**TEAM REFERENCE**

**ACM-ICPC**

**ÍNDICE**

1. **Fórmulas de aritmética modular ………………………………….…. 1**
2. **Teoría de números …………………………………………………………... 1**
3. **Fórmula Catalana (Árboles Binarios) ………………………………… 1**
4. **Dígitos de n! …………………………………………………………………….. 1**
5. **Suma de una serie geométrica …………………………………………. 1**
6. **Factores primos ……………………………………………………………….. 1**
7. **Operaciones con Bits ……………………………………………………….. 2**
8. **Calcular el determinante de una matriz ……………………….…… 3**
9. **Buscar divisores de un número …………………………………...…… 3**
10. **Resolver SEL de NxN con Gauss ………………………………………… 3**
11. **Buscar el Rank de una matriz ……………………………………………. 4**
12. **Cantidad de dígitos de n! ………………………………………………….. 4**
13. **Cantidad de números palíndromes …………………………………… 4**
14. **Cantidad de números primos en n! ………………………………...… 4**
15. **Descomponer número en sumandos sin repetir ……………….. 4**
16. **Cantidad de dígitos 0-9 desde1 hasta n …………………………….. 4**
17. **Cantidad de divisores, suma y multiplicación ……………………. 5**
18. **Criba de Erastótenes …………………………………………………………. 5**
19. **Función Euler Totien (cant coprimos con n de 1 a n) …………. 5**
20. **Exponenciación modular …………………………………………………… 5**
21. **Euclides extendido (división modular) ………………………………. 5**
22. **Calcular n! mod P ……………………………………………………………… 5**
23. **Máximo común divisor (GCD) ……………………………..……………. 5**
24. **Inverso modular ………………………………………..……………………… 5**
25. **Newton Paphson (raíz cuadrada) ……………………………………… 6**
26. **Polard Ro (descomponer en factores primos) ………….……….. 6**
27. **Rabin Miller (Primality Test) ……………………………………………… 6**
28. **Cantidad de ceros al final de n! ………………………………………… 7**
29. **Calcular término de Fibonacci MOD …………………………………. 7**
30. **Lados de un triángulo conociendo las 3 medianas ……………. 7**
31. **Radio de la circunferencia inscrita en un triángulo …………… 7**
32. **Volumen de un Tetraedro ………………………………………….…….. 7**
33. **Área de intersección de 2 circunferencias ………………….…….. 8**
34. **Plantilla de geometría computacional ……………………….……... 8**
35. **Área de un polígono ………………………………………………….……… 8**
36. **Centro de circunferencia dado 3 puntos …………………….…….. 8**
37. **Determinar si un polígono es convexo ………………………….…… 9**
38. **Determinar si un punto pertenece a un polígono convexo … 9**
39. **Distancia de un punto a una recta ……………………………………. 10**
40. **Distancia máxima entre dos puntos …………………………………. 10**
41. **Radio del mayor círculo inscrito en un polígono convexo …. 11**
42. **Intersección de dos círculos ……………………………………………… 11**
43. **Intersección línea-círculo ………………………………………………….. 12**
44. **Intersección línea-línea …………………………………………………….. 12**
45. **Perímetro de la unión de rectángulos ……………………………….. 12**
46. **Determinar si existe un substring cuadrado en un string …… 13**
47. **Longest increasing substring ……………………………………………… 13**
48. **String manager ………………………………………………………………….. 13**
49. **Merge Sort ………………………………………………………………………… 13**
50. **Mochila múltiples elementos ……………………………………………. 14**
51. **Mochila elementos únicos …………………………..……………………. 14**
52. **Torres de Hanoi …………………………………………………..……………. 14**
53. **Salto de caballo ………………………………………………………………... 14**
54. **Bellman Ford ………………………….…………………………………………. 14**
55. **Dijkstra …………………………………………………………………..…………. 15**
56. **Template para C++ ……….………………………………………..…………. 16**
57. **Day of week ………………….……………………………………………….…. 16**

**ARITMETICA MODULAR**

* (x + y) mod n = ((x mod n) + (y mod n)) mod n
* (x - y) mod n = ((x mod n) - (y mod n) + n) mod n
* xy mod n = (x mod n)(y mod n) mod n

**TEORIA DE NUMEROS**

* Dos números primos son primos relativos (cooprimos) su GCD es 1.
* Si **a** y **p** son cooprimos, entonces se cumple: **a^p – a** es divisible por **p**
* Cada entero positivo es la suma de a lo sumo 4 cuadrados.
* Todo primo de la forma 4n+1 se puede descomponer únicamente como suma de dos cuadrados.
* Todo entero es o bien un cubo o la suma de nueve cubos; además todo entero es o una cuarta potencia o la suma de como máximo 19 cuartas potencias.
* Todo entero par es la suma de dos primos y todo entero impar es o un primo o la suma de tres primos.
* Un primo de la forma 3n + 1 puede expresarse únicamente en la forma: x^2 + 3y^2.
* Para todo primo p, la cantidad (p-1)! + 1 es divisible por p; además si la cantidad es divisible por q, entonces q es primo*.*

**Fórmula para calcular el número de *Arboles Binarios* que se pueden formar con N nodos:**

2n! / (n! \* n! \* (n+1))

C (N / K) = N! / ((N-K)!\*K!)

P (N / K) = N! / (N-K)!

**Digitos de n!**

int factDigits(int n) {

return (int)(0.5\*Math.log10(2\*n\*Math.PI) + n\*Math.log10(n/Math.E)+1);

}

**Suma de una serie geometrica:**

S = a \* ( 1 – r ^ n ) / ( 1 - r)

**Factores primos**

static ArrayList<Integer> facts(int x) {

boolean[] num = new boolean[x];

ArrayList<Integer> dP = new ArrayList<Integer>();

int[] primos = new int[x];

int pos = 0;

int sq = (int) Math.sqrt(x);

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

if (!num[i])

for (int j = 2 \* i; j < x; j += i)

num[j] = true;

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

if (!num[i])

primos[pos++] = i;

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

while (x % primos[i] == 0) {

dP.add(primos[i]);

x /= primos[i];

}

}

return dP;

}

**BIT OPERATIONS**

number of bits 1 in “n” in binary.

