

# Competitive Programming Notebook

Programadores Roblox

## Contents

<b>1 DS</b>	<b>2</b>	<b>7 String</b>	<b>11</b>
1.1 Psum 2d . . . . .	2	7.1 Hashing . . . . .	11
1.2 Segtree Gcd . . . . .	2	7.2 Lcs . . . . .	12
1.3 Segtree Sum . . . . .	2	7.3 Z Function . . . . .	12
1.4 Segtree Iterativa . . . . .	3	7.4 Trie . . . . .	12
1.5 Dsu . . . . .	3	7.5 Trie Ponteiros . . . . .	12
1.6 Ordered Set E Map . . . . .	3	7.6 Countpermutations . . . . .	13
1.7 Bit . . . . .	4	7.7 Kmp . . . . .	13
<b>2 Search and sort</b>	<b>4</b>	<b>8 General</b>	<b>13</b>
2.1 Mergeandcount . . . . .	4	8.1 Struct . . . . .	13
2.2 Dfs . . . . .	4	8.2 Bitwise . . . . .	13
2.3 Bfs . . . . .	5	<b>9 String copy</b>	<b>13</b>
<b>3 Primitives</b>	<b>5</b>	9.1 Hashing . . . . .	13
<b>4 Geometry</b>	<b>5</b>	9.2 Lcs . . . . .	14
4.1 Inside Polygon . . . . .	5	9.3 Z Function . . . . .	14
4.2 Convex Hull . . . . .	5	9.4 Trie Ponteiros . . . . .	14
4.3 Point Location . . . . .	6	9.5 Countpermutations . . . . .	14
4.4 Lattice Points . . . . .	6	9.6 Kmp . . . . .	15
<b>5 Math</b>	<b>7</b>	<b>10 DP</b>	<b>15</b>
5.1 Divisores . . . . .	7	10.1 Lcs . . . . .	15
5.2 Equacao Diofantina . . . . .	7	10.2 Lis . . . . .	15
5.3 Fexp . . . . .	7	10.3 Knapsack . . . . .	15
5.4 Segment Sieve . . . . .	7		
5.5 Discrete Log . . . . .	7		
5.6 Mod Inverse . . . . .	7		
5.7 Crivo . . . . .	7		
5.8 Exgcd . . . . .	8		
<b>6 Graph</b>	<b>8</b>		
6.1 Bellman Ford . . . . .	8		
6.2 Kruskal . . . . .	8		
6.3 Topological Sort . . . . .	8		
6.4 Floyd Warshall . . . . .	8		
6.5 Lca . . . . .	9		
6.6 Eulerian Path . . . . .	9		
6.7 Pega Ciclo . . . . .	10		
6.8 Kosaraju . . . . .	10		
6.9 Khan . . . . .	11		
6.10 Lca Jc . . . . .	11		
6.11 Dijkstra . . . . .	11		

# 1 DS

## 1.1 Psum 2d

```

1 vector<vector<int>> psum(h+1, vector<int>(w+1, 0));
2
3 for (int i=1; i<=h; i++){
4     for (int j=1; j<=w; j++){
5         cin >> psum[i][j];
6         psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[
7             i-1][j-1];
8     }
9
10 // retorna a psum2d do intervalo inclusivo [(a, b), (
11     c, d)]
12 int retangulo(int a, int b, int c, int d){
13     c = min(c, h), d = min(d, w);
14     a = max(0LL, a-1), b = max(0LL, b-1);
15
16     return v[c][d]-v[a][d]-v[c][b]+v[a][b];
17 }

```

## 1.2 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    }

```

```

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

```

## 1.3 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>
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     }

```

```

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;

```

## 1.4 Segtree Iterativa

```

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

```

```

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

```

## 1.5 Dsu

```

1 struct DSU {
2     vector<int> par, rank, sz;
3     int c;
4     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
5 1, 1), c(n) {
6         for (int i = 1; i <= n; ++i) par[i] = i;
7     }
8     int find(int i) {
9         return (par[i] == i ? i : (par[i] = find(par[
10 i])));
11    }
12    bool same(int i, int j) {
13        return find(i) == find(j);
14    }
15    int get_size(int i) {
16        return sz[find(i)];
17    }
18    int count() {
19        return c; // quantos componentes conexos
20    }
21    int merge(int i, int j) {
22        if ((i = find(i)) == (j = find(j))) return
23 -1;
24        else --c;
25        if (rank[i] > rank[j]) swap(i, j);
26        par[i] = j;
27        sz[j] += sz[i];
28        if (rank[i] == rank[j]) rank[j]++;
29        return j;
30    }
31 };

