Competitive Programming Notebook

Programadores Roblox

Contents

1 String

1.1 Trie

```cpp
// Trie por array
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]++;
    }
    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];
}
```

1.2 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

mt19937 rng(time(nullptr));

struct Hash {
    const int X = rng();
    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;
    }
};
```

1.3 Z Function

```cpp
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

1.4 Kmp

```cpp
vector<int> kmp(string s) {
    int n = (int)s.length();
    vector<int> p(n+1);
    p[0] = -1;
    for (int i = 1; i < n; i++) {
        int j = p[i-1];
        while (j >= 0 && s[j] != s[i-1])
            j = p[j-1];
        p[i] = j+1;
    }
    return p;
}
```

2 DS

2.1 Segtree Iterativa

```cpp
// Exemplo de uso:
// SegTree<int> st(vetor);
// range query e point update
template <typename T>
struct SegTree {
    int n;
    vector<T> tree;
    T neutral_value = 0;
    T combine(T a, T b) {
        return a + b;
    }

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

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

```
20         for (int i = n - 1; i > 0; --i)
21             tree[i] = combine(tree[i * 2], tree[i * 2
     + 1]);
22     }
23     T range_query(int l, int r) {
24         T res_l = neutral_value, res_r =
     neutral_value;
25
26         for (l += n, r += n + 1; l < r; l >>= 1, r
     >>= 1) {
27             if (l & 1) res_l = combine(res_l, tree[l
     ++]);
28             if (r & 1) res_r = combine(tree[--r],
     res_r);
29         }
30
31         return combine(res_l, res_r);
32     }
33     void update(int pos, T new_val) {
34         tree[pos += n] = new_val;
35         for (pos >>= 1; pos > 0; pos >>= 1)
36             tree[pos] = combine(tree[2 * pos], tree[2
     * pos + 1]);
37     }
38 };
```

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         if (lx == rx) {
12             tree[x] = { a[lx] };
13             return;
14         }
15         int mid = lx + (rx - lx)/2;
16         build(2 * x, lx, mid, a);
17         build(2 * x + 1, mid + 1, rx, a);
18         auto &L = tree[2 * x], &R = tree[2 * x + 1];
19         tree[x].resize(L.size() + R.size());
20         merge(L.begin(), L.end(), R.begin(), R.end(),
     tree[x].begin());
21     }
22     int query(int x, int lx, int rx, int l, int r) {
23         if (lx >= l && rx <= r) {
24             auto &v = tree[x];
25             return v.end() - upper_bound(v.begin(), v
     .end(), r);
26         }
27         if (rx < l || lx > r) {
28             return 0;
29         }
30         int mid = lx + (rx - lx)/2;
31         return query(2 * x, lx, mid, l, r) + query(2
     * x + 1, mid + 1, rx, l, r);
32     }
33     int query(int l, int r) {
34         return query(1, 0, n - 1, l, r);
35     }
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         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,
         null_type, less_equal<T>, rb_tree_tag,
         tree_order_statistics_node_update>;
8 template <typename T> using o_set = tree<T, null_type
         , less<T>, rb_tree_tag,
         tree_order_statistics_node_update>;
9 template <typename T, typename R> using o_map = tree<
         T, R, less<T>, rb_tree_tag,
         tree_order_statistics_node_update>;
10
11 int main() {
12     int i, j, k, n, m;
13     o_set<int>st;
14     st.insert(1);
15     st.insert(2);
16     cout << *st.find_by_order(0) << endl; /// k-esimo
         elemento
17     cout << st.order_of_key(2) << endl; ///numero de
         elementos menores que k
18     o_map<int, int>mp;
19     mp.insert({1, 10});
20     mp.insert({2, 20});
21     cout << mp.find_by_order(0)->second << endl; /// k-
         esimo elemento
22     cout << mp.order_of_key(2) << endl; /// numero de
         elementos (chave) menores que k
23     return 0;
24 }
```

2.4   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 };
```

2.5   Psum 2d

