# Data Structures and Algorithms
Spring 2019
Assignment 2

IT University Innopolis

## Contents

# 1 About this homework

## 1.1 Coding part

For practical (coding) part you have to provide you program as a single Java class file that contains all necessary definitions and a `main` method.

The program should read from standard input and write to standard output unless mentioned otherwise.

### 1.1.1 Preliminary tests

For some coding parts a CodeForces or CodeTest system will be used for preliminary tests. You will have to specify a submission number in those systems as a proof that your solution is correct. Only correct solutions will be assessed further by TAs.

### 1.1.2 Code style and documentation

The source code should contain adequate internal documentation in the form of comments. Internal documentation should explain how the code has been tested, e.g. the various scenarios addressed in the test cases.

Do not forget to include your name in the internal documentation.

All important exceptions should be handled properly.

Bonus points might be awarded for elegant solutions, great documentation and the coverage of test cases. However, these bonus points will only be able to cancel the effects of penalties.

## 1.2 Theoretical part

Solutions to theoretical exercises have to be elaborate and clear. Write all solutions in a single document and make it a PDF for submitting.

Do not forget to include your name in the document.

## 1.3 Submission

You are required to submit your solutions to all parts via Moodle.

For this assignment you will need to submit: - Java class file(s) for the coding parts; - submission number or a link to submission in CodeForces as a proof of correctness; - a PDF file with solutions to theoretical parts.

Submit files as a single archive with your name and group number. E.g. `BS18-00_Ivanov_Ivan.zip`.

## 1.4 Plagiarism

Plagiarism will not be tolerated and a plagiarised solutions will be heavily penalised for all parties involved. Remember that you learn nothing when you copy someone else's work which defeats the purpose of the exercise!

You are allowed to collaborate on general ideas with other students as well as consult books and Internet resources. However, be sure to credit all the sources you use to make it clear what part of your solution comes from elsewhere.

## 2 Spell autocorrection (70 points)

In this homework assignment you are asked to implement a spelling autocorrection program, also known as a spell checker[1].

### 2.1 Misspellings (20 points)

Before you can autocorrect words, you need to be able to recognise if a given string of letters is a misspelling of a word. Moreover, if there are multiple misspelled word candidates you need to have a measure to compare these candidates.

In this exercise you are required to implement an algorithm that will determine if two words can be misspellings of each other and estimate by how much.

In this assignment we consider the following spelling mistakes:

- missed letter (e.g. *asignment* has a missing letter *s*);
- extra letter (e.g. *problemo* has an extra letter *o* at the end);
- different letter (e.g. *vutter* has letter *v* instead of *b*);
- swapping adjacent letters (e.g. *presetn* has *n* and *t* swapped).

A word can have multiple spelling mistakes. For example:

- *assynmment* is a misspelled *assignment* with *i* switched to *y*, missing *g* and extra *m* (3 mistakes total);
- *clot* can be considered a misspelling of *cat* with extra *l* and *a* changed to *o* (2 mistakes total);
- *persnetadg* as a misspelling of *percentage* has *c* changed to *s*, *e* and *n* swapped, extra *d* and missing *e* (4 mistakes total).

Two words are said to be $M$ *misspellings apart* if the minimal number of misspellings to get from one word to the other is $M$. For instance, *«cat»* and *«clot»* are 2 misspellings apart while *«giraffe»* and *«pirate»* are 3 misspellings apart. Every word is 0 misspellings apart from itself.

[1] in general a *checker* only checks and, possibly, makes suggestions, while an *autocorrector* also performs the substitutions

Implement a function `estimate(word1, word2)` taking two `String` words and returning an `int` value, specifying how far apart are those words are in terms of possible misspellings.

To verify your solution, write a program that reads $N$ pairs of words from standard input and returns the result of `estimate` for every pair.

For full grade, write the documentation, providing names of the data structures and algorithmic strategies you have used and mention (in detail) the reasons of your choice.

The input consists of an integer $N$ $(0 < N < 10\,000)$, followed by $N$ pairs of words. Every pair is located on a separate line. Words may consist of small English letters ($a$ to $z$).

Output should contain $N$ integer values $m_i$, where $m_i$ is how many misspellings are words in pair $i$ apart.

| Standard Input | Standard output |
|---|---|
| `10` | |
| `cat cat` | `0` |
| `cut cat` | `1` |
| `cat clot` | `2` |
| `giraffe pirate` | `3` |
| `assynmment assignment` | `3` |
| `persnetadg percentage` | `4` |

## 2.2 Correction suggestions (20 points)

Now that you can tell how likely one word is a misspelling of another, you can construct a list of correction suggestions for an unknown word.

A correction suggestion is a word from a dictionary of known words. The best correction suggestions are those that are closest to the corrected word by the number of misspellings. If a word is in dictionary (so is a known word) it is the best (and unique) correction for itself.

Implement a program that reads a list of dictionary words and a word to spell-check and outputs a sequence of possible best substitutions.

For full grade, write the documentation, providing names of the data structures and algorithmic strategies you have used and mention (in detail) the reasons of your choice.

The input consists of an integer $N$ ($0 < N < 10\,000$), followed by a line with $N$ different dictionary words and a separate line with a word that you need to spell-check.

