

## Contents

1 Basic	1	3.2 2 SAT	2
1.1 Default Code	1	3.3 Max flow min cut	2
1.2 PBDS	1	4 String	3
1.3 int128 Input Output	1	4.1 Hash	3
2 Math	1	4.2 Zvalue	3
2.1 快速幂	1	4.3 Suffix Array	3
2.2 擴展歐幾里得	1	5 Geometry	3
3 Graph	1	5.1 Static Convex Hull	3
3.1 Tarjan SCC	1	6 Data Structure	4
		6.1 Sparse Table	4
		6.2 Segement Tree	4

## 1 Basic

### 1.1 Default Code

```
#include <bits/stdc++.h>
#define int long long
// #pragma GCC target("popcnt")
// #pragma GCC optimize("O3")
using namespace std;

void solve() {

}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int tt = 1;
    cin >> tt;
    while (t--) {
        solve();
    }
    return 0;
}
```

### 1.2 PBDS

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;

template
    <class T> using Tree = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;

/*
如果有 define int long long 記得拿掉
Tree<int> t 就跟 set<int> t 一樣，有包好 template
rb_tree_tag 使用紅黑樹
第三個參數 less<T> 為由小到大，greater<T> 為由大到小
插入 t.insert(); 刪除 t.erase();
t.order_of_key
(k); 從前往後數 k 是第幾個 (0-base 且回傳 int 型別)
t.find_by_order(k);
從前往後數第 k 個元素 (0-base 且回傳 iterator 型別)
t.lower_bound
(); t.upper_bound(); 用起來一樣 回傳 iterator
可以用 Tree<pair<int, int>> T 來模擬 mutiset
*/
```

### 1.3 int128 Input Output

```
// 抄 BBuf github 的

#include <bits/stdc++.h>
using namespace std;

void scan(__int128 &x) // 輸入
{
    x = 0;
    int f = 1;
    char ch;
    if((ch = getchar()) == '-') f = -f;
    else x = x*10 + ch-'0';
    while((ch = getchar()) >= '0' && ch <= '9')
        x = x*10 + ch-'0';
    x *= f;
}
```

```
}

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

int main()
{
    __int128 a, b;
    scan(a);
    scan(b);
    print(a + b);
    puts("");
    print(a*b);
    return 0;
}
```

## 2 Math

### 2.1 快速幂

```
// 根據費馬小定
理，若 a, p 互質，a^(p-2) 為 a 在 mod p 時的乘法逆元
int fast_pow(int a, int b, int mod)
{
    // a^b % mod
    int res = 1;
    while(b)
    {
        if(b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
```

### 2.2 擴展歐幾里得

```
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b)
{
    return a * b / gcd(a, b);
}

pair<int, int> ext_gcd
(int a, int b) //擴展歐幾里得 ax+by = gcd(a,b)
{
    if (b == 0)
        return {1, 0};
    if (a == 0)
        return {0, 1};
    int x, y;
    tie(x, y) = ext_gcd(b % a, a);
    return make_pair(y - b * x / a, x);
}
```

## 3 Graph

### 3.1 Tarjan SCC

```
class tarjan{
    // 1-base
    int time = 1;
    int id = 1;
    stack<int> s;
    vector<int> low;
    vector<int> dfn;
    vector<bool> in_stack;
    void dfs(int node, vector<vector<int>> &graph){
        in_stack[node] = true;
        s.push(node);
        dfn[node] = low[node] = time++;
        for(auto &j : graph[node]){
            if(dfn[j] == 0){

```

