CS 101: Introduction to Computer Science

Mahindra École Centrale
Duration: 2 hours | *CS101 First Lab Examination* | Total 20 marks
Solution for the Examination

February 23rd to 27th, 2015

For the first two problems, three things were needed (in this order):

1. Find a logical or mathematical solution, that you will write and explain in English and using mathematical notations. You will write this on an external piece of paper, with your name, roll number, date and title of the problem.
2. Write a *valid* Python program that solves the problem, and produces correct output, for the two values (the example one, which has to be the same as the one given in the problem, and the target)

   These Python programs will be saved on the Desktop of the computer used for the exam, under a folder named with your roll number (e.g. `14XJ00579`), and used to evaluate you (for no more than 50% of the grade for that exercise),
3. Additionnaly, write the target output on you answer paper **and** on this question paper that you have to give back at the end.

These 3 components has been evaluated to produce a grade for each problem.
Note: there were two kinds of problem I and two kinds also of problem II. There were exactly of the same difficulty.

Please find below a mathematical explanation, then a simple Python program, then the desired output; for each of these 4 problems. The Python programs have also been uploaded to Moodle, merger into one single file.

Problem I (1/2) : sum of multiple of 7 and 11 (***Marks:*** 5)

> Find the sum of all the positive integer numbers ($n > 0$) that are divisible by 7 **or** 11 and that are below 10 thousand (ie. $n < 10000$).
> For example, for the numbers below 50, we have 7, 11, 14, 21, 22, 28, 33, 35, 42, 44 and 49 so the sum is $7 + 11 + 14 + 21 + 22 + 28 + 33 + 35 + 42 + 44 + 49 = 306$.

Mathematical solution

That problem was quite simple to solve, but too long to compute manually if you do it naively.

That sum is simply

$$S = \sum_{1 \leqslant n < 10000,\ 7 \mid n \text{ or } 11 \mid n} n.$$

It can therefore be written as:

$$S = \left(\sum_{1 \leqslant n < 10000,\ 7 \mid n} n \right) + \left(\sum_{1 \leqslant n < 10000,\ 11 \mid n} n \right) - \left(\sum_{1 \leqslant n < 10000,\ 7 \mid n \text{ and } 11 \mid n} n \right)$$

Because we count twice the comon multiples of 7 and 11.

$$S = \left(\sum_{1 \leqslant k \leqslant \lfloor 10000/7 \rfloor} 7k \right) + \left(\sum_{1 \leqslant k \leqslant \lfloor 10000/11 \rfloor} 11k \right) - \left(\sum_{1 \leqslant n < 10000,\ 7 \times 11 \mid n} n \right)$$

The three $<$ became $\leqslant$ because all the three integer division are rounded ($7 \nmid 10000, 11 \nmid 10000$). And because 7 and 11 are primes with each other, $7 \mid n$ and $11 \mid n \Leftrightarrow 11 \mid n$.

$$S = 7 \times \left(\sum_{1 \leqslant k \leqslant 1428} k \right) + 11 \times \left(\sum_{1 \leqslant k \leqslant 909} k \right) - \left(\sum_{1 \leqslant k \leqslant \lfloor 10000/77 \rfloor} 77k \right)$$

We use the well-known formula, three times:

$$S = 7 \times \left(\frac{1428 \times (1428 + 1)}{2} \right) + 11 \times \left(\frac{909 \times (909 + 1)}{2} \right) - 77 \times \left(\frac{129 \times (129 + 1)}{2} \right)$$

And now the computation can be done easily:

$$S = 7 \times 1020306 + 11 \times 413595 - 77 \times 8385$$
$$S = 11046042$$

Python program

```
1 # First computation, for numbers below 50
2 firstSum = 0
3
4 for n in xrange(1, 50):  # 1 <= n < 50
5     if (n % 7 == 0) or (n % 11 == 0):
6         firstSum += n
7
8 print "Problem 1 (1/2) ==> The sum of all the positive integer numbers↩
      that are multiple of 7 or 11 and smaller than", 50, "is", firstSum
```

```
9  # We expect 306, as the example given in the question.
10
11
12 # Second computation, for numbers below 10000
13 requiredSum = 0
14
15 for n in xrange(1, 10000):  # 1 <= n < 10000
16     if (n % 7 == 0) or (n % 11 == 0):
17         requiredSum += n
18
19 print "Problem 1 (1/2) ==> The sum of all the positive integer numbers↩
       that are multiple of 7 or 11 and smaller than", 10000, "is", ↩
     requiredSum
20 # We expect 11046042, as computed mathematically.
```

