***Học viện công nghệ bưu chính viễn thông HN***



**TEAM 3**

**NOTEBOOK**

***MUC LUC***

[NUMBER THEORY 4](#_Toc96164476)

[I Prime 4](#_Toc96164477)

[Primality tests MillerRabin 4](#_Toc96164478)

[Sieve of Eratosthenes 4](#_Toc96164479)

[Sieve of segment(L, R) 5](#_Toc96164480)

[Counting the number prime < n 6](#_Toc96164481)

[II Number of divisor , sum of divisor 6](#_Toc96164482)

[Pollard rho 7](#_Toc96164483)

[Euler’s totient function 8](#_Toc96164484)

[III Modular Arithmetic 9](#_Toc96164485)

[Find modular Inverse Using Extended Euclidean 9](#_Toc96164486)

[Find the Modular Inverse using Binary Exponentiation 9](#_Toc96164487)

[Find the Modular Inverse for every number modulo m 9](#_Toc96164488)

[Find x, y in equation ax + by = gcd(a, b) 10](#_Toc96164489)

[Find x, y in equation ax + by = c 10](#_Toc96164490)

[Fractorial Modular p 11](#_Toc96164491)

[IV Combinatorics 11](#_Toc96164492)

[Conbinatorics important 11](#_Toc96164493)

[Stars and bars 12](#_Toc96164494)

[The Inclusion – Exclusion Principle 12](#_Toc96164495)

[DYNAMIC PROGRAMMING 13](#_Toc96164496)

[Quy hoạch động trạng thái 13](#_Toc96164497)

[MAXTRIX 14](#_Toc96164498)

[GRAPH 16](#_Toc96164499)

[disjoint-set 16](#_Toc96164500)

[Dijkstra 16](#_Toc96164501)

[Bellman Ford (Shortest Path Faster Algorithm improvement) 17](#_Toc96164502)

[Floyd Warshall 18](#_Toc96164503)

[LCA 18](#_Toc96164504)

[KRUSKA 19](#_Toc96164505)

[Khớp và cầu 20](#_Toc96164506)

[Thành phần liên thông mạnh 21](#_Toc96164507)

[BIG – INTEGER 22](#_Toc96164508)

[DATA STRUCTURES 24](#_Toc96164509)

[ST + Lazy 24](#_Toc96164510)

[BIT 25](#_Toc96164511)

[Stack 26](#_Toc96164512)

[Trie 27](#_Toc96164513)

[HashTable 30](#_Toc96164514)

[KMP 30](#_Toc96164515)

[GEOMTRY 31](#_Toc96164516)

[Template 31](#_Toc96164517)

[Kiem tra 2 doan thang cat nhau 32](#_Toc96164518)

[Kiểm tra 1 điểm có nằm trong đoạn hay không 32](#_Toc96164519)

[Vị trí tương đối của điểm với đa giac (inside, outside, bound) 32](#_Toc96164520)

[Khoang cach ngăn nhất giữa 2 điêm 33](#_Toc96164521)

NUMBER THEORY

I Prime

Primality tests MillerRabin

**--------------**MillerRabin**-----------**

https**:**//www.spoj.com/problems/PON/

https**:**//ideone.com/UGF6MR

n **<=** 1e18

**using** u64 **=** uint64\_t**;**

**using** u128 **=** \_\_uint128\_t**;**

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**)** **{** // returns true if n is prime, else returns false.

**if** **(**n **<** 2**)**

**return** **false;**

int r **=** 0**;**

u64 d **=** n **-** 1**;**

**while** **((**d **&** 1**)** **==** 0**)** **{**

d **>>=** 1**;**

r**++;**

**}**

**for** **(**int a **:** **{**2**,** 3**,** 5**,** 7**,** 11**,** 13**,** 17**,** 19**,** 23**,** 29**,** 31**,** 37**})** **{**

**if** **(**n **==** a**)**

**return** **true;**

**if** **(**check\_composite**(**n**,** a**,** d**,** r**))**

**return** **false;**

**}**

**return** **true;**

**}**

Sieve of Eratosthenes

//------------- Sàng nguyên tố ----------------------

O**(**nlog**(**n**))**

vector**<**int**>** prime**(**N**,** 0**);**

void sieve**(){**

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

**if(**prime**[**i**]** **==** 0**){**

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

**if(**prime**[**j**]** **==** 0**)** prime**[**j**]** **=** i**;**

**}**

**}**

**}**

**for(**int i **=** 2**;** i **<** N**;** i**++)** **if(**prime**[**i**]** **==** 0**){**

prime**[**i**]** **=** i**;**

**}**

**}**

Sieve of segment(L, R)

//Find Prime in range(L, R) R - L + 1 <= 1e7 && R < 1e12;

//Time complexity of this approach is O((R−L+1)loglog(R) + √RloglogR√R).

//https://www.spoj.com/problems/PRIME1/

//https://ideone.com/EUb7vb

vector**<**char**>** segmentedSieve**(**long long L**,** long long R**)** **{**

// generate all primes up to sqrt(R)

long long lim **=** sqrt**(**R**);**

vector**<**char**>** mark**(**lim **+** 1**,** **false);**

vector**<**long long**>** primes**;**

**for** **(**long long i **=** 2**;** i **<=** lim**;** **++**i**)** **{**

**if** **(!**mark**[**i**])** **{**

primes**.**emplace\_back**(**i**);**

**for** **(**long long j **=** i **\*** i**;** j **<=** lim**;** j **+=** i**)**

mark**[**j**]** **=** **true;**

**}**

**}**

vector**<**char**>** isPrime**(**R **-** L **+** 1**,** **true);**

**for** **(**long long i **:** primes**)**

**for** **(**long long j **=** max**(**i **\*** i**,** **(**L **+** i **-** 1**)** **/** i **\*** i**);** j **<=** R**;** j **+=** i**)**

isPrime**[**j **-** L**]** **=** **false;**

**if** **(**L **==** 1**)**

isPrime**[**0**]** **=** **false;**

// isPrime[i - L]

**return** isPrime**;**

**}**

// shorted , maybe faster

vector**<**char**>** segmentedSieveNoPreGen**(**long long L**,** long long R**)** **{**

vector**<**char**>** isPrime**(**R **-** L **+** 1**,** **true);**

long long lim **=** sqrt**(**R**);**

**for** **(**long long i **=** 2**;** i **<=** lim**;** **++**i**)**

**for** **(**long long j **=** max**(**i **\*** i**,** **(**L **+** i **-** 1**)** **/** i **\*** i**);** j **<=** R**;** j **+=** i**)**

isPrime**[**j **-** L**]** **=** **false;**

**if** **(**L **==** 1**)**

isPrime**[**0**]** **=** **false;**

**return** isPrime**;**

**}**

Counting the number prime < n

// counting number prime < n

//O(nlog(n))

//https://www.spoj.com/problems/VECTAR8/

int count\_primes**(**int n**)** **{**

const int S **=** 10000**;**

vector**<**int**>** primes**;**

int nsqrt **=** sqrt**(**n**);**

vector**<**char**>** is\_prime**(**nsqrt **+** 2**,** **true);**

**for** **(**int i **=** 2**;** i **<=** nsqrt**;** i**++)** **{**

**if** **(**is\_prime**[**i**])** **{**

primes**.**push\_back**(**i**);**

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

is\_prime**[**j**]** **=** **false;**

**}**

**}**

int result **=** 0**;**

vector**<**char**>** block**(**S**);**

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

fill**(**block**.**begin**(),** block**.**end**(),** **true);**

int start **=** k **\*** S**;**

**for** **(**int p **:** primes**)** **{**

int start\_idx **=** **(**start **+** p **-** 1**)** **/** p**;**

int j **=** max**(**start\_idx**,** p**)** **\*** p **-** start**;**

**for** **(;** j **<** S**;** j **+=** p**)**

block**[**j**]** **=** **false;**

**}**

**if** **(**k **==** 0**)**

block**[**0**]** **=** block**[**1**]** **=** **false;**

**for** **(**int i **=** 0**;** i **<** S **&&** start **+** i **<=** n**;** i**++)** **{**

**if** **(**block**[**i**])**

result**++;**

**}**

**}**

**return** result**;**

**}**

II Number of divisor , sum of divisor

Pollard rho

//Couting Number Divisor and print divisor (N <= 1e18)

https**:**//codeforces.com/gym/100753/attachments

https**:**//codeforces.com/gym/100753/submission/136755538

#include<bits/stdc++.h>

**using** **namespace** std**;**

function**<**void**(**void**)>** \_\_\_\_ **=** **[](){**ios\_base**::**sync\_with\_stdio**(false);** cin**.**tie**(**0**);** cout**.**tie**(**0**);};**

