

Contents

1	Basic	1
1.1	PyMath	1
2	Tree	1
2.1	SegmentTree	1
2.2	HLD	1
2.3	Trie	2
3	String	2
3.1	KMP	2
4	Algorithm	3
4.1	LCA	3
4.2	MST	3
4.3	SG	3
4.4	Convex	3

1 Basic

1.1 PyMath

```

1 import math
2
3 math.ceil(x) #上高斯
4 math.floor(x) #下高斯
5 math.factorial(x) #階乘
6 math.fabs(x) #絕對值
7 math.fsum(arr) #求和
8 math.gcd(x, y)
9 math.exp(x) # e^x
10 math.log(x, base)
11 math.log2(x)
12 math.log10(x)
13 math.sqrt(x)
14 math.pow(x, y, mod)
15 math.sin(x) # cos, tan, asin, acos, atan,
    atan2, sinh ...
16 math.hypot(x, y) #歐幾里德範數
17 math.degrees(x) #x從弧度轉角度
18 math.radians(x) #x從角度轉弧度
19 math.gamma(x) #x的gamma函數
20 math.pi #const
21 math.e #const
22 math.inf

```

2 Tree

2.1 SegmentTree

```

1 #define lc (id << 1)
2 #define rc ((id << 1) | 1)
3
4 struct LazyTag{
5     // type 0 : increase val
6     // type 1 : set to val
7     // type 1 can overwrite type 0
8     int type ;
9     ll val ;
10 } ;
11
12 struct Node{
13     LazyTag tag ;
14     ll sum ;
15     int sz ;
16 }seg[Maxn << 2] ;
17
18 class SegmentTree{
19 private:
20     void pull(int id){
21         seg[id].sum = seg[lc].sum +
            seg[rc].sum ;
22     }
23
24     void AddTag(int id, LazyTag &tag){
25         if(tag.type == 0){
26             seg[id].sum += tag.val *
                seg[id].sz ;
27             seg[id].tag.val += tag.val ;
28         }
29         else{
30             seg[id].sum = tag.val *
                seg[id].sz ;
31             seg[id].tag = {1, tag.val} ;
32         }
33     }
34
35     void push(int id){
36         AddTag(lc, seg[id].tag) ;
37         AddTag(rc, seg[id].tag) ;
38         seg[id].tag = {0, 0} ;
39     }
40

```

```

41 public:
42     void build(int L=1, int R=n, int id=1){
43         seg[id].sum = 0 ;
44         seg[id].tag = {0, 0} ;
45         seg[id].sz = 1 ;
46
47         if(L == R){
48             seg[id].sum = arr[L] ;
49             return ;
50         }
51
52         int M = (L + R) >> 1 ;
53         build(L, M, lc) ;
54         build(M+1, R, rc) ;
55
56         pull(id) ;
57         seg[id].sz = seg[lc].sz + seg[rc].sz ;
58     }
59
60     void modify(int l, int r, LazyTag &tag,
        int L=1, int R=n, int id=1){
61         if(l <= L && R <= r){
62             AddTag(id, tag) ;
63             return ;
64         }
65
66         push(id) ;
67         int M = (L + R) >> 1 ;
68         if(r <= M) modify(l, r, tag, L, M,
            lc) ;
69         else if(l > M) modify(l, r, tag, M+1,
            R, rc) ;
70         else{
71             modify(l, r, tag, L, M, lc) ;
72             modify(l, r, tag, M+1, R, rc) ;
73         }
74         pull(id) ;
75     }
76
77     ll query(int l, int r, int L=1, int R=n,
        int id=1){
78         if(l <= L && R <= r) return
            seg[id].sum ;
79
80         push(id) ;
81         int M = (L + R) >> 1 ;
82         if(r <= M) return query(l, r, L, M,
            lc) ;
83         else if(l > M) return query(l, r,
            M+1, R, rc) ;
84         else return query(l, r, L, M, lc) +
            query(l, r, M+1, R, rc) ;
85     }
86 }tree ;

```

2.2 HLD

```

1 /* HLD */
2 int fa[Maxn], top[Maxn], son[Maxn],
    sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
    rnk[Maxn], dfscnt = 0 ;
3
4 void dfs1(int u, int from){
5     fa[u] = from ;
6     dep[u] = dep[from] + 1 ;
7     sz[u] = 1 ;
8
9     for ( auto v : g[u] ) if(v != from){
10         dfs1(v, u) ;
11         sz[u] += sz[v] ;
12         if(son[u] == -1 || sz[v] > sz[son[u]])
            son[u] = v ;
13     }
14 }
15
16 void dfs2(int u, int t){
17     top[u] = t ;