```cpp
vector<vector<int>> psum(h+1, vector<int>(w+1, 0));
for (int i=1; i<=h; i++){
    for (int j=1; j<=w; j++){
        cin >> psum[i][j];
        psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[
        i-1][j-1];
    }
}
// retorna a psum2d do intervalo inclusivo [(a, b), (
    c, d)]
int retangulo(int a, int b, int c, int d){
    c = min(c, h), d = min(d, w);
    a = max(0LL, a-1), b = max(0LL, b-1);
    return v[c][d]-v[a][d]-v[c][b]+v[a][b];
}
```

2.6   Segtree Sum

```cpp
struct SegTree {
    vector<int> tree;
    vector<int> lazy;
    int n;
    SegTree(int N) {
        n = N;
        tree.resize(4 * n);
        lazy.resize(4 * n, 0);
        // build(a);
    }
    void push(int lx, int rx, int x) {
        if (lazy[x] == 0) {
            return;
        }
        // tree[x] += lazy[x];
        tree[x] += (rx - lx + 1) * lazy[x];
        if (lx != rx) {
            lazy[2 * x] += lazy[x];
            lazy[2 * x + 1] += lazy[x];
        }
        lazy[x] = 0;
    }
    void build(vector<int> &a) {
        build(0, n - 1, 1, a);
    }
    void build(int lx, int rx, int x, vector<int> &a)
     {
        if (lx == rx) {
            tree[x] = a[lx];
            return;
        }
        int mid = lx + (rx - lx) / 2;
        build(lx, mid, 2 * x, a);
        build(mid + 1, rx, 2 * x + 1, a);
        tree[x] = tree[2 * x] + tree[2 * x + 1];
    }
    void update(int l, int r, int val) {
        update(0, n - 1, 1, l, r, val);
    }
    void update(int lx, int rx, int x, int l, int r,
    int val) {
        push(lx, rx, x);
        if (rx < l || lx > r) {
            return;
        }
        if (lx >= l && rx <= r) {
            lazy[x] += val;
            push(lx, rx, x);
            return;
        }
        int mid = lx + (rx - lx) / 2;
        update(lx, mid, 2 * x, l, r, val);
        update(mid + 1, rx, 2 * x + 1, l, r, val);
        tree[x] = tree[2 * x] + tree[2 * x + 1];
    }
```

```cpp
    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

```cpp
struct DSU {
    vector<int> par, rank, sz;
    int c;
    DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
    1, 1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[
    i])));
    }
    bool same(int i, int j) {
        return find(i) == find(j);
    }
    int get_size(int i) {
        return sz[find(i)];
    }
    int count() {
        return c;   // quantos componentes conexos
    }
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return
    -1;
        else --c;
        if (rank[i] > rank[j]) swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rank[i] == rank[j]) rank[j]++;
        return j;
    }
};
```

2.8   Bit

```cpp
struct BIT {
    int n;
    vector<int> bit;
    BIT(int n = 0): n(n), bit(n + 1, 0) {}
    void add(int i, int delta) {
        for(; i <= n; i += i & -i) bit[i] += delta;
    }
    int sum(int i) {
        int r = 0;
        for(; i > 0; i -= i & -i) r += bit[i];
        return r;
    }
    int range_sum(int l, int r){
        if (r < l) return 0;
        return sum(r) - sum(l - 1);
    }
};
```

3 Search and sort

3.1 Merge Sort

```cpp
int mergeAndCount(vector<int>& v, int l, int m, int r
    ) {
    int x = m - l + 1;
    int y = r - m;
    vector<int> left(x), right(y);
    for (int i = 0; i < x; i++) left[i] = v[l + i];
    for (int j = 0; j < y; j++) right[j] = v[m + 1 +
        j];
    int i = 0, j = 0, k = l;
    int swaps = 0;
    while (i < x && j < y) {
        if (left[i] <= right[j]) {
            v[k++] = left[i++];
        } else {
            v[k++] = right[j++];
            swaps += (x - i);
        }
    }
    while (i < x) v[k++] = left[i++];
    while (j < y) v[k++] = right[j++];
    return swaps;
}

int mergeSort(vector<int>& v, int l, int r) {
    int swaps = 0;
    if (l < r) {
        int m = l + (r - l) / 2;.
        swaps += mergeSort(v, l, m);
        swaps += mergeSort(v, m + 1, r);
        swaps += mergeAndCount(v, l, m, r);
    }
    return swaps;
}
```

