**KCA UNIVERSITY**

SCHOOL OF INFORMATION TECHNOLOGY

(Department of Information Technology)

***Brute Force Search Algorithm, also Exhaustive Search Algorithm, Simulation***

Supervisor Submitted

Clive onsomuPascal Onyango

Submitted in partial fulfillment of the requirements of the diploma in Information

## DECLARATION

I hereby declare that I carried out the work reported in this report and I solemnly declare that to the best of my knowledge, no part of this report has been submitted here or elsewhere for the award of a diploma. All sources of knowledge used have been dully acknowledged.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

## Students name

## Pascal onyango

Signature:

## Supervisor

Clive onsomu

Signature:

## ACKNOWLEDGEMENT

I express my gratitude to everyone whose efforts helped me reach this far. Feel appreciated.

## ABSTRACT

**Brute-force search** or **exhaustive search**, also known as **generate and test**, is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. This means that it takes much more time to solve a problem as compared to other algorithms. Certain well known problems where it is applied include solving the Travelling Tradesman Problem which is tied to the Hamiltonian Circuit. It is also used in explaining and solving the Knapsack Problem. The main objective of this project is to help people understand the underlying principles and methods of the Brute Force search algorithm. One of its areas of application is in sub-string search. This is where a particular string of words or text (T) is given and the algorithm is used to find a pattern (P) of strings from the given text. This project uses simulation to help learners understand how exhaustive search occurs by creating a webpage which employs DOM manipulation to elaborate how a pattern (P) is located from a string of text (T).This algorithm is also used as a method to benchmark other algorithms or meta-heuristics. This is due to the fact that it is the simplest meta-heuristic. However, its shortcomings are, among others, the fact that it is slow.

CONTENTS [DECLARATION i](#_bookmark0)

[ACKNOWLEDGEMENT ii](#_bookmark1)

[ABSTRACT iii](#_bookmark2)

[LIST OF FIGURES v](#_bookmark3)

[CHAPTER ONE 1](#_bookmark4)

[INTRODUCTION 1](#_bookmark5)

[CHAPTER TWO 2](#_bookmark6)

[BACKGROUND INFORMATION 2](#_bookmark7)

[CHAPTER THREE 4](#_bookmark9)

[METHODOLOGY 4](#_bookmark10)

[CHAPTER FOUR 6](#_bookmark12)

[RESULTS 6](#_bookmark13)

[CHAPTER FIVE 8](#_bookmark17)

[DISCUSSION 8](#_bookmark18)

[CONCLUSION 8](#_bookmark19)

[RECOMMENDATIONS AND FUTURE WORK 9](#_bookmark20)

[Future Work 9](#_bookmark21)

[REFERENCES AND REFERENCE LIST 10](#_bookmark22)

[APPENDICES 11](#_bookmark23)

## LIST OF FIGURES

**[Fig. 1.1:](#_bookmark8)** [Brute Force Search Algorithm illustration 2](#_bookmark8)

**[Fig. 1.2:](#_bookmark11)** [Project flowchart 5](#_bookmark11)

**[Fig. 1.3:](#_bookmark14)** [User interface and character alignment 6](#_bookmark14)

**[Fig. 1.4:](#_bookmark15)** [Search found (Results) 7](#_bookmark15)

**[Fig. 1.5:](#_bookmark16)** [Search not found (Results) 7](#_bookmark16)

## CHAPTER ONE

# INTRODUCTION

Brute Force Algorithm is one of the most used algorithms when it comes to implementing sequential methods. Exhaustive or sequential search entails the evaluation of all possible values that answer to a particular criterion defined by the user or programmer so as to find or reach a result.

This project implements the same algorithm in explaining how sub-string search is done. A user searches for a pattern (P) from a string of text (T) and it shows the results. The simulation used is able to show how the pattern moves through the text (T) until the search (match) is found or not. This breaks the transparency in most systems where the users do not understand what goes on in the background during a search process.

