

Competitive Programming Notebook

Programadores Roblox

Contents

1 String	2	8.5 Dijkstra	10
1.1 Trie	2	8.6 Lca Jc	10
1.2 Countpermutations	2	8.7 Kruskal	11
1.3 Trie Ponteiros	2	8.8 Floyd Warshall	11
1.4 Z Function	2	9 Search and sort	11
1.5 Hashing	3	9.1 Mergeandcount	11
1.6 Kmp	3	9.2 Bfs	12
1.7 Lcs	3	9.3 Dfs	12
2 String copy	3	10 Math	12
2.1 Countpermutations	3	10.1 Equacao Diofantina	12
2.2 Trie Ponteiros	3	10.2 Crivo	12
2.3 Z Function	4	10.3 Fexp	12
2.4 Hashing	4	10.4 Exgcd	13
2.5 Kmp	4	10.5 Mod Inverse	13
2.6 Lcs	4		
3 DS	4		
3.1 Segtree Gcd	4		
3.2 Bit	5		
3.3 Psum 2d	5		
3.4 Ordered Set E Map	5		
3.5 Dsu	6		
3.6 Segtree Iterativa	6		
3.7 Segtree Sum	6		
4 Primitives	7		
5 Geometry	7		
5.1 Point Location	7		
5.2 Convex Hull	7		
5.3 Lattice Points	8		
5.4 Inside Polygon	8		
6 DP	9		
6.1 Lis	9		
6.2 Knapsack	9		
6.3 Lcs	9		
7 General	9		
7.1 Struct	9		
7.2 Bitwise	9		
8 Graph	9		
8.1 Bellman Ford	9		
8.2 Lca	9		
8.3 Dfs	10		
8.4 Topological Sort	10		

1 String

1.1 Trie

```

1 // Trie por array
2 // Inserir, busca e consulta de prefixo em O(N)
3
4 int trie[MAXN][26];
5 int tot_nos = 0;
6 vector<bool> acaba(MAXN, false);
7 vector<int> contador(MAXN, 0);
8
9 void insere(string s) {
10     int no = 0;
11     for(auto &c : s) {
12         if(trie[no][c - 'a'] == 0) {
13             trie[no][c - 'a'] = ++tot_nos;
14         }
15         no = trie[no][c - 'a'];
16         contador[no]++;
17     }
18     acaba[no] = true;
19 }
20
21 bool busca(string s) {
22     int no = 0;
23     for(auto &c : s) {
24         if(trie[no][c - 'a'] == 0) {
25             return false;
26         }
27         no = trie[no][c - 'a'];
28     }
29     return acaba[no];
30 }
31
32 int isPref(string s) {
33     int no = 0;
34     for(auto &c : s) {
35         if(trie[no][c - 'a'] == 0){
36             return -1;
37         }
38         no = trie[no][c - 'a'];
39     }
40     return contador[no];
41 }

```

1.2 Countpermutations

```

1 // Returns the number of distinct permutations
2 // that are lexicographically less than the string t
3 // using the provided frequency (freq) of the
4 // characters
5 // O(n*freq.size())
6 int countPermLess(vector<int> freq, const string &t)
7 {
8     int n = t.size();
9     int ans = 0;
10
11     vector<int> fact(n + 1, 1), invfact(n + 1, 1);
12     for (int i = 1; i <= n; i++)
13         fact[i] = (fact[i - 1] * i) % MOD;
14     invfact[n] = fexp(fact[n], MOD - 2, MOD);
15     for (int i = n - 1; i >= 0; i--)
16         invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
17
18     // For each position in t, try placing a letter
19     // smaller than t[i] that is in freq
20     for (int i = 0; i < n; i++) {
21         for (char c = 'a'; c < t[i]; c++) {
22             if (freq[c - 'a'] > 0) {
23                 freq[c - 'a']--;

```

```

21         int ways = fact[n - i - 1];
22         for (int f : freq)
23             ways = (ways * invfact[f]) % MOD;
24         ans = (ans + ways) % MOD;
25         freq[c - 'a']++;
26     }
27 }
28 if (freq[t[i] - 'a'] == 0) break;
29 freq[t[i] - 'a']--;
30 }
31 return ans;
32 }

```

1.3 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserir, busca e consulta de prefixo em O(N)
3
4 struct Node {
5     Node *filhos[26] = {};
6     bool acaba = false;
7     int contador = 0;
8 };
9
10 void insere(string s, Node *raiz) {
11     Node *cur = raiz;
12     for(auto &c : s) {
13         cur->contador++;
14         if(cur->filhos[c - 'a'] != NULL) {
15             cur = cur->filhos[c - 'a'];
16             continue;
17         }
18         cur->filhos[c - 'a'] = new Node();
19         cur = cur->filhos[c - 'a'];
20     }
21     cur->contador++;
22     cur->acaba = true;
23 }
24
25 bool busca(string s, Node *raiz) {
26     Node *cur = raiz;
27     for(auto &c : s) {
28         if (cur->filhos[c - 'a'] != NULL) {
29             cur = cur->filhos[c - 'a'];
30             continue;
31         }
32         return false;
33     }
34     return cur->acaba;
35 }
36
37 // Retorna se Ãl' prefixo e quantas strings tem s como
38 // prefixo
39 int isPref(string s, Node *raiz) {
40     Node *cur = raiz;
41     for(auto &c : s) {
42         if (cur->filhos[c - 'a'] != NULL) {
43             cur = cur->filhos[c - 'a'];
44             continue;
45         }
46         return -1;
47     }
48     return cur->contador;
49 }

```

1.4 Z Function

```

1 vector<int> z_function(string s) {
2     int n = s.size();
3     vector<int> z(n);
4     int l = 0, r = 0;
5     for(int i = 1; i < n; i++) {

```

