# Roll: 32

# Jahir Sadik Monon

## Fundamentals of Genomics and Proteomics Lab (Task 3)

## Department of Computer Science and Engineering, University of Dhaka

References used: Needleman-Wunsch dynamic programing algorithm

Problem statement: Do a pairwise global alignment with the following:

Seq 1: CTCGCAGC

Seq 2: CATTCAG

for a=b: +10, for a<>b: -2, for (a,-) or (-,b): -5

```
# printing utility function for printing trace matrix or edit graph matrix
def print_matrix(mat, seq1, seq2, trace_print):
    # Print the horizontal header
    print('   ', end='')
    if not trace_print:
        print('   ', end='')
    for char in seq1:
        if trace_print:
            print(f' {char}  ', end='   ')
        else:
            print(f' {char}  ', end='')
    print()

    # Print the matrix values
    for i, row in enumerate(mat):
        # Print the vertical header
        if trace_print:
            print(f'{seq2[i]} ', end='')
        else:
            if i!=0:
                print(f'{seq2[i-1]} ', end='')
            else:
                print('  ', end='')

        # Print the row values
        if not trace_print:
            for val in row:
                print(f'{val:3d} ', end='')
        else:
            for val in row:
                print(f"{val} ", end='')
        print()

# evaluation function corresponding to input costs
def evaluate(a, b):
    if a == b:
        return 10
    elif (a != b) and a != '-' and b != '-':
        return -2
    elif a == '-' and b == '-':
        raise Exception("Two dashes cannot match!")
    return -5

# function for finding maximum possible value at mat index i,j
def fill_elem(seq1, seq2, mat, trace_mat, i, j):
    max_val = max(mat[i-1][j-1]+evaluate(seq1[i-1], seq2[j-1]),
```

```
                     mat[i-1][j]+evaluate(seq1[i-1], '-'),
                    mat[i][j-1]+evaluate('-', seq2[j-1])
                    )
    if max_val == (mat[i-1][j-1]+evaluate(seq1[i-1], seq2[j-1])):
        trace_mat[i][j] = (i-1, j-1)
        return max_val, trace_mat
    elif max_val == (mat[i-1][j]+evaluate(seq1[i-1], '-')):
        trace_mat[i][j] = (i-1, j)
        return max_val, trace_mat
    trace_mat[i][j] = (i, j-1)
    return max_val, trace_mat

# function for finding max value position from index i, j in directions
#   (up, left, diagonal)
def find_max_val_index(mat, i, j):
    max_val = -float('inf')
    max_val_index = (i, j)
    if i > 0 and j > 0:
        if mat[i-1][j-1] > max_val:
            max_val = mat[i-1][j-1]
            max_val_index = (i-1, j-1)
        if mat[i-1][j] > max_val:
            max_val = mat[i-1][j]
            max_val_index = (i-1, j)
        if mat[i][j-1] > max_val:
            max_val = mat[i][j-1]
            max_val_index = (i, j-1)
        return max_val_index
    elif i > 0 and j == 0:
        return (i-1, j)
    elif j > 0 and i == 0:
        return (i-1, j)
    else:
        raise Exception("find_max_val_index index error {i}{j}")



# test_seq_str1_slide = "AGCTAAGCTAA"
# test_seq_str2_slide = "AGCCTAGCCAA"
# Input Strings:
seq_str2 = "CTCGCAGC"
seq_str1 = "CATTCAG"
seq1 = list(seq_str1)
seq2 = list(seq_str2)
len_row = len(seq2) + 1
len_col = len(seq1) + 1
mat = [[0 for _ in range(len_col)] for _ in range(len_row)]

for i in range(0, len_row):
    mat[i][0] = i * evaluate(seq2[i-1], '-')

for j in range(0, len_col):
    mat[0][j] = j * evaluate(seq1[j-1], '-')

trace_mat = [[(0,0) for _ in range(len_col)] for _ in range(len_row)]
for i in range(1, len_row):
    for j in range(1, len_col):
        mat[i][j], trace_mat = fill_elem(seq2, seq1, mat, trace_mat, i, j)
