Guía de ejercicios 1 - Recursión

Parte I –Recursión con números y funciones

1. Definir una función recursiva sumatoria que, dado un entero n, retorne el resultado de la suma 1+2+3+...+(n-1)+n.

2. Definir una función recursiva factorial que, dado un número entero positivo, retorne el factorial de ese número.

3. Definir una función recursiva fibonacci que, dado un entero n, retorne el n-ésimo término de la sucesión de Fibonacci (1, 1, 2, 3, 5, 8, 13, ...).

4. Definir una función recursiva producto que, dados dos números enteros z y v, retorne z*v mediante sumas sucesivas.

5. Definir una función recursiva potencia que, dados dos números enteros x e y, retorne x^y.

6. Definir funciones recursivas cociente y resto que, a partir de dos números enteros, retorne el cociente y el resto entre ellos respectivamente, a partir de la técnica de restas sucesivas. Expresar cuál sería el orden bien fundado entre los elementos del dominio para esta función.

7. Definir los siguiente:

   a) Una función recursiva decimalBinario que, dado un número decimal, retorne el mismo en base binaria.

   b) Una función recursiva cambioBaseDecimal que, dado un número decimal *n* y un entero *b*, retorne la conversión del número *n* en base *b*.

   c) Una función unosBinario que, dado un número binario (100101), retorne un entero indicando la cantidad de 1s que tiene el mismo. *El parámetro de la función puede ser un entero que respete el formato binario (sólo acepte 0s y 1s).*

d) Una función binarioDecimal que, dado un número binario (100101), retorne el mismo número en base decimal (37).

8. Definir la función recursiva mcd (máximo común divisor), que dados dos números enteros positivos, retorne el máximo común divisor entre ellos.

9. Definir un procedimiento factores que muestre los factores primos de un número natural.

a) Los factores pueden mostrarse repetidos. Por ejemplo, dado el número 20 mostrar los números 2, 2 y 5.

b) Utilizando una lista, modificar el procedimiento para que muestre los factores sin repetir. Por ejemplo, dado el número 20 mostrar los números 2 y 5.

10. Definir la función recursiva combinatorio, que dados dos números enteros no negativos n y k (0 <= k <= n), retorne el número combinatorio en base a ellos.

11. Definir lo siguiente:

a) Una función recursiva digitos, que dado un número entero, retorne su cantidad de dígitos.

b) Una función recursiva reversaNum que, dado un número entero, retorne su imagen especular. Por ejemplo: reversaNum(345) = 543

c) Una función recursiva sumaDigitos que, dado un número entero, retorne la suma de sus dígitos.

d) Intente definir una función recursiva que retorne los dos valores anteriores a la vez como un par, aprovechando la recursión. Definirla como función y como procedimiento algorítmico.

12. Definiremos la función de Ackermann como una función de los enteros no negativos N y M sobre los enteros mediante las siguientes definiciones axiomáticas:

$$A(M, N) \begin{cases} N+1 & \text{si } M = 0 \\ A(M-1, 1) & \text{si } N = 0 \\ A(M-1, A(M, N-1)) & \text{si } M \neq 0, N \neq 0 \end{cases}$$

Definir la función recursiva ackermann que permita calcular A(M, N) para todo M>=0 y N>=0. Exprese cuál sería el orden bien fundado sobre los elementos del dominio.

13. La raíz cuadrada de un número positivo A puede calcularse mediante un proceso que va generando términos según la siguiente fórmula:

- $R_1 = 1$
- $R_i = 1 / 2 (R_{i-1} + (A / R_{i-1}))$ (para i > 1)

Definir la función recursiva raiz, que dado un número entero positivo A, retorne la raíz cuadrada aproximada de ese número.

14. Definir la función recursiva taylor, que dados dos valores reales x y tol, retorna el valor de sen(x) desarrollado utilizando el polinomio de Taylor. Esta función recursiva deberá generar la sumatoria siguiente, obteniendo términos sucesivos hasta encontrar uno menor al valor de tolerancia dado tol (en valor absoluto):

$$x - x^3/3! + x^5/5! - x^7/7! \ldots + (-1)^n * x^{2n+1}/(2n+1)!$$

15. Definir la operación pares, que dado un número entero, muestre todos los pares de números enteros positivos que son suma del número entero dado. Por ejemplo, 5 = (1 , 4), (2, 3).

