AI EX 1

CAMEL AND BANANA PROBLEM

AIM:A person has 3000 bananas and a camel. The person wants to transport the maximum number of bananas to a destination which is 1000 KMs away, using only the camel as a mode of transportation. The camel cannot carry more than 1000 bananas at a time and eats a banana every km it travels. What is the maximum number of bananas that can be transferred to the destination using only camel.

PROCEDURE:

First of all, the brute-force approach does not work. If the Camel starts by picking up the 1000 bananas and try to reach point B, then he will eat up all the 1000 bananas on the way and there will be no bananas left for him to return to point A.

So we have to take an approach that the Camel drops the bananas in between and then returns to point A to pick up bananas again.

Since there are 3000 bananas and the Camel can only carry 1000 bananas, he will have to make 3 trips to carry them all to any point in between.

```
<---p1---><--------p2-----><-----p3---->
A--------------------------------------->B
```

When bananas are reduced to 2000 then the Camel can shift them to another point in 2 trips and when the number of bananas left are <= 1000, then he should not return and only move forward.

In the first part, P1, to shift the bananas by 1Km, the Camel will have to

1. Move forward with 1000 bananas – Will eat up 1 banana in the way forward
2. Leave 998 banana after 1 km and return with 1 banana – will eat up 1 banana in the way back
3. Pick up the next 1000 bananas and move forward – Will eat up 1 banana in the way forward
4. Leave 998 banana after 1 km and return with 1 banana – will eat up 1 banana in the way back
5. Will carry the last 1000 bananas from point a and move forward – will eat up 1 banana

So to shift 3000 bananas by 1km, the Camel will eat up 5 bananas. After moving to 200 km the Camel would have eaten up 1000 bananas and is now left with 2000 bananas.

Now in the Part P2, the Camel needs to do the following to shift the Bananas by 1km.

1. Move forward with 1000 bananas – Will eat up 1 banana in the way forward
2. Leave 998 banana after 1 km and return with 1 banana – will eat up this 1 banana in the way back
3. Pick up the next 1000 bananas and move forward – Will eat up 1 banana in the way forward

So to shift 2000 bananas by 1km, the Camel will eat up 3 bananas. After moving to 333 km the camel would have eaten up 1000 bananas and is now left with the last 1000 bananas.

The Camel will actually be able to cover 333.33 km, I have ignored the decimal part because it will not make a difference in this example.

Hence the length of part P2 is 333 Km.

