

## 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 1 import math
1 2
1 3 math.ceil(x) #上高斯
1 4 math.floor(x) #下高斯
1 5 math.factorial(x) #階乘
1 6 math.fabs(x) #絕對值
1 7 math.fsum(arr) #求和
1 8 math.gcd(x, y)
1 9 math.exp(x) # e^x
1 10 math.log(x, base)
1 11 math.log2(x)
1 12 math.log10(x)
1 13 math.sqrt(x)
1 14 math.pow(x, y, mod)
1 15 math.sin(x) # cos, tan, asin, acos, atan,
1 16 atan2, sinh ...
1 17 math.hypot(x, y) #歐幾里德範數
1 18 math.degrees(x) #x從弧度轉角度
1 19 math.radians(x) #x從角度轉弧度
1 20 math.gamma(x) #x的gamma函數
1 21 math.pi #const
1 22 math.e #const
1 23 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 1 ll extgcd(ll a, ll b, llc, 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
60   void modify(int l, int r, LazyTag &tag,
61   int L=1, int R=n, int id=1){
62     if(l <= L && R <= r){
63       AddTag(id, tag) ;
64       return ;
65     }
66
67     push(id) ;
68     int M = (L + R) >> 1 ;
69     if(r <= M) modify(l, r, tag, L, M,
70       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     rnk[Maxn], dfscnt = 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] = ++dfscnt ;
22     rnk[dfscnt] = u ;
23
24     if(son[u] == -1) return ;
25
26     for ( auto v : g[u] ) if(v != fa[u] && v
27         != son[u]){
28         dfs2(v, v) ;
29     }
30
31 /* Segment Tree */
32 #define lc (id << 1)
33 #define rc ((id << 1) | 1)
34
35 struct ColorSeg{
36     int left, right, tot ;
37
38     ColorSeg operator+(const ColorSeg &o)
39     const {
40         if(tot == 0) return o ;
41         if(o.tot == 0) return *this ;
42
43         ColorSeg tmp ;
44         tmp.left = left ;
45         tmp.right = o.right ;
46         tmp.tot = tot + o.tot - (right ==
47             o.left) ;

```

```

48     }
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
80     public:
81         void build(int L=1, int R=n, int id=1){
82             // normal build
83         }
84
85         // update val from u to v (simple path)
86         void update(int u, int v, int val){
87             while(top[u] != top[v]){
88                 if(dep[top[u]] < dep[top[v]]) swap(u,
89                     v) ;
90                 modify(dfn[top[u]], dfn[u], val) ;
91                 u = fa[top[u]] ;
92             }
93
94             if(dep[u] < dep[v]) swap(u, v) ;
95             modify(dfn[v], dfn[u], val) ;
96
97             // get sum from u to v (simple path)
98             int get(int u, int v){
99                 pair<int, ColorSeg> U, V ;
100                ColorSeg M ;
101                U = {u, {0, 0, 0}} ;
102                V = {v, {0, 0, 0}} ;
103
104                while(top[U.first] != top[V.first]){
105                    if(dep[top[U.first]] <
106                        dep[top[V.first]]) swap(U, V) ;
107                    U.second = query(dfn[top[U.first]],
108                                     dfn[U.first]) + U.second ;
109                    U.first = fa[top[U.first]] ;
110
111                    if(dep[U.first] < dep[V.first]) swap(U,
112                        V) ;
113
114                    M = query(dfn[V.first], dfn[U.first]) ;
115
116                    return (U.second.tot + V.second.tot +
117                            M.tot) - (U.second.left == M.right)
118                            - (V.second.left == M.left) ;
119                }
120            }
121
122 }tree ;

```

