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 Tree

2.1 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) ;
        else if(l > M) modify(l, r, tag, M+1, R, rc) ;
        else{
            modify(l, r, tag, L, M, lc) ;
            modify(l, r, tag, M+1, R, rc) ;
        }
        pull(id) ;
    }

    ll query(int l, int r, int L=1, int R=n, int id=1){
        if(l <= L && R <= r) return seg[id].sum ;

        push(id) ;
        int M = (L + R) >> 1 ;
        if(r <= M) return query(l, r, L, M, lc) ;
        else if(l > M) return query(l, r, M+1, R, rc) ;
        else return query(l, r, L, M, lc) + query(l, r, M+1, R, rc) ;
    }
}tree ;
```

2.2 HLD

```cpp
/* HLD */
int fa[Maxn], top[Maxn], son[Maxn], sz[Maxn], dep[Maxn] = {0}, dfn[Maxn], rnk[Maxn], dfscnt = 0 ;

void dfs1(int u, int from){
  fa[u] = from ;
  dep[u] = dep[from] + 1 ;
  sz[u] = 1 ;

  for ( auto v : g[u] ) if(v != from){
    dfs1(v, u) ;
    sz[u] += sz[v] ;
    if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
  }
}

void dfs2(int u, int t){
  top[u] = t ;
```

```
  dfn[u] = ++dfscnt ;
  rnk[dfscnt] = u ;

  if(son[u] == -1) return ;

  dfs2(son[u], t) ;

  for ( auto v : g[u] ) if(v != fa[u] && v != son[u]){
    dfs2(v, v) ;
  }
}

/* Segment Tree */
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct ColorSeg{
  int left, right, tot ;

  ColorSeg operator+(const ColorSeg &o) const {
    if(tot == 0) return o ;
    if(o.tot == 0) return *this ;

    ColorSeg tmp ;
    tmp.left = left ;
    tmp.right = o.right ;
    tmp.tot = tot + o.tot - (right == o.left) ;

    return tmp ;
  }
} ;

struct Node{
  ColorSeg color ;
  int tag ;
}seg[Maxn << 2] ;

class SegmentTree{
private:
  void pull(int id){
    // normal pull
  }

  void AddTag(int id, int tag){
    // normal AddTag
  }

  void push(int id){
    // normal push
  }

  void modify(int l, int r, int tag, int L=1, int R=n, int id=1){
    // normal modify
  }

  ColorSeg query(int l, int r, int L=1, int R=n, int id=1){
    // normal query
  }
public:
  void build(int L=1, int R=n, int id=1){
    // normal build
  }

  // update val from u to v (simple path)
  void update(int u, int v, int val){
    while(top[u] != top[v]){
      if(dep[top[u]] < dep[top[v]]) swap(u, v) ;
      modify(dfn[top[u]], dfn[u], val) ;
      u = fa[top[u]] ;
    }

    if(dep[u] < dep[v]) swap(u, v) ;
    modify(dfn[v], dfn[u], val) ;
  }

  // get sum from u to v (simple path)
  int get(int u, int v){
    pair<int, ColorSeg> U, V ;
    ColorSeg M ;
    U = {u, {0, 0, 0}} ;
    V = {v, {0, 0, 0}} ;

    while(top[U.first] != top[V.first]){
      if(dep[top[U.first]] < dep[top[V.first]]) swap(U, V) ;
      U.second = query(dfn[top[U.first]], dfn[U.first]) + U.second ;
      U.first = fa[top[U.first]] ;
    }

    if(dep[U.first] < dep[V.first]) swap(U, V) ;

    M = query(dfn[V.first], dfn[U.first]) ;

    return (U.second.tot + V.second.tot + M.tot) - (U.second.left == M.right) - (V.second.left == M.left) ;
  }
}tree ;

void init(){
  memset(son, -1, sizeof(son)) ;
}
```

2.3 Trie

```
class TrieNode{
public:
    set<int> end ;
    TrieNode *next[26] ;

    TrieNode(){
        for ( int i=0 ; i<26 ; i++ ) next[i] = nullptr ;
    }
};

class Trie{
private:
    int cnt ;
    TrieNode *root ;
public:
    Trie() : cnt(0) {
        root = new TrieNode() ;
    }

    void insert(string &str, int n){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;

            if(node->next[path] == nullptr) node->next[path] = new TrieNode() ;
            node = node->next[path] ;
        }
        node->end.insert(n) ;
    }

    void search(string &str){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;
            if(node->next[path] == nullptr) return ;
            node = node->next[path] ;
        }

        int flg = 0 ;
        for ( auto n : node->end ){
            if(flg) cout << " " ;
            else flg = 1 ;

            cout << n ;
        }
    }

    void clear(TrieNode* node) {
        if (!node) return ;
        for (int i = 0; i < 26; i++) {
            if (node->next[i]) {
                clear(node->next[i]) ;
            }
        }
        delete node ;
    }

    ~Trie(){
        clear(root) ;
    }
};
```

3 String

3.1 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 ;
}
```

