Computational Algebra. Presentation/Lecture 1

M.Eulàlia Montoro

15 de febrero de 2023

Welcome

My e-mail is:

eula.montoro @ub.edu

Our lectures are one session a week of two hours.
The first hour (8.00-9.00h) is the **computer part** and the second hour (9.00-10.00h) is the **problems part**.

Computer Part (8.00-9.00h)

- In this hour we will have to do an assignement with Mathematica (other programs are not allowed).
  Then, you will have to do some homework related to it.
  This homework must be upload to the Campus Virtual in the corresponding task with a deadline of one week.
- After the spring break, everyone will have to prepare one of the lab's session.
  The material necessary for the presentations will be provided by me well in advance.

Problems Part (9.00-10.00 h)

- In the second hour you will present some problems.
- You have to present one problem of every list in class, and to upload to the campus virtual at least another two.
- Please, don't forget to tell me in advance (via e-mail) which problem do you want to present in class.
- The problems to be presented will be assigned by order of demand.
- As soon as possible, the exercises to be explained in class will be announced in advance in the virtual campus.

Summary

You must to

- Do the homework of the computer part every week.
- Present one lab of the computer part.
- Present on the board one exercise of each list.
- Deliver at least another two exercises of each list.

With this part you can obtain 40 % of the note.

Today

The part of the exercises begins next week.
Today we only work on the computer part.
If you have never used Mathematica before, press this link:

Introduction to Mathematica

Now.... Open the file of lab 1 in the Campus Virtual and follow the instructions.

Fer Function

- What does Fer compute?

Fer Function

- What does Fer compute?

$$Fer[n] = 2^{2^n}$$

- Algebraic Complexity?

Fer Function

- What does Fer compute?

$$Fer[n] = 2^{2^n}$$

- Algebraic Complexity?

$$O(2^n)$$

Fer Function

- What does Fer compute?

$$Fer[n] = 2^{2^n}$$

- Algebraic Complexity?

$$O(2^n)$$

  We calculate two times $Fer[n-1]$ in each step of the recursion.

- Better way?

Fer Function

- What does Fer compute?

$$Fer[n] = 2^{2^n}$$

- Algebraic Complexity?

$$O(2^n)$$

  We calculate two times $Fer[n-1]$ in each step of the recursion.

- Better way?

$$Fer[n] := (Fer[n-1])^2$$

  or

Fer Function

- What does Fer compute?

$$Fer[n] = 2^{2^n}$$

- Algebraic Complexity?

$$O(2^n)$$

  We calculate two times $Fer[n-1]$ in each step of the recursion.

- Better way?

$$Fer[n] := (Fer[n-1])^2$$

  or

$$Fer[n] := Fer[n] = Fer[n-1] * Fer[n-1]$$

Fer Function

- New complexity?

Fer Function

- New complexity?

$$O(n)$$

- Test: Use Table and Timing instructions.

Fibonacci

- c)To prove that $c(n) = Fib[n+1] - 1$ (where $c(n)$ is the number of aditions needed to compute $Fib[n]$ we use induction:

$$c(0) = 0,\ c(1) = 0 = Fib[2] - 1$$

$$c(2) = 1 = Fib[3] - 1$$

Now, suppose that $c(n-1) = Fib[n] - 1$, then

$$c(n) = c(n-1) + c(n-2) + 1 = Fib[n] - 1 + Fib[n-1] - 1 + 1 =$$

$$= Fib[n+1] - 1$$

Fibonacci

- d)Design a recursive algorithm of complexity $O(n)$ in order to compute $Fib[n]$.

```
MFib[0,x_,y_]:=x;
MFib[1,x_,y_]:=y;
MFib[n_,x_,y_]:=MFib[n-1,y,x+y]
```

Calc versus Horner

Given n a positive integer, $T = \{T_0, \ldots, T_n\}$ and x any number.
How we can compute $T(x) = T_0 + T_1 x + \ldots + T_n x^n$?.
We study two different ways:

- Calc:

$$S_0 = T_0$$

$$S_1 = S_0 + T_1 x = T_0 + T_1 x$$

$$x^2 = x \cdot x,\ S_2 = S_1 + T_2 x^2$$

$$x^3 = x^2 \cdot x,\ S_3 = S_2 + T_3 x^3$$

and so on...

