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 - 1]);
        }
        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 m = s.size();
    vector<int> lsp(m, 0);
    for (int i = 1, j = 0; i < m; i++) {
        while (j > 0 && s[j] != s[i]) j = lsp[j - 1];
        if (s[i] == s[j]) j++;
        lsp[i] = j;
    }
    return lsp;
}
```

2 DS

2.1 Seg Lazy Pa

```cpp
/* Notas
  PA eh da forma a0 (a) e razÃčo (d)
  na hora de propagar o lazy
  aplica no no S = (a0 + (a0 + (n - 1) * d)) * n / 2  ( variante da soma total do no)
  pros filhos, o da esquerda eh normal
  lazy_a[esq] += lazy_a[x]
  lazy_d[esq] += lazy_d[x]
  pro da direita tem que mudar o a (que eh o elemento inicial naquele no da direita)
  lazy_a[dir] += a + len_esq * (d)
  lazy_d[dir] += lazy_d[x]
*/

int gauss(int n, int a, int d) {
  // (a0 + an) * n / 2
  if (n <= 0)
    return 0;
  return (a + (a + (n - 1) * d)) * n / 2;
}

struct SegTree {
  int n;
  vector<int> v, lazy_a, lazy_d, tree;
  SegTree(vector<int> &a) : v(a), n(a.size()) {
    tree.resize(4 * n);
    lazy_a.resize(4 * n, 0ll);
    lazy_d.resize(4 * n, 0ll);
    build(1, 0, n - 1);
  }
  void build(int x, int lx, int rx) {
    if (lx == rx) {
      tree[x] = v[lx];
      return;
    }
    int mid = (lx + rx) / 2;
    build(2 * x, lx, mid);
    build(2 * x + 1, mid + 1, rx);
    tree[x] = tree[2 * x] + tree[2 * x + 1];
  }
  int query(int x, int lx, int rx, int l, int r) {
    push(x, lx, rx);
    if (lx >= l && rx <= r)
      return tree[x];
    if (lx > r || rx < l)
      return 0;
    int mid = (lx + rx) / 2;
    return query(2 * x, lx, mid, l, r) + query(2 * x + 1, mid + 1, rx, l, r);
  }
  int query(int l, int r) {
    return query(1, 0, n - 1, l, r);
  }
  void push(int x, int lx, int rx) {
    if (lazy_a[x] && lazy_d[x]) {
      int tam = rx - lx + 1;
      tree[x] += gauss(tam, lazy_a[x], lazy_d[x]);
      if (lx != rx) {
        int mid = (lx + rx) / 2;
        int tam_esq = mid - lx + 1;
        lazy_a[2 * x] += lazy_a[x];
        lazy_d[2 * x] += lazy_d[x];
        lazy_a[2 * x + 1] += (lazy_a[x] + tam_esq * lazy_d[x
  ]);
        lazy_d[2 * x + 1] += lazy_d[x];
      }
      lazy_a[x] = lazy_d[x] = 0;
    }
  }
  void update(int x, int lx, int rx, int l, int r, int a, int
     d) {
    push(x, lx, rx);
    if (lx >= l && rx <= r) {
      lazy_a[x] += a + (lx - l) * d;
      lazy_d[x] += d;
      push(x, lx, rx);
      return;
    }
    if (lx > r || rx < l)
      return;
    int mid = (lx + rx) / 2;
    update(2 * x, lx, mid, l, r, a, d);
    update(2 * x + 1, mid + 1, rx, l, r, a, d);
    tree[x] = tree[2 * x] + tree[2 * x + 1];
  }
  void update(int l, int r, int a, int d) {
    update(1, 0, n - 1, l, r, a, d);
  }
};
```

2.2 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];

        for (int i = n - 1; i > 0; --i)
            tree[i] = combine(tree[i * 2], tree[i * 2 + 1]);
    }
    T range_query(int l, int r) {
        T res_l = neutral_value, res_r = neutral_value;

        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) res_l = combine(res_l, tree[l++]);
            if (r & 1) res_r = combine(tree[--r], res_r);
        }

        return combine(res_l, res_r);
    }
    void update(int pos, T new_val) {
        tree[pos += n] = new_val;
        for (pos >>= 1; pos > 0; pos >>= 1)
            tree[pos] = combine(tree[2 * pos], tree[2 * pos +
  1]);
    }
};
```

2.3 Merge Sort Tree

```cpp
#define all(x) x.begin(), x.end()

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

/* mst.query(l, r, l) retorna quantos caras distintos no
    range
  map<int, int> last;
  for (int i = 0; i < n; i++) {
    if (last.count(a[i])) {
      esq[i] = last[a[i]];
    } else {
      esq[i] = -1;
    }
    last[a[i]] = i;
  }
  MST mst(esq);
  jogar vetor de ultima aparicao na seg
*/
```

2.4 Ordered Set E Map

```cpp

#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template<typename T> using ordered_multiset = tree<T,
    null_type, less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
template <typename T> using o_set = tree<T, null_type, less<T
    >, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using o_map = tree<T, R,
    less<T>, rb_tree_tag, tree_order_statistics_node_update
    >;

int main() {
  int i, j, k, n, m;
  o_set<int>st;
  st.insert(1);
  st.insert(2);
  cout << *st.find_by_order(0) << endl; /// k-esimo elemento
  cout << st.order_of_key(2) << endl; ///numero de elementos
    menores que k
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second << endl; /// k-esimo
    elemento
  cout << mp.order_of_key(2) << endl; /// numero de elementos
     (chave) menores que k
  return 0;
}
```

