جـــامـعـــة الـــشرق الأوسـط
MIDDLE EAST UNIVERSITY

An Enhanced Boyer- Moore Algorithm

تحسين خوارزمية بوير - مور

By

Mu'ath Mousa Al-Mahasneh

Supervisor

Dr. Maamoun K. Ahmad

Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Science

Department of Computer Science

Faculty of Information Technology

Middle East University

June, 2014

Authorization form

إقرار تفويض

أنا معاذ موسى المحاسنه أفوض جامعة الشرق الأوسط للدراسات العليا بتزويد نسخ من رسالتي للمكتبات ، أو الهيئات أو الأفراد عند طلبها .

الاسم : معاذ موسى حمدان المحاسنه

التاريخ : ....2/6/2014**........**

التوقيع : [illegible]

Authorization form

I, Mu'ath Mousa Al-Mahasneh , authorize Middle East University to supply hardcopies and electronic copies of my thesis to libraries, establishments, or bodies and institutions concerned with research and scientific studies upon request, according to the university regulations.

Name : Mu'ath Mousa Al-mahasneh

Date: 2/6/2014

Signature: [illegible]

Examination Committee Decision

This is to certify that the thesis entitled "**An Enhanced Boyer- Moore Algorithm** " was successfully defended and approved on. **2 / 6 / 2014**

Examination Committee Member	**Signature**
1- **Dr. Maamoun Ahmed** **Computer Science Department** **Middle East University**	[illegible]
2- **Prof. Mohammed AlHamid** **Department of Computer Information Systems** **Middle East University**	[illegible]
3- **Jehad Al Saadi** **Computer Science Department** **Arab Open University**	[illegible]

Acknowledgements

(وَإِذْ تَأَذَّنَ رَبُّكُمْ لَئِنْ شَكَرْتُمْ لَأَزِيدَنَّكُمْ وَلَئِنْ كَفَرْتُمْ إِنَّ عَذَابِي لَشَدِيدٌ) سورة إبراهيم (7)

First and foremost, all the praises and thanks be to Allah, Almighty, Who has taught man that he knew not and gave me the strength to complete this work.

I would like to express my sincere thank to Dr. Maamoun Ahmad for his continuous support, efforts, and dedication.

I also would like to thank the Information Technology Faculty members at the Middle East University .

Dedication

I would like to express my thanks to my lovely parents who supported in my masters and in all academic stages of my life. They are the light in my path.

&

My beloved Wife

&

My sons

&

My brothers and family and patience for their love and support.

Table of contents

List of tables

List of figures

تحسين خوارزمية بوير - مور

الطالب

معاذ موسى المحاسنه

المشرف

الدكتور : مأمون خالد احمد

(الملخص)

بلغ حجم المعلومات وعدد الوثائق المحفوظة على شكل نصوص ومكتوبة بكافة لغات العالم حدا لا يمكن إحصاؤه , ولا يمكن التنبؤ بالحد الذي قد تصل إليه مستقبلا , وأصبحت عملية البحث عن معلومة معينة وسط هذا الكم الهائل تحتاج الى وقت كبير. من هنا فان الضرورة تقتضي إيجاد وسائل بحث سريعة وغير تقليدية لتخدم هذا الغرض , والتي تعتمد على عدد قليل من المقارنات للأحرف , والقفزات الكبيرة التي تجريها على النص أثناء عملية البحث.

في هذه الدراسة ، فإننا نقترح خوارزمية مطابقة السلسلة التي ادت الى تحسين خوارزميات مطابقة النصوص،الخوارزمية المقترحة تقوم بفحص النص من كلا الجانبين في وقت واحد باستخدام اثنين من النوافذ كل نافذه تحتوي على عدد من الاحرف في النص المساوية لطول النمط ، تقوم كل من النافذتين بالتحرك على النص من الجهتين حتى يتم العثور على التواجد الأول من نمط أو حتى تصل إلى منتصف النص أي التقاطع بين النافذتين، كل عملية في هذه الخوارزمية المقترحة تعتمد على مرحلة التجهيزات التي تعطينا المعلومات الكافية اثناء البحث.

نعتمد في هذه الخوارزمية المقترحة على خوارزمية البوير – مور بالنسبة للنافذة اليسرى واما من جهة اليمين فاننا اعتمدنا على آلية البحث لخوارزمية Quick-Search

تظهر النتائج التجريبية أن الخوارزمية المقترحة عززت عملية مطابقة السلسلة من خلال تقليل عدد المقارنات التي تحدث ويهدف هذا التحسين إلى الحصول على زيادة الانتقائية لمثل هذا النوع من الخوارزميات وينعكس ذلك على أداء وفعالية البحث كما يقلل من الوقت الذي يقضيه في عملية البحث.

في هذه الدراسة تمت برمجة كل من الخوارزميات واعتماد لغة c# لاظهار النتائج , حيث تم قياس الوقت المستنفذ في عملية البحث داخل النصوص وتم احتساب سرعة الخوارزمية اثناء البحث ومقارنتها مع خوارزميتي BM و QS .

An Enhanced Boyer- Moore Algorithm

by

Mu'ath Mousa Al-Mahasneh

Supervised by

Dr. Maamoun K. Ahmad

Abstract

The volume of information and the number of computer documents has been increasing over the last years. Thus, there is an urgent need for finding new fast efficient and non-traditional searching methods. The fastest known traditional searching algorithm is the Boyer Moore algorithm, which depends on a small number of character comparisons, and large shifts that are performed on the text during search.

In this study, we propose a string matching algorithm which made an improvement on the pattern matching technique, the algorithm scans the text from both sides simultaneously using two windows; each window has a size that is equal to the pattern length. Both windows move in parallel over the text until the first occurrence of the pattern is found or until both windows reach the middle of the text or intersection between both of the windows, all the process in this algorithm depend on the preprocessing phase in the Boyer Moore algorithm BM and Quick Search QS algorithm on the left window and the right window respectively over the text. The experimental results show that the proposed algorithm BBQ algorithm has enhanced the process of pattern matching by reducing the number of comparisons performed. The best time case is calculated where m is the length of the pattern and n is length of the text.

All previous enhancements aims to getting increase the selectivity of the algorithm and it affects the performance and effectiveness as reduces the time spent in the search process.

The BBQ algorithms and some traditional algorithms were implemented and compared. The result shows that the performance of the BBQ algorithms is much better than of the traditional algorithms, including the Boyer-Moore algorithm.

Chapter one

Chapter One
Introduction

1.1 Preface

The algorithms are defined as a series of steps to resolve the problems ,some problems may take a long time to be solved, while others get solved in fractions of a second depending on many factors that affect the operations of the algorithms. On the other hand, analysis of algorithms is one of the hot topics in computer science because the complexity of an algorithm is the cost measured in running time, storage, or whatever units are relevant (Wilf, H, S, (1994) .

The string, whose occurrences are searched for is called the pattern; it can be a word, partial word, or even a text fragment. The document, which is examined during the search, is called the text. It commonly consists of some natural language, but it can be any kind of binary data in a computer. In computer science string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (patterns) are found within a larger string or (text). In other words; the goal of any string matching algorithm is to determine whether or not a match of a particular string exists within another (Zhang, M, et al, (2013)).

The string matching problem is to find all the occurrences of a given pattern (P) in a large text (T). One approach to string matching is linear searching, which means the text is not preprocessed. Thus these algorithms need to scan the text when searching, the second approach, indexed searching, tries to speed up searching by preprocessing the text and building a data structure that allows searching.

The search problem in the written texts for all occurrence of pattern are classical problem, and the string matching process used in many areas of computer science including text editing, data retrieval and symbol manipulation. Also it's important to a number of fields including computational biology, computer science, and mathematics (Wilf, H, S, (1994)). This problem has emerged clearly in the seventies of the last century, when the companies programming began produces software word processing, clerical and programming languages , and increased the need to find quick and effective solutions to this problem. It is also clear that the search for text of the basics of the process of retrieval of information in many fields, such as databases , especially when the database is irregular , or if the cost of indexing database of high Consuming in both time and space from the hard drive or memory (yamashita, et al, 1996, Hoffman and McCullough, 1971)

In computer science, the "search process" has been the subject of discussion and debate. There have been many research studies carried out by researchers that focus on the development and improvement of algorithms used in this area. Much of the recent research has focused on developing algorithms to increase the efficiency of using software tools and programs; algorithms design and analysis are the core of developing a successful computer programs.

1.2 Pattern matching

Pattern matching is one of the most important areas which have been studied in computer science, and it is generally divides into multiple pattern matching and single pattern matching algorithms(Bhukya et al, 2011).

In a standard problem, we are required to find all occurrences of the pattern in the given input text, known as single pattern matching. Suppose, if more than one pattern are matched against the given input text simultaneously, then it is known as, multiple pattern matching. Whereas single pattern matching algorithm is widely used in network security environments. In network security the pattern is a string indicating a network intrusion, attack, virus, and snort, spam or dirty network information, etc.

Multiple pattern matching can search multiple patterns in a text at the same time. It has a high performance and good practicability, and is more useful than the single pattern matching algorithms.

Many existing pattern matching algorithms are reviewed and classified in two categories Exact string matching algorithm and Inexact/approximate string matching algorithms. in exact pattern matching algorithm will find that whether the probability will lead to either successful or unsuccessful search. The problem can be stated as: Given a pattern p of length m and a string Text T of length n ($m \leq n$). Find all the occurrences of p in T. The matching needs to be exact, which means that the exact word or pattern is found. Some exact string matching algorithms are Naïve Brute force algorithm, Boyer-Moore algorithm, KMP Algorithm. Inexact/Approximate pattern matching is sometimes referred as approximate string matching or matches with k mismatches/ differences. This problem in general can be stated as: Given a pattern P of length m and string/text T of length n ($m \leq n$). Find all the occurrences of sub string X in T that are similar to P, allowing a limited number, say k different characters in similar matches. The Edit/transformation operations are insertion, deletion and substitution.

In many cases most of the algorithm operates in two stages (Raju et al 2011), depending upon the algorithm some of the algorithm uses preprocessing phase and some algorithm will search without it. Some pattern matching algorithm concentrates on pattern itself. Other algorithm compare the corresponding characters of the patterns and text from the left to right and some other perform the character from the right to left. The performance of the algorithm can be measured based upon the specific order they are compared. Pattern matching algorithms has two different phases: Pre-processing phase or study of the pattern and Processing phase or searching phase. The pre-processing phase collects the full information and is used to optimize the number of comparisons. Whereas searching phase finds the pattern by the information collected in pre-processing. Pattern analysis plays a major part for various analysis like discrimination of the cancer from gene expression, mutation evolution, data analysis, feature extraction, searching, disease analysis, structural and functional analysis, e-books, text processing, linguistic translation, data compression, search engine, speech reorganization, information retrieval, genomic data, protein-protein interaction in cellular activities, computer virus detection, network intrusion detection, parsers, spam filters, digital libraries, screen scrapers, word processors, natural language processing and computational biology.

1.3 Algorithms Techniques

Every algorithm uses some special techniques to find pattern matching; following table shows the different techniques used by different algorithms:

"Table (1-1): string matching algorithms summary" (bhukya and somayajulu, 2010).

Algorithms	***Author***	***Comparison Order***	***Preprocessing***	***Searching time Complexity***
Boyer Moore	*R.S.Boyer And J.S.Moore*	*From right to left*	*Yes*	*O(mn)*
Horspool	*Nigel Horspool*	*Is not relevant*	*Yes*	*O(mn)*
Brute Force	-	*Is not relevant*	*No*	*O(mn)*
Kunth Morris Pratt	*Donald Knuth and Vaughan Pratt*	*From left to right*	*Yes*	*O(n+m) Independent from the alphabet size*
Quick Search	*Sunday*	*Is not relevant*	*Yes*	*O(mn)*
Karp Rabin	*Michael O Rabin And Richard M Karp*	*From left to right*	*Yes*	*O(mn)*
Zhu Takaoka	*R.F.Zhu And T Takaoka*	*From right to left*	*Yes*	*O(mn)*
Index Based	*IFBMPM Model*	*From left to rig*	*Yes*	*O(mn)*

1.4 Need of Pattern Matching

Pattern matching is the process of checking a perceived sequence of string for the presence of the constituents of some pattern. Pattern matching concept is used in many applications, the following figure shows the different applications. Such that web search engine amongst others application. Now a day's almost every people use the web application to get the desire results. But it is not necessary that peoples will only searching for text every time. They may want different type of data like audio, image and video. To handle such kind of data we need better method for searching. Pattern matching will help to find right and appropriate result. There are many algorithms used to find patterns matching (Diwate , Alaspurkar, 2013).

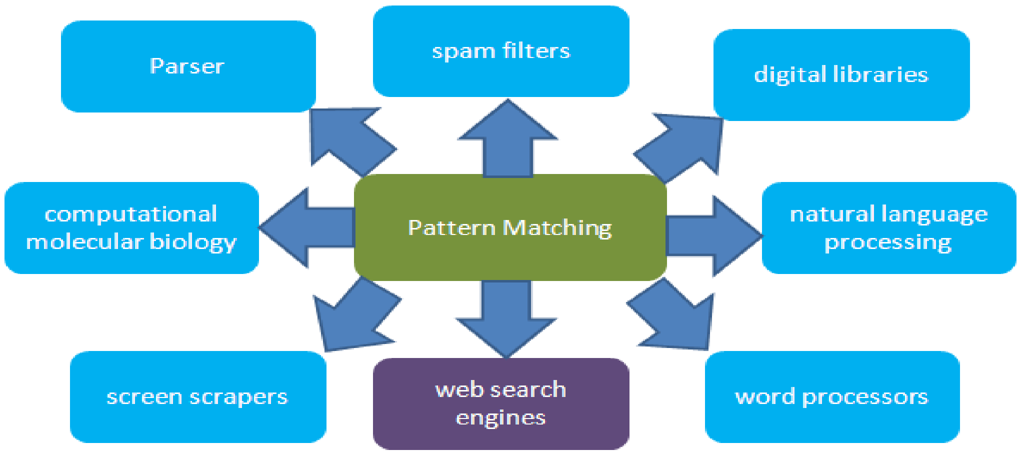


Figure (1-1): Applications of Pattern Matching

1.5 Basic definitions

Assume here that the alphabet that are elements (or characters) of both the text and the pattern. Any string or (pattern) will be represent ***P*** of length ***m***, and represent the input text ***y*** of length ***n*** , and assume that all of the text and the string are based on the group finished the letters of the alphabet and denoted by the symbol (**Σ**) and the length of the symbol(□), then we can build the following definitions :

- String (pattern): an ordered set of a single character or more, like (***abc***), and will consider the pattern as to be searched for in this research (Makinen, 2003).
- Text: an ordered set is made up of a string or more, such as: (***abcabadfdsdsd***).
- The word ***U***: is the prefix of the word ***W*** if and only if there is a word that ***V*** ***W* = *UV.***

- The word $\boldsymbol{V}$: is suffix for the word $\boldsymbol{W}$ if and only if there is a word that $\boldsymbol{U}$ $\boldsymbol{W = UV}$.
- Σ: alphabet, a set of characters here (***ASCII***).
- P: the string or pattern of length $\boldsymbol{m}$.
- Y: a text of length **n.**
- Window: a subtext (any part of the text) and equal to $\boldsymbol{Y_i \dots Y_{i+m}}$, where $\boldsymbol{o} \leq \boldsymbol{i} < \boldsymbol{n}$.
- Σ € $\boldsymbol{P_i}$ for each **i.**
- Σ € $\boldsymbol{Y_i}$ for each $\boldsymbol{i}$.

All the algorithms in this study aims to Find one or all occurrences of the pattern in the text, the match will happen if there was a subtext (window) $\boldsymbol{Y}$: $\boldsymbol{Y_i \dots Y_{i+m}}$, where: $\boldsymbol{o} \leq \boldsymbol{i} < \boldsymbol{n}$ and $\boldsymbol{P_i \dots P_m}$ equal $\boldsymbol{Y_i \dots Y_{i+m}}$ for each i (Charras and Lecroq, 2004).

1.6 Problem Definition

String matching is a problem of finding occurrence(s) of a pattern string within another string or body of text. The problem is also known as exact string matching, string searching, and text searching. The input pattern of $\boldsymbol{p}$ with size $\boldsymbol{m}$ and text body of $\boldsymbol{y}$ with size $\boldsymbol{n}$ where $m \leq n$.

Matching problem appears in many areas, for example the problem of finding occurrence(s) of a string in an english text, is the set of english alphabet; for the problem of finding occurrence(s) of a nucleotide in a DNA sequence in the area of genetic engineering, $\Sigma = \{a, c, t, g\}$; for the problem of finding occurrence(s) of an amino acid in a protein sequence, the problem is transformed to the binary matching, which is commonly used in the area of computer security. In general, string searching algorithms represented as the follow:

```
Algorithm string – searching
//find all occurrences of Pat [0, m-1] in Text [0, n-1]
{
Preprocessing phase;
Search phase;
Align Pat [0] with Text [0]
While (end of text is not reached)
{
Checking step;
Candidate checking phase;
Detailed comparison phase;
If an occurrence of pat has been found then
Report occurrence;
End if
Skipping step;
Move forward;
}
}
                                        (Moh'd Mhashi, Alwakeel 2010)
```

Figure (1- 2): string searching algorithms

The BM relies on searching text from right to left and building a table that contains a set of characters. The BM pattern matching algorithm depends on found the first occurrence in the large text to do match between them and this algorithm will be repeating the comparison between character in the pattern and text.

The worst case scenario of this algorithm is when it matches all the characters in the pattern from right to left with the text except the last character (the first character in the pattern) it will lead to increase the number of comparisons that will also waste the time spent on the search process.

Here; we have a simple example which explains the worst-case scenario in the BM algorithm

- ***Example of worst case scenario:***
- ***T =aaa … a***
- ***P = baaaaa***

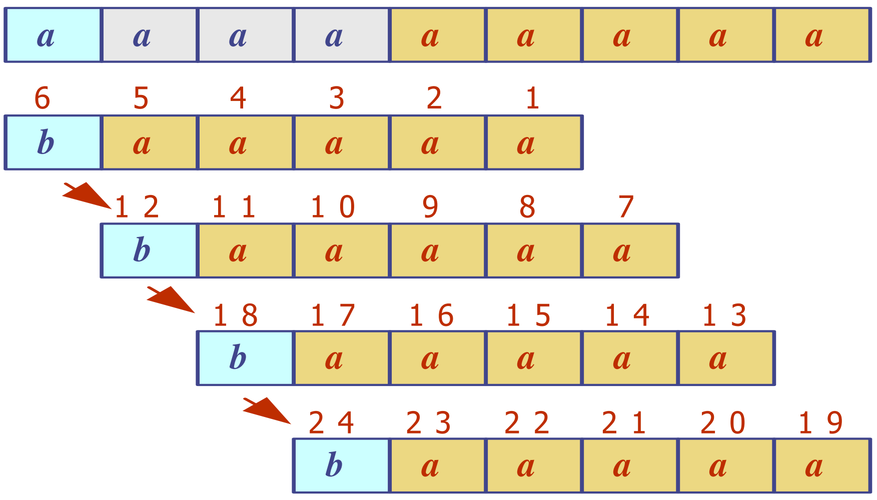


Figure (1-3): worst case scenario of BM

From the previous example we observed match all characters except the first character in the pattern, this led us to think about an efficient algorithm in this case, in this research and after analysis algorithms of string matching, we have to make some attempts to overcome the failed comparisons in the text to get a fewer comparisons also have improvement on the search process i.e. We are searching at both sides of the text at the same time by adopting the concept of Bidirectional search algorithm.

