

Estufa en Piloto



Contents

1 Basic	2
1.1 Binary Search	2
1.2 LIS	2
1.2.1 Naive	2
1.2.2 Fast	2
2 Data Structures	3
2.1 DSU	3
2.2 FenwickTree	4
2.3 Segment Tree	5
2.4 Sparse Table	6
3 Graph	7
3.1 BFS	7
3.2 DFS	7
3.3 Dijkstra	7
3.4 Floyd-Warshall	8
3.5 Max Flow (Ford Fulkerson)	8
4 Math	9
4.1 Identities	9
4.2 GCD	9
4.3 Euler Totient (Phi)	9

4.4 Extended Euclides	9
4.5 Modexp	10
4.6 allModInv	10
4.7 Iter multiplication	10
4.8 Test de primalidad y Descomposicion en Primos	10
4.8.1 Is Prime	10
4.8.2 Miller Rabin Test	11
4.8.3 Prime Factors	12
4.8.4 Integer Fact	12
4.9 Diofanticas	15
4.10 Cribas	15
4.10.1 Criba Primos	15
4.10.2 Block Sieving	16
4.10.3 Linear Sieve	16
4.11 Divisores	17
4.11.1 Count Divisors	17
4.11.2 Suma Divisores	17
4.12 Factorial	18
4.12.1 Factorial mod	18
4.12.2 Factorial Divisors	18
4.13 FFT	18
4.14 Lagrange Interpolation	19
4.15 Discrete Log	19
4.16 Discrete Root	20
4.17 Primitive Root	20
4.18 Fibonacci	21
4.19 Gray Code	21
4.20 Matrix	22
5 Geo	22
5.1 Convex Hull	22
5.2 Determinant	22
5.3 GreatCircleDistance	23
5.4 Perimeter	23
5.5 Turn	23
6 Strings	24
6.1 Hash	24
6.2 Prefix Function	24
6.3 KMP	24
6.4 Rabin Karp	24
6.5 Aplicaciones de KMP y Rabin	25
6.6 Unique Substrings	25

6.7 Group Identical Strings	25
7 Tricks With Bits	26
8 Plantilla	27

1 Basic

1.1 Binary Search

```

1 def _binsearch(lista, x, l, r):
2     """Recursive binsearch between l and r index in lista"""
3     #Couldn't find x in lista
4     if(r<l): return -1
5     mid = (l+r)//2
6     if(lista[mid] == x): return mid
7     return _binsearch(lista, x, l, mid-1) if(x<lista[mid]) else
8         _binsearch(lista, x, mid+1, r)
9
10 def binsearch(l, element):
11     """Wrapper of recursive binsearch"""
12     return _binsearch(l, element, 0, len(l)-1)

```

1.2 LIS

1.2.1 Naive

```

1 # O(n^2) LIS
2
3 def LIS(l):
4     """Return the lenght of Longest Increasing Subsequence in l"""
5     dp = [1 for _ in range(len(l))] #dp[i]: lenght of LIS containing l[i]
6     ] as its last element
7
8     for i in range(len(l)):
9         for j in range(i):
10             if(l[j]<l[i]):
11                 dp[i] = max(dp[i], dp[j]+1)
12
13     return max(dp)

```

1.2.2 Fast

```

1 from bisect import bisect_right
2 import heapq
3 def find_gt(a, x):
4     'Find leftmost value greater than x'
5     i = bisect_right(a, x)
6     return i if i != len(a) else -1
7

```

```

8 def LIS(arr):
9     piles = []
10
11     for idx in range(len(arr)):
12         aux = [piles[i][-1] for i in range(len(piles))]
13         pileNumber = find_gt(aux, arr[idx])
14         if(pileNumber == -1):
15             piles.append([arr[idx]])
16         else:
17             piles[pileNumber].append(arr[idx])
18
19     print(piles)
20     aux = [piles[i][:-1] for i in range(len(piles))]
21     orden = list(heapq.merge(*aux))
22     print(orden)

```

2 Data Structures

2.1 DSU

```

1 class DisjointSetUnion:
2     def __init__(self, array):
3         self.parent = [i for i in array]
4         self.rank = [0 for _ in range(len(array))]
5         self.size = [1] * (len(array) + 1)
6         self.group = [[a[i]] for i in array]
7
8     def find(self, x):
9         # If x is root
10        if self.parent[x] == x:
11            return x
12        # If x is not root, search again by using x's parent
13        else:
14            self.parent[x] = self.find(self.parent[x])
15            return self.parent[x]
16
17    def union(self, x, y):
18        x = self.find(x)
19        y = self.find(y)
20        # Make an edge from the root of lower tree to the root of higher
21        # tree
22        if self.rank[x] < self.rank[y]:
23            self.parent[x] = y
24            self.size[y] += self.size[x]
25        else:
26            self.parent[y] = x
27            self.size[x] += self.size[y]
28            # If the height of tree the tree is the same, increase one
29            # of the heights by 1
30            if self.rank[x] == self.rank[y]:
31                self.rank[x] += 1
32
33    def merge(self, x, y):
34        x = self.find(x)
35        y = self.find(y)
36        if len(self.group[x]) < len(self.group[y]):
37            x, y = y, x
38        self.group[x].extend(self.group[y])
39        self.group[y] = []

```

```

38     self.parent[y] = x
39
40     def check_same(self, x, y):
41         return self.find(x) == self.find(y)
42
43     def get_size(self, x):
44         return self.size[self.find(x)]

```

2.2 FenwickTree

```

1 class FenwickTreeSum:
2     """ BIT de Sumas,
3     Resuelve las queries de intervalos y modificacion del array original
4     en O(log n). """
5     def __init__(self, n):
6         self.bit = [0]*n #Binary Indexed Tree
7         self.n = n
8
9     def initArray(self, array):
10         for i in range(len(array)):
11             self.update(i, array[i])
12
13     def sum(self, r):
14         ret = 0
15         while(r>=0):
16             ret += self.bit[r]
17             r = (r&(r+1))-1
18
19         return ret
20
21     def rangeSum(self, l, r):
22         return self.sum(r)-self.sum(l-1)
23
24     def update(self, idx, delta): #Add delta to a[idx]
25         while(idx<self.n):
26             self.bit[idx] += delta
27             idx |= (idx+1)
28
29
30
31 from math import inf
32 class FenwickTreeMin:
33     """ BIT de Min,

