GeeksforGeeks
A computer science portal for geeks

Custom Search

Suggest a Topic

Write an Article

Overlapping Subproblems Property in Dynamic Programming | DP-1

Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into subproblems and stores the results of subproblems to avoid computing the same results again. Following are the two main properties of a problem that suggests that the given problem can be solved using Dynamic programming.

In this post, we will discuss first property (Overlapping Subproblems) in detail. The second property of Dynamic programming is discussed in next post i.e. Set 2.

1) Overlapping Subproblems
2) Optimal Substructure

1) Overlapping Subproblems:

Like Divide and Conquer, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that these don’t have to be recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. For example, Binary Search doesn’t have common subproblems. If we take an example of following recursive program for Fibonacci Numbers, there are many subproblems which are solved again and again.

```
/* simple recursive program for Fibonacci numbers */
int fib(int n)
{
   if ( n <= 1 )
      return n;
   return fib(n-1) + fib(n-2);
}
```

Run on IDE

Recursion tree for execution of *fib(5)*

```
                         fib(5)
                      /             \
               fib(4)                fib(3)
             /      \                /     \
         fib(3)      fib(2)         fib(2)    fib(1)
        /     \        /    \       /    \
  fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
  /    \
fib(1) fib(0)
```

We can see that the function fib(3) is being called 2 times. If we would have stored the value of fib(3), then instead of computing it again, we could have reused the old stored value. There are following two different ways to store the values so that these values can be reused:

a) Memoization (Top Down)

b) Tabulation (Bottom Up)

a) Memoization (Top Down): The memoized program for a problem is similar to the recursive version with a small modification that it looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL. Whenever we need the solution to a subproblem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise, we calculate the value and put the result in the lookup table so that it can be reused later.

Following is the memoized version for nth Fibonacci Number.

C/C++

```
/* C/C++ program for Memoized version for nth Fibonacci number */
#include<stdio.h>
#define NIL -1
#define MAX 100

int lookup[MAX];

/* Function to initialize NIL values in lookup table */
void _initialize()
{
  int i;
  for (i = 0; i < MAX; i++)
    lookup[i] = NIL;
}

/* function for nth Fibonacci number */
int fib(int n)
{
   if (lookup[n] == NIL)
   {
      if (n <= 1)
         lookup[n] = n;
      else
         lookup[n] = fib(n-1) + fib(n-2);
   }
```

```
    return lookup[n];
}

int main ()
{
  int n = 40;
  _initialize();
  printf("Fibonacci number is %d ", fib(n));
  return 0;
}
```

Run on IDE

Java

```
/* Java program for Memoized version */
public class Fibonacci
{
  final int MAX = 100;
  final int NIL = -1;

  int lookup[] = new int[MAX];

  /* Function to initialize NIL values in lookup table */
  void _initialize()
  {
    for (int i = 0; i < MAX; i++)
        lookup[i] = NIL;
  }

  /* function for nth Fibonacci number */
  int fib(int n)
  {
    if (lookup[n] == NIL)
    {
      if (n <= 1)
          lookup[n] = n;
      else
          lookup[n] = fib(n-1) + fib(n-2);
    }
    return lookup[n];
  }

  public static void main(String[] args)
  {
    Fibonacci f = new Fibonacci();
    int n = 40;
    f._initialize();
    System.out.println("Fibonacci number is" + " " + f.fib(n));
  }

}
// This Code is Contributed by Saket Kumar
```

Run on IDE

Python

```
# Python program for Memoized version of nth Fibonacci number

# Function to calculate nth Fibonacci number
def fib(n, lookup):

    # Base case
    if n == 0 or n == 1 :
        lookup[n] = n

    # If the value is not calculated previously then calculate it
    if lookup[n] is None:
        lookup[n] = fib(n-1 , lookup)  + fib(n-2 , lookup)

    # return the value corresponding to that value of n
    return lookup[n]
# end of function

# Driver program to test the above function
def main():
    n = 34
    # Declaration of lookup table
    # Handles till n = 100
    lookup = [None]*(101)
    print "Fibonacci Number is ", fib(n, lookup)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

C#

```
// C# program for Memoized versionof nth Fibonacci number
using System;

class GFG
{

    static int MAX = 100;
    static int NIL = -1;
    static int []lookup = new int[MAX];

    /* Function to initialize NIL
    values in lookup table */
    static void initialize()
    {
        for (int i = 0; i < MAX; i++)
            lookup[i] = NIL;
    }

    /* function for nth Fibonacci number */
    static int fib(int n)
    {
        if (lookup[n] == NIL)
        {
        if (n <= 1)
            lookup[n] = n;
        else
            lookup[n] = fib(n - 1) + fib(n - 2);
        }
        return lookup[n];
    }

    // Driver code
    public static void Main()
    {

        int n = 40;
        initialize();
        Console.Write("Fibonacci number is" + " " + fib(n));
    }
}

