Grep

COMP0015 Term 1 Coursework 5 – 20% of the module

This document explains the arrangements for the coursework. You will create an application to query a set of files and print the results – a search engine of sorts. We use search engines all day everyday but how do they work? In short, search engines use programs called web-crawlers to create an index of all the web pages in the world. When you type a query into a search engine the components of the query are checked against these pre-built indexes. In this coursework you will build a simple search engine which builds indexes for a given set of files on your computer. This is basically how a search engine works but the difference is in the scale of operation. A utility called grep for searching the text files has been available on computers running Unix operating systems since the 1970s and that is why this coursework is called 'grep'. You don't need to know about grep to complete this coursework.

Deadline

9.00am 30th November 2020.

Aim

At a high-level, the aim of this coursework is for you to demonstrate that you can use dictionaries, sets, lists and files. You are given some starter code, you do not need to change this code. You are required to complete some of the functions as described in this document.

Your Assignment

You are provided with some starter code and some text files which you must download and save before starting the assignment. These materials are also available on the Codio platform.

Functions

The skeleton code contains the empty functions: `common_elements(), build_index()` and `search()` you must complete them according to the instructions here and the specification in the docstrings (comments) at the start of the functions.

(a) Function `common_elements(lis_1, lis_2)`

`lis_1, lis_2` are simple one-dimensional lists. This function returns a list which contains the elements common to both lists. You will use this function later on in your program. Here are some examples illustrating usage and output.

`common_elements([‘a’, ‘b’, ‘c’], [‘a’, ‘p’, ‘c’])` will return list `[‘a’, ‘c’]`

`common_elements([‘a’, ‘b’, ‘c’], [‘x’, ‘y’, ‘z’])` will return list `[]`

`common_elements([‘a’, ‘b’, ‘a’], [‘a’, ‘a’, ‘a’])` will return list `[‘a’]`

Note: in any list returned, the elements may appear in a different order.

Hint: use sets and find which elements are common.

There is some code in your `main()` method for running these tests.

(b) Function `build_index(file_list, index, title_index)`

In this function you will use the list of file names in the variable `file_list`. You will create two indexes `index` and `title_index` which are both dictionaries. For the most part, the small files distributed with the coursework contain news articles from the BBC about sporting events. A list containing the file names of some of these files would be:

```
[‘001.txt’, ‘002.txt’, ‘003.txt’, 004.txt’, ‘005.txt’, ‘006.txt’, ‘007.txt’, 008.txt’, ‘009.txt’, ‘010.txt’]
```

Stage 1

In the first part of this problem, you will build an index containing all the words in the file. This makes sense, when we query a word, we will look up the word in our index and we’ll know in which files the word appears.

Complete these tasks:

1. You will open and read the files in the list and create a list of words in the file with no duplicates.
2. The words must be ‘cleaned’ and you should produce a list of cleaned words. You will clean a word by removing the punctuation from either end of the word and converting the word to lower case. You can remove the punctuation from either end of a word by using the string function `strip()` like this:

    ```
    word.strip(string.punctuation)
    ```

    `string.punctuation` is a constant from the string module which is imported at the top of your program. Note: strip will not remove punctuation in the middle of a word and you are not required to do so.

    `“-14.9.”.strip(string.punctuation)` will return the string `“14.9”`. This is acceptable.

    `“hi-top!”.strip(string.punctuation)` will return the string `“hi-top”`. This is acceptable.

    `“,’:-!”.strip(string.punctuation)` will return the string “”. Empty strings must not be added to the list of cleaned words.