```

```

34     Resuelve las queries de intervalos y modificacion del array original
35     en O(log n). """
36     def __init__(self, n):
37         self.bit = [inf]*n #Binary Indexed Tree
38         self.n = n
39
40     def initArray(self, array):
41         for i in range(len(array)):
42             self.update(i, array[i])
43
44     def getMin(self, r):
45         ret = inf
46         while(r>=0):
47             ret = min(ret, self.bit[r])
48             r = (r&(r+1))-1
49
50         return ret
51
52
53     def update(self, idx, val):
54         while(idx<self.n):
55             self.bit[idx] = min(self.bit[idx], val)
56             idx |= (idx+1)
57
58 """Revisar
59 class FenwickTreeSum2D:
60     def __init__(self, n, m):
61         self.bit = [[0]*(m+1) for _ in range(n+1)] #Binary Indexed Tree
62         2D
63         self.n = n
64         self.m = m
65
66     def initArray(self, array):
67         aux = [[0]*(self.m+1) for _ in range(self.n+1)]
68         for i in range(1, self.n+1):
69             for j in range(1, self.m+1):
70                 aux[i][j] = array[self.n-j][i-1]
71             #It's a matrix now
72             for j in range(1, self.m+1):
73                 for i in range(1, self.n+1):
74                     v1 = self.getSum(i, j)
75                     v2 = self.getSum(i, j-1)
76                     v3 = self.getSum(i-1, j-1)

```

```

76         v4 = self.getSum(i-1,j)
77         self.update(i,j,aux[i][j]-(v1-v2-v4+v3))
78
79     def getSum(self, i, j):
80         suma = 0
81         while(i>0):
82             while(j>0):
83                 suma += self.bit[i][j]
84                 j = (j & (j+1))-1
85             i = (i & (i+1))-1
86         return suma
87
88
89     def update( self, i, j, val):
90         while( i <= self.n ):
91             while(j <= self.m):
92                 self.bit[i][j] += val
93                 j |= (j+1)
94             i |= (i+1)
95
96     def answerQuery(self,i1,j1, i2, j2):
97         ans = self.getSum(i2+1, j2+1)-self.getSum(i2+1, j1)-self.getSum(
98             i1, j2+1)+self.getSum(i1, j1)
99
100     return ans
"""

```

2.3 Segment Tree

```

1 class SegmentTree:
2     """Segment tree of sums"""
3
4     def __init__(self, array):
5         self.n = len(array)
6         self.t = [0]*self.n + array
7
8         for i in range(self.n-1, 0, -1):
9             self.t[i] = self.t[i<<1] + self.t[(i<<1)|1]
10
11     def modify(self, idx, val):
12         idx+=self.n; self.t[idx] = val
13         while(idx>1):
14             self.t[idx>>1] = self.t[idx] + self.t[idx^1]

```

```

15         idx >>=1
16
17     def modifyInterval(self, l, r, value):
18         """Modifica intervalo [l,r] poniendo value"""
19         l+=self.n; r+=self.n
20         while(l<r):
21             if(l&1):
22                 l+=1; self.t[l] += value
23             if(r&1):
24                 r-=1; self.t[r] += value
25             l>>=1; r>>=1
26
27     def push(self):
28         """Si necesitamos inspeccionar todos los elementos del array,
29         es conveniente pushear la info a las hojas, reduce O(nlogn) a O(
30             n)
31         """
32         for i in range(1,self.n):
33             self.t[i<<1] += self.t[i]
34             self.t[(i<<1)|1] += self.t[i]
35             self.t[i] = 0
36
37     def query(self, l, r):
38         """Responde al intervalo [l,r]"""
39         res = 0 #Se usa el neutro de la operacion
40         l+=self.n ; r+=self.n
41
42         while(l<r):
43             if(l&1):
44                 res += self.t[l]
45                 l+=1
46             if(r&1):
47                 r-=1
48                 res += self.t[r]
49             l>>=1; r>>=1
50
51         return res
52
53     def queryElement(self, p):
54         """Devuelve el valor de un elemento"""
55         res = 0 ; p+= self.n
56         while(p>0):
57             res+= self.t[p]

```

```

57         p>=1
58         return res
59
60
61 a = SegmentTree([1,2,3,4])
62 a.modifyInterval(0,3,10)
63 print(a.queryElement(3))
64 print(a.t)
65
66 class SegmentTreeGeneric:
67     """Generic Segment Tree. f es una funcion asociativa"""
68
69     def __init__(self, array, f):
70         self.n = len(array)
71         self.t = [0]*self.n + array
72         self.f = f #Me guardo la funcion aca para los otros metodos
73
74         for i in range(self.n-1, 0, -1):
75             self.t[i] = self.f(self.t[i<<1],self.t[(i<<1)|1])
76
77     def modify(self, idx, val):
78         idx+=self.n; self.t[idx] = val
79         while(idx>1):
80             self.t[idx>>1] = self.f(self.t[idx] + self.t[idx^1])
81             idx >>=1
82
83     def query(self, l, r):
84         res = 0 #Se usa el neutro de la operacion
85         l+=self.n ; r+=self.n
86
87         while(l<r):
88             if(l&1):
89                 res = self.t[l] if res==0 else self.f(res,self.t[l])
90                 l+=1
91             if(r&1):
92                 r-=1
93                 res = self.t[r] if res==0 else self.f(self.t[r],res)
94             l>>=1; r>>=1
95
96         return res

```

2.4 Sparse Table

```

1 from math import log
2
3 MAXN = 10**7 #Biggest possible array lenght
4 K = 25 # Must satisfy K >= floor(log_2{MAXN})+1
5
6 """Generic precomputation"""
7 def precomputation(array, f):
8     n = len(array)
9     K = int(log(n,2))+1
10    st = [[None for __ in range(K)] for _ in range(n)]
11
12    for i in range(n):
13        st[i][0] = f([array[i]])
14    for j in range(1,K+1):
15        for i in range(n-(1<<j)+1):
16            st[i][j] = f([st[i][j-1], st[i+(1<<(j-1))][j-1]])
17
18    return st
19
20 """Range Sum Queries"""
21 array = [1, 4, -1, 6, 9]
22 n = len(array)
23 K = int(log(n,2))+1
24 st = precomputation(array, sum)
25
26 def rangeSumQuery(L, R):
27     sum = 0
28     for j in range(K,-1,-1):
29         if((1<<j) <= R-L+1):
30             sum+= st[L][j]
31             L += 1<<j
32
33     return sum
34
35
36 """Range Minimun Queries (RMQ)"""
37 def precomputeLogs(n):
38     logs = {1:0}
39     for i in range(2,n+1):
40         logs[i] = logs[i//2] + 1
41
42     return logs
43

```

```
44 array = [1, 4, -1, 6, 9]
45 n = len(array)
46 K = int(log(n,2))+1
47 st = precomputation(array, min)
48 logs = precomputeLogs(n)
49
50 def rangeMinimumQuery( L, R):
51     j = logs[R-L+1]
52     return min(st[L][j], st[R-(1<<j)+1][j])
```

3 Graph

3.1 BFS

```
1 from collections import deque
2
3 empty_queue = lambda q: len(q)==0
4
5 def bfs(graph, root, visited = set()):
6     travel = []
7     q = deque([])
8
9     q.append(root)
10    visited.add(root)
11
12    while( not empty_queue(q) ):
13
14        actNode = q.popleft() #Dequeue
15        travel.append(actNode)
16        for adjacent in graph[actNode]:
17            if(adjacent not in visited):
18                q.append(adjacent) #Enqueue adjacent
19                visited.add(adjacent)
20
21    return travel
```