//via 1

int count(long n) {

int num = 0;

for (int i = 0; i < 32; i++)

if (n & (1 << i))

num++;

return num;

}

//via 2

int count(long n) {

int num = 0;

if (n)

do

num++;

while (n &= n - 1);

return num;

}

// number of parity bits of a binary representation of a number of n.

int parity(long n) {

n = ((0xFFFF0000 & n) >> 16) ^ (n & 0xFFFF);

n = ((0xFF00 & n) >> 8) ^ (n & 0xFF);

n = ((0xF0 & n) >> 4) ^ (n & 0xF);

n = ((12 & n) >> 2) ^ (n & 3);

n = ((2 & n) >> 1) ^ (n & 1);

return n;

}

//Determine the least significant bit 1 in the binary representation of n.

int low1(long n) {

return n ^ (n & (n - 1));

}

// most significant bit with value 1 in the binary representation of n.

int high1(long n) {

long num = 0;

if (!n)

return -1;

if (0xFFFF0000 & n) {

n = (0xFFFF0000 & n) >> 16;

num += 16;

}

if (0xFF00 & n) {

n = (0xFF00 & n) >> 8;

num += 8;

}

if (0xF0 & n) {

n = (0xF0 & n) >> 4;

num += 4;

}

if (12 & n) {

n = (12 & n) >> 2;

num += 2;

}

if (2 & n) {

n = (2 & n) >> 1;

num += 1;

}

return 1 << num;

}

//Index of the most significant bit 1 in the binary representation of n.

int indexHigh1(long n) {

n = n | (n >> 1);

n = n | (n >> 2);

n = n | (n >> 4);

n = n | (n >> 8);

n = n | (n >> 16);

return count(n) - 1;

}

//Determine if n is a power of 2

int isTwoPower(long n) {

return (n & (n - 1)) == 0;

}

**CALCULAR EL DETERMINANTE DE UNA MATRIZ**

const double EPS = 1E-9;

int n;

scanf("%d", &n);

vector<vector<double> > a(n, vector<double>(n));

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

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

scanf("%lf", &a[i][j]);

double det = 1;

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

int k = i;

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

if (abs(a[j][i]) > abs(a[k][i]))

k = j;

if (abs(a[k][i]) < EPS) {

det = 0;

break; }

swap(a[i], a[k]);

if (i != k)

det = -det;

det \*= a[i][i];

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

a[i][j] /= a[i][i];

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

if (j != i && abs(a[j][i]) > EPS)

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

a[j][k] -= a[i][k] \* a[j][i];}

cout << det;

**BUSCAR DIVISORES DE UN NUMERO**

ll divisor[1000000];

ll ndiv;

void find\_div(ll N) {

ll i, p, old;

if (N < 0)

N = -N;

ndiv = 0;

divisor[ndiv++] = 1;

for (p = 2; p <= 46337 && p \* p <= N; p++) {

old = ndiv;

while (N % p == 0) {

for (i = 0; i < old; i++) {

ll tmp = divisor[ndiv - old] \* p;

divisor[ndiv] = tmp;

ndiv++;}

N /= p;}}

if (N > 1) {

old = ndiv;

for (i = 0; i < old; i++) {

divisor[ndiv] = divisor[ndiv - old] \* N;

ndiv++; }}

//sort(divisor, divisor + ndiv); }

**GAUSS PARA SISTEMA DE ECUACIONES DE NxN**

int n; //cantidad de variables

scanf("%d", &n);

vector<vector<double> > a(n, vector<double>(n + 1));

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

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

scanf("%lf", &a[i][j]);

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

int k = i;

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

if (abs(a[j][i]) > abs(a[k][i]))

k = j;

swap(a[i], a[k]);

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

a[i][j] /= a[i][i];

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

if (j != i)

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

a[j][k] -= a[i][k] \* a[j][i]; }

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

printf("%.15lf\n", a[i][n]);

**BUSCAR EL RANK DE UNA MATRIZ**

int n, m; //filas y columnas de la matrix

int a[10][10]; //matrix

scanf("%d%d", &n, &m);

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

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

scanf("%d", &a[i][j]);

const double EPS = 1E-9;

int rank = max(n, m);

vector<char> line\_used(n);

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

int j;

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

if (!line\_used[j] && abs(a[j][i]) > EPS)

break;

if (j == n)

--rank;

else {

line\_used[j] = true;

for (int p = i + 1; p < m; ++p)

a[j][p] /= a[j][i];

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

if (k != j && abs(a[k][i]) > EPS)

for (int p = i + 1; p < m; ++p)

a[k][p] -= a[j][p] \* a[k][i]; }}

cout << rank;

**CANTIDAD DE DIGITOS EN N!**

//para n > 3

int n = 20;

int PI = 2 \* acos(0);

cout << (int)(0.5 \* log10(2 \* n \* PI) + n \* log10(n / M\_E) + 1);

**CANTIDAD DE NUMEROS PALINDROMES**

//cant de palindromes con ‘n’ o menos digitos

a(n) = 2 \*(10^(n/2) -1) si n es par

a(n) = 11\*(10^((n-1)/2))-2 si n es impar

**CANTIDAD DE NUMEROS PRIMOS EN N!**

//p es el numero primo, to es el limite

//aux inicialmente = p, inicialmente s = 0

#define get\_cnt\_primes(p, to)\

while(aux <= to) {\

s += to / aux;\

aux \*= p;\

}

**DESCOMPONER NÚMERO EN SUMADOS SIN REPETICION**

//primera llamada DescEnSum(n, 1, 0, "", n);

//array es un StringBuffer para acelerar la impresión en Java

void DescEnSum(int n, int pos, int ninc, String s, int numero) {

if (ninc == numero) {

if(array[numero] == null)

array[numero] = new StringBuffer();

array[numero].append(numero + " = " + s.substring(0, s.length() - 3) + "\n");

