

# Contents

|     |                                 |   |
|-----|---------------------------------|---|
| 1   | Basic                           | 1 |
| 1.1 | PyMath                          | 1 |
| 2   | Tree                            | 1 |
| 2.1 | SegmentTree                     | 1 |
| 2.2 | HLD                             | 1 |
| 2.3 | PST                             | 2 |
| 2.4 | Trie                            | 2 |
| 3   | Graph                           | 3 |
| 3.1 | articulation points AND bridges | 3 |
| 3.2 | SCC - Tarjan                    | 3 |
| 3.3 | BCC - Tarjan                    | 3 |
| 4   | String                          | 3 |
| 4.1 | KMP                             | 3 |
| 4.2 | ACAM                            | 4 |
| 5   | Algorithm                       | 4 |
| 5.1 | LCA                             | 4 |
| 5.2 | MST                             | 4 |
| 5.3 | SG                              | 5 |
| 5.4 | Convex                          | 5 |
| 6   | DP                              | 5 |
| 6.1 | 輪廓線 DP                          | 5 |
| 6.2 | 數位 DP                           | 5 |
| 6.3 | 樹 DP                            | 6 |

# 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         modify(dfn[v], dfn[u], val) ;
95     }
96
97     int get(int u, int v){
98         pair<int, ColorSeg> U, V ;
99         ColorSeg M ;
100         U = {u, {0, 0, 0}} ;
101         V = {v, {0, 0, 0}} ;
102
103         while(top[U.first] != top[V.first]){
104             if(dep[top[U.first]] <
105                 dep[top[V.first]]) swap(U, V) ;
106             U.second = query(dfn[top[U.first]],
107                 dfn[U.first]) + U.second ;
108             U.first = fa[top[U.first]] ;
109         }
110         if(dep[U.first] < dep[V.first]) swap(U,
111             V) ;
112         M = query(dfn[V.first], dfn[U.first]) ;
113         return (U.second.tot + V.second.tot +
114             M.tot) - (U.second.left == M.left) ;
115     }
116 }tree ;
117
118 void init(){
119     memset(son, -1, sizeof(son)) ;
120 }

```

## 2.3 PST

```

1 // Find the k-th largest number within a
2 // given range
3 struct Node{
4     int sum, left, right ;
5 }seg[Maxn + 20 * Maxn] ;
6
7 class PersistentSegmentTree{
8 private:
9     int n ;
10    int cnt ;
11    vector<int> version ;
12
13    int build(int L, int R){
14        int cur_cnt = cnt++ ;
15        if(L == R){
16            seg[cur_cnt] = {0, 0, 0} ;
17            return cur_cnt ;
18        }
19
20        int M = (L + R) >> 1 ;
21        int lc = build(L, M) ;
22        int rc = build(M+1, R) ;
23
24        seg[cur_cnt] = {0, lc, rc} ;
25        return cur_cnt ;
26    }
27 public:
28    PersistentSegmentTree(int _n){
29        n = _n ;
30        cnt = 0 ;
31
32        int root = build(1, n) ;
33        version.push_back(root) ;
34    }
35
36    void update(int ver, int idx){
37        auto upd = [&](auto &&self, const int
38            cur, int L, int R){
39            if(L == R){

```

