Standard Code Library

Your TeamName

Your School

July 29, 2025

Contents

一切的开始 **2**
宏定义 2

数据结构 **2**
并查集 2
线段树 2
树链剖分 4
重链剖分 4

数学 **6**
位运算 6
整型的位操作 6
位运算内建函数 7
线性基 7
异或空间线性基 7

图论 **8**
图的存储 8
邻接矩阵 8

计算几何 **8**
二维几何：点与向量 8
距离 9
距离 9

字符串 **11**
后缀自动机 11

杂项 **11**
STL 11

一切的开始

宏定义

- 需要 C++11

```cpp
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif
// -----------------------------------------------------------------------------
```

try

数据结构

并查集

```cpp
struct DSU{
    vector<size_t> p, size;

    explicit DSU(size_t _size): p(_size), size(_size, 1){
        iota(p.begin(), p.end(), 0);
    }

    size_t find(size_t x){ // 查找属于的集合
        return p[x] == x ? x : p[x] = find(p[x]);
    }

    size_t unite(size_t x, size_t y){ // 返回合并后集合的大小
        x = find(x), y = find(y);
        if(x == y) return size[x];
        if(size[x] < size[y]) swap(x, y); // 启发式合并
        p[y] = x;
        size[x] += size[y];
        return size[x];
    }
};

```

线段树

SGT.cpp

```cpp
template <typename T>
class SGT {
    vector<T> tree_sum, tree_max, tree_min;
    vector<T> lazy;
    vector<T> *arr;
    int n, root, n4, end;

    void update(int cl, int cr, int p) {
        int cm = cl + (cr - cl) / 2;
        if (cl != cr && lazy[p] != 0) {
            T val = lazy[p];
            lazy[p * 2] += val;
            lazy[p * 2 + 1] += val;

```

```
            tree_sum[p * 2] += val * (cm - cl + 1);
            tree_sum[p * 2 + 1] += val * (cr - cm);

            tree_max[p * 2] += val;
            tree_max[p * 2 + 1] += val;

            tree_min[p * 2] += val;
            tree_min[p * 2 + 1] += val;

            lazy[p] = 0;
        }
    }

    T range_sum(int l, int r, int cl, int cr, int p) {
        if (l > cr || r < cl) return 0;
        if (l <= cl && cr <= r) return tree_sum[p];
        int m = cl + (cr - cl) / 2;
        update(cl, cr, p);
        return range_sum(l, r, cl, m, p * 2) + range_sum(l, r, m + 1, cr, p * 2 + 1);
    }

    T range_max(int l, int r, int cl, int cr, int p) {
        if (l > cr || r < cl) return numeric_limits<T>::min();
        if (l <= cl && cr <= r) return tree_max[p];
        int m = cl + (cr - cl) / 2;
        update(cl, cr, p);
        return max(range_max(l, r, cl, m, p * 2), range_max(l, r, m + 1, cr, p * 2 + 1));
    }

    T range_min(int l, int r, int cl, int cr, int p) {
        if (l > cr || r < cl) return numeric_limits<T>::max();
        if (l <= cl && cr <= r) return tree_min[p];
        int m = cl + (cr - cl) / 2;
        update(cl, cr, p);
        return min(range_min(l, r, cl, m, p * 2), range_min(l, r, m + 1, cr, p * 2 + 1));
    }

    void range_add(int l, int r, T val, int cl, int cr, int p) {
        if (l > cr || r < cl) return;
        if (l <= cl && cr <= r) {
            lazy[p] += val;
            tree_sum[p] += val * (cr - cl + 1);
            tree_max[p] += val;
            tree_min[p] += val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        update(cl, cr, p);
        range_add(l, r, val, cl, m, p * 2);
        range_add(l, r, val, m + 1, cr, p * 2 + 1);

        tree_sum[p] = tree_sum[p * 2] + tree_sum[p * 2 + 1];
        tree_max[p] = max(tree_max[p * 2], tree_max[p * 2 + 1]);
        tree_min[p] = min(tree_min[p * 2], tree_min[p * 2 + 1]);
    }

    void build(int s, int t, int p) {
        if (s == t) {
            tree_sum[p] = (*arr)[s];
            tree_max[p] = (*arr)[s];
            tree_min[p] = (*arr)[s];
            return;
        }
        int m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);

        tree_sum[p] = tree_sum[p * 2] + tree_sum[p * 2 + 1];
        tree_max[p] = max(tree_max[p * 2], tree_max[p * 2 + 1]);
        tree_min[p] = min(tree_min[p * 2], tree_min[p * 2 + 1]);
    }
```

```cpp

public:
    explicit SGT<T>(vector<T> v) {
        n = v.size();
        n4 = n * 4;
        tree_sum = vector<T>(n4, 0);
        tree_max = vector<T>(n4, numeric_limits<T>::min());
        tree_min = vector<T>(n4, numeric_limits<T>::max());
        lazy = vector<T>(n4, 0);
        arr = &v;
        end = n - 1;
        root = 1;
        build(0, end, 1);
        arr = nullptr;
    }

    void show(int p, int depth = 0) {
        if (p > n4 || (tree_max[p] == numeric_limits<T>::min() &&
                       tree_min[p] == numeric_limits<T>::max())) return;
        show(p * 2, depth + 1);
        for (int i = 0; i < depth; ++i) putchar('\t');
        printf("sum:%d max:%d min:%d lazy:%d\n", tree_sum[p], tree_max[p], tree_min[p], lazy[p]);
        show(p * 2 + 1, depth + 1);
    }

    T range_sum(int l, int r) {
        return range_sum(l, r, 0, end, root);
    }

    T range_max(int l, int r) {
        return range_max(l, r, 0, end, root);
    }

    T range_min(int l, int r) {
        return range_min(l, r, 0, end, root);
    }

    void range_add(int l, int r, T val) {
        range_add(l, r, val, 0, end, root);
    }

    long long size() {
        return n;
    }
};
```

