Stock Maximize

Dynamic Programming Basics

Introduction

Dynamic programming is one of the problem solving paradigms where we try to find a solution by breaking the larger problem into subproblems, and exploit the fact that the optimal solution to the problem depends upon the optimal solution to its subproblems.

The following two charateristics suggest that the given problem can be solved using dynamic programming :

1. **Overlapping subproblems :** It ensures that the given problem can be divided into a number of subproblems of similar type, but of smaller size than the original one.
2. **Optimal Substructure Property :** It ensures that the optimal solution to the problem can be formulated from the optimal solution to its subproblems.

We can also compare with a greedy approach that the local optimum choice may not always be the global optimal choice. An example is finding the shortest path between two cities. A local junction may appear shorter but turn out to be long, while a longer junction may turn out to be a shorter route over all.

Dynamic Programming has two methods that can be used according to the problem at hand.

Top-Down Approach

A top-down DP approach tries to solve the bigger problem by recursively finding the solution to smaller problems while also storing local solutions in a look-up table such that those are not computed each time. This technique is called **Memo-ization**

An example :-

Problem statement : *Let us define a recurrence* F.

$$F(n) = F(n-1) + F(n-3), if(n >= 3)$$

$$F(n) = 7, otherwise$$

Given n find the n^{th} term of the recurrrence ?

Solution using Top - Down Approach :

```
int F[MAXSIZE] ;
// set F to some unique value like -1 in this case.
int solve(int n){
    if(n < 3){
        return 7 ;      // recurrence defination
    }
    int &ret = F[n] ;   // cache technique
    if(ret != -1)       // absence of -1 indicate that this is already computed
        return ret ;            // use this computed result
    ret = solve(n-3) + solve(n-1) ;         // compute otherwise
    return F[n] = ret ;      // final return computed answer
}
```

Bottom Up Approach

This technique is the opposite of the former and here we start from the smallest solution and go up to the required solution. It is assumed that coming up with a bottom up solution is relatively easy.

An example :-

Problem statement : *Let us define a recurrence* F.

$$F(n) = F(n-1) + F(n-3), if(n >= 3)$$

$$F(n) = 7, otherwise$$

Given n find the n^{th} term of the recurrrence ?

Solution using Bottom - Up Approach :

Solution using Bottom-Up Approach :

```
int F[MAXSIZE] ;
int solve(int n){
    F[0] = F[1] = F[2] = 7 ;    // recurrence definition
    for(int i=3;i<=n;i++)
        F[i] = F[i-1] + F[i-3] ;    // recurrence definition
    return F[n] ;
}
```

Uses

Dynamic programming can be used to solve

- Recurrence Relations
- Optimization Problems

Related challenge for **Dynamic Programming Basics**

String Reduction

Success Rate: **56.83%** Max Score: **70** Difficulty:

Solve Challenge