

Team Notebook

University of Toronto

February 15, 2020

Contents

1 Algorithms	2	2 DP	6	5.4 numbertheory	12
1.1 AP-Bridges	2	2.1 ConvexHullTrick	6	5.5 simplex	13
1.2 Hungarian	2	3 Geo	6	5.6 tricks	14
1.3 SCC-Tarjans	2	3.1 ConvexHull	6	6 Misc	14
1.4 TernarySearch	2	3.2 template	6	6.1 OrderedSet	14
1.5 UnionFind	3	4 Graph	10	6.2 template	15
1.6 binarylifting-lca	3	4.1 Flow	10	7 Python	15
1.7 centroiddecomposition	3	4.1.1 Dinic	10	7.1 InputArray	15
1.8 dijkstras	4	4.1.2 MinCostMaxFlow	10	8 Strings	15
1.9 findcycles	4	5 Math	11	8.1 kmp	15
1.10 matrixexponentiation	4	5.1 Miller-Rabin	11	8.2 suffix-array	15
1.11 segtreeInteractive	5	5.2 fft	11	8.3 z	16
1.12 segtreeRMQTemplate	5	5.3 math	12		
1.13 segtreeRSQTemplate	5				

1 Algorithms

1.1 AP-Bridges

```
int dfs(int u,int p){
    dfs_num[u] = dfs_low[u] = ++dfs_counter;
    for(auto v : adjList[u]){
        if(dfs_num[v]==0){
            dfs(v,u);
            if(dfs_low[v] >= dfs_num[u]){
                articulation[u]=true;
            }
            if(dfs_low[v] > dfs_num[u])
                bridge = true;
            dfs_low[u] = min(dfs_low[u],dfs_low[v]);
        } else if(v!=p)
            dfs_low[u] = min(dfs_low[u],dfs_num[v]);
    }
}

int main(){
    memset(dfs_num,0,sizeof(dfs_num));
    memset(dfs_low,0,sizeof(dfs_low));
    bridge=false;
    dfs_counter=0;
    dfs(0,-1);
    for(int i = 0; i < N; ++i)
        if(dfs_num[i]==0)
            bridge=true;
    puts(bridge ? "Yes" : "No");
    return 0;
}
```

1.2 Hungarian

```
/*
    This will take a matrix a[N][N] and choose one item for
    each row such that the sum of all items
    is minimized.
    O(n^3)
*/
#define N 107

ll INF = 1000000000000000000ll;
int n;

ll a[N][N];
```

```
void hungarian(){
    vector<ll> u(n+1), v(n+1), p(n+1), way(n+1);
    for(int i=1; i<=n; i++){
        p[0] = i;
        ll j0 = 0;
        vector<ll> minv(n+1, INF);
        vector<char> used(n+1, false);
        do {
            used[j0] = true;
            ll i0 = p[j0], delta = INF, j1;
            for(int j=1; j<=n; j++){
                if(!used[j]) {
                    ll cur = a[i0][j]-u[i0]-v[j];
                    if(cur < minv[j])
                        minv[j] = cur, way[j] = j0;
                    if(minv[j] < delta)
                        delta = minv[j], j1 = j;
                }
            }
            for(int j=0; j<=n; j++){
                if(used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            }
            j0 = j1;
        } while(p[j0] != 0);
        do {
            ll j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while(j0);
    }
}
```

```
vector<int> ans(n+1);
for(int j=1; j<=n; j++)
    ans[p[j]] = j;
```

1.3 SCC-Tarjans

```
typedef pair<int, int> ii;

int N,M;
vector<int> adjList[MX_N];
int dfs_num[MX_N],dfs_low[MX_N];
bool vis[MX_N];
stack<int> scc;
```

```
int dfsCounter=1;
int sccIdx=1;

map<int, int> sccMap;

void tarjans(int u){
    scc.push(u);
    vis[u]=true;

    dfs_low[u]=dfs_num[u]=dfsCounter++;

    for(int i = 0; i < adjList[u].size(); i++){
        int v = adjList[u][i];
        if(dfs_num[v]==0){
            tarjans(v);
            dfs_low[u]=min(dfs_low[u],dfs_low[v]);
        } else if(vis[v]){
            dfs_low[u]=min(dfs_low[u],dfs_num[v]);
        }
    }
    if(dfs_low[u]==dfs_num[u]){
        while(1){
            int v = scc.top(); scc.pop();
            sccMap[v]=sccIdx;
            vis[v]=false;
            if(v==u)
                break;
        }
        sccIdx++;
    }
}
```

1.4 TernarySearch

```
double ternary_search(double l, double r) {
    double eps = 1e-9; //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1); //evaluates the function at m1
        double f2 = f(m2); //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l); //return the maximum of f(x)
    in [l, r]
}
```

1.5 UnionFind

```
/**
 * Union find algorithm
 * Complexity  $O(\log n)$  for Join or Find.
 */
```

```
int pai[N];

void init(int n){
    for(int i=1; i<=n; i++){
        pai[i]=i;
    }
}

int find(int i){
    if(pai[i]==i) return i;
    return pai[i]=find(pai[i]);
}

int join(int a, int b){
    a=find(a);
    b=find(b);
    pai[a]=pai[b];
}
```

1.6 binarylifting-lca

```
#include <bits/stdc++.h>
#define MAXN 100100
typedef long long ll;

using namespace std;

int n, m, s[MAXN], depth[MAXN], anc[MAXN][40];
vector<int> g[MAXN];
bool vis[MAXN];

int dfs(int x, int d, int p){
    vis[x] = true;
    depth[x] = d;
    s[x] = 1;
    anc[x][0] = p;
    for(int i = 1; pow(2, i) <= d; i++){
        anc[x][i] = anc[anc[x][i-1]][i-1];
    }
}
```

```
    }
    for(int i = 0; i < g[x].size(); i++){
        if(vis[g[x][i]]) continue;
        s[x] += dfs(g[x][i], d + 1, x);
    }

    return s[x];
}

int walk(int x, int d){
    int i = 0;
    while(d){
        if(d & 1) x = anc[x][i];
        d /= 2;
        i++;
    }
    //cout << "\n";
    return x;
}

int lca(int x, int y){
    //cout << x<<y;

    if(depth[x] < depth[y]) y = walk(y, depth[y] - depth[x]);
    if(depth[x] > depth[y]) x = walk(x, depth[x] - depth[y]);
    //cout << x<<y;
    if(x == y) return x;
    for(int i = 30; i >= 0; i--){
        if(depth[x] >= pow(2, i) && anc[x][i] != anc[y][i]){
            return lca(anc[x][i], anc[y][i]);
        }
    }
    return anc[x][0];
}

int main(){
    ios_base::sync_with_stdio(false);
    cin >> n;
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1, 0, -1);
    cin >> m;
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        if(depth[a] > depth[b]) swap(a, b);
    }
}
```

```
    if(a == b) cout << n;
    else{
        int l = lca(a, b);
        int d = -2 * depth[l] + depth[a] + depth[b];
        if(d % 2) cout << "0";
        else{
            if(depth[a] == depth[b]) cout << s[l] - s[walk(b, d / 2 - 1)] - s[walk(a, d / 2 - 1)];
            else cout << s[walk(b, d / 2)] - s[walk(b, d / 2 - 1)];
        }
    }
    cout << "\n";
}
```

