BOYER-MOORE

Name: Sriharsha Oddiraju
Department:Computer Science
Indiana State University
TerreHaute IN,USA

December 16, 2011

Abstract

The Boyer-Moore is a string searching algorithm. It was developed by Bob Boyer and J Strother Moore. The algorithm pre-processes the target string that is being searched for. When the string is being searched, the algorithm starts searching at the beginning of the word, it checks the last position of the string to see if it contains the matching letter. If it finds the letter in the last position it moves to the previous position and so on it checks the first position of the string. The Boyer-Moore algorithm is backward approach,instead of an target letter in the last position if it finds the other letter in that position, this means there no match for the search string at the very start of the text or at the next seven positions following it.

1 Statement of the Problem

The goal of any string-searching algorithm is to determine whether kor nota match of a particular string exists within another string. Many such algorithms exist,with varying efficiencies. The Boyer-Moore algorithm successively aligns Pattern string with the Target string and then checks whether Pattern string matches the opposing characters of Target string. target string. After check is complete, Pattern string is shifted right relative to target string. Boyer-Moore algorithm contains three steps.

- Right to Left Scan
- The bad character rule
- Good suffix rule

These steps lead to method that examines fewer than $m + n$ characters(an expected sub-linear-time method), and that (with certain extension)runs in linear worst case time.

2 Time Complexity

2.1 History

In the year the algorithm was devised in 1977, the Boyer-Moore string search algorithm is a particularly efficient algorithm, and has served as a standard benchmark for string search algorithm ever since, the maximum number of comparisons was shown to be no more than $6n$; in 1980 it was shown to be no more than $4n$, until Cole's result in Sept 1991. It performed the character comparison in reverse order to the pattern being searched for and had a method that did not require the full pattern to be searched if a mismatch was found. Legend has it that the original algorithm was coded in PDP-10 assembler.Computer hardware has changed very considerably since that time yet the fundamental logic is still sound in a different world under very different conditions. Modern personal computers now have the capacity to work on very large sources and often have enough memory to handle those sources without requiring any form of tiling scheme and this capacity very well suits the design of the Boyer Moore exact pattern matching algorithm.

3 Variants of Boyer-Moore

- Turbo Boyer-Moore takes more space but needs $2 * N$ steps in the worst case instead of the original $3 * N$ steps.
- Boyer-Moore-Horspool uses less space (only requires the good suffix table) and has a simpler inner loop with less constant overhead per iteration. Its average performance is $O(N)$ but it has a worst-case performance of $O(M * N)$.
- Boyer-Moore-Horspool itself also has variants. In Timo Raita's 1992 paper .Tuning the Boyer-Moore-Horspool String Searching Algorithm. he asserts that text is rarely random and that there usually exist strong dependencies between characters. Thus there are smarter ways to match the needle: instead of matching it backwards from the last character, one should first match the last character, then the first character, then the middle one etc.

3.1 Best-Case

The **Best-case** performance of the algorithm, for a text of length n and a fixed pattern of length m, is $O(n/m)$ in the best case, only one in m characters needs to be checked.

3.2 Worst-Case

The **Worst-case** to find all occurrences in an aperiodic text needs approximately $3n$ comparisons, hence the complexity is $O(n)$, regardless whether the text contains a match or not.

