Dynamic Programming | Set 1 (Overlapping Subproblems Property)

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 suggest that the given problem can be solved using Dynamic programming.

1) Overlapping Subproblems
2) Optimal Substructure

1) Overlapping Subproblems:

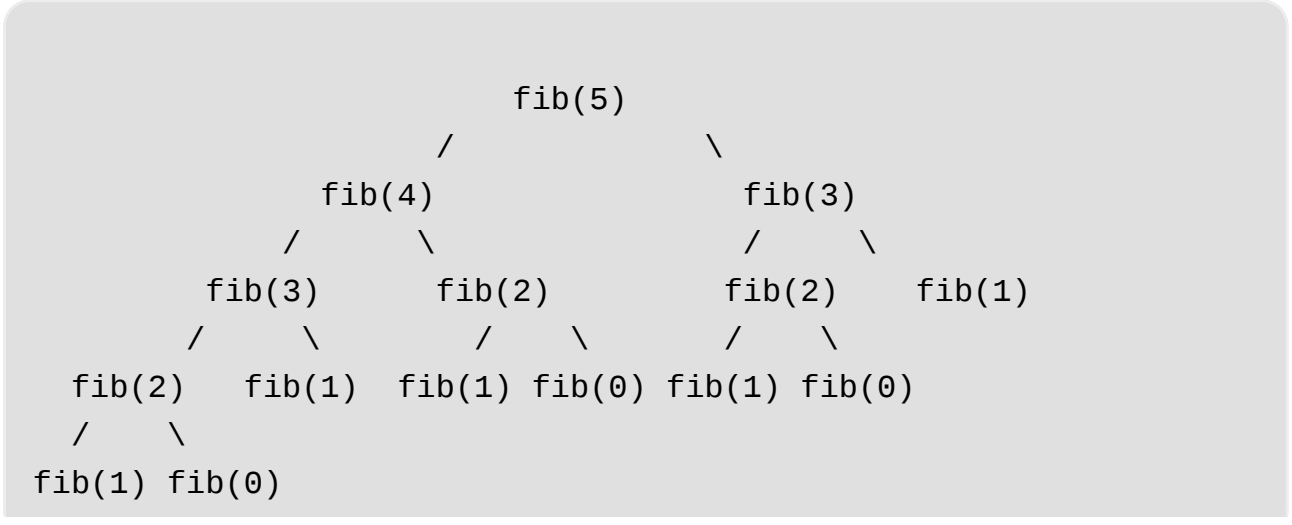
Like Divide and Conquer, Dynamic Programming combines solutions to sub-

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 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 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)*



Popular Posts

- Top 10 Algorithms and Data Structures for Competitive Programming
- Top 10 algorithms in Interview Questions

We can see that the function f(3) is being called 2 times. If the function had stored the value of f(3), then instead of computing it again, we would have reused the old stored value.
There are two 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 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 lookup table so that it can be reused later.

Following is the memoized version for nth Fibonacci Number.

C/C++ Python

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

int lookup[MAX];

/* Function to initialize NIL values in lookup table */
void  initialize()
```

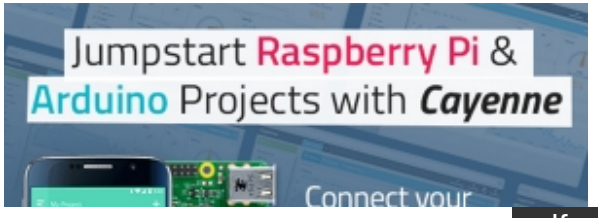
- How to begin with Competitive Programming?
- Step by Step Guide for Placement Preparation
- Reflection in Java
- Memory Layout of C Programs
- Heavy Light Decomposition
- Sorted Linked List to Balanced BST
- Generics in Java
- Aho-Corasick Algorithm for Pattern Searching
- Insertion Sort , Binary Search , QuickSort , MergeSort , HeapSort

```
{
  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

in bottom up fashion and returns the last entry from table.

