

Competitive Programming Notebook

Programadores Roblox

Contents

1 String	2	6.7 Acha Pontes	10
1.1 Trie	2	6.8 Edmonds-karp	10
1.2 Hashing	2	6.9 Kruskal	10
1.3 Z Function	2	6.10 Bellman Ford	10
1.4 Kmp	2	6.11 Lca Jc	11
2 DS	2	6.12 Lca	11
2.1 Seg Lazy Pa	2	6.13 Kosaraju	11
2.2 Segtree Iterativa	3	6.14 Min Cost Max Flow	12
2.3 Merge Sort Tree	3	6.15 Diametro	12
2.4 Ordered Set E Map	3	7 Primitives	12
2.5 Sparse Table	4	8 DP	12
2.6 Psum 2d	4	8.1 Lis	12
2.7 Segtree Lazy	4	8.2 Edit Distance	13
2.8 Min Queue	4	8.3 Bitmask	13
2.9 Dsu	4	8.4 Lcs	13
2.10 Bit	5	8.5 Digit	13
3 Search and sort	5	8.6 Knapsack	13
3.1 Merge Sort	5	8.7 Lis Seg	13
3.2 Monotonic Stack	5	8.8 Disjoint Blocks	13
4 Stress	5	9 General	14
4.1 Gen	5	9.1 Brute Choose	14
5 Math	5	9.2 Struct	14
5.1 Combinatorics	5	9.3 Mex	14
5.2 Equacao Diofantina	6	9.4 Count Permutations	14
5.3 Multiplicacao Matriz	6	9.5 Bitwise	14
5.4 Discrete Log	6	10 Geometry	15
5.5 Segment Sieve	6	10.1 Convex Hull	15
5.6 Totient	6	10.2 Inside Polygon	15
5.7 Menor Fator Primo	7	10.3 Point Location	16
5.8 Exgcd	7	10.4 Lattice Points	16
5.9 Fexp	7		
5.10 Divisores	7		
5.11 Crivo	7		
5.12 Mod Inverse	7		
5.13 Base Calc	7		
5.14 Fft	8		
6 Graph	8		
6.1 Dijkstra	8		
6.2 Floyd Warshall	8		
6.3 Eulerian Path	8		
6.4 Dinitz	9		
6.5 Khan	9		
6.6 Topological Sort	10		

1 String

1.1 Trie

```

1 // Trie por array
2 int trie[MAXN][26];
3 int tot_nos = 0;
4 vector<bool> acaba(MAXN, false);
5 vector<int> contador(MAXN, 0);
6
7 void insere(string s) {
8     int no = 0;
9     for(auto &c : s) {
10         if(trie[no][c - 'a'] == 0) {
11             trie[no][c - 'a'] = ++tot_nos;
12         }
13         no = trie[no][c - 'a'];
14         contador[no]++;
15     }
16     acaba[no] = true;
17 }
18
19 bool busca(string s) {
20     int no = 0;
21     for(auto &c : s) {
22         if(trie[no][c - 'a'] == 0) {
23             return false;
24         }
25         no = trie[no][c - 'a'];
26     }
27     return acaba[no];
28 }
29
30 int isPref(string s) {
31     int no = 0;
32     for(auto &c : s) {
33         if(trie[no][c - 'a'] == 0) {
34             return -1;
35         }
36         no = trie[no][c - 'a'];
37     }
38     return contador[no];
39 }
```

1.2 Hashing

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r] from
4 // left to right - O(1)
5 // query_inv(l, r) from right to left - O(1)
6 // patrocinado por tiagodfs
7 mt19937 rng(time(nullptr));
8
9 struct Hash {
10     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
28 : 0));
29         return hash < 0 ? hash + MOD : hash;
30     }
}
```

1.3 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 }
```

1.4 Kmp

```

1 vector<int> kmp(string &s) {
2     int m = s.size();
3     vector<int> lsp(m, 0);
4     for (int i = 1, j = 0; i < m; i++) {
5         while (j > 0 && s[j] != s[i]) j = lsp[j - 1];
6         if (s[i] == s[j]) j++;
7         lsp[i] = j;
8     }
9     return lsp;
10 }
```

2 DS

2.1 Seg Lazy Pa

```

1 /* Notas
2  PA eh da forma a0 (a) e razÄo (d)
3  na hora de propagar o lazy
4  aplica no no S = (a0 + (a0 + (n - 1) * d)) * n / 2 ( variante da soma total do no)
5  pros filhos, o da esquerda eh normal
6  lazy_a[esq] += lazy_a[x]
7  lazy_d[esq] += lazy_d[x]
8  pro da direita tem que mudar o a (que eh o elemento inicial
naquele no da direita)
9  lazy_a[dir] += a + len_esq * (d)
10 lazy_d[dir] += lazy_d[x]
11 */
12
13 int gauss(int n, int a, int d) {
14     // (a0 + an) * n / 2
15     if (n <= 0)
16         return 0;
17     return (a + (a + (n - 1) * d)) * n / 2;
18 }
19
20 struct SegTree {
21     int n;
22     vector<int> v, lazy_a, lazy_d, tree;
23     SegTree(vector<int> &a) : v(a), n(a.size()) {
24         tree.resize(4 * n);
25         lazy_a.resize(4 * n, 0);
26         lazy_d.resize(4 * n, 0);
27         build(1, 0, n - 1);
28     }
29     void build(int x, int lx, int rx) {
30         if (lx == rx) {
31             tree[x] = v[lx];
32             return;
33         }
34         int mid = (lx + rx) / 2;
35         build(2 * x, lx, mid);
36         build(2 * x + 1, mid + 1, rx);
37         tree[x] = tree[2 * x] + tree[2 * x + 1];
38     }
39     int query(int x, int lx, int rx, int l, int r) {
40         push(x, lx, rx);
41         if (lx >= l && rx <= r)
42             return tree[x];
43         if (lx > r || rx < l)
44             return 0;
45         int mid = (lx + rx) / 2;
46         return query(2 * x, lx, mid, l, r) + query(2 * x + 1, mid
+ 1, rx, l, r);
```

```

47 }
48 int query(int l, int r) {
49   return query(1, 0, n - 1, l, r);
50 }
51 void push(int x, int lx, int rx) {
52   if (lazy_a[x] && lazy_d[x]) {
53     int tam = rx - lx + 1;
54     tree[x] += gauss(tam, lazy_a[x], lazy_d[x]);
55     if (lx != rx) {
56       int mid = (lx + rx) / 2;
57       int tam_esq = mid - lx + 1;
58       lazy_a[2 * x] += lazy_a[x];
59       lazy_d[2 * x] += lazy_d[x];
60       lazy_a[2 * x + 1] += (lazy_a[x] + tam_esq * lazy_d[x]
61     });
62     lazy_d[2 * x + 1] += lazy_d[x];
63   }
64   lazy_a[x] = lazy_d[x] = 0;
65 }
66 void update(int x, int lx, int rx, int l, int r, int a, int d) {
67   push(x, lx, rx);
68   if (lx >= l && rx <= r) {
69     lazy_a[x] += a + (lx - 1) * d;
70     lazy_d[x] += d;
71     push(x, lx, rx);
72   }
73   if (lx > r || rx < l)
74     return;
75   int mid = (lx + rx) / 2;
76   update(2 * x, lx, mid, l, r, a, d);
77   update(2 * x + 1, mid + 1, rx, l, r, a, d);
78   tree[x] = tree[2 * x] + tree[2 * x + 1];
79 }
80 void update(int l, int r, int a, int d) {
81   update(1, 0, n - 1, l, r, a, d);
82 }
83 }