3.2 DFS

```
1 def rec_dfs(graph , node , visited):
2
3     if node not in visited:
4         visited.append(node)
5
6     for ady in graph[node]:
7         dfs_rec(graph , ady , visited)
8
9
10 def dfs(graph , node):
11     visited = []
12     rec_dfs(graph , node , visited)
13     print(visited)
```

3.3 Dijkstra

```

1 class Graph():
2
3     def __init__(self, vertices):
4         self.V = vertices
5         self.graph = [[0 for column in range(vertices)]
6                       for row in range(vertices)]
7
8     def printSolution(self, dist):
9         print "Vertex\tDistance from Source"
10        for node in range(self.V):
11            print node, "t", dist[node]
12
13    def minDistance(self, dist, sptSet):
14
15        min = float('inf')
16
17        for v in range(self.V):
18            if dist[v] < min and sptSet[v] == False:
19                min = dist[v]
20                min_index = v
21
22        return min_index
23
24    def dijkstra(self, src):
25
26        dist = [sys.maxint] * self.V
27        dist[src] = 0
28        sptSet = [False] * self.V
29
30        for cout in range(self.V):
31            u = self.minDistance(dist, sptSet)
32            sptSet[u] = True
33
34            for v in range(self.V):
35                if self.graph[u][v] > 0 and sptSet[v] == False and \
36                    dist[v] > dist[u] + self.graph[u][v]:
37                    dist[v] = dist[u] + self.graph[u][v]
38
39        self.printSolution(dist)

```

3.4 Floyd-Warshall

```

1 def floydWarshall(adyMatrix):

```

```

2     "APSP_problem, O(V^3)"
3     v = len(adyMatrix)
4     for k in range(v):
5         for i in range(v):
6             for j in range(v):
7                 adyMatrix[i][j] = min(adyMatrix[i][j], adyMatrix[i][k] + adyMatrix[k][j])
8
9
10    def transitiveClosure(d):
11        """d[i][j] tiene 1 si hay un camino entre i y j,
12           0 en caso contrario, O(V^3)"""
13        v = len(d)
14        for k in range(v):
15            for i in range(v):
16                for j in range(v):
17                    d[i][j] |= (d[i][k] & d[k][j])
18
19
20    def minimax(d):
21        """Encuentra el minimo entre los maximos edges de cada path"""
22        v = len(d)
23        for k in range(v):
24            for i in range(v):
25                for j in range(v):
26                    d[i][j] = min(d[i][k], max(d[i][k], d[k][j]))
27
28    def maximin(d):
29        v = len(d)
30        for k in range(v):
31            for i in range(v):
32                for j in range(v):
33                    d[i][j] = max(d[i][j], min(d[i][k], d[k][j]))

```

3.5 Max Flow (Ford Fulkerson)

```

1 from collections import deque
2 from math import inf
3 class Graph:
4     def __init__(self, graph):
5         self.graph = graph
6         self.ROW = len(graph)
7

```



```

8 def BFS(self,s,t,parent):
9     visited = [False]*(self.ROW)
10    queue = deque([])
11    queue.append(s); visited[s]=True
12    while queue:
13        u = queue.popleft()
14        for ind, val in enumerate(self.graph[u]):
15            if(visited[ind]==False and val>0):
16                queue.append(ind)
17                visited[ind]=True
18                parent[ind] = u
19    return visited[t]

21 def FordFulkerson(self, source, sink):
22     parent = [-1]*(self.ROW)
23     max_flow = 0
24     while self.BFS(source,sink,parent):
25         path_flow = inf
26         s = sink
27         while(s!=source):
28             path_flow = min(path_flow,self.graph[parent[s]][s])
29             s = parent[s]
30         max_flow += path_flow
31         v = sink
32         while(v!= source):
33             u = parent[v]
34             self.graph[u][v] -= path_flow
35             self.graph[v][u] += path_flow
36             v = parent[v]
37     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)

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

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

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

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

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

4.2 GCD

```

1 gcd = lambda a, b : a if(b==0) else gcd(b,a%b)
2
3 def it_gcd(a, b):
4     while(b):
5         a%=b
6         a, b = b, a #Swap para tener el mas chico en b
7     return a

```

4.3 Euler Totient (Phi)

```

1 def phi(n):
2     """O(sqrt(n)) approach using factorization"""
3     result = n
4     i=2
5     while(i*i<=n):
6         if(n%i==0):
7             while(n%i == 0):
8                 n/=i
9             result -= result//i
10        i+=1
11    if(n>1):
12        result-= result//n
13
14    return result

```

4.4 Extended Euclides

```

1 def gcd(a, b):
2     """Devuelve el gcd entre a y b, y coefx y coe fy tales
3     que a*coefx+b*coe fy = gcd"""
4     if(a==0):
5         return b, 0, 1
6     d, x1, y1 = gcd(b%a, a)
7     x = y1-(b//a)*x1
8     y = x1
9     return d, x, y
10
11
12 def modinv(a, m):
13     g, x, y = gcd(a,m)
14     if(g!=1):
15         return None
16     x = (x%m + m)%m
17     return x

```

4.5 Modexp

```

1 def modexp( x, y, p ):
2     """Exponenciacion logaritmica iterativa,
3     x^y (mod p), el orden el O(log y)"""
4
5     res = 1
6     while(y>0):
7         if(y & 1):
8             res*= x
9             res%=p
10        y >>= 1
11        x*= x
12
13    return res%p
14
15
16
17 """Inverso si m es primo"""
18 modinv = lambda a, m : modexp(a, m-2, m)

```

4.6 allModInv

```

1 """Find all invmods in range [1,m-1] in O(m)"""
2 def allModInvs(m):
3     inv = [1]*(m) #Remember that inv[i] has the inverse of i

```

```

4     inv[0] = None
5     for i in range(2,m):
6         inv[i] = -(m//i)*inv[m%i] %m
7
8     return inv

```

4.7 Iter multiplication

```

1 def logmul(a, b):
2     """No creo que sea necesario en python, pero version recursiva
3     de la multiplicacion que sirve en C++"""
4     if(a == 0):
5         return 0
6     return 2*logmul((a-1)//2,b)+b if(a%2) else 2*logmul(a//2,b)
7
8
9 def logmulmod(a, b, p):
10    """Multiplicacion recursiva mod p"""
11    if(a == 0):
12        return 0
13
14    res = 2*logmulmod((a-1)//2,b, p)+b if(a%2) else 2*logmulmod(a//2,b,
15    p)
16    return res%p

```

4.8 Test de primalidad y Descomposicion en Primos

4.8.1 Is Prime

```

1 def is_prime(n):
2     """Naive O(sqrt(n)) approach"""
3     d = 2
4     while(d*d<=n):
5         if(x%d == 0):
6             return False
7         d+=1
8     return True
9
10 from random import randint
11
12 RAND_MAX = 10**9
13 def modexp( x, y, p ):
14     res = 1
15     while(y>0):

```