1.7 centroiddecomposition

```
#include <bits/stdc++.h>
#define MAXN 100100
typedef long long ll;

using namespace std;

int n, sz[MAXN];
bool deleted[MAXN], vis[MAXN];
char ch[MAXN];
vector<int> g[MAXN];

void dfs(int x, int p){
    if(vis[x]) return;
    vis[x] = true;
    sz[x] = 1;
    for(auto i : g[x]){
        if(i == p || deleted[i]) continue;
        dfs(i, x);
        sz[x] += sz[i];
    }
    //cout << x << " " << sz[x] << "\n";
}

int findCentroid(int x){
    memset(vis, 0, sizeof(vis));
    dfs(x, -1);
    int p = -1, c = sz[x] / 2;
    while(true){
        bool found = false;
        for(auto i : g[x]){
            if(!deleted[i] && abs(sz[i] - c) < c){
                p = i;
                found = true;
                break;
            }
        }
        if(!found) break;
        deleted[x] = true;
        x = p;
    }
}
```

```

    if(!deleted[i] && i != p && sz[i] > c){
        found = true;
        p = x;
        x = i;
        break;
    }
}
if(!found) return x;
}
}

void decomp(int x, char c){
    int cen = findCentroid(x);
    ch[cen] = c;
    deleted[cen] = true;
    for(auto i : g[cen]){
        if(deleted[i]) continue;
        decomp(i, c + 1);
    }
}

int main(){
    #ifndef ONLINE_JUDGE
        freopen("input.txt", "r", stdin);
    #endif
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    memset(deleted, 0, sizeof(deleted));
    cin >> n;
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    //cout << findCentroid(1);
    decomp(1, 'A');
    for(int i = 1; i <= n; i++){
        cout << ch[i] << " ";
    }
}

```

1.8 dijkstras

```

#include <bits/stdc++.h>
#include <utility>
#define MAXN 505

using namespace std;

```

```

typedef long long ll;
typedef pair<int, int> ii;
int n;

vector<pair<int, int> > g[MAXN];
int dist[MAXN];

void dijkstra(int x){
    for(int i = 0; i < n; i++){
        dist[i] = 999999999;
    }
    priority_queue<pair<int, int>, vector<pair<int, int> >,
        greater<pair<int, int> > > pq;
    pq.push({0, x});
    dist[x] = 0;
    while(!pq.empty()){
        pair<int, int> v = pq.top();
        pq.pop();
        for(int i = 0; i < g[v.second].size(); i++){
            pair<int, int> u = g[v.second][i];
            if(dist[v.second] + u.second < dist[u.first])
                pq.push({dist[u.first] = dist[v.second] + u.second, u.first});
        }
    }
}

int main(){
    #ifndef ONLINE_JUDGE
        freopen("input.txt", "r", stdin);
    #endif
    ios_base::sync_with_stdio(false);
    //cin >> n;
}

```

1.9 findcycles

```

int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end; // In O(M)

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;

```

```

            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
}

```

1.10 matrixexponentiation

```

#include <bits/stdc++.h>
#define MAXN 100100
#define DIM 2
#define pii pair<int, int>
#define pb push_back
typedef long long ll;

```

```

using namespace std;

ll mod = 1e9 + 7;
struct matrix{
    ll a[DIM][DIM];
    matrix(){
        memset(a, 0, sizeof(ll) * DIM * DIM);
    }

    void init(){
        a[0][0] = 0; a[0][1] = 1;
        a[1][0] = 1; a[1][1] = 1;
    }

    matrix operator*(matrix b){
        matrix c;
        for(int k = 0; k < DIM; k++) {
            for(int i = 0; i < DIM; i++) {
                for(int j = 0; j < DIM; j++) {
                    c.a[i][j] = (c.a[i][j] + a[i][k] * b.a[k][j]) % mod;
                }
            }
        }
        return c;
    }

    vector<ll> times(vector<ll> v){
        vector<ll> c(DIM, 0);
        for(int i = 0; i < DIM; i++){
            for(int j = 0; j < DIM; j++){
                c[i] += v[j] * a[i][j];
                c[i] %= mod;
            }
        }
        return c;
    }

};

matrix pow_matrix(matrix a, ll n) {
    if (n == 1) return a;
    matrix b = pow_matrix(a, n / 2);
    b = b * b;
    if (n & 1) b = b * a;
    return b;
}

int n;

int main(){

```

```

    #ifndef ONLINE_JUDGE
        freopen("input.txt", "r", stdin);
    #endif
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    n = 10;
    vector<ll> base = {1, 1}, v;
    matrix m;
    m.init();
    m = pow_matrix(m, 4);
    v = m.times(base);
    cout << v[0];
}

```

1.11 segtreeInteractive

```

int t[2*N], n; // When debugging, the prob is most likely
               // that you have multiple n's. need this one here!

int query(int l, int r){ // This r is exclusive!
    int ans=0;
    for(l+=n, r+=n; l<r; l>>=1, r>>=1){
        if(l&1)ans+=t[l++];
        if(r&1)ans+=t[--r];
    }
    return ans;
}

void update(int p, int v){
    for(t[p+=n]+=v; p>1; p>>=1){
        t[p>>1]=t[p]+t[p^1];
    }
}

```

1.12 segtreeRMQTemplate

```

#include <bits/stdc++.h>
#define MAXN 505
typedef long long ll;

using namespace std;

int n; // Range query, range update w/ lazy

int lo[4 * MAXN + 1], hi[4 * MAXN + 1], tree[4 * MAXN + 1],
    delta[4 * MAXN + 1];

```

```

void init(int i, int l, int r){
    lo[i] = l;
    hi[i] = r;
    if(l == r) return;
    int m = (l + r) / 2;
    init(2 * i, l, m);
    init(2 * i + 1, m + 1, r);
}

void prop(int i){
    delta[2 * i] += delta[i];
    delta[2 * i + 1] += delta[i];
    delta[i] = 0;
}

void update(int i){
    tree[i] = min(tree[2 * i] + delta[2 * i], tree[2 * i + 1] +
        delta[2 * i + 1]);
}

void inc(int i, int l, int r, int val){
    if(r < lo[i] || hi[i] < l) return;
    if(hi[i] <= r && lo[i] >= l){
        delta[i] += val;
        return;
    }
    //partial cover case
    prop(i);
    inc(2 * i, l, r, val);
    inc(2 * i + 1, l, r, val);
    update(i);
}

int query(int i, int l, int r){
    if(r < lo[i] || hi[i] < l) return INT_MAX; //not in range
    if(hi[i] <= r && lo[i] >= l) return tree[i] + delta[i]; //
        completely in range
    prop(i);
    int minLeft = query(2 * i, l, r);
    int minRight = query(2 * i + 1, l, r);
    update(i);
    return min(minLeft, minRight);
}

int main(){
    init(1, 0, n - 1);
}

