BIBLIOTECA Stenio-

UFMG

[1- Matematica 2](#_Toc461220202)

[1.1- Crivo de Eratóstenes 2](#_Toc461220203)

[1.2- Inverso Multiplicativo 2](#_Toc461220204)

[1.3- Exponenciaçao Matriz 2](#_Toc461220205)

[1.4- Funcao Totiente 2](#_Toc461220206)

[1.5- Euclides Estendido 2](#_Toc461220207)

[1.6 Triangulo de Pascal 2](#_Toc461220208)

[1.7 Exp mat vazek 3](#_Toc461220209)

[1.8 Pollard Rho 3](#_Toc461220210)

[1.9 Teorema do Resto Chines 4](#_Toc461220211)

[2- Strings 4](#_Toc461220212)

[2.1- Rabin Karp 4](#_Toc461220213)

[2.2- KMP 4](#_Toc461220214)

[2.3- AHO- CORASICK 5](#_Toc461220215)

[3- Programacao dinamica 5](#_Toc461220216)

[3.2- Numero de Palindromes 5](#_Toc461220217)

[3.3 Divide and Conquer Example 5](#_Toc461220218)

[3.4 Convex Hull Trick 5](#_Toc461220219)

[4- Grafos 6](#_Toc461220220)

[4.1- Fluxo Maximo 6](#_Toc461220221)

[4.1.1- Edmonds Karp 6](#_Toc461220222)

[4.1.2- Dinic ( Dilson) 6](#_Toc461220223)

[4.2.1- Matching Maximo 7](#_Toc461220224)

[4.2- Menor Caminho 7](#_Toc461220225)

[4.2.1- Floyd Warshall 7](#_Toc461220226)

[4.3- Arvore Geradora Minima 7](#_Toc461220227)

[4.4- Kosaraju - com compressão 7](#_Toc461220228)

[4.5- Isomorfismo Arvore 8](#_Toc461220229)

[4 5- Achar Pontes/Pontos de Articulação 9](#_Toc461220230)

[4.6- LCA 10](#_Toc461220231)

[4.7- Min cost max flow 10](#_Toc461220232)

[4.8- 2-Sat – com recuperacao 11](#_Toc461220233)

[5- Estruturas de Dados 11](#_Toc461220234)

[5.1- Segtree 11](#_Toc461220235)

[5.1- RMQ- indice 11](#_Toc461220236)

[5.2- RSQ Lazy-Propagation 11](#_Toc461220237)

[5.4- BIT 12](#_Toc461220238)

[5.5- Disjoint Set Com Rank 12](#_Toc461220239)

[5.6 Algoritmo de Mo 12](#_Toc461220240)

[5.7- Seg de vector 13](#_Toc461220241)

[5.8- Sparce Table 13](#_Toc461220242)

[6-Geometria 13](#_Toc461220243)

[6.1- Distancia de dois ponts (ao quadrado) 14](#_Toc461220244)

[6.2- Distancia Ponto a Segmento (ao quadrado) 14](#_Toc461220245)

[6.3- Produto Vetorial 14](#_Toc461220246)

[6.4- Produto Escalar 14](#_Toc461220247)

[6.5- Minimum Enclosing Circle 14](#_Toc461220248)

[7- Problemas a se considerar 15](#_Toc461220249)

[7.1- Invertion Count: 15](#_Toc461220250)

[8- Big Numbers 15](#_Toc461220251)

[8.1- Tirar zero a Esquerda 15](#_Toc461220252)

[8.2- Somar 2 numeros 15](#_Toc461220253)

[8.3- Multiplicar 2 numeros 16](#_Toc461220254)

[9-FFTZINHO 16](#_Toc461220255)

[Template 16](#_Toc461220256)

[Coisas a saber: 17](#_Toc461220257)

[Numeros de Fibonacci 17](#_Toc461220258)

[Primos para o Hash: 17](#_Toc461220259)

[String Stream 17](#_Toc461220260)

[Lista de Primos : 17](#_Toc461220261)

[/\* Josephus Problem - It returns the position to be, in order to not die. O(n)\*/ 18](#_Toc461220262)

[Identidade Binomial ( Binômio de Newton ): 18](#_Toc461220263)

# 1- Matematica

## 1.1- Crivo de Eratóstenes

vector<int> primes;

void gen\_primes()

{

int i, j;

int MAX=1e6;

primes.assign(MAX,1);

for (i = 2; i <= (int)sqrt(MAX); i++)

if (primes[i])

for (j = i; j \* i < MAX; j++) primes[i \* j] = 0;

}

## 1.2- Inverso Multiplicativo

ll mul\_inv(ll a)

{

ll pin0 = MOD,pin=MOD, t, q;

ll x0 = 0, x1 = 1;

if (pin == 1) return 1;

while (a > 1) {

q = a / pin;

t = pin, pin = a % pin, a = t;

t = x0, x0 = x1 - q \* x0, x1 = t;

}

if (x1 < 0) x1 += pin0;

return x1;

}

## 1.3- Exponenciaçao Matriz

vvi matmul(vvi &m1, vvi &m2)

{

vvi ans;

ans.resize(m1.size(), vi(m2.size(), 0));

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

for (int k = 0; k < n; k++) {

ans[i][j] += m1[i][k] \* m2[k][j];

ans[i][j] %= MOD;

}

return ans;

}

vvi matpow(vvi &m1, ll p)

{

vvi ans;

ans.resize(m1.size(), vi(m1.size(), 0));

for (int i = 0; i < n; i++) ans[i][i] = 1;

while (p) {

if (p & 1) ans = matmul(ans, m1);

m1 = matmul(m1, m1);

p >>= 1;

}

return ans;

}

// VETOR TEM N LINHAS E A MATRIZ E QUADRADA

vi mulvet(vvi &m1, vi &vet)

{

vi ans;

ans.resize(vet.size(), 0);

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++) {

ans[i] += (m1[i][j] \* vet[j]);

ans[i] %= MOD;

}

return ans;

}

## 1.4- Funcao Totiente

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) result -= result / n;

return result;

}

## 1.5- Euclides Estendido

struct ext {

ll x;

ll y;

ll mdc;

};

ext tmp;

// ax + by=c, se mdc(a,b) nao divide c, nao tem solucao, caso contrario, x = x0

// +(b/mdc)\*n, y=yo-(a/mdc)\*n

ll ee(ll a, ll b, ll &x, ll &y)

{

if (b == 0) {

x = 1;

y = 0;

return a;

}

ll x1, y1;

ll tmp = ee(b, a % b, x1, y1);

x = y1;

y = x1 - (a / b) \* y1;

return tmp;

}

ext extended\_euclid(ll a, ll b)

{

ll tmp, tmp1;

ext ret;

ret.mdc = ee(a, b, tmp, tmp1);

ret.x = tmp;

ret.y = tmp1;

return ret;

}

## 1.6 Triangulo de Pascal

unsigned long long comb[61][61];

for(int i = 0; i < 61; i++)

{

 comb[i][i] = 1;

 comb[i][0] = 1;

}

for(int i = 2; i < 61; i++)

 for(int j = 1; j < i; j++)

    comb[i][j] = comb[i-1][j] + comb[i-1][j-1];

## 1.7 Exp mat vazek

int N;

ll reta[MAXN][MAXN];

ll aux[MAXN][MAXN];

ll ma[MAXN][MAXN];

ll v[MAXN];

ll resp[MAXN];

void mul(ll(&m1)[MAXN][MAXN],ll(&m2)[MAXN][MAXN], ll(&m3)[MAXN][MAXN])

{

memset(reta,0LL,sizeof(reta));

for(int i=0;i<N;i++)

{

for(int j=0;j<N;j++)

{

for(int k=0;k<N;k++)

{

reta[i][j]+=m1[i][k]\*m2[k][j];

}

reta[i][j]=reta[i][j]%MOD;

}

}

for(int i=0;i<N;i++) for(int j=0;j<N;j++) m3[i][j]=reta[i][j];

}

void exp(ll (&dp)[MAXN][MAXN], ll e)

{

memset(aux,0LL,sizeof(aux));

for (int i = 0; i < N; i++) aux[i][i] = 1;

while (e >= 1)

{

if (e & 1) mul(dp, aux,aux);

mul(dp, dp,dp);

e >>= 1;

}

for(int i=0;i<N;i++) for(int j=0;j<N;j++) dp[i][j]=aux[i][j];

}

void mulvet(ll (&m1)[MAXN][MAXN], ll (&v)[MAXN], ll (&ret)[MAXN])

{

for (int i=0;i<N;i++)

{

for (int j=0;j<N;j++)

{

ret[i]+=m1[i][j]\*v[j];

ret[i]=ret[i]%MOD;