```

16     if(y & 1):
17         res*= x
18         res%=p
19     y >>= 1
20     x*= x
21
22     return res%p
23
24 def probablyPrimeFermat(n, iter=5):
25     if (n < 4):
26         return n == 2 or n == 3
27
28     for i in range(iter):
29         a = 2 + randint(1, RAND_MAX)%(n - 3);
30         if (modexp(a, n - 1, n)!=1):
31             return False
32
33     return True
34
35 for i in range(100):
36     print(i,probablyPrimeFermat(i,5))

```

4.8.2 Miller Rabin Test

```

1 from random import randint
2 RAND_MAX = 10**9
3 def modexp( x, y, p ):
4     res = 1
5     while(y>0):
6         if(y & 1):
7             res*= x
8             res%=p
9         y >>= 1
10        x*= x
11
12    return res%p
13
14 def check_composite( n, a, d, s):
15     x = modexp(a, d, n)
16     if (x == 1 or x == n - 1):
17         return False
18     for r in range(1,s):
19         x = x * x % n;

```

```

20         if (x == n - 1):
21             return False
22
23     return True
24
25 def MillerRabinProb(n, iter = 5):
26     """Returns true if n is probably prime, else returns false."""
27     if (n < 4):
28         return n == 2 or n == 3
29
30     s = 0;
31     d = n - 1;
32     while ((d & 1) == 0):
33         d >>= 1;
34         s+=1
35
36     for i in range(iter):
37         a = 2 + randint(1,RAND_MAX) % (n - 3);
38         if(check_composite(n, a, d, s)):
39             return False
40     return True
41
42
43 """Deterministic Version"""
44 def MillerRabin(n):
45     if(n<2):
46         return False
47
48     r = 0
49     d = n - 1
50     while ((d & 1) == 0):
51         d >>= 1
52         r+=1
53
54     for a in {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}:
55         if (n == a):
56             return True
57         if(check_composite(n, a, d, r)):
58             return False
59     return True
60 cnt = 0
61 for i in range(100000):
62     if(MillerRabin(i)):

```

```

63     cnt+=1
64     print(cnt)

```

4.8.3 Prime Factors

```

1  # Program to fiind prime factors and their powers
2  # using Sieve Of Eratosthenes
3
4  def sieveOfEratosthenes(N, s):
5      """Sieve with smaller prime factor of s[i]"""
6
7      prime = [False] * (N+1)
8
9      for i in range(2, N+1, 2):
10         s[i] = 2
11
12     for i in range(3, N+1, 2):
13         if (prime[i] == False):
14             s[i] = i
15
16         for j in range(i, N//i + 1, 2):
17             if (prime[i*j] == False):
18                 prime[i*j] = True
19                 s[i * j] = i
20
21
22 def generatePrimeFactors(N):
23
24     #s[i]: smallest prime factor
25     s = [0] * (N+1)
26     sieveOfEratosthenes(N, s)
27
28     factors = [] #Contains tuples (p_i, alpha_i)
29
30     curr = s[N]
31     # Power of current prime factor
32     cnt = 1
33
34     while (N > 1):
35         N //= s[N]
36
37         if (curr == s[N]):
38             cnt += 1

```

```

39         continue
40
41         factors.append((curr,cnt))
42         curr = s[N]
43         cnt = 1
44     return factors

```

4.8.4 Integer Fact

```

1  """Naive O(sqrt(n)) approach"""
2  def trial_division1(n):
3      factorization = []
4      d = 2
5      while(d*d<=n):
6          while(n%d == 0):
7              factorization.append(d)
8              n//=d
9              d+=1
10
11     if(n>1):
12         factorization.append(n)
13     return factorization
14
15     """Don't try even numbers if it's odd. (Wheel factorization)"""
16 def trial_division2(n):
17     factorization = []
18     while(n%2 == 0):
19         factorization.append(2)
20         n >>= 1
21
22     d = 3
23     while(d*d<=n):
24         while(n%d==0):
25             factorization.append(d)
26             n //= d
27             d+=2
28     if(n>1):
29         factorization.append(n)
30     return factorization
31
32     """Same Wheel idea, but with factors 2,3,5 at the same time."""
33 def trial_division3(n):
34     factorization = []

```

```

35     for d in {2,3,5}:
36         while(n%d==0):
37             factorization.append(d)
38             n //= d
39
40     increments = [4, 2, 4, 2, 4, 6, 2, 6]
41     i = 0
42     d = 7
43     while(d*d<=n):
44         while(n%d == 0):
45             factorization.append(d)
46             n //= d
47             i %= 8
48             d+=increments[i]
49
50     if(n>1):
51         factorization.append(n)
52     return factorization
53
54     """Precomputed primes"""
55
56
57
58
59 def trial_division4(n):
60     """Fast way to find primes"""
61     MAX_PRIME = 10**6
62     def listPrimesFast(max_n):
63         """sundaram3"""
64         numbers = [i for i in range(3, max_n+1, 2)]
65         half = (max_n)//2
66         initial = 4
67
68         for step in range(3, max_n+1, 2):
69             for i in range(initial, half, step):
70                 numbers[i-1] = 0
71                 initial += 2*(step+1)
72
73         if initial > half:
74             return [2] + filter(None, numbers)
75
76     primes = listPrimesFast(MAX_PRIME)
77     factorization = []

```

```

78     for d in primes:
79         if(d*d>n):
80             break
81         while(n%d == 0):
82             factorization.append(d)
83             n//=d
84     if(n>1):
85         factorization.append(n)
86
87     return factorization
88
89
90     """Pollard's p-1 method"""
91     from random import randint
92     from math import gcd
93     MAX RAND = 10**9
94     def pollard_p_minus_1(n):
95         """Probabilistic method, O(BlogB log^2 n), encuentra un primo
96         que divide a n"""
97         MAX_PRIME = 10**6
98         def listPrimesFast(max_n):
99             """sundaram3"""
100             numbers = [i for i in range(3, max_n+1, 2)]
101             half = (max_n)//2
102             initial = 4
103
104             for step in range(3, max_n+1, 2):
105                 for i in range(initial, half, step):
106                     numbers[i-1] = 0
107                     initial += 2*(step+1)
108
109             if initial > half:
110                 return [2] + list(filter(None, numbers))
111
112     def modexp( x, y, p ):
113         res = 1
114         while(y>0):
115             if(y & 1):
116                 res*= x
117                 res%=p
118             y >>= 1
119             x*= x
120

```

