16 May 2023

Analysis of Algorithms

CSE2246

Homework 1

String-Matching Algorithms

Student ID	Name/Surname
150120079	Omar Sameh BELAL
150120078	Mais SABBAGH
150121921	Leen I. A. SHAQALAIH

1. Introduction

The objective of this report is to compare the performance of three string matching algorithms when applied to HTML files. The analysis focuses on evaluating the efficiency and the effectiveness of these algorithms in matching patterns within HTML text documents. The report is structured into three main sections: Designing and Experiment, Coding and Running, and Illustrating and Analyzing Results.

1.1. Algorithms

1. **Brute Force Algorithm:** The naïve algorithm, and the most straightforward string-matching algorithm. The algorithm starts by aligning the first character of the pattern with the first character of the text and starts checking for a match. If a mismatch occurs, it shifts the pattern by one position and continues comparing until either match is found at the end of the text is reached.

2. **Horspool Algorithm:** It precomputes a table “Bad-Symbol / Shift Table” that determines how far the pattern can be shifted when a mismatch occurs based on the character that caused the mismatch in the text. The algorithm aligns the last character of the pattern with the corresponding character in the text and compares. If a mismatch occurs, the algorithm consults the table to determine the maximum number of positions the pattern can be shifted then it continues comparing until the end of the text.

3. **Boyer Moore Algorithm:** Like the Horspool algorithm, Boyer-Moore precomputes the “Bad Symbol” based on the pattern. However, it also introduces the “Good Suffix” table to handle mismatches that occur due to a mismatched suffix of the pattern. This new table allows the algorithm to skip larger portions of the text when a mismatch occurs. It checks if there is a matching suffix of the pattern already present in the text and shifts the pattern accordingly to align the next potential match.

2. Designing the Experiment

2.1. Selection of HTML Files

In this project a total of six sample files were used. The first 3 samples are HTML files of size 2100000 (2 MB) made by using a function that randomly generates bits. The bits were randomly generated to ensure a uniform distribution of characters, making sure that no specific pattern is favored or repeated. They can also be easily reproduced, allowing us to repeat the performance evaluations of the algorithms and compare results accurately. The other three files are text files where the first two samples are novels and the third is a dictionary. Two novels were chosen because they represent a realistic example of real-world text documents and they have a large number of complex and simple words, providing more accurate performance comparisons. A dictionary was chosen because it has different words with different lengths from short to long words and may also have a significant number of words.

2.2. Selection of Patterns

Since for each file type (text and bits) there are three different samples, a different pattern was chosen from each file to represent best case, average case and worst-case scenarios for both the bit files and text files. The following patterns were chosen for each sample file:

1. **Bits Sample 1 (pattern: text [: 20])**

   This pattern represents the **Best Case** of the algorithms, since it is the very first 20 bits from the sample file, therefore the number of comparisons needed will not be a lot.

2. **Bits Sample 2 (pattern: 00111111101110101110)**

   This pattern represents the **Average Case** of the algorithms, since it is present approximately in the middle of the sample file therefore being neither in the very end of the file nor in the beginning of it.

3. **Bits Sample 3 (pattern: text[len(text) – 20])**

   This pattern represents the **Worst Case** of the algorithms, since it is present at the very end of the file. Considering the size of the file (1 MB),

there will be a lot of comparisons till we reach the end of the file, therefore accurately representing out worst case scenarios.

4. **Text Sample 1 (pattern: bed-clothes)**

   This pattern represents the **Worst Case** of the algorithms. It is present at the end of the text file, therefore requiring a lot of comparisons. It is also present one time in the entire text therefore accurately representing the worst-case scenario.

5. **Text Sample 2 (pattern: captain)**

   This pattern represents the **Average Case** of the algorithms. The pattern represents one of the main concepts/characters in the novel, meaning it is repeated multiple times throughout the entire novel or sample text, making it best fit for the average case scenario.

6. **Text Sample 3 (pattern: DIPLOBLASTIC)**

   This pattern represents the **Best Case** of the algorithms. The sample file is a dictionary, and the pattern is a word with only one occurrence at the very beginning of the file, meaning there won't be a lot of comparisons till a match is encountered and since it is present only once, there won't be any more comparisons in the rest of the file.

2.3 Experiment Setup

- The code begins with a simple menu allowing the user to select a file type (bits/text) and which sample file of that specific file type to run the algorithms on. The algorithms menu is then displayed, allowing the users to either run each algorithm separately on the sample file and its corresponding predefined pattern or run all the algorithms simultaneously.
- Despite giving the user the freedom to choose whichever sample file and algorithm, the patterns for each sample file are predefined in the code to help portray the best, average, and worst-case scenarios for each algorithm as mentioned above.
- If we choose to run an algorithm separately, the code of each algorithm breaks down the pattern into corresponding sub patterns, starting with the first character/bit.

- The algorithm's code runs for each sub pattern, allowing us to clearly see the difference in time and no. of comparisons as the sub pattern grows to become our original, full pattern. This helps clearly display the differences in metric such as number of occurrences, number of comparisons, time intervals, and performance of the algorithm for that specific sub pattern of our predefined pattern.
- This is done for each algorithm if the user picks either option 1, 2, or 3. The fourth option runs all the algorithms only on the full pattern without breaking it down into sub patterns, to simply display the workings of the algorithm when trying to find a match in a file.

3. Coding & Running

In this Section, we discuss the implementation details of the string-matching algorithms, execution of the experiment and outputs that provide details of how the program runs. Our implementation is written in Python programming language.

Each algorithm is calculated in its own functions with "occurrence" and "comparisons" variable declared and initiated to 0 at the beginning of the function. A timer is also used to calculate the running time of each algorithm separately in its own function.