```

    dfs(j, graph);
    // 看看往下有沒有辦法回到更上面的點
    low[node] = min(low[node], low[j]);
}
else if(in_stack[j]){
    low[node] = min(low[node], low[j]);
}
}
vector<int> t; // 儲存這個強連通分量
if(dfn[node] == low[node]){
    while(s.top() != node){
        t.push_back(s.top());
        in_stack[s.top()] = false;
        scc_id[s.top()] = id;
        s.pop();
    }
    t.push_back(s.top());
    scc_id[s.top()] = id;
    in_stack[s.top()] = false;
    s.pop();
    id++;
}
if(!t.empty()) ans.push_back(t);
}
public:
    vector<int> scc_id;
    vector<vector<int>>> ans;
    // ans ans[i] 代表第 i 個強連通分量裡面包涵的點
    // scc_id[i] 代表第 i 個點屬於第幾個強連通分量
    vector
        <vector<int>> scc(vector<vector<int>>> &graph){
        int num = graph.size();
        scc_id.resize(num, -1);
        dfn.resize(num, 0);
        low.resize(num, 0);
        in_stack.resize(num, false);
        for(int i = 1; i < num; i++){
            if(dfn[i] == 0) dfs(i, graph);
        }
        return ans;
    }
};

```

## 3.2 2 SAT

```

// 用
// 下面的 tarjan scc 算法來解 2 sat 問題，若 事件 a 發
// 生時，事件 b 必然發生，我們須在 a -> b 建立一條有向
// 用
// cses 的 Giant Pizza 來舉例子，給定 n 個人 m 個配料
// 表，每個人可以提兩個要求，兩個要求至少要被滿足一個

// 3 5
// + 1 + 2
// - 1 + 3
// + 4 - 2

// 以這
// 個例子來說，第一個人要求要加 配料1 或者 配料2 其中
// 一項，第二個人要求不要 配料1 或者 要配料3 其中一項
// 試問能不能滿足所有人的要求，我們可以把 要加
// 配料 i 當作點 i，不加配料 i 當作點 i + m(配料數量)
// 關於第一個人的要求 我們可以看成若不加 配
// 料1 則必定要 配料2 以及 若不加 配料2 則必定要 配料1
// 關於第二個人要求 可看做加了 配料
// 1 就必定要加 配料3 以及 不加 配料3 就必定不加 配料1

// 以這些條件建立有像圖，並且
// 找尋 scc，若 i 以及 i + m 在同一個 scc 中代表無解
// 若要求解，則若 i 的 scc_id
// 小於 i + m 的 scc_id 則 i 為 true，反之為 false
// tarjan 的模板在上面
cin >> n >> m;

vector<vector<int>> graph(m * 2 + 1);
function<int(int)> tr = [&](int x){
    if(x > m) return x - m;
    return x + m;
};

for(int i = 0; i < n; i++){
    char c1, c2;
    int a, b;

```

```

    cin >> c1 >> a >> c2 >> b;
    // a 代表 a 為真，m + a 代表 a 為假
    if(c1 == '-') a += m;
    if(c2 == '-') b += m;
    graph[tr(a)].push_back(b);
    graph[tr(b)].push_back(a);
}

```

```

tarjan t;
auto scc = t.scc(graph);

for(int i = 1; i <= m; i++){
    if(t.scc_id[i] == t.scc_id[tr(i)]){
        cout << "IMPOSSIBLE\n";
        return 0;
    }
}

for(int i = 1; i <= m; i++){
    if(t.scc_id[i] < t.scc_id[tr(i)]){
        cout << '+';
    }
    else cout << '-';
    cout << ' ';
}
cout << '\n';

```

## 3.3 Max flow min cut

```
#define int long long
```

```

// Edmonds-Karp Algorithm Time: O(VE^2) 實際上會快一點
// 記得在 main 裡面 resize graph
// 最小割，找
// 到最少條的邊切除，使得從 src 到 end 的 maxflow 為 0
// 枚舉所有邊 i -> j，src 可
// 以到達 i 但無法到達 j，那這條邊為最小割裡的邊之一

```

```

class edge{
public:
    int next;
    int capacity;
    int rev;
    bool is_rev;
    edge(int _n, int _c, int _r, int _ir) :
        next(_n), capacity(_c), rev(_r), is_rev(_ir){};
};

```