3. After you have cleaned each word, you should add it to the `index` dictionary as a key along with the filename in which the word appears. Filenames will be added to a list. Here is an example of two files and the corresponding dictionary `index`:

'sentimental_1.txt'

Sentimental Journey by Duke Ellington

Gonna take a sentimental journey
Gonna set my heart at ease

'sentimental_2.txt'

In a Sentimental Mood by Duke Ellington and John Coltrane

In A Sentimental Mood
I can see the stars come thru my room

Contents of the dictionary `index`:

```
{
'at': ['sentimental_1.txt'],
'gonna': ['sentimental_1.txt'],
'ease': ['sentimental_1.txt'],
'a': ['sentimental_1.txt', 'sentimental_2.txt'],
'sentimental': ['sentimental_1.txt', 'sentimental_2.txt'],
'by': ['sentimental_1.txt', 'sentimental_2.txt'],
'take': ['sentimental_1.txt'],
'journey': ['sentimental_1.txt'],
'set': ['sentimental_1.txt'],
'my': ['sentimental_1.txt', 'sentimental_2.txt'],
'ellington': ['sentimental_1.txt', 'sentimental_2.txt'],
'duke': ['sentimental_1.txt', 'sentimental_2.txt'],
'heart': ['sentimental_1.txt'],
'stars':['sentimental_2.txt'],
'john': ['sentimental_2.txt'],
'thru': ['sentimental_2.txt'],
'coltrane': ['sentimental_2.txt'],
'the': ['sentimental_2.txt'],
'and': ['sentimental_2.txt'],
'room': ['sentimental_2.txt'],
'can': ['sentimental_2.txt'],
'come': ['sentimental_2.txt'],
'i': ['sentimental_2.txt'],
'see': ['sentimental_2.txt'],
'mood': ['sentimental_2.txt'],
'in': ['sentimental_2.txt']
}
```

Stage 2:

You are now ready to continue and build a dictionary of file titles called `title_index`. Why is this important? Well, when returning search results, it will be nice to return the name of the file that the word appeared in and the title of the file. All the files we will use in this assignment have titles.

In your code, you will do this at the same time as you are building the index of words. Here is an example illustrating what your code will do. The function call:

```
build_index(['001.txt', '002.txt', '003.txt'], index, title_index)
```

will create a dictionary called `title_index` containing the following entries:

```
{
        '001.txt': 'Man Utd stroll to Cup win',
        '002.txt': 'Van Nistelrooy set to return',
        '003.txt': 'Moyes U-turn on Beattie dismissal'
}
```

Important notes:

- the '\n' has been stripped from the end of the title.
- You must add the words in the title to the dictionary of words in the file!!!!!!!

Stage 3:

You've been testing the code you've written so far step-by-step haven't you? Well now you should test your code on small files. In the main method, you will find this code:

```
# Test 2: test with small files
build_index(['sentimental_1.txt', 'sentimental_2.txt', 'sentimental_3.txt'], index, title_index)
pretty_print(index, title_index)
```

Function `pretty_print(index, title_index)` is provided for you in the starter code. Your output should look like the output below although the elements in your list may appear in a different order.

```
Index:

{'duke': ['sentimental_1.txt', 'sentimental_2.txt'], 'ellington': ['sentimental_1.txt', 'sentimental_2.txt'], 'sentimental': ['sentimental_1.txt',
'sentimental_2.txt', 'sentimental_3.txt'], 'set': ['sentimental_1.txt'], 'gonna': ['sentimental_1.txt'], 'a': ['sentimental_1.txt',
'sentimental_2.txt'], 'ease': ['sentimental_1.txt'], 'at': ['sentimental_1.txt'], 'heart': ['sentimental_1.txt'], 'journey': ['sentimental_1.txt'], 'my':
['sentimental_1.txt', 'sentimental_2.txt'], 'take': ['sentimental_1.txt'], 'by': ['sentimental_1.txt', 'sentimental_2.txt', 'sentimental_3.txt'],
'thru': ['sentimental_2.txt'], 'see': ['sentimental_2.txt'], 'mood': ['sentimental_2.txt'], 'john': ['sentimental_2.txt'], 'room':
['sentimental_2.txt'], 'come': ['sentimental_2.txt'], 'coltrane': ['sentimental_2.txt'], 'can': ['sentimental_2.txt'], 'i': ['sentimental_2.txt',
'sentimental_3.txt'], 'and': ['sentimental_2.txt'], 'in': ['sentimental_2.txt'], 'stars': ['sentimental_2.txt'], 'the': ['sentimental_2.txt'], 'do':
['sentimental_3.txt'], 'nat': ['sentimental_3.txt'], 'love': ['sentimental_3.txt'], 'believe': ['sentimental_3.txt'], 'king': ['sentimental_3.txt'], 'me':
['sentimental_3.txt'], 'for': ['sentimental_3.txt'], 'reasons': ['sentimental_3.txt'], 'cole': ['sentimental_3.txt'], 'you': ['sentimental_3.txt'],
'hope': ['sentimental_3.txt']}

File names and titles:

{'sentimental_1.txt': 'Sentimental Journey by Duke Ellington', 'sentimental_2.txt': 'In a Sentimental Mood by Duke Ellington and John
Coltrane', 'sentimental_3.txt': 'Sentimental Reasons by Nat King Cole'}
```

