algo-know-yeah Team Note

ID	이름
singun11	김신건
antifly55	서형빈
Neogulee	최성훈

Table of contents

4. Fermat
5. FFT
6. ConvexHullTrick
7. Lis
8. Knapsack
9. Coin Change
10. Knuth Opti
11. twonearpoint

0. Base

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

#define for1(s,n) for(int i = s; i < n; i++)
#define for1j(s,n) for(int j = s; j < n; j++)
#define foreach(k) for(auto i : k)
#define foreachj(k) for(auto j : k)
#define pb(a) push_back(a)
#define sz(a) a.size()

using namespace std;
typedef unsigned long long ull;
typedef long long ll;
typedef vector <int> iv1;
typedef vector <vector<int>> iv2;
typedef vector <ll> llv1;
typedef vector <llv1> llv2;
typedef unsigned int uint;
typedef vector <ull> ullv1;
typedef vector <vector <ull>> ullv2;
typedef pair<int, int> pii;
typedef pair<ll, ll> ll;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

}
```

1. Graph

1.1. dijkstra

```
#define MAX 100000
#define INF (ll)1e18

struct edge {
    ll node;
```

```
    ll cost;
    bool operator<(const edge &to) const {
        return cost > to.cost;
    }
};

struct WGraph {
    ll n;
    vector<vector<edge>> adj;
    llv1 prev;
    WGraph(ll n) : n{n}, adj(n+1) {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({e, cost});
    }

    void input(ll m) { // 단방향
        ll a, b, c;
        while(m--) {
            cin >> a >> b >> c;
            addEdge(a,b,c);
        }
    }

    void inputD(ll m) { // 양방향
        ll a, b, c;
        while(m--){
            cin >> a >> b >> c;
            addEdge(a,b,c);
            addEdge(b,a,c);
        }
    }

    llv1 dijkstra(ll s) {
        llv1 dist(n+1, INF);
        prev.resize(n+1, -1);
        priority_queue<edge> pq;
        pq.push({ s, 0ll });
        dist[s] = 0;
        while (!pq.empty()) {
            edge cur = pq.top();
            pq.pop();
            if (cur.cost > dist[cur.node]) continue;
            for (auto &nxt : adj[cur.node])
                if (dist[cur.node] + nxt.cost < dist[nxt.node]) {
                    prev[nxt.node] = cur.node;
                    dist[nxt.node] = dist[cur.node] + nxt.cost;
                    pq.push({ nxt.node, dist[nxt.node] });
                }
        }
        return dist;
    }

    llv1 getPath(ll s, ll e) {
        llv1 ret;
```

```
        ll current = e;
        while(current != -1) {
            ret.push_back(current);
            current = prev[current];
        }
        reverse(ret.begin(), ret.end());
        return ret;
    }
};
```

1.2. bellman-ford

```
#define MAX 100010
#define INF (ll)1e18

struct edge {
    int to, cost;
};

int n;
vector<edge> v[MAX];
ll D[MAX];
bool bellman(ll start_point){
    fill(D,D+n+1, INF);
    D[start_point] = 0;

    bool isCycle = false;
    for1(1, n+1) {
        for1j(1, n+1) {
        for(int k=0; k<sz(v[j]); k++) {
            edge p = v[j][k];
            int end = p.to;
            ll dist = D[j] + p.cost;
                if (D[j] != INF && D[end] > dist) {
                    D[end] = dist;
                    if (i == n) isCycle = true;
                }
            }
        }
    }
    return isCycle;
}
```

1.3. kruskal

```
int root[MAXN];
int level[MAXN];

class Edge{
```

```
public:
    int node[2];
    int distance;
    Edge(int a, int b, int distance){
        this->node[0] = a;
        this->node[1] = b;
        this->distance = distance;
    }

    bool operator<(Edge &edge){
        return this->distance < edge.distance;
    }
};

void init(int n) {
    for1(0, n){
        root[i] = i;
        level[i] = 1;
    }
}

int find(int x) {
    return root[x] == x ? x : root[x] = find(root[x]);
}

// merge와 동시에 cycle 여부 확인
bool merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return true;
    if (level[x] < level[y]) root[x] = y;
    else root[y] = x;
    if (level[x] == level[y]) level[x]++;
    return false;
}
```

1.4. prim

```
struct edge {
    ll crt;
    ll node, cost;
};

struct WGraph {
    ll V;
    vector<edge> adj[MAX];
    vector<ll> prev;
    WGraph(ll V) : V{V} {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({s, e, cost});
        adj[e].push_back({e, s, cost});
```