```

```

18 dfn[u] = ++dfscnt ;
19 rnk[dfscnt] = u ;
20
21 if(son[u] == -1) return ;
22
23 dfs2(son[u], t) ;
24
25 for ( auto v : g[u] ) if(v != fa[u] && v
26     != son[u]){
27     dfs2(v, v) ;
28 }
29
30 /* Segment Tree */
31 #define lc (id << 1)
32 #define rc ((id << 1) | 1)
33
34 struct ColorSeg{
35     int left, right, tot ;
36
37     ColorSeg operator+(const ColorSeg &o)
38     const {
39         if(tot == 0) return o ;
40         if(o.tot == 0) return *this ;
41
42         ColorSeg tmp ;
43         tmp.left = left ;
44         tmp.right = o.right ;
45         tmp.tot = tot + o.tot - (right ==
46             o.left) ;
47
48         return tmp ;
49 } ;
50
51 struct Node{
52     ColorSeg color ;
53     int tag ;
54 }seg[Maxn << 2] ;
55
56 class SegmentTree{
57 private:
58     void pull(int id){
59         // normal pull
60     }
61
62     void AddTag(int id, int tag){
63         // normal AddTag
64     }
65
66     void push(int id){
67         // normal push
68     }
69
70     void modify(int l, int r, int tag, int
71         L=1, int R=n, int id=1){
72         // normal modify
73     }
74
75     ColorSeg query(int l, int r, int L=1, int
76         R=n, int id=1){
77         // normal query
78     }
79 public:
80     void build(int L=1, int R=n, int id=1){
81         // normal build
82     }
83
84     // update val from u to v (simple path)
85     void update(int u, int v, int val){
86         while(top[u] != top[v]){
87             if(dep[top[u]] < dep[top[v]]) swap(u,
88                 v) ;
89             modify(dfn[top[u]], dfn[u], val) ;
90             u = fa[top[u]] ;
91         }
92         if(dep[u] < dep[v]) swap(u, v) ;
93     }
94
95     modify(dfn[v], dfn[u], val) ;
96 }
97
98 // get sum from u to v (simple path)
99 int get(int u, int v){
100     pair<int, ColorSeg> U, V ;
101     ColorSeg M ;
102     U = {u, {0, 0, 0}} ;
103     V = {v, {0, 0, 0}} ;
104
105     while(top[U.first] != top[V.first]){
106         if(dep[top[U.first]] <
107             dep[top[V.first]]) swap(U, V) ;
108         U.second = query(dfn[top[U.first]],
109             dfn[U.first]) + U.second ;
110         U.first = fa[top[U.first]] ;
111     }
112     if(dep[U.first] < dep[V.first]) swap(U,
113         V) ;
114     M = query(dfn[V.first], dfn[U.first]) ;
115     return (U.second.tot + V.second.tot +
116         M.tot) - (U.second.left == M.right)
117         - (V.second.left == M.left) ;
118 }
119
120 void init(){
121     memset(son, -1, sizeof(son)) ;
122 }

```

2.3 Trie

```

1 class TrieNode{
2 public:
3     set<int> end ;
4     TrieNode *next[26] ;
5
6     TrieNode(){
7         for ( int i=0 ; i<26 ; i++ ) next[i]
8             = nullptr ;
9     }
10 };
11
12 class Trie{
13 private:
14     int cnt ;
15     TrieNode *root ;
16 public:
17     Trie() : cnt(0) {
18         root = new TrieNode() ;
19     }
20
21     void insert(string &str, int n){
22         TrieNode* node = root ;
23         for ( auto s : str ){
24             int path = s - 'a' ;
25
26             if(node->next[path] == nullptr)
27                 node->next[path] = new
28                     TrieNode() ;
29             node = node->next[path] ;
30         }
31         node->end.insert(n) ;
32
33     void search(string &str){
34         TrieNode* node = root ;
35         for ( auto s : str ){
36             int path = s - 'a' ;
37             if(node->next[path] == nullptr)
38                 return ;
39             node = node->next[path] ;
40         }

