

A new string matching algorithm for analyzing university curriculum with respect to current job circular

MD. Obaidullah Al-Faruk

ID: 2013-2-60-038

K.M. Akib Hussain

ID: 2013-2-60-045

MD. Adnan Shahriar

ID: 2013-1-68-046

**A thesis submitted in partial fulfillment of the requirements for the
degree of Bachelor of Science in Computer Science and Engineering**



**Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh**

May , 2018

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of our united efforts and under the wonderful support and supervision of our honorable supervisor Shakila Mahjabin Tonni, Lecturer, Department of Computer Science and Engineering, East West University. We also declare that no part of this thesis done under CSE497 has been or is being submitted elsewhere for the requirement of any degree or diploma other than publication and journal purpose.

Supervisor

.....
(Shakila Mahjabin Tonni)
Lecturer,
Dept. of Computer Science and Engineering
East West University, Dhaka, Bangladesh.

Students

.....
(MD. Obaidullah Al-Faruk)
(ID: 2013-2-60-038)
.....
(K.M. Akib Hussain)
(ID: 2013-2-60-045)
.....
(MD. Adnan Shahriar)
(ID: 2013-1-68-046)

Letter of Acceptance

I hereby acknowledge that this Thesis Report has been submitted by MD. Obaidullah Al-Faruk (ID: 2013-2-60-038), K.M. Akib Hussain (ID: 2013-2-60-045) and MD. Adnan Shahriar (ID: 2013-1-68-046), to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh, and is accepted by the department in partial fulfillment of requirements for the Award of the Degree of Bachelor of Science and Engineering on April, 2018.

Supervisor

.....

(Shakila Mahjabin Tonni)

Lecturer, Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh.

Chairperson

.....

(Dr. Ahmed Wasif Reza)

Chairperson and Associate Professor,

Department of CSE, East West University.

Abstract

Mining data from text is often becomes a crucial part of data mining tasks. With the growing tendency of using cloud and sharing more and more files over the internet, the necessity of applying a string matching algorithm in text mining has increased rapidly in present time. In recent years many pattern matching algorithms are proposed to enhance information retrieval from large file(s), especially in search engines as a mean of searching a certain term throughout multiple web pages to rank pages. These tasks require a faster string matching that can find a certain pattern from a text with a very minimal waste of time. This can be ensured by using an algorithm that makes less character comparisons and pattern shifts while searching. In this paper, we're proposing a new algorithm named *Back and Forth Matching (BFM)* algorithm to perform string matching tasks in faster way by matching a pattern from both the forward and backward direction. A comparison of this algorithm with other algorithm shows a tremendous improvement in matching strings in large text files. For this advantage, we have implemented this algorithm in searching through universities course curriculum in Bangladesh context and compare it with the existing job circulars, in order to find out how much the university curriculum is relevant with respect to current job markets. This will provide universities to learn about their laggings and thereby make necessary improvements as per the suggestions generated from our project.

Acknowledgments

First of all, we would like to thank Almighty Allah for giving us the patience, courage, and strength that was necessary to complete this thesis work.

We express our sincere gratitude to honorable supervisor Shakila Mahjabin Tonni, Lecturer, Department of Computer Science and Engineering, East West University, for her kind continuous support, valuable suggestions, and guidance for the past 8 months. We are indebted to her, for keeping us aloft and spirited throughout the entire thesis period. Also, worth mentioning is about our beloved parents for their endless support, untold sacrifices, and constant encouragements toward us.

Last but not the least, we would like to thank all those personnel, who despite of their valuable time, shared knowledge and experiences, with us as well as gave their valuable support and suggestions regarding this thesis work to extract the best possible outcome from us.

MD. Obaidullah Al-Faruk

April, 2018

K.M. Akib Hussain

April, 2018

MD. Adnan Shahriar

April, 2018

Table of Contents

Declaration of Authorship	i
Letter of Acceptance	ii
Abstract	iii
Acknowledgment	iv
Table of Contents	v
List of Figures	vii
List of Tables	x
List of Algorithms	xi
Chapter 1 Introduction	1
1.1 Background	1
1.2 String Matching Procedure	1
1.3 Literature Review	2
1.4 Objective of Research	3
1.5 Organization of the Report	3

Chapter 2 Related Work	4
2.1 Boyer-Moore Algorithm	5
2.2 Boyer Moore Hoorspool Algorithm	6
2.3 Knuth-Morris-Pratt algorithm	7
Chapter 3 Proposed Algorithm	11
3.1 Pseudocode	11
3.2 Example	12
3.2.1 Preprocessing phase	13
3.2.2 Searching phage	13
3.3 Flowchart of the working process	16
3.3.1 Flowchart of Preprocessing phase	16
3.3.2 Flowchart of Searching phase	17
Chapter 4 System Development	19
4.1 Applied Tools	19
4.1.1 Django framework v1.11	19
4.1.2 SQLite	19
4.1.3 Python v3.6.2	20
4.1.4 PyCharm v2017.3.3	20
4.1.5 NLTK 3.2.5	21
4.1.6 CSS	21
4.1.7 Bootstrap v3.3.7	21
4.1.8 Jinja v2	21
4.2 User Interface description	22
4.2.1 Home page	22
4.2.2 Data Insertion options	23
4.2.2.1 Job data insert	23

4.2.2.2	University course content insert	24
4.2.3	View extracted keyword page	25
4.2.3.1	Extracted Job Keywords	25
4.2.3.2	Extracted Curriculum Keywords	26
4.2.4	Content Compare option	26
4.2.5	Comparison of Applied algorithms page	29
4.3	Flowchart of the working process	30
Chapter 5	Result Analysis	31
5.1	Evaluation of job Suggestion	31
5.2	Evaluation of Algorithm with Real Data	39
Chapter 6	Conclusion and Future Works	43
6.1	Conclusion	43
6.2	Future Works	43
	Bibliography	44
	Appendix A Source Code	49
	Appendix B List of Publications	74

List of Figures

2.1	Preprocessing table for Boyer Moore Hoorspool	6
2.2	Searching Phase For BMH	6
2.3	Preprocessing table for KMP	8
2.4	Searching Phase For KMP	9
3.1	The structure of <i>posIndex</i> table	13
3.2	Preprocessing table for the proposed algorithm	13
3.3	step 1 of searching phase	14
3.4	step 2 of searching phase	14
3.5	step 3 of searching phase	15
3.6	step 4 of searching phase	15
3.7	exact pattern match in searching phase	15
3.8	Flowchart of Preprocessing phase	16
3.9	Flowchart of Searching phase	17
4.1	Home Page	22
4.2	Job data insert	23
4.3	University course content insert	24
4.4	Search Job Contents	25
4.5	Search University Contents	26
4.6	Content Compare page	27

4.7	Find Similarity page	28
4.8	Comparison of Applied algorithms page	29
4.9	Flowchart of System Implementation	30
5.1	Graph of Result analysis of applied algorithms	38
5.2	Pie chart of execution time	39
5.3	Total number of shifts to search “Bangladesh”	40
5.4	Total number of comparisons made to search “Bangladesh”	41
5.5	Execution time (sec) required to search “Bangladesh”	41

List of Tables

5.1	Performance analysis on different character sized text	42
-----	--	----

List of Algorithms

1	<i>Preprocessing</i> (T, P)	12
2	<i>Searching</i> ($T, P, m, n, posIndex[]$)	12

Chapter 1

Introduction

1.1 Background

As more and more data and files are being used and needed to be processed on real time over the Internet, so at present retrieving information by processing huge text file, has become an imperative data mining task [1]. To find a desired text (pattern) in a text document is popularly known as the *string matching*. Though it creates the impression of being a straightforward task, yet it may become a huge time-consuming process if the file size gets increased. Because of this reason, a fast pattern matching algorithm is an indispensable part of page ranking in search engines and digital libraries, checking syntax and spelling mistakes, detecting network breach, and in many other applications [2]. It has also find its way in the application of bio informatics, DNA sequences matching, and behavior analysis [3, 4] sentiment analysis and online advertisements as well [1].

1.2 String Matching Procedure

Algorithms of string matching follow a general principle. They try to find out all the occurrences of a pattern P of length m in the text T of length n . For this, the algorithms first form a window of length m over the text. Next the leftmost character of the pattern and window are aligned. Then the algorithms try to match the characters of the pattern with the current window characters. This specific approach is known as *attempt* [5]. After a whole match between pattern characters and window characters, or upon a mismatch,

the window shifts to the right of the text and the pattern changes its position respective to the window. This succession of *attempts* and window shifts continues, until the shift causes the right end of the window exceeds the length of the text. Such type of window mechanism came to be known as the *sliding window mechanism* [6]. In most cases, the algorithms have two phases, precisely the *preprocessing phase* and the *searching phase* [1]. In the *preprocessing phase*, generally the pattern is preprocessed to form a table that determines at what position the pattern needs to be shifted in finding a mismatch [1]. Then in the *searching phase* comparisons are made between pattern and text characters from right to left, or left to right, or in specific ways to find out all the occurrences of exact pattern match.

1.3 Literature Review

The main challenges of these algorithms are to minimize the character comparisons and to maximize the length of shifts [1, 5]. Thus, in the hope of developing a more efficient string matching algorithm, numerous algorithms have been developed from time to time [4]. Among them, the *Knuth-Morris-Pratt algorithm(KMP)*, the *Boyer-Moore algorithm(BM)* and the *Boyer-Moore Hoorspool(BMH)* algorithm are quite famous [7, 8, 9].

In our research we took two renowned raw string matching algorithms as the base for developing the proposed algorithm. These two algorithms are: *Boyer-Moore-Hoorspool* string matching algorithm and *Knuth-Morris-Pratt string matching algorithm*. The *Boyer-Moore algorithm* is one of the most renowned, efficient and extensively used pattern matching algorithms. It has a significant difference compared to *Nave* approach of string matching. A simpler and an improvement of *BM* is the *Boyer-Moore-Hoorspool(BMH)* algorithm. It tries to find the occurrences of pattern in a text, by making comparisons from right to left instead of from the beginning. It preprocesses the pattern and determines the maximum shifts of the pattern in case of a mismatch based on preprocessing

table known as *bad character shifts*. The *Knuth-Moris-Pratt(KMP)* algorithm works like the *Naive* algorithm, but instead of checking all the characters from left to right after each shift, the algorithm preprocesses a table that helps skipping comparisons of those pattern characters that had already matched with the window characters.

1.4 Objective of Research

In this thesis report titled “**A new string matching algorithm for analyzing university curriculum with respect to current job circular(BFM)**”, we’ve presented a new string-matching algorithm **BFM** to find all the occurrences of exact patterns or a string in a text. The presented algorithm shows much less character comparisons and more shift lengths compared to other algorithms. More importantly, the algorithm does both *forward* and *backward* checking and managed by a preprocessing table that decreases the number of efforts required to match the window with the text during the matching phase.

1.5 Organization of the Report

We have structured our rest of the Thesis works as following: In Chapter 2, we survey related work in string matching; Chapter 3, introduces our proposed algorithm, and its methodology are discussed meticulously; In Chapter 4 we develop a project to analyze university curriculum with respect to current job circular by implementing our proposed algorithm. Moreover, in Chapter 5 the comparative results between our algorithm with two other renowned algorithms are also presented; In Chapter 6, the conclusion and future research directions are highlighted ; Lastly, proper references of our thesis works, source code of our proposed algorithm and implementation, as well as our list of publications has been illustrated respectively.

Chapter 2

Related Work

The main objective of string matching algorithms is to result in fast search by comparing less characters of text and pattern, and also to skip as much unnecessary text characters as possible to quickly find a matched pattern in the text. Due to the huge importance of string matching algorithms in this computing world, so constantly new algorithms are being developed from time to time. In this chapter, we have focused on two famous string matching algorithms- the Boyer-Moore Horspool(BMH) and the Knuth-Morris-Pratt(KMP) algorithm, to relate our proposed algorithm in the next chapter. We also have discussed the Boyer-Moore algorithm(BM), as the BMH is a simple and improved form of BM. Beside the two discussed algorithms, Quick Search algorithm is a recent popular algorithm . Though, according to [10] this algorithm is more appropriate when applied on a large character set to find a small pattern. Quick-Skip Search algorithm [11] proposed a combination of Quick Search and the Skip Search algorithm. Again, in [4] another hybrid algorithm was proposed using the idea of Quick-Skip and Boyer-Moore algorithms.A parallel string matching algorithm based on this work is proposed in [12]. [13] Proposes a new algorithm combining the idea of Boyer-Moore and KMP algorithm. Beside all these algorithms, a comparatively old string matching technique is Rabin Carp string searching algorithm[14] that can search both single and multiple patterns in a string. A more recent enhancement of this algorithm is proposed in [15], that deploys a modified version of the Rabin Carp algorithm using a GPU processor.