```

2.2 Segtree Iterativa

```

1 // Exemplo de uso:
2 // SegTree<int> st(vetor);
3 // range_query e point_update
4 template <typename T>
5 struct SegTree {
6   int n;
7   vector<T> tree;
8   T neutral_value = 0;
9   T combine(T a, T b) {
10     return a + b;
11   }
12
13 SegTree(const vector<T>& data) {
14   n = data.size();
15   tree.resize(2 * n, neutral_value);
16
17   for (int i = 0; i < n; i++)
18     tree[n + i] = data[i];
19
20   for (int i = n - 1; i > 0; --i)
21     tree[i] = combine(tree[i * 2], tree[i * 2 + 1]);
22 }
23 T range_query(int l, int r) {
24   T res_l = neutral_value, res_r = neutral_value;
25
26   for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
27     if (l & 1) res_l = combine(res_l, tree[l++]);
28     if (r & 1) res_r = combine(tree[-r], res_r);
29   }
30
31   return combine(res_l, res_r);
32 }
33 void update(int pos, T new_val) {
34   tree[pos += n] = new_val;
35   for (pos >>= 1; pos > 0; pos >>= 1)
36     tree[pos] = combine(tree[2 * pos], tree[2 * pos +
37   1]);
38 }

```

2.3 Merge Sort Tree

```

1 #define all(x) x.begin(), x.end()
2
3 struct MST {
4   int n;
5   vector<vector<int>> tree;
6   MST(vector<int> &a) {
7     n = a.size();
8     tree.resize(4 * n);
9     build(1, 0, n - 1, a);
10 }
11 void build(int x, int lx, int rx, vector<int> &a) {
12   if (lx == rx) {
13     tree[x] = {a[lx]};
14     return;
15   }
16   int mid = lx + (rx - lx) / 2;
17   build(2 * x, lx, mid, a);
18   build(2 * x + 1, mid + 1, rx, a);
19   auto &L = tree[2 * x], &R = tree[2 * x + 1];
20   tree[x].resize(L.size() + R.size());
21   merge(all(L), all(R), tree[x].begin());
22 }
23 int query(int x, int lx, int rx, int l, int r, int val) {
24   if (lx > r || rx < l) return 0;
25   if (lx >= l && rx <= r) {
26     auto &v = tree[x];
27     return lower_bound(all(v), val) - v.begin();
28   }
29   int mid = lx + (rx - lx) / 2;
30   return query(2 * x, lx, mid, l, r, val) + query(2 * x +
31   1, mid + 1, rx, l, r, val);
32 }
33 int query(int l, int r, int val) {
34   if (l > r) return 0;
35   return query(1, 0, n - 1, l, r, val);
36 }
37
38 /* mst.query(l, r, 1) retorna quantos caras distintos no
range
39 map<int, int> last;
40 for (int i = 0; i < n; i++) {
41   if (last.count(a[i])) {
42     esq[i] = last[a[i]];
43   } else {
44     esq[i] = -1;
45   }
46   last[a[i]] = i;
47 }
48 MST mst(esq);
49 jogar vetor de ultima aparicao na seg
50 */

```

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

```

24 }

2.5 Sparse Table

```

1 // 0-index, O(1)
2 struct SparseTable {
3     vector<vector<int>> st;
4     int max_log;
5     SparseTable(vector<int>& arr) {
6         int n = arr.size();
7         max_log = floor(log2(n)) + 1;
8         st.resize(n, vector<int>(max_log));
9         for (int i = 0; i < n; i++) {
10             st[i][0] = arr[i];
11         }
12         for (int j = 1; j < max_log; j++) {
13             for (int i = 0; i + (1 << j) <= n; i++) {
14                 st[i][j] = max(st[i][j - 1], st[i + (1 << (j
15                     - 1))][j - 1]);
16             }
17         }
18     }
19     int query(int L, int R) {
20         int tamanho = R - L + 1;
21         int k = floor(log2(tamanho));
22         return max(st[L][k], st[R - (1 << k) + 1][k]);
23     }

```

2.6 Psum 2d

```

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

```

2.7 Segtree Lazy

```

1 /*
2 Seg com double Lazy de soma e de set
3 tipo 1 - soma em range
4 tipo 2 - set em range
5 */
6 struct SegTree {
7     int n;
8     vector<int> v, tree;
9     vector<int> lazy_soma, lazy_set;
10    SegTree(vector<int> &a) : v(a), n(a.size()) {
11        tree.resize(4 * n);
12        lazy_soma.resize(4 * n, 0);
13        lazy_set.resize(4 * n, -1);
14        build(1, 0, n - 1);
15    };
16    void build(int x, int lx, int rx) {
17        if (lx == rx) {
18            tree[x] = v[lx];
19            return;
20        }
21        int mid = (lx + rx) / 2;
22        build(2 * x, lx, mid);
23        build(2 * x + 1, mid + 1, rx);
24        tree[x] = tree[2 * x] + tree[2 * x + 1];
25    }
26    void seta(int x, int lx, int rx, int val) {
27        tree[x] = val * (rx - lx + 1);
28        lazy_set[x] = val;
29        lazy_soma[x] = 0;
30    }
31    void soma(int x, int lx, int rx, int val) {
32        tree[x] += val * (rx - lx + 1);
33        if (lazy_set[x] != -1) {
34            lazy_set[x] += val;
35        } else {

```

