Computer Science

COMPSCI 105 S2 C - Assignment 2

Due Date: Friday 19th October 2018 at 6:00pm

Assessment

- Due: Part 1: **6:00pm, 12th of October 2018** and Part 2: **6:00pm, 19th October 2018**
- Worth: 100 marks in total = 5% of the final grade

Warning

DO NOT SUBMIT SOMEONE ELSE'S WORK:

- The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help
- Under no circumstances should you take or pay for a copy of someone else's work. This will be penalized heavily.
- Under no circumstances should you give a copy of your work to someone else
- The Computer Science department uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken

To ensure you are not identified as cheating you should:

- Always do individual assignments by yourself
- Never give another person your code
- Never put your code in a public place (e.g. forum or website)
- Never leave your computer unattended, you are responsible for the security of your account
- Ensure you always remove your USB flash drive from the computer before you log off

Part I of Assignment

Resources and Submission

- All submitted files must contain **your name and UPI in a comment at the top of the file**.
- All your files should be able to be compiled without requiring any editing.
- All your files should include a good layout structure, meaningful variable names, and comments explaining the key ideas of your solution
- All required resources are found on the assignment section in the course web page

Q1: Resources – `NodeDLL.py` and `LinkedListDLL.py`. Submission – `LinkedListDLL.py` and `LinkedListIterator.py`.
Q2: Resources – `NumberConverter.py`. Submission – `NumberConverter.py`.
Q3: Resources –`HashTable.py`, `A2Q3.py`, Submission – `HashTable.py`.

Aims of Part I of Assignment

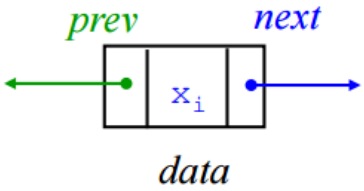
Understanding and solving problems using:

- Linked lists
- Recursion
- Hash tables

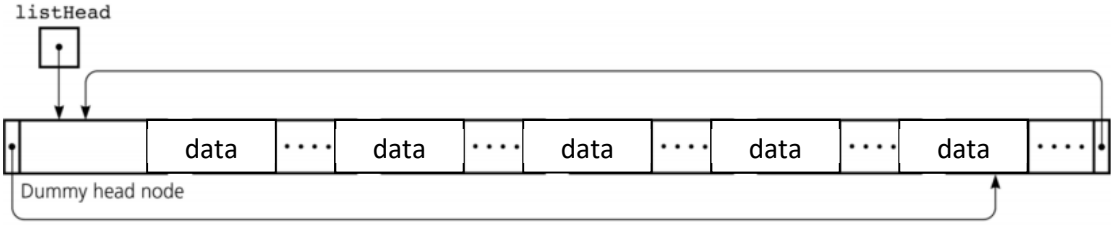
Q1. Implementing an iterator for the doubly linked list (15 Marks)

In Lecture 21 we discussed the implementation of a circular doubly linked list. Each node maintains a reference to:

- `prev` – the node before it in the linked list.
- `next` – the node after it in the linked list.



The circular doubly linked list also contains a dummy head node which is always present, even when the list is empty. The first node in the list is **next** to the dummy head node while the last node in the list is **previous** to the dummy head node.



In this exercise you will be provided with the implementation of the node class `NodeDLL` and an implementation of the list itself `LinkedListDLL`. What you will need to do is implement an iterator class, `LinkedListIterator` that will traverse the linked list from the first node in the list (next to the dummy head node) to the last node in the list (previous to the dummy head node). Your `LinkedListDLL` class should return an instance of the `LinkedListIterator` such that it is compatible with a for loop as shown in the example below. Note that your iterator should stop traversal once every node in the list has been visited.

```
my_list = LinkedListDLL()
for node in my_list:
    print(node,end="")
print()
for x in range(1,10):
    if x % 2 == 0:
        my_list.add_to_head(x)
    else:
        my_list.add_to_tail(x)
print("Contents:",end=" ")
for node in my_list:
    print(node,end=" ")
print()
my_list.remove_from_head()
my_list.remove_from_tail()
my_list.remove_from_tail()
print("Contents:",end=" ")
for node in my_list:
    print(node,end=" ")
print()
```

Will produce the output:

```
Contents: 8 6 4 2 1 3 5 7 9
Contents: 6 4 2 1 3 5
```

Marking Scheme for Question 1

Iterator traverse doubly linked list correctly	5 marks
Iteration stops once all nodes are visited	5 marks
Doubly linked list compatible with for loop syntax	5 marks

Q2. Implementing a recursive number converter (15 Marks)

In our everyday life we use numbers with a base of 10 – decimal numbers. In computer science we also use numbers with the following bases:

- base 2 – binary numbers
- base 8 – octal numbers
- base 16 – hexadecimal numbers

In this exercise we will develop a recursive algorithm to convert a positive decimal integer to its corresponding equivalent using one of the 3 bases described above.
The program that will use this algorithm is called `NumberConverter.py` which is one of the assignment resources. A screenshot of the program running is shown on the following page. Most of the code for this program has been implemented for you.
You will need to implement the `converter()` function which handles converting a decimal number to its binary, octal or hexadecimal equivalent. The `converter()` function takes 2 parameters:

1. `number` – decimal value to convert
2. `base` – base you will use in the conversion.

