Lab 4: Queues

Week of October 29, 2018

Objective

To compare radix sort to other sorts (by adding it to the Assignment 1 solution).

Exercise

The file `Lab4.java` contains a nearly-complete program to compare insertion sort, merge sort, quick sort, hybrid quick sort, and a newly-added radix sort. The only missing code needs to be added to the appendable queue class (details below).

The newly-added radix sort implements the list of numbers and the 10 buckets / bins as appendable queues. In each iteration of the radix sort, after each number has been added to the appropriate bucket, the buckets are catenated into one list by appending the queues to each other in order from the 0th bucket to the 9th bucket.

You will complete the implementation of the appendable queues in the file `Lab4.java` (details below). You will find the `AppendableQueue` class at the end of the file.

Note that the file `Lab4.java` contains THREE already-completed classes:

- A `Stats` class, which is the same `Stats` class that we gave you for Assignment 1,
- An `Item` class, which is used by radix sort — it stores an item to be sorted and a working copy of the item (from which we remove the digits one by one), and
- The `Lab4` class containing the `main()` method, the code that tests the sorting methods, and the sorting methods themselves, including radix sort.

What is an appendable queue? An appendable queue is an ordinary queue with the usual operations (enter, leave, front) and an extra (new) operation called append. Operation append takes two queues and appends the contents of one queue onto the end of the other queue, keeping the order of the items unchanged. For example, if `q1` is an `AppendableQueue` containing

```
q1 = (front) < 31 14 45 < (end)
```

and `q2` is another `AppendableQueue` containing

```
q2 = (front) < 65 17 < (end)
```

then `q1.append(q2)` should take the contents of `q2` and append it to the end of `q1`. After this append operation, `q1` should contain

```
q1 = (front) < 31 14 45 65 17 < (end)
```

and `q2` will now be empty. Notice that the items that were in `q2` (65 and 17) are still in the same order that they were in `q2` — 65 is closer to the front of the queue than 17.

What queue implementation must you use? You must implement the appendable queue as a singly-linked list with a `front` pointer (pointing to the first node in the queue) and an `end` pointer (pointing to the last node in the queue), and no dummy nodes. (The `Node` class you must use is already provided as a `private` class inside the `AppendableQueue` class with `public` instance members.)

The code you will write: You are to implement three methods for the appendable queue **using the required implementation** (above):

- `public void enter( Item newItem )`, which performs the standard queue enter operation.
- `public Item leave()`, which performs the standard queue leave operation.
- `public void append( AppendableQueue q2 )`, which is a destructive method that performs the append operation described above. It appends the contents of `q2` to the end of the calling `AppendableQueue` "`this`", leaving `q2` empty.

Note that all three operations, including `append`, must take **constant time**, that is, $O(1)$ time.

Add the required `AppendableQueue` code (and any helper methods needed) to `Lab4.java` without changing any of the existing code.

Note: You can change the constants defined at the beginning of the `Lab4` class, which control the testing that the code does, but don't change any of the already-written code.