

# AHO

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

struct Node {
    bool terminal;
    Node *ptr[2];
    Node *fail;
    int color;
    int d;

    Node() : d(0), color(0), fail(NULL), terminal(false) {
        ptr[0] = ptr[1] = NULL; }

    ~Node() {
        for (int i = 0; i < 2; i++)
            if (ptr[i] && ptr[i]->d > d)
                delete ptr[i];
    }
};

void add(Node *root, std::string &s) {
    Node *temp = root;
    for (int i = 0; i < s.length(); i++) {
        int c = s[i] - '0';
        if (!temp->ptr[c])
            temp->ptr[c] = new Node();
        temp = temp->ptr[c];
    }
    temp->terminal = true;
}

void find_fail(Node *root) {
    std::queue<Node*> q;
    for (int i = 0; i < 2; i++)
        if (!root->ptr[i])
            root->ptr[i] = root;
        else {
            root->ptr[i]->fail = root;
            q.push(root->ptr[i]);
        }

    while (!q.empty()) {
        Node *v = q.front();
        q.pop();
        for (int i = 0; i < 2; i++) {
            Node *u = v->ptr[i];
            if (u) {
                q.push(u);

                Node *x = v->fail;
                while (!x->ptr[i])
                    x = x->fail;

                u->fail = x->ptr[i];
                if (u->fail->terminal)
                    u->terminal = true;
            }
        }
    }
}
```

```
void build_move(Node *root) {
    std::queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        Node *v = q.front();
        q.pop();
        for (int i = 0; i < 2; i++) {
            if (!v->ptr[i])
                v->ptr[i] = v->fail->ptr[i];
            else {
                v->ptr[i]->d = (v->d) + 1;
                q.push(v->ptr[i]);
            }
        }
    }

    bool find_cycle(Node *v) {
        v->color = 1;
        for (int i = 0; i < 2; i++) {
            if (v->ptr[i]->color == 1)
                return true;
            else if (v->ptr[i]->color == 0 && !v->ptr[i]->terminal
                && find_cycle(v->ptr[i]))
                return true;
        }
        v->color = 2;
        return false;
    }

    void solve() {
        int n;
        std::string s;
        std::cin >> n;

        Node *root = new Node();

        for (int i = 1; i <= n; i++) {
            std::cin >> s;
            add(root, s);
        }

        root->d = 0;
        find_fail(root);
        build_move(root);

        std::cout << ((find_cycle(root)) ? "TAK\n" : "NIE\n");

        delete root;
    }

    int main() {
        std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
        int z;
        std::cin >> z;
        while (z--)
            solve();
    }
}
```

## CENTROID DECOMPOSITION

```
#include <bits/stdc++.h>
#define st first
#define nd second
#define mk std::make_pair

/*
Clearing the lowest 1 bit:  $x \& (x - 1)$ , all trailing 1's:  $x \& (x + 1)$ 
Setting the lowest 0 bit:  $x | (x + 1)$ 
Enumerating subsets of a bitmask  $m$ :  $x=0$ ; do { ...;  $x=(x+1+m)\&m$ ; } while ( $x!=0$ );
__builtin_ctz/__builtin_clz returns the number of trailing/leading zero bits.
__builtin_popcount(unsigned x) counts 1-bits (slower than table lookups). */

// ===== CENTROID-DECOMPOSITION by Piotr Bejda =====
// usage: decompose(any vertex, 0, 0)
const int MAXN = 100010;
std::vector<int> edges[MAXN]; // input
bool taken[MAXN]; // input, = 0 before start
int size[MAXN]; // auxiliary
int jump[MAXN]; // output: parent
int depth[MAXN]; // output: layer
int dist[20][MAXN]; // output: dist from ancestor at given depth

void update_size(int u) {
    taken[u] = 1;
    size[u] = 1;
    for (int v : edges[u]) if (!taken[v]) {
        update_size(v);
        size[u] += size[v];
    }
    taken[u] = 0;
}

std::pair<int, int> centroid(int u, int n) {
    taken[u] = 1;
    std::pair<int, int> r = mk(n - size[u], u);
    for (int v : edges[u]) if (!taken[v]) r.st = std::max(r.st, size[v]);
    for (int v : edges[u]) if (!taken[v]) r = std::min(r, centroid(v, n));
    taken[u] = 0;
    return r;
}

void fill_dist(int u, int d, int layer){
    taken[u] = 1;
    dist[layer][u] = d;
    for (int v : edges[u]) if (!taken[v]) fill_dist(v, d+1, layer);
    taken[u] = 0;
}

void decompose(int u, int p, int layer){
    update_size(u);
    int n = size[u];
    u = centroid(u, n).nd;
    jump[u] = p;
    depth[u] = layer;
    fill_dist(u, 0, layer);
    taken[u] = 1;
    for (int v : edges[u]) if (!taken[v]) decompose(v, u, layer+1);
}
```

## CLOSEST POINTS

```
#include <bits/stdc++.h>
#define ll long long
#define st first
#define nd second
#define mk std::make_pair
#define debug if(0)

struct xy{
    ll x, y;
    xy() {}
    xy(ll X, ll Y) : x(X), y(Y) {}
};

int n;
std::vector<xy> v;
const ll inf = LLONG_MAX;

void input(){
    std::cin >> n;
    v.clear();
    ll x, y;
    for (int i = 1; i <= n; i++){
        std::cin >> x >> y;
        v.emplace_back(x, y);
    }
}

bool cmp1(const xy &A, const xy &B){
    return A.x < B.x;
}

bool cmp2(const xy &A, const xy &B){
    return A.y < B.y;
}

ll odl(const xy &A, const xy &B){
    return (A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y - B.y);
}

ll go(int L, int R){
    if (L == R)
        return inf;
    int mid = (L+R)/2;
    ll d = go(L, mid);
    d = std::min(d, go(mid+1, R));
    ll dd = ceil(sqrtl(d));
    ll res = inf;
    ll line = v[mid].x;
    std::vector<xy> left, right;
    for (int i = L; i <= mid; i++){
        if (line - v[i].x <= dd)
            left.push_back(v[i]);
    }
    for (int i = mid+1; i <= R; i++){
        if (v[i].x - line <= dd)
            right.push_back(v[i]);
    }
    std::sort(left.begin(), left.end(), cmp2);
    std::sort(right.begin(), right.end(), cmp2);
```

```

    int it, it1; it = it1 = 0;
    for (auto p: left){
        while (it < right.size() && p.y - right[it].y > dd)
            it++;
        it1 = it;
        while (it1 < right.size() && right[it1].y - p.y <= dd){
            res = std::min(res, odl(p, right[it1]));
            it1++;
        }
    }
    return std::min(d, res);
}

void solve(){
    input();
    std::sort(v.begin(), v.end(), cmp1);
    ll res = go(0, v.size()-1);
    std::cout << res << "\n";
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

## DINIC

```

#include <bits/stdc++.h>
#define ll long long
#define mk std::make_pair
#define st first
#define nd second

/*
 * Check whether you should change flows for long long!
 * MAX_N is graph size (vertices)
 * residual edges should be next to original
 */

```

```
const int inf = INT_MAX / 2;
```

```

struct Edge {
    int f, cap;
    Edge(int F, int CAP) : f(F), cap(CAP) {}
};

```

```
const int MAX_N = 205;
```

```

std::vector<Edge> e;
std::vector<std::pair<int, int> > list[MAX_N+3];
int n, m, N;

```

```

void input() {
    std::cin >> n >> m;
    N = n+m+2;

    for (int i = 0; i < N; i++)
        list[i].clear();
}

```

```
e.clear();
```

```

for (int i = 1; i <= n; i++) {
    int x; std::cin >> x;
    e.emplace_back(0, x);
    list[0].push_back(mk(i, e.size()-1));
    e.emplace_back(0, 0);
    list[i].push_back(mk(0, e.size()-1));
}

```

```

for (int i = n+1; i <= n+m; i++) {
    int x; std::cin >> x;
    e.emplace_back(0, x);
    list[i].push_back(mk(N-1, e.size()-1));
    e.emplace_back(0, 0);
    list[N-1].push_back(mk(i, e.size()-1));
}

```

```

for (int i = 1; i <= n; i++) {
    for (int j = n+1; j <= n+m; j++) {
        int x; std::cin >> x;
        if (x) {
            e.emplace_back(0, inf);
            list[i].push_back(mk(j, e.size()-1));
            e.emplace_back(0, 0);
            list[j].push_back(mk(i, e.size()-1));
        }
    }
}

```

```
int d[MAX_N+3];
```

```

bool bfs() {
    for (int i = 0; i < N; i++)
        d[i] = -1;
    d[0] = 0;
    std::queue<int> q;
    q.push(0);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (auto p : list[v]) {
            if (e[p.nd].f < e[p.nd].cap && d[p.st] == -1) {
                d[p.st] = d[v] + 1;
                q.push(p.st);
            }
        }
    }

    return d[N-1] != -1;
}

```

```
int next[MAX_N+3];
```

```

int dfs(int v, int flow) {
    if (!flow)
        return 0;
    if (v == N-1)

```

```

        return flow;

    while (next[v] < list[v].size()) {
        int i = next[v];
        next[v]++;

        int u = list[v][i].st;
        int id = list[v][i].nd;

        if (d[u] != d[v]+1)
            continue;

        if (e[id].f < e[id].cap) {
            int new_flow = dfs(u, std::min(flow, e[id].cap - e[id].f));
            if (new_flow > 0) {
                e[id].f += new_flow;
                e[id^1].f -= new_flow;
                return new_flow;
            }
        }
    }

    return 0;
}

int dinic() {
    int flow = 0;
    while (bfs()) {
        for (int i = 0; i < N; i++)
            next[i] = 0;
        while (1) {
            int new_flow = dfs(0, inf);
            if (!new_flow)
                break;
            flow += new_flow;
        }
    }
    return flow;
}

void solve() {
    input();
    std::cout << dinic() << "\n";
}

int main() {
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

## EULER

```

#include <bits/stdc++.h>
#define st first
#define nd second
#define mk std::make_pair

```

```

const int MAX_M = 5e5;

```

```

const int MAX_N = 5e4;
std::vector<std::pair<int, int>> list[MAX_N+4];
bool used[MAX_M+MAX_N+3];
int deg[MAX_N+4];
int n, m;
std::vector<int> e;

void clear(){
    for (int i = 0; i <= n+1; i++){
        list[i].clear();
        deg[i] = 0;
    }
    for (int i = 0; i <= m+n+2; i++)
        used[i] = false;
    e.clear();
}

void input(){
    std::cin >> n >> m;
    clear();
    for (int i = 1; i <= m; i++){
        int u, v;
        std::cin >> u >> v;
        list[u].push_back(mk(v, i));
        list[v].push_back(mk(u, i));
        deg[u]++; deg[v]++;
    }
}

void findCycle(int v){
    while (!list[v].empty()){
        while (!list[v].empty() && used[list[v].back().nd])
            list[v].pop_back();
        if (list[v].empty())
            break;
        std::pair<int, int> p = list[v].back();
        list[v].pop_back();
        used[p.nd] = true;
        findCycle(p.st);
    }
    e.push_back(v);
}

void goCycle(){
    // po prostu szukanie cyklu eulera w tym grafie
    findCycle(1);
    std::cout << 1 << "\n" << e.size() << "\n";
    for (auto x: e)
        std::cout << x << "\n";
    std::cout << "\n";
}

void goNormal(){
    // tworze ten wirtualny wierzcholek i wtedy szukam cyklu
    n++;
    int id = m+1;
    for (int i = 1; i <= n; i++){
        if (deg[i] % 2 == 1){
            list[i].push_back(mk(n, id));
            list[n].push_back(mk(i, id));

```

