Project 2: Dynamic programming

COT 4400, Summer 2017

Due July 16, 2017

1 Overview

For this project, you will develop an algorithm to identify the most likely sequence of typos a text underwent in the process of being typed. Designing and implementing this solution will require you to model the problem using dynamic programming, then understand and implement your model.

You are only allowed to consult the class slides, the textbook, the TAs, and the professor. In particular, you are not allowed to use the Internet. This is a group project. The only people you can work with on this project are your group members. This policy is strictly enforced. You may wish to review section 8.2 in the text, which describes a related problem.

In addition to the group submission, you will also evaluate your teammates' cooperation and contribution. These evaluations will form a major part of your grade on this project, so be sure that you respond to messages promptly, communicate effectively, and contribute substantially to your group's solution. Details for your team evaluations are in Section 5.2. You will submit the peer evaluations to another assignment on Canvas, labelled "Project 2 (individual)."

A word of warning: this project is team-based, but it is quite extensive and a nontrivial task. You are highly encouraged to start working on (and start asking questions about) this project early; teams who wait to start until the week before the due date may find themselves unable to complete it in time.

2 Problem Description

Your algorithm will accept a target string and a typo string, and it will compute the minimum cost for transforming the target string into the typo string, as well as a sequence of typos with this cost. The algorithm should recognize 4 different kinds of typos: insertions (adding extra characters), deletions (leaving out characters), substitutions (typing one character in place of another), and transpositions (swapping the order of two characters that appear next to one another).

2.1 Typo cost

Computing the cost for a particular typo, though, is a complex process, depending on the typo involved and where the keys for the related characters are located on the keyboard. The rules for computing the typo cost are summarized in the table below:

Typo	Condition	Cost
Inserting	Repeated character	1
	Space after key on bottom row	2
	Space after something else	6
	Character before a space	6
	Before or after another key on same hand	$d(k_1, k_2)$
	Before or after a key on opposite hand	5
Deleting	Repeated character	1
	Space	3
	Character after another key on same hand	2
	Character after space or key on different hand	6
	First character in string	6
Substituting	Space for anything or anything for space	6
	Key for another on same hand	$d(k_1, k_2)$
	Key for another on same finger, other hand	1
	Key for another on different finger, other hand	5
Transposing	Space with anything else	3
	Keys on different hands	1
	Keys on the same hand	2

Note that rules in the table above that reference a "hand" do not apply to spaces, which have separate rules. For ambiguous cases, you should report the *minimum* cost. This is especially notable when inserting a character between two others. For example, if you inserted a 'm' between an 'a' and an 'm', the cost would be 1 (for the repeated character), not 5 (for inserting after a character on the opposite hand).

The cost for two of the cases in this chart are given as $d(k_1, k_2)$, which represents the *keyboard distance* between these two keys. When computing keyboard distance, we assume that the typist is using a standard QWERTY keyboard where the keys can be arranged in 4 rows and 10 columns:

```
1234567890
qwertyuiop
asdfghjkl;
zxcvbnm,.
```

(You may assume that the strings will consist only of these characters and space.) The distance between a key in row r_1 and column c_1 and a key in row r_2 and column c_2 can be computed as:

$$d(k_1, k_2) = \max\{|r_2 - r_1|, |c_2 - c_1|\}.$$

When minimizing this cost function, we can assume that typos are generated in left-to-right order (i.e., the order in which the typist is typing). This rule exists to prevent the algorithm from generating dubious typos like substituting one character for another on the same finger of the other hand, inserting or deleting the next character, then undoing the substitution. It also has the happy side effect of making this problem an excellent candidate for dynamic programming. You should consider transpositions to affect the first character being transposed, deletions to affect the position of the deleted character, and insertions and substitutions to affect the location of the new character.

An example for how to score various typos in a string is given in section 4.4.

3 Project report

In your project report, you should include brief answers to 10 questions.

1. How you can break down a large problem instance into one or more smaller instances? Your answer should include how the solution to the original problem is constructed from the sub-problems.
2. What should the parameters be for your recursive function?
3. What recurrence can you use to model this problem using dynamic programming?
4. What are the base cases of this recurrence?
5. What data structure would you use to recognize repeated problems? You should describe both the abstract data structure, as well as its implementation.
6. Give pseudocode for a memoized dynamic programming algorithm to solve the problem. Your pseudocode should *not* describe how to compute the cost for each possible typo.
7. What is the *worst-case* time complexity of your memoized algorithm?
8. Give a set of nested loops that could be used in an *iterative* dynamic programming solution to this problem.
9. Can the space complexity of the iterative algorithm be improved relative to the memoized algorithm? Justify your answer.
10. Describe at least one advantage and disadvantage of the iterative algorithm, and indicate which implementation you believe is better overall.

4 Coding your solutions

In addition to the report, you should implement a *memoized* dynamic programming algorithm that can identify the most likely typos in the given string. Your code may be in C++ or Java, but it must compile and run in a Linux environment. If you are using C++ and compiling your code cannot be accomplished by the command

```
g++ -o typo *.cpp
```

you should include a Makefile that is capable of compiling the code via the `make` command.

If you choose to implement your code in Java, you should submit an executable jar file with your source. In either case, your source code may be split into any number of files.

Your code will not need to handle invalid input nor strings with more than 1000 characters.

4.1 Input format

Your program should read its input from the file `input.txt`, in the following format. The first line of the file has a positive integer x specifying the number of problem instances. The rest of the file contains x pairs of lines, each separated by a blank line. The first line of each pair is the target string for that problem instance, and the second, the typo string.