}

}

}

## 1.8 Pollard Rho

ll u;

ll t;

const int tamteste=5;

ll abss(ll v){ return v>=0 ? v : -v;}

ll randerson()

{

ld pseudo=(ld)rand()/(ld)RAND\_MAX;

return (ll)(round((ld)range\*pseudo))+1LL;

}

ll mulmod(ll a, ll b, ll mod)

{

ll ret=0;

while(b>0)

{

if(b%2!=0) ret=(ret+a)%mod;

a=(a+a)%mod;

b=b/2LL;

}

return ret;

}

ll expmod(ll a, ll e, ll mod)

{

ll ret=1;

while(e>0)

{

if(e%2!=0) ret=mulmod(ret,a,mod);

a=mulmod(a,a,mod);

e=e/2LL;

}

return ret;

}

bool jeova(ll a, ll n)

{

ll x = expmod(a,u,n);

ll last=x;

for(int i=0;i<t;i++)

{

x=mulmod(x,x,n);

if(x==1 and last!=1 and last!=(n-1)) return true;

last=x;

}

if(x==1) return false;

return true;

}

bool isprime(ll n)

{

u=n-1;

t=0;

while(u%2==0)

{

t++;

u/=2LL;

}

if(n==2) return true;

if(n==3) return true;

if(n%2==0) return false;

if(n<2) return false;

for(int i=0;i<tamteste;i++)

{

ll v = randerson()%(n-2)+1;

//cout<<"jeova "<<v<<" "<<n<<endl;

if(jeova(v,n)) return false;

}

return true;

}

ll gcd(ll a, ll b){ return !b ? a : gcd(b,a%b);}

ll calc(ll x, ll n, ll c)

{

return (mulmod(x,x,n)+c)%n;

}

ll pollard(ll n)

{

ll d=1;

ll i=1;

ll k=1;

ll x=2;

ll y=x;

ll c;

do

{

c=randerson()%n;

}while(c==0 or (c+2)%n==0);

while(d!=n)

{

if(i==k)

{

k\*=2LL;

y=x;

i=0;

}

x=calc(x,n,c);

i++;

d=gcd(abss(y-x),n);

if(d!=1) return d;

}

}

vector<ll> getdiv(ll n)

{

vector<ll> ret;

if(n==1) return ret;

if(isprime(n))

{

ret.pb(n);

return ret;

}

ll d = pollard(n);

ret=getdiv(d);

vector<ll> ret2=getdiv(n/d);

for(int i=0;i<ret2.size();i++) ret.pb(ret2[i]);

return ret;

}

## 1.9 Teorema do Resto Chines

ll mulmod(ll a, ll b, ll m)

{

ll ret=0;

while(b>0)

{

if(b%2!=0) ret=(ret+a)%m;

a=(a+a)%m;

b>>=1;

}

return ret;

}

ll expmod(ll a, ll e, ll m)

{

ll ret=1;

while(e>0)

{

if(e%2!=0) ret=mulmod(ret,a,m);

a=mulmod(a,a,m);

e>>=1;

}

return ret;

}

ll invmul(ll a, ll m)

{

return expmod(a,m-2,m);

}

ll chinese(vector<ll> r, vector<ll> m)

{

int sz=m.size();

ll M =1;

for(int i=0;i<sz;i++)

{

M\*=m[i];

}

ll ret=0;

for(int i=0;i<sz;i++)

{

ret+=mulmod(mulmod(M/m[i],r[i],M),invmul(M/m[i],M),M);

ret=ret%M;

}

return ret;

}

# 2- Strings

## 2.1- Rabin Karp

const ll M= 1000004099;

const ll B= 31;

ll int\_mod(ll a, ll b) { return (a % b + b) % b; }

ll eleva(ll a, ll b, ll mod){

if(b==0) return 1;

else if(b==1) return a;

ll x= eleva(a,b/2,mod);

if(b%2==0) return (x\*x)%mod;

else return (a\*((x\*x)%mod))%mod;

}

bool Rabin\_karp(string text, string pattern)

{

int n = text.size();

int m = pattern.size();

if (n < m) return false;

ll hp = 0;

for (int i = 0; i < m; i++) hp = int\_mod(hp \* B + pattern[i], M);

ll ht = 0;

for (int i = 0; i < m; i++) ht = int\_mod(ht \* B + text[i], M);

if (ht == hp) return true;

ll E=eleva(B, m-1, M);

for (int i = m; i < n; i++) {

ht=int\_mod(ht - int\_mod(text[i-m]\*E,M), M);

ht= int\_mod(ht\*B, M);

ht= int\_mod(ht + text[i], M);

if(ht==hp) return true;

}

return false;

}

## 2.2- KMP

int b[100000];

int sizet,sizep;

void kmpPreprocess(string &text, string &pattern){

int i=0,j=-1; b[0]=-1;

while(i<sizep){

while(j>=0 and pattern[i] != pattern[j]) j=b[j];

i++,j++;

b[i]=j;

}

}

void kmpSearch(string &text, string &pattern){

kmpPreprocess(text,pattern);

int i=0, j=0;

while(i<sizet){

while(j>=0 and text[i] != pattern[j]) j=b[j];

i++,j++;

if(j==sizep){

cout<<"Olha a substring do texto "<<i-j<<endl;

j=b[j];

}

}

}

## 2.3- AHO- CORASICK

int to[N][M], Link[N], fim[N];

int idx = 1;

void add\_str(string &s)

{

int v = 0;

for (int i = 0; i < s.size(); i++) {

if (!to[v][s[i]]) to[v][s[i]] = idx++;

v = to[v][s[i]];

}

fim[v] = 1;

}

void process()

{

queue<int> fila;

fila.push(0);

while (!fila.empty()) {

int cur = fila.front();

fila.pop();

int l = Link[cur];

fim[cur] |= fim[l];

for (int i = 0; i < 200; i++) {

if (to[cur][i]) {

if (cur != 0) {

Link[to[cur][i]] = to[l][i];

}

else

Link[to[cur][i]] = 0;

fila.push(to[cur][i]);

}

else {

to[cur][i] = to[l][i];

}

}

}

}

int resolve(string &s)

{

int v = 0, r = 0;

;

for (int i = 0; i < s.size(); i++) {

v = to[v][s[i]];

if (fim[v]) r++, v = 0;

}

return r;

}

# 3- Programacao dinamica

**3.1- Lis com recuperação de resposta**

//asw -> vetor com resposta!!

//asw.size() é o tamanho da maior lis

void lis( const vector< int > & v, vector< int > & asw )

{

vector<int> pd(v.size(),0), pd\_index(v.size()), pred(v.size());

int maxi = 0, x=0, j=0, ind=0;

for(int i=0;i<v.size();i++)

{

x = v[i];

j=lower\_bound(pd.begin(),pd.begin()+maxi,x) -pd.begin();

pd[j] = x;

pd\_index[j] = i;

if(j==maxi)

{

maxi++;

ind = i;

}

if(pred[i] == j) pd\_index[j-1] = -1;

}

int pos=maxi-1,k=v[ind];

asw.resize( maxi );

while ( pos >= 0 )

{

asw[pos--] = k;

ind = pred[ind];

k = v[ind];

}

}

## 3.2- Numero de Palindromes

vector<vector<bool> > dp(n + 1, vector<bool>(n + 1, 0));

for (int i = 1; i <= n; i++) dp[i][i] = true;

for (int i = n; i >= 1; i--) {

for (int j = i + 1; j <= n; j++) {

if (s[i - 1] == s[j - 1]) {

dp[i][j] = dp[i + 1][j - 1];

if (j - i <= 2) dp[i][j] = 1;

}

}

}

## 3.3 Divide and Conquer Example

const int MOD=1e9+7;

const int N=1010;

int dp[N][N],cost[N][N],v[N],pref[N],n,m;

void compDP(int j, int L, int R, int b, int e){

if(L>R) return ;

int mid=(L+R)/2;

int idx=-1;

for(int i=b;i<=min(mid,e);i++)

if(dp[mid][j]>dp[i][j-1]+cost[i+1][mid]){

idx=i;

dp[mid][j]=dp[i][j-1]+cost[i+1][mid];

}

compDP(j,L,mid-1,b,idx);

compDP(j,mid+1,R,idx,e);

}

## 3.4 Convex Hull Trick

bool domeioehlixo(int r1, int r2, int r3, int j)

{

return (B[j][r1]-B[j][r3])\*(A[j][r2]-A[j][r1])<(B[j][r1]-B[j][r2])\*(A[j][r3]-A[j][r1]);

}

void add(double a, double b, int j)

{

B[j].pb(b);

A[j].pb(a);

while(B[j].size()>=3 and domeioehlixo(B[j].size()-3,B[j].size()-2,B[j].size()-1,j))

