MEMORIA DE LA PRÁCTICA 3:

ENTRADA Y SALIDA

Modelos Avanzados de Computación

Universidad de Huelva

Grado en Ingeniería Informática
Especialidad en Computación
Curso 2023/24
Manuel Ramírez Ballesteros

Índice:

1- Introducción
2- Implementación del menú de búsquedas
3- Ejemplos de ejecución
4- Conclusión

1- Introducción

En esta memoria se desarrollará la implementación de un juego de adivinanzas en Haskell en el cual podremos pensar un número y dar el conjunto de elementos entre los que queremos que el programa busque el elemento que hemos pensado. Esto se realizará mediante un menú que nos ofrezca distintos tipos de búsquedas y una opción de salir. Cabe destacar que el conjunto de elementos que se introduzcan deben ser números enteros, sino se produciría un error debido al mal uso de la aplicación.

Los tipos de búsquedas que se ofrecen son:

- Búsqueda secuencial: Recorrerá todos los elementos del conjunto de entrada uno a uno, ofreciéndoselos al usuario como solución al número que ha pensado. Solo filtrará los elementos repetidos sin tener en cuenta el orden.
- Búsqueda binaria: Esta búsqueda parte de un conjunto de elementos sin repetidos y ordenados y preguntará si el elemento situado en la mitad del conjunto es el pensado por el usuario. En caso de no acertar en este intento, se preguntará al usuario si el número que ha pensado es mayor o menor que el elemento de la mitad y se trabajará de manera similar con la mitad del conjunto adecuada.
- Búsqueda por interpolación: Esta búsqueda parte de un conjunto de elementos sin repetidos y ordenados y preguntará si el elemento del conjunto más cercano a la media de los elementos es el pensado por el usuario. En caso de no acertar en este intento, se preguntará al usuario si el número que ha pensado es mayor o menor que el elemento de la mitad y se trabajará de manera similar con la porción del conjunto adecuada.

Para la estructura del programa se ha creado tres módulos Main, Busqueda y Auxiliar: En Main tendremos el menú principal que se ha descrito, en Busqueda contaremos con las tres funciones de busqueda y en Auxiliar estarán el resto funciones empleadas en el programa.

2- Implementación del menú de búsquedas

Como se describe en las especificaciones de diseño, el menu inicialmente borrará la pantalla y nos mostrará lo siguiente:

```haskell
module Main
( main
) where

import Busqueda
import Auxiliar
import System.Info
import System.Process
import Data.List (nub)

main :: IO()
main = do
    putStrLn ""
    cls
    putStrLn "Alumno : Manuel Ramirez Ballesteros"
    putStrLn "-------------------------------------"
    putStrLn ""
    putStrLn "Piensa un numero.."
    putStrLn ".."
    putStrLn ".."
    putStrLn "-------------------------------------"
    putStrLn ""
    putStrLn "Tipos de busqueda : "
    putStrLn "1 - Busqueda secuencial"
    putStrLn "2 - Busqueda binaria"
    putStrLn "3 - Busqueda por interpolacion"
    putStrLn "4 - Salir"
    putStrLn ""
    putStrLn "Escriba la opcion que desee :"
    t <- getLine
    putStrLn ""
```

Donde se escogerá el tipo de búsqueda entre las 4 opciones propuestas. Sólo haste escoger la opción 4, podremos salir del programa. Sino jugaremos hasta que el programa acierte el número pensado. Para ello se ha propuesto lo siguiente:

```haskell
    t <- getLine
    putStrLn ""
    case t of
        "1" -> do
            putStrLn "Ingrese el vector de elementos en el que desea buscar su numero separados por espacios : "
            vec <- getLine
            let v = nub $ convertirVector vec
            putStrLn $ "El vector ingresado es = " ++ show v
            busqueda_secuencial v
            main
        "2" -> do
            putStrLn "Ingrese el vector de elementos en el que desea buscar su numero separados por espacios : "
            vec <- getLine
            let v = nub $ quicksort $ convertirVector vec
            busqueda_binaria v
            main
        "3" -> do
            putStrLn "Ingrese el vector de elementos en el que desea buscar su numero separados por espacios : "
            vec <- getLine
            let v = nub $ quicksort $ convertirVector vec
            busqueda_interpolacion v
            main
        "4" -> do
            putStrLn "¡Fin del juego!"
            putStrLn ""
            return ()
        _ -> do
            putStrLn "Error : Opcion no disponible.."
            putStrLn "Pulse cualquier tecla para continuar.."
            getLine
            main
```

Donde solo saldremos con la opción 4. Con las opciones 1, 2 ó 3, buscaremos el número de la manera que se indique y, si la opción no es ninguna de las anteriores, se indicará que dicha opción no está disponible y esperaremos a que el usuario escriba algo para continuar, dando tiempo así de que lea el mensaje antes de borrar pantalla.