4 Brute Force Method

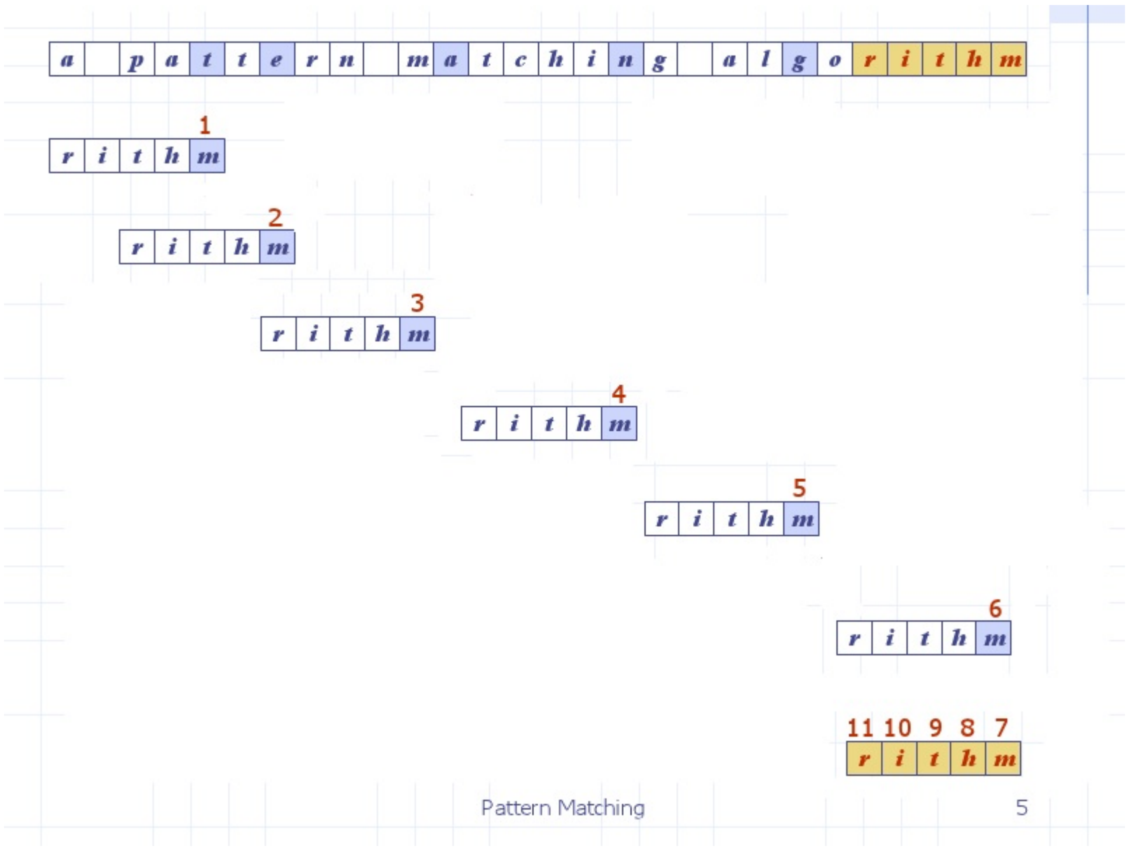
1. The brute-force pattern matching algorithm compares the pattern P with the text T for each possible shift of P relative to T, until either
   - a match is found, or
   - all placements of the pattern have been tried
2. Brute-force pattern matching runs in time $O(nm)$
3. Example of worst case:
   - $T = aaa....ah$
   - $P = aaah$
   - may occur in images and DNA sequences
   - unlikely in English text

5 Brute Force Algorithm

```
**Algorithm** $BruteForceMatch(T, P)$
Input text $T$ of size $n$ and pattern $P$ of size $m$
Output starting index of a substring $T$ equal to $P$ or $-1$ if no such substring
exists
**for** $i \leftarrow 0 \rightarrow n - m$ **do**
  test shift $i$ of the pattern
  $j \leftarrow 0$
  **while** $jm \wedge T[i + j] = P[j]$ **do**
    $j \leftarrow j + 1$
    **if** $jm$ **then**
      **return** $i$ match at $i$
    **else**
      **while** $mismatch$ **do**
        **return** $-1$ no mismatch anywhere WHILE
      **end if**-WHILE
```

6 Boyer Moore Heuristics

- The Boyer-Moore pattern matching algorithm is based on two heuristics
  Looking-glass heuristic: Compare P with a subsequence of T moving backwards
  Character-jump heuristic: When a mismatch occurs at $T[i] = c$
  If P contains c, shift P to align the last occurrence of c in P with $T[i]$
  Else, shift P to align $P[0]$ with $T[i + 1]$


a pattern matching algorithm
1
rithm
2
rithm
3
rithm
4
rithm
5
rithm
6
rithm
11 10 9 8 7
rithm
Pattern Matching
5

