California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Fall 2022 – Project #2 – Radix Sort

Revision 2

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 Class

To store the values in the array, we need to create a key-value class. In this case, I called it "Entry", but you are welcome any other name that you like.

```
class Entry
   public String key
   public String value
end class
```

Note, this is far different from the Node class used by the linked list. Your Node class should be modified as follows:

```
class Node
   public Entry value
   public Node next
end class
```

Part 2: Your Update Linked-list Class

So, how do you implement the buckets? Well, they are essentially queues.

Modify your linked-list class to use the Entry class above. In your last assignment, it simply stored Strings. To receive credit in this assignment – you **must** use the solution to Assignment #1.

Do not change the interface – well, except for changing String to Entry. Do not submit a Queue class.

Part 3: 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".

```
Digits in the key
Key 1, Value 1
Key 2, Value 2
Key 3, Value 3
...
Key n, Value n
END
```

The following is one of the most basic test files on the CSc 35 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)
END
```

Part 4: Radix Sort

For the actual Radix Sort, you are going to take advantage of your linked list class. In particular, you are going to treat it like a queue (so, remove the front and add to the back). You will need **11** total. The main queue will contain the data being sorting, and the remaining 10 will be the buckets. You can create an array[10] of linked lists.

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.
   (Please note: The ZIP code file is so ridiculously large that the recursive Print(), you wrote before, will cause a stack overflow. You can, if you like, print the contents using a classic while loop).

DO NOT convert the key to a number. Keep it 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. Do not submit a Queue class (even if you wrapped your linked list).

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	Entry class	5%
2	Reading the test file properly	15%
3	Sort implemented correct	50%
4	Output the sorted results	20%
5	Proper Style	10%

Due Date

Due **November 1, 2022** 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.