```

36            lazy_soma[x] += val;
37        }
38    }
39    void push(int x, int lx, int rx) {
40        int mid = (lx + rx) / 2;
41        if (lazy_set[x] != -1) {
42            if (lx != rx) {
43                seta(2 * x, lx, mid, lazy_set[x]);
44                seta(2 * x + 1, mid + 1, rx, lazy_set[x]);
45            }
46            lazy_set[x] = -1;
47        }
48        if (lazy_soma[x] != 0) {
49            if (lx != rx) {
50                soma(2 * x, lx, mid, lazy_soma[x]);
51                soma(2 * x + 1, mid + 1, rx, lazy_soma[x]);
52            }
53            lazy_soma[x] = 0;
54        }
55    }
56    void update(int x, int lx, int rx, int l, int r, int val,
57                int type) {
58        push(x, lx, rx);
59        if (lx >= l && rx <= r) {
60            if (type == 1) soma(x, lx, rx, val);
61            else seta(x, lx, rx, val);
62            push(x, lx, rx);
63            return;
64        }
65        if (lx > r || rx < l) return;
66        int mid = (lx + rx) / 2;
67        update(2 * x, lx, mid, l, r, val, type);
68        update(2 * x + 1, mid + 1, rx, l, r, val, type);
69        tree[x] = tree[2 * x] + tree[2 * x + 1];
70    }
71    void update(int l, int r, int val, int type) {
72        update(1, 0, n - 1, l, r, val, type);
73    }
74    int query(int x, int lx, int rx, int l, int r) {
75        push(x, lx, rx);
76        if (lx >= l && rx <= r) return tree[x];
77        if (lx > r || rx < l) return 0;
78        int mid = (lx + rx) / 2;
79        return query(2 * x, lx, mid, l, r) + query(2 * x + 1, mid
80            + 1, rx, l, r);
81    }
82    int query(int l, int r) {
83        return query(1, 0, n - 1, l, r);
84    }
85 }

```

2.8 Min Queue

```

1 struct MinQueue {
2     deque<pair<int, int>> dq;
3     void push(int val, int idx) {
4         while (!dq.empty() && dq.back().second >= val) dq.
5         pop_back();
6         dq.emplace_back(val, idx);
7     }
8     void pop(int idx) {
9         if (!dq.empty() && dq.front().second == idx) {
10            dq.pop_front();
11        }
12        int get() {
13            return dq.front().first;
14        }
15        bool empty() {
16            return dq.empty();
17        }
18    };
19 /*
20  considerando janela de tamanho K
21  mq.push(v[i], i);
22  if (i >= k) {
23      mq.pop(i - k);
24  }
25  if (i >= k - 1) {
26      result.push_back(mq.get());
27  }
28 */

```

2.9 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 };

```

2.10 Bit

```

1 struct BIT {
2   int n;
3   vector<int> bit;
4   BIT(int n = 0): n(n), bit(n + 1, 0) {}
5   void add(int i, int delta) {
6     for(; i <= n; i += i & -i) bit[i] += delta;
7   }
8   int sum(int i) {
9     int r = 0;
10    for(; i > 0; i -= i & -i) r += bit[i];
11    return r;
12  }
13  int range_sum(int l, int r) {
14    if (r < l) return 0;
15    return sum(r) - sum(l - 1);
16  }
17 };

```

3 Search and sort

3.1 Merge Sort

```

1 int mergeAndCount(vector<int>& v, int l, int m, int r) {
2   int x = m - l + 1;
3   int y = r - m;
4   vector<int> left(x), right(y);
5   for (int i = 0; i < x; i++) left[i] = v[l + i];
6   for (int j = 0; j < y; j++) right[j] = v[m + 1 + j];
7   int i = 0, j = 0, k = l;
8   int swaps = 0;
9   while (i < x && j < y) {
10     if (left[i] <= right[j]) {
11       v[k++] = left[i++];
12     } else {
13       v[k++] = right[j++];
14       swaps += (x - i);
15     }
16   }
17   while (i < x) v[k++] = left[i++];
18   while (j < y) v[k++] = right[j++];
19   return swaps;
20 }
21
22 int mergeSort(vector<int>& v, int l, int r) {
23   int swaps = 0;
24   if (l < r) {
25     int m = l + (r - 1) / 2;
26     swaps += mergeSort(v, l, m);
27     swaps += mergeSort(v, m + 1, r);
28     swaps += mergeAndCount(v, l, m, r);
29   }

```

3.2 Monotonic Stack

```

30   return swaps;
31 }

1 vector<int> find_esq(vector<int> &v, bool maior) {
2   int n = v.size();
3   vector<int> result(n);
4   stack<int> s;
5   for (int i = 0; i < n; i++) {
6     while (!s.empty() && (maior ? v[s.top()] <= v[i] : v[s.top()] >= v[i])) {
7       s.pop();
8     }
9     if (s.empty()) {
10       result[i] = -1;
11     } else {
12       result[i] = v[s.top()];
13     }
14     s.push(i);
15   }
16   return result;
17 }

20 vector<int> find_dir(vector<int> &v, bool maior) {
21   int n = v.size();
22   vector<int> result(n);
23   stack<int> s;
24   for (int i = n - 1; i >= 0; i--) {
25     while (!s.empty() && (maior ? v[s.top()] <= v[i] : v[s.top()] >= v[i])) {
26       s.pop();
27     }
28     if (s.empty()) {
29       result[i] = -1;
30     } else {
31       result[i] = v[s.top()];
32     }
33     s.push(i);
34   }
35   return result;
36 }

```

4 Stress

4.1 Gen

```

1 // pre-compilar os headers:
2 // compilar template com g++ -H e procurar onde esta stdc++.h
3 // criar a pasta bits e incluir com "" ou inves de <>
4 // compilar stress com: g++ -pipe -O3 -fno -march=native -mtune=native gen.ccp
5 // faz bastante diferen a no runtime
6
7 #include <bits/stdc++.h>
8 #include <cstdlib>
9 #include <ctime>
10 using namespace std;
11
12 int randi(int L, int R) { return L + rand() % (R - L + 1); }
13 char randc(char L, char R) { return char(L + rand() % (R - L + 1)); }
14
15 int main(int argc, char** argv) {
16   if (argc > 1) srand(atoi(argv[1]));
17   else srand(time(0));
18
19   int n = randi(1, 100);
20   cout << n << '\n';
21   for (int i = 0; i < n; i++) {
22     cout << randi(-10, 20) << ' ';
23   }
24   cout << '\n';
25 }

```

5 Math

5.1 Combinatorics

```

1 const int MAXN_FATORIAL = 200005;
2 const int MOD = 1e9 + 7;
3 // DEFINE INT LONG LONG PLMDS
4 int fat[MAXN_FATORIAL], fati[MAXN_FATORIAL];
5
6 // (a^b) % m em O(log b)
7 // coloque o fexp
8
9 int inv(int n) { return fexp(n, MOD - 2); }
10
11 void precalc() {
12     fat[0] = 1;
13     fati[0] = 1;
14     for (int i = 1; i < MAXN_FATORIAL; i++) fat[i] = (fat[i - 1] * i) % MOD;
15     fati[MAXN_FATORIAL - 1] = inv(fat[MAXN_FATORIAL - 1]);
16     for (int i = MAXN_FATORIAL - 2; i >= 0; i--) fati[i] = (fati[i + 1] * (i + 1)) % MOD;
17 }
18
19 int choose(int n, int k) {
20     if (k < 0 || k > n) return 0;
21     return (((fat[n] * fati[k]) % MOD) * fati[n - k]) % MOD;
22 }
23
24 // n! / (n-k)!
25 int perm(int n, int k) {
26     if (k < 0 || k > n) return 0;
27     return (fat[n] * fati[n - k]) % MOD;
28 }
29
30 // C_n = (1 / (n+1)) * C(2n, n)
31 int catalan(int n) {
32     if (n < 0 || 2 * n >= MAXN_FATORIAL) return 0;
33     int c2n_n = choose(2 * n, n);
34     return (c2n_n * inv(n + 1)) % MOD;
35 }