}

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

DescomponEnSumando(n - i, i, ninc + i, s + i + " + ", numero);}

**CANTIDAD DE DIGITOS ENTRE 0-9 DESDE 1 HASTA N (DIGIT COUNT)**

typedef long long ll;

//sol es un arreglo de int, donde va a estar la respuesta

void DigitCount(int n, ll \*sol) {

ll aux = n, sum = 0, p = 1, d;

while (aux) {

d = aux % 10, aux /= 10;

sol[d] += ((n % p) + 1);

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

sol[i] += p;

for (int i = 0; i < 10; i++)

sol[i] += sum \* d;

sol[0] -= p;

sum = p + 10 \* sum;

p \*= 10; } }

**CANTIDAD DE DIVISORES, SUMA DIVISORES, MULT DIVISORES**

N=p^a\*q^b\*r^c //descomposicion de N

CantDiv = D = (a+1)\*(b+1)\*(c+1)

SumaDiv = FOR(i,k)

sum\*=(prim[i]^(cant[i]+1)-1)/(prim[i]-1)

ProdDiv = P = N^(D/2)=Sqrt(N^D)

**CRIBA DE ERASTOTENES**

inline void Erastotenes() {

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

primes[i] = i;

for (int i = 2; i \* i < MAXN; ++i)

if (primes[i])

for (int j = i \* i; j < MAXN; j += i)

primes[j] = 0;

remove(primes, primes + MAXN, 0); }

**EULER TOTIEN FUNCTION**

//EulerTotientFunction(n) es la cantidad de numeros de 1 a n coprimos con n

ll EulerTotientFunction(ll n){

ll ans = n;

for(ll i=2;i\*i<= n;i++){

if(n %i==0) ans -= ans/i;

while(n%i==0) n/=i; }

if(n>1) ans -=ans/n;

return ans; }

**EXPONENCIACION MODULAR**

|  |  |
| --- | --- |
| int modulo(long a, long b, long c) {  long x = 1, y = a;  while (b > 0) {  if (b % 2 == 1) {  x = (x \* y) % c;}  y = (y \* y) % c;  b /= 2;}  return (int) (x % c); } | int bigmod(int b, int p, int m){  int mask = 1;  int pow2 = b % m;  int r = 1;  while (mask){  if (p & mask) r = (r \* pow2) % m;  pow2 = (pow2 \* pow2) % m;  mask <<= 1;  }  return r; } |

**EUCLIDES EXTENDIDO**

//para division modular se llama de la sgte forma:

//a = MOD, b = den, x = 1, y = 0

//la respuesta es ((((y + MOD) % MOD) \* num)) % MOD

ll extendedGCD(ll a, ll b, ll& x, ll& y) {

ll g = a;

x = 1;

y = 0;

if (b != 0) {

g = extendedGCD(b, a % b, y, x);

y -= (a / b) \* x; }

return g; }

**CALCULAR N! MOD P**

int factMod (int n, int p) {

int res = 1,i;

while (n > 1) {

if ((n/p) & 1)

res = (res \* (p-1)) % p;

for (i=n%p; i > 1;i--)

res = (res \* i) % p;

n /= p; }

return res % p; }

**GCD**

int GCD(int a, int b) {

int r;

while (b != 0) {

a = a % b;

r = a;

a = b;

b = r;}

return a; }

**INVERSO MODULAR**

ll invMod(ll a, ll m, ll &inv) {

ll x, y;

if (GCDext(a, m, x, y) != 1)

return 0 ; // noSolucion

inv = (x + m) % m;

return 1; }

**NEWTON PAPHSON**

//calcular la raiz cuadrada de un numero

double NewtonRaphson(double n) {

double x = 1, nx;

while (true) {

nx = (x + n / x) / 2;

if (fabs(x-nx) < EPS) break;

x = nx; }

return x; }

int NewtonRaphson(int n){

int a = 1; bool low = 0;

while(1) {

int nx=(a+n/a)/2;

if (a==nx || (nx>a &&low))

break;

low =nx<a, a=nx; }

return a; }

C[n] => FOR(k=0,n-1) C[k] \* C[n-1-k]

C[n] => Comb(2\*n,n) / (n + 1)

C[n] => 2\*(2\*n-3)/n \* C[n-1]

**POLARD RO**

//descomponer en factores primos, un numero de hasta 64 bit

ll factores[70];

int nfactor;

ll pollard\_rho(ll c, ll num) {

ll x = rand() % num;

ll i = 1, k = 2, y = x, comDiv;

do {

i++;

if ((x = mul(x, x, num) - c) < 0)

x += num;

if (x == y)

break;

comDiv = GCD((y - x + num) % num, num);

if (comDiv > 1 && comDiv < num)

return comDiv;

if (i == k) {

y = x;

k <<= 1; }

} while (true);

return num; }

void fFindFactor(ll num) {

if (is\_prime(num)) {

factores[nfactor++] = num;

return; }

ll factor = num + 1;

while (factor >= num)

factor = pollard\_rho(rand() % (num - 1) + 1, num);

fFindFactor(factor);

fFindFactor(num / factor); }

**RABIN MILLER**

//chequear si un numero es primo, hasta 64 bit

typedef unsigned long long ull;

#define MAXP 10

const int prime[MAXP] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };

ull N;

ull mul(ull a, ull b, ull m) {

if (!b)

return 0;

ull ret = 2 \* mul(a, b / 2, m) % m;

if (b % 2)

ret = (ret + a) % m;

return ret; }

ull power(ull x, ull n, ull m) {

ull ret = 1;

for (; n; x = mul(x, x, m), n /= 2)

if (n % 2)

ret = mul(ret, x, m);

return ret; }

bool rabin\_miller(ull p) {

if (p < 2 || (p > 2 && !(p % 2)))

return false;

for (int i = 0; i < MAXP && prime[i] < p; i++)

if (power(prime[i], p - 1, p) != 1)

return false;

return true; }

**CANTIDAD DE CEROS AL FINAL DE N!**

long N, B, i, j, p, c, noz, k;

while (true) {

st = new StringTokenizer(cin.readLine());