```

        id++;
    }
}
findCycle(n);
int i = 0;
int j;
std::vector<std::vector<int> > res;
std::vector<int> tmp;
while (i < e.size()-1){
    tmp.clear();
    j = i + 1;
    while (e[j] != n)
        j++;
    for (int k = i+1; k < j; k++)
        tmp.push_back(e[k]);
    res.push_back(tmp);
    i = j;
}
std::cout << res.size() << "\n";
for (auto w: res){
    std::cout << w.size() << " ";
    for (auto x: w)
        std::cout << x << " ";
    std::cout << "\n";
}
}

void solve(){
    input();
    bool found = false;
    for (int i = 1; i <= n; i++)
        if (deg[i] % 2 == 1)
            found = true;
    if (!found)
        goCycle();
    else
        goNormal();
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

### EXTENDED EUCLIDES

```

#include <bits/stdc++.h>

struct Tuple{
    int d, x, y;
    void set(int d, int x, int y){
        this->d = d; this->x = x; this->y = y;
    }
};

Tuple nwd(int a, int b){
    Tuple res;
    if (b == 0){

```

```

        res.set(a, 1, 0);
        return res;
    }
    res = nwd(b, a%b);
    res.set(res.d, res.y, res.x - (res.y * (a/b)));
    return res;
}

void solve(){
    int a, n;
    std::cin >> a >> n;
    Tuple res = nwd(a, n);
    if (res.d != 1)
        std::cout << "NIE ISTNIEJE\n";
    else
        std::cout << (res.x + n) % n << "\n";
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

### FFT

```

#include <bits/stdc++.h>

#define K long long

/* P = 2^k * c + 1, where c is odd
 * then we set N = 2^k
 * we first find generator g over field Z_P with hit and try method
 * then calculate g^c, that's the result
 *
 * if we want smaller N = 2^q for q < k,
 * then multiply the result times g^{2^{k-q}}
 *
 * the function below returns g^c,
 */

K P, g, N;

K fast_pow(K a, K b, K P) {
    if (b == 0) return 1;
    if (b % 2 == 0) { K tmp = fast_pow(a, b/2, P); return tmp * tmp % P; }
    return a * fast_pow(a, b-1, P) % P;
}

bool is_gen(K g, K P) {
    K q = g;
    for (K i = 1; i < P-1; i++) {
        if (g == 1)
            return false;
        g = g * q % P;
    }
    return true;
}

```

```
K find_gen(K P) {
    // copied from https://en.cppreference.com/w/cpp/numeric/random/
    // uniform_int_distribution
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(2, P-1);

    K g = 1;
    while (1) {
        g = distrib(gen);
        std::cout << "Checking␣" << g << "\n";
        fflush(stdout);
        if (is_gen(g, P)) {
            K Q = P-1; while (Q % 2 == 0) Q /= 2;
            std::cout << "Q=␣" << Q << "\n";
            return fast_pow(g, Q, P);
        }
    }
}

K find_root(K M) {
    if (M > N) return 1;
    return fast_pow(g, (1<<(N-M)), P);
}

void ntt(std::vector<K> &x, int d) {
    std::vector<K> e;
    int n = x.size(); e.resize(n+1);
    int pow = 0;
    while ((1<<pow) < n) pow++;
    e[0] = 1; e[1] = find_root(pow);
    if (d == -1) e[1] = fast_pow(e[1], P-2, P);
    for (int i = 2; i < n; i++) e[i] = e[i-1] * e[1] % P;
    for (int i = 0; i < n; i++) {
        int j = 0;
        for (int k = 1; k < n; k <= 1, j <= 1) if (k & i) j++;
        j >= 1; if (i < j) std::swap(x[i], x[j]);
    }
    int k=0;
    while ((1 << k) < n) k++;
    for(int s = 1; s < n; s <= 1){
        —k;
        for(int i = 0; i < n; i += 2*s) for (int j = 0; j < s; j++) {
            K u = x[i+j], v = x[i+j+s] * e[j<<k] % P;
            x[i+j] = u + v - (u+v >= P ? P : 0);
            x[i+j+s] = u - v + (u-v < 0 ? P : 0);
        }
    }
    if (d == -1) {
        K n_inv = fast_pow(n, P-2, P);
        for (int i = 0; i < n; i++) x[i] = x[i] * n_inv % P;
    }
}

std::vector<K> mul(std::vector<K> A, std::vector<K> B) {
    K pow = 1; int l = std::max(A.size(), B.size());
    while ((1<<pow) < l) ++pow;
    ++pow;
    A.resize(1<<pow); B.resize(1<<pow);
    ntt(A, 1); ntt(B, 1);
```

