Theory: Knuth-Morris-Pratt algorithm in Java

🕐 54 minutes   0 / 5 problems solved

Skip this topic   Start practicing

The Knuth-Morris-Pratt algorithm is an approach that allows solving the substring searching problem in linear time in the worst case. Similarly to the naive one, the algorithm performs a symbol-by-symbol comparison of a pattern with each substring of a text. To reduce the number of comparisons, the algorithm uses the prefix function to identify an optimal pattern shift in case of symbols mismatch. In this topic, we will learn how to implement this algorithm in Java.

§1. Implementing a method for calculation of the prefix function

First, let's see how we can implement a method for calculation of the prefix function:

```java
public static int[] prefixFunction(String str) {
    /* 1 */
    int[] prefixFunc = new int[str.length()];

    /* 2 */
    for (int i = 1; i < str.length(); i++) {
        /* 3 */
        int j = prefixFunc[i - 1];

        while (j > 0 && str.charAt(i) != str.charAt(j)) {

            j = prefixFunc[j - 1];

        }


        /* 4 */

        if (str.charAt(i) == str.charAt(j)) {

            j += 1;

        }


        /* 5 */

        prefixFunc[i] = j;

    }



    /* 6 */

    return prefixFunc;

}
```

The method takes a string `str` as an argument and returns the prefix function it. The steps of the method are:

1. Create an array of length `str.length()` for storing the prefix function. Recall that by convention the first value of the array should be equal to zero. Since in Java an array of ints is initialized with zeroes by default, we needn't care about it.

Current topic:

Knuth-Morris-Pratt
algorithm in Java   ...

Topic depends on

✕   Knuth-Morris-Pratt
    algorithm   ...

✕   Searching a substring in
    Java   ...

Table of contents:

2. Calculate the values of the prefix function for each substring of `str` in the `for` loop. At each iteration, we try to find the longest border of the previous substring that can be extended by the current symbol.

3. Initially, the variable `j` is equal to the length of the previous longest border. Then, in the `while` loop, we consider the next longest borders until an appropriate extension is found or while all borders are examined. If the first previous border can be extended, the `for` loop will be omitted.

4. Check if the current border indeed can be extended and if it is so, increase its length.

5. Assign the length of the current border to the current substring.

6. Return the `prefixFunc` array as a final result.

§2. Implementing the Knuth-Morris-Pratt algorithm

Using the `prefixFunction` method, the Knuth-Morris-Pratt algorithm can be implemented as follows:

```
1    public static List<Integer> KMPSearch(String text, String pattern) {
2        /* 1 */
3        int[] prefixFunc = prefixFunction(pattern);
4        ArrayList<Integer> occurrences = new ArrayList<Integer>();
5        int j = 0;
6        /* 2 */
7        for (int i = 0; i < text.length(); i++) {
8            /* 3 */
9            while (j > 0 && text.charAt(i) != pattern.charAt(j)) {
10               j = prefixFunc[j - 1];
11           }
12           /* 4 */
13           if (text.charAt(i) == pattern.charAt(j)) {
14               j += 1;
15           }
16           /* 5 */
17           if (j == pattern.length()) {
18               occurrences.add(i - j + 1);
19               j = prefixFunc[j - 1];
20           }
21       }
22       /* 6 */
23       return occurrences;
24   }
```

The method takes two strings, a `pattern` and a `text`, as arguments and returns a list of all occurrences of the pattern in the text. The steps of the method are:

1. Calculate the prefix function for the pattern and create a list for storing all occurrences. Also, create a variable `j` to store the index of the current symbol of the pattern.

2. Perform a symbol-by-symbol comparison of the pattern with the current substring of the text using the `for` loop.

3. If the corresponding symbols are not matched, try to find the longest prefix of the pattern that can be extended by the current symbol of the text using the precalculated prefix function.

4. If the corresponding symbols are matched, increase the value of `j`, that is, move to the next symbol of the pattern.

5. if the value of `j` is equal to the length of the pattern, an occurrence is found. We need to add the corresponding index to the list of all occurrences. In this case, the index of the pattern symbol to be considered at the next iteration is equal to `prefixFunc[j-1]` since we know that the previous ones already match.

6. Finally, we return the list of all occurrences.

§3. Examples

Below are several examples of how to use the method:

```
1   List<Integer> occurrences = KMPSearch("ABACABAD", "ABA");
2   System.out.println(occurrences); // [0, 4]
3
4   List<Integer> occurrences = KMPSearch("ABABA", "ABA");
5   System.out.println(occurrences); // [0, 2]
```

§4. Summary

In this topic, we have learned how to implement the Knuth-Morris-Pratt algorithm in Java. We first have implemented a method for finding the prefix function, and then have applied this method to find all occurrences of a string in another string. Now, you can use this approach to efficiently solve the substring searching problem.

11 users liked this theory. **0** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

This content was created about 1 year ago and updated 11 days ago. Share your feedback below in comments to help us improve it!

💬 Show discussion (3)