HandBook de romerproblem

Math

matrizFibo

```cpp
struct Matrix {
    long long mat[2][2];
    Matrix friend operator *(const Matrix &a, const
        Matrix &b){
        Matrix c;
        for (int i = 0; i < 2; i++) {
          for (int j = 0; j < 2; j++) {
              c.mat[i][j] = 0;
              for (int k = 0; k < 2; k++) {
                  c.mat[i][j] += a.mat[i][k] * b.mat
                      [k][j];
              }
          }
        }
        return c;
    }
};
Matrix matpow(Matrix base, long long n) {
    Matrix ans{ {
      {1, 0},
      {0, 1}
    } };
    while (n) {
        if(n&1)
            ans = ans*base;
        base = base*base;
        n >>= 1;
    }
    return ans;
}
ll fib(int n) {
    Matrix base{ {
      {1, 1},
      {1, 0}
    } };
    return matpow(base, n).mat[0][1];
}
```

Miscelanea

Plantilla

```cpp
#include <bits/stdc++.h>
// #include <bits/extc++.h>
using namespace std;
// using namespace __gnu_pbds;
typedef long long ll;
typedef long double ld;

// typedef tree<pair<ll, ll>, null_type, less<pair<
    ll, ll>>, rb_tree_tag,
    tree_order_statistics_node_update>
    ordered_set_men;
// typedef tree<int, null_type, greater<int>,
    rb_tree_tag, tree_order_statistics_node_update>
    ordered_set_may;

#define CRISTIANO_RONALDO_GANO_35_COPAS \
    ios_base::sync_with_stdio(false);   \
    cin.tie(NULL);                      \
    cout.tie(nullptr);
#define hola cout << "hola" << endl;
#define YES cout << "YES" << endl;
#define NO cout << "NO" << endl;
#define dbg(x) cout << #x << ": " << x << endl;
#define dbg2(x, y) cout << #x << ": " << x << " __ "
     << #y << ": " << y << endl;
#define printvii(v_v)                       \
    for (auto [x_x, y_y] : v_v)             \
    {                                       \
        cout << x_x << " " << y_y << endl; \
    }                                       \
    cout << endl;

#define printst(st)          \
    for (auto num : st)      \
        cout << num << " "; \
    cout << endl;
template <typename T>
void printv(T v)
```

```cpp
{
    for (auto x : v)
        cout << x << " ";
    cout << endl;
}

#define RAYA cout << "----------------------------"
    << endl;
// #define F first
// #define S second
const ll MOD = 1'000'000'007;
const vector<int> F = {0, 1, 0, -1};
const vector<int> C = {1, 0, -1, 0};

int main()
{
    CRISTIANO_RONALDO_GANO_35_COPAS
    //sol
    return 0;
}
```

Datastructures

LowestCommonAncestor

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

using namespace std;

int n, m;//nodos, links
vector<int> h;//altura del nodo
vector<int> primera;//primera aparicion
// set<int> st;
vector<bool> vis;
vector<vector<int>> gf;
vector<int> nodo;//nodo

void dfs(int currt, int alt)
{
    if (!vis[currt])
    {
        primera[currt] = h.size();
    }
    vis[currt] = 1;
    h.push_back(alt);
    nodo.push_back(currt);

    for (int hijo : gf[currt])
    {
        if (!vis[hijo])
        {
            dfs(hijo, alt + 1);
            h.push_back(alt);
            nodo.push_back(currt);
        }
    }
}

int main()
{
    vis.assign(n, 0);
    primera.assign(n, -1);
    gf.assign(n, vector<int>());
    // st.clear();
    h.clear();
    nodo.clear();
    //0i
    dfs(0, 0);

    //hacer un segmet tree sobre h que es la altura,
        y responder cout<<nodo[res] donde res es el
        indice del minimo en el rango, para minimo en
        rango, donde l y r son los dos nodos
    return 0;
}
```

QueueMin

```cpp
struct quemin
{
  stack<pair<int,int>> bo, to;
  void push(int n)
  {
    if(bo.empty())
      bo.push(mp(n, n));
    else
```

