CS2510: Data Structures and Algorithms – Assignment 2

Muhammad Yaseen Khan
DCS, MAJU

Fall – October 27, 2018

Instructions.

(i) Do not cheat. (ii) The assignment is consisting of four questions. (iii) You have to submit one `***.java` file that accommodate answers to all questions. (iv) Those questions where a descriptive answer is required, you can do in the same `***.java` file, as/in multi-line comments. (v) Read the first and third instructions again. (vi) This is an individual assignment. (vii) `***` in the instructions iii and iv means your student id. (viii) Failing to comply the instructions is same as failing to get assignment graded.

Glossary.

Token: an individual occurrence of a linguistic unit in speech or writing. Or in computing, the smallest meaningful unit of information in a sequence of data for a compiler.

Term: tokens considered singularly/uniquely.

Vocabulary: a dictionary of current scientific and technical terms.

Scenario.

The Internet is the new electricity. Over billions of websites consisting of trillion of web-pages, we deal a lot of things. Amongst all of those websites, the most amazing one is the *Search Engine*. If we consider search engines and closely observe the questions that: How come they work so quickly? How accurately they answer the query? How correctly they list the results? With more than 200 languages on the internet, how the search engines process the data? Above all, how do they maintain the *data* of all web-pages over the internet for answering our queries?

Well, we know that for the sake of maintaining and organisation of the data, *data structures* are has got the utmost importance in usage. A wide range of data structures are available, and (trust me) if (they are) not (available), we can simply synthesise one or more data structures such that the modified data structure will work for us.

Typically, as a very baseline approach, the search engines use a matrix-like data structure for saving the data of web-pages. I can argue, a matrix is nothing, but a list of lists. Precisely, list of lists of the same length. For example, $M_{[r \times c]}$ is the matrix (that, I am convincing, is a list of lists) then every row in M has got an equal number of columns. Thus, in my version of the list of lists, it will be one list, where every item inside the list is a list, considering the length of every nested list is same. Additionally, one more point I can raise in the favour of the list of lists is, a list of lists can be *jagged*, which means it is not necessary to have the length of every nested list equal to a specific number.

Coming back to the baseline approach that search engines use to maintain the data utilises a matrix-like / list of lists data structure. It is called "Term-Document Incidence" (TDI). How do they work? It consists of 2 steps:

0. Construction of a universal vocabulary ($\mathbb{V}$).
1. Employing $\mathbb{V}$ to construct TDI.

Before we proceed further, let's assume we have following corpus ($\mathbb{C}$) consisting of 10 lines, treat each line as the sample document:

0. لب پہ آتی ہے دعا بن کی تمنا میری
1. لب پہ دعا ہے دل میں لگن
2. تمھارے کام کا کیا بنا؟
3. وہ آیا کرسی کھینچی اور بیٹھ گیا
4. وقت کی مکھیاں تیر کو پسند کرتی ہیں
5. کام کام اور کام
6. اسلم سلیم کے ساتھ شوروم گیا
7. اس نے گاڑی خریدی
8. ہوشیار باش
9. نقالوں سے ہوشیار

Task 1: consider the first point of constructing Term-Document Incidence i.e. construction of the universal vocabulary. It means in having a list of *unique* words in your corpus. How do we construct it? The answer is quite simple: *Split* every document with *space* and *punctuation marks*. This will result in the list of *tokens*. Append these words in a *global* list of tokens. We have to repeat this process for every document in our corpus. Thus, the complexity of the whole operation is somewhat near to $O(n)$ where $n = |\mathbb{C}|$. Hold on... The construction of $\mathbb{V}$ is not finished yet. The global list of tokens of the whole corpus does not mean in the vocabulary. A vocabulary should be a *unique* set of tokens. Hence, we have to modify our approach as it was stated above. What we actually have to do is: Initialise a global list for vocabulary (say $\mathbb{V}$). Then. (i) iterate every document in corpus ($\mathbb{C}$), (ii) tokenise the document (d) [such that $d \in \mathbb{C}$] with the split function to get the list of tokens ($\mathbb{T}$), and (iii) iterate $\mathbb{T}$ to look up whether the token (t) [such that $t \in \mathbb{T}$] is in vocabulary [mathematically, $t \ni \mathbb{V}$], if the lookup fails—insert t into $\mathbb{V}$, otherwise proceed towards the next token. The complexity massively increases to $O(n \cdot m \cdot v)$ where $n = |\mathbb{C}|$, $m =$ number of tokens w.r.t i^{th} document in $\mathbb{C}$, and $v =$ size of $\mathbb{V}$. Considering the same size of n, m, v we have the complexity $\approx O(n^3)$. The algorithm 1 makes you understand it more comfortably.

```
Result: A list of strings ($\mathbb{V}$)
$\mathbb{C} \leftarrow$ be the corpus;
$\mathbb{V} \leftarrow$ be the empty list;
for each document $d$ in $\mathbb{C}$ do
    $\mathbb{T} \leftarrow$ split $d$ into tokens;
    for each token $t$ in $\mathbb{T}$ do
        if $t \not\ni \mathbb{V}$ then
            [$t \not\ni \mathbb{V}$ alternatively means that you are searching $t$ in $\mathbb{V}$. Thus, searching has a hidden loop];
            insert $t$ into $\mathbb{V}$;
        end
    end
end
```

Algorithm 1: Algorithm for building vocabulary (universal set of distinct terms).

