

# Standard Code Library

ONGLU

North Eastern University

August 2021

# Contents

<b>初始化</b>	<b>2</b>
<b>数据结构</b>	<b>2</b>
轻重链剖分 . . . . .	2
二维树状数组 . . . . .	2
平衡树 . . . . .	3
可持久化数据结构 . . . . .	4
可持久化 Trie . . . . .	4
主席树 (静态第 k 小) . . . . .	5
<b>数学</b>	<b>6</b>
<b>图论</b>	<b>6</b>
树上问题 . . . . .	6
树的直径 . . . . .	6
求 LCA . . . . .	7
树上启发式合并 . . . . .	8
图算法 . . . . .	9
第 k 短路 . . . . .	9
网络流 . . . . .	9
Dinic 算法 . . . . .	9
EK 算法费用流 . . . . .	10
连通性算法 . . . . .	10
Tarjan 强连通分量 . . . . .	10
<b>计算几何</b>	<b>11</b>
<b>字符串</b>	<b>11</b>
字串哈希 . . . . .	11
Trie . . . . .	12
KMP 算法 . . . . .	12
manacher 算法 . . . . .	13
AC 自动机 . . . . .	14
<b>杂项</b>	<b>15</b>
int128 . . . . .	15
tips: . . . . .	15

## 初始化

## 数据结构

### 轻重链剖分

```
1 void dfs1(int x, int pre) {
2     siz[x] = 1; mson[x] = 0;
3     dth[x] = dth[pre] + 1;
4     fa[x] = pre;
5     for(auto y : son[x]) if(y != pre) {
6         dfs1(y, x);
7         siz[x] += siz[y];
8         if(!mson[x] || siz[y] > siz[mson[x]])
9             mson[x] = y;
10    }
11 }
12 void dfs2(int x, int pre, int ntp) {
13     id[x] = ++idcnt;
14     ltp[x] = ntp;
15     if(mson[x]) dfs2(mson[x], x, ntp);
16     for(auto y : son[x]) {
17         if(y == mson[x] || y == pre) continue;
18         dfs2(y, x, y);
19     }
20 }
21 void link_modify(int x, int y, int z) {
22     z %= mod;
23     while(ltp[x] != ltp[y]) {
24         dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
25         modify(1, n, id[ltp[x]], id[x], 1, z);
26         x = fa[ltp[x]];
27     }
28     dth[x] < dth[y] && (x ^= y ^= x ^= y);
29     modify(1, n, id[y], id[x], 1, z);
30 }
31 }
32 int link_query(int x, int y) {
33     int ans = 0;
34     while(ltp[x] != ltp[y]) {
35         dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
36         ans = (1ll * ans + query(1, n, id[ltp[x]], id[x], 1)) % mod;
37         x = fa[ltp[x]];
38     }
39     dth[x] < dth[y] && (x ^= y ^= x ^= y);
40     ans = (1ll * ans + query(1, n, id[y], id[x], 1)) % mod;
41     return ans;
42 }
```

### 二维树状数组

- 矩阵修改，矩阵查询

查询前缀和公式：

令  $d[i][j]$  为差分数组，定义  $d[i][j] = a[i][j] - (a[i-1][j] - a[i][j-1] - a[i-1][j])$

$$\sum_{i=1}^x \sum_{j=1}^y a[i][j] = (x+1) * (y+1) * d[i][j] - (y+1) * i * d[i][j] + d[i][j] * i * j$$

```
1 void modify(int x, int y, int v) {
2     for(int rx = x; rx <= n; rx += rx & -rx) {
3         for(int ry = y; ry <= m; ry += ry & -ry) {
4             tree[rx][ry][0] += v;
5             tree[rx][ry][1] += v * x;
6             tree[rx][ry][2] += v * y;
7             tree[rx][ry][3] += v * x * y;
8         }
9     }
10 }
```

```

11 void range_modify(int x, int y, int xx, int yy, int v) {
12     modify(xx + 1, yy + 1, v);
13     modify(x, yy + 1, -v);
14     modify(xx + 1, y, -v);
15     modify(x, y, v);
16 }
17 int query(int x, int y) {
18     int ans = 0;
19     for(int rx = x; rx; rx -= rx & -rx) {
20         for(int ry = y; ry; ry -= ry & -ry) {
21             ans += (x + 1) * (y + 1) * tree[rx][ry][0]
22                 - tree[rx][ry][1] * (y + 1) - tree[rx][ry][2] * (x + 1)
23                 + tree[rx][ry][3];
24         }
25     }
26     return ans;
27 }
28 int range_query(int x, int y, int xx, int yy) {
29     return query(xx, yy) + query(x - 1, y - 1)
30         - query(x - 1, yy) - query(xx, y - 1);
31 }