```cpp
            bo.push(mp(n, min(bo.top().s, n)));
    }
    void pop()
    {
        if(to.empty())
        {
            while(!bo.empty())
            {
                if(to.empty())
                    to.push(mp(bo.top().f, bo.top().f));
                else
                    to.push(mp(bo.top().f, min(bo.top().f, to.
                        top().s)));
                bo.pop();
            }
        }
        to.pop();
    }
    int mini()
    {
        int mini = MOD;
        if(!bo.empty())
            mini = bo.top().s;
        if(!to.empty())
            mini = min(mini, to.top().s);
        return mini;
    }
};

struct quemin
{
    pair<int,int> bo[100010], to[100010];
    int boto = -1, toto = -1, ax;
    void push(int n)
    {
        ax = boto + 1;
        if(boto == -1)
            bo[ax] = mp(n, n);
        else
            bo[ax] = mp(n, min(bo[boto].s, n));
        boto++;
    }
    void pop()
    {
        if(toto == -1)
        {
            while(boto > -1)
            {
                ax = toto + 1;
                if(toto == -1)
                    to[ax] = mp(bo[boto].f, bo[boto].f);
                else
                    to[ax] = mp(bo[boto].f, min(bo[boto].f, to
                        [toto].s));
                toto++;
                boto--;
            }
        }
        if(toto > -1)
            toto--;
    }
    int mini()
    {
        int mini = MOD;
        if(boto > -1)
            mini = bo[boto].s;
        if(toto > -1)
            mini = min(mini, to[toto].s);
        return mini;
    }
};
```

SegmentTree

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

using namespace std;
typedef long long ll;

struct Data
{
    ll cant = 0;
    Data() { cant = 1e18; }

    Data(ll c) { cant = c; }
};

```

```
struct SegTree
{
private:
    vector<Data> st;

public:
    int sz;
    Data merge(Data a, Data b)
    {
        return min(a.cant, b.cant);
    }
    void init(int n, vector<Data> v)
    {
        while (__builtin_popcount(n) != 1)
        {
            n++;
        }
        st.resize(2 * n, Data());
        sz = n; // solo n, NO 2*n
        for (int i = 0; i < (int)v.size(); i++)
        {
            st[n + i] = v[i];
        }

        for (int i = n - 1; i > 0; --i)
        {
            st[i] = merge(st[i << 1], st[(i << 1) + 1]);
        }
    }

    void updateTreeNode(int p, Data nuevoValor) // 0 i pos
    {
        st[p + sz] = nuevoValor;
        p = p + sz;
        for (int i = p; i > 1; i >>= 1)
            st[i >> 1] = merge(st[i], st[i ^ 1]);
    }

    Data query(int nodo, int left_nodo, int right_nodo, int l_q, int r_q) //
    {
        // query(1,0,st.sz-1,l_q,r_q) tipo->[l_q,r_q]0i
        /*
         los indices de los nodos empieza desde 1;
         */
        if (l_q <= left_nodo && right_nodo <= r_q)
        {
            return st[nodo];
        }
        if (l_q > right_nodo || left_nodo > r_q)
        {
            return Data();
        }

        int mitad = (left_nodo + right_nodo) / 2; ///[l:r] --> [l:mitad] [mitad+1:r]
        return merge(query(nodo * 2, left_nodo, mitad, l_q, r_q), query(nodo * 2 + 1, mitad + 1, right_nodo, l_q, r_q));
    }
};
```

SparseTable

```
struct SparseTable {
    int n;
    vector<int> log2;
    vector<vector<int>> st;

    SparseTable(const vector<int>& a) {
        n = a.size();
        log2.resize(n + 1);
        log2[1] = 0;
        for (int i = 2; i <= n; i++)
            log2[i] = log2[i/2] + 1;

        int k = log2[n] + 1;
        st.assign(n, vector<int>(k));

        for (int i = 0; i < n; i++) st[i][0] = a[i];

        for (int j = 1; j < k; j++)
            for (int i = 0; i + (1 << j) <= n; i++)
                st[i][j] = min(st[i][j-1], st[i + (1
                         << (j-1))][j-1]);
    }

    // Consulta de minimo en el rango [l, r]oi
    int query(int l, int r) {
        int j = log2[r - l + 1];
        return min(st[l][j], st[r - (1 << j) + 1][j
            ]);
    }
};
```