2.1 Boyer-Moore Algorithm

The BM is one of the most renowned, efficient and extensively used pattern matching algorithms [1]. Unlike the previous pattern matching algorithms, it tries to find the occurrences of pattern in a text, by making comparisons from right to left instead of from the beginning. It preprocesses the pattern and determines the maximum shifts of the pattern in case of a mismatch based on two heuristics. These heuristics, the *bad character heuristic* and the *good character heuristic* works independently over the pattern to produce two different arrays known as the *preprocessing table* [16]. During the searching phase at each mismatch, BM determines best of the two heuristics and thus can slide the pattern by maximum [17, 18].

The *bad character heuristics* works on certain principles. If there is a mismatch between the current text character $T[i]$ and the pattern character $P[j]$, then the heuristics looks up for the last occurrence of the mismatching character on the pattern. If it exists, then the pattern is shifted to the right and aligned with the text in such a way that the mismatch $T[i]$ becomes a match. And if the mismatched character $T[i]$ does not exist in the pattern, then the pattern is shifted past the mismatched character by m positions to the right [19].

The *good suffix heuristics* works in the following way. In the first case, while scanning characters from right to left order, suppose there is a substring t in the text just before a mismatch between $T[i]$ and $P[j]$, within the length m of the text. If substring t exists in the pattern within $P[0...j]$ then the pattern is shifted to the right so that the t in P is aligned with the t in T . If t is not found in $P[0...j]$, then in this second case the heuristics looks for a suffix of t that matches with the prefix of P within $P[0...j]$. If so, then P is shifted so that the obtained prefix of P is aligned with the suffix of t . And if both the cases failed, then P is shifted by its entire length m , past the mismatched character t [19].

2.2 Boyer Moore Hoorspool Algorithm

The BMH algorithm is a simpler form and an improvement of the BM algorithm [20]. It only uses the *bad character heuristics* of the BM algorithm, whereas the *good suffix heuristics* is ignored, as its practical implementation is difficult and more complex [2]. Yet, BMH's algorithm shows similar kind of performance as the BM algorithm [1]. Given below is an illustration of the working procedure of BMH algorithm:

We consider,

a text **T**: GCATCGTATACAGCAGAGAGTAC

and a pattern **P**: GCAGAGAG

The preprocessing table constructed using the pattern is as follows:

G	C	A	*
2	6	1	8

Figure 2.1: Preprocessing table for Boyer Moore Hoorspool

During the searching phase, the following is observed:

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
TEXT	G	C	A	T	C	G	T	A	T	A	C	A	G	C	A	G	A	G	A	G	T	A	C	G
STEP 1	G	C	A	G	A	G	A	G																
								x																
STEP 2		G	C	A	G	A	G	A	G															
									x															
STEP 3										G	C	A	G	A	G	A	G							
																	x							
STEP 4											G	C	A	G	A	G	A	G						
													x	✓	✓	✓	✓							
STEP 5														G	C	A	G	A	G	A	G			
														✓	✓	✓	✓	✓	✓	✓	✓			

Figure 2.2: Searching Phase For BMH

- **Step 1:** There is a mismatch found between the characters $T[7]$ and $P[7]$ while checking from right to left. And the value of the mismatched text character $T[7]$ in the preprocessing table is 1. Hence the pattern is right shifted by 1 character.
- **Step 2:** Again a mismatch is found between $T[8]$ and $P[7]$. The character value of $T[8]$ according to the table is 8. So, the pattern is right shifted by 8 characters.
- **Step 3:** For the mismatch between $T[16]$ and $P[7]$, the character at $T[16]$ is 'A' whose value in the table is 1, and hence the pattern is shifted by 1 character.
- **Step 4:** In this step, checking from right to left we found matches up to $P[4]$. Then for the mismatch found between $P[3]$ and $T[13]$ we check value of the first matched character of the pattern which is G. So, the pattern is shifted by 2 characters.
- **Step 5:** Finally, in this step, every successive character of the pattern and current window are matched. So, the pattern is found in the text. Total number of shifts involved is 4.

2.3 Knuth-Morris-Pratt algorithm

The KMP algorithm works like the Naive algorithm, but alongside using the *degenerative property* of the pattern. That is, instead of checking all the characters after each shift, the algorithm preprocesses P to construct a table $pos[]$ that helps skipping comparisons of those pattern characters that had already matched with the window characters. The $pos[i]$ holds the value that helps finding the longest proper prefix of $P[0..i]$, that matches a suffix of $P[0..i]$ for each substring $P[0..i]$ where $i=0$ to $m-1$.

The KMP does the following in the searching phase. It initially forms a window over T , starting from $T[0]$ equal to length m and aligns $P[0]$ with $T[0]$. Then it scans from left to right order to see whether the successive characters of the current window match with that of P [12, 13, 21, 22, 23]. Whenever checking for $P[0]$, if there is mismatch then

the pattern is shifted by one character. Other than $P[0]$, if suppose there is a mismatch between $T[i]$ and $P[j]$. Then from the preprocessing table $pos[]$ it is found out whether there exists a suffix before $P[j]$ that has a proper prefix within the $P[0\dots j-1]$. If this is the case, then, the pattern is shifted to the right so that the obtained proper suffix replaces the suffix position and the next comparison starts from after the replaced suffix position. Failure in finding a proper prefix, the pattern is right shifted to align $P[0]$ with $T[i]$ and successive characters are compared. In this way it continues, in order to find all occurrences of P in T . The following shows an example of KMP algorithm:

We consider,

a text **T**: GCATCGTATACAGCAGAGAGTAC

and a pattern **P**: GCAGAGAG

The preprocessing table constructed based on the pattern is as follows:

Preprocessing Table:

0.	1.	2.	3.	4.	5.	6.	7.
G	C	A	G	A	G	A	G
0	0	0	1	0	1	0	1

Figure 2.3: Preprocessing table for KMP

Searching Phase:

- **Step 1:** There is a mismatch between the characters $T[3]$ and $P[3]$ while checking from left to right. And the value of the last matched pattern character $P[2]$ in the preprocessing table is 0. Hence the pattern is shifted to the right so that the first character in the pattern is aligned with the mismatched character $T[3]$.
- **Step 2:** Again mismatch is found between $T[3]$ and $P[0]$. So, the pattern is right shifted by 1 character.

- **Step 3:** Yet, again another mismatch occurred between $T[4]$ and $P[0]$. So, the pattern is shifted by 1 character.
- **Step 4:** In this step, checking from left to right, a mismatch found between $P[1]$ and $T[6]$. The value of the last matched character $P[0]$ in the preprocessing table is 0. As such, the pattern is right shifted so that $P[0]$ is aligned with the mismatched character at $T[6]$.
- **Step 5-10:** At every step, mismatch takes place between $P[0]$ and the first character of the aligned window. So, at each step, the pattern is shifted to the right by one character.
- **Step 11:** Finally, in this step, every successive character of the pattern and current window are matched. So, the pattern is found in the text. Total number of shifts involved is 10.

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
TEXT	G	C	A	T	C	G	T	A	T	A	C	A	G	C	A	G	A	G	A	G	T	A	C	G
STEP 1	G	C	A	G	A	G	A	G																
	✓	✓	✓	x																				
STEP 2				G	C	A	G	A	G	A	G													
				x																				
STEP 3					G	C	A	G	A	G	A	G												
					x																			
STEP 4						G	C	A	G	A	G	A	G											
						✓	x																	
STEP 5							G	C	A	G	A	G	A	G										
							x																	
STEP 6								G	C	A	G	A	G	A	G									
								x																
STEP 7									G	C	A	G	A	G	A	G								
									x															
STEP 8										G	C	A	G	A	G	A	G							
										x														
STEP 9											G	C	A	G	A	G	A	G						
											x													
STEP 10												G	C	A	G	A	G	A	G					
												x												
STEP 11													G	C	A	G	A	G	A	G				
													✓	✓	✓	✓	✓	✓	✓	✓				

Figure 2.4: Searching Phase For KMP

In this chapter, we have focused on two algorithms, namely the *BMH* and the *KMP*. We have seen that though the objective is the same, but yet their approach is somewhat different. In the preprocessing stage *BMH* and *KMP*, both preprocesses the pattern. But during comparison of text and pattern characters, *BMH* compares from right to left, whereas *KMP* applies left to right comparison technique. In preprocessing phase *BMH* produces $O(m+n)$ time and $O(n)$ space complexity, and in the searching phase, a time complexity of $O(mn)$. In contrast, *KMP* generates $O(m)$ space and time complexity in the preprocessing phase, and a time complexity of $O(n+m)$ in the searching phase. It has been found that most applications uses *BMH* or *KMP* algorithms for their effective and efficient functionality and other applications uses the basics of these algorithms for their functionalities as the *KMP* algorithm has less time complexity and *BM* and *BMH* algorithms has preprocessing time complexity less. So both algorithms have huge importance for this world.

Chapter 3

Proposed Algorithm

In this chapter we put forward our algorithm *Back and Forth String Matching algorithm (BFM)* and describe about how it is developed targeting larger shifts of pattern window for each mismatch, thus minimizing the total number of shifts while matching strings. Such description will also clear the concept of how this algorithm ensures less character comparison in searching for a pattern in a text. We found out that our algorithm takes a *Preprocessing time complexity* of $\mathbf{O(n)}$ and a *Searching time complexity* of $\mathbf{O(mn)}$

3.1 Pseudocode

To serve the purpose, the algorithm compares characters from both left and right side in each attempt while searching. As the first step of the algorithm, a preprocessing table is prepared to find out the exact positions in the text, starting from where the pattern can be aligned so that its first and last characters finds a match with the aligned text characters.

Algorithm 1 depicts the preprocessing phase of *BFM*.

Algorithm 1 *Preprocessing*(T, P)

```

1.  $i \leftarrow 0$ 
2. while  $i \leq (n - m)$  do
3.   if  $T[i] == P[0]$  then
4.     if  $T[i + m - 1] == P[m - 1]$  then
5.        $posIndex[i] \leftarrow i$ 
6.     end if
7.   end if
8.    $i \leftarrow i + 1$ 
9. end while
10. Searching( $T, P, m, n, posIndex[]$ )

```

To search a pattern in search phase, the algorithm checks for the pattern only at the positions as referred by the preprocessing table. The whole process is given in Algorithm 2.

Algorithm 2 *Searching*($T, P, m, n, posIndex[]$)

```

1.  $length \leftarrow len(posIndex[])$ 
2. for  $i = 0$  to  $length - 1$  do
3.    $k \leftarrow posIndex[i]$ 
4.    $s \leftarrow 0$ 
5.    $txtlast \leftarrow m + k - 1$ 
6.    $patlast \leftarrow m - 1$ 
7.   while  $k \leq txtlast$  do
8.     if  $T[k + 1] == P[s + 1]$  and  $T[txtlast - 1] == P[patlast - 1]$  then
9.        $k \leftarrow k + 1, s \leftarrow s + 1$ 
10.       $txtlast \leftarrow txtlast - 1, patlast \leftarrow patlast - 1,$ 
11.    else
12.      break
13.    end if
14.  end while
15.  if all character matched then
16.    Pattern found
17.  end if
18.   $i \leftarrow i + 1$ 
19. end for

```

3.2 Example

To demonstrate the mechanism of BFM consider the same text and pattern we used to describe KMP and BMH algorithm.

T: GCATCGTATACAGCAGAGAGTACG

P: GCAGAGAG

3.2.1 Preprocessing phase

In the preprocessing phase, our algorithm finds out all the possible positions in the text, where the first and last character of the pattern are matched with the first and last character of the aligned text. In such cases, a table named as *posIndex* is constructed, for keeping only the text character indices, where $P[0]$ is matched.

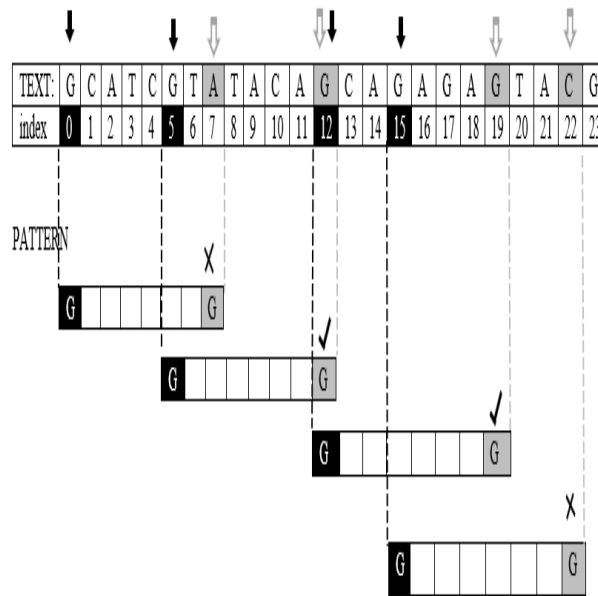


Figure 3.1: The structure of *posIndex* table

The preprocessing table *posIndex*, is thus constructed as follows:

5	12
---	----

Figure 3.2: Preprocessing table for the proposed algorithm

3.2.2 Searching phase

According to *posIndex*, the pattern is positioned below $T[5]$. Then by forward check, we check for a match between the second character of the aligned pattern and text. At the

same time by doing backward check, we check if there is a match in the aligned second last character of the pattern and text. We ignored checking the first and last character, as it has been already checked during the preprocessing phase.

