## Algoritmo: DFS

```
vector<int> g[MAXN];
int n, m, idx, order[MAXN];
bool visited[MAXN];
stack<int> s;

// dfs recursivo
void dfs_visit(int x) {
    visited[x] = 1;
    order[++idx] = x;
    for (vector<int>::iterator it = g[x].begin(); it != g[x].end(); it++) {
        if (!visited[*it]) {
            dfs_visit(*it);
        }
    }
}

void dfs_recur() {
    for (int i = 1; i <= n; i++) {
        visited[i] = 0;
        order[i] = 0;
    }
    idx = 0;
    for (int i = 1; i <= n; i++) {
        if (!visited[i]) {
            dfs_visit(i);
        }
    }
}

// dfs iterativo
void dfs() {
    for (int i = 1; i <= n; i++) {
        visited[i] = 0;
        order[i] = 0;
    }
    int idx = 0;
    s.push(1);
    while (!s.empty()) {
        int x = s.top();
        s.pop();
        if (!visited[x]) {
            visited[x] = 1;
            order[++idx] = x;
            vector<int>::iterator it;
            for (it = g[x].begin(); it != g[x].end(); it++) {
                s.push(*it);
            }
        }
    }
}
```

## Algoritmo: BFS

```
vector<int> g[MAXN];
int n, m, order[MAXN];
bool visited[MAXN];
queue<int> q;

void bfs() {
    for (int i = 1; i <= n; i++) {
        visited[i] = 0;
        order[i] = 0;
    }
    int idx = 0;
    q.push(1);
    visited[1] = 1;
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        order[++idx] = x;
        vector<int>::iterator it;
        for (it = g[x].begin(); it != g[x].end(); it++) {
            if (!visited[*it]) {
                visited[*it] = 1;
                q.push(*it);
            }
        }
    }
}
```

## Algoritmo: Floyd-Warshall

```
int path[MAXN][MAXN];
int n, m;

void fw() {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                path[i][j] = min(path[i][j], path[i][k] + path[k][j]);
            }
        }
    }
}
```

## Algoritmo: Dijkstra

```
struct tedge {
    int from, to, weight;
    tedge(int _from = 0, int _to = 0, int _weight = 0) {
        from = _from;
        to = _to;
        weight = _weight;
    }
    bool operator() (const tedge &e1, const tedge &e2) {
        if (e1.weight < e2.weight)
            return 1;
        if (e1.weight > e2.weight)
            return 0;
        return e1.to < e2.to;
    }
};

vector<tedge> g[MAXN];
int n, m;

int dijkstra(int u, int v) {
    int sum = -1;
    set<tedge, tedge> dist;
    vector<tedge>::iterator it;
    bool chosen[MAXN];

    for (int i = 1; i <= n; i++) {
        chosen[i] = 0;
    }

    dist.insert(tedge(u, u, 0));
    while (!dist.empty()) {
        tedge e = *dist.begin();
        dist.erase(dist.begin());
        if (!chosen[e.to]) {
            chosen[e.to] = 1;
            if (e.to == v) {
                sum = e.weight;
                break;
            }
            for (it = g[e.to].begin(); it != g[e.to].end(); it++) {
                if (!chosen[it->to]) {
                    dist.insert(
                        tedge(it->from, it->to, e.weight + it->weight)
                    );
                }
            }
        }
    }

    return sum;
}
```

## Algoritmo: Prim

```
struct tedge {
    int from, to, weight;
    tedge(int _from = 0, int _to = 0, int _weight = 0) {
        from = _from;
        to = _to;
        weight = _weight;
    }
};

vector<tedge> g[MAXN];
int n, m;

struct comp {
    bool operator() (const tedge &e1, const tedge &e2) {
        if (e1.weight < e2.weight)
            return 1;
        if (e1.weight > e2.weight)
            return 0;
        if (e1.to < e2.to)
            return 1;
        return 0;
    }
};

int prim() {
    int sum = 0;
    set<tedge, comp> dist;
    vector<tedge>::iterator it;
    bool chosen[MAXN];

    for (int i = 1; i <= n; i++) {
        chosen[i] = 0;
    }

    dist.insert(tedge(1, 1, 0));
    while (!dist.empty()) {
        tedge e = *dist.begin();
        dist.erase(dist.begin());
        if (!chosen[e.to]) {
            chosen[e.to] = 1;
            sum += e.weight;
            for (it = g[e.to].begin(); it != g[e.to].end(); it++) {
                if (!chosen[it->to]) {
                    dist.insert(*it);
                }
            }
        }
    }

    return sum;
}
```