```
vector<vector<edge>> graph;
```

```

void add_edge(int a, int b, int capacity){
    graph[a].push_back
        (edge(b, capacity, graph[b].size(), false));
    graph[b].
        push_back(edge(a, 0, graph[a].size() - 1, true));
}

```

```

int dfs(int now, int end
        , int flow, vector<pair<int, int>> &path, int idx){
    if(now == end) return flow;
    auto &e = graph[now][path[idx + 1].second];
    if(e.capacity > 0){
        auto ret = dfs(e.next
            , end, min(flow, e.capacity), path, idx + 1);
        if(ret > 0){
            e.capacity -= ret;
            graph[e.next][e.rev].capacity += ret;
            return ret;
        }
    }
    return 0;
}

```

```

vector<pair<int, int>> search_path(int start, int end){
    vector<pair<int, int>> ans;
    queue<int> q;
    vector
        <pair<int, int>> parent(graph.size(), {-1, -1});
    q.push(start);
    while(!q.empty()){
        int now = q.front();
        q.pop();
        for(int i = 0; i < (int)graph[now].size(); i++){
            auto &e = graph[now][i];
            if(e.
                capacity > 0 and parent[e.next].first == -1){

```

```

    parent[e.next] = {now, i};
    if(e.next == end) break;
    q.push(e.next);
}
}
if(parent[end].first == -1) return ans;
int now = end;
while(now != start){
    auto [node, idx] = parent[now];
    ans.emplace_back(node, idx);
    now = node;
}
ans.emplace_back(start, -1);
reverse(ans.begin(), ans.end());
return ans;
}

int maxflow(int start, int end, int node_num){
    int ans = 0;
    while(1){
        vector<bool> visited(node_num + 1, false);
        auto tmp = search_path(start, end);
        if(tmp.size() == 0) break;
        auto flow = dfs(start, end, 1e9, tmp, 0);
        ans += flow;
    }
    return ans;
}
}

```

## 4 String

### 4.1 Hash

```

vector<int> Pow(int num){
    int p = 1e9 + 7;
    vector<int> ans = {1};
    for(int i = 0; i < num; i++){
        ans.push_back(ans.back() * b % p);
    }
    return ans;
}

vector<int> Hash(string s){
    int p = 1e9 + 7;
    vector<int> ans = {0};
    for(char c:s){
        ans.push_back((ans.back() * b + c) % p);
    }
    return ans;
}

// 閉區間[l, r]
int query(
    vector<int> &vec, vector<int> &pow, int l, int r){
    int p = 1e9 + 7;
    int length = r - l + 1;
    return
        (vec[r + 1] - vec[l] * pow[length] % p + p) % p;
}

```

### 4.2 Zvalue

```

vector<int> z_func(string s1){
    int l = 0, r = 0, n = s1.size();
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i
            <= r and z[i - l] < r - i + 1) z[i] = z[i - l];
        else{
            z[i] = max(z[i], r - i + 1);
            while(i + z
                [i] < n and s1[i + z[i]] == s1[z[i]]) z[i]++;
            if(i + z[i] - 1 > r){
                l = i;
                r = i + z[i] - 1;
            }
        }
    }
    return z;
}

```

### 4.3 Suffix Array

