Competitive Programming Notebook

Programadores Roblox

Contents

1 String

1.1 Trie Ponteiros

```cpp
// Trie por ponteiros
// Inserção, busca e consulta de prefixo em O(N)

struct Node {
    Node *filhos[26] = {};
    bool acaba = false;
    int contador = 0;
};

void insere(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        cur->contador++;
        if(cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        cur->filhos[c - 'a'] = new Node();
        cur = cur->filhos[c - 'a'];
    }
    cur->contador++;
    cur->acaba = true;
}

bool busca(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        return false;
    }
    return cur->acaba;
}

// Retorna se é prefixo e quantas strings tem s como
//  prefixo
int isPref(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        return -1;
    }
    return cur->contador;
}
```

1.2 Trie

```cpp
// Trie por array
// Inserção, busca e consulta de prefixo em O(N)

int trie[MAXN][26];
int tot_nos = 0;
vector<bool> acaba(MAXN, false);
vector<int> contador(MAXN, 0);

void insere(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0) {
            trie[no][c - 'a'] = ++tot_nos;
        }
        no = trie[no][c - 'a'];
        contador[no]++;
```

```cpp
    }
    acaba[no] = true;
}

bool busca(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0) {
            return false;
        }
        no = trie[no][c - 'a'];
    }
    return acaba[no];
}

int isPref(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0){
            return -1;
        }
        no = trie[no][c - 'a'];
    }
    return contador[no];
}
```

1.3 Hashing

```cpp
// String Hash template
// constructor(s) - O(|s|)
// query(l, r) - returns the hash of the range [l,r]
//     from left to right - O(1)
// query_inv(l, r) from right to left - O(1)
// patrocinado por tiagodfs

struct Hash {
    const int X = 2147483647;
    const int MOD = 1e9+7;
    int n; string s;
    vector<int> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X) % MOD;
    }
    int query(int l, int r) {
        int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

1.4 Lcs

```cpp
int lcs(string &s1, string &s2) {
    int m = s1.size();
    int n = s2.size();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
```

```
7      for (int i = 1; i <= m; ++i) {
8          for (int j = 1; j <= n; ++j) {
9              if (s1[i - 1] == s2[j - 1])
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11             else
12                 dp[i][j] = max(dp[i - 1][j], dp[i][j
   - 1]);
13         }
14     }
15
16     return dp[m][n];
17 }
```

1.5 Countpermutations

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

1.6 Z Function

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

```
18 }
```

1.7 Kmp

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

2 String copy

2.1 Trie Ponteiros

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

2.2   Hashing

```
1  // String Hash template
2  // constructor(s) - O(|s|)
3  // query(l, r) - returns the hash of the range [l,r]
       from left to right - O(1)
4  // query_inv(l, r) from right to left - O(1)
5  // patrocinado por tiagodfs
6
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     } int query_inv(int l, int r) {
26     int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
27        +1] % MOD : 0));
28         return hash < 0 ? hash + MOD : hash;
29     }
30 };
```

2.3   Lcs

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

2.4   Countpermutations

```
1  // Returns the number of distinct permutations
2  // that are lexicographically less than the string t
3  // using the provided frequency (freq) of the
       characters
4  // O(n*freq.size())
5  int countPermLess(vector<int> freq, const string &t)
        {
6      int n = t.size();
7      int ans = 0;
```

```
8
9      vector<int> fact(n + 1, 1), invfact(n + 1, 1);
10     for (int i = 1; i <= n; i++)
11         fact[i] = (fact[i - 1] * i) % MOD;
12     invfact[n] = fexp(fact[n], MOD - 2, MOD);
13     for (int i = n - 1; i >= 0; i--)
14         invfact[i] = (invfact[i + 1] * (i + 1)) % MOD
        ;
15
16     // For each position in t, try placing a letter
        smaller than t[i] that is in freq
17     for (int i = 0; i < n; i++) {
18         for (char c = 'a'; c < t[i]; c++) {
19             if (freq[c - 'a'] > 0) {
20                 freq[c - 'a']--;
21                 int ways = fact[n - i - 1];
22                 for (int f : freq)
23                     ways = (ways * invfact[f]) % MOD;
24                 ans = (ans + ways) % MOD;
25                 freq[c - 'a']++;
26             }
27         }
28         if (freq[t[i] - 'a'] == 0) break;
29         freq[t[i] - 'a']--;
30     }
31     return ans;
32 }
```

2.5   Z Function

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

2.6   Kmp

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

3   DS

3.1   Segtree Iterativa