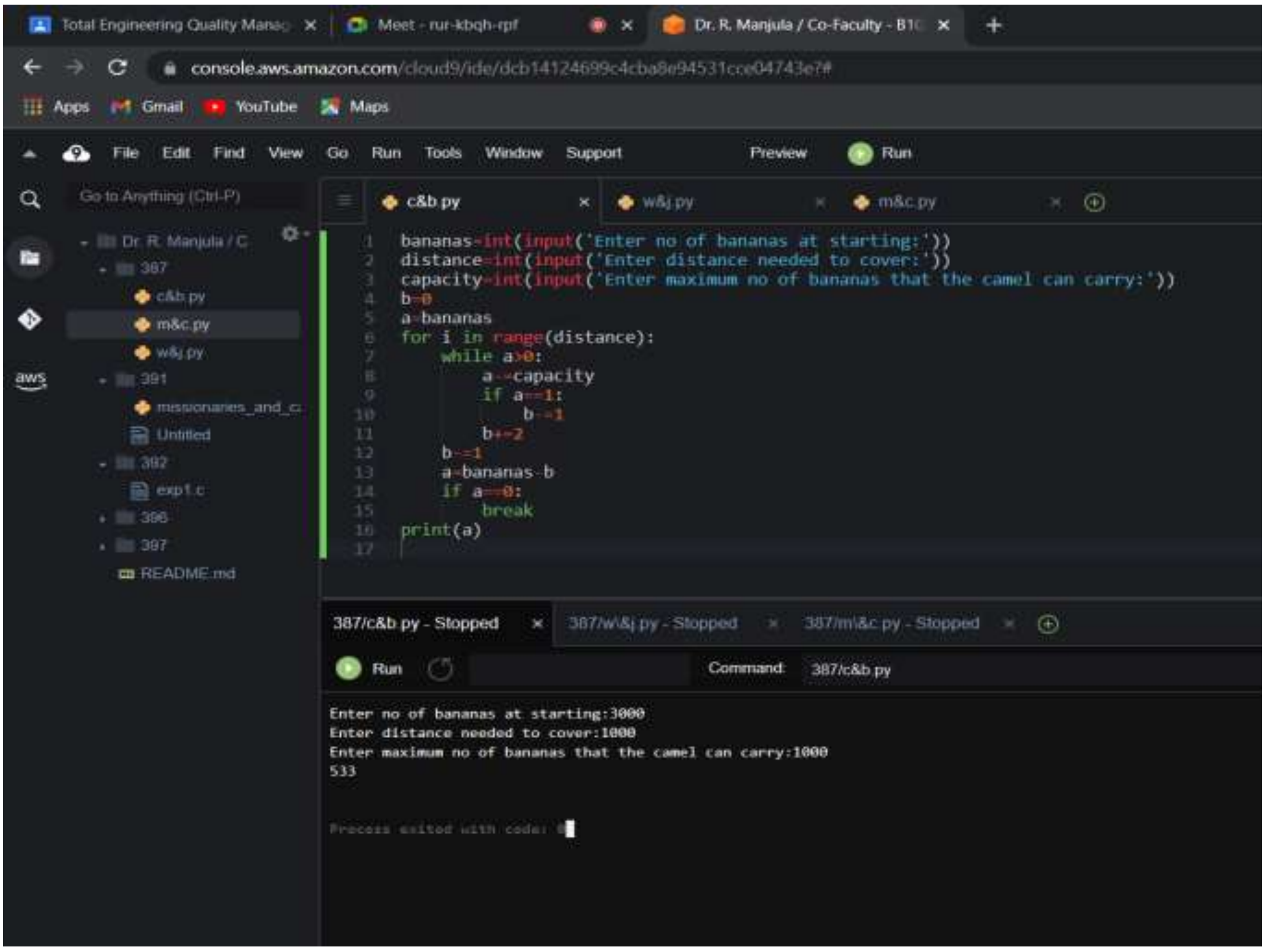
Now, for the last part, P3, the Camel only has to move forward. He has already covered 533 (200+333) out of 1000 km in Parts P1 & P2. Now he has to cover only 467 km and he has 1000 bananas.

He will eat up 467 bananas on the way forward, and at point B the Camel will be left with only 533 Bananas.

CODE:

```
bananas=int(input('Enter no of bananas at starting:'))
distance=int(input('Enter distance needed to cover:'))
capacity=int(input('Enter maximum no of bananas that the camel can carry:'))
b=0
a=bananas
for i in range(distance):
    while a>0:
        a-=capacity
        if a==1:
            b-=1
        b+=2
    b-=1
    a=bananas-b
    if a==0:
        break
print(a)
```

<u>OUTPUT:</u>



```
bananas=int(input('Enter no of bananas at starting:'))
distance=int(input('Enter distance needed to cover:'))
capacity=int(input('Enter maximum no of bananas that the camel can carry:'))
b=0
a=bananas
for i in range(distance):
    while a>0:
        a-=capacity
        if a==1:
            b-=1
        b+=2
    b-=1
    a=bananas-b
    if a==0:
        break
print(a)
```

```
Enter no of bananas at starting:3000
Enter distance needed to cover:1000
Enter maximum no of bananas that the camel can carry:1000
533


Process exited with code: 0
```

<u>RESULT</u>: The camel and banana problem was successfully executed

MISSIONARIES AND CANNIBALS PROBLEM

AIM: Three missionaries and three cannibals find themselves on one side of a river. They have would like to get to the other side. But the missionaries are not sure what else the cannibals agreed to. So the missionaries managed the trip across the river in such a way that the number of missionaries on either side of the river is never less than the number of cannibals who are on the same side. The only boar available holds only two at a time. How can everyone get across the river without the missionaries risking being eaten?