UnionFind

```
struct unionFind {
  vi p;
  unionFind(int n) : p(n, -1) {}
  int findParent(int v) {
    if (p[v] == -1) return v;
    return p[v] = findParent(p[v]);
  }
  bool join(int a, int b) {
    a = findParent(a);
    b = findParent(b);
    if (a == b) return false;
    p[a] = b;
    return true;
  }
};
```

fenwickTree

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

using namespace std;
typedef long long ll;


struct Bit
{
    private:
        vector<ll> bit;
        int sz;
    public:

    Bit(int n){bit.resize(n+1,0);this->sz=n;}

    void init(vector<ll> vec)
    {
        for(int i=0;i<sz;i++)
        {
            update(i+1,vec[i]);
        }
    }

    void update(int p,ll val)//1i
    {
        while(p<=sz)
        {
            bit[p]+=val;
            p+=(p&(-p));
        }
    }
    ll query(int p)//[1,p]1i
    {
        ll ans=0;
        while(p>0)
        {
            ans+=bit[p];
            p-=(p&-p);
        }
        return ans;
    }

    ll rquery(ll left,ll right)//[l,r]1i
    {
        return query(right)-query(left-1);
    }
};
```

Strings

hashing

```
struct StrHash
{ // Hash polinomial con exponentes decrecientes.
```

```
    static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
    static constexpr ll b = 500'000'000;
    vector<ll> hs[2], bs[2];
    StrHash(string const &s)
    {
        int n = s.length();
        for(int k=0;k<2;k++)
        {
            hs[k].resize(n + 1), bs[k].resize(n + 1, 1);
            for(int i=0;i<n;i++)
            {
                hs[k][i + 1] = (hs[k][i] * b + s[i]) % ms[k];
                bs[k][i + 1] = bs[k][i] * b % ms[k];
            }
        }
    }
    ll get(int idx, int len) const
    { // Hashes en `s[idx, idx+len)`.
        ll h[2];
        for(int k=0;k<2;k++)
        {
            h[k] = hs[k][idx + len] - hs[k][idx] * bs[k][len] % ms[k];
            if (h[k] < 0)
                h[k] += ms[k];
        }
        return (h[0] << 32) | h[1];
    }
};

//concate substrings(or strings) from non necesary two differents strings [idx,indx+lex)
ll concat_cross_hashes(const StrHash& A, int i1, int len1, const StrHash& B, int i2, int len2)
{
    ll res[2];
    for (int k = 0; k < 2; ++k)
    {
        // hash de substring A[i1..i1+len1-1]
        ll h1 = A.hs[k][i1 + len1] - A.hs[k][i1] * A.bs[k][len1] % A.ms[k];
        if (h1 < 0) h1 += A.ms[k];

        // hash de substring B[i2..i2+len2-1]
        ll h2 = B.hs[k][i2 + len2] - B.hs[k][i2] * B.bs[k][len2] % B.ms[k];
        if (h2 < 0) h2 += B.ms[k];

        // combinacion: h1 * b^len2 + h2
        res[k] = (h1 * A.bs[k][len2] + h2) % A.ms[k];
    }
    return (res[0] << 32) | res[1];
}