树链剖分

重链剖分

HLD.cpp

```cpp
#include "SGT.cpp"
// 点编号从 1 开始! 点编号从 1 开始! 点编号从 1 开始!
// 0 代表无! 0 代表无! 0 代表无!
// n 是大小! n 是大小! n 是大小!
template <typename T>
class HLD {
private:
    int n, root;
    vector<vector<int>> adj;
    vector<int> parent, depth, size, heavy, top, in, out, values;
    int time;

    void dfs1(int u, int p, int d) {
        parent[u] = p;
        depth[u] = d;
        size[u] = 1;
        heavy[u] = 0;
        int max_size = 0;

```

```cpp
        for (int v : adj[u]) {
            if (v == p) continue;
            dfs1(v, u, d + 1);
            size[u] += size[v];
            if (size[v] > max_size) {
                max_size = size[v];
                heavy[u] = v;
            }
        }
    }

    void dfs2(int u, int top_node) {
        top[u] = top_node;
        in[u] = time++;

        if (heavy[u] != -1) {
            dfs2(heavy[u], top_node);
            for (int v : adj[u]) {
                if (v != parent[u] && v != heavy[u]) {
                    dfs2(v, v);
                }
            }
        }
        out[u] = time - 1;
    }

    unique_ptr<SGT<T>> segTree;

public:
    HLD(int _n, int _root = 1) : n(_n), root(_root) {
        n++;
        adj.resize(n);
        parent.resize(n);
        depth.resize(n);
        size.resize(n);
        heavy.resize(n);
        top.resize(n);
        in.resize(n);
        out.resize(n);
        values.resize(n);
        time = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void setValue(int u, T val) {
        values[u] = val;
    }

    void init() {
        dfs1(root, 0, 0);
        time = 0;
        dfs2(root, root);

        vector<T> seg_values(n);
        for (int i = 0; i < n; i++) {
            seg_values[in[i]] = values[i];
        }
        segTree = make_unique<SGT<T>>(seg_values);
    }

    T pathSum(int u, int v) {
        T res = 0;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            res += segTree->range_sum(in[top[u]], in[u]);
            u = parent[top[u]];
        }
```

```cpp
        if (depth[u] > depth[v]) swap(u, v);
        res += segTree->range_sum(in[u], in[v]);
        return res;
    }

    T pathMax(int u, int v) {
        T res = numeric_limits<T>::min();
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            res = max(res, segTree->range_max(in[top[u]], in[u]));
            u = parent[top[u]];
        }
        if (depth[u] > depth[v]) swap(u, v);
        res = max(res, segTree->range_max(in[u], in[v]));
        return res;