```

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
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                int cur_cnt = cnt++ ;
40
41                if(L == R){
42                    seg[cur_cnt] = {seg[cur].sum + 1, 0,
43                                    0} ;
44                    return cur_cnt ;
45                }
46
47                int M = (L + R) >> 1 ;
48                int lc = seg[cur].left ;
49                int rc = seg[cur].right ;
50
51                if(idx <= M) lc = self(self,
52                                         seg[cur].left, L, M) ;
53                else rc = self(self, seg[cur].right,
54                               M+1, R) ;
55
56                seg[cur_cnt] = {seg[lc].sum +
57                               seg[rc].sum, lc, rc} ;
58
59                return cur_cnt ;
60            };
61
62            int root = upd(upd, version[ver], 1, n) ;
63            version.push_back(root) ;
64        }
65
66        int query(int verL, int verR, int k){
67            auto qry = [&](auto &self, const int
68                          cur_old, const int cur_new, int L,
69                          int R){
70                if(L == R) return L ;
71
72                int old_l = seg[cur_old].left, old_r =
73                    seg[cur_old].right ;

```

```

65     int new_l = seg[cur_new].left, new_r =
66         seg[cur_new].right ;
67
68     int dl = seg[new_l].sum -
69         seg[old_l].sum ;
70     int dr = seg[new_r].sum -
71         seg[old_r].sum ;
72
73     int M = (L + R) >> 1 ;
74
75     if(dl >= k) return self(self, old_l,
76         new_l, L, M) ;
77     k -= dl ;
78     return self(self, old_r, new_r, M+1,
79         R) ;
80 }
81
82 int idx = qry(qry, version[verL-1],
83     version[verR], 1, n) ;
84 return idx ;
85 }
```

## 4 Graph

### 4.1 cut vertex AND bridges

```

1 int dfn[Maxn] = {-1}, low[Maxn] = {-1},
2     dfscnt ;
3
4 void dfs(int u, int fa){
5     dfn[u] = low[u] = ++dfscnt ;
6     int child = 0 ;
7
8     for ( auto v : g[u] ) if(v != fa){
9         if(dfn[v] == -1){
10             child++ ;
11             dfs(v, u) ;
12             low[u] = min(low[u], low[v]) ;
13
14             if(low[v] >= dfn[u]){
15                 // this edge is a bridge
16             }
17
18             if(u != fa && low[v] >= dfn[u]){
19                 // this node v is a articulation point
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],
3     dfscnt = 0, cnt_scc = 0 ;
4 stack<int> st ;
5 bitset<Maxn> inSt, vis ;
6
7 void dfs(int u, int from){
8     dfn[u] = low[u] = ++dfscnt ;
9     st.push(u) ;
10    inSt[u] = 1 ;
11
12    for ( auto v : g[u] ){
13        if(!inSt[v] && dfn[v] != -1) continue ;
14        if(dfn[v] == -1) dfs(v, u) ;
15        low[u] = min(low[u], low[v]) ;
16    }
17 }
```

```

16     if(dfn[u] == low[u]){
17         cnt_scc++ ;
18         int x ;
19         do{
20             x = st.top() ;
21             st.pop() ;
22
23             inSt[x] = 0 ;
24             sccId[x] = cnt_scc ;
25             scc[cnt_scc].push_back(x) ;
26         }
27         while(x != u) ;
28     }
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     {
38         int v = e[i].v ;
39
40         if(vis_bcc[v] != -1 || bz[i]) continue ;
41         dfs2(v, id) ;
42     }
43 }
44
45 void init(){
46     memset(dfn, -1, sizeof(dfn)) ;
47     memset(head, -1, sizeof(head)) ;
48     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
49 }
50
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
55         == -1){
56         dfs1(i, 0) ;
57     }
58 }
```

### 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    vector<Coordinate> nodes ;
16
17    long long cross(const Coordinate& o, const
18        Coordinate& a, const Coordinate& b){
19        return (a.x - o.x) * (b.y - o.y) - (a.y -
20            o.y) * (b.x - o.x) ;
21    }
22
23    void input(){
24        nodes.clear() ;
25
26        int n, x, y ;
27        char c ;
28        cin >> n ;
29
30        for ( int i=0 ; i<n ; i++ ){
31            cin >> x >> y >> c ;
32            if(c == 'Y') nodes.push_back({x, y}) ;
33        }
34    }
35 }
```

```

void monotone(){
    sort(nodes.begin(), nodes.end()) ;

    int n = unique(nodes.begin(), nodes.end()
                  - nodes.begin()) ;

    vector<Coordinate> ch(n+1) ;

    int m = 0 ;

    for ( int i=0 ; i<n ; i++ ){
        while(m > 1 && cross(ch[m-2], ch[m-1],
                               nodes[i]) < 0) m-- ;
        ch[m++] = nodes[i] ;
    }
    for ( int i=n-2, t=m ; i>=0 ; i-- ){
        while(m > t && cross(ch[m-2], ch[m-1],
                               nodes[i]) < 0) m-- ;
        ch[m++] = nodes[i] ;
    }
    if(n > 1) m-- ;
    cout << m << endl ;

    for ( int i=0 ; i<m ; i++ ) cout <<
        ch[i].x << " " << ch[i].y << endl ;
}

