Homework 4

The following exercise will have questions you are required to do and submit, and questions you don't have to submit. We added more questions to help you prepare for the midterm. You will have the signature for the questions in those files. If a question (function) is colored green and have a plus (+) in the end, that means this question is mandatory (YOU MUST DO). Otherwise, it is your choice we recommend however, you to do those questions as part of your midterm prep'. Test your code yourself before using getfeed and keep the tests.

For the entire Homework: When a function returns an array the size must match the output. No extra unused elements.

PrimeOps

1. **The Sieve of Eratosthenes +**
   A prime number is a number strictly bigger than 1 which is only divisible by 1 and by itself. There is an infinite number of prime numbers, and here are the first few of them: 2, 3, 5, 7, 11, 13, 17, 19, ... Your program should implement the Eratosthenes algorithm presented in lecture 4-2. Here is an example of the program's execution:

   seive(40) ; // in main function , second line is the output in the terminal.
   The prime numbers up to 40 are: 2 3 5 7 11 13 17 19 23 29 31 37.

   Implementation note: When implementing the algorithm, you can stop the "skipping process" at $\sqrt{N}$. The reason is related to the following observation: if $x = a \times b$, then $a \leq \sqrt{x} \leq b$.

StringOps

In this part you may only use the following String functions:

1. str.charAt(int i);
2. str.length();
3. str.substring(int start);
4. str.substring(int start,int end);

You can use functions you build in Homework 3, StringOps by just copying them to this file.

1. **Trim +**

   The Trim function gets a string and a char and remove the char from the beginning and the end of the function.

   Examples:

   trim(“aaaaaahappy birthdayaaa”,’a’); // “happy birthday”

   trim(“aaaaaahappy birthdayaaa”,’b’); // “aaaaaahappy birthdayaaa”

2. **Last index of**

   The function gets a string and a char and returns the last appearance of the char in the string. If doesn’t appear returns -1.
   lastIndexOf(“abccb”,’b’); // 4
   lastIndexOf(“abccb”,’a’); // 0
   lastIndexOf(“abccb”,’v’); // -1

3. toBinary

The following function receives an int and returns it as the binary equivalent .
Note: you may not you any build in function other than basic operators.
Converting a number from a decimal base (base 10) to binary base is being done in a series of division operations. The first thing you need to do is get the reminder of the number by dividing the number by 2 and write it down, then we will divide the number by 2. The process will repeat itself until we reach 0. The letters should be written from right to left.
For example, let's take the number 6 and find its representation in binary:

- We will divide 6 by 2 and get 3 with reminder 0. We will write down the reminder 0. (Current solution: 0)
- We will divide 3 by 2 and get 1 with reminder 1. We will write down the reminder 1, (Current solution: 10)
- We will divide 1 by 2 and get 0 with reminder 1. We will write down the reminder 1. (Current solution: 110)
- We reached 0. Therefore, we finished.

toBinary(6) = "110"

4. Tokenize +

The following function receives a string and returns an array of string which that in each element in the array has a word (a word is defined by a sequence of non space chars)

```
String [] arr = tokenize("   x  + Math.sqrt(x)    - rate  ");

printStringArray(arr); // {"x", "+", "Math.sqrt(x)","-","rate"}

System.out.println(arr.length); // 5
```

CodeOps

You can use functions you build in Homework 3, StringOps by just copying them into the StringOps file. And call them from there like you do with Math library:

Example: StringOps.expand(str)

1. **letter frequencies +**
   Many text analysis programs need to count how many times the letters of the alphabet appear in a given text. The frequencies program does just that, and the printHistogram function shows the results using a histogram. (a star for every appearance in the string).
   For example:
   printHistogram(frequencies("As Easy as A, B, C"));

```
a:****
b:*
c:*
d:
e:*
f:
```

… (the program actually prints 26 lines, one per letter, but we skip a few to save space)

```
r:
s:***
t:
x:
y:*
z:
```

In cryptography, “cyclic shift encoding” is a simple encryption technique in which each letter in a given text is replaced by a letter some fixed number of positions down the alphabet. The fixed offset is called *key*. Without loss of generality, let’s assume that we shift to the right, cyclically. To illustrate, suppose we have an English alphabet consisting of 5 letters only (instead of 26), and we use *key* = 2. This implies that the letters will be shifted as follows:

Assume you have 5 letters: a b c d e

Decoded input: abcde

Encoded output: cdeab

position:
0 1 2 3 4 *N*=5,*key*=2
2 3 4 0 1 (*x* + *key*) % *N* // where *x* represents the position

The reverse decoding operation, i.e. going from an encoded input “back” to a decoded output using a given key, can be performed using a similar algebraic operation.