N = Long.parseLong(st.nextToken());

B = Long.parseLong(st.nextToken());

if (N == 0 && B == 0)

break;

noz = N;

j = B;

boolean test = true;

for (i = 2; i <= B && j > 1; i++) {

if(test && isPrime(j, 4))

if(i <= j)

i = j;

if (j % i == 0) {

test = true;

p = 0;

while (j % i == 0) {

p++;

j /= i;}

c = 0;

k = N;

while (k / i > 0) {

c += k / i;

k /= i;}

noz = Math.min(noz, c / p);

} else

test = false; }

System.out.println(noz); }

**CALCULAR TERMINO DE FIBBONACCI MODULO MOD**

long fibo(long n) {

long[][] fib = { { 1, 1 }, { 1, 0 } };

long[][] ret = { { 1, 0 }, { 0, 1 } };

long[][] tmp = { { 0, 0 }, { 0, 0 } };

while (n > 0) {

if (n % 2 == 1) {

tmp[0][0] = 0; tmp[0][1] = 0; tmp[1][0] = 0; tmp[1][1] = 0;

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

for (int j = 0; j < 2; j++)

for (int k = 0; k < 2; k++)

tmp[i][j] = (tmp[i][j] + ret[i][k] \* fib[k][j])% MOD;

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

for (int j = 0; j < 2; j++)

ret[i][j] = tmp[i][j];}

tmp[0][0] = 0;tmp[0][1] = 0;tmp[1][0] = 0;tmp[1][1] = 0;

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

for (int j = 0; j < 2; j++)

for (int k = 0; k < 2; k++)

tmp[i][j] = (tmp[i][j] + fib[i][k] \* fib[k][j]) % MOD;

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

for (int j = 0; j < 2; j++)

fib[i][j] = tmp[i][j];

n /= 2; }

return ret[0][1] % MOD; }

**ENCONTRAR LOS LADOS DE UN TRIANGULO CONOCIENDO LAS 3 MEDIANAS**

l1 = (2 \* sqrt(2 \* (pow(b, 2) + pow(c, 2)) - pow(a, 2))) / 3;

l2 = (2 \* sqrt(2 \* (pow(a, 2) + pow(c, 2)) - pow(b, 2))) / 3;

l3 = (2 \* sqrt(2 \* (pow(b, 2) + pow(a, 2)) - pow(c, 2))) / 3;

**RADIO DE LA CIRCUNFERENCIA INSCRITA EN UN TRIANGULO**

//a, b, c son los lados del triangulo

double s = (a + b + c) / 2;

double r = sqrt((s - a) \* (s - b) \* (s - c) / s);

**VOLUMEN DE UN TETRAHEDRO**

double Volumen1() {

double COSB = (WX\*WX+WZ\*WZ-XZ\*XZ)/(2.0\*WX\*WZ);

double COSC = (WX\*WX+WY\*WY-XY\*XY)/(2.0\*WX\*WY);

double COSD = (WZ\*WZ+WY\*WY-YZ\*YZ)/(2.0\*WZ\*WY);

Raiz = pow(1+2\*COSB\*COSC\*COSD-COSB\*COSB-COSC\*COSC-COSD\*COSD,0.5);

return Raiz\*(WX\*WY\*WZ/6.0); }

double VOL(double u, double v, double w, double U, double V, double W) {

return Volumen1(); //Usando Metodo 1...

WX = u, WY = v, WZ = w, YZ = U, XZ = V, XY = W;

return Volumen2(u,v,w,U,V,W); //Usando Metodo 2

//Importante ...

// u, v, y w son 3 lados con un vertice comun...

// U lado opuesto a u (no tienen vertices comunes).

// V lado opuesto a v (no tienen vertices comunes).

// W lado opuesto a w (no tienen vertices comunes). }

**AREA DE INTERSECCION DE 2 CIRCUNFERENCIAS**

//center[i] es el centro de la circunferencia, deben estar en la misma recta

//r es el radio de la circunferencia, beben ser iguales

double d = centers[i] - centers[i - 1];

double ang = Math.acos(d / (2 \* r));

double s = ang \* r \* r;

double t = (d / 2) \* Math.pow(r \* r - (d \* d / 4), 0.5);

double areaInters = 2 \* (s - t);

**PLANTILLA DE GEOMETRIA COMPUTACIONAL**

const double EPS = 1e-8;

const double oo = 1e12;

const double PI = 3.14159265358979323846264;

#define X real()

#define Y imag()

typedef complex<double> P;

typedef vector<P> Pol;

//cross product en 3D

//i = y1\*z2 - y2\*z1;

//j = x2\*z1 - x1\*z2;

//k = x1\*y2 - x2\*y1;

struct circle {

P p;

double r;

circle() {}

circle(P x, double rr) {

p = x, r = rr; } };

struct L: public vector<P> { //Linea

L(P a, P b) {

push\_back(a);

push\_back(b); } };

inline bool operator<(const P a, const P b) {

return a.X != b.X ? a.X < b.X : a.Y < b.Y; }

double cross(P a, P b) { //1

return imag(conj(a) \* b); }

double dot(P a, P b) { //2

return (conj(a) \* b).X; }

double distSqr(P &p1, P &p2){

return (p1.X-p2.X)\*(p1.X-p2.X) + (p1.Y-p2.Y)\*(p1.Y-p2.Y); }

bool contain(circle c,P p){

return distSqr(c.p,p)<= c.r\*c.r; }

**AREA DE UN POLYGONO**

struct point {

int x, y; };

//P es un poligono ordenado anticlockwise.

//Si es clockwise,retorna el area negativa

//Si no esta ordenado no sirve.