```

## 1.6 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 template <typename T> using o_set = tree<T, null_type,
11 less<T>, rb_tree_tag,
12 tree_order_statistics_node_update>;
13 template <typename T, typename R> using o_map = tree<
14 T, R, less<T>, rb_tree_tag,
15 tree_order_statistics_node_update>;
16

```

```

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 }

```

## 1.7 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    }

```

## 2 Search and sort

### 2.1 Mergeandcount

```

1
2 // Realiza a mesclagem de dois subarrays e conta o
    número de trocas necessárias.
3 int mergeAndCount(vector<int>& v, int l, int m, int r
    ) {
4     int x = m - 1 + 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 = 1;
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 }

```

### 2.2 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 }

```

## 2.3 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 // 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 nº 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 Primitives

## 4 Geometry

### 4.1 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 }

```

```

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
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 }

```

### 4.2 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 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             // ÃI -1 pq eu nÃo quero excluir os
70             // colineares
71             L.pop_back();
72         }
73         L.push_back(pp);
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 {

```

```

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
128     return 0;
129 }

```

### 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), deltay1 = (y1-y2);
13
14         int compx = (x1-x3), compy = (y1-y3);
15
16         int ans = (deltax1*compy) - (compx*deltay1);
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 }

```

### 4.4 Lattice Points

```

1 ll gcd(ll a, ll b) {
2     return b == 0 ? a : gcd(b, a % b);
3 }
4 ll area_triangulo(ll x1, ll y1, ll x2, ll y2, ll x3,
5     ll y3) {
6     return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 *
7     (y1 - y2));
8 }
9 ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
10     return gcd(abs(x2 - x1), abs(y2 - y1));
11 }
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);

```

```

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 }

```

## 5 Math

### 5.1 Divisores

```

1 // Retorna um vetor com os divisores de x
2 // eh preciso ter o crivo implementado
3 // 0(divisores)
4
5 vector<int> divs(int x){
6     vector<int> ans = {1};
7     vector<array<int, 2>> primos; // {primo, expoente
8
9     while (x > 1) {
10         int p = crivo[x], cnt = 0;
11         while (x % p == 0) cnt++, x /= p;
12         primos.push_back({p, cnt});
13     }
14
15     for (int i=0; i<primos.size(); i++){
16         int cur = 1, len = ans.size();
17
18         for (int j=0; j<primos[i][1]; j++){
19             cur *= primos[i][0];
20             for (int k=0; k<len; k++)
21                 ans.push_back(cur*ans[k]);
22         }
23     }
24
25     return ans;
26 }

```

### 5.2 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 }

```

### 5.3 Fexp

```

1 // a^e mod m
2 // 0(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 }

```

### 5.4 Segment Sieve

```

1 // Retorna quantos primos tem entre [l, r] (inclusivo
2 // precisa de um vetor com os primos atÃ sqrt(r)
3 int seg_sieve(int l, int r){
4     if (l > r) return 0;
5     vector<bool> is_prime(r - l + 1, true);
6     if (l == 1) is_prime[0] = false;
7
8     for (int p : primos){
9         if (p * p > r) break;
10        int start = max(p * p, (l + p - 1) / p * p);
11        for (int j = start; j <= r; j += p){
12            if (j >= l) {
13                is_prime[j - l] = false;
14            }
15        }
16    }
17
18    return accumulate(all(is_prime), 0ll);
19 }

```

### 5.5 Discrete Log