3.1 Algorithm Implementation

- **Brute Force Algorithm:** This algorithm iterates through every single character in the string (either bits or text) until it reaches index (len(text) – len(pattern) + 1). This guarantees that pattern will not pass the string and only compares the pattern to the end of the string. Another loop is used to compare each character of the pattern to the strings. Once a match between the pattern and the string is encountered, we will increment count and comparisons. If the count equals the length of the pattern, this means that all the pattern matches with the string, and we increment the occurrence. We will reset the count variable to check for other matchings if any are found. The timer is set at the beginning of the function, and once all the occurrences are found, we will calculate the time interval. We then call the update_sample_html function to mark the occurrences in the HTML file.

- **Horspool's Algorithm:** This algorithm is implemented by utilizing a table called "Bad-Symbol".

  This table is precomputed at the beginning of the function of the algorithm.

  - **Bad-Symbol Function:** A dictionary is declared at the beginning of the function which will contain the character of the pattern and shift distance. A value equal to the pattern length is assigned to all distinct characters of the pattern then it will be decremented according to the placement of the characters in the pattern. Shift distance is the distance between the last appearance of the character to the end of the pattern.

  - Using a while loop, we initiate a variable named "matched" which is incremented every time a character in the pattern is matched with a character in the text. We also initiated a variable "i" which will be the pointer in the text which is being compared with the character in the text. Using this variable and a loop we check if the variable has a value less than the pattern length or not and if the rightmost character matches with the one in the text if it does "matched" variable is incremented by 1. Every time the loop executes it also increments the "comparisons" variable. When the loop breaks, we check if the "matched" variable has a value equals the pattern length. If it is true, that means that the pattern occurred in the text, and we increment the "occurrence" variable by 1. Otherwise, the pattern is shifted according to the "Bad-Symbol" table. Time function is used to calculate the running time for the algorithm which is going to be printed when the algorithm is executed alongside with # of occurrences and comparisons.

- **Boyer Moore Algorithm:** This algorithm is implemented by combining two tables one of them is mentioned in the Horspool Algorithm which is the "Bad-Symbol" table and a new table called "Good-Suffix", to handle mismatches occurring due to a mismatch suffix of the pattern.

  Both tables are precomputed at the beginning of the algorithm.

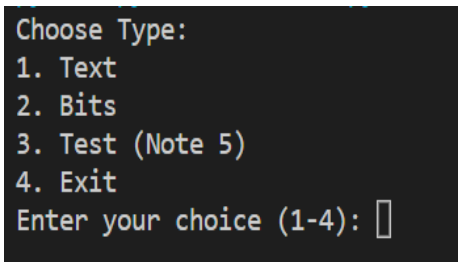
  - **Good Suffix Table:** It allows the algorithm to skip larger portions of the text than the "Bad-Symbol" table. When a mismatch occurs due to a mismatching suffix (which are also prefixes of the pattern), the table stores the maximum shift value that allows the pattern to align with a matching substring in the text. When a mismatch occurs to skip unnecessary comparisons and locate the next

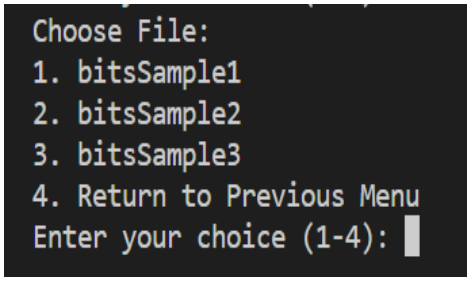
potential match by shifting the pattern based on the information in the table. The table consists of 2 main parts which are Key (Length of suffix matched with text) and Shift value.

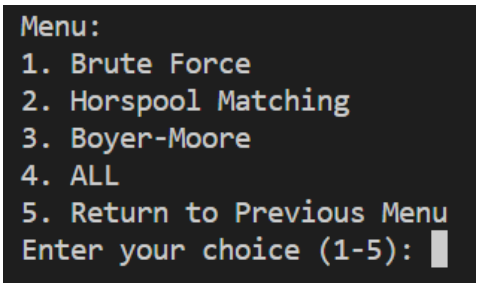
- As done in the beginning of Horspool algorithm, aligning the pattern at the beginning of the text. Using the same loop, we used in Horspool algorithm only this time we check which one of the shift values in both tables are larger, after subtracting the number of matching characters in the suffix of the pattern from the "Bad-Symbol" table value, and then the pattern is shifted according to the larger shift.

3.2 Execution & Outputs

- When running the code, a menu will pop up for the user to choose the type of file then which sample he/she wants to apply the algorithm on. Another menu to choose which algorithm to execute or if the user wants, he can choose to execute them all at the same time. A Test file is also added (Note 5) at the beginning which will appear at the beginning if the user wants to test if the program is running properly or not.







- After either one of the algorithms or all of them is executed, it'll show every detail of the algorithm such as # comparisons, occurrences and running time of the algorithm for that specific sample after wise it is going to create a file named "Updatedhtmlfile" for the sample file the user executed the program for which is going to highlight the pattern found in each sample file.

- Bad Symbol & Good Suffix tables are printed alongside Horspool & Boyer Moore algorithms.

