Optimal Shift-Or String Matching Algorithm for Multiple Patterns

Rajesh Prasad, Suneeta Agarwal

ABSTRACT

In this paper, we develop a new algorithm for handling multiple patterns, which is based on average optimal shift-or algorithm. We have assumed that the pattern representation fits into a single computer word and length of each pattern is equal. We have adopted the concept of classes of characters for handling multiple patterns. We compare the performance of the proposed algorithm with the standard shift-or algorithm. The experimental results show that our algorithm is the faster in most of the cases.

KEY WORDS

String Matching, finite automata, shift-or, multiple patterns

1. Introduction

The problem of searching the occurrences of a pattern P[0...m-1] in the text T[0…n-1] with m≤n, where the symbols of P and T are drawn from some alphabet Σ of size σ, is called exact string matching problem. Numerous efficient algorithms for solving this problem exist. The first linear time algorithm was given in (Knuth et al., 1977), and the first sub linear expected time algorithm in (Boyer and Moore, 1977). In multi-pattern string matching problem, the set of patterns P_1, P_2...P_r, with r>1, each of length m on the same alphabet, is searched simultaneously in the text T [0…m-1].

We extend well-known average optimal shift-or algorithm [1] of single pattern to work with multiple patterns. The average optimal shift-or algorithm is based on standard shift-or algorithm, which uses bit parallel approach to simulate the non-deterministic finite automaton efficiently.

Experimental results show that our algorithm is faster in the majority of the cases.

Manuscript received April 11, 2007. This work was partially supported in part by the Ewing Christian College Society, Allhabad, India-211004

Rajesh Prasad is research scholar at Motilal Nehru National Institute of Technology - Deemed University, Allahabad,India,211004,(email:rajesh_ucer@yahoo.com)

Suneeta Agarwal is with the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology- Deemed University, Allahabad, 211004, India (email: suneeta@mnnit.ac.in).

2. Multiple Patterns

In multiple pattern string-matching problems, there are r patterns P_1, P_2...P_r, r>1, each of length m. The objective is to search all occurrences of each pattern in a given text T[0…n-1]. We further assume that m≤n. The basic idea in handling multiple patterns using concept of **classes of characters** [2] is that each text character is allowed to match to the characters at any position j in any of the patterns. For example, if we have patterns Rajesh. and Prasad then we form a super pattern {R, P}, {a, r}, {j, a}, {e, s}, {s, a}, {h, d}. This matches, for example, the string Raassh, Paaeah etc. in the given text string. Therefore, this is a filter and potential matches require further verification.

3. Shift-Or

It assumes that a machine word has w bits, numbered from the least significant bit to the most significant bit. We use C language-like notation for the bit wise operations of words; & is bitwise AND, | is bit-wise OR, ^ is bit-wise XOR, ~ negates all bits, << is shift to left, and >> shift to right. For brevity, we make the assumption that m≤w, unless explicitly stated otherwise.

3.1 Standard Shift-Or

We discuss the standard shift-or algorithm for single and multiple patterns

3.1.1 Standard Shift-or for Single Patterns

The standard shift-Or [6] automaton is constructed as follows:

The automaton has states 0…m. The state 0 is starting state and state m is the only accepting state. For i=0...m-1, there is a transition from state i to state i+1 for character P [i]. In addition, there is a transition for every cεΣ from and to the initial state, which makes the automaton nondeterministic. The preprocessing algorithm builds a table B, having one bit-mask entry for each cεΣ . For 0≤ i≤m-1, the mask B[c] has i^{th} bit set to 0 if and only if P[i]=c. These correspond to the transition of implicit automaton, that is, if the i^{th} bit in B[c] is 0, then there is a transition from the state i to state i+1 with character c, otherwise there is transition from state i to previous states (or on itself). It uses a bit state vector D of length m for the state of the automaton. The i^{th} bit of the state vector is set to 0, if

and only if it is active. Initially, each bit of D is set to 1(which denotes starting state). For each symbol c of text T, the vector D is updated by D← (D<<1)|B[c] (A new state is obtained only when there is a transition from the current state on the input symbol c, otherwise it remains in the current state). This simulates all the possible transitions of the automaton in a single step. If after the updating of D, the $(m-1)^{th}$ bit of D is zero, then there is an occurrence of P. If m≤w, then the algorithm runs in time O (n).