//P[0]!=P[n-1]

double PolygonArea(const vector<point>&p) {

double r = 0.0;

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

int j = (i + 1) % p.size();

r += p[i].x \* p[j].y - p[j].x \* p[i].y; }

return r / 2.0; }

**CENTRO DE CIRCUNFERENCIA DADO 3 PTOS**

Point calcuateCenter(Point p1, Point p2, Point p3) {

double A1 = p2.y - p1.y;

double B1 = -(p1.x - p2.x);

double midpointx1 = (p1.x + p2.x) / 2;

double midpointy1 = (p1.y + p2.y) / 2;

double D1 = B1 \* midpointx1 + A1 \* midpointy1;

double A2 = p3.y - p2.y;

double B2 = -(p2.x - p3.x);

double midpointx2 = (p2.x + p3.x) / 2;

double midpointy2 = (p2.y + p3.y) / 2;

double D2 = B2 \* midpointx2 + A2 \* midpointy2;

double det = B1 \* A2 - B2 \* A1;

if (det == 0)

return null;

double x = (A2 \* D1 - A1 \* D2) / det;

double y = (B1 \* D2 - B2 \* D1) / det;

return new Point(x, y);}

**DETERMINAR SI UN POLYGONO ES CONVEXO**

Returns positive if a-b-c makes a left turn.

Returns negative if a-b-c makes a right turn.

Returns 0.0 if a-b-c are colineal.

double turn(const point& a, const point& b, const point& c) {

double z = (b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x);

if (fabs(z) < 1e-9)

return 0.0;

return z; }

Returns true if polygon p is convex.

False if its concave or it can’t be determined

(For example, if all points are colineal we can’t make a choice).

bool isConvexPolygon(const vector<point>& p) {

int mask = 0;

int n = p.size();

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

int j = (i + 1) % n;

int k = (i + 2) % n;

double z = turn(p[i], p[j], p[k]);

if (z < 0.0) {

mask |= 1;

} else if (z > 0.0) {

mask |= 2; }

if (mask == 3)

return false; }

return mask != 0; }

**DETERMINAR SI UN PTO PERTENECE A UN POLYGONO CUALQUIERA**

//los ptos tienen que estar ordenados clockwise o counterclockwise

//manejar el caso de que el pto este en la misma arista del polygono

//en caso de que halla que incluirlo

#define EPS 1e-9

double polarAngle(point p) {

if (fabs(p.x) <= EPS && fabs(p.y) <= EPS)

return -1.0;

if (fabs(p.x) <= EPS)

return (p.y > EPS ? 1.0 : 3.0) \* acos(0);

double theta = atan(1.0 \* p.y / p.x);

if (p.x > EPS)

return (p.y >= -EPS ? theta : (4 \* acos(0) + theta));

return (2 \* acos(0) + theta); }

bool pointInPoly(point p, point poly[], int n) {

double ang = 0.0;

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

point v = (point) {poly[i].x - p.x, poly[i].y - p.y};

point w = (point) {poly[j].x - p.x, poly[j].y - p.y};

double va = polarAngle(v);

double wa = polarAngle(w);

double xx = wa - va;

if (va < -0.5 || wa < -0.5 || fabs(fabs(xx) - 2 \* acos(0)) < EPS) {

//ON THE EDGE

ang += 2 \* acos(0);

continue; }

if (xx < -2 \* acos(0))

ang += xx + 4 \* acos(0);

else if (xx > 2 \* acos(0))

ang += xx - 4 \* acos(0);

else

ang += xx; }

return (ang \* ang > 1.0); }

**DETERMINAR SI UN PTO PERTENECE A UN POLYGONO CONVEXO**

//no pueden existir ptos duplicados y el area no puede ser cero

//los vertices tienen que estar ordenados anti-clockwise

struct pt {

int x, y; };

struct ang {

int a, b; };

bool operator <(const ang & p, const ang & q) {

if (p.b == 0 & &q.b == 0)

return p.a < q.a;

return p.a \* 1ll \* q.b < p.b \* 1ll \* q.a; }

long long sq(pt & a, pt & b, pt & c) {

return a.x \* 1ll \* (b.y - c.y) + b.x \* 1ll \* (c.y - a.y)

+ c.x \* 1ll \* (a.y - b.y); }

int main() {

int n; cin >> n; vector<pt> p(n); int zero\_id = 0;

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

scanf("%d %d", &p[i].x, &p[i].y);

if (p[i].x < p[zero\_id].x || (p[i].x == p[zero\_id].x && p[i].y < p[zero\_id].y))

zero\_id = i;}

pt zero = p[zero\_id]; rotate(p.begin(), p.begin() + zero\_id, p.end());

p.erase(p.begin()); --n; vector<ang> a(n);

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

a[i].a = p[i].y - zero.y;

a[i].b = p[i].x - zero.x;

if (a[i].a == 0)

a[i].b = a[i].b < 0 ? -1 : 1;}

for (;;) {

pt q;// leer pto

bool in = false;

if (q.x >= zero.x)

if (q.x == zero.x && q.y == zero.y)

in = true;

else {

ang my = { q.y - zero.y, q.x - zero.x };

if (my.a == 0)

my.b = my.b < 0 ? -1 : 1;

vector<ang>::iterator it = upper\_bound(a.begin(), a.end(), my);

if (it == a.end() & &my.a == a[n - 1].a & &my.b == a[n - 1].b)

it = a.end() - 1;

if (it != a.end() && it != a.begin()) {

int p1 = int(it - a.begin());

if (sq(p[p1], p[p1 - 1], q) <= 0)

in = true; }}

puts(in ? "INSIDE" : "OUTSIDE"); } }

**DISTANCIA DE UN PTO A UNA RECTA**

double dot(double[] A, double[] B, double[] C) {

double[] AB, BC; AB = new double[2]; BC = new double[2];

AB[0] = B[0] - A[0]; AB[1] = B[1] - A[1]; BC[0] = C[0] - B[0];

