

Competitive Programming Notebook

Programadores Roblox

Contents

1 String	2	6.10 Bellman Ford	11
1.1 Trie	2	6.11 Lca Jc	11
1.2 Hashing	2	6.12 Lca	12
1.3 Z Function	2	6.13 Kosaraju	12
1.4 Kmp	2	6.14 Pega Ciclo	13
		6.15 Min Cost Max Flow	13
2 DS	2	7 Primitives	14
2.1 Segtree Iterativa	2	8.1 Lis	14
2.2 Merge Sort Tree	3	8.2 Edit Distance	14
2.3 Ordered Set E Map	3	8.3 Bitmask	14
2.4 Sparse Table	3	8.4 Lcs	14
2.5 Psum 2d	3	8.5 Digit	14
2.6 Segtree Sum	4	8.6 Knapsack	15
2.7 Dsu	4	8.7 Lis Seg	15
2.8 Bit	4	8.8 Disjoint Blocks	15
3 Search and sort	5	9 General	15
3.1 Merge Sort	5	9.1 Brute Choose	15
3.2 Monotonic Stack	5	9.2 Struct	15
4 Stress	5	9.3 Mex	15
4.1 Gen	5	9.4 Count Permutations	16
5 Math	5	9.5 Bitwise	16
5.1 Combinatorics	5	10 Geometry	16
5.2 Equacao Diofantina	6	10.1 Convex Hull	16
5.3 Discrete Log	6	10.2 Inside Polygon	17
5.4 Segment Sieve	6	10.3 Point Location	17
5.5 Totient	6	10.4 Lattice Points	18
5.6 Menor Fator Primo	6		
5.7 Exgcd	7		
5.8 Fexp	7		
5.9 Divisores	7		
5.10 Crivo	7		
5.11 Mod Inverse	7		
5.12 Base Calc	7		
5.13 Fft	8		
6 Graph	8		
6.1 Dijkstra	8		
6.2 Floyd Warshall	8		
6.3 Eulerian Path	9		
6.4 Dinitz	9		
6.5 Khan	10		
6.6 Topological Sort	10		
6.7 Acha Pontes	10		
6.8 Edmonds-karp	10		
6.9 Kruskal	11		

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]
4 // from left to right - O(1)
5 // query_inv(l, r) from right to left - O(1)
6 // patrocinado por tiagodfs
7
8 mt19937 rng(time(nullptr));
9
10 struct Hash {
11     const int X = rng();
12     const int MOD = 1e9+7;
13     int n; string s;
14     vector<int> h, hi, p;
15     Hash() {}
16     Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
17         for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1) % MOD;
18         for (int i=0;i<n;i++)
19             h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
20         for (int i=n-1;i>=0;i--)
21             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X) % MOD;
22     }
23     int query(int l, int r) {
```

```

23         int hash = (h[r] - (l ? h[l-1]*p[r-l]:0)) % MOD;
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]:0)) % MOD;
28         return hash < 0 ? hash + MOD : hash;
29     }
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 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 }
```

2 DS

2.1 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         for (int i = 0; i < n; i++)
17             tree[n + i] = data[i];
18     }
19 }
```

```

1   for (int i = n - 1; i > 0; --i)
2     tree[i] = combine(tree[i * 2], tree[i * 2
3     + 1]);
4 }
5 T range_query(int l, int r) {
6   T res_l = neutral_value, res_r =
7   neutral_value;
8
9   for (l += n, r += n + 1; l < r; l >>= 1, r
10 >>= 1) {
11     if (l & 1) res_l = combine(res_l, tree[l
12 + 1]);
13     if (r & 1) res_r = combine(tree[--r],
14     res_r);
15   }
16
17   return combine(res_l, res_r);
18 }
19 void update(int pos, T new_val) {
20   tree[pos += n] = new_val;
21   for (pos >= 1; pos > 0; pos >>= 1)
22     tree[pos] = combine(tree[2 * pos], tree[2
23     * pos + 1]);
24 }
25 };
26
27 
```

