Theory: Edit distance in Java

33 minutes 0 / 5 problems solved

Edit distance is a metric that allows estimating the similarity between two strings of different sizes. The metric is defined as the minimum number of insertions, deletions, or substitutions required to transform one string into the other. In this topic, we will learn how an algorithm for calculating edit distance can be implemented in Java.

Current topic:

Edit distance in Java

Topic depends on:

✕ Edit distance

✕ Algorithms in Java

Topic is required for:

Edit distance alignment in Java

§1. Implementation in Java

An algorithm for calculation of edit distance can be implemented using the dynamic programming technique. We can start the calculation of edit distance from smaller prefixes of both strings (divide the problem into smaller subproblems in terms of dynamic programming), then use the answers for these subproblems to find edit distance for bigger prefixes and finally get the answer for the initial strings.

Below is an implementation of this algorithm in Java:

```
public static int match(char a, char b) {
    return (a == b) ? 0 : 1;
}

public static int editDistance(String s, String t) {
    int[][] d = new int[s.length() + 1][t.length() + 1];

    for (int i = 0; i < s.length() + 1; i++) {
        d[i][0] = i;

    }



    for (int j = 0; j < t.length() + 1; j++) {

        d[0][j] = j;

    }



    for (int i = 1; i < s.length() + 1; i++) {

        for (int j = 1; j < t.length() + 1; j++) {

            int insCost = d[i][j - 1] + 1;

            int delCost = d[i - 1][j] + 1;

            int subCost = d[i - 1]
[j - 1] + match(s.charAt(i - 1), t.charAt(j - 1));

            d[i][j] = Math.min(Math.min(insCost, delCost), subCost);

        }

    }



    return d[s.length()][t.length()];

}
```

The `editDistance` method takes two strings `s` and `t` as arguments and returns the edit distance for the strings.

1. Initially, we create an array `d` to store intermediate answers.

2. Then, we fill in the first row and the first column of the array (recall that they correspond to the transformation of an empty string into `s` or `t`).
3. Next, we start successively calculating the edit distance for each pair of prefixes of `s` and `t`.
4. When all prefixes are processed, the lower-right cell of the array contains the edit distance for the initial strings and we return it as a final result.

§2. Usage example

Here is an example of how the implemented method can be used:

```
int editDist = editDistance("editing", "distance");
System.out.println(editDist); // 5
```

According to the output, the minimum number of edit operations required to transform the string `editing` to the string `distance` is 5.

§3. Summary

In this topic, we have learned how to implement an algorithm for calculation of edit distance in Java and have considered how to use the implemented method. Note that there exist various modification of this algorithm that allows solving more sophisticated problems connected with strings comparison. Some of such modifications will be suggested to you as a programming exercise. So move on to practice and try to solve them!

Report a typo

15 users liked this theory. **3** didn't like it. **What about you?**

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion