University of Saskatchewan
Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 5

Node chains

Date Due: June 21, 2019, 11pm **Total Marks: 50**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Version History

- **06/14/2019**: released to students

Question 0 (5 points):

Purpose: To force the use of Version Control in Assignment 5

Degree of Difficulty: Easy

You are expected to practice using Version Control for Assignment 5. This is a tool that you need to be required to use, even when you don't need it, so that when you do need it, you are already familiar with it. Do the following steps.

1. Create a new PyCharm project for Assignment 5.
2. Use `Enable Version Control Integration...` to initialize Git for your project.
3. Download the Python and text files provided for you with the Assignment, and add them to your project.
4. Before you do any coding or start any other questions, make an initial commit.
5. As you work on each question, use Version Control frequently at various times when you have implemented an initial design, fixed a bug, completed a question, or want to try something different. Make your most professional attempt to use the software appropriately.
6. When you are finished your assignment, open the terminal in your Assignment 5 project folder, and enter the command: `git --no-pager log` (double dash before the word 'no'). The easiest way to do this is to use PyCharm, locate PyCharm's `Terminal` panel at the bottom of the PyCharm window, and type your command-line work there.
   Note: You might have trouble with this if you are using Windows. Hopefully you are using the department's network filesystem to store your files. If so, you can log into a non-Windows computer (Linux or Mac) and do this. Just open a command-line, `cd` to your A3 folder, and run `git --no-pager log` there. If you did all your work in this folder, git will be able to see it even if you did your work on Windows. Git's information is out of sight, but in your folder.
   Note: If you are working at home on Windows, Google for how to make git available on your command-line window. You basically have to tell the command-line app where the git app is.

You may need to work in the lab for this; Git is installed there. Not having Git installed is not really an excuse. It's like driving a car without wearing a seatbelt. It's not an excuse to say "My car doesn't have a seatbelt."

What to Hand In

After completing and submitting your work for Questions 1-3, open a command-line window in your Assignment 5 project folder. Run the following command in the terminal: `git --no-pager log` (double dash before the word 'no'). Git will output the full contents of your interactions with Git in the console. Copy/-paste this into a text file named `a5-git.log`.

If you are working on several different computers, you may copy/paste output from all of them, and submit them as a single file. It's not the way to use git, but it is the way students work on assignments.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: The log file shows that you used Git as part of your work for Assignment 5. For full marks, your log file contains
  - Meaningful commit messages.
  - At least two commits per question for a total of at least 6 commits. And frankly, if you only have 6 commits, you're pretending.

IMPORTANT! Addendum on stepping through the node chain

Always make a copy of the reference stored in your node-chain variable, for example *anchor*, to another variable, for example *anode*. We will use *anode* to step through the chain, one node at a time, using a while loop without losing the reference of *anchor* which refers to the first node in the chain. We must use a while loop for our nodes, because Python has no idea how to step through our chain using a for-loop. The condition for the while loop checks whether we've reached the end of the chain. There are no more nodes to look at if the variable *anode* == None.

```
anchor = node.create(2, node.create(5))
anode = anchor
while anode != None:
    print(node.get_data(anode))
    anode = node.get_next(anode)
print(anchor)
```

Notice the last line especially. This is a key technique! $anode = node.get\_next(anode)$ replaces the reference stored in anode with the reference stored in anode's next field. Visually, this is stepping from one node to another by following the arrow out of the box labelled next that we saw in class.

It's important to point out that we **always** use a temporary variable to step through a node-chain. If we step through the node-chain by changing the anchor point, and we reach the end of the node-chain, we have effectively unhooked the whole node-chain from the anchor. It's really important to understand that **without** any anchor point, we have **no** way to access the node-chain when we're done, and it is absolutely, totally, and in all other ways, unaccessible. Try to run the previous code using *anode* and the code bellow without *anode*. You are going to see that *anchor* will be None in the second code and there is no way to access the original node-chain anymore.

```
anchor = node.create(2, node.create(5))
while anchor != None:
    print(node.get_data(anchor))
    anchor = node.get_next(anchor)
print(anchor)
```

Question 1 (5 points):

Purpose: To practice debugging a function that works with node chains created using the Node ADT.

Degree of Difficulty: Easy

On Moodle, you will find a *starter file* called `a5q1.py`. It has a broken implementation of the function `to_string()`, which is a function used by the rest of the assignment.

You will also find a test script named `a5q1_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

Debug and fix the function `to_string()`. The error in the function is pretty typical of novice errors with this kind of programming task.

The interface for the function is:

```
def to_string(node_chain):
    """
    Purpose:
        Create a string representation of the node chain.  E.g.,
        [ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]
    Pre-conditions:
        :param node_chain:  A node-chain, possibly empty
    Post_conditions:
        None
    Return: A string representation of the nodes.
    """
```

Note carefully that the function does not do any console output. It should return a string that represents the node chain.

