

# Competitive Programming Notebook

Programadores Roblox

## Contents

<b>1</b>	<b>DP</b>	<b>2</b>
1.1	Lis . . . . .	2
1.2	Lcs . . . . .	2
1.3	Knapsack . . . . .	2
<b>2</b>	<b>String</b>	<b>2</b>
<b>3</b>	<b>Geometry</b>	<b>2</b>
3.1	Point Location . . . . .	2
3.2	Convex Hull . . . . .	2
3.3	Lattice Points . . . . .	3
3.4	Inside Polygon . . . . .	3
<b>4</b>	<b>Graph</b>	<b>3</b>
4.1	Lca . . . . .	3
4.2	Kruskal . . . . .	4
4.3	Topological Sort . . . . .	4
4.4	Floyd Warshall . . . . .	4
4.5	Bellman Ford . . . . .	5
4.6	Dfs . . . . .	5
4.7	Dijkstra . . . . .	5
<b>5</b>	<b>Math</b>	<b>5</b>
5.1	Crivo . . . . .	5
5.2	Exgcd . . . . .	5
5.3	Fexp . . . . .	5
5.4	Equacao Diofantina . . . . .	5
<b>6</b>	<b>DS</b>	<b>6</b>
6.1	Ordered Set E Map . . . . .	6
6.2	Psum 2d . . . . .	6
6.3	Dsu . . . . .	6
6.4	Segtrees Sum . . . . .	6
6.5	Bit . . . . .	7
6.6	Segtrees Gcd . . . . .	7
<b>7</b>	<b>Search</b>	<b>8</b>
7.1	Dfs . . . . .	8
7.2	Bfs . . . . .	8
<b>8</b>	<b>Primitives</b>	<b>8</b>
<b>9</b>	<b>General</b>	<b>8</b>
9.1	Struct . . . . .	8
9.2	Bitwise . . . . .	8

# 1 DP

## 1.1 Lis

## 1.2 Lcs

## 1.3 Knapsack

```

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

```

# 2 String

# 3 Geometry

## 3.1 Point Location

```

1
2 int32_t main(){
3     sws;
4
5     int t; cin >> t;
6
7     while(t--){
8
9         int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1
10 >> x2 >> y2 >> x3 >> y3;
11
12         int deltax1 = (x1-x2), deltax1 = (y1-y2);
13
14         int compx = (x1-x3), compy = (y1-y3);
15
16         int ans = (deltax1*compy) - (compx*deltax1);
17
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     return 0;
23 }

```

## 3.2 Convex Hull

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {

```

```

9     cod x,y;
10 point(cod x = 0, cod y = 0): x(x), y(y)
11 {}
12
13 double modulo()
14 {
15     return sqrt(x*x + y*y);
16 }
17
18 point operator+(point o)
19 {
20     return point(x+o.x, y+o.y);
21 }
22 point operator-(point o)
23 {
24     return point(x - o.x, y - o.y);
25 }
26 point operator*(cod t)
27 {
28     return point(x*t, y*t);
29 }
30 point operator/(cod t)
31 {
32     return point(x/t, y/t);
33 }
34
35 cod operator*(point o)
36 {
37     return x*o.x + y*o.y;
38 }
39 cod operator^(point o)
40 {
41     return x*o.y - y * o.x;
42 }
43 bool operator<(point o)
44 {
45     if( x != o.x) return x < o.x;
46     return y < o.y;
47 }
48
49 };
50
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             // Ãl -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     {

```

```

80     while(U.size() >= 2 and ccw(U[U.size()-2], U
      .back(), pp) == -1)
81     {
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 }

```

### 3.3 Lattice Points

```

1 ll gcd(ll a, ll b) {
2     return b == 0 ? a : gcd(b, a % b);
3 }
4 ll area_trianguelo(ll x1, ll y1, ll x2, ll y2, ll x3,
      ll y3) {
5     return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 *
      (y1 - y2));
6 }
7 ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
8     return gcd(abs(x2 - x1), abs(y2 - y1));
9 }
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_trianguelo(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 }

```

### 3.4 Inside Polygon

```

1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4     int x = ccw(a, b, e);
5     int y = ccw(b, c, e);
6     int z = ccw(c, a, e);
7     return !((x==1 or y==1 or z==1) and (x==-1 or y
      ==-1 or z==-1));
8 }
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)
      ==0) return false;
22     // if(r==2 and ccw(p[0], p[1], e)==0) return
      false;
23     // 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
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
      [i], p[j], pp)==1)
42             inter++; // up
43         else if(p[j].x <= pp.x and pp.x < p[i].x and
      ccw(p[i], p[j], pp)==-1)
44             inter++; // down
45     }
46
47     if(inter%2==0) return -1; // outside
48     else return 1; // inside
49 }

```

## 4 Graph

### 4.1 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 at  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++;