In this way at each step, one character from the first, and one character from the last is checked, until we have checked all the characters in the pattern. If at any step a mismatch is found, then immediately the pattern is shifted to the right, and aligned with the text at the position indicated by the next index of the *posIndex*[].

We illustrate each step of the searching phase in the following:

- **Step 1:** The first index of the *posIndex*[] indicates $T[5]$. So, the pattern is aligned with the text at $T[5]$, and we found a mismatch between $P[1]$ and $T[6]$ during forward checking, whereas there is a match found at $P[6]$ with $T[11]$ during backward checking.

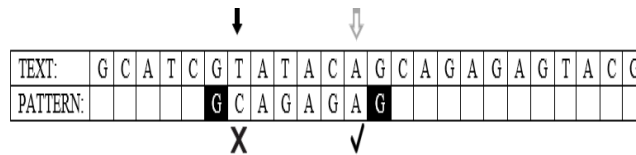


Figure 3.3: step 1 of searching phase

Since a mismatch occurred, hence we choose the next index of the *posIndex*[], that suggests moving the pattern to the new position at $T[12]$.

- **Step 2:** Then we check characters at $P[1]$ and $T[13]$, and characters at $P[6]$ and $T[18]$. Both the forward and backward check resulted in a match.

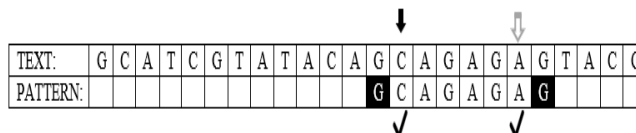


Figure 3.4: step 2 of searching phase

- **Step 3:** Next by forward check, $P[2]$ and $T[14]$ are compared. At the same time, through backward check we compared the characters at $P[5]$ and $T[17]$. Since both comparisons proved to be a match, so we moved to the next step.

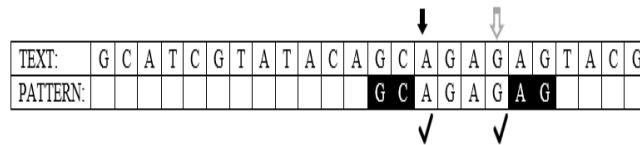


Figure 3.5: step 3 of searching phase

- **Step 4:** Finally, in this last step, it is found that $P[3]$ matches with $T[15]$, and $P[4]$ matches with $T[16]$.

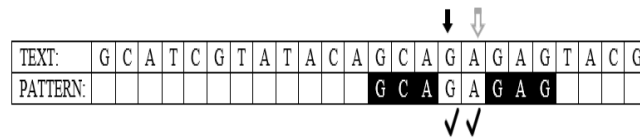


Figure 3.6: step 4 of searching phase

Thus, all the characters are matched, and we can conclude that there is an occurrence of the pattern in the text. Total number of shifts performed is 1.

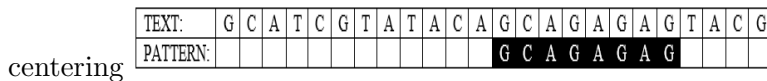


Figure 3.7: exact pattern match in searching phase

3.3 Flowchart of the working process

3.3.1 Flowchart of Preprocessing phase

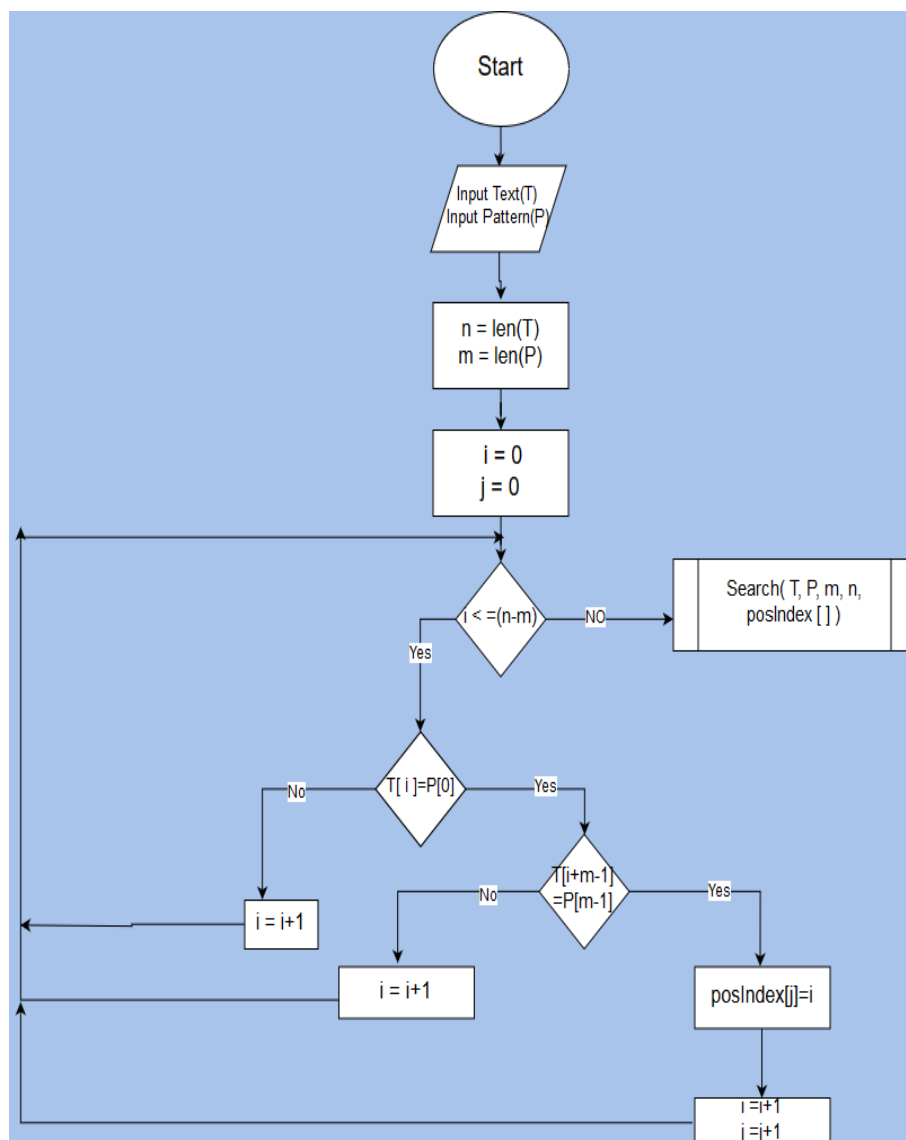


Figure 3.8: Flowchart of Preprocessing phase

3.3.2 Flowchart of Searching phase

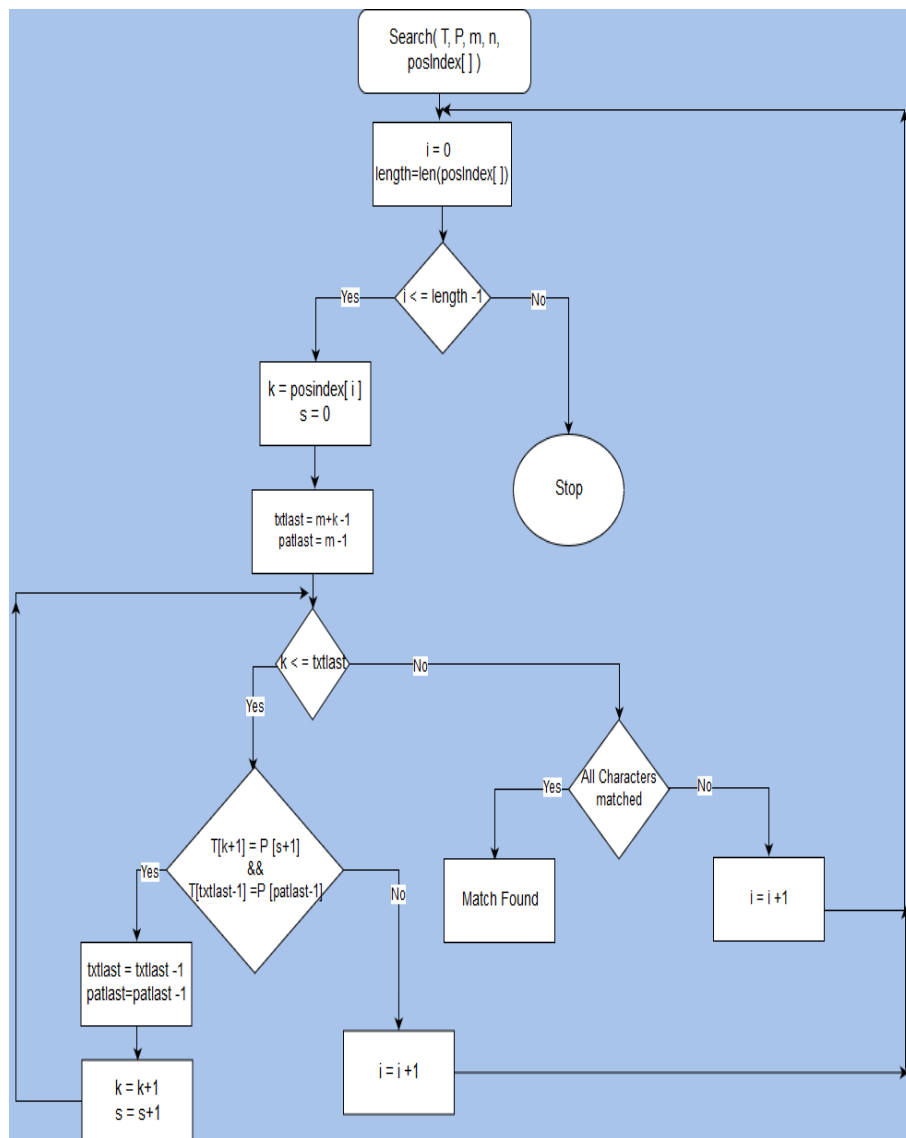


Figure 3.9: Flowchart of Searching phase

In this chapter we have tried to explain the methodology of our proposed algorithm. We started explaining by dividing our whole algorithm into two phases- the preprocessing

phase and the searching phase. We provided the pseudocode for each phase. Then for further clarification of our methodology, we gave a working example of our algorithm. Here we discussed in a stepwise manner about the working process in lieu for a given text and pattern. And in the end we provided a flowchart of the algorithm. Thus this chapter can create the visualization about the effectiveness and the performance of the proposed algorithm.

Chapter 4

System Development

For the thesis implementation, we have build up an application applying our string matching algorithm at its core functionality, whereas using NLTK together with Django framework to give a it a complete shape. Rest of the tools we have used are briefly explained in the next section.

4.1 Applied Tools

4.1.1 Django framework v1.11

Django is a high-level *Python Web framework* that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of *Web* development, so you can focus on writing your app without needing to reinvent the wheel. *Django* was designed to help developers take applications from concept to completion as quickly as possible. *Django* takes security seriously and helps developers avoid many common security mistakes. Django has been referred to as an MTV frameqwork because the controller is handled by the framework itself and most of the excitement happens in models, templates and views

4.1.2 SQLite

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, *SQLite* is not a client-

server database engine. Rather, it is embedded into the end program. *SQLite* is *ACID-compliant* and implements most of the *SQL* standard, using a dynamically and weakly typed *SQL* syntax that does not guarantee the domain integrity. *SQLite* is a popular choice as embedded database software for local/client storage in application software such as web browsers.

4.1.3 Python v3.6.2

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, *Python* has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. *Python* features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. *Python* interpreters are available for many operating systems. *CPython*, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

4.1.4 PyCharm v2017.3.3

PyCharm is an *Integrated Development Environment* used in computer programming, specifically for the Python language. It is developed by the Czech company *JetBrains*. *Pycharm* is an *IDE* tool kit basically meant for developing programs and/or building softwares in *Python*. Developed by *JetBrains* it provides Assistive toolkit and build-in features like most of the *IDE's* like *eclipse*. It also supports development for *JavaScript*, *CoffeeScript*, *TypeScript*, *CSS* etc.

4.1.5 NLTK 3.2.5

The *Natural Language Toolkit*, or more commonly *NLTK*, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the *Python* programming language. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as *WordNet*, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength *NLP* libraries, and an active discussion forum.

