

---

# FIT2004 S2/2018: Assignment 1

Total marks: 5

**DEADLINE:** Friday, 10-Aug-2018 23:55:00 AEST

**LATE PENALTY:** 25 % per day

**CLASS:** You may be interviewed during your week 04 tutorial by the tutor who may ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. You will also be given feedback on your assignment. Failing to attend will result in you being awarded zero mark for the assignment (unless you have an approved special consideration). It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the time and space complexity of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! “Forgetting” is not an acceptable explanation for lack of understanding. Markers are not obliged to mark programs that do not run or that crash.

**SUBMISSION REQUIREMENT:** You will need to submit a zipped file containing your Python program (named `scrabble.py`) as well as a PDF file briefly describing your solution and its space and time complexity. The PDF file must give an outline of your solution (e.g., a high level idea of how did you solve it) and the **worst-case** space and time complexity of your solution. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline.

**Important:** The assignments will be checked for plagiarism using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. “Helping” others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.

## Scrabble Helper

Your friend Alice is sad – she lost yet another game of Scrabble to her friend Cindy. However, she is not ready to give up and has come up with a plan. She has downloaded a dictionary from the internet and has cleaned it such that it only contains words (and not their definitions or other information). Her plan was to write a program such that it takes some letters as an input and returns every word in the dictionary that can be made by using *all* of the input letters. However, her algorithm is too slow and requires a long time to return the results. She knows that you are taking FIT2004 which focuses on developing efficient algorithms and data structures. She really hopes that you can help her and has come to seek your help.

Alice: “I need a favor. I have a text file which contains over 80,000 English words. Can you please help me write a program that takes some letters as input and returns every word from the text file that can be made by using *all* of the input letters?”

You: “Sure, it looks easy enough”.

Alice: “Actually, I have written a brute-force algorithm myself but it is too slow :( I know you are taking FIT2004 and was hoping that you can help me.”

You: “Hmmm, I don’t know much details of your algorithm but I think we could use anagrams and a linear sorting algorithm to solve this problem efficiently. We will be taught the linear sorting algorithms in Week 2, the next week!!!”

Alice: “Greattttt!!! Please do not miss the lecture then! I am counting on you!!!”

You: “Don’t worry! When do you need the program?”

Alice: “I have a rematch with Cindy on Saturday. It will be great if you can send me the program by Friday midnight (10th August).”

You: “Sure, will do!”

Alice: “ummmm... actually, I was wondering if you can also handle wildcard? A wildcard tile can be used as *any* letter of the player’s choice. For example, if the input letters are **alpp** and I have one wildcard tile, it can be used to make the word **apple** where the wildcard tile is used as an **e**. The other words in the dictionary that can be made using **alpp** and a wildcard tile are **appel** (wildcard used as **e**), **palpi** (wildcard used as **i**), **lapps** (wildcard used as **s**) and **pupal** (wildcard used as **u**).”

You: “But the word **pale** can also be made using **e** as a wildcard, no?”

Alice: “Yes! But I am only interested in the words that use *all* of the input letters *and* the wildcard tile. You can assume that there will only be one wildcard tile.”

You: “Hmmm... I see. I think I can do that.”

Alice: “Yuppie....you are a true friend! Thank you so much!”, and, in her excitement, she challenges Cindy in her imagination, “Cindy! Get ready for the revenge game!!! I will be equipped with superpowers developed by a super friend! Beat me if you can!!!”.

She is full of excitement and you don’t want to disappoint her. As soon as she leaves, you start working on the program.

## Input

The input file named `Dictionary.txt` (available on moodle) consists of 80,098 words not necessarily in a sorted order. Each word consists of only lowercase English characters, i.e., there is no uppercase letter, whitespace, and hyphen “-” etc. Below are first few words in the dictionary.

```
abaca
abacinate
abacination
abaciscus
abacist
aback
```

**Input size.** Let  $N$  be the total number of words in the dictionary and  $M$  be the maximum number of characters in a word in the dictionary. The total input size is  $O(MN)$ .

Your program must be able to do several tasks as described below.

## Task 1: Largest group of anagrams

Out of curiosity, you want to find the largest group of anagrams. Specifically, two words are called anagrams if they consist of exactly the same letters but in different order. For example,

the words `leap` and `pale` are anagrams. A group of anagrams is a group of words such that all words in the groups have exactly the same letters but in different order. For example, the words `leap`, `pale`, and `peal` form a group of anagrams. Size of this group is 3 where size of a group is defined as the number of words it contains. Your task is to find the largest group of anagrams (in terms of size). In the file `Dictionary.txt`, the largest group of anagrams is `astel`, `laste`, `lates`, `least`, `satle`, `slate`, `stale`, `steal`, `stela`, `tales` which consists of 10 words all consisting of same letters but in different order.

In this task, you must write a function named `largestAnagram("Dictionary.txt")` which computes and prints the largest group of anagrams in the file `Dictionary.txt`. If there is a tie (two or more largest groups have the same size), you can choose to print any group of your choice. **However, the words in the group must be printed in alphabetical order.** Make sure to decompose and comment your code properly.

**Complexity Requirement.** Your program must print the largest group of anagrams in  $O(MN)$  worst-case time complexity. Note that this requires sorting in linear time which we will cover during week 2 lecture. ~~This is an optimal algorithm because reading the input itself requires  $O(MN)$ , i.e., the lower bound complexity is  $O(MN)$ . We will touch upon lower bound complexity in week 3 lecture<sup>1</sup>.~~ The space complexity of your program must also be  $O(MN)$ .