3.1.2 Standard Shift-or for Multiple Patterns

The algorithm and example for handling multiple patterns using standard shift-or algorithm is given below

```
MULTIPLE_STANDARD_SHIFT_OR (T, n, P [k][m])
 // T is Text; P is pattern containing k number of
patterns //each of equal size m, n is the length of Text, σ
is the //size of alphabet Σ. We assume that m≤n
1 for i←0 to σ-1
2    do for j←0 to k-1
3          do B[j][Σ[i]] ←~0
4 for i←0 to m-1
5    do for j←0 to k-1
6          do B[j][P[j][i]] ←B[j][P[j][i]]&~(1<<i)
7 for i←0 to σ-1
8     do Bit[Σ [i]] ← ~0
9 for i←0 to σ-1
10  do for j←0 to k-1
11        do Bit[Σ[i]] ← Bit[Σ[i]] & B[j][Σ[i]]
12 D←~0, mm←1<<(m-1), i←0
13 while i<n
14   do D← (D<<1) | Bit [T[i]]
15       if (D & mm≠mm)
16        then for j←0 to k-1
17                 do verify occurrence of j^th pattern at
                        shift i-m-1
18        i←i+1
```

3.2 Average Optimal Shift-Or

This algorithm is a category of Shift-Or algorithm, which allows skipping of text characters. This algorithm takes a parameter q, and from the original pattern, generates a set *P* of q new pattern, i.e. P= $\{P_0, P_1, .., P_{q-1}\}$ each of length m'=⌊m/q⌋ as follows:

$P^j[i]=P[j+i\times q]$, j=0,1,2…q-1, i=0,1,2 …⌊m/q⌋-1

In other words, we generate q different alignments of the original pattern P each alignment containing only q^{th} character. The total length of the pattern P^j is q×⌊m/q⌋≤m.

For example, if P=a b c d e f and q=3, then P_0=ad, P_1=be and P_2=c f. Assume now that P occurs at T [i…i+m-1]. From the definition of P^j it directly follows that $P^j[h]=T[i+j+h\times q]$, j=i mod q, h=0, 1, 2…m'-1

This means that we can use the set P as a filter for the pattern P, and that the filter needs only to scan every q^{th} character of T.

Figure given below illustrates the concepts.

P= a b c d e f

T= x x a b c d e f x x x

P^0= a d

P^1 =b e

P^2= c f

P'= a d b e c f

Assume that P occurs at text position T[i…i+m-1] and q=3. The current text position is p=10 and T [p]=b. The next character the algorithm reads is T [p + q]=T [13]=e This triggers a match of $P^{p \bmod q}=P^1$ and the text area T [p-1…p-1+m-1]= T [i...i+m-1] is verified. The set of patterns can be searched simultaneously using the shift-or algorithm, as long as q×m'≤ w. All the patterns are preprocessed together, as if they were concatenated

4. The Proposed Algorithm

The proposed algorithm for handling **multiple patterns** has been divided into two parts namely Preprocessing + Matching and Verification part

4.1 Preprocessing +Matching Algorithm

```
MULTI_AVG_SHIFT_OR (T, P [k][m], q, n)
//T is a text, P is pattern, k is the number of patterns
//each of size m and n is the size of text
1 for i←0 to σ-1 //σ is the size of Σ
2    do for j←0 to k-1
3          do B [j][Σ [i]] ← ~0
4 for l←0 to k-1
5    do mm←0,h←0
6         for j←0 to q-1
7            do for i←0 to (m/q)-1
8               do B [l][ P [l][i×q +j]] ←B [l][P[l][i×q +
                      j]]&~(1<<h)
9                    h←h+1
10 mm←mm|(1<<(h-1))
11 for i←0 to σ-1
12   do Bit [sigma [i]] ← ~0
13 for i←0 to σ-1
14    do for j←0 to k-1
15          do Bit [sigma [i]]& ← B [j][sigma [i]]
16 D←~0, i←0
17 while i<n
18  do D← ((D &~ mm)<<1)|Bit[T[i]]
19       if((D &mm)≠mm)
20          then print(" Pattern may occur and hence
                      Verify");
21               VERIFY (T, i, P, q, D, mm)
```