C/C++ Python



```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

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 LCS problem doesn't necessarily fill all entries.

Tags

Accolite Adobe Advance Data Structures Advanced Data Structures Amazon array D-E-Shaw Directi Divide and Conquer Dynamic Programming Flipkart GATE GATE-CS-DS-&-Algo GATE-CS-Older GFacts Goldman Sachs Google Graph Greedy Algorithm Hashing Interview Experience Java MakeMyTrip MAQ Software MathematicalAlgo Matrix Microsoft Morgan Stanley Operating systems Oracle Pattern Searching Recursion Samsung SAP Labs SnapDeal stack STL Zoho

To see the optimization achieved by memoized and tabulated versions over the basic recursive version, see the time taken by following runs for 40th Fibonacci number.

Simple recursive program
Memoized version
tabulated version

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 version for LCS problem. Note that the tabular version is given in the CLRS book.
2) How would you choose between Memoization and Tabulation?

All Categories

Select Category

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

Company Wise Coding Practice Topic Wise Coding Practice

81 Comments Category: Dynamic Programming Tags: Dynamic Programming

Related Posts:

- All ways to add parenthesis for evaluation
- Count all increasing subsequences
- Count distinct occurrences as a subsequence
- Longest repeating and non-overlapping substring

- Find minimum sum such that one of every three consecutive elements is taken
- Count Distinct Subsequences
- Non-crossing lines to connect points in a circle
- Count digit groupings of a number with given constraints

Previous post in category Next post in category

(Login to Rate and Mark)

1.6 Average Difficulty : **1.6/5.0**
Based on **115** vote(s)

Add to TODO List

Mark as DONE

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

81 Comments GeeksforGeeks

Recommend 3 Share

Join the discussion…

Kaushal Gosaliya • a month ago

Solution to Java Language ::

http://code.geeksforgeeks.org/...

Reply • Share ›

Kaushal Gosaliya • a month ago

JAVA Solution ::

```
import java.io.*;
import java.util.*;

public class First {

private static int MAX = 0;
private static int look[];