#define S 3

#define C 2730

**using** ll **=** int\_fast64\_t**;**

ll N**;**

map**<**ll**,**int**>** msk**;**

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

inline ll mul**(**ll a**,**ll b**,**ll n**){return** **(**a**\***b**-(**ll**)(**a**/(**long double**)**n**\***b**+**1e-3**)\***n**+**n**)%**n**;}**

inline ll pow**(**ll a**,**ll b**,** ll n**){**

ll d**=**1**;**

a**%=**n**;**

**while(**b**){**

**if(**b**&**1**)**d**=**mul**(**d**,**a**,**n**);**

a**=**mul**(**a**,**a**,**n**);**

b**>>=**1**;**

**}**

**return** d**;**

**}**

inline bool check**(**ll a**,**ll n**){**

ll m**=**n**-**1**,**x**,**y**;** int i**,**j**=**0**;**

**while(!(**m**&**1**))**m**>>=**1**,**j**++;**

x**=**pow**(**a**,**m**,**n**);**

**for(**i**=**1**;**i**<=**j**;**x**=**y**,**i**++){**

y**=**pow**(**x**,**2**,**n**);**

**if((**y**==**1**)&&(**x**!=**1**)&&(**x**!=**n**-**1**))return** 1**;**

**}**

**return** y**!=**1**;**

**}**

inline bool miller\_rabin**(**int times**,** ll n**){**

**if(**n**==**1**)return** 0**;**

**if(**n**==**2**)return** 1**;**

**if(!(**n**&**1**))return** 0**;**

**while(**times**--)if(**check**(**rand**()%(**n**-**1**)+**1**,**n**))return** 0**;**

**return** 1**;**

**}**

inline ll pollard\_rho**(**ll n**,**int c**){**

ll i**=**1**,**k**=**2**,**x**=**rand**()%**n**,**y**=**x**,**d**;**

**while(**1**){**

i**++,**x**=(**mul**(**x**,**x**,**n**)+**c**)%**n**,**d**=**gcd**(**y**-**x**,**n**);**

**if(**d**>**1**&&**d**<**n**)return** d**;**

**if(**y**==**x**)return** n**;**

**if(**i**==**k**)**y**=**x**,**k**<<=**1**;**

**}**

**}**

ll F**=-**1**;**

inline void findfac**(**ll n**,**int c**){**

**if(**n**==**1**)return** **;**

**if(**miller\_rabin**(**S**,**n**)){**

F**=**max**(**F**,**n**);**

**return** **;**

**}**

ll m**=**n**;**

**while(**m**==**n**)**m**=**pollard\_rho**(**n**,**c**--);**

findfac**(**m**,**c**),**findfac**(**n**/**m**,**c**);**

**}**

void solve**(**ll x**){**

**if(**x**<=**1e9**){**

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

**if(**x**%**i**!=**0**)** **continue;**

**while(**x**%**i**==**0**){**

msk**[**i**]++;**

x**/=**i**;**

**}**

**}**

**if(**x**!=**1**)** msk**[**x**]++;**

**return;**

**}**

F **=** **-**1**;**

findfac**(**x**,**C**);**

**if(**F**==**x**){**

msk**[**F**]++;**

**return;**

**}**

ll y **=** x **/** F**;**

x **=** F**;**

solve**(**x**);** solve**(**y**);**

**}**

int main**(){**

cin **>>** N**;**

solve**(**N**);**

//findfac(N,C);

ll ret **=** 1**;**

**for(**auto p **:** msk**)** ret **\*=** p**.**second **+** 1**,** cout **<<** p**.**first **<<** '\n'**;**

cout **<<** ret **<<** endl**;**

**return** 0**;**

**}**

Euler’s totient function

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**;**

**}**

III Modular Arithmetic

Find modular Inverse Using Extended Euclidean

//Finding the Modular Inverse using Extended Euclidean algorithm by equation ax + my = 1

//proof

//modulo m of both side

//Thus, the modular inverse of a is x.

//ax = 1 mod m --> x = a^−1 .

//note that gcd(a, m) = 1 is also the condition for the modular inverse to exist.

int moduloInverse**(**int a**,** int m**,** int x**,** int y**){**

int x**,** y**;**

int g **=** extended\_euclidean**(**a**,** m**,** x**,** y**);**

**if** **(**g **!=** 1**)** **{**

cout **<<** "No solution!"**;**

**}**

**else** **{**

x **=** **(**x **%** m **+** m**)** **%** m**;**

**return** x**;**

**}**

**}**

Find the Modular Inverse using Binary Exponentiation

//Finding the Modular Inverse using Binary Exponentiation

// a^ϕ(m) ≡ 1 mod m

//Multiply both sides of the above equations by a−1, and we get:

// a^(ϕ(m) − 1) ≡ (a^−1) modm

// note that gcd(a, m) = 1 is also the condition for the modular inverse to exist.

Find the Modular Inverse for every number modulo m

//Finding the modular inverse for every number modulo m {#mod-inv-all-num}

// require that the modulus m is prime.

vector**<**int**>** compute**(**int m**){**

vector**<**int**>** inv**(**m**,** 1**);**

inv**[**1**]** **=** 1**;**

**for(**int i **=** 2**;** i **<** m**;** **++**i**)**

inv**[**i**]** **=** m **-** **(**m**/**i**)** **\*** inv**[**m**%**i**]** **%** m**;**

**return** inv**;**

Find x, y in equation ax + by = gcd(a, b)

//Extended Euclidean Algorithm

//find x, y in equation ax + by = gcd(a, b)

int gcd**(**int a**,** int b**,** int**&** x**,** int**&** y**)** **{**

**if** **(**b **==** 0**)** **{**

x **=** 1**;**

y **=** 0**;**

**return** a**;**

**}**

int x1**,** y1**;**

int d **=** gcd**(**b**,** a **%** b**,** x1**,** y1**);**

x **=** y1**;**

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

**return** d**;**

**}**

Find x, y in equation ax + by = c

//find x, y in equation ax + by = c

// if c divisible by g = gcd(a, b) , equation has a solution, otherwise it does not have any solution

//note that this code does not consider the case a=b=0

https**:**//codeforces.com/contest/633/problem/A

https**:**//codeforces.com/contest/633/submission/121523795

int gcd**(**int a**,** int b**,** int**&** x**,** int**&** y**)** **{**

**if** **(**b **==** 0**)** **{**

x **=** 1**;**

y **=** 0**;**

**return** a**;**

**}**

int x1**,** y1**;**

int d **=** gcd**(**b**,** a **%** b**,** x1**,** y1**);**

x **=** y1**;**

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

**return** d**;**

**}**

bool find\_any\_solution**(**int a**,** int b**,** int c**,** int **&**x0**,** int **&**y0**,** int **&**g**)** **{**

g **=** gcd**(**abs**(**a**),** abs**(**b**),** x0**,** y0**);**

**if** **(**c **%** g**)** **{**

**return** **false;**

**}**

x0 **\*=** c **/** g**;**

y0 **\*=** c **/** g**;**

**if** **(**a **<** 0**)** x0 **=** **-**x0**;**

**if** **(**b **<** 0**)** y0 **=** **-**y0**;**

**return** **true;**

**}**

//x = xo + k \* b/g

//y = yo - k \* a/g

//-- find min(x + y)

