CSCI 102 assignment 3 – Iterators and Priority Queues

February 29, 2024

1 Iterators

Say we want to iterate through a linked list `list` and print every element. We consider the code

```
for (int i = 0; i < list.size(); i++){
    System.out.println(list.getAtIndex(i));
}
```

However this code has complexity $O(n^2)$. Ideally we could instead use the $O(n)$ code

```
Node<E> current_node = list.head;
for (int i = 0; i < list.size(); i++){
    System.out.println(current_node.getElement());
    current_node = current_node.next;
}
```

Unfortunately, `head` is private. One solution is to make it public, but we'd rather define an ADT that can be implemented without necessarily using nodes – for instance we might want to use the same code if we'd rather use an `ArrayList`.

Our solution allow each list implementation implement their own method to iterate. Java comes with the default interface for a list that can be iterated through, `Iterable<E>`. A class that implements `Iterable` must have a method `iterator` that returns an object that can iterate through the elements of the list. This object that can iterate through the elements of the list is another default class in `java.utils`, `Iterator<E>`. Here are the definitions of the two interfaces:

```
interface Iterable<E>{
    Iterator<E> iterator();
}

interface Iterator<E>{
    boolean hasNext();
    E Next();
}
```

Conceptually, `Iterator` starts at the beginning of the list and provides the next element of the list with each call to `Next()`; `hasNext()` simply returns whether or not `Iterator` has reached the end of the list.

Now if we have any list that implements `Iterable<E>` we can iterate through its elements as

```
Iterator<E> iter = list.iterator();
while (iter.hasNext()){
    E element = iter.next();
    System.out.println(element);
}
```

Java also allows one to write the above code more compactly using the syntax

```
for (E element : list){
    System.out.println(element);
}
```

which can be read as code that executes a command "for each element in the list".

- Modify the class code for `ArrayList` and `DoublyLinkedList` so that both classes implement `Iterator<E>`.
- Include a main method in each class that adds the numbers 1 through 10 to the list and then prints the contents of the list using the code

```
Iterator<E> iter = list.iterator();
while (iter.hasNext()){
    E element = iter.next();
    System.out.println(element);
}
```

Your code should be written so that this runs in $O(n)$ time!

2 Priority Queues

In the stack ADT, the element that was added last has the highest *priority* for removal. In the queue ADT, the element with the highest priority for removal was the one which was added first. Here we will define an ADT where one specifies the priority with a `double` when inserting it to a list; this ADT will be called a `PriorityQueue`:

```
public interface PriorityQueue<E> {
    int size();
    boolean isEmpty();
    void insert(doubly priority, E element);
    E removeMaxPriority();
}
```

`insert` adds a `element` to the PriorityQueue with priority `priority`; `removeMaxPriority` removes and returns the element with the highest priority for removal.

- Implement a `PriorityQueue<E>`.
- Include a main method that adds the numbers 1 through 10 to a `PriorityQueue<E>` with priorities $3, 4, 2, 7, 6, 8, 5, 9, 1, 0$ then iteratively removes each element in order of priority and prints them.
- Bonus: Implement a `Stack<E>` and `Queue<E>` as subclasses of your implementation of `PriorityQueue<E>`.
- Bonus (not for extra marks): Come up with an example of data that would naturally be stored in a `PriorityQueue<E>`.

Please submit your code and answers to the questions in a zipped folder on Brightspace by March 4.