    }

    T pathMin(int u, int v) {
        T res = numeric_limits<T>::max();
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            res = min(res, segTree->range_min(in[top[u]], in[u]));
            u = parent[top[u]];
        }
        if (depth[u] > depth[v]) swap(u, v);
        res = min(res, segTree->range_min(in[u], in[v]));
        return res;
    }

    void pathAdd(int u, int v, T val) {
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            segTree->range_add(in[top[u]], in[u], val);
            u = parent[top[u]];
        }
        if (depth[u] > depth[v]) swap(u, v);
        segTree->range_add(in[u], in[v], val);
    }

    T subtreeSum(int u) {
        return segTree->range_sum(in[u], out[u]);
    }

    T subtreeMax(int u) {
        return segTree->range_max(in[u], out[u]);
    }

    T subtreeMin(int u) {
        return segTree->range_min(in[u], out[u]);
    }

    void subtreeAdd(int u, T val) {
        segTree->range_add(in[u], out[u], val);
    }
};
```

数学

位运算

整型的位操作

```cpp
// 获取 a 的第 b 位，最低位编号为 0
int getBit(int a, int b) { return (a >> b) & 1; }

// 将 a 的第 b 位设置为 0 ，最低位编号为 0
int unsetBit(int a, int b) { return a & ~(1 << b); }

// 将 a 的第 b 位设置为 1 ，最低位编号为 0
int setBit(int a, int b) { return a | (1 << b); }
```

```cpp

// 将 a 的第 b 位取反 ，最低位编号为 0
int flapBit(int a, int b) { return a ^ (1 << b); }
```

位运算内建函数

1. `int __builtin_ffs(int x)`：返回 x 的二进制末尾最后一个 1 的位置，位置的编号从 1 开始（最低位编号为 1 ）。当 x 为 0 时返回 0 。
2. `int __builtin_clz(unsigned int x)`：返回 x 的二进制的前导 0 的个数。当 x 为 0 时，结果未定义。
3. `int __builtin_ctz(unsigned int x)`：返回 x 的二进制末尾连续 0 的个数。当 x 为 0 时，结果未定义。
4. `int __builtin_clrsb(int x)`：当 x 的符号位为 0 时返回 x 的二进制的前导 0 的个数减一，否则返回 x 的二进制的前导 1 的个数减一。
5. `int __builtin_popcount(unsigned int x)`：返回 x 的二进制中 1 的个数。
6. `int __builtin_parity(unsigned int x)`：判断 x 的二进制中 1 的个数的奇偶性。

这些函数都可以在函数名末尾添加 `l` 或 `ll`（如 `__builtin_popcountll`）来使参数类型变为 (`unsigned`) `long` 或 (`unsigned`) `long long`（返回值仍然是 `int` 类型）。

线性基

异或空间线性基

贪心法

可查询最大异或和

```cpp
struct BasisGreedy{
    ULL p[64];
    BasisGreedy(){memset(p, 0, sizeof p);}
    void insert(ULL x) {
        for (int i = 63; ~i; --i) {
            if (!(x >> i))  // x 的第 i 位是 0
                continue;
            if (!p[i]) {
                p[i] = x;
                break;
            }
            x ^= p[i];
        }
    }
    ULL query_max(){
        ULL ans = 0;
        for (int i = 63; ~i; --i) {
            ans = std::max(ans, ans ^ p[i]);
        }
        return ans;
    }
};
```