```

1 // Returns minimum x for which a^x = b (mod m), a and
2 // m are coprime.
3 // if the answer dont need to be greater than some
4 // value, the vector<int> can be removed
5 int discrete_log(int a, int b, int m) {
6     a %= m, b %= m;
7     int n = sqrt(m) + 1;
8
9     int an = 1;
10    for (int i = 0; i < n; ++i)
11        an = (an * 1ll * a) % m;
12
13    unordered_map<int, vector<int>> vals;
14    for (int q = 0, cur = b; q <= n; ++q) {
15        vals[cur].push_back(q);
16        cur = (cur * 1ll * a) % m;
17    }
18
19    int res = LLONG_MAX;
20
21    for (int p = 1, cur = 1; p <= n; ++p) {
22        cur = (cur * 1ll * an) % m;
23        if (vals.count(cur)) {
24            for (int q: vals[cur]){
25                int ans = n * p - q;
26                res = min(res, ans);
27            }
28        }
29    }
30
31    return res;
32 }

```

### 5.6 Mod Inverse

```

1 array<int, 2> extended_gcd(int a, int b) {
2     if (b == 0) return {1, 0};
3     auto [x, y] = extended_gcd(b, a % b);
4     return {y, x - (a / b) * y};
5 }
6
7 int mod_inverse(int a, int m) {
8     auto [x, y] = extended_gcd(a, m);
9     return (x % m + m) % m;
10 }

```

### 5.7 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 }

```

## 5.8 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 }

```

## 6 Graph

### 6.1 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             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());

```

### 6.2 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 + 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[
12            i])));
13    }
14    bool same(int i, int j) {
15        return find(i) == find(j);
16    }
17    int get_size(int i) {
18        return sz[find(i)];
19    }
20    int count() {
21        return c; // quantos componentes conexos
22    }
23    int merge(int i, int j) {
24        if ((i = find(i)) == (j = find(j))) return
25            -1;
26        else --c;
27        if (rank[i] > rank[j]) swap(i, j);
28        par[i] = j;
29        sz[j] += sz[i];
30        if (rank[i] == rank[j]) rank[j]++;
31        return j;
32    }
33 }
34 struct Edge {
35     int u, v, w;
36     bool operator <(Edge const & other) {
37         return weight < other.weight;
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 }

```

### 6.3 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;
20 }

```

### 6.4 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;
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 }

```

## 6.5 Lca

```

1 // LCA - CP algorithm
2 // preprocessing O(NlogN)
3 // lca O(logN)
4 // Uso: criar LCA com a quantidade de v rtices (n) e
5 // lista de adjac ncia (adj)
6 // chamar a fun  o preprocess com a raiz da  rvore
7 struct LCA {
8     int n, l, timer;
9     vector<vector<int>> adj;
10    vector<int> tin, tout;
11    vector<vector<int>> up;
12
13    LCA(int n, const vector<vector<int>>& adj) : n(n), adj(adj) {}
14
15    void dfs(int v, int p) {
16        tin[v] = ++timer;
17        up[v][0] = p;
18        for (int i = 1; i <= l; ++i)
19            up[v][i] = up[up[v][i-1]][i-1];
20
21        for (int u : adj[v]) {
22            if (u != p)
23                dfs(u, v);
24        }
25
26        tout[v] = ++timer;
27    }
28
29    bool is_ancestor(int u, int v) {

```

```

30        return tin[u] <= tin[v] && tout[u] >= tout[v];
31    }
32
33    int lca(int u, int v) {
34        if (is_ancestor(u, v))
35            return u;
36        if (is_ancestor(v, u))
37            return v;
38        for (int i = l; i >= 0; --i) {
39            if (!is_ancestor(up[u][i], v))
40                u = up[u][i];
41        }
42        return up[u][0];
43    }
44
45    void preprocess(int root) {
46        tin.resize(n);
47        tout.resize(n);
48        timer = 0;
49        l = ceil(log2(n));
50        up.assign(n, vector<int>(l + 1));
51        dfs(root, root);
52    }
53 };

```

## 6.6 Eulerian Path

