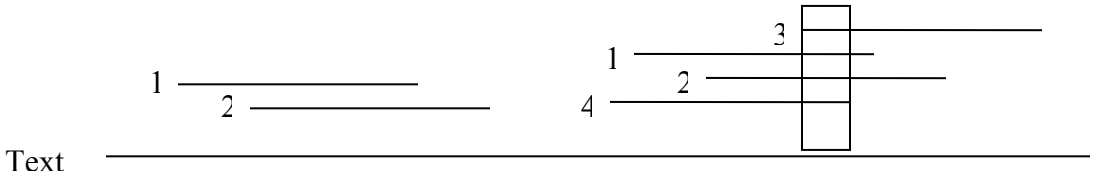
1. It is possible to incorporate the bad character shift rule from the Boyer Moore method to the KMP method.
    a. Show how to do that.
    b. Evaluate how efficient this rule is and explain why it is more effective in the Boyer-Moore algorithm.

2. Assume we used the Aho-Corasick algorithm to match a set of k strings against a text T of length n. The output of our program produced, for each of the k strings, a list of coordinates representing the locations in the text where these strings match. The total number of matches will be denoted by m.
    a. Describe an algorithm running in O(m log m) time that identifies the location in the text covered by the largest number of patterns. Specifically, report the coordinates i and j in the text such that the substring T[i..j] is contained within more pattern matches than any other substring of the text.
    b. Can your algorithm also report which patterns match at this location, under the same time constraints? We are only interested in reporting the matching patterns for the first "maximally covered" substring in case more than one substring follows the description above.
    c. Assume we want to identify a place in the text covered by **all** k patterns. Can you describe a method for pre-processing the set of patterns that will allow you to quickly home into areas of the text where all patterns match? Also describe how you can use this to reduce the amount of work performed by the Aho-Corasick algorithm. (Here I'm looking for the big picture, not the full details of an algorithm. What tools (e.g. algorithms we've learned about) will you use? How will you put them together to get the answer?)

HINT: this question has nothing to do with the Aho-Corasick algorithm
HINT: assume that for any two strings there's only one "best" way in which they match each other.
Note: the figure below might be useful also. You are trying to identify the region highlighted with a box. Note that some patterns may match multiple places.



3. In the class we showed how to use a suffix tree for a small string P to compute matching statistics ms(i) for each position i in the long text string T. Suppose we also want to compute matching statistics ms(j) for each position j in P. Number ms(j) is defined as the length of the longest substring starting at position j in P that matches some substring in T. Show how to find all the matching statistics for both T and P in O(|T|) time, using only a suffix tree for P.

4. The least common ancestor of two nodes in a tree is the lowest node shared by the paths from the two nodes to the root. Assume n is the least common ancestor of nodes i and j in a suffix tree for string S.
    a. What does this node represent?
    b. Describe an algorithm that will compute the Z values for S, using the suffix tree.

Note: for any location i in S, Z[i] is the longest prefix of S[i..n] that matches a prefix S.

5. Suppose we have sequences $v = v_1, \ldots, v_n$ and $w = w_1, \ldots, w_m$, where v is longer than w. We wish to find a substring of v which best matches all of w. Global alignment won't work because it will try to align all of v. Local alignment won't work because it may not align all of w. The problem (called the fitting problem) can be formulated as the problem of finding a substring v' of v that maximizes the score of alignment s(v', w) among all substrings of v. Give an algorithm which computes the optimal fitting alignment in O(nm) time. Describe both how to compute the score of this alignment and how to compute the actual alignment.

Note: You don't need to spell out all the details of the algorithm. If you use the traditional dynamic programming approach, it's sufficient to specify the initial conditions (what values do you add to the first row/column of the matrix), where you will the score for this best alignment be located in the matrix, and whether you use the "traditional" recurrence in the algorithm, or the recurrence that allows a 0, in addition to the three possible edits.