2.5 Sparse Table

```
// 0-index, O(1)
struct SparseTable {
    vector<vector<int>> st;
    int max_log;
    SparseTable(vector<int>& arr) {
        int n = arr.size();
        max_log = floor(log2(n)) + 1;
        st.resize(n, vector<int>(max_log));
        for (int i = 0; i < n; i++) {
            st[i][0] = arr[i];
        }
        for (int j = 1; j < max_log; j++) {
            for (int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    int query(int L, int R) {
        int tamanho = R - L + 1;
        int k = floor(log2(tamanho));
        return max(st[L][k], st[R - (1 << k) + 1][k]);
    }
};
```

2.6 Psum 2d

```
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];
}
```

2.7 Segtree Lazy

```
/*
Seg com double Lazy de soma e de set
tipo 1 - soma em range
tipo 2 - set em range
*/
struct SegTree {
  int n;
  vector<int> v, tree;
  vector<int> lazy_soma, lazy_set;
  SegTree(vector<int> &a) : v(a), n(a.size()) {
    tree.resize(4 * n);
    lazy_soma.resize(4 * n, 0);
    lazy_set.resize(4 * n, -1);
    build(1, 0, n - 1);
  };
  void build(int x, int lx, int rx) {
    if (lx == rx) {
      tree[x] = v[lx];
      return;
    }
    int mid = (lx + rx) / 2;
    build(2 * x, lx, mid);
    build(2 * x + 1, mid + 1, rx);
    tree[x] = tree[2 * x] + tree[2 * x + 1];
  }
  void seta(int x, int lx, int rx, int val) {
    tree[x] = val * (rx - lx + 1);
    lazy_set[x] = val;
    lazy_soma[x] = 0;
  }
  void soma(int x, int lx, int rx, int val) {
    tree[x] += val * (rx - lx + 1);
    if (lazy_set[x] != -1) {
      lazy_set[x] += val;
    } else {
      lazy_soma[x] += val;
    }
  }
  void push(int x, int lx, int rx) {
    int mid = (lx + rx) / 2;
    if (lazy_set[x] != -1) {
      if (lx != rx) {
        seta(2 * x, lx, mid, lazy_set[x]);
        seta(2 * x + 1, mid + 1, rx, lazy_set[x]);
      }
      lazy_set[x] = -1;
    }
    if (lazy_soma[x] != 0) {
      if (lx != rx) {
        soma(2 * x, lx, mid, lazy_soma[x]);
        soma(2 * x + 1, mid + 1, rx, lazy_soma[x]);
      }
      lazy_soma[x] = 0;
    }
  }
  void update(int x, int lx, int rx, int l, int r, int val, int type) {
    push(x, lx, rx);
    if (lx >= l && rx <= r) {
      if (type == 1) soma(x, lx, rx, val);
      else seta(x, lx, rx, val);
      push(x, lx, rx);
      return;
    }
    if (lx > r || rx < l) return;
    int mid = (lx + rx) / 2;
    update(2 * x, lx, mid, l, r, val, type);
    update(2 * x + 1, mid + 1, rx, l, r, val, type);
    tree[x] = tree[2 * x] + tree[2 * x + 1];
  }
  void update(int l, int r, int val, int type) {
    update(1, 0, n - 1, l, r, val, type);
  }
  int query(int x, int lx, int rx, int l, int r) {
    push(x, lx, rx);
    if (lx >= l && rx <= r) return tree[x];
    if (lx > r || rx < l) return 0;
    int mid = (lx + rx) / 2;
    return query(2 * x, lx, mid, l, r) + query(2 * x + 1, mid + 1, rx, l, r);
  }
  int query(int l, int r) {
    return query(1, 0, n - 1, l, r);
  }
};
```

2.8 Dsu

```
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;
    }
};
```

2.9 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 diferenÃ§a 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;
}

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

// C_n = (1 / (n+1)) * C(2n, n)
int catalan(int n) {
    if (n < 0 || 2 * n >= MAXN_FATORIAL) return 0;
    int c2n_n = choose(2 * n, n);
    return (c2n_n * inv(n + 1)) % MOD;
}
```

5.2 Equacao Diofantina

```
int extended_gcd(int a, int b, int& x, int& y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int gcd = extended_gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return gcd;
}

bool solve(int a, int b, int c, int& x0, int& y0) {
    int x, y;
    int g = extended_gcd(abs(a), abs(b), x, y);
    if (c % g != 0) {
        return false;
    }
    x0 = x * (c / g);
    y0 = y * (c / g);
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

5.3 Multiplicacao Matriz

```
// multiplica matrizes de tamanhos variados, resultando em
    uma matrix N*M
vector<vector<int>> mm(vector<vector<int>> A, vector<vector<
    int>> B) {
    int N = A.size(), M = B[0].size(), K = B.size();
    vector<vector<int>> C(N, vector<int>(M));

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < M; ++j)
            for (int k = 0; k < K; ++k)
                C[i][j] = (C[i][j]+A[i][k] * B[k][j] % mod)%
    mod;

    return C;
}
```