```

## 平衡树

- luogu P3369 【模板】普通平衡树

```

1  #define val(x) tree[x].val
2  #define cnt(x) tree[x].cnt
3  #define siz(x) tree[x].siz
4  #define fa(x) tree[x].fa
5  #define son(x, k) tree[x].ch[k]
6  struct Tree {
7      struct node {
8          int val, cnt, siz, fa, ch[2];
9      } tree[N];
10     int root, tot;
11     int chk(int x) {
12         return son(fa(x), 1) == x;
13     }
14     void update(int x) {
15         siz(x) = siz(son(x, 0)) + siz(son(x, 1)) + cnt(x);
16     }
17     void rotate(int x) {
18         int y = fa(x), z = fa(y), k = chk(x), w = son(x, k ^ 1);
19         son(y, k) = w; fa(w) = y;
20         son(z, chk(y)) = x; fa(x) = z;
21         son(x, k ^ 1) = y; fa(y) = x;
22         update(y); update(x);
23     }
24     void splay(int x, int goal = 0) {
25         while(fa(x) != goal) {
26             int y = fa(x), z = fa(y);
27             if(z != goal) {
28                 //双旋
29                 if(chk(y) == chk(x)) rotate(y);
30                 else rotate(x);
31             }
32             rotate(x);
33         }
34         if(!goal) root = x;
35     }
36     int New(int x, int pre) {
37         tot++;
38         if(pre) son(pre, x > val(pre)) = tot;
39         val(tot) = x; fa(tot) = pre;
40         siz(tot) = cnt(tot) = 1;
41         son(tot, 0) = son(tot, 1) = 0;
42         return tot;
43     }

```

```

44 void Insert(int x) {
45     int cur = root, p = 0;
46     while(cur && val(cur) != x) {
47         p = cur;
48         cur = son(cur, x > val(cur));
49     }
50     if(cur) cnt(cur)++;
51     else cur = New(x, p);
52     splay(cur);
53 }
54 void Find(int x) {
55     if(!root) return ;
56     int cur = root;
57     while(val(cur) != x && son(cur, x > val(cur)))
58         cur = son(cur, x > val(cur));
59     splay(cur);
60 }
61 int Pre(int x) {
62     Find(x);
63     if(val(root) < x) return root;
64     int cur = son(root, 0);
65     while(son(cur, 1))
66         cur = son(cur, 1);
67     return cur;
68 }
69 int Succ(int x) {
70     Find(x);
71     if(val(root) > x) return root;
72     int cur = son(root, 1);
73     while(son(cur, 0))
74         cur = son(cur, 0);
75     return cur;
76 }
77 void Del(int x) {
78     int lst = Pre(x), nxt = Succ(x);
79     splay(lst); splay(nxt, lst);
80     int cur = son(nxt, 0);
81     if(cnt(cur) > 1) cnt(cur)--, splay(cur);
82     else son(nxt, 0) = 0, splay(nxt);
83 }
84 int Kth(int k) {
85     int cur = root;
86     while(1) {
87         if(son(cur, 0) && siz(son(cur, 0)) >= k) cur = son(cur, 0);
88         else if(siz(son(cur, 0)) + cnt(cur) >= k) return cur;
89         else k -= siz(son(cur, 0)) + cnt(cur), cur = son(cur, 1);
90     }
91 }
92 } T;