2.2 Merge Sort Tree

```

1 struct SegTree {
2   int n;
3   vector<vector<int>> tree;
4
5   SegTree(vector<int> &a) {
6     n = a.size();
7     tree.resize(4 * n);
8     build(1, 0, n - 1, a);
9   }
10  void build(int x, int lx, int rx, vector<int> &a)
11  {
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(L.begin(), L.end(), R.begin(), R.end(),
22    tree[x].begin());
23  }
24  int query(int x, int lx, int rx, int l, int r) {
25    if (lx >= l && rx <= r) {
26      auto &v = tree[x];
27      return v.end() - upper_bound(v.begin(),
28        .end(), r);
29    }
30    if (rx < l || lx > r) {
31      return 0;
32    }
33    int mid = lx + (rx - lx)/2;
34    return query(2 * x, lx, mid, l, r) + query(2
35    * x + 1, mid + 1, rx, l, r);
36  }
37
38 // Checar se o range eh todo distinto
39 vector<int> nr(n);
40 map<int, int> mp;
41 for (int i = n - 1; i >= 0; i--) {
42   mp[tree[i]]++;
43   if (mp[tree[i]] == 1) nr[i] = 1;
44 }
```

2.2 Merge Sort Tree

```

struct SegTree {
    int n;
    vector<vector<int>> tree;
}

SegTree(vector<int> &a) {
    n = a.size();
    tree.resize(4 * n);
    build(1, 0, n - 1, a);
}
void build(int x, int lx, int rx, vector<int> &a)
{
    if (lx == rx) {
        tree[x] = { a[lx] };
        return;
    }
    int mid = lx + (rx - lx)/2;
    build(2 * x, lx, mid, a);
    build(2 * x + 1, mid + 1, rx, a);
    auto &L = tree[2 * x], &R = tree[2 * x + 1];
    tree[x].resize(L.size() + R.size());
    merge(L.begin(), L.end(), R.begin(), R.end(),
          tree[x].begin());
}
int query(int x, int lx, int rx, int l, int r) {
    if (lx >= l && rx <= r) {
        auto &v = tree[x];
        return v.end() - upper_bound(v.begin(), v.end(), r);
    }
    if (rx < l || lx > r) {
        return 0;
    }
    int mid = lx + (rx - lx)/2;
    return query(2 * x, lx, mid, l, r) + query(2 * x + 1, mid + 1, rx, l, r);
}
int query(int l, int r) {
    return query(1, 0, n - 1, l, r);
}
}

// Checar se o range eh todo distinto
vector<int> nr(n);
map<int, int> mp;
for (int i = n - 1; i >= 0; i--) {

```

```

42     auto it = mp.find(a[i]);
43     nr[i] = it != mp.end() ? it->second : n;
44     mp[a[i]] = i;
45 }
46 SegTree seg(nr);

```

2.3 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
17 int main() {
18     int i, j, k, n, m;
19     o_set<int>st;
20     st.insert(1);
21     st.insert(2);
22     cout << *st.find_by_order(0) << endl; // k-esimo
23     elemento
24     cout << st.order_of_key(2) << endl; //numero de
25     elementos menores que k
26     o_map<int, int>mp;
27     mp.insert({1, 10});
28     mp.insert({2, 20});
29     cout << mp.find_by_order(0)->second << endl; // k-
30     esimo elemento
31     cout << mp.order_of_key(2) << endl; // numero de
32     elementos (chave) menores que k
33     return 0;
34 }

```

2.4 Sparse Table

```

1 // 0-index, 0(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 - 1))][j - 1]);
15             }
16         }
17     }
18     int query(int L, int R) {
19         int tamanho = R - L + 1;
20         int k = floor(log2(tamanho));
21         return max(st[L][k], st[R - (1 << k) + 1][k]);
22     }
23 };

```

2.5 Psum 2d

```

1 vector<vector<int>> psum(h+1, vector<int>(w+1, 0));    54
2 for (int i=1; i<=h; i++){                                55
3     for (int j=1; j<=w; j++){                            56
4         cin >> psum[i][j];                                57
5         psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[ 58
6             i-1][j-1];                                59
7     }                                              60
8 // retorna a psum2d do intervalo inclusivo [(a, b), ( 61
9 int retangulo(int a, int b, int c, int d){          62
10    c = min(c, h), d = min(d, w);                  63
11    a = max(0LL, a-1), b = max(0LL, b-1);          64
12    return v[c][d]-v[a][d]-v[c][b]+v[a][b];        65
13 }

```

2.6 Segtree Sum