4.1.6 CSS

Cascading Style Sheets is a style sheet language. *CSS* is the language for describing the presentation of *Web* pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. *CSS* is independent of *HTML* and can be used with any *XML*-based markup language.

4.1.7 Bootstrap v3.3.7

Bootstrap is a free and open-source front-end library for designing websites and web applications. It contains *HTML*- and *CSS*-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional *JavaScript* extensions. Unlike many web frameworks, it concerns itself with front-end development only.

4.1.8 Jinja v2

Jinja is a template engine for the *Python* programming language and is licensed under a BSD License created by Armin Ronacher. It is a text-based template language and thus can be used to generate any markup as well as source code. The *Jinja* template engine allows customization of tags, filters, tests, and global.

4.2 User Interface description

4.2.1 Home page

- As soon as user start the application he/she enters into the home page. This page contains information for a quick glance of the project
- The navigation bar has got options to access other functionality of the application like:
 1. Data Insertion
 2. Viewing extracted keywords and
 3. Content Comparison

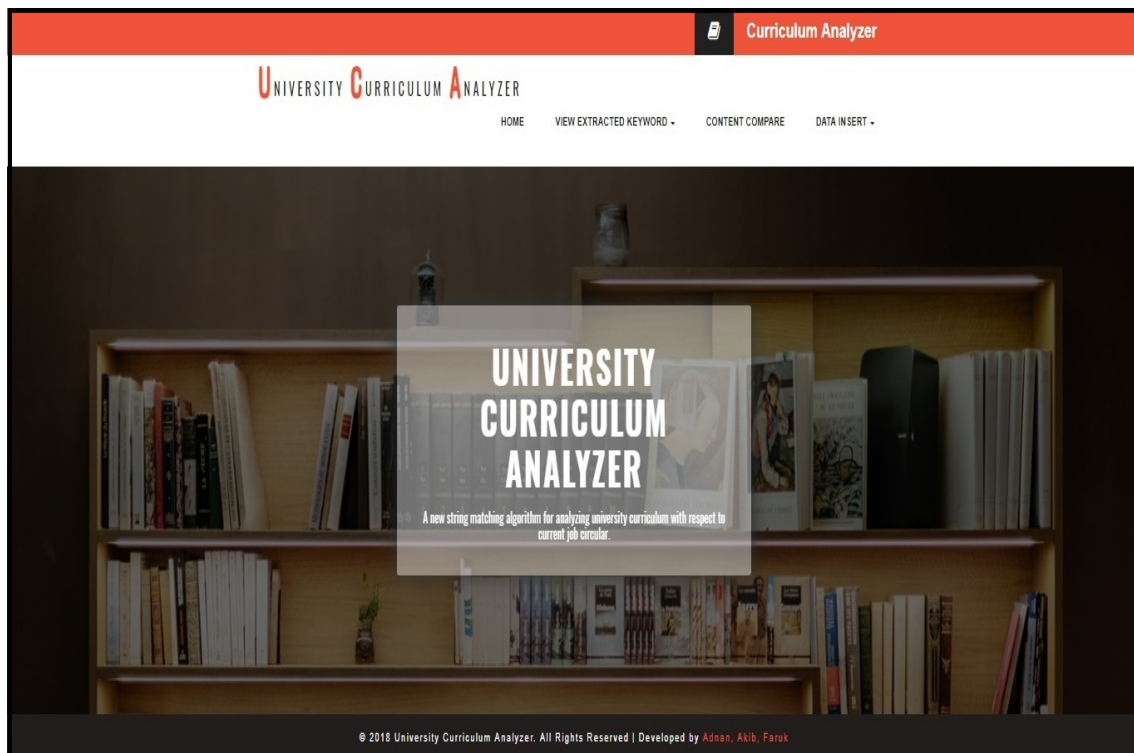


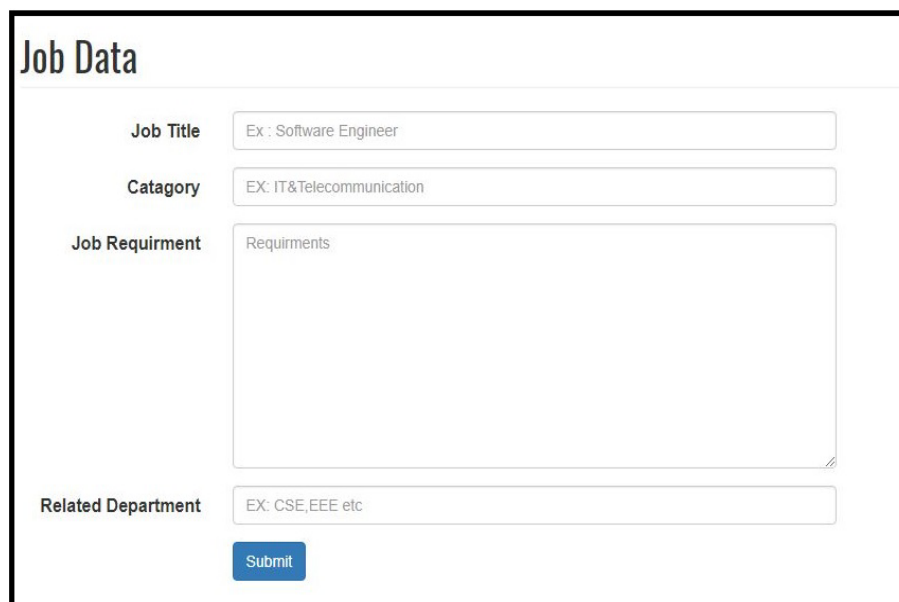
Figure 4.1: Home Page

4.2.2 Data Insertion options

- This option is used if the user wants to store data into the database. Hovering to the "Data insertion" option will instantly generate two options.
 1. Job data insert and
 2. University course content insert
- The user can store two different kind of data in the database. He/she can either insert the information of a job circular using the "job data insertion" option. Or else through the "University course content insert" option he/she can store course contents of a specific department of a university.

4.2.2.1 Job data insert

- This is a part of Data insert option.

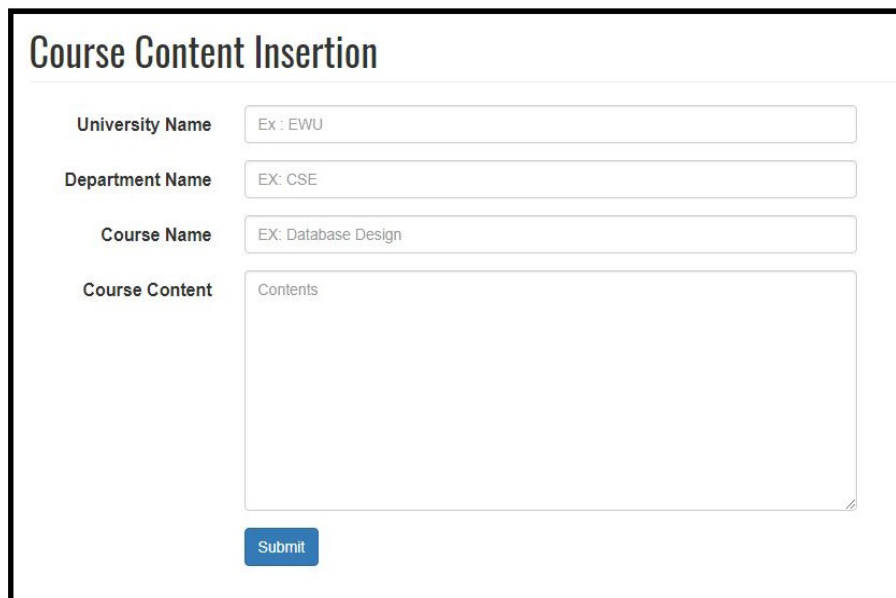


The screenshot shows a web form titled "Job Data". It contains four input fields and a submit button. The fields are: "Job Title" with the example "Ex : Software Engineer", "Catagory" (note the typo) with the example "EX: IT&Telecommunication", "Job Requirement" with the example "Requirments" (note the typo), and "Related Department" with the example "EX: CSE,EEE etc". A blue "Submit" button is located at the bottom of the form.

Figure 4.2: Job data insert

- This page consists of a form where user has to manually insert job data from job circulars. The form contains “Job Title”, “Category”, “Job requirement”, “Related Department” named text fields.
- After completely filling up all the necessary fields of the form, user should press the submit button.
- As a result, the contents of job requirements are analyzed using *Natural language processing toolkit(NLTK)* to extract nouns proper nouns from noun phrases, and the extracted words are stored in database.

4.2.2.2 University course content insert



The image shows a web form titled "Course Content Insertion". It contains four input fields and a submit button. The fields are labeled "University Name", "Department Name", "Course Name", and "Course Content". Each field has a text input area with a small example text above it: "Ex : EWU", "EX: CSE", "EX: Database Design", and "Contents". A blue "Submit" button is located at the bottom center of the form.

Figure 4.3: University course content insert

- This is another part of Data insert option
- This page consists of a form where user has to manually insert course contents

of a university. In this page, there is form which contains “University name”, “Department name”, “Course name”, “Course content” named text fields.

- After completely filling up all the necessary fields of the form, user should press the submit button.
- As a result, the data placed in the “Course Contents” are analyzed using *NLTK* to extract proper nouns from noun phrases, and the extracted words are stored in database.

4.2.3 View extracted keyword page

4.2.3.1 Extracted Job Keywords

EXTRACTED JOB REQUIREMENT KEYWORDS

Select Job Category: IT & Telecommunication

Submit

Selected Category : IT&Telecommunication

Related Department :
EEE
ENG
CSE
BBA

Extracted Keyword:

foundation
analytics
isp
ericsson
template
smart
expert
dreamwaver
telco

Figure 4.4: Search Job Contents

When user has job contents of different job circulars stored in the database, then at any time he/she can enter the extracted job keywords option and all he/she has to do is select a job category from the dropdown menu and click submit button. Then at an instant,

all the extracted keywords of job requirements, and departments of university related to that provided category name is displayed.

4.2.3.2 Extracted Curriculum Keywords

When user has university course contents stored in the database, then user entering this option can see the extracted course content keywords, generated from specific departments of desired universities. These specific departments and their contents are automatically retrieved from the database using the provided category name.

EXTRACTED UNIVERSITY CONTENTS

University Name: EWU University Name : EWU Department Name : CSE

Department Name: CSE

Submit

Extracted Keyword University :

- probabilistic
- csma
- test
- observations
- interoperability
- attack
- statistical
- cryptography

Figure 4.5: Search University Contents

4.2.4 Content Compare option

- User entering this option has the privileges of choosing a specific university, department, and job category from a list of options.
- After selecting desired choices, the user needs to press submit button.

- This will generate extracted keywords of university course contents according to the selected department name and university. Also it will fetch and display extracted keywords of job requirements according to the provided job category, from the database.

The screenshot shows the 'Curriculum Analyzer' web application. The main heading is 'KEYWORDS OF UNIVERSITY, DEPARTMENT AND JOB CATEGORY'. Below this, there are three input fields: 'University' (EWU), 'Department' (CSE), and 'Job Category' (IT & Telecommunication). A 'Submit' button is located below these fields. The results are displayed in two columns: 'Extracted Keyword Of University' and 'Extracted Keyword Of Job'. The university keywords include: operg, communications, flip-flops, pci, basics, develop, boolean, mlps. The job keywords include: designing, laravel, maven/gradle, webdriver, sketch, azure, lamp, setting. A 'Find Similarity' button is located at the bottom center of the page.

Figure 4.6: Content Compare page

- Next, to see how many extracted keywords from job requirements matched with the extracted keywords of university course contents, the user has to press the button entitled "Find Similarity" located at the bottom of the page. It will redirect to a new page.