- When executing any algorithm alone it will show how the algorithm is executed for every prefix sub pattern of the pattern alongside its details. However, when choosing option 4 in the menu, it's going to show the results of the pattern specified for every html file for the 3 algorithms with its own details.

```
Brute Force Algorithm with Pattern 01010011001101111011:
occurrence = 3, comparisons = 4202365, time = 440.52779999765335 ms.

Horspool's Algorithm with Pattern 01010011001101111011:
Bad-Symbol Table:
0| 2
1| 1
Horspool occurrence = 3, comparisons = 3149352, time = 662.0374999984051 ms.

Boyer Moore's Algorithm with Pattern 01010011001101111011:
Bad-Symbol Table:
0| 2
1| 1
Good-Suffix Table:
1|1
2|3
3|8
4|20
5|5
6|20
7|20
8|20
9|20
10|20
11|20
12|20
13|20
14|20
15|20
16|20
17|20
18|20
19|20
Boyer Moore's occurence = 3, comparisons = 1026494, time = 233.5910000037984 ms.
```

- When executing the algorithms on the test sentence, we obtain the following outputs:

```
<style>body {word-wrap: break-word;}</style>WHICH_FINALLY_HALTS. _ _ <mark>AT_THAT</mark> POINT
```

WHICH_FINALLY_HALTS. _ _ AT_THAT POINT

```
Brute Force Algorithm with Pattern AT_THAT:
occurrence = 1, comparisons = 43, time = 0.01200000406242907 ms.

Horspool's Algorithm with Pattern AT_THAT:
Bad-Symbol Table:
A| 1
T| 3
_| 4
H| 2
Horspool occurrence = 1, comparisons = 14, time = 0.005099995178170502 ms.

Boyer Moore's Algorithm with Pattern AT_THAT:
Bad-Symbol Table:
A| 1
T| 3
_| 4
H| 2
Good-Suffix Table:
1|3
2|5
3|5
4|5
5|5
6|5
Boyer Moore's occurence = 1, comparisons = 14, time = 0.009399998816661537 ms.
```

4. Illustrating and Analyzing Results

4.1. Complexity Results

For each of the sample files, a table was constructed to give a visual representation of the metrics that were extracted from each algorithm for each of the three cases

1. Brute Force Algorithm Metrics

▪ Bit Sample 1 (Best Case)

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050281	2100000	340.81	6161.791
01	525555	3150279	404.38	7790.393
010	262907	3675833	445.47	8251.584
0101	131901	3938739	513.01	7677.704
01010	65879	4070639	484.92	8394.455
010100	32971	4136515	477.95	8654.702
0101001	16522	4169485	505.52	8247.913
01010011	8273	4186002	483.16	8663.801
010100110	4114	4194274	545.84	7684.072
0101001100	2053	4198382	512.30	8195.163
01010011001	1027	4200434	480.77	8736.889
010100110011	472	4201455	483.73	8685.537
0101001100110	242	4201925	490.47	8567.14
01010011001101	112	4202165	480.67	8742.308
010100110011011	56	4202276	485.00	8664.487
0101001100110111	25	4202328	499.72	8409.365
01010011001101111	12	4202352	506.49	8297.009
010100110011011110	6	4202357	474.66	8853.405
0101001100110111101	4	4202362	512.53	8199.251
01010011001101111011	3	4202365	476.70	8815.534

- **Bit Sample 2 (Average Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050844	2100000	350.91	5984.44
00	525622	3150843	424.43	7423.705
001	262968	3676463	439.69	8361.489
0011	131263	3939430	496.22	7938.878
00111	65708	4070691	485.88	8377.976
001111	32784	4136395	488.14	8473.788
0011111	16417	4169178	463.22	9000.427
00111111	8133	4185593	511.07	8189.862
001111111	4098	4193722	461.69	9083.415
0011111110	2072	4197817	458.48	9155.944
00111111101	1068	4199886	476.66	8811.073
001111111011	526	4200951	466.86	8998.31
0011111110111	265	4201474	469.36	8951.496
00111111101110	132	4201738	501.92	8371.33
001111111011101	67	4201868	484.85	8666.326
0011111110111010	33	4201931	460.41	9126.498
00111111101110101	20	4201963	481.20	8732.259
001111111011101011	13	4201982	458.76	9159.434
0011111110111010111	8	4201993	459.69	9140.928
00111111101110101110	6	4202000	479.37	8765.672

- **Bit Sample 3 (Worst Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1049855	2100000	330.57	6352.664
00	524526	3149854	422.62	7453.159
001	262133	3674379	447.16	8217.146
001	130897	3936510	467.57	8419.082
0010	65451	4067406	461.14	8820.328
001000	32873	4132856	466.19	8865.175
0010000	16437	4165728	462.67	9003.67
00100001	8287	4182164	459.11	9109.285
001000011	4090	4190450	461.19	9086.168
0010000111	2061	4194539	460.39	9110.839
00100001111	1047	4196599	465.35	9018.156
001000011111	548	4197645	468.14	8966.645
0010001111111	270	4198192	464.01	9047.633
00100011111111	142	4198461	468.54	8960.731
001000111111111	63	4198601	472.83	8879.726
0010001111111111	32	4198660	462.62	9075.829
00100011111111111	18	4198689	465.02	9029.05
001000111111111110	10	4198704	476.42	8813.031
0010001111111111101	4	4198713	464.29	9043.298
00100011111111111011	1	4198715	460.45	9118.721

- **Text Sample 1 (Worst Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
B	11242	1026316	276.85	3707.119
BE	3697	1037557	353.55	2934.683
BED	149	1041253	308.02	3380.472
BED-	1	1041401	292.17	3564.367
BED-C	1	1041401	345.71	3012.354
BED-CL	1	1041401	297.80	3496.981
BED-CLO	1	1041401	284.33	3662.649
BED-CLOT	1	1041401	297.29	3502.98
BED-CLOTH	1	1041401	291.69	3570.232
BED-CLOTHE	1	1041401	302.73	3440.032
BED-CLOTHES	1	1041401	304.65	3418.352

- **Text Sample 2 (Average Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
C	22139	1238355	164.401	7532.527
CA	2830	1260493	167.594	7521.111
CAP	284	1263322	167.91	7523.804
CAPT	169	1263605	165.591	7630.88
CAPTA	132	1263773	169.151	7471.271
CAPTAI	132	1263904	156.571	8072.402
CAPTAIN	132	1264035	163.433	7734.27