```

121     return res%p
122
123     B = 10
124     g = 1
125     primes = listPrimesFast(MAX_PRIME)
126     while(B <= 10**7 and g<n):
127         a = 2+ randint(1,MAX_RAND) % (n-3)
128         g = gcd(a,n)
129         if(g>1):
130             return g
131
132     #Computo a^M
133     for p in primes:
134         if(p>=B):
135             continue
136         p_power = 1
137         while(p_power * p <= B):
138             p_power *= p
139         a = modexp(a, p_power, n)
140
141         g = gcd(a-1, n)
142         if(g>1 and g<n):
143             return g
144     B*=2
145
146     return 1
147
148
149     """Pollard Rho algorithm to find a factor of n"""
150     from math import gcd
151     def floyd(f, x0):
152         tortoise = x0
153         hare = f(x0)
154         while(tortoise != hare):
155             tortoise = f(tortoise)
156             hare = f(f(hare))
157
158         return True
159
160     mult = lambda a, b, mod: a*b % mod
161     f = lambda x, c, mod: (mult(x,x,mod)+c)%mod
162
163     def rho(n, x0=2, c=1):

```

```

164     x, y, g = x0, x0, 1
165     while(g==1):
166         x = f(x, c, n)
167         y = f(y, c, n)
168         y = f(y, c, n)
169         g = gcd(abs(x-y), n)
170
171     return g
172
173
174     """Brent, direct implementation"""
175     from math import gcd
176     mult = lambda a, b, mod: a*b % mod
177     f = lambda x, c, mod: (mult(x,x,mod)+c)%mod
178
179     def brent(n, x0 = 2, c=1):
180         x, g, q= x0, 1, 1
181         m = 128
182         l = 1
183         while(g==1):
184             y = x
185             for i in range(1,l):
186                 x = f(x, c, n)
187             k = 0
188
189             while(k<l and g==1):
190                 xs = x
191                 i = 0
192                 while(i<m and i<l-k):
193                     x = f(x, c, n)
194                     q = mult(q, abs(y-x), n)
195                     i+=1
196                 g = gcd(q, n)
197                 k+=m
198             l*=2
199
200         if(g==n):
201             while True:
202                 xs = f(xs, c, n)
203                 g = gcd(abs(xs-y), n)
204                 if(g!=1 and g!=n):
205                     break
206         return g

```

4.9 Diofanticas

```

1 def gcd(a, b):
2     """Devuelve el gcd entre a y b, y coefx y coefy tales
3     que a*coefx+b*coefy = gcd"""
4     if(a==0):
5         return b, 0, 1
6     g, x1, y1 = gcd(b%a, a)
7     x = y1-(b//a)*x1
8     y = x1
9     return g, x, y
10
11 def find_any_solution(a, b, c):
12     """Returns g = gcd(a,b) and a pair of coef x0, y0 such that
13     a*x0+b*y0 = g"""
14     g, x0, y0 = gcd(abs(a), abs(b))
15     if (c % g):
16         return False
17
18     x0 *= c // g;
19     y0 *= c // g;
20     if (a < 0): x0 = -x0;
21     if (b < 0): y0 = -y0;
22     return g, x0, y0
23
24
25 def shift_solution(x, y, a, b, cnt):
26     """Dada una solucion a la ecuacion diofantica, encuentra otra"""
27     x += cnt * b;
28     y -= cnt * a;
29     return x, y
30
31
32 def find_all_solutions( a, b, c, minx, maxx, miny, maxy):
33     sol = find_any_solution(a, b, c)
34     if (not sol):
35         return 0
36     g, x, y = sol
37     a //= g
38     b //= g
39
40     sign_a = 1 if(a > 0) else -1
41     sign_b = 1 if(b > 0) else -1

```

```

42
43     x, y = shift_solution(x, y, a, b, (minx - x) // b)
44     if (x < minx):
45         x, y = shift_solution(x, y, a, b, sign_b)
46     if (x > maxx):
47         return 0
48     lx1 = x
49
50     x, y = shift_solution(x, y, a, b, (maxx - x) // b);
51     if (x > maxx):
52         x, y = shift_solution(x, y, a, b, -sign_b)
53     rx1 = x
54
55     x, y = shift_solution(x, y, a, b, -(miny - y) // a)
56     if (y < miny):
57         x, y = shift_solution(x, y, a, b, -sign_a)
58     if (y > maxy):
59         return 0
60     lx2 = x
61
62     x, y = shift_solution(x, y, a, b, -(maxy - y) // a)
63     if (y > maxy):
64         shift_solution(x, y, a, b, sign_a)
65     rx2 = x
66
67     if (lx2 > rx2):
68         lx2, rx2 = rx2, lx2
69     lx = max(lx1, lx2);
70     rx = min(rx1, rx2);
71
72     if (lx > rx):
73         return 0
74
75     # Para enumerarlas, basta iterar desde x = lx+ k*b//g, el numero
76     # necesario de soluciones
77
78     return (rx - lx) // abs(b) + 1;

```

4.10 Cribas

4.10.1 Criba Primos

```

1 """Criba sin optimizar, O(n* log log n)"""
2

```

```

3 n = 10**6
4 is_prime = [True]*(n+1)
5
6 is_prime[0] = False
7 is_prime[1] = False
8
9 for i in range(2,n+1):
10     if(is_prime[i] and i*i<=n):
11         for j in range(i*i,n+1,i):
12             is_prime[j] = False
13
14 """Sieving till root"""
15
16 n = 10**6
17 is_prime = [True]*(n+1)
18
19 is_prime[0] = False
20 is_prime[1] = False
21
22 for i in range(2,n+1):
23     if(is_prime[i]):
24         for j in range(i*i,n+1,i):
25             is_prime[j] = False
26

```

4.10.2 Block Sieving

```

1 """Cuento los primos menores a n usando block_sieving"""
2
3 def count_primes(n):
4     S = 10000
5     primes = []
6     nsqrt = int(n**0.5)
7     is_prime = [True]*(nsqrt+1)
8
9     for i in range(2, nsqrt+1):
10         if(is_prime[i]):
11             primes.append(i)
12             for j in range(i*i,nsqrt+1,i):
13                 is_prime[j] = False
14
15     result = 0
16     block = [True]*S

```

```

17
18 k = 0
19 while(k*S <= n):
20     block = [True]*S
21     start = k*S
22
23     for p in primes:
24         start_idx = (start+p-1)//p
25         j = max(start_idx, p) * p - start
26         while(j<S):
27             block[j] = False
28             j+=p
29
30     if( k == 0):
31         block[0] = False
32         block[1] = False
33     i = 0
34     while(i<S and start+i <= n):
35         if(block[i]):
36             result+=1
37             i+=1
38         k+=1
39
40     return result

```

4.10.3 Linear Sieve

```

1 N = 10**6
2 lp = [0]*(N+1)
3 pr = []
4
5 import timeit
6 start = timeit.default_timer()
7 for i in range(2,N+1):
8     if(lp[i]==0):
9         lp[i]=i
10        pr.append(i)
11
12    j = 0
13    while(j<len(pr) and pr[j]<=lp[i] and i*pr[j]<=N):
14        lp[i*pr[j]] = pr[j]
15        j+=1
16    end = timeit.default_timer()

```



```

17 print(end-start)
18
19 print(lp[:20])
20 print(pr[:20])

```

4.11 Divisores

4.11.1 Count Divisors