```

1.13 segtreeRSQTemplate

```
#include <bits/stdc++.h>
#define MAXN 505
typedef long long ll;

using namespace std;

int n;

// Range query, range update w/ lazy

ll lo[4 * MAXN + 1], hi[4 * MAXN + 1], tree[4 * MAXN + 1],
    delta[4 * MAXN + 1];

void init(int i, int l, int r){
    lo[i] = l;
    hi[i] = r;
    if(l == r) return;
    int m = (l + r) / 2;
    init(2 * i, l, m);
    init(2 * i + 1, m + 1, r);
}

void prop(int i){
    delta[2 * i] += delta[i];
    delta[2 * i + 1] += delta[i];
    delta[i] = 0;
}

void update(int i){
    tree[i] = tree[2 * i] + delta[2 * i] * (hi[2 * i] - lo[2 * i] + 1)
    + tree[2 * i + 1] + delta[2 * i + 1] * (hi[2 * i + 1] - lo[2 * i + 1] + 1);
}

void inc(int i, int l, int r, ll val){
    if(r < lo[i] || hi[i] < l) return;
    if(hi[i] <= r && lo[i] >= l){
        delta[i] += val;
        return;
    }
    //partial cover case
    prop(i);
    inc(2 * i, l, r, val);
    inc(2 * i + 1, l, r, val);
    update(i);
}

void simpleinc(int i, ll val){
```

```
i += n;
while(i > 0){
    tree[i] += val;
    i /= 2;
}
}

ll query(int i, int l, int r){
    if(r < lo[i] || hi[i] < l) return 0;
    if(hi[i] <= r && lo[i] >= l) return tree[i] + delta[i] * (
        hi[i] - lo[i] + 1);
    prop(i);
    ll sumLeft = query(2 * i, l, r);
    ll sumRight = query(2 * i + 1, l, r);
    update(i);
    return sumLeft + sumRight;
}

int main(){
    init(1, 0, n - 1);
}
```

2 DP

2.1 ConvexHullTrick

```
/**
 * Source: Simon Lindholm
 * Description: Container where you can add lines of the
 *              form  $kx+m$ , and query maximum values at points  $x$ .
 * Useful for dynamic programming.
 * Requires C++ 14
 * Use when  $dp[i] = \max(m(j) * i + b(j))$  where  $m$  and  $b$  are
 *              determined by some  $j < i$ 
 * Negate everything to get min
 * Time:  $O(\log N)$ 
 */

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use  $\text{inf} = 1/.0$ ,  $\text{div}(a,b) = a/b$ )
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
};
```

```
bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
}

void add(ll k, ll m) { //slope k, intercept m
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
        isect(x, erase(y));
}

ll query(ll x) { //max value at point x
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
}
};
```

3 Geo

3.1 ConvexHull

```
bool cmp(pt a, pt b){return mk(a.y, a.x) < mk(b.y, b.x);}

vector<pt> convexhull(vector<pt> p){ //counterclockwise, no
    collinear points
    sort(p.begin(), p.end(), cmp);
    p.erase(unique(p.begin(), p.end(), p.end()));
    vector<pt> up, dn;
    for(pt i : p){
        while(up.size() > 1 and orient(up[up.size() - 2], up.back(), i) >= 0) up.pop_back();
        while(dn.size() > 1 and orient(dn[dn.size() - 2], dn.back(), i) <= 0) dn.pop_back();
        up.pb(i);
        dn.pb(i);
    }
    for(int i = (int) up.size() - 2; i >= 1; i--)
        dn.pb(up[i]);
    return dn;
}
```

3.2 template

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

using namespace std;

#define ll long long int
#define pb push_back
#define mk make_pair
#define mt make_tuple
#define fi first
#define se second
#define ii pair<int, int>
#define all(x) (x).begin(), (x).end()
#define N 1000007 // 10e6 + 7

struct stableSum {
    /*
     * Use stableSum to add (positive) elements that are doubles.
     * It greatly reduces imprecision.
     */
    int cnt = 0;
    vector<double> v, pref{0};
    void operator+=(double a) {
        assert(a >= 0);
        int s = ++cnt;
        while (s % 2 == 0) {
            a += v.back();
            v.pop_back(), pref.pop_back();
            s /= 2;
        }
        v.push_back(a);
        pref.push_back(pref.back() + a);
    }
    double val() {return pref.back();}
};

int quadRoots(double a, double b, double c, pair<double,
double> &out) {
    /*
     * quadRoots will give the quadratic answer to equation
     *  $x^2*a + x*b + c$  for  $a \neq 0$ 
     * Returns how many solutions, place them in out.
     */
    assert(a != 0);
    double disc = b*b - 4*a*c;
    if (disc < 0) return 0;
    double sum = (b >= 0) ? -b-sqrt(disc) : -b+sqrt(disc);
    out = {sum/(2*a), sum == 0 ? 0 : (2*c)/sum};
}
```

```
return 1 + (disc > 0);
}

/*
 * Tips:
 * - Use Integers whenever possible.
 * - Minimize division and square root operations.
 * - Try to write code that handles many situations at once.
 */

/*
 * ----- Points
 * -----
 */

typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()

// abs(p) = sqrt(x*x + y*y)

T sq(pt p) {return p.x*p.x + p.y*p.y;}

pt translate(pt v, pt p) {
    // Translate a point p by a vector v.
    return p+v;
}

pt scale(pt c, double factor, pt p) {
    // Scale point p by factor around a center c.
    return c + (p-c)*factor;
}

pt rot(pt p, double a) {
    // Rotate point p by an angle a, counterclockwise.
    return p * polar(1.0, a);
}

pt perp(pt p) {
    // Rotate point p by 90 degrees, good for integer coords.
    return {-p.y, p.x};
}

T dot(pt v, pt w) {
    /*
     *  $v \cdot w = |v| \cdot |w| \cdot \cos(\text{angle})$ 
     * Check sign of dot product to see if two vectors are going
     * in the same dir.
     * Positive if angle <  $\pi/2$ , neg if >, 0 if =
     */
}
```

```
*/
return v.x*w.x + v.y*w.y;
}

bool isPerp(pt v, pt w) {return dot(v,w) == 0;}

double angle(pt v, pt w) {
    // Angle between two vectors.
    return acos(clamp(dot(v,w) / abs(v) / abs(w), -1.0, 1.0));
}