5.4 Discrete Log

```
// returns minimum x for which a^x = b (mod m), a and m are
    coprime.
// if the answer dont need to be greater than some value, the
     vector<int> can be removed
int discrete_log(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, vector<int>> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur].push_back(q);
        cur = (cur * 1ll * a) % m;
    }

    int res = LLONG_MAX;

    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            for (int q: vals[cur]){
                int ans = n * p - q;
                res = min(res, ans);
            }
        }
    }
    return res;
}
```

5.5 Segment Sieve

```
// Retorna quantos primos tem entre [l, r] (inclusivo)
// precisa de um vetor com os primos ate sqrt(r)
int seg_sieve(int l, int r){
    if (l > r) return 0;
    vector<bool> is_prime(r - l + 1, true);
    if (l == 1) is_prime[0] = false;

    for (int p : primos){
        if (p * p > r) break;
        int start = max(p * p, (l + p - 1) / p * p);
        for (int j = start; j <= r; j += p){
            if (j >= l) {
                is_prime[j - l] = false;
            }
        }
    }

    return accumulate(all(is_prime), 0ll);;
}
```

5.6 Totient

```
// phi(n) = n * (1 - 1/p1) * (1 - 1/p2) * ...
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1) // SE n sobrou, ele Ãľ um fator primo
        result -= result / n;
    return result;
}

// crivo phi
const int MAXN_PHI = 1000001;
int phiv[MAXN_PHI];
void phi_sieve() {
    for (int i = 0; i < MAXN_PHI; i++) phiv[i] = i;
    for (int i = 2; i < MAXN_PHI; i++) {
        if (phiv[i] == i) {
            for (int j = i; j < MAXN_PHI; j += i) phiv[j] -=
    phiv[j] / i;
        }
    }
}
```

5.7 Menor Fator Primo

```
const int MAXN = 1000001;
int spf[MAXN];
vector<int> primos;

void crivo() {
    for (int i = 0; i < MAXN; i++) spf[i] = [i];
    for (int i = 2; i * i < MAXN; i++) {
        if (spf[i] == i) {
            for (int j = i * i; j < MAXN; j += i) {
                if (spf[j] == j) {
                    spf[j] = i;
                }
            }
        }
    }
    for (int i = 2; i < MAXN; i++) {
        if (spf[i] == i) {
            primos.push_back(i);
        }
    }
}

map<int, int> fatora(int n) {
    map<int, int> fatores;
    while (n > 1) {
        fatores[spf[n]]++;
        n /= spf[n];
    }
    return fatores;
}
```

```cpp

int numero_de_divisores(int n) {
    if (n == 1) return 1;
    map<int, int> fatores = fatorar(n);
    int nod = 1;
    for (auto &[primo, expoente] : fatores) nod *= (expoente
     + 1);
    return nod;
}

// DEFINE INT LONG LONG
int soma_dos_divisores(int n) {
    if (n == 1) return 1;
    map<int, int> fatores = fatorar(n);
    int sod = 1;
    for (auto &[primo, expoente] : fatores) {
        int termo_soma = 1;
        int potencia_primo = 1;
        for (int i = 0; i < expoente; i++) {
            potencia_primo *= primo;
            termo_soma += potencia_primo;
        }
        sod *= termo_soma;
    }
    return sod;

}
```

5.8 Exgcd

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

5.9 Fexp

```cpp
// a^e mod m
// O(log n)

int fexp(int a, int e, int m) {
    a %= m;
    int ans = 1;
    while (e > 0){
        if (e & 1) ans = ans*a % m;
        a = a*a % m;
        e /= 2;
    }
    return ans%m;
}
```

5.10 Divisores

```cpp
// Retorna um vetor com os divisores de x
// eh preciso ter o crivo implementado
// O(divisores)

vector<int> divs(int x){
    vector<int> ans = {1};
    vector<array<int, 2>> primos; // {primo, expoente}

    while (x > 1) {
        int p = crivo[x], cnt = 0;
        while (x % p == 0) cnt++, x /= p;
        primos.push_back({p, cnt});
    }

    for (int i=0; i<primos.size(); i++){
        int cur = 1, len = ans.size();

        for (int j=0; j<primos[i][1]; j++){
            cur *= primos[i][0];
            for (int k=0; k<len; k++)
                ans.push_back(cur*ans[k]);
        }
    }

    return ans;
}
```

5.11 Crivo

```cpp
// O(n*log(log(n)))
bool composto[MAX]
for(int i = 1; i <= n; i++) {
    if(composto[i]) continue;
    for(int j = 2*i; j <= n; j += i)
        composto[j] = 1;
}
```

5.12 Mod Inverse

```cpp
array<int, 2> extended_gcd(int a, int b) {
    if (b == 0) return {1, 0};
    auto [x, y] = extended_gcd(b, a % b);
    return {y, x - (a / b) * y};
}

int mod_inverse(int a, int m) {
    auto [x, y] = extended_gcd(a, m);
    return (x % m + m) % m;
}
```

5.13 Base Calc

```cpp
int char_to_val(char c) {
    if (c >= '0' && c <= '9') return c - '0';
    else return c - 'A' + 10;
}

char val_to_char(int val) {
    if (val >= 0 && val <= 9) return val + '0';
    else return val - 10 + 'A';
}

int to_base_10(string &num, int bfrom) {
    int result = 0;
    int pot = 1;
    for (int i = num.size() - 1; i >= 0; i--) {
        if (char_to_val(num[i]) >= bfrom) return -1;
        result += char_to_val(num[i]) * pot;
        pot *= bfrom;
    }
    return result;
}

string from_base_10(int n, int bto) {
    if (n == 0) return "0";
    string result = "";
    while (n > 0) {
        result += val_to_char(n % bto);
        n /= bto;
    }
    reverse(result.begin(), result.end());
    return result;
}

string convert_base(string &num, int bfrom, int bto) {
    int n_base_10 = to_base_10(num, bfrom);
    return from_base_10(n_base_10, bto);
}
```