```

5.2 Equacao Diofantina

```

1 int extended_gcd(int a, int b, int& x, int& y) {
2     if (a == 0) {
3         x = 0;
4         y = 1;
5         return b;
6     }
7     int x1, y1;
8     int gcd = extended_gcd(b % a, a, x1, y1);
9     x = y1 - (b / a) * x1;
10    y = x1;
11    return gcd;
12 }
13
14 bool solve(int a, int b, int c, int& x0, int& y0) {
15     int x, y;
16     int g = extended_gcd(abs(a), abs(b), x, y);
17     if (c % g != 0) {
18         return false;
19     }
20     x0 = x * (c / g);
21     y0 = y * (c / g);
22     if (a < 0) x0 = -x0;
23     if (b < 0) y0 = -y0;
24     return true;
25 }

```

5.3 Multiplicacao Matriz

```

1 // multiplica matrizes de tamanhos variados, resultando em
2 // uma matriz N*M
3 vector<vector<int>> mm(vector<vector<int>> A, vector<vector<int>> B) {
4     int N = A.size(), M = B[0].size(), K = B.size();
5     vector<vector<int>> C(N, vector<int>(M));
6
7     for (int i = 0; i < N; ++i)
8         for (int j = 0; j < M; ++j)
9             for (int k = 0; k < K; ++k)
10                 C[i][j] = (C[i][j] + A[i][k] * B[k][j] % mod) %
11                     mod;
12
13     return C;
14 }

```

5.4 Discrete Log

```

1 // returns minimum x for which a^x = b (mod m), a and m are
2 // coprime.
3 // if the answer dont need to be greater than some value, the
4 // 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 }

```

5.5 Segment Sieve

```

1 // Retorna quantos primos tem entre [l, r] (inclusivo)
2 // precisa de um vetor com os primos ate 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 - 1] = false;
14            }
15        }
16    }
17
18    return accumulate(all(is_prime), 0ll);
19 }

```

5.6 Totient

```

1 // phi(n) = n * (1 - 1/p1) * (1 - 1/p2) * ...
2 int phi(int n) {
3     int result = n;
4     for (int i = 2; i * i <= n; i++) {
5         if (n % i == 0) {
6             while (n % i == 0)
7                 n /= i;
8             result -= result / i;
9         }
10    }
11    if (n > 1) // SE n sobrou, ele Ã' um fator primo
12        result -= result / n;
13    return result;
14 }
15
16 // crivo phi
17 const int MAXN_PHI = 1000001;
18 int phiv[MAXN_PHI];
19 void phi_sieve() {
20     for (int i = 0; i < MAXN_PHI; i++) phiv[i] = i;
21     for (int i = 2; i < MAXN_PHI; i++) {
22         if (phiv[i] == i) {
23             for (int j = i; j < MAXN_PHI; j += i) phiv[j] =
24                 phiv[j] / i;
25         }
26     }

```

5.7 Menor Fator Primo

```

1 const int MAXN = 1000001;
2 int spf[MAXN];
3 vector<int> primos;
4
5 void crivo() {
6     for (int i = 0; i < MAXN; i++) spf[i] = [i];
7     for (int i = 2; i * i < MAXN; i++) {
8         if (spf[i] == i) {
9             for (int j = i * i; j < MAXN; j += i) {
10                 if (spf[j] == j) {
11                     spf[j] = i;
12                 }
13             }
14         }
15     }
16     for (int i = 2; i < MAXN; i++) {
17         if (spf[i] == i) {
18             primos.push_back(i);
19         }
20     }
21 }
22
23 map<int, int> fatora(int n) {
24     map<int, int> fatores;
25     while (n > 1) {
26         fatores[spf[n]]++;
27         n /= spf[n];
28     }
29     return fatores;
30 }
31
32 int numero_de_divisores(int n) {
33     if (n == 1) return 1;
34     map<int, int> fatores = fatorar(n);
35     int nod = 1;
36     for (auto &[primo, expoente] : fatores) nod *= (expoente
37         + 1);
38     return nod;
39 }
40 // DEFINE INT LONG LONG
41 int soma_dos_divisores(int n) {
42     if (n == 1) return 1;
43     map<int, int> fatores = fatorar(n);
44     int sod = 1;
45     for (auto &[primo, expoente] : fatores) {
46         int termo_soma = 1;
47         int potencia_primo = 1;
48         for (int i = 0; i < expoente; i++) {
49             potencia_primo *= primo;
50             termo_soma += potencia_primo;
51         }
52         sod *= termo_soma;
53     }
54     return sod;
55 }
56 }
```

5.8 Exgcd

```

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

5.9 Fexp

```

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

```

12     return ans%m;
13 }
```

5.10 Divisores

```

1 // Retorna um vetor com os divisores de x
2 // eh preciso ter o crivo implementado
3 // O(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 }
```

5.11 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 }
```

5.12 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 }
```

5.13 Base Calc

```

1 int char_to_val(char c) {
2     if (c >= '0' && c <= '9') return c - '0';
3     else return c - 'A' + 10;
4 }
5
6 char val_to_char(int val) {
7     if (val >= 0 && val <= 9) return val + '0';
8     else return val - 10 + 'A';
9 }
10
11 int to_base_10(string &num, int bfrom) {
12     int result = 0;
13     int pot = 1;
14     for (int i = num.size() - 1; i >= 0; i--) {
15         if (char_to_val(num[i]) >= bfrom) return -1;
16         result += char_to_val(num[i]) * pot;
17         pot *= bfrom;
18     }
19     return result;
20 }
21
22 string from_base_10(int n, int bto) {
23     if (n == 0) return "0";
24     string result = "";
25     while (n > 0) {
```

```

26     result += val_to_char(n % bto);
27     n /= bto;
28 }
29 reverse(result.begin(), result.end());
30 return result;
31 }
32
33 string convert_base(string &num, int bfrom, int bto) {
34     int n_base_10 = to_base_10(num, bfrom);
35     return from_base_10(n_base_10, bto);
36 }
```

