目录

1 基本操作

1.1 快读 fread

```
char buf[100000], *p1 = buf, *p2 = buf;
inline char nc() {
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF :
        *p1++;
}
inline bool read(int &x) {
    char c = nc(); x = 0;
    if (c == EOF) return false;
    for (; !isdigit(c); c = nc());
    for (; isdigit(c); x = x * 10 + c - '0', c = nc());
    return true;
}
```

1.2 朝鲜人快读

```
#define FI(n) FastIO::read(n)
#define FO(n) FastIO::write(n)
#define Flush FastIO::Fflush()
namespace FastIO {  // 超级快读
    const int SIZE = 1 << 16;
    char buf[SIZE], obuf[SIZE], str[60];
    int bi = SIZE, bn = SIZE, opt;
    double D[] = {0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001,
        0.000000001, 0.0000000001};
    int read(char *s) {
        while (bn) {
            for (; bi < bn && buf[bi] <= ' '; bi++);
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        int sn = 0;
        while (bn) {
            for (; bi < bn && buf[bi] > ' '; bi++) s[sn++] = buf[bi];
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        s[sn] = 0;
        return sn;
    }
    bool read(int &x) {
        int n = read(str), bf = 0;
        if (!n) return 0;
        int i = 0;
        if (str[i] == '-') bf = 1, i++; else if (str[i] == '+') i++;
        for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
        if (bf) x = -x;
        return 1;
```

```
    }
    bool read(long long &x) {
        int n = read(str), bf;
        if (!n) return 0;
        int i = 0;
        if (str[i] == '-') bf = -1, i++; else bf = 1;
        for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
        if (bf < 0) x = -x;
        return 1;
    }
    void write(int x) {
        if (x == 0) obuf[opt++] = '0';
        else {
            if (x < 0) obuf[opt++] = '-', x = -x;
            int sn = 0;
            while (x) str[sn++] = x % 10 + '0', x /= 10;
            for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
        }
        if (opt >= (SIZE >> 1)) {
            fwrite(obuf, 1, opt, stdout);
            opt = 0;
        }
    }
    void write(long long x) {
        if (x == 0) obuf[opt++] = '0';
        else {
            if (x < 0) obuf[opt++] = '-', x = -x;
            int sn = 0;
            while (x) str[sn++] = x % 10 + '0', x /= 10;
            for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
        }
        if (opt >= (SIZE >> 1)) {
            fwrite(obuf, 1, opt, stdout);
            opt = 0;
        }
    }
    void write(unsigned long long x) {
        if (x == 0) obuf[opt++] = '0';
        else {
            int sn = 0;
            while (x) str[sn++] = x % 10 + '0', x /= 10;
            for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
        }
        if (opt >= (SIZE >> 1)) {
            fwrite(obuf, 1, opt, stdout);
            opt = 0;
        }
    }
    void write(char x) {
        obuf[opt++] = x;
        if (opt >= (SIZE >> 1)) {
            fwrite(obuf, 1, opt, stdout);
            opt = 0;
        }
    }
    void Fflush() {
```

```
        if (opt) fwrite(obuf, 1, opt, stdout);
        opt = 0;
    }
};
```

1.3 __int128

```
inline __int128 read() {
    __int128 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;
}

inline void write(__int128 x) {
    if(x<0) {
        putchar('-');
        x=-x;
    }
    if(x>9)
        write(x/10);
    putchar(x%10+'0');
}
```

1.4 二维离散化

```
#define ll long long
const int SIZE=1e5;
struct point{
    ll x,y;
}p[SIZE];
bool cmp_x(point a,point b){return a.x < b.x;}
bool cmp_y(point a,point b){return a.y < b.y;}
void Discrete(int n){//n个点,下标[1,n]
    sort(p+1,p+n+1,cmp_x);
    int last=p[1].x,num=1;
    p[1].x = num = 1;
    for(int i=2;i<=n;i++){
        if(p[i].x == last)p[i].x=num;
        else{
            last = p[i].x;
            p[i].x=++num;
        }
    }
    sort(p+1,p+n+1,cmp_y);
```

```
    last=p[1].y,num=1;
    p[1].y=num=1;
    for(int i=2;i<=n;i++){
        if(p[i].y == last)p[i].y=num;
        else{
            last=p[i].y;
            p[i].y=++num;
        }
    }
}
```

1.5 矩阵快速幂

```
ll powmod(ll a, ll b) {
    ll res = 1;
    a = a % mod;
    while(b) {
        if(b & 1) {
            res = res * a % mod;
        }
        a = a * a % mod;
        b >>= 1;
    }
    return res % mod;
}
ll inv(ll p) { return powmod(p, mod - 2); }
ll is[N], js[N];
class mat { public:
    int n,m;
    ll v[N][N];
    mat(int n,int m) : n(n), m(m){memset(v, 0, sizeof(v));}
    void init() {   memset(v, 0, sizeof(v)); }
    void init1() {
        for(int i = 0; i < N; i++)
            for(int j = 0; j < N; j++)
                v[i][j] = (i == j); //单位矩阵
    }
    mat operator * (const mat B) const {//矩阵乘法 A(n,k)*B(k,m)=C(n,m);
        mat C(n, B.m);
        C.init();
        for(int i = 1; i <= n; i++)
        for(int j = 1; j <= B.m; j++)
        for(int k = 1; k <= m; k++)
            C.v[i][j] = (C.v[i][j] + v[i][k] * B.v[k][j]) % mod;//Mod
        return C;
    }
    mat operator ^ (int t) {//矩阵快速幂 n=m时可用
        mat ans(n, n), now(n, n);
        ans.init1();
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                now.v[i][j] = v[i][j];
        while(t > 0) {
            if(t & 1) ans = ans * now;
```

```
            now = now * now;
            t >>= 1;
        }
        return ans;
    }
    void change() { // 转置
        swap(n, m);
        for(int i = 1; i <= max(n, m); i++) {
            for(int j = i + 1; j <= max(n, m); j++) {
                swap(v[i][j], v[j][i]);
            }
        }
    }
    void Minv() {  // 逆矩阵
        for(int k = 1; k <= n; k++) {
            for(int i = k; i <= n; i++) // 1
                for(int j = k; j <= n; j++)
                    if(v[i][j]) {
                        is[k] = i, js[k] = j;
                        break;
                    }
            for(int i = 1; i <= n; i++) swap(v[k][i], v[is[k]][i]); // 2
            for(int i = 1; i <= n; i++) swap(v[i][k], v[i][js[k]]);
            v[k][k] = inv(v[k][k]); // 3
            for(int j = 1; j <= n; j++)
                if(j != k) // 4
                    v[k][j] = v[k][j] * v[k][k] % mod;
            for(int i = 1; i <= n; i++)
                if(i != k) // 5
                for(int j = 1; j <= n; j++)
                    if(j != k)
                        v[i][j] = (v[i][j] + mod - v[i][k] * v[k][j] % mod) % mod;
            for(int i = 1; i <= n; i++)
                if(i != k)
                v[i][k] = (mod - v[i][k] * v[k][k] % mod) % mod;
        }
        for(int k = n; k; k--) { // 6
            for(int i = 1; i <= n; i++)
                swap(v[js[k]][i], v[k][i]);
            for(int i = 1; i <= n; i++)
                swap(v[i][is[k]], v[i][k]);
        }
    }
};
```

1.6 状态压缩

```
//判断一个数字x二进制下第i位是不是等于1,反之为0
if(((1<<(i-1))&x)>0)

//将一个数字x二进制下第i位更改成1
x=x|(1<<(i-1))

//将一个数字x二进制下第i位更改成0
```

```
x=x^i(1<<(i-1));

//把一个数字二进制下最靠右的第一个1去掉
x=x&(x-1)
```

1.7 二进制枚举

```
for(int i=0;i<(1<<n);i++) {//n个物品取或不取
    for(int j=0;j<n;j++) {
        if( i & (1<<j) ) //取
        else//不取
    }
 }
```

1.8 bitset

```
/*
C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，
它的每一个元素只能是 0 或 1，每个元素仅用 1 bit空间。
*/
//————————————构造方法————————————
bitset<4> bitset1;    //无参构造，长度为 4，默认每一位为 0
bitset<8> bitset2(12);//长度为 8，二进制保存，前面用 0 补充
string s = "100101";
bitset<10> bitset3(s);//长度为10，前面用 0 补充
char s2[] = "10101";
bitset<13> bitset4(s2);//长度为13，前面用 0 补充
cout << bitset1 << endl;//0000
cout << bitset2 << endl;//00001100
cout << bitset3 << endl;//0000100101
cout << bitset4 << endl;//0000000010101
//————————————可用函数————————————
bitset<8> foo ("10011011");

cout << foo.count() << endl;//5    count函数用来求bitset中1的位数
cout << foo.size() << endl;//8    size函数用来求bitset的大小

cout << foo.test(0) << endl;//true    test函数用来查下标处的元素是 0 还是 1
cout << foo.test(2) << endl;//false

cout << foo.any() << endl;//true    any函数检查bitset中是否有 1
cout << foo.none() << endl;//false    none函数检查bitset中是否没有 1
cout << foo.all() << endl;//false    all函数检查bitset中是全部为 1


bitset<8> foo ("10011011");

cout << foo.flip(2) << endl;//10011111    flip函数传参数时，用于将参数位取反
cout << foo.flip() << endl;//01100000    flip函数不指定参数时，将bitset每一位全部取反

cout << foo.set() << endl;//11111111    set函数不指定参数时，将bitset的每一位全部置为 1
cout << foo.set(3,0) <<
    endl;//11110111    set函数指定两位参数时，将第一参数位的元素置为第二参数的值
```

```
cout << foo.set(3) << endl;//11111111    set函数只有一个参数时，将参数下标处置为1

cout << foo.reset(4) << endl;//11101111    reset函数传一个参数时将参数下标处置为0
cout << foo.reset() << endl;//00000000    reset函数不传参数时将bitset的每一位全部置为0

bitset<8> foo ("10011011");

string s = foo.to_string();//将bitset转换成string类型
unsigned long a = foo.to_ulong();//将bitset转换成unsigned long类型
unsigned long long b = foo.to_ullong();//将bitset转换成unsigned long long类型

cout << s << endl;//10011011
cout << a << endl;//155
cout << b << endl;//155
```

1.9 高效位运算 __builtin_ 函数

```
int __builtin_ffs (unsigned int x)
//返回x的最后一位1的是从后向前第几位，比如7368（1110011001000）返回4。
int __builtin_clz (unsigned int x)
//返回前导的0的个数。
int __builtin_ctz (unsigned int x)
//返回后面的0个个数，和__builtin_clz相对。
int __builtin_popcount (unsigned int x)
//返回二进制表示中1的个数。
int __builtin_parity (unsigned int x)
//返回x的奇偶校验位，也就是x的1的个数模2的结果。
//
//此外，这些函数都有相应的usigned long和usigned long long版本，
//只需要在函数名后面加上l或ll就可以了，比如int __builtin_clzll。
```

1.10 散列处理异或碰撞

常用于处理判断出现偶数次（HDU6291），重排后为值域上的连续一段（洛谷 P3792、YNOI）。

```
ull Newrnd() {
    return ((ull) rand() << 45) | ((ull) rand() << 30) | (rand() << 15) | rand();
}
int main() {
    srand(114514);  // random
}
```

1.11 对拍

```
//——————————————data.cpp——————————————
int main() {
    freopen("in","w",stdout);
    srand(time(0));
    int n,m,q;
    n = rand()%100000;
    m = rand()%100000;
    q = rand()%100000;
```

```
    printf("%d %d %d\n",n,m,q);
    for(int i = 1;i <= q;i++){
        int a = rand()%n+1;
        int b = rand()%n+1;
        int c = rand()%2;
        printf("%d %d %d\n",a,b,c);
    }
    return 0;
}

//——————————————1.cpp&&2.cpp————————————————————
int main() {
    freopen("in","r",stdin);
    freopen("1.out","w",stdout);
    //freopen("2.out","w",stdout);
    .....
}

//——————————————duipai.cpp————————————————————
int main() {//Windows
    int cases = 0;
    do{
        if(cases) printf("#%d AC\n",cases);
        cases++;
        system("data.exe > data.txt");
        system("1.exe < data.txt > 1.txt");
        system("2.exe < data.txt > 2.txt");
    }while(!system("fc 1.txt 2.txt"));
    printf("#%d WA",cases);
    return 0;
}
int main() {//Linux
    int i;
    for (i=1;i<=1000;i++) {
        system("./data");
        system("./1");
        system("./2");
        printf("%d : ",i);
        if (system("diff 1.out 2.out"))
            {
                printf("WA\n");
                return 0;
            }
        else printf("AC\n");
    }
    return 0;
}
```

2 STL

2.1 unordered_map 对 pii 进行哈希

```
// call: unordered_map<pii, int, pair_hash> ma;
```

```cpp
// 哈希方法1（较快）
template<typename T>
inline void hash_combine(std::size_t &seed, const T &val) {
    seed ^= std::hash<T>()(val) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
}

template<typename T>
inline void hash_val(std::size_t &seed, const T &val) {
    hash_combine(seed, val);
}

template<typename T, typename... Types>
inline void hash_val(std::size_t &seed, const T &val, const Types &... args) {
    hash_combine(seed, val);
    hash_val(seed, args...);
}

template<typename... Types>
inline std::size_t hash_val(const Types &... args) {
    std::size_t seed = 0;
    hash_val(seed, args...);
    return seed;
}

struct pair_hash {
    template<class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        return hash_val(p.first, p.second);
    }
};

// 哈希方法2（部分冲不过）
struct pair_hash {
    template<class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        auto h1 = std::hash<T1>{}(p.first);
        auto h2 = std::hash<T2>{}(p.second);
        return h1 ^ h2;
    }
};
```

2.2 pb_ds 实现平衡树

```cpp
#define _EXT_CODECVT_SPECIALIZATIONS_H 1
#define _EXT_ENC_FILEBUF_H 1
#undef __MINGW32__

#include<bits/stdc++.h>
#include<bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;

class PBDS {
```

```
private:
    typedef tree<ll, null_type, less<ll>, rb_tree_tag, tree_order_statistics_node_update> Tree;
    Tree T;
public:
    void insert(ll x, int id) { // 插入x数，需传入第几次操作进行简单处理
        T.insert((x << 20) + id);    // 简单的处理
    }

    void remove(ll x) { // 删除x数(若有多个相同的数，因只删除一个)
        T.erase(T.lower_bound(x << 20));
    }

    ll get_rank(ll x) { // 查询x数的排名(排名定义为比当前数小的数的个数+1)
        return T.order_of_key(x << 20) + 1;
    }

    ll get_val(ll x) {  // 查询排名为x的数
        return *T.find_by_order(x - 1) >> 20;
    }

    ll get_pre(ll x) {  // 求x的前驱(前驱定义为小于x，且最大的数)
        return *--T.lower_bound(x << 20) >> 20;
    }

    ll get_next(ll x) { // 求x的后继(后继定义为大于x，且最小的数)
        return *T.lower_bound((x + 1) << 20) >> 20;
    }
} rbt;  // 注意不能叫tree，否则会冲突
```

2.3 rope

写一种数据结构，支持任意位置插入、删除和修改

```
#include <bits/stdc++.h>
#include <bits/extc++.h>

using namespace std;
using namespace __gnu_cxx;
rope<char> tree;
inline void read_str(char *s, int len) {    // 读入长度为len的字符串
    s[len] = '\0';
    len--;
    for (int i = 0; i <= len; i++) {
        s[i] = '\0';
        while (s[i] < 32 || 126 < s[i]) s[i] = getchar();
    }
}

inline void read_int(int &x) {
    x = 0;
    char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar())) x = x * 10 + ch - '0';
}
```

```
const int MAXN = 2e6 + 5;
char word[MAXN];

int main() {
    int T;
    int now = 0;
    scanf("%d", &T);
    while (T--) {
        int opt = '1', x;
        while (!isalpha(opt = getchar()));
        while (isalpha(getchar()));
        // 格式:Move k
        // 将光标移动到第k个字符之后，如果k=0，将光标移到文本开头
        if (opt == 'M') read_int(now);
        // Insert n s
        // 在光标处插入长度为n的字符串s，光标位置不变n>=1
        else if (opt == 'I') {
            read_int(x);
            read_str(word, x);
            tree.insert(now, word);
        }
        // Delete n
        // 删除光标后的n个字符，光标位置不变，n>=1
        else if (opt == 'D') {
            read_int(x);
            tree.erase(now, x);
        }
        // Get n
        // 输出光标后的n个字符，光标位置不变，n>=1
        else if (opt == 'G') {
            read_int(x);
            x--;
            for (int i = now; i <= now + x; i++) printf("%c", tree[i]);
            printf("\n");
        } else if (opt == 'P') now--;   // 光标前移
        else now++; //光标后移
    }
}
```

3 数论

3.1 中国剩余定理

$$\begin{cases} x \equiv a_1 (\mod m_1) \\ x \equiv a_2 (\mod m_2) \\ x \equiv a_3 (\mod m_3) \\ x \equiv a_4 (\mod m_4) \end{cases}$$

其中，$m_1, m_2, ..., m_k$ 为两两互质的整数

```
ll a[MAXN], m[MAXN];
void exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1, y = 0;
        return;
```

```
    }
    exgcd(b, a%b, x, y);
    ll z = x;
    x = y, y = z - y * (a / b);
}
int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld%lld", &m[i], &a[i]);
    }
    ll mul_sum = 1;
    for (int i = 1; i <= n; i++) mul_sum *= m[i];
    ll res = 0;
    for (int i = 1; i <= n;i++) {
        ll M = mul_sum / m[i];
        ll x = 0, y = 0;
        exgcd(M, m[i], x, y);
        res += a[i] * M * (x < 0 ? x + m[i] : x);
    }
    printf("%lld\n", res % mul_sum);
}
```

3.2 蔡勒公式（1582 之后）

```
int WhatDay(int year, int month, int dday) {//0是周日，1是周一，6是周六
    if (month < 3) {
        year -= 1;
        month += 12;
    }
    int c = int(year / 100), y = year - 100 * c;
    int w = int(c / 4) - 2 * c + y + int(y / 4) + (26 * (month + 1) / 10) + dday - 1;
    w = (w % 7 + 7) % 7;
    return w;
}
```

4 字符串

4.1 字符串哈希

4.1.1 区间一维哈希

```
typedef unsigned long long ull;
namespace hash {
    const ull seed = 19260817;
    ull base[SIZE], hash[SIZE];
    void init() {
        base[0] = 1;
        for (int i = 1; i < SIZE; i++)base[i] = base[i - 1] * seed;
    }
    ull _hash(int l, int r) { return hash[r] - hash[l - 1] * base[r - l + 1]; }
    void getHash(char str[], int len) {
        for (int i = 1; i <= len; i++) hash[i] = hash[i - 1] * seed + str[i] - 'a' + 3;
```

```
    }
}
```

4.1.2 二维哈希

```
const ull base1 = 19260817;
const ull base2 = 233;
const int MAXN = 1005;
ull b1[MAXN], b2[MAXN];
void init() {
    b1[0] = 1, b2[0] = 1;
    for (int i = 1; i < MAXN; i++) b1[i] = b1[i - 1] * base1;
    for (int i = 1; i < MAXN; i++) b2[i] = b2[i - 1] * base2;
}
char pat[55][55];
ull get_hash1(int p, int q) {
    ull ans = 0;
    for (int i = 1; i <= p; i++) {
        ull tmp = 0;
        for (int j = 1; j <= q; j++) {
            tmp = tmp * base1 + pat[i][j];
        }
        ans = ans * base2 + tmp;
    }
    return ans;
}

char str[MAXN][MAXN];
ull Hash[2][MAXN][MAXN];
void get_hash2(int n, int m, int p, int q) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= q; j++) Hash[0][i][j] = (j == 1) ? str[i][j] : Hash[0][i][j - 1]
            * base1 + str[i][j];
        for (int j = q + 1; j <= m; j++)
            Hash[0][i][j] = Hash[0][i][j - 1] * base1 + str[i][j] - str[i][j - q] * b1[q];
    }
    for (int j = 1; j <= m; j++) {
        for (int i = 1; i <= p; i++)
            Hash[1][i][j] = (i == 1) ? Hash[0][i][j] : Hash[1][i - 1][j] * base2 +
                Hash[0][i][j];
        for (int i = p + 1; i <= n; i++)
            Hash[1][i][j] = Hash[1][i - 1][j] * base2 + Hash[0][i][j] - Hash[0][i - p][j] *
                b2[p];
    }
}
multiset<ull> st;
int main() {
    init();
    int n, m, t, p, q; int kass = 1;
    while (~scanf("%d%d%d%d%d", &n, &m, &t, &p, &q) && (n + m + t + p + q)) {
        st.clear();
        for (int i = 1; i <= n; i++) {
            scanf("%s", str[i] + 1);
        }
```

```
        for (int i = 1; i <= t; i++) {
            for (int i = 1; i <= p; i++) {
                scanf("%s", pat[i] + 1);
            }
            st.insert(get_hash1(p, q));
        }
        get_hash2(n, m, p, q);
        int res = 0;
        for (int i = p; i <= n; i++) {
            for (int j = q; j <= m; j++) st.erase(Hash[1][i][j]);
        }
        printf("Case %d: %d\n", kass++, t - SZ(st));
    }
}
```

4.2 Next 函数

4.2.1 求 next 函数

```
/*
    input: ABCABDABCD
    index   0   1   2   3   4   5   6   7   8   9   10
    x[]     A   B   C   A   B   D   A   B   C   D   \0
    nxt[]   -1  -1  -1  0   1   -1  0   1   2   -1  0
*/
// call: scanf("%s", str+1); get_next(str+1, strlen(str+1), nex+1);
void get_next(char x[], int x_len, int nxt[]) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < x_len; nxt[i++] = j) {
        while (~j && x[j + 1] != x[i])j = nxt[j];
        if (x[j + 1] == x[i])j++;
    }
}

/*
    input: ABCABDABCD
    index   1   2   3   4   5   6   7   8   9   10  11
    x[]     A   B   C   A   B   D   A   B   C   D   \0
    nxt[]   0   0   0   1   2   0   1   2   3   0   0
*/
// call: scanf("%s", str+1); get_next(str, strlen(str+1), nex);
void get_next(char x[], int x_len, int nxt[]) {
    nxt[1] = 0;
    for (int i = 2, j = 0; i <= x_len; i++) {
        while (j && x[j + 1] != x[i]) j = nxt[j];
        if (x[j + 1] == x[i]) ++j;
        nxt[i] = j;
    }
}
```

4.2.2 求出每个循环节的数量和终点位置（HDU1358）

```
/*
```

```
    input   output
    3       2 2
    aaa     3 3

    12              2 2
    aabaabaabaab    6 2
                    9 3
                    12 4
*/
int main() {
    int n;
    int cas = 0;
    while (~scanf("%d", &n) && n) {
        scanf("%s", str+1);
        get_next(str+1, n, nex+1);  // 方法1
        for (int i = 2; i <= n; i++) {
            if (nex[i] != -1 && (i % (i - nex[i] - 1) == 0))
                printf("%d %d\n", i, i / (i - nex[i] - 1));
        }
        puts("");
    }
}
```

4.2.3 求同时是前缀和后缀的串长（POJ2752）

```
/*
    input               output
    ababcababababcabab  2 4 9 18
    aaaaa               1 2 3 4 5
*/
int main() {
    while (~scanf("%s", s + 1)) {
        int cnt = 0;
        int len;
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        for (int t = nex[len]; ~t; t = nex[t+1]) {
            if (s[t+1] == s[len]) ans[cnt++] = t + 1;
        }
        for (int i = cnt - 1; i >= 0; i--)printf("%d ", ans[i]);
        printf("%d\n", len);
    }
}
```

4.2.4 求字符串每个前缀和串匹配成功的次数和（HDU3336）

```
/*
    input   output
    4       6
    abab
    6       12
    ababab
*/
int get_next(char x[], int x_len, int nxt[], int i, int j) {
```

```
    while (~j && x[j + 1] != x[i])j = nxt[j];
    if (x[j + 1] == x[i])j++;
    nxt[i++] = j;
    return j;
}

char str[MAXN];
int nex[MAXN], val[MAXN];

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n;
        scanf("%d", &n);
        scanf("%s", str + 1);
        int len = strlen(str + 1);
        int last = -1;
        nex[1] = -1;
        int res = 0;
        for (int i = 1; i <= len; i++) {
            last = get_next(str + 1, len, nex + 1, i, last);
            if (nex[i] < 0) val[i] = 1;
            else val[i] = (val[nex[i] + 1] + 1) % mod;
            res = (res + val[i]) % mod;
        }
        printf("%d\n", res);
    }
}
```

4.2.5 求循环节数量（POJ2406）

```
/*
    input   output
    abcd    1
    aaaa    4
    ababab  3
*/
int main() {
    while (~scanf("%s", s + 1)) {
        if (s[1] == '.')break;
        int len;
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        printf("%d\n", len % (len - nex[len] - 1) ? 1 : len / (len - nex[len] - 1));
    }
}
```

4.2.6 求第一个串的前缀和第二个串的后缀的最大匹配（HDU2594）

```
/*
    input       output
    clinton     0
    homer
```

```
    riemann     rie 3
    marjorie
*/
int main() {
    while (~scanf("%s%s", a + 1, b + 1)) {
        int la = strlen(a + 1), lb = strlen(b + 1);
        strcat(a + 1, b + 1);
        int len = la + lb;
        get_next(a + 1, len, nex + 1);  // 方法1
        int k;
        for (k = nex[len]; k >= la || k >= lb; k = nex[k+1]);
        if (k == -1) puts("0");
        else {
            for (int i = 0; i <= k; i++)printf("%c", a[i+1]);
            printf(" %d\n", k + 1);
        }
    }
}
```

4.2.7 求补上最少字母数量使得这是个循环串（HDU3746）

```
/*
    input:  output
    3
    aaa     0
    abca    2
    abcde   5
*/
int main() {
    int T, len;
    scanf("%d", &T);
    while (T--) {
        scanf("%s", s + 1);
        get_next(s + 1, len = strlen(s + 1), nex + 1);  // 方法1
        int L = len - (nex[len] + 1);
        if (L < len && len % L == 0) puts("0");
        else printf("%d\n", L - len % L);
    }
}
```

4.2.8 习题整理

[NOI2014] 动物园

对于字符串 S 的前 i 个字符构成的子串，既是它的后缀同时又是它的前缀，并且该后缀与该前缀不重叠，将这种字符串的数量记作 $num[i]$.

res 为 $(num[i]+1)$ 的乘积.

时间复杂度：$O(n)$.

```
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%s", str + 1);
        int len;
```

```
        get_next(str, len = strlen(str + 1), nex);  // 方法2
        for (int i = 1; i <= len; i++) {
            val[i] = (val[nex[i]] + 1) % mod;
        }
        ll res = 1ll;
        for (int i = 2, j = 0; i <= len; i++) {
            while (j && str[j + 1] != str[i]) j = nex[j];
            if (str[j + 1] == str[i]) j++;
            while (j * 2 > i) j = nex[j];   // 去除重叠的
            num[i] = val[j];
            // res = (res * ((ll) val[j] + 1ll)) % mod;
        }
        printf("%lld\n", res);
    }
}
```

4.3 KMP

4.3.1 统计模式串出现次数，出现位置，前缀 border 长度

```
/*
    时间复杂度O(x_len + y_len)，空间复杂度O(x_len)
*/
int nex[MAXN];
// x为模式串， y为文本串
// call: scanf("%s %s", a+1, b+1); kmp(b+1, strlen(b+1), a+1, strlen(a+1));
int kmp(char x[], int x_len, char y[], int y_len) {
    int i, j;
    int ans = 0;
    get_next(x, x_len, nex);    // 方法1
    for (j = -1, i = 0; i < y_len; i++) {
        while (~j && x[j + 1] != y[i])j = nex[j];
        if (x[j + 1] == y[i])j++;
        if (j == x_len - 1) {
            printf("%d\n", i - x_len + 2);  // 出现的位置，可选，从1开始计数
            ans++, j = nex[j];
        }
    }
    for (i = 0; i < x_len; i++) printf("%d ", nex[i] + 1);  // 每个前缀的最长border的长度
    return ans;
}
```

4.3.2 矩阵加速 KMP, 求长度为 n 的不包含长度为 m 的子串的串个数（[HNOI2008]GT 考试）

$$\sum_{k=0}^{m-1} f[i-1][k] * g[k][j]$$

$f[i][j]$ 为长串匹配到第 i 位，短串最多可以匹配到第 j 位的方案数

$g[j][k]$ 为了计算长度为 j 的已经匹配好了的串可以用多少种数字变为 k，枚举一个数字，看它在短串中最长可以匹配到最多多长的前缀

```
/*
    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
```

```
    input   4 3 100
            111
    output  81
*/
int nex[MAXN];
mat get_g(char x[], int x_len) {
    get_next(x, x_len, nex);
    mat g = mat(x_len, x_len);
    for (int i = 0; i < x_len; i++) {
        for (char ch = '0'; ch <= '9'; ch++) {
            int j = i;
            while (j && x[j+1] != ch) j = nex[j];
            if (x[j+1] == ch) j++;
            g.v[i][j] = (ll)(g.v[i][j] + 1ll) % mod;
        }
    }
    return g;
}
int n, m;
char str[MAXN];
int main() {
    scanf("%d%d%lld", &n, &m, &mod);
    scanf("%s", str + 1);
    mat g = get_g(str, strlen(str+1));
    g = g^n;
    mat f(m, 1);
    f.v[0][0] = 1;
    f =f*g;
    ll res = 0;
    for (int i = 0; i < m; i++) {
        res = (ll)(res + f.v[0][i]) % mod;
    }
    printf("%lld\n", res);
}
```

4.4 EXKMP

4.4.1 求 z 函数和 LCP

LCP：最长公共前缀

z 函数数组 z：串 b 与 b 的每一个后缀的 LCP 长度。

$extend$ 数组：串 b 与串 a 的每一个后缀的 LCP 长度。总时间复杂度：$O(|a|+|b|)$.

```
/*
    index       1   2   3   4   5   6   7   8
    char a[]    a   a   a   a   b   a   a   '\0'
    extend[]    4   3   2   1   0   2   1
    char b[]    a   a   a   a   a   '\0'
    z[]         5   4   3   2   1
*/
// call: scanf("%s", a+1); getLCP(a+1, strlen(a+1), z+1);
void getLCP(char T[], int T_len, int z[]) {
    int i, len = T_len;
    z[0] = len;
    for (i = 0; i < len - 1 && T[i] == T[i + 1]; i++);
```

```
    z[1] = i;
    int a = 1;
    for (int k = 2; k < len; k++) {
        int p = a + z[a] - 1, L = z[k - a];
        if ((k - 1) + L >= p) {
            int j = max((p - k + 1), 0);
            while (k + j < len && T[k + j] == T[j])j++;
            z[k] = j, a = k;
        } else z[k] = L;
    }
}

// call: scanf("%s%s", a+1, b+1); exkmp(a+1, strlen(a+1), b+1, strlen(b+1), ex+1, z+1);
void exkmp(char S[], int S_len, char T[], int T_len, int extend[], int z[]) {
    getLCP(T, T_len, z);
    int a = 0;
    int MinLen = min(S_len, T_len);
    while (a < MinLen && S[a] == T[a])a++;
    extend[0] = a, a = 0;
    for (int k = 1; k < S_len; k++) {
        int p = a + extend[a] - 1, L = z[k - a];
        if ((k - 1) + L >= p) {
            int j = max((p - k + 1), 0);
            while (k + j < S_len && j < T_len && S[k + j] == T[j])j++;
            extend[k] = j;
            a = k;
        } else extend[k] = L;
    }
}
```

4.4.2 循环位移有多少数比原数大小相等，去重（HDU4333）

包含对获得的串进行去重。

总时间复杂度：$O(n)$

```
/*
    input   output
    1       Case 1: 1 1 1
    341
*/
char str[MAXN];
int z[MAXN];
int main() {
    int T; scanf("%d", &T);
    int kase = 1;
    while (T--) {
        scanf("%s", str + 1);
        int len = strlen(str + 1);
        for (int i = 1; i <= len; i++) str[len + i] = str[i];
        getLCP(str + 1, len * 2, z + 1);
        int L = 0, E = 0, G = 0;
        for (int i = 1; i <= len; i++) {
            if (z[i] >= len) E++;
            else if (str[z[i]+1] > str[z[i]+i]) L++;
            else G++;
        }
```

```
        printf("Case %d: ", kase++);
        printf("%d %d %d\n", L/E, E/E, G/E);    // 去重相关
    }
}
```

4.4.3 选出 $n \times n$ 对并把每一对连接成一个单词求回文对数（POJ3376）

```
/*  input   output
    3       5
    1 a     (aa aba aba abba baab)
    2 ab
    2 ba
*/
bool pali[2][MAXN];
namespace EXKMP {
    void getLCP(char T[], int T_len, int z[]) {
        int i, len = T_len;
        z[0] = len;
        for (i = 0; i < len - 1 && T[i] == T[i + 1]; i++);
        z[1] = i;
        int a = 1;
        for (int k = 2; k < len; k++) {
            int p = a + z[a] - 1, L = z[k - a];
            if ((k - 1) + L >= p) {
                int j = max((p - k + 1), 0);
                while (k + j < len && T[k + j] == T[j])j++;
                z[k] = j, a = k;
            } else z[k] = L;
        }
    }
    void exkmp(char S[], int S_len, char T[], int T_len, int extend[], int z[], int flag, int
        be) {
        getLCP(T, T_len, z);
        int a = 0;
        int MinLen = min(S_len, T_len);
        while (a < MinLen && S[a] == T[a])a++;
        extend[0] = a, a = 0;
        for (int k = 1; k < S_len; k++) {
            int p = a + extend[a] - 1, L = z[k - a];
            if ((k - 1) + L >= p) {
                int j = max((p - k + 1), 0);
                while (k + j < S_len && j < T_len && S[k + j] == T[j])j++;
                extend[k] = j;
                a = k;
            } else extend[k] = L;
        }
        for (int i = 0; i < S_len; i++) {
            if (i + extend[i] == S_len) pali[flag][be + i] = 1;
        }
    }
}

class TRIE {    public:
    int T[MAXN][26], cnt[MAXN], sum[MAXN];
```

```
    int top;
    void init() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        memset(cnt, 0, sizeof(cnt)), memset(sum, 0, sizeof(sum));
    }
    void insert(char str[], int n, int be) {
        int u = 0;
        for (int i = 0; i < n; i++) {
            int ch = str[i] - 'a';
            sum[u] += pali[0][be + i];
            if (!T[u][ch]) {
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
        cnt[u]++;
    }
    ll find(char str[], int n, int be) {
        ll ans = 0;
        int u = 0;
        for (int i = 0; i < n; i++) {
            int ch = str[i] - 'a';

            u = T[u][ch];
            if (!u) break;
            if ((i < n - 1 && pali[1][be + i + 1]) || (i == n - 1)) ans += cnt[u];
        }
        if (u) ans += sum[u];
        return ans;
    }
} tree;

int be[MAXN], en[MAXN]; // save begin and end
char str_[MAXN], _str[MAXN];
int extend[MAXN], z[MAXN];

int main() {
    tree.init();
    int n; scanf("%d", &n);
    int sumlen = 0;
    for (int i = 1; i <= n; i++) {
        int len; scanf("%d", &len);
        scanf("%s", str_ + sumlen);
        for (int j = 0; j < len; j++) {
            _str[sumlen + j] = str_[sumlen + len - j - 1];
        }
        be[i] = sumlen;
        en[i] = sumlen + len - 1;
        sumlen = sumlen + len;
        EXKMP::exkmp(str_ + be[i], len, _str + be[i], len, extend, z, 0, be[i]);
        EXKMP::exkmp(_str + be[i], len, str_ + be[i], len, extend, z, 1, be[i]);
        tree.insert(str_ + be[i], len, be[i]);
    }
    ll res = 0;
```

```
    for (int i = 1; i <= n; i++) {
        res += tree.find(_str + be[i], en[i] - be[i] + 1, be[i]);
    }
    printf("%lld\n", res);
}
```

4.5 AC 自动机

4.5.1 标准的 AC 自动机

```
const int MAXN = 1e5 + 5;

class AC_Automaton {
public:
    int T[MAXN][26], val[MAXN], top;  //Trie相关
    int fail[MAXN];
    queue<int> q;
    // int pid[SIZE];  //对应字符串编号

    void init() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        memset(val, 0, sizeof(val));
//        memset(pid, 0, sizeof(pid));
    }

    AC_Automaton() {
        init();
    }

    void insert(char str[], int lenstr, int _pid) {
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
            int ch = str[i] - 'a';
            if (!T[u][ch]) {
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
        val[u]++;
        // pid[u] = _pid;
    }

    void build() {
        for (int i = 0; i < 26; i++)
            if (T[0][i]) {
                fail[T[0][i]] = 0;
                q.push(T[0][i]);
            }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
```

```
                if (T[u][i]) {
                    fail[T[u][i]] = T[fail[u]][i];
                    q.push(T[u][i]);
                } else T[u][i] = T[fail[u]][i];
        }
    }

    int query(char str[], int lenstr) {
        int u = 0, ans = 0;
        for (int i = 1; i <= lenstr; i++) {
            int id = str[i] - 'a';
            u = T[u][id];
            for (int j = u; j && (~val[j]); j = fail[j]) {
                ans += val[j]; //val[j]=-1;
                if (pid[j]) {
                    qs[pid[j]].cnt++;
                }
            }

        }
        return ans;
    }
} tree;
```

4.5.2 AC 自动机上检查无限长循环串（[POI2000] 病毒）

```
class AC {  public:
    int T[MAXN][2], top;
    bool endpos[MAXN];
    int fail[MAXN];
    queue<int> q;
    void init() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        endpos[0] = fail[0] = 0;
    }

    void insert(char str[], int lenstr) {
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
            int ch = str[i] - '0';
            if (!T[u][ch]) {
                endpos[0] = fail[top] = 0;
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
        endpos[u] = 1;
    }

    void build() {
        for (int i = 0; i < 2; i++) {
            if (T[0][i]) {
```

```
                fail[T[0][i]] = 0;
                q.push(T[0][i]);
            }
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (endpos[fail[u]]) endpos[u] = 1;
            for (int i = 0; i < 2; i++) {
                if (T[u][i]) {
                    fail[T[u][i]] = T[fail[u]][i];
                    q.push(T[u][i]);
                } else T[u][i] = T[fail[u]][i];
            }
        }
    }

    bool vis[MAXN];
    void dfs(int u) {
        if (vis[u]) {
            printf("TAK\n");    // 存在无限长的安全代码
            exit(0);
        }
        vis[u] = 1;
        for (int i = 0; i < 2; i++) {
            int v = T[u][i];
            if (!endpos[v]) {
                dfs(v);
            }
        }
        vis[u] = 0;
    }
} ac;

char str[MAXN];
int len[MAXN];

int main() {
    int n; scanf("%d", &n);
    ac.init();
    for (int i = 1; i <= n; i++) {
        scanf("%s", str + 1);
        len[i] = strlen(str + 1);
        ac.insert(str, len[i]);
    }
    ac.build();
    ac.dfs(0);
    printf("NIE\n");    // 不存在无限长的安全代码
}
```

4.5.3 AC 自动机 + 矩阵快速幂

有 m 种 DNA 序列是致病的，问长为 n 且不包含致病序列的 DNA 有多少种

设 $dp[i][j]$ 为走了 j 步到达节点 i 的方案数，显然 $dp[0][0]=1$，当 $i\neq 0$ 时，$dp[i][0]=0$。

设 $a[i][j]$ 为从节点 i 到达节点 j 是否存在边。

最终得 $dp[i][j]=\sum_{x=0}^{N}a[x][i]*dp[x][j-1]$

```
int char2id(char ch) {
    switch (ch) {
        case 'A': return 0;
        case 'T': return 1;
        case 'G': return 2;
        case 'C': return 3;
    }
}
class AC {  public:
    int T[MAXN][4], top;
    bool endpos[MAXN];
    int fail[MAXN];
    queue<int> q;
    void init() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        endpos[0] = fail[0] = 0;
    }
    void insert(char str[], int lenstr) {
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
            int ch = char2id(str[i]);
            if (!T[u][ch]) {
                endpos[top] = fail[top] = 0;    // init a node
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
        endpos[u] = 1;
    }
    void build() {
        for (int i = 0; i < 4; i++)
            if (T[0][i]) {
                fail[T[0][i]] = 0;
                q.push(T[0][i]);
            }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (endpos[fail[u]]) endpos[u] = 1;
            for (int i = 0; i < 4; i++)
                if (T[u][i]) {
                    fail[T[u][i]] = T[fail[u]][i];
                    q.push(T[u][i]);
                } else T[u][i] = T[fail[u]][i];
        }
    }
    void build_mat(mat &mat) {
        mat.init();
        for (int i = 0; i < top; i++) {
            if (endpos[i]) continue;
            for (int j = 0; j < 4; j++) {
                if (!endpos[T[i][j]]) mat.v[i][T[i][j]]++;
```

```
            }
        }
    }
} ac;

char str[MAXN];
int main() {
    int m, n;
    while (~scanf("%d%d", &m, &n)) {
        ac.init();
        while (m--) {
            scanf("%s", str + 1);
            ac.insert(str, strlen(str + 1));
        }
        ac.build();
        mat ma(ac.top, ac.top);
        ac.build_mat(ma);
        ma = ma ^ n;
        int res = 0;
        for (int i = 0; i < ac.top; i++) {
            res = (res + ma.v[0][i]) % mod;
        }
        printf("%d\n", res);
    }
}
```

4.6 字典树/Trie 树

```
class Trie {
public:
    int T[MAXN][26], val[MAXN], top;

    Trie() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
        memset(val, 0, sizeof(val));
    }

    void insert(char str[], int lenstr) { // cal: scanf("%s", str+1), len = strlen(str+1),
        tree.insert(str, len);
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
            int ch = str[i] - 'a';
            if (!T[u][ch]) {
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
    }

    int search(char str[], int lenstr) {
        int u = 0;
        for (int i = 1; i <= lenstr; i++) {
```

```
            int ch = str[i] - 'a';
            if (!T[u][ch])return -1; //找不到
            u = T[u][ch];
        }
        if (!val[u]) {
            val[u] = 1;
            return 0;
        }
        return val[u];
    }
} tree;
```

4.7 后缀数组 SA

4.7.1 获取 SA 和 rank 数组

```
/*
    [input] aaabaabaaaab
    [output]
    rank    Suffix          pos(sa)     index
    1       aaaab           8           1
    2       aaab            9           2
    3       aaabaabaaaab    1           3
    4       aab             10          4
    5       aabaaaab        5           5
    6       aabaabaaaab     2           6
    7       ab              11          7
    8       abaaaab         6           8
    9       abaabaaaab      3           9
    10      b               12          10
    11      baaaab          7           11
    12      baabaaaab       4           12
*/
namespace SA {  // private ver.
    int len;
    int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];

    void run(char s[], int _len) {   // call: run(str, strlen(str+1));
        len = _len;
        int m = max(len, 300);
        // memset(cnt, 0, sizeof(cnt));
        for (int i = 0; i <= m; i++) cnt[i] = 0;
        for (int i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
        for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;
        for (int i = len+1; i <= (len<<1); i++) oldrk[i] = rk[i] = 0;

        for (int w = 1; w <= len; w <<= 1) {
            // memset(cnt, 0, sizeof(cnt));
            for (int i = 0; i <= m; i++) cnt[i] = 0;
            for (int i = 1; i <= len; i++) id[i] = sa[i];
            for (int i = 1; i <= len; i++) ++cnt[rk[id[i] + w]];
            for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (int i = len; i >= 1; i--) sa[cnt[rk[id[i] + w]]--] = id[i];
```

```
            for (int i = 0; i <= m; i++) cnt[i] = 0;
            // memset(cnt, 0, sizeof(cnt));
            for (int i = 1; i <= len; i++) id[i] = sa[i];
            for (int i = 1; i <= len; i++) ++cnt[rk[id[i]]];
            for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (int i = len; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
            // memcpy(oldrk, rk, sizeof(rk));
            for (int i = 0; i <= len; i++) oldrk[i] = rk[i];
            for (int p = 0, i = 1; i <= len; i++) {
                if (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] == oldrk[sa[i - 1] +
                    w]) rk[sa[i]] = p;
                else rk[sa[i]] = ++p;
            }
        }
    }
}

namespace SA {  // 77 ver.
    int len;
    int sa[MAXN], rk[MAXN], oldrk[MAXN << 1], id[MAXN], cnt[MAXN], px[MAXN];

    bool cmp(int x, int y, int w) {
        return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
    }

    void run(char s[], int _len) {   // call: run(str, strlen(str+1));
        int i, m = 300, p, w;
        len = _len;
        // memset(cnt, 0, sizeof(cnt));
        for (i = 1; i <= m; i++) cnt[i] = 0;
        for (i = 1; i <= len; i++) ++cnt[rk[i] = s[i]];
        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (i = len; i >= 1; i--) sa[cnt[rk[i]]--] = i;

        for (w = 1; w <= len; w <<= 1, m = p) {
            for (p = 0, i = len; i > len - w; i--) id[++p] = i;
            for (i = 1; i <= len; i++)
                if (sa[i] > w) id[++p] = sa[i] - w;

            // memset(cnt, 0, sizeof(cnt));
            for (i = 0; i <= m; i++) cnt[i] = 0;
            for (i = 1; i <= len; i++) ++cnt[px[i] = rk[id[i]]];
            for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for (i = len; i >= 1; i--) sa[cnt[px[i]]--] = id[i];

            // memcpy(oldrk, rk, sizeof(rk));
            for (i = 0; i <= len; i++) oldrk[i] = rk[i];
            for (p = 0, i = 1; i <= len; i++) {
                rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
            }
        }
    }
}
```

4.7.2 后缀数组 +ST 表求 lcp

```cpp
int height[MAXN];
namespace SA {  // private ver.
    int len;
    int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];

    void run(char s[], int _len) {}   // step1 call: run(str, strlen(str+1));

    void get_height(char s[]) { // step2 call: get_height(str)
        int k = 0;
        for (int i = 1; i <= len; i++) rk[sa[i]] = i;
        for (int i = 1; i <= len; i++) {
            if (rk[i] == 1) continue;
            if (k) --k;
            int j = sa[rk[i] - 1];
            while (j + k <= len && i + k <= len && s[i + k] == s[j + k]) k++;
            height[rk[i]] = k;
        }
    }
}
const int MAXL = 22;
namespace RMQ { // ST, O(1) get LCP
    int mm[MAXN], best[MAXL][MAXN];

    void init(int n) {  // step3 call: init(strlen(str+1))
        mm[0] = -1;
        for (int i = 1; i <= n; i++)
            mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
        for (int i = 1; i <= n; i++)best[0][i] = i;
        for (int i = 1; i <= mm[n]; i++)
            for (int j = 1; j + (1 << i) - 1 <= n; j++) {
                int a = best[i - 1][j];
                int b = best[i - 1][j + (1 << (i - 1))];
                if (height[a] < height[b])best[i][j] = a;
                else best[i][j] = b;
            }
    }

    int askRMQ(int a, int b) {
        int t = mm[b - a + 1];
        b -= (1 << t) - 1;
        a = best[t][a];
        b = best[t][b];
        return height[a] < height[b] ? a : b;
    }

    /*
        get_SA.cpp example's index
        get_LCP(2, 4) => 2
    */
    int get_LCP(int a, int b) {
        if (a == b)return INF;
        if (a > b)swap(a, b);
        return height[askRMQ(a + 1, b)];
```

```
    }

}
```

4.7.3 后缀链接字典序最小（arc050_d）

```
/*
    input   ouput
    3       1 3 2
    arc
    2
    zz      1 2 / 2 1
    5
    abaab   3 1 4 2 5
*/
using RMQ::get_LCP; using SA::rk;
int num[MAXN];
int N; char str[MAXN];
bool cmp(int x, int y) {
    if (x < y) {
        int lcp = get_LCP(rk[x], rk[y]);
        if (lcp < N - y + 1) {  // part 1
            if (str[x + lcp] < str[y + lcp]) return 1;
            else return 0;
        }
        lcp = get_LCP(rk[x + (N - y + 1)], rk[x]);
        if (str[x + (N - y + 1) + lcp] < str[x + lcp]) return 1;
        else return 0;
    } else {
        int lcp = get_LCP(rk[x], rk[y]);
        if (lcp < N - x+ 1) {  // part 1
            if (str[x + lcp] < str[y + lcp]) return 1;
            else return 0;
        }
        lcp = get_LCP(rk[y + (N - x + 1)], rk[y]);
        if (str[y + lcp] < str[y+ (N - x + 1) + lcp]) return 1;
        else return 0;
    }
}
int main() {
    scanf("%d", &N); scanf("%s", str + 1);
    SA::run(str, N); SA::get_height(str);
    RMQ::init(N);
    for (int i = 1; i <= N; i++) num[i] = i;
    sort(num+1, num+1+N, cmp);
    for (int i = 1; i <= N; i++) {
        printf("%d\n", num[i]);
    }
}
```

4.8 后缀自动机 SAM

4.8.1 后缀自动机板子

应用 1：不同子串个数

给一个字符串 S，计算不同子串的个数。

解法：利用后缀自动机的树形结构。每个节点对应的不同子串数量 (不同位置算作同一个) 是 $maxlen[i] - maxlen[link[i]]$。

总时间复杂度：$O(|S|)$.

