

# Standard Code Library

Your TeamName

Your School

July 23, 2025

# Contents

<b>一切的开始</b>	<b>2</b>
宏定义 . . . . .	2
<b>数据结构</b>	<b>2</b>
线段树 . . . . .	2
树链剖分 . . . . .	4
重链剖分 . . . . .	4
<b>数学</b>	<b>6</b>
线性基 . . . . .	6
异或空间线性基 . . . . .	6
<b>图论</b>	<b>7</b>
图的存储 . . . . .	7
邻接矩阵 . . . . .	7
<b>计算几何</b>	<b>8</b>
二维几何：点与向量 . . . . .	8
<b>字符串</b>	<b>9</b>
后缀自动机 . . . . .	9
<b>杂项</b>	<b>9</b>
STL . . . . .	9

## 一切的开始

### 宏定义

- 需要 C++11

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using LL = long long;
4 #define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
5 #define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
6 #ifdef zero1
7 #define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
8 void err() { cout << "\033[39;0m" << endl; }
9 template<template<typename...> class T, typename t, typename... A>
10 void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
11 template<typename T, typename... A>
12 void err(T a, A... x) { cout << a << ' '; err(x...); }
13 #else
14 #define dbg(...)
15 #endif
16 // -----
```

try

## 数据结构

### 线段树

SGT.cpp

```
1 template <typename T>
2 class SGT {
3     vector<T> tree_sum, tree_max, tree_min;
4     vector<T> lazy;
5     vector<T> *arr;
6     int n, root, n4, end;
7
8     void update(int cl, int cr, int p) {
9         int cm = cl + (cr - cl) / 2;
10        if (cl != cr && lazy[p] != 0) {
11            T val = lazy[p];
12            lazy[p * 2] += val;
13            lazy[p * 2 + 1] += val;
14
15            tree_sum[p * 2] += val * (cm - cl + 1);
16            tree_sum[p * 2 + 1] += val * (cr - cm);
17
18            tree_max[p * 2] += val;
19            tree_max[p * 2 + 1] += val;
20
21            tree_min[p * 2] += val;
22            tree_min[p * 2 + 1] += val;
23
24            lazy[p] = 0;
25        }
26    }
27
28    T range_sum(int l, int r, int cl, int cr, int p) {
29        if (l > cr || r < cl) return 0;
30        if (l <= cl && cr <= r) return tree_sum[p];
31        int m = cl + (cr - cl) / 2;
32        update(cl, cr, p);
33        return range_sum(l, r, cl, m, p * 2) + range_sum(l, r, m + 1, cr, p * 2 + 1);
34    }
35
36    T range_max(int l, int r, int cl, int cr, int p) {
37        if (l > cr || r < cl) return numeric_limits<T>::min();
38        if (l <= cl && cr <= r) return tree_max[p];
39        int m = cl + (cr - cl) / 2;
```

```

40     update(cl, cr, p);
41     return max(range_max(l, r, cl, m, p * 2), range_max(l, r, m + 1, cr, p * 2 + 1));
42 }
43
44 T range_min(int l, int r, int cl, int cr, int p) {
45     if (l > cr || r < cl) return numeric_limits<T>::max();
46     if (l <= cl && cr <= r) return tree_min[p];
47     int m = cl + (cr - cl) / 2;
48     update(cl, cr, p);
49     return min(range_min(l, r, cl, m, p * 2), range_min(l, r, m + 1, cr, p * 2 + 1));
50 }
51
52 void range_add(int l, int r, T val, int cl, int cr, int p) {
53     if (l > cr || r < cl) return;
54     if (l <= cl && cr <= r) {
55         lazy[p] += val;
56         tree_sum[p] += val * (cr - cl + 1);
57         tree_max[p] += val;
58         tree_min[p] += val;
59         return;
60     }
61     int m = cl + (cr - cl) / 2;
62     update(cl, cr, p);
63     range_add(l, r, val, cl, m, p * 2);
64     range_add(l, r, val, m + 1, cr, p * 2 + 1);
65
66     tree_sum[p] = tree_sum[p * 2] + tree_sum[p * 2 + 1];
67     tree_max[p] = max(tree_max[p * 2], tree_max[p * 2 + 1]);
68     tree_min[p] = min(tree_min[p * 2], tree_min[p * 2 + 1]);
69 }
70
71 void build(int s, int t, int p) {
72     if (s == t) {
73         tree_sum[p] = (*arr)[s];
74         tree_max[p] = (*arr)[s];
75         tree_min[p] = (*arr)[s];
76         return;
77     }
78     int m = s + (t - s) / 2;
79     build(s, m, p * 2);
80     build(m + 1, t, p * 2 + 1);
81
82     tree_sum[p] = tree_sum[p * 2] + tree_sum[p * 2 + 1];
83     tree_max[p] = max(tree_max[p * 2], tree_max[p * 2 + 1]);
84     tree_min[p] = min(tree_min[p * 2], tree_min[p * 2 + 1]);
85 }
86
87 public:
88     explicit SGT<T>(vector<T> v) {
89         n = v.size();
90         n4 = n * 4;
91         tree_sum = vector<T>(n4, 0);
92         tree_max = vector<T>(n4, numeric_limits<T>::min());
93         tree_min = vector<T>(n4, numeric_limits<T>::max());
94         lazy = vector<T>(n4, 0);
95         arr = &v;
96         end = n - 1;
97         root = 1;
98         build(0, end, 1);
99         arr = nullptr;
100     }
101
102     void show(int p, int depth = 0) {
103         if (p > n4 || (tree_max[p] == numeric_limits<T>::min() &&
104             tree_min[p] == numeric_limits<T>::max())) return;
105         show(p * 2, depth + 1);
106         for (int i = 0; i < depth; ++i) putchar('\t');
107         printf("sum:%d max:%d min:%d lazy:%d\n", tree_sum[p], tree_max[p], tree_min[p], lazy[p]);
108         show(p * 2 + 1, depth + 1);
109     }
110

```