//x + y = x + y + k(b - a)/g

//if a < b we need to select smallest possible value of k

//else we need to select the largest possible value of k

Fractorial Modular p

//Factorial modulo p

int factmod**(**int n**,** int p**)** **{**

vector**<**int**>** f**(**p**);**

f**[**0**]** **=** 1**;**

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

f**[**i**]** **=** f**[**i**-**1**]** **\*** i **%** p**;**

int res **=** 1**;**

**while** **(**n **>** 1**)** **{**

**if** **((**n**/**p**)** **%** 2**)**

res **=** p **-** res**;**

res **=** res **\*** f**[**n**%**p**]** **%** p**;**

n **/=** p**;**

**}**

**return** res**;**

**}**

IV Combinatorics

Conbinatorics important

//Combinatorics

//fomular important

//Pascal's Triangle

const int maxn **=** **...;**

int C**[**maxn **+** 1**][**maxn **+** 1**];**

C**[**0**][**0**]** **=** 1**;**

**for** **(**int n **=** 1**;** n **<=** maxn**;** **++**n**)** **{**

C**[**n**][**0**]** **=** C**[**n**][**n**]** **=** 1**;**

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

C**[**n**][**k**]** **=** C**[**n **-** 1**][**k **-** 1**]** **+** C**[**n **-** 1**][**k**];**

**}**

//Binomial coefficient modulo large prime

factorial**[**0**]** **=** 1**;**

**for** **(**int i **=** 1**;** i **<=** MAXN**;** i**++)** **{**

factorial**[**i**]** **=** factorial**[**i **-** 1**]** **\*** i **%** m**;**

**}**

long long binomial\_coefficient**(**int n**,** int k**)** **{**

**return** factorial**[**n**]** **\*** inverse**(**factorial**[**k**]** **\*** factorial**[**n **-** k**]** **%** m**)** **%** m **\*** binpow**(**factorial**[**k**]** **\*** factorial**[**n **-** k**]** **%** m**,** m **-** 2**);**

**}**

**(finbonaci)**

Stars and bars

Number of non-negative integer sum

You want count the number of solution of equation

x1 + x2 + x3 + ... + xk = n;

with xi >= 0;

🡪 solution is

---------------------------------------------------------------------------------------------------------

Number of lower-bound integer sums

You want count the number of solution of equation

X1 + x2 + x3 + … + xk = n;

With xi >= ai

⬄

Y1 + y2 + … + yk = n – a1 – a2 – a3 - … - ak

With yi >= 0

The Inclusion – Exclusion Principle

The Inclusion**-**Exclusion Principle

//e number of relative primes in a given interval

S**(**A∪B∪C**)=**S**(**A**)+**S**(**B**)+**S**(**C**)**−S**(**A∩B**)**−S**(**A∩C**)**−S**(**B∩C**)+**S**(**A∩B∩C**)**

//task :given two numbers n and r, count the number of integers in the interval [1;r] that are relatively prime to n (their greatest common divisor is 1).

int solve **(**int n**,** int r**)** **{**

vector**<**int**>** p**;**

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

**if** **(**n **%** i **==** 0**)** **{**

p**.**push\_back **(**i**);**

**while** **(**n **%** i **==** 0**)**

n **/=** i**;**

**}**

**if** **(**n **>** 1**)**

p**.**push\_back **(**n**);**

int sum **=** 0**;**

**for** **(**int msk**=**1**;** msk**<(**1**<<**p**.**size**());** **++**msk**)** **{**

int mult **=** 1**,**

bits **=** 0**;**

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

**if** **(**msk **&** **(**1**<<**i**))** **{**

**++**bits**;**

mult **\*=** p**[**i**];**

**}**

int cur **=** r **/** mult**;**

**if** **(**bits **%** 2 **==** 1**)**

sum **+=** cur**;**

**else**

sum **-=** cur**;**

**}**

**return** r **-** sum**;**

**}**

//Practice

//problem OLP035 Xoa phan tu

//https://code.ptit.edu.vn/student/question/OLP035

//https://www.spoj.com/problems/KPRIMESB/

// -------------Stars and Bar -------------------

DYNAMIC PROGRAMMING

Quy hoạch động trạng thái

int onBit**(**int x**,** int i**){**

**return** **(**x**|(**1**<<**i**));**

**}**

int offBit**(**int x**,** int i**){**

**return** **(**x**^(**1**<<**i**));**

**}**

int getBit**(**int x**,** int i**){**

**return** **((**x**>>**i**)&**1**);**

**}**

/\*

- Input:

+ n: Kích thước ma trận

+ a: matrix số

3

1 2 3

3 2 1

2 2 2

- output:

+ Tổng hoán vị lớn nhất của mỗi hàng

8

\*/

int dp**[**1300000**];**

int n**;**

int a**[**20**][**20**];**

void solve**(){**

int last **=** **(**1**<<**n**)-**1**;**

**for** **(**int state**=**1**;** state**<=**last **;**state**++){**

int cnt**=-**1**;**

**for** **(**int i**=**0**;**i**<**n**;**i**++)** **if** **((**state**>>**i**)&**1**){**

cnt**++;**

**}**

**for** **(**int i**=**0**;**i**<**n**;**i**++)** **if** **((**state**>>**i**)&**1**){**

int mask**=** state **^** **(**1**<<**i**);**

dp**[**state**]** **=** max**(**dp**[**mask**]+**a**[**cnt**][**i**],**dp**[**state**]);**

**}**

**}**

cout**<<**dp**[**last**];**

**}**

MAXTRIX

Nhân ma trận

#define ll long long

#define MOD 1000000000000007

#define vi64 vector<long long>

#define vii64 vector<vector<long long> >

void printMatrix**(**vii64 m**){**

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

**for** **(**int j **=**0**;**j**<**m**[**0**].**size**();**j**++){**

cout**<<**m**[**i**][**j**]<<**" "**;**

**}**

cout**<<**endl**;**

**}**

**}**

vii64 get\_matrix\_unit**(**int s**){**

vii64 matrix\_unit**=** vii64**(**s**,** vi64**(**s**,**0**));**

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

matrix\_unit**[**i**][**i**]=**1**;**

**}**

**return** matrix\_unit**;**

**}**// matrix unit

ll mul\_int64**(**ll a**,** ll b**){**

**if** **(**b**==**0**)** **return** 0**;**

**if** **(**b**==**1**)** **return** a**;**

ll p **=** mul\_int64**(**a**,**b**/**2**);**

p**=(**p**+**p**)%**MOD**;**

**if** **(**b**%**2**==**1**)** p**=(**p**+**a**)%**MOD**;**

**return** p**;**

**}**//mul mix MOD

vii64 multip\_matrix**(**vii64 a**,** vii64 b**){**

vii64 ans **=** vii64**(**a**.**size**(),** vi64**(**b**[**0**].**size**(),**0**));**

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

**for** **(**int j**=**0**;**j**<**ans**[**0**].**size**();**j**++){**

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

ans**[**i**][**j**]=(**ans**[**i**][**j**]+**mul\_int64**(**a**[**i**][**k**],**b**[**k**][**j**])%**MOD**)%**MOD**;**