高斯消元法

可查询任意大异或和

```cpp
struct BasisGauss{
    vector<ULL> a;
    LL n, tmp, cnt;

    BasisGauss(){a = {0};}

    void insert(ULL x){
        a.push_back(x);
    }

    void init(){
        n = (LL)a.size() - 1;
```

```
        LL k=1;
        for(int i=63;i>=0;i--){
            int t=0;
            for(LL j=k;j<=n;j++){
                if((a[j]>>i)&1){
                    t=j;
                    break;
                }
            }
            if(t){
                swap(a[k],a[t]);
                for(LL j=1;j<=n;j++){
                    if(j!=k&&(a[j]>>i)&1) a[j]^=a[k];
                }
                k++;
            }
        }
        cnt = k-1;
        tmp = 1LL << cnt;
        if(cnt==n) tmp--;
    }

    LL query_xth(LL x){ // 从小到大, 若 x 为负数, 则查询倒数第几个
        if(x<0) x = tmp + x + 1;
        if(x>tmp) return -1;
        else{
            if(n>cnt) x--;
            LL ans=0;
            for(LL i=0; i<cnt; i++){
                if((x>>i)&1) ans^=a[cnt-i];
            }
            return ans;
        }
    }
};
```

图论

图的存储

邻接矩阵

```
struct Graph {
    std::vector< std::vector<int> > table;

    void init(int _n) {
        table.assign(_n + 1, {});
    }

    void add_edge(int u, int v) {
        table[u].push_back(v);
    }
} G;
```

计算几何

二维几何：点与向量

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
```

```cpp
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

距离

距离

欧氏距离

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

曼哈顿距离

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

便于求一个点任意一点到其他所有点的距离之和。

切比雪夫距离

$$d(A, B) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

便于求任意两点间距离的最值。

距离转化

假设 $A(x_1, y_1), B(x_2, y_2)$,

- A, B 两点的曼哈顿距离为 $(x_1 + y_1, x_1 - y_1), (x_2 + y_2, x_2 - y_2)$ 两点之间的切比雪夫距离。
- A, B 两点的切比雪夫距离为 $(\frac{x_1 + y_1}{2}, \frac{x_1 - y_1}{2}), (\frac{x_2 + y_2}{2}, \frac{x_2 - y_2}{2})$ 两点之间的曼哈顿距离。