```

1 def countDivisors(n):
2     """Naive implementation to count divisors of n, in O(sqrt(n))"""
3     cnt = 0
4     for i in range(1, int(n**0.5)+1):
5         if (n % i == 0):
6             cnt += 1 if(i*i == n) else 2
7
8     return cnt
9
10
11
12 """Improved method in O(n^(1/3))"""
13 def SieveOfEratosthenes(n, prime, primesquare, a):
14     """Sieve of primes and squares of primes."""
15     for i in range(2, n+1):
16         prime[i] = True
17
18     for i in range((n * n + 1)+1):
19         primesquare[i] = False
20
21     prime[1] = False
22
23     p = 2
24     while(p * p <= n):
25         if (prime[p] == True):
26             i = p * 2
27             while(i <= n):
28                 prime[i] = False
29                 i += p
30
31             p+=1
32
33     j = 0
34     for p in range(2, n+1):
35         if (prime[p]==True):
36             a[j] = p

```

```

36
37         primesquare[p * p] = True
38         j+=1
39
40 def countDivisorsFast(n):
41     """Count divisors in O(n^(1/3))"""
42     if (n == 1):
43         return 1
44
45     prime = [False]*(n + 2)
46     primesquare = [False]*(n * n + 2)
47
48     a = [0]*n
49
50     SieveOfEratosthenes(n, prime, primesquare, a)
51
52     ans = 1
53
54     i=0
55     while(1):
56         if(a[i]**3 > n):
57             break
58
59         cnt = 1
60         while (n % a[i] == 0):
61             n //= a[i]
62             cnt += 1
63         ans *= cnt
64         i+=1
65
66     if(prime[n]==True):
67         ans *= 2
68
69     elif(primesquare[n]==True):
70         ans *= 3
71
72     elif(n != 1):
73         ans *= 4
74
75     return ans

```

4.11.2 Suma Divisores

```

1 # Sum of all divisors of n.
2 def sumofFactors(n):
3     res = 1
4     for i in range(2, int(n**0.5 + 1)):
5         curr_sum = 1
6         curr_term = 1
7
8         while n % i == 0:
9             n //= i
10            curr_term *= i
11            curr_sum += curr_term;
12
13        res *= curr_sum
14
15    if n > 2:
16        res *= (1 + n)
17
18    return res

```

4.12 Factorial

4.12.1 Factorial mod

```

1 def factmod( n, p ):
2     """0(p log n ) aproach to calculate n! (mod p)"""
3     if(n>p):
4         return 0
5     res = 1
6     while(n > 1):
7         res = (res* (p-1 if((n//p)%2) else 1)) % p;
8         for i in range(2, n%p+1):
9             res = (res * i) % p
10        n //= p
11
12    return res % p;

```

4.12.2 Factorial Divisors

```

1 # Count of divisors of n!
2
3 allPrimes = []
4 def sieve(n):
5     prime = [True] * (n + 1)
6

```

```

7     p = 2
8     while(p * p <= n):
9         if (prime[p] == True):
10            i = p * 2
11            while(i <= n):
12                prime[i] = False
13                i += p
14            p += 1
15
16    for p in range(2, n + 1):
17        if (prime[p]):
18            allPrimes.append(p)
19
20    def factorialDivisors(n):
21
22        sieve(n)
23        result = 1
24
25        for i in range(len(allPrimes)):
26            p = allPrimes[i]
27            exp = 0
28            while (p <= n):
29                exp += n // p
30                p *= allPrimes[i]
31
32            result *= (exp + 1)
33
34    return result

```

4.13 FFT

```

1 from cmath import exp
2 from math import pi
3
4 # A simple class to simulate n-th root of unity
5 class NthRootOfUnity:
6     def __init__(self, n, k = 1):
7         self.k = k
8         self.n = n
9
10    def __pow__(self, other):
11        if type(other) is int:
12            n = NthRootOfUnity(self.n, self.k * other)

```

```

13         return n
14
15     def __eq__(self, other):
16         if other == 1:
17             return abs(self.n) == abs(self.k)
18
19     def __mul__(self, other):
20         return exp(2*1j*pi*self.k/self.n)*other
21
22     def __repr__(self):
23         return str(self.n) + "-th root of unity to the " + str(self.k)
24
25     @property
26     def th(self):
27         return abs(self.n // self.k)
28
29 """ The Fast Fourier Transform Algorithm
30
31 Input: A, An array of integers of size n representing a polynomial
32       w , a n-root of unity
33 Output: [A(w), A(w^2), ..., A(w^(n-1))]
34 Complexity: O(n logn)"""
35
36 def FFT(A, omega):
37     if omega == 1:
38         return [sum(A)]
39     o2 = omega**2
40     C1 = FFT(A[0::2], o2)
41     C2 = FFT(A[1::2], o2)
42     C3 = [None]*omega.th
43     for i in range(omega.th//2):
44         C3[i] = C1[i] + omega**i * C2[i]
45         C3[i+omega.th//2] = C1[i] - omega**i * C2[i]
46     return C3
47
48 """ The Fast Polynomial Multiplication Algorithm
49
50 Input: A,B, two arrays of integers representing polynomials
51       (coef in increasing deg) their length is in O(n)
52 Output: Coefficient representation of AB
53 Complexity: O(n logn)"""
54
55 def FPM(A,B):
56     n = 1<<(len(A)+len(B)-2).bit_length()

```

```

56     o = NthRootOfUnity(n)
57     AT = FFT(A, o)
58     BT = FFT(B, o)
59     C = [AT[i]*BT[i] for i in range(n)]
60     D = [round((a/n).real) for a in FFT(C, o ** -1)]
61     while len(D) > 0 and D[-1] == 0:
62         del D[-1]
63     return D

```

4.14 Lagrange Interpolation

```

1 def interpolacion(muestra, x):
2     """Recibe una muestra de n puntos (tuplas (xi,yi))
3     y evalua el polinomio de Lagrange de
4     en el punto x desconocido"""
5     ans = 0
6     n = len(muestra)
7     for i in range(n):
8         term = muestra[i][1]
9         for j in range(n):
10             if(j!=i):
11                 term = term*(x-muestra[j][0])/(muestra[i][0]-muestra[j][0])
12         ans+=term
13     return ans

```

4.15 Discrete Log

```

1 def discreteLog(a, b, m):
2     n = int(m**0.5)+ 1
3     an = 1;
4     for i in range(n):
5         an = (an * a) % m
6
7     vals = {}
8     cur = an
9     for p in range(1,n+1):
10         if not cur in vals:
11             vals[cur] = p
12         cur = (cur * an) % m
13
14     cur = b
15     for q in range(n+1):
16         if cur in vals:
17             ans = vals[cur] * n - q

```

```

18         return ans
19         cur = (cur * a) % m
20
21     return -1

```

4.16 Discrete Root