## Task 2: Scrabble words finder

After your program has printed the largest group of anagrams, you will ask the user to enter a string of letters called `query` string. Your program must then print *all* words in the dictionary that can be made using *all* letters in the query string. For example, if the query string is `alppe`, the output will consist of the words `appel`, and `apple`. **The words in the output must be printed in alphabetical order.** Note that `pale` is not in the output because it does not contain *all* of the letters in the query string. Similarly, `appeal` is not in the output because it cannot be made using the input letters, e.g., the input contains only one `a`. You must write a function called `getScrabbleWords` which prints the words that can be made using *all* letters in the query string (printed in alphabetical order).

Your program must continue asking the user for a query string until the user enters `***` in which case your program must quit.

**Complexity Requirement.** For a query string consisting of  $k$  characters, `getScrabbleWords` must return the output in  $O(k \log N + W)$  in the worst-case where  $W$  is the output size (i.e., the number of characters in the output). Note that string comparison (e.g., `str1 < str2` or `str1 == str2`) takes  $O(k)$  in the worst-case where  $k$  is the number of characters in the smaller string.

## Task 3: Query with wildcard

This task extends Task 2 by considering one wildcard tile. Specifically, it returns all the words in the dictionary that can be made using *all* letters in the query string *and* one wildcard tile which can be used as *any* letter. For example, if the query string is `alppe`, the output will consist of the words `appeal`, `dapple`, `palped`, `lapper`, `rappel`, `lappet`, and `papule` where the wildcard is used as `a`, `d`, `d`, `r`, `t` and `u`, respectively. **The words in your output must be printed in alphabetical order** (see the sample output shown later). In this task, you must

---

<sup>1</sup>see <https://moodle.vle.monash.edu/mod/forum/discuss.php?d=1405041> for details of why this has been crossed out.

write a function called `getWildcardWords` to print all the words that can be made using the letters in the query and the wildcard tile (printed in alphabetical order).

**Complexity Requirement.** For each query string of size  $k$ , `getWildcardWords` must return the output in  $O(k \log N + W)$  in the worst-case where  $W$  is the output size. You will need to carefully analyze your program to see if the complexity contains constants that can be ignored.

## Output

Below is a sample execution of the program `scrabble.py` which reads from the provided file `Dictionary.txt`. You can assume that the query string will consist of only lowercase characters and will only contain English letters. Note that, the words in the output of each task are printed in alphabetical order.

```
The largest group of anagrams: astel laste latest least satle slate stale
steal stela tales

Enter the query string: alppe

Words without using a wildcard: appel apple
Words using a wildcard: appeal dapple lapper lappet palped papule rappel

Enter the query string: ablet

Words without using a wildcard: bleat table
Words using a wildcard: albeit albite ballet balter batlet battel battle
batule belate cablet gablet labent labret oblate stable tablet

Enter the query string: algorithm

Words without using a wildcard: logarithm
Words using a wildcard: lithomarge

Enter the query string: acre

Words without using a wildcard: care race
Words using a wildcard: acerb aced arace areca brace caber cader cadre caper
caret carex carse carte carve cater cedar ceria chare chear clare clear craie
crake crane crape crare crase crate crave creak cream creat erica eruca farce
grace macer nacre ocrea pacer perca racer racle reach react sacre scare serac
trace varec

Enter the query string: zorts

Words without using a wildcard:
Words using a wildcard: zoster

Enter the query string: traesoz
```

```
Words without using a wildcard: zostera
```

```
Words using a wildcard:
```

```
Enter the query string: ***
```

**Note:** Your program will be tested on a different dictionary file, i.e., the largest group of anagram may be different. You must strictly follow the output format above (e.g., do not add un-necessary words or symbols in your output). Your output will be tested by an automated program and failing to follow the output format defined above may result in your output being marked incorrect.

## Things to note

If you decide to use in-built Python functions and structures, you must carefully consider their worst-case complexities. For example, inserting/retrieving an element in Python dictionary (which uses hashing) takes  $O(N)$  in worst-case. This is because, as we will later see in week 5, although hashing is quite efficient in practice its worst-case complexity for insertion/retrieval is still  $O(N)$ . It may not always be easy to determine the worst-case complexities of all in-built functions and structures. Therefore, it is recommended that you use only the basic data structures (such as Python lists). This assignment can be easily completed using the basic data structures without using any advanced in-built functions.

## Marking Principles

The assignments are marked based on the following marking principles.

- Your program will be tested on a variety of different test cases and you will lose marks if your implementation fails to produce correct results for any of these test cases.
- Your implementation must produce correct output for **at least** the sample test cases provided in the assignment sheet. Otherwise, you will lose almost all marks (even if bugs are minor). Therefore, it is critical that your program produces the correct output for at least the sample test cases.
- Naive algorithms will receive zero mark, i.e., there are no marks for getting the correct output if your algorithm is naive.
- If your algorithm fails to meet the space and complexity requirements but the complexities are reasonably close to the required complexities, you will get partial marks (depending on how good is your overall submission).
- You are also required to demonstrate understanding of your solution (e.g., correctness of complexity analysis). Make sure the description of your algorithm is clear and the complexity analysis included in the PDF file is correct.

```
--oOo--
```

```
END
```

```
--oOo--
```

## Change log

[30-07-2018 12:10 PM] - Added late penalty (25% per day) on the first page

[30-07-2018 12:10 PM] - Added the section "Marking Principles" on page 5

[31-07-2018 04:59 PM] - Crossed-out the claim that  $O(MN)$  is lower bound (page 3)