```

## 可持久化数据结构

### 可持久化 Trie

```

1 namespace Trie {
2     struct node {
3         int ch[2], ed, siz;
4     } tree[N * 40];
5     int tot = 0;
6     int _new() {
7         tot++;
8         tree[tot].ch[0] = 0;
9         tree[tot].ch[1] = 0;
10        tree[tot].ed = tree[tot].siz = 0;
11        return tot;
12    }
13    void init() {
14        tot = 0;
15        rt[0] = _new();
16    }
17    int Insert(int x, int t, int i = 15) {

```

```

18     int u = _new(), f = (x >> i) & 1;
19     tree[u] = tree[t];
20     if(i == -1) {
21         ed(u)++;
22         siz(u)++;
23         return u;
24     }
25     son(u, f) = Insert(x, son(t, f), i - 1);
26     siz(u) = siz(son(u, 0)) + siz(son(u, 1));
27     return u;
28 }
29 void print(int u, int now) {
30     if(u == 0) return;
31     for(int i = 1; i <= ed(u); i++) printf("%d ", now);
32     if(son(u, 0)) print(son(u, 0), now * 2);
33     if(son(u, 1)) print(son(u, 1), now * 2 + 1);
34 }
35 int query(int u1, int u2, int x, int i = 15, int now = 0) {
36     if(i == -1) return now;
37     int f = (x >> i) & 1;
38     if(siz(son(u1, f ^ 1)) - siz(son(u2, f ^ 1)) > 0)
39         return query(son(u1, f ^ 1), son(u2, f ^ 1), x, i - 1, now * 2 + (f ^ 1));
40     else return query(son(u1, f), son(u2, f), x, i - 1, now * 2 + (f));
41 }
42 }

```

### 主席树（静态第 k 小）

建立权值树，那么  $[l, r]$  的区间权值树就是第  $r$  个版本减去第  $l - 1$  个版本的树。

```

1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cmath>
5  #include <assert.h>
6  #define Mid ((l + r) / 2)
7  #define lson (rt << 1)
8  #define rson (rt << 1 | 1)
9  using namespace std;
10 int read() {
11     char c; int num, f = 1;
12     while(c = getchar(), !isdigit(c)) if(c == '-') f = -1; num = c - '0';
13     while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
14     return f * num;
15 }
16 const int N = 1e7 + 1009;
17 const int M = 2e5 + 1009;
18 struct node {
19     int ls, rs, v;
20 } tree[N];
21 int tb;
22 int n, m, tot, a[M], b[M], rt[M];
23 int _new(int ls, int rs, int v) {
24     tree[++tot].ls = ls;
25     tree[tot].rs = rs;
26     tree[tot].v = v;
27     return tot;
28 }
29 void update(int rt) {
30     tree[rt].v = tree[tree[rt].ls].v + tree[tree[rt].rs].v;
31 }
32 int build(int l, int r) {
33     if(l == r) return _new(0, 0, 0);
34     int x = _new(build(l, Mid), build(Mid + 1, r), 0);
35     update(x);
36     return x;
37 }
38 int add(int l, int r, int p, int rt, int v) {
39     int x = ++tot;
40     tree[x] = tree[rt];
41     if(l == r) {

```

```

42     tree[x].v += v;
43     return x;
44 }
45 if(p <= Mid) tree[x].ls = add(l, Mid, p, tree[x].ls, v);
46 else tree[x].rs = add(Mid + 1, r, p, tree[x].rs, v);
47 update(x);
48 return x;
49 }
50 int query(int l, int r, int rt1, int rt2, int k) {
51     if(l == r) return l;
52     if(k <= tree[tree[rt1].ls].v - tree[tree[rt2].ls].v) return query(l, Mid, tree[rt1].ls, tree[rt2].ls, k);
53     else return query(Mid + 1, r, tree[rt1].rs, tree[rt2].rs, k - (tree[tree[rt1].ls].v - tree[tree[rt2].ls].v));
54 }
55 void Debug(int l, int r, int rt) {
56     printf("%d %d %d\n", l, r, tree[rt].v);
57     if(l == r) return ;
58     Debug(l, Mid, tree[rt].ls);
59     Debug(Mid + 1, r, tree[rt].rs);
60 }
61 signed main()
62 {
63     n = read(); m = read();
64     for(int i = 1; i <= n; i++) a[i] = b[i] = read();
65     sort(b + 1, b + 1 + n);
66     tb = unique(b + 1, b + 1 + n) - b - 1;
67     rt[0] = build(1, tb);
68     for(int i = 1; i <= n; i++) {
69         rt[i] = add(1, tb, lower_bound(b + 1, b + 1 + tb, a[i]) - b, rt[i - 1], 1);
70     }
71     for(int i = 1; i <= m; i++) {
72         int l, r, k;
73         l = read(); r = read(); k = read();
74         assert(r - l + 1 >= k);
75         printf("%d\n", b[query(1, tb, rt[r], rt[l - 1], k)]);
76     }
77     return 0;
78 }