```

5 String

## 5.1 KMP

```

int Next[N] ;
void kmp(string &str){
    Next[0] = -1 ;
    if(str.size() <= 1) return ;
    Next[1] = 0 ;

    int cur = 2, check = 0 ;

    while(cur < str.size()){
        if(str[cur - 1] == str[check])
            Next[cur++] = ++check ;
        else if(check > 0) check =
            Next[check] ;
        else Next[cur++] = 0 ;
    }
}

int main(){
ios::sync_with_stdio(false) ;
cin.tie(nullptr) ;
cout.tie(nullptr) ;

string s1, s2 ;
while(cin >> s1){
    s2 = s1 ;
    reverse(s2.begin(), s2.end()) ;
    kmp(s2) ;

    int x=0, y=0 ;
    while(x < s1.size() && y < s2.size()){
        if(s1[x] == s2[y]){
            x++ ;
            y++ ;
        }
        else if(y > 0) y = Next[y] ;
        else x++ ;
    }
    cout << s1 << s2.substr(y) << endl
}

return 0 ;
}

```

## 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
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
34     void search(string &str){
35         TrieNode* node = root ;
36         for ( auto s : str ){
37             int path = s - 'a' ;
38             if(node->next[path] == nullptr)
39                 return ;
40             node = node->next[path] ;
41         }
42
43         int flg = 0 ;
44         for ( auto n : node->end ){
45             if(flg) cout << " " ;
46             else flg = 1 ;
47
48             cout << n ;
49         }
50     }
51
52     void clear(TrieNode* node) {
53         if (!node) return ;
54         for (int i = 0; i < 26; i++) {
55             if (node->next[i]) {
56                 clear(node->next[i]) ;
57             }
58         }
59         delete node ;
60     }
61
62     ~Trie(){
63         clear(root) ;
64     }
65 }
```

5.3 ACAM