应用 2：所有不同子串的总长度

给定一个字符串 S，计算所有不同子串的总长度。解法：利用上述后缀自动机的树形结构。每个节点对应的所有后缀长度是 $\frac{maxlen[i]*(maxlen[i]+1)}{2}$，减去其 $linke$ 节点的对应值 $\frac{maxlen[link[i]]*(maxlen[link[i]]+1)}{2}$ 就是该节点的净贡献

```
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];  // 用于统计某一串出现的次数

    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            // 注意：这里返回的这个节点保存了多个模式串的状态，
            // 即将多个不同模式串的相同子串信息压缩在了这一个节点内，
            // 如果要记录endpos大小的话，
            // 则需要给每个模式串都单独维护一个siz数组依次更新，
            // 而不能全部揉成一坨
            if (maxlen[p] + 1 == maxlen[x]) {   // 特判1：这个节点已经存在于SAM中
                val[x]++;   // 统计在整颗字典树上出现次数
                return x;
            }
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                val[y]++;   // 统计在整颗字典树上出现次数
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z] = 1; // 统计在整颗字典树上出现次数
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
```

```
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        ans_1 += maxlen[z] - maxlen[link[z]]; // 【应用1】统计不同字串个数
        return z;
    }

    struct Edge {
        int to, nex;
    } e[MAXN << 1];
    int head[MAXN], tol;

    void addEdge(int u, int v) {
        e[tol].to = v; e[tol].nex = head[u]; head[u] = tol; tol++;
    }


/*  统计出现次数为k的字串个数
    intput          output
    2               (输入组数)
    2               6
    abcabc
    3               9
    abcabcabcabc
*/
    ll ans = 0;
    void dfs(int u, int k) {
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            dfs(v, k);
            val[u] += val[v];
        }
        if (val[u] == k) {  // val为出现次数
            ans += 1ll * (maxlen[u] - maxlen[link[u]]); // 以当前状态st为结尾的长度
        }
    }

    int build(int k) {
        tol = 0;
        for (int i = 0; i <= rt; i++) head[i] = -1;
        for (int i = 2; i <= rt; i++) addEdge(link[i], i);  // 建fail树
        dfs(1, k);
    }
} sa;
```

4.8.2 每个子串在多少个主串中出现过（SPOJ8093）

暴力跳 Link 链.
时间复杂度：均摊$O(\sum|S|\sqrt{\sum|S|})$

```
/*
```

```
    input       output
    3 3
    abcabcabc
    aaa
    aafe
    abc         1
    a           3
    ca          1
*/
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];

    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) {   // 特判1：这个节点已经存在于SAM中
                return x;
            } else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
            }
        }
        int z = ++rt, p = last;
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        return z;
    }
} sa;

```

```
char str_sum[MAXN];
char str[MAXN];
int len[MAXN];

int las[MAXN];  // 记得清空las数组

inline void update(int x, int id) { // 暴力跳Link链
    for (; x && las[x] != id; x = sa.link[x]) {
        sa.val[x]++;
        las[x] = id;
    }
}

int main() {
   int n, q;
   scanf("%d%d", &n, &q);
   int tot = 0;
   for (int i = 1; i <= n; i++) {
       scanf("%s", str + 1);
       int last = 1;
       len[i] = strlen(str + 1);
       for (int j = 1; j <= len[i]; j++) {
           str_sum[++tot] = str[j];
           last = sa.insert(str[j] - 'a', last);
       }
   }
   sa.debug();
   tot = 0;
   for (int i = 1; i <= n; i++) {
       for (int j = 1, x = 1; j <= len[i]; j++) {
           update(x = sa.trans[x][str_sum[++tot] - 'a'], i);
       }
   }
   while (q--) {
       scanf("%s", str + 1);
       len[0] = strlen(str + 1);
       int flag = 1, u = 1;
       for (int i = 1; i <= len[0]; i++) {
           int ch = str[i]- 'a';
           if (sa.trans[u][ch]) {
               u = sa.trans[u][ch];
           } else {
               flag = 0; break;
           }
       }
       if (flag) printf("%d\n", sa.val[u]);
       else printf("0\n");
   }
}
```

4.8.3 第 k 小字串

```
/*
    input   output
```

```
    aabc    aab
    0 3
    aabc    aa
    1 3
    aabc    -1
    1 11
*/
const int MAXN = 1e6 + 5;
const int MAXC = 26;

class Suffix_Automaton { public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];

    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[0], 0, sizeof(trans[0]));
    }
    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) return x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z] = 1; // dfs树统计出现次数
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        return z;
    }

    int sz1[MAXN], sz2[MAXN];
    int topo[MAXN], topo_id[MAXN];

```

```
    // Graph
    struct Edge {
        int to, nex;
    } e[MAXN];
    int head[MAXN], tol;

    void addEdge(int u, int v) {
        e[tol].to = v;
        e[tol].nex = head[u];
        head[u] = tol;
        tol++;
    }

    void dfs(int u) {
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            dfs(v);
            val[u] += val[v];
        }
    }

    void build() {
        // get topo index
        for (int i = 1; i <= rt; i++) topo[maxlen[i]]++;
        for (int i = 1; i <= rt; i++) topo[i] += topo[i - 1];
        for (int i = 1; i <= rt; i++) topo_id[topo[maxlen[i]]--] = i;
        // when t = 0
        for (int i = rt; i >= 1; i--) {
            sz1[topo_id[i]] = 1;
            if (topo_id[i] == 1) sz1[topo_id[i]] = 0;
            for (int j = 0; j < MAXC; j++) {
                int v = trans[topo_id[i]][j];
                if (!v) continue;
                sz1[topo_id[i]] += sz1[v];
            }
        }
        // fail tree build begin
        for (int i = 0; i <= rt; i++) head[i] = -1;
        tol = 0;
        for (int i = 2; i <= rt; i++) addEdge(link[i], i);
        // fail tree build end
        dfs(1); // dfs val
        // when t = 1
        for (int i = rt; i >= 1; i--) {
            sz2[topo_id[i]] = val[topo_id[i]];
            if (topo_id[i] == 1) sz2[topo_id[i]] = 0;
            for (int j = 0; j < MAXC; j++) {
                int v = trans[topo_id[i]][j];
                if (!v) continue;
                sz2[topo_id[i]] += sz2[v];
            }
        }
    }

    void query0(int k) {    // 不同位置的相同子串算作一个
        if (sz1[1] < k) {
```

```
            printf("-1\n"); return ;
        }
        int u = 1;
        while (k) {
            for (int i = 0; i < MAXC; i++) {
                if (trans[u][i]) {
                    if (sz1[trans[u][i]] >= k) {
                        printf("%c", 'a' + i);
                        u = trans[u][i];
                        k--;
                        break;
                    } else k -= sz1[trans[u][i]];
                }
            }
        }
        printf("\n");
    }

    void query1(int k) {    // 不同位置的相同子串算作多个
        if (sz2[1] < k) {
            printf("-1\n"); return ;
        }
        int u = 1;
        while(k > 0) {
            for (int i = 0; i < MAXC; i++) {
                if (trans[u][i]) {
                    int v = trans[u][i];
                    if (sz2[v] >= k) {
                        printf("%c", 'a'+i);
                        u = v;
                        k -= val[v];
                        break;
                    } else k -= sz2[v];
                }
            }
        }
        printf("\n");
    }
} sa;
```

4.8.4 字典树建后缀自动机

```
void bfs() {    // 传说常数小
    queue<node> q;
    q.push(node(1, 1));
    int last = 1;
    while (!q.empty()) {
        node u = q.front(); q.pop();
        int nls = sa.insert(str[u.v]-'A', u.last);
        pos[u.v] = nls;
        for (int i = head[u.v]; ~i; i = e[i].nex) {
            int to = e[i].to;
            q.push(node(to, nls));
        }
```

```
    }
}

void dfs(int u, int last = 1) { // 传说常数大
    pos[u] = last = sa.insert(str[u] - 'A', last);
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        dfs(v, last);
    }
}
```

4.8.5 暴力在线统计出现次数为 k 次的字符串个数（HDU4641）

```
const int MAXN = 6e5 + 5;
const int MAXC = 26;
int K;
class Suffix_Automaton {
public:
    int rt, link[MAXN], maxlen[MAXN], trans[MAXN][MAXC];
    int val[MAXN];
    int ans = 0;
    void init() {
        rt = 1;
        link[1] = maxlen[1] = 0;
        memset(trans[1], 0, sizeof(trans[1]));
        ans = 0;
    }

    Suffix_Automaton() { init(); }

    inline int insert(int ch, int last) {   // main: last = 1
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) return x;
            else {
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                val[y] = val[x];
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[x] = y;
                return y;
            }
        }
        int z = ++rt, p = last;
        val[z]=0;
        memset(trans[z], 0, sizeof(trans[z]));
        maxlen[z] = maxlen[last] + 1;
        while (p && !trans[p][ch]) trans[p][ch] = z, p = link[p];
        if (!p) link[z] = 1;
        else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) link[z] = x;
            else {
```

```
                int y = ++rt;
                maxlen[y] = maxlen[p] + 1;
                val[y] = val[x];
                for (int i = 0; i < MAXC; i++) trans[y][i] = trans[x][i];
                while (p && trans[p][ch] == x) trans[p][ch] = y, p = link[p];
                link[y] = link[x], link[z] = link[x] = y;
            }
        }
        int t = z;
        while (t && val[t] < K) {
            val[t]++;
            if (val[t] >= K) ans+=maxlen[t] - maxlen[link[t]];
            t = link[t];
        }
        return z;
    }

} sa;
char str[MAXN];
int main() {
    int n, m;
    while(~scanf("%d%d%d", &n, &m, &K)) {
        sa.init();
        scanf("%s", str + 1);
        int last = 1;
        for (int i = 1; i <= n; i++) last = sa.insert(str[i] - 'a', last);
        while (m--) {
            int op;
            scanf("%d", &op);
            if (op == 1) {
                getchar();
                char ch;
                scanf("%c", &ch);
                last = sa.insert(ch - 'a', last);
            } else {
                printf("%d\n", sa.ans);
            }
        }
    }
}
```

4.9 Manacher

```
/* H[1] <= H[2] <= H[3] .... <= H[mid]  */
void Manacher(int s[], int len, int A[], int B[]) {
    int l = 0;
    A[l++] = -1;
    A[l++] = 0;
    for (int i = 0; i < len; i++) {
        A[l++] = s[i];
        A[l++] = 0;
    }
    A[l] = 0;
    int mx = 0, id = 0;
```

```
    for (int i = 0; i < l; i++) {
        B[i] = mx > i ? min(B[2*id-i], mx-i) : 1;
        while (A[i+B[i]] == A[i-B[i]]) {
            if (A[i-B[i]] != 0 && A[i-B[i]] > A[i-B[i]+2]) break;
            B[i]++;
        }
        if (i + B[i] > mx) mx = i+ B[i], id = i;
    }
}
```

4.10 回文自动机 PAM

4.10.1 回文自动机 PAM

```
int pos[MAXN];  // +1后对应为所建的fail树上的点
class Palindrome_Tree { public:
    struct node {
        int ch[MAXC], fail, len, num;   // num：以该位置结尾的回文子串个数
    } T[MAXN];
    int las, tot, c[MAXN];
    inline int get_fail(int x, int pos) {
        while (c[pos - T[x].len - 1] != c[pos]) {
            x = T[x].fail;
        }
        return x;
    }
    void init() { // 传入字符串长度
        memset(T[0].ch, 0, sizeof(T[0].ch));
        memset(T[1].ch, 0, sizeof(T[1].ch));
        T[0].len = 0, T[1].len = -1;
        T[0].fail = 1, T[1].fail = 0;
        las = 0, tot = 1;
    }
    void insert(char s[], int len) {  // call: insert(str, strlen(str+1));
        c[0] = -1;
        for (int i = 1; i <= len; i++) {
            c[i] = s[i] - 'a';
            int p = get_fail(las, i);
            if (!T[p].ch[c[i]]) {
                T[++tot].len = T[p].len + 2;
                memset(T[tot].ch, 0, sizeof(T[tot].ch));
                int u = get_fail(T[p].fail, i);
                T[tot].fail = T[u].ch[c[i]];
                T[tol].num = T[T[tol].fail].num + 1;
                T[p].ch[c[i]] = tot;
            }
            las = T[p].ch[c[i]];    // 一般是对las指向进行操作和计数
            pos[i] = las;
        }
    }
    struct Edge {
        int to, nex;
    } e[MAXN << 1];
    int head[MAXN], tol;
```

```
    int len[MAXN];
    void addEdge(int u, int v) {
        e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
    }
    void build() { // build tree
        tol = 0;
        for (int i = 0; i <= tot; i++) head[i+1] = -1;
        for (int i = 1; i <= tot; i++) addEdge(T[i].fail + 1, i + 1);
        for (int i = 0; i <= tot; i++) len[i + 1] = T[i].len;
    }
} tree;

char str[MAXN];
int main() {
    scanf("%s", str + 1);
    tree.init();
    tree.insert(str, strlen(str + 1));
    tree.build();
    for (int i = 1; i <= len; i++) printf("%d ", pos[i]+1); // fail树上对应位置
}
```

4.10.2 前向星 PAM

给定一个长度为 n 的小写字母串。问你有多少对相交的回文子串（包含也算相交）。

```
/*  [input]     [output]
    4           6
    babb
    2           2
    aa
*/
char str[MAXN];
ll num[MAXN][2];
class PAM { public:
    struct node {
        int fail, len;
        ll num;
    } T[MAXN];
    int las, tot;
    struct Edge {
        int to, w, nex;
    } e[MAXN << 1];
    int head[MAXN], tol;
    void addEdge(int u, int v, int w) { e[tol].to = v, e[tol].w = w, e[tol].nex = head[u],
        head[u] = tol, tol++; }
    inline int get_fail(int x, int pos) {
        while (str[pos - T[x].len - 1] != str[pos]) x = T[x].fail;
        return x;
    }
    void init() {
        head[0] = head[1] = -1;
        T[0].fail = 1, T[1].fail = 0;
        T[0].len = 0, T[1].len = -1;
        T[0].num = 0, T[1].num = 0;
        las = 0, tot = 1;
        tol = 0;
```

```
    }
    void insert1(char s[], int len) {
        s[0] = 26;
        for (int i = 1; i <= len; i++) {
            int p = get_fail(las, i);
            int flag = 0;
            for (int j = head[p]; ~j; j = e[j].nex) {
                if (e[j].w == s[i] - 'a') {
                    flag = e[j].to;
                    break;
                }
            }
            if (!flag) {
                T[++tot].len = T[p].len + 2;
                head[tot] = -1;
                int u = get_fail(T[p].fail, i);
                int fg = 0;
                for (int j = head[u]; ~j; j = e[j].nex) {
                    if (e[j].w == str[i] - 'a') {
                        fg = e[j].to;
                        break;
                    }
                }
                T[tot].fail = fg;
                T[tot].num = T[T[tot].fail].num + 1;
                addEdge(p, tot, str[i] - 'a');
                flag = tot;
            }
            las = flag;
            num[i][0] = T[las].num;
        }
    }
    void insert2(char s[], int len) {
        s[0] = 26;
        for (int i = 1; i <= len; i++) {
            int p = get_fail(las, i);
            int flag = 0;
            for (int j = head[p]; ~j; j = e[j].nex) {
                if (e[j].w == s[i] - 'a') {
                    flag = e[j].to;
                    break;
                }
            }
            if (!flag) {
                T[++tot].len = T[p].len + 2;
                head[tot] = -1;
                int u = get_fail(T[p].fail, i);
                int fg = 0;
                for (int j = head[u]; ~j; j = e[j].nex) {
                    if (e[j].w == str[i] - 'a') {
                        fg = e[j].to;
                        break;
                    }
                }
                T[tot].fail = fg;
                T[tot].num = T[T[tot].fail].num + 1;
```

```
                addEdge(p, tot, str[i] - 'a');
                flag = tot;
            }
            las = flag;
            num[len - i + 1][1] = T[las].num;
        }
    }
} tree;
int main() {
    int n; scanf("%d", &n);
    scanf("%s", str + 1);
    tree.init();
    tree.insert1(str, n);
    int n2 = n >> 1;
    for (int i = 1; i <= n2; i++) swap(str[i], str[n - i + 1]);
    tree.init();
    tree.insert2(str, n);
    for (int i = n - 1; i >= 1; i--) {
        num[i][1] += num[i + 1][1];
        if (num[i][1] >= mod) num[i][1] -= mod;
    }
    ll res1 = 0;
    for (int i = 1; i <= n; i++) {
        res1 += num[i][0] * num[i + 1][1] % mod;
    }
    ll res2 = (ll) (num[1][1]) * (num[1][1] - 1) / 2 % mod;
    ll res = res2 - res1;
    if (res < 0) res += ((0ll - res) / mod + 1) * mod;
    printf("%lld\n", res % mod);
}
```

4.10.3 前后插入 PAM

```
const int MAXN = 1e5 + 5;
const int base = MAXN;
int L, R;
class PAM { public:
    struct node {
        int ch[26];
        int len, fail, num;
    } T[MAXN];
    int pre, nex, tot;
    ll sum;
    void init() {
        memset(T[0].ch, 0, sizeof(T[0].ch));
        memset(T[1].ch, 0, sizeof(T[1].ch));
        T[0].len = 0, T[1].len = -1;
        T[0].fail = 1, T[1].fail = 0;
        tot = 1;
        pre = nex = 0;
        sum = 0;
    }
    void pre_insert(char s[], int i) {
        while (s[i + T[pre].len + 1] != s[i]) pre = T[pre].fail;
```

```
        if (!T[pre].ch[s[i] - 'a']) {
            T[++tot].len = T[pre].len + 2;
            memset(T[tot].ch, 0, sizeof(T[tot].ch));
            T[tot].fail = T[tot].num = 0;
            int j = T[pre].fail;
            while (s[i + T[j].len + 1] != s[i]) j = T[j].fail;
            T[tot].fail = T[j].ch[s[i] - 'a'];
            T[pre].ch[s[i] - 'a'] = tot;
            T[tot].num = T[T[tot].fail].num + 1;
        }
        pre = T[pre].ch[s[i] - 'a'];
        if (T[pre].len == R - L + 1) nex = pre;
        sum = sum + T[pre].num;
    }
    void bac_insert(char s[], int i) {
        while (s[i - T[nex].len - 1] != s[i]) nex = T[nex].fail;
        if (!T[nex].ch[s[i] - 'a']) {
            T[++tot].len = T[nex].len + 2;
            memset(T[tot].ch, 0, sizeof(T[tot].ch));
            T[tot].fail = T[tot].num = 0;
            int j = T[nex].fail;
            while (s[i - T[j].len - 1] != s[i]) j = T[j].fail;
            T[tot].fail = T[j].ch[s[i] - 'a'];
            T[nex].ch[s[i] - 'a'] = tot;
            T[tot].num = T[T[tot].fail].num + 1;
        }
        nex = T[nex].ch[s[i] - 'a'];
        if (T[nex].len == R - L + 1) pre = nex;
        sum = sum + T[nex].num;
    }
} tree;
char str[MAXN<<1]; // 两倍空间注意!
int main() {
    int n;
    while (~scanf("%d", &n)) {
        tree.init();
        L = base + 1, R = base;
        memset(str, 0,  sizeof(str));
        while (n--) {
            int opt; scanf("%d", &opt);
            if (opt == 1) { // 向前面插入
                char s[2]; scanf("%s", s);
                str[--L] = s[0];
                tree.pre_insert(str, L);
            } else if (opt == 2) { // 向后面插入
                char s[2]; scanf("%s", s);
                str[++R] = s[0];
                tree.bac_insert(str, R);
            } else if (opt == 3) { // 统计不同回文串个数
                printf("%d\n", tree.tot - 1);
            } else printf("%lld\n", tree.sum); // 统计回文串个数
        }
    }
}
```

4.11 序列自动机（[HEOI2015] 最短不公共子串）

时间复杂度：$O(n|\sum|)$，其中 $|\sum|$ 为字符集大小

```
/*
    input       output
    aabbcc      2 4 2 4
    abcabc
    aabbcc      -1 -1 2 -1
    aabbcc
*/
char str1[MAXN], str2[MAXN];
int nex[MAXC], na[MAXN][MAXC], nb[MAXN][MAXC];
int dp[MAXN][MAXN]; // 记得开大两倍
int main() {
    scanf("%s%s", str1 + 1, str2 + 1);
    int len1 = strlen(str1 + 1), len2 = strlen(str2 + 1);
    int last = 1;
    for (int i = 1; i <= len2; i++) last = sa.insert(str2[i] - 'a', last);  // 调用后缀自动机
    for (int i = 0; i < 26; i++) nex[i] = len1 + 1;
    for (int i = len1; i >= 0; i--) {
        memcpy(na[i], nex, sizeof(nex));
        nex[str1[i] - 'a'] = i;
    }
    for (int i = 0; i < 26; i++) nex[i] = len2 + 1;
    for (int i = len2; i >= 0; i--) {
        memcpy(nb[i], nex, sizeof(nex));
        nex[str2[i] - 'a'] = i;
    }
    int res = MAXN;
    for (int l = 1; l <= len1; l++) {
        for (int r = l, u = 1; r <= len1; r++) {
            u = sa.trans[u][str1[r] - 'a'];
            if (!u) {
                res = min(res, r - l + 1);
                break;
            }
        }
    }
    printf("%d\n", res == MAXN ? -1 : res); // str1的一个最短的子串，它不是str2的子串。
    res = MAXN;
    for (int l = 1; l <= len1; l++) {
        for (int r = l, u = 0; r <= len1; r++) {
            u = nb[u][str1[r] - 'a'];
            if (u == len2 + 1) {
                res = min(res, r - l + 1);
                break;
            }
        }
    }
    printf("%d\n", res == MAXN ? -1 : res); // str1的一个最短的子串，它不是str2的子序列。
    for (int i = len1; i >= 0; i--) {
        for (int j = 1; j <= sa.rt; j++) {
            dp[i][j] = MAXN;
            for (int ch = 0; ch < MAXC; ch++) {
                int u = na[i][ch], v = sa.trans[j][ch];
```

```
                if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
            }
        }
    }
    printf("%d\n", dp[0][1] == MAXN ? -1 : dp[0][1]);   //
        str1的一个最短的子序列，它不是str2的子串。
    memset(dp, 0, sizeof(dp));
    for (int i = len1; i >= 0; i--) {
        for (int j = 0; j <= len2; j++) {
            dp[i][j] = MAXN;
            for (int ch = 0; ch < MAXC; ch++) {
                int u = na[i][ch], v = nb[j][ch];
                if (u <= len1) dp[i][j] = min(dp[i][j], dp[u][v] + 1);
            }
        }
    }
    printf("%d\n", dp[0][0] == MAXN ? -1 : dp[0][0]);   //
        str1的一个最短的子序列，它不是str2的子序列。
}
```

4.12 最小表示法

时间复杂度：$O(n)$

```
/*
    input
    10
    10 9 8 7 6 5 4 3 2 1
    output
    1 10 9 8 7 6 5 4 3 2
*/
int Min_show(int arr[], int n) {
    int i = 0, j = 1, k = 0;
    while (i < n && j < n && k < n) {
        if (arr[(i + k) % n] == arr[(j + k) % n]) k++;
        else {
            if (arr[(i + k) % n] > arr[(j + k) % n]) i += k + 1;
            else j += k + 1;
            if (i == j)i++;
            k = 0;
        }
    }
    return min(i, j);
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    int ans = Min_show(a,n);
    for (int i = 0; i < n; i++) printf("%d ", a[(i + ans) % n]);
}
```

4.13 Lyndon 分解

将字符串分成若干部分 $s = s_1 s_2 s_3 ... s_m$，使得每个 s_i 都是 $LyndonWord$。

$LyndonWord$：当且仅当 s 是其所有后缀中最小字符串。

```
/*
    input               output
    ababa               2 4 5
    ababa = ab + ab + a
    bbababaabaaabaaaab  1 2 4 6 9 13 18
    bbababaabaaabaaaab
    = b + b + ab + ab + aab + aaaab + aaaab
*/
vector<int> Lyndon_Arr; // 分解后的串的右端点
// call: Lyndon_Word(str, strlen(str+1));
void Lyndon_Word(char s[], int s_len) {
    int i = 1;
    while (i <= s_len) {
        int j = i, k = j + 1;
        while (k <= s_len && s[j] <= s[k]) {
            if (s[j] < s[k]) j = i;
            else j++;
            k++;
        }
        while (i <= j) {
            Lyndon_Arr.push_back(i-j+k-1);
            i += k - j;
        }
    }
}
```

5 数据结构

5.1 ST 表

```
const int MAXN = 5e4 + 5;   // n
const int MAXL = 22;
int f[MAXN][MAXL], g[MAXN][MAXL], two[MAXN];
int h[MAXN];
void pre(int n) {
    for (int i = 1; i <= n; i++) {
        f[i][0] = h[i]; g[i][0] = h[i]; // 读入h
    }
    two[1] = 0; two[2] = 1;
    for (int i = 3; i < MAXN; i++) {
        two[i] = two[i / 2] + 1;
    }
    for (int j = 1; j <= MAXL; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
        }
    for (int j = 1; j <= MAXL; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            g[i][j] = min(g[i][j - 1], g[i + (1 << (j - 1))][j - 1]);
    }
}

```

```
int get_max(int x, int y) {
    int s = two[y - x + 1];
    return max(f[x][s], f[y - (1 << s) + 1][s]);
}

int get_min(int x, int y) {
    int s = two[y - x + 1];
    return min(g[x][s], g[y - (1 << s) + 1][s]);
}
```

5.2 树状数组

5.2.1 lowbit 之和

```
/*
    HDU5975 有n个集合，第 i 个集合的数是[i-lowbit(i)+1,i-1]+i
    第一种方式是集合[a,b]共有多少个数=lowbit之和？(sol1)
    第二种方式是数字x在几个集合里面？(sol2)
    long long型注意！
*/
ll lowbit(ll x) {
    return x & (-x);
}

ll sum(ll x) {            //计算[1,x]中所有数lowbit(k)之和
    ll ans = 0ll;
    for (ll p = 1ll; p <= x; p <<= 1) {
        ans += (x / p - x / (p << 1)) * p;
    }
    return ans;
}

ll sol1(ll x, ll y) {        //计算[x,y]中所有树的lowbit(k)之和
    return sum(y) - sum(x - 1ll);
}

ll sol2(ll x, ll n) {
    ll res = 0ll;
    while (x <= n) {
        res++;
        x += lowbit(x);
    }
    return res;
}
```

5.2.2 区间加减 + 区间和查询

```
ll T1[SIZE], T2[SIZE], T[SIZE];//T[]用于存结点值，用过一次即仍
int lowbit(int k) {
    return k & -k;
}

void update(ll x, ll, N, ll w) {//把x位置之后所有数的值+w
```

```
    for (ll i = x; i <= N; i += lowbit(i)) {
        T1[i] += w;
        T2[i] += w * (x - 1);
    }
}

void range_update(ll l, ll r, ll v) {//在[l,r]上加v
    update(l, v);
    update(r + 1, -v);
}

ll sum(ll x) {
    ll ans = 0;
    for (ll i = x; i > 0; i -= lowbit(i)) {
        ans += x * T1[i] - T2[i];
    }
    return ans;
}

ll range_ask(ll l, ll r) {//返回[l,r]的和
    return sum(r) - sum(l - 1);
}

void init(int N) {
    for (int i = 1; i <= N; i++) {
        update(i, T[i] - T[i - 1]);
    }
}
```

5.2.3 统计前后顺序不同数字对个数（三维偏序问题）

```
ll arr[3][SIZE];//数据存放为[1,n]的范围
ll tree[SIZE];
int n;//n为每组数字个数
int lowbit(int k) {
    return k & -k;
}

void add(int x, int k) {
    while (x <= n) {
        tree[x] += k;
        x += lowbit(x);
    }
}

ll sum(int x) {
    ll ans = 0;
    while (x != 0) {
        ans += tree[x];
        x -= lowbit(x);
    }
    return ans;
}

```

```
int pos[SIZE];

ll CountInversions(int x, int y) {
    memset(tree, 0, sizeof(tree));
    for (int i = 1; i <= n; i++) {
        pos[arr[x][i]] = i;
    }
    ll ans = 0;
    for (int i = n; i; i--) {
        ans += sum(pos[arr[y][i]]);
        add(pos[arr[y][i]], 1);
    }
    return ans;
}

//求三组数中有多少对数的前后顺序在三组数中都相同(三维偏序问题)
ll invers = (CountInversions(0, 1) + CountInversions(1, 2) + CountInversions(2, 0)) / 2ll;
ll tot = ((ll) n * (ll) (n - 1)) / 2ll;//这里一定要加ll!!不然会爆
printf("%lld\n",tot-invers);
```

5.3 二维树状数组

5.3.1 单点修改 + 区间查询

```
class BIT { public:
    ll val[MAXN][MAXN];
    int n, m;
    void init(int _n, int _m) {
        n = _n, m = _m;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                val[i][j] = 0;
            }
        }
    }
    inline int lowbit(int x) { return x & (-x); }
    void add(int x, int y, ll v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= m; j += lowbit(j)) {
                val[i][j] += v;
            }
        }
    }
    inline ll query(int x, int y) {
        ll ans = 0;
        for (int i = x; i >= 1; i -= lowbit(i)) {
            for (int j = y; j >= 1; j -= lowbit(j)) {
                ans += val[i][j];
            }
        }
        return ans;
    }
    ll query(int x1, int y1, int x2, int y2) {  // x1 <= y1, x2 <= y2
        return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) + query(x1 - 1, y1 - 1);
```

```
    }
} tree;
```

5.3.2 区间修改 + 单点查询

```
class BIT { public:
    ll val[MAXN][MAXN];
    int n, m;
    void init(int _n, int _m);
    inline int lowbit(int x);
    inline void add(int x, int y, ll v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= m; j += lowbit(j)) {
                val[i][j] += v;
            }
        }
    }
    ll query(int x, int y) {
        ll ans = 0;
        for (int i = x; i >= 1; i -= lowbit(i)) {
            for (int j = y; j >= 1; j -= lowbit(j)) {
                ans += val[i][j];
            }
        }
        return ans;
    }
    void change(int x1, int y1, int x2, int y2, ll v) { // x1 <= y1, x2 <= y2
        add(x1, y1, v);
        add(x2 + 1, y2 + 1, v);
        add(x2 + 1, y1, -v);
        add(x1, y2 + 1, -v);
    }
} tree;
```

5.3.3 区间修改 + 区间查询

```
class BIT { public:
    ll val[MAXN][MAXN][4];
    int n, m;
    void init(int _n, int _m) {
        n = _n, m = _m;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                for (int k = 0; k < 4; k++) {
                    val[i][j][k] = 0;
                }
            }
        }
    }
    inline int lowbit(int x) { return x & (-x); }
    inline void add(int x, int y, ll v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= m; j += lowbit(j)) {
```

```
                val[i][j][0] += v, val[i][j][1] += v * x, val[i][j][2] += v * y, val[i][j][3]
                    += v * x * y;
            }
        }
    }
    void change(int x1, int y1, int x2, int y2, ll v) { // x1 <= y1, x2 <= y2
        add(x1, y1, v);
        add(x2 + 1, y2 + 1, v);
        add(x2 + 1, y1, -v);
        add(x1, y2 + 1, -v);
    }
    inline ll query(int x, int y) {
        ll ans = 0;
        for (int i = x; i >= 1; i -= lowbit(i)) {
            for (int j = y; j >= 1; j -= lowbit(j)) {
                ans += (ll) (x + 1) * (y + 1) * val[i][j][0] - (ll) (y + 1) * val[i][j][1] -
                       (ll) (x + 1) * val[i][j][2] + val[i][j][3];
            }
        }
        return ans;
    }
    ll sum(int x1, int y1, int x2, int y2) {  // x1 <= y1, x2 <= y2
        return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) + query(x1 - 1, y1 - 1);
    }
} tree;
```

5.4 线段树

5.4.1 单点修改 + 区间查询

```
#define lson l, mid, rt << 1
#define rson mid + 1, r, rt << 1 | 1
const int SIZE = 5e5+50;
struct node{
    int l,r;int val;
}T[SIZE*3];
int arr[SIZE];// 用于存从[1,n]中的数组
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l == r){
        T[rt].val = arr[l];
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    build(lson); build(rson);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
void update(int tar,int rt,int k){// 单点修改将tar改为k
    if(T[rt].l==T[rt].r){
        T[rt].val += k;return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(tar <= mid) update(tar,rt<<1,k);
    else update(tar,rt<<1|1,k);
```

```
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
int ans = 0;
void Find(int rt,int l,int r){
    if(T[rt].l>=l && T[rt].r<=r){
        ans += T[rt].val;return ;
    }
    if(T[rt<<1].r>=l) Find(rt<<1,l,r);
    if(T[rt<<1|1].l<=r) Find(rt<<1|1,l,r);
}
```

5.4.2 区间修改 + 区间查询

```
#define ll long long
const int SIZE = 1e5+5;
struct node{
    int l,r;int val;
}T[SIZE*3];
int lazy[SIZE*3];
int arr[SIZE];// arr数组用于记录[1,n]的数据
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l==r){
        T[rt].val = arr[l];return ;
    }
    int mid = (l+r)>>1;
    build(l,mid,rt<<1);
    build(mid+1,r,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
// l,r为指针，L,R是要修改的区间[L,R],为整个区间加上x
void update(int L,int R,int x,int rt){
    if(L<=T[rt].l && R>=T[rt].r){
        T[rt].val += x*(T[rt].r-T[rt].l+1);
        lazy[rt] += x;
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(lazy[rt]){ // 下推标记更新一波
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    if(L <= mid)update(L,R,x,rt<<1);
    if(R >mid)update(L,R,x,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}

ll query(int L,int R,int rt){
    // printf("rt=%d",rt);
    if(L<=T[rt].l && R>=T[rt].r)return T[rt].val;
    int mid = (T[rt].l + T[rt].r)>>1;
```

```
    if(lazy[rt]){
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    ll sum=0;
    if(L <= mid)sum += query(L,R,rt<<1);
    if(R > mid)sum += query(L,R,rt<<1|1);
    return sum;
}
```

5.4.3 区间染色

```
const int SIZE=1e5+50;
const int col_SIZE=1e5+50;
bool col_vis[col_SIZE];
struct node{
    int l,r;int col;
}T[SIZE*3];

void build(int root,int l,int r){
    T[root].l = l;
    T[root].r = r;
    if(l == r)return ;
    int mid=(T[root].l + T[root].r)>>1;
    build(root<<1,l,mid);
    build((root<<1)+1,mid+1,r);
}
void update(int root,int l,int r,int col){
    if(T[root].l>=l && T[root].r<=r){
        T[root].col = col;return ;
    }
    else{
        if(T[root].col > 0){
            T[root<<1].col = T[root].col;
            T[(root<<1)+1].col = T[root].col;
            T[root].col = 0;
        }
        int mid = (T[root].l + T[root].r)>>1;
        if(l>mid){
            update((root<<1)+1,l,r,col);
        }
        else if(r<=mid){
            update(root<<1,l,r,col);
        }
        else{
            update(root<<1,l,mid,col);
            update((root<<1)+1,mid+1,r,col);
        }
    }
}

```

```
void Find(int root,int l,int r){
    if(T[root].l==0 || T[root].r==0)return ;
    if(T[root].col > 0 ){
        col_vis[T[root].col] = true;
    }
    else{
        int mid = (T[ root ].l + T[ root ].r ) >> 1;
        if(l>mid){
            Find((root<<1)+1,l,r);
        }
        else if(r<=mid){
            Find(root<<1,l,r);
        }
        else{
            Find(root<<1,l,mid);
            Find((root<<1)+1,mid+1,r);
        }
    }
}
void init(int N){//N为最大的颜色
    for(int i=0;i<=N;i++){
        col_vis[i]=0;
    }
}

int tot(int l,int r,int N){//统计[l,r]内颜色范围为[1,N]共有几种
    init(N);
    Find(1,l,r);
    int tot=0;
    for(int i=0;i<=N;i++){
        if(col_vis[i])tot++;
    }
    return tot;
}
```

5.4.4 区间修改 + 区间查询：矩阵

```
int n;
mat A(2, 2), B(2, 2);
const int SIZE = 1e5 + 50;
char str[SIZE];
struct node {
    int l, r;
    bool lazy = 0;
} T[SIZE * 3];
mat ma[SIZE * 3][2];//0是正序，1是反序
void push_up(int root) {
    ma[root][0] = ma[root << 1][0] * ma[(root << 1) | 1][0];
    ma[root][1] = ma[root << 1][1] * ma[(root << 1) | 1][1];
    return;
}

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

```
    T[root].r = r;
    if (l == r) {
        if (str[l] == 'A') {
            ma[root][0].n = A.n;
            ma[root][0].m = A.m;
            for (int i = 0; i < A.n; i++)
                for (int j = 0; j < A.m; j++)
                    ma[root][0].v[i][j] = A.v[i][j];
            //————————————————————————————
            ma[root][1].n = B.n;
            ma[root][1].m = B.m;
            for (int i = 0; i < B.n; i++)
                for (int j = 0; j < B.m; j++)
                    ma[root][1].v[i][j] = B.v[i][j];
        } else {
            ma[root][0].n = B.n;
            ma[root][0].m = B.m;
            for (int i = 0; i < B.n; i++)
                for (int j = 0; j < B.m; j++)
                    ma[root][0].v[i][j] = B.v[i][j];
            //————————————————————————————
            ma[root][1].n = A.n;
            ma[root][1].m = A.m;
            for (int i = 0; i < A.n; i++)
                for (int j = 0; j < A.m; j++)
                    ma[root][1].v[i][j] = A.v[i][j];
        }
        return;
    }
    int mid = (T[root].l + T[root].r) >> 1;
    build(root << 1, l, mid);
    build((root << 1) | 1, mid + 1, r);
    push_up(root);
    return;
}

void push_down(int root) {
    if (T[root].lazy) {
        swap(ma[root << 1][0], ma[root << 1][1]);
        swap(ma[(root << 1) | 1][0], ma[(root << 1) | 1][1]);
        T[root << 1].lazy = !T[root << 1].lazy;
        T[(root << 1) | 1].lazy = !T[(root << 1) | 1].lazy;
        T[root].lazy = 0;
    }
}

void update(int root, int l, int r) {
    if (l <= T[root].l && T[root].r <= r) {
        T[root].lazy = !T[root].lazy;
        swap(ma[root][0], ma[root][1]);
        return;
    }
    push_down(root);
    int mid = (T[root].l + T[root].r) >> 1;
    if (l <= mid)update(root << 1, l, r);
    if (r > mid)update((root << 1) | 1, l, r);
```

```
    push_up(root);
}

mat tmp(2, 2);// 对最后的答案矩阵进行修改
void find(int root, int l, int r) {
    if (l <= T[root].l && T[root].r <= r) {
        tmp = tmp * ma[root][0];
        return;
    }
    push_down(root);
    int mid = (T[root].l + T[root].r) >> 1;
    if (l <= mid)find(root << 1, l, r);
    if (mid < r)find((root << 1) | 1, l, r);
    return;
}

int main() {
    A.v[0][0] = 1, A.v[0][1] = 0, A.v[1][0] = 1, A.v[1][1] = 1;
    B.v[0][0] = 1, B.v[0][1] = 1, B.v[1][0] = 0, B.v[1][1] = 1;
    int q; scanf("%d%d", &n, &q);
    scanf("%s", str + 1);
    build(1, 1, n);
    while (q--) {
        int op; scanf("%d", &op);
        if (op == 1) {
            int L, R;
            scanf("%d%d", &L, &R);
            update(1, L, R);
        } else {
            int L, R;
            ll AA, BB;
            scanf("%d%d%lld%lld", &L, &R, &AA, &BB);
            tmp.v[0][0] = 1, tmp.v[0][1] = 0, tmp.v[1][0] = 0, tmp.v[1][1] = 1;
            mat qwq(1, 2);
            qwq.v[0][0] = AA % mod;
            qwq.v[0][1] = BB % mod;
            find(1, L, R);
            qwq = qwq * tmp;
            qwq.display();
        }
    }
}
```

5.4.5 区间中所有元素都严格出现三次的区间个数（CF1418G）

```
/*
    input               output
    9                   3
    1 2 2 2 1 1 2 2 2
    10                  0
    1 2 3 4 1 2 3 1 2 3
    12                  1
    1 2 3 4 3 4 2 1 3 4 2 1
*/
```

```cpp
int arr[MAXN];
vector<int> vec[MAXN];
struct node {
    int l, r, v;
    node(int _l = 0, int _r = 0, int _v = 0) : l(_l), r(_r), v(_v) {}
};

class SEG { public:
    struct node {
        int l, r, mx, cnt;
    } T[MAXN << 2];
    int lazy[MAXN << 2];

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r, T[rt].mx = 0, T[rt].cnt = r - l;
        if (l + 1 == r) {
            T[rt << 1].l = T[rt << 1].r = T[rt << 1].mx = T[rt << 1].cnt = 0;
            T[rt << 1 | 1].l = T[rt << 1 | 1].r = T[rt << 1 | 1].mx = T[rt << 1 | 1].cnt = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid, r);
    }

    inline void push_up(int rt) {
        if (T[rt << 1].mx > T[rt << 1 | 1].mx) T[rt].mx = T[rt << 1].mx, T[rt].cnt = T[rt << 1].cnt;
        else if (T[rt << 1].mx < T[rt << 1 | 1].mx) T[rt].mx = T[rt << 1 | 1].mx, T[rt].cnt = T[rt << 1 | 1].cnt;
        else T[rt].mx = T[rt << 1].mx, T[rt].cnt = T[rt << 1].cnt + T[rt << 1 | 1].cnt;
    }

    inline void push_down(int rt) {
        if (lazy[rt]) {
            T[rt << 1].mx += lazy[rt], lazy[rt << 1] += lazy[rt];
            T[rt << 1 | 1].mx += lazy[rt], lazy[rt << 1 | 1] += lazy[rt];
            lazy[rt] = 0;
        }
    }

    void update(int rt, int L, int R, int val) {
        if (L <= T[rt].l && R >= T[rt].r) {
            T[rt].mx += val, lazy[rt] += val;
            return;
        }
        if (L >= T[rt].r || R <= T[rt].l)  return ;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update(rt << 1, L, R, val);
        if (R >= mid) update(rt << 1 | 1, L, R, val);
        push_up(rt);
    }
} tree;

vector<node> event[MAXN];
```

```
void add(int l1, int l2, int r1, int r2) {
    event[l1].push_back(node(r1, r2, 1));
    event[l2].push_back(node(r1, r2, -1));
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
        vec[arr[i]].push_back(i);
    }
    for (int x = 1; x <= n; x++) {
        int sz = SZ(vec[x]);
        for (int i = 0; i <= sz; i++) {
            add(i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] + 1,
                i == 0 ? 1 : vec[x][i - 1] + 1, i == sz ? n + 2 : vec[x][i] + 1);
        }
        for (int i = 0; i <= sz - 3; i++) {
            add(i == 0 ? 1 : vec[x][i - 1] + 1, vec[x][i] + 1,
                vec[x][i + 2] + 1, i + 3 == sz ? n + 2 : vec[x][i + 3] + 1);
        }
    }
    tree.build(1, 1, n + 2);
    ll res = 0;
    for (int l = 1; l <= n; l++) {  // enum left
        for (auto e: event[l]) {
            int el = e.l, er = e.r, ev = e.v;
            tree.update(1, el, er, ev);
        }
        tree.update(1, l, l+1, -1);
        if (tree.T[1].mx == n) {
            res += tree.T[1].cnt;
        }
    }
    printf("%lld\n", res);
}
```

5.4.6 线段树分裂合并

```
class SEG { public:
    int ch[MAXM][2];
    ll sum[MAXM];
#define lson ch[rt][0]
#define rson ch[rt][1]
    int pool[MAXM], pool_cnt, cnt;
    int New() { // recycle
        return pool_cnt ? pool[pool_cnt--] : ++cnt;
    }
    void Del(int rt) {
        pool[++pool_cnt] = rt;
        ch[rt][0] = ch[rt][1] = sum[rt] = 0;
    }
    inline void push_up(int rt) {
```

```
        sum[rt] = sum[lson] + sum[rson];
    }
    int change(int rt, int pos, int v, int be, int en) {
        if (!rt) rt = New();
        if (be == en) {
            sum[rt] += (ll) v;
            return rt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) lson = change(lson, pos, v, be, mid);
        else
            rson = change(rson, pos, v, mid + 1, en);
        push_up(rt);
        return rt;
    }
    ll query_sum(int rt, int L, int R, int be, int en) {
        if (!rt) return 0;
        if (L <= be && R >= en) return sum[rt];
        int mid = (be + en) >> 1;
        ll ans = 0;
        if (L <= mid) ans += query_sum(lson, L, R, be, mid);
        if (R > mid) ans += query_sum(rson, L, R, mid + 1, en);
        return ans;
    }
    int Kth(int rt, int be, int en, int k) {
        if (be == en) return be;
        int mid = (be + en) >> 1;
        int ans = -1;
        if (sum[ch[rt][0]] >= k) ans = Kth(lson, be, mid, k);
        else ans = Kth(rson, mid + 1, en, k - sum[ch[rt][0]]);
        return ans;
    }
    int merge(int x, int y, int be, int en) {
        if (!x || !y) return x + y;
        if (be == en) {
            sum[x] += sum[y];
            return x;
        }
        int mid = (be + en) >> 1;
        ch[x][0] = merge(ch[x][0], ch[y][0], be, mid);
        ch[x][1] = merge(ch[x][1], ch[y][1], mid + 1, en);
        push_up(x);
        Del(y);
        return x;
    }
    void split(int &rt1, int &rt2, int L, int R, int be, int en) {
        if (en < L || R < be) return;
        if (!rt1) return;
        if (L <= be && en <= R) {
            rt2 = rt1, rt1 = 0;
            return;
        }
        if (!rt2) rt2 = New();
        int mid = (be + en) >> 1;
        split(ch[rt1][0], ch[rt2][0], L, R, be, mid);
        split(ch[rt1][1], ch[rt2][1], L, R, mid + 1, en);
```