```
    for (int i = 0; i < A.size(); i++) A[i] = A[i] * B[i] % P;
    ntt(A, -1);
    while (A.size() > 1 && A.back() == 0) A.pop_back();
    return A;
}

std::vector<K> inv(std::vector<K> A) {
    int pow = 0;
    while ((1<<pow) < A.size())
        ++pow;
    A.resize((1<<pow)-1);
    std::vector<K> R(1, fast_pow(A[0], P-2, P));
    for (int k = 1; k <= pow; k++) {
        std::vector<K> B(A.begin(), A.begin() + (1<<k));
        R.resize(1<<(k+1)); B.resize(1<<(k+1));
        K w = find_root(k+1);
        ntt(R, 1); ntt(B, 1);
        for (int i = 0; i < (1<<(k+1)); i++)
            R[i] = (2 * R[i] % P - (R[i] * R[i] % P * B[i] % P)) % P;
        for (int i = 0; i < (1<<(k+1)); i++)
            R[i] = (R[i] + P) % P;
        ntt(R, -1);
        R.resize(1<<k);
    }
    return R;
}

int main() {
    srand(time(NULL));
    K P; std::cin >> P;
    /*
     * P = 998244353;
     * g = 879587319; (find_gen(P))
     * N = 23;
     */
}
```

GRAHAM

```
#include <bits/stdc++.h>
#define ld long double

struct XY{
    ld x, y;
    XY(){}
    XY(ld a, ld b) : x(a), y(b) {}
};

const ld EPS = 1e-9;
const int MAX_N = 2e5;
int n;
ld R;
std::vector<XY> v;
std::vector<XY> hull;

void input(){
    std::cin >> n >> R;
    ld x, y;
    v.clear();
    hull.clear();
```

```

    for (int i = 1; i <= n; i++){
        std::cin >> x >> y;
        v.emplace_back(x, y);
    }
}

int root;
XY P;

ld dist(XY A, XY B){
    return sqrt((A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y - B.y));
}

ld det(XY A, XY B, XY C){
    return (A.x*B.y + A.y*C.x + B.x*C.y) - (A.y*B.x + A.x*C.y + B.y*C.x);
}

bool cmp1(const XY &p1, const XY &p2){
    ld D = det(P, p1, p2);
    if (D > 0+EPS)
        return true;
    else if (D > 0-EPS)
        return dist(P, p1) <= dist(P, p2);
    return false;
}

void graham(){
    hull.push_back(P);
    if (!v.empty())
        hull.push_back(v[0]);
    for (int i = 1; i < v.size(); i++){
        while (v.size() >= 2 && det(hull[hull.size()-2], hull[hull.size()-1],
            v[i]) < 0+EPS)
            hull.pop_back();
        hull.push_back(v[i]);
    }
}

void solve(){
    input();
    root = 0;
    for (int i = 1; i < n; i++)
        if (v[i].y < v[root].y || (v[i].y == v[root].y && v[i].x < v[root].x))
            root = i;
    P = v[root];
    v.erase(v.begin()+root);
    std::stable_sort(v.begin(), v.end(), cmp1);
    graham();
    ld res = 0;
    for (int i = 0; i < hull.size()-1; i++)
        res += dist(hull[i], hull[i+1]);
    res += dist(hull.back(), hull[0]);
    res += 2*M_PI*R;
    std::cout << std::setprecision(2) << std::fixed << res << "\n";
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
```

