**CSC 325 Adv Data Structures**

**Homework #3**

**KMP and Ropes**

Name: \_**Answer Key**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (8 pts) Perform the KMP algorithm using the following text and pattern:

Text = “hohonohohohoyeah”

Pattern = “hohoho”

First, fill in the LPS table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pattern | h | o | h | o | h | o |
| longest prefix | 0 | 0 | 1 | 2 | 3 | 4 |

Now show how the algorithm would attempt to match the pattern in the text using the LPS (just like in the notes). I’ve started things off for you. Show the characters being matched with an underline in the text and pattern. When you get a mismatch, put an X over the character that mismatched in the text and pattern. Then go to the next table and show where the pattern jumps to and which character it starts the matching at. Underline the characters in the text and pattern that are being matched and again draw an ‘X’ around the mismatched character that causes the algorithm to jump again. Do not underline characters that would not be examined by the algorithm as a potential match. Finally, after it successfully matches the pattern, circle the index that would be returned if we were using this as part of the indexOf function. Note: if one or more tables aren’t needed (i.e. the pattern has already been found), leave them blank.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern | h | o | h | o | h | o |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern |  |  | h | o | h | o | h | o |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern |  |  |  |  | h | o | h | o | h | o |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern |  |  |  |  |  | h | o | h | o | h | o |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern |  |  |  |  |  |  | h | o | h | o | h | o |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| text | h | o | h | o | n | o | h | o | h | o | h | o | y | e | a | h |
| pattern |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

2. (4 pts) Given that n is the size of the text and m is the size of the pattern, what is the time complexity (in Big Oh) of the brute force algorithm to pattern matching? What is the time complexity of the KMP algorithm? You may need to research this online but be careful as you may have to look it up in multiple places to get the right answer. Hint: KMP is NOT just O(n). Clearly state which time complexity goes with which algorithm.

KMP: O(n + m)

Brute Force: O(n \* m)­

3. (8 pts) Assume a rope data structure is used behind the scenes when creating strings (like in the notes). Draw **all** ropes after each line of code and show all string pointers (s1, s2, s3, etc) next to the node they point to. Assume there is no max character limit to the strings in our nodes. Make sure to show the number in each node and for leaf nodes, show the string it contains underneath it (as in the notes). For calls to the charAt function, double circle ◎ the nodes that would be traversed, showing the resulting search index as we go down the tree (just like in the notes). Circle the resulting char that would be found at the index. The first line of code has been done for you:

String s1 = "a";



String s2 = "bc";

String s3 = "def"; // continue to show all created ropes

String s4 = "g";

A white circles with black background

Description automatically generated

String s5 = s1 + s2;

String s6 = s3 + s5;