```

1 from math import gcd
2
3 def modexp( x, y, p ):
4     """Exponenciacion logaritmica iterativa,
5     x^y (mod p), el orden es O(log y)"""
6
7     res = 1
8     while(y>0):
9         if(y & 1):
10             res*= x
11             res%=p
12             y >>= 1
13             x*= x
14
15     return res%p
16
17
18 # Finds the primitive root modulo p
19 def generator(p):
20     fact = []
21     phi = p-1
22     n = phi
23     i = 2
24     while(i*i<=n):
25         if(n % i == 0):
26             fact.append(i);
27             while (n % i == 0):
28                 n //= i
29             i+=1
30
31     if(n > 1): fact.push_back(n)
32
33     for res in range(2,p+1):
34         ok = True;
35         for factor in fact:
36             if(modexp(res, phi // factor, p) == 1):

```

```

37         ok = False
38         break
39     if(ok): return res
40     return -1;

```

4.17 Primitive Root

```

1 from math import sqrt
2
3 def isPrime(n):
4     if (n <= 1):
5         return False
6     if (n <= 3):
7         return True
8
9     if (n % 2 == 0 or n % 3 == 0):
10         return False
11     i = 5
12     while(i * i <= n):
13         if (n % i == 0 or n % (i + 2) == 0) :
14             return False
15         i = i + 6
16
17     return True
18
19 def modexp( x, y, p ):
20     res = 1
21     while(y>0):
22         if(y & 1):
23             res*= x
24             res%=p
25             y >>= 1
26             x*= x
27
28     return res%p
29
30
31 """O(log^6 p)"""
32 def findPrimefactors(s, n):
33     while (n % 2 == 0) :
34         s.add(2)
35         n //= 2
36

```

```

37     for i in range(3, int(sqrt(n)), 2):
38         while (n % i == 0):
39             s.add(i)
40             n //= i
41
42     if (n > 2):
43         s.add(n)
44
45 def findPrimitive(n):
46     s = set()
47
48     if (isPrime(n) == False):
49         return -1
50
51     phi = n-1
52     findPrimefactors(s, phi)
53
54     for r in range(2, phi + 1):
55         flag = False
56         for it in s:
57             if (modexp(r, phi // it, n) == 1):
58                 flag = True
59                 break
60         if (flag == False):
61             return r
62     return -1

```

4.18 Fibonacci

```

1 def matrix_multiplication( A, B):
2     """O(n^3) matrix multiplication"""
3     result = [[sum(a * b for a, b in zip(A_row, B_col))
4                 for B_col in zip(*B)]
5                 for A_row in A]
6
7     return result
8
9 def identity(n):
10     """Matriz identidad de dimension n"""
11     return [[1 if i==j else 0 for j in range(n)] for i in range(n)]
12
13 def matrix_exp( A, n ):
14     """Exponenciacion logaritmica iterativa de matrices,
    A^n, el orden es O(d^3*log n), con d la dimension de la

```

```

15     matriz"""
16
17     res = identity(len(A))
18     while(n>0):
19         if(n & 1):
20             res = matrix_multiplication(res,A)
21
22         n >>= 1
23         A = matrix_multiplication(A, A)
24
25     return res
26
27 def fibo_log(n):
28     """Returns f_n, using matrix log exponenciacion"""
29     f = [[0],[1]]
30     transition = [[0,1],[1,1]]
31     result = matrix_multiplication(matrix_exp(transition,n),f)
32     return result[0][0]
33
34     """Could lead to a WA, but O(1) solution (check if **n does not lead to
35     O(n))
36
37 def fibo_approx(n):
38     phi, phi_conj = (1+5**0.5)/2, (1-5**0.5)/2
39
40     return round((phi**n-phi_conj**n)/(5**0.5))
41
42     """Fast doubling Method"""
43 def fibo_fast_doubling(n):
44     if (n == 0):
45         return (0, 1)
46
47     p = fibo_fast_doubling(n >> 1)
48     c = p[0]*(2*p[1]- p[0])
49     d = p[0]**2 + p[1]**2
50     return (d, c + d) if(n & 1) else (c, d)

```

4.19 Gray Code

```

1 def g(n):
2     return n^(n>>1)
3

```

```

4 def rev_g(g):
5     n = 0
6     while(g):
7         n^=g
8         g>>=1
9     return n

```

4.20 Matrix

```

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

```

```

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

```

5 Geo

5.1 Convex Hull

```

1 from functools import reduce
2 def convex_hull_graham(points):
3     TURN_LEFT, TURN_RIGHT, COLLINEAR = (1, -1, 0)
4     def cmp(a, b):
5         return (a > b) - (a < b)
6
7     def turn(p, q, r):
8         return cmp((q[0] - p[0])*(r[1] - p[1]) - (r[0] - p[0])*(q[1] - p
9             [1]), 0)
10
11     def _keep_left(hull, r):
12         while len(hull) > 1 and turn(hull[-2], hull[-1], r) not in {
13             TURN_LEFT}:
14             #Agregar a {TURN_LEFT} el elemento COLLINEAR si se quieren
15             #alineados en
16             #En la frontera
17             hull.pop()
18             if not len(hull) or hull[-1] != r:
19                 hull.append(r)
20         return hull
21
22     points = sorted(points)
23     l = reduce(_keep_left, points, [])
24     u = reduce(_keep_left, reversed(points), [])
25     return l.extend(u[i] for i in range(1, len(u) - 1)) or l

```

5.2 Determinant

```

1 def determinant(puntos):
2     """Recibe una lista de puntos (tuplas 2D)
3     y calcula su -determinante-"""
4     result = 0
5     size = len(puntos)
6     for i in range(size):
7         x1, x2 = puntos[i][0], puntos[(i+1)%size][0]
8         y1, y2 = puntos[i][1], puntos[(i+1)%size][1]
9
10        result += (x1*y2-x2*y1)
11

```

```
12 | return result
```

5.3 GreatCircleDistance

```
1 | from math import *
2 |
3 | def greatCircleDistance( pLat, pLong, qLat, qLong, radius ):
4 |
5 |     pLat *= pi/180
6 |     pLong *= pi/180
7 |     qLat *= pi/180
8 |     qLong *= pi/180
9 |
10 |    return radius*acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(qLong)+
11 |                      cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong)+
12 |                      sin(pLat)*sin(qLat))
```

5.4 Perimeter

```
1 | from math import sqrt
2 | def perimeter(puntos):
3 |     """Recibe una lista de puntos 2D que representan
4 |     los vertices de un poligono y devuelve el perimetro
5 |     del mismo."""
6 |     result = 0
7 |     size = len(puntos)
8 |     for i in range(size):
9 |         x1, x2 = puntos[i][0], puntos[(i+1)%size][0]
10 |        y1, y2 = puntos[i][1], puntos[(i+1)%size][1]
11 |        dx, dy = x2-x1, y2-y1
12 |        result += sqrt(dx**2+dy**2)
13 |
14 |    return result
```

5.5 Turn

```
1 | def turn( p, q, r):
2 |     """Recibe tres puntos p, q y r (2D) y
3 |     devuelve si se encuentran en sentido horario,
4 |     antihorario o alineados"""
5 |
6 |     result = (r[0]-q[0])*(p[1]-q[1]) - (r[1]-q[1])*(p[0]-q[0])
7 |
8 |     if(result < 0):
```