BC[1] = C[1] - B[1];

double dot = AB[0] \* BC[0] + AB[1] \* BC[1];

return dot;}

double cross(double[] A, double[] B, double[] C) {

double[] AB, AC; AB = new double[2];AC = new double[2];

AB[0] = B[0] - A[0];AB[1] = B[1] - A[1]; AC[0] = C[0] - A[0];

AC[1] = C[1] - A[1];

double cross = AB[0] \* AC[1] - AB[1] \* AC[0];

return cross;}

double distance(double[] A, double[] B) {

double d1 = A[0] - B[0]; double d2 = A[1] - B[1];

return Math.sqrt(d1 \* d1 + d2 \* d2);}

double linePointDist(double[] A, double[] B, double[] C, boolean isSegement) {

double dist = cross(A, B, C) / distance(A, B);

if (isSegement) {

double dot1 = dot(A, B, C);

if (dot1 > 0)

return distance(B, C);

double dot2 = dot(B, A, C);

if (dot2 > 0)

return distance(A, C);}

return Math.abs(dist); }

**DISTANCIA MAXIMA ENTRE 2 PTOS**

double diameter(Pol pt) {//24, 1

int is=0,js=0, n=pt.size();

FAB(i,1,n){

if(pt[i].Y >pt[is].Y) is=i;

if(pt[i].Y <pt[js].Y) js=i; }

double maxd=norm(pt[is]-pt[js]);

int i,maxi,j,maxj;

i = maxi = is; j = maxj = js;

do {

if(cross(pt[(i+1)%n]-pt[i],

pt[(j+1)%n]-pt[j])>=0)

j=(j+1)%n; else i=(i+1)%n;

if (norm(pt[i]-pt[j])>maxd){

maxd =norm(pt[i]-pt[j]);

maxi=i; maxj=j;

} }while(i!=is || j!=js);

return maxd; }

**RADIO DEL MAYOR CIRCULO INSCRITO EN UN POLYGONO CONVEXO**

const double EPS = 1E-9;

#define INF 1000000000

int steps = 60;

struct pt {

double x, y; };

struct line {

double a, b, c; };

double dist(double x, double y, line & l) {

return abs(x \* l.a + y \* l.b + l.c); }

double radius(double x, double y, vector<line> & l) {

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

double res = INF;

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

res = min(res, dist(x, y, l[i]));

return res; }

double y\_radius(double x, vector<pt> & a, vector<line> & l) {

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

double ly = INF, ry = -INF;

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

int x1 = a[i].x, x2 = a[(i + 1) % n].x, y1 = a[i].y, y2 =

a[(i + 1) % n].y;

if (x1 == x2)

continue;

if (x1 > x2)

swap(x1, x2), swap(y1, y2);

if (x1 <= x + EPS & &x - EPS <= x2) {

double y = y1 + (x - x1) \* (y2 - y1) / (x2 - x1);

ly = min(ly, y);

ry = max(ry, y);} }

for (int sy = 0; sy < steps; ++sy) {

double diff = (ry - ly) / 3;

double y1 = ly + diff, y2 = ry - diff;

double f1 = radius(x, y1, l), f2 = radius(x, y2, l);

if (f1 < f2)

ly = y1;

else

ry = y2; }

return radius(x, ly, l); }

int main() {

int n;

vector<pt> a(n);//poligono

vector<line> l(n);

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

l[i].a = a[i].y - a[(i + 1) % n].y;

l[i].b = a[(i + 1) % n].x - a[i].x;

double sq = sqrt(l[i].a \* l[i].a + l[i].b \* l[i].b);

l[i].a /= sq, l[i].b /= sq;

l[i].c = -(l[i].a \* a[i].x + l[i].b \* a[i].y); }

double lx = INF, rx = -INF;

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

lx = min(lx, a[i].x);

rx = max(rx, a[i].x); }

for (int sx = 0; sx < steps; ++sx) {

double diff = (rx - lx) / 3;

double x1 = lx + diff, x2 = rx - diff;

double f1 = y\_radius(x1, a, l), f2 = y\_radius(x2, a, l);

if (f1 < f2)

lx = x1;

else

rx = x2;

}

double ans = y\_radius(lx, a, l);

printf("%.7lf", ans); }

**INTERSECCION DE 2 CIRCULOS**

//supongamos que el circulo 1 está en el origen, de lo contrario solo hay que trasladarlo

//con ambos circulos tenemos el sistema:

x^2+y^2=r1^2

(x-x2)^2+(y-y2)^2=r2^2

//sustraemos la segunda ecuacion de la primera y obtenemos el sistema:

x^2+y^2=r1^2

x(-2x2)+y(-2y2)+(x2^2+y2^2+r1^2-r2^2)=0

//reducir el problema al problema de la interseccion

//de un circulo y una line:

Ax+By+C=0,A=-2x2,B=-2y2,C=x2^2+y2^2+r1^2-r2^2

**INTERSECCION LINEA CIRCULO**

//el centro del circulo debe estar en el origen de coordenadas, sino solo hay que trasladarlo

//a, b, c son los coeficientes de la ecuacion de la recta

double r, a, b, c; // input

#define EPS 1e-9

void line\_circle\_inters() {

double x0 = -a \* c / (a \* a + b \* b), y0 = -b \* c / (a \* a + b \* b);

if (c \* c > r \* r \* (a \* a + b \* b) + EPS)

puts("no points");

else if (abs(c \* c - r \* r \* (a \* a + b \* b)) < EPS) {

puts("one point");

cout << x0 << ' ' << y0 << '\n';

