# 博弈——威佐夫

using namespace std;

typedef long long ll;

int main()

{

int a, b;

while(~scanf("%d%d", &a, &b)){

if (a > b) swap(a, b);

int k = b-a;

int tmpa = (int)((1 + sqrt(5.0)) / 2 \* k);

int tmpb = tmpa + k;

if(tmpa == a && tmpb == b){

printf("0\n");

}

else printf("1\n");

}

return 0;

}

# 博弈——antiNim

using namespace std;

typedef long long ll;

int t, n;

int num;

int main()

{

scanf("%d", &t);

while(t--){

scanf("%d", &n);

bool flag = false;

int res = 0;

for (int i = 0; i < n; i++){

scanf("%d", &num);

if(num > 1) flag = true;

res ^= num;

}

if(res == 0 && !flag)

printf("John\n");

else if(res != 0 && flag)

printf("John\n");

else printf("Brother\n");

}

return 0;

}

# 博弈——比胜态走法

using namespace std;

typedef long long ll;

int n;

int num[200010];

int main()

{

while(~scanf("%d", &n) && n){

int res = 0;

for (int i = 0; i < n; i++){

scanf("%d", &num[i]);

res ^= num[i];

}

sort(num, num + n);

if(res != 0){

printf("Yes\n");

for (int i = 0; i < n; i++){

if((res ^ num[i]) <= num[i]){

printf("%d %d\n", num[i], res^num[i]);

}

}

}

else{

printf("No\n");

}

}

return 0;

}

# 博弈——Multinum，sg

using namespace std;

int sg[1000];

bool visit[1000];

void getsg(){

memset(sg, 0, sizeof sg);

sg[0] = 0;

sg[1] = 1;

sg[2] = 2;

for (int i = 3; i < 1000; i++){

memset(visit, 0, sizeof visit);

for (int j = 0; j < i; j++){

visit[sg[j]] = true;

}

for(int j = 1; j < i; j++){

for (int k = 1; j+k < i; k++){

int l = i-j-k;

int yihuo = sg[j]^sg[k]^sg[l];

visit[yihuo] = true;

}

}

for (int j = 0; j < 1000; j++){

if(!visit[j]){

sg[i] = j;

break;

}

}

}

for (int i = 0; i < 100; i++){

cout << "sg[i] = " << i << " " << sg[i] << endl;

}

}

int t, n, num;

int main()

{

scanf("%d", &t);

while(t--){

scanf("%d", &n);

int res = 0;

for(int i = 0; i < n; i++){

scanf("%d", &num);

if(num > 0 && num % 8 == 0){

res ^= num-1;

}

else if(num > 0 && num % 8 == 7){

res ^= num+1;

}

else res ^= num;

}

if(res != 0)

printf("First player wins.\n");

else printf("Second player wins.\n");

}

return 0;

}

# 线段树——查询比a大的第k个数

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

#include <vector>

#include <cmath>

using namespace std;

typedef long long ll;

#define maxn 100010

int m;

int sum[maxn\*4];

void build(){

memset(sum, 0, sizeof sum);

}

void maintain(int o){

int lc=o<<1, rc=o<<1|1;

sum[o] = sum[lc] + sum[rc];

}

bool update(int o, int l, int r, int pos, int val){//正常的更新函数

if(l == r){

if(sum[o] == 0 && val < 0) return false;

sum[o] += val;

return true;

}

int mid=(l+r)>>1, lc=o<<1, rc=o<<1|1;

bool res = false;

if(pos <= mid) res = update(lc, l, mid, pos, val);

else res = update(rc, mid+1, r, pos, val);

maintain(o);

return res;

}

int ret;//存储大于等于pos的第k个数是几

void query(int o, int l, int r, int pos, int& rk){//查询pos（包含）之后的第k个数是多少。题目要求是求比pos大的第k个数，见第85行

if(rk <= 0 || sum[o] == 0) return;//要查的rk值是0或者此区间内没有数

if(l == r){//每次到根节点就更新ret值

ret = l;

rk -= sum[o];

return;

}

if(pos <= l && rk > sum[o]){//如果该区间在pos后并且该区间的数之和小于rk，那么rk直接减去，因为要查的值一定不在该区间。

rk -= sum[o];

return;

}

int mid = (l+r)>>1, lc=o<<1, rc=o<<1|1;

if(pos <= mid) query(lc, l, mid, pos, rk);//如果pos在左区间，那么要查左区间

query(rc, mid+1, r, pos, rk);//否则直接查找右区间

}

int main()

