
            if (code[i] == lft) code[i] = up;
        dp[cur].push(Encode(m), V + mp[x][y]);
    }
}
LL PlugDp() {
    cur = 0;
    dp[0].init();
    dp[0].push(0, 0);
    for (int i = 1; i <= n; i++) {
        pre = cur, cur ^= 1;
        dp[cur].init();
        for (int k = 0; k < dp[pre].size; k++)
            dp[cur].push(dp[pre].state[k] << dig, dp[pre].value[k]);
        for (int j = 1; j <= m; j++) {
            pre = cur, cur ^= 1, dp[cur].init();
            for (int k = 0; k < dp[pre].size; k++) DP(i, j, k);
        }
    }
    return ans;
}
int T, cas = 0;
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) scanf("%d", &mp[i][j]);
        ans = -0x3f3f3f3f3f3f3f3f;
        edx = n, edy = m;
        if (n == 1 & m == 1) {
            printf("Case %d: %d\n", ++cas, mp[1][1]);
            continue;
        }
        printf("Case %d: %lld\n", ++cas, PlugDp());
    }
    return 0;
}
/*
    Example
        给出一个方格图，格子之间的边有一定的价值，
        求一条经过所有格子的回路使得穿过的边权和最小
    Solution
        插头dp，在每个状态加入hashtable的时候更新该状态的最小值，
        在终结状态更新答案即可
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
```

```
using namespace std;
typedef long long LL;
const int bit = 7, dig = 3;
const int N = 200010;
const int SIZE = 4001;
struct Hash_Table {
    int size, g[SIZE], nxt[N];
    LL state[N], value[N];
    void init() {
        size = 0;
        memset(g, -1, sizeof(g));
    }
    void push(LL S, LL V) {
        int id = S % SIZE;
        // 对于哈密顿最短回路的时候注意hash里面的累加改为取最小值
        for (int i = g[id]; ~i; i = nxt[i]) {
            if (state[i] == S) {
                value[i] = min(value[i], V);
                return;
            }
        }
        state[size] = S, value[size] = V;
        nxt[size] = g[id], g[id] = size++;
    }
} dp[2];
int n, m;
int code[20], vis[20];
int cur, pre;
char str[10010];
int rht[20][20], dwn[20][20];
int edx, edy;
LL ans;
inline void Decode(LL S, int m) {
    for (int i = 0; i <= m; i++) code[i] = S & bit, S >>= dig;
}
inline LL Encode(int m) {
    LL res = 0;
    int x = 1;
    memset(vis, -1, sizeof(vis));
    vis[0] = 0;
    for (int i = m; i >= 0; i--) {
        if (-1 == vis[code[i]]) vis[code[i]] = x++;
        code[i] = vis[code[i]];
        res <<= dig;
        res |= code[i];
    }
```

```
    return res;
}
inline bool check(int m) {
    for (int i = 0; i <= m; i++)
        if (code[i]) return 0;
    return 1;
}
inline void DP(int x, int y, int k) {
    Decode(dp[pre].state[k], m);
    int lft = code[y - 1], up = code[y];
    LL V = dp[pre].value[k], S = dp[pre].state[k];
    code[y] = code[y - 1] = 0;
    if (!lft && !up) {
        if (x < n && y < m) {
            code[y] = code[y - 1] = bit;
            dp[cur].push(Encode(m), V + dwn[x][y] + rht[x][y]);
        }
    } else if (!lft || !up) {
        if (x < n) {
            code[y - 1] = lft + up;
            dp[cur].push(Encode(m), V + dwn[x][y]), code[y - 1] = 0;
        }
        if (y < m) {
            code[y] = lft + up;
            dp[cur].push(Encode(m), V + rht[x][y]), code[y] = 0;
        }
    } else if (lft != up) {
        for (int i = 0; i <= m; i++)
            if (code[i] == lft) code[i] = up;
        dp[cur].push(Encode(m), V);
    } else if ((x == edx && y == edy) && lft == up && check(m))
        ans = min(ans, V);
}
LL PlugDp() {
    cur = 0;
    dp[0].init();
    dp[0].push(0, 0);
    for (int i = 1; i <= n; i++) {
        pre = cur, cur ^= 1;
        dp[cur].init();
        for (int k = 0; k < dp[pre].size; k++)
            dp[cur].push(dp[pre].state[k] << dig, dp[pre].value[k]);
        for (int j = 1; j <= m; j++) {
            pre = cur, cur ^= 1, dp[cur].init();
            for (int k = 0; k < dp[pre].size; k++) DP(i, j, k);
        }
```

```
    }
    return ans;
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        getchar();
        gets(str);
        for (int i = 1; i < n; i++) {
            gets(str);
            for (int j = 1; j < m; j++) rht[i][j] = str[2 * j] - '0';
            gets(str);
            for (int j = 1; j <= m; j++) dwn[i][j] = str[2 * j - 1] - '0';
        }
        gets(str);
        for (int i = 1; i < m; i++) rht[n][i] = str[2 * i] - '0';
        gets(str);
        ans = 1e10;
        edx = n, edy = m;
        printf("%lld\n", PlugDp());
    }
    return 0;
}
```

8.13 矩阵优化插头 DP

```
/*
    插头DP+矩阵快速幂
    求一个n*m的格子图中，从左上角到右下角经过所有格子的方案数
    2<=n<=7
    1<=m<=10^9
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int mod = 7777777;
struct mat {
    int n, m;
    int v[200][200];
};
mat mul(mat a, mat b) {
```

```
    mat ret;
    ret.n = a.n;
    ret.m = b.m;
    for (int i = 0; i < a.n; i++) {
        for (int j = 0; j < b.m; j++) {
            LL sum = 0;
            for (int k = 0; k < a.m; k++) sum += 1LL * a.v[i][k] * b.v[k][j];
            ret.v[i][j] = sum % mod;
        }
    }
    return ret;
}
mat pow(mat a, int n) {
    mat ret = a;
    memset(ret.v, 0, sizeof(ret.v));
    for (int i = 0; i < a.n; i++) ret.v[i][i] = 1;
    mat tmp = a;
    while (n) {
        if (n & 1) ret = mul(ret, tmp);
        tmp = mul(tmp, tmp);
        n >>= 1;
    }
    return ret;
}
const int bit = 3, dig = 2;
const int N = 200010;
const int SIZE = 4001;
struct Hash_Table {
    int size, g[SIZE], nxt[N];
    LL state[N];
    void init() {
        size = 0;
        memset(g, -1, sizeof(g));
    }
    int push(LL S) {
        int id = S % SIZE;
        for (int i = g[id]; ~i; i = nxt[i]) {
            if (state[i] == S) {
                return i;
            }
        }
        state[size] = S;
        nxt[size] = g[id], g[id] = size++;
        return size - 1;
    }
} dp;
```

```
int n, m;
int code[20], vis[20];
int cur, pre;
char mp[20][20];
int edx, edy;
inline void Decode(LL S, int m) {
    for (int i = m - 1; i >= 0; i--) code[i] = S & bit, S >>= dig;
}
inline LL Encode(int m) {
    LL res = 0;
    int x = 1;
    memset(vis, -1, sizeof(vis));
    vis[0] = 0;
    for (int i = 0; i < m; i++) {
        if (-1 == vis[code[i]]) vis[code[i]] = x++;
        code[i] = vis[code[i]];
        res <<= dig;
        res |= code[i];
    }
    return res;
}
bool check(int st, int nst) {
    Decode(st, n);
    int flag = 0, cnt = 0, k;
    for (int i = 0; i < n; i++) {
        if (flag == 0) {
            if (code[i] == 0 && (nst & (1 << i)) == 0) return 0;
            if (code[i] && (nst & (1 << i))) continue;
            if (code[i])
                flag = code[i];
            else
                flag = -1;
            k = i;
        } else {
            if (code[i] && (nst & (1 << i))) return 0;
            if (code[i] == 0 && (nst & (1 << i)) == 0) continue;
            if (code[i]) {
                if (code[i] == flag && ((nst != 0) || i != n - 1)) return 0;
                if (flag > 0) {
                    for (int j = 0; j < N; j++)
                        if (code[j] == code[i] && j != i) code[j] = code[k];
                    code[i] = code[k] = 0;
                } else {
                    code[k] = code[i];
                    code[i] = 0;
                }
```

```
            } else {
                if (flag > 0)
                    code[i] = code[k], code[k] = 0;
                else
                    code[i] = code[k] = n + (cnt++);
            }
            flag = 0;
        }
    }
    if (flag != 0) return 0;
    return 1;
}
int D, g[200][200];
struct Node {
    int g[200][200];
    int D;
} node[20];
void init() {
    if (node[n].D != 0) {
        memcpy(g, node[n].g, sizeof(node[n].g));
        D = node[n].D;
        return;
    }
    int st, nst;
    dp.init();
    memset(code, 0, sizeof(code));
    code[0] = code[n - 1] = 1;
    dp.push(0);
    dp.push(Encode(n));
    memset(g, 0, sizeof(g));
    for (int i = 1; i < dp.size; i++) {
        st = dp.state[i];
        for (nst = 0; nst < (1 << n); nst++)
            if (check(st, nst)) {
                int j = dp.push(Encode(n));
                g[i][j] = 1;
            }
    }
    D = dp.size;
    memcpy(node[n].g, g, sizeof(g));
    node[n].D = D;
}
int main() {
    for (int i = 0; i < 20; i++) node[i].D = 0;
    while (~scanf("%d%d", &n, &m)) {
        init();
```

```
        mat tmp;
        tmp.n = tmp.m = D;
        memcpy(tmp.v, g, sizeof(g));
        mat ans = pow(tmp, m);
        if (ans.v[1][0] == 0)
            printf("Impossible\n");
        else
            printf("%d\n", ans.v[1][0] % mod);
    }
    return 0;
}
```

8.14 凸壳维护

```
/*
    凸壳维护
    可解决形如 dp[i]=min( dp[j] + b[j]*a[i] ) for j<i 的问题
    要求 b[j]>=b[j+1] a[i]<=a[i+1]
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
struct Hull {
    long long a[N], b[N];
    double x[N];
    int head, tail;
    Hull() : head(1), tail(0) {}
    long long get(long long xQuery) {
        if (head > tail) return 0;
        while (head < tail && x[head + 1] <= xQuery) head++;
        x[head] = xQuery;
        return a[head] * xQuery + b[head];
    }
    void add(long long aNew, long long bNew) {
        double xNew = -1e18;
        while (head <= tail) {
            if (aNew == a[tail]) return;
            xNew = 1.0 * (b[tail] - bNew) / (aNew - a[tail]);
            if (head == tail || xNew >= x[tail]) break;
            tail--;
        }
        a[++tail] = aNew;
        b[tail] = bNew;
        x[tail] = xNew;
```

```
    }
};
/*
    f[i][j]=min(f[i-1][k]+a[j]*(j-k)-(s[j]-s[k]))
    与k相关:
        斜率: -k
        截距: f[i-1][k]+s[k]
    求解方法:
        for(int i=1;i<=nFeed;i++){
            Hull hull;
            for(int j=1;j<=m;j++){
                f[i][j]=hull.get(a[j])+a[j]*j-s[j];
                if(i>1)hull.add(-j,f[i-1][j]+s[j]);
            }
        }
*/
int n, m, nFeed;
long long d[N], p[N], h[N], a[N], s[N], t[N];
long long f[110][N];
int main() {
    while (~scanf("%d%d%d", &n, &m, &nFeed)) {
        for (int i = 2; i <= n; i++) {
            scanf("%lld", &d[i]);
            p[i] = p[i - 1] + d[i];
        }
        for (int i = 1; i <= m; i++) {
            scanf("%lld%lld", &h[i], &t[i]);
            a[i] = t[i] - p[h[i]];
        }
        sort(a + 1, a + m + 1);
        for (int i = 1; i <= m; i++) s[i] = s[i - 1] + a[i];
        for (int i = 1; i <= nFeed; i++) {
            Hull hull;
            for (int j = 1; j <= m; j++) {
                f[i][j] = hull.get(a[j]) + a[j] * j - s[j];
                if (i > 1) hull.add(-j, f[i - 1][j] + s[j]);
            }
        }
        long long res = f[1][m];
        for (int i = 1; i <= nFeed; i++) res = min(res, f[i][m]);
        printf("%lld\n", res);
    }
    return 0;
}
```

8.15 凸壳 + 分治

```
/*
    凸包+分治
    分上下凸壳维护
*/
/*
    Problem: 给出一个直线集Ax+B，现在给出一些查询，每次查询一条直线Cx+D，
    问与直线集中直线交点最大的x是多少
    Solution: 我们得到交点x=-(B-D)/(A-C)，我们对所有的A和C取负，
    那么答案就是求点(-Cq,Dq)到点集(-Ai,Bi)的最大斜率，
    我们按照横纵坐标排序，维护上左凸壳和下右凸壳，询问点到凸壳的斜率是一个单峰函数，
    可以三分找到最大斜率的位置，也可以二分找到点到mid和mid+1大小状况突变的位置
*/
// 三分
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 100010;
const double eps = 1e-8;
struct Point {
    int x, y, id;
} p[N], st[N];
bool cmp(Point a, Point b) {
    if (a.x == b.x) return a.y < b.y;
    return a.x < b.x;
}
Point operator-(const Point &a, const Point &b) {
    return (Point){a.x - b.x, a.y - b.y};
}
LL operator^(const Point &a, const Point &b) {
    return 1LL * a.x * b.y - 1LL * a.y * b.x;
}
double GetK(Point p) { return 1.0 * p.y / p.x; }
double ans[N];
void Halfhull(int n) {
    int top = 0;
    for (int i = 0; i < n; i++) {
        if (p[i].id) {
            int l = 0, r = top - 1;
            while (l < r) {
                int m1 = (l + l + r) / 3, m2 = (l + r + r + 2) / 3;
                if (((p[i] - st[m1]) ^ (p[i] - st[m2])) < 0)
                    r = m2 - 1;
                else
                    l = m1 + 1;
```

```
            }
            if (l < top) ans[p[i].id] = max(ans[p[i].id], GetK(p[i] - st[l]));
        } else {
            while (top > 1 &&
                   ((st[top - 1] - st[top - 2]) ^ (p[i] - st[top - 2])) <= 0)
                top--;
            st[top++] = p[i];
        }
    }
}
void Convexhull(int n) {
    sort(p, p + n, cmp);
    Halfhull(n);
    reverse(p, p + n);
    Halfhull(n);
}
int n, m;
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d%d", &p[i].x, &p[i].y);
    scanf("%d", &m);
    for (int i = 0; i < m; i++)
        scanf("%d%d", &p[n + i].x, &p[n + i].y), p[n + i].id = i + 1;
    for (int i = 0; i < n + m; i++) p[i].x = -p[i].x;
    Convexhull(n + m);
    for (int i = 1; i <= m; i++) {
        if (ans[i] < eps)
            puts("No cross");
        else
            printf("%.10f\n", ans[i]);
    }
    return 0;
}
// 二分
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 100010;
const double eps = 1e-8;
struct Point {
    int x, y, id;
} p[N], st[N];
bool cmp(Point a, Point b) {
    if (a.x == b.x) return a.y < b.y;
    return a.x < b.x;
}
```

```
Point operator-(const Point &a, const Point &b) {
    return (Point){a.x - b.x, a.y - b.y};
}
LL operator^(const Point &a, const Point &b) {
    return 1LL * a.x * b.y - 1LL * a.y * b.x;
}
double GetK(Point p) { return 1.0 * p.y / p.x; }
double ans[N];
void Halfhull(int n) {
    int top = 0;
    for (int i = 0; i < n; i++) {
        if (p[i].id) {
            int l = 0, r = top - 1;
            while (l < r) {
                int mid = (l + r) >> 1;
                if (((p[i] - st[mid]) ^ (p[i] - st[mid + 1])) <= 0)
                    r = mid;
                else
                    l = mid + 1;
            }
            if (l < top) ans[p[i].id] = max(ans[p[i].id], GetK(p[i] - st[l]));
        } else {
            while (top > 1 &&
                   ((st[top - 1] - st[top - 2]) ^ (p[i] - st[top - 2])) <= 0)
                top--;
            st[top++] = p[i];
        }
    }
}
void Convexhull(int n) {
    sort(p, p + n, cmp);
    Halfhull(n);
    reverse(p, p + n);
    Halfhull(n);
}
int n, m;
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d%d", &p[i].x, &p[i].y);
    scanf("%d", &m);
    for (int i = 0; i < m; i++)
        scanf("%d%d", &p[n + i].x, &p[n + i].y), p[n + i].id = i + 1;
    for (int i = 0; i < n + m; i++) p[i].x = -p[i].x;
    Convexhull(n + m);
    for (int i = 1; i <= m; i++) {
        if (ans[i] < eps)
```

```
            puts("No cross");
        else
            printf("%.10f\n", ans[i]);
    }
    return 0;
}
```

8.16 动态凸包

```
/*
    动态凸包
    insert_line(m,b): 插入y=mx+b
    query(x): 查询max(mx+b)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const LL is_query = -(1LL << 62);
struct Line {
    LL m, b;
    mutable function<const Line *()> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        LL x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> {
    bool check(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (long double)(z->m - y->m) >=
               (y->b - z->b) * (long double)(y->m - x->m);
    }
    void insert_line(LL m, LL b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (check(y)) {
```

```
            erase(y);
            return;
        }
        while (next(y) != end() && check(next(y))) erase(next(y));
        while (y != begin() && check(prev(y))) erase(prev(y));
    }
    LL query(LL x) {
        auto l = *lower_bound((Line){x, is_query});
        return l.m * x + l.b;
    }
};
/*
    题目大意：给出一棵树，树的根为1，树上每个节点有两个值a和b，
    从一个节点i可以跳到子树上任意一个节点j，代价是a[i]*b[j]，
    从一个点到达另一个节点的代价是图中所有代价之和，
    问每个节点跳到任意一个叶节点的代价最小为多少。
    题解：我们递归计算答案，对于每个节点来说，他的答案为min(a[i]*b[j]+ans[j])，
    那么我们对于每个节点维护一个子树的动态凸包，求最大mx+b即可
    为避免过多的插入和比较，我们把信息都保存在节点最大子树的代表节点u[x]中，
    通过只改变标记u[x]以替代信息的大量转移。
*/
const int N = 100010;
int n, a[N], b[N], size[N], u[N], res[N], st[N], en[N];
vector<int> g[N];
LL ans[N];
HullDynamic T[N];
int dfn = 0;
void dfs(int x, int fx) {
    st[x] = ++dfn;
    res[dfn] = x;
    int t = -1;
    size[x] = 1;
    for (int y : g[x]) {
        if (y == fx) continue;
        dfs(y, x);
        size[x] += size[y];
        if (t == -1 || size[y] > size[t]) t = y;
    }
    if (t == -1) {
        T[x].insert_line(-b[x], 0);
        ans[x] = 0;
        u[x] = x;
    } else {
        u[x] = u[t];
        for (int y : g[x])
            if (y != t) {
```

```
            for (int i = st[y]; i <= en[y]; i++)
                T[u[x]].insert_line(-b[res[i]], -ans[res[i]]);
        }
        ans[x] = -T[u[x]].query(a[x]);
        T[u[x]].insert_line(-b[x], -ans[x]);
    }
    en[x] = dfn;
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", a + i);
    for (int i = 1; i <= n; i++) scanf("%d", b + i);
    for (int i = 1; i < n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(1, 0);
    for (int i = 1; i <= n; i++) printf("%lld ", ans[i]);
    return 0;
}
```

8.17 楼层扔鸡蛋

```
/*
    楼层扔鸡蛋问题
    G[i][j]表示用i个鸡蛋尝试j次所能测出的楼层最高高度
*/
typedef long long LL;
LL G[200][100];
void init() {
    G[0][0] = G[1][0] = 0;
    for (int i = 1; i <= 100; i++) {
        for (int j = 1; j <= 63; j++) {
            G[i][j] = G[i - 1][j - 1] + G[i][j - 1] + 1;
        }
    }
}
LL k, n;
void solve() {
    for (int i = 1; i <= 63; i++)
        if (G[k][i] >= n) {
            printf("%d\n", i);
            return;
```

```
        }
    puts("More than 63 trials needed.");
}
int main() {
    init();
    while (scanf("%lld%lld", &k, &n), k) solve();
    return 0;
}
```

8.18 最长反链

```
/*
    在有向无环图中
    最长反链=DAG最小路径覆盖（对偶定理）
    链：点的集合，这个集合中任意两个元素v和u，要么v能走到u，要么u能走到v。
    反链：集合中所有点不能相互抵达
*/
/*
    Example1
    一般有向无环图做法
    floyd求传递闭包，对于每对可达点拆点连边
    最长反链=n-二分图匹配数
*/
bitset<210> mp[210];
int x, y, n, m, Link[210], used[210];
void R(int& a) {
    char ch;
    while (!((ch = getchar()) >= '0') && (ch <= '9'))
        ;
    a = ch - '0';
    while (((ch = getchar()) >= '0') && (ch <= '9')) a *= 10, a += ch - '0';
}
int Dfs(int x) {
    for (int i = 1; i <= n; i++)
        if (mp[x][i] && !used[i]) {
            used[i] = 1;
            if (Link[i] == -1 || Dfs(Link[i])) {
                Link[i] = x;
                return 1;
            }
        }
    return 0;
}
int main() {
    R(n);
```

```
    R(m);
    for (int i = 1; i <= m; i++) {
        R(x);
        R(y);
        mp[x][y] = 1;
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (mp[j][i]) mp[j] |= mp[i];
    int res = 0;
    memset(Link, -1, sizeof(Link));
    for (int i = 1; i <= n; i++) {
        memset(used, 0, sizeof(used));
        res += Dfs(i);
    }
    printf("%d\n", n - res);
    return 0;
}
/*
    Example2
    特殊图做法
    网格图只能往下走或者往右走，
    每次经过的格子如果数字大于0，则数字减1
    问最少要从左上角到右下角几次，才能把所有格子变成0
*/
const int N = 1010;
int T, n, m, a[N][N];
long long dp[N][N];
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) scanf("%d", &a[i][j]);
        for (int i = 1; i <= n; i++) {
            for (int j = m; j; j--)
                dp[i][j] = max(dp[i - 1][j + 1] + a[i][j],
                               max(dp[i - 1][j], dp[i][j + 1]));
        }
        printf("%lld\n", dp[n][1]);
    }
    return 0;
}
```

8.19 最长公共子串

```
/*
    最长公共子串
    dp[i][j]为i位置和j位置匹配往前的最长公共子串
*/
void LCS(char *a, char *b, int n, int m, int dp[][N]) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = 0;
        }
}
/*
    最短唯一子串
    字符串自匹配最长公共子串
    加一即为每个位置往前的最短唯一子串
*/
void GetMin(int n, int dp[][N], int mn[]) {
    for (int i = 1; i <= n; i++) {
        mn[i] = 0;
        for (int j = 1; j <= n; j++) {
            if (i == j) continue;
            mn[i] = max(mn[i], dp[i][j]);
        }
        mn[i]++;
    }
}
/*
    Example
    求出两个串的最短公共子串，使得该串在两个串中唯一
*/
const int N = 5005;
int dpa[N][N], dpb[N][N], dpab[N][N], la, lb, mna[N], mnb[N];
char a[N], b[N];
int main() {
    scanf(" %s %s", a, b);
    la = strlen(a);
    lb = strlen(b);
    LCS(a, a, la, la, dpa);
    LCS(b, b, lb, lb, dpb);
    GetMin(la, dpa, mna);
    GetMin(lb, dpb, mnb);
    LCS(a, b, la, lb, dpab);
```

```
    int ans = la + 1;
    for (int i = 1; i <= la; i++) {
        for (int j = 1; j <= lb; j++) {
            int t = max(mna[i], mnb[j]);
            if (dpab[i][j] >= t) {
                ans = min(ans, t);
            }
        }
    }
    if (ans > la) ans = -1;
    printf("%d\n", ans);
}
```

8.20 GarsiaWachs 算法

```
/*
    GarsiaWachs算法
    石子归并问题O(nlogn)解法
    N堆石子摆成一条线。现要将石子有次序地合并成一堆。
    规定每次只能选相邻的2堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的代价。
    计算将N堆石子合并成一堆的最小代价。
*/
int a[N], n, i, t;
long long ans;
void combine(int k) {
    int tmp = a[k] + a[k - 1], i, j;
    ans += tmp;
    for (i = k; i < t - 1; i++) a[i] = a[i + 1];
    for (t--, j = k - 1; j > 0 && a[j - 1] < tmp; j--) a[j] = a[j - 1];
    a[j] = tmp;
    while (j >= 2 && a[j] >= a[j - 2]) i = t - j, combine(j - 1), j = t - i;
}
int main() {
    for (scanf("%d", &n), i = 0; i < n; i++) scanf("%d", a + i);
    for (t = i = 1; i < n; i++) {
        a[t++] = a[i];
        while (t >= 3 && a[t - 3] <= a[t - 1]) combine(t - 2);
    }
    while (t > 1) combine(t - 1);
    printf("%lld", ans);
    return 0;
}
```

8.21 经典贪心问题