```

1 /**
2  * Vers o que assume: #define int long long
3  *
4  * Retorna um caminho/ciclo euleriano em um grafo (se
5  * existir).
6  * - g: lista de adjac ncia (vector<vector<int>>).
7  * - directed: true se o grafo for dirigido.
8  * - s: v rtice inicial.
9  * - e: v rtice final (opcional). Se informado,
10    * tenta caminho de s at  e.
11    * - O(Nlog(N))
12    * Retorna vetor com a sequ ncia de v rtices, ou
13    * vazio se imposs vel.
14 */
15 vector<int> eulerian_path(const vector<vector<int>>&
16    g, bool directed, int s, int e = -1) {
17     int n = (int)g.size();
18     // c pia das adjac ncias em multiset para
19     // permitir remo  o espec fica
20     vector<multiset<int>> h(n);
21     vector<int> in_degree(n, 0);
22     vector<int> result;
23     stack<int> st;
24     // preencher h e indegrees
25     for (int u = 0; u < n; ++u) {
26         for (auto v : g[u]) {
27             ++in_degree[v];
28             h[u].emplace(v);
29         }
30     }
31     st.emplace(s);
32     if (e != -1) {
33         int out_s = (int)h[s].size();
34         int out_e = (int)h[e].size();
35         int diff_s = in_degree[s] - out_s;
36         int diff_e = in_degree[e] - out_e;
37         if (diff_s * diff_e != -1) return {}; //
38         imposs vel
39     }
40     for (int u = 0; u < n; ++u) {
41         if (e != -1 && (u == s || u == e)) continue;
42         int out_u = (int)h[u].size();
43         if (in_degree[u] != out_u || (!directed && (
44             in_degree[u] & 1))) {
45             return {};
46         }
47     }
48 }

```

```

39     }
40 }
41 while (!st.empty()) {
42     int u = st.top();
43     if (h[u].empty()) {
44         result.emplace_back(u);
45         st.pop();
46     } else {
47         int v = *h[u].begin();
48         auto it = h[u].find(v);
49         if (it != h[u].end()) h[u].erase(it);
50         --in_degree[v];
51         if (!directed) {
52             auto it2 = h[v].find(u);
53             if (it2 != h[v].end()) h[v].erase(it2);
54         }
55         --in_degree[u];
56         st.emplace(v);
57     }
58 }
59 for (int u = 0; u < n; ++u) {
60     if (in_degree[u] != 0) return {};
61 }
62 reverse(result.begin(), result.end());
63 return result;
64 }

```

## 6.7 Pega Ciclo

```

1 // encontra um ciclo em g (direcionado ou não)
2 // g[u] = vector<pair<id_aresta, vizinho>>
3 // rec_arestas: true -> retorna ids das arestas do
4 // ciclo; false -> retorna vÃrtices do ciclo
5 // directed: grafo direcionado?
6
7 const int MAXN = 5 * 1e5 + 2;
8 vector<pair<int, int>> g[MAXN];
9 int N;
10 bool DIRECTED = false;
11 vector<int> color(MAXN), parent(MAXN, -1), edgein(
12     MAXN, -1); // color: 0,1,2 ; edgein[v] = id da
13 // aresta que entra em v
14 int ini_ciclo = -1, fim_ciclo = -1, back_edge_id =
15     -1;
16
17 bool dfs(int u, int pai_edge){
18     color[u] = 1; // cinza
19     for (auto [id, v] : g[u]) {
20         if (!DIRECTED && id == pai_edge) continue; //
21         ignorar aresta de volta ao pai em não-dir
22         if (color[v] == 0) {
23             parent[v] = u;
24             edgein[v] = id;
25             if (dfs(v, id)) return true;
26         } else if (color[v] == 1) {
27             // back-edge u -> v detectado
28             ini_ciclo = u;
29             fim_ciclo = v;
30             back_edge_id = id;
31             return true;
32         }
33     }
34     // se color[v] == 2, ignora
35     color[u] = 2; // preto
36     return false;
37 }
38
39 // retorna ids das arestas do ciclo (vazio se não
40 // há)
41 vector<int> pega_ciclo(bool rec_arestas) {
42     for (int u = 1; u <= N; u++) {

```

```

38         if (color[u] != 0) continue;
39         if (dfs(u, -1)) {
40             // reconstrói caminho u -> ... -> v via
41             parent
42             vector<int> path;
43             int cur = ini_ciclo;
44             path.push_back(cur);
45             while (cur != fim_ciclo) {
46                 cur = parent[cur];
47                 path.push_back(cur);
48             }
49             // path = [u, ..., v] -> inverter para [v
50             // ..., u]
51             reverse(path.begin(), path.end());
52             if (!rec_arestas) return path;
53             // converte para ids das arestas: edgein[
54             // node] Ã a aresta que entra em node
55             vector<int> edges;
56             for (int i = 1; i < path.size(); i++)
57                 edges.push_back(edgein[path[i]]);
58             // adiciona a aresta de retorno u -> v
59             edges.push_back(back_edge_id);
60             return edges;
61         }
62     }
63     return {};
64 }

```

## 6.8 Kosaraju

