zachary_hightower_433_p4_report

Program explanation

main method

We initialize variables for m, n, and the switcher here. m and n are defined by the program requirements as
m : number of guess tokens
n : upper range of the number set we're working with 0 -> n
The switcher is what allows the program to go between the bonus game method and the regular implementation that outputs the minimum number of guesses necessary to arrive at the correct answer.

Once we initialize, we create the tables that we're going to use, and then proceed to perform whatever the user argument indicates. If the user argument can't be processed, we display the user manual to allow them to find out what they need to input to get the output they need.

userManual

This is a manual that tells the user what sort of input the program is expecting and what kind of things they can do with the program. It provides a brief explanation of all the variables and what to do to get the effects the user wants.

createTablesFindMin

This method makes the tables that we use to determine the correct number of guesses in various different conditions.

To manage this we have two arrays for our costs and guesses. We establish the initial states of the tables with the two for loops, and then calculate the interior of the 2d matrix, so we know under all conditions what the minimum amount of guesses will be. This helps us to store the previous solutions, which is the heart of dynamic programming.

findMinGuesses

This method handles the printing for the output and looks up the solution in the table for us.

awakenZOLTAR

This initializes the guessing game that swaps the positions of player and referee so that the human is now the referee and the computer is the player. It's also in the form of an old style fortune teller to add a small amount of joy to life. We put the m and n values into more easily readable variables for the program, and then find the first position of currentGuess from the guesses table. We also initialize questionNum and sum to 0.

The rest of the logic is mostly for the user input and the print statements that show where ZOLTAR is in the table and how he's determining what question to ask next.

Pseudo-code

```
Initialize:
- costs: 2D array to store minimum costs for guessing
- guesses: 2D array to store guesses for each token count and target number
- m: number of guess tokens
- n: upper range of the target number
- switcher: switch between different functionalities

Process command-line arguments (m, n, switcher):
Parse m, n, and switcher from command-line arguments
If switcher == 0:
Find minimum guesses to identify target number
Else if switcher == 1:
Play game with ZOLTAR
Else:
Print "Invalid input" and display user manual

Else:
Display user manual

User manual:
Print instructions for command-line input format and options

Function createTablesFindMin(m, n):
Initialize costs and guesses arrays with dimensions (m+1) x (n+1)
Iterate over range from 1 to n:
Set costs[1][i] = i
Set guesses[1][i] = 1
Iterate over range from 1 to m:
Set costs[i][1] = 1
Set guesses[i][1] = 1
Iterate over range from 2 to m:
```

```
Iterate over range from 2 to n:
Initialize minCost to n + 1
Iterate over range from 1 to j:
Calculate cost = max(costs[i-1][k-1], costs[i][j-k]) + 1
If cost < minCost:
Update minCost and guesses[i][j]
Set costs[i][j] = minCost

Function findMinGuesses(m, n):
Print maximum questions needed to identify target number

Function awakenZOLTAR(m, n):
Initialize scanner, remainingTokens, targetNum, currentGuess, questionNum, and sum
Print introduction message
Iterate until currentGuess is 0:
Print remaining tokens and current question number
Read user input for whether the number is less than current guess
If answer is no:
Update targetNum and sum
Else:
Decrement remainingTokens and update targetNum
Update currentGuess
Print ZOLTAR's guess
```

Optimal Substructure (recurrence)

We want to define the recurrence relation for the problem. To do that we need to examine everything that happens in the createTablesAndFindMin method. That's where we stored all the logic for making the table that stores our solutions.

We initialize the arrays according to the dimensions $(m+1) \times (n+1)$

We use two for loops to fill in the starting positions.

We go through and fill in the table with the amount of guesses necessary for each condition of tokens and number ranges

So, we can say that the recurrence is

$$C[i][j]$$

Where i is the number of tokens necessary to find the correct guess in the range from 0 to j

We can express this recurrence relation as

$$C[i][j] = min_{w/\ respect\ to\ k}[max(cost[i-1, k-1], cost[j-k]) + 1]$$

Where k is between 1 and j ,$1 \leq k \leq j$