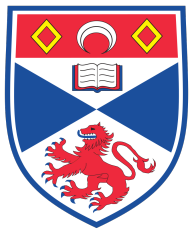

# University of St Andrews
# School of Computer Science

---

## CS1003 — Programming with Data

### P1— Text Processing

Deadline: **5 February 2021** Credits: **10%** of coursework mark

**MMS is the definitive source for deadline and credit details**

---

**You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. you must contact the lecturer regarding any queries well in advance of the deadline.**

This practical involves reading data from a file and using basic text processing techniques to solve a specified problem. You will need to decompose the problem into a number of methods as appropriate and classes if necessary. You will also need to test your solution carefully and write a report.

## Task

The task is to write a Java program to perform string similarity search among words stored in a text file. The code you are going to write is similar to code that is found in spell-checkers. Your program should accept two command line arguments, the first is the path of a text file (which contains a dictionary of commonly used English words) and the second is a query string. The program should then read the text in the file and split it into lines, where each line contains a single word. Then, calculate a similarity score between the query word and each word read from the file. Finally, print the closest match from the file (the word with the highest similarity score) to standard output, together with the similarity score. Place your main method in a class called *CS1003P1.java*. Some example runs are as follows.

Searching for the closest word to 'strawberry'

```
> java CS1003P1 ../data/words_alpha.txt strawberry
Result: strawberry
Score: 1.0
```

Searching for the closest word to 'stravberry'

```
> java CS1003P1 ../data/words_alpha.txt stravberry
Result: strawberry
Score: 0.6923077
```

Searching for the closest word to 'ztravberry'

```
> java CS1003P1 ../data/words_alpha.txt ztravberry
Result: strawberry
Score: 0.46666667
```

### String similarity

There are several ways of calculating a similarity score between strings, in this practical we ask you to use a *Jaccard index on character bigrams*. This might sound scary at first, but don't worry! We will now define what we mean and give an example.

### Jaccard index

The Jaccard index is a similarity measure between sets of objects. It is calculated by dividing the size of the intersection of the two sets by the size of the union of the same two sets. If the two sets are very similar, the value of the Jaccard index will be close to 1 (if the two sets are identical it will be exactly 1). On the other hand, if the two sets are very dissimilar, the value of the Jaccard index will be close to 0 (if the two sets are disjoint it will be exactly 0). Try drawing a few simple Venn diagrams to convince yourselves of this! Wikipedia has a good article on the Jaccard index as well: https://en.wikipedia.org/wiki/Jaccard_index

### Character bigrams

A character bigram is a sequence of two consecutive characters in a string. Bigrams have applications in several areas of text processing like linguistics, cryptography, speech recognition, and text search. In this practical, we will calculate the Jaccard index on sets of bigrams for calculating a similarity score between strings. Following is an example of the set of bigrams for the string 'cocoa': 'co', 'oc', 'oa'. Notice that since we generate a set of bigrams, we avoid repeating 'co' twice.

Your program should contain a method to create a set of bigrams for a given string.

### Top and tail

Adding special characters to the start and the end of a string before calculating the set of bigrams can improve string similarity search. This is often done by adding a '^' character to the beginning and a '$' character to the end of the string. On the same example, 'cocoa', we first add the special characters to either side and get to '^cocoa$'. The set of bigrams becomes: '^c', 'co', 'oc', 'oa', 'a$'.

## Suggested steps

- Download the text file *words_alpha.txt* [1] from StudRes and save it to a known location. You should not submit this file as part of your submission.
- Create a Java class called *CS1003P1* and write a program that is able to read the data stored in this text file line by line. In order to check that this works, print each line to standard output. See method **`readAllLines`** from the **`Files`** class.
- Write a method to calculate character bigrams of a given string and store them in a set. See the **`Set`** and **`HashSet`** classes and the **`add`** method that they implement. You may test your method with the string 'cocoa', the output should match the given output above.
- Implement top-and-tail as described above and update your bigram calculation to use this functionality.
- Implement the Jaccard index calculation. Which two sets will you calculate the Jaccard index on? We suggest that you use the **`retainAll`** method for implementing set intersection and the **`addAll`** method for implementing set union. The **`size`** method returns the size of a set. If you

[1] Source: https://github.com/dwyl/english-words

calculate the Jaccard index between the set {"1","2","3"} and the set {"1","2","4"} (where the size of the intersection is 2 and the size of the union is 4) you should get $2/4 = 0.5$ as the result.