```

6         if(i < r) {
7             z[i] = min(r - i, z[i - 1]);
8         }
9         while(i + z[i] < n && s[z[i]] == s[i + z[i]])
10        {
11            z[i]++;
12        }
13        if(i + z[i] > r) {
14            l = i;
15            r = i + z[i];
16        }
17        return z;
18    }

```

1.5 Hashing

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
4 // from left to right - O(1)
5 // query_inv(l, r) from right to left - O(1)
6 // patrocinado por tiagodfs
7 struct Hash {
8     const int X = 2147483647;
9     const int MOD = 1e9+7;
10    int n; string s;
11    vector<int> h, hi, p;
12    Hash() {}
13    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
14    (n) {
15        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
16        % MOD;
17        for (int i=0;i<n;i++)
18            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
19        for (int i=n-1;i>=0;i--)
20            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
21            % MOD;
22    }
23    int query(int l, int r) {
24        int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
25        0));
26        return hash < 0 ? hash + MOD : hash;
27    }
28    int query_inv(int l, int r) {
29        int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
30        +1] % MOD : 0));
31        return hash < 0 ? hash + MOD : hash;
32    }
33 };

```

1.6 Kmp

```

1 vector<int> kmp(string s) {
2     int n = (int)s.length();
3     vector<int> p(n+1);
4     p[0] = -1;
5     for (int i = 1; i < n; i++) {
6         int j = p[i-1];
7         while (j >= 0 && s[j] != s[i-1])
8             j = p[j-1];
9         p[i] = j+1;
10    }
11    return p;
12 }

```

1.7 Lcs

```

1 int lcs(string &s1, string &s2) {
2     int m = s1.size();
3     int n = s2.size();

```

```

4     vector<vector<int>> dp(m + 1, vector<int>(n + 1,
5     0));
6     for (int i = 1; i <= m; ++i) {
7         for (int j = 1; j <= n; ++j) {
8             if (s1[i - 1] == s2[j - 1])
9                 dp[i][j] = dp[i - 1][j - 1] + 1;
10            else
11                dp[i][j] = max(dp[i - 1][j], dp[i][j
12            - 1]);
13        }
14    }
15    return dp[m][n];
16 }

```

2 String copy

2.1 Countpermutations

```

1 // Returns the number of distinct permutations
2 // that are lexicographically less than the string t
3 // using the provided frequency (freq) of the
4 // characters
5 // O(n*freq.size())
6 int countPermLess(vector<int> freq, const string &t)
7 {
8     int n = t.size();
9     int ans = 0;
10
11    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
12    for (int i = 1; i <= n; i++)
13        fact[i] = (fact[i - 1] * i) % MOD;
14    invfact[n] = fexp(fact[n], MOD - 2, MOD);
15    for (int i = n - 1; i >= 0; i--)
16        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
17
18    // For each position in t, try placing a letter
19    // smaller than t[i] that is in freq
20    for (int i = 0; i < n; i++) {
21        for (char c = 'a'; c < t[i]; c++) {
22            if (freq[c - 'a'] > 0) {
23                freq[c - 'a']--;
24                int ways = fact[n - i - 1];
25                for (int f : freq)
26                    ways = (ways * invfact[f]) % MOD;
27                ans = (ans + ways) % MOD;
28                freq[c - 'a']++;
29            }
30        }
31        if (freq[t[i] - 'a'] == 0) break;
32        freq[t[i] - 'a']--;
33    }
34    return ans;
35 }

```

2.2 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserir e buscar prefixo em O(N)
3
4 struct Node {
5     Node *filhos[26] = {};
6     bool acaba = false;
7     int contador = 0;
8 };
9
10 void insere(string s, Node *raiz) {
11     Node *cur = raiz;

```

```

12     for(auto &c : s) {
13         cur->contador++;
14         if(cur->filhos[c - 'a'] != NULL) {
15             cur = cur->filhos[c - 'a'];
16             continue;
17         }
18         cur->filhos[c - 'a'] = new Node();
19         cur = cur->filhos[c - 'a'];
20     }
21     cur->contador++;
22     cur->acaba = true;
23 }
24
25 bool busca(string s, Node *raiz) {
26     Node *cur = raiz;
27     for(auto &c : s) {
28         if (cur->filhos[c - 'a'] != NULL) {
29             cur = cur->filhos[c - 'a'];
30             continue;
31         }
32         return false;
33     }
34     return cur->acaba;
35 }
36
37 // Retorna se Ãl prefixo e quantas strings tem s como
38     prefixo
39 int isPref(strings, Node *raiz) {
40     Node *cur = raiz;
41     for(auto &c : s) {
42         if (cur->filhos[c - 'a'] != NULL) {
43             cur = cur->filhos[c - 'a'];
44             continue;
45         }
46         return -1;
47     }
48     return cur->contador;
49 }

```

2.3 Z Function

```

1 vector<int> z_function(string s) {
2     int n = s.size();
3     vector<int> z(n);
4     int l = 0, r = 0;
5     for(int i = 1; i < n; i++) {
6         if(i < r) {
7             z[i] = min(r - i, z[i - l]);
8         }
9         while(i + z[i] < n && s[z[i]] == s[i + z[i]])
10             z[i]++;
11     }
12     if(i + z[i] > r) {
13         l = i;
14         r = i + z[i];
15     }
16 }
17 return z;
18 }

```

2.4 Hashing

```

1 // String Hash template
2 // constructor(s) - 0(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
4 // from left to right - 0(1)
5 // query_inv(l, r) from right to left - 0(1)
6 // patrocinado por tiagodfs
7
8 struct Hash {
9     const int X = 2147483647;

```