```

1 struct SegTree {
2     vector<int> tree;
3     vector<int> lazy;
4     int n;
5     SegTree(int N) {
6         n = N;
7         tree.resize(4 * n);
8         lazy.resize(4 * n, 0);
9         // build(a);
10    }
11    void push(int lx, int rx, int x) {
12        if (lazy[x] == 0) {
13            return;
14        }
15        // tree[x] += lazy[x];
16        tree[x] += (rx - lx + 1) * lazy[x];
17        if (lx != rx) {
18            lazy[2 * x] += lazy[x];
19            lazy[2 * x + 1] += lazy[x];
20        }
21        lazy[x] = 0;
22    }
23    void build(vector<int> &a) {
24        build(0, n - 1, 1, a);
25    }
26    void build(int lx, int rx, int x, vector<int> &a)
27    {
28        if (lx == rx) {
29            tree[x] = a[lx];
30            return;
31        }
32        int mid = lx + (rx - lx) / 2;
33        build(lx, mid, 2 * x, a);
34        build(mid + 1, rx, 2 * x + 1, a);
35        tree[x] = tree[2 * x] + tree[2 * x + 1];
36    }
37    void update(int l, int r, int val) {
38        update(0, n - 1, 1, l, r, val);
39    }
40    void update(int lx, int rx, int x, int l, int r,
41    int val) {
42        push(lx, rx, x);
43        if (rx < l || lx > r) {
44            return;
45        }
46        if (lx >= l && rx <= r) {
47            lazy[x] += val;
48            push(lx, rx, x);
49            return;
50        }
51        int mid = lx + (rx - lx) / 2;
52        update(lx, mid, 2 * x, l, r, val);
53        update(mid + 1, rx, 2 * x + 1, l, r, val);
54        tree[x] = tree[2 * x] + tree[2 * x + 1];
55    }

```

```

int query(int l, int r) {
    return query(0, n - 1, 1, l, r);
}
int query(int lx, int rx, int x, int l, int r) {
    push(lx, rx, x);
    if (rx < l || lx > r) {
        return 0;
    }
    if (lx >= l && rx <= r) {
        return tree[x];
    }
    int mid = lx + (rx - lx) / 2;
    int s1 = query(lx, mid, 2 * x, l, r);
    int s2 = query(mid + 1, rx, 2 * x + 1, l, r);
    return s1 + s2;
}

```

2.7 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[i])));
10    }
11    bool same(int i, int j) {
12        return find(i) == find(j);
13    }
14    int get_size(int i) {
15        return sz[find(i)];
16    }
17    int count() {
18        return c; // quantos componentes conexos
19    }
20    int merge(int i, int j) {
21        if ((i = find(i)) == (j = find(j))) return
22        -1;
23        else --c;
24        if (rank[i] > rank[j]) swap(i, j);
25        par[i] = j;
26        sz[j] += sz[i];
27        if (rank[i] == rank[j]) rank[j]++;
28    }

```

2.8 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 {
3     int x = m - l + 1;
4     int y = r - m;
5     vector<int> left(x), right(y);
6     for (int i = 0; i < x; i++) left[i] = v[l + i];
7     for (int j = 0; j < y; j++) right[j] = v[m + 1 + j];
8     int i = 0, j = 0, k = l;
9     int swaps = 0;
10    while (i < x && j < y) {
11        if (left[i] <= right[j]) {
12            v[k++] = left[i++];
13        } else {
14            v[k++] = right[j++];
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 - l) / 2;
26         swaps += mergeSort(v, l, m);
27         swaps += mergeSort(v, m + 1, r);
28         swaps += mergeAndCount(v, l, m, r);
29     }
30     return swaps;
31 }
```

3.2 Monotonic Stack