5.14 Fft

```cpp
// multiplica dois polinomios em O(NlogN)

using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &A, bool invert) {
  int N = size(A);

  for (int i = 1, j = 0; i < N; i++) {
    int bit = N >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(A[i], A[j]);
  }

  for (int len = 2; len <= N; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < N; i += len) {
      cd w(1);
      for (int j = 0; j < len/2; j++) {
        cd u = A[i+j], v = A[i+j+len/2] * w;
        A[i+j] = u + v;
        A[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }

  if (invert) {
    for (auto &x : A)
      x /= N;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const& B) {
  vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < size(A) + size(B))
    N <<= 1;
  fa.resize(N);
  fb.resize(N);

  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < N; i++)
    fa[i] *= fb[i];
  fft(fa, true);

  vector<int> result(N);
  for (int i = 0; i < N; i++)
    result[i] = round(fa[i].real());
  return result;
}
```

6 Graph

6.1 Dijkstra

```cpp
// SSP com pesos positivos.
// O((V + E) log V).

vector<int> dijkstra(int S) {
    vector<bool> vis(MAXN, 0);
    vector<ll> dist(MAXN, LLONG_MAX);
    dist[S] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, S});
    while(pq.size()) {
        ll v = pq.top().second;
        pq.pop();
        if(vis[v]) continue;
        vis[v] = 1;
        for(auto &[peso, vizinho] : adj[v]) {
            if(dist[vizinho] > dist[v] + peso) {
                dist[vizinho] = dist[v] + peso;
                pq.push({dist[vizinho], vizinho});
            }
        }
    }
    return dist;
}
```

6.2 Floyd Warshall

```cpp
// SSP e acha ciclos.
// Bom com constraints menores.
// O(n^3)

int dist[501][501];

void floydWarshall() {
    for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
void solve() {
    int m, q;
    cin >> n >> m >> q;
    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            if(i == j) {
                dist[i][j] = dist[j][i] = 0;
            } else {
                dist[i][j] = dist[j][i] = linf;
            }
        }
    }
    for(int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w; u--; v--;
        dist[u][v] = min(dist[u][v], w);
        dist[v][u] = min(dist[v][u], w);
    }
    floydWarshall();
    while(q--) {
        int u, v;
        cin >> u >> v; u--; v--;
        if(dist[u][v] == linf) cout << -1 << '\n';
        else cout << dist[u][v] << '\n';
    }
}
```

6.3 Eulerian Path

```cpp
/**
 * versao que assume: #define int long long
 *
 * Retorna um caminho/ciclo euleriano em um grafo (se existir).
 * - g: lista de adjacencia (vector<vector<int>>).
 * - directed: true se o grafo for dirigido.
 * - s: vertice inicial.
 * - e: vertice final (opcional). Se informado, tenta caminho de s ate e.
 * - O(Nlog(N))
 * Retorna vetor com a sequencia de vertices, ou vazio se impossivel.
 */
vector<int> eulerian_path(const vector<vector<int>>& g, bool directed, int s, int e = -1) {
    int n = (int)g.size();
    // copia das adjacencias em multiset para permitir remoÃğÃčo especifica
    vector<multiset<int>> h(n);
    vector<int> in_degree(n, 0);
    vector<int> result;
    stack<int> st;
    // preencher h e indegrees
    for (int u = 0; u < n; ++u) {
        for (auto v : g[u]) {
            ++in_degree[v];
            h[u].emplace(v);
        }
    }
    st.emplace(s);
    if (e != -1) {
        int out_s = (int)h[s].size();
        int out_e = (int)h[e].size();
        int diff_s = in_degree[s] - out_s;
        int diff_e = in_degree[e] - out_e;
        if (diff_s * diff_e != -1) return {}; // impossivel
    }
    for (int u = 0; u < n; ++u) {
        if (e != -1 && (u == s || u == e)) continue;
        int out_u = (int)h[u].size();
        if (in_degree[u] != out_u || (!directed && (in_degree[u] & 1))) {
            return {};
        }
    }
    while (!st.empty()) {
        int u = st.top();
        if (h[u].empty()) {
            result.emplace_back(u);
            st.pop();
        } else {
            int v = *h[u].begin();
            auto it = h[u].find(v);
            if (it != h[u].end()) h[u].erase(it);
            --in_degree[v];
            if (!directed) {
                auto it2 = h[v].find(u);
                if (it2 != h[v].end()) h[v].erase(it2);
                --in_degree[u];
            }
            st.emplace(v);
        }
    }
    for (int u = 0; u < n; ++u) {
        if (in_degree[u] != 0) return {};
    }
    reverse(result.begin(), result.end());
    return result;
}
```

