import math

import random

comparisons = 0

def alphaBetaPruning(idx, depth, alpha, beta, attacker):

global comparisons, branch

if depth == 0: return tree[idx]

if attacker:

tree[idx] = -math.inf

for i in range(branch \* idx + 1, (branch \* idx + branch) + 1, 1):

val = alphaBetaPruning(i, depth - 1, alpha, beta, False)

tree[idx] = max(tree[idx], val)

alpha = max(alpha, val)

if alpha >= beta:

break

return tree[idx]

else:

tree[idx] = math.inf

for i in range(branch \* idx + 1, (branch \* idx + branch) + 1, 1):

val = alphaBetaPruning(i, depth - 1, alpha, beta, True)

tree[idx] = min(tree[idx], val)

if depth == 1: comparisons += 1

beta = min(beta, val)

if alpha >= beta:

break

return tree[idx]

#-------------- tester code -------------

id = input("Enter your student id: \n")

minDamage, maxDamage = map(int, input("Minimum and Maximum value for the range of negative HP: \n").split(' '))

initial\_hp = int(id[:-3:-1])

depth = int(id[0]) \* 2

branch = int(id[2])

print(f'\nDepth and Branches ratio is {depth}:{branch}')

tree = []

for i in range(depth):

nodes = pow(branch, i)

for x in range(nodes):

if i % 2 == 0: tree.append(-math.inf)

else: tree.append(math.inf)

random\_leaf\_nodes = [random.randint(minDamage, maxDamage) for x in range(pow(branch, depth))]

print('Terminal States(Leaf Nodes) are', \*random\_leaf\_nodes, sep=', ', end='.\n')

tree += random\_leaf\_nodes

alphaBetaPruning(0, depth, -math.inf, math.inf, True)

left\_hp = initial\_hp - tree[0]

print('Left life(HP) of the defender after maximum damage caused by the attacker is', left\_hp)

print('After Alpha-Beta Pruning Leaf Node Comparisons', comparisons)