- **Text Sample 3 (Best Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
D	13524	4192772	533.848	7853.869
DI	2075	4206295	587.761	7156.472
DIP	47	4208369	539.216	7804.607
DIPL	17	4208415	558.792	7531.273
DIPLO	16	4208431	538.528	7814.693
DIPLOB	1	4208446	543.845	7738.319
DIPLOBL	1	4208446	513.771	8191.288
DIPLOBLA	1	4208446	524.409	8025.122
DIPLOBLAS	1	4208446	546.702	7697.879
DIPLOBLAST	1	4208446	533.703	7885.371
DIPLOBLASTI	1	4208446	509.777	8255.465
DIPLOBLASTIC	1	4208446	524.463	8024.295

2. Horspool's Algorithm Metrics

- **Bit Sample 1 (Best Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050281	2100000	702.94	2987.453
01	525555	2099999	577.84	3634.222
010	262907	2625554	686.81	3822.824
0101	131901	2888561	714.48	4042.886
01010	65879	3019726	700.28	4312.169
010100	32971	3083548	742.23	4154.437
0101001	16522	2461721	593.12	4150.46
01010011	8273	3132318	457.59	6845.25
010100110	4114	2486656	368.35	6750.797
0101001100	2053	3146306	449.01	7007.207
01010011001	1027	2492076	367.36	6783.743
010100110011	472	3146735	443.57	7094.111
0101001100110	242	2494180	362.18	6886.576
01010011001101	112	3150329	461.83	6821.404
010100110011011	56	3149313	463.58	6793.462
0101001100110111	25	2488843	353.39	7042.766
01010011001101111	12	2044708	285.46	7162.853
010100110011011110	6	1726855	250.68	6888.683
0101001100110111101	4	3150660	441.16	7141.763
01010011001101111011	3	3149352	449.47	7006.812

Bad Symbol Table (Sample 1)

0	
0	1

01	
0	1
1	2

010	
0	2
1	1

0101	
0	1
1	2

01010	
0	2
1	1

010100	
0	1
1	2

0101001	
0	1
1	3

01010011	
0	2
1	1

0101001100	
0	1
1	2

01010011001	
0	1
1	3

010100110011	
0	2
1	1

0101001100110	
0	3
1	1

01010011001101	
0	1
1	2

0101001100110111	
0	3
1	1

01010011001101111	
0	4
1	1

- **Bit Sample 2 (Average Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050844	2100000	427.29	4914.695
00	525622	2100675	349.63	6008.28
001	262968	1968884	312.16	6307.291
0011	131263	2885072	428.41	6734.371
00111	65708	2359345	370.24	6372.475
001111	32784	1980787	274.35	7219.927
0011111	16417	1691304	247.64	6829.688
00111111	8133	1470733	199.30	7379.493
001111111	4098	1298507	181.43	7157.069
0011111110	2072	1160523	157.10	7387.161
00111111101	1068	3146832	452.52	6954.018
001111111011	526	3145080	503.81	6242.591
0011111110111	265	2489127	395.89	6287.421
00111111101110	132	2046733	277.17	7384.396
001111111011101	67	3148730	453.96	6936.14
0011111110111010	33	3150287	449.61	7006.71
00111111101110101	20	3150244	452.22	6966.176
001111111011101011	13	3147192	469.78	6699.289
0011111110111010111	8	2489903	353.71	7039.391
00111111101110101110	6	2047314	281.82	7264.616

Bad Symbol Table (Sample 2)

00111111101	
0	1
1	2

001111111011	
0	2
1	1

0011111110111	
0	3
1	1

00111111101110	
0	4
1	1

001111111011101	
0	1
1	2

0011111110111010	
0	2
1	1

00111111101110101	
0	1
1	2

001111111011101011	
0	2
1	1

001111111	
0	7
1	1

0011111110	
0	8
1	1

0	
0	1

00	
0	1

001	
0	1
1	3

0011	
0	2
1	1

00111	
0	3
1	1

001111	
0	4
1	1

0011111	
0	5
1	1

00111111	
0	6
1	1

00111111101110101111	
0	3
1	1

001111111011101011110	
0	4
1	1

- **Bit Sample 3 (Worst Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1049855	2100000	444.82	4721.011
00	524526	2099683	341.55	6147.513
001	262133	1968959	304.98	6456.027
001	130897	2887776	433.55	6660.768
0010	65451	3017251	441.78	6829.759
001000	32873	2427675	349.24	6951.309
0010000	16437	2011441	279.05	7208.174
00100001	8287	1711619	240.92	7104.512
001000011	4090	3140893	442.46	7098.705
0010000111	2061	2489808	359.21	6931.344
00100001111	1047	2040125	278.66	7321.198
001000011111	548	1724050	240.61	7165.33
0010001111111	270	1485960	208.78	7117.348
00100011111111	142	1305386	179.56	7269.915
001000111111111	63	1161092	175.72	6607.626
0010001111111111	32	1046113	142.76	7327.774
00100011111111111	18	949301	133.70	7100.232
001000111111111110	10	874797	120.36	7268.17
001000111111111101	4	3151364	449.96	7003.654
0010001111111111011	1	3150192	481.64	6540.553

Bad Symbol Table (Sample 3)

0	
0	1

00	
0	1

001	
0	1
1	3

0010	
0	2
1	1

00100	
0	1
1	2

001000	
0	1
1	3

0010000	
0	1
1	4

00100001	
0	1
1	5

001000011	
0	2
1	1

0010000111	
0	3
1	1

00100001111	
0	4
1	1

001000011111	
0	5
1	1

0010000111111	
0	6
1	1

00100001111111	
0	7
1	1

001000011111111	
0	8
1	2

0010000111111111	
0	9
1	1

00100001111111111	
0	10
1	1

001000011111111110	
0	11
1	1

001000011111111111101	
0	1
1	2

00100001111111111011	
0	2
1	1

- **Text Sample 1 (Worst Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
B	11242	1026316	369.55	2777.205
BE	3697	565472	228.86	2470.821
BED	149	385853	141.59	2725.143
BED-	1	277639	147.39	1883.703
BED-C	1	221886	142.30	1559.283
BED-CL	1	188925	78.67	2401.487
BED-CLO	1	171581	66.06	2597.351
BED-CLOT	1	156099	72.10	2165.035
BED-CLOTH	1	150425	58.43	2574.448
BED-CLOTHE	1	153149	58.84	2602.804
BED-CLOTHES	1	127873	78.15	1636.251