6.4 Dinitz

```cpp
// Complexidade: O(V^2E)

struct FlowEdge {
    int from, to;
    long long cap, flow = 0;
    FlowEdge(int from, int to, long long cap) : from(from), to(to), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int from, int to, long long cap) {
        edges.emplace_back(from, to, cap);
        edges.emplace_back(to, from, 0);
        adj[from].push_back(m);
        adj[to].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int from = q.front();
            q.pop();
            for (int id : adj[from]) {
                if (edges[id].cap == edges[id].flow)
                    continue;
                if (level[edges[id].to] != -1)
                    continue;
                level[edges[id].to] = level[from] + 1;
                q.push(edges[id].to);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int from, long long pushed) {
        if (pushed == 0)
            return 0;
        if (from == t)
            return pushed;
        for (int& cid = ptr[from]; cid < (int)adj[from].size(); cid++) {
            int id = adj[from][cid];
            int to = edges[id].to;
            if (level[from] + 1 != level[to])
                continue;
            long long tr = dfs(to, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

6.5 Khan

```cpp
// topo-sort DAG
// lexicograficamente menor.
// N: numero de vÃľrtices (1-indexado)
// adj: lista de adjacencia do grafo

const int MAXN = 5 * 1e5 + 2;
vector<int> adj[MAXN];
int N;

vector<int> kahn() {
    vector<int> indegree(N + 1, 0);
    for (int u = 1; u <= N; u++) {
        for (int v : adj[u]) {
            indegree[v]++;
        }
    }
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 1; i <= N; i++) {
        if (indegree[i] == 0) {
            pq.push(i);
        }
    }
    vector<int> result;
    while (!pq.empty()) {
        int u = pq.top();
        pq.pop();
        result.push_back(u);
        for (int v : adj[u]) {
            indegree[v]--;
            if (indegree[v] == 0) {
                pq.push(v);
            }
        }
    }
    if (result.size() != N) {
        return {};
    }
    return result;
}
```

6.6 Topological Sort

```cpp
vector<int> adj[MAXN];
vector<int> estado(MAXN); // 0: nao visitado 1: processamento 2: processado
vector<int> ordem;
bool temCiclo = false;

void dfs(int v) {
    if(estado[v] == 1) {
        temCiclo = true;
        return;
    }
    if(estado[v] == 2) return;
    estado[v] = 1;
    for(auto &nei : adj[v]) {
        if(estado[v] != 2) dfs(nei);
    }
    estado[v] = 2;
    ordem.push_back(v);
    return;
}
```

6.7 Acha Pontes

```cpp
vector<int> d, low, pai;       // d[v] Tempo de descoberta (discovery time)
vector<bool> vis;
vector<int> pontos_articulacao;
vector<pair<int, int>> pontes;
int tempo;

vector<vector<int>> adj;

void dfs(int u) {
    vis[u] = true;
    tempo++;
    d[u] = low[u] = tempo;
    int filhos_dfs = 0;
    for (int v : adj[u]) {
        if (v == pai[u]) continue;
        if (vis[v]) { // back edge
            low[u] = min(low[u], d[v]);
        } else {
            pai[v] = u;
            filhos_dfs++;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (pai[u] == -1 && filhos_dfs > 1) {
                pontos_articulacao.push_back(u);
            }
            if (pai[u] != -1 && low[v] >= d[u]) {
                pontos_articulacao.push_back(u);
            }
            if (low[v] > d[u]) {
                pontes.push_back({min(u, v), max(u, v)});
            }
        }
    }
}
```

6.8 Edmonds-karp

```cpp
// Edmonds-Karp com scalling O(EÂšlog(F))

int n, m;
const int MAXN = 510;
vector<vector<int>> capacity(MAXN, vector<int>(MAXN, 0));
vector<vector<int>> adj(MAXN);

int bfs(int s, int t, int scale, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, LLONG_MAX});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next] >= scale) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(MAXN);
    int new_flow;
    int scalling = 1ll << 62;

    while (scalling > 0) {
        while (new_flow = bfs(s, t, scalling, parent)){
            if (new_flow == 0) continue;
            flow += new_flow;
            int cur = t;
            while (cur != s) {
                int prev = parent[cur];
                capacity[prev][cur] -= new_flow;
                capacity[cur][prev] += new_flow;
                cur = prev;
            }
        }
        scalling /= 2;
    }

    return flow;
}
```

6.9 Kruskal

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

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++) {
        // 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

```
// 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)
            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);
    }
};
```

6.13 Kosaraju

```
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 (pos-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();
    fill(vis + 1, vis + N + 1, false);
    for (int i = 1; i <= N; i++) {
        if (!vis[i]) {
            dfs1(i);
        }
    }
    fill(component + 1, component + N + 1, -1);
    int c = 0;
    reverse(order.begin(), order.end());
    for (int u : order) {
        if (component[u] == -1) {
            dfs2(u, c++);
        }
    }
    return c;
}
```

6.14 Pega Ciclo

```
// encontra um ciclo em g
// g[u] = vector<pair<id_aresta, vizinho>>
// rec_arestas: true -> retorna ids das arestas do ciclo;
     false -> retorna vertices do ciclo
// directed: grafo direcionado?

const int MAXN = 5 * 1e5 + 2;
vector<pair<int, int>> g[MAXN];
int N;
bool DIRECTED = false;
```

