1 Math

1.1 Gauss-Jordan Elimination

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

using namespace std;

typedef struct _Matrix{
    int N;
    vector<vector<double>> matrix;

    _Matrix(int X) {
        N = X;
        matrix.resize(N, vector<double>(N + 1));
    } // N by N + 1 matrix
} Matrix;

void row_swap(Matrix& A, int i) {
    vector<double> temp = A.matrix[i];
    A.matrix.erase(A.matrix.begin() + i);
    A.matrix.push_back(temp);
}

//Gauss-Jordan Elmination
void gauss_jordan(Matrix& A) {
    for(int i = 0; i < A.N; ++i) {
        while(A.matrix[i][i] == 0) row_swap(A, i); //check diagonal components
          are non-zero, when if, rotate row(swap)

        for(int j = 0; j < A.N; ++j) { //make RREF
            if(i != j) {
                double ratio = A.matrix[j][i] / A.matrix[i][i];
                for(int k = 0; k <= A.N; ++k) {
                    A.matrix[j][k] = A.matrix[j][k] - ratio * A.matrix[i][k];
                }
            }
        }
    }
}
```

1.2 Miller-Rabin Prime Test

```
#include <bits/stdc++.h>
using namespace std;

using ull = unsigned long long;

vector<ull> prime_list = {2, 7, 61};
//~ int range
//{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}
//~ long long range
//use __int128_t instead of ull
//O(klog^3(x))
ull mod_pow(ull a, ull b, ull M) {
    if(b == 0) return 1;

    ull temp = mod_pow(a, b / 2, M);

    if(b & 1) return (((temp % M) * (temp % M)) % M * (a % M)) % M;
    else return ((temp % M) * (temp % M)) % M;
}

bool miller_rabin(ull x) {
    if(x < 2) return false;

    bool ret = true;
    for(auto& p : prime_list) {
        if(x == p) return true;
        ull k = x - 1;
        while(true) {
            ull val = mod_pow(p, k, x) % x;

            if(val == x - 1) {
                ret = true;
                break;
            }
            if(k & 1) {
                ret = (val == 1 || val == x - 1);
                break;
            }

            k /= 2;
        }

        if(!ret) break;
    }

    return ret;
}
```

1.3 Fast-Fourier-Transform

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

using namespace std;
using cpx = complex<double>;
//Cooley-Tukey FFT
void FFT(vector<cpx>& A, cpx w) {
    int n = (int)A.size();
    if(n == 1) return;

    vector<cpx> even(n / 2), odd(n / 2);
    for(int i = 0; i < n; ++i) {
        if(i & 1) odd[i / 2] = A[i];
        else even[i / 2] = A[i];
    }

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

    cpx w_e(1, 0);

    for(int i = 0; i < n / 2; ++i) {
        A[i] = even[i] + w_e * odd[i];
        A[i + n / 2] = even[i] - w_e * odd[i];
        w_e *= w;
    }
}

void product(vector<cpx>& A, vector<cpx>& B) {
    int n = (A.size() <= B.size()) ? ceil(log2((double)B.size())) : ceil(log2((
      double)A.size()));
    n = pow(2, n + 1);

    A.resize(n);
    B.resize(n);
    vector<cpx> C(n);

    cpx w(cos(2 * acos(-1) / n), sin(2 * acos(-1) / 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()));
    }
}

/*
void FFT(vector<cpx>& A, bool invert) {
    int n = (int)A.size();

    for(int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;

        while(j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;

        if(i < j) swap(A[i], A[j]);
    }

    for(int length = 2; length <= n; length <<= 1) {
        double ang = 2 * PI / length * (invert ? -1 : 1);
        cpx w(cos(ang), sin(ang));

        for(int i = 0; i < n; i += length) {
            cpx w_i(1, 0);
            for(int j = 0; j < length / 2; ++j) {
                cpx u = A[i + j], v = A[i + j + length / 2] * w_i;
                A[i + j] = u + v, A[i + j + length / 2] = u - v;
                w_i *= w;
            }
        }
    }

    if(invert) {
        for(int i = 0; i < n; ++i) {
            A[i] /= cpx(n, 0);
            A[i] = cpx(round(A[i].real()), round(A[i].imag()));
        }
    }
} // referenced from https://blog.myungwoo.kr/54
*/ //faster version of FFT
```

1.4 수학 관련 노트

1. 모듈러 역원은 M이 소수일 때, $a^{M-2} mod M$ 이다.

