# Report a1q2

## Modifications made and Preprocessing

For the bad-character rule, the logic was inverted since now we had to scan the pattern from left to right and the pattern starts from right most of the text and is shifted leftwards. In the algorithm we learned in week 2, when a mismatch occurs at position j in the pattern, the algorithm looks to the left of j for the rightmost occurrence of the mismatched text character. In the reverse version, we look to the right of j for the leftmost occurrence of that character. If such an occurrence exists at position k, the shift is set to k – j, which is the smallest safe move that realigns this occurrence with the mismatched text position. If the character does not occur again in the suffix of the pattern, the shift is set to m – j, which guarantees that the mismatched text position is outside of the pattern in the next iteration.

The good-suffix rule was changed into a good-prefix rule. Since scanning is left-to-right, what we have matched before the mismatch is a prefix of the pattern, not a suffix. If this prefix reoccurs anywhere else in the pattern starting at some position p ≥ 1, then the pattern is shifted left by p, aligning the reoccurrence with the text. If there is no such reoccurrence, the shift uses the border of the prefix which is the longest substring that is both a prefix and a suffix of the matched prefix. In that case, the shift is j – borderLen, ensuring that part of the prefix can still be reused. When the entire pattern matches, the shift is calculated using the period of the pattern, calculated as m – longest_border(pat).

For the reverse bad-character rule, an extended bad-character table is built. For each character and each position j, this table records the leftmost position of that character in the suffix pat[j+1..], or –1 if it does not appear. We scan backwards through the pattern and update the nearest known future position of the character. For the good-prefix rule, two types of information are needed: reoccurrences of prefixes and border lengths. Both are derived using the Z-algorithm. The Z-array of the pattern identifies where prefixes reappear; from this, minimal left shifts are recorded for each possible prefix length. Border lengths are then obtained by converting the Z-array into a prefix-function-like array, which provides the longest border for every prefix. Together, these allow the construction of a good-prefix table, where each entry specifies the correct shift for a given matched prefix length.