{

B[j].erase(B[j].end()-2);

A[j].erase(A[j].end()-2);

}

}

double query(double isi,int j)

{

if(pont[j]>=B[j].size()) pont[j]=B[j].size()-1;

while(pont[j]<B[j].size()-1 and (A[j][pont[j]+1]\*isi + B[j][pont[j]+1] < A[j][pont[j]]\*isi + B[j][pont[j]])) pont[j]++;

return A[j][pont[j]]\*isi + B[j][pont[j]];

}

# 4- Grafos

## 4.1- Fluxo Maximo

### 4.1.1- Edmonds Karp

struct Edge {

int at,where;

ll cap;

void init(int \_at, ll \_cap, int \_where) { at = \_at, cap = \_cap, where = \_where; }

};

struct dad {

int at, up, down;

dad() {at=-1;}

dad(int \_at, int \_up, int \_down) { at = \_at, up = \_up, down = \_down; }

};

class MaxFlow {

private:

vector<vector<Edge> > g;

ll mf, f;

int s, t;

vector<dad> p;

public:

void augment(int v, ll minEdge)

{

if (v == s) {

f = minEdge;

return;

}

else if (p[v].at != -1) {

augment(p[v].at, min(minEdge, g[p[v].at][p[v].up].cap));

g[p[v].at][p[v].up].cap -= f;

g[v][p[v].down].cap += f;

}

}

void init(int N)

{

for (int i = 0; i < g.size(); i++) g[i].clear();

mf = 0, f = 0;

g.resize(N);

}

void addEdge(int u, int v, ll cap)

{

Edge A;

A.init(v, cap, g[v].size());

Edge B;

B.init(u, 0, g[u].size());

g[u].pb(A);

g[v].pb(B);

}

int maxFlow(int source, int sink)

{

s = source;

t = sink;

mf = 0;

while (true) {

f = 0;

vector<int> dist(g.size(), INF);

dist[s] = 0;

queue<int> q;

q.push(s);

p.clear();

p.resize(g.size());

while (!q.empty()) {

int u = q.front();

q.pop();

if (u == t) break;

for (int i = 0; i < g[u].size(); i++) {

Edge prox = g[u][i];

if (dist[prox.at] == INF and prox.cap > 0) {

dist[prox.at] = dist[u] + 1;

q.push(prox.at);

dad paizao(u, i, prox.where);

p[prox.at] = paizao;

}

}

}

augment(t, INF);

if (f == 0) break;

mf += f;

}

return mf;

}

};

### 4.1.2- Dinic ( Dilson)

struct Edge {

int v, rev;

int cap;

Edge(int v\_, int cap\_, int rev\_) : v(v\_), rev(rev\_), cap(cap\_) {}

};

struct MaxFlow {

vector<vector<Edge> > g;

vector<int> level;

queue<int> q;

int flow, n;

MaxFlow(int n\_) : g(n\_), level(n\_), n(n\_) {}

void addEdge(int u, int v, int cap)

{

if (u == v) return;

Edge e(v, cap, int(g[v].size()));

Edge r(u, 0, int(g[u].size()));

g[u].push\_back(e);

g[v].push\_back(r);

}

bool buildLevelGraph(int src, int sink)

{

fill(level.begin(), level.end(), -1);

while (not q.empty()) q.pop();

level[src] = 0;

q.push(src);

while (not q.empty()) {

int u = q.front();

q.pop();

for (auto e = g[u].begin(); e != g[u].end(); ++e) {

if (not e->cap or level[e->v] != -1) continue;

level[e->v] = level[u] + 1;

if (e->v == sink) return true;

q.push(e->v);

}

}

return false;

}

int blockingFlow(int u, int sink, int f)

{

if (u == sink or not f) return f;

int fu = f;

for (auto e = g[u].begin(); e != g[u].end(); ++e) {

if (not e->cap or level[e->v] != level[u] + 1) continue;

int mincap = blockingFlow(e->v, sink, min(fu, e->cap));

if (mincap) {

g[e->v][e->rev].cap += mincap;

e->cap -= mincap;

fu -= mincap;

}

}

if (f == fu) level[u] = -1;

return f - fu;

}

int maxFlow(int src, int sink)

{

flow = 0;

while (buildLevelGraph(src, sink))

flow += blockingFlow(src, sink, numeric\_limits<int>::max());

return flow;

}

};

### 4.2.1- Matching Maximo

class MaxFlow{

vi graph[N];

int match[N],us[N];

public:

MaxFlow(){};

void addEdge(int u, int v){

graph[u].pb(v);

}

int dfs(int u){

if(us[u]) return 0;

us[u]=1;

for(int v: graph[u]){

if(match[v]==-1 or (dfs(match[v]))){

match[v]=u;

return 1;

}

}

return 0;

}

int maxFlow(int n){

memset(match,-1,sizeof(match));

int ret=0;

for(int i=0;i<n;i++){

memset(us,0,sizeof(us));

ret+=dfs(i);

}

return ret;

}

};

## 4.2- Menor Caminho

### 4.2.1- Floyd Warshall

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

if (graph[i][j] != INF) pai[i][j] = i;

for (int k = 0; k < n; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (graph[i][j] > graph[i][k] + graph[k][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

pai[i][j] = pai[k][j];

}

}

}

}

## 4.3- Arvore Geradora Minima

int n, m;

vector<pair<int, ii> > edge;// aresta, u->v;

vector<int> pset;

void initset(int tam)

{

pset.resize(tam);

for (int i = 0; i < tam; i++) pset[i] = i;

}

int findset(int i)

{

if (pset[i] == i) return i;

return pset[i] = findset(pset[i]);

}

void unionset(int i, int j) { pset[findset(i)] = findset(j); }

bool issameset(int i,int j) { return findset(i)==findset(j);}

void kruskal() {

resp.clear();

sort(edge.begin(), edge.end());

initset(n);

int mst\_cost=0;

for(int i=0;i<m;i++){

pair<int,ii > front= edge[i];

if(!issameset(front.second.first, front.second.second)){

mst\_cost+=front.first;

unionset(front.second.first, front.second.second);

}

}

}

## 4.4- Kosaraju - com compressão

class kosaraju {

private:

vi usados;

vvi graph;

vvi trans;

vi pilha;

public:

kosaraju(int N)

{

graph.resize(N);

trans.resize(N);

}

void AddEdge(int u, int v)

graph[u].pb(v);

trans[v].pb(u);

}

void dfs(int u, int pass, int color)

{

usados[u] = color;

vi vizinhos;

if (pass == 1)

vizinhos = graph[u];

else

vizinhos = trans[u];

for (int j=0;j<vizinhos.size();j++) {

int v=vizinhos[j];

if (usados[v]==0) {

dfs(v, pass, color);

}

}

pilha.pb(u);

}

int SSC(int n)

{

pilha.clear();

usados.assign(n, 0);

for (int i = 0; i < n; i++) {

if (!usados[i]) dfs(i, 1, 1);

}

usados.assign(n, 0);

int color = 1;

for (int i = n - 1; i >= 0; i--) {

if (usados[pilha[i]] == 0) {

dfs(pilha[i], 2, color);

color++;

}

}

return color - 1;

}

vvi compression(int n)

{

int tam = SSC(n);

vvi resp;

vvi Trans;

resp.resize(tam);

Trans.resize(tam);

for (int u = 0; u < graph.size(); u++) {

for (int j=0;j<graph[u].size();j++) {

int v=graph[u][j];

if (usados[u] != usados[v]) {

resp[usados[u] - 1].pb(usados[v] - 1);

Trans[usados[v] - 1].pb(usados[u] - 1);

}

}

}

return Trans;

}

};

## 4.5- Isomorfismo Arvore

vvi children, subtreeLabels, tree, L;

vi pred, map;

int n;

bool compare(int a, int b) {

    return subtreeLabels[a] < subtreeLabels[b];

}

bool equals(int a, int b) {

    return subtreeLabels[a] == subtreeLabels[b];

}

void generateMapping(int r1, int r2) {

    map.resize(n);

    map[r1] = r2 - n;

    sort(children[r1].begin(), children[r1].end(), compare);

    sort(children[r2].begin(), children[r2].end(), compare);

    for (int i = 0; i < (int) children[r1].size(); i++) {

        int u = children[r1][i];

        int v = children[r2][i];

        generateMapping(u, v);

    }

}

vi findCenter(int offset = 0) {

    int cnt = n;

    vi a;

    vi deg(n);

    for (int i = 0; i < n; i++) {

        deg[i] = tree[i + offset].size();

        if (deg[i] <= 1) {

            a.push\_back(i + offset);

            --cnt;