```

```

39     int flg = 0 ;
40     for ( auto n : node->end ){
41         if(flg) cout << " " ;
42         else flg = 1 ;
43
44         cout << n ;
45     }
46 }
47
48 void clear(TrieNode* node) {
49     if (!node) return ;
50     for (int i = 0; i < 26; i++) {
51         if (node->next[i]) {
52             clear(node->next[i]) ;
53         }
54     }
55     delete node ;
56 }
57
58 ~Trie(){
59     clear(root) ;
60 }
61 };

```

3 String

3.1 KMP

```

1 int Next[N] ;
2 void kmp(string &str){
3     Next[0] = -1 ;
4     if(str.size() <= 1) return ;
5     Next[1] = 0 ;
6
7     int cur = 2, check = 0 ;
8
9     while(cur < str.size()){
10         if(str[cur - 1] == str[check])
11             Next[cur++] = ++check ;
12         else if(check > 0) check =
13             Next[check] ;
14         else Next[cur++] = 0 ;
15     }
16
17 int main(){
18     ios::sync_with_stdio(false) ;
19     cin.tie(nullptr) ;
20     cout.tie(nullptr) ;
21
22     string s1, s2 ;
23     while(cin >> s1){
24         s2 = s1 ;
25         reverse(s2.begin(), s2.end()) ;
26         kmp(s2) ;
27
28         int x=0, y=0 ;
29         while(x < s1.size() && y < s2.size()){
30             if(s1[x] == s2[y]){
31                 x++ ;
32                 y++ ;
33             }
34             else if(y > 0) y = Next[y] ;
35             else x++ ;
36         }
37
38         cout << s1 << s2.substr(y) << endl ;
39     }
40
41     return 0 ;
42 }

```

4 Algorithm

4.1 LCA

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 const int Maxn = 500005 ;
6
7 vector<int> e[Maxn] ;
8 int depth[Maxn] ;
9 int up[Maxn][40] ;
10 int MaxLog ;
11
12 void dfs(int u, int from, int d){
13     up[u][0] = from ;
14     depth[u] = d ;
15
16     for ( int i=1 ; i<=MaxLog ; i++ ){
17         up[u][i] = up[up[u][i-1]][i-1] ;
18     }
19
20     for ( auto v : e[u] ){
21         if(v == from) continue ;
22         dfs(v, u, d + 1) ;
23     }
24 }
25
26 int lca(int u, int v){
27     if(depth[u] < depth[v]) swap(u, v) ;
28
29     for ( int i=MaxLog ; i>=0 ; i-- )
30         if(depth[u] - (1 << i) >= depth[v]){
31             u = up[u][i] ;
32         }
33     if(u == v) return u ;
34
35     for ( int i=MaxLog ; i>=0 ; i-- )
36         if(up[u][i] != up[v][i]){
37             u = up[u][i] ;
38             v = up[v][i] ;
39         }
40     return up[u][0] ;
41 }
42
43 int main(){
44     int n, q, root ;
45     scanf("%d%d%d", &n, &q, &root) ;
46     MaxLog = __lg(n) ;
47
48     for ( int i=0 ; i<n-1 ; i++ ){
49         int u, v ;
50         scanf("%d%d", &u, &v) ;
51         e[u].push_back(v) ;
52         e[v].push_back(u) ;
53     }
54
55     dfs(root, root, 0) ;
56
57     while(q--){
58         int u, v ;
59         scanf("%d%d", &u, &v) ;
60         printf("%d\n", lca(u, v)) ;
61     }
62
63     return 0 ;
64 }

```

4.2 MST