距离之和

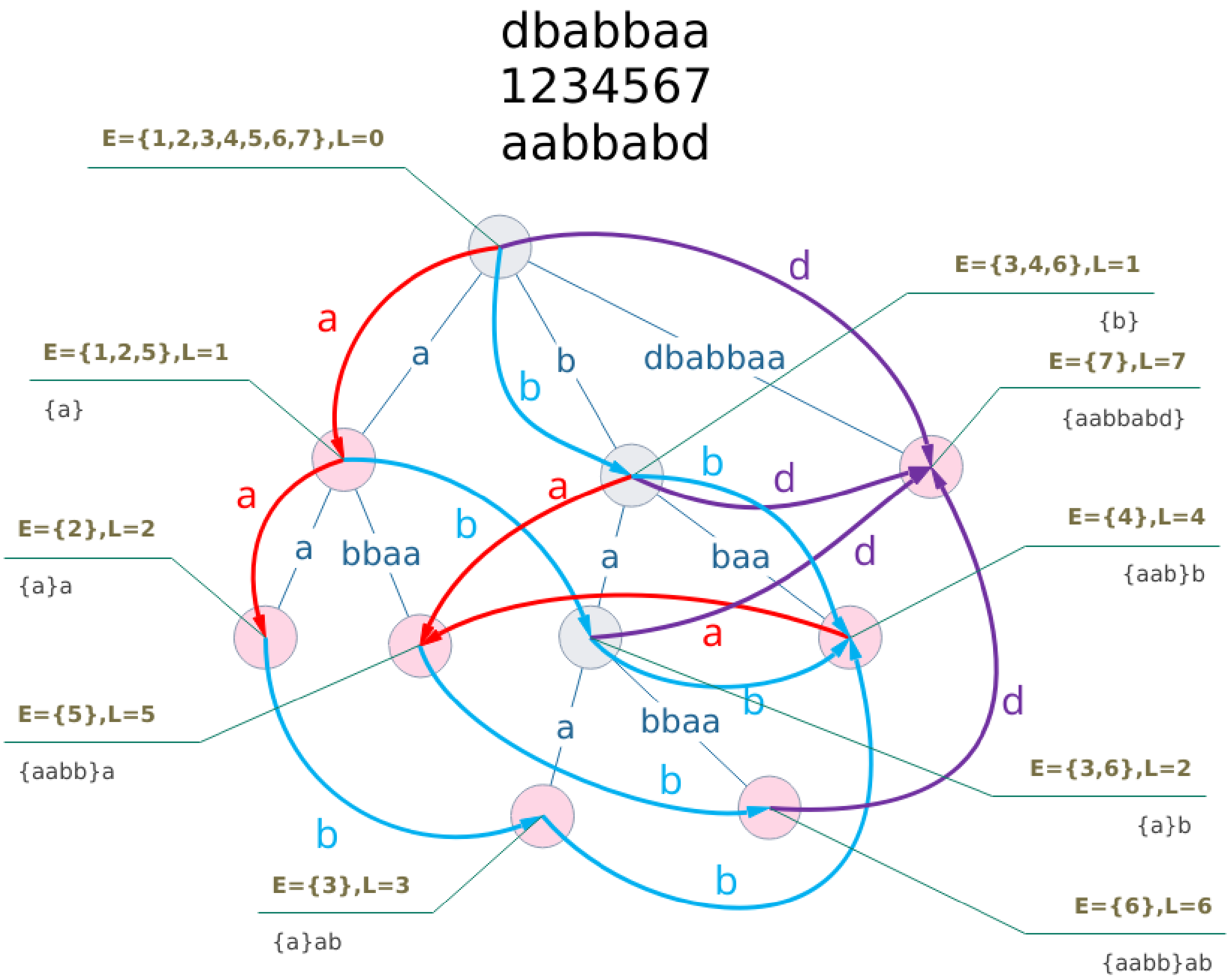
```
sumx[0] = 0;
sumy[0] = 0;
LL i, tx, ty;
cin >> n;
for(i=1; i<=n; i++){
    cin >> tx >> ty;
    // 求曼哈顿距离之和
    x[i] = hx[i] = tx;
    y[i] = hy[i] = ty;
    // 求切比雪夫距离之和
    x[i] = hx[i] = tx + ty;
    y[i] = hy[i] = tx - ty;
}
sort(hx+1, hx+1+n);
sort(hy+1, hy+1+n);
for(i=1; i<=n; i++){
    sumx[i] = sumx[i-1] + hx[i];
    sumy[i] = sumy[i-1] + hy[i];
}

LL calc_sum(LL i){
    LL xi = lower_bound(hx+1, hx+1+n, x[i]) - hx;
    LL yi = lower_bound(hy+1, hy+1+n, y[i]) - hy;
    return xi * x[i] - sumx[xi] + sumx[n] - sumx[xi] - (n-xi) * x[i]
    + yi * y[i] - sumy[yi] + sumy[n] - sumy[yi] - (n-yi) * y[i];
}

// 求 i 点与其他所有点曼哈顿距离之和
calc_sum(i);
// 求 i 点与其他所有点切比雪夫距离之和
calc_sum(i) / 2;
```

字符串

后缀自动机



杂项

STL

- copy

```
template <class InputIterator, class OutputIterator>
OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```