## Algoritmo: Ordenação Topológica (recursivo)

```
void topo_visit(int x) {
    visited[x] = 1;
    for (vector<int>::iterator it = g[x].begin(); it != g[x].end(); it++) {
        if (!visited[*it]) {
            topo_visit(*it);
        }
    }
    lifo.push(x);
}

void topo_sort() {
    for (int i = 1; i <= n; i++) {
        visited[i] = 0;
    }
    for (int i = 1; i <= n; i++) {
        if (!visited[i]) {
            topo_visit(i);
        }
    }
    int idx = 1;
    while (!lifo.empty()) {
        order[idx++] = lifo.top();
        lifo.pop();
    }
}
```

## Algoritmo: Ordenação Topológica (iterativo)

```
vector<int> g[MAXN];
int n, m, order[MAXN], degree[MAXN];

void topo_sort() {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (degree[i] == 0) {
            q.push(i);
        }
    }
    int idx = 0;
    while (!q.empty()) {
        int x = q.front();
        order[++idx] = x;
        q.pop();
        vector<int>::iterator it;
        for (it = g[x].begin(); it != g[x].end(); it++) {
            degree[*it]--;
            if (degree[*it] == 0) {
                q.push(*it);
            }
        }
    }
}
```

## Algoritmo: Kruskal

```
struct tedge {
    int from, to, weight;
    tedge() {}
    tedge(int _from, int _to, int _weight) {
        from = _from;
        to = _to;
        weight = _weight;
    }
};

tedge edges[MAXE];
int n, m, root[MAXN];

bool cmp(const tedge& e1, const tedge& e2) {
    return e1.weight < e2.weight;
}

int find_set(int x) {
    if (root[x] != x) {
        root[x] = find_set(root[x]);
    }
    return root[x];
}

void union_set(int x, int y) {
    int root_x = find_set(x), root_y = find_set(y);
    if (root_x != root_y) {
        root[root_y] = root[root_x];
    }
}

int kruskal() {
    int s = 0;
    sort(edges, edges+m, cmp);
    for (int i = 0; i <= n; i++) {
        root[i] = i;
    }
    for (int i = 0; i < m; i++) {
        tedge e = edges[i];
        if (find_set(e.from) != find_set(e.to)) {
            s += e.weight;
            union_set(e.from, e.to);
        }
    }
    return s;
}
```

## Algoritmo: Tarjan

```
vector<int> g[MAXN];
int n, m, index[MAXN], low[MAXN], idx, cont;
stack<int> s;

void visit(int v) {
    index[v] = low[v] = idx++;
    s.push(v);
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (index[*it] == -1) {
            visit(*it);
            low[v] = min(low[v], low[*it]);
        }
        else if (low[*it] != n) {
            low[v] = min(low[v], index[*it]);
        }
    }
    if (low[v] == index[v]) {
        int w;
        printf("%d -> ", cont);
        do {
            w = s.top();
            s.pop();
            printf("%d; ", w);
            low[w] = n;
        } while (v != w);
        printf("\n");
        cont++;
    }
}

void tarjan() {
    idx = cont = 0;
    for (int i = 1; i <= n; i++) {
        if (index[i] == -1) {
            visit(i);
        }
    }
}
```

## Algoritmo: Articulação ou Ponte em um grafo

```
int n, m, lbl[MAXN], low[MAXN], parent[MAXN], idx;
bool art[MAXN], has_art, has_brigde;
vector<int> g[MAXN];

void visit(int v) {
    int qnt = 0;
    lbl[v] = low[v] = idx++;
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (lbl[*it] == -1) {
            parent[*it] = v;
            visit(*it);
            low[v] = min(low[v], low[*it]);

            // Articulação
            {
                if (low[*it] >= lbl[v]) {
                    qnt++;
                }
            }

            // Ponte
            {
                if (low[*it] == lbl[*it]) {
                    printf("%d -> %d\n", v, *it);
                    has_bridge = 1;
                }
            }
        }
        else if (*it != parent[v]) {
            low[v] = min(low[v], lbl[*it]);
        }
    }

    // Articulação
    {
        if ((lbl[v] > 0 && qnt > 0) || (lbl[v] == 0 && qnt > 1)) {
            art[v] = 1;
        }
        has_art = has_art || art[v];
    }
}

void tarjan() {
    idx = 0;
    for (int i = 1; i <= n; i++) {
        if (lbl[i] == -1) {
            parent[i] = i;
            visit(i);
        }
    }
}
```