2. a + b = c 인 (a, b, c) 개수 찾는 문제면 FFT 시도해보기.

3. 모든 케이스에 대한 경우의 수도 FFT 생각해보기.

4. 순서가 있는 쌍을 찾으라고 하면 정렬한 것과 비교하는 식으로 스위핑일 수 있음. 스위핑이면 보통 세그먼트 트리로 최적화해서 풀 수 있음.

5. 최적화 문제에서 뭔가 안되면, 이분탐색 / 삼분탐색 떠올려보기.

6. 가장 가까운 두 쌍 찾기는 거리를 기준으로 한 분할정복으로 찾을 수 있음.

7. 누가 이기는 건지 묻는걸로 바꿀 수 있으면, 게임이론 -> 스프라그 그런디 정리 쓸 수 있는 지 생각해보기

8. $dp[i] = max/min(a[i]b[j]+c[j])+d[i]$ 꼴에, b가 단조증가 단조감소이면 CHT임.

2 DP optimization

2.1 Convex Hull Trick

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

struct line {
    ll m, n;
    long double s;
};

ll dp[100'001];
line line_stack[100'001];
```

```
long double get_intersection(const line& a, const line& b) {
    return (long double)(a.n - b.n) / (long double)(b.m - a.m);
}

ll solve(vector<ll>& a, vector<ll>& b, int n) {
    int top = 0, cur = 0;

    for(int i = 2; i <= n; ++i) {
        line g = {b[i - 1], dp[i - 1], 0};

        while(top > 0) {
            g.s = get_intersection(line_stack[top - 1], g);

            if(line_stack[top - 1].s < g.s) break;

            if(--top == cur) cur--;
        }

        line_stack[top++] = g;

        ll x = a[i];

        while(cur + 1 < top && line_stack[cur + 1].s < x) cur++;

        dp[i] = line_stack[cur].m * a[i] + line_stack[cur].n;
    }

    return dp[n];
}
```

DP 아닌데 저런 꼴이 보여도 적용할 수 있음.

만약 a가 단조 증가하지 않으면 구간을 이분탐색으로 찾아주면 됨.

3 Segment Tree

3.1 Segment Tree with Lazy Propagation

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

using namespace std;
typedef long long ll;

class segment_tree {
    private:
        vector<ll> tree;
        vector<ll> lazy;

    public:
        segment_tree(int N) {
            tree.resize(N * 4);
            lazy.resize(N * 4);
        }

        void propagate(int start, int end, int node) {
            if(lazy[node] == 0) return;

            tree[node] += (ll)(end - start + 1) * lazy[node];

            if(start != end) {
                lazy[node * 2] += lazy[node];
                lazy[node * 2 + 1] += lazy[node];
            }

            lazy[node] = 0;
        }

        void update(int start, int end, int node, int left, int right, ll diff)
          {
            propagate(start, end, node);

            if(start > right || end < left) return;

            if(left <= start && end <= right) {
                tree[node] += (ll)(end - start + 1) * diff;

                if(start != end) {
                    lazy[node * 2] += diff;
                    lazy[node * 2 + 1] += diff;
                }

                return;
            }

            int mid = (start + end) / 2;

            update(start, mid, node * 2, left, right, diff);
            update(mid + 1, end, node * 2 + 1, left, right, diff);
            tree[node] = tree[node * 2] + tree[node * 2 + 1];
        }

        ll query(int start, int end, int node, int left, int right) {
            propagate(start, end, node);

            if(start > right || end < left) return 0;

            if(left <= start && end <= right) return tree[node];

            int mid = (start + end) / 2;
            return query(start, mid, node * 2, left, right) + query(mid + 1, end
              , node * 2 + 1, left, right);
        }
};
```