```

9     const int MOD = 1e9+7;
10     int n; string s;
11     vector<int> h, hi, p;
12     Hash() {}
13     Hash(string s): s(s), n(s.size()), h(n), hi(n), p
14         (n) {
15         for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
16             % MOD;
17         for (int i=0;i<n;i++)
18             h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
19         for (int i=n-1;i>=0;i--)
20             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
21                 % MOD;
22     }
23     int query(int l, int r) {
24         int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
25             0));
26         return hash < 0 ? hash + MOD : hash;
27     }
28     int query_inv(int l, int r) {
29         int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
30             +1] % MOD : 0));
31         return hash < 0 ? hash + MOD : hash;
32     }
33 }
34 };

```

2.5 Kmp

```

1 vector<int> kmp(string s) {
2     int n = (int)s.length();
3     vector<int> p(n+1);
4     p[0] = -1;
5     for (int i = 1; i < n; i++) {
6         int j = p[i-1];
7         while (j >= 0 && s[j] != s[i-1])
8             j = p[j-1];
9         p[i] = j+1;
10     }
11     return p;
12 }

```

2.6 Lcs

```

1 int lcs(string &s1, string &s2) {
2     int m = s1.size();
3     int n = s2.size();
4
5     vector<vector<int>> dp(m + 1, vector<int>(n + 1,
6         0));
7
8     for (int i = 1; i <= m; ++i) {
9         for (int j = 1; j <= n; ++j) {
10             if (s1[i - 1] == s2[j - 1])
11                 dp[i][j] = dp[i - 1][j - 1] + 1;
12             else
13                 dp[i][j] = max(dp[i - 1][j], dp[i][j
14                     - 1]);
15         }
16     }
17     return dp[m][n];
18 }

```

3 DS

3.1 Segtree Gcd

```

1 int gcd(int a, int b) {
2     if (b == 0)
3         return a;
4     return gcd(b, a % b);

```

```

5 }
6
7 class SegmentTreeGCD {
8 private:
9     vector<int> tree;
10    int n;
11
12    void build(const vector<int>& arr, int node, int
13    start, int end) {
14        if (start == end) {
15            tree[node] = arr[start];
16        } else {
17            int mid = (start + end) / 2;
18            build(arr, 2 * node + 1, start, mid);
19            build(arr, 2 * node + 2, mid + 1, end);
20            tree[node] = gcd(tree[2 * node + 1], tree
21            [2 * node + 2]);
22        }
23    }
24
25    void update(int node, int start, int end, int idx
26    , int value) {
27        if (start == end) {
28            tree[node] = value;
29        } else {
30            int mid = (start + end) / 2;
31            if (idx <= mid) {
32                update(2 * node + 1, start, mid, idx,
33                value);
34            } else {
35                update(2 * node + 2, mid + 1, end,
36                idx, value);
37            }
38            tree[node] = gcd(tree[2 * node + 1], tree
39            [2 * node + 2]);
40        }
41    }
42
43    int query(int node, int start, int end, int l,
44    int r) {
45        if (r < start || l > end) {
46            return 0;
47        }
48        if (l <= start && end <= r) {
49            return tree[node];
50        }
51        int mid = (start + end) / 2;
52        int left_gcd = query(2 * node + 1, start, mid
53        , l, r);
54        int right_gcd = query(2 * node + 2, mid + 1,
55        end, l, r);
56        return gcd(left_gcd, right_gcd);
57    }
58
59 public:
60    SegmentTreeGCD(const vector<int>& arr) {
61        n = arr.size();
62        tree.resize(4 * n);
63        build(arr, 0, 0, n - 1);
64    }
65
66    void update(int idx, int value) {
67        update(0, 0, n - 1, idx, value);
68    }
69
70    int query(int l, int r) {
71        return query(0, 0, n - 1, l, r);
72    }
73 };

```

3.2 Bit

```

1 class BIT {
2     vector<int> bit;
3     int n;

```

```

4     int sum(int idx) {
5         int result = 0;
6         while (idx > 0) {
7             result += bit[idx];
8             idx -= idx & -idx;
9         }
10        return result;
11    }
12
13 public:
14    BIT(int size) {
15        n = size;
16        bit.assign(n + 1, 0); // BIT indexada em 1
17    }
18
19    void update(int idx, int delta) {
20        while (idx <= n) {
21            bit[idx] += delta;
22            idx += idx & -idx;
23        }
24    }
25
26    int query(int idx) {
27        return sum(idx);
28    }
29
30    int range_query(int l, int r) {
31        return sum(r) - sum(l - 1);
32    }
33 };
34
35 BIT fenwick(n);
36 for(int i = 1; i <= n; i++) {
37     fenwick.update(i, arr[i]);
38 }

```

3.3 Psum 2d

```

1 // retangulo retorna a psum2d do intervalo inclusivo
2 vector<vector<int>> psum(n+1, vector<int>(m+1, 0));
3
4 for (int i=1; i<n+1; i++){
5     for (int j=1; j<m+1; j++){
6         cin >> psum[i][j];
7         psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum
8         [i-1][j-1];
9     }
10 }
11
12 // y1 eh variavel reservada
13 int retangulo(int x1, int yy1, int x2, int yy2){
14     x2 = min(x2, n), yy2 = min(yy2, m);
15     x1 = max(0LL, x1-1), yy1 = max(0LL, yy1-1);
16
17     return psum[x2][yy2]-psum[x1][yy2]-psum[x2][yy1]+
18     psum[x1][yy1];
19 }

```

3.4 Ordered Set E Map

```

1
2 #include<ext/pb_ds/assoc_container.hpp>
3 #include<ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5 using namespace std;
6
7 template<typename T> using ordered_multiset = tree<T,
8     null_type, less_equal<T>, rb_tree_tag,
9     tree_order_statistics_node_update>;
10
11 template <typename T> using o_set = tree<T, null_type
12     , less<T>, rb_tree_tag,
13     tree_order_statistics_node_update>;
14
15 template <typename T, typename R> using o_map = tree<
16     T, R, less<T>, rb_tree_tag,
17     tree_order_statistics_node_update>;

```