3.2 Monotonic Stack

```cpp
vector<int> find_esq(vector<int> &v, bool maior) {
    int n = v.size();
    vector<int> result(n);
    stack<int> s;

    for (int i = 0; i < n; i++) {
        while (!s.empty() && (maior ? v[s.top()] <= v
[i] : v[s.top()] >= v[i])) {
            s.pop();
        }
        if (s.empty()) {
            result[i] = -1;
        } else {
            result[i] = v[s.top()];
        }
        s.push(i);
    }
    return result;
}

vector<int> find_dir(vector<int> &v, bool maior) {
    int n = v.size();
    vector<int> result(n);
    stack<int> s;
    for (int i = n - 1; i >= 0; i--) {
        while (!s.empty() && (maior ? v[s.top()] <= v
[i] : v[s.top()] >= v[i])) {
            s.pop();
        }
        if (s.empty()) {
            result[i] = -1;
        } else {
            result[i] = v[s.top()];
        }
        s.push(i);
    }
    return result;
}
```

4 Stress

4.1 Gen

```cpp
// pre-compilar os headers:
// compilar template com g++ -H e procurar onde esta
    stdc++.h
// criar a pasta bits e incluir com "" ou inves de <>
// compilar stress com: g++ -pipe -O3 -flto -march=
    native -mtune=native gen.ccp
// faz bastante diferenÃğa no runtime

#include <bits/stdc++.h>
#include <cstdlib>
#include <ctime>
using namespace std;

int randi(int L, int R) { return L + rand() % (R - L
    + 1); }
char randc(char L, char R) { return char(L + rand() %
    (R - L + 1)); }

int main(int argc, char** argv) {
    if (argc > 1) srand(atoi(argv[1]));
    else srand(time(0));

    int n = randi(1, 100);
    cout << n << '\n';
    for (int i = 0; i < n; i++) {
        cout << randi(-10, 20) << ' ';
    }
    cout << '\n';
}
```

5 Math

5.1 Combinatorics

```cpp
const int MAXN_FATORIAL = 200005;
const int MOD = 1e9 + 7;
// DEFINE INT LONG LONG PLMDS
int fat[MAXN_FATORIAL], fati[MAXN_FATORIAL];

// (a^b) % m em O(log b)
// coloque o fexp

int inv(int n) { return fexp(n, MOD - 2); }

void precalc() {
    fat[0] = 1;
    fati[0] = 1;
    for (int i = 1; i < MAXN_FATORIAL; i++) fat[i] =
(fat[i - 1] * i) % MOD;
    fati[MAXN_FATORIAL - 1] = inv(fat[MAXN_FATORIAL -
        1]);
    for (int i = MAXN_FATORIAL - 2; i >= 0; i--) fati
[i] = (fati[i + 1] * (i + 1)) % MOD;
}

int choose(int n, int k) {
    if (k < 0 || k > n) return 0;
    return (((fat[n] * fati[k]) % MOD) * fati[n - k])
        % MOD;
}
```

```cpp
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

```cpp
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

```cpp
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              }
```

```cpp
26          }
27      }
28      return res;
29  }
```

5.4   Segment Sieve

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

5.5   Totient

```cpp
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 ÃŠ 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

```cpp
1   const int MAXN = 1000001;
2   int spf[MAXN];
3   vector<int> primos;
4
5   void crivo() {
6       for (int i = 0; i < MAXN; i++) spf[i] = [i];
7       for (int i = 2; i * i < MAXN; i++) {
8           if (spf[i] == i) {
9               for (int j = i * i; j < MAXN; j += i) {
10                  if (spf[j] == j) {
11                      spf[j] = i;
12                  }
```