```
1  // Exemplo de uso:
2  // SegTree<int> st(vetor);
```

```
3  // range query e point update
4
5  template <typename T>
6  struct SegTree {
7      int n;
8      vector<T> tree;
9      T neutral_value = 0;
10     T combine(T a, T b) {
11         return a + b;
12     }
13
14     SegTree(const vector<T>& data) {
15         n = data.size();
16         tree.resize(2 * n, neutral_value);
17
18         for (int i = 0; i < n; i++)
19             tree[n + i] = data[i];
20
21         for (int i = n - 1; i > 0; --i)
22             tree[i] = combine(tree[i * 2], tree[i * 2
    + 1]);
23     }
24
25     T range_query(int l, int r) {
26         T res_l = neutral_value, res_r =
    neutral_value;
27
28         for (l += n, r += n + 1; l < r; l >>= 1, r
    >>= 1) {
29             if (l & 1) res_l = combine(res_l, tree[l
    ++]);
30             if (r & 1) res_r = combine(tree[--r],
    res_r);
31         }
32
33         return combine(res_l, res_r);
34     }
35
36     void update(int pos, T new_val) {
37         tree[pos += n] = new_val;
38
39         for (pos >>= 1; pos > 0; pos >>= 1)
40             tree[pos] = combine(tree[2 * pos], tree[2
     * pos + 1]);
41     }
42 };
```

3.2   Segtree Gcd

```
1  int gcd(int a, int b) {
2      if (b == 0)
3          return a;
4      return gcd(b, a % b);
5  }
6
7  class SegmentTreeGCD {
8  private:
9      vector<int> tree;
10     int n;
11
12     void build(const vector<int>& arr, int node, int
     start, int end) {
13         if (start == end) {
14             tree[node] = arr[start];
15         } else {
16             int mid = (start + end) / 2;
17             build(arr, 2 * node + 1, start, mid);
18             build(arr, 2 * node + 2, mid + 1, end);
19             tree[node] = gcd(tree[2 * node + 1], tree
    [2 * node + 2]);
20         }
21     }
22
```

```
23     void update(int node, int start, int end, int idx
    , int value) {
24         if (start == end) {
25             tree[node] = value;
26         } else {
27             int mid = (start + end) / 2;
28             if (idx <= mid) {
29                 update(2 * node + 1, start, mid, idx,
     value);
30             } else {
31                 update(2 * node + 2, mid + 1, end,
    idx, value);
32             }
33             tree[node] = gcd(tree[2 * node + 1], tree
    [2 * node + 2]);
34         }
35     }
36
37     int query(int node, int start, int end, int l,
    int r) {
38         if (r < start || l > end) {
39             return 0;
40         }
41         if (l <= start && end <= r) {
42             return tree[node];
43         }
44         int mid = (start + end) / 2;
45         int left_gcd = query(2 * node + 1, start, mid
    , l, r);
46         int right_gcd = query(2 * node + 2, mid + 1,
    end, l, r);
47         return gcd(left_gcd, right_gcd);
48     }
49
50 public:
51     SegmentTreeGCD(const vector<int>& arr) {
52         n = arr.size();
53         tree.resize(4 * n);
54         build(arr, 0, 0, n - 1);
55     }
56     void update(int idx, int value) {
57         update(0, 0, n - 1, idx, value);
58     }
59     int query(int l, int r) {
60         return query(0, 0, n - 1, l, r);
61     }
62 };
```

3.3   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
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(L.begin(), L.end(), R.begin(), R.end(),
     tree[x].begin());
```

```
22          }
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(), v
    .end(), r);
28          }
29          if (rx < l || lx > r) {
30              return 0;
31          }
32          int mid = lx + (rx - lx)/2;
33          return query(2 * x, lx, mid, l, r) + query(2
    * x + 1, mid + 1, rx, l, r);
34      }
35
36      int query(int l, int r) {
37          return query(1, 0, n - 1, l, r);
38      }
39 }
40
41 // Checar se o range é todo distinto
42 // Cada cara e sua próxima aparição a direita,
       conta quantos caras que a próxima aparição a
       direita ta dentro do range ainda
43 vector<int> nr(n);
44 map<int, int> mp;
45 for (int i = n - 1; i >= 0; i--) {
46      auto it = mp.find(a[i]);
47      nr[i] = it != mp.end() ? it->second : n;
48      mp[a[i]] = i;
49 }
50 SegTree seg(nr);
```

3.4   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,
       null_type, less_equal<T>, rb_tree_tag,
       tree_order_statistics_node_update>;
8 template <typename T> using o_set = tree<T, null_type
    , less<T>, rb_tree_tag,
       tree_order_statistics_node_update>;
9 template <typename T, typename R> using o_map = tree<
    T, R, less<T>, rb_tree_tag,
       tree_order_statistics_node_update>;
10
11 int main() {
12   int i, j, k, n, m;
13   o_set<int>st;
14   st.insert(1);
15   st.insert(2);
16   cout << *st.find_by_order(0) << endl; /// k-esimo
     elemento
17   cout << st.order_of_key(2) << endl; ///numero de
     elementos menores que k
18   o_map<int, int>mp;
19   mp.insert({1, 10});
20   mp.insert({2, 20});
21   cout << mp.find_by_order(0)->second << endl; /// k-
     esimo elemento
22   cout << mp.order_of_key(2) << endl; /// numero de
     elementos (chave) menores que k
23   return 0;
24 }
```

3.5   Sparse Table

```
1 // 1-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 - 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 };
```

3.6   Psum 2d

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

3.7   Segtree Sum

```
1 struct SegTree {
2     ll merge(ll a, ll b) { return a + b; }
3     const ll neutral = 0;
4     int n;
5     vector<ll> t, lazy;
6     vector<bool> replace;
7     inline int lc(int p) { return p * 2; }
8     inline int rc(int p) { return p * 2 + 1; }
9     void push(int p, int l, int r) {
10         if (replace[p]) {
11             t[p] = lazy[p] * (r - l + 1);
12             if (l != r) {
13                 lazy[lc(p)] = lazy[p];
14                 lazy[rc(p)] = lazy[p];
15                 replace[lc(p)] = true;
16                 replace[rc(p)] = true;
17             }
18         } else if (lazy[p] != 0) {
19             t[p] += lazy[p] * (r - l + 1);
20             if (l != r) {
21                 lazy[lc(p)] += lazy[p];
22                 lazy[rc(p)] += lazy[p];
23             }
```

```
24        }
25        replace[p] = false;
26        lazy[p] = 0;
27    }
28    void build(int p, int l, int r, const vector<ll>
      &v) {
29        if (l == r) {
30            t[p] = v[l];
31        } else {
32            int mid = (l + r) / 2;
33            build(lc(p), l, mid, v);
34            build(rc(p), mid + 1, r, v);
35            t[p] = merge(t[lc(p)], t[rc(p)]);
36        }
37    }
38    void build(int _n) {
39        n = _n;
40        t.assign(n * 4, neutral);
41        lazy.assign(n * 4, 0);
42        replace.assign(n * 4, false);
43    }
44    void build(const vector<ll> &v) {
45        n = (int)v.size();
46        t.assign(n * 4, neutral);
47        lazy.assign(n * 4, 0);
48        replace.assign(n * 4, false);
49        build(1, 0, n - 1, v);
50    }
51    void build(ll *bg, ll *en) {
52        build(vector<ll>(bg, en));
53    }
54    ll query(int p, int l, int r, int L, int R) {
55        push(p, l, r);
56        if (l > R || r < L) return neutral;
57        if (l >= L && r <= R) return t[p];
58        int mid = (l + r) / 2;
59        auto ql = query(lc(p), l, mid, L, R);
60        auto qr = query(rc(p), mid + 1, r, L, R);
61        return merge(ql, qr);
62    }
63    ll query(int l, int r) { return query(1, 0, n -
      1, l, r); }
64    void update(int p, int l, int r, int L, int R, ll
       val, bool repl = 0) {
65        push(p, l, r);
66        if (l > R || r < L) return;
67        if (l >= L && r <= R) {
68            lazy[p] = val;
69            replace[p] = repl;
70            push(p, l, r);
71        } else {
72            int mid = (l + r) / 2;
73            update(lc(p), l, mid, L, R, val, repl);
74            update(rc(p), mid + 1, r, L, R, val, repl
      );
75            t[p] = merge(t[lc(p)], t[rc(p)]);
76        }
77    }
78    void sumUpdate(int l, int r, ll val) { update(1,
      0, n - 1, l, r, val, 0); }
79    void assignUpdate(int l, int r, ll val) { update
      (1, 0, n - 1, l, r, val, 1); }
80 } segsum;
```

3.8 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 };
```

