

# 资料

## kmp 与 z函数

### 前缀函数

```
vector<int> prefix_function(string s) {  
    int n = (int)s.length();  
    vector<int> pi(n);  
    for (int i = 1; i < n; i++) {  
        int j = pi[i - 1];  
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];  
        if (s[i] == s[j]) j++;  
        pi[i] = j;  
    }  
    return pi;  
}
```

### kmp

```
vector<int> find_occurrences(string text, string pattern) {  
    string cur = pattern + '#' + text;  
    int sz1 = text.size(), sz2 = pattern.size();  
    vector<int> v;  
    vector<int> lps = prefix_function(cur);  
    for (int i = sz2 + 1; i <= sz1 + sz2; i++) {  
        if (lps[i] == sz2) v.push_back(i - 2 * sz2);  
    }  
    return v;  
}
```

对于一个长度为  $n$  的字符串  $s$ ，定义函数  $z[i]$  表示  $s$  和  $s[i, n-1]$ （即以  $s[i]$  开头的后缀）的最长公共前缀（LCP）的长度，则  $z$  被称为  $s$  的 Z 函数。特别地， $z[0] = 0$ 。

```

vector<int> z_function(string s) {
    int n = (int)s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r && z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        } else {
            z[i] = max(0, r - i + 1);
            while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        }
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}

```

## exGcd

```

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

```

# 组合数学

```
i64 qpow(i64 x, i64 p) {
    i64 ret = 1;
    while (p) {
        if (p & 1) ret = ret * x % mod;
        p >>= 1;
        x = x * x % mod;
    }
    return ret;
}

#define inv(x) qpow(x, mod-2)

std::vector<int> fact(1, 1);
std::vector<int> inv_fact(1, 1);

auto get_fact(int x, bool inv = 0) {
    while ((int)fact.size() < x + 1) {
        fact.push_back(1ll * fact.back() * fact.size() % mod);
        inv_fact.push_back(inv(fact.back()));
    }
    return (inv ? inv_fact[x] : fact[x]);
}

auto get_inv_fact(int x) { return get_fact(x, 1); }

i64 C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return 1ll * get_fact(n) * get_inv_fact(k) % mod * get_inv_fact(n - k) % mod;
}

i64 A(int n, int k) {
    return 1ll * get_fact(n) * get_inv_fact(n - k) % mod;
}

i64 F(int n) { return get_fact(n); }
```

# 马拉车

```
struct Manacher {
    std::vector<int> d1, d2;
    Manacher(std::string s) {
        int n = s.length();
        d1.assign(n, 0);
        d2.assign(n, 0);
        for (int i = 0, l = 0, r = -1; i < n; ++i) {
            int k = (i > r) ? 1 : std::min(d1[l + r - i], r - i + 1);
            while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) k++;
            d1[i] = k--;
            if (i + k > r) {
                r = i + k;
                l = i - k;
            }
        }
        for (int i = 0, l = 0, r = -1; i < n; ++i) {
            int k = (i > r) ? 0 : std::min(d2[l + r - i + 1], r - i + 1);
            while (i + k < n && i - k - 1 >= 0 && s[i + k] == s[i - k - 1]) k++;
            d2[i] = k--;
            if (i + k > r) {
                r = i + k;
                l = i - k - 1;
            }
        }
    }
    bool check(int l, int r) {
        if (r < l) return false;
        int len = r - l + 1;
        if (len % 2) {
            return d1[l + len / 2] * 2 - 1 < len;
        }
        else {
            return d2[l + len / 2] * 2 < len;
        }
    }
};
```

