CS310: Advanced Data Structures and Algorithms

Fall 2022 Assignment 2

Due: Sunday, October 2, 2022 on Gradescope

Goal

Practice lists, stacks, Sets and Maps. Some runtime revision.

Questions

1. (a) Suppose a `List<String> list1` has elements “A”, “B”, “C”, and “D”. What is returned by:

    1. `list1.iterator().next();`

       “A”

    2. `list1.listIterator().next();`

       “A”

    3. `list1.listIterator(2).next();`

       “C”

    4. `list1.listIterator(4).previous();`

       “D”

   (b) Say what is deleted (or what happens) if next/previous is followed by removed in all of the above operations. Explain (**answer each part separately. That is — assume the list is back intact after you answer part 1, and you start with a fresh copy of a 4-item list for part 2**).

    1. “A” will be deleted.
    2. “A” will be deleted.
    3. “C” will be deleted.
    4. “D” will be deleted.

   (c) If we had the following sequence of commands:
   `list1.listIterator(2).next();`
   `list1.listIterator(2).remove();`
   `list1.listIterator(4).previous();`
   What would be returned? What would the list look like following these operations?

   The first command would return “C”. Then the second command would delete that value. However, the last command wouldn’t run because the list at this stage would not have a value at index 3.

2. (a) Say you have a `HashMap<Integer, Integer>` of size 1000 and one search operation takes approximately 1ms. How long will one search take (approximately, on average) on a `HashMap<Integer, Integer>` of size 2000?

   The average runtime for a search through a HashMap is O(1). Therefore, the average runtime is independent of the size of the map. So a HashMap with size 2000 would also run in 1ms.

(b) Same question, only `TreeMap<Integer, Integer>`.

The average runtime for a TreeMap is $\log(n)$. Therefore, if a TreeMap of size 1000 runs in 1ms, a TreeMap of size 2000 should run a little bit longer than the 1000-size TreeMap.

(c) Same, but a `LinkedList<Integer>`.

A LinkedList has an average runtime of O(n). Therefore a LinkedList of size 2000 would run in 2ms.

3. Write Java functions (static methods) that provide the following computed mappings. Do not use Maps, just very simple functions **using character arithmetics**, 1-2 lines should suffice:

(a) 'a' $\rightarrow$ 0, 'b' $\rightarrow$ 1, ..., 'z' $\rightarrow$ 25, and also, 'A' $\rightarrow$ 0, ..., 'Z' $\rightarrow$ 25 (in one map). Note that Java supports arithmetic with char variables: ch - 'a' is 0 if char ch is 'a', 1 if it is 'b', and so on.

LetterMappings.java

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

(b) "aa" $\rightarrow$ 0, "ab" $\rightarrow$ 1, "ac" $\rightarrow$ 2, ..., "az" $\rightarrow$ 25, "ba" $\rightarrow$ 26, "bb" $\rightarrow$ 27, ..., "zz" $\rightarrow$ (26*26*-1)

LetterMappings.java

```
public static int StringToNumber(String sequence) {
    char[] letters = sequence.toCharArray();
    return 26 * ((int)(letters[0]) - 'a') + (letters[1] - 'a');
}
```

(c) The inverse of b: input a number and return a pair of letters (in other words — reverse the directions of the arrows in b).

LetterMappings.java

```
public static String NumberToString(int num) {
    char[] letters = new char[]{(char)(num / 26 + 'a'), (char)(num % 26 + 'a')};
    return new String(letters);
}
```

4. What methods of Map can be implemented in O(1) time with a good hash function and a properly-sized hash table? What methods of Set? Could we implement List with a hash table? Explain.

For a Map, the following methods have a constant runtime: `get()`, `remove()`, `containsKey()`, `put()`, and `size()`. Similarly, all of the similar methods in Set are also constant time methods. Theoretically, we can implement a List using a hash map since all the functions associated with searching, adding, and iterating through them have the same runtime.

5. Given below are descriptions of some computer programs. Each of them reads a text file from standard input. For each of them specify:

- Which APIs would you use. Choose from APIs discussed in class: List / Map / Set
- Which implementations of the APIs would you use, e.g. Linked List / HashTable / Tree ...
- Describe how you would use the API to implement the program. Give pseudocode or describe step by step how your program would work. No need to write an actual program.

- Give the time complexity of your program in terms of the number of words or lines read (whichever is appropriate).

Try to make your programs as simple and efficient as possible. What the programs should do:

(a) Print lines of the file in reverse order.

For this program, I would use a List API. More specifically, I would use either the `ArrayList` or `LinkedList` implementations. Because we are only tracking each line and their order, a list is ideal for this situation. The program would read and scan through each line of the file (using a `Scanner` object). Each time it reads a line, we would call the list's `add` method. After we complete the scan, we can use a `for` loop to iterate through the list backward with the list's `get` method. The runtime of this program would be O(n).

(b) Print all different words in the file, each word printed exactly once (order not important).

There are two different ways to complete this task, either a list or a map. Because we don't care about printing out each word's occurrence in the file, a list could be used. However, with a list API, duplicates aren't checked, meaning that we would need to check if the word already exists in the list before calling `add`. The other way is to use a map where duplicates are not possible with the `put` method. Because the `contains` method in the list API has a constant runtime, I would use a list (similar to above). For each line, the program would check if the word exists in the list (with the aforementioned `contains` method). If it already exists, we would move on to the next line. If it doesn't, then the program would add the word to the list and move on to the next word. After the whole file is scanned, we would again loop through the list (with `for`) and print each word. The runtime would again be O(n).

(c) Print all different words in the file, with each word print how many times it occurs in the file (order not important).

This program would use the `HashMap` implementation of the Map API. This allows us to keep track of each word and the number of occurrences of each word. We can then read through each word of the file using a loop. For each iteration of this loop, the program would create a temp integer variable `count` which would be equal to 0. Then we can check if the word we scan already exists in the map. If it doe, we set the temp variable to the current value associated with the word key. Then we can call the `put` method with the word and `++count` (this would set the count to 1 for new words or increment for existing words). Then we can use HashMap's `forEach` method to print each key and value pair. The runtime for this program is O(n).