CPE202 Winter 2020

Lab 2

Lists

Objective

In this lab, we are going to learn to implement two kinds of list: array based list (ArrayList), and linked list (LinkedList).
ArrayList uses an array (Python list) as a data structure to store items. In order to learn the pros and cons of array based list such as Python list, you are prohibited to use Python's built-in list functions such as append, remove, pop, insert, index, count, and so on in this lab. Instead, you are going to implement your own functions for ArrayList.
LinkedList uses our custom data structure Node to store an item. Objects of Node are linked to each other unidirectionally by the next pointer. You are responsible for writing functions to create and maintain LinkedLists.
Both ArrayList and LinkedList have pros and cons. As you implement these two kinds of list, identify their pros and cons.

Array List

1. Download a starter file, array_list.py from Polylearn.
2. Create a class definition for ArrayList class, which realizes a list. The class has following fields (attributes):
    - arr (list) : an array for storing items
    - capacity (int): an integer number indicating the maximum number of items the array can store.
    - num_items (int): an integer number indicating the current number of items stored in the array.

    Write a constructor, __init__ method, for ArrayList. The constructor shall initialize all the fields of the class:
    - The capacity shall be initialized to 2.
    - The num_items shall be initialized to 0.
    - The arr shall be initialized to [None] * self.capacity.

3. Write headers, signatures, and purposes for the following standalone functions:
    - **enlarge** function: enlarge(lst), which takes an object of ArrayList lst and returns an object of ArrayList whose arr's capacity is double the original capacity.
    - **shrink** function: shrink(lst), which takes an object of ArrayList lst and returns an object of ArrayList whose arr's capacity is half the original capacity.

- **insert** function: insert(lst, val, idx), which takes an object of ArrayList lst, an integer val, and an integer idx, and insert the integer val to the arr of the ArrayList object at the index indicated by the integer idx, and returns the ArrayList object. The function shall enlarge the ArrayList by calling the enlarge function when the ArrayList is full (num_items == capacity).
- **get** function: get(lst, idx), which takes an object of ArrayList lst, and an integer idx, and get an item stored at the index indicated by the integer idx, and returns the item (integer). It raises IndexError if the index is out of bound (>= num_items).
- **search** function: search(lst, val), which takes an object of ArrayList, and an integer, and returns the index where the integer is stored in the arr of the ArrayList object. It returns None if the integer is not found.
- **contains** function: contains(lst, val), which takes an object of ArrayList lst and an integer val as arguments, and searches for the value in the list, and returns True if the value is found or False if not. This function shall call the search function.
- **remove** function: remove(lst, val), which takes an object of ArrayList lst, and an integer val, and removes the integer val from the lst by shifting items on the right by one to the left. If the item to be removed is the last item in the ArrayList (index == num_items - 1), simply decrement the value of num_items by 1 (num_items -= 1). The function returns the ArrayList object lst. The function shall shrink the ArrayList by calling the shrink function when the ArrayList is a quarter full (4 * num_items <= capacity), and the capacity is greater than 2 (capacity > 2).
- **pop** function: pop(lst, idx), which takes an object of ArrayList lst and an integer idx as arguments, and removes the item at the index idx in the list, and returns the list and the removed item's value. The function shall shrink the ArrayList by calling the shrink function when the ArrayList is a quarter full (4 * num_items <= capacity), and the capacity is greater than 2 (capacity > 2).
- **size** function: size(lst), which take an object of ArrayList object lst, and returns the number of items stored in the ArrayList object (returns num_items).

4. Write tests for the functions by creating array_list_tests.py.

5. Implement the functions.

6. Test the functions. Fix problems, if your program does not pass all the tests, until it passes all the tests.

Linked List

1. Download a starter file, linked_list.py from Polylearn.
2. Create a class definition for Node class, which realizes a list. The class has following fields (attributes):
    - value (str): an array for storing items
    - next (Node): the next item in the list.

    Write a constructor, __init__, for Node. The constructor shall initialize all the fields of the class:
    - The value shall be initialized to an argument passed to the constructor.
    - The next shall be initialized to an argument passed to the constructor or None if the argument has not passed.

3. Write headers, signatures, and purposes for the following standalone functions:
    - **insert** function: insert(lst, val, pos), which takes an object of Node (LinkedList) lst, an integer val, and an integer pos as arguments, and inserts the integer at the position pos in the linked list, and returns the head of the linked list (Node object). If the pos is equal to num_items, it appends the val at the end of the list. It raises IndexError: when the position is out of bound (> num_items).
    - **get** function: get(lst, pos), which takes an object of Node (LinkedList) lst, and an integer pos, and get an item stored at the position indicated by the integer pos, and returns the item (integer). It raises IndexError if the position is out of bound (>= num_items).
    - **search** function: search(lst, val), which takes an object of Node (LinkedList), and an integer value val, and returns the position where the integer is stored in the list. It returns None if the integer is not found.
    - **contains** function: contains(lst, val), which takes an object of Node (LinkedList) lst and an integer val as arguments, and searches for the integer recursively in the LinkedList, and returns True if the integer is found or False if not. This function shall call the search function.
    - **remove** function: remove(lst, val), which takes an object of Node (LinkedList) lst and an integer val as arguments, and searches for the integer recursively in the linked list, and returns the head of the linked list with the integer removed.
    - **pop** function: pop(lst, pos), which takes an object of Node (LinkedList) lst and an integer pos as arguments, and removes the item at the position pos in the list, and returns the head of the LinkedList and the removed item's value. It raises IndexError: when the position is out of bound (>= num_items).
    - **size** function: size(lst), which take an object of Node (LinkedList) object lst, and returns the number of items stored in the LinkedList object (returns num_items).

4. Write tests for the functions by creating linked_list_tests.py.

5. Implement the functions. You are free to add helper functions, especially for recursive functions, which take extra arguments. In that case, your helper functions should become recursive functions.
    a. Example:

```
def search(lst, val):
    """searches for a specified value in a given list.
    Args:
        lst (Node): an object of Node (LinkedList)
        val (str): a value to search for
    Returns:
        int: the position where the value is stored in the list. It returns None if the value is not found.
    """
    return search_helper(lst, val, 0)

def search_helper(lst, val, pos):
    """A helper function to search for a specified value in a given list recursively.
    Args:
        lst (Node): an object of Node (LinkedList)
        val (str): a value to search for
        pos (int): the current position in the list
    Returns:
        int: the position where the value is stored in the list. It returns None if the value is not found.
    """
    if lst is None:
        return None
    if lst.val == val:
        return pos
    return search_helper(lst.next, val, pos + 1)
```

6. Test the functions. Fix problems, if your program does not pass all the tests, until it passes all the tests.

Submission

Zip four files, array_list.py, array_list_tests.py, linked_list.py, and linked_list_tests.py into one zip file and submit it to the grader to get it graded. Then submit the same file to polylearn. **Do not include a directory structure in your zip file.**