T cross(pt v, pt w) {
    /*
     *  $v \times w = |v| \cdot |w| \cdot \sin(\text{angle})$ 
     * Order of v, w matters! Angle is the ORIENTED angle between
     * v and w.
     * Positive if  $0 < \text{angle} < \pi$ , neg if  $-\pi < \text{angle} < 0$ , zero
     * if angle = 0 or  $\pi$ .
     */
    return v.x*w.y - v.y*w.x;
}

T orient(pt a, pt b, pt c) {
    // I'll go from a to b to c. If turn left to c, positive.
    // Right negative, straight zero.
    return cross(b-a, c-a);
}

double orientedAngle(pt a, pt b, pt c) {
    // Return the oriented angle between ab and ac, going from
    // b to c.
    if (orient(a,b,c) >= 0)
        return angle(b-a, c-a);
    return 2*M_PI - angle(b-a, c-a);
}

bool inAngle(pt a, pt b, pt c, pt p) {
    // Use this to check if p lies in the angle that ab and ac
    // form.
    assert(orient(a,b,c) != 0);
    if (orient(a,b,c) < 0) swap(b,c);
    return orient(a,b,p) >= 0 && orient(a,c,p) <= 0;
}

bool isConvex(vector<pt> p) {
    // To check if a polygon is convex, the orientation of all
    // three consecutive
    // points should be the same.
    bool hasPos=false, hasNeg=false;
    for (int i=0, n=p.size(); i<n; i++) {

```

```

    int o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
    if (o > 0) hasPos = true;
    if (o < 0) hasNeg = true;
}
return !(hasPos && hasNeg);
}

bool half(pt p, pt v = {-1.0, 0.0}) { // true if in blue half
    // Modify v if you want a different starting angle.
    assert(p.x != 0 || p.y != 0); // the argument of (0,0) is undefined
    return cross(v,p) < 0 || (cross(v,p) == 0 && dot(v,p) < 0);
}

void polarSort(vector<pt> &v) {
    /*
    This will sort points according to their angle based on
    the origin.
    If I want to do the same thing but with a point not the
    origin, I have
    to subtract that point from all other points.
    If I want to add parameters in the sort such as magnitude,
    just add terms
    to the tuple.
    */
    sort(v.begin(), v.end(), [](pt v, pt w) {
        return make_tuple(half(v), 0) < make_tuple(half(w), cross(
            v,w));
    });
}

/*
----- Lines
-----
*/

struct line {
    pt v; T c;
    // From direction vector v and offset c
    line(pt v, T c) : v(v), c(c) {}
    // From equation ax+by=c
    line(T a, T b, T c) : v(b,-a), c(c) {}
    // From points P and Q
    line(pt p, pt q) : v(q-p), c(cross(v,p)) {}

    // - these work with T = int
    T side(pt p);
    double dist(pt p);
    double sqDist(pt p);

```

```

    double slope();
    line perpThrough(pt p);
    bool cmpProj(pt p, pt q);
    line translate(pt t);
    // - these require T = double
    line shiftLeft(double dist);
    pt proj(pt p);
    pt refl(pt p);
};

T line::side(pt p) {
    // This says what side of the line a point is.
    // Positive side is on the left (remember the line has
    orientation).
    return cross(v,p)-c;
}

double line::dist(pt p) {
    // Dist point -> line
    return abs(side(p)) / abs(v);
}

double line::sqDist(pt p) {
    // Dist point -> line squared.
    return side(p)*side(p) / (double)sq(v);
}

double line::slope(){
    return v.y/v.x;
}

line line::perpThrough(pt p) {
    // Line that is perpendicular to this line, and goes
    through p.
    return {p, p + perp(v)};
}

bool line::cmpProj(pt p, pt q) {
    // Use this if you want to sort points through a line.
    return dot(v,p) < dot(v,q);
}

line line::translate(pt t) {
    // Translate this line by vector t.
    return {v, c + cross(v,t)};
}

line line::shiftLeft(double dist) {
    // Shift this line to the left by dist. Note: you gotta
    substitute.

```

```

    return {v, c + dist*abs(v)};
}

bool inter(line l1, line l2, pt &out) {
    // Check if l1 and l2 intersect.
    T d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d; // requires floating-
    point coordinates
    return true;
}

pt line::proj(pt p) {
    // Projects a point into a line.
    return p - perp(v)*side(p)/sq(v);
}

pt line::refl(pt p) {
    // This is the point that is the same distance from line as
    p, but on the other side.
    return p - perp(v)*2.0*side(p)/sq(v);
}

line bisector(line l1, line l2, bool interior) {
    // This returns the line that is between l1 and l2,
    dividing the angle in 2.
    assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be
    parallel!
    double sign = interior ? 1 : -1;
    return {l2.v/abs(l2.v) + l1.v/abs(l1.v) * sign, l2.c/abs(l2
        .v) + l1.c/abs(l1.v) * sign};
}

/*
----- Segments
-----
*/

bool inDisk(pt a, pt b, pt p) {
    // Pts a, b are the diameter of a disk, want to know if
    point p is inside.
    return dot(a-p, b-p) <= 0;
}

bool onSegment(pt a, pt b, pt p) {
    // Check if point p is in the segment formed by [a, b].
    return orient(a,b,p) == 0 && inDisk(a,b,p);
}

bool properInter(pt a, pt b, pt c, pt d, pt &out) {
    // Check if two segments [a, b], [c, d] intersect.

```



```

// The proper interception is an interception that is a
// single point, but not an endpoint.
double oa = orient(c,d,a),
ob = orient(c,d,b),
oc = orient(a,b,c),
od = orient(a,b,d);
// Proper intersection exists iff opposite signs
if (oa*ob < 0 && oc*od < 0) {
    out = (a*ob - b*oa) / (ob-oa);
    return true;
}
return false;
}

// To create sets of points we need a comparison function
struct cmpX {
    bool operator()(const pt &a, const pt &b) const{
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    }
};

set<pt,cmpX> inters(pt a, pt b, pt c, pt d) {
    // If |set| = 0, no interception.
    // If |set| = 1, point interception.
    // If |set| = 2, segment interception.
    pt out;
    if (properInter(a,b,c,d,out)) return {out};
    set<pt,cmpX> s;
    if (onSegment(c,d,a)) s.insert(a);
    if (onSegment(c,d,b)) s.insert(b);
    if (onSegment(a,b,c)) s.insert(c);
    if (onSegment(a,b,d)) s.insert(d);
    return s;
}

double segPoint(pt a, pt b, pt p) {
    // Dist of point p to segment [a, b]
    if (a != b) {
        line l(a,b);
        if (l.cmpProj(a,p) && l.cmpProj(p,b)) // if closest to
            projection
        return l.dist(p); // output distance to line
    }
    return min(abs(p-a), abs(p-b)); // otherwise distance to A
    or B
}

double segSeg(pt a, pt b, pt c, pt d) {
    // Dist of seg [a, b] to seg [c, d]

```

