

Contents

1	Basic	1
1.1	PyMath	1
2	Math	1
2.1	formula	1
2.2	extended gcd	1
3	Tree	1
3.1	SegmentTree	1
3.2	HLD	2
3.3	PST	2
4	Graph	3
4.1	cut vertex AND bridges	3
4.2	SCC - Tarjan	3
4.3	BCC - Tarjan	3
4.4	Convex	3
5	String	4
5.1	KMP	4
5.2	Trie	4
5.3	ACAM	4
6	Algorithm	5
6.1	LCA	5
6.2	MST	5
6.3	SG	5
6.4	Max Flow	5
6.5	min cut max flow	6
7	DP	6
7.1	輪廓線 DP	6
7.2	數位 DP	6
7.3	樹 DP	7

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 Math

2.1 formula

1. Catalan Number

$$C_n = \frac{1}{n} \binom{2n}{n}, C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

$$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \dots$$

2. Euler's Formula

對於 v 個點, e 條邊, f 個面, c 個連通分量

$$V + F = E + 2$$

$$V + F = E + C + 1$$

3. Pick's Theorem

點座標均是整數或是正方形格子點的簡單多邊形, 其面積 A

和內部點數量 i , 邊上格點數量 b 的關係為

$$A = i + \frac{b}{2} - 1$$

2.2 extended gcd

給定 a, b, c , 求 $ax + by = c$ 的解

```

1 ll extgcd(ll a, ll b, ll&c, ll&x, ll&y){
2     if(b == 0){
3         x = c/a ;
4         y = 0 ;
5         return a ;
6     }
7     ll d = extgcd(b, a%b, c, x, y), tmp =
8         x ;
9     x = y ;
10    y = tmp - (a/b)*y ;
11    return d ;

```

3 Tree

3.1 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 +
22             seg[rc].sum ;
23     }
24     void AddTag(int id, LazyTag &tag){
25         if(tag.type == 0){
26             seg[id].sum += tag.val *
27                 seg[id].sz ;
28             seg[id].tag.val += tag.val ;
29         }
30         else{
31             seg[id].sum = tag.val *
32                 seg[id].sz ;
33             seg[id].tag = {1, tag.val} ;
34         }
35     }
36     void push(int id){
37         AddTag(lc, seg[id].tag) ;
38         AddTag(rc, seg[id].tag) ;
39         seg[id].tag = {0, 0} ;
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     void modify(int l, int r, LazyTag &tag,
60         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,
69             lc) ;

```

```

69     else if(l > M) modify(l, r, tag, M+1,
70         R, rc) ;
71     else{
72         modify(l, r, tag, L, M, lc) ;
73         modify(l, r, tag, M+1, R, rc) ;
74     }
75     pull(id) ;
76 }
77 ll query(int l, int r, int L=1, int R=n,
78     int id=1){
79     if(l <= L && R <= r) return
80         seg[id].sum ;
81     push(id) ;
82     int M = (L + R) >> 1 ;
83     if(r <= M) return query(l, r, L, M,
84         lc) ;
85     else if(l > M) return query(l, r,
86         M+1, R, rc) ;
87     else return query(l, r, L, M, lc) +
88         query(l, r, M+1, R, rc) ;
89 }
90 }tree ;

```

3.2 HLD

```

1  /* HLD */
2  int fa[Maxn], top[Maxn], son[Maxn],
3      sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
4      rn timer, dfsct = 0 ;
5
6  void dfs1(int u, int from){
7      fa[u] = from ;
8      dep[u] = dep[from] + 1 ;
9      sz[u] = 1 ;
10
11      for ( auto v : g[u] ) if(v != from){
12          dfs1(v, u) ;
13          sz[u] += sz[v] ;
14          if(son[u] == -1 || sz[v] > sz[son[u]])
15              son[u] = v ;
16      }
17  }
18
19  void dfs2(int u, int t){
20      top[u] = t ;
21      dfn[u] = ++dfsct ;
22      rn[dfsct] = u ;
23
24      if(son[u] == -1) return ;
25      dfs2(son[u], t) ;
26
27      for ( auto v : g[u] ) if(v != fa[u] && v
28          != son[u]){
29          dfs2(v, v) ;
30      }
31  }
32
33  /* Segment Tree */
34  #define lc (id << 1)
35  #define rc ((id << 1) | 1)
36
37  struct ColorSeg{
38      int left, right, tot ;
39
40      ColorSeg operator+(const ColorSeg &o)
41          const {
42          if(tot == 0) return o ;
43          if(o.tot == 0) return *this ;
44
45          ColorSeg tmp ;
46          tmp.left = left ;
47          tmp.right = o.right ;
48          tmp.tot = tot + o.tot - (right ==
49              o.left) ;

```