1.7 Motivation

Since the early times of computing , the researchers and developers have been giving attention to string matching algorithms and trying to find a new approach to get an efficient algorithm while preserving less complexity because the string matching algorithms which have become important in several applications. The amount of information available on the internet has grown enormously, thus making the extraction and location of information more laborious. Compared to the images and videos, searching for textual information is relatively painless. However, search for occurrences of given string is most common task, and therefore the solutions should be efficient. Good solutions were already devised several decades ago. On the other hand, development of hardware offers opportunities and causes needs for better solutions. Therefore new methods for string matching are still being developed. String matching is a very important subject in the domain of text processing and it has been one of the most extensive problems in computer technologies during past two decades.

This has motivated us to start thinking to find a new method for searching in texts to get better results

1.8 Objectives of the Thesis

The main objectives of this research are to enhance the string matching technique, by enhancing the searching algorithm. This study aims to improve the search of algorithms in the text, because this is important topic and increases with the passage of time due to the increase in the quantity and quality of information, especially through the Internet.

This study aims to reduce the time of searching within the text by using the idea of searching from both sides based on combining between BM and QS algorithms; we will focus on the number of comparisons and number of shifts in our research. In this research, we will make some changes to the preprocessing phase to get better results in the search process, and will be based on reducing the time; we need to find any pattern in the text.

In this research will be study of the algorithms and the proposed algorithm previously to solve this problem in this area, and comparing them through the programming of all the algorithms and their implementation in the same environment and under the same conditions and to study the best and worst cases, and also studied in the normal position when the text is real , and when shapes are variable-length and number , taking into account both the consumer time and memory space reserved for all algorithms.

1.9 Questions:

This research will try to answer the following questions:

1. How the worst-case scenario affects in the string matching algorithms performance?
2. How the proposed search algorithm can improve the searching process by reducing the number of comparisons?
3. How the proposed search algorithm can improve the searching process by reducing the number of attempt to find all occurrences of the pattern in the text?
4. How the proposed search algorithm can improve the searching process by increasing of the number of displacement?
5. How can we combine the bidirectional algorithm with the proposed algorithm in order to reduce the time in the string matching process?
6. What are the main challenges in enhancing the string matching process?
7. How can we compare the enhanced proposed string matching process with the original one and what is the appropriate way of validation?

1.10 Organization of the Thesis

In the first chapter defined the search problem in written texts also clarifies some of the important definitions in this study. We also identify and describe the search problem and identify the questions and hypotheses in the search and we are describing set of goals that we want to achieve from this study.

Chapter 2 contains an explanation of the degree of complexity of the algorithm also contains a description of some of the string matching algorithms Brute Force algorithm (BF) , Boyer More algorithm BM and Quick Search algorithm QS through mention the most important characteristics and implement the algorithm using C++ program .

Chapter 3 presents the related work, and indicates the most important improvements that received the best algorithms and discuss the results of this research.

Chapter 4 presents the proposed model and the process of preprocessing and searching phase. It also describes the details of our approach .This chapter talking about the mechanics of the comparison in the proposed algorithm and compares it with the previous algorithms.

Chapter 5 discusses the results and presents the experimental results. Finally; Chapter 6 presents a discussion of the thesis, conclusion and future work.

Chapter Two

Chapter Two
String searching algorithms

2.1. Background

The string matching algorithms include Brute Force algorithm (BF), Knuth Morris Pratt algorithm (KMP), and Boyer Moore algorithm (BM) since we'll talk briefly about the degree of the complexity theory algorithms and display some of the most famous search algorithms (charras and lecroq, 2004) all of these algorithms work as the follow:

1- The algorithms scan in the text by the window; each window is equal length of pattern to be searched and it is equal m.

2- Start of the search by compared to characters window with characters of pattern this process is called an attempt.

3- These algorithms are shifting the window to the right to search for matching may be found elsewhere on the length of the text.

4- Repeated all these steps until you reach the end of the text, because the goal of these algorithms is to find all the matches in the text, if it's found, or declaration about not having pattern in the text.

In case of comparing the pattern with the window to make sure the match or not (during the search process) , the algorithm must be built on the basis of quick rejection and return the match does not exist as soon as possible , without to compare all the characters (baker, 1991). in other words, when comparing the characters of pattern with the characters of window during the search process must be carried out this process quickly, that means we should be look for the characters that are not similar, because one character is not similar means no match, since we are interested in full match in this study exact matching not approximate matching.

2.2. The complexity theory of the algorithm.

commonly used function called O( ) to express complexity of the algorithm during execution , also called code landau (Landau's Symbol) and this code is used in (complexity theory) , computer science and mathematics to describe the behavior of functions an approximation , and mainly shows the rapid growth or decline functions . The name landau came from the name of the German Scientist -Edmund Landau- who created this symbols, uses the character (O) of the expression of growth functions because the rate of growth of the association called arranged functions (order), this growth or regression is used to express the sources that used by functions such as time or reserved space from computer memory (naps and pothering, 1992). For example, when analyzing a specific algorithm can be a time (or the number of steps) necessary to resolve the problem of size **n** gives the form:

$$T(n) = 4n^2 - 2n + 2.$$

If ignored constants (and this makes sense because it depends on the computer that is being used to implement the vary program from one computer to another), then we can say that: T (n) grows n^2 and writes:

$$T(n) = o(n)^2$$

2.3. Analysis of algorithms (asymptotic notations)

We have discussed Asymptotic Analysis, and Worst, Average and Best Cases of Algorithms (Boissonnat and Mariette.1998). The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. The following asymptotic notations are mostly used to represent time complexity of algorithms:

Notation	Used for
N	the length of the text
M	the length of the pattern (string)
C	the size of the alphabet *

Table (2-1): notations used for algorithms

1. Omega Ω Notation:-

The big O notation gives only upper bound on the function. If we are interested in lower bound values of the function then we have Ω notation. Let f(n) and g(n) be function whose domain as a subset of the positive integer, if there exists a positive constant c for all $n>n_0$ where n_0 is threshold integer such that, $f(n)>=C^* g(n)$, Then we write it as $f(n) = \Omega(g(n))$, Read as “f of n equals omega of g of n”

2. Big-O Notation:-

Let f(n) and g(n) be functions whose domain is subset of the positive integers, if there is exists a positive constant c for all $n>=n_0$,Where n_0 is threshold integer such that,

$f(n) <= C^*g(n)$, then we write it as

$f(n) <= O(g(n))$, and read as “f of n equals big O of g of n”

3. Theta (ϴ) Notation:-

In some cases the time for an algorithm f(n) will be as, f (n)=Ω(g (n)) and f(n)=O(g(n)) i.e. f and g have same order of magnitude and it is expressed

f (n) = (ϴ) (g (n))

f (n)= (ϴ) (g (n)) if and only if there exits positive constant C1,C2 and n0 such that for all n >= n0, C1*g (n)<=f (n) <= C2*g (n)

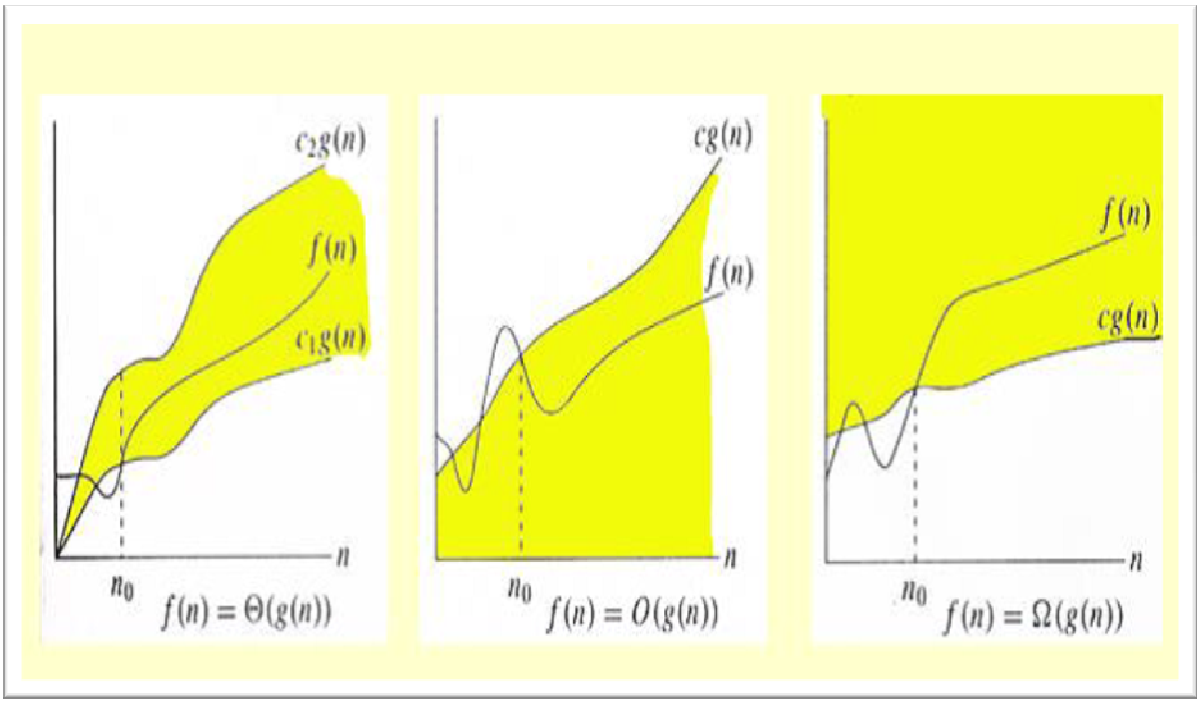


Figure (2-1): Relations of Big O, Omega Ω and Theta ϴ

The following table represents the most important associations that we face during the analysis of the work of algorithms (naps and pothering, 1992).

Function	**Name**
O(1)	**(constant)**
O(log(n))	**(logarithmic)**
O(n log(n))	**(linearithmic)**
$O((\log(n))^c)$	**(polylogarithmic)**
O(n)	**(linear)**
$O(n)^2$	**(quadratic)**
$O(n)^c$	**(geometric)**
$O(c)^n$	**(exponential)**
O (n!)	**(factorial)**

Table (2-2): the most important of functions that could accommodate most of the algorithms

3.1. Brute Force Algorithm (Bf)

The BF approach is easy to understand and implement but it can be too slow in some cases. If the length of the text is n and the length of the pattern m, in the worst case it may take as much as (n * m) iterations to complete the task.

3.1.1. The main features.

This algorithm is characterized by the following characteristics:

(charras and lecroq, 2004)

- There is no preprocessing phase.

- Need additional fixed space.
- The pattern moved to the right only by one.
- The process of comparison make in any order
- The time of complexity is o (mn).
- The numbers of comparisons are expected to average in the (2n) characters.

3.1.2. Description

In BF no pre-processing phase is performed. It compares the pattern with the text from left to right. After each attempt, it shifts the pattern by exactly one position to the right. And again the algorithm re- comparisons all until you reach the end of the text (charras and lecroq, 2004).

The displacement is always constant and equal to one, because this algorithm does not have the mechanisms to assist them in determining the length of the jump (shift), and make one shift when matching to make sure that non-overlapping pattern.

3.1.3. Pseudo code of the BF algorithm

```
Void BF (char *x, int m, char *y, int n) {

/* x: characters of pattern
Y : characters of text
M: length of pattern
N: length of text */

Int i , j;

/* Searching */

For (j=0;j<=n-m; ++j) {
  For (i=0; i<m && x[i] == y[i+j]; ++i);
  If  (i >m)
OUTPUT (j);
```

Figure (2-2): algorithm of BF (Boyer and J. Strother 1977)

3.1.4. An Example of (BF) Algorithm

First attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
	1	2	3	4																				
Pat	G	S	A	G	A	G	A	G																

Second attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
		1																						
Pat		G	S	A	G	A	G	A	G															

Third attempt

Text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
			1																					
pat			G	S	A	G	A	G	A	G														

Fourth attempt:

Text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
				1																				
Pat				G	S	A	G	A	G	A	G													

Fifth attempt:

Text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
					1																			
pat					G	S	A	G	A	G	A	G												

Sixth attempt

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
						1	2	3	4	5	6	7	8											
pat						G	S	A	G	A	G	A	G											

Seventh attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
							1																	
pat							G	S	A	G	A	G	A	G										

Eighth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
								1																
pat								G	S	A	G	A	G	A	G									

Ninth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
									1	2														
pat									G	S	A	G	A	G	A	G								

Tenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
										1														
pat										G	S	A	G	A	G	A	G							

Eleventh attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
											1	2												
pat											G	S	A	G	A	G	A	G						

Twelfth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
												1												
pat												G	S	A	G	A	G	A	G					

Thirteenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
													1	2										
pat													G	S	A	G	A	G	A	G				

Fourteenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
														1										
pat														G	S	A	G	A	G	A	G			

Fifteenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
															1									
pat															G	S	A	G	A	G	A	G		

Sixteenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
																1								
pat																G	S	A	G	A	G	A	G	

Seventeenth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
																	1							
pat																	G	S	A	G	A	G	A	G

Figure (2-3): example of (BF) algorithm

We note from the previous example, the algorithm executed by (30) comparisons for the characters and the amount of displacement in the each stage are (1)

3.2. Boyer Moore algorithm.

This algorithm - and until the date writing of this study – more effectiveness and speed in normal applications in this area, this algorithm is used as a reference for comparison for most researchers in this area (Hume and Sunday, 1991; Lovis, 2000); this algorithm for finding substrings, BM algorithm a widely known search algorithm for a single pattern and is easy to understand. It was developed by Bob Boyer and J Strother Moore in 1977.

3.2.1. Main features.

This algorithm is characterized by the following characteristics:

(Charras and lecroq, 2004)

1- Implement compared to characters from right to left.

2- The complexity for the time and the reserved space O (m +∑).

3- The complexity of the search time O (mn).

4- The best performance of this algorithm is O (n / m).

3.2.2. Description.

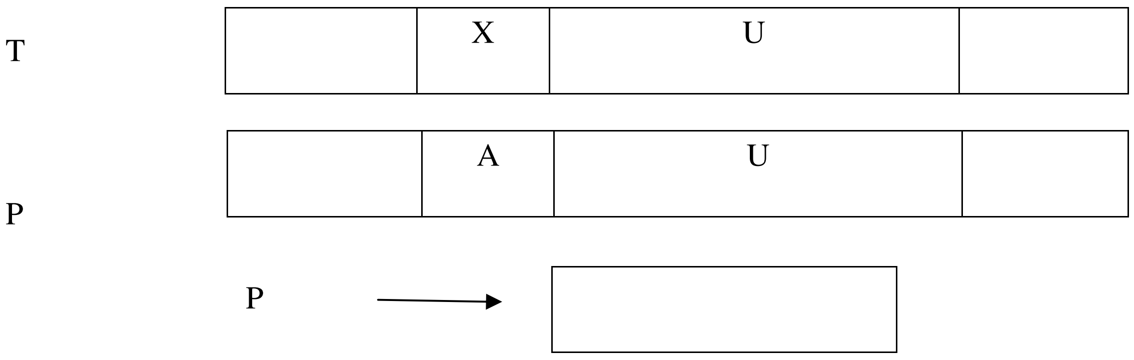
The BM algorithm scans the characters of the pattern from right to left beginning with the rightmost one and performs the comparisons from right to left. In case of a mismatch (or a complete match of the whole pattern) it uses two pre-computed functions to shift the window to the right. BM algorithm uses good-suffix function and bad-char function to calculate the new comparing position, shifting rightward *P* by taking maximum of these two values. BM Algorithm is fast in the case of larger alphabet. It uses the information gained from that attempt to rule out as many positions of the text as possible where the string could not match. Its basic idea is: First P and T align left in an attempt window. From right to left characters in P compare with corresponding characters in T. If all characters are matched, then the algorithm successfully exits. If mismatching, then the algorithm calculates the distance of P to right shift. Pattern string P right shifts and starts a new round of match attempt, at the time of pattern matching.

1) Bad character rule

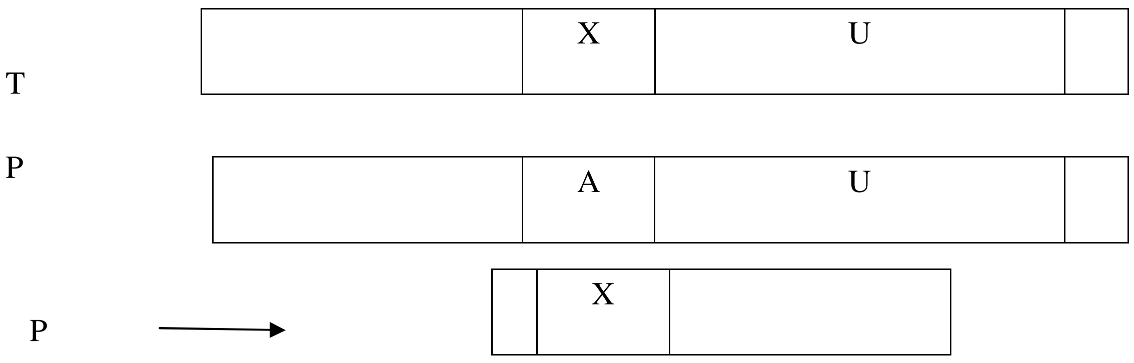
Let function Skip (x) is the distance of P right shift, Skip (x) is defined as follows:

$$\text{Shift } (x) = \left\{ \begin{array}{ll} M & \text{if character x does not appearing in the p} \\ M - Max(x) & \text{if character x does appearing in the p} \end{array} \right\}$$

Here m is the length of pattern string P, and max(x) is the nearest right location character x appears in P. In the process of scanning from right to left, if finding a mismatching character x, then shifts P according to the following two situations:

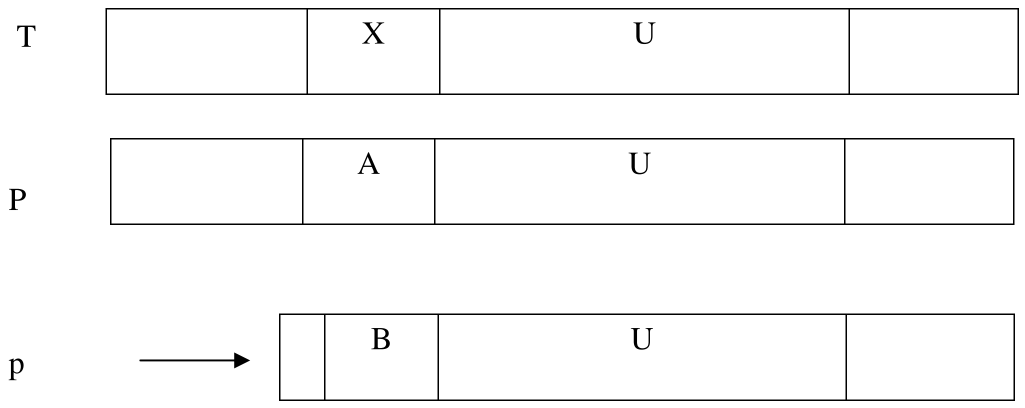


(a): character x does not occurs in p in an attempt

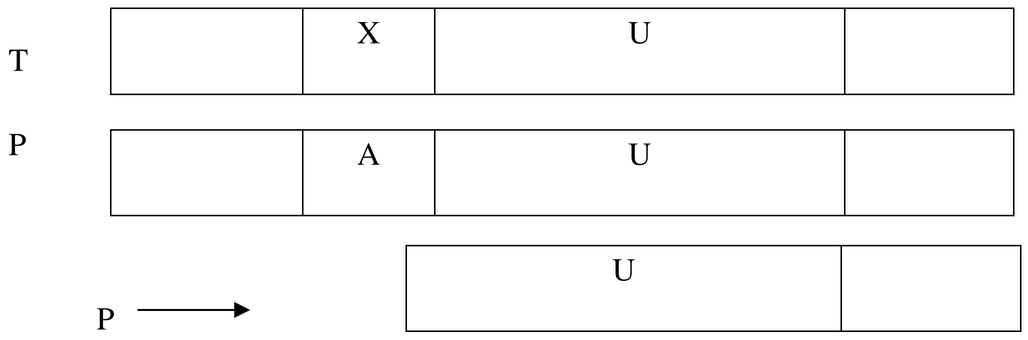


(b): character x occurs in p in an attempt window

Figure (2-4): Bad Character Rule



(a): suffix u repeated occurrence in p



(b): suffix u only a occurrence in p

Figure (2-5): Good Suffix Rule

Pseudo code of Boyer-Moore algorithm (x, m, y, n)

```
1-  $j \leftarrow 0$
2-  while $j \leq n - m$
3-  do $i \leftarrow m - 1$
4-  while $i \geq 0$ and $x[i] = y[i + j]$
5-  do $i \leftarrow i - 1$
6-  if $i < 0$
7-  then Report(j)
8-  $j \leftarrow j$ +Match(0, $y[i + j]$)
9-  else $j \leftarrow j + \max(\text{Match}(i, y[i + j]), occ[y[i + j]] - m + i + 1)$
```

Figure (2-6): The Boyer–Moore string matching algorithm(Boyer and J. Strother 1977).