```

## 数学

## 图论

### 树上问题

#### 树的直径

模板: POJ - 1985

- 两遍 DFS

```

1 void dfs(int x, int fa) {
2     for(int i = 0; i < E[x].size(); i++) {
3         int y = E[x][i].ver;
4         int w = E[x][i].val;
5         if(y == fa) continue;
6         d[y] = d[x] + w;
7         if(d[y] > d[c]) c = y;
8         dfs(y, x);
9     }
10 }
11 signed main()
12 {
13     n = read();
14     for(int i = 1; i < n; i++) {
15         int x = read(), y = read(), w = read();
16         E[x].push_back((Edge) {y, w});
17         E[y].push_back((Edge) {x, w});
18     }

```

```

19     dfs(1, 0);
20     d[c] = 0;
21     dfs(c, 0);
22     printf("%d\n", d[c]);
23     return 0;
24 }

```

### ● 树形 DP

```

1  void dfs(int x, int fa) {
2      d1[x] = d2[x] = 0;
3      for(int i = 0; i < E[x].size(); i++) {
4          int y = E[x][i].ver;
5          int w = E[x][i].val;
6          if(y == fa) continue;
7          dfs(y, x);
8          int t = d1[y] + w;
9          if(t > d1[x]) {
10             d2[x] = d1[x];
11             d1[x] = t;
12         } else if(t > d2[x]) {
13             d2[x] = t;
14         }
15     }
16     d = max(d, d1[x] + d2[x]);
17 }
18 signed main()
19 {
20     n = read();
21     for(int i = 1; i < n; i++) {
22         int x = read(), y = read(), w = read();
23         E[x].push_back((Edge) {y, w});
24         E[y].push_back((Edge) {x, w});
25     }
26     dfs(1, 0);
27     printf("%d\n", d);
28     return 0;
29 }

```

## 求 LCA

### ● 树链剖分

```

1  namespace Tree {
2      int siz[N], mson[N], ltp[N], fa[N], dth[N];
3      vector<int> son[N];
4      void dfs1(int x, int pre) {
5          siz[x] = 1;
6          mson[x] = 0;
7          fa[x] = pre;
8          dth[x] = dth[pre] + 1;
9          for(auto y : son[x]) if(y != pre) {
10             dfs1(y, x);
11             if(mson[x] == 0 || siz[y] > siz[mson[x]]) mson[x] = y;
12         }
13     }
14     void dfs2(int x, int pre, int tp) {
15         ltp[x] = tp;
16         if(mson[x]) dfs2(mson[x], x, tp);
17         for(auto y : son[x]) if(y != pre && y != mson[x]) {
18             dfs2(y, x, y);
19         }
20     }
21     void init() {
22         dfs1(1, 0);
23         dfs2(1, 0, 1);
24     }
25     int LCA(int x, int y) {
26         while(ltp[x] != ltp[y]) {

```



```

27         if(dth[ltp[x]] > dth[ltp[y]]) x = fa[ltp[x]];
28         else y = fa[ltp[y]];
29     }
30     return dth[y] > dth[x] ? x : y;
31 }
32 }