```
    }

    ll prim(vector<edge> &selected) { // selected에 선택된 간선정보 vector 담김
        selected.clear();

        vector<bool> added(V, false);
        llv1 minWeight(V, INF), parent(V, -1);

        ll ret = 0;
        minWeight[0] = parent[0] = 0;
        for (int iter = 0; iter < V; iter++) {
            int u = -1;
            for (int v = 0; v < V; v++) {
                if (!added[v] && (u == -1 || minWeight[u]>minWeight[v]))
                    u = v;
            }

            if (parent[u] != u)
                selected.push_back({parent[u], u, minWeight[u]});

            ret += minWeight[u];
            added[u] = true;

            for1(0, sz(adj[u])) {
                int v = adj[u][i].node, weight = adj[u][i].cost;
                if (!added[v] && minWeight[v]>weight) {
                    parent[v] = u;
                    minWeight[v] = weight;
                }
            }
        }
        return ret;
    }
};
```

1.5. topological sort

```
int n;
int link[MAXN];
iv1 graph[MAXN];

void topologySort() {
    iv1 result;
    queue<int> q;

    for1(1, n+1) {
        if(link[i] == 0) q.push(i);
    }

    while(!q.empty()) {
        int x = q.front();
```

```
        q.pop();
        result.pb(x);

        for1(0, sz(graph[x])) {
            int y = graph[x][i];
            if(--link[y]==0) q.push(y);
        }
    }

    for1(0, n) {
        cout << result[i] << " ";
    }
}
```

1.6. union-find

```
int root[MAXN];
int level[MAXN];

void init(int n) {
    for1(0, n){
        root[i] = i;
        level[i] = 1;
    }
}

int find(int x) {
    return root[x] == x ? x : root[x] = find(root[x]);
}

void merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (level[x] < level[y]) root[x] = y;
    else root[y] = x;
    if (level[x] == level[y]) level[x]++;
}
```

1.7. SCC

```
int ans, cnt;
int visit[MAX], sn[MAX];
bool finished[MAX];
vector<int> adj[MAX];
stack<int> st;
vector<vector<int> > scc;

int dfs(int curr){
```

```
    visit[curr] = ++cnt;
    st.push(curr);

    int result = visit[curr];
    for(int i = 0; i < adj[curr].size(); i++){
        int next = adj[curr][i];
        if(visit[next] == 0) result = min(result, dfs(next));
        else if(!finished[next]) result = min(result, visit[next]);
    }

    if(result == visit[curr]){
        vector<int> currSCC;
        while(1){
            int t = st.top();
            st.pop();
            currSCC.push_back(t);
            finished[t] = true;
            sn[t] = ans;
            if(t == curr) break;
        }
        sort(currSCC.begin(), currSCC.end());
        scc.push_back(currSCC);
        ans++;
    }
    return result;
}

void makeSCC(int v){
    for(int i=1; i<=v; i++)
        if(!visit[i]) dfs(i);
    sort(scc.begin(), scc.end());
}
```

1.8. Maximum flow(dinic)

```
#define MAX_V 101
#define SRC 1
#define SINK MAX_V-1
#define INF (ll)1e18

struct Edge {
    ll v, capacity, rev;
    Edge(ll v, ll capacity, ll rev): v(v), capacity(capacity), rev(rev) {}
};

vector<Edge> vt[MAX_V];
ll level[MAX_V];
ll work[MAX_V];


void addEdge(ll start, ll end, ll capacity) {
```

```
    vt[start].emplace_back(end, capacity, (ll)vt[end].size());
    vt[end].emplace_back(start, capacity, (ll)vt[start].size()-1);
}

