Project 2 Linked Lists

CECS 274 Data Structures Fall 2020

Deadline: 11:55pm, October 11st (Sunday), 2020

Note: the sample codes are given in Python format, you can flexibly choose other languages like Java, etc.

1. (10pt) Describe a good algorithm for concatenating two sorted singly linked lists L and M, with references to the first node of each list, into a single list LM that contains all the nodes of L and M, and all nodes are sorted (increasing order). Name your code to a file as *1_1_merge_lists.py*. (You can either use the given sample code or design by yourself)

   Sample Outputs*: L {3, 6, 9, 14, 17}, M {2, 8, 15, 19, 22}, after merge, LM {2, 3, 6, 9, 8, 14, 15, 17, 19, 22}*

2. (10pt) Design a circular linked list, with the operations of add operation, print operation to print the list from the head, and count operation to count the number of nodes. Name your code file to *1_2_circular_lists.py* (or .java, .etc). (You can either use the given sample code or design by yourself)

   Sample Outputs*: add the values {4, 5, 7, 8, 12, 56, 85, 41} and print the list {4, 5, 7, 8, 12, 56, 85, 41} with count number is 8.*

3. (15pt) Implement a FIFO queue with the linked list implementation. Use the singly linked list complete the enqueue and dequeue operations. Next, Figure 2.1 provides a *rotate( )* method that has semantics equivalent to *Q.enqueue(Q.dequeue())*, for a nonempty queue. Implement such a method for the *LinkedQueue* class (in 1_3_queue_lists.py) without the creation of any new nodes.
   You can use the given sample "*1_3_queue_lists.py*" in *sample_project_2* folder and test with the test code. (you can also write your own code with same name).

```
def rotate(self):
  """Rotate front element to the back of the queue."""
  if self._size > 0:
    self._tail = self._tail._next              # old head becomes new tail
```

Figure 2.1

4.(15pt) Design the doubly linked list. Complete the *get, set, remove*, and *add* operations. Add a new method *is_palindrome()* that returns true if the list is a palindrome, i.e., the element at position *i* is equal to the element at position $n-i-1$ for all $i \in \{0,...n-1\}$. Your code should run in O(n) time. You can use the given sample "*1_4_doubly_lists.py*" in *sample_project_2* folder and test with the test code. (you can also write your own code with the same name).

Hint: Traverse the list forward and backward simultaneously. The menu should give an option to test palindromes. It returns true if it is an empty word; a palindrome word with one letter 0r even length, e.g., "a", "hawwah", "bananab"; a word in not palindrome

5.(15pt) Design the PositionalList ADT. You can use the sample code of PositionalList from our slides. Implement a new function *parisum()* that accepts a PositionalList *L* of *n* integers sorted in nondecreasing order, and another value V, and determines in O(n) time if there are two elements of *L* that sum precisely to V. The function should return a pair of positions of such elements, if found, or None otherwise. Please use the name "*1_5_positional_lists.py*" in the submission (or use .java, etc.).

Sample output: Add a few data with position into the list L {2a, 4b, 7c, 9d, 11e, 13f, 17g, 22h}, define V = 15, return {b, e} {a, f}; define V = 36, return None.

Submission Requirements

You need to strictly follow the instructions listed below:

1) Submit a .zip/.rar file that contains all files.
2) The submission should include your **source code**. Do not submit your binary code.
3) Your code must **be able to compile**; otherwise, you will receive a grade of zero.
4) Your code should not produce anything else other than the required information in the output file.
6) If you code is partially completed, also explain in the begin comment of specific submission what has been completed and the status of the missing parts.
7) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Submission follows the right formats	10 %
Have a README file shows how to compile and test your submission	15 %
Submitted code has proper comments to show the design details	15 %
Code can be compiled and shows right outputs	60 %

Rubrics for code outputs

	Level 4 (100%)	Level 3 (70%)	Level 2 (40%)	Level 1 (20%)
Code outputs	It is always correct without crashes	It is always correct and eventually it crashes	It is not always correct and eventually it crashes	It is not correct or incomplete