Bad Symbol Table (Sample 1)

b	
b	1

be	
b	1
e	2

bed	
b	2
e	1
d	3

bed-	
b	3
e	2
d	1
-	4

bed-c	
b	4
e	3
d	2
-	1
c	5

bed-cl	
b	5
e	4
d	3
-	2
c	1
l	6

bed-clo	
b	6
e	5
d	4
-	3
c	2
l	1
o	7

bed-clot	
b	7
e	6
d	5
-	4
c	3
l	2
o	1
t	8

bed-cloth	
b	8
e	7
d	6
-	5
c	4
l	3
o	2
t	1
h	9

bed-clothe	
b	9
e	8
d	7
-	6
c	5
l	4
o	3
t	2
h	1

bed-clothes	
b	10
e	1
d	8
-	7
c	6
l	5
o	4
t	3
h	2
s	11

- **Text Sample 2 (Average Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
C	22139	1238355	350.72	3530.894
CA	2830	664019	189.79	3498.704
CAP	284	439524	102.78	4276.357
CAPT	169	348243	97.05	3588.284
CAPTA	132	290247	78.21	3711.124
CAPTAI	132	245814	68.26	3601.143
CAPTAIN	132	226894	64.79	3501.991

Bad Symbol Table (Sample 2)

c	
c	1

ca	
c	1
a	2

cap	
c	2
a	1
p	3

capt	
c	3
a	2
p	1
t	4

capta	
c	4
a	3
p	2
t	1

captai	
c	5
a	1
p	3
t	2
i	6

captain	
c	6
a	2
p	4
t	3
i	1
n	7

- **Text Sample 3 (Best Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
D	13524	4192772	744.33	5632.948
DI	2075	2118494	374.69	5653.991
DIP	47	1412818	258.39	5467.774
DIPL	17	1063459	192.66	5519.874
DIPLO	16	856972	175.77	4875.531
DIPLOB	1	712685	139.37	5113.618
DIPLOBL	1	613254	119	5153.395
DIPLOBLA	1	541155	104.56	5175.545
DIPLOBLAS	1	483176	90.57	5334.835
DIPLOBLAST	1	439491	86.22	5097.321
DIPLOBLASTI	1	402713	76.65	5253.92
DIPLOBLASTIC	1	369798	72.69	5087.33

Bad Symbol Table (Sample 3)

D	
D	1

DI	
D	1
I	2

DIP	
D	2
I	1
P	3

DIPL	
D	3
I	2
P	1
L	4

DIPLO	
D	4
I	3
P	2
L	1
O	5

DIPLOB	
D	5
I	4
P	3
L	2
O	1
B	6

DIPLOBL	
D	6
I	5
P	4
L	3
O	2
B	1

DIPLOBLA	
D	7
I	6
P	5
L	1
O	3
B	2
A	8

DIPLOBLAS	
D	8
I	7
P	6
L	2
O	4
B	3
A	1
S	9

DIPLOBLAST	
D	9
I	8
P	7
L	3
O	5
B	4
A	2
S	1
T	10

DIPLOBLASTI	
D	10
I	9
P	8
L	4
O	6
B	5
A	3
S	2
T	1

DIPLOBLASTIC	
D	11
I	1
P	9
L	5
O	7
B	6
A	4
S	3
T	2
C	12

3. Boyer Moore's Algorithm Metrics

- **Bit Sample 1 (Best Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050281	2100000	415.4	5055.368
01	525555	2625554	450.09	5833.398
010	262907	2888461	474.787	6083.699
0101	131901	2482290	387.33	6408.721
01010	65879	2008751	313	6417.735
010100	32971	1840050	291.76	6306.725
0101001	16522	1661140	276.96	5997.761
01010011	8273	1396759	223.36	6253.398
010100110	4114	1484722	235.6	6301.876
0101001100	2053	1479854	234.019	6323.649
01010011001	1027	1299005	192.119	6761.46
010100110011	472	1104391	167.96	6575.322
0101001100110	242	1033497	155.055	6665.357
01010011001101	112	1533725	270.994	5659.627
010100110011011	56	1202832	202.47	5940.791
0101001100110111	25	962555	145.125	6632.593
01010011001101111	12	959445	142.729	6722.145
010100110011011110	6	827885	122.033	6784.108
0101001100110111101	4	1044553	177.45	5886.464
01010011001101111011	3	1026494	219.547	4675.509

- **Good Suffix Table (Sample 1)**

K	MATCHING SUFFIX	SHIFT DISTANCE
1	1	1
2	11	3
3	011	8
4	1011	20
5	11011	5
6	111011	20
7	1111011	20
8	01111011	20
9	101111011	20
10	1101111011	20
11	01101111011	20
12	001101111011	20
13	1001101111011	20
14	11001101111011	20
15	011001101111011	20
16	0011001101111011	20
17	10011001101111011	20
18	010011001101111011	20
19	1010011001101111011	20