3.2 Dynamic Segment Tree

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

```
using namespace std;
using ll = long long;
using pii = pair<int, int>;
using ppii = pair<int, pii>;

struct node {
    node *l, *r;
    ll s;

    node() {
        l = r = NULL;
        s = 0;
    }
};

class dynamic_segment_tree {
    private:
        node* root;

        void delete_nodes(node* v) {
            if(!v) return;
            delete_nodes(v->l);
            delete_nodes(v->r);
            delete v;
        }

    public:
        dynamic_segment_tree() {
            root = new node();
        }

        node* get_root() {
            return root;
        }

        void update(int start, int end, node* cur_node, int idx, ll val) {
            if(idx < start || idx > end) return;

            if(start == end) {
                cur_node->s = val;
                return;
            }

            int mid = (start + end) / 2;
            if(idx <= mid) {
                if(!cur_node->l) cur_node->l = new node();
                update(start, mid, cur_node->l, idx, val);
            }
            else {
                if(!cur_node->r) cur_node->r = new node();
                update(mid + 1, end, cur_node->r, idx, val);
            }

            ll l_val = (cur_node->l ? cur_node->l->s : 0);
            ll r_val = (cur_node->r ? cur_node->r->s : 0);
            cur_node->s = l_val + r_val;
        }

        ll query(int start, int end, node* cur_node, int left, int right) {
            if(!cur_node) return 0;

            if(right < start || left > end) return 0;

            if(left <= start && end <= right) return cur_node->s;

            int mid = (start + end) / 2;

            return query(start, mid, cur_node->l, left, right) + query(mid + 1,
              end, cur_node->r, left, right);
        }

        ~dynamic_segment_tree() {
            delete_nodes(root);
        }
};
```

3.3 __gnu_pbds Ordered Set (can be replaced by k-th segtree)

```
#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;

//less <- set, less_equal <- multiset
#define ordered_set tree<int, null_type, less_equal<int>, rb_tree_tag,
  tree_order_statistics_node_update>

int main() {
    ordered_set pbds_set;

    int X; cin >> X;
    pbds_set.insert(X); //insert
    cout << pbds_set.order_of_key(X) << '\n'; //Number of elements smaller than
      X : O(log N)
    cout << *pbds_set.find_by_order(X) << '\n'; //X-th element in a set (0-based
      ) : O(log N)
    //based on red-black tree.

    return 0;
}
```

4 Graph

4.1 Bellman-Ford

```
#include <bits/stdc++.h>
using namespace std;
const long long INF = 1e18;

vector<long long> dist(501, INF);

void solve(vector<pair<int, int>> (&adj)[501], int& N) {
    dist[1] = 0;
    for(int i = 0; i < (N - 1); ++i) {
        for(int j = 1; j <= N; ++j) {
            for(auto edge : adj[j]) {
                if(dist[j] != INF) {
                    dist[edge.first] = min(dist[j] + edge.second, dist[edge.
                      first]);
                }
            }
        }
    }

    for(int i = 1; i <= N; ++i) {
        for(auto edge : adj[i]) {
            if(dist[i] != INF && dist[edge.first] > dist[i] + edge.second) {
                cout << "-1\n";
                return;
            }
        }
    }

    for(int i = 2; i <= N; ++i) {
        if(dist[i] != INF) {
            cout << dist[i] << '\n';
        }
        else cout << "-1\n";
    }
}
```

4.2 Heavy-Light Decomposition

```
#include <bits/stdc++.h>
#define MAX 100'000
#define INF 987654321

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using ppii = pair<int, pii>;

class segment_tree { /* segment tree implementation */ };

class heavy_light_decomposition {
private:
    int group_cnt;
    int tree_size[MAX], depth[MAX], parent[MAX], top_chain[MAX], in[MAX],
      out[MAX];
    bool visit[MAX];
    vector<int> child[MAX];
    vector<int> adj[MAX];
    segment_tree tree;

public:
    void init() {
        group_cnt = 0;

        int N; cin >> N;
        tree.resize(MAX);

        for(int i = 0; i < N - 1; ++i) {
            int u, v; cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }

        dfs_child_set();
        dfs_size();
        dfs_grouping();
    }

    void dfs_child_set(int v = 1) {
        visit[v] = true;
        for(auto& next : adj[v]) {
            if(visit[next]) continue;
            visit[next] = true;
            child[v].push_back(next);
            dfs_child_set(next);
        }
    }

    void dfs_size(int v = 1) {
        tree_size[v] = 1;
        for(auto& next : child[v]) {
            depth[next] = depth[v] + 1;
            parent[next] = v;

            dfs_size(next);
            tree_size[v] += tree_size[next];
            if(tree_size[next] > tree_size[child[v][0]]) swap(child[v][0],
              next);
        }
    }

    void dfs_grouping(int v = 1) {
        in[v] = ++group_cnt;
        for(auto& next : child[v]) {
            top_chain[next] = (next == child[v][0] ? top_chain[v] : next);
            dfs_grouping(next);
            }
            out[v] = group_cnt;
        }

        void update(int v, int w) {
            tree.update(1, MAX, 1, in[v], w);
        }

        int query(int a, int b) {
            int ret = 0;
            while(top_chain[a] != top_chain[b]) {
                if(depth[top_chain[a]] < depth[top_chain[b]]) swap(a, b);
                int v = top_chain[a];
                ret += tree.query(1, MAX, 1, in[v], in[a]);
                a = parent[v];
            }

            if(depth[a] > depth[b]) swap(a, b);
            ret += tree.query(1, MAX, 1, in[a], in[b]);

            return ret;
        }
};
```