```

pt dummy;
if (properInter(a,b,c,d,dummy))
    return 0;
return min({segPoint(a,b,c), segPoint(a,b,d), segPoint(c,d,
    a), segPoint(c,d,b)});
}

/*
----- Poligons
*/

double areaTriangle(pt a, pt b, pt c) {
    return abs(cross(b-a, c-a)) / 2.0;
}

double areaPolygon(vector<pt> p) {
    double area = 0.0;
    for (int i = 0, n = p.size(); i < n; i++) {
        area += cross(p[i], p[(i+1)%n]); // wrap back to 0 if i ==
            n-1
    }
    return abs(area) / 2.0;
}

bool above(pt a, pt p) {
    // True if P at least as high as A (blue part).
    return p.y >= a.y;
}

bool crossesRay(pt a, pt p, pt q) {
    // Check if [PQ] crosses ray from A.
    return (above(a,q) - above(a,p)) * orient(a,p,q) > 0;
}

bool inPolygon(vector<pt> p, pt a, bool strict = true) {
    // Check if point a is in polygon p.
    // If strict, returns false when A is on the boundary.

    int numCrossings = 0;
    for (int i = 0, n = p.size(); i < n; i++) {
        if (onSegment(p[i], p[(i+1)%n], a))
            return !strict;
        numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
    }
    return numCrossings & 1; // inside if odd number of
        crossings
}

```

```

/*
----- Circle
*/

pt circumCenter(pt a, pt b, pt c) {
    // Gives the center of the circle that goes though a, b, c.
    b = b-a, c = c-a; // consider coordinates relative to A
    assert(cross(b,c) != 0); // no circumcircle if A,B,C
        aligned
    return a + perp(b*sq(c) - c*sq(b))/cross(b,c)/2.0;
}

template <typename T> int sgn(T k) {
    // Return -1, 0, 1 depending on sign of k.
    return (T(0) < k) - (k < T(0));
}

int circleLine(pt o, double r, line l, pair<pt,pt> &out) {
    // Circle-Line intersection (0, 1, 2).
    // If only 1 intersection, out.fi == out.se.
    double h2 = r*r - l.sqDist(o);
    if (h2 >= 0) { // the line touches the circle
        pt p = l.proj(o); // point P
        pt h = l.v*sqrt(h2)/abs(l.v); // vector parallel to l, of
            length h
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}

int circleCircle(pt o1, double r1, pt o2, double r2, pair<pt
    ,pt> &out) {
    // Circle-Circle intersection (0, 1, 2, inf).
    // Similar to circleLine.
    pt d=o2-o1; double d2=sq(d);
    if (d2 == 0) {assert(r1 != r2); return 0;} // concentric
        circles
    double pd = (d2 + r1*r1 - r2*r2)/2; // = |O1P| * d
    double h2 = r1*r1 - pd*pd/d2; // = h2
    if (h2 >= 0) {
        pt p = o1 + d*pd/d2, h = perp(d)*sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}

int tangents(pt o1, double r1, pt o2, double r2, bool inner,
    vector<pair<pt,pt>> &out) {

```

```
// There can be (0, 1, 2) tangents.
// If 2 tangents, there are two pairs (p1, p2) of points of
// that tangent on the circles.
// If 1 tangent, pairs are equal.
if (inner) r2 = -r2;
pt d = o2-o1;
double dr = r1-r2, d2 = sq(d), h2 = d2-dr*dr;
if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
for (double sign : {-1,1}) {
    pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
    out.push_back({o1 + v*r1, o2 + v*r2});
}
return 1 + (h2 > 0);
}

pt minEnclosingCircle(vector<pt>>v){
    // Given a bunch of points, what is the smallest circle
    // that contains all of them?
    // Return center.
    pt p = {0, 0};
    for(int i=0; i<v.size(); i++){
        p+=v[i];
    }
    if(v.size() == 0)return p;
    p/=v.size();
    double walk = 0.1;
    double d;
    for(int i=0; i<30000; i++){
        int k = 0;
        d = abs(p-v[0]);
        for(int j=1; j<v.size(); j++){
            if(abs(p-v[j]) > d){
                d = abs(p-v[j]);
                k = j;
            }
        }
        p += (v[k] - p)*walk;
        walk *= 0.999;
    }
    // d is the radius
    return p;
}

int main(int argc, char const *argv[]){
    pt p = {3.4, 2.1};
    cout << p << endl;
    return 0;
}
```

4 Graph

4.1 Flow

4.1.1 Dinic

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

using namespace std;

const int MAXN = 2002; //XXX
//Set to the number of nodes in the flow graph.
const int MAXE = 2100012; //XXX
//Number of edges in the flow graph.

int from[MAXN], to[MAXN], cap[MAXN], prv[MAXN], head[MAXN],
    pt[MAXN], ec;

void addEdge(int u, int v, int uv, int vu = 0){
    from[ec] = u, to[ec] = v, cap[ec] = uv, prv[ec] = head[u],
        head[u] = ec++;
    from[ec] = v, to[ec] = u, cap[ec] = vu, prv[ec] = head[v],
        head[v] = ec++;
}

int lv[MAXN], q[MAXN];
bool bfs(int source, int sink){
    memset(lv, 63, sizeof(lv));
    int h = 0, t = 0;
    lv[source] = 0;
    q[t++] = source;
    while (t-h){
        int v = q[h++];
        for (int e = head[v]; ~e; e = prv[e])
            if (cap[e] && lv[v] + 1 < lv[to[e]]){
                lv[to[e]] = lv[v] + 1;
                q[t++] = to[e];
            }
    }
    return lv[sink] < 1e8;
}

int dfs(int v, int sink, int f = 1e9){
    if (v == sink || f == 0)
        return f;
    int ret = 0;
    for (int &e = pt[v]; ~e; e = prv[e])
        if (lv[v]+1 == lv[to[e]]){
            int x = dfs(to[e], sink, min(f, cap[e]));
            cap[e] -= x;
```

```
cap[e^1] += x;
ret += x;
f -= x;
if (!f)
    break;
}
return ret;
}
```

```
int dinic(int source, int sink){
    int ret = 0;
    while (bfs(source, sink)){
        memcpy(pt, head, sizeof(head));
        ret += dfs(source, sink);
    }
    return ret;
}
```

```
int main(){
    memset(head, -1, sizeof(head));
    return 0;
}
```

4.1.2 MinCostMaxFlow

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

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define F first
#define S second

const int MAXN = 1e3 + 10;
const int MAXE = 1e4 + 10;

int from[MAXN], to[MAXN], cap[MAXN], ww[MAXN], prv[MAXN],
    head[MAXN], ec;

