Data Structures and Algorithms in Python

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter

7 Linked Lists

Hints

Reinforcement

R-7.1) It is okay to have an algorithm running in linear time.

R-7.2) This concatenation operation need not search all of L and M.

R-7.3) Consider passing a node as a parameter.

R-7.4) Performing the swap for a singly linked list will take longer than for a doubly linked list.

R-7.5) You need to keep track of where you start or your method will have an infinite loop.

R-7.6) Your only need to go around one of the lists once.

R-7.7) You must adjust links so that the first node is moved to the end of the list.

R-7.8) Consider a combined search from both ends. Also, recall that a link hop is an assignment of the form ”p = p._next” or ”p = p._prev”.

R-7.9) Splice the end of L into the beginning of M.

R-7.10) Is there a scenario in which these substitions fail?

R-7.11) Keep track of the maximum thus far while walking the list

R-7.12) Within a method of the class, you may access nonpublic members

R-7.13) Start looking at the beginning of the list.

R-7.14) Consider parameterizing the method with a node of the list.

R-7.15) Model your solution on the original implementation with appropriate symmetry.

R-7.16) Be careful when working with an empty list.

R-7.17) Be careful to repair the list in the neighborhood abandoned by the moved node.

R-7.18) Implement the move-to-front using a pencil and eraser. Better yet, write the six letters on separate pieces of paper and simulate the actions physically.

R-7.19) Consider the two extreme cases of how we could distribute m accesses across n elements.

R-7.20) The first should be last, both physically and in terms of how long ago it has been accessed.

R-7.21) For this lower bound, assume that when an element is accessed we search for it by traversing the list starting at the front.

R-7.22) You can either clear the underlying list or start over with a new list.

R-7.23) You will need to adjust instances of the nested _Item class.

Creativity

C-7.24) Admittedly, it is not clear that there is any advantage to the sentinel for this purpose.

C-7.25) You should be able to avoid the conditional within enqueue.

C-7.26) Make sure to leave the head and tail members of both lists with appropriate values.

C-7.27) View the chain of nodes following the head node as themselves forming another list.

C-7.28) Recur on the first $n-1$ positions.

C-7.29) Consider changing the orientation of links while making a single pass through the list.

C-7.30) Think carefully about the orientation of the linked list.

C-7.31) Consider using an abstraction that is a subset of the positional list ADT.

C-7.32) You should replace the first() and last() methods with a method abstracting the cursor.

C-7.33) You will need to carefully switch next and prev pointers and properly manage the sentinels.

C-7.34) Watch out for the special case when p and q are neighbors.

C-7.35) See Section 2.3.4 for discussion of iterators.

C-7.36) Watch out for special cases when the length is one or less.

C-7.37) To get you started, consider if the smallest and largest values add to V. If not, you should be able to eliminate one of the two as unnecessary.

C-7.38) It would be helpful to implement a swap subroutine.

C-7.39) Carefully map the public methods of the queue interface to the concrete behaviors of the PositionalList class.

C-7.40) Note well that there may be fewer than n elements included in the most recent n accesses, due to duplication.

C-7.41) Be sure to handle the case where every pair (x,y) in A and every pair (y,z) in B have the same y value.

C-7.42) You should keep track of the number of game entries explicitly.

C-7.43) Convert the two parts to two separate lists as sublists.

Projects

P-7.44) Use a position instance variable to keep track of the cursor location.

P-7.45) There is a trade-off between insertion and searching depending on whether the entries in L are sorted.

P-7.46) It is okay to be inefficient in this case.

P-7.47) Keep all cards in a single list, and use four positions to demark the beginning of the respective suits.