Report a1q2

Modifications made and Preprocessing

For Boyer-Moore, we had 2 shift rules, bad character shift rule and good suffix shift rule. Now that we have to start pattern matching from the right to left, we have to make some modifications to both shift rules. For the bad character shift rule, in the original Boyer-Moore, we do explicit character comparison until we find a mismatching character. Then, we look for the rightmost occurrence of the mismatched character in the pattern and calculate the shift. We then shift so that the mismatched character in text lines up with the rightmost occurrence of the mismatched character in pattern. Now when we do reverse Boyer-Moore, we have to instead look for the leftmost occurrence of the mismatched character in pattern from the right side of the current position then shift leftwards to align this occurrence. As for the good suffix rule, based on the matching suffix appearing somewhere in the pattern, we shift rightwards to align the next occurrence of the suffix. Now, in the reverse Boyer-Moore, we look for matching prefixes of the pattern rather than the suffix. We then shift leftwards instead of rightwards to align with the previous occurrence of the matching prefix.

In the extended bad character preprocessing for reverse Boyer-Moore, we scan the pattern from right to left. For each character c, we track the next occurrence as we move leftward. At each position j, we store the leftmost occurrence of c in the suffix pat[j+1:]. This results in a table[c][j] that gives us constant-time lookup of the first occurrence of c strictly after position j. During a mismatch at position j, this table helps determine how far to shift the pattern leftward to align the mismatched character with a later occurrence in the pattern. For the good suffix rule preprocessing, when a prefix of the pattern has been matched but a mismatch occurs, we want to determine how far to shift the pattern leftward to realign with another occurrence of that prefix or a compatible border. We get the border array, which captures the longest proper prefix of the pattern, since we're scanning from left to right. we initialize pointers i and j at the end of the pattern and walk backwards, updating the border array whenever mismatches are found. If a mismatch occurs and no shift has been recorded yet, we set the shift to j - i, which represents the distance needed to skip over the mismatch and align with the next valid border. We also handle the case where only part of the matched prefix appears at the end of the pattern. This ensures that even if no full border exists, the pattern can still be shifted to align with a partial match. The result is a good prefix table where each entry i tells us how far to shift the pattern when a prefix of length i has been matched.