**}**

**}**

**}**

**return** ans**;**

**}**// X= A\*B

vii64 plus\_matrix**(**vii64 a**,** vii64 b**){**

vii64 ans **=** vii64**(**a**.**size**(),**vi64**(**b**.**size**(),**0**));**

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

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

ans**[**i**][**j**]=** **(**a**[**i**][**j**]+**b**[**i**][**j**])%**MOD**;**

**}**

**}**

**return** ans**;**

**}**//X= A+B

vii64 powMatrix**(**vii64 matrix**,** ll hat**){**

**if** **(**hat **==** 0**)** **return** get\_matrix\_unit**(**matrix**.**size**());**

**if** **(**hat **==** 1**)** **return** matrix**;**

vii64 p **=** powMatrix**(**matrix**,** hat**/**2**);**

p **=** multip\_matrix**(**p**,**p**);**

**if** **(**hat**%**2**==**1**)** **return** multip\_matrix**(**p**,**matrix**);**

**return** p**;**

**}**//X = A^k

vii64 sumPowMatrix**(**vii64 matrix**,** ll k**){**

**if** **(**k **==** 0**)** **return** get\_matrix\_unit**(**matrix**.**size**());**

**if** **(**k **==** 1**)** **return** matrix**;**

vii64 p **=** multip\_matrix**(**

plus\_matrix**(**get\_matrix\_unit**(**matrix**.**size**()),**powMatrix**(**matrix**,**k**/**2**)),**

sumPowMatrix**(**matrix**,** k**/**2**)**

**);**

**if** **(**k**%**2**==**1**)** **return** plus\_matrix**(**p**,** powMatrix**(**matrix**,**k**));**

**return** p**;**

**}**

//B(k) = A + A^2+ ...+A^k

ll sumTribonaci**(**ll n**){**

vii64 weight **=** **{{**1**,**1**,**1**},{**1**,**0**,**0**},{**0**,**1**,**0**}};**

weight **=** sumPowMatrix**(**weight**,**n**);**

**return** weight**[**0**][**1**];**

**}**

//F(N) = T(1) +T(2)+...+ T(N)

GRAPH

disjoint-set

vector**<**int**>** parent**(**N**);**

vector**<**int**>** size**(**N**,** 1**);**

void make\_set**(**int v**)** **{**

parent**[**v**]** **=** v**;**

**}**

int find\_set**(**int v**)** **{**

**if** **(**v **==** parent**[**v**])**

**return** v**;**

**return** parent**[**v**]** **=** find\_set**(**parent**[**v**]);**

**}**

void union\_sets**(**int a**,** int b**)** **{**

a **=** find\_set**(**a**);**

b **=** find\_set**(**b**);**

**if** **(**a **!=** b**)** **{**

**if** **(**size**[**a**]** **<** size**[**b**])**

swap**(**a**,** b**);**

parent**[**b**]** **=** a**;**

size**[**a**]** **+=** size**[**b**];**

**}**

**}**

Dijkstra

void dijkstra**(**int s**,** vector**<**int**>** **&** d**,** vector**<**int**>** **&** p**)** **{**

int n **=** adj**.**size**();**

d**.**assign**(**n**,** INF**);**

p**.**assign**(**n**,** **-**1**);**

d**[**s**]** **=** 0**;**

**using** pii **=** pair**<**int**,** int**>;**

priority\_queue**<**pii**,** vector**<**pii**>,** greater**<**pii**>>** q**;**

q**.**push**({**0**,** s**});**

**while** **(!**q**.**empty**())** **{**

int v **=** q**.**top**().**second**;**

int d\_v **=** q**.**top**().**first**;**

q**.**pop**();**

**if** **(**d\_v **!=** d**[**v**])**

**continue;**

**for** **(**auto edge **:** adj**[**v**])** **{**

int to **=** edge**.**first**;**

int len **=** edge**.**second**;**

**if** **(**d**[**v**]** **+** len **<** d**[**to**])** **{**

d**[**to**]** **=** d**[**v**]** **+** len**;**

p**[**to**]** **=** v**;**

q**.**push**({**d**[**to**],** to**});**

**}**

**}**

**}**

**}**

Bellman Ford (Shortest Path Faster Algorithm improvement)

const int INF **=** 1000000000**;**

vector**<**vector**<**pair**<**int**,** int**>>>** adj**;**

bool spfa**(**int s**,** vector**<**int**>&** d**)** **{**

int n **=** adj**.**size**();**

d**.**assign**(**n**,** INF**);**

vector**<**int**>** cnt**(**n**,** 0**);**

vector**<**bool**>** inqueue**(**n**,** **false);**

queue**<**int**>** q**;**

d**[**s**]** **=** 0**;**

q**.**push**(**s**);**

inqueue**[**s**]** **=** **true;**

**while** **(!**q**.**empty**())** **{**

int v **=** q**.**front**();**

q**.**pop**();**

inqueue**[**v**]** **=** **false;**

**for** **(**auto edge **:** adj**[**v**])** **{**

int to **=** edge**.**first**;**

int len **=** edge**.**second**;**

**if** **(**d**[**v**]** **+** len **<** d**[**to**])** **{**

d**[**to**]** **=** d**[**v**]** **+** len**;**

**if** **(!**inqueue**[**to**])** **{**

q**.**push**(**to**);**

inqueue**[**to**]** **=** **true;**

cnt**[**to**]++;**

**if** **(**cnt**[**to**]** **>** n**)**

**return** **false;** // negative cycle

**}**

**}**

**}**

**}**

**return** **true;**

**}**

Floyd Warshall

const int INF **=** 1000000000**;**

vector**<**vector**<**int**>** **>** d**;**

bool FloydWarshall**(**int V**){**

**for(**int k **=** 0**;** k **<** V**;** k**++){**

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

**for(**int j **=** 0**;** j **<** V**;** j**++){**

**if(**d**[**i**][**k**]** **<** INF **&&** d**[**k**][**j**]** **<** INF**){**

d**[**i**][**j**]** **=** min**(**d**[**i**][**j**],** d**[**i**][**k**]** **+** d**[**k**][**j**]);**

