

Competitive Programming Notebook

Programadores Roblox

Contents

1 String	7 Graph	11
1.1 Trie Ponteiros	7.1 Dijkstra	11
1.2 Trie	7.2 Floyd Warshall	11
1.3 Hashing	7.3 Eulerian Path	11
1.4 Lcs	7.4 Dinitz	12
1.5 Countpermutations	7.5 Khan	12
1.6 Z Function	7.6 Topological Sort	13
1.7 Kmp	7.7 Acha Pontes	13
2 String copy	7.8 Edmonds-karp	13
2.1 Trie Ponteiros	7.9 Kruskal	13
2.2 Hashing	7.10 Bellman Ford	14
2.3 Lcs	7.11 Lca Jc	14
2.4 Countpermutations	7.12 Lca	15
2.5 Z Function	7.13 Kosaraju	15
2.6 Kmp	7.14 Pega Ciclo	15
3 DS	7.15 Min Cost Max Flow	16
3.1 Segtree Iterativa	8 Primitives	17
3.2 Segtree Gcd	9.1 Lis	17
3.3 Merge Sort Tree	9.2 Edit Distance	17
3.4 Ordered Set E Map	9.3 Bitmask	17
3.5 Sparse Table	9.4 Lcs	17
3.6 Psum 2d	9.5 Digit	17
3.7 Segtree Sum	9.6 Knapsack	18
3.8 Dsu	9.7 Lis Seg	18
3.9 Bit	9.8 Disjoint Blocks	18
4 Search and sort	9 DP	17
4.1 Pilha Monotonic	9.1 Lis	17
4.2 Mergeandcount	9.2 Edit Distance	17
5 Stress	9.3 Bitmask	17
5.1 Gen	9.4 Lcs	17
6 Math	9.5 Digit	17
6.1 Combinatorics	9.6 Knapsack	18
6.2 Equacao Diofantina	9.7 Lis Seg	18
6.3 Discrete Log	9.8 Disjoint Blocks	18
6.4 Segment Sieve	10 General	18
6.5 Totient	10.1 Brute Choose	18
6.6 Menor Fator Primo	10.2 Struct	18
6.7 Exgcd	10.3 Mex	18
6.8 Fexp	10.4 Bitwise	19
6.9 Divisores	11 Geometry	19
6.10 Crivo	11.1 Convex Hull	19
6.11 Mod Inverse	11.2 Inside Polygon	20
6.12 Base Calc	11.3 Point Location	20
	11.4 Lattice Points	20

1 String

1.1 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserção, 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 o prefixo é 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;
}

```

1.2 Trie

```

1 // Trie por array
2 // Inserção, 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 }

```

1.3 Hashing

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
// from left to right - O(1)
4 // query_inv(l, r) from right to left - O(1)
5 // patrocinado por tiagodfs
6
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(n) {
14        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1) %
MOD;
15        for (int i=0;i<n;i++)
16            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
17        for (int i=n-1;i>=0;i--)
18            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X) %
MOD;
19    }
20    int query(int l, int r) {
21        int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
0));
22        return hash < 0 ? hash + MOD : hash;
23    }
24    int query_inv(int l, int r) {
25        int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
26        return hash < 0 ? hash + MOD : hash;
27    }
28 };

```

1.4 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,
0));
6

```

```

7     for (int i = 1; i <= m; ++i) {
8         for (int j = 1; j <= n; ++j) {
9             if (s1[i - 1] == s2[j - 1])
10                dp[i][j] = dp[i - 1][j - 1] + 1;
11            else
12                dp[i][j] = max(dp[i - 1][j], dp[i][j
13 - 1]);
14        }
15    }
16    return dp[m][n];
17 }
```

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

1.6 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             z[i]++;
11         }
12         if(i + z[i] > r) {
13             l = i;
14             r = i + z[i];
15         }
16     }
17     return z;
18 }
```

1.7 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 String copy

2.1 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserção, 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 o 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 }
```

2.2 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
8 mt19937 rng(time(nullptr));
9
10 struct Hash {
11     const int X = rng();
12     const int MOD = 1e9+7;
13     int n; string s;
14     vector<int> h, hi, p;
15     Hash() {}
16     Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
17         for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1) % MOD;
18         for (int i=0;i<n;i++)
19             h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
20         for (int i=n-1;i>=0;i--)
21             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X) % MOD;
22     }
23     int query(int l, int r) {
24         int hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0) % MOD : 0));
25         return hash < 0 ? hash + MOD : hash;
26     }
27     int query_inv(int l, int r) {
28         int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1]:0)) % MOD : 0));
29         return hash < 0 ? hash + MOD : hash;
30     }
}

```

2.3 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, 0));
6
7     for (int i = 1; i <= m; ++i) {
8         for (int j = 1; j <= n; ++j) {
9             if (s1[i - 1] == s2[j - 1])
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11             else
12                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
13         }
14     }
15
16     return dp[m][n];
17 }

```

2.4 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 int countPermLess(vector<int> freq, const string &t)
{
6     int n = t.size();
7     int ans = 0;

```

```

8         vector<int> fact(n + 1, 1), invfact(n + 1, 1);
9
10        for (int i = 1; i <= n; i++)
11            fact[i] = (fact[i - 1] * i) % MOD;
12        invfact[n] = fexp(fact[n], MOD - 2, MOD);
13        for (int i = n - 1; i >= 0; i--)
14            invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
15
16        // For each position in t, try placing a letter
17        // smaller than t[i] that is in freq
18        for (int i = 0; i < n; i++) {
19            for (char c = 'a'; c < t[i]; c++) {
20                if (freq[c - 'a'] > 0) {
21                    freq[c - 'a']--;
22                    int ways = fact[n - i - 1];
23                    for (int f : freq)
24                        ways = (ways * invfact[f]) % MOD;
25                    ans = (ans + ways) % MOD;
26                    freq[c - 'a']++;
27                }
28                if (freq[t[i] - 'a'] == 0) break;
29                freq[t[i] - 'a']--;
30            }
31        }
32    }

```

2.5 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             z[i]++;
11         }
12         if(i + z[i] > r) {
13             l = i;
14             r = i + z[i];
15         }
16     }
17     return z;
18 }

```

2.6 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 }

```

3 DS

3.1 Segtree Iterativa

```

1 // Exemplo de uso:
2 // SegTree<int> st(vetor);

```

3.2 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 start, int end) {
13        if (start == end)
14            tree[node] = arr[start];
15        else {
16            int mid = (start + end) / 2;
17            build(arr, start, mid);
18            build(arr, mid + 1, end);
19            tree[node] = gcd(tree[2 * node], tree[2 * node + 1]);
20        }
21    }
22}
```

3.3 Merge Sort Tree

```

struct SegTree {
    int n;
    vector<vector<int>> tree;

