Programming Project- Dynamic Programming

Shashank Sathish: sxs180355
Sharanya Sathish: sxs180358
Sreekara Mokshagundam: sxm190039

10/03/2020

Abstract:

The program intends to create a dynamic programming approach to solve the Frog Jump problem. The suggested method was figured out from the recursive solution, where the solution first figures which frog should move first step, will it be Frog P or Frog Q or both the Frog at the same time. The intended behavior is to Minimize the Maximum Distance Between the Frogs. This reduces the length of Rope used by both the frogs.

Recursive Approach:

The below is the pseudocode for Recursive Approach that was later converted to a Dynamic Programming.

```
frogJump(p,q):
        if ( p > m) or ( q > n): // m & n denotes the length of points in P Frog and Q Frog
                return 0
        r = max{frogJump(p+1,q), frogJump(p,q+1), frogJump(p+1,q+1), dist(p,q)}
        temp = ∞
        best = min (temp, r)
        temp = r
        return best
```

The program computes the different position of frogJump along with the value in its position.

Dynamic Programming Approach:

The table required to fill is (m+1) x (n+1) where m is the length or size of P Frog Points and n is the length of points of Frog Q , and the time taken to fill the table is constant time. For filling up the table the program uses O(m*n), when m = n, then we get $O(n^2)$. For traversing to find the minimum value of the table the program uses n time.

T(n) = m*n + n

Overall Running Time is O(m*n) and when both are equal then we get $O(n^2)$.

The table is filled in reverse order, that is it is filled from end as dp[i+1] and dp[j+1] are strictly larger, where i and j are positions of P and Q Frogs.

Consider the below input:

Input: P = [[0,-1],[-1,0]]

Q = [[0,0],[2,0], [0,0],[2,0]]

Expected Output: 3

DP Table –

	Q1	Q2	Q3	Q4
P1	3	3	3	3
P2	3	3	3	3

The DP Table is filled and returns 3.