//edge from u to v, capacity uv, cost w, vu is backwards
//capacity(not necessary)
void addEdge(int u, int v, int uv, int w, int vu = 0){
    from[ec] = u, to[ec] = v, cap[ec] = uv, ww[ec] = w, prv[ec]
        = head[u], head[u] = ec++;
    from[ec] = v, to[ec] = u, cap[ec] = vu, ww[ec] = -w, prv[ec]
        = head[v], head[v] = ec++;
}
```

```

int lv[MAXN], q[MAXN], rec[MAXN], parE[MAXN];
bool mark[MAXN];
pii spfa(int source, int sink) {
    memset(lv, 63, sizeof(lv));
    memset(parE, -1, sizeof(parE));
    memset(rec, 0, sizeof(rec));
    memset(mark, 0, sizeof(mark));
    int h = 0, t = 0;
    lv[source] = 0;
    rec[source] = 1e9;
    q[t++] = source;
    mark[source] = true;
    while (t < h){
        int v = q[h++];
        if (h == MAXN) h = 0;
        mark[v] = false;
        for (int e = head[v]; ~e; e = prv[e])
            if (cap[e] && lv[v] + ww[e] < lv[to[e]]){
                lv[to[e]] = lv[v] + ww[e];
                parE[to[e]] = e;
                rec[to[e]] = min(cap[e], rec[v]);
                if (!mark[to[e]]){
                    q[t++] = to[e];
                    if (t == MAXN) t = 0;
                    mark[to[e]] = true;
                }
            }
    }
    if (lv[sink] > 1e8) return {0, 0};

    int v = sink;
    int totW = 0;
    while (~parE[v]){
        int e = parE[v];
        cap[e] -= rec[sink];
        cap[e^1] += rec[sink];
        totW += ww[e] * rec[sink];
        v = from[e];
    }
    return {rec[sink], totW};
}

//returns (flow, cost)
pii getMCMF(int source, int sink){
    pii ret(0, 0), x;
    while (true){
        x = spfa(source, sink);
        if (!x.F) break;
        ret.F += x.F;
    }
}

```

```

    ret.S += x.S;
}
return ret;
}

int main(){
    //only need to initialize these
    memset(head, -1, sizeof(head));
    ec = 0;
    return 0;
}

```

5 Math

5.1 Miller-Rabin

```

void factor(ll x, ll& e, ll& k){
    while(x%2LL==0LL){
        x/=2LL;
        ++e;
    }
    k = x;
}

//increase x for higher certainty, 5 works well
bool is_prime(ll n, int x){
    if(n%2LL==0 || n==1LL)
        return false;
    if(n==2 || n==3 || n==5 || n==7)
        return true;
    ll e, k;
    factor(n-1, e, k);
    while(x-->0){
        ll a = (rand())%(n-5LL) + 2LL;
        ll p = mod_exp(a, k, n);
        if(p==1LL || p==n-1LL)
            continue;
        bool all_fail = true;
        for(int i = 0; i < e-1; ++i){
            p = mod_exp(p, 2, n);
            if(p==n-1LL){
                all_fail = false;
                break;
            }
        }
        if(all_fail)
            return false;
    }
}

```

```

    return true;
}

```

5.2 fft

```

/* emaxx implementation */
/* Multiplication with arbitrary modulus
 * use ntt if mod is prime and can be written as 2**k * c
 * + 1
 * if not, use Chinese Remainder Theorem
 * or transform A(x) = A1(x) + A2(x)*c decompose into A(x)
 * /c and A(x)%c
 * B(x) = B1(x) + B2(x)*c
 * where c ~= sqrt(mod)
 * A * B = A1*B1 + c*(A1*B2 + A2*B1) * c**2(A2*B2)
 * with all values < sqrt(mod) subpolynomials have
 * coefficients < mod * N after fft multiply decreasing
 * changes of rounding error
 */

const double PI=acos(-1);

typedef complex<double> base;

void fft (vector<base> & a, bool invert) {
    int n=(int) a.size();
    for (int i=1, j=0; i<n; ++i) {
        int bit=n>>1;
        for (;j>=bit;bit>>=1)
            j-=bit;
        j+=bit;
        if(i<j)
            swap(a[i],a[j]);
    }
    for (int len=2; len<=n; len<=<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            base w(1);
            for (int j=0; j<len/2; ++j) {
                base u=a[i+j], v=a[i+j+len/2]*w;
                a[i+j]=u+v;
                a[i+j+len/2]=u-v;
                w*=wlen;
            }
        }
    }
    if (invert)
        for(int i=0;i<n;++i)

```