```

#### ● 倍增

```

1 namespace Tree {
2     vector<int> son[N];
3     int root, fa[N][31], dth[N];
4     void dfs(int x, int pre) {
5         fa[x][0] = pre;
6         dth[x] = dth[pre] + 1;
7         for(int i = 1; i <= 30; i++)
8             fa[x][i] = fa[fa[x][i-1]][i-1];
9         for(auto y : son[x]) if(y != pre)
10             dfs(y, x);
11     }
12     void init() {
13         dfs(root, 0);
14     }
15     int LCA(int x, int y) {
16         if(dth[x] > dth[y]) swap(x, y);
17         for(int i = 30; ~i; i--)
18             if(dth[fa[y][i]] >= dth[x])
19                 y = fa[y][i];
20         if(x == y) return x;
21         for(int i = 30; ~i; i--)
22             if(fa[y][i] != fa[x][i]) {
23                 x = fa[x][i];
24                 y = fa[y][i];
25             }
26         return fa[x][0];
27     }
28 }

```

### 树上启发式合并

长春站的痛.jpg

- 先递归计算轻儿子的答案
- 计算重儿子的答案，并且保留重儿子的状态数组
- 把其他所有轻儿子的答案加到状态数组中，更新当前点的答案

```

1 void dfs1(int x, int pre) {
2     siz[x] = 1;
3     mson[x] = 0;
4     for(auto y : son[x]) if(y != pre) {
5         dfs1(y, x);
6         siz[x] += siz[y];
7         if(!mson[x] || siz[y] > siz[mson[x]]) mson[x] = y;
8     }
9 }
10 void add(int x, int pre, int v) {
11     cnt[col[x]] += v;
12     if(cnt[col[x]] > Mx) Mx = cnt[col[x]], sum = col[x];
13     else if(cnt[col[x]] == Mx) sum += col[x];
14     for(auto y : son[x]) {
15         if(y == pre || y == Son) continue;
16         add(y, x, v);
17     }
18 }
19 void dfs2(int x, int pre, int keep) {
20     for(auto y : son[x]) {
21         if(y == pre || y == mson[x]) continue;
22         dfs2(y, x, 0);
23     }
24     if(mson[x]) dfs2(mson[x], x, 1), Son = mson[x];
25     add(x, pre, 1); Son = 0;

```

```

26     ans[x] = sum;
27     if(!keep) add(x, pre, -1), sum = 0, Mx = 0;
28
29 }

```

## 图算法

### 第 k 短路

模板: HDU-6351

```

1  bool operator<(const node &a, const node &b) {
2      return a.f + a.g > b.f + b.g;
3  }
4  priority_queue<node> q;
5  signed main()
6  {
7      n = read(); m = read();
8      for(int i = 1; i <= m; i++) {
9          int x, y, w;
10         x = read(); y = read(); w = read();
11         E[x].push_back((Edge) {y, w});
12         re[y].push_back((Edge) {x, w});
13     }
14     s = read(); t = read(); k = read();
15     memset(dis, 0x3f, sizeof(dis)); dis[t] = 0;
16     q.push((node) {t, 0, 0});
17     while(q.size()) {
18         int x = q.top().x, d = q.top().f;
19         q.pop();
20         if(dis[x] < d) continue;
21         for(int i = 0; i < re[x].size(); i++) {
22             int y = re[x][i].y, w = re[x][i].w;
23             if(dis[y] > dis[x] + w) {
24                 dis[y] = dis[x] + w;
25                 q.push((node) {y, dis[y], 0});
26             }
27         }
28     }
29     for(int i = 1; i <= n; i++) cnt[i] = k;
30     cnt[s]++;
31     q.push((node) {s, 0, dis[s]});
32     while(q.size()) {
33         int x = q.top().x, f = q.top().f, g = q.top().g;
34         q.pop();
35         if(cnt[x] == 0) continue;
36         cnt[x]--;
37         if(x == t && cnt[x] == 0) {
38             printf("%lld\n", f);
39             return 0;
40         }
41         for(int i = 0; i < E[x].size(); i++) {
42             int y = E[x][i].y, w = E[x][i].w;
43             q.push((node) {y, f + w, dis[y]});
44         }
45     }
46     printf("-1\n");
47     return 0;
48 }