Given this encryption scheme, the text "bad deed", for example, will be encoded as "dca abba". For simplicity, we assume that the text contains only lowercase English letters, and possibly space characters. The encoding operation ignores the space characters, i.e. leaves them as is.

Note the relationship between the letters and their positions, or index values. In the above example, the index values of 'a', 'b', 'c', 'd', 'e' are 0, 1, 2, 3, 4, 5. In ASCII, the numeric values of the char values 'a', 'b', 'c', ..., 'z' are 97, 98, 99, ... 122. Thus, if we subtract 97 (the ASCII value of 'a') from each one of these char values, we get that 'a', 'b', 'c', ..., 'z' will be represented by 0, 1, 2, ..., 25 (altogether, 26 letters). We emphasize, once again, that in Java, a char value like 'a' is actually represented by the numeric value 97. Therefore, a statement like System.out.print('a' + 2) will print 99. Taken together, all these tips give you everything you need in order to write a program that encodes and decodes texts using the method described above. And, after writing this program you will understand perfectly well how char values are handled, and how they can be manipulated.

2. Encode +

The program receives a string and a key (which is bigger or equal to -26, represented by an int), and eturnss the encoded string

Example:

```
encode("defend the northern wall", 4);    // “hijirh xli rsvxlivr aepp”
```

3. Decode +

The program receives an encoded string and a key (which is bigger or equal to -26, represented by an int), and prints the decoded string.

Example:

```
decode("hijirh xli rsvxlivr aepp", 4); // “defend the northern wall”
```

4. Decode

The program works the same way as decode does but without key. How does the second version of decode work? The most frequent letter in English texts that are large enough is 'e'. 4. you have to make two assumptions:

(i) the given text was encoded from a text written in the English language,

(ii) The most frequent letter in the original text was indeed 'e'.

To illustrate the importance of these assumptions, note that the program will have no problem decoding the encoded version of "far out in the uncharted backwaters of the unfashionable end of the western spiral", but it will fail to decode correctly the encoded version of "far out in the uncharted backwaters". Why? Because in the former text, the most frequent letter is 'e', and this is not the case in the latter text. In general, if the text is sufficiently long, and is in English, we can be quite sure that the bonus decoding will work correctly.

ArrayOps

1. Sum array

This function takes an int array and returns the sum of the array.

Examples:

```
System.out.println(sumArray({1,2,3})); // 6

System.out.println(sumArray({1,-8,10})); // 3
```

2. isSorted +

This function takes an int array and returns if the array is sorted from the minimum value to the maximum value.

Examples:

```
System.out.println(isSorted({1,2,3})); //true

System.out.println(sumArray({1,-2,3})); // false

System.out.println(sumArray({1,1,3})); // true
```

3. BruteForce

This function takes an int array and an int value and returns the index the value appears in if it does appear. Otherwise returns -1. The way the algorithm works is by going over the array and checking every element.

Examples:

```
bruteForce({8, 2, 3, 4, 5}, 1); // 0
bruteForce({1, 7, 3, 4, 5}, 5); // 4
bruteForce({1, 2, 3, 4, 5}, 6); // -1
```

4. Binary Search

Binary search works on sorted arrays. Binary search begins by comparing an element in the middle of the array with the target value. If the target value matches the element, its position in the array is returned. If the target value is less than the element, the search continues in the lower half of the array. If the target value is greater than the element, the search continues in the upper half of the array. By doing this, the algorithm eliminates the half in which the target value cannot lie in each iteration. (Wikipedia)
Note: you can assume array is sorted.
Try and test both bruteForce and binarySearch, see how much time it takes to find the element in similar arrays.
Examples:

```
binarySearch ({1, 2, 3, 4, 5}, 1); // 0
binarySearch ({1, 2, 3, 4, 5}, 5); // 4
binarySearch ({1, 2, 3, 4, 5}, 6); // -1
```