        # print(f"Filled {i},{j} as: {mat[i][j]} using {trace_mat[i][j]}'s value")

print("Global sequence alignment final edit matrix:")
print_matrix(mat, seq1, seq2, False)
print(f'\nTrace matrix for Needleman-Wunsch dynamic programing algorithm \n
            (showing what index the max value for each element came from):')
print_matrix(trace_mat, [i for i in range(len_col)], [j for j in range(len_row)], True)
```

```
aligned_seq1 = []
aligned_seq2 = []
aligned_seq1_n = []
aligned_seq2_n = []
index = (len_row - 1, len_col - 1)
n_index = (len_row-1, len_col-1)

# Loop to implement algorithm showed in slide (Max value path)
while index[0] != 0 or index[1] != 0:
    index_n = find_max_val_index(mat, index[0], index[1])
    # print(f"{index[0]},{index[1]} -> {index_n[0]},{index_n[1]}")
    if index_n[0] == index[0]-1 and index_n[1] == index[1]-1:
        aligned_seq1.append(seq1[index[1]-1])
        aligned_seq2.append(seq2[index[0]-1])
        # print(f"{seq1[index[1]-1]} to seq1")
        # print(f"{seq2[index[0]-1]} to seq2")
        index = index_n
    elif index_n[0] == index[0]-1 and index_n[1] == index[1]:
        aligned_seq1.append("-")
        aligned_seq2.append(seq2[index[0]-1])
        # print(f"- to seq1")
        # print(f"{seq2[index[0]-1]} to seq2.")
        index = index_n
    elif index_n[0] == index[0] and index_n[1] == index[1]-1:
        aligned_seq1.append(seq1[index[1]-1])
        aligned_seq2.append("-")
        # print(f"{seq1[index[1]-1]} to seq1, ({index[1]-1} index at seq1)")
        # print(f"- to seq2")
        index = index_n

# Loop to implement Needleman-Wunsch dynamic programing algorithm
while n_index[0] != 0 or n_index[1] != 0:

    prev_row = trace_mat[n_index[0]][n_index[1]][0]
    prev_col = trace_mat[n_index[0]][n_index[1]][1]
    if prev_row == n_index[0]-1 and prev_col == n_index[1]-1:
        aligned_seq1_n.append(seq1[n_index[1]-1])
        aligned_seq2_n.append(seq2[n_index[0]-1])
        # print(f"{seq1[n_index[1]-1]} to seq1")
        # print(f"{seq2[n_index[0]-1]} to seq2")
        n_index = (n_index[0]-1, n_index[1]-1)
    elif prev_row == n_index[0]-1 and prev_col == n_index[1]:
        aligned_seq1_n.append("-")
        aligned_seq2_n.append(seq2[n_index[0]-1])
        # print(f"- to seq1")
        # print(f"{seq2[n_index[0]-1]} to seq2.")
        n_index = (n_index[0]-1, n_index[1])
    elif prev_row == n_index[0] and prev_col == n_index[1]-1:
        aligned_seq1_n.append(seq1[n_index[1]-1])
        aligned_seq2_n.append("-")
        # print(f"{seq1[n_index[1]-1]} to seq1, ({n_index[1]-1} index at seq1)")
        # print(f"- to seq2")
        n_index = (n_index[0], n_index[1]-1)

aligned_seq1.reverse()
aligned_seq2.reverse()
aligned_seq1_n.reverse()
aligned_seq2_n.reverse()
print("Sequences to be globally aligned were:")
print(seq_str1)
print(seq_str2)
print("Aligned sequences (according to max value path showed in slide) are:")
print(aligned_seq1)
print(aligned_seq2)
```

```
print("Aligned sequences
            (according to Needleman-Wunsch dynamic programing algorithm) are")
print(aligned_seq1_n)
print(aligned_seq2_n)
alignment_score = 0
for i in range(len(aligned_seq1)):
    alignment_score += evaluate(aligned_seq1[i], aligned_seq2[i])
print(f"Alignment score (according to max value path showed in slide) is:
            {alignment_score}")
alignment_score_n = 0
for i in range(len(aligned_seq1_n)):
    alignment_score_n += evaluate(aligned_seq1_n[i], aligned_seq2_n[i])
print(f"Alignment score (according to Needleman-Wunsch dynamic programing algorithm)
          is: {alignment_score_n}")
```