```

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
6     for (int i = 0; i < n; i++) {
7         while (!s.empty() && (maior ? v[s.top()] <= v
8 [i] : v[s.top()] >= v[i])) {
9             s.pop();
10        }
11        if (s.empty()) {
12            result[i] = -1;
13        } else {
14            result[i] = v[s.top()];
15        }
16        s.push(i);
17    }
18    return result;
19 }
20
21 vector<int> find_dir(vector<int> &v, bool maior) {
22     int n = v.size();
23     vector<int> result(n);
24     stack<int> s;
25     for (int i = n - 1; i >= 0; i--) {
26         while (!s.empty() && (maior ? v[s.top()] <= v
27 [i] : v[s.top()] >= v[i])) {
28             s.pop();
29         }
30         if (s.empty()) {
31             result[i] = -1;
32         } 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
3 // stdc++.h
4 // criar a pasta bits e incluir com "" ou inves de <>
5 // compilar stress com: g++ -pipe -O3 -fno -march=
6 // native -mtune=native gen.ccp
7 // faz bastante diferen a no runtime
8
9 #include <bits/stdc++.h>
10 #include <cstdlib>
11 #include <ctime>
12 using namespace std;
13
14 int randi(int L, int R) { return L + rand() % (R - L
15 + 1); }
16 char randc(char L, char R) { return char(L + rand() %
17 (R - L + 1)); }
18
19 int main(int argc, char** argv) {
20     if (argc > 1) srand(atoi(argv[1]));
21     else srand(time(0));
22
23     int n = randi(1, 100);
24     cout << n << '\n';
25     for (int i = 0; i < n; i++) {
26         cout << randi(-10, 20) << ' ';
27     }
28     cout << '\n';
29 }
```

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] =
15     (fat[i - 1] * i) % MOD;
16     fati[MAXN_FATORIAL - 1] = inv(fat[MAXN_FATORIAL -
17 1]);
17     for (int i = MAXN_FATORIAL - 2; i >= 0; i--) fati
18     [i] = (fati[i + 1] * (i + 1)) % MOD;
19 }
20
21 int choose(int n, int k) {
22     if (k < 0 || k > n) return 0;
23     return (((fat[n] * fati[k]) % MOD) * fati[n - k])
24     % 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 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
8     int an = 1;
9     for (int i = 0; i < n; ++i)
10        an = (an * 111 * a) % m;
11
12     unordered_map<int, vector<int>> vals;
13     for (int q = 0, cur = b; q <= n; ++q) {
14         vals[cur].push_back(q);
15         cur = (cur * 111 * a) % m;
16     }
17
18     int res = LLONG_MAX;
19
20     for (int p = 1, cur = 1; p <= n; ++p) {
21         cur = (cur * 111 * an) % m;
22         if (vals.count(cur)) {
23             for (int q: vals[cur]){
24                 int ans = n * p - q;
25                 res = min(res, ans);
26             }
27         }
28     }
29 }

```

```

26     }
27 }
28 return res;
29 }

```

5.4 Segment Sieve

```

1 // Retorna quantos primos tem entre [l, r] (inclusivo
2 // )
3 // precisa de um vetor com os primos ate sqrt(r)
4 int seg_sieve(int l, int r){
5     if (l > r) return 0;
6     vector<bool> is_prime(r - l + 1, true);
7     if (l == 1) is_prime[0] = false;
8
9     for (int p : primos){
10        if (p * p > r) break;
11        int start = max(p * p, (l + p - 1) / p * p);
12        for (int j = start; j <= r; j += p){
13            if (j >= l) {
14                is_prime[j - l] = false;
15            }
16        }
17    }
18
19    return accumulate(all(is_prime), 0ll);
20 }

```

5.5 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)
24                 phiv[j] -= phiv[j] / i;
25         }
26     }
27 }

```

5.6 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 }

```

```

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 + 1);
37     return nod;
38 }
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 }
```

5.7 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 }
```

5.8 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    return ans%m;
13 }
```

5.9 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 }
```

5.10 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 }
```

5.11 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 }
```

5.12 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 {
35     int n_base_10 = to_base_10(num, bfrom);
36     return from_base_10(n_base_10, bto);
37 }
```