- Combining character bigrams, top-and-tail and Jaccard index you now have a way of calculating a similarity score between two strings. Use this to calculate the score between the query word and each word from the file in a loop. Keep track of the best score (and the word that has the best score!) for reporting at the end.
- We suggest that you print the best matching string and the corresponding similarity score as you iterate through the dictionary during development. This can help you with testing your program.

## Auto-checker and Testing

This assignment makes use of the School's automated checker 'stacscheck'. You should therefore ensure that your program can be tested using the auto-checker. It should help you see how well your program performs on the tests we have made public and will hopefully give you an insight into any issues prior to submission. The automated checking system is simple to run from the command line in your *CS1003-P1* directory:

```
stacscheck /cs/studres/CS1003/Practicals/P1/Tests
```

Make sure to type the command exactly – occasionally copying and pasting from the PDF specification will not work correctly. If you are struggling to get it working, ask a demonstrator.

The automated checking system will only check the basic operation of your program. It is up to you to provide evidence that you have thoroughly tested your program.

## Submission

### Report

Your report **must** be structured as follows:

- **Overview:** Give a short overview of the practical: what were you asked to do, and what did you achieve? Clearly list which parts you have completed, and to what extent.
- **Design:** Describe the design of your program. Justify the decisions you made. In particular, describe the classes you chose, the methods they contain, a brief explanation of why you designed your solution in the way that you did, and any interesting features of your Java implementation.
- **Testing:** Describe how you tested your program. In particular, describe how you designed different tests. Your report should include the output from a number of test runs to demonstrate that your program satisfies the specification. Please note that simply reporting the result of stacscheck is not enough; you should do further testing and explain in the report how you convinced *yourself* that your program works correctly.
- **Evaluation:** Evaluate the success of your program against what you were asked to do.
- **Conclusion:** Conclude by summarising what you achieved, what you found difficult, and what you would like to do given more time.

Don't forget to add a header including your matriculation number, the name of your tutor and the date.

### Upload

Package up your `CS1003-P1` folder and a PDF copy of your report into a zip file as in previous weeks, and submit it using MMS, in the slot for Practical P1. After doing this, **it is important to verify that you have uploaded your submission correctly by downloading it from MMS. You should also double check that you have uploaded your work to the correct slot**. You can then run `stacscheck` directly on your zip file to make sure that your code still passes stacscheck. For example, if your file is called **`190000000-P1.zip`**, save it to your `Downloads` directory and run:

```
cd ~/Downloads
stacscheck --archive 190000000-P1.zip /cs/studres/CS1003/Practicals/P1/Tests
```

## Rubric

### Marking

| | |
|---|---|
| **1-6** | Very little evidence of work, software which does not compile or run, or crashes before doing any useful work. You should seek help from your tutor immediately. |
| **7-10** | An acceptable attempt to complete the main task with serious problems such as not compiling, or crashing often during execution. |
| **11-13** | A competent attempt to complete the main task. Serious weaknesses such as using wrong data types, poor code design, weak testing, or a weak report riddled with mistakes. |
| **14-16** | A good attempt to complete the main task together with good code design, testing and a report. |
| **17-18** | Evidence of an excellent submission with no serious defects, good testing, accompanied by an excellent report. |
| **19-20** | An exceptional submission. A correct implementation of the main task, extensive testing, accompanied by an excellent report. In addition it goes beyond the basic specification in a way that demonstrates use of concepts covered in class and other concepts discovered through self-learning. |

See also the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

### Lateness penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

### Good academic practice

The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice/

## Going Further

Here are some additional questions and pointers for the interested student.

- Character bigrams are a special case. If you are interested look into character n-grams which are sets of n-characters where n is not necessarily 2 (as in bigrams).
- N-grams can be constructed at the word level instead of at the character level. Look into applications of word-level n-grams and think about the use cases of character-level vs word-level n-grams.
- There are many other string similarity methods! See https://en.wikipedia.org/wiki/String_metric as a starting point.
- Think about use cases for string similarity methods.