

Report on Algorithm of Sunday and Horspool

Qaiser Abbas

Roll No. 07-0906, MS(CS)

Department of Computer Science

National University of Computer & Emerging Science, Lahore

L070906@lhr.nu.edu.pk

1. Fast Searching Algorithm by Sunday

To find substring search in large string of text is the common necessity and two famous the Knuth Morris Pratt (KMP) and Boyer Moore (BM) algorithms have linear search behavior in worst case which is the proof of their success. After modifying the BM algorithm, it is observed that it becomes faster than any other algorithm and moreover, it does not depend on some particular order of scanning. Three approaches are adopted to strengthen the BM algorithm, which are Quick Search, Maximal Shift and Optimal Mismatch. Let $p[i]$ be the i^{th} character in the pattern string $p = p[0], \dots, p[m-1]$ of length m and $t[j]$ be the j^{th} character in the text string $t = t[0], \dots, t[n-1]$ of length $n > m$. we assume further that the pattern string p is located at position k in the text string t such that $p[0]$ is located at $t[k]$ in text string and $p[1]$ at $t[k+1]$ and so on up to $p[i]$ at $t[k+i]$ where $i = m-1$. The running of straight forward (SF) algorithm, KMP and BM algorithm can be seen in article [1]. We discuss only the improved version here.

A new $\Delta 1$ is computed to be the index of the first leftward occurrence of character from the end of text string, then $\Delta 1 = \text{TD1}[t[k+m]]$ and whenever a mismatched is found then the value of this $\Delta 1$ is the amount of shift p to the right. This aligns character in p with the text string character $t[k+m]$, or character does not occur in p , shifts p right past it to text position $(k+m+1)$. Using $\Delta 1$ instead of $\partial 1$ has benefits like $\Delta 1 \geq 1$ always to simply and quickly code an algorithm, and $\Delta 1 \geq \partial 1 + 1$ when the last character of pattern matches then $\Delta 1 \geq \partial 1 + 2$ and so on only for short pattern string because for long string the speed decreases, $\Delta 1$ also independent of order of scanning of the pattern string while $\partial 1$ is dependent from right to left order. This last feature is important and can be represented by an index array as $I[j] = \{I[0], \dots, I[m-1]\}$ which is permutation of $\{0, \dots, m-1\}$ and $I[j]$ and $P[I[j]]$ is the location of j^{th} element and the character at location respectively for each $j = 0, \dots, m-1$. For any specific scanning order $\Delta 2$ shift can be defined, then for substring search algorithm Δ pattern shift can be used which is the maximum of $\Delta 1$ and $\Delta 2$. Now we suppose that $\text{TD2}[j]$ is the precomputed

$\Delta 2$ for first mismatch occurrence at $I[j]$. The procedure of precomputing is given in article [1] thus continue defining $\text{TD2}[j]$ to be the minimum left shift in a way that $p[I[0]] \dots p[I[j-1]]$ will match their aligned character in p and $p[I[j]]$ does not. The $\Delta 2$ shift table for specific order can be computed by the algorithm1 given in the paper [1] in which $\text{matchshift}(j, lshift)$ returns the value of next leftward shift where first j ordered pattern character matched their aligned string characters which is minimum value of $mshift \geq lshift \geq 0$ such that $(I[i] - mshift) < 0$ or $p[I[i]] = p[I[i] - mshift]$ for each $i = 0, \dots, j-1$. It is pertinent to note that for forward scanning $\Delta 2$ is same as KMP ∂ and for backward scanning $\Delta 2$ is equivalent to BM $\partial 2$. Now since the specific ordered is maintained and table $\text{TD1}[\]$ and $\text{TD2}[\]$ are computed then algorithm2 for new substring searching can be codes easily [1]. This algorithm has linear $O(n)$ worst case behavior which is similar to KMP and BM algorithms. The proof of the linearity is through some conjecture as “the arbitrary scan order substring searching algorithm has $O(n)$ worst case behavior”. Variation based of different scan orderings are as follows:

1.1. Quick Search

SF and $\Delta 1$ are used to code algorithm quickly as mentioned earlier in detail and $\Delta 2$ will not be used. Moreover, it is debugged and executed quickly. Hence it is called quick search algorithm which can be seen as algorithm3 in the article [1]. One can also augment SF with $\Delta 2$ to obtain a fast KMP algorithm.

1.2. Maximal Shift

To maximize $\Delta 2$ shift values, one is to pick the first character in the pattern string p whose next leftward occurrence in p is a maximal distance away. If it matches, we shift the pattern string to right before the next valid comparison position is reached. Repeat this process until the end of the pattern string. In this way we can get maximal $\Delta 2$ shift with some other refinement possible. The C code is given in appendix of article [1]

1.3. Optimal Mismatch

The optimal mismatch can be obtained by ordering the characters of pattern string p from the one least likely to occur in the text alphabet to one most likely to occur. This increases the probability of mismatch and result will be efficient. The C code is also given in the appendix of article [1].

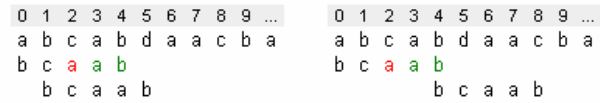
Each of the four algorithms was tested on same string in a large text fixed buffer and comparisons are counted. The first text buffer used was formed from the UNIX dictionary file by discarding non alphabetic characters and converting alphabets to lower case. 200K of characters was found. Now by using each of four algorithms including BM, QS, MS and OM, all occurrences of word in text buffer is searched. Counting number of character comparisons for word and algorithm, the fraction of total characters of the text was computed and recorded along with the average of this fraction as shown in table 1 of the article [1]. Table 1 compares the four algorithms as function of pattern string length “Plen” and “words” give the number of words of each length searched for in the text. This shows that Optimal Mismatch Algorithm is fastest among all algorithms. The increase in speed of algorithm is calculated by ration BM/OM and average for each pattern length is shown in Table I and Table II in article [1]. It shows large increase in speed for short string and general increase of almost 10 percent for longer strings. Table II also shows the max and min values of ratio for any individual word that occurred for each length string. This proves that OM algorithm is as good as BM and significantly faster sometime. Table III is calculated from another test made to compare the ratio of BM/OM on UNIX concatenated manual pages. Format command lines are used to filter raw and unformatted manual pages and conversion of characters to lower case and nonalphabets including white spaces were remained in the text. This resulted in 3MB of technical English text. It was observed the result of this search is same as computed earlier for text search test.