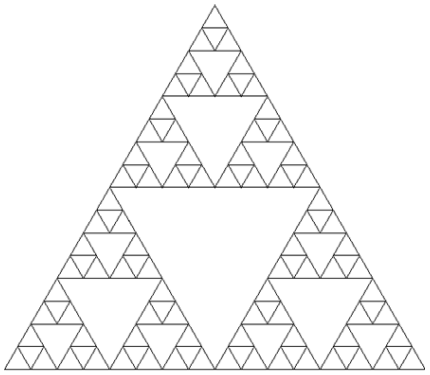
16.

a) Definir la función desdeHasta, que dados dos números enteros retorne una lista de números consecutivos donde el primer elemento de la lista resultante sea el primer elemento dado, y el último elemento de la lista resultante sea el segundo elemento dado.

b) Redefinir las funciones sumatoria y factorial utilizando desdeHasta.

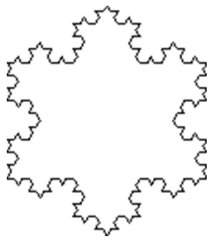
Parte II –Recursión en imágenes

17. Implementar un procedimiento recursivo sierpinski que permita dibujar el triángulo de Sierpinski de 4 niveles, tal como se describe a continuación:



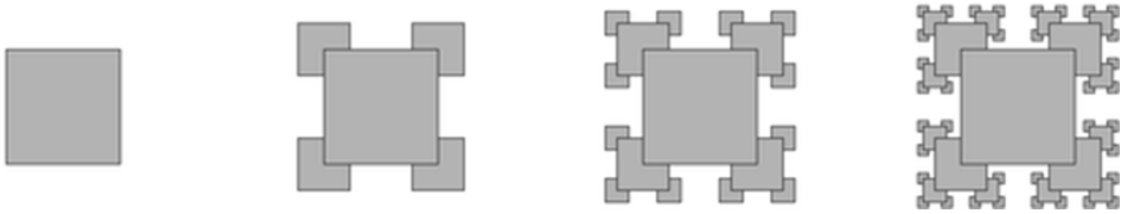
18. Implementar un procedimiento recursivo curvas que represente la curva del fractal conocido como copo de nieve de Koch, tal como se describe a continuación:

19. Implementar un procedimiento recursivo copos que represente el fractal copo de nieve de Koch, tal como se describe a continuación:

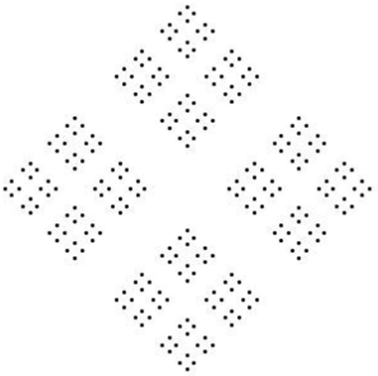


20. Implementar un procedimiento recursivo haches que permita dibujar el siguiente gráfico de 4 niveles, tal como se describe a continuación:

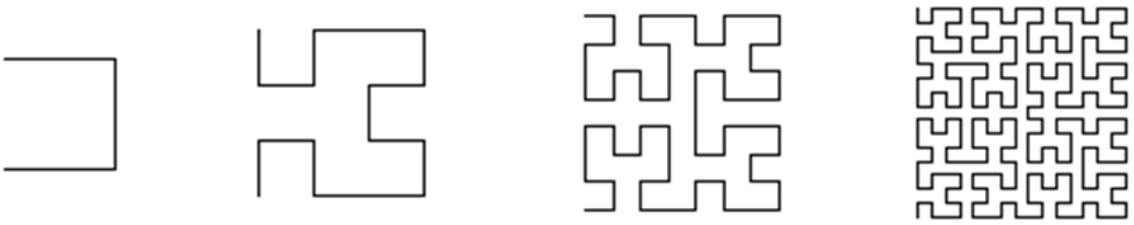
21. Implementar un procedimiento recursivo cuadrados que permita dibujar el siguiente gráfico de 4 niveles, tal como se describe a continuación:



22. Implementar un procedimiento recursivo rombos que permita dibujar el siguiente gráfico de 4 niveles.



23. Implementar un procedimiento recursivo hilbert que permita dibujar el siguiente gráfico de 4 niveles.



24. Implementar un procedimiento recursivo curvasierpinski que represente el siguiente gráfico de 3 niveles de recursión:

Parte III – Operaciones sobre estructuras recursivas

25. Implementar el TAD ListaEnt (lista de enteros) como estructura recursiva con sus operaciones constructoras y proyectores.