// 레벨 그래프 만드는 BFS
bool bfs() {
    memset(level, -1, sizeof(level));        //레벨 그래프 초기화
    queue <ll> q;
    level[SRC] = 0;
    q.push(SRC);

    while(!q.empty()){
        int here = q.front(); q.pop();
        for (auto i : vt[here]) {
            ll there = i.v;
            if(level[there] == -1 && i.capacity > 0) {
                level[there] = level[here] + 1;
                q.push(there);
            }
        }
    }
    return level[SINK] != -1;
}


ll dfs(ll here, ll crt_capacity) {
    if(here == SINK) return crt_capacity;

    for(ll &i = work[here]; i < vt[here].size(); i++) {
        ll there = vt[here][i].v;
        ll capacity = vt[here][i].capacity;

        if(level[here] + 1 == level[there] && capacity > 0) {
            ll next_capacity = dfs(there, min(crt_capacity, capacity));

            if(next_capacity > 0) {
                vt[here][i].capacity -= next_capacity;
                vt[there][vt[here][i].rev].capacity += next_capacity;
                return next_capacity;
            }
        }
    }
    return 0;
}

ll dinic() {
    ll ret = 0;
    while(bfs()) {
        memset(work, 0, sizeof(work));

        while(1) {
            ll flow = dfs(SRC, INF);
            if(!flow) break;
            ret += flow;
```

```
        }
    }
    return ret;
}
```

1.9. Maximum flow minimum cost

```
#define MX_N 100
#define MX_NODE 2*(MX_N+2)
#define SRC MX_NODE-2
#define SINK MX_NODE-1
#define INF 1000000000

ll N, M;
ll cost[MX_NODE][MX_NODE]; // 각 간선의 Cost
ll capacity[MX_NODE][MX_NODE]; // 각 간선의 용량
ll flow[MX_NODE][MX_NODE]; // 각 간선에 흐르고 있는 유량
llv1 edge[MX_NODE]; // 각 정점의 인접리스트

ll MCMF() {
    ll ret = 0;
    while(1) {
        ll prev[MX_NODE], dist[MX_NODE];
        bool isInQ[MX_NODE];
        queue<ll> Q;
        fill(prev, prev+MX_NODE, -1);
        fill(dist, dist+MX_NODE, INF);
        fill(isInQ, isInQ+MX_NODE, false);

        dist[SRC] = 0;
        Q.push(SRC);
        isInQ[SRC] = true;

        while(!Q.empty()) {
            ll current = Q.front();
            Q.pop();

            isInQ[current] = false;

            for(ll next: edge[current])
                if(capacity[current][next] - flow[current][next] > 0 && dist[next]
> dist[current] + cost[current][next]) {
                    dist[next] = dist[current] + cost[current][next];
                    prev[next] = current;

                    if(!isInQ[next]) {
                        Q.push(next);
                        isInQ[next] = true;
                    }
                }
        }
```

```
        if(prev[SINK] == -1) break;

        ll current_flow = INF;

        for(ll i = SINK; i != SRC; i = prev[i])
            current_flow = min(current_flow, capacity[prev[i]][i] - flow[prev[i]]
[i]);

        for(ll i = SINK; i != SRC; i = prev[i]) {
            ret += current_flow * cost[prev[i]][i];
            flow[prev[i]][i] += current_flow;
            flow[i][prev[i]] -= current_flow;
        }
    }
    return ret;
}
```

2. Tree

2.1. segment tree

```
ll a[MAX], tree[MAX * 4];

void init(int node, int x, int y) {
    if (x == y) {
        tree[node] = a[x];
        return;
    }
    int mid = (x + y)/2;
    init(node*2, x, mid);
    init(node*2 + 1, mid + 1, y);
    tree[node] = tree[node*2] + tree[node*2 + 1];
}

void update(int pos, ll val, int node, int x, int y) {
    if (pos < x || pos > y) return;
    if (x==y) {
        tree[node] = val;
        return;
    }
    int mid = (x + y)/2;
    update(pos, val, node*2, x, mid);
    update(pos, val, node*2 + 1, mid + 1, y);
    tree[node] = tree[node*2] + tree[node*2 + 1];
}

ll query(int lo, int hi, int node, int x, int y) {
    if (lo > y || hi < x) return 0;
    if (lo <= x && y <= hi) return tree[node];
    int mid = (x + y)/2;
```

```
    return query(lo, hi, node*2, x, mid) + query(lo, hi, node*2 + 1, mid + 1, y);
}
```

2.2. segment tree with lazy propagation