Suffixes(x,m)

```
1-  suf $[m - 1] \leftarrow m$
2-  $g \leftarrow m - 1$
3-  for $i \leftarrow m - 2$ downto 0
4-  do if $i > g$ and suf $[i + m - 1 - f]$ 6= $i - g$
5-  then suf $[i] \leftarrow \min\{$suf $[i + m - 1 - f], i - g\}$
6-  else $g \leftarrow \min\{g, i\}$
7-  $f \leftarrow i$
8-  while $g \geq 0$ and $x[g] = x[g + m - 1 - f]$
9-  do $g \leftarrow g - 1$
10- suf $[i] \leftarrow f - g$
11- return suf
```

Figure (2-7): Algorithm Suffixes(Boyer and J. Strother 1977).

(1) good-suffix function

The algorithm looks up string *u* leader character is not *b* in *P* from right to left. If there exist such segment, shift right *P* to get a new attempt window. If there exists no such segment, the shift consists in aligning the longest suffix *v* of *T[i+j+1 .. j+m- 1]* with a matching prefix of *P*.

(2) bad-char function

The bad-character shift consists in aligning the text character *T [i+j]* with its rightmost occurrence in *P [0 ... m- 2*]. If *T[i+j]* does not occur in the pattern *P*, no occurrence of *P* in *T* can include *T[i+j]*, and the left end of the window is aligned with the character immediately after *T[i+j]*, namely *T[i+j+1]*

- ***Advantages***

- The both good-suffix and bad-char combined provides a good shift value as maximum of two is taken as shift value.

- ***Disadvantages***

- The preprocessing of good-suffix is complex to implement and understand.
- Bad-char of mismatch character may give small shift, if mismatch after many matches.

3.2.3. An example of BM algorithm

In the beginning the algorithm is preprocessing two table of the pattern as showing the following table:

C	**A**	**S**	**G**	**T**
BMbc[C]	**1**	**6**	**2**	**8**

Bad character table

I	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
X[i]	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**
Suff[i]	**1**	**0**	**0**	**2**	**0**	**4**	**0**	**8**
bmGs[i]	**7**	**7**	**7**	**2**	**7**	**4**	**7**	**1**

Goode suffix table

Table (2-3): bad character and good suffix

The first attempt:

text	**G**	**S**	**A**	**T**	**S**	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**	**T**	**A**	**T**	**A**	**S**	**A**	**G**	**T**	**A**	**S**	**G**
								1																
pat	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**																

Shift by (1)

Second attempt:

text	**G**	**S**	**A**	**T**	**S**	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**	**T**	**A**	**T**	**A**	**S**	**A**	**G**	**T**	**A**	**S**	**G**
							3	**2**	**1**															
pat		**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**															

Shift by (4)

Third attempt:

text	**G**	**S**	**A**	**T**	**S**	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**	**T**	**A**	**T**	**A**	**S**	**A**	**G**	**T**	**A**	**S**	**G**
						8	**7**	**6**	**5**	**4**	**3**	**2**	**1**											
pat						**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**											

Shift by (7)

Fourth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
																		3	2	1				
pat													G	S	A	G	A	G	A	G				

Shift by (4)

Fifth attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
																							2	1
Pat																	G	S	A	G	A	G	A	G

Shift by (7)

Figure (2-8): example of BM algorithm

We note from the previous example, the algorithm executed by (17) comparisons for the characters.

3.3. Quick Search algorithm QS.

It is a simplification of the BM algorithm and it is considered the easiest and it is very fast for short pattern and large alphabet(Sunday, 1990).

3.3.1. Main features.

This algorithm is characterized by the following characteristics (Sunday, 1990):

- Simplification of the [BM] algorithm.
- Uses only the bad-character shift.
- Easy to implement and programming.
- Preprocessing phase in $\boldsymbol{O}(m+\sigma)$ time and $\boldsymbol{O}(\sigma)$ space complexity;
- Searching phase in $\boldsymbol{O}(mn)$ time complexity;
- Very fast in practice for short patterns and large alphabets.

3.3.2. **Description**

Daniel Sunday has proposed an algorithm related to the Boyer–Moore –Horspool algorithm, he called it Quick Search algorithm QS. The Quick Search algorithm uses only the bad-character shift table, after an attempt where the window is positioned on the text factor $y[j\,..\,j+m-1]$, the length of the shift is at least equal to one. So the character $y\ [j+m]$ is necessarily involved in the next attempt. In QS, the shift is based on the text character immediately following the current alignment instead the last text character of the alignment.

Because the pattern is moved in any case at least one position, thus the shift for Sunday's Quick Search algorithm is one position longer than for the Boyer – Moore –Horspool algorithm on other characters except possibly for the last character of the pattern:- if the last two characters of the pattern are the same, then the shift for that character is the same equal (1). Otherwise, the shift for the last character of the pattern in Sunday's Quick Search algorithm is 1, and thus shorter obviously.

3.3.3. An Example on QS Algorithm

In the beginning the algorithm is preprocessing the bad character table of the pattern as shown in the following table:

A	A	S	G	T
qsBc[a]	2	7	1	9

Table (2-4): bad character of quick search algorithm

The first attempt

Shift by (1)

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
	1	2	3	4																				
pat	G	S	A	G	A	G	A	G																

Second attempt:

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
		1																						
pat		G	S	A	G	A	G	A	G															

Shift by (2)

Third attempt

text	G	S	A	T	S	G	S	A	G	A	G	A	G	T	A	T	A	S	A	G	T	A	S	G
				1																				
pat				G	S	A	G	A	G	A	G													

Shift by (2)

Fourth attempt

text	**G**	**S**	**A**	**T**	**S**	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**	**T**	**A**	**T**	**A**	**S**	**A**	**G**	**T**	**A**	**S**	**G**
						1	**2**	**3**	**4**	**5**	**6**	**7**	**8**											
pat						**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**											

Shift by (9)

Fifth attempt

text	**G**	**S**	**A**	**T**	**S**	**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**	**T**	**A**	**T**	**A**	**S**	**A**	**G**	**T**	**A**	**S**	**G**
															1									
pat															**G**	**S**	**A**	**G**	**A**	**G**	**A**	**G**		

Shift by (7)

Figure (2-9): example of QS algorithm

The number of shift are reached to the maximum (n – m), thus the algorithm stopped. We note from the previous example, the algorithm executed by (15) compared for the characters.

3.4. Rabin Karp String Search Algorithm

It is a string searching algorithm that uses hashing to find any one of a set of pattern strings in a text. For text of length *n* and *p* patterns of combined length *m*, its average and best case running time is O (*n*+*m*) in space O (*p*), but its worst-case time is O (*nm*)(Karp, R. M., & Rabin, M. O. 1987).

3.5. Knuth–Morris–Pratt algorithm

KMP string searching algorithm searches for occurrences of a "word" W within a main "text string" T by employing the observation that when a mismatch occurs, the word itself contains sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters (Xian-feng, Yu-bao, 2010).

3.6. Bidirectional Exact of Pattern Matching

The bidirectional algorithm compares a given pattern with any input text from both sides simultaneously on another word, this algorithm using two pointer (left and right pointer) for compare each character of the pattern with corresponding character in the text one character at time with text (Hussain, I., Kazmi, S. Z. H., Khan, I. A., & Mehmood, R. (2013) , a complete match will be found when the both left and right pointers cross each other at the middle of the pattern , the comparison order of pattern's characters with selected text window is shown in fig(2-10)

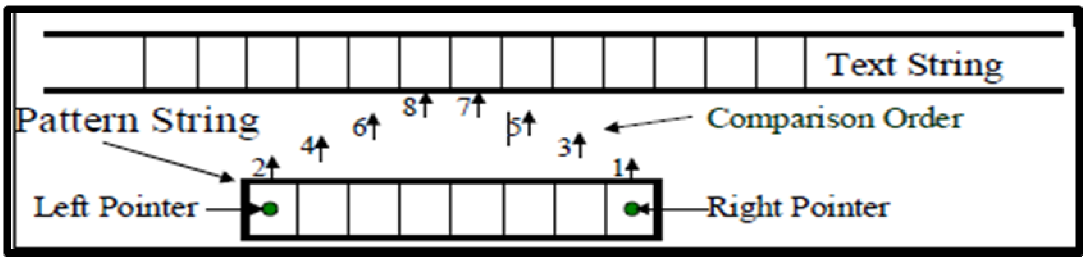


Figure (2-10): comparison order of pattern's character with text

Chapter Three

Chapter Three
Literature Survey

This chapter presents theoretical background about the string matching algorithm. There are two main processes that are directly related to our work; the first is to build the initial preprocessing phase and searching phase.

3.1. Theoretical Background

In literature, many exact string searching and pattern matching algorithms were introduced and their performance were investigate against classical exact string searching algorithm such as BF algorithm and Boyer-Moore-Horsepool (BMH) gorithm, some of these algorithms preprocess both the text and the pattern while others need only to preprocess the pattern. (Moh'dMhashi, M & Alwakeel, M 2010).

3.2. Stages of development the string matching algorithm

In computer science, the BM algorithm is a particularly efficient string searching algorithm. And it has been the standard benchmark for the practical string search literature. It was developed by Bob Boyer and J Strother Moore in 1977. The algorithm preprocesses the pattern string that is being searched in the text string.

After BM algorithm was proposed there were some algorithms are proposed to improve it. In 1980, Horspool simplified BM algorithm and proposed BMH algorithm although it only used the information of the table Right, BMH algorithm acquired no bad efficiency. In 1990 Sunday proposed BMHS algorithm that improved the BMH algorithm. In 2010, Lin quanXie, Xiao mingliu proposed BMHS2, which is strictly based on the analysis of BMHS algorithm to improve is in the match fails, the text string matches last bit characters to participate in the next match, a character string in the case appear to increase the last bit character and appear in the character string matching the first characters of a position if there is consideration.

In 2010 BMI algorithm is proposed by Jingbo Yuan, JisenZheng, Shunli Ding which is improvement of BM algorithm. The BMI algorithm combines with the good suffix function and the advantages of BMH and BMHS at the same time the BMI algorithm also takes into account the singleness and combination features of the Next-Character and the Last- Character. There are two important factors which influence the efficiency and speed of pattern matching and they are the cost to find the mismatching character in the text string and the shift distance to right, on basis of the two factors, an improved algorithm called Improved BMHS algorithm which is given by Yuting Han, GuoaiXu in 2010. Another improved algorithm called composite Boyer- Moore was proposed in 2010 by ZhengdaXiong. The key issue of the composite Boyer-Moore algorithm is how to utilize the history comparison information achieved at previous iteration. So a new concept of two-dimensional table Jump[m][m] is introduced.

There are many other researchers are worked to enhance the string matching algorithms to be fully adopted to work with many researchers have introduced various algorithms to find the exact pattern matching by making use of windowing technique whose length is equal to the pattern length. Al-Emery and Japer, proposed an algorithm to improve the search process (El emery and Jaber, 2008). In the preprocessing phase, they split the unchangeable text into n equal parts depending on the length of the text and then construct n tables. Each table consists of two columns for each part of the text, the first one is the words' length and the second one is the start position of each word in the text classified by the same length. The algorithm searches for the words that consist of the same length in each table.

Devaki-Paul algorithm (DP), results in better performance and efficiency. Before starting the search, the algorithm requires a preprocessing of the pattern which prepares a table of occurrences of the first and the last characters of the pattern in the given input text. The search phase uses the table to find the probability of having an occurrence of a pattern in the given input text and find if the probability will lead to successful or unsuccessful search. The time complexity of the preprocessing phase of the DP algorithm is O (m) while the time complexity of the search phase is directly proportional to the total number of occurrences of the first and the last characters of the pattern in the given input text (Pendlimarri, D., Petlu, P., & Satrasala, R. 2011).

3.3. Related works

Several pattern matching algorithms have been developed with a view to enhance the Searching processes by minimizing the number of comparisons performed, to reduce the number of comparisons. The matching process is usually divided into two phases. The pre-processing phase and the searching phase. The pre-processing phase determines the distance (shift value) that the pattern window will move. The searching phase uses this shift value while searching for the pattern in the text with as minimum character comparisons as possible.

This section presents some of studies that have contributed to improvement of the search algorithms and Classified as follows:

3.4. Exact pattern matching algorithms.

A)-Searching from one side

- (Hasan & Rashid, 2012) The Internet users are wildly spread besides; the intrusion activities are also increased, the IDS depends on string matching algorithms to perform its detection, this paper introduced hash function HBMH to enhance the BMH algorithm whereas; BMH spends a long time in character comparison in each attempt. Therefore, it considers faster than the original BMH algorithm, the hash function is really reduced the number of character comparison thus, the HBMH is faster in its performance and can be very useful in network security applications such IDS.

- The comparison of BM and its relative algorithm is performed on the basis two factors; one is number of comparison performed and second is search time. BM algorithm is standard benchmark of string matching algorithm so this paper explain the BM algorithm and then explain its improvement as BMH (Boyer-Moore-Horspool), BMHS (Boyer-Moore-Horspool-Sundays), BMHS2 (Boyer-Moore-Horspool-Sundays 2), improved BMHS(improved Boyer- Moore-Horspool-Sundays) ,BMI (Boyer-Moore improvement) and CBM (composite Boyer-Moore), also analyze and compare them using an example and find which one is better in which conditions The performance of algorithm depends on two factors, first on Input, number of inputs and type of inputs, Second is Methodology of algorithm, so there may be possible that some variation in performance occur as input changes.(Choudhary, R et al , 2012).

- Rasool and Khare introduced an enhanced algorithms based on BMH and BMHS algorithms this work has developed improved algorithms EBMH (Enhanced BMH) and EBMHS (Enhanced BMHS). This algorithm uses the newly introduced PDJ (possible double jump) and MValue (Match Value) concepts. While searching these concepts helps to provide longer jump of characters. The algorithm EBMH emphasize on BMH algorithm with the inclusion of PDJ and MValue and the algorithm EBMHS emphasize on BMHS algorithm with the inclusion of PDJ and MValue. Through these algorithm the number of comparison of characters between text and pattern are reduced to a significant amount. In this paper PDJ, MValue, EBMH and EBMHS are described and analyzed. Experimental results show that in the algorithms searching time is reduced as compared to BM, BMH, and BMHS. The algorithms are analyzed on the basis of time requirement in best, worst and average case (Rasool & Khare, 2013).

- (Mhashi & Alwakeel 2010),In this paper, a new Enhanced Checking and Skipping Algorithm (ECSA) is introduced. The new algorithm enhance the classical string searching algorithms by converting the character-comparison into character-access, by using the condition type character-access rather than the number-comparison, and by starting the comparison at the latest mismatch in the previous checking, which in turn increases the probability of finding the mismatch faster if there is any. The results of the experiment show that the performance of the enhanced algorithm is outperform the performance of the introduced algorithms. On the other hand, in the skipping phase ECSA focuses on increasing the shift distance. The search clock time criteria were used in an experiment to compare the performance of ECSA against Naïve, and BMH.

- (Senapati, KAdhikary, D & Sahoo, G. 2012) proposed an algorithm, for finding motif in DNA sequence. This algorithm is based on preprocessing of the pattern string(motif) by considering four consecutive nucleotides of the DNA that immediately follow the aligned pattern window in an event of mismatch between pattern(motif) and DNA sequence. Theoretically, found the proposed algorithms work efficiently for motif identification in the same way, the proposed algorithm is implemented method of searching in the searching phase of Berry-Ravindran algorithm to get the Improved Berry- Ravindran algorithm. This paper based on preprocessing of the pattern string by considering four consecutive characters of the text. The concept of searching from both sides makes the algorithm efficient when a mismatch present at the end of the pattern with that of align text window. This work proves that the proposed algorithms faster than other compared algorithms.

- (Rawan A. Abdeen ,2011) proposed an improvement of the brute force searching algorithm. The algorithm is named Start to- End Algorithm. This paper has proposed a string searching algorithm without the need to preprocess neither the pattern nor the text. The improvement that this algorithm has offered over the brute-force algorithm is that it does not allow performing character by character matching between the segment taken from the text and the pattern only after it checks that the first and last characters in the pattern match the first and last characters in the segment taken from the text.

- Based on text segmentation this paper proposed a fast pattern matching algorithm based on text segmentation by slicing the text in to segments each equal to size of the pattern. The idea is to perform preprocessing of both text and pattern strings before beginning to search for the pattern in the text so as to achieve substantial speed up in the search process. The experimental results show that the proposed algorithm is superior to other algorithms even when the pattern is in the end of the text. The algorithm uses the idea of preprocessing both the text and pattern strings as against to other existing algorithms which either pre-process text or pattern or does no preprocessing such as BF algorithm (Radha , Sravya & Anji , 2011).

- This paper presented a new method to improve the (avg-case) performance of the BM , The basic idea is to utilize two character for a recomputed table instead of one character as in the original BM whenever a mismatch occurs, they can the slide of pattern to the right a longer distance than in the original version .In their work the solution preserved all good properties of the original BM (Feng, Z. R., & Takaoka, T. 1988) .

B) - Searching from both sides (bidirectional)

- This research presents a new idea to compare pattern with selected text window from both sides of the pattern simultaneously by using right and left pointers, Bidirectional(Exact Pattern Matching) EPM algorithm is basically based on the bad character rule of Boyer-Moore algorithm where only one character is used to identify the shifts. Bidirectional EPM algorithm has number of cases to shift the pattern maximum to right of text window. Suppose T(1…n) is the text string and P(1…m) is the pattern and we compare P(1…m) with T(i…i+m-1) from both sides of the pattern, one character at a time, start from right side of the pattern , the time complexity of preprocessing phase of BD exact pattern matching is O (m) and searching phase takes O (mn/2) (Hussain, I. et al,2013).

- This algorithm enhances the performance of the Two Sliding Windows (TSW) algorithm. Both ETSW and TSW algorithms employs the main idea of BR by maximizing the shift value and using two sliding windows rather than using one sliding window working in parallel, to scan all text characters. In both algorithms, two arrays are used to store the calculated shift values for the two sliding windows. Each array is a one dimensional array of length (m-1). The experimental results show that the ETSW algorithm has enhanced the process of pattern matching by reducing the number of comparisons performed (Suleiman,D.et al , 2013).

- This paper proposed a new algorithm ERS-A, that made enhancements on both two sliding windows (TSW) and Fast Pattern Matching (RS-A) algorithms. In ERS-A and TSW algorithms two sliding windows are used to scan the text from the left and right simultaneously, but while TSW utilizes the idea of Berry Ravindran bad character shift function (BR), ERS-A adds an improvement by using the shift technique provided by RS-A algorithm . RS-A algorithm uses four consecutive characters in the text immediately following the pattern window, instead of using two consecutive characters as in BR. In this paper ERS-A is implemented and ERS-A improves the performance by reducing the number of comparisons and the number of attempts needed to search for a particular pattern (Suleiman,D.et al , 2013).

