.

# Estufa en Piloto

## Contents

# 1 Basic

## 1.1 Binary Search

```
def _binsearch(lista, x, l , r):
  """Recursive binsearch between l and r index in lista"""
  #Couldn’t find x in lista
  if(r<l): return -1
  mid = (l+r)//2
  if(lista[mid]  == x): return mid
  return _binsearch(lista, x, l , mid-1) if(x<lista[mid]) else _binsearch(lista, x, mid+1, r)

def binsearch(l, element):
  """Wrapper of recursive binsearch"""
  return _binsearch(l, element , 0 , len(l)-1)
```

## 1.2 LIS

### 1.2.1 Naive

```
# O(n^2) LIS

def LIS(l):
    """Return the lenght of Longest Increasing Subsequence in l"""
    dp = [1 for _ in range(len(l))] #dp[i]: lenght of LIS containing l[i] as its last element

    for i in range(len(l)):
        for j in range(i):
            if(l[j]<l[i]):
                dp[i] = max(dp[i],dp[j]+1)

    return max(dp)
```

### 1.2.2 Fast

```
from bisect import bisect_right
import heapq
def find_gt(a, x):
    ’Find␣leftmost␣value␣greater␣than␣x’
    i = bisect_right(a, x)
    return i if i != len(a) else -1

```

```
def LIS(arr):
    piles = []

    for idx in range(len(arr)):
        aux = [piles[i][-1] for i in range(len(piles))]
        pileNumber = find_gt(aux,arr[idx])
        if(pileNumber==-1):
            piles.append([arr[idx]])
        else:
            piles[pileNumber].append(arr[idx])

    print(piles)
    aux = [piles[i][::-1] for i in range(len(piles))]
    orden = list(heapq.merge(*aux))
    print(orden)
```

# 2 Data Structures

## 2.1 DSU

```
class DisjointSetUnion:
    def __init__(self, array):
        self.parent = [i for i in array]
        self.rank = [0 for _ in range(len(array))]
        self.size = [1] * (len(array) + 1)
        self.group = [[a[i]] for i in array]

    def find(self, x):
        # If x is root
        if self.parent[x] == x:
            return x
        # If x is not root, search again by using x's parent
        else:
            self.parent[x] = self.find(self.parent[x])
            return self.parent[x]

    def union(self, x, y):
        x = self.find(x)
        y = self.find(y)
        # Make an edge from the root of lower tree to the root of higher tree
        if self.rank[x] < self.rank[y]:
            self.parent[x] = y
            self.size[y] += self.size[x]
        else:
            self.parent[y] = x
            self.size[x] += self.size[y]
            # If the height of tree the tree is the same, increase one of the heights by 1
            if self.rank[x] == self.rank[y]:
                self.rank[x] += 1

    def merge(self, x, y):
        x = self.find(x)
        y = self.find(y)
        if len(self.group[x]) < len(self.group[y]):
            x, y = y, x
        self.group[x].extend(self.group[y])
        self.group[y] = []
        self.parent[y] = x

    def check_same(self, x, y):
        return self.find(x) == self.find(y)

    def get_size(self, x):
        return self.size[self.find(x)]
```

## 2.2 FenwickTree

```
class FenwickTreeSum:
    """ BIT de Sumas,
    Resuelve las queries de intervalos y modificacion del array original
    en O(log n)."""
    def __init__(self, n):
        self.bit = [0]*n #Binary Indexed Tree
        self.n = n

    def initArray(self, array):
        for i in range(len(array)):
            self.update(i, array[i])

    def sum(self, r):
        ret = 0
        while(r>=0):
            ret += self.bit[r]
            r = (r&(r+1))-1

        return ret

    def rangeSum(self, l, r):
        return self.sum(r)-self.sum(l-1)

    def update(self, idx, delta): #Add delta to a[idx]
        while(idx<self.n):
            self.bit[idx] += delta
            idx |= (idx+1)



from math import inf
class FenwickTreeMin:
    """ BIT de Min,
    Resuelve las queries de intervalos y modificacion del array original
    en O(log n)."""
    def __init__(self, n):
        self.bit = [inf]*n #Binary Indexed Tree
        self.n = n

    def initArray(self, array):
        for i in range(len(array)):
            self.update(i, array[i])

    def getMin(self, r):
        ret = inf
        while(r>=0):
            ret = min(ret, self.bit[r])
            r = (r&(r+1))-1

        return ret


    def update(self, idx, val):
        while(idx<self.n):
            self.bit[idx] = min(self.bit[idx], val)
            idx |= (idx+1)

"""Revisar
class FenwickTreeSum2D:
    def __init__(self, n, m):
        self.bit = [[0]*(m+1) for _ in range(n+1)] #Binary Indexed Tree 2D
        self.n = n
        self.m = m

    def initArray(self,array):
        aux = [[0]*(self.m+1) for _ in range(self.n+1)]
        for i in range(1,self.n+1):
            for j in range(1,self.m+1):
                aux[i][j] = array[self.n-j][i-1]
        #It's a matrix now
        for j in range(1,self.m+1):
            for i in range(1,self.n+1):
                v1 = self.getSum(i,j)
                v2 = self.getSum(i,j-1)
                v3 = self.getSum(i-1,j-1)
                v4 = self.getSum(i-1,j)
                self.update(i,j,aux[i][j]-(v1-v2-v4+v3))

    def getSum(self, i, j):
        suma = 0
        while(i>0):
            while(j>0):
                suma += self.bit[i][j]
                j = (j & (j+1))-1
            i = (i & (i+1))-1
        return suma


    def update( self, i, j, val):
        while( i <= self.n ):
            while(j <= self.m):
                self.bit[i][j] += val
                j |= (j+1)
            i |= (i+1)

    def answerQuery(self,i1,j1, i2, j2):
        ans = self.getSum(i2+1, j2+1)-self.getSum(i2+1, j1)-self.getSum(i1, j2+1)+self.getSum(i1, j1)

        return ans
"""
```