```
ll seg[4 * MAX], lazy[4 * MAX];

void update_lazy(ll node, ll x, ll y) {
    if (!lazy[node])
        return;
    seg[node] += (y - x + 1)*lazy[node];
    if (x != y) {
        lazy[node * 2] += lazy[node];
        lazy[node * 2 + 1] += lazy[node];
    }
    lazy[node] = 0;
}

ll update(ll lo, ll hi, ll val, ll node, ll x, ll y) {
    update_lazy(node, x, y);
    if (y < lo || hi < x)
        return seg[node];
    if (lo <= x && y <= hi) {
        lazy[node] += val;
        update_lazy(node, x, y);
        return seg[node];
    }
    ll mid = (x + y)/2;
    return seg[node] = update(lo, hi, val, node * 2, x, mid) + update(lo, hi, val,
node * 2 + 1, mid + 1, y);
}

ll query(ll lo, ll hi, ll node, ll x, ll y) {
    update_lazy(node, x, y);
    if (y < lo || hi < x)
        return 0;
    if (lo <= x && y <= hi)
        return seg[node];
    ll mid = (x + y)/2;
    return query(lo, hi, node * 2, x, mid) + query(lo, hi, node * 2 + 1, mid + 1,
y);
}
```

2.3. merge sort tree

```
llv1 a;
llv1 mTree[Mx];
void makeTree(ll idx, ll ss, ll se) {
    if(ss == se) {
```

```
        mTree[idx].push_back(a[ss]);
        return;
    }

    ll mid = (ss+se)/2;

    makeTree(2*idx+1, ss, mid);
    makeTree(2*idx+2, mid+1, se);

    merge(mTree[2*idx+1].begin(), mTree[2*idx+1].end(), mTree[2*idx+2].begin(),
mTree[2*idx+2].end(), back_inserter(mTree[idx]));
}

ll query(ll node, ll start, ll end, ll q_s, ll q_e, ll k) {
    //i j k: Ai, Ai+1, ..., Aj로 이루어진 부분 수열 중에서 k보다 큰 원소의 개수를 출력
한다.
    if (q_s > end || start > q_e) return 0;

    if (q_s <= start && q_e >= end) {
        return mTree[node].size() - (upper_bound(mTree[node].begin(),
mTree[node].end(), k) - mTree[node].begin());
    }

    ll mid = (start+end)/2;
    ll p1 = query(2*node+1, start, mid, q_s, q_e, k);
    ll p2 = query(2*node+2, mid+1, end, q_s, q_e, k);
    return p1 + p2;
}
```

2.4. LCA

```
struct LCA {
    vector<int> serials, no2se, se2no, loc; // length, loc의 index는 no
    vector<ll> length;
    SegmentTree *seg; // 최솟값 segment tree
    LCA(vector<vector<pair<int, ll>>> &edges) {
        int N = edges.size();
        no2se = vector<int>(N, -1);
        se2no = vector<int>(N, -1);
        loc = vector<int>(N, -1);
        length = vector<ll>(N, -1);
        length[0] = 0;
        vector<bool> visited(N, false);

        init_serials(0, visited, edges);
        seg = new SegmentTree(serials);
        seg->init(1, 1, serials.size());
    }

    void init_serials(int current, vector<bool>& visited, vector<vector<pair<int,
ll>>>&edges) {
```

```
        static int cnt = 0;
        visited[current] = true;
        if (no2se[current] == -1) {
            no2se[current] = cnt++;
            se2no[no2se[current]] = current;
            loc[current] = serials.size();
        }
        serials.push_back(no2se[current]);
        for1(0, edges[current].size()) {
            int next = edges[current][i].first;
            int cost = edges[current][i].second;
            if (visited[next])
                continue;
            length[next] = length[current] + cost;
            init_serials(next, visited, edges);
            serials.push_back(no2se[current]);
        }
        visited[current] = false;
    }

    ll query(int u, int v) { // 두 정점 사이의 거리
        if (loc[u] > loc[v])
            swap(u, v);
        ll lca = seg->query(loc[u] + 1, loc[v] + 1, 1, 1, serials.size());
        return length[u] + length[v] - 2ll * length[se2no[lca]];
    }
};
```

3. String

3.1. KMP

```
string content;
string obj;
int fail[MX];

vector <int> kmp (string s, string o) {
    fill(fail,fail+MX,0);
    vector<int> result;
    int N = s.length();
    int M = o.length();
    for(int i=1, j=0; i<M; i++){
        while(j > 0 && o[i] != o[j]) j = fail[j-1];
        if(o[i] == o[j]) fail[i] = ++j;
    }
    for(int i = 0, j = 0; i < N; i++) {
        while(j > 0 && s[i] != o[j]) j = fail[j-1];
        if(s[i] == o[j]) {
            if(j == M-1) { // matching OK;
                result.push_back(i - M + 2);
                j = fail[j];
```

```
            }
            else j++;
        }
    }
    return result;
}
```

3.2. Trie

