Олимпиада СПбГУ по информатике 2022/23 учебного года

A	B	C	D	E	F	Sum
100	100	80	60	28	10	378

Task A ()

```
//#include <bits/stdc++.h>
#include <deque>
#include <iostream>
#include <random>
#include <vector>
#include <algorithm>
#include <bitset>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <map>
#include <cmath>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <fstream>
#include <random>
#include <queue>
#include <complex>
#include <ctime>

/*#ifdef DEBUG
    #define _GLIBCCX_DEBUG
#endif*/

//#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
/*#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
```

```
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")*/

using namespace std;
using namespace __gnu_pbds;

//#define int unsigned long long
#define int long long
//#define int __int128
//#define int unsigned int
//typedef long long ll;
//typedef long double ld;
#define ll long long
#define double long double
typedef complex<double> cld;

typedef tree<
        pair<int, int>,
        null_type,
        less< pair <int, int> >,
        rb_tree_tag,
        tree_order_statistics_node_update
> ordered_set;

template<typename T>
ostream &operator<<(ostream &ws, const vector<T> &v) {
    for (auto e: v) {
        ws << e << " ";
    }
    return ws;
}

//const int mod = 998244353;
const ll mod = 1000000007;
//const int mod = 1000000009;
//const int mod = 11019929;
mt19937 rnd;
//(int32_t)mt19937(0)()

/*int gcd(int a, int b)
{
    while(b > 0){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
    //if (b == 0) return a;
    //return gcd(b, a % b);
}*/

/*#define abs abs1

int abs1(int x){
    return x < 0 ? -x : x;
}*/

struct seg_tr {
    vector<int> t, p;

    void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
```

```
    }

    void update(int v, int l, int r, int tl, int tr, int val) {
        if (tl >= r || tr <= l) {
            return;
        }
        if (tl >= l && tr <= r) {
            t[v] += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return -1;
        }
        if (tl >= l && tr <= r && t[v] >= 0) {
            return -1;
        }
        if (tl + 1 == tr) {
            return tl;
        }

        push(v);
        int tm = (tl + tr) / 2;
        int get = quer(v * 2 + 1, l, r, tm, tr);
        if (get == -1) {
            return quer(v * 2, l, r, tl, tm);
        }
        return get;
    }

    /*int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return -1;
        }
        if(tl >= l && tr <= r && t[v] > 0){
            return -1;
        }

        if(tl + 1 == tr){
            return tl;
        }

        int tm = (tl + tr) / 2;
        int get = quer(v * 2, l, r, tl, tm);
        if(get == -1){
            return quer(v * 2 + 1, l, r, tm, tr);
        }
        return get;
    }*/
};

/*struct merge_tree{
    vector < ordered_set > t;

    void update(int v, int pos, int tl, int tr, int val){
        if(tl > pos || tr <= pos){
            return;
        }

        if(tl + 1 == tr && tl == pos){
            t[v].insert({val, k});
            ++k;
            return;
        }
```

```
        t[v].insert({val, k});
        ++k;

        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }

        if(tl >= l && tr <= r){
            return t[v].size() - t[v].order_of_key({x, 1e9});
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};*/

/*struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int>& h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x, int y){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return upper_bound(t[v].begin(), t[v].end(), y) - lower_bound(t[v].begin(), t[v].end()
                , x);
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x, y) + quer(v * 2 + 1, l, r, tm, tr, x, y);
    }
};*/

struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int> &h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }
```

```
        if(tl >= l && tr <= r){
            return lower_bound(t[v].begin(), t[v].end(), x) - t[v].begin();
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};

/*int n;

struct fenvik{
    vector <int> t;

    int quer(int pos){
        if(pos == -1){
            return 0;
        }

        int sm = 0;
        for(int i = pos; i >= 0; i = (i & (i + 1)) - 1){
            sm += t[i];
        }
        return sm;
    }

    void update(int pos, int val){
        for(int i = pos; i < n; i = (i | (i + 1))){
            t[i] += val;
        }
    }
};*/

struct val {
    int l, r, val;
};

const double pi = 3.14159265358979323846;

int binpow(int x, int k) {
    if (k == 0) {
        return 1;
    }
    int y = binpow(x, k / 2);
    if (k % 2 == 0) {
        return (y * y) % mod;
    } else {
        return (((y * y) % mod) * x) % mod;
    }
}

/*struct pers_seg_tree {
    vector<val> t;
    int id = 0;

    void build(int v, int tl, int tr) {
        id = max(id, v);
        if (tl + 1 == tr) {
            t[v] = {-1, -1, 0};
            return;
        }

        t[v] = {v * 2, v * 2 + 1, 0};
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm);
        build(v * 2 + 1, tm, tr);
    }

    int update(int v, int pos, int tl, int tr, int val) {
        if (tl == pos && tl + 1 == tr) {
            //t.push_back({-1, -1, t[v].val + val});
            //return t.size() - 1;
            t[id] = {-1, -1, t[v].val + val};
            id++;
```

```
            return id - 1;
        }

        int tm = (tl + tr) / 2;
        if (pos < tm) {
            int res = update(t[v].l, pos, tl, tm, val);
            //t.push_back({res, t[v].r, t[res].val + t[t[v].r].val});
            //return t.size() - 1;
            t[id] = {res, t[v].r, t[res].val + t[t[v].r].val};
            id++;
            return id - 1;
        } else {
            int res = update(t[v].r, pos, tm, tr, val);
            //t.push_back({t[v].l, res, t[t[v].l].val + t[res].val});
            //return t.size() - 1;
            t[id] = {t[v].l, res, t[t[v].l].val + t[res].val};
            id++;
            return id - 1;
        }
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v].val;
        }

        int tm = (tl + tr) / 2;
        return quer(t[v].l, l, r, tl, tm) + quer(t[v].r, l, r, tm, tr);
    }

    int find_k(int vl, int vr, int tl, int tr, int k) {
        if (tl + 1 == tr) {
            return tl;
        }

        int tm = (tl + tr) / 2;

        int kol = t[t[vr].l].val - t[t[vl].l].val;

        if (kol >= k) {
            return find_k(t[vl].l, t[vr].l, tl, tm, k);
        } else {
            return find_k(t[vl].r, t[vr].r, tm, tr, k - kol);
        }
    }
};*/

struct elem {
    int l, r, g;
};

double rand_double() {