```

111     T range_sum(int l, int r) {
112         return range_sum(l, r, 0, end, root);
113     }
114
115     T range_max(int l, int r) {
116         return range_max(l, r, 0, end, root);
117     }
118
119     T range_min(int l, int r) {
120         return range_min(l, r, 0, end, root);
121     }
122
123     void range_add(int l, int r, T val) {
124         range_add(l, r, val, 0, end, root);
125     }
126
127     long long size() {
128         return n;
129     }
130 };

```

## 树链剖分

### 重链剖分

HLD.cpp

```

1  #include "SGT.cpp"
2  // 点编号从 1 开始! 点编号从 1 开始!
3  // 0 代表无! 0 代表无! 0 代表无!
4  // n 是大小! n 是大小! n 是大小!
5  template <typename T>
6  class HLD {
7  private:
8      int n, root;
9      vector<vector<int>>> adj;
10     vector<int> parent, depth, size, heavy, top, in, out, values;
11     int time;
12
13     void dfs1(int u, int p, int d) {
14         parent[u] = p;
15         depth[u] = d;
16         size[u] = 1;
17         heavy[u] = 0;
18         int max_size = 0;
19
20         for (int v : adj[u]) {
21             if (v == p) continue;
22             dfs1(v, u, d + 1);
23             size[u] += size[v];
24             if (size[v] > max_size) {
25                 max_size = size[v];
26                 heavy[u] = v;
27             }
28         }
29     }
30
31     void dfs2(int u, int top_node) {
32         top[u] = top_node;
33         in[u] = time++;
34
35         if (heavy[u] != -1) {
36             dfs2(heavy[u], top_node);
37             for (int v : adj[u]) {
38                 if (v != parent[u] && v != heavy[u]) {
39                     dfs2(v, v);
40                 }
41             }
42         }
43         out[u] = time - 1;
44     }

```

```

45
46     unique_ptr<SGT<T>> segTree;
47
48 public:
49     HLD(int _n, int _root = 1) : n(_n), root(_root) {
50         n++;
51         adj.resize(n);
52         parent.resize(n);
53         depth.resize(n);
54         size.resize(n);
55         heavy.resize(n);
56         top.resize(n);
57         in.resize(n);
58         out.resize(n);
59         values.resize(n);
60         time = 0;
61     }
62
63     void addEdge(int u, int v) {
64         adj[u].push_back(v);
65         adj[v].push_back(u);
66     }
67
68     void setValue(int u, T val) {
69         values[u] = val;
70     }
71
72     void init() {
73         dfs1(root, 0, 0);
74         time = 0;
75         dfs2(root, root);
76
77         vector<T> seg_values(n);
78         for (int i = 0; i < n; i++) {
79             seg_values[in[i]] = values[i];
80         }
81         segTree = make_unique<SGT<T>>(seg_values);
82     }
83
84     T pathSum(int u, int v) {
85         T res = 0;
86         while (top[u] != top[v]) {
87             if (depth[top[u]] < depth[top[v]]) swap(u, v);
88             res += segTree->range_sum(in[top[u]], in[u]);
89             u = parent[top[u]];
90         }
91         if (depth[u] > depth[v]) swap(u, v);
92         res += segTree->range_sum(in[u], in[v]);
93         return res;
94     }
95
96     T pathMax(int u, int v) {
97         T res = numeric_limits<T>::min();
98         while (top[u] != top[v]) {
99             if (depth[top[u]] < depth[top[v]]) swap(u, v);
100             res = max(res, segTree->range_max(in[top[u]], in[u]));
101             u = parent[top[u]];
102         }
103         if (depth[u] > depth[v]) swap(u, v);
104         res = max(res, segTree->range_max(in[u], in[v]));
105         return res;
106     }
107
108     T pathMin(int u, int v) {
109         T res = numeric_limits<T>::max();
110         while (top[u] != top[v]) {
111             if (depth[top[u]] < depth[top[v]]) swap(u, v);
112             res = min(res, segTree->range_min(in[top[u]], in[u]));
113             u = parent[top[u]];
114         }
115         if (depth[u] > depth[v]) swap(u, v);

```