    SegTree(vector<int> &a) {
        n = a.size();
        tree.resize(4 * n);
        build(1, 0, n - 1, a);
    }

    void build(int x, int lx, int rx, vector<int> &a)
    {
        if (lx == rx) {
            tree[x] = { a[lx] };
            return;
        }
        int mid = lx + (rx - lx)/2;
        build(2 * x, lx, mid, a);
        build(2 * x + 1, mid + 1, rx, a);
        auto &L = tree[2 * x], &R = tree[2 * x + 1];
        tree[x].resize(L.size() + R.size());
        merge(L.begin(), L.end(), R.begin(), R.end(),
              tree[x].begin());
    }
};

```

```

22 }
23
24 int query(int x, int lx, int rx, int l, int r) {
25     if (lx >= l && rx <= r) {
26         auto &v = tree[x];
27         return v.end() - upper_bound(v.begin(), v
28             .end(), r);
29     }
30     if (rx < l || lx > r) {
31         return 0;
32     }
33     int mid = lx + (rx - lx)/2;
34     return query(2 * x, lx, mid, l, r) + query(2
35     * x + 1, mid + 1, rx, l, r);
36 }
37
38 int query(int l, int r) {
39     return query(1, 0, n - 1, l, r);
40 }
41 // Checar se o range é todo distinto
42 // Cada cara e sua próxima aparição a direita,
43 // conta quantos caras que a próxima aparição a
44 // direita ta dentro do range ainda
45 vector<int> nr(n);
46 map<int, int> mp;
47 for (int i = n - 1; i >= 0; i--) {
48     auto it = mp.find(a[i]);
49     nr[i] = it != mp.end() ? it->second : n;
50     mp[a[i]] = i;
51 }
52 SegTree seg(nr);

```

3.4 Ordered Set E Map

```

1 #include<ext/pb_ds/assoc_container.hpp>
2 #include<ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 using namespace std;
5
6 template<typename T> using ordered_multiset = tree<T,
7     null_type, less_equal<T>, rb_tree_tag,
8     tree_order_statistics_node_update>;
9 template <typename T> using o_set = tree<T, null_type
10 , less<T>, rb_tree_tag,
11     tree_order_statistics_node_update>;
12 template <typename T, typename R> using o_map = tree<
13     T, R, less<T>, rb_tree_tag,
14     tree_order_statistics_node_update>;
15
16 int main() {
17     int i, j, k, n, m;
18     o_set<int> st;
19     st.insert(1);
20     st.insert(2);
21     cout << *st.find_by_order(0) << endl; // k-esimo
22     elemento
23     cout << st.order_of_key(2) << endl; // numero de
24     elementos menores que k
25     o_map<int, int> mp;
26     mp.insert({1, 10});
27     mp.insert({2, 20});
28     cout << mp.find_by_order(0)->second << endl; // k-
29     esimo elemento
30     cout << mp.order_of_key(2) << endl; // numero de
31     elementos (chave) menores que k
32     return 0;
33 }

```

3.5 Sparse Table

```

1 // 1-index, 0(1)
2 struct SparseTable {
3     vector<vector<int>> st;
4     int max_log;
5     SparseTable(vector<int>& arr) {
6         int n = arr.size();
7         max_log = floor(log2(n)) + 1;
8         st.resize(n, vector<int>(max_log));
9         for (int i = 0; i < n; i++) {
10             st[i][0] = arr[i];
11         }
12         for (int j = 1; j < max_log; j++) {
13             for (int i = 0; i + (1 << j) <= n; i++) {
14                 st[i][j] = max(st[i][j - 1], st[i +
15                     (1 << (j - 1))][j - 1]);
16             }
17         }
18     }
19     int query(int L, int R) {
20         int tamanho = R - L + 1;
21         int k = floor(log2(tamanho));
22         return max(st[L][k], st[R - (1 << k) + 1][k]);
23     }

```

3.6 Psum 2d

```

1 vector<vector<int>> psum(h+1, vector<int>(w+1, 0));
2
3 for (int i=1; i<=h; i++){
4     for (int j=1; j<=w; j++){
5         cin >> psum[i][j];
6         psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[
7             i-1][j-1];
8     }
9 }
10 // retorna a psum2d do intervalo inclusivo [(a, b), (
11 c, d)]
12 int retangulo(int a, int b, int c, int d){
13     c = min(c, h), d = min(d, w);
14     a = max(0LL, a-1), b = max(0LL, b-1);
15
16     return v[c][d]-v[a][d]-v[c][b]+v[a][b];
17 }

```

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     }

```

```

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 - 1, l, r); }
64 void update(int p, int l, int r, int L, int R, ll val, bool repl = 0) {
65     push(p, l, r);
66     if (l > R || r < L) return;
67     if (l >= L && r <= R) {
68         lazy[p] = val;
69         replace[p] = repl;
70         push(p, l, r);
71     } else {
72         int mid = (l + r) / 2;
73         update(lc(p), l, mid, L, R, val, repl);
74         update(rc(p), mid + 1, r, L, R, val, repl);
75     }
76     t[p] = merge(t[lc(p)], t[rc(p)]);
77 }
78 void sumUpdate(int l, int r, ll val) { update(1, 0, n - 1, l, r, val, 0); }
79 void assignUpdate(int l, int r, ll val) { update(1, 0, n - 1, l, r, val, 1); }
80 } segsum;

```

3.8 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 }

```

3.9 Bit

```

1 struct BIT {
2     int n;
3     vector<int> bit;
4     BIT(int n = 0) : n(n), bit(n + 1, 0) {}
5     void add(int i, int delta) {
6         for(; i <= n; i += i & -i) bit[i] += delta;
7     }
8     int sum(int i) {
9         int r = 0;
10        for(; i > 0; i -= i & -i) r += bit[i];
11        return r;
12    }
13    int range_sum(int l, int r) {
14        if (r < l) return 0;
15        return sum(r) - sum(l - 1);
16    }
17 }

```

4 Search and sort

4.1 Pilha Monotonic

```

1 vector<int> find_esq(vector<int> &v, bool maior) {
2     int n = v.size();
3     vector<int> result(n);
4     stack<int> s;
5
6     for (int i = 0; i < n; i++) {
7         while (!s.empty() && (maior ? v[s.top()] <= v[i] : v[s.top()] >= v[i])) {
8             s.pop();
9         }
10        if (s.empty()) {
11            result[i] = -1;
12        } else {
13            result[i] = v[s.top()];
14        }
15        s.push(i);
16    }
17    return result;
18 }
19
20 // maior = true -> encontra o primeiro maior à direita

```

```

21 vector<int> find_dir(vector<int> &v, bool maior) { 42
22     int n = v.size(); 43
23     vector<int> result(n); 44
24     stack<int> s; 45
25     for (int i = n - 1; i >= 0; i--) { 46
26         while (!s.empty() && (maior ? v[s.top()] <= v[47
27             [i] : v[s.top()] >= v[i]))) { 48
28             s.pop(); 49
29         } 50
30         if (s.empty()) { 51
31             result[i] = -1; 52
32         } else { 53
33             result[i] = v[s.top()]; 54
34         } 55
35         s.push(i); 56
36     } 57
37 } 58

```

4.2 Mergeandcount