```

1 bool vis[MAXN];
2 vector<int> order;
3 int component[MAXN];
4 int N, m;
5 vector<int> adj[MAXN], adj_rev[MAXN];
6
7 // dfs no grafo original para obter a ordem (pÃss-
8 // order)
9 void dfs1(int u) {
10     vis[u] = true;
11     for (int v : adj[u]) {
12         if (!vis[v]) {
13             dfs1(v);
14         }
15     }
16     order.push_back(u);
17 }
18
19 // dfs o grafo reverso para encontrar os SCCs
20 void dfs2(int u, int c) {
21     component[u] = c;
22     for (int v : adj_rev[u]) {
23         if (component[v] == -1) {
24             dfs2(v, c);
25         }
26     }
27 }
28
29 int kosaraju() {
30     order.clear();
31     fill(vis + 1, vis + N + 1, false);
32     for (int i = 1; i <= N; i++) {
33         if (!vis[i]) {
34             dfs1(i);
35         }
36     }
37     fill(component + 1, component + N + 1, -1);
38     int c = 0;
39     reverse(order.begin(), order.end());
40     for (int u : order) {
41         if (component[u] == -1) {
42             dfs2(u, c++);
43         }
44     }
45 }

```

```

43     }
44     return c;
45 }

```

## 6.9 Khan

```

1 // topo-sort DAG
2 // lexicograficamente menor.
3 // N: número de vértices (1-indexado)
4 // adj: lista de adjacência do grafo
5
6 const int MAXN = 5 * 1e5 + 2;
7 vector<int> adj[MAXN];
8 int N;
9
10 vector<int> kahn() {
11     vector<int> indegree(N + 1, 0);
12     for (int u = 1; u <= N; u++) {
13         for (int v : adj[u]) {
14             indegree[v]++;
15         }
16     }
17     priority_queue<int, vector<int>, greater<int>> pq;
18     for (int i = 1; i <= N; i++) {
19         if (indegree[i] == 0) {
20             pq.push(i);
21         }
22     }
23     vector<int> result;
24     while (!pq.empty()) {
25         int u = pq.top();
26         pq.pop();
27         result.push_back(u);
28         for (int v : adj[u]) {
29             indegree[v]--;
30             if (indegree[v] == 0) {
31                 pq.push(v);
32             }
33         }
34     }
35     if (result.size() != N) {
36         return {};
37     }
38     return result;
39 }

```

## 6.10 Lca Jc

```

1 int LOG;
2
3 int get_lca(int a, int b) {
4     if (profundidade[b] > profundidade[a]) {
5         swap(a, b);
6     }
7     int k = profundidade[a] - profundidade[b]; //
8     tanto que tenho que subir
9     for (int j = LOG-1; j >= 0; j--) {
10         if ((1 << j) & k) {
11             a = cima[a][j];
12         }
13     }
14     if (a == b) return a; // ja to no lca
15
16     for (int j = LOG-1; j >= 0; j--) { // subo com os
17         dois até 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 }

```

```

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++;

```

## 6.11 Dijkstra

```

1 // SSP com pesos positivos.
2 // O((V + E) log V).
3
4 vector<int> dijkstra(int S) {
5     vector<bool> vis(MAXN, 0);
6     vector<ll> dist(MAXN, LLONG_MAX);
7     dist[S] = 0;
8     priority_queue<pii, vector<pii>, greater<pii>> pq;
9     pq.push({0, S});
10    while (pq.size()) {
11        ll v = pq.top().second;
12        pq.pop();
13        if (vis[v]) continue;
14        vis[v] = 1;
15        for (auto &[peso, vizinho] : adj[v]) {
16            if (dist[vizinho] > dist[v] + peso) {
17                dist[vizinho] = dist[v] + peso;
18                pq.push({dist[vizinho], vizinho});
19            }
20        }
21    }
22    return dist;
23 }

```

## 7 String

### 7.1 Hashing

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
4 // from left to right - O(1)
5 // query_inv(l, r) from right to left - O(1)
6 // patrocinado por tiagodfs
7
8 struct Hash {
9     const int X = 2147483647;
10    const int MOD = 1e9+7;
11    int n; string s;
12    vector<int> h, hi, p;
13    Hash() {}
14    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
15        for (int i=0; i<n; i++) p[i] = (i ? X*p[i-1]:1)
16        % MOD;
17        for (int i=0; i<n; i++)
18            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
19    }
20 }