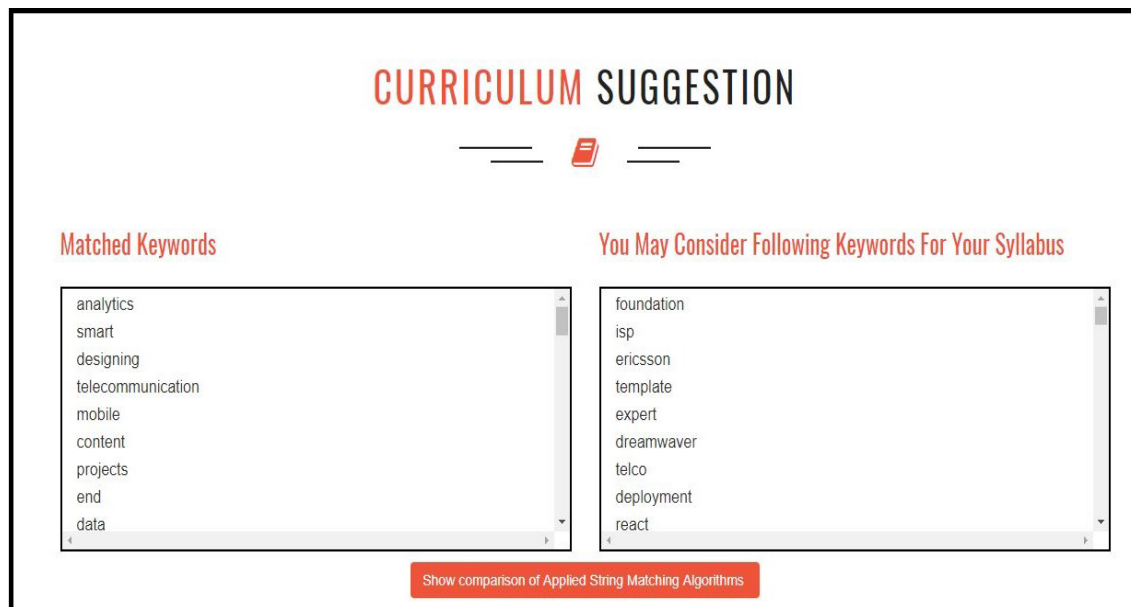


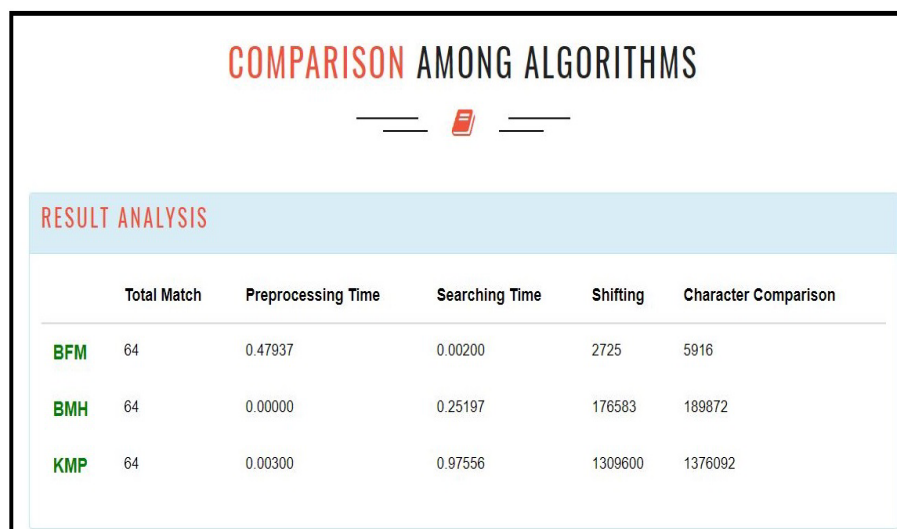
Figure 4.7: Find Similarity page

- This page is about to show the similarity between extracted keywords of job requirements and university course content.
- The page displays two outputs- Matched contents and unmatched contents or Suggestions.
 1. **Matched contents:** The matched contents will show only those keywords that are common to both job requirements of the specified job category and course contents of the predefined university department.
 2. **Unmatched contents or Suggestions:** The unmatched contents will show only those keywords of job requirements that are not available in the course contents of the specified university department, and suggest the user to add these unmatched contents in that course curriculum.
- To find similarity between the job requirements and the university course curricu-

lum, we have considered the extracted keyword course contents as "text" and each extracted keywords of job requirements as "pattern" and thus determine whether there is a match by implementing our proposed string matching algorithm.

- At the bottom of the page a new option titled as "Show comparison of Applied Algorithms" will generate automatically. This option is provided for users, to get a comparative study of the performance of our algorithm with the two other algorithms namely Boyer-Moore Hoorspool and the Knuth-Morris-Pratt algorithm in finding the matched and unmatched contents.

4.2.5 Comparison of Applied algorithms page



The screenshot displays a web page titled "COMPARISON AMONG ALGORITHMS" with a decorative icon of a book. Below the title is a section labeled "RESULT ANALYSIS" containing a table with the following data:

	Total Match	Preprocessing Time	Searching Time	Shifting	Character Comparison
BFM	64	0.47937	0.00200	2725	5916
BMH	64	0.00000	0.25197	176583	189872
KMP	64	0.00300	0.97556	1309600	1376092

Figure 4.8: Comparison of Applied algorithms page

Here we can see the performance comparison of the our algorithm with two other algorithms: BMH and KMP. We show our results in case of preprocessing time, searching time, number of character comparisons, and number of shifts required to complete the searching process.

4.3 Flowchart of the working process

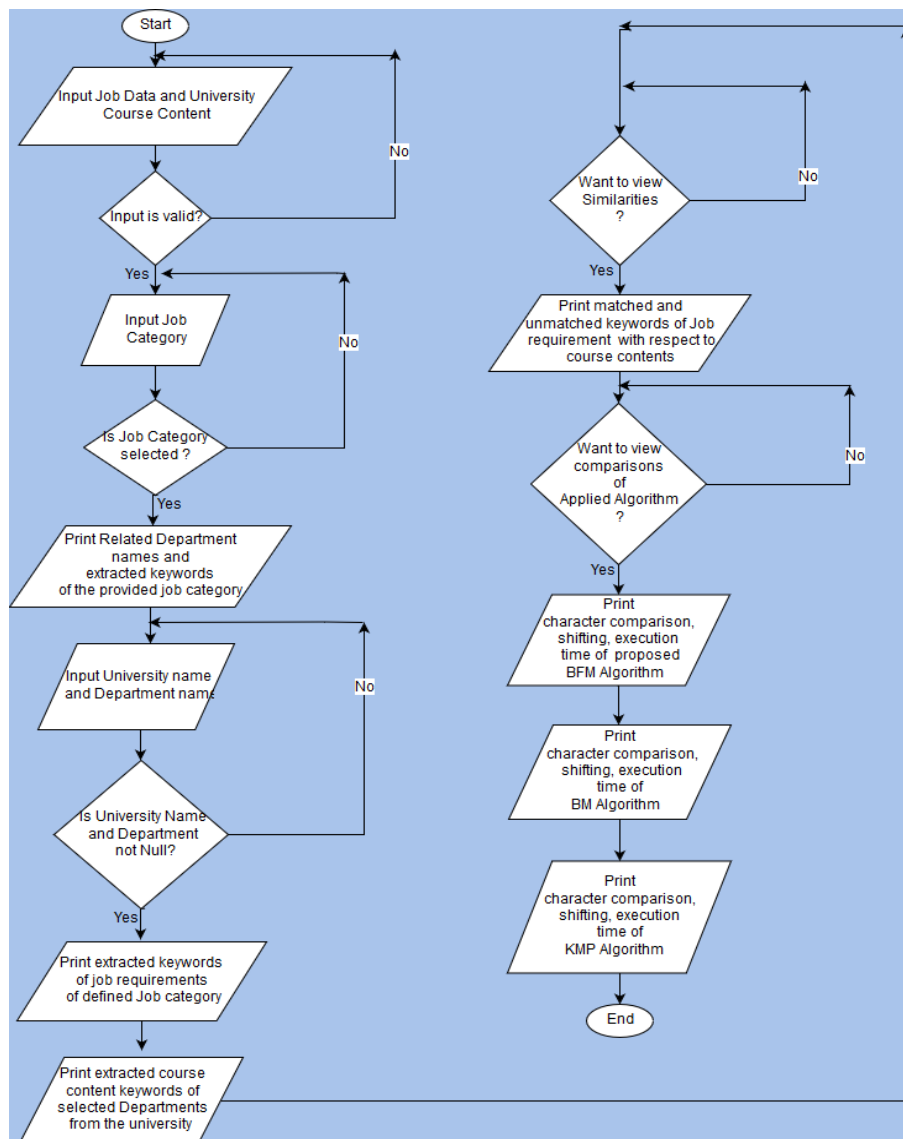


Figure 4.9: Flowchart of System Implementation

Chapter 5

Result Analysis

For generating results we have used PC of following configuration:

Processor : Intel (R) core(TM) i5-3230M CPU @2.6 GHz

RAM : 8.00GB (7.90 usable)

System type : 64-bit Operation System , x64-based processor

OS: Windows 10

5.1 Evaluation of job Suggestion

We have collected University course content from East West University official site. Extracted keywords of course contents, along with the course name and department name are stored in the database.

Job data are collected from www.bdjobs.com named as Bdjobs official site . We collected data from 01 April 2018 to 11 April 2018 for the IT and telecommunication category. Then we applied NLTK to extracts nouns, thus generating the following extracted keywords.

Extracted Keyword Job :

graphic designing adobe illustrator adobe photoshop adobe indesign dreamweaver adobe flash design related google analytics seo google ad , ios swift cocoa ios mac os html css javascript ui/ux ios jira trello , vm esx esxi virtual center scsi fc iscsi sas advanced esx openstack/vmware configuration vmware vmware heartbeat auto host

, mobile java development ios tomcat apache dynamic , minimum practical , digital marketing facebook google video youtube photoshop illustrator graphics design , cisco network security routing cisco security technologies content security firewall vpn ids ips aaa source fire firewalls vpns ips , agile scrum master certification , young expert experienced dynamic quick learner punctual team members cloud computing lamp setting maintaining explain , ericsson bss development sound ericson cbio sound java telco billing sound shell rdbms linux unix/ linux php/ java bscs api , fair , computer hardware/ network hosting software service , mobile , html css php excellent , laravel php mysql laravel json rest api linux os css javascript bootstrap enterprise laravel laravel jira trello , ui/ux expert acceptance neoload selenium ide selenium webdriver protractor android/iphone/ipad uat sql chrome dev english , javascript , payroll sql data , english report learning centre semi structural learning centre mobile influencing english , asp.net , css javascript jquery xhtml cms end software architect technical software , php wordpress html css bootstrap javascript jquery ajax json php mysql seo work open cms , english , software marketing advertising promotion support/ client service market research software fluent english smart honest sincere oriented challenge custer service , data center telecommunication iig isp sound data , aws azure cloud platform docker microservices oracle sql visual studio oracle sql html javascript css team foundation crystal reports enterprise build deployment github maven/gradle jenkins agile development enterprise architecture/systems , jquery , graphic designing logos adobe illustrator adobe photoshop adobe indesign dreamweaver adobe flash sketch capable all kinds design related projects ui good google analytics seo web web google ad email sms excellent motivated quick project , javascript jquery asp.net , oracle object oriented html css

Then we took East West University course contents from their website for CSE departments and applied NLTK noun extraction process to generate the following keywords:

Extracted Keyword University :

software quality quality assurance quality control sqa cmm cmmi software quality review formal cost software stlc software test software software white black test pair path boundary decision design designing technical iot basics networking communication protocols sensor networks communications interoperability iot actuators implementation iotwith software networking sdn data handling analytics cloud computing fog computing smart homes connected vehicles smart grid industrial iot healthcare activity monitoring structures abstract data type implementation stack implementation application queue implementation application iterative solution recursive solution basic tree concepts tree traversals binary trees binary search trees insert delete search traversal algorithms avl tree binary heap priority queue graph terminology graph tree mst shortest path problem.hashing software engineering software sdlc overview software process models incremental agile software development agile ux lean ux extreme scrum software requirement requirement questionnaire designing software uml behavioral use interaction code halstead project functional point analysis fp basics- uat integration system testing data introduction protocol architecture osi tcp/ip analog digital transmission transmission channel shannons guided wireless signal synchronous transmission asynchronous transmission interfacing types error crc error hamming flow hdlc arq arq arq multiplexing fdm tdm wdm mobile mobile software mobile design end networking overview wireless communication networking mobile computing historical adaptation wireless channel rayleigh coherence frequency tracking multiple access techniques tdma fdma cdma aloha slotted-aloha csma/ca maca lan ieee ieee wpan ieee satellite hidden request rts cts network nav convex polygon orthogonal point voronoi delaunay linear randomized graph computer computer pci bus computer pentium powerpc computer dram computer riad input/output i/o programmed i/o interrupt-driven i/o input/output i/o computer arithmetic alu interger instruction machine instruction cpu cpu risc computer information ethics stanford encyclopedia philosophy acm code ethics professional conduct software engineering