```

1 // Realiza a mesclagem de dois subarrays e conta o
2 // nÃºmero de trocas necessÃ¡rias.
3 int mergeAndCount(vector<int>& v, int l, int m, int r)
4 {
5     int x = m - l + 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 = l;
18    int swaps = 0;
19
20    while (i < x && j < y) {
21        if (left[i] <= right[j]) {
22            // Se o elemento da esquerda for menor ou
23            igual, coloca no vetor original.
24            v[k++] = left[i++];
25        } else {
26            // Caso contrario, coloca o elemento da
27            direita e conta as trocas.
28            v[k++] = right[j++];
29            swaps += (x - i);
30        }
31
32        // Adiciona os elementos restantes do subarray
33        // esquerdo (se houver).
34        while (i < x) v[k++] = left[i++];
35
36        // Adiciona os elementos restantes do subarray
37        // direito (se houver).
38        while (j < y) v[k++] = right[j++];
39
40    return swaps; // Retorna o numero total de
41    // trocas realizadas.
42 }
43
44 int mergeSort(vector<int>& v, int l, int r) {
45     int swaps = 0;
46
47     if (l < r) {
48         // Encontra o ponto medio para dividir o
49         // vetor.
50         int m = l + (r - 1) / 2;
51
52         // Chama merge sort para a metade esquerda.
53         swaps += mergeSort(v, l, m);
54         // Chama merge sort para a metade direita.
55         swaps += mergeSort(v, m + 1, r);
56
57         // Mescla as duas metades e conta as trocas.
58         swaps += mergeAndCount(v, l, m, r);
59     }
60
61     return swaps; // Retorna o numero total de
62     // trocas no vetor.
63 }

```

5 Stress

5.1 Gen

```

1 #include <bits/stdc++.h>
2 #include <cstdlib>
3 #include <ctime>
4 using namespace std;
5
6 int randi(int L, int R) { return L + rand() % (R - L
7 + 1); }
8 char randc(char L, char R) { return char(L + rand() %
(R - L + 1)); }
9
10 int main(int argc, char** argv) {
11     if (argc > 1) srand(atoi(argv[1]));
12     else srand(time(0));
13
14     int n = randi(1, 100);
15     cout << n << '\n';
16     for (int i = 0; i < n; i++) {
17         cout << randi(-10, 20) << ' ';
18     }
19     cout << '\n';

```

6 Math

6.1 Combinatorics

```

1 const int MAXN_FATORIAL = 200005;
2 const int MOD = 1e9 + 7;
3 // DEFINE INT LONG LONG PLMDS
4 int fat[MAXN_FATORIAL], fati[MAXN_FATORIAL];
5
6 // (a^b) % m em O(log b)
7 // coloque o fexp
8
9 int inv(int n) { return fexp(n, MOD - 2); }
10
11 void precalc() {
12     fat[0] = 1;
13     fati[0] = 1;
14     for (int i = 1; i < MAXN_FATORIAL; i++) fat[i] =
(fat[i - 1] * i) % MOD;
15     fati[MAXN_FATORIAL - 1] = inv(fat[MAXN_FATORIAL -
1]);
16     for (int i = MAXN_FATORIAL - 2; i >= 0; i--) fati
[i] = (fati[i + 1] * (i + 1)) % MOD;
17 }
18
19 int choose(int n, int k) {
20     if (k < 0 || k > n) return 0;
21     return (((fat[n] * fati[k]) % MOD) * fati[n - k])
% MOD;
22 }

```

```

23
24 // n! / (n-k)!
25 int perm(int n, int k) {
26     if (k < 0 || k > n) return 0;
27     return (fat[n] * fati[n - k]) % MOD;
28 }
29
30 // C_n = (1 / (n+1)) * C(2n, n)
31 int catalan(int n) {
32     if (n < 0 || 2 * n >= MAXN_FATORIAL) return 0;
33     int c2n_n = choose(2 * n, n);
34     return (c2n_n * inv(n + 1)) % MOD;
35 }

```

6.2 Equacao Diofantina

```

1 int extended_gcd(int a, int b, int& x, int& y) {
2     if (a == 0) {
3         x = 0;
4         y = 1;
5         return b;
6     }
7     int x1, y1;
8     int gcd = extended_gcd(b % a, a, x1, y1);
9     x = y1 - (b / a) * x1;
10    y = x1;
11    return gcd;
12 }
13
14 bool solve(int a, int b, int c, int& x0, int& y0) {
15     int x, y;
16     int g = extended_gcd(abs(a), abs(b), x, y);
17     if (c % g != 0) {
18         return false;
19     }
20     x0 = x * (c / g);
21     y0 = y * (c / g);
22     if (a < 0) x0 = -x0;
23     if (b < 0) y0 = -y0;
24     return true;
25 }

```

6.3 Discrete Log

```

1 // Returns minimum x for which a^x = b (mod m), a and
2 // m are coprime.
3 // if the answer dont need to be greater than some
4 // value, the vector<int> can be removed
5 int discrete_log(int a, int b, int m) {
6     a %= m, b %= m;
7     int n = sqrt(m) + 1;
8
8     int an = 1;
9     for (int i = 0; i < n; ++i)
10        an = (an * 1ll * a) % m;
11
12     unordered_map<int, vector<int>> vals;
13     for (int q = 0, cur = b; q <= n; ++q) {
14         vals[cur].push_back(q);
15         cur = (cur * 1ll * a) % m;
16     }
17
18     int res = LLONG_MAX;
19
20     for (int p = 1, cur = 1; p <= n; ++p) {
21         cur = (cur * 1ll * an) % m;
22         if (vals.count(cur)) {
23             for (int q: vals[cur]){
24                 int ans = n * p - q;
25                 res = min(res, ans);
26             }
27         }
28     }

```

```

27     }
28     return res;
29 }

```

6.4 Segment Sieve

```

1 // Retorna quantos primos tem entre [l, r] (inclusivo
2 // )
3 // precisa de um vetor com os primos ate sqrt(r)
4 int seg_sieve(int l, int r){
5     if (l > r) return 0;
6     vector<bool> is_prime(r - l + 1, true);
7     if (l == 1) is_prime[0] = false;
8
8     for (int p : primos){
9         if (p * p > r) break;
10        int start = max(p * p, (l + p - 1) / p * p);
11        for (int j = start; j <= r; j += p){
12            if (j >= l) {
13                is_prime[j - l] = false;
14            }
15        }
16    }
17
18    return accumulate(all(is_prime), 0ll);;
19 }

```

6.5 Totient

```

1 // phi(n) = n * (1 - 1/p1) * (1 - 1/p2) * ...
2 int phi(int n) {
3     int result = n;
4     for (int i = 2; i * i <= n; i++) {
5         if (n % i == 0) {
6             while (n % i == 0)
7                 n /= i;
8             result -= result / i;
9         }
10    }
11    if (n > 1) // SE n sobrou, ele tem um fator primo
12        result -= result / n;
13    return result;
14 }
15
16 // crivo phi
17 const int MAXN_PHI = 1000001;
18 int phiv[MAXN_PHI];
19 void phi_sieve() {
20     for (int i = 0; i < MAXN_PHI; i++) phiv[i] = i;
21     for (int i = 2; i < MAXN_PHI; i++) {
22         if (phiv[i] == i) {
23             for (int j = i; j < MAXN_PHI; j += i)
24                 phiv[j] -= phiv[j] / i;
25         }
26     }

```

6.6 Menor Fator Primo