Para los diferentes tipos de búsqueda se han implementado las siguientes funciones:

- ***Búsqueda secuencial :***

```haskell
module Busqueda
( busqueda_secuencial
, busqueda_binaria
, busqueda_interpolacion
) where

import Auxiliar

busqueda_secuencial :: [Int] -> IO ()
busqueda_secuencial (c:r) = do
    if r == [] then do
            putStrLn $ "¿Su numero es : " ++ show c ++ "?"
            res <- getLine
            if res == "NO" then do
                    putStrLn $ "El numero pensado no parece estar en la lista.. Pruebe a jugar de nuevo.."
                    putStrLn "Pulse cualquier tecla para continuar : "
                    x <- getLine
                    putStrLn ""
            else if res == "SI" then do
                    putStrLn "¡Numero acertado!"
                    putStrLn "Pulse cualquier tecla para continuar : "
                    x <- getLine
                    putStrLn ""
            else do
                putStrLn "Respuesta incorrecta : SI o NO"
                putStrLn ""
                busqueda_secuencial (c:r)
    else do
            putStrLn $ "¿Su numero es : " ++ show c ++ "?"
            res <- getLine
            if res == "NO" then
                busqueda_secuencial r
            else if res == "SI" then do
                putStrLn ""
                putStrLn "¡Numero acertado!"
                putStrLn ""
            else do
                putStrLn ""
                putStrLn "Respuesta incorrecta : SI o NO"
                putStrLn ""
                busqueda_secuencial (c:r)
```

Para esta búsqueda se ha distinguido el caso de que haya un solo elemento o no. Si sólo hay uno, preguntamos si es el que ha pensado y, en caso de fallar en el intento, comunicamos el fallo de no incluir el elemento en la lista. También se avisa en caso de introducir una respuesta diferente a SI o No. Cuando hay más de un elemento, se ofrece el primero como el elemento que ha pensado el usuario y, en caso de fallo, llamamos recursivamente a la función. Se han incorporado varias salidas por pantalla para observar el seguimiento de la tarea en todas las búsquedas.

- ***Búsqueda binaria :***

```
busqueda_binaria :: [Int] -> IO ()
busqueda_binaria [] = do
    putStrLn "La lista esta vacia. ¡Fin del juego!"
    putStrLn ""
    putStrLn "Pulse cualquier tecla para continuar : "
    x <- getLine
    putStrLn ""

busqueda_binaria l = do
    putStrLn $ "El vector ingresado es = " ++ show l
    let long = length l
    let medio = div long 2
    let elem = l !! medio
    putStrLn $ "¿Su numero es : " ++ show elem ++ "?"
    respuesta <- getLine
    case respuesta of
        "NO" -> do
            putStrLn $ "¿Su numero es mayor que " ++ show elem ++ "?"
            res <- getLine
            if res == "SI" then
                    busqueda_binaria (drop (medio + 1) l)
            else if res == "NO" then
                    busqueda_binaria (take medio l)
            else do
                    putStrLn ""
                    putStrLn "La respuesta debe ser: SI o NO"
                    busqueda_binaria l
        "SI" -> do
            putStrLn ""
            putStrLn "¡Numero acertado!"
            putStrLn "Pulse cualquier tecla para continuar : "
            x <- getLine
            putStrLn ""
        _ -> do
            putStrLn ""
            putStrLn $ "Respuesta incorrecta : SI o NO"
            putStrLn ""
            busqueda_binaria l
```

En esta búsqueda se avisará al usuario si la lista está vacía para que repita el juego o se obtendrá el elemento situado en la mitad de la lista de enteros y se propondrá dicho elemento como el pensado por el usuario. En caso de fallar, se preguntará si el elemento pensado el mayor o menor y se reducirá el conjunto de elementos ordenados en función de su respuesta, quedándonos con la mitad mayor o menor.

