CMPE 343
Fall 2020
Programming Homework #5

This assignment is due by 23:55 on Wednesday, 6 January 2021.

Late submissions: 20 points will be deducted for each late day.

PART I (60 points)

Molecular biology scientist, Sheldon James, works in a company to find a drug for COVID-19 disease. For this purpose, DNA structure should be investigated carefully and since it has a remarkable amount of repetitive behavior, James needs an efficient algorithm for detecting these redundancy. At that point, he asks for your help to find the longest repeated sequence in a long string of characters. As an example, `abra` is the longest repeated sequence in the string `abracadabra`.

At first step, you need to implement a brute-force approach to solve the problem, to see how much computation is required with respect to time and available space. That algorithm checks the length of the match at every possible pair of starting positions, so it has ~O (n^2) character comparisons, with the constant depending on match lengths. The working brute-force algorithm presents a basis testbed for improving.

At second phase, you are required to improve brute-force method by removing some inefficiencies or discovering some completely different method. Even though there are considerable amount of algorithmic solutions to this problem, your approach should yield the fastest one. Beside these, you should use available space reasonably as well.

Your program should read the input string from your own generated file and print the results to standard output. You need to filter out any redundancies (non-printable characters etc.) to eliminate the noises. Then, it should output the length of the repeated substring on the first line and substring itself on the second line. If `sampleinput1.txt` file includes `abracadabra` string only, the output would be as follows:

```
% java MyMain sampleinput1.txt

4
abra
```

You need to run your designed algorithm on your own generated inputs (at least 2) and show the running time in seconds. For comparison, you should select two more available approaches to this problem from literature and analyze the asymptotic running times and memory consumptions as a function of the string length *N* and the length of the longest match *L*.

WHAT TO HAND IN

A zip file containing:

- The Java sources for your program.
- A **maximum-2 page** PDF report that explains:
  - the new function you have implemented, and that compares performance with others,
  - a short discussion on why your approach works better than the others.

PART II (40 points)

The Zeybek is our marvelous traditional culture and affects many others in the world. Sheldon James is also one of them and wants to learn to dance Zeybek with a great enthusiasm. Zeybek is danced by following four fundamental moves:

1. Crouching and turning
2. Overthrow
3. Raising arms
4. Proper step progress

Sheldon asks for your help to learn Zeybek and it includes 4 fundamental moves (as listed above) that are arranged into a particular sequence. Then this sequence can be repeated an arbitrary number of times. For example, if the sequence is "123", then the Zeybek is danced like this: "12312312312312…".But if the sequence is "123124", then the correct moves are "123124123124123…"

You are given some of the moves of a particular dancing. Those steps will contain $2 \leq N < 3$ times the basic sequence. You must continue the dancing.

For example, if the basic sequence is "123", we can have the following possibilities:

Input	Output
123123	12312312...
1231231	23123123...
12312312	31231231...

First line includes an integer that shows the number of test cases. Each case contains some of the first moves of a Zeybek. It is a single line with a list of digits (1, 2, 3 or 4) with no spaces between them. It will not have more than 1000 steps. You should remember that the case contains the basic sequence twice, and possibly has some more steps. Sample input looks like that:

```
6
123123
1231231
12312312
123124123124
12312412312412
12312412312412312
```

For each test case, the output should contain the 8 following steps of the dancing, followed by three dots "…" as follows:

```
% java MyMain2 sampleinput2.txt
12312312...
23123123...
31231231...
12312412...
31241231...
41231241...
```

WHAT TO HAND IN

A zip file containing:

- ➔ The Java sources of your program.
- ➔ The Java sources should be WELL DOCUMENTED as comments, as part of your grade will be based on the level of your comments.

For both parts of the homework, you may use the Strings and related classes from our textbook and its web site (https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/). Two zip files from both parts should be merged into a one and uploaded to Moodle.

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. You should submit your homework to course Moodle page before the deadline. No hardcopy submission is needed. You should send files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., .zip, .rar).
2. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.
3. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.
4. For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. You ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).
5. The submissions that do not obey these rules will not be graded.
6. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.
7. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-----------------------------------------------------

// Title: MyStrings

// Author: Ibrahim ILERI

// ID: 2100000000

// Section: 0

// Assignment: 3

// Description: This class defines Strings

//-----------------------------------------------------
```

8. Since your codes will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc.

Make sure that you explain each function in the beginning of your function structure. Example:

```
 void setVariable(char varName, int varValue)
   //---------------------------------------------------------------
   // Summary: Assigns a value to the variable whose name is given.
   // Precondition: varName is a char and varValue is an integer
   // Postcondition: The value of the variable is set.
   //---------------------------------------------------------------
   {
           // body of the function
   }
```

- Indentation, indentation, indentation...

9. This homework will be graded by your TAs, İbrahim İleri and Hamid Ahmadlouei (ibrahim.ileri@tedu.edu.tr, hamid.ahmadlouei@tedu.edu.tr). Thus, you may ask them your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructor Tolga Çapın and Ulaş Güleç for help.

GRADING RUBRICs

PART I (60 points)						
Performance Element	**Weight**	**Master (4/4)**	**Advanced (3/4)**	**Developing (2/4)**	**Beginner (1/4)**	**Insufficient (0/4)**
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc.).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.

Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: Strings	10%	Uses strings based data structure for implementati on. And provides all functionality.	Uses strings based data structure for implementat ion. And provides most of expected functionaliti es.	Implements strings with major deviations from the specification.	Used different data structure from strings to solve this problem.	Not even an attempt.
Functionality	40%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemente d correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set.	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.
Documentation (PDF Report)	25%	Every methodology step has a clear and compact explanation within page limits. Analyses are complete. Supported with tables and graphics.	Most steps have an explanation, but missing in some parts like complete visualization elements or exceed page limits too much.	Attempted to document but they are not clear. No visualization elements. Steps don't have explanations and conclusion.	Sporadic.	Not even an attempt.

PART II (40 points)						
Performance Element	**Weight**	**Master (4/4)**	**Advanced (3/4)**	**Developing (2/4)**	**Beginner (1/4)**	**Insufficient (0/4)**
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc.).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: Strings	10%	Uses strings based data structure for implementation. And provides all functionality.	Uses strings based data structure for implementation. And provides most of expected functionalities.	Implements strings with major deviations from the specification.	Used different data structure from directed graphs to solve this problem.	Not even an attempt.
Functionality	65%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemented correctly. Compiles but several warnings.	No working solution or does not compile.

Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set.	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.