```
/*
    经典贪心问题
*/
/*
    题目大意:
        给出一列数，求最多取m段连续的数字，使得总和最大
    题解:
        首先我们对数据进行合并处理，连续的一段正数或者连续的一段负数处理成一个数字，
        之后我们发现，如果正数的个数小于等于m，那么直接输出正数的总和即可，
        如果大于m，我们有些正数不选，或者选择一些负数把左右两端的正数并起来。
        这个负数的选择过程相当于减去这个数的绝对值，
        正数选择拿出去的过程也相当于减去这个数的绝对值，
        在选择一个负数合并的过程中，两边的正数必须是没有被操作过的，
        同样，选择一个正数删去的过程中，两边的负数肯定也必须是没有操作过的，
        那么问题就转化为，给你一些数，请你选择其中k个不相邻的数，使得其和最小，
*/
#include <algorithm>
#include <cstdio>
#include <queue>
using namespace std;
typedef pair<int, int> P;
const int N = 100010;
const int INF = 0x3f3f3f3f;
int n, m, a[N], b[N], l[N], r[N];
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        while (a[n] <= 0) n--;
        int st = 1;
        while (a[st] <= 0) st++;
        int cnt = 0, ans = 0;
        for (; st <= n; st++) {
            if (!((a[st] > 0) ^ (a[st - 1] > 0)))
                b[cnt] += a[st];
            else
                b[++cnt] = a[st];
        }
        for (int i = 1; i <= cnt; i++)
            if (b[i] > 0) {
                ans += b[i];
                m--;
            } else
                b[i] = -b[i];
        if (m >= 0) {
```

```
            printf("%d\n", ans);
            continue;
        }
        priority_queue<P, vector<P>, greater<P> > Q;
        for (int i = 1; i <= cnt; i++)
            l[i] = i - 1, r[i] = i + 1, Q.push(P(b[i], i));
        r[cnt] = 0;
        for (int i = 1; i <= -m; i++) {
            while (b[Q.top().second] != Q.top().first) Q.pop();
            int x = Q.top().second;
            Q.pop();
            ans -= b[x];
            if (!l[x]) {
                b[r[x]] = INF;
                l[r[x]] = 0;
            } else if (!r[x]) {
                b[l[x]] = INF;
                r[l[x]] = 0;
            } else {
                b[x] = b[l[x]] + b[r[x]] - b[x];
                b[l[x]] = b[r[x]] = INF;
                r[l[x] = l[l[x]]] = l[r[x] = r[r[x]]] = x;
                Q.push(P(b[x], x));
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    题目大意:
        现在有一些线段[l,r]的需求需要满足，i位置最多允许a[i]条线段堆叠，
        问最多能满足多少条线段的需求
    题解:
        我们将所有的线段按照右端点排序，那么从头到尾考虑能不能满足需求一定能得到最优解，
        因为对于相同右端点的来说，先后顺序不影响放入，
        而对于右端点不同的来说，右端点靠前的先处理一定比靠后的先处理更优。
        处理方式相当于线段树的区间查询和区间修改。
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
const int INF = 0x3f3f3f3f;
struct data {
    int l, r;
```

```
} p[N];
bool cmp(data a, data b) { return a.r < b.r; }
int tag[N << 2], T[N << 2], a[N];
void up(int x) { T[x] = min(T[x << 1], T[x << 1 | 1]); }
void pd(int x) {
    if (tag[x]) {
        T[x << 1] += tag[x];
        T[x << 1 | 1] += tag[x];
        tag[x << 1] += tag[x];
        tag[x << 1 | 1] += tag[x];
        tag[x] = 0;
    }
}
void build(int x, int l, int r) {
    int mid = (l + r) >> 1;
    if (l == r) {
        T[x] = a[l];
        tag[x] = 0;
        return;
    }
    build(x << 1, l, mid);
    build(x << 1 | 1, mid + 1, r);
    up(x);
}
void update(int x, int l, int r, int L, int R, int p) {
    int mid = (l + r) >> 1;
    if (L <= l && r <= R) {
        T[x] += p;
        tag[x] += p;
        return;
    }
    pd(x);
    if (L <= mid) update(x << 1, l, mid, L, R, p);
    if (R > mid) update(x << 1 | 1, mid + 1, r, L, R, p);
    up(x);
}
int query(int x, int l, int r, int L, int R) {
    int mid = (l + r) >> 1;
    if (L <= l && r <= R) return T[x];
    pd(x);
    int res = INF;
    if (L <= mid) res = min(res, query(x << 1, l, mid, L, R));
    if (R > mid) res = min(res, query(x << 1 | 1, mid + 1, r, L, R));
    return res;
}
int n, m;
```

```
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        build(1, 1, n);
        for (int i = 1; i <= m; i++) scanf("%d%d", &p[i].l, &p[i].r);
        sort(p + 1, p + m + 1, cmp);
        int ans = 0;
        for (int i = 1; i <= m; i++) {
            int x = query(1, 1, n, p[i].l, p[i].r);
            if (x) {
                ans++;
                update(1, 1, n, p[i].l, p[i].r, -1);
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

8.22 三元环问题

```
/*
    三元环问题
    题目大意:
        N个点m条边，每个点有一个点权a。
        对于任意一个三元环(j，j，k)其贡献为max(a[i],a[j],a[k])，请你求出贡献值之和。
    题解:
        我们将无向边转化成从权值大的点指向权值小的点的有向边，
        按权值从小到大的顺序枚举起始点，枚举相连的点，
        如果其出度小于sqrt(m)，那么枚举与其相连的点，
        判断是否和起始点相连，否则，枚举起始点相连的点，判断是否和枚举点相连，
        由于边有向性，因此不会出现重复枚举的情况。
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <set>
#include <vector>
using namespace std;
const int N = 250005;
vector<int> v[N];
int i, val[N], sa[N], x, y, n, m, size, Rank[N], Mark[N];
long long ans;
set<int> s[N];
bool cmp(int a, int b) { return val[a] < val[b]; }
```

```
int main() {
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++) scanf("%d", val + i), sa[i] = i;
    for (sort(sa + 1, sa + n + 1, cmp), i = 1; i <= n; i++) Rank[sa[i]] = i;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &x, &y);
        if (Rank[x] < Rank[y]) swap(x, y);
        v[x].push_back(y);
    }
    size = sqrt(m);
    int cnt = 0;
    for (i = 1; i <= n; i++) {
        int x = sa[i];
        ++cnt;
        for (int j = 0; j < v[x].size(); j++) Mark[v[x][j]] = cnt;
        for (int j = 0; j < v[x].size(); j++) {
            int y = v[x][j];
            if (v[y].size() < size) {
                for (int k = 0; k < v[y].size(); k++)
                    ans += (Mark[v[y][k]] == cnt) ? val[x] : 0;
            } else {
                for (int k = 0; k < v[x].size(); k++)
                    ans += s[y].count(v[x][k]) ? val[x] : 0;
            }
            s[x].insert(y);
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

8.23 前缀和

```
/*
    题目大意:
        给出一个只包含三种字母的字符串，求出三种字母数量相等的最长子串
    题解:
        我们记录三个字母的前缀和，我们发现只要一个区段左右端点前缀和之差相等，
        就说明这个区段三个字母出现次数相等，其充要条件为SJ-SO和SO-SI相等，
        所以我们在map中保存双关键字第一次出现的位置，
        每次查询当前位置和第一次出现位置的距离
        求最大值即可。
*/
#include <algorithm>
#include <cstdio>
```

```
#include <map>
#include <utility>
using namespace std;
const int N = 200010;
int n;
char s[N];
int main() {
    while (~scanf("%d", &n)) {
        map<pair<int, int>, int> M;
        scanf("%s", s + 1);
        M[make_pair(0, 0)] = 0;
        int SJ = 0, SO = 0, SI = 0, ans = 0;
        for (int i = 1; i <= n; i++) {
            SJ += (s[i] == 'J');
            SO += (s[i] == 'O');
            SI += (s[i] == 'I');
            if (M.find(make_pair(SJ - SO, SO - SI)) == M.end())
                M[make_pair(SJ - SO, SO - SI)] = i;
            else
                ans = max(ans, i - M[make_pair(SJ - SO, SO - SI)]);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

8.24 高维前缀

```
/*
    高维前缀和
    Example:
        给出n个数字c，求非负整数序列a，满足a<2^m
        并且有a[i]&a[i+1]=0，对于每个a[i]，要保证a[i]不是c[i]的倍数，
        求这样的a[i]序列的个数
    Solution:
        我们用dp[i]表示以i为结尾的方案数，
        我们发现要满足a[i]&a[i+1]=0，
        则dp[i]是从上一次结果中所有满足i&j=0的地方转移过来的
        i&j=0即i&(~j)=i，即i为~j的子集，那么我们每次对上一次的结果进行下标取反操作，
        那么求当前dp[i]，就是求出以i为子集的上一次计算出的dp值的高维前缀和。
        对于c[i]这个条件，我们每轮计算后将c[i]倍数为下标的dp值置0即可。

*/
int T, n, m, c[100];
const int mod = 1000000000;
```

```
struct data {
    int val;
    data operator+(const data &rhs) const {
        int t_val = val + rhs.val;
        if (t_val >= mod) t_val -= mod;
        if (t_val < 0) t_val += mod;
        return data{t_val};
    }
} dp[(1 << 15) + 10], res;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
        int all = 1 << m;
        for (int j = 0; j < all; j++) dp[j].val = (j == 0);
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < all; j += 2) swap(dp[j], dp[j ^ (all - 1)]);
            for (int j = 0; j < m; j++)
                for (int k = 0; k < all; k++) {
                    if (~k & (1 << j)) dp[k] = dp[k] + dp[k | (1 << j)];
                }
            for (int j = 0; j < all; j += c[i]) dp[j].val = 0;
        }
        res.val = 0;
        for (int j = 0; j < all; j++) res = res + dp[j];
        printf("%d\n", res.val);
    }
    return 0;
}
/*
    高维前缀最大次大
    Example:
        给定N个数A1, A2, A3, ... AN,
        从中找到两个数Ai和Aj(i j)使得乘积Ai*Aj*(Ai&Aj)最大
    Solution:
        我们可以枚举x&y的结果z，找出两个数x&y==z使得x*y最大，更新答案即可，
        条件可以被削弱为z为x&y的子集，这种条件放缩不会导致最优解的丢失，
        z为x&y的子集等价于z为x的子集并且z为y的子集。
        那么我们只要找出以z为子集的最大值和次大值，然后枚举z即可计算出答案。
        复杂度O(k*2^k).
*/
struct data {
    int val[2];
    data operator+(const data &rhs) const {
        int t_val[2] = {val[0], val[1]};
```

```
        for (int i = 0; i < 2; i++) {
            if (rhs.val[i] > t_val[0]) {
                t_val[1] = t_val[0];
                t_val[0] = rhs.val[i];
            } else if (rhs.val[i] > t_val[1])
                t_val[1] = rhs.val[i];
        }
        return data{t_val[0], t_val[1]};
    }
} dp[(1 << 20) + 10];
int T, n;
int main() {
    scanf("%d", &T);
    int all = 1 << 20;
    while (T--) {
        for (int i = 0; i < all; i++) dp[i] = data{0, 0};
        scanf("%d", &n);
        for (int i = 1, x; i <= n; i++) {
            scanf("%d", &x);
            dp[x] = dp[x] + data{x, 0};
        }
        for (int i = 0; i < 20; i++)
            for (int j = 0; j < all; j++)
                if (~j & (1 << i)) dp[j] = dp[j] + dp[j | (1 << i)];
        long long ans = 0;
        for (int i = 0; i < all; i++)
            ans = max(ans, 1LL * i * dp[i].val[0] * dp[i].val[1]);
        printf("%lld\n", ans);
    }
    return 0;
}
/*
    高维前缀差分
    Example:
        给出一些数字，问其选出一些数字作and为0的方案数有多少
    Solution:
        题目等价于给出一些集合，问其交集为空集的方案数，
        我们先求交集为S的方案数，记为dp[S]，发现处理起来还是比较麻烦，
        我们放缩一下条件，求出交集包含S的方案数，记为dp[S]，
        我们发现dp[S]，是以其为子集的方案的高维前缀和，
        我们逆序求高维前缀和即可，之后考虑容斥，求出交集为0的情况，
        我们发现这个容斥实质上等价于高维的前缀差分，
        那么我们利用之前的代码，修改一下参数就能得到答案。


*/
```

```
const int mod = 1000000007;
typedef long long LL;
int all, n, m, x;
struct data {
    int val;
    data operator+(const data &rhs) const {
        int t_val = val + rhs.val;
        if (t_val >= mod) t_val -= mod;
        if (t_val < 0) t_val += mod;
        return data{t_val};
    }
    data operator*(const int &rhs) const {
        int t_val = val * rhs;
        return data{t_val};
    }
} dp[(1 << 20) + 10];
LL pow(LL a, LL b, LL p) {
    LL t = 1;
    for (a %= p; b; b >>= 1LL, a = a * a % p)
        if (b & 1LL) t = t * a % p;
    return t;
}
void doit(data dp[], int n, int f) {
    for (int i = 0; i < n; i++) {
        for (int j = all - 1; j >= 0; j--) {
            if (~j & (1 << i)) dp[j] = dp[j] + dp[j | (1 << i)] * f;
        }
    }
}
int main() {
    scanf("%d", &n);
    all = (1 << 20);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        dp[x].val++;
    }
    doit(dp, 20, 1);
    for (int i = 0; i < all; i++) dp[i].val = pow(2, dp[i].val, mod);
    doit(dp, 20, -1);
    printf("%d\n", dp[0].val);
    return 0;
}
```

8.25 高低位统计

```
/*
    高低位统计
*/
/*
    题目大意:
        给出一个随机数生成器，有m个询问，问第bi小的元素是啥
        询问中对于bi<bk,bj<bk,有bi+bj<=bk
    题解: (By Claris)
        对于所有的数字，我们将其按照高位分类，统计每个高16位有几个数字，
        然后定位每个询问的高16位是什么，因为只有100个高16位是被询问到的，
        把这100个高16位的数放入桶里，每次暴力查找，因为数据随机，
        因此每个高16位期望n/65536=153个数，一共只有15300个数字有用
*/
#include <algorithm>
#include <cstdio>
using namespace std;
int T, n, m, k[200], pos[200];
unsigned s[10000010], b[10000010], x, y, z;
unsigned q[65536], cnt[65536], u[65536], tot[65536];
unsigned xorshf96() {
    unsigned t;
    x ^= x << 16;
    x ^= x >> 5;
    x ^= x << 1;
    t = x;
    x = y;
    y = z;
    z = t ^ x ^ y;
    return z;
}
int getpos(int x) {
    for (int i = 0;; i++)
        if (tot[i] >= x) return i;
}
unsigned query(int k, int p) {
    int i = p ? tot[p - 1] : 0, m = 0;
    for (k -= ++i; i <= tot[p]; i++) q[m++] = b[i];
    nth_element(q, q + k, q + m);
    return q[k];
}
int Cas = 1;
int main() {
    while (~scanf("%d%d%u%u%u", &n, &m, &x, &y, &z)) {
        for (int i = 0; i < 65536; i++) cnt[i] = u[i] = 0;
        for (int i = 1; i <= n; i++) {
```

```
            s[i] = xorshf96();
            cnt[s[i] >> 16]++;
        }
        for (int i = 1; i < 65536; i++) cnt[i] += cnt[i - 1];
        for (int i = 0; i < 65535; i++) tot[i] = cnt[i];
        for (int i = 1; i <= m; i++) {
            scanf("%d", &k[i]);
            pos[i] = getpos(++k[i]);
            u[pos[i]] = 1;
        }
        for (int i = 1; i <= n; i++)
            if (u[s[i] >> 16]) b[cnt[s[i] >> 16]--] = s[i];
        printf("Case #%d:", Cas++);
        for (int i = 1; i <= m; i++) printf(" %u", query(k[i], pos[i]));
        puts("");
    }
    return 0;
}
/*
    题目大意:
        给出一棵树，树上每个节点都有一个权值w，w不超过216，树的根为1，
        从一个点往根的方向走，可以得到他的祖先序列，
        现在需要从v1点的祖先序列中挑选出一定数量的点，
        组成数列v1，v2，v3……vm，要求vi是vi-1的祖先，
        求dp[v1]=max(dp[vi]+(w[v1] opt w[vi]))，
        opt是一种运算，在题目中可为xor，or或者and，
        最后求出ans=sum_{i=1}^{n}(i*(w[i]+dp[i]))
    题解:
        对于这道题，我们首先考虑它的简化版本，
        在一个一维数组上求dp[i]=max(dp[j]+(w[i] opt w[j])) (j<i)，
        显然枚举前缀的O(n2)的用脚趾头都能想出来的算法，出题人是不会给过的。
        那么我们观察一下题目，发现一个很奇巧的东西，w的值不超过216，
        难道说每次计算以w结尾的dp最大值，然后枚举二进制？
        一次6w多的计算量，明显也没有产生太大的优化，
        顺着这个思路下去，这道题采用了一种拆值DP的神奇的方式，
        dp[i]=max(dp[j]+([w[i]前八位]opt[w[j]前八位])<<8+[w[i]后八位]opt[w[j]后八位])
        记dp[A][B]=以前八位为A结尾，后八位以B结尾的dp值，于是就可以发现:
            dp[A][B]=max(dp[i][B]+([w[i]前八位]opt[w[A]前八位])<<8)
        那么，在知道了后八位的情况下，前八位就能轻松dp，
        既然这样，那我们就在计算完每个节点之后，预处理后八位的dp值:
            dp[A][i]=max(dp[A][j]+([w[i]后八位]opt[w[j]后八位]))
        这样子每次转移所需要的复杂度就只有28，可以接受。顺利完成。
        而这道题所处理的却是树上的问题，
        那么在每条链上DP的过程中预处理祖先节点dp数组，
        按照上述方法计算子节点的dp值即可，而对于不同的子节点，dp数组备份，然后回溯即可。
*/
```

```
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;
typedef unsigned int UI;
const int N = 65540, mod = 1e9 + 7;
UI T, n, i, w[N], nxt[N], x, f[256][256], tmp[N][256], v[256], ans;
vector<UI> g[N];
char op[5];
UI opt(UI a, UI b) {
    if (op[0] == 'A') return a & b;
    if (op[0] == 'O') return a | b;
    if (op[0] == 'X') return a ^ b;
}
void dfs(UI x) {
    UI dp = 0, A = w[x] >> 8, B = w[x] & 255;
    for (int i = 0; i < 256; i++)
        if (v[i]) dp = max(dp, f[i][B] + (opt(A, i) << 8));
    ans = (1LL * x * (dp + w[x]) + ans) % mod;
    for (v[A]++, i = 0; i < 256; i++)
        tmp[x][i] = f[A][i], f[A][i] = max(f[A][i], opt(B, i) + dp);
    for (int i = 0; i < g[x].size(); i++) dfs(g[x][i]);
    for (v[A]--, i = 0; i < 256; i++) f[A][i] = tmp[x][i];
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d %s", &n, op);
        for (int i = 1; i <= n; i++) scanf("%d", &w[i]), g[i].clear();
        for (int i = 2; i <= n; i++) scanf("%d", &x), g[x].push_back(i);
        ans = 0;
        dfs(1);
        printf("%d\n", ans);
    }
    return 0;
}
```

8.26 尺取法

```
/*
    尺取法
    Example:
        给出一些签到区间和一些补签卡，问可以创造的最长连续签到区间
    Solution:
        如果我们知道选定的最左和最右的签到区间，
```

```
        我们就可以计算出需要补多少的补签卡，如果数量小于等于给定数量，
        那么这个左右可以用来更新答案。所以尺取法就显而易见了。
*/
const int N = 100010;
struct data {
    int l, r;
} p[N];
int l[N], r[N], cnt;
bool cmp(data a, data b) {
    if (a.l == b.l) return a.r < b.r;
    return a.l < b.l;
}
int n, m;
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++) scanf("%d%d", &p[i].l, &p[i].r);
        sort(p + 1, p + n + 1, cmp);
        cnt = 1;
        l[cnt] = p[1].l, r[cnt] = p[1].r;
        for (int i = 2; i <= n; i++) {
            if (p[i].l - 1 > r[cnt]) {
                l[++cnt] = p[i].l;
                r[cnt] = p[i].r;
            }
            r[cnt] = max(r[cnt], p[i].r);
        }
        int ans = r[1] - l[1] + 1 + m, gap = 0, lft = 1;
        for (int rt = 2; rt <= cnt; rt++) {
            gap += l[rt] - 1 - r[rt - 1];
            while (gap > m) {
                lft++;
                gap -= l[lft] - 1 - r[lft - 1];
            }
            ans = max(ans, r[rt] - l[lft] + 1 + m - gap);
        }
        printf("%d\n", ans);
    }
    return 0;
}
/*
    Example:
        给出一个序列，求一个子段和，使得其绝对值最接近给出值，
        输出这个区间的左右端点和区间和。
    Solution:
        因为原序列的前缀和不具有单调性，难以处理，
        因此我们对前缀和进行排序，同时保留前缀和的右端点做标识作用，
```

```
        题目要求区段和的绝对值最接近目标，因此排序不会造成前后顺序变化造成的影响
        现在题目转化为在一个有序数列中，求一个数对，使得差值最接近给出数，
        利用单调性，可以尺取解决问题。


*/
const int N = 100010, INF = INT_MAX;
typedef pair<int, int> P;
int n, m, s, t, ans, ansl, ansr;
P p[N];
int main() {
    while (scanf("%d%d", &n, &m), n && m) {
        p[0] = P(s = 0, 0);
        for (int i = 1; i <= n; i++) scanf("%d", &t), p[i] = P((s += t), i);
        sort(p, p + n + 1);
        while (m--) {
            scanf("%d", &t);
            int l = 0, r = 1, Min = INF;
            while (l <= n && r <= n) {
                int tmp = p[r].first - p[l].first;
                if (abs(tmp - t) < Min) {
                    Min = abs(tmp - t);
                    ans = tmp;
                    ansl = p[l].second;
                    ansr = p[r].second;
                }
                if (tmp > t)
                    l++;
                else if (tmp < t)
                    r++;
                else
                    break;
                if (l == r) r++;
            }
            if (ansl > ansr) swap(ansl, ansr);
            printf("%d %d %d\n", ans, ansl + 1, ansr);
        }
    }
    return 0;
}
```

8.27 蒙特卡罗

```
/*
    蒙特卡罗
```

```
*/
/*
    Example:
        求图中的最大团
    Solution:
        我们随机一个排列，按照这个顺序求答案，对于一个放入答案集中的数，
        我们将其余不与其连接的数删除，顺序统计即可。
*/
int ans = 0, mp[55][55], del[55], n, b[55], x, y;
void r() {
    for (int i = 2; i <= n; i++) swap(b[i], b[rand() % i + 1]);
}
void cal() {
    memset(del, 0, sizeof(del));
    int t = 0;
    for (int i = 1; i <= n; i++)
        if (!del[i]) {
            t++;
            for (int j = i + 1; j <= n; j++)
                if (!mp[b[i]][b[j]]) del[j] = 1;
        }
    ans = max(t, ans);
}
int main() {
    scanf("%d", &n);
    while (~scanf("%d%d", &x, &y)) mp[x][y] = mp[y][x] = 1;
    for (int i = 1; i <= n; i++) b[i] = i;
    for (int i = 1; i <= 10000; i++) {
        r();
        cal();
    }
    printf("%d\n", ans);
    return 0;
}
/*
    Example:
        给一个长度为n的序列a。1 a[i] n。
        m组询问，每次询问一个区间[l,r]，是否存在一个数在[l,r]中出现的次数大于(r-l+1)/2。
        如果存在，输出这个数，否则输出0。
*/
const int N = 500010;
vector<int> pos[N];
int a[N], l, r, n, m;
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; i++) pos[i].clear();
```

```
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            pos[a[i]].push_back(i);
        }
        while (m--) {
            scanf("%d%d", &l, &r);
            int len = r - l + 1, find = 0;
            for (int k = 1; k <= 20; k++) {
                int u = a[l + rand() % len];
                int st = lower_bound(pos[u].begin(), pos[u].end(), l) -
                        pos[u].begin();
                int en = upper_bound(pos[u].begin(), pos[u].end(), r) -
                        pos[u].begin();
                if ((en - st) * 2 > len) {
                    find = u;
                    break;
                }
            }
            if (find)
                printf("%d\n", find);
            else
                puts("0");
        }
    }
    return 0;
}
```

