School of Computer Science

COMP20010: Data Structures and Algorithms I
Semester I

Assignment 2
On: 22/11/2018
Due: 2/12/2018

Q1: _____ **(100points)**

(a) Write the pseudocode for a function to recursively remove all elements from a stack. [20]

Solution: *key points:*

- *standalone recursive function*
- *base case : empty stack*
- *stack.pop()*
- *recursive function call*

```
public void popAllRecursive() {

     if(stack.isEmpty()) {
        //nothing to remove, return
        return;
     }
     stack.pop();  // remove one stack element

     popAllRecursive(); // recursive invocation of your method
}
```

(b) Write the pseudocode algorithm which reverses the elements on a Stack using two additional Stack's (no other data structures are allowed). [20]

Solution: *push from S -> S1, S1 -> S2, S2 -> S*

```
function reverseStack(Stack S):
    Stack S1, S2

    while !S.isEmpty():
        S1.push(S.pop())

    while !S1.isEmpty():
```

```
        S2.push(S1.pop())

    while !S2.isEmpty():
        S.push(S2.pop())
```

(c) Write the pseudocode for a function to recursively reverse a queue. This function should work in-place, that is, it works directly on the queue and no other data structures can be used. [20]

Solution:

```
queue reverseFunction(queue)
{
    if (queue is empty)
       return queue;
    else {
       data = queue.front()
       queue.pop()
       queue = reverseFunction(queue);
       q.push(data);
       return queue;
    }
}
```

(d) Describe how to implement the deque ADT using two stacks as the only instance variables. What are the running times of the methods? [20]

Solution: *Outline using two stacks to implement a deque:*

use separate stacks for the left and right S_l, S_r. The Deque method `pushLeft()` *pushes to S_l, and similarly for* `pushRight()`. *The Deque* `popLeft()` *pop's from the S_l, similarly for* `popRight()`.

These methods are constant time complexity $O(1)$

(e) You have to provide an implementation of the Deque ADT but the only data structure you can use is a Stack. You will need 2 stacks S_A and S_B. Write the pseudocode to implement each of functions in Deque ADT using only these two stacks. [20]

Solution: *The basic elements of the ADT would look like this:*

```
pushLeft(E e)
  1) Push element e to S_l.

pushRight(E e)
```

```
  1) Push element e to S_r.

popLeft()
  1) If S_l and S_r are empty then error
  2) If S_l is empty, then
       while (S_r.isEmpty()) do
           S_l.push(S_r.pop())
        end while
  3) return S_l.pop()
```

There are more details on an dual array based Deque implementation using two stacks: Dual Array Deque: Building a Deque from Two Stacks.

Q2: _____ **(100points)**

(a) Write a java class to model a Student, using the student name (String), age(Integer), GPA(Double). The class should have a suitable `toString()` method and implement the `Comparable` interface. [20]

(b) The following Students are in a class: [80]

```
name,            age, GPA
Nataly Ware,     21, 4.0
Mira Weiss,      19, 3.5
Emilie Gibbs,    20, 3.2
Lisa Boone,      22, 4.7
Karsyn Terry,    20, 4.8
Jeremy Schwartz,18, 4.6
Aleah Gaines,    19, 4.1
Arianna Reeves, 20, 3.9
Walker Holloway,22, 3.8
Adelyn Walter,   24, 4.95
Damion Sanders, 25, 3.2
Aimee Quinn,     21, 2.7
```

When the students receive their end of year certificate, they are called in order of GPA. Use a priority queue to show the order the students are awarded the certificate. The ordering is determined by the Student.compareTo function. You need to write some java code, following this outline:

```java
1  public static void main(String[] args){
2      // Creating Priority queue
3      PriorityQueue<> pq = new PriorityQueue<Student>();
4
5      // Invoking a parameterised Student constructor with
6      Student student1 = new Student("Nataly Ware", 21, 4.0);
7      Student student2 = ...;
8
```

```
        pq.add(student1);
        pq.add(sutdent2);
        //...

        // Printing names of students in priority order

        while (!pq.isEmpty()) {
            System.out.println(pq.removeMin().toString());
        }
    }
```

You should implement the code in Java and submit the code. Also submit the output of the code: the priority queue sorting by GPA.

Solution: *The first part of the question is looking for an implementation of the Student class with a* `compareTo()` *function:*

```
// public class Student implements Comparable<Student> { ...
    public int compareTo(Student other) {
            if(this.gpa < other.gpa) {
                    return -1;
            }
            else if(this.gpa > other.gpa) {
                    return +1;
            }
            return 0;
    }
```

or

```
    public int compareTo(Student other) {
        return Double.compareTo(this.gpa, other.gpa);
    }
```

and it is also possible to use a comparator object:

```
private static Comparator<Student> GPACompare = new
    Comparator<Student>() {
    public int compare(final Student a, final Student b) {

        if (a.getGPA() > b.getGPA()) {
            return 1;
        } else if (a.getGPA() < b.getGPA()) {
            return -1;
        } else {
            return 0;
        }
```

```
        }
    };

    //
    PriorityQueue<Student> pq = new PriorityQueue<Student>(new
        GPACompare());
```

Q3: _____ **(100points)**

(a) Name one of the hash table collision-handling schemes which can tolerate a load factor above 1 and one which can not? [20]

Solution: *Separate chaining can support collisions. Linear Probing does not.*

(b) This was the original hashCode function used for Strings in Java. [20]

```
public int hashCode() {
   int hash = 0;
   int skip = Math.max(1, length() / 8);
   for (int i = 0; i < length(); i += skip)
      hash = (hash * 37) + charAt(i);
   return hash;
}
```

Suggest some problems with this hash code implementation and why the version mentioned in the lectures might be a better option.