5.14 Fft

```

1 // multiplica dois polinomios em O(NlogN)
2
3 using cd = complex<double>;
4 const double PI = acos(-1);
5
6 void fft(vector<cd> &A, bool invert) {
7     int N = size(A);
8
9     for (int i = 1, j = 0; i < N; i++) {
10        int bit = N >> 1;
11        for (; j & bit; bit >>= 1)
12            j ^= bit;
13        j ^= bit;
14
15        if (i < j)
16            swap(A[i], A[j]);
17    }
18
19    for (int len = 2; len <= N; len <= 1) {
20        double ang = 2 * PI / len * (invert ? -1 : 1);
21        cd wlen(cos(ang), sin(ang));
22        for (int i = 0; i < N; i += len) {
23            cd w(1);
24            for (int j = 0; j < len/2; j++) {
25                cd u = A[i+j], v = A[i+j+len/2] * w;
26                A[i+j] = u + v;
27                A[i+j+len/2] = u - v;
28                w *= wlen;
29            }
30        }
31    }
32
33    if (invert) {
34        for (auto &x : A)
35            x /= N;
36    }
37 }
38
39 vector<int> multiply(vector<int> const &A, vector<int> const &
40     B) {
41     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
42     int N = 1;
43     while (N < size(A) + size(B))
44         N *= 2;
45     fa.resize(N);
46     fb.resize(N);
47
48     fft(fa, false);
49     fft(fb, false);
50     for (int i = 0; i < N; i++)
51         fa[i] *= fb[i];
52     fft(fa, true);
53
54     vector<int> result(N);
55     for (int i = 0; i < N; i++)
56         result[i] = round(fa[i].real());
57 }
```

6 Graph

6.1 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 }
```

6.2 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 }
```

6.3 Eulerian Path

```

1 /**
2 * versao que assume: #define int long long
3 *
4 * Retorna um caminho/ciclo euleriano em um grafo (se existir).
5 * - g: lista de adjacencia (vector<vector<int>>).
6 * - directed: true se o grafo for dirigido.
7 * - s: vertice inicial.
8 * - e: vertice final (opcional). Se informado, tenta caminho
8 * de s ate e.
9 * - O(Nlog(N))
10 * Retorna vetor com a sequencia de vertices, ou vazio se
11 * impossivel.
12 */
13 vector<int> eulerian_path(const vector<vector<int>>& g, bool
14     directed, int s, int e = -1) {
15     int n = (int)g.size();
16     // copia das adjacencias em multiset para permitir
17     remoção especifica
```

```

15  vector<multiset<int>> h(n);
16  vector<int> in_degree(n, 0);
17  vector<int> result;
18  stack<int> st;
19  // preencher h e indegrees
20  for (int u = 0; u < n; ++u) {
21      for (auto v : g[u]) {
22          ++in_degree[v];
23          h[u].emplace(v);
24      }
25  }
26  st.emplace(s);
27  if (e != -1) {
28      int out_s = (int)h[s].size();
29      int out_e = (int)h[e].size();
30      int diff_s = in_degree[s] - out_s;
31      int diff_e = in_degree[e] - out_e;
32      if (diff_s * diff_e != -1) return {};
33  }
34  for (int u = 0; u < n; ++u) {
35      if (e != -1 && (u == s || u == e)) continue;
36      int out_u = (int)h[u].size();
37      if (in_degree[u] != out_u || (!directed && (in_degree
38 [u] & 1))) {
39          return {};
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              --in_degree[u];
55          }
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 }

```

6.4 Dinitz

```

1 // Complexidade: O(V^2E)
2
3 struct FlowEdge {
4     int from, to;
5     long long cap, flow = 0;
6     FlowEdge(int from, int to, long long cap) : from(from),
7               to(to), cap(cap) {}
7 };
8
9 struct Dinic {
10    const long long flow_inf = 1e18;
11    vector<FlowEdge> edges;
12    vector<vector<int>> adj;
13    int n, m = 0;
14    int s, t;
15    vector<int> level, ptr;
16    queue<int> q;
17
18    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
19        adj.resize(n);
20        level.resize(n);
21        ptr.resize(n);
22    }
23
24    void add_edge(int from, int to, long long cap) {
25        edges.emplace_back(from, to, cap);
26        edges.emplace_back(to, from, 0);
27        adj[from].push_back(m);
28        adj[to].push_back(m + 1);
29        m += 2;
30    }
31

```

```

32    bool bfs() {
33        while (!q.empty()) {
34            int from = q.front();
35            q.pop();
36            for (int id : adj[from]) {
37                if (edges[id].cap == edges[id].flow)
38                    continue;
39                if (level[edges[id].to] != -1)
40                    continue;
41                level[edges[id].to] = level[from] + 1;
42                q.push(edges[id].to);
43            }
44        }
45        return level[t] != -1;
46    }
47
48    long long dfs(int from, long long pushed) {
49        if (pushed == 0)
50            return 0;
51        if (from == t)
52            return pushed;
53        for (int& cid = ptr[from]; cid < (int)adj[from].size
54 (); cid++) {
55            int id = adj[from][cid];
56            int to = edges[id].to;
57            if (level[from] + 1 != level[to])
58                continue;
59            long long tr = dfs(to, min(pushed, edges[id].cap
60 - edges[id].flow));
61            if (tr == 0)
62                continue;
63            edges[id].flow += tr;
64            edges[id ^ 1].flow -= tr;
65            return tr;
66        }
67        return 0;
68    }
69
70    long long flow() {
71        long long f = 0;
72        while (true) {
73            fill(level.begin(), level.end(), -1);
74            level[s] = 0;
75            q.push(s);
76            if (!bfs())
77                break;
78            fill(ptr.begin(), ptr.end(), 0);
79            while (long long pushed = dfs(s, flow_inf)) {
80                f += pushed;
81            }
82        }
83    }
84 }