```

45     return tmp ;
46 }
47 } ;
48 } ;
49
50 struct Node{
51     ColorSeg color ;
52     int tag ;
53 }seg[Maxn << 2] ;
54
55 class SegmentTree{
56 private:
57     void pull(int id){
58         // normal pull
59     }
60
61     void AddTag(int id, int tag){
62         // normal AddTag
63     }
64
65     void push(int id){
66         // normal push
67     }
68
69     void modify(int l, int r, int tag, int
70         L=1, int R=n, int id=1){
71         // normal modify
72     }
73
74     ColorSeg query(int l, int r, int L=1, int
75         R=n, int id=1){
76         // normal query
77     }
78 public:
79     void build(int L=1, int R=n, int id=1){
80         // normal build
81     }
82
83     // update val from u to v (simple path)
84     void update(int u, int v, int val){
85         while(top[u] != top[v]){
86             if(dep[top[u]] < dep[top[v]]) swap(u,
87                 v) ;
88             modify(dfn[top[u]], dfn[u], val) ;
89             u = fa[top[u]] ;
90         }
91
92         if(dep[u] < dep[v]) swap(u, v) ;
93         modify(dfn[u], dfn[v], val) ;
94     }
95
96     // get sum from u to v (simple path)
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
111         if(dep[U.first] < dep[V.first]) swap(U,
112             V) ;
113         M = query(dfn[V.first], dfn[U.first]) ;
114
115         return (U.second.tot + V.second.tot +
116             M.tot) - (U.second.left == M.right)
117             - (V.second.left == M.left) ;
118     }
119 }tree ;
120
121 void init(){

```

```

115     memset(son, -1, sizeof(son)) ;
116 }

```

3.3 PST

```

1  // Find range k-th largest number
2  struct Node{
3      int sum, left, right ;
4  }seg[Maxn + 20 * Maxn] ;
5
6  class PersistentSegmentTree{
7 private:
8     int n ;
9     int cnt ;
10     vector<int> version ;
11
12     int build(int L, int R){
13         int cur_cnt = cnt++ ;
14         if(L == R){
15             seg[cur_cnt] = {0, 0, 0} ;
16             return cur_cnt ;
17         }
18
19         int M = (L + R) >> 1 ;
20         int lc = build(L, M) ;
21         int rc = build(M+1, R) ;
22
23         seg[cur_cnt] = {0, lc, rc} ;
24         return cur_cnt ;
25     }
26 public:
27     PersistentSegmentTree(int _n){
28         n = _n ;
29         cnt = 0 ;
30
31         int root = build(1, n) ;
32         version.push_back(root) ;
33     }
34
35     void update(int ver, int idx){
36         auto upd = [&](auto &&self, const int
37             cur, int L, int R){
38             int cur_cnt = cnt++ ;
39
40             if(L == R){
41                 seg[cur_cnt] = {seg[cur].sum + 1, 0,
42                     0} ;
43                 return cur_cnt ;
44             }
45
46             int M = (L + R) >> 1 ;
47             int lc = seg[cur].left ;
48             int rc = seg[cur].right ;
49
50             if(idx <= M) lc = self(self,
51                 seg[cur].left, L, M) ;
52             else rc = self(self, seg[cur].right,
53                 M+1, R) ;
54
55             seg[cur_cnt] = {seg[lc].sum +
56                 seg[rc].sum, lc, rc} ;
57
58             return cur_cnt ;
59         };
60
61         int root = upd(upd, version[ver], 1, n) ;
62         version.push_back(root) ;
63     }
64
65     int query(int verL, int verR, int k){
66         auto qry = [&](auto &&self, const int
67             cur_old, const int cur_new, int L,
68             int R){
69             if(L == R) return L ;
70
71             int old_l = seg[cur_old].left, old_r =
72                 seg[cur_old].right ;

```

```

65     int new_l = seg[cur_new].left, new_r =
        seg[cur_new].right ;
66
67     int dl = seg[new_l].sum -
        seg[old_l].sum ;
68     int dr = seg[new_r].sum -
        seg[old_r].sum ;
69
70     int M = (L + R) >> 1 ;
71
72     if(dl >= k) return self(self, old_l,
        new_l, L, M) ;
73     k -= dl ;
74     return self(self, old_r, new_r, M+1,
        R) ;
75 };
76
77     int idx = qry(qry, version[verL-1],
        version[verR], 1, n) ;
78     return idx ;
79 }
80 };

```

4 Graph

4.1 cut vertex 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 }

