Project 1

Group 7

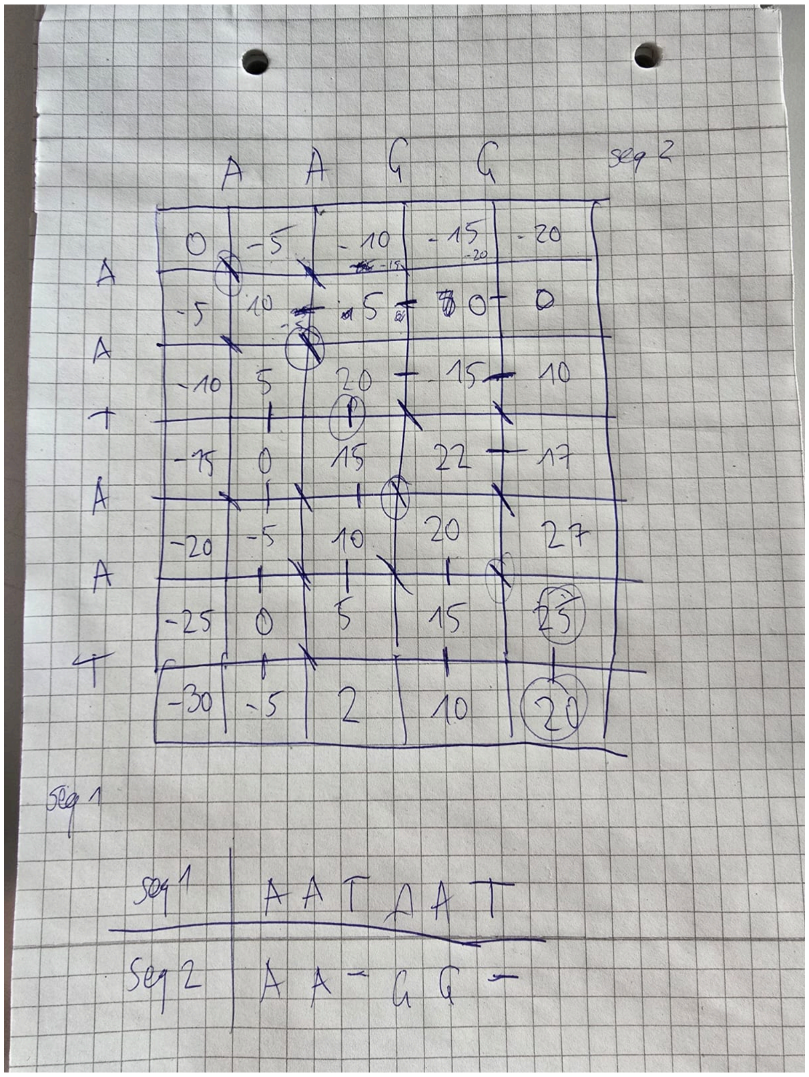
Al Ashrif Bin Ahamed - 202402610

Shakil Ahmed - 202402609

Swantje Katharina Hänsch - 202007286

Question 1: What is the optimal (here maximal) cost of an alignment of AATAAT and AAGG using the above substitution matrix and gap cost -5?

Answer (By hand):



		A	A	G	G
	0	-5	-10	-15	-20
A	-5	10	5	0	0
A	-10	5	20	15	10
T	-15	0	15	22	17
A	-20	-5	10	20	27
A	-25	0	5	15	25
T	-30	-5	2	10	20

seq 1	A A T A A T
seq 2	A A - G G -

```
In [1]: # importing the necessary modules for the global alignment sequencing
import numpy as np
from Bio import SeqIO
from Bio.Align import PairwiseAligner
from Bio.Align.substitution_matrices import Array
```

```
In [2]: # For the fasta file parsing the sequence
def read_fasta(file_path):
    with open(file_path, "r") as file:
        return str(next(SeqIO.parse(file, "fasta")).seq)
```

```
In [3]: # Given bases, substituition matrix and the sequences
        matrix_values = np.array([
            [10,  2,  5,  2],  # A
            [ 2, 10,  2,  5],  # C
            [ 5,  2, 10,  2],  # G
            [ 2,  5,  2, 10]   # T
        ])
        bases = "ACGT"

        seq1 = "AATAAT"
        seq2 = "AAGG"

        seq1_fasta = read_fasta("seq1.fasta")
        seq2_fasta = read_fasta("seq2.fasta")
```

Question 1: What is the optimal (here maximal) cost of an alignment of AATAAT and AAGG using the above substitution matrix and gap cost -5?

Answer: We implemented using biopython.

and

Question 2: What is the optimal (here maximal) cost of an alignment of seq1.fasta and seq2.fasta using the same substitution matrix and gap cost? (You probably want to implement the algorithm for computing the cost of an optimal alignment.)

Answer (Implementation): Below is the python implementation (with the help of biopython package) of optimal score calculation of given sequences `AATAAT` and `AAGG` including the `FASTA` file sequences.

```
In [4]: substitution_matrix = Array(alphabet=bases, dims=2, data=matrix_values)

        # Initialize pairwise aligner
        aligner = PairwiseAligner()
        aligner.mode = "global"
        aligner.substitution_matrix = substitution_matrix
        aligner.open_gap_score = -5
        aligner.extend_gap_score = -5

        # Compute optimal alignment score
        optimal_score = aligner.score(seq1, seq2)
        optimal_score_fasta = aligner.score(seq1_fasta, seq2_fasta)
        print(f"Optimal Alignment Score for {seq1} and {seq2}: {optimal_score}")
        print("Optimal Alignment Score for FASTA file:", optimal_score_fasta)
```

```
Optimal Alignment Score for AATAAT and AAGG: 20.0
Optimal Alignment Score for FASTA file: 1346.0
```

Question 3 (optional): How does an optimal alignment look like for the above two pairs of sequences using the given substitution matrix and gap cost -5? (you probably want to implement the algorithm for finding an optimal alignment by backtracking through the dynamic programming table.)