```
int chToIdx(char ch) { return ch - 'a'; }
struct Trie {
    int terminal = -1;
    Trie* fail; // fail, output은 아호 코라식에 사용
    vector<int> output;
    Trie* chil[ALPHABETS];
    Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            chil[i] = NULL;
    }
    ~Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            if (chil[i])
                delete chil[i];
    }
    void insert(string& s, int number, int idx) {
        if (idx == s.size()) {
            terminal = number;
            return;
        }
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            chil[next] = new Trie();
        chil[next]->insert(s, number, idx + 1);
    }
    int find(string& s, int idx = 0) {
        if (idx == s.size())
            return terminal;
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            return false;
        return chil[next]->find(s, idx + 1);
    }
};
```

3.3 Aho-Corasick

```
void computeFail(Trie* root) {
    queue<Trie*> q;
    root->fail = root;
```

```
    q.push(root);
    while (!q.empty()) {
        Trie* here = q.front();
        q.pop();
        for (int i = 0; i < ALPHABETS; i++) {
            Trie* child = here->chil[i];
            if (!child) continue;
            if (here == root)
                child->fail = root;
            else {
                Trie* t = here->fail;
                while (t != root && t->chil[i] == NULL)
                    t = t->fail;
                if (t->chil[i]) t = t->chil[i];
                child->fail = t;
            }
            child->output = child->fail->output;
            if (child->terminal != -1)
                child->output.push_back(child->terminal);
            q.push(child);
        }
    }
}
vector<pair<int, int>> ahoCorasick(string& s, Trie* root) {
    vector<pair<int, int>> ret;
    Trie* state = root;
    for (int i = 0; i < s.size(); i++) {
        int idx = chToIdx(s[i]);
        while (state != root && state->chil[idx] == NULL)
            state = state->fail;
        if (state->chil[idx])
            state = state->chil[idx];
        for (int j = 0; j < state->output.size(); j++)
            ret.push_back({ i, state->output[j] });
    }
    return ret;
}
```

3.4 SuffixArray

```
struct SuffixComparator {
    const vector<int>& group;
    int t;
    SuffixComparator(const vector<int>& _group, int _t) :group(_group), t(_t) { }
    bool operator() (int a, int b) {
        if (group[a] != group[b])
            return group[a] < group[b];
        return group[a + t] < group[b + t];
    }
};
vector<int> getSuffixArr(const string& s) {
```

```
    int n = s.size();
    int t = 1;
    vector<int> group(n + 1);
    for (int i = 0; i < n; i++) group[i] = s[i];
    group[n] = -1;
    vector<int> perm(n);
    for (int i = 0; i < n; i++) perm[i] = i;
    while (t < n) {
        SuffixComparator compare(group, t);
        sort(perm.begin(), perm.end(), compare);
        t *= 2;
        if (t >= n) break;
        vector<int> new_group(n + 1);
        new_group[n] = -1;
        new_group[perm[0]] = 0;
        for (int i = 1; i < n; i++)
            if (compare(perm[i - 1], perm[i]))
                new_group[perm[i]] = new_group[perm[i - 1]] + 1;
            else
                new_group[perm[i]] = new_group[perm[i - 1]];
        group = new_group;
    }
    return perm;
}
int getHeight(const string& s, vector<int>& pos) // 최장 중복 부분 문자열의 길이
{
    const int n = pos.size();
    vector<int> rank(n);
    for (int i = 0; i < n; i++)
        rank[pos[i]] = i;
    int h = 0, ret = 0;
    for (int i = 0; i < n; i++)
    {
        if (rank[i] > 0) {
            int j = pos[rank[i] - 1];
            while (s[i + h] == s[j + h])
                h++;
            ret = max(ret, h);
            if (h > 0)
                h--;
        }
    }
    return ret;
}
```

4. Geometry

4.1. convexHull