```

6.5 Khan

```

1 // topo-sort DAG
2 // lexicograficamente menor.
3 // N: numero de Vértices (1-indexado)
4 // adj: lista de adjacencia 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     if (result.size() != N) {
35         return {};
36     }
37     return result;
38 }

6.6 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}
19}

6.7 Acha Pontes

1 vector<int> d, low, pai;      // d[v] Tempo de descoberta ( discovery time)
2 vector<bool> vis;
3 vector<int> pontos_articulacao;
4 vector<pair<int, int>> pontes;
5 int tempo;
6
7 vector<vector<int>> adj;
8
9 void dfs(int u) {
10    vis[u] = true;
11    tempo++;
12    d[u] = low[u] = tempo;
13    int filhos_dfs = 0;
14    for (int v : adj[u]) {
15        if (v == pai[u]) continue;
16        if (vis[v]) { // back edge
17            low[u] = min(low[u], d[v]);
18        } else {
19            pai[v] = u;
20            filhos_dfs++;
21            dfs(v);
22            low[u] = min(low[u], low[v]);
23            if (pai[u] == -1 && filhos_dfs > 1) {
24                pontos_articulacao.push_back(u);
25            }
26            if (pai[u] != -1 && low[v] >= d[u]) {
27                pontos_articulacao.push_back(u);
28            }
29            if (low[v] > d[u]) {
30                pontes.push_back({min(u, v), max(u, v)});
31            }
32        }
33    }
34}

6.8 Edmonds-karp

1 // Edmonds-Karp com scaling O(E log(F))
2
3 int n, m;
4 const int MAXN = 510;
5 vector<vector<int>> capacity(MAXN, vector<int>(MAXN, 0));
6 vector<vector<int>> adj(MAXN);
7
8 int bfs(int s, int t, int scale, vector<int>& parent) {
9
10    fill(parent.begin(), parent.end(), -1);
11    parent[s] = -2;
12    queue<pair<int, int>> q;
13    q.push({s, LLONG_MAX});
14
15    while (!q.empty()) {
16        int cur = q.front().first;
17        int flow = q.front().second;
18        q.pop();
19
20        for (int next : adj[cur]) {
21            if (parent[next] == -1 && capacity[cur][next] >=
22 scale) {
23                parent[next] = cur;
24                int new_flow = min(flow, capacity[cur][next]);
25
26                if (next == t)
27                    return new_flow;
28                q.push({next, new_flow});
29            }
30        }
31    }
32
33    int maxflow(int s, int t) {
34        int flow = 0;
35        vector<int> parent(MAXN);
36        int new_flow;
37        int scaling = 1ll << 62;
38
39        while (scaling > 0) {
40            while (new_flow = bfs(s, t, scaling, parent)) {
41                if (new_flow == 0) continue;
42                flow += new_flow;
43                int cur = t;
44                while (cur != s) {
45                    int prev = parent[cur];
46                    capacity[prev][cur] -= new_flow;
47                    capacity[cur][prev] += new_flow;
48                    cur = prev;
49                }
50            }
51            scaling /= 2;
52        }
53
54    return flow;
55 }

6.9 Kruskal

1 // Ordena as arestas por peso, insere se ja nao estiver no
   mesmo componente
2 // O(E log E)
3
4 struct Edge {
5     int u, v, w;
6     bool operator < (Edge const & other) {
7         return weight < other.weight;
8     }
9 }
10
11 vector<Edge> kruskal(int n, vector<Edge> edges) {
12     vector<Edge> mst;
13     DSU dsu = DSU(n + 1);
14     sort(edges.begin(), edges.end());
15     for (Edge e : edges) {
16         if (dsu.find(e.u) != dsu.find(e.v)) {
17             mst.push_back(e);
18             dsu.join(e.u, e.v);
19         }
20     }
21     return mst;
22 }

6.10 Bellman Ford

1 struct Edge {
2     int u, v, w;
3 };
4
5 // se x = -1, nao 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());

```

6.11 Lca Jc

```

1 const int MAXN = 200005;
2 int N;
3 int LOG;
4
5 vector<vector<int>> adj;
6 vector<int> profundidade;
7 vector<vector<int>> cima; // cima[v][j] eh o 2^j-esimo
8 // ancestral de v
9
10 void dfs(int v, int p, int d) {
11     profundidade[v] = d;
12     cima[v][0] = p; // o pai direto eh o 2^0-Ã¡simo ancestral
13     for (int j = 1; j < LOG; j++) {
14         // se o ancestral 2^(j-1) existir, calculamos o 2^j
15         if (cima[v][j - 1] != -1) {
16             cima[v][j] = cima[cima[v][j - 1]][j - 1];
17         } else {
18             cima[v][j] = -1; // nao tem ancestral superior
19         }
20         for (int nei : adj[v]) {
21             if (nei != p) {
22                 dfs(nei, v, d + 1);
23             }
24     }
25 }
26
27 void build(int root) {
28     LOG = ceil(log2(N));
29     profundidade.assign(N + 1, 0);
30     cima.assign(N + 1, vector<int>(LOG, -1));
31     dfs(root, -1, 0);
32 }
33
34 int get_lca(int a, int b) {
35     if (profundidade[a] < profundidade[b]) {
36         swap(a, b);
37     }
38     // sobe 'a' ate a mesma profundidade de 'b'
39     for (int j = LOG - 1; j >= 0; j--) {
40         if (profundidade[a] - (1 << j) >= profundidade[b]) {
41             a = cima[a][j];
42         }
43     }
44     // se 'b' era um ancestral de 'a', entÃ£o 'a' agora Ã©
45     // igual a 'b'
46     if (a == b) {
47         return a;
48     }
49     // sobe os dois juntos ate encontrar os filhos do LCA
50     for (int j = LOG - 1; j >= 0; j--) {
51         if (cima[a][j] != -1 && cima[a][j] != cima[b][j]) {
52             a = cima[a][j];
53             b = cima[b][j];
54         }
55     }

```

```

55     return cima[a][0];
56 }

```

6.12 Lca

```

1 // LCA - CP algorithm
2 // preprocessing O(NlogN)
3 // lca O(logN)
4 // Uso: criar LCA com a quantidade de vertices (n) e lista de
5 // adjacencias (adj)
6 // chamar a funcao preprocess com a raiz da arvore
7
8 struct LCA {
9     int n, l, timer;
10    vector<vector<int>> adj;
11    vector<int> tin, tout;
12    vector<vector<int>> up;
13
14    LCA(int n, const vector<vector<int>>& adj) : n(n), adj(
15        adj) {}
16
17    void dfs(int v, int p) {
18        tin[v] = ++timer;
19        up[v][0] = p;
20        for (int i = 1; i <= l; ++i)
21            up[v][i] = up[up[v][i-1]][i-1];
22
23        for (int u : adj[v]) {
24            if (u != p)
25                dfs(u, v);
26        }
27
28        tout[v] = ++timer;
29    }
30
31    bool is_ancestor(int u, int v) {
32        return tin[u] <= tin[v] && tout[u] >= tout[v];
33    }
34
35    int lca(int u, int v) {
36        if (is_ancestor(u, v))
37            return u;
38        if (is_ancestor(v, u))
39            return v;
40        for (int i = l; i >= 0; --i) {
41            if (!is_ancestor(up[u][i], v))
42                u = up[u][i];
43        }
44        return up[u][0];
45    }
46
47    void preprocess(int root) {
48        tin.resize(n);
49        tout.resize(n);
50        timer = 0;
51        l = ceil(log2(n));
52        up.assign(n, vector<int>(l + 1));
53        dfs(root, root);
54    }