7 Last-Occurrence Function

1. Boyer-Moore algorithm pre-processes the pattern P and the alphabet σ to build the last-occurrence function L mapping σ to integers, where $L(c)$ is defined as

   – the largest index i such that $P[i] = c$ or

   – -1 if no such index exists

   – The computation of the last function takes $O(m + |\Sigma|)$

$$last(c) = \begin{cases} \textit{index of last occurance of } \text{c} \textit{ in pattern } \text{P} & \text{if } c \text{ is in P} \\ \text{-1} & \text{otherwise} \end{cases}$$

char	**a**	**b**	**c**	**d**
last(char)	4	5	3	-1

2. **Example:**

   – $\sigma = \{a, b, c, d\}$

   – $P = abacaabacc$

8 Bad Character Rule

For any x $\in \sigma, letR(x) = max(0 \ \cup \{i < n | P[i] = x\})$
Easy to compute in time $O(|\sigma| + |P|)$ (σ *is the alphabet*)
Bad character shift: When $P[i] = T[h] = x$, shift P to the right by $max\{1, i - R(x)\}$.This means:

– if the right-most occurrence of x in $P[1...n-1]$ is at $j < i$, chars $P[j]$ and $T[h]$ get aligned

– if the right-most occurrence of x in $P[1...n-1]$ is at $j > i$, the pattern is shifted to the right by one

– if x doesn't occur in $P[1...n-1]$, $shift = i$, and the pattern is next aligned with $T\ [h+1...h+n]$

Bad character shift rule

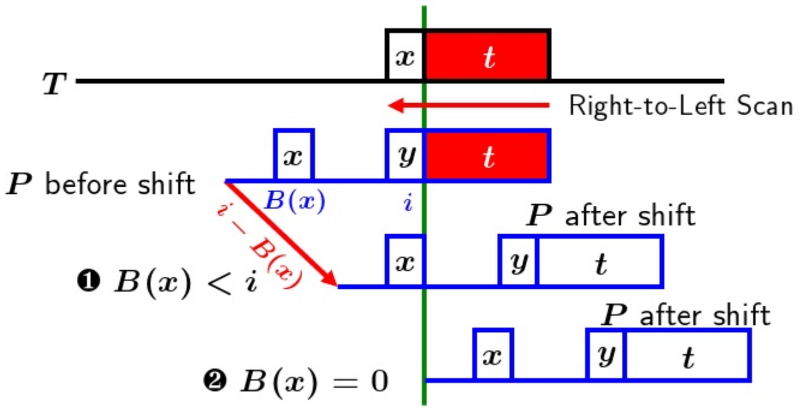


Figure 1: Bad Character Rule

Bad character shift rule

If $B(x) > i$, then bad character rule has no effect.

