MEMORIA PRÁCTICA 1

Victoria Pelayo e Ignacio Rabuñal.

-**Ejercicio1**

1.1 Para este ejercicio, a parte de la función pedida por el enunciado, hemos creado una auxiliar.

Una que realiza el producto escalar entre dos vectores

Ejemplos de ejecución, con la función recursiva (hemos comprobado con una calculadora que el resultado es el correcto):

- (cosine-distance-rec '(1 2) '(1 2 3)) → 0,40238577
- (cosine-distance-rec nil '(1 2 3)) → nil
- (cosine-distance-rec '() '()) → nil
- (cosine-distance-rec '(0 0) '(0 0)) → nil

Ejemplos de ejecución, con la función que utiliza mapcar:

- (cosine-distance-mapcar '(1 2) '(1 2 3)) → 0,40238577
- (cosine-distance-mapcar nil '(1 2 3)) → nil
- (cosine-distance-mapcar '() '()) → nil
- (cosine-distance-mapcar '(0 0) '(0 0)) → nil

1.2 Para este apartado hemos creado una función auxiliar que devuelve la lista de tuplas de la confianza y el vector correspondiente.

Como no se especifica nada en el enunciado, para calcular la distancia coseno utilizamos la función implementada con mapcar.

Ejemplos de ejecución:

- (order-vectors-cosine-distance '(1 2 3) '((32 454 123) (133 12 1) (4 2 2)) 0.99)
  NIL
- (order-vectors-cosine-distance '(1 2 3) '((32 454 123) (133 12 1) (4 2 2)) 0.3)
  ((4 2 2) (32 454 123) (133 12 1))
- (order-vectors-cosine-distance '(1 2 3) '((32 454 123) (133 12 1) (4 2 2)) 0.5)
  ((4 2 2) (32 454 123)) NIL
- (order-vectors-cosine-distance '(1 2 3) '()) NIL
- (order-vectors-cosine-distance '() '((4 3 2) (1 2 3))) NIL

1.3 y 1.4.

Para esta última función hemos creado dos funciones auxiliares. Una para calcular la distancia coseno entre cada una de las categorías y un vector; y otra que elige la mejor categoría para un vector.

Pruebas de ejecución:

- (get-vectors-category '(()) '(()) #'cosine-distance-rec) NIL

- (get-vectors-category '(()) '(()) #'cosine-distance-mapcar) NIL

- (get-vectors-category '((1 4 2) (2 1 2)) '((1 1 2 3)) #'cosine-distance-mapcar) ((2 0.40238577))

- (get-vectors-category '((1 4 2) (2 1 2)) '((1 1 2 3)) #'cosine-distance-rec) ((2 0.40238577))

- (get-vectors-category '(()) '((1 1 2 3) (2 4 5 6)) #'cosine-distance-rec) ((NIL NIL) (NIL NIL))

- (get-vectors-category '(()) '((1 1 2 3) (2 4 5 6)) #'cosine-distance-mapcar) ((NIL NIL) (NIL NIL))

Pruebas de tiempos de ejecucion:

- (time (get-vectors-category '(()) '(()) #'cosine-distance-rec))
  `Real time: 2.0E-6 sec.`
- (time (get-vectors-category '(()) '(()) #'cosine-distance-mapcar))
  `Real time: 2.0E-6 sec.`
- (time (get-vectors-category '((1 4 2) (2 1 2)) '((1 1 2 3)) #'cosine-distance-rec))
  `Real time: 3.0E-5 sec.`
- (time (get-vectors-category '((1 4 2) (2 1 2)) '((1 1 2 3)) #'cosine-distance-mapcar))
  `Real time: 2.3E-5 sec.`
- (time (get-vectors-category '(()) '((1 1 2 3) (2 4 5 6)) #'cosine-distance-rec))
  `Real time: 2.0E-5 sec.`
- (time (get-vectors-category '(()) '((1 1 2 3) (2 4 5 6)) #'cosine-distance-mapcar))
  `Real time: 1.8E-5 sec.`

Observamos que excepto en las primeras pruebas de ejecución (las que calculan la distancia coseno de listas vacias) el tiempo usando mapcar es ligeramente menor que usando la funcion recursiva.