```

1 const int MAXN = 1000001; // Limite para o Crivo.
2 int spf[MAXN];
3 vector<int> primos;
4
5 void crivo() {
6     for (int i = 0; i < MAXN; i++) spf[i] = [i];
7     for (int i = 2; i * i < MAXN; i++) {
8         if (spf[i] == i) {
9             for (int j = i * i; j < MAXN; j += i) {
10                 if (spf[j] == j) {
11                     spf[j] = i;
12                 }
13             }
14         }
15     }
16 }

```

```

14     }
15 }
16 for (int i = 2; i < MAXN; i++) {
17     if (spf[i] == i) {
18         primos.push_back(i);
19     }
20 }
21 }
22 map<int, int> fatora(int n) {
23     map<int, int> fatores;
24     while (n > 1) {
25         fatores[spf[n]]++;
26         n /= spf[n];
27     }
28     return fatores;
29 }
30 }
31 int numero_de_divisores(int n) {
32     if (n == 1) return 1;
33     map<int, int> fatores = fatorar(n);
34     int nod = 1;
35     for (auto &[primo, expoente] : fatores) nod *= (expoente + 1);
36     return nod;
37 }
38 }
39
40 // DEFINE INT LONG LONG
41 int soma_dos_divisores(int n) {
42     if (n == 1) return 1;
43     map<int, int> fatores = fatorar(n);
44     int sod = 1;
45     for (auto &[primo, expoente] : fatores) {
46         int termo_soma = 1;
47         int potencia_primo = 1;
48         for (int i = 0; i < expoente; i++) {
49             potencia_primo *= primo;
50             termo_soma += potencia_primo;
51         }
52         sod *= termo_soma;
53     }
54     return sod;
55 }
56 }
```

6.7 Exgcd

```

1 // O 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 }
```

6.8 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 }
```

6.9 Divisores

```

1 // Retorna um vetor com os divisores de x
2 // eh preciso ter o crivo implementado
3 // O(divisores)
4
5 vector<int> divs(int x){
6     vector<int> ans = {1};
7     vector<array<int, 2>> primos; // {primo, expoente}
8
9     while (x > 1) {
10         int p = crivo[x], cnt = 0;
11         while (x % p == 0) cnt++, x /= p;
12         primos.push_back({p, cnt});
13     }
14
15     for (int i=0; i<primos.size(); i++){
16         int cur = 1, len = ans.size();
17
18         for (int j=0; j<primos[i][1]; j++){
19             cur *= primos[i][0];
20             for (int k=0; k<len; k++)
21                 ans.push_back(cur*ans[k]);
22         }
23     }
24
25     return ans;
26 }
```

6.10 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 }
```

6.11 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 }
```

6.12 Base Calc

```

1 int char_to_val(char c) {
2     if (c >= '0' && c <= '9') return c - '0';
3     else return c - 'A' + 10;
4 }
5
6 char val_to_char(int val) {
7     if (val >= 0 && val <= 9) return val + '0';
8     else return val - 10 + 'A';
9 }
10
11 int to_base_10(string &num, int bfrom) {
12     int result = 0;
13     int pot = 1;
14     for (int i = num.size() - 1; i >= 0; i--) {
15         if (char_to_val(num[i]) >= bfrom) return -1;
16         result += char_to_val(num[i]) * pot;
17         pot *= bfrom;
18 }
```

```

18     }
19     return result;
20 }
21
22 string from_base_10(int n, int bto) {
23     if (n == 0) return "0";
24     string result = "";
25     while (n > 0) {
26         result += val_to_char(n % bto);
27         n /= bto;
28     }
29     reverse(result.begin(), result.end());
30     return result;
31 }
32
33 string convert_base(string &num, int bfrom, int bto) {
34     int n_base_10 = to_base_10(num, bfrom);
35     return from_base_10(n_base_10, bto);
36 }

```

7 Graph

7.1 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    }
23    return dist;
}

```

7.2 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    void solve() {
17        int m, q;
18        cin >> n >> m >> q;
}

```

```

19    for( int i = 0; i < n; i++) {
20        for( int j = i; j < n; j++) {
21            if(i == j) {
22                dist[i][j] = dist[j][i] = 0;
23            } else {
24                dist[i][j] = dist[j][i] = lINF;
25            }
26        }
27    }
28    for( int i = 0; i < m; i++) {
29        int u, v, w;
30        cin >> u >> v >> w; u--; v--;
31        dist[u][v] = min(dist[u][v], w);
32        dist[v][u] = min(dist[v][u], w);
33    }
34    floydWarshall();
35    while(q--) {
36        int u, v;
37        cin >> u >> v; u--; v--;
38        if(dist[u][v] == lINF) cout << -1 << '\n';
39        else cout << dist[u][v] << '\n';
40    }
41 }

```

7.3 Eulerian Path

```

1 /**
2  * Versão que assume: #define int long long
3  *
4  * Retorna um caminho/ciclo euleriano em um grafo (se
5  * existir).
6  * - g: lista de adjacência (vector<vector<int>>).
7  * - directed: true se o grafo for dirigido.
8  * - s: vértice inicial.
9  * - e: vértice final (opcional). Se informado,
10   tenta caminho de s até e.
11  * - O(Nlog(N))
12  * Retorna vetor com a sequência de vértices, ou
13   vazio se impossível.
14  */
15 vector<int> eulerian_path(const vector<vector<int>>&
16 g, bool directed, int s, int e = -1) {
17     int n = (int)g.size();
18     // Câmpia das adjacências em multiset para
19     // permitir remoção específica
20     vector<multiset<int>> h(n);
21     vector<int> in_degree(n, 0);
22     vector<int> result;
23     stack<int> st;
24     // Preencher h e indegrees
25     for (int u = 0; u < n; ++u) {
26         for (auto v : g[u]) {
27             ++in_degree[v];
28             h[u].emplace(v);
29         }
30     }
31     st.emplace(s);
32     if (e != -1) {
33         int out_s = (int)h[s].size();
34         int out_e = (int)h[e].size();
35         int diff_s = in_degree[s] - out_s;
36         int diff_e = in_degree[e] - out_e;
37         if ((diff_s * diff_e) != -1) return {};
38         // impossível
39     }
40     for (int u = 0; u < n; ++u) {
41         if (e != -1 && (u == s || u == e)) continue;
42         int out_u = (int)h[u].size();
43         if (in_degree[u] != out_u || (!directed && (
44             in_degree[u] & 1))) {
45             return {};
46         }
47     }
48 }