# 凸包

定义

凸多边形

凸多边形是指所有内角大小都在 $[0, \pi]$ 范围内的 简单多边形。

凸包

在平面上能包含所有给定点的最小凸多边形叫做凸包。

其定义为：对于给定集合  $X$ ，所有包含  $X$  的凸集的交集  $S$  被称为  $X$  的 凸包。

实际上可以理解为用一个橡皮筋包含住所有给定点的形态。

凸包用最小的周长围住了给定的所有点。如果一个凹多边形围住了所有的点，它的周长一定不是最小，如下图。根据三角不等式，凸多边形在周长上一定是最优的。

Andrew 算法求凸包

常用的求法有 Graham 扫描法和 Andrew 算法，这里主要介绍 Andrew 算法。

性质

该算法的时间复杂度为  $O(n \log n)$ ，其中  $n$  为待求凸包点集的大小，复杂度的瓶颈在于对所有点坐标的双关键字排序。

过程

首先把所有点以横坐标为第一关键字，纵坐标为第二关键字排序。

显然排序后最小的元素和最大的元素一定在凸包上。而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是「左拐」的，一旦出现右拐，就说明这一段不在凸包上。因此我们可以用一个单调栈来维护上下凸壳。

因为从左向右看，上下凸壳所旋转的方向不同，为了让单调栈起作用，我们首先 升序枚举 求出下凸壳，然后 降序 求出上凸壳。

求凸壳时，一旦发现即将进栈的点 ( $P$ ) 和栈顶的两个点 ( $S_1, S_2$ ，其中  $S_1$  为栈顶) 行进的方向向右旋转，即叉积小于 0：

$\overrightarrow{S_2 S_1} \times \overrightarrow{S_1 P} < 0$ ，则弹出栈顶，回到上一步，继续检测，直到

$\overrightarrow{S_2 S_1} \times \overrightarrow{S_1 P} \geq 0$  或者栈内仅剩一个元素为止。

通常情况下不需要保留位于凸包边上的点，因此上面一段中

$\overrightarrow{S_2 S_1} \times \overrightarrow{S_1 P} < 0$  这个条件中的「<」可以视情况改为  $\leq$ ，同时后面一个条件应改为  $>$ 。

实现

代码实现

```
// stk[] 是整型，存的是下标
// p[] 存储向量或点
tp = 0; // 初始化栈
std::sort(p + 1, p + 1 + n); // 对点进行排序
stk[++tp] = 1;
// 栈内添加第一个元素，且不更新 used，使得 1 在最后封闭凸包时也对单调栈更新
for (int i = 2; i <= n; ++i) {
    while (tp >= 2 // 下一行 * 操作符被重载为叉积
           && (p[stk[tp]] - p[stk[tp - 1]]) * (p[i] - p[stk[tp]]) <= 0)
        used[stk[tp--]] = 0;
    used[i] = 1; // used 表示在凸壳上
    stk[++tp] = i;
}
int tmp = tp; // tmp 表示下凸壳大小
for (int i = n - 1; i > 0; --i)
    if (!used[i]) {
        // ↓求上凸壳时不影响下凸壳
        while (tp > tmp && (p[stk[tp]] - p[stk[tp - 1]]) * (p[i] - p[stk[tp]]) <= 0)
            used[stk[tp--]] = 0;
        used[i] = 1;
        stk[++tp] = i;
    }
for (int i = 1; i <= tp; ++i) // 复制到新数组中去
    h[i] = p[stk[i]];
int ans = tp - 1;
```

根据上面的代码，最后凸包上有  $ans$  个元素（额外存储了 1 号点，因此  $h$  数组中有  $ans + 1$  个元素），并且按逆时针方向排序。周长就是所有相邻点的距离和。

# lca 最近公共祖先

```
std::vector<int> adj[MAXN];
int depth[MAXN], lg[MAXN], p[MAXN][30];
int lca(int x, int y) {
    if (depth[x] < depth[y])std::swap(x, y);
    while (depth[x] > depth[y])
        x = p[x][lg[depth[x]] - depth[y] - 1];
    if (x == y)return x;
    for (int k = lg[depth[x]] - 1;k >= 0;--k)
        if (p[x][k] != p[y][k])
            x = p[x][k], y = p[y][k];
    return p[x][0];
}

void dfs(int x, int par) {
    p[x][0] = par;
    depth[x] = depth[par] + 1;
    for (int i = 1;i <= lg[depth[x]];++i)
        p[x][i] = p[p[x][i - 1]][i - 1];

    for (int nxt : adj[x])if (nxt != par)dfs(nxt, x);
}

void init() {
    for (int i = 1;i <= n;++i)
        lg[i] = lg[i >> 1] + 1;
}
```