```

## 网络流

### Dinic 算法

```

1  const int inf = 0x3f3f3f3f;
2  int bfs() {
3      memset(d, 0, sizeof(int) * (t + 10)); d[s] = 1;
4      while(q.size()) q.pop(); q.push(s);
5      while(q.size()) {
6          int x = q.front(); q.pop();

```

```

7         for(int i = head[x]; i; i = nxt[i]) {
8             if(d[ver[i]]) continue;
9             if(edge[i] <= 0) continue;
10            d[ver[i]] = d[x] + 1;
11            q.push(ver[i]);
12        }
13    }
14    return d[t];
15 }
16 int dinic(int x, int flow) {
17     if(x == t) return flow;
18     int k, res = flow;
19     for(int i = head[x]; i && res; i = nxt[i]) {
20         if(d[ver[i]] != d[x] + 1 || edge[i] <= 0) continue;
21         k = dinic(ver[i], min(flow, edge[i]));
22         if(k == 0) d[ver[i]] = 0;
23         edge[i] -= k;
24         edge[i ^ 1] += k;
25         res -= k;
26     }
27     return flow - res;
28 }

```

## EK 算法费用流

```

1 //反向边 cost 为负数, 容量为 0
2 int SPFA() {
3     queue<int> q; q.push(s);
4     memset(dis, 0x3f, sizeof(dis)); dis[s] = 0;
5     memset(vis, 0, sizeof(vis)); vis[s] = 1;
6     q.push(s); flow[s] = 0x3f3f3f3f;
7     while(q.size()) {
8         int x = q.front();
9         vis[x] = 0; q.pop();
10        for(int i = head[x]; i; i = nxt[i]) {
11            if(edge[i] <= 0) continue;
12            if(dis[ver[i]] > dis[x] + cost[i]) {
13                dis[ver[i]] = dis[x] + cost[i];
14                pre[ver[i]] = i;
15                flow[ver[i]] = min(flow[x], edge[i]);
16                if(!vis[ver[i]]) {
17                    q.push(ver[i]);
18                    vis[ver[i]] = 1;
19                }
20            }
21        }
22    }
23    return dis[t] != 0x3f3f3f3f;
24 }
25 void update() {
26     int x = t;
27     while(x != s) {
28         int i = pre[x];
29         edge[i] -= flow[t];
30         edge[i ^ 1] += flow[t];
31         x = ver[i ^ 1];
32     }
33     maxflow += flow[t];
34     minncost += dis[t] * flow[t];
35 }

```

## 连通性算法

### Tarjan 强连通分量

```

1 void tarjan(int x) {
2     low[x] = dfn[x] = ++dfncnt;
3     s[++t] = x; vis[x] = 1;
4     for(int i = head[x]; i; i = nxt[i]) {
5         if(!dfn[ver[i]]) {

```

```

6         tarjan(ver[i]);
7         low[x] = min(low[x], low[ver[i]]);
8     } else if(vis[x]) {
9         low[x] = min(low[x], dfn[ver[i]]);
10    }
11 }
12 if(dfn[x] == low[x]) {
13     int pp = -1;
14     ++sc;
15     while(pp != x) {
16         scc[s[t]] = sc;
17         siz[sc]++;
18         vis[s[t]] = 0;
19         pp = s[t];
20         t--;
21     }
22 }
23 }
24 //从任意点开始跑，但是注意如果图不连通，需要
25 for(int i = 1; i <= n; i++)
26     if(!dfn[i])
27         tarjan(i);