- **Bit Sample 2 (Average Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1050844	2100000	427.834	4908.446
00	525622	2100675	356.609	5890.695
001	262968	2396826	385.94	6210.359
0011	131263	2075176	346.895	5982.144
00111	65708	1671104	267.395	6249.571
001111	32784	1324958	199.689	6635.108
0011111	16417	1057834	164.84	6417.338
00111111	8133	862378	134.19	6426.544
001111111	4098	719581	111.767	6438.224
0011111110	2072	1076576	155.983	6901.88
00111111101	1068	1136109	169.249	6712.648
001111111011	526	1228664	197.103	6233.614
0011111110111	265	1223449	183.857	6654.351
00111111101110	132	841966	126.761	6642.153
001111111011101	67	886202	135.204	6554.555
0011111110111010	33	955877	148.628	6431.339
00111111101110101	20	811297	129.78	6251.325
001111111011101011	13	944964	141	6701.872
0011111110111010111	8	1053000	166.796	6313.101
00111111101110101110	6	724265	106.72	6786.591

- **Good Suffix Table (Sample 2)**

K	MATCHING SUFFIX	SHIFT DISTANCE
1	0	18
2	10	4
3	110	19
4	1110	10
5	01110	19
6	101110	6
7	0101110	19
8	10101110	19
9	110101110	19
10	1110101110	19
11	01110101110	19
12	101110101110	19
13	1101110101110	19
14	11101110101110	19
15	111101110101110	19
16	1111101110101110	19
17	11111101110101110	19
18	111111101110101110	19
19	0111111101110101110	19

- **Bit Sample 3 (Worst Case)**

PATTERNS	OCCURRENCES	# OF COMPARISONS	TIME INTERVAL	PERFORMANCE
0	1049855	2100000	414.649	5064.524
00	524526	2099683	350.63	5988.315
001	262133	2394531	396.478	6039.505
001	130897	2545369	434.882	5853.011
0010	65451	2296286	372.597	6162.921
001000	32873	1743511	286.277	6090.294
0010000	16437	1342332	217.25	6178.743
00100001	8287	1376098	199.453	6899.36
001000011	4090	1294467	189.087	6845.88
0010000111	2061	1256880	182.809	6875.373
00100001111	1047	1121995	204.106	5497.119
001000011111	548	961580	152.136	6320.529
0010001111111	270	815473	132.352	6161.395
00100011111111	142	694674	106.07	6549.203
001000111111111	63	601675	93.234	6453.386
0010001111111111	32	526159	81.429	6461.568
00100011111111111	18	469282	78.243	5997.751
001000111111111110	10	665019	100.806	6597.018
001000111111111101	4	785963	116.545	6743.859
0010001111111111011	1	905352	139.408	6494.261

- **Good Suffix Table (Sample 3)**

K	MATCHING SUFFIX	SHIFT DISTANCE
1	1	1
2	11	3
3	011	11
4	1011	20
5	11011	20
6	111011	20
7	1111011	20
8	11111011	20
9	111111011	20
10	1111111011	20
11	11111111011	20
12	111111111011	20
13	1111111111011	20
14	01111111111011	20
15	001111111111011	20
16	0001111111111011	20
17	00001111111111011	20
18	100001111111111011	20
19	0100001111111111011	20

- **Text Sample 1 (Worst Case)**

Patterns	Occurrences	# of Comparisons	Time Interval	Performance
b	11242	1026316	369.55	2777.205
be	3697	567898	228.86	2470.821
bed	149	385942	141.59	2725.143
bed-	1	277641	147.39	1883.703
bed-c	1	221886	142.30	1559.283
bed-cl	1	188926	78.67	2401.487
bed-clo	1	171581	66.06	2597.351
bed-clot	1	156099	72.10	2165.035
bed-cloth	1	150432	58.43	2574.448
bed-clothe	1	151635	58.84	2602.804
bed-clothes	1	127873	78.15	1636.251

- **Good Suffix Table (Sample 1)**

K	Matching suffix	Shift Distance
1	s	11
2	es	11
3	hes	11
4	thes	11
5	othes	11
6	lothes	11
7	clothes	11
8	-clothes	11
9	d-clothes	11
10	ed-clothes	11

- **Text Sample 2 (Average Case)**

Patterns	Occurrences	# of Comparisons	Time Interval	Performance
c	22139	1238355	341.75	3623.57
ca	2830	665579	181.73	3662.461
cap	284	439705	131.76	3337.166
capt	169	348443	99.76	3492.813
capta	132	289886	84.71	3422.099
captai	132	245926	71.26	3451.109
captain	132	227015	66.3	3424.057

- **Good Suffix Table (Sample 2)**

K	Matching suffix	Shift Distance
1	n	7
2	in	7
3	ain	7
4	tain	7
5	ptain	7
6	aptain	7

- **Text Sample 3 (Best Case)**

Patterns	Occurrences	# of Comparisons	Time Interval	Performance
D	13524	4192772	792	5293.904
DI	2075	2119571	393.36	5388.375
DIP	47	1412842	268.22	5267.474
DIPL	17	1063469	197.85	5375.128
DIPLO	16	856976	159.36	5377.61
DIPLOB	1	712686	140.1	5086.981
DIPLOBL	1	613089	118.17	5188.195
DIPLOBLA	1	541155	110.37	4903.099
DIPLOBLAS	1	483177	94.22	5128.179
DIPLOBLAST	1	439492	88.79	4949.792
DIPLOBLASTI	1	402484	78.48	5128.491
DIPLOBLASTIC	1	369800	74.51	4963.092