BM algorithm is the fastest string searching approach till today because a lot of versions for fastest string search algorithm are introduced but they all depend mainly on the BM algorithm. Quick Search approach is choice of every programmer when search string technique is to be implemented. OM algorithm is fastest among all and gain 20 percentage increase in speed as compared to all others in case of short strings and 10 percent for longer string of English. Other than English, performance at some level is expected than BM algorithm. However the degree of improvement or performance would be different depend

on the size of the text and frequency of occurrence of characters.

2. Horspool Approach:

The BM uses two heuristics to determine the shift distance of a pattern in case of mismatch: the bad character and the good suffix heuristics. Good suffix is complicated to implement so algorithm based on bad character heuristic is required. The idea of the Horspool is that in each case right most character of current text is used for determining shift distance instead of bad character.



(a) Boyer-Moore

(b) Horspool

Figure: Flow of BM and Horspool [3]

We have to compare $t_0...t_4 = a b c a b$ is the current text window with the pattern. Suffix $a b$ has matched but c cause mismatch. The bad character heuristic of the BM used c to determine the shift distance (a) and Horspool algorithm used the rightmost character b as shown in figure above. The pattern continues to shift until the rightmost occurrence of b in the pattern matches the text character b where the occurrence of the last position of pattern does not count [3]. Horspool algorithm presumes its best case when every time in the first comparison a text symbol is found which does not occur at all in the pattern. Then the algorithm performs $O(n/m)$ comparisons.

Many programmers believe that BM approach is not applicable and the machine instructions for searching string are more reliable than BM approach. The Horspool negate this belief and performs experimentation as proof. STRING represents the text and STRINGLEN be the length of the string and PAT is the pattern while PATLEN is the length of the pattern. $S[I...J]$ represents substring in algorithm SFC [2]. If algorithm returns 0 then pattern does not found in the text otherwise the position of the first occurrence is returned by the algorithm. For the case of upper case text only and if the STRING is ‘EXTRA’, then E have probability of match after every 10 characters but X does not, since it is one of the least frequent letters in English and the skipping of words during search maximize the speed of your algorithm. This method (Scan of Lowes Frequency Character) is also discussed in [2]. The SLFC algorithm uses character frequency information into order as per expected frequency of occurrence; even perfect ordering

is not really required. SFC and SLFC performance is same however; SLFC performs better in random ordering. The effect of character frequency information is shown in Table I in the article [2] with detail. No need to discuss BM algorithm here because it is already discussed and also given in the article [2]. However, to verify the unimportance of $\partial 2$ in BM, we present a modified version of BM as SBM. Our table of $\partial 12$ is same as $\partial 1$ except that $\partial 12[\text{LASTCH}]$ has its value taken from $\partial 2[\text{PATLEN}]$. Algorithm is given in [2]. All the four algorithm are coded on 370 assembler and timed them to locate every occurrence of some pattern within a very large block of 80,000 characters of text. The result are given in Table II of the article and measured on Amdahl V7 computer. BM and SBM give nearly identical timings demonstrating the unimportance of the $\partial 2$ table and SLFC is superior to SFC. We have also seen that BM and SBM have similar speed to SFC for pattern length equal to five and for larger length SFC is inferior and for smaller length it is superior. Since timing does not include initializing tables and $\partial 2$ requires the most computation and hence another justification for SBM version of BM. It is concluded that BM perform than hardware if pattern length is six or greater. So finally, the computers which have lack of search instruction may use the SBM version otherwise the composite strategy is adopted as *if patlen <= threshold then search with slfc method else search with sbm method*; moreover, the value of THRESHOLD is 5 on above mentioned computer and may differ on others.

In both algorithms, shift function is implemented through an array. The main difference is the argument for the shift function, which is, the subscript of array. HORSPOOL used the current text character while SUNDAY used current text character plus pattern length. Sunday's approach is to skip further, *i.e.*, $Plen+1$, when character text $[i+Plen]$ does not occur in the pattern. Both the algorithms are also differing in their pattern matching order. SUNDAY's algorithm matches pattern from left to right *i.e.* the starting position of a possible match is return by shift function while HORSPOOL's algorithm matches pattern from right to left *i.e.* the end position of a possible match is returned. Both algorithms align the current text character to the rightmost occurrence of the current text character in the pattern; thus, conflicts from multiple alignments are automatically resolved by aligning to the rightmost occurrence of the current text character.

3. References

[1] Daniel M. Sunday, "A Very Fast Substring Search Algorithm", Communication of ACM August 1999, Vol 33, No. 8,

[2] R N Horspool, "Practical Fast Searching in String", School of Computer Science, McGill University, 805 Sherbrook Street West, Monteval Quebec, H3A2K6, Canada
 [3] H W Lang, "String Matching: Horspool Algorithm", FH Flensburg, Germany, 2008.
 [4] Sun Kim, "A New String-Pattern Matching Algorithm Using Partitioning and Hashing Efficiently", The Biotechnology Center, The University of Illinois at Urbana Champaign Urbana, IL 61801.