5.13 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        if (invert) {
33            for (auto &x : A)
34                x /= N;
35        }
36    }
37 }
38
39 vector<int> multiply(vector<int> const& A, vector<int>
40 > const& 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 <= 1;
45     fa.resize(N);
46     fb.resize(N);
47     fft(fa, false);
48     fft(fb, false);
49     for (int i = 0; i < N; i++)
50         fa[i] *= fb[i];
51     fft(fa, true);
52
53     vector<int> result(N);
54     for (int i = 0; i < N; i++)
55         result[i] = round(fa[i].real());
56     return result;
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     ;
10    pq.push({0, S});
11    while(pq.size()) {
12        ll v = pq.top().second;
13        pq.pop();
14        if(vis[v]) continue;
15        vis[v] = 1;
16        for(auto &[peso, vizinho] : adj[v]) {
17            if(dist[vizinho] > dist[v] + peso) {
18                dist[vizinho] = dist[v] + peso;
19                pq.push({dist[vizinho], vizinho});
20            }
21        }
22    }
23    return dist;
24 }
```

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
17 void solve() {
18     int m, q;
19     cin >> n >> m >> q;
20     for(int i = 0; i < n; i++) {
21         for(int j = i; j < n; j++) {
22             if(i == j) {
23                 dist[i][j] = dist[j][i] = 0;
24             } else {
25                 dist[i][j] = dist[j][i] = inf;
26             }
27         }
28     }
29     for(int i = 0; i < m; i++) {
30         int u, v, w;
31         cin >> u >> v >> w;
32         u--;
33         v--;
34         dist[u][v] = min(dist[u][v], w);
35     }
36 }
```

```

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
5 * existir).
6 * - g: lista de adjacencia (vector<vector<int>>).
7 * - directed: true se o grafo for dirigido.
8 * - s: vertice inicial.
9 * - e: vertice final (opcional). Se informado, tenta
10 * caminho de s ate e.
11 * - O(Nlog(N))
12 * Retorna vetor com a sequencia de vertices, ou
13 * vazio se impossivel.
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     // copia das adjacencias em multiset para
19     // permitir remoção específica
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         // impossivel
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     while (!st.empty()) {
49         int u = st.top();
50         if (h[u].empty()) {
51             result.emplace_back(u);
52             st.pop();
53         } else {
54             int v = *h[u].begin();
55             auto it = h[u].find(v);
56             if (it != h[u].end()) h[u].erase(it);
57             --in_degree[v];
58             if (!directed) {
59                 auto it2 = h[v].find(u);
60                 if (it2 != h[v].end()) h[v].erase(it2);
61             }
62         }
63     }
64 }
```

```

53             if (it2 != h[v].end()) h[v].erase(it2);
54         );
55         --in_degree[u];
56     }
57     st.emplace(v);
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(
7         from), to(to), cap(cap) {}
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] +
42                1;
43                q.push(edges[id].to);
44            }
45        }
46        return level[t] != -1;
47    }
48
49    long long dfs(int from, long long pushed) {
50        if (pushed == 0)
51            return 0;
52        if (from == t)
53            return pushed;
54        for (int& cid = ptr[from]; cid < (int)adj[
55            from].size(); cid++) {
56            int id = adj[from][cid];
57            if (edges[id].cap - edges[id].flow > 0) {
58                long long new_pushed = dfs(id, pushed -
59                edges[id].cap);
60                if (new_pushed != 0) {
61                    edges[id].flow += edges[id].cap;
62                    edges[edges[id].to].flow -= edges[id].cap;
63                    ptr[from] = cid + 1;
64                    return new_pushed;
65                }
66            }
67        }
68        return 0;
69    }
70
71    long long max_flow() {
72        long long total_flow = 0;
73        while (bfs())
74            total_flow += dfs(s, flow_inf);
75        return total_flow;
76    }
77}
```

```

55         int to = edges[id].to;
56         if (level[from] + 1 != level[to])
57             continue;
58         long long tr = dfs(to, min(pushed, edges[
59             id].cap - edges[id].flow));
60         if (tr == 0)
61             continue;
62         edges[id].flow += tr;
63         edges[id ^ 1].flow -= tr;
64         return tr;
65     }
66     return 0;
67 }
68 long long flow() {
69     long long f = 0;
70     while (true) {
71         fill(level.begin(), level.end(), -1);
72         level[s] = 0;
73         q.push(s);
74         if (!bfs())
75             break;
76         fill(ptr.begin(), ptr.end(), 0);
77         while (long long pushed = dfs(s, flow_inf
78 )) {
79             f += pushed;
80         }
81     }
82     return f;
83 }

```

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     }
35     if (result.size() != N) {
36         return {};
37     }
38     return result;

```

6.6 Topological Sort

```

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

```

```

6 vector<vector<int>> adj(MAXN);
7
8 int bfs(int s, int t, int scale, vector<int>& parent) {
9     fill(parent.begin(), parent.end(), -1);
10    parent[s] = -2;
11    queue<pair<int, int>> q;
12    q.push({s, LLONG_MAX});
13
14    while (!q.empty()) {
15        int cur = q.front().first;
16        int flow = q.front().second;
17        q.pop();
18
19        for (int next : adj[cur]) {
20            if (parent[next] == -1 && capacity[cur][next] >= scale) {
21                parent[next] = cur;
22                int new_flow = min(flow, capacity[cur][next]);
23                if (next == t)
24                    return new_flow;
25                q.push({next, new_flow});
26            }
27        }
28    }
29
30    return 0;
31 }
32
33 int maxflow(int s, int t) {
34     int flow = 0;
35     vector<int> parent(MAXN);
36     int new_flow;
37     int scalling = 111 << 62;
38
39     while (scalling > 0) {
40         while (new_flow = bfs(s, t, scalling, parent))
41         {
42             if (new_flow == 0) continue;
43             flow += new_flow;
44             int cur = t;
45             while (cur != s) {
46                 int prev = parent[cur];
47                 capacity[prev][cur] -= new_flow;
48                 capacity[cur][prev] += new_flow;
49                 cur = prev;
50             }
51         }
52         scalling /= 2;
53     }
54
55     return flow;
56 }

```

6.9 Kruskal