PROCEDURE:

First let us consider that both the missionaries (M) and cannibals(C) are on the same side of the river.

Left Right

Initially the positions are : 0M , 0C and 3M , 3C (B)

Now send 2 Cannibals to left of bank : 0M , 2C (B) & 3M , 1C

Send one cannibal from left to right : 0M , 1C and 3M , 2C (B)

Now send the 2 Cannibals to left : 0M , 3C (B) and 3M , 0C

Send 1 cannibal to the right : 0M , 2C and 3M , 1C (B)

Now send 2 missionaries to the left : 2M , 2C (B) and 1M . 1C

Send 1 M and 1 C to right : 1M , 1C and 2M , 2C (B)

Send 2 missionaries to left : 3M , 1C (B) and 0M , 2C

Send 1 cannibal to right : 3M , 0C and 0M , 3C (B)

Send 2 cannibals to left : 3M , 2C (B) and 0M , 1C

Send 1 cannibal to right : 3M , 1C and 0M , 2C (B)’

Send 2 cannibals to left : 3M , 3C (B) and 0M , 0C

Here (B) shows the position of the boat after the action is performed.

Therefore all the missionaries and cannibals have crossed the river safely.

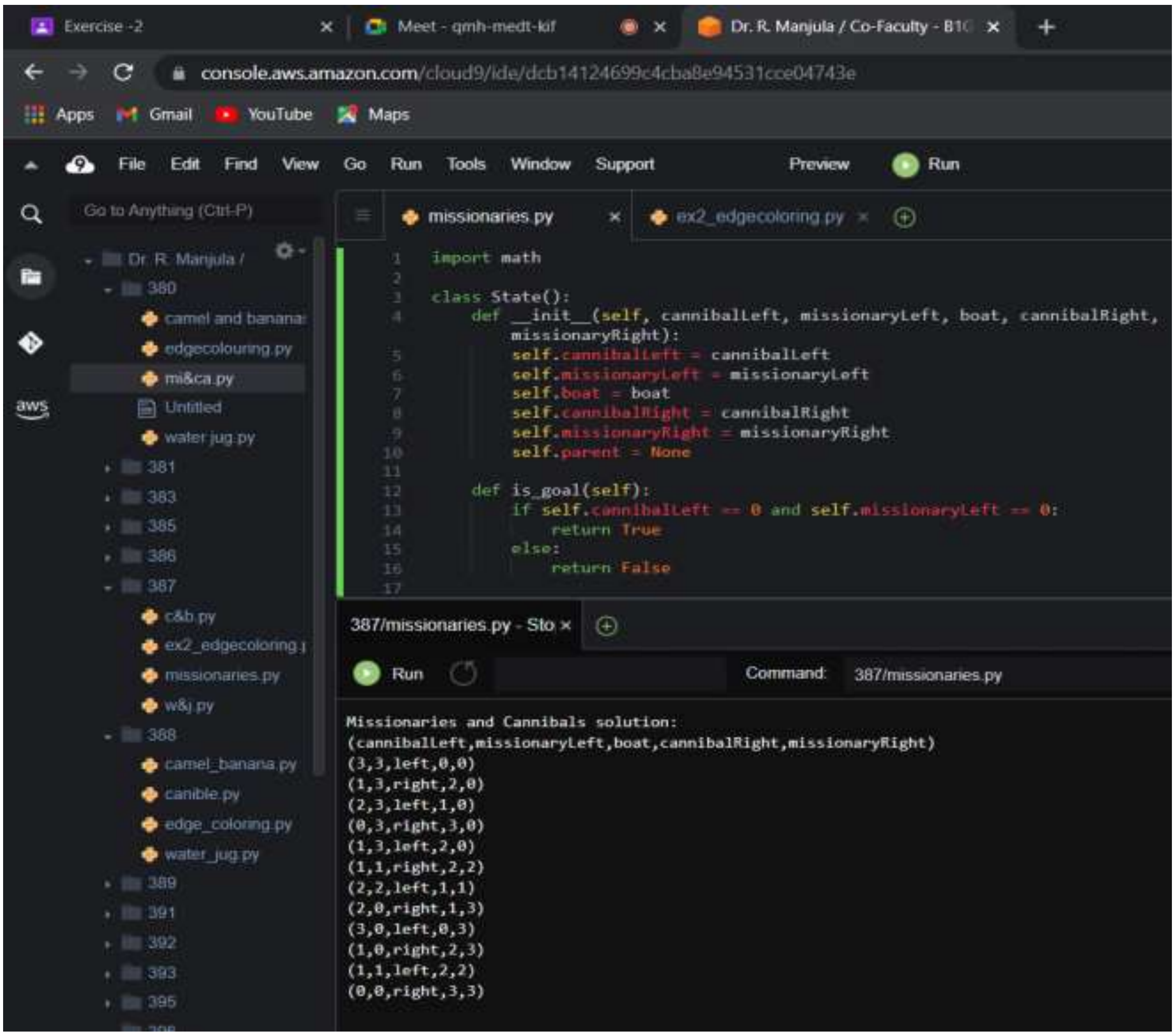
CODE:

```
import math


class State():
	def __init__(self, cannibalLeft, missionaryLeft, boat, cannibalRight, missionaryRight):
		self.cannibalLeft = cannibalLeft
		self.missionaryLeft = missionaryLeft
		self.boat = boat
		self.cannibalRight = cannibalRight
		self.missionaryRight = missionaryRight
		self.parent = None

	def is_goal(self):
		if self.cannibalLeft == 0 and self.missionaryLeft == 0:
			return True
		else:
			return False

	def is_valid(self):
		if self.missionaryLeft >= 0 and self.missionaryRight >= 0 \
  and self.cannibalLeft >= 0 and self.cannibalRight >= 0 \
  and (self.missionaryLeft == 0 or self.missionaryLeft >= self.cannibalLeft) \
  and (self.missionaryRight == 0 or self.missionaryRight >= self.cannibalRight):
			return True
		else:
			return False

	def __eq__(self, other):
		return self.cannibalLeft == other.cannibalLeft and self.missionaryLeft == other.missionaryLeft \
  and self.boat == other.boat and self.cannibalRight == other.cannibalRight \
  and self.missionaryRight == other.missionaryRight

	def __hash__(self):
		return hash((self.cannibalLeft, self.missionaryLeft, self.boat, self.cannibalRight, self.missionaryRight))


def successors(cur_state):
	children = [];
	if cur_state.boat == 'left':
		new_state = State(cur_state.cannibalLeft, cur_state.missionaryLeft - 2, 'right',
		  cur_state.cannibalRight, cur_state.missionaryRight + 2)
		## Two missionaries cross left to right.
		if new_state.is_valid():
			new_state.parent = cur_state
			children.append(new_state)
		new_state = State(cur_state.cannibalLeft - 2, cur_state.missionaryLeft, 'right',