```

```

41     while (!st.empty()) {
42         int u = st.top();
43         if (h[u].empty()) {
44             result.emplace_back(u);
45             st.pop();
46         } else {
47             int v = *h[u].begin();
48             auto it = h[u].find(v);
49             if (it != h[u].end()) h[u].erase(it);
50             --in_degree[v];
51             if (!directed) {
52                 auto it2 = h[v].find(u);
53                 if (it2 != h[v].end()) h[v].erase(it2);
54             }
55             --in_degree[u];
56         }
57         st.emplace(v);
58     }
59     for (int u = 0; u < n; ++u) {
60         if (in_degree[u] != 0) return {};
61     }
62     reverse(result.begin(), result.end());
63     return result;
64 }

// Complexidade: O(V^2E)
struct FlowEdge {
    int from, to;
    long long cap, flow = 0;
    FlowEdge(int from, int to, long long cap) : from(from),
                                                to(to), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
    void add_edge(int from, int to, long long cap) {
        edges.emplace_back(from, to, cap);
        edges.emplace_back(to, from, 0);
        adj[from].push_back(m);
        adj[to].push_back(m + 1);
        m += 2;
    }
    bool bfs() {
        while (!q.empty()) {
            int from = q.front();
            q.pop();
            for (int id : adj[from]) {
                if (edges[id].cap == edges[id].flow)
                    continue;
                if (level[edges[id].to] != -1)
                    continue;
                level[edges[id].to] = level[from] + 1;
                q.push(edges[id].to);
            }
        }
    }
    long long dfs(int from, long long pushed) {
        if (pushed == 0)
            return 0;
        if (from == t)
            return pushed;
        for (int& cid = ptr[from]; cid < (int)adj[from].size(); cid++) {
            int id = adj[from][cid];
            int to = edges[id].to;
            if (level[from] + 1 != level[to])
                continue;
            long long tr = dfs(to, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }
    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf))
                f += pushed;
        }
        return f;
    }
}

// topo-sort DAG
// lexicograficamente menor.
// N: nÃºmero de vÃ¡rtices (1-indexado)
// adj: lista de adjacÃªncia do grafo
const int MAXN = 5 * 1e5 + 2;
vector<int> adj[MAXN];
int N;
vector<int> kahn() {
    vector<int> indegree(N + 1, 0);
    for (int u = 1; u <= N; u++) {
        for (int v : adj[u]) {
            indegree[v]++;
        }
    }
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 1; i <= N; i++) {
        if (indegree[i] == 0) {
            pq.push(i);
        }
    }
    vector<int> result;
    while (!pq.empty()) {
        int u = pq.top();
        pq.pop();

```

```

27         result.push_back(u);
28         for (int v : adj[u]) {
29             indegree[v]--;
30             if (indegree[v] == 0) {
31                 pq.push(v);
32             }
33         }
34     }
35     if (result.size() != N) {
36         return {};
37     }
38     return result;
39 }
```

7.6 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;
19 }
```

7.7 Acha Pontes

```

1 vector<int> d, low, pai;      // d[v] Tempo de
   descoberta (discovery time)
2 vector<bool> vis;
3 vector<int> pontos_articulacao;
4 vector<pair<int, int>> pontes;
5 int tempo;
6
7 vector<vector<int>> adj;
8
9 void dfs(int u) {
10    vis[u] = true;
11    tempo++;
12    d[u] = low[u] = tempo;
13    int filhos_dfs = 0;
14    for (int v : adj[u]) {
15        if (v == pai[u]) continue;
16        if (vis[v]) { // back edge
17            low[u] = min(low[u], d[v]);
18        } else {
19            pai[v] = u;
20            filhos_dfs++;
21            dfs(v);
22            low[u] = min(low[u], low[v]);
23            if (pai[u] == -1 && filhos_dfs > 1) {
24                pontos_articulacao.push_back(u);
25            }
26            if (pai[u] != -1 && low[v] >= d[u]) {
27                pontos_articulacao.push_back(u);
28            }
29            if (low[v] > d[u]) {
30                pontes.push_back({min(u, v), max(u, v)});
31            }
32        }
33    }
34 }
```

```

32         }
33     }
34 }
```

7.8 Edmonds-karp

```

1 // Edmonds-Karp com scaling O(E log(F))
2
3 int n, m;
4 const int MAXN = 510;
5 vector<vector<int>> capacity(MAXN, vector<int>(MAXN,
   0));
6 vector<vector<int>> adj(MAXN);
7
8 int bfs(int s, int t, int scale, vector<int>& parent)
9 {
10    fill(parent.begin(), parent.end(), -1);
11    parent[s] = -2;
12    queue<pair<int, int>> q;
13    q.push({s, LLONG_MAX});
14
15    while (!q.empty()) {
16        int cur = q.front().first;
17        int flow = q.front().second;
18        q.pop();
19
20        for (int next : adj[cur]) {
21            if (parent[next] == -1 && capacity[cur][next] >= scale) {
22                parent[next] = cur;
23                int new_flow = min(flow, capacity[cur][next]);
24
25                if (next == t)
26                    return new_flow;
27                q.push({next, new_flow});
28            }
29        }
30    }
31    return 0;
32}
33
34 int maxflow(int s, int t) {
35    int flow = 0;
36    vector<int> parent(MAXN);
37    int new_flow;
38    int scaling = 111 << 62;
39
40    while (scaling > 0) {
41        while (new_flow = bfs(s, t, scaling, parent))
42        {
43            if (new_flow == 0) continue;
44            flow += new_flow;
45            int cur = t;
46            while (cur != s) {
47                int prev = parent[cur];
48                capacity[prev][cur] -= new_flow;
49                capacity[cur][prev] += new_flow;
50                cur = prev;
51            }
52            scaling /= 2;
53        }
54    }
55    return flow;
56 }
```

// Ordena as arestas por peso, insere se ja nao estiver no mesmo componente
// O(E log E)

7.9 Kruskal

```

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 + 1, 1), c(n) {
8         for (int i = 1; i <= n; ++i) par[i] = i;
9     }
10    int find(int i) {
11        return (par[i] == i ? i : (par[i] = find(par[i])));
12    }
13    bool same(int i, int j) {
14        return find(i) == find(j);
15    }
16    int get_size(int i) {
17        return sz[find(i)];
18    }
19    int count() {
20        return c; // quantos componentes conexos
21    }
22    int merge(int i, int j) {
23        if ((i = find(i)) == (j = find(j))) return -1;
24        else --c;
25        if (rank[i] > rank[j]) swap(i, j);
26        par[i] = j;
27        sz[j] += sz[i];
28        if (rank[i] == rank[j]) rank[j]++;
29        return j;
30    }
31 };
32
33 struct Edge {
34     int u, v, w;
35     bool operator <(Edge const & other) {
36         return weight < other.weight;
37     }
38 }
39
40 vector<Edge> kruskal(int n, vector<Edge> edges) {
41     vector<Edge> mst;
42     DSU dsu = DSU(n + 1);
43     sort(edges.begin(), edges.end());
44     for (Edge e : edges) {
45         if (dsu.find(e.u) != dsu.find(e.v)) {
46             mst.push_back(e);
47             dsu.join(e.u, e.v);
48         }
49     }
50     return mst;
51 }
```

7.10 Bellman Ford

