Comments for Assignment 1 q2 FIT3155

I took advantage of the fact that there will only be at most one wildcard present in the pattern, so I saved the index of it in a variable for later use. This process is done during the pre-processing of the extended bad character matrix since I need to iterate through the pattern once. I think that pre-processing the wildcard character into the Rk matrix is not worth the effort. This is because I have to treat it as any possible character, and I have to fill in the values for every column of the unique characters in the alphabet, wasting the optimisation to save space and also increasing the number of computations.

In a bad character shift, we only need to take care of the wildcard if it appears before the point of mismatch. In that case, we treat the wildcard as an instance of bad character, and perform the same bad character shift (i.e. align the rightmost instance of the bad character in the pattern with the mismatch in the text). Although the wildcard character is not pre-processed, we can still obtain the shift to align the wildcard with the bad character in one operation, that is we calculate the distance between the current point of mismatch with the position of the wildcard character. Then we simply take the minimum of the Rk value and the wildcard shift to determine which is the rightmost instance of the bad character.

For the good suffix shift and matched prefix shift, I pre-processed the wildcard altogether, this is because any instance of a matched substring in the pattern will also be a match with the text if a shift to align is made. However it is important to note that this is not the case for Galil's optimisation. So I implemented a check in the code to see if the wildcard is present before or after the point of mismatch. If it is after the point of mismatch, we cannot guarantee that the current match will also be a match after the shift.

To further elaborate the last point, suppose the character in the text that matches the wildcard (**ω**) is **α,** and the character in the good suffix / matched prefix that matches the wildcard is **β**. Although **α = ω**, and **β** = **ω**, we can't prove that **α = β**. Therefore we still need to perform an explicit comparison of the two characters to check if this is the case.

The overall time complexity of the solution is O(N + M) where N is the length of the text and M is the length of the pattern. It takes O(M) time to pre-process each of the extended bad character matrix, z-suffix array, good suffix array and matched prefix array, which totals up to O(4M) = O(M) time. Whereas the main algorithm itself only performs comparisons of each character in the text at most once in the worst case, which takes O(N) time.

The overall space complexity of the solution is O(N + M) where N is the length of the text and M is the length of the pattern. The total space needed to pre-process the extended bad character matrix, z-suffix array, good suffix array and matched prefix array is O(kM + M + M + M), which is O((3+k)M) = O(M), and the space needed to store the result of the algorithm is at most O(N). Totalling up the space needed would be O(N + M).