Homework 11

Rohit Kamat rgk359

Problem 2 of this homework is due on April 25, 2017 at 7:00pm.. Please submit Part 2 as a PDF file on Canvas. Before submission, please re-run all cells by clicking "Kernel" and selecting "Restart & Run All."

Problem 1 is due, on paper, at the beginning of lab on April 26, 2017. Problem 1 should not contain any code!

Problem 1 (5 points): Using **Smith-Waterman** (not Needleman-Wunsch!), align the following two sequences by hand:

```
CCAGT
ACAAGT
```

Draw out a score matrix, **with the back-tracing arrows**, using the following scoring function:

```
Match: +2
Mismatch: -1
Gap: -1
```

After you have filled out your score matrix, be sure to **write out the final alignment**.

Problem 2 (5 points): Modify the code from the Lab 13 Worksheet, Part 1 so that it runs the **Smith-Waterman** algorithm. Several helper functions are provided for you below. Your function final should produce the matrix of scores only. You **do not** need to do back-tracing. Use the same scoring function as in Problem 1.

Run the sequences from Problem 1 through your function and print the output using `print_matrix()`.

In [1]:
```
# Use these values to calculate scores
match_award = 2
mismatch_penalty = -1
gap_penalty = -1

# Make a score matrix with these two sequences
seq1 = "CCAGT"
seq2 = "ACAAGT"

# Here is a helper function to print out matrices
def print_matrix(mat):
    # Loop over all rows
    for i in range(0, len(mat)):
        print("[", end = "")
        # Loop over each column in row i
        for j in range(0, len(mat[i])):
            # Print out the value in row i, column j
            print(mat[i][j], end = "")
            # Only add a tab if we're not in the last column
            if j != len(mat[i]) - 1:
                print("\t", end = "")
        print("]\n")

# A function for making a matrix of zeroes
def zeros(rows, cols):
    # Define an empty list
    retval = []
    # Set up the rows of the matrix
    for x in range(rows):
        # For each row, add an empty list
        retval.append([])
        # Set up the columns in each row
        for y in range(cols):
            # Add a zero to each column in each row
            retval[-1].append(0)
    # Return the matrix of zeros
    return retval

# A function for determining the score between any two bases in alignment
def match_score(alpha, beta):
    if alpha == beta:
        return match_award
    elif alpha == '-' or beta == '-':
        return gap_penalty
    else:
        return mismatch_penalty

# The function that actually fills out a matrix of scores
def smith_waterman(seq1, seq2):

    # length of two sequences
    n = len(seq1)
    m = len(seq2)

    # Generate matrix of zeros to store scores
    score = zeros(m+1, n+1)
```

```
    ########################
    # Your code goes here
    ########################

    #Fill Out First Column
    for i in range(0, m + 1):
        score[i][0] = 0

    #Fill Out First Row
    for j in range(0, n + 1):
        score[0][j] = 0

    # Fill out all other values in the score matrix
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            # Calculate the score by checking the top, left, and diagonal ce
            match = score[i - 1][j-1] + match_score(seq1[j-1], seq2[i-1])
            delete = score[i - 1][j] + gap_penalty
            insert = score[i][j - 1] + gap_penalty
            # Record the maximum score from the three possible scores calcul
            score[i][j] = max(0, match, delete,insert)


    return score

print_matrix(smith_waterman(seq1, seq2))
```

```
[0      0      0      0      0      0]

[0      0      0      2      1      0]

[0      2      2      1      1      0]

[0      1      1      4      3      2]

[0      0      0      3      3      2]

[0      0      0      2      5      4]

[0      0      0      1      4      7]
```