        }

    }

    while (cnt > 0) {

        vi na;

        for (int i = 0; i < (int) a.size(); i++) {

            int u = a[i];

            for (int j = 0; j < (int) tree[u].size(); j++) {

                int v = tree[u][j];

                if (--deg[v - offset] == 1) {

                    na.push\_back(v);

                    --cnt;

                }

            }

        }

        a = na;

    }

    return a;

}

int dfs(int u, int p = -1, int depth = 0) {

    L[depth].push\_back(u);

    int h = 0;

    for (int i = 0; i < (int) tree[u].size(); i++) {

        int v = tree[u][i];

        if (v == p)

            continue;

        pred[v] = u;

        children[u].push\_back(v);

        h = max(h, dfs(v, u, depth + 1));

    }

    return h + 1;

}

bool rootedTreeIsomorphism(int r1, int r2) {

    L.assign(n, vi());

    pred.assign(2 \* n, -1);

    children.assign(2 \* n, vi());

    int h1 = dfs(r1);

    int h2 = dfs(r2);

    if (h1 != h2)

        return false;

    int h = h1 - 1;

    vi label(2 \* n);

    subtreeLabels.assign(2 \* n, vi());

    for (int i = h - 1; i >= 0; i--) {

        for (int j = 0; j < (int) L[i + 1].size(); j++) {

            int v = L[i + 1][j];

            subtreeLabels[pred[v]].push\_back(label[v]);

        }

        for (int j = 0; j < (int) L[i].size(); j++) {

            int v = L[i][j];

            sort(subtreeLabels[v].begin(), subtreeLabels[v].end());

        }

        sort(L[i].begin(), L[i].end(), compare);

        for (int j = 0, cnt = 0; j < (int) L[i].size(); j++) {

            if (j && !equals(L[i][j], L[i][j - 1]))

                ++cnt;

            label[L[i][j]] = cnt;

        }

    }

    if (!equals(r1, r2))

        return false;

    generateMapping(r1, r2);

    return true;

}

bool treeIsomorphism() {

    vi c1 = findCenter();

    vi c2 = findCenter(n);

    if (c1.size() == c2.size()) {

        if (rootedTreeIsomorphism(c1[0], c2[0]))

            return true;

        else if (c1.size() > 1)

            return rootedTreeIsomorphism(c1[1], c2[0]);

    }

    return false;

}

int main() {

    n = 5;

    vvi t1(n);

    t1[0].push\_back(1);

    t1[1].push\_back(0);

    t1[1].push\_back(2);

    t1[2].push\_back(1);

    t1[1].push\_back(3);

    t1[3].push\_back(1);

    t1[0].push\_back(4);

    t1[4].push\_back(0);

    vvi t2(n);

    t2[0].push\_back(1);

    t2[1].push\_back(0);

    t2[0].push\_back(4);

    t2[4].push\_back(0);

    t2[4].push\_back(3);

    t2[3].push\_back(4);

    t2[4].push\_back(2);

    t2[2].push\_back(4);

    tree.assign(2 \* n, vi());

    for (int u = 0; u < n; u++) {

        for (int i = 0; i < t1[u].size(); i++) {

            int v = t1[u][i];

            tree[u].push\_back(v);

        }

        for (int i = 0; i < t2[u].size(); i++) {

            int v = t2[u][i];

            tree[u + n].push\_back(v + n);

        }

    }

    bool res = treeIsomorphism();

    cout << res << endl;

    if (res)

        for (int i = 0; i < n; i++)

            cout << map[i] << endl;

}

## 4 5- Achar Pontes/Pontos de Articulação

class ponte{

private:

vvi graph;

vi usados;

vi e\_articulacao;

vi dfs\_low;;

vi dfs\_prof;

vector<ii> pontes;

int tempo;

public:

ponte(int N){

graph.clear();

graph.resize(N);

usados.assign(N,0);

dfs\_low.assign(N,0);

dfs\_prof.assign(N,0);

e\_articulacao.assign(N,0);

tempo=0;

}

void AddEdge(int u, int v){

graph[u].pb(v);

graph[v].pb(u);

}

void dfs(int u, int pai){

usados[u]=1;

int nf=0;

dfs\_low[u]= dfs\_prof[u] = tempo++;

for(int v: graph[u]){

if(!usados[v]){

dfs(v,u);

nf++;

if(dfs\_low[v]>=dfs\_prof[u] and pai!=-1) e\_articulacao[u]=true;

if(pai==-1 and nf>1) e\_articulacao[u]=true;

if(dfs\_low[v] > dfs\_prof[u]) pontes.pb(mp(u,v));

dfs\_low[u]=min(dfs\_low[u],dfs\_low[v]);

}

else if(v!=pai) dfs\_low[u]= min(dfs\_low[u], dfs\_prof[v]);

}

}

void olha\_as\_pontes(){

for(int i=0;i<graph.size();i++) if(!usados[i]) dfs(i,-1);

if(pontes.size()==0) cout<<" Que merda! nao tem ponte!"<<endl;

else{

for(ii i: pontes) cout<<i.first<<" "<<i.second<<endl;

}

}

void olha\_as\_art(){

for(int i=0;i<graph.size();i++) if(!usados[i]) dfs(i,-1);

for(int i=0;i<e\_articulacao.size();i++) if(e\_articulacao[i]) cout<<" OIAAA A PONTE "<<i<<endl;

}

};

## 4.6- LCA

const int N=100000;

const int M=22;

int P[N][M];

int big[N][M], low[N][M], level[N];

vii graph[N];

int n;

void dfs(int u, int last, int l)

{

level[u] = l;

P[u][0] = last;

for (ii v : graph[u])

if (v.first != last) {

big[v.first][0] = low[v.first][0] = v.second;

dfs(v.first, u, l + 1);

}

}

void process()

{

for (int j = 1; j < M; j++)

for (int i = 1; i <= n; i++) {

P[i][j] = P[P[i][j - 1]][j - 1];

big[i][j] = max(big[i][j - 1], big[P[i][j - 1]][j - 1]);

low[i][j] = min(low[i][j - 1], low[P[i][j - 1]][j - 1]);

}

}

int lca(int u, int v)

{

if (level[u] < level[v]) swap(u, v);

for (int i = M - 1; i >= 0; i--)

if (level[u] - (1 << i) >= level[v]) u = P[u][i];

if (u == v) return u;

for (int i = M - 1; i >= 0; i--) {

if (P[u][i] != P[v][i]) u = P[u][i], v = P[v][i];

}

return P[u][0];

}

int maximum(int u, int v, int x)

{

int resp = 0;

for (int i = M-1; i >= 0; i--)

if (level[u] - (1 << i) >= level[x]) {

resp = max(resp, big[u][i]);

u = P[u][i];

}

for (int i = M-1; i >= 0; i--)

if (level[v] - (1 << i) >= level[x]) {

resp = max(resp, big[v][i]);

v = P[v][i];

}

return resp;

}

int minimum(int u, int v, int x)

{

int resp = INF;

for (int i = M-1; i >= 0; i--)

if (level[u] - (1 << i) >= level[x]) {

resp = min(resp, low[u][i]);

u = P[u][i];

}

for (int i =M-1; i >= 0; i--)

if (level[v] - (1 << i) >= level[x]) {

resp = min(resp, low[v][i]);

v = P[v][i];

}

return resp;

}

## 4.7- Min cost max flow

int flow[N][N];

vector< pair<int, int> > g[N];

int n, m, k;

inline int ent(int a){ return a \* 2; }

inline int out(int a){ return a \* 2 + 1; }

inline void addEdge(int a, int b, int custo, int fluxo) {

flow[a][b] += fluxo;

g[a].push\_back(make\_pair(b, custo));

g[b].push\_back(make\_pair(a, -custo));

}

int src = N - 1, tgt = N - 2;

int dis[N], pai[N];

inline int dij() {

memset(dis, INF, sizeof dis);

memset(pai, -1, sizeof pai);

priority\_queue< pair<int, int> > q;

dis[src] = 0;

q.push(make\_pair(0, src));

while(!q.empty()) {

pair<int, int> foo = q.top(); q.pop();

int x = foo.second, cost = -foo.first;

if(dis[x] != cost) continue;

for(int i = 0; i < g[x].size(); ++i) {

int y = g[x][i].first, w = g[x][i].second;

if(flow[x][y] <= 0) continue;

if(dis[y] > dis[x] + w) {

dis[y] = dis[x] + w;

pai[y] = x;

q.push(make\_pair(-dis[y], y));

}

}

}

return dis[tgt] != INF;

}

int minCost() {

int maxFlow = 0;

int minC = 0;

while(dij()) {

int u = tgt;

int minFlow = INF;

while(pai[u] != -1) {

minFlow = min(minFlow, flow[pai[u]][u]);

u = pai[u];