26. Implementar el TAD Lista(a) (lista con elementos del tipo genérico a) con sus operaciones más elementales, utilizando los tipos de datos adecuados para las siguientes variantes de implementación:

i. Listas simplemente encadenadas

ii. Listas doblemente encadenadas

iii. Arreglos unidimensionales estáticos (con una dimensión N grande)

27. Definir la función longitud, que dada una lista de enteros, retorne la cantidad de elementos que contiene.

28. Definir la función = (igualdad de listas) que dadas dos listas de enteros, retorne si ambas son iguales (contienen exactamente la misma cantidad de elementos, y poseen igualdad de elementos posición por posición).

29. Definir la función sumatoriaLista, que dada una lista de enteros, retorne la sumatoria de todos sus elementos.

30. Definir la función pertenece, que dada una lista de enteros, y un entero dado, retorne si el entero dado es igual a algún elemento de la lista dada.

31. Definir la función concatenacion, que dadas dos listas de enteros, retorne la lista resultante de la concatenación de la primera lista a la segunda lista.

32. Definir la función ultimo, que dada una listas de enteros, retorne su último elemento.

33. Definir una versión recursiva y otra versión no recursiva a la función penultimo, que dada una listas de enteros, retorne su penúltimo elemento.

34. Definir la función primeros, que dada una lista de enteros y un número n dado, retorne los primeros n elementos de la lista dada.

35. Definir la función posicion, que dada una lista de enteros y un número n dado, retorne el elemento de la lista correspondiente a la posición n.

36. Definir la función maximo, que dada una lista de enteros, retorne el máximo elemento de todos los que conforman a la lista.

37. Definir la función reversa, que dada una lista de enteros, retorne una lista de enteros con todos sus elementos invertidos en sus posiciones.

Por ejemplo: reversa([4,5,2,9]) = [9,2,5,4].

38.

a) Definir la función esPalindromo, que dada una lista de enteros, retorne si es o no es palíndromo, utilizando recursividad explícita.

b) Idem a), pero definiéndola a partir de otras funciones.

39. Definir la función cantidad, que dada una lista de enteros y un número n, retorne la cantidad de apariciones del número n en la lista dada.

40. Definir la función sublista, que dada una lista de enteros, un número que represente una posición y otro número que represente una longitud, devuelva una lista de enteros (que se basa en la lista dada) que comience en la posición dada y que tenga la longitud dada desde esa posición.

41. Definir la función intercalar, que dadas dos listas de enteros, retorne una lista de enteros que corresponda al intercalado elemento a elemento de las dos listas dadas.

Por ejemplo: intercalar([4,5,2], [3,8,9]) = [4,3,5,8,2,9].

42. Definir la función aplanar, que dada una lista de listas de enteros, retorne un lista de enteros que corresponda a la concatenación de elementos-lista de la lista original.

Por ejemplo: aplanar([[5,7], [], [3,7,2], [9]) = [5,7,3,7,2,9].

43. Definir la función longitudLL, que dada una lista de listas, retorne un número que sea la cantidad total de elementos acumulados entre todos sus elementos-listas. Utilizar la función aplanar.

44. Definir la función quicksort, que dada una lista de enteros, retorne la lista original, pero con sus elementos ordenados, utilizando el método de ordenamiento quicksort.

45. Definir la función partes, que dada una lista de enteros, retorne una lista de listas de enteros, en que cada elemento de la lista resultado sea cada una de las sublistas de la lista original (respetando la posición).

Por ejemplo: partes([6,2,3]) = [[],[6],[2],[3],[6,2],[6,3],[2,3],[6,2,3]].

(no necesariamente en ese orden)

46. Definir la función permutaciones, que dada una lista de enteros, retorne una lista de listas de enteros, donde cada lista-elemento es cada una de las posibles permutaciones de la lista original.

Por ejemplo: permutaciones([6,2,3]) = [[6,2,3], [6,3,2], [2,3,6], [2,6,3], [3,2,6], [3,6,2]].

(no necesariamente en ese orden)

47. Definir la función todosConTodos, que dadas dos listas de enteros, retorne una lista de pares de enteros que formen parte de combinaciones de elementos de la primera lista con elementos de la segunda lista.

Por ejemplo: todosConTodos([6,2,3],[7,5]) = [(6,7), (6,5), (2,7), (2,5), (3,7), (3,5)].

