Kaustav Malik

Mirror-Laser-Safe Problem

- Data Structures
    - Lists
        - Horizontal travel segments: item format is [[row1, col1],[row1, col2]]
        - Vertical travel segments: item format is [[row1, row2], col]
        - Intersections: item format is a point [r , c]
    - Dictionaries
        - Columns with mirrors: item format is (column index: [row, mirror orientation])
        - Rows with mirrors: item format is (row index: [column, mirror orientation])
- Algorithm Approach
    - The first step is to traverse the path of the laser from the starting point to find its ending position, intake each linear path segment, and determine whether the safe is already solved.
    - If the safe is already solved, return 0.
    - If the safe is not solved, the next step is to find any potential locations for a mirror to be placed to solve the maze. This is accomplished by traversing the safe backwards, starting from the desired destination, and finding any intersections of the backwards-path with any of the forwards-path segments that were accounted for by the segment lists.
    - Each intersection accounts for a possible mirror placement to solve the safe. The length of the intersection dictionary indicates the number of possible mirror placement solutions that exist. By sorting the intersection list ((r, c) coordinates), the lexicographically smallest row/column value is the first value of the list
- Time Complexity Analysis
    - Using the directional information of the laser as well as dictionary lookups for mirror locations, the safe could be traversed simply by looking at the mirrors that exist in the current row or column being traversed. Since the lookup time of a key in a dictionary is O(1) , the worst case time complexity for mirror lookups while traversing through the safe is O(N), where N represents the number of mirrors in the safe.
    - However, after finding all the mirror locations for a specified row or column, the nearest mirror in the direction of the laser must be determined. This involves O(n) time complexity to loop through the mirrors found in the row or column. While the time complexity of the traversal remains the same, this part of the algorithm adds running time to the solution.
    - If the forward traversal of the safe determines that the safe is not solved, the required backward trace adds a similar O(n) time complexity, except the dictionary lookup time for mirrors in the row or column is replaced by a list lookup of traversed segments from the forward path. List slicing has a time complexity of O(k) where k is the number of elements sliced.
    - Furthermore, the intersection list is sorted with O(n*log(n)) complexity, where n represents the number of intersections.
- Space Complexity Analysis

  - The space complexity of storing the mirror locations in two separate dictionaries is O(a) + O(b), where “a” and “b” represent the number of rows and columns that contain mirrors, respectively. In the worst case, where every row and column has a mirror, the space complexity would be O(m*n)+O(m*n), where “m” and “n” represent the number of rows and columns in the safe, respectively.
  - The horizontal and vertical segments are stored in lists. This adds and O(n) space complexity, where n is one more than the total number of mirrors traversed in the safe.
  - Lastly, the intersections are stored in a list with O(n) space complexity, where n represents the number of intersections between the forwards and backwards safe traversal.
- Running Instructions
  - This program was tested with Python 2.7.15 on Windows
  - The program is run with the following command:
    - python main.py input.txt
    - Note: modify input.txt to test other cases