String vs. MyStringBuilder vs. StringBuilder

For this assignment one needed to implement a linked list version of the standard Java StringBuilder (SB) class in a new class called MyStringBuilder (MSB). A template with detailed method descriptions was provided to help guide students in creating the methods.

The biggest difference between the standard SB class and the MSB class is that the underlying data structure of SB is a character array, while the structure for MSB is a linked list. The linked list is implemented using a private inner class called CNode Each node has 2 fields, a character, and the address of the next node.

The main advantage of a linked node versus a standard array is that new nodes (i.e. new data) only allocate memory when the data is added whiles arrays are of a fixed size once created. If an array becomes full a new contiguous block of memory must be allocated that is double the size of the current block. The other advantage of a linked list is that if a piece of data (i.e. a char) is deleted from the list, no other operations need to be performed. In an array implementation, all data after the deletion point must be shifted down one index to maintain the array’s order.

For this assignment students were also tasked with creating a benchmarking program (Assig2B.java) to test the run-time (RT) of the standard Java String class versus the MSB class vs the SB class on 3 main operations: append to the end, delete from the beginning, and insert at the middle. Since standard Strings are immutable, it is expected that the String class would perform the worst in mostly all cases. The data from testing also supported this hypothesis.

Append:

Both MSB and SB outperformed the standard String by several orders of magnitude. The graph of the data (Figure 1) shows average RT plotted on a log2 scale. For the lower operation tests (<100k) SB does have a slight advantage, however above 100k, MSB does on average seem to outperform SB, but not by a big difference. This is most likely due to the fact that MSB can simply add a new node each time, while the SB occasionally has to double the size of the array and copy everything into the new array which takes time and increases the average RT

Delete:

For the delete method MSB significantly outperformed both SB and the standard String by several orders of magnitude and has fairly constant RT of < 50 ns on average. The graph of the data (Figure 2) shows average RT plotted on a log2 scale. For SB and String, as N increased, the RT increased in a fairly linear manner while the RT for MSB stayed constant.

Insert:

In all tests with insert at the middle, SB was the most efficient (see Figure 3). Surprisingly MSB actually became worse with higher N’s compared to the String. Since the SB is comprised of an array, it has direct access to the insertion point. With the MSB however, each node up until the insertion point must be traversed in a linear fashion. This take significantly more time than going straight to the insertion point and shifting the rest of the elements up by one as the SB does.

Extra credit:

For extra credit I expanded the Assig2B program to also test the RT for the reverse operation, which is a standard method for SB. While there is no standard method for strings, I created one to see how the other two (SB & MSB) stacked up against it (see Figure 4). Reversing the string took so long that I only ended up testing it for 10K and 20K. The 20k had a RT of about 45 minutes so I did not test any higher. I did however test SB and MSB for all 5 N’s (10k, 20k, 40k, 80, and 160k). In all tests SB outperformed both MSB and String (by several orders of magnitude). Since strings are immutable it is no surprise that the string reverse method was enormously inefficient. MSB is still less efficient than SB once again due to the fact that in order to reverse the lists order, each link must be traversed until the end, giving a RT of close to O(N). SB probably uses a recursive algorithm, like quick sort in order to quickly reverse sort the character array.

Overall performance:

Based on the RT from the 4 tests it seems that SB is the overall most efficient implementation. Having direct access to each element in its data structure gives it an advantage in almost every situation except deleting at the front. If we had tested with delete at the end or middle, SB would have most likely been the best in all tests due to its direct access to every element. On average it takes less time to shift elements or allocate a whole new array than it does to proceeed through every link in a linked list.

If the goal is to be more efficient with memory allocation, linked lists (MSB) provide lower memory utilization compared to an array (SB). However, if the main concern is speed and efficiency, arrays appear to offer a big advantage.

Figure 1. Average Append Run-Time

Figure 2. Average Delete Run-Time

Figure 3. Average Insert Run-Time

Figure 4. Average Reverse Run-Time