code ethics professional practice data protection act computer misuse act impact computer misuse act copyright designs patents act freedom information act security internet communications bangladesh communication technology act bangladesh copyright act bangladesh telecommunication regulatory act pornography act ethical virtue virtue analysis evaluation design service tos end user license agreement eula privacy policy finite automata regular expressions nondeterminism properties regular languages context-free grammars pushdown automata grammars equivalences properties context-free languages turing machines variations turing machines decidable problems undecidability time complexity space complexity np-completeness micro types architecture design representation documentation reuse case study database management systems sql structured query language using e-r model database indexing hashing techniques basic database normalization lossy decomposition functional dependency first form boyce-codd normal form vlsi ic cmos circuits dc logical combinational arithmetic cmos vlsi data memories vlsi time distributed remote global distributed deadlock check consensus line ellipse clipping geometric graphics opengl input interaction opengl color rgb cmy hls opengl rgb indexed generate opengl curve image mobile phones network technologies android programming android application frameworks building simple user interface activities intents services broadcast receivers data persistence processes threads asynchronous tasks internet resources apps publishing business models objective-c application concepts dc circuit law kirchhoff voltage law kirchhoff current law series-parallel voltage current division wye-delta transformations circuit analysis methods nodal mesh linearity superposition source transformation thevenin norton maximum concepts ac sinusoids phasors phasor circuit elements impedance kirchhoff frequency impedance combinations superposition source transformation thevenin norton nodal mesh analysis instantaneous maximum effective rms complex design design design compiler lexical analyzer regular expression transition diagram finite automata nfa regular expression nfa dfa nfa dfa subset construction dfa context free grammar ambiguity left recursion top down parsing bottom up parsing se-

mantic analysis run time environment code generation optimization diode rectifier diode clipper and clamper characteristics op comparator op voltage adder difference integrator differentiator design device structure physical operation bjt modes operation current-voltage characteristics bjt dc bjt device structure physical operation mosfet modes operation current-voltage characteristics channel length modulation effect mosfet amplifier switch dc mosfet small mosfet designing mosfet data information root finding methods bracketing methods root finding methods open-end methods introduction solution direct curve line interpolation overview overview channel power trunking cell cell microcell big data big data skills sources big data characteristics big data four v key big data platform storage analytics governance big data data data science relational databases sql data cleansing preparation data summarization visualization descriptive statistics correlation association analysis cluster analysis linear regression principles classification decision trees linear classifiers neural networks r introduction hadoop hadoop mapreduce/pig/hive/hbase cloud big data web fundamentals programming languages web html basics php html php css database php dynamic asp.net ajax dhtml security artificial intelligence intelligent agents uniformed search strategies informed search strategies constraint satisfaction problem games games game alpha-beta introduction genetic algorithm ga terminology first-order logic knowledge engineering first-order logic planning planning problem planning algorithms game theory nash equilibrium mixed strategy uncertainty uncertainty basic probability notation bayes rule convolutional neural network convolution layer pooling layer fully connected layer probabilistic reasoning time hidden markov models kalman filters learning observations knowledge learning statistical learning methods reinforcement learning data mining data mining goals stages data mining process data mining techniques knowledge representation methods data data data data data association correlation classification basic decision prediction statistical linear clustering basic first hierarchical conceptual advanced data mining text web assignment mini project report presentation counting hall cuts network vertex mycielski chromatic euler kuratowski data

protocol mac channel csma/cd contention beb csma introduction link hierarchical broadcast multicast internet protocol ip ipv4 ip network nat icmp arp rarp bootp dhcp qos rsvp internetworking congestion transport tcp/udp congestion transport hardware interaction process inter process communication ipc mutual memory i/o storage management implementing file management digital image visual perception light electromagnetic spectrum image sensing acquisition image sampling quantization pixels linear nonlinear operations image enhancement gray level transformations histogram processing basics spatial filtering filters color color models pseudocolor image processing basics full-color image processing color transformations smoothing sharpening color segmentation image discontinuities edge linking boundary detection thresholding segmentation segmentation morphological watersheds morphological image processing erosion opening closing extensions gray-scale images video lossy image jpeg video mpeg object recognition learning recognition bagof-wordsmodel review random number generators generating random variates output data analysis single system verification/validation assignments mini projects humancomputer interaction human capabilities computer interaction paradigms hci design process design basics hci software process design rules universal design implementation support tools users models issues stakeholder requirements evaluation user support user support task models dialogs analyzing tasks dialog notations design augmented reality hypertext multimedia groupware computer-supported collaborative work ubiquitous computing virtual reality augmented reality hypertext multimedia world wide web biological sequence msa blast fasta protein gene protein machine learning learning setup linear prediction decision tree logistic regression probabilistic modeling method nave bias learning apriori principal hierarchical artificial neural networks mlps deep learning support vector machines reinforcement view auto cad drafting modify commands auto cad drafting isometric isometric projection/view.introduction lines lettering dimensioning plain diagonal vernier scale orthographic projection first third oblique lines traces projection lines polygonal lamina circular lamina cube prism pyramid cylinder cone suspended solids isometric lines planes

scale embedded architecture device drivers develop sequential data flow state concurrent testing simulation debugging task design case graphs bfs dfs dfs topological single dijkstra bellman-ford dag co3 algorithm divide conquer closest counting greedy coin knapsack huffman optimal activity dynamic dp memorized lis knapsack longest lcs rock network flow max flow min-cut residual network augmenting ford-fulkerson edmonds-karp euclid gcd extended euclid number recurrence iteration substitution recursion master pattern rabin-karp strings p np binary binary boolean logic minimization boolean k-map combinational design design design design combinational verilog hdl flip-flops representation design design design design verilog hdl c programming information system analysis design project system requirements modeling feasibility analysis application architecture modeling input-output user interface object-oriented design modeling set combinations discrete probability algorithm growth relations graph algebraic project well-known social cryptography authentication public key infrastructure ipsec vpns e-commerce attack security security intrusion detection prevention response containment digital disaster recovery network vpns intrusion detection memory i/o architecture microprocessor registers addressing modesof microprocessor hardware specificationsof microprocessor memory interfacing interrupts embedded processors structure functions microcontroller addressing modes microcontrollers programming microcontroller assembly language interfacing microcontroller peripherals interrupts microcontroller robotics robot mechanical structure kinematics actuators sensors trajectory planning motion planning control architecture motion control force control visual servoing

Then we considered each words of extracted job keyword list as “pattern” and university course extracted keywords list as “text”. And finally by applying each of the algorithms-BFM, KMP, and BMH separately we found out whether there is a match between job and university extracted keywords. Based on the result obtained, we plot the following graph to show the performance comparison of the applied algorithms:

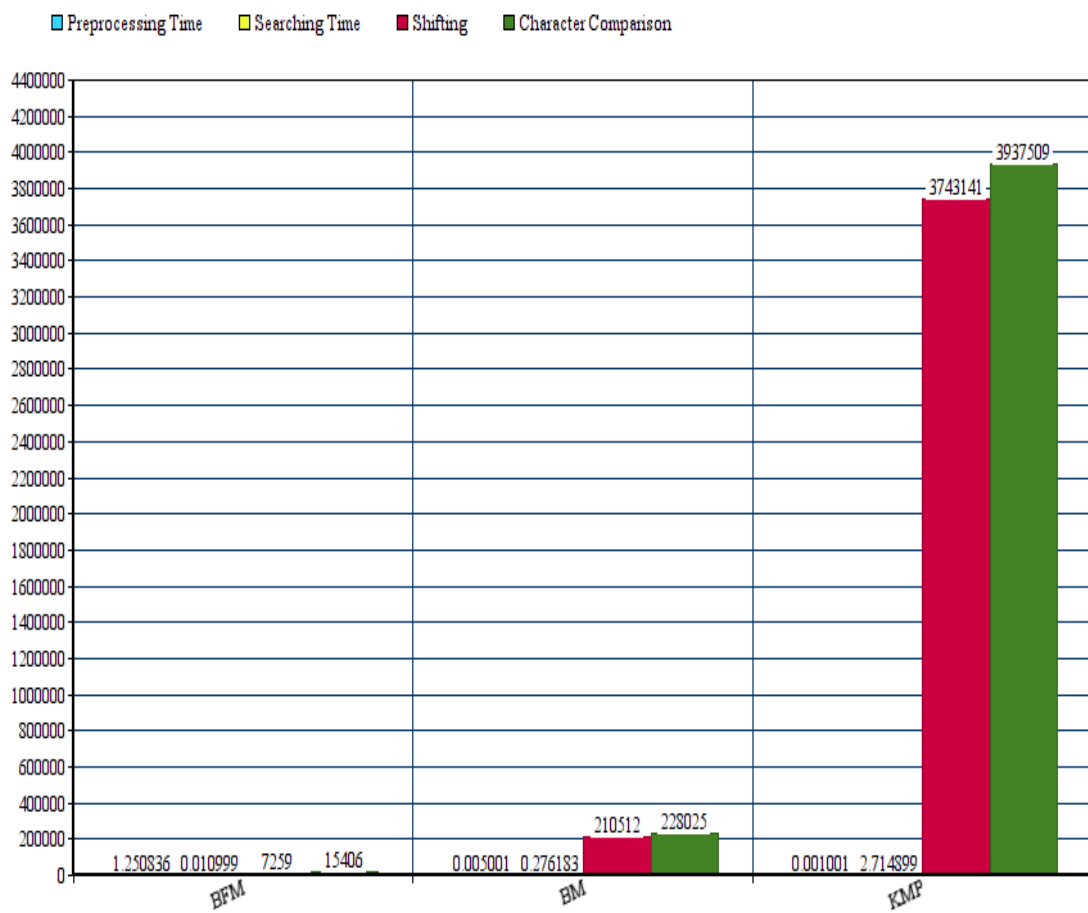


Figure 5.1: Graph of Result analysis of applied algorithms

Here, in the Figure 5.1 we have shown a comparative study of BFM, BMH, and KMP algorithm based on preprocessing time, searching time, number of shifts and number of character comparisons.

We have also found out execution time which is illustrated in the follow graph:

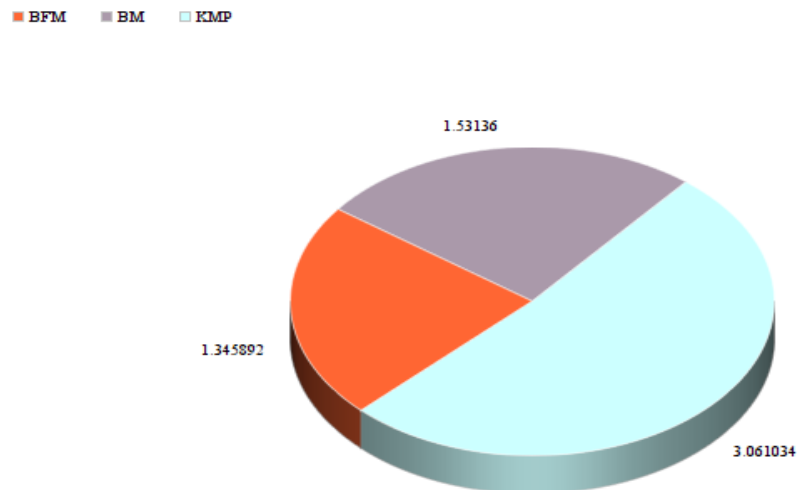


Figure 5.2: Pie chart of execution time

5.2 Evaluation of Algorithm with Real Data

To evaluate the performance of *BFM*, we've used the large corpus available at the Canterbury Corpus [24]. We've used the "World192.txt" file with 1,905,891 characters and the "bible.txt" file with 3,250,898 characters files for our purpose. Along with our algorithm *BFM*, we evaluated both *BMH* and *KMP* algorithms to compare our algorithm's performance on both files.

As in many previous works, the total no. of comparisons and shifts needed to find a pattern is considered as the measures of an algorithm's quality [13], we are using these

two criteria to evaluate our algorithm. The first evaluation was done on “World192.txt” file, to find the pattern “Bangladesh”. This pattern appears 9 times in file. Total shifts and comparisons required for the task is displayed in figure 5.3 and 5.4. The execution time taken to accomplish the search is given in figure 5.5.

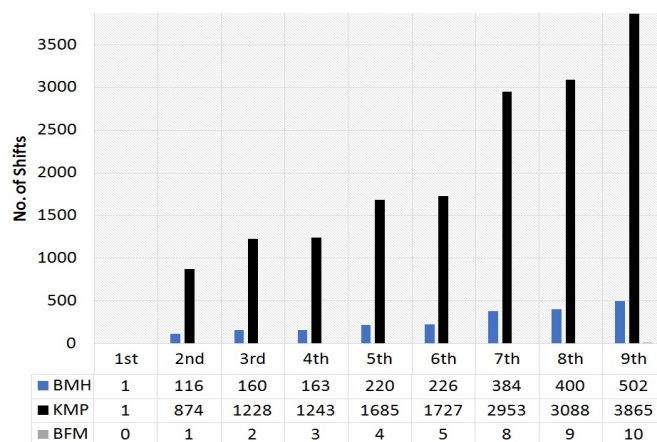


Figure 5.3: Total number of shifts to search “Bangladesh”

From these result it can be said that the number of shifts, comparison and execution time of our algorithm *BFM* are very much lower than *BMH* and *KMP*.