{

while(~scanf("%d", &m)){

build();

for (int i = 0; i < m; i++){

int op1, op2, op3;

scanf("%d%d", &op1, &op2);

if(op1 == 0){

update(1, 1, 100000, op2, 1);

}

else if(op1 == 1){

if(update(1, 1, 100000, op2, -1) == false){

printf("No Elment!\n");

}

}

else if(op1 == 2){

scanf("%d", &op3);

ret = 0;

query(1, 1, 100000, op2+1, op3);//本行在传递的时候已经处理了比pos大的第k个数，即求大于等于op2+1的第k个数

if(ret == 0 || op3 > 0) printf("Not Find!\n");//因为每次到叶节点就更新，如果op3的值最后小于等于0了，说名找到了，如果没有的话，说明此时找的这个ret不是结果，没找到。

else printf("%d\n", ret);

}

}

}

return 0;

}

# 主席树——树上两点路径第k小，树上主席树，动态lca

//本题是让求树上两点路径之间的第k大，树上主席树+动态lca+输入外挂

//bzoj上需要输入外挂，而且要用边集数组来存边，用vector可能会TLE

//本题的思路是每个节点对应的线段树是从其父亲节点的线段树修改过来的，这样每个节点的线段树的结果，都是到根节点的路径上所有点的结果

//在进行查询的时候，只需要两个节点都对他们的lca节点做差就是u，v路径上所有点的线段树的结果。

//

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

#include <vector>

#include <cmath>

using namespace std;

typedef long long ll;

#define maxn 100005

#define maxm 100005

int n, m;

int u, v;

int x, y, z;

int val[maxn], valc[maxn];

int len;//离散后的长度

int tot, trcnt;

int tr[maxn];

int lc[maxn\*30], rc[maxn\*30], sum[maxn\*30];

//输入外挂

inline int iread() {

int f = 1; ll x = 0; char ch = getchar();

for(; ch < '0' || ch > '9'; ch = getchar()) f = ch == '-' ? -1 : 1;

for(; ch >= '0' && ch <= '9'; ch = getchar()) x = x \* 10 + ch - '0';

return f \* x;

}

//---------------edge array----------------

struct Edge{

int to, next;

}edge[maxn\*2];

int totedge, head[maxn];

void init\_edge(){

totedge = 0;

memset(head, -1, sizeof head);

}

void add\_edge(int u, int v){//无向图要加两条边

edge[totedge].to = v;

edge[totedge].next = head[u];

head[u] = totedge++;

}

//-----------------------------------------

//-----------presistent seg tree--------------

void init\_segtree(){//建树之前的初始化

tot = trcnt = 0;

memcpy(valc, val, sizeof val);

sort(valc, valc+n);

len = unique(valc, valc+n)-valc;

}

int build\_segtree(int l, int r){//建树操作

int root = tot++;

sum[root] = 0;

if(l != r){

int mid = (l+r)>>1;

lc[root] = build\_segtree(l, mid);

rc[root] = build\_segtree(mid+1, r);

}

return root;

}

int update\_segtree(int Lroot, int l, int r, int pos){//主席树正常的更新操作，根据Lroot创建一个新的树，与Lroot公用节点

int newroot = tot++;

sum[newroot] = sum[Lroot] + 1;

if(l != r){

int mid = (l+r)>>1;

if(pos <= mid){

lc[newroot] = update\_segtree(lc[Lroot], l, mid, pos);

rc[newroot] = rc[Lroot];

}

else{

lc[newroot] = lc[Lroot];

rc[newroot] = update\_segtree(rc[Lroot], mid+1, r, pos);

}

}

return newroot;

}

void dfs\_update(int child, int fa){//主席树的dfs更新操作，用子节点的树通过父节点的树来建，与父节点的树公用节点。

int hash = lower\_bound(valc, valc+len, val[child-1])-valc;//这样的话，每个节点存储的是从根节点到当前节点的修改操作后的结果。查询操作好维护。

tr[child] = update\_segtree(tr[fa], 0, len-1, hash);

for (int i = head[child]; i != -1; i=edge[i].next){

int to = edge[i].to;

if(to == fa) continue;

dfs\_update(to, child);

}

}

int query\_segtree(int one, int two, int father, int l, int r, int rk, int fatherhash){//查询u，v路径上的第k小。

int mid = (l+r)>>1;

if(l == r) return l;

int tmp = sum[lc[one]] - sum[lc[father]] + sum[lc[two]] - sum[lc[father]];//u，v路径上不包括公共祖先的点的线段树在左子树上的结果。与静态区间第k大思路相仿。

if(fatherhash >= l && fatherhash <= mid) tmp++;//看公共祖先这个点的哈希值是否应该包含在线段树的左树上，是就加上

if(tmp >= rk) return query\_segtree(lc[one], lc[two], lc[father], l, mid, rk, fatherhash);//如果在排名在左子树上的点的数量大于等于k，左子上寻找

else return query\_segtree(rc[one], rc[two], rc[father], mid+1, r, rk-tmp, fatherhash);//否则就在右子上寻找