- This paper proposed an Improved-Bidirectional (IBD) exact pattern matching algorithm based on window sliding method of exact pattern matching which will be helpful in various needs of pattern matching and searching. The basic idea of Improved-Bidirectional EPM algorithm is scans partial text window of the text string for leftmost character of pattern and pattern for the rightmost character of the partial text window. Improved-Bidirectional EPM algorithm compares a given pattern character wise with selected text window from both sides simultaneously as Bidirectional algorithm does. The worst case time-complexity of Improved-Bidirectional EPM algorithm is O (mn/2) in searching phase and O (m) in preprocessing phase. Comparison results show that the Improved-Bidirectional EPM algorithm is quite efficient than the existing algorithms, when the pattern length is short as well as on long pattern's lengths (Hussain, I. et al, 2013).

- (Hussain, I. et al, 2013).This research, presents an improved version of Bidirectional (BD) algorithm to solve the problem of exact pattern matching. Fastest-Bidirectional (FBD) exact pattern matching algorithm introduced a new idea of scanning partial text window (PTW) as well with the pattern by taking Berry-Ravindran (BR) consecutive characters to take decision of moving pattern to the right of PTW. FBD algorithm compares the characters of pattern to selected text window from both sides simultaneously as BD. The time complexity of preprocessing phase of FBD algorithm is O (m+ |Σ|) and searching phase takes *O (mn/2).*In other hand the Searching Phase of FBD algorithm runs at most 'm/2' times so, its worst case time-complexity is O (m/2) because two pointers are used. FDB algorithm requires O (2m) extra memory space in worst case to execute in addition with the text and pattern string.

- This research presents a new idea of comparing the pattern with text string from both sides of the pattern simultaneously. After analysis the BD algorithm shows the Bidirectional algorithm is better than exiting algorithms, it takes *O(mn/2)* time in searching phase The worst case time complexity of preprocessing phase of Bidirectional algorithm is *O(m),* because only one loop is used to scan the pattern to find the characters. The total time complexity of searching phase is *O (mn/2),* because O (n) takes to shift pattern to right of the text and O (m/2) to search pattern in text string. Bidirectional algorithm requires O (m) extra memory space in worst case in addition with the text and pattern string.

C) - In exact pattern matching algorithms (approximate)

- (Wu, S., &Manber, U,(1992)) proposed a new tools , called A-GREP, for approximate pattern matching A GREP is based on a new efficient and flexible algorithm for approximate string matching A-GREP is also competitive with other tools for exact string matching; it include many options that make searching more powerful and convenient . This paper discussed errors in the text or in the query can result from misspelling or from experimental error (e.g, when the text is a DNA sequence).

D) - Indexed searching

- This paper proposed a new algorithm with a simple logic which is very easy to implement. This method can be use for pattern matching in protein sequences and also for English Text (Devi, S. N, et al. 2013). This method depends on the preprocessing phase which retrieves the index based on the frequent character of the pattern and it is suitable for unlimited size of input sequence. This approach provides good performance related to DNA and protein sequence dataset, this algorithm is decreasing the number of comparisons.

- Pendlimarri, D., Petlu, P., & Satrasala, R. (2011), this paper presented a new algorithm for string matching called, DP algorithm (Devaki – Paul algorithm). In case of unsuccessful search, the DP algorithm has zero character comparisons, irrespective of the sizes of the text and pattern, provided if either the first or the last character was not present in the given input text. Whereas, the BM and QS algorithms will do search as usual. The algorithm also doesn't require any pre-processing phase, if the search is on the same given input text and with different patterns, provided the first and the last characters are same as in the case of first pattern. The time complexity of the preprocessing phase of the DP algorithm is O (m), which is less than the BM and QS algorithms. The time complexity of the search phase of the DP algorithm is directly proportional to the total number of occurrences of the first and the last characters of the pattern in the given input text.

Chapter Four

Chapter Four
The Proposed Algorithm
Bidirectional Boyer Moore and Quick Search (BBQ)

4.1. Overview

By studying the described algorithms in the previous chapter, we have noticed that the best algorithm is making shift as much as possible in every attempt (the largest displacement of the window to the right) also whenever comparisons are decreasing, so that's leads to increase the speed of the algorithm. In order to make these jumps should be available advance information (are obtained through an initial processing) which are used to determine how much the jumps without exceeding any part of the text, the omission of any part of the text is possible, because some of the patterns in the text are overlapping with each other (overlapped).

For example, if the text "babab" and search for the pattern "bab", the number of occurrences of this pattern in the text (with consideration the overlap) is a two-time, first "ba<u>bab</u>", and the second "<u>bab</u>ab". And the advance information is taken by processing the pattern or processing of the text itself. Most of the algorithms were mentioned above, as well as most of the other algorithms that are looking for strings in the text, The main idea of the work algorithms are processing of the pattern, and the best jump can be achieved in these algorithms are (m); the length of the pattern because the pattern does not have the longest of (m) as an information to offer to the algorithm to be used in determine how much shift.

This chapter presents our approach for string matching algorithm which combined two techniques; BM and QS algorithms resulting an enhanced algorithm that we named BBQ (for Bidirectional Boyer-Moore and Quick Search) algorithm. Our approach will be affect the comparison and search processes between the text and the pattern .This chapter explains the architecture of BBQ algorithm and presents the process of building the pre-processing phase and also presents the process of searching phase technique and computing the complexity then programming by using the **C#** language. BM and QS are the two core concepts in our approach.

4.2. Description of BBQ algorithm

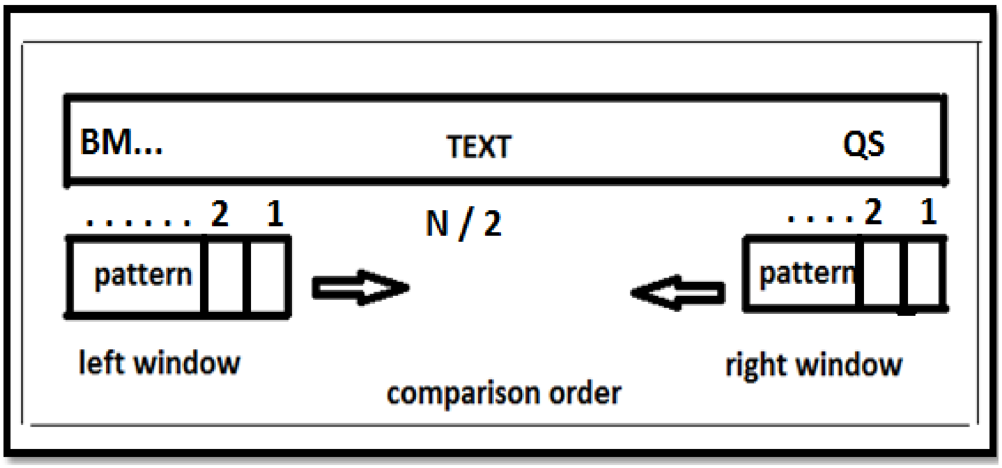
The BBQ algorithm based on scans the text from both sides simultaneously. It uses two sliding windows; the size of each window is m which is the same size as the pattern. The two windows search the text in parallel. The text is divided into two parts: the left and the right parts, each part is of size (n/2). The left part is scanned from left to right using the left window and the right part is scanned from right to left using the right window. Both windows slide in parallel which makes the algorithm suitable for parallel processors structures. The BBQ algorithm stops when one of the two sliding windows finds all occurrences of the pattern or the pattern is not found within the text string at all. Also if the pattern is exactly in the middle of the text, the BBQ algorithm can find it easily.

The BBQ algorithm utilizes the idea of BM and QS to get better shift values during the searching phase. BM algorithm provides a maximum shift value in most cases without losing any characters.

4.3. Basic Idea

The BBQ algorithm of pattern matching compares a given text character wise from both sides simultaneously as Bidirectional algorithm. The difference between work of the existing algorithms and the BBQ algorithm is that the search process will be the beginning of the text using an BM algorithm and shift pattern to the right , from the other side and at the same time be searching from the end of the text and shift pattern to the left using an QS algorithm ; The process of search will be reduce the number of attempts needed to find a pattern in the text and thus reduce the number of comparisons of characters i.e. we move the pattern over the text from both sides simultaneously . A complete match will be found when left window cross right window at middle of the text.

In our work we used two algorithms for search the pattern in the text as shown in the following figure:



Figure(4-1) : Comparison order between the text and pattern from both sides

- **The following steps details our approach implementation:**

1- We will be using the text as input

2- We will search for any pattern found or not found in the text

3- The text is divided into two parts: the left and the right parts, each part is of size (n/2) where n is length of text.

4- The BBQ algorithm combines two algorithms; every shift to right or shift to left based on pre-processing phase before the searching process.
5- The left part is scanned from left to right using the left window and the right part is scanned from right to left using the right window.
6- The left parts based on the idea of BM bad character shift function and good suffix function to get better shift values during the searching phase.
7- The right parts based on the idea of QS bad character shift function to get better shift values during the searching phase.
8- Both windows move in parallel.
9- The search process continues until the algorithm finds all the positions of pattern within the text.
10- If not found the algorithm return " the string is not found in the text".

◈ **The main differences between BBQ algorithm and BM algorithm are:**

- BBQ uses two algorithms.
- The search process in BBQ from both sides instead of one side over the text.
- The shift values are calculated only for the pattern characters; while the original BM algorithm is calculated the shift values for all the alphabets, therefore reduces the search processing time and at the same time reduces the memory requirements needed to store the shift values.

4.4. Pre-processing phase:

The pre-processing phase is used to divide the text in two parts then make pre-processing for the pattern. The values of the left window are calculated according to BM algorithm with two functions (bad character and good suffix rule) without change. The left window contains the shift values needed to search the text from the left side. To calculate the shift values, the algorithm preprocessing the pattern only.

On the other hand, the values of the right window are calculated by another algorithm to make comparisons from the other side which called QS algorithm; the right window contains the shift values needed to search the text from the right side, to calculate the shift values, the algorithm preprocessing the pattern only

The maximum shift can we show in this algorithm is (m) from right window and (m+1) from left window, where ***m*** is the length of pattern.

The only thing we want to do is to construct a formula as follows. Let(x) be a character in the alphabets. We record the position of the characters x where x is the first character previous of the window (left of window), if it exists in P; we are calculating the amount of displacement. If x does not exist in P, we record it as (m+1) as shown in the following:

$$\text{Shift [R window]} = \left\{ \begin{array}{ll} m+1 & \text{if character x doesn't occurs in pattern} \\ m-((m-1)-j) & \begin{array}{c} \text{if } p[\,j\,] = x \\ \text{Character x does occurs in pattern} \end{array} \end{array} \right\}$$

The first step is align the right ends of the pattern on the text from both sides, then compare the characters of the text aligned with the characters of the pattern this specific work is called an –attempt- and after a whole match of the pattern or after a mismatch ;shift the pattern to the right from left side and to the left from right side according of bad character table. Then repeat the same procedure again.

The pre- processing that we do is important to calculate the amount of jumps during the search process The following explains briefly the pre-processing that occur on the left side using the BM and the other side using QS.

4.4.1. The pre-processing of pattern from left window:

Here we are using The BM algorithm because it's considered as the most efficient string-matching algorithm in usual applications the algorithm scans the characters of the pattern from right to left beginning with the rightmost symbol. In case of a mismatch (or a complete match of the whole pattern) it uses two pre-computed functions to shift the pattern to the right. These two shift functions are called the bad-character shift and the good-suffix shift. Assume that a mismatch occurs between the character $x[j] = b$ of the pattern and the character $y[i+j] = a$ of the text during an attempt at position I . Then $y[I + j + 1 \ldots I + m - 1] = x[j + 1 \ldots m - 1] = u$ and $y[I + j] \neq x[j]$, The good-suffix shift consists in aligning the segment $y[I + j + 1 \ldots i + m - 1] = x[j + 1 \ldots m - 1]$ with its rightmost occurrence in x that is preceded by a character different from $x[j]$. If there exists no such segment, the shift consists in aligning the longest suffix v of $y[I + j + 1 \ldots i + m - 1]$ with a matching prefix of x as shown in the following figures:

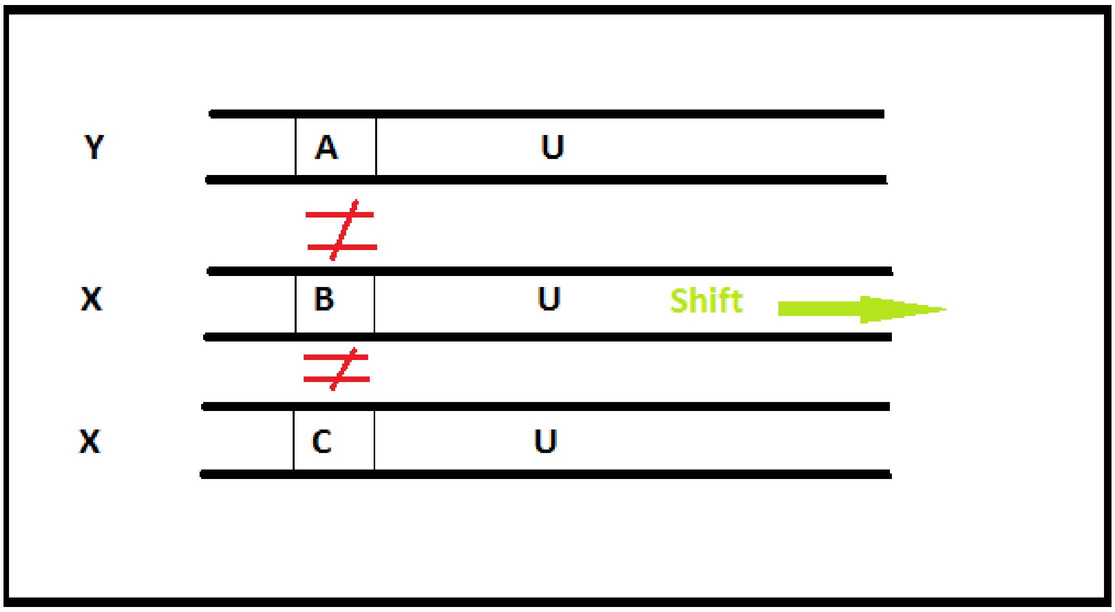


Figure (4-2): Good suffix shift u reappears preceded by a character different from b.

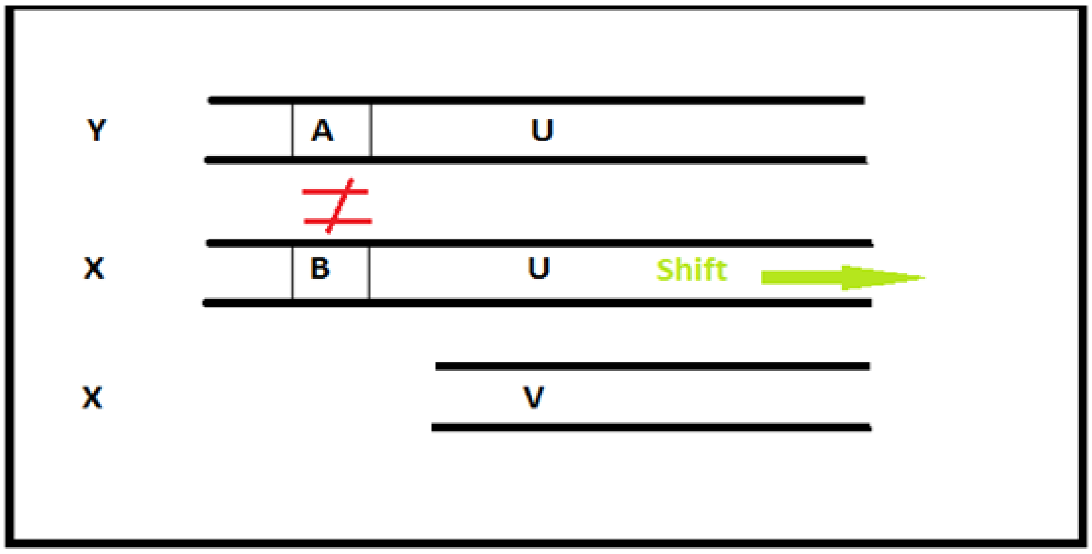


Figure (4-3): Good suffix shift only a prefix of u reappears in x

Example 1:

Y = a b b a a b b a b b a

X = a b b a a b $\underline{b}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$

X = $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$

The shift is driven by the suffix abba of (x) found in the text. After the shift, the segment abba of (y) matches a segment of (x) the same mismatch does not recur.

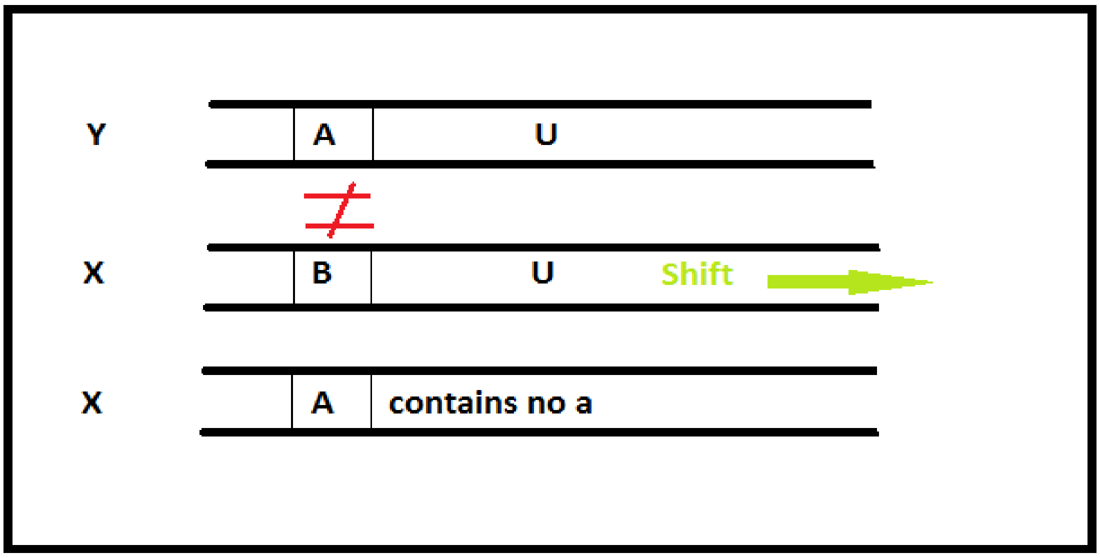


Figure (4-4) : bad – character – shift a appears in x

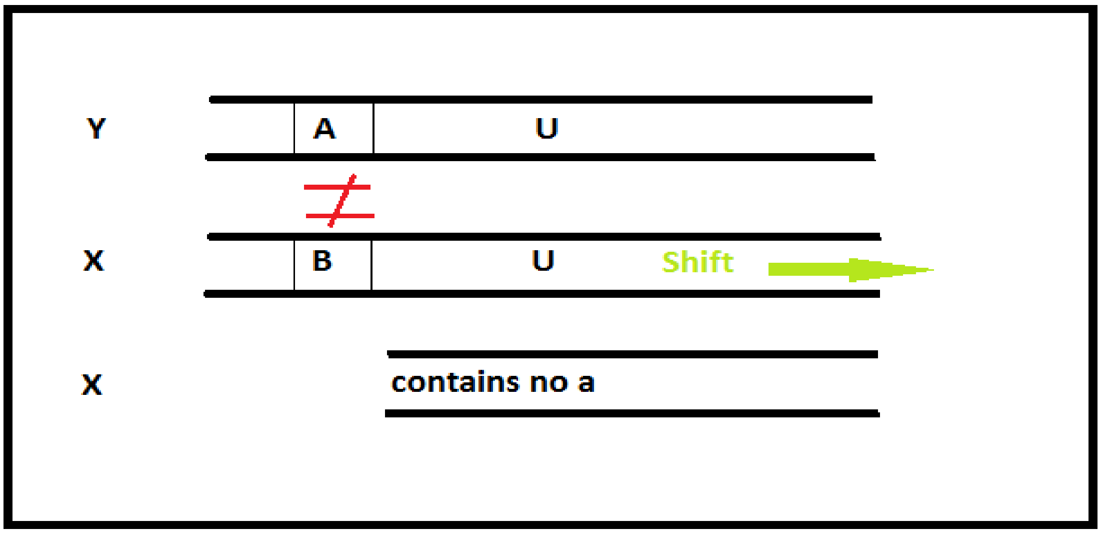


Figure (4-5) : bad – character – shift a does not appears in x

Example 2:

y = . . . a b b a a b b a b b a b b a . .

x = b b a b $\underline{b}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$

x = $\underline{b}$ $\underline{b}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$ $\underline{b}$ $\underline{b}$ $\underline{a}$

The segment (abba) found in (y) partially matches a prefix of (x) after the shift.