// This Code is Contributed by Sam007
```

Run on IDE

b) Tabulation (Bottom Up): The tabulated program for a given problem builds a table in bottom up fashion and returns the last entry from table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3) and so on. So literally, we are building the solutions of subproblems bottom-up.

Following is the tabulated version for nth Fibonacci Number.

C/C++

```
/* C program for Tabulated version */
#include<stdio.h>
int fib(int n)
{
  int f[n+1];
  int i;
  f[0] = 0;   f[1] = 1;
  for (i = 2; i <= n; i++)
      f[i] = f[i-1] + f[i-2];

  return f[n];
}

int main ()
{
  int n = 9;
  printf("Fibonacci number is %d ", fib(n));
  return 0;
}
```

Run on IDE

Java

```
/* Java program for Tabulated version */
public class Fibonacci
{
  int fib(int n)
  {
    int f[] = new int[n+1];
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i <= n; i++)
          f[i] = f[i-1] + f[i-2];
    return f[n];
  }

  public static void main(String[] args)
  {
    Fibonacci f = new Fibonacci();
    int n = 9;
    System.out.println("Fibonacci number is" + " " + f.fib(n));
  }
}
// This Code is Contributed by Saket Kumar
```

Run on IDE

Python

```python
# Python program Tabulated (bottom up) version
def fib(n):

    # array declaration
    f = [0]*(n+1)

    # base case assignment
    f[1] = 1

    # calculating the fibonacci and storing the values
    for i in xrange(2 , n+1):
        f[i] = f[i-1] + f[i-2]
    return f[n]

# Driver program to test the above function
def main():
    n = 9
    print "Fibonacci number is " , fib(n)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh (nickzuck_007)
```

Run on IDE

C#

```csharp
// C# program for Tabulated version
using System;

class GFG
{
    static int fib(int n)
    {
        int []f = new int[n + 1];
        f[0] = 0;
        f[1] = 1;
        for (int i = 2; i <= n; i++)
            f[i] = f[i - 1] + f[i - 2];
        return f[n];
    }

    public static void Main()
    {

        int n = 9;
        Console.Write("Fibonacci number is" + " " + fib(n));
    }
}

// This Code is Contributed by Sam007
```

Run on IDE

PHP

```php
<?php
// PHP program for Tabulated version

function fib($n)
{
    $f[$n + 1]=0;
    $i;
    $f[0] = 0;
    $f[1] = 1;
    for ($i = 2; $i <= $n; $i++)
        $f[$i] = $f[$i - 1] +
                 $f[$i - 2];

    return $f[$n];
}

// Driver Code
$n = 9;
echo("Fibonacci number is ");
echo(fib($n));

// This code is contributed by nitin mittal.
?>
```

Run on IDE

Output:

```
Fibonacci number is 34
```

Both Tabulated and Memoized store the solutions of subproblems. In Memoized version, table is filled on demand while in Tabulated version, starting from the first entry, all entries are filled one by one. Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. For example, Memoized solution of the LCS problem doesn't necessarily fill all entries.

To see the optimization achieved by Memoized and Tabulated solutions over the basic Recursive solution, see the time taken by following runs for calculating 40th Fibonacci number:

Recursive solution

Memoized solution

Tabulated solution

Time taken by Recursion method is much more than the two Dynamic Programming techniques mentioned above – Memoization and Tabulation!

Also, see method 2 of Ugly Number post for one more simple example where we have overlapping subproblems and we store the results of subproblems.

We will be covering Optimal Substructure Property and some more example problems in future posts on Dynamic Programming.

Try following questions as an exercise of this post.

1) Write a Memoized solution for LCS problem. Note that the Tabular solution is given in the CLRS book.

2) How would you choose between Memoization and Tabulation?

Dynamic Programming | Set 1 (Overlapping Subproblems Property) | Geeksfor…

Dynamic Programming | Set 1 (Solution using Memoization) | GeeksforGeeks

Dynamic Programming | Set 1 (Solution using Tabulation) | GeeksforGeeks

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:
http://www.youtube.com/watch?v=V5hZoJ6uK-s

Improved By : nitin mittal

Practice Tags : Dynamic Programming Fibonacci

Article Tags : Dynamic Programming Fibonacci

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Overlapping Subproblems Property in Dynamic Programming | DP-1
Ugly Numbers
How to solve a Dynamic Programming Problem ?

Weird Number
Semiperfect Number
Maximum sum in circular array such that no two elements are adjacent
Number of ways to reach Nth floor by taking at-most K leaps
Number of ways to represent a number as sum of k fibonacci numbers
Minimum number of single digit primes required whose sum is equal to N
Minimum cost to reach the top of the floor by climbing stairs

(Login to Rate)

1.6 Average Difficulty : **1.6/5.0**
Based on **316** vote(s)

Add to TODO List

Mark as DONE

Basic Easy Medium Hard Expert

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!