King Abdullah University of Science and Technology
Division of Mathematical and Computer Sciences and Engineering
Design and Analysis of Algorithms

Midterm Project Report

Diverse Approaches to Exact Pattern Matching

Affara, Lama
Almansour, Durrah
Al-Shedivat, Maruan
Chen, Gui
Fujii, Chisato
Rapakoulia, Trisevgeni

October 29, 2013

1 Introduction

Write introduction here

2 Boyre Moore

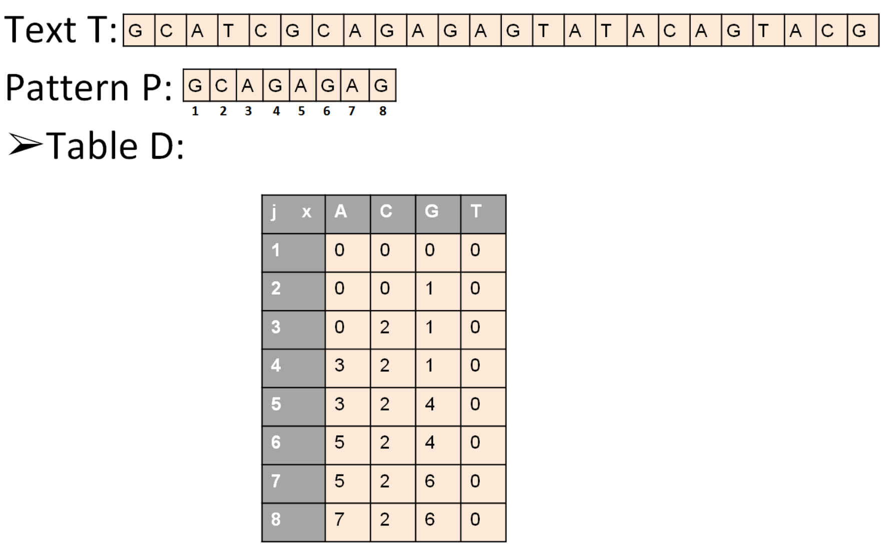
Boyre Moore algorithm [1] searches for all the occurrences of the pattern in the text. It is in some way similar to the naive search algorithm. Initially, it aligns the first character in P with the first character in T. The algorithm then compares characters between P and T sequentially from right to left. Once a mismatch occurs, a shift rule is applied thus moving the pattern by $s \geq 1$. The algorithm is basically divided into two stages: preprocessing and searching. There are two different preprocessing approaches in the literature: Bad Character Rule and Good Suffix Tree [2]. We decided to choose the Bad Character rule due to its simplicity and applicability to our dataset. In the following sections, we describe the two stages of the algorithm.

2.1 Preprocessing Stage

In the preprocessing stage, the algorithm makes use of the alphabet Σ and the pattern P. A two dimensional table D is constructed by processing the pattern according to the available alphabet. D is of size $k \times |\Sigma|$ where for each mismatch index in P, the position of rightmost occurrence of a character in Σ is stored. Figure 2.1 shows an example of the table stored by processing the pattern GCAGAGAG based on the DNA alphabet={A,C,G,T}. Starting from the last row corresponding to a mismatch occurring at position i=k in P, i=8 for this example, the algorithm scans P to find the rightmost index of the given character. In the below example, the last occurrence of A before position 8 is 7, G is 6, C is 2, and T is 0. It is important to note here that if a character does not exist in the pattern, its position in the table is always 0. Now, the algorithm iterates from i=k to 1. If the mismatch occurs in position i-1, the algorithm updates only the value for the specific character placed in this position, A for this example, and all the other values remain the same as D[i,x]. The last occurrence of A before position 7 is 5, while G, C, and T stay the same.

2.2 Searching Phase

In the searching phase, the algorithm needs shift the pattern and sequentially match it with the aligned text. Starting from the rightmost character in P, the algorithm checks the aligned character in T. If the pair of characters are matching, it sequentially continues the check to the next left character. If a mismatch occurs at position j in P, the algorithm needs to shift P according to the mismatched character in the text. For example, if at position j, the text contains a character that is not found in P, the pattern should be shifted by j. However, if the mismatched character is found in P, the pattern should be shifted by j-i, where i corresponds to the rightmost occurrence of this character in P. The index i of the last occurrence is retrieved from table D and i=D[j,x] where x is the character in the text. Figure 2.2 shows the searching phase for the example shown in the previous section.



Text T: G C A T C G C A G A G A G T A T A C A G T A C G

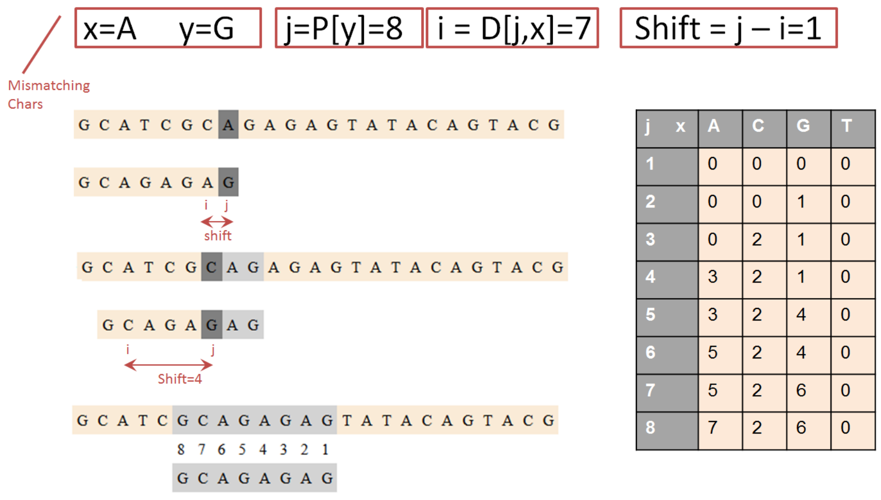
Pattern P: G C A G A G A G

1	2	3	4	5	6	7	8
G	C	A	G	A	G	A	G

➢Table D:

j x	A	C	G	T
1	0	0	0	0
2	0	0	1	0
3	0	2	1	0
4	3	2	1	0
5	3	2	4	0
6	5	2	4	0
7	5	2	6	0
8	7	2	6	0

Figure 2.1: Table of preprocessing phase



x=A y=G
j=P[y]=8
i = D[j,x]=7
Shift = j – i=1
Mismatching Chars
G C A T C G C A G A G A G T A T A C A G T A C G
G C A G A G A G
i j
shift
G C A T C G C A G A G A G T A T A C A G T A C G
G C A G A G A G
i j
Shift=4
G C A T C G C A G A G A G T A T A C A G T A C G
8 7 6 5 4 3 2 1
G C A G A G A G

j x	A	C	G	T
1	0	0	0	0
2	0	0	1	0
3	0	2	1	0
4	3	2	1	0
5	3	2	4	0
6	5	2	4	0
7	5	2	6	0
8	7	2	6	0

Figure 2.2: Searching Phase

2.3 COMPLEXITY

References

[1] Robert S Boyer and J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[2] Richard Cole. Tight bounds on the complexity of the boyer–moore string matching algorithm. *SIAM J. Comput.*, 23(5):1075–1091, October 1994.