Tizen CoBY – Sales man problem

1504/2016

Problem statement (1/2)

- In a city (C), sales man (S) can sell his product only on referral basis. Rectangle in the form of array of strings is given; where each value in the rectangle represents distance of the referred customer. It indicates how many steps S has to move (in any direction) for the next customer. S can move in 4 directions – top, down, left, right.
- Each element in the rectangle can have values 0 to 9(inclusive); meaning referral distance is up to 9.
- Initially, S sells product to first customer in the rectangle (who is at location (0x0)).
- S will sell only one product each time when he visits a customer.
- S will stop his sales when he encounters a cell with value 0or if the number of moves goes out of the rectangle area.
- Print the maximum number of products that S can sell if he chooses the moves strategically.
- If S can sell infinitely large amount of products, then print -1.

Problem statement (2/2)

- Consider below example (rectangle size 1x10),
- 2030040005
- S will first sell to customer at (0x0) location then he moves 2 steps right to reach location (0x2). Now, moving 3 steps left will go beyond the rectangle limit, so he chooses 3 steps to right and reach (0x5).Now, moving 4 steps left will reach (0x1) who doesn't provide referral, but if he moves 4 steps right he will reach (0x9). Now, moving 5 steps on left will reach (0x4) who doesn't refer and moving to right will cross the rectangle boundary. Hence S stops the sale with maximum sale of 4 products.

- Similarly in the below example,
- 3942178
- 1234567
- 9123532

- S can sell maximum of 5 products. (0x0) -> (0x3) ->(2x3) -> (2x6) -> ((0x6) OR (2x4))

- In below example, S can sell infinite products, hence answer is -1.
- 11

- Constraints
- 1 <= H, W <= 50

Approach (1/2)

- Recursion function:
  - Start with initial cell (0, 0)
  - Make the cell visited
  - At each step, recursively explore 4 available options – top, down, left, right.
  - Answer is maximum of above 4 paths + 1
  - Make the cell unvisited so that it is available for other paths
  - Recursion base cases:
    - Cell index out of bounds
    - Cell value 0
  - Loop case: if visited cell is visited again
- Above recursion takes about 4^n time in worst case as 4 options are explored at every node. This leads to timeout
- If we observe solution closely, we can figure out that sub-problem solutions are used again and again. This makes the problem fit for memorization (dynamic programming)

Approach (2/2)

- Recursion function including memorization:
  - Start with initial cell (0, 0)
  - If answer computed already, return it
  - Make the cell visited
  - At each step, recursively explore 4 available options – top, down, left, right.
  - Answer is maximum of above 4 paths + 1
  - Remember the answer
  - Make the cell unvisited so that it is available for other paths
  - Loop case: if visited cell is visited again
  - Recursion base cases:
    - Cell index out of bounds
    - Cell value 0

Pseudo-code

```
Initialize: MemoryMatrix[H][W] = 0 ; Visited[H][W] = 0;
Input: InputMatrix[H][W]
Procedure Traverse(location x, y)
        if (x >= W OR x < 0 OR y >= H OR y < 0 OR Input[x][y] == 0)   // out of bounds or
cell value 0
        return 0
        if (Visited[x][y])
        return -1  //infinity case

        if (MemoryMatrix[x][y] > 0)     //memorized value
        return MemoryMatrix[x][y]

        Steps <- Input[x][y]
        Soln1 = Traverse(x + steps, y)
        Soln2 = Traverse(x – steps, y)
        Soln3 = Traverse(x, y + steps)
        Soln4 = Traverse(x, y – steps)

        Visited[x][y] = 0;
        MemoryMatrix[x][y] = Max of (Soln1, Soln2, Soln3, Soln4) + 1
        return MemoryMatrix[x][y]
```

For full code, check submission by winners at: [Sotong CoBY Contest]