Samai García Gonzalez
S. Xiao Fernández Marín
2192_07

PRACTICE 4

Cuestiones relacionadas con el Ejercicio 1:

Sobre algoritmia:

• ¿Qué cambios habría que hacer en el prototipo y pseudocódigo de la función listMergeOrdered para que en vez de 2 listas concatenase N listas? No hace falta que proporciones el pseudocódigo de este nuevo algoritmo, tan solo cuenta la idea general.

First two lists would have to be compared. After this, compare the list that would have resulted from this merge with a new one, like this N times.

• Suponga este otro algoritmo para concatenar dos listas ordenadas de menor a mayor. A diferencia del algoritmo anterior, este nuevo algoritmo sí modifica las listas originales. ¿Cual es su coste temporal (complejidad algorítmica)? Compárelo con el algoritmo que tuvo que programar. Justifique su respuesta.

```
E1:    while l1 is not empty AND l2 is not empty :
E1.1:         if the element at the front of l1 < element at the front of l2 :
E1.1.1:              pop from the front of l1 and push it at the end into lout
E1.2:         else :
E1.2.1:              pop from the front of l2 and push it at the end into lout
E2:    determine the non empty list
E3:    while the "non_empty_list" is not empty :
E2.1:  pop from the front and push it at the end into lout
```

In this algorithm, the cost will be O (m + n), with m and n being the size of lists 1 and 2, respectively.

In the algorithm that we had to program, the cost was O (N^2), where N is the total number of nodes in the lists. Therefore, the proposed algorithm would be more efficient, since the lists it has to go through become smaller and smaller as it empties.

Sobre implementación:

• En una buen implementación, la función listMergeOrdered debería ser completamente independiente del tipo de dato que se almacenase en las listas. Esto es, podría utilizarse sin necesidad de reprogramarla de nuevo si, por ejemplo, los elementos de las listas l1 y l2 fuesen cadenas de caracteres. ¿Por qué no podemos ahora? ¿Qué debería ocurrir para ello?

Now it would not be possible because when we initialize the stack we do it by calling the functions that work with integers (int_copy, int_free ...), so the elements that we insert in the stack must be integers.

In order to store any type of data, we should use function pointers that allow us to use a generic stack.

Samai García Gonzalez
S. Xiao Fernández Marín
2192_07

Cuestiones relacionadas con el Ejercicio 2:

• Teniendo en cuenta que la estructura utilizada para almacenar nodos es una lista enlazada (cada nodo tiene la referencia del siguiente), ¿es más eficiente el uso que hace de la memoria que en el caso del array de nodos?. Justifique su respuesta

Yes, it is more efficient to store nodes in a linked list than in a static array, because a static array will use the same amount of memory every time, independently of the number of nodes, so memory will be wasted. But in the case of the list, it reserves memory dynamically as nodes are inserted, so it allocates only the memory that is needed.

• Si se quiere buscar un nodo concreto, ¿qué estructura sería mejor en cuanto a coste temporal, el array de nodos o la lista de nodos? Justifique su respuesta. Asimismo, conocido el índice i que indica la posición de un nodo dentro de la estructura, ¿qué coste temporal tendría en cada estructura (array o lista de nodos) obtener ese nodo?

It would be better to search for a node in an array than in a list, as in a list it is necessary to traverse it completely to find it, but in an array you can access directly to the position you want.
The temporal cost for obtaining a node in “i” position in an array will be O(1).
In a list it will be O(n), being n the number of nodes in the list.

• Considerando las 4 estructuras planteadas hasta ahora para el TAD Graph (la de la P1, y las 3 descritas en este ejercicio), discute para qué situaciones puede ser cada una de ellas más adecuada, considerando tanto los costes de memoria como los de tiempo (y no sólo en el acceso a los nodos sino también a sus conexiones y, si se desea, otras operaciones no planteadas hasta ahora, como podría ser el borrado de nodos).

The Graph ADT from practice 1 would be useful for searching for a particular node, as it uses an static array. But it is the worst structure in terms of memory waste, because it uses a static memory that not always is used in its whole.

The second structure, the one used in this exercise, uses a list for nodes. This saves memory because it uses only the necessary for the nodes that exist, and it is efficient in terms of searching for connections, but not for searching for a node.

The third structure, uses a static array with the nodes and an adjacency static array list, so it works better in terms of searching for a connection, as you can access to the connections of a node knowing its index position. But it will be the less efficient in terms of temporal cost in functions such as node deletion.

The last structure uses dynamic memory in both arrays, for the nodes and the adjacency connections. This structure is the more efficient in terms of memory, but it is also the more complex one to code and understand.

Cuestiones relacionadas con el Ejercicio 3:

• Analice la salida de los dos programas anteriores teniendo en cuenta la estructura de los árboles generados. Para ello es interesante ejecutar el programa p4_e3b con valores grandes de n (n > 1000). Para evitar repetir muchos números, genérelos entre 0 y INT_MAX (para ello deberá incluir en su programa la biblioteca).

• Proporcione el pseudocódigo de un algoritmo que ordene los elementos de un BST de mayor a menor.

• ¿Cuál sería la complejidad algorítmica en tiempo de ejecución de la función para buscar un elemento en los árboles BST generados en el ejercicio p4_e3_b.c? Justifique brevemente su respuesta.

• Suponiendo que no pudiese modificar la funcionalidad de los TAD Node y Graph, esto es los ficheros .h, proporcione una estructura de datos (EdD) eficiente para el TAD Graph. Se considerará que la EdD es eficiente cuando la complejidad algorítmica (en tiempo de ejecución) de las funciones de la interfaz fuese igual o inferior, en promedio, a O(log n) (excepto, obviamente, para aquellas funciones que impliquen recorrer todo los nodos del grafo como graph_print o graph_free). Justifique su respuesta.