static void Initialization(){
for(int i=0;i<max;i++) look[i]="-1;" }="" static="" int="" fib(int="" n){=
}else{="" look[n]="fib(n-1)" +="" fib(n-2);="" }="" }="" return="" look[
args[]){="" scanner="" sc="new" scanner(system.in);="" int="" n='
initialization();="" system.out.println("="" ans="" is="" ::="" "="" +="
```

Reply • Share ›

Kaushal Gosaliya ➚ Kaushal Gosaliya • a month ago

This is optimal solution of Fibonacci number

Reply • Share ›

This comment is awaiting moderation. Show comment

This comment is awaiting moderation. Show comment.

Kaushal Gosaliya ➚ Mahendra • a month ago

Thank you for suggestions

Link is ::

http://code.geeksforgeeks.org/...

Reply • Share ›

Shiva Kumar • 9 months ago

In C:

We can avoid #define MAX 100 and #define NIL -1.
Using calloc we can avoid NIL and there won't be need to intialize

http://code.geeksforgeeks.org/...

Also, the problem can be solved in O(n) time with O(1) extra spac

Reply • Share ›

DAVINDER PAL SINGH ➚ Shiva Kumar • 4 months ago

We can solve Fibonacci problem in O(logn) time, see Matl

Reply • Share ›

Ravindra Kholia ➚ DAVINDER PAL SINGH • 2 months ag

thats counting the number of fibonacci terms in giv
range

• Reply • Share ›

Mains ➚ Shiva Kumar • 5 months ago

if(lookup[n] == 0)
lookup[n] = _fib(n-1, lookup) + _fib(n-2, lookup);
return lookup[n];

This is dynamic initialization.

• Reply • Share ›

Vinoth • 10 months ago

In this fibonacci problem, can someone explain me how to calcula
recursion, memoization, bottom-up.

Thanks in advance

• Reply • Share ›

Mriganka Ghosh ➚ Vinoth • 7 months ago

Recursion:

Let's consider the fibonacci tree structure that forms as pa
nodes fib(n-1) and fib(n-2). So we can see that this tree is
Let's assume the first function (node) fib(n) for a positive n
to the problem fib(n-1) and fib(n-2). Going by the same log
which is the base case of fibonacci. Meaning the tree stop

Hence, the height of the solution tree is 'h'.
We know, height h is given by:

We know, height h is given by:

h = log (N) [log base 2]
where, N = total number of nodes in the tree.

Thus, N = 2 ^ h

Now, we know that h = n (because to find fib(n) we have to

Which gives the final solution, N = 2^n
And complexity : O (N) or O(2^n) [Exponential]

see more

2 ▲ ▼ • Reply • Share ›

hassan ➚ Mriganka Ghosh • 4 months ago
thanks,for giving clear description.....
▲ ▼ • Reply • Share ›

Rahul Singal • 10 months ago
http://shadowhackit.blogspot.i...
must see these tricks and knowledgeable stuff
▲ ▼ • Reply • Share ›

Asen • a year ago
Is there optimal substructure property in fibonacci no
▲ ▼ • Reply • Share ›

shiva • a year ago

check http://cedric005.blogspot.in/2...
to continuously check output of programs just copying text.
easy to use

Reply • Share ›

Joshua Lamusga • a year ago

For reference, recursive time is 0.88s and memoized/tabulated at

Reply • Share ›

Akshay Aradhya • a year ago

The program to compute the 40th Fibonacci Number, is 4 years o
Update it. Thank You

1 Reply • Share ›

user007 Akshay Aradhya • a year ago

But, it does show the time of execution at the bottom. Prob

Reply • Share ›

Holden user007 • a year ago

there is no time of execution there ...

Reply • Share ›

dumborakesh Holden • a year ago

Success #stdin #stdout 0.88s 2728KB

1 Reply • Share ›

Guangming Wang • a year ago

Here is my space optimized O(1) solution using tabulation:

```
// bottom up space optimized
public static int fibTabulationOptimized(int n){
if(n < 2){
return n;
}
int prePre = 0, pre =1;
int ret = 0;
for(int i=2; i <=n; i++){
ret = prePre + pre;
prePre = pre;
pre = ret;
}

return ret;
}
```

Hatem Faheem ➚ Guangming Wang • a year ago

Sure this solution is more memory optimized, but the mair function (global) and check it everytime you call the functic

So, if I call fib(50) it will fill the table from 0 to 50 then any c O(1) by checking the table.

Nguyễn Minh Tuấn ➚ Hatem Faheem • a month ago

you good.. thanks for explain...

^ | v • Reply • Share ›

Vibhor Garg • 2 years ago

In the memoization part, there should be a precondition to check t loop should run.

1 ^ | v • Reply • Share ›

Anil Bhaskar • 2 years ago

The tabulation method is a simple iterative way of computing Fibo

^ | v • Reply • Share ›

Navroze • 2 years ago

The answer to the second question is that we can use memoizati calculation and the tabulated version can be used in case we kno

^ | v • Reply • Share ›

Jeremy Shi • 2 years ago

There is another O(lgn) solution .

^ | v • Reply • Share ›

Ishan Gupta ➚ Jeremy Shi • 5 months ago

Yeah, it's Matrix Exponentiation. It's a great method for cor preparation.

• Reply • Share ›

vaibhav • 2 years ago

how is this code handling negative numbers like if i give n = -40 th
please explain ?

• Reply • Share ›

Manraj Singh vaibhav • 2 years ago

How can you find fibonacci number -40th?

• Reply • Share ›

vaibhav Manraj Singh • 2 years ago

if (n <= 1)

lookup[n] = n;

this means that if i give n =-20 it will return lookup[-
possible only yet the output on n=-20 is 0

1 • Reply • Share ›

Sriram Ganesh vaibhav • 2 years ago

I understsand ur question. Its not relevant to

• Reply • Share ›

Manraj Singh vaibhav • 2 years ago

It will give Segment fault!

1 • Reply • Share ›

SHIVAM DIXIT • 2 years ago

Can you please provide a post on space optimised knapsack?? In we dont have to take such a large 2-d array....Its sometimes not possible to take array of size arr[n+1][capacity+1] for dp...example for such a problem is http://www.spoj.com/problems/L...

 • Reply • Share ›

Goky ➚ SHIVAM DIXIT • 2 years ago

What we can do is we take array as arr[2][capacity+1].If yo row just one above the current row.So instead of using me current and prev row only.So after computing current row, row.We will continue to work this way till N items.If you hav

2  • Reply • Share ›

Sivakumar K R ➚ SHIVAM DIXIT • 2 years ago