```cpp
vector<int> color(MAXN), parent(MAXN, -1), edgein(MAXN, -1);
    // color: 0,1,2 ; edgein[v] = id da aresta que entra em v
int ini_ciclo = -1, fim_ciclo = -1, back_edge_id = -1;


bool dfs(int u, int pai_edge){
    color[u] = 1; // cinza
    for (auto [id, v] : g[u]) {
        if (!DIRECTED && id == pai_edge) continue; // ignorar aresta de volta ao pai em n-dir
        if (color[v] == 0) {
            parent[v] = u;
            edgein[v] = id;
            if (dfs(v, id)) return true;
        } else if (color[v] == 1) {
            // back-edge u -> v detectado
            ini_ciclo = u;
            fim_ciclo = v;
            back_edge_id = id;
            return true;
        }
        // se color[v] == 2, ignora
    }
    color[u] = 2; // preto
    return false;
}

// retorna ids das arestas do ciclo
vector<int> pega_ciclo(bool rec_arestas) {
    for (int u = 1; u <= N; u++) {
        if (color[u] != 0) continue;
        if (dfs(u, -1)) {
            // caminho u -> ... -> v via parent
            vector<int> path;
            int cur = ini_ciclo;
            path.push_back(cur);
            while (cur != fim_ciclo) {
                cur = parent[cur];
                path.push_back(cur);
            }
            // path = [u, ..., v] -> inverter para [v, ..., u]
            reverse(path.begin(), path.end());
            if (!rec_arestas) return path;
            // converte para ids das arestas: edgein[node] eh a aresta que entra em node
            vector<int> edges;
            for (int i = 1; i < path.size(); i++) edges.push_back(edgein[path[i]]);
            // adiciona a aresta de retorno u -> v
            edges.push_back(back_edge_id);
            return edges;
        }
    }
    return {};
}
```

6.15 Min Cost Max Flow

```cpp
// Encontra o menor custo para passar K de fluxo em um grafo com N vertices
// Funciona com multiplas arestas para o mesmo par de vertices
// Para encontrar o min cost max flow eh so fazer K = infinito

struct Edge {
    int from, to, capacity, cost, id;
};

vector<vector<array<int, 2>>> adj;
vector<Edge> edges; // arestas pares sao as normais e suas reversas sao as impares

const int INF = LLONG_MAX;

void shortest_paths(int n, int v0, vector<int>& dist, vector<int>& edge_to) {
    dist.assign(n, INF);
    dist[v0] = 0;
    vector<bool> in_queue(n, false);
    queue<int> q;
    q.push(v0);
    edge_to.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        in_queue[u] = false;
        for (auto [v, id] : adj[u]) {
            if (edges[id].capacity > 0 && dist[v] > dist[u] + edges[id].cost) {
                dist[v] = dist[u] + edges[id].cost;
                edge_to[v] = id;
                if (!in_queue[v]) {
                    in_queue[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

void add_edge(int from, int to, int capacity, int cost){
    edges.push_back({from, to, capacity, cost, (int)edges.size()});
    edges.push_back({to, from, 0, -cost, (int)edges.size()}); // reversa
}

int min_cost_flow(int N, int K, int s, int t) {
    adj.assign(N, vector<array<int, 2>>());

    for (Edge e : edges) {
        adj[e.from].push_back({e.to, e.id});
    }

    int flow = 0;
    int cost = 0;
    vector<int> dist, edge_to;
    while (flow < K) {
        shortest_paths(N, s, dist, edge_to);
        if (dist[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, edges[edge_to[cur]].capacity);
            cur = edges[edge_to[cur]].from;
        }

        // apply flow
        flow += f;
        cost += f * dist[t];
        cur = t;
        while (cur != s) {
            int edge = edge_to[cur];
            int rev_edge = edge^1;

            edges[edge].capacity -= f;
            edges[rev_edge].capacity += f;
            cur = edges[edge].from;
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

7 Primitives

8 DP

8.1 Lis

```cpp
int lis_nlogn(vector<int> &v) {
    vector<int> lis;
    lis.push_back(v[0]);
    for (int i = 1; i < v.size(); i++) {
        if (v[i] > lis.back()) {
            // estende a lis
            lis.push_back(v[i]);
        } else {
            // encontra o primeiro elemento em lis que >= v[i
    ].
            auto it = lower_bound(lis.begin(), lis.end(), v[i
    ]);
            *it = v[i];
        }
    }
    return lis.size();
}

// lis na tree do problema da sub
const int MAXN_TREE = 100001;
vector<int> adj[MAXN_TREE];
int values[MAXN_TREE];
int ans = 0;

void dfs(int u, int p, vector<int>& tails) {
    auto it = lower_bound(tails.begin(), tails.end(), values[
    u]);
    int prev = -1;
    bool coloquei = false;
    if (it == tails.end()) {
        tails.push_back(values[u]);
        coloquei = true;
    } else {
        prev = *it;
        *it = values[u];
    }
    ans = max(ans, (int)tails.size());
    for (int v : adj[u]) {
        if (v != p) {
            dfs(v, u, tails);
        }
    }
    if (coloquei) {
        tails.pop_back();
    } else {
        *it = prev;
    }
}
```

8.2 Edit Distance

```
    vector<vector<int>> dp(n+1, vector<int>(m+1, LLONG_MIN));
    for(int j = 0; j <= m; j++) dp[0][j] = j;
    for(int i = 0; i <= n; i++) dp[i][0] = i;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if(a[i-1] == b[j-1]) {
                dp[i][j] = dp[i-1][j-1];
            } else {
                dp[i][j] = min({dp[i-1][j] + 1, dp[i][j-1] +
    1, dp[i-1][j-1] + 1});
            }
        }
    }
    cout << dp[n][m];
```

