

Competitive Programming Notebook

Programadores Roblox

Contents

1 String	2	8 General	9
1.1 Lcs	2	8.1 Struct	9
1.2 Kmp	2	8.2 Bitwise	10
1.3 Trie Ponteiros	2	9 Geometry	10
1.4 Hashing	2	9.1 Convex Hull	10
1.5 Countpermutations	2	9.2 Point Location	11
1.6 Trie	3	9.3 Inside Polygon	11
1.7 Z Function	3	9.4 Lattice Points	11
2 Graph	3	10 String copy	12
2.1 Lca Jc	3	10.1 Lcs	12
2.2 Dfs	3	10.2 Kmp	12
2.3 Dijkstra	4	10.3 Trie Ponteiros	12
2.4 Kruskal	4	10.4 Hashing	12
2.5 Bellman Ford	4	10.5 Countpermutations	13
2.6 Lca	4	10.6 Z Function	13
2.7 Floyd Warshall	5		
2.8 Topological Sort	5		
3 DP	5		
3.1 Lcs	5		
3.2 Lis	5		
3.3 Knapsack	5		
4 Primitives	5		
5 Search and sort	5		
5.1 Mergeandcount	5		
5.2 Bfs	6		
5.3 Dfs	6		
6 Math	6		
6.1 Mod Inverse	6		
6.2 Fexp	7		
6.3 Equacao Diofantina	7		
6.4 Crivo	7		
6.5 Exgcd	7		
6.6 Divisores	7		
7 DS	7		
7.1 Segtree Iterativa	7		
7.2 Ordered Set E Map	7		
7.3 Psum 2d	8		
7.4 Bit	8		
7.5 Segtree Sum	8		
7.6 Segtree Gcd	9		
7.7 Dsu	9		

1 String

1.1 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 }
```

1.2 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 }
```

1.3 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 }
```

```

31     }
32     return false;
33 }
34 return cur->acaba;
35 }
36
37 // Retorna se Ã 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 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 struct Hash {
9     const int X = 2147483647;
10    const int MOD = 1e9+7;
11    int n; string s;
12    vector<int> h, hi, p;
13    Hash() {}
14    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
15        (n) {
16        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
17            % 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)
22                % MOD;
23    }
24    int query(int l, int r) {
25        int hash = (h[r] - (l ? h[l-1]:0)*p[r-l+1]%MOD :
26            0));
27        return hash < 0 ? hash + MOD : hash;
28    }
29    int query_inv(int l, int r) {
30        int hash = (hi[l] - (r+1 < n ? hi[r+1]:0)*p[r-l
31            +1] % MOD : 0));
32        return hash < 0 ? hash + MOD : hash;
33    }
34 };
35
```

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 // 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 }
```

```

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     return ans;
32 }

```

1.6 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 }

```

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

```

2 Graph

2.1 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]; //
8     // tanto que tenho que subir
9     for(int j = LOG-1; j >= 0; j--) {
10         if((1 << j) & k) {
11             a = cima[a][j];
12         }
13     }
14     if(a == b) return a; // ja to no lca
15
16     for(int j = LOG-1; j >= 0; j--) { // subo com os
17         // dois atÃ chegar no lca fazendo binary lifting
18         if(cima[a][j] != cima[b][j]) {
19             a = cima[a][j];
20             b = cima[b][j];
21         }
22     }
23     return cima[a][0];
24 }
25
26 void dfs(int v, int p) {
27     if(v != 1) profundidade[v] = profundidade[p] + 1;
28     cima[v][0] = p;
29     for(int j = 1; j < LOG; j++) {
30         if(cima[v][j-1] != -1) {
31             cima[v][j] = cima[cima[v][j-1]][j-1];
32         } else {
33             cima[v][j] = -1;
34         }
35     }
36     for(auto &nei : adj[v]) {
37         if(nei != p) {
38             dfs(nei, v);
39         }
40     }
41 }
42
43 while((1 << LOG) <= n) LOG++;

```

2.2 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 }

```

2.3 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     pq.push({0, S});
10    while(pq.size()) {
11        ll v = pq.top().second;
12        pq.pop();
13        if(vis[v]) continue;
14        vis[v] = 1;
15        for(auto &[peso, vizinho] : adj[v]) {
16            if(dist[vizinho] > dist[v] + peso) {
17                dist[vizinho] = dist[v] + peso;
18                pq.push({dist[vizinho], vizinho});
19            }
20        }
21    }
22    return dist;
23 }

```

2.4 Kruskal

```

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

```

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