```

7 int new_node(){
8     tree.emplace_back(alpha, 0) ;
9     fail.push_back(0) ;
10
11    return tree.size() - 1 ;
12}
13
14public:
15ACAutomation(int _base= 'a', int _alpha=26)
16    : base(_base), alpha(_alpha) {
17    clear() ;
18}
19
20void clear(){
21    fail.assign(1, 0) ;
22    order.clear() ;
23    end.clear() ;
24    tree.assign(1, vector<int>(alpha, 0)) ;
25}
26
27void 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
37    end.push_back(u) ;
38}
39
40void build(){
41    queue<int> q ;
42    order.clear() ;
43    order.push_back(0) ;
44
45    for ( int i=0 ; i<alpha ; i++ )
46        if(tree[0][i] > 0){
47            q.push(tree[0][i]) ;
48        }
49
50    while(!q.empty()){
51        int u = q.front() ; q.pop() ;
52        order.push_back(u) ;
53
54        for ( int i=0 ; i<alpha ; i++ ){
55            if(tree[u][i] == 0) tree[u][i] =
56                tree[fail[u]][i] ;
57            else{
58                fail[tree[u][i]] = tree[fail[u]][i] ;
59                ;
60                q.push(tree[u][i]) ;
61            }
62        }
63    }
64}
65
66vector<int> count_per_pattern(const string
67    &text) const {
68    int u = 0 ;
69    vector<int> vis(tree.size(), 0) ;
70
71    for ( char ch : text ){
72        u = tree[u][ch - base] ;
73        vis[u]++;
74    }
75
76    for ( int i=order.size()-1 ; i>=1 ; i-- )
77    ){
78        int x = order[i] ;
79        vis[fail[x]] += vis[x] ;
80    }
81
82    vector<int> ans(end.size(), 0) ;
83    for ( int id=0 ; id<end.size() ; id++ ){
84        ans[id] = vis[end[id]] ;
85    }
86}

```

```

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
34     if(u == v) return u ;
35
36     for ( int i=MaxLog ; i>=0 ; i-- )
37         if(up[u][i] != up[v][i]){
38             u = up[u][i] ;
39             v = up[v][i] ;
40         }
41
42     return up[u][0] ;
43 }
44
45 int main(){
46     int n, q, root ;
47     scanf("%d%d%d", &n, &q, &root) ;
48     MaxLog = __lg(n) ;
49
50     for ( int i=0 ; i<n-1 ; i++ ){
51         int u, v ;
52         scanf("%d%d", &u, &v) ;
53         e[u].push_back(v) ;
54         e[v].push_back(u) ;
55     }
56
57     dfs(root, root, 0) ;
58
59     while(q--){
60         int u, v ;
61         scanf("%d%d", &u, &v) ;
62         printf("%d\n", lca(u, v)) ;
63     }
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
42                ; i++ ){
43            auto [u, v, w] = edge[i] ;
44            if(merge(u, v)){
45                cnt++ ;
46                sum -= w ;
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        }
60    }
61 }
```

### 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     int SG[30];
31     int vis[Maxn], stone[Maxn];
32
33     void build(){
34         SG[0] = 0;
35         memset(vis, 0, sizeof(vis));
36
37         for ( int i=1 ; i<30 ; i++ ){
38             int cur = 0;
39             for ( int j=0 ; j<i ; j++ ) for ( int
40                     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         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     }
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    }
21
22    int dfs(int v, int f){
23        if(v == T) return f ;
24
25        for ( int i=cur[v] ; i<e[v].size(); i++ ){
26            Edge &e = e[v][i] ;
27
28            if(e.cap > 0 && dep[e.v] < dep[v] + 1){
29                int d = dfs(e.v, min(f, e.cap)) ;
30
31                if(d > 0){
32                    e.cap -= d ;
33                    e.v = v ;
34
35                    cur[v] = i + 1 ;
36                    dep[v] = dep[e.v] + 1 ;
37
38                    return d ;
39                }
40            }
41        }
42
43        return 0 ;
44    }
45
46    int maxflow(){
47        int f = 0 ;
48
49        while(bfs()){
50            for ( int v=S ; v<T ; v++ ){
51                if(dfs(v, INT_MAX) <= 0) break ;
52            }
53
54            if(v >= T) break ;
55        }
56
57        return f ;
58    }
59 }
```

```

20
21     while(!q.empty()){
22         int u = q.front() ; q.pop() ;
23
24         for ( int i=head[u] ; i!=-1 ;
25             i=e[i].next ){
26             int v = e[i].v ;
27             if(dep[v] == -1 && e[i].cap > 0){
28                 dep[v] = dep[u] + 1 ;
29                 if(v == T) return 1 ;
30                 q.push(v) ;
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
61         public:
62         MaxFlow(int n, int s, int t){
63             N = n ; S = s ; T = t ;
64             e.reserve(n*n) ;
65             head.assign(n+1, -1) ;
66             cur.resize(n+1) ;
67             dep.resize(n+1) ;
68         }
69
70         void AddEdge(int u, int v, int cap){
71             e.push_back({v, cap, head[u]}) ;
72             head[u] = e.size() - 1 ;
73             e.push_back({u, 0, head[v]}) ;
74             head[v] = e.size() - 1 ;
75         }
76
77         int run(){
78             int ans = 0 ;
79             while(bfs()){
80                 ans += dfs(S, 0x3f3f3f3f) ;
81             }
82             return ans ;
83         }
84     }

