REPORT

Objective: To conduct a theoretical study of the **"Lumberjack"** problem listed in Optil.io.

Team Members

1. Chinnavengannagari Vinoothna - 200010009
2. Kavali Sri Vyshnavi Devi - 200010023
3. Manche Pavanitha - 200010027
4. Sahaja Nandyala - 200010032

Functions Used

upProfit / downProfit / leftProfit / rightProfit functions:

- Initially, the variables uProfit,dProfit,rProfit and lProfit are assigned the value "0" globally and these variables are used in functions upProfit, downProfit, leftProfit, rightProfit respectively.
- After that, the trees are sorted and added to a list according to the direction in which they are to the tree passed in the argument of the function.
- After that, for each tree in this list, the condition for the domino effect is checked and if it is matched, then it is appended to another list to keep track of trees that are fallen due to the domino effect, and profit for it is calculated.
- These steps are the same in each of these functions but variables for each one are different.

Cal_Profit function :

- In this function, the above mentioned functions uProfit, dProfit, rProfit, and lProfit are being called to check in which direction the profit is more.
- At the end of this function, the direction in which profit is more and is returned.

Path1/Path2 functions :

- Initially, the function checks for the trees which can be cut in the remaining time left and append those trees into an empty list.
- In the path1 function, this list is sorted using the ratio of total profit (including profit by domino effect) to the time required to travel and cut a given tree from the current position, and the next priority for sorting is given to the value of the tree.
- In the path2 function, this list is sorted using the ratio of value to the time required to travel and cut a given tree from the current position, and the last priority for sorting is given to the value of the tree.
- At last, these functions return the sorted list of trees and the total profit gained from the first tree in the sorted list.

Description of algorithm

- In the first step, the input is taken from the user, and the properties of each tree are taken in a dictionary and appended to a list called trees.
- And for each tree in the input, the value for the key “tree_profit” is the current profit returned from the cal_profit function.
- After that, the trees list is assigned the list that was returned when the path1 function is called.
- Next, we run a while loop until the total time taken to travel and to cut the trees does not exceed the time limit and also until the number of trees which are yet to be cut reaches 0.
- In the first step of this while loop, the direction in which the tree should be cut is decided by calling the cal_profit function.
- The directions needed to go to that particular tree from the current position are printed in the next step.
- After going to that tree, the direction in which the tree is supposed to be cut is checked by the direction value returned from the cal_profit function and then the cutting direction is printed.
- The trees which were cut in this process are removed from the trees list.
- Later, the path1 and path2 functions are called. The trees list is assigned the list that was returned from the function which gives the highest path_profit.
- This is the end of the while loop and this loop runs until the given condition is false.