Output should contain a sequence of best word corrections in lexicographic order.

| Standard Input | Standard output |
|---|---|
| `5`<br>`cat cut clot plot blog`<br>`bloat` | `blog plot` |

## 2.3 Text autocorrection (30 points)

In this final part implement a program that automatically corrects words in a text given a source text as a dictionary.

Source text contains all the correct known words. Each known word may appear multiple times in the source text. The number of appearences is called the *frequency*.

You need to automatically correct another text by autocorrecting every word in it with a word from the source text.

To automatically correct a word, your program should always pick the best correction substitution. If several substitutions are possible, it should pick the one with higher frequency in the source text. If there are multiple words with the same frequency, the program can choose any of those.

For full grade, write the documentation, providing names of the data structures and algorithmic strategies you have used and mention (in detail) the reasons of your choice.

The input consists of two lines of text of arbitrary length $N$ ($0 < N < 10\,000$). The first line is the source text and the second line is the text for autocorrection. Both texts might include punctuation marks and spaces. All words in both texts are lower case and in English alphabet ($a$ to $z$).

Output should be a single line with autocorrected text. All punctuation marks and spaces should be preserved.

| Standard Input | Standard output |
|---|---|
| `this is a great world!`<br>`i ate... a worm` | `is a... a world` |

## 2.4 Bonus features (+2 final course points)

Bonus features in this section may advance your course grade. Note that these give *extra final course points*, not regular bonus points[2].

### 2.4.1 Flexible misspelling rules (+1 final course point)

For extra course points adjust the `estimate(word1, word2)` function so that it can easily work with different kinds of misspellings. Specifically, it should accept an extra argument that will (somehow) specify possible misspellings together with their cost.

Document your design choices clearly.

### 2.4.2 Extra rules (+1 final course point)

Implement two extra misspelling rules with custom weights:

- consider phonetic substition as one misspelling (e.g. *f* for *ph*);
- consider some letter replacements as half-misspelling if the letter as next to each other on the QWERTY keyboard (e.g. *g* is next to *y*).

[2] regular bonus points can only cancel some penalties

## 3 Theoretical part (30 points)

### 3.1 Applications of Master Theorem (8 points)

The list bellow depicts recurrences for which you should provide the runtime $T(n)$ if the Master Theorem can be applied to solve it. Otherwise, you should justify the reason why the Master Theorem does not apply.

**Example I.** $T(n) = 3T(\frac{n}{2}) + n^2$.

**Answer.** $T(n) = \Theta(n^2)$. Case 3. Justification: $f(n) = \Omega(n^{log_2 3+\epsilon})$ for $\epsilon \approx 0.42$ (which is greater than 0) and $3f(\frac{n}{2}) \leq cf(n)$ for $c = \frac{3}{4}$ (which is smaller than 1) and all sufficiently large $n$.

**Example II.** $T(n) = 2^n T(\frac{n}{2}) + n^n$.

**Answer**. The Master Theorem does not apply since $a = 2^n$ is not constant.

Recurrencies:

a. $T(n) = 16T(\frac{n}{4}) + n$
b. $T(n) = T(\frac{n}{2}) + 2^n$
c. $T(n) = 2T(\frac{n}{2}) + \frac{n}{logn}$
d. $T(n) = 4T(\frac{n}{2}) + n^2$

### 3.2 Dynamic Programming (22 points)

You are asked design a dynamic programming algorithm for a given problem. You must also provide and prove the algorithm complexity of your solution to the problem.

There are 6 main steps for such:

- **Step 1. Define your sub-problem.** Describe in English what your sub-problem means, whether it look like $P(k)$ or $R(i, j)$ or anything else. For example, you might write *«Let $S(k)$ be the largest number of monkeys you can cram into k boxcars».*
- **Step 2. Present your recurrence.** Give a mathematical definition of your sub-problem in terms of "smaller" sub-problems.

- **Step 3. Prove that your recurrence is correct.** Usually a small paragraph. This is equivalent to arguing your inductive step in your proof of correctness.
- **Step 4. State your base cases.** Sometimes only one or two or three bases cases are needed, and sometimes you need only to state the asymptotic notation (let's say $O(n)$). The latter case typically comes up when dealing with multi-variate sub-problems.
- **Step 5. Present the algorithm.** This often involves initializing the base cases and then using your recurrence to solve all the remaining sub-problems. You want to ensure that by filling in your table of sub-problems in the correct order, you can compute all the required solutions. Finally, generate the desired solution. Often this is the solution to one of your sub-problems, but not always.
- **Step 6. Running time.** Provide the recurrence (in the form $T(n) = \ldots$) and prove its complexity.

**Problem.** A travel by train (in our case, a single line with no alternative paths) has $n$ stations on its way. Before starting your journey, you are given for each $1 \leq i \leq j \leq n$, the price of the ticket $f_{i,j}$ for travelling from station $i$ to station $j$. In our case, it is possible that $f_{1,5} = 11$ and $f_{1,6} = 3$. You start at the station 1 (let's say Moscow) and must end at the station $n$ (let's say Kazan). Your objective is to minimize the total cost. Provide a solution using dynamic programming. You are expected to follow and write about the 6 steps above.

*Hint: here is the answer for the Step 1 — say $m(i)$ be the cost for the best solution to travel from station $i$ to station $n$ for $1 \leq i \leq n$. The total cost then would be $m(1)$.*