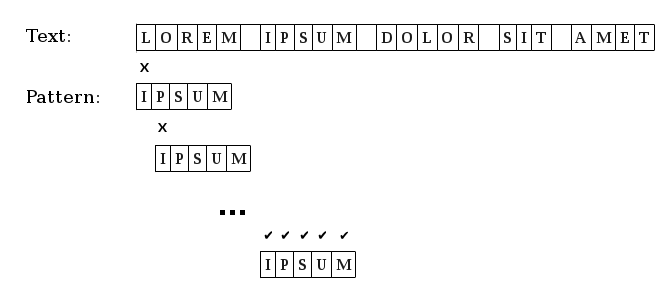
For its implementation, a webpage is developed using HTML, CSS and JavaScript. The project is supported and run by Chrome and Opera browsers since they are able to handle most of the scripts used. Other browsers such as Internet Explorer and Microsoft Edge have a variance when it comes to pixels hence they interfere with the simulation process.

## CHAPTER TWO

# BACKGROUND INFORMATION

Brute Force Search algorithm is extensively used in searching to solve various problems by finding every possible solution through analysis of every component or aspect of the said problem. In the case of sub-string search, the algorithm aligns the first position of the pattern (P) with that of the text (T) and then compares their characters one by one until all pattern characters are found or a mismatch is detected.

The figure below illustrates how it occurs:



**Fig. 1.1: Brute Force Search Algorithm Illustration-Adapted from [www.java.net](http://www.java.net/) (posted by Abdel Martinez)**

From the figure above it becomes easy to understand how the algorithm matches the pattern to the text. Rewriting or drawing the pattern under the text becomes tedious especially when illustrating the process to students where the text and pattern have many characters. This creates the need to develop a simulation that can be easily used as an aid when illustrating how brute force search algorithm works.

According to [www.java.net](http://www.java.net/) [1] Exhaustive Search or Brute Force Search Algorithm has the following characteristics:

* The pattern is not preprocessed.
* The algorithm compares from left to right character by character.
* The algorithm returns the first occurrence of the pattern

**AIMS AND OBJECTIVES**

* To create a simulation showing how the pattern moves through the text during the search.
* To make use of Document Object Module manipulation and showcase ability to apply knowledge gained from Data Structures coursework.
* To enhance creativity and critical thinking in solving problems
* To test on research and development skills in programming and scripting using JavaScript, CSS and HTML

## CHAPTER THREE

# METHODOLOGY

In order to carry out brute force search algorithm simulation, a webpage is required in this project as an interface through which users can input both text and pattern when carrying out sub-string search. The webpage is the prototype used to test and aids in showing how sub-string search occurs. It makes is easy to format user input and the output generated by the project.

It is important to distinguish the text (T) from the pattern (P) in terms of font color. Font color is also used to show whether a match has been found or not. When a match is found, the font color of the matching text is changed to that of the pattern which in this case is blue.

The movement of the pattern through the text from left to right is done in such a way that the characters in both pattern and text are vertically aligned to each other. The text (T) paragraph is above the pattern (P) paragraph. This avoids the overlap of characters. In most font faces, the characters have the same **x**-height but not the same **n** and **m** width. When using *Times New Roman*, for example, characters M and L do not have the same width. This makes it hard to achieve the vertical alignment of characters in both the text (T) and pattern (P). To overcome this challenge, *Courier New* font face is used since it gives all characters the same width and helps to achieve the vertical alignment of characters.

When a match has been established and the matching text (T) font color has been changed to that of the pattern (P), the movement of the pattern from left to right is automatically stopped. This is followed by a notification “*Search has been found*”, otherwise, the pattern moves all through the text then it is stopped at the end of the text (T) and “*Search not found*” notification appears.

It is important to determine how the pattern (P) is able to move from left to right through the text

1. on the webpage. JavaScript provides for methods that can be used to time execution of methods or commands in functions. In this project, *set Timeout ()* method is used to facilitate the movement of pattern (P) over a predetermined amount of pixels after a set time interval. This method is implemented by the use of an ID that can be used by other functions to call it. Its ID is instrumental in stopping the movement of the pattern (P) by invoking another method, *clear Timeout (),* that uses that ID to stop the set time interval counter.