Pseudocode of algorithm

```
SET time_limit, grid_size, no_of_trees TO map(int,input().split())
SET temp1, temp2, temp3, temp4 TO []
SET uProfit, lProfit, rProfit, dProfit TO 0

FUNCTION cal_profit(tree):
    SET currentProfit TO 0

    upProfit(tree)
    SET currentProfit TO uProfit
    SET direction TO 0

    downProfit(tree)
    IF dProfit > currentProfit:
        SET currentProfit TO dProfit
        SET direction TO 1
    ENDIF

    rightProfit(tree)
    IF rProfit > currentProfit:
        SET currentProfit TO rProfit
        SET direction TO 2
    ENDIF

    leftProfit(tree)
    IF lProfit > currentProfit:
        SET currentProfit TO lProfit
        SET direction TO 3
    ENDIF

    RETURN direction,currentProfit
```

```
ENDFUNCTION

FUNCTION upProfit(near_tree):
    global uProfit
    SET uptrees TO sorted([i FOR i in trees IF near_tree["x"] = i["x"] AND
near_tree["y"] < i["y"]],key=lambda j: j["y"])
    FOR i in uptrees:
        IF near_tree["h"] > abs(i["position"] - near_tree["position"]) AND near_tree["weight"] >
i["weight"]:
            uProfit += i["value"]
            temp1.append(i)
            SET near_tree TO i
        ELSE:
            BREAK
        ENDIF
    ENDFOR
ENDFUNCTION

FUNCTION downProfit(near_tree):
    global dProfit
    SET downtrees TO sorted([i FOR i in trees IF near_tree["x"] = i["x"] AND near_tree["y"] >
i["y"]],key=lambda j: -j["y"])

    FOR i in downtrees:
        IF near_tree["h"] > abs(i["position"] - near_tree["position"]) AND near_tree["weight"] >
i["weight"]:
            dProfit += i["value"]
            temp2.append(i)
            SET near_tree TO i
        ELSE:
            BREAK
        ENDIF
    ENDFOR
ENDFUNCTION

FUNCTION rightProfit(near_tree):
    global rProfit
    SET righttrees TO sorted([i FOR i in trees IF near_tree["y"] = i["y"] AND near_tree["x"] <
i["x"]],key=lambda j: j["x"])

    FOR i in righttrees:
        IF near_tree["h"] > abs(i["position"] - near_tree["position"]) AND near_tree["weight"] >
i["weight"]:
            rProfit += i["value"]
            temp3.append(i)
            SET near_tree TO i
        ELSE:
            BREAK
        ENDIF
    ENDFOR
ENDFUNCTION

FUNCTION leftProfit(near_tree):
    global lProfit
    SET lefttrees TO sorted([i FOR i in trees IF near_tree["y"] = i["y"] AND near_tree["x"] >
i["x"]],SET key TO lambda j: -j["x"])

    FOR i in lefttrees:
        IF near_tree["h"] > abs(i["position"] - near_tree["position"]) AND near_tree["weight"] >
i["weight"]:
            lProfit += i["value"]
            temp4.append(i)
            SET near_tree TO i
```

```
        ELSE:
            BREAK
        ENDIF
    ENDFOR
ENDFUNCTION

FUNCTION path1():
SET list TO [i FOR i in trees IF abs(i["x"]-current_x)+abs(i["y"]-current_y)+i["d"] <= t]
list.sort(SET key TO lambda x:
(-((x["value"]+x["tree_profit"])/(abs(x["x"]-current_x)+abs(x["y"]-current_y)+x["d"])),-(x["value"])))
    IF len(list) != 0:
        RETURN list,cal_profit(list[0])[1]
    ELSE:
        RETURN [],0
    ENDIF
ENDFUNCTION

FUNCTION path2():
SET list TO [i FOR i in trees IF abs(i["x"]-current_x)+abs(i["y"]-current_y)+i["d"] <= t]
list.sort(SET key TO lambda x: (-(x["value"]/(abs(x["x"]-current_x)+abs(x["y"]-current_y)+x["d"])),
(x["x"]-current_x)**2 + (x["y"]-current_y)**2,-(x["value"])))
    IF len(list) != 0:
        RETURN list,cal_profit(list[0])[1]
    ELSE:
        RETURN [],0
    ENDIF
ENDFUNCTION

FOR i in range(no_of_trees):
    x, y, h, d, c, p TO map(int,input().split())
    trees.append({"position":x+y,"x":x,"y":y,"h":h,"d":d,"c":c,"p":p, "value":p*h*d, "weight":c*d*h,
"tree_profit":0})
ENDFOR

SET time TO 0
SET current_x,current_y TO 0
SET t TO time_limit
FOR i in trees:
    SET i["tree_profit"] TO cal_profit(i)[1]
    SET temp1, temp2, temp3, temp4 TO []
    SET uProfit, lProfit, rProfit, dProfit TO 0
ENDFOR

SET trees TO path1()[0]

SET temp1,temp2,temp3,temp4 TO []
SET uProfit, lProfit, rProfit, dProfit TO 0
WHILE time <= time_limit AND len(trees)>0:

    SET direction TO cal_profit(trees[0])[0]

    IF current_x < trees[0]["x"]:
        OUTPUT "move right\n"*(trees[0]["x"]-current_x),end=""
        t -= trees[0]["x"]-current_x

    ELSEIF current_x > trees[0]["x"]:
        OUTPUT "move left\n"*(current_x-trees[0]["x"]),end=""
        t -= current_x-trees[0]["x"]
    ENDIF

    IF current_y < trees[0]["y"]:
        OUTPUT "move up\n"*(trees[0]["y"]-current_y),end=""
        t -= trees[0]["y"]-current_y
```

```
ELSEIF current_y > trees[0]["y"]:
    OUTPUT "move down\n"*(current_y-trees[0]["y"]),end=""
    t -= current_y-trees[0]["y"]
ENDIF

IF direction = 0 AND (t - trees[0]["d"]) >= 0 :
    OUTPUT "cut up"
    SET a TO temp1
    t -= trees[0]["d"]

ELSEIF direction = 1 AND (t - trees[0]["d"]) >= 0:
    OUTPUT "cut down"
    SET a TO temp2
    t -= trees[0]["d"]

ELSEIF direction = 2 AND (t - trees[0]["d"]) >= 0:
    OUTPUT "cut right"
    SET a TO temp3
    t -= trees[0]["d"]

ELSEIF direction = 3 AND (t - trees[0]["d"]) >= 0:
    OUTPUT "cut left"
    SET a TO temp4
    t -= trees[0]["d"]
ENDIF

time += trees[0]["d"] + abs(trees[0]["x"]-current_x)+ abs(trees[0]["y"]-current_y)

SET current_x TO trees[0]["x"]
SET current_y TO trees[0]["y"]

trees.remove(trees[0])

IF len(a) != 0 AND len(trees) != 0:
    FOR i in a:
        trees.remove(i)
    ENDFOR
ENDIF

SET temp1,temp2,temp3,temp4 TO []
SET uProfit, lProfit, rProfit, dProfit TO 0

SET list1,path1_profit TO path1()

SET temp1,temp2,temp3,temp4 TO []
SET uProfit, lProfit, rProfit, dProfit TO 0

SET list2,path2_profit TO path2()

SET temp1,temp2,temp3,temp4 TO []
SET uProfit, lProfit, rProfit, dProfit TO 0

IF path1_profit > path2_profit:
    SET trees TO list1
ELSE:
    SET trees TO list
ENDIF
```

Data structures used

- The data structures used in our code are lists and dictionaries.
- The reason to use dictionaries is that it is easy to access data and time complexity is less.
- Lists are more convenient to work with because they are mutable and it is an ordered collection of items.

Complexity analysis of algorithm

The worst-case time complexity of the algorithm used -$O(n^2 \log n)$

- For taking input - $O(n)$
- For assigning values to the key “tree profit” - $O(n^2 \log n)$
- For path1and path2 function - $O(n \log n)$
- upProfit / downProfit / rightProfit / leftProfit - $O(n \log n)$
- Calculate profit function - $O(n \log n)$
- While loop checking the time limit - $O(n^2 \log n)$

List of theoretical observations on the problem

- Firstly, when trees were sorted with respect to the distance of the nearest trees to the current position without considering the value of a tree, in this case, the profit was less as compared to the final profit.
- Also, there were many TLE’s when all the trees were taken into the list without considering whether a tree would be cut in the remaining time left.
- When only the value of the tree was considered for sorting, it didn’t yield much profit.
- The sorting done using the ratio of value by the distance of the tree from the current position gave profit which was good enough.