```

40        seg[cur_cnt] = {seg[cur].sum + 1, 0,
41            0} ;
42        return cur_cnt ;
43    }
44
45    int M = (L + R) >> 1 ;
46    int lc = seg[cur].left ;
47    int rc = seg[cur].right ;
48
49    if(idx <= M) lc = self(self,
50        seg[cur].left, L, M) ;
51    else rc = self(self, seg[cur].right,
52        M+1, R) ;
53
54    seg[cur_cnt] = {seg[lc].sum +
55        seg[rc].sum, lc, rc} ;
56
57    return cur_cnt ;
58 } ;
59
60 int root = upd(upd, version[ver], 1, n) ;
61 version.push_back(root) ;
62
63 int query(int verL, int verR, int k){
64     auto qry = [&](auto &&self, const int
65         cur_old, const int cur_new, int L,
66         int R){
67         if(L == R) return L ;
68
69         int old_l = seg[cur_old].left, old_r =
70             seg[cur_old].right ;
71         int new_l = seg[cur_new].left, new_r =
72             seg[cur_new].right ;
73
74         int dl = seg[new_l].sum -
75             seg[old_l].sum ;
76         int dr = seg[new_r].sum -
77             seg[old_r].sum ;
78
79         int M = (L + R) >> 1 ;
80
81         if(dl >= k) return self(self, old_l,
82             new_l, L, M) ;
83         k -= dl ;
84         return self(self, old_r, new_r, M+1,
85             R) ;
86     } ;
87
88     int idx = qry(qry, version[verL-1],
89         version[verR], 1, n) ;
90     return idx ;
91 }

```

## 2.4 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 void insert(string &str, int n){
21     TrieNode* node = root ;
22     for ( auto s : str ){
23         int path = s - 'a' ;
24
25         if(node->next[path] == nullptr)
26             node->next[path] = new
27             TrieNode() ;
28         node = node->next[path] ;
29     }
30
31 void search(string &str){
32     TrieNode* node = root ;
33     for ( auto s : str ){
34         int path = s - 'a' ;
35         if(node->next[path] == nullptr)
36             return ;
37         node = node->next[path] ;
38     }
39
40 int flg = 0 ;
41 for ( auto n : node->end ){
42     if(flg) cout << " " ;
43     else flg = 1 ;
44
45     cout << n ;
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 Graph

#### 3.1 articulation points AND bridges

```

1 int dfn[Maxn] = {-1}, low[Maxn] = {-1},
   dfsCnt ;
2
3 void dfs(int u, int fa){
4     dfn[u] = low[u] = ++dfsCnt ;
5     int child = 0 ;
6
7     for ( auto v : g[u] ) if(v != fa){
8         if(dfn[v] == -1){
9             child++ ;
10            dfs(v, u) ;
11            low[u] = min(low[u], low[v]) ;
12
13            if(low[v] >= dfn[u]){
14                // this edge is a bridge
15            }
16
17            if(u != fa && low[v] >= dfn[u]){
18                // this node v is a articulation point
19            }
20        }
21        else low[u] = min(low[u], dfn[v]) ;
22    }
23
24    if(u == fa && child > 1){

```

```

25 // this node u is a articulation point
26 }
27 }

```

#### 3.2 SCC - Tarjan

```

1 vector<int> scc[Maxn] ;
2 int dfn[Maxn], low[Maxn], sccId[Maxn],
   dfsCnt = 0, cnt_scc = 0 ;
3 stack<int> st ;
4 bitset<Maxn> inSt, vis ;
5
6 void dfs(int u, int from){
7     dfn[u] = low[u] = ++dfsCnt ;
8     st.push(u) ;
9     inSt[u] = 1 ;
10
11     for ( auto v : g[u] ){
12         if(!inSt[v] && dfn[v] != -1) continue ;
13         if(dfn[v] == -1) dfs(v, u) ;
14         low[u] = min(low[u], low[v]) ;
15     }
16
17     if(dfn[u] == low[u]){
18         cnt_scc++ ;
19         int x ;
20
21         do{
22             x = st.top() ;
23             st.pop() ;
24
25             inSt[x] = 0 ;
26             sccId[x] = cnt_scc ;
27             scc[cnt_scc].push_back(x) ;
28         }
29         while(x != u) ;
30     }
31 }
32
33 // SCC to DAG (after dfs)
34 vector<int> dag[Maxn] ;
35
36 void scc_to_dag(){
37     vector<int> dag[Maxn] ;
38     for ( int u=1 ; u<=n ; u++ ){
39         for ( auto v : g[u] ){
40             if(sccId[u] != sccId[v]){
41                 dag[sccId[u]].push_back(sccId[v]) ;
42             }
43         }
44     }
45 }
46
47 void init(){
48     memset(dfn, -1, sizeof(dfn)) ;
49     memset(low, -1, sizeof(low)) ;
50 }
51
52 int main(){
53     init() ;
54     input() ;
55     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
56         == -1){
57         dfs(i, i) ;
58     }

```

#### 3.3 BCC - Tarjan