```
        push_up(rt1), push_up(rt2);
    }
} tree;

int root[MAXN];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    int max_n_m = max(n, m);
    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        root[1] = tree.change(root[1], i, x, 1, max_n_m);
    }
    int id = 1;
    while (m--) {
        int opt;
        scanf("%d", &opt);
        if (opt == 0) { // 将可重集p中>=x且<=y的值放入一个新的可重集中
            int p, x, y;
            scanf("%d%d%d", &p, &x, &y);
            tree.split(root[p], root[++id], x, y, 1, max_n_m);
        } else if (opt == 1) {  // 将可重集t中的数放入可重集p，且清空可重集t
            int p, t;
            scanf("%d%d", &p, &t);  // 数据保证在此后的操作中不会出现可重集t
            root[p] = tree.merge(root[p], root[t], 1, max_n_m);
        } else if (opt == 2) {  // 在p这个可重集中加入x个数字q
            int p, x, q;
            scanf("%d%d%d", &p, &x, &q);
            root[p] = tree.change(root[p], q, x, 1, max_n_m);
        } else if (opt == 3) {  // 查询可重集p中大>=x且<=y的值的个数
            int p, x, y;
            scanf("%d%d%d", &p, &x, &y);
            printf("%lld\n", tree.query_sum(root[p], x, y, 1, max_n_m));
        } else {  // 查询在p这个可重集中第k小的数，不存在时输出-1
            int p, k;
            scanf("%d%d", &p, &k);
            if (tree.sum[root[p]] < k) {
                printf("-1\n");
            } else printf("%d\n", tree.Kth(root[p], 1, max_n_m, k));
        }
    }
}
```

5.4.7 单点修改 + 单点最大连通数（HDU1540）

```
class SEG { public:
    struct Node {
        int l, r;
        int pre, suf, len;
    } T[MAXN << 2];

    void Build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
```

```
        T[rt].pre = T[rt].suf = T[rt].len = r - l + 1;
        if (l == r) return;
        int mid = (l + r) >> 1;
        Build(rt << 1, l, mid);
        Build(rt << 1 | 1, mid + 1, r);
    }

    inline void push_up(int rt) {
        T[rt].pre = T[rt << 1].pre;
        T[rt].suf = T[rt << 1 | 1].suf;
        T[rt].len = max(T[rt << 1].len, T[rt << 1 | 1].len);
        T[rt].len = max(T[rt].len, T[rt << 1].suf + T[rt << 1 | 1].pre);
        if (T[rt << 1].pre == T[rt << 1].r - T[rt << 1].l + 1)T[rt].pre += T[rt << 1 | 1].pre;
        if (T[rt << 1 | 1].suf == T[rt << 1 | 1].r - T[rt << 1 | 1].l + 1)
            T[rt].suf += T[rt << 1].suf;
    }

    // call: tree.update(1, x, 0); 摧毁
    //       tree.update(1, x, 1); 修复

    void update(int rt, int pos, int val) {
        if (T[rt].l == T[rt].r) {
            T[rt].pre = T[rt].suf = T[rt].len = val;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid)update(rt << 1, pos, val);
        else update(rt << 1 | 1, pos, val);
        push_up(rt);
    }

    int query(int rt, int pos) {
        if (T[rt].l == T[rt].r || T[rt].len == 0 || T[rt].len == T[rt].r - T[rt].l + 1) {
            return T[rt].len;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) {
            if (pos >= T[rt << 1].r - T[rt << 1].suf + 1)
                return query(rt << 1, pos) + query(rt << 1 | 1, mid + 1);
            else return query(rt << 1, pos);
        } else {
            if (pos <= T[rt << 1 | 1].l + T[rt << 1 | 1].pre - 1)
                return query(rt << 1 | 1, pos) + query(rt << 1, mid);
            else return query(rt << 1 | 1, pos);
        }
    }
} tree;
```

5.4.8 找到最前的长度为 k 的序列

```
class SEG { public:
    struct node {
        int l, r;
        int len[2], pre[2], suf[2];
```

```
} T[MAXN << 2];
inline void push_up(int rt) {
    for (int i = 0; i < 2; i++) {
        T[rt].pre[i] = T[rt << 1].pre[i];
        T[rt].suf[i] = T[rt << 1 | 1].suf[i];
        T[rt].len[i] = max(T[rt << 1].len[i], T[rt << 1 | 1].len[i]);
        T[rt].len[i] = max(T[rt].len[i], T[rt << 1].suf[i] + T[rt << 1 | 1].pre[i]);
        if (T[rt << 1].pre[i] == T[rt << 1].r - T[rt << 1].l + 1) T[rt].pre[i] += T[rt <<
            1 | 1].pre[i];
        if (T[rt << 1 | 1].suf[i] == T[rt << 1 | 1].r - T[rt << 1 | 1].l + 1) T[rt].suf[i]
            += T[rt << 1].suf[i];
    }
}
inline void push_down(int rt) {
    for (int i = 0; i < 2; i++) {
        if (T[rt].len[i] == 0) {
            T[rt << 1].len[i] = T[rt << 1].pre[i] = T[rt << 1].suf[i] = 0;
            T[rt << 1 | 1].len[i] = T[rt << 1 | 1].pre[i] = T[rt << 1 | 1].suf[i] = 0;
        }
        if (T[rt].len[i] == T[rt].r - T[rt].l + 1) {
            T[rt << 1].len[i] = T[rt << 1].pre[i] = T[rt << 1].suf[i] = T[rt << 1].r -
                T[rt << 1].l + 1;
            T[rt << 1 | 1].len[i] = T[rt << 1 | 1].pre[i] = T[rt << 1 | 1].suf[i] =
                    T[rt << 1 | 1].r - T[rt << 1 | 1].l + 1;
        }
    }
}
void build(int rt, int l, int r) {
    T[rt].l = l, T[rt].r = r;
    if (l == r) {
        for (int i = 0; i < 2; i++) {
            T[rt].len[i] = T[rt].pre[i] = T[rt].suf[i] = 1;
        }
        return;
    }
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
    push_up(rt);
}
void study(int rt, int L, int R) {
    if (L <= T[rt].l && T[rt].r <= R) {
        for (int i = 0; i < 2; i++) {
            T[rt].len[i] = T[rt].pre[i] = T[rt].suf[i] = T[rt].r - T[rt].l + 1;
        }
        return;
    }
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    if (L <= mid) study(rt << 1, L, R);
    if (R > mid) study(rt << 1 | 1, L, R);
    push_up(rt);
}
void play(int rt, int L, int R, int op) {
    if (L <= T[rt].l && T[rt].r <= R) {
        T[rt].len[op] = T[rt].pre[op] = T[rt].suf[op] = 0;
        return;
```

```
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) play(rt << 1, L, R, op);
        if (R > mid) play(rt << 1 | 1, L, R, op);
        push_up(rt);
    }
    int query(int rt, int v, int op) {
        if (T[rt].l == T[rt].r) return T[rt].l;
        push_down(rt);
        int ans = -1;
        if (T[rt << 1].len[op] >= v)
            ans = query(rt << 1, v, op);
        if (ans == -1 && T[rt<<1].suf[op] + T[rt<<1|1].pre[op] >= v)
            ans = T[rt << 1].r - T[rt << 1].suf[op] + 1;
        if (ans == -1 && T[rt << 1 | 1].len[op] >= v)
            ans = query(rt << 1 | 1, v, op);
        return ans;
    }
} tree;

int main() {
    int T; scanf("%d", &T);
    int kass = 0;
    while (T--) {
        int n, m; scanf("%d%d", &n, &m);
        tree.build(1, 1, n);
        while (m--) {
            char opt[10]; scanf("%s", opt);
            if (opt[0] == 'N') {
                int x; scanf("%d", &x);
                int ans = tree.query(1, x, 1);
                if (ans == -1) ans = tree.query(1, x, 0);
                if (ans == -1) printf("wait for me\n");
                else {
                    printf("%d,don't put my gezi\n", ans);
                    tree.play(1, ans, ans + x - 1, 0);
                    tree.play(1, ans, ans + x - 1, 1);
                }
            } else if (opt[0] == 'D') {
                int x; scanf("%d", &x);
                int ans = tree.query(1, x, 1);
                if (ans == -1) printf("fly with yourself\n");
                else {
                    printf("%d,let's fly\n", ans);
                    tree.play(1, ans, ans + x - 1, 1);
                }
            } else {
                int l, r; scanf("%d%d", &l, &r);
                tree.study(1, l, r);
                printf("I am the hope of chinese chengxuyuan!!\n");
            }
        }
    }
}
```

5.4.9 加乘赋值线段树

1 x y c $a_i = a_i + c(x \le i \le y)$

2 x y c $a_i = a_i * c(x \le i \le y)$

3 x y c $a_i = c$

4 x y p 打印 $a_x^p + a_{x+1}^p + ... + a_y^p$，$1 \le p \le 3$.

```
class Seg_tree { public:
    struct node {
        int l, r; ll sum1, sum2, sum3;
    }T[SIZE<<2];
    ll add[SIZE<<2], set[SIZE<<2], mul[SIZE<<2];
    void push_up(int rt) {
        T[rt].sum1 = (T[rt<<1].sum1 + T[rt<<1|1].sum1) % MOD;
        T[rt].sum2 = (T[rt<<1].sum2 + T[rt<<1|1].sum2) % MOD;
        T[rt].sum3 = (T[rt<<1].sum3 + T[rt<<1|1].sum3) % MOD;
    }
    void build(int l, int r, int rt) {
        add[rt] = set[rt] = 0;
        mul[rt] = 1;
        T[rt].l = l, T[rt].r = r;
        if(l == r) {
            T[rt].sum1 = T[rt].sum2 = T[rt].sum3 = 0;
            return ;
        }
        int mid = (l + r) >> 1;
        build(l, mid, rt<<1); build(mid+1, r, rt<<1|1);
        push_up(rt);
    }
    void push_down(int rt, int len) {
        if (set[rt]) {
            set[rt<<1] = set[rt<<1|1] = set[rt];
            add[rt<<1] = add[rt<<1|1] = 0;
            mul[rt<<1] = mul[rt<<1|1] = 1;
            ll tmp = ((set[rt] * set[rt]) % MOD) * set[rt] % MOD;
            T[rt<<1].sum1 = ((len - (len>>1)) % MOD) * (set[rt] % MOD) % MOD;
            T[rt<<1|1].sum1 = ((len>>1) % MOD) * (set[rt] % MOD) % MOD;
            T[rt<<1].sum2 = ((len - (len>>1)) % MOD) * ((set[rt] * set[rt]) % MOD) % MOD;
            T[rt<<1|1].sum2 = ((len>>1) % MOD) * ((set[rt] * set[rt]) % MOD) % MOD;
            T[rt<<1].sum3 = ((len - (len>>1)) % MOD) * tmp % MOD;
            T[rt<<1|1].sum3 = ((len>>1) % MOD) * tmp % MOD;
            set[rt] = 0;
        }
        if (mul[rt] != 1) {
            mul[rt<<1] = (mul[rt<<1] * mul[rt]) % MOD;
            mul[rt<<1|1] = (mul[rt<<1|1] * mul[rt]) % MOD;
            if(add[rt<<1]) add[rt<<1] = (add[rt<<1] * mul[rt]) % MOD;
            if(add[rt<<1|1]) add[rt<<1|1] = (add[rt<<1|1] * mul[rt]) % MOD;
            ll tmp = ((mul[rt] * mul[rt]) % MOD * mul[rt]) % MOD;
            T[rt<<1].sum1 = (T[rt<<1].sum1 * mul[rt]) % MOD;
            T[rt<<1|1].sum1 = (T[rt<<1|1].sum1 * mul[rt]) % MOD;
            T[rt<<1].sum2 = (T[rt<<1].sum2 % MOD) * ((mul[rt] * mul[rt]) % MOD) % MOD;
            T[rt<<1|1].sum2 = (T[rt<<1|1].sum2 % MOD) * ((mul[rt] * mul[rt]) % MOD) % MOD;
            T[rt<<1].sum3 = (T[rt<<1].sum3 % MOD) * tmp % MOD;
            T[rt<<1|1].sum3 = (T[rt<<1|1].sum3 % MOD) * tmp % MOD;
            mul[rt] = 1;
```

```
        }
        if (add[rt]) {
            add[rt<<1] += add[rt];
            add[rt<<1|1] += add[rt];
            ll tmp = (add[rt] * add[rt] % MOD) * add[rt] % MOD;
            T[rt<<1].sum3 = (T[rt<<1].sum3 + (tmp * (len - (len>>1)) % MOD) + 3 * add[rt] *
                ((T[rt<<1].sum2 + T[rt<<1].sum1 * add[rt] % MOD)) % MOD);
            T[rt<<1|1].sum3 = (T[rt<<1|1].sum3 + (tmp * (len>>1) % MOD) + 3 * add[rt] *
                ((T[rt<<1|1].sum2 + T[rt<<1|1].sum1 * add[rt] % MOD)) % MOD);
            T[rt<<1].sum2 = (T[rt<<1].sum2 + ((add[rt] * add[rt] % MOD) * (len - (len>>1)) %
                MOD) + (2 * T[rt<<1].sum1 * add[rt] % MOD)) % MOD;
            T[rt<<1|1].sum2 = (T[rt<<1|1].sum2 + ((add[rt] * add[rt] % MOD) * (len>>1) % MOD)
                + (2 * T[rt<<1|1].sum1 * add[rt] % MOD)) % MOD;
            T[rt<<1].sum1 = (T[rt<<1].sum1 + (len - (len>>1)) * add[rt]) % MOD;
            T[rt<<1|1].sum1 = (T[rt<<1|1].sum1 + (len>>1) * add[rt]) % MOD;
            add[rt] = 0;
        }
    }
    void update(int l, int r, int op, int rt, int c) {
        if(l <= T[rt].l && T[rt].r <= r) {
            if(op == 3) {
                set[rt] = c;
                add[rt] = 0;
                mul[rt] = 1;
                T[rt].sum1 = ((T[rt].r - T[rt].l + 1) * c) % MOD;
                T[rt].sum2 = ((T[rt].r - T[rt].l + 1) * ((c * c) % MOD)) % MOD;
                T[rt].sum3 = ((T[rt].r - T[rt].l + 1) * (((c * c) % MOD) * c % MOD)) % MOD;
            }
            else if(op == 2) {
                mul[rt] = (mul[rt] * c) % MOD;
                if(add[rt]) add[rt] = (add[rt] * c) % MOD;
                T[rt].sum1 = (T[rt].sum1 * c) % MOD;
                T[rt].sum2 = (T[rt].sum2 * (c * c % MOD)) % MOD;
                T[rt].sum3 = (T[rt].sum3 * ((c * c % MOD) * c % MOD)) % MOD;
            }
            else if(op == 1){
                add[rt] += c;
                ll tmp = (((c * c) % MOD * c) % MOD * (T[rt].r - T[rt].l + 1)) % MOD;
                T[rt].sum3 = (T[rt].sum3 + tmp + 3 * c * ((T[rt].sum2 + T[rt].sum1 * c) %
                    MOD)) % MOD;
                T[rt].sum2 = (T[rt].sum2 + (c * c % MOD * (T[rt].r - T[rt].l + 1) % MOD) + 2 *
                    T[rt].sum1 * c) % MOD;
                T[rt].sum1 = (T[rt].sum1 + (T[rt].r - T[rt].l + 1) * c) % MOD;
            }
            return ;
        }
        push_down(rt, T[rt].r - T[rt].l + 1);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (l > mid) update(l, r, op, rt<<1|1, c);
        else if(r <= mid) update(l, r, op, rt<<1, c);
        else {
            update(l, r, op, rt<<1, c); update(l, r, op, rt<<1|1, c);
        }
        push_up(rt);
    }
    ll query(int l, int r, int op, int rt) {
```

```
        if (l <= T[rt].l && T[rt].r <= r) {
            if (op == 1) return T[rt].sum1 % MOD;
            else if(op == 2) return T[rt].sum2 % MOD;
            else return T[rt].sum3 % MOD;
        }
        push_down(rt, T[rt].r - T[rt].l + 1);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if(l > mid) return query(l, r, op, rt<<1|1);
        else if(r <= mid) return query(l, r, op, rt<<1);
        else return (query(l, r, op, rt<<1) + query(l, r, op, rt<<1|1)) % MOD;
    }
}tree;
int main() {
    int n, m;
    while (~scanf("%d %d", &n, &m)) {
        if(n == 0 && m == 0) break;
        tree.build(1, n, 1);
        while (m--) {
            int op, a, b, c; scanf("%d %d %d %d", &op, &a, &b, &c);
            if(op == 4) printf("%lld\n", tree.query(a, b, c, 1));
            else tree.update(a, b, op, 1, c);
        }
    }
}
```

5.5 二维线段树

```
class SEG2D {   public:
    int n;
    int maxx[MAXN << 2][MAXN << 2], minn[MAXN << 2][MAXN << 2];
    void init(int _n) { n = _n; }
    inline void subPush_up(int frt, int rt) {
        maxx[frt][rt] = max(maxx[frt][rt << 1], maxx[frt][rt << 1 | 1]);
        minn[frt][rt] = min(minn[frt][rt << 1], minn[frt][rt << 1 | 1]);
    }
    inline void push_up(int frt, int rt) {
        maxx[frt][rt] = max(maxx[frt << 1][rt], maxx[frt << 1 | 1][rt]);
        minn[frt][rt] = min(minn[frt << 1][rt], minn[frt << 1 | 1][rt]);
    }

    inline void subBuild(int frt, int fl, int fr, int rt, int l, int r) {
        if (l == r) {
            if (fl == fr) maxx[frt][rt] = minn[frt][rt] = a[fl][l];
            else push_up(frt, rt);
            return;
        }
        int mid = (l + r) >> 1;
        subBuild(frt, fl, fr, rt << 1, l, mid), subBuild(frt, fl, fr, rt << 1 | 1, mid + 1, r);
        subPush_up(frt, rt);
    }
    void build(int rt, int l, int r) {
        if (l == r) {
            subBuild(rt, l, r, 1, 1, n);
            return;
```

```
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        subBuild(rt, l, r, 1, 1, n);
    }

    inline void subUpdate(int frt, int fl, int fr, int rt, int x, int y, int v, int be, int
        en) {
        if (be == en) {
            if (fl == fr) maxx[frt][rt] = minn[frt][rt] = v;
            else push_up(frt, rt);
            return;
        }
        int mid = (be + en) >> 1;
        if (y <= mid) subUpdate(frt, fl, fr, rt << 1, x, y, v, be, mid);
        else subUpdate(frt, fl, fr, rt << 1 | 1, x, y, v, mid + 1, en);
        subPush_up(frt, rt);
    }
    void update(int rt, int x, int y, int v, int be, int en) {
        if (be == en) {
            subUpdate(rt, be, en, 1, x, y, v, 1, n);
            return;
        }
        int mid = (be + en) >> 1;
        if (x <= mid) update(rt << 1, x, y, v, be, mid);
        else update(rt << 1 | 1, x, y, v, mid + 1, en);
        subUpdate(rt, be, en, 1, x, y, v, 1, n);
    }

    inline pii push_pii(const pii &ta, const pii &tb) {
        return mp(max(ta.first, tb.first), min(ta.second, tb.second));
    }
    inline pii subQuery(int frt, int rt, int x1, int x2, int y1, int y2, int be, int en) {
        if (y1 <= be && en <= y2) return mp(maxx[frt][rt], minn[frt][rt]);
        int mid = (be + en) >> 1;
        pii ans = mp(-inf, inf);
        if (y1 <= mid) ans = push_pii(ans, subQuery(frt, rt << 1, x1, x2, y1, y2, be, mid));
        if (y2 > mid) ans = push_pii(ans, subQuery(frt, rt << 1 | 1, x1, x2, y1, y2, mid + 1,
            en));
        return ans;
    }
    pii query(int rt, int x1, int x2, int y1, int y2, int be, int en) { // x1 <= x2, y1 <= y2
        if (x1 <= be && en <= x2) return subQuery(rt, 1, x1, x2, y1, y2, 1, n);
        int mid = (be + en) >> 1;
        pii ans = mp(-inf, inf);
        if (x1 <= mid) ans = push_pii(ans, query(rt << 1, x1, x2, y1, y2, be, mid));
        if (x2 > mid) ans = push_pii(ans, query(rt << 1 | 1, x1, x2, y1, y2, mid + 1, en));
        return ans;
    }
} tree;
```

5.6 ZKW 线段树

5.6.1 开局

```
for (M = 1; M <= n;M <<= 1);//获得层数M
for (int i = 1; i <= M<<1; i++) {//初始化
    T[i]=0;
}
for (int i = 1, x; i <= n; i++) {//建树
    scanf("%d", &x);
    modify(i, x);
}
```

5.6.2 单点修改 + 区间查询

```
void modify(int n, int v) {//单点修改
    for (T[n += M] += v, n >>= 1; n; n >>= 1)
        T[n] = T[n + n] + T[n + n + 1];
}

ll query(int l, int r) {//区间查询和，可调为最大
    ll ans = 0;
    for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (~l & 1) ans += T[l ^ 1];
        if (r & 1) ans += T[r ^ 1];
    }
    return ans;
}
```

5.6.3 单点修改 + 区间查询最大字段和

```
struct Node {
    ll pre, suf, max, sum;
} T[200000], null;

Node merge(Node l, Node r) {
    Node res;
    res.sum = l.sum + r.sum;//区间和
    res.pre = max(l.pre, l.sum + r.pre);//最大前缀
    res.suf = max(r.suf, l.suf + r.sum);//最大
    res.max = max(l.max, r.max);//最大子段和
    res.max = max(res.max, l.suf + r.pre);
    return res;
}

void modify(int n, int v) {//单点修改
    for (T[n += M] = {v, v, v, v}, n >>= 1; n; n >>= 1)
        T[n] = merge(T[n + n], T[n + n + 1]);
}

ll query(int l, int r) {//查询
    Node resl(null), resr(null);
//  resl.pre=resl.suf=resl.sum=0;
//  resr.pre=resr.suf=resr.sum=0;
    resl.max = resr.max = -inf;
    for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (~l & 1) resl = merge(resl, T[l ^ 1]);
```

```
        if (r & 1) resr = merge(T[r ^ 1], resr);
    }
    return merge(resl, resr).max;
}
```

5.6.4 区间加减 + 单点查询

```
void add(int l, int r, int v) {//区间加减
    for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (~l & 1) T[l ^ 1] += v;
        if (r & 1) T[r ^ 1] += v;
    }
}

ll query(int n) {//单点查询
    ll ans = 0;
    for (n += M; n; n >>= 1) ans += T[n];
    return ans;
}
```

5.6.5 区间加减 + 区间最值查询（lazy 标记）

```
void add(int L, int R, int v) {//区间修改
    L += M - 1, R += M + 1;
    for (int l = L, r = R; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (~l & 1) lazy[l ^ 1] += v, T[l ^ 1] += v;
        if (r & 1) lazy[r ^ 1] += v, T[r ^ 1] += v;
    }
    for (int l = L >> 1, r = R >> 1; l; l >>= 1, r >>= 1) {
        T[l] = max(T[l + l], T[l + l + 1]) + lazy[l];
        T[r] = max(T[r + r], T[r + r + 1]) + lazy[r];
    }
}

int query(int l, int r) {//区间查询
    int lmax = -inf, rmax = -inf;
    for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (lazy[l] && lmax != -inf) lmax += lazy[l];
        if (lazy[r] && rmax != -inf) rmax += lazy[r];
        if (~l & 1) lmax = max(lmax, T[l ^ 1]);
        if (r & 1) rmax = max(rmax, T[r ^ 1]);
    }
    for (; l; l >>= 1, r >>= 1) {
        lmax += lazy[l], rmax += lazy[r];
    }
    return max(lmax, rmax);
}
```

5.7 吉司机线段树

5.7.1 区间取 min+ 区间查询 $O(m \log n)$

```
class JLS { public:
    struct node {
        int l, r;
        int fi_max, se_max, max_cnt;    // 最大值，次大值，最大值个数
        ll sum;
    } T[MAXN << 2];
    int lazy[MAXN << 2];

    inline void push_up(int rt) {
        T[rt].sum = T[rt << 1].sum + T[rt << 1 | 1].sum;
        if (T[rt << 1].fi_max == T[rt << 1 | 1].fi_max) { // 左右儿子的最大值相同
            T[rt].fi_max = T[rt << 1].fi_max;
            T[rt].se_max = max(T[rt << 1].se_max, T[rt << 1 | 1].se_max);
            T[rt].max_cnt = T[rt << 1].max_cnt + T[rt << 1 | 1].max_cnt;
        } else if (T[rt << 1].fi_max > T[rt << 1 | 1].fi_max) {
            T[rt].fi_max = T[rt << 1].fi_max;
            T[rt].se_max = max(T[rt << 1].se_max, T[rt << 1 | 1].fi_max);
            T[rt].max_cnt = T[rt << 1].max_cnt;
        } else {
            T[rt].fi_max = T[rt << 1 | 1].fi_max;
            T[rt].se_max = max(T[rt << 1].fi_max, T[rt << 1 | 1].se_max);
            T[rt].max_cnt = T[rt << 1 | 1].max_cnt;
        }
    }

    inline void push_down(int rt) {
        if (lazy[rt] != -1) {
            if (T[rt << 1].fi_max > lazy[rt]) {   // left son
                T[rt << 1].sum += (1ll * lazy[rt] - T[rt << 1].fi_max) * T[rt << 1].max_cnt;
                T[rt << 1].fi_max = lazy[rt], lazy[rt << 1] = lazy[rt];
            }
            if (T[rt << 1 | 1].fi_max > lazy[rt]) { // right son
                T[rt << 1 | 1].sum += (1ll * lazy[rt] - T[rt << 1 | 1].fi_max) * T[rt << 1 |
                    1].max_cnt;
                T[rt << 1 | 1].fi_max = lazy[rt], lazy[rt << 1 | 1] = lazy[rt];
            }
            lazy[rt] = -1;
        }
    }

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        lazy[rt] = -1;
        if (l == r) {
            T[rt].sum = T[rt].fi_max = a[l], T[rt].se_max = -1, T[rt].max_cnt = 1;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }

    void update_min(int rt, int L, int R, int val) {    // a[L],...,a[R] <- min(val, a[i])
        if (T[rt].fi_max <= val) return;
        if (L <= T[rt].l && T[rt].r <= R && T[rt].se_max < val) {
            if (T[rt].fi_max > val) {
```

```
                T[rt].sum += (1ll * val - T[rt].fi_max) * T[rt].max_cnt;
                T[rt].fi_max = val, lazy[rt] = val;
            }
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update_min(rt << 1, L, R, val);
        if (R > mid) update_min(rt << 1 | 1, L, R, val);
        push_up(rt);
    }

    int query_max(int rt, int L, int R) {   // find max value
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_max;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = -1;
        if (L <= mid) ans = max(ans, query_max(rt << 1, L, R));
        if (R > mid) ans = max(ans, query_max(rt << 1 | 1, L, R));
        return ans;
    }

    ll query_sum(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        ll ans = 0;
        if (L <= mid) ans += query_sum(rt << 1, L, R);
        if (R > mid) ans += query_sum(rt << 1 | 1, L, R);
        return ans;
    }
} tree;
```

5.7.2 支持区间加（**BZOJ4695** 最假女选手）$O(m\log^2 n)$

```
class JLS { public:
    struct node {
        int l, r;
        int fi_max, se_max, fi_min, se_min;
        int cnt_max, cnt_min;
        ll sum;
    } T[MAXN << 2];
    ll add[MAXN << 2];
#define lson rt<<1
#define rson rt<<1|1
    inline void push_up(int rt) {
        T[rt].sum = T[lson].sum + T[rson].sum;
        // max
        if (T[lson].fi_max == T[rson].fi_max) {
            T[rt].fi_max = T[lson].fi_max;
            T[rt].se_max = max(T[lson].se_max, T[rson].se_max);
            T[rt].cnt_max = T[lson].cnt_max + T[rson].cnt_max;
        } else if (T[lson].fi_max > T[rson].fi_max) {
            T[rt].fi_max = T[lson].fi_max;
```

```
            T[rt].se_max = max(T[lson].se_max, T[rson].fi_max);
            T[rt].cnt_max = T[lson].cnt_max;
        } else {
            T[rt].fi_max = T[rson].fi_max;
            T[rt].se_max = max(T[lson].fi_max, T[rson].se_max);
            T[rt].cnt_max = T[rson].cnt_max;
        }
        // min
        if (T[lson].fi_min == T[rson].fi_min) {
            T[rt].fi_min = T[lson].fi_min;
            T[rt].se_min = min(T[lson].se_min, T[rson].se_min);
            T[rt].cnt_min = T[lson].cnt_min + T[rson].cnt_min;
        } else if (T[lson].fi_min < T[rson].fi_min) {
            T[rt].fi_min = T[lson].fi_min;
            T[rt].se_min = min(T[lson].se_min, T[rson].fi_min);
            T[rt].cnt_min = T[lson].cnt_min;
        } else {
            T[rt].fi_min = T[rson].fi_min;
            T[rt].se_min = min(T[lson].fi_min, T[rson].se_min);
            T[rt].cnt_min = T[rson].cnt_min;
        }
    }

    inline void push_add(int rt, int tg) {
        T[rt].sum += (ll) (T[rt].r - T[rt].l + 1) * tg;
        T[rt].fi_max += tg, T[rt].fi_min += tg;
        if (T[rt].se_max != -inf) T[rt].se_max += tg;
        if (T[rt].se_min != inf) T[rt].se_min += tg;
        add[rt] += tg;
    }

    inline void push_min(int rt, int tg) {
        T[rt].sum = T[rt].sum - (ll) (T[rt].fi_max - tg) * T[rt].cnt_max;
        if (T[rt].fi_max == T[rt].fi_min) T[rt].fi_max = T[rt].fi_min = tg;
        else if (T[rt].fi_max == T[rt].se_min) T[rt].fi_max = T[rt].se_min = tg;
        else T[rt].fi_max = tg;
    }

    inline void push_max(int rt, int tg) {
        T[rt].sum = T[rt].sum + (ll) (tg - T[rt].fi_min) * T[rt].cnt_min;
        if (T[rt].fi_min == T[rt].fi_max) T[rt].fi_min = T[rt].fi_max = tg;
        else if (T[rt].fi_min == T[rt].se_max) T[rt].fi_min = T[rt].se_max = tg;
        else T[rt].fi_min = tg;
    }

    inline void push_down(int rt) {
        if (add[rt]) {
            push_add(lson, add[rt]), push_add(rson, add[rt]);
            add[rt] = 0;
        }
        if (T[rt].fi_max < T[lson].fi_max) push_min(lson, T[rt].fi_max);
        if (T[rt].fi_max < T[rson].fi_max) push_min(rson, T[rt].fi_max);
        if (T[rt].fi_min > T[lson].fi_min) push_max(lson, T[rt].fi_min);
        if (T[rt].fi_min > T[rson].fi_min) push_max(rson, T[rt].fi_min);
    }

```

```
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        add[rt] = 0;
        if (l == r) {
            T[rt].sum = T[rt].fi_max = T[rt].fi_min = a[l];
            T[rt].se_max = -inf, T[rt].se_min = inf;
            T[rt].cnt_min = T[rt].cnt_max = 1;
            return;
        }
        int mid = (l + r) >> 1;
        build(lson, l, mid), build(rson, mid + 1, r);
        push_up(rt);
    }

    void update_add(int rt, int L, int R, int v) {  // add v to [L, R]
        if (L <= T[rt].l && T[rt].r <= R) {
            push_add(rt, v);
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update_add(lson, L, R, v);
        if (R > mid) update_add(rson, L, R, v);
        push_up(rt);
    }

    void update_min(int rt, int L, int R, int v) {  // a[L],...,a[R] <- min(val, a[i])
        if (v >= T[rt].fi_max) return;
        if (L <= T[rt].l && T[rt].r <= R && T[rt].se_max < v) {
            push_min(rt, v);
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update_min(lson, L, R, v);
        if (R > mid) update_min(rson, L, R, v);
        push_up(rt);
    }

    void update_max(int rt, int L, int R, int v) {  // a[L],...,a[R] <- max(val, a[i])
        if (v <= T[rt].fi_min) return;
        if (L <= T[rt].l && T[rt].r <= R && T[rt].se_min > v) {
            push_max(rt, v);
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update_max(lson, L, R, v);
        if (R > mid) update_max(rson, L, R, v);
        push_up(rt);
    }

    ll query_sum(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
```

```
        ll ans = 0;
        if (L <= mid) ans += query_sum(lson, L, R);
        if (R > mid) ans += query_sum(rson, L, R);
        return ans;
    }

    int query_max(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_max;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = -inf;
        if (L <= mid) ans = max(ans, query_max(lson, L, R));
        if (R > mid) ans = max(ans, query_max(rson, L, R));
        return ans;
    }

    int query_min(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_min;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = inf;
        if (L <= mid) ans = min(ans, query_min(lson, L, R));
        if (R > mid) ans = min(ans, query_min(rson, L, R));
        return ans;
    }
} tree;
```

5.7.3 维护区间最值操作与区间历史最值（洛谷线段树 3）$O(m\log^2 n)$

给出一个长度为 n 的数列 A，同时定义一个辅助数组 B，B 开始与 A 完全相同。接下来进行了 m 次操作，操作有五种类型，按以下格式给出：

1 l r k 对于所有的 $i\in[l,r]$，将 A_i 加上 k（k 可以为负数）。

2 l r v 对于所有的 $i\in[l,r]$，将 A_i 变成 $\min(A_i,v)$。

3 l r 求 $\sum_{i=l}^{r}A_i$。

4 l r 对于所有的 $i\in[l,r]$，求 A_i 的最大值。

5 l r 对于所有的 $i\in[l,r]$，求 B_i 的最大值。

在每一次操作后，我们都进行一次更新，让 $B_i\leftarrow\max(B_i,A_i)$。

```
class JLS { public:
    struct node1 {  // 实时信息
        int l, r;   // 左端点，右端点
        int fi_max, se_max, cnt_max; // 当前区间最大值，区间次大值，区间最大值个数
        int fi_add, se_add; // 区间最大值lazy标记，区间次大值lazy标记
        ll sum;    // 当前区间和
    } now[MAXN << 2];
    struct node2 {  // 历史信息
        int fi_max; // 历史区间最大值
        int fi_add, se_add; // 历史区间最大值lazy标记，区间次大值lazy标记
    } old[MAXN << 2];
#define lson rt<<1
#define rson rt<<1|1
    inline void push_up(int rt) {
        now[rt].sum = now[lson].sum + now[rson].sum;
        old[rt].fi_max = max(old[lson].fi_max, old[rson].fi_max);
        // max
```

```
        if (now[lson].fi_max == now[rson].fi_max) {
            now[rt].fi_max = now[lson].fi_max;
            now[rt].se_max = max(now[lson].se_max, now[rson].se_max);
            now[rt].cnt_max = now[lson].cnt_max + now[rson].cnt_max;
        } else if (now[lson].fi_max > now[rson].fi_max) {
            now[rt].fi_max = now[lson].fi_max;
            now[rt].se_max = max(now[lson].se_max, now[rson].fi_max);
            now[rt].cnt_max = now[lson].cnt_max;
        } else {
            now[rt].fi_max = now[rson].fi_max;
            now[rt].se_max = max(now[lson].fi_max, now[rson].se_max);
            now[rt].cnt_max = now[rson].cnt_max;
        }
    }

    // v1, v3 change now; v2, v4 change old;
    inline void push_node(int rt, int v1, int v2, int v3, int v4) {
        old[rt].fi_max = max(old[rt].fi_max, now[rt].fi_max + v2);
        old[rt].fi_add = max(old[rt].fi_add, now[rt].fi_add + v2);
        old[rt].se_add = max(old[rt].se_add, now[rt].se_add + v4);

        now[rt].sum += 1ll * v1 * now[rt].cnt_max +
                       1ll * v3 * (now[rt].r - now[rt].l + 1 - now[rt].cnt_max);
        now[rt].fi_max += v1;
        if (now[rt].se_max != inf) now[rt].se_max += v3;
        now[rt].fi_add += v1, now[rt].se_add += v3;
    }

    inline void push_down(int rt) {
        int tmp = max(now[lson].fi_max, now[rson].fi_max);

        if (now[lson].fi_max == tmp) push_node(lson, now[rt].fi_add, old[rt].fi_add,
            now[rt].se_add, old[rt].se_add);
        else push_node(lson, now[rt].se_add, old[rt].se_add, now[rt].se_add, old[rt].se_add);

        if (now[rson].fi_max == tmp) push_node(rson, now[rt].fi_add, old[rt].fi_add,
            now[rt].se_add, old[rt].se_add);
        else push_node(rson, now[rt].se_add, old[rt].se_add, now[rt].se_add, old[rt].se_add);

        now[rt].fi_add = now[rt].se_add = old[rt].fi_add = old[rt].se_add = 0;
    }

    void build(int rt, int l, int r) {
        now[rt].l = l, now[rt].r = r;
        now[rt].fi_add = now[rt].se_add = 0;
        old[rt].fi_add = old[rt].se_add = 0;
        if (l == r) {
            now[rt].sum = now[rt].fi_max = a[l];
            now[rt].cnt_max = 1;
            now[rt].se_max = -inf;

            old[rt].fi_max = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(lson, l, mid), build(rson, mid + 1, r);
```

```
72          push_up(rt);
73      }
74
75      void update_add(int rt, int L, int R, int v) {  // op1
76          if (L <= now[rt].l && now[rt].r <= R) {
77              push_node(rt, v, v, v, v);
78              return;
79          }
80          push_down(rt);
81          int mid = (now[rt].l + now[rt].r) >> 1;
82          if (L <= mid) update_add(lson, L, R, v);
83          if (R > mid) update_add(rson, L, R, v);
84          push_up(rt);
85      }
86
87      void update_min(int rt, int L, int R, int v) {  // op2
88          if (v >= now[rt].fi_max) return;
89          if (L <= now[rt].l && now[rt].r <= R && now[rt].se_max < v) {
90              push_node(rt, v - now[rt].fi_max, v - now[rt].fi_max, 0, 0);
91              return;
92          }
93          push_down(rt);
94          int mid = (now[rt].l + now[rt].r) >> 1;
95          if (L <= mid) update_min(lson, L, R, v);
96          if (R > mid) update_min(rson, L, R, v);
97          push_up(rt);
98      }
99
100     ll query_sum(int rt, int L, int R) {    // op3
101         if (L <= now[rt].l && now[rt].r <= R) return now[rt].sum;
102         push_down(rt);
103         int mid = (now[rt].l + now[rt].r) >> 1;
104         ll ans = 0;
105         if (L <= mid) ans += query_sum(lson, L, R);
106         if (R > mid) ans += query_sum(rson, L, R);
107         return ans;
108     }
109
110     int query_max1(int rt, int L, int R) {  // op4
111         if (L <= now[rt].l && now[rt].r <= R) return now[rt].fi_max;
112         push_down(rt);
113         int mid = (now[rt].l + now[rt].r) >> 1;
114         int ans = -inf;
115         if (L <= mid) ans = max(ans, query_max1(lson, L, R));
116         if (R > mid) ans = max(ans, query_max1(rson, L, R));
117         return ans;
118     }
119
120     int query_max2(int rt, int L, int R) {  // op5
121         if (L <= now[rt].l && now[rt].r <= R) return old[rt].fi_max;
122         push_down(rt);
123         int mid = (now[rt].l + now[rt].r) >> 1;
124         int ans = -inf;
125         if (L <= mid) ans = max(ans, query_max2(lson, L, R));
126         if (R > mid) ans = max(ans, query_max2(rson, L, R));
127         return ans;
```

```
    }
#undef lson
#undef rson
} tree;
```

5.8 李超线段树

5.8.1 函数定点最值（[HEOI2013]Segment）

```
/*
  input       output
  6
  1 8 5 10 8
  1 6 7 2 6
  0 2         2
  0 11        0
  1 4 7 6 7
  0 5         3
*/
class LC {  public:
    struct LINE {
        double k, b;
    } p[MAXN];
    int sum[MAXN << 2];
    int cnt = 0;
    void init() { cnt = 0; }
    void add_seg(int x0, int y0, int x1, int y1) {
        cnt++;
        if (x0 == x1) p[cnt].k = 0, p[cnt].b = max(y0, y1);
        else p[cnt].k = 1.0 * (y1 - y0) / (x1 - x0), p[cnt].b = y0 - p[cnt].k * x0;
    }
    void update(int rt, int L, int R, int be, int en, int u) {
        int mid = (be + en) >> 1;
        if (L <= be && en <= R) {
            int v = sum[rt];
            double ansu = p[u].b + p[u].k * mid, ansv = p[v].b + p[v].k * mid;
            if (be == en) {
                if (ansu > ansv) sum[rt] = u;
                return;
            }
            if (p[v].k < p[u].k) {
                if (ansu > ansv) {
                    sum[rt] = u;
                    update(rt << 1, L, R, be, mid, v);
                } else update(rt << 1 | 1, L, R, mid + 1, en, u);
            } else if (p[v].k > p[u].k) {
                if (ansu > ansv) {
                    sum[rt] = u;
                    update(rt << 1 | 1, L, R, mid + 1, en, v);
                } else update(rt << 1, L, R, be, mid, u);
            } else {
                if (p[u].b > p[v].b) sum[rt] = u;
            }
            return;
```

```
        }
        if (L <= mid) update(rt << 1, L, R, be, mid, u);
        if (R > mid) update(rt << 1 | 1, L, R, mid + 1, en, u);
    }
    typedef pair<double, int> pdi;
    pdi pmax(pdi x, pdi y) {
        if (x.first < y.first) return y;
        else if (x.first > y.first) return x;
        else return x.second < y.second ? x : y;
    }
    pdi query(int rt, int d, int be, int en) {
        if (be == en) {
            int v = sum[rt];
            double ans = (p[v].b + p[v].k * d);
            return {ans, sum[rt]};
        }
        int mid = (be + en) >> 1;
        int v = sum[rt];
        double res = (p[v].b + p[v].k * d);
        pdi ans = {res, sum[rt]};
        if (d <= mid) return pmax(ans, query(rt << 1, d, be, mid));
        else return pmax(ans, query(rt << 1 | 1, d, mid + 1, en));
    }
} tree;
int main() {
    int n; scanf("%d", &n);
    while (n--) {
        int opt; scanf("%d", &opt);
        if (opt == 1) {
            int x0, y0, x1, y1; scanf("%d%d%d%d", &x0, &y0, &x1, &y1);
            if (x0 > x1) swap(x0, x1), swap(y0, y1);  // notice x0, x1
            tree.add_seg(x0, y0, x1, y1);
            tree.update(1, x0, x1, 1, MOD1, tree.cnt);
        } else {
            int x; scanf("%d", &x);
            printf("%d\n", tree.query(1, x, 1, MOD1).second);
        }
    }
}
```

5.8.2 李超上树（[SDOI2016] 游戏）$O(m\log^3 n)$

有时，$Alice$ 会选择一条从 s 到 t 的路径，在这条路径上的每一个点上都添加一个数字。对于路径上的一个点 r，若 r 与 s 的距离是 dis，那么 $Alice$ 在点 r 上添加的数字是 $a \times dis + b$。

有时，Bob 会选择一条从 s 到 t 的路径。他需要先从这条路径上选择一个点，再从那个点上选择一个数字。

Bob 选择的数字越小越好，但大量的数字让 Bob 眼花缭乱。Bob 需要你帮他找出他能够选择的最小的数字。

```
struct Edge {
    int to, nex, w;
} e[MAXN << 1];
int head[MAXN], tol;
void addEdge(int u, int v, int w) {
    e[tol].to = v, e[tol].w = w, e[tol].nex = head[u], head[u] = tol, tol++;
}

int son[MAXN], dfn[MAXN], _dfn[MAXN], dfn_cnt, fa[MAXN], dep[MAXN], siz[MAXN], top[MAXN];
```

```
ll dis[MAXN];
int LCA(int u, int v) {
    while (top[u] != top[v]) {
        dep[top[u]] > dep[top[v]] ? u = fa[top[u]] : v = fa[top[v]];
    }
    return dep[u] > dep[v] ? v : u;
}
void dfs1(int u, int f, int deep) {
    dep[u] = deep, fa[u] = f, siz[u] = 1;
    int maxson = -1;
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        if (v == f) continue;
        dis[v] = dis[u] + e[i].w;
        dfs1(v, u, deep + 1);
        siz[u] += siz[v];
        if (siz[v] > maxson) son[u] = v, maxson = siz[v];
    }
}
void dfs2(int u, int topf) {
    dfn[u] = ++dfn_cnt, _dfn[dfn_cnt] = u;
    top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        if (v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
}

class LC {  public:
    struct Line {
        int k; ll b;
    } p[MAXN];
    int cnt;

    void init() {
        cnt = 0;
        p[0].k = 0, p[0].b = inf;
    }

    void addLine(int k, ll b) {
        cnt++;
        p[cnt].k = k, p[cnt].b = b;
    }

    inline ll cal(int x, int id) {
        return (ll) p[id].k * dis[_dfn[x]] + p[id].b;
    }

    int s[MAXN<<2]; ll minn[MAXN<<2];

    inline void push_up(int rt) {
        minn[rt] = min(minn[rt], min(minn[rt << 1], minn[rt << 1 | 1]));
    }
```

```
    void build(int rt, int l, int r) {
        s[rt] = 0, minn[rt] = inf;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
    }

    void update(int rt, int L, int R, int be, int en, int u) {
        int mid = (be + en) >> 1;
        if (L <= be && en <= R) {
            int v = s[rt];
            if (cal(be, u) <= cal(be, v) && cal(en, u) <= cal(en, v)) {
                s[rt] = u, minn[rt] = min(minn[rt], min(cal(be, u), cal(en, u)));
                return;
            }
            if (cal(be, u) >= cal(be, v) && cal(en, u) >= cal(en, v)) return;
            if (p[u].k < p[v].k) {
                if (cal(mid, u) <= cal(mid, v)) {
                    s[rt] = u;
                    update(rt << 1, L, R, be, mid, v);
                } else update(rt << 1 | 1, L, R, mid + 1, en, u);
            } else {
                if (cal(mid, u) <= cal(mid, v)) {
                    s[rt] = u;
                    update(rt << 1 | 1, L, R, mid + 1, en, v);
                } else update(rt << 1, L, R, be, mid, u);
            }
            minn[rt] = min(minn[rt], min(cal(be, u), cal(en, u)));
            push_up(rt);
            return;
        }
        if (L <= mid) update(rt << 1, L, R, be, mid, u);
        if (R > mid) update(rt << 1 | 1, L, R, mid + 1, en, u);
        push_up(rt);
    }

    ll query(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return minn[rt];
        int mid = (be + en) >> 1;
        ll ans = inf;
        if (p[s[rt]].b != inf) ans = min(cal(max(L, be), s[rt]), cal(min(R, en), s[rt]));
        if (L <= mid) ans = min(ans, query(rt << 1, L, R, be, mid));
        if (R > mid) ans = min(ans, query(rt << 1 | 1, L, R, mid + 1, en));
        return ans;
    }
} tree;

int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) head[i] = -1;  // init graph
    for (int i = 2; i <= n; i++) {
        int u, v, w; scanf("%d%d%d", &u, &v, &w);
        addEdge(u, v, w), addEdge(v, u, w);
    }
    dfs1(1, 1, 1);  dfs2(1, 1);
    tree.init();
```