Try your program on a larger set of files to make sure that it doesn't break. You will find these lines in your `main()` function.

```
 # Test 3: test with bbc sport news files
  build_index(get_filenames('bbc_football'), index, titles)
  pretty_print(index, titles)
```

Uncomment the code and run the program.

(c) Function `search(index, query)`

Now you are ready to implement the `search()` function which searches your index. The parameter `query` is a string containing the words to search for. For the purposes of this coursework you may assume that query is lowercase and will never contain punctuation characters. You may not assume that the query contains unique words.

Valid contents for a string `query` are:

```
‘chelsea cup’
‘manchester united win’
‘arsenal arsenal arsenal’
```

The function must return a list of the files that contain **all** words in the query. You can use the function `common_elements()` which you wrote earlier to achieve this.

As you process each word in the query, you should store the file list from the index dictionary in a variable and then use `common_elements()` to find the shared file names with the last word you processed.

There is a high-level description of one algorithm you may use to do this in the appendices. You may use a different algorithm if you wish. You are responsible for testing the function you write properly.

You should test your function with the small files, sentimental1.txt, sentimental2.txt and sentimental3.txt. There are some lines of code in the main() function labelled *# Test 4: test with small files* to do this. Here are the results you can expect with the given queries:

Query string	Search results printed in main()
'sentimental'	sentimental: ['sentimental_2.txt', 'sentimental_1.txt', 'sentimental_3.txt']
'coltrane'	coltrane: ['sentimental_2.txt']
'ellington'	ellington: ['sentimental_2.txt', 'sentimental_1.txt']
'sentimental journey'	sentimental journey: ['sentimental_1.txt']
'not_in_files'	not_in_files []
'long journey'	long journey []

Putting it all together

The function menu() is provided for you so that you can check the behaviour of your program on the BBC football news files in the folder bbc_football. Try it out, uncomment Test 5 in main().

Here is a sample of the output. Note: your results may appear in a different order depending on how you wrote your code. The text in **bold** shows what was typed in.

```
Enter a search query, (empty to finish): southend
Results:  southend
No results
Enter a search query, (empty to finish): watford
Results:  watford
File:  bbc_football\111.txt Title:  Yeading face Newcastle in FA Cup
File:  bbc_football\098.txt Title:  Redknapp's Saints face Pompey tie
Enter a search query, (empty to finish): fifa goal-line technology
Results:  fifa goal-line technology
File:  bbc_football\214.txt Title:  FA ready to test out goal bleeper
File:  bbc_football\208.txt Title:  Fifa agrees goal-line technology
File:  bbc_football\227.txt Title:  Blatter suggests offside change
Enter a search query, (empty to finish): beckham real madrid spain
Results:  beckham real madrid spain
File:  bbc_football\200.txt Title:  Beckham hints at Man Utd return
File:  bbc_football\201.txt Title:  Beckham's chat reigns in Spain
Enter a search query, (empty to finish):
```

Testing:

You are responsible for testing your program carefully. Make sure that you have thought about all the things that can go wrong and test your program to ensure that you know it works correctly in all cases. Make sure that you have tested your code on the Codio platform.

Submitting your assignment

At the submission link on Codio:

1. Make sure your student number (not your name) is included in comments at the top of your program.
2. Upload your program.
3. If you have included any additional functionality not specified in the brief, please submit a short text document on Codio describing what you have done. Keep this short, it is not graded. The purpose of the document is to guide the marking.

Assessment

You are expected to show that you can code competently using the programming concepts covered so far in the course up to the topics lists and strings.