The bad-character shift consists in aligning the text character y [I + j] with its rightmost occurrence in x [0 …m – 2]. if y [I + j] does not appear in the pattern x , no occurrence of x in y can include y [I + j] and the left end of the pattern is aligned with the character immediately after y [I + j] , namely y [I + j + 1]

Example 3:

Y = ... a b c d . . . .

X = c d a h g f $\underline{e}$ $\underline{b}$ $\underline{c}$ $\underline{d}$

X = c d a h g f e b c $\underline{d}$

The shift aligns the symbol **(a)** in **(x)** with the mismatch symbol **(a)** in the text.

Example 4:

y = ... a b c d . . . . . .

x = c d h g f $\underline{e}$ $\underline{b}$ $\underline{c}$ $\underline{d}$

x = c d h g f e b c $\underline{d}$

The shift positions the left end of **(x)** right after the symbol **(a)** of y.

The BM algorithm is shown in (Figure 4-6). For shifting the pattern, it applies the maximum between the bad-character shift and the good-suffix shift. More formally the two shift functions are defined as follows. The bad-character shift is stored in (**bc**) of size **α** and the good-suffix shift is stored in a table (**gs**) of size (m – 1). For a € Σ:

$$Bc[a] = \begin{cases} \min\{ j / 1 \leq j < m \text{ and } x[m-1-j] = a \} & \text{if } a \text{ appears in } x \\ M & \text{other wise} \end{cases}$$

Let us define two conditions:

Cond 1(j ,s): for each k such that $j < k < m$, $s \geq k$ or $x[k-s] = x[k]$

Cond 2 (j ,s) : if $s < j$ then $x[j-s] \neq x[j]$

Then for $0 \leq I < m$

$Gs[i+1] = \min \{ \alpha > 0 / \text{cond } 1 (I , s) \text{ and cond } 2 (I , s) \text{ hold} \}$

// (Boyer, Robert S., and J. Strother Moore,1977)

Figure (4-6): shifting of the BM

```
void BM(char *y, char *x, int n, int m)
{
int i, j, gs[XSIZE], bc[ASIZE];
/* Preprocessing */
PRE_GS(x, m, gs);
PRE_BC(x, m, bc);
/* Searching */
i=0;
while (i <= n-m) {
j=m-1;
while (j >= 0 && x[j] == y[i+j]) j--;
if (j < 0) OUTPUT(i);
i+=MAX(gs[j+1], bc[y[i+j]]-m+j+1); /* shift */
}
} // ( Boyer, Robert S., and J. Strother Moore,1977)
```

Figure (4-7): The Boyer-Moore string-matching algorithm.

Tables bad-character and good suffix can be recomputed in time O (m + α) before the search phase and require an extra-space in O (m + α). The worst-case running time of the algorithm is quadratic. However, on large alphabets (relatively to the length of the pattern) the algorithm is extremely fast. Slight modifications of the strategy yield linear-time algorithms the algorithm makes only O (n / m) comparisons, which is the absolute minimum for any string-matching algorithm in the model where the pattern only is preprocessed.

```
Void PRE_BC (char *x, int m, int bc[])
{
int j;
For (j=0; j < ASIZE; j++) bc[j] =m;
For (j=0; j < m-1; j++) bc[x[j]] =m-j-1;
}// ( Boyer, Robert S., and J. Strother Moore,1977)
```

Figure (4-8): Computation of the bad-character shift.

```
void PRE_GS (char *x, int m, int gs[])
{
int i, j, p, f[XSIZE];
For (i=0; i <= m; i++) gs[i] =0;
F[m] =j=m+1;
For (i=m; i > 0; i--) {
While (j <= m && x [i-1]! = x [j-1]) {
If (! gs[j]) gs[j] =j-i;
j=f[j];
}
F [i-1] =--j;
}
p=f [0];
For (j=0; j <= m; j++) {
If (! gs[j]) gs[j] =p;
If (j == p) p=f[p];
}
}// ( Boyer, Robert S., and J. Strother Moore,1977)
```

Figure (4-9): Computation of the good-suffix shift.

4.4.2. The pre-processing of pattern from right window:

Here we are using The idea of QS algorithm because The bad-character shift used in the BM algorithm is not very efficient for small alphabets, but when the alphabet is large compared with the length of the pattern, as it is often the case with the ASCII table and ordinary searches made under a text editor, it becomes very useful. Using it alone produces a very efficient algorithm in practice.

The first attempt where a right end of pattern is aligned with right ends of the text, the length of the shift is at least equal to one. So, the character that previous of window in each attempt is necessarily involved in the next attempt, and thus can be used for the bad-character shift of the current attempt. The bad-character shift of the present algorithm is slightly modified to take into account the last symbol of x as follows:

$$\text{bc}\,[\alpha] = \left\{ \begin{array}{ll} \min\ \{\, j \,/\, 0 \leq j < m \ \text{ and } \ x\ [\, m-((m-1)-j\,] = \alpha & \text{if } \alpha \text{ appears in } x \\ m+1 & \text{otherwise} \end{array} \right\}$$

The comparisons between text and pattern characters during each attempt can be done in any order. The algorithm of Figure (4-10) performs the comparisons from right to left. The algorithm is based on idea of QS algorithm.

Example 5:

```
y = s t r i n g - m a t c h i n g
x =                         i n g
x                   i n g
x =         i n g
x =       i n g
```

Quick Search algorithm makes **8** comparisons to find the two occurrences of(ing) inside the text of length **15**.

```
void QS (char *y, char *x, int n, int m)
{
int i, j, bc[ASIZE];

/* Preprocessing */
For (j=0; j < ASIZE; j++) bc[j]=m+1;
For (j=m-1; j > 0; j--) bc[x[j]]= m-((m-1)-j);

/* Searching */

i=n-1;
while (i >= m-1) {

j=m-1;

while (j >0 && x[j] == y[i]) j - -
i--;

if (j < 0) OUTPUT(i);

            i-= bc[y[i-m]];                      /* shift */
}
}
```

Figure (4-10): Computation of the shift for right window.

4.4.3. The work of BBQ algorithm from right side is described in the following steps:

1. Make pre-processing before begin of search for the character of pattern.
2. Using one dimensional array to fill the value of character
3. Using the formula (m – ((m-1)-j) to find amount of jumps for each character in the pattern where ***m*** Where m length of the pattern and ***j*** position of character in the pattern.
4. The algorithm scans the characters of the pattern from right to left beginning with the rightmost symbol
5. Start of comparison In case of a mismatch (or a complete match of the whole pattern) it uses shift functions are called the bad-character shift as the following :

Index (j)	0	1	2	3	Another character *
Pat	F	R	F	D	
F (x)	m-((m-1)-j) 4-((4-1)-0)= 1	m-((m-1)-j) 4-((4-1)-1)= 2	m-((m-1)-j) 4-((4-1)-2)= 3	m-((m-1)-j) 4-((4-1)-3)= 4	m + 1 4+1 = 5

Since the character is repeated in the pattern always we take at least

For (j=m-1; j > 0; j--)
bc[x[j]]= m-((m-1)-j)

Table (4-1): pre-processing phase

	Shift	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Text		Q	W	S	F	R	F	D	E	T	F	R	f	d	s
Attempt 1	1											F	R	F	D
Attempt 2	5										F	R	F	D	
Attempt 3	1					F	R	F	D						
Attempt 4	5 stop				F	R	F	D							

Start of compari son from end s of pattern and text

This character detected number of jump by its value in the pre-processing of BC table

Table (4-2): comparisons of the right window

4.5. Searching phase:

In this phase, the text string is scanned from two directions, from left to right and from right to left According to the text. In mismatch cases, during the searching process from the left, the left window is shifted to the right, while during the searching process from the right; the right window is shifted to the left. Both windows are shifted until the pattern is found or the windows reach the middle of the text (n/2).

Step1: Compare the characters of the two windows with the corresponding text characters from both sides:

1- If there is a mismatch during comparison from both sides, the algorithm goes to step2.

2- Otherwise, the comparison process continues until a complete match is found.

3- The algorithm stops and displays the corresponding position of the pattern on the text string.

4- If we search for all the pattern occurrences in the text string, the algorithm continues to step2.

Step2: In this step, we use the shift values from the left and right arrays of windows depending on the characters which make mismatch with pattern after pre-processing of pattern. The corresponding windows are shifted to the correct positions based on the shift values, the left window is shifted to the right and the right window is shifted to the left. Both steps are repeated until the first occurrence of the pattern is found from either sides or until both windows are positioned beyond (n/2). If the first occurrence of the pattern exists in the middle of the text, the algorithm continues comparing pattern characters with text characters.

4.5.1. Searching from left side

The searching algorithm compares the character of the pattern from right to left with the text. After a complete match the pattern is shifted according to how much its widest border allows. After a mismatch the pattern is shifted by the maximum of the values given by the good-suffix and the bad-character heuristics.

4.5.2. Searching from right side

The searching algorithm compares the character of the pattern from right to left with text. After a mismatch the pattern is shifted to the left depending on the value of the character that precedes the current window in the bad-character shift table the length of the shift is at least equal to one.

Suppose that P_1 is aligned to T_s now, and we perform a pair-wise comparing between text T and pattern P from right to left. Assume that the first mismatch occurs when comparing character *[y]* with *[z]*.

Since($y \neq z$), we move the pattern *P* to left such that the position *j* in the left of *P* is equal *T [n-m-1]* We can shift the pattern at least (*m-((m-1)-j)*) positions to the left.

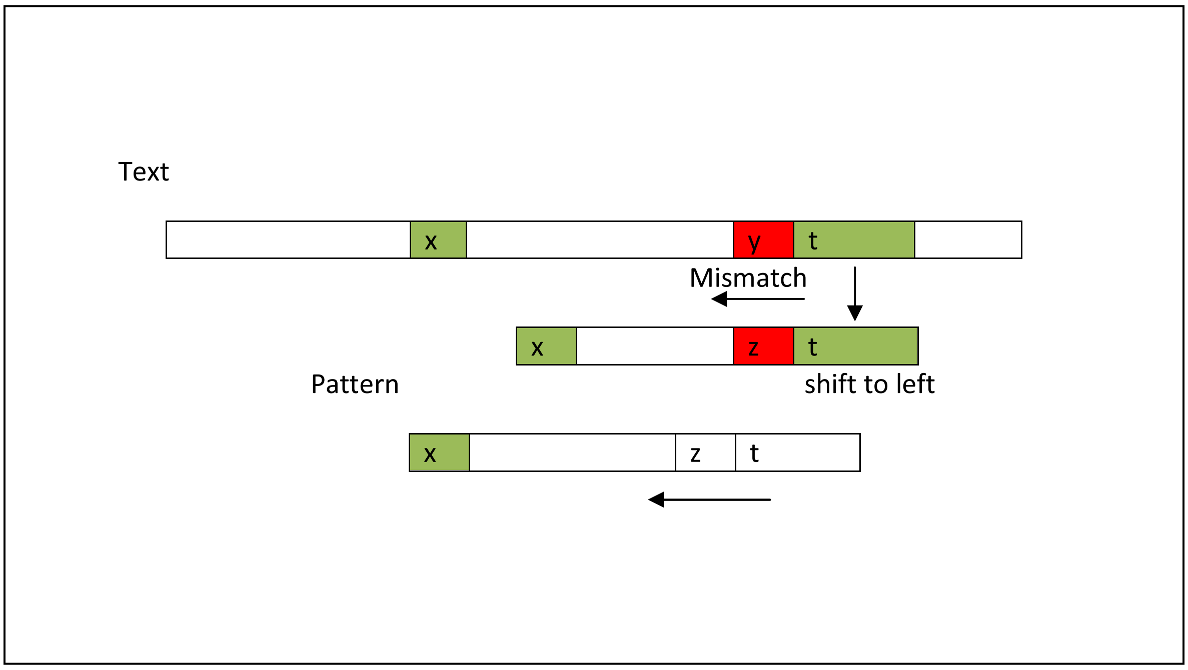


Figure (4-11): quick search inverse

4.6. Working example:

In this study we will present an example to clarify our algorithm. In this example we will explain the pre-processing phase for each side then we will scan for the pattern from both side as parallel.

Given pattern (p): BAOBAB, m = 6

Text (t): B E S S _ K N E W _ A B O U T _ B A O B A B S C G C A G B A O B A B G A G A G T A C G, n = 43

1- Pre-processing phase:

Initially, the shift values are obtained in two algorithms BM and QS since the pre-processing only be on the pattern; here we explain briefly the preprocessing for each algorithm as the follow:

- **Pre-processing phase from the left window (shift value):**

Step 1: Fill in the bad-character shift table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	-
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

Step 2: Fill in the good-suffix shift table

K	Pattern	Gs-shift
1	B A O B A B	2
2	B A O B A B	5
3	B A O B A B	5
4	B A O B A B	5
5	B A O B A B	5

Table (4-3): example of pre-processing phase from the left side

- **Pre-processing phase from the right window:**

Step1: Fill in the bad-character shift table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	-
2	1	7	7	7	7	7	7	7	7	7	7	7	7	3	7	7	7	7	7	7	7	7	7	7	7	7

2- Searching phase:

The searching process for the pattern p is illustrated through the working example as shown in Fig (4-13). At the beginning of the algorithm, in each window, the character of the pattern is compared with the corresponding character of the text while at the same time the right window also compared the character of the same pattern with the corresponding character of the text from right to left. This primary step will reduce the number of comparisons done later in the left and the right windows. We will discuss searching by the left and right windows as the following steps:

- Align the pattern against the beginning of the text
- Align the pattern against the end of the text
- Compare the corresponding characters right to left from left window:
- If no characters match, retrieve entry **t_1(c)** from the bad-symbol table for the text's character **C** causing the mismatch and shift the pattern to the right by **t_1(c).**
- If $0 < k < m$ characters are matched, retrieve entry **t_1(c)** from the bad-symbol table for the text's character **C** causing the mismatch and entry **d_2(k)** from the good- suffix table and shift the pattern to the right by

$$\mathbf{d = \max\{d_1, d_2\}}$$

$$\textbf{where } \mathbf{d_1 = \max\{t_1(c) - k, 1\}.}$$

- Compare the corresponding characters right to left from right window:
- If no characters match, retrieve entry **t_1(c)** from the bad-symbol table for the text's character **C** causing the mismatch and shift the pattern to the right by **t_1(c)**

First attempt: In the first attempt (Fig4-12a), we align the first sliding window with the text from the left. In this case, a mismatch occurs between text character (k) and pattern character (b), therefore we take the value of characters from the bad character table in BM so the window is shifted to the right by **6**.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
B	E	S	S	-	K	N	E	W	-	A	B	O	U	T	-	B	A	O	B	A	B	S............		
B	A	O	B	A	B																			
d_1 = t(K) = 6						B	A	O	B	A	B	Pattern after shift												

(a)

Second attempt: In the second attempt (Fig4-13b), we align the second sliding window with the text from the right. In this case, a mismatch occurs between text character (G) and pattern character (B), therefore, we will rely on the value of the character G at index of 42 to move the pattern to the right so we move the pattern to the right **7** steps.

29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	G	C	A	G	B	A	O	B	A	B	G	A	G	A	G	T	A	C	G
							B	A	O	B	A	B		B	A	O	B	A	B
							Pattern after shift							Bc [g] = 7					

(b)

Third attempt: In the third attempt (Fig4-13c), after match it's between character of **B** and **A** the mismatch occurs from the left between text character **(-)** and pattern character (B), therefore we take the maximum shift between two functions (bad-character, good-suffix) respectively, since, so the window is shifted to the right **5** steps.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
B	E	S	S	-	K	N	E	W	-	A	B	O	B	A	O	B	A	B	B	A	B	S............		
d_1 = *t*(_)-2 = 4 d_2(2) = 5						B	A	O	B	A	B													
											B	A	O	B	A	B	Pattern after shift							

(c)

Fourth attempt: In the fourth attempt (Fig4-13d), a mismatch occurs from the right between text character (A) and pattern character (D), therefore, we will rely on the value of the character A at index of 35 to move the pattern to the right so we move the pattern to the right **2** steps.

29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	G	C	A	G	B	A	O	B	A	B	G	A	G	A	G	T	A	C	G
							B	A	O	B	A	B							
Bc [a] = 2					B	A	O	B	A	B	Pattern after shift								

(d)

Fifth attempt: In the Fifth attempt (Fig4-13e), After match its between character of B in the text and B in the pattern ,the mismatch occurs from the left between text character (O) and pattern character (A), therefore we take the maximum shift between two functions (bad-character, good-suffix) respectively, since, so the window is shifted to the right **2** steps.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
B	E	S	S	-	K	N	E	W	-	A	B	O	B	A	O	B	A	B	B	A	B	S............		
d_1 = t(o)-1 = 2 d_2(1) = 2											B	A	O	B	A	B								
													B	A	O	B	A	B		Pattern after shift				

(e)

Sixth attempt: In the Sixth attempt (Fig4-13f), and after compare all character the pattern is found in the text from the right window at position **34(success matches)**.

29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	G	C	A	G	B	A	O	B	A	B	G	A	G	A	G	T	A	C	G
					B	A	O	B	A	B		Pattern match I = 34							

(f)

Seventh attempt: In the seventh attempt (Fig4-13g), and after compare all character the pattern is found in the text from the left window at position **13 (success matches).**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
B	E	S	S	-	K	N	E	W	-	A	B	O	B	A	O	B	A	B	B	A	B	S............		
													B	A	O	B	A	B		Pattern match I = 13				

(g)

Chapter Five

Chapter five
Experimental Results

To assess the performance of my algorithm, we considered all the well-known algorithms stated before for comparison with the BBQ algorithm. The algorithms are compared with test cases and their corresponding results are discussed.

5.1. Environment

In the experiments we used a PC with Intel(R) Core(TM) i3 CPU m 370 @ 2.40 GHz, with 3GB of RAM. The host operating system is windows 7 ultimate and the operating system type of 32-bit. The source codes were compiled using the C#. We are using of different types of texts and group of comparisons to prove the following assumptions:

- The comparison of the characters, which are consumes most of the time in the search process.
- Make a long jumps on the text reduces the comparisons and thus reduces the time spent during the search, but it is not be if the pre-processing phase does not exist to determine the length of these jumps.
- The pre-processing phase on the pattern does not provide more than the length of the pattern (m) as information's to the algorithms.
- Using the idea of search from both sides has led to reducing the time required for the search process.
- Using the BM and QS algorithms, led to increase the speed of search because both of the algorithms rely on analysis of pattern using the bad-character-table.

The speed of the algorithm was calculated by the following equation:

Speed = (number of examined of character * number of pattern) / time spent where:

- The number of character in the text that are examined = noy.
- The number of pattern = nox.
- Execution time = T.

We are note by the results in the table (5-1) there is a difference in the execution of time and speed of algorithm between the BBQ algorithm, QS and BM.

