

Competitive Programming Notebook

Programadores Roblox

Contents

1	General	2
1.1	Struct	2
1.2	Bitwise	2
2	String	2
3	Math	2
3.1	Exgcd	2
3.2	Crivo	2
3.3	Fexp	2
3.4	Equacao Diofantina	2
4	Search and sort	2
4.1	Bfs	2
4.2	Mergeandcount	3
4.3	Dfs	3
5	Primitives	3
6	DP	3
6.1	Lcs	3
6.2	Knapsack	3
6.3	Lis	3
7	DS	4
7.1	Segtree Gcd	4
7.2	Ordered Set E Map	4
7.3	Psum 2d	4
7.4	Dsu	4
7.5	Segtree Sum	5
7.6	Segtree Iterativa	5
7.7	Bit	6
8	Graph	6
8.1	Topological Sort	6
8.2	Bellman Ford	6
8.3	Floyd Warshall	6
8.4	Dijkstra	7
8.5	Kruskal	7
8.6	Dfs	7
8.7	Lca	7
9	Geometry	8
9.1	Lattice Points	8
9.2	Inside Polygon	8
9.3	Point Location	8
9.4	Convex Hull	8

1 General

1.1 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 }
```

1.2 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 }
```

2 String

3 Math

3.1 Exgcd

```

1 // 0 retorno da funcao eh {n, m, g}
```

```

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

3.2 Crivo

```

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

3.3 Fexp

```

1 // a^e mod m
2 // O(log n)
3
4 int fexp(int a, int e, int m) {
5     a %= m;
6     int ans = 1;
7     while (e > 0){
8         if (e & 1) ans = ans*a % m;
9         a = a*a % m;
10        e /= 2;
11    }
12    return ans%m;
13 }
```

3.4 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 }
```

4 Search and sort

4.1 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
7 void bfs(vector<vector<int>>&grafo, int inicio){
8     set<int> visited;
9     queue<int> fila;
10
11     fila.push(inicio);
12     visited.insert(inicio);
13
14     while(!fila.empty()){
15         int cur = fila.front();
16         fila.pop();
17
18         cout << cur << " "; // printa o nós atual
19
20         for(int vizinho: grafo[cur]){
```

```

21         if(visited.find(vizinho) == visited.end()) 52
    ){
22             fila.push(vizinho);
23             visited.insert(vizinho)
24         }
25     }
26 }
27 }

```

4.2 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 }

```

```

return swaps; // Retorna o numero total de
trocas no vetor.
53 }

```

4.3 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.
   end()){
23                     pilha.push(vizinho);
24                 }
25             }
26         }
27     }
28 }

```

5 Primitives

6 DP

6.1 Lcs

6.2 Knapsack

```

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

```

6.3 Lis

7 DS

7.1 Segtree Gcd

```

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

```

```

60         return query(0, 0, n - 1, l, r);
61     }
62 };

```

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>;
10
11 template<typename T> using o_set = tree<T, null_type
12     , less<T>, rb_tree_tag,
13     tree_order_statistics_node_update>;
14
15 template<typename T, typename R> using o_map = tree<
16     T, R, less<T>, rb_tree_tag,
17     tree_order_statistics_node_update>;
18
19 int main() {
20     int i, j, k, n, m;
21     o_set<int>st;
22     st.insert(1);
23     st.insert(2);
24     cout << *st.find_by_order(0) << endl; /// k-esimo
25     elemento
26     cout << st.order_of_key(2) << endl; ///numero de
27     elementos menores que k
28     o_map<int, int>mp;
29     mp.insert({1, 10});
30     mp.insert({2, 20});
31     cout << mp.find_by_order(0)->second << endl; /// k-
32     esimo elemento
33     cout << mp.order_of_key(2) << endl; /// numero de
34     elementos (chave) menores que k
35     return 0;
36 }

```

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[
8         i-1][j-1];
9     }
10 }
11
12 // y1 eh variavel reservada
13 int retangulo(int x1, int yy1, int x2, int yy2){
14     x2 = min(x2, n), y2 = min(y2, m);
15     x1 = max(0LL, x1-1), y1 = max(0LL, y1-1);
16
17     return psum[x2][y2]-psum[x1][y2]-psum[x2][y1]+
18     psum[x1][y1];
19 }