```

## 计算几何

## 字符串

### 字符串哈希

```

1 namespace String {
2     const int x = 135;
3     const int p1 = 1e9 + 7, p2 = 1e9 + 9;
4     ull xp1[N], xp2[N], xp[N];
5     void init_xp() {
6         xp1[0] = xp2[0] = xp[0] = 1;
7         for(int i = 1; i < N; i++) {
8             xp1[i] = xp1[i - 1] * x % p1;
9             xp2[i] = xp2[i - 1] * x % p2;
10            xp[i] = xp[i - 1] * x;
11        }
12    }
13    struct HashString {
14        char s[N];
15        int length, subsize;
16        bool sorted;
17        ull h[N], hl[N];
18        ull init(const char *t) {
19            if(xp[0] != 1) init_xp();
20            length = strlen(t);
21            strcpy(s, t);
22            ull res1 = 0, res2 = 0;
23            h[length] = 0;
24            for(int j = length - 1; j >= 0; j--) {
25                #ifdef ENABLE_DOUBLE_HASH
26                    res1 = (res1 * x + s[j]) % p1;
27                    res2 = (res2 * x + s[j]) % p2;
28                    h[j] = (res1 << 32) | res2;
29                #else
30                    res1 = res1 * x + s[j];
31                    h[j] = res1;
32                #endif
33            }
34            return h[0];
35        }
36        //获取子串哈希，左闭右开
37        ull get_substring_hash(int left, int right) {
38            int len = right - left;
39            #ifdef ENABLE_DOUBLE_HASH
40                unsigned int mask32 = ~(0u);

```

```

41     ull left1 = h[left] >> 32, right1 = h[right] >> 32;
42     ull left2 = h[left] & mask32, right2 = h[right] & mask32;
43     return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
44            (((left2 - right2 * xp2[len] % p2 + p2) % p2));
45 #else
46     return h[left] - h[right] * xp[len];
47 #endif
48 }
49 void get_all_subs_hash(int sublen) {
50     subsize = length - sublen + 1;
51     for (int i = 0; i < subsize; ++i)
52         hl[i] = get_substring_hash(i, i + sublen);
53     sorted = 0;
54 }
55
56 void sort_substring_hash() {
57     sort(hl, hl + subsize);
58     sorted = 1;
59 }
60
61 bool match(ull key) const {
62     if (!sorted) assert (0);
63     if (!subsize) return false;
64     return binary_search(hl, hl + subsize, key);
65 }
66 };
67 }

```

## Trie

```

1 namespace trie {
2     int t[N][26], sz, ed[N];
3     int _new() {
4         sz++;
5         memset(t[sz], 0, sizeof(t[sz]));
6         return sz;
7     }
8     void init() {
9         sz = 0;
10        _new();
11        memset(ed, 0, sizeof(ed));
12    }
13    void Insert(char *s, int n) {
14        int u = 1;
15        for (int i = 0; i < n; i++) {
16            int c = s[i] - 'a';
17            if (!t[u][c]) t[u][c] = _new();
18            u = t[u][c];
19        }
20        ed[u]++;
21    }
22    int find(char *s, int n) {
23        int u = 1;
24        for (int i = 0; i < n; i++) {
25            int c = s[i] - 'a';
26            if (!t[u][c]) return -1;
27            u = t[u][c];
28        }
29        return u;
30    }
31 }

```

## KMP 算法

```

1 namespace KMP {
2     void get_next(char *t, int m, int *nxt) {
3         int j = nxt[0] = 0;
4         for (int i = 1; i < m; i++) {
5             while (j && t[i] != t[j]) j = nxt[j - 1];
6             nxt[i] = j += (t[i] == t[j]);

```

```

7     }
8 }
9 vector<int> find(char *t, int m, int *nxt, char *s, int n) {
10     vector<int> ans;
11     int j = 0;
12     for(int i = 0; i < n; i++) {
13         while(j && s[i] != t[j]) j = nxt[j - 1];
14         j += s[i] == t[j];
15         if(j == m) {
16             ans.push_back(i - m + 1);
17             j = nxt[j - 1];
18         }
19     }
20     return ans;
21 }
22 }