**}**

**}**

**if** **(**d**[**i**][**i**]** **<** 0**)** **return** **false;** // negative cycle

**}**

**}**

**return** **true;**

**}**

LCA

#include <bits/stdc++.h>

**using** **namespace** std**;**

const int MAX\_N **=** 100005**;**

const int LOG **=** 17**;**

vector**<**int**>** children**[**MAX\_N**];**

int up**[**MAX\_N**][**LOG**];**

int depth**[**MAX\_N**];**

void dfs**(**int a**,** int parent**)** **{**

**for(**int b **:** children**[**a**])** **{**

**if(**b **==** parent**)** **continue;**

depth**[**b**]** **=** depth**[**a**]** **+** 1**;**

up**[**b**][**0**]** **=** a**;**

**for(**int j **=** 1**;** j **<** LOG**;** j**++)** **{**

up**[**b**][**j**]** **=** up**[**up**[**b**][**j**-**1**]][**j**-**1**];**

**}**

dfs**(**b**,** a**);**

**}**

**}**

int get\_lca**(**int a**,** int b**)** **{**

**if(**depth**[**a**]** **<** depth**[**b**])** **{**

swap**(**a**,** b**);**

**}**

int k **=** depth**[**a**]** **-** depth**[**b**];**

**for(**int j **=** LOG **-** 1**;** j **>=** 0**;** j**--)** **{**

**if(**k **&** **(**1 **<<** j**))** **{**

a **=** up**[**a**][**j**];**

**}**

**}**

**if(**a **==** b**)** **{**

**return** a**;**

**}**

**for(**int j **=** LOG **-** 1**;** j **>=** 0**;** j**--)** **{**

**if(**up**[**a**][**j**]** **!=** up**[**b**][**j**])** **{**

a **=** up**[**a**][**j**];**

b **=** up**[**b**][**j**];**

**}**

**}**

**return** up**[**a**][**0**];**

**}**

KRUSKA

#include<bits/stdc++.h>

**using** **namespace** std**;**

int const N **=** 1e5 **+** 123**;**

vector**<**int**>** parent**(**N**);**

#define int long long

void make\_set**(**int v**)** **{**

parent**[**v**]** **=** v**;**

**}**

int find\_set**(**int v**)** **{**

**if** **(**v **==** parent**[**v**])**

**return** v**;**

**return** parent**[**v**]** **=** find\_set**(**parent**[**v**]);**

**}**

void union\_sets**(**int a**,** int b**)** **{**

a **=** find\_set**(**a**);**

b **=** find\_set**(**b**);**

**if** **(**a **!=** b**)**

parent**[**b**]** **=** a**;**

**}**

struct item**{**

int x**,** y**,** v**;**

**};**

bool cmp**(**item a**,** item b**){**

**return** a**.**v **<** b**.**v**;**

**}**

void TestCase**(){**

int n**,** m**;** cin **>>** n **>>** m**;**

vector**<**item**>** v**;**

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

int x**,** y**,** val**;** cin **>>** x **>>** y **>>** val**;**

v**.**push\_back**({**x**,** y**,** val**});**

**}**

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

make\_set**(**i**);**

**}**

sort**(**v**.**begin**(),** v**.**end**(),** cmp**);**

int answer **=** 0**;**

**for(**item d **:** v**){**

int p **=** find\_set**(**d**.**x**);**

int q **=** find\_set**(**d**.**y**);**

**if(**p **!=** q**){**

answer **+=** d**.**v**;**

union\_sets**(**d**.**x**,** d**.**y**);**

**}**

**}**

cout **<<** answer **<<** '\n'**;**

**}**

Khớp và cầu

#include <bits/stdc++.h>

**using** **namespace** std**;**

const int maxN **=** 10010**;**

int n**,** m**;**

bool joint**[**maxN**];**

int timeDfs **=** 0**,** bridge **=** 0**;**

int low**[**maxN**],** num**[**maxN**];**

vector **<**int**>** g**[**maxN**];**

void dfs**(**int u**,** int pre**)** **{**

int child **=** 0**;**

num**[**u**]** **=** low**[**u**]** **=** **++**timeDfs**;**

**for** **(**int v **:** g**[**u**])** **{**

**if** **(**v **==** pre**)** **continue;**

**if** **(!**num**[**v**])** **{**

dfs**(**v**,** u**);**

low**[**u**]** **=** min**(**low**[**u**],** low**[**v**]);**

**if** **(**low**[**v**]** **==** num**[**v**])** bridge**++;**

child**++;**

**if** **(**u **==** pre**)** **{**

**if** **(**child **>** 1**)** joint**[**u**]** **=** **true;**

**}**

**else** **if** **(**low**[**v**]** **>=** num**[**u**])** joint**[**u**]** **=** **true;**

**}**

**else** low**[**u**]** **=** min**(**low**[**u**],** num**[**v**]);**

**}**

**}**

int main**()** **{**

cin **>>** n **>>** m**;**

**for** **(**int i **=** 1**;** i **<=** m**;** i**++)** **{**

int u**,** v**;**

cin **>>** u **>>** v**;**

g**[**u**].**push\_back**(**v**);**

g**[**v**].**push\_back**(**u**);**

**}**

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

**if** **(!**num**[**i**])** dfs**(**i**,** i**);**

int cntJoint **=** 0**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** cntJoint **+=** joint**[**i**];**

cout **<<** cntJoint **<<** ' ' **<<** bridge**;**

**}**

Thành phần liên thông mạnh

#include <bits/stdc++.h>

**using** **namespace** std**;**

const int maxN **=** 100010**;**

int n**,** m**;**

int timeDfs **=** 0**,** scc **=** 0**;**

int low**[**maxN**],** num**[**maxN**];**

bool deleted**[**maxN**];**

vector **<**int**>** g**[**maxN**];**

stack **<**int**>** st**;**

void dfs**(**int u**)** **{**

num**[**u**]** **=** low**[**u**]** **=** **++**timeDfs**;**

st**.**push**(**u**);**

**for** **(**int v **:** g**[**u**])** **{**

**if** **(**deleted**[**v**])** **continue;**

**if** **(!**num**[**v**]){**

dfs**(**v**);**

low**[**u**]** **=** min**(**low**[**u**],** low**[**v**]);**

**}**

**else** low**[**u**]** **=** min**(**low**[**u**],** num**[**v**]);**

**}**

**if** **(**low**[**u**]** **==** num**[**u**])** **{**

scc**++;**

int v**;**

**do** **{**

v **=** st**.**top**();**

st**.**pop**();**

deleted**[**v**]** **=** **true;**

**}**

**while** **(**v **!=** u**);**

**}**

**}**

int main**()** **{**

cin **>>** n **>>** m**;**

**for** **(**int i **=** 1**;** i **<=** m**;** i**++)** **{**

int u**,** v**;**

cin **>>** u **>>** v**;**

g**[**u**].**push\_back**(**v**);**

**}**

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

**if** **(!**num**[**i**])** dfs**(**i**);**

cout **<<** scc**;**

**}**

BIG – INTEGER

class BigInt **{**

public**:**

int sign**;** string s**;**

BigInt**()** **:** s**(**""**)** **{}**

BigInt**(**string x**)** **{** **\*this** **=** x**;** **}**

BigInt**(**int x**)** **{** **\*this** **=** to\_string**(**x**);** **}**

BigInt negative**()** **{**

BigInt x **=** **\*this;** x**.**sign **\*=** **-**1**;** **return** x**;**

**}**

BigInt normalize**(**int newSign**)** **{**

**for** **(**int a **=** s**.**size**()** **-** 1**;** a **>** 0 **&&** s**[**a**]** **==** '0'**;** a**--)**

s**.**erase**(**s**.**begin**()** **+** a**);**

sign **=** **(**s**.**size**()** **==** 1 **&&** s**[**0**]** **==** '0' **?** 1 **:** newSign**);**

**return** **\*this;**

**}**

void **operator=(**string x**){**

int newSign **=** **(**x**[**0**]** **==** '-' **?** **-**1 **:** 1**);**

s **=** **(**newSign **==** **-**1 **?** x**.**substr**(**1**)** **:** x**);**

reverse**(**s**.**begin**(),** s**.**end**());**

**this->**normalize**(**newSign**);**

**}**

bool **operator==(**const BigInt**&** x**)** const **{**

**return** **(**s **==** x**.**s **&&** sign **==** x**.**sign**);**

**}**

bool **operator<(**const BigInt**&** x**)** const **{**

**if** **(**sign **!=** x**.**sign**)** **return** sign **<** x**.**sign**;**

**if** **(**s**.**size**()** **!=** x**.**s**.**size**())**

**return** **(**sign **==** 1 **?** s**.**size**()** **<** x**.**s**.**size**()** **:** s**.**size**()** **>** x**.**s**.**size**());**

**for** **(**int a **=** s**.**size**()** **-** 1**;** a **>=** 0**;** a**--)**

**if** **(**s**[**a**]** **!=** x**.**s**[**a**])**

**return** **(**sign **==** 1 **?** s**[**a**]** **<** x**.**s**[**a**]** **:** s**[**a**]** **>** x**.**s**[**a**]);**

**return** **false;**

**}**

bool **operator<=(**const BigInt**&** x**)** const **{**

**return** **(\*this** **<** x **||** **\*this** **==** x**);**

**}**

bool **operator>(**const BigInt**&** x**)** const **{**

**return** **(!(\*this** **<** x**)** **&&** **!(\*this** **==** x**));**

**}**

bool **operator>=(**const BigInt**&** x**)** const **{**

**return** **(\*this** **>** x **||** **\*this** **==** x**);**

**}**

BigInt **operator+(**BigInt x**)** **{**

BigInt curr **=** **\*this,** res**;**

**if** **(**curr**.**sign **!=** x**.**sign**)** **return** curr **-** x**.**negative**();**

**for** **(**int a **=** 0**,** carry **=** 0**;** a **<** s**.**size**()** **||** a **<** x**.**s**.**size**()** **||** carry**;** a**++)** **{**

carry **+=** **(**a **<** curr**.**s**.**size**()** **?** curr**.**s**[**a**]** **-** '0' **:** 0**)** **+** **(**a **<** x**.**s**.**size**()** **?** x**.**s**[**a**]** **-** '0' **:** 0**);**

res**.**s **+=** **(**carry **%** 10 **+** '0'**);**

carry **/=** 10**;**

**}**

**return** res**.**normalize**(**sign**);**

**}**

BigInt **operator-(**BigInt x**)** **{**

BigInt curr **=** **\*this,** res**;**

**if** **(**curr**.**sign **!=** x**.**sign**)**

**return** curr **+** x**.**negative**();**

int realSign **=** curr**.**sign**;** curr**.**sign **=** x**.**sign **=** 1**;**