4.2 Verification Algorithm

```
VERIFY (T, i, P [k][m], q, D, mm)
1 j←0
2 D← (D & mm) ^mm
3 while D≠0
```

```
4   do s←log (D)/log (2)
5       c←-(m/q-1)×q-s/(m/q)
6      for l←0 to k
7        do for j←0 to m-1
8             do if (P [l][j]=T [i+ c+ j])
9                then j←j+1
10               else break
11            if (j=m)
12             then flag←1
13                  print ("Pattern k+1 occur with shift
                          i+c-1)
14 D←D&~ (1<<s)
```

4.3 The Example

Let T=a b a a b a b, P_1=a b a a , P_2= a b a b ,q=2
P_1'=a a b a, P_2'= a a b b
B_1[a]=0100 , B_1[b]=1011
B_2[a]=1100 , B_2[b]=0011

B[a]=0100, B[b]=0011 , mm=1010, D=1111 , i=0
Step 1(i=0): D= (1111&0101) <<1|0100
=1110
Step 2(i=2):D=((1110&0101)<<1)|0100
=1100
Now D &mm ≠mm, therefore we need to verify
Call **Verify (T, 2, 7, P, 4, 2, 1100, 1010)**
Begin D=(1100&1010)^1010=0010
s=1,c=-2
Now, P[0…3]=T[0…3]
Therefore, pattern occur with **shift i+c-1= -1**
End
Step 3:(i=4)
D=((1100&0101)<<1)|0011
=1011
Step 4: (i=6)
D=((1011&0101)<<1)|0011
=(0001<<1)|0011
=0011
Now, D&mm=0011&1010=0010 ≠mm, therefore we need to verify
Call **Verify (T, 6, 7, P, 4, 2, 0011, 1010)**
Begin
D=(0011&1010)^1010
=(0010)^1010=1000
s=3, c= -(4/2-1)×2-3/2=-3
Now, T[i+c…….i+c+m-1]=P[0….m-1]
i.e. T[3…6]=P[0…3]
Therefore, pattern occur with shift i+c-1=6-3-1=2
End

5. Experimental Results

The performance of the proposed algorithm compared with the performance of the standard shift or algorithm for multiple patterns. We have analyzed these algorithms for ASCII character set where s=256. All algorithms are tested under on Celeron(R) processor, 2.40 GHz and 256 MB RAM. The fig. 5.1, 5.2 and 5.3 shows the comparison of the said algorithms

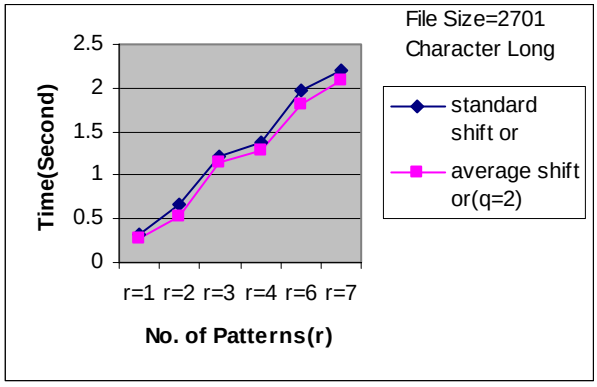

Fig. 5.1 Comparison between algorithms for multiple patterns (Each pattern is of length m=3)

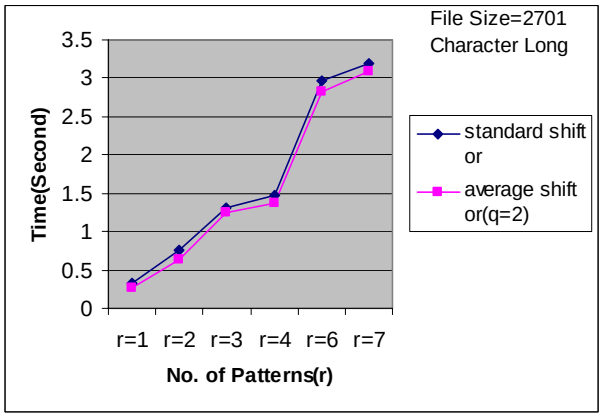

Fig. 5.2 Comparison between algorithms for multiple patterns (Each pattern is of length m=5)