Example of the expected output from the conversion function is shown below.

```
*********************************************
**       A Number Converter Program        **
** Decimal --> Binary, Octal or Hexadecimal **

*********************************************
1. Convert to binary value
2. Convert to octal value
3. Convert to hexadecimal value
4. Quit the program
Please make a selection: 1
Please enter a positive integer: 14
14 is 1110 in binary
1. Convert to binary value
2. Convert to octal value
3. Convert to hexadecimal value
4. Quit the program
Please make a selection: 2
Please enter a positive integer: 15
15 is 17 in octal
1. Convert to binary value
2. Convert to octal value
3. Convert to hexadecimal value
4. Quit the program
Please make a selection: 3
Please enter a positive integer: 16
16 is 10 in hexadecimal
1. Convert to binary value
2. Convert to octal value
3. Convert to hexadecimal value
4. Quit the program
Please make a selection: 4
```

The algorithm you use when implementing the `converter()` function ***must be recursive***. Marks will not be allocated for non-recursive solutions. One approach that suits a recursive solution is:

- Repeatedly divide the decimal value by the base you want to convert to until you reach zero. Keep track of the remainders as you do so.
- Once you reach zero the remainders in order of most recent to least recent will give you your converted number.

An example of this can be seen below. The decimal value 13 is equal to 1101 in binary.

2	13	1
2	6	0
2	3	1
2	1	1
	0	

You also need to take into account the fact that hexadecimal numbers also include letters. These letters represent values 10 to 15 where a = 10, b = 11, c = 12, d = 13, e = 14, and f = 15.

Marking Scheme for Question 2

Binary conversion handled correctly	5 marks
Octal conversion handled correctly	5 marks
Hexadecimal conversion handled correctly	5 marks

Q3. Implementing a hash table that uses double hashing (30 Marks)

In lectures we discussed the implementation of a hash table that used linear probing for collision resolution. In this exercise you will modify this hash table class so that it uses double hashing for collision resolution. The hash table will also increase its size whenever its load factor is 0.75 or more. The hash table will contain key-data pairs. You can assume that keys will be integer values and data strings.

Your hash table class will be called `HashTable`. The constructor for this class has no parameter. Five instance variables will be initialized:

`size` – the size of the hash table

`count` – the number of items in the hash table

`slots` – a python list with size number of elements.

`data` – a python list with size number of elements.

`deleted` – a string consisting of the null character "`\0`".

You will need to modify the `put()` and `delete()` functions discussed in class.
You will also need to implement a number of your own functions. This should be done in steps as listed below:

1. Modify the `put()` and `delete()` functions so that the variable `count` is altered appropriately whenever a key-data pair is added to or deleted from the hash table. Change the `__len__()` function so that it returns this count.
2. Your hash table will use the remainder method as discussed in lectures: $key \% size$.
For collision resolution you will need to implement a second hash function: $size - (key \% (size - 1) + 1)$. You will need to alter the implementation of the `rehash()` function to use this second hash function.
3. You will need to implement a function `load()` that calculates the load factor of the hash table. Remember that the load factor is calculated using the formula: $\lambda = count/size$.
4. To increase the size of the hash table you want to find a prime number that approximately doubles its

current size. For example if your hash table has a size of 7, you will be looking at increasing the size of the table to 13. You will need to implement a function that finds the largest prime number in the range $n + 1$ to $2n$.

5. You will need to implement a `resize()` function that increases the size of the hash table appropriately and rehashes all existing key-data pairs into the appropriate slots in the larger hash table. You will need to modify the `put()` function to use this `resize()` function appropriately.

A file has been provided for you to test your hash table class – `A2Q3.py`. An example of this program running is shown below.

```
Test 1
------
Insertion:
key: 0 data: aardvark
{0:aardvark, , , , , , } count: 1
key: 1 data: bull
{0:aardvark, 1:bull, , , , , } count: 2
key: 2 data: cat
{0:aardvark, 1:bull, 2:cat, , , , } count: 3
key: 3 data: dog
{0:aardvark, 1:bull, 2:cat, 3:dog, , , } count: 4
key: 4 data: elephant
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, , } count: 5
key: 5 data: frog
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, , , , , , , } count: 6
key: 6 data: goat
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, 6:goat, , , , , , } count: 7
key: 7 data: horse
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, 6:goat, 7:horse, , , , , } count: 8

Deletion:
{, , , , , , , , , , , , , } count: 0

Test 2
------
Insertion:
key: 708 data: aardvark
{, 708:aardvark, , , , , } count: 1
key: 126 data: bull
{126:bull, 708:aardvark, , , , , } count: 2
key: 863 data: cat
{126:bull, 708:aardvark, 863:cat, , , , } count: 3
key: 21 data: dog
{126:bull, 708:aardvark, 863:cat, 21:dog, , , } count: 4
key: 495 data: elephant
{126:bull, 708:aardvark, 863:cat, 21:dog, , 495:elephant, } count: 5
key: 717 data: frog
{, 495:elephant, 717:frog, , , 863:cat, 708:aardvark, , 21:dog, 126:bull, , , } count: 6
key: 519 data: goat
{, 495:elephant, 717:frog, , , 863:cat, 708:aardvark, , 21:dog, 126:bull, , , 519:goat} count: 7
key: 750 data: horse
{, 495:elephant, 717:frog, , , 863:cat, 708:aardvark, 750:horse, 21:dog, 126:bull, , , 519:goat} count: 8
```

Marking Scheme for Question 3

Step 1	3 marks
Step 2	6 marks
Step 3	3 marks
Step 4	9 marks
Step 5	9 marks