```

10
11 int main() {
12     int i, j, k, n, m;
13     o_set<int>st;
14     st.insert(1);
15     st.insert(2);
16     cout << *st.find_by_order(0) << endl; /// k-esimo
        elemento
17     cout << st.order_of_key(2) << endl; ///numero de
        elementos menores que k
18     o_map<int, int>mp;
19     mp.insert({1, 10});
20     mp.insert({2, 20});
21     cout << mp.find_by_order(0)->second << endl; /// k-
        esimo elemento
22     cout << mp.order_of_key(2) << endl; /// numero de
        elementos (chave) menores que k
23     return 0;
24 }

```

3.5 Dsu

```

1 struct DSU {
2     vector<int> par, rank, sz;
3     int c;
4     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
        1, 1), c(n) {
5         for (int i = 1; i <= n; ++i) par[i] = i;
6     }
7     int find(int i) {
8         return (par[i] == i ? i : (par[i] = find(par[
            i])));
9     }
10    bool same(int i, int j) {
11        return find(i) == find(j);
12    }
13    int get_size(int i) {
14        return sz[find(i)];
15    }
16    int count() {
17        return c; // quantos componentes conexos
18    }
19    int merge(int i, int j) {
20        if ((i = find(i)) == (j = find(j))) return
            -1;
21        else --c;
22        if (rank[i] > rank[j]) swap(i, j);
23        par[i] = j;
24        sz[j] += sz[i];
25        if (rank[i] == rank[j]) rank[j]++;
26        return j;
27    }
28 };

```

3.6 Segtree Iterativa

```

1 // Exemplo de uso:
2 // SegTree<int> st(vetor);
3 // range query e point update
4
5 template <typename T>
6 struct SegTree {
7     int n;
8     vector<T> tree;
9     T neutral_value = 0;
10    T combine(T a, T b) {
11        return a + b;
12    }
13
14    SegTree(const vector<T>& data) {
15        n = data.size();
16        tree.resize(2 * n, neutral_value);

```

```

17
18        for (int i = 0; i < n; i++)
19            tree[n + i] = data[i];
20
21        for (int i = n - 1; i > 0; --i)
22            tree[i] = combine(tree[i * 2], tree[i * 2
                + 1]);
23    }
24
25    T range_query(int l, int r) {
26        T result = neutral_value;
27
28        for (l += n, r += n + 1; l < r; l >= 1, r
            >= 1) {
29            if (l & 1) result = combine(result, tree[
                l++]);
30            if (r & 1) result = combine(result, tree
                [--r]);
31        }
32
33        return result;
34    }
35
36    void update(int pos, T new_val) {
37        tree[pos += n] = new_val;
38
39        for (pos >= 1; pos > 0; pos >= 1)
40            tree[pos] = combine(tree[2 * pos], tree[2
                * pos + 1]);
41    }
42 };

```

3.7 Segtree Sum

```

1 struct SegTree {
2     ll merge(ll a, ll b) { return a + b; }
3     const ll neutral = 0;
4     int n;
5     vector<ll> t, lazy;
6     vector<bool> replace;
7     inline int lc(int p) { return p * 2; }
8     inline int rc(int p) { return p * 2 + 1; }
9     void push(int p, int l, int r) {
10        if (replace[p]) {
11            t[p] = lazy[p] * (r - l + 1);
12            if (l != r) {
13                lazy[lc(p)] = lazy[p];
14                lazy[rc(p)] = lazy[p];
15                replace[lc(p)] = true;
16                replace[rc(p)] = true;
17            }
18        } else if (lazy[p] != 0) {
19            t[p] += lazy[p] * (r - l + 1);
20            if (l != r) {
21                lazy[lc(p)] += lazy[p];
22                lazy[rc(p)] += lazy[p];
23            }
24        }
25        replace[p] = false;
26        lazy[p] = 0;
27    }
28    void build(int p, int l, int r, const vector<ll>
        &v) {
29        if (l == r) {
30            t[p] = v[l];
31        } else {
32            int mid = (l + r) / 2;
33            build(lc(p), l, mid, v);
34            build(rc(p), mid + 1, r, v);
35            t[p] = merge(t[lc(p)], t[rc(p)]);
36        }
37    }
38    void build(int _n) {

```

```

39     n = _n;
40     t.assign(n * 4, neutral);
41     lazy.assign(n * 4, 0);
42     replace.assign(n * 4, false);
43 }
44 void build(const vector<ll> &v) {
45     n = (int)v.size();
46     t.assign(n * 4, neutral);
47     lazy.assign(n * 4, 0);
48     replace.assign(n * 4, false);
49     build(1, 0, n - 1, v);
50 }
51 void build(ll *bg, ll *en) {
52     build(vector<ll>(bg, en));
53 }
54 ll query(int p, int l, int r, int L, int R) {
55     push(p, l, r);
56     if (l > R || r < L) return neutral;
57     if (l >= L && r <= R) return t[p];
58     int mid = (l + r) / 2;
59     auto ql = query(lc(p), l, mid, L, R);
60     auto qr = query(rc(p), mid + 1, r, L, R);
61     return merge(ql, qr);
62 }
63 ll query(int l, int r) { return query(1, 0, n -
64 1, l, r); }
65 void update(int p, int l, int r, int L, int R, ll
66 val, bool repl = 0) {
67     push(p, l, r);
68     if (l > R || r < L) return;
69     if (l >= L && r <= R) {
70         lazy[p] = val;
71         replace[p] = repl;
72         push(p, l, r);
73     } else {
74         int mid = (l + r) / 2;
75         update(lc(p), l, mid, L, R, val, repl);
76         update(rc(p), mid + 1, r, L, R, val, repl);
77     }
78     t[p] = merge(t[lc(p)], t[rc(p)]);
79 }
80 void sumUpdate(int l, int r, ll val) { update(1,
81 0, n - 1, l, r, val, 0); }
82 void assignUpdate(int l, int r, ll val) { update
83 (1, 0, n - 1, l, r, val, 1); }
84 } segsum;

```

4 Primitives

5 Geometry

5.1 Point Location