		  cur_state.cannibalRight + 2, cur_state.missionaryRight)
		## Two cannibals cross left to right.
		if new_state.is_valid():
			new_state.parent = cur_state
			children.append(new_state)
		new_state = State(cur_state.cannibalLeft - 1, cur_state.missionaryLeft - 1, 'right',
		  cur_state.cannibalRight + 1, cur_state.missionaryRight + 1)
		## One missionary and one cannibal cross left to right.
		if new_state.is_valid():
			new_state.parent = cur_state
			children.append(new_state)
		new_state = State(cur_state.cannibalLeft, cur_state.missionaryLeft - 1, 'right',
		  cur_state.cannibalRight, cur_state.missionaryRight + 1)
		## One missionary crosses left to right.
		if new_state.is_valid():
			new_state.parent = cur_state
			children.append(new_state)
		new_state = State(cur_state.cannibalLeft - 1, cur_state.missionaryLeft, 'right',
		  cur_state.cannibalRight + 1, cur_state.missionaryRight)
		## One cannibal crosses left to right.
		if new_state.is_valid():
			new_state.parent = cur_state
			children.append(new_state)
	else:
		new_state = State(cur_state.cannibalLeft, cur_state.missionaryLeft + 2, 'left',
		  cur_state.cannibalRight, cur_state.missionaryRight - 2)
		## Two missionaries cross right to left.
		if new_state.is_valid():
			new_state.parent = cur_state
```

```
                children.append(new_state)
            new_state = State(cur_state.cannibalLeft + 2, cur_state.missionaryLeft, 'left',
             cur_state.cannibalRight - 2, cur_state.missionaryRight)
            ## Two cannibals cross right to left.
            if new_state.is_valid():
                new_state.parent = cur_state
                children.append(new_state)
            new_state = State(cur_state.cannibalLeft + 1, cur_state.missionaryLeft + 1, 'left',
             cur_state.cannibalRight - 1, cur_state.missionaryRight - 1)
            ## One missionary and one cannibal cross right to left.
            if new_state.is_valid():
                new_state.parent = cur_state
                children.append(new_state)
            new_state = State(cur_state.cannibalLeft, cur_state.missionaryLeft + 1, 'left',
             cur_state.cannibalRight, cur_state.missionaryRight - 1)
            ## One missionary crosses right to left.
            if new_state.is_valid():
                new_state.parent = cur_state
                children.append(new_state)
            new_state = State(cur_state.cannibalLeft + 1, cur_state.missionaryLeft, 'left',
             cur_state.cannibalRight - 1, cur_state.missionaryRight)
