Objective: To gain experience in String Problems using the algorithm design technique Dynamic Programming; to practise formulating a technical algorithm design starting from an informal specification of a problem; to practise implementing an abstract algorithm design as a working Java program.

***Please note:** This assignment specification aims to provide as complete a description of this assessment task as possible. However, as with any specification, there will always be things we should have said that we have left out and areas in which we could have done a better job of explanation. As a result, you are strongly encouraged to ask any questions of clarification you might have, either by raising them during a lecture or by posting them on the iLearn discussion forum devoted to this assignment.*

Splitting words

A certain string processing language allows the programmer to break a string into two pieces. It costs n units of time to break a string of n characters into two pieces, since this involves copying the old string. A programmer wants to break a string into many pieces, and the order in which the breaks are made can affect the total amount of time used. For example, suppose we wish to break a 20-character string (for example "abcdefghijklmnopqrst") after characters at indices 3, 8, and 10 to obtain for (sub)strings: "abcd", "efghi", "jk" and "lmnopqrst". If the breaks are made in left-right order, then the first break costs 20 units of time, the second break costs 16 units of time, and the third break costs 11 units of time, for a total of 47 steps. If the breaks are made in right-left order, the first break costs 20 units of time, the second break costs 11 units of time, and the third break costs 9 units of time, for a total of only 40 steps.

The problem is: given a string and a (strictly increasing) integer sequence indicating the indices of the breaks, compute the order of breaks which results in the least cost.

In this assignment you'll develop a **dynamic programming algorithm** to solve this problem. You will also be asked to answer some questions about your design.

More about the problem

In order to approach the design of a dynamic programming solution we must first compute the optimal cost for breaking the words. We will use a dynamic programming technique first to compute this optimal value, and then use the table used in the computation of that value to compute the order in which words should be broken.

Problem statement

Assume that we have a string *word* with indices {0…N-1}. We identify a contiguous substring of *word* by two indices indicating the first and last item of the substring.

Suppose that *word* = "elephant"

We write *word*[i,j] for the substring that starts at index i and ends at index j. For example *word*[2,4] consist of characters 'e' , 'p' and 'h', and *word*[5,5] consists of the single character 'a'. Of course the whole word is expressed in this way as *word*[0,7].

Now suppose we want to break the word after index 3; the result would be two two substrings; *word*[0,3] and *word*[4,7]. We assume that if a substring *word*[i,j] needs to be broken after index b then the two substrings produced will be *word*[i, b] and *word*[b+1,j] if $i <= b < j$.

The cost for performing the break is written as a function:

Cost(*word*[i,j], b) = (j-i+1), if $i <= b < j$

= 0, otherwise.

Now we can describe the problem statement formally as follows:

Input:

A *word* (of type String), and

An (array) list breakList= {b_0, b_2,…,b_k} with $b_0 < b_1 < b_2 < \ldots < b_k$

Output:

An (array) list of breaks in the order in which the breaks should occur so as to achieve the *minimal total cost* for breaking word according to breaks.

totalCost(*word*[i, j], breakList) = the optimal minimal cost of performing all the breaks in breakList.

To solve the problem above, you need to develop a *dynamic programming strategy* to compute the optimal cost by building a table of partial solutions for the optimal cost. As a second task you need to traverse the table in order to determine the optimal order of breaks.

Task 1: Sub problems and building the table [15% of total assignment marks]

Complete this task via the iLearn quiz for this assignment. Answer the 8 questions available in the quiz.

Questions 1—3 Concern identifying sub-problems and how to decompose the problem.

Questions 4—6 Concern decomposing the problem and identifying termination conditions.

Questions 7—8 Concern determining time complexity and simplifying the dynamic programming table.

Task 2: Implementing the table and computing the optimal total cost [50% of total assignment marks]

Download the assignment 2 bundle for this assignment from iLearn. You are asked to implement two methods in the class BreakSchedule.