## Algoritmo: Fluxo Máximo

```
int g[MAXN][MAXN], n, m, parent[MAXN];
set<int> rg[MAXN];

bool bfs(int s, int t) {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        parent[i] = -1;
    }
    q.push(s);
    parent[s] = 0;
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        for (set<int>::iterator it = rg[x].begin(); it != rg[x].end(); it++)
{
            if (parent[*it] == -1) {
                parent[*it] = x;
                q.push(*it);
                if (*it == t) {
                    return 1;
                }
            }
        }
    }
    return 0;
}

int maxflow(int s, int t) {
    int flow = 0, f, x, px;
    while (bfs(s, t)) {
        x = t;
        f = g[parent[x]][x];
        while (px = parent[x], px != 0) {
            f = min(f, g[px][x]);
            x = px;
        }
        flow += f;
        x = t;
        while (px = parent[x], px != 0) {
            g[px][x] -= f;
            if (g[px][x] == 0) {
                rg[px].erase(x);
            }
            if (g[x][px] == 0) {
                rg[x].insert(px);
            }
            g[x][px] += f;
            x = px;
        }
    }
    return flow;
}
```

## Algoritmo: Grafo Bipartido

```
int n, m;
vector<int> g[MAXN];
bool visited[MAXN], color[MAXN];

bool colorbg(int v, bool c) {
    visited[v] = 1;
    color[v] = c;
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (!visited[*it] && !colorbg(*it, !c)) {
            return 0;
        } else if (color[*it] == c) {
            return 0;
        }
    }
    return 1;
}

bool isbg() {
    for (int i = 1; i <= n; i++) {
        if (!visited[i] && !colorbg(i, 0)) {
            return 0;
        }
    }
    return 1;
}
```

## Algoritmo: Kadane (Maximum Subarray Problem)

```
int n, a[MAXN];

int kadane() {
    int maxsum = 0, maxi = 0;
    for (int i = 0; i < n; i++) {
        maxi = max(0, maxi + a[i]);
        maxsum = max(maxsum, maxi);
    }
    return maxsum;
}
```

## Algoritmo: Coins Problem

```
int m, n, coin[MAXN], sum[MAXM];

void solve() {
    sum[0] = 0;
    for (int i = 1; i <= m; i++) {
        sum[i] = INF;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= m-coin[i]; j++) {
            sum[j+coin[i]] = min(sum[j+coin[i]], sum[j]+1);
        }
    }
}
```

## Algoritmo: Knapsack Problem

```
int p, w, value[MAXP], weight[MAXP], m[MAXP][MAXW];

void knapsack() {
    for (int i = 0; i < MAXP; i++)
        m[i][0] = 0;
    for (int j = 0; j < MAXW; j++)
        m[0][j] = 0;

    for (int i = 1; i <= p; i++) {
        for (int j = 1; j <= w; j++) {
            if (j >= weight[i]) {
                m[i][j] = max(m[i-1][j], m[i-1][j-weight[i]] + value[i]);
            } else {
                m[i][j] = m[i-1][j];
            }
        }
    }
}
```

## Algoritmo: Longest Common Substring

```
char a[MAXN], b[MAXN];
int lena, lenb, lenlcs, t[MAXN][MAXN];

void lcs() {
    lenlcs = 0;
    for (int i = 0; i <= lena; i++)
        t[i][0] = 0;
    for (int i = 0; i <= lenb; i++)
        t[0][i] = 0;

    for (int i = 1; i <= lena; i++) {
        for (int j = 1; j <= lenb; j++) {
            if (a[i] == b[j]) {
                t[i][j] = t[i-1][j-1] + 1;
                lenlcs = max(lenlcs, t[i][j]);
            } else {
                t[i][j] = 0;
            }
        }
    }
}

void print_lcs(int i, int j) {
    if (t[i][j] != 0) {
        print_lcs(i-1, j-1);
        printf("%c", a[i]);
    }
}

void find_lcs() {
    for (int i = lena; i >= lenlcs; i--) {
        for (int j = lenb; j >= lenlcs; j--) {
            if (t[i][j] == lenlcs) {
                print_lcs(i, j);
                printf("\n");
            }
        }
    }
}
```

