

# Competitive Programming Notebook

Programadores Roblox

## Contents

<b>1</b>	<b>Math</b>	<b>2</b>
1.1	Fexp . . . . .	2
1.2	Crivo . . . . .	2
1.3	Equacao Diofantina . . . . .	2
1.4	Exgcd . . . . .	2
<b>2</b>	<b>DS</b>	<b>2</b>
2.1	Bit . . . . .	2
2.2	Psum 2d . . . . .	2
2.3	Segtree Sum . . . . .	2
2.4	Segtree Gcd . . . . .	3
2.5	Ordered Set E Map . . . . .	3
2.6	Dsu . . . . .	4
2.7	Segtree Iterativa . . . . .	4
<b>3</b>	<b>Search and sort</b>	<b>4</b>
3.1	Dfs . . . . .	4
3.2	Bfs . . . . .	4
3.3	Mergeandcount . . . . .	5
<b>4</b>	<b>Geometry</b>	<b>5</b>
4.1	Convex Hull . . . . .	5
4.2	Lattice Points . . . . .	6
4.3	Point Location . . . . .	6
4.4	Inside Polygon . . . . .	6
<b>5</b>	<b>String</b>	<b>7</b>
<b>6</b>	<b>Primitives</b>	<b>7</b>
<b>7</b>	<b>General</b>	<b>7</b>
7.1	Struct . . . . .	7
7.2	Bitwise . . . . .	7
<b>8</b>	<b>DP</b>	<b>7</b>
8.1	Lis . . . . .	7
8.2	Lcs . . . . .	7
8.3	Knapsack . . . . .	7
<b>9</b>	<b>Graph</b>	<b>8</b>
9.1	Bellman Ford . . . . .	8
9.2	Dijkstra . . . . .	8
9.3	Kruskal . . . . .	8
9.4	Dfs . . . . .	8
9.5	Lca . . . . .	8
9.6	Floyd Warshall . . . . .	9
9.7	Topological Sort . . . . .	9

# 1 Math

## 1.1 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 = ans*a % m;
9         a = a*a % m;
10        e /= 2;
11    }
12    return ans%m;
13 }
```

## 1.2 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 }
```

## 1.3 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 }
```

## 1.4 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 }
9 }
```

# 2 DS

## 2.1 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 }
```

## 2.2 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;
```

## 2.3 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;

```

## 2.4 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 };

```

## 2.5 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 }

```

## 2.6 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 };

```

## 2.7 Segtree Iterativa

```

1  struct SegTree {
2      int n;
3      vector<int> t;
4      int neutral_value = 0;
5
6      int combine(int a, int b) {
7          return a+b;
8      }
9

```

```

SegTree(const vector<int>& data) {
    n = data.size();
    t.resize(2 * n, neutral_value);

    for (int i = 0; i < n; i++)
        t[n + i] = data[i];

    for (int i = n - 1; i > 0; --i)
        t[i] = combine(t[i * 2], t[i * 2 + 1]);
}

int range_query(int l, int r) {
    int result = neutral_value;
    for (l += n, r += n + 1; l < r; l >>= 1, r
        >>= 1) {
        if (l & 1) result = combine(result, t[l
            ++]);
        if (r & 1) result = combine(result, t[--r
            ]);
    }
    return result;
}

void update(int pos, int new_val) {
    t[pos += n] = new_val;
    for (pos >>= 1; pos > 0; pos >>= 1)
        t[pos] = combine(t[2 * pos], t[2 * pos +
            1]);
}
};

```

## 3 Search and sort

### 3.1 Dfs

```

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

```

### 3.2 Bfs

```

1  // Printa os nos na ordem em que sãõ 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 n 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 }

```

### 3.3 Mergeandcount

```

1
2 // Realiza a mesclagem de dois subarrays e conta o
   nmero de trocas necessrias.
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 temporrios 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
19            igual, coloca no vetor original.
20            v[k++] = left[i++];
21        } else {
22            // Caso contrrio, coloca o elemento da
23            direita e conta as trocas.
24            v[k++] = right[j++];
25            swaps += (x - i);
26        }
27    }
28
29    // Adiciona os elementos restantes do subarray
   esquerdo (se houver).
30    while (i < x) v[k++] = left[i++];
31
32    // Adiciona os elementos restantes do subarray
   direito (se houver).
33    while (j < y) v[k++] = right[j++];
34
35    return swaps; // Retorna o numero total de
   trocas realizadas.
36 }
37
38 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 }

```

## 4 Geometry

### 4.1 Convex Hull

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {
9     cod x,y;
10    point(cod x = 0, cod y = 0): x(x), y(y)
11    {}
12
13    double modulo()
14    {
15        return sqrt(x*x + y*y);
16    }
17
18    point operator+(point o)
19    {
20        return point(x+o.x, y+o.y);
21    }
22
23    point operator-(point o)
24    {
25        return point(x - o.x , y - o.y);
26    }
27
28    point operator*(cod t)
29    {
30        return point(x*t, y*t);
31    }
32
33    point operator/(cod t)
34    {
35        return point(x/t, y/t);
36    }
37
38    cod operator*(point o)
39    {
40        return x*o.x + y*o.y;
41    }
42
43    cod operator^(point o)
44    {
45        return x*o.y - y * o.x;
46    }
47
48    bool operator<(point o)
49    {
50        if( x != o.x) return x < o.x;
51        return y < o.y;
52    }
53 }

```

