Project #4 Description

In Homework 4, we explored the use of dynamic programming in measuring the similarity between two sequences of characters. Given an alphabet Σ and a scoring matrix M defined over $\Sigma \cup \{'-'\}$, the dynamic programming method computed a score that measured the similarity of two sequences X and Y based on the values of this scoring matrix. In particular, this method involved computing an *alignment* matrix S between X and Y whose entry S_{ij} scored the similarity of the substrings $X[0 \ldots i-1]$ and $Y[0 \ldots j-1]$. These notes provided an overview of the process.

In Project 4, we will implement four functions. The first pair of functions will return matrices that we will use in computing the alignment of two sequences. The second pair of functions will return global and local alignments of two input sequences based on a provided alignment matrix. You will then use these functions in Application 4 to analyze two problems involving comparison of similar sequences.

Modeling matrices

For Project 4, you will work with two types of matrices: alignment matrices and scoring matrices. Alignment matrices will follow the same indexing scheme that we used for grids in "Principles of Computing". Entries in the alignment matrix will be indexed by their row and column with these integer indices starting at zero. We will model these matrices as lists of lists in Python and can access a particular entry via an expression of the form `alignment_matrix[row][col]`.

For scoring matrices, we take a different approach since the rows and the columns of the matrix are indexed by characters in $\Sigma \cup \{'-'\}$. In particular, we will represent a scoring matrix in Python as a dictionary of dictionaries. Given two characters `row_char` and `col_char`, we can access the matrix entry corresponding to this pair of characters via `scoring_matrix[row_char][col_char]`.

Matrix functions

The first two functions that you will implement compute a common class of scoring matrices and compute the alignment matrix for two provided sequences, respectively. The first function builds a scoring matrix as a dictionary of dictionaries.

- `build_scoring_matrix(alphabet, diag_score, off_diag_score, dash_score)`: Takes as input a set of characters `alphabet` and three scores `diag_score`, `off_diag_score`, and `dash_score`. The function returns a dictionary of dictionaries whose entries are indexed by pairs of characters in `alphabet` plus '-'. The score for any entry indexed by one or more dashes is `dash_score`. The score for the remaining diagonal entries is `diag_score`. Finally, the score for the remaining off-diagonal entries is `off_diag_score`.

One final note for `build_scoring_matrix` is that, although an alignment with two matching dashes is not allowed, the scoring matrix should still include an entry for two dashes (which will never be used).

The second function computes an alignment matrix using the method **ComputeGlobalAlignmentScores** described in Homework 4. The function computes either a global alignment matrix or a local alignment matrix depending on the value of `global_flag`.

- `compute_alignment_matrix(seq_x, seq_y, scoring_matrix, global_flag)`: Takes as input two sequences `seq_x` and `seq_y` whose elements share a common alphabet with the scoring matrix `scoring_matrix`. The function computes and returns the alignment matrix for `seq_x` and `seq_y` as described in the Homework. If `global_flag` is `True`, each entry of the alignment matrix is computed using the method described in Question 8 of the Homework. If `global_flag` is `False`, each entry is computed using the method described in Question 12 of the Homework.

Alignment functions

For the second part of Project 4, you will use the alignment matrix returned by `compute_alignment_matrix` to compute global and local alignments of two sequences `seq_x` and `seq_y`. The first function will implement the method **ComputeAlignment** discussed in Question 9 of the Homework.

- `compute_global_alignment(seq_x, seq_y, scoring_matrix, alignment_matrix)`: Takes as input two sequences `seq_x` and `seq_y` whose elements share a common alphabet with the scoring matrix `scoring_matrix`. This function computes a global alignment of `seq_x` and `seq_y` using the global alignment matrix `alignment_matrix`.The function returns a tuple of the form `(score, align_x, align_y)` where `score` is the score of the global alignment `align_x` and `align_y`. Note that `align_x` and `align_y` should have the same length and may include the padding character '-'.

This second function will compute an optimal local alignment starting at the maximum entry of the local alignment matrix and working backwards to zero as described in Question 13 of the Homework.

- `compute_local_alignment(seq_x, seq_y, scoring_matrix, alignment_matrix)`: Takes as input two sequences `seq_x` and `seq_y` whose elements share a common alphabet with the scoring matrix `scoring_matrix`. This function computes a local alignment of `seq_x` and `seq_y` using the local alignment matrix `alignment_matrix`.The function returns a tuple of the form `(score, align_x, align_y)` where `score` is the score of the optimal local alignment `align_x` and `align_y`. Note that `align_x` and `align_y` should have the same length and may include the padding character '-'.

Grading and coding standards

As you implement each matrix function, test it thoroughly. Use the function `build_scoring_matrix` to generate scoring matrices for alphabets such as `"ACTG"` and `"abcdefghijklmnopqrstuvwxyz"`. Realistic choices for the scoring matrix should have large scores on the diagonal to reward matching the same character and low scores off the diagonal to penalize mismatches. Use these scoring matrices to test `compute_alignment_matrix` for simple examples that you can verify by hand. Once you are confident that your implementation of these two functions is correct, submit your code to this Owltest page, which will automatically test your project.

Next, implement your alignment functions. Use the alignment matrices to generate global and local alignments for simple test examples. Verifying that the global alignments are correct is relatively easy. Generating interesting local alignments and verifying that they are correct is harder. In general, the off-diagonal and dash scores in your scoring matrix should be negative for local alignments. One interesting test for your local alignment function is to choose a scoring matrix that mimics the computation of the longest common subsequence of `seq_x` and `seq_y`. Once you are confident that your implementation of these two functions is correct, submit your code to this Owltest page, which will automatically test your project.

OwlTest uses Pylint to check that you have followed the coding style guidelines for this class. Deviations from these style guidelines will result in deductions from your final score. Please read the feedback from Pylint closely. If you have questions, feel free to consult this page and the class forums. When you are ready to submit your code to be graded formally, submit your code to the CourseraTest page for this project that is linked on the main programming assignment page. Remember that submitting to OwlTest does not record a grade for the assignment.

Mark as completed