```

    while (z--)
        solve();
}

KMR

#include <bits/stdc++.h>
#define mk std::make_pair
#define st first
#define nd second

const int MAX_N = 5e5;
int n;
std::string s;

int kmr[21][MAX_N+3];
int lcp[MAX_N+3];
int result[MAX_N+3];

void build_kmr() {
    std::vector<std::pair<std::pair<int, int>, int> > v;
    int h = 1;
    for (int k = 2; k/2 <= n; k *= 2, h++) {
        v.clear();
        for (int j = 0; j < n; j++) {
            if (j + k/2 < n)
                v.push_back(mk(mk(kmr[h-1][j], kmr[h-1][j + k/2]), j));
            else
                v.push_back(mk(mk(kmr[h-1][j], -1), j));
        }
        std::sort(v.begin(), v.end());
        int id = 0;
        int i = 0;
        int j;
        while (i < n) {
            j = i;
            while (j < n-1 && v[j+1].st == v[i].st)
                j++;
            for (int l = i; l <= j; l++)
                kmr[h][v[l].nd] = id;
            i = j+1;
            id++;
        }
    }
}

std::pair<int, int> get_hash(int i, int j) {
    int l = 0;
    while ((1<<(l+1)) <= (j-i+1))
        l++;
    return mk(kmr[l][i], kmr[l][j-(1<<l)+1]);
}

void solve() {
    std::cin >> n >> s;
    for (int i = 0; i < n; i++)
        kmr[0][i] = s[i];
    build_kmr();

    int l = 0;
```

```

    while ((1<<l) <= n)
        l++;

    std::vector<std::pair<int, int> > v;
    for (int i = 0; i < n; i++)
        v.push_back(mk(kmr[l][i], i));
    std::sort(v.begin(), v.end());

    for (int i = 0; i < n-1; i++) {
        int x = v[i].nd;
        int y = v[i+1].nd;

        int b = 0;
        int e = std::min(n-x, n-y);
        int mid;
        while (b<e) {
            mid = (b+e+1)/2;
            if (get_hash(x, x+mid-1) == get_hash(y, y+mid-1))
                b = mid;
            else
                e = mid-1;
        }
        lcp[i] = b;
    }

    for (int i = 0; i < n; i++) {
        int r = 0;
        if (i < n-1)
            r = std::max(r, lcp[i]);
        if (i > 0)
            r = std::max(r, lcp[i-1]);
        result[v[i].nd] = r;
    }

    for (int i = 0; i < n; i++)
        std::cout << result[i] << " ";
    std::cout << "\n";
}

int main() {
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

### MILLER RABIN

```

#include <bits/stdc++.h>
#define ll long long
#define int128 __int128_t

int128 pot(int128 a, int128 p, int128 mod){
    if (p == 0)
        return 1;
    else if (p % 2 == 0){
        int128 tmp = pot(a, p/2, mod);
        return (tmp * tmp)%mod;
    }
    return (pot(a, p-1, mod) * a)%mod;
}

```

```

}

bool witness(int128 a, int128 n){
    int128 _n = n-1;
    while (_n % 2 == 0)
        _n /= 2;
    int128 b = pot(a, _n, n);
    if (b == 1)
        return false;
    if (b == n-1)
        return false;
    while (_n < n-1){
        b = (b * b) % n;
        _n *= 2;
        if (b == n-1)
            return false;
    }
    return true;
}

void solve(){
    ll n; std::cin >> n;
    for (int i = 1; i <= 20; i++){
        ll a = rand()%(n-1)+1;
        if (witness(a, n)){
            std::cout << "NIE\n";
            return;
        }
    }
    std::cout << "TAK\n";
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}

```

### MINDISC

```

#include <bits/stdc++.h>
#define ld long double

const int MAX_N = 2e5;
const ld eps = 1e-10;

struct xy{
    ld x, y;

    xy(){ x = y = 0; }
    xy(ld a, ld b) : x(a), y(b) {}
};

struct Line{
    ld A, B, C;

    Line(){}
    Line(ld a, ld b, ld c) : A(a), B(b), C(c) {}
}

