Day Order - 2

Avinash Reddy Vasipalli
RA1911027010007
N1 (lab batch: 1)

04/02/2022

ExpNo:1 Toy Program Implementation

① Camel and Banana

Aim: To find a logical solution for a given toy program Puzzle namely "Camel and banana" puzzle.

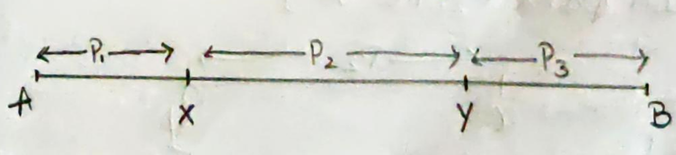
Description of Concept

A owner of banana plantation has a camel. He has 3000 bananas to be transpoted to a market which is 1000km away. Camel is the only ~~and~~ mode of transportation and it can carry only 1000 bananas once at max. Camel also eats one banana for every one kilometer travelled. So we need to find the maximum ~~no~~ number of banana that can be transported.

Manual solution

Firstlly a direct ~~ep~~ appraoched is futile, as for 1km otravelled 1 banana is eaten so no use in going all the way. Best approch is to have a drop point in middle where the camel can drop the bananas & get back to previous spot to pick bananas.

So lets divide the path into 3 distance i.e 2 drop points namly x, y

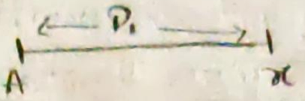


let $AX \rightarrow P_1$

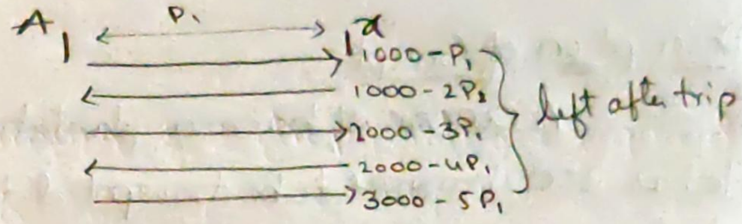
$XY \rightarrow P_2$

$YB \rightarrow P_2$

* Consider path P_1 i.e Ax.



As A is starting point it has 3000 bananas so to shift them to x we need to make 3 trips from A to x & 2 trips from x to A. i.e



So for every 1km 1 banana is eaten so. While carrying 1000 bananas from A to x camel eats 'P_1' bananas for each trip as it need banana even when going back

So a total of 5 trips ~~to travel 3000 bananas to x~~ are ~~to~~ required to transport 3000 bananas from A to x

∴ Equation to find bananas at x.

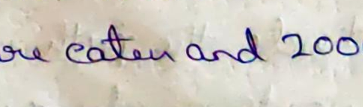
$$3000 - 5P_1 = 2000 \; [$$

$$5P_1 = 1000$$

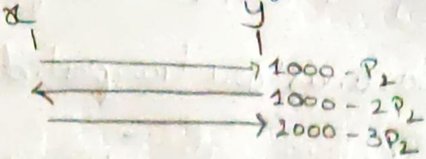
$$P_1 = 200.$$

So distance from A to x is 200. So for 5 trips 1000 bananas are eaten and 2000 are left at x.

* Consider path P_2 i.e xy



at x we have 2000 banana lets consider 1000 at y. So now for 2000 we need 2 trips for camel to transport from x & y. so one back trip is required. So total 3 trips of distance P_2 is made.



Equation to find P_2 (or) bananas at y

$$2000 - 3P_2 = 1000$$

$$3P_2 = 1000$$

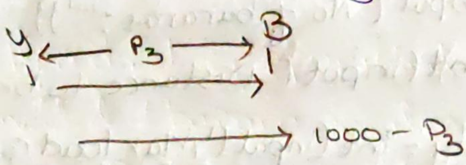
$$P_2 = 333.33.$$

as for every 1 km 1 banana is eaten we can take only numericals so let P_2 be 333 km.

