COMP4500/7500
Advanced Algorithms and Data Structures

School of Information Technology and Electrical Engineering
The University of Queensland, Semester 2, 2014

Assignment 2

Due at 1pm, Monday 20 October 2014.
This assignment is worth 20% (COMP4500) or 15% (COMP7500) of your final grade.

Please read this entire handout before attempting any of the questions.

This assignment is to be attempted **individually**. Your solutions consist of two parts.

- The programming parts which are the Java files
  - RecursiveStringTransformation.java
  - DynamicStringTransformation.java

  No other Java files will be accepted. These files must be submitted via BlackBoard.
- All remaining parts may be submitted in one of two ways and must contain your name, course code and student number.
  a. As a paper copy complete with cover sheet via the EAIT submission service. More information on the cover sheet and submission service is available from: `https://student.eait.uq.edu.au/coversheets/`
  b. as an A4 size **PDF** (no other format) called exactly "Assignment2.pdf" via the course Blackboard site.

  The files should be submitted individually, not as a zipped archive. You can submit multiple times before the assignment deadline and only the last copy submitted will be retained for marking.

Submitted work should be neat and legible – you may be penalised for untidy or illegible work. For the programming part, you will be penalised for submitting files that are not compatible with the assignment requirements, in particular, your solution should be compatible with the supplied testing framework. As usual, the programming parts will be assessed for correctness as well as good programming style. They should be written in Java level 1.6.

Late submission. A penalty of 5% of the maximum mark for an assignment will be deducted for each day or part thereof late. Late submissions must be notified via email to the course coordinator. As we plan to hand back assignments about 1 week after submission, requests for extension will not be accepted more than one week late. Requests for extensions should be directed to the course co-ordinator and should be accompanied by suitable documentation (see Section 6.1 of the course profile for details). Personal hardware or computer failures are not grounds for extension. Assignments must be backed up on the university system.

School Policy on Student Misconduct. You are required to read and understand the School Statement on Misconduct, available at the School's website at:

`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

Question 1 (20 marks total)

Your coach has assigned you to fetch n flags, where flag i is placed f_i units away from the start line. You are to start your training session from the start line, and each time you fetch a flag you must bring it back to the start line before fetching a new flag. You may only carry one flag at a time.

To make your training extra challenging, your coach as asked that you also run the sum of the distances to the flags previously collected each time you return to the start line.

For example, if $n = 3$, $f_1 = 5$, $f_2 = 3$, $f_3 = 11$ There are $3! = 6$ possible orders in which you can fetch the flags.

Order	Distance travelled
$1, 2, 3$	$10 + (5 + 6) + (5 + 3 + 22) = 52$
$1, 3, 2$	$10 + (5 + 22) + (5 + 11 + 6) = 59$
$2, 1, 3$	$6 + (3 + 10) + (3 + 5 + 22) = 49$
$2, 3, 1$	$6 + (3 + 22) + (3 + 11 + 10) = 55$
$3, 1, 2$	$22 + (11 + 10) + (11 + 5 + 6) = 65$
$3, 2, 1$	$22 + (11 + 6) + (11 + 3 + 10) = 63$

a. (5 marks) Provide an efficient algorithm that allows you to collect all the flags and minimises the distance travelled. In the example above, the best order is $2, 1, 3$.

b. (15 marks) Prove that your algorithm in part (1) is optimal.

Question 2 (80 marks total)

In order to transform one source string of text x of length m to a target string y of length n, we can perform various transformation operations. Our goal is, given x and y, to produce a series of transformations that change x to y. We use a string buffer z to hold the result of the transformations. Initially, z is empty, and at termination, we should have $z = y$. We maintain current indices i into x and j into z, and the transformations are allowed to alter z and these indices. Initially, $i = j = 0$. We are required to examine every character in x during the transformation, which means that at the end of the sequence of transformation operations, we must have $i = m$. Because each character of y is written, we also have $j = n$.

There are six transformation operations:

- Copy a character from x to z by setting $z[j] = x[i]$ and then incrementing both i and j. This operation examines $x[i]$.
- Replace a character from x by another character ch, by setting $z[j] = ch$, and then incrementing both i and j. This operation examines $x[i]$.
- Delete a character from x by incrementing i but leaving j alone. This operation examines $x[i]$.
- Insert the character ch into z by setting $z[j] = ch$ and then incrementing j, but leaving i alone. This operation examines no characters of x.
- Swap (i.e., exchange) the next two characters by copying them from x to z but in the opposite order; we do so by setting $z[j] = x[i + 1]$ and $z[j + 1] = x[i]$ and then setting $i = i + 2$ and $j = j + 2$. This operation examines $x[i]$ and $x[i + 1]$.
- Kill the remainder of x by setting $i = m$. This operation examines all characters in x that have not yet been examined. If this operation is performed, it must be the final operation.