```

```

17     for (int i=n-1;i>=0;i--)
18         hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
19     % MOD;
20 }
21 int query(int l, int r) {
22     int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
23     0));
24     return hash < 0 ? hash + MOD : hash;
25 }
26 int query_inv(int l, int r) {
27     int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
28     +1] % MOD : 0));
29     return hash < 0 ? hash + MOD : hash;
30 }
31 };

```

## 7.2 Lcs

```

1 int lcs(string &s1, string &s2) {
2     int m = s1.size();
3     int n = s2.size();
4
5     vector<vector<int>> dp(m + 1, vector<int>(n + 1,
6     0));
7
8     for (int i = 1; i <= m; ++i) {
9         for (int j = 1; j <= n; ++j) {
10             if (s1[i - 1] == s2[j - 1])
11                 dp[i][j] = dp[i - 1][j - 1] + 1;
12             else
13                 dp[i][j] = max(dp[i - 1][j], dp[i][j
14             - 1]);
15         }
16     }
17     return dp[m][n];
18 }

```

## 7.3 Z Function

```

1 vector<int> z_function(string s) {
2     int n = s.size();
3     vector<int> z(n);
4     int l = 0, r = 0;
5     for(int i = 1; i < n; i++) {
6         if(i < r) {
7             z[i] = min(r - i, z[i - l]);
8         }
9         while(i + z[i] < n && s[z[i]] == s[i + z[i]])
10             z[i]++;
11         if(i + z[i] > r) {
12             l = i;
13             r = i + z[i];
14         }
15     }
16     return z;
17 }
18 }

```

## 7.4 Trie

```

1 // Trie por array
2 // Inserir e buscar prefixo em O(N)
3
4 int trie[MAXN][26];
5 int tot_nos = 0;
6 vector<bool> acaba(MAXN, false);
7 vector<int> contador(MAXN, 0);
8
9 void insere(string s) {
10     int no = 0;

```

```

11     for(auto &c : s) {
12         if(trie[no][c - 'a'] == 0) {
13             trie[no][c - 'a'] = ++tot_nos;
14         }
15         no = trie[no][c - 'a'];
16         contador[no]++;
17     }
18     acaba[no] = true;
19 }
20
21 bool busca(string s) {
22     int no = 0;
23     for(auto &c : s) {
24         if(trie[no][c - 'a'] == 0) {
25             return false;
26         }
27         no = trie[no][c - 'a'];
28     }
29     return acaba[no];
30 }
31
32 int isPref(string s) {
33     int no = 0;
34     for(auto &c : s) {
35         if(trie[no][c - 'a'] == 0){
36             return -1;
37         }
38         no = trie[no][c - 'a'];
39     }
40     return contador[no];
41 }

```

## 7.5 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserir e buscar prefixo em O(N)
3
4 struct Node {
5     Node *filhos[26] = {};
6     bool acaba = false;
7     int contador = 0;
8 };
9
10 void insere(string s, Node *raiz) {
11     Node *cur = raiz;
12     for(auto &c : s) {
13         cur->contador++;
14         if(cur->filhos[c - 'a'] != NULL) {
15             cur = cur->filhos[c - 'a'];
16             continue;
17         }
18         cur->filhos[c - 'a'] = new Node();
19         cur = cur->filhos[c - 'a'];
20     }
21     cur->contador++;
22     cur->acaba = true;
23 }
24
25 bool busca(string s, Node *raiz) {
26     Node *cur = raiz;
27     for(auto &c : s) {
28         if (cur->filhos[c - 'a'] != NULL) {
29             cur = cur->filhos[c - 'a'];
30             continue;
31         }
32         return false;
33     }
34     return cur->acaba;
35 }
36
37 // Retorna se o prefixo e quantas strings tem s como
38 // prefixo
39 int isPref(string s, Node *raiz) {

```