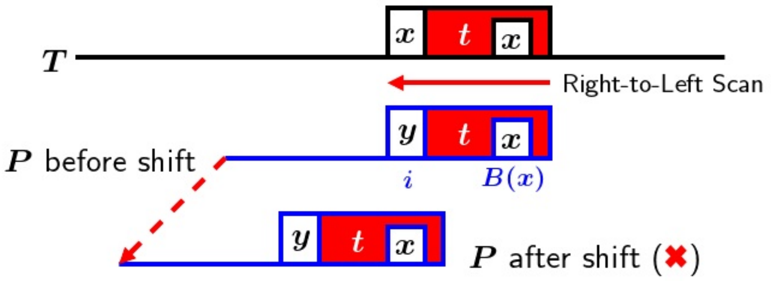


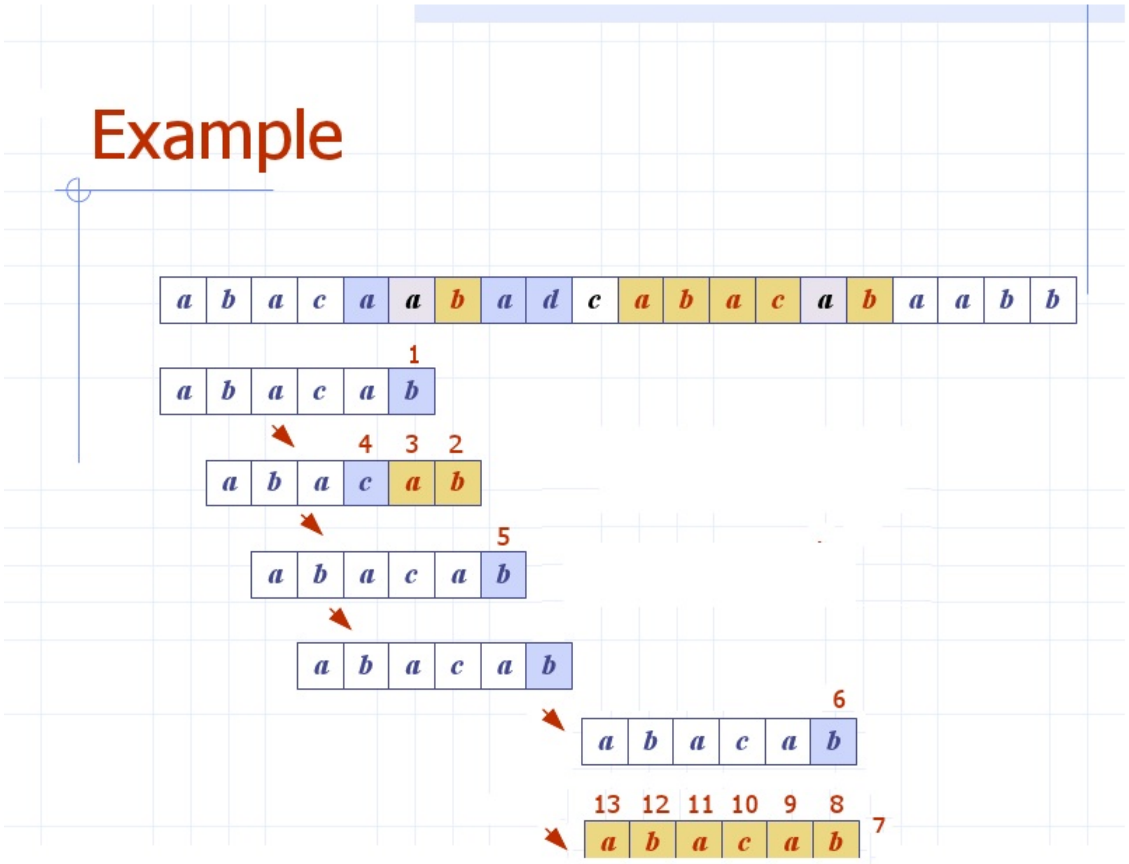
Figure 2: Bad Character Rule

9 Working of Algorithm

– The B-M algorithm takes a backward approach: the target string is aligned with the start of the check string, and the last character of the target string is checked against the corresponding character in the check string.

– In the case of a match, then the second-to-last character of the target string is compared to the corresponding check string character.

– In the case of a mismatch, the algorithm computes a new alignment for the target string based on the mismatch. This is where the algorithm gains considerable efficiency.



10 Comparison of String Searching Algorithm Complexities

- Boyer-Moore: $O(n)$
- Naive string search algorithm: $O((n-m+1)m)$
- Bi-tap Algorithm: $O(mn)$
- Rabin-Karp string search algorithm: [average $O(n+m)$]
  (n = length of search string, m = length of target string)

References

[1] Wikipedia entry,
`http://en.wikipedia.org/wiki/Boyer`

[2] Boyer-Moore string search algorithm,
`http://www.movsd.com/bm.htm`

[3] A Fast String Searching Algorithm Robert S.boyer,Stanford Research Institute J strother Moore Xerox Palo Alto Research Center

[4] http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html