```
13              }
14          }
15      }
16      for (int i = 2; i < MAXN; i++) {
17          if (spf[i] == i) {
18              primos.push_back(i);
19          }
20      }
21  }
22
23  map<int, int> fatora(int n) {
24      map<int, int> fatores;
25      while (n > 1) {
26          fatores[spf[n]]++;
27          n /= spf[n];
28      }
29      return fatores;
30  }
31
32  int numero_de_divisores(int n) {
33      if (n == 1) return 1;
34      map<int, int> fatores = fatorar(n);
35      int nod = 1;
36      for (auto &[primo, expoente] : fatores) nod *= (
        expoente + 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     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   * versao que assume: #define int long long
3   *
4   * Retorna um caminho/ciclo euleriano em um grafo (se
         existir).
5   * - g: lista de adjacencia (vector<vector<int>>).
6   * - directed: true se o grafo for dirigido.
7   * - s: vertice inicial.
8   * - e: vertice final (opcional). Se informado, tenta
          caminho de s ate e.
9   * - O(Nlog(N))
10  * Retorna vetor com a sequencia de vertices, ou
         vazio se impossivel.
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     // copia das adjacencias em multiset para
        permitir remoção especifica
15     vector<multiset<int>> h(n);
16     vector<int> in_degree(n, 0);
17     vector<int> result;
18     stack<int> st;
19     // preencher h e indegrees
20     for (int u = 0; u < n; ++u) {
21         for (auto v : g[u]) {
22             ++in_degree[v];
23             h[u].emplace(v);
24         }
25     }
26     st.emplace(s);
27     if (e != -1) {
28         int out_s = (int)h[s].size();
29         int out_e = (int)h[e].size();
30         int diff_s = in_degree[s] - out_s;
31         int diff_e = in_degree[e] - out_e;
32         if (diff_s * diff_e != -1) return {}; //
        impossivel
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   Dinitz

```cpp
1  // Complexidade: O(V^2E)
2
3  struct FlowEdge {
4      int from, to;
5      long long cap, flow = 0;
6      FlowEdge(int from, int to, long long cap) : from(
   from), to(to), cap(cap) {}
7  };
8
9  struct Dinic {
10     const long long flow_inf = 1e18;
11     vector<FlowEdge> edges;
12     vector<vector<int>> adj;
13     int n, m = 0;
14     int s, t;
15     vector<int> level, ptr;
16     queue<int> q;
17
18     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
19         adj.resize(n);
20         level.resize(n);
21         ptr.resize(n);
22     }
23
24     void add_edge(int from, int to, long long cap) {
25         edges.emplace_back(from, to, cap);
26         edges.emplace_back(to, from, 0);
27         adj[from].push_back(m);
28         adj[to].push_back(m + 1);
29         m += 2;
30     }
31
32     bool bfs() {
33         while (!q.empty()) {
34             int from = q.front();
35             q.pop();
36             for (int id : adj[from]) {
37                 if (edges[id].cap == edges[id].flow)
38                     continue;
39                 if (level[edges[id].to] != -1)
40                     continue;
41                 level[edges[id].to] = level[from] +
    1;
42                 q.push(edges[id].to);
43             }
44         }
```

```cpp
45         return level[t] != -1;
46     }
47
48     long long dfs(int from, long long pushed) {
49         if (pushed == 0)
50             return 0;
51         if (from == t)
52             return pushed;
53         for (int& cid = ptr[from]; cid < (int)adj[
    from].size(); cid++) {
54             int id = adj[from][cid];
55             int to = edges[id].to;
56             if (level[from] + 1 != level[to])
57                 continue;
58             long long tr = dfs(to, min(pushed, edges[
    id].cap - edges[id].flow));
59             if (tr == 0)
60                 continue;
61             edges[id].flow += tr;
62             edges[id ^ 1].flow -= tr;
63             return tr;
64         }
65         return 0;
66     }
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                 f += pushed;
79             }
80         }
81         return f;
82     }
83  };
```

6.5   Khan

```cpp
1  // topo-sort DAG
2  // lexicograficamente menor.
3  // N: numero de vÃ©rtices (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;
39  }
```

6.6   Topological Sort

```
1  vector<int> adj[MAXN];
2  vector<int> estado(MAXN); // 0: nao visitado 1:
       processamento 2: processado
3  vector<int> ordem;
4  bool temCiclo = false;
5
6  void dfs(int v) {
7      if(estado[v] == 1) {
8          temCiclo = true;
9          return;
10     }
11     if(estado[v] == 2) return;
12     estado[v] = 1;
13     for(auto &nei : adj[v]) {
14         if(estado[v] != 2) dfs(nei);
15     }
16     estado[v] = 2;
17     ordem.push_back(v);
18     return;
19 }
```