Task 2 requires you to implement a dynamic program to compute the optimal total cost for a given *word* and breakList of breaks. You will need to download the java programming template which includes program stubs and JUnit tests which will be used in the automarking. For this task you must implement the following method:

```
// Precondition: word is a string and breakList is an array of integers in strictly increasing
// order with last element of breakList no more than the number of characters in word.

// Postcondition: Return the minimum total cost of breaking the string word into
//|breakList|+1 pieces, where the position of each each break is specified by the
//integers in breakList.

 int totalCost (String word, ArrayList<Integer> breakList)
```

Task 3: Compute the optimal order for breakages [35% of total assignment marks]

Task 3 requires you to traverse the table from Task 3 to compute the optimal total cost for a given word and list of breaks.

```
// Precondition: word is a string and breakList is an array of integers in strictly increasing
// order the last element of breakList is no more than than the number of
//  characters in word.

// Postcondition: Return the schedule of breaks so that word is broken according to the list of
// breaks specified in breakList which yields the minimum total cost

 ArrayList<Integer> breakSchedule (String word, ArrayList<Integer> breakList)
```

Format for Tasks 2&3

Submit a single file called BreakSchedule.java containing the implementations of all your programs.

Please make sure that you do not add a package name.
Please be sure to remove all syntax errors before you submit. Occasionally eclipse ignores these errors but the automarker does not.
Please make sure that the methods you implement are all public.
If you decide not to implement a method, please do not remove the method stub.
You must not change the definition of the *BreakSchedule* class. You may only add additional methods and/or datafields to the class. If you do add additional data fields you must make sure that your program compiles and that all your methods and data fields are public.

Your source code should be submitted via the Assignment 2 Tasks 2&3 link on the COMP3010 iLearn page.

Tasks 2&3 will be entirely marked by an automaker. This means that it is crucial that you submit files in the right format and with the right names. Unfortunately we are unable to mark individual assignments, but we will do a trial run of the automaker on **Tuesday 22nd October** so that you can check before the deadline that you have the right format. If you would like your programs to be trialled please make sure they are submitted **before Tuesday 22nd October** You will be able to resubmit as many times as you like before the deadline.

(Please note that you really must submit **before** 21st October to ensure that your files are trialled — I will retrieve any submissions early on Tuesday and aim to get feedback before the end of the day. Please note that auto marking is not really automatic as there is a human in the loop.)

What you must hand in and when

Task 1: Answer the questions in the iLearn quiz associated with this assignment.

Tasks 2 & 3: A single file called BreakSchedule.java according to the format outlined above. Submit your final versions by **Sunday 27 October**. (See above for details concerning trial auto marking.)

Mark allocation for assignment 2

This assignment is structured to allow you to decide how much effort you want to expend for the return in marks that you might hope for. You can choose whether to implement only the P level functionality, or to put in more effort and complete the full problem. So you can decide upfront whether you are shooting for a pass or a high distinction and know exactly how much work will be required to obtain that mark.

Here is what is roughly what is required to obtain marks in one of the performance bands for this assignment:

- **Pass** A successful specification and implementation of Task 2, together with a basic description and explanation of its correctness in the answers provided in Task 1.
- **Credit** The P level plus a successful implementation of Task 3 together with a more detailed appreciation of correctness via Task 1.
- **Distinction+** The CR level plus a detailed correctness and performance analysis of the programs via Task 1.

Detailed Marking Scheme

	P	CR	D/HD
Task 1 (3 marks)	Answers to questions 1 —3 **(Up to 5 quiz mark)**	Answers for questions 1 —6 **(Up to 10 quiz marks)**	Answers for questions 1 —8 (**Up to 15 marks)**
Implementation (Tasks 2 and/or 3) (27 marks)	A working implementation of Task 2. **(Up to 60 automarking marks)**	A working implementation of Tasks 2&3. **(Up to 65 automarking marks)**	A fully working implementation of Tasks 2&3. **(Up to 85 automarking marks)**

Late penalty

For all Tasks, late late work can be accepted, but please note the late assessment penalty.