```

4.2 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     if(dfn[u] == low[u]){
17         cnt_scc++ ;
18         int x ;
19
20         do{
21             x = st.top() ;
22             st.pop() ;
23
24             inSt[x] = 0 ;
25             sccId[x] = cnt_scc ;
26             scc[cnt_scc].push_back(x) ;
27         } while(x != u) ;
28     }
29 }
30
31 // SCC to DAG (after dfs)
32 vector<int> dag[Maxn] ;
33
34 void scc_to_dag(){
35     vector<int> dag[Maxn] ;
36     for ( int u=1 ; u<=n ; u++ ){
37         for ( auto v : g[u] ){
38             if(sccId[u] != sccId[v]){
39                 dag[sccId[u]].push_back(sccId[v]) ;
40             }
41         }
42     }
43 }
44
45 void init(){
46     memset(dfn, -1, sizeof(dfn)) ;
47     memset(low, -1, sizeof(low)) ;
48 }
49
50 int main(){
51     init() ;
52     input() ;
53     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
        == -1){
54         dfs1(i, 0) ;
55     }
56 }

```

4.3 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 dfnCnt = 0, dfn[Maxn], low[Maxn],
    vis_bcc[Maxn], bccCnt = 0 ;
16
17 void dfs1(int u, int from){
18     dfn[u] = low[u] = ++dfnCnt ;
19
20     for ( int i=head[u] ; i!=-1 ; i=e[i].next
        ){
21         int v = e[i].v ;
22
23         if(dfn[v] == -1){
24             dfs1(v, i) ;
25             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
                1 ;
26             low[u] = min(low[u], low[v]) ;
27         }

```

```

28     else if(i != (from ^ 1)) low[u] =
        min(low[u], dfn[v]) ;
29 }
30 }
31
32 void dfs2(int u, int id){
33     vis_bcc[u] = id ;
34     bcc[id].push_back(u) ;
35
36     for ( int i=head[u] ; i!=-1 ; i=e[i].next
        ){
37         int v = e[i].v ;
38
39         if(vis_bcc[v] != -1 || bz[i]) continue ;
40         dfs2(v, id) ;
41     }
42 }
43
44 void init(){
45     memset(dfn, -1, sizeof(dfn)) ;
46     memset(head, -1, sizeof(head)) ;
47     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
48 }
49
50 int main(){
51     init() ;
52     input() ;
53     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
        == -1){
54         dfs1(i, 0) ;
55     }
56
57     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
        == -1){
58         bcc.push_back(vector<int>()) ;
59         dfs2(i, bccCnt++) ;
60     }
61 }

```

4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a,
        const Coordinate&b){
5         if(a.x == b.x) return a.y < b.y ;
6         return a.x < b.x ;
7     }
8
9     friend bool operator==(const Coordinate&
        a, const Coordinate&b){
10        return a.x == b.x && a.y == b.y ;
11    } ;
12 } ;
13
14 vector<Coordinate> nodes ;
15
16 long long cross(const Coordinate&o, const
    Coordinate&a, const Coordinate&b){
17     return (a.x - o.x) * (b.y - o.y) - (a.y -
        o.y) * (b.x - o.x) ;
18 }
19
20 void input(){
21     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
49     for ( int i=n-2, t=m ; i>=0 ; i-- ){
50         while(m > t && cross(ch[m-2], ch[m-1],
51             nodes[i]) < 0) m-- ;
52         ch[m++] = nodes[i] ;
53     }
54
55     if(n > 1) m-- ;
56     cout << m << endl ;
57
58     for ( int i=0 ; i<m ; i++ ) cout <<
59         ch[i].x << " " << ch[i].y << endl ;
60 }

```

5 String

5.1 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 }