```



```

xy get_intersection(Line k){
    ld W = A*k.B - k.A*B;
    ld Wx = (-C)*k.B - (-k.C)*B;
    ld Wy = A*(-k.C) - k.A*(-C);
    xy M(Wx/W, Wy/W);
    return M;
}

};

struct Segment{
    xy A, B;

    Segment(){}
    Segment(xy X, xy Y) : A(X), B(Y) {}

    xy get_midpoint(){
        xy C((A.x+B.x)/2, (A.y+B.y)/2);
        return C;
    }

    Line get_line(){
        Line L(A.y - B.y, B.x - A.x, A.x*B.y - B.x*A.y);
        return L;
    }

    Line get_bisector(){
        xy M = get_midpoint();
        Line K = get_line();
        Line L(K.B, -K.A, -1);
        L.C = -(L.A * M.x + L.B * M.y);
        return L;
    }
};

ld distance(xy A, xy B){
    return sqrt((A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y - B.y));
}

struct Circle {
    int cnt; /* how many points define the circle ,
              if cnt==2, then A—B is the diameter */
    xy A, B, C;

    Circle(){}
    Circle(xy X) : A(X), cnt(1) {}
    Circle(xy X, xy Y) : A(X), B(Y), cnt(2) {}
    Circle(xy X, xy Y, xy Z) : A(X), B(Y), C(Z), cnt(3) {}

    xy get_center() {
        if (cnt == 2){
            Segment AB(A, B);
            return AB.get_midpoint();
        }
        Segment AB(A, B);
        Segment AC(A, C);
        Line K = AB.get_bisector();
        Line L = AC.get_bisector();
        xy center = K.get_intersection(L);
        return center;
    }
};

```

```

ld get_radius(){
    xy center = get_center();
    return distance(center, A);
}

bool is_inside(xy X){
    xy center = get_center();
    ld r = get_radius();
    return distance(center, X) <= r + eps;
}

};

int n;
std::vector<xy> v;

void input(){
    std::cin >> n;
    v.clear();
    ld x, y;
    for (int i = 1; i <= n; i++){
        std::cin >> x >> y;
        v.emplace_back(x, y);
    }
}

Circle min_disk3(int i, int j, int h){
    Circle CC(v[i], v[j], v[h]);
    return CC;
}

Circle min_disk2(int i, int j){
    Circle C(v[i], v[j]);
    for (int h = 0; h < j; h++){
        if (!C.is_inside(v[h]))
            C = min_disk3(i, j, h);
    }
    return C;
}

Circle min_disk1(int i){
    Circle C(v[i], v[0]);
    for (int j = 1; j < i; j++){
        if (!C.is_inside(v[j]))
            C = min_disk2(i, j);
    }
    return C;
}

void solve(){
    input();
    std::random_shuffle(v.begin(), v.end());
    Circle C(v[0], v[1]);
    for (int i = 2; i < n; i++){
        if (!C.is_inside(v[i])){
            C = min_disk1(i);
        }
    }
    std::cout << std::setprecision(12) << std::fixed << C.get_radius() << "\n";
}

int main(){

```

```
srand(time(NULL));
int z;
std::cin >> z;
while (z--)
    solve();
}
```

PICK

```
#include <bits/stdc++.h>
#define ll long long

struct xy{
    ll x, y;
    xy(ll X, ll Y) : x(X), y(Y) {}
};

std::vector<xy> v;
int n;

void input(){
    std::cin >> n;
    v.clear();
    ll x, y;
    for (int i = 1; i <= n; i++){
        std::cin >> x >> y;
        v.emplace_back(x, y);
    }
}

ll abss(ll x){
    if (x < 0)
        return x * (ll)(-1);
    return x;
}

ll gcd(ll x, ll y){
    if (y == 0)
        return x;
    return gcd(y, x%y);
}

ll det(xy A, xy B){
    // 0 0 1 0 0
    // Ax Ay 1 Ax Ay
    // Bx By 1 Bx By
    return A.x*B.y - A.y*B.x;
}

void solve(){
    input();
    // wynik = Pole - 1/2B + 1
    // 2*wynik = 2*Pole - B + 2
    ll P = 0;
    for (int i = 0; i < v.size()-1; i++)
        P += det(v[i], v[i+1]);
    P += det(v.back(), v[0]);
    P = abss(P);
    ll B = 0;
    for (int i = 0; i < v.size()-1; i++)
        B += gcd( abss(v[i].x - v[i+1].x), abss(v[i].y - v[i+1].y));
}
```

```
B += gcd( abss(v.back().x - v[0].x), abss(v.back().y - v[0].y));
ll res = P - B + 2;
std::cout << res/2 << "\n";
}

int main(){
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}
```