```

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             // ÃŁ -1 pq eu nÃŁo quero excluir os
71             // colineares
72             L.pop_back();
73         }
74         L.push_back(pp);
75     }
76
77     reverse(p.begin(), p.end());
78
79     //Upper
80     for(auto pp : p)
81     {
82         while(U.size() >= 2 and ccw(U[U.size()-2], U
83             .back(), pp) == -1)
84         {
85             U.pop_back();
86         }
87         U.push_back(pp);
88     }
89
90     L.pop_back();
91     L.insert(L.end(), U.begin(), U.end()-1);
92     return L;
93 }
94
95 cod area(vector<point> v)
96 {
97     int ans = 0;
98     int aux = (int)v.size();
99     for(int i = 2; i < aux; i++)
100     {
101         ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
102     }
103     ans = abs(ans);
104     return ans;
105 }
106
107 int bound(point p1 , point p2)
108 {
109     return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
110 }
111
112 //teorema de pick [pontos = A - (bound+points)/2 + 1]
113
114 int32_t main()
115 {
116     int n;
117     cin >> n;
118
119     vector<point> v(n);
120     for(int i = 0; i < n; i++)
121     {

```

```

119         cin >> v[i].x >> v[i].y;
120     }
121
122     vector <point> ch = convex_hull(v);
123
124     cout << ch.size() << '\n';
125     for(auto p : ch) cout << p.x << " " << p.y << "\n
126     ";
127
128     return 0;
129 }

```

## 4.2 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
19     ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
20     ll tot_borda = pontos_borda(x1, y1, x2, y2) +
21     pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,
22     y3, x1, y1);
23
24     ll ans = (area - tot_borda) / 2 + 1;
25     cout << ans << endl;
26
27     return 0;
28 }

```

## 4.3 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         deltax3 = (x2-x3), deltay1 = (y1-y2),
14         deltay2 = (y1-y3), deltay3 = (y2-y3);
15
16         int compx = (deltax1*deltay2 - deltax2*deltay1);
17         int compy = (deltax1*deltay3 - deltax3*deltay1);
18
19         if(ans == 0){cout << "TOUCH\n"; continue;}
20         if(ans < 0){cout << "RIGHT\n"; continue;}
21         if(ans > 0){cout << "LEFT\n"; continue;}
22     }
23
24     return 0;
25 }

```

## 4.4 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
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 }

```

## 5 String

## 6 Primitives

## 7 General

### 7.1 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 }

```

### 7.2 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 }

```

## 8 DP

### 8.1 Lis

### 8.2 Lcs

### 8.3 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 }

```

## 9 Graph

### 9.1 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
19             pai[e.v] = e.u;
20             x = e.v;
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());

```

### 9.2 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;

```

### 9.3 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);
29        par[i] = j;
30        sz[j] += sz[i];
31        if (rank[i] == rank[j]) rank[j]++;
32        return j;
33    }
34 };
35 struct Edge {
36     int u, v, w;
37     bool operator <(Edge const & other) {
38         return weight < other.weight;
39     }
40 };
41 vector<Edge> kruskal(int n, vector<Edge> edges) {
42     vector<Edge> mst;
43     DSU dsu = DSU(n + 1);
44     sort(edges.begin(), edges.end());
45     for (Edge e : edges) {
46         if (dsu.find(e.u) != dsu.find(e.v)) {
47             mst.push_back(e);
48             dsu.join(e.u, e.v);
49         }
50     }
51     return mst;

```

### 9.4 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 }

```

### 9.5 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++;

```

## 9.6 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]
12                 + dist[k][j]);
13             }
14         }
15     }
16 void solve() {
17     int m, q;
18     cin >> n >> m >> q;
19     for(int i = 0; i < n; i++) {
20         for(int j = i; j < n; j++) {
21             if(i == j) {
22                 dist[i][j] = dist[j][i] = 0;
23             } else {
24                 dist[i][j] = dist[j][i] = linf;
25             }
26         }
27     }
28     for(int i = 0; i < m; i++) {
29         int u, v, w;
30         cin >> u >> v >> w; u--; v--;
31         dist[u][v] = min(dist[u][v], w);
32         dist[v][u] = min(dist[v][u], w);
33     }
34     floydWarshall();
35     while(q--) {
36         int u, v;
37         cin >> u >> v; u--; v--;
38         if(dist[u][v] == linf) cout << -1 << '\n';
39         else cout << dist[u][v] << '\n';
40     }
41 }

```

## 9.7 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;

```