```

6.13 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 (pos-order)
8 void dfs1(int u) {
9     vis[u] = true;
10    for (int v : adj[u]) {
11        if (!vis[v]) {
12            dfs1(v);
13        }
14    }
15    order.push_back(u);
16}
17
18 // dfs o grafo reverso para encontrar os SCCs
19 void dfs2(int u, int c) {
20    component[u] = c;
21    for (int v : adj_rev[u]) {

```

```

22     if (component[v] == -1) {
23         dfs2(v, c);
24     }
25 }
26 }
27
28 int kosaraju() {
29     order.clear();
30     fill(vis + 1, vis + N + 1, false);
31     for (int i = 1; i <= N; i++) {
32         if (!vis[i]) {
33             dfs1(i);
34         }
35     }
36     fill(component + 1, component + N + 1, -1);
37     int c = 0;
38     reverse(order.begin(), order.end());
39     for (int u : order) {
40         if (component[u] == -1) {
41             dfs2(u, c++);
42         }
43     }
44     return c;
45 }

```

6.14 Min Cost Max Flow

```

1 // Encontra o menor custo para passar K de fluxo em um grafo
2 // com N vertices
3 // Funciona com multiplas arestas para o mesmo par de
4 // vertices
5 // Para encontrar o min cost max flow eh so fazer K =
6 // infinito
7
8 struct Edge {
9     int from, to, capacity, cost, id;
10};
11
12 const int INF = LLONG_MAX;
13
14 void shortest_paths(int n, int v0, vector<int>& dist, vector<
15     int>& edge_to) {
16     dist.assign(n, INF);
17     dist[v0] = 0;
18     vector<bool> in_queue(n, false);
19     queue<int> q;
20     q.push(v0);
21     edge_to.assign(n, -1);
22
23     while (!q.empty()) {
24         int u = q.front();
25         q.pop();
26         in_queue[u] = false;
27         for (auto [v, id] : adj[u]) {
28             if (edges[id].capacity > 0 && dist[v] > dist[u] +
29                 edges[id].cost) {
30                 dist[v] = dist[u] + edges[id].cost;
31                 edge_to[v] = id;
32                 if (!in_queue[v]) {
33                     in_queue[v] = true;
34                     q.push(v);
35                 }
36             }
37         }
38     }
39     void add_edge(int from, int to, int capacity, int cost){
40         edges.push_back({from, to, capacity, cost, (int)edges.
41             size()});
42         edges.push_back({to, from, 0, -cost, (int)edges.size()});
43         // reversa
44     }
45     int min_cost_flow(int N, int K, int s, int t) {
46         adj.assign(N, vector<array<int, 2>>());
47
48         for (Edge e : edges) {
49             adj[e.from].push_back({e.to, e.id});
50
51         int flow = 0;

```

```

52         int cost = 0;
53         vector<int> dist, edge_to;
54         while (flow < K) {
55             shortest_paths(N, s, dist, edge_to);
56             if (dist[t] == INF)
57                 break;
58
59             // find max flow on that path
60             int f = K - flow;
61             int cur = t;
62             while (cur != s) {
63                 f = min(f, edges[edge_to[cur]].capacity);
64                 cur = edges[edge_to[cur]].from;
65             }
66
67             // apply flow
68             flow += f;
69             cost += f * dist[t];
70             cur = t;
71             while (cur != s) {
72                 int edge = edge_to[cur];
73                 int rev_edge = edge^1;
74
75                 edges[edge].capacity -= f;
76                 edges[rev_edge].capacity += f;
77                 cur = edges[edge].from;
78             }
79
80             if (flow < K)
81                 return -1;
82             else
83                 return cost;
84         }
85     }

```

6.15 Diametro

```

1 /*
2 O(N + M), retorna cara mais distante, len
3 do caminho em arestas e os caras do caminho em um vetor
4 Para pegar o diametro da arvore, chamar duas vezes
5 auto [a, DA, PA] = calc(1);
6 auto [b, len_diametro, caminho_diametro] = calc(a);
7 */
8 auto calc = [&](int S) -> tuple<int, int, vector<int>> {
9     queue<pair<int, int>> q;
10    q.push({S, 0});
11    vector<int> dp(n + 1, -1), pai(n + 1, 0);
12    dp[S] = 0;
13    pai[S] = -1;
14    int longe = S;
15    int dist = 0;
16    while (!q.empty()){
17        auto [u, d] = q.front();
18        q.pop();
19        if (d > dist){
20            dist = d;
21            longe = u;
22        }
23        for (int v : adj[u]){
24            if (dp[v] == -1){
25                dp[v] = d + 1;
26                pai[v] = u;
27                q.push({v, d + 1});
28            }
29        }
30    }
31    int cur = longe;
32    vector<int> caminho;
33    while (cur != -1){
34        caminho.push_back(cur);
35        cur = pai[cur];
36    }
37    return {longe, dist, caminho };
38 };

```

7 Primitives

8 DP

8.1 Lis

```

1 int lis_nlogn(vector<int> &v) {
2     vector<int> lis;
3     lis.push_back(v[0]);
4     for (int i = 1; i < v.size(); i++) {
5         if (v[i] > lis.back()) {
6             // estende a lis
7             lis.push_back(v[i]);
8         } else {
9             // encontra o primeiro elemento em lis que >= v[i]
10            auto it = lower_bound(lis.begin(), lis.end(), v[i]);
11            *it = v[i];
12        }
13    }
14    return lis.size();
15 }
16
17 // lis na tree do problema da sub
18 const int MAXN_TREE = 100001;
19 vector<int> adj[MAXN_TREE];
20 int values[MAXN_TREE];
21 int ans = 0;
22
23 void dfs(int u, int p, vector<int>& tails) {
24     auto it = lower_bound(tails.begin(), tails.end(), values[u]);
25     int prev = -1;
26     bool coloquei = false;
27     if (it == tails.end()) {
28         tails.push_back(values[u]);
29         coloquei = true;
30     } else {
31         prev = *it;
32         *it = values[u];
33     }
34     ans = max(ans, (int)tails.size());
35     for (int v : adj[u]) {
36         if (v != p) {
37             dfs(v, u, tails);
38         }
39     }
40     if (coloquei) {
41         tails.pop_back();
42     } else {
43         *it = prev;
44     }
45 }

```

8.2 Edit Distance

```

1 vector<vector<int>> dp(n+1, vector<int>(m+1, LLONG_MIN));
2 for(int j = 0; j <= m; j++) dp[0][j] = j;
3 for(int i = 0; i <= n; i++) dp[i][0] = i;
4 for(int i = 1; i <= n; i++) {
5     for(int j = 1; j <= m; j++) {
6         if(a[i-1] == b[j-1]) {
7             dp[i][j] = dp[i-1][j-1];
8         } else {
9             dp[i][j] = min({dp[i-1][j] + 1, dp[i][j-1] +
10                1, dp[i-1][j-1] + 1});
11        }
12    }
13 }
cout << dp[n][m];

```

8.3 Bitmask