8.28 启发式分解

```
/*
    启发式分解
    题目大意:
        一个序列被称为是不无聊的，仅当它的每个连续子序列存在一个独一无二的数字，
        即每个子序列里至少存在一个数字只出现一次。
        给定一个整数序列，请你判断它是不是不无聊的。
    题解:
        预处理每个元素上一次出现位置和下一个出现位置，
        我们发现对于一个子序列[L,R]来说，
        如果存在pre[i]<L&&nxt[i]>R那么这个子序列一定是满足条件的，
        否则就不满足，那么我们分治处理这个问题，
        从两边往中间寻找这个i，那么每次拆开的复杂度就是拆成的两个序列中较小的一个，
        所以这是一个逆启发式合并的过程，复杂度O(nlogn)
*/
#include <algorithm>
#include <cstdio>
```

```
#include <map>
using namespace std;
const int N = 200010;
int n, T, a[N], pre[N], nxt[N];
bool check(int L, int R) {
    if (L >= R) return 1;
    int l = L, r = R;
    for (int i = L; i <= R; i++) {
        if (i & 1) {
            if (pre[l] < L && nxt[l] > R)
                return (check(L, l - 1) && check(l + 1, R));
            l++;
        } else {
            if (pre[r] < L && nxt[r] > R)
                return (check(L, r - 1) && check(r + 1, R));
            r--;
        }
    }
    return 0;
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        map<int, int> M;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
            nxt[M[a[i]]] = i;
            pre[i] = M[a[i]];
            M[a[i]] = i;
        }
        for (int i = 1; i <= n; i++) nxt[M[a[i]]] = n + 1;
        if (check(1, n))
            puts("non-boring");
        else
            puts("boring");
    }
    return 0;
}
```

8.29 K 叉哈夫曼树

```
/*
    k叉哈夫曼树编码问题
    首先将不足的用0补齐，保证每次都能选出k个来合并
```

```
    然后以权值为第一关键字，最大深度为第二关键字维护小根堆，
    每次取出最小的k个元素进行合并
*/
#include <algorithm>
#include <cstdio>
using namespace std;
typedef long long ll;
typedef pair<ll, int> P;
int n, k, i, b, l;
ll x, a, ans;
P y, h[100010];
void read(ll& a) {
    char c;
    while (!((c = getchar()) >= '0') && (c <= '9')))
        ;
    a = c - '0';
    while (((c = getchar()) >= '0') && (c <= '9')) (a *= 10) += c - '0';
}
void put(P x) {
    h[++l] = x;
    for (int i = l; i > 1 && h[i] < h[i >> 1]; i >>= 1) swap(h[i], h[i >> 1]);
}
void get() {
    y = h[1], h[1] = h[l--];
    for (int i = 1;;) {
        P tmp = h[i];
        int j = 0;
        if ((i << 1) <= l && h[i << 1] < tmp) tmp = h[j = i << 1];
        if ((i << 1 | 1) <= l && h[i << 1 | 1] < tmp) j = i << 1 | 1;
        if (j)
            swap(h[i], h[j]), i = j;
        else
            return;
    }
}
int main() {
    for (scanf("%d%d", &n, &k); n--; put(P(x, 0))) read(x);
    if (k > 2)
        while (l % (k - 1) != 1) put(P(0, 0));
    for (; l > 1; ans += a, put(P(a, b + 1)))
        for (i = a = b = 0; i < k; i++)
            get(), a += y.first, b = max(b, y.second);
    return printf("%lld\n%d", ans, h[1].second), 0;
}
```

8.30 骑士跳跃算法

```
/*
    骑士跳跃算法
    只能走日字，求从(0,0)跳到(x,y)所需要的最小步数
*/
int KSP(int x, int y) {
    if (x < 0) x = -x;
    if (y < 0) y = -y;
    if (x < y) swap(x, y);
    if (x == 1 && y == 0) return 3;
    if (x == 2 && y == 2) return 4;
    int d = x - y;
    if (y > d) return 2 * ((y - d + 2) / 3) + d;
    return d - 2 * ((d - y) / 4);
}
```

9 计算几何

9.1 极角排序

```
/*
    题目大意：一个二维平面上有一些黑点和白点，现在用一块木板去划分他们，
    木板A面的每个黑点计一分，B面每个白点计一分，木板上每个点计一分，
    求最高得分
    题解：我们枚举木板的旋转点，将所有黑点关于旋转点做轴对称，
    那么问题就转化为统计旋转线一面的点数量，对于枚举的旋转点对剩余点做极角排序，
    之后做旋转线扫描，统计扫描线上方点和下方点，更新答案即可。
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
const double eps = 1e-8;
const int N = 1010;
int n;
int sgn(double x) {
    if (x >= eps) return 1;
    if (x <= -eps) return -1;
    return 0;
}
struct Point {
    int x, y, k;
    double ang;
    Point(){};
    Point(int _x, int _y) : x(_x), y(_y) {}
} p0[N], p[N];
Point operator-(Point a, Point b) { return Point(a.x - b.x, a.y - b.y); }
int Cross(Point a, Point b) { return a.x * b.y - a.y * b.x; }
bool cmp(Point a, Point b) { return a.ang < b.ang; }
int main() {
    while (scanf("%d", &n), n) {
        int ans = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%d%d%d", &p[i].x, &p[i].y, &p[i].k);
            p0[i] = p[i];
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                p[j] = p0[j];
                if (p[j].k) { // 权值为1的点关于旋转点对称
                    p[j].x = p0[i].x * 2 - p[j].x;
                    p[j].y = p0[i].y * 2 - p[j].y;
```

```
            }
            p[j].ang = atan2(p[j].y - p0[i].y, p[j].x - p0[i].x);
        }
        swap(p[i], p[1]);
        sort(p + 2, p + n + 1, cmp);
        for (int s = 2, t = 3; s <= n && sgn(p[s].ang) <= 0;
             s++) { // sgn<=0来防止重复枚举线
            int on = 2;
            for (; t <= n && Cross(p[s] - p[1], p[t] - p[1]) >= 0; t++) {
                if (Cross(p[s] - p[1], p[t] - p[1]) == 0) on++;
            }
            ans = max(ans, max(t - s + 1, n - (t - s + 1) + on));
        }
    }
    printf("%d\n", ans);
  }
  return 0;
}
/*
  题目大意：给出n个点，求点集的每个子集能构成的凸包的面积和的两倍
  题解：凸包面积两倍可以由凸包上每条有向边的两点与原点构成向量计算叉积的和来表示，
  我们枚举旋转点，对点集进行旋转线扫描，对于每条边，我们计算其逆时针180度以内点的个数x，
  那么这条边的贡献就是其端点与原点叉积乘上(pw[x]-1)，即将其作为凸包边的子集个数，
  我们只要枚举每个旋转点，对有向边进行线扫描，统计贡献和即可。
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
typedef long long LL;
const int N = 1010;
const LL mod = 998244353;
const double PI = acos(-1.0);
LL pw2[N];
int T, n;
struct Point {
  LL x, y;
  double ang;
  Point(){};
  Point(LL _x, LL _y) : x(_x), y(_y) {}
} p0[N], p[N << 1];
Point operator-(Point a, Point b) { return Point(a.x - b.x, a.y - b.y); }
LL Cross(Point a, Point b) { return a.x * b.y - a.y * b.x; }
bool cmp(Point a, Point b) { return a.ang < b.ang; }
int main() {
  for (int i = pw2[0] = 1; i < N; i++) pw2[i] = (pw2[i - 1] << 1) % mod;
```

```
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        LL ans = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%lld%lld", &p[i].x, &p[i].y);
            p0[i] = p[i];
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                p[j] = p0[j];
                p[j].ang = atan2(p[j].y - p0[i].y, p[j].x - p0[i].x);
            }
            swap(p[i], p[1]);
            sort(p + 2, p + n + 1, cmp);
            for (int j = 1; j < n; j++)
                p[n + j] = p[j + 1], p[n + j].ang += 2.0 * PI;
            for (int s = 2, t = 2; s <= n; s++) {
                while (p[t].ang < p[s].ang + PI) t++;
                LL ctb = (pw2[t - s - 1] - 1) *
                        ((Cross(p[1], p[s]) % mod + mod) % mod) % mod;
                ans = (ans + ctb) % mod;
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

9.2 最近点对

```
/*
    平面最近点对
    分治+最近查找三个优化
*/
const int N = 100010;
struct P {
    double x, y;
} s[N];
int n, p[N];
inline bool cmpx(P const& a, P const& b) {
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
inline bool cmpy(int const& a, int const& b) { return s[a].y < s[b].y; }
inline double dis(int const& a, int const& b) {
```

```
    return sqrt((s[a].x - s[b].x) * (s[a].x - s[b].x) +
                (s[a].y - s[b].y) * (s[a].y - s[b].y));
}
double fuc(int l, int r) {
    double d = 1e20;
    if (l == r) return d;
    if (r - l == 1) {
        return dis(l, r);
    }
    if (r - l == 2) {
        double s1 = min(dis(l, r), dis(l, l + 1));
        s1 = min(s1, dis(l + 1, r));
        return s1;
    }
    int mid = (l + r) >> 1;
    d = min(fuc(l, mid), fuc(mid + 1, r));
    int cnt = 0;
    for (int i = l; i <= r; i++) {
        if (fabs(s[mid].x - s[i].x) <= d) p[cnt++] = i;
    }
    sort(p, p + cnt, cmpy);
    for (int i = 0; i < cnt; i++)
        for (int j = i + 1; j < cnt && j < i + 3; j++) {
            if (s[p[j]].y - s[p[i]].y >= d) break;
            double d1 = dis(p[i], p[j]);
            if (d > d1) d = d1;
        }
    return d;
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; i++) scanf("%lf%lf", &s[i].x, &s[i].y);
        sort(s, s + n, cmpx);
        printf("%.2lf\n", fuc(0, n - 1));
    }
    return 0;
}
```

9.3 三角形

```
/*
    求两个三角形的关系判断
    contain: 内含
    intersect: 相交
    disjoint: 相离
```

```
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
using namespace std;
const double EPS = 1e-12;
int sgn(double x) {
    if (fabs(x) < EPS) return 0;
    if (x < 0)
        return -1;
    else
        return 1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double x1, double y1) {
        x = x1;
        y = y1;
    }
    Point operator+(const Point &b) const { return Point(x + b.x, y + b.x); }
    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); }
    double operator*(const Point &b) const { return x * b.x + y * b.y; } //点积
    double operator^(const Point &b) const { return x * b.y - y * b.x; } //叉积
    Point operator/(const double b) const { return Point(x / b, y / b); }
    //绕原点旋转角度B（弧度值），后x,y的变化
    void transXY(double B) {
        double tx = x, ty = y;
        x = tx * cos(B) - ty * sin(B);
        y = tx * sin(B) + ty * cos(B);
    }
};
double dist(Point a, Point b) { return sqrt((a - b) * (a - b)); }
//直线类，包含位置判断
struct Line {
    Point u, v;
    Line() {}
    Line(Point u1, Point v1) {
        u = u1;
        v = v1;
    }
    pair<Point, int> operator&(const Line &b) const {
        Point res = u;
        if (sgn((u - v) ^ (b.u - b.v)) == 0) {
            if (sgn((b.u - u) ^ (b.v - u)) == 0)
```

```
            return make_pair(res, 0); //重合
        else
            return make_pair(res, 1); //平行
    }
    double k = ((u - b.u) ^ (b.u - b.v)) / ((u - v) ^ (b.u - b.v));
    res.x += (v.x - u.x) * k;
    res.y += (v.y - u.y) * k;
    return make_pair(res, 2); //相交
  }
};
//判断线段是否相交
bool inter(Line l1, Line l2) {
  return max(l1.u.x, l1.v.x) >= min(l2.u.x, l2.v.x) &&
      max(l2.u.x, l2.v.x) >= min(l1.u.x, l1.v.x) &&
      max(l1.u.y, l1.v.y) >= min(l2.u.y, l2.v.y) &&
      max(l2.u.y, l2.v.y) >= min(l1.u.y, l1.v.y) &&
      sgn((l2.u - l1.u) ^ (l1.v - l1.u)) *
          sgn((l2.v - l1.u) ^ (l1.v - l1.u)) <=
        0 &&
      sgn((l1.u - l2.u) ^ (l2.v - l2.u)) *
          sgn((l1.v - l2.u) ^ (l2.v - l2.u)) <=
        0;
}
//判断点在线段上
bool OnSeg(Point P, Line L) {
  return sgn((L.u - P) ^ (L.v - P)) == 0 &&
      sgn((P.x - L.u.x) * (P.x - L.v.x)) <= 0 &&
      sgn((P.y - L.u.y) * (P.y - L.v.y)) <= 0;
}
/*
  inConvexPoly:判断点在凸多边形内
    点形成一个凸包，而且按逆时针排序（如果是顺时针把里面的<0改为>0）
    点的编号:0~n-1
    返回值:
      -1:点在凸多边形外
      0:点在凸多边形边界上
      1:点在凸多边形内
*/
int inConvexPoly(Point a, Point p[], int n) {
  for (int i = 0; i < n; i++) {
    if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
      return -1;
    else if (OnSeg(a, Line(p[i], p[(i + 1) % n])))
      return 0;
  }
  return 1;
```

```
}
Point A[10], B[10];
bool contain(Point A[], Point B[]) {
    int flag = 1;
    for (int i = 0; i < 3; i++)
        if (inConvexPoly(A[i], B, 3) != 1) flag = 0;
    return flag;
}
bool Intersect() {
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (inter(Line(A[i], A[(i + 1) % 3]), Line(B[j], B[(j + 1) % 3])))
                return 1;
    return 0;
}
Point o;
bool _cmp(Point p1, Point p2) {
    double tmp = (p1 - o) ^ (p2 - o);
    if (sgn(tmp) > 0)
        return true;
    else if (sgn(tmp) == 0 && sgn(dist(p1, o) - dist(p2, o)) <= 0)
        return true;
    else
        return false;
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        for (int i = 0; i < 3; i++) scanf("%lf%lf", &A[i].x, &A[i].y);
        o = (A[0] + A[1] + A[2]) / 3.0;
        sort(A, A + 3, _cmp);
        for (int i = 0; i < 3; i++) scanf("%lf%lf", &B[i].x, &B[i].y);
        o = (B[0] + B[1] + B[2]) / 3.0;
        sort(B, B + 3, _cmp);
        if (contain(A, B) || contain(B, A))
            puts("contain");
        else if (Intersect())
            puts("intersect");
        else
            puts("disjoint");
    }
    return 0;
}
```

9.4 矩形面积交

```
/*
    矩形面积交
    线段树+扫描线
    求被覆盖超过一次的面积和
*/
int n;
struct Seg {
    double l, r, h;
    int d;
    Seg() {}
    Seg(double l, double r, double h, double d) : l(l), r(r), h(h), d(d) {}
    bool operator<(const Seg& rhs) const { return h < rhs.h; }
} a[N];
int cnt[N << 2];
double one[N << 2], two[N << 2], all[N];
#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
void push_up(int l, int r, int rt) {
    if (cnt[rt] >= 2)
        two[rt] = one[rt] = all[r + 1] - all[l];
    else if (cnt[rt] == 1) {
        one[rt] = all[r + 1] - all[l];
        if (l == r)
            two[rt] = 0;
        else
            two[rt] = one[rt << 1] + one[rt << 1 | 1];
    } else {
        if (l == r)
            one[rt] = two[rt] = 0;
        else {
            one[rt] = one[rt << 1] + one[rt << 1 | 1];
            two[rt] = two[rt << 1] + two[rt << 1 | 1];
        }
    }
}

void update(int L, int R, int v, int l, int r, int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += v;
        push_up(l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if (L <= m) update(L, R, v, lson);
```

```
    if (R > m) update(L, R, v, rson);
    push_up(l, r, rt);
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            a[i] = Seg(x1, x2, y1, 1);
            a[i + n] = Seg(x1, x2, y2, -1);
            all[i] = x1;
            all[i + n] = x2;
        }
        n <<= 1;
        sort(a + 1, a + 1 + n);
        sort(all + 1, all + 1 + n);
        int m = unique(all + 1, all + 1 + n) - all - 1;
        memset(cnt, 0, sizeof(cnt));
        memset(one, 0, sizeof(one));
        memset(two, 0, sizeof(two));
        double ans = 0;
        for (int i = 1; i < n; i++) {
            int l = lower_bound(all + 1, all + 1 + m, a[i].l) - all;
            int r = lower_bound(all + 1, all + 1 + m, a[i].r) - all;
            if (l < r) update(l, r - 1, a[i].d, 1, m, 1);
            ans += two[1] * (a[i + 1].h - a[i].h);
        }
        printf("%.2f\n", ans);
    }
    return 0;
}
```

9.5 矩形面积并

```
/*
    矩形面积并
    线段树+扫描线
*/
const int N = 2010;
int n;
struct Seg {
    double l, r, h;
```

```
    int d;
    Seg() {}
    Seg(double l, double r, double h, int d) : l(l), r(r), h(h), d(d) {}
    bool operator<(const Seg& rhs) const { return h < rhs.h; }
} a[N];
int cnt[N << 2];
double sum[N << 2], all[N];
#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
void push_up(int l, int r, int rt) {
    if (cnt[rt])
        sum[rt] = all[r + 1] - all[l];
    else if (l == r)
        sum[rt] = 0;
    else
        sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void update(int L, int R, int v, int l, int r, int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += v;
        push_up(l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if (L <= m) update(L, R, v, lson);
    if (R > m) update(L, R, v, rson);
    push_up(l, r, rt);
}
int main() {
    while (scanf("%d", &n) && n) {
        for (int i = 1; i <= n; i++) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            a[i] = Seg(x1, x2, y1, 1);
            a[i + n] = Seg(x1, x2, y2, -1);
            all[i] = x1;
            all[i + n] = x2;
        }
        n <<= 1;
        sort(a + 1, a + 1 + n);
        sort(all + 1, all + 1 + n);
        int m = unique(all + 1, all + 1 + n) - all - 1;
        memset(cnt, 0, sizeof(cnt));
        memset(sum, 0, sizeof(sum));
        double ans = 0;
        for (int i = 1; i < n; i++) {
```

```
            int l = lower_bound(all + 1, all + 1 + m, a[i].l) - all;
            int r = lower_bound(all + 1, all + 1 + m, a[i].r) - all;
            if (l < r) update(l, r - 1, a[i].d, 1, m, 1);
            ans += sum[1] * (a[i + 1].h - a[i].h);
        }
        printf("%.0f\n", ans);
    }
    puts("*");
    return 0;
}
```

9.6 矩形周长并

```
/*
    矩形周长并
*/
int n, m[2];
int sum[N << 2], cnt[N << 2], all[2][N];
struct Seg {
    int l, r, h, d;
    Seg() {}
    Seg(int l, int r, int h, int d) : l(l), r(r), h(h), d(d) {}
    bool operator<(const Seg& rhs) const { return h < rhs.h; }
} a[2][N];
#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
void push_up(int p, int l, int r, int rt) {
    if (cnt[rt])
        sum[rt] = all[p][r + 1] - all[p][l];
    else if (l == r)
        sum[rt] = 0;
    else
        sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void update(int p, int L, int R, int v, int l, int r, int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += v;
        push_up(p, l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if (L <= m) update(p, L, R, v, lson);
    if (R > m) update(p, L, R, v, rson);
    push_up(p, l, r, rt);
}
```

```
int main() {
    while (scanf("%d", &n) == 1) {
        for (int i = 1; i <= n; i++) {
            int x1, y1, x2, y2;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            all[0][i] = x1, all[0][i + n] = x2;
            all[1][i] = y1, all[1][i + n] = y2;
            a[0][i] = Seg(x1, x2, y1, 1);
            a[0][i + n] = Seg(x1, x2, y2, -1);
            a[1][i] = Seg(y1, y2, x1, 1);
            a[1][i + n] = Seg(y1, y2, x2, -1);
        }
        n <<= 1;
        sort(all[0] + 1, all[0] + 1 + n);
        m[0] = unique(all[0] + 1, all[0] + 1 + n) - all[0] - 1;
        sort(all[1] + 1, all[1] + 1 + n);
        m[1] = unique(all[1] + 1, all[1] + 1 + n) - all[1] - 1;
        sort(a[0] + 1, a[0] + 1 + n);
        sort(a[1] + 1, a[1] + 1 + n);
        int ans = 0;
        for (int i = 0; i < 2; i++) {
            int t = 0, last = 0;
            memset(cnt, 0, sizeof(cnt));
            memset(sum, 0, sizeof(sum));
            for (int j = 1; j <= n; j++) {
                int l = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].l) -
                        all[i];
                int r = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].r) -
                        all[i];
                if (l < r) update(i, l, r - 1, a[i][j].d, 1, m[i], 1);
                t += abs(sum[1] - last);
                last = sum[1];
            }
            ans += t;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

9.7 凸包面积交

```
/*
    凸包面积交
    逆时针输入两个凸包，求相交的面积
```

```
    用一个凸包的边不断去切割另一个凸包，保留切割线内侧的点和交点
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
const int N = 1010;
struct Point {
    double x, y;
} p[N], q[N], t[N];
int T, n, m;
double a, b, c;
double Cal(Point p1) { return a * p1.x + b * p1.y + c; }
void Get_line(Point p1, Point p2) {
    a = p2.y - p1.y;
    b = p1.x - p2.x;
    c = p2.x * p1.y - p2.y * p1.x;
}
Point Intersetion(Point p1, Point p2) {
    Point ret;
    double u = fabs(Cal(p1)), v = fabs(Cal(p2));
    ret.x = (v * p1.x + u * p2.x) / (u + v);
    ret.y = (v * p1.y + u * p2.y) / (u + v);
    return ret;
}
void Cut() {
    int tm = 0;
    for (int i = 1; i <= m; i++) {
        if (Cal(q[i]) <= 0)
            t[++tm] = q[i];
        else {
            if (Cal(q[i - 1]) < 0) t[++tm] = Intersetion(q[i - 1], q[i]);
            if (Cal(q[i + 1]) < 0) t[++tm] = Intersetion(q[i], q[i + 1]);
        }
    }
    for (int i = 1; i <= tm; i++) q[i] = t[i];
    q[0] = q[tm];
    q[tm + 1] = q[1];
    m = tm;
}
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &m, &n);
        for (int i = 1; i <= m; i++) scanf("%lf%lf", &q[i].x, &q[i].y);
        q[0] = q[m], q[m + 1] = q[1];
```

```
        for (int i = 1; i <= n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);
        p[n + 1] = p[1];
        for (int i = 1; i <= n; i++) {
            Get_line(p[i], p[i + 1]);
            Cut();
        }
        double ans = 0;
        for (int i = 1; i <= m; i++)
            ans += q[i].x * q[i + 1].y - q[i + 1].x * q[i].y;
        printf("%.4f\n", ans / 2);
    }
    return 0;
}
```

9.8 凸包面积并

```
/*
    凸包面积并
*/
#include <bits/stdc++.h>
#define all(x) x.begin(), x.end()
using namespace std;
const double pi = acos(-1.0);
typedef long long LL;
typedef unsigned long long ULL;
void umax(int &a, int b) { a = max(a, b); }
void umin(int &a, int b) { a = min(a, b); }
namespace Solver {
#define PDI pair<double, int>
#define point pair<double, double>
#define mp make_pair
#define pb push_back
#define x first
#define y second
#define zero 1e-8
#define maxN 111
#define maxp 30
point operator+(point a, point b) { return mp(a.x + b.x, a.y + b.y); }
point operator-(point a, point b) { return mp(a.x - b.x, a.y - b.y); }
double operator*(point a, point b) { return a.x * b.y - b.x * a.y; }
double operator^(point a, point b) { return a.x * b.x + a.y * b.y; }
inline double cross(point o, point a, point b) { return (a - o) * (b - o); }
inline int cmp(double x) {
    if (fabs(x) < zero) return 0;
    return x > 0 ? 1 : -1;
```

```
}
class Polygon {
  private:
   int i;
   double s;

  public:
   int n;
   point p[maxp];
   point &operator[](int idx) { return p[idx]; }
   void input() {
      for (i = 0; i < n; i++) scanf("%lf %lf", &p[i].x, &p[i].y);
      p[n] = p[0];
   }
   double Area() {
      for (s = 0, i = 0; i < n; i++) s += p[i] * p[i + 1];
      return s / 2;
   }
};
PDI s[maxN * maxp * 2];
Polygon P[maxN];
double S, ts;
int N;
inline double seg(point o, point a, point b) {
   if (cmp(b.x - a.x) == 0) return (o.y - a.y) / (b.y - a.y);
   return (o.x - a.x) / (b.x - a.x);
}
double PolygonUnion() {
   int M, c1, c2;
   double s1, s2, ret = 0;
   for (int i = 0; i < N; i++) {
      for (int ii = 0; ii < P[i].n; ii++) {
         M = 0;
         s[M++] = mp(0.00, 0);
         s[M++] = mp(1.00, 0);
         for (int j = 0; j < N; j++)
            if (i != j)
               for (int jj = 0; jj < P[j].n; jj++) {
                  c1 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj]));
                  c2 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj + 1]));
                  if (c1 == 0 && c2 == 0) {
                     if (((P[i][ii + 1] - P[i][ii]) ^
                          (P[j][jj + 1] - P[j][jj])) > 0 &&
                         i > j) {
                         s[M++] = mp(
                            seg(P[j][jj], P[i][ii], P[i][ii + 1]), 1);
```

```
                        s[M++] = mp(
                            seg(P[j][jj + 1], P[i][ii], P[i][ii + 1]),
                            -1);
                    }
                } else {
                    s1 = cross(P[j][jj], P[j][jj + 1], P[i][ii]);
                    s2 = cross(P[j][jj], P[j][jj + 1], P[i][ii + 1]);
                    if (c1 >= 0 && c2 < 0)
                        s[M++] = mp(s1 / (s1 - s2), 1);
                    else if (c1 < 0 && c2 >= 0)
                        s[M++] = mp(s1 / (s1 - s2), -1);
                }
            }
        sort(s, s + M);
        double pre = min(max(s[0].x, 0.0), 1.0), now;
        double sum = 0;
        int cov = s[0].y;
        for (int j = 1; j < M; j++) {
            now = min(max(s[j].x, 0.0), 1.0);
            if (!cov) sum += now - pre;
            cov += s[j].y;
            pre = now;
        }
        ret += P[i][ii] * P[i][ii + 1] * sum;
    }
  }
  return ret / 2;
}
void solve() {
  scanf("%d", &N);
  for (int i = 0; i < N; i++) {
    scanf("%d", &P[i].n);
    P[i].input();
    S -= P[i].Area();
    reverse(P[i].p, P[i].p + P[i].n);
    P[i][P[i].n] = P[i][0];
  }
  // 凸包面积和，凸包面积并
  printf("%.9f %.9f\n", S, PolygonUnion());
}
}; // namespace Solver
int main() {
  Solver::solve();
  return 0;
}
```

