Recursión I

Factorial

Dado n de 1 o más, retorne el factorial de n, el cual es n\*(n-1) \*(n-2) …1. Calcular el resultado de forma recursiva (sin ciclos).

BunnyEars

Tenemos un numero de conejitos y cada conejito tiene dos grandes orejas, queremos calcular el número de orejas en todos los conejitos de forma recursiva (sin ciclos o multiplicación).

Fibonacci\*

La secuencia Fibonacci es una famosa parte de las matemáticas, y esto pasa por tener una definición recursiva. Los primeros dos valores en la secuencia son 0 y 1 (esencialmente 2 casos base). Cada sub secuencia de valores es la suma de los 2 valores previos, entonces la secuencia completa es: 0, 1, 1, 2, 3, 5, 8, 13, 21 y así. Defina un método recursivo fibonacci(n) que retorne el n numero Fibonacci, con n=0 representando el inicio de la secuencia.

bunnyEars2

tenemos conejitos posicionados en una línea, enumerados 1,2, … Los impares conejos (1, 3, …) tienen 2 orejas normales. Los (2, 4, …) bien dicho tienen 3 orejas, porque cada uno tienen un pie levantado, recursivamente retorne en número de “orejas” en la línea de conejos de 1, 2, … n (sin ciclos, ni multiplicación).

triangle

tenemos un triángulo hecho de bloques, la fila superior tiene 1 bloque, la siguiente fila abajo tiene 2 bloques, la siguiente fila tiene 3 bloques y así sucesivamente. Calcular recursivamente (no ciclos o multiplicación) el número total de bloques en un triángulo con el número de filas dadas.

sumDigits

dado un entero n no negativo, retornar la suma de los dígitos recursivamente (no ciclos). Note que el modulo (%) por 10 se obtiene el número más a la derecha (126 % 10 es 6), mientras que dividiendo (/) por 10 removemos el número más a la derecha (126 / 10 es 12).

count7

dado un entero no negativo n, retornar la cantidad de ocurrencias de 7 como un dígito, así que por ejemplo 717 redimimos 2. (sin bucles). Tenga en cuenta que mod (%) por 10 da el dígito más a la derecha (126% 10 es 6), mientras que dividir (/) por 10 elimina el dígito más a la derecha (126/10 es 12).

count8

Dado un entero no negativo n, calcular de forma recursiva (sin bucles) la cantidad de ocurrencias de 8 como un dígito, excepto un 8 con otro 8 inmediatamente a la izquierda cuenta por 2, entonces 8818 redimimos 4. Tenga en cuenta que mod (%) por 10 produce el dígito más a la derecha (126% 10 es 6), mientras que dividir (/) por 10 elimina el dígito más a la derecha (126/10 es 12).

powerN

dados Base y n donde ambos son 1 o más, calcular recursivamente (no ciclos) el valor de base a la n potencia, entonces (3, 2) es 9 (3 al cuadrado).

countX

dado una cadena, calcular recesivamente (ni ciclos) el número de caracteres ‘x’ minúsculas en la cadena.

countHi

dado una cadena, calcular recesivamente (ni ciclos) el número de veces en minúscula que aparece ‘hi’ en la cadena.

changeXY

dado una cadena, calcular recesivamente (ni ciclos) una nueva cadena donde todos los caracteres ‘x’ minúsculas han sido cambiados por el carácter ‘y’.

changePi \*

dado una cadena, calcular recesivamente (ni ciclos) una nueva cadena donde todas las apariciones de ‘pi’ han sido remplazadas por “3.14”.

noX

dado una cadena, calcular recesivamente una nueva cadena donde todos los caracteres ‘x’ han sido removidos.

array6

dado un arreglo de enteros, calcular recursivamente si el arreglo contiene un 6. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial pasara con índice 0.

Array11

Dado un arreglo de enteros, calcular recursivamente el número de veces que aparece el número 11 en el arreglo. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial pasara con índice 0.

Array220

Dado un arreglo de enteros, calcular recursivamente si el arreglo contiene en algún lado un valor que en arreglo esta seguido su multiplico por 10. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial pasara con índice 0.

allStar

dada una cadena, calcular recursivamente una nueva cadena donde todos los caracteres adyacentes ahora son separados por un ‘\*’.

pairStar

dada una cadena, calcular recursivamente una nueva cadena donde caracteres iguales que son adyacentes en la cadena original son separados uno del otro por un ‘\*’.

endX

dada una cadena, calcular recursivamente una nueva cadena donde todos los caracteres ‘x’ en minúsculas son movidas al final de la cadena.

countPairs

Diremos que un "par" en una cadena son dos instancias de un carácter separados por otro carácter. Así "AXA" las A hacen un par. Par de pueden superponerse, por lo que "AxAxA" contiene 3 pares - 2 para A y 1 para x. Recursivamente calcular el número de pares en la cadena dada.