```
struct point{
    ll x,y;
```

```
    ll p=0,q=0;
};

bool comp1(point a, point b) {
    if(a.y != b.y) return a.y < b.y;
    return a.x < b.x;
}

bool comp2 (point a, point b) {
    if(a.q * b.p != a.p*b.q)
        return a.q * b.p < a.p*b.q;
    return comp1(a,b);
}

ll ccw(point p1, point p2, point p3) {
    ll ret = (p1.x * p2.y + p2.x * p3.y + p3.x * p1.y - p2.x * p1.y - p3.x * p2.y
- p1.x * p3.y);
    return ret >0?1:(ret<0?-1:0);
}

vector <ll> getConvexHull(vector <point> ar) {
    vector <ll> stk;
    stk.push_back(0);
    stk.push_back(1);
    int next = 2;
    while(next < ar.size()) {
        while(stk.size() >= 2) {
            int s = stk.back();
            stk.pop_back();
            int f = stk.back();
            if(ccw(ar[f],ar[s],ar[next]) > 0) {
                stk.push_back(s);
                break;
            }
        }
        stk.push_back(next++);
    }

    return stk;
}


ll N;
point Z;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    vector <point> ar;

    cin >> N;
    for(int x=0; x<N; x++) {
```

```
        cin >> Z.x >> Z.y;
        ar.push_back(Z);
    }

    sort(ar.begin(),ar.end(),comp1);
    for(int x=1; x<N; x++) {
        ar[x].p = ar[x].x - ar[0].x;
        ar[x].q = ar[x].y - ar[0].y;
    }
    sort(ar.begin()+1,ar.end(),comp2);

    vector <ll> ret = getConvexHull(ar);
    cout << ret.size();
}
```

4.2. twofarpoint

```
ll getDist(point p, point q){
    return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y);
}

int main(){
    // convexhull
    int i, j=1;
    ll ans = 0;
    point p1, p2;
    for(i=0;i<ret.size();i++){
        int ni = (i+1)%ret.size();
        while(1){
            int nj = (j+1) % ret.size();
            int v = ccw({0, 0}, {ar[ret[ni]].x - ar[ret[i]].x, ar[ret[ni]].y -
ar[ret[i]].y}, {ar[ret[nj]].x - ar[ret[j]].x, ar[ret[nj]].y - ar[ret[j]].y});
            if(v > 0) j = nj;
            else break;
        }
        ll v = getDist(ar[ret[i]], ar[ret[j]]);
        if(ans < v){
            ans = v;
            p1 = ar[ret[i]];
            p2 = ar[ret[j]];
        }
    }
}
```

5. Extra

5.1. Treap

```
// Treap* root = NULL;
// root = insert(root, new Treap(3));
typedef int type;
struct Treap {
    Treap* left = NULL, * right = NULL;
    int size = 1, prio = rand();
    type key;
    Treap(type key) : key(key) { }
    void calcSize() {
        size = 1;
        if (left != NULL) size += left->size;
        if (right != NULL) size += right->size;
    }
    void setLeft(Treap* l) { left = l, calcSize(); }
    void setRight(Treap* r) { right = r, calcSize(); }
};
typedef pair<Treap*, Treap*> TPair;
TPair split(Treap* root, type key) {
    if (root == NULL) return TPair(NULL, NULL);
    if (root->key < key) {
        TPair rs = split(root->right, key);
        root->setRight(rs.first);
        return TPair(root, rs.second);
    }
    TPair ls = split(root->left, key);
    root->setLeft(ls.second);
    return TPair(ls.first, root);
}
Treap* insert(Treap* root, Treap* node) {
    if (root == NULL) return node;
    if (root->prio < node->prio) {
        TPair s = split(root, node->key);
        node->setLeft(s.first);
        node->setRight(s.second);
        return node;
    }
    else if (node->key < root->key)
        root->setLeft(insert(root->left, node));
    else
        root->setRight(insert(root->right, node));
    return root;
}
Treap* merge(Treap* a, Treap* b) {
    if (a == NULL) return b;
    if (b == NULL) return a;
    if (a->prio < b->prio) {
        b->setLeft(merge(a, b->left));
        return b;
    }
    a->setRight(merge(a->right, b));
    return a;
}
Treap* erase(Treap* root, type key) {
```

```
    if (root == NULL) return root;
    if (root->key == key) {
        Treap* ret = merge(root->left, root->right);
        delete root;
        return ret;
    }
    if (key < root->key)
        root->setLeft(erase(root->left, key));
    else
        root->setRight(erase(root->right, key));
    return root;
}
Treap* kth(Treap* root, int k) { // kth key
    int l_size = 0;
    if (root->left != NULL) l_size += root->left->size;
    if (k <= l_size) return kth(root->left, k);
    if (k == l_size + 1) return root;
    return kth(root->right, k - l_size - 1);
}
int countLess(Treap* root, type key) { // count less than key
    if (root == NULL) return 0;
    if (root->key >= key)
        return countLess(root->left, key);
    int ls = (root->left ? root->left->size : 0);
    return ls + 1 + countLess(root->right, key);
}
```

5.2. MCC