## 2.3 Segment Tree

```
class SegmentTree:
    """Segment tree of sums"""

    def __init__(self, array):
        self.n = len(array)
        self.t = [0]*self.n + array

        for i in range(self.n-1, 0, -1):
            self.t[i] = self.t[i<<1]+ self.t[(i<<1)|1]

    def modify(self, idx, val):
        idx+=self.n; self.t[idx] = val
        while(idx>1):
            self.t[idx>>1] = self.t[idx] + self.t[idx^1]
            idx >>=1

    def modifyInterval(self, l, r, value):
        """Modifica intervalo [l,r) poniendo value"""
        l+=self.n; r+=self.n
        while(l<r):
            if(l&1):
                l+=1; self.t[l]+= value
            if(r&1):
                r-=1; self.t[r] += value
            l>>=1; r>>=1

    def push(self):
        """Si necesitamos inspeccionar todos los elementos del array,
        es conveniente pushear la info a las hojas, reduce O(nlogn) a O(n)
        """
        for i in range(1,self.n):
            self.t[i<<1] += self.t[i]
            self.t[(i<<1)|1] += self.t[i]
            self.t[i] = 0

    def query(self, l, r):
        """Responde al intervalo [l,r)"""
        res = 0 #Se usa el neutro de la operacion
        l+=self.n ; r+=self.n

        while(l<r):
            if(l&1):
                res += self.t[l]
                l+=1
            if(r&1):
                r-=1
                res += self.t[r]
            l>>=1; r>>=1

        return res

    def queryElement(self, p):
        """Devuelve el valor de un elemento"""
        res = 0 ; p+= self.n
        while(p>0):
            res+= self.t[p]
```

```python
            p>>=1
        return res


a = SegmentTree([1,2,3,4])
a.modifyInterval(0,3,10)
print(a.queryElement(3))
print(a.t)

class SegmentTreeGeneric:
    """Generic Segment Tree. f es una funcion asociativa"""

    def __init__(self, array, f):
        self.n = len(array)
        self.t = [0]*self.n + array
        self.f = f #Me guardo la funcion aca para los otros metodos

        for i in range(self.n-1, 0, -1):
            self.t[i] = self.f(self.t[i<<1],self.t[(i<<1)|1])

    def modify(self, idx, val):
        idx+=self.n; self.t[idx] = val
        while(idx>1):
            self.t[idx>>1] = self.f(self.t[idx] + self.t[idx^1])
            idx >>=1

    def query(self, l, r):
        res = 0 #Se usa el neutro de la operacion
        l+=self.n ; r+=self.n

        while(l<r):
            if(l&1):
                res = self.t[l] if res==0 else self.f(res,self.t[l])
                l+=1
            if(r&1):
                r-=1
                res = self.t[r] if res==0 else self.f(self.t[r],res)
            l>>=1; r>>=1

        return res
```

## 2.4 Sparse Table

```python
from math import log

MAXN = 10**7 #Biggest possible array lenght
K = 25 # Must satisfy K >= floor(log_2{MAXN})+1

"""Generic precomputation"""
def precomputation(array, f):
    n = len(array)
    K = int(log(n,2))+1
    st = [[None for __ in range(K)] for _ in range(n)]

    for i in range(n):
        st[i][0] = f([array[i]])
    for j in range(1,K+1):
        for i in range(n-(1<<j)+1):
            st[i][j] = f([st[i][j-1], st[i+(1<<(j-1))][j-1]])

    return st

"""Range Sum Queries"""
array = [1, 4, -1, 6, 9]
n = len(array)
K = int(log(n,2))+1
st = precomputation(array, sum)

def rangeSumQuery(L, R):
    sum = 0
    for j in range(K,-1,-1):
        if((1<<j) <= R-L+1):
            sum+= st[L][j]
            L += 1<<j

    return sum


"""Range Minimun Queries (RMQ)"""
def precomputeLogs(n):
    logs = {1:0}
    for i in range(2,n+1):
        logs[i] = logs[i//2] + 1

    return logs

```

```
array = [1, 4, -1, 6, 9]
n = len(array)
K = int(log(n,2))+1
st = precomputation(array, min)
logs = precomputeLogs(n)

def rangeMinimumQuery( L, R):
    j = logs[R-L+1]
    return min(st[L][j], st[R-(1<<j)+1][j])
```

# 3 Graph

## 3.1 BFS

```
from collections import deque

empty_queue = lambda q: len(q)==0

def bfs(graph, root, visited = set()):
    travel = []
    q = deque([])

    q.append(root)
    visited.add(root)

    while( not empty_queue(q) ):

        actNode = q.popleft() #Dequeue
        travel.append(actNode)
        for adyacent in graph[actNode]:
            if(adyacent not in visited):
                q.append(adyacent) #Enqueue adyacent
                visited.add(adyacent)

    return travel
```

## 3.2 DFS

```
def rec_dfs(graph , node , visited):

  if node not in visited:
    visited.append(node)

    for ady in graph[node]:
      dfs_rec(graph , ady , visited)


def dfs(graph , node):
  visited = []
  rec_dfs(graph , node , visited)
  print(visited)
```

## 3.3 Dijkstra

```
class Graph():

  def __init__(self, vertices):
    self.V = vertices
    self.graph = [[0 for column in range(vertices)]
          for row in range(vertices)]

  def printSolution(self, dist):
    print "Vertex␣tDistance␣from␣Source"
    for node in range(self.V):
      print node, "t", dist[node]

  def minDistance(self, dist, sptSet):

    min = float('inf')

    for v in range(self.V):
      if dist[v] < min and sptSet[v] == False:
        min = dist[v]
        min_index = v

    return min_index

  def dijkstra(self, src):

    dist = [sys.maxint] * self.V
    dist[src] = 0
    sptSet = [False] * self.V

    for cout in range(self.V):
      u = self.minDistance(dist, sptSet)
      sptSet[u] = True

      for v in range(self.V):
        if self.graph[u][v] > 0 and sptSet[v] == False and \
        dist[v] > dist[u] + self.graph[u][v]:
            dist[v] = dist[u] + self.graph[u][v]

    self.printSolution(dist)
```