# 双模哈希 哈希字符串 树上哈希

```
#define M1 998244853
#define M2 1000000009
#define N 500000

i64 qpow(i64 x, i64 p, i64 mod) {
    i64 ret = 1;
    while (p) {
        if (p & 1) ret = ret * x % mod;
        p >>= 1;
        x = x * x % mod;
    }
    return ret;
}

struct hsh {
    i64 w1, w2;
    hsh operator * (const int w) {
        return { w1 * w % M1, w2 * w % M2 };
    }
    hsh operator * (const hsh w) {
        return { w1 * w.w1 % M1, w2 * w.w2 % M2 };
    }
    hsh operator + (const hsh w) {
        return { (w1 + w.w1) % M1, (w2 + w.w2) % M2 };
    }
    hsh operator - (const hsh w) {
        return { (w1 + M1 - w.w1) % M1, (w2 + M2 - w.w2) % M2 };
    }
    bool operator == (const hsh w) {
        return (w1 == w.w1) && (w2 == w.w2);
    }
    i64 wt() {
        return M2 * w1 + w2;
    }
    void show() { std::cout << w1 << ' ' << w2 << '\n'; }
}pw[N + 50], inv[N + 50];

std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());

int dep[MAXN], lg[MAXN], p[MAXN][30];
```



```

void init() {
    int b1 = rng() % M1 + 1, b2 = rng() % M2 + 1;
    pw[0] = inv[0] = { 1,1 };
    pw[1] = { b1,b2 };
    inv[1] = { qpow(b1,M1 - 2,M1),qpow(b2,M2 - 2,M2) };
    for (int i = 2;i <= N;i++) {
        pw[i] = pw[i - 1] * pw[1];
        inv[i] = inv[i - 1] * inv[1];
    }
    for (int i = 1;i <= N;++i)
        lg[i] = lg[i >> 1] + 1;
}

```

```

int lca(int x, int y) {
    if (dep[x] < dep[y])std::swap(x, y);
    while (dep[x] > dep[y])
        x = p[x][lg[dep[x]] - dep[y] - 1];
    if (x == y)return x;
    for (int k = lg[dep[x]] - 1;k >= 0;--k)
        if (p[x][k] != p[y][k])
            x = p[x][k], y = p[y][k];
    return p[x][0];
}

```

```

hsh h1[MAXN], h2[MAXN];
int Fa[MAXN];
std::string s;
void dfs(int x, int par) {
    Fa[x] = par;
    p[x][0] = par;
    dep[x] = dep[par] + 1;
    h1[x] = h1[par] + pw[dep[x]] * s[x];
    h2[x] = h2[par] + inv[dep[x]] * s[x];
    for (int i = 1;i <= lg[dep[x]];++i)
        p[x][i] = p[p[x][i - 1]][i - 1];
    for (int nxt : adj[x])if (nxt != par)dfs(nxt, x);
}