```

116         res = min(res, segTree->range_min(in[u], in[v]));
117         return res;
118     }
119
120     void pathAdd(int u, int v, T val) {
121         while (top[u] != top[v]) {
122             if (depth[top[u]] < depth[top[v]]) swap(u, v);
123             segTree->range_add(in[top[u]], in[u], val);
124             u = parent[top[u]];
125         }
126         if (depth[u] > depth[v]) swap(u, v);
127         segTree->range_add(in[u], in[v], val);
128     }
129
130     T subtreeSum(int u) {
131         return segTree->range_sum(in[u], out[u]);
132     }
133
134     T subtreeMax(int u) {
135         return segTree->range_max(in[u], out[u]);
136     }
137
138     T subtreeMin(int u) {
139         return segTree->range_min(in[u], out[u]);
140     }
141
142     void subtreeAdd(int u, T val) {
143         segTree->range_add(in[u], out[u], val);
144     }
145 };

```

## 数学

### 线性基

#### 异或空间线性基

#### 贪心法

#### 可查询最大异或和

```

1  struct BasisGreedy{
2      ULL p[64];
3      BasisGreedy(){memset(p, 0, sizeof p);}
4      void insert(ULL x) {
5          for (int i = 63; ~i; --i) {
6              if (!(x >> i)) // x 的第 i 位是 0
7                  continue;
8              if (!p[i]) {
9                  p[i] = x;
10                 break;
11             }
12             x ^= p[i];
13         }
14     }
15     ULL query_max(){
16         ULL ans = 0;
17         for (int i = 63; ~i; --i) {
18             ans = std::max(ans, ans ^ p[i]);
19         }
20         return ans;
21     }
22 };

```

### 高斯消元法

#### 可查询任意大异或和

```

1  struct BasisGauss{
2      vector<ULL> a;

```

```

3     LL n, tmp, cnt;
4
5     BasisGauss(){a = {0};}
6
7     void insert(ULL x){
8         a.push_back(x);
9     }
10
11    void init(){
12        n = (LL)a.size() - 1;
13        LL k=1;
14        for(int i=63;i>=0;i--){
15            int t=0;
16            for(LL j=k;j<=n;j++){
17                if((a[j]>>i)&1){
18                    t=j;
19                    break;
20                }
21            }
22            if(t){
23                swap(a[k],a[t]);
24                for(LL j=1;j<=n;j++){
25                    if(j!=k&&(a[j]>>i)&1) a[j]^=a[k];
26                }
27                k++;
28            }
29        }
30        cnt = k-1;
31        tmp = 1LL << cnt;
32        if(cnt==n) tmp--;
33    }
34
35    LL query_xth(LL x){ // 从小到大, 若 x 为负数, 则查询倒数第几个
36        if(x<0) x = tmp + x + 1;
37        if(x>tmp) return -1;
38        else{
39            if(n>cnt) x--;
40            LL ans=0;
41            for(LL i=0; i<cnt; i++){
42                if((x>>i)&1) ans^=a[cnt-i];
43            }
44            return ans;
45        }
46    }
47 };

```

## 图论

### 图的存储

#### 邻接矩阵