## Algoritmo: Longest Common Subsequence

```
char a[MAXN], b[MAXN];
int lena, lenb, t[MAXN][MAXN];

void lcs() {
    for (int i = 0; i <= lena; i++)
        t[i][0] = 0;
    for (int i = 0; i <= lenb; i++)
        t[0][i] = 0;
    for (int i = 1; i <= lena; i++) {
        for (int j = 1; j <= lenb; j++) {
            if (a[i] == b[j])
                t[i][j] = t[i-1][j-1] + 1;
            else
                t[i][j] = max(t[i-1][j], t[i][j-1]);
        }
    }
}

void print_lcs(int i, int j) {
    if (i == 0 || j == 0)
        return;
    if (a[i] == b[j]) {
        print_lcs(i-1, j-1);
        printf("%c", a[i]);
    }
    else if (t[i-1][j] > t[i][j-1])
        print_lcs(i-1, j);
    else
        print_lcs(i, j-1);
}
```

## Algoritmo: Longest Increasing Subsequence

```
int n, m, a[MAXN], b[MAXN], p[MAXN];

void lis() {
    int u, v;
    b[m++] = 0;
    for (int i = 1; i < n; i++) {
        if (a[b[m-1]] < a[i]) {
            p[i] = b[m-1];
            b[m++] = i;
            continue;
        }
        for (u = 0, v = m-1; u < v; ) {
            int c = (u + v)/2;
            if (a[b[c]] < a[i])
                u = c + 1;
            else
                v = c;
        }
        if (a[i] < a[b[u]]) {
            if (u > 0)
                p[i] = b[u-1];
            b[u] = i;
        }
    }
    for (u = m, v = b[m-1]; u--; v = p[v])
        b[u] = v;
}
```

## Algoritmo: Optimal Tree

```
struct node {
    int r, f, c;
};

int n;
node tree[MAXN][MAXN];

void print_tree(int rmin, int rmax) {
    int r = tree[rmin-1][rmax].r;
    printf("%d ", r);
    if (rmin < r)
        print_tree(rmin, r-1);
    if (rmax > r)
        print_tree(r+1, rmax);
}

void optimal_tree() {
    int i, j, k, d, rmin, tmax, cmin, ck;
    scanf("%d", &n);
    for (i = 0; i <= n; i++) {
        tree[i][i].c = 0;
    }
    for (i = 0; i < n; i++) {
        j = i + 1;
        tree[i][j].r = j;
        scanf("%d", &tree[i][j].f);
        tree[i][j].c = tree[i][j].f;
    }
    for (d = 2; d <= n; d++) {
        for (i = 0; i <= n-d; i++) {
            j = i + d;
            tree[i][j].f = tree[i][j-1].f + tree[j-1][j].f;
            rmin = tree[i][j-1].r;
            rmax = tree[i+1][j].r;
            cmin = tree[i][rmin-1].c + tree[rmin][j].c + tree[i][j].f;
            for (k = rmin+1; k <= rmax; k++) {
                ck = tree[i][k-1].c + tree[k][j].c + tree[i][j].f;
                if (ck < cmin) {
                    rmin = k;
                    cmin = ck;
                }
            }
            tree[i][j].r = rmin;
            tree[i][j].c = cmin;
        }
    }

    printf("%d\n", tree[0][n].c);
}
```

## Algoritmo: KMP

```
char txt[MAXN], patt[MAXN];
int n, m, next[MAXN];

void pre_kmp() {
    int i, j;
    i = 2; j = 0;
    next[0] = -1; next[1] = 0;
    while (i < m) {
        if (patt[i-1] == patt[j]) {
            next[i++] = ++j;
        }
        else if (j > 0) {
            j = next[j];
        }
        else {
            next[i++] = 0;
        }
    }
}

int kmp() {
    int i, j;
    pre_kmp();
    i = j = 0;
    while (i+j < n) {
        if (txt[i+j] == patt[j]) {
            if (j == m-1)
                return i;
            j++;
        }
        else {
            i += j - next[j];
            if (next[j] > -1) {
                j = next[j];
            }
            else {
                j = 0;
            }
        }
    }
    return -1;
}
```