9.9 半平面交

```
/*
    半平面交+凸包内缩
*/
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-8;
const double PI = acos(-1.0);
const int N = 1010;
int sgn(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0)
        return -1;
    else
        return 1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double _x, double _y) {
        x = _x;
        y = _y;
    }
    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); }
    double operator^(const Point &b) const { return x * b.y - y * b.x; }
    double operator*(const Point &b) const { return x * b.x + y * b.y; }
};
int dist2(Point a, Point b) { return (a - b) * (a - b); }
struct Line {
    Point s, e;
    double k;
    Line() {}
    Line(Point _s, Point _e) {
        s = _s;
        e = _e;
        k = atan2(e.y - s.y, e.x - s.x);
    }
    Point operator&(const Line &b) const {
        Point res = s;
        double t = ((s - b.s) ^ (b.s - b.e)) / ((s - e) ^ (b.s - b.e));
        res.x += (e.x - s.x) * t;
        res.y += (e.y - s.y) * t;
        return res;
    }
};
```

```
// 半平面交，直线的左边代表有效区域
bool HPIcmp(Line a, Line b) {
    if (fabs(a.k - b.k) > eps) return a.k < b.k;
    return ((a.s - b.s) ^ (b.e - b.s)) < 0;
}
Line Q[N];
void HPI(Line line[], int n, Point res[], int &resn) {
    int tot = n;
    sort(line, line + n, HPIcmp);
    tot = 1;
    for (int i = 1; i < n; i++)
        if (fabs(line[i].k - line[i - 1].k) > eps) line[tot++] = line[i];
    int head = 0, tail = 1;
    Q[0] = line[0];
    Q[1] = line[1];
    resn = 0;
    for (int i = 2; i < tot; i++) {
        if (fabs((Q[tail].e - Q[tail].s) ^ (Q[tail - 1].e - Q[tail - 1].s)) <
                eps ||
            fabs((Q[head].e - Q[head].s) ^ (Q[head + 1].e - Q[head + 1].s)) <
                eps)
            return;
        while (head < tail && (((Q[tail] & Q[tail - 1]) - line[i].s) ^
                               (line[i].e - line[i].s)) > eps)
            tail--;
        while (head < tail && (((Q[head] & Q[head + 1]) - line[i].s) ^
                               (line[i].e - line[i].s)) > eps)
            head++;
        Q[++tail] = line[i];
    }
    while (head < tail && (((Q[tail] & Q[tail - 1]) - Q[head].s) ^
                           (Q[head].e - Q[head].s)) > eps)
        tail--;
    while (head < tail && (((Q[head] & Q[head - 1]) - Q[tail].s) ^
                           (Q[tail].e - Q[tail].e)) > eps)
        head++;
    if (tail <= head + 1) return;
    for (int i = head; i < tail; i++) res[resn++] = Q[i] & Q[i + 1];
    if (head < tail - 1) res[resn++] = Q[head] & Q[tail];
}
Point p[N];
Line line[N];
// 两点间距离
double dist(Point a, Point b) { return sqrt((a - b) * (a - b)); }
// 将线段ab往左移动距离p
void change(Point a, Point b, Point &c, Point &d, double p) {
```

```
    double len = dist(a, b);
    double dx = (a.y - b.y) * p / len;
    double dy = (b.x - a.x) * p / len;
    c.x = a.x + dx;
    c.y = a.y + dy;
    d.x = b.x + dx;
    d.y = b.y + dy;
}
/*
    Problem:
        给出一个凸包，现在要往里面放三个半径为r的圆，圆可以相交，
        求三个圆圆心相连构成的三角形最大面积的两倍
    Solution:
        半平面交获取圆心可放置位置的边界，对凸包顶点用旋转卡壳求最大面积三角形
*/
// 旋转卡壳，求三角形面积最大值的两倍
double rotating_calipers(Point p[], int n) {
    double ans = 0;
    Point v;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        while (j != i && k != i) {
            ans = max(ans, abs((p[j] - p[i]) ^ (p[k] - p[i])));
            while (((p[i] - p[j]) ^ (p[(k + 1) % n] - p[k])) < 0)
                k = (k + 1) % n;
            j = (j + 1) % n;
        }
    }
    return ans;
}
// 求凸包
Point List[N];
int Stack[N], top; // top为凸包的大小
bool _cmp(Point p1, Point p2) {
    double tmp = (p1 - List[0]) ^ (p2 - List[0]);
    if (tmp > 0)
        return true;
    else if (tmp == 0 && dist2(p1, List[0]) <= dist2(p2, List[0]))
        return true;
    else
        return false;
}
void Graham(int n) {
    Point p0;
    int k = 0;
```

```
    p0 = List[0];
    for (int i = 1; i < n; i++)
        if (p0.y > List[i].y || (p0.y == List[i].y && p0.x > List[i].x)) {
            p0 = List[i];
            k = i;
        }
    swap(List[0], List[k]);
    sort(List + 1, List + n, _cmp);
    if (n == 1) {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if (n == 2) {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for (int i = 2; i < n; i++) {
        while (top > 1 && ((List[Stack[top - 1]] - List[Stack[top - 2]]) ^
                           (List[i] - List[Stack[top - 2]])) <= 0)
            top--;
        Stack[top++] = i;
    }
}
int main() {
    int n, T;
    double r;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%lf", &n, &r);
        for (int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);
        reverse(p, p + n);
        for (int i = 0; i < n; i++) {
            Point t1, t2;
            change(p[i], p[(i + 1) % n], t1, t2, r);
            line[i] = Line(t1, t2);
        }
        int resn;
        HPI(line, n, p, resn);
        for (int i = 0; i < resn; i++) List[i].x = p[i].x, List[i].y = p[i].y;
        Graham(resn);
```

```
        for (int i = 0; i < top; i++) p[i] = List[Stack[i]];
        double ans = rotating_calipers(p, top); // 注意参数
        printf("%.12f\n", ans);
    }
    return 0;
}
```

9.10 旋转卡壳

```
/*
    旋转卡壳
    注意在使用前先求凸包
*/
// 旋转卡壳，求平面最远点对
double rotating_calipers(Point p[], int n) {
    double ans = 0;
    Point v;
    int cur = 1, ans1 = 0, ans2 = 0;
    for (int i = 0; i < n; i++) {
        v = p[i] - p[(i + 1) % n];
        while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0) cur = (cur + 1) % n;
        double tmp = dist(p[i], p[cur]);
        if (tmp > ans) ans = tmp, ans1 = i, ans2 = cur;
        tmp = dist(p[(i + 1) % n], p[(cur + 1) % n]);
        if (tmp > ans) ans = tmp, ans1 = (i + 1) % n, ans2 = (cur + 1) % n;
    }
    printf("%.12f %.12f %.12f %.12f\n", p[ans1].x, p[ans1].y, p[ans2].x,
          p[ans2].y);
    return ans;
}
// 旋转卡壳，求三角形面积最大值的两倍
double rotating_calipers(Point p[], int n) {
    double ans = 0;
    Point v;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        while (j != i && k != i) {
            ans = max(ans, abs((p[j] - p[i]) ^ (p[k] - p[i])));
            while (((p[i] - p[j]) ^ (p[(k + 1) % n] - p[k])) < 0)
                k = (k + 1) % n;
            j = (j + 1) % n;
        }
    }
    return ans;
```

```
}
// 旋转卡壳，求两个凸包的最小距离
// 点p0到线段p1p2的距离
double pointtoseg(Point p0, Point p1, Point p2) {
    return dist(p0, NearestPointToLineSeg(p0, Line(p1, p2)));
}
// 平行线段p0p1和p2p3的距离
double dispallseg(Point p0, Point p1, Point p2, Point p3) {
    double ans1 = min(pointtoseg(p0, p2, p3), pointtoseg(p1, p2, p3));
    double ans2 = min(pointtoseg(p2, p0, p1), pointtoseg(p3, p0, p1));
    return min(ans1, ans2);
}
// 得到向量a1a2和b1b2的位置关系
double Get_angle(Point a1, Point a2, Point b1, Point b2) {
    Point t = b1 - (b2 - a1);
    return (a2 - a1) ^ (t - a1);
}
double rotating_calipers(Point p[], int np, Point q[], int nq) {
    int sp = 0, sq = 0;
    for (int i = 0; i < np; i++)
        if (sgn(p[i].y - p[sp].y) < 0) sp = i;
    for (int i = 0; i < nq; i++)
        if (sgn(q[i].y - q[sq].y) > 0) sq = i;
    double tmp;
    double ans = 1e99;
    for (int i = 0; i < np; i++) {
        while (sgn(tmp = Get_angle(p[sp], p[(sp + 1) % np], q[sq],
                                   q[(sq + 1) % nq])) < 0)
            sq = (sq + 1) % nq;
        if (sgn(tmp) == 0)
            ans = min(ans, dispallseg(p[sp], p[(sp + 1) % np], q[sq],
                                      q[(sq + 1) % nq]));
        else
            ans = min(ans, pointtoseg(q[sq], p[sp], p[(sp + 1) % np]));
        sp = (sp + 1) % np;
    }
    return ans;
}
double solve(Point p[], int n, Point q[], int m) {
    return min(rotating_calipers(p, n, q, m), rotating_calipers(q, m, p, n));
}
```

9.11 圆交

```
/*
```

```
    求圆交
    Example:
        给出两个圆环，圆心位置不同，但是大小圆半径相同，求两个圆环相交的面积
    Solution:
        面积为大圆交面积-2*大小圆交面积+小圆交面积
*/
using namespace std;
const double EPS = 1e-12;
const double PI = acos(-1.0);
const int inf = ~0U >> 1;
int sgn(double x) {
    if (fabs(x) < EPS) return 0;
    if (x < 0)
        return -1;
    else
        return 1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double x1, double y1) {
        x = x1;
        y = y1;
    }
    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); }
    double operator*(const Point &b) const { return x * b.x + y * b.y; } //点积
    double operator^(const Point &b) const { return x * b.y - y * b.x; } //叉积
};
struct Line {
    Point u, v;
    Line() {}
    Line(Point u1, Point v1) {
        u = u1;
        v = v1;
    }
};
double dist(Point a, Point b) { return sqrt((a - b) * (a - b)); }
// 两圆相交部分面积，a，b是圆心，r1，r2是半径
double Area(Point a, double r1, Point b, double r2) {
    double k = dist(a, b);
    if (k + EPS >= r1 + r2) return 0; //相切或者相离
    if (k <= fabs(r1 - r2) + EPS) {
        double R = min(r1, r2);
        return PI * R * R;
    }
    double x = (k * k + r1 * r1 - r2 * r2) / (2.0 * k);
```

```
    double w1 = acos(x / r1);
    double w2 = acos((k - x) / r2);
    return (w1 * r1 * r1 + w2 * r2 * r2 - k * r1 * sin(w1));
}
int main(int T) {
    scanf("%d", &T);
    int cnt = 1;
    while (T--) {
        double x, y, r, R;
        Point p1, p2;
        scanf("%lf%lf", &r, &R);
        scanf("%lf%lf", &x, &y);
        p1 = Point(x, y);
        scanf("%lf%lf", &x, &y);
        p2 = Point(x, y);
        double ans =
            Area(p1, R, p2, R) - 2 * Area(p1, R, p2, r) + Area(p1, r, p2, r);
        printf("Case #%d: %.6f\n", cnt++, ans);
    }
    return 0;
}
```

9.12 K 次圆并

```
/*
    K次圆并
    memset(cir,0,sizeof(cir))
    >> [圆赋值]
    memset(area,0,sizeof(area))
    CirUnion(cir,n);
    area[K]: 表示被至少K个圆覆盖的面积大小
*/
namespace KD_Circle_Merge {
#define sqr(x) ((x) * (x))
const double eps = 1e-8;
const double pi = acos(-1.0);
double area[N];
int dcmp(double x) {
    if (x < -eps)
        return -1;
    else
        return x > eps;
}
struct cp {
    double x, y, r, angle;
```

```
    int d;
    cp() {}
    cp(double xx, double yy, double ang = 0, int t = 0) {
        x = xx;
        y = yy;
        angle = ang;
        d = t;
    }
    void get() {
        scanf("%lf%lf%lf", &x, &y, &r);
        d = 1;
    }
} cir[N], tp[N * 2];
double dis(cp a, cp b) { return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)); }
double cross(cp p0, cp p1, cp p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
int CirCrossCir(cp p1, double r1, cp p2, double r2, cp& cp1, cp& cp2) {
    double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
    double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
    double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
    if (d + eps < 0) return 0;
    if (d < eps)
        d = 0;
    else
        d = sqrt(d);
    double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
    double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d;
    sq *= 2;
    cp1.x = (x - dy) / sq;
    cp1.y = (y + dx) / sq;
    cp2.x = (x + dy) / sq;
    cp2.y = (y - dx) / sq;
    if (d > eps)
        return 2;
    else
        return 1;
}
bool circmp(const cp& u, const cp& v) { return dcmp(u.r - v.r) < 0; }
bool cmp(const cp& u, const cp& v) {
    if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
    return u.d > v.d;
}
double calc(cp cir, cp cp1, cp cp2) {
    double ans = (cp2.angle - cp1.angle) * sqr(cir.r) - cross(cir, cp1, cp2) +
```

```
            cross(cp(0, 0), cp1, cp2);
    return ans / 2;
}
void CirUnion(cp cir[], int n) {
    cp cp1, cp2;
    sort(cir, cir + n, circmp);
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
                cir[i].d++;
    for (int i = 0; i < n; i++) {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; j++) {
            if (i == j) continue;
            if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r, cp2, cp1) < 2)
                continue;
            cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
            cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
            cp1.d = 1;
            tp[tn++] = cp1;
            cp2.d = -1;
            tp[tn++] = cp2;
            if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
        }
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
        sort(tp, tp + tn, cmp);
        int p, s = cir[i].d + tp[0].d;
        for (int j = 1; j < tn; j++) {
            p = s;
            s += tp[j].d;
            area[p] += calc(cir[i], tp[j - 1], tp[j]);
        }
    }
}
} // namespace KD_Circle_Merge
```

9.13 圆上扫描线

```
/*
    圆上扫描线
    Example: 给出若干个圆，可以互相嵌套但不相交或相切。
    每次删去一个圆和它内部的圆，进行博弈，问谁赢。
    Solution: 我们将最外面一层空间看做一个大圆，则这个游戏等价于树上删边游戏，
    则只要处理圆上关系即可，我们用圆上扫描线来处理。
```

```
    (注意只能处理包含和相离的情况)
*/
#include <algorithm>
#include <cstdio>
#include <set>
#include <vector>
using namespace std;
const int N = 100010;
vector<int> v[N];
namespace Cir_SCLine {
const int U = 0, D = 1, L = 0, R = 1;
int Time, n;
struct Circle {
    int x, y, r, w, f;
    void read() {
        w = 0;
        scanf("%d%d%d", &x, &y, &r);
    }
    int getX(int dir) { return dir == L ? x - r : x + r; }
    double getY(int dir) {
        double res = sqrt((double)r * r - (double)(Time - x) * (Time - x));
        return dir == U ? res + y : -res + y;
    }
} cir[N];
struct Event {
    int x, y, id, dir;
    void get(int _x, int _y, int _id, int _dir) {
        x = _x;
        y = _y;
        id = _id;
        dir = _dir;
    };
    bool operator<(const Event rhs) const {
        return x < rhs.x || x == rhs.x && y > rhs.y;
    }
} et[N << 1];
struct node {
    int id, dir;
    node() {}
    node(int _id, int _dir) {
        id = _id;
        dir = _dir;
    }
    bool operator<(const node rhs) const {
        double y1 = cir[id].getY(dir);
        double y2 = cir[rhs.id].getY(rhs.dir);
```

```
        return y1 > y2 || y1 == y2 && dir < rhs.dir;
    }
};
set<node> line;
set<node>::iterator it, it1, it2;
int etn;
void SCLine() {
    line.clear();
    for (int i = 0; i < etn; i++) {
        Time = et[i].x;
        if (et[i].dir == R) {
            line.erase(node(et[i].id, U));
            line.erase(node(et[i].id, D));
        } else {
            it = line.insert(node(et[i].id, U)).first;
            it1 = it2 = it;
            it1++;
            int id = it->id;
            // 当该圆上的点在扫描线边界时，为一层圆
            if (it == line.begin() || it1 == line.end())
                cir[id].w = 1;
            else {
                it2--;
                if (it2->id == it1->id) {
                    // 当扫描线上在当前圆上方和下方的点属于同个圆时，被该圆包含
                    cir[id].w = cir[it2->id].w + 1;
                    cir[id].f = it2->id;
                    v[it2->id].push_back(id);
                } else {
                    // 当上下方的点不属于同个圆时
                    cir[id].w = max(cir[it1->id].w, cir[it2->id].w);
                    if (cir[it1->id].w == cir[it2->id].w) {
                        // 当上下方圆层次相同，则两个圆是该圆的同级圆，被同个圆包含
                        cir[id].f = cir[it2->id].f;
                        v[cir[it2->id].f].push_back(id);
                    } else {
                        // 当上下方圆层次不同，被层次较小的圆包含，较大的是同级圆
                        int f = it1->id;
                        if (cir[it1->id].w > cir[it2->id].w) f = it2->id;
                        cir[id].f = f;
                        v[f].push_back(id);
                    }
                }
            }
            line.insert(node(et[i].id, D));
        }
```

```
    }
}
} // namespace Cir_SCLine
using namespace Cir_SCLine;
int val[N];
int SG(int x) {
    int sg = 0;
    for (int i = 0; i < v[x].size(); i++) sg ^= SG(v[x][i]);
    val[x] += sg;
    return val[x];
}
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        for (int i = 0; i <= n; i++) v[i].clear();
        etn = 0;
        cir[0].x = cir[0].y = 0;
        cir[0].r = 100000;
        et[etn++].get(cir[0].getX(L), cir[0].y, 0, L);
        et[etn++].get(cir[0].getX(R), cir[0].y, 0, R);
        for (int i = 1; i <= n; i++) {
            cir[i].read();
            et[etn++].get(cir[i].getX(L), cir[i].y, i, L);
            et[etn++].get(cir[i].getX(R), cir[i].y, i, R);
        }
        sort(et, et + etn);
        SCLine();
        val[0] = 0;
        for (int i = 1; i <= n; i++) val[i] = 1;
        puts(SG(0) ? "Alice" : "Bob");
    }
    return 0;
}
```

9.14 二维几何模板

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
#define EPS 1e-11
```

```
#define INF 1e18
#define PI 3.14159265358979323846
#define EQ(t1, t2) (abs((t1) - (t2)) < EPS)
#define LE(t1, t2) ((t1) <= (t2)-EPS)
#define LEQ(t1, t2) ((t1) < (t2) + EPS)
#define NEXT(i, n) ((i) + 1 >= (n) ? 0 : (i) + 1)
#define PREV(i, n) ((i) > 0 ? (i)-1 : (n)-1)
inline int SGN(double t) { return LE(t, 0) ? -1 : LE(0, t) ? 1 : 0; }
struct Point {
    double x, y;
    bool operator==(const Point& p) const { return EQ(x, p.x) && EQ(y, p.y); }
    bool operator<(const Point& p) const {
        return LEQ(x, p.x) && (LE(x, p.x) || LE(y, p.y));
    }
    Point operator+(Point& p) { return {x + p.x, y + p.y}; }
    Point operator-(Point& p) { return {x - p.x, y - p.y}; }
    double operator*(Point& p) { return x * p.y - y * p.x; }
    Point operator*(double value) { return {x * value, y * value}; }
    Point operator/(double value) { return {x / value, y / value}; }
    double dot(Point& p) { return x * p.x + y * p.y; }
    double r2() { return x * x + y * y; }
    double r() { return sqrt(x * x + y * y); }
    double dis2(Point& p) { return (*this - p).r2(); }
    double dis(Point& p) { return (*this - p).r(); }
    // 1: 点在直线左边 -1: 点在直线右边 0: 点在直线上
    int direction(Point& p1, Point& p2) {
        return SGN(x * (p1.y - p2.y) + p1.x * (p2.y - y) + p2.x * (y - p1.y));
    }
    bool onLine(Point& p1, Point& p2) { return direction(p1, p2) == 0; }
    // 判断点是否在线段上
    bool onLineSeg(Point& p1, Point& p2) {
        return onLine(p1, p2) && inRect(p1, p2);
    }
    /*
        0: p1p垂直p1p2
        1: p2p垂直p1p2
        (0,1): p在p1点垂线与p2点垂线之间
    */
    double lineRelation(Point& p1, Point& p2) {
        Point t = p2 - p1;
        return t.dot(*this - p1) / t.r2();
    }
    Point footPoint(Point& p1, Point& p2) {
        double r = lineRelation(p1, p2);
        return p1 + (p2 - p1) * r;
    }
```

```
    // 与直线的距离
    double lineDis(Point& p1, Point& p2) {
        return abs((p1 - *this) * (p2 - *this)) / p1.dis(p2);
    }
    double lineSegDis(Point& p1, Point& p2, Point& ret);
    double lineSegArrayDis(Point* p, int lineNum, Point& ret);
    // 关于直线的对称点
    Point mirror(Point& p1, Point& p2) {
        Point foot = footPoint(p1, p2);
        return foot * 2 - *this;
    }
    // 逆时针旋转
    Point rotate(double angle) {
        Point f = {sin(angle), cos(angle)};
        return {*this * f, dot(f)};
    }
    Point rotate90() { return {-y, x}; }
    double cosAngle(Point& p1, Point& p2) {
        Point t1 = *this - p1, t2 = *this - p2;
        return t1.dot(t2) / sqrt(t1.r2() * t2.r2());
    }
    double sinAngle(Point& p1, Point& p2) {
        Point t1 = *this - p1, t2 = *this - p2;
        return abs(t1 * t2) / sqrt(t1.r2() * t2.r2());
    }
    double tanAngle(Point& o) {
        if (EQ(x, o.x)) return y - o.y >= 0 ? INF : -INF;
        return (y - o.y) / (x - o.x);
    }
    double angle(Point& p1, Point& p2) { return acos(cosAngle(p1, p2)); }
    double angle(Point& o) { return atan2(y - o.y, x - o.x); }
    bool inRect(Point& p1, Point& p2) {
        return LEQ((p1.x - x) * (p2.x - x), 0) &&
               LEQ((p1.y - y) * (p2.y - y), 0);
    }
    int inPolygon(Point* p, int n);
    int inConvex(Point* p, int n);
    int inCircle(Point& o, double r) {
        double dist = dis2(o);
        return SGN(r * r - dist);
    }
    void pointcut(Point& o, double r, Point& ret1, Point& ret2);
    Point nearnestPoint(Point& o, double r);
};
double Point::lineSegDis(Point& p1, Point& p2, Point& ret) {
    double r = lineRelation(p1, p2);
```