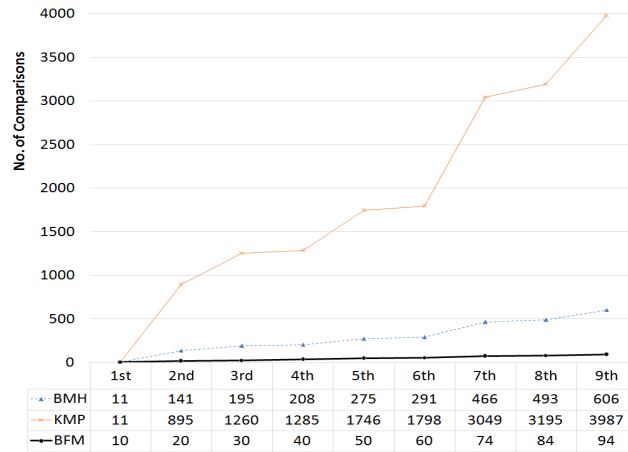


Figure 5.4: Total number of comparisons made to search “Bangladesh”

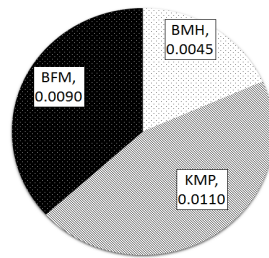


Figure 5.5: Execution time (sec) required to search “Bangladesh”

To evaluate *BFM* algorithms perform with respect to the file size, we applied all three algorithms on the “bible.txt” file. On this file we search for the word “God” as this word has preferably many occurrences throughout the file. First of all we considered the total character size of the file and then we decrease the character size of the search text and perform the search again on it. The result of the process is displayed in 5.1. This analysis also shows our proposed algorithm is performing better than the other algorithms.

Table 5.1: Performance analysis on different character sized text

Char Size	3250898	1577526	677382	416172	242632
No. of shifts					
BFM	8694136	8694136	1759121	552104	195329
BMH	3.1E+09	3.1E+09	2.4E+09	9.8E+07	3.2E+07
KMP	8.9E+09	8.9E+09	2.4E+09	2.8E+08	9.2E+07
No. of Comparisons					
BFM	3.4E+07	3.4E+07	6946292	2211962	783158
BMH	3.3E+09	3.3E+09	2.4E+09	1E+08	3.4E+07
KMP	8.9E+09	8.9E+09	2.4E+09	2.8E+08	9.3E+07
Execution Time (s)					
BFM	0.0625	0.0313	0.0313	0.0313	0.0156
BMH	1.0157	2.1721	0.8595	0.2031	0.0938
KMP	1.9221	1.9846	1.2189	0.1719	0.1094

Chapter 6

Conclusion and Future Works

6.1 Conclusion

The *Back and Forth Matching (BFM)* algorithm is preprocessing a text beforehand by applying *forward* and *backward* matching to match both the first and last characters of the pattern. This preprocessing task enhances the searching by enabling the algorithm to search only on the indexed positions, thus decreasing both the number of shifting and character comparisons. One limitation of this algorithm is if the text and pattern both are small, then *BFM's* performance degrades, as there is a preprocessing phase to complete before searching. Nevertheless, we found the performance of *BFM* in respect to execution time, no. of shifts and comparisons to be exceptionally high in contrast to *KMP* and *BMH*. Hence, this algorithm would certainly an efficient choice in case of the tasks that require huge amount of text searches. For this, considering our algorithm for building up applications other than the one applied in this thesis, will obviously result in good performance.

6.2 Future Works

In future, we have a vision to take the challenge in addressing following issues:

- Here, in our implementation we have included only one university. So there is a scope of applying the proposed algorithm for multiple universities.

- We will also take the challenge in including our project methodology for more job portals. Thus can compare our effectiveness.
- We will also use machine learning techniques to reduce amount of garbages to give more accurate suggestions.

Bibliography

- [1] D. Gurung, U. K. Chakraborty, and P. Sharma, “Intelligent predictive string search algorithm,” *Procedia Computer Science*, vol. 79, pp. 161–169, 2016.
- [2] N. Singla and D. Garg, “String matching algorithms and their applicability in various applications,” *International journal of soft computing and engineering*, vol. 1, no. 6, pp. 218–222, 2012.
- [3] R. C. Roistacher, “On-line computer text processing: A tutorial,” *Behavior Research Methods*, vol. 6, no. 2, pp. 159–166, 1974.
- [4] S. S. M. Al-Dabbagh, M. A. S. Naser, and N. H. Barnouti, “Fast hybrid string matching algorithm based on the quick-skip and tuned boyer-moore algorithms,” *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 6, pp. 117–127, 2017.
- [5] C. Charras, T. Lecrog, and J. D. Pehoushek, “A very fast string matching algorithm for small alphabets and long patterns,” in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 1998, pp. 55–64.
- [6] A. Rasool, N. Khare, H. Arora, A. Varshney, and G. Kumar, “Multithreaded implementation of hybrid string matching algorithm,” *International Journal on Computer Science and Engineering*, vol. 4, no. 3, p. 438, 2012.

-
- [7] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM journal on computing*, vol. 6, no. 2, pp. 323–350, 1977.
- [8] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.
- [9] R. N. Horspool, "Practical fast searching in strings," *Software: Practice and Experience*, vol. 10, no. 6, pp. 501–506, 1980.
- [10] A. F. Klaib, Z. Zainol, N. H. Ahamed, R. Ahmad, and W. Hussin, "Application of exact string matching algorithms towards smiles representation of chemical structure," *International journal of computer and information science and engineering*, vol. 1, pp. 235–239, 2007.
- [11] M. A. S. Naser, M. F. Aboalmaaly *et al.*, "Quick-skip search hybrid algorithm for the exact string matching problem," *International Journal of Computer Theory and Engineering*, vol. 4, no. 2, p. 259, 2012.
- [12] C. S. Rao, K. B. Raju, and S. V. Raju, "Parallel string matching with multi core processors-a comparative study for gene sequences," *Global Journal of Computer Science and Technology*, 2013.
- [13] R. Y. Tsarev, A. Chernigovskiy, E. Tsareva, V. Brezitskaya, A. Y. Nikiforov, and N. Smirnov, "Combined string searching algorithm based on knuth-morris-pratt and boyer-moore algorithms," in *IOP Conference Series: Materials Science and Engineering*, vol. 122, no. 1. IOP Publishing, 2016, p. 012034.
- [14] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.

-
- [15] P. Shah and R. Oza, "Improved parallel rabin-karp algorithm using compute unified device architecture," in *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, 2017, pp. 236–244.
- [16] J. Bhandari, "String matching rules used by variants of boyer-moore algorithm," *Journal of Global Research in Computer Science*, vol. 5, no. 1, pp. 8–11, 2014.
- [17] R. Sharma, V. Gupta, and V. Kumar, "Efficient parameterized string matching algorithm," *International Journal of Emerging Research in Management and Technology (IJERMT)*, 2015.
- [18] J. Tarhio and E. Ukkonen, "Approximate boyer-moore string matching," *SIAM Journal on Computing*, vol. 22, no. 2, pp. 243–260, 1993.
- [19] M. Sahli and T. Shibuya, "Max-shift bm and max-shift horspool: Practical fast exact string matching algorithms," *Journal of Information Processing*, vol. 20, no. 2, pp. 419–425, 2012.
- [20] T. Raita, "Tuning the boyer-moore-horspool string searching algorithm," *Software: Practice and Experience*, vol. 22, no. 10, pp. 879–884, 1992.
- [21] R. Rahim, I. Zulkarnain, and H. Jaya, "A review: search visualization with knuth morris pratt algorithm," in *IOP Conference Series: Materials Science and Engineering*, vol. 237, no. 1. IOP Publishing, 2017, p. 012026.
- [22] I. M. Abu-Zaid and E. K. El-Rayyes, "Parallel search using kmp algorithm in arabic string," *International Journal of Science and Technology*, vol. 2, no. 7, 2012.
- [23] R. Janani and S. Vijayarani, "An efficient text pattern matching algorithm for retrieving information from desktop," *Indian Journal of Science and Technology*, vol. 9, no. 43, 2016.

- [24] M. Powell, "The canterbury corpus," Jul 2007,
<http://corpus.canterbury.ac.nz/descriptions>, accessed 2018-02-15.

Appendix A

Source Code

1. BFM Code

```
import math
import time

def search_Func ( txt , pat , n , m , posIndex ) :
    start=time.time ()
    if m%2==0:
        midPoint=int ( m / 2 )
    else :
        midPoint=math.floor ( m / 2 ) + 1

    length=len ( posIndex )
    flag=0
    shift=0
    charComparison = 0
    for pos in posIndex :
        k=pos
        l=0
        p=m+k-1
```

```
LastCharPat=m-1
count=0
while k<=p:
    charComparison = charComparison + 2
    if txt[k+1]==pat[l+1] and txt[p-1]==pat[LastCharPat-1]:
        k+=1
        l+=1
        p-=1
        LastCharPat-=1
        count+=1
    else:
        break
if count==midPoint:
    flag=1
    print("Match_found_at_position: {}".format(pos))
    print("Total_Shift: {}".format(shift))
    print("Total_Character_Comparison: {}".format(charComparison))
    shift=shift+1

if flag==0:
    print("No_match_found")
end=time.time()
print("Total_Execution_time_in_searching_phase: ", end-start)

def preProcess_Func(txt, pat):
    start=time.time()
    n=len(txt)
```

```
m=len(pat)
if m==1:
    return
i=0
posIndex=[]
while i<=(n-m):
    if txt[i]==pat[0]:
        if txt[i+m-1]==pat[m-1]:
            posIndex.append(i)
        i+=1
end=time.time()
print("Execution time in Preprocessing Phase: ", end-start)
search_Func(txt,pat,n,m,posIndex)

def input_Func():
    while 1:
        txt=input("Enter the text: ")
        pat = input("Enter the pattern: ")
        preProcess_Func(txt,pat)

input_Func()
```

2. BMH Code

```
import time
```

```
NO_OF_CHARS = 256
```

```
def badCharHeuristic(string , size):
    start=time.time()

    badChar = [-1]*NO_OF_CHARS

    for i in range(size):
        badChar[ord(string[i])] = i;
    end=time.time()
    print(" Execution_time_in_preprocessing_Phase:_", end-start)
    return badChar

def search(txt , pat):
    m = len(pat)
    badChar = badCharHeuristic(pat , m)

    start=time.time()
    n = len(txt)
    pos = 0
    shift=0
    charComparison=0
    while(pos <= n-m):
        j = m-1

        while j>=0 and pat[j] == txt[pos+j]:
            charComparison=charComparison+1
            j -= 1
```



```
        if j<0:
            print("Pattern occur at position {}".format(pos))
            print("Total shift {}".format(shift))
            print("Total Character Comparison {}".format(charComparison))

            pos += (m-badChar[ord(txt[pos+m])] if pos+m<n else 1)
            charComparison = charComparison + 1
        else:
            pos += max(1, j-badChar[ord(txt[pos+j])])
            charComparison = charComparison + 1
        shift = shift + 1

end=time.time()
print("Execution time in searching phase: ",end-start)

def input_Func():
    while 1:
        txt=input("Enter TXT: ")
        pat=input("Enter PAT: ")
        search(txt,pat)

input_Func()

3. KMP Code

import time
def PrefixArrayCreation(pat,m,preArray):
    start=time.time()
```

```
j=0
i=1
preArray[0]=0
while i<m:
    if pat[i]==pat[j]:
        preArray[i]=j+1
        j=j+1
        i=i+1
    else:
        if j!=0:
            j=preArray[j-1]
        else:
            preArray[i]=0
            i=i+1
end=time.time()
print("Total_Execution_time_in_Preprocessing_phase:_", end-start)

def KMPSearch(text, pat):
    n=len(text)
    m=len(pat)
    preArray=[0]*m
    PrefixArrayCreation(pat, m, preArray)

    start = time.time()
    i=0
    j=0
    k=0
```

```
shift=0
charComparsion=0
while i<n:
    if text[i]==pat[j]:
        charComparsion=charComparsion+1
        i+=1
        j+=1
    if j==m:
        print("Pattern found at position {}".format(i-j))
        print("Total Shift: {}".format(shift))
        print("Total Character Comparison: {}".format(charComparsion))
        j=preArray[j-1]
        k+=1
    elif i<n and text[i]!=pat[j]:
        charComparsion=charComparsion+1
        if j!=0:
            j=preArray[j-1]
        else:
            i+=1
            shift=shift+1
    if k==0:
        print("No match found")
end=time.time()
print("Total Execution time in searching phase: ", end-start)

def inputFunc():
    while 1:
```

```
#text= input('Enter the text : ')
f=open(" bible.txt")
text=f.read()
f.close()
pat=input('Enter the pattern:_:_')
KMPSearch(text,pat);
```

```
inputFunc()
```