3.9 Bit

```
1 class BIT {
2     vector<int> bit;
3     int n;
4     int sum(int idx) {
5         int result = 0;
6         while (idx > 0) {
7             result += bit[idx];
8             idx -= idx & -idx;
9         }
10        return result;
11    }
12
13 public:
14    BIT(int size) {
15        n = size;
16        bit.assign(n + 1, 0);   // BIT indexada em 1
17    }
18    void update(int idx, int delta) {
19        while (idx <= n) {
20            bit[idx] += delta;
21            idx += idx & -idx;
22        }
23    }
24    int query(int idx) {
25        return sum(idx);
26    }
27    int range_query(int l, int r) {
28        return sum(r) - sum(l - 1);
29    }
30 };
31
32 BIT fenwick(n);
33 for(int i = 1; i <= n; i++) {
34     fenwick.update(i, arr[i]);
35 }
```

4 Search and sort

4.1 Pilha Monotonic

```
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  }
18
19  // maior = true -> encontra o primeiro maior Ãã
20      direita
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
            [i] : v[s.top()] >= v[i])) {
27              s.pop();
28          }
29          if (s.empty()) {
30              result[i] = -1;
31          } else {
32              result[i] = v[s.top()];
33          }
34          s.push(i);
35      }
36      return result;
37  }
```

4.2 Mergeandcount

```
 1
 2  // Realiza a mesclagem de dois subarrays e conta o
        nÃžmero de trocas necessÃąrias.
 3  int mergeAndCount(vector<int>& v, int l, int m, int r
        ) {
 4      int x = m - l + 1;  // Tamanho do subarray
        esquerdo.
 5      int y = r - m;  // Tamanho do subarray direito.
 6
 7      // Vetores temporarios para os subarray esquerdo
        e direito.
 8      vector<int> left(x), right(y);
 9
10      for (int i = 0; i < x; i++) left[i] = v[l + i];
11      for (int j = 0; j < y; j++) right[j] = v[m + 1 +
        j];
12
13      int i = 0, j = 0, k = l;
14      int swaps = 0;
15
16      while (i < x && j < y) {
17          if (left[i] <= right[j]) {
18              // Se o elemento da esquerda for menor ou
         igual, coloca no vetor original.
19              v[k++] = left[i++];
20          } else {
21              // Caso contrario, coloca o elemento da
        direita e conta as trocas.
22              v[k++] = right[j++];
23              swaps += (x - i);
24          }
25      }
26
```

```
27      // Adiciona os elementos restantes do subarray
        esquerdo (se houver).
28      while (i < x) v[k++] = left[i++];
29
30      // Adiciona os elementos restantes do subarray
        direito (se houver).
31      while (j < y) v[k++] = right[j++];
32
33      return swaps;  // Retorna o numero total de
        trocas realizadas.
34  }
35
36  int mergeSort(vector<int>& v, int l, int r) {
37      int swaps = 0;
38
39      if (l < r) {
40          // Encontra o ponto medio para dividir o
        vetor.
41          int m = l + (r - l) / 2;
42
43          // Chama merge sort para a metade esquerda.
44          swaps += mergeSort(v, l, m);
45          // Chama merge sort para a metade direita.
46          swaps += mergeSort(v, m + 1, r);
47
48          // Mescla as duas metades e conta as trocas.
49          swaps += mergeAndCount(v, l, m, r);
50      }
51
52      return swaps;  // Retorna o numero total de
        trocas no vetor.
53  }
```

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 Discrete Log

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

5.4 Segment Sieve

```
1 // Retorna quantos primos tem entre [l, r] (inclusivo
     )
2 // precisa de um vetor com os primos atÃ© sqrt(r)
```

```
3 int seg_sieve(int l, int r){
4     if (l > r) return 0;
5     vector<bool> is_prime(r - l + 1, true);
6     if (l == 1) is_prime[0] = false;
7
8     for (int p : primos){
9         if (p * p > r) break;
10         int start = max(p * p, (l + p - 1) / p * p);
11         for (int j = start; j <= r; j += p){
12             if (j >= l) {
13                 is_prime[j - l] = false;
14             }
15         }
16     }
17
18     return accumulate(all(is_prime), 0ll);;
19 }
```

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

5.6 Menor Fator Primo

```
1 const int MAXN = 1000001; // Limite para o Crivo.
2 int spf[MAXN];
3 vector<int> primos;
4
5 void crivo() {
6     for (int i = 2; i * i < MAXN; i++) {
7         if (spf[i] == i) {
8             for (int j = i * i; j < MAXN; j += i) {
9                 if (spf[j] == j) {
10                     spf[j] = i;
11                 }
12             }
13         }
14     }
15     for (int i = 2; i < MAXN; i++) {
16         if (spf[i] == i) {
17             primos.push_back(i);
18         }
19     }
20 }
21
22 map<int, int> fatora(int n) {
```

```
23      map<int, int> fatores;
24      while (n > 1) {
25          fatores[spf[n]]++;
26          n /= spf[n];
27      }
28      return fatores;
29  }
30
31  int numero_de_divisores(int n) {
32      if (n == 1) return 1;
33      map<int, int> fatores = fatorar(n);
34      int nod = 1;
35      for (auto &[primo, expoente] : fatores) nod *= (
        expoente + 1);
36      return nod;
37  }
38
39  // DEFINE INT LONG LONG
40  int soma_dos_divisores(int n) {
41      if (n == 1) return 1;
42      map<int, int> fatores = fatorar(n);
43      int sod = 1;
44      for (auto &[primo, expoente] : fatores) {
45          int termo_soma = 1;
46          int potencia_primo = 1;
47          for (int i = 0; i < expoente; i++) {
48              potencia_primo *= primo;
49              termo_soma += potencia_primo;
50          }
51          sod *= termo_soma;
52      }
53      return sod;
54  }
```

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

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

6.3 Eulerian Path

```
1   /**
2    * Versão que assume: #define int long long
3    *
4    * Retorna um caminho/ciclo euleriano em um grafo (se
         existir).
5    * - g: lista de adjacência (vector<vector<int>>).
6    * - directed: true se o grafo for dirigido.
7    * - s: vértice inicial.
8    * - e: vértice final (opcional). Se informado,
         tenta caminho de s até e.
9    * - O(Nlog(N))
10   * Retorna vetor com a sequência de vértices, ou
         vazio se impossível.
11   */
12  vector<int> eulerian_path(const vector<vector<int>>&
        g, bool directed, int s, int e = -1) {
13      int n = (int)g.size();
14      // cópia das adjacências em multiset para
        permitir remoção específica
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 {}; //
        impossível
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[u] & 1))) {
38              return {};
39          }
40      }
41      while (!st.empty()) {
42          int u = st.top();
43          if (h[u].empty()) {
44              result.emplace_back(u);
45              st.pop();
46          } else {
47              int v = *h[u].begin();
48              auto it = h[u].find(v);
49              if (it != h[u].end()) h[u].erase(it);
50              --in_degree[v];
51              if (!directed) {
```

```
52              auto it2 = h[v].find(u);
53              if (it2 != h[v].end()) h[v].erase(it2
   );
54              --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   Khan

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

6.5   Topological Sort

```
1  vector<int> adj[MAXN];
2  vector<int> estado(MAXN); // 0: nao visitado 1:
   processamento 2: processado
3  vector<int> ordem;
4  bool temCiclo = false;
5
6  void dfs(int v) {
7      if(estado[v] == 1) {
8          temCiclo = true;
9          return;
10     }
11     if(estado[v] == 2) return;
```

```
12     estado[v] = 1;
13     for(auto &nei : adj[v]) {
14         if(estado[v] != 2) dfs(nei);
15     }
16     estado[v] = 2;
17     ordem.push_back(v);
18     return;
19 }
```

6.6   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 }
```

6.7   Edmonds-karp

```
1  // Edmonds-Karp com scalling O(Eálog(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      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 = 1ll << 62;
38
39      while (scalling > 0) {
40          while (new_flow = bfs(s, t, scalling, 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          scalling /= 2;
52      }
53
54      return flow;
55  }
```

6.8 Kruskal

```
1  // Ordena as arestas por peso,  insere se ja nao
     estiver no mesmo componente
2  // O(E log E)
3
4  struct DSU {
5      vector<int> par, rank, sz;
6      int c;
7      DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
    1, 1), c(n) {
8          for (int i = 1; i <= n; ++i) par[i] = i;
9      }
10     int find(int i) {
11         return (par[i] == i ? i : (par[i] = find(par[
    i])));
12     }
13     bool same(int i, int j) {
14         return find(i) == find(j);
15     }
16     int get_size(int i) {
17         return sz[find(i)];
18     }
19     int count() {
20         return c;  // quantos componentes conexos
21     }
22     int merge(int i, int j) {
23         if ((i = find(i)) == (j = find(j))) return
    -1;
24         else --c;
25         if (rank[i] > rank[j]) swap(i, j);
26         par[i] = j;
27         sz[j] += sz[i];
28         if (rank[i] == rank[j]) rank[j]++;
29         return j;
```

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

6.9 Bellman Ford

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

6.10 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] Ãľ o 2^j-
    Ãľsimo ancestral de v
8
9  void dfs(int v, int p, int d) {
10     profundidade[v] = d;
```