Here's how it might be used:

```
empty_chain = None
chain = node.create(1, node.create(2, node.create(3)))

print('empty_chain --->', to_string(empty_chain))
print('chain --------->', to_string(chain))
```

Here's what the above code is supposed to do when the function is working:

```
empty_chain ---> EMPTY
chain ---------> [ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]
```

Notice that the string makes use of the characters `'[ | *-]-->'` to reflect the chain of references. The function also uses the character `'/'` to abbreviate the value `None` that indicates the end of a chain. Note especially that the empty chain is represented by the string `'EMPTY'`.

What to Hand In

A file named `a5q1.py` with the corrected definition of the function. Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: The function `to_string()` works correctly

Question 2 (20 points):

Purpose: To practice working with node chains created using the Node ADT.

Degree of Difficulty: Easy.

In this question you'll write three functions for node-chains that are a little more challenging. On Moodle, you can find a *starter file* called `a5q2.py`, with all the functions and doc-strings in place, and your job is to write the bodies of the functions. You will also find a test script named `a5q2_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

(a) (5 points) Implement the function `count_chain()`. The interface for the function is:

```
def count_chain(node_chain):
    """
    Purpose:
        Counts the number of nodes in the node chain.
    Pre-conditions:
        :param node_chain: a node chain, possibly empty
    Return:
        :return: The number of nodes in the node chain.
    """
```

Note carefully that the function is not to do any console output.
A demonstration of the application of the function is as follows:

```
empty_chain = None
chain = node.create(1, node.create(2, node.create(3)))

print('empty chain has', count_chain(empty_chain), 'elements')
print('chain has', count_chain(chain), 'elements')
```

The output from the demonstration is as follows:

```
empty chain has 0 elements
chain has 3 elements
```

(b) (5 points) Implement the function `delete_front_nodes()`. The interface for the function is:

```
def delete_front_nodes(node_chain, n):
    """
    Purpose:
        Deletes the first n nodes from the front of the node chain.
    Pre-Conditions:
        :param node_chain: a node-chain, possibly empty
        :param n: integer, how many nodes that should be removed off
        the front of the node chain
    Post-conditions:
        The node-chain is changed, by removing the first n nodes.
        If n>length of node_chain, node_chain is set to be empty (None)
    Return:
        :return: The resulting node chain, which may now be empty (None)
    """
```

A demonstration of the application of the function is as follows:

```
chain = node.create(1, node.create(2, node.create(3)))
smaller_chain = delete_front_nodes(chain, 2)

print("before:", to_string(chain))
print("after:", to_string(smaller_chain))
```

The output from the demonstration is as follows:

```
before: [ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]
after: [ 3 | / ]
```

We can see that the first two nodes were deleted from the chain.

(c) (5 points) Implement the function `replace_last()`. The interface for the function is:

```
def replace_last(node_chain, target_val, replacement_val):
    """
    Purpose:
        Replaces the last occurrence of target data value with the
        new_value. The chain should at most have 1 data value changed.
    Pre-conditions:
        :param node_chain: a node chain, possibly None
        :param target_val: the target data value we are searching to
        replace the last instance of
        :param replacement_val: the data value to replace the
        target_val that we found
    Post-conditions:
        The node-chain is changed, by replacing the last occurrence of
        target_val. If target_val is not present, then the node_chain
        returns unaltered.
    Return:
        :return: The altered node chain where any data occurrences of
        target_val has been replaced with replacement_val.
    """
```

This function affects the data values stored in the node-chain, but should not change any of the nodes in the chain. If the target does not appear in the node-chain, the node-chain is not modified at all.

A demonstration of the application of the function is as follows:

```
chain = node.create(1,
         node.create(2,
         node.create(3,
         node.create(2,
         node.create(5)))))
print('before:', to_string(chain))
altered_chain = replace_last(chain, 2, 4)
print('after:', to_string(altered_chain))
```

The output from the demonstration is as follows:

```
before: [ 1 | *-]-->[ 2 | *-]-->[ 3 | *-]-->[ 2 | *-]-->[ 5 | / ]
after: [ 1 | *-]-->[ 2 | *-]-->[ 3 | *-]-->[ 4 | *-]-->[ 5 | / ]
```

(d) (5 points) Before you submit your work, review it, and edit it for programming style. Make sure your variables are named well, and that you have appropriate (not excessive) internal documentation (do not change the doc-string, which we have given you).