}

maxFlow += minFlow;

minC += minFlow \* dis[tgt];

u = tgt;

while(pai[u] != -1) {

flow[pai[u]][u] -= minFlow;

flow[u][pai[u]] += minFlow;

u = pai[u];

}

}

if(maxFlow != n \* k) minC = -1;

return minC;

}

inline void init() {

memset(flow, 0, sizeof flow);

for(int i = 0; i < N; ++i) {

g[i].clear();

}

}

## 4.8- 2-Sat – com recuperacao

const int N = 510;

vi graph[N], rev[N];

int us[N];

stack<int> pilha;

int resposta[N];

void dfs1(int u)

{

us[u] = 1;

for (int v : graph[u])

if (!us[v]) dfs1(v);

pilha.push(u);

}

void dfs2(int u, int color)

{

us[u] = color;

for (int v : rev[u])

if (!us[v]) dfs2(v, color);

}

int Sat(int n)

{

for (int i = 0; i < n; i++)

if (!us[i]) dfs1(i);

int color = 1;

vi r;

memset(us, 0, sizeof(us));

while (!pilha.empty()) {

int topo = pilha.top();

r.pb(topo);

pilha.pop();

if (!us[topo]) dfs2(topo, color++);

}

for (int i = 0; i < n; i += 2) {

if (us[i] == us[i + 1]) return 0;

}

memset(resposta, -1, sizeof(resposta));

for (int i = r.size() - 1; i >= 0; i--) {

int vert = r[i] / 2;

int ok = r[i] % 2;

if (resposta[vert] == -1) resposta[vert] = !ok;

}

return 1;

}

inline void add(int u, int v)

{

graph[u].pb(v);

rev[v].pb(u);

}

inline int pos(int u) { return 2 \* u; }

inline int neg(int u) { return 2 \* u + 1; }

# 5- Estruturas de Dados

# 5.1- Segtree

## 5.1- RMQ- indice

class RMQ {

private:

vi A;

vi M;

public:

RMQ(vi &v)

{

A = v;

M.resize(4 \* v.size());

build(1, 0, v.size() - 1);

}

void build(int node, int b, int e)

{

if (b == e)

M[node] = b;

else {

build(2 \* node, b, (b + e) / 2);

build(2 \* node + 1, (b + e) / 2 + 1, e);

if (A[M[2 \* node]] <= A[M[2 \* node + 1]])

M[node] = M[2 \* node];

else

M[node] = M[2 \* node + 1];

}

}

int query(int node, int b, int e, int i, int j)

{

int p1, p2;

if (i > e || j < b) return -1;

if (b >= i and e <= j) return M[node];

p1 = query(2 \* node, b, (b + e) / 2, i, j);

p2 = query(2 \* node + 1, (b + e) / 2 + 1, e, i, j);

if (p1 == -1) return p2;

if (p2 == -1) return p1;

if (A[p1] <= A[p2]) return p1;

return p2;

}

void atualiza(int node, int b, int e, int i, int val)

{

if (i > e || i < b) return;

if (e == b) {

A[i] = val;

}

else {

atualiza(2 \* node, b, (b + e) / 2, i, val);

atualiza(2 \* node + 1, (b + e) / 2 + 1, e, i, val);

if (A[M[2 \* node]] <= A[M[2 \* node + 1]])

M[node] = M[2 \* node];

else

M[node] = M[2 \* node + 1];

}

}

};

## 5.2- RSQ Lazy-Propagation

class RSQ{

private:

vll A;

vll M;

vll lazy;

public:

RSQ(vll &v){

A=v;

M.resize(v.size()\*4);

lazy.assign(v.size()\*4,0);

build(1,0,v.size()-1);

}

void build(int node, int b, int e)

{

if(b==e){

M[node]=A[b];

return;

}

build(2\*node, b, (b+e)/2);

build(2\*node+1,(b+e)/2+1,e);

M[node]=M[2\*node]+M[2\*node+1];

}

void atualiza(int node, int b, int e, int i, int j, ll val)

{

if(lazy[node]!=0){

M[node]+=lazy[node];

if(b!=e){

ll inter=(e-b+1);

ll i1=(b+e)/2-b+1;

ll i2=e-(b+e)/2;

ll un=lazy[node]/inter;

lazy[2\*node]+=un\*i1;

lazy[2\*node+1]+=un\*i2;

}

lazy[node]=0;

}

if(i>e or j<b) return;

if(i<=b and j>=e){

ll inter=(e-b+1);

M[node]+=val\*inter;

if(b!=e){

ll i1=(b+e)/2-b+1;

ll i2=e-(b+e)/2;

lazy[2\*node]+=i1\*(ll)val;

lazy[2\*node+1]+=i2\*(ll)val;

}

return;

}

atualiza(2\*node,b,(b+e)/2,i,j,val);

atualiza(2\*node+1,(b+e)/2+1,e,i,j,val);

M[node]=M[2\*node]+M[2\*node+1];

}

ll query(int node, int b, int e, int i, int j)

{

if(i>e or j<b) return 0;

ll p1,p2;

if(lazy[node]!=0){

M[node]+=lazy[node];

if(b!=e){

ll inter=(e-b+1);

ll i1=(b+e)/2-b+1;

ll i2=e-(b+e)/2;

ll un=lazy[node]/inter;

lazy[2\*node]+=un\*i1;

lazy[2\*node+1]+=un\*i2;

}

lazy[node]=0;

}

if(i<=b and j>=e) return M[node];

p1= query(2\*node,b,(b+e)/2,i,j);

p2= query(2\*node+1,(b+e)/2+1,e,i,j);

return p1+p2;

}

};

## 5.4- BIT

class BIT {

private:

vi bit;

public:

BIT(int n) { bit.assign(n + 1, 0); }

int rsq(int b)

{

int sum = 0;

for (; b; b -= (b & (-b))) sum += bit[b];

return sum;

}

void adjust(int k, int v)

{

for (; k < bit.size(); k += (k & (-k))) bit[k] += v;

}

};

## 5.5- Disjoint Set Com Rank

class UnionFind {

private:

vector<int> p, rank;

public:

UnionFind(int N)

{

rank.assign(N, 0);

p.assign(N, 0);

for (int i = 0; i < N; i++) p[i] = i;

}

int findSet(int i)

{

if (p[i] == i) return i;

return p[i] = findSet(p[i]);

}

bool isSameSet(int i, int j)

{

if (findSet(i) == findSet(j))

return true;

else

return false;

}

void unionSet(int i, int j)

{

if (!isSameSet(i, j)) {

int x = findSet(i);

int y = findSet(j);

if (rank[x] > rank[y])

p[y] = x;

else {

p[x] = y;

if (rank[x] == rank[y]) rank[y]++;

}

}

}

};

## 5.6 Algoritmo de Mo

#include<bits/stdc++.h>

using namespace std;

const int N= 1e7+5;

#define ll long long

int tam\_block;

struct query{

int e,d,idx,lixo;

};

int A[N];

int ocor[N];

query fila[N];

bool compare(query a, query b){

a.idx= a.e/tam\_block;

b.idx= b.e/tam\_block;

if(a.idx==b.idx) return a.d < b.d;

return a.idx < b.idx;

}

void add(int x){

ocor[x]++;

}

void remove(int x){

ocor[x]--;

}

int main()

{

memset(A,0,sizeof(A));

memset(ocor,0,sizeof(ocor));

int n,m,k;

cin>>n>>m>>k;

vector<ll> resposta(m,0);

tam\_block= sqrt(n);

vector<int> prefix(n+1,0);

for(int i=0;i<n;i++) cin>>A[i];

for(int i=0;i<n;i++) prefix[i+1]=prefix[i] ^ A[i];

for(int i=0;i<m;i++)

{

cin>>fila[i].e>>fila[i].d;

fila[i].lixo=i;

}

sort(fila,fila+m,compare);

ll resp=0;

int R=-1,L=0;

for(int i=0;i<m;i++){

int qr=fila[i].d;

int qe=fila[i].e-1;

while(R > qr){

remove(prefix[R]);

resp-=ocor[prefix[R]^k];

R--;

}

while(R < qr){

R++;

resp+=ocor[prefix[R]^k];

add(prefix[R]);

}

while(L < qe){

remove(prefix[L]);

resp-=ocor[prefix[L]^k];

L++;

}

while(L > qe){

L--;

resp+=ocor[prefix[L]^k];

add(prefix[L]);

}

resposta[fila[i].lixo]=resp;

}

for(int i=0;i<m;i++) cout<<resposta[i]<<endl;

return 0;

}

## 5.7- Seg de vector

vi tr[5 \* N];

void build(int node, int b, int e)