- ***Búsqueda por interpolación:***

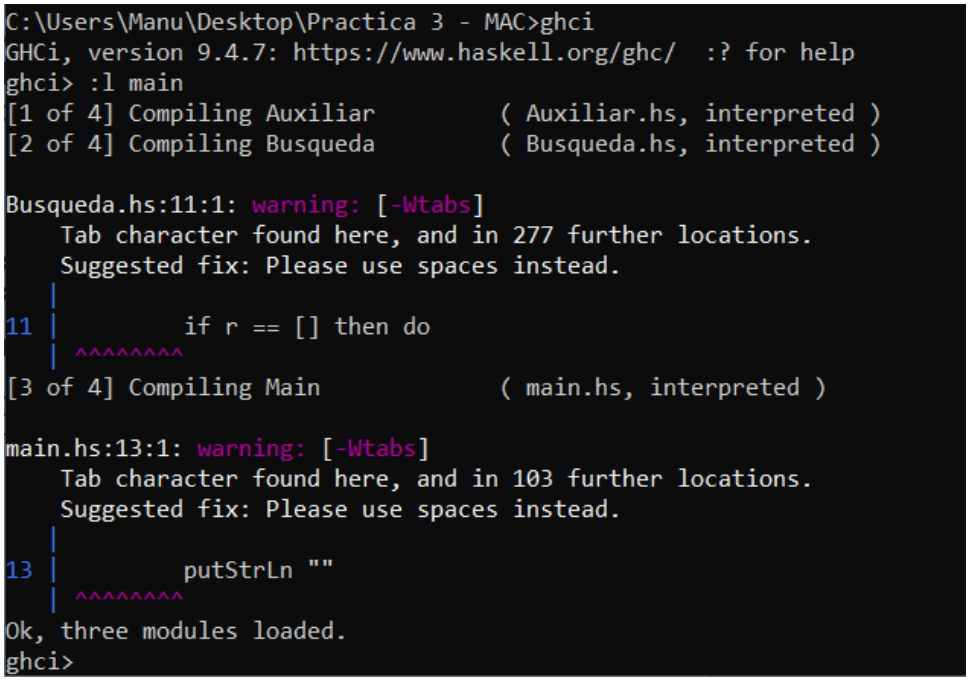
```
busqueda_interpolacion :: [Int] -> IO ()
busqueda_interpolacion [] = do
    putStrLn "La lista esta vacia. ¡Fin del juego!"
    putStrLn ""
    putStrLn "Pulse cualquier tecla para continuar : "
    x <- getLine
    putStrLn ""

busqueda_interpolacion l = do
    putStrLn $ "El vector ingresado es = " ++ show l
    let media = div (sum l) (length l)
    let elem = elementoCerca l media
    putStrLn $ "¿Su numero es : " ++ show elem ++ "?"
    respuesta <- getLine
    if respuesta == "NO" then do
            putStrLn $ "¿Su numero es mayor que " ++ show elem ++ "?"
            res <- getLine
            putStrLn ""
            if res == "SI" then do
                busqueda_interpolacion (dropWhile (<= elem) l)
            else do
                busqueda_interpolacion (takeWhile (/= elem) l)
    else if respuesta == "SI" then do
            putStrLn ""
            putStrLn "¡Numero acertado!"
            putStrLn ""
            putStrLn "Pulse cualquier tecla para continuar : "
            x <- getLine
            putStrLn ""
    else do
            putStrLn ""
            putStrLn "Respuesta incorrecta : SI o NO"
            putStrLn ""
            busqueda interpolacion l
```

En esta búsqueda se avisará al usuario si la lista está vacía para que repita el juego o se obtendrá el elemento más cercano a la media de los elementos de la lista de enteros y se propondrá dicho elemento como el pensado por el usuario. En caso de fallar, se preguntará si el elemento pensado el mayor o menor y se reducirá el conjunto de elementos ordenados en función de su respuesta, quedándonos con la mitad mayor o menor.

3- Ejemplos de ejecución

Este programa podemos ejecutarlo de dos formas distintas:

En primer lugar, podemos hacerlo desde la cmd desde el directorio que contenga los tres módulos usando el entorno de programación interactivo GHCi. Para ello hacemos lo siguiente:



No tendremos en cuenta los warnings debidos al tabulado excesivo para mejor visualización del código, pero podrían corregirse fácilmente como en auxiliar.hs.

Al introducir en la consola interactiva “main”, arrancaremos el programa principal y veremos lo siguiente:

```
Alumno : Manuel Ramirez Ballesteros
-------------------------------------

Piensa un numero..
..
..
-------------------------------------

Tipos de busqueda :
1 - Busqueda secuencial
2 - Busqueda binaria
3 - Busqueda por interpolacion
4 - Salir

Escriba la opcion que desee :
```

Si escogemos la opción 1 tenemos:

```
Escriba la opcion que desee :
1

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :
1 3 2 2 4 5 7 1 1
El vector ingresado es = [1,3,2,4,5,7]
¿Su numero es : 1?
NO
¿Su numero es : 3?
no

Respuesta incorrecta : SI o NO

¿Su numero es : 3?
NO
¿Su numero es : 2?
NO
¿Su numero es : 4?
SI
```