```

1 struct Edge{
2     int v, next ;
3 }e[Maxm << 1] ;
4 int head[Maxm], tot = 1 ;
5

```

```

6 void add(int u, int v){
7     e[++tot] = {v, head[u]} ;
8     head[u] = tot ;
9     e[++tot] = {u, head[v]} ;
10    head[v] = tot ;
11 }
12
13 bitset<Maxm << 1> bz ;
14 vector<vector<int> > bcc ;
15 int dfn_cnt = 0, dfn[Maxn], low[Maxn],
   vis_bcc[Maxn], bcc_cnt = 0 ;
16
17 void dfs1(int u, int from){
18     dfn[u] = low[u] = ++dfn_cnt ;
19
20     for ( int i=head[u] ; i!=-1 ; i=e[i].next
21         ){
22         int v = e[i].v ;
23
24         if(dfn[v] == -1){
25             dfs1(v, i) ;
26             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
27                 1 ;
28             low[u] = min(low[u], low[v]) ;
29         }
30         else if(i != (from ^ 1)) low[u] =
31             min(low[u], dfn[v]) ;
32     }
33 }
34
35 void dfs2(int u, int id){
36     vis_bcc[u] = id ;
37     bcc[id].push_back(u) ;
38
39     for ( int i=head[u] ; i!=-1 ; i=e[i].next
40         ){
41         int v = e[i].v ;
42
43         if(vis_bcc[v] != -1 || bz[i]) continue ;
44         dfs2(v, id) ;
45     }
46 }
47
48 void init(){
49     memset(dfn, -1, sizeof(dfn)) ;
50     memset(head, -1, sizeof(head)) ;
51     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
52 }
53
54 int main(){
55     init() ;
56     input() ;
57     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
58         == -1){
59         dfs1(i, 0) ;
60     }
61
62     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
63         == -1){
64         bcc.push_back(vector<int>()) ;
65         dfs2(i, bcc_cnt++) ;
66     }
67 }

```

### 4 String

#### 4.1 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   int main(){
17       ios::sync_with_stdio(false) ;
18       cin.tie(nullptr) ;
19       cout.tie(nullptr) ;
20
21       string s1, s2 ;
22       while(cin >> s1){
23           s2 = s1 ;
24           reverse(s2.begin(), s2.end()) ;
25           kmp(s2) ;
26
27           int x=0, y=0 ;
28           while(x < s1.size() && y < s2.size()){
29               if(s1[x] == s2[y]){
30                   x++ ;
31                   y++ ;
32               }
33               else if(y > 0) y = Next[y] ;
34               else x++ ;
35           }
36           cout << s1 << s2.substr(y) << endl ;
37       }
38   }
39   return 0 ;
40 }

```

## 4.2 ACAM

```

1   class ACAutomation{
2   private:
3       vector<int> fail, end, order ;
4       vector<vector<int>> tree ;
5
6       int base, alpha ;
7
8       int new_node(){
9           tree.emplace_back(alpha, 0) ;
10          fail.push_back(0) ;
11
12          return tree.size() - 1 ;
13      }
14  public:
15      ACAutomation(int _base='a', int _alpha=26)
16          : base(_base), alpha(_alpha) {
17          clear() ;
18      }
19
20      void clear(){
21          fail.assign(1, 0) ;
22          order.clear() ;
23          end.clear() ;
24          tree.assign(1, vector<int>(alpha, 0)) ;
25      }
26
27      void add_pattern(const string &pattern){
28          int u = 0 ;
29          for ( auto ch : pattern ){
30              int v = ch - base ;
31
32              if(tree[u][v] == 0) tree[u][v] =
33                  new_node() ;
34              u = tree[u][v] ;
35          }
36          end.push_back(u) ;
37      }
38  }