**if** **(**curr **<** x**)** **return** **((**x **-** curr**).**negative**()).**normalize**(-**realSign**);**

**for** **(**int a **=** 0**,** borrow **=** 0**;** a **<** s**.**size**();** a**++)** **{**

borrow **=** **(**curr**.**s**[**a**]** **-** borrow **-** **(**a **<** x**.**s**.**size**()** **?** x**.**s**[**a**]** **:** '0'**));**

res**.**s **+=** **(**borrow **>=** 0 **?** borrow **+** '0' **:** borrow **+** '0' **+** 10**);**

borrow **=** **(**borrow **>=** 0 **?** 0 **:** 1**);**

**}**

**return** res**.**normalize**(**realSign**);**

**}**

BigInt **operator\*(**BigInt x**)** **{**

BigInt res**(**"0"**);**

**for** **(**int a **=** 0**,** b **=** s**[**a**]** **-** '0'**;** a **<** s**.**size**();** a**++,** b **=** s**[**a**]** **-** '0'**)** **{**

**while** **(**b**--)** res **=** **(**res **+** x**);**

x**.**s**.**insert**(**x**.**s**.**begin**(),** '0'**);**

**}**

**return** res**.**normalize**(**sign **\*** x**.**sign**);**

**}**

BigInt **operator/(**BigInt x**)** **{**

**if** **(**x**.**s**.**size**()** **==** 1 **&&** x**.**s**[**0**]** **==** '0'**)** x**.**s**[**0**]** **/=** **(**x**.**s**[**0**]** **-** '0'**);**

BigInt temp**(**"0"**),** res**;**

**for** **(**int a **=** 0**;** a **<** s**.**size**();** a**++)** res**.**s **+=** "0"**;**

int newSign **=** sign **\*** x**.**sign**;** x**.**sign **=** 1**;**

**for** **(**int a **=** s**.**size**()** **-** 1**;** a **>=** 0**;** a**--)** **{**

temp**.**s**.**insert**(**temp**.**s**.**begin**(),** '0'**);**

temp **=** temp **+** s**.**substr**(**a**,** 1**);**

**while** **(!(**temp **<** x**))** **{**

temp **=** temp **-** x**;**

res**.**s**[**a**]++;**

**}**

**}**

**return** res**.**normalize**(**newSign**);**

**}**

BigInt **operator%(**BigInt x**)** **{**

**if** **(**x**.**s**.**size**()** **==** 1 **&&** x**.**s**[**0**]** **==** '0'**)** x**.**s**[**0**]** **/=** **(**x**.**s**[**0**]** **-** '0'**);**

BigInt res**(**"0"**);** x**.**sign **=** 1**;**

**for** **(**int a **=** s**.**size**()** **-** 1**;** a **>=** 0**;** a**--)** **{**

res**.**s**.**insert**(**res**.**s**.**begin**(),** '0'**);**

res **=** res **+** s**.**substr**(**a**,** 1**);**

**while** **(!(**res **<** x**))** res **=** res **-** x**;**

**}**

**return** res**.**normalize**(**sign**);**

**}**

string toString**()** const **{**

string ret **=** s**;**

reverse**(**ret**.**begin**(),** ret**.**end**());**

**return** **(**sign **==** **-**1 **?** "-" **:** ""**)** **+** ret**;**

**}**

**};**

DATA STRUCTURES

ST + Lazy

//ST + Lazy

//Find MAX in range(l, r)

const int N **=** 5e4 **+** 11**;**

struct**{**

int f**[**N **\*** 4**],** lazy**[**N **\*** 4**];**

void init**(**int k**,** int l**,** int r**){**

f**[**k**]** **=** 0**;**

**if(**l **<** r**){**

int mid **=** **(**l **+** r**)** **>>** 1**;**

init**(**k **<<** 1**,** l**,** mid**);**

init**(**k **<<** 1 **|** 1**,** mid **+** 1**,** r**);**

**}**

**}**

void down**(**int k**,** int l**,** int r**){**

f**[**k**]** **+=** lazy**[**k**];**

**if(**l **<** r**){**

lazy**[**k **<<** 1**]** **+=** lazy**[**k**];**

lazy**[**k **<<** 1 **|** 1**]** **+=** lazy**[**k**];**

**}**

lazy**[**k**]** **=** 0**;**

**}**

void update**(**int k**,** int l**,** int r**,** int L**,** int R**,** int x**){**

**if(**l **>** r**)** **return;**

**if(**lazy**[**k**]** **!=** 0**)** down**(**k**,** l**,** r**);**

**if(**l **>** R **||** L **>** r**)** **return;**

**if(**L **<=** l **&&** r **<=** R**){**

f**[**k**]** **+=** x**;**

**if(**l **<** r**){**

lazy**[**k **<<** 1**]** **+=** x **;**

lazy**[**k **<<** 1 **|** 1**]** **+=** x **;**

**}**

**return;**

**}**

int mid **=** **(**l **+** r**)** **>>** 1**;**

update**(**k **<<** 1**,** l**,** mid**,** L**,** R**,** x**);**

update**(**k **<<** 1 **|** 1**,** mid **+** 1**,** r**,** L**,** R**,** x**);**

f**[**k**]** **=** max**(**f**[**k **<<** 1**],** f**[**k **<<** 1 **|** 1**]);**

**}**

int query**(**int k**,** int l**,** int r**,** int L**,** int R**){**

**if(**l **>** r**)** **return** 0**;**

**if(**lazy**[**k**]** **!=** 0**)** down**(**k**,** l**,** r**);**

**if(**l **>** R **||** L **>** r**)** **return** 0**;**

**if(**L **<=** l **&&** r **<=** R**)** **return** f**[**k**];**

int mid **=** **(**l **+** r**)** **>>** 1**;**

**return** max**(**query**(**k **<<** 1**,** l**,** mid**,** L**,** R**),** query**(**k **<<** 1 **|** 1**,** mid **+** 1**,** r**,** L**,** R**));**

**}**

**}**ST**;**

BIT

int getSum**(**int BITree**[],** int index**)** **{**

int sum **=** 0**;**

index **=** index **+** 1**;**

**while** **(**index **>** 0**)** **{**

sum **+=** BITree**[**index**];**

index **-=** index **&** **(-**index**);**

**}**

**return** sum**;**

**}**

void updateBIT**(**int BITree**[],** int n**,** int index**,** int val**)** **{**

index **=** index **+** 1**;**

**while** **(**index **<=** n**)** **{**

BITree**[**index**]** **+=** val**;**

index **+=** index **&** **(-**index**);**

**}**

**}**

int **\***constructBITree**(**int arr**[],** int n**)** **{**

int **\***BITree **=** **new** int**[**n **+** 1**];**

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

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** updateBIT**(**BITree**,** n**,** i**,** arr**[**i**]);**

**return** BITree**;**

**}**

Stack

// tim phan tu xa nhat ve phia ben trai ma lon hon x

vector**<**int**>** processLeft**(**vector**<**int**>** **&**a**){**

stack**<**int**>** st**;**

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

vector**<**int**>** L**(**n **+** 5**);**

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

**while** **(!**st**.**empty**()** **&&** a**[**st**.**top**()]** **>=** a**[**i**]){**

st**.**pop**();**

**}**

**if** **(**st**.**empty**()){**

L**[**i**]** **=** 0**;**

**}else{**

L**[**i**]** **=** st**.**top**()** **+** 1**;**

**}**

st**.**push**(**i**);**

**}**

**return** L**;**

**}**

// tim phan tu xa nhat ve phia ben phai lon hon x

vector**<**int**>** processRight**(**vector**<**int**>** **&**a**){**

stack**<**int**>** st**;**

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

vector**<**int**>** R**(**n **+** 5**);**

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

**while** **(!**st**.**empty**()** **&&** a**[**st**.**top**()]** **>=** a**[**i**]){**

st**.**pop**();**

**}**

**if** **(**st**.**empty**()){**

R**[**i**]** **=** n**-**1**;**

**}else{**

R**[**i**]** **=** st**.**top**()** **-** 1**;**

**}**

st**.**push**(**i**);**

**}**

**return** R**;**

**}**

//Biến đổi trung tố hậu tố

int degree**(**char x**){**

**if** **(**x**==**'^'**)** **return** 500**;**

**if** **(**x**==**'\*'**||**x**==**'/'**)** **return** 300**;**

**if** **(**x**==**'+'**||**x**==**'-'**)** **return** 200**;**

**return** 100**;**

**}**

void BienDoiTrungToHauTo**(){**

stack**<**char**>** st**;**

string res**=**""**;**

s**=**"("**+**s**+**")"**;**

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

**if** **(**s**[**i**]==**')'**){**

**while(**st**.**top**()!=**'('**){**

res**+=**st**.**top**();**

st**.**pop**();**

**}**

st**.**pop**();**

**}else** **if** **(**s**[**i**]==**'('**){**

st**.**push**(**s**[**i**]);**

**}else** **if** **(**s**[**i**]==**'+'**||**s**[**i**]==**'-'**||**s**[**i**]==**'\*'**||**s**[**i**]==**'/'**||**s**[**i**]==**'^'**){**