EJERCICIO 2.

En todo el ejercicio no hemos utilizado funciones auxiliares

2.1

La función 'newton' que hemos implementado acaba cuando se sobrepase el numero máximo de iteraciones o cuando converge para la tolerancia exigida. Si no se da el caso, la funcion continua con llamadas recursivas a la función utilizando como argumentos un número menos de iteraciones y el nuevo x0 dado por la ecuacion de newton.

Pruebas de ejecucion:

- (newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 20 3.0)

  ```
  4.0
  ```
- (newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 20 0.6)

  ```
  0.99999946
  ```
- (newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 30 -2.5)

  ```
  -3.0000203
  ```
- (newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 10 100)

  ```
  NIL
  ```

2.2

La función 'one-root-newton' la hemos implementado de manera recursiva. Si la lista de semillas es nula, la funcion acaba y devuelve nil. Se ejecuta la función para la primera semilla de la lista y continua con llamadas recursivas sobre (rest semillas)

Pruebas de ejecucion:

- (one-root-newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 20 '(0.6 3.0 -2.5))

  ```
  0.99999946
  ```
- (one-root-newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 20 '(3.0 -2.5))

  ```
  4.0
  ```
- (one-root-newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 1 '(3.0 -2.5))

  ```
  NIL
  ```

2.3

La función 'all-roots-newton' ha sido implementada usando 'mapcar' y una expresion lambda para aplicar a todas las semillas la función 'newton'.

Pruebas de ejecucion:

- (all-roots-newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 20 '(0.6 3.0 -2.5))

  ```
  (0.99999946 4.0 -3.0000203)
  ```

- (all-roots-newton #'(lambda(x) (* (- x 4) (- x 1) (+ x 3))) #'(lambda (x) (- (* x (- (* x 3) 4)) 11)) 1 '(0.6 3.0 -2.5))

  ```
  (NIL NIL NIL)
  ```

2.3.1

Para la función 'list-not-nil-roots-newton' hemos utilizado 'mapcan' junto con 'nconc' para eliminar los nil de la lista devuelta por 'all-roots-newton'.

EJERCICIO3.

3.1

Para este apartado no hemos necesitado crear ninguna función auxiliar.

Ejemplos de ejecución:

- (combine-elt-lst 'a '(1 2 3)) → ((A 1) (A 2) (A 3))
- (combine-elt-lst nil nil) → NIL
- (combine-elt-lst nil '(a b)) → ((NIL A) (NIL B))

3.2

En este ejercicio teníamos que crear una función que hiciese el producto cartesiano de dos listas.

Ejemplo de ejecución:

- (combine-lst-lst nil nil) → NIL
- (combine-lst-lst '(a b c) nil) → NIL
- (combine-lst-lst nil '(a b c)) → NIL

Si una de las dos listas es NIL el producto cartesiano es NIL.

3.3

En este apartado hay que crear una función que combine todos los elementos de varias listas dadas.

Para ello hemos creado una función auxiliar que es una modificación de la del ejercicio1, esta modificación evita los problemas de paréntesis.

Y otra función auxiliar que combina todos los elementos de dos listas.

Ejemplos de ejecución:

- (combine-list-of-lsts '(() (+ -) (1 2 3 4))) → ((+ 1) (+ 2) (+ 3) (+ 4) (- 1) (- 2) (- 3) (- 4))

  (Si una lista es nil, contamos como si no estuviese)

- (combine-list-of-lsts '((a b c) () (1 2 3 4))) → ((A 1) (A 2) (A 3) (A 4) (B 1) (B 2) (B 3) (B 4) (C 1) (C 2) (C 3) (C 4))
- (combine-list-of-lsts '((a b c) (1 2 3 4) ())) → ((A . 1) (A . 2) (A . 3) (A . 4) (B . 1) (B . 2) (B . 3) (B . 4) (C . 1) (C . 2) (C . 3) (C . 4))
- (combine-list-of-lsts '((1 2 3 4))) → ((1) (2) (3) (4))
- (combine-list-of-lsts '(nil)) → NIL