```

```

39   void build(){
40       queue<int> q ;
41       order.clear() ;
42       order.push_back(0) ;
43
44       for ( int i=0 ; i<alpha ; i++ )
45           if(tree[0][i] > 0){
46               q.push(tree[0][i]) ;
47           }
48       while(!q.empty()){
49           int u = q.front() ; q.pop() ;
50           order.push_back(u) ;
51
52           for ( int i=0 ; i<alpha ; i++ ){
53               if(tree[u][i] == 0) tree[u][i] =
54                   tree[fail[u]][i] ;
55               else{
56                   fail[tree[u][i]] = tree[fail[u]][i] ;
57                   q.push(tree[u][i]) ;
58               }
59           }
60       }
61
62       vector<int> count_per_pattern(const string
63           &text) const {
64           int u = 0 ;
65           vector<int> vis(tree.size(), 0) ;
66
67           for ( char ch : text ){
68               u = tree[u][ch - base] ;
69               vis[u]++ ;
70           }
71
72           for ( int i=order.size()-1 ; i>=1 ; i-- )
73               vis[fail[i]] += vis[i] ;
74       }
75
76       vector<int> ans(end.size(), 0) ;
77       for ( int id=0 ; id<end.size() ; id++ ){
78           ans[id] = vis[end[id]] ;
79       }
80
81       return ans ;
82   }
83 };

```

## 5 Algorithm

### 5.1 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  }

```

### 5.2 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     sort(edge.begin(), edge.end(),
58           greater<Edge>()) ;
59     MST() ;
60 }

```

### 5.3 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++ ){

```

```

41         vis[SG[j] ^ SG[k]] = i ;
42     }
43     while(vis[cur] == i) cur++ ;
44     SG[i] = cur ;
45 }
46 }
47
48 int main(){
49     build() ;
50
51     int T = 0 ;
52     while(~scanf("%d", &n) && n){
53         int ans = 0 ;
54
55         for ( int i=1 ; i<=n ; i++ ) scanf("%d",
56               &stone[i]) ;
57
58         for ( int i=1 ; i<=n ; i++ ) if(stone[i]
59               & 1){
60             ans ^= SG[n-i] ;
61         }
62     }
63 }

```

### 5.4 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() ;
26
27     int n, x, y ;
28     char c ;
29     cin >> n ;
30
31     for ( int i=0 ; i<n ; i++ ){
32         cin >> x >> y >> c ;
33         if(c == 'Y') nodes.push_back({x, y}) ;
34     }
35 }
36
37 void monotone(){
38     sort(nodes.begin(), nodes.end()) ;
39
40     int n = unique(nodes.begin(), nodes.end())
41         - nodes.begin() ;
42
43     vector<Coordinate> ch(n+1) ;
44
45     int m = 0 ;
46
47     for ( int i=0 ; i<n ; i++ ){
48         while(m > 1 && cross(ch[m-2], ch[m-1],
49               nodes[i]) < 0) m-- ;
50         ch[m++] = nodes[i] ;
51     }
52 }

```

```

46 for ( int i=n-2, t=m ; i>=0 ; i-- ){
47     while(m > t && cross(ch[m-2], ch[m-1],
48           nodes[i]) < 0) m-- ;
49     ch[m++] = nodes[i] ;
50 }
51
52 if(n > 1) m-- ;
53 cout << m << endl ;
54
55 for ( int i=0 ; i<m ; i++ ) cout <<
56     ch[i].x << " " << ch[i].y << endl ;
57 }

```

## 6 DP

### 6.1 輪廓線 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4 using ll = long long ;
5
6 ll dp[2][1<<10 + 5] ;
7 int n, m ;
8 int cur ;
9
10 void update(int s1, int s2){
11     if(s2 & (1<<m)){
12         dp[cur][s2 ^ (1<<m)] += dp[cur ^
13             1][s1] ;
14     }
15 }
16
17 int main(){
18     while(~scanf("%d%d", &n, &m)){
19         if(m > n) swap(n, m) ;
20         memset(dp, 0, sizeof(dp)) ;
21         cur = 0 ;
22         dp[cur][(1<<m) - 1] = 1 ;
23         for ( int i=0 ; i<n ; i++ ) for ( int
24               j=0 ; j<m ; j++ ){
25             cur ^= 1 ;
26             memset(dp[cur], 0, sizeof(dp[cur])) ;
27
28             for ( int k=0 ; k<(1<<m) ; k++ ){
29                 update(k, k<<1) ; // not put
30                 if(i && !(k & (1<<(m-1))))
31                     update(k, (k<<1) | (1<<m) |
32                     1) ; // put up
33                 if(j && !(k & 1)) update(k, (k<<1)
34                     | 3) ; // put left
35             }
36         }
37         printf("%lld\n", dp[cur][(1<<m) - 1]) ;
38     }
39     return 0 ;
40 }

```

### 6.2 數位 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 int K ;
6 int dp[20][105][105][2] ;
7 vector<int> dig ;
8
9 int solve(int pos, int sum, int dsum, bool
10           lim){
11     if(pos == -1){
12         if(sum == 0 && dsum == 0) return 1 ;
13         return 0 ;
14     }
15 }

