Recursion - Sums in a Triangle

REVISIONS VIEW

An Introduction

Recursion, by definition, is a method of defining a function in a way such that the function being defined is applied within its own definition. A lot of problems in computer science can be broken down into smaller sub-problems. Most of these can have recursive solutions where the answer for each state is calculated from answers of smaller sub-states. Recursion, however, is very inefficient and most of the times, the answers for a particular state are calculated again and again. To overcome this limitation, a technique called memoization can be used. Memoization is an optimization technique used primarily to speed up computer programs by having function calls avoid repeating the calculation of results for previously-processed inputs.

Thus, if the answers for each visited state are stored in a cache of sorts in the recursive solution, we can avoid re-calculating values we have already calculated.

We will see how to use these techniques to solve one of the easy level problems on Codechef.

Problem Tutorial - Sums in a Triangle

Sums in a Triangle http://www.codechef.com/problems/SUMTRIAN represents a broad range of problems that can be solved by using a recursive approach. As such, we will see one of the methods to solve it.

The problem asks one to take as input the number of test cases as the first input. Each test case consists of the number of rows 'n' and then 'n' lines follow containing each row.

The first row has 1 number, the second has 2 numbers, the 3rd has 3 numbers and so on. Now, We have to find a path from row 1 to row 'n' such that the cost of the path is maximized.

The cost of a path is the sum of all the numbers that make up the path. The additional constraint is that from a particular cell, we can only go to a cell in the next row directly beneath the cell or to the one situated to the right of the one beneath it. We start off in the topmost row and in its left most cell.

A very naive way to do this is to generate all paths, find the cost of the paths and choose the best one. Such an approach would time out because we can have a max of 100 rows.

Now, to model a problem in a recursive manner, we need subproblems which are similar to the problem to be solved. The prerequisites to modelling a recursive solution are

1. There should be subproblems

2. There should be terminating conditions called base conditions.

3. The sub-problem to be solved must be the same as the parent problem, but of a smaller magnitude or size.

4. There should be no cycles. One should not be able to reach a state, by starting off from it.

To solve this problem, we will try to see if it satisfies the prerequisites for a recursive problem.

1. First of all, we need to find subproblems. Consider a particular cell (i, j). From this cell, we can go either to cell (i + 1, j) or (i + 1, j + 1), where the first index represents the row and the second one represents the columns. Now, we need to maximize the sum for paths from cell (0, 0). Now, the maximum path value for cell (0, 0) will equal the value at cell(0, 0) + max(value of max path from cell(0, 1), value of max path from cell(1, 1)) Thus, we get the subproblems that we were looking for. The max path value at a particular cell equals the value at that cell + the max path values of cells reachable from it.

2. We can fix the terminating conditions as follows. The value of the max path if we reach a row beyond the 'n' rows specified is 0. This becomes our stopping condition for each path.

3. The subproblem to be solved is the same as the original problem. At each step we are making the size of the rows to be checked lesser and lesser and moving towards our base condition.

4. There are no cycles. There won't be a path such that we start from cell (i, j) and move on to some cells and reach cell(i, j) again. This can be seen by the fact that at each step, the row number keeps on increasing. It never decreases also, it never remains the same.

Thus, having modelled it as a recursive solution, we can create a recursive solution to the problem as follows.

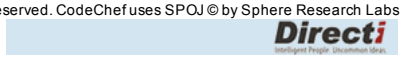
```
Function solve(i, j)

if i is greater than 'n'

return 0

t1 equals solve(i + 1, j)

t2 equals solve(i + 1, j + 1)
```



The time now is: 04:58:18 PM