6.7   Acha Pontes

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

```
33     }
34 }
```

6.8   Edmonds-karp

```
1  // Edmonds-Karp com scalling O(EÂšlog(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 }
```

6.9   Kruskal

```
1  // Ordena as arestas por peso, insere se ja nao
       estiver no mesmo componente
2  // O(E log E)
3
```

```cpp
struct Edge {
    int u, v, w;
    bool operator <(Edge const & other) {
        return weight <other.weight;
    }
}

vector<Edge> kruskal(int n, vector<Edge> edges) {
    vector<Edge> mst;
    DSU dsu = DSU(n + 1);
    sort(edges.begin(), edges.end());
    for (Edge e : edges) {
        if (dsu.find(e.u) != dsu.find(e.v)) {
            mst.push_back(e);
            dsu.join(e.u, e.v);
        }
    }
    return mst;
}
```

6.10   Bellman Ford

```cpp
struct Edge {
    int u, v, w;
};

// se x = -1, nao tem ciclo
// se x != -1, pegar pais de x pra formar o ciclo

int n, m;
vector<Edge> edges;
vector<int> dist(n);
vector<int> pai(n, -1);

    for (int i = 0; i < n; i++) {
        x = -1;
        for (Edge &e : edges) {
            if (dist[e.u] + e.w < dist[e.v]) {
                dist[e.v] = max(-INF, dist[e.u] + e.w
);
                pai[e.v] = e.u;
                x = e.v;
            }
        }
    }

// achando caminho (se precisar)
for (int i = 0; i < n; i++) x = pai[x];

vector<int> ciclo;
for (int v = x;; v = pai[v]) {
    cycle.push_back(v);
    if (v == x && ciclo.size() > 1) break;
}
reverse(ciclo.begin(), ciclo.end());
```

6.11   Lca Jc

```cpp
const int MAXN = 200005;
int N;
int LOG;

vector<vector<int>> adj;
vector<int> profundidade;
vector<vector<int>> cima; // cima[v][j] eh o 2^j-
    esimo ancestral de v

void dfs(int v, int p, int d) {
    profundidade[v] = d;
    cima[v][0] = p; // o pai direto eh o 2^0-Ãľsimo
    ancestral
    for (int j = 1; j < LOG; j++) {
```

```cpp
        // 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; // nao 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' ate 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', entÃčo 'a'
    agora Ãľ igual a 'b'
    if (a == b) {
        return a;
    }
    // sobe os dois juntos ate 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];
}
```

6.12   Lca

```cpp
// LCA - CP algorithm
// preprocessing O(NlogN)
// lca O(logN)
// Uso: criar LCA com a quantidade de vertices (n) e
    lista de adjacencias (adj)
// chamar a funcao preprocess com a raiz da arvore

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) {}

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

```
19            up[v][i] = up[up[v][i-1]][i-1];
20
21        for (int u : adj[v]) {
22            if (u != p)
23                dfs(u, v);
24        }
25
26        tout[v] = ++timer;
27    }
28
29    bool is_ancestor(int u, int v) {
30        return tin[u] <= tin[v] && tout[u] >= tout[v
    ];
31    }
32
33    int lca(int u, int v) {
34        if (is_ancestor(u, v))
35            return u;
36        if (is_ancestor(v, u))
37            return v;
38        for (int i = l; i >= 0; --i) {
39            if (!is_ancestor(up[u][i], v))
40                u = up[u][i];
41        }
42        return up[u][0];
43    }
44
45    void preprocess(int root) {
46        tin.resize(n);
47        tout.resize(n);
48        timer = 0;
49        l = ceil(log2(n));
50        up.assign(n, vector<int>(l + 1));
51        dfs(root, root);
52    }
53 };
```

6.13   Kosaraju

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