### 3.4 Floyd-Warshall

```
def floydWarshall(adyMatrix):
  "APSP␣problem,␣O(V^3)"
  v = len(adyMatrix)
  for k in range(v):
    for i in range(v):
      for j in range(v):
        adyMatrix[i][j] = min(adyMatrix[i][j],adyMatrix[i][k]+adyMatrix[
            k][j])


def transitiveClosure(d):
  """d[i][j] tiene 1 si hay un camino enre i y j,
   0 en caso contrario, O(V^3)"""
  v = len(d)
  for k in range(v):
    for i in range(v):
      for j in range(v):
        d[i][j] |= (d[i][k] & d[k][j])


def minimax(d):
  """Encuentra el minimo entre los maximos edges de cada path"""
  v = len(d)
  for k in range(v):
    for i in range(v):
      for j in range(v):
        d[i][j] = min(d[i][k], max(d[i][k],d[k][j]))

def maximin(d):
  v = len(d)
  for k in range(v):
    for i in range(v):
      for j in range(v):
        d[i][j] = max(d[i][j], min(d[i][k],d[k][j]))
```

### 3.5 Max Flow (Ford Fulkerson)

```
from collections import deque
from math import inf
class Graph:
    def __init__(self, graph):
        self.graph = graph
        self.ROW = len(graph)

```

```python
    def BFS(self,s,t,parent):
        visited = [False]*(self.ROW)
        queue = deque([])
        queue.append(s); visited[s]=True
        while queue:
            u = queue.popleft()
            for ind, val in enumerate(self.graph[u]):
                if(visited[ind]==False and val>0):
                    queue.append(ind)
                    visited[ind]=True
                    parent[ind] = u
        return visited[t]

    def FordFulkerson(self, source, sink):
        parent = [-1]*(self.ROW)
        max_flow = 0
        while self.BFS(source,sink,parent):
            path_flow = inf
            s = sink
            while(s!=source):
                path_flow = min(path_flow,self.graph[parent[s]][s])
                s = parent[s]
            max_flow += path_flow
            v = sink
            while(v!= source):
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                v = parent[v]
        return max_flow
```

# 4 Math

## 4.1 Identities

$$C_n = \frac{2(2n-1)}{n+1}C_{n-1}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

$\sigma(n) = O(\log(\log(n)))$ (number of divisors of $n$)

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$

$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$$

(Möbius Inv. Formula) Let $g(n) = \sum_{d|n} f(d)$, then $f(n) = \sum d \mid ng(d)\mu\left(\frac{n}{d}\right))$.

## 4.2 GCD

```python
gcd = lambda a, b : a if(b==0) else gcd(b,a%b)

def it_gcd(a, b):
    while(b):
        a%=b
        a, b = b, a #Swap para tener el mas chico en b
    return a
```

## 4.3 Euler Totient (Phi)

```python
def phi(n):
  """O(sqrt(n)) approach using factorization"""
  result = n
  i=2
  while(i*i<=n):
    if(n%i==0):
      while(n%i == 0):
        n//=i
      result -= result//i
    i+=1
  if(n>1):
    result-= result//n

  return result
```

## 4.4 Extended Euclides

```python
def gcd(a, b):
    """Devuelve el gcd entre a y b, y coefx y coefy tales
    que a*coefx+b*coefy = gcd"""
    if(a==0):
        return b, 0, 1
    d, x1, y1 = gcd(b%a, a)
    x = y1-(b//a)*x1
    y = x1
    return d, x, y


def modinv(a, m):
    g, x, y = gcd(a,m)
    if(g!=1):
        return None
    x = (x%m + m)%m
    return x
```

## 4.5 Modexp

```python
def modexp( x, y, p ):
  """Exponenciacion logaritmica iterativa,
  x^y (mod p), el orden el O(log y)"""

  res = 1
  while(y>0):
    if(y & 1):
      res*= x
      res%=p
    y >>= 1
    x*= x

  return res%p



"""Inverso si m es primo"""
modinv = lambda a, m : modexp(a, m-2, m)
```

## 4.6 allModInv

```python
"""Find all invmods in range [1,m-1] in O(m)"""
def allModInvs(m):
    inv = [1]*(m) #Remember that inv[i] has the inverse of i
    inv[0] = None
    for i in range(2,m):
        inv[i] = -(m//i)*inv[m%i] %m

    return inv
```

## 4.7 Iter multiplication

```python
def logmul(a, b):
    """No creo que sea necesario en python, pero version recursiva
    de la multiplicacion que sirve en C++"""
    if(a == 0):
        return 0
    return 2*logmul((a-1)//2,b)+b if(a%2) else 2*logmul(a//2,b)


def logmulmod(a, b, p):
    """Multiplicacion recursiva mod p"""
    if(a == 0):
        return 0

    res = 2*logmulmod((a-1)//2,b, p)+b if(a%2) else 2*logmulmod(a//2,b,
        p)
    return res%p
```

## 4.8 Test de primalidad y Descomposicion en Primos

### 4.8.1 Is Prime

```python
def is_prime(n):
    """Naive O(sqrt(n)) approach"""
    d = 2
    while(d*d<=x):
        if(x%d == 0):
            return False
        d+=1
    return True

from random import randint

RAND_MAX = 10**9
def modexp( x, y, p ):
  res = 1
  while(y>0):
    if(y & 1):
      res*= x
      res%=p
    y >>= 1
    x*= x

  return res%p

def probablyPrimeFermat(n, iter=5):
    if (n < 4):
        return n == 2 or n == 3

    for i in range(iter):
        a = 2 + randint(1, RAND_MAX)%(n - 3);
        if (modexp(a, n - 1, n)!=1):
            return False

    return True

for i in range(100):
    print(i,probablyPrimeFermat(i,5))
```

### 4.8.2 Miller Rabin Test

```python
from random import randint
RAND_MAX = 10**9
def modexp( x, y, p ):
  res = 1
  while(y>0):
    if(y & 1):
      res*= x
      res%=p
    y >>= 1
    x*= x

  return res%p

def check_composite( n, a, d, s):
    x = modexp(a, d, n)
    if (x == 1 or x == n - 1):
        return False
    for r in range(1,s):
        x = x * x % n;
        if (x == n - 1):
            return False

    return True

def MillerRabinProb(n, iter = 5):
    """Returns true if n is probably prime, else returns false."""
    if (n < 4):
        return n == 2 or n == 3

    s = 0;
    d = n - 1;
    while ((d & 1) == 0):
        d >>= 1;
        s+=1

    for i in range(iter):
        a = 2 + randint(1,RAND_MAX) % (n - 3);
        if(check_composite(n, a, d, s)):
            return False
    return True


"""Deterministic Version"""
def MillerRabin(n):
    if(n<2):
        return False

    r = 0
    d = n - 1
    while ((d & 1) == 0):
        d >>= 1
        r+=1

    for a in {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
cnt = 0
for i in range(100000):
    if(MillerRabin(i)):
        cnt+=1
print(cnt)
```