{

if (b == e)

tr[node].pb(v[b]);

else {

build(2 \* node, b, (b + e) / 2);

build(2 \* node + 1, (b + e) / 2 + 1, e);

merget(tr[2\*node],tr[2\*node+1],tr[node]);

merge(tr[2 \* node].begin(), tr[2 \* node].end(), tr[2 \* node + 1].begin(),

tr[2 \* node + 1].end(), back\_inserter(tr[node]));

}

}

int query(int node, int b, int e, int i, int j, int k)

{

if (i > e or b > j) return 0;

if (i <= b and j >= e) {

int resp =

upper\_bound(tr[node].begin(), tr[node].end(), k) - tr[node].begin();

return tr[node].size() - resp;

}

return query(2 \* node, b, (b + e) / 2, i, j, k) +

query(2 \* node + 1, (b + e) / 2 + 1, e, i, j, k);

}

## 5.8- Sparce Table

int dp[200100][22];

int n;

int d[200100];

void build()

{

d[0]=d[1]=0;

for(int i=2;i<n;i++) d[i]=d[i>>1]+1;

for(int j=1;j<22;j++){

for(int i=0;i+(1<<(j-1))<n;i++){

dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);

}

}

}

int query(int i, int j){

int k=d[j-i];

int x=min(dp[i][k],dp[j-(1<<k)+1][k]);

return x;

}

# 6-Geometria

typedef struct sPoint {

double x, y;

sPoint() {}

sPoint(double \_x, double \_y) { x = \_x, y = \_y; }

} point;

typedef point vec;

## 6.1- Distancia de dois ponts (ao quadrado)

double sqDistPoints(point a, point b)

{

point p(a.x - b.x, a.y - b.y);

return p.x \* p.x + p.y \* p.y;

}

## 6.2- Distancia Ponto a Segmento (ao quadrado)

double sqDistPointLine(point a, point b, point c)

{

vec ab(b.x - a.x, b.y - a.y), ac(c.x - a.x, c.y - a.y);

double prod = cross(ab, ac);

prod = (prod \* prod) / (ab.x \* ab.x + ab.y \* ab.y);

vec bc(c.x - b.x, c.y - b.y), ba(a.x - b.x, a.y - b.y);

if (dot(ab, bc) > 1e-8) return sqDistPoints(c, b);

if (dot(ba, ac) > 1e-8) return sqDistPoints(c, a);

return prod;

}

## 6.3- Produto Vetorial

double cross(vec a, vec b) { return a.x \* b.y - a.y \* b.x; }

## 6.4- Produto Escalar

double dot(vec a, vec b) { return a.x \* b.x + a.y \* b.y; }

## 6.5- Minimum Enclosing Circle

const double eps = 1e-6;

#define CIRCLE circ

#define PT Ponto

#define MP 101

#define eps 1e-9

#define x first

#define y second

typedef double cood;

typedef int num;

typedef int point;

double resp;

cood x[MP], y[MP], ar, ax, ay;

int p[MP];

typedef pair<double, double> ponto;

typedef pair<double, double> Ponto;

double dista(ponto a, ponto b)

{

return sqrt((a.first - b.first) \* (a.first - b.first) +

(a.second - b.second) \* (a.second - b.second));

}

bool in(ponto a, pair<double, ponto> c)

{

if (dista(a, c.second) - eps < c.first) return true;

return false;

}

bool same(point a, point b)

{

return (fabs(x[a] - x[b]) < eps && fabs(y[a] - y[b]) < eps);

}

bool lexLess(point a, point b)

{

if (fabs(x[a] - x[b]) < eps) return y[a] < y[b];

return x[a] < x[b];

}

inline cood dist(cood xx, cood yy, point a)

{

return sqrt((xx - x[a]) \* (xx - x[a]) + (yy - y[a]) \* (yy - y[a]));

}

inline cood cP(point a, point b, point c)

{

return (x[a] - x[b]) \* (y[c] - y[b]) - (x[c] - x[b]) \* (y[a] - y[b]);

}

void findCircle(point a, point b, point c, cood& cx, cood& cy)

{

cx = 0.5 \* (x[a] \* x[a] + y[a] \* y[a] - x[b] \* x[b] - y[b] \* y[b]) \*

(y[b] - y[c]) -

0.5 \* (x[b] \* x[b] + y[b] \* y[b] - x[c] \* x[c] - y[c] \* y[c]) \*

(y[a] - y[b]),

cy = 0.5 \* (x[b] \* x[b] + y[b] \* y[b] - x[c] \* x[c] - y[c] \* y[c]) \*

(x[a] - x[b]) -

0.5 \* (x[a] \* x[a] + y[a] \* y[a] - x[b] \* x[b] - y[b] \* y[b]) \*

(x[b] - x[c]);

cx /= (x[a] - x[b]) \* (y[b] - y[c]) - (x[b] - x[c]) \* (y[a] - y[b]);

cy /= (x[a] - x[b]) \* (y[b] - y[c]) - (x[b] - x[c]) \* (y[a] - y[b]);

}

void spanCircle2(int k, point p0, point p1, cood& cx, cood& cy, cood& r)

{

cx = 0.5 \* (x[p0] + x[p1]);

cy = 0.5 \* (y[p0] + y[p1]);

r = dist(cx, cy, p0);

for (int i = 0; i < k; i++)

if (dist(cx, cy, p[i]) > r) {

findCircle(p0, p1, p[i], cx, cy);

r = dist(cx, cy, p[i]);

}

}

void spanCircle1(int k, point p0, cood& cx, cood& cy, cood& r)

{

cx = 0.5 \* (x[p0] + x[p[0]]);

cy = 0.5 \* (y[p0] + y[p[0]]);

r = dist(cx, cy, p0);

for (int i = 0; i < k; i++)

if (dist(cx, cy, p[i]) > r) spanCircle2(i, p0, p[i], cx, cy, r);

}

void spanCircle(int n, cood& cx, cood& cy, cood& r)

{

// Bem importante, retirar repetidos

sort(p, p + 1, lexLess);

n = unique(p, p + n) - p;

random\_shuffle(p, p + n);

if (n > 1) {

cx = 0.5 \* (x[p[0]] + x[p[1]]);

cy = 0.5 \* (y[p[0]] + y[p[1]]);

r = dist(cx, cy, p[1]);

for (int i = 2; i < n; i++)

if (dist(cx, cy, p[i]) > r) spanCircle1(i, p[i], cx, cy, r);

}

else {

cx = x[0];

cy = y[0];

r = 0.0;

}

}

void solve(vector<pair<double, double> >& v)

{

int N = v.size();

for (int i = 0; i < N; i++) {

x[i] = v[i].first;

y[i] = v[i].second;

p[i] = i;

}

spanCircle(N, ax, ay, ar);

}

6.6- Convex Hull

typedef struct sPoint {  
 int x, y;  
 sPoint(int \_x, int \_y) {x=\_x; y=\_y;}  
} point;

bool comp(point a, point b)  
{  
 if(a.x==b.x) return a.y<b.y;  
 return a.x<b.x;  
}

int cross(point a, point b, point c) //AB x BC  
{  
 a.x-=b.x;  a.y-=b.y;  
 b.x-=c.x;  b.y-=c.y;  
 return a.x\*b.y - a.y\*b.x;  
}  
  
bool isCw(point a, point b, point c) //Clockwise  
{ return cross(a, b, c) < 0; }

// >= if you want to put collinear points on the convex hull  
bool isCcw(point a, point b, point c) //Counter Clockwise  
{ return cross(a, b, c) > 0; }  
  
vector<point> convexHull(vector<point> p)  
{  
 vector<point> u, l; // Upper and Lower hulls  
   
 sort(p.begin(), p.end(), comp);  
 for(unsigned int i=0; i < p.size(); i++)  
 {  
 while(l.size() > 1 && !isCcw(l[l.size()-1], l[l.size()-2], p[i]))  
 l.erase(l.begin()+(l.size()-1));  
 l.push\_back(p[i]);  
 }  
   
 for(int i=p.size()-1; i >=0; i--)  
 {  
 while(u.size() > 1 && !isCcw(u[u.size()-1], u[u.size()-2], p[i]))  
 u.erase(u.begin()+(u.size()-1));  
 u.push\_back(p[i]);  
 }

       u.erase(u.begin()+(u.size()-1));  
       l.erase(l.begin()+(l.size()-1));  
       l.insert(l.end(), u.begin(), u.end());  
       return l;  
}

# 7- Problemas a se considerar

## 7.1- Invertion Count:

ll inversoes = 0;

void merge\_sort(int \*v, int x)