```

1 struct Edge {
2     int u, v, w;
3 };
4
5 // se x = -1, nÃ¢o 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     }
20 }
```

```

18                     pai[e.v] = e.u;
19                     x = e.v;
20                 }
21             }
22         }
23     } // achando caminho (se precisar)
24     for (int i = 0; i < n; i++) x = pai[x];
25
26     vector<int> ciclo;
27     for (int v = x;; v = pai[v]) {
28         cycle.push_back(v);
29         if (v == x && ciclo.size() > 1) break;
30     }
31     reverse(ciclo.begin(), ciclo.end());
32 }
```

7.11 Lca Jc

```

1 const int MAXN = 200005;
2 int N;
3 int LOG;
4
5 vector<vector<int>> adj;
6 vector<int> profundidade;
7 vector<vector<int>> cima; // cima[v][j] Ã¢o 2^j-Ã¢lximo ancestral de v
8
9 void dfs(int v, int p, int d) {
10     profundidade[v] = d;
11     cima[v][0] = p; // o pai direto Ã¢o 2^0-Ã¢lximo ancestral
12     for (int j = 1; j < LOG; j++) {
13         // se o ancestral 2^(j-1) existir, calculamos
14         // o 2^j
15         if (cima[v][j - 1] != -1) {
16             cima[v][j] = cima[cima[v][j - 1]][j - 1];
17         } else {
18             cima[v][j] = -1; // nÃ¢o tem ancestral
19         }
20     }
21     for (int nei : adj[v]) {
22         if (nei != p) {
23             dfs(nei, v, d + 1);
24         }
25     }
26 }
27 void build(int root) {
28     LOG = ceil(log2(N));
29     profundidade.assign(N + 1, 0);
30     cima.assign(N + 1, vector<int>(LOG, -1));
31     dfs(root, -1, 0);
32 }
33
34 int get_lca(int a, int b) {
35     if (profundidade[a] < profundidade[b]) {
36         swap(a, b);
37     }
38     // sobe 'a' atÃ¢l a mesma profundidade de 'b'
39     for (int j = LOG - 1; j >= 0; j--) {
40         if (profundidade[a] - (1 << j) >=
41             profundidade[b]) {
42             a = cima[a][j];
43         }
44     }
45     // se 'b' era um ancestral de 'a', entÃ¢o 'a'
46     // agora Ã¢o igual a 'b'
47     if (a == b) {
48         return a;
49     }
50 }
```

```

49 // sobe os dois nães juntos atÃl encontrar os
50 // filhos do LCA
51 for (int j = LOG - 1; j >= 0; j--) {
52     if (cima[a][j] != -1 && cima[a][j] != cima[b]
53         [j]) {
54         a = cima[a][j];
55         b = cima[b][j];
56     }
57 }

```

7.12 Lca

```

1 // LCA - CP algorithm
2 // preprocessing O(NlogN)
3 // lca O(logN)
4 // Uso: criar LCA com a quantidade de vÃrtices (n) e
5 // lista de adjacÃncia (adj)
6 // chamar a funÃ§Ão preprocess com a raiz da Ãrvore
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    };

```

7.13 Kosaraju

```

1     bool vis[MAXN];
2     vector<int> order;
3     int component[MAXN];
4     int N, m;
5     vector<int> adj[MAXN], adj_rev[MAXN];
6
7     // dfs no grafo original para obter a ordem (pÃss-
8     // order)
9     void dfs1(int u) {
10        vis[u] = true;
11        for (int v : adj[u]) {
12            if (!vis[v]) {
13                dfs1(v);
14            }
15        }
16    }
17
18    // dfs o grafo reverso para encontrar os SCCs
19    void dfs2(int u, int c) {
20        component[u] = c;
21        for (int v : adj_rev[u]) {
22            if (component[v] == -1) {
23                dfs2(v, c);
24            }
25        }
26    }
27
28    int kosaraju() {
29        order.clear();
30        fill(vis + 1, vis + N + 1, false);
31        for (int i = 1; i <= N; i++) {
32            if (!vis[i]) {
33                dfs1(i);
34            }
35        }
36        fill(component + 1, component + N + 1, -1);
37        int c = 0;
38        reverse(order.begin(), order.end());
39        for (int u : order) {
40            if (component[u] == -1) {
41                dfs2(u, c++);
42            }
43        }
44        return c;
45    }

```

7.14 Pega Ciclo

```

1 // encontra um ciclo em g (direcionado ou nÃo)
2 // g[u] = vector<pair<id_aresta, vizinho>>
3 // rec_arestas: true -> retorna ids das arestas do
4 // ciclo; false -> retorna vÃrtices do ciclo
5 // directed: grafo direcionado?
6
7 const int MAXN = 5 * 1e5 + 2;
8 vector<pair<int, int>> g[MAXN];
9 int N;
10 bool DIRECTED = false;
11
12 vector<int> color(MAXN), parent(MAXN, -1), edgein(
13     MAXN, -1); // color: 0,1,2 ; edgein[v] = id da
14 // aresta que entra em v
15 int ini_ciclo = -1, fim_ciclo = -1, back_edge_id =
16 -1;
17
18 bool dfs(int u, int pai_edge){
19     color[u] = 1; // cinza
20     for (auto [id, v] : g[u]) {
21         if (!DIRECTED && id == pai_edge) continue; // /
22         ignorar aresta de volta ao pai em nÃo-dir
23         if (color[v] == 0) {
24             parent[v] = u;
25         }
26     }
27 }

```

```

20     edgein[v] = id;
21     if (dfs(v, id)) return true;
22 } else if (color[v] == 1) {
23     // back-edge u -> v detectado
24     ini_ciclo = u;
25     fim_ciclo = v;
26     back_edge_id = id;
27     return true;
28 }
29 // se color[v] == 2, ignora
30 }
31 color[u] = 2; // preto
32 return false;
33 }
34
35 // retorna ids das arestas do ciclo (vazio se nÃ£o
36 // hÃ¡)
37 vector<int> pega_ciclo(bool rec_arestas) {
38     for (int u = 1; u <= N; u++) {
39         if (color[u] != 0) continue;
40         if (dfs(u, -1)) {
41             // reconstrÃ£i caminho u -> ... -> v via
42             parent
43                 vector<int> path;
44                 int cur = ini_ciclo;
45                 path.push_back(cur);
46                 while (cur != fim_ciclo) {
47                     cur = parent[cur];
48                     path.push_back(cur);
49                 }
50                 // path = [u, ..., v] -> inverter para [v
51                 , ..., u]
52                 reverse(path.begin(), path.end());
53                 if (!rec_arestas) return path;
54                 // converte para ids das arestas: edgein[
55                 node] Ãºl a aresta que entra em node
56                 vector<int> edges;
57                 for (int i = 1; i < path.size(); i++)
58                     edges.push_back(edgein[path[i]]);
59                 // adiciona a aresta de retorno u -> v
60                 edges.push_back(back_edge_id);
61                 return edges;
62             }
63         }
64     }
65     return {};
66 }

```

7.15 Min Cost Max Flow