As an example, one way to transform the source string `recursive` to the target string `iterative` is to use the following sequence of transformations, where the underlined characters are $x[i]$ and $z[j]$ after the transformation:

Transformation	x	z
initial strings	`recursive`	`_`
INSERT `i`	`recursive`	`i_`
INSERT `t`	`recursive`	`it_`
SWAP	`recursive`	`iter_`
REPLACE by `a`	`recursive`	`itera_`
REPLACE by `t`	`recursive`	`iterat_`
DELETE	`recursive`	`iterat_`
DELETE	`recursive`	`iterat_`
COPY	`recursive`	`iterati_`
COPY	`recursive`	`iterativ_`
COPY	`recursive_`	`iterative_`

The method applyTransformations in StringTransformationTest applies a list of transformations to a string x to return the transformed string z.

Each of the transformation operations has an associated cost. The cost of an transformation depends on the specific application, but we assume that each transformation's cost is a constant that is known to us. We also assume that the individual costs of the copy and replace operations are less than the combined costs of the delete and insert operations; otherwise, the copy and replace operations would not be used. The cost of a given sequence of transformation operations is the sum of the costs of the individual operations in the sequence. For the table above, the cost of transforming `recursive` to `iterative` is

$$(3 \times cost(\text{COPY})) + (2 \times cost(\text{REPLACE})) + (2 \times cost(\text{DELETE})) + (2 \times cost(\text{INSERT})) + cost(\text{SWAP}).$$

Note that there are many other sequences of transformation operations that transform `recursive` to `iterative`. For example, a second transformation from `recursive` to `iterative` is: INSERT `i`, INSERT `t`, INSERT `e`, ... INSERT `e`, KILL, which has associated cost:

$$9 \times cost(\text{INSERT}) + cost(\text{KILL}).$$

Given two strings x and y and a given set of transformation costs, the cost from x to y is the cost of a least expensive operation sequence that transforms x to y. The assignment is to develop, analyze and apply recursive and dynamic programming solutions to this problem. The test framework defines a particular set of costs for the transformation operations (in TransCode in class TransElement) but your algorithms should work for any non-negative costs. **Hint:** This problem has some similarities with the longest-common-subsequence problem. It may help you to review that problem. In particular you may find it useful to define a similar kind of recurrence to that used in the solution to the longest-common-subsequence problem.

a. (25 marks) Complete the method stringTransformation in class RecursiveStringTransformation to provide a recursive algorithm to determine the minimum cost of transforming x to y. The recursive solution does NOT need to find the sequence of transformations that gives the minimum cost. Efficiency is not at all a concern for this part, so focus on an elegant solution.

b. (10 marks) It is expected that your recursive algorithm will not be polynomial-time in the worst case. Give an exponential asymptotic lower bound on the worst-case time complexity of your recursive algorithm. Give an argument explaining why the time-complexity is exponential in terms of a (lower bound) recurrence derived from your algorithm.

c. (25 marks) Develop a bottom-up dynamic programming solution to the problem (not memoised) by completing method stringTransformation in class DynamicStringTransformation. Your dynamic programming solution should run in polynomial time.

d. (5 marks) Extend your dynamic programming solution to calculate a sequence of transformations which achieves a minimal cost by completing the implementation of method getTransList in class DynamicStringTransformation.

e. (5 marks) Explain the data structures you use for part (d) and how the sequence of transformations is obtained from them. Explain the additional time and space cost of this extension. Either explain circumstances which would lead to the existence of more than one optimal solution or give an example showing this.

f. (5 marks) Formally analyse the time **and** space complexity of your dynamic programming solution for both parts (c) and (d).

g. (5 marks) Confirm your complexity analyses for both your recursive and dynamic programming algorithms by empirical testing. Provide a table of worst-case 'times' for a range of values for n and m, where the 'times' may be statement counts, execution times or some other similar metric (please specify what it is). A graph would be useful. Explain why your results confirm your theoretical analyses. (Due to the inefficiency of the recursive algorithm, it would be prudent to limit n and m appropriately: start with low values when testing. As your dynamic programming algorithm should be more efficient it should be safe to use inputs with larger values of n and m.)

For the supplied test framework, the input file must be formatted as follows:

Line 1 - contains x

Line 2 - contains y

Below is an example input file

```
recursive
iterative
```

The name of the input file is supplied as an argument to the test framework in a similar way to as used for assignment 1. Your programming solutions should be compatible with the test framework and use Java level 1.6.