```

1
2 int32_t main(){
3     sws;
4
5     int t; cin >> t;
6
7     while(t--){
8
9         int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1
10         >> x2 >> y2 >> x3 >> y3;
11
12         int deltax1 = (x1-x2), deltax1 = (y1-y2);
13
14         int compx = (x1-x3), compy = (y1-y3);
15
16         int ans = (deltax1*compy) - (compx*deltax1);

```

```

17         if(ans == 0){cout << "TOUCH\n"; continue;}
18         if(ans < 0){cout << "RIGHT\n"; continue;}
19         if(ans > 0){cout << "LEFT\n"; continue;}
20     }
21     return 0;
22 }

```

5.2 Convex Hull

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {
9     cod x,y;
10     point(cod x = 0, cod y = 0): x(x), y(y)
11     {}
12
13     double modulo()
14     {
15         return sqrt(x*x + y*y);
16     }
17
18     point operator+(point o)
19     {
20         return point(x+o.x, y+o.y);
21     }
22     point operator-(point o)
23     {
24         return point(x - o.x, y - o.y);
25     }
26     point operator*(cod t)
27     {
28         return point(x*t, y*t);
29     }
30     point operator/(cod t)
31     {
32         return point(x/t, y/t);
33     }
34
35     cod operator*(point o)
36     {
37         return x*o.x + y*o.y;
38     }
39     cod operator^(point o)
40     {
41         return x*o.y - y * o.x;
42     }
43     bool operator<(point o)
44     {
45         if( x != o.x) return x < o.x;
46         return y < o.y;
47     }
48 };
49
50 int ccw(point p1, point p2, point p3)
51 {
52     cod cross = (p2-p1) ^ (p3-p1);
53     if(cross == 0) return 0;
54     else if(cross < 0) return -1;
55     else return 1;
56 }
57
58 vector <point> convex_hull(vector<point> p)
59 {
60     sort(p.begin(), p.end());
61     vector<point> L,U;
62
63     //Lower
64

```

```

65     for(auto pp : p)
66     {
67         while(L.size() >= 2 and ccw(L[L.size() - 2],
68             L.back(), pp) == -1)
69             {
70                 // Ãl -1 pq eu nÃo quero excluir os
71                 colineares
72                 L.pop_back();
73             }
74             L.push_back(pp);
75     }
76     reverse(p.begin(), p.end());
77     //Upper
78     for(auto pp : p)
79     {
80         while(U.size() >= 2 and ccw(U[U.size()-2], U
81             .back(), pp) == -1)
82             {
83                 U.pop_back();
84             }
85             U.push_back(pp);
86     }
87     L.pop_back();
88     L.insert(L.end(), U.begin(), U.end()-1);
89     return L;
90 }
91
92 cod area(vector<point> v)
93 {
94     int ans = 0;
95     int aux = (int)v.size();
96     for(int i = 2; i < aux; i++)
97     {
98         ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
99     }
100     ans = abs(ans);
101     return ans;
102 }
103
104 int bound(point p1 , point p2)
105 {
106     return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
107 }
108 //teorema de pick [pontos = A - (bound+points)/2 + 1]
109
110 int32_t main()
111 {
112     int n;
113     cin >> n;
114
115     vector<point> v(n);
116     for(int i = 0; i < n; i++)
117     {
118         cin >> v[i].x >> v[i].y;
119     }
120
121     vector <point> ch = convex_hull(v);
122
123     cout << ch.size() << '\n';
124     for(auto p : ch) cout << p.x << " " << p.y << "\n
125     ";
126
127     return 0;
128 }

```

5.3 Lattice Points

```

1 ll gcd(ll a, ll b) {
2     return b == 0 ? a : gcd(b, a % b);

```

```

3 }
4 ll area_triangulo(ll x1, ll y1, ll x2, ll y2, ll x3,
5     ll y3) {
6     return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 *
7         (y1 - y2));
8 }
9 ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
10     return gcd(abs(x2 - x1), abs(y2 - y1));
11 }
12
13 int32_t main() {
14     ll x1, y1, x2, y2, x3, y3;
15     cin >> x1 >> y1;
16     cin >> x2 >> y2;
17     cin >> x3 >> y3;
18     ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
19     ll tot_borda = pontos_borda(x1, y1, x2, y2) +
20         pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,
21             y3, x1, y1);
22
23     ll ans = (area - tot_borda) / 2 + 1;
24     cout << ans << endl;
25
26     return 0;
27 }

```

5.4 Inside Polygon

```

1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4     int x = ccw(a, b, e);
5     int y = ccw(b, c, e);
6     int z = ccw(c, a, e);
7     return !((x==1 or y==1 or z==1) and (x==-1 or y
8         ==-1 or z==-1));
9 }
10
11 bool inside(vp &p, point e){ // ccw
12     int l=2, r=(int)p.size()-1;
13     while(l<r){
14         int mid = (l+r)/2;
15         if(ccw(p[0], p[mid], e) == 1)
16             l=mid+1;
17         else{
18             r=mid;
19         }
20     }
21     // bordo
22     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
23     // ==0) return false;
24     // if(r==2 and ccw(p[0], p[1], e)==0) return
25     // false;
26     // if(ccw(p[r], p[r-1], e)==0) return false;
27     return insideT(p[0], p[r-1], p[r], e);
28 }
29
30 // Any O(n)
31
32 int inside(vp &p, point pp){
33     // 1 - inside / 0 - boundary / -1 - outside
34     int n = p.size();
35     for(int i=0;i<n;i++){
36         int j = (i+1)%n;
37         if(line({p[i], p[j]}).inside_seg(pp))
38             return 0;
39     }
40
41     int inter = 0;
42     for(int i=0;i<n;i++){
43         int j = (i+1)%n;
44         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
45             [i], p[j], pp)==1)