Output for problem I:

11046042

Problem I (2/2) : product of multiple of 3 and 5 (*Marks:* 5)

> Find the product of all the positive integer numbers ($n > 0$) that are divisible by 3 **or** 5 and that are below 30 (ie. $n < 30$).
> For example, for the numbers below 15, we have 3, 5, 6, 9, 10 and 12 so the product is $3 \times 5 \times 6 \times 9 \times 10 \times 12 = 97200$.

Mathematical solution

That problem was quite simple to solve, but too long to compute manually if you do it naively.

That product is simply

$$P = \prod_{1 \leqslant n < 30,\ 3 \mid n \text{ or } 5 \mid n} n.$$

It can therefore be written as:

$$P = \frac{\left(\prod_{1 \leqslant n < 30,\ 3 \mid n} n\right) \times \left(\prod_{1 \leqslant n < 30,\ 5 \mid n} n\right)}{\left(\prod_{1 \leqslant n < 30,\ 3 \mid n \text{ and } 5 \mid n} n\right)}$$

Because we count twice the comon multiples of 3 and 5.

$$P = \frac{\left(\prod_{1 \leqslant k < \lfloor 30/3 \rfloor} 3k\right) \times \left(\prod_{1 \leqslant k < \lfloor 30/5 \rfloor} 5k\right)}{\left(\prod_{1 \leqslant n < 30,\ 3 \times 5 \mid n} n\right)}$$

The $<$ still are $<$ because the integer divisions are not rounded here ($3 \mid 30$, and $5 \mid 30$, and $15 \mid 30$).
And because 3 and 5 are primes with each other, $3 \mid n$ and $5 \mid n \Leftrightarrow 15 \mid n$.

$$P = \frac{\left(3^9 \times \prod_{1 \leqslant k < 10} k\right) \times \left(5^5 \times \prod_{1 \leqslant k < 6} k\right)}{\left(\prod_{1 \leqslant k < \lfloor 30/15 \rfloor} 15k\right)}$$

We see now some factorial numbers:

$$P = \frac{\left(3^9 \times 9!\right) \times \left(5^5 \times 5!\right)}{15^1 \times (1!)}$$

And now the computation can be done "easily" (even if it is too big to be done by hand):

$$P = 3^8 \times 9! \times 5^4 \times 5!$$
$$P = 6561 \times 362880 \times 625 \times 120$$
$$P = 178564176000000$$

Python program

```
1  # First computation, for numbers below 15
2  firstProduct = 1
3
4  for n in xrange(1, 15):  # 1 <= n < 15
5      if (n % 3 == 0) or (n % 5 == 0):
6          firstProduct *= n
7
8  print "Problem 1 (2/2) ==> The product of all the positive integer numbers that are multiple of 3 or 5 and smaller than", 15, "is", firstProduct
9  # We expect 97200, as the example given in the question.
10
11
12 # Second computation, for numbers below 30
13 requiredProduct = 1
14
15 for n in xrange(1, 30):  # 1 <= n < 30
16     if (n % 3 == 0) or (n % 5 == 0):
17         requiredProduct *= n
18
19 print "Problem 1 (2/2) ==> The product of all the positive integer numbers that are multiple of 3 or 5 and smaller than", 30, "is", requiredProduct
20 # We expect 178564176000000, as computed mathematically.
```

Output for problem I:

178564176000000

Problem II (1/2) : sum of the first Fibonacci numbers (*Marks:* 5)

> Find the sum of the first 30 Fibonacci numbers $F_0 + F_1 + \cdots + F_{28} + F_{29}$.
> Hint: We remind that the sequence $(F_n)_{n\in\mathbb{N}}$ is given by $F_0 = 0$, $F_1 = 1$, and $F_{n+2} = F_{n+1} + F_n$. For example, the first 5 Fibonacci numbers are 0, 1, 1, 2, 3 so their sum is $0 + 1 + 1 + 2 + 3 = 7$.