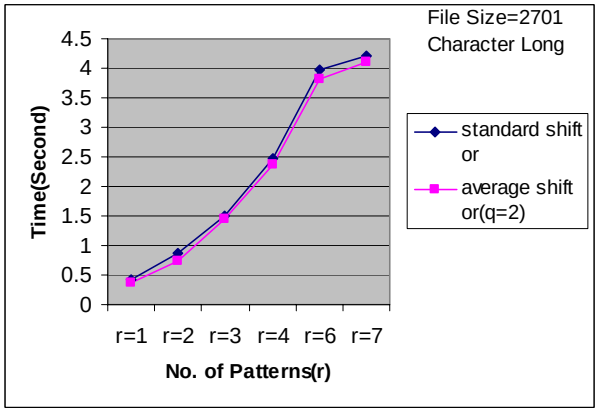

Fig. 5.3 Comparison between algorithms for multiple patterns (Each pattern is of length m=7)

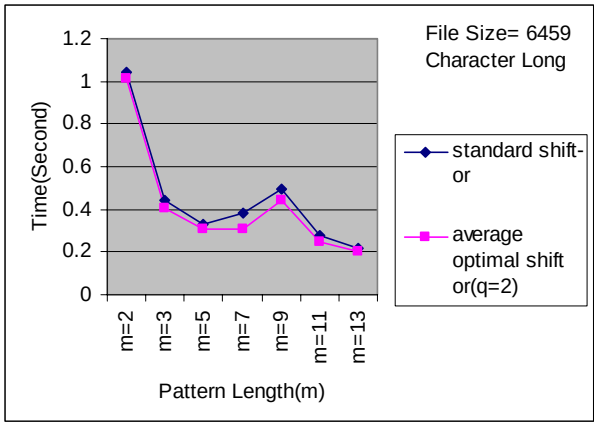

Fig. 5.4 Comparison between algorithms for single pattern

6. Conclusions:

We have implemented the average optimal shift-or and standard shift-or algorithm for various values of r (number of patterns) keeping m (pattern length) fixed. Fig. 5.1, 5.2 and 5.3 shows that average optimal shift-or give better performance when compared with standard shift-or algorithm. We implemented the same algorithm of single pattern for multiple patterns and compared with same algorithm of multiple patterns and found that algorithm of multiple pattern takes less time/space than algorithm of single patterns. Our approach reduces the processing time and the space requirement of the standard shift-or algorithm for any types of alphabet

References:

[1] Kimmo Fredriksson, Szymon Grabowski, "Practical and optimal String Matching" In Proceedings of SPIRE.2005, Lecture Notes in Computer Science 3772, Springer Verlag, Berlin, 2005 ,pp. 374-385

[2] Kimmo Fredrifsson, Jorma Tarhio "Efficient String Matching in Huffman Compressed Texts" In Fundamenta Informaticae 63(1), 2004, pp. 1-16

[3] Kimmo Fredriksson "Shift or String Matching with Super Alphabet" In Information Processing Letters 87(4), 2003, pp. 201-204

[4] Kimmo Fredriksson "Faster String Matching with super alphabets" In Proceedings of SPIRE.2002, Lecture Notes in Computer Science 2476, Springer Verlag, Berlin, 2002, pp. 44-57

[5] Jeri Kytojoki, Leena Salmela, Jorma Tarhio "Tuning String matching for Huge Pattern set" In proceeding of Combinatorial Pattern Matching (CPM), Lecture Notes in computer Science, 2676, Springer, 2003, pp. 211-224

[6] R.Baeza-Yates, G.H.Gonnet "A New Approach to text Searching, Communication of the ACM 20 (10),1992 ,pp.762-772.

[7] A.V. Aho, M.J. Corasick "Efficient string matching An aid to bibliographic search" Communication of ACM 18(6), 1975, pp.333-340

[8] Leena Salmela, Jorma Tarhio, Jari Kytojoki "Multi-Pattern String matching with q-Grams" ACM Journal of Experimental Algorithmics, to be published

[9] P.D. Michailidis, K.G. Margaritis "On -line String Matching Algorithms" Survey and Experimental Results International Journal of Computer Mathematics, 76(4), 2001, pp. 411-434

[10] Gonzalo Navarro, Mathieu Raffinot "A Bit-Parallel Approach to Suffix automata: Fast extended string matching" In proceeding of the 9th Annual Symposium on Combinatorial Pattern Matching LNCS 1448,1998, pp.14-33