A Collection of Basic Programming Problems for Dynamic Data Structures

Ernest Davis

Linked Lists

Implement the functionalities listed below for any combination of
[singly / doubly linked list] × [without header / with header / with pointers to first and last nodes]
× [destructively / non-destructively] × [as an owned method / with the list passed as argument] ×
[iteratively / recursively]. Assume that the values are ints.

For the non-destructive version, the value returned can be of any of the linked list types; for example,
the input can be a singly linked list without header, and the output can be a doubly-linked list with
a pointer to the last element.

Therefore, each problem below in principle has 168 variants. However, not all combinations make
sense; some have identical solutions; and a lot have nearly identical solutions.

In destructive versions of problems involving two lists, assume that the two lists share no cells in
common.

- Delete all elements larger than 30.
  Example: Input: [25, 45, 62, 21, 19]. Output: [25, 21, 19].

- Reverse the list.
  Example: Input: [25, 45, 62, 21, 19]. Output: [19, 21, 62, 45, 25].

- Delete all but the first occurrence of each element.
  Example: Input: [25, 45, 45, 62, 25, 19, 62]. Output: [25, 45, 62, 19]

- Delete all but the last occurrence of each element.
  Example: Input: [25, 45, 45, 62, 25, 19, 62]. Output: [45, 25, 19, 62]

- Double each element:
  Example: Input: [25, 45, 62, 21, 19]. Output: [25, 25, 45, 45, 62, 62, 21, 21, 19, 19]

- Double the 2nd, triple the 3rd, quadruple the 4th and so on.
  Example: Input: [25, 45, 62, 21]. Output: [25, 45, 45, 62, 62, 62, 21, 21, 21, 21]

- Interleave two lists.
  Example: Input: [25, 45, 62, 21] and [31, 22]. Output: [25, 31, 45, 22, 62, 21]

- Append two lists.
  Example: Input: [25, 45, 62, 21] and [31, 22]. Output: [25, 45, 62, 21, 31, 22].

Lists of lists

Suppose you have two classes: IntList, a linked list of ints of one of the above types; ListOfIntList
a linked list of one of the above types whose values are IntList. Implement the following function-
alities [destructively/non-destructively]. Note: Some of these do not have good recursive implement-
tions.

- Append the sublists.
  Example: Input: The ListOfIntLists [[6, 12, 3], [2, 17] [18, 11, 4, 2]] Output: The IntList
  [6, 12, 3, 2, 17, 18, 11, 4, 2].
- Interleave the sublists. Assume that the sublists all have the same length.
  Example: Input: The ListOfIntLists \([6, 12, 3], [2, 17, 18], [11, 4, 2]\) Output: The IntList [6, 2, 11, 12, 17, 4, 3, 18, 2].

- Transpose the sublists; that is, if you think of the sublists as making a rectangle, turn the rows into columns and the columns into rows. Assume that the sublists all have the same length.
  Example: Input: The ListOfIntLists \([6, 12, 3], [2, 17, 18], [11, 4, 2]\) Output: The ListOfIntList [6, 2, 11], [12, 17, 4], [3, 18, 2].

- Partition a list into sublists of a specified length.
  Example: Input: The IntList [6, 12, 3, 2, 17, 18, 11, 4, 2, 9] and the integer 4. Output: the ListOfIntLists [6, 12, 3, 2], [17, 18, 11, 4], [2, 9].

Trees

Consider trees where the nodes are labelled with two integer fields. The first is the value, which is some arbitrary input value. The second is the result, which is the result of some computation over the values in the tree. Trees can be implemented, either as binary trees with a left child and a right child, or with arbitrary branching, using firstChild and nextSibling.

All of the following should be computed using recursion down the tree.

- Set the result field at each node to be the number of nodes in the subtree.
  Example (with arbitrary branching)

  Input
  \[
  [20| __|]---->[15| __|]---->[12| __|]---->[10| __|]
  |
  |   |
  |   |->[23| __|]
  |
  |   |->[7| __|]
  |
  |->[6| __|]---->[17| __|]
  |
  |   |->[2| __|]

  Output
  \[
  [20| 9|]---->[15| 5|]---->[12| 2|]---->[10| 1|]
  |
  |   |
  |   |->[23| 1|]
  |
  |   |->[7| 1|]
  |
  |->[6| 3|]---->[17| 1|]
  |
  |   |->[2| 1|]
• Set the result field at each node to be the height of the subtree.
  Example

Input
```
[20|__]-->[15|__]-->[12|__]-->[10|__]
  |
  |->[23|__]
  |
  |->[ 7|__]
  |
  |->[ 6|__]-->[17|__]
  |
  |->[ 2|__]
```

Output
```
[20| 3]-->[15| 2]-->[12| 1]-->[10| 0]
  |
  |->[23| 0]
  |
  |->[ 7| 0]
  |
  |->[ 6| 1]-->[17| 0]
  |
  |->[ 2| 0]
```

• Set the result field at each node to be the number of leaves in the subtree.
  Example

Input
```
[20|__]-->[15|__]-->[12|__]-->[10|__]
  |
  |->[23|__]
  |
  |->[ 7|__]
  |
  |->[ 6|__]-->[17|__]
  |
  |->[ 2|__]
```

Output
```
[20| 5]-->[15| 3]-->[12| 1]-->[10| 1]
  |
  |->[23| 1]
  |
  |->[ 7| 1]
  |
  |->[ 6| 2]-->[17| 1]
  |
  |->[ 2| 1]
```
- Set the result field at each node to be maximum value in the subtree.
  
  Example

  Input
  
  \[ 20|_|---\rightarrow 15|_|---\rightarrow 12|_|---\rightarrow 10|_ \]
  | |
  | |---\rightarrow 23|_| 
  | |
  | |---\rightarrow 7|_| 
  | |
  \rightarrow 6|_|---\rightarrow 17|_ 
  | |
  \rightarrow 2|_

  Output
  
  \[ 20\_|23\_---\rightarrow 15\_|23\_---\rightarrow 12\_|12\_---\rightarrow 10\_|10 \]
  | |
  | |---\rightarrow 23|23 | |
  | |
  | |---\rightarrow 7|7 | |
  | |
  \rightarrow 6|17\_---\rightarrow 17|17 
  | |
  \rightarrow 2|2 |

  Trees to Lists

  Given a tree, in either of the above implementation, generate a [preorder / postorder / inorder (for binary trees) / breadth-first order] in any of the list implementations.