Kevin Sekuj
1/25/2022
CS325: Analysis of Algorithms
Homework 3

1. Solve a problem using top-down and bottom-up DP
   a. DNAMatch.py
   b. DNAMatch.py
   c.

The top-down approach is recursive – it involves saving the result of our subproblems and storing them in a data structure. When the same subproblem is encountered again, the already computed value is returned instead of computing the subproblem once more. This approach begins at the topmost problem in the DP structure, working down the subproblems recursively to the base case.

The bottom-up approach is iterative – we approach the overall problem by solving subproblems. Thus, the smaller subproblem is the smallest possible subsequence of the two strings, which begins at 1 character each. We compare strings for a subsequence (or alignment) according to the base cases, progressively increasing our pointers as we go. We iterate backwards through our DP Table to construct our answer, computing the main problem at the [0][0]th index of the table – this is what makes the approach "bottom up".

   d. Top-down DP - Time: O(MN), Space: O(MN)
   e. Bottom-up DP - Time: O(MN), Space: O(MN)
   f.

Subproblem: The longest common string alignment of strings of lengths i and j

Recurrence formula:

$$\text{dp[i]} = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ 1 + dp[i+1][j+1] & *1 \\ \max([dp[i][j+1], dp[i+1][j]) & *2 \end{cases}$$

*1 – if the first characters of DNA1[i] and DNA2[j] do match

*2 if the first characters of DNA1[i] and DNA2[j] do **not** match

2. Solve Dynamic programming and compare with the naïve approach

a. Pseudocode for DP approach

```
def blocksPuzzle(N: int) -> int
   // base cases
   if (N == 1) return 1
   if (N == 2) return 2
   // initialize one dimensional DP array to store subproblems
   // and fill in base cases
   Array dp = [0s for length of N+1]
   dp[1] = 1
   dp[2] = 2
   for i = 3 to i = N+1:
        dp[i] = dp[i-1] + dp[i-2]
   return dp[-1]
```

b. Pseudocode for bruteforce approach

```
def blocksPuzzle(N: int) -> int
   // base cases
   if (N <= 2) return N
   return blocksPuzzle(N – 1) + blocksPuzzle(N – 2)
```

c. Time complexity

The optimal bottom-up DP approach has a time complexity of **O(N)** – step i, represented by array index i, can be determined by the previously calculated subproblems i-1 and i-2.

The brute force approach has a time complexity of **$O(2^n)$** – all possible arrangements of blocks must be calculated up to the value N.

d. Recurrence formula

(From the homework description I assume input is valid, i.e n >= 1)

$$\text{dp[i]} = \begin{cases} 1 & if\ n = 1 \\ 2 & if\ n = 2 \\ dp[i-1] + dp[i-2] \end{cases}$$