## Algoritmo: Minimum Enclosing Circle

```
const double INF = 1.0/0.0;
const double EPSILON = 1e-9;
const int LEFT = -1;
const int RIGHT = 1;
const int COLINEAR = 0;
const int MAXDOTS = 1001;

int cmp_double(const double &a, const double &b) {
    if (a == INF) {
        if (b == INF)
            return 0;
        return 1;
    }
    if (b == INF)
        return -1;
    if (a + EPSILON > b) {
        if (b + EPSILON > a)
            return 0;
        return 1;
    }
    return -1;
}

struct tdot {
    double x, y;
    tdot() {}
    tdot(double _x, double _y) {
        x = _x;
        y = _y;
    }
    bool operator() (const tdot &p, const tdot &q) {
        int cmp = cmp_double(p.y, q.y);
        if (cmp == -1)
            return 1;
        if (cmp == 1)
            return 0;
        cmp = cmp_double(p.x, q.x);
        return cmp == 1 ? 1 : 0;
    }
    bool operator== (const tdot &p) {
        return cmp_double(x, p.x) == 0 && cmp_double(y, p.y) == 0;
    }
    bool operator!= (const tdot &p) {
        return cmp_double(x, p.x) != 0 || cmp_double(y, p.y) != 0;
    }
    tdot operator+ (const tdot &p) { return tdot(x + p.x, y + p.y); }
    tdot operator- (const tdot &p) { return tdot(x - p.x, y - p.y); }
    tdot operator* (const double &k) { return tdot(x*k, y*k); }
    tdot operator/ (const double &k) { return tdot(x/k, y/k); }
};

struct tline {
    tdot p, v;
    tline() {}
    tline(tdot _p, tdot _v) {
        p = _p;
        v = _v;
    }
    tdot operator[] (const double &t) { return p + v*t; }
};
```

```
struct tcircle {
    tdot c;
    double rr;
    tcircle() {}
    tcircle(tdot _c, double _rr) {
        c = _c;
        rr = _rr;
    }
};

const tdot O = tdot(0, 0);
const tdot UNDEF_DOT = tdot(INF, INF);

int n;
tdot dot[MAXDOTS];
set<tdot, tdot> v;

double sqr(const double &a) {
    return a*a;
}

double dist(const tdot &a, const tdot &b) {
    return sqr(b.x-a.x)+sqr(b.y-a.y);
}

tline bisection(tdot p, tdot q) {
    tdot d = q - p;
    swap(d.x, d.y);
    d.x = -d.x;
    return tline((p+q)/2, d);
}

tdot intersection(tline r, tline s) {
    double ts = 0;
    if (s.v == O)
        return UNDEF_DOT;
    if (r.v.x == 0)
        ts = -(s.p.x - r.p.x)/s.v.x;
    else if (r.v.y == 0)
        ts = -(s.p.y - r.p.y)/s.v.y;
    else {
        double tmp1 = r.v.x*(s.p.y - r.p.y) - r.v.y*(s.p.x - r.p.x);
        double tmp2 = r.v.y*s.v.x - r.v.x*s.v.y;
        if (tmp2 != 0)
            ts = tmp1/tmp2;
        else
            return UNDEF_DOT;
    }
    return s[ts];
}

void find_dots(const tcircle &c) {
    v.clear();
    for (int i = 0; i < n; i++) {
        if (cmp_double(c.rr, dist(c.c, dot[i])) == 0) {
            v.insert(dot[i]);
        }
    }
}
```

```
pair<tdot, tdot> find_max_dist() {
    tdot p, q;
    double d;
    p = q = *v.begin(); d = 0;
    for (set<tdot>::iterator i = v.begin(); i != v.end(); i++) {
        for (set<tdot>::iterator j = i; ++j != v.end(); ) {
            if (cmp_double(d, dist(*i, *j)) < 0) {
                p = *i;
                q = *j;
                d = dist(p, q);
            }
        }
    }
    return make_pair(p, q);
}

int find_side(const tdot &p, const tdot &q, const tdot &c) {
    double d = (p.x*q.y + q.x*c.y + c.x*p.y) - (p.y*q.x + q.y*c.x + c.y*p.x);
    int cmp = cmp_double(d, 0);
    if (cmp > 0)
        return LEFT;
    if (cmp < 0)
        return RIGHT;
    return COLINEAR;
}

bool find_dot_arc(const tdot &p, const tdot &q, const int &side) {
    for (set<tdot>::iterator it = v.begin(); it != v.end(); it++) {
        if (find_side(p, q, *it) == side)
            return 1;
    }
    return 0;
}

void print_dot(const tdot &d, bool flag = 1) {
    printf("(%lf %lf)", d.x, d.y);
    if (flag)
        printf("\n");
    else
        printf("; ");
}

void print_line(const tline &r) {
    printf("(%lf, %lf) + (%lf, %lf)t\n", r.p.x, r.p.y, r.v.x, r.v.y);
}

void print_dots() {
    for (set<tdot, tdot>::iterator it = v.begin(); it != v.end(); it++) {
        print_dot(*it);
    }
}
```

```
int main() {
    int side;
    double newr;
    tdot p, q, newc;
    tcircle c;
    pair<tdot, tdot> pdots;
    scanf("%d", &n);
    newr = 0;
    for (int i = 0; i < n; i++) {
        scanf("%lf %lf", &dot[i].x, &dot[i].y);
        newr = max(newr, dist(O, dot[i]));
    }
    c = tcircle(O, newr);
    find_dots(c);
    if (v.size() < 2) {
        p = *v.begin();
        tline r = tline(c.c, p);
        newc = p; newr = 0;
        for (int i = 0; i < n; i++) {
            q = intersection(r, bisection(p, dot[i]));
            if (q != UNDEF_DOT) {
                if (cmp_double(newr, dist(p, q)) < 0) {
                    newc = q;
                    newr = dist(p, q);
                }
            }
        }
    }
    c = tcircle(newc, newr);
    find_dots(c);
    pdots = find_max_dist();
    p = pdots.first;
    q = pdots.second;
    side = find_side(p, q, c.c);
    while (side != COLINEAR && !find_dot_arc(p, q, side)) {
        tline r = bisection(p, q);
        newc = (p + q)/2;
        newr = dist(c.c, newc);
        for (int i = 0; i < n; i++) {
            if (find_side(p, q, dot[i]) == side) {
                tdot d = intersection(r, bisection(p, dot[i]));
                if (d != UNDEF_DOT) {
                    if (cmp_double(newr, dist(c.c, d)) > 0) {
                        newc = d;
                        newr = dist(c.c, d);
                    }
                }
            }
        }
        c = tcircle(newc, dist(newc, p));
        find_dots(c);
        pdots = find_max_dist();
        p = pdots.first;
        q = pdots.second;
        side = find_side(p, q, c.c);
    }
    printf("C = ");
    print_dot(newc);
    printf("Radius: %lf\n", sqrt(c.rr));
    printf("Dots at the circumference:\n");
    print_dots();
}
```