```

7.4 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
9     int find(int i) {

```

```

8         return (par[i] == i ? i : (par[i] = find(par[
9         i])));
10     }
11     bool same(int i, int j) {
12         return find(i) == find(j);
13     }
14     int get_size(int i) {
15         return sz[find(i)];
16     }
17     int count() {
18         return c; // quantos componentes conexos
19     }
20     int merge(int i, int j) {
21         if ((i = find(i)) == (j = find(j))) return
22         -1;
23         else --c;
24         if (rank[i] > rank[j]) swap(i, j);
25         par[i] = j;
26         sz[j] += sz[i];
27         if (rank[i] == rank[j]) rank[j]++;
28         return j;
29     }
30 };

```

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>
29     &v) {
30         if (l == r) {
31             t[p] = v[l];
32         } else {
33             int mid = (l + r) / 2;
34             build(lc(p), l, mid, v);
35             build(rc(p), mid + 1, r, v);
36             t[p] = merge(t[lc(p)], t[rc(p)]);
37         }
38     }
39     void build(int _n) {
40         n = _n;
41         t.assign(n * 4, neutral);
42         lazy.assign(n * 4, 0);
43         replace.assign(n * 4, false);
44     }
45     void build(const vector<ll> &v) {
46         n = (int)v.size();
47         t.assign(n * 4, neutral);

```

```

48         lazy.assign(n * 4, 0);
49         replace.assign(n * 4, false);
50         build(1, 0, n - 1, v);
51     }
52     void build(ll *bg, ll *en) {
53         build(vector<ll>(bg, en));
54     }
55     ll query(int p, int l, int r, int L, int R) {
56         push(p, l, r);
57         if (l > R || r < L) return neutral;
58         if (l >= L && r <= R) return t[p];
59         int mid = (l + r) / 2;
60         auto ql = query(lc(p), l, mid, L, R);
61         auto qr = query(rc(p), mid + 1, r, L, R);
62         return merge(ql, qr);
63     }
64     ll query(int l, int r) { return query(1, 0, n -
65     1, l, r); }
66     void update(int p, int l, int r, int L, int R, ll
67     val, bool repl = 0) {
68         push(p, l, r);
69         if (l > R || r < L) return;
70         if (l >= L && r <= R) {
71             lazy[p] = val;
72             replace[p] = repl;
73             push(p, l, r);
74         } else {
75             int mid = (l + r) / 2;
76             update(lc(p), l, mid, L, R, val, repl);
77             update(rc(p), mid + 1, r, L, R, val, repl
78             );
79             t[p] = merge(t[lc(p)], t[rc(p)]);
80         }
81     }
82     void sumUpdate(int l, int r, ll val) { update(1,
83     0, n - 1, l, r, val, 0); }
84     void assignUpdate(int l, int r, ll val) { update
85     (1, 0, n - 1, l, r, val, 1); }
86 } segsum;

```

7.6 Segtree Iterativa

```

1 // Exemplo de uso:
2 // auto cmp = [](int a, int b) {return a+b;};
3 // SegTree<int> st(vetor, 0, cmp);
4
5 template <typename T>
6 struct SegTree {
7     int n;
8     vector<T> t;
9     T neutral_value;
10    function<T(T, T)> combine;
11
12    SegTree(const vector<T>& data, T neutral,
13    function<T(T, T)> comb)
14        : neutral_value(neutral), combine(comb) {
15        n = data.size();
16        t.resize(2 * n, neutral_value);
17
18        for (int i = 0; i < n; i++)
19            t[n + i] = data[i];
20
21        for (int i = n - 1; i > 0; --i)
22            t[i] = combine(t[i * 2], t[i * 2 + 1]);
23    }
24
25    T range_query(int l, int r) {
26        T result = neutral_value;
27        for (l += n, r += n + 1; l < r; l >>= 1, r
28        >>= 1) {
29            if (l & 1) result = combine(result, t[l
30            ++]);

