Carleton University
Department of Systems and Computer Engineering
SYSC 2100 — Algorithms and Data Structures — Winter 2021

Lab 5 - ADT Unordered List

Submitting Lab Work for Grading

Remember, you don't have to finish the lab by the end of your lab period. For this lab, the deadline for all sections to submit solutions to cuLearn for grading is 11:55 pm (Ottawa time) Sunday, February 14. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

Prerequisite Reading

- If you haven't done so already, read *Problem Solving with Algorithms and Data Structures using Python, Third Edition,* Sections 4.19 *Lists*, 4.20 *The Unordered List Abstract Data Type*, and 4.21 *Implementing an Unordered List: Linked Lists* to the end of Section 4.21.2.

Getting Started

Download unorderedlist.py from the *Lab Materials* section of the main cuLearn course page. Classes `Node` and `UnorderedList` are based on the classes presented in the Sections 4.21.1 and 4.21.2 in the textbook.

Class `UnorderedList` uses a singly-linked list as the ADT's underlying data structure. The following changes were made to this class:

- Instance variable `head` has been renamed `_head`, to denote that it is a "private" attribute of `UnorderedList` objects.
- Type annotations were added to the method headers. Note: if a parameter is not annotated, we assume that its type is the enclosing class. For example, parameter `self` is not annotated, so we assume that its type is `UnorderedList`. An `UnorderedList` can store any type of object that is comparable; however, for this lab, all the items will be values of type `int`.
- Docstrings were added to all the methods. Each doctring has a concise summary of what the method does and examples of tests that we can execute in the Python shell.
- `__init__` in the textbook's class always creates an empty linked list:

```
def __init__(self):
    self.head = None
```

This method has been changed to take an optional argument, which must be an *iterable*. (An iterable is an object that is capable of returning its members one at a time. Instances of Python's `list` and `tuple` types and `range` objects, are three examples of iterables.) The contents of the iterable are used to initialize the `UnorderedList` object. See the method's docstring for an example.

- `__str__` has been defined.
- `__iter__` has been defined. This method returns an object that can be used to iterate over an `UnorderedList`. See the method's docstring for an example. Note: `__iter__` is implemented as a *generator function*. You aren't expected to know how to write generators, but if you're interested in learning about them, check the tutorial and language reference documents at python.org.
- `size` has been renamed `__len__`, to allow Python's built-in `len` function to work with `UnorderedList` objects.
- `search` has been renamed `__contains__`, to allow Python's `in` operator to work with `UnorderedList` objects.
- `remove` has been rewritten: the code that unlinks a node from the linked list has been moved inside the `while` loop.
- "Stub" implementations have been provided for methods `__repr__`, `count`, `append`, `index`, `pop and insert`. If you call any of these methods on an `UnorderedList` object, Python will throw a `NotImplementedError` exception.

Exercise 1: Change `__init__` to create an instance variable named `_size`, which keeps track of the number of items in the `UnorderedList`.

- Change `__len__` to use this instance variable. This will let you improve the method's complexity from $O(n)$ to $O(1)$.
- Change `add` to update this instance variable. This method's complexity should remain $O(1)$.
- Change `remove` to update this instance variable. This method's complexity should remain $O(n)$ worst case.

Class `UnorderedList` cannot contain instance variables other than `_head` and `_size`.

Use the shell (or write a short script) to test the four methods.

Exercise 2: Read the docstring for `__repr__`. Notice that `__repr__` will return an expression that would create an instance of `UnorderedList` that is identical to the object on which `__repr__` is called. Replace the `raise` statement with a correct implementation of the method.

Your method should be $O(n)$. Hint: feel free to "borrow" the code from `__str__`, but note that there will be some important differences between the two methods. Test `__repr__`.

Exercise 3: Read the docstring for `count`. Replace the `raise` statement with a correct implementation of the method. Your method should be $O(n)$. Hint: this method is very similar to `__contains__`, in that both methods traverse the linked list, but neither method adds or removes nodes. Test `count`.

Exercise 4: Read the docstring for `append`. Replace the `raise` statement with a correct implementation of the method. Your method should be $O(n)$. Hint: you should be able to reuse much of your solution to the append-to-end-of-linked-list exercise from one of the lectures. Test `append`.

Exercise 5: Read the docstring for `index`. Replace the `raise` statement with a correct implementation of the method. Your method should be $O(n)$ worst case. Nodes are numbered sequentially starting from 0, so the first node is considered to be at index 0, the second node is at index 1, and so on. Test `index`.

Exercise 6: Review the code for `remove`. Make sure you understand how the algorithm handles these three cases:

- the target item is not in the `UnorderedList`.
- the target item is in the first (head) node in the linked list.
- the target item is in one of the other nodes in the linked list.

Read the docstring for `pop`. Replace the `raise` statement with a correct implementation of the method. Your method should be $O(n)$ worst case. Hint: think carefully about the different cases the method must handle. Consider drawing "before and after" diagrams of the linked list for each case, and use these diagrams as a guide while you develop the code. Test `pop`.

Exercise 7: Read the docstring for `insert`. Replace the `raise` statement with a correct implementation of the method. Your method should be $O(n)$ worst case. Hint: think carefully about the different cases the method must handle. Consider drawing "before and after" diagrams of the linked list for each case, and use these diagrams as a guide while you develop the code. Test `insert`.

Wrap Up

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline. The submission deadlines for this lab are:

Lab Section	Lab Date/Time	Submission Deadline (Ottawa Time)
L5	Tuesday, 11:35 - 13:25	Sunday, Feb. 14, 23:55
L2	Thursday, 9:35 - 11:25	Sunday, Feb. 14, 23:55
L4	Thursday, 12:35 - 14:25	Sunday, Feb. 14, 23:55
L3	Friday, 9:35 - 11:25	Sunday, Feb. 14, 23:55
L1	Friday, 14:35 - 16:25	Sunday, Feb. 14, 23:55

To submit your lab work, go to the cuLearn page **for your lab section** (not the main course page). Submit unorderedlist.py. Ensure you submit the version of the file that contains your solutions, and not the unmodified file you downloaded from cuLearn! You are permitted to make changes to your solutions and resubmit the file as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn.

Last edited: Feb. 7, 2021