# ------------ python code ------------- #

from copy import copy

def add\_available\_resources(i): # this func. is for adding the allocated resource of the executed process

global resource\_allocation\_matrix, resources\_available, change\_in\_available\_resource\_matrix

a = []

[a.append(int(x)) for x in resource\_allocation\_matrix[i].split()]

for j in range(len(resources\_available)):

resources\_available[j] += a[j]

change\_in\_available\_resource\_matrix.append(copy(resources\_available))

# ----------- driver code ------------ #

with open('input.txt') as f: # reading all the lines of the input file

lines = f.readlines() # at once

processes = int(lines[0])

resources = int(lines[1])

max\_resource\_matrix = [] # these are

resource\_allocation\_matrix = [] # the

resources\_available = [] # lists

need\_matrix = [] # we will

safe\_sequence = [] # be needing

change\_in\_available\_resource\_matrix = [] # for our

flag\_for\_need\_matrix = [False] \* processes # task

[max\_resource\_matrix.append(lines[x].strip()) for x in range(2, 2+processes)] # handling the input data

[resource\_allocation\_matrix.append(lines[x].strip()) for x in range(2+processes, 2+(2\*processes))] # handling the input data

for i in lines[-1].strip().split(): # handling the input data

resources\_available.append(int(i))

for i in range(processes): # this for-loop is to form the need matrix

a = [int(x) for x in max\_resource\_matrix[i].split()]

b = [int(x) for x in resource\_allocation\_matrix[i].split()]

need\_matrix.append([a[j] - b[j] for j in range(resources)])

while True: # this is the main loop for implementing Banker's algorithm

flag = False # this flag will remain False if no processes are executed in this round

for i in range(processes): # every time we have to check if there's any process left to execute

if not flag\_for\_need\_matrix[i]: # if there is a process left to execute

for j in range(resources): # we are checking if available resources >= needed resources of that process

if need\_matrix[i][j] > resources\_available[j]: # even if 1 instance of a resource is insufficient,

break # we break and move on the next process

if j == resources-1: # if all the instances meet the criteria

flag\_for\_need\_matrix[i] = True # we mark the process as executed

flag = True # we understand that at least 1 process has been executed in this round

safe\_sequence.append(chr(i+65)) # we add the process giving it a alphabetic name

add\_available\_resources(i) # we add back the allocated resource of that process to the available resource

if all(flag\_for\_need\_matrix) or not flag: # if all the processes are done executing or

break # no processes were executed in this round, we break the loop

# ------- printing the results ------- #

if not flag:

print("Safe sequence is :")

[print(x, end=' ') for x in safe\_sequence]

print("\nNo more processes left whose resource needs can be met so there'll be a Deadlock")

exit()

print("\nNeed Matrix :")

for i in need\_matrix:

[print(x, end=' ') for x in i]

print()

print("\nSafe sequence is :")

[print(x, end=' ') for x in safe\_sequence]

print('\n\nChange in available resource matrix :')

for i in change\_in\_available\_resource\_matrix:

[print(x, end=' ') for x in i]

print()