8.3 Bitmask

```
// dp de intervalos com bitmask
int prox(int idx) {
    return lower_bound(S.begin(), S.end(), array<int, 4>{S[
    idx][1], 0ll, 0ll, 0ll}) - S.begin();
}

int dp[1002][(int)(1ll << 10)];

int rec(int i, int vis) {
    if (i == (int)S.size()) {
        if (__builtin_popcountll(vis) == N) return 0;
        return LLONG_MIN;
    }
    if (dp[i][vis] != -1) return dp[i][vis];
    int ans = rec(i + 1, vis);
    ans = max(ans, rec(prox(i), vis | (1ll << S[i][3])) + S[i
    ][2]);
    return dp[i][vis] = ans;
}
```

8.4 Lcs

```
string s1, s2;
int dp[1001][1001];
int lcs(int i, int j) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    if (s1[i] == s2[j]) return dp[i][j] = 1 + lcs(i - 1, j -
    1);
    else return dp[i][j] = max(lcs(i - 1, j), lcs(i, j - 1));
}
```

8.5 Digit

```
vector<int> digits;

int dp[20][10][2][2];

int rec(int i, int last, int flag, int started) {
    if (i == (int)digits.size()) return 1;
    if (dp[i][last][flag][started] != -1) return dp[i][last][
    flag][started];
    int lim;
    if (flag) lim = 9;
    else lim = digits[i];
    int ans = 0;
    for (int d = 0; d <= lim; d++) {
        if (started && d == last) continue;
        int new_flag = flag;
        int new_started = started;
        if (d > 0) new_started = 1;
        if (!flag && d < lim) new_flag = 1;
        ans += rec(i + 1, d, new_flag, new_started);
    }
    return dp[i][last][flag][started] = ans;
}
```

8.6 Knapsack

```
// dp[i][j] => i-esimo item com j-carga sobrando na mochila
// O(N * W)
vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));
for (int i = 1; i <= N; i++) dp[i][0] = 0;
for (int j = 0; j <= W; j++) dp[0][j] = 0;
for (int i = 1; i <= N; i++) {
    for (int j = 0; j <= W; j++) {
        dp[i][j] = dp[i - 1][j];
        if (j >= items[i-1].first) {
            dp[i][j] = max(dp[i][j], dp[i - 1][j - items[i
    -1].first] + items[i-1].second);
        }
    }
}
cout << dp[N][W] << '\n';
```

8.7 Lis Seg

```
// comprime coordenadas pra quando ai <= 1e9
vector<int> vals(a.begin(), a.end());
sort(vals.begin(), vals.end());
vals.erase(unique(vals.begin(), vals.end()), vals.end());
auto id = [&](int x) -> int {
    return lower_bound(vals.begin(), vals.end(), x) - vals.
    begin();
};
for (int i = 0; i < n; i++) a[i] = id(a[i]);
// fim da parte de compr
SegTree seg(n);
// dp[i]: lis que termina em i
vector<int> dp(n, 1); // dp[i] = 1 base case
for (int i = 0; i < n; i++) {
    if (a[i] > 0) dp[i] = seg.query(0, max(0ll, a[i] - 1)) +
    1;
    seg.update(a[i], dp[i]);
}
cout << *max_element(dp.begin(), dp.end()) << '\n';
```

8.8 Disjoint Blocks

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

```cpp
    best[0] = 0;
    int pref = 0;
    vector<int> dp(n + 1, 0); // dp[0] = 0
    for (int i = 1; i <= n; i++) {
        pref += a[i - 1];
        dp[i] = dp[i-1];
        auto it = best.find(pref - x);
        if (it != best.end()) {
            dp[i] = max(dp[i], it->second + 1);
        }
        best[pref] = max(best[pref], dp[i]);
    }
    return dp[n];
}
```

9 General

9.1 Brute Choose

```cpp
vector<int> elements;
int N, K;
vector<int> comb;

void brute_choose(int i) {
    if (comb.size() == K) {
        for (int j = 0; j < comb.size(); j++) {
            cout << comb[j] << ' ';
        }
        cout << '\n';
        return;
    }
    if (i == N) return;
    int r = N - i;
    int preciso = K - comb.size();
    if (r < preciso) return;
    comb.push_back(elements[i]);
    brute_choose(i + 1);
    comb.pop_back();
    brute_choose(i + 1);
}
```

9.2 Struct

```cpp
struct Pessoa{
    // Atributos
    string nome;
    int idade;

    // Comparador
    bool operator<(const Pessoa& other) const{
        if(idade != other.idade) return idade > other.idade;
        else return nome > other.nome;
    }
}
```

9.3 Mex

```cpp
struct MEX {
    map<int, int> f;
    set<int> falta;
    int tam;
    MEX(int n) : tam(n) {
        for (int i = 0; i <= n; i++) falta.insert(i);
    }
    void add(int x) {
        f[x]++;
        if (f[x] == 1 && x >= 0 && x <= tam) {
            falta.erase(x);
        }
    }
    void rem(int x) {
        if (f.count(x) && f[x] > 0) {
            f[x]--;
            if (f[x] == 0 && x >= 0 && x <= tam) {
                falta.insert(x);
            }
        }
    }
    int get() {
        if (falta.empty()) return tam + 1;
        return *falta.begin();
    }
};
```