```cpp
        cima[v][0] = p; // o pai direto é o 2^0-ésimo
    ancestral
        for (int j = 1; j < LOG; j++) {
            // se o ancestral 2^(j-1) existir, calculamos
     o 2^j
            if (cima[v][j - 1] != -1) {
                cima[v][j] = cima[cima[v][j - 1]][j - 1];
            } else {
                cima[v][j] = -1; // não tem ancestral
    superior
            }
        }
        for (int nei : adj[v]) {
            if (nei != p) {
                dfs(nei, v, d + 1);
            }
        }
    }
}

void build(int root) {
    LOG = ceil(log2(N));
    profundidade.assign(N + 1, 0);
    cima.assign(N + 1, vector<int>(LOG, -1));
    dfs(root, -1, 0);
}

int get_lca(int a, int b) {
    if (profundidade[a] < profundidade[b]) {
        swap(a, b);
    }
    // sobe 'a' até a mesma profundidade de 'b'
    for (int j = LOG - 1; j >= 0; j--) {
        if (profundidade[a] - (1 << j) >=
    profundidade[b]) {
            a = cima[a][j];
        }
    }
    // se 'b' era um ancestral de 'a', então 'a'
    agora é igual a 'b'
    if (a == b) {
        return a;
    }

    // sobe os dois nós juntos até encontrar os
    filhos do LCA
    for (int j = LOG - 1; j >= 0; j--) {
        if (cima[a][j] != -1 && cima[a][j] != cima[b
    ][j]) {
            a = cima[a][j];
            b = cima[b][j];
        }
    }
    return cima[a][0];
}
```

6.11   Lca

```cpp
// LCA - CP algorithm
// preprocessing O(NlogN)
// lca O(logN)
// Uso: criar LCA com a quantidade de vértices (n) e
    lista de adjacência (adj)
// chamar a função preprocess com a raiz da árvore

struct LCA {
    int n, l, timer;
    vector<vector<int>> adj;
    vector<int> tin, tout;
    vector<vector<int>> up;

    LCA(int n, const vector<vector<int>>& adj) : n(n)
    , adj(adj) {}
```

```cpp
    void dfs(int v, int p) {
        tin[v] = ++timer;
        up[v][0] = p;
        for (int i = 1; i <= l; ++i)
            up[v][i] = up[up[v][i-1]][i-1];

        for (int u : adj[v]) {
            if (u != p)
                dfs(u, v);
        }

        tout[v] = ++timer;
    }

    bool is_ancestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v
    ];
    }

    int lca(int u, int v) {
        if (is_ancestor(u, v))
            return u;
        if (is_ancestor(v, u))
            return v;
        for (int i = l; i >= 0; --i) {
            if (!is_ancestor(up[u][i], v))
                u = up[u][i];
        }
        return up[u][0];
    }

    void preprocess(int root) {
        tin.resize(n);
        tout.resize(n);
        timer = 0;
        l = ceil(log2(n));
        up.assign(n, vector<int>(l + 1));
        dfs(root, root);
    }
};
```

6.12   Kosaraju

```cpp
bool vis[MAXN];
vector<int> order;
int component[MAXN];
int N, m;
vector<int> adj[MAXN], adj_rev[MAXN];

// dfs no grafo original para obter a ordem (pós-
    order)
void dfs1(int u) {
    vis[u] = true;
    for (int v : adj[u]) {
        if (!vis[v]) {
            dfs1(v);
        }
    }
    order.push_back(u);
}

// dfs o grafo reverso para encontrar os SCCs
void dfs2(int u, int c) {
    component[u] = c;
    for (int v : adj_rev[u]) {
        if (component[v] == -1) {
            dfs2(v, c);
        }
    }
}

int kosaraju() {
    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    }
```

6.13   Pega Ciclo