```
    if (LEQ(r, 0))
        ret = p1;
    else if (LEQ(1, r))
        ret = p2;
    else
        ret = footPoint(p1, p2);
    return dis(ret);
}
// input lineNum+1 points
double Point::lineSegArrayDis(Point* p, int lineNum, Point& ret) {
    Point tp;
    double td, mind = INF;
    for (int i = 0; i < lineNum; i++) {
        td = lineSegDis(p[i], p[i + 1], tp);
        if (LE(td, mind)) {
            mind = td;
            ret = tp;
        }
    }
    return mind;
}
// donnot include extream points, and donnot include coincidence.
inline bool lineSegLineSegIntersect(Point& p1, Point& p2, Point& q1,
                                    Point& q2) {
    Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;
    return SGN(pq1 * q12) * SGN((p2 - q1) * q12) < 0 &&
           SGN(pq1 * p12) * SGN((p1 - q2) * p12) < 0;
}
// include extream points and coincidence.
inline bool lineSegLineSegIntersect2(Point& p1, Point& p2, Point& q1,
                                     Point& q2) {
    if (!LEQ(min(q1.x, q2.x), max(p1.x, p2.x)) ||
        !LEQ(min(p1.x, p2.x), max(q1.x, q2.x)) ||
        !LEQ(min(q1.y, q2.y), max(p1.y, p2.y)) ||
        !LEQ(min(p1.y, p2.y), max(q1.y, q2.y)))
        return false;
    Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;
    return SGN(pq1 * q12) * SGN((p2 - q1) * q12) <= 0 &&
           SGN(pq1 * p12) * SGN((p1 - q2) * p12) <= 0;
}
// donot include extream points, and donot include coincidence.
inline bool lineLineSegIntersect(Point& l1, Point& l2, Point& p1, Point& p2) {
    Point line = l2 - l1;
    return SGN((p1 - l1) * line) * SGN((p2 - l1) * line) < 0;
}
// donnot include coincidence.
```

```
inline bool lineLineIntersect(Point& p1, Point& p2, Point& q1, Point& q2) {
    return !EQ((p2 - p1) * (q2 - q1), 0);
}
inline Point lineLineIntersectPoint(Point& p1, Point& p2, Point& q1,
                                    Point& q2) {
    Point q12 = q2 - q1;
    double k = (p2 - p1) * q12;
    if (EQ(k, 0)) return {INF * INF, INF * INF};
    double r = ((q1 - p1) * q12) / k;
    return p1 + (p2 - p1) * r;
}
// 外心
Point circumcenter(Point& p1, Point& p2, Point& p3) {
    Point t1 = (p1 + p2) * 0.5, t2, t3 = (p2 + p3) * 0.5, t4;
    t2 = t1 + (p1 - p2).rotate90();
    t4 = t3 + (p2 - p3).rotate90();
    return lineLineIntersectPoint(t1, t2, t3, t4);
}
// 内心
Point incenter(Point& p1, Point& p2, Point& p3) {
    double r12 = p1.dis(p2), r23 = p2.dis(p3), r31 = p3.dis(p1);
    Point t1 = (p2 * r31 + p3 * r12) / (r12 + r31),
          t2 = (p1 * r23 + p3 * r12) / (r12 + r23);
    return lineLineIntersectPoint(p1, t1, p2, t2);
}
// 垂心
Point prepencenter(Point& p1, Point& p2, Point& p3) {
    Point t1 = p1 + (p2 - p3).rotate90();
    Point t2 = p2 + (p1 - p3).rotate90();
    return lineLineIntersectPoint(p1, t1, p2, t2);
}
// 重心
inline Point barycenter(Point& p1, Point& p2, Point& p3) {
    return (p1 + p2 + p3) / 3;
}
// 内切圆
inline double apothem(Point& p1, Point& p2, Point& p3) {
    Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;
    return abs(p12 * p23) / (p12.r() + p13.r() + p23.r());
}
// 外接圆
inline double circumradius(Point& p1, Point& p2, Point& p3) {
    Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;
    return sqrt(p12.r2() * p23.r2() * p13.r2()) / (2 * abs(p12 * p23));
}
// 逆时针1, 顺时针-1
```

```
int getPolygonDirection(Point* p, int n) {
    int index = 0;
    for (int i = 1; i < n; i++) {
        if (p[i] < p[index]) index = i;
    }
    return p[index].direction(p[NEXT(index, n)], p[PREV(index, n)]);
}
bool checkConvex(Point* p, int n) {
    int direction = p[0].direction(p[n - 1], p[1]);
    if (direction == 0) return false;
    if (p[n - 1].direction(p[n - 2], p[0]) != direction) return false;
    for (int i = n - 2; i > 0; i--) {
        if (p[i].direction(p[i - 1], p[i + 1]) != direction) return false;
    }
    return true;
}
// 注意顺时针面积为负
double polygonArea(Point* p, int n) {
    double area = 0;
    for (int i = n - 2; i > 0; i--) area += p[i].y * (p[i - 1].x - p[i + 1].x);
    area += p[0].y * (p[n - 1].x - p[1].x);
    area += p[n - 1].y * (p[n - 2].x - p[0].x);
    return area / 2;
}
// 内部返回1, 边界0, 外部-1
int Point::inPolygon(Point* p, int n) {
    int i, j = n - 1, odd = -1;
    for (i = 0; i < n; j = i++) {
        if (LE(p[i].y, y) != LE(p[j].y, y)) {
            double tx =
                (y - p[j].y) / (p[i].y - p[j].y) * (p[i].x - p[j].x) + p[j].x;
            if (LEQ(tx, x)) {
                if (LE(tx, x))
                    odd = -odd;
                else
                    return 0;
            }
        } else if (onLineSeg(p[i], p[j]))
            return 0;
    }
    return odd;
}
int Point::inConvex(Point* p, int n) {
    int _direction = p[1].direction(p[2], p[0]);
    if (direction(p[0], p[1]) != _direction) {
        if (onLineSeg(p[0], p[1])) return 0;
```

```
        return -1;
    }
    if (direction(p[n - 1], p[0]) != _direction) {
        if (onLineSeg(p[n - 1], p[0])) return 0;
        return -1;
    }
    int left = 2, right = n - 1;
    while (left < right) {
        int mid = (left + right) >> 1;
        if (direction(p[0], p[mid]) == _direction)
            left = mid + 1;
        else
            right = mid;
    }
    int ret = direction(p[left - 1], p[left]);
    return ret == _direction ? 1 : ret == 0 ? 0 : -1;
}
// 以下三函数只允许逆时针方向
// angle array size >= 2*n,return offset
int lineConvexIntersectPointInit(Point* p, int n, double angle[]) {
    int ret = 0;
    for (int i = 0, j = n - 1; i < n; j = i++) angle[j] = p[i].angle(p[j]);
    do
        angle[ret + n] = angle[ret++];
    while (LE(angle[ret - 1], angle[ret]) && ret < n);
    return ret;
}
// ret和ret2分别为距直线最近和最远的点下标
int lineConvexIntersect(Point& p1, Point& p2, Point* p, int n, double angle[],
                        int offset, int& ret1, int& ret2) {
    int pos[2];
    double k[2];
    k[0] = p1.angle(p2);
    k[1] = k[0] <= 0 ? k[0] + PI : k[0] - PI;
    for (int i = 0; i < 2; i++) {
        pos[i] = (upper_bound(angle + offset, angle + offset + n, k[i] - EPS) -
                  angle) %
                 n;
        if (p[pos[i]].onLine(p1, p2))
            return p[NEXT(pos[i], n)].onLine(p1, p2) ? 3 : 1;
    }
    ret1 = pos[0];
    ret2 = pos[1];
    return p[pos[0]].direction(p1, p2) == p[pos[1]].direction(p1, p2) ? 0 : 2;
}
void lineConvexIntersectPoint(Point& p1, Point& p2, Point* p, int n, int i1,
```

```
                        int i2, Point& ret1, Point& ret2) {
    for (int i = 0, l, r; i < 2; i++) {
        if (i) {
            l = min(i1, i2);
            r = max(i1, i2);
        } else {
            l = max(i1, i2);
            r = min(i1, i2) + n;
        }
        while (l < r) {
            int mid = (l + r) >> 1;
            if (p[mid % n].direction(p1, p2) == p[r % n].direction(p1, p2))
                r = mid;
            else
                l = mid + 1;
        }
        l %= n;
        (i ? ret1 : ret2) = lineLineIntersectPoint(p1, p2, p[l], p[PREV(l, n)]);
    }
}
bool lineSegInPolygon(Point p1, Point p2, Point* p, int n) {
    if (p2 < p1) swap(p1, p2);
    int s1 = p1.inPolygon(p, n), s2 = p2.inPolygon(p, n), id = -1, pos = 0;
    if (s1 == -1 || s2 == -1) return false;
    while (p[pos].onLine(p1, p2)) pos++;
    int i = pos, j = pos, d = p[j].direction(p1, p2), d1, d2 = d;
    do {
        i = NEXT(i, n);
        d1 = d2;
        d2 = p[i].direction(p1, p2);
        if (d2 * d == -1) {
            if (lineSegLineSegIntersect(p[i], p[j], p1, p2)) return false;
            if (d1 == 0 && p1 < p[id] && p[id] < p2 && p1 < p[j] && p[j] < p2)
                return false;
            d = d2;
        }
        if (d1 == 0 && d2 && p1 < p[j] && p[j] < p2) id = j;
        if (d2 == 0 && d1 && p1 < p[i] && p[i] < p2) id = i;
    } while ((j = i) != pos);
    if (s1 == 0 && s2 == 0) {
        if (id == -1) return ((p1 + p2) * 0.5).inPolygon(p, n) >= 0;
        Point q1 = p1, q2 = p2;
        for (int i = 0; i < n; i++) {
            if (p[i].onLine(p1, p2)) {
                if (p[i] < p[id]) q1 = max(q1, p[i]);
                if (p[id] < p[i]) q2 = min(q2, p[i]);
```

```
            }
        }
        return ((q1 + p[id]) * 0.5).inPolygon(p, n) >= 0 &&
               ((q2 + p[id]) * 0.5).inPolygon(p, n) >= 0;
    }
    return true;
}
Point gravityCenter(Point* p, int n) {
    if (n < 3) {
        if (n == 1)
            return p[0];
        else
            return (p[0] + p[1]) * 0.5;
    }
    double area = 0;
    Point ret = {0, 0};
    for (int i = 0, j = n - 1; i < n; j = i++) {
        double t = p[i] * p[j];
        area += t;
        ret.x += (p[i].x + p[j].x) * t;
        ret.y += (p[i].y + p[j].y) * t;
    }
    return ret / (3 * area);
}
// sort p[] first , ret[n] must be available to visit.
int convexHullSorted(Point* p, int n, Point* ret) {
    int j = 0;
    for (int i = 0; i < n; i++) {
        while (j >= 2 && p[i].direction(ret[j - 2], ret[j - 1]) != 1) j--;
        ret[j++] = p[i];
    }
    int mid = j + 1;
    for (int i = n - 2; i >= 0; i--) {
        while (j >= mid && p[i].direction(ret[j - 2], ret[j - 1]) != 1) j--;
        ret[j++] = p[i];
    }
    return j - 1;
}
void convexHullSorted(Point* p, int n, Point* up, int& retUp, Point* down,
                      int& retDown) {
    retUp = retDown = 0;
    for (int i = 0; i < n; i++) {
        while (retUp >= 2 && p[i].direction(up[retUp - 2], up[retUp - 1]) != -1)
            retUp--;
        while (retDown >= 2 &&
               p[i].direction(down[retDown - 2], down[retDown - 1]) != 1)
```

```
            retDown--;
        up[retUp++] = p[i];
        down[retDown++] = p[i];
    }
}
// p2绕p1逆时针转90度代表平面内部，自行增加4个半平面做边界
#define judge(p, q) (p.direction(q.second[0], q.second[1]) < 0)
#define intersect(p, q) \
    lineLineIntersectPoint(p.second[0], p.second[1], q.second[0], q.second[1])
int halfPlainIntersect(Point (*p)[2], int n, Point* ret) {
    vector<pair<double, Point*> > v(n), line(n);
    for (int i = 0; i < n; i++) v[i] = make_pair(p[i][1].angle(p[i][0]), p[i]);
    sort(v.begin(), v.end());
    int m = 0, l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (!EQ(v[i].first, v[m].first))
            v[++m] = v[i];
        else if (!judge(v[i].second[0], v[m]))
            v[m] = v[i];
    }
    if (EQ(v[0].first + 2 * PI, v[m].first) && !judge(v[m--].second[0], v[0]))
        v[0] = v[m + 1];
    vector<Point> q(n);
    line[0] = v[0];
    for (int i = 1; i <= m; i++) {
        while (l < r && judge(q[r - 1], v[i])) r--;
        while (l < r && judge(q[l], v[i])) l++;
        if (l == r && LEQ(line[l].first + PI, v[i].first)) return 0;
        line[++r] = v[i];
        q[r - 1] = intersect(line[r - 1], line[r]);
    }
    while (l < r && judge(q[r - 1], line[l])) r--;
    if (l == r) return 0;
    q[r] = intersect(line[l], line[r]);
    int num = unique(q.begin() + l, q.begin() + r + 1) - q.begin() - l;
    memcpy(ret, &q[l], sizeof(Point) * num);
    if (num > 1 && ret[0] == ret[num - 1]) num--;
    return num;
}
// These points must be put counter-clockwise.
int polygonKernel(Point* p, int n, Point* ret) {
    Point(*q)[2] = new Point[n][2];
    for (int i = 0, j = n - 1; i < n; j = i++) q[j][0] = p[j], q[j][1] = p[i];
    int m = halfPlainIntersect(&q[0], n, ret);
    delete[] q;
    return m;
```

```
}
// return two points representing ax+by<=c
void getPlain(double a, double b, double c, Point* p) {
    if (EQ(a, 0))
        p[0] = {0, c / b};
    else
        p[0] = {c / a, 0};
    p[1] = {-b, a};
    p[1] = p[1] + p[0];
}
// These points must be put counter-clockwise.
// Ensure p[n] exists and p[n]=p[0].
double convexDiameter(Point* p, int n, Point& ret1, Point& ret2) {
    double ret = 0;
    for (int i = 0, j = 1; i < n; i++) {
        double t1 = (p[i + 1] - p[i]) * (p[j] - p[i]), t2;
        for (; LE(t1, t2 = (p[i + 1] - p[i]) * (p[j + 1] - p[i])); t1 = t2) {
            if (++j == n) j = 0;
        }
        double td2 = p[i].dis2(p[j]);
        if (ret < td2) {
            ret = td2;
            ret1 = p[i];
            ret2 = p[j];
        }
        td2 = p[i + 1].dis2(p[j]);
        if (ret < td2) {
            ret = td2;
            ret1 = p[i + 1];
            ret2 = p[j];
        }
    }
    return sqrt(ret);
}
// These points must be put counter-clockwise.
// Ensure p[n] exists and p[n]=p[0].
double convexWidth(Point* p, int n) {
    double ret = INF;
    for (int i = 0, j = 1; i < n; i++) {
        double t1 = (p[i + 1] - p[i]) * (p[j] - p[i]), t2;
        for (; LE(t1, t2 = (p[i + 1] - p[i]) * (p[j + 1] - p[i])); t1 = t2) {
            if (++j == n) j = 0;
        }
        ret = min(ret, t1 / p[i].dis(p[i + 1]));
    }
    return ret;
```

```
}
struct NearestPointsStruct {
    Point p1, p2;
    double d2;
    vector<Point> v;
};
inline bool nearestPointsCmp(Point& p1, Point& p2) {
    return LEQ(p1.y, p2.y) && (LE(p1.y, p2.y) || LE(p1.x, p2.x));
}
void nearestPointsInternal(Point* p, int left, int right,
                           NearestPointsStruct& s) {
    if (right - left < 8) {
        for (int i = left; i < right; i++) {
            for (int j = i + 1; j < right; j++) {
                double td2 = p[j].dis2(p[i]);
                if (td2 < s.d2) {
                    s.d2 = td2;
                    s.p1 = p[i];
                    s.p2 = p[j];
                }
            }
        }
        return;
    }
    int mid = (left + right) >> 1;
    nearestPointsInternal(p, left, mid, s);
    nearestPointsInternal(p, mid, right, s);
    s.v.clear();
    double l = (p[mid - 1].x + p[mid].x) / 2, d = sqrt(s.d2);
    for (int i = mid - 1; i >= left && l - p[i].x < d; i--) s.v.push_back(p[i]);
    for (int i = mid; i < right && p[i].x - l < d; i++) s.v.push_back(p[i]);
    sort(s.v.begin(), s.v.end(), nearestPointsCmp);
    for (unsigned int i = 0; i < s.v.size(); i++) {
        for (unsigned int j = i + 1; j < s.v.size() && s.v[j].y - s.v[i].y < d;
             j++) {
            double td2 = s.v[j].dis2(s.v[i]);
            if (td2 < s.d2) {
                s.d2 = td2;
                s.p1 = s.v[i];
                s.p2 = s.v[j];
            }
        }
    }
}
double nearestPointsSorted(Point* p, int n, Point& ret1, Point& ret2) {
    NearestPointsStruct s;
```

```
    s.d2 = INF;
    s.v.reserve(n);
    nearestPointsInternal(p, 0, n, s);
    ret1 = s.p1;
    ret2 = s.p2;
    return sqrt(s.d2);
}
double farthestPointsSorted(Point* p, int n, Point& ret1, Point& ret2) {
    vector<Point> v(n + 1);
    int cnt = convexHullSorted(p, n, &*v.begin());
    v[n] = v[0];
    return convexDiameter(&*v.begin(), cnt, ret1, ret2);
}
int circleLineRelation(Point& o, double r, Point& p1, Point& p2) {
    double d = o.lineDis(p1, p2);
    if (LE(d, r)) return 1;
    if (LE(r, d)) return -1;
    return 0;
}
int circleCircleRelation(Point& o1, double r1, Point& o2, double r2) {
    double r = o1.dis(o2);
    if (LE(r1 + r2, r)) return 4;
    if (LEQ(r1 + r2, r)) return 3;
    double sub = abs(r1 - r2);
    if (LE(sub, r)) return 2;
    if (LEQ(sub, r)) return 1;
    return 0;
}
// include extream points.
bool circleLineSegIntersect(Point& o, double r, Point& p1, Point& p2) {
    int t1 = p1.inCircle(o, r), t2 = p2.inCircle(o, r);
    if (t1 >= 0 || t2 >= 0) return t1 != 1 || t2 != 1;
    double t = o.lineRelation(p1, p2);
    if (t >= 1 || t <= 0) return false;
    Point foot = p1 + (p2 - p1) * t;
    return foot.inCircle(o, r) >= 0;
}
// ret1 is near p1,ret2 is near p2.
void circleLineIntersect(Point& o, double r, Point& p1, Point& p2, Point& ret1,
                         Point& ret2) {
    Point foot = o.footPoint(p1, p2);
    double t = r * r - o.dis2(foot);
    t = LEQ(t, 0) ? 0 : sqrt(t / p1.dis2(p2));
    ret1 = foot + (p1 - p2) * t;
    ret2 = foot * 2 - ret1;
}
```

```
void circleCircleIntersectPoint(Point& o1, double r1, Point& o2, double r2,
                                Point& ret1, Point& ret2) {
    double d2 = o1.dis2(o2);
    double t1 = (r1 * r1 - r2 * r2) / (2 * d2) + 0.5;
    double t2 = r1 * r1 / d2 - t1 * t1;
    t2 = LEQ(t2, 0) ? 0 : sqrt(t2);
    Point foot = o1 + (o2 - o1) * t1;
    ret1 = foot + (o2 - o1).rotate90() * t2;
    ret2 = foot * 2 - ret1;
}
double circleCircleIntersectArea(Point& o1, double r1, Point& o2, double r2) {
    int r = circleCircleRelation(o1, r1, o2, r2);
    if (r >= 3) return 0;
    if (r <= 1) return min(r1, r2) * min(r1, r2) * PI;
    Point p1, p2;
    circleCircleIntersectPoint(o1, r1, o2, r2, p1, p2);
    double ret = r1 * r1 * o1.angle(p1, o2) + r2 * r2 * o2.angle(p1, o1);
    return ret - sqrt(o1.dis2(o2) * p1.dis2(p2)) / 2;
}
void Point::pointcut(Point& o, double r, Point& ret1, Point& ret2) {
    double t1 = r * r / dis2(o);
    Point foot = o + (*this - o) * t1;
    double t2 = t1 - t1 * t1;
    t2 = LEQ(t2, 0) ? 0 : sqrt(t2);
    ret1 = foot + (*this - o).rotate90() * t2;
    ret2 = foot * 2 - ret1;
}
// ret[0] and ret[2] are on circle o1,ret[1] and ret[3] are on circle o2.
void circleCirclePointcutOuter(Point& o1, double r1, Point& o2, double r2,
                               Point* ret) {
    Point o12 = o2 - o1;
    double d12 = o12.r2(), r = (r1 - r2) / d12;
    Point foot1 = o1 + o12 * (r * r1), foot2 = o2 + o12 * (r * r2);
    double t = 1 / d12 - r * r;
    t = LEQ(t, 0) ? 0 : sqrt(t);
    Point line = o12.rotate90();
    ret[0] = foot1 + line * (t * r1);
    ret[1] = foot2 + line * (t * r2);
    ret[2] = foot1 * 2 - ret[0];
    ret[3] = foot2 * 2 - ret[1];
}
void circleCirclePointcutInner(Point& o1, double r1, Point& o2, double r2,
                               Point* ret) {
    Point o12 = o2 - o1;
    double d12 = o12.r2(), r = (r1 + r2) / d12;
    Point foot1 = o1 + o12 * (r * r1), foot2 = o2 - o12 * (r * r2);
```

```
    double t = 1 / d12 - r * r;
    t = LEQ(t, 0) ? 0 : sqrt(t);
    Point line = o12.rotate90();
    ret[0] = foot1 + line * (t * r1);
    ret[1] = foot2 - line * (t * r2);
    ret[2] = foot1 * 2 - ret[0];
    ret[3] = foot2 * 2 - ret[1];
}
Point Point::nearnestPoint(Point& o, double r) {
    Point p = *this - o;
    double d = p.r();
    if (EQ(d, 0)) return o;
    return o + p * (r / d);
}
// Upset the order before using this function.
double minCoveringCircle(Point* p, int n, Point& ret) {
    if (n == 1) {
        ret = p[0];
        return 0;
    }
    double r2 = p[0].dis2(p[1]) / 4;
    ret = (p[0] + p[1]) * 0.5;
    for (int i = 2; i < n; i++) {
        if (LE(r2, ret.dis2(p[i]))) {
            ret = (p[0] + p[i]) * 0.5;
            r2 = p[0].dis2(p[i]) / 4;
            for (int j = 1; j < i; j++) {
                if (LE(r2, ret.dis2(p[j]))) {
                    ret = (p[i] + p[j]) * 0.5;
                    r2 = p[i].dis2(p[j]) / 4;
                    for (int k = 0; k < j; k++) {
                        if (LE(r2, ret.dis2(p[k]))) {
                            ret = circumcenter(p[i], p[j], p[k]);
                            r2 = ret.dis2(p[k]);
                        }
                    }
                }
            }
        }
    }
    return sqrt(r2);
}
int unitCoveringCircle(Point* p, int n, double r) {
    int ret = 0;
    vector<pair<double, bool> > v;
    v.reserve(2 * n);
```

```
    double t = r * r * 4;
    for (int i = 0; i < n; i++) {
        v.clear();
        int value = 0;
        for (int j = 0; j < n; j++) {
            if (LEQ(p[i].dis2(p[j]), t) && i != j) {
                double a = p[j].angle(p[i]);
                double b = acos(p[i].dis(p[j]) / r / 2);
                double t1 = a - b, t2 = a + b;
                if (t1 < -PI / 2) {
                    t1 += 2 * PI;
                    if (t2 < -PI / 2)
                        t2 += 2 * PI;
                    else
                        value++;
                }
                v.push_back(make_pair(t1, true));
                v.push_back(make_pair(t2, false));
            }
        }
        sort(v.begin(), v.end());
        if (value > ret) ret = value;
        for (unsigned int j = 0; j < v.size(); j++) {
            if (v[j].second) {
                value++;
                if (value > ret) ret = value;
            } else
                value--;
        }
    }
    return ret + 1;
}
/*
double circlePolygonAreaIntersect(Point& o,double r,Point* p,int n){
    double area=0;
    Point p1,p2;
    for(int i=0,j=n-1;i<n;j=i++){
        int f1=p[i].inCircle(o,r),f2=p[j].inCircle(o,r);
        if(f1>=0&&f2>=0)area+=(o-p[i])*(o-p[j]);
        else if(f1>=0&&f2<0){
            circleLineIntersect(o,r,p[i],p[j],p1,p2);
            area+=(o-p[i])*(o-p2);
            area+=asin(o.sinAngle(p2,p[j]))*r*r;
        }
        else if(f1<0&&f2>=0){
            circleLineIntersect(o,r,p[i],p[j],p1,p2);
```

