Competitive Programming Notebook

Programadores Roblox

Contents

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-
   items[i].first] + items[i].second);
14         }
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
   >> x2 >> y2 >> x3 >> y3;
10
11         int deltax1 = (x1-x2), deltay1 = (y1-y2);
12
13         int compx = (x1-x3), compy = (y1-y3);
14
15         int ans = (deltax1*compy) - (compx*deltay1);
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 }
```

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],
   L.back(), pp) == -1)
68         {
69             // Ã‰ -1 pq eu nÃ£o quero excluir os
   colineares
70             L.pop_back();
71         }
72         L.push_back(pp);
73     }
74
75     reverse(p.begin(), p.end());
76
77     //Upper
78     for(auto pp : p)
79     {
```

```cpp
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

```cpp
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,
       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_triangulo(x1, y1, x2, y2, x3, y3);
```

```cpp
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

```cpp
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
28 // Any O(n)
29
30 int inside(vp &p, point pp){
31     // 1 - inside / 0 - boundary / -1 - outside
32     int n = p.size();
33     for(int i=0;i<n;i++){
34         int j = (i+1)%n;
35         if(line({p[i], p[j]}).inside_seg(pp))
36             return 0;
37     }
38     int inter = 0;
39     for(int i=0;i<n;i++){
40         int j = (i+1)%n;
41         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
       [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]; //
        tanto que tenho que subir
8       for(int j = LOG-1; j >= 0; j--) {
9           if((1 << j) & k) {
10              a = cima[a][j];
11          }
12      }
13      if(a == b) return a; // ja to no lca
14
15      for(int j = LOG-1; j >= 0; j--) { // subo com os
        dois atÃľ chegar no lca fazendo binary lifting
16          if(cima[a][j] != cima[b][j]) {
17              a = cima[a][j];
18              b = cima[b][j];
19          }
20      }
21      return cima[a][0];
22  }
23
24  void dfs(int v, int p) {
25      if(v != 1) profundidade[v] = profundidade[p] + 1;
26      cima[v][0] = p;
27      for(int j = 1; j < LOG; j++) {
28          if (cima[v][j-1] != -1) {
29              cima[v][j] = cima[cima[v][j-1]][j-1];
30          } else {
31              cima[v][j] = -1;
32          }
33      }
34      for(auto &nei : adj[v]) {
35          if(nei != p) {
36              dfs(nei, v);
37          }
38      }
39  }
40
41  while((1 << LOG) <= n) LOG++;
```

4.2 Kruskal

```
1   // Ordena as arestas por peso,  insere se ja nao
        estiver no mesmo componente
2   // O(E log E)
3
4   struct DSU {
5       vector<int> par, rank, sz;
6       int c;
7       DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
        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  }
```

4.3 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;
```

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][
        k] + dist[k][j]);
12              }
13          }
14      }
15  }
16  void solve() {
17      int m, q;
18      cin >> n >> m >> q;
```

```cpp
    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            if(i == j) {
                dist[i][j] = dist[j][i] = 0;
            } else {
                dist[i][j] = dist[j][i] = linf;
            }
        }
    }
    for(int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w; u--; v--;
        dist[u][v] = min(dist[u][v], w);
        dist[v][u] = min(dist[v][u], w);
    }
    floydWarshall();
    while(q--) {
        int u, v;
        cin >> u >> v; u--; v--;
        if(dist[u][v] == linf) cout << -1 << '\n';
        else cout << dist[u][v] << '\n';
    }
}
```

4.5 Bellman Ford

```cpp
struct Edge {
    int u, v, w;
};

// se x = -1, nÃčo tem ciclo
// se x != -1, pegar pais de x pra formar o ciclo

int n, m;
vector<Edge> edges;
vector<int> dist(n);
vector<int> pai(n, -1);

    for (int i = 0; i < n; i++) {
        x = -1;
        for (Edge &e : edges) {
            if (dist[e.u] + e.w < dist[e.v]) {
                dist[e.v] = max(-INF, dist[e.u] + e.w);
                pai[e.v] = e.u;
                x = e.v;
            }
        }
    }

// achando caminho (se precisar)
for (int i = 0; i < n; i++) x = pai[x];

vector<int> ciclo;
for (int v = x;; v = pai[v]) {
    cycle.push_back(v);
    if (v == x && ciclo.size() > 1) break;
}
reverse(ciclo.begin(), ciclo.end());
```

4.6 Dfs

```cpp
int dfs(int x, int p) {
    for (auto e : adj[x]) {
        if (e != p) {
            dfs(e, x);
        }
    }
}
```

4.7 Dijkstra

```cpp
// SSP com pesos positivos.
// O((V + E) log V).

vector<int> dijkstra(int S) {
    vector<bool> vis(MAXN, 0);
    vector<ll> dist(MAXN, LLONG_MAX);
    dist[S] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, S});
    while(pq.size()) {
        ll v = pq.top().second;
        pq.pop();
        if(vis[v]) continue;
        vis[v] = 1;
        for(auto &[peso, vizinho] : adj[v]) {
            if(dist[vizinho] > dist[v] + peso) {
                dist[vizinho] = dist[v] + peso;
                pq.push({dist[vizinho], vizinho});
            }
        }
    }
    return dist;
}
```

5 Math

5.1 Crivo

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

5.2 Exgcd

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

5.3 Fexp

```cpp
// a^e mod m
// O(log n)

ll fexp(ll a, ll e, ll m) {
    a %= m;
    ll ans = 1;
    while (e > 0){
        if (e & 1) ans = ansa % m;
        a = aa % m;
        e /= 2;
    }
    return ans%m;
}
```

5.4 Equacao Diofantina

```cpp
// resolve equacao ax + by = c
// retorno {existe sol., x, y, g}
array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    auto[x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    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,
        null_type, less_equal<T>, rb_tree_tag,
        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 }
```

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

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
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 }
```

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
       start, int end) {
13         if (start == end) {
14             tree[node] = arr[start];
15         } else {
16             int mid = (start + end) / 2;
17             build(arr, 2 * node + 1, start, mid);
18             build(arr, 2 * node + 2, mid + 1, end);
19             tree[node] = gcd(tree[2 * node + 1], tree
       [2 * node + 2]);
20         }
21     }
22
23     void update(int node, int start, int end, int idx
       , int value) {
24         if (start == end) {
25             tree[node] = value;
26         } else {
27             int mid = (start + end) / 2;
28             if (idx <= mid) {
29                 update(2 * node + 1, start, mid, idx,
        value);
30             } else {
31                 update(2 * node + 2, mid + 1, end,
       idx, value);
32             }
33             tree[node] = gcd(tree[2 * node + 1], tree
       [2 * node + 2]);
34         }
35     }
36
37     int query(int node, int start, int end, int l,
       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
       , l, r);
46         int right_gcd = query(2 * node + 2, mid + 1,
       end, l, r);
47         return gcd(left_gcd, right_gcd);
48     }
49
50 public:
51     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 and sort

7.1   Dfs

```
1  // Printa os nos na ordem em que sÃčo 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 }
```

7.2   Bfs

```
1  // Printa os nos na ordem em que sÃčo 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 nÃş atual
19
20         for(int vizinho: grafo[cur]){
21             if(visited.find(vizinho) == visited.end()
   ){
22                 fila.push(vizinho);
```

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

7.3   Mergeandcount

```
1
2  // Realiza a mesclagem de dois subarrays e conta o
   nÃžmero de trocas necessÃąrias.
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 }
```

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
        .idade;
9          else return nome > other.nome;
10     }
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 }
```