Q2

The implementation of Boyer-Moore pattern matching with Galil's optimisation, including the possibility of a wild card. To handle a wild card, if present, the pattern is broken up into two parts. The first part is the pattern before the wild card and the pattern after. This allows matches to be found with the first pattern and then the second pattern. Post computation is then done to realise if the matches occur with one space in-between them, being the wild card.

Extended bad character (extBadChar) is used to shift the pattern along the string to the next occurrence of the mismatched character in the pattern. If this character does not exist, then the whole pattern will move past the character being compared. This is done by construction of a 2d array. The rows correspond to each character in the pattern and the columns correspond to each letter in the alphabet. The first column represents "a" and the last column represents "z". The length of the alphabet is a global variable, ALPHABET_SIZE, and to compute the column of the letter in question, the difference in Unicode of the character and character "a" is computed. The values within the 2d array represent the index of the right most occurrence of each character, relative to the current character being compared. A value of -1 suggests that the character does not exist. For example, row n represents the right most occurrence of each character from index n-1 in the pattern. The values are also indexed at 0. Each row is computed by adding the index of the character at index i to row i+1. The row is then copied over into the next row to save the values.

The good suffix array (goodSuffix) is an array where each value refers to the index of the right endpoint of the right most substring that matches the suffix in a given range. Here the index starts at 1. The z algorithm is utilised here, processing the reversed pattern and then reversing the output to utilise the suffixes of pattern. The difference of the length of the pattern with the z suffix value, gives an occurrence of a pattern and this value is used to index the good suffix array to populate it with the right endpoint of the right most substring that matches the suffix. It is also noted that if the preceding character is the same for the suffix and substring, then the good suffix value is not recorded, as it will result in a mismatch.

Matched prefix (matchedPrefix) represents an array where each value corresponds to the longest suffix of pattern[i:] that is also a prefix of the pattern. This is calculated utilising the reversed z array of the pattern and populating the matched prefix array from the end. Each value in the z array represents the length of a substring which matches the prefix of the pattern. These values are then added to the matched prefix array, starting at the end, and only using the largest value in the z array as it goes on.

Utilising each of these three pre-processing components, the Boyer-Moore pattern matching algorithm can be run optimally. This algorithm starts by comparing the right most characters of the pattern first with the string. This allows the pre-processed lists to be utilised to skip comparisons where it is known there will be mismatches or reoccurring substrings. Firstly, comparisons are made from the right of the pattern, where a mismatch occurs, the bad character array or good suffix rules are used to shift the pattern along the string accordingly. Both are utilised by using the max value of the two. Lines 192 and 196 represent a substring that will not need to be compared in the next iteration of the algorithm. Line 192 represents this as a substring that ends at the start of the pattern. Therefore, in the next iteration when comparing, if the comparisons reach this substring, it can be skipped. Similarly for line 196, this represents a substring that occurs within the pattern. Therefore, a start and stop value is recorded, where the algorithm can skip comparing during this interval. This represents Galil's optimisation within Boyer-Moore's algorithm. Within the iterative process, these Galil variables are reset to -1 to insure they are not used in the next iteration, as it will result in unwanted matches. When the end of the pattern is reached, the match is recorded.