```

39 Node *cur = raiz;
40 for(auto &c : s) {
41     if (cur->filhos[c - 'a'] != NULL) {
42         cur = cur->filhos[c - 'a'];
43         continue;
44     }
45     return -1;
46 }
47 return cur->contador;
48 }

```

## 7.6 Countpermutations

```

1 // Returns the number of distinct permutations
2 // that are lexicographically less than the string t
3 // using the provided frequency (freq) of the
4 // characters
5 // 0(n*freq.size())
6 int countPermLess(vector<int> freq, const string &t)
7 {
8     int n = t.size();
9     int ans = 0;
10
11     vector<int> fact(n + 1, 1), invfact(n + 1, 1);
12     for (int i = 1; i <= n; i++)
13         fact[i] = (fact[i - 1] * i) % MOD;
14     invfact[n] = fexp(fact[n], MOD - 2, MOD);
15     for (int i = n - 1; i >= 0; i--)
16         invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
17
18     // For each position in t, try placing a letter
19     // smaller than t[i] that is in freq
20     for (int i = 0; i < n; i++) {
21         for (char c = 'a'; c < t[i]; c++) {
22             if (freq[c - 'a'] > 0) {
23                 freq[c - 'a']--;
24                 int ways = fact[n - i - 1];
25                 for (int f : freq)
26                     ways = (ways * invfact[f]) % MOD;
27                 ans = (ans + ways) % MOD;
28                 freq[c - 'a']++;
29             }
30         }
31         if (freq[t[i] - 'a'] == 0) break;
32         freq[t[i] - 'a']--;
33     }
34     return ans;
35 }

```

## 7.7 Kmp

```

1 vector<int> kmp(string s) {
2     int n = (int)s.length();
3     vector<int> p(n+1);
4     p[0] = -1;
5     for (int i = 1; i < n; i++) {
6         int j = p[i-1];
7         while (j >= 0 && s[j] != s[i-1])
8             j = p[j-1];
9         p[i] = j+1;
10    }
11    return p;
12 }

```

## 8 General

### 8.1 Struct

```

1 struct Pessoa{
2     // Atributos

```

```

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

```

## 8.2 Bitwise

```

1 int check_kth_bit(int x, int k) {
2     return (x >> k) & 1;
3 }
4
5 void print_on_bits(int x) {
6     for (int k = 0; k < 32; k++) {
7         if (check_kth_bit(x, k)) {
8             cout << k << ' ';
9         }
10    }
11    cout << '\n';
12 }
13
14 int count_on_bits(int x) {
15     int ans = 0;
16     for (int k = 0; k < 32; k++) {
17         if (check_kth_bit(x, k)) {
18             ans++;
19         }
20    }
21    return ans;
22 }
23
24 bool is_even(int x) {
25     return ((x & 1) == 0);
26 }
27
28 int set_kth_bit(int x, int k) {
29     return x | (1 << k);
30 }
31
32 int unset_kth_bit(int x, int k) {
33     return x & ~(1 << k);
34 }
35
36 int toggle_kth_bit(int x, int k) {
37     return x ^ (1 << k);
38 }
39
40 bool check_power_of_2(int x) {
41     return count_on_bits(x) == 1;
42 }

```

## 9 String copy

### 9.1 Hashing

```

1 // String Hash template
2 // constructor(s) - 0(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
4 // from left to right - 0(1)
5 // query_inv(l, r) from right to left - 0(1)
6 // patrocinado por tiagodfs
7
8 mt19937 rng(time(nullptr));
9
10 struct Hash {
11     const int X = rng();

```

```

11  const int MOD = 1e9+7;
12  int n; string s;
13  vector<int> h, hi, p;
14  Hash() {}
15  Hash(string s): s(s), n(s.size()), h(n), hi(n), p
(n) {
16      for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
% MOD;
17      for (int i=0;i<n;i++)
18          h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
19      for (int i=n-1;i>=0;i--)
20          hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
% MOD;
21  }
22  int query(int l, int r) {
23      int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
0));
24      return hash < 0 ? hash + MOD : hash;
25  }
26  int query_inv(int l, int r) {
27      int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
+1] % MOD : 0));
28      return hash < 0 ? hash + MOD : hash;
29  }
30 };

```

## 9.2 Lcs