    return (double) rand() / RAND_MAX;
}

/*struct gcd_seg_tree{
    vector <elem> t;
    vector <int> p;

    void push(int v){
        if(p[v] != 0){
            t[v * 2].l += p[v];
            t[v * 2 + 1].l += p[v];
            t[v * 2].r += p[v];
            t[v * 2 + 1].r += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }
```

```
    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = {h[tl], h[tl], 0};
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    void update(int v, int l, int r, int tl, int tr, int val){
        if(tl >= r || tr <= l){
            return;
        }
        if(tl >= l && tr <= r){
            t[v].l += val;
            t[v].r += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return gcd(t[v].l, t[v].g);
        }

        push(v);
        int tm = (tl + tr) / 2;
        return gcd(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
    }
};*/

/*int C(int x, int k){
    if(x < 0 || k < 0 || k > x){
        return 0;
    }

    return (((f[x] * fr[k]) % mod) * fr[x - k]) % mod;
}*/

struct seg_tree {
    vector <int> t;
    //vector <int> p;

    /*void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }*/

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = h[tl];
            return;
        }
```

```
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    void update(int v, int pos, int tl, int tr, int val) {
        if (tl > pos || tr <= pos) {
            return;
        }
        if (tl == pos && tl + 1 == tr) {
            t[v] = val;
            //p[v] += val;
            return;
        }

        //push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v];
        }

        //push(v);
        int tm = (tl + tr) / 2;
        //return min(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
        return quer(v * 2, l, r, tl, tm) + quer(v * 2 + 1, l, r, tm, tr);
    }
};

void solve() {
    int n = 6;

    vector <int> a(n);
    for(int i = 0; i < n; ++i){
        cin >> a[i];
    }

    vector <int> p = {0, 1, 2, 3, 4, 5};
    int f = 1 * 2 * 3 * 4 * 5 * 6;
    for(int _ = 0; _ < f; ++_){
        vector <int> b;
        bool ok = 1;
        for(int i = 0; i < n; ++i){
            int k = 1;
            for(auto x : b){
                if(x < p[i]){
                    ++k;
                }
            }
            if(k != a[i]){
                ok = 0;
                break;
            }
            b.push_back(p[i]);
        }

        if(ok){
            for(auto x : p){
                cout << x + 1 << "␣";
            }
            return;
        }
```

```
        next_permutation(p.begin(), p.end());
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cout.precision(20);
#ifndef DEBUG
    //freopen("schools.in", "r", stdin);
    //freopen("schools.out", "w", stdout);
#endif

    //freopen("output.txt", "w", stdout);

    //solve();

    ll tests = 1;
    //cin >> tests;
    while (tests--) {
        solve();
    }
}
```

Task B ()

```
//#include <bits/stdc++.h>
#include <deque>
#include <iostream>
#include <random>
#include <vector>
#include <algorithm>
#include <bitset>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <map>
#include <cmath>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <fstream>
#include <random>
#include <queue>
#include <complex>
#include <ctime>

/*#ifdef DEBUG
    #define _GLIBCCX_DEBUG
#endif*/

//#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
/*#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")*/

using namespace std;
using namespace __gnu_pbds;

//#define int unsigned long long
#define int long long
//#define int __int128
//#define int unsigned int
```

```
//typedef long long ll;
//typedef long double ld;
#define ll long long
#define double long double
typedef complex<double> cld;

typedef tree<
        pair<int, int>,
        null_type,
        less< pair <int, int> >,
        rb_tree_tag,
        tree_order_statistics_node_update
> ordered_set;

template<typename T>
ostream &operator<<(ostream &ws, const vector<T> &v) {
    for (auto e: v) {
        ws << e << " ";
    }
    return ws;
}

//const int mod = 998244353;
const ll mod = 1000000007;
//const int mod = 1000000009;
//const int mod = 11019929;
mt19937 rnd;
//(int32_t)mt19937(0)()

/*int gcd(int a, int b)
{
    while(b > 0){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
    //if (b == 0) return a;
    //return gcd(b, a % b);
}*/

/*#define abs abs1

int abs1(int x){
    return x < 0 ? -x : x;
}*/

struct seg_tr {
    vector<int> t, p;

    void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void update(int v, int l, int r, int tl, int tr, int val) {
        if (tl >= r || tr <= l) {
            return;
        }
        if (tl >= l && tr <= r) {
            t[v] += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
```

```
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return -1;
        }
        if (tl >= l && tr <= r && t[v] >= 0) {
            return -1;
        }
        if (tl + 1 == tr) {
            return tl;
        }

        push(v);
        int tm = (tl + tr) / 2;
        int get = quer(v * 2 + 1, l, r, tm, tr);
        if (get == -1) {
            return quer(v * 2, l, r, tl, tm);
        }
        return get;
    }

    /*int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return -1;
        }
        if(tl >= l && tr <= r && t[v] > 0){
            return -1;
        }

        if(tl + 1 == tr){
            return tl;
        }

        int tm = (tl + tr) / 2;
        int get = quer(v * 2, l, r, tl, tm);
        if(get == -1){
            return quer(v * 2 + 1, l, r, tm, tr);
        }
        return get;
    }*/
};

/*struct merge_tree{
    vector < ordered_set > t;

    void update(int v, int pos, int tl, int tr, int val){
        if(tl > pos || tr <= pos){
            return;
        }

        if(tl + 1 == tr && tl == pos){
            t[v].insert({val, k});
            ++k;
            return;
        }

        t[v].insert({val, k});
        ++k;

        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }

        if(tl >= l && tr <= r){
            return t[v].size() - t[v].order_of_key({x, 1e9});
        }
```

```
        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};*/

/*struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int>& h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x, int y){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return upper_bound(t[v].begin(), t[v].end(), y) - lower_bound(t[v].begin(), t[v].end()
                , x);
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x, y) + quer(v * 2 + 1, l, r, tm, tr, x, y);
    }
};*/

struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int> &h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return lower_bound(t[v].begin(), t[v].end(), x) - t[v].begin();
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};

/*int n;

struct fenvik{
    vector <int> t;

    int quer(int pos){
        if(pos == -1){
```

```
            return 0;
        }

        int sm = 0;
        for(int i = pos; i >= 0; i = (i & (i + 1)) - 1){
            sm += t[i];
        }
        return sm;
    }

    void update(int pos, int val){
        for(int i = pos; i < n; i = (i | (i + 1))){
            t[i] += val;
        }
    }
};*/

struct val {
    int l, r, val;
};

const double pi = 3.14159265358979323846;

int binpow(int x, int k) {
    if (k == 0) {
        return 1;
    }
    int y = binpow(x, k / 2);
    if (k % 2 == 0) {
        return (y * y) % mod;
    } else {
        return (((y * y) % mod) * x) % mod;
    }
}

/*struct pers_seg_tree {
    vector<val> t;
    int id = 0;

    void build(int v, int tl, int tr) {
        id = max(id, v);
        if (tl + 1 == tr) {
            t[v] = {-1, -1, 0};
            return;
        }

        t[v] = {v * 2, v * 2 + 1, 0};
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm);
        build(v * 2 + 1, tm, tr);
    }

    int update(int v, int pos, int tl, int tr, int val) {
        if (tl == pos && tl + 1 == tr) {
            //t.push_back({-1, -1, t[v].val + val});
            //return t.size() - 1;
            t[id] = {-1, -1, t[v].val + val};
            id++;
            return id - 1;
        }

        int tm = (tl + tr) / 2;
        if (pos < tm) {
            int res = update(t[v].l, pos, tl, tm, val);
            //t.push_back({res, t[v].r, t[res].val + t[t[v].r].val});
            //return t.size() - 1;
            t[id] = {res, t[v].r, t[res].val + t[t[v].r].val};
            id++;
            return id - 1;
        } else {
            int res = update(t[v].r, pos, tm, tr, val);
            //t.push_back({t[v].l, res, t[t[v].l].val + t[res].val});
            //return t.size() - 1;
            t[id] = {t[v].l, res, t[t[v].l].val + t[res].val};
```

```
            id++;
            return id - 1;
        }
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v].val;
        }

        int tm = (tl + tr) / 2;
        return quer(t[v].l, l, r, tl, tm) + quer(t[v].r, l, r, tm, tr);
    }

    int find_k(int vl, int vr, int tl, int tr, int k) {
        if (tl + 1 == tr) {
            return tl;
        }

        int tm = (tl + tr) / 2;

        int kol = t[t[vr].l].val - t[t[vl].l].val;

        if (kol >= k) {
            return find_k(t[vl].l, t[vr].l, tl, tm, k);
        } else {
            return find_k(t[vl].r, t[vr].r, tm, tr, k - kol);
        }
    }
};*/

struct elem {
    int l, r, g;
};

double rand_double() {
    return (double) rand() / RAND_MAX;
}

/*struct gcd_seg_tree{
    vector <elem> t;
    vector <int> p;

    void push(int v){
        if(p[v] != 0){
            t[v * 2].l += p[v];
            t[v * 2 + 1].l += p[v];
            t[v * 2].r += p[v];
            t[v * 2 + 1].r += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = {h[tl], h[tl], 0};
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    void update(int v, int l, int r, int tl, int tr, int val){
        if(tl >= r || tr <= l){
            return;
```

```
        }
        if(tl >= l && tr <= r){
            t[v].l += val;
            t[v].r += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return gcd(t[v].l, t[v].g);
        }

        push(v);
        int tm = (tl + tr) / 2;
        return gcd(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
    }
};*/

/*int C(int x, int k){
    if(x < 0 || k < 0 || k > x){
        return 0;
    }

    return (((f[x] * fr[k]) % mod) * fr[x - k]) % mod;
}*/

struct seg_tree {
    vector <int> t;
    //vector <int> p;

    /*void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }*/

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = h[tl];
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    void update(int v, int pos, int tl, int tr, int val) {
        if (tl > pos || tr <= pos) {
            return;
        }
        if (tl == pos && tl + 1 == tr) {
            t[v] = val;
            //p[v] += val;
            return;
        }
```

```
        //push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v];
        }

        //push(v);
        int tm = (tl + tr) / 2;
        //return min(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
        return quer(v * 2, l, r, tl, tm) + quer(v * 2 + 1, l, r, tm, tr);
    }
};

void solve() {
    string s;
    cin >> s;
    if(s == "first"){
        int n;
        cin >> n;
        vector <int> a(n);
        int sm = 0;
        for(int i = 0; i < n; ++i){
            cin >> a[i];
            sm += a[i];
        }

        cout << (sm << 30ll);
    }else{
        int n;
        cin >> n;
        vector <int> a(n);
        int sm = 0;
        for(int i = 0; i < n; ++i){
            cin >> a[i];

            if(i == 0) {
                sm += (a[i] >> 30ll);
            }

            sm += (a[i] % (1 << 30ll));
        }

        cout << sm;
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cout.precision(20);
#ifndef DEBUG
    //freopen("schools.in", "r", stdin);
    //freopen("schools.out", "w", stdout);
#endif

    //freopen("output.txt", "w", stdout);

    //solve();

    ll tests = 1;
    //cin >> tests;
    while (tests--) {
```

```
            solve();
        }
    }
```

Task C ()

```
//#include <bits/stdc++.h>
#include <deque>
#include <iostream>
#include <random>
#include <vector>
#include <algorithm>
#include <bitset>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <map>
#include <cmath>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <fstream>
#include <random>
#include <queue>
#include <complex>
#include <ctime>

/*#ifdef DEBUG
    #define _GLIBCCX_DEBUG
#endif*/

//#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
/*#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")*/

using namespace std;
using namespace __gnu_pbds;

//#define int unsigned long long
#define int long long
//#define int __int128
//#define int unsigned int
```

```
//typedef long long ll;
//typedef long double ld;
#define ll long long
#define double long double
typedef complex<double> cld;

typedef tree<
        pair<int, int>,
        null_type,
        less< pair <int, int> >,
        rb_tree_tag,
        tree_order_statistics_node_update
> ordered_set;

template<typename T>
ostream &operator<<(ostream &ws, const vector<T> &v) {
    for (auto e: v) {
        ws << e << " ";
    }
    return ws;
}