```



```

42         inter++; // up
43         else if(p[j].x <= pp.x and pp.x < p[i].x and
44             ccw(p[i], p[j], pp) == -1)
45             inter++; // down
46     }
47     if(inter%2==0) return -1; // outside
48     else return 1; // inside
49 }

```

6 DP

6.1 Lis

6.2 Knapsack

```

1 // dp[i][j] => i-esimo item com j-carga sobrando na
  mochila
2 // O(N * W)
3
4 for(int j = 0; j < MAXN; j++) {
5     dp[0][j] = 0;
6 }
7 for(int i = 1; i <= N; i++) {
8     for(int j = 0; j <= W; j++) {
9         if(items[i].first > j) {
10             dp[i][j] = dp[i-1][j];
11         }
12         else {
13             dp[i][j] = max(dp[i-1][j], dp[i-1][j-
14                 items[i].first] + items[i].second);
15         }
16 }

```

6.3 Lcs

7 General

7.1 Struct

```

1 struct Pessoa{
2     // Atributos
3     string nome;
4     int idade;
5
6     // Comparador
7     bool operator<(const Pessoa& other) const{
8         if(idade != other.idade) return idade > other
9             .idade;
10        else return nome > other.nome;
11    }

```

7.2 Bitwise

```

1 int check_kth_bit(int x, int k) {
2     return (x >> k) & 1;
3 }
4
5 void print_on_bits(int x) {
6     for (int k = 0; k < 32; k++) {
7         if (check_kth_bit(x, k)) {
8             cout << k << ' ';
9         }
10    }
11    cout << '\n';

```

```

12 }
13
14 int count_on_bits(int x) {
15     int ans = 0;
16     for (int k = 0; k < 32; k++) {
17         if (check_kth_bit(x, k)) {
18             ans++;
19         }
20     }
21     return ans;
22 }
23
24 bool is_even(int x) {
25     return ((x & 1) == 0);
26 }
27
28 int set_kth_bit(int x, int k) {
29     return x | (1 << k);
30 }
31
32 int unset_kth_bit(int x, int k) {
33     return x & ~(1 << k);
34 }
35
36 int toggle_kth_bit(int x, int k) {
37     return x ^ (1 << k);
38 }
39
40 bool check_power_of_2(int x) {
41     return count_on_bits(x) == 1;
42 }

```

8 Graph

8.1 Bellman Ford

```

1 struct Edge {
2     int u, v, w;
3 };
4
5 // se x = -1, não tem ciclo
6 // se x != -1, pegar pais de x pra formar o ciclo
7
8 int n, m;
9 vector<Edge> edges;
10 vector<int> dist(n);
11 vector<int> pai(n, -1);
12
13 for (int i = 0; i < n; i++) {
14     x = -1;
15     for (Edge &e : edges) {
16         if (dist[e.u] + e.w < dist[e.v]) {
17             dist[e.v] = max(-INF, dist[e.u] + e.w
18                 );
19             pai[e.v] = e.u;
20             x = e.v;
21         }
22     }
23 }
24
25 // achando caminho (se precisar)
26 for (int i = 0; i < n; i++) x = pai[x];
27
28 vector<int> ciclo;
29 for (int v = x; v = pai[v]) {
30     cycle.push_back(v);
31     if (v == x && ciclo.size() > 1) break;
32 }
33 reverse(ciclo.begin(), ciclo.end());

```

8.2 Lca

```

1 // LCA - CP algorithm
2 // preprocessing O(NlogN)
3 // lca O(logN)
4 // Uso: criar LCA com a quantidade de vÃrtices (n) e
   lista de adjacÃncia (adj)
5 // chamar a funÃÃo preprocess com a raiz da Ãrvore
6
7 struct LCA {
8     int n, l, timer;
9     vector<vector<int>> adj;
10    vector<int> tin, tout;
11    vector<vector<int>> up;
12
13    LCA(int n, const vector<vector<int>>& adj) : n(n)
14    , adj(adj) {}
15
16    void dfs(int v, int p) {
17        tin[v] = ++timer;
18        up[v][0] = p;
19        for (int i = 1; i <= l; ++i)
20            up[v][i] = up[up[v][i-1]][i-1];
21
22        for (int u : adj[v]) {
23            if (u != p)
24                dfs(u, v);
25        }
26
27        tout[v] = ++timer;
28    }
29
30    bool is_ancestor(int u, int v) {
31        return tin[u] <= tin[v] && tout[u] >= tout[v];
32    }
33
34    int lca(int u, int v) {
35        if (is_ancestor(u, v))
36            return u;
37        if (is_ancestor(v, u))
38            return v;
39        for (int i = l; i >= 0; --i) {
40            if (!is_ancestor(up[u][i], v))
41                u = up[u][i];
42        }
43        return up[u][0];
44    }
45
46    void preprocess(int root) {
47        tin.resize(n);
48        tout.resize(n);
49        timer = 0;
50        l = ceil(log2(n));
51        up.assign(n, vector<int>(l + 1));
52        dfs(root, root);
53    }

```

8.3 Dfs

```

1 int dfs(int x, int p) {
2     for (auto e : adj[x]) {
3         if (e != p) {
4             dfs(e, x);
5         }
6     }
7 }