// show all

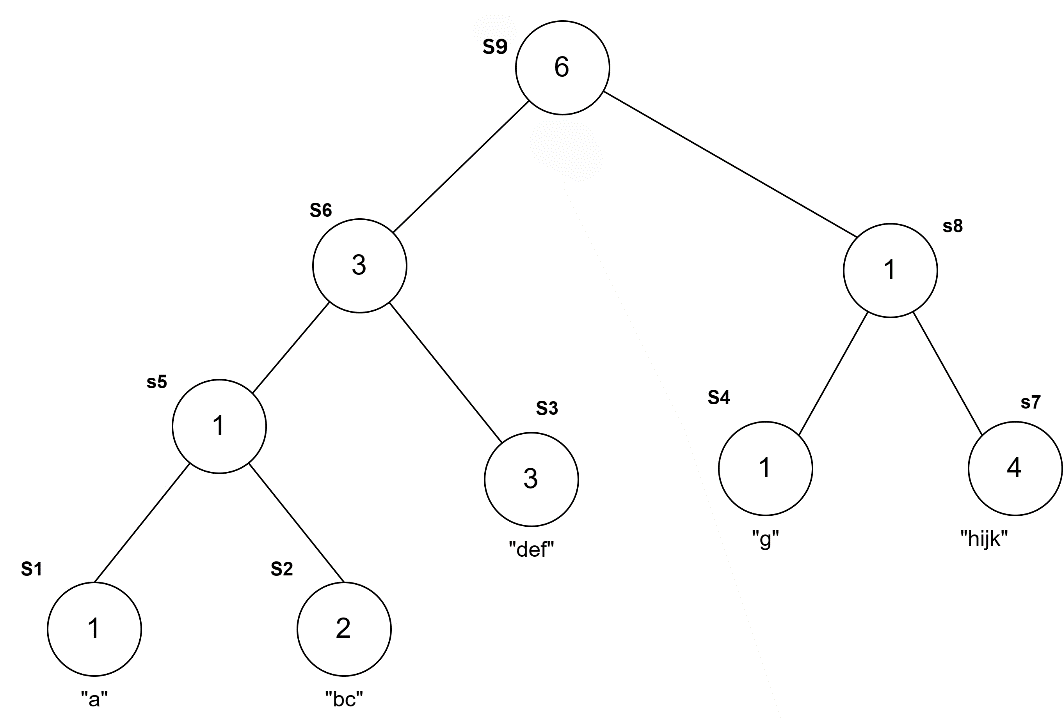
A black background with white circles

Description automatically generated

String s7 = ”hijk”;

String s8 = s4 + s7;

String s9 = s6 + s8;



System.out.println(s5); // what would this print to the screen?

abc

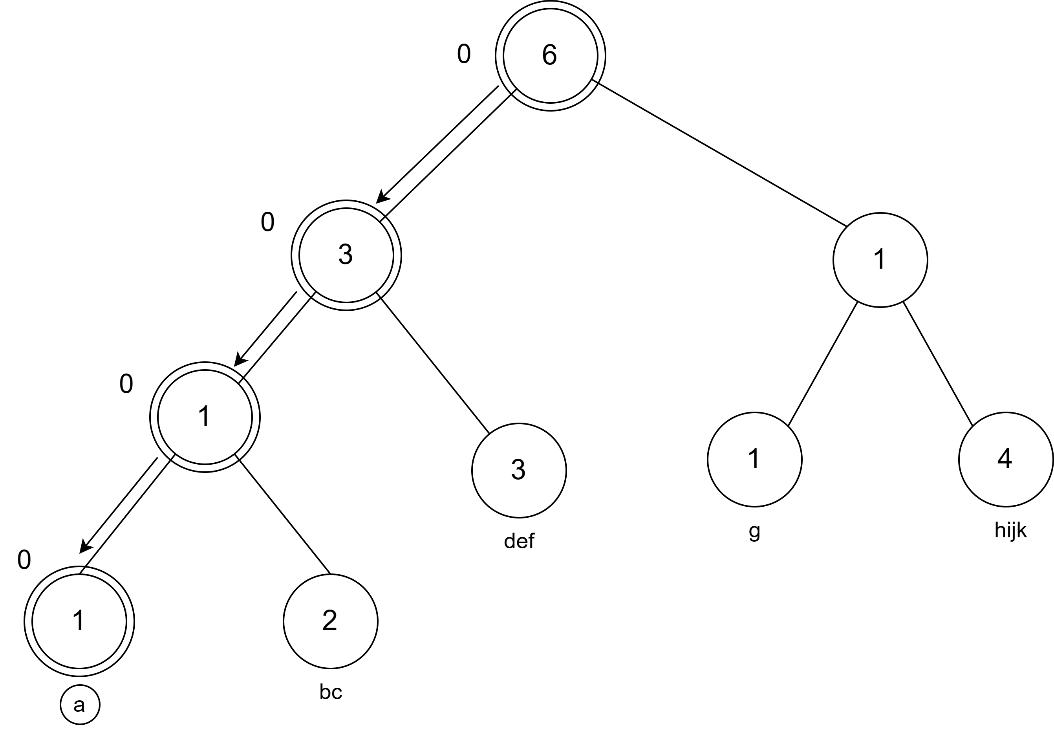
System.out.println(s8); // what would this print to the screen?

ghijk

System.out.println(s9); // what would this print to the screen?

abcdefghijk

s9.charAt(0);



s9.charAt(4);

A black background with white circles

Description automatically generated

s9.charAt(6);

A black background with white circles

Description automatically generated