```

1  int lcs(string &s1, string &s2) {
2      int m = s1.size();
3      int n = s2.size();
4
5      vector<vector<int>> dp(m + 1, vector<int>(n + 1,
0));
6
7      for (int i = 1; i <= m; ++i) {
8          for (int j = 1; j <= n; ++j) {
9              if (s1[i - 1] == s2[j - 1])
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11              else
12                 dp[i][j] = max(dp[i - 1][j], dp[i][j
- 1]);
13          }
14      }
15
16      return dp[m][n];
17  }

```

## 9.3 Z Function

```

1  vector<int> z_function(string s) {
2      int n = s.size();
3      vector<int> z(n);
4      int l = 0, r = 0;
5      for(int i = 1; i < n; i++) {
6          if(i < r) {
7              z[i] = min(r - i, z[i - 1]);
8          }
9          while(i + z[i] < n && s[z[i]] == s[i + z[i]])
10             z[i]++;
11      }
12      if(i + z[i] > r) {
13          l = i;
14          r = i + z[i];
15      }
16  }
17  return z;
18 }

```

## 9.4 Trie Ponteiros

```

1  // Trie por ponteiros
2  // Inserção, busca e consulta de prefixo em O(N)
3
4  struct Node {
5      Node *filhos[26] = {};
6      bool acaba = false;
7      int contador = 0;
8  };
9
10 void insere(string s, Node *raiz) {
11     Node *cur = raiz;
12     for(auto &c : s) {
13         cur->contador++;
14         if(cur->filhos[c - 'a'] != NULL) {
15             cur = cur->filhos[c - 'a'];
16             continue;
17         }
18         cur->filhos[c - 'a'] = new Node();
19         cur = cur->filhos[c - 'a'];
20     }
21     cur->contador++;
22     cur->acaba = true;
23 }
24
25 bool busca(string s, Node *raiz) {
26     Node *cur = raiz;
27     for(auto &c : s) {
28         if (cur->filhos[c - 'a'] != NULL) {
29             cur = cur->filhos[c - 'a'];
30             continue;
31         }
32         return false;
33     }
34     return cur->acaba;
35 }
36
37 // Retorna se Ãl prefixo e quantas strings tem s como
prefixo
38 int isPref(string s, Node *raiz) {
39     Node *cur = raiz;
40     for(auto &c : s) {
41         if (cur->filhos[c - 'a'] != NULL) {
42             cur = cur->filhos[c - 'a'];
43             continue;
44         }
45         return -1;
46     }
47     return cur->contador;
48 }

```

## 9.5 Countpermutations

```

1  // Returns the number of distinct permutations
2  // that are lexicographically less than the string t
3  // using the provided frequency (freq) of the
characters
4  // O(n*freq.size())
5  int countPermLess(vector<int> freq, const string &t)
{
6      int n = t.size();
7      int ans = 0;
8
9      vector<int> fact(n + 1, 1), invfact(n + 1, 1);
10     for (int i = 1; i <= n; i++)
11         fact[i] = (fact[i - 1] * i) % MOD;
12     invfact[n] = fexp(fact[n], MOD - 2, MOD);
13     for (int i = n - 1; i >= 0; i--)
14         invfact[i] = (invfact[i + 1] * (i + 1)) % MOD
;
15
16     // For each position in t, try placing a letter
smaller than t[i] that is in freq
17     for (int i = 0; i < n; i++) {

```

```

18     for (char c = 'a'; c < t[i]; c++) {
19         if (freq[c - 'a'] > 0) {
20             freq[c - 'a']--;
21             int ways = fact[n - i - 1];
22             for (int f : freq)
23                 ways = (ways * invfact[f]) % MOD;
24             ans = (ans + ways) % MOD;
25             freq[c - 'a']++;
26         }
27     }
28     if (freq[t[i] - 'a'] == 0) break;
29     freq[t[i] - 'a']--;
30 }
31 return ans;
32 }

```

## 9.6 Kmp

```

1 vector<int> kmp(string s) {
2     int n = (int)s.length();
3     vector<int> p(n+1);
4     p[0] = -1;
5     for (int i = 1; i < n; i++) {
6         int j = p[i-1];
7         while (j >= 0 && s[j] != s[i-1])
8             j = p[j-1];
9         p[i] = j+1;
10    }
11    return p;
12 }

```

## 10 DP

### 10.1 Lcs

### 10.2 Lis

### 10.3 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 }

```