**while** **(!**st**.**empty**()&&**degree**(**s**[**i**])<=**degree**(**st**.**top**())){**

res**+=**st**.**top**();**

st**.**pop**();**

**}**

st**.**push**(**s**[**i**]);**

**}else{**

res**+=**s**[**i**];**

**}**

**}**

cout**<<**res**;**

**}**

Trie

#include<bits/stdc++.h>

**using** **namespace** std**;**

#define MAX 26

struct Node**{**

struct Node **\***child**[**MAX**];**

int countWord**;**

**};**

// add a node to the tries

struct Node **\***newNode**(){**

struct Node **\***node **=** **new** Node**;**

node**->**countWord **=** 0**;**

**for(**int i **=** 0**;** i **<** MAX**;** i**++){**

node**->**child**[**i**]** **=** **NULL;**

**}**

**return** node**;**

**}**

void addWord**(**struct Node **\***root**,** string s**){**

int ch**;**

struct Node **\***temp **=** root**;**

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

ch **=** s**[**i**]** **-** 'a'**;**

**if(**temp**->**child**[**ch**]** **==** **NULL){**

temp**->**child**[**ch**]** **=** newNode**();**

**}**

temp **=** temp**->**child**[**ch**];**

**}**

temp**->**countWord**++;**

**}**

bool findWord**(**Node **\***root**,** string s**){**

int ch**;**

struct Node **\***temp **=** root**;**

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

ch **=** s**[**i**]** **-** 'a'**;**

**if(**temp**->**child**[**ch**]** **==** **NULL){**

**return** **false;**

**}**

temp **=** temp**->**child**[**ch**];**

**}**

**return** temp**->**countWord **>** 0**;**

**}**

bool isWord**(**struct Node **\***node**){**

**return** **(**node **->** countWord **!=** 0**);**

**}**

bool isEmpty**(**struct Node **\***node**){**

**for(**int i **=** 0**;** i **<** MAX**;** i**++){**

**if(**node**->**child**[**i**]** **!=** **NULL){**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

bool removeWord**(**struct Node **\***root**,** string s**,** int level**,** int len**){**

**if(!**root**)** **return** **false;**

**if(**level **==** len**){**

**if(**root**->**countWord **>** 0**){**

root**->**countWord**--;**

**return** **true;**

**}**

**return** **false;**

**}**

int ch **=** s**[**level**]** **-** 'a'**;**

int flag **=** removeWord**(**root **->** child**[**ch**],** s**,** level **+** 1**,** len**);**

**if(**flag **&&** **!**isWord**(**root **->**child**[**ch**])** **&&** isEmpty**(**root **->**child**[**ch**])){**

**delete** root **->** child**[**ch**];**

root**->**child**[**ch**]** **=NULL;**

**}**

**return** flag**;**

**}**

void printWord**(**struct Node **\*** root**,** string s**){**

**if(**isWord**(**root**)){**

cout **<<** s **<<** '\n'**;**

**}**

**for(**int i **=** 0**;** i **<** MAX**;** i**++){**

**if(**root**->**child**[**i**]){**

printWord**(**root**->**child**[**i**],** s **+** **(**char**)(**'a' **+** i**));**

**}**

**}**

**}**

void consider**(**string **&**answer**,** string s**){**

**if(**answer**.**size**()** **<** s**.**size**())** answer **=** s**;**

**if(**answer**.**size**()** **==** s**.**size**()){**

answer **=** min**(**answer**,** s**);**

**}**

**}**

class Solution **{**

public**:**

void dfs**(**struct Node **\***root**,** string **&**answer**,** string res**){**

**for(**int i **=** 0**;** i **<** MAX**;** i**++){**

**if(**root **->** child**[**i**]** **&&** isWord**(**root **->** child**[**i**])){**

dfs**(**root **->** child**[**i**],** answer**,** res **+** **(**char**)(**i **+** 'a'**));**

consider**(**answer**,** res **+** **(**char**)(**i **+** 'a'**));**

**}**

**}**

**}**

string longestWord**(**vector**<**string**>&** words**)** **{**

struct Node **\***root **=** newNode**();**

**for(**auto s **:** words**){**

addWord**(**root**,** s**);**

**}**

string answer **=** ""**;**

dfs**(**root**,** answer**,** ""**);**

**return** answer**;**

**}**

**};**

HashTable

void hashS**(**string **&**s**,** vector**<**int**>** **&**H**){**

int hashcode **=** 0**;**

int base **=** 26**;**

H**[**0**]** **=** s**[**0**]** **-** 'a'**;**

**for(**int i **=** 1**;** i **<** s**.**size**();** i**++){**

H**[**i**]** **=** **(**H**[**i **-** 1**]** **\*** base **+** s**[**i**]** **-** 'a'**)** **%** mod**;**

**}**

**}**

vector**<**int**>** p**(**N**,** 1**);**

unordered\_map **<**int**,** int**>** mp**;**

void init**(){**

p**[**1**]** **=** 26**;**

**for(**int i **=** 2**;** i **<=** N**;** i**++){**

p**[**i**]** **=** **(**p**[**i **-** 1**]** **%** mod **\*** 26**)** **%** mod**;**

**}**

**}**

int getHash**(**int l**,** int r**,** vector**<**int**>** **&**hash**){**

**return** **((**hash**[**r**]** **-** **(**hash**[**l **-** 1**]** **\*** p**[**r **-** l **+** 1**])** **%** mod **+** mod**)** **%** mod**);**

KMP

void KMPPreprocess**(**string**&** p**,** vector**<**int**>** **&**prefix**)** **{**

prefix**[**0**]** **=** 0**;**

int m **=** p**.**length**();**

int j **=** 0**;**

int i **=** 1**;**

**while** **(**i **<** m**)** **{**

**if** **(**p**[**i**]** **==** p**[**j**])** **{**

j**++;**

prefix**[**i**]** **=** j**;**

i**++;**

**}** **else** **{**

**if** **(**j **!=** 0**)** **{**

j **=** prefix**[**j **-** 1**];**

**}** **else** **{**

prefix**[**i**]** **=** 0**;**

i**++;**

**}**

**}**

**}**

**}**

int KMPSearch**(**string**&** t**,** string**&** p**,** vector**<**int**>** **&**prefix**)** **{**