```
            area+=(o-p1)*(o-p[j]);
            area+=asin(o.sinAngle(p[i],p1))*r*r;
        }
        else if(circleLineSegIntersect(o,r,p[i],p[j])){
            circleLineIntersect(o,r,p[i],p[j],p1,p2);
            area+=(o-p1)*(o-p2);
            area+=(asin(o.sinAngle(p[i],p1))+asin(o.sinAngle(p2,p[j])))*r*r;
        }
        else area+=asin(o.sinAngle(p[i],p[j]))*r*r;
    }
    return abs(area/2);
}
*/
```

10 卡常优化

10.1 Read

```
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;
}
```

10.2 Fread

```
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p2 = (p1 = buf) + fread(buf, 1, 100000, stdin);
        if (p1 == p2) return EOF;
    }
    return *p1++;
}
inline void read(int &x) {
    char c = nc(), b = 1;
    for (; !(c >= '0' && c <= '9'); c = nc())
        if (c == '-') b = -1;
    for (x = 0; c >= '0' && c <= '9'; x = x * 10 + c - '0', c = nc())
        ;
    x *= b;
}
```

10.3 数字快速读入

```
namespace fastIO {
#define BUF_SIZE 100000
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;
    while (blank(ch = nc()))
        ;
    if (IOerror) return;
    for (x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0')
        ;
}
#undef BUF_SIZE
}; // namespace fastIO
```

10.4 字符串快速读入

```
namespace fastIO {
#define BUF_SIZE 100000
#define OUT_SIZE 1000000
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 int read(char *s) {
    char ch = nc();
    for (; blank(ch); ch = nc())
        ;
    if (IOerror) return 0;
    for (; !blank(ch) && !IOerror; ch = nc()) *s++ = ch;
    *s = 0;
    return 1;
}
inline int RI(int &a) {
    char ch = nc();
    a = 0;
    for (; blank(ch); ch = nc())
        ;
    if (IOerror) return 0;
    for (; !blank(ch) && !IOerror; ch = nc()) a = a * 10 + ch - '0';
    return 1;
}
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;
}; // namespace fastIO
```

10.5 UnsafeMod

```
/*
    用于模为奇数且取模次数特别多的情况
    UnsafeMod::set_mod(P) 初始化模数
    x.get() 得到值
    各种运算后取模直接运算即可
*/
#include <bits/stdc++.h>
using namespace std;
typedef unsigned int uint32;
typedef long long int64;
typedef unsigned long long uint64;
typedef uint32 word;
typedef uint64 dword;
typedef int sword;
word mod;
struct UnsafeMod {
    word x;
    UnsafeMod() : x(0) {}
    UnsafeMod(word _x) : x(init(_x)) {}
    UnsafeMod& operator+=(const UnsafeMod& rhs) {
        (x += rhs.x) >= mod && (x -= mod);
        return *this;
    }
    UnsafeMod& operator-=(const UnsafeMod& rhs) {
        sword(x -= rhs.x) < 0 && (x += mod);
        return *this;
    }
    UnsafeMod& operator*=(const UnsafeMod& rhs) {
        x = reduce(dword(x) * rhs.x);
        return *this;
    }
```

```
    UnsafeMod operator+(const UnsafeMod& rhs) const { return UnsafeMod(*this) += rhs; }
    UnsafeMod operator-(const UnsafeMod& rhs) const { return UnsafeMod(*this) -= rhs; }
    UnsafeMod operator*(const UnsafeMod& rhs) const { return UnsafeMod(*this) *= rhs; }
    word get() const { return reduce(x); }
    static word modulus() { return mod; }
    static word init(word w) { return reduce(dword(w) * r2); }
    static void set_mod(word m) {
        mod = m;
        Modinv = mul_inv(mod);
        r2 = -dword(mod) % mod;
    }
    static word reduce(dword x) {
        word y = word(x >> word_bits) - word((dword(word(x) * Modinv) * mod) >> word_bits);
        return sword(y) < 0 ? y + mod : y;
    }
    static word mul_inv(word n, int e = 6, word x = 1) { return !e ? x : mul_inv(n, e - 1, x * (2
        - x * n)); }
}x,y;
```

11 SPOJ 系列

11.1 SPOJ-GSS

11.1.1 GSS1

```
/*
    题目大意：给出一个数列，多次询问区间最大连续子段和
    题解：线段树维护区间最大连续子段和gss，区间从最左元素开始的最大连续子段和lgss
    区间以最右元素为结尾的最大连续子段和rgss以及区间和s，信息传递并合并即可
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010, M = N << 2;
struct data {
    int s, lgss, rgss, gss;
} T[M];
int a[N], n, m;
data merge(data L, data R) {
    data res;
    res.s = L.s + R.s;
    res.lgss = max(L.lgss, L.s + R.lgss);
    res.rgss = max(L.rgss + R.s, R.rgss);
    res.gss = max(max(L.gss, R.gss), L.rgss + R.lgss);
    return res;
}
void build(int x, int l, int r) {
    if (l == r) {
        T[x].s = T[x].lgss = T[x].rgss = T[x].gss = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(x << 1, l, mid);
    build(x << 1 | 1, mid + 1, r);
    T[x] = merge(T[x << 1], T[x << 1 | 1]);
}
data query(int x, int l, int r, int L, int R) {
    if (L <= l && r <= R) return T[x];
    int mid = (l + r) >> 1;
    data lft, rht;
    if (L <= mid) lft = query(x << 1, l, mid, L, R);
    if (R > mid) rht = query(x << 1 | 1, mid + 1, r, L, R);
    if (R <= mid) return lft;
    if (L > mid) return rht;
    return merge(lft, rht);
```

```
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        build(1, 1, n);
        scanf("%d", &m);
        while (m--) {
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", query(1, 1, n, l, r).gss);
        }
    }
    return 0;
}
```

11.1.2 GSS2

```
/*
    题目大意：给出一个数列，多次询问区间最大连续子段和，计数时重复元素只算进子段和中一次
    题解：考虑离线对询问进行扫描线，线段树i位置表示i时间一直到当前的时间节点的累加值，
    区间最大值nmx表示当前时间为终点的后缀最大子段和，记mx为区间最大子段和，
    我们发现区间最大子段和mx其实就是后缀最大子段和的历史最大值，
    每次处理到节点i，只要在区间加上a[i]，就能更新所有的后缀和nmx，
    对于非重元素，更新区间为[1,i]，对于重复元素，更新区间为[lst[a[i]]+1,i]，
    区间更新需要打tag，相对的，我们还要维护一个历史最大标记mxtag，
    历史最大值的是需要当前的最大值加上下传的历史最大标记来更新的，
    在历史最大值更新之后才能用tag去更新当前的最大值，
    因为线段树标记发生变化的前提条件是上方没有任何标记，
    所以上下标记作用的时间域不会有交集，从而能保证正确性。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const LL INF = 0x3f3f3f3f3f3f3f3f;
const int N = 100010, M = N << 2;
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b;
    LL tag, mx, mxtag, nmx;
} T[M];
void build(int, int);
void Initialize(int n) {
```

```
    tot = 0;
    build(1, n);
}
void pb(int x) {
    T[T[x].l].mxtag = max(T[T[x].l].mxtag, T[T[x].l].tag + T[x].mxtag);
    T[T[x].r].mxtag = max(T[T[x].r].mxtag, T[T[x].r].tag + T[x].mxtag);
    T[T[x].l].tag += T[x].tag;
    T[T[x].r].tag += T[x].tag;
    T[T[x].l].mx = max(T[T[x].l].mx, T[T[x].l].nmx + T[x].mxtag);
    T[T[x].r].mx = max(T[T[x].r].mx, T[T[x].r].nmx + T[x].mxtag);
    T[T[x].l].nmx += T[x].tag;
    T[T[x].r].nmx += T[x].tag;
    T[x].tag = 0;
    T[x].mxtag = -INF;
}
void up(int x) {
    T[x].mx = max(T[T[x].l].mx, T[T[x].r].mx);
    T[x].nmx = max(T[T[x].l].nmx, T[T[x].r].nmx);
}
void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].l = T[x].r = T[x].tag = 0;
    T[x].mxtag = T[x].nmx = T[x].mx = -INF;
    if (l == r) {
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
}
void modify(int x, int y, int p) {
    if (T[x].a == T[x].b) {
        T[x].mx = T[x].nmx = p;
        return;
    }
    int mid = (T[x].a + T[x].b) >> 1;
    pb(x);
    if (mid >= y) modify(T[x].l, y, p);
    if (mid < y) modify(T[x].r, y, p);
    up(x);
}
void change(int x, int a, int b, int p) {
```

```
    if (T[x].a >= a && T[x].b <= b) {
        T[x].nmx += p;
        T[x].mx = max(T[x].mx, T[x].nmx);
        T[x].tag += p;
        T[x].mxtag = max(T[x].mxtag, T[x].tag);
        return;
    }
    int mid = (T[x].a + T[x].b) >> 1;
    pb(x);
    if (mid >= a) change(T[x].l, a, b, p);
    if (mid < b) change(T[x].r, a, b, p);
    up(x);
}
LL query(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].mx;
    int mid = (T[x].a + T[x].b) >> 1;
    pb(x);
    LL res = -INF;
    if (mid >= a) res = max(res, query(T[x].l, a, b));
    if (mid < b) res = max(res, query(T[x].r, a, b));
    return res;
}
} // namespace Segment_Tree
const int base = 100000;
LL ans[N];
int head[N], nxt[N], v[N], lst[N + base];
void add(int x, int y, int id) {
    nxt[id] = head[y];
    head[y] = id;
    v[id] = x;
}
int n, m, a[N];
int main() {
    while (~scanf("%d", &n)) {
        memset(head, 0, sizeof(head));
        memset(lst, 0, sizeof(lst));
        Segment_Tree::Initialize(n);
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        scanf("%d", &m);
        for (int i = 1; i <= m; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            add(x, y, i);
        }
        for (int i = 1; i <= n; i++) {
            Segment_Tree::modify(1, i, a[i]);
```

```
            if (lst[a[i] + base] + 1 < i)
                Segment_Tree::change(1, lst[a[i] + base] + 1, i - 1, a[i]);
            lst[a[i] + base] = i;
            for (int j = head[i]; j; j = nxt[j])
                ans[j] = Segment_Tree::query(1, v[j], i);
        }
        for (int i = 1; i <= m; i++) printf("%lld\n", max(ans[i], 0ll));
    }
    return 0;
}
```

11.1.3 GSS3

```
/*
    题目大意：给出一个数列，多次询问区间最大连续子段和，支持单点修改
    题解：线段树维护区间最大连续子段和gss，区间从最左元素开始的最大连续子段和lgss
    区间以最右元素为结尾的最大连续子段和rgss以及区间和s，信息传递并合并即可
    单点修改最多影响树上log个点，重新合并计算即可
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010, M = N << 2;
struct data {
    int s, lgss, rgss, gss;
} T[M];
int a[N], n, m;
data merge(data L, data R) {
    data res;
    res.s = L.s + R.s;
    res.lgss = max(L.lgss, L.s + R.lgss);
    res.rgss = max(L.rgss + R.s, R.rgss);
    res.gss = max(max(L.gss, R.gss), L.rgss + R.lgss);
    return res;
}
void build(int x, int l, int r) {
    if (l == r) {
        T[x].s = T[x].lgss = T[x].rgss = T[x].gss = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(x << 1, l, mid);
    build(x << 1 | 1, mid + 1, r);
    T[x] = merge(T[x << 1], T[x << 1 | 1]);
}
```

```
void change(int x, int l, int r, int y, int p) {
    if (l == r) {
        T[x].s = T[x].lgss = T[x].rgss = T[x].gss = p;
        return;
    }
    int mid = (l + r) >> 1;
    if (y <= mid)
        change(x << 1, l, mid, y, p);
    else
        change(x << 1 | 1, mid + 1, r, y, p);
    T[x] = merge(T[x << 1], T[x << 1 | 1]);
}
data query(int x, int l, int r, int L, int R) {
    if (L <= l && r <= R) return T[x];
    int mid = (l + r) >> 1;
    data lft, rht;
    if (L <= mid) lft = query(x << 1, l, mid, L, R);
    if (R > mid) rht = query(x << 1 | 1, mid + 1, r, L, R);
    if (R <= mid) return lft;
    if (L > mid) return rht;
    return merge(lft, rht);
}
int main() {
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        build(1, 1, n);
        scanf("%d", &m);
        while (m--) {
            int op, l, r;
            scanf("%d%d%d", &op, &l, &r);
            if (op == 0)
                change(1, 1, n, l, r);
            else
                printf("%d\n", query(1, 1, n, l, r).gss);
        }
    }
    return 0;
}
```

11.1.4 GSS4

```
/*
    题目大意：维护一个数列，支持区间开方和区间求和
    题解：我们记录区间最大值，暴力开方，因为开方到1之后无需再开方，
    因此当区间最大值为1时该区间不再递归处理
```

```
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
typedef long long LL;
const int N = 1000010, M = 2000010;
namespace Segment_Tree {
int tot;
struct node {
    int l, r, a, b;
    LL mx, val;
} T[M];
void build(int, int);
void Initialize(int n) {
    tot = 0;
    build(1, n);
}
void up(int x) {
    T[x].val = T[T[x].l].val + T[T[x].r].val;
    T[x].mx = max(T[T[x].l].mx, T[T[x].r].mx);
}
void build(int l, int r) {
    int x = ++tot;
    T[x].a = l;
    T[x].b = r;
    T[x].l = T[x].r = T[x].val = T[x].mx = 0;
    if (l == r) {
        scanf("%lld", &T[x].val);
        T[x].mx = T[x].val;
        return;
    }
    int mid = (l + r) >> 1;
    T[x].l = tot + 1;
    build(l, mid);
    T[x].r = tot + 1;
    build(mid + 1, r);
    up(x);
}
void change(int x, int a, int b) {
    if (T[x].mx == 1) return;
    if (T[x].a == T[x].b) {
        T[x].mx = T[x].val = sqrt(T[x].val);
        return;
    }
    int mid = (T[x].a + T[x].b) >> 1;
```

```
    if (mid >= a && T[x].l) change(T[x].l, a, b);
    if (mid < b && T[x].r) change(T[x].r, a, b);
    up(x);
}
LL query(int x, int a, int b) {
    if (T[x].a >= a && T[x].b <= b) return T[x].val;
    int mid = (T[x].a + T[x].b) >> 1;
    LL res = 0;
    if (mid >= a && T[x].l) res += query(T[x].l, a, b);
    if (mid < b && T[x].r) res += query(T[x].r, a, b);
    return res;
}
} // namespace Segment_Tree
int n, m, t, l, r, cnt = 0;
int main() {
    while (~scanf("%d", &n)) {
        Segment_Tree::Initialize(n);
        if (cnt) puts("");
        printf("Case #%d:\n", ++cnt);
        scanf("%d", &m);
        while (m--) {
            scanf("%d%d%d", &t, &l, &r);
            if (l > r) swap(l, r);
            if (t == 0)
                Segment_Tree::change(1, l, r);
            else
                printf("%lld\n", Segment_Tree::query(1, l, r));
        }
    }
    return 0;
}
```

11.1.5 GSS5

```
/*
    题目大意：给出一个数列，多次询问区间最大连续子段和，
    要求区间的左端点在x1,y1之间，右端点在x2,y2之间
    题解：线段树维护区间最大连续子段和gss，区间从最左元素开始的最大连续子段和lgss
    区间以最右元素为结尾的最大连续子段和rgss以及区间和s，信息传递并合并即可
    考虑询问区间的特殊性，如果两个端点区间不相交，
    则答案为[x1,y1].rgss+[y1+1,x2-1].s+[x2,y2].lgss
    如果区间相交，则中间区间[x2,y1]中必须有取到的部分，剩余两个区间可取可不取，
    但是必须与中间取到部分连接才能用于更新答案
*/
#include <algorithm>
```

```cpp
#include <cstdio>
using namespace std;
const int N = 100010, M = N << 2, INF = 0x3f3f3f3f;
struct data {
    int s, lgss, rgss, gss;
} T[M];
int a[N], s[N], n, m;
data merge(data L, data R) {
    data res;
    res.s = L.s + R.s;
    res.lgss = max(L.lgss, L.s + R.lgss);
    res.rgss = max(L.rgss + R.s, R.rgss);
    res.gss = max(max(L.gss, R.gss), L.rgss + R.lgss);
    return res;
}
void build(int x, int l, int r) {
    if (l == r) {
        T[x].s = T[x].lgss = T[x].rgss = T[x].gss = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(x << 1, l, mid);
    build(x << 1 | 1, mid + 1, r);
    T[x] = merge(T[x << 1], T[x << 1 | 1]);
}
data query(int x, int l, int r, int L, int R) {
    if (L <= l && r <= R) return T[x];
    int mid = (l + r) >> 1;
    data lft, rht;
    if (L <= mid) lft = query(x << 1, l, mid, L, R);
    if (R > mid) rht = query(x << 1 | 1, mid + 1, r, L, R);
    if (R <= mid) return lft;
    if (L > mid) return rht;
    return merge(lft, rht);
}
int Cas, ans;
data Q;
int main() {
    scanf("%d", &Cas);
    while (Cas--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]), s[i] = s[i - 1] + a[i];
        build(1, 1, n);
        scanf("%d", &m);
        while (m--) {
            int x1, x2, y1, y2;
```

```
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            if (y1 < x2) { // 区间不相交的情况
                ans = query(1, 1, n, x1, y1).rgss;
                ans += query(1, 1, n, x2, y2).lgss;
                printf("%d\n", ans + s[x2 - 1] - s[y1]);
            } else { // 区间相交
                ans = -INF;
                int L = 0, M = 0, MR = 0, R = 0;
                if (x2 > x1) {
                    Q = query(1, 1, n, x1, x2 - 1);
                    L = max(Q.rgss, L);
                }
                Q = query(1, 1, n, x2, y1);
                ans = max(L + Q.lgss, Q.gss);
                MR = Q.rgss;
                M = Q.s;
                if (y1 < y2) {
                    Q = query(1, 1, n, y1 + 1, y2);
                    R = max(R, Q.lgss);
                }
                ans = max(ans, L + M + R);
                ans = max(ans, MR + R);
                printf("%d\n", ans);
            }
        }
    }
    return 0;
}
```

11.1.6 GSS6

```
/*
    题目大意：维护一个数列，支持在x-1和x位置之间插入数字y，删除x位置上的数字，
    改变x位置上的数字以及询问区间最大子段和
    题解：splay维护区间最大连续子段和gss，区间从最左元素开始的最大连续子段和lgss
    区间以最右元素为结尾的最大连续子段rgss和以及区间和s
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 200010;
namespace Splay {
int a[N]; //原数列
int val[N], s[N], lgss[N], rgss[N], gss[N], size[N], son[N][2], f[N], tot, root;
```

```
int build(int, int, int);
void Initialize(int n) {
    tot = 0;
    root = build(0, n + 1, 0);
}
void up(int x) {
    int ls = son[x][0], rs = son[x][1];
    size[x] = size[ls] + size[rs] + 1;
    s[x] = s[ls] + s[rs] + val[x];
    lgss[x] = max(lgss[ls], s[ls] + val[x] + max(lgss[rs], 0));
    rgss[x] = max(rgss[rs], s[rs] + val[x] + max(rgss[ls], 0));
    gss[x] = val[x] + max(max(rgss[ls], lgss[rs]), max(0, lgss[rs] + rgss[ls]));
    gss[x] = max(gss[x], max(gss[ls], gss[rs]));
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y) son[z][0] = x;
        if (son[z][1] == y) son[z][1] = x;
    }
    f[x] = f[y];
    son[x][w ^ 1] = y;
    f[y] = x;
    up(y);
}
void splay(int x, int w) {
    int s = 1, i = x, y;
    a[1] = x;
    while (f[i]) a[++s] = i = f[i];
    while (f[x] != w) {
        y = f[x];
        if (f[y] != w) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    if (!w) root = x;
    up(x);
}
// root=build(0,n+1,0);
```

```
int build(int l, int r, int fa) {
    int x = ++tot, mid = (l + r) >> 1;
    f[x] = fa;
    val[x] = a[mid];
    if (l < mid) son[x][0] = build(l, mid - 1, x);
    if (r > mid) son[x][1] = build(mid + 1, r, x);
    up(x);
    return x;
}
int kth(int k) {
    int x = root, tmp;
    while (1) {
        tmp = size[son[x][0]] + 1;
        if (k == tmp) return x;
        if (k < tmp)
            x = son[x][0];
        else
            k -= tmp, x = son[x][1];
    }
}
// 删去第x个数字
void DELETE(int x) {
    int y = x;
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), son[y][0] = 0;
    up(y), up(x);
}
// 在第x-1个数字后面插入y
void INSERT(int x, int y) {
    x = kth(x);
    splay(x, 0);
    f[++tot] = x, val[tot] = y;
    son[tot][1] = son[x][1], f[son[x][1]] = tot, son[x][1] = tot;
    up(tot), up(x);
}
// 替换第x个数字
void REPLACE(int x, int y) {
    x = kth(x + 1);
    splay(x, 0);
    val[x] = y;
    up(x);
}
// 查询区间最大子段和
void QUERY(int x, int y) {
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x);
```

```
    printf("%d\n", gss[son[y][0]]);
}
} // namespace Splay
using namespace Splay;
int n, m;
int main() {
    scanf("%d", &n);
    // up里面数组太复杂了，没有判定子节点是否存在，所以要对0节点赋值-INF
    gss[0] = lgss[0] = rgss[0] = -INF;
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    Initialize(n);
    scanf("%d", &m);
    while (m--) {
        char op[10];
        int x, y;
        scanf("%s%d", op, &x);
        if (op[0] == 'D')
            DELETE(x);
        else
            scanf("%d", &y);
        if (op[0] == 'I') INSERT(x, y);
        if (op[0] == 'R') REPLACE(x, y);
        if (op[0] == 'Q') QUERY(x, y);
    }
    return 0;
}
```

11.1.7 GSS7

```
/*
    题目大意：维护一棵树，允许链上修改，询问链上最大连续子链和
    题解：我们用动态树维护操作，对于链上修改，我们提取链后打标记即可，
    考虑到标记修改值绝对值在10000以内，我们用0x3f来标记累加值，
    维护区间最长连续子段和gss，区间从最左元素开始的最大连续子段和lgss
    区间以最右元素为结尾的最大连续子段和rgss以及区间和s，
    当标记传递时，lgss和rgss以及gss将会变为0或者区间长度乘新值，s正常维护即可。
    需要注意的是，当发生区间翻转的时候，因为左右子树的交换，lgss和rgss的值也要发生交换，
*/
#include <algorithm>
#include <cstdio>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 200010;
namespace Link_Cut_Tree {
int f[N], son[N][2], tmp[N], size[N], tag[N], val[N];
```

```
bool rev[N];
int s[N], lgss[N], rgss[N], gss[N];
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void rev1(int x) {
    if (!x) return;
    swap(son[x][0], son[x][1]);
    // 子树旋转代表区间的左右最长连续子段和也发生交换
    swap(lgss[x], rgss[x]);
    rev[x] ^= 1;
}
void addtag(int x, int y) {
    if (!x) return;
    lgss[x] = rgss[x] = gss[x] = y > 0 ? y * size[x] : 0;
    s[x] = y * size[x];
    tag[x] = val[x] = y;
}
void pb(int x) {
    if (rev[x]) rev1(son[x][0]), rev1(son[x][1]), rev[x] = 0;
    if (tag[x] ^ INF)
        addtag(son[x][0], tag[x]), addtag(son[x][1], tag[x]), tag[x] = INF;
}
void up(int x) {
    int ls = son[x][0], rs = son[x][1];
    size[x] = size[ls] + size[rs] + 1;
    s[x] = s[ls] + s[rs] + val[x];
    lgss[x] = max(lgss[ls], s[ls] + val[x] + max(lgss[rs], 0));
    rgss[x] = max(rgss[rs], s[rs] + val[x] + max(rgss[ls], 0));
    gss[x] = val[x] + max(max(rgss[ls], lgss[rs]), max(0, lgss[rs] + rgss[ls]));
    gss[x] = max(gss[x], max(gss[ls], gss[rs]));
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
```

```
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void makeroot(int x) {
    access(x);
    splay(x);
    rev1(x);
}
void link(int x, int y) {
    makeroot(x);
    f[x] = y;
    access(x);
}
void split(int x, int y) {
    makeroot(y);
    access(x);
    splay(x);
}
} // namespace Link_Cut_Tree
int n, m, op, x, y, z;
using namespace Link_Cut_Tree;
int main() {
    while (~scanf("%d", &n)) {
```

```
        for (int i = 1; i <= n; i++) {
            scanf("%d", &val[i]);
            rev[i] = son[i][0] = son[i][1] = f[i] = 0;
            size[i] = 1;
            tag[i] = INF;
            lgss[i] = rgss[i] = gss[i] = val[i] > 0 ? val[i] : 0;
            s[i] = val[i];
        }
        for (int i = 1; i < n; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            link(x, y);
        }
        scanf("%d", &m);
        while (m--) {
            scanf("%d%d%d", &op, &x, &y);
            split(x, y);
            if (op == 1)
                printf("%d\n", gss[x]);
            else {
                scanf("%d", &z);
                rgss[x] = lgss[x] = gss[x] = z > 0 ? z * size[x] : 0;
                val[x] = tag[x] = z;
                s[x] = z * size[x];
            }
        }
    }
    return 0;
}
```

11.1.8 GSS8