```

2.6 Lca

```

1 // LCA - CP algorithm
2 // preprocessing O(NlogN)
3 // lca O(logN)
4 // Uso: criar LCA com a quantidade de vértices (n) e
5 // lista de adjacência (adj)
6 // chamar a função preproc com a raiz da árvore
7
8 struct LCA {
9     int n, l, timer;
10    vector<vector<int>> adj;
11    vector<int> tin, tout;
12    vector<vector<int>> up;
13
14    LCA(int n, const vector<vector<int>>& adj) : n(n), adj(adj) {}

```

```

15 void dfs(int v, int p) {
16     tin[v] = ++timer;
17     up[v][0] = p;
18     for (int i = 1; i <= 1; ++i)
19         up[v][i] = up[up[v][i-1]][i-1];
20
21     for (int u : adj[v]) {
22         if (u != p)
23             dfs(u, v);
24     }
25
26     tout[v] = ++timer;
27 }
28
29 bool is_ancestor(int u, int v) {
30     return tin[u] <= tin[v] && tout[u] >= tout[v];
31 }
32
33 int lca(int u, int v) {
34     if (is_ancestor(u, v))
35         return u;
36     if (is_ancestor(v, u))
37         return v;
38     for (int i = 1; i >= 0; --i) {
39         if (!is_ancestor(up[u][i], v))
40             u = up[u][i];
41     }
42     return up[u][0];
43 }
44
45 void preprocess(int root) {
46     tin.resize(n);
47     tout.resize(n);
48     timer = 0;
49     l = ceil(log2(n));
50     up.assign(n, vector<int>(l + 1));
51     dfs(root, root);
52 }
53 };

```

2.7 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][k] + dist[k][j]);
12             }
13         }
14     }
15 }
16
17 void solve() {
18     int m, q;
19     cin >> n >> m >> q;
20     for(int i = 0; i < n; i++) {
21         for(int j = i; j < n; j++) {
22             if(i == j) {
23                 dist[i][j] = dist[j][i] = 0;
24             } else {
25                 dist[i][j] = dist[j][i] = linf;
26             }
27         }
28     }
29     for(int i = 0; i < m; i++) {
30         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 }

```

2.8 Topological Sort

```

1 vector<int> adj[MAXN];
2 vector<int> estado(MAXN); // 0: nao visitado 1:
3     processamento 2: processado
4 vector<int> ordem;
5 bool temCiclo = false;
6
7 void dfs(int v) {
8     if(estado[v] == 1) {
9         temCiclo = true;
10        return;
11    }
12    if(estado[v] == 2) return;
13    estado[v] = 1;
14    for(auto &nei : adj[v]) {
15        if(estado[nei] != 2) dfs(nei);
16    }
17    estado[v] = 2;
18    ordem.push_back(v);
19    return;

```

3 DP

3.1 Lcs

3.2 Lis

3.3 Knapsack

```

1 // dp[i][j] => i-esimo item com j-carga sobrando na
2     mochila
3 // O(N * W)
4
5 for(int j = 0; j < MAXN; j++) {
6     dp[0][j] = 0;
7 }
8
9 for(int i = 1; i <= N; i++) {
10    for(int j = 0; j <= W; j++) {
11        if(items[i].first > j) {
12            dp[i][j] = dp[i-1][j];
13        } else {
14            dp[i][j] = max(dp[i-1][j], dp[i-1][j-
15                items[i].first] + items[i].second);
16        }
17    }
18 }

```

4 Primitives

5 Search and sort

5.1 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     int x = m - l + 1; // Tamanho do subarray
      esquerdo.
5     int y = r - m; // Tamanho do subarray direito.
6
7     // Vetores temporarios para os subarray esquerdo
      e direito.
8     vector<int> left(x), right(y);
9
10    for (int i = 0; i < x; i++) left[i] = v[l + i];
11    for (int j = 0; j < y; j++) right[j] = v[m + 1 +
      j];
12
13    int i = 0, j = 0, k = l;
14    int swaps = 0;
15
16    while (i < x && j < y) {
17        if (left[i] <= right[j]) {
18            // Se o elemento da esquerda for menor ou
              igual, coloca no vetor original.
19            v[k++] = left[i++];
20        } else {
21            // Caso contrario, coloca o elemento da
              direita e conta as trocas.
22            v[k++] = right[j++];
23            swaps += (x - i);
24        }
25    }
26
27    // Adiciona os elementos restantes do subarray
      esquerdo (se houver).
28    while (i < x) v[k++] = left[i++];
29
30    // Adiciona os elementos restantes do subarray
      direito (se houver).
31    while (j < y) v[k++] = right[j++];
32
33    return swaps; // Retorna o numero total de
      trocas realizadas.
34 }
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
          vetor.
41         int m = l + (r - l) / 2;
42
43         // Chama merge sort para a metade esquerda.
44         swaps += mergeSort(v, l, m);
45         // Chama merge sort para a metade direita.
46         swaps += mergeSort(v, m + 1, r);
47
48         // Mescla as duas metades e conta as trocas.
49         swaps += mergeAndCount(v, l, m, r);
50     }
51
52     return swaps; // Retorna o numero total de
      trocas no vetor.
53 }