5 Geometry

5.1 선분교차 3 (선분교차 여부 + 교점 좌표 출력)

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

using namespace std;
typedef long long ll;

const long double eps = 1e-7;

typedef struct _Point {
    long double x;
    long double y;
} Point;

Point getVector(Point& A, Point& B) {
    Point vec = {B.x - A.x, B.y - A.y};
    return vec;
}

int ccw(Point& v, Point& u) {
    ll val = v.x * u.y - v.y * u.x;
    if(val > 0) return 1;
    else if(val < 0) return -1;
    else return 0;
}

bool operator==(const Point& A, const Point& B) {
    return (abs(A.x - B.x) < eps && abs(A.y - B.y) < eps);
}

Point& operator/=(Point& A, const long double div) {
    A = {A.x / div, A.y / div};
    return A;
}

Point operator*(const Point& A, const long double mul) {
    Point X = A;
    X = {X.x * mul, X.y * mul};
    return X;
}


bool operator<=(const Point& A, const Point& B) {
    if(A.x < B.x) return true;
    else if(A.x == B.x && A.y <= B.y) return true;
    else return false;
}

int is_cross(Point& A, Point& B, Point& C, Point& D) {
    Point vecAB = getVector(A, B);
    Point vecCD = getVector(C, D);

    Point vecBC = getVector(B, C);
    Point vecBD = getVector(B, D);

    Point vecDA = getVector(D, A);
    Point vecDB = getVector(D, B);

    if(ccw(vecAB, vecBC) * ccw(vecAB, vecBD) == 0 && ccw(vecCD, vecDA) * ccw(
      vecCD, vecDB) == 0) {
        if(B <= A) swap(A, B);
        if(D <= C) swap(C, D);

        if(A <= D && C <= B) return 1;
        else return -1;
    }
    else if(ccw(vecAB, vecBC) * ccw(vecAB, vecBD) <= 0 && ccw(vecCD, vecDA) *
      ccw(vecCD, vecDB) <= 0) return 2;
    else return -1;
}

long double get_size(Point& vec) {
    return sqrt(vec.x * vec.x + vec.y * vec.y);
}

Point get_meet(Point& A, Point& B, Point& C, Point& D) {
    Point ret;
    long double a, b, c, d, e, f;

    a = A.y - B.y;
    b = B.x - A.x;
    c = A.x * a + A.y * b;
    d = C.y - D.y;
    e = D.x - C.x;
    f = C.x * d + C.y * e;

    long double dn = a * e - b * d;

    ret = {((e * c - b * f) / dn), ((a * f - c * d) / dn)};

    return ret;
}

int main() {
    Point A, B, C, D;
    cin >> A.x >> A.y >> B.x >> B.y;
    cin >> C.x >> C.y >> D.x >> D.y;

    cout << fixed;
    cout.precision(15);

    if(is_cross(A, B, C, D) == 1) {
        cout << "1\n";

        Point unit_dir_vec_AB = {B.x - A.x, B.y - A.y};
        long double sz_AB = get_size(unit_dir_vec_AB);
        unit_dir_vec_AB /= sz_AB;

        Point unit_dir_vec_CD = {D.x - C.x, D.y - C.y};
        long double sz_CD = get_size(unit_dir_vec_CD);
        unit_dir_vec_CD /= sz_CD;

        if(unit_dir_vec_AB == unit_dir_vec_CD || unit_dir_vec_AB ==
          unit_dir_vec_CD * (-1)) {
            if(B == C) cout << B.x << ' ' << B.y << '\n';
            else if(A == D) cout << A.x << ' ' << A.y << '\n';
        }
        else {
            if(A == C || A == D) cout << A.x << ' ' << A.y << '\n';
            else if(B == C || B == D) cout << B.x << ' ' << B.y << '\n';
        }
    }
    else if(is_cross(A, B, C, D) == 2) {
        cout << "1\n";