//const int mod = 998244353;
const ll mod = 1000000007;
//const int mod = 1000000009;
//const int mod = 11019929;
mt19937 rnd;
//(int32_t)mt19937(0)()

/*int gcd(int a, int b)
{
    while(b > 0){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
    //if (b == 0) return a;
    //return gcd(b, a % b);
}*/

/*#define abs abs1

int abs1(int x){
    return x < 0 ? -x : x;
}*/

struct seg_tr {
    vector<int> t, p;

    void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void update(int v, int l, int r, int tl, int tr, int val) {
        if (tl >= r || tr <= l) {
            return;
        }
        if (tl >= l && tr <= r) {
            t[v] += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
```

```
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return -1;
        }
        if (tl >= l && tr <= r && t[v] >= 0) {
            return -1;
        }
        if (tl + 1 == tr) {
            return tl;
        }

        push(v);
        int tm = (tl + tr) / 2;
        int get = quer(v * 2 + 1, l, r, tm, tr);
        if (get == -1) {
            return quer(v * 2, l, r, tl, tm);
        }
        return get;
    }

    /*int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return -1;
        }
        if(tl >= l && tr <= r && t[v] > 0){
            return -1;
        }

        if(tl + 1 == tr){
            return tl;
        }

        int tm = (tl + tr) / 2;
        int get = quer(v * 2, l, r, tl, tm);
        if(get == -1){
            return quer(v * 2 + 1, l, r, tm, tr);
        }
        return get;
    }*/
};

/*struct merge_tree{
    vector < ordered_set > t;

    void update(int v, int pos, int tl, int tr, int val){
        if(tl > pos || tr <= pos){
            return;
        }

        if(tl + 1 == tr && tl == pos){
            t[v].insert({val, k});
            ++k;
            return;
        }

        t[v].insert({val, k});
        ++k;

        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }

        if(tl >= l && tr <= r){
            return t[v].size() - t[v].order_of_key({x, 1e9});
        }
```

```
        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};*/

/*struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int>& h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x, int y){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return upper_bound(t[v].begin(), t[v].end(), y) - lower_bound(t[v].begin(), t[v].end()
                , x);
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x, y) + quer(v * 2 + 1, l, r, tm, tr, x, y);
    }
};*/

struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int> &h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return lower_bound(t[v].begin(), t[v].end(), x) - t[v].begin();
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};

/*int n;

struct fenvik{
    vector <int> t;

    int quer(int pos){
        if(pos == -1){
```

```
            return 0;
        }

        int sm = 0;
        for(int i = pos; i >= 0; i = (i & (i + 1)) - 1){
            sm += t[i];
        }
        return sm;
    }

    void update(int pos, int val){
        for(int i = pos; i < n; i = (i | (i + 1))){
            t[i] += val;
        }
    }
};*/

struct val {
    int l, r, val;
};

const double pi = 3.14159265358979323846;

int binpow(int x, int k) {
    if (k == 0) {
        return 1;
    }
    int y = binpow(x, k / 2);
    if (k % 2 == 0) {
        return (y * y) % mod;
    } else {
        return (((y * y) % mod) * x) % mod;
    }
}

/*struct pers_seg_tree {
    vector<val> t;
    int id = 0;

    void build(int v, int tl, int tr) {
        id = max(id, v);
        if (tl + 1 == tr) {
            t[v] = {-1, -1, 0};
            return;
        }

        t[v] = {v * 2, v * 2 + 1, 0};
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm);
        build(v * 2 + 1, tm, tr);
    }

    int update(int v, int pos, int tl, int tr, int val) {
        if (tl == pos && tl + 1 == tr) {
            //t.push_back({-1, -1, t[v].val + val});
            //return t.size() - 1;
            t[id] = {-1, -1, t[v].val + val};
            id++;
            return id - 1;
        }

        int tm = (tl + tr) / 2;
        if (pos < tm) {
            int res = update(t[v].l, pos, tl, tm, val);
            //t.push_back({res, t[v].r, t[res].val + t[t[v].r].val});
            //return t.size() - 1;
            t[id] = {res, t[v].r, t[res].val + t[t[v].r].val};
            id++;
            return id - 1;
        } else {
            int res = update(t[v].r, pos, tm, tr, val);
            //t.push_back({t[v].l, res, t[t[v].l].val + t[res].val});
            //return t.size() - 1;
            t[id] = {t[v].l, res, t[t[v].l].val + t[res].val};
```

```
            id++;
            return id - 1;
        }
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v].val;
        }

        int tm = (tl + tr) / 2;
        return quer(t[v].l, l, r, tl, tm) + quer(t[v].r, l, r, tm, tr);
    }

    int find_k(int vl, int vr, int tl, int tr, int k) {
        if (tl + 1 == tr) {
            return tl;
        }

        int tm = (tl + tr) / 2;

        int kol = t[t[vr].l].val - t[t[vl].l].val;

        if (kol >= k) {
            return find_k(t[vl].l, t[vr].l, tl, tm, k);
        } else {
            return find_k(t[vl].r, t[vr].r, tm, tr, k - kol);
        }
    }
};*/

struct elem {
    int l, r, g;
};

double rand_double() {
    return (double) rand() / RAND_MAX;
}

/*struct gcd_seg_tree{
    vector <elem> t;
    vector <int> p;

    void push(int v){
        if(p[v] != 0){
            t[v * 2].l += p[v];
            t[v * 2 + 1].l += p[v];
            t[v * 2].r += p[v];
            t[v * 2 + 1].r += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = {h[tl], h[tl], 0};
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    void update(int v, int l, int r, int tl, int tr, int val){
        if(tl >= r || tr <= l){
            return;
```

```
        }
        if(tl >= l && tr <= r){
            t[v].l += val;
            t[v].r += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return gcd(t[v].l, t[v].g);
        }

        push(v);
        int tm = (tl + tr) / 2;
        return gcd(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
    }
};*/

/*int C(int x, int k){
    if(x < 0 || k < 0 || k > x){
        return 0;
    }

    return (((f[x] * fr[k]) % mod) * fr[x - k]) % mod;
}*/

struct seg_tree {
    vector <int> t;
    //vector <int> p;

    /*void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }*/

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = h[tl];
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    void update(int v, int pos, int tl, int tr, int val) {