So no. of bananas after 3 trips from x to y are 1001 [$2000 - 3(333) = 1001$] ~~as~~

* Now for last path y to B. We have around 1001 where 1000 ~~are~~ ~~too~~ is limit so '1' is left out.

Out of 1000 we loss P_3 amount of bananas as each km need one banana i.e only one trip is required.



As mention distance between market & plantation is 1000

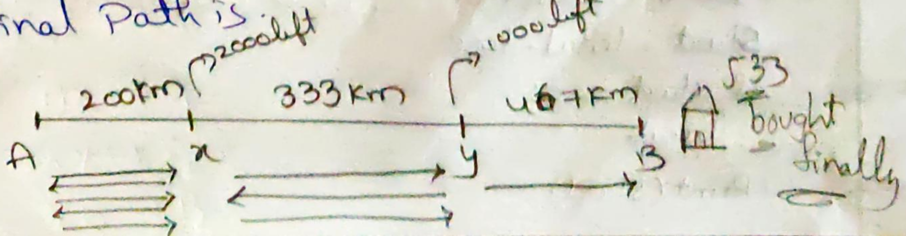
$$\therefore 1000 - P_1 - P_2 = P_3$$

$$P_3 = 1000 - 200 - 333$$

$$P_3 = 1000 - 533$$

$$P_3 = 467.$$

So final Path is



So finally the camel transported 1000 - 467 i.e

533 bananas are transported out of 3000.

This ~~the~~ is the most efficent output of all. Consider 3000 and 533 is very hard but the output could be worst ~~or~~ impossible. This is the optimal way.

Note

-> A minimum of 2 drop points are required for this Solution.

-> It doesn't matter if we have more than 2 drop points as the amount of banana consumed is same for distance travelled.

Program Implementation (Coding)

```
total = int(input('No. of bananas: '))
distance = int(input('Distance to be travelled: '))
load_capacity = int(input('Max load capacity of camel: '))
lose = 0
Start = total
for i in range(distance):
    while start > 0:
        Start = start - load_capacity
        if start == 1:
            lose = lose - 1
        lose = lose + 2
    lose = lose - 1
    Start = total - lose
    if Start == 0:
        break
    Print(Start)
```

AI(lab1).ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

RAM Disk Editing

[16]
```
total=int(input('No. of bananas : '))
distance=int(input('Distance to be travelled : '))
load_capacity=int(input('Max load capacity of your camel : '))
```

```
No. of bananas : 3000
Distance to be travelled : 1000
Max load capacity of your camel : 1000
```

[17]
```
lose = 0
start = total
```

[18]
```
for i in range(distance):
    while start>0:
        start=start-load_capacity
        if start == 1:
          lose = lose - 1
        lose = lose + 2
    lose = lose - 1
    start = total - lose
    if start == 0:
      break
print(start)
```

```
533
```

AI(lab1).ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

[24]
```
total=int(input('No. of bananas : '))
distance=int(input('Distance to be travelled : '))
load_capacity=int(input('Max load capacity of your camel : '))
```

```
No. of bananas : 5000
Distance to be travelled : 1000
Max load capacity of your camel : 500
```

[25]
```
lose = 0
start = total
```

```
for i in range(distance):
    while start>0:
        start=start-load_capacity
        if start == 1:
          lose = lose - 1
        lose = lose + 2
    lose = lose - 1
    start = total - lose
    if start == 0:
      break
print(start)
```

```
65
```

→ Input commands are used for no. of bananas, load capacity of camel and distance needed to be covered.

→ Consider a variable named lose equal to '0' and also start = total.

→ Go, Implement a for loop in range of distance. where start is checked above '0'. We update start value for every time 'i' value i.e start = start - load - capacity.

→ If condition is checked next for camel ~~doesn't~~ doesn't move back if only one banana left.

→ for lose = lose - 1 step lose is decreased because if camel tries to get left one banana he will loose one extra banana.