```

8.4 Topological Sort

```

1 vector<int> adj[MAXN];
2 vector<int> estado(MAXN); // 0: nao visitado 1:
   processamento 2: processado

```

```

3 vector<int> ordem;
4 bool temCiclo = false;
5
6 void dfs(int v) {
7     if(estado[v] == 1) {
8         temCiclo = true;
9         return;
10    }
11    if(estado[v] == 2) return;
12    estado[v] = 1;
13    for(auto &nei : adj[v]) {
14        if(estado[v] != 2) dfs(nei);
15    }
16    estado[v] = 2;
17    ordem.push_back(v);
18    return;

```

8.5 Dijkstra

```

1 // SSP com pesos positivos.
2 // O((V + E) log V).
3
4 vector<int> dijkstra(int S) {
5     vector<bool> vis(MAXN, 0);
6     vector<ll> dist(MAXN, LLONG_MAX);
7     dist[S] = 0;
8     priority_queue<pii, vector<pii>, greater<pii>> pq;
9
10    pq.push({0, S});
11    while(pq.size()) {
12        ll v = pq.top().second;
13        pq.pop();
14        if(vis[v]) continue;
15        vis[v] = 1;
16        for(auto &[peso, vizinho] : adj[v]) {
17            if(dist[vizinho] > dist[v] + peso) {
18                dist[vizinho] = dist[v] + peso;
19                pq.push({dist[vizinho], vizinho});
20            }
21        }
22        return dist;
23    }

```

8.6 Lca Jc

```

1 int LOG;
2
3 int get_lca(int a, int b) {
4     if(profundidade[b] > profundidade[a]) {
5         swap(a, b);
6     }
7     int k = profundidade[a] - profundidade[b]; //
   tanto que tenho que subir
8     for(int j = LOG-1; j >= 0; j--) {
9         if((1 << j) & k) {
10            a = cima[a][j];
11        }
12    }
13    if(a == b) return a; // ja to no lca
14
15    for(int j = LOG-1; j >= 0; j--) { // subo com os
   dois atÃ chegar no lca fazendo binary lifting
16        if(cima[a][j] != cima[b][j]) {
17            a = cima[a][j];
18            b = cima[b][j];
19        }
20    }
21    return cima[a][0];
22 }
23
24 void dfs(int v, int p) {

```

```

25     if(v != 1) profundidade[v] = profundidade[p] + 1; 50     return mst;
26     cima[v][0] = p; 51 }
27     for(int j = 1; j < LOG; j++) {
28         if (cima[v][j-1] != -1) {
29             cima[v][j] = cima[cima[v][j-1]][j-1];
30         } else {
31             cima[v][j] = -1;
32         }
33     }
34     for(auto &nei : adj[v]) {
35         if(nei != p) {
36             dfs(nei, v);
37         }
38     }
39 }
40
41 while((1 << LOG) <= n) LOG++;

```

8.7 Kruskal

```

1 // Ordena as arestas por peso, insere se ja nao
  // estiver no mesmo componente
2 // O(E log E)
3
4 struct DSU {
5     vector<int> par, rank, sz;
6     int c;
7     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
8         1, 1), c(n) {
9         for (int i = 1; i <= n; ++i) par[i] = i;
10    }
11    int find(int i) {
12        return (par[i] == i ? i : (par[i] = find(par[
13            i])));
14    }
15    bool same(int i, int j) {
16        return find(i) == find(j);
17    }
18    int get_size(int i) {
19        return sz[find(i)];
20    }
21    int count() {
22        return c; // quantos componentes conexos
23    }
24    int merge(int i, int j) {
25        if ((i = find(i)) == (j = find(j))) return
26        -1;
27        else --c;
28        if (rank[i] > rank[j]) swap(i, j);
29        par[i] = j;
30        sz[j] += sz[i];
31        if (rank[i] == rank[j]) rank[j]++;
32        return j;
33    }
34 };
35
36 struct Edge {
37     int u, v, w;
38     bool operator <(Edge const & other) {
39         return weight < other.weight;
40     }
41 }
42
43 vector<Edge> kruskal(int n, vector<Edge> edges) {
44     vector<Edge> mst;
45     DSU dsu = DSU(n + 1);
46     sort(edges.begin(), edges.end());
47     for (Edge e : edges) {
48         if (dsu.find(e.u) != dsu.find(e.v)) {
49             mst.push_back(e);
50             dsu.join(e.u, e.v);
51         }
52     }
53 }

```

8.8 Floyd Warshall

```

1 // SSP e acha ciclos.
2 // Bom com constraints menores.
3 // O(n^3)
4
5 int dist[501][501];
6
7 void floydWarshall() {
8     for(int k = 0; k < n; k++) {
9         for(int i = 0; i < n; i++) {
10             for(int j = 0; j < n; j++) {
11                 dist[i][j] = min(dist[i][j], dist[i][
12                     k] + dist[k][j]);
13             }
14         }
15     }
16 }
17
18 void solve() {
19     int m, q;
20     cin >> n >> m >> q;
21     for(int i = 0; i < n; i++) {
22         for(int j = i; j < n; j++) {
23             if(i == j) {
24                 dist[i][j] = dist[j][i] = 0;
25             } else {
26                 dist[i][j] = dist[j][i] = 1inf;
27             }
28         }
29     }
30     for(int i = 0; i < m; i++) {
31         int u, v, w;
32         cin >> u >> v >> w; u--; v--;
33         dist[u][v] = min(dist[u][v], w);
34         dist[v][u] = min(dist[v][u], w);
35     }
36     floydWarshall();
37     while(q--) {
38         int u, v;
39         cin >> u >> v; u--; v--;
40         if(dist[u][v] == 1inf) cout << -1 << '\n';
41         else cout << dist[u][v] << '\n';
42     }
43 }