```

    a[i]/=n;
}

// a, b => coefs to multiply, res => resulting coefs
// a[0], b[0], res[0] = coef x^0
// Doesn't work with negative coefs
void multiply (const vector<int> & a, const vector<int> & b,
               vector<int> & res) {
    vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end()
    ());
    size_t n=1;
    while (n<max(a.size(),b.size())) n<<=1;
    n<<=1;
    fa.resize(n),fb.resize(n);
    fft (fa,false); fft(fb,false);
    for (size_t i=0; i<n; ++i)
        fa[i]*=fb[i];
    fft (fa, true);
    res.resize (n);
    // avoid precision errors, mess up with negative values of
    coefs
    for(size_t i=0; i<n; ++i)
        res[i]=int(fa[i].real() + 0.5);
}

```

5.3 math

/*
Picks theorem
Given a certain lattice polygon with non-zero area.

We denote its area by S , the number of points with integer coordinates lying strictly inside the polygon by I and the number of points lying on polygon sides by B .

Then, the Pick's formula states:

$$S = I + B/2 + 1$$

Burnsides Lemma
Let G be the finite group of operations we can perform on X . The number of orbits of X is the average of the number of fixed points for each g in G .
 G must be closed, associative, have identity, and inverses. A fixed point wrt g is an element of X such that $g.x = x$, That is, x is unchanged by the group operation.
For an element x , the orbit $G.x$ is the set of all possible results of transforming x . Orbits partition X .

Example: color a square under rotation
 G has 4 elements: rotate by x clockwise
 X contains 16 colorings if rotations are distinct
Count number of fixed points = $16 + 2 + 2 + 4 = 24$
Number of distinct colorings = $24/4 = 6$

Nim
Win if xor of pile sizes is not zero

Sprague-Grundy theorem

Let's consider a state v of a two-player impartial game and let vi be the states reachable from it (where $i \in \{1, 2, \dots, k\}$). To this state, we can assign a fully equivalent game of Nim with one pile of size x . The number x is called the Grundy value or nim-value of state v .

Moreover, this number can be found in the following recursive way:

$x = \text{mex}\{x_1, \dots, x_k\}$, where x_i is the Grundy value for state vi and the function mex (minimum excludant) is the smallest non-negative integer not found in the given set.

Viewing the game as a graph, we can gradually calculate the Grundy values starting from vertices without outgoing edges. Grundy value being equal to zero means a state is losing.

Number Theory

Totient function is the number of positive integers no more than n which is coprime with n .
Formula is $n * (\text{product over } p|n) (1 - 1/p)$
 $\sum_{d|n} \text{totient}(d) = n$

Mobius function

What is mobius function? This function has lots of definitions.

However, the main definition is the following.

$\mu(n)$ is 1 if $n=1$ or n is square-free and has even number of prime divisors.
 $\mu(n)$ is -1 if n is square-free and has odd number of prime divisors.
 $\mu(n)$ is 0 if n is not square-free.

$\mu(n)$ also has a lot of interesting properties that make $\mu(n)$ so important.

$\sum_{d|n} \mu(d) = 0$ for all $n > 1$, and $\mu(n)$ is multiplicative.
(A function f is multiplicative if $f(mn) = f(m)f(n)$ for all $(m, n) = 1$.)

Enumerating submasks in $O(3^n)$
for (int m=0; m<(1<<n); ++m)
for (int s=m; s; s=(s-1)&m)
... s and m ...

Catalan numbers

The first few numbers Catalan numbers, C_n (starting from zero):

1, 1, 2, 5, 14, 42, 132, 429, 1430,

$C_n = \frac{(2n \text{ choose } n)}{(n+1)}$

The Catalan number C_n is the solution for:

Number of correct bracket sequence consisting of n opening and n closing brackets.

The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.

The number of ways to completely parenthesize $n+1$ factors.
The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).

The number of ways to connect the $2n$ points on a circle to form n disjoint chords.

The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).

The number of monotonic lattice paths from point $(0,0)$ to point (n,n) in a square lattice of size nn , which do not pass above the main diagonal (i.e. connecting $(0,0)$ to (n,n)).

Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).

The number of non-crossing partitions of a set of n elements.

The number of ways to cover the ladder $1n$ using n rectangles (The ladder consists of n columns, where i th column has a height i).

*/

5.4 numbertheory

//Inverse of a mod m by extended euclidean

```

int inv(int a, int m)
{
    int temp=m, q, t, u=0, v=1;
    if(m==1) return 0;
    while(a>1)
    {
        q=a/m;
        t=m;
        m=a%m;
        a=t;
        t=u;
        u=v-q*u;
        v=t;
    }
    if(v<0) v+=temp;
    return v;
}

//modular inverse of all numbers in [1,n] in O(n).
int inv[111111], n;
ll mod=1e9+7;
int main(void)
{
    cin>>n;
    int i;
    inv[1]=1;
    for(i=2 ; i<=n ; i++)
    {
        inv[i]=((mod-mod/i)*inv[mod%i])%mod;
        cout<<inv[i]<<endl;
    }
}

//Totient function sieve
void preprocess(void)
{
    int i, j;
    eulerphi[1]=1;
    for(i=2 ; i<=122000 ; i++)
    {
        eulerphi[i]=i;
        primechk[i]=1;
    }
    for(i=2 ; i<=122000 ; i++)
    {
        if(primechk[i]==1)
        {
            eulerphi[i]=eulerphi[i]/i;
            for(j=2 ; i*j<=122000 ; j++)

```

```

        {
            primechk[i*j]=0;
            eulerphi[i*j]=eulerphi[i*j]/i;
        }
    }
}

//Mobius function sieve
void preprocess(void)
{
    int i, j;
    for(i=1 ; i<=111100 ; i++)
    {
        mu[i]=1;
        primechk[i]=1;
    }
    primechk[1]=0;
    for(i=2 ; i<=111100 ; i++)
    {
        if(primechk[i]==1)
        {
            mu[i]=-mu[i];
            for(j=2 ; i*j<=111100 ; j++)
            {
                primechk[i*j]=0;
                if(j%i==0)
                {
                    mu[i*j]=0;
                }
                else
                {
                    mu[i*j]=-mu[i*j];
                }
            }
        }
    }
}

```

5.5 simplex

```

// Two-phase simplex algorithm for solving linear programs
// of the form
//
// maximize    c^T x
// subject to  Ax <= b
//              x >= 0
//
// INPUT: A -- an m x n matrix

```

```

//      b -- an m-dimensional vector
//      c -- an n-dimensional vector
//      x -- a vector where the optimal solution will be
//           stored
//
// OUTPUT: value of the optimal solution (infinity if
//         unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b,
// and c as
// arguments. Then, call Solve(x).

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD &A, const VD &b, const VD &c) :
        m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D
            [i][j] = A[i][j];
        for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D
            [i][n+1] = b[i]; }
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j];
            }
        N[n] = -1; D[m+1][n] = 1;
    }

    void Pivot(int r, int s) {
        DOUBLE inv = 1.0 / D[r][s];
        for (int i = 0; i < m+2; i++) if (i != r)
            for (int j = 0; j < n+2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n+2; j++) if (j != s) D[r][j] *= inv;
        for (int i = 0; i < m+2; i++) if (i != r) D[i][s] *= -inv;
        ;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool Simplex(int phase) {
        int x = phase == 1 ? m+1 : m;
    }
}

```

```

while (true) {
    int s = -1;
    for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s]
            && N[j] < N[s]) s = j;
    }
    if (s < 0 || D[x][s] > -EPS) return true;
    int r = -1;
    for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||
            D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i]
                < B[r]) r = i;
    }
    if (r == -1) return false;
    Pivot(r, s);
}

DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] <= -EPS) {
        Pivot(r, n);
        if (!Simplex(1) || D[m+1][n+1] < -EPS) return -
            numeric_limits<DOUBLE>::infinity();
        for (int i = 0; i < m; i++) if (B[i] == -1) {
            int s = -1;
            for (int j = 0; j <= n; j++)
                if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s]
                    && N[j] < N[s]) s = j;
            Pivot(i, s);
        }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];
}
};

```

5.6 tricks

```

ll fexp(ll a, int x, ll mod){ // Fast exponenciation
    returns a^x % mod
    if(x==0)return 1ll;
    if(x%2==0){
        ll y=fexp(a, x/2, mod);
        return (y*y)%mod;
    }
    return (a*fexp(a, x-1, mod))%mod;
}

ll divv(ll a, ll b, ll mod){ // Division with mod returns a/
    b % mod
    return (a*fexp(b, mod-2, mod))%mod;
}

ll f[N];

ll fat(ll a, ll mod){ // Calculates factorial and stores in
    f % mod
    if(a<=1)return 1;
    return f[a]?f[a]:(f[a]=(a*fat(a-1, mod))%mod);
}

ll choose(ll n, ll k, ll mod){ // Returns n choose k % mod
    return divv(fat(n, mod), (fat(k, mod)*fat(n-k, mod))%mod,
        mod)%mod;
}

ll gcd(ll a, ll b){ // Greatest common divisor
    return b?gcd(b, a%b):a;
}

ll lcm(ll a, ll b){ // Least common multiple
    return (a*b)/gcd(a, b);
}

/* Fast factorization */

int p[N];

void start_fast(int MAX){ // Runs O(nlog(n)) Needs to be
    called to use fast_fact or ammount_of_divisors.
    for(int i=2; i<=MAX; i++){
        if(p[i]==0){
            for(int j=i; j<=MAX; j+=i){
                p[j]=i;
            }
        }
    }
}

```

