121 users solved this topic. Latest completion was about 19 hours ago.

Theory: Edit distance alignment in Java

55 minutes 0 / 5 problems solved

Skip this topic Start practicing

An algorithm for finding the edit distance between two strings calculates the minimum number of operations required to transform one string into the other. One limitation of this algorithm is that it returns only the number of such operations. In some cases, however, we would like to know the exact sequence of operations required to turn one string into the other.

A structure that contains information about the sequence of such operations is called an alignment. In this topic, we will learn how an algorithm for finding an alignment for two strings can be implemented in Java.

§1. Implementation in Java

An algorithm for finding an alignment between two strings s and t consists of two steps. First, using the dynamic programming approach, we need to construct a 2d-table that contains the edit distance for each pair of prefixes of s and t. Then, we need to reconstruct edit operations. To do that, we need to move from the lower-right cell of the table (that contains the edit distance for the initial strings) to the upper-left cell choosing at each step the direction that corresponds to the optimal transformation. Considering this sequence of transformation in the reverse order we will get an alignment for the two strings.

Now, let's see how this algorithm can be implemented in Java. To store an alignment, we will use the following class:

```java
class Alignment {
    public int editDistance;
    public String source;
    public String target;

    public Alignment(int editDist, String source, String target) {
        this.editDistance = editDist;
        this.source = source;
        this.target = target;

    }

}
```

The class stores two strings named `source` and `target` that correspond to alignment, and an integer variable named `editDistance` that store the edit distance between the strings.

Using this class, the first step of the algorithm can be implemented as follows:

Current topic:

Edit distance alignment in Java

Topic depends on:

Edit distance alignment

Edit distance in Java

Table of contents:

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

public static Alignment editDistanceAlignment(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 reconstructAlignment(d, s, t);
}
```

The method named `editDistanceAlignment` takes two strings as arguments and returns an alignment for them. The method works as follows:

1. First, we create a 2d-table to store the edit distance for prefixes of the strings.
2. Then, we initialize the first row and the first column of the table, and after that start filling the table row by row. Then the `for` loops are completed, the lower right cell of the table contains the edit distance for the strings.
3. Finally, we use the `reconstructAlignment` method that finds an alignment for the strings. An output of this method is returned as a final result.

An implementation of the `reconstructAlignment` method is the following:

```java
public static Alignment reconstructAlignment(int[][] d, String s, String t) {
    StringBuilder ssBuilder = new StringBuilder();
    StringBuilder ttBuilder = new StringBuilder();
    int i = s.length();
    int j = t.length();

    while (i > 0 || j > 0) {
        if (i > 0 && j > 0 && d[i][j] == d[i - 1][j - 1] + match(s.charAt(i - 1), t.charAt(j - 1))) {
            ssBuilder.append(s.charAt(i - 1));
            ttBuilder.append(t.charAt(j - 1));
            i -= 1;
            j -= 1;
        } else if (j > 0 && d[i][j] == d[i][j - 1] + 1) {
            ssBuilder.append("-");
            ttBuilder.append(t.charAt(j - 1));
            j -= 1;
        } else if (i > 0 && d[i][j] == d[i - 1][j] + 1) {
            ssBuilder.append(s.charAt(i - 1));
            ttBuilder.append("-");
            i -= 1;
        }
    }

    String ss = ssBuilder.reverse().toString();
    String tt = ttBuilder.reverse().toString();

    return new Alignment(d[s.length()][t.length()], ss, tt);
}
```

The method takes a table of intermediate answers and two strings s and t as arguments and reconstructs the edit operations for the strings using the table. The steps of the method are:

1. First, we create two `StringBuilder` objects to store an alignment for the strings. The variables i and j are used to store the current position in the table. Initially, i and j point to the lower right cell of the table.
2. Then, on each iteration of the `while` loop, we move to the direction corresponding to the optimal transformation. First, we check whether it is possible to move on the diagonal (which corresponds to a substitution). If it is, we append the corresponding symbols of the strings to the current alignment and decrease both i and j by one. If it is not, we check whether we might get the current value from the left cell of the table (which corresponds to a deletion). If it is the case, we append the corresponding symbols to the current alignment and decrease the value of j by one. If it's not the case, we move to the upper cell of the table (which corresponds to an insertion).
3. Then the left upper cell of the table is reached (i and j both equal to zero) we reverse the edit operation and return the alignment for the strings.

§2. Usage examples

Given below is an example of how the `editDistanceAlignment` method can be used:

```
Alignment alignment = editDistanceAlignment("editing", "distance");
System.out.println(alignment.editDistance);
System.out.println(alignment.source);
System.out.println(alignment.target);
```

The code prints the following:

```
5
edi-tin-g
-distance
```

Thus, the edit distance between the strings `editing` and `distance` is 5. The alignment gives us the exact sequence edit operations that should be applied to transform one of the strings into the other.

§3. Summary

In this topic, we have learned how an algorithm for finding an alignment for two strings can be implemented in Java and have considered an example of how to use the implemented method. Note that this algorithm is a basic one, and it is often used as a basis for other more sophisticated string-processing algorithms. Some of such algorithms will be suggested to you as programming assignments. So move on to practice and try to solve them!

Report a typo

12 users liked this theory. 1 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

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