Design & Analysis of Algorithms

Monsoon Semester III 2020-21

**Lab - 5**  Date: **7 October 2020**

**Topics: Dynamic Programming**

**AIM**

Solving problems using Dynamic Programming.

**INTRODUCTION**

* General, powerful algorithm design technique
* Careful brute force
* Subproblems + ‘reuse’

# EXERCISE

1. **Find nth element of Fibonacci number**

Given a positive number n,

We find value of f0 + f1 + f2 + …. + fn

Remember that

**f0 = 0 seed values**

**f1 = 1 seed values**

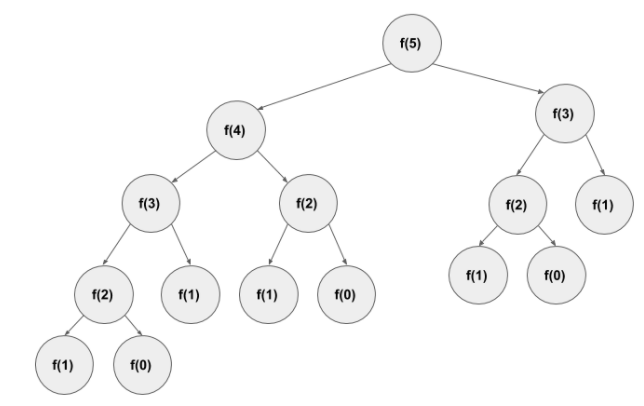
f2 = 1

f3 = 2

f4 = 3

f5 = 5 …

f(n) = f(n-1) + f(n-2)



1. **Brute Force** approach

It is a straight forward approach, finds all the Fibonacci numbers till f(n) and then add them up.

1. **Dynamic Programming Approaches using Memoisation**

**Top-Down approach (Memoization)**

* Top-Down breaks the large problem into multiple sub-problems.
  + - If the sub-problem solved already exists then just reuse the answer.
      * + Otherwise, solve the sub-problem and store the result.

Top-Down uses memoization to avoid re-computing the same sub-problem again.

Solving the same Fibonacci problem using the top-down approach.

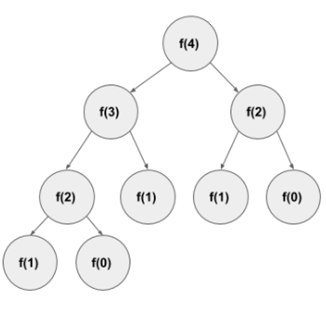
If we want to compute Fibonacci (4), the top-down approach will do the following

Fibonacci (4) = Compute Fibonacci (3) and Fibonacci (2) and return the results.

Fibonacci (3) = Compute Fibonacci (2) and Fibonacci (1) and return the results.

Fibonacci(2) = Compute Fibonacci(1) and Fibonacci(0) and return the results.

Finally, Fibonacci(1) will return 1 and Fibonacci(0) will return 0.



1. **Bottom-Up approach (Tabulation)**

It starts computing result for the sub-problem. Using the sub-problem result solve another sub-problem and finally solve the whole problem.

#### Example

To find the nth value of a Fibonacci series. It starts calculating like

Fibonacci(0) = 0

Fibonacci(1) = 1

Fibonacci(2) = 1 [Fibonacci(0) + Fibonacci(1)]

Fibonacci(3) = 2 [Fibonacci(1) + Fibonacci(2)]

By doing this we can easily find the nth member.

**2. Programming Program to find the longest common sub-sequence using Dynamic**.

If a set of sequences are given, the longest common subsequence problem is to find a common subsequence of all the sequences that is of maximal length

LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.  
LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

* The longest common subsequence problem is a classic computer science problem, the basis of data comparison programs such as the diff-utility, and has applications in bioinformatics. It is also widely used by revision control systems, such as SVN and Git, for reconciling multiple changes made to a revision-controlled collection of files.

## **Analysis**

1. **nth element of Fibonacci number**
2. **Brute Force** approach

Exponential

T(n) = T(n-1) + T(n-2) + Θ(1)

T(n) >= 2T(n-2)

= Θ(2n/2)

1. Top-Down approach (Memoization)

* Memoized calls cost = Θ(1)
* Number of non-memoized calls is n : fib(1), fib(2),……..fib(n)
* Non recursion work per call = Θ(1)
* Time = Θ(n)
* DP ~ guessing + recursion + memorization
* Time = (No. of subproblems(n)) \* {time / subproblems(Θ(1))}
* Don’t count recursion

1. Bottom-Up approach (Tabulation)

* Exactly same computation
* Topological sort of subproblem dependency
* Can often save space

1. **Longest common sub-sequence**

To populate the table, the outer **for** loop iterates ***m*** times and the inner **for** loop iterates ***n*** times. Hence, the complexity of the algorithm is *O(m, n)*, where ***m*** and ***n*** are the length of two strings.

Time = Θ(mn) = const work/entry

* Reconstructs LCS by tracing backwards

Space = Θ(mn)

**CONCLUSION**

Hence, Dynamic Programming methods was implemented and analysed.