Marking criteria will include:

- Correctness – your code must perform as specified
- Programming style – your variable names should be meaningful and your code as simple and clear as possible. See section 'Style Guide' for more detail.
- Your assignment will be marked using the rubric at the end of this document. This is the standard rubric used in the Department of Computer Science. Marks for your project work will be awarded for the capabilities (i.e. functional requirements) your system achieves, and the quality of the code.

Additional Challenges

- Additional marks may also be gained by taking on extra challenges but you should only attempt an additional challenge if you have satisfied all requirements for the coursework.
- It's up to you what you choose to do, if anything.
- Note: You are strongly encouraged to follow the specification carefully and to use programming techniques as described in the course materials and textbooks. Poor quality code with additional functionality will not improve your marks.

Plagiarism

Plagiarism will not be tolerated. Your code will be checked using a plagiarism detection tool.

Style Guide

You must adhere to the style guidelines in this section.

Formatting Style

1. Use Python style conventions for your variable names (snake case: lowercase letters with words separated by underscores (_) to improve readability).
2. Name constants properly.
3. Choose good names for your variables. For example, `num_bright_spots` is more helpful and readable than `nbs`.
4. Use a tab width of 4 or 8. The best way to make sure your program will be formatted correctly is never to mix spaces and tabs -- use only tabs, or only spaces.
5. Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

   ```
   b = 3 > x and 4 - 5 < 32
   b= 3>x and 4-5<32
   ```

6. Each line must be less than **80 characters** long *including tabs and spaces*. You should break up long lines using \.
7. Functions should be no longer than about 12 lines in length. Longer functions should be decomposed into 2 or more smaller functions.

Docstrings

If you add your own functions you should comment them using docstrings. Take a look at the code you've been given for some examples. Your comments should:

1. Describe precisely *what* the function does.
2. Do not reveal *how* the function does it.
3. Make the purpose of every parameter clear.
4. Refer to every parameter by name.
5. Be clear about whether the function returns a value, and if so, what.
6. Explain any conditions that the function assumes are true. Examples: "n is an int", "n != 0", "the height and width of p are both even."
7. Be concise and grammatically correct.
8. Write the docstring as a command (e.g., "Return the first ...") rather than a statement (e.g., "Returns the first ...")

Acknowledgements

The BBC news text files used in this assignment were created by staff working for the Insight project at the University of Dublin, Ireland: http://mlg.ucd.ie/index.html#data

Search algorithm

The table below illustrates an algorithm for finding the list of filenames in which all the words in query_words appear. Note: the algorithm is not exactly written in Python. Your job is to understand the algorithm and write it in Python.

Given these values for a list called query_words and the dictionary index:

```
query_words = [‘one’, ‘two’, ‘three’]

index = { ‘one’: [‘filea.txt’, ‘fileb.txt’],

        ‘two’: [‘filea.txt’, ‘filec.txt’],

        ‘three’: [‘filea.txt’, ‘filed.txt’] }
```

Here is an algorithm:

Algorithm	Variables			
	word	**current_file_list**	**last_file_list**	**common_files**
`last_file_list = value from index for key query_words[0] or [] if key doesn’t exist in the index`			[‘filea.txt’, ‘fileb.txt’]	
`for word in the remaining query_words:`	(1) ‘two’ (2) ‘three’			
`    current_file_list = index entry for word or [] if it doesn’t exist`		(1) [‘filea.txt’, ‘filec.txt’] (2) [‘filea.txt’, ‘filed.txt’]		
`    common_files = common_elements(current_file_list, last_file_list)`				(1) [‘filea.txt’] (2) [‘filea.txt’]
`    last_file_list = common_files`			(1) [‘filea.txt’] (2) [‘filea.txt’]	

Key:

(1) – the value of the variable on the first iteration of the loop at the corresponding line of code
(2) – the value of the variable on the second iteration of the loop at the corresponding line of code