```

struct SuffixArray {
    int n; string s;
    vector<int> sa, rk, lc;
};

```

```

// 想法：
// 排序過了，因此前綴長得像的會距離很近在差不多位置
// n: 字串長度
// sa: 後綴數組，sa[i] 表示第 i 小的後綴的起始位置
// rk: 排名數組，rk[i] 表示從位置 i 開始的後綴的排名
// lc: LCP 數組，
// lc[i] 表示 sa[i] 和 sa[i + 1] 的最長公共前綴長度
// 求 sa[i] 跟 sa[j] 的
// LCP 長度 當 i < j : min(lc[i] ..... lc[j - 1])
SuffixArray(const string &s_) {
    s = s_; n = s.length();
    sa.resize(n);
    lc.resize(n - 1);
    rk.resize(n);
    iota(sa.begin(), sa.end(), 0);
    sort(sa.begin(), sa.end(),
        [&](int a, int b) { return s[a] < s[b]; });
    rk[sa[0]] = 0;
    for(int i = 1; i < n; ++i)
        rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
    int k = 1;
    vector<int> tmp, cnt(n);
    tmp.reserve(n);
    while(rk[sa[n - 1]] < n - 1) {
        tmp.clear();
        for(int i = 0; i < k; ++i)
            tmp.push_back(n - k + i);
        for(auto i : sa)
            if(i >= k)
                tmp.push_back(i - k);
        fill(cnt.begin(), cnt.end(), 0);
        for(int i = 0; i < n; ++i)
            ++cnt[rk[i]];
        for(int i = 1; i < n; ++i)
            cnt[i] += cnt[i - 1];
        for(int i = n - 1; i >= 0; --i)
            sa[--cnt[rk[tmp[i]]]] = tmp[i];
        swap(rk, tmp);
        rk[sa[0]] = 0;
        for(int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] ||
                sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
        k *= 2;
    }
    for(int i = 0, j = 0; i < n; ++i) {
        if(rk[i] == 0) {
            j = 0;
        } else {
            for(j -= j > 0; i + j < n && sa[rk[i] - 1] + j
                < n && s[i + j] == s[sa[rk[i] - 1] + j];)
                ++j;
            lc[rk[i] - 1] = j;
        }
    }
}

```

## 5 Geometry

### 5.1 Static Convex Hull

```

// 需要使
// 用前一個向量模板的 point，需要 operator - 以及 <
// 需要前面向量模板的 cross

template<typename T>
vector<point<T>> getConvexHull(vector<point<T>> &pnts){
    sort(pnts.begin(), pnts.end());
    auto cmp = [&](point<T> a, point<T> b) {
        return a.x == b.x && a.y == b.y;
    };
    pnts.erase(unique(
        pnts.begin(), pnts.end(), cmp), pnts.end());
    if(pnts.size() <= 1) return pnts;
    vector<point<T>> hull;
    for(int i = 0; i < 2; i++){
        int t = hull.size();
        for(point<T> pnt : pnts){
            while(hull.size() - t >= 2 &&
                cross(hull.back() - hull[hull.size() - 2],
                    pnt - hull[hull.size() - 2]) < 0)
                // <= 0 或者 < 0 要看點有沒有在邊上
                hull.pop_back();
            hull.push_back(pnt);
        }
    }
}

```

```

        hull.push_back(pnt);
    }
    hull.pop_back();
    reverse(pnts.begin(), pnts.end());
}
return hull;
}

```

## 6 Data Structure

### 6.1 Sparse Table

```

class Sparse_Table{
// 0-base
// 要改成找最大把min換成max就好
private:
public:
    int spt[500005][22][2];
    Sparse_Table(vector<int> &ar){
        int n = ar.size();
        for (int i = 0; i < n; i++){
            spt[i][0][0] = ar[i];
            // spt[i][0][1] = ar[i];
        }
        for (int j = 1; (1 << j) <= n; j++) {
            for (int i = 0; (i + (1 << j) - 1) < n; i++) {
                spt[i][j][0] = min(spt[i + (1 << j) - 1][j - 1][0],
                                   spt[i][j - 1][0]);
                // spt[i][j][1] = max(spt[i + (1 << j) - 1][j - 1][1],
                                   spt[i][j - 1][1]);
            }
        }
    }
    int query_min(int l, int r)
    {
        if(l>r) return INT_MAX;
        int j = (int)__lg(r - l + 1);
        ///j = 31 - __builtin_clz(r - l + 1);
        return min
            (spt[l][j][0], spt[r - (1 << j) + 1][j][0]);
    }
    int query_max(int l, int r)
    {
        if(l>r) return INT_MAX;
        int j = (int)__lg(r - l + 1);
        ///j = 31 - __builtin_clz(r - l + 1);
        return max
            (spt[l][j][1], spt[r - (1 << j) + 1][j][1]);
    }
};

```

### 6.2 Segment Tree