5.2. Tool used:

C# (pronounced as see sharp) is a high-level language and multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, procedural, object (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative; C# is one of the programming languages designed for the Common Language Infrastructure. C# is built on the syntax and semantics of C++, allowing C programmers to take advantage of .NET and the common language runtime.

5.3. Tests with The plant genome (Arabidopsis thaliana)

"In this section we present several experiments comparing our algorithm to the existing algorithms on different pattern sizes the below DNA sequence dataset has been taken from IBSPC for testing IPMAFC algorithm. The DNA biological sequence S of size n = 1024 and pattern P of different sizes are taken. Let S be the following DNA sequence" (Devi, et al2013).

***"AGAACGCAGAGACAAGGTTCTCATTGTGTCTCGCAATAGTGTTACCAACTCGGGT
GCCTATTGGCCTCCAAAAAAGGCTGTTCAACGCTCCAAGCTCGTGACCTCGTCACT
ACGACGGCGAGTAAGAACGCCGAGAAGGTAAGGGAACTAATGACGCGTGGTGAAT
CCTATGGGTTAGGATCGTGTCTACCCCAAATTCTTAATAAAAAACCTAGGACCCCCT
TCGACCTAGACTATCGTATTATGGACAAGCTTTAACTGTCGTACTGTGGAGGCTTCA
AAACGGAGGGACCAAAAAATTTGCTTCTAGCGTCAATGAAAAGAAGTCGGGTGTAT
GCCCCAATTCCTTGCTGCCCGGACGGCCAGGCTTATGTACAATCCACGCGGTACTA
CATCTTGTCTCTTATGTAGGGTTCAGTTCTTCGCGCAATCATAGCGGTACTTCATAA
TGGGACACAACGAATCGCGGCCGGATATCACATCTGCTCCTGTGATGGAATTGCTG
AATGCGCAGGTGTGAATACTGCGGCTCCATTCGTTTTGCCGTGTTGATCGGGAATG
CACCTCGGGGACTGTTCGATACGACCTGGGATTTGGCTATACTCCATTCCTCGCGA
GTTTTCGATTGCTCATTAGGCTTTGCGGTAAGTAAGTTCTGGCCACCCACTTCGAGA
AGTGAATGGCTGGCTCCTGAGCGCGTCCTCCGTACAATGAAGACCGGTCTCGCGCT
AAATTTCCCCCAGCTTGTACAATAGTCCAGTTTATTATCAAAGATGCGACAAATAAA
TTGATCAGCATAATCGAAGATTGCGGAGCATAAGTTTGGAAAACTGGGAGGTTGCC
AGAAAACTCCGCGCCT".***

We have implemented and tested the BBQ technique using C# and the experimental Results for the BBQ algorithm for different pattern sizes which has been chosen from the above DNA sequence, the execution time for (QS , BM and BBQ algorithm), speed of these algorithms and the ratio compare with the BM is shown in Table (5-1).

Table (5-1): the maximum number of running time and speed between three algorithms to find all position of the pattern:

Pattern	Size of pattern	Q S time (ms)	Speed QS (ms)	BM time (ms)	Speed BM(ms)	(BBQ) algorithm (ms)	Speed BBQ (ms)	Ratio compared with the BM (ms)
CAT	3	0.398	21595.6	0.465	18481.5	0.384	22383.0	17%
AACG	4	0.397	21658.1	0.149	57552.0	0.125	68679.1	16%
AAGCG	5	0.189	45529.1	0.170	50719.5	0.011	773381.3	93%
AAGCGA	6	0.134	64337.5	0.079	108544.7	0.019	456960.7	76%
AAGCGA A	7	0.005	1682974.6	0.235	36528.9	0.006	1341653.7	97%
AAAAAAG G	8	0.079	108640.7	0.018	489192.3	0.009	925726.6	47%
AAGCGA ACG	9	0.031	281413.6	0.142	60465.4	0.008	1068323.0	94%
CCTTTTC CGG	10	0.002	3706896.6	0.087	98510.9	0.007	1207865.2	92%
AAGCGA ACGAC	11	0.012	706075.5	0.132	65112.1	0.008	1140583.6	94%
AAGCGA ACGACC	12	0.042	203694.9	0.045	189135.7	0.006	1329211.7	86%

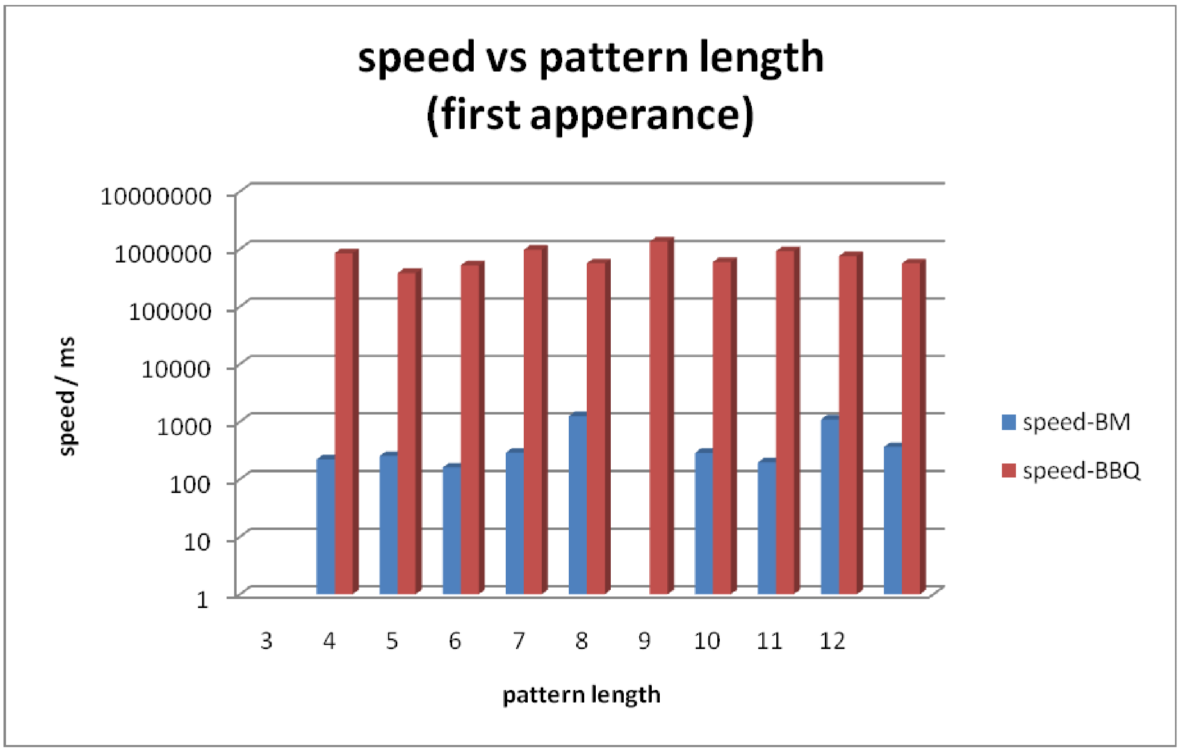


Figure (5-1): speed of the algorithms

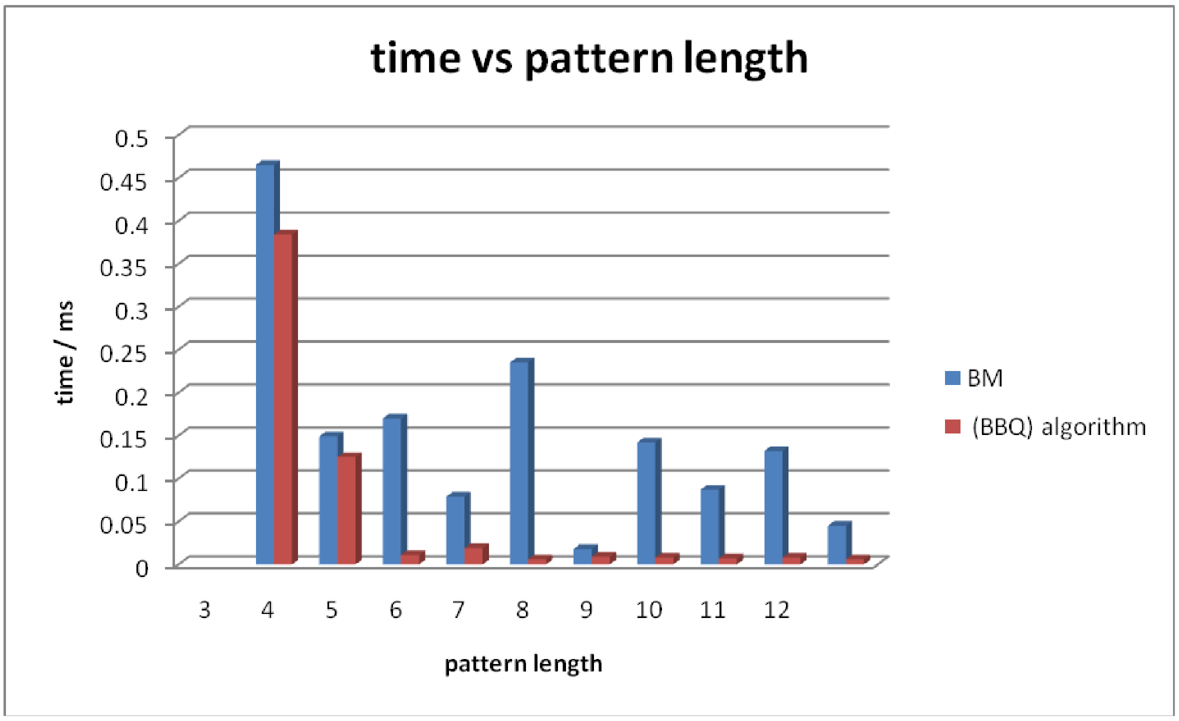


Figure (5-2): running time of the algorithms

Table (5-2): the maximum number of running time and speed between three algorithms to find the first occurrence of the pattern:

Pattern	Size	BM (*10^3)	speed-BM	QS (*10^3)	speed-QS	BBQ (*10^3)	speed-BBQ
CAT	3	39.1	219.9488491	0.0034	2529411.765	0.0102	843137.2549
AACG	4	34.1	252.1994135	0.0248	346774.1935	0.0226	380530.9735
AAGCG	5	54	159.2592593	0.0185	464864.8649	0.0166	518072.2892
AAGCGA	6	30	286.6666667	0.0119	722689.0756	0.0089	966292.1348
AAGCGAA	7	6.93	1240.981241	0.0149	577181.2081	0.0153	562091.5033
AAAAAAGG	8			0.0072	1194444.444	0.0064	1343750
AAGCGAACG	9	30.3	283.8283828	0.0119	722689.0756	0.0145	593103.4483
CCTTTTCCGG	10	44.1	195.0113379	0.0171	502923.9766	0.0094	914893.617
AAGCGAACGAC	11	7.94	1083.123426	0.0158	544303.7975	0.0115	747826.087
AAGCGAACGACC	12	23.8	361.3445378	0.0149	577181.2081	0.0153	562091.5033

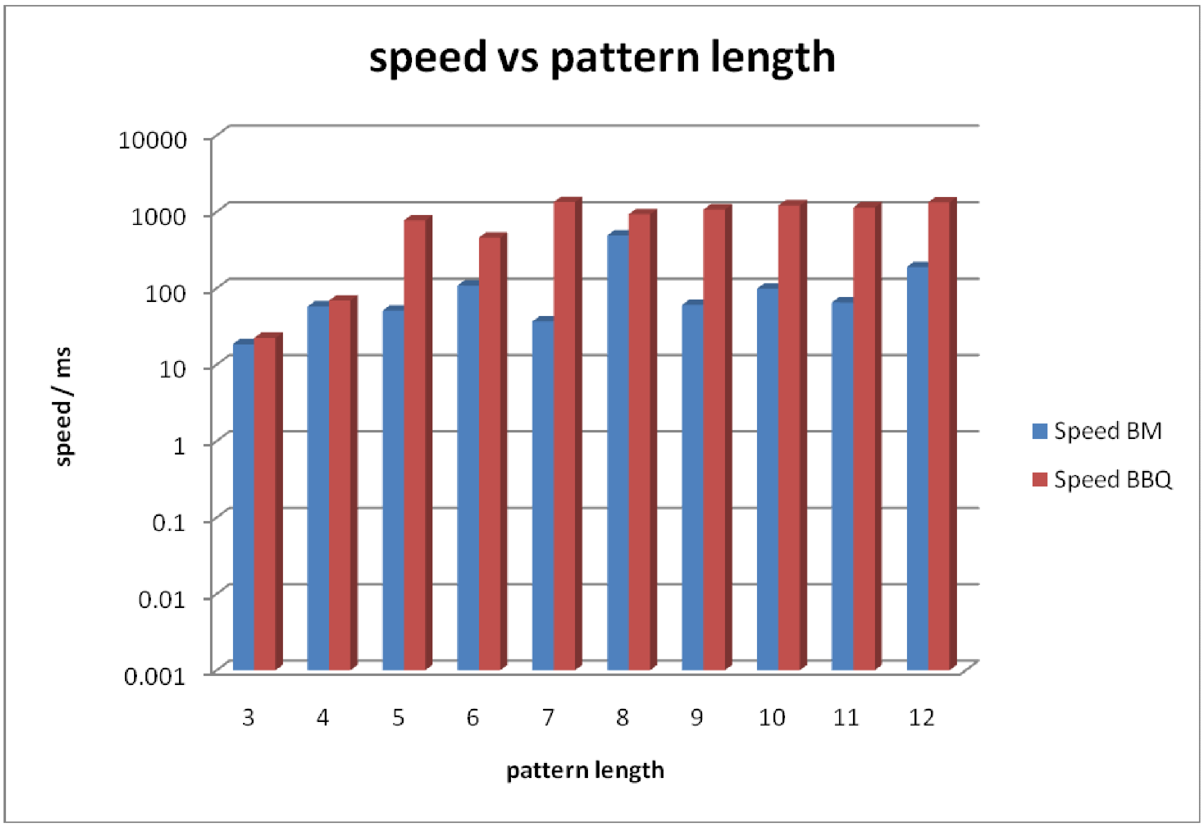


Figure (5-3): speed of the algorithms

5.4. Protein Sequence

"The below Protein Sequence dataset has taken from NCBI site Full Sequence in Fasta Format. (http://www.ncbi.nlm.nih.gov/protein/269849759?report=fasta) (Devi, et al 2013).

"MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLS

PDDIEQWFTEDPGPDEAPRMPEAAPPVAPAPAAPTPAAPAPAPSWPL

SSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKMFCQLAKTC

PVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVVRRCPHHERCSDSDG

LAPPQHLIRVEGNLRVEYLDDRNTFRHSVVVPYEPPEVGSDCTTIHYN

YMCNSSCMGGMNRRPILTIITLEDSSGNLLGRNSFEVRVCACPGRDRR

TEEENLRKKGEPHHELPPGSTKRALPNNTSSSPQPKKKPLDGEYFTL

QIRGRERFEMFRELNEALELKDAQAGKEPGGSRAHSSHLKSKKGQST

SRHKKLMFKTEGPDSD"

Table (5-3): the maximum number of running time and speed between three algorithms to find all occurrence of the pattern:

Pattern	Size	Occur	BM (ms)	speed – BM(ms)	QS (ms)	speed –QS (ms)	BBQ (ms)	speed- BBQ(ms)	Ratio
P	**1**	**45**	**2.520**	**1286.94**	**2.443**	**1286.94**	**0.290**	**10826**	**88%**
LP	**2**	**8**	**0.136**	**23151.69**	**0.055**	**57111.72**	**0.011**	**289236**	**92%**
WKL	**3**	**2**	**0.041**	**77134.45**	**0.008**	**416148.25**	**0.006**	**526633**	**85%**
PPPG	**4**	**2**	**0.026**	**122764.5**	**0.004**	**789949.75**	**0.005**	**646914**	**81%**
PAPAA	**5**	**2**	**0.030**	**105503.3**	**0.004**	**861369.86**	**0.008**	**419200**	**75%**
HHELPP	**6**	**2**	**0.048**	**65760.30**	**0.004**	**880672.27**	**0.005**	**660504**	**90%**
GTAKSVT	**7**	**2**	**0.033**	**94414.41**	**0.004**	**885633.80**	**0.006**	**566895**	**83%**
QETFSDLWKLLPENN	**15**	**2**	**0.021**	**153365.8**	**0.003**	**1107042.25**	**0.005**	**696346**	**78%**

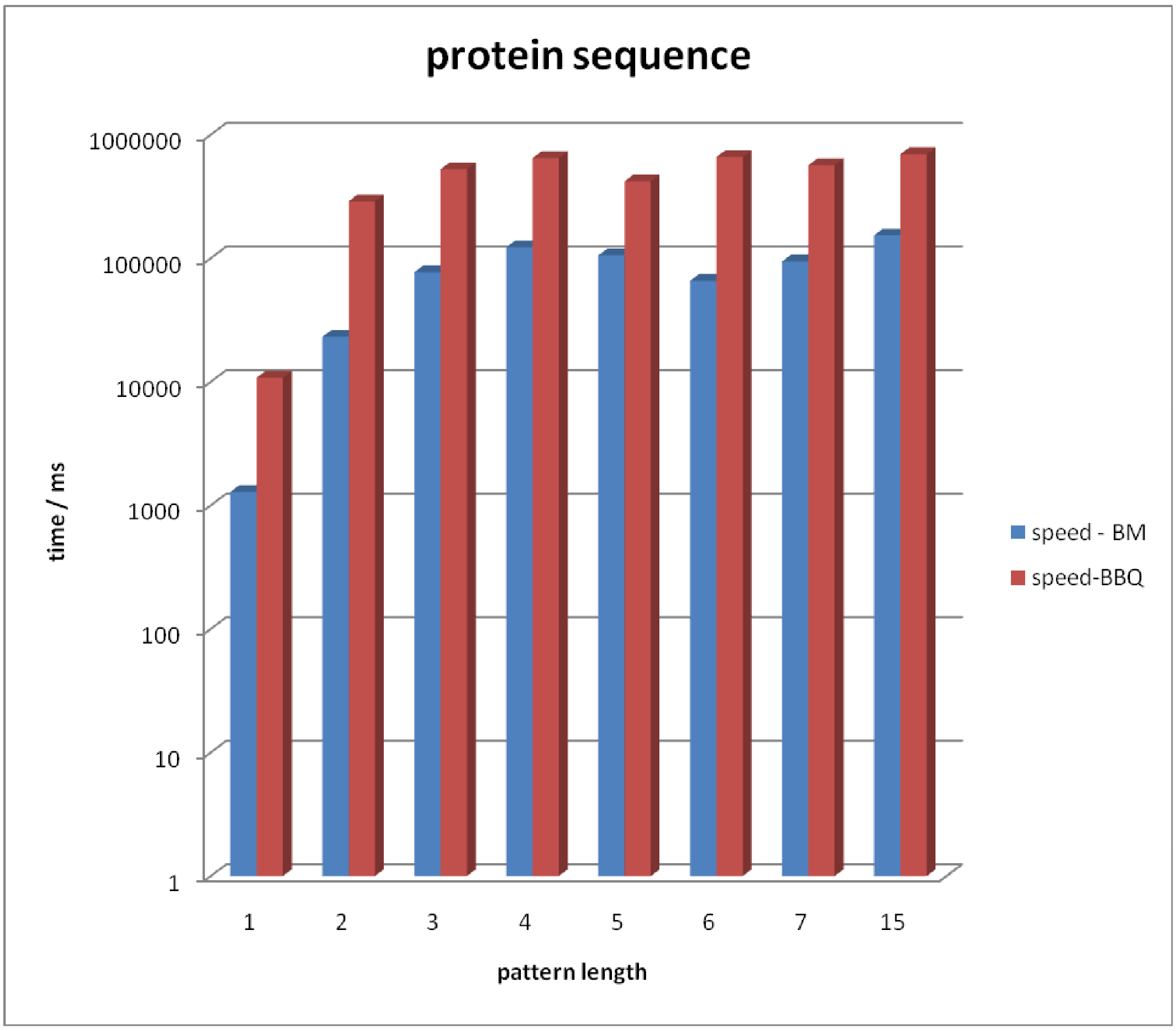


Figure (5-4): speed of the algorithms

Table (5-5) shows the experimental result of IPMAFC for the above sequence S of 393 protein characters and different pattern sizes. The number of occurrences, speed of these algorithms and the speed of the algorithm is shown in figure (5-6).