```

1 struct Graph {
2     std::vector< std::vector<int> > table;
3
4     void init(int _n) {
5         table.assign(_n + 1, {});
6     }
7
8     void add_edge(int u, int v) {
9         table[u].push_back(v);
10    }
11 } G;

```



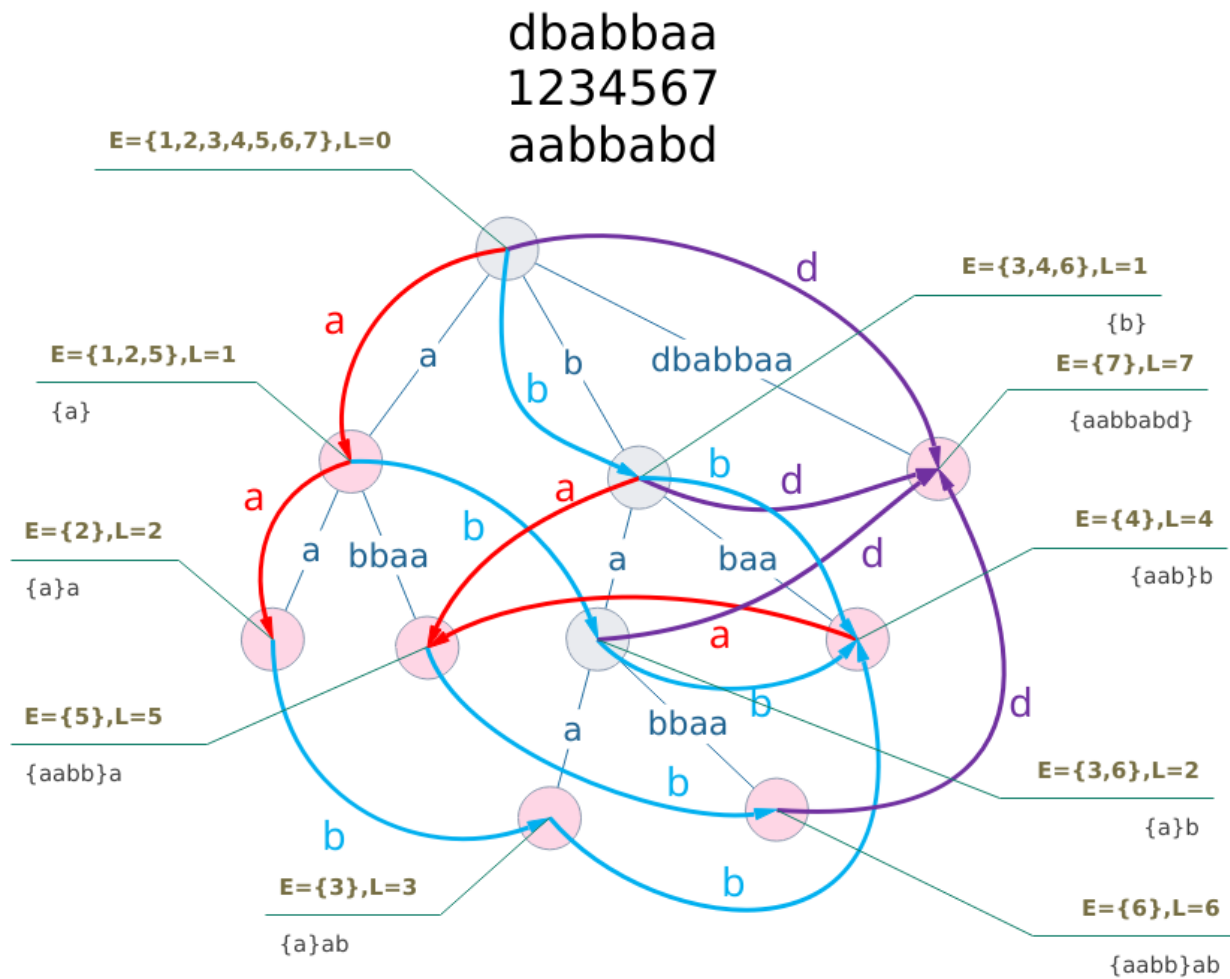
# 计算几何

## 二维几何：点与向量

```
1  #define y1 yy1
2  #define nxt(i) ((i + 1) % s.size())
3  typedef double LD;
4  const LD PI = 3.14159265358979323846;
5  const LD eps = 1E-10;
6  int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
7  struct L;
8  struct P;
9  typedef P V;
10 struct P {
11     LD x, y;
12     explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
13     explicit P(const L& l);
14 };
15 struct L {
16     P s, t;
17     L() {}
18     L(P s, P t): s(s), t(t) {}
19 };
20
21 P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
22 P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
23 P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
24 P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
25 inline bool operator < (const P& a, const P& b) {
26     return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
27 }
28 bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
29 P::P(const L& l) { *this = l.t - l.s; }
30 ostream &operator << (ostream &os, const P &p) {
31     return (os << "(" << p.x << "," << p.y << ")");
32 }
33 istream &operator >> (istream &is, P &p) {
34     return (is >> p.x >> p.y);
35 }
36
37 LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
38 LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
39 LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
40 LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
41 // -----
```

## 字符串

### 后缀自动机



## 杂项

### STL

- copy

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