Dynamic Programming

Dynamic Programming

- **Dynamic programming like the divide and conquer method, solves problem by combining the solutions of sub problems**

- **Divide and conquer method partition the problem into independent sub problems, solves the sub problems recursively and then combine their solutions to solve the original problem.**

Dynamic Programming

- **Dynamic programming is applicable, when the sub-problems are NOT independent, that is when sub-problems share sub sub-problems.**

- **It is making a set of choices to arrive at optimal solution.**

- **If sub-problems are not independent, we have to further divide the problem.**

- **In worst case, we may end-up with an exponential time algorithm.**

Dynamic Programming

- **Frequently, there is a polynomial number of sub-problems, but they get repeated.**
- **A dynamic programming algorithm solves every sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time the sub-problem is encountered**
- **So we end up having a polynomial time algorithm.**
- **Which is better, Dynamic Programming or Divide & conquer?**

Optimization Problems

- **Dynamic problem is typically applied to <u>Optimization Problems</u>**

- **In optimization problems there can be many possible solutions. Each solution has a value and the task is to find the solution with the optimal (Maximum or Minimum) value. There can be several such solutions.**

4 steps of Dynamic Programming Algorithm

1. **Characterize the structure of an optimal solution.**

2. **Recursively define the value of an optimal solution.**

3. **Compute the value of an optimal solution bottom-up.**

4. **Construct an optimal solution from computed information**

Often only the value of the optimal solution is required so step-4 is not necessary.

Recursive Definition of the Fibonacci Numbers

The Fibonacci numbers are a series of numbers as follows:

fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
...

$$fib(n) = \begin{cases} 1, & n <= 2 \\ fib(n-1) + fib(n-2), & n > 2 \end{cases}$$

fib(3) = 1 + 1 = 2
fib(4) = 2 + 1 = 3
fib(5) = 2 + 3 = 5

Recursive Algorithm (seen earlier)

- **Takes Exponential time, seen few lectures back!**
- **Actual sub problems are polynomial (O(n)) but they get repeated**
- **Sub problems are not INDEPENDENT.**
- **Sub problems share sub-sub problems.**
- **We can solve it using Dynamic programming.**

Memoization\Caching

```
Dictionary m;
m[0] = 0, m[1] = 1
Integer fib(n)
    if m[n] == null
        m[n] = fib(n-1) + fib(n-2)
    return m[n]
```

Bottom Up Approach

```
int fib(int n)
{
  /* Declare an array to store Fibonacci numbers.
*/
  int f[n+2];   // 1 extra to handle case, n = 0
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
      /* Add the previous 2 numbers in the series
         and store it */
      f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}
```