```

1 struct Edge{
2     int u, v, w ;
3     // 這是最大生成樹，最小生成樹要改成 w < o.w

```

```

4     bool operator>(const Edge &o) const
5     {return w > o.w ;} ;
6
7     int par[N] ;
8     int sz[N] ;
9     int sum ;
10
11     vector<Edge> edge ;
12
13     void init(){
14         edge.clear() ;
15         for ( int i=0 ; i<N ; i++ ){
16             par[i] = i ;
17             sz[i] = 1 ;
18         }
19         sum = 0 ;
20     }
21
22     int find(int x){
23         if(x == par[x]) return x ;
24         return par[x] = find(par[x]) ;
25     }
26
27     int merge(int x, int y){
28         x = find(x) ;
29         y = find(y) ;
30
31         if(x == y) return 0 ;
32         if(sz[x] > sz[y]) swap(x, y) ;
33         par[x] = y ;
34         sz[y] += sz[x] ;
35
36         return 1 ;
37     }
38
39     void MST(){
40         int cnt = 0 ;
41         for ( int i=0 ; i<edge.size() && cnt < n-1
42             ; i++ ){
43             auto [u, v, w] = edge[i] ;
44             if(merge(u, v)){
45                 cnt++ ;
46                 sum += w ;
47             }
48         }
49     }
50
51     int main(){
52         for ( int i=0 ; i<m ; i++ ){
53             scanf("%d%d%d", &u, &v, &w) ;
54             edge.push_back({u, v, w}) ;
55             sum += w ;
56         }
57
58         sort(edge.begin(), edge.end(),
59             greater<Edge>()) ;
60         MST() ;
61     }

```

4.3 SG

```

1 long long SG(long long k){
2
3     if(k % 2 == 0){
4         return k / 2 ;
5     }
6     else{
7         return SG(k / 2) ;
8     }
9 }
10
11
12 int main(){
13     int cas, n ;
14
15     scanf("%d", &cas) ;

```

```

16     while(cas--){
17         scanf("%d", &n) ;
18
19         long long s, v = 0 ;
20
21         for(int i = 0; i < n; i++){
22             scanf("%lld", &s) ;
23             v ^= SG(s) ; //XOR
24         }
25
26         if(v) printf("YES\n") ;
27         else printf("NO\n") ;
28     }
29 }
30
31 int SG[30] ;
32 int vis[Maxn], stone[Maxn] ;
33
34 void build(){
35     SG[0] = 0 ;
36     memset(vis, 0, sizeof(vis)) ;
37
38     for ( int i=1 ; i<30 ; i++ ){
39         int cur = 0 ;
40         for ( int j=0 ; j<i ; j++ ) for ( int
41             k=0 ; k<=j ; k++ ){
42             vis[SG[j] ^ SG[k]] = i ;
43         }
44         while(vis[cur] == i) cur++ ;
45         SG[i] = cur ;
46     }
47 }
48
49 int main(){
50     build() ;
51
52     int T = 0 ;
53     while(~scanf("%d", &n) && n){
54         int ans = 0 ;
55
56         for ( int i=1 ; i<=n ; i++ ) scanf("%d",
57             &stone[i]) ;
58
59         for ( int i=1 ; i<=n ; i++ ) if(stone[i]
60             & 1){
61             ans ^= SG[n-i] ;
62         }
63     }
64 }

```

4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a,
5         const Coordinate& b){
6         if(a.x == b.x) return a.y < b.y ;
7         return a.x < b.x ;
8     }
9
10     friend bool operator==(const Coordinate&
11         a, const Coordinate& b){
12         return a.x == b.x && a.y == b.y ;
13     }
14 } ;
15
16 vector<Coordinate> nodes ;
17
18 long long cross(const Coordinate& o, const
19     Coordinate& a, const Coordinate& b){
20     return (a.x - o.x) * (b.y - o.y) - (a.y -
21         o.y) * (b.x - o.x) ;
22 }
23
24 void input(){
25     nodes.clear() ;

```

```
22
23     int n, x, y ;
24     char c ;
25     cin >> n ;
26
27     for ( int i=0 ; i<n ; i++ ){
28         cin >> x >> y >> c ;
29         if(c == 'Y') nodes.push_back({x, y}) ;
30     }
31 }
32
33 void monotone(){
34     sort(nodes.begin(), nodes.end()) ;
35
36     int n = unique(nodes.begin(), nodes.end())
37         - nodes.begin() ;
38
39     vector<Coordinate> ch(n+1) ;
40
41     int m = 0 ;
42
43     for ( int i=0 ; i<n ; i++ ){
44         while(m > 1 && cross(ch[m-2], ch[m-1],
45             nodes[i]) < 0) m-- ;
46         ch[m++] = nodes[i] ;
47     }
48     for ( int i=n-2, t=m ; i>=0 ; i-- ){
49         while(m > t && cross(ch[m-2], ch[m-1],
50             nodes[i]) < 0) m-- ;
51         ch[m++] = nodes[i] ;
52     }
53
54     if(n > 1) m-- ;
55     cout << m << endl ;
56
57     for ( int i=0 ; i<m ; i++ ) cout <<
58         ch[i].x << " " << ch[i].y << endl ;
59 }
```