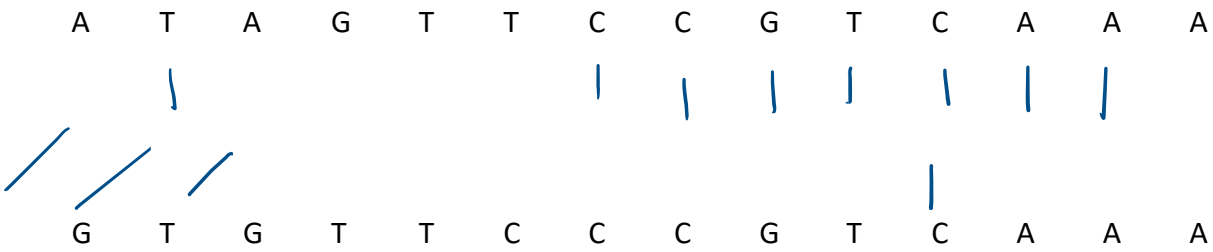
Assignment: Dynamic Programming

1. **Solve a problem using top-down and bottom-up approaches of Dynamic Programming technique**

   DNA sequence is made of characters A, C, G and T, which represent nucleotides. A sample DNA string can be given as 'ACCGTTTAAAG'. Finding similarities between two DNA sequences is a critical computation problem that is solved in bioinformatics.

   Given two DNA strings find the length of the longest common string alignment between them (it need not be continuous). Assume empty string does not match with anything.

   Example: DNA string1: ATAGTTCCGTCAAA ; DNA string2: GTGTTCCCGTCAAA



   Length of the best continuous length of the DNA string alignment: 12 (TGTTCCGTCAAA)

   a. Implement a solution to this problem using Top-down Approach of Dynamic Programming, name your function **dna_match_topdown(DNA1, DNA2)**
   b. Implement a solution to this problem using Bottom-up Approach of Dynamic Programming, name your function **dna_match_bottomup(DNA1, DNA2)** Write implementation of a & b in a single python file, name your file **DNAMatch.py** Do not write the functions in a Python class.
   c. Explain how your top-down approach different from the bottom-up approach?
      a In the top-down approach, we use memorization to find the longest sub-sequence. During every recursive step, the last characters are checked and if they match, it will increment 'result' by 1, where result is saved into them memo variable as a possible longer sub-sequence. In bottom-up we start from the an empty 2-d array and populate the array with the possible longest lengths of the DNA sub-sequences.
   d. What is the time complexity and Space complexity using Top-down Approach
      a Time and space complexity is O(m*n), where m and n are the lengths of the DNA sequences.
   e. What is the time complexity and Space complexity using Bottom-up Approach
      a Time and space complexity is O(m*n), where m and n are the lengths of the DNA sequences.
   f. Write the subproblem and recurrence formula for your approach. If the top down and bottom-up approaches have the subproblem recurrence formula you may write it only once, if not write for each one separately.

a The subproblem is finding the longest common subsequence between the two DNA sequences, of length len(DNA1)-1 and len(DNA2)-1.

b F(i, j) =

0	if i = 0 or j = 0
1 + F(i - 1, j - 1)	if DNA1[i - 1] == DNA2[j - 1]
max(F(i - 1, j), F(i, j - 1))	otherwise

2. Solve Dynamic Programming Problem and Compare with Naïve approach

You are playing a puzzle. A random number N is given, you have blocks of length 1 unit and 2 units. You need to arrange the blocks back to back such that you get a total length of N units. In how many distinct ways can you arrange the blocks for given N.

a. Write a description/pseudocode of approach to solve it using Dynamic Programming paradigm (either top-down or bottom-up approach)

1. Create an array dp[] of size (N+1) to store the number of arrangements for each length from 0 to N.
2. Initialize dp[0] = 1 and dp[1] = 1, as there is only one way to arrange blocks of length 0 and 1.
3. Iterate from 2 to N:
   a For each length i, the number of arrangements can be calculated by summing up the number of arrangements for lengths (i-1) and (i-2).
   b dp[i] = dp[i-1] + dp[i-2]
4. The final result will be stored in dp[N], representing the number of distinct ways to arrange blocks for length N.

b. Write pseudocode/description for the brute force approach

a

```
function countArrangements(N):
  if N == 0 or N == 1:
    return 1

  count = 0
  count += countArrangements(N-1)
  count += countArrangements(N-2)
```

c. Compare the time complexity of both the approaches

a The time complexity of the dynamic programming approach if O(N) because we iterate from 2 to n once. The time complexity of the brute force if O(2^N) since for each length of N, we recursively explore two branches, N-1 and N-2, until we reach the base case.

d. Write the recurrence formula for the problem

a dp[i] = dp[i-1] + dp[i-2] + 1

Example 1:

Input: N=2, Result: 2
Explanation: There are two ways. (1+1 , 2)

Example 2:

Input: N=3, Result: 3

Explanation: There are three ways (1+1+1, 1+2, 2+1)