        if (tl > pos || tr <= pos) {
            return;
        }
        if (tl == pos && tl + 1 == tr) {
            t[v] = val;
            //p[v] += val;
            return;
        }
```

```
        //push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v];
        }

        //push(v);
        int tm = (tl + tr) / 2;
        //return min(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
        return quer(v * 2, l, r, tl, tm) + quer(v * 2 + 1, l, r, tm, tr);
    }
};

void solve() {
    vector < pair <int, int> > b(3);

    set < pair <int, int> > s;
    auto add = [&](int x, int y){
        if(x > y){
            swap(x, y);
        }

        s.insert({x, y});
    };

    for(int i = 0; i < 3; ++i){
        cin >> b[i].first >> b[i].second;
        add(b[i].first, b[i].second);
    }

    vector <int> p = {0, 1, 2};
    for(int _ = 0; _ < 6; ++_) {
        vector < pair <int, int > > a(3);
        for(int i = 0; i < 3; ++i){
            a[i] = b[p[i]];
        }

        for (int mask = 0; mask < 8; ++mask) {
            for (int i = 0; i < 3; ++i) {
                if ((mask >> i) & 1) {
                    swap(a[i].first, a[i].second);
                }
            }

            int x1 = a[0].first;
            int y1 = a[0].second;
            int x2 = a[1].first;
            int y2 = a[1].second;
            int x3 = a[2].first;
            int y3 = a[2].second;

            if(x2 <= x1 && y2 <= y1){
                if(x2 == x1){
                    add(x1, y1 - y2);
                    if(x3 == x2){
                        /*if(y3 < y2){
                            add(x1, y2 - y3);
                        }*/
                        if(y3 < y1 - y2){
                            add(x1, y1 - y2 - y3);
                        }
                    }
                    /*if(y3 == y2){
```

```
                        if(x3 < x2){
                            add(x2 - x3, y3);
                        }
                    }*/
                    if(y3 == y1 - y2){
                        if(x3 < x2){
                            add(x2 - x3, y3);
                        }
                    }
                }else{
                    if(x3 == x2 && y1 - y2 == y3){
                        add(x1 - x2, y1);
                    }
                    if(x3 <= x1 && y3 <= y1 - y2 && x2 + x3 - 1 >= x1){
                        add(x1 - x2, y2);
                        if(y3 == y1 - y2){
                            add(x1 - x3, y1 - y2);
                        }
                    }
                    if(x3 == x1 && y3 < y1 - y2){
                        add(x1, y1 - y2 - y3);
                    }
                    /*if(x2 + x3 == x1 && y2 == y3 && y2 <= y1){
                        add(x1, y1 - y2);
                    }
                    if(y3 <= y1 && y3 + y2 - 1 >= y1 && x3 <= x1 - x2){
                        add(x2, y1 - y2);
                        if(x3 == x1 - x2){
                            add(x1 - x2, y1 - y3);
                        }
                    }
                    if(y3 == y1 && x3 < x1 - x2){
                        add(x1 - x2 - x3, y1);
                    }*/
                }
            }

            for (int i = 0; i < 3; ++i) {
                if ((mask >> i) & 1) {
                    swap(a[i].first, a[i].second);
                }
            }
        }

        next_permutation(p.begin(), p.end());
    }

    for(auto x : s){
        if(x.first <= 0 || x.second <= 0){
            continue;
        }
        cout << x.first << " " << x.second << '\n';
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cout.precision(20);
#ifndef DEBUG
    //freopen("schools.in", "r", stdin);
    //freopen("schools.out", "w", stdout);
#endif

    //freopen("output.txt", "w", stdout);

    //solve();

    ll tests = 1;
    //cin >> tests;
    while (tests--) {
        solve();
    }
```

```
}
```

Task D ()

```
//#include <bits/stdc++.h>
#include <deque>
#include <iostream>
#include <random>
#include <vector>
#include <algorithm>
#include <bitset>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <map>
#include <cmath>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <fstream>
#include <random>
#include <queue>
#include <complex>
#include <ctime>

/*#ifdef DEBUG
    #define _GLIBCCX_DEBUG
#endif*/

//#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
/*#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")*/

using namespace std;
using namespace __gnu_pbds;

//#define int unsigned long long
#define int long long
//#define int __int128
//#define int unsigned int
```

```
//typedef long long ll;
//typedef long double ld;
#define ll long long
#define double long double
typedef complex<double> cld;

typedef tree<
        pair<int, int>,
        null_type,
        less< pair <int, int> >,
        rb_tree_tag,
        tree_order_statistics_node_update
> ordered_set;

template<typename T>
ostream &operator<<(ostream &ws, const vector<T> &v) {
    for (auto e: v) {
        ws << e << " ";
    }
    return ws;
}

//const int mod = 998244353;
const ll mod = 1000000007;
//const int mod = 1000000009;
//const int mod = 11019929;
mt19937 rnd;
//(int32_t)mt19937(0)()

/*int gcd(int a, int b)
{
    while(b > 0){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
    //if (b == 0) return a;
    //return gcd(b, a % b);
}*/

/*#define abs abs1

int abs1(int x){
    return x < 0 ? -x : x;
}*/

struct seg_tr {
    vector<int> t, p;

    void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void update(int v, int l, int r, int tl, int tr, int val) {
        if (tl >= r || tr <= l) {
            return;
        }
        if (tl >= l && tr <= r) {
            t[v] += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
```

```
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return -1;
        }
        if (tl >= l && tr <= r && t[v] >= 0) {
            return -1;
        }
        if (tl + 1 == tr) {
            return tl;
        }

        push(v);
        int tm = (tl + tr) / 2;
        int get = quer(v * 2 + 1, l, r, tm, tr);
        if (get == -1) {
            return quer(v * 2, l, r, tl, tm);
        }
        return get;
    }

    /*int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return -1;
        }
        if(tl >= l && tr <= r && t[v] > 0){
            return -1;
        }

        if(tl + 1 == tr){
            return tl;
        }

        int tm = (tl + tr) / 2;
        int get = quer(v * 2, l, r, tl, tm);
        if(get == -1){
            return quer(v * 2 + 1, l, r, tm, tr);
        }
        return get;
    }*/
};

/*struct merge_tree{
    vector < ordered_set > t;

