Contents

1 Basic

1.1 PyMath

```python
import math

math.ceil(x) #上高斯
math.floor(x) #下高斯
math.factorial(x) #階乘
math.fabs(x) #絕對值
math.fsum(arr) #求和
math.gcd(x, y)
math.exp(x) # e^x
math.log(x, base)
math.log2(x)
math.log10(x)
math.sqrt(x)
math.pow(x, y, mod)
math.sin(x) # cos, tan, asin, acos, atan,
    atan2, sinh ...
math.hypot(x, y) #歐幾里德範數
math.degrees(x) #x從弧度轉角度
math.radians(x) #x從角度轉弧度
math.gamma(x) #x的gamma函數
math.pi #const
math.e #const
math.inf
```

2 Math

2.1 formula

1. Catalan Number

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

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

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

```cpp
ll extgcd(ll a, ll b, llc, ll &x, ll &y){
    if(b == 0){
        x = c/a ;
        y = 0 ;
        return a ;
    }
    ll d = extgcd(b, a%b, c, x, y), tmp =
        x ;
    x = y ;
    y = tmp - (a/b)*y ;
    return d ;
}
```

3 Tree

3.1 SegmentTree

```cpp
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct LazyTag{
    // type 0 : increase val
    // type 1 : set to val
    // type 1 can overwrite type 0
    int type ;
    ll val ;
} ;

struct Node{
    LazyTag tag ;
    ll sum ;
    int sz ;
}seg[Maxn << 2] ;

class SegmentTree{
private:
    void pull(int id){
        seg[id].sum = seg[lc].sum +
            seg[rc].sum ;
    }

    void AddTag(int id, LazyTag &tag){
        if(tag.type == 0){
            seg[id].sum += tag.val *
                seg[id].sz ;
            seg[id].tag.val += tag.val ;
        }
        else{
            seg[id].sum = tag.val *
                seg[id].sz ;
            seg[id].tag = {1, tag.val} ;
        }
    }

    void push(int id){
        AddTag(lc, seg[id].tag) ;
        AddTag(rc, seg[id].tag) ;
        seg[id].tag = {0, 0} ;
    }

public:
    void build(int L=1, int R=n, int id=1){
        seg[id].sum = 0 ;
        seg[id].tag = {0, 0} ;
        seg[id].sz = 1 ;

        if(L == R){
            seg[id].sum = arr[L] ;
            return ;
        }

        int M = (L + R) >> 1 ;
        build(L, M, lc) ;
        build(M+1, R, rc) ;

        pull(id) ;
        seg[id].sz = seg[lc].sz + seg[rc].sz ;
    }

    void modify(int l, int r, LazyTag &tag,
        int L=1, int R=n, int id=1){
        if(l <= L && R <= r){
            AddTag(id, tag) ;
            return ;
        }

        push(id) ;
        int M = (L + R) >> 1 ;
        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 ;
```

3.2 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
          != son[u]){
26     dfs2(v, v) ;
27   }
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)
          const {
38     if(tot == 0) return o ;
39     if(o.tot == 0) return *this ;
40
41     ColorSeg tmp ;
42     tmp.left = left ;
43     tmp.right = o.right ;
44     tmp.tot = tot + o.tot - (right ==
          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
          L=1, int R=n, int id=1){
70      // normal modify
71    }
72
73    ColorSeg query(int l, int r, int L=1, int
          R=n, int id=1){
74      // normal query
75    }
76  public:
77    void build(int L=1, int R=n, int id=1){
78      // normal build
79    }
80
81      // update val from u to v (simple path)
82    void update(int u, int v, int val){
83      while(top[u] != top[v]){
84        if(dep[top[u]] < dep[top[v]]) swap(u,
             v) ;
85        modify(dfn[top[u]], dfn[u], val) ;
86        u = fa[top[u]] ;
87      }
88
89      if(dep[u] < dep[v]) swap(u, v) ;
90      modify(dfn[v], dfn[u], val) ;
91    }
92
93      // get sum from u to v (simple path)
94    int get(int u, int v){
95      pair<int, ColorSeg> U, V ;
96      ColorSeg M ;
97      U = {u, {0, 0, 0}} ;
98      V = {v, {0, 0, 0}} ;
99
100     while(top[U.first] != top[V.first]){
101       if(dep[top[U.first]] <
             dep[top[V.first]]) swap(U, V) ;
102       U.second = query(dfn[top[U.first]],
             dfn[U.first]) + U.second ;
103       U.first = fa[top[U.first]] ;
104     }
105
106     if(dep[U.first] < dep[V.first]) swap(U,
             V) ;
107
108     M = query(dfn[V.first], dfn[U.first]) ;
109
110     return (U.second.tot + V.second.tot +
             M.tot) - (U.second.left == M.right)
             - (V.second.left == M.left) ;
111   }
112 }tree ;
113
114 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
          cur, int L, int R){
37       int cur_cnt = cnt++ ;
38
39       if(L == R){
40         seg[cur_cnt] = {seg[cur].sum + 1, 0,
             0} ;
41         return cur_cnt ;
42       }
43
44       int M = (L + R) >> 1 ;
45       int lc = seg[cur].left ;
46       int rc = seg[cur].right ;
47
48       if(idx <= M) lc = self(self,
             seg[cur].left, L, M) ;
49       else rc = self(self, seg[cur].right,
             M+1, R) ;
50
51       seg[cur_cnt] = {seg[lc].sum +
             seg[rc].sum, lc, rc} ;
52
53       return cur_cnt ;
54     };
55
56     int root = upd(upd, version[ver], 1, n) ;
57     version.push_back(root) ;
58   }
59
60   int query(int verL, int verR, int k){
61     auto qry = [&](auto &&self, const int
          cur_old, const int cur_new, int L,
          int R){
62       if(L == R) return L ;
63
64       int old_l = seg[cur_old].left, old_r =
             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
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]
        == -1){
56     dfs(i, i) ;
57   }
58 }
```

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 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     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, bcc_cnt++) ;
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())
          - nodes.begin() ;
37
38    vector<Coordinate> ch(n+1) ;
39
40    int m = 0 ;
41
42    for ( int i=0 ; i<n ; i++ ){
43      while(m > 1 && cross(ch[m-2], ch[m-1],
            nodes[i]) < 0) m-- ;
44      ch[m++] = nodes[i] ;
45    }
46    for ( int i=n-2, t=m ; i>=0 ; i-- ){
47      while(m > t && cross(ch[m-2], ch[m-1],
            nodes[i]) < 0) m-- ;
48      ch[m++] = nodes[i] ;
49    }
50
51    if(n > 1) m-- ;
52    cout << m << endl ;
53
54    for ( int i=0 ; i<m ; i++ ) cout <<
          ch[i].x << " " << ch[i].y << endl ;
55  }
```