### 4.8.3 Prime Factors

```
# Program to fiind prime factors and their powers
# using Sieve Of Eratosthenes

def sieveOfEratosthenes(N, s):
    """Sieve with smaller prime factor of s[i]"""

    prime = [False] * (N+1)

    for i in range(2, N+1, 2):
        s[i] = 2

    for i in range(3, N+1, 2):
        if (prime[i] == False):
            s[i] = i

            for j in range(i, N//i + 1, 2):
                if (prime[i*j] == False):
                    prime[i*j] = True
                    s[i * j] = i


def generatePrimeFactors(N):

    #s[i]: smallest prime factor
    s = [0] * (N+1)
    sieveOfEratosthenes(N, s)

    factors = [] #Contains tuples (p_i, alpha_i)

    curr = s[N]
    # Power of current prime factor
    cnt = 1

    while (N > 1):
        N //= s[N]

        if (curr == s[N]):
            cnt += 1
            continue

        factors.append((curr,cnt))
        curr = s[N]
        cnt = 1
    return factors
```

### 4.8.4 Integer Fact

```
"""Naive O(sqrt(n)) approach"""
def trial_division1(n):
    factorization = []
    d = 2
    while(d*d<=n):
        while(n%d == 0):
            factorization.append(d)
            n//=d
        d+=1

    if(n>1):
        factorization.append(n)
    return factorization

"""Don't try even numbers if it's odd. (Wheel factorization)"""
def trial_division2(n):
    factorization = []
    while(n%2 == 0):
        factorization.append(2)
        n >>= 1

    d = 3
    while(d*d<=n):
        while(n%d==0):
            factorization.append(d)
            n //= d
        d+=2
    if(n>1):
        factorization.append(n)
    return factorization

"""Same Wheel idea, but with factors 2,3,5 at the same time."""
def trial_division3(n):
    factorization = []
    for d in {2,3,5}:
        while(n%d==0):
            factorization.append(d)
            n //= d

    increments = [4, 2, 4, 2, 4, 6, 2, 6]
    i = 0
    d = 7
    while(d*d<=n):
        while(n%d == 0):
            factorization.append(d)
            n//= d
        i %= 8
        d+=increments[i]

    if(n>1):
        factorization.append(n)
    return factorization


"""Precomputed primes"""



def trial_division4(n):
    """Fast way to find primes"""
    MAX_PRIME = 10**6
    def listPrimesFast(max_n):
        """sundaram3"""
        numbers = [i for i in range(3, max_n+1, 2)]
        half = (max_n)//2
        initial = 4

        for step in range(3, max_n+1, 2):
            for i in range(initial, half, step):
                numbers[i-1] = 0
            initial += 2*(step+1)

        if initial > half:
            return [2] + filter(None, numbers)

    primes = listPrimesFast(MAX_PRIME)
    factorization = []
    for d in primes:
        if(d*d>n):
            break
        while(n%d == 0):
            factorization.append(d)
            n//=d
    if(n>1):
        factorization.append(n)

    return factorization


"""Pollard's p-1 method"""
from random import randint
from math import gcd
MAX_RAND = 10**9
def pollard_p_minus_1(n):
    """Probabilistic method, O(BlogB log^2 n), encuentra un primo
    que divide a n"""
    MAX_PRIME = 10**6
    def listPrimesFast(max_n):
        """sundaram3"""
        numbers = [i for i in range(3, max_n+1, 2)]
        half = (max_n)//2
        initial = 4

        for step in range(3, max_n+1, 2):
            for i in range(initial, half, step):
                numbers[i-1] = 0
            initial += 2*(step+1)

        if initial > half:
            return [2] + list(filter(None, numbers))

    def modexp( x, y, p ):
      res = 1
      while(y>0):
        if(y & 1):
          res*= x
          res%=p
        y >>= 1
        x*= x

```

```python
      return res%p

    B = 10
    g = 1
    primes = listPrimesFast(MAX_PRIME)
    while(B <= 10**7 and g<n):
        a = 2+ randint(1,MAX_RAND) % (n-3)
        g = gcd(a,n)
        if(g>1):
            return g

        #Computo a^M
        for p in primes:
            if(p>=B):
                continue
            p_power = 1
            while(p_power * p <= B):
                p_power *= p
            a = modexp(a, p_power, n)

            g = gcd(a-1, n)
            if(g>1 and g<n):
                return g
        B*=2

    return 1


"""Pollard Rho algorithm to find a factor of n"""
from math import gcd
def floyd(f, x0):
    tortoise = x0
    hare = f(x0)
    while(tortoise != hare):
        tortoise = f(tortoise)
        hare = f(f(hare))

    return true

mult = lambda a, b, mod: a*b % mod
f = lambda x, c, mod: (mult(x,x,mod)+c)%mod

def rho(n, x0=2, c=1):
    x, y, g = x0, x0, 1
    while(g==1):
        x = f(x, c, n)
        y = f(y, c, n)
        y = f(y, c, n)
        g = gcd(abs(x-y), n)

    return g


"""Brent, direct implementation"""
from math import gcd
mult = lambda a, b, mod: a*b % mod
f = lambda x, c, mod: (mult(x,x,mod)+c)%mod

def brent(n, x0 = 2, c=1):
    x, g, q= x0, 1, 1
    m = 128
    l = 1
    while(g==1):
        y = x
        for i in range(1,l):
            x = f(x, c, n)
        k = 0

        while(k<l and g==1):
            xs = x
            i = 0
            while(i<m and i<l-k):
                x = f(x, c, n)
                q = mult(q, abs(y-x), n)
                i+=1
            g = gcd(q, n)
            k+=m
        l*=2

    if(g==n):
        while True:
            xs = f(xs, c, n)
            g = gcd(abs(xs-y), n)
            if(g!=1 and g!=n):
                break
    return g
```