→ lose = lose + 2 we increase lose because for moving backward and forward by one mile. 2 banana are lost.

→ For last trip camel will not go back so we decrease '1' from lose

→ Finally we need to check possibility of carrying one banana (or) not and break to print the final output.

V Avinash Reddy

Avinash Reddy Vasipalli

04/01/2022

② Three Water Jugs Problems

Aim

To implement 3 water jugs problem in python

Description of Concept

We have 3 jugs that can be used to fill water. They can hold 12L, 8L, 5L respectively, where initial state is 12,0,0. The required output is 6,6,0. The jugs has no markings.

Manual Solution

Take 8L and pour water from 12L to 8L and remaining 4L in 5L finally (0,8,4). Now transfer 8L of 2nd jug water to 1st and 4L to 2nd jug so (8,4,0) is final.

Now transfer 5L to 3rd jug to (3,4,5). Now take 3rd jug and fill 2nd jug to brim to make the level (3,8,1). Now transfer 8L from 2nd jug to 1st & 1L from 3rd to 2nd to (11,1,0). Now from 1st jug pour 5L in 3rd jug ~~and pour that 5L~~ (6,1,5)

finally pour 5L of 3rd jug to 2nd jug this gives the output (6,6,0) the required output.

Program Implementation [Coding]

```python
capacity = (12,8,5)
# Maximum capacities of 3 jugs -> x,y,z
x = capacity[0]
y = capacity[1]
z = capacity[2]

# to mark visited states
memory = {}

# store solution path
ans = []

def get_all_states(state):
  # Let the 3 jugs be called a,b,c
  a = state[0]
  b = state[1]
  c = state[2]

  if(a==6 and b==6):
    ans.append(state)
    return True

  # if current state is already visited earlier
  if((a,b,c) in memory):
    return False

  memory[(a,b,c)] = 1

  #empty jug a
  if(a>0):
    #empty a into b
    if(a+b<=y):
      if( get_all_states((0,a+b,c)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((a-(y-b), y, c)) ):
        ans.append(state)
        return True
```

```python
    #empty a into c
    if(a+c<=z):
      if( get_all_states((0,b,a+c)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((a-(z-c), b, z)) ):
        ans.append(state)
        return True

  #empty jug b
  if(b>0):
    #empty b into a
    if(a+b<=x):
      if( get_all_states((a+b, 0, c)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((x, b-(x-a), c)) ):
        ans.append(state)
        return True
    #empty b into c
    if(b+c<=z):
      if( get_all_states((a, 0, b+c)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((a, b-(z-c), z)) ):
        ans.append(state)
        return True

  #empty jug c
  if(c>0):
    #empty c into a
    if(a+c<=x):
      if( get_all_states((a+c, b, 0)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((x, b, c-(x-a))) ):
        ans.append(state)
        return True
```

```python
  #empty jug c
  if(c>0):
    #empty c into a
    if(a+c<=x):
      if( get_all_states((a+c, b, 0)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((x, b, c-(x-a))) ):
        ans.append(state)
        return True
    #empty c into b
    if(b+c<=y):
      if( get_all_states((a, b+c, 0)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((a, y, c-(y-b))) ):
        ans.append(state)
        return True

  return False

initial_state = (12,0,0)
print("Starting work...\n")
get_all_states(initial_state)
ans.reverse()
for i in ans:
  print(i)
```

Output

AI(lab1).ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```python
    #empty c into b
    if(b+c<=y):
      if( get_all_states((a, b+c, 0)) ):
        ans.append(state)
        return True
    else:
      if( get_all_states((a, y, c-(y-b))) ):
        ans.append(state)
        return True

  return False

initial_state = (12,0,0)
print("Starting work...\n")
get_all_states(initial_state)
ans.reverse()
for i in ans:
  print(i)
```

```
Starting work...

(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)
```

V. Avinash Reddy

Avinash Reddy Vasipalli