You may assume that both input strings consist only of spaces, lowercase letters, digits, and the punctuation symbols comma, period, and semicolon. You may also assume that neither string begins nor ends in a space; however, due to the properties of the cost function, you may append and/or prepend a space without changing the cost for transforming the input. (You should *not* change the locations of the typos, though.)

Each line should contain the number of Chomp games with exactly r rows and c columns, followed by the number of Chomp games with r or fewer rows and c or fewer columns. These two numbers should be separated by a space.

4.2 Output

Your program should write its output to the file `output.txt`. For each problem, your program should output the minimum cost to transform the target string into the typo string, as well as a sequence of typos with this cost. The output for multiple problem instances should be separated by a blank line.

The minimum cost should appear on a line by itself. Then, the sequence of typos should be printed, one typo per line, in left-to-right order of how they would affect the target string.

The table below gives the format specifications to describe the transformations. Do not change case, add extraneous spaces, or add any additional text to this specification—your output should match this format exactly.

Typo	Text
Insertion	Insert [CHAR] before [#]
Deletion	Delete [#]
Substitution	Substitute [CHAR] at [#]
Transposition	Transpose [#]-[#+1]

In this table, [CHAR] represents the character to insert or substitute, [#] represents the string index of the typo (starting with 0 for the first character of the string), and the second number for transposition should be one more than the first (e.g., "Transpose 6-7"). Note that you should use the same text for insertions regardless of how the cost is computed (i.e., you do not need to distinguish between "Insert d after 3" and "Insert d before 4").

Be aware that insertions and deletions will affect the offsets for future typos. For example, if you wish to insert a character before an 'a' in position 3 *and* transpose the 'a' with the next character, the transposition would apply to index 4.

4.3 Example

```
target:  the rain in spain stays mainly on the plain
typo:    teh driafna i pasin staya ksjnmly in th eplani
         012345678901234567890123456789012345678901234
```

Correct output:

```
27
Transpose 1-2
Insert d before 4
```

```
Transpose 6-7
Insert f before 8
Insert a before 10
Delete 13
Transpose 14-15
Transpose 15-16
Substitute a at 24
Substitute k at 26
Substitute s at 27
Substitute j at 28
Insert m before 30
Substitute i at 34
Transpose 39-40
Transpose 44-45
```

The transpositions in 14–16 could also be accomplished by an insertion and a deletion, for the same total cost.

4.4 Validing your cost function

The following table gives the cost for performing various typos on each of the first few characters in the example string in the previous section. The three scores for insertion represent the cost of inserting the given character after the character that precedes it, the cost of inserting the character before the character that follows it, and the cost of inserting the character in its position, as determined by the cost function, respectively. (This last value is the min of the previous two.) The deletion row represents the cost of deleting the character, while the substitution row represents the cost of substituting this character with the one that follows it. The cost given for transposition is the cost of swapping this character with the next.

Typo	t	h	e		r	a	i	n		i	n		s	p	a	i	n
Inserting after	*n/a*	5	5	6	6	3	5	2	2	6	2	2	6	5	5	5	2
Inserting before	5	5	6	6	3	5	2	6	6	2	6	6	5	5	5	2	*n/a*
Inserting	5	5	5	6	3	3	2	2	2	2	2	2	5	5	5	2	2
Deleting	6	2	2	3	6	2	2	2	3	6	2	3	6	2	2	2	2
Substituting	1	5	6	6	3	5	2	6	6	2	6	6	5	1	5	2	*n/a*
Transposing	1	1	3	3	2	1	2	3	3	2	3	3	1	1	1	2	*n/a*

Note that this example does not cover all possible cases for the cost function; you are responsible for fully testing your implementation.

5 Submission

Your submission for this project will be in two parts, the group submission and your individual peer evaluations.

5.1 Group submission

The submission for your group should be a zip archive containing 1) your report (described in Section 3) as a PDF document, and 2) your code (described in Section 4). If your code requires

more than a simple command to compile and run then you must also provide a Makefile and/or shell script. You should submit this zip archive to the "Project 2 (group)" assignment on Canvas.

Be aware that your project report and code will be checked for plagiarism.

5.2 Teamwork evaluation

The second part of your project grade will be determined by a peer evaluation. Your peer evaluation should be a text file that includes 1) the names of all of your teammates (including yourself), 2) the team member responsibilities, 3) whether or not your teammates were cooperative, 4) a numeric rating indicating the proportional amount of effort each of you put into the project, and 5) other issues we should be aware of when evaluating your and your teammates' relative contribution. The numeric ratings must be integers that sum to 30.

It's important that you be honest in your evaluation of your peers. In addition to letting your team members whether they do (or do not) need to work on their teamwork and communication skills, we will also evaluate your group submission in light of your team evaluations. For example, a team in which one member refused to contribute would be assessed differently than a team with three functioning members.

You should submit your peer evaluation to the "Project 2 (individual)" assignment on Canvas.

6 Grading

Report	**40 points**
Questions 1–5	2 each
Question 6	10 points
Questions 7–10	5 each
Code	**30 points**
Compiles	5
Computes the correct minimum cost	5
Computes a valid sequence of transformations with the given cost	15
Good coding style	5
Teamwork	**30 points**

Note that if your algorithm is inefficient, you may lose points for both your pseudocode and your submission. Also, in extreme cases, the teamwork portion of your grade may become negative or greater than 30. In particular, if you do not contribute to your group's solution at all, you can expect to receive a total grade of 0.