```

}

vector<int>fast_fact(int x){ // Fast factorization in O(log2
    (x))
    vector<int>ret;
    while(x>1){
        ret.pb(p[x]);
        x/=p[x];
    }
    return ret;
}

int amount_of_divisors(int x){ // Calculate the ammount of
    divisors of a number in O(log2(x)) assume already ran
    start_fast.
    if(x==1)return 1;
    vector<int>v=fast_fact(x);
    int ret=1, curr=2;
    for(int i=1; i<v.size(); i++){
        if(v[i]==v[i-1])curr++;
        else{
            ret*=curr;
            curr=2;
        }
    }
    return ret*curr;
}

```

6 Misc

6.1 OrderedSet

```

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

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

#define F first
#define S second

int main() {

```

```

OrderedSet st;
st.insert({1, 22});
st.insert({1, 33});
st.insert({1, 44});
st.insert({1, 55});
cout << st.order_of_key({1,33}) << endl;
cout << st.order_of_key({1,35}) << endl; // Where would it
    be?
cout << (*st.find_by_order(2)).S << endl;
return 0;
}

```

6.2 template

```

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define F first
#define S second
#define se second
#define fi first
#define pb push_back
#define eb emplace_back
#define mk make_pair

#define N 1000007 //10e6 +7

int main(){
    ios::sync_with_stdio(false);

}

```

7 Python

7.1 InputArray

```

n = int(input())
a = list(map(int, input().split()));
ans = 1
v = []
for i in range(1, n):
    if(a[i] == 1):

```

```

    ans += 1
    v.append(a[i - 1])
v.append(a[n - 1])
print(len(v))
print(' '.join(map(str, v)));

```

8 Strings

8.1 kmp

```

/*
border = proper prefix that is suffix
p[i] = length of longest border of prefix of length i, s
[0...i-1]
*/

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

const int N = 1e6 + 6;
int pi[N];
string p, t;

void pre () {
    p += '#';

    pi[0] = pi[1] = 0;
    for (int i = 2; i <= (int)p.size(); i++) {
        pi[i] = pi[i-1];

        while (pi[i] > 0 and p[pi[i]] != p[i-1])
            pi[i] = pi[pi[i]];

        if (p[pi[i]] == p[i-1])
            pi[i]++;
    }
}

void report (int at) {

}

void KMP () {
    pre ();

    int k = 0;

```

```

int m = p.size() - 1;

for (int i = 0; i < (int)t.size(); i++) {
    while (k > 0 and p[k] != t[i])
        k = pi[k];

    if (p[k] == t[i])
        k++;
    if (k == m)
        report (i - m + 1);
}

}

int main (void) {
    ios_base::sync_with_stdio(false);

    return 0;
}

```

8.2 suffix-array

```

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define rank asdkflj
#define F first
#define S second

const int MAXN = 1e6 + 10;
const int LOG = 21;

/*
rank[i][j] = position of lex smallest (not continuous)
            string of len 2^i starting in position j
2^i = len of string, j = starting position

sa[i] = index where it starts
i = ith smallest suffix

lc[i] = longest common prefix between i, i+1 smallest
        suffix
Note: This is not the ith guy, but the ith smallest guy.

```

```

call lcp(i, j) to get the longest common prefix between the
    suffix starting
in i and the suffix starting in j
O(nlogn)
*/

// IMPORTANT: See if need to change to long long!

string s;
int rank[LOG][MAXN], n, lg;
pair<pair<int, int>, int> sec[MAXN];
int sa[MAXN], lc[MAXN];

int lcp(int a, int b){
    int _a = a;
    for (int w = lg-1; ~w && max(a, b) < n; w--){
        if (max(a, b) + (1<<w) <= n && rank[w][a] == rank[w][b])
            a += 1<<w, b += 1<<w;
    }
    return a - _a;
}

int cnt[MAXN];
pair<pii, int> gec[MAXN];
void srt() {
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < n; i++) cnt[sec[i].F.S+1]++;
    for (int i = 1; i < MAXN; i++) cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++) gec[--cnt[sec[i].F.S+1]] = sec[i];

    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < n; i++) cnt[gec[i].F.F+1]++;
    for (int i = 1; i < MAXN; i++) cnt[i] += cnt[i-1];
    for (int i = n-1; ~i; i--) sec[--cnt[gec[i].F.F+1]] = gec[i];
}

```

```

}

void build() {
    n = s.size();
    {
        int cur = 1; lg = 0;
        while (cur < n){
            lg++;
            cur <<= 1;
        }
        lg++;
    }

    for (int i = 0; i < n; i++) rank[0][i] = s[i];
    for (int w = 1; w < lg; w++){
        for (int i = 0; i < n; i++){
            if (i + (1<<w-1) >= n)
                sec[i] = {{rank[w-1][i], -1}, i};
            else
                sec[i] = {{rank[w-1][i], rank[w-1][i+(1<<w-1)]}, i};
        }
        //sort(sec, sec + n); This makes the code n*log^2, faster
        //to code.
        srt(); // Comment this if using above code

        rank[w][sec[0].S] = 0;
        for (int i = 1; i < n; i++){
            if (sec[i].F == sec[i-1].F)
                rank[w][sec[i].S] = rank[w][sec[i-1].S];
            else
                rank[w][sec[i].S] = i;
        }
        for (int i = 0; i < n; i++)
            sa[rank[lg-1][i]] = i;
        for (int i = 0; i + 1 < n; i++)
            lc[i] = lcp(sa[i], sa[i+1]);
    }
}

```

```

}



---


8.3 z

/*          {0, if i = 0
   z[i] = {length longest common prefix of s and s[i...n-1]
*/

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

const int N = 2e5 + 5;
string s;
int z[N];

void go () {
    int l = 0, r = 0;
    int n = s.size();
    memset (z, 0, sizeof z);

    for (int i = 1; i < n; i++) {
        if (i <= r)
            z[i] = min (z[i-1], r - i + 1);
        while (z[i] + i < n and s[z[i] + i] == s[z[i]])
            z[i]++;
        if (r < i + z[i] - 1) {
            l = i;
            r = i + z[i] - 1;
        }
    }
}

```