{

if (x == 1) return;

int tam\_esq = (x + 1) / 2, tam\_dir = x / 2;

int esq[tam\_esq], dir[tam\_dir];

for (int i = 0; i < tam\_esq; i++) esq[i] = v[i];

for (int i = 0; i < tam\_dir; i++) dir[i] = v[i+tam\_esq];

merge\_sort(esq, tam\_esq);

merge\_sort(dir, tam\_dir);

int i\_esq = 0, i\_dir = 0, i = 0;

while (i\_esq < tam\_esq or i\_dir < tam\_dir) {

if (i\_esq == tam\_esq) {

while (i\_dir != tam\_dir) {

v[i] = dir[i\_dir];

i\_dir++, i++;

}

}

else if (i\_dir == tam\_dir) {

while (i\_esq != tam\_esq) {

v[i] = esq[i\_esq];

i\_esq++, i++;

inversoes += i\_dir;

}

}

else {

if(esq[i\_esq]<=dir[i\_dir]){

v[i]=esq[i\_esq];

i++,i\_esq++;

inversoes+=i\_dir;

}

else{

v[i]=dir[i\_dir];

i++,i\_dir++;

}

}

}

}

# 8- Big Numbers

## 8.1- Tirar zero a Esquerda

void zero\_esq(string &resp){

string retorno=resp;

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

int i= resp.size()-1;

while(retorno[i]=='0' and i>0)

{

retorno.erase(i);

i--;

}

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

resp=retorno;

}

## 8.2- Somar 2 numeros

string sum\_big(string a, string b)

{

string resp;

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

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

if (a.size() <= b.size()) {

int carry = 0;

for (int i = 0; i < a.size(); i++) {

int x = b[i] - '0' + a[i] - '0' + carry;

resp.push\_back((char)(x % 10 + '0'));

carry = x / 10;

}

for (int i = a.size(); i < b.size(); i++) {

int x = b[i] - '0' + carry;

resp.push\_back((char)(x % 10 + '0'));

carry = x / 10;

}

if (carry > 0) resp.push\_back((char)(carry + '0'));

}

else {

int carry = 0;

for (int i = 0; i < b.size(); i++) {

int x = a[i] - '0' + b[i] - '0' + carry;

resp.push\_back((char)(x % 10 + '0'));

carry = x / 10;

}

for (int i = b.size(); i < a.size(); i++) {

int x = a[i] - '0' + carry;

resp.push\_back((char)(x % 10 + '0'));

carry = x / 10;

}

if (carry > 0) resp.push\_back((char)(carry + '0'));

}

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

zero\_esq(resp);

return resp;

}

## 8.3- Multiplicar 2 numeros

string mul\_big(string a, string b)

{

string resp;

resp.push\_back('0');

string temp;

int carry = 0;

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

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

for (int i = 0; i < a.size(); i++) {

temp.clear();

for (int k = 0; k < i; k++) temp.push\_back('0');

int x = a[i] - '0';

for (int j = 0; j < b.size(); j++) {

int y = b[j] - '0';

int novo = (x \* y + carry);

temp.push\_back((novo % 10) + '0');

carry = novo / 10;

}

if (carry > 0) temp.push\_back(carry + '0');

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

carry = 0;

resp = sum\_big(temp, resp);

}

zero\_esq(resp);

return resp;

}

# 9-FFTZINHO

typedef complex<double> base;

void fft (vector<base> & a, bool invert) {

int n = (int) a.size();

if (n == 1) return;

vector<base> a0 (n/2), a1 (n/2);

for (int i=0, j=0; i<n; i+=2, ++j) {

a0[j] = a[i];

a1[j] = a[i+1];

}

fft (a0, invert);

fft (a1, invert);

double ang = 2\*PI/n \* (invert ? -1 : 1);

base w (1), wn (cos(ang), sin(ang));

for (int i=0; i<n/2; ++i) {

a[i] = a0[i] + w \* a1[i];

a[i+n/2] = a0[i] - w \* a1[i];

if (invert)

a[i] /= 2, a[i+n/2] /= 2;

w \*= wn;

}

}

void multiply (const vector<int> & a, const vector<int> & b, vector<int> & res) {

vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());

size\_t n = 1;

while (n < max (a.size(), b.size())) n <<= 1;

n <<= 1;

fa.resize (n), fb.resize (n);

fft (fa, false), fft (fb, false);

for (size\_t i=0; i<n; ++i)

fa[i] \*= fb[i];

fft (fa, true);

res.resize (n);

for (size\_t i=0; i<n; ++i)

res[i] = int (fa[i].real() + 0.5);

}

# Template

#include<bits/stdc++.h>

using namespace std;

#define sc(a) scanf("%d", &a)

#define sc2(a,b) scanf("%d%d", &a, &b)

#define sc3(a,b,c) scanf("%d%d%d", &a, &b, &c)

#define scs(s) scanf("%s", s)

#define pri(x) printf("%d\n", x)

#define mp make\_pair

#define pb push\_back

#define BUFF ios::sync\_with\_stdio(false);

#define imprime(v) for(int X=0;X<v.size();X++) cout<<v[X]<<" "; cout<<endl;

#define grid(v) for(int X=0;X<v.size();X++){for(int Y=0;Y<v[X].size();Y++) cout<<v[X][Y]<<" ";cout<<endl;}

#define endl "\n"

const int INF= 0x3f3f3f3f;

const long double pi= acos(-1);

typedef long long int ll;

typedef long double ld;

typedef pair<int,int> ii;

typedef vector<int> vi;

typedef vector<ii> vii;

typedef vector< vector< int > > vvi;

typedef vector< vector< ii > > vvii;

const int MOD=1e9+7;

int main()

{

return 0;

}

**Compilar c++11: g++ -std=c++1 - stdlib=libc++**

# Coisas a saber:

## Numeros de Fibonacci

n : F(n)=factorisation

0 : 0  
1 : 1  
2 : 1  
3 : 2  
4 : 3  
5 : 5  
6 : 8 = 23  
7 : 13  
8 : 21 = 3 x 7  
9 : 34 = 2 x 17  
10 : 55 = 5 x 11  
11 : 89  
12 : 144 = 24 x 32  
13 : 233  
14 : 377 = 13 x 29  
15 : 610 = 2 x 5 x 61  
16 : 987 = 3 x 7 x 47  
17 : 1597  
18 : 2584 = 23 x 17 x 19  
19 : 4181 = 37 x 113  
20 : 6765 = 3 x 5 x 11 x 41  
21 : 10946 = 2 x 13 x 421  
22 : 17711 = 89 x 199  
23 : 28657  
24 : 46368 = 25 x 32 x 7 x 23  
25 : 75025 = 52 x 3001  
26 : 121393 = 233 x 521  
27 : 196418 = 2 x 17 x 53 x 109  
28 : 317811 = 3 x 13 x 29 x 281  
29 : 514229  
30 : 832040 = 23 x 5 x 11 x 31 x 61

## Primos para o Hash:

1000000009

1000000021

1000000033

1000000087

1000000093

1000000097

1000000103

1000000123

1000000181

1000000207

1000000223

1000000241

1000000271

1000000289

1000000297

1000000321

1000000349

1000000363

1000000403

1000000409

2000003273

2000003281

2000003293

2000003303

2000003333

2000003351

2000003353

2000003359

a\*ap-2=1 mod p

inv(a) = ap-2 mod p

ap-1= 1 mod p

## String Stream

int main()

{

int n, x;

string s;

stringstream ss;

while (scanf("%d", &n), n) {

getchar(); // removendo o \n

while (getline(cin, s)) { // getline pra ler a string toda

if (s.compare("0") == 0)

break; // parar se for 0

else {

ss.clear(); // limpando o stringstream

ss.str(s); // jogando a string no stringstream

while (ss >> x) { // lendo todos os ints do stringstream

printf("%d ", x);

}

printf("\n");