What to Hand In

A file named `a5q2.py` with the definitions of the three functions. Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: Your function `count_chain()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to return the number of nodes in the chain correctly.
  - Works on node-chains of any length.
- 5 marks: Your function `delete_front_nodes()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to delete the front n nodes, which may result in an empty node-chain.
  - Works on node-chains of any length.
- 5 marks: Your function `replace_last()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to replace the last target value appropriately, while leaving the node-chain intact.
  - Works on node-chains of any length.
- 5 marks: Overall, you used good programming style, including:
  - Good variable names
  - Appropriate internal comments (outside of the given doc-strings)

Question 3 (20 points):

Purpose: To practice working with node chains created using the Node ADT to implement slightly harder functionality.

Degree of Difficulty: Moderate to Tricky.

In this question you'll write some functions for node-chains that are even more challenging. On Moodle, you can find a *starter file* called `a5q3.py`, with all the functions and doc-strings in place, and your job is to write the bodies of the functions. You will also find a test script named `a5q3_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

(a) (5 points) Implement the function `contains_duplicates()`. The interface for the function is:

```
def contains_duplicates(node_chain):
    """
    Purpose:
        Returns whether or not the given node_chain contains one
        or more duplicate data values.
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty
    Return:
        :return: True if duplicate data value(s) were found,
        False otherwise
    """
```

For this question, you are allowed to use a List to store data values we've seen while iterating through the chain. We should simply return True if we find ANY duplicate values, or False otherwise.

A demonstration of the application of the function is as follows:

```
chain = node.create(1,
         node.create(2,
         node.create(3,
         node.create(4,
         node.create(5)))))
print('Duplicates?', contains_duplicates(chain))
altered_chain = replace_last(chain, 5, 1)
print('Duplicates?', contains_duplicates(altered_chain))
```

The output from the demonstration is as follows:

```
Duplicates? False
Duplicates? True
```

(b) (5 points) Implement the function `reverse_chain()`. The interface for the function is:

```
def reverse_chain(node_chain):
    """
    Purpose:
        Completely reverses the order of the given node_chain.
    Pre-conditions:
        :param node_chain: a node chain, possibly empty
    Post-conditions:
        The front of the node_chain is altered to be the back,
        with all nodes now pointing next the opposite direction.
    Return:
        :return: The node chain with its order reversed
    """
```

Note that the function should return the original node-chain if the node chain is of size 1 or 0. A demonstration of the application of the function is as follows:

```
chain = node.create(1, node.create('two', node.create(3)))
print('before:', to_string(chain))
reversed_chain = reverse_chain(chain)
print('after:', to_string(reversed_chain))
```

The output from the demonstration is as follows:

```
before: [ 1 | *-]-->[ two | *-]-->[ 3 | / ]
after: [ 3 | *-]-->[ two | *-]-->[ 1 | / ]
```

(c) (5 points) Implement the function `insert_value_sorted()`. The interface for the function is:

```
def insert_value_sorted(node_chain, number_value):
    """
    Purpose:
        Insert the given number_value into the node-chain so that
        it is inserted into proper ascending order.
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty,
        containing only numbers
        :param number_value: a numerical value to be inserted
        Assumption:  node_chain only contains numbers,
        and is already in ascending order
    Post-condition:
        The node-chain is modified to include a new node in
        the proper spot to make the node_chain be in ascending order
    Return
        :return: the node-chain with the new value in it.
        Ex: Insert 3 into: 1->2->5. Becomes 1->2->->3->5
    """
```

This one is tricky, as there are a few specials cases: inserting at the front of the chain, the back, and the middle.

A demonstration of the application of the function is as follows:

```
chain = node.create(1, node.create(2, node.create(4, node.create(5))))
print('before:', to_string(chain))
chain = insert_value_sorted(chain, 3)
print('after:',to_string(chain))
```

The output from the demonstration is as follows:

```
before: [ 1 | *-]-->[ 2 | *-]-->[ 4 | *-]-->[ 5 | / ]
after: [ 1 | *-]-->[ 2 | *-]-->[ 3 | *-]-->[ 4 | *-]-->[ 5 | / ]
```

(d) (5 points) Before you submit your work, review it, and edit it for programming style. Make sure your variables are named well, and that you have appropriate (not excessive) internal documentation (do not change the doc-string, which we have given you).

What to Hand In

A file named `a5q3.py` with the definition of your functions. Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: Your function `contains_duplicates()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT (and maybe a list) to check if the node-chain contains any duplicate data values.
  - Works on node-chains of any length.
- 5 marks: Your function `reverse_chain()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to completely reverse the order of a given node-chain.
  - Works on node-chains of any length.
- 5 marks: Your function `insert_value_sorted()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to insert a new node with the given value so that the node-chain remains sorted in ascending order (node-chain only contains numbers).
  - Works on node-chains of any length.
- 5 marks: Overall, you used good programming style, including:
  - Good variable names
  - Appropriate internal comments (outside of the given doc-strings)