```
9 |         return -1 # P->Q->R es una terna derecha (CCW)
10 |
11 |     if(result > 0):
12 |         return 1 # P->Q->R es una terna izquierda
13 |
14 |     return 0 # P->Q->R colineales
15 |
16 | # Wrapper para chequear directamente CCW, si se toleran colineales usar
17 | >=
17 | ccw = lambda p, q, r: turn(p, q, r)>0
```

6 Strings

6.1 Hash

```

1 def compute_hash(s):
2     p = 31 #Cambiar esto por un primo un poco mas grande
3     m = 10**9+9
4     p_pow = 1
5     hash_value = 0
6     for c in s:
7         hash_value = (hash_value + (ord(c)-ord('a')+1)*p_pow)%m
8         p_pow = (p_pow * p)%m
9     return hash_value

```

6.2 Prefix Function

```

1 def prefixFunction(s):
2     """Devuelve un array pi, donde pi[i]
3     coincide con el mayor prefijo propio que ademas
4     es sufixo de s[0...i]"""
5     n = len(s)
6     pi = [0 for _ in range(n)]
7
8     for i in range(1,n):
9         j = pi[i-1]
10        while(j>0 and s[i]!=s[j]):
11            j = pi[j-1]
12        if(s[i]==s[j]): j+=1
13        pi[i] = j
14
15    return pi

```

6.3 KMP

```

1 def prefixFunction(s):
2     """Devuelve un array pi, donde pi[i]
3     coincide con el mayor prefijo propio que ademas
4     es sufixo de s[0...i]"""
5     n = len(s)
6     pi = [0 for _ in range(n)]
7
8     for i in range(1,n):
9         j = pi[i-1]
10        while(j>0 and s[i]!=s[j]):

```

```

11        j = pi[j-1]
12        if(s[i]==s[j]): j+=1
13        pi[i] = j
14
15    return pi
16
17 def KMP(s,t):
18     """Encuentra todas las ocurrencias de s en t en
19     O(|s|+|t|)"""
20     n = len(s); m = len(t)
21     separator = "#"
22     #Elegir algun caracter que no este en s ni en t
23
24     pi = prefixFunction(s+separator+t)
25     occurences = []
26
27     for i in range(n+1,n+m+1):
28         if(pi[i]==n): occurences.append(i-2*n)
29     return occurences
30
31 print(prefixFunction("ab#abab"))
32 print(KMP("ab", "abab"))

```

6.4 Rabin Karp

```

1 def rabin_karp(s, t):
2     """O(|s|+|t|). Given a pattern s and a text t,
3     determine if the pattern appears in the text and if it does,
4     enumerate all its occurrences."""
5     p = 31
6     m = 10**9+9
7     S = len(s); T = len(t)
8     p_pow = [1 for _ in range(max(S,T))]
9     for i in range(1,len(p_pow)):
10        p_pow[i] = (p_pow[i-1]*p)%m
11    h = [0 for i in range(T+1)]
12    for i in range(T):
13        h[i+1] = (h[i]+ (ord(t[i])-ord('a')+1)*p_pow[i])%m
14    h_s = 0
15    for i in range(S):
16        h_s = (h_s + (ord(s[i])-ord('a')+1)*p_pow[i])%m
17
18    occurences = []

```



```

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

```

6.5 Aplicaciones de KMP y Rabin

```

1 def prefixFunction(s):
2     """Devuelve un array pi, donde pi[i]
3     coincide con el mayor prefijo propio que ademas
4     es sufijo de s[0...i]"""
5     n = len(s)
6     pi = [0 for _ in range(n)]
7
8     for i in range(1,n):
9         j = pi[i-1]
10        while(j>0 and s[i]!=s[j]):
11            j = pi[j-1]
12        if(s[i]==s[j]): j+=1
13        pi[i] = j
14
15    return pi
16
17 def numOccurencesPrefix(s):
18     """Cuenta el numero de apariciones de
19     cada prefijo de s en s"""
20     pi = prefixFunction(s)
21     n = len(s)
22     ans = [0 for i in range(n+1)]
23     for i in range(n):
24         ans[pi[i]]+=1
25     i = n-1
26     while(i>0):
27         ans[pi[i-1]] += ans[i]
28         i-=1
29     for i in range(n+1):
30         ans[i]+=1
31     return ans
32
33 def numDifferentSubstring(s):
34     """Dada una string s, cuenta la cantidad de
35     substrings diferentes que contiene. 0(|s|^2)"""

```

```

36 ans = 1; n = len(s)
37 for i in range(1,n):
38     pimax = max(prefixFunction(s[:i+1][::-1]))
39     ans += i+1-pimax
40 return ans

```

6.6 Unique Substrings

```

1 def countUniqueSubstrings(s):
2     n = len(s)
3     p = 31; m = 10**9+9
4     p_pow = [1 for _ in range(n)]
5
6     for i in range(1,n):
7         p_pow[i] = (p_pow[i-1]*p)%m
8
9     h = [0 for i in range(n+1)]
10    for i in range(n):
11        h[i+1] = (h[i]+(ord(s[i])-ord('a')+1) * p_pow[i])%m
12    cnt = 0
13    for l in range(1,n+1):
14        hs = set()
15        for i in range(n-l+1):
16            cur_h = (h[i+l]+m-h[i])%m
17            cur_h = (cur_h*p_pow[n-i-1]) % m
18            hs.add(cur_h)
19        cnt += len(hs)
20    return cnt
21
22 print(countUniqueSubstrings('abcaa'))

```

6.7 Group Identical Strings

```

1 def compute_hash(s):
2     p = 31 #Cambiar esto por un primo un poco mas grande
3     m = 10**9+9
4     p_pow = 1
5     hash_value = 0
6     for c in s:
7         hash_value = (hash_value + (ord(c)-ord('a')+1)*p_pow)%m
8         p_pow = (p_pow * p)%m
9     return hash_value
10
11 def groupIdenticalStrings(s):

```

```

12 """Recibo una lista de strings, devuelvo los grupos de strings
    identicas
13 0(n*m+n*log n) (n=len(s), m = maxlen entre las strings"""
14 n = len(s)
15 hashes = [None for i in range(n)]
16 for i in range(n):
17     hashes[i] = (compute_hash(s[i]),i)
18 hashes.sort()
19 groups = []
20 for i in range(n):
21     if(i==0 or hashes[i][0] != hashes[i-1][0]):
22         groups.append([])
23     groups[-1].append(hashes[i][1])
24 return groups
25
26 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 repeat length 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

```
1 I = lambda : int(input())
2 LI = lambda : [int(x) for x in input().split()]
3 MI = lambda : map(int, input().split())
4 SI = lambda : input()
5
6 """
7 #Leer de archivo
8 for line in sys.stdin:
9     ...
10 """
11
12 from collections import *
13 from heapq import *
14 from math import *
15 import itertools
```