5.5. Tests with an English Text and a Random Pattern:

Let us assume, the given input text as,

"patternmatchingisoneofthebasicandmostimportantissuesi ntheresearchareasofcomputersciencethemeaningofthepatte rnmatchingisthatfindingtheoccurrencesofagivenpatterninth egiventext",

And the random patterns as "a", "of", "and", "most",, "pattern", "matching", of sizes, 1, 2 …8, respectively. The results were compared with the BM and Quick Search algorithm and were shown in table (5-6):

Table (5-4): the maximum number of running time and average between three algorithms to find all occurrence of the pattern:

Pattern	Size	Occur	QS(ms)	BM(ms)	Max BBQ (ms)	Average BBQ (ms)
A	**1**	**14**	**0.02867**	**0.44192**	**0.0249**	**0.0219**
OF	**2**	**4**	**0.00643**	**0.00944**	**0.0065**	**0.0056**
AND	**3**	**2**	**0.00349**	**0.00516**	**0.0050**	**0.0042**
MOST	**4**	**2**	**0.00268**	**0.005011**	**0.0030**	**0.0025**
PATTERN	**7**	**2**	**0.003042**	**0.00512**	**0.0046**	**0.0036**
MATCHING	**8**	**2**	**0.002923**	**0.00535**	**0.0050**	**0.0040**

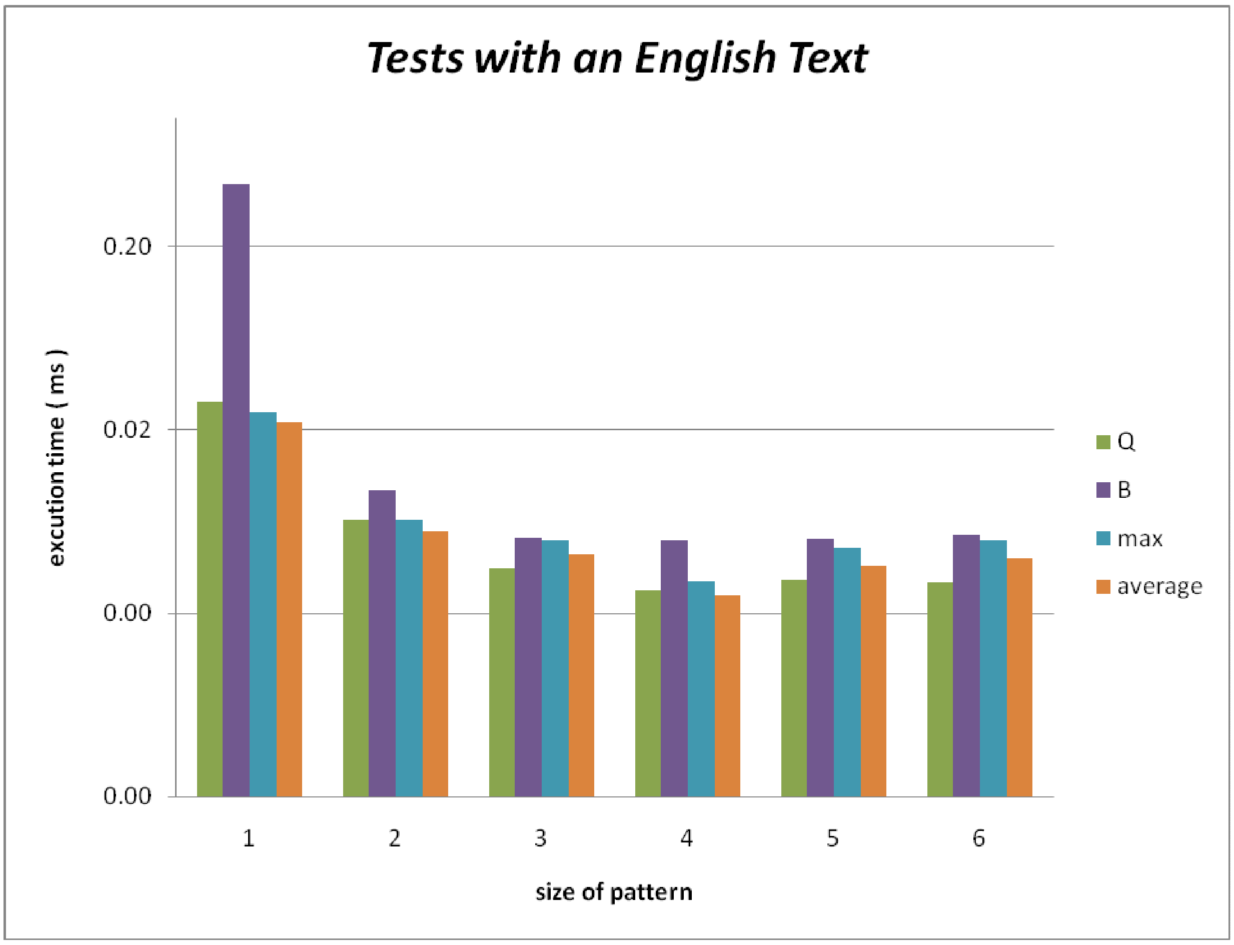


Figure (5-5): comparing between the algorithms

Table (5-5): the maximum number of running time between three algorithms to find the first occurrence of the pattern:

Pattern	Size	Occur	QS(ms)	BM(ms)	BBQ(ms)
A	1	14	0.000004	0.0453	0.000003
OF	2	4	0.000004	0.0010	0.000004
AND	3	2	0.000002	0.0007	0.000001
MOST	4	2	0.000004	0.0012	0.000004
PATTERN	7	2	0.000005	0.0010	0.000004
MATCHING	8	2	0.000004	0.0011	0.000003

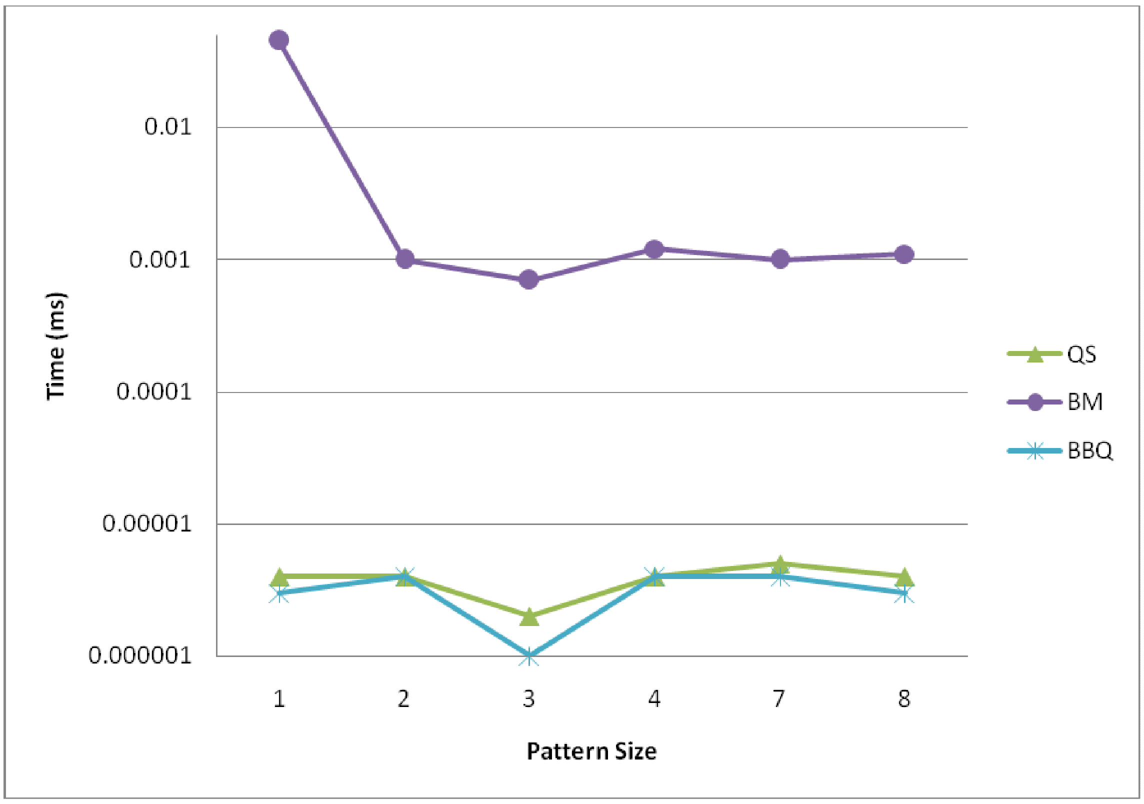


Figure (5-6): time to find the first occurrence for the algorithms

5.6. Test with different length of the text

Table (5-6): the maximum number of running time between BM algorithms and our (BBQ) of algorithm to find all occurrences of the pattern:

TEXT (SIZE)	Occur	BM(ms)	BBQ algorithm(ms)
1463	11	0.17	0.03
1872	16	0.50	0.05
2179	15	0.59	0.04
2527	16	0.61	0.05
5054	32	0.80	0.28
8778	60	1.50	0.66

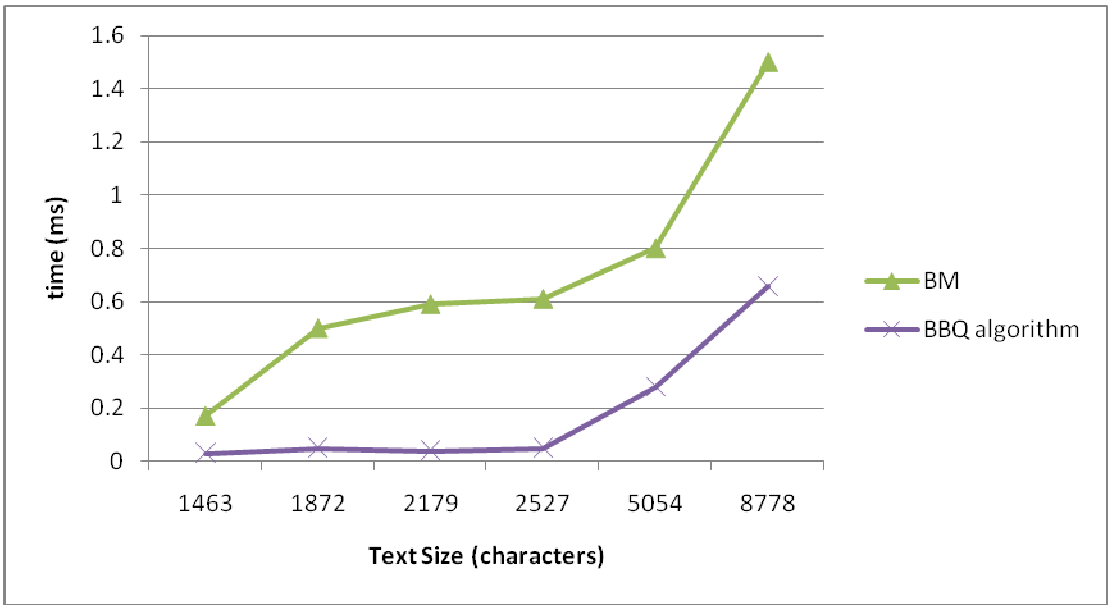


Figure (5-7): Comparison between BM and BBQ for different lengths of text (all occurrences)

Table (5-7): the maximum number of running time between BM algorithms and our BBQ of algorithm to find the first occurrence of the pattern:

TEXT (SIZE)	OCCURE	BM(ms)	BBQ(ms)
1463	11	0.0200680	0.0000016
1872	16	0.0305630	0.000002
2179	15	0.0011100	0.0000101
2527	16	0.0434000	0.0000044
5054	32	0.0668560	0.0000028
8778	60	0.0517160	0.0000016

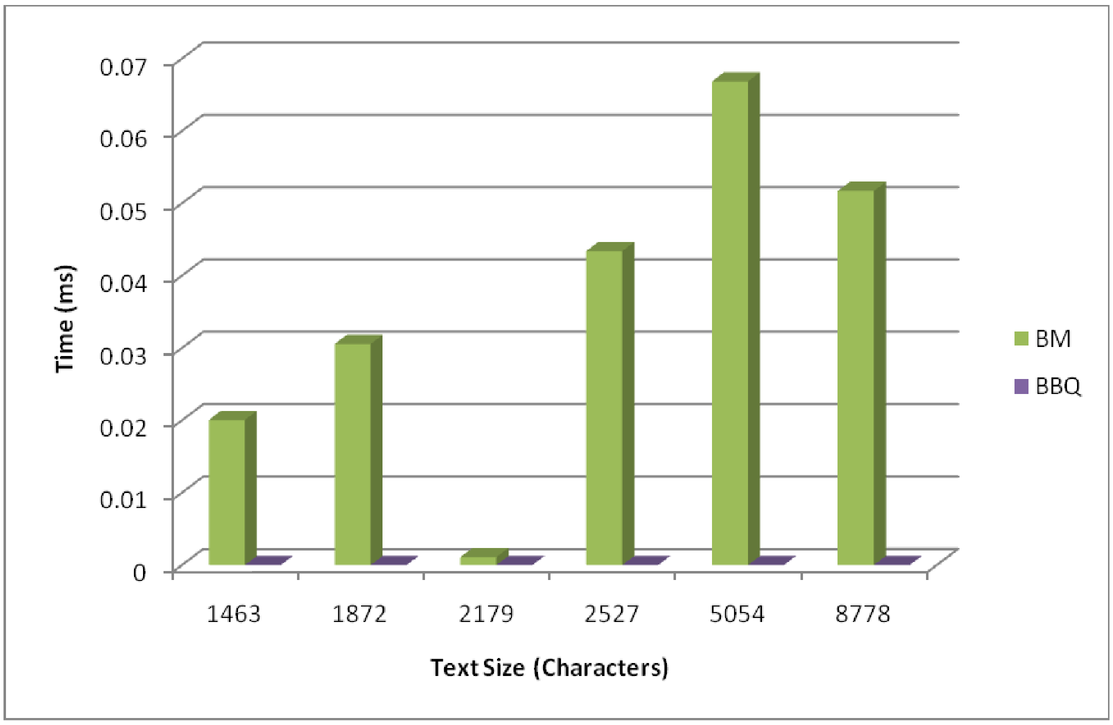


Figure (5-8): Comparison between BM and BBQ for different lengths of text (Finding First Occurrence)

Table (5-8): the maximum number of running time between BM algorithms and our BBQ of algorithm, number of shifts and comparisons:

Algorithm	Pattern	no.comp	no.shift	Time(ms)	First appearance
BM	7	53	32	0.007580	0.001005
	8	41	27	0.005530	0.000980
	12	93	89	0.009163	0.001046
BBQ	7	26	15	0.005700	0.000002
	8	21	14	0.004800	0.000005
	12	38	38	0.006190	0.000003

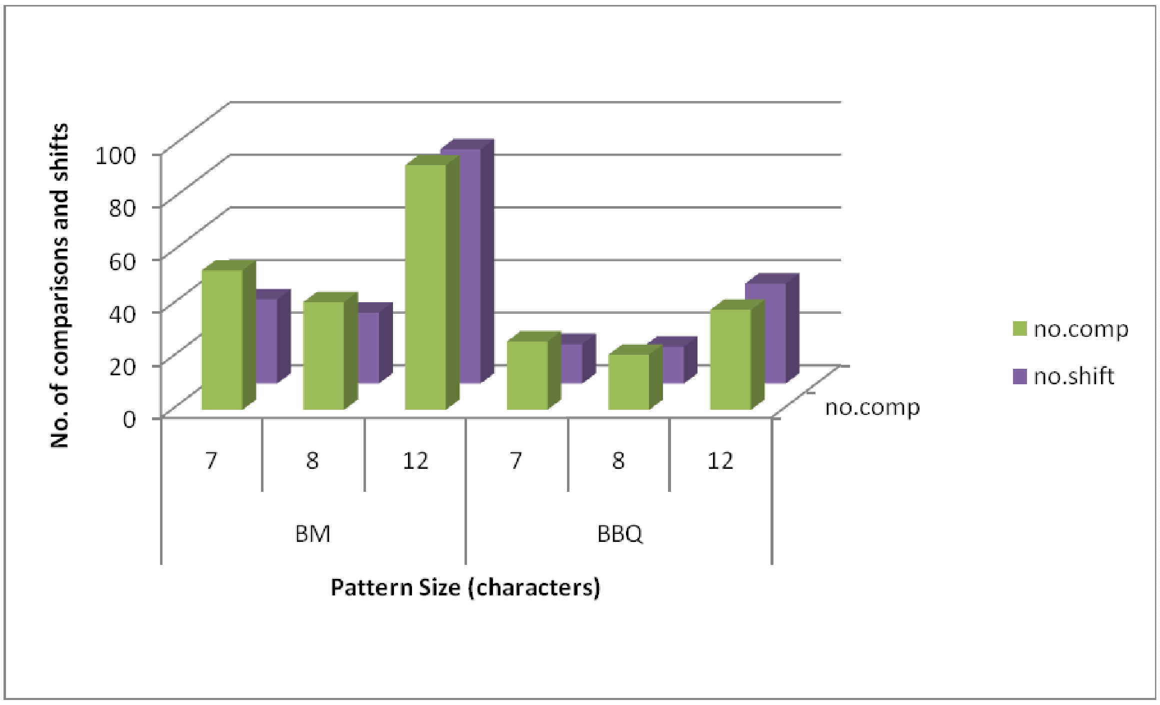


Figure (5-9): comparing No. of shifts and Comparisons between BM and BBQ

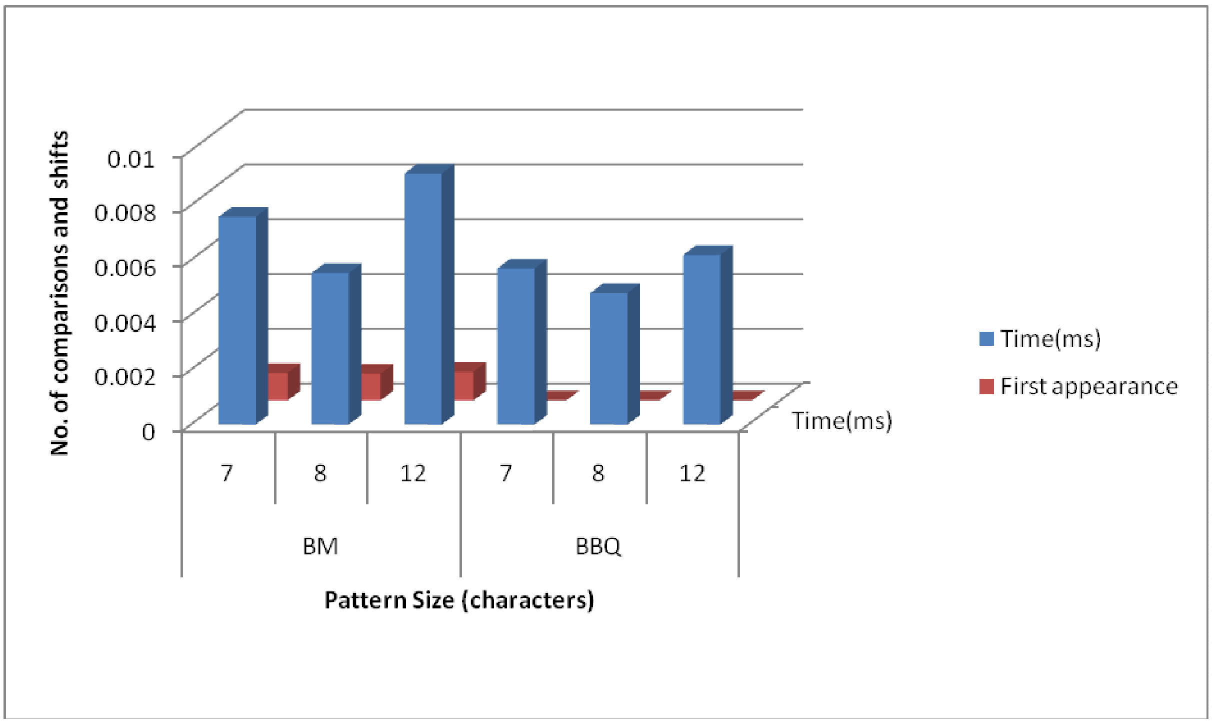


Figure (5-10): comparing Total time and First appearance time between BM and BBQ

5.7. The Asymptotic Analysis:

The BBQ was proposed to divide the searching time of the BM by two, in other words, by searching from both sides of the text simultaniusly the time of searching a full length text should be theoritcially divided by two. Therfore, the complexity of the BBQ can be given as follows:

A- Worst Case Time Complexity:

Worst case complexity is what matters when comparing algorithms when measuring their performance. For the BM and QS, the worst case complexity is O(nm) where m is the length of the pattern, and n is the length of the text, and since the BBQ - theortically – uses half of the time the BM or the QS use, its worst case complexity should be O(nm)/2, therefore, by neglecting the constant multiplier, the worst case time complexity is O(nm)

B- Space Complexity:

Similar to space complexity of the QS and BM, which is $O(m+|\Sigma|)$, where $\sum$ is the alphabet. The BBQ's space complexity is $O(m+|\Sigma|)$ because the BBQ uses the extra space that the BM uses for pre-processing phase (constructing the bad-character and good-suffix tables) plus the extra space used by the QS algorithm for the same reason, hence by neglecting the constant multiples, the space complexity of the BBQ is $O(m+|\Sigma|)$

Chapter Six

Chapter six

Conclusion and Future Work

6.1. Conclusion

The BBQ algorithm for pattern matching was proposed as a combination between two efficient algorithms BM and QS. The BBQ utilizes the pre-processing and searching mechanisms used by the BM and QS algorithms to search from both sides of the text simultaneously. On the one hand, the BBQ searches bidirectional for match in the text, on the other hand it utilizes BM and QS to implement the bidirectional phase of the search process.