4. URL

```
from django.conf.urls import url
from . import views
```

```
urlpatterns = [
    url(r'^insertdata', views.insert_data ),
    url(r'^insertCourseData', views.insert_course_data ),
    url(r'^$', views.index , name='index'),
    url(r'^index', views.index , name='index'),
    url(r'^jobDataForm', views.job_data , name='job_data'),
    url(r'^course_content_form', views.course_data , name='course_data'),
    url(r'^selJobCategory', views.jobCategory_Select ,
        name='jobCategory_Select'),
    url(r'^selectUni', views.university_Select , name='university_Select'),
    url(r'^similarity', views.similarity , name='similarity'),
    url(r'^extractedKeywordJob', views.extractedKeywordJob ,
        name='extractedKeywordJob'),
    url(r'^extractedKeywordUniversity',
```

```
views.extractedKeywordUniversity ,
        name='extractedKeywordUniversity'),
url(r'^comparison', views.comparison , name='comparison'),

]
```

5. Models

```
from django.db import models

class UniversityCurriculum(models.Model):
    universityName=models.CharField(max_length=50)
    deptName=models.CharField(max_length=50)
    courseName=models.CharField(max_length=50)
    courseContent=models.TextField()
    courseKeyword=models.TextField(null=True)

    def __str__(self):
        return self.universityName

class jobData(models.Model):
    jobTitle=models.CharField(max_length=70)
    category=models.CharField(max_length=40)
    jobRequirments=models.TextField()
    relatedDept=models.CharField(max_length=50)
    keywords=models.TextField()
```



```
keys2 = jobData.objects.filter(category=jobCategory)

keywordList1 = []
keywordList2 = []

keywordSet1 = set()
keywordSet2 = set()
for data in keys1: # For University
    keywordList1 = data.courseKeyword.split()
    keywordSet1.add(data.courseKeyword)
for item in keywordList1:
    keywordSet1.add(item)

for data in keys2:
    keywordList2 = data.keywords.split()
    keywordSet2.add(data.keywords)
    #relatedDeptSet.add(data.relatedDept)
for item in keywordList2:
    keywordSet2.add(item)

keywordSet1 = list(keywordSet1)
keywordSet2 = list(keywordSet2)

matchedList = []
unmatchedList = []
#Text split()
```

```
#txt= ', '.join(str(s) for s in keywordSet1)
txt = ""
for s in keywordSet1:
    txt+=s+" ";
patt = ""
for s in keywordSet2:
    patt+=s+" ";

print(patt)

pattern = patt.split()
if 't' in request.session:
    del request.session['t']
if 'p' in request.session:
    del request.session['p']

request.session['t']= txt
request.session['p']= pattern

for pat in pattern:
    if len(pat)!= 1:
        isMatch = False
        charComparison, shift, preProcessTime, searchTime,
            isMatch = preProcess_Func(txt, pat)
        if isMatch == True :
            matchedList.append(pat)
        else :
```



```
unmatchedList.append(pat)

title1 = "Matched_Syllabus:_:"
title2 = "This_contents_are_suggested_to_add_on_your_syllabus:_:"

contextUniForSimilarityPage = { 'matchedKey' : matchedList ,
                                'unmatchedKey' : unmatchedList ,
                                'pattern' : pattern ,
                                'txt' : txt ,
                                'title1' : title1 ,
                                'title2' : title2
                                }

def insert_data(request):
    data = request.POST

    jobTitle = data['jTitle']
    category = data['catag']
    jobRequirments = data['jReq']
    relatedDept = data['RelatedDept']

    jobReqTxt = data['jReq']
    import nltk
    from nltk import word_tokenize
```

```
import os

java_path = "C:/Program_Files/Java/jdk1.8.0_60/bin/java.exe"
os.environ['JAVA_HOME'] = java_path

nltk.internals.config_java('C:/Program_Files/Java
...../jdk1.8.0_60/bin/java.exe')

from nltk.tag import StanfordPOSTagger

jar = 'C:/Users/Pc/PycharmProjects/new/
.....stanford-postagger-2016-10-31/stanford-postagger.jar'
model = 'C:/Users/Pc/PycharmProjects/new/stanford-postagger-2016-10-31
...../models/english-left3words-distsim.tagger'
pos_tagger = StanfordPOSTagger(model, jar, encoding='utf8')

tokens = word_tokenize(jobReqTxt)
tagged = pos_tagger.tag(tokens) # changed
nouns = [word for word, pos in tagged \
         if (pos == 'NNP')] # changed
downcased = [x.lower() for x in nouns]
key1 = '_'.join(str(e) for e in downcased)

JobData = jobData(jobTitle=jobTitle, category=category,
                  jobRequirments=jobRequirments, relatedDept=relatedDept,
```

```
        keywords= key1 )

    JobData.save()

    return render(request, 'education_gap_analyzer/jobDataForm.html')

#
#
#
#

def insert_course_data(request):
    courseData = request.POST

    universityName = courseData['UniName']
    deptName = courseData['DeptName']
    courseName = courseData['courseName']
    courseContent = courseData['courseContent']

    try:
        import nltk
        tokens = nltk.word_tokenize(courseContent)
        tagged = nltk.pos_tag(tokens)
        nouns = [word for word, pos in tagged \
                 if (pos == 'NNP' or pos == 'NNPS')]
        downcased = [x.lower() for x in nouns]

        key1 = '_'.join(str(e) for e in downcased) #String Convert
```

```
except:
    print("Noun_Extraction_Problem.")

universityCurriculum =UniversityCurriculum( universityName=universityName ,
                                              deptName =deptName ,courseName=courseName ,
                                              courseContent=courseContent ,
                                              courseKeyword=key1 )

universityCurriculum.save()

return render(request , 'education_gap_analyzer/course_content_form.html')

#
#
#
#

def extractedKeywordJob(request):

    jCategory = request.POST.get('job')

    itTelecommunication = False
    accountingFinance = False
    engineerArchitect = False
    medicalPharma = False
    marketingSales = False
```

```
if jCategory == "IT&Telecommunication":
    itTelecommunication = True
elif jCategory == "AccountingFinance":
    accountingFinance = True
elif jCategory == "EngineerArchitect":
    engineerArchitect = True
elif jCategory == "MedicalPharma":
    medicalPharma = True
elif jCategory == "MarketingSales":
    marketingSales = True

keys = jobData.objects.filter(category=jCategory)

keywordsSet = set()
keywordList = []
relatedDeptSet = set()

for data in keys:
    keywordList = data.keywords.split()
    keywordsSet.add(data.keywords)
    relatedDeptSet.add(data.relatedDept)

for item in keywordList:
    keywordsSet.add(item)

title1 = "Related_Department:_:"
title2 = "Extracted_Keyword:_:"
title3 = "Selected_Category:_:"
```

```
context= { 'relatedDeptSet' : list(relatedDeptSet),
           'keywordsSet' : list(keywordsSet),
           'category' : jCategory,
           'it' : itTelecommunication,
           'accounting' : accountingFinance,
           'engineerArchitect' : engineerArchitect,
           'medicalPharma' : medicalPharma,
           'marketingSales' : marketingSales,
           'title1' : title1,
           'title2' : title2,
           'title3' : title3
         }

return render(request, 'education_gap_analyzer/selJobCategory.html',
              context)

#
#
#
#

def extractedKeywordUniversity(request):

    univ = request.POST.get('uni')
    dept = request.POST.get('dep')
    categ = request.POST.get('job')
```

```
request.session['university'] = univ
request.session['department'] = dept
request.session['category'] = categ

keys1 = UniversityCurriculum.objects.filter(universityName=univ)
        .filter(deptName=dept)
keys2 = jobData.objects.filter(category=categ)

keywordList1 = []
keywordList2 = []

keywordSet1 = set()
keywordSet2 = set()
for data in keys1: # For University
    keywordList1 = data.courseKeyword.split()
    keywordSet1.add(data.courseKeyword)
for item in keywordSet1:
    keywordSet1.add(item)

for data in keys2: # For Job
    keywordList2 = data.keywords.split()
    keywordSet2.add(data.keywords)
for item in keywordList2:
    keywordSet2.add(item)

title1 = "Extracted_Keyword_University_:_:"
title2 = "Extracted_Keyword_Job_:_:"
```

```
title3 = "Category : :"
```

```
contextUni= { 'keywordSet1' : list(keywordSet1),
              'keywordSet2': list(keywordSet2),
              'univ' : univ ,
              'dept' : dept ,
              'categ' : categ ,
              'title1' : title1 ,
              'title2' : title2 ,
              'title3' : title3
            }
return render(request, 'education_gap_analyzer/selectUni.html',
              contextUni)
```

```
#
#
#
#
```

```
def comparison(request):

    txt = ""
    pattern = ""
    if 't' in request.session:
        txt = request.session['t']
    if 'p' in request.session:
        pattern = request.session['p']
```



```
title1 = ""
title2 = ""
title3 = ""
title4 = ""
title5 = ""
title6 = ""

# For BMF

totalPreprocessingTimeOur = 0
totalSearchingTimeOur = 0
totalShiftOur = 0
totalCharacterComparisonOur = 0

matchCountOur = 0
unmatchCountOur = 0

for pat in pattern:
    if len(pat) != 1:
        isMatch = False
        charComparison, shift, preProcessTime, searchTime,
            isMatch = preProcess_Func(txt, pat)
        totalCharacterComparisonOur += charComparison
        totalShiftOur += shift
        totalPreprocessingTimeOur += preProcessTime
        totalSearchingTimeOur += searchTime
```

```
        if isMatch == True :
            matchCountOur += 1
        else :
            unmatchedCountOur += 1

totalPreprocessingTimeBM = 0
totalSearchingTimeBM = 0
totalShiftBM = 0
totalCharacterComparisonBM = 0

matchCountBM = 0
unmatchCountBM = 0
charComparison = 0
shift = 0
preProcessTime = 0
searchTime = 0
for pat in pattern:
    if len(pat) != 1:

        isMatch = False
        charComparison, shift, preProcessTime, searchTime,
            isMatch = BMsearch(txt, pat)
        totalCharacterComparisonBM += charComparison
        totalShiftBM += shift
        totalPreprocessingTimeBM += preProcessTime
        totalSearchingTimeBM += searchTime
```

```
        if isMatch == True :
            matchCountBM += 1
        else :
            unmatchedCountBM += 1

# For KMP
totalPreprocessingTimeKMP = 0
totalSearchingTimeKMP = 0
totalShiftKMP = 0
totalCharacterComparisonKMP = 0

matchCountKMP = 0
unmatchedCountKMP = 0
charComparison = 0
shift = 0
preProcessTime = 0
searchTime = 0
for pat in pattern:
    if len(pat) != 1:
        isMatch = False
        charComparison, shift, preProcessTime, searchTime,
            isMatch = KMPSearch(txt, pat)
        totalCharacterComparisonKMP += charComparison
        totalShiftKMP += shift
        totalPreprocessingTimeKMP += preProcessTime
        totalSearchingTimeKMP += searchTime
    if isMatch == True :
```

```
        matchCountKMP += 1
    else :
        unmatchedCountKMP += 1

contextComparsion = { 'matchCountOur' : matchCountOur ,
                    'unmatchCountOur' : unmatchedCountOur ,
                    'searchTimeOur' : totalSearchingTimeOur ,
                    'preprocessTimeOur' : totalPreprocessingTimeOur ,
                    'shiftComparisonOur' : totalShiftOur ,
                    'charComparisonOur' : totalCharacterComparisonOur ,
                    # BM
                    'matchCountBM' : matchCountBM ,
                    'unmatchCountBM' : unmatchedCountBM ,
                    'searchTimeBM' : totalSearchingTimeBM ,
                    'preprocessTimeBM' : totalPreprocessingTimeBM ,
                    'shiftComparisonBM' : totalShiftBM ,
                    'charComparisonBM' : totalCharacterComparisonBM ,
                    #KMP
                    'matchCountKMP' : matchCountKMP ,
                    'unmatchCountKMP' : unmatchedCountKMP ,
                    'searchTimeKMP' : totalSearchingTimeKMP ,
                    'preprocessTimeKMP' : totalPreprocessingTimeKMP ,
                    'shiftComparisonKMP' : totalShiftKMP ,
                    'charComparisonKMP' : totalCharacterComparisonKMP ,

                    'pattern' : pattern ,
                    'txt' : txt ,
```

```
        'title1': title1 ,
        'title2': title2 ,
        'title3': title3 ,
        'title4': title4 ,
        'title5': title5 ,
        'title6': title6
    }

    return render(request, 'education_gap_analyzer/comparison.html' ,
                  contextComparsion)
```

Appendix B

List of Publications

International Conference Paper

1. MD. Obaidullah Al-Faruk, K. M. Akib Hussain, MD. Adnan Shahriar, Shakila Mahjabin Tonni, (“BFM: A Forward Backward String Matching Algorithm with Improved Shifting for Information Retrieval”), *IEEE International Conference on Recent Trends in Computer Science and Technology, (ICRTCST-2018)*, Kolkata, India, 2018.