```

## 4.2 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    void merge(int i, int j) {
25        if ((i = find(i)) == (j = find(j))) return
26        -1;

```

```

27        else --c;
28        if (rank[i] > rank[j]) swap(i, j);
29        par[i] = j;
30        sz[j] += sz[i];
31        if (rank[i] == rank[j]) rank[j]++;
32        return j;
33    }
34 };
35
36 struct Edge {
37     int u, v, w;
38     bool operator <(Edge const & other) {
39         return weight < other.weight;
40     }
41 }
42
43 vector<Edge> kruskal(int n, vector<Edge> edges) {
44     vector<Edge> mst;
45     DSU dsu = DSU(n + 1);
46     sort(edges.begin(), edges.end());
47     for (Edge e : edges) {
48         if (dsu.find(e.u) != dsu.find(e.v)) {
49             mst.push_back(e);
50             dsu.join(e.u, e.v);
51         }
52     }
53     return mst;
54 }

```

## 4.3 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[v] != 2) dfs(nei);
16    }
17    estado[v] = 2;
18    ordem.push_back(v);
19    return;

```

## 4.4 Floyd Warshall

```

1 // SSP e acha ciclos.
2 // Bom com constraints menores.
3 // O(n^3)
4
5 int dist[501][501];
6
7 void floydWarshall() {
8     for(int k = 0; k < n; k++) {
9         for(int i = 0; i < n; i++) {
10            for(int j = 0; j < n; j++) {
11                dist[i][j] = min(dist[i][j], dist[i][
12                k] + dist[k][j]);
13            }
14        }
15    }
16 }
17
18 void solve() {
19     int m, q;
20     cin >> n >> m >> q;

```

```

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 }

```

## 4.5 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     cycle.push_back(v);
30     if (v == x && ciclo.size() > 1) break;
31 }
32 reverse(ciclo.begin(), ciclo.end());

```

## 4.6 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 }

```

## 4.7 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 }

```

## 5 Math

### 5.1 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 }

```

### 5.2 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 }

```

### 5.3 Fexp

```

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

```

### 5.4 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 }

```

## 6 DS

### 6.1 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 }

```

### 6.2 Psum 2d

```

1 // entrada: matrix com ponto e X
2 // saber em O(1) quantidade de X em um retangulo
3
4 vector<vector<int>>> v(n+1, vector<int>(n+1, 0));
5
6 for (int i=1; i<n+1; i++){
7     for (int j=1; j<n+1; j++){
8         char x; cin >> x;
9         if (x == 'X') v[i][j] += 1 + v[i][j-1] + v[i-1][j] - v[i-1][j-1];
10        else v[i][j] = v[i][j-1] + v[i-1][j] - v[i-1][j-1];
11    }
12 }
13
14 // Pegar retângulo (x, y) - (z, w)
15 // ponto superior esquerdo e inferior direito
16
17 cin >> x >> y >> z >> w;
18 cout << v[z][w] - v[x-1][w] - v[z][y-1] + v[x-1][y-1]
19     << endl;

```

### 6.3 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 };

```

### 6.4 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;

```

## 6.5 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 public:
13     BIT(int size) {
14         n = size;
15         bit.assign(n + 1, 0); // BIT indexada em 1
16     }
17     void update(int idx, int delta) {
18         while (idx <= n) {
19             bit[idx] += delta;
20             idx += idx & -idx;
21         }
22     }
23     int query(int idx) {
24         return sum(idx);
25     }

```

```

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 }

```

## 6.6 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
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 public:
59     SegmentTreeGCD(const vector<int>& arr) {

```

```

52     n = arr.size();
53     tree.resize(4 * n);
54     build(arr, 0, 0, n - 1);
55 }
56 void update(int idx, int value) {
57     update(0, 0, n - 1, idx, value);
58 }
59 int query(int l, int r) {
60     return query(0, 0, n - 1, l, r);
61 }
62 };

```

## 7 Search

### 7.1 Dfs

```

1 // Printa os nÃss na ordem em que sÃo visitados
2 // Explora os nÃss em profundidade
3 // Complexidade: O(V+A) V = vÃrtices e A = arestas
4 // Espaço: 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     }

```

### 7.2 Bfs

```

1 // Printa os nÃss na ordem em que sÃo visitados
2 // Explora os nÃss em largura (camadas)
3 // Complexidade: O(V+A) V = vÃrtices e A = arestas
4 // Espaço: 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 nÃss atual
19
20         for(int vizinho: grafo[cur]){
21             if(visited.find(vizinho) == visited.end()
22 ){
23                 fila.push(vizinho);

```

```

23         visited.insert(vizinho)
24     }
25 }
26 }
27 }

```

## 8 Primitives

## 9 General

### 9.1 Struct

```

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

```

### 9.2 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 }

```