//0 indexed
ll h=StrHash("Hola").get(0,0+"Hola".size());
```

hashing2d

```
struct Hashing
{
    vector<vector<int>> hs;
    vector<int> PWX, PWY;
    int n, m;
    static const int PX = 3731, PY = 2999, mod = 998244353;
    Hashing() {}
    Hashing(vector<string> &s)
    {
        n = (int)s.size(), m = (int)s[0].size();
        hs.assign(n + 1, vector<int>(m + 1, 0));
        PWX.assign(n + 1, 1);
        PWY.assign(m + 1, 1);
        for (int i = 0; i < n; i++)
            PWX[i + 1] = 1LL * PWX[i] * PX % mod;
        for (int i = 0; i < m; i++)
            PWY[i + 1] = 1LL * PWY[i] * PY % mod;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                hs[i + 1][j + 1] = s[i][j] - 'a' +
                    1;
            }
        }
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                hs[i][j + 1] = (hs[i][j + 1] + 1LL *
                    hs[i][j] * PY % mod) % mod;
            }
        }
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j <= m; j++)
            {
                hs[i + 1][j] = (hs[i + 1][j] + 1LL *
                    hs[i][j] * PX % mod) % mod;
            }
        }
    }
    int get_hash(int x1, int y1, int x2, int y2)
    { // 1-indexed
        assert(1 <= x1 && x1 <= x2 && x2 <= n);
        assert(1 <= y1 && y1 <= y2 && y2 <= m);
        x1--;
        y1--;
        int dx = x2 - x1, dy = y2 - y1;
        return (1LL * (hs[x2][y2] - 1LL * hs[x2][y1]
            * PWY[dy] % mod + mod) % mod -
                1LL * (hs[x1][y2] - 1LL * hs[x1][y1]
                    * PWY[dy] % mod + mod) % mod *
                  PWX[dx] % mod + mod) %
              mod;
    }
    int get_hash()
    {
        return get_hash(1, 1, n, m);
    }
};
```

Geometry

Point

```
/*typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()*/

//typedef long long ll;
//typedef long double ll;

struct point
{
  ll x, y;
  point() {}
  point(ll x, ll y): x(x), y(y) {}
  point operator -(point p) {return point(x - p.x, y
      - p.y);}
  point operator +(point p) {return point(x + p.x, y
      + p.y);}
  ll sq() {return x * x + y * y;}
  double abs() {return sqrt(sq());}
  ll operator ^(point p) {return x * p.y - y * p.x;}
    ll operator *(point p) {return x * p.x + y * p.y
      ;}
    point operator *(ll a) {return point(x * a, y *
      a);}
  bool operator <(const point& p) const {return x ==
      p.x ? y < p.y : x < p.x;}
  bool left(point a, point b) {return ((b - a) ^ (*
      this - a)) >= 0;}
  ostream& operator<<(ostream& os) {
    return os << "("<< x << "," << y << ")";
  }

};

void polarSort(vector<point>& v) {
  sort(v.begin(), v.end(), [] (point a, point b) {
    const point origin{0, 0};
    bool ba = a < origin, bb = b < origin;
    if (ba != bb) { return ba < bb; }
    return (a^b) > 0;
  });
}
```

convexHull

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

typedef long long ll;

struct pt
{
  double x, y;
  bool operator==(pt const &t) const
  {
    return x == t.x && y == t.y;
  }
};

int orientation(pt a, pt b, pt c)
{
  double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) +
      c.x * (a.y - b.y);
  if (v < 0)
    return -1; // clockwise
  if (v > 0)
    return +1; // counter-clockwise
  return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear)
{
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return
    orientation(a, b, c) == 0; }

void convex_hull(vector<pt> &a, bool
    include_collinear = false)
{
  pt p0 = *min_element(a.begin(), a.end(), [](pt a,
      pt b)
                   { return make_pair(a.y, a.x) <
                       make_pair(b.y, b.x); });
  sort(a.begin(), a.end(), [&p0](const pt &a, const
      pt &b)
      {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y
                )*(p0.y-a.y)
                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)
                    *(p0.y-b.y);
        return o < 0; });
  if (include_collinear)
  {
    int i = (int)a.size() - 1;
    while (i >= 0 && collinear(p0, a[i], a.back()))
      i--;
    reverse(a.begin() + i + 1, a.end());
  }

  vector<pt> st;
  for (int i = 0; i < (int)a.size(); i++)
  {
    while (st.size() > 1 && !cw(st[st.size() - 2],
        st.back(), a[i], include_collinear))
      st.pop_back();
    st.push_back(a[i]);
  }

  if (include_collinear == false && st.size() == 2
      && st[0] == st[1])
    st.pop_back();

  a = st;
}

vector<pt> a;

int main()
{

  int n;cin>>n;
  a.resize(n);
  //leer como puntos
  convex_hull(a,true);//true incluye colinear, false
      no lo hace

}
```