UCL Computer Science: Marking Criteria and Grade Descriptors

	Fail		Pass (2:2)	Merit (2:1)	Distinction (1st)		
1-19: Misunderstanding of assignment or similar 20-29: 5 inadequate 30-34: 4 inadequate 34-39: 3 inadequate	Inadequate	Weak	Satisfactory	Good	Excellent	Outstanding	Exceptional
	Below 40: BSc: Fail MEng: Fail	40-49: BSc: 3rd MEng: Fail	50-54: Low pass 55-59: High pass	60-64: Low merit 65-69: High merit	70-79	80-89	90+
1 Quality of the response to the task set: answer, structure and conclusions	Either no argument or argument presented is inappropriate and irrelevant. Conclusions absent or irrelevant.	An indirect response to the task set, towards a relevant argument and conclusions.	A reasonable response with a limited sense of argument and partial conclusions.	A sound response with a reasonable argument and straightforward conclusions, logical conclusions.	A distinctive response that develops a clear argument and sensible conclusions, with evidence of nuance.	↔	Exceptional response with a convincing, sophisticated argument with precise conclusions.
2 Understanding of relevant issues	Misunderstanding of the issues under discussion.	Rudimentary, intermittent grasp of issues with confusions.	Reasonable grasp of the issues and their broader implications.	Sound understanding of issues, with insights into broader implications.	Thorough grasp of issues; some sophisticated insights.	↔	Exceptional grasp of complexities and significance of issues.
3 Engagement with related work, literature and earlier solutions	Very limited or irrelevant reading.	Significant omissions in reading with weak understanding of literature consulted.	Evidence of relevant reading and some understanding of literature consulted.	Evidence of plentiful relevant reading and sound understanding of literature consulted.	Extensive reading and thorough understanding of literature consulted. Excellent critical analysis of literature.	↔	Expert-level review and innovative synthesis (to a standard of academic publications).
4 Analysis: reflection, discussion, limitations	Erroneous analysis. Misunderstanding of the basic core of the taught materials. No conceptual material.	Analysis relying on the partial reproduction of ideas from taught materials. Some concepts absent or wrongly used.	Reasonable reproduction of ideas from taught materials. Rudimentary definition and use of concepts.	Evidence of student's own analysis. Concepts defined and used systematically/ effectively.	Evidence of innovative analysis. Concepts deftly defined and used with some sense of theoretical context.	↔	Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context.
5 Algorithms and/or technical solution	No solution to the given problem, completely incorrect code for the given task.	Rudimentary algorithmic/technical solution, but mostly incomplete.	Reasonable solution, using basic required concepts, several flaws in implementation.	Good solution, skilled use of concepts, mostly correct and only minor faults.	Excellent algorithmic solution, novel and creative approach.	↔	Exceptional solution and advanced algorithm/technical design.
6 Testing of solution (e.g., correctness, performance, evaluation)	No testing or evaluation done.	Few test cases and/or evaluation, but weak execution.	Basic testing done, but important test cases or parts of evaluation missing or incomplete.	Solid testing or evaluation of solution, well done evaluation with good summary of findings.	Very well done test cases, excellent evaluation and very high quality summary of findings.	↔	Exceptionally comprehensive testing, extremely thorough approach to testing and/or evaluation.
7 Oral presentation or demonstration of solution	Poorly done presentation or demonstration, very low quality.	Ineffective oral presentation or demo of the solution.	Able to communicate, present and/or demonstrate solution and summarise work in appropriate format.	Overall good presentation or demo, persuasive and compelling.	Very high quality of delivery. Use of presentation medium with professional style.	↔	Flawless and polished presentation, exceptional quality of demonstration.
8 Writing, communication and documentation	Style and word choice seriously interfere with comprehension.	Style and word choice seriously detract from conveying of ideas.	Style and word choice sometimes detract from conveying of ideas.	Style and word choice work well to convey most important ideas. Well documented.	Style and word choice show fluency with ideas and excellent communication skills.	↔	Reads as if professionally copy edited. Exceptional high quality of writing.
9 Formatting aspects, visuals, clarity, references	Poorly formatted, inappropriate visuals, and incorrect reference formatting.	Formatting, visuals and referencing seriously distract from argument.	Formatting, visuals and referencing sometimes distract from argument.	Formatting well-done and consistent, good visuals and consistent referencing.	Formatting, visuals and referencing are impeccable.	↔	Exceptional presentation, impeccable formatting of the document and references.