```
    tree.build(1, 1, n);

    while (m--) {
        int opt;
        scanf("%d", &opt);
        if (opt == 1) {
            int s, t, a, b; scanf("%d%d%d%d", &s, &t, &a, &b);
            int lca = LCA(s, t);
            auto update = [&](int u, int v, int a, int b) {
                while (top[u] != top[v]) {
                    tree.update(1, dfn[top[u]], dfn[u], 1, n, tree.cnt);
                    u = fa[top[u]];
                }
                tree.update(1, dfn[v], dfn[u], 1, n, tree.cnt);
            };
            tree.addLine(-a, dis[s] * a + b);
            update(s, lca, a, b);
            tree.addLine(a, (dis[s] - (dis[lca] << 1)) * a + b);
            update(t, lca, a, b);
        } else {
            int s, t; scanf("%d%d", &s, &t);
            auto query = [&](int u, int v) {
                ll ans = inf;
                while (top[u] != top[v]) {
                    if (dep[top[u]] < dep[top[v]]) swap(u, v);
                    ans = min(ans, tree.query(1, dfn[top[u]], dfn[u], 1, n));
                    u = fa[top[u]];
                }
                if (dep[u] > dep[v]) swap(u, v);
                ans = min(ans, tree.query(1, dfn[u], dfn[v], 1, n));
                return ans;
            };
            printf("%lld\n", query(s, t));
        }
    }
}
```

5.9 扫描线

5.9.1 矩形并面积

```
namespace Discrete {
    ll b[MAXN << 1];
    int tol = 1, blen = 0;
    inline void push(ll x) { b[tol++] = x; }
    void init() { blen = 0; tol = 1;}
    void build() {
        sort(b + 1, b + tol);
        blen = unique(b + 1, b + tol) - (b + 1);
    }
};
using namespace Discrete;
struct Line {
    ll x, y1, y2;
```

```
    int mark;
    Line() {}
    Line(ll _x, ll _y1, ll _y2, int _mark) {
        x = _x, y1 = _y1, y2 = _y2, mark = _mark;
    }
    bool operator<(const Line &tb) { return x < tb.x;}
} line[MAXN << 1];

class Seg_Tree { public:
    struct node {
        int l, r, val;
        ll len;
    } T[MAXN << 2];
    inline void push_up(int rt) {
        int l = T[rt].l, r = T[rt].r;
        if (T[rt].val) T[rt].len = b[r + 1] - b[l];
        else T[rt].len = T[rt << 1].len + T[rt << 1 | 1].len;
    }

    void build(int l, int r, int rt) {
        T[rt].l = l, T[rt].r = r;
        T[rt].val = 0;
        T[rt].len = 0;
        if (l == r) {
            T[rt<<1].val = T[rt<<1|1].val = T[rt<<1].len = T[rt<<1|1].len = 0;
            return ;
        }
        int mid = (l + r) >> 1;
        build(l, mid, rt << 1), build(mid + 1, r, rt << 1 | 1);
    }
    void update(ll L, ll R, int c, int rt) {
        int l = T[rt].l, r = T[rt].r;
        if (b[r + 1] <= L || R <= b[l]) return;
        if (L <= b[l] && b[r + 1] <= R) {
            T[rt].val += c;
            push_up(rt);
            return;
        }
        update(L, R, c, rt << 1); update(L, R, c, rt << 1 | 1);
        push_up(rt);
    }
} tree;

int main() {
    int n; int kase = 0;
    while (~scanf("%d", &n) && n) {
        init();
        for (int i = 1; i <= n; i++) {
            ll x1, y1, x2, y2; scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
            push(y1); push(y2);
            line[2 * i - 1] = Line(x1, y1, y2, 1);
            line[2 * i] = Line(x2, y1, y2, -1);
        }
        n <<= 1;
        sort(line + 1, line + 1 + n);
        build();
```

```
        tree.build(1, blen - 1, 1);
        ll res = 0;
        for (int i = 1; i < n; i++) {
            tree.update(line[i].y1, line[i].y2, line[i].mark, 1);
            res += tree.T[1].len * (line[i + 1].x - line[i].x);
        }
        printf("Test case #%d\n", ++kase);
        printf("Total explored area: %.2f\n", res);
        printf("\n");
    }
}
```

5.9.2 矩形并周长

```
using namespace Discrete; // 矩形并面积中的namespace Discrete
struct Line {
    int x, y1, y2;
    int mark;
    Line() {}
    Line(int _x, int _y1, int _y2, int _mark) {
        x = _x, y1 = _y1, y2 = _y2, mark = _mark;
    }
    bool operator<(const Line &tb) const {
        if (x == tb.x) return mark > tb.mark; //
            如果出现了两条高度相同的扫描线，也就是两矩形相邻,那么需要先扫底边再扫顶边，否则就会多算这
        return x < tb.x;
    }
} line[MAXN << 1];
class Seg_Tree { public:
    struct SegTree {
        int l, r, sum, len, c;
//  c表示区间线段条数
        bool lc, rc;
//  lc, rc分别表示左、右端点是否被覆盖
//  统计线段条数(tree[x].c)会用到
    } tree[MAXN << 3];

    void build_tree(int x, int l, int r) {
        tree[x].l = l, tree[x].r = r;
        tree[x].lc = tree[x].rc = false;
        tree[x].sum = tree[x].len = 0;
        tree[x].c = 0;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build_tree(x<<1, l, mid);
        build_tree(x<<1|1, mid + 1, r);
    }

    void pushup(int x) {
        int l = tree[x].l, r = tree[x].r;
        if (tree[x].sum) {
            tree[x].len = b[r + 1] - b[l];
            tree[x].lc = tree[x].rc = true;
            tree[x].c = 1;
```

```
//      做好相应的标记
        } else {
            tree[x].len = tree[x<<1].len + tree[x<<1|1].len;
            tree[x].lc = tree[x<<1].lc, tree[x].rc = tree[x<<1|1].rc;
            tree[x].c = tree[x<<1].c + tree[x<<1|1].c;
//      如果左儿子左端点被覆盖，那么自己的左端点也肯定被覆盖；右儿子同理
            if (tree[x<<1].rc && tree[x<<1|1].lc) tree[x].c -= 1;
//      如果做儿子右端点和右儿子左端点都被覆盖，
//      那么中间就是连续的一段，所以要 -= 1
        }
    }

    void edit_tree(int x, int L, int R, int c) {
        int l = tree[x].l, r = tree[x].r;
        if (b[l] >= R || b[r + 1] <= L) return;
        if (L <= b[l] && b[r + 1] <= R) {
            tree[x].sum += c;
            pushup(x);
            return;
        }
        edit_tree(x<<1, L, R, c), edit_tree(x<<1|1, L, R, c);
        pushup(x);
    }
}tree;
int main() {
    int n; scanf("%d", &n);
    init();
    for (int i = 1; i <= n; i++) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        push(y1), push(y2);
        line[2 * i - 1] = Line(x1, y1, y2, 1);
        line[2 * i] = Line(x2, y1, y2, -1);
    }
    n <<= 1;
    sort(line + 1, line + 1 + n);
    build(); tree.build_tree(1, 1, blen-1);
    ll res = 0; int pre = 0;
    for (int i = 1; i < n; i++) {
        tree.edit_tree(1, line[i].y1, line[i].y2, line[i].mark);
        res += 1ll * abs(pre - tree.tree[1].len);
        pre = tree.tree[1].len;
        res += 2ll * tree.tree[1].c * (line[i + 1].x - line[i].x);
    }
    res += 1ll * (line[n].y2 - line[n].y1);
    printf("%lld\n", res);
}
```

5.9.3 矩阵求和最值（POJ-2482）

不含边框注意！

```
using namespace Discrete; // 矩形并面积中的namespace Discrete
struct LINE {
    int x, y1, y2, l;
    LINE() {}
```

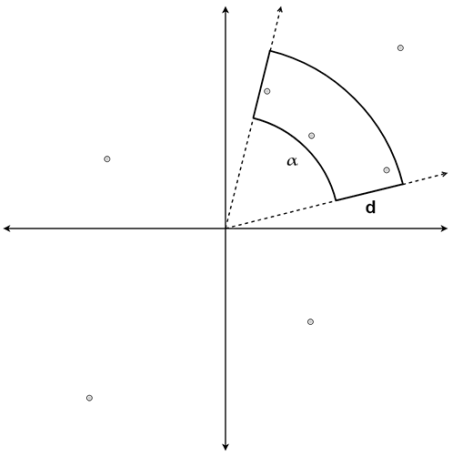
input	output
2 3 5 4 1 2 3 2 3 2 6 3 1 3 5 4 1 2 3 2 3 2 5 3 1	5 6

```
    LINE(int _x, int _y1, int _y2, int _l) {
        x = _x, y1 = _y1, y2 = _y2, l = _l;
    }
} line[MAXN << 1];
class SEG { public:
    struct node {
        int l, r, maxx;
    } T[MAXN << 3]; // 8倍注意!
    int lazy[MAXN << 3];

    inline void push_up(int rt) {
        T[rt].maxx = max(T[rt << 1].maxx, T[rt << 1 | 1].maxx);
    }
    inline void push_down(int rt) {
        if (lazy[rt]) {
            T[rt << 1].maxx += lazy[rt], lazy[rt << 1] += lazy[rt];
            T[rt << 1 | 1].maxx += lazy[rt], lazy[rt << 1 | 1] += lazy[rt];
            lazy[rt] = 0;
        }
    }

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        lazy[rt] = 0;
        if (l == r) {
            T[rt].maxx = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }

    void update(int rt, int L, int R, int v) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].maxx += v;
            lazy[rt] += v;
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update(rt << 1, L, R, v);
        if (R > mid) update(rt << 1 | 1, L, R, v);
        push_up(rt);
    }
```

```
} tree;

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        int n, w, h; scanf("%d%d%d", &n, &w, &h);
        init();
        for (int i = 1; i <= n; i++) {
            int x, y, l; scanf("%d%d%d", &x, &y, &l);
            line[i] = LINE(x, y, y + h - 1, l);
            line[i+n] = LINE(x + w - 1, y, y + h - 1, -l);
            push(y + h - 1);
            push(y);
        }
        n <<= 1;
        build();
        sort(line + 1, line + 1 + n, [&](const LINE &ta, const LINE &tb) {
            if (ta.x != tb.x) return ta.x < tb.x;
            return ta.l > tb.l;
        });
        tree.build(1, 1, blen);
        int res = 0;
        for (int i = 1; i <= n; i++) {
            int l = lower_bound(b+1, b+1+blen, line[i].y1) - b;
            int r = lower_bound(b+1, b+1+blen, line[i].y2) - b;
            tree.update(1, l, r, line[i].l);
            res = max(res, tree.T[1].maxx);
        }
        printf("%d\n", res);
    }
}
```

5.9.4 旋转扫描线

含边框注意！




```
const double eps = 1e-8;
SEG tree; // 矩阵求和最值（POJ-2482）中的SEG
struct point {
    int x, y;
    bool operator < (const point& tb) const {return y < tb.y;}
}p[SIZE<<1];
```

```
int main() {
    int n, d; scanf("%d%d", &n, &d);
    double _alpha; scanf("%lf", &_alpha);
    int alpha = (int)(_alpha * 100.0 + eps);
    for (int i = 0; i < n; i++) {
        int r; double w;
        scanf("%d%lf", &r, &w);
        p[i].x = r; p[i].y = (int)(w * 100.0 + eps);
    }
    std::sort(p, p+n);
    for (int i = 0; i < n; i++) {
        p[i+n].x = p[i].x; p[i+n].y = p[i].y + 36000;
    }
    n *= 2;
    int l = 0, r = 0;
    int res = 0;
    tree.build(0, 100000, 1);
    while (r < n) {
        int cnt = 0;
        for (int i = r; i < n; i++) {
            if (p[i].y == p[r].y) {
                tree.update(std::max(1, p[i].x - d), p[i].x, 1, 1);
                cnt++;
            }
            else break;
        }
        while (l < r) {
            if (p[r].y - p[l].y > alpha) {
                tree.update(std::max(1, p[l].x - d), p[l].x, -1, 1);
                l++;
            }
            else break;
        }
        r += cnt;
        res = std::max(res, tree.T[1].val);
    }
    printf("%d\n", res);
}
```

5.9.5 三维求面积交

```
namespace Discrete_y {
    int b[MAXN << 1], blen, btol;
    void insert(int x) { b[btol++] = x; }
    void init() {
        sort(b, b + btol);
        blen = unique(b, b + btol) - b;
    }
    int val2id(int x) { return lower_bound(b, b + blen, x) - b + 1; }
    int id2val(int x) { return b[x - 1]; }
}
using Discrete_y::id2val;
using Discrete_y::val2id;

```

```
namespace Discrete_z {
    int b[MAXN << 1], blen, btol;
    void insert(int x) { b[btol++] = x; }
    void init() {
        sort(b, b + btol);
        blen = unique(b, b + btol) - b;
    }
    int id2val(int x) { return b[x - 1]; }
}
class SEG { public:
    struct node {
        int l, r, val;
        int len[3];
    } T[MAXN << 3]; // 两倍的点*4

    inline void push_up(int rt) {   // 重点!
        if (T[rt].val >= 1) {
            T[rt].len[0] = id2val(T[rt].r + 1) - id2val(T[rt].l);
        } else if (T[rt].l != T[rt].r) {
            T[rt].len[0] = T[rt << 1].len[0] + T[rt << 1 | 1].len[0];
        } else T[rt].len[0] = 0;

        if (T[rt].val >= 2) {
            T[rt].len[1] = id2val(T[rt].r + 1) - id2val(T[rt].l);
        } else if (T[rt].l != T[rt].r) {
            if (T[rt].val == 1) {
                T[rt].len[1] = T[rt << 1].len[0] + T[rt << 1 | 1].len[0];
            } else {
                T[rt].len[1] = T[rt << 1].len[1] + T[rt << 1 | 1].len[1];
            }
        } else T[rt].len[1] = 0;

        if (T[rt].val >= 3) {
            T[rt].len[2] = id2val(T[rt].r + 1) - id2val(T[rt].l);
        } else if (T[rt].l != T[rt].r) {
            if (T[rt].val == 2) {
                T[rt].len[2] = T[rt << 1].len[0] + T[rt << 1 | 1].len[0];
            } else if (T[rt].val == 1) {
                T[rt].len[2] = T[rt << 1].len[1] + T[rt << 1 | 1].len[1];
            } else {
                T[rt].len[2] = T[rt << 1].len[2] + T[rt << 1 | 1].len[2];
            }
        } else T[rt].len[2] = 0;
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        T[rt].val = 0;
        if (l == r) {
            for (int i = 0; i < 3; i++) {
                T[rt].len[i] = 0;
            }
            push_up(rt);
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
```

```
        push_up(rt);
    }
    void update(int rt, int L, int R, int c) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].val += c;
            push_up(rt);
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update(rt << 1, L, R, c);
        if (R > mid) update(rt << 1 | 1, L, R, c);
        push_up(rt);
    }
} tree;
struct Line {
    int x, y1, y2, z1, z2;
    int mark;
    Line(int _x = 0, int _y1 = 0, int _y2 = 0, int _z1 = 0, int _z2 = 0, int _mark = 0) {
        x = _x, y1 = _y1, y2 = _y2, z1 = _z1, z2 = _z2, mark = _mark;
    }
    bool operator<(const Line &tb) {
        return x < tb.x;
    }
} line[MAXN << 1];

int main() {
    int T;  scanf("%d", &T);
    while (T--) {
        int n;  scanf("%d", &n);
        Discrete_y::btol = 0;
        Discrete_z::btol = 0;
        for (int i = 1; i <= n; i++) {
            int x1, y1, z1, x2, y2, z2;
            scanf("%d%d%d%d%d%d", &x1, &y1, &z1, &x2, &y2, &z2);
            Discrete_y::insert(y1), Discrete_y::insert(y2);
            Discrete_z::insert(z1), Discrete_z::insert(z2);
            line[2 * i - 1] = Line(x1, y1, y2, z1, z2, 1);
            line[2 * i] = Line(x2, y1, y2, z1, z2, -1);
        }
        Discrete_y::init();
        Discrete_z::init();
        n <<= 1;
        sort(line + 1, line + 1 + n);
        ll res = 0;
        for (int zz = 1; zz < Discrete_z::blen; zz++) {
            int z = Discrete_z::id2val(zz);
            tree.build(1, 1, Discrete_y::blen - 1);
            int lastx = 0;  // 注意一定要用lastx, 否则可能导致下一个z不在统计范围内
            ll ans = 0;
            for (int i = 1; i <= n; i++) {
                if (line[i].z1 <= z && z < line[i].z2) {
                    ans += (ll) tree.T[1].len[2] * (line[i].x - lastx), lastx = line[i].x;
                    tree.update(1, val2id(line[i].y1), val2id(line[i].y2)-1, line[i].mark);
                }
            }
            res += ans * (ll)(Discrete_z::id2val(zz + 1) - Discrete_z::id2val(zz));
```

```
        }
        printf("Case %d: %lld\n", kass++, res);
    }
}
```

5.10 可持久化线段树（主席树）

5.10.1 静态区间第 K 小

```
/*
  input                          output
  5 5
  25957 6405 15770 26287 26465
  2 2 1                          6405
  3 4 1                          15770
  4 5 1                          26287
  1 2 2                          25957
  4 4 1                          26287
*/
class HJT { public:
    int ch[MAXN * 70][2], sum[MAXN * 70];
    int tot = 0;
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int update(int rt, int pos, int val, int be, int en) {
        int nrt = ++tot;
        ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        if (be == en) {
            sum[nrt] = sum[rt] + val;
            return nrt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
    }
    int query(int lrt, int rrt, int k, int be, int en) {
        if (be >= en) return be;
        int delta = sum[ch[rrt][0]] - sum[ch[lrt][0]];
        int mid = (be + en) >> 1;
        if (delta >= k) return query(ch[lrt][0], ch[rrt][0], k, be, mid);
        else return query(ch[lrt][1], ch[rrt][1], k - delta, mid + 1, en);
    }
} tree;

int ai[MAXN], root[MAXN];
int main() {
```

```
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &ai[i]);
        Discrete::insert(ai[i]);
    }
    Discrete::init();
    root[0] = 0;
    for (int i = 1; i <= n; i++) {
        root[i] = tree.update(root[i-1], Discrete::val2id(ai[i]), 1, 1, Discrete::blen);
    }
    while(m--) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", Discrete::id2val(tree.query(root[l-1], root[r], k, 1, Discrete::blen)));
    }
}
```

5.10.2 区间内不同数个数

```
class HJT { public:
    int query(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return sum[rt];
        int mid = (be + en) >> 1;
        int ans = 0;
        if (L <= mid) ans += query(ch[rt][0], L, R, be, mid);
        if (R > mid) ans += query(ch[rt][1], L, R, mid + 1, en);
        return ans;
    }
} tree;
int val[MAXN], root[MAXN];
unordered_map<int, int> pre;

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &val[i]);
    root[0] = 0;
    for (int i = 1; i <= n; i++) {
        if (pre[val[i]]) {
            int tmp = tree.change(root[i - 1], pre[val[i]], -1, 1, n);
            root[i] = tree.change(tmp, i, 1, 1, n);
        } else {
            root[i] = tree.change(root[i - 1], i, 1, 1, n);
        }
        pre[val[i]] = i;
    }
    int q; scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", tree.query(root[r], l, r, 1, n));
    }
}
```

5.10.3 树上路径点权第 K 大

```
class HJT { public:
    int query(int lrt, int rrt, int fa, int faa, int k, int be, int en) {
        if (be >= en) return be;
        int mid = (be + en) >> 1;
        int delta = sum[ch[lrt][0]] + sum[ch[rrt][0]] - sum[ch[fa][0]] - sum[ch[faa][0]];
        if (delta >= k) return query(ch[lrt][0], ch[rrt][0], ch[fa][0], ch[faa][0], k, be,
            mid);
        else return query(ch[lrt][1], ch[rrt][1], ch[fa][1], ch[faa][1], k - delta, mid + 1,
            en);
    }
} tree;

struct Edge {
    int to, nex;
} e[MAXN << 1];
int head[MAXN], tol;
void addEdge(int u, int v) {
    e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}
int val[MAXN], root[MAXN], new_val[MAXN];
int dep[MAXN], fa[MAXN][32], lg[MAXN];
void init(int _n) {
    for (int i = 1; i <= _n; i++) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
}
void dfs(int u, int f) {
    fa[u][0] = f;
    dep[u] = dep[f] + 1;
    root[u] = tree.update(root[f], new_val[u], 1, 1, Discrete::blen);
    for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; ~i; i = e[i].nex) {
        int v = e[i].to;
        if (v == f) continue;
        dfs(v, u);
    }
}
int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    while (dep[u] > dep[v]) u = fa[u][lg[dep[u] - dep[v]] - 1];
    if (u == v) return u;
    for (int k = lg[dep[u]] - 1; k >= 0; k--) {
        if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
    }
    return fa[u][0];
}
int main() {
    int n, m; scanf("%d%d", &n, &m);
    init(n);
    memset(fa, 0, sizeof(fa)); // for (int i = 1; i <= n; i++) memset(fa[i], 0, sizeof(fa[i]));
    for (int i = 1; i <= n; i++) head[i] = -1;
    for (int i = 1; i <= n; i++) scanf("%d", &val[i]);

    for (int i = 1; i <= n; i++) Discrete::insert(val[i]);
    Discrete::init();
```

```
    for (int i = 1; i <= n; i++) new_val[i] = Discrete::val2id(val[i]);
    for (int i = 2; i <= n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        addEdge(u, v), addEdge(v, u);
    }
    root[0] = 0;
    dfs(1, 0);
    while (m--) {
        int u, v, k;
        scanf("%d%d%d", &u, &v, &k);
        int lca = LCA(u, v);
        printf("%d\n",
               Discrete::id2val(tree.query(root[u], root[v], root[lca], root[fa[lca][0]], k,
                   1, Discrete::blen)));
    }
}
```

5.10.4 区间 MEX

```
int blen, bep;
class HJT {
public:
    struct node {
        int lson, rson, maxx;
    } T[MAXN * 70];
    int tol;
#define ls T[rt].lson
#define rs T[rt].rson
    inline void push_up(int rt) {
        T[rt].maxx = max(T[ls].maxx, T[rs].maxx);
    }
    int build(int l, int r) {
        int nrt = ++tol;
        int mid = (l + r) >> 1;
        T[nrt].lson = T[nrt].rson = 0; T[nrt].maxx = bep;
        if (l < r) {
            T[nrt].lson = build(l, mid);
            T[nrt].rson = build(mid + 1, r);
            push_up(nrt);
        }
        return nrt;
    }
    int query(int rt, int R, int be, int en) {
        if (be == en) return be;
        int ans = -1;
        int mid = (be + en) >> 1;
        if (T[ls].maxx > R) ans = query(T[rt].lson, R, be, mid);
        if (ans == -1 && T[rs].maxx > R) ans = query(T[rt].rson, R, mid + 1, en);
        return ans;
    }
} tree;

int arr[MAXN], root[MAXN];
int main() {
```

```
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);
    Discrete::push(0);
    for (int i = 1; i <= n; i++) Discrete::push(arr[i]), Discrete::push(arr[i] + 1);
    Discrete::init();
    bep = n + 1;
    root[n + 1] = tree.build(1, blen);
    for (int i = n; i >= 1; i--) {
        int x = Discrete::lb(arr[i]);
        root[i] = tree.update(root[i + 1], 1, blen, x, i);
    }
    while (m--) {
        int l, r;
        scanf("%d%d", &l, &r);
        int res = tree.query(root[l], r, 1, blen);
        if (res == -1) res = blen;
        printf("%d\n", Discrete::get_num(res));
    }
}
```

5.11 有旋 Treap

5.11.1 普通平衡树 Treap

```
// MAXN为最多维护的树中数的个数
class Treap {
private:
    int ch[MAXN][2];
    int val[MAXN], dat[MAXN], sz[MAXN], cnt[MAXN];
    int tot, root;

    int New(int v) {
        val[++tot] = v;
        dat[tot] = rand();
        sz[tot] = 1;
        cnt[tot] = 1;
        return tot;
    }

    void pushup(int id) {
        sz[id] = sz[ch[id][0]] + sz[ch[id][1]] + cnt[id];
    }

    void Rotate(int &id, int d) {
        int temp = ch[id][d ^ 1];
        ch[id][d ^ 1] = ch[temp][d];
        ch[temp][d] = id;
        id = temp;
        pushup(ch[id][d]), pushup(id);
    }

    void _insert(int &id, int v) {
        if (!id) {
            id = New(v);
```

```
31          return;
32      }
33      if (v == val[id])cnt[id]++;
34      else {
35          int d = v < val[id] ? 0 : 1;
36          _insert(ch[id][d], v);
37          if (dat[id] < dat[ch[id][d]]) Rotate(id, d ^ 1);
38      }
39      pushup(id);
40  }
41
42  void _Remove(int &id, int v) {
43      if (!id)return;
44      if (v == val[id]) {
45          if (cnt[id] > 1) {
46              cnt[id]--, pushup(id);
47              return;
48          }
49          if (ch[id][0] || ch[id][1]) {
50              if (!ch[id][1] || dat[ch[id][0]] > dat[ch[id][1]]) {
51                  Rotate(id, 1), _Remove(ch[id][1], v);
52              } else Rotate(id, 0), _Remove(ch[id][0], v);
53              pushup(id);
54          } else id = 0;
55          return;
56      }
57      v < val[id] ? _Remove(ch[id][0], v) : _Remove(ch[id][1], v);
58      pushup(id);
59  }
60
61  int _get_rank(int id, int v) {
62      if (!id) return 0;
63      if (v == val[id]) return sz[ch[id][0]] + 1;
64      else if (v < val[id]) return _get_rank(ch[id][0], v);
65      else return sz[ch[id][0]] + cnt[id] + _get_rank(ch[id][1], v);
66  }
67
68  int _get_val(int id, int rank) {
69      if (!id) return inf_int;
70      if (rank <= sz[ch[id][0]]) return _get_val(ch[id][0], rank);
71      else if (rank <= sz[ch[id][0]] + cnt[id]) return val[id];
72      else return _get_val(ch[id][1], rank - sz[ch[id][0]] - cnt[id]);
73  }
74
75  int _get_pre(int v) {
76      int id = root, pre;
77      while (id) {
78          if (val[id] < v)pre = val[id], id = ch[id][1];
79          else id = ch[id][0];
80      }
81      return pre;
82  }
83
84  int _get_next(int v) {
85      int id = root, next;
86      while (id) {
```

```
            if (val[id] > v)next = val[id], id = ch[id][0];
            else id = ch[id][1];
        }
        return next;
    }

public:
    void init() {
        tot = 0;
        root = 0;
        root = New(-inf_int), ch[root][1] = New(inf_int);
        pushup(root);
    }

    void insert(int x) { _insert(root, x); }    // 插入x数

    void remove(int x) { _Remove(root, x); }    // 删除x数(若有多个相同的数，因只删除一个)

    int get_rank(int x) { return _get_rank(root, x) - 1; }  //
        查询x数的排名(排名定义为比当前数小的数的个数+1)

    int get_val(int x) { return _get_val(root, x + 1); }    // 查询排名为x的数

    int get_pre(int x) { return _get_pre(x); }  // 求x的前驱(前驱定义为小于x，且最大的数)

    int get_next(int x) { return _get_next(x); }    // 求x的后继(后继定义为大于x，且最小的数)
} tree;
```

5.11.2 并查集 + 启发式合并（HDU3726）

```
class Treap { public:
    int ch[MAXN][2], dat[MAXN], siz[MAXN], val[MAXN], cnt[MAXN];
    int tot;
    int pool[MAXN], pool_cnt;
    void init() { tot = 0, pool_cnt = 0; }
    inline int Newid() {
        return pool_cnt ? pool[pool_cnt--] : ++tot;
    }
    inline void Delid(int &rt) {
        if (!rt) return;
        pool[++pool_cnt] = rt;
        dat[rt] = siz[rt] = val[rt] = cnt[rt] = 0;
        ch[rt][0] = ch[rt][1] = val[rt] = 0;
        rt = 0;
    }
    inline int Newnode(int v, int _cnt = 1) {
        int nid = Newid();
        val[nid] = v, dat[nid] = rand(), siz[nid] = _cnt, cnt[nid] = _cnt;
        ch[nid][0] = ch[nid][1] = 0;
        return nid;
    }
    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + cnt[rt];
    }
```

```
25      inline void Rotate(int &rt, int d) {
26          int temp = ch[rt][d ^ 1];
27          ch[rt][d ^ 1] = ch[temp][d];
28          ch[temp][d] = rt;
29          rt = temp;
30          push_up(ch[rt][d]), push_up(rt);
31      }
32
33      void insert(int &rt, int v, int _cnt = 1) {
34          if (!rt) {
35              rt = Newnode(v, _cnt);
36              return;
37          }
38          if (v == val[rt]) cnt[rt] += _cnt;
39          else {
40              int d = v < val[rt] ? 0 : 1;
41              insert(ch[rt][d], v, _cnt);
42              if (dat[rt] < dat[ch[rt][d]]) Rotate(rt, d ^ 1);
43          }
44          push_up(rt);
45      }
46      void remove(int &rt, int v) {
47          if (!rt) return;
48          if (v == val[rt]) {
49              if (cnt[rt] > 1) {
50                  cnt[rt]--, push_up(rt);
51                  return;
52              }
53              if (ch[rt][0] || ch[rt][1]) {
54                  if (!ch[rt][1] || dat[ch[rt][0]] > dat[ch[rt][1]]) {
55                      Rotate(rt, 1), remove(ch[rt][1], v);
56                  } else {
57                      Rotate(rt, 0), remove(ch[rt][0], v);
58                  }
59                  push_up(rt);
60              } else Delid(rt);
61              return;
62          }
63          v < val[rt] ? remove(ch[rt][0], v) : remove(ch[rt][1], v);
64          push_up(rt);
65      }
66
67      int Kth(int rt, int k) {  // call:tree.Kth(root[find(x)], k); 与x相连的第k大值
68          if (!rt) return 0;
69          if (k <= siz[ch[rt][1]]) return Kth(ch[rt][1], k);
70          else if (k <= siz[ch[rt][1]] + cnt[rt]) return val[rt];
71          else return Kth(ch[rt][0], k - siz[ch[rt][1]] - cnt[rt]);
72      }
73      void merge(int &x, int &y) {
74          if (ch[x][0]) merge(ch[x][0], y);
75          if (ch[x][1]) merge(ch[x][1], y);
76          if (cnt[x] > 0) insert(y, val[x], cnt[x]);
77          Delid(x);
78      }
79  } tree;
80
```

```
int F[MAXN];
int root[MAXN];

int find(int x) {
    if (F[x] == x) return x;
    else return F[x] = find(F[x]);
}
void join(int u, int v) { // 两个点相连
    int fu = find(u), fv = find(v);
    if (fu != fv) {
        if (tree.siz[fu] < tree.siz[fv]) F[fu] = fv, tree.merge(root[fu], root[fv]);
        else F[fv] = fu, tree.merge(root[fv], root[fu]);
    }
}

void change(int x, int v, int oldv) { // 将点x的值从oldv -> v
    int fx = find(x);
    tree.remove(root[fx], oldv);
    tree.insert(root[fx], v);
}
```

5.12 无旋 Treap（FHQ Treap）

5.12.1 区间翻转

```
class Treap { public:
    int ch[MAXN][2];
    int dat[MAXN], siz[MAXN], val[MAXN];
    bool fl[MAXN];
    int tot, root;

    void init() {
        tot = 0, root = 0;
    }
    Treap() { init(); }

    inline int Newnode(int v) {
        val[++tot] = v;
        dat[tot] = rand();
        siz[tot] = 1;
        return tot;
    }
    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
    }

    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        int newnode = Newnode(mid);
        ch[newnode][0] = build(l, mid - 1);
        ch[newnode][1] = build(mid + 1, r);
        push_up(newnode);
        return newnode;
```

```
    }

    // 在外面
    if (fl[rt]) {
        if (ch[rt][0]) swap(ch[ch[rt][0]][0], ch[ch[rt][0]][1]), fl[rt] ^= 1;
        if (ch[rt][1]) swap(ch[ch[rt][1]][0], ch[ch[rt][1]][1]), fl[rt] ^= 1;
        fl[rt] = 0;
    }
    // lazy的打标记与线段树类似,在外面 val +=, lazy += ,再向下推(POJ3580)
    if (lazy[rt]) {
        if ()
        if ()
        lazy[rt] = 0;
    }
    inline void push_down(int rt) {
        if (fl[rt]) {
            swap(ch[rt][0], ch[rt][1]);
            if (ch[rt][0]) fl[ch[rt][0]] ^= 1;
            if (ch[rt][1]) fl[ch[rt][1]] ^= 1;
            fl[rt] = 0;
        }

    }



    void split(int rt, int k, int &x, int &y) { // 按照编号进行分裂
        if (!rt) x = y = 0;
        else {
            push_down(rt);
            if (k <= siz[ch[rt][0]]) {
                y = rt;
                split(ch[rt][0], k, x, ch[rt][0]);
            } else {
                x = rt;
                split(ch[rt][1], k - siz[ch[rt][0]] - 1, ch[rt][1], y);
            }
            push_up(rt);
        }
    }

    void split(int rt, int v, int &x, int &y) { // 按权值进行分裂
        if (!rt) x = y = 0;
        else {
            if (val[rt] <= v) {
                x = rt;
                split(ch[rt][1], v, ch[rt][1], y);
            } else {
                y = rt;
                split(ch[rt][0], v, x, ch[rt][0]);
            }
            push_up(rt);
        }
    }

    int merge(int x, int y) {
```

```
        if (!x || !y) return x + y;
        if (dat[x] < dat[y]) {
            push_down(x);
            ch[x][1] = merge(ch[x][1], y);
            push_up(x);
            return x;
        } else {
            push_down(y);
            ch[y][0] = merge(x, ch[y][0]);
            push_up(y);
            return y;
        }
    }

    void dfs(int rt) {
        push_down(rt);
        if (ch[rt][0]) dfs(ch[rt][0]);
        printf("%d ", val[rt]);
        if (ch[rt][1]) dfs(ch[rt][1]);
    }
} tree;

int main() {
    int n, q; scanf("%d%d", &n, &q);
    tree.build(1, n);
    while (q--) {
        int l, r; scanf("%d%d", &l, &r);
        int a, b, c;
        tree.split(tree.root, l - 1, a, b);
        tree.split(b, r - l + 1, b, c);
        tree.fl[b] ^= 1;
        tree.root = tree.merge(a, tree.merge(b, c));
    }
    tree.dfs(tree.root);
}
```

5.12.2 可持久化 FHQ

第一行包含一个正整数 n，表示操作的总数。

接下来 n 行，每行包含三个整数，第 i 行记为 v_i, opt_i, x_i。

v_i 表示基于的过去版本号，opt_i 表示操作的序号，x_i 表示参与操作的数值。

```
class FHQ { public:
    const int MAXM = 50 * MAXN; // 50倍注意！
    int ch[MAXM][2];
    int dat[MAXM], siz[MAXM], val[MAXM];
    int tot, root;

    void init() {   // 初始化
        tot = 0;
        root = 0;
        siz[0] = val[0] = 0;
    }

    inline void push_up(int rt) {   // 传递信息
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
```

```
15      }
16
17      inline int New(int v) { // 新建一个节点，value = v
18          val[++tot] = v;
19          dat[tot] = rand();
20          siz[tot] = 1;
21          ch[tot][0] = ch[tot][1] = 0;
22          return tot;
23      }
24
25      inline int Copy(int rt) {   // 复制点的信息
26          int newnode = ++tot;
27          val[tot] = val[rt], dat[tot] = dat[rt], siz[tot] = siz[rt];
28          ch[tot][0] = ch[rt][0], ch[tot][1] = ch[rt][1];
29          return newnode;
30      }
31
32      inline int merge(int x, int y) {    // 合并
33          if (!x || !y) return x + y;
34          if (dat[x] < dat[y]) {
35              int newnode = Copy(x);
36              ch[newnode][1] = merge(ch[newnode][1], y);
37              push_up(newnode);
38              return newnode;
39          } else {
40              int newnode = Copy(y);
41              ch[newnode][0] = merge(x, ch[newnode][0]);
42              push_up(newnode);
43              return newnode;
44          }
45      }
46
47      inline void split(int rt, int v, int &x, int &y) { // 按照权值进行分裂
48          if (!rt) x = y = 0;
49          else {
50              if (val[rt] <= v) {
51                  x = Copy(rt);
52                  split(ch[x][1], v, ch[x][1], y);
53                  push_up(x);
54              } else {
55                  y = Copy(rt);
56                  split(ch[y][0], v, x, ch[y][0]);
57                  push_up(y);
58              }
59          }
60      }
61
62      void del(int &rt, int v) {  // 删除value为v的数
63          int x = 0, y = 0, z = 0;
64          split(rt, v, x, z);
65          split(x, v - 1, x, y);
66          y = merge(ch[y][0], ch[y][1]);
67          rt = merge(x, merge(z, y));
68      }
69
70      void insert(int &rt, int v) {   // 插入value为v的数
```

```
        int x = 0, y = 0, z = 0;
        split(rt, v, x, y);
        z = New(v);
        rt = merge(x, merge(z, y));
    }

    int get_val(int rt, int k) {    // 得到第k大的数（从小到大）的value
        if (k == siz[ch[rt][0]] + 1) return val[rt];
        else if (k <= siz[ch[rt][0]]) return get_val(ch[rt][0], k);
        else return get_val(ch[rt][1], k - siz[ch[rt][0]] - 1);
    }

    int get_Kth(int &rt, int v) {   // 查询v的排名
        int x, y;
        split(rt, v - 1, x, y);
        int ans = siz[x] + 1;
        rt = merge(x, y);
        return ans;
    }

    int get_pre(int &rt, int v) {   // 求前驱，若不存在返回-2147483647
        int x, y, ans;
        split(rt, v - 1, x, y);
        if (!x) return -2147483647;
        ans = get_val(x, siz[x]);
        rt = merge(x, y);
        return ans;
    }

    int get_next(int &rt, int v) {  // 求后驱，若不存在返回2147483647
        int x, y, ans;
        split(rt, v, x, y);
        if (!y) return 2147483647;
        ans = get_val(y, 1);
        rt = merge(x, y);
        return ans;
    }
} tree;

int root[MAXN];

int main() {
    int q; scanf("%d", &q);
    tree.init();
    for (int i = 1; i <= q; i++) {
        int ver, opt, x;
        scanf("%d%d%d", &ver, &opt, &x);
        root[i] = root[ver];
        switch (opt) {
            // 插入x
            case 1: tree.insert(root[i], x); break;
            // 删除x（若有多个相同的数，应只删除一个，如果没有请忽略该操作）
            case 2: tree.del(root[i], x); break;
            // 查询x的排名（排名定义为比当前数小的数的个数+1）
            case 3: printf("%d\n", tree.get_Kth(root[i], x)); break;
            // 查询排名为x的数
```

```
            case 4: printf("%d\n", tree.get_val(root[i], x)); break;
            // 求x的前驱（前驱定义为小于x，且最大的数，如不存在输出-2147483647）
            case 5: printf("%d\n", tree.get_pre(root[i], x)); break;
            // 求x的后继（后继定义为大于x，且最小的数，如不存在输出2147483647）
            default: printf("%d\n", tree.get_next(root[i], x));
        }
    }

}
```

5.13 树套树

5.13.1 带修主席树

```
/*
  input       output
  5 3
  3 2 1 4 7
  Q 1 4 3     3
  C 2 6
  Q 2 5 3     6
*/
class BIT { public:
    // HJT begin
    int ch[MAXN * 400][2], sum[MAXN * 400], tot = 0;
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int update(int rt, int pos, int val, int be, int en) {
        int nrt = ++tot;
        ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        if (be == en) {
            sum[nrt] = sum[rt] + val;
            return nrt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
    }
    // HJT end
    int n, c_len, root[MAXN];
    void init(int _n, int _c_len) {
        c_len = _c_len, n = _n;
        for (int i = 1; i <= c_len; i++) root[i] = i;
        tot = c_len;
    }
    inline int lowbit(int x) { return x & (-x); }
```

```
    void insert(int pos, int pos_val, int val) {
        for (int i = pos; i <= n; i += lowbit(i)) root[i] = update(root[i], pos_val, val, 1,
            c_len);
    }
    int t1[MAXN], t2[MAXN], n1, n2;
    inline int Kth(int be, int en, int k) {
        if (be >= en) return be;
        int mid = (be + en) >> 1, delta = 0;

        for (int i = 1; i <= n1; i++) delta -= sum[ch[t1[i]][0]];
        for (int i = 1; i <= n2; i++) delta += sum[ch[t2[i]][0]];
        if (delta >= k) {
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][0];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][0];
            return Kth(be, mid, k);
        } else {
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][1];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][1];
            return Kth(mid + 1, en, k - delta);
        }
    }
    int query(int l, int r, int k) {
        n1 = n2 = 0;
        for (int i = l - 1; i >= 1; i -= lowbit(i)) t1[++n1] = root[i];
        for (int i = r; i >= 1; i -= lowbit(i)) t2[++n2] = root[i];
        return Kth(1, c_len, k);
    }
} tree;
int ai[MAXN];
int opt[MAXN], l[MAXN], r[MAXN], k[MAXN], x[MAXN], y[MAXN];
int main() {
    int n, m; cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> ai[i];
    for (int i = 1; i <= n; i++) Discrete::insert(ai[i]);
    for (int i = 1; i <= m; i++) {
        char op; cin >> op;
        if (op == 'Q') {
            opt[i] = 1; cin >> l[i] >> r[i] >> k[i];
        } else {
            opt[i] = 2; cin >> x[i] >> y[i];
            Discrete::insert(y[i]);
        }
    }
    Discrete::init();
    for (int i = 1; i <= n; i++) ai[i] = Discrete::val2id(ai[i]);
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 2) y[i] = Discrete::val2id(y[i]);
    }
    tree.init(n, Discrete::blen);

    for (int i = 1; i <= n; i++) tree.insert(i, ai[i], 1);
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 1) {  // 表示查询下标在区间[l,r]中的第k小的数
            cout << Discrete::id2val(tree.query(l[i], r[i], k[i])) << '\n';
        } else {  // 表示将a[x]改为y
            tree.insert(x[i], ai[x[i]], -1);
```

```
            ai[x[i]] = y[i];
            tree.insert(x[i], ai[x[i]], 1);
        }
    }
}
```

5.13.2 区间修改区间查询第 K 大（[ZJOI2013]K 大数查询）

```
/*
  树状数组套主席套线段树
  可重集的并是不去除重复元素的，如{1,1,4} & {5,1,4}={1,1,4,5,1,4}
  input     output
  2 5
  1 1 2 1
  1 1 2 2
  2 1 1 2   1
  2 1 1 1   2
  2 1 2 3   1
*/
class BIT { public:
    class SEG { public:
        int tot_rt, ch[MAXN * 400][2]; ll lazy[MAXN * 400], sum[MAXN * 400];
#define lson ch[rt][0]
#define rson ch[rt][1]
        inline void push_up(int rt) {
            sum[rt] = sum[lson] + sum[rson];
        }
        inline void push_down(int rt, int len_left, int len_right) {
            if (lazy[rt]) {
                if (!ch[rt][0]) ch[rt][0] = ++tot_rt;
                if (!ch[rt][1]) ch[rt][1] = ++tot_rt;
                sum[lson] += lazy[rt] * len_left, sum[rson] += lazy[rt] * len_right;
                lazy[lson] += lazy[rt], lazy[rson] += lazy[rt];
                lazy[rt] = 0;
            }
        }
        int change(int rt, int L, int R, int val, int be, int en) {
            if (!rt) rt = ++tot_rt;
            if (L <= be && R >= en) {
                sum[rt] += (ll) (en - be + 1);
                lazy[rt] += (ll) val;
                return rt;
            }
            int mid = (be + en) >> 1;
            push_down(rt, mid - be + 1, en - (mid + 1) + 1);
            if (L <= mid) lson = change(lson, L, R, val, be, mid);
            if (R > mid) rson = change(rson, L, R, val, mid + 1, en);
            push_up(rt);
            return rt;
        }
        ll query(int rt, int L, int R, int be, int en) {
            if (!rt) return 0;
            if (L <= be && R >= en) return sum[rt];
            int mid = (be + en) >> 1;
```

```
            push_down(rt, mid - be + 1, en - (mid + 1) + 1);
            ll ans = 0;
            if (L <= mid) ans += query(lson, L, R, be, mid);
            if (R > mid) ans += query(rson, L, R, mid + 1, en);
            return ans;
        }
    } seg;

    int n, c_len, root[MAXN];
    void init(int _n, int _c_len) {
        c_len = _c_len, n = _n;
        for (int i = 1; i <= c_len; i++) root[i] = i;
        seg.tot_rt = _c_len;
    }
    inline int lowbit(int x) { return x & (-x); }
    inline int log(int x) { return 1ll << (int) (log2(x)); }
    void insert(int l, int r, int c) {
        for (int i = c_len - c + 1; i <= c_len; i += lowbit(i)) seg.change(root[i], l, r, 1,
            1, n);
    }
    int query(int l, int r, ll k) {
        int ans = 0;
        ll sum = 0;
        for (int i = log(c_len); i != 0; i >>= 1) {
            if (ans + i > c_len) continue;
            ll tmp = seg.query(root[ans + i], l, r, 1, n) + sum;
            if (tmp < k) ans += i, sum = tmp;
        }
        ans++;
        return c_len - ans + 1;
    }
} tree;

int opt[MAXN], l[MAXN], r[MAXN]; ll c[MAXN];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%lld", &opt[i], &l[i], &r[i], &c[i]);
        if (opt[i] == 1) Discrete::push(c[i]);
    }
    Discrete::init(); // 离散化
    int c_len = Discrete::blen;
    tree.init(n, c_len);  // n为[l,r]，c_len表示离散化后数字个数
    for (int i = 1; i <= m; i++) {
        if (opt[i] == 1) {  // 表示将c加入到编号在[l,r]内的集合中
            tree.insert(l[i], r[i], Discrete::val2id(c[i]));
        } else {  // 表示查询编号在[l,r]内的集合的并集中，第c大的数是多少
            printf("%lld\n", Discrete::id2val(tree.query(l[i], r[i], c[i])));
        }
    }
}
```

5.14 笛卡尔树

5.14.1 建树

```
// p[i]存放[1, n]数据
class cartesian {
    int stk[MAXN], lson[MAXN], rson[MAXN];
    void build() {
        int tmp = 0, x;
        for (int i = 1; i <= n; i++) {
            x = tmp;
            while (x && p[stk[x]] > p[i]) x--;
            if (x) rson[stk[x]] = i;
            if (x < tmp) lson[i] = stk[x + 1];
            stk[++x] = i;
            tmp = x;
        }
    }
} ;
```

5.15 动态树 LCT

5.15.1 lct 连链、断链、更改点权、查询链上点权异或和

```
class LCT { public:
    int val[MAXN], sum[MAXN];
    int st_top, st[MAXN];   // stack操作
    int fa[MAXN], ch[MAXN][2];
    bool rev[MAXN];

    inline bool isroot(int x) { // 判断x是否为一个splay的根
        return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
    }
    inline void push_up(int x) {
        int l = ch[x][0], r = ch[x][1];
        sum[x] = sum[l] ^ sum[r] ^ val[x];  // 记录链上异或值
    }
    inline void push_down(int x) {
        int l = ch[x][0], r = ch[x][1];
        if (rev[x]) {
            if (l) swap(ch[l][0], ch[l][1]), rev[l] ^= 1;
            if (r) swap(ch[r][0], ch[r][1]), rev[r] ^= 1;
            rev[x] = 0;
        }
    }
    inline void rotate(int x) { // x向上旋转
        int y = fa[x], z = fa[y], l, r;
        if (ch[y][0] == x) l = 0;
        else l = 1;
        r = l ^ 1;
        if (!isroot(y)) {
            if (ch[z][0] == y) ch[z][0] = x;
            else ch[z][1] = x;
        }
```

```
        fa[x] = z, fa[y] = x;
        fa[ch[x][r]] = y;
        ch[y][l] = ch[x][r];
        ch[x][r] = y;
        push_up(y), push_up(x);
    }
    inline void splay(int x) {  // 使得x成为当前splay中的根
        st_top = 0;
        st[++st_top] = x;
        for (int i = x; !isroot(i); i = fa[i]) st[++st_top] = fa[i];
        for (int i = st_top; i; i--) push_down(st[i]);
        while (!isroot(x)) {
            int y = fa[x], z = fa[y];
            if (!isroot(y)) {
                if (ch[y][0] == x ^ ch[z][0] == y) rotate(x);
                else rotate(y);
            }
            rotate(x);
        }
    }
    inline void access(int x) { //把x到根节点的路径搞成一个splay
        for (int i = 0; x; i = x, x = fa[x])
            splay(x), ch[x][1] = i, push_up(x);
    }
    inline void makeroot(int x) {   // 使得p成为原树的根
        access(x);
        splay(x);
        swap(ch[x][0], ch[x][1]), rev[x] ^= 1;
    }
    inline int find(int x) {    // 找到x在原树的根
        access(x);
        splay(x);
        while (ch[x][0]) x = ch[x][0];
        splay(x);   // 非常重要！一定注意！
        return x;
    }

    void split(int x, int y) {  // 拉出x-y的路径搞成一个splay
        makeroot(x);
        access(y);
        splay(y);   // y为根, call: tree.sum[y]
    }
    void link(int x, int y) {   // 连接x,y
        makeroot(x);
        if (find(y) == x) return;
        fa[x] = y;
        return;
    }
    void cut(int x, int y) {    // 断开x,y
        makeroot(x);
        if (find(y) != x || fa[y] != x || ch[y][0]) return; // 两条不连通
        ch[x][1] = fa[y] = 0;
        push_up(x);
        return ;
    }
    void change(int x, int v) { // 修改某一点的值
```

```
        splay(x);
        val[x] = v;
        push_up(x);
    }
    bool isconnect(int x, int y) {  // 判断两点是否连通
        makeroot(x);
        if (find(y) != x) return 0; // 两条不连通
        else return 1;
    }
} tree;
```

5.15.2 树上路径染色（[SDOI2011] 染色）

```
// 颜色段的定义是极长的连续相同颜色被认为是一段。例如112221由三段组成：11、222、1。
class LCT { public:
    int st_top, st[MAXN];
    int fa[MAXN], ch[MAXN][2];
    bool rev[MAXN];
    int col[MAXN], colL[MAXN], colR[MAXN], sum[MAXN];
    int lazy[MAXN];

    inline void push_up(int x) {
        int l = ch[x][0], r = ch[x][1];
        colL[x] = l ? colL[l] : col[x];
        colR[x] = r ? colR[r] : col[x];
        if (l && r) sum[x] = sum[l] + sum[r] + 1 - (colR[l] == col[x]) - (colL[r] == col[x]);
        if (l && !r) sum[x] = sum[l] + 1 - (colR[l] == col[x]);
        if (!l && r) sum[x] = sum[r] + 1 - (colL[r] == col[x]);
        if (!l && !r) sum[x] = 1;
    }

    inline void push_down(int x) {
        int l = ch[x][0], r = ch[x][1];
        if (rev[x]) {
            if (l) swap(ch[l][0], ch[l][1]), swap(colL[l], colR[l]), rev[l] ^= 1;
            if (r) swap(ch[r][0], ch[r][1]), swap(colL[r], colR[r]), rev[r] ^= 1;
            rev[x] = 0;
        }
        if (lazy[x]) {
            if (l) colL[l] = colR[l] = col[l] = lazy[x], sum[l] = 1, lazy[l] = lazy[x];
            if (r) colL[r] = colR[r] = col[r] = lazy[x], sum[r] = 1, lazy[r] = lazy[x];
            lazy[x] = 0;
        }
    }

    inline void makeroot(int x) {   // 使得p成为原树的根
        access(x);
        splay(x);
        swap(ch[x][0], ch[x][1]), swap(colL[x], colR[x]), rev[x] ^= 1;
    }

} tree;

int main() {
```

```
    int n, q;   cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> tree.col[i];
        tree.colL[i] = tree.colR[i] = tree.col[i];
        tree.sum[i] = 1;
    }
    for (int i = 2; i <= n; i++) {
        int u, v;   cin >> u >> v;
        tree.link(u, v);
    }
    while (q--) {
        char opt;   cin >> opt;
        if (opt == 'C') {   // 将节点u到节点v的路径上的所有点（包括u和v）都染成颜色c
            int u, v, c;    cin >> u >> v >> c;
            tree.split(u, v);
            tree.colL[v] = tree.colR[v] = tree.col[v] = c, tree.sum[v] = 1, tree.lazy[v] = c;
        } else {    // 询问节点u到节点v的路径上的颜色段数量
            int u, v;   cin >> u >> v;
            tree.split(u, v);
            printf("%d\n", tree.sum[v]);
        }
    }
}
```

5.15.3 SAM+ 线段树 +LCT 离线统计区间本质不同字串个数

时间复杂度：$O(n\log^2 n + q\log n)$.