## Algoritmo: Convex Hull

```
const int INF = 1<<30;
const int MAXDOTS = 1001;

struct tdot {
    int x, y;
    tdot() {}
    tdot(int _x, int _y) {
        x = _x;
        y = _y;
    }
};

int n;
tdot dot[MAXDOTS], s[MAXDOTS];

bool cmp(const tdot &a, const tdot &b) {
    return atan2(a.y-dot[0].y,a.x-dot[0].x)<atan2(b.y-dot[0].y,b.x-dot[0].x);
}

int find_area(const tdot &a, const tdot &b, const tdot &c) {
    return (a.x*b.y + b.x*c.y + c.x*a.y) - (a.y*b.x + b.y*c.x + c.y*a.x);
}

int convex_hull() {
    int idx = 2;
    s[0] = dot[0];
    s[1] = dot[1];
    for (int i = 2; i < n; i++) {
        if (find_area(s[idx-2], s[idx-1], dot[i]) >= 0) {
            s[idx++] = dot[i];
        }
        else {
            do {
                idx--;
            } while (find_area(s[idx-2], s[idx-1], dot[i]) < 0);
            s[idx++] = dot[i];
        }
    }
    return idx;
}

int main() {
    int mindot = 0;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &dot[i].x, &dot[i].y);
        if (dot[i].x < dot[mindot].x)
            mindot = i;
        else if (dot[i].x == dot[mindot].x && dot[i].y < dot[mindot].y)
            mindot = i;
    }
    swap(dot[mindot], dot[0]);
    sort(dot+1, dot+n, cmp);
    int qnt = convex_hull();
    for (int i = 0; i < qnt; i++) {
        printf("(%d, %d)\n", s[i].x, s[i].y);
    }
}
```