The BBQ was evaluated using 3 testing benchmarks, one for matching a pattern with a DNA sequence, then matching a pattern with a protein sequence; finally, the BBQ was tested to match a pattern with a normal text.

In terms of complexity, the BBQ algorithm has shown similar worst case time complexity and space complexity to BM and QS, the worst case time complexity is $O(nm)$ and the space complexity is $O(m+\Sigma)$

The concept of searching the text from both sides simultaneously gives the BBQ algorithm a preference over other algorithms in the number of comparisons and attempts especially if the pattern searched exists at the end of the text because when the BM fails to find it quickly, the QS will find it immediately utilizing the Bidirectional search property,

6.2. Recommendations and Future work

In future research, we intend to implement our approach on real parallel processors to minimize the number of comparisons and attempts. Also we intend to implement the idea of search from both sides on the pattern and text rather than on the text alone.

References

Boyer, Robert S., and J. Strother Moore. "A fast string searching algorithm." Communications of the ACM 20.10 (1977): 762-772.

Choudhary, R., Rasool, A., & Khare, N. Variation of Boyer-Moore String Matching Algorithm: A Comparative Analysis.

Crochemore, M., & Lecroq, T. (2008). A fast implementation of the Boyer-Moore string matching algorithm. *submitted for publication*.

Devi, S. N., Rajagopalan, S. P., & Anuradha, V. (2013). Index Based Multiple Pattern Matching Algorithm Using Frequent Character Count in Patterns. *International Journal*, *3*(5).

Diwate, M. R. B., & Alaspurkar, S. J. (2013). Study of Different Algorithms for Pattern Matching. *International Journal*, *3*(3).

Franek, F., Jennings, C. G., & Smyth, W. F. (2007). A simple fast hybrid pattern-matching algorithm. *Journal of Discrete Algorithms*, *5*(4), 682-695.

Hasan, A. A., & Rashid, N. A. A. (2012). Hash-Boyer-Moore-Horspool String Matching Algorithm for Intrusion Detection System. *International Proceedings of Computer Science & Information Technology*, *35*.

Hasan, A. A., & Rashid, N. A. A. (2012). Hash-Boyer-Moore-Horspool String Matching Algorithm for Intrusion Detection System. *International Proceedings of Computer Science & Information Technology*, *35*.

Hussain, I., Ali, I., Zubair, M., & Bibi, N. (2010, June). Fastest approach to exact pattern matching. In *Information and Emerging Technologies (ICIET), 2010 International Conference on* (pp. 1-5). IEEE.

Hussain, I., Kazmi, S. Z. H., Khan, I. A., & Mehmood, R. Improved-Bidirectional Exact Pattern Matching.

Itriq, M., Hudaib, A., Al-Anani, A., Al-Khalid, R., & Suleiman, D. (2012). Enhanced Two Sliding Windows Algorithm For Pattern Matching (ETSW). *Journal of American Science*, *8*(5).

Karp, R. M., & Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31(2), 249-260.

Moh'd Mhashi, M., & Alwakeel, M. (2010). New Enhanced Exact String Searching Algorithm. *IJCSNS*, *10*(4), 193.

Papanicolau, G., A. Bensoussan, and J-L. Lions. Asymptotic analysis for periodic structures. Elsevier, 1978.

Pendlimarri, D., & Petlu, P. B. B. (2010). Novel Pattern Matching Algorithm for Single Pattern Matching. *International Journal on Computer Science & Engineering*.

Pendlimarri, D., Petlu, P., & Satrasala, R. (2011). Novel Devaki-Paul Algorithm for Multiple Pattern Matching. International Journal of Computer Applications, 13(3), 37-42.

Rasool, A., & Khare, N. (2013). Performance Improvement of BMH and BMHS using PDJ (Possible Double Jump) and MValue (Match Value). *International Journal of Computer Applications*, *72*.

Sedgewick, R., & Flajolet, P. (2013). An introduction to the analysis of algorithms. Addison-Wesley.

Senapati, K. K., Adhikary, D. D., & Sahoo, G. (2012). An Application of Pattern Matching for Motif Identification. International Journal of Biometrics and Bioinformatics (IJBB), 6(5), 135.

Suleiman, D., Hudaib, A., Al-Anani, A., Al-Khalid, R., & Itriq, M. (2013). ERS-A Algorithm for Pattern Matching. *Middle East Journal of Scientific Research*, *15*(7).

Xian-feng, H., Yu-bao, Y., & Lu, X. (2010, August). Hybrid pattern-matching algorithm based on BM-KMP algorithm. In Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on (Vol. 5, pp. V5-310). IEEE.

Yuan, J., Yang, J., & Ding, S. (2012, October). An Improved Pattern Matching Algorithm Based on BMHS. In *Distributed Computing and Applications to Business, Engineering & Science (DCABES), 2012 11th International Symposium on* (pp. 441-445). IEEE.

Yuan, J., Zheng, J., & Ding, S. (2010, April). An Improved Pattern Matching Algorithm. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on* (pp. 599-603). IEEE.

Appendices

Appendix A:

Programming the algorithm of BM using C++

```
Void preBmBc( CHAR * X , INT m ,INT BmBc [] ) {
IntI ;
for ( i=0 ; I < ASIZE ;++I )
bmBc[i] = m ;
for ( i= 0 ; I < m-1 ; ++I )
bmBc[x[i]] = m-i-1 ;
}
suff*int  ,int m ,x*char )Voidsuffixes)
{
Int f, g,I;
Suff [m-1] = m ;
G = m-1 ;
For ( I =m-2 ; i>=0 ; --I ) {
if (i>g &&suff[i+m-1-f]<i-g)
suff[i]=suff[i+m-1-f];
else {
if (i<g)
g=I;
f=i
while (g>=0 && x[g]== x[g+m-1-f])
--g;
Suff[i]=f-g;
          }
       }
}
Void preBmGs(char *x, int m, intbmGs[]) {
Int I, j, suff[XSIZE];
Suffixes(x, m, suff);
for (i=0; i<m; ++i)
bmGs[i]=m;
```

```
j=0;
for (i=m-1;i>=-1; --1)
if (i==-1 ||suff[i] ==I +1)
for (; j <m-1-I; ++j)
if (bmGs[j] ==m)
bmGs[j] = m-1-I;
for (i=0; i<=m-2; ++i)
bmGs[m-1-suff[i]] = m-1-I;
}
Void BM(char *x, int m, char *y, int n) {
int I, j, bmGs[XSIZE], bmBc[ASIZE];
/* preprocessing */
preBmGs(x, m, bmGs);
preBmBc(x, m, bmBc);
/* searching */
J=0;
While (j<= n-m) {
for (i=m-1;i>=0 && x[i] == y[i+j]; --i);
if (i<0) {
OUTPUT(j);
J+=bmGs[0];
}
else
J+= MAX(bmGS[i], bmBc[y[i+j]]-m+1+i);}}
```

Appendix B

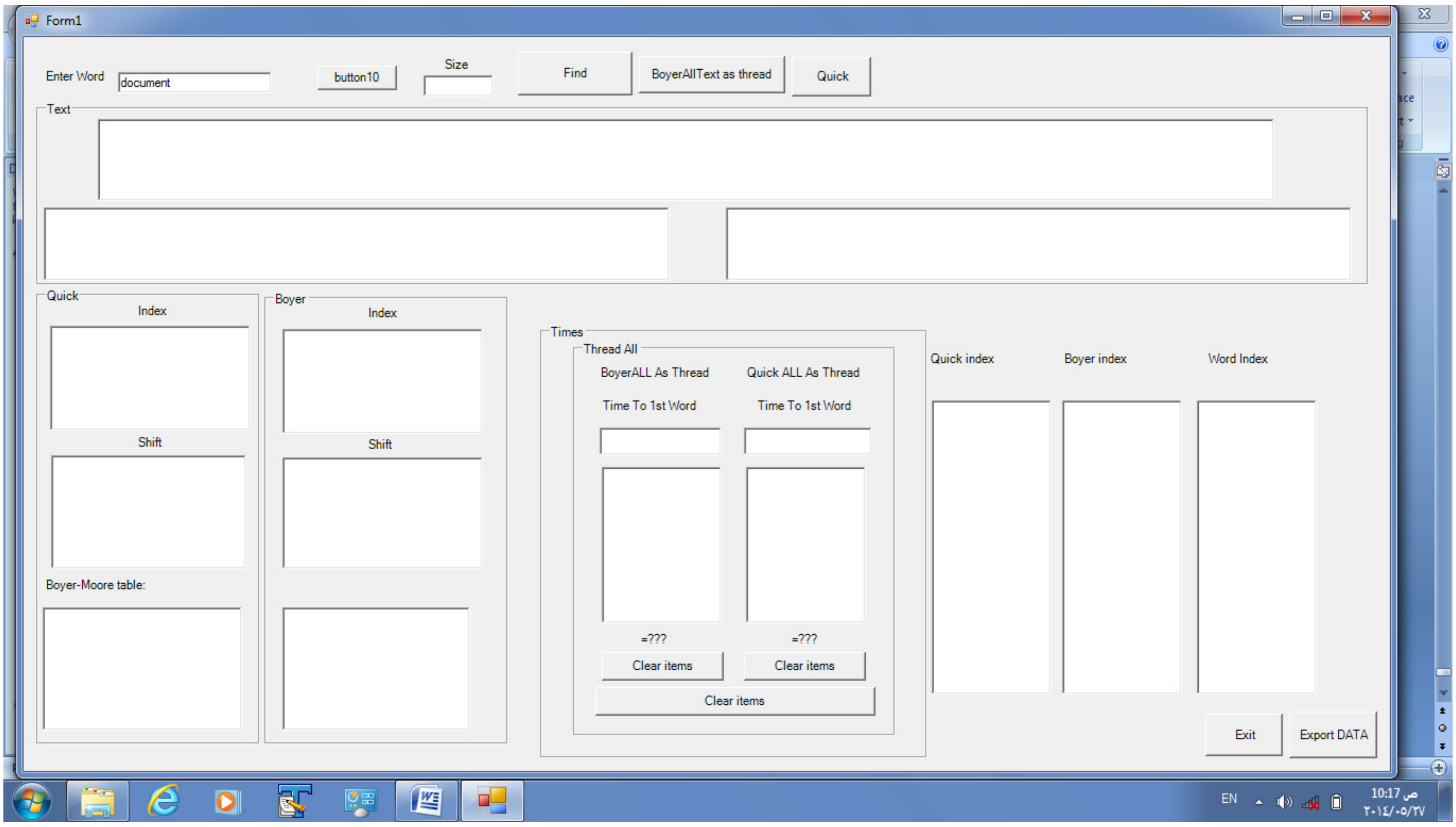
Programming the algorithm of QS using C++

```
Void preQsBc(char *x, int m, intqsBc[]) {
Int I;
for (i=0; i<ASIZE; ++i)
qsBc[i]=m+1;
for (i=0; i<m; ++i)
qsBc[x[i]]=m-I;
}
Void QS(char *x, int m, char *y, int n) {
Intj ,qsBc[ASIZE];
/* preprocessing */
preQsBc(x, m, qsBc);
```

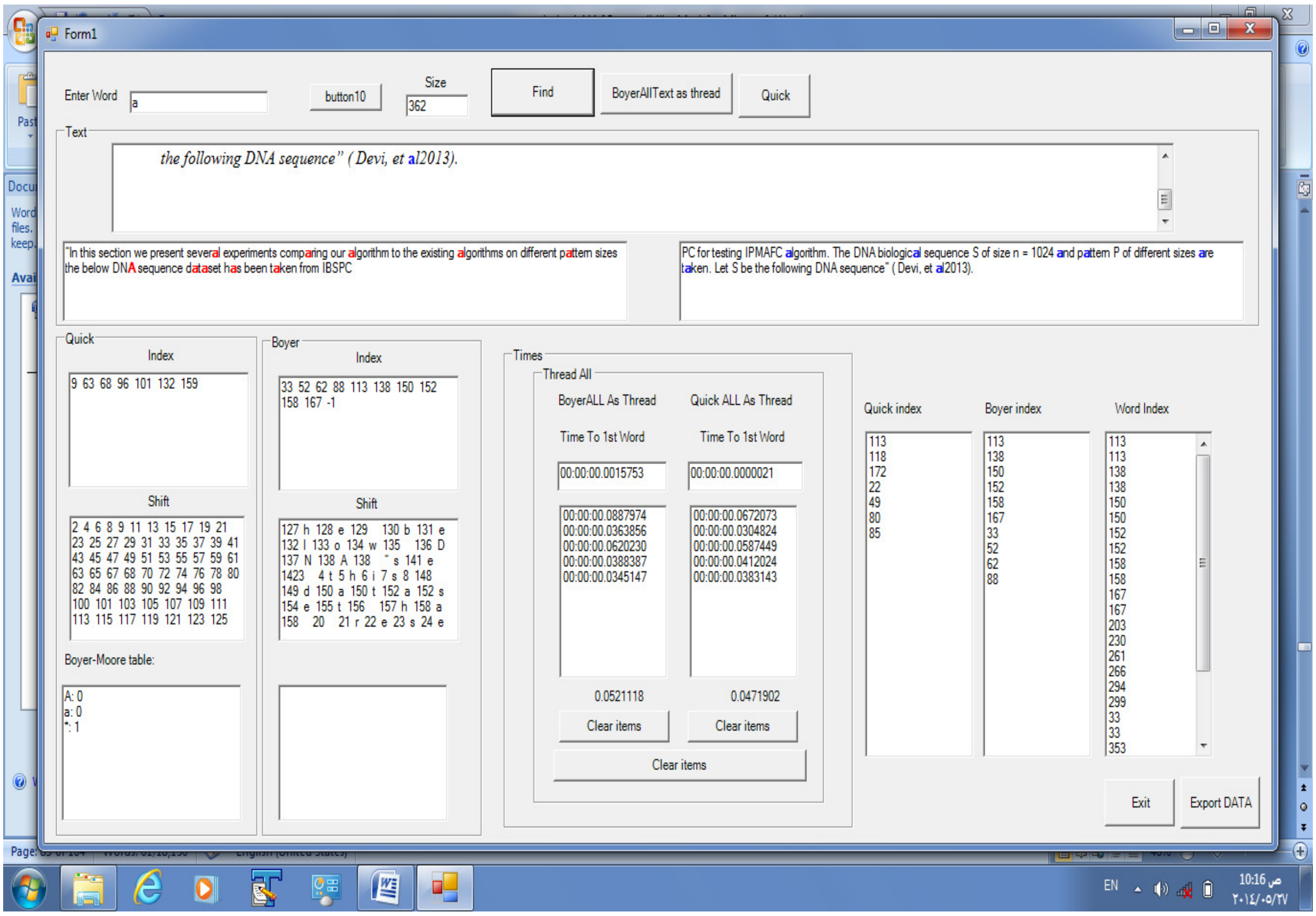
```
/* Searching */
J=0;
While (j<=n-m) {
If (memcmp(x, y+j, m)==0)
OUTPUT(j);
j+=qsBc[y[j+m]];                    /*shift*/
        }
```

Appendix d

The following figure represents the main screen of the program shows where the search process from both sides by the division of the text .



The following figure represents the partitioning process that occurs during the search, as shown in shades of red and blue and the program gives a set of specific information such as the number of comparisons and the amount of displacement.



Enter Word: a

Size: 362

Text:

the following DNA sequence" (Devi, et al2013).

"In this section we present several experiments comparing our algorithm to the existing algorithms on different pattern sizes the below DNA sequence dataset has been taken from IBSPC

PC for testing IPMAFC algorithm. The DNA biological sequence S of size n = 1024 and pattern P of different sizes are taken. Let S be the following DNA sequence" (Devi, et al2013).

Quick

Index
```
9 63 68 96 101 132 159
```

Shift
```
2 4 6 8 9 11 13 15 17 19 21
23 25 27 29 31 33 35 37 39 41
43 45 47 49 51 53 55 57 59 61
63 65 67 68 70 72 74 76 78 80
82 84 86 88 90 92 94 96 98
100 101 103 105 107 109 111
113 115 117 119 121 123 125
```

Boyer-Moore table:
```
A: 0
a: 0
*: 1
```

Boyer

Index
```
33 52 62 88 113 138 150 152
158 167 -1
```

Shift
```
127 h 128 e 129   130 b 131 e
132 l 133 o 134 w 135   136 D
137 N 138 A 138  " s 141 e
1423   4 t 5 h 6 i 7 s 8 148
149 d 150 a 150 t 152 a 152 s
154 e 155 t 156   157 h 158 a
158   20   21 r 22 e 23 s 24 e
```

Times

Thread All

BoyerALL As Thread	Quick ALL As Thread
Time To 1st Word	Time To 1st Word
00:00:00.0015753	00:00:00.0000021
00:00:00.0887974	00:00:00.0672073
00:00:00.0363856	00:00:00.0304824
00:00:00.0620230	00:00:00.0587449
00:00:00.0388387	00:00:00.0412024
00:00:00.0345147	00:00:00.0383143
0.0521118	0.0471902

Quick index	Boyer index	Word Index
113	113	113
118	138	113
172	150	138
22	152	138
49	158	150
80	167	150
85	33	152
	52	152
	62	158
	88	158
		167
		167
		203
		230
		261
		266
		294
		299
		33
		33
		353

EN 10:16 ص ٢٠١٤/٠٥/٢٧