Lab 5: Dynamic Programming

Due date: Friday, December 1, 11:59pm.

Lab Assignment

Assignment Preparation

This is an individual lab. The goal of this lab to give you an opportunity to implement a number of dynamic programming algorithms - both those covered in class and some that you have to design from scratch.

"Is It a Text" Problem

You are given a string of n characters $s[1..n]$ which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "thisisatest"). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function dict(i,j), which returns for a string s[i..j] the value **true** if it is a dictionary word, and the value **false** otherwise.

Goal. Develop and implement an algorithm that takes as input a string s, and given a pre-defined dictionary dict() returns `yes` if s can be represented as a sequence of dictionary words, and `no` otherwise.

Commentary. This is NOT an optimization problem. Rather, this is a **yes/no** problem. The answer we are looking for is either **yes**, if a given string consists of a sequence of valid words, or **no**, if it does not.

However, this problem can also be solved using dynamic programming.

Greedy Algorithms don't Work. The following simple example demonstrates that a greedy approach that can be characterized as follows:

Find the first substring $s[1..k]$ of $s[1..n]$, such that $dict(1,k)$ is **true**.
Then solve the problem recursively for $s[k+1..n]$

DOES NOT WORK!

Consider the following simple dictionary:

```
i
id
idol
red
```

Consider the string `"idolred"`. The greedy algorithm based on the rule above will break the string as follows: `i-dolred`. Since `"dolred"` is NOT in the dictionary, the greedy algorithm will output **false**.

At the same time, this string is a valid text sequence: `idol-red`.

Task

Each team shall develop and implement a dynamic programming algorithm for determining if a given string is a valid sequence of dictionary words.

To make grading straightforward, your solution shall satisfy the following conditions.

1. Create a Java or Python class `TextChecker`. Instances of this class shall have as their instance variables the text string which will be checked, and the list of dictionary words. The choice of the specific data structures for you to use is left up to you.
2. Implement `TextChecker.dict(int i, int j)` method, which will determine if the substring of an input string starting at character i and ending at character j is a dictionary word. This method shall encapsulate the search over the list of words stored in its dictionary instance variable. You can use a naïve implementation of this method or use existing functionality (e.g. Python's dictionaries) to help you.
3. Implement `TextChecker()` constructor which creates an empty instance.
4. Implement `TextChecker.setDictionary(dict)` method, which takes as input the data structure containing the entire dictionary and initializes it in the instance of the `TextChecker` class.
5. Implement `TextChecker.setString(s)` method, which sets the input string.
6. Implement `TextChecker.isText()` method, which runs your dynamic programming algorithm to determine if the input string is a sequence of words from the dictionary.

7. Implement `TextChecker.split()` method, which recovers the actual split of the input string into dictionary words *for input strings on which* `TextChecker.isText()` returns **`true`**. The `split()` method prints the split of the string into dictionary words.

8. Implement a test environment for the `TestChecker` class, in which an instance of the class is created, gets filled with the dictionary and a string, and the string is analyzed. Make sure your code has the ability to read the list of dictionary words from a file (similarly, make sure your code has the ability to read the input string from file).

Sample Domain

Your program shall work on any dictionary and any input text.

For testing purposes, please use the following dictionary

```
i
item
am
sam
ma
red
dare
rare
re
in
into
to
a
the
main
and
an
```

Some test strings you can use (all these are proper word sequences):

```
samiam
iamsam
iamiamiam
maiamsam
iteminred
themainitemandtherarereddare
itemandtheraredare
maintothered
redare
reddare
rareanddare
```

```
redandrareinmain
themaindare
```

And some strings that are not proper word sequences:

```
aandb
intotooandthe
themainsambutnotthelastone
stop
iamtired
redraredareinannitem
```

Edit Distance

The Edit Distance problem is stated as follows: given two strings x and y, find the minimal number of edits required to change x into y. The edits we consider for this assignment are:

- Insertion. A character y_j is inserted in position i of the string.
- Deletion. A character x_i is deleted from position i.
- Replacement. A character x_i is replaced with character y_j.