```
double getR(double x, double y){
    return x*x + y*y;
}

double avg(vector<double> x){
    double ans=0;
    for(int i=0; i<sz(x); i++) ans+=x[i];
    return ans/sz(x);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    double inputx, inputy, rx, ry, distance, lr=1;
    int n, index;
    vector<double> x, y;

    cin >> n;
    for1(0, n){
        cin >> inputx >> inputy;
```

```
        x.pb(inputx);
        y.pb(inputy);
    }

    rx = avg(x);
    ry = avg(y);

    for1(0, 100000){
        distance = -1; index = -1;
        for1j(0, n){
            if(distance < getR(x[j] - rx, y[j] - ry)){
                distance = getR(x[j] - rx, y[j] - ry);
                index = j;
            }
        }
        rx = rx + (x[index] - rx) * lr;
        ry = ry + (y[index] - ry) * lr;
        lr *= 0.999;
    }

    cout << fixed;
    cout.precision(2);
    cout << sqrt(distance) << endl;

    return 0;
}
```

5.3. ExtendEuclid

```
int gcd(int a, int b){
    if(b==0) return a;
    return gcd(b, a%b);
}

// ax+by=gcd(a,b)
pii ext_gcd(int a, int b){
    if(b==0) return pii(1, 0);
    pii tmp = ext_gcd(b, a%b);
    return pii(tmp.second, tmp.first - (a/b) * tmp.second);
}

// ax = 1 (mod b)
ll mod_inv(int a, int b){
    return (ext_gcd(a, b).first + b) % b;
}

/*
Ax + By = C일때
x0 = s * C/D
y0 = t * C/D
x = x0 + k * B/D
```

```
y = y0 - k * A/D
*/
```

5.4. Fermat

```
ll pow(ll a, ll b){
    if(b == 0) return 1;
    ll n = pow(a, b/2)%p;
    ll temp = (n * n)%p;

    if(b%2==0) return temp;
    return (a * temp)%p;
}

ll fermat(ll a, ll b){
    return a%p*pow(b, p-2)%p;
}
```

5.5. FFT

```
const double PI = acos(-1);
typedef complex<double> cpx;

void FFT(vector<cpx> &f, cpx w){
    int n = f.size();
    if(n == 1) return;

    vector<cpx> even(n/2), odd(n/2);
    for(int i = 0; i < n; ++i)
        (i%2 ? odd : even)[i/2] = f[i];

    FFT(even, w*w);
    FFT(odd, w*w);

    cpx wp(1, 0);
    for(int i = 0; i < n/2; ++i){
        f[i] = even[i] + wp*odd[i];
        f[i + n/2] = even[i] - wp*odd[i];
        wp *= w;
    }
}

vector<cpx> multiply(vector<cpx> a, vector<cpx> b){
    int n = 1;
    while(n < a.size()+1 || n < b.size()+1) n *= 2;
    n *= 2;
    a.resize(n);
    b.resize(n);
    vector<cpx> c(n);
```

```
    cpx w(cos(2*PI/n), sin(2*PI/n));

    FFT(a, w);
    FFT(b, w);

    for(int i = 0; i < n; ++i)
        c[i] = a[i]*b[i];

    FFT(c, cpx(1, 0)/w);
    for(int i = 0; i < n; ++i){
        c[i] /= cpx(n, 0);
        c[i] = cpx(round(c[i].real()), round(c[i].imag()));
    }
    return c;
}
```

5.6. ConvexHullTrick

```
struct linear{
    ll a, b;
    double s;
};

ll dp[MAX], top=0;
linear f[MAX];

double cross(linear &f, linear &g){
    return (g.b-f.b)/(f.a-g.a);
}

void addLine(ll a, ll b){ // y = ax + b
    linear g({a, b, 0});
    while(top > 0){
        g.s = cross(f[top-1], g);
        if(f[top-1].s < g.s) break;
        top--;
    }
    f[top++] = g;
}

ll searchLine(ll x){
    ll pos = top-1;
    if(x < f[top-1].s){
        ll lo = 0, hi = top-1;
        while(lo+1 < hi){
            ll mid = (lo+hi)/2;
            (x < f[mid].s ? hi:lo) = mid;
        }
        pos = lo;
    }
```