```

5.2 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 class Trie{
12 private:
13     int cnt ;
14     TrieNode *root ;
15 public:
16     Trie() : cnt(0) {
17         root = new TrieNode() ;
18     }
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         node->end.insert(n) ;
31
32     void search(string &str){
33         TrieNode* node = root ;
34         for ( auto s : str ){
35             int path = s - 'a' ;
36             if(node->next[path] == nullptr)
37                 return ;
38             node = node->next[path] ;
39         }
40
41         int flg = 0 ;
42         for ( auto n : node->end ){
43             if(flg) cout << " " ;
44             else flg = 1 ;
45         }
46         cout << n ;
47     }
48
49     void clear(TrieNode* node) {
50         if (!node) return ;
51         for ( int i = 0 ; i < 26 ; i++ ) {
52             if (node->next[i]) {
53                 clear(node->next[i]) ;
54             }
55             delete node ;
56         }
57     }
58
59 ~Trie(){
60     clear(root) ;
61 };

```

5.3 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
49        while(!q.empty()){
50            int u = q.front() ; q.pop() ;
51            order.push_back(u) ;
52
53            for ( int i=0 ; i<alpha ; i++){
54                if(tree[u][i] == 0) tree[u][i] =
55                    tree[fail[u]][i] ;
56                else{
57                    fail[tree[u][i]] = tree[fail[u]][i] ;
58                    q.push(tree[u][i]) ;
59                }
60            }
61        }
62
63        vector<int> count_per_pattern(const string
64            &text) const {
65            int u = 0 ;
66            vector<int> vis(tree.size(), 0) ;
67
68            for ( char ch : text ){
69                u = tree[u][ch - base] ;
70                vis[u]++ ;
71            }
72
73            for ( int i=order.size()-1 ; i>=1 ; i-- )
74                {
75                    int x = order[i] ;
76                    vis[fail[x]] += vis[x] ;
77                }
78
79            vector<int> ans(end.size(), 0) ;
80            for ( int id=0 ; id<end.size() ; id++ ){
81                ans[id] = vis[end[id]] ;
82            }
83        }

```

```

79 }
80
81 return ans ;
82 }
83 };

```

6 Algorithm

6.1 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 }

```

6.2 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 ; i++ ){
42         auto [u, v, w] = edge[i] ;
43         if(merge(u, v)){
44             cnt++ ;
45             sum += w ;
46         }
47     }
48 }
49
50 int main(){
51     for ( int i=0 ; i<m ; i++ ){
52         scanf("%d%d%d", &u, &v, &w) ;
53         edge.push_back({u, v, w}) ;
54         sum += w ;
55     }
56
57     sort(edge.begin(), edge.end(),
58          greater<Edge>()) ;
59     MST() ;
60 }

```

6.3 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 }

```

6.4 Max Flow

```

1 struct Edge{
2     int v, cap, next ;
3 };
4
5 class MaxFlow{
6 private:
7     int N, S, T ;
8     vector<Edge> e ;
9     vector<int> head, cur, dep ;
10
11     bool bfs(){
12         queue<int> q ;
13         for ( int i=0 ; i<=N ; i++ ){
14             cur[i] = head[i] ;
15             dep[i] = -1 ;
16         }
17
18         q.push(S) ;
19         dep[S] = 0 ;

```

```

20 while(!q.empty()){
21     int u = q.front() ; q.pop() ;
22
23     for ( int i=head[u] ; i!=-1 ;
24         i=e[i].next ){
25         int v = e[i].v ;
26         if(dep[v] == -1 && e[i].cap > 0){
27             dep[v] = dep[u] + 1 ;
28             if(v == T) return 1 ;
29             q.push(v) ;
30         }
31     }
32 }
33
34 return 0 ;
35 }
36
37 int dfs(int u, int flow){
38     if(u == T) return flow ;
39     int d, rest = 0 ;
40
41     for ( int &i=cur[u] ; i!=-1 ;
42         i=e[i].next ){
43         int v = e[i].v ;
44         if(dep[v] == dep[u] + 1 && e[i].cap >
45             0){
46             d = dfs(v, min(flow - rest,
47                 e[i].cap)) ;
48
49             if(d > 0){
50                 e[i].cap -= d ;
51                 e[i^1].cap += d ;
52                 rest += d ;
53
54                 if(rest == flow) break ;
55             }
56         }
57         if(rest != flow) dep[u] = -1 ;
58         return rest ;
59     }
60     public:
61     MaxFlow(int n, int s, int t){
62         N = n ; S = s ; T = t ;
63         e.reserve(n*n) ;
64         head.assign(n+1, -1) ;
65         cur.resize(n+1) ;
66         dep.resize(n+1) ;
67     }
68
69     void AddEdge(int u, int v, int cap){
70         e.push_back({v, cap, head[u]}) ;
71         head[u] = e.size() - 1 ;
72         e.push_back({u, 0, head[v]}) ;
73         head[v] = e.size() - 1 ;
74     }
75
76     int run(){
77         int ans = 0 ;
78         while(bfs()){
79             ans += dfs(S, 0x3f3f3f3f) ;
80         }
81         return ans ;
82     }
};

```

6.5 min cut max flow