```
1  // encontra um ciclo em g (direcionado ou nÃčo)
2  // g[u] = vector<pair<id_aresta, vizinho>>
3  // rec_arestas: true -> retorna ids das arestas do
       ciclo; false -> retorna vÃŕtices do ciclo
4  // directed: grafo direcionado?
5
6  const int MAXN = 5 * 1e5 + 2;
7  vector<pair<int, int>> g[MAXN];
8  int N;
9  bool DIRECTED = false;
10 vector<int> color(MAXN), parent(MAXN, -1), edgein(
       MAXN, -1); // color: 0,1,2 ; edgein[v] = id da
       aresta que entra em v
11 int ini_ciclo = -1, fim_ciclo = -1, back_edge_id =
       -1;
12
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 nÃčo-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 (vazio se nÃčo
       hÃą)
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             // reconstrÃşi 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
                  , ..., u]
49             reverse(path.begin(), path.end());
50             if (!rec_arestas) return path;
51             // converte path ids das arestas: edgein[
                  node] Ãŕ a aresta que entra em node
52             vector<int> edges;
53             for (int i = 1; i < path.size(); i++)
                  edges.push_back(edgein[path[i]]);
54             // adiciona a aresta de retorno u -> v
55             edges.push_back(back_edge_id);
56             return edges;
57         }
58     }
59     return {};
60 }
```

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 Ãŕ >= v[i].
10             // subsequÃŧncia de mesmo comprimento,
                  mas com um final menor.
11             auto it = lower_bound(lis.begin(), lis.
                  end(), v[i]);
12             *it = v[i];
13         }
14     }
15     return lis.size();
16 }
17
18 // LIS NA ARVORE
19 const int MAXN_TREE = 100001;
20 vector<int> adj[MAXN_TREE];
21 int values[MAXN_TREE];
22 int ans = 0;
23
24
25 void dfs(int u, int p, vector<int>& tails) {
26     auto it = lower_bound(tails.begin(), tails.end(),
            values[u]);
27     int prev = -1;
28     bool coloquei = false;
29     if (it == tails.end()) {
30         tails.push_back(values[u]);
31         coloquei = true;
32     } else {
33         prev = *it;
34         *it = values[u];
35     }
36     ans = max(ans, (int)tails.size());
37     for (int v : adj[u]) {
38         if (v != p) {
39             dfs(v, u, tails);
40         }
41     }
42     if (coloquei) {
43         tails.pop_back();
44     } else {
```

```
45        *it = prev;
46    }
47 }
```

8.2 Edit Distance

```
1    vector<vector<int>> dp(n+1, vector<int>(m+1, LINF
     ));
2
3    for(int j = 0; j <= m; j++) {
4        dp[0][j] = j;
5    }
6
7    for(int i = 0; i <= n; i++) {
8        dp[i][0] = i;
9    }
10
11   for(int i = 1; i <= n; i++) {
12       for(int j = 1; j <= m; j++) {
13           if(a[i-1] == b[j-1]) {
14               dp[i][j] = dp[i-1][j-1];
15           } else {
16               dp[i][j] = min({dp[i-1][j] + 1, dp[i
     ][j-1] + 1, dp[i-1][j-1] + 1});
17           }
18       }
19   }
20
21   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], 0ll, 0ll, 0ll}) - S.begin();
4 }
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][2]);
16    return dp[i][vis] = ans;
17 }
```

8.4 Lcs

```
1 string s1, s2;
2 int dp[1001][1001];
3
4 int lcs(int i, int j) {
5     if (i < 0 || j < 0) return 0;
6     if (dp[i][j] != -1) return dp[i][j];
7     if (s1[i] == s2[j]) {
8         return dp[i][j] = 1 + lcs(i - 1, j - 1);
9     } else {
10        return dp[i][j] = max(lcs(i - 1, j), lcs(i, j
     - 1));
11    }
12 }
```

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][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
     mochila
2 // O(N * W)
3
4 for(int j = 0; j < MAXN; j++) {
5     dp[0][j] = 0;
6 }
7 for(int i = 1; i <= N; i++) {
8     for(int j = 0; j <= W; j++) {
9         if(items[i].first > j) {
10            dp[i][j] = dp[i-1][j];
11        }
12        else {
13            dp[i][j] = max(dp[i-1][j], dp[i-1][j-
     items[i].first] + items[i].second);
14        }
15    }
16 }
```

8.7 Lis Seg

```
1    vector<int> a(n);
2    for (int i = 0; i < n; i++) cin >> a[i];
3    vector<int> sorted_a = a;
4    sort(sorted_a.begin(), sorted_a.end());
5    for (int i = 0; i < n; i++) {
6        a[i] = lower_bound(sorted_a.begin(), sorted_a
     .end(), a[i]) - sorted_a.begin();
7    }
8    SegTreeMx segmx;
9    segmx.build(n);
10   vector<int> dp(n, 1);
11   for (int k = 0; k < n; k++) {
12       if (a[k] > 0) {
13           dp[k] = segmx.query(0, a[k] - 1) + 1;
14       }
15       segmx.update(a[k], dp[k]);
16   }
17   cout << *max_element(dp.begin(), dp.end()) << '\n
     ';
```

8.8 Disjoint Blocks