Para la opción 2:

```
Escriba la opcion que desee :
2

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :
1 2 3 4 5 6 88 3 1 2 3 1 1 10
El vector ingresado es = [1,2,3,4,5,6,10,88]
¿Su numero es : 5?
NO
¿Su numero es mayor que 5?
SI
El vector ingresado es = [6,10,88]
¿Su numero es : 10?
NO
¿Su numero es mayor que 10?
NO
El vector ingresado es = [6]
¿Su numero es : 6?
Sii

Respuesta incorrecta : SI o NO

El vector ingresado es = [6]
¿Su numero es : 6?
SI

¡Numero acertado!
Pulse cualquier tecla para continuar :
```

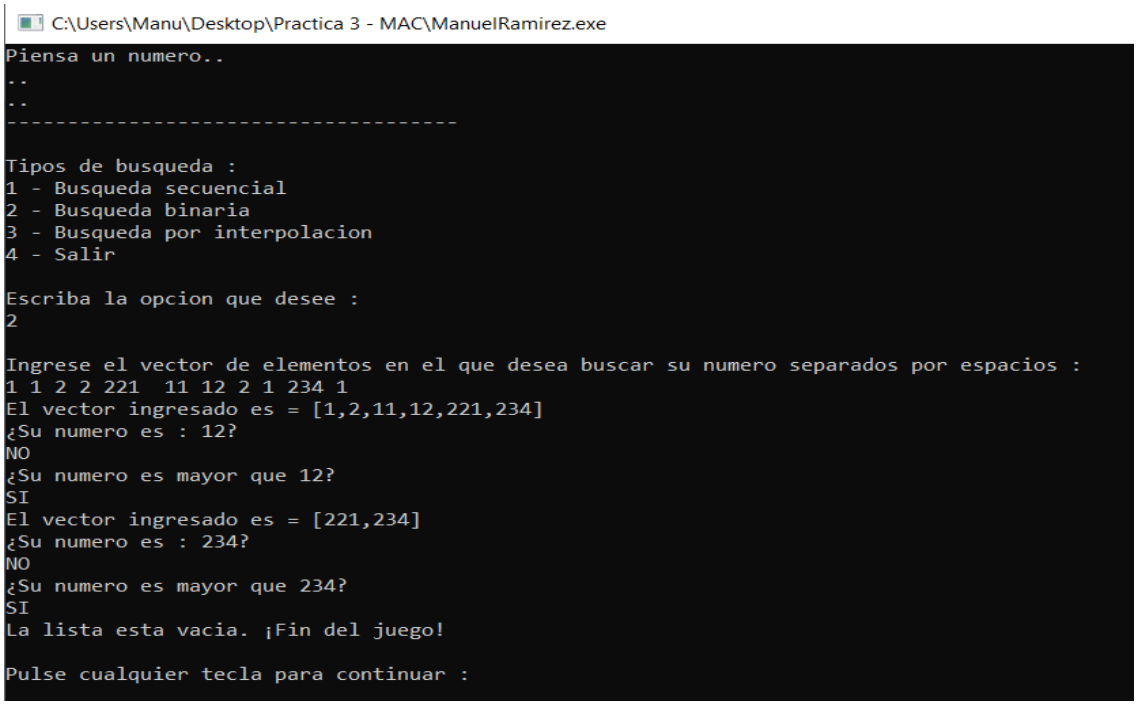
Para la opción 3:

```
Escriba la opcion que desee :
3

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :
1 2 3 4 52 1 12 4 1 2 2 1 23 21
El vector ingresado es = [1,2,3,4,12,21,23,52]
¿Su numero es : 12?
NO
¿Su numero es mayor que 12?
SI

El vector ingresado es = [21,23,52]
¿Su numero es : 23?
NO
¿Su numero es mayor que 23?
NO

El vector ingresado es = [21]
¿Su numero es : 21?
SI

¡Numero acertado!

Pulse cualquier tecla para continuar :
```

Podemos ver también lo que ocurre al incluir una lista vacía:

```
Escriba la opcion que desee :
3

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :

La lista esta vacia. ¡Fin del juego!

Pulse cualquier tecla para continuar :
```

También podemos probar la aplicación simplemente con el ejecutable ManuelRamirez.exe, como vemos a continuación:

