Report a1q1

Main Idea

The main idea is to use z algorithm to find pattern matches while allowing for wildcard characters ‘#’ to be used in the pattern to match any characters in the text.

The z algorithm part computes for each position i in a string s the length of the longest substring starting at a that matches the prefix of s. The computed z values are put into the z array. We have a z box from l to r and for case 1, if our i > r then we have to do explicit character comparisons till we find a mismatch and update l and r accordingly. For case 2a, we have k = i - l and if our z[k] < remaining length in the current z box, then we can be sure that the match at i confirm cannot be longer than the current z box and we can simply put z[i] = z[k]. This is because the character at i + z[k] will mismatch like the character at k + z[k]. In case 2b, when z[k] is greater than or equal to the remaining length of the current z box, the we cannot be sure whether the match at i will extend beyond the current z box or not. so from the r onwards, meaning from beyond the current z box onwards, we have to perform explicit character comparison. We then have to update the l and r accordingly.

We concatenate the pattern string with a separator character ‘$’ which does not appear in both the pat and txt then concatenate it with the text string to get pattern + ‘$’ + text. We use z algorithm to get the z array of the whole string. We then iterate through the text and if our z value at i equals the length of our pattern then we get a full match. Otherwise, if we get z[i] < m meaning our 1st z[i] characters match. For the remaining characters, we do explicit character comparison and if pattern is ‘#’ it will match any character in the text. We then return the positions of the valid matches.

The z algorithm has O(n) worst-case time complexity where n is the number of characters in the string s. The total time is proportional to the sum of the number of iterations and explicit character comparisons, with n-1 iterations. Since any iteration performing explicit comparisons terminates at the first mismatch, there are at most n-1 mismatches, and the number of matches is at most n because the right boundary r_k is non-decreasing ($r_k \geq r_{(k-1)}$) and cannot exceed n, as each update takes the form $r_k = r_{(k-1)} + \delta$ where $\delta \geq 0$ but $r_k \leq n$. This guarantees that both matches and mismatches are linearly bounded, resulting in overall linear time complexity (time complexity analysis taken from lecture slides). For the algorithm, z array computation time complexity is O(n+m), where n is length of text and m is length of pattern since we have length of combined string = n+m and in our pattern matching, we iterate through the entire text string to find pattern matched and for each partial match we need O(m) time to check all the remaining pattern characters. In the worst case we get text string like aaaa…. and pattern string of a so we have to check for partial match at every iteration so O(nm). So we have overall worst case time complexity of O(n+m) + O(nm) = O(nm)

For worst case space complexity, our combined string is O(n+m) and z array is also O(n+m). O(n+m) + O(n+m) = O(n+m)