countAbc

contar recursivamente el número de sub cadenas “abc” and “aba” que aparecen en la cadena dada.

count11

Dada una cadena, calcular de forma recursiva (sin ciclos) el número de subcadenas "11" en la cadena. Las subcadenas "11" no deben superponerse.

stringClean

Dada una cadena, retornar2 recurrentemente una cadena "limpia" en caracteres adyacentes que son los mismos así se ha reducido a un solo carácter. Así rendimientos "yyzzza" "yza".

countHi2

Dada una cadena, calcular de forma recursiva el número de veces que aparece "hi" en minúscula en la cadena, sin embargo, no cuentan "hi" que tiene una 'x' inmediatamente antes de él.

parenBit

Dada una cadena que contiene un solo par de paréntesis, calcular de forma recursiva una nueva cadena hecha de sólo del paréntesis y su contenido, por lo que "xyz(abc)123"rendimimos a "(abc)"

nestParen

Dada una cadena, retornar true si esta es un anidamiento de cero o más pares de paréntesis, al igual que "(())" o "((()))". Sugerencia: compruebe los primeros y últimos caracteres, después repetir con lo que está dentro de ellos.

strCount

Dada una cadena y una subcadena sub no vacía, calcular de forma recursiva el número de veces que sub aparece en la cadena, sin las subcadenas superpuestas.

strCopies

Dada una cadena y una subcadena sub no vacía, calcular de forma recursiva si al menos n copias de sub aparecen en la cadena en algún lugar, posiblemente con la superposición. N será no negativo.

strDist

Dada una cadena y una subcadena sub no vacía, calcular de forma recursiva la mayor subcadena que comienza y termina con sub y retornar su longitud.

Recursión II

groupSum

dado un arreglo de enteros, es posible escoger un grupo de algunos de los enteros, tal que la suma del grupo nos dé el objetivo (target)? Este es un clásico problema de recursión backtracking. Una vez que entiendas la recursividad estratégica backtracking en este problema. Puedes usar el mismo patrón para muchos problemas para buscar un espacio de decisiones. En lugar de buscar en el arreglo completo, nuestra conveniencia es considerar empezar en el arreglo a partir del índice (start) y continuando hasta el final del arreglo. El llamado puede especificar todo el arreglo simplemente pasando el inicio como cero -- los ciclos no son necesarios – las llamadas recursivas progresan hacia abajo en el arreglo.

groupSum6

dado un arreglo de enteros, es posible escoger un grupo de algunos de los enteros, comenzando en el índice (start), tal que el grupo sume el objetivo dado (target)? Sin embargo, con la siguiente restricción, que todos los 6 debe ser elegido (no necesita ciclos).

groupNoAdjt

dado un arreglo de enteros, es posible escoger un grupo de algunos de los enteros, tal que el grupo sume el objetivo dado (target) con la siguiente restricción: si un calor en el arreglo es escogido en el grupo, el valor que sigue inmediatamente a este en el arreglo no debe ser elegido. (no necesita ciclos).

groupSum5

dado un arreglo de enteros, es posible escoger un grupo de algunos de los enteros, tal que el grupo sume el objetivo dado (target) con la siguiente restricción: todos los múltiplos de 5 en el arreglo deben ser incluidos en el grupo. Si el valor que sigue inmediatamente a el múltiplo de 5 es 1, este no debe ser elegido. (no necesita ciclos).

groupSumClump

dado un arreglo de enteros, es posible escoger un grupo de alguno los enteros. Tal que el grupo sume el objetivo dado (target), con la siguiente restricción: si hay números en el arreglo que están adyacentes y de valor idéntico,

splitArray

dado un arreglo de enteros, es posible dividir los enteros en dos grupos, que la suma de los dos grupos sean la misma. cada entero debe estar en un grupo o en otro. Escribir un método ayudante recursivo que tome cualquier argumento que gustes, y hacer el llamado inicial a tu ayudante recursiva desde SplitArray(). (no necesita ciclos).

splitOdd10

dado un arreglo de enteros, es posible dividir los enteros en dos grupos, de modo que la suma de un grupo es un múltiplo de 10, y la suma del otro grupo es impar, cada entero debe estar en un grupo o en otro. Escribir un método ayudante recursivo que tome cualquier argumento que gustes, y hacer el llamado inicial a tu ayudante recursiva desde splitOdd10(). (no necesita ciclos).

split53

dado un arreglo de enteros, es posible dividir los enteros en dos grupos, de modo que la suma de los dos grupos son las mismas, con la siguiente restricción: todos los valores múltiplos de 5 deben estar en un grupo y todos los valores que son múltiplos de 3 (y no múltiplo de 5) deben estar en el otro. (no necesita ciclos).