```

```

14 int &d = dp[pos][sum][dsum][lim] ;
15 if(d != -1) return d ;
16
17 int up = lim ? dig[pos] : 9 ;
18 int res = 0 ;
19 for ( int i=0 ; i<=up ; i++){
20     res += solve(pos-1, (sum * 10 + i) %
21         K, (dsum + i) % K, lim && i==up)
22         ;
23 }
24 return d = res ;
25 }
26
27 int count(int n){
28     memset(dp, -1, sizeof(dp)) ;
29     dig.clear() ;
30
31     while(n > 0){
32         dig.push_back(n % 10) ;
33         n /= 10 ;
34     }
35
36     return solve(dig.size() - 1, 0, 0, 1) ;
37 }
38
39 int main(){
40     int T ;
41     scanf("%d", &T) ;
42
43     int a, b ;
44     while(T--){
45         scanf("%d%d", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) -
48             count(a-1)) ;
49     }
50     return 0 ;
51 }

```

```

33 for ( int i=cnt[u] ; i>1 ; i-- ) for (
34     int j=1 ; j<i && j<=cnt[v] ; j++ ){
35     dp[u][i][1] = min(dp[u][i][1],
36         dp[u][i-j][1] + dp[v][j][1] + 2 *
37         w) ;
38     dp[u][i][0] = min(dp[u][i][0],
39         dp[u][i-j][1] + dp[v][j][0] + w) ;
40     dp[u][i][0] = min(dp[u][i][0],
41         dp[u][i-j][0] + dp[v][j][1] + 2 *
42         w) ;
43 }
44 }
45
46 int main(){
47     int t = 0 ;
48
49     while(~scanf("%d", &n) && n){
50         init() ;
51         for ( int i=0 ; i<n-1 ; i++ ){
52             int u, v, w ;
53             scanf("%d%d%d", &v, &u, &w) ;
54             edge[u].push_back({v, w}) ;
55         }
56
57         DFS(0) ;
58         printf("Case %d:\n", ++t) ;
59
60         int q, e ;
61         scanf("%d", &q) ;
62
63         while(q--){
64             scanf("%d", &e) ;
65
66             for ( int i=n ; i>=1 ; i-- )
67                 if(dp[0][i][0] <= e){
68                     printf("%d\n", i) ;
69                     break ;
70                 }
71         }
72     }
73
74     return 0 ;
75 }

```

### 6.3 樹 DP

```

1 #include <bits/stdc++.h>
2
3 #define N 505
4 #define INF 0x3f3f3f3f
5
6 using namespace std ;
7
8 struct Edge{
9     int v, w ;
10 } ;
11
12 vector<Edge> edge[N] ;
13 int n ;
14 int cnt[N] ;
15 int dp[N][N][2] ;
16
17 void init(){
18     for ( int i=0 ; i<N ; i++ )
19         edge[i].clear() ;
20     memset(cnt, 0, sizeof(cnt)) ;
21     memset(dp, INF, sizeof(dp)) ;
22 }
23
24 void DFS(int u){
25     cnt[u] = 1 ;
26     for ( auto [v, w] : edge[u] ){
27         DFS(v) ;
28         cnt[u] += cnt[v] ;
29     }
30
31     dp[u][1][0] = dp[u][1][1] = 0 ;
32
33     for ( auto [v, w] : edge[u] ){

```