48. Definir la función todosConTodosN generalizada de la anterior, que dadas n listas de enteros (una lista de listas de enteros), retorne una lista de listas de enteros en que en cada lista-elemento exista un elemento de cada una de las listas-elemento de la lista original.

Por ejemplo: todosConTodosN([[6,2,3],[7,5],[9,4]]) = [[6,7,9], [6,7,4], [6,5,9], [6,5,4], [2,7,9], [2,7,4], [2,5,9], [2,5,4], [3,7,9], [3,7,4], [3,5,9], [3,5,4]].
(no necesariamente en ese orden)

49. Reespecificar el ya definido TAD ListaEnt para que pueda aceptar elementos de cualquier tipo/TAD, aunque deberá ser el mismo para todos sus elementos. Llamar a este nueva TAD Lista(a).

50. Implementar el TAD Nat (números naturales) de forma recursiva, con operaciones constructoras y las de suma y producto.

51. Implementar el TAD Nat al igual que el punto anterior pero utilizando una estructura a partir de las listas.

52. Especificar el TAD Conjunto(a) (conjunto de elementos de tipo a, donde los elementos no se repiten y no tienen una posición en el conjunto) con operaciones de agregar elemento, eliminar elemento, pertenece, y conversión a lista. Definirlo a partir del TAD de las listas.

53. Implementar el procedimiento recursivo mostrarElementos, que dada una lista, imprima el contenido de todos sus elementos. Implementar este procedimiento sobre cada una de las estructuras implementadas del tipo lista.

54. Implementar el procedimiento recursivo posicionesPares, que dado un arreglo de enteros, imprima el contenido de sus posiciones pares.

55. Implementar la función recursiva productoEscalar, que dados dos vectores de enteros, retorne un entero que represente el producto escalar de ambos.

56. Implementar el procedimiento recursivo busquedaBinaria, que dado un vector ordenado de enteros y un entero dado, retorne si el entero dado pertenece a alguna posición del vector. La búsqueda deberá efectuarse con la técnica de búsqueda binaria.

57. Implementar el procedimiento recursivo mostrarElementosTriangular, que dado un vector A, retorne todos sus elementos de la siguiente forma:

$A_1, A_2, ..., A_N$

$A_2, ..., A_N$

.....

A_N

58. Implementar el TAD MatrizCuadrada(a), con constructores propios, y con proyectores y operaciones elementales, utilizando las siguientes estructuras:

a) Listas simplemente encadenadas de listas simplemente encadenadas

b) Arreglos bidimensionales estáticos (con una dimensión N grande)

59. Definir e implementar la función minimoCuad, que dada una matriz cuadrada, retorne su elemento mínimo.

60. Definir e implementar el procedimiento maximosCuad, que dada una matriz cuadrada, retorne una lista con los valores máximos correspondientes a cada una de sus filas.

61. Definir e implementar el procedimiento diagonalPrincipal, que dada una matriz cuadrada, retorne una lista que corresponda a cada uno de los elementos que conformen su diagonal principal (de la primera fila a la última).

62. Definir e implementar la función esSimetrica, que dada una matriz cuadrada, retorne si es simétrica con respecto a su diagonal principal.

63. Definir e implementar el procedimiento triangularInferior, que dada una matriz cuadrada, retorne una lista de listas que corresponda a cada una de la filas de la matriz que conformen su triangular inferior (de la segunda fila a la última). En la implementación, no debe recorrer elementos innecesarios de la matriz.

64. Definir e implementar el procedimiento cantidadElementoDiagonal, que dada una matriz cuadrada de caracteres, retorne una lista que indique por cada columna, la cantidad de veces que se repite el carácter ubicado en la diagonal principal de la matriz.

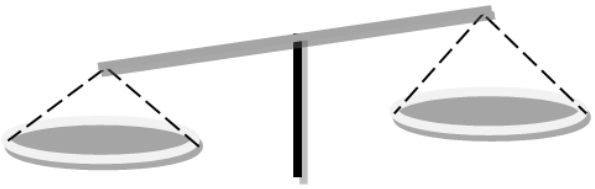
Por ejemplo:

A	B	H	P
A	I	J	P
K	I	H	L
A	N	H	O

3
2
3
1

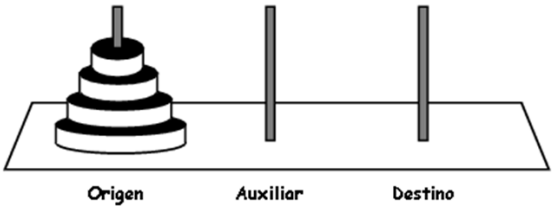
Parte IV – Problemas recursivos

65. Dada una pila de monedas de misma denominación donde se sabe que una de ellas es falsa, implementar un procedimiento que detecte cuál es la moneda falsa. La característica que marca la diferencia es el peso de la moneda, donde la falsa siempre pesa menos que una original. Pensar el procedimiento teniendo en cuenta que sólo se dispone de una balanza de dos platos y no se puede saber el peso exacto de cada moneda.



