Assignment 03
Dynamic Programming

Please submit your solutions in PDF format. The PDF must be typed, NOT handwritten. Solution for each problem must start on a new page. The solutions should be concise; complicated and half-witted solutions might receive low marks, even when they are correct. Solutions should be submitted on the course website.

Problem 1: Collaborators [2 points]

List the name of the collaborators for this assignment. If you did not collaborate with anyone, write "None" (without the quotes).

Solution:

Algo Rythm

Problem 2: Fair and Square [4 points]

You are given a bag with m coins of different denominations that sums up to a value, $total$. You need to divide the coins between two friends in such a way that both of them have a fair share, that means the difference between the amount these two gets should be minimum.

(a) [2 points] What is the maximum amount a friend can get?

Solution:

$\lceil \frac{total}{2} \rceil$

(b) [2 points] What is the difference between the amount these two friends will get?

Solution:

Let, one friend gets M. The different will be $(total - (2 \times M))$

Problem 3: Parenthesization? Nope! [11 points]

According to Wikipedia, C_n is the number of different ways $n+1$ factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator). For $n = 3$, for example, we have the following five different parenthesizations of four factors:

- $((ab)c)d$
- $(a(bc))d$
- $(ab)(cd)$
- $a((bc)d)$
- $a(b(cd))$

We are interested in finding C_n, the nth Catalan number, which can be recursively defined as:

$$C_1 = 1$$

$$C_n = \sum_{i=1}^{n-1} C_i C_{n-i}$$

We can use the following code to find the nth Catalan Number:

```
int cat(int n)
{
    int c = 0;
    if(n == 1)
        c = 1
    else
    {
        for(int i = 1; i < n; i++)
            c += cat(i) * cat(n - i);
    }
    return c;
}
```

For simplicity, we are only interested in the Catalan Numbers that can be stored in memory. But the algorithm can be generalized for larger values too.

(a) [4 points] Analyze the running time of the given code.

Solution:

A call to `cat(n)` performs two recursive calls to `cat(n-1)` along with many others. Thus the running time is lower bounded by:

$$T(n) > 2T(n-1)$$
$$T(n) = \Omega(2^n)$$

Rubric: 2 points for correct reduction to smaller problems. 2 points for correct runtime.

(b) [7 points] Describe the DAG to compute the first n Catalan Numbers using the recursive definition.

Solution:

The graph of dependencies consists of n vertices, with vertex v_i representing Catalan number C_i. For vertex v_k, there is a directed edge from v_k to vertex v_i for all $1 \le i < k$, since the recursive definition depends on every smaller Catalan number. Thus, there are $\sum\limits_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ directed edges in the graph.

Problem 4: Capsicum [23 points]

Arun Nahaj loves bell peppers, known locally as capsicums. That's why he planted n capsicum plants in a line at his garden. Due to his excitement, he forgot to keep enough space between the plants. After a while, the plants started growing capsicums. To be specific, the ith plant grew c_i capsicums. But the plants were not getting enough nutrition and became weak. Arun deduced that the reason might be the small distance between the consecutive plants. So he decided to uproot a few of the plants so that the others can grow properly. For each plant, Arun will either leave it as is or uproot the plant. In case he doesn't uproot the ith plant, he will uproot both the neighboring plants, $(i-1)$, and $(i+1)$. Since the capsicums have not matured properly, Arun wants to uproot the plants in such a way that the number of capsicums in the uprooted plants is minimized. Help him figure out which plants to uproot and which one to leave as is.

Solution:

Subproblem

Let, $c(i)$ be the minimum number of capsicums from the uprooted plants in range 1 to i, where the ith plant can be left as is or uprooted.

Relate
We can either uproot the ith plant or leave it be. If we uproot the ith plant, then we can either uproot the $(i-1)$th plant or leave it as is. Here, $c(i) = p_i + c(i-1)$. If we leave the ith plant as is, then we must uproot the $(i-1)$th plant and we can either uproot the $(i-2)$th plant, or leave it be. Here, $c(i) = p_{i-1} + c(i-2)$.
Since we want to minimize the number of capsicums from the uprooted plants, $c(i) = \min(p_i + c(i-1), p_{i-1} + c(i-2))$. Here $c(i)$ only depends on the subproblems with strictly smaller i. So the dependency graph does not contain any cycle.
Base
We can consider $c(0) = c(1) = 0$. That means, if we have zero or one plant, we do not need to uproot any of them.
Solution
Find $c(n-1)$.
To find out the plants to be uprooted, we can keep parent pointer to reconstruct the optimal solution.
Time
Here, the # of subproblems depend on the value of i, which is in range $[0, n]$. So total subproblems $n+1$. Time per subproblem is $O(1)$, since we simply perform recursive calls and storing. Total running time: $O(n)$.