## Implementação: Bignum

```
struct bignum {
    char n[MAXN];
    int lenn;

    bignum () {
        strcpy(n, "0");
        lenn = 1;
    }

    int reverse_number(char *num) {
        int i = 0, j = strlen(num) - 1;
        char aux;
        while (num[j] == '0')
            j--;
        num[j+1] = '\0';
        while (i < j) {
            aux = num[i];
            num[i] = num[j];
            num[j] = aux;
            i++, j--;
        }
        return i+j+1;
    }

    int remove_leading_zeros(char *num) {
        int i = 0, j = 0;
        while (num[i] == '0')
            i++;
        if (i > 0) {
            while (num[i] != '\0') {
                num[j++] = num[i++];
            }
            num[j] = '\0';
        }
        return j;
    }

    bignum& operator= (const char *_n) {
        strcpy(n, _n);
        lenn = strlen(n);
        return *this;
    }

    bignum operator+ (const bignum &a) {
        bignum s;
        int in = lenn-1, ia = a.lenn-1, is = 0, flag = 0;
        for ( ; in >= 0 && ia >= 0; in--, ia--, is++) {
            s.n[is] = n[in] + a.n[ia] - '0' + flag;
            flag = 0;
            if (s.n[is] > '9') {
                s.n[is] -= 10;
                flag = 1;
            }
        }
        for ( ; in >= 0; in--, is++) {
            s.n[is] = n[in] + flag;
            flag = 0;
            if (s.n[is] > '9') {
                s.n[is] -= 10;
                flag = 1;
            }
        }
        for ( ; ia >= 0; ia--, is++) {
            s.n[is] = a.n[ia] + flag;
            flag = 0;
            if (s.n[is] > '9') {
                s.n[is] -= 10;
                flag = 1;
            }
        }
        if (flag > 0)
            s.n[is++] = '1';
        s.n[is] = '\0';
        s.lenn = reverse_number(s.n);
        return s;
    }

    bignum operator- (const bignum &a) {
        bignum s;
        int in = lenn-1, ia = a.lenn-1, is = 0, flag = 0;
        for ( ; in >= 0 && ia >= 0; in--, ia--, is++) {
            s.n[is] = n[in] - a.n[ia] + '0' - flag;
            flag = 0;
            if (s.n[is] < '0') {
                s.n[is] += 10;
                flag = 1;
            }
        }
        for ( ; in >= 0; in--, is++) {
            s.n[is] = n[in] - flag;
            flag = 0;
            if (s.n[is] < '0') {
                s.n[is] += 10;
                flag = 1;
            }
        }
        for ( ; ia >= 0; ia--, is++) {
            s.n[is] = a.n[ia] + flag;
            flag = 0;
            if (s.n[is] > '9') {
                s.n[is] -= 10;
                flag = 1;
            }
        }
        s.n[is] = '\0';
        s.lenn = reverse_number(s.n);
        return s;
    }

    bignum operator* (const int &k) {
        bignum s;
        int in, is, tmp, flag = 0;
        for (in = lenn-1, is = 0; in >= 0; in--, is++) {
            tmp = (n[in] - '0')*k + flag;
            flag = tmp/10;
            tmp %= 10;
            s.n[is] = '0' + tmp;
        }
        while (flag > 0) {
            s.n[is++] = '0' + flag % 10;
            flag /= 10;
        }
        s.n[is] = '\0';
        s.lenn = reverse_number(s.n);
        return s;
    }
```

```
    bignum operator/ (const int &k) {
        bignum s;
        int in, is, mod = 0;
        for (in = 0, is = 0; in < lenn; in++, is++) {
            mod = 10*mod + n[in] - '0';
            if (mod >= k) {
                s.n[is] = '0' + mod/k;
                mod %= k;
            }
            else {
                s.n[is] = '0';
            }
        }
        s.n[is] = '\0';
        s.lenn = remove_leading_zeros(s.n);
        return s;
    }

    int operator% (const int &k) {
        int in, mod = 0;
        for (in = 0; in < lenn; in++) {
            mod = 10*mod + n[in] - '0';
            if (mod >= k) {
                mod %= k;
            }
        }
        return mod;
    }
};
```