SIMPLE TEXT

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

/* pref pref */
void calc_pp(std::string &A, std::vector<int> &pp) {
    pp[1] = 0;
    int t = 1;
    for (int i = 2; i <= A.length()-1; i++) {
        if (i <= t+pp[t]-1)
            pp[i] = std::min(t+pp[t]-i, pp[i-t+1]);
        while (A[pp[i]+1] == A[i+pp[i]])
            pp[i]++;
        if (i + pp[i] - 1 >= t + pp[t] - 1)
            t = i;
    }
}

/* pref suf */
void calc_ps(std::string &A, std::vector<int> &ps) {
    int n = A.length();
    ps[0] = ps[1] = 0;
    for (int i = 2; i <= n; i++) {
        ps[i] = 0;
        int p = ps[i-1];
        while (p > 0 && A[i] != A[p+1])
            p = ps[p];
        if (A[i] == A[p+1])
            ps[i] = p+1;
    }
}

/* manacher */
void calc_manacher(std::string &s, std::vector<int> &pp) {
    int n = s.length();
    pp[1] = 0;
    int i = 1;
    int t = 0;
    while (i <= n){
        while (s[i-t-1] == s[i+t+1])
            t++;
        pp[i] = t;
        int k = 1;
        while (k <= t && pp[i]-k != pp[i-k]){
            pp[i+k] = std::min(pp[i]-k, pp[i-k]);
            k++;
        }
        t -= k;
    }
}
```

```
        t = std::max(0, t);
        i += k;
    }
}
```

TURBOMATCH

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

/*
 * to find min vertex cover / max independent set
 * take visited on the left and not visited on the right
 * or make it the other way around (i dont remember)
 */

const int MAX_N = 3e3;
int n, m;
std::vector<int> list [MAX_N+3];

void input() {
    std::cin >> n >> m;

    for (int i = 0; i <= 2*n-1; i++)
        list[i].clear();

    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;
        u--;
        v--;
        list[2*u+1].push_back(2*v);
        list[2*v].push_back(2*u+1);
    }
    n*=2;
}

bool visited [MAX_N+3];
int mate [MAX_N+3];

bool path_dfs(int v) {
    visited[v] = true;
    for (auto u : list[v])
        if (mate[u] == -1 || (!visited[mate[u]] && path_dfs(mate[u]))) {
            mate[u] = v;
        }
    }
```

```
        mate[v] = u;
        return true;
    }
    return false;
}

bool find_paths() {
    for (int i = 0; i < n; i++)
        visited[i] = false;
    bool new_path = false;
    for (int i = 0; i < n; i += 2)
        if (mate[i] == -1 && !visited[i])
            if (path_dfs(i))
                new_path = true;
    return new_path;
}

int turbo_matching() {
    for (int i = 0; i < n; i++)
        mate[i] = -1;
    while (1)
        if (!find_paths())
            break;

    int match_size = 0;
    for (int i = 0; i < n; i += 2)
        if (mate[i] != -1)
            match_size++;
    return match_size;
}

void solve() {
    input();
    std::cout << ( (turbo_matching() == n/2) ? "TAK\n" : "NIE\n" );
}

int main() {
    std::ios_base::sync_with_stdio(0); std::cin.tie(NULL);
    int z;
    std::cin >> z;
    while (z--)
        solve();
}
```