```

5.2 Bfs

```

1 // Printa os nos na ordem em que são 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 void bfs(vector<vector<int>>& grafo, int inicio){
7     set<int> visited;
8     queue<int> fila;
9
10    fila.push(inicio);
11    visited.insert(inicio);
12
13    while(!fila.empty()){
14        int cur = fila.front();
15        fila.pop();
16
17        cout << cur << " "; // printa o não atual
18
19        for(int vizinho: grafo[cur]){
20            if(visited.find(vizinho) == visited.end()
21            ){
22                fila.push(vizinho);
23                visited.insert(vizinho)
24            }
25        }
26    }
27 }

```

5.3 Dfs

```

1 // Printa os nos na ordem em que são 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 }

```

6 Math

6.1 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 }

```

6.2 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 }
```

6.3 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 }
```

6.4 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 }
```

6.5 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 }
```

6.6 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
21            for (int k=0; k<len; k++)
22                ans.push_back(cur*ans[k]);
23        }
24        return ans;
25    }
26 }
```

```

20         for (int k=0; k<len; k++)
21             ans.push_back(cur*ans[k]);
22     }
23 }
24
25 return ans;
26 }
```

7 DS

7.1 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
23 + 1]);
24    }
25
26    T range_query(int l, int r) {
27        T res_l = neutral_value, res_r =
28            neutral_value;
29
30        for (l += n, r += n + 1; l < r; l >>= 1, r
31 >>= 1) {
32            if (l & 1) res_l = combine(res_l, tree[l
33 ++]);
34            if (r & 1) res_r = combine(tree[--r],
35 res_r);
36        }
37
38        return combine(res_l, res_r);
39    }
40
41    void update(int pos, T new_val) {
42        tree[pos += n] = new_val;
43
44        for (pos >>= 1; pos > 0; pos >>= 1)
45            tree[pos] = combine(tree[2 * pos], tree[2
46 * pos + 1]);
47    }
48 }
```

7.2 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>;
```

```

8  template <typename T> using o_set = tree<T, null_type
    , less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
9  template <typename T, typename R> using o_map = tree<
    T, R, less<T>, rb_tree_tag,
    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 }

```

7.3 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[
            i-1][j-1];
8      }
9  }
10
11 // y1 eh variavel reservada
12 int retangulo(int x1, int yy1, int x2, int yy2){
13     x2 = min(x2, n), yy2 = min(yy2, m);
14     x1 = max(0LL, x1-1), yy1 = max(0LL, yy1-1);
15
16     return psum[x2][yy2]-psum[x1][yy2]-psum[x2][yy1]+
        psum[x1][yy1];
17 }

```

7.4 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     void update(int idx, int delta) {
19         while (idx <= n) {
20             bit[idx] += delta;
21             idx += idx & -idx;
22         }

```

```

    }
    int query(int idx) {
        return sum(idx);
    }
    int range_query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
31
32 BIT fenwick(n);
33 for(int i = 1; i <= n; i++) {
34     fenwick.update(i, arr[i]);
35 }

```

7.5 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;

```

7.6 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 }

```

```

36 int query(int node, int start, int end, int l,
37 int r) {
38     if (r < start || l > end) {
39         return 0;
40     }
41     if (l <= start && end <= r) {
42         return tree[node];
43     }
44     int mid = (start + end) / 2;
45     int left_gcd = query(2 * node + 1, start, mid
46     , l, r);
47     int right_gcd = query(2 * node + 2, mid + 1,
48     end, l, r);
49     return gcd(left_gcd, right_gcd);
50 }
51 public:
52 SegmentTreeGCD(const vector<int>& arr) {
53     n = arr.size();
54     tree.resize(4 * n);
55     build(arr, 0, 0, n - 1);
56 }
57 void update(int idx, int value) {
58     update(0, 0, n - 1, idx, value);
59 }
60 int query(int l, int r) {
61     return query(0, 0, n - 1, l, r);
62 }
63 };

```

7.7 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 +
5     1, 1), c(n) {
6         for (int i = 1; i <= n; ++i) par[i] = i;
7     }
8     int find(int i) {
9         return (par[i] == i ? i : (par[i] = find(par[
10         i])));
11     }
12     bool same(int i, int j) {
13         return find(i) == find(j);
14     }
15     int get_size(int i) {
16         return sz[find(i)];
17     }
18     int count() {
19         return c; // quantos componentes conexos
20     }
21     int merge(int i, int j) {
22         if ((i = find(i)) == (j = find(j))) return
23         -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 };

```

8 General

8.1 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
          .idade;
9         else return nome > other.nome;
10    }
11 }