} else {

double d = r \* r - c \* c / (a \* a + b \* b);

double mult = sqrt(d / (a \* a + b \* b));

double ax, ay, bx, by;

ax = x0 + b \* mult;

bx = x0 - b \* mult;

ay = y0 - a \* mult;

by = y0 + a \* mult;

puts("2 points");

cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n'; } }

**INTERSECCION LINEA LINEA**

//si lo que tenemos es un segmento delimitado por 2 ptos

//entonces primero tenemos que encontrar A, B, C para cada segmento

A = y2-y1

B = x1-x2

C = A\*x1+B\*y1

//pto interseccion

double det = A1\*B2 - A2\*B1

if(det == 0){

//Lines are parallel

}else{

double x = (B2\*C1 - B1\*C2)/det

double y = (A1\*C2 - A2\*C1)/det }

**PERIMETRO DE LA UNION DE RECTANGULOS**

#define MAXC 1 << 17

#define LEFT( v ) 2 \* ( v ) + 1

#define RIGHT( v ) 2 \* ( v ) + 2

int N, last, size;

long long perimeter;

struct event {

int start, lo, hi, flag;

bool operator < ( const event &e ) const { return start < e.start; } };

vector< event > events[2];

int amount[ 3 \* MAXC ], times[ 3 \* MAXC ];

void update( int node, int lo, int hi, int& start, int& end, int& value ) {

if ( lo > end || hi < start ) return;

if ( start <= lo && hi <= end )

times[node] += value;

else {

int mid = ( lo + hi ) / 2;

update( LEFT( node ), lo, mid, start, end, value );

update( RIGHT( node ), mid + 1, hi, start, end, value ); }

if ( !times[node] )

amount[node] = ( lo == hi ) ? 0 :

amount[ LEFT( node ) ] + amount[ RIGHT( node ) ];

else amount[node] = hi - lo + 1;}

void lineSweep( vector< event >& ls ) {

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

last = 0; size = ls.size();

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

update( 0, 0, MAXC, ls[i].lo, ls[i].hi, ls[i].flag );

perimeter += abs( last - amount[0] );

last = amount[0]; }}

int main() {

scanf( "%d", &N );

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

int x1, x2, y1, y2;

scanf( "%d %d %d %d", &x1, &y1, &x2, &y2 );

if ( x1 > x2 ) swap( x1, x2 );

if ( y1 > y2 ) swap( y1, y2 );

events[0].push\_back( ( event ) { y1, x1, x2 - 1, +1 } );

events[0].push\_back( ( event ) { y2, x1, x2 - 1, -1 } );

events[1].push\_back( ( event ) { x1, y1, y2 - 1, +1 } );

events[1].push\_back( ( event ) { x2, y1, y2 - 1, -1 } ); }

lineSweep( events[0] );

lineSweep( events[1] );

printf( "%lld\n", perimeter ); }

**DETERMINAR DADO UN STRING SI EXISTE UN SUBSTRING CUADRADO**

//un substring es un cuadrado si se cumple por ejemplo: "abab" es cuadrado por tener "ab" consecutivo

//no importa en que lugar de la cadena aparezca, pero si tiene que ser consecutivo

bool rtest(char \*u, int m, char \*v, int n) {

for (int i = n, j, q = n; i >= 0; i = max(j, i / 2) - 1) {

for (j = i; j >= 0 && m - 1 - i + j >= 0 && u[m - 1 - i + j] == v[j]; --j);

if (j < 0)

return true;

if (i + j <= q) {

for (q = i + j + 1; q > i && v[q] == v[q - i - 1]; --q);

if (q <= i)

return true;}}

return false; }

bool rec(char \*s, char \*r, int n) {

int m = n / 2;

if (n <= 1)

return false;

if (rec(s, r + n - m, m) || rec(s + m, r, n - m))

return true;

if (rtest(s, m, s + m, n - m) || rtest(r, n - m, r + n - m, m))

return true;

return false; }

bool square(char \*s, int n) {

char r[n]; copy(s, s + n, r); reverse(r, r + n);return rec(s, r, n);}

**LONGEST INCREASING SUBSTRING**

const int inf = 99999999;

#define index distance(A.begin(), lower\_bound(A.begin(), A.end(), a[i]))

vector<int> lis\_fast(const vector<int> & a) {

const int n = a.size(); vector<int> A(n, inf); vector<int> id(n);

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

id[i] = index;

A[id[i]] = a[i];}

int m = \*max\_element(id.begin(), id.end());

vector<int> b(m + 1);

for (int i = n - 1; i >= 0; --i)if (id[i] == m) b[m--] = a[i];return b; }

**STRING MANAGER**

/\*Vector rad[] is really the length of the palindromes,

that is, rad[i] is the length of the palindrome centered

at i (odd palindromes are in rad[ 2 \* i ] and even ones

in rad[ 2 \* i + 1 ])\*/

#define MAXLEN 100000

int n, i, j, k;

int rad[ 2 \* MAXLEN ]; char s[MAXLEN];

int main() {

scanf( "%s", s );

n = strlen( s );

for ( i = 0, j = 0; i < 2 \* n - 1; i += k ) {

while ( i - j >= 0 && i + j + 1 < 2 \* n &&

s[ ( i - j ) / 2 ] == s[ ( i + j + 1 ) / 2 ] )

j++;

rad[i] = j;

for ( k = 1; k <= rad[i] && rad[ i - k ] != rad[i] - k; k++ )

rad[ i + k ] = min( rad[ i - k ], rad[i] - k );

j = max( j - k, 0 ); } }}

**MERGE SORT**

long cont;//cantidad minima de cambios de posiciones adyacentes

static long[] a, tmp;//a contiene los valores a ordenar

MergeSort( 0, t - 1 );

void MergeSort(int lo, int hi) {

if (lo == hi)

return;

int mid = (lo + hi) >> 1;

MergeSort(lo, mid);

MergeSort(mid + 1, hi);

int cnt = 0;

int llo = lo, lhi = mid, rlo = mid + 1, rhi = hi;

while (llo <= lhi && rlo <= rhi)

if (a[llo] < a[rlo])

tmp[cnt++] = a[llo++];

else {

cont += rlo - llo - (rlo - (mid + 1));

tmp[cnt++] = a[rlo++];}

while (llo <= lhi)

tmp[cnt++] = a[llo++];

while (rlo <= rhi)

tmp[cnt++] = a[rlo++];

cnt = 0;

for (int i = lo; i <= hi; i++)

a[i] = tmp[cnt++];}}

**MOCHILA MULTIPLES ELEMENTOS**

int capacidad, cantidadEl;

int APORTE[10010], PESOS[10010], M[10010];

int main() {

memset(M, 0, sizeof(M));

scanf("%d%d", &capacidad, &cantidadEl);