Primalidad

RabinMiller

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

using namespace std;


using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) { // returns
   true if n is probably prime, else returns false.
    cout<<iter<<endl;
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}


int main()
{

    cout<<MillerRabin(100000001,30)<<endl;
    return 0;
}
```

pollardRho

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <cassert>
#include <map>

using namespace std;

typedef long long ll;

#define forn(i, n) for (int i = 0; i < (int)(n); i
   ++)
#define forsn(i, s, n) for (int i = int(s); i < (int
    )(n); i++)

// rabin miller

ll potlog(ll a, ll b, const ll M)
{
    ll res = 1;
    while (b)
    {
        if (b % 2)
            res = (__int128(res) * a) % M;
        a = (__int128(a) * a) % M;
        b /= 2;
    }
    return res;
}

bool primo(ll n)
{
    if (n < 2)
        return false;
    if (n == 2)
        return true;
    ll D = n - 1, S = 0;
    while (D % 2 == 0)
    {
        D /= 2;
        S++;
    }
    // n-1 = 2^S * D
    static const int STEPS = 16;
    forn(pasos, STEPS)
    {
        const ll A = 1 + rand() % (n - 1);
        ll M = potlog(A, D, n);
        if (M == 1 || M == (n - 1))
            goto next;
        forn(k, S - 1)
        {
            M = (__int128(M) * M) % n;
            if (M == (n - 1))
                goto next;
        }
        return false;
    next:;
    }
    return true;
}

// pollard's rho

ll mcd(ll a, ll b) { return (a == 0) ? b : mcd(b % a
    , a); }

ll factor(ll n)
{
    static ll A, B;
    A = 1 + rand() % (n - 1);
    B = 1 + rand() % (n - 1);
#define f(x) ((__int128(x) * (x + B)) % n + A)
    ll x = 2, y = 2, d = 1;
    while (d == 1 || d == -1)
    {
        x = f(x);
        y = f(f(y));
        d = mcd(x - y, n);
    }
    return abs(d);
}

map<ll, ll> fact;

void factorize(ll n)
{
    assert(n > 0);
    while (n > 1 && !primo(n))
    {
        ll f;
        do
        {
            f = factor(n);
        } while (f == n);
        n /= f;
        factorize(f);
        for (auto &it : fact)
            while (n % it.first == 0)
            {
                n /= it.first;
```

```cpp
                it.second++;
            }
    }
    if (n > 1)
        fact[n]++;
}

int main()
{
    ll N;
    while (cin >> N && N)
    {
        fact.clear();
        factorize(N);
        for (const auto &it : fact)
            cout << it.first << "^" << it.second <<
                " ";
        cout << endl;
    }
    return 0;
}
```

Graphs

isDag

```cpp
vector<vector<int>> gf;  // lista de adyacencia
vector<int> visited;     // 0 = no visitado, 1 =
   visitando, 2 = visitado

bool dfs(int u) {
    visited[u] = 1; // visitando
    for (int v : gf[u]) {
        if (visited[v] == 1) return true;  // ciclo
            detectado
        if (visited[v] == 0 && dfs(v)) return true;
    }
    visited[u] = 2; // visitado
    return false;
}

bool isDAG(int n) {
    visited.assign(n, 0);
    for (int i = 0; i < n; i++)
        if (visited[i] == 0 && dfs(i))
            return false;  // hay ciclo
    return true; // no hay ciclos
}
```

toposort

```cpp
#include <bits/stdc++.h>
using namespace std;
int n; // number of vertices
vector<vector<int>> gf;
vector<bool> vis;
vector<int> ans;

void dfs(int v) {//Oi
    vis[v] = true;
    for (int u : gf[v]) {
        if (!vis[u]) {
            dfs(u);
        }
    }
    ans.push_back(v);
}

void topological_sort() {
    vis.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

Overloads

pq

```cpp
struct cmp
{
    //mayor tiene prioridad
    bool operator()(const int& a, const int& b)
        const {
         return a<b;
    }
};

//priority_queue<Data,vector<Data>,cmp> pq
```

set

```
struct cmpST
{
    //de menor a mayor
    bool operator()(const int &a,const int &b)
    {
        return a<b;
    }
};

//set<int,cmpST> st;
```