```
/*
    题目大意：维护一个数列，支持数字的插入，删除和替换，
    以及查询[l,r]中a[i]*(i-l+1)^k的和，maxk=10，答案对2^32取模
    题解：利用splay可以维护数字的插入和删改，接下来考虑如何在splay上维护a[i]*(i-l+1)^k
    设d[x][i]表示子树在k=i时候的答案，我们发现对于左子树来说，
    d[ls][i]是直接累加到d[x][i]上的，同时d[x][i]还要加上(pos-l+1)^k*a[x]，
    我们发现pos-l恰好是左子树的大小，所以子树根贡献为(size[ls]+1)^k*a[x]
    考虑右子树的贡献，对于单个节点x来说，答案是(size[ls[root]]+1+size[ls]+1)^k*a[x]
    对size[ls[root]]+1和size[ls]+1进行二项式拆分，
    发现右子树每个节点拆分后的项可以合并，合并项包括k=1~i的d数组，
    其对根节点x的值d[x][i]的贡献为d[rs][j]*C[i][j]*(size[ls]+1)^(k-j)
    对d数组进行维护即可，考虑到取模数字的特殊性，直接用uint自然溢出
*/
```

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 200010;
const int K = 10;
typedef unsigned int uint;
uint C[20][20], d[N][K + 5];
namespace Splay {
int a[N]; //原数列
int val[N], size[N], son[N][2], f[N], tot, root;
int build(int, int, int);
void Initialize(int n) {
    tot = 0;
    root = build(0, n + 1, 0);
}
void up(int x) {
    int ls = son[x][0], rs = son[x][1];
    size[x] = size[ls] + size[rs] + 1;
    int sz = size[ls] + 1;
    uint sum = 1;
    for (int i = 0; i <= K; i++) {
        d[x][i] = d[ls][i] + val[x] * sum;
        sum = sum * sz;
    }
    for (int i = 0; i <= K; i++) {
        uint sum = 1;
        for (int j = i; j >= 0; sum = sum * sz, j--) {
            d[x][i] += d[rs][j] * sum * C[i][j];
        }
    }
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y) son[z][0] = x;
        if (son[z][1] == y) son[z][1] = x;
    }
    f[x] = f[y];
    son[x][w ^ 1] = y;
    f[y] = x;
    up(y);
}
```

```
void splay(int x, int w) {
    int s = 1, i = x, y;
    a[1] = x;
    while (f[i]) a[++s] = i = f[i];
    while (f[x] != w) {
        y = f[x];
        if (f[y] != w) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    if (!w) root = x;
    up(x);
}
// root=build(0,n+1,0);
int build(int l, int r, int fa) {
    int x = ++tot, mid = (l + r) >> 1;
    f[x] = fa;
    val[x] = a[mid];
    for (int i = 0; i <= K; i++) d[x][i] = a[mid];
    if (l < mid) son[x][0] = build(l, mid - 1, x);
    if (r > mid) son[x][1] = build(mid + 1, r, x);
    up(x);
    return x;
}
int kth(int k) {
    int x = root, tmp;
    while (1) {
        tmp = size[son[x][0]] + 1;
        if (k == tmp) return x;
        if (k < tmp)
            x = son[x][0];
        else
            k -= tmp, x = son[x][1];
    }
}
// 删去第x个数字
void DELETE(int x) {
    int y = x;
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x), son[y][0] = 0;
    up(y), up(x);
}
```

```
// 在第x-1个数字后面插入y
void INSERT(int x, int y) {
    x = kth(x);
    splay(x, 0);
    f[++tot] = x, val[tot] = y;
    for (int i = 0; i <= K; i++) d[tot][i] = y;
    son[tot][1] = son[x][1], f[son[x][1]] = tot, son[x][1] = tot;
    up(tot), up(x);
}
// 替换第x个数字
void REPLACE(int x, int y) {
    x = kth(x + 1);
    splay(x, 0);
    val[x] = y;
    for (int i = 0; i <= K; i++) d[x][i] = y;
    up(x);
}
// 查询[l,r]中a[i]*(i-l+1)^k的和
void QUERY(int x, int y, int z) {
    x = kth(x), y = kth(y + 2);
    splay(x, 0), splay(y, x);
    printf("%u\n", d[son[y][0]][z]);
}
} // namespace Splay
using namespace Splay;
void initC() {
    C[0][0] = 1;
    for (int i = 1; i <= K; i++) {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
    }
}
int n, m;
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    initC();
    Initialize(n);
    scanf("%d", &m);
    while (m--) {
        char op[10];
        int x, y, z;
        scanf("%s%d", op, &x);
        x++;
        if (op[0] == 'D')
            DELETE(x);
```

```
        else
            scanf("%d", &y);
        if (op[0] == 'I') INSERT(x, y);
        if (op[0] == 'R') REPLACE(x, y);
        if (op[0] == 'Q') {
            y++;
            scanf("%d", &z);
            QUERY(x, y, z);
        }
    }
    return 0;
}
```

11.2 SPOJ-QTREE

11.2.1 QTREE1

```
/*
    题目大意：维护一棵树，允许修改边权以及查询链上最大值
    题解：我们将边权转为点权，标记在深度较深的点上，树链剖分后用线段树处理即可
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 20010;
int top[N], nxt[N], v[N], g[N], d[N], son[N], f[N], size[N], st[N], en[N], dfn,
    ed;
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void init() {
    memset(g, dfn = ed = 0, sizeof(g));
    memset(v, 0, sizeof(v));
    memset(nxt, 0, sizeof(nxt));
    memset(son, -1, sizeof(son));
}
void dfs(int x) {
    size[x] = 1;
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != f[x]) {
            f[v[i]] = x, d[v[i]] = d[x] + 1;
            dfs(v[i]), size[x] += size[v[i]];
```

```
            if (size[v[i]] > size[son[x]]) son[x] = v[i];
        }
}
void dfs2(int x, int y) {
    st[x] = ++dfn;
    top[x] = y;
    if (son[x]) dfs2(son[x], y);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != son[x] && v[i] != f[x]) dfs2(v[i], v[i]);
    en[x] = dfn;
}
struct Node {
    int l, r;
    int Max;
} Tree[N * 3];
void build(int x, int l, int r) {
    Tree[x].l = l;
    Tree[x].r = r;
    Tree[x].Max = 0;
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(x << 1, l, mid);
    build((x << 1) | 1, mid + 1, r);
}
void up(int x) { Tree[x].Max = max(Tree[x << 1].Max, Tree[(x << 1) | 1].Max); }
void update(int x, int k, int val) {
    if (Tree[x].l == k && Tree[x].r == k) {
        Tree[x].Max = val;
        return;
    }
    int mid = (Tree[x].l + Tree[x].r) >> 1;
    if (k <= mid)
        update(x << 1, k, val);
    else
        update((x << 1) | 1, k, val);
    up(x);
}
int query(int x, int l, int r) {
    if (Tree[x].l == l && Tree[x].r == r) return Tree[x].Max;
    int mid = (Tree[x].l + Tree[x].r) >> 1;
    if (r <= mid)
        return query(x << 1, l, r);
    else if (l > mid)
        return query((x << 1) | 1, l, r);
    else
        return max(query(x << 1, l, mid), query((x << 1) | 1, mid + 1, r));
```

```
}
int find(int x, int y) {
    int tmp = 0;
    for (; top[x] != top[y]; x = f[top[x]]) {
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
        tmp = max(tmp, query(1, st[top[x]], st[x]));
    }
    if (x == y) return tmp;
    if (d[x] > d[y]) {
        int z = x;
        x = y;
        y = z;
    }
    return max(tmp, query(1, st[son[x]], st[y]));
}
int e[N][3];
int main(int n, int T) {
    scanf("%d", &T);
    while (T--) {
        init();
        scanf("%d", &n);
        for (int i = 0; i < n - 1; i++) {
            scanf("%d%d%d", &e[i][0], &e[i][1], &e[i][2]);
            add(e[i][0], e[i][1]);
            add(e[i][1], e[i][0]);
        }
        dfs(1);
        dfs2(1, 1);
        build(1, 1, dfn);
        for (int i = 0; i < n - 1; i++) {
            if (d[e[i][0]] > d[e[i][1]]) swap(e[i][0], e[i][1]);
            update(1, st[e[i][1]], e[i][2]);
        }
        char op[10];
        int u, w;
        while (scanf("%s", op) == 1) {
            if (op[0] == 'D') break;
            scanf("%d%d", &u, &w);
            if (op[0] == 'Q')
                printf("%d\n", find(u, w));
            else
                update(1, st[e[u - 1][1]], w);
```

```
        }
    }
    return 0;
}
```

11.2.2 QTREE2

```
/*
    题目大意：给出一棵边权树，要求实现树链求和以及求链上第k个元素
    题解：边权转点权后用LCT维护，注意求Qkth和sum时区别于点权树的讨论
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const int N = 300010;
namespace Link_Cut_Tree {
vector<int> v[N], w[N];
int f[N], son[N][2], tmp[N], size[N], val[N], sum[N];
void init() {
    memset(val, 0, sizeof(val));
    memset(sum, 0, sizeof(sum));
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    for (int i = 1; i < N; i++) size[i] = 1, v[i].clear(), w[i].clear();
}
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void up(int x) {
    sum[x] = val[x] + sum[son[x][0]] + sum[son[x][1]];
    size[x] = size[son[x][0]] + size[son[x][1]] + 1;
}
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
```

```
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
// 求边权树的链和
int ask(int x, int y) {
    access(y), y = 0;
    int ans = 0;
    while (x) {
        splay(x);
        if (!f[x]) return sum[son[x][1]] + sum[y];
        son[x][1] = y;
        up(x);
        x = f[y = x];
    }
}
void dfs(int x, int fx) {
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i];
        if (y == fx) continue;
        f[y] = x;
        val[y] = w[x][i];
        dfs(y, x);
    }
    up(x);
}
int kth(int x, int k) {
    while (size[son[x][0]] + 1 != k) {
```

```
        if (k < size[son[x][0]] + 1)
            x = son[x][0];
        else {
            k -= size[son[x][0]] + 1;
            x = son[x][1];
        }
    }
    return x;
}
// 求边权树u->v的第k个点
int Qkth(int u, int v, int k) {
    access(u), u = 0;
    while (v) {
        splay(v);
        if (!f[v]) {
            if (size[son[v][1]] + 1 == k)
                return v;
            else if (size[son[v][1]] + 1 > k)
                return kth(son[v][1], size[son[v][1]] + 1 - k);
            else
                return kth(u, k - (size[son[v][1]] + 1));
        }
        son[v][1] = u;
        up(v);
        v = f[u = v];
    }
}
} // namespace Link_Cut_Tree
using namespace Link_Cut_Tree;
int T, x, y, z, k, n;
char op[10];
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        init();
        for (int i = 1; i < n; i++) {
            scanf("%d%d%d", &x, &y, &z);
            v[x].push_back(y);
            w[x].push_back(z);
            v[y].push_back(x);
            w[y].push_back(z);
        }
        dfs(1, 0);
        while (scanf("%s", op)) {
            if (op[1] == 'O') break;
```

```
            if (op[0] == 'D') {
                scanf("%d%d", &x, &y);
                printf("%d\n", ask(x, y));
            } else {
                scanf("%d%d%d", &x, &y, &k);
                printf("%d\n", Qkth(x, y, k));
            }
        }
    }
    return 0;
}
```

11.2.3 QTREE3

```
/*
    题目大意：给出一棵一开始都是白点的树，要求维护操作：
        1.将某个点的颜色取反(黑白互换)
        2.查询1到x的路径上第一个黑色的点
    题解：LCT维护子树中黑点的数量，对于颜色取反，则直接flap，
    对于查询操作通过access(x)将1到x这条链取出，splay上查找即可
    注意是从1到x，所以是makeroot(1)，保证1节点的键值最小
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100010;
namespace Link_Cut_Tree {
int f[N], son[N][2], val[N], sum[N], tmp[N];
bool rev[N];
void init() {
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    memset(val, 0, sizeof(val));
    memset(rev, 0, sizeof(rev));
    memset(sum, 0, sizeof(sum));
}
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void rev1(int x) {
    if (!x) return;
    swap(son[x][0], son[x][1]);
    rev[x] ^= 1;
}
void pb(int x) {
    if (rev[x]) rev1(son[x][0]), rev1(son[x][1]), rev[x] = 0;
```

```
}
void up(int x) { sum[x] = sum[son[x][0]] + sum[son[x][1]] + val[x]; }
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (s) pb(tmp[s--]);
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void makeroot(int x) {
    access(x);
```

```
    splay(x);
    rev1(x);
}
void link(int x, int y) {
    makeroot(x);
    f[x] = y;
    access(x);
}
int flip(int x) {
    makeroot(x);
    val[x] ^= 1;
    up(x);
}
// 查询从1->x的第一个黑点
int ask(int x) {
    makeroot(1);
    access(x);
    splay(x); // splay只保证了x到根节点标记的下传
    if (!sum[x]) return -1;
    while (x) {
        pb(x); // 因为有makeroot操作，所以需要下传标记
        if (!sum[son[x][0]] && val[x] == 1)
            return x;
        else if (sum[son[x][0]])
            x = son[x][0];
        else
            x = son[x][1];
    }
}
} // namespace Link_Cut_Tree
using namespace Link_Cut_Tree;
int n, q, x, y;
int main() {
    while (~scanf("%d%d", &n, &q)) {
        init();
        for (int i = 1; i < n; i++) {
            scanf("%d%d", &x, &y);
            link(x, y);
        }
        while (q--) {
            scanf("%d%d", &x, &y);
            if (x)
                printf("%d\n", ask(y));
            else
                flip(y);
        }
```

```
    }
    return 0;
}
```

11.2.4 QTREE3+

```
/*
    题目大意：给出一棵一开始都是白点的树，要求维护操作：
        1.将某个点的颜色取反(黑白互换)
        2.查询1到x的路径上第一个黑色的点
    题解：LCT维护子树中黑点的数量，对于颜色取反，则直接flap，
    对于查询操作通过access(x)将1到x这条链取出，splay上查找即可
    考虑根为1不会变动，因此直接dfs建树，效率更高
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const int N = 100010;
namespace Link_Cut_Tree {
int f[N], son[N][2], val[N], sum[N], tmp[N];
vector<int> v[N];
void init() {
    memset(f, 0, sizeof(f));
    memset(son, 0, sizeof(son));
    memset(val, 0, sizeof(val));
    memset(sum, 0, sizeof(sum));
}
bool isroot(int x) { return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x; }
void up(int x) { sum[x] = sum[son[x][0]] + sum[son[x][1]] + val[x]; }
void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y)
            son[z][0] = x;
        else if (son[z][1] == y)
            son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
```

```
    up(y);
}
void splay(int x) {
    int s = 1, i = x, y;
    tmp[1] = i;
    while (!isroot(i)) tmp[++s] = i = f[i];
    while (!isroot(x)) {
        y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) splay(x), son[x][1] = y, up(x);
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void dfs(int x, int fx) {
    for (int i = 0; i < v[x].size(); i++) {
        int y = v[x][i];
        if (y == fx) continue;
        f[y] = x;
        dfs(y, x);
    }
    up(x);
}
int flip(int x) {
    splay(x);
    val[x] ^= 1;
    up(x);
}
int ask(int x) {
    access(x);
    splay(x);
    if (!sum[x]) return -1;
    while (x) {
```

```
        if (!sum[son[x][0]] && val[x] == 1)
            return x;
        else if (sum[son[x][0]])
            x = son[x][0];
        else
            x = son[x][1];
    }
}
} // namespace Link_Cut_Tree
using namespace Link_Cut_Tree;
int n, q, x, y;
int main() {
    while (~scanf("%d%d", &n, &q)) {
        init();
        for (int i = 1; i <= n; i++) v[i].clear();
        for (int i = 1; i < n; i++) {
            scanf("%d%d", &x, &y);
            v[x].push_back(y);
            v[y].push_back(x);
        }
        dfs(1, 0);
        while (q--) {
            scanf("%d%d", &x, &y);
            if (x)
                printf("%d\n", ask(y));
            else
                flip(y);
        }
    }
    return 0;
}
```

11.2.5 QTREE4

```
/*
    题目大意：给出一棵边权树，一开始所有点都是白点，
    要求维护两种操作：
        1.颜色取反（黑白互变）
        2.查询树上最远的两个白点之间的距离
    题解：我们维护重心树，对于每个重心维护分治层所有白点到其距离，
    将其保存在对应方向子重心的优先队列中，
    重心处另外维护一个队列保存该分治层每个子重心所产生的最大答案，即最大值加次大值，
    那么每个重心处产生的最大答案的最值就是答案，
    考虑修改问题，等价于在处理优先队列的删除问题，
    对于每个需要删除操作的优先队列，我们额外创建一个优先队列将删除元素加入其中，
```

```
    当两个队列top元素相同时同时删去即可。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int N = 200010;
const int INF = 0x3f3f3f3f;
int tot, g[N], nxt[N << 1], v[N << 1], w[N << 1];
int TOT, G[N], NXT[N * 20], V[N * 20], W[N * 20];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
    memset(G, -1, sizeof(G));
    TOT = 0;
}
void add_edge(int x, int y, int z) {
    v[tot] = y, w[tot] = z, nxt[tot] = g[x], g[x] = tot++;
}
void ADD_EDGE(int x, int y, int z) {
    V[TOT] = y, W[TOT] = z, NXT[TOT] = G[x], G[x] = TOT++;
}
int sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
struct Que {
    priority_queue<int> Q, D;
    void clear() {
        while (!Q.empty()) Q.pop();
        while (!D.empty()) D.pop();
    }
    void add(int x) { Q.push(x); }
    void del(int x) { D.push(x); }
    int top() {
```

```
        for (;;) {
            if (Q.empty())
                return -INF;
            else if (!D.empty() && Q.top() == D.top())
                Q.pop(), D.pop();
            else
                return Q.top();
        }
    }
    int toptwo() {
        int x = top();
        del(x);
        int y = top();
        add(x);
        if (y == -INF) return x == -INF ? x : 0;
        return max(x + y, 0);
    }
} P[N], Q[N], ans;
int dis[N], col[N];
void getdis(int rt, int x, int fx) {
    Q[rt].add(dis[x]);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx || vis[v[i]]) continue;
        dis[v[i]] = dis[x] + w[i];
        getdis(rt, v[i], x);
    }
}
void getdeep(int rt, int x, int fx) {
    ADD_EDGE(x, rt, dis[x]);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx || vis[v[i]]) continue;
        dis[v[i]] = dis[x] + w[i];
        getdeep(rt, v[i], x);
    }
}
int belong[N];
void build(int x, int fx, int from, int d) {
    belong[x] = fx;
    dis[x] = 0;
    getdeep(x, x, fx);
    P[x].add(0);
    if (fx) {
        dis[from] = d;
        getdis(x, from, fx);
        P[fx].add(Q[x].top());
    }
```

```
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) {
            root = 0;
            sum = size[v[i]];
            getroot(v[i], x);
            build(root, x, v[i], w[i]);
        }
    ans.add(P[x].toptwo());
}
void change(int rt) {
    int a = P[rt].toptwo();
    if (col[rt])
        P[rt].add(0);
    else
        P[rt].del(0);
    int b = P[rt].toptwo();
    if (a != b) ans.del(a), ans.add(b);
    int x = rt;
    for (int i = G[rt]; ~NXT[i]; i = NXT[i]) {
        int y = V[NXT[i]], len = W[NXT[i]];
        x = V[i];
        a = Q[x].top();
        if (col[rt])
            Q[x].add(len);
        else
            Q[x].del(len);
        b = Q[x].top();
        if (a != b) {
            int c = P[y].toptwo();
            P[y].del(a), P[y].add(b);
            int d = P[y].toptwo();
            if (c != d) ans.del(c), ans.add(d);
        }
    }
}
int n, m, a, b, c, x;
char op[10];
int main() {
    scanf("%d", &n);
    init();
    for (int i = 1; i <= n; i++) col[i] = 1;
    for (int i = 1; i < n; i++) {
        scanf("%d%d%d", &a, &b, &c);
        add_edge(a, b, c);
        add_edge(b, a, c);
```

```
    }
    memset(vis, 0, sizeof(vis));
    dp[root = 0] = sum = n;
    getroot(1, 0);
    build(root, 0, 0, 0);
    scanf("%d", &m);
    int cnt = n;
    while (m--) {
        scanf("%s", op);
        if (op[0] == 'A') {
            if (cnt)
                printf("%d\n", ans.top());
            else
                puts("They have disappeared.");
        } else {
            scanf("%d", &x);
            col[x] ^= 1;
            if (col[x])
                cnt++;
            else
                cnt--;
            change(x);
        }
    }
    return 0;
}
```

11.2.6 QTREE5

```
/*
    题目大意：给出一棵树，边权均为1，一开始所有点都是白点，
    要求维护两种操作：
        1.颜色取反（黑白互变）
        2.查询树上某点距离最近的白点与其距离
    题解：我们维护重心树，对于每个重心维护分治层所有白点到其距离，
    将其保存在对应方向子重心的优先队列中，
    重心处另外维护一个队列表示每个子重心的最小答案，
    考虑修改问题，等价于在处理优先队列的删除问题，
    对于每个需要删除操作的优先队列，我们额外创建一个优先队列将删除元素加入其中，
    当两个队列top元素相同时同时删去即可。
    对于查询操作，我们沿着重心链用经过每个重心的最优值更新答案
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
```

```
#include <queue>
using namespace std;
const int N = 200010;
const int INF = 0x3f3f3f3f;
int tot, g[N], nxt[N << 1], v[N << 1], w[N << 1];
int TOT, G[N], NXT[N * 20], V[N * 20], W[N * 20];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
    memset(G, -1, sizeof(G));
    TOT = 0;
}
void add_edge(int x, int y, int z) {
    v[tot] = y, w[tot] = z, nxt[tot] = g[x], g[x] = tot++;
}
void ADD_EDGE(int x, int y, int z) {
    V[TOT] = y, W[TOT] = z, NXT[TOT] = G[x], G[x] = TOT++;
}
int sum, root, size[N], dp[N], vis[N];
void getroot(int x, int fx) {
    size[x] = 1;
    dp[x] = 0;
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (!vis[v[i]] && v[i] != fx) {
            getroot(v[i], x);
            size[x] += size[v[i]];
            dp[x] = max(dp[x], size[v[i]]);
        }
    }
    dp[x] = max(dp[x], sum - size[x]);
    if (dp[x] < dp[root]) root = x;
}
struct Que {
    priority_queue<int, vector<int>, greater<int> > Q, D;
    void clear() {
        while (!Q.empty()) Q.pop();
        while (!D.empty()) D.pop();
    }
    void add(int x) { Q.push(x); }
    void del(int x) { D.push(x); }
    int top() {
        for (;;) {
            if (Q.empty())
                return INF;
            else if (!D.empty() && Q.top() == D.top())
                Q.pop(), D.pop();
```

```
            else
                return Q.top();
        }
    }
} P[N], Q[N];
int dis[N], col[N];
void getdis(int rt, int x, int fx) {
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx || vis[v[i]]) continue;
        dis[v[i]] = dis[x] + w[i];
        getdis(rt, v[i], x);
    }
}
void getdeep(int rt, int x, int fx) {
    ADD_EDGE(x, rt, dis[x]);
    for (int i = g[x]; ~i; i = nxt[i]) {
        if (v[i] == fx || vis[v[i]]) continue;
        dis[v[i]] = dis[x] + w[i];
        getdeep(rt, v[i], x);
    }
}
int belong[N];
void build(int x, int fx, int from, int d) {
    belong[x] = fx;
    dis[x] = 0;
    getdeep(x, x, fx);
    if (fx) {
        dis[from] = d;
        getdis(x, from, fx);
    }
    vis[x] = 1;
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) {
            root = 0;
            sum = size[v[i]];
            getroot(v[i], x);
            build(root, x, v[i], w[i]);
        }
}
void change(int rt) {
    if (col[rt])
        P[rt].add(0);
    else
        P[rt].del(0);
    int x = rt;
    for (int i = G[rt]; ~NXT[i]; i = NXT[i]) {
```

```
        int y = V[NXT[i]], len = W[NXT[i]];
        x = V[i];
        int a = Q[x].top();
        if (col[rt])
            Q[x].add(len);
        else
            Q[x].del(len);
        int b = Q[x].top();
        if (a != b) P[y].del(a), P[y].add(b);
    }
}
int ask(int rt) {
    int ans = INF;
    for (int i = G[rt]; ~i; i = NXT[i]) {
        int x = V[i], len = W[i];
        int a = P[x].top();
        if (a + len < ans) ans = a + len;
    }
    return ans;
}
int n, m, a, b, c, x, op;
int main() {
    scanf("%d", &n);
    init();
    for (int i = 1; i <= n; i++) col[i] = 0;
    for (int i = 1; i < n; i++) {
        scanf("%d%d", &a, &b);
        add_edge(a, b, 1);
        add_edge(b, a, 1);
    }
    memset(vis, 0, sizeof(vis));
    dp[root = 0] = sum = n;
    getroot(1, 0);
    build(root, 0, 0, 0);
    scanf("%d", &m);
    int cnt = 0;
    while (m--) {
        scanf("%d%d", &op, &x);
        if (op == 1) {
            if (cnt)
                printf("%d\n", ask(x));
            else
                puts("-1");
        } else {
            col[x] ^= 1;
            if (col[x])
```

```
            cnt++;
        else
            cnt--;
        change(x);
    }
  }
  return 0;
}
```