```
    return pos;
}
```

5.7. LIS

```
void lis(){
    int n, i, x;
    iv1 v, buffer;
    iv1::iterator vv;
    vector<pair<int, int> > print;
    v.pb(2000000000);

    cin >> n;
    for1(0, n){
        cin >> x;
        if(x > *v.rbegin()) {
            v.pb(x);
            print.push_back({v.size()-1, x});
        }
        else{
            vv = lower_bound(v.begin(), v.end(), x);
            *vv = x;
            print.push_back({vv-v.begin(), x});
        }
    }
    cout << sz(v) << endl;

    for(i=sz(print)-1;i>-1;i--){
        if(print[i].first == sz(v)-1){
            buffer.pb(print[i].second);
            v.pop_back();
        }
    }
    for(i=sz(buffer)-1;i>-1;i--) cout << buffer[i] << " ";
}
```

5.8. Knapsack

```
ll N, maxWeight,ans;
ll D[2][11000];
ll weight[110], cost[110];

void knapsack() {
    for(int x=1; x<=N; x++) {
        for(int y=0; y<=maxWeight; y++) {
            if(y>=weight[x]) {
                D[x%2][y] = max(D[(x+1)%2][y],D[(x+1)%2][y-weight[x]]+cost[x]);
            }
            else {
```

```
                D[x%2][y] = D[(x+1)%2][y];
            }
            ans = max(ans, D[x%2][y]);
        }
    }
}

void input() {
    cin >> N >> maxWeight;
    for(int x=1; x<=N; x++) {
        cin >> weight[x] >> cost[x];
    }
}
```

5.9. Coin Change

```
ll CC(llv1& coin, ll money, ll MX) {
    ll D[MX];
    fill(D, D+MX, 0);
    D[0] = 1;
    for(int i = coin.size()-1; i >=0; i--) {
        for(int j = coin[i]; j <= money; j++) {
            D[j] += D[j - coin[i]];
            D[j] %= MOD;
        }
    }
    return D[money] % MOD;
}
```

5.10. Knuth Opti

```
int solve(int n) {
    for (int m = 2; m <= n; m++) {
        for (int i = 0; m + i <= n; i++) {
            int j = i + m;
            for (int k = K[i][j - 1]; k <= K[i + 1][j]; k++) {
                int now = dp[i][k] + dp[k][j] + sum[j] - sum[i];
                if (dp[i][j] > now)
                    dp[i][j] = now, K[i][j] = k;
            }
        }
    }
    return dp[0][n];
}

int main() {
    int n;
    cin >> n;
    fill(&dp[0][0], &dp[MAX-1][MAX-1], INF);
```

```
    for (int i = 1; i <= n; i++){
        cin >> arr[i];
        sum[i] = sum[i - 1] + arr[i];
        K[i - 1][i] = i;
        dp[i - 1][i] = 0;
    }
    cout << solve(n) << "\n";
}

/*
if
C[a][c] + C[b][d] <= C[a][d] + C[b][c] (a<=b<=c<=d)
C[b][c] <= C[a][d] (a<=b<=c<=d)

then
dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]
range of k: A[i, j-1] <= A[i][j]=k <= A[i+1][j]
*/
```

5.11. twonearpoint

```
struct Point {
    int x, y;
};

int dist(Point &p, Point &q) {
    return (p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y);
}

struct Comp {
    bool comp_in_x;
    Comp(bool b) : comp_in_x(b) {}
    bool operator()(Point &p, Point &q) {
        return (this->comp_in_x? p.x < q.x : p.y < q.y);
    }
};

int nearest(vector<Point>::iterator it, int n) {
    if (n == 2)
        return dist(it[0], it[1]);
    if (n == 3)
        return min({dist(it[0], it[1]), dist(it[1], it[2]), dist(it[2], it[0])});

    int line = (it[n/2 - 1].x + it[n/2].x) / 2;
    int d = min(nearest(it, n/2), nearest(it + n/2, n - n/2));

    vector<Point> mid;

    for (int i = 0; i < n; i++) {
        int t = line - it[i].x;
        if (t*t < d)
```

```
            mid.push_back(it[i]);
    }

    sort(mid.begin(), mid.end(), Comp(false));

    int mid_sz = mid.size();
    for (int i = 0; i < mid_sz - 1; i++)
        for (int j = i + 1; j < mid_sz && (mid[j].y - mid[i].y)*(mid[j].y -
mid[i].y) < d; j++)
            d = min(d, dist(mid[i], mid[j]));

    return d;
}
```