Reporte Practica 0

López Cortés Adamari Gianina - 320268458
Peña Mata Juan Luis - 317100664

February 2025

1 Recursión

- Buscar (con recursión de cola): Primero "buscar" llama a la función auxiliar "buscar_aux". El acumulador (acc) se usa para almacenar el estado de la búsqueda.
  Despues se declara la función auxiliar "buscar_aux", que devuelve el booleano. El caso base de la recursión ed la lista vacia [], se devulve Falso (que es el valor inicial del acumulador). En el paso recursivo, se busca si "x" es igual a "elemento", si es asi entonces devuelve True, sino pues se hace recursion en "buscar_aux" con el resto de la lista "xs" con "elemento" para ver si son igual, hasta que la lista este vacia, entonces devolvera False.

- Sumar (con recursión de cola): Primero "sumar_lista" llama a la función auxiliar "sumar_aux" y con el acumulador (acc) iniciando en 0. Despues se declara la función auxiliar "sumar_aux", que devuelve un entero.
  El caso base es la lista vacia [] que devulve 0, ya que el acumulador inicia en 0. El paso recursivo se actualiza el acumulador (acc + x) siendo "x" el primer elemento y "xs" el resto de la lista, entonces se realiza la llamada recursiva con el resto de la lista (xs).

La implementacion de buscar y sumar en forma ordinaria:
Para buscar, toma el primer elemento, y despues lo compara con el numero que se quiere comparar "elemento" si son iguales devuve True y en otro caso sigue comparando con el resto de la lista, hasta que sea vacia entonves devolvera False.
Para sumar lista, igual hay un caso base que es cuando esta vacia [] devuelve 0, y el paso recursivo se suma el primer elemento "x" sobre el resto de la lista.
De la forma de buscar con recursion de cola

$$it :: Bool$$

$$(0.00secs, 74, 992bytes)$$

Y con recursion ordinaria:

$$it :: Bool$$

$$(0.00secs, 74, 800bytes)$$

Pues supongo que usa más memoria el de recursión de cola ya que tiene el acumulador y de recursión ordinaria no. Con la suma de la lista con recursion de cola
tambien

$$it :: Int$$

$$(0.00secs, 73, 024bytes)$$

Y con la suma de lista con recursión ordinaria

$$it :: Int$$

$$(0.00secs, 72, 856bytes)$$

Igual creo que usa más memoria el de recursión de cola que el ordinario ya que tiene el acumulador.

2 Funciones

- Filter: Caso base es la lista vacía [], devuelve la lista vacía ya que no hay elementos que filtrar. El caso recursivo, verifica si el predicado "p" es verdadero para "x", si "p x" es True, entonces "x" se incluye en la nueva lista y continua con el resto de la lista "xs". Si "p x" es False el elemento se omite, se llama recursivamente hasta que la lista este vacía [] que es el caso base y termine la recursión.

- Mapear: Caso base es la lista vacía [] ya que no hay elementos que aplicar a la función "f". En el caso recursivo se aplica la función "f" al primer elemento "x", con el operador ":" agrega "f x" al resultado de mapear al resto de la lista "xs".

- Comprehension: Se aplica "f x" para cada "x" en "list". "$x <- list$" es un generador que recorre cada elemento de "list". Entonces para cada elemento "x" en la lista "list", se aplica la función "f" a "x", y los resultados se recopilan en una nueva lista.

3 Tipos,clases y Estructuras de Datos

- buscar_tree: El caso base es cuando el árbol es vació, en caso caso se devuelve falso ya que no importa que elemento se busque, este nunca estará en un árbol vació, en caso de que el árbol sea no vació, se obtendrán todos los elementos del árbol haciendo preorder para usar '$elem$' con el elemento buscado.

- hojas: El caso mas simple es cuando el árbol es vació, así este devuelve cero ya que no se tienen hojas, el siguiente caso es cuando se tiene un nodo que no tiene sub árboles, en este caso de devuelve 1 ya que este nodo cumple la definición de hoja, por ultimo, cuando se tienen sub árboles se suman la cantidad de hojas del árbol izquierdo y la cantidad de hojas del árbol derecho.

- isConnected: El caso base es cuando la gráfica es vacía, en este caso se devuelve verdadero ya que por definición es conexa, para el caso general tendremos en cuenta que si se hace dfs, este dará como resultado todos los vértices a los que pudo llegar según un vértice dado, así se comparará esa cantidad de vértices con la cantidad de vértices en la gráfica, si este numero es igual significa que dfs pudo encontrar todos los vértices y por consecuencia, veremos que la gráfica es conexa.

- isTree: El caso base es cuando la gráfica es vacía, en este caso se devuelve verdadero ya que por definición es un árbol, para el caso general veremos si la gráfica dada es conexa con la función isConnected, también veremos si tiene ciclos con la función hayCiclos esta recibe la gráfica, una lista con los vértices que se han visitado y una lista con los vértices que faltan por ser visitados, esta función toma un vértice en la lista de pendientes, si ese vértice ya fue visitado se concluye que hay un ciclo, en caso contrario el vértice se añade a los vértices visitados y se añaden los vecinos de ese vértice a los pendientes.

- leafSum: El caso base es cuando el árbol es vació, devuelve cero ya que no tiene que sumar el contenido de ningún nodo, cuando el árbol es no vació hacemos preorder del árbol y usamos la función sumar_lista implementada en esta practica para obtener la suma de las hojas del árbol.