```

1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:

```

```

8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11    public:
12    MCMF(int n, int _s, int _t){
13        N = n ;
14        s = _s ;
15        t = _t ;
16        e.resize(n*n + 5) ;
17        head.assign(n+5, -1) ;
18        tot = -1 ;
19    }
20
21    void AddEdge(int u, int v, int cap, int
22        cost){
23        e[++tot] = {v, cap, cost, head[u]} ;
24        head[u] = tot ;
25        e[++tot] = {u, 0, -cost, head[v]} ;
26        head[v] = tot ;
27    }
28
29    int run(){
30        vector<int> dis(N+1), pot(N+1, 0),
31            preE(N+1) ;
32        int flow = 0, cost = 0 ;
33
34        auto dijkstra = [&]() {
35            fill(dis.begin(), dis.end(), INF) ;
36            priority_queue<pii, vector<pii>,
37                greater<pii>> pq ;
38            dis[s] = 0 ;
39            pq.push({0, s}) ;
40
41            while(!pq.empty()){
42                auto [d, u] = pq.top() ; pq.pop() ;
43                if(d > dis[u]) continue ;
44                for ( int i=head[u] ; i!=-1 ;
45                    i=e[i].next ){
46                    int v = e[i].v, cap = e[i].cap, w =
47                        e[i].cost ;
48                    if(cap && dis[v] > d + w + pot[u] -
49                        pot[v]){
50                        dis[v] = d + w + pot[u] - pot[v] ;
51                        preE[v] = i ;
52                        pq.push({dis[v], v}) ;
53                    }
54                }
55            }
56
57            return dis[t] != INF ;
58        };
59
60        while(dijkstra()){
61            for ( int v=1 ; v<=N ; v++ ) if(dis[v]
62                < INF){
63                pot[v] += dis[v] ;
64            }
65
66            int aug = INT_MAX ;
67            for ( int v=t ; v!=s ;
68                v=preE[v^1].v ){
69                aug = min(aug, e[preE[v]].cap) ;
70            }
71
72            for ( int v=t ; v!=s ;
73                v=preE[v^1].v ){
74                e[preE[v]].cap -= aug ;
75                e[preE[v^1]].cap += aug ;
76                cost += aug * e[preE[v]].cost ;
77            }
78
79            return cost ;
80        }
81    }
};

```

7 DP

7.1 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 }

```

7.2 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
16     int &d = dp[pos][sum][dsum][lim] ;
17     if(d != -1) return d ;
18
19     int up = lim ? dig[pos] : 9 ;
20     int res = 0 ;
21     for ( int i=0 ; i<=up ; i++){
22         res += solve(pos-1, (sum * 10 + i) %
23             K, (dsum + i) % K, lim && i==up) ;
24     }
25
26     return d = res ;
27 }

```



```

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 }

```

```

41 int main(){
42     int t = 0 ;
43
44     while(~scanf("%d", &n) && n){
45         init() ;
46         for ( int i=0 ; i<n-1 ; i++ ){
47             int u, v, w ;
48             scanf("%d%d%d", &v, &u, &w) ;
49             edge[u].push_back({v, w}) ;
50         }
51
52         DFS(0) ;
53         printf("Case %d:\n", ++t) ;
54
55         int q, e ;
56         scanf("%d", &q) ;
57
58         while(q--){
59             scanf("%d", &e) ;
60
61             for ( int i=n ; i>=1 ; i-- )
62                 if(dp[0][i][0] <= e){
63                     printf("%d\n", i) ;
64                     break ;
65                 }
66             }
67         }
68     return 0 ;
69 }

```

7.3 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] ){
34         for ( int i=cnt[u] ; i>1 ; i-- ) for (
35             int j=1 ; j<i && j<=cnt[v] ; j++ ){
36             dp[u][i][1] = min(dp[u][i][1],
37                 dp[u][i-j][1] + dp[v][j][1] + 2 *
38                 w) ;
39             dp[u][i][0] = min(dp[u][i][0],
40                 dp[u][i-j][1] + dp[v][j][0] + w) ;
41             dp[u][i][0] = min(dp[u][i][0],
42                 dp[u][i-j][0] + dp[v][j][1] + 2 *
43                 w) ;
44         }
45     }
46 }
47
48 }
49
50 }

```