Mathematical solution (First idea)

That problem was harder than the first one, but not that hard anyway. Let us call S_n the sum of the first $n + 1$ Fibonacci numbers.

We have $S_{n+1} = S_n + F_{n+1} = S_n + F_n + F_{n-1}$ if $n > 0$. But $S_n - S_{n-2} = F_n + F_{n-1}$. So $S_{n+1} = 2S_n - S_{n-2}$ is a recurrent sequence of order 3. We can then define S_n with $S_0 = 0$, $S_1 = 1$, $S_2 = 2$ and $S_{n+3} = 2S_{n+2} - S_n$.

The required value S_{29} can then be computed in n steps of computation (n being 30 here).

Mathematical solution (simpler idea)

An even simple solution, as used in the Python program, is to keep a list of values of the Fibonacci numbers, and to create the new value F_{n+2} as $F_n + F_{n+1}$.

Note: it was also possible to just keep two variables, like `f1` and `f2`, with `f1` representing F_n and `f2` F_{n+1}. They get updated by `f1', f2' = f2, f1 + f2`, as $F_{n'} = F_{n+1}$ and $F_{n'+1} = F_n + F_{n+1}$ if $n' = n + 1$ (here, `f1', f2'` mean the new values of `f1, f2`).

Python program

```
1  # We use functions, just for the sake of the example
2  def sumFirstFibonacciNumbers(k = 5):
3      currentSum = 0
4      # List to store F_n as fibs[n], the nth element of the list
5      fibs = [0, 1]
6      n = 0
7      while n < k :
8          # Computing F_{n+2} is just appending a new value to our list
9          fibs.append(fibs[n] + fibs[n+1])
10         currentSum += fibs[n]
11         print "We added F_n =", fibs[n], "( for n =", n, ") to the ↩
               current sum, which is now", currentSum
12         n += 1
13
14     print "Problem 2 (1st kind) ==> the sum of the first", k, "↩
           Fibonacci numbers is", currentSum
15
16
17 sumFirstFibonacciNumbers(5)   # We expect 7, as given in the example
18 sumFirstFibonacciNumbers(30)  # We get 1346268
```

Output of the Python program

```
We added F_n = 0 ( for n = 0 ) to the current sum, which is now 0
We added F_n = 1 ( for n = 1 ) to the current sum, which is now 1
```

```
We added F_n = 1 ( for n = 2 ) to the current sum, which is now 2
We added F_n = 2 ( for n = 3 ) to the current sum, which is now 4
We added F_n = 3 ( for n = 4 ) to the current sum, which is now 7
We added F_n = 5 ( for n = 5 ) to the current sum, which is now 12
We added F_n = 8 ( for n = 6 ) to the current sum, which is now 20
We added F_n = 13 ( for n = 7 ) to the current sum, which is now 33
We added F_n = 21 ( for n = 8 ) to the current sum, which is now 54
We added F_n = 34 ( for n = 9 ) to the current sum, which is now 88
We added F_n = 55 ( for n = 10 ) to the current sum, which is now 143
We added F_n = 89 ( for n = 11 ) to the current sum, which is now 232
We added F_n = 144 ( for n = 12 ) to the current sum, which is now 376
We added F_n = 233 ( for n = 13 ) to the current sum, which is now 609
We added F_n = 377 ( for n = 14 ) to the current sum, which is now 986
We added F_n = 610 ( for n = 15 ) to the current sum, which is now 1596
We added F_n = 987 ( for n = 16 ) to the current sum, which is now 2583
We added F_n = 1597 ( for n = 17 ) to the current sum, which is now 4180
We added F_n = 2584 ( for n = 18 ) to the current sum, which is now 6764
We added F_n = 4181 ( for n = 19 ) to the current sum, which is now 10945
We added F_n = 6765 ( for n = 20 ) to the current sum, which is now 17710
We added F_n = 10946 ( for n = 21 ) to the current sum, which is now 28656
We added F_n = 17711 ( for n = 22 ) to the current sum, which is now 46367
We added F_n = 28657 ( for n = 23 ) to the current sum, which is now 75024
We added F_n = 46368 ( for n = 24 ) to the current sum, which is now 121392
We added F_n = 75025 ( for n = 25 ) to the current sum, which is now 196417
We added F_n = 121393 ( for n = 26 ) to the current sum, which is now 317810
We added F_n = 196418 ( for n = 27 ) to the current sum, which is now 514228
We added F_n = 317811 ( for n = 28 ) to the current sum, which is now 832039
We added F_n = 514229 ( for n = 29 ) to the current sum, which is now 1346268
Problem 2 (1st kind) ==> the sum of the first 30 Fibonacci numbers is 1346268
```

Output for problem II:

1346268

Problem II (2/2) : sum of the first factorial numbers (*Marks:* 5)

> Give the first 14 values of the factorial numbers $(0!, 1!, \ldots, 13!)$, and give their sum.
> You also need to find from which value of n its factorial $n!$ is strictly greater than 10^{15}.
>
> Hint: We remind that the factorial function is defined with $0! = 1, n! = 1 \times 2 \times \cdots \times n = \prod_{i=1}^{n} i$
>
> (and satisfies $(n+1)! = (n+1) \times n!$ recursively).

Mathematical solution (simpler idea)

That problem was harder than the first one, but not that hard anyway.

The computation of the factorial of n, $n!$, is so obvious that there is basically nothing I can explain. Just see the Python code below.

Python program

```
1  # We again use functions, just for the sake of the example
2  def printFirstFactorial(k):
3      n = 0
4      theirSum = 0
5      # List to store n! as facts[n], the nth element of the list
6      facts = [1]  # 0! = 1
7      while n < k:
8          facts.append(facts[n] * (n+1))
9          print "For n =", n, "its factorial n! is", facts[n]
10         theirSum += facts[n]
11         n += 1
12
13     print "Done, I printed the first", k, "values of factorial n ( for↩
           n from 0 to", k-1, ")."
14     print "And their sum is", theirSum
15
16
17 print "Problem 2 (2nd kind) ==> "
18 printFirstFactorial(14)
19
20
21 def smallestFactorialBiggerThan(x):
22     n = 0
23     # List to store n! as facts[n], the nth element of the list
24     facts = [1]  # 0! = 1
25     while facts[n] <= x:  # As long as n! is smaller than x:
26         facts.append(facts[n] * (n+1))
27         n += 1
28
29     print "For n =", n, "its factorial is", facts[n]
30     return n
31
32
33 n = smallestFactorialBiggerThan(1e15)
```

```
34 print "Problem 2 (2nd kind) ==> the smallest n such that n! > 10**15 ↩
       is", n
```

Output of the Python program

```
Problem 2 (2nd kind) ==>
For n = 0 its factorial n! is 1
For n = 1 its factorial n! is 1
For n = 2 its factorial n! is 2
For n = 3 its factorial n! is 6
For n = 4 its factorial n! is 24
For n = 5 its factorial n! is 120
For n = 6 its factorial n! is 720
For n = 7 its factorial n! is 5040
For n = 8 its factorial n! is 40320
For n = 9 its factorial n! is 362880
For n = 10 its factorial n! is 3628800
For n = 11 its factorial n! is 39916800
For n = 12 its factorial n! is 479001600
For n = 13 its factorial n! is 6227020800
Done, I printed the first 14 values of factorial n ( for n from 0 to 13 ).
For n = 18 its factorial is 6402373705728000
Problem 2 (2nd kind) ==> the smallest n such that n! > 10**15 is 18
```

Output for problem II:

The successive factorials are shown just above.
The smallest n such that $n! > 10^{15}$ is 18.

Obvious solution (simpler but more clever idea)

One student at least though about that: maybe the factorial function was already available in Python?!

