Tarea 1

Profesor: Pablo Barceló
Auxiliares: Javier Oliva, Bernardo Subercaseaux
Ayudantes: Joaquín Cruz, Heinich Porro, Lucas Torrealba, Florencia Yáñez

Instrucciones.-

- En material docente se publicará un archivo con políticas de colaboración, reglamentando como está autorizado que colabore con sus compañer@s.
- Deberá entregar un único archivo en formato `.zip`, que deberá contener tanto su código como un archivo en formato `.pdf` con sus demostraciones.
- Procure que sus demostraciones sean claras, sin ambigüedades y ordenadas. Se recomienda, aunque no es obligatorio, que elabore sus informes usando LaTeX
- Puede programar en cualquiera de los siguientes lenguajes: Python, Java, C++
- Su código debe recibir input y escribir output de la manera descrita, no introduzca nada adicional. Debe imprimir un salto de línea (`'\n'`) al final de cada línea impresa.

P1.-

En lógica proposicional, un literal es una variable o la negación de una variable. Es decir, p y $\neg q$ son literales, pero no $q \vee \neg p$. Una cláusula es la disyunción de literales. Es decir, $(p \vee q \vee \neg r)$ y $(\neg p \vee p \vee \neg q)$ son cláusulas. Mientras que $(p \vee q \wedge r)$ no lo es. Decimos que una fórmula de la lógica proposicional está en *CNF* si se escribe como conjunción de literales, es decir como $C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_n$ donde cada C_i es una cláusula.

1. Pruebe que toda fórmula de la lógica proposicional se puede escribir en *CNF*.
2. Pruebe que para toda fórmula en *CNF* de n cláusulas existe una valuación donde al menos $n/2$ cláusulas se satisfacen.
3. Pruebe que el resultado anterior no se puede mejorar. Es decir, no existe ningún $r < \frac{1}{2}$ tal que para toda fórmula en *CNF* de n cláusulas exista una valuación donde al menos rn cláusulas se satisfacen.

P2.-

De la misma manera en que el número 19 se escribe en binario como

$$10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

decimos que se escribe de forma factorial como $3010_! = 3 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! + 0 \cdot 0!$. Más formalmente el valor de un número escrito de forma factorial $(a_n a_{n-1} a_{n-2} ... a_0)_!$ es igual a $\sum_{i=0}^{n} a_i \cdot i!$, donde $0 \leq a_i \leq i$.

1. Pruebe que todo natural se puede escribir de forma factorial, y que además, esa expresión es única (no hay dos formas distintas de escribir un número en forma factorial).

2. Programe una función `baseFact` que recibe un natural n y retorna un vector (cada posición corresponde a un dígito) m que representa al número en forma factorial , es decir $m_! = n_{10}$. Por ejemplo `baseFact(463) = [3,4,1,0,1,0]`, ya que $341010_! = 463_{10}$

3. Entregue un archivo `baseFact.{py,java,cpp}` que al ser ejecutado reciba por entrada estándar un número natural n, e imprima el resultado de la función anterior aplicada sobre n, separando los distintos dígitos con un signo '#'. Por ejemplo, si su programa recibe 781238417823, deberá retornar:

   $$8\#13\#5\#11\#7\#3\#5\#6\#4\#0\#2\#2\#1\#1\#0$$

   Note que el separador cobra sentido ya que en forma factorial los "dígitos"pueden tener más de un dígito decimal. En el ejemplo, tanto 13 como 11 son un solo dígito.

Hint: Al intentar convertir un número n a forma factorial, considere que ocurre si busca el factorial más grande que es menor que n. Por ejemplo, el factorial más grande menor que 463 *es* $5! = 120$*, ya que el siguiente es* $6! = 720$*. Note que puede multiplicar* $3 \cdot 5! = 360$ *y sigue siendo menor que* 463*, pero* $4 \cdot 5! = 480$ *ya no lo es.*

P3.- Recordemos que los números de Fibonnaci están dados por la secuencia $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$.

¿Por qué solo considerar números factoriales cuando también podemos considerar números de Fibonacci? Un número está escrito en forma de Fibonacci si se escribe como suma de números de Fibonacci diferentes. Por ejemplo $64 = 55+8+1$. Para representar estas sumas utilizaremos arreglos binarios, donde un 1 en la posición i (de derecha a izquierda y contando desde 1) representa que la suma contiene al $(i+1)$-ésimo número de Fibonacci. Por ejemplo, en este caso la representación de 64 sería $[1, 0, 0, 0, 1, 0, 0, 0, 1]$ ya que las posiciones $1, 5$ y 9 tienen unos, y por lo tanto estamos sumando f_2, f_6 y f_9 que son justamente $1, 8$ y 55. Note que esta representación no es única, ya que por ejemplo $64 = 34 + 21 + 8 + 1$.

1. Pruebe que todo natural se puede escribir en forma de Fibonacci sin que en su representación aparezcan dos unos consecutivos. Pruebe además que bajo esta condición la representación es única.

2. Programe una función `baseFib` que recibe un natural n y retorna un vector m que representa al número en forma de Fibonacci sin unos consecutivos. Por ejemplo: `baseFib(65) = [1,0,0,0,1,0,0,0,1]`.

3. Entregue un archivo `baseFact.{py,java,cpp}` que al ser ejecutado reciba por entrada estándar un número natural n, e imprima el resultado de la función anterior aplicada sobre n, separando las distintas posiciones con un signo '#'.