9.4 Count Permutations

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

    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
    for (int i = 1; i <= n; i++)
        fact[i] = (fact[i - 1] * i) % MOD;
    invfact[n] = fexp(fact[n], MOD - 2, MOD);
    for (int i = n - 1; i >= 0; i--)
        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;

    // For each position in t, try placing a letter smaller
     than t[i] that is in freq
    for (int i = 0; i < n; i++) {
        for (char c = 'a'; c < t[i]; c++) {
            if (freq[c - 'a'] > 0) {
                freq[c - 'a']--;
                int ways = fact[n - i - 1];
                for (int f : freq)
                    ways = (ways * invfact[f]) % MOD;
                ans = (ans + ways) % MOD;
                freq[c - 'a']++;
            }
        }
        if (freq[t[i] - 'a'] == 0) break;
        freq[t[i] - 'a']--;
    }
    return ans;
}
```

9.5 Bitwise

```cpp
int check_kth_bit(int x, int k) {
  return (x >> k) & 1;
}

void print_on_bits(int x) {
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      cout << k << ' ';
    }
  }
  cout << '\n';
}

int count_on_bits(int x) {
  int ans = 0;
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      ans++;
    }
  }
  return ans;
}

bool is_even(int x) {
  return ((x & 1) == 0);
}

int set_kth_bit(int x, int k) {
  return x | (1 << k);
}

int unset_kth_bit(int x, int k) {
  return x & (~(1 << k));
}

int toggle_kth_bit(int x, int k) {
  return x ^ (1 << k);
}

bool check_power_of_2(int x) {
  return count_on_bits(x) == 1;
}
```

10 Geometry

10.1 Convex Hull

```cpp
#include <bits/stdc++.h>

using namespace std;
#define int long long
typedef int cod;

struct point
{
    cod x,y;
    point(cod x = 0, cod y = 0): x(x), y(y)
    {}

    double modulo()
    {
        return sqrt(x*x + y*y);
    }

    point operator+(point o)
    {
        return point(x+o.x, y+o.y);
    }
    point operator-(point o)
    {
        return point(x - o.x , y - o.y);
    }
    point operator*(cod t)
    {
        return point(x*t, y*t);
    }
    point operator/(cod t)
    {
        return point(x/t, y/t);
    }

    cod operator*(point o)
    {
        return x*o.x + y*o.y;
    }
    cod operator^(point o)
    {
        return x*o.y - y * o.x;
    }
    bool operator<(point o)
    {
        if( x != o.x) return x < o.x;
        return y < o.y;
    }

};

int ccw(point p1, point p2, point p3)
{
    cod cross = (p2-p1) ^ (p3-p1);
    if(cross == 0) return 0;
    else if(cross < 0) return -1;
    else return 1;
}

vector <point> convex_hull(vector<point> p)
{
    sort(p.begin(), p.end());
    vector<point> L,U;

    //Lower
    for(auto pp : p)
    {
        while(L.size() >= 2 and ccw(L[L.size() - 2], L.back(), pp) == -1)
        {
            // Ãľ -1 pq eu nÃčo quero excluir os colineares
            L.pop_back();
        }
        L.push_back(pp);
    }

    reverse(p.begin(), p.end());

    //Upper
    for(auto pp : p)
    {
        while(U.size() >= 2 and ccw(U[U.size()-2], U .back(), pp) == -1)
        {
            U.pop_back();
        }
        U.push_back(pp);
    }

    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}

cod area(vector<point> v)
{
    int ans = 0;
    int aux = (int)v.size();
    for(int i = 2; i < aux; i++)
    {
        ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
    }
    ans = abs(ans);
    return ans;
}

int bound(point p1 , point p2)
{
    return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
}
//teorema de pick [pontos = A - (bound+points)/2 + 1]

int32_t main()
{

    int n;
    cin >> n;

    vector<point> v(n);
    for(int i = 0; i < n; i++)
    {
        cin >> v[i].x >> v[i].y;
    }

    vector <point> ch = convex_hull(v);

    cout << ch.size() <<  '\n';
    for(auto p : ch) cout << p.x << " " << p.y << "\n";

    return 0;
}
```

10.2 Inside Polygon

```cpp
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

10.3 Point Location

```

int32_t main(){
    sws;

    int t; cin >> t;

    while(t--){

        int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

        int deltax1 = (x1-x2), deltay1 = (y1-y2);

        int compx = (x1-x3), compy = (y1-y3);

        int ans = (deltax1*compy) - (compx*deltay1);

        if(ans == 0){cout << "TOUCH\n"; continue;}
        if(ans < 0){cout << "RIGHT\n"; continue;}
        if(ans > 0){cout << "LEFT\n"; continue;}
    }
    return 0;
}
```

10.4 Lattice Points

```
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
ll area_triangulo(ll x1, ll y1, ll x2, ll y2, ll x3, ll y3) {
    return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2));
}
ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
    return gcd(abs(x2 - x1), abs(y2 - y1));
}

int32_t main() {
    ll x1, y1, x2, y2, x3, y3;
    cin >> x1 >> y1;
    cin >> x2 >> y2;
    cin >> x3 >> y3;
    ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
    ll tot_borda = pontos_borda(x1, y1, x2, y2) + pontos_borda(x2, y2, x3, y3) + pontos_borda(x3, y3, x1, y1);

    ll ans = (area - tot_borda) / 2 + 1;
    cout << ans << endl;

    return 0;
}
```