## 4.9 Diofanticas

```python
def gcd(a, b):
    """Devuelve el gcd entre a y b, y coefx y coefy tales
    que a*coefx+b*coefy = gcd"""
    if(a==0):
        return b, 0, 1
    g, x1, y1 = gcd(b%a, a)
    x = y1-(b//a)*x1
    y = x1
    return g, x, y

def find_any_solution(a, b, c):
    """Returns g = gcd(a,b) and a pair of coef x0, y0 such that
    a*x0+b*y0 = g"""
    g, x0, y0 = gcd(abs(a), abs(b))
    if (c % g):
        return False

    x0 *= c // g;
    y0 *= c // g;
    if (a < 0): x0 = -x0;
    if (b < 0): y0 = -y0;
    return g, x0, y0


def shift_solution(x, y, a, b, cnt):
    """Dada una solucion a la ecuacion diofantica, encuentra otra"""
    x += cnt * b;
    y -= cnt * a;
    return x, y


def find_all_solutions( a, b, c, minx, maxx, miny, maxy):
    sol = find_any_solution(a, b, c)
    if (not sol):
        return 0
    g, x, y = sol
    a //= g
    b //= g

    sign_a = 1 if(a > 0) else -1
    sign_b = 1 if(b > 0) else -1

    x, y = shift_solution(x, y, a, b, (minx - x) // b)
    if (x < minx):
        x, y = shift_solution(x, y, a, b, sign_b)
    if (x > maxx):
        return 0
    lx1 = x

    x, y = shift_solution(x, y, a, b, (maxx - x) // b);
    if (x > maxx):
        x, y = shift_solution(x, y, a, b, -sign_b)
    rx1 = x

    x, y = shift_solution(x, y, a, b, -(miny - y) // a)
    if (y < miny):
        x, y = shift_solution(x, y, a, b, -sign_a)
    if (y > maxy):
        return 0
    lx2 = x

    x, y = shift_solution(x, y, a, b, -(maxy - y) // a)
    if (y > maxy):
        shift_solution(x, y, a, b, sign_a)
    rx2 = x

    if (lx2 > rx2):
        lx2, rx2 = rx2, lx2
    lx = max(lx1, lx2);
    rx = min(rx1, rx2);

    if (lx > rx):
        return 0

    # Para enumerarlas, basta iterar desde x = lx+ k*b//g, el numero
    # necesario de soluciones

    return (rx - lx) // abs(b) + 1;
```

## 4.10 Cribas

### 4.10.1 Criba Primos

```python
"""Criba sin optimizar, O(n* log log n)"""

```

```python
n = 10**6
is_prime = [True]*(n+1)

is_prime[0] = False
is_prime[1] = False

for i in range(2,n+1):
  if(is_prime[i] and i*i<=n):
    for j in range(i*i,n+1,i):
      is_prime[j] = False


"""Sieving till root"""

n = 10**6
is_prime = [True]*(n+1)

is_prime[0] = False
is_prime[1] = False

for i in range(2,n+1):
  if(is_prime[i]):
    for j in range(i*i,n+1,i):
      is_prime[j] = False
```

### 4.10.2 Block Sieving

```python
"""Cuento los primos menores a n usando block_sieving"""

def count_primes(n):
  S = 10000
  primes = []
  nsqrt = int(n**0.5)
  is_prime = [True]*(nsqrt+1)

  for i in range(2, nsqrt+1):
    if(is_prime[i]):
      primes.append(i)
      for j in range(i*i,nsqrt+1,i):
        is_prime[j] = False

  result = 0
  block = [True]*S

  k = 0
  while(k*S <= n):
    block = [True]*S
    start = k*S

    for p in primes:
      start_idx = (start+p-1)//p
      j = max(start_idx, p )* p - start
      while(j<S):
        block[j] = False
        j+=p

    if( k == 0):
        block[0] = False
        block[1] = False
    i = 0
    while(i<S and start+i <= n):
      if(block[i]):
        result+=1
      i+=1
    k+=1

  return result
```

### 4.10.3 Linear Sieve

```python
N = 10**6
lp = [0]*(N+1)
pr = []

import timeit
start = timeit.default_timer()
for i in range(2,N+1):
    if(lp[i]==0):
        lp[i]=i
        pr.append(i)

    j = 0
    while(j<len(pr) and pr[j]<=lp[i] and i*pr[j]<=N):
        lp[i*pr[j]] = pr[j]
        j+=1
end = timeit.default_timer()
```

```python
print(end-start)

print(lp[:20])
print(pr[:20])
```

## 4.11 Divisores

### 4.11.1 Count Divisors

```python
def countDivisors(n):
    """Naive implementation to count divisors of n, in O(sqrt(n))"""
    cnt = 0
    for i in range(1, int(n**0.5)+1):
        if (n % i == 0):
            cnt += 1 if(i*i == n) else 2

    return cnt



"""Improved method in O(n^(1/3))"""
def SieveOfEratosthenes(n, prime,primesquare, a):
    """Sieve of primes and squares of primes."""
    for i in range(2,n+1):
        prime[i] = True

    for i in range((n * n + 1)+1):
        primesquare[i] = False

    prime[1] = False

    p = 2
    while(p * p <= n):
        if (prime[p] == True):
            i = p * 2
            while(i <= n):
                prime[i] = False
                i += p
        p+=1

    j = 0
    for p in range(2,n+1):
        if (prime[p]==True):
            a[j] = p

            primesquare[p * p] = True
            j+=1

def countDivisorsFast(n):
    """Count divisors in O(n^(1/3))"""
    if (n == 1):
        return 1

    prime = [False]*(n + 2)
    primesquare = [False]*(n * n + 2)

    a = [0]*n

    SieveOfEratosthenes(n, prime, primesquare, a)

    ans = 1

    i=0
    while(1):
        if(a[i]**3 > n):
            break

        cnt = 1
        while (n % a[i] == 0):
            n //= a[i]
            cnt += 1
        ans *= cnt
        i+=1

    if(prime[n]==True):
        ans *= 2

    elif(primesquare[n]==True):
        ans *= 3

    elif(n != 1):
        ans *= 4

    return ans
```

