# 復習済み

A-E

# A Pairing

正解

from collections import defaultdict

d = defaultdict(int)

A = input().split()

for a in A:

d[a] += 1

count = 0

for v in d.values():

if v >= 2:

count += v//2

print(count)

# B Garbage Collection

正解

N = int(input())

dash\_day = []

for \_ in range(N):

q, r = map(int, input().split())

dash\_day.append((q, r))

Q = int(input())

for \_ in range(Q):

t, d = map(int, input().split())

q = dash\_day[t-1][0]

r = dash\_day[t-1][1]

if d % q <= r:

print(d + (r-d%q))

else:

print( d + q - (d%q -r ) )

# C Repeating

正解

N = int(input())

A = list(map(int, input().split()))

B = [None]\*N

d = {}

for i in range(N):

if A[i] not in d:

B[i] = -1

d[A[i]] = i+1

else:

B[i] = d[A[i]]

d[A[i]] = i+1

print(\*B)

# D Count Simple Paths

正解(chatGPTのコピー)

最初BFSでやっていたが、chatからDFSでやることがわかった。BFSでやった場合、キューに存在する状態がとても多くなり、メモリアクセスに時間がかかるため、常に小さいサイズで保持しておけるスタックが向いている。さらに、dfsは再帰の方がいい。

## コード(dfs 再帰)

H, W, K = map(int, input().split())

S = [input().strip() for \_ in range(H)]

ans = 0

def dfs(y, x, steps, visited):

global ans

if steps == K:

ans += 1

return

for dy, dx in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

ny, nx = y + dy, x + dx

if 0 <= ny < H and 0 <= nx < W and (ny, nx) not in visited:

if S[ny][nx] == ".":

visited.add((ny, nx))

dfs(ny, nx, steps + 1, visited)

visited.remove((ny, nx))

for i in range(H):

for j in range(W):

if S[i][j] == ".":

dfs(i, j, 0, {(i, j)})

print(ans)

## コード(dfs スタック)

H, W, K = map(int, input().split())

S = [input().strip() for \_ in range(H)]

ans = 0

# スタックを使ったDFS

for i in range(H):

for j in range(W):

if S[i][j] == ".":

# スタックには (y, x, steps, visited) を追加

stack = [(i, j, 0, {(i, j)})]

while stack:

y, x, steps, visited = stack.pop()

# 移動回数がKなら1つの経路をカウント

if steps == K:

ans += 1

continue

# 上下左右の隣接セルに移動

for dy, dx in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

ny, nx = y + dy, x + dx

if 0 <= ny < H and 0 <= nx < W and (ny, nx) not in visited:

if S[ny][nx] == ".":

# 次の状態をスタックに追加

new\_visited = visited | {(ny, nx)}

stack.append((ny, nx, steps + 1, new\_visited))

print(ans)

## コード(bfs キュー)

h, w, k = map(int, input().split())

s = []

ans = 0

for \_ in range(h):

s.append(input())

for i in range(h):

for j in range(w):

if s[i][j] == ".":

q = [(i, j, 0, {(i, j)})]

while q:

y, x, cost, visited = q.pop(0)

if cost == k:

ans += 1

continue

for dy, dx in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

ny = y+dy

nx = x+dx

if 0<=ny<=h-1 and 0<=nx<=w-1 and (ny, nx) not in visited:

if s[ny][nx] == ".":

buf = visited | {(ny, nx)}

q.append((ny, nx, cost+1, buf))

print(ans)

## 復習正解コード

def dfs(ni, nj, cost):

if cost == K:

global ans

ans += 1

return

for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

nxi = ni+di

nxj = nj+dj

if 0<=nxi<H and 0<=nxj<W and visited[nxi][nxj] == False and S[nxi][nxj] == ".":

visited[nxi][nxj] = True

dfs(nxi, nxj, cost+1)

visited[nxi][nxj] = False

H, W, K = map(int, input().split())

S = [list(input()) for \_ in range(H)]

visited = [[False]\*W for \_ in range(H)]

ans = 0

for i in range(H):

for j in range(W):

if S[i][j] == ".":

visited[i][j] = True

dfs(i, j, 0)

visited[i][j] = False

print(ans)

# E Mod Sigma Problem

不正解

フェニック木(BIT : Binary Indexed tree)を使って実装。

## コード(rを固定したとき; これは解説と同じ思考)

from itertools import accumulate

class FenwickTree:

def \_\_init\_\_(self, m):

self.bit = [0]\*(m+1)

self.m = m

def add(self, i, x):

j = i+1

while j <= self.m:

self.bit[j] += x

j += (-j)&j

def sum(self, i):

result = 0

j = i+1

while 0 < j:

result += self.bit[j]

j -= (-j)&j

return result

def solve():

# 入力の取得

N, M = map(int, input().split())

A = list(map(int, input().split()))

S = [0] + [x % M for x in accumulate(A)]

SS = [0]\*(N+1)

for i in range(N):

SS[i+1] = SS[i] + S[i+1]

ft = FenwickTree(M)

result = 0

for r in range(1, N + 1):

result += S[r]\*r - SS[r-1] + (ft.sum(M-1)-ft.sum(S[r]))\* M

ft.add(S[r], 1)

print(result)

solve()

## コード(lを固定したとき)

from itertools import accumulate

class FenwickTree:

def \_\_init\_\_(self, m):

self.bit = [0]\*(m+1)

self.m = m

def add(self, i, x):

j = i+1

while j <= self.m:

self.bit[j] += x

j += (-j)&j

def sum(self, i):

result = 0

j = i+1

while 0 < j:

result += self.bit[j]

j -= (-j)&j

return result

N, M = map(int, input().split())

A = list(map(int, input().split()))

S = [0] + [x % M for x in accumulate(A)]

rSS = [0]\*(N+2)

x = [0]\*(N+1)

ans = 0

for i in range(N, 0, -1):

rSS[i] = rSS[i+1] + S[i]

rest = FenwickTree(M)

rest.add(S[N],1)

for l in range(N,0, -1):

x[l] = rest.sum(S[l-1]-1)

rest.add(S[l-1], 1)

for l in range(1, N+1):

ans += rSS[l] - S[l-1]\*(N-l+1) + x[l]\*M

print(ans)

## 復習コード

class Fenwicktree:

def \_\_init\_\_(self, n):

self.A = [0]\*n

self.n = n

def add(self, i, x):

id = i+1

while id <= self.n:

self.A[id-1] += x

id += id & (-id)

def sum(self, l, r):

id = l

sl = 0

while id > 0:

sl += self.A[id-1]

id -= id & (-id)

id = r

sr = 0

while id > 0:

sr += self.A[id-1]

id -= id & (-id)

return sr-sl

N, M = map(int, input().split())

A = list(map(int, input().split()))

S = [0]\*(N+1)

count = [0]\*N

for i in range(N):

S[i+1] = (A[i]+S[i])%M

ft = Fenwicktree(M)

for i in range(1, N+1):

count[i-1] = ft.sum(S[i]+1, M)

ft.add(S[i], 1)

ans = 0

S2 = [0]\*N

for i in range(N-1):

S2[i+1] += S2[i]+S[i+1]

for r in range(1, N+1):

ans += r\*S[r] - S2[r-1] + M\*count[r-1]

print(ans)

# F Add One Edge 2

解いてない

# G Everlasting LIDS

解いてない