}

//------------------------------------------------

//----------------lca dynamic------------------kuangbin模板

int rmq[2\*maxn];

struct ST{

int mm[2\*maxn];

int dp[2\*maxn][20];

void init(int n){

mm[0] = -1;

for (int i = 1; i <= n; i++){

mm[i] = ((i&(i-1)) == 0) ? mm[i-1]+1 : mm[i-1];

dp[i][0] = i;

}

for (int j = 1; j <= mm[n]; j++)

for (int i = 1; i + (1<<j)-1 <= n; i++)

dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1<<(j-1))][j-1]] ? dp[i][j-1]:dp[i+(1<<(j-1))][j-1];

}

int query(int a, int b){

if(a > b) swap(a, b);

int k = mm[b-a+1];

return rmq[dp[a][k]] <= rmq[dp[b-(1<<k)+1][k]] ? dp[a][k] : dp[b-(1<<k)+1][k];

}

};

int F[maxn\*2];//欧拉序列

int P[maxn];

int cnt;

ST st;

void dfs\_lca(int u, int pre, int dep){//求三个序列

F[++cnt] = u;

rmq[cnt] = dep;

P[u] = cnt;

for (int i = head[u]; i != -1; i = edge[i].next){

int v = edge[i].to;

if(v == pre) continue;

dfs\_lca(v, u, dep+1);

F[++cnt] = u;

rmq[cnt] = dep;

}

}

void init\_lca(int root, int node\_num){

cnt = 0;

dfs\_lca(root, root, 0);

st.init(2\*node\_num-1);

}

int query\_lca(int u, int v){

return F[st.query(P[u],P[v])];

}

//--------------------------------------------

int main()

{

while(~scanf("%d%d", &n, &m)){

int pre = 0;//题目要求，要求记录前一个值，并且要与下一个询问的前边的那个范围做异或操作

for (int i = 0; i < n; i++) val[i] = iread();

init\_segtree();//建树

tr[0] = build\_segtree(0, len-1);

init\_edge();//读图

for (int i = 0; i < n-1; i++){

u = iread();

v = iread();

add\_edge(u, v);

add\_edge(v, u);

}

dfs\_update(1, 1);//继续建主席树

init\_lca(1, n);//lca询问初始化

while(m--){

x = iread(),y = iread(), z = iread();

x = x^pre;

int father = query\_lca(x, y);

int fatherhash = lower\_bound(valc, valc+len, val[father-1])-valc;

pre = valc[query\_segtree(tr[x], tr[y], tr[father], 0, len-1, z, fatherhash)];

printf("%d", pre);

if(m != 0) printf("\n");

}

}

return 0;

}

# 字符串——exkmp

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#define next nxt

using namespace std;

typedef long long ll;

int next[50010];

int ex[50010];

void exkmp(char s1[], char s2[], int next[], int ex[]){

int i, j, p;

for (i = 0, j = 0, p = -1; s1[i] != '\0'; i++, j++, p--){

if(p == -1){

j = 0;

do

p++;

while(s1[i+p] != '\0' && s1[i+p] == s2[j+p]);

ex[i] = p;

}

else if(next[j] < p)

ex[i] = next[j];

else if(next[j] > p)

ex[i] = p;

else{

j = 0;

while(s1[i + p] != '\0' && s1[i+p] == s2[j+p]) p++;

ex[i] = p;

}

}

ex[i] = 0;

}

int main()

{

char s1\_p[50010];

char s2\_s[50010];

while(~scanf("%s", s1\_p)){

scanf("%s", s2\_s);

next[0] = 0;

exkmp(s1\_p + 1, s1\_p, next, next+1);

exkmp(s2\_s, s1\_p, next, ex);

int len = strlen(s2\_s);

int mx = 0, mx\_index = -1;

for (int i = 0; i <= len; i++){

if(ex[i] == len - i){

if(ex[i] > mx){

mx = ex[i];

mx\_index = i;

}

}

}

if(mx == 0){

printf("0\n");

}

else{

for (int i = mx\_index; i < len; i++){

printf("%c", s2\_s[i]);

}

printf(" %d\n", mx);

}

}

return 0;

}