```

1 // dp de intervalos com bitmask
2 int prox(int idx) {
3     return lower_bound(S.begin(), S.end(), array<int, 4>{S[
4         idx][1], 0ll, 0ll, 0ll}) - S.begin();
5 }
6 int dp[1002][(int)(1ll << 10)];
7
8 int rec(int i, int vis) {
9     if (i == (int)S.size()) {
10        if (__builtin_popcountll(vis) == N) return 0;
11        return LLONG_MIN;
12    }
13    if (dp[i][vis] != -1) return dp[i][vis];
14    int ans = rec(i + 1, vis);
15    ans = max(ans, rec(prox(i), vis | (1ll << S[i][3])) + S[i
16        ][2]);
17 }

```

```

16     return dp[i][vis] = ans;
17 }

```

8.4 Lcs

```

1 string s1, s2;
2 int dp[1001][1001];
3 int lcs(int i, int j) {
4     if (i < 0 || j < 0) return 0;
5     if (dp[i][j] != -1) return dp[i][j];
6     if (s1[i] == s2[j]) return dp[i][j] = 1 + lcs(i - 1, j -
7         1);
8     else return dp[i][j] = max(lcs(i - 1, j), lcs(i, j - 1));
8 }

```

8.5 Digit

```

1 vector<int> digits;
2
3 int dp[20][10][2][2];
4
5 int rec(int i, int last, int flag, int started) {
6     if (i == (int)digits.size()) return 1;
7     if (dp[i][last][flag][started] != -1) return dp[i][last][
8         flag][started];
9     int lim;
10    if (flag) lim = 9;
11    else lim = digits[i];
12    int ans = 0;
13    for (int d = 0; d <= lim; d++) {
14        if (started && d == last) continue;
15        int new_flag = flag;
16        int new_started = started;
17        if (d > 0) new_started = 1;
18        if (!flag && d < lim) new_flag = 1;
19        ans += rec(i + 1, d, new_flag, new_started);
20    }
21    return dp[i][last][flag][started] = ans;
21 }

```

8.6 Knapsack

```

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

```

8.7 Lis Seg

```

1 // comprime coordenadas pra quando ai <= 1e9
2 vector<int> vals(a.begin(), a.end());
3 sort(vals.begin(), vals.end());
4 vals.erase(unique(vals.begin(), vals.end()), vals.end());
5 auto id = [&](int x) -> int {
6     return lower_bound(vals.begin(), vals.end(), x) - vals.
7     begin();
8 }
9 for (int i = 0; i < n; i++) a[i] = id(a[i]);
10 SegTree seg(n);
11 // dp[i]: lis que termina em i
12 vector<int> dp(n, 1); // dp[i] = 1 base case
13 for (int i = 0; i < n; i++) {
14     if (a[i] > 0) dp[i] = seg.query(0, max(0ll, a[i] - 1)) +
15         1;
16     seg.update(a[i], dp[i]);
17 }
17 cout << *max_element(dp.begin(), dp.end()) << '\n';

```

8.8 Disjoint Blocks

```

1 // num max de subarrays disjuntos com soma x usando apenas
2 // prefixo i (ou seja, considerando prefixo a[1..i]).
3 int disjointSumX(vector<int> &a, int x) {
4     int n = a.size();
5     map<int, int> best; // best[pref] = melhor dp visto para
6     // esse pref
7     best[0] = 0;
8     int pref = 0;
9     vector<int> dp(n + 1, 0); // dp[0] = 0
10    for (int i = 1; i <= n; i++) {
11        pref += a[i - 1];
12        dp[i] = dp[i - 1];
13        auto it = best.find(pref - x);
14        if (it != best.end()) {
15            dp[i] = max(dp[i], it->second + 1);
16        }
17        best[pref] = max(best[pref], dp[i]);
18    }
19    return dp[n];

```

9 General

9.1 Brute Choose

```

1 vector<int> elements;
2 int N, K;
3 vector<int> comb;
4
5 void brute_choose(int i) {
6     if (comb.size() == K) {
7         for (int j = 0; j < comb.size(); j++) {
8             cout << comb[j] << ',';
9         }
10        cout << '\n';
11        return;
12    }
13    if (i == N) return;
14    int r = N - i;
15    int preciso = K - comb.size();
16    if (r < preciso) return;
17    comb.push_back(elements[i]);
18    brute_choose(i + 1);
19    comb.pop_back();
20    brute_choose(i + 1);
21 }

```

9.2 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 }

```

9.3 Mex

```

1 struct MEX {
2     map<int, int> f;
3     set<int> falta;
4     int tam;
5     MEX(int n) : tam(n) {
6         for (int i = 0; i <= n; i++) falta.insert(i);
7     }
8     void add(int x) {
9         f[x]++;
10        if (f[x] == 1 && x >= 0 && x <= tam) {
11            falta.erase(x);
12        }
13    }
14    void rem(int x) {
15        if (f.count(x) && f[x] > 0) {
16            f[x]--;
17            if (f[x] == 0 && x >= 0 && x <= tam) {
18                falta.insert(x);
19            }

```

```

20        }
21    }
22    int get() {
23        if (falta.empty()) return tam + 1;
24        return *falta.begin();
25    }
26 }

```

9.4 Count Permutations

```

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

```

9.5 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 }

```

10 Geometry

10.1 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    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(),
68             pp) == -1)
69         {
70             // Àl -1 pq eu nÃ¢o quero excluir os 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.back(),
81             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 }
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 }

```

10.2 Inside Polygon

```

1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4     int x = ccw(a, b, e);
5     int y = ccw(b, c, e);
6     int z = ccw(c, a, e);
7     return !((x==1 or y==1 or z==1) and (x== -1 or y== -1 or z
8     == -1));
9 }
10
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
21     // bordo
22     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0)
23     // return false;
24     // if(r==2 and ccw(p[0], p[1], e)==0) return false;
25     // if(ccw(p[r], p[r-1], e)==0) return false;
26     return insideT(p[0], p[r-1], p[r], e);
27 }

```

```

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

10.3 Point Location

```

1 int32_t main(){
2     sws;
3
4     int t; cin >> t;
5
6     while(t--){
7
8         int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1 >> x2 >>
9         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 }
```

10.4 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, ll y3) {
5      return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 -
6          y2));
7  }
8  ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
9      return gcd(abs(x2 - x1), abs(y2 - y1));
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);
17     ll tot_borda = pontos_borda(x1, y1, x2, y2) +
18         pontos_borda(x2, y2, x3, y3) + pontos_borda(x3, y3, x1,
19         y1);
20     ll ans = (area - tot_borda) / 2 + 1;
21     cout << ans << endl;
22 }
23 }
```