UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Lab 09 - Heaps

Dr. Mark R. Floryan

November 2, 2018

1 PRE-LAB

This week, you will be building a custom MinHeap class as discussed in class. You will use this heap to implement an efficient HeapSort algorithm.

1. Download the provided starter code.
2. Implement the MinHeap.java class.
3. Implement the heapSort() method.
4. Test the correctness of your hash table using the provided tester.
5. **FILES TO DOWNLOAD:** heaps.zip
6. **FILES TO SUBMIT:** lab09.zip

1.1 MINHEAP.JAVA

First, you will be implementing the MinHeap.java class, which implements the following PriorityQueue interface:

```java
public interface PriorityQueue<T extends Comparable<T>> {

    /* places the value T onto the heap */
    public void push(T data);

    /* removes and returns the item with next priority
     * (i.e., lowest value)
     */
    public T poll();

    /* returns the next item to be polled, without removing */
    public T peek();

    /* returns number of elements on the heap */
    public int size();

}
```

Your MinHeap class will also have a few extra supporting methods necessary to make it work. You heap MUST be a heap implementation, as discussed in class, using an array (no linked list / tree like implementations). The other methods necessary are:

```java
// Constructs a heap from the given array
// pre-filled with the data in the heap
// data may need to be restructured
public MinHeap(ArrayList<T> data);

// Turns the internal array without
// heap ordering property into the
// equivalent heap with the ordering property
private void heapify();

// Percolate the item at index up until
// the ordering property is restored
private void percolateUp(int index);

// Percolate the item at index down until
// the ordering property is restored.
private void percolateDown(int index);
```

Once you are done, you can test your implementation using the provided tester. Remember that the tester is NOT meant to be a thorough check of your implementation. You should still write your own tests to make sure that it works. Also note that not all the tests will pass at this point, because you haven't implemented HeapSort yet (see next section).

1.2 HEAP SORT

Heap Sort is an algorithm that sorts a list using the methods from a MinHeap or MaxHeap. In the provided tester file, you'll find a method called heapSort(). Implement this method so that it uses the MinHeap you built to sort the provided list of numbers. Once you are done, you can re-run the provided tester to confirm that it seems to work.

2 In-Lab

The goal of this in-lab is to continue practicing with heaps in a laid back environment:

1. Get into small groups of two

2. The TAs will present you with some programming challenges regarding heaps. These will not be graded, but you are required to attend and to participate.

3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.

4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Heaps

The TAs will lead you in going through the following challenges. It is ok if you do not get through each of these.

1. Add a new method to your MinHeap called updatePriority(T oldElement, T newElement). This method should find oldElement in the heap, update it to be set to newElement instead, and then make sure the heap ordering property still holds.

2. Write a method called computeProduct(ArrayList<Integer> list, int i). This method should return the product of the smallest i elements in the given list. Note that list is not provided to you in sorted order. Use a MinHeap to solve this problem.

3. If you have extra time, try tackling this problem.

If you have coded up all of these and they work, then show the TA and you may leave early.

3 POST-LAB

The goal of this post-lab is to produce a report analyzing the performance of a min-heap.

You will perform some experiment(s) by doing the following:

1. Implement a priority queue naively by using one of our other data structures (e.g., list, tree, or hash table). You may use Java's built in data structures to make this easier if you'd like.

2. Run an experiment where you test the performance of your MinHeap (from prelab) to the naive implementation of the priority queue. Make sure you test the methods push(), poll(), and peek(). Which operations are slower? Which are faster? Do the results you see match the theoretical (i.e., big-theta runtimes) for the respective methods? Why or why not?

3. **FILES TO DOWNLOAD:** None

4. **FILES TO SUBMIT:** PostLabNine.pdf

3.1 REPORT

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.

- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.

- You should write your report as if it is original novel research.

- The grammar / spelling / professionalism of this document should be sound.

- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph
- **Introduction**: Present the problem, and provide details regarding the data structues you implemented.
- **Methods**: Describe your methodology for collecting data. How many executions, which inputs, when did the timer start/stop, etc.
- **Results**: Describe your results from your execution runs.
- **Conclusion**: Interpret your results. Which algorithm/data structure was fastest in each situation? If so, why? Does the performance you see match the theoretical runtimes of the algorithms? Why or why not?

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the experiments and how many execution runs were done in each group.
- At least one table (results section) summarizing the results of all of your experiments.
- Some kind of graph visualizing the results of the table from the previous bullet.