```

## segment\_tree

A:

```

#include <bits/stdc++.h>
#define int long long
#define MAXN 1000000
using namespace std;

#define ls(i) i<<1
#define rs(i) (i<<1)|1

int nums[MAXN], tree[4 * MAXN], lazy[4 * MAXN];

int set_up(int l, int r, int i = 1) {
    lazy[i] = 0;
    if (l == r)
        return tree[i] = nums[l];
    int mid = l + r >> 1;
    return tree[i] = set_up(l, mid, ls(i)) + set_up(mid + 1, r, rs(i));
}

inline void push_down(int i, int l, int r) {
    int mid = l + r >> 1;
    lazy[ls(i)] += lazy[i];
    lazy[rs(i)] += lazy[i];
    tree[ls(i)] += lazy[i] * (mid - l + 1);
    tree[rs(i)] += lazy[i] * (r - mid);
    lazy[i] = 0;
}

void update(int l, int r, int ql, int qr, int d, int i = 1) {
    if (ql > r || qr < l)
        return;
    if (ql <= l && r <= qr) {
        tree[i] += (r - l + 1) * d;
        if (l < r)
            lazy[i] += d;
    }
    else {
        int mid = l + r >> 1;
        push_down(i, l, r);
        update(l, mid, ql, qr, d, ls(i));
        update(mid + 1, r, ql, qr, d, rs(i));
        tree[i] = tree[ls(i)] + tree[rs(i)];
    }
}

```

```

int query(int l, int r, int ql, int qr, int i = 1) {
    if (ql > r || qr < l)
        return 0;
    if (ql <= l && r <= qr)
        return tree[i];
    else {
        int mid = l + r >> 1;
        push_down(i, l, r);
        return query(l, mid, ql, qr, ls(i)) + query(mid + 1, r, ql, qr, rs(i));
    }
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) cin >> nums[i];
    set_up(1, n);
    while (m--) {
        int t, x, y;
        cin >> t >> x >> y;
        if (x > y) swap(x, y);
        if (t == 1) {
            int k; cin >> k;
            update(1, n, x, y, k);
        }
        else
            cout << query(1, n, x, y) << '\n';
    }
    return 0;
}

```

B:

```

#include <bits/stdc++.h>
using namespace std;
#define int long long
#define endl "\n"
const int N = 200010, mod = 1e9 + 7;
const int inf = 1e15;
int power(int a, int b) {
    int res = 1;
    for (; b >= 1, a = a * a % mod) {
        if (b & 1) res = res * a % mod;
    }
    return res;
}
template <typename T, typename F>
struct segtree {
    #define ls (x << 1)
    #define rs (x << 1 | 1)
    #define mid (pl + pr >> 1)

    vector<T> tr;
    vector<T> d;
    vector<F> tag;
    vector<bool> istag;
    int n;

    void init() {
        tr.resize(n * 4);
        tag.resize(n * 4);
        istag.resize(n * 4);
        build(1, 1, n);
    }

    segtree(int _n):n(_n) {
        d.assign(n, T());
        init();
    }

    segtree(int _n, T *a):n(_n) {
        d.resize(n);
        for(int i = 0; i < n; i++) d[i] = a[i];
        init();
    }
}

```

```

segtree(int _n,vector<T> &a):n(_n) {
    d.resize(n);
    for(int i = 0; i < n;i++) d[i] = a[i];
    init();
}

void build(int x, int pl, int pr) {
    tag[x] = F();
    if (pl == pr) {
        tr[x] = d[pl - 1];
        return;
    }
    build(ls, pl, mid);
    build(rs, mid + 1, pr);
    push_up(x);
}

inline void add_tag(int x,F &val) {
    istag[x] = true;
    tr[x] = tr[x] + val;
    tag[x] = tag[x] + val;
}

inline void push_down(int x) {
    if (!istag[x]) return;
    add_tag(ls, tag[x]);
    add_tag(rs, tag[x]);
    tag[x] = F();
    istag[x] = false;
}

inline void push_up(int x) {
    tr[x] = tr[ls] + tr[rs];
}

void modify(int x, F val) {modify(1, 1, n, x, x, val);}
void modify(int l, int r, F val) {modify(1, 1, n, l, r, val);}
void modify(int x, int pl, int pr, int l, int r, F val) {
    if (pl >= l && pr <= r) {
        add_tag(x, val);
        return;
    }
    push_down(x);
}

```

```

        if (l <= mid) modify(ls, pl, mid, l, r, val);
        if (r > mid) modify(rs, mid + 1, pr, l, r, val);
        push_up(x);
    }