- **Good Suffix Table (Sample 3)**

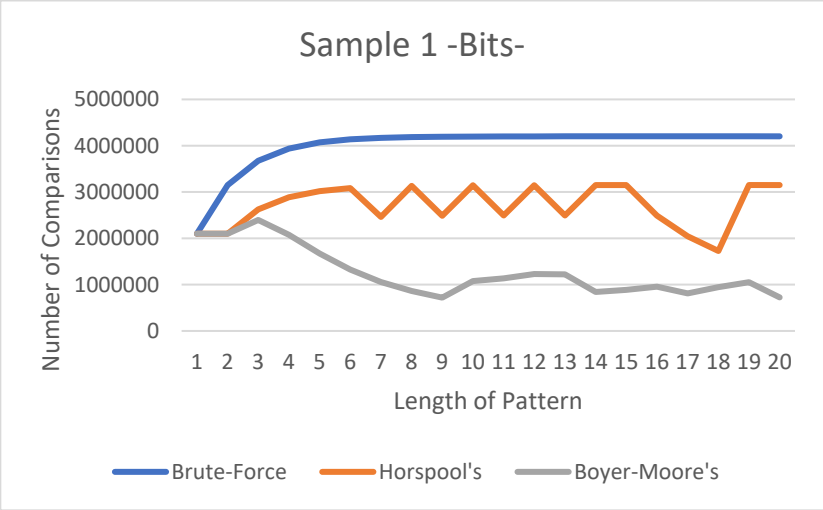
K	Matching suffix	Shift Distance
1	C	12
2	IC	12
3	TIC	12
4	STIC	12
5	ASTIC	12
6	LASTIC	12
7	BLASTIC	12
8	OBLASTIC	12
9	LOBLASTIC	12
10	PLOBLASTIC	12
11	IPLOBLASTIC	12

4.2 Performance Comparisons

According to the information stored in the above tables, we plotted a set of graphs to visually display the performance for each algorithm for each of the three samples.

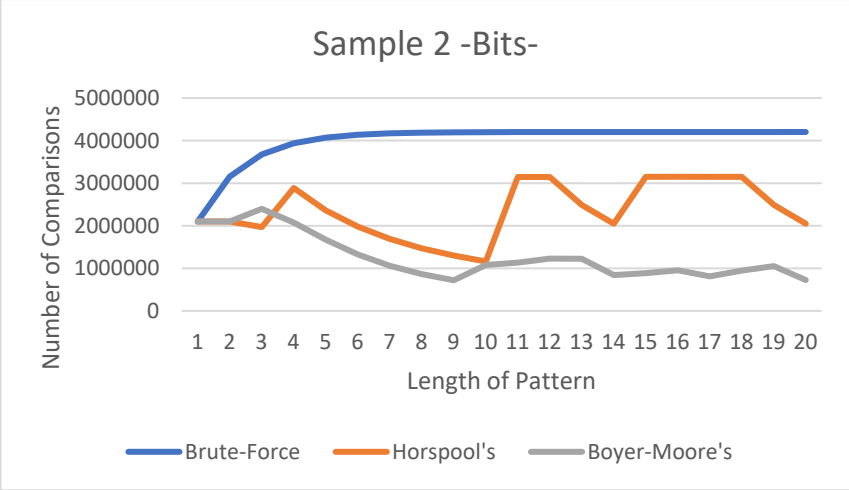
1. Bit Sample 1 Results

We can observe that as the length of the pattern increases, the number of comparisons for the brute force algorithm rises then remains constant while for Horspool's algorithm, it alternates between increasing and decreasing with the steepest decrease happening when the number of comparisons is 1726855. For Boyer Moore's the number of comparisons starts at 2100000 and continues to decrease after that.



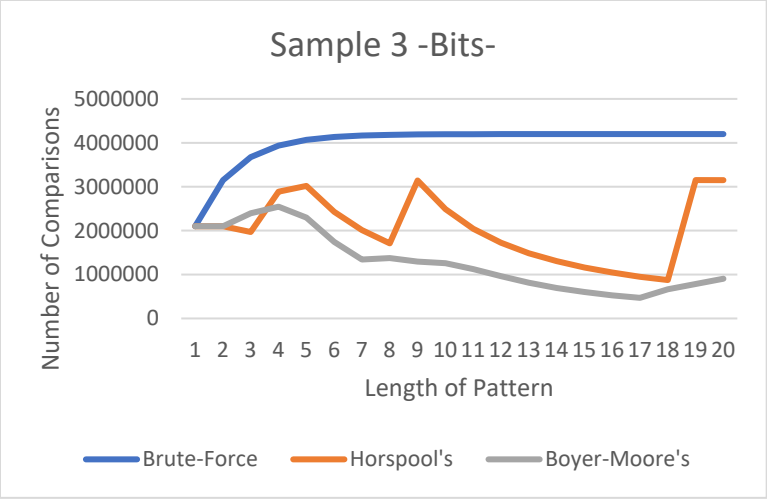
2. Bit Sample 2 Results

We can observe that as the length of the pattern increases, the number of comparisons for the brute force algorithm rises then remains constant while for Horspool's algorithm, it alternates between increasing and decreasing with the steepest decrease happening when the number of comparisons is about 1000000. For Boyer Moore's, the number of comparisons starts at 2100000 and continues to decrease after that, with occasional slight increase.