```
/*
    input       output
    aababc
    3
    1 2         2
    2 4         5
    3 6         9
*/
namespace SAM_SEG_LCT {
    SEG seg, SAM sam;   // 后缀自动机长度为两倍
    class LCT { public:
        int val[MAXN], lazy[MAXN];
        int st_top, st[MAXN];   // stack操作
        int fa[MAXN], ch[MAXN][2];

        inline bool isroot(int x) { // 判断x是否为一个splay的根
            return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
        }
        inline void push_up(int x) {    }
        inline void push_down(int x) {
            int l = ch[x][0], r = ch[x][1];
            if (lazy[x]) {
                if (l) val[l] = lazy[x], lazy[l] = lazy[x];
                if (r) val[r] = lazy[x], lazy[r] = lazy[x];
                lazy[x] = 0;
            }
        }
        inline void rotate(int x) { // x向上旋转
```

```
            int y = fa[x], z = fa[y], l, r;
            if (ch[y][0] == x) l = 0;
            else l = 1;
            r = l ^ 1;
            if (!isroot(y)) {
                if (ch[z][0] == y) ch[z][0] = x;
                else ch[z][1] = x;
            }
            fa[x] = z, fa[y] = x;
            fa[ch[x][r]] = y;
            ch[y][l] = ch[x][r];
            ch[x][r] = y;
            push_up(y), push_up(x);
        }
        inline void splay(int x) {  // 使得x成为当前splay中的根
            st_top = 0;
            st[++st_top] = x;
            for (int i = x; !isroot(i); i = fa[i]) st[++st_top] = fa[i];
            for (int i = st_top; i; i--) push_down(st[i]);
            while (!isroot(x)) {
                int y = fa[x], z = fa[y];
                if (!isroot(y)) {
                    if (ch[y][0] == x ^ ch[z][0] == y) rotate(x);
                    else rotate(y);
                }
                rotate(x);
            }
        }

        void access(int x, int p) { //把x到根节点的路径搞成一个splay，主要是这里修改
            int y = 0;
            while (x) {
                splay(x);
                if (int k = val[x]) seg.change(1, k - sam.maxlen[x] + 1, k -
                    sam.maxlen[fa[x]], -1);
                ch[x][1] = y, y = x, x = fa[x];
            }
            val[y] = p, lazy[y] = p;
            seg.change(1, 1, p, 1); // 线段树区间修改chang(rt, L, R, val);
        }

        void build() {
            st_top = 0;
            fa[1] = ch[1][0] = ch[1][1] = val[1] = lazy[1] = 0;
            fa[0] = ch[0][0] = ch[0][1] = val[0] = lazy[0] = 0;
            for (int i = 2; i <= sam.rt; i++) {
                val[i] = lazy[i] = 0;
                ch[i][0] = ch[i][1] = 0;
                fa[i] = sam.link[i];
            }
        }
    } lct;
}

using namespace SAM_SEG_LCT;
struct Query {
```

```
    int l, r, id;
    bool operator<(const Query &tb) const { return r < tb.r;}
} query[MAXN];

char str[MAXN]; int endpos[MAXN];   ll res[MAXN];
int main() {
    scanf("%s", str + 1);   int len = strlen(str + 1);
    sam.init(); int sam_last = 1;
    for (int i = 1; i <= len; i++) sam_last = endpos[i] = sam.insert(str[i] - 'a', sam_last);
    lct.build(), seg.build(1, 1, len);

    int q;  scanf("%d", &q);
    for (int i = 1; i <= q; i++) scanf("%d%d", &query[i].l, &query[i].r), query[i].id = i;
    sort(query + 1, query + 1 + q);

    int pos = 1;
    for (int i = 1; i <= q; i++) {
        while (pos <= query[i].r) lct.access(endpos[pos], pos), pos++;
        res[query[i].id] = seg.query(1, query[i].l, query[i].r);
    }
    for (int i = 1; i <= q; i++) printf("%lld\n", res[i]);  // 输出答案
}
```

5.15.4 主席树 +LCT 在线查询区间连通块个数

```
struct Edge {
    int u, v;
} e[MAXM];

class LCT { public:
    int val[MAXN+MAXM], minn_id[MAXN+MAXM];
    int stk_top, stk[MAXN+MAXM];
    int fa[MAXN+MAXM], ch[MAXN+MAXM][2];
    bool rev[MAXN+MAXM];

    inline bool isroot(int x) {
        return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
    }
    inline void push_up(int x) {
        int l = ch[x][0], r = ch[x][1];
        minn_id[x] = x;
        if (val[minn_id[l]] < val[minn_id[x]]) minn_id[x] = minn_id[l];
        if (val[minn_id[r]] < val[minn_id[x]]) minn_id[x] = minn_id[r];
    }
    inline void split(int x, int y) {
        makeroot(x);
        access(y);
        splay(y);
    }

    int query(int x, int y) {
        split(x, y);
        return minn_id[y];
    }
```

```
} lct;

class HJT { public:
    int ch[MAXM * 70][2], sum[MAXM * 70];
    int tot;
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int query(int lrt, int rrt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return sum[rrt] - sum[lrt];
        int mid = (be + en) >> 1;
        int ans = 0;
        if (L <= mid) ans += query(ch[lrt][0], ch[rrt][0], L, R, be, mid);
        if (R > mid) ans += query(ch[lrt][1], ch[rrt][1], L, R, mid+1, en);
        return ans;
    }
} tree;

int del[MAXM], root[MAXM];
int main() {
    int n, m, q, type;
    scanf("%d%d%d%d", &n, &m, &q, &type);   // type标识在线参数

    lct.val[0] = inf;  // init
    for (int i = 1; i <= n; i++) lct.minn_id[i] = i, lct.val[i] = inf;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &e[i].u, &e[i].v);
    }

    // pre begin
    int tot = n;
    for (int i = 1; i <= m; i++) {
        int u = e[i].u, v = e[i].v;
        if (u == v) {
            del[i] = i; continue;
        }
        if (lct.find(u) == lct.find(v)) {
            int tmp = lct.query(u, v), x = lct.val[tmp];
            del[i] = x;
            lct.cut(e[x].u, tmp), lct.cut(e[x].v, tmp);
        }
        tot++;
        lct.minn_id[tot] = tot, lct.val[tot] = i;
        lct.link(u, tot), lct.link(v, tot);
    }
    root[0] = 0;
    for (int i = 1; i <= m; i++) {
        del[i]++; // [0, m] -> [1, m+1]
        root[i] = tree.update(root[i - 1], del[i], 1, 1, m + 1);
    }
    // pre end
    int lastans = 0;
    while (q--) {
        int l, r;
        scanf("%d%d", &l, &r);
        lastans = n - tree.query(root[l-1], root[r], 1, l, 1, m+1);
```

```
        printf("%d\n", lastans);
    }
}
```

5.16 KD 树

5.16.1 平面最近点对

时间复杂度：单次查询最近点的时间复杂度 $O(n)$.

```
const int MAXN = 2e5 + 5;   // 点的个数
class KD {
public:
    struct node {
        double x, y;
    } T[MAXN];
    int ch[MAXN][2];
    double L[MAXN], R[MAXN], D[MAXN], U[MAXN];
    inline void push_up(int rt) {
        L[rt] = R[rt] = T[rt].x;
        D[rt] = U[rt] = T[rt].y;
        if (ch[rt][0]) {
            L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt], R[ch[rt][0]]),
            D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt], U[ch[rt][0]]);
        }
        if (ch[rt][1]) {
            L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt], R[ch[rt][1]]),
            D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt], U[ch[rt][1]]);
        }
    }

    int d[MAXN];
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
        for (int i = l; i <= r; i++) av1 += T[i].x, av2 += T[i].y;
        av1 /= (r - l + 1);
        av2 /= (r - l + 1);
        for (int i = l; i <= r; i++) va1 += (av1 - T[i].x) * (av1 - T[i].x), va2 += (av2 -
            T[i].y) * (av2 - T[i].y);
        if (va1 > va2)
            d[mid] = 1, nth_element(T + l, T + mid, T + r + 1,
                                    [&](const node &ta, const node &tb) { return ta.x < tb.x;
                                        });
        else
            d[mid] = 2, nth_element(T + l, T + mid, T + r + 1,
                                    [&](const node &ta, const node &tb) { return ta.y < tb.y;
                                        });
        ch[mid][0] = build(l, mid - 1);
        ch[mid][1] = build(mid + 1, r);
        push_up(mid);
        return mid;
    }

```

```
    double f(int a, int b) {
        double ans = 0;
        if (L[b] > T[a].x) ans += (L[b] - T[a].x) * (L[b] - T[a].x);
        if (R[b] < T[a].x) ans += (T[a].x - R[b]) * (T[a].x - R[b]);
        if (D[b] > T[a].y) ans += (D[b] - T[a].y) * (D[b] - T[a].y);
        if (U[b] < T[a].y) ans += (T[a].y - U[b]) * (T[a].y - U[b]);
        return ans;
    }
    double ans;
    double dist(int a, int b) {
        return (T[a].x - T[b].x) * (T[a].x - T[b].x) + (T[a].y - T[b].y) * (T[a].y - T[b].y);
    }
    void query(int l, int r, int rt) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        if (mid != rt) ans = min(ans, dist(rt, mid));
        if (l == r) return ;
        double distl = f(rt, ch[mid][0]), distr = f(rt, ch[mid][1]);
        if (distl < ans && distr < ans) {
            if (distl < distr) {
                query(l, mid - 1, rt);
                if (distr < ans) query(mid + 1, r, rt);
            } else {
                query(mid + 1, r, rt);
                if (distl < ans) query(l, mid - 1, rt);
            }
        } else {
            if (distl < ans) query(l, mid - 1, rt);
            if (distr < ans) query(mid + 1, r, rt);
        }
    }
    void init() { ans = inf_ll; }
    double get_res() { return ans; }
} tree;

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%lf%lf", &tree.T[i].x, &tree.T[i].y);
    tree.init();
    tree.build(1, n);
    for (int i = 1; i <= n; i++) {
        tree.query(1, n, i);
    }
    printf("%.4lf\n", sqrt(tree.get_res()));
}
```

5.16.2 K 远点对 ([CQOI2016])

已知平面内 N 个点的坐标，求欧氏距离下的第 K 远点对。

两个点的欧氏距离为 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

原题数据范围：$N \le 1e5, 1 \le K \le 100$

时间复杂度：$O(kn \log n)$.

```
/*
    input
```

```
    10 5
    0 0, 0 1, 1 0, 1 1, 2 0, 2 1, 1 2, 0 2, 3 0, 3 1
    ouput
    9
*/
const int MAXN = 1e5 + 5;   // 点的个数
class KD {
public:
    priority_queue<ll, vector<ll>, greater<ll> > q;
    struct node {
        int x, y;
    } T[MAXN];
    int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];

    inline void push_up(int rt) {
        L[rt] = R[rt] = T[rt].x;
        D[rt] = U[rt] = T[rt].y;
        if (ch[rt][0]) {
            L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt], R[ch[rt][0]]),
            D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt], U[ch[rt][0]]);
        }
        if (ch[rt][1]) {
            L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt], R[ch[rt][1]]),
            D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt], U[ch[rt][1]]);
        }
    }

    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
        for (int i = l; i <= r; i++) av1 += T[i].x, av2 += T[i].y;
        av1 /= (r - l + 1);
        av2 /= (r - l + 1);
        for (int i = l; i <= r; i++) va1 += (av1 - T[i].x) * (av1 - T[i].x), va2 += (av2 -
            T[i].y) * (av2 - T[i].y);
        if (va1 > va2)
            nth_element(T + l, T + mid, T + r + 1, [&](const node &ta, const node &tb) {
                return ta.x < tb.x; });
        else nth_element(T + l, T + mid, T + r + 1, [&](const node &ta, const node &tb) {
            return ta.y < tb.y; });
        ch[mid][0] = build(l, mid - 1);
        ch[mid][1] = build(mid + 1, r);
        push_up(mid);
        return mid;
    }

    ll sq(int x) { return (ll) x * x; }
    ll dist(int a, int b) {
        return max(sq(T[a].x - L[b]), sq(T[a].x - R[b])) + max(sq(T[a].y - D[b]), sq(T[a].y -
            U[b]));
    }
    void query(int l, int r, int rt) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        ll tmp = sq(T[mid].x - T[rt].x) + sq(T[mid].y - T[rt].y);
```

```
        if (tmp > q.top()) q.pop(), q.push((tmp));
        ll distl = dist(rt, ch[mid][0]), distr = dist(rt, ch[mid][1]);
        if (distl > q.top() && distr > q.top()) {
            if (distl > distr) {
                query(l, mid - 1, rt);
                if (distr > q.top()) query(mid + 1, r, rt);
            } else {
                query(mid + 1, r, rt);
                if (distl > q.top()) query(l, mid - 1, rt);
            }
        } else {
            if (distl > q.top()) query(l, mid - 1, rt);
            if (distr > q.top()) query(mid + 1, r, rt);
        }
    }
    void init(int k) {
        k *= 2;
        for (int i = 1; i <= k; i++) q.push(0);
    }
    ll get_res() {
        return q.top();
    }
} tree;

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    tree.init(k);
    for (int i = 1; i <= n; i++) scanf("%d%d", &tree.T[i].x, &tree.T[i].y);
    tree.build(1, n);
    for (int i = 1; i <= n; i++) {
        tree.query(1, n, i);
    }
    printf("%lld\n", tree.get_res());
}
```

5.16.3 高维空间上的操作

在一个初始值全为 0 的 $n \times n$ 的二维矩阵上，进行若干次操作，每次操作为以下两种之一：

1 x y A 将坐标 (x, y) 上的数加上 A。

2 x1 y1 x2 y2 输出以 $(x1, y1)$ 为左下角，$(x2, y2$ 为右上角的矩形内（包括矩形边界）的数字和。

原题数据范围：$1 \leq n \leq 5e5, 1 \leq q \leq 2e5$

时间复杂度：单次查询时间最优 $O(\log n)$，最坏 $O(\sqrt{n})$。将结论扩展至 k 维，最坏复杂度 $O(n^{1-\frac{1}{k}})$

```
/*
    intput      output
    4
    1 2 3 3
    2 1 1 3 3   3
    1 1 1 1
    2 1 1 0 7   5
    3
*/
const int MAXN = 2e5 + 5;   // 操作次数
class KD {
public:
```

```
13      struct node {
14          int x, y, v;
15      } T[MAXN];
16      int ch[MAXN][2], L[MAXN], R[MAXN], D[MAXN], U[MAXN];
17      int siz[MAXN], sum[MAXN], g[MAXN], d[MAXN];
18
19      inline void push_up(int rt) {
20          siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
21          sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]] + T[rt].v;
22          L[rt] = R[rt] = T[rt].x;
23          U[rt] = D[rt] = T[rt].y;
24          if (ch[rt][0]) {
25              L[rt] = min(L[rt], L[ch[rt][0]]), R[rt] = max(R[rt], R[ch[rt][0]]),
26              D[rt] = min(D[rt], D[ch[rt][0]]), U[rt] = max(U[rt], U[ch[rt][0]]);
27          }
28          if (ch[rt][1]) {
29              L[rt] = min(L[rt], L[ch[rt][1]]), R[rt] = max(R[rt], R[ch[rt][1]]),
30              D[rt] = min(D[rt], D[ch[rt][1]]), U[rt] = max(U[rt], U[ch[rt][1]]);
31          }
32      }
33
34      int build(int l, int r) {
35          if (l > r) return 0;
36          int mid = (l + r) >> 1;
37          double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
38          for (int i = l; i <= r; i++) av1 += T[g[i]].x, av2 += T[g[i]].y;
39          av1 /= (r - l + 1);
40          av2 /= (r - l + 1);
41          for (int i = l; i <= r; i++) {
42              va1 += (av1 - T[g[i]].x) * (av1 - T[g[i]].x),
43                      va2 += (av2 - T[g[i]].y) * (av2 - T[g[i]].y);
44          }
45          if (va1 > va2)
46              nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) { return T[ta].x <
                    T[tb].x; }), d[g[mid]] = 1;
47          else nth_element(g + l, g + mid, g + r + 1, [&](int ta, int tb) { return T[ta].y <
                T[tb].y; }), d[g[mid]] = 2;
48          ch[g[mid]][0] = build(l, mid - 1);
49          ch[g[mid]][1] = build(mid + 1, r);
50          push_up(g[mid]);
51          return g[mid];
52      }
53
54      const double bad_para = 0.725;
55      int tot;
56      bool bad(int rt) {
57          return bad_para * siz[rt] <= (double) max(siz[ch[rt][0]], siz[ch[rt][1]]);
58      }
59      void dfs(int rt) {
60          if (!rt) return;
61          dfs(ch[rt][0]);
62          g[++tot] = rt;
63          dfs(ch[rt][1]);
64      }
65      void rebuild(int &rt) {
66          tot = 0;
```

```
        dfs(rt);
        rt = build(1, tot);
    }
    void insert(int &rt, int v) {
        if (!rt) {
            rt = v;
            push_up(rt);
            return;
        }
        if (d[rt] == 1) {
            if (T[v].x <= T[rt].x) insert(ch[rt][0], v);
            else insert(ch[rt][1], v);
        } else {
            if (T[v].y <= T[rt].y) insert(ch[rt][0], v);
            else insert(ch[rt][1], v);
        }
        push_up(rt);
        if (bad(rt)) rebuild(rt);
    }
    int query(int rt, int xl, int xr, int yl, int yr) {
        if (!rt ||  xr < L[rt] || xl > R[rt] || yr < D[rt] || yl > U[rt]) return 0;
        if (xl <= L[rt] && R[rt] <= xr &&  yl <= D[rt] && U[rt] <= yr) return sum[rt];
        int ans = 0;
        if (xl <= T[rt].x && T[rt].x <= xr && yl <= T[rt].y && T[rt].y <= yr) ans += T[rt].v;
        return query(ch[rt][0], xl, xr, yl, yr) + query(ch[rt][1], xl, xr, yl, yr) + ans;
    }

} tree;

int main() {
    int N, opt;
    scanf("%d", &N);
    int tot = 0, lastans = 0, root = 0;
    while (~scanf("%d", &opt)) {
        if (opt == 1) {
            tot++;
            scanf("%d%d%d", &tree.T[tot].x, &tree.T[tot].y, &tree.T[tot].v);
            tree.T[tot].x ^= lastans, tree.T[tot].y ^= lastans, tree.T[tot].v ^= lastans;
            tree.insert(root, tot);
        } else if (opt == 2) {
            int xl, yl, xr, yr;
            scanf("%d%d%d%d", &xl, &yl, &xr, &yr);
            xl ^= lastans, yl ^= lastans, xr ^= lastans, yr ^= lastans;
            lastans = tree.query(root, xl, xr, yl, yr);
            printf("%d\n", lastans);
        } else break;
    }
}
```

5.17 珂朵莉树/老司机树/ODT

5.17.1 set 实现珂朵莉树

1 l r x 将 $[l,r]$ 区间所有数加上 x

2 l r x 将 $[l,r]$ 区间所有数改成 x

3 l r x 输出将 $[l,r]$ 区间从小到大排序后的第 x 个数是的多少 (即区间第 x 小，数字大小相同算多次，保证 $1 \le x \le r-l+1$)

4 l r x y 输出 $[l,r]$ 区间每个数字的 x 次方的和模 y 的值 (即 $\sum_{i=l}^{r} a_i^x \% y$)

时间复杂度：用 set 实现 $O(n \log \log n)$

如果要保证复杂度正确，必须保证数据随机。

```
#define IT set<node>::iterator
const int MAXN = 1e5 + 5;
const int MOD7 = 1e9 + 7;

ll powmod(ll base, ll times, ll mod) {
    ll p = 1;
    ll ans = base % mod;
    while (times) {
        if (times & 1) p = p * ans % mod;
        ans = ans * ans % mod;
        times >>= 1;
    }
    return p;
}
class ODT {
public:
    struct node {
        int l, r;
        mutable ll val; // 玄学mutable注意!
        node() {}
        node(int _l, int _r = -1, ll _val = 0) { l = _l, r = _r, val = _val; }
        bool operator<(const node &tb) const { return l < tb.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 ul = it->l, ur = it->r; ll uv = it->val;
        st.erase(it);
        st.insert(node(ul, pos - 1, uv));
        return st.insert(node(pos, ur, uv)).first;
    }
    void add(int l, int r, ll v = 1) {  // 对一段区间加上一个数
        IT itr = split(r + 1), itl = split(l); // 一定要先split右再split左!
        for (; itl != itr; itl++) itl->val += v;
    }
    void assign_val(int l, int r, ll v = 0) {   // 对一段区间进行赋值
        IT itr = split(r + 1), itl = split(l); // 一定要先split右再split左!
        st.erase(itl, itr);
        st.insert(node(l, r, v));
    }
    ll rank(int l, int r, int k) {
        vector<pair<ll, int>> vp;
        IT itr = split(r + 1), itl = split(l); // 一定要先split右再split左!
        vp.clear();
        for (; itl != itr; itl++)
            vp.push_back(pair<ll, int>(itl->val, (itl->r) - (itl->l) + 1));
        sort(vp.begin(), vp.end());
        for (vector<pair<ll, int>>::iterator it = vp.begin(); it != vp.end(); it++) {
            k -= it->second;
```

```
            if (k <= 0) return it->first;
        }
        return -1ll;
    }
    ll sum(int l, int r, int times, int mod) {
        IT itr = split(r + 1), itl = split(l); // 一定要先split右再split左！
        ll ans = 0;
        for (; itl != itr; itl++)
            ans = (ans + (ll) (itl->r - itl->l + 1) * powmod(itl->val, (ll) times, (ll) mod) %
                mod) % mod;
        return ans;
    }
    void insert(int l, int r, ll v) {
        st.insert(node(l, r, v));
    }
} tree;

ll seed, vmax;
ll rnd() {
    ll ret = seed;
    seed = (seed * 7 + 13) % MOD7;
    return ret;
}

ll a[MAXN];
int main() {
    int n, m;
    scanf("%d%d%lld%lld", &n, &m, &seed, &vmax);
    for (int i = 1; i <= n; i++) {
        a[i] = (rnd() % vmax) + 1;
        tree.insert(i, i, a[i]);
    }
    tree.insert(n + 1, n + 1, 0);
    int lines = 0;
    for (int i = 1; i <= m; i++) {
        int opt = int(rnd() % 4) + 1, l = int(rnd() % n) + 1, r = int(rnd() % n) + 1;
        if (l > r) swap(l, r);
        int x, y;
        // 原题神奇的操作，可以不用理会
        if (opt == 3) x = int(rnd() % (r - l + 1)) + 1;
        else x = int(rnd() % vmax) + 1;
        if (opt == 4) y = int(rnd() % vmax) + 1;

        if (opt == 1) tree.add(l, r, x);
        else if (opt == 2) tree.assign_val(l, r, x);
        else if (opt == 3) printf("%lld\n", tree.rank(l, r, x));
        else printf("%lld\n", tree.sum(l, r, x, y));
    }
}
```

5.18 01 字典树

5.18.1 路径为点权异或值求最小生成树（CF888G）

input	output
4 1 2 3 4	8

```
class Trie {    public:
    int T[SIZE<<4][2], top;
    Trie() {
        top = 1;
        memset(T[0], 0, sizeof(T[0]));
    }
    void insert(int x) {    // call: tree.insert(x);
        int u = 0;
        for (int i = 30; ~i; i--) {
            int ch = (x >> i) & 1;
            if(!T[u][ch]) {
                memset(T[top], 0, sizeof(T[top]));
                T[u][ch] = top++;
            }
            u = T[u][ch];
        }
    }
    ll query(int rt1, int rt2, int dp) {
        if(dp < 0) return (ll)0;
        ll res1 = -1, res2 = -1;
        if(T[rt1][0] && T[rt2][0]) res1 = query(T[rt1][0], T[rt2][0], dp-1);
        if(T[rt1][1] && T[rt2][1]) res2 = query(T[rt1][1], T[rt2][1], dp-1);
        if(~res1 && ~res2) return std::min(res1, res2);
        if(~res1) return res1; if(~res2) return res2;
        if(T[rt1][0] && T[rt2][1]) res1 = query(T[rt1][0], T[rt2][1], dp-1) + (1 << dp);
        if(T[rt1][1] && T[rt2][0]) res2 = query(T[rt1][1], T[rt2][0], dp-1) + (1 << dp);
        if(~res1 && ~res2) return std::min(res1, res2);
        if(~res1) return res1; if(~res2) return res2;
    }
} tree;
ll res;
void dfs(int a, int b) {    // call: dfs(0, 30);
    if(b<0) return ;
    if(tree.T[a][0] && tree.T[a][1]) {
        res += 1ll * tree.query(tree.T[a][0], tree.T[a][1], b-1) + 1ll * (1 << b);
    }
    if(tree.T[a][0]) dfs(tree.T[a][0], b-1);
    if(tree.T[a][1]) dfs(tree.T[a][1], b-1);
}
```

5.18.2 可持久化 01 字典树

初始有 n 个数，有 m 个操作:

1 A x 添加操作，表示在序列末尾添加一个数 x ，序列的长度 $n+1$

Q l r x 询问操作，你需要找到一个位置 p ，满足 $l \leq p \leq r$，使得：$a[p] \oplus a[p+1] \oplus ... \oplus a[N] \oplus x$ 最大，输出最大是多少。

```
class HJT_01 {  public:
    int ch[MAXN * 70][2], sum[MAXN * 70];
    int tot;
    int update(int rt, int v, int dep) {
        int nrt = ++tot, tmp = nrt;
```

```
        for (int i = 30; i >= 0; i--) {
            sum[nrt] = sum[rt] + 1;  // 在原版本的基础上更新
            if ((v & (1 << i)) == 0) {
                if (!ch[nrt][0]) ch[nrt][0] = ++tot;
                ch[nrt][1] = ch[rt][1];
                nrt = ch[nrt][0];
                rt = ch[rt][0];
            } else {
                if (!ch[nrt][1]) ch[nrt][1] = ++tot;
                ch[nrt][0] = ch[rt][0];
                nrt = ch[nrt][1];
                rt = ch[rt][1];
            }
        }
        sum[nrt] = sum[rt] + 1;
        return tmp;
    }
    int query(int lrt, int rrt, int v) {
        int ans = 0;
        for (int i = 30; i >= 0; i--) {
            int t = ((v & (1 << i)) ? 1 : 0);
            if (sum[ch[rrt][!t]] - sum[ch[lrt][!t]]) {
                ans += (1 << i);
                lrt = ch[lrt][!t], rrt = ch[rrt][!t];
            } else lrt = ch[lrt][t], rrt = ch[rrt][t];
        }
        return ans;
    }
} tree;
int a[MAXN], pre[MAXN], root[MAXN];
char opt[5];
int main() {
    int n, m;   scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]), pre[i] = pre[i - 1] ^ a[i];
    int root_cnt = n;
    root[0] = 0;
    for (int i = 1; i <= n; i++) root[i] = tree.update(root[i - 1], pre[i], 30);
    while (m--) {
        scanf("%s", opt + 1);
        if (opt[1] == 'A') {
            root_cnt++;
            scanf("%d", &a[root_cnt]);
            pre[root_cnt] = pre[root_cnt - 1] ^ a[root_cnt];
            root[root_cnt] = tree.update(root[root_cnt - 1], pre[root_cnt], 30);
        } else {
            int l, r, x;
            scanf("%d%d%d", &l, &r, &x);
            l--, r--;
            if (l == r && l == 0) printf("%d\n", pre[root_cnt] ^ x);
            else printf("%d\n", tree.query(root[max(0, l - 1)], root[r], x ^ pre[root_cnt]));
        }
    }
}
```

5.19 左偏树（可并堆）

5.19.1 左偏树 $O(\log n)$

```
class LT {  public:
    int pool[MAXN], pool_cnt;

    int fa[MAXN];
    int find(int x) {
        if (x == fa[x]) return x;
        else return fa[x] = find(fa[x]);
    }

    int val[MAXN], ch[MAXN][2], dist[MAXN];

    int tot;
    inline int New() {
        return pool_cnt ? pool[pool_cnt--] : ++tot;
    }
    inline void Del(int &rt) {
        pool[++pool_cnt] = rt;
        fa[rt] = val[rt] = ch[rt][0] = ch[rt][1] = dist[rt] = 0;
        rt = 0;
    }
    void init() { tot = 0, pool_cnt = 0; }
    inline int Newnode(int v) { // 初始化左偏树节点
        int nrt = New();
        val[nrt] = v, ch[nrt][0] = ch[nrt][1] = dist[nrt] = 0;
        fa[nrt] = nrt;
        return nrt;
    }

    int merge(int x, int y) {   // 合并左偏树，
                                // call：root[fx] = tree.merge(root[fx], root[fy]);
        if (!x || !y) return x + y;
        if (val[x] == val[y] ? x > y : val[x] > val[y]) swap(x, y); // 小根堆
        ch[x][1] = merge(ch[x][1], y);
        if (dist[ch[x][0]] < dist[ch[x][1]]) swap(ch[x][0], ch[x][1]);
        fa[ch[x][0]] = fa[ch[x][1]] = fa[x] = x;
        dist[x] = dist[ch[x][1]] + 1;
        return x;
    }
    int insert(int rt, int v) { // call：root[x] = tree.insert(root[x], y);
        return merge(rt, Newnode(v));
    }
    int pop(int rt) {   // call：root[fx] = tree.pop(root[fx]);
        int tl = ch[rt][0], tr = ch[rt][1];
        Del(rt);
        return merge(tl, tr);
    }
    bool isempty(int x) {   // call：tree.isempty(root[fx])
                            // 为空返回1，不为空返回0
        return x == 0;
    }
    int top(int x) {    // call：tree.top(root[fx])
```

```
        return val[x];
    }

} tree;
```

5.19.2 带 push_down 操作的左偏树子树节点合并（[JLOI2015] 城池攻占）

小铭铭最近获得了一副新的桌游，游戏中需要用 m 个骑士攻占 n 个城池。这 n 个城池用 1 到 n 的整数表示。除 1 号城池外，城池 i 会受到另一座城池 fi 的管辖，其中 fi <i。也就是说，所有城池构成了一棵有根树。这 m 个骑士用 1 到 m 的整数表示，其中第 i 个骑士的初始战斗力为 si，第一个攻击的城池为 ci。

每个城池有一个防御值 hi，如果一个骑士的战斗力大于等于城池的生命值，那么骑士就可以占领这座城池；否则占领失败，骑士将在这座城池牺牲。占领一个城池以后，骑士的战斗力将发生变化，然后继续攻击管辖这座城池的城池，直到占领 1 号城池，或牺牲为止。

除 1 号城池外，每个城池 i 会给出一个战斗力变化参数 ai;vi。若 ai =0，攻占城池 i 以后骑士战斗力会增加 vi；若 ai =1，攻占城池 i 以后，战斗力会乘以 vi。注意每个骑士是单独计算的。也就是说一个骑士攻击一座城池，不管结果如何，均不会影响其他骑士攻击这座城池的结果。

现在的问题是，对于每个城池，输出有多少个骑士在这里牺牲；对于每个骑士，输出他攻占的城池数量。

```
/*  [input]             [output]
    5 5
    50 20 10 10 30      2
    1 1 2               2
    2 0 5               0
    2 0 -10             0
    1 0 10              0
    20 2                1
    10 3                1
    40 4                3
    20 4                1
    35 5                1
*/
#define pii pair<int , ll>
class LT {  public:
    int pool[MAXN], pool_cnt;
    int fa[MAXN];
    int find(int x) {
        if (x == fa[x]) return x;
        else return fa[x] = find(fa[x]);
    }
    struct node {
        int id; ll v;
        node() {}
        node(int _id, ll _v) { id = _id, v = _v; }
        bool operator<(const node &tb) { return v < tb.v; }
        bool operator==(const node &tb) { return v == tb.v; }
        bool operator>(const node &tb) { return v > tb.v; }
    } val[MAXN];
    ll add[MAXN], mul[MAXN];

    int ch[MAXN][2], dist[MAXN];
    int tot;

    void init() { tot = 0, pool_cnt = 0; }
    inline int Newnode(ll v, int id) { // 初始化左偏树节点
        int nrt = New();
        val[nrt].v = v, val[nrt].id = id, ch[nrt][0] = ch[nrt][1] = dist[nrt] = 0;
```

```
        mul[nrt] = 1, add[nrt] = 0;
        fa[nrt] = nrt;
        return nrt;
    }
#define lson ch[rt][0]
#define rson ch[rt][1]
    inline void push_down(int rt) {
        if (mul[rt] != 1) {
            if (lson) val[lson].v *= mul[rt], add[lson] *= mul[rt], mul[lson] *= mul[rt];
            if (rson) val[rson].v *= mul[rt], add[rson] *= mul[rt], mul[rson] *= mul[rt];
            mul[rt] = 1;
        }
        if (add[rt]) {
            if (lson) val[lson].v += add[rt], add[lson] += add[rt];
            if (rson) val[rson].v += add[rt], add[rson] += add[rt];
            add[rt] = 0;
        }
    }
    int merge(int x, int y) {   // 合并左偏树
        if (!x || !y) return x + y;
        if (val[x] == val[y] ? x > y : val[x] > val[y]) swap(x, y); // 小根堆
        push_down(x);
        ch[x][1] = merge(ch[x][1], y);
        if (dist[ch[x][0]] < dist[ch[x][1]]) swap(ch[x][0], ch[x][1]);
        fa[ch[x][0]] = fa[ch[x][1]] = fa[x] = x;
        dist[x] = dist[ch[x][1]] + 1;
        return x;
    }

    int insert(int rt, ll v, int id) {  return merge(rt, Newnode(v, id));}
    int pop(int rt) {   // call: root[fx] = tree.pop(root[fx]);
        push_down(rt);
        int tl = ch[rt][0], tr = ch[rt][1];
        Del(rt);
        return merge(tl, tr);
    }
    bool isempty(int x) { return x == 0;}
    pii top(int x) { return mp(val[x].id, val[x].v); }
} tree;

ll h[MAXN];
int f[MAXN], a[MAXN], c[MAXN];
ll v[MAXN], s[MAXN];

int root[MAXN]; int dep[MAXN]; int res1[MAXN], res2[MAXN];
void dfs(int u, int father) {
    dep[u] = dep[father] + 1;
    for (int i = head[u]; ~i; i = e[i].nex) {
        int tv = e[i].to;
        dfs(tv, u);
    }
    while (!tree.isempty(root[u]) && tree.top(root[u]).second < h[u]) {
        res1[u]++;
        int id = tree.top(root[u]).first;
        res2[id] = dep[c[id]] - dep[u];
        root[u] = tree.pop(root[u]);
```

```
    }
    if (tree.isempty(root[u])) return;
    if (a[u]) { // mul
        int ru = root[u];
        tree.val[ru].v *= v[u], tree.add[ru] *= v[u], tree.mul[ru] *= v[u];
    } else {
        int ru = root[u];
        tree.val[ru].v += v[u], tree.add[ru] += v[u];
    }
    if (u == 1) {
        while (!tree.isempty(root[1])) {
            int id = tree.top(root[1]).first;
            res2[id] = dep[c[id]] - dep[1] + 1;
            root[1] = tree.pop(root[1]);
        }
    } else {
        root[f[u]] = tree.merge(root[f[u]], root[u]);
    }
}

int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) head[i] = -1;
    for (int i = 1; i <= n; i++) scanf("%lld", &h[i]);
    for (int i = 2; i <= n; i++) {
        scanf("%d%d%lld", &f[i], &a[i], &v[i]);
        addEdge(f[i], i);
    }
    for (int i = 1; i <= m; i++) {
        scanf("%lld%d", &s[i], &c[i]);
        root[c[i]] = tree.insert(root[c[i]], s[i], i);  // build a lot of heap
    }
    dfs(1, 1);
    for (int i = 1; i <= n; i++) printf("%d\n", res1[i]);
    for (int i = 1; i <= m; i++) printf("%d\n", res2[i]);
}
```

6 分块

6.1 区间加法求和

```
/*
    input       output
    4
    1 2 2 3
    0 1 3 1     1
    1 1 4 4
    0 1 2 2     4
    1 1 2 4
*/
ll a[MAXN], b[sqrt(MAXN)], sum[sqrt(MAXN)];
int bl[MAXN], unit;
void add(int l, int r, ll c) {
```

```
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        for (int i = l; i <= r; i++) a[i] += c, sum[sid] += c;
        return;
    }
    for (int i = l; bl[i] == sid; i++) a[i] += c, sum[sid] += c;
    for (int i = sid + 1; i < eid; i++) b[i] += c, sum[i] += unit * c;
    for (int i = r; bl[i] == eid; i--) a[i] += c, sum[eid] += c;
}
ll query(int l, int r, ll mod) {
    int sid = bl[l], eid = bl[r];
    ll ans = 0;
    if (sid == eid) {
        for (int i = l; i <= r; i++) ans = ((ans + a[i]) % mod + b[sid]) % mod;
        return ans;
    }
    for (int i = l; bl[i] == sid; i++) ans = ((ans + a[i]) % mod + b[sid]) % mod;
    for (int i = sid + 1; i < eid; i++) ans = (ans + sum[i]) % mod;
    for (int i = r; bl[i] == eid; i--) ans = ((ans + a[i]) % mod + b[eid]) % mod;
    return ans;
}
int main() {
    int n; scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        bl[i] = (i - 1) / unit + 1;
        sum[bl[i]] += a[i];
    }
    for (int i = 1; i <= n; i++) {
        int opt, l, r; ll c; scanf("%d%d%d%lld", &opt, &l, &r, &c);
        if (opt == 0) add(l, r, c); // [l, r] 的数+c
        else printf("%lld\n", query(l, r, c + 1));  // sum[l, r] % (c + 1)
    }
}
```

6.2 询问区间内小于某个值的元素个数

```
ll a[MAXN], b[sqrt(MAXN)];
int bl[sqrt(MAXN)], unit;
vector<ll> vec[sqrt(MAXN)];
int n;
void reset(int x) {
    vec[x].clear();
    for (int i = (x - 1) * unit + 1; i <= min(x * unit, n); i++)
        vec[x].push_back(a[i]);
    sort(vec[x].begin(), vec[x].end());
}
void add(int l, int r, ll c) {
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        for (int i = l; i <= r; i++) a[i] += c;
        reset(sid);
        return;
```

```
    }
    for (int i = l; bl[i] == sid; i++) a[i] += c;
    for (int i = sid + 1; i < eid; i++) b[i] += c;
    for (int i = r; bl[i] == eid; i--) a[i] += c;
    reset(sid), reset(eid);
}
int query(int l, int r, ll c) {
    int ans = 0;
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        for (int i = l; i <= r; i++) {
            if (a[i] + b[sid] < c) ans++;
        }
        return ans;
    }
    for (int i = l; bl[i] == sid; i++) {
        if (a[i] + b[sid] < c) ans++;
    }

    for (int i = sid + 1; i < eid; i++) {
        ans += lower_bound(vec[i].begin(), vec[i].end(), c - b[i]) - vec[i].begin();
    }
    for (int i = r; bl[i] == eid; i--) {
        if (a[i] + b[eid] < c) ans++;
    }
    return ans;
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        bl[i] = (i - 1) / unit + 1;
        vec[bl[i]].push_back(a[i]);
    }
    for (int i = 1; i <= bl[n]; i++) sort(vec[i].begin(), vec[i].end());
    for (int i = 1; i <= n; i++) {
        int opt, l, r; ll c; scanf("%d%d%d%lld", &opt, &l, &r, &c);
        if (opt) printf("%d\n", query(l, r, c * c));
        else add(l, r, c);
    }

}
```

6.3 询问区间内某个值的前驱

```
ll a[MAXN], b[sqrt(MAXN)];
int bl[sqrt(MAXN)], unit;
set<ll> st[sqrt(MAXN)];
void add(int l, int r, ll c) {
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        for (int i = l; i <= r; i++) {
```

```
            st[sid].erase(a[i]); a[i] += c; st[sid].insert(a[i]);
        }
        return;
    }
    for (int i = l; bl[i] == sid; i++) {
        st[sid].erase(a[i]), a[i] += c, st[sid].insert(a[i]);
    }
    for (int i = sid + 1; i < eid; i++) b[i] += c;
    for (int i = r; bl[i] == eid; i--) {
        st[eid].erase(a[i]), a[i] += c, st[eid].insert(a[i]);
    }
}

ll query(int l, int r, ll c) {
    int sid = bl[l], eid = bl[r];
    ll ans = -1;
    if (sid == eid) {
        for (int i = l; i <= r; i++) {
            if (a[i] + b[sid] < c) ans = max(ans, a[i] + b[sid]);
        }
        return ans;
    }
    for (int i = l; bl[i] == sid; i++) {
        if (a[i] + b[sid] < c) ans = max(ans, a[i] + b[sid]);
    }
    for (int i = sid + 1; i < eid; i++) {
        set<ll>::iterator it = st[i].lower_bound(c - b[i]);
        if (it == st[i].begin()) continue;
        it--;
        ans = max(ans, (*it) + b[i]);
    }
    for (int i = r; bl[i] == eid; i--) {
        if (a[i] + b[eid] < c) ans = max(ans, a[i] + b[eid]);
    }
    return ans;
}

int main() {
    int n; scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        bl[i] = (i - 1) / unit + 1;
        st[bl[i]].insert(a[i]);
    }
    for (int i = 1; i <= n; i++) {
        int opt, l, r; ll c; scanf("%d%d%d%lld", &opt, &l, &r, &c);
        if (opt) printf("%lld\n", query(l, r, c));
        else add(l, r, c);
    }

