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 [2] 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 [3]. 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.

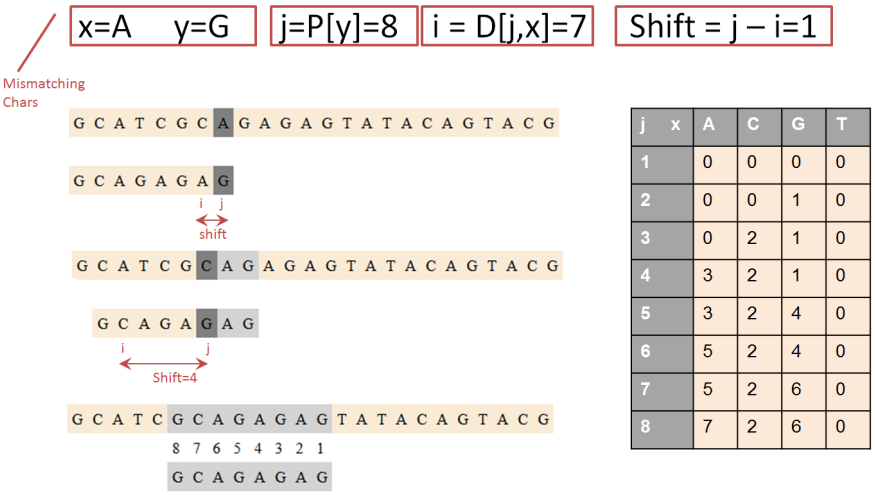
Text T: G C A T C G C A G A G A G T A T A C A G T A C G

Pattern P: G C A G A G A G
1 2 3 4 5 6 7 8

➢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



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

3 AHO-CORASICK ALGORITHM

Aho-Corasick algorithm [1] is a generalization of the Knuth-Morris-Pratt algorithm. It takes one or more patterns (a dictionary) to be searched in the text. Suppose, the total length of all patterns is k. Then, the algorithm pre-processes the patterns and constructs a Deterministic Finite Automaton (DFA) [4] in O(k) time. The obtained DFA processes a text and reports pattern occurrences in linear time too – O(n + z), where z is the number of occurrences to be

found. The O(z) term is for that we assume that we can report all the occurrences in linear time. The example of the DFA is given in the figure 3.1.

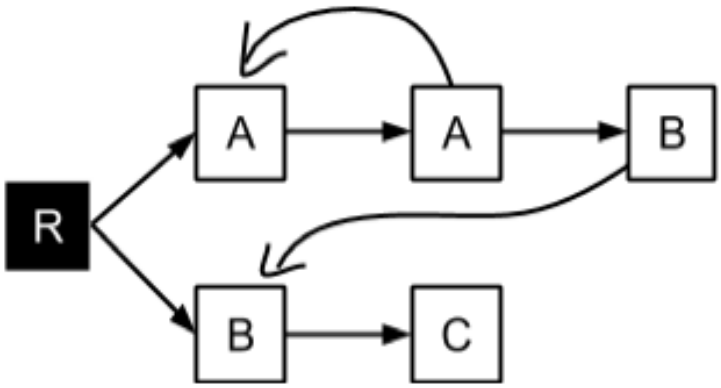


Figure 3.1: A DFA representing a set of patterns AAB, BC

3.1 Construction of DFA

Let the index of the first character in a pattern be 1, and let i be an integer between 0 and the length of the longest pattern.
We build a DFA of nodes and edges, where each unique character at position i of all the patterns is represented by a node at level i. a dictionary node is a special kind of node which marks the end of a pattern. This dictionary node is marked at the time of building DFA.

1. Forward Edge: Connecting a node representing character at position i in pattern p to the one representing the character at position i+1 in the same pattern.

2. Reverse Edge: Connecting a node to another node representing the same value in a higher level (closer to the root) provided they have the same value parents, which is insured during the building.

3.1.1 Setting Up the Nodes and Forward Edges

1. First a root node is created and set to be the current node. The first pattern is selecter and a pointer is set to point to the first character in it.

2. If the current node does not have a forward edge to a node representing the current character, then such a node is created and a forward edge connects the current node to it. The new node becomes the the current node.

3. If the current node has a forward edge to a node representing the current character, then that node becomes the current node.

4. The pointer in the pattern moves to the next character
5. If there are no more characters in the pattern, mark the node as dictionary node with the corresponding pattern, then select the next pattern in the list and make the pointer point to the first character in it. Set the current node to the root and repeat steps 2 through 5 until there are no more patterns in the list.

3.1.2 SETTING UP THE REVERSE EDGES

Expanding a node means dequeuing it and enqueuing its child nodes.

1. Start from the node and expand it into its children, put the child nodes into a queue.
2. Point the reverse edges of the children of the root back to the root.
3. Expand the node at the front of the queue.
4. For each node x of the children of the newly expanded node do the following:
   a) Set the parent of the newly expanded node as the current node.
   b) If the current node has a child node with the same value as node x, point x's reverse edge to that child. If it does nott, check that it is the root, and if so, point x's reverse edge to it. Otherwise go to the node pointed to by the reverse edge of the current node set it as the current node and repeat 4.b.

3.1.3 SEARCHING

It keeps an array of found patterns. The first character of the text is pass to DFA where current node is the root. If this character finds a forward edge meaning the character matches the node pointed by the forward edge, then the current node of DFA becomes the node that the forward edge points to. If this character does not finds a forward edge, the current node reminds the same. Either case, the second character of the text is then passed to the current node of DFA. If this second character of the text does not have a forward edge, it takes a reverse edge. When the current node of DFA reaches the dictionary node, it increments the occurrences array. It repeats this process until the end of the text has been reached.

3.2 THEORETICAL ANALYSIS

k = total length of all patterns.
n = number of characters in input string
z = pattern occurrences in the text

3.2.1 PREPROCESSING

A Deterministic Finite Automaton (DFA) is built according to the patterns as described. Setting up nodes and creating forward edges requires k time, and traversing the DFA to add the reverse edges take a maximum of k time. This yields a runtime of 2*k = O(k).

3.2.2 SEARCHING

The text is processed in a single pass. It is true that for a text of length n, minimum of n computations are required for comparing to DFA nodes. In addition, when a dictionary node was reached, this pattern would be inserted into array of found patterns. Thus the search runtime complexity is O(n+z) where z is occurrences of the matched patterns.

REFERENCES

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

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

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

[4] John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.