    void update(int v, int pos, int tl, int tr, int val){
        if(tl > pos || tr <= pos){
            return;
        }

        if(tl + 1 == tr && tl == pos){
            t[v].insert({val, k});
            ++k;
            return;
        }

        t[v].insert({val, k});
        ++k;

        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }

        if(tl >= l && tr <= r){
            return t[v].size() - t[v].order_of_key({x, 1e9});
        }
```

```
        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};*/

/*struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int>& h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x, int y){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return upper_bound(t[v].begin(), t[v].end(), y) - lower_bound(t[v].begin(), t[v].end()
                , x);
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x, y) + quer(v * 2 + 1, l, r, tm, tr, x, y);
    }
};*/

struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int> &h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return lower_bound(t[v].begin(), t[v].end(), x) - t[v].begin();
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};

/*int n;

struct fenvik{
    vector <int> t;

    int quer(int pos){
        if(pos == -1){
```

```
            return 0;
        }

        int sm = 0;
        for(int i = pos; i >= 0; i = (i & (i + 1)) - 1){
            sm += t[i];
        }
        return sm;
    }

    void update(int pos, int val){
        for(int i = pos; i < n; i = (i | (i + 1))){
            t[i] += val;
        }
    }
};*/

struct val {
    int l, r, val;
};

const double pi = 3.14159265358979323846;

int binpow(int x, int k) {
    if (k == 0) {
        return 1;
    }
    int y = binpow(x, k / 2);
    if (k % 2 == 0) {
        return (y * y) % mod;
    } else {
        return (((y * y) % mod) * x) % mod;
    }
}

/*struct pers_seg_tree {
    vector<val> t;
    int id = 0;

    void build(int v, int tl, int tr) {
        id = max(id, v);
        if (tl + 1 == tr) {
            t[v] = {-1, -1, 0};
            return;
        }

        t[v] = {v * 2, v * 2 + 1, 0};
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm);
        build(v * 2 + 1, tm, tr);
    }

    int update(int v, int pos, int tl, int tr, int val) {
        if (tl == pos && tl + 1 == tr) {
            //t.push_back({-1, -1, t[v].val + val});
            //return t.size() - 1;
            t[id] = {-1, -1, t[v].val + val};
            id++;
            return id - 1;
        }

        int tm = (tl + tr) / 2;
        if (pos < tm) {
            int res = update(t[v].l, pos, tl, tm, val);
            //t.push_back({res, t[v].r, t[res].val + t[t[v].r].val});
            //return t.size() - 1;
            t[id] = {res, t[v].r, t[res].val + t[t[v].r].val};
            id++;
            return id - 1;
        } else {
            int res = update(t[v].r, pos, tm, tr, val);
            //t.push_back({t[v].l, res, t[t[v].l].val + t[res].val});
            //return t.size() - 1;
            t[id] = {t[v].l, res, t[t[v].l].val + t[res].val};
```

```
            id++;
            return id - 1;
        }
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v].val;
        }

        int tm = (tl + tr) / 2;
        return quer(t[v].l, l, r, tl, tm) + quer(t[v].r, l, r, tm, tr);
    }

    int find_k(int vl, int vr, int tl, int tr, int k) {
        if (tl + 1 == tr) {
            return tl;
        }

        int tm = (tl + tr) / 2;

        int kol = t[t[vr].l].val - t[t[vl].l].val;

        if (kol >= k) {
            return find_k(t[vl].l, t[vr].l, tl, tm, k);
        } else {
            return find_k(t[vl].r, t[vr].r, tm, tr, k - kol);
        }
    }
};*/

struct elem {
    int l, r, g;
};

double rand_double() {
    return (double) rand() / RAND_MAX;
}

/*struct gcd_seg_tree{
    vector <elem> t;
    vector <int> p;

    void push(int v){
        if(p[v] != 0){
            t[v * 2].l += p[v];
            t[v * 2 + 1].l += p[v];
            t[v * 2].r += p[v];
            t[v * 2 + 1].r += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = {h[tl], h[tl], 0};
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    void update(int v, int l, int r, int tl, int tr, int val){
        if(tl >= r || tr <= l){
            return;
```

```
        }
        if(tl >= l && tr <= r){
            t[v].l += val;
            t[v].r += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return gcd(t[v].l, t[v].g);
        }

        push(v);
        int tm = (tl + tr) / 2;
        return gcd(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
    }
};*/

/*int C(int x, int k){
    if(x < 0 || k < 0 || k > x){
        return 0;
    }

    return (((f[x] * fr[k]) % mod) * fr[x - k]) % mod;
}*/

struct seg_tree {
    vector <int> t;
    //vector <int> p;

    /*void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }*/

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = h[tl];
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    void update(int v, int pos, int tl, int tr, int val) {
        if (tl > pos || tr <= pos) {
            return;
        }
        if (tl == pos && tl + 1 == tr) {
            t[v] = val;
            //p[v] += val;
            return;
        }
```

```
        //push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v];
        }

        //push(v);
        int tm = (tl + tr) / 2;
        //return min(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
        return quer(v * 2, l, r, tl, tm) + quer(v * 2 + 1, l, r, tm, tr);
    }
};

void solve() {
    int n;
    cin >> n;

    vector <int> a(n), b(n), c(n, 1);
    for(int i = 0; i < n; ++i){
        cin >> a[i];
        b[i] = a[i];
    }

    auto hod = [&](int x, int y){
        if(y == 0){
            c[x] = 0;
            b[x] = a[x];
        }else{
            b[x] -= y;
        }

        cout << x + 1 << " " << y << endl;

        cin >> x >> y;
        --x;

        if(y == -1){
            exit(0);
        }

        if(y == 0){
            b[x] = a[x];
            c[x] = 0;
        }else{
            b[x] -= y;
        }
    };

    if(n == 2){
        if(a[0] == a[1]){
            cout << "-1 -1" << endl;
            return;
        }

        if(a[0] > a[1]){
            hod(0, a[0] - a[1] - 1);
            if(b[0] == b[1]){
                hod(1, 0);
            }else{
                if(b[0] < b[1]){
                    hod(1, b[1] - b[0]);
                }
            }
```

```
        }else{
            hod(1, a[1] - a[0] - 1);
            if(b[0] == b[1]){
                hod(0, 0);
            }else{
                if(b[0] > b[1]){