}
```

6.4 区间寻找某个数并赋值

```
ll a[MAXN];
int bl[MAXN], unit;
int flag[sqrt(MAXN)]; ll tag[sqrt(MAXN)];
int n;
void reset(int pos) {
    if (!flag[pos]) return;
    for (int i = (pos - 1) * unit + 1; i <= min(pos * unit, n); i++) a[i] = tag[pos];
    flag[pos] = 0;
}
int solve(int l, int r, ll c) {
    int ans = 0;
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        reset(sid);
        for (int i = l; i <= r; i++) {
            if (a[i] == c) ans++;
            else a[i] = c;
        }
        return ans;
    }
    reset(sid);
    for (int i = l; bl[i] == sid; i++) {
        if (a[i] == c) ans++;
        else a[i] = c;
    }
    for (int i = sid + 1; i < eid; i++) {
        if (flag[i]) {
            if (tag[i] == c) ans += unit;
            else tag[i] = c;
        } else {
            for (int j = (i - 1) * unit + 1; j <= min(n, i * unit); j++) {
                if (a[j] == c) ans++;
                else a[j] = c;
            }
            tag[i] = c, flag[i] = 1;
        }
    }
    reset(eid);
    for (int i = r; bl[i] == eid; i--) {
        if (a[i] == c) ans++;
        else a[i] = c;
    }
    return ans;
}
int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        bl[i] = (i - 1) / unit + 1;
    }
    for (int i = 1; i <= n; i++) {
        int l, r; ll c; scanf("%d%d%lld", &l, &r, &c);
        printf("%d\n", solve(l, r, c));
    }
}
```

6.5 区间最小众数

```cpp
using Discrete::id2val;
using Discrete::val2id;
int a[MAXN];
int bl[MAXN], unit;
int f[sqrt(n / (log(n) / log(2)))][sqrt(n / (log(n) / log(2)))];
vector<int> vec[MAXN];
int cnt[MAXN];
int n;
void pre(int pos) {
    memset(cnt, 0, sizeof(cnt));
    int ans = 0;
    for (int i = (pos - 1) * unit + 1; i <= n; i++) {
        cnt[a[i]]++;
        if (cnt[a[i]] > cnt[ans] || (cnt[a[i]] == cnt[ans] && a[i] < ans)) ans = a[i];
        f[pos][bl[i]] = ans;
    }
}
int _query(int l, int r, int x) {
    return upper_bound(vec[x].begin(), vec[x].end(), r) - lower_bound(vec[x].begin(),
        vec[x].end(), l);
}
int query(int l, int r) {
    int ans = f[bl[l] + 1][bl[r] - 1], mx = _query(l, r, ans);
    int sid = bl[l], eid = bl[r];
    if (sid == eid) {
        for (int i = l; i <= r; i++) {
            int t = _query(l, r, a[i]);
            if (t > mx || (t == mx && a[i] < ans)) ans = a[i], mx = t;
        }
        return ans;
    }
    for (int i = l; bl[i] == sid; i++) {
        int t = _query(l, r, a[i]);
        if (t > mx || (t == mx && a[i] < ans)) ans = a[i], mx = t;
    }
    for (int i = r; bl[i] == eid; i--) {
        int t = _query(l, r, a[i]);
        if (t > mx || (t == mx && a[i] < ans)) ans = a[i], mx = t;
    }
    return ans;
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n / (log(n) / log(2)));
    Discrete::btol = 0;
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        Discrete::insert(a[i]);
    }
    Discrete::init();
    for (int i = 1; i <= n; i++) {
        a[i] = val2id(a[i]);
```

```
        vec[a[i]].push_back(i);
    }
    for (int i = 1; i <= n; i++) {
        bl[i] = (i - 1) / unit + 1;
    }
    for (int i = 1; i <= bl[n]; i++) pre(i);
    for (int i = 1; i <= n; i++) {
        int l, r; scanf("%d%d", &l, &r);
        printf("%d\n", id2val(query(l, r)));
    }
}
```

6.6 莫队

6.6.1 区间查询，统计两个相同概率

```
/*
    HYSBZ - 2038 小Z的袜子
    统计[L,R]区间内选两只袜子，颜色相同的概率。区间查询，统计两个相同概率。
    N,M ≤ 50000，1 ≤ L < R ≤ N，Ci ≤ N。
*/
#define ll long long
const ll SIZE = 50050;
struct node {//查询
    ll L, R;
    ll id, res1, res2;
} q[SIZE];

ll c[SIZE];//n只袜子的颜色
ll n, m, unit;//n只瓦子，m次查询
ll num[SIZE];

ll com(ll nn, ll mm) {
    if (mm > nn / 2)mm = nn - mm;
    ll aa = 1, bb = 1;
    for (ll i = 1; i <= mm; i++) {
        aa *= nn + 1 - i;
        bb *= i;
        if (aa % bb == 0) {
            aa /= bb;
            bb = 1;
        }
    }
    ll tmp = aa / bb;
    return tmp;
}

//朴素排序
bool cmp(node a, node b) {
    if (a.L / unit != b.L / unit)return a.L / unit < b.L / unit;
    else return a.R < b.R;
}

//对答案进行排序
```

```
bool cmp_id(node a, node b) {
    return a.id < b.id;
}

int main() {
    cin >> n >> m;
    unit = sqrt(n);//分块
    for (ll i = 1; i <= n; i++)cin >> c[i];
    for (ll i = 1; i <= m; i++) {
        q[i].id = i;
        cin >> q[i].L >> q[i].R;
    }
    sort(q + 1, q + 1 + m, cmp);
    ll L = 1, R = 0;
    ll sum = 0;
    for (ll i = 1; i <= m; i++) {
        while (R < q[i].R) {
            R++;
            if (num[c[R]] > 1)sum -= com(num[c[R]], 2);
            num[c[R]]++;
            if (num[c[R]] > 1)sum += com(num[c[R]], 2);
        }
        while (R > q[i].R) {
            if (num[c[R]] > 1)sum -= com(num[c[R]], 2);
            num[c[R]]--;
            if (num[c[R]] > 1)sum += com(num[c[R]], 2);
            R--;
        }
        while (L < q[i].L) {
            if (num[c[L]] > 1)sum -= com(num[c[L]], 2);
            num[c[L]]--;
            if (num[c[L]] > 1)sum += com(num[c[L]], 2);
            L++;
        }
        while (L > q[i].L) {
            L--;
            if (num[c[L]] > 1)sum -= com(num[c[L]], 2);
            num[c[L]]++;
            if (num[c[L]] > 1)sum += com(num[c[L]], 2);
        }
        ll under = com(q[i].R - q[i].L + 1, 2);
        ll gcdd = __gcd(sum, under);
        q[i].res1 = sum / gcdd;
        q[i].res2 = under / gcdd;
    }
}
```

6.6.2 时间戳 + 统计有多少个不同的数

```
struct node {
    int l, r, time, id;
} q[SIZE];
struct Update {
    int x, y;
```

```
} upd[SIZE];
int l, r;
const int COLSIZE = 1000500;
int col[COLSIZE];
int unit;
int bl[SIZE];
int vis[COLSIZE];
int ans = 0;
int res[SIZE];

void pop(int x) {
    ans -= !--vis[x];
}

void push(int x) {
    ans += !vis[x]++;
}

bool cmp(node a, node b) {// 奇偶排序
    return (bl[a.l] ^ bl[b.l]) ? bl[a.l] < bl[b.l] : ((bl[a.r] ^ bl[b.r]) ? bl[a.r] < bl[b.r]
        : a.time < b.time);
}

void modify(int x) {
    if (l <= upd[x].x && upd[x].x <= r) {
        pop(col[upd[x].x]);
        push(upd[x].y);
    }
    swap(upd[x].y, col[upd[x].x]);
}

int main() {
    int n, m; scanf("%d%d", &n, &m);
    int tmpb = pow(double(n), double(2.0 / 3.0));
    for (int i = 1; i <= n; i++) {
        scanf("%d", &col[i]);
        bl[i] = i / tmpb;
    }
    int time = 0;
    char ch;
    int x, y;
    int mm = 1, updm = 1;
    for (int i = 1; i <= m; i++) {
        cin >> ch >> x >> y;
        if (ch == 'Q') {// Q查询
            q[mm].l = x;
            q[mm].r = y;
            q[mm].time = time;
            q[mm].id = mm;
            mm++;
        } else if (ch == 'R') {// R更换画笔
            upd[updm].x = x, upd[updm].y = y;
            time++;
            updm++;
        }
    }
```

```
    sort(q + 1, q + mm, cmp);
    l = 1, r = 0;
    int tp = 0;
    for (int i = 1; i < mm; i++) {
        while (l < q[i].l)pop(col[l++]);
        while (l > q[i].l)push(col[--l]);
        while (r < q[i].r)push(col[++r]);
        while (r > q[i].r)pop(col[r--]);
        while (tp < q[i].time)modify(++tp);
        while (tp > q[i].time)modify(tp--);
        res[q[i].id] = ans;
    }
}
```

6.6.3 树状数组维护区间两数之差

```
/*
    HDU – 6534 树状数组维护区间两数之差
    i<j，|ai–aj|≤ K，n  (1≤n≤27000), m (1≤m≤27000) and K (1≤K≤109)
*/
//树状数组部分————————————
int lowbit(int k) { return k & –k; }

void add(int x, int k) {
    while (x <= n) {
        tree[x] += k;
        x += lowbit(x);
    }
}

int Get_sum(int x) {
    int ans = 0;
    while (x != 0) {
        ans += tree[x];
        x –= lowbit(x);
    }
    return ans;
}
//————————————————————————
/*
// 统计 a[i]^a[i+1]^...a[j]=k,非树状数组
// 1 ≤ n , m ≤ 100 000 ,  0 ≤ k ≤ 1 000 000 ,(0 ≤ ai ≤ 1 000 000)
ll tmp=0;
void add(ll x){
    tmp += cnt[sum[x]^k];
    cnt[sum[x]]++;
}
void del(ll x){
    cnt[sum[x]]--;
    tmp –= cnt[sum[x]^k];
}
*/
int sum = 0;

```

```
void push(int x) {
    sum += Get_sum(rr[x]) - Get_sum(ll[x] - 1);
    add(a[x], 1);
}

void pop(int x) {
    add(a[x], -1);
    sum -= Get_sum(rr[x]) - Get_sum(ll[x] - 1);
}

int main() {
    sort(q + 1, q + 1 + m, cmp);
    int L = 1, R = 0;
    for (int i = 1; i <= m; i++) {
        while (L > q[i].l)push(--L);
        while (R < q[i].r)push(++R);
        while (L < q[i].l)pop(L++);
        while (R > q[i].r)pop(R--);
        display[q[i].id] = sum;//用于输出
    }
}
```

6.6.4 统计有多少个不同的数

```
/*
    HYSBZ - 1878 统计[L,R]区间内有多少个不同的数
*/
#define ll long long
const ll SIZE = 1005000;
struct node {
    ll L, R;
    ll id, res;
} q[SIZE];
ll c[SIZE], num[SIZE], dis[SIZE], pos[SIZE];
ll n, m;
ll sum = 0;

bool cmp(const node &a, const node &b) {//奇偶排序
    return (pos[a.L] ^ pos[b.L]) ? pos[a.L] < pos[b.L] : ((pos[a.L] & 1) ? a.R < b.R : a.R >
        b.R);
}

ll ans = 0;

int main() {
    scanf("%lld", &n);//n个数
    ll tmpb = sqrt(n);
    sort(q + 1, q + 1 + m, cmp);
    ll L = 1, R = 0;
    for (ll i = 1; i <= m; i++) {
        while (L < q[i].L)ans -= !--num[c[L++]];
        while (L > q[i].L)ans += !num[c[--L]]++;
        while (R < q[i].R)ans += !num[c[++R]]++;
        while (R > q[i].R)ans -= !--num[c[R--]];
```

```
        dis[q[i].id] = ans;//用于输出
    }

}
```

6.6.5 回滚莫队

```
/*
    给定一个长为n的序列{a1,a2,a3..}, 询问区间a1*cnt(a1) + a2*cnt(a2) + ... 的最大值,
        即某个值乘上出现次数
*/
#define inf 0x3f3f3f3f3f3f
#define N 100005
#define M 4000005
typedef long long ll;
using namespace std;
struct point {
    ll l, r, id;
    ll res;
} q[N];
ll vis[M];
ll a[N], pos[N];
ll s[N], as[N];

bool cmp(const point &a, const point &b) {
    return (pos[a.l] ^ pos[b.l]) ? pos[a.l] < pos[b.l] : a.r < b.r;
}

bool cmp2(const point &a, const point &b) {
    return a.id < b.id;
}

ll rpos[N];
ll tempv[N];

int main() {
    ll n, m; scanf("%lld%lld", &n, &m);
    ll b = sqrt(n);
    ll bnum = ceil((double) n / b);
    for (int i = 1; i <= bnum; i++) {
        rpos[i] = b * i;
        for (int j = b * (i - 1) + 1; j <= rpos[i]; j++) {
            pos[j] = i;
        }
    }
    rpos[bnum] = n;
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        s[i] = a[i];
    }
    sort(s + 1, s + 1 + n);
    ll tot = unique(s + 1, s + 1 + n) - s - 1;
    for (int i = 1; i <= n; i++) {
        as[i] = lower_bound(s + 1, s + 1 + tot, a[i]) - s;
```

```
    }
    for (int i = 1; i <= m; i++) {
        scanf("%lld%lld", &q[i].l, &q[i].r);
        q[i].id = i;
    }
    sort(q + 1, q + 1 + m, cmp);
    int i = 1;
    for (ll k = 0; k <= bnum; k++) {
        ll l = rpos[k] + 1, r = rpos[k];
        ll ans = 0;
        memset(vis, 0, sizeof(vis));
        for (; pos[q[i].l] == k; i++) {
            if (pos[q[i].l] == pos[q[i].r]) {
                ll res = 0;
                for (int j = q[i].l; j <= q[i].r; j++) {
                    tempv[as[j]] = 0;
                }
                for (int j = q[i].l; j <= q[i].r; j++) {
                    res = max(res, (++tempv[as[j]]) * a[j]);
                }
                q[i].res = res;
                continue;
            }
            while (r < q[i].r) {
                r++;
                ans = max(ans, (++vis[as[r]]) * a[r]);
            }
            ll temp = ans;
            while (l > q[i].l) {
                l--;
                ans = max(ans, (++vis[as[l]]) * a[l]);
            }
            q[i].res = ans;
            while (l < rpos[k] + 1) {
                --vis[as[l++]];
            }
            ans = temp;
        }
    }
    sort(q + 1, q + 1 + m, cmp2);
    for (int i = 1; i <= m; i++) {
        printf("%lld\n", q[i].res);
    }
}
```

6.6.6 普通莫队代替回滚莫队（AT1219）

IOI 国历史研究的第一人——JOI 教授，最近获得了一份被认为是古代 IOI 国的住民写下的日记。JOI 教授为了通过这份日记来研究古代 IOI 国的生活，开始着手调查日记中记载的事件。

日记中记录了连续 NN 天发生的时间，大约每天发生一件。事件有种类之分。第 ii 天发生的事件的种类用一个整数 X_i 表示，X_i 越大，事件的规模就越大。

JOI 教授决定用如下的方法分析这些日记：

1. 选择日记中连续的几天 $[L, R]$ 作为分析的时间段；定义事件 A 的重要度 W_A 为 A× T_A，其中 T_A 为该事件在区间 $[L, R]$ 中出现的次数。

2. 现在，您需要帮助教授求出所有事件中重要度最大的事件是哪个，并输出其重要度。

```
const int SIZE = 100500;
ll T[SIZE * 3], lazy[SIZE * 3];
ll M;
inline void modify(ll n, ll v) {
    for (T[n += M] += v, n >>= 1; n; n >>= 1)
        T[n] = max(T[n + n], T[n + n + 1]);
}
struct node {
    ll l, r, id;
} q[SIZE];
ll arr[SIZE], bl[SIZE], unit;
bool cmp(node x, node y) {
    if (x.l / unit != y.l / unit) {
        return x.l < y.l;
    }
    return x.r < y.r;
}
ll mp[SIZE], dis[SIZE];

bool cmp_dis(ll x, ll y) { return x < y;}

void Discrete(int N) {
    sort(dis + 1, dis + N + 1, cmp_dis);
    ll dis_len = unique(dis + 1, dis + N + 1) - dis;
    for (M = 1; M <= dis_len; M <<= 1);
    for (ll i = 1; i <= N; i++) {
        ll lb = lower_bound(dis + 1, dis + 1 + dis_len, arr[i]) - dis;
        mp[lb] = arr[i];
        arr[i] = lb;
    }
}

ll res[SIZE];

void push(ll x) { modify(arr[x], mp[arr[x]]); }
void pop(ll x) { modify(arr[x], -mp[arr[x]]); }
int main() {
    int N, Q; scanf("%d%d", &N, &Q);
    unit = sqrt(N);
    for (int i = 1; i <= N; i++) {
        scanf("%lld", &arr[i]);
        dis[i] = arr[i];
    }
    Discrete(N);
    for (int i = 1; i <= Q; i++) {
        q[i].id = i;
        scanf("%lld%lld", &q[i].l, &q[i].r);
    }
    sort(q + 1, q + 1 + Q, cmp);
    ll l = 1, r = 0;
    for (int i = 1; i <= Q; i++) {
        while (l < q[i].l)pop(l++);
        while (l > q[i].l)push(--l);
        while (r < q[i].r)push(++r);
        while (r > q[i].r)pop(r--);
        res[q[i].id] = T[1];
```

```
    }
    for (int i = 1; i <= Q; i++)
        printf("%lld\n", res[i]);
}
```

7 杂项

7.1 数列归纳

$f(n) = \frac{(2*n+1)!}{(n+1)}$
1, 3, 40, 1260, 72576,
6652800, 889574400, 163459296000, 39520825344000, 12164510040883200,
4644631106519040000, 2154334728240414720000, 1193170003333152768000000, 777776389315596582912000000

$f(n) = \sum_{i=1}^{n} p, p \in prime$
转 min_25筛
0, 2, 5, 10, 17, 28, 41, 58, 77, 100,
129, 160, 197, 238, 281, 328, 381, 440, 501, 568,
639, 712, 791, 874, 963, 1060, 1161, 1264, 1371, 1480,
1593, 1720, 1851, 1988, 2127, 2276, 2427, 2584, 2747, 2914,
3087, 3266, 3447, 3638, 3831, 4028, 4227, 4438, 4661, 4888

7.2 CDQ

7.2.1 三位偏序 $O(n \log n \log k)$

n：元素的数量，k：最大属性值。
有 n 个元素，第 i 个元素有 a_i,b_i,c_i 三个属性，设 $f(i)$ 表示满足 $a_j \le a_i$ 且 $b_j \le b_i$ 且 $c_j \le c_i$ 且 $j \ne i$ 的 j 的数量。
对于 $d \in [0, n)$，求 $f(i) = d$ 的数量。

```
class TREE { public:
    int T[MAXN], n;
    inline int lowbit(int x) { return x & (-x); }
    void add(int pos, int val) {
        while (pos <= n) {
            T[pos] += val; pos += lowbit(pos);
        }
    }
    int query(int pos) {
        int ans = 0;
        while (pos) {
            ans += T[pos]; pos -= lowbit(pos);
        }
        return ans;
    }
} tree;

struct node {
    int a, b, c;
    int cnt, ans;
    bool operator==(const node &tb) const {
        if (a == tb.a && b == tb.b && c == tb.c) return 1;
        else return 0;
```

```
    }
} p1[MAXN], p2[MAXN];

void cdq(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    cdq(l, mid), cdq(mid + 1, r);
    sort(p2 + l, p2 + mid + 1, [&](const node &x, const node &y) {
        if (x.b != y.b) return x.b < y.b;
        else return x.c < y.c;
    });
    sort(p2 + mid + 1, p2 + r + 1, [&](const node &x, const node &y) {
        if (x.b != y.b) return x.b < y.b;
        else return x.c < y.c;
    });
    int j = l;
    for (int i = mid + 1; i <= r; i++) {
        while (p2[i].b >= p2[j].b && j <= mid) {
            tree.add(p2[j].c, p2[j].cnt);
            j++;
        }
        p2[i].ans += tree.query(p2[i].c);
    }
    for (int i = l; i < j; i++) tree.add(p2[i].c, -p2[i].cnt);
}

int res[MAXN];
int main() {
    int n, k; scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++) scanf("%d%d%d", &p1[i].a, &p1[i].b, &p1[i].c);
    p1[n+1].a = p1[n+1].b = p1[n+1].c = 0;
    sort(p1 + 1, p1 + 1 + n, [&](const node &x, const node &y) {
        if (x.a != y.a) return x.a < y.a;
        if (x.b != y.b) return x.b < y.b;
        return x.c < y.c;
    });
    int tot = 0, m = 0;
    for (int i = 1; i <= n; i++) {
        tot++;
        if (p1[i].a != p1[i + 1].a || p1[i].b != p1[i + 1].b || p1[i].c != p1[i + 1].c) {
            m++;
            p2[m].a = p1[i].a, p2[m].b = p1[i].b, p2[m].c = p1[i].c, p2[m].cnt = tot;
            tot = 0;
        }
    }
    tree.n = k;
    cdq(1, m);
    for (int i = 1; i <= m; i++) res[p2[i].ans + p2[i].cnt - 1] += p2[i].cnt;
    for (int d = 0; d < n; d++) printf("%d\n", res[d]); // 输出每个f(i) = d的数量
}
```

7.3 LCA

```
const int MAXLOG = 22;
```

```
struct Edge {
    int to, nex;
} e[MAXM << 1];
int head[MAXN], tol;
void addEdge(int u, int v) {
    e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}
namespace LCA {
    int dep[MAXN], fa[MAXN][MAXLOG], lg[MAXN];
    void init(int _n) { // n为点的个数，最坏情况为一条链
        for (int i = 1; i <= _n; i++) {
            lg[i] = lg[i-1] + (1 << lg[i-1] == i);
        }
    }
    void dfs(int u, int f) {
        fa[u][0] = f; dep[u] = dep[f] + 1;
        for (int i = 1; i <= lg[dep[u]]; i++) fa[u][i] = fa[fa[u][i-1]][i-1];
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            if (v == f) continue;
            dfs(v, u);
        }
    }
    int LCA(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        while (dep[u] > dep[v]) u = fa[u][lg[dep[u] - dep[v]] - 1];
        if (u == v) return u;
        for (int k = lg[dep[u]] - 1; k >= 0; k--) {
            if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
        }
        return fa[u][0];
    }
}
```

7.4 全 1 矩阵个数（51nod1291）

input	output
3 3 011 110 100	6 3 0 3 1 0 1 0 0

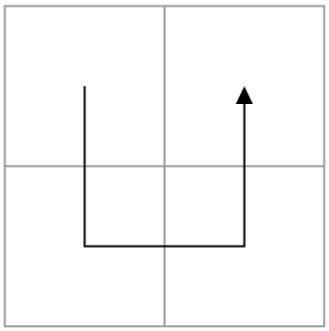
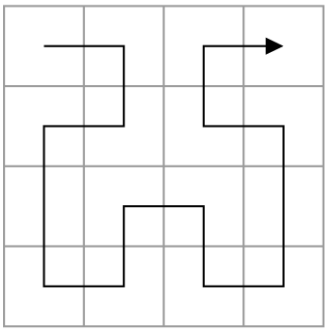
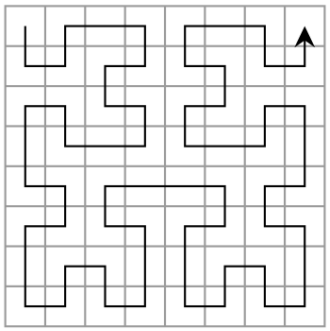
```
char ss[N];
int h[N], sta[N], top, ans[N][N];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int u = 1; u <= n; u++) {
        scanf("%s", ss + 1);
        top = 0;
        for (int i = 1; i <= m + 1; i++) {
            h[i] = ss[i] == '1' ? h[i] + 1 : 0;
            while (top > 0 && h[sta[top]] > h[i]) {
                ans[max(h[sta[top - 1]], h[i]) + 1][i - sta[top - 1] - 1]++;
                ans[h[sta[top]] + 1][i - sta[top - 1] - 1]--;
```

```
                top--;
            }
            while (top > 0 && h[sta[top]] == h[i]) top--;
            sta[++top] = i;
        }
    }
    for (int u = 1; u <= n; u++)
        for (int i = 1; i <= m; i++)
            ans[u][i] = ans[u][i] + ans[u - 1][i];
    for (int u = 1; u <= n; u++) {
        int sum = 0, lalal = 0;
        for (int i = 2; i <= m; i++) {
            lalal = lalal + ans[u][i];
            sum = sum + ans[u][i] * i;
        }
        for (int i = 1; i <= m; i++) {
            ans[u][i] = ans[u][i] + sum;
            sum = sum - lalal - ans[u][i + 1];
            lalal = lalal - ans[u][i + 1];
        }
        // 原题解疑惑代码
        // for (int i=1;i<=m;i++) {
        //     for (int j=i+1;j<=m;j++)
        //         ans[u][i]=ans[u][i]+ans[u][j]*(j-i+1);
        // }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            printf("%d ", ans[i][j]);   // i * j的全1矩阵个数
        }
        printf("\n");
    }
    printf("%lld\n", res);
    return 0;
}
```

7.5 华容道

```
//判断是否有解
int map[16],ans=0;
for(int i=0;i<16;i++){
    scanf("%d",&map[i]);
    if(!map[i])
        ans+=6-i%4-i/4;
    for(int j=0;j<i;j++)
        if(map[j]>map[i])
            ans++;
}
if(ans&1)
    printf("Yes\n");
else
    printf("No\n");
```

7.6 希尔伯特曲线

```
#define ll long long
int two[50];//2的次方
ll f(int n, int x, int y) {//返回第几位
    if (n == 0) return 1;
    int m = two[n-1];//1 << (n - 1);//2的n-1次方
    if (x <= m && y <= m) {
        return f(n - 1, y, x);
    }
    if (x > m && y <= m) {
        return 3LL * m * m + f(n - 1, m-y+ 1, m * 2 - x + 1); // 3LL表示ll 类型的3
    }
    if (x <= m && y > m) {
        return 1LL * m * m + f(n - 1, x, y - m);
    }
    if (x > m && y > m) {
        return 2LL * m * m + f(n - 1, x - m, y - m);
    }
}
const int SIZE=1e6+50;
struct node{                                    //用于存点
    int x,y;
    ll no;
}p[SIZE];
int main() {
    int n;int k;
    scanf("%d%d",&n,&k);
    two[0]=1;                                   //tow[1]=2;
    for(int i=1;i<=32;i++){
        two[i]=2*two[i-1];
    }
    for(int i=1;i<=n;i++){
        scanf("%d%d",&p[i].y,&p[i].x);          //注意y,x的读入顺序!
        p[i].no=f(k,p[i].x,p[i].y);             //用于存点的编号
    }
}
```

7.7 非整数希尔伯特曲线

```
const int SIZE=2e5+5;
struct node{
```

```
    string id;string name;
    bool operator < (const node &b) const {
        return id<b.id;
    }
}a[SIZE];
string f(double x,double y,double s,int dp){
    double mid = s/2;
    string ans;
    if(dp >= 1){
        if(x <= mid && y<= mid)ans = "1" + f(y,x,mid,dp-1);
        else if(x <= mid && y>mid)ans = "2" + f(x,y-mid,mid,dp-1);
        else if(x >mid && y>mid)ans="3" + f(x-mid,y-mid,mid,dp-1);
        else ans = "4" + f(mid-y,s-x,mid,dp-1);
    }
    return ans;
}
int main(){
    int n,k; //n个数，边长为k
    while(~scanf("%d %d",&n,&k)){
        int x,y;
        for(int i=1;i<=n;i++){
            string strtmp;
            cin>>x>>y>>strtmp;
            a[i].name=strtmp;
            a[i].id=f(x,y,k,30);
        }
    }
}
```

7.8 约瑟夫环

7.8.1 一般方法

```
/* *  n个  人(编号 1...n),先去掉第m个数,然后从m+1个开始报1, *
  报到k的退出,剩下的  人继续从1开始报数.求胜利者的编号. */
int main(int argc, const char *argv[]) {
    int n, k, m;
    while (cin >> n >> k >> m, n || k || m) {
        int i, d, s = 0;
        for (i = 2; i <= n; i++) {
            s = (s + k) % i;
        }
        k = k % n;
        if (k == 0) {
            k = n;
        }
        d = (s + 1) + (m - k);
        if (d >= 1 && d <= n) {
            cout << d << '\n';
        }
        else if (d < 1) {
            cout << n + d << '\n';
        }
        else if (d > n) {
```

```
            cout << d % n << '\n';
        }
    }
}
```

7.8.2 函数图像解

```
/* * n 个  人数到 k 出列列，后剩下的人编号 */
unsigned long long n, k;
int main() {
    cin >> n >> k;
    long long y = k % 2;
    long long x = 2, t = 0;
    long long z1 = y, z2 = x;
    while (x <= n) {
        z1 = y;
        z2 = x;
        t = (x - y) / (k - 1);
        if (t == 0) {
            t++;
        }
        y = y + t * k - ((y + t * k) / (x + t)) * (x + t);
        x += t;
    }
    cout << (z1 + (n - z2) * k) % n + 1 << endl;
    return 0;
}
```

8 计算几何

8.1 长方体在三维空间中运动离目标点最近距离（2017ICPC 青岛 H）

```
struct Point3 be, en; // kuangbin
double ang(Point3 v1, Point3 v2) {
    return acos((v1 * v2) / (v1.len() * v2.len()));
}
struct Line3; // kuangbin

//点p绕向量ov旋转ang角度，旋转方向是向量ov叉乘向量op
Point3 rotate3(Point3 p, Point3 v, double angle) {
    double ret[3][3], a[3];
    v = v / v.len();
    ret[0][0] = (1.0 - cos(angle)) * v.x * v.x + cos(angle);
    ret[0][1] = (1.0 - cos(angle)) * v.x * v.y - sin(angle) * v.z;
    ret[0][2] = (1.0 - cos(angle)) * v.x * v.z + sin(angle) * v.y;
    ret[1][0] = (1.0 - cos(angle)) * v.y * v.x + sin(angle) * v.z;
    ret[1][1] = (1.0 - cos(angle)) * v.y * v.y + cos(angle);
    ret[1][2] = (1.0 - cos(angle)) * v.y * v.z - sin(angle) * v.x;
    ret[2][0] = (1.0 - cos(angle)) * v.z * v.x - sin(angle) * v.y;
    ret[2][1] = (1.0 - cos(angle)) * v.z * v.y + sin(angle) * v.x;
    ret[2][2] = (1.0 - cos(angle)) * v.z * v.z + cos(angle);
    for (int i = 0; i < 3; i++) a[i] = ret[i][0] * p.x + ret[i][1] * p.y + ret[i][2] * p.z;
```

```
    return Point3(a[0], a[1], a[2]);
}

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        scanf("%lf%lf%lf", &be.x, &be.y, &be.z);    // 起始点
        scanf("%lf%lf%lf", &en.x, &en.y, &en.z);    // 目标点
        Point3 face = Point3(1, 0, 0), head = Point3(0, 0, 1);
        int m; char opt[3]; scanf("%d", &m);
        double res = 1.0 * inf;
        while (m--) {
            double d, t; scanf("%lf%s%lf", &d, opt, &t);
            double dx = d * cos(ang(Point3(1, 0, 0), face)), dy = d * cos(ang(Point3(0, 1, 0),
                face)), dz =
                    d * cos(ang(Point3(0, 0, 1), face));
            Point3 nbe = be + Point3(dx, dy, dz);
            Line3 lane = Line3(be, nbe);

            res = min(res, lane.dispointtoseg(en));
            if (opt[0] == 'U') { // 抬头
                Point3 v = face ^head;
                face = rotate3(face, v, t);
                head = rotate3(head, v, t);
            } else if (opt[0] == 'D') { // 低头
                Point3 v = head ^face;
                face = rotate3(face, v, t);
                head = rotate3(head, v, t);
            } else if (opt[0] == 'L') { // 左转
                face = rotate3(face, head, t);
            } else if (opt[0] == 'R') { // 后转
                Point3 v = head * (-1);
                face = rotate3(face, v, t);
            }
            be = nbe;
        }
        printf("%.2f\n", res);
    }
}
```

8.2 最小球覆盖（模拟退火）

```
const double eps = 1e-8;
const double start_T = 10000; // 初始温度记得设足够高
struct point3d {
    double x, y, z;
} data[150];
int n;
double dis(point3d a, point3d b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) * (a.z -
        b.z));
}
double solve() {
    double step = start_T, ans = 1e30, mt;
```

```
    point3d z;
    z.x = z.y = z.z = 0;
    int s = 0;
    while (step > eps) {
        for (int i = 0; i < n; i++)
            if (dis(z, data[s]) < dis(z, data[i])) s = i;
        mt = dis(z, data[s]);
        ans = min(ans, mt);
        z.x += (data[s].x - z.x) / mt * step;
        z.y += (data[s].y - z.y) / mt * step;
        z.z += (data[s].z - z.z) / mt * step;
        step *= 0.98;
    }
    return ans;
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lf%lf%lf", &data[i].x, &data[i].y, &data[i].z);
    }
    double ans = solve();
    printf("%.15f\n", ans);
}
```

8.3 求 n 个点的带权类费马点（模拟退火）

```
const int MAXN = 10005;
const double eps = 1e-8;
struct node {
    double x, y, weight;
} nd[MAXN];
int n;
node solve() {
    double step = 1000;
    node ans;
    while (step > eps) {
        double x = 0, y = 0;
        for (int i = 1; i <= n; i++) {
            double tmp = sqrt((ans.x - nd[i].x) * (ans.x - nd[i].x) + (ans.y - nd[i].y) *
                (ans.y - nd[i].y));
            if (fabs(tmp) < eps) continue;
            x += nd[i].weight / tmp * (nd[i].x - ans.x);
            y += nd[i].weight / tmp * (nd[i].y - ans.y);
        }
        double tmp = sqrt(x * x + y * y);
        if (fabs(tmp) >= eps) {
            ans.x += step / tmp * x;
            ans.y += step / tmp * y;
        }
        step *= 0.98;
    }
    return ans;
}
int main() {
```

```
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lf%lf%lf", &nd[i].x, &nd[i].y, &nd[i].weight);
    }
    node res = solve();
    printf("%.3lf %.3lf", res.x, res.y);

}
```

9 动态规划

9.1 #2 字符串 T 在字符串 S 子序列出现的次数

```
/*
  input:S = "babgbag", T = "bag"
  output:5
*/
ll dp[10][MAXN];
ll fun(const string &S, int slen, const string &T, int tlen) {
    for (int i = 0; i <= tlen + 1; i++) dp[i][0] = 0;
    for (int i = 0; i <= slen + 1; i++) dp[0][i] = 1;
    for (int i = 1; i <= tlen + 1; i++) {
        for (int j = 1; j <= slen + 1; j++) {
            if (T[i - 1] == S[j - 1]) dp[i][j] = (dp[i - 1][j - 1] + dp[i][j - 1]) % mod;
            else dp[i][j] = dp[i][j - 1];
        }
    }
    return dp[tlen][slen];
}
```

9.2 #3 N 种长度为 1 元素填充 L

```
/*
  input N = 3, L = 3, K = 1
  output 6 [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1].
  input N = 2, L = 3, K = 0
  output 6 [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], [1, 2, 2]
  input N = 2, L = 3, K = 1
  output [1, 2, 1], [2, 1, 2]
*/
vector<vector<long long>> dp;
// 空间复杂度O（NL）
int numMusicPlaylists(int N, int L, int K) {
    dp.resize(L + 1);
    for (int i = 0; i <= L; i++) dp[i].resize(N + 1);
    dp[0][0] = 1;
    for (int i = 1; i <= L; i++) {
        for (int j = 1; j <= N; j++) {
            dp[i][j] = (dp[i][j] + dp[i - 1][j - 1] * (long long) (N - j + 1)) % mod;
            dp[i][j] = (dp[i][j] + dp[i-1][j] * max(j - K, 0)) % mod;
        }
    }
```

```
    return dp[L][N];
}
// 空间复杂度O（L）
int numMusicPlaylists(int N, int L, int K) {
    dp.resize(2);
    for (int i = 0; i < 2; i++) dp[i].resize(N + 1);
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j <= N; j++) dp[i][j] = 0;
    }
    dp[0][0] = 1;
    int flag = 0;
    for (int i = 1; i <= L; i++) {
        dp[!flag][0] = 0;
        for (int j = 1; j <= N; j++) {
            dp[!flag][j] = 0;
            dp[!flag][j] = (dp[!flag][j] + dp[flag][j - 1] * (long long) (N - j + 1)) % mod;
            dp[!flag][j] = (dp[!flag][j] + dp[flag][j] * max(j - K, 0)) % mod;
        }
        flag = !flag;
    }
    return dp[flag][N];
}
```

9.3 #4 分割数组

```
/*
  nums = [7,2,5,10,8], m = 2
  res = 18
*/
vector<int> dp, sum;
int splitArray(vector<int> &nums, int m) {
    int n = (int) nums.size();
    dp.resize(n), sum.resize(n);
    sum[0] = nums[0];
    for (int i = 1; i < n; i++) sum[i] = sum[i - 1] + nums[i];
    for (int i = 0; i < n; i++) dp[i] = sum[i];
    for (int i = 2; i <= m; i++) {
        int pos = n - 1;
        for (int j = n - 1; j >= 1; j--) {
            while (pos >= 0 && dp[pos] >= sum[j] - sum[pos]) {
                pos--;
            }
            if (pos >= 0) dp[j] = min(dp[j], max(dp[pos], sum[j] - sum[pos]));
            if (pos < n - 1) dp[j] = min(dp[j], max(dp[pos + 1], sum[j] - sum[pos + 1]));
        }
    }
    return dp[n - 1];
}
```

9.4 #5 划分为 K 个相等的子集

```
/*
  nums = [4, 3, 2, 3, 5, 2, 1], k = 4
```

```
  return true
*/
bool canPartitionKSubsets(vector<int> &nums, int k) { // 调用函数
    int sum = 0;
    for (int i = 0; i < nums.size(); ++i) {
        sum += nums[i];
    }
    if (sum % k != 0) return false; //不能被整除,返回
    int target = sum / k;
    sort(nums.begin(), nums.end(),
        greater<int>());//从大到小排序,在计算dfs中的sum时,从最大的元素开始累加可以减少递归的次数
    if (target < nums[0]) return false;//不加这一句会超时
    vector<int> mark(nums.size(), 0);//标记数组,标记该元素已经被使用过,减少重复次数
    return dfs(nums, mark, 0, k, target);
}

bool dfs(vector<int> &nums, vector<int> &mark, int sum, int k, int target) {
    if (k == 0) return true;//找到了k个子集
    if (sum == target) return dfs(nums, mark, 0, k - 1,
        target);//找到了一个和为target的子集,sum置0,k减去一
    for (int i = 0; i < nums.size(); ++i) {//依次循环计算sum
        if (mark[i]) continue;//如果该元素已经被其他子集占有,则直接跳过
        if (sum > target) return false;//sum比target大,则直接返回false
        //将该元素标记为使用过
        mark[i] = 1;
        if (dfs(nums, mark, sum + nums[i], k, target)) return
            true;//一直递归知道找到k个子集时,返会true
        mark[i] = 0;//回溯后,该元素不符合要求,将该元素的使用标志置为0
    }
    return false;//遍历完整个数组都不能找到符合要求的子集
}
```

10 Java & Python

eclipse 下 ALT+/，自动补全代码。

10.1 Java

```
import java.io.*;
import java.util.*;
import java.math.*;
public class Main {//大数加法
    static Scanner cin = new Scanner(System.in);
    static PrintWriter cout = new PrintWriter(System.out);
    public static void main(String[] args) throws IOException {
        BigInteger a=cin.nextBigInteger(),b...;
        cout.println(a.multiply(b));
        cout.flush();
    }
}
public class Main {//排序
    static BufferedReader reader = new BufferedReader(new InputStreamReader(System.in), 1 <<
        16);
```

```
    static BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(System.out), 1 << 16);
    public static void main(String[] args) throws IOException {
        int n = Integer.parseInt(reader.readLine());
        int[] array = new int[n];
        for(int i = 0; i < n; i++) array[i] = Integer.parseInt(reader.readLine());
        Arrays.sort(array);
        for(int i = 0; i < n; i++)
            writer.write(array[i] + "\r\n");
        writer.flush();
    }
}
public class Main {//大数开方
    static BufferedReader reader = new BufferedReader(new InputStreamReader(System.in), 1 << 16);
    static BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(System.out), 1 << 16);
    public static void main(String[] args) throws Exception {
        writer.write(BigIntSqrt(reader.readLine()) + "\r\n");
        writer.flush();
    }
    public static String BigIntSqrt(String nStr) {
        BigDecimal n = new BigDecimal(nStr);
        BigDecimal ans = new BigDecimal(nStr.substring(0, nStr.length()/2+1));
        BigDecimal tmp = BigDecimal.ONE;
        BigDecimal two = new BigDecimal("2");
        int length = 2;
        while (true) {
            tmp = ans.add(n.divide(ans, length, RoundingMode.HALF_DOWN));
            tmp = tmp.divide(two, length, RoundingMode.HALF_DOWN);
            if (tmp.subtract(ans).abs().compareTo(BigDecimal.ONE) == -1) break;
            ans = tmp;
        }
        String str = ans.toString();
        return str.substring(0, str.length() - length - 1);
    }
}
```

```
public class Main {
    public static BigInteger value0=BigInteger.valueOf(0);
    public static BigInteger value1=BigInteger.valueOf(1);
    public static int MAXN=100005;
    public static void main(String args[]) {
        Scanner cin=new Scanner(System.in);
        int n,m;
        n=cin.nextInt();
        m=cin.nextInt();
        int a[]=new int[MAXN];
        int r[]=new int[MAXN];
        for(int i=1;i<=n;i++) {
            a[i]=cin.nextInt();
            r[i]=cin.nextInt();
        }
        BigInteger ans=excrt(a, r, n);
        System.out.println(ans);
    }
```

```
    public static BigInteger[] exgcd(BigInteger a,BigInteger b){
        BigInteger ans;
        BigInteger[] result=new BigInteger[3];
        if(b.equals(value0)){
            result[0]=a;
            result[1]=value1;
            result[2]=value0;
            return result;
        }
        BigInteger [] temp=exgcd(b,a.mod(b));
        ans = temp[0];
        result[0]=ans;
        result[1]=temp[2];
        result[2]=temp[1].subtract(a.divide(b).multiply(temp[2]));
        return result;
    }
    public static BigInteger excrt(int a[],int r[],int n){
        BigInteger M=BigInteger.valueOf(a[1]),R=BigInteger.valueOf(r[1]);
        BigInteger tmp[]=new BigInteger[3];
        for(int i=2;i<=n;i++){
            tmp=exgcd(M,BigInteger.valueOf(a[i]));
            if(!R.subtract(BigInteger.valueOf(r[i])).mod(tmp[0]).equals(value0))return
                BigInteger.valueOf(-1);
            tmp[1]=(R.subtract(BigInteger.valueOf(r[i]))).divide(tmp[0]).multiply(tmp[1]).mod(BigInteger.
            R=R.subtract(M.multiply(tmp[1]));
            M=M.divide(tmp[0]).multiply(BigInteger.valueOf(a[i]));
            R=R.mod(M);
        }
        R=R.mod(M);
        R=R.add(M);
        R=R.mod(M);
        return R;
    }
}
```

10.2 Python

10.2.1 python

```
#print怎么输出后不换行？
print(待输出,end = '')
#python是允许这样赋值的
a,b,c = 1,2,3
print(a,b,c)
#python玩acm读取输入应该这么干
a,b,c = input().strip().split()#其实strip()可有可无
print(a,b,c)
#strip('可选字符，默认为空格')的用处：去掉字符串首位连续的某字符
#split('可选指定分隔符',可选分割次数)的用法：通过分隔符将字符串切片处理
#注意，python的格式控制是这样的。
print(a+b+c,a*b*c,"{:.2f}".format((a+b+c)/3))
#也是这样的
print("{}\n{}\n{:.6f}".format(100,'A',3.14))
print("{:02d}:{:02d}:{:02d}".format(timeA,timeB,timeC))
```

```
#d代表输出int，2代表输出宽度，0代表剩余位用0来填充。
01:08:31
#如果你想在字符串中表示\  请用"\\"  转义

# input: 2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)
# output: 1315
a = input()
a = a.replace("2(", "pow(2,")
a = int(eval(a))
print(a)
```

10.2.2 计算表达式

```
#多组输入，读取到文件末尾EOF结束
while True:
    try:
        line = input()
    except EOFError:
        break
    z = eval(line) #将line转化为表达式类型并运算
    print(z)
```

10.2.3 正则表达式

```
>>> import re
>>> match = re.match(r"Hello(\s*)(.*)World!", "Hello              Python World!")
>>> match.groups()
('\t\t  ', 'Python ')

'''
模式 描述
^ 匹配字符串的开头
$ 匹配字符串的末尾。
. 匹配任意字符，除了换行符，当re.DOTALL标记被指定时， 则可以匹配包括换行符的任意字符。
[...] 用来表示一组字符,单独列出：[amk] 匹配'a'，'m'或'k'
[^...] 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。
re* 匹配0个或多个的表达式。
re+ 匹配1个或多个的表达式。
re? 匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式
re{ n} 精确匹配 n 个前面表达式。例如， o{2} 不能匹配 "Bob" 中的"o"，但是能匹配 "food" 中的两个
    o。
re{ n,} 匹配 n 个前面表达式。例如， o{2,} 不能匹配"Bob"中的"o"，
但能匹配 "fooooood"中的所有 o。"o{1,}" 等价于 "o+"。"o{0,}"
则等价于 "o*"。
re{ n, m} 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式a| b 匹配a或b
(re) 匹配括号内的表达式，也表示一个组
(?imx) 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。
(?-imx) 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。
(?: re) 类似 (...), 但是不表示一个组
(?imx: re) 在括号中使用i, m, 或 x 可选标志
(?-imx: re) 在括号中不使用i, m, 或 x 可选标志
(?#...) 注释.
```

```
(?= re) 前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，
    否则失败。但一旦所含表达式已经尝试
匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。
(?! re) 前向否定界定符。与肯定界定符相反； 当所含表达式不能在字符串当前位置匹配时成功
(?> re) 匹配的独立模式，省去回溯。
\w 匹配字母数字及下划线
\W 匹配非字母数字及下划线
\s 匹配任意空白字符，等价于 [\t\n\r\f].
\S 匹配任意非空字符
\d 匹配任意数字，等价于 [0-9].
\D 匹配任意非数字
\A 匹配字符串开始
\Z 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。
\z 匹配字符串结束
\G 匹配最后匹配完成的位置。
\b 匹配一个单词边界，也就是指单词和空格间的位置。例如，
'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配
"never" 中的 'er'。
\n, \t, 等. 匹配一个换行符。匹配一个制表符。等
\1...\9 匹配第n个分组的内容。
\10 匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。


用户名  /^[a-z0-9_-]{3,16}$/
密码    /^[a-z0-9_-]{6,18}$/
十六进制值  /^#?([a-f0-9]{6}|[a-f0-9]{3})$/
电子邮箱    /^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/
URL /^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
IP 地址 /((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5] | [01]?\d\d?)/
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5] | 2[0-4][0-9]|[01]?[0-9][0-9]?)$/
HTML 标签   /^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
删除代码\\注释  (?<!http:|\S)//.*$
Unicode编码中的汉字范围 /^[\u2E80-\u9FFF]+$/
'''
```

11 YNOI

11.1 带修查询能否连续重排为值域连续的序列（线段树 + 散列异或）（洛谷 P3792）

1 x y 修改 x 位置的值为 y

2 l r 查询区间 $[l,r]$ 是否可以重排为值域上连续的一段