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

scanf("%d%d", &APORTE[i], &PESOS[i]);

for (int i = PESOS[0]; i <= capacidad; i++)

M[i] = (i / PESOS[0]) \* (APORTE[0]);

for (int i = 1; i < cantidadEl; i++)

for (int k = PESOS[i]; k <= capacidad; k++)

M[k] = max(M[k], M[k - PESOS[i]] + APORTE[i]);

printf("%d\n", M[capacidad]);}

**MOCHILA ELEMENTOS UNICOS**

//array contiene el peso de los objetos

//mat es donde se va ha computar la dinamica

sort(array + 1, array + n);

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

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

(array[i] <= j) ? mat[i][j] = max(mat[i - 1][j], mat[i - 1][j - array[i]] + array[i]):mat[i][j] = mat[i - 1][j];

printf("%d", mat[n][h]);

**TORRES DE HANOI**

int n;

void move( int n, char from, char to, char aux ) {

if ( n == 1 )

printf( "Move disk from %c to %c\n", from, to );

else {

move( n - 1, from, aux, to );

printf( "Move disk from %c to %c\n", from, to );

move( n - 1, aux, to, from ); }}

int main() {

scanf( "%d", &n );

move( n, 'A', 'C', 'B' ); }

**SALTO DE CABALLO**

//cantidad minima necesaria para ir de una posicion a otra

//solo se puede saltar al igual que un caballo de ajedrez

ll SaltoCaballo(ll x1,ll y1,ll x2,ll y2){

ll dx =abs(x2-x1);

ll dy =abs(y2-y1);

ll lb= max(dx+1 , dy + 1)/2;

lb = max(lb, (dx + dy + 2)/3);

while((lb % 2) != (dx+ dy)%2) lb++;

if(abs(dx)==1 && !dy) return 3;

if(abs(dy)==1 && !dx) return 3;

if(abs(dx)==2 && abs(dy)==2) return 4;

return lb;

}

**BELLMAN FORD (**camino más corto en un Grafo dirigido ponderado**)**

#define MAXV 1000

#define MAXE 5000

#define oo 1000000000

struct Edge {

int u, v, w;

} edge[MAXE];

int V, E, source;int u, v, w;bool neg\_cycle;

int d[MAXV], p[MAXV];

void find( int idx ) {

if ( idx == source ) return;

find( p[idx] );

printf( " -> %d", idx + 1 );}

int main() {

scanf( "%d %d %d", &V, &E, &source );

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

scanf( "%d %d %d", &u, &v, &w );

edge[i] = ( Edge ) { u - 1, v - 1, w }; }

/\* Bellman-Ford Algorithm \*/

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

d[i] = oo;

d[ --source ] = 0;

neg\_cycle = false;

for ( int i = 0; i < V - 1; i++ )

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

u = edge[j].u;

v = edge[j].v;

w = edge[j].w;

if ( d[u] + w < d[v] ) { /\* relaxation \*/

d[v] = d[u] + w;

p[v] = u;} }

/\* Checking for negative-weight cycles \*/

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

if ( d[ edge[i].u ] + edge[i].w < d[ edge[i].v ] ) {

neg\_cycle = true;

break;}

if ( neg\_cycle )

printf( "Negative-weight cycle encountered!!!\n" );

else {

printf( "Shortest paths from source = %d:\n", source + 1 );

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

if ( i != source ) {

printf( "Vertex %d: %d", i + 1, source + 1 );

find( i );

printf( " Cost: %d\n", d[i] ); } }}

**DIJKSTRA**

define MAXV 50000

#define MAXE 100000

#define oo 1000000000

typedef pair< int, int > pii;

struct edge {

int v, w, next;

} edges[MAXE];

int V, E, source;int u, v, w, dist;

int p[MAXV], d[MAXV];

int main() {

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

scanf( "%d %d %d", &V, &E, &source );

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

scanf( "%d %d %d", &u, &v, &w );

u--; v--;

edges[i] = ( edge ) { v, w, p[u] };

p[u] = i;

}

/\* Dijkstra's Algorithm \*/

priority\_queue< pii, vector< pii >, greater< pii > > Q;

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

d[i] = oo;

d[ --source ] = 0;

for ( Q.push( make\_pair( 0, source ) ); !Q.empty(); Q.pop() ) {

pii top = Q.top();

u = top.second;

dist = top.first;

if ( dist <= d[u] )

for ( int i = p[u]; i != -1; i = edges[i].next ) {

v = edges[i].v; w = edges[i].w;

if ( d[u] + w < d[v] ) {

d[v] = d[u] + w;

Q.push( make\_pair( d[v], v ) ); } } } }

**TEMPLATE PARA C++**

#include <algorithm>

#include <numeric>

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <stack>

#include <cstdio>

#include <cstdlib>

#include <cctype>

#include <cassert>

#include <cmath>

#include <complex>

#include <memory.h>

#include <time.h>

#include <list>

using namespace std;

typedef long long ll;

template<typename T> string str(const T &a) {

ostringstream s;

s << a;

return s.str();

}

template<typename T> ll Int(const T &a) {

stringstream s;

s << a;

ll r;

s >> r;

return r;

}

template<typename T> double Double(const T &a) {

stringstream s;

s << a;

double r;

s >> r;

return r;

}

**DAY OF WEEK**

**int** DayOfWeek(**int** d, **int** m, **int** y) {

**if** (m < 3) {

y--;

m += 10;

} **else**

m -= 2;

**int** c = y / 100;

y %= 100;

c = y - 2 \* c + d + y / 4 + c / 4;

**return** ((**int**) (2.6 \* m - 0.2) + c + 7) % 7;

}