```

1 // Ordena as arestas por peso, insere se ja nao
2 // estiver no mesmo componente
3 // O(E log E)
4
5 struct Edge {
6     int u, v, w;
7     bool operator < (Edge const & other) {
8         return weight < other.weight;
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         };
19         pai[e.v] = e.u;
20         x = e.v;
21     }
22 }
23
24 // achando caminho (se precisar)
25 for (int i = 0; i < n; i++) x = pai[x];
26
27 vector<int> ciclo;
28 for (int v = x;; v = pai[v]) {
29     cycle.push_back(v);
30     if (v == x && ciclo.size() > 1) break;
31 }
32 reverse(ciclo.begin(), ciclo.end());

```

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 ancestral de v
8
9 void dfs(int v, int p, int d) {
10     profundidade[v] = d;
11     cima[v][0] = p; // o pai direto eh o 2^0-Ãºltimo ancestral
12     for (int j = 1; j < LOG; j++) {
13         // se o ancestral 2^(j-1) existir, calculamos
14         // 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     }
21     for (int nei : adj[v]) {
22         if (nei != p) {
23             dfs(nei, v, d + 1);
24         }
25     }
26 }

```

```

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) >=
41             profundidade[b]) {
42             a = cima[a][j];
43         }
44     }
45     // se 'b' era um ancestral de 'a', entÃ£o 'a'
46     // agora Ã© igual a 'b'
47     if (a == b) {
48         return a;
49     }
50     // sobe os dois juntos ate encontrar os filhos do
51     // LCA
52     for (int j = LOG - 1; j >= 0; j--) {
53         if (cima[a][j] != -1 && cima[a][j] != cima[b]
54             [j]) {
55             a = cima[a][j];
56             b = cima[b][j];
57         }
58     }
59     return cima[a][0];
60 }

```

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
5 // lista de 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)
15        , adj(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        tout[v] = ++timer;
28    }
29
30    bool is_ancestor(int u, int v) {

```

```

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

```

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-
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    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 Pega Ciclo

```

1 // encontra um ciclo em g
2 // g[u] = vector<pair<id_aresta, vizinho>>
3 // rec_arestas: true -> retorna ids das arestas do
4 // ciclo; false -> retorna vertices 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(
    MAXN, -1); // color: 0,1,2 ; edgein[v] = id da
    // aresta que entra em v
12 int ini_ciclo = -1, fim_ciclo = -1, back_edge_id =
    -1;
13
14 bool dfs(int u, int pai_edge){
15     color[u] = 1; // cinza
16     for (auto [id, v] : g[u]) {
17         if (!DIRECTED && id == pai_edge) continue; // ignorar aresta de volta ao pai em n-dir
18         if (color[v] == 0) {
19             parent[v] = u;
20             edgein[v] = id;
21             if (dfs(v, id)) return true;
22         } else if (color[v] == 1) {
23             // back-edge u -> v detectado
24             ini_ciclo = u;
25             fim_ciclo = v;
26             back_edge_id = id;
27             return true;
28         }
29         // se color[v] == 2, ignora
30     }
31     color[u] = 2; // preto
32     return false;
33 }
34
35 // retorna ids das arestas do ciclo
36 vector<int> pega_ciclo(bool rec_arestas) {
37     for (int u = 1; u <= N; u++) {
38         if (color[u] != 0) continue;
39         if (dfs(u, -1)) {
40             // caminho u -> ... -> v via parent
41             vector<int> path;
42             int cur = ini_ciclo;
43             path.push_back(cur);
44             while (cur != fim_ciclo) {
45                 cur = parent[cur];
46                 path.push_back(cur);
47             }
48             // path = [u, ..., v] -> inverter para [v
49             , ..., u]
50             reverse(path.begin(), path.end());
51             if (!rec_arestas) return path;
52             // converte para ids das arestas: edgein[
53             node] eh a aresta que entra em node
54             vector<int> edges;
55             for (int i = 1; i < path.size(); i++)
56                 edges.push_back(edgein[path[i]]);
57             // adiciona a aresta de retorno u -> v
58             edges.push_back(back_edge_id);
59             return edges;
60         }
61     }
62     return {};
63 }

```

6.15 Min Cost Max Flow