```
1 // Número máximo de subarrays disjuntos com soma x
     usando apenas
2 // prefixo até 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 esse pref
6       best[0] = 0;
7       int pref = 0;
8       vector<int> dp(n + 1, 0); // dp[0] = 0
9       for (int i = 1; i <= n; i++) {
10          pref += a[i - 1];
11          // nÃčo pegar subarray terminando em i
12          dp[i] = dp[i-1];
13          // pega se existir prefixo anterior e
        atualiza best
14          auto it = best.find(pref - x);
15          if (it != best.end()) {
16              dp[i] = max(dp[i], it->second + 1);
17          }
18          best[pref] = max(best[pref], dp[i]);
19      }
20      return dp[n];
21  }
```

9 General

9.1 Brute Choose

```
1  vector<int> elements;
2  int N, K;
3  vector<int> comb;
4
5
6  void brute_choose(int i) {
7      if (comb.size() == K) {
8          for (int j = 0; j < comb.size(); j++) {
9              cout << comb[j] << ' ';
10          }
11          cout << '\n';
12          return;
13      }
14      if (i == N) return;
15      int r = N - i;
16      int preciso = K - comb.size();
17      if (r < preciso) return;
18      comb.push_back(elements[i]);
19      brute_choose(i + 1);
20      comb.pop_back();
21      brute_choose(i + 1);
22  }
```

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 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(), pp) == -1)
68          {
69              // Ãl' -1 pq eu nÃčo quero excluir os
     colineares
70              L.pop_back();
71          }
72          L.push_back(pp);
73      }
74
75      reverse(p.begin(), p.end());
76
77      //Upper
78      for(auto pp : p)
79      {
80          while(U.size() >= 2 and ccw(U[U.size()-2], U
     .back(), pp) == -1)
81          {
82              U.pop_back();
83          }
84          U.push_back(pp);
85      }
86
87      L.pop_back();
88      L.insert(L.end(), U.begin(), U.end()-1);
89      return L;
90  }
91
92  cod area(vector<point> v)
93  {
94      int ans = 0;
95      int aux = (int)v.size();
96      for(int i = 2; i < aux; i++)
97      {
98          ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
99      }
100     ans = abs(ans);
101     return ans;
```

```
102  }
103
104  int bound(point p1 , point p2)
105  {
106      return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
107  }
108  //teorema de pick [pontos = A - (bound+points)/2 + 1]
109
110  int32_t main()
111  {
112
113      int n;
114      cin >> n;
115
116      vector<point> v(n);
117      for(int i = 0; i < n; i++)
118      {
119          cin >> v[i].x >> v[i].y;
120      }
121
122      vector <point> ch = convex_hull(v);
123
124      cout << ch.size() <<  '\n';
125      for(auto p : ch) cout << p.x << " " << p.y << "\n
     ";
126
127      return 0;
128  }
```

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==-1));
8   }
9
10  bool inside(vp &p, point e){ // ccw
11      int l=2, r=(int)p.size()-1;
12      while(l<r){
13          int mid = (l+r)/2;
14          if(ccw(p[0], p[mid], e) == 1)
15              l=mid+1;
16          else{
17              r=mid;
18          }
19      }
20      // bordo
21      // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
     ==0) return false;
22      // if(r==2 and ccw(p[0], p[1], e)==0) return
     false;
23      // if(ccw(p[r], p[r-1], e)==0) return false;
24      return insideT(p[0], p[r-1], p[r], e);
25  }
26
27
28  // Any O(n)
29
30  int inside(vp &p, point pp){
31      // 1 - inside / 0 - boundary / -1 - outside
32      int n = p.size();
33      for(int i=0;i<n;i++){
34          int j = (i+1)%n;
35          if(line({p[i], p[j]}).inside_seg(pp))
36              return 0;
37      }
38      int inter = 0;
39      for(int i=0;i<n;i++){
```

```
40        int j = (i+1)%n;
41        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
   [i], p[j], pp)==1)
42            inter++; // up
43        else if(p[j].x <= pp.x and pp.x < p[i].x and
   ccw(p[i], p[j], pp)==-1)
44            inter++; // down
45    }
46
47    if(inter%2==0) return -1; // outside
48    else return 1; // inside
49 }
```

10.3   Point Location

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

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 - y2));
6  }
7  ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
8      return gcd(abs(x2 - x1), abs(y2 - y1));
9  }
10
11 int32_t main() {
12     ll x1, y1, x2, y2, x3, y3;
13     cin >> x1 >> y1;
14     cin >> x2 >> y2;
15     cin >> x3 >> y3;
16     ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
17     ll tot_borda = pontos_borda(x1, y1, x2, y2) +
   pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,
   y3, x1, y1);
18
19     ll ans = (area - tot_borda) / 2 + 1;
20     cout << ans << endl;
21
22     return 0;
23 }
```