Algorithms III

Dynamic Programming

David Esparza Alba

February 8, 2021

Universidad Panamericana

Outline

1. What is Dynamic Programming?

2. DP Examples

3. Exercises

4. Conclusion

5. Famous DP Problems

6. Largest Increasing Sub-sequence (LIS)

What is Dynamic Programming?

Definition

- DP can be defined as a tool that uses previously calculated values to obtain a new value.
- In other words, it uses what is already know in time t to answer a question in time $t + 1$.

Characteristics

Most of the DP problems have these two following properties:

1. A recursive function. A way to express a new value using previously obtained values.
2. Memory usage. Is common the use of arrays and multidimensional arrays to store the information needed to compute a new value.

DP Examples

Factorial

The factorial of number of a number n is defined as:

$$n! = 1 \times 2 \times \cdots \times n - 1 \times n \tag{1}$$
$$= (n-1)! \times n \tag{2}$$

If we write this as a programmatic function it would look like this:

$$f(n) = f(n-1) \cdot n$$

where $f(n)$ is a function that returns the factorial of n.

Factorial Implementation

```
int f(unsigned int n) {
  if (n == 0) {
    return 1;
  }

  return n * f(n - 1);
}
```

Fibonacci Sequence

The First two elements of the Fibonacci sequence are $1, 1$, after that the next elements are calculated by the sum of their two previous elements. So the first 10 numbers of the Fibonacci sequence are:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Fibonacci Sequence as Function

If we see the Fibonacci sequence as a function, it would look like this:

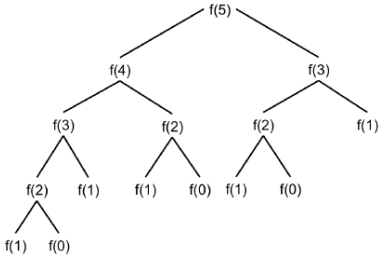
$$fibo(n) = \begin{cases} 1 & n \in \{1, 2\}, \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

Fibonacci Implementation

```
int fibo(int n) {
  if (n == 0 || n == 1) {
    return 1;
  }

  return fibo(n - 1) + fibo(n - 2);
}
```

Fibonacci - Is recursion the right approach?



Fancy Thing from *Source et al.*

- Some
- Interesting
- Points

A better approach

Use DP. Store into memory all Fibonacci values computed, that way

```
fibo[n] = fibo[n-1] + fibo[n-2]
```

where ***fibo*** is a vector or array.

Exercises

Domino Tiles

There are two sizes of domino pieces, 2×1, and 2×2, in how many ways can we fill a board of size $2 \times n$?

UP-Numbers

An UP-number is a number that doesn't contain two consecutive pairs in its digits. For example: 314385

Conclusion

Conclusions

There is no easy way to learn how to use DP, it more like a habit that must be acquired trough practice and solving new problems, and gradually it becomes easier to identify when is a good idea to implement a DP solution.

Famous DP Problems

Famous DP Problems

- Knapsack problem (Mochila)
- Largest Increasing Sub-sequence
- Largest Common Sub-sequence
- Edit Distance
- Coin change problem
- Optimal matrix multiplication

Largest Increasing Sub-sequence (LIS)

LIS

Given a sequence X of n integers, the objective is to find the longest sub-sequence, $X_{k_1}, X_{k_2}, \ldots, X_{k_m}$, such that $k_i > k_{i-1}$ and $X_{k_i} > X_{k_{i-1}}$. For example, for the following sequence:

$$3, 8, 2, 7, 3, 9, 12, 4, 1, 6, 10,$$

the longest increasing sub-sequence would be:

$$2, 3, 4, 6, 10.$$

The idea of the algorithm is to keep an array L where L_i represents the length of a *LIS* with X_i as its final element.

First start L_i with 1, then for every j from 0 to $i-1$, check if $X_j < X_i$ and $L_j + 1 > L_i$, if that happens then make $L_i = L_j + 1$. What we are doing here is to check if we can add the element X_i to the *LIS* that ends in X_j

LIS Example

For the sequence above, the array L will look like this.

X =	3	8	2	7	3	9	12	4	1	6	10
L =	1	2	1	2	2	3	4	3	1	4	5

The value of L_i represents the length of the *LIS* ending with X_i.

Q & A

1. How the LIS algorithm looks like?
2. What is the time complexity of the algorithm?
3. How can we print a valid LIS?
4. How can we improve the time of the algorithm?

Longest Common Sub-sequence (LCS)

LCS

A sub-sequence is a sequence that can be obtained from another sequence by removing some of its elements and preserving the order of the remaining elements.

The *LCS* of two sequences is a sub-sequence that is common to both sequences and has a maximal length.

LCS - Example

$$X = mexico$$
$$Y = america,$$

their *LCS* is *meic*.

LCS - Algorithm

The algorithm to find the *LCS* of two strings consists on having a matrix C of $n \times m$, where $C_{i,j}$ represents the length of the *LCS* using the first i letters from X and the first j letters from Y. So the result will be stored in position $C_{n,m}$

LCS - Algorithm

For the case where X_i is equal to Y_j that means that adding letter X_i and Y_j increments the length of the *LCS* by one, $C_{i,j} = C_{i-1,j-1} + 1$.

And for the case where X_i and Y_j are different, we only need to keep the greatest value in C so far, $C_{i,j} = \max(C_{i-1,j}, C_{i,j-1})$.

LCS - Algorithm Example

		m	a	c	e	t	a
	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0
t	0	0	0	0	0	1	1
e	0	0	0	0	1	1	1
l	0	0	0	0	1	1	1
l	0	0	0	0	1	1	1
a	0	0	1	1	1	1	2

The value of C_{ij} represents the length of the *LCS* considering the first *i* characters of the