```
int knapSack1(int W, int *wt, int *val, int n)
{
int dp[W + 1];
memset(dp, -1e9, sizeof dp);
for (int i = 0; i < n; i++)
{
for (int j = W; j >= 0; j--)
{
int k = j + wt[i];
if (k <= W)
{
```

```
{
dp[k] = max(dp[k], dp[j] + val[i]);
}
}
cout<<'\n';
}
cout << '\n';
/*for (int j = 0; j <= W; j++)
cout << dp[j] << " ";*/
return dp[W];
}
```

^ | v • Reply • Share ›

lucky • 2 years ago

lookup must be of long type

2 ^ | v • Reply • Share ›

arsh • 2 years ago

First two links are broken

http://www.cs.uiuc.edu/class/f...
http://web.iiit.ac.in/~avidull...

^ | v • Reply • Share ›

GeeksforGeeks Mod ➦ arsh • 2 years ago

Thanks for pointing this out. Looks like the pdf files have be
post.

• Reply • Share ›

quantized • 2 years ago

First two links in References are broken.

2 • Reply • Share ›

dg • 2 years ago

Please give comparison between top down approach(memorizati
As in memorization(top down) we can avoid computation of some
in bottom up approach we need to find solution of every sub proble
And how should we choose between Memoization and Tabulation

3 • Reply • Share ›

Ealham Al Musabbir dg • 2 months ago

When the number of recursive call gets too large to reach
example, http://www.spoj.com/problems/M... , this problem
only, but for larger input (i.e greater than 1000), stack mem
the only way.

• Reply • Share ›

ankit • 3 years ago

@GFG
which approach is better bottom-up or top-down?

• Reply • Share ›

Anand Barnwal ankit • 2 years ago

Memoization(top down) usually becomes more complicate

problems, mainly those which you do not need to compute answer like: LCS.

Tabulation(bottom up) is easy to implement but it may con benefits of less overhead.

2 ^ | v • Reply • Share ›

rihansh ➚ Anand Barnwal • 2 years ago

Agree:D

^ | v • Reply • Share ›

Gaurav Kapoor • 3 years ago

With Normal recursion OUTPUT for n=30 : time taken to calculate is 832040

With Memorization OUTPUT for n=30 : time taken to calculate the 832040

Why its taking less time in Recursion ?? It should be the reverse \

12 ^ | v • Reply • Share ›

Ramendu Shandilya ➚ Gaurav Kapoor • a year ago

Afait, it will prove better for higher values of n, i.e. the asym recursion approach ! Try comparing 2^n and n^3 for higher

^ | v • Reply • Share ›

mateor ➚ Gaurav Kapoor • a year ago

Increase your n=50 and tell us which is faster. In my recoll recursion is reasonable with n=30.

 • Reply • Share ›

aa1992 Gaurav Kapoor • 2 years ago

i think because a lot of unnecessary computation is done i the space complexity.

 • Reply • Share ›

coder Gaurav Kapoor • 2 years ago

because in recursion there are so many steps involved in (return the value to calling function

 • Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy