Genetic Alignment

Kyle Pontius – Section 2

GitHub For Code: https://github.com/kbpontius/CS312-Project4

Scoring Algorithm Analysis

Space Complexity (n): The scoring algorithm works by reusing two List<int> to calculate the new values. The first row always holds previous values, and the second rows is where the new values are computed. Then, after each iteration, the two lists are swapped and reused. Because the scoring algorithm uses two n-length rows, the total space complexity is 2n, which falls in the O(n) space complexity. The algorithm never exceeds two lists.

Time Complexity (n^2): The scoring algorithm still requires the same number of calculations to derive each cell in the DP table as the extraction algorithm does. This calculation is performed almost identically, by iterating over each cell (at most n^2 cells), then calculating the minimum cost for moving down, right, or diagonally and assigning it to the new cell. In the Grid class, the CalculateScoreSolution() method iterates over each row, and each column in that row (the while-loop w/ a nested for-loop). While iterating, it finds the minimum cost for moving into that cell and places that as the new cell's cost. Once finished with each row, it assigns the newly calculated row to be the old-value reference row and generates a new, empty List<int> to calculate the new values with, dumping the old one to preserve memory.

Summary: See summary in Extraction Algorithm Analysis for rationale on why complexity may not always be exactly n^2 or n.

Extraction Algorithm Analysis

Space Complexity (n^2): This algorithm differs in how it calculates the aligned strings. Instead of just keeping two rows to calculate the new values with, it maintains the entire 2d array of objects. These objects (referred to as DirectionCost objects in the code) store the incoming direction for a cell. Additionally, the DirectionCost held the new total cost a cell has, which is calculated from the left, top, and diagonal cells.

Time Complexity (n^2): My CalculateExtractionSolution() method is responsible for looping over each cell, calculating the new cost and the direction the move originated from. Inside CalculateExtractionSolution() there are two for-loops, which is where the n^2 time complexity is derived. It is necessary to iterate over each cell in the 2d array, just like the Scoring Algorithm, to determine each new value for the cells, working toward the end which provides the total cost for finding aligning the two strings.

Algorithm Analysis Summary: In both the space and time complexity, depending on the lengths of both the left-sequence and the top-sequence, you may get at most n^2 for both space and time complexity. There are situations, however, where one string will be shorter than the other and you're left with a space and time complexity of O(n*m).

Alignment Extraction

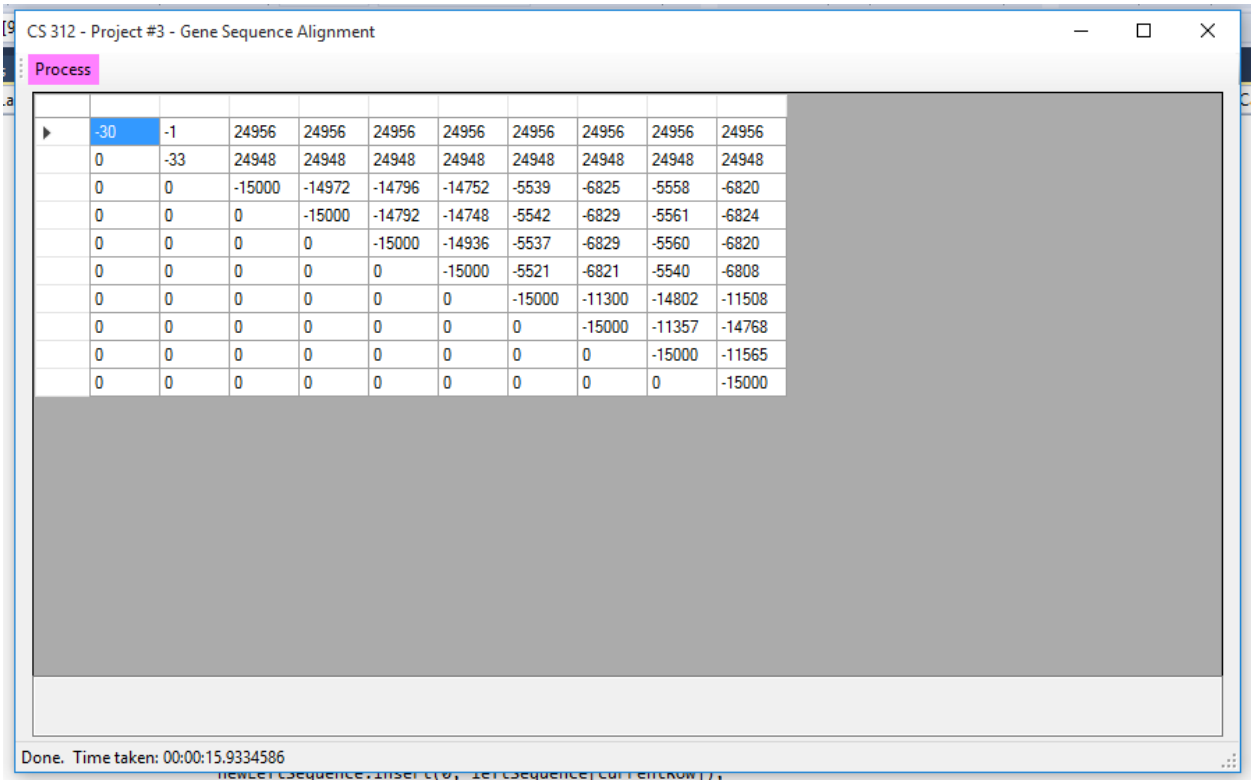
For extracting the path which aligns the strings, I begin at the end of the path, in the cell where the total score is provided. From there, I work backward through the array ***aligning all 5000 characters***, and then clipping the array to the first 100 after the backtracking has finished. This code can be found in the Grid class, inside the ExtractPath() method:

1. First, from the current cell, which direction did the move come (up, left, diagonal).
2. Depending on the direction, change the currentRow and currentCol to setup for the next cell.
    - Left: currentCol -= 1, currentRow remains the same, currentDirection = newCell.direction
    - Up: currentRow -= 1, currentCol remains the same, currentDirection = newCell.direction
    - Diag: currentRow -= 1, currentCol -= 1, currentDirection = newCell.direction.
3. Step #2 is happening, the strings are also being built by inserting the new character at the beginning of the string:
    - Left: newLeftSequence.Insert(0, "-"), newTopSequence.Insert(0, topSequence.currentLetter)
    - Up: newLeftSequence.Insert(0, leftSequence.currentLetter), newTopSequence.Insert(0, topSequence.currentLetter)
    - Diag: newLeftSequence.Insert(0, leftSequence.currentLetter), newTopSequence.Insert(0, topSequence.currentLetter)

4. Step #3 generates the new, aligned strings based on the movement direction determined in Step #2. Then the strings are returned to the GUI to be printed out.

Results (images)

Screenshot for 10x10 matrix:



Screenshot for Row #3, Col #10 Result (aligned 5000, then trimmed):