                    hod(0, b[0] - b[1]);
                }
            }
        }
    }

    while(true){
        int x = 0, sm = 0;
        for(int i = 0; i < n; ++i){
            x ^= b[i];
            sm += b[i] + c[i];
        }

        if(sm == 0){
            cout << -1 << " " << -1 << endl;
            return;
        }

        if(x == 0){
            for(int i = 0; i < n; ++i){
                if(b[i] == a[i] && c[i]){
                    hod(i, 0);
                    break;
                }
            }
            for(int i = 0; i < n; ++i){
                if(c[i]){
                    hod(i, 0);
                    break;
                }
            }
            for(int i = 0; i < n; ++i){
                if(b[i] > 0){
                    hod(i, 1);
                    break;
                }
            }
        }else{
            for(int i = 0; i < n; ++i){
                if((x ^ b[i]) < b[i]){
                    hod(i, b[i] - (x ^ b[i]));
                    break;
                }
            }
        }
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cout.precision(20);
#ifndef DEBUG
    //freopen("schools.in", "r", stdin);
    //freopen("schools.out", "w", stdout);
#endif

    //freopen("output.txt", "w", stdout);

    //solve();

    ll tests = 1;
    //cin >> tests;
    while (tests--) {
        solve();
    }
}
```

Task E ()

```
//#include <bits/stdc++.h>
#include <deque>
#include <iostream>
#include <random>
#include <vector>
#include <algorithm>
#include <bitset>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <map>
#include <cmath>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <fstream>
#include <random>
#include <queue>
#include <complex>
#include <ctime>

/*#ifdef DEBUG
    #define _GLIBCCX_DEBUG
#endif*/

//#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
/*#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")*/

using namespace std;
using namespace __gnu_pbds;

//#define int unsigned long long
//#define int long long
//#define int __int128
//#define int unsigned int
```

```
//typedef long long ll;
//typedef long double ld;
#define ll long long
#define double long double
typedef complex<double> cld;

typedef tree<
        pair<int, int>,
        null_type,
        less< pair <int, int> >,
        rb_tree_tag,
        tree_order_statistics_node_update
> ordered_set;

template<typename T>
ostream &operator<<(ostream &ws, const vector<T> &v) {
    for (auto e: v) {
        ws << e << "␣";
    }
    return ws;
}

//const int mod = 998244353;
const ll mod = 1000000007;
//const int mod = 1000000009;
//const int mod = 11019929;
mt19937 rnd;
//(int32_t)mt19937(0)()

/*int gcd(int a, int b)
{
    while(b > 0){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
    //if (b == 0) return a;
    //return gcd(b, a % b);
}*/

/*#define abs abs1

int abs1(int x){
    return x < 0 ? -x : x;
}*/

struct seg_tr {
    vector<int> t, p;

    void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void update(int v, int l, int r, int tl, int tr, int val) {
        if (tl >= r || tr <= l) {
            return;
        }
        if (tl >= l && tr <= r) {
            t[v] += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
```

```
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return -1;
        }
        if (tl >= l && tr <= r && t[v] >= 0) {
            return -1;
        }
        if (tl + 1 == tr) {
            return tl;
        }

        push(v);
        int tm = (tl + tr) / 2;
        int get = quer(v * 2 + 1, l, r, tm, tr);
        if (get == -1) {
            return quer(v * 2, l, r, tl, tm);
        }
        return get;
    }

    /*int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return -1;
        }
        if(tl >= l && tr <= r && t[v] > 0){
            return -1;
        }

        if(tl + 1 == tr){
            return tl;
        }

        int tm = (tl + tr) / 2;
        int get = quer(v * 2, l, r, tl, tm);
        if(get == -1){
            return quer(v * 2 + 1, l, r, tm, tr);
        }
        return get;
    }*/
};

/*struct merge_tree{
    vector < ordered_set > t;

    void update(int v, int pos, int tl, int tr, int val){
        if(tl > pos || tr <= pos){
            return;
        }

        if(tl + 1 == tr && tl == pos){
            t[v].insert({val, k});
            ++k;
            return;
        }

        t[v].insert({val, k});
        ++k;

        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }

        if(tl >= l && tr <= r){
            return t[v].size() - t[v].order_of_key({x, 1e9});
        }
```

```
        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};*/

/*struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int>& h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x, int y){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return upper_bound(t[v].begin(), t[v].end(), y) - lower_bound(t[v].begin(), t[v].end()
                , x);
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x, y) + quer(v * 2 + 1, l, r, tm, tr, x, y);
    }
};*/

struct merge_tree{
    vector < vector <int> > t;

    void build(int v, int tl, int tr, vector <int> &h){
        if(tl + 1 == tr){
            t[v].push_back(h[tl]);
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v].resize(t[v * 2].size() + t[v * 2 + 1].size());
        merge(t[v * 2].begin(), t[v * 2].end(), t[v * 2 + 1].begin(), t[v * 2 + 1].end(), t[v].
            begin());
    }

    int quer(int v, int l, int r, int tl, int tr, int x){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return lower_bound(t[v].begin(), t[v].end(), x) - t[v].begin();
        }

        int tm = (tl + tr) / 2;
        return quer(v * 2, l, r, tl, tm, x) + quer(v * 2 + 1, l, r, tm, tr, x);
    }
};

/*int n;

struct fenvik{
    vector <int> t;

    int quer(int pos){
        if(pos == -1){
```

```
            return 0;
        }

        int sm = 0;
        for(int i = pos; i >= 0; i = (i & (i + 1)) - 1){
            sm += t[i];
        }
        return sm;
    }

    void update(int pos, int val){
        for(int i = pos; i < n; i = (i | (i + 1))){
            t[i] += val;
        }
    }
};*/

struct val {
    int l, r, val;
};

const double pi = 3.14159265358979323846;

int binpow(int x, int k) {
    if (k == 0) {
        return 1;
    }
    int y = binpow(x, k / 2);
    if (k % 2 == 0) {
        return (y * y) % mod;
    } else {
        return (((y * y) % mod) * x) % mod;
    }
}

/*struct pers_seg_tree {
    vector<val> t;
    int id = 0;

    void build(int v, int tl, int tr) {
        id = max(id, v);
        if (tl + 1 == tr) {
            t[v] = {-1, -1, 0};
            return;
