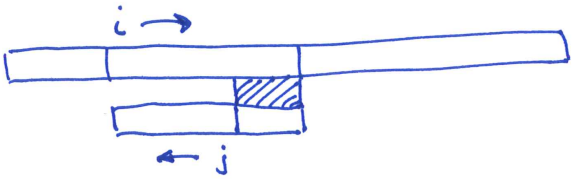
Institut for Matematik og Datalogi
Syddansk Universitet, Odense

23. januar 2017
RF

The BM Algorithm

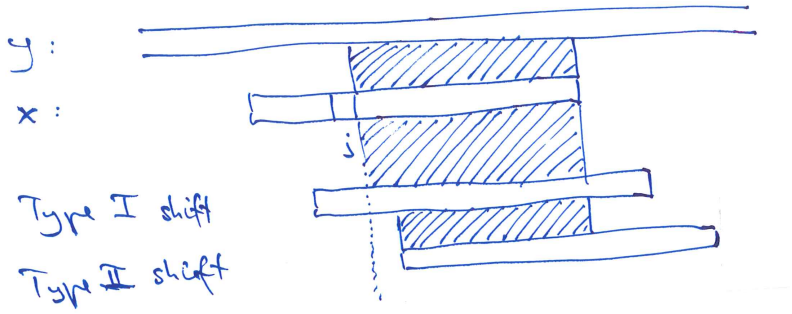
The Boyer-Moore (BM) algorithm for exact pattern matching was published in 1977 by Robert S. Boyer and J. Strother More (with some input from others, see the original paper for details). Like the KMP algorithm, the BM algorithm uses a sliding window method, and it preprocesses the string to create a shift-table (different from the KMP shift table). In contrast to the KMP-algorithm, the sliding window is checked in the opposite direction of the window's slide direction, as illustrated below (shaded area means matching characters).



For a pattern x and a string y, of length m and n, respectively, the code can be expressed as follows (assuming the first character of a string has index 0):

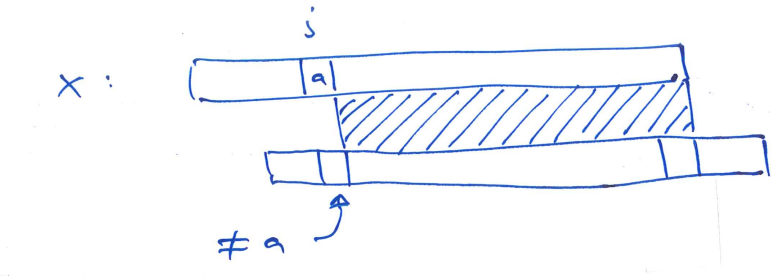
```
i = 0
WHILE i < n-m
  j = m-1
  WHILE j >= 0 and x[j] == y[i+j]
    j--
  IF j < 0
    RETURN "Match at position i"
  i = i + BMShift[j]
```

Here, BMShift(j) is a safe shift (no possible occurrences of x in y are jumped over) of the window, given the value of j at the exit of the inner `WHILE`-loop. To define BMShift(j) in details, we first consider the case $0 \leq j \leq m-1$, i.e., the current window position was an unsuccesful attempt. The two upper levels of the figure below show the string y, the current window position for the pattern x, and the area of known matching characters.



For a shifted window position overlapping the current window position, we can deduce that for the shifted position to be able to contain an occurrence (a full match between x and y), there must be matches between areas *within* the pattern x (due to both areas in x are matching the same positions in y). In the two lower levels of the figure above, these matching areas are indicated by shading. This requirement of match within x we call R_1.

As can be seen in the figure above, we separate the possible shifts into Type I and Type II, depending on whether the start of the new window position is past the point j of failure for the current window position. For Type I shifts, we can deduce another requirement for the new window position to allow an occurrence (a full match between x and y): the character `a` at index j in x should *not* match the character in x which in the new window position lines up with `a`, as illustrated in the following figure.



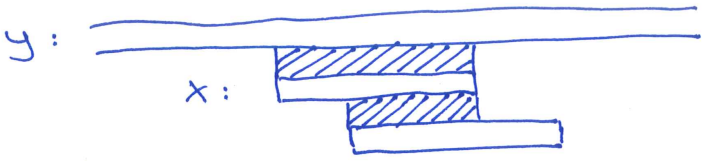
This restriction follows because we know that the corresponding character in y is not `a`, since the current window position is unsuccessful. We call this requirement between two characters of x for R_2.

For a given value of j, we call a shift distance d ($1 \leq d \leq m$) for *eligible* if it is of Type II and R_1 is fulfilled, or it is of Type I and both R_1 and

R_2 are fulfilled. The shift distance m is always eligible. We then (for each $0 \leq j \leq m-1$) define BMShift(j) to be the minimum eligible shift distance for that j.

These values of BMShift(j) for $0 \leq j \leq m-1$ can be calculated in $O(m)$ time in total. We relegate to a separate note (two set of notes, actually) the description of how to do it, since it is not entirely trivial.

We now consider the case $j = -1$, i.e., where the current window position was a succesful attempt. In this case, we want the minimal shift distance $d \geq 1$ of x for which the overlap between the two positions of x matches, as illustrated below.



This distance is called the *period* of x. Recalling from the discussion of the KMP algorithm that the border of x is the largest proper prefix of x which is also a suffix of x, it is clear from the figure that the two notions are related as follows:

$$\text{period}(x) = m - \text{border}(x)$$

Since the preprocessing for the KMP algorithm involves computing the borders of all prefixes of x, including of x self, we already know how to compute the period of x in $O(n)$ time.

Summing up, the preprocessing for the BM algorithm involves computing BMShift(j) for $-1 \leq j \leq m-1$, which can be done in $O(m)$ time.

The worst case complexity of the BM algorithm itself can be proven to be $O(n)$ for finding the first match. The proof of this is non-trivial, and is not given here. Finding *all* matches of a pattern x in a string y can in the worst case be $\Theta(mn)$, no better than the naive algorithm. This is for instance the case if x and y are all `a`'s.

However, in most real-life applications, in particular with long patterns, the shifts made during unsuccessful attempts mean that the BM algorithm often reads only a subset of the characters in y. This is in contrast to the KMP

algorithm, which always reads all characters in y. In almost all situations, BM is significantly faster than KMP. Additionally, a small extension of BM given by Galil runs in $O(n)$ worstcase time, even when reporting all matches (see separate note on the Galil version).

As a final note, the original BM algorithm also added a requirement R_3, which is the same as the one we in earlier notes have described under the name Horspool heuristic. Thus, Horspool did not invent that heuristic. Rather, he proposed (in 1980) to simplify the algorithm to use *only* R_3. Worstcase, this is $\Theta(mn)$ (also for unsuccessful searches), but even this simple algorithm is often efficient in practice.