### 4.11.2 Suma Divisores

```python
# Sum of all divisors of n.
def sumofFactors(n):
    res = 1
    for i in range(2, int(n**0.5 + 1)):
        curr_sum = 1
        curr_term = 1

        while n % i == 0:
            n //= i
            curr_term *= i
            curr_sum += curr_term;

        res *= curr_sum

    if n > 2:
        res *= (1 + n)

    return res
```

## 4.12 Factorial

### 4.12.1 Factorial mod

```python
def factmod( n, p ):
    """O(p log n ) aproach to calculate n! (mod p)"""
    if(n>p):
        return 0
    res = 1
    while(n > 1):
        res = (res* (p-1 if((n//p)%2) else 1)) % p;
        for i in range(2, n%p+1):
            res = (res * i) % p
        n //= p

    return res % p;
```

### 4.12.2 Factorial Divisors

```python
# Count of divisors of n!

allPrimes = []
def sieve(n):
    prime = [True] * (n + 1)

    p = 2
    while(p * p <= n):
        if (prime[p] == True):
            i = p * 2
            while(i <= n):
                prime[i] = False
                i += p
        p += 1

    for p in range(2, n + 1):
        if (prime[p]):
            allPrimes.append(p)

def factorialDivisors(n):

    sieve(n)
    result = 1

    for i in range(len(allPrimes)):
        p = allPrimes[i]
        exp = 0
        while (p <= n):
            exp += n // p
            p *= allPrimes[i]

        result *= (exp + 1)

    return result
```

## 4.13 FFT

```python
from cmath import exp
from math import pi

# A simple class to simulate n-th root of unity
class NthRootOfUnity:
    def __init__(self, n, k = 1):
        self.k = k
        self.n = n

    def __pow__(self, other):
        if type(other) is int:
            n = NthRootOfUnity(self.n, self.k * other)
            return n

    def __eq__(self, other):
        if other == 1:
            return abs(self.n) == abs(self.k)

    def __mul__(self, other):
        return exp(2*1j*pi*self.k/self.n)*other

    def __repr__(self):
        return str(self.n) + "-th␣root␣of␣unity␣to␣the␣" + str(self.k)

    @property
    def th(self):
        return abs(self.n // self.k)


""" The Fast Fourier Transform Algorithm

 Input: A, An array of integers of size n representing a polynomial
        w , a n-root of unity
 Output: [A(w), A(w^2), ..., A(w^(n-1))]
 Complexity: O(n logn)"""
def FFT(A, omega):
    if omega == 1:
        return [sum(A)]
    o2 = omega**2
    C1 = FFT(A[0::2], o2)
    C2 = FFT(A[1::2], o2)
    C3 = [None]*omega.th
    for i in range(omega.th//2):
        C3[i] = C1[i] + omega**i * C2[i]
        C3[i+omega.th//2] = C1[i] - omega**i * C2[i]
    return C3

""" The Fast Polynomial Multiplication Algorithm

 Input: A,B, two arrays of integers representing polynomials
 (coef in increasing deg) their length is in O(n)
 Output: Coefficient representation of AB
 Complexity: O(n logn)"""
def FPM(A,B):
    n = 1<<(len(A)+len(B)-2).bit_length()
    o = NthRootOfUnity(n)
    AT = FFT(A, o)
    BT = FFT(B, o)
    C = [AT[i]*BT[i] for i in range(n)]
    D = [round((a/n).real) for a in FFT(C, o ** -1)]
    while len(D) > 0 and D[-1] == 0:
        del D[-1]
    return D
```

## 4.14 Lagrange Interpolation

```
def interpolacion(muestra, x):
  """Recibe una muestra de n puntos (tuplas (xi,yi))
  y evalua el polinomio de Lagrange de
  en el punto x desconocido"""
  ans = 0
  n = len(muestra)
  for i in range(n):
    term = muestra[i][1]
    for j in range(n):
      if(j!=i):
        term = term*(x-muestra[j][0])/(muestra[i][0]-muestra[j][0])
    ans+=term
  return ans
```

## 4.15 Discrete Log

```
def discreteLog(a, b, m):
    n = int(m**0.5)+ 1
    an = 1;
    for i in range(n):
        an = (an * a) % m

    vals = {}
    cur = an
    for p in range(1,n+1):
        if not cur in vals:
            vals[cur] = p
        cur = (cur * an) % m

    cur = b
    for q in range(n+1):
        if cur in vals:
            ans = vals[cur] * n - q
```

```
      return ans
    cur = (cur * a) % m

  return -1
```

## 4.16 Discrete Root

```
from math import gcd

def modexp( x, y, p ):
 """Exponenciacion logaritmica iterativa,
 x^y (mod p), el orden el O(log y)"""

 res = 1
 while(y>0):
  if(y & 1):
   res*= x
   res%=p
  y >>= 1
  x*= x

 return res%p


# Finds the primitive root modulo p
def generator(p):
  fact = []
  phi = p-1
  n = phi
  i = 2
  while(i*i<=n):
    if(n % i == 0):
      fact.append(i);
      while (n % i == 0):
        n //= i
    i+=1

  if(n > 1): fact.push_back(n)

  for res in range(2,p+1):
    ok = True;
    for factor in fact:
      if(modexp(res, phi // factor, p) == 1):
        ok = False
        break
    if(ok): return res
  return -1;
```

## 4.17 Primitive Root

```
from math import sqrt

def isPrime(n):
  if (n <= 1):
    return False
  if (n <= 3):
    return True

  if (n % 2 == 0 or n % 3 == 0):
    return False
  i = 5
  while(i * i <= n):
    if (n % i == 0 or n % (i + 2) == 0) :
      return False
    i = i + 6

  return True

def modexp( x, y, p ):
 res = 1
 while(y>0):
  if(y & 1):
   res*= x
   res%=p
  y >>= 1
  x*= x

 return res%p


"""O(log^6 p)"""
def findPrimefactors(s, n):
  while (n % 2 == 0) :
    s.add(2)
    n //= 2

```

```python
    for i in range(3, int(sqrt(n)), 2):
        while (n % i == 0):
            s.add(i)
            n //= i

    if (n > 2):
        s.add(n)

def findPrimitive(n):
    s = set()

    if (isPrime(n) == False):
        return -1

    phi = n-1
    findPrimefactors(s, phi)

    for r in range(2, phi + 1):
        flag = False
        for it in s:
            if (modexp(r, phi // it, n) == 1):
                flag = True
                break
        if (flag == False):
            return r
    return -1
```