```

1 // Encontra o menor custo para passar K de fluxo em
2 // um grafo com N vertices
3 // Funciona com multiphas 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,
15                     vector<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] >
29                 dist[u] + 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
40 void add_edge(int from, int to, int capacity, int
41               cost){
42     edges.push_back({from, to, capacity, cost, (int)
43                     edges.size()});
44     edges.push_back({to, from, 0, -cost, (int)edges.
45                     size()}); // reversa
46
47 int min_cost_flow(int N, int K, int s, int t) {
48     adj.assign(N, vector<array<int, 2>>());
49
50     for (Edge e : edges) {
51         adj[e.from].push_back({e.to, e.id});
52     }
53
54     int flow = 0;
55     int cost = 0;
56     vector<int> dist, edge_to;
57     while (flow < K) {
58         shortest_paths(N, s, dist, edge_to);
59         if (dist[t] == INF)
60             break;
61
62         // find max flow on that path
63         int f = K - flow;
64         int cur = t;
65         while (cur != s) {
66             int to = edge_to[cur];
67             int id = adj[cur][to];
68             edges[id].capacity -= f;
69             edges[id ^ 1].capacity += f;
70             cur = to;
71         }
72         flow += f;
73         cost += f * dist[t];
74     }
75
76     return cost;
77 }

```

```

63     f = min(f, edges[edge_to[cur]].capacity); 39
64     cur = edges[edge_to[cur]].from;           40
65   }                                         41
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 }

```

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
10      que >= v[i].
11      auto it = lower_bound(lis.begin(), lis.
12      end(), v[i]);
13      *it = v[i];
14    }
15  }
16  return lis.size();
17
18 // lis na tree do problema da sub
19 const int MAXN_TREE = 100001;
20 vector<int> adj[MAXN_TREE];
21 int values[MAXN_TREE];
22 int ans = 0;
23 void dfs(int u, int p, vector<int>& tails) {
24   auto it = lower_bound(tails.begin(), tails.end(),
25   values[u]);
26   int prev = -1;
27   bool coloquei = false;
28   if (it == tails.end()) {
29     tails.push_back(values[u]);
30     coloquei = true;
31   } else {
32     prev = *it;
33     *it = values[u];
34   }
35   ans = max(ans, (int)tails.size());
36   for (int v : adj[u]) {
37     if (v != p) {
38       dfs(v, u, tails);
39     }
40   }
41 }
42
43   if (coloquei) {
44     tails.pop_back();
45   } else {
46     *it = prev;
47   }
48 }

```

8.2 Edit Distance

```

1   vector<vector<int>> dp(n+1, vector<int>(m+1,
2   LLONG_MIN));
3   for(int j = 0; j <= m; j++) dp[0][j] = j;
4   for(int i = 0; i <= n; i++) dp[i][0] = i;
5   for(int i = 1; i <= n; i++) {
6     for(int j = 1; j <= m; j++) {
7       if(a[i-1] == b[j-1]) {
8         dp[i][j] = dp[i-1][j-1];
9       } else {
10         dp[i][j] = min({dp[i-1][j] + 1, dp[i]
11 [j-1] + 1, dp[i-1][j-1] + 1});
12       }
13     }
14   }
15   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[idx][1], 011, 011, 011}) - S.begin();
5
6 int dp[1002][(int)(111 << 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 | (111 << S[i
16 ][3])) + S[i][2]);
17   return dp[i][vis] = ans;
18 }

```

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 -
7   1, j - 1);
8   else return dp[i][j] = max(lcs(i - 1, j), lcs(i,
j - 1));
}

```

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
7 ][last][flag][started];
8   int lim;
9   if (flag) lim = 9;

```

```

10     else lim = digits[i];
11     int ans = 0;
12     for (int d = 0; d <= lim; d++) {
13       if (started && d == last) continue;
14       int new_flag = flag;
15       int new_started = started;
16       if (d > 0) new_started = 1;
17       if (!flag && d < lim) new_flag = 1;
18       ans += rec(i + 1, d, new_flag, new_started);
19     }
20   return dp[i][last][flag][started] = ans;
21 }
```