        }

        t[v] = {v * 2, v * 2 + 1, 0};
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm);
        build(v * 2 + 1, tm, tr);
    }

    int update(int v, int pos, int tl, int tr, int val) {
        if (tl == pos && tl + 1 == tr) {
            //t.push_back({-1, -1, t[v].val + val});
            //return t.size() - 1;
            t[id] = {-1, -1, t[v].val + val};
            id++;
            return id - 1;
        }

        int tm = (tl + tr) / 2;
        if (pos < tm) {
            int res = update(t[v].l, pos, tl, tm, val);
            //t.push_back({res, t[v].r, t[res].val + t[t[v].r].val});
            //return t.size() - 1;
            t[id] = {res, t[v].r, t[res].val + t[t[v].r].val};
            id++;
            return id - 1;
        } else {
            int res = update(t[v].r, pos, tm, tr, val);
            //t.push_back({t[v].l, res, t[t[v].l].val + t[res].val});
            //return t.size() - 1;
            t[id] = {t[v].l, res, t[t[v].l].val + t[res].val};
```

```
            id++;
            return id - 1;
        }
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v].val;
        }

        int tm = (tl + tr) / 2;
        return quer(t[v].l, l, r, tl, tm) + quer(t[v].r, l, r, tm, tr);
    }

    int find_k(int vl, int vr, int tl, int tr, int k) {
        if (tl + 1 == tr) {
            return tl;
        }

        int tm = (tl + tr) / 2;

        int kol = t[t[vr].l].val - t[t[vl].l].val;

        if (kol >= k) {
            return find_k(t[vl].l, t[vr].l, tl, tm, k);
        } else {
            return find_k(t[vl].r, t[vr].r, tm, tr, k - kol);
        }
    }
};*/

struct elem {
    int l, r, g;
};

double rand_double() {
    return (double) rand() / RAND_MAX;
}

/*struct gcd_seg_tree{
    vector <elem> t;
    vector <int> p;

    void push(int v){
        if(p[v] != 0){
            t[v * 2].l += p[v];
            t[v * 2 + 1].l += p[v];
            t[v * 2].r += p[v];
            t[v * 2 + 1].r += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = {h[tl], h[tl], 0};
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    void update(int v, int l, int r, int tl, int tr, int val){
        if(tl >= r || tr <= l){
            return;
```

```
        }
        if(tl >= l && tr <= r){
            t[v].l += val;
            t[v].r += val;
            p[v] += val;
            return;
        }

        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, l, r, tl, tm, val);
        update(v * 2 + 1, l, r, tm, tr, val);
        t[v] = {t[v * 2].l, t[v * 2 + 1].r, gcd(t[v * 2].r - t[v * 2 + 1].l, gcd(t[v * 2].g, t[v *
             2 + 1].g))};
    }

    int quer(int v, int l, int r, int tl, int tr){
        if(tl >= r || tr <= l){
            return 0;
        }
        if(tl >= l && tr <= r){
            return gcd(t[v].l, t[v].g);
        }

        push(v);
        int tm = (tl + tr) / 2;
        return gcd(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
    }
};*/

/*int C(int x, int k){
    if(x < 0 || k < 0 || k > x){
        return 0;
    }

    return (((f[x] * fr[k]) % mod) * fr[x - k]) % mod;
}*/

struct seg_tree {
    vector <int> t;
    //vector <int> p;

    /*void push(int v) {
        if (p[v] != 0) {
            t[v * 2] += p[v];
            t[v * 2 + 1] += p[v];
            p[v * 2] += p[v];
            p[v * 2 + 1] += p[v];
            p[v] = 0;
        }
    }*/

    void build(int v, int tl, int tr, vector <int> & h){
        if(tl + 1 == tr){
            t[v] = h[tl];
            return;
        }

        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, h);
        build(v * 2 + 1, tm, tr, h);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    void update(int v, int pos, int tl, int tr, int val) {
        if (tl > pos || tr <= pos) {
            return;
        }
        if (tl == pos && tl + 1 == tr) {
            t[v] = val;
            //p[v] += val;
            return;
        }
```

```
        //push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, pos, tl, tm, val);
        update(v * 2 + 1, pos, tm, tr, val);
        //t[v] = min(t[v * 2], t[v * 2 + 1]);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }

    int quer(int v, int l, int r, int tl, int tr) {
        if (tl >= r || tr <= l) {
            return 0;
        }
        if (tl >= l && tr <= r) {
            return t[v];
        }

        //push(v);
        int tm = (tl + tr) / 2;
        //return min(quer(v * 2, l, r, tl, tm), quer(v * 2 + 1, l, r, tm, tr));
        return quer(v * 2, l, r, tl, tm) + quer(v * 2 + 1, l, r, tm, tr);
    }
};

map <int, vector <int> > m1;
map < vector <int>, int > m2;
int nxt = 1;

void solve1() {
    int n = 10;
    vector < vector <int> > a(n, vector <int> (n));

    int k;
    cin >> k;

    vector <int> b = m1[k];

    for(int i = 0; i < n; ++i){
        int kol = b[i];
        for(int j = 0; j < n; ++j){
            if(kol > 0){
                a[i][j] = 1;
                kol--;
            }
        }
    }

    for(int i = 0; i < n; ++i){
        for(int j = 0; j < n; ++j){
            cout << a[i][j];
        }
        cout << '\n';
    }
}

void solve2(){
    int n = 10;
    vector < vector <int> > a(n, vector <int> (n));

    vector <int> b;
    for(int i = 0; i < n; ++i){
        for(int j = 0; j < n; ++j){
            char x;
            cin >> x;
            a[i][j] = x - '0';
        }
    }

    int k = 0;

    for(int i = 0; i < n; ++i){
        int kol = 0;
        for(int j = 0; j < n; ++j){
            kol += a[i][j];
```

```
        }
        b.push_back(kol);
    }

    sort(b.begin(), b.end());

    cout << m2[b] << '\n';
}

void gen(vector <int> & a){
    if(!m2.count(a)) {
        m1[nxt] = a;
        m2[a] = nxt;
        nxt++;
    }else{
        return;
    }

    for(int i = 0; i + 1 < 10; ++i){
        if(a[i] + 1 <= a[i + 1]){
            a[i]++;
            gen(a);
            a[i]--;
        }
    }
    if(a[9] < 10){
        a[9]++;
        gen(a);
        a[9]--;
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cout.precision(20);
#ifndef DEBUG
    //freopen("schools.in", "r", stdin);
    //freopen("schools.out", "w", stdout);
#endif

    //freopen("output.txt", "w", stdout);

    //solve();

    vector <int> _(10, 0);
    gen(_);

    //cout << nxt << endl;

    ll tests = 1;
    cin >> tests;
    string s;
    cin >> s;
    if(s == "transmit") {
        while (tests--) {
            solve1();
        }
    }else{
        while (tests--) {
            solve2();
        }
    }
}
```

Task F ()

```
from itertools import permutations
from math import gcd

a = int(input())
b = int(input())
print(a + b)
```