11.2.7 QTREE6

```
/*
  题目大意：给出一棵树，一开始树上点均为黑色，要求维护操作
    1.翻转某点的颜色(黑白互换)，
    2.查询某个点所在连通块(颜色相同则连通)大小
  题解：我们分别维护黑点LCT和白点LCT，当一个点从黑变白时，
  将其从黑点LCT中与父节点断开，然后在白点LCT中与父节点相连，
  这样我们就保证了每个连通块在LCT中只有父节点是不同色的而其余一定是连通的。
  考虑用LCT维护子树信息，根据在LCT上的连接情况，我们将点的儿子分为实儿子和虚儿子
  实儿子是原本树结构上相连的点，实儿子的子树为实子树，虚儿子的子树为虚子树
  x的LCT子树的信息和等于x的实儿子的LCT子树信息和加上x的虚子树的信息和加上x自己
  在进行access操作时，我们会有更换一个点的x右儿子的操作，
  这时我们要把x原来的右儿子的LCT子树信息加入x的虚子树信息，
  把x的新的右儿子的LCT子树信息从x的虚子树信息中减去
  同时在link x到y的时候，我们需要对y进行access再splay
  这样就只会对y的虚子树信息和LCT子树信息产生影响，而不会影响到y的祖先节点
*/
#include <algorithm>
#include <cstdio>
#include <set>
const int N = 100010;
using namespace std;
int n, m, i, k, x, y, c[N], fa[N], g[N], v[N << 1], nxt[N << 1], ed;
struct LCT {
  int son[N][2], f[N], sum[N], s[N];
  bool isroot(int x) {
    return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x;
  }
  void up(int x) { sum[x] = 1 + sum[son[x][0]] + sum[son[x][1]] + s[x]; }
  void rotate(int x) {
    int y = f[x], w = son[y][1] == x;
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
      int z = f[y];
```

```
        if (son[z][0] == y) son[z][0] = x;
        if (son[z][1] == y) son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    while (!isroot(x)) {
        int y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) {
        splay(x);
        if (son[x][1]) s[x] += sum[son[x][1]];
        if (son[x][1] = y) s[x] -= sum[y];
        up(x);
    }
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void link(int x) {
    access(fa[x]);
    splay(fa[x]);
    splay(x);
    son[fa[x]][1] = x;
    up(f[x] = fa[x]);
}
void cut(int x) {
    access(x);
    splay(x);
    f[son[x][0]] = 0;
```

```
        son[x][0] = 0;
        up(x);
    }
    int ask(int x) {
        splay(x = root(x));
        return sum[son[x][1]];
    }
} T[2];
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != fa[x]) {
            fa[v[i]] = T[c[v[i]]].f[v[i]] = x;
            dfs(v[i]);
            T[c[v[i]]].s[x] += T[c[v[i]]].sum[v[i]];
        }
    T[0].up(x), T[1].up(x);
}
void read(int& a) {
    char c;
    while (!(((c = getchar()) >= '0') && (c <= '9')))
        ;
    a = c - '0';
    while (((c = getchar()) >= '0') && (c <= '9')) (a *= 10) += c - '0';
}
int main() {
    read(n);
    for (i = 1; i < n; i++) read(x), read(y), add(x, y), add(y, x);
    for (i = 1; i <= n; i++) c[i] = 0;
    fa[1] = T[c[1]].f[1] = n + 1;
    dfs(1);
    T[c[1]].s[n + 1] += T[c[1]].sum[1];
    read(m);
    while (m--) {
        read(k);
        read(x);
        if (!k)
            printf("%d\n", T[c[x]].ask(x));
        else
            T[c[x]].cut(x), T[c[x] ^= 1].link(x);
    }
    return 0;
```

```
}
```

11.2.8 QTREE7

```
/*
    题目大意：给出一棵树，树上每个点为黑色或者白色，每个点有个点权，
    我们认为树上颜色相同并通过边连接的块为连通块
    要求维护操作：
        0.查询x所在连通块中权值最大的点
        1.翻转某点的颜色(黑白互换)
        2.改变某点的权值
    题解：我们分别维护黑点LCT和白点LCT，当一个点从黑变白时，
    将其从黑点LCT中与父节点断开，然后在白点LCT中与父节点相连，
    这样我们就保证了每个连通块在LCT中只有父节点是不同色的而其余一定是连通的。
    考虑用LCT维护子树信息，根据在LCT上的连接情况，我们将点的儿子分为实儿子和虚儿子
    实儿子是原本树结构上相连的点，实儿子的子树为实子树，虚儿子的子树为虚子树
    x的LCT子树的信息和等于x的实儿子的LCT子树信息和加上x的虚子树的信息和加上x自己
    在进行access操作时，我们会有更换一个点的x右儿子的操作，
    这时我们要把x原来的右儿子的LCT子树信息加入x的虚子树信息，
    把x的新的右儿子的LCT子树信息从x的虚子树信息中减去
    同时在link x到y的时候，我们需要对y进行access再splay
    这样就只会对y的虚子树信息和LCT子树信息产生影响，而不会影响到y的祖先节点
    因为要维护的信息是极值，因此我们要用multiset来维护虚子树信息
*/
#include <ctype.h>
#include <algorithm>
#include <cstdio>
#include <set>
const int N = 100010;
using namespace std;
int n, m, k, x, y, c[N], fa[N], g[N], v[N << 1], nxt[N << 1], ed;
struct LCT {
    int son[N][2], f[N], val[N], mx[N];
    multiset<int> s[N];
    bool isroot(int x) {
        return !f[x] || son[f[x]][0] != x && son[f[x]][1] != x;
    }
    void up(int x) {
        mx[x] = val[x];
        if (s[x].size()) mx[x] = max(mx[x], *s[x].rbegin());
        if (son[x][0]) mx[x] = max(mx[x], mx[son[x][0]]);
        if (son[x][1]) mx[x] = max(mx[x], mx[son[x][1]]);
    }
    void rotate(int x) {
        int y = f[x], w = son[y][1] == x;
```

```
    son[y][w] = son[x][w ^ 1];
    if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
    if (f[y]) {
        int z = f[y];
        if (son[z][0] == y) son[z][0] = x;
        if (son[z][1] == y) son[z][1] = x;
    }
    f[x] = f[y];
    f[y] = x;
    son[x][w ^ 1] = y;
    up(y);
}
void splay(int x) {
    while (!isroot(x)) {
        int y = f[x];
        if (!isroot(y)) {
            if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                rotate(x);
            else
                rotate(y);
        }
        rotate(x);
    }
    up(x);
}
void access(int x) {
    for (int y = 0; x; y = x, x = f[x]) {
        splay(x);
        if (son[x][1]) s[x].insert(mx[son[x][1]]);
        if (son[x][1] = y) s[x].erase(s[x].find(mx[y]));
        up(x);
    }
}
int root(int x) {
    access(x);
    splay(x);
    while (son[x][0]) x = son[x][0];
    return x;
}
void link(int x) {
    access(fa[x]);
    splay(fa[x]);
    splay(x);
    son[fa[x]][1] = x;
    up(f[x] = fa[x]);
}
```

```
    void cut(int x) {
        access(x);
        splay(x);
        f[son[x][0]] = 0;
        son[x][0] = 0;
        up(x);
    }
    int ask(int x) {
        int t = c[x];
        splay(x = root(x));
        return t == c[x] ? mx[x] : mx[son[x][1]];
    }
    void modify(int x, int v) { access(x), splay(x), val[x] = v, up(x); }
} T[2];
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != fa[x]) {
            fa[v[i]] = T[c[v[i]]].f[v[i]] = x;
            dfs(v[i]);
            T[c[v[i]]].s[x].insert(T[c[v[i]]].mx[v[i]]);
        }
    T[0].up(x), T[1].up(x);
}
void read(int &x) {
    char ch;
    for (ch = getchar(); !isdigit(ch) && ch != '-'; ch = getchar())
        ;
    int t;
    if (ch == '-')
        t = -1, ch = getchar();
    else
        t = 1;
    for (x = 0; isdigit(ch); ch = getchar()) x = x * 10 + ch - '0';
    x = x * t;
}
int main() {
    read(n);
    for (int i = 1; i < n; i++) read(x), read(y), add(x, y), add(y, x);
    for (int i = 1; i <= n; i++) read(c[i]);
    for (int i = 1; i <= n; i++) read(T[0].val[i]), T[1].val[i] = T[0].val[i];
    dfs(1);
```

```
    read(m);
    while (m--) {
        read(k);
        read(x);
        if (k == 0)
            printf("%d\n", T[c[x]].ask(x));
        else if (k == 1)
            T[c[x]].cut(x), T[c[x] ^= 1].link(x);
        else if (k == 2)
            read(y), T[0].modify(x, y), T[1].modify(x, y);
    }
    return 0;
}
```

11.3 SPOJ-COT

11.3.1 COT

```
/*
    题目大意：静态链上第k大
    题解：随意确立一个根，对每个节点建立一棵权值线段树保存从根节点到当前节点的权值集合，
    查询链x-y时用T[l[x]]+T[l[y]]-T[l[lca]]-T[l[flca]]做判断在树上二分即可，
    每次建树从父节点加链创建，复杂度logn，
    由于权值范围是int，所以需要权值映射。
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int N = 100010;
int l[N * 40], r[N * 40], T[N * 40], tot, root[N];
int change(int x, int a, int b, int c, int p) {
    int y = ++tot;
    T[y] = T[x] + p;
    if (a == b) return y;
    int mid = (a + b) >> 1;
    if (c <= mid)
        l[y] = change(l[x], a, mid, c, p), r[y] = r[x];
    else
        l[y] = l[x], r[y] = change(r[x], mid + 1, b, c, p);
    return y;
}
int query(int x, int y, int a, int b, int lca, int flca, int k) {
    if (a == b) return a;
    int mid = (a + b) >> 1;
```

```
    int cnt = T[l[x]] + T[l[y]] - T[l[lca]] - T[l[flca]];
    if (k <= cnt)
        return query(l[x], l[y], a, mid, l[lca], l[flca], k);
    else
        return query(r[x], r[y], mid + 1, b, r[lca], r[flca], k - cnt);
}
void read(int& x) {
    int f = 1;
    x = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if ('-' == ch) f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + ch - '0';
        ch = getchar();
    }
}
int siz;       // 离散化之后的大小
int w[N], b[N]; // 原值和映射
int ed, g[N << 1], v[N << 1], nxt[N << 1];
int f[N], size[N], son[N], dfn, top[N], d[N];
void add_edge(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    size[x] = 1;
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != f[x]) {
            f[v[i]] = x, d[v[i]] = d[x] + 1;
            dfs(v[i]), size[x] += size[v[i]];
            if (size[v[i]] > size[son[x]]) son[x] = v[i];
        }
}
void dfs2(int x, int y) {
    if (x == -1) return;
    top[x] = y;
    root[x] =
        change(root[f[x]], 1, siz, w[x], 1); // dfs的同时对每个节点建立线段树
    if (son[x]) dfs2(son[x], y);
    for (int i = g[x]; i; i = nxt[i])
        if (v[i] != son[x] && v[i] != f[x]) dfs2(v[i], v[i]);
}
```

```
int lca(int x, int y) {
    for (; top[x] != top[y]; x = f[top[x]])
        if (d[top[x]] < d[top[y]]) {
            int z = x;
            x = y;
            y = z;
        }
    return d[x] < d[y] ? x : y;
}
void Initialize() {
    root[0] = 0;
    tot = 0;
    memset(g, dfn = ed = 0, sizeof(g));
    memset(v, 0, sizeof(v));
    memset(nxt, 0, sizeof(nxt));
    memset(son, -1, sizeof(son));
}
int n, m;
int main() {
    Initialize();
    read(n);
    read(m);
    for (int i = 1; i <= n; i++) read(w[i]), b[i] = w[i];
    for (int i = 1; i < n; i++) {
        int x, y;
        read(x);
        read(y);
        add_edge(x, y);
        add_edge(y, x);
    }
    sort(b + 1, b + n + 1);
    siz = unique(b + 1, b + n + 1) - (b + 1);
    for (int i = 1; i <= n; i++)
        w[i] = lower_bound(b + 1, b + siz + 1, w[i]) - b;
    dfs(1);
    dfs2(1, 1);
    while (m--) {
        int x, y, k;
        read(x), read(y), read(k);
        int LCA = lca(x, y);
        int kth = query(root[x], root[y], 1, siz, root[LCA], root[f[LCA]], k);
        printf("%d\n", b[kth]);
    }
    return 0;
}
```

11.3.2 COT2

```
/*
    题目大意：给出一棵权值树，查询树链上不同权值的数量
    题解：树上莫队，注意查询时额外考虑lca位置
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
const int N = 100010;
const int K = 17;
using namespace std;
struct P {
    int l, r, z, id;
} Q[N];
int lim, pos[N << 1], l, r, c[N], g[N], v[N << 1], nxt[N << 1], ed;
int n, m, x, y, z, loc[N << 1], dfn, st[N], en[N], d[N], f[N][18];
int ans[N], cnt[N], sum;
bool vis[N];
bool cmp(const P& a, const P& b) {
    return pos[a.l] == pos[b.l] ? a.r < b.r : pos[a.l] < pos[b.l];
}
void add(int x, int y) {
    v[++ed] = y;
    nxt[ed] = g[x];
    g[x] = ed;
}
void dfs(int x) {
    for (int i = vis[loc[st[x] = ++dfn] = x] = 1; i <= K; i++)
        f[x][i] = f[f[x][i - 1]][i - 1];
    for (int i = g[x]; i; i = nxt[i])
        if (!vis[v[i]]) d[v[i]] = d[f[v[i]][0] = x] + 1, dfs(v[i]);
    loc[en[x] = ++dfn] = x;
}
int lca(int x, int y) {
    if (x == y) return x;
    if (d[x] < d[y]) swap(x, y);
    for (int i = K; ~i; i--)
        if (d[f[x][i]] >= d[y]) x = f[x][i];
    if (x == y) return x;
    for (int i = K; ~i; i--)
        if (f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
    return f[x][0];
}
void deal(int x) {
    if (!vis[x]) {
```

```
        if (!(--cnt[c[x]])) sum--;
    } else if (!(cnt[c[x]]++))
        sum++;
    vis[x] ^= 1;
}
void read(int& x) {
    int f = 1;
    x = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if ('-' == ch) f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + ch - '0';
        ch = getchar();
    }
}
int b[N];
int main() {
    read(n), read(m);
    for (int i = 1; i <= n; i++) read(c[i]), b[i] = c[i];
    sort(b + 1, b + n + 1);
    int siz = unique(b + 1, b + n + 1) - (b + 1);
    for (int i = 1; i <= n; i++)
        c[i] = lower_bound(b + 1, b + siz + 1, c[i]) - b;
    for (int i = 1; i < n; i++) read(x), read(y), add(x, y), add(y, x);
    dfs(d[1] = 1), lim = (int)sqrt(n * 2 + 0.5);
    for (int i = 1; i <= dfn; i++) pos[i] = (i - 1) / lim + 1;
    for (int i = 1; i <= m; i++) {
        read(x), read(y);
        Q[i].id = i;
        if (st[x] > st[y]) swap(x, y);
        z = lca(x, y);
        if (z == x)
            Q[i].l = st[x], Q[i].r = st[y];
        else
            Q[i].l = en[x], Q[i].r = st[y], Q[i].z = z;
    }
    sort(Q + 1, Q + m + 1, cmp);
    for (int i = 1, l = 1, r = 0; i <= m; i++) {
        if (r < Q[i].r) {
            for (r++; r <= Q[i].r; r++) deal(loc[r]);
            r--;
        }
        if (r > Q[i].r)
```

```
            for (; r > Q[i].r; r--) deal(loc[r]);
        if (l < Q[i].l)
            for (; l < Q[i].l; l++) deal(loc[l]);
        if (l > Q[i].l) {
            for (l--; l >= Q[i].l; l--) deal(loc[l]);
            l++;
        }
        if (Q[i].z) deal(Q[i].z);
        ans[Q[i].id] = sum;
        if (Q[i].z) deal(Q[i].z);
    }
    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

11.3.3 COT3

```
/*
    题目大意：给定一棵N个点的树，1号点为根，每个节点是白色或者黑色。
    双方轮流操作，每次选择一个白色节点，将从这个点到根的路径上的点全部染成黑色。
    无法操作者为败方。
    问先手是否必胜，以及第一步可选节点有哪些。
    题解：首先是博弈方面的分析。
    令SG[x]为，只考虑以x为根的子树时的SG值。
    令g[x]为，只考虑以x为根的子树时，所有后继局面的SG值的集合。那么SG[x]=mex{g[x]}。
    我们考虑怎么计算g[x]。假设x的儿子为v1,v2,...,vk,
    令sum[x]=SG[v1] xor SG[v2] xor .. xor SG[vk]。考虑两种情况：
        1、x为黑色。不难发现以x的每个儿子为根的子树是互相独立的。
        假设这一步选择了vi子树的某一个节点，
        那么转移到的局面的SG值就是sum[x] xor SG[vi] xor (在g[vi]中的某个值)。
        那么我们只需将每个g[vi]整体xor上sum[x] xor SG[vi]再合并到g[x]即可。
        2、x为白色。这时候我们多了一种选择，即选择x点。
        可以发现，选择x点之后x点变成黑色，所有子树仍然独立，
        而转移到的局面的SG值就是sum[x]。
        如果此时不选择x而是选择x子树里的某个白色节点，那么x一样会被染成黑色，
        所有子树依然独立。所以x为白色时只是要向g[x]中多插入一个值sum[x]。
    这样我们就有一个自底向上的DP了。朴素的复杂度是O(N^2)的。
    接下来再考虑第一步可选的节点。我们要考虑选择哪些节点之后整个局面的SG值会变成0。
    假设我们选择了x点，那么从x到根的路径都会被染黑，将原来的树分成了一堆森林。
    我们令up[x]为，不考虑以x为根的子树，将从x到根的路径染黑，剩下的子树的SG值的xor和。
    那么up[x]=up[fa[x]] xor sum[fa[x]] xor sg[x]，其中fa[x]为x的父亲节点编号。
    那么如果点x初始颜色为白色且up[x] xor
   sum[x]=0，那么这个点就是第一步可选的节点。
    剩下的就是优化求SG了。我们需要一个可以快速整体xor并合并的数据结构。
    整体xor可以用二进制Trie打标记实现，合并类似线段树合并。
```

```
*/
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long ll;
const int N = 100010, LOG = 17;
int tot, g[N], nxt[N << 1], v[N << 1];
void init() {
    memset(g, -1, sizeof(g));
    tot = 0;
}
void add_edge(int x, int y) { v[tot] = y, nxt[tot] = g[x], g[x] = tot++; }
struct Node {
    int l, r, tag;
    bool full;
} T[20 * N];
int root[N], tcnt = 0;
int n, col[N], sg[N], sum[N], up[N];
bool vis[N];
inline void update(int x) { T[x].full = T[T[x].l].full && T[T[x].r].full; }
inline void pb(int x) {
    if (T[x].tag) {
        if (T[x].l) T[T[x].l].tag ^= T[x].tag >> 1;
        if (T[x].r) T[T[x].r].tag ^= T[x].tag >> 1;
        if (T[x].tag & 1) swap(T[x].l, T[x].r);
        T[x].tag = 0;
    }
}
int merge(int a, int b) {
    if (!a || T[b].full) return b;
    if (!b || T[a].full) return a;
    pb(a), pb(b);
    int x = ++tcnt;
    T[x].l = merge(T[a].l, T[b].l);
    T[x].r = merge(T[a].r, T[b].r);
    update(x);
    return x;
}
inline int rev(int x) {
    int r = 0;
    for (int i = LOG; i; i--)
        if (x >> i - 1 & 1) r += 1 << LOG - i;
    return r;
}
void insert(int x, int v, int p) {
```

```
    pb(x);
    if (v >> p - 1 & 1) {
        if (!T[x].r) T[x].r = ++tcnt;
        if (p != 1)
            insert(T[x].r, v, p - 1);
        else
            T[T[x].r].full = 1;
    } else {
        if (!T[x].l) T[x].l = ++tcnt;
        if (p != 1)
            insert(T[x].l, v, p - 1);
        else
            T[T[x].l].full = 1;
    }
    update(x);
}
inline int mex(int x) {
    int r = 0;
    for (int i = LOG; x; i--) {
        pb(x);
        if (T[T[x].l].full)
            r += 1 << i - 1, x = T[x].r;
        else
            x = T[x].l;
    }
    return r;
}
void calc(int x) {
    vis[x] = 1;
    int xorsum = 0;
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) {
            calc(v[i]);
            vis[v[i]] = 0;
            xorsum ^= sg[v[i]];
        }
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) {
            T[root[v[i]]].tag ^= rev(xorsum ^ sg[v[i]]);
            root[x] = merge(root[x], root[v[i]]);
        }
    if (!col[x]) insert(root[x], xorsum, LOG);
    sg[x] = mex(root[x]);
    sum[x] = xorsum;
}
int ans[N], cnt = 0;
```

```
void find(int x) {
    vis[x] = 1;
    if ((up[x] ^ sum[x]) == 0 && col[x] == 0) ans[++cnt] = x;
    for (int i = g[x]; ~i; i = nxt[i])
        if (!vis[v[i]]) {
            up[v[i]] = up[x] ^ sum[x] ^ sg[v[i]];
            find(v[i]);
        }
}
int main() {
    scanf("%d", &n);
    init();
    for (int i = 1; i <= n; i++) scanf("%d", col + i);
    for (int i = 1; i < n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        add_edge(x, y), add_edge(y, x);
    }
    for (int i = 1; i <= n; i++) root[i] = ++tcnt;
    calc(1);
    for (int i = 1; i <= n; i++) vis[i] = 0;
    find(1);
    if (!cnt)
        printf("-1\n");
    else {
        sort(ans + 1, ans + cnt + 1);
        for (int i = 1; i <= cnt; i++) printf("%d\n", ans[i]);
    }
    return 0;
}
```

11.4 DIVCNT

11.4.1 DIVCNT1

```
/*
    求约数个数函数d的前缀和
    n<=1e18
*/
#include <bits/stdc++.h>
#define push(x) (stack[++top] = (x))
typedef long long ll;
typedef __int128 lll;
const int N = 1000005;
struct pr {
```

```
    ll x, y;
    pr(ll x0 = 0, ll y0 = 0) : x(x0), y(y0) {}
    inline pr operator+(const pr &B) const { return pr(x + B.x, y + B.y); }
};
ll n;
int top = 0;
pr stack[N];
inline void putint(lll x) {
    static char buf[36];
    if (!x) {
        putchar(48);
        return;
    }
    int i = 0;
    for (; x; buf[++i] = x % 10 | 48, x /= 10)
        ;
    for (; i; --i) putchar(buf[i]);
}
inline bool inner(ll x, ll y) { return n < x * y; }
inline bool steep(ll x, pr v) { return (lll)n * v.x <= (lll)x * x * v.y; }
lll S1() {
    int i, crn = cbrt(n);
    ll srn = sqrt(n), x = n / srn, y = srn + 1;
    lll ret = 0;
    pr L, R, M;
    push(pr(1, 0));
    push(pr(1, 1));
    for (;;) {
        for (L = stack[top--]; inner(x + L.x, y - L.y); x += L.x, y -= L.y)
            ret += x * L.y + (L.y + 1) * (L.x - 1) / 2;
        if (y <= crn) break;
        for (R = stack[top]; !inner(x + R.x, y - R.y); R = stack[--top]) L = R;
        for (; M = L + R, 1;)
            if (inner(x + M.x, y - M.y))
                push(R = M);
            else {
                if (steep(x + M.x, R)) break;
                L = M;
            }
    }
    for (i = 1; i < y; ++i) ret += n / i;
    return ret * 2 - srn * srn;
}
int main() {
    int T;
    for (scanf("%d", &T); T; --T) {
```

```
        scanf("%lld", &n);
        putint(S1());
        putchar(10);
    }
    return 0;
}
```