int n **=** t**.**length**();**

int m **=** p**.**length**();**

int i **=** 0**,** j **=** 0**,** found **=** 0**;**

**while** **(**i **<** n**)** **{**

**if** **(**p**[**j**]** **==** t**[**i**])** **{**

j**++;**

i**++;**

**}**

**if** **(**j **==** m**)** **{**

found**++;**

j **=** prefix**[**j **-** 1**];**

**}** **else** **if** **(**i **<** n **&&** p**[**j**]** **!=** t**[**i**])** **{**

**if** **(**j **!=** 0**)** **{**

j **=** prefix**[**j **-** 1**];**

**}** **else** **{**

i**++;**

**}**

**}**

**}**

**return** found**;**

**}**

GEOMTRY

Template

struct P**{**

int x**,** y**;**

void read**(){**

cin **>>** x **>>** y**;**

**}**

P **operator** **-** **(**const P **&**b**)** const**{**

**return** P**{**x **-** b**.**x**,** y **-** b**.**y**};**

**}**

void **operator** **-=** **(**const P **&**b**){**

x **-=** b**.**x**;**

y **-=** b**.**y**;**

**}**

P **operator** **+** **(**const P **&**b**){**

**return** P**{**x **+** b**.**x**,** y **+** b**.**y**};**

**}**

int **operator** **\*** **(**const P **&**b**)** const**{**

**return** x **\*** b**.**y **-** y **\*** b**.**x**;**

**}**

int triagle**(**P p1**,** P p2**){**

**return** **(**p1 **-** **\*this)** **\*** **(**p2 **-** **\*this)** **/**2**;**

**}**

**};**

Kiem tra 2 doan thang cat nhau

bool intersect**(**P p1**,** P p2**,** P p3**,** P p4**){**

**if((**p2 **-** p1**)** **\*** **(**p4 **-** p3**)** **==** 0**){**

**if(**p1**.**triagle**(**p2**,** p3**)** **!=** 0**){**

**return** **false;**

**}**

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

**if(**max**(**p1**.**x**,** p2**.**x**)** **<** min**(**p3**.**x**,** p4**.**x**)** **||** max**(**p1**.**y**,** p2**.**y**)** **<** min**(**p3**.**y**,** p4**.**y**)){**

**return** **false;**

**}**

swap**(**p1**,** p3**);**

swap**(**p2**,** p4**);**

**}**

**return** **true;**

**}**

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

int x **=** **(**p2 **-** p1**)** **\*** **(**p3 **-** p1**);**

int y **=** **(**p2 **-** p1**)** **\*** **(**p4 **-** p1**);**

**if((**x **<** 0 **&&** y **<** 0**)** **||** **(**x **>** 0 **&&** y **>** 0**))** **{**

**return** **false;**

**}**

swap**(**p1**,** p3**);**

swap**(**p2**,** p4**);**

**}**

**return** **true;**

**}**

Kiểm tra 1 điểm có nằm trong đoạn hay không

// kiem tra 1 điểm có năm trong đoạn hay không

bool segment\_contains**(**P a**,** P b**,** P c**){**

**if(**a**.**triagle**(**b**,** c**))** **return** **false;**

**if(**min**(**a**.**x**,** b**.**x**)** **<=** c**.**x **&&** max**(**a**.**x**,** b**.**x**)** **>=** c**.**x **&&** min**(**a**.**y**,** b**.**y**)** **<=** c**.**y **&&** c**.**y **<=** max**(**a**.**y**,** b**.**y**))** **return** **true;**

**return** **false;**

**}**

Vị trí tương đối của điểm với đa giac (inside, outside, bound)

void solve**(){**

int n**;** cin **>>** n**;** P a**;** a**.**read**();**

vector**<**P**>** points**;**

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

P p**;** p**.**read**();**

points**.**push\_back**(**p**);**

**}**

**for(**int rep **=** 0**;** rep **<** 1**;** rep**++){**

P p **=** a**;**

P out **=** P**{**p**.**x **+** 1**,** 3000000001LL**};**

bool is\_on\_boundary **=** **false;**

int count **=** 0**;**

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

**if(**intersect**(**points**[**i**],** **(**i **+** 1 **<** n**)** **?** points**[**i **+** 1**]** **:** points**[**0**],** p**,** out**))** count **+=** 1**;**

**if(**segment\_contains**(**points**[**i**],** **(**i **+** 1 **<** n**)** **?** points**[**i **+** 1**]** **:** points**[**0**],** p**))** **{**

is\_on\_boundary **=** **true;**

**break;**

**}**

**}**

**if(**is\_on\_boundary**)** **{**

cout **<<** "0\n"**;**

**continue;**

**}**

**if(**count **%** 2 **==** 0**){**

cout **<<** "0\n"**;**

**}** **else** cout **<<** "1\n"**;**

**}**

**}**

Khoang cach ngăn nhất giữa 2 điêm

#include <stdio.h>

#include <float.h>

#include <stdlib.h>

#include <math.h>

#include <iostream>

#include <vector>

#include <algorithm>

#include <iomanip>

**using** **namespace** std**;**

#define INF 1e9

struct Point **{**

double x**,** y**;**

Point**(**double x **=** 0**,** double y **=** 0**)** **:** x**(**x**),** y**(**y**)** **{}**

**};**

bool xCompare**(**const Point **&**p1**,** const Point **&**p2**)** **{**

**return** p1**.**x **<** p2**.**x**;**

**}**

bool yCompare**(**const Point **&**p1**,** const Point **&**p2**)** **{**

**return** p1**.**y **<** p2**.**y**;**

**}**

double distance**(**Point **&**p1**,** Point **&**p2**)** **{**

double x **=** p1**.**x **-** p2**.**x**;**

double y **=** p1**.**y **-** p2**.**y**;**

**return** sqrt**(**x **\*** x **+** y **\*** y**);**

**}**

double bruteForce**(**vector**<**Point**>** **&**point\_set**,** int left**,** int right**)** **{**

double min\_dist **=** INF**;**

**for** **(**int i **=** left**;** i **<** right**;** **++**i**)** **{**

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

min\_dist **=** min**(**min\_dist**,** distance**(**point\_set**[**i**],** point\_set**[**j**]));**

**}**

**}**

**return** min\_dist**;**

**}**

double stripClosest**(**vector**<**Point**>** **&**point\_set**,** int left**,** int right**,** int mid**,** double dist\_min**)** **{**

Point point\_mid **=** point\_set**[**mid**];**

vector**<**Point**>** splitted\_points**;**

**for** **(**int i **=** left**;** i **<** right**;** i**++)** **{**

**if** **(**abs**(**point\_set**[**i**].**x **-** point\_mid**.**x**)** **<=** dist\_min**)** **{**

splitted\_points**.**push\_back**(**point\_set**[**i**]);**

**}**

**}**

sort**(**splitted\_points**.**begin**(),** splitted\_points**.**end**(),** yCompare**);**

double smallest **=** INF**;**

int l **=** splitted\_points**.**size**();**

**for** **(**int i **=** 0**;** i **<** l**;** i**++)** **{**

**for** **(**int j **=** i **+** 1**;** j **<** l **&&** **(**splitted\_points**[**j**].**y **-** splitted\_points**[**i**].**y**)** **<** dist\_min**;** j**++)** **{**

double d **=** distance**(**splitted\_points**[**i**],** splitted\_points**[**j**]);**

smallest **=** min**(**smallest**,** d**);**

**}**

**}**

**return** smallest**;**

**}**

double closestUtil**(**vector**<**Point**>** **&**point\_set**,** int left**,** int right**)** **{**

**if** **(**right **-** left **<=** 3**)**

**return** bruteForce**(**point\_set**,** left**,** right**);**

int mid **=** **(**right **+** left**)** **/** 2**;**

double dist\_left **=** closestUtil**(**point\_set**,** left**,** mid**);**

double dist\_right **=** closestUtil**(**point\_set**,** mid **+** 1**,** right**);**

double dist\_min **=** min**(**dist\_left**,** dist\_right**);**

**return** min**(**dist\_min**,** stripClosest**(**point\_set**,** left**,** right**,** mid**,** dist\_min**));**

**}**

int main**()** **{**

int n**;**

double x**,** y**;**

**while** **(true)**

**{**

cin **>>** n**;**

**if** **(**n **==** 0**)**

**break;**

vector**<**Point**>** point\_set**;**

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

cin **>>** x **>>** y**;**

point\_set**.**push\_back**(**Point**(**x**,** y**));**

**}**

sort**(**point\_set**.**begin**(),** point\_set**.**end**(),** xCompare**);**

double ans **=** closestUtil**(**point\_set**,** 0**,** n**);**

**if** **(**ans **<** 10000**)**

cout **<<** fixed **<<** setprecision**(**4**)** **<<** ans **<<** endl**;**

**else**

cout **<<** "INFINITY" **<<** endl**;**

**}**

**return** 0**;**

**}**