California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Spring 2024 – Project #2 – Radix Sort

Overview

For this assignment, you will implement one of the most-clever sorting algorithms of all time – Herman Hollerith's Radix Sort.

To properly test your program, I'm going to provide some data files. The format of each file will be the same.

Also, as not to make this assignment too difficult, all the keys will be base-10 numbers. In other words, you can assume their will be 10 buckets in your sort.

Part 1: Key-Value Nodes

You are going to modify your linked-list class from the last assignment. In particular, you need to update the Node class. But, first, we need to create a new node to store Key-Value pairs.

We will use class this throughout the assignment. So, it will be public (and not embedded inside the Linked-List Class).

```
class Field
    public String Key
    public String Value
end class
```

Part 2: Updating Your Node Class

To store the key-value fields in the list, we need to update the Node Class from your last project..

```
class Node
    public Field Value     ← This was a string

    public Node Next
    public Node Prev
end class
```

Node Class		
	`Node(Field)`	Constructor
`Node`	**`Prev`**	The previous node in the chain.
`Node`	**`Next`**	The next node in the chain.
`Field`	**`Value`**	The value that the node contains. Do not declare it of type "object".

Part 3: Updating Your Linked List Class

Updating the Linked-List interface should be fairly straightforward. The following is the modified interface:

LinkedList Class		
`String`	**`About()`**	Returns text about you – the author of this class.
`void`	**`AddHead(Field item)`**	Adds an object to the head of the list. **This must be O(1).**
`void`	**`AddTail(Field item)`**	Adds an object to the tail of the list. **This must be O(1).**
`Field`	**`RemoveHead()`**	Removed an object from the head of the list. **This must be O(1).** If the list is empty, return an empty string.
`Field`	**`RemoveTail()`**	Removed an object from the tail of the list. **This must be O(1).** If the list is empty, return an empty string.
`Boolean`	**`IsEmpty()`**	Returns true if the linked list is empty. In other words, the head is null.
`String`	**`ToString()`**	Returns the contents of the linked list with comma separated values. **This must be O(n). Do not use recursion.**
`String`	**`ToStringReverse()`**	Returns the contents of the linked list, from last to first, with comma separated values. **This must be O(n). Do not use recursion.**

Part 4: The Queue

Once you have your Linked-List class working, it is time to create a queue class. You **must** use your linked list.. If you use another data structure (such as an array or one of the language libraries built-in classes), you will receive a zero.

Rather than using inheritance, you should embed an instance of the LinkedList Class inside your Queue Class. This will be fairly easy to write. The following is the interface for the Queue Class.

public class FieldQueue		
void	**Enqueue(Field item)**	Enqueues a Field onto the queue. Call the AddLast() method on the linked list instance.
Field	**Dequeue()**	Dequeues (removes) a Field from the front of the queue. Call the RemoveFront() method on the linked list instance.
boolean	**IsEmpty()**	Returns true of the stack is empty Call the IsEmpty() method on the linked list instance.
String	**ToString()**	Returns ToString() on the linked list instance.

Part 5: Input File Format

A number of test files will be provided to you for testing your code. The format is designed to be easy to read in multiple programming languages. You need to use the classes, built in your programming language, to read the source files.

The first line of the data contains the total digits in the key. You should save this value and use it to control the number of passes the Radix Sort will perform. Naturally, this can also be inferred from the data itself, but it's useful to have it explicitly stated. The records end with an entry called "END" with key of 0.

```
Digits in the key
Key 1
Value 1
Key 2
Value 2
...
Key n
Value n
0
END
```

The following is one of the most basic test files on the CSc 130 Website. It consists of only 13 records.

File: years.txt

```
4
2020
Great Toilet Paper Shortage
1839
Sutter's Fort founded
1846
Bear Flag Revolt
1947
Sacramento State founded
1977
Star Wars was born
2017
Star Wars franchise almost destroyed
1964
Buffalo wings invented
1783
United States Constitution enacted
1850
California joins the United States
1848
Gold Rush begins
1776
American Revolution
1980
Pacman was released
1953
Cheese Whiz was invented (Nacho Era began)
0
END
```

Each key is 4 digits. You can assume leading zeroes.

Part 6: The Radix Sort

For the actual Radix Sort, you are going to take advantage of your Queue class. You will need to create a total of **11** instances of your Queue.. The main queue will contain the data being sorted, and the remaining 10 will be the buckets. You can create an array[10] of Queues

Remember to save the first value from the file – this is the number of digits in the key and will control your For Loop. The basic logic of your program is as follows:

1. Read the contents of the file into your main queue (linked list).
2. The Radix Sort
   a) Empty the main queue into the 10 buckets based on the 1's digit
   b) Empty buckets back into the main queue (in order from 0 to 9)
   c) Repeat with the 10's digit, 100's digit, etc…
3. Print the results.

DO NOT convert the key to a number. Keep the key a string. That will make it far easier to grab the different digits.

Requirements

⚠	**You must use your linked-list (modified) from Assignment #1.** **Do not use any built-in queue library class, etc… If you do, you will receive a zero.** **No exceptions. No resubmissions.**

- This **must** be completely all your code. If you share your solution with another student or re-use code from another class, you will receive a zero.
- You **must** use your Linked-List from Assignment #1.
- You may use any programming language you are comfortable with. I strongly recommend not using C (C++, Java, C#, Visual Basic are all good choices).
- Proper style.

Proper Style

Well-formatted code

Points will be deducted if your program doesn't adhere to basic programming style guidelines. The requirements are below:

1. If programming C++, Java, or C#, I don't care where you put the starting curly bracket. Just be consistent.
2. Use the proper naming convention for your programming language. The interfaces, above, are using Microsoft's C# standard, but it's different in other languages.
3. Indentation must be used.
4. Indentation must be consistent. Three or four spaces works. **Beware the tab character**. It might not appear correctly on my computer (tabs are inconsistent in size).

5. Proper commenting. Not every line needs a comment, but sections that contain logic often do. Add a comment before every section of code – such as a loop or If Statement. Any complex idea, such as setting a link, must have a comment. Please see below.

The following code is well formatted and commented.

```
void foo()
{
    int x;

    //Print off the list
    x = 0;
    while (x < this.count)
    {
       items.Add(this[x]);  //Add the item to the temporary list
       x++;
    }
}
```

Poorly-formatted code

The following code is poor formatted and documented.

```
void foo()
{
    int x;

x = 0;
while (x < this.count)   {
     items.Add(this[x]);  //Call add on items.
       x++;
}
}
```

Grading

1	Field class	5%
2	Updated Linked-List Class	10%
3	Queue Class	10%
4	Reading the test file properly	15%
5	Sort implemented correctly	40%
6	Output the sorted results	10%
7	Proper Style	10%

Due Date

Due **March 15, 2024** by 11:59 pm.

Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment. **Do not send it to Canvas. No assignments will be accepted on Canvas.**

E-Mail the following to dcook@csus.edu:

- The source code.
- The main program that runs the tests.
- Output generated by your tests

⚠ **The e-mail server will delete all attachments that have file extensions it deems dangerous. This includes .py, .exe, and many more.**

So, please send a ZIP File containing all your files.