66. Implementar una mejora al procedimiento del punto anterior donde se utilicen menos pesajes de monedas para encontrar la que es falsa. Por ejemplo, si se tuviera una pila de 30 monedas, el procedimiento tendría que resolverlo en 4 pesajes como máximo.

67. El juego de las Torres de Hanoi consiste de tres postes, llamados Origen, Destino y Auxiliar, y *n* discos de diferentes tamaños apilados en el poste Origen, desde el de mayor tamaño en su base hasta el de menor tamaño en su tope. El objetivo del juego es dejar todos los *n* discos en el poste Destino, también ordenados por tamaño, como estaban originalmente en el poste Origen. Para lograr el objetivo, es necesario realizar movimientos de discos entre postes. Cada movimiento consiste en la extracción del disco ubicado en el tope de un poste, y su ubicación en otro poste, siempre que si existiera un disco en el tope del otro disco, el tamaño de éste deberá ser mayor al del disco a mover en ese poste. Se puede utilizar el poste Auxiliar para mover temporalmente algunos discos.



a) Implementar el procedimiento recursivo hanoi, que dado un número n que represente la cantidad de discos del juego, imprima la sucesión de movimientos necesarias para resolver el problema. Cada movimiento se representa como un par (poste desde - poste hasta).

b) Demostrar por inducción utilizando el procedimiento implementado en el punto a) que la cantidad de movimientos del juego para n discos es 2^n-1.

68. Implementar el procedimiento recursivo laberinto, que dada una lista de coordenadas (x,y) que representan casillas libres de un laberinto bidimensional (las no indicadas representan “paredes”), una coordenada de salida, una dirección obligatoria de salida y una coordenada de llegada, retorne una lista de movimientos posibles “continuos” desde la coordenada de salida hasta la de llegada, realizando en cada paso movimientos que sólo pueden ser arriba, abajo, izquierda y derecha.

Ayuda: Se puede considerar como criterio, la recorrida exhaustiva por el laberinto desde la posición inicial hasta la final, volviendo atrás en los pasos ya recorridos si se concluyera que ese camino no llega a la solución. También deberá considerarse el camino ya recorrido para no volverlo a repetir en el caso de algún cruce, ya que puede producirse un ciclo sin fin.

69. Implementar el procedimiento recursivo ochoReinas, que dé cómo resultado una lista de ocho coordenadas de un tablero de 8 x 8, que representen las ubicaciones de las 8 reinas en el mismo, tal que no existan reinas que se ataquen entre sí como en el juego de Ajedrez.

70. Explicar si puede evitarse el uso del tipo puntero en la implementación de TADs de estructuras de datos recursivas. Justifique su respuesta.

Parte V – Recursión de pila y de cola

71. Identificar la recursividad de pila y de cola en las siguientes operaciones implementadas de ejercicios anteriores:

- sumatoria
- factorial
- producto
- pares
- desdeHasta
- pertenece
- concatenacion
- reversa
- intercalar
- todosConTodos

a) Reemplazar en los casos anteriores la recursividad de cola por iteración.

b) Reemplazar en los casos anteriores la recursividad de pila por el uso de una estructura de pila explícita.

72. Reimplementar la función mcd expresada como recursividad de cola, reemplazándola por iteraciones.

73. Definir la función recursiva sumaAlternada, que dada una lista de enteros, retorne el resultado de evaluar el primer elemento menos el segundo más el tercero, menos el cuarto, y así sucesivamente, sin importar la cantidad de elementos que la lista posea.

Definir una versión utilizando recursividad de pila, y otra versión utilizando recursividad de cola.

Por ejemplo: sumaAlternada([4,5,3,8,7]) = 4-5+3-8+7 = 1

74. Reimplementar las siguientes funciones y procedimientos expresados con recursividad de pila, por el uso de una estructura de pila explícita.

- fibonacci
- combinatorio
- ackermann
- sierpinski
- partes
- hanoi