}

}

}

}

## Lista de Primos :

2 3 5 7 11 13 17 19 23 29

31 37 41 43 47 53 59 61 67 71

73 79 83 89 97 101 103 107 109 113

127 131 137 139 149 151 157 163 167 173

179 181 191 193 197 199 211 223 227 229

233 239 241 251 257 263 269 271 277 281

283 293 307 311 313 317 331 337 347 349

353 359 367 373 379 383 389 397 401 409

419 421 431 433 439 443 449 457 461 463

467 479 487 491 499 503 509 521 523 541

547 557 563 569 571 577 587 593 599 601

607 613 617 619 631 641 643 647 653 659

661 673 677 683 691 701 709 719 727 733

739 743 751 757 761 769 773 787 797 809

811 821 823 827 829 839 853 857 859 863

877 881 883 887 907 911 919 929 937 941

947 953 967 971 977 983 991 997 1009 1013

1019 1021 1031 1033 1039 1049 1051 1061 1063 1069

1087 1091 1093 1097 1103 1109 1117 1123 1129 1151

1153 1163 1171 1181 1187 1193 1201 1213 1217 1223

1229 1231 1237 1249 1259 1277 1279 1283 1289 1291

1297 1301 1303 1307 1319 1321 1327 1361 1367 1373

1381 1399 1409 1423 1427 1429 1433 1439 1447 1451

1453 1459 1471 1481 1483 1487 1489 1493 1499 1511

1523 1531 1543 1549 1553 1559 1567 1571 1579 1583

1597 1601 1607 1609 1613 1619 1621 1627 1637 1657

1663 1667 1669 1693 1697 1699 1709 1721 1723 1733

1741 1747 1753 1759 1777 1783 1787 1789 1801 1811

1823 1831 1847 1861 1867 1871 1873 1877 1879 1889

1901 1907 1913 1931 1933 1949 1951 1973 1979 1987

1993 1997 1999 2003 2011 2017 2027 2029 2039 2053

2063 2069 2081 2083 2087 2089 2099 2111 2113 2129

2131 2137 2141 2143 2153 2161 2179 2203 2207 2213

2221 2237 2239 2243 2251 2267 2269 2273 2281 2287

2293 2297 2309 2311 2333 2339 2341 2347 2351 2357

2371 2377 2381 2383 2389 2393 2399 2411 2417 2423

2437 2441 2447 2459 2467 2473 2477 2503 2521 2531

2539 2543 2549 2551 2557 2579 2591 2593 2609 2617

2621 2633 2647 2657 2659 2663 2671 2677 2683 2687

2689 2693 2699 2707 2711 2713 2719 2729 2731 2741

2749 2753 2767 2777 2789 2791 2797 2801 2803 2819

2833 2837 2843 2851 2857 2861 2879 2887 2897 2903

2909 2917 2927 2939 2953 2957 2963 2969 2971 2999

3001 3011 3019 3023 3037 3041 3049 3061 3067 3079

3083 3089 3109 3119 3121 3137 3163 3167 3169 3181

3187 3191 3203 3209 3217 3221 3229 3251 3253 3257

3259 3271 3299 3301 3307 3313 3319 3323 3329 3331

3343 3347 3359 3361 3371 3373 3389 3391 3407 3413

3433 3449 3457 3461 3463 3467 3469 3491 3499 3511

|  |
| --- |
| **// Other primes:** //    The largest prime smaller than 10      is 7. //    The largest prime smaller than 10^2    is 97. //    The largest prime smaller than 10^3    is 997. //    The largest prime smaller than 10^4    is 9973. //    The largest prime smaller than 10^5    is 99991. //    The largest prime smaller than 10^6    is 999983. //    The largest prime smaller than 10^7    is 9999991. //    The largest prime smaller than 10^8    is 99999989. //    The largest prime smaller than 10^9    is 999999937. //    The largest prime smaller than 10^10   is 9999999967. //    The largest prime smaller than 10^11   is 99999999977. //    The largest prime smaller than 10^12   is 999999999989. //    The largest prime smaller than 10^13   is 9999999999971. //    The largest prime smaller than 10^14   is 99999999999973. //    The largest prime smaller than 10^15   is 999999999999989. //    The largest prime smaller than 10^16   is 9999999999999937. //    The largest prime smaller than 10^17   is 99999999999999997. //    The largest prime smaller than 10^18   is 99999999999999998  **https://lh4.googleusercontent.com/lCIQhh40v-KFKA2BbFD0jkqNSXnIEu41t9bKDDK3KWRzZr4YnjQiYHnotUADt6q60qFN8lHJICjt00HnDwTd23GgYUUzb7XqUckGols8Isxp6iVVsDo2sZSRU1Fpxx5VupyFf0w** |

## /\* Josephus Problem - It returns the position to be, in order to not die. O(n)\*/

/\* With k=2, for instance, the game begins with 2 being killed and then n+2, n+4, ... \*/

ll josephus(ll n, ll k) {

 if(n==1) return 1;

 else return (josephus(n-1, k)+k-1)%n+1;

}

/\* Another Way to compute the last position to be killed… O ( d \* log n ) **\*/**

ll josephus(ll n, ll d) {

 ll K = 1;

 while (K <= (d−1)\*n) K = (d \* K + d − 2) / (d − 1);

 return d \* n + 1 − K;

}

## Identidade Binomial ( Binômio de Newton ):

**x+y)^n = \sum_{k=0}^n {n \choose k}x^{n-**

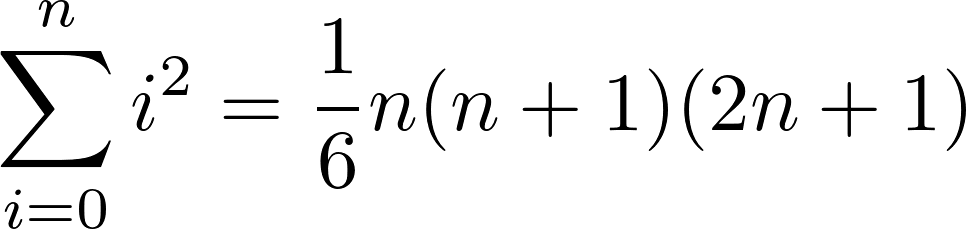
**1+x)^n = \sum_{k=0}^n {n \choose k}x^k.**

**// Maximal Prime Gaps:  
//** For numbers until 10^9 the maximal gap is 400.  
// For numbers until 10^18 the maximal gap is 1500.

**// Number of prime numbers in intervals:**// Há aprox. 8\*10^4 primos entre 1 e 10^6.  
// Há aprox. 6\*10^5 primos entre 1 e 10^7.  
// Há aprox. 5\*10^6  primos entre 1 e 10^8.  
// Há aprox. 5\*10^7 primos entre 1 e 10^9.

**Função de Ackermann-Péter**

**A(m, n) =
 \begin{cases}
 n+1 & **

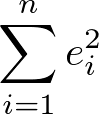
****

**Catalan Number_n = \frac{1}{n+1}{2n\choose n} = \frac{(2n)!}{(n+1)!\,n!} = \prod\**_n = {2n\choose n} - {2n\choose n+1} = {1\over n+1}{2n\choose n} \q

* The number of balanced expressions built from n pairs of parentheses.
* The number of paths in an n × n grid that stays on or below the diagonal.
* The number of words of size 2n over the alphabet Σ = {a, b} having an equal number of a symbols and b symbols containing no prefix with more a symbols than b symbols.
* *Cn* is the number of different ways a convex polygon with *n* + 2 sides can be cut into triangles by connecting vertices with straight lines

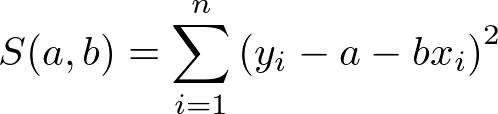
**Método dos mínimos quadrados**

Queremos estimar valores de determinada variável **y**. Para isso, consideramos os valores de outra variável **x** que acreditamos ter poder de explicação sobre **y** conforme a fórmula:

https://lh3.googleusercontent.com/H8f4CAxfEDKjNzzRb94qpRamXE9RlFToOzcadP9sA3g7KKF-v1WiyWvDdx6mc2i0AjcEuXw2OxdmtMwEpk5ZzN8wS2e6oFSLj-BBHcuW9qIlsX8aeFjgH-tADv_1OdmcykDbw-Ta

O método dos mínimos quadrados minimiza a soma dos quadrado dos resíduos, ou seja, minimiza

A ideia por trás dessa técnica é que, minimizando a soma do quadrado dos resíduos, encontraremos **a** e **b** que trarão a menor diferença entre a previsão de **y** e o **y** realmente observado.

Substituindo https://lh6.googleusercontent.com/HyT8NinxcB2irm10Rbbi7mm7i0HQ8lwvluVlCMymTWPnWo7mRkck6KFu1zevH83Kaq4DMsHufulCwGo0D9-FGBdCW9R8z1QfkhUC5OZieKun6owK-_-lUQSxUsKd_C3Dl_WsaLllpor https://lh3.googleusercontent.com/pDMYpzMIWhGGv1o_P1F1_QwEKTEDcnYZSD5IlWjrgYoQbC2wozKOfqAsC1BynnlbHnKaguWVIF3VyuDjHatwH6gBsIzATKH6k3Qr4uS0obJ-_IGOB7i69Nx1qfqlXFL3oMZ0pE7x, temos:. O quadrado das distâncias entre os pontos e a sua projeção ortogonal na função definida por **a** e **b**.

O **a** e **b** ótimos são definidos por:      https://lh5.googleusercontent.com/LTBeuIaZHGnYio7K2eyWCVC7fG9gjFjbjTeiBgjuwgv3GXLmojODUYHNMdUg_CsVBlY1XIiXImnpOVbKLIYQCuWri90_PsTiDtgvLOiP0E_Irj1A1OtAcCvym1IwvOyStvCr0tpm         e        