Answer: See the manual code below (without package). Because there is no specific package to do backtracking.

and

Question 4 (optional): How many optimal alignments are for the above two pairs of sequences using the given substitution matrix and gap cost -5? Explain how you can compute the number of optimal alignments.

Answer:

Fill the Dynamic Programming (DP) Table

Firstly, we need to fill the DP (Dynamic Programming) table.

- The DP table is constructed using the given **substitution matrix** and **gap penalty**.
- Each cell `(i, j)` stores the **maximum alignment score** computed using:
  - **Match/mismatch**: `dp[i-1][j-1] + substitution_score(seq1[i-1], seq2[j-1])`
  - **Insertion (gap in seq1)**: `dp[i][j-1] + gap_penalty`
  - **Deletion (gap in seq2):** `dp[i-1][j] + gap_penalty`

Construct the Backtracking Matrix

Secondly, we need to backtrack.

- Instead of storing just the **best** move, we store **all possible moves** that result in the optimal score at each step.
- At any cell `(i, j)`, if multiple moves lead to the **same optimal score**, we store **all of them**.
- Starting from `dp[m][n]`, we **recursively explore all paths** back to `dp[0][0]`.
- Each unique path corresponds to a **valid optimal alignment**.

In [5]:
```python
"""
    Here we consider seq1= "AATAAT" and seq2= "AAGG". If you want to use the
    FASTA file just change the seq1 =  read_fasta("seq1.fasta") and
    seq2 =  read_fasta("seq2.fasta") file path.S
"""
base_to_index = {base: i for i, base in enumerate(bases)}

gap_penalty = -5

# Initialize DP table
m, n = len(seq1), len(seq2)
dp = np.zeros((m+1, n+1), dtype=int)

# Fill the DP table
for i in range(1, m+1):
    dp[i][0] = dp[i-1][0] + gap_penalty
for j in range(1, n+1):
    dp[0][j] = dp[0][j-1] + gap_penalty

for i in range(1, m+1):
    for j in range(1, n+1):
        match = dp[i-1][j-1] + matrix_values[base_to_index[seq1[i-1]], base_to_index[seq2[j-1]]]
        delete = dp[i-1][j] + gap_penalty
        insert = dp[i][j-1] + gap_penalty
        dp[i][j] = max(match, delete, insert)

# Get the optimal alignment score
optimal_score = dp[m][n]
```

```python
# Function to backtrack and find all optimal alignments
def backtrack(i, j, aligned_seq1, aligned_seq2, results):
    if i == 0 and j == 0:
        results.append((aligned_seq1[::-1], aligned_seq2[::-1]))  # Reverse the alignment and st
        return

    # Move diagonally (match/mismatch)
    if i > 0 and j > 0 and dp[i][j] == dp[i-1][j-1] + matrix_values[base_to_index[seq1[i-1]], ba
        backtrack(i-1, j-1, aligned_seq1 + seq1[i-1], aligned_seq2 + seq2[j-1], results)

    # Move up (deletion)
    if i > 0 and dp[i][j] == dp[i-1][j] + gap_penalty:
        backtrack(i-1, j, aligned_seq1 + seq1[i-1], aligned_seq2 + "-", results)

    # Move left (insertion)
    if j > 0 and dp[i][j] == dp[i][j-1] + gap_penalty:
        backtrack(i, j-1, aligned_seq1 + "-", aligned_seq2 + seq2[j-1], results)

# Store all optimal alignments
results = []
backtrack(m, n, "", "", results)

"""
    This section (below) is a construction of dynamic programming table.
    This is visually appealing when the sequences are small. However, it
    can be ignored when we deal with FASTA file (large) sequence.
"""

# Print the DP table in a visually appealing format
print("\nDynamic Programming Table:\n")

# Print column headers (seq2 with spaces for alignment)
print("      ", end="")
for char in "-" + seq2:
    print(f"  {char}  ", end="")
print("\n   " + "─────" * (n+1))  # Horizontal separator

# Print DP table row by row
for i in range(m+1):
    row_label = seq1[i-1] if i > 0 else "-"
    print(f" {row_label} |", end="")

    # Print DP values with spacing
    for j in range(n+1):
        print(f"{dp[i][j]:^5}", end="")
    print()

print(f"\nOptimal Alignment Score for {seq1} and {seq2}: {optimal_score}")
print("Optimal possible alignments: ", len(results))
print("The all optimal alignment look like:")
print("-" * 30)
for aligned_seq1, aligned_seq2 in results:
    print(aligned_seq1)
    print(aligned_seq2)
    print("-" * 30)
```

```
Dynamic Programming Table:

       -    A    A    G    G
   ──────────────────────────
 - │  0   -5   -10  -15  -20
 A │ -5   10    5    0   -5
 A │ -10   5   20   15   10
 T │ -15   0   15   22   17
 A │ -20  -5   10   20   27
 A │ -25  -10   5   15   25
 T │ -30  -15   0   10   20

Optimal Alignment Score for AATAAT and AAGG: 20
Optimal possible alignments:  1
The all optimal alignment look like:
------------------------------
AATAAT
AA-GG-
------------------------------
```