```

// #define int long long
// 要改最大或者最小值線段樹需改 build 跟 queryRange
// 0-base 注意
template<typename T>
class segment_tree {
private:
    vector<T> tree, lazy;
    int size;
    void build
        (vector<T> &save, int node, int start, int end) {
        if (start == end) tree[node] = save[start];
        else {
            int mid = (start + end) / 2;
            build(save, 2 * node, start, mid);
            build(save, 2 * node + 1, mid + 1, end);
            tree[node] = tree[2 * node] + tree[2 * node + 1];
        }
    }
    void updateRange(int node
        , int start, int end, int l, int r, T delta) {
        if (lazy[node] != 0) {
            tree[node] += (end - start + 1) * lazy[node];
            if (start != end) {
                lazy[2 * node] += lazy[node];
                lazy[2 * node + 1] += lazy[node];
            }
            lazy[node] = 0;
        }
        if (start > end or start > r or end < l) return;
        if (start >= l and end <= r) {
            tree[node] += (end - start + 1) * delta;
            if (start != end) {

```

```

                lazy[2 * node] += delta;
                lazy[2 * node + 1] += delta;
            }
            return;
        }
        int mid = (start + end) / 2;
        updateRange(2 * node, start, mid, l, r, delta);
        updateRange
            (2 * node + 1, mid + 1, end, l, r, delta);
        tree[node] = tree[2 * node] + tree[2 * node + 1];
    }
    T queryRange
        (int node, int start, int end, int l, int r) {
        if (lazy[node] != 0) {
            tree[node] += (end - start + 1) * lazy[node];
            if (start != end) {
                lazy[2 * node] += lazy[node];
                lazy[2 * node + 1] += lazy[node];
            }
            lazy[node] = 0;
        }
        if (start > end or start > r or end < l){
            // return numeric_limits
                <T>::max(); // 找區間最小值用這行
            // return numeric_limits
                <T>::min(); // 找區間最大值用這行
            return 0; // 區間和
        }
        if (start >= l and end <= r) return tree[node];
        int mid = (start + end) / 2;
        T p1 = queryRange(2 * node, start, mid, l, r);
        T p2
            = queryRange(2 * node + 1, mid + 1, end, l, r);
        return p1 + p2;
    }
    void updateNode(
        int node, int start, int end, int idx, T delta) {
        if (start == end) tree[node] += delta;
        else {
            int mid = (start + end) / 2;
            if (start <= idx and idx <= mid)
                updateNode(2 * node, start, mid, idx, delta);
            else updateNode
                (2 * node + 1, mid + 1, end, idx, delta);
            tree[node] = tree[2 * node] + tree[2 * node + 1];
        }
    }
public:
    void build(vector<T> &save, int l, int r) {
        int n = size = save.size();
        tree.resize(4 * n);
        lazy.resize(4 * n);
        build(save, 1, l, r);
    }
    void modify_scope(int l, int r, T delta) {
        updateRange(1, 0, size - 1, l, r, delta);
    }
    void modify_node(int idx, T delta) {
        updateNode(1, 0, size - 1, idx, delta);
    }
    T query(int l, int r) {
        return queryRange(1, 0, size - 1, l, r);
    }
};

signed main()
{
    int n, q;
    cin >> n >> q;
    vector<int> save(n, 0);
    for(int i = 0; i < n; i++){
        cin >> save[i];
    }
    segment_tree<int> s;
    // init [0, n - 1]
    s.build(save, 0, n - 1);
    // modify [a, b] add c
    s.modify_scope(a, b, c);
    // query [a, b]
    s.query(a, b)
}

```