In fact yes, it is, in the `math` package/module (module that has been used and showed in the lab #3). As the Python documation shows (), the factorial function can be obtained with `math.factorial` (which return a `float` number and not an integer).

Hence, an obvious and simpler solution can be:

```
1  # We now use math.factorial:
2  import math
3  print "Problem 2 (2nd kind):"
4
5  # First question
6  theirSum = 0
7  for n in xrange(0, 14):
8          print "For n =", n, "its factorial n! is", math.factorial(n)
9          theirSum += math.factorial(n)
10
11 print "Done, I printed the first 14 values of factorial n."
12 print "And their sum is", theirSum
13
14 # Second question
15 n = 0
```

```
16 while math.factorial(n) <= 10**15:  # As long as n! is smaller than x:
17     n += 1
18
19 print "The smallest n such that n! > 10**15 is", n
```

Problem III : the railway problem (longer) (*Marks:* 10)

There are 2 classes of trains – A (express) and B (goods) – that can travel on a section of railway track $PQRS$ (see Fig.) 30 kms long. This track section is equally split into 3 parts.

- The first PQ part has a positive (i.e. adverse) gradient that decelerates a goods train by 20 kms/hours and the express train by 80% of that rate.
- The second stretch QR is a level plain.
- The third stretch RS has a negative (i.e. downward) gradient that accelerates a goods train by 16 kms/hours and an express train by 75% of that amount.

Now assume that you are writing a program that throws commands to the train driver on a screen. The commands are to "Apply Power" immediately if the speed falls below 20 kms/hours for the train type B and 25 kms/hours for train type A, and alternately to "Apply Brake" if the speed goes beyond 50 kms/hours for the train type B and 60 kms/hours for type A.

Importantly, the "power" or "brake" applied by the driver is just sufficient to nullify the corresponding deceleration or acceleration of that stretch, respectively, and is taken off as soon as the driver enters the level section. Further, the driver is asked to enter the "Entry Speed" at the instant he enters this track zone of interest, i.e. crosses point P.

Now write the program using your knowledge of basic arithmetic and what you have so far learnt in programming, which provides the driver with the above commands at entry and other action points ("action" is for driver). At the action points, also print in float the "number of kms entered into that section". Your first inputs that you will request from driver will be the train type and train speed.

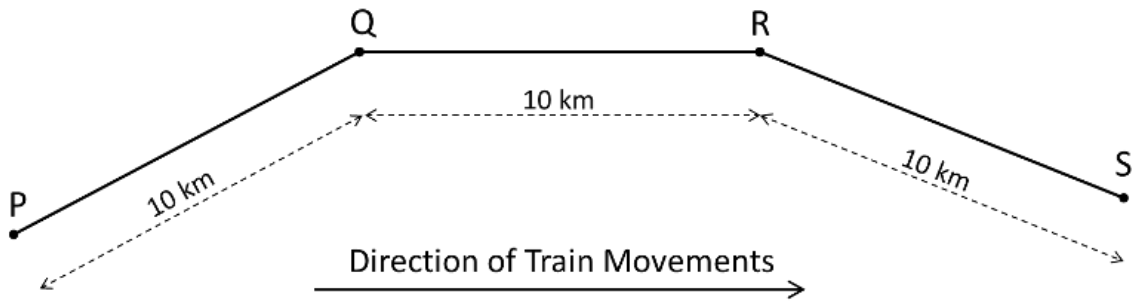

Figure 1: Section of ralway track $PQRS$.

Important:

- You are **not** supposed to start coding right away.
- Instead, first write out systematically the logical steps of the computation, from start to finish, on your answer script.
- Only then convert this into code. Your written computational steps contain as many marks as the code itself.

Hint: Use these three equations of kinematics:

(a) $v = u + at$, (b) $s = ut + \frac{1}{2}at^2$, (c) $v^2 = u^2 + 2as$

where u and v are the initial and final velocities covered in time t along distance s, and a is the constant rate of acceleration.

TODO: write the solution for that last problem. Prof. Arya could do it?

Python program

The Python code for that problem is quite long, so please consult it from Moodle instead.