```

## manacher 算法

```

1 namespace manacher {
2     char s[N];
3     int p[N], len;
4     void getp(string tmp) {
5         len = 0;
6         for(auto x : tmp) {
7             s[len++] = '#';
8             s[len++] = x;
9         }
10        s[len++] = '#';
11        memset(p, 0, sizeof(int) * (len + 10));
12        int c = 0, r = 0;
13        for(int i = 0; i < len; i++) {
14            if(i <= r) p[i] = min(p[2 * c - i], r - i);
15            else p[i] = 1;
16            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
17                p[i]++;
18            if(i + p[i] - 1 > r) {
19                r = i + p[i] - 1;
20                c = i;
21            }
22        }
23        for(int i = 0; i < len; i++) p[i]--;
24    }
25    void getp(char *tmp, int n) {
26        len = 0;
27        for(int i = 0; i < n; i++) {
28            s[len++] = '#';
29            s[len++] = tmp[i];
30        }
31        s[len++] = '#';
32        memset(p, 0, sizeof(int) * (len + 10));
33        int c = 0, r = 0;
34        for(int i = 0; i < len; i++) {
35            if(i <= r) p[i] = min(p[2 * c - i], r - i);
36            else p[i] = 1;
37            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
38                p[i]++;
39            if(i + p[i] - 1 > r) {
40                r = i + p[i] - 1;
41                c = i;
42            }
43        }
44        for(int i = 0; i < len; i++) p[i]--;
45    }
46    int getlen() {
47        return *max_element(p, p + len);
48    }
49    int getlen(string s) {
50        getp(s);
51        return getlen();
52    }

```

```
53 }
```

## AC 自动机

```
1 struct ac_automaton {
2     int t[N][26], danger[N], tot, fail[N];
3     int dp[N][N];
4     void init() {
5         tot = -1;
6         _new();
7     }
8     int _new() {
9         tot++;
10        memset(t[tot], 0, sizeof(t[tot]));
11        danger[tot] = 0;
12        fail[tot] = 0;
13        return tot;
14    }
15    void Insert(const char *s) {
16        int u = 0;
17        for(int i = 0; s[i]; i++) {
18            if(!t[u][mp[s[i]]]) t[u][s[i] - 'a'] = _new();
19            u = t[u][mp[s[i]]];
20        }
21        danger[u] = 1;
22    }
23    void build() {
24        queue<int> q;
25        for(int i = 0; i < 26; i++) {
26            if(t[0][i]) {
27                fail[i] = 0;
28                q.push(t[0][i]);
29            }
30        }
31        while(q.size()) {
32            int u = q.front(); q.pop();
33            danger[u] |= danger[fail[u]];
34            for(int i = 0; i < 26; i++) {
35                if(t[u][i]) {
36                    fail[t[u][i]] = t[fail[u]][i];
37                    q.push(t[u][i]);
38                } else t[u][i] = t[fail[u]][i];
39            }
40        }
41    }
42    int query(const char *s) {
43        memset(dp, 0x3f, sizeof(dp));
44        int n = strlen(s);
45        dp[0][0] = 0;
46        for(int i = 0; i < n; i++) {
47            for(int j = 0; j <= tot; j++) if(!danger[j]) {
48                for(int k = 0; k < 26; k++) if(!danger[t[j][k]]) {
49                    dp[i + 1][t[j][k]] = min(dp[i + 1][t[j][k]], dp[i][j] + (s[i] - 'a' != k));
50                }
51            }
52        }
53        int ans = 0x3f3f3f3f;
54        for(int i = 0; i <= tot; i++) if(!danger[i]) {
55            ans = min(ans, dp[n][i]);
56        }
57        return ans == 0x3f3f3f3f ? -1 : ans;
58    }
59 };
```

## 杂项

### int128

```
1  typedef __uint128_t u128;
2  inline u128 read() {
3      static char buf[100];
4      scanf("%s", buf);
5      // std::cin >> buf;
6      u128 res = 0;
7      for(int i = 0; buf[i]; ++i) {
8          res = res << 4 | (buf[i] <= '9' ? buf[i] - '0' : buf[i] - 'a' + 10);
9      }
10     return res;
11 }
12 inline void output(u128 res) {
13     if(res >= 16)
14         output(res / 16);
15     putchar(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
16     //std::cout.put(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
17 }
```

### tips:

- 如果使用 sort 比较两个函数，不能出现  $a < b$  和  $a > b$  同时为真的情况，否则会运行错误。
- 多组数据清空线段树的时候，不要忘记清空全部数组（比如说 lazytag 数组）。
- 注意树的深度和节点到根的距离是两个不同的东西，深度是点数，距离是边长，如果求 LCA 时用距离算会出错。