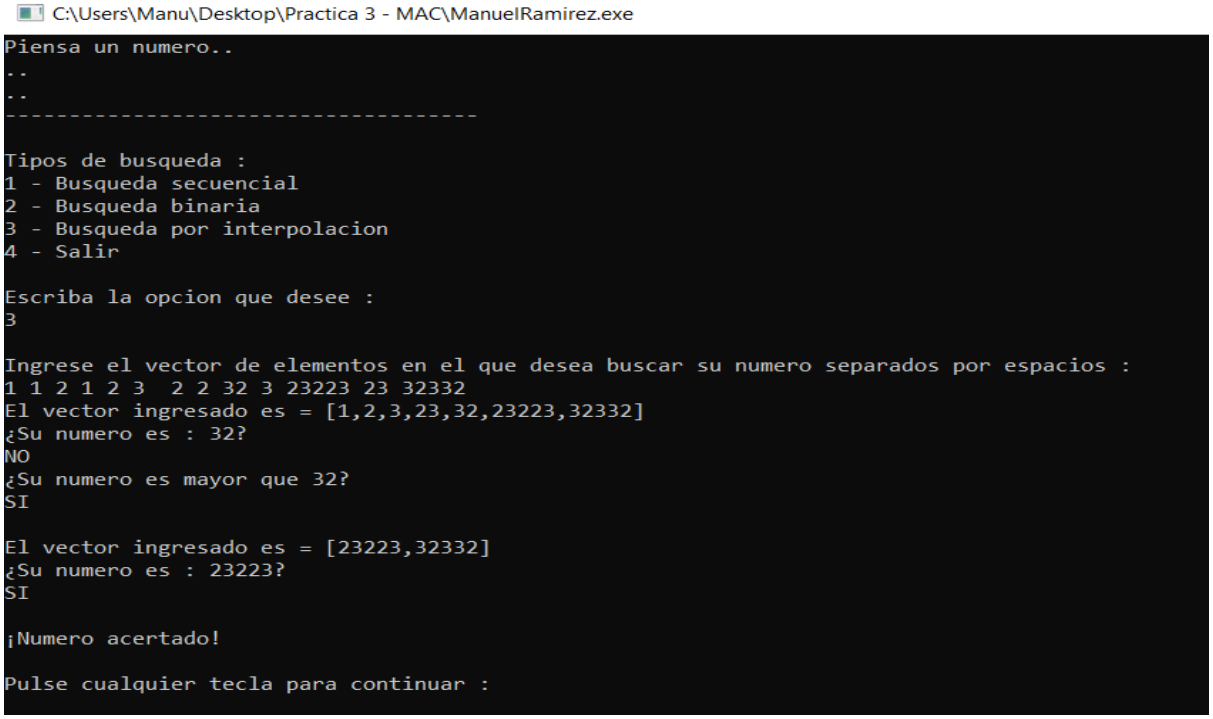

C:\Users\Manu\Desktop\Practica 3 - MAC\ManuelRamirez.exe
Piensa un numero..
..
..

Tipos de busqueda :
1 - Busqueda secuencial
2 - Busqueda binaria
3 - Busqueda por interpolacion
4 - Salir

Escriba la opcion que desee :
2

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :
1 1 2 2 221 11 12 2 1 234 1
El vector ingresado es = [1,2,11,12,221,234]
¿Su numero es : 12?
NO
¿Su numero es mayor que 12?
SI
El vector ingresado es = [221,234]
¿Su numero es : 234?
NO
¿Su numero es mayor que 234?
SI
La lista esta vacia. ¡Fin del juego!

Pulse cualquier tecla para continuar :


C:\Users\Manu\Desktop\Practica 3 - MAC\ManuelRamirez.exe
Piensa un numero..
..
..

Tipos de busqueda :
1 - Busqueda secuencial
2 - Busqueda binaria
3 - Busqueda por interpolacion
4 - Salir

Escriba la opcion que desee :
3

Ingrese el vector de elementos en el que desea buscar su numero separados por espacios :
1 1 2 1 2 3 2 2 32 3 23223 23 32332
El vector ingresado es = [1,2,3,23,32,23223,32332]
¿Su numero es : 32?
NO
¿Su numero es mayor que 32?
SI

El vector ingresado es = [23223,32332]
¿Su numero es : 23223?
SI

¡Numero acertado!

Pulse cualquier tecla para continuar :

4- Conclusión

Con este programa hemos trabajado la creación de módulos y la importación de librerías en Haskell, así como el tipo IO en Haskell para leer y pedir datos por teclado o mostrarlos por pantalla. Además, se ha trabajado más libremente en un proyecto más elaborado, tratando de manejar todo lo posible los errores y las necesidades del programa.

Además de estos tres módulos mencionados y de este documento, se incluye en el zip un ejecutable del programa para poder probarlo fácilmente sin usar la cmd, aunque hay un problema a la hora de borrar pantalla en el exe que no he conseguido solucionar, pero que por cmd no ocurre.