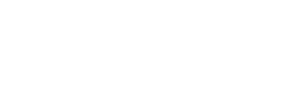
The project can be illustrated using a flowchart as shown below. It summarizes the major parts of the project and shows the flow of methods from the beginning to the end of the search process.



START

Enter Text

Enter Pattern



|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
| Yes | String Search  **Match Found** | | No |
|  |  |

**Fig. 1.2: Project flowchart.**



Change Matched

Text Color

Move pattern to

end of text

Search found

Search not found

STOP

## CHAPTER FOUR

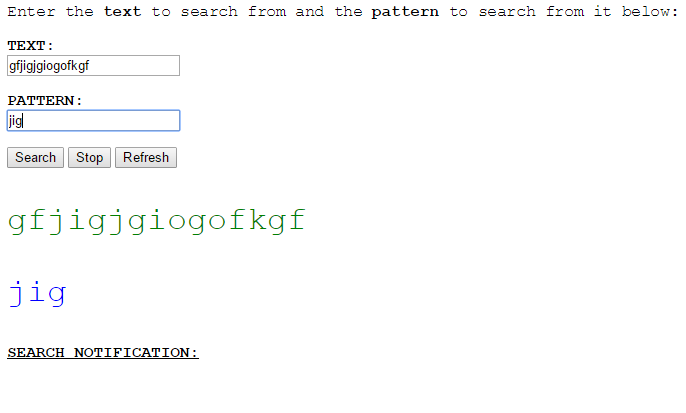
**RESULTS**

With a lot of research on how to implement the various parts of the sub-string search simulation, I was able to come up with a webpage that meets the set objectives of the project.

Firstly, there is the aspect of improving research skills. This was greatly achieved since in most aspects, getting the right brute force algorithm implementation in JavaScript was hard. Many online forums had people using brute force algorithm to implement other projects far from sub- string search. However, the *Stack Overflow* [2] online forum for programmers made me have

access to other people handling the same project or other projects with similar aspects.

Secondly, the interface is easy to use as it is simple with not so many items. The text (T) and pattern (P) characters are vertically aligned and the user is able to see the movement from one character to the next. This can be seen as shown below:



Text and Pattern character alignment

Direction of pattern movement

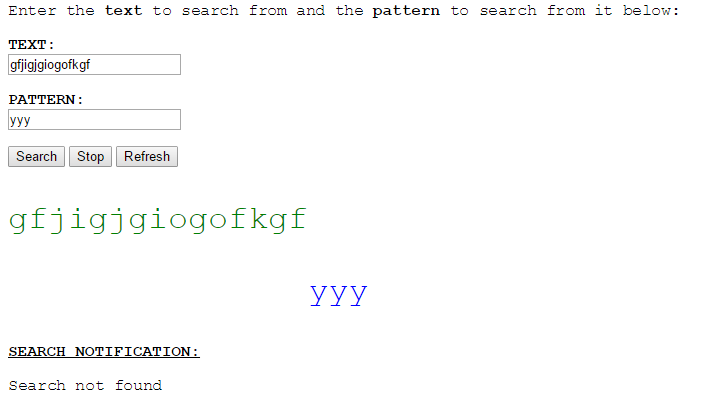
**Fig. 1.3: User interface and character alignment**

With an input like the one shown in Fig. 1.3, the results will the as shown below when the

**Search** button is clicked.



**Fig. 1.4: Search found**



**Fig. 1.5: Search not found**

As can be seen, the sub-string search simulation webpage is able to show by use of simple text and formatting that can be used to enhance the understanding of Document Object Manipulation and be used as a teaching aid during Data Structures coursework.

## 

## CHAPTER FIVE

# DISCUSSION

From the results, Document Object Module manipulation can be used to develop various projects and other systems that can be used to solve particular challenges in the education sector.