```

```

T query(int x) {return query(1, 1, n, x, x);}
T query(int l, int r) {return query(1, 1, n, l, r);}
T query(int x, int pl, int pr, int l, int r) {
    if(pl >= l && pr <= r) {
        return tr[x];
    }
    push_down(x);
    if(r <= mid) return query(ls, pl, mid, l, r);
    if(l > mid) return query(rs, mid + 1, pr, l, r);
    return query(ls, pl, mid, l, r) + query(rs, mid + 1, pr, l, r);
}

```

// 若区间范围为 [l,r] ,存在分段点 p 满足 [l,p) 的前缀代入函数为false, [p,r] 代入为true, 返回下标

```

template<class FU> int lower_bound(FU &&f) {return lower_bound(1, n, f);}
template<class FU> int lower_bound(int l, FU &&f) {return lower_bound(l, n, f);}
template<class FU> int lower_bound(int l, int r, FU &&f) {
    T cur = T();
    auto nlb = [&](auto self, int x, int pl, int pr, int l, int r) {
        // cout << " now_node " << x << ' ' << pl << " " << pr << " " << l << " " << r << endl;
        if (pl >= l && pr <= r) {
            // 如果当前区间已经整个在待查询区间内, 则测试加上此区间是否为 false ,若为 false ,则直接返回
            if(!f(cur + tr[x])) {
                cur = cur + tr[x];
                return pr + 1;
            }
            if(pl == pr) return pl;
        }
        push_down(x);
        if (l <= mid) {
            int la = self(self, ls, pl, mid, l, r);
            if(la != mid + 1) return la;
        }
        if (r > mid) {
            int lb = self(self, rs, mid + 1, pr, l, r);
            return lb;
        }
        return mid + 1;
    };
}

```

```

};

    int p = nlb(nlb, 1, 1, n, 1, r);
    return p;
}

#undef ls
#undef rs
#undef mid
};

struct lazy_tag {
    int add;
    lazy_tag(): add(0) {}
    lazy_tag(int a): add(a){}

    lazy_tag operator +(lazy_tag &a) {
        auto res = *this;
        if (a.add) res.add += a.add;
        return res;
    }
};

struct node {
    int sum, len;
    node(): sum(0), len(0) {}
    node(int a, int b): sum(a), len(b) {}

    node operator +(const node &a) {
        node res = *this;
        res.sum = a.sum + sum;
        res.len = a.len + len;
        return res;
    }

    node operator +(const lazy_tag &a) {
        node res = *this;
        if (a.add) res.sum += a.add * len;
        return res;
    }
};

using tree = segtree<node, lazy_tag>;
void solve() {

```

```

int n, m;
cin >> n >> m;
vector<node> a(n);
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    a[i] = {x, 1};
}
tree tr(n, a);
while (m--) {
    int op;
    cin >> op;
    if (op == 1) {
        int l, r, k;
        cin >> l >> r >> k;
        tr.modify(l, r, k);
    }
    else {
        int l, r;
        cin >> l >> r;
        cout << tr.query(l, r).sum << endl;
    }
}
}

signed main() {
    //freopen("title.in", "r", stdin);
    //freopen("title.out", "w", stdout);
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int t = 1;
    //cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}

```



# BIT 树状数组

```
#define lowbit(x) x&-x
int BIT[MAXN];
void update(int pos, int num = 1) {
    for (int i = pos; i <= n; i += lowbit(i))
        BIT[i] += num;
}
int query(int pos) {
    int ret = 0;
    for (int i = pos; i; i -= lowbit(i))
        ret += BIT[i];
    return ret;
}
//求逆序数
i64 cal(int nums[MAXN]) {
    for (int i = 1; i <= n; ++i)
        BIT[i] = 0;
    i64 rev = 0;
    for (int i = 1; i <= n; ++i) {
        cnt[i] = query(nums[i]);
        rev += i - 1 - cnt[i];
        update(nums[i]);
    }
    return rev;
}
```