```

1 // Encontra o menor custo para passar K de fluxo em
2 // um grafo com N vertices
3 // Funciona com multiplas arestas para o mesmo par de
4 // vertices
5 // Para encontrar o min cost max flow Ãºs sÃ¡o fazer K
6 // infinito
7
8 struct Edge {
9     int from, to, capacity, cost, id;
10 }
11
12 const int INF = LLONG_MAX;
13
14 void shortest_paths(int n, int v0, vector<int>& dist,
15     vector<int>& edge_to) {
16     dist.assign(n, INF);
17     dist[v0] = 0;
18     vector<bool> in_queue(n, false);
19     queue<int> q;
20     q.push(v0);
21
22     edge_to.assign(n, -1);
23
24     while (!q.empty()) {
25         int u = q.front();
26         q.pop();
27         in_queue[u] = false;
28         for (auto [v, id] : adj[u]) {
29             if (edges[id].capacity > 0 && dist[v] >
30                 dist[u] + edges[id].cost) {
31                 dist[v] = dist[u] + edges[id].cost;
32                 edge_to[v] = id;
33                 if (!in_queue[v]) {
34                     in_queue[v] = true;
35                     q.push(v);
36                 }
37             }
38         }
39     }
40
41     void add_edge(int from, int to, int capacity, int
42     cost){
43         edges.push_back({from, to, capacity, cost, (int)
44         edges.size()});
45         edges.push_back({to, from, 0, -cost, (int)edges.
46         size()}); // reversa
47     }
48
49     int min_cost_flow(int N, int K, int s, int t) {
50         adj.assign(N, vector<array<int, 2>>());
51
52         for (Edge e : edges) {
53             adj[e.from].push_back({e.to, e.id});
54         }
55
56         int flow = 0;
57         int cost = 0;
58         vector<int> dist, edge_to;
59         while (flow < K) {
60             shortest_paths(N, s, dist, edge_to);
61             if (dist[t] == INF)
62                 break;
63
64             // find max flow on that path
65             int f = K - flow;
66             int cur = t;
67             while (cur != s) {
68                 f = min(f, edges[edge_to[cur]].capacity);
69                 cur = edges[edge_to[cur]].from;
70             }
71
72             // apply flow
73             flow += f;
74             cost += f * dist[t];
75             cur = t;
76             while (cur != s) {
77                 int edge = edge_to[cur];
78                 int rev_edge = edge^1;
79
80                 edges[edge].capacity -= f;
81                 edges[rev_edge].capacity += f;
82                 cur = edges[edge].from;
83             }
84
85         }
86
87         if (flow < K)
88             return -1;
89         else
90             return cost;
91     }
92
93 }

```

8 Primitives

9 DP

9.1 Lis

```

1 int lis_nlogn(vector<int> &v) {
2     vector<int> lis;
3     lis.push_back(v[0]);
4     for (int i = 1; i < v.size(); i++) {
5         if (v[i] > lis.back()) {
6             // estende a LIS.
7             lis.push_back(v[i]);
8         } else {
9             // encontra o primeiro elemento em lis
10            que Āl >= v[i].
11            // subsequÃancia de mesmo comprimento,
12            mas com um final menor.
13            auto it = lower_bound(lis.begin(), lis.
14            end(), v[i]);
15            *it = v[i];
16        }
17    }
18    return lis.size();
19 }
20
21 // LIS NA ARVORE
22 const int MAXN_TREE = 100001;
23 vector<int> adj[MAXN_TREE];
24 int values[MAXN_TREE];
25 int ans = 0;
26
27 void dfs(int u, int p, vector<int>& tails) {
28     auto it = lower_bound(tails.begin(), tails.end(),
29     values[u]);
30     int prev = -1;
31     bool coloquei = false;
32     if (it == tails.end()) {
33         tails.push_back(values[u]);
34         coloquei = true;
35     } else {
36         prev = *it;
37         *it = values[u];
38     }
39     ans = max(ans, (int)tails.size());
40     for (int v : adj[u]) {
41         if (v != p) {
42             dfs(v, u, tails);
43         }
44     }
45     if (coloquei) {
46         tails.pop_back();
47     } else {
48         *it = prev;
49     }
50 }
```

9.2 Edit Distance

```

1 vector<vector<int>> dp(n+1, vector<int>(m+1, LINF
2 ));
```

for(int j = 0; j <= m; j++) {
4 dp[0][j] = j;
5 }
6
7 for(int i = 0; i <= n; i++) {
8 dp[i][0] = i;
9 }

```

11     for( int i = 1; i <= n; i++) {
12         for( int j = 1; j <= m; j++) {
13             if(a[i-1] == b[j-1]) {
14                 dp[i][j] = dp[i-1][j-1];
15             } else {
16                 dp[i][j] = min({dp[i-1][j] + 1, dp[i].
17 [j-1] + 1, dp[i-1][j-1] + 1});
18             }
19         }
20     }
21     cout << dp[n][m];
22 }
```

9.3 Bitmask

```

1 // dp de intervalos com bitmask
2 int prox(int idx) {
3     return lower_bound(S.begin(), S.end(), array<int>,
4     {S[idx][1], 011, 011, 011}) - S.begin();
5 }
6 int dp[1002][(int)(111 << 10)];
7
8 int rec(int i, int vis) {
9     if (i == (int)S.size()) {
10         if (__builtin_popcountll(vis) == N) return 0;
11         return LLONG_MIN;
12     }
13     if (dp[i][vis] != -1) return dp[i][vis];
14     int ans = rec(i + 1, vis);
15     ans = max(ans, rec(prox(i), vis | (111 << S[i].
16 [3])) + S[i][2]);
17     return dp[i][vis] = ans;
18 }
```

9.4 Lcs

```

1 string s1, s2;
2 int dp[1001][1001];
3
4 int lcs(int i, int j) {
5     if (i < 0 || j < 0) return 0;
6     if (dp[i][j] != -1) return dp[i][j];
7     if (s1[i] == s2[j]) {
8         return dp[i][j] = 1 + lcs(i - 1, j - 1);
9     } else {
10         return dp[i][j] = max(lcs(i - 1, j), lcs(i, j
11 - 1));
12     }
13 }
```

9.5 Digit

```

1 vector<int> digits;
2
3 int dp[20][10][2][2];
4
5 int rec(int i, int last, int flag, int started) {
6     if (i == (int)digits.size()) return 1;
7     if (dp[i][last][flag][started] != -1) return dp[i].
8     [last][flag][started];
9     int lim;
10    if (flag) lim = 9;
11    else lim = digits[i];
12    int ans = 0;
13    for (int d = 0; d <= lim; d++) {
14        if (started && d == last) continue;
15        int new_flag = flag;
16        int new_started = started;
17        if (d > 0) new_started = 1;
18        if (!flag && d < lim) new_flag = 1;
19        ans += rec(i + 1, d, new_flag, new_started);
20    }
21 }
```

```

19     }
20     return dp[i][last][flag][started] = ans;
21 }

```

9.6 Knapsack

```

1 // dp[i][j] => i-esimo item com j-carga sobrando na
2 // mochila
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 }

```

9.7 Lis Seg