8.6 Knapsack

```

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

```

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) -
7     vals.begin();
8 };
9 // fim da parte de compr
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(0, a[i] -
15     1)) + 1;
16   seg.update(a[i], dp[i]);
17 cout << *max_element(dp.begin(), dp.end()) << '\n';

```

8.8 Disjoint Blocks

```

1 // num max de subarrays disjuntos com soma x usando
2 // apenas
3 // prefixo i (ou seja, considerando prefixo a[1..i]).
4 int disjointSumX(vector<int> &a, int x) {
5   map<int, int> best; // best[pref] = melhor dp
6   visto para 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()) {

```

```

14      dp[i] = max(dp[i], it->second + 1);
15    }
16    best[pref] = max(best[pref], dp[i]);
17  }
18  return dp[n];
19 }
```

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  }
12  if (i == N) return;
13  int r = N - i;
14  int preciso = K - comb.size();
15  if (r < preciso) return;
16  comb.push_back(elements[i]);
17  brute_choose(i + 1);
18  comb.pop_back();
19  brute_choose(i + 1);
20 }
```

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
9     .idade;
10    else return nome > other.nome;
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
4 // characters
5 int countPermLess(vector<int> freq, const string &t)
6 {
7     int n = t.size();
8     int ans = 0;
9
10    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
11    for (int i = 1; i <= n; i++)
12        fact[i] = (fact[i - 1] * i) % MOD;
13    invfact[n] = fexp(fact[n], MOD - 2, MOD);
14    for (int i = n - 1; i >= 0; i--)
15        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD
16 ;
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            if (freq[t[i] - 'a'] == 0) break;
31            freq[t[i] - 'a']--;
32        }
33    }
34    return ans;
35 }

```

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 }

```

```

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(), pp) == -1)
68         {
69             // Í -1 pq eu nÃo quero excluir os
70             colineares
71             L.pop_back();
72         }
73         L.push_back(pp);
74     }
75
76     reverse(p.begin(), p.end());
77
78     //Upper
79     for(auto pp : p)
80     {
81         while(U.size() >= 2 and ccw(U[U.size()-2], U.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 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     int n;
113     cin >> n;
114
115     vector<point> v(n);
116     for(int i = 0; i < n; i++)
117     {
118         cin >> v[i].x >> v[i].y;
119     }
120 }
```

```

121     vector<point> ch = convex_hull(v);
122
123     cout << ch.size() << '\n';
124     for(auto p : ch) cout << p.x << " " << p.y << "\n"
125     ";
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
8 ==-1 or z==-1));
9
10 bool inside(vp &p, point e){ // ccw
11     int l=2, r=(int)p.size()-1;
12     while(l<r){
13         int mid = (l+r)/2;
14         if(ccw(p[0], p[mid], e) == 1)
15             l=mid+1;
16         else{
17             r=mid;
18         }
19     }
20     // bordo
21     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e
22 ==0) return false;
23     // if(r==2 and ccw(p[0], p[1], e)==0) return
24     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 }
28
29 // Any O(n)
30
31 int inside(vp &p, point pp){
32     // 1 - inside / 0 - boundary / -1 - outside
33     int n = p.size();
34     for(int i=0; i<n; i++){
35         int j = (i+1)%n;
36         if(line({p[i], p[j]}).inside_seg(pp))
37             return 0;
38     }
39     int inter = 0;
40     for(int i=0; i<n; i++){
41         int j = (i+1)%n;
42         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
43 [i], p[j], pp)==1)
44             inter++; // up
45         else if(p[j].x <= pp.x and pp.x < p[i].x and
46 ccw(p[i], p[j], pp)==-1)
47             inter++; // down
48     }
49 }
```

10.3 Point Location

```

1
2 int32_t main(){
3     sws;
```

```
4           3 }
5     int t; cin >> t;
6
7   while(t--){
8
8     int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1
9       >> x2 >> y2 >> x3 >> y3;
10
11   int deltax1 = (x1-x2), deltay1 = (y1-y2);
12
13   int compx = (x1-x3), compy = (y1-y3);
14
15   int ans = (deltax1*compy) - (compx*deltay1);
16
17   if(ans == 0){cout << "TOUCH\n"; continue;}
18   if(ans < 0){cout << "RIGHT\n"; continue;}
19   if(ans > 0){cout << "LEFT\n"; continue;}
20
21 }   return 0;
22 }
```

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

10.4 Lattice Points