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])
           Next[cur++] = ++check ;
11     else if(check > 0) check =
           Next[check] ;
12     else Next[cur++] = 0 ;
13   }
14 }
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
37     cout << s1 << s2.substr(y) << endl ;
38   }
39
40   return 0 ;
41 }
```

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]
              = nullptr ;
8      }
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)
                   node->next[path] = new
                   TrieNode() ;
26             node = node->next[path] ;
27         }
28         node->end.insert(n) ;
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)
                   return ;
36             node = node->next[path] ;
37         }
38
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 };
```

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] =
             new_node() ;
33       u = tree[u][v] ;
34     }
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++ )
             if(tree[0][i] > 0){
45       q.push(tree[0][i]) ;
46     }
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] =
               tree[fail[u]][i] ;
54         else{
55           fail[tree[u][i]] = tree[fail[u]][i]
                 ;
56           q.push(tree[u][i]) ;
57         }
58       }
59     }
60   }
61
62   vector<int> count_per_pattern(const string
         &text) const {
63     int u = 0 ;
64     vector<int> vis(tree.size(), 0) ;
65
66     for ( char ch : text ){
67       u = tree[u][ch - base] ;
68       vis[u]++ ;
69     }
70
71     for ( int i=order.size()-1 ; i>=1 ; i--
           ){
72       int x = order[i] ;
73       vis[fail[x]] += vis[x] ;
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 };
```

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]){
30     u = up[u][i] ;
31   }
32
33   if(u == v) return u ;
34
35   for ( int i=MaxLog ; i>=0 ; i-- )
35       if(up[u][i] != up[v][i]){
36     u = up[u][i] ;
37     v = up[v][i] ;
38   }
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 ;} ;
5  } ;
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(),
         greater<Edge>()) ;
58     MST() ;
59 }
```

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
         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",
         &stone[i]) ;
56
57     for ( int i=1 ; i<=n ; i++ ) if(stone[i]
         & 1){
58       ans ^= SG[n-i] ;
59     }
60   }
61 }
```

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

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 ^
          1][s1] ;
13   }
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
          j=0 ; j<m ; j++ ){
23       cur ^= 1 ;
24       memset(dp[cur], 0, sizeof(dp[cur])) ;
25
26       for ( int k=0 ; k<(1 << m) ; k++ ){
27         update(k, k << 1) ; // not put
28         if(i && !(k & (1 << (m - 1))))
              update(k, (k << 1) | (1 << m) |
              1) ; // put up
29         if(j && !(k & 1)) update(k, (k << 1)
              | 3) ; // put left
30       }
31     }
32     printf("%lld\n", dp[cur][(1 << m) - 1]) ;
33   }
34   return 0 ;
35 }
```

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
      lim){
10   if(pos == -1){
11     if(sum == 0 && dsum == 0) return 1 ;
12     return 0 ;
13   }
14
15   int &d = dp[pos][sum][dsum][lim] ;
16   if(d != -1) return d ;
17
18   int up = lim ? dig[pos] : 9 ;
19   int res = 0 ;
20   for ( int i=0 ; i<=up ; i++ ){
21     res += solve(pos-1, (sum * 10 + i) %
            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%d", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) -
                count(a-1)) ;
48     }
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-- )
              if(dp[0][i][0] <= e){
62         printf("%d\n", i) ;
63         break ;
64       }
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++ )
         edge[i].clear() ;
19   memset(cnt, 0, sizeof(cnt)) ;
20   memset(dp, INF, sizeof(dp)) ;
21 }
22
23 void DFS(int u){
24   cnt[u] = 1 ;
25   for ( auto [v, w] : edge[u] ){
26     DFS(v) ;
27     cnt[u] += cnt[v] ;
28   }
29
30   dp[u][1][0] = dp[u][1][1] = 0 ;
31
32   for ( auto [v, w] : edge[u] ){
33     for ( int i=cnt[u] ; i>1 ; i-- ) for (
           int j=1 ; j<i && j<=cnt[v] ; j++ ){
34       dp[u][i][1] = min(dp[u][i][1],
             dp[u][i-j][1] + dp[v][j][1] + 2 *
             w) ;
35       dp[u][i][0] = min(dp[u][i][0],
             dp[u][i-j][1] + dp[v][j][0] + w) ;
36       dp[u][i][0] = min(dp[u][i][0],
             dp[u][i-j][0] + dp[v][j][1] + 2 *
             w) ;
37     }
38   }
39 }
40
```