Factually there are 55 tokens in the corpus, and as the result of above algorithm, we have got the list of 44 *terms* (set of unique tokens) as below:

0. روم 1. نے 2. لگن 3. آیا 4. میں 5. شو 6. لب 7. کے 8. پسند 9. کی 10. کا 11. بیٹھ 12. وہ 13. آتی 14. اس 15. باش 16. تیر 17. ساتھ 18. ہیں 19. کرتی 20. ہے 21. گاڑی 22. خریدی 23. پہ 24. بنا 25. کیا 26. کام 27. سلیم 28. دعا 29. نقالوں 30. کو 31. تمنا 32. کھینچی 33. سے 34. بن 35. تمھارے 36. گیا 37. ہوشیار 38. کرسی 39. اسلم 40. مکھیاں 41. اور 42. دل 43. وقت

Task 2: Once, we have the vocabulary we can employ it for the construction of Term-Document Incidence. What exactly a “Term-Document Incidence” is? The answer is a bit interesting. To understand it, let’s consider 3 documents from our corpus:

0. لب پہ آتی ہے دعا بن کی تمنا میری

1. لب پہ دعا ہے دل میں لگن

2. تمھارے کام کا کیا بنا؟

You can observe that the lengths of these documents are different. The d_0 has 9 tokens whereas d_1 and d_2 has 6 and 5 tokens respectively. The TDI represents every document in the corpus in/with the same length of tokens. How? For example, as we know there are 3 documents in our corpus (consider the above 3 documents only), and the length of vocabulary $= |\mathbb{V}| = 16$ (w.r.t the 3 documents), then, a TDI will be a matrix $\mathbb{M}_{[3\times16]}$. Each row in $\mathbb{M}$ represents a single document. Similarly, every column in $\mathbb{M}$ represents a specific term of vocabulary. Thus for the document i, we can mark 1 if a term j occurs in i as $\mathbb{M}[i][j] = 1$, otherwise $\mathbb{M}[i][j] = 0$. Hence, we can imagine $\mathbb{M}$ as a boolean representation of corpus as well. You can see in the table below the TDI constructed for the selected documents.

	دعا	میری	ہے	کا	کی	پہ	لگن	بنا	تمھارے	بن	تمنا	کیا	میں	آتی	لب	دل	کام
d_0	1	1	1	0	0	1	0	0	0	1	1	0	0	1	1	0	0
d_1	1	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0
d_2	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1

So, the table has 3 rows (leave the header-row which has the Urdu terms for examples), each row is representing a single document. Every column entertains the specific term of vocabulary. Hence, every row/document in the table, we have an equal number of columns. The 1s in the row representing the presence of word (corresponding column) in the respective document. How would you construct Term-Document Incidence? (i) Initialise a matrix, where the number of row = number of documents, and number of columns = number of terms in vocabulary i.e. $\mathbb{M}_{[|\mathbb{C}|,|\mathbb{V}|]}$. (ii) Iterate a loop for every document (d) in corpus $\mathbb{C}$ [such that $d \in \mathbb{C}$], (iii) tokenise d by splitting it into words, it will generate a list of tokens $\mathbb{T}$, (iv) iterate every token t in $\mathbb{T}$, and search whether t exists in the vocabulary $\mathbb{V}$. (v) And as we know, the search operation will return the index of the item where it occurs first in the list, therefore, if the result of searching t in vocabulary, is *true* then mark 1 at $\mathbb{M}[d][\mathbb{V}.search(t)]$ *otherwise*, $\mathbb{M}[d][\mathbb{V}.search(t)] = 0$. Refer the following pseudocode for a more clear representation.

```
Result: A list of strings ($\mathbb{V}$)
$\mathbb{C} \leftarrow$ be the corpus;
$\mathbb{V} \leftarrow$ be the vocabulary;
$\mathbb{M} \leftarrow$ be the matrix$[|\mathbb{C}|][|\mathbb{V}|]$;
for each document $d$ in $\mathbb{C}$ do
    $\mathbb{T} \leftarrow$ split $d$ into tokens;
    for each token $t$ in $\mathbb{T}$ do
        result $\leftarrow \mathbb{V}.search(t)$;
        [$\mathbb{V}.search(t)$ alternatively means in $t \exists \mathbb{V}$];
        if $result \neq -1$ then
            $\mathbb{M}[d][result] \leftarrow 1$;
        else
            $\mathbb{M}[d][result] \leftarrow 0$;
        end
    end
end
```

Algorithm 2: Algorithm for building term-document incidence.

Hints and Challenges.

0. Make a list of *zeros*. Update the value to 1 where needed.
1. Beware of your computation powers.
2. Work smartly. Try to work with the smaller sample of file. Then gradually increase the number of documents in file.
3. Since the matrix is sparse, which means a lot of zeros and very few ones out there in a row. It will make your TDI needy for more space in memory. Hence, it is raising voice in favour of "list of lists" rather than going for "2D Matrix". Why should we bother about the *zeros* in TDI? Why don't we keep information about the terms that appears in the document? In this case, we are actually making an sparse *jagged list*.

Dataset.

The Urdu corpus is available at: `https://github.com/MuhammadYaseenKhan/Urdu-Sentiment-Dataset`. See the .txt file.

What you can do.

0. You are allowed to use the built-in data structures taught in class i.e. ArrayLists, Linked-Lists, Queues, and Stacks.
1. You can use the variations for the construction of TDI as explained above (2D matrix is a list of lists).
2. Reference for reading file in UTF-8 format in Java `https://howtodoinjava.com/java/io/how-to-read-write-utf-8-encoded-data-in-java/`

Submit timely on your Google Classroom
The due date is November 6, 2018