```

1 vector<int> a(n);
2 for (int i = 0; i < n; i++) cin >> a[i];
3 vector<int> sorted_a = a;
4 sort(sorted_a.begin(), sorted_a.end());
5 for (int i = 0; i < n; i++) {
6     a[i] = lower_bound(sorted_a.begin(), sorted_a
7 .end(), a[i]) - sorted_a.begin();
8 }
9 SegTreeMx segmx;
10 segmx.build(n);
11 vector<int> dp(n, 1);
12 for (int k = 0; k < n; k++) {
13     if (a[k] > 0) {
14         dp[k] = segmx.query(0, a[k] - 1) + 1;
15     }
16     segmx.update(a[k], dp[k]);
17 }
18 cout << *max_element(dp.begin(), dp.end()) << '\n';
19 ;

```

9.8 Disjoint Blocks

```

1 // Número máximo de subarrays disjuntos com soma x
2 // usando apenas
3 // prefixo até i (ou seja, considerando prefixo a
4 // [1..i]).
5 int disjointSumX(vector<int> &a, int x) {
6     int n = a.size();
7     map<int, int> best; // best[pref] = melhor dp
8     visto para esse pref
9     best[0] = 0;
10    int pref = 0;
11    vector<int> dp(n + 1, 0); // dp[0] = 0
12    for (int i = 1; i <= n; i++) {
13        pref += a[i - 1];
14        // não pegar subarray terminando em i
15        dp[i] = dp[i-1];
16        // pega se existir prefixo anterior e
17        // atualiza best
18        auto it = best.find(pref - x);
19        if (it != best.end()) {
20            dp[i] = max(dp[i], it->second + 1);
21        }
22        best[pref] = max(best[pref], dp[i]);
23    }
24 }

```

```

20     return dp[n];
21 }

```

10 General

10.1 Brute Choose

```

1 vector<int> elements;
2 int N, K;
3 vector<int> comb;
4
5
6 void brute_choose(int i) {
7     if (comb.size() == K) {
8         for (int j = 0; j < comb.size(); j++) {
9             cout << comb[j] << ' ';
10        }
11        cout << '\n';
12        return;
13    }
14    if (i == N) return;
15    int r = N - i;
16    int preciso = K - comb.size();
17    if (r < preciso) return;
18    comb.push_back(elements[i]);
19    brute_choose(i + 1);
20    comb.pop_back();
21    brute_choose(i + 1);
22 }

```

10.2 Struct

```

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

```

10.3 Mex

```

1 struct MEX {
2     map<int, int> f;
3     set<int> falta;
4     int tam;
5     MEX(int n) : tam(n) {
6         for (int i = 0; i <= n; i++) falta.insert(i);
7     }
8     void add(int x) {
9         f[x]++;
10        if (f[x] == 1 && x >= 0 && x <= tam) {
11            falta.erase(x);
12        }
13    }
14    void rem(int x) {
15        if (f.count(x) && f[x] > 0) {
16            f[x]--;
17            if (f[x] == 0 && x >= 0 && x <= tam) {
18                falta.insert(x);
19            }
20        }
21    }
22    int get() {
23        if (falta.empty()) return tam + 1;
24        return *falta.begin();
25    }
26 }

```

```

25     }
26 }

10.4 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 }

```

11 Geometry

11.1 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
51 int ccw(point p1, point p2, point p3)
52 {
53     cod cross = (p2-p1) ^ (p3-p1);
54     if(cross == 0) return 0;
55     else if(cross < 0) return -1;
56     else return 1;
57 }
58
59 vector<point> convex_hull(vector<point> p)
60 {
61     sort(p.begin(), p.end());
62     vector<point> L,U;
63
64     //Lower
65     for(auto pp : p)
66     {
67         while(L.size() >= 2 and ccw(L[L.size() - 2],
68         L.back(), pp) == -1)
69         {
70             // Àl -1 pq eu nÃo quero excluir os
71             colineares
72             L.pop_back();
73         }
74         L.push_back(pp);
75     }
76
77     reverse(p.begin(), p.end());
78
79     //Upper
80     for(auto pp : p)
81     {
82         while(U.size() >= 2 and ccw(U[U.size() - 2], U
83         .back(), pp) == -1)
84         {
85             U.pop_back();
86         }
87         U.push_back(pp);
88     }
89
90     L.pop_back();
91     L.insert(L.end(), U.begin(), U.end() - 1);
92     return L;
93 }

```

```

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
113     int n;
114     cin >> n;
115
116     vector<point> v(n);
117     for(int i = 0; i < n; i++)
118     {
119         cin >> v[i].x >> v[i].y;
120     }
121
122     vector<point> ch = convex_hull(v);
123
124     cout << ch.size() << '\n';
125     for(auto p : ch) cout << p.x << " " << p.y << "\n"
126     ";
127
128     return 0;
}

```

11.2 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 bool inside(vp &p, point e){ // ccw
11     int l=2, r=(int)p.size()-1;
12     while(l<r){
13         int mid = (l+r)/2;
14         if(ccw(p[0], p[mid], e) == 1)
15             l=mid+1;
16         else{
17             r=mid;
18         }
19     }
20     // bordo
21     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
22     ==0) return false;
23     // if(r==2 and ccw(p[0], p[1], e)==0) return
24     // false;
25     return insideT(p[0], p[r-1], p[r], e);
}
26

```

```

28 // Any O(n)
29
30 int inside(vp &p, point pp){
31     // 1 - inside / 0 - boundary / -1 - outside
32     int n = p.size();
33     for(int i=0; i<n; i++){
34         int j = (i+1)%n;
35         if(line({p[i], p[j]}).inside_seg(pp))
36             return 0;
37     }
38     int inter = 0;
39     for(int i=0; i<n; i++){
40         int j = (i+1)%n;
41         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
42 [i], p[j], pp)==1)
43             inter++; // up
44         else if(p[j].x <= pp.x and pp.x < p[i].x and
45         ccw(p[i], p[j], pp)==-1)
46             inter++; // down
47     }
48     if(inter%2==0) return -1; // outside
49     else return 1; // inside
}

```

11.3 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), deltay1 = (y1-y2);
13
14         int compx = (x1-x3), compy = (y1-y3);
15
16         int ans = (deltax1*compy) - (compx*deltay1);
17
18         if(ans == 0){cout << "TOUCH\n"; continue;}
19         if(ans < 0){cout << "RIGHT\n"; continue;}
20         if(ans > 0){cout << "LEFT\n"; continue;}
21     }
22 }

```

11.4 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 int32_t main() {
13     ll x1, y1, x2, y2, x3, y3;
14     cin >> x1 >> y1;
15     cin >> x2 >> y2;
16     cin >> x3 >> y3;
17     ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
18
19 }
20

```

```
17     11 tot_borda = pontos_borda(x1, y1, x2, y2) +      20      cout << ans << endl;
18     12               pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,      21
19     13               y3, x1, y1);                                22      return 0;
20   14 }                                              23 }
```