Solution: *This was done in the hopes of computing the hash function more quickly. Indeed, the hash values were computed more quickly, but it became more likely that many strings hashed to the same values. This resulted in a significant degradation in performance on many real-world inputs (e.g., long URLs) which all hashed to the same value, e.g., http://www.cs.princeton.edu/algs4/34hash/*****java.html. Read: Hash Tables.*

Some more information on the choice of 31 or 37 is given here: What's Wrong With Hashcode in java.lang.String?

```
    public int hashCode() {
        int h = hash;
        if (h == 0 && value.length > 0) {
            char val[] = value;
            for (int i = 0; i < value.length; i++) {
                h = 31 * h + val[i];
            }
            hash = h;
        }
```

```
        return h;
    }
```

(c) Write out the 11-entry hash table that results from using the hash function, $h(i) = (3i + 5)mod11$, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining. [20]

Solution:

```
0 13
1 94,39
2
3
4
5 44,88,11
6
7
8 12,23
9 16,5
10 20
```

(d) Write a Java program which uses `java.util.Hashmap` to count the words in a file. Use the words from this file: sample_text.txt. Report the top 10 most frequently used words. [20]

Solution:

```
public static void main(String []args) throws FileNotFoundException {
        File f = new File("sample_text.txt");
        Scanner scanner = new Scanner(f);
        HashMap<String, Integer> counter = new HashMap<String,
            Integer>();

        while(scanner.hasNext()) { // read the file word at a time
                String word = scanner.next();
                //System.out.println("word:" + word);
                if(counter.get(word) == null) { // if the word is NOT
                    in the hashmap add a count of 1
                        counter.put(word, 1);
                } else {
                        counter.put(word, counter.get(word) + 1); //
                            otherwise increment the count by 1
                }
        }

```

```
        // sort the key, values...
    Map<String, Integer> sortedMapAsc = sortByValue(counter, ASC);
    printMap(sortedMapAsc);
}
```

```
a: 18
is: 18
pain: 18
pleasure: 18
that: 18
who: 21
the: 36
and: 42
of: 42
to: 46
```

(e) Using the list of words from words.txt

i. Find the number of collisions using polynomial accumulation with $a = 41$ [5]

Solution: *4*

ii. Find the number of collisions using polynomial accumulation with $a = 17$ [5]

Solution: *387*

iii. Find the number of collisions using a cyclic shift with a shift value of 7 [5]

Solution:

```
result = (result << shift) | (result >>> (32-shift));
ans: 97

result = (result >>> shift) | (result << (32-shift));
ans: 170
```

iv. Find the number of collisions which occur using the old Java hash code function: [5]

```
public int hashCode() {
   int hash = 0;
   int skip = Math.max(1, length() / 8);
   for (int i = 0; i < length(); i += skip)
      hash = (hash * 37) + charAt(i);
   return hash;
}
```

Solution: *5*

Solution: *To find the number of collisions I read all the words into a linked list, then compute the hash for each word. I use a map to count the number of times each hash code is computed:*

```
1   // read all the words into a linked list
2       public static void collisionCount(int a) {
3               File f = new File("words.txt");
4               try {
5                       Scanner scanner = new Scanner(f);
6                       LinkedList<String> words = new
                            LinkedList<String>();
7                       while (scanner.hasNext()) {
8                               String word = scanner.next();
9                               //System.out.println(word);
10                              words.add(word);
11                      }
12                      scanner.close();
13
14                      System.out.println("polyHash Collisions a(" + a
                            + "): " + polyHashCounter(a, words));
15
16              } catch (Exception e) {
17                      e.printStackTrace();
18              }
19      }
20
21      //
22
23      private static int polyHash(int a, String s) {
24          int result = s.charAt(0);
25
26              for (int i = 1; i < s.length(); i++) {
27                      result = result * a + s.charAt(i);
28              }
29              return result;
30      }
31
32      public static int polyHashCounter(int a, LinkedList<String>
            words) {
33              HashSet<Integer> set = new HashSet<Integer>();
34              int collisionCount = 0;
35
36              for (String word : words) {
37                      if (set.add(polyHash(a, word)) == false) {
```

```
38                                  collisionCount++;
39                              }
40                          }
41                          return collisionCount;
42                  }
```

You can do the same for different hash code functions (polynomials, cyclic shift etc).