The implementation of the steps outlined in the methodology led to the achievement of the results which prove how with the right motivation, student can develop various systems so as to understand disciplines better.

To achieve the above results, brute force search algorithm implements two *for loops* for its comparison between the text (T) and pattern (P) based on the index of the first matching character in the text.

When the first character in the text (T) matches the first character in the pattern (P), the first *for loop* that increments the index of the text character passes control to the inner *for loop* that increments the index of the pattern in line with the remaining pattern characters in comparison to the text characters.

The algorithm then returns the index of the first character of the text (T) where the search is found. In case a match is not found, the algorithm returns -1 as the value. The returned value, which is the index of the text character where the match has occurred or otherwise, is compared with the value returned (integer) by *setTimeout()* method so as to determine when to stop the movement of the pattern (P).

Change in font color is effected using CSS through a *div* where the text (T) characters matched are placed in a *span* to separate them from the rest of the text characters. This allows for the customization of font color.

On the other hand, the project has its own shortcomings when it comes to browser integration. During the development, I exclusively used *Google Chrome* browser to test and carry out implementation of various modules of the project. Further work is required in order to make the sub-string search simulation work well with other browsers other than *Google Chrome* and *Opera* browsers.

# 

# CONCLUSION

In conclusion, the major objectives of this project have been achieved since the prototype developed is able to be used for better understanding of exhaustive search. However, for better understanding, one needs to have the executable file so as to test and see the results as shown above.

# 

# RECOMMENDATIONS AND FUTURE WORK

For this project, I would make the following recommendations:

* + Further work involving use of libraries to make the scripts used in the project universal to all browsers other than *Chrome* and *Opera* browsers.

## Future Work

Since brute force algorithm only returns the first occurrence, it would be better if a simulation project is done on other algorithms that return more than the first occurrence. The two projects than then be compared to each other so as to improve the results.

**REFERENCES AND REFERENCE LIST**

1. Abel Martinez, *String Searching Algorithms (Part I),* 2015. Retrieved on 9th November 2015 at 3:03pm from [www.java.net](http://www.java.net/)
2. Alex Shesterov, *Brute Force String Matching Algorithm*, posted on 19th November 2014

at 12:22. Retrieved on 8th October 2015 at 10:43am from [www.stackoverflow.com](http://www.stackoverflow.com/)

1. Guy Mograbi, *How to highlight text in JavaScript,* posted on 8th February 2014 at 09:36. Retrieved on 23rd December 2015 at 1:45pm from [www.stackoverflow.com](http://www.stackoverflow.com/)
2. *JavaScript Timers with setTimeout and setInterval*, 2010. Retrieved on 19th December 2015 at 10:47am from [www.elated.com](http://www.elated.com/)
3. J¨urg Nievergelt, *Exhaustive Search, Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power,* ND. ETH, 8092 Zurich, Switzerland [jn@inf.ethz.ch](mailto:jn@inf.ethz.ch)

**APPENDICES**

1. **Read me**:

The project works best when run using Google Chrome browser.

1. **Source code**

<! --doctype html -->

<html>

<head>

<style type="text/css">

.andika{

font-size: 25pt; color: green;

}

. chizi{

font-size: 25pt; color: blue;

}

p{

font-family: courier new;

}

#inputText span{ color:blue;

}

</style>

<script>

/\* Displays Search Notification

Calls the moveRight () and toTheEnd () functions, depending on the search results. \*/

function rEturn (){

var me;

me = mySearch ();

if (me! = -1) {

moveRight ();

}

else {

toTheEnd ();

}

}

</script>

<script>

//Uses Brute force Algorithm through its application of Exhaustive Search to determine if the PATTERN is contained in the TEXT.

function mySearch (){

var tExt = dIsplay();

var userInput = dIsplay2(); var M = userInput.length; var N = tExt.length;

var i; var j;

for (var i = 0; i <= N - M; ++i) {

var matched = true; for (var j = 0; j < M; ++j) {

if (tExt.charAt(i + j)!= userInput.charAt(j)){

matched = false;

break;