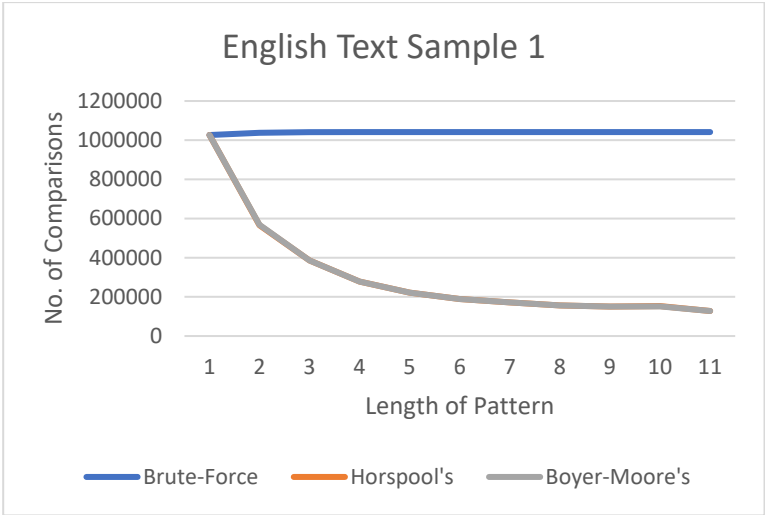
3. Bit Sample 3 Results

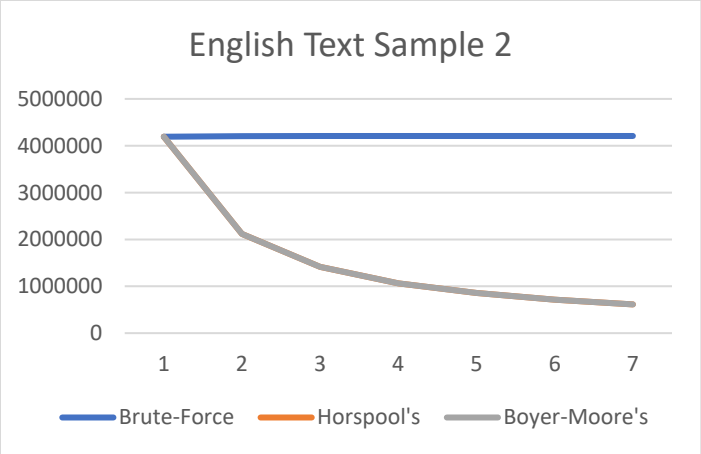
We can observe that as the length of the pattern increases, the number of comparisons for the brute force algorithm rises then remains constant while for Horspool's algorithm, it alternates between increasing and decreasing with the steepest decrease happening when the number of comparisons is 874797. For Boyer Moore's the number of comparisons starts at 2100000 and continues to decrease after that, with occasional slight increase.

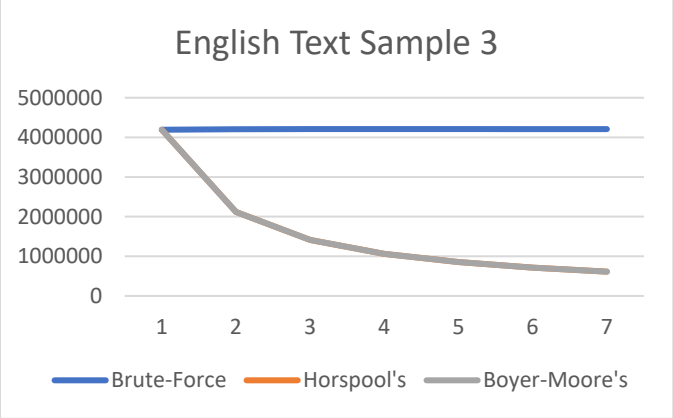


4. Text Samples 1, 2 and 3 Results

We can observe in the three graphs that in the brute force algorithm, as the length of the pattern increases, the number of comparisons stays almost constant, while for both Boyer Moore's and Horspool's as the length of the pattern increases, the number of comparisons decreases. The lines for both Boyer Moore's and Horspool's overlap. This indicates that their results are almost equal.




English Text Sample 2
5000000
4000000
3000000
2000000
1000000
0
1
2
3
4
5
6
7
Brute-Force
Horspool's
Boyer-Moore's


English Text Sample 3
5000000
4000000
3000000
2000000
1000000
0
1
2
3
4
5
6
7
Brute-Force
Horspool's
Boyer-Moore's

5. Conclusion

In this study, we conducted a comparative analysis of three string-matching algorithms (Brute Force, Horspool, and Boyer-Moore) with the goal of evaluating their performance when applied to HTML files. Through the process of designing and experimenting, coding and running, and illustrating and analyzing results, we gained valuable insights into the efficiency and effectiveness of these algorithms.

The experimental results revealed that the Brute Force algorithm, while simple to understand and implement, exhibited the highest time complexity, requiring O(m * n) comparisons for pattern matching in HTML files. As a result, its performance was less favorable, especially for larger patterns or texts.

The Horspool's algorithm, utilizing the "bad character shift rule," showed improvements over the Brute Force algorithm. By precomputing the bad character shift table, it reduced the number of comparisons and demonstrated an average-case time complexity of O(n). However, for patterns with many repeating characters, its worst-case behavior approached O(m * n).

Among the three algorithms, the Boyer-Moore algorithm stood out as the most efficient and powerful. By incorporating both the "bad character shift rule" and the "good suffix shift rule," it exhibited remarkable performance gains. The precomputed bad character shift table allowed it to skip unnecessary comparisons efficiently, and the good suffix table enabled the algorithm to locate the next potential match by aligning the pattern with matching substrings in the text. With an average-case time complexity of O(n/m) and O(n * m) for the worst-case scenario, Boyer-Moore demonstrated superior performance, especially for longer patterns.

The comparative analysis also highlighted the importance of selecting appropriate algorithms based on the specific characteristics of the problem domain. For HTML files, where patterns and texts can vary in size and complexity, the choice of an efficient algorithm such as Boyer-Moore can significantly impact the matching process.

In conclusion, our findings emphasize the significance of evaluating and selecting appropriate string-matching algorithms when working with HTML files. The Boyer-Moore algorithm, with its efficient search strategies, proved to be the most effective for matching patterns within HTML documents. However, it is important to consider the trade-offs between algorithm complexity and specific requirements of the problem domain.

6. Division of Labor

Name/Surname	Tasks
Omar Sameh BELAL	Boyer Moore, good suffix table, menu
Mais SABBAGH	Brute Force, marking function
Leen I. A. SHAQALAIH	Horspool's , Bad Symbol Table, menu

References

Navarro, G., & Raffinot, M. (2002). Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences. Cambridge University Press.

introduction to design and analysis of algorithms 3rd edition

https://www.gutenberg.org/browse/scores/top