```
34        }
35    }
36    fill(component + 1, component + N + 1, -1);
37    int c = 0;
38    reverse(order.begin(), order.end());
39    for (int u : order) {
40        if (component[u] == -1) {
41            dfs2(u, c++);
42        }
43    }
44    return c;
45 }
```

6.14   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
     ciclo; false -> retorna vertices 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 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
     , ..., u]
49            reverse(path.begin(), path.end());
50            if (!rec_arestas) return path;
51            // converte para ids das arestas: edgein[
     node] eh 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 }
```

6.15   Min Cost Max Flow

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

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

7   Primitives

8   DP

8.1   Lis

```
1  int lis_nlogn(vector<int> &v) {
2      vector<int> lis;
3      lis.push_back(v[0]);
4      for (int i = 1; i < v.size(); i++) {
5          if (v[i] > lis.back()) {
6              // estende a lis
7              lis.push_back(v[i]);
8          } else {
9              // encontra o primeiro elemento em lis
      que >= v[i].
10             auto it = lower_bound(lis.begin(), lis.
      end(), v[i]);
11             *it = v[i];
12         }
13     }
14     return lis.size();
15 }
16
17 // lis na tree do problema da sub
18 const int MAXN_TREE = 100001;
19 vector<int> adj[MAXN_TREE];
20 int values[MAXN_TREE];
21 int ans = 0;
22
23 void dfs(int u, int p, vector<int>& tails) {
24     auto it = lower_bound(tails.begin(), tails.end(),
       values[u]);
25     int prev = -1;
26     bool coloquei = false;
27     if (it == tails.end()) {
```

```
28          tails.push_back(values[u]);
29          coloquei = true;
30      } else {
31          prev = *it;
32          *it = values[u];
33      }
34      ans = max(ans, (int)tails.size());
35      for (int v : adj[u]) {
36          if (v != p) {
37              dfs(v, u, tails);
38          }
39      }
40      if (coloquei) {
41          tails.pop_back();
42      } else {
43          *it = prev;
44      }
45  }
```

8.2   Edit Distance

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

8.3   Bitmask

```
1  // dp de intervalos com bitmask
2  int prox(int idx) {
3      return lower_bound(S.begin(), S.end(), array<int,
   4>{S[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  int lcs(int i, int j) {
4      if (i < 0 || j < 0) return 0;
5      if (dp[i][j] != -1) return dp[i][j];
6      if (s1[i] == s2[j]) return dp[i][j] = 1 + lcs(i -
   1, j - 1);
7      else return dp[i][j] = max(lcs(i - 1, j), lcs(i,
   j - 1));
8  }
```

8.5   Digit

```
1  vector<int> digits;
2
3  int dp[20][10][2][2];
4
5  int rec(int i, int last, int flag, int started) {
6      if (i == (int)digits.size()) return 1;
7      if (dp[i][last][flag][started] != -1) return dp[i
   ][last][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  vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));
4  for (int i = 1; i <= N; i++) dp[i][0] = 0;
5  for (int j = 0; j <= W; j++) dp[0][j] = 0;
6  for (int i = 1; i <= N; i++) {
7      for (int j = 0; j <= W; j++) {
8          dp[i][j] = dp[i - 1][j];
9          if (j >= items[i-1].first) {
10             dp[i][j] = max(dp[i][j], dp[i - 1][j -
   items[i-1].first] + items[i-1].second);
11         }
12     }
13 }
14 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) -
    vals.begin();
7  };
8  for (int i = 0; i < n; i++) a[i] = id(a[i]);
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(0ll, a[i]
   - 1)) + 1;
15     seg.update(a[i], dp[i]);
16 }
17 cout << *max_element(dp.begin(), dp.end()) << '\n';
```

8.8   Disjoint Blocks

```
1  // num max de subarrays disjuntos com soma x usando
   apenas
2  // prefixo i (ou seja, considerando prefixo a[1..i]).
3  int disjointSumX(vector<int> &a, int x) {
```

```
4        int n = a.size();
5        map<int, int> best; // best[pref] = melhor dp
         visto para 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           dp[i] = dp[i-1];
12           auto it = best.find(pref - x);
13           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          return;
12      }
13      if (i == N) return;
14      int r = N - i;
15      int preciso = K - comb.size();
16      if (r < preciso) return;
17      comb.push_back(elements[i]);
18      brute_choose(i + 1);
19      comb.pop_back();
20      brute_choose(i + 1);
21  }
```