}

}

if (matched){

return i;

}

}

return -1;

}

</script>

<script>

// Captures tExt(TEXT) and returns it to the searching function. function dIsplay(){

var tExt = document.getElementById("tExt").value;

document.getElementById("dem").innerHTML = tExt;

return tExt;

}

</script>

<script>

// Captures userInput(PATTERN) and returns it to the searching function. function dIsplay2(){

var userInput = document.getElementById("userInput").value; document.getElementById("demo").innerHTML= userInput;

return userInput;

}

</script>

<script>

var patObj = null; var internId;

var nIntervId;

//Initialises the Timeout IDs during Window Onload function init(){

patObj = document.getElementById('demo'); patObj.style.position= 'relative'; patObj.style.left = '0px';

}

/\* Moves the PATTERN to where the search has been found in the TEXT

Calls the function that stops the PATTEN and that which changes the matching TEXT color \*/ function moveRight(){

nIntervId = setTimeout( function (){moveRight();},900); var me = mySearch();

var t = me;

if(t == 0 ){

highlight(); stop();

//patObj.style.left = '0px';

}

else {

hop ();

if ( nIntervId == t ){

highlight (); stop ();

}

}

}

//Moves the PATTERN by 20px for every movement/scroll function hop (){

patObj.style.left = parseInt(patObj.style.left) + 20 + 'px';

}

//Scrolls/moves the PATTERN to the end of the TEXT and stops in when the search is NOT FOUND.

function toTheEnd () {

internId = setTimeout( function (){toTheEnd();},900);

var tExt = document.getElementById("tExt").value; var l = tExt.length;

hop();

if ( internId == l){ stopper();

}

}

//Stops the PATTERN at the point where the search has been FOUND. function stop(){

document.getElementById("dem1").innerHTML = ("Search found"); clearTimeout(nIntervId);

}

//Stops the PATTERN at the end of the TEXT when the search is NOT FOUND.

function stopper () {

document.getElementById("dem1").innerHTML = ("Search not found"); clearTimeout(internId);

}

function stopper1(){

//document.getElementById("dem1").innerHTML = ("Search process stopped"); clearTimeout(internId);

clearTimeout(nIntervId);

}

window.onload =init;

</script>

<script>

//Changes the TEXT color when the PATTERN scroll and reaches the point where the search has been FOUND.

function highlight(){

var pattern = document.getElementById("userInput").value; inputText = document.getElementById("inputText");

var innerHTML = inputText.innerHTML; var index = innerHTML.indexOf(text);

if (index >= 0)

{

innerHTML = innerHTML.substring(0, index) + "<span class='highlight'>"

+ innerHTML.substring(index, index+pattern. length)

+ "</span>" + innerHTML.substring(index + pattern. length); inputText.innerHTML = innerHTML;

}

}

</script>

</head>

<body>

<!--

This part contains:

* Buttons used to call various functions.
* Paragraphs displaying the TEXT and PATTERN.
* The DIVs that define various classes used in CSS for text formatting.

-->

<p> Enter the <b>text</b> to search from and the <b>pattern</b> to search from it below:</p>

<p><b> TEXT:</br><input type="text” oninput = " dIsplay ()" id="tExt"> </b></p>

<! -- <button >TEXT</button></br> -->

<p><b>PATTERN:</br><input type="text" oninput =" dIsplay2()" name = "see" id="userInput"> </b></p>

<! -- <button PATTERN</button> -->

<button onclick=" rEturn()">Search</button>

<button onclick=" stopper1()">Stop</button>

<a href = "javascript:history.go(0)"> <button>Refresh</button> </a>

<div id="inputText" class = "highlight">

<p id="dem" class = "andika"></p></div>

<p id="demo" class = "chizi"></p>

<p> <b> <u> SEARCH NOTIFICATION: </u> </b> </p>

<p id="dem1"></p>

</body>

</html>