```

8.2 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 }

```

9 Geometry

9.1 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
14
15     double modulo()
16     {
17         return sqrt(x*x + y*y);
18     }
19
20     point operator+(point o)
21     {
22         return point(x+o.x, y+o.y);
23     }
24
25     point operator-(point o)
26     {
27         return point(x - o.x , y - o.y);
28     }
29
30     point operator*(cod t)
31     {
32         return point(x*t, y*t);
33     }
34
35     point operator/(cod t)
36     {
37         return point(x/t, y/t);
38     }
39
40     cod operator*(point o)
41     {
42         return x*o.x + y*o.y;
43     }
44
45     cod operator^(point o)
46     {
47         return x*o.y - y * o.x;
48     }
49
50     bool operator<(point o)
51     {
52         if( x != o.x) return x < o.x;
53         return y < o.y;
54     }
55 };
56
57 int ccw(point p1, point p2, point p3)
58 {
59     cod cross = (p2-p1) ^ (p3-p1);
60     if(cross == 0) return 0;
61     else if(cross < 0) return -1;
62     else return 1;
63 }
64
65 vector <point> convex_hull(vector<point> p)
66 {
67     sort(p.begin(), p.end());
68     vector<point> L,U;
69
70     //Lower
71     for(auto pp : p)
72     {
73         while(L.size() >= 2 and ccw(L[L.size() - 2],
74             L.back(), pp) == -1)
75         {
76             // Ãl -1 pq eu nÃo quero excluir os
77             // colineares
78             L.pop_back();
79         }
80         L.push_back(pp);
81     }
82
83     reverse(p.begin(), p.end());
84
85     //Upper
86     for(auto pp : p)
87     {
88         while(U.size() >= 2 and ccw(U[U.size() -2], U
89             .back(), pp) == -1)
90         {

```

```

82         U.pop_back();
83     }
84     U.push_back(pp);
85 }
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
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     return 0;
128 }

```

9.2 Point Location

```

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

```

9.3 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     int inter = 0;
41     for(int i=0;i<n;i++){
42         int j = (i+1)%n;
43         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
44         [i], p[j], pp)==1)
45             inter++; // up
46         else if(p[j].x <= pp.x and pp.x < p[i].x and
47         ccw(p[i], p[j], pp)==-1)
48             inter++; // down
49     }
50
51     if(inter%2==0) return -1; // outside
52     else return 1; // inside
53 }

```

9.4 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 }

```

```

10
11 int32_t main() {
12     ll x1, y1, x2, y2, x3, y3;
13     cin >> x1 >> y1;
14     cin >> x2 >> y2;
15     cin >> x3 >> y3;
16     ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
17     ll tot_borda = pontos_borda(x1, y1, x2, y2) +
        pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,
        y3, x1, y1);
18
19     ll ans = (area - tot_borda) / 2 + 1;
20     cout << ans << endl;
21
22     return 0;
23 }

```

10 String copy

10.1 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 }

```

10.2 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 }

```

10.3 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserir e buscar, 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 }

```

10.4 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 struct Hash {
9     const int X = 2147483647;
10    const int MOD = 1e9+7;
11    int n; string s;
12    vector<int> h, hi, p;
13    Hash() {}
14    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
15    (n) {
16        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
17        % 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)
22            % MOD;
23    }
24    int query(int l, int r) {
25        int hash = (h[r] - (l ? h[l-1]:0)*p[r-l+1]%MOD :
26        0));
27        return hash < 0 ? hash + MOD : hash;
28    }
29    int query_inv(int l, int r) {
30        int hash = (hi[l] - (r+1 < n ? hi[r+1]:0)*p[r-l
31        +1] % MOD : 0));
32        return hash < 0 ? hash + MOD : hash;
33    }
34 };

```

10.5 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
  characters
4 // O(n*freq.size())
5 int countPermLess(vector<int> freq, const string &t)
6 {
7     int n = t.size();
8     int ans = 0;
9
10    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
11    for (int i = 1; i <= n; i++)
12        fact[i] = (fact[i - 1] * i) % MOD;
13    invfact[n] = fexp(fact[n], MOD - 2, MOD);
14    for (int i = n - 1; i >= 0; i--)
15        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
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        }
30    }
31    return ans;
32 }
```

```

26     }
27 }
28 if (freq[t[i] - 'a'] == 0) break;
29 freq[t[i] - 'a']--;
30 }
31 return ans;
32 }
```

10.6 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 - 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 }
```