```

## 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
12    public:
13        MCMF(int n, int _s, int _t){
14            N = n ;
15            s = _s ;
16            t = _t ;
17            e.resize(n*n + 5) ;
18            head.assign(n+5, -1) ;
19            tot = -1 ;
20        }
21
22        void AddEdge(int u, int v, int cap, int
23                      cost){
24            e[++tot] = {v, cap, cost, head[u]} ;
25            head[u] = tot ;
26            e[++tot] = {u, 0, -cost, head[v]} ;
27            head[v] = tot ;
28        }
29
30        int run(){
31            vector<int> dis(N+1), pot(N+1, 0),
32                        preE(N+1) ;
33            int flow = 0, cost = 0 ;
34
35            auto dijkstra = [&](){
36                fill(dis.begin(), dis.end(), INF) ;
37                priority_queue<pii, vector<pii>,
38                                greater<pii>> pq ;
39                dis[s] = 0 ;
40                pq.push({0, s}) ;
41
42                while(!pq.empty()){
43                    auto [d, u] = pq.top() ; pq.pop() ;
44                    if(d > dis[u]) continue ;
45                    for ( int i=head[u] ; i!=-1 ;
46                        i=e[i].next ){
47                        int v = e[i].v, cap = e[i].cap, w =
48                            e[i].cost ;
49                        if(cap && dis[v] > d + w + pot[u] -
50                            pot[v]){
51                            dis[v] = d + w + pot[u] - pot[v] ;
52                            preE[v] = i ;
53                            pq.push({dis[v], v}) ;
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                        int aug = INT_MAX ;
66                        for ( int v=t ; v!=s ;
67                            v=e[preE[v]^1].v ){
68                            aug = min(aug, e[preE[v]].cap) ;
69
70                            for ( int v=t ; v!=s ;
71                                v=e[preE[v]^1].v ){
72                                e[preE[v]].cap -= aug ;
73                                e[preE[v]^1].cap += aug ;
74                                cost += aug * e[preE[v]].cost ;
75                            }
76
77                            return cost ;
78                        }
79
80                    }
81                }
82            };
83
84        }

```

## 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 int main(){
17    while(~scanf("%d%d", &n, &m)){
18        if(m > n) swap(n, m) ;
19        memset(dp, 0, sizeof(dp)) ;
20        cur = 0 ;
21        dp[cur][(1 << m) - 1] = 1 ;
22        for ( int i=0 ; i<n ; i++ ) for ( int
23            j=0 ; j<m ; j++ ){
24            cur ^= 1 ;
25            memset(dp[cur], 0, sizeof(dp[cur])) ;
26
27            for ( int k=0 ; k<(1 << m) ; k++ ){
28                update(k, k << 1) ; // not put
29                if(i && !(k & (1 << (m - 1))) )
30                    update(k, (k << 1) | (1 << m) |
31                        1) ; // put up
32                if(j && !(k & 1)) update(k, (k << 1)
33                    | 3) ; // put left
34            }
35            printf("%lld\n", dp[cur][(1 << m) - 1]) ;
36        }
37        return 0 ;
38    }

```

### 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%d", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) -
48             count(a-1)) ;
49     }
50
51     return 0 ;
}

```

```

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", &u, &v, &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     }
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 }

```