9.2 Struct

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

9.3 Mex

```
1  struct MEX {
2      map<int, int> f;
3      set<int> falta;
4      int tam;
5      MEX(int n) : tam(n) {
6          for (int i = 0; i <= n; i++) falta.insert(i);
7      }
8      void add(int x) {
9          f[x]++;
10          if (f[x] == 1 && x >= 0 && x <= tam) {
```

```
11              falta.erase(x);
12          }
13      }
14      void rem(int x) {
15          if (f.count(x) && f[x] > 0) {
16              f[x]--;
17              if (f[x] == 0 && x >= 0 && x <= tam) {
18                  falta.insert(x);
19              }
20          }
21      }
22      int get() {
23          if (falta.empty()) return tam + 1;
24          return *falta.begin();
25      }
26  };
```

9.4 Count Permutations

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

9.5 Bitwise

```
1  int check_kth_bit(int x, int k) {
2      return (x >> k) & 1;
3  }
4
5  void print_on_bits(int x) {
6      for (int k = 0; k < 32; k++) {
7          if (check_kth_bit(x, k)) {
8              cout << k << ' ';
9          }
10      }
11      cout << '\n';
12  }
13
14  int count_on_bits(int x) {
```

```
15    int ans = 0;
16    for (int k = 0; k < 32; k++) {
17      if (check_kth_bit(x, k)) {
18        ans++;
19      }
20    }
21    return ans;
22 }
23
24 bool is_even(int x) {
25    return ((x & 1) == 0);
26 }
27
28 int set_kth_bit(int x, int k) {
29    return x | (1 << k);
30 }
31
32 int unset_kth_bit(int x, int k) {
33    return x & (~(1 << k));
34 }
35
36 int toggle_kth_bit(int x, int k) {
37    return x ^ (1 << k);
38 }
39
40 bool check_power_of_2(int x) {
41    return count_on_bits(x) == 1;
42 }
```

10   Geometry

10.1   Convex Hull

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {
9     cod x,y;
10    point(cod x = 0, cod y = 0): x(x), y(y)
11    {}
12
13    double modulo()
14    {
15        return sqrt(x*x + y*y);
16    }
17
18    point operator+(point o)
19    {
20        return point(x+o.x, y+o.y);
21    }
22    point operator-(point o)
23    {
24        return point(x - o.x , y - o.y);
25    }
26    point operator*(cod t)
27    {
28        return point(x*t, y*t);
29    }
30    point operator/(cod t)
31    {
32        return point(x/t, y/t);
33    }
34
35    cod operator*(point o)
36    {
37        return x*o.x + y*o.y;
38    }
39    cod operator^(point o)
40    {
41        return x*o.y - y * o.x;
42    }
43    bool operator<(point o)
44    {
45        if( x != o.x) return x < o.x;
46        return y < o.y;
47    }
48
49 };
50
51 int ccw(point p1, point p2, point p3)
52 {
53    cod cross = (p2-p1) ^ (p3-p1);
54    if(cross == 0) return 0;
55    else if(cross < 0) return -1;
56    else return 1;
57 }
58
59 vector <point> convex_hull(vector<point> p)
60 {
61    sort(p.begin(), p.end());
62    vector<point> L,U;
63
64    //Lower
65    for(auto pp : p)
66    {
67        while(L.size() >= 2 and ccw(L[L.size() - 2],
    L.back(), pp) == -1)
68        {
69            // Ã‰ -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  // Any O(n)
28
29  int inside(vp &p, point pp){
30      // 1 - inside / 0 - boundary / -1 - outside
31      int n = p.size();
32      for(int i=0;i<n;i++){
33          int j = (i+1)%n;
34          if(line({p[i], p[j]}).inside_seg(pp))
35              return 0;
36      }
37      int inter = 0;
38      for(int i=0;i<n;i++){
39          int j = (i+1)%n;
40          if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
```

```
     [i], p[j], 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  }
```