5. findMissingInt +

This function takes an int array of length n, all elements of the array are the numbers 0 to n and every element appears only once, return the missing number from the array
Challenge: use only primitive variables.
Examples:

```
findMissingInt ({0,1, 2, 3, 4, 6}); // 5
findMissingInt ({2, 3, 1}); // 0
findMissingInt ({0}); // 1
```

6. secondMaxValue +

This function takes an int array and return the second largest number in the array.
You can assume arr.length > 1.
Examples:

```
    secondMaximum ({1, 2, 3, 4, 5}); // 4
    secondMaximum ({2, 8, 3, 7, 8}); // 8
```

7. merge +

This function takes two given sorted int arrays and return one array contains all elements from the two arrays, sorted.

Examples:

merge({1, 3, 5}, {2, 4, 6}); // {1, 2, 3, 4, 5, 6}
merge({1}, {1, 2, 3}); // {1, 1, 2, 3}

SetOps

A set is a distinct group of elements. In the following class you will have to do Set operations using arrays. While first you must build a set from an array.

1. **Contains +**

   The function receives an array and an int value and checks if an element appears in the array or not.
   Examples:
   contains({1,2,3},1); //true
   contains({1,2,3},4); //false

2. **Contains +**

   The function receives an array and an int value, and int index and checks if an element appears in the array up to that index not.
   Examples:
   contains({1,2,3,1},1,0); //false
   contains({1,2,3,1},1,4); //true
   contains({1,2,3},4,2); //false

3. **Unique elements +**

   The function receives an array and returns the number of distinct elements in the array.
   Examples:
   uniqueElements({1,2,3,1}); //3
   uniqueElements({1,1,1,1}); //1
   uniqueElements({1,2,3,4}); //4

4. **buildSet +**

   The function receives an array and returns an array of distinct elements.
   Examples:
   buildSet({1,2,3,1}); //{1,2,3}
   buildSet({1,2,2,1,1,4}); //{1,2,4}

   implementation notes :

   - Use the first 3 questions to solve this.

5. Union

The function receives 2 arrays and returns the union set of them. The union of set1 and set2 contains elements that appear in either of in set1 or in set2. Note: each element should appear once.

Examples:

unionSets({1,2,1,2,3},{4,3,2,7}); // {1,2,3,4,7}
unionSets({4,3,2,7},{1,2,1,2,3}); // {1,2,3,4,7}
unionSets({1,2,3},{3,2,1}); // {1,2,3}

6. Intersection +

The function receives 2 arrays and returns the intersection set of them. The intersection of set1 and set2 contains elements that appear in set1 and in set2. Note: each element should appear once. Assume the result will have at least 1 element.

intersectionSets({1,2,3},{2,3}); // {2,3}
intersectionSets({1,2,3,4,5},{6,2,3,8,11}); // {2,3}
intersectionSets({6,2,3,8,11},{1,2,3,4,5}); // {2,3}

7. Difference +

The function receives 2 arrays and returns the difference between them. The difference of set1 - set2, contains elements that appear in set1 doesn't appear in set2. Note: each element should appear once. Assume the result will have at least 1 element.

diffSets({1,2,3},{2,3}); // {1}
diffSets({1,2,3,5},{2,4,3,6,7}); // {1,5}
diffSets({2,4,3,6,7},{1,2,3,5}); // {4,6,7}

8. SymmetricDiff

The function receives 2 arrays and returns the symmetric difference between them. The symmetric difference of set1, set2, contains elements that appear in exactly 1 set. Note: each element should appear once. Assume the result will have at least 1 element.

symDiffSets({1,2,3,5},{2,4,3,6,7}); // {1,5,4,6,7}
synDiffSets({2,4,3,6,7},{1,2,3,5}); // {1,5,4,6,7}

Submission

Before submitting your work for grading, make sure that your code is written according to our Updated Java Coding Style Guidelines.

Submit the following five files:

- ➢ PrimesOps.java
- ➢ StringOps.java
- ➢ CodeOps.java
- ➢ ArrayOps.java
- ➢ SetOps.java

Get feedback: To get feedback about your programs before submitting them, use GETFEED, as many times as you want.

Deadline: Submit Homework 4 no later than Wednesday, December 7th , 2022, 23:55. You are welcome to submit earlier.