        Point X = get_meet(A, B, C, D);
        cout << X.x << ' ' << X.y << '\n';
    }
    else cout << "0\n";

    return 0;
}
```

5.2 Convex Hull

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

typedef struct _Point {
    int x;
    int y;
} Point;

//Standard Point to Sort
Point S;

Point get_vector(const Point& A, const Point& B) {
    Point v = {B.x - A.x, B.y - A.y};
    return v;
}

//ccw test
int ccw(const Point& v, const Point& u) {
    ll val = (ll)v.x * u.y - (ll)v.y * u.x;
    if(val > 0) return 1;
    else if(val < 0) return -1;
    else return 0;
}

int ccw(const Point& A, const Point& B, const Point& C) {
    Point v = get_vector(A, B);
    Point u = get_vector(B, C);
    return ccw(v, u);
}

//to sort by ccw
bool comp(const Point& A, const Point& B) {
    Point v = get_vector(S, A);
    Point u = get_vector(S, B);

    if(ccw(v, u) > 0) return true;
    else if(ccw(v, u) < 0) return false;

    return (v.x == u.x) ? (v.y < u.y) : (v.x < u.x);
}

bool operator<(const Point& A, const Point& B) {
    return (A.x == B.x) ? (A.y < B.y) : (A.x < B.x);
}

//Graham's Scan Method
vector<Point> get_convex_hull(vector<Point>& A) {
    S = *min_element(A.begin(), A.end());
    sort(A.begin(), A.end(), comp);
    int n = (int)A.size();

    vector<Point> convex_hull;

    //get Convex Hull
    for(int i = 0; i < n; ++i) {
        while((int)convex_hull.size() > 1
        && ccw(convex_hull[(int)convex_hull.size() - 2], convex_hull.back(), A[i
          ]) <= 0) {
            convex_hull.pop_back();
        }
        convex_hull.push_back(A[i]);
    }

    return convex_hull;
}
```

5.3 Rotating Calipers

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct Point { double x, y; };
int ccw(Point, Point);
Point get_vector(Point, Point);

ll get_dist(Point& A, Point& B) { return (B.x - A.x) * (B.x - A.x) + (B.y - A.y)
   * (B.y - A.y); }

ll max_dist(vector<Point>& convex_hull) {
    if(convex_hull.size() == 1) return 0;
    if(convex_hull.size() == 2) return get_dist(convex_hull[0], convex_hull[1]);

    ll ret = 0;
    int a_idx = 1, b_idx = 2;

    Point a_start = convex_hull[0], a_end = convex_hull[1], b_start =
      convex_hull[1], b_end = convex_hull[2];
    while(true) {
        ret = max(ret, get_dist(a_start, b_start));

        Point v1 = get_vector(a_start, a_end);
        Point v2 = get_vector(b_start, b_end);

        if(ccw(v1, v2) > 0) {
            b_idx = (b_idx + 1) % convex_hull.size();
            b_start = b_end;
            b_end = convex_hull[b_idx];
        }
        else {
            a_idx++;

            if(a_idx == convex_hull.size()) {
                a_start = a_end;
                a_end = convex_hull[0];
            }
            else if(a_idx == convex_hull.size() + 1) break; //end condition

            a_start = a_end;
            a_end = convex_hull[a_idx];
        }
    }

    return ret;
}
```

5.4 Point in Convex Hull Test

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

using namespace std;

typedef struct _Point {
    int x;
    int y;
} Point;

Point get_vector(const Point& A, const Point& B);
int ccw(const Point& v, const Point& u);
int ccw(const Point& A, const Point& B, const Point& C);

//convext_hull size >= 3
bool isInside(vector<Point>& convext_hull, Point& A) {
    int O = 0;
    int L = 1, R = (int)convext_hull.size() - 1;
    int M = (L + R) / 2;

    Point vecOL = get_vector(convext_hull[O], convext_hull[L]);
    Point vecOA = get_vector(convext_hull[O], A);
    Point vecOR = get_vector(convext_hull[O], convext_hull[R]);
    Point vecOM = get_vector(convext_hull[O], convext_hull[M]);

    if(ccw(vecOL, vecOA) < 0) return false;
    if(ccw(vecOR, vecOA) > 0) return false;

    while(L + 1 != R) {
        M = (L + R) / 2;
        vecOM = get_vector(convext_hull[O], convext_hull[M]);

        if(ccw(vecOM, vecOA) > 0) L = M;
        else R = M;
    }

    if(ccw(convext_hull[L], A, convext_hull[R]) <= 0) return true;
    else return false;
}
```