Calc versus Horner

Given n a positive integer, $T = \{T_0, \ldots, T_n\}$ and x any number.
How we can compute $T(x) = T_0 + T_1 x + \ldots + T_n x^n$?.
We study two different ways:

- Calc:

$$S_0 = T_0$$

$$S_1 = S_0 + T_1 x = T_0 + T_1 x$$

$$x^2 = x \cdot x,\ S_2 = S_1 + T_2 x^2$$

$$x^3 = x^2 \cdot x,\ S_3 = S_2 + T_3 x^3$$

and so on...
$n-1$ multiplications, n multiplications, n additions.

Calc versus Horner

Given n a positive integer, $T = \{T_0, \ldots, T_n\}$ and x any number.

- Horn:

$$S_0 = T_n$$

$$S_1 = T_{n-1} + S_0 x = T_{n-1} + xT_n$$

$$S_2 = T_{n-2} + xS_1 = T_{n-2} + x(T_{n-1} + xT_n)$$

$$S_3 = T_{n-3} + xS_2 =$$

$$T_{n-3} + x(T_{n-2} + x(T_{n-1} + xT_n))$$

and so on...

Calc versus Horner

Given n a positive integer, $T = \{T_0, \ldots, T_n\}$ and x any number.

- Horn:

$$S_0 = T_n$$

$$S_1 = T_{n-1} + S_0 x = T_{n-1} + xT_n$$

$$S_2 = T_{n-2} + xS_1 = T_{n-2} + x(T_{n-1} + xT_n)$$

$$S_3 = T_{n-3} + xS_2 =$$

$$T_{n-3} + x(T_{n-2} + x(T_{n-1} + xT_n))$$

and so on...

n multiplications, n additions.

Calc versus Horner

- Implement a procedure which generates a random polynomial with integer coefficients chosen from a given interval.

```
Table[Random[Integer,{-500,500}],{i,0,n}];
```

Homework.

Given a positive integer n, we consider the binary expansion of $n = \sum_{i=0}^{m} a(i)2^i$ with $a(i)$ either 0 or 1. Note that this is the way this number is defined on a machine. The goal of this exercise is to compute x^n, with x a given real number.

a) Show that if one knows x^{2^i}, then we can compute x^n.

b) Using a) and also the relation between the successive divisions of n by 2 and the $a(i)'s$, design a recursive algorithm for computing x^n.

c) Design an iterative version for computing x^n.

d) Compare the algebraic complexity of these algorithms with those "naive" versions computing $x^n = xx^{n-1}$ (recursive case) and $x^n = xx \cdot \cdot x$ (iterative case).

d) Test with examples your algorithms and compare them with implementations of the naive versions of both the iterative and recursive cases.

Solutions.

b) Using a) and also the relation between the successive divisions of n by 2 and the $a(i)'s$, design a recursive algorithm for computing x^n.
RecBin[x_,0]:=1;
RecBin[x_,1]:=x;
RecBin[x_,n_]:=If[EvenQ[n],RecBin[xx,$\frac{n}{2}$],x ·RecBin[xx,$\frac{n-1}{2}$]]

c) Design an iterative version for computing x^n.

Solutions.

d) Compare the algebraic complexity of these algorithms with those "naive" versions computing $x^n = xx^{n-1}$ (recursive case) and $x^n = xx \cdot \cdot x$ (iterative case).

- Binary exponentiation: We only need to do $2(\log_2(n) + 1)$ operations. Therefore, its complexity is $O(\log_2(n))$
- Naive exponentiation: We do n products in both cases. Therefore its complexity is $O(n)$.