```
struct Query {
    int opt, x, y;
} q[MAXN];
ull rnd[MAXN<<2], pre_rnd[MAXN<<2]; // 空间注意！
int a[MAXN];

class SEG { public:
    struct node {
        int l, r, minn;
        ull sum;
```

```
    } T[MAXN << 2];

    inline void push_up(int rt) {
        T[rt].minn = min(T[rt << 1].minn, T[rt << 1 | 1].minn);
        T[rt].sum = T[rt << 1].sum ^ T[rt << 1 | 1].sum;
    }

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            T[rt].minn = a[l];
            T[rt].sum = rnd[T[rt].minn];
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }

    void update(int rt, int pos, int val) {
        if (T[rt].l == T[rt].r) {
            T[rt].minn = val;
            T[rt].sum = rnd[T[rt].minn];
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) update(rt << 1, pos, val);
        else update(rt << 1 | 1, pos, val);
        push_up(rt);
    }

    ull query_sum(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
        int mid = (T[rt].l + T[rt].r) >> 1;
        ull ans = 0;
        if (L <= mid) ans ^= query_sum(rt << 1, L, R);
        if (R > mid) ans ^= query_sum(rt << 1 | 1, L, R);
        return ans;
    }

    int query_min(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].minn;
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = inf;
        if (L <= mid) ans = min(ans, query_min(rt << 1, L, R));
        if (R > mid) ans = min(ans, query_min(rt << 1 | 1, L, R));
        return ans;
    }
} tree;

int main() {
    srand(19260817);
    int n, m;   scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        Discrete::insert(a[i]), Discrete::insert(a[i] + 1);
```

```
    }

    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &q[i].opt, &q[i].x, &q[i].y);
        if (q[i].opt == 1) Discrete::insert(q[i].y), Discrete::insert(q[i].y + 1);
    }

    Discrete::init();
    for (int i = 1; i <= n; i++) {
        a[i] = val2id(a[i]);
    }

    pre_rnd[0] = 0;
    for (int i = 1; i < (MAXN<<2); i++) rnd[i] = Newrnd(), pre_rnd[i] = pre_rnd[i - 1] ^
        rnd[i];

    tree.build(1, 1, n);
    for (int i = 1; i <= m; i++) {
        if (q[i].opt == 1) {
            tree.update(1, q[i].x, val2id(q[i].y));
        } else {
            int l = tree.query_min(1, q[i].x, q[i].y);
            int r = l + (q[i].y - q[i].x);
            ull tmp1 = pre_rnd[r] ^ pre_rnd[l-1];
            ull tmp2 = tree.query_sum(1, q[i].x, q[i].y);
            if (tmp1 == tmp2) printf("damushen\n"); // 能够重排为值域上连续的一段
            else printf("yuanxing\n");  // 不能够重排为值域上连续的一段
        }
    }
}
```

12 习题整理

12.1 可重边集的点能否和当前询问边构成三角形（20 牛客 2H）（动态点开线段树）

input	output
8	
1 1	
3 1	No
1 1	
3 2	No
3 1	Yes
1 2	
2 1	
3 1	No

```
map<int, int> ma;
SEG tree; // 动态点开线段树维护最小值，记得初始化val[0] = inf;
int root;
void add(int x) {
    ma[x]++;
    if (ma[x] == 1) {
        auto it = ma.lower_bound(x); auto R = it;
```

```
        if (++R != ma.end() && R-> second == 1) root = tree.update(root, R->first, R->first-
            x, 1, MAXR);
        if (it != ma.begin()) root = tree.update(root, x, x - (--it)->first, 1, MAXR);
        else root = tree.update(root, x, inf, 1, MAXR);
    } else if (ma[x] == 2) root = tree.update(root, x, 0, 1, MAXR);
}
void del(int x) {
    auto it = ma.lower_bound(x);
    ma[x]--;
    int L = -inf;
    if (it != ma.begin()) {
        L = (--it) -> first;
        ++it;
    }
    if (ma[x] == 0) {
        if ((++it) != ma.begin() && it->second == 1)
            root = tree.update(root, it->first, it->first-L, 1, MAXR);
        root = tree.update(root, x, inf, 1, MAXR);
        ma.erase(x);
    } else if (ma[x] == 1) root = tree.update(root, x, x-L, 1, MAXR);
}
int ask(int x) {
    auto it = ma.lower_bound(x/2+1);
    if (it == ma.end()) return inf;
    if (it->second > 1) return it->first;
    if (it != ma.begin()) {
        auto L = it; --L;
        if (L ->first + it ->first > x) return it->first;
    }
    if ((++it) != ma.begin()) return it->first;
    return inf;
}

int main() {
    tree.init();
    int q; scanf("%d", &q);
    while (q--) {
        int opt, x; scanf("%d%d", &opt, &x);
        if (opt == 1) add(x);
        else if (opt == 2) del(x);
        else {
            if (tree.query_min(root, ask(x), MAXR, 1, MAXR) < x) printf("Yes\n");
            else printf("No\n");
        }
    }
}
```

12.2 左偏树离线处理查询成立最多数（HDU5575）

0 代表没有水，1 代表有水

```
struct Query {
    int x, y;
    Query() {}
    Query(int _x, int _y) { x = _x, y = _y; }
```

input	output
3 4 3 4 1 3 1 2 1 0 2 2 0 3 3 1	3

```
};
int fa[MAXN];
int find(int x) {
    if (x == fa[x]) return x;
    else return fa[x] = find(fa[x]);
}
vector<Query> q;
int LH[MAXN], RH[MAXN], L[MAXN], R[MAXN];
int root[MAXN], siz[MAXN], edge[MAXN];
void join(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx == fy) return;
    fa[fy] = fx;
    if (fx < fy) RH[fx] = RH[fy], L[R[fx]] = fx, R[fx] = R[fy];
    else LH[fx] = LH[fy], R[L[fx]] = fx, L[fx] = L[fy];
    root[fx] = tree.merge(root[fx], root[fy]);
    edge[fx] += edge[fy];
    siz[fx] += siz[fy];
}
int main() {
    int T;  scanf("%d", &T);
    while (T--) {
        int n, m;   scanf("%d%d", &n, &m);
        LH[1] = RH[n] = inf;
        L[n] = n-1;
        for (int i = 1; i < n; i++) {
            scanf("%d", &RH[i]);
            LH[i + 1] = RH[i];
            L[i] = i - 1, R[i] = i + 1;
        }
        tree.init(); q.clear();
        for (int i = 1; i <= n; i++) root[i] = 0;   // init LT's root
        int res = 0;
        for (int i = 1; i <= m; i++) {
            int x, y, z;    scanf("%d%d%d", &x, &y, &z);
            if (z == 1) {
                q.pb(Query(x, y + 1));
            } else {
                if (root[x]) {
                    root[x] = tree.insert(root[x], y);
                } else root[x] = tree.Newnode(y);
                res++;
            }
        }
        sort(q.begin(), q.end(), [&](const Query &ta, const Query &tb) {
            if (ta.y != tb.y) return ta.y < tb.y;
            else return ta.x < tb.x;
        });
```

```
        for (int i = 1; i <= n; i++) fa[i] = i;
        for (int i = 1; i <= n; i++) siz[i] = edge[i] = 0;
        for (auto &e : q) {
            int x = find(e.x), y = e.y;
            // 向左溢出
            while (y > LH[x]) join(x, L[x]), x = find(x);
            // 向右溢出
            while (y > RH[x]) join(x, R[x]), x = find(x);
            // 删除水位以下的X
            while (!tree.isempty(root[x]) && tree.top(root[x]) < y) {
                root[x] = tree.pop(root[x]);
                siz[x]++;
            }
            // update result
            if (++edge[x] >= siz[x]) {
                res += (edge[x] - siz[x]);
                siz[x] = edge[x] = 0;
            }
        }
        printf("Case #%d: %d\n", kass++, res);
    }
}
```

12.3 图上加边最多最少连通块（线段树二分贪心）（ZOJ4100）

时间复杂度：$O(q \log^2 n)$.

```
/*
    input   ouput
    5 5
    1 1 2
    2 1     3 3
    1 1 3
    2 1     2 3
    2 3     1 2
*/
ll com[MAXN];
int n;
class SEG { public:
    struct node {
        int l, r, cnt, sum;
        ll edge;
    } T[MAXN << 2];
    inline void push_up(int rt) {
        T[rt].cnt = T[rt << 1].cnt + T[rt << 1 | 1].cnt;
        T[rt].edge = T[rt << 1].edge + T[rt << 1 | 1].edge;
        T[rt].sum = T[rt << 1].sum + T[rt << 1 | 1].sum;
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            if (l == 1) {
                T[rt].edge = com[l] * n, T[rt].cnt = n, T[rt].sum = n;
            } else {
                T[rt].edge = 0, T[rt].cnt = 0, T[rt].sum = 0;
```

```
            }
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
    void update(int rt, int pos, int v) {
        if (T[rt].l == T[rt].r) {
            T[rt].cnt += v;
            T[rt].sum += v * T[rt].l;
            T[rt].edge += com[T[rt].l] * v;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) update(rt << 1, pos, v);
        else update(rt << 1 | 1, pos, v);
        push_up(rt);
    }
    int query(int rt, ll k, int vs, ll vk) {    // 二分查找
        if (T[rt].l == T[rt].r) {
            int L = 0, R = T[rt].cnt;
            while (L < R) {
                int mid = (L + R) >> 1;
                if (com[mid * T[rt].l + vs] - mid * com[T[rt].l] - vk < k) L = mid + 1;
                else R = mid;
            }
            return L;
        }
        if (com[vs + T[rt << 1 | 1].sum] - T[rt << 1 | 1].edge >= k) return query(rt << 1 | 1,
            k, vs, vk);
        else return T[rt << 1 | 1].cnt + query(rt << 1, k, vs + T[rt << 1 | 1].sum, vk + T[rt
            << 1 | 1].edge);
    }
} tree;
int cnt[MAXN];  ll edge[MAXN];
int fa[MAXN];
int find(int x) {
    if (fa[x] == x) return x;
    else return fa[x] = find(fa[x]);
}
int main() {
    int T;  scanf("%d", &T);
    for (int i = 1; i < MAXN; i++) com[i] = (ll) i * (i - 1) / 2;
    while (T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) fa[i] = i;
        for (int i = 1; i <= n; i++) cnt[i] = 1, edge[i] = 0;
        tree.build(1, 1, n);
        int q;  scanf("%d", &q);
        int blocks = n;
        ll free = 0;
        while (q--) {
            int opt;    scanf("%d", &opt);
            if (opt == 1) {
                int x, y;   scanf("%d%d", &x, &y);
```

```
                int fx = find(x), fy = find(y);
                if (fx == fy) {
                    edge[fx]++; free--;
                } else {
                    blocks--;
                    free -= com[cnt[fx]] - edge[fx];
                    free -= com[cnt[fy]] - edge[fy];
                    tree.update(1, cnt[fx], -1);
                    tree.update(1, cnt[fy], -1);
                    fa[fx] = fy;
                    cnt[fy] += cnt[fx];
                    edge[fy] += edge[fx] + 1;
                    free += com[cnt[fy]] - edge[fy];
                    tree.update(1, cnt[fy], 1);
                }
            } else {
                ll k;
                scanf("%lld", &k);
                printf("%lld ", max(1ll, (ll) blocks - k)); // minn
                if (free >= k) printf("%d\n", blocks);  // maxx
                else {
                    k -= free;
                    int tmp = tree.query(1, k, 0, 0);
                    printf("%d\n", blocks - tmp + 1);   // maxx
                }
            }
        }
    }
}
```

12.4 错排后字典序最小（ZOJ4102）

时间复杂度：$O(n \log n)$.

```
/*
    input       output
    4 1 3 2     1 2 4 3
    1 1 2 3     2 3 1 1
    1 1 1       Impossible
*/
int a[MAXN], ban[MAXN], todo[MAXN];
int res[MAXN];  // 字典序最小的存在res中
int main() {
    priority_queue<pii > pq;
    vector<int> vec;
    int n;  scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) ban[i] = todo[i] = 0;

    int flag = 1;
    for (int i = 1; i <= n; i++) ban[a[i]]++, todo[a[i]]++;
    for (int i = 1; i <= n; i++) {
        if (ban[i]) {
            vec.push_back(i);
            pq.push(mp(ban[i] + todo[i], i));
```

```
            if (ban[i] + todo[i] > n) {
                flag = 0;   break;
            }
        }
    }
    if (!flag) {    printf("Impossible\n"); continue;   }

    int pos = 0;
    for (int i = 1; i <= n; i++) {
        pii u = pq.top();   pq.pop();
        while (u.first != ban[u.second] + todo[u.second]) u = pq.top(), pq.pop();
        if (u.first == n - i + 1 && u.second != a[i]) {
            res[i] = u.second;
        } else {
            pq.push(u);
            for (int j = pos; j < SZ(vec); j++) {
                if (vec[j] != a[i] && todo[vec[j]] > 0) {
                    res[i] = vec[j];    break;
                }
            }
        }
        todo[res[i]]--; ban[a[i]]--;
        pq.push(mp(todo[res[i]] + ban[res[i]], res[i]));
        pq.push(mp(todo[a[i]] + ban[a[i]], a[i]));
        if (todo[vec[pos]] == 0) pos++;
    }
}
```

12.5 若干个区间选数字使相与之和最小（ZOJ4135）

```
/*
    input               output
    3                   6
    [0,8],[2,6],[3,9]
    3                   1
    [0,7],[0,3],[4,5]
*/
int L[MAXN], R[MAXN];
int main() {
    int n;  scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d%d", &L[i], &R[i]);
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        int tmp = (1 << i);
        int flag = 1;
        for (int j = 1; j <= n; j++) {
            if (tmp > R[j]) {
                flag = 0;   break;
            }
        }
        if (flag) { // can provide 1
            res += tmp;
            for (int j = 1; j <= n; j++) {
                L[j] = max(L[j], tmp) - tmp;
```

```
                R[j] = R[j] - tmp;
            }
        } else {
            for (int j = 1; j <= n; j++) {
                if (L[j] >= tmp && R[j] >= tmp) {
                    L[j] -= tmp, R[j] -= tmp;
                } else if (L[j] < tmp && R[j] >= tmp) {
                    R[j] = tmp - 1;
                }
            }
        }
    }
    printf("%d\n", res);
}
```

12.6 2019 徐州 L

给一颗字符串树，1 为根，求从哪个结点向上 L 长度的字符串共有多少种本质不同的字符串

```
/*
    input       output
    6 3
    ABABBA
    1 1 3 3 4
    2 2         3
    2 1         3
    6 4         1
*/
class SAM { public:
    struct Edge {
        int to, nex;
    } e[MAXN];
    int head[MAXN], tol;

    void addEdge(int u, int v) {
        e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
    }
    void dfs(int u) {
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            dfs(v);
            val[u] += val[v];
        }
    }

    int fa[MAXN][32];
    void build() {
        for (int i = 1; i <= rt; i++) head[i] = -1;
        for (int i = 2; i <= rt; i++) addEdge(link[i], i);
        dfs(1);
        for (int i = 1; i <= rt; i++) fa[i][0] = link[i];
        for (int i = 1; i < 32; i++) {
            for (int j = 1; j <= rt; j++) {
                fa[j][i] = fa[fa[j][i - 1]][i - 1];
            }
```

```
        }
    }

    int query(int X, int L) {
        for (int i = 31; ~i; i--) { if (maxlen[fa[X][i]] >= L) X = fa[X][i];    }
        return val[X];
    }

    void debug() {
        for (int i = 1; i <= rt; i++) printf("link[%d] = %d\n", i, link[i]);
        for (int i = 1; i <= rt; i++) printf("val[%d] = %d\n", i, val[i]);
        for (int i = 1; i <= rt; i++) printf("maxlen[%d] = %d\n", i, maxlen[i]);
    }
} sa;

struct Edge {
    int to, nex;
} e[MAXNODE];
int head[MAXNODE], tol;

void addEdge(int u, int v) {
    e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}

char str[MAXNODE];  int pos[MAXN];

struct node {
    int v, last;
    node(int _v = 0, int _last = 0) : v(_v), last(_last) {}
};

void bfs() {
    queue<node> q;
    q.push(node(1, 1));
    int last = 1;
    while (!q.empty()) {
        node u = q.front(); q.pop();
        int nls = sa.insert(str[u.v]-'A', u.last);
        pos[u.v] = nls;
        for (int i = head[u.v]; ~i; i = e[i].nex) {
            int to = e[i].to;
            q.push(node(to, nls));
        }
    }
//    pos[u] = last = sa.insert(str[u] - 'A', last);
//    for (int i = head[u]; ~i; i = e[i].nex) {
//        int v = e[i].to;
//        dfs(v, last);
//    }
}

void init(int n) {  for (int i = 1; i <= n; i++) head[i] = -1;  }
int main() {
    int n, q;   scanf("%d%d", &n, &q);
    init(n);
    scanf("%s", str + 1);
```

```
    for (int i = 2; i <= n; i++) {
        int x;  scanf("%d", &x);
        addEdge(x, i);
    }
    bfs();
//    dfs(1, 1);  // make sam
    sa.build(); // get fail tree
    while (q--) {
        int X, L;   scanf("%d%d", &X, &L);
        printf("%d\n", sa.query(pos[X], L));
    }
}
```

12.7 双哈希（2019CCPC 哈尔滨 L）

给定一个 n 个数的数字序列，第 i 个数为 $a[i]$，每次操作会将 $a[i]$ 插入或移到最前端：

1. 若 $a[i]$ 已经在序列中出现过，则将其移到最前端，并删除原出现位置
2. 若 $a[i]$ 未出现过，则直接将其插入到最前端

```
/*  [input]         [output]
    7 5
    4 3 4 2 3 1 4
    1 4             Yes
    2 2 3           No
    3 3 2 1         No
    4 4 1 3 2       Yes
    4 3 4 0 0       Yes
*/
const int MAXN = 5e3 + 5;
const ll seed1 = 233;
const ll seed2 = 19260817;
int arr[MAXN], brr[MAXN], pos[MAXN];

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        int n, q; scanf("%d%d", &n, &q);
        vector<ll> v1[MAXN], v2[MAXN], t1[MAXN], t2[MAXN];
        for (int i = 1; i <= n; i++) pos[i] = 0; // init pos
        for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);
        for (int i = 1; i <= n; i++) {
            ll hash1 = 0, hash2 = 0;
            int cnt = 0;
            for (int j = i; j >= 1; j--) {
                if (pos[arr[j]] <= j) {
                    pos[arr[j]] = j;
                    cnt++;
                    hash1 = hash1 * seed1 + arr[j] + 3;
                    hash2 = hash2 * seed2 + arr[j] + 3;
                    v1[cnt].push_back(hash1);
                    v2[cnt].push_back(hash2);
                }
                if (j == 1) {
                    t1[cnt].push_back(hash1);
                    t2[cnt].push_back(hash2);
```

```
                }
            }
        }
        while (q--) {
            int m; scanf("%d", &m);
            for (int i = 1; i <= m; i++) scanf("%d", &brr[i]);
            if (!brr[m]) {  // have other 0
                while(!brr[m] && m) m--;
                if (!m) {
                    printf("Yes\n");
                    continue;
                }
                ll hash1 = 0, hash2 = 0;
                for (int i = 1; i <= m; i++) hash1 = hash1 * seed1 + brr[i] + 3;
                for (int i = 1; i <= m; i++) hash2 = hash2 * seed2 + brr[i] + 3;
                int flag = 0;
                for (int i = 0; i < SZ(t1[m]); i++) {
                    if (t1[m][i] == hash1 && t2[m][i] == hash2) {
                        flag = 1; break;
                    }
                }
                if (flag) printf("Yes\n");
                else printf("No\n");
            } else {
                ll hash1 = 0, hash2 = 0;
                for (int i = 1; i <= m; i++) hash1 = hash1 * seed1 + brr[i] + 3;
                for (int i = 1; i <= m; i++) hash2 = hash2 * seed2 + brr[i] + 3;
                int flag = 0;
                for (int i = 0; i < SZ(v1[m]); i++) {
                    if (v1[m][i] == hash1 && v2[m][i] == hash2) {
                        flag = 1; break;
                    }
                }
                if (flag) printf("Yes\n");
                else printf("No\n");
            }
        }
    }
}
```

12.8 最短的涵盖从 1 到 i 的线段（线段树）

给定 n 个数的序列，对于每一个 m，求最短的涵盖从 1 到 i 的线段的长度。

$R_{i,l}$ 代表以 l 为线段左端点，包含数从 1 i 中所有数字的最近右端点。

对于 $i+1$ 来说，其所在的位置为 $p_1, p_2, p_3, ..., p_k$。这些数将长度为 n 的区间划分为若干部分，如下所示：

[1, p_1]，[p_1+1, p_2]，...，[p_k+1,n] 对于每一个区间，其内部所有的 l 在从 $R_{i,l}$ 转移至 $R_{i+1,l}$ 时，式子为 $R_{i+1,l} = max(R_{i,l}, p_k)$

其中，p_k 代表其所在区间的右端点。

特别的，当所在的 l 位于最后一个线段中，不可能构成包含数从 1 i 中所有数字的线段，此时应将 $R_{i+1,l}$ 置为无穷大，也就是取不到

显然在任何情况下 $R_{i,l}$ 是 ** 单调递增 ** 的。最后，将问题转移成了对于每一个区间，找到 $R_{i,l} < p_k$ 的一段数并将其值赋为 p_k 。这个操作可以利用线段树进行维护。

```
/*  [input]     [output]
    5 3         1 3 3
    1 3 2 3 1
```

```
*/
int init_val[MAXN];
class SEG { public:
    struct node {
        int l, r;
        int val, minn;  // val是维护的右端点，minn是线段长度
    } T[MAXN << 2];
    int lazy[MAXN << 2];
    inline void push_up(int rt) {
        T[rt].minn = min(T[rt << 1].minn, T[rt << 1 | 1].minn);
        T[rt].val = min(T[rt << 1].val, T[rt << 1 | 1].val);
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            T[rt].minn = init_val[l] - r + 1;
            T[rt].val = init_val[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
    inline void push_down(int rt) {
        if (lazy[rt]) {
            T[rt << 1].val = lazy[rt], T[rt << 1 | 1].val = lazy[rt];
            T[rt << 1].minn = lazy[rt] - T[rt << 1].r + 1, T[rt << 1 | 1].minn = lazy[rt] -
                T[rt << 1 | 1].r + 1;
            lazy[rt << 1] = lazy[rt], lazy[rt << 1 | 1] = lazy[rt];
            lazy[rt] = 0;
        }
    }
    int query_left(int rt, int L, int R, int v) {
        if (T[rt].r < L || T[rt].l > R) return -1;
        if (T[rt].l == T[rt].r) return T[rt].l;
        push_down(rt);
        int ans = -1;
        if (T[rt << 1].val < v) ans = query_left(rt << 1, L, R, v);
        if (ans == -1 && T[rt << 1 | 1].val < v) ans = query_left(rt << 1 | 1, L, R, v);
        return ans;
    }
    int query_right(int rt, int L, int R, int v) {
        if (T[rt].r < L || T[rt].l > R) return -1;
        if (T[rt].l == T[rt].r) return T[rt].l;
        push_down(rt);
        int ans = -1;
        if (T[rt << 1 | 1].val < v) ans = query_right(rt << 1 | 1, L, R, v);
        if (ans == -1 && T[rt << 1].val < v) ans = query_right(rt << 1, L, R, v);
        return ans;
    }
    void change(int rt, int L, int R, int v) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].val = v;
            T[rt].minn = v - T[rt].r + 1;
            lazy[rt] = v;
            return;
```

```
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) change(rt << 1, L, R, v);
        if (R > mid) change(rt << 1 | 1, L, R, v);
        push_up(rt);
    }
} tree;
int a[MAXN];
vector<int> vec[MAXN];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) vec[a[i]].pb(i);
    int pos = inf;
    for (int i = n; i >= 1; i--) {
        if (a[i] == 1) pos = i;
        init_val[i] = pos;
    }
    tree.build(1, 1, n);
    printf("%d", tree.T[1].minn);   // m = 1
    for (int i = 2; i <= m; i++) {
        for (int j = 0; j < SZ(vec[i]); j++) {
            int l, r, p = vec[i][j];
            if (j == 0) { // segment no 1
                l = 1, r = vec[i][j];
            } else l = vec[i][j - 1] + 1, r = vec[i][j];
            int left = tree.query_left(1, l, r, p);
            int right = tree.query_right(1, l, r, p);
            if (left == -1 && right == -1) {

            } else {
                tree.change(1, left, right, p);
            }
        }
        if (vec[i][SZ(vec[i]) - 1] == n) {}
        else {
            tree.change(1, vec[i][SZ(vec[i]) - 1] + 1, n, inf);   // 忘记+1导致debug了好久
        }
        printf(" %d", tree.T[1].minn);
    }
}
```

12.9 二分线段树 DP（成电多校 HDU6606）

给定 n 个数，在不改变数的排序状态的情况下，取前 x 个数，将这 x 个数分为 k 段，要求每段的和的最大值最小，问这个值是多少。

考虑对二分后的值进行 check，需要在二分后中的 check 中进行 DP 操作。

题解给出了一个 DP 转移式子：$dp[i]=max(dp[j])+1$，其中 $dp[j]$ 要求满足：$sum[i]-sum[j]\le x$（$sum[i]$ 代表第 i 位的前缀和，x 为枚举的 mid）并且 $i\le j$。

将 $sum[i]-sum[j]\le x$ 转化 $sum[j]\ge sum[i]-x$。

将 $sum[i]$ 离散化后作为线段树下标进行维护，维护的值为 $dp[i]$。那么就是在区间 $[val2id(sum[i]-x),m]$ 中找到最大的值，其中 $val2id(sum[i]-x)$ 是离散化后的 $sum[i]-x$ 的值，m 为离散化后的数字数目。

```
/*  [input]         [output]
```

```
    2
    4 2             2
    3 -2  4 -2
    5 4             -1
    -1 -1 -1 -1 6
*/
class SEG { public:
    struct node {
        int l, r, maxx;
    } T[MAXN << 2];

    inline void push_up(int rt) {
        T[rt].maxx = max(T[rt << 1].maxx, T[rt << 1 | 1].maxx);
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            T[rt].maxx = -inf;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
    void update(int rt, int pos, int v) {
        if (T[rt].l == T[rt].r) {
            T[rt].maxx = v;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) update(rt << 1, pos, v);
        else update(rt << 1 | 1, pos, v);
        push_up(rt);
    }
    int query(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].maxx;
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = -inf;
        if (L <= mid) ans = max(ans, query(rt << 1, L, R));
        if (R > mid) ans = max(ans, query(rt << 1 | 1, L, R));
        return ans;
    }
} tree;
int n, k;
int a[MAXN];
ll sum[MAXN];
bool check(ll mid) {
    tree.build(1, 1, Discrete::blen);
    int pos = val2id(0);
    tree.update(1, pos, 0);
    for (int i = 1; i <= n; i++) {
        pos = val2id(sum[i] - mid);
        int tmp = tree.query(1, pos, Discrete::blen);
        tree.update(1, val2id(sum[i]), tmp + 1);
    }
    if (tree.T[1].maxx >= k) return 1;
```

```
    else return 0;
}
int main() {
    int T; scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &k);
        Discrete::btol = 0;
        for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
        Discrete::insert(0);
        Discrete::insert(-llinf), Discrete::insert(llinf);      // 防止出锅
        for (int i = 1; i <= n; i++) sum[i] = sum[i - 1] + a[i], Discrete::insert(sum[i]);
        Discrete::init();

        ll L = -llinf, R = llinf;   // 二分范围注意！不然会在超大数据挂掉……
        while (L < R) {
            ll mid = (L + R) >> 1;
            if (check(mid)) R = mid;
            else L = mid+1;
        }
        printf("%lld\n", L);
    }
}
```

12.10 CRT+ 线段树乱搞（HDU5238）

给出 n 个操作，只含有加、乘、次方计算。有两种操作：

1. 询问一个数 x，将这个 x 把所有给出需要的计算式来计算一遍。
2. 将其中某一步计算修改。

```
const int muls[4] = {7, 13, 17, 19}; // 4 divers
namespace CRT {
    int a[4], m[4];
    void exgcd(int a, int b, int &x, int &y) {
        if (b == 0) {
            x = 1, y = 0;
            return;
        }
        exgcd(b, a % b, x, y);
        int z = x;
        x = y, y = z - y * (a / b);
    }
    int solve() {
        int mul_sum = 29393;
        int ans = 0;
        for (int i = 0; i < 4; i++) {
            int M = mul_sum / m[i];
            int x = 0, y = 0;
            exgcd(M, m[i], x, y);
            ans += a[i] * M * (x < 0 ? x + m[i] : x);
        }
        return ans % mul_sum;
    }
}
using CRT::solve;
int qpow[4][20][MAXN];
```

```
void init() {
    for (int i = 0; i < 4; i++) {
        CRT::m[i] = muls[i];
    }
    for (int i = 0; i < 4; i++) {
        int p = muls[i];
        for (int j = 0; j < p; j++) {
            int now = 1;
            for (int k = 0; k < MAXN; k++) {
                qpow[i][j][k] = now;
                now = now * j % p;
            }
        }
    }
}
int opt[MAXN], a[MAXN];
class SEG { public:
    struct node {
        int l, r;
        int val[4][20];
    } T[MAXN << 2];
    inline void push_up(int rt) {
        for (int i = 0; i < 4; i++) {
            int p = muls[i];
            for (int j = 0; j < p; j++) {
                T[rt].val[i][j] = T[rt << 1 | 1].val[i][T[rt
                        << 1].val[i][j]]; // T[rt<<1].val[i][j] -> calculate left,
                            T[rt<<1|1].val[i][left] -> answer
            }
        }
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            for (int i = 0; i < 4; i++) {
                int p = muls[i];
                for (int j = 0; j < p; j++) {
                    int now = j;
                    if (opt[l] == 0) now = (now + a[l]) % p;
                    else if (opt[l] == 1) now = (now * a[l]) % p;
                    else now = qpow[i][now][a[l]]; // now ^ (a[l]) % mul[i]
                    T[rt].val[i][j] = now;
                }
            }
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
    void update(int rt, int pos) {
        if (T[rt].l == T[rt].r) {
            for (int i = 0; i < 4; i++) {
                int p = muls[i];
                for (int j = 0; j < p; j++) {
                    int now = j;
```

```
                    if (opt[T[rt].l] == 0) now = (now + a[T[rt].l]) % p;
                    else if (opt[T[rt].l] == 1) now = (now * a[T[rt].l]) % p;
                    else now = qpow[i][now][a[T[rt].l]]; // now ^ (a[l]) % mul[i]
                    T[rt].val[i][j] = now;
                }
            }
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) update(rt << 1, pos);
        else update(rt << 1 | 1, pos);
        push_up(rt);
    }
} tree;

int main() {
    init();
    int T; scanf("%d", &T); int kass = 1;
    while (T--) {
        int n, m; scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) {
            char s[10]; scanf("%s", s);
            if (s[0] == '+') opt[i] = 0;
            else if (s[0] == '*') opt[i] = 1;
            else opt[i] = 2;
            int s_len = strlen(s);
            a[i] = 0;
            for (int j = 1; j < s_len; j++) a[i] = a[i] * 10 + s[j] - '0';
        }
        tree.build(1, 1, n);
        printf("Case #%d:\n", kass++);
        while (m--) {
            int ops; scanf("%d", &ops);
            if (ops == 1) {
                int x; scanf("%d", &x);
                for (int i = 0; i < 4; i++) {
                    int p = muls[i];
                    CRT::a[i] = tree.T[1].val[i][x % p];
                }
                printf("%d\n", solve());
            } else {
                int pos; scanf("%d", &pos);
                char s[10]; scanf("%s", s);
                if (s[0] == '+') opt[pos] = 0;
                else if (s[0] == '*') opt[pos] = 1;
                else opt[pos] = 2;
                int s_len = strlen(s);
                a[pos] = 0;
                for (int j = 1; j < s_len; j++) a[pos] = a[pos] * 10 + s[j] - '0';
                tree.update(1, pos);
            }
        }

    }
}
```

12.11 曼哈顿距离转切比雪夫距离 + 大力分类讨论线段树（2018ICPC 沈阳 E）

在无限宽广的二维平面上，分布着 n 个忍者，他们各自的派别，有 m 个操作，共 3 种：

1. 将编号为 k 的忍者的位置变为 $(x'+x, y'+y)$。
2. 将编号为 k 的忍者的派别变为 c。
3. 询问编号在 $[l,r]$ 之间的忍者中，派别不同的最大的曼哈顿距离。

```
/*  [input]     [output]
    2 8
    0 0 1
    1 1 2
    3 1 2       2
    1 1 1 1
    3 1 2       0
    1 1 1 1
    2 1 2
    3 1 2       0
    2 1 1
    3 1 2       2
*/
pil a1[MAXN], a2[MAXN];
struct node {
    int l, r;
    pil fi_max, se_max;
    pil fi_min, se_min;
    node() {
        l = 0, r = 0;
        fi_max = se_max = mp(0, -llinf);
        fi_min = se_min = mp(0, llinf);
    }
};
class JLS { public:
    node T[MAXN << 2];
    inline node merge(const node &L, const node &R) {
        node ans;
        ans.l = L.l, ans.r = R.r;
        // max
        if (L.fi_max.second > R.fi_max.second) ans.fi_max = L.fi_max;
        else ans.fi_max = R.fi_max;

        if (L.fi_max.first != R.fi_max.first) {
            if (L.fi_max.second <= R.fi_max.second) ans.se_max = L.fi_max;
            else ans.se_max = R.fi_max;
        }
        if (L.se_max.second > ans.se_max.second && L.se_max.first != ans.fi_max.first) {
            ans.se_max = L.se_max;
        }
        if (R.se_max.second > ans.se_max.second && R.se_max.first != ans.fi_max.first) {
            ans.se_max = R.se_max;
        }

        // min
        if (L.fi_min.second < R.fi_min.second) ans.fi_min = L.fi_min;
        else ans.fi_min = R.fi_min;

        if (L.fi_min.first != R.fi_min.first) {
```

```
50            if (L.fi_min.second >= R.fi_min.second) ans.se_min = L.fi_min;
51            else ans.se_min = R.fi_min;
52        }
53        if (L.se_min.second < ans.se_min.second && L.se_min.first != ans.fi_min.first) {
54            ans.se_min = L.se_min;
55        }
56        if (R.se_min.second < ans.se_min.second && R.se_min.first != ans.fi_min.first) {
57            ans.se_min = R.se_min;
58        }
59
60        return ans;
61    }
62
63    void update1(int rt, int pos) {
64        if (T[rt].l == T[rt].r) {
65            T[rt].fi_max = a1[T[rt].l], T[rt].se_max = mp(0, -llinf);
66            T[rt].fi_min = a1[T[rt].l], T[rt].se_min = mp(0, llinf);
67            return;
68        }
69        int mid = (T[rt].l + T[rt].r) >> 1;
70        if (pos <= mid) update1(rt << 1, pos);
71        else update1(rt << 1 | 1, pos);
72        T[rt] = merge(T[rt << 1], T[rt << 1 | 1]);
73    }
74
75    void update2(int rt, int pos) {
76        if (T[rt].l == T[rt].r) {
77            T[rt].fi_max = a2[T[rt].l], T[rt].se_max = mp(0, -llinf);
78            T[rt].fi_min = a2[T[rt].l], T[rt].se_min = mp(0, llinf);
79            return;
80        }
81        int mid = (T[rt].l + T[rt].r) >> 1;
82        if (pos <= mid) update2(rt << 1, pos);
83        else update2(rt << 1 | 1, pos);
84        T[rt] = merge(T[rt << 1], T[rt << 1 | 1]);
85    }
86
87    void build1(int rt, int l, int r) {
88        T[rt].l = l, T[rt].r = r;
89        if (l == r) {
90            T[rt].fi_max = a1[l], T[rt].se_max = mp(0, -llinf);
91            T[rt].fi_min = a1[l], T[rt].se_min = mp(0, llinf);
92            return;
93        }
94        int mid = (l + r) >> 1;
95        build1(rt << 1, l, mid), build1(rt << 1 | 1, mid + 1, r);
96        T[rt] = merge(T[rt << 1], T[rt << 1 | 1]);
97    }
98
99    void build2(int rt, int l, int r) {
100       T[rt].l = l, T[rt].r = r;
101       if (l == r) {
102           T[rt].fi_max = a2[l], T[rt].se_max = mp(0, -llinf);
103           T[rt].fi_min = a2[l], T[rt].se_min = mp(0, llinf);
104           return;
105       }
```

```
        int mid = (l + r) >> 1;
        build2(rt << 1, l, mid), build2(rt << 1 | 1, mid + 1, r);
        T[rt] = merge(T[rt << 1], T[rt << 1 | 1]);
    }

    node query(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt];
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (R <= mid) return query(rt << 1, L, R);
        else if (L > mid) return query(rt << 1 | 1, L, R);
        else {
            node ansl = query(rt << 1, L, R), ansr = query(rt << 1 | 1, L, R);
            return merge(ansl, ansr);
        }
    }
} tree1, tree2;

int main() {
    int T; scanf("%d", &T); int kass = 1;
    while (T--) {
        int n, m; scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) {
            ll x, y; int c; scanf("%lld%lld%d", &x, &y, &c);
            a1[i].first = c, a1[i].second = x + y;
            a2[i].first = c, a2[i].second = x - y;
        }
        tree1.build1(1, 1, n), tree2.build2(1, 1, n);
        printf("Case #%d:\n", kass++);
        while (m--) {
            int opt;
            scanf("%d", &opt);
            if (opt == 1) {
                int k; ll x, y; scanf("%d%lld%lld", &k, &x, &y);
                a1[k].second += x + y, a2[k].second += x - y;
                tree1.update1(1, k), tree2.update2(1, k);
            } else if (opt == 2) {
                int k, c; scanf("%d%d", &k, &c);
                a1[k].first = c, a2[k].first = c;
                tree1.update1(1, k), tree2.update2(1, k);
            } else {
                int l, r; scanf("%d%d", &l, &r);
                if (l == r) {
                    printf("0\n");
                } else {
                    ll res = -1;
                    node ans1 = tree1.query(1, l, r);
                    if (ans1.fi_max.first != ans1.fi_min.first)
                        res = max(res, ans1.fi_max.second - ans1.fi_min.second);
                    if (ans1.fi_max.first != ans1.se_min.first)
                        res = max(res, ans1.fi_max.second - ans1.se_min.second);
                    if (ans1.se_max.first != ans1.fi_min.first)
                        res = max(res, ans1.se_max.second - ans1.fi_min.second);

                    node ans2 = tree2.query(1, l, r);
                    if (ans2.fi_max.first != ans2.fi_min.first)
                        res = max(res, ans2.fi_max.second - ans2.fi_min.second);
```

```
                    if (ans2.fi_max.first != ans2.se_min.first)
                        res = max(res, ans2.fi_max.second - ans2.se_min.second);
                    if (ans2.se_max.first != ans2.fi_min.first)
                        res = max(res, ans2.se_max.second - ans2.fi_min.second);

                    if (res == -1) printf("0\n");
                    else printf("%lld\n", res);
                }
            }
        }
    }
}
```

12.12 只选一个区间听课（CF1452E）

给定 m 个人，他们感兴趣的区间在 $[1,n]$ 的区间内，有两个老师进行授课，老师们的授课区间长度为 k，每个人只能听一个老师授课，求所有人听到课的长度总和最大。

对于一个老师和一个同学，当老师区间从左向右移动时，同学能听到的课的长度为 0-> 最大->0，因此贪心地想能够在中点处达到最大。

对每个同学的中点位置进行排序，将中点位置小的排在前面，重点位置大的排在后面，从后往前扫，每次记录 $[j,m]$ 位同学能够达到的和的最大值，用这个作为第二个老师授课的根据。

再枚举第一个老师的授课位置，枚举听第一个老师授课的同学数量，再加上之前记录的听第二个授课老师的 $[j,m]$ 位同学能够达到的和的最大值，求最大值。

```
/*  [input]     [output]
    5 4 5       8
    1 2
    2 3
    3 4
    4 5
*/
struct node {
    int l, r;
} nd[MAXN];
int sum[MAXN];
int main() {
    int n, m, k; scanf("%d%d%d", &n, &m, &k);
    for (int i = 1; i <= m; i++) scanf("%d%d", &nd[i].l, &nd[i].r);
    sort(nd + 1, nd + 1 + m, [&](const node &ta, const node &tb) {
        return ta.l + ta.r < tb.l + tb.r;
    });
    for (int i = 1; i <= n - k + 1; i++) {
        int cur = 0;
        for (int j = m; j >= 1; j--) {
            cur += max(0, min(i + k, nd[j].r+1) - max(i, nd[j].l));
            sum[j] = max(sum[j], cur);
        }
    }
    int res = sum[0];
    for (int i = 1; i <= n - k + 1; i++) {
        int cur = 0;
        for (int j = 1; j <= m; j++) {
            cur += max(0, min(i + k, nd[j].r+1) - max(i, nd[j].l));
            res = max(res, cur + sum[j + 1]);
        }
```

```
    }
    printf("%d\n", res);
}
```

12.13 图上倍增 + 后缀数组（2017ICPC 沈阳）

给定一张图有 N 个点，编号为 $0,...,N-1$，每个点都从自身有一条到点 i^2+1 的边（有向），点上有点权，点权为 $0,...,9$，走过一条路径最终获得的字符串为点权字符串拼起来。问走 N 步，能得到的最大字符串为多少。

```
/*  [input]     [output]
    4
    3           Case #1: 999
    149
    5           Case #2: 53123
    12345
    7           Case #3: 7166666
    3214567
    9           Case #4: 615015015
    261025520
*/
int to[MAXN][22];
int sa[MAXN], rk[MAXN << 1], oldrk[MAXN << 1], id[MAXN], cnt[MAXN];
void run(int s[], int n) {
    int m = 10;
    for (int i = 0; i <= m; i++) cnt[i] = 0;
    for (int i = 1; i <= n; i++) ++cnt[rk[i] = s[i]];
    for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i >= 1; i--) sa[cnt[rk[i]]--] = i;
    int n2 = (n << 1);
    for (int i = n + 1; i <= n2; i++) oldrk[i] = rk[i] = 0;
    for (int w = 0; (1 << w) <= n; w++) {
        for (int i = 0; i <= m; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) id[i] = sa[i];
        for (int i = 1; i <= n; i++) ++cnt[rk[to[id[i]][w]]];
        for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i >= 1; i--) sa[cnt[rk[to[id[i]][w]]]--] = id[i];
        for (int i = 0; i <= m; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) id[i] = sa[i];
        for (int i = 1; i <= n; i++) ++cnt[rk[id[i]]];
        for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
        for (int i = 0; i <= n; i++) oldrk[i] = rk[i];
        int p = 0;
        for (int i = 1; i <= n; i++) {
            if (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[to[sa[i]][w]] == oldrk[to[sa[i -
                1]][w]]) rk[sa[i]] = p;
            else rk[sa[i]] = ++p;
        }
        m = p;
        if (p >= n) break;
    }
}

char str[MAXN];
int a[MAXN];
```

```
int main() {
    int T, kass = 1;
    FI(T);//scanf("%d", &T);
    while (T--) {
        int n;
        FI(n);// scanf("%d", &n);
        FI(str + 1);//scanf("%s", str + 1);
        for (int i = 1; i <= n; i++) {
            a[i] = str[i] - '0' + 1; // 1,2,..10
            to[i][0] = ((ll) (i - 1) * (i - 1) + 1) % n + 1;
        }

        for (int i = 1; i <= 20; i++) {
            for (int j = 1; j <= n; j++)
                to[j][i] = to[to[j][i - 1]][i - 1];
        }
        run(a, n);

        FO('C'), FO('a'), FO('s'), FO('e'), FO(' '), FO('#'), FO(kass++), FO(':'), FO(
                ' ');// printf("Case #%d: ", kass++);
        int tmp = sa[n];
        for (int i = 1; i <= n; i++) {
            FO(a[tmp] - 1);//printf("%d", a[tmp] - 1);
            tmp = to[tmp][0];
        }
        FO('\n');// printf("\n");
    }
    Flush;
}
```

12.14 寻找一个半字串 _ 式子转换 + 马拉车 + 主席树（2017CCPC 哈尔滨 A）

1234543212345 是按照 5 和 1 对称的，那么在保证 5 和 1 是回文串的状态下，要能够之间相互覆盖。

设 5 的位置为 i，1 的位置为 j, 要求 $i \le j$，则可得公式

$$\begin{cases} j - i \le len[i] \\ j - i \le len[j] \end{cases}$$

移动式子，可得

$$\begin{cases} j \le i + len[i] \\ j - len[j] \le i \end{cases}$$

将其转化为区间 $[i+1, i+len[i]]$ 内求有多少个数，使得 $j - len[j] \le i$。

```
/*  [input] ababcbabccbaabc
    [output] 2  */
class HJT { public:
    int ch[MAXN * 70][2], sum[MAXN * 70];
    int tot;
    void init() { tot = 0; }
    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }
    int update(int rt, int pos, int val, int be, int en) {
        int nrt = ++tot;
        ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        if (be == en) {
            sum[nrt] = sum[rt] + val;
            return nrt;
```

```
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
    }
    int query(int lrt, int rrt, int R, int be, int en) {
        if (R == en) return sum[rrt] - sum[lrt];
        int mid = (be + en) >> 1;
        if (R <= mid) {
            return query(ch[lrt][0], ch[rrt][0], R, be, mid);
        } else {
            return sum[ch[rrt][0]] - sum[ch[lrt][0]] + query(ch[lrt][1], ch[rrt][1], R, mid +
                1, en);
        }
    }
} tree;

void Manacher(char s[], int len, char A[], int B[]);
char A[MAXN * 2];
int B[MAXN * 2];

char str[MAXN];
int ren[MAXN], root[MAXN];

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        scanf("%s", str + 1);
        int n = strlen(str + 1);
        Manacher(str + 1, n, A + 1, B + 1);

        int maxR = 0;
        for (int i = 1; i <= n; i++) ren[i] = (B[i * 2 + 1] - 2) >> 1, maxR = max(maxR, i -
            ren[i]);

        tree.init();
        root[0] = 0;
        for (int i = 1; i <= n; i++) {
            root[i] = tree.update(root[i - 1], i - ren[i], 1, 1, maxR);
        }

        ll res = 0;
        for (int i = 1; i <= n; i++) {
            res = res + tree.query(root[i], root[i+ren[i]], i, 1, maxR);
        }
        printf("%lld\n", res);
    }
}
```

12.15 区间子集和的 MEX

给定一个序列，有以下几种操作：

1. 将第 x 位置的数修改为 y。

2. 询问区间 $[L,R]$，问其子集之和的 MEX。

首先有一个性质：先看看是否存在 1。若没有 1，则表明必然无法凑出 1，那么答案就是 1。

然后假设上一次求出来的和的值为 x，表明 $[1,x]$ 都可以通过子集表示出来。再求 $[1,x+1]$ 的和，若和 $[1,x]$ 求和的值相等，则表示 $x+1$ 不能被表示。

最坏情况为斐波那契数列。

```
class DS {  public:
    // HJT begin
    int ch[MAXN * 150][2], tot = 0;
    ll sum[MAXN * 150];

    inline void push_up(int rt) {
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
    }

    int update(int rt, int pos, int val, int be, int en) {
        int nrt = rt;
        if (!rt) {
            nrt = ++tot;
            ch[nrt][0] = ch[nrt][1] = sum[nrt] = 0;
        }
        if (be == en) {
            sum[nrt] += (ll) be * val;
            return nrt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            ch[nrt][0] = update(ch[rt][0], pos, val, be, mid);
            ch[nrt][1] = ch[rt][1];
        } else {
            ch[nrt][0] = ch[rt][0];
            ch[nrt][1] = update(ch[rt][1], pos, val, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
    }

    // HJT end
    int n, c_len, root[MAXN];

    void init(int _n, int _c_len) {
        c_len = _c_len, n = _n;
        for (int i = 1; i <= c_len; i++) root[i] = i;
        tot = c_len;
    }

    inline int lowbit(int x) { return x & (-x); }
    void insert(int pos, int pos_val, int val) {
        for (int i = pos; i <= n; i += lowbit(i)) root[i] = update(root[i], pos_val, val, 1,
            c_len);
    }
    int t1[MAXN], t2[MAXN], n1, n2;
```

```
    inline ll SUM(int R, int be, int en) {
        if (be == en) {
            ll ans = 0;
            for (int i = 1; i <= n1; i++) ans -= sum[t1[i]];
            for (int i = 1; i <= n2; i++) ans += sum[t2[i]];
            return ans;
        }
        int mid = (be + en) >> 1;
        ll ans = 0;
        if (mid >= R) {
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][0];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][0];
            return SUM(R, be, mid);
        } else {
            for (int i = 1; i <= n1; i++) ans -= sum[ch[t1[i]][0]];
            for (int i = 1; i <= n2; i++) ans += sum[ch[t2[i]][0]];
            for (int i = 1; i <= n1; i++) t1[i] = ch[t1[i]][1];
            for (int i = 1; i <= n2; i++) t2[i] = ch[t2[i]][1];
            return ans + SUM(R, mid + 1, en);
        }
    }
    ll query(int l, int r, int x) {
        n1 = n2 = 0;
        for (int i = l - 1; i >= 1; i -= lowbit(i)) t1[++n1] = root[i];
        for (int i = r; i >= 1; i -= lowbit(i)) t2[++n2] = root[i];
        return SUM(x, 1, c_len);
    }
} tree;

int a[MAXN];
int main() {
    int n, q; scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    tree.init(n, 2e5);
    for (int i = 1; i <= n; i++) tree.insert(i, a[i], 1);
    while (q--) {
        int opt; scanf("%d", &opt);
        if (opt == 1) {
            int x, y; scanf("%d%d", &x, &y);
            tree.insert(x, a[x], -1);
            a[x] = y;
            tree.insert(x, a[x], 1);
        } else {
            int L, R; scanf("%d%d", &L, &R);
            ll now = 1, sum = 0;
            while (1) {
                int t = min(now, (ll) 2e5);
                ll tmp = tree.query(L, R, t);
                if (tmp == sum) {
                    printf("%lld\n", now);
                    break;
                } else {
                    sum = tmp, now = sum + 1;
                }
            }
        }
```

```
    }
}
```

12.16 询问中位数区间（[国家集训队]middle）

一个长度为 n 的序列 a，设其排过序之后为 b，其中位数定义为 $b_{n/2}$，其中 a,b 从 0 开始标号, 除法取下整。

给你一个长度为 n 的序列 s。

回答 Q 个这样的询问：s 的左端点在 $[a,b]$ 之间，右端点在 $[c,d]$ 之间的子区间中，最大的中位数。

其中 $a < b < c < d$。

位置也从 0 开始标号。

我会使用一些方式强制你在线。

```
class HJT { public:
    struct node {
        int ch[2];
        int pre, suf, sum;
        node() {
            ch[0] = ch[1] = pre = suf = sum = 0;
        }
    } T[MAXN * 70];
    int tot;
#define lson T[rt].ch[0]
#define rson T[rt].ch[1]
    inline void push_up(int rt) {
        T[rt].sum = T[lson].sum + T[rson].sum;
        T[rt].pre = max(T[lson].pre, T[lson].sum + T[rson].pre);
        T[rt].suf = max(T[rson].suf, T[rson].sum + T[lson].suf);
    }
    int build(int l, int r) {
        int nrt = ++tot;
        if (l == r) {
            T[nrt].sum = T[nrt].pre = T[nrt].suf = 1;
            return nrt;
        }
        int mid = (l + r) >> 1;
        T[nrt].ch[0] = build(l, mid), T[nrt].ch[1] = build(mid + 1, r);
        push_up(nrt);
        return nrt;
    }
    int update(int rt, int pos, int v, int be, int en) {
        int nrt = ++tot;
        if (be == en) {
            T[nrt].sum = T[nrt].pre = T[nrt].suf = v;
            return nrt;
        }
        int mid = (be + en) >> 1;
        if (pos <= mid) {
            T[nrt].ch[0] = update(lson, pos, v, be, mid);
            T[nrt].ch[1] = rson;
        } else {
            T[nrt].ch[0] = lson;
            T[nrt].ch[1] = update(rson, pos, v, mid + 1, en);
        }
        push_up(nrt);
        return nrt;
```

```
    }
    int query_sum(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return T[rt].sum;
        int mid = (be + en) >> 1;
        int ans = 0;
        if (L <= mid) ans += query_sum(lson, L, R, be, mid);
        if (R > mid) ans += query_sum(rson, L, R, mid + 1, en);
        return ans;
    }
    node query_pre(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return T[rt];
        int mid = (be + en) >> 1;
        if (R <= mid) return query_pre(lson, L, R, be, mid);
        else if (L > mid) return query_pre(rson, L, R, mid + 1, en);
        else {
            node ans;
            node ansl = query_pre(lson, L, R, be, mid);
            node ansr = query_pre(rson, L, R, mid + 1, en);
            ans.sum = ansl.sum + ansr.sum;
            ans.pre = max(ansl.pre, ansl.sum + ansr.pre);
            return ans;

        }
    }
    node query_suf(int rt, int L, int R, int be, int en) {
        if (L <= be && en <= R) return T[rt];
        int mid = (be + en) >> 1;
        if (R <= mid) return query_suf(lson, L, R, be, mid);
        else if (L > mid) return query_suf(rson, L, R, mid + 1, en);
        else {
            node ans;
            node ansl = query_suf(lson, L, R, be, mid);
            node ansr = query_suf(rson, L, R, mid + 1, en);
            ans.sum = ansl.sum + ansr.sum;
            ans.suf = max(ansr.suf, ansr.sum + ansl.suf);
            return ans;
        }
    }
} tree;

int a[MAXN], root[MAXN];
vector<int> vec[MAXN];

int n;
bool check(int mid, const vector<int> &q) {
    int sum = 0;
    if (q[1] + 1 <= q[2] - 1) sum = tree.query_sum(root[mid - 1], q[1] + 1, q[2] - 1, 1, n);
    sum += tree.query_suf(root[mid - 1], q[0], q[1], 1, n).suf;
    sum += tree.query_pre(root[mid - 1], q[2], q[3], 1, n).pre;
    if (sum >= 0) return 1;
    else return 0;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
```

```
        scanf("%d", &a[i]);
        Discrete::insert(a[i]);
    }
    Discrete::init();
    for (int i = 1; i <= n; i++) a[i] = val2id(a[i]);
    for (int i = 1; i <= n; i++) {
        vec[a[i]].pb(i);
    }
    root[0] = tree.build(1, n);
    for (int i = 1; i <= Discrete::blen; i++) {
        root[i] = root[i - 1];
        for (auto e: vec[i]) {
            root[i] = tree.update(root[i], e, -1, 1, n);
        }
    }
    int m; scanf("%d", &m);
    int lastans = 0;
    while (m--) {
        vector<int> q(4);
        scanf("%d%d%d%d", &q[0], &q[1], &q[2], &q[3]);
        for (int i = 0; i < 4; i++) q[i] = (q[i] + lastans) % n + 1;
        sort(q.begin(), q.end());
        int L = 1, R = Discrete::blen;
        while (L < R) {
            int mid = (L + R + 1) >> 1;
            if (check(mid, q)) L = mid;
            else R = mid - 1;
        }
        lastans = id2val(L);
        printf("%d\n", lastans);
    }
}
```

12.17 双线段树根据最终状态判断

n 只猴子坐在椅子上，有 m 次区间更新（笑话），没听过这个笑话的猴子会掉到地上，原来听过这个笑话的猴子会做到椅子上。问你最后有几只猴子坐在椅子上

猴子最后的状态其实只与他最后听到的那个笑话有关

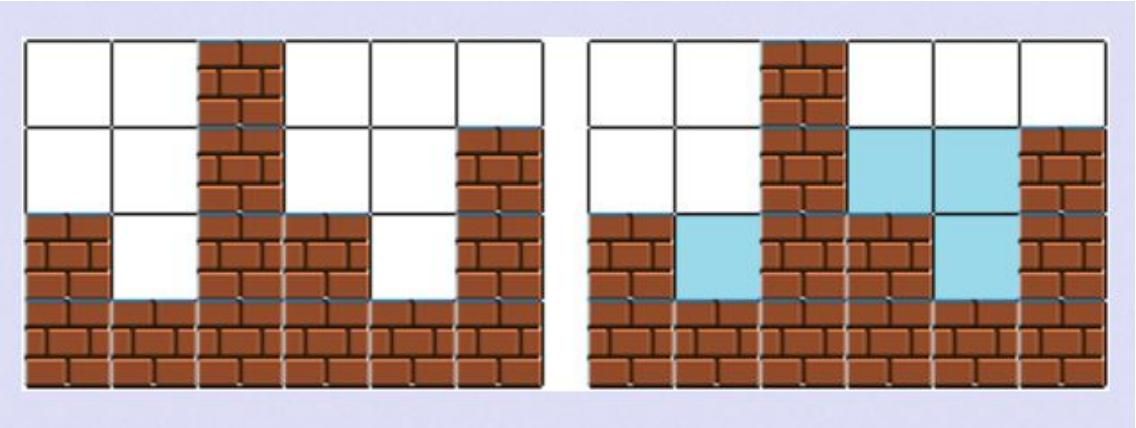
```
/*  [input]     [output]
    1           3
    10 7
    3 11 0
    3 11 2
    5 12 1
    8 13 2
    7 11 2
    10 12 1
    9 12 0
*/
class SEG { public:
    struct node {
        int l, r, val;
    } T[MAXN << 2];
    int lazy[MAXN << 2];
```

```
    inline void push_down(int rt) {
        if (lazy[rt]) {
            lazy[rt << 1] = lazy[rt], lazy[rt << 1 | 1] = lazy[rt];
            T[rt << 1].val = lazy[rt], T[rt << 1 | 1].val = lazy[rt];
            lazy[rt] = 0;
        }
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r, T[rt].val = 0;
        lazy[rt] = 0;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid + 1, r);
    }
    void update(int L, int R, int v, int rt) {
        if (L <= T[rt].l && R >= T[rt].r) {
            T[rt].val = v;
            lazy[rt] = v;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        push_down(rt);
        if (L <= mid) update(L, R, v, rt << 1);
        if (R > mid) update(L, R, v, rt << 1 | 1);
    }
    int query(int rt, int pos, int l, int r) {
        if (l == r) return T[rt].val;
        int mid = (l + r) >> 1;
        push_down(rt);
        if (pos <= mid) return query(rt << 1, pos, l, mid);
        else return query(rt << 1 | 1, pos, mid + 1, r);
    }
} tree;
class SEG2 { public:
    struct node {
        int l, r, val;
    } T[MAXN << 2];
    int lazy[MAXN << 2];
    inline void push_down(int rt) {
        if (lazy[rt]) {
            lazy[rt << 1] += lazy[rt], lazy[rt << 1 | 1] += lazy[rt];
            T[rt << 1].val += lazy[rt], T[rt << 1 | 1].val += lazy[rt];
            lazy[rt] = 0;
        }
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r, T[rt].val = 0;
        lazy[rt] = 0;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid + 1, r);
    }
    void update(int L, int R, int v, int rt) {
        if (L <= T[rt].l && R >= T[rt].r) {
```

```
            T[rt].val += v;
            lazy[rt] += v;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        push_down(rt);
        if (L <= mid) update(L, R, v, rt << 1);
        if (R > mid) update(L, R, v, rt << 1 | 1);
    }
    int query(int rt, int pos, int l, int r) {
        if (l == r) return T[rt].val;
        int mid = (l + r) >> 1;
        push_down(rt);
        if (pos <= mid) return query(rt << 1, pos, l, mid);
        else return query(rt << 1 | 1, pos, mid + 1, r);
    }
} tree2;
typedef pair<int, int> pii;
vector<pii> col[MAXN];
vector<int> vec[MAXN];
int main() {
    int T;mscanf("%d", &T);
    while (T--) {
        int n, q;mscanf("%d%d", &n, &q);
        for (int i = 1; i < MAXN; i++) col[i].clear(), vec[i].clear();
        tree.build(1, 1, n);
        int maxL = -1;
        while (q--) {
            int x, l, k;
            scanf("%d%d%d", &x, &l, &k);
            int L = x - k, R = x + k;
            tree.update(L, R, l, 1);
            maxL = max(maxL, l);
            col[l].push_back(make_pair(x - k, x + k));
        }
        int res = 0;
        for (int i = 1; i <= n; i++) {
            int color = tree.query(1, i, 1, n);
            if (color == 0) res++;
            else vec[color].push_back(i);
        }
        tree2.build(1, 1,n);
        for (int i = 1; i <= maxL; i++) {
            if (SZ(vec[i])) {
                for (auto e: col[i]) {
                    int L = e.first, R = e.second;
                    tree2.update(L, R, 1, 1);
                }
                for (auto e: vec[i]) {
                    if (tree2.query(1, e, 1, n) > 1) res++;
                }
                for (auto e: col[i]) {
                    int L = e.first, R = e.second;
                    tree2.update(L, R, -1, 1);
                }
            }
```

```
        }
        printf("%d\n", res);
    }
}
```

12.18 围墙算容积

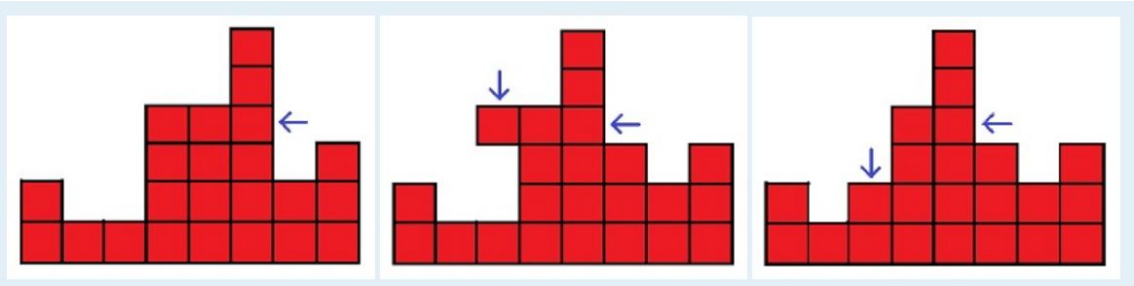


```
/*  [input]     [output]
    1
    6 3
    2 1 4 2 1 3
    P           4
    U 1 2
    P           6
*/
ll a[MAXN];
class SEG { public:
    struct node {
        int l, r;
        ll maxx, sum, lx, rx;
    } T[MAXN << 2];
    inline void push_up(int rt) {
        T[rt].sum = T[rt << 1].sum + T[rt << 1 | 1].sum;
        T[rt].maxx = max(T[rt << 1].maxx, T[rt << 1 | 1].maxx);
    }
    inline void push_down(int rt) {
        T[rt << 1].lx = max(T[rt].lx, T[rt << 1].lx);
        T[rt << 1 | 1].lx = max(T[rt].lx, T[rt << 1 | 1].lx);
        T[rt << 1].rx = max(T[rt].rx, T[rt << 1].rx);
        T[rt << 1 | 1].rx = max(T[rt].rx, T[rt << 1 | 1].rx);
    }
    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        T[rt].lx = T[rt].rx = 0;
        if (l == r) {
            T[rt].maxx = T[rt].sum = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
```

```
    void change(int rt, int pos, ll v) {
        if (T[rt].l == T[rt].r) {
            T[rt].maxx += v;
            T[rt].sum += v;
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) change(rt << 1, pos, v);
        else change(rt << 1 | 1, pos, v);
        push_up(rt);
    }
    void change1(int rt, int L, int R, ll v) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].lx = max(T[rt].lx, v);
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) change1(rt << 1, L, R, v);
        if (R > mid) change1(rt << 1 | 1, L, R, v);
    }
    void change2(int rt, int L, int R, ll v) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].rx = max(T[rt].rx, v);
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) change2(rt << 1, L, R, v);
        if (R > mid) change2(rt << 1 | 1, L, R, v);
    }
    ll query(int rt) {
        if (T[rt].maxx <= min(T[rt].lx, T[rt].rx)) {
            return min(T[rt].lx, T[rt].rx) * (T[rt].r - T[rt].l + 1) - T[rt].sum;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        return query(rt << 1) + query(rt << 1 | 1);
    }
} tree;

int main() {
    int T; FI(T);
    while (T--) {
        int n, q; FI(n), FI(q);
        for (int i = 1; i <= n; i++) FI(a[i]);
        tree.build(1, 1, n);
        for (int i = 1; i <= n; i++) {
            tree.change1(1, i, n, a[i]);
            tree.change2(1, 1, i, a[i]);
        }
        while (q--) {
            char opt[2]; FI(opt);
            if (opt[0] == 'P') {
                FO(tree.query(1)); FO('\n');
```

```
            } else {
                int pos; ll v; FI(pos), FI(v); // add v in pos
                tree.change(1, pos, v);
                a[pos] += v;
                tree.change1(1, pos, n, a[pos]);
                tree.change2(1, 1, pos, a[pos]);
            }
        }
    }
    Flush;
}
```

12.19 推格子（HDU 多校 FHQ）



```
/*  [input]     [output]
    1
    8 4
    2 1 1 4 4 6 2 3
    1 6 4       5
    2 5         6
    1 1 1       0
    1 8 2       14
                2 2 4 6 3 2 3 1
*/
int b[MAXN];
class FHQ { public:
    int ch[MAXN][2];
    int val[MAXN], minn[MAXN], siz[MAXN], dat[MAXN];
    ll sum[MAXN];
    int tot, root;
    int seq[MAXN], seq_tot;
    void init() {
        root = 1; tot = 0;
        val[0] = sum[0] = siz[0] = 0;
        minn[0] = inf_int;
        seq_tot = 0;
    }
    void dfs(int rt) {
        if (ch[rt][0]) dfs(ch[rt][0]);
        seq[seq_tot++] = val[rt];
        if (ch[rt][1]) dfs(ch[rt][1]);
    }
    void get_seq() {
        dfs(root);
        for (int i = 0; i < seq_tot; i++) {
            if (i) printf(" ");
```

```
            printf("%d", seq[i]);
        }
        printf("\n");
    }
    int New(int v) {
        val[++tot] = v, dat[tot] = rand(), minn[tot] = v, siz[tot] = 1;
        sum[tot] = (ll) v;
        ch[tot][0] = ch[tot][1] = 0;
        return tot;
    }
    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
        minn[rt] = min(val[rt], min(minn[ch[rt][0]], minn[ch[rt][1]]));
        sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]] + (ll) val[rt];
    }
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        int newnode = New(b[mid]);
        ch[newnode][0] = build(l, mid - 1);
        ch[newnode][1] = build(mid + 1, r);
        push_up(newnode);
        return newnode;
    }
    void split_id(int rt, int k, int &x, int &y) {
        if (!rt) x = y = 0;
        else {
            if (k <= siz[ch[rt][0]]) {
                y = rt;
                split_id(ch[rt][0], k, x, ch[rt][0]);
            } else {
                x = rt;
                split_id(ch[rt][1], k - siz[ch[rt][0]] - 1, ch[rt][1], y);
            }
            push_up(rt);
        }
    }
    void split_minn(int rt, int k, int &x, int &y) {
        if (!rt) x = y = 0;
        else {
            if (minn[ch[rt][1]] < k || val[rt] < k) {
                x = rt;
                split_minn(ch[rt][1], k, ch[rt][1], y);
            } else {
                y = rt;
                split_minn(ch[rt][0], k, x, ch[rt][0]);
            }
            push_up(rt);
        }
    }
    int get_Kth(int rt, int k) {
        if (siz[ch[rt][0]] + 1 == k) return val[rt];
        else if (siz[ch[rt][0]] >= k) return get_Kth(ch[rt][0], k);
        else return get_Kth(ch[rt][1], k - siz[ch[rt][0]] - 1);
    }
    int merge(int x, int y) {
```

```
        if (!x || !y) return x + y;
        if (dat[x] < dat[y]) {
            ch[x][1] = merge(ch[x][1], y);
            push_up(x);
            return x;
        } else {
            ch[y][0] = merge(x, ch[y][0]);
            push_up(y);
            return y;
        }
    }
    void op1(int x, int y) {
        int h = get_Kth(root, x);
        if (h < y) {
            printf("0\n"); return ;
        }
        int left_, _right;
        split_id(root, x, left_, _right);
        int minn_, _minn;
        split_minn(left_, y, minn_, _minn);
        if (minn_ == 0) {
            root = merge(merge(minn_, _minn), _right);
            printf("0\n"); return ;
        }
        int pos_3 = get_Kth(minn_, siz[minn_]);
        int pos_4 = get_Kth(_minn, 1);
        int new_v = pos_4 - (y-1) + pos_3;
        int yx, shk, hjy;
        int yx_, _yx;
        split_id(minn_, siz[minn_]-1, yx_, _yx);
        int new_3 = New(new_v);
        yx = merge(yx_, new_3);
        printf("%lld\n", sum[_minn] - (ll)siz[_minn] * (ll)(y-1));  // 打印有多少格子有动
        int shk_, _shk;
        split_id(_minn, 1, shk_, _shk);
        int new_6 = New(y-1);
        shk = merge(_shk, new_6);
        hjy = _right;
        root = merge(yx, merge(shk, hjy));
    }
    void op2(int x) {
        printf("%d\n", get_Kth(root, x));
    }
} tree;

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        tree.init();
        int n, q; scanf("%d%d", &n, &q);
        for (int i = 1; i <= n; i++) scanf("%d", &b[i]);
        tree.root = tree.build(1, n);
        while (q--) {
            int op; scanf("%d", &op);
            if (op == 1) {
                int x, y; scanf("%d%d", &x, &y);
```

```
                tree.op1(x, y);
            } else {
                int x; scanf("%d", &x); // 查询第x列
                tree.op2(x);
            }
        }
        tree.get_seq();
    }
}
```

13 他人计算几何

13.1 st1vdy

```
namespace geometry {
#define db long double
#define pi acos(-1.0)
    constexpr db eps = 1e-7;
    int sign(db k) {
        if (k > eps) return 1;
        else if (k < -eps) return -1;
        return 0;
    }
    int cmp(db k1, db k2) { // k1 < k2 : -1, k1 == k2 : 0, k1 > k2 : 1
        return sign(k1 - k2);
    }
    int inmid(db k1, db k2, db k3) { // k3 在 [k1, k2] 内
        return sign(k1 - k3) * sign(k2 - k3) <= 0;
    }

    struct point { // 点类
        db x, y;
        point() {}
        point(db x_, db y_) :x(x_), y(y_) {}
        point operator + (const point& k) const { return point(k.x + x, k.y + y); }
        point operator - (const point& k) const { return point(x - k.x, y - k.y); }
        point operator * (db k) const { return point(x * k, y * k); }
        point operator / (db k1) const { return point(x / k1, y / k1); }
        point turn(db k1) { return point(x * cos(k1) - y * sin(k1), x * sin(k1) + y *
            cos(k1)); } // 逆时针旋转
        point turn90() { return point(-y, x); } // 逆时针方向旋转 90 度
        db len() { return sqrt(x * x + y * y); } // 向量长度
        db len2() { return x * x + y * y; } // 向量长度的平方
        db getPolarAngle() { return atan2(y, x); } // 向量极角
        db dis(point k) { return ((*this) - k).len(); } // 到点k的距离
        point unit() { db d = len(); return point(x / d, y / d); } // 单位向量
        point getdel() { // 将向量的方向调整为指向第一/四象限 包括y轴正方向
            if (sign(x) == -1 || (sign(x) == 0 && sign(y) == -1))
                return (*this) * (-1);
            else return (*this);
        }
        bool operator < (const point& k) const { // 水平序排序
            x坐标为第一关键字,y坐标第二关键字
```

```
38              return x == k.x ? y < k.y : x < k.x;
39          }
40          bool getP() const { // 判断点是否在上半平面 含x负半轴 不含x正半轴及零点
41              return sign(y) == 1 || (sign(y) == 0 && sign(x) == -1);
42          }
43      };
44      db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; } // 向量 k1,k2 的叉积
45      db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; }   // 向量 k1,k2 的点积
46      db rad(point k1, point k2) { // 向量 k1,k2 之间的有向夹角
47          return atan2(cross(k1, k2), dot(k1, k2));
48      }
49      int inmid(point k1, point k2, point k3) { // k1 k2 k3共线时 判断点 k3 是否在线段 k1k2 上
50          return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y);
51      }
52      int compareAngle(point k1, point k2) { // 比较向量 k1,k2 的角度大小 角度按照atan2()函数定义
53          // k1 < k2 返回 1, k1 >= k2 返回 0
54          return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sign(cross(k1, k2)) > 0);
55      }
56      point proj(point k1, point k2, point q) { // q 到直线 k1,k2 的投影
57          point k = k2 - k1; return k1 + k * (dot(q - k1, k) / k.len2());
58      }
59      point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; } // q
            关于直线 k1,k2 的对称点
60      int counterclockwise(point k1, point k2, point k3) { // k1 k2 k3 逆时针1 顺时针-1 否则0
61          return sign(cross(k2 - k1, k3 - k1));
62      }
63      int checkLL(point k1, point k2, point k3, point k4) { // 判断直线 k1k2 和直线k3k4 是否相交
64          // 即判断直线 k1k2 和 k3k4 是否平行 平行返回0 不平行返回1
65          return sign(cross(k2 - k1, k4 - k3)) != 0;
66      }
67      point getLL(point k1, point k2, point k3, point k4) { // 求 k1k2 k3k4 两直线交点
68          db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);
69          return (k1 * w2 + k2 * w1) / (w1 + w2);
70      }
71      int intersect(db l1, db r1, db l2, db r2) { // 判断 [l1,r1] 和 [l2, r2] 是否相交
72          if (l1 > r1) swap(l1, r1);
73          if (l2 > r2) swap(l2, r2);
74          return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
75      }
76      int checkSS(point k1, point k2, point k3, point k4) { // 判断线段 k1k2 和线段 k3k4 是否相交
77          return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y) &&
78              sign(cross(k3 - k1, k4 - k1)) * sign(cross(k3 - k2, k4 - k2)) <= 0 &&
79              sign(cross(k1 - k3, k2 - k3)) * sign(cross(k1 - k4, k2 - k4)) <= 0;
80      }
81      db disSP(point k1, point k2, point q) { // 点 q 到线段 k1k2 的最短距离
82          point k3 = proj(k1, k2, q);
83          if (inmid(k1, k2, k3)) return q.dis(k3);
84          else return min(q.dis(k1), q.dis(k2));
85      }
86      db disLP(point k1, point k2, point q) { // 点 q 到直线 k1k2 的最短距离
87          point k3 = proj(k1, k2, q);
88          return q.dis(k3);
89      }
90      db disSS(point k1, point k2, point k3, point k4) { // 线段 k1k2 和线段 k3k4 的最短距离
91          if (checkSS(k1, k2, k3, k4)) return 0;
92          else return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)),
```

```
93              min(disSP(k3, k4, k1), disSP(k3, k4, k2)));
94      }
95      bool onLine(point k1, point k2, point q) { // 判断点 q 是否在直线 k1k2 上
96          return sign(cross(k1 - q, k2 - q)) == 0;
97      }
98      bool onSegment(point k1, point k2, point q) { // 判断点 q 是否在线段 k1k2 上
99          if (!onLine(k1, k2, q)) return false;
100         return inmid(k1, k2, q);
101     }
102     void polarAngleSort(vector<point>& p, point t) { // p为待排序点集 t为极角排序中心
103         sort(p.begin(), p.end(), [&](const point& k1, const point& k2) {
104             return compareAngle(k1 - t, k2 - t);
105             });
106     }
107
108     struct line { // 直线 / 线段类
109         point p[2];
110         line() {}
111         line(point k1, point k2) { p[0] = k1, p[1] = k2; }
112         point& operator [] (int k) { return p[k]; }
113         point dir() { return p[1] - p[0]; } // 向量 p[0] -> p[1]
114         bool include(point k) { // 判断点是否在直线上
115             return sign(cross(p[1] - p[0], k - p[0])) > 0;
116         }
117         bool includeS(point k) { // 判断点是否在线段上
118             return onSegment(p[0], p[1], k);
119         }
120         line push(db len) { // 向外（左手边）平移 len 个单位
121             point delta = (p[1] - p[0]).turn90().unit() * len;
122             return line(p[0] - delta, p[1] - delta);
123         }
124     };
125
126     bool parallel(line k1, line k2) { // 判断是否平行
127         return sign(cross(k1.dir(), k2.dir())) == 0;
128     }
129     bool sameLine(line k1, line k2) { // 判断是否共线
130         return parallel(k1, k2) && parallel(k1, line(k2.p[0], k1.p[0]));
131     }
132     bool sameDir(line k1, line k2) { // 判断向量 k1 k2 是否同向
133         return parallel(k1, k2) && sign(dot(k1.dir(), k2.dir())) == 1;
134     }
135     bool operator < (line k1, line k2) {
136         if (sameDir(k1, k2)) return k2.include(k1[0]);
137         return compareAngle(k1.dir(), k2.dir());
138     }
139     point getLL(line k1, line k2) {  // 求 k1 k2 两直线交点 不要忘了判平行!
140         return getLL(k1[0], k1[1], k2[0], k2[1]);
141     }
142     bool checkpos(line k1, line k2, line k3) {  // 判断是否三线共点
143         return k3.include(getLL(k1, k2));
144     }
145
146     struct circle { // 圆类
147         point o;
148         double r;
```

```
149         circle() {}
150         circle(point o_, double r_) : o(o_), r(r_) {}
151         int inside(point k) {  // 判断点 k 和圆的位置关系
152             return cmp(r, o.dis(k)); // 圆外:-1, 圆上:0, 圆内:1
153         }
154     };
155
156     int checkposCC(circle k1, circle k2) { // 返回两个圆的公切线数量
157         if (cmp(k1.r, k2.r) == -1) swap(k1, k2);
158         db dis = k1.o.dis(k2.o);
159         int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);
160         if (w1 > 0) return 4; // 外离
161         else if (w1 == 0) return 3; // 外切
162         else if (w2 > 0) return 2;  // 相交
163         else if (w2 == 0) return 1; // 内切
164         else return 0; // 内离(包含)
165     }
166     vector<point> getCL(circle k1, point k2, point k3) { // 求直线 k2k3 和圆 k1 的交点
167         // 沿着 k2->k3 方向给出 相切给出两个
168         point k = proj(k2, k3, k1.o);
169         db d = k1.r * k1.r - (k - k1.o).len2();
170         if (sign(d) == -1) return {};
171         point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
172         return { k - del,k + del };
173     }
174     vector<point> getCC(circle k1, circle k2) { // 求圆 k1 和圆 k2 的交点
175         // 沿圆 k1 逆时针给出, 相切给出两个
176         int pd = checkposCC(k1, k2); if (pd == 0 || pd == 4) return {};
177         db a = (k2.o - k1.o).len2(), cosA = (k1.r * k1.r + a -
178             k2.r * k2.r) / (2 * k1.r * sqrt(max(a, (db)0.0)));
179         db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
180         point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
181         return { m - del,m + del };
182     }
183     vector<point> tangentCP(circle k1, point k2) { // 点 k2 到圆 k1 的切点 沿圆 k1 逆时针给出
184         db a = (k2 - k1.o).len(), b = k1.r * k1.r / a, c = sqrt(max((db)0.0, k1.r * k1.r - b *
                b));
185         point k = (k2 - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
186         return { m - del,m + del };
187     }
188     vector<line> tangentOutCC(circle k1, circle k2) {
189         int pd = checkposCC(k1, k2);
190         if (pd == 0) return {};
191         if (pd == 1) {
192             point k = getCC(k1, k2)[0];
193             return { line(k,k) };
194         }
195         if (cmp(k1.r, k2.r) == 0) {
196             point del = (k2.o - k1.o).unit().turn90().getdel();
197             return { line(k1.o - del * k1.r,k2.o - del * k2.r),
198                 line(k1.o + del * k1.r,k2.o + del * k2.r) };
199         }
200         else {
201             point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r - k2.r);
202             vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
203             vector<line> ans; for (int i = 0; i < A.size(); i++)
```

```
            ans.push_back(line(A[i], B[i]));
        return ans;
    }
}
vector<line> tangentInCC(circle k1, circle k2) {
    int pd = checkposCC(k1, k2);
    if (pd <= 2) return {};
    if (pd == 3) {
        point k = getCC(k1, k2)[0];
        return { line(k, k) };
    }
    point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
    vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
    vector<line> ans;
    for (int i = 0; i < (int)A.size(); i++) ans.push_back(line(A[i], B[i]));
    return ans;
}
vector<line> tangentCC(circle k1, circle k2) { // 求两圆公切线
    int flag = 0;
    if (k1.r < k2.r) swap(k1, k2), flag = 1;
    vector<line> A = tangentOutCC(k1, k2), B = tangentInCC(k1, k2);
    for (line k : B) A.push_back(k);
    if (flag) for (line& k : A) swap(k[0], k[1]);
    return A;
}
db getAreaUnionCT(circle k1, point k2, point k3) { // 圆 k1 与三角形 k2k3k1.o 的有向面积交
    point k = k1.o; k1.o = k1.o - k; k2 = k2 - k; k3 = k3 - k;
    int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
    vector<point> A = getCL(k1, k2, k3);
    if (pd1 >= 0) {
        if (pd2 >= 0) return cross(k2, k3) / 2;
        return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;
    }
    else if (pd2 >= 0) {
        return k1.r * k1.r * rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;
    }
    else {
        int pd = cmp(k1.r, disSP(k2, k3, k1.o));
        if (pd <= 0) return k1.r * k1.r * rad(k2, k3) / 2;
        return cross(A[0], A[1]) / 2 + k1.r * k1.r * (rad(k2, A[0]) + rad(A[1], k3)) / 2;
    }
}
circle getCircle(point k1, point k2, point k3) { // 三点确定一个圆
    db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
    db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
    db d = a1 * b2 - a2 * b1;
    point o = point(k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) / d);
    return circle(o, k1.dis(o));
}
circle minCircleCovering(vector<point> A) { // 最小圆覆盖 O(n)随机增量法
    random_shuffle(A.begin(), A.end());
    circle ans = circle(A[0], 0);
    for (int i = 1; i < A.size(); i++) {
        if (ans.inside(A[i]) == -1) {
            ans = circle(A[i], 0);
            for (int j = 0; j < i; j++) {
```

```
                if (ans.inside(A[j]) == -1) {
                    ans.o = (A[i] + A[j]) / 2;
                    ans.r = ans.o.dis(A[i]);
                    for (int k = 0; k < j; k++) {
                        if (ans.inside(A[k]) == -1)
                            ans = getCircle(A[i], A[j], A[k]);
                    }
                }
            }
        }
    }
    return ans;
}

struct polygon { // 多边形类
    int n; // 点数
    vector<point> p;
    polygon() {}
    polygon(vector<point> a) {
        n = (int)a.size();
        p = a;
    }
    db area() { // 多边形有向面积
        if (n < 3) return 0;
        db ans = 0;
        for (int i = 1; i < n - 1; i++)
            ans += cross(p[i] - p[0], p[i + 1] - p[0]);
        return 0.5 * ans;
    }
    int inConvexHull(point a) { // O(logn)判断点是否在凸包内 1内部 0边界 -1外部
        // 必须保证凸多边形是一个水平序凸包且不能退化
        // 退化情况 比如凸包退化成线段 可使用 onSegment() 函数特判
        auto check = [&](int x) {
            int ccw1 = counterclockwise(p[0], a, p[x]),
                ccw2 = counterclockwise(p[0], a, p[x + 1]);
            if (ccw1 == -1 && ccw2 == -1) return 1;
            else if (ccw1 == 1 && ccw2 == 1) return -1;
            else if (ccw1 == -1 && ccw2 == 1) return 0;
            else return 0;
        };
        if (counterclockwise(p[0], a, p[1]) <= 0 && counterclockwise(p[0], a, p.back()) >=
            0) {
            int l = 1, r = n - 2, mid;
            while (l <= r) {
                mid = (l + r) >> 1;
                int chk = check(mid);
                if (chk == 1) l = mid + 1;
                else if (chk == -1) r = mid;
                else break;
            }
            int res = counterclockwise(p[mid], a, p[mid + 1]);
            if (res < 0) return 1;
            else if (res == 0) return 0;
            else return -1;
        }
        else return -1;
```

```
        }
    };

    int checkPolyP(polygon poly, point q) { // O(n)判断点是否在一般多边形内
        // 必须保证简单多边形的点按逆时针给出 返回 2 内部, 1 边界, 0 外部
        int pd = 0;
        for (int i = 0; i < poly.n; i++) {
            point u = poly.p[i], v = poly.p[(i + 1) % poly.n];
            if (onSegment(u, v, q)) return 1;
            if (cmp(u.y, v.y) > 0) swap(u, v);
            if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0) continue;
            if (sign(cross(u - v, q - v)) < 0) pd ^= 1;
        }
        return pd << 1;
    }
    bool checkConvexHull(polygon poly) { // 检测多边形是否是凸包
        int sgn = counterclockwise(poly.p[0], poly.p[1], poly.p[2]);
        for (int i = 1; i < poly.n; i++) {
            int ccw = counterclockwise(poly.p[i], poly.p[(i + 1) % poly.n], poly.p[(i + 2) %
                poly.n]);
            if (sgn != ccw) return false;
        }
        return true;
    }
    db convexDiameter(polygon poly) { // 0(n)旋转卡壳求凸包直径 / 平面最远点对的平方
        int n = poly.n; // 请保证多边形是凸包
        db ans = 0;
        for (int i = 0, j = n < 2 ? 0 : 1; i < j; i++) {
            for (;; j = (j + 1) % n) {
                ans = max(ans, (poly.p[i] - poly.p[j]).len2());
                if (sign(cross(poly.p[i + 1] - poly.p[i], poly.p[(j + 1) % n] - poly.p[j])) <=
                    0) break;
            }
        }
        return ans;
    }

    vector<point> convexHull(vector<point> A, int flag = 1) { // 凸包 flag=0 不严格 flag=1 严格
        int n = A.size(); vector<point> ans(n + n);
        sort(A.begin(), A.end()); int now = -1;
        for (int i = 0; i < A.size(); i++) {
            while (now > 0 && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)
                now--;
            ans[++now] = A[i];
        }
        int pre = now;
        for (int i = n - 2; i >= 0; i--) {
            while (now > pre && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) <
                flag)
                now--;
            ans[++now] = A[i];
        }
        ans.resize(now);
        return ans;
    }
    polygon getConvexHull(vector<point> A, int flag = 1) { // 凸包 flag=0 不严格 flag=1
```

```
368         return polygon(convexHull(A, flag));
369     }
370     vector<point> convexCut(vector<point> A, point k1, point k2) { // 半平面 k1k2 切凸包 A
371         int n = A.size(); // 保留所有满足 k1 -> p -> k2 为逆时针方向的点
372         A.push_back(A[0]);
373         vector<point> ans;
374         for (int i = 0; i < n; i++) {
375             int ccw1 = counterclockwise(k1, k2, A[i]);
376             int ccw2 = counterclockwise(k1, k2, A[i + 1]);
377             if (ccw1 >= 0) ans.push_back(A[i]);
378             if (ccw1 * ccw2 <= 0) ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
379         }
380         return ans;
381     }
382 
383     vector<line> getHL(vector<line>& L) { // 求半平面交 逆时针方向存储
384         sort(L.begin(), L.end());
385         deque<line> q;
386         for (int i = 0; i < (int)L.size(); ++i) {
387             if (i && sameDir(L[i], L[i - 1])) continue;
388             while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i]))
                    q.pop_back();
389             while (q.size() > 1 && !checkpos(q[1], q[0], L[i])) q.pop_front();
390             q.push_back(L[i]);
391         }
392         while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
393         while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1])) q.pop_front();
394         vector<line> ans;
395         for (int i = 0; i < q.size(); ++i) ans.push_back(q[i]);
396         return ans;
397     }
398 
399     db closestPoint(vector<point>& A, int l, int r) { // 最近点对，先要按照 x 坐标排序
400         if (r - l <= 5) {
401             db ans = 1e20;
402             for (int i = l; i <= r; ++i)
403                 for (int j = i + 1; j <= r; j++)
404                     ans = min(ans, A[i].dis(A[j]));
405             return ans;
406         }
407         int mid = l + r >> 1;
408         db ans = min(closestPoint(A, l, mid), closestPoint(A, mid + 1, r));
409         vector<point> B;
410         for (int i = l; i <= r; i++)
411             if (abs(A[i].x - A[mid].x) <= ans)
412                 B.push_back(A[i]);
413         sort(B.begin(), B.end(), [&](const point& k1, const point& k2) {
414             return k1.y < k2.y;
415             });
416         for (int i = 0; i < B.size(); i++)
417             for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)
418                 ans = min(ans, B[i].dis(B[j]));
419         return ans;
420     }
421 }
422 using namespace geometry;
```

13.2 forever97

```
const double EPS = 1e-8;
//-----------------------------------------------------------------------
// double cmp
int dcmp(double x) { return fabs(x) < EPS ? 0 : (x < 0 ? -1 : 1); }
struct Point {
    double x, y, z;
    Point() { x = y = z = 0; }
    Point(double x, double y, double z) : x(x), y(y), z(z) {}
    Point operator+(Point a) { return Point(x + a.x, y + a.y, z + a.z); }
    Point operator-(Point a) { return Point(x - a.x, y - a.y, z - a.z); }
    Point operator*(double k) { return Point(x * k, y * k, z * k); }
    Point operator/(double k) { return Point(x / k, y / k, z / k); }
    double operator*(Point a) { return x * a.x + y * a.y + z * a.z; }  // 点积
    Point operator^(Point a) {
        return Point(y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x);
    }  // 叉积
    double length() { return sqrt(x * x + y * y + z * z); }
};
double Angle(Point a, Point b) { return acos(a * b / a.length() / b.length()); }
Point projection(Point v, Point u) {  // 向量v 在u 上投影
    double scalar = (v * u) / (u * u);
    return u * scalar;
}
Point projection(Point p, Point a, Point b,
                 Point c) {  // 点p 在平面ABC 上的投影
    Point u = (b - a) ^ (c - a), v = p - a;
    double scalar = (v * u) / (u * u);
    return p - (u * scalar);
}
double dist(Point p, Point a, Point b) {  // 点p 到直线ab 的距离
    p = p - a;
    Point proj = projection(p, b - a);
    return sqrt(p * p - proj * proj);
}
//点到线段
double DistanceToSegment(Point p, Point a, Point b) {
    if (a == b) return (p - a).length();
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if (dcmp(v1 * v2) < 0)
        return v2.length();
    else if (dcmp(v1 * v3) > 0)
        return v3.length();
    else
        return (v1 ^ v2).length() / v1.length();
}
double area(Point a, Point b, Point c) {  // 三角形ABC 的面积
    double h = dist(a, b, c);
    return (h * (b - c).length()) / 2;
}
double volume(Point x, Point y, Point z) {  // 三个向量构成的体积
    Point base = Point(y.y * z.z - y.z * z.y, y.z * z.x - y.x * z.z,
                       y.x * z.y - y.y * z.x);
    return fabs(x.x * base.x + x.y * base.y + x.z * base.z) / 3;
```

```
}
//————————————————————————————————————————————————————————————
//空间直线
struct Line {
    Point a, b;
};
//空间直线间距离
double LineToLine(Line u, Line v, Point &tmp) {
    tmp = (u.a - u.b) ^ (v.a - v.b);
    return fabs((u.a - v.a) * tmp) / tmp.length();
}
//————————————————————————————————————————————————————————————
//面:  点+法线
//面交线:  两面交线与两面法线均垂直，法线叉积为其方向矢量.
//角平分面:  法向量为两平面法向量相加(内角) 或相减(外角).
struct Plane {
    Point p0, n;  // n:法线
    Plane() {}
    Plane(Point nn, Point pp0) {
        n = nn / nn.length();
        p0 = pp0;
    }
    Plane(Point a, Point b, Point c) {
        Point nn = (b - a) ^ (c - a);
        n = nn / nn.length();
        p0 = a;
    }
};
//角平分面
Plane jpfPlane(Point a1, Point a2, Point b, Point c) {
    Plane p1(a1, b, c), p2(a2, c, b);
    Point temp = p1.n + p2.n;  // 法向量为两平面法向量相加(内角)或相减(外角)
    return Plane(temp ^ (c - b), b);
}
//线面交点取线上任意两点
Point LinePlaneIntersection(Point p1, Point p2, Plane a) {
    Point p0 = a.p0;
    Point n = a.n, v = p2 - p1;
    double t = n * (p0 - p1) / (n * (p2 - p1));  //映射到法向量的比例
    return p1 + v * t;
}
//三面交点
Point PlaneInsertion(Plane a, Plane b, Plane c) {
    //两面交线与两面法线均垂直，法线叉积为其方向矢量
    Point nn = a.n ^ b.n, use = nn ^ a.n;
    Point st = LinePlaneIntersection(a.p0, a.p0 + use, b);  //得交线上一点
    return LinePlaneIntersection(st, st + nn, c);
}
double DistanceToPlane(Point p, Plane a) {
    Point p0 = a.p0, n = a.n;
    return fabs((p - p0) * n / n.length());
}
// 判定四点共面
bool isOnePlane(Point a, Point b, Point c, Point d) {
    double t = (d - a) * ((b - a) ^ (c - a));
    return dcmp(t) == 0;
```

```
}
//————————————————————————————————————————————————————————
//已知3点坐标，求平面ax+by+cz+d=0;
void getPlane(Point p1, Point p2, Point p3, double &a, double &b, double &c,
              double &d) {
    a = ((p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y));
    b = ((p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1.x) * (p3.z - p1.z));
    c = ((p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x));
    d = (0 - (a * p1.x + b * p1.y + c * p1.z));
}
/*
    题意：给定四点，求是否能够成四面体，若能则求出其内接圆心和半径
    分析：
        是否能构成四面体：
   三点成面的法线和另一点与三点中任一点相连的向量是否垂直？ 四面体内接球 球心：
   任意三个角平分面的交点 半径： 交点到任意面的距离
*/
int main() {
    Point p[4];
    while (~scanf("%lf%lf%lf", &p[0].x, &p[0].y, &p[0].z)) {
        for (int i = 1; i <= 3; i++)
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        if (isOnePlane(p[0], p[1], p[2], p[3])) {
            puts("0 0 0 0");
            continue;
        }
        Plane a = jpfPlane(p[3], p[2], p[1],
                           p[0]),  // 三个角平分面的交点即为圆心
            b = jpfPlane(p[3], p[0], p[1], p[2]),
              c = jpfPlane(p[3], p[1], p[0], p[2]);
        Plane d(p[0], p[1], p[2]);
        Point center = PlaneInsertion(a, b, c);
        double r = DistanceToPlane(center, d);
        printf("%.4f %.4f %.4f %.4f\n", center.x, center.y, center.z, r);
    }
}
```