### 4.18 Fibonacci

```python
def matrix_multiplication( A, B):
  """O(n^3) matrix multiplication"""
  result = [[sum(a * b for a, b in zip(A_row, B_col))
                        for B_col in zip(*B)]
                                for A_row in A]
  return result

def identity(n):
  """Matriz identidad de dimension n"""
  return [[1 if i==j else 0 for j in range(n)] for i in range(n)]

def matrix_exp( A, n ):
  """Exponenciacion logaritmica iterativa de matrices,
  A^n, el orden es O(d^3*log n), con d la dimension de la
  matriz"""

  res = identity(len(A))
  while(n>0):
    if(n & 1):
      res = matrix_multiplication(res,A)

    n >>= 1
    A = matrix_multiplication(A, A)

  return res

def fibo_log(n):
    """Returns f_n, using matrix log exponenciation"""
    f = [[0],[1]]
    transition = [[0,1],[1,1]]
    result = matrix_multiplication(matrix_exp(transition,n),f)
    return result[0][0]

"""Could lead to a WA, but O(1) solution (check if **n does not lead to O(n))
"""
def fibo_approx(n):
    phi, phi_conj = (1+5**0.5)/2, (1-5**0.5)/2

    return round((phi**n-phi_conj**n)/(5**0.5))


"""Fast doubling Method"""
def fibo_fast_doubling(n):
    if (n == 0):
        return (0, 1)

    p = fibo_fast_doubling(n >> 1)
    c = p[0]*(2*p[1]- p[0])
    d = p[0]**2 + p[1]**2
    return (d, c + d) if(n & 1) else (c, d)
```

### 4.19 Gray Code

```python
def g(n):
    return n^(n>>1)

```

```python
def rev_g(g):
    n = 0
    while(g):
        n^=g
        g>>=1
    return n
```

### 4.20 Matrix

```python
matrix_sum = lambda X,Y: [list(map(sum, zip(*t))) for t in zip(X, Y)]
```

```python
matrix_sum = lambda X,Y: [list(map(sum, zip(*t))) for t in zip(X, Y)]
```

# 5 Geo

## 5.1 Convex Hull

```python
from functools import reduce
def convex_hull_graham(points):
    TURN_LEFT, TURN_RIGHT, COLLINEAR = (1, -1, 0)
    def cmp(a, b):
        return (a > b) - (a < b)

    def turn(p, q, r):
        return cmp((q[0] - p[0])*(r[1] - p[1]) - (r[0] - p[0])*(q[1] - p[1]), 0)

    def _keep_left(hull, r):
        while len(hull) > 1 and turn(hull[-2], hull[-1], r) not in {TURN_LEFT}:
          #Agregar a {TURN_LEFT} el elemento COLLINEAR si se quieren alineados en
          #En la frontera
            hull.pop()
        if not len(hull) or hull[-1] != r:
            hull.append(r)
        return hull

    points = sorted(points)
    l = reduce(_keep_left, points, [])
    u = reduce(_keep_left, reversed(points), [])
    return l.extend(u[i] for i in range(1, len(u) - 1)) or l
```

## 5.2 Determinant

```python
def determinant(puntos):
  """Recibe una lista de puntos (tuplas 2D)
  y calcula su -determinante-"""
  result = 0
  size = len(puntos)
  for i in range(size):
    x1, x2 = puntos[i][0], puntos[(i+1)%size][0]
    y1, y2 = puntos[i][1], puntos[(i+1)%size][1]

    result += (x1*y2-x2*y1)

  return result
```

## 5.3 GreatCircleDistance

```
from math import *

def gratCircleDistance( pLat, pLong, qLat, qLong, radius ):

  pLat *= pi/180
  pLong *= pi/180
  qLat *= pi/180
  qLong *= pi/180

  return radius*acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(qLong)+
            cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong)+
            sin(pLat)*sin(qLat))
```

## 5.4 Perimeter

```
from math import sqrt
def perimeter(puntos):
  """Recibe una lista de puntos 2D que representan
    los vertices de un poligono y devuelve el perimetro
    del mismo."""
  result = 0
  size = len(puntos)
  for i in range(size):
    x1, x2 = puntos[i][0], puntos[(i+1)%size][0]
    y1, y2 = puntos[i][1], puntos[(i+1)%size][1]
    dx, dy = x2-x1, y2-y1
    result += sqrt(dx**2+dy**2)

  return result
```

## 5.5 Turn

```
def turn( p, q, r):
  """Recibe tres puntos p, q y r (2D) y
    devuelve si se encuentran en sentido horario,
    antihorario o alineados"""

  result = (r[0]-q[0])*(p[1]-q[1]) - (r[1]-q[1])*(p[0]-q[0])

  if(result < 0):
    return -1 # P->Q->R es una terna derecha (CCW)

  if(result > 0):
    return 1 # P->Q->R es una terna izquierda

  return 0 # P->Q->R colineales

# Wrapper para chequear directamente CCW, si se toleran colineales usar >=
ccw = lambda p, q, r: turn(p, q, r)>0
```

# 6 Strings

## 6.1 Hash

```python
def compute_hash(s):
  p = 31 #Cambiar esto por un primo un poco mas grande
  m = 10**9+9
  p_pow = 1
  hash_value = 0
  for c in s:
    hash_value = (hash_value + (ord(c)-ord(’a’)+1)*p_pow)%m
    p_pow = (p_pow * p)%m
  return hash_value
```

## 6.2 Prefix Function

```python
def prefixFunction(s):
  """Devuelve un array pi, donde pi[i]
  coincide con el mayor prefijo propio que ademas
  es sufijo de s[0...i]"""
  n = len(s)
  pi = [0 for _ in range(n)]

  for i in range(1,n):
    j = pi[i-1]
    while(j>0 and s[i]!=s[j]):
      j = pi[j-1]
    if(s[i]==s[j]): j+=1
    pi[i] = j

  return pi
```