            ## One cannibal crosses right to left.
            if new_state.is_valid():
                new_state.parent = cur_state
                children.append(new_state)
        return children

def breadth_first_search():
        initial_state = State(3,3,'left',0,0)
        if initial_state.is_goal():
            return initial_state
        frontier = list()
        explored = set()
        frontier.append(initial_state)
        while frontier:
            state = frontier.pop(0)
            if state.is_goal():
                return state
            explored.add(state)
            children = successors(state)
            for child in children:
                    if (child not in explored) or (child not in frontier):
                            frontier.append(child)
        return None

def print_solution(solution):
                path = []
                path.append(solution)
                parent = solution.parent
                while parent:
                        path.append(parent)
                        parent = parent.parent

                for t in range(len(path)):
                        state = path[len(path) - t - 1]
                        print ("(" + str(state.cannibalLeft) + "," + str(state.missionaryLeft) \
               + "," + state.boat + "," + str(state.cannibalRight) + "," + \
               str(state.missionaryRight) + ")")

def main():
        solution = breadth_first_search()
        print ("Missionaries and Cannibals solution:")
        print ("(cannibalLeft,missionaryLeft,boat,cannibalRight,missionaryRight)")
        print_solution(solution)

if __name__ == "__main__":
  main()
```

<u>RESULT:</u> The missionaries and cannibal problem was successfully executed

OUTPUT:



```python
import math

class State():
    def __init__(self, cannibalLeft, missionaryLeft, boat, cannibalRight,
        missionaryRight):
        self.cannibalLeft = cannibalLeft
        self.missionaryLeft = missionaryLeft
        self.boat = boat
        self.cannibalRight = cannibalRight
        self.missionaryRight = missionaryRight
        self.parent = None

    def is_goal(self):
        if self.cannibalLeft == 0 and self.missionaryLeft == 0:
            return True
        else:
            return False
```

```
Missionaries and Cannibals solution:
(cannibalLeft,missionaryLeft,boat,cannibalRight,missionaryRight)
(3,3,left,0,0)
(1,3,right,2,0)
(2,3,left,1,0)
(0,3,right,3,0)
(1,3,left,2,0)
(1,1,right,2,2)
(2,2,left,1,1)
(2,0,right,1,3)
(3,0,left,0,3)
(1,0,right,2,3)
(1,1,left,2,2)
(0,0,right,3,3)
```

RA1911003010387

CHITRALEKHA.CH