Alternatively, we can cast the edit distance problem as follows. An alignment between two strings x and y, is a mapping that pits each character x_i of x against either some character y_j of y or against an empty slot , and which respectively pits every character y_i of y against some character x_i of x or, such that if $i < j$ and both x_i and x_j are mapped to characters y_k and y_l respectively of y, then $k < l$ (i.e., the mapping follows the order of characters in the words).

Example. Consider two strings $x =$"STAND" and $y =$"SNEEZE". Below we show two possible alignments of these two strings:

`STAND_` `SNEEZE`	`STAND___` `S__NEEZE`

The cost of alignment is the number of positions in the alignment where the two characters do not match. For example, the cost of the first alignment above is 5, and the cost of the second alignment is 6. The edit distance between x and y is the smallest alignmnet cost.

Implementation. Your implementation of the Edit Distance algorithm shall take as input two strings, compute all necessary data structures and return the edit distance. A separate method shall be implemented to retrieve an alignmnet corre- sponding to the computed edit distnce. You can implement the edit distance algorithm by creating a Java or Python class (say `EditDistance`) whose instance variables include the tables used in the dynamic programming algorithm for finding the the edit distance. Then, you will need to implement two methods:

- `findEditDistance(String x, String y)`: this method takes two strings and runs the dynamic programming algorithm that dis- covers their edit distance by filling out the contents of the two tables. The method returns the edit distance.
- `void getAlignment()`: this method shall assume that `findEditDistance()` has successfully run. It will traverse the created tables to recover the alignment associated with the edit distance and print it out. To print out the alignment use the convention as above: print each string on a line of its own, and indicate insertions and deletions using the character.

(as usual, you may implement any other methods in this class)

Program for validation. Your submission shall include a program `EDTest.java`.

The program will take as input either one or two file names.

- If only one file name is provided, it shall contain two strings separated by a line break in it.
- If two file names are provided, each file shall contain one string. The string shall be assembled from the entire content of the file by removing all line-feed and carriage return characters from it.

The `EDTest.java` program shall read the input strings from either one or two files, shall run the edit distance algorithm on them, and shall output (print) the edit distance between the two strings and the corresponding alignment.

Program for experimental study. Conduct a short ex- perimental study using the edit distance implementation. You will measure the running time of your implementation of the edit distance algorithm (only the part that finds the edit distance, not the alignment recovery part), as it depends on the size of the input.

The study shall consist of two parts. In one part you will compare a string to itself. In the other part you will compare randomly generated strings. In both parts you will generate strings of various increasing sizes. For simplicity, use the alphabet

$\{A, T, C, G\}$

For the first part, for each string size you consider, generate 1000 random strings of that size and run the edit distance algorithm on each string compared to itself.

For the second part, for each string size you consider, generate 5000 pairs of random strings and run the edit distance algorithm on each pair.

In both studies, for each size considered, collect the average running time of the runs. In the second study, for each size, collect the average edit distance observed.

You can choose string sizes on which you run your algorithm in any way you want, provided that your study accomplishes the following:

- You report results for at least 10 different sizes.
- The study pushes the limits of productivity of your algorithm, i.e., at the 2-3 longest string sizes, your implementation starts taking noticable time to complete.

Name the program running your study `EDStudy.java` or `EDStudy.py`. Based on the results reported by the program, prepare a report that contains the following information:

- graphs/plots documenting the results of your study.
- your observations on how far your implementation goes (i.e., when your implementation starts taking considerable time to finish).
- your observations on whether it appears to be easier or harder to find the edit distance of a string with itself than it is to find the edit distance of two random strings, or whether there is no difference.
- a plot documenting the observed behavior of the average edit distance between two random strings on the size of the strings.
- your observations concerning the behavior of the average edit distance as the size of the strings increases.

Deliverables

Use `handin` to submit:

```
$ handin dekhtyar lab05 <files>
```

Submit all the code for both parts of the assignment, the `README` file with any comments on how to run your program, and the PDF of the report (name it `lab05-report.pdf`).

Good Luck!