```

```

28         if (r & 1) result = combine(result, t[--r]);
29     }
30     return result;
31 }
32
33 void update(int pos, T new_val) {
34     t[pos += n] = new_val;
35     for (pos >= 1; pos > 0; pos >= 1)
36         t[pos] = combine(t[2 * pos], t[2 * pos +
37     1]);
38 }

```

7.7 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         }
23     }
24     int query(int idx) {
25         return sum(idx);
26     }
27     int range_query(int l, int r) {
28         return sum(r) - sum(l - 1);
29     }
30 };
31
32 BIT fenwick(n);
33 for(int i = 1; i <= n; i++) {
34     fenwick.update(i, arr[i]);
35 }

```

8 Graph

8.1 Topological Sort

```

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

```

```

15    }
16    estado[v] = 2;
17    ordem.push_back(v);
18    return;

```

8.2 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());

```

8.3 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] = 1inf;
26             }
27         }
28     }
29     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 }

```

8.4 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;
24 }

```

8.5 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 +
8     1, 1), c(n) {
9         for (int i = 1; i <= n; ++i) par[i] = i;
10    }
11    int find(int i) {
12        return (par[i] == i ? i : (par[i] = find(par[
13        i])));
14    }
15    bool same(int i, int j) {
16        return find(i) == find(j);
17    }
18    int get_size(int i) {
19        return sz[find(i)];
20    }
21    int count() {
22        return c; // quantos componentes conexos
23    }
24    int merge(int i, int j) {
25        if ((i = find(i)) == (j = find(j))) return
26        -1;
27        else --c;
28        if (rank[i] > rank[j]) swap(i, j);

```

```

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 }

```

8.6 Dfs

```

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

```

8.7 Lca

```

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;

```

```

32     }
33 }
34 for(auto &nei : adj[v]) {
35     if(nei != p) {
36         dfs(nei, v);
37     }
38 }
39 }
40
41 while((1 << LOG) <= n) LOG++;

```

9 Geometry

9.1 Lattice Points

```

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

```

9.2 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
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;

```

```

24     return insideT(p[0], p[r-1], p[r], e);
25 }
26
27 // Any O(n)
28
29 int inside(vp &p, point pp){
30     // 1 - inside / 0 - boundary / -1 - outside
31     int n = p.size();
32     for(int i=0;i<n;i++){
33         int j = (i+1)%n;
34         if(line({p[i], p[j]}).inside_seg(pp))
35             return 0;
36     }
37     int inter = 0;
38     for(int i=0;i<n;i++){
39         int j = (i+1)%n;
40         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
41         [i], p[j], pp)==1)
42             inter++; // up
43         else if(p[j].x <= pp.x and pp.x < p[i].x and
44         ccw(p[i], p[j], pp)==-1)
45             inter++; // down
46     }
47     if(inter%2==0) return -1; // outside
48     else return 1; // inside
49 }

```

9.3 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), deltax2 = (x1-x3);
13         int compx = (x1-x3), compy = (y1-y3);
14
15         int ans = (deltax1*compy) - (compx*deltax2);
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 }

```

9.4 Convex Hull

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {
9     cod x,y;
10     point(cod x = 0, cod y = 0): x(x), y(y)
11     {}
12
13     double modulo()
14     {
15         return sqrt(x*x + y*y);

```



```

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

```

```

73     }
74
75     reverse(p.begin(), p.end());
76
77     //Upper
78     for(auto pp : p)
79     {
80         while(U.size() >= 2 and ccw(U[U.size()-2], U
81         .back(), pp) == -1)
82         {
83             U.pop_back();
84         }
85         U.push_back(pp);
86     }
87
88     L.pop_back();
89     L.insert(L.end(), U.begin(), U.end()-1);
90     return L;
91 }
92
93 cod area(vector<point> v)
94 {
95     int ans = 0;
96     int aux = (int)v.size();
97     for(int i = 2; i < aux; i++)
98     {
99         ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
100     }
101     ans = abs(ans);
102     return ans;
103 }
104
105 int bound(point p1 , point p2)
106 {
107     return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
108 }
109 //teorema de pick [pontos = A - (bound+points)/2 + 1]
110
111 int32_t main()
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 }

```