CS310: Advanced Data Structures and Algorithmns

Spring 2014 Assignment 2 – Solution

Thanks to Prof. Betty O'Neil for parts of this solution.

1. Transfer qual solution to cs310/hw2. Test: cd to student dir, hw2 subdir.

   du should show (any nontrivial numbers at left):

   ```
   11  ./src/cs310
   12  ./src
   10  ./classes/cs310
   11  ./classes
   ```

   where the src/cs310 has the .java files for the qual. For hw2, enough to have src/cs310 there.

2. a. 'a' → 0, 'b'→ 1, ... , 'z' → 25, and also 'A' → 0,...'Z'→26. in one map:

   ```
   public static int mapOneLetter(char c)
   {
       return Character.toLowerCase(c) - 'a';
   }
   ```

   b. "aa" → 0, "ab"→ 1, ..., "az"→ 26, "ba"→ 26...

   ```
   public static int mapTwoLetter(String s)
   {
       return ((s.charAt(0) - 'a') * 26) + s.charAt(1) - 'a');
   }
   ```

   c. inverse of b

   ```
   public static String inverseMapTwoLetter(int x)
   {
        StringBuffer sb = new StringBuffer();
        sb.setCharAt(0, 'a' + x / 26);
        sb.setCharAt(1, 'a' + x % 26);
        return sb.toString();
   }
   ```

3. a. **Stack:** The resulting stack is: {8, 4} (read from left to right).

   b. **Queue:** The resulting queue is: {6, 1}.

   c. **Priority Queue:** Assuming 1 is the highest priority: {6, d8}.

4. From the onlne docs, the hash code for a String object is computed as:

   $s[0] * 31^{(n-1)} + s[1] * 31^{(n-2)} + ... + s[n-1]$

   using int arithmetic, where s[i] is the ith character of the string and n is the length of the string. The hash value of the empty string is zero.

   "".hashCode() = zero // by definition

   "G".hashCode() = $s[0] * 31^{(1-1)} = s[0] * 31^0 = s[0] * 1 = 71$

   "GH".hashCode() = $s[0] * 31^{(2-1)} + s[1] * 31^{(2-2)} = 71 * 31^1 + 72 = 2201 + 72 = 2273$

From the online docs, a hash code value for an Integer object is equal to the primitive int value represented by this Integer object.

5. a. We could have setup an array "int count[676]" (that's 26^2), which used the above function and the curPair as follows:

   count[mapTwoLetters(curPair)]++;

   b. If we didn't have the above function, and we used the HashMap, we'd dosomething like the following:

```
HashMap<String> counts = new HashMap<String>();
// iterate through key values: "aa", "ab", "ac", ... , "yz", "zz"
// and put an 0 in each spot.
for (...)
counts.put(key, 0);
...
...
// then we could just increment the count by saying
int count = counts.get(curPair);
counts.put(curPair, count+1);
```

6. Print a Collection in reverse order without using ListIterator. We can't iterate backwards, only forwards. However, we can easily dump any Collection to an arrayof Objects, and printing can be done via toString, an Object method, so that's the simplest way to go here.

```
public static <T> printReverse(Collection<T> c)
{
    Object a[] = c.toArray();
    for (int i= c.size()-1; i>= 0; i--)
       System.out.println(a[i]);
}
```

7. Problem 6.2:

   a.
```
public static int count(Collection<Collection<String>> c, String str)
{
  int counter = 0;
  // Iterate over every member
  Iterator<Collection<String>> it = c.iterator();
  while(it.hasNext()) {
      Iterator<String> its = it.next();
      while(its.hasNext()) {
         String s = its.next();
         if(s.equals(str)) counter++;
      }
  }
  return counter;
}
```

   b. The runtime is $O(N^2)$ since we go over all the N collections and for each one of them we go over N strings.

   c. A quadratic algorithm takes 9 times more when the input size is triplicated, hence 18ms.

8. List<String> list1 with ("A","B","C","D")

   a.
   1. list1.iterator().next(): "A"
   2. list1.listIterator().next(): "A"
   3. list1.listIterator(2).next(): "C"

4. list1.listIterator(4).previous(): "D"

b. remove instead of next:

1. fails with IllegalStateException
2. fails with IllegalStateException
3. fails with IllegalStateException
4. fails with IllegalStateException

Here, none have done a next or previous yet to set up for the remove.

c. The sequence of commands will result in the following: list1.listIterator(2).next(); list1.listIterator().remove(); list1.listIterator(4).previous() At first "C" will be returned and then removed (the last call to next). The last command will throw an exception because the list has only 3 members.

9. a. . {, {[, [{(, [{, [{(, [{, [{(, [{, [, [(, {, empty: OK, balanced

b. {, {[, [{(, [{, unmatched symbol seen: not balanced

10. Modified checkBalance:

```
// modified a little to return its conclusion to caller
// (not required for hw solution)
public boolean checkBalance(String brackets)
{
  char ch = 0;
  char match;
  Stack<Character> pendingTokens = new Stack<Character>();
  for (int i = 0; i< brackets.length(); i++) {
    ch = brackets.charAt(i);
    switch (ch) {
     case '(': case '[': case '{':
         pendingTokens.push(ch);
         break;
     case ')': case ']': case '}':
         if (pendingTokens.isEmpty()) {
           System.out.println("Extraneous "+ ch + " in string");
           return false;
         } else {
           match = pendingTokens.pop();
           if (!checkMatch(match, ch))  <--Bug in Weiss!!
             return false;
         }
         break;
     default:
                System.out.println("Non-bracket: "+ ch + " in string");
        return false;
    }
    if (!pendingTokens.isEmpty())
      return false; // incomplete expr (missing test in Weiss)
    return true;  // passed tests
}

private boolean checkMatch(char openCh, char closeCh)
{
    if (openCh=='(' && closeCh != ')' ||
          openCh=='[' && closeCh != ']' ||
          openCh=='{' && closeCh != '}') {
          System.out.println("Found " + closeCh + " not matching "+ openCh);
          return false;
     } else return true;
}
```