4 Algorithm

4.1 LCA

```cpp
#include <bits/stdc++.h>

using namespace std ;

const int Maxn = 500005 ;

vector<int> e[Maxn] ;
int depth[Maxn] ;
int up[Maxn][40] ;
int MaxLog ;

void dfs(int u, int from, int d){
  up[u][0] = from ;
  depth[u] = d ;

  for ( int i=1 ; i<=MaxLog ; i++ ){
    up[u][i] = up[up[u][i - 1]][i - 1] ;
  }

  for ( auto v : e[u] ){
    if(v == from) continue ;
    dfs(v, u, d + 1) ;
  }
}

int lca(int u, int v){
  if(depth[u] < depth[v]) swap(u, v) ;

  for ( int i=MaxLog ; i>=0 ; i-- )
       if(depth[u] - (1 << i) >= depth[v]){
    u = up[u][i] ;
  }

  if(u == v) return u ;

  for ( int i=MaxLog ; i>=0 ; i-- )
       if(up[u][i] != up[v][i]){
    u = up[u][i] ;
    v = up[v][i] ;
  }

  return up[u][0] ;
}

int main(){
  int n, q, root ;
  scanf("%d%d%d", &n, &q, &root) ;
  MaxLog = __lg(n) ;

  for ( int i=0 ; i<n-1 ; i++ ){
    int u, v ;
    scanf("%d%d", &u, &v) ;
    e[u].push_back(v) ;
    e[v].push_back(u) ;
  }

  dfs(root, root, 0) ;

  while(q--){
    int u, v ;
    scanf("%d%d", &u, &v) ;
    printf("%d\n", lca(u, v)) ;
  }

  return 0 ;
}
```

4.2 MST

```cpp
struct Edge{
  int u, v, w ;
    // 這是最大生成樹, 最小生成樹要改成 w < o.w
  bool operator>(const Edge &o) const
       {return w > o.w ;} ;
} ;

int par[N] ;
int sz[N] ;
int sum ;

vector<Edge> edge ;

void init(){
  edge.clear() ;
  for ( int i=0 ; i<N ; i++ ){
    par[i] = i ;
    sz[i] = 1 ;
  }
  sum = 0 ;
}

int find(int x){
  if(x == par[x]) return x ;
  return par[x] = find(par[x]) ;
}

int merge(int x, int y){
  x = find(x) ;
  y = find(y) ;

  if(x == y) return 0 ;
  if(sz[x] > sz[y]) swap(x, y) ;
  par[x] = y ;
  sz[y] += sz[x] ;

  return 1 ;
}

void MST(){
  int cnt = 0 ;
  for ( int i=0 ; i<edge.size() && cnt < n-1
       ; i++ ){
    auto [u, v, w] = edge[i] ;
    if(merge(u, v)){
      cnt++ ;
      sum -= w ;
    }
  }
}

int main(){
    for ( int i=0 ; i<m ; i++ ){
        scanf("%d%d%d", &u, &v, &w) ;
        edge.push_back({u, v, w}) ;
        sum += w ;
    }

    sort(edge.begin(), edge.end(),
         greater<Edge>()) ;
    MST() ;
}
```

4.3 SG

```cpp
long long SG(long long k){

  if(k % 2 == 0){
    return k / 2;
  }
  else{
    return SG(k / 2);
  }

}

int main(){
  int cas, n;

  scanf("%d", &cas);
  while(cas--){
    scanf("%d", &n);

    long long s, v = 0;

    for(int i = 0; i < n; i++){
      scanf("%lld", &s);
      v ^= SG(s); //XOR
    }

    if(v) printf("YES\n");
    else printf("NO\n");
  }
}

int SG[30] ;
int vis[Maxn], stone[Maxn] ;

void build(){
  SG[0] = 0 ;
  memset(vis, 0, sizeof(vis)) ;

  for ( int i=1 ; i<30 ; i++ ){
    int cur = 0 ;
    for ( int j=0 ; j<i ; j++ ) for ( int
         k=0 ; k<=j ; k++ ){
      vis[SG[j] ^ SG[k]] = i ;
    }
    while(vis[cur] == i) cur++ ;
    SG[i] = cur ;
  }
}

int main(){
  build() ;

  int T = 0 ;
  while(~scanf("%d", &n) && n){
    int ans = 0 ;

    for ( int i=1 ; i<=n ; i++ ) scanf("%d",
         &stone[i]) ;

    for ( int i=1 ; i<=n ; i++ ) if(stone[i]
         & 1){
      ans ^= SG[n-i] ;
    }
    }
}
```

4.4 Convex

```cpp
struct Coordinate{
  long long x, y ;

  friend bool operator<(const Coordinate&a,
       const Coordinate& b){
    if(a.x == b.x) return a.y < b.y ;
    return a.x < b.x ;
  }

  friend bool operator==(const Coordinate&
       a, const Coordinate& b){
    return a.x == b.x && a.y == b.y ;
  }
} ;

vector<Coordinate> nodes ;

long long cross(const Coordinate& o, const
     Coordinate& a, const Coordinate& b){
  return (a.x - o.x) * (b.y - o.y) - (a.y -
       o.y) * (b.x - o.x) ;
}

void input(){
  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  }
```