```

9 Search and sort

9.1 Mergeandcount

```

1
2 // Realiza a mesclagem de dois subarrays e conta o
  // número de trocas necessárias.
3 int mergeAndCount(vector<int> & v, int l, int m, int r
4 ) {
5     int x = m - 1 + 1; // Tamanho do subarray
6     esquerdo.
7     int y = r - m; // Tamanho do subarray direito.
8
9     // Vetores temporarios para os subarray esquerdo
10    e direito.
11    vector<int> left(x), right(y);
12
13    for (int i = 0; i < x; i++) left[i] = v[l + i];
14    for (int j = 0; j < y; j++) right[j] = v[m + 1 +
15        j];
16
17    int i = 0, j = 0, k = 1;
18    int swaps = 0;

```

```

15 while (i < x && j < y) {
16     if (left[i] <= right[j]) {
17         // Se o elemento da esquerda for menor ou
18         igual, coloca no vetor original.
19         v[k++] = left[i++];
20     } else {
21         // Caso contrario, coloca o elemento da
22         direita e conta as trocas.
23         v[k++] = right[j++];
24         swaps += (x - i);
25     }
26 }
27 // Adiciona os elementos restantes do subarray
28 esquerdo (se houver).
29 while (i < x) v[k++] = left[i++];
30 // Adiciona os elementos restantes do subarray
31 direito (se houver).
32 while (j < y) v[k++] = right[j++];
33 return swaps; // Retorna o numero total de
34 trocas realizadas.
35 }
36 int mergeSort(vector<int>& v, int l, int r) {
37     int swaps = 0;
38
39     if (l < r) {
40         // Encontra o ponto medio para dividir o
41         vetor.
42         int m = l + (r - l) / 2;
43
44         // Chama merge sort para a metade esquerda.
45         swaps += mergeSort(v, l, m);
46         // Chama merge sort para a metade direita.
47         swaps += mergeSort(v, m + 1, r);
48
49         // Mescla as duas metades e conta as trocas.
50         swaps += mergeAndCount(v, l, m, r);
51     }
52
53     return swaps; // Retorna o numero total de
54 trocas no vetor.
55 }

```

9.2 Bfs

```

1 // Printa os nos na ordem em que sÃ£o visitados
2 // Explora em largura (camadas)
3 // Complexidade: O(V+A) V = vertices e A = arestas
4 // Espaco: O(V)
5 // Uso: busca pelo caminho mais curto
6
7 void bfs(vector<vector<int>>&grafo, int inicio){
8     set<int> visited;
9     queue<int> fila;
10
11     fila.push(inicio);
12     visited.insert(inicio);
13
14     while(!fila.empty()){
15         int cur = fila.front();
16         fila.pop();
17
18         cout << cur << " "; // printa o nÃo atual
19
20         for(int vizinho: grafo[cur]){
21             if(visited.find(vizinho) == visited.end())
22             ){
23                 fila.push(vizinho);
24                 visited.insert(vizinho)

```

```

24     }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

9.3 Dfs

```

1 // Printa os nos na ordem em que sÃ£o visitados
2 // Explora em profundidade
3 // Complexidade: O(V+A) V = vertices e A = arestas
4 // Espaco: O(V)
5 // Uso: explorar caminhos e backtracking
6
7 void dfs(vector<vector<int>>& grafo, int inicio){
8     set<int> visited;
9     stack<int> pilha;
10
11     pilha.push(inicio);
12
13     while(!pilha.empty()){
14         int cur = pilha.top();
15         pilha.pop();
16
17         if(visited.find(cur) == visited.end()){
18             cout << cur << " ";
19             visited.insert(cur);
20
21             for(int vizinho: grafo[cur]){
22                 if(visited.find(vizinho) == visited.
23                 end()){
24                     pilha.push(vizinho);
25                 }
26             }
27         }
28     }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

10 Math

10.1 Equacao Diofantina

```

1 // resolve equacao ax + by = c
2 // retorno {existe sol., x, y, g}
3 array<ll, 4> find_any_solution(ll a, ll b, ll c) {
4     auto[x, y, g] = exgcd(a, b);
5     if (c % g) return {false, 0, 0, 0};
6     x *= c / g;
7     y *= c / g;
8     return {true, x, y, g};
9 }

```

10.2 Crivo

```

1 // O(n*log(log(n)))
2 bool composto[MAX]
3 for(int i = 1; i <= n; i++) {
4     if(composto[i]) continue;
5     for(int j = 2*i; j <= n; j += i)
6         composto[j] = 1;
7 }

```

10.3 Fexp

```

1 // a^e mod m
2 // O(log n)
3
4 int fexp(int a, int e, int m) {
5     a %= m;
6     int ans = 1;
7     while (e > 0){
8         if (e & 1) ans = ans*a % m;
9         a = a*a % m;

```

```

10     e /= 2;
11 }
12 return ans%m;
13 }

```

10.4 Exgcd

```

1 // 0 retorno da funcao eh {n, m, g}
2 // e significa que gcd(a, b) = g e
3 // n e m sao inteiros tais que an + bm = g
4 array<ll, 3> exgcd(int a, int b) {
5     if(b == 0) return {1, 0, a};
6     auto [m, n, g] = exgcd(b, a % b);
7     return {n, m - a / b * n, g};
8 }

```

10.5 Mod Inverse

```

1 array<int, 2> extended_gcd(int a, int b) {
2     if (b == 0) return {1, 0};
3     auto [x, y] = extended_gcd(b, a % b);
4     return {y, x - (a / b) * y};
5 }
6
7 int mod_inverse(int a, int m) {
8     auto [x, y] = extended_gcd(a, m);
9     return (x % m + m) % m;
10 }

```