## 6.3 KMP

```python
def prefixFunction(s):
  """Devuelve un array pi, donde pi[i]
  coincide con el mayor prefijo propio que ademas
  es sufijo de s[0...i]"""
  n = len(s)
  pi = [0 for _ in range(n)]

  for i in range(1,n):
    j = pi[i-1]
    while(j>0 and s[i]!=s[j]):
      j = pi[j-1]
    if(s[i]==s[j]): j+=1
    pi[i] = j

  return pi

def KMP(s,t):
  """Encuentra todas las ocurrencias de s en t en
  O(|s|+|t|)"""
  n = len(s); m = len(t)
  separator = "#"
  #Elegir algun caracter que no este en s ni en t

  pi = prefixFunction(s+separator+t)
  occurences = []

  for i in range(n+1,n+m+1):
    if(pi[i]==n): occurences.append(i-2*n)
  return occurences

print(prefixFunction("ab#abab"))
print(KMP("ab","abab"))
```

## 6.4 Rabin Karp

```python
def rabin_karp(s, t):
  """O(|s|+|t|). Given a pattern s and a text t,
  determine if the pattern appears in the text and if it does,
  enumerate all its occurrences."""
  p = 31
  m = 10**9+9
  S = len(s); T = len(T)
  p_pow = [1 for _ in range(max(S,T))]
  for i in range(1,len(p_pow)):
    p_pow[i] = (p_pow[i-1]*p)%m
  h = [0 for i in range(T+1)]
  for i in range(T):
    h[i+1] = (h[i]+ (ord(t[i])-ord(’a’)+1)*p_pow[i])%m
  h_s = 0
  for i in range(S):
    h_s = (h_s + (ord(s[i])-ord(’a’)+1)*p_pow[i])%m

  occurences = []
  for i in range(T-S+1):
    cur_h = (h[i+S]-h[i])%m
    if(cur_h == h_s * p_pow[i]%m):
      occurences.append(i)
  return occurences
```

## 6.5 Aplicaciones de KMP y Rabin

```
def prefixFunction(s):
  """Devuelve un array pi, donde pi[i]
  coincide con el mayor prefijo propio que ademas
  es sufijo de s[0...i]"""
  n = len(s)
  pi = [0 for _ in range(n)]

  for i in range(1,n):
    j = pi[i-1]
    while(j>0 and s[i]!=s[j]):
      j = pi[j-1]
    if(s[i]==s[j]): j+=1
    pi[i] = j

  return pi

def numOccurencesPrefix(s):
  """Cuenta el numero de apariciones de
  cada prefijo de s en s"""
  pi = prefixFunction(s)
  n = len(s)
  ans = [0 for i in range(n+1)]
  for i in range(n):
    ans[pi[i]]+=1
  i = n-1
  while(i>0):
    ans[pi[i-1]] += ans[i]
    i-=1
  for i in range(n+1):
    ans[i]+=1
  return ans

def numDifferentSubstring(s):
  """Dada una string s, cuenta la cantidad de
  substrings diferentes que contiene. O(|s|^2)"""
  ans = 1; n = len(s)
  for i in range(1,n):
    pimax = max(prefixFunction(s[:i+1][::-1]))
    ans += i+1-pimax
  return ans
```

## 6.6 Unique Substrings

```
def countUniqueSubstrings(s):
    n = len(s)
    p = 31; m = 10**9+9
    p_pow = [1 for _ in range(n)]

    for i in range(1,n):
        p_pow[i] = (p_pow[i-1]*p)%m

    h = [0 for i in range(n+1)]
    for i in range(n):
        h[i+1] = (h[i]+(ord(s[i])-ord(’a’)+1) * p_pow[i])%m
    cnt = 0
    for l in range(1,n+1):
        hs = set()
        for i in range(n-l+1):
            cur_h = (h[i+l]+m-h[i])%m
            cur_h = (cur_h*p_pow[n-i-1]) % m
            hs.add(cur_h)
        cnt += len(hs)
    return cnt

print(countUniqueSubstrings(’abcaa’))
```

## 6.7 Group Identical Strings

```
def compute_hash(s):
  p = 31 #Cambiar esto por un primo un poco mas grande
  m = 10**9+9
  p_pow = 1
  hash_value = 0
  for c in s:
    hash_value = (hash_value + (ord(c)-ord(’a’)+1)*p_pow)%m
    p_pow = (p_pow * p)%m
  return hash_value

def groupIdenticalStrings(s):
```

```python
  """Recibo una lista de strings, devuelvo los grupos de strings identicas
  O(n*m+n*log n) (n=len(s), m = maxlen entre las strings"""
  n = len(s)
  hashes = [None for i in range(n)]
  for i in range(n):
    hashes[i] = (compute_hash(s[i]),i)
  hashes.sort()
  groups = []
  for i in range(n):
    if(i==0 or hashes[i][0] != hashes[i-1][0]):
      groups.append([])
    groups[-1].append(hashes[i][1])
  return groups

print(groupIdenticalStrings(["aaa","bca","aaa","amclakm"]))
```

# 7 Tricks With Bits

```
In python3, ~x (flip all bits in other languages) is achieved with
(~x & 0xFFFFFFFF) (use repit1 lenght of HEXA as you wish)


x & (x-1)
clear the lowest set bit of x
x & ~(x-1)
extracts the lowest set bit of x (all others are clear).
Pretty patterns when applied to a linear sequence.
x & (x + (1 << n))
the run of set bits (possibly length 0) starting at bit n cleared.
x & ~(x + (1 << n))
the run of set bits (possibly length 0) in x, starting at bit n.
x | (x + 1)
x with the lowest cleared bit set.
x | ~(x + 1)
Extracts the lowest cleared bit of x (all others are set),
if ~ wrapping the expression, you have that cleared value.
x | (x - (1 << n))
x With the run of cleared bits (possibly length 0) starting at bit n set.
x | ~(x - (1 << n))
The lowest run of cleared bits (possibly length 0) in x,
starting at bit n are the only clear bits.


By 'run' is intended the number formed by all consecutive
1's at the left of n-th bit, starting at n-th bit.
```

# 8 Plantilla

```python
I = lambda : int(input())
LI = lambda : [int(x) for x in input().split()]
MI = lambda : map(int, input().split())
SI = lambda : input()

"""
#Leer de archivo
for line in sys.stdin:
    ...
"""

from collections import *
from heapq import *
from math import *
import itertools
```