Technical Report: Algorithm for Automatic Syntax Error Correction in Programming Languages

Introduction

Ponoran Claudiu Mihai CEN 1.2B
Faculty of Automatics, Computer Science and Electronics
-
May 2024
-

The source code:

https://github.com/Claudiu201/Algorithms-Design.git

Problem Statement

In modern software development, ensuring that code adheres to specified syntax rules is crucial for maintaining code quality and reducing errors. This report presents an algorithm designed to automatically correct syntax errors in code fragments. The algorithm determines the minimum number of operations (substitutions, insertions, or deletions) required to transform an erroneous code fragment into one that complies with a given syntax rule.

Problem Definition

Given:
A valid syntax rule for a specific programming language (e.g., "func(myFunction)").
A code fragment that contains syntax errors (e.g., "fnuc(myFuncion").

Objective:
To compute the minimum number of operations needed to convert the erroneous code fragment into one that matches the valid syntax rule.

Dynamic Programming Approach

The problem is analogous to the edit distance problem, where we aim to find the minimum number of edits required to convert one string into another. We use a dynamic programming (DP) approach to solve this problem efficiently.

DP Table Construction

We define a 2D DP table dp where dp[i][j] represents the minimum number of operations required to transform the first i characters of the code fragment into the first j characters of the valid syntax. The table is filled based on the following rules:

Initialization:

dp[0][0] = 0: No operations are needed to transform an empty string into an empty string.
dp[i][0] = i: i deletions are required to transform the first i characters of the code fragment into an empty string.
dp[0][j] = j: j insertions are required to transform an empty string into the first j characters of the valid syntax.

Filling the DP Table:

If the characters match (code_fragment[i-1] == valid_syntax[j-1]), then dp[i][j] = dp[i-1][j-1].
If they don't match, consider the minimum cost among insertion, deletion, and substitution:
dp[i][j] = min(dp[i-1][j] + 1, dp[i][j-1] + 1, dp[i-1][j-1] + 1).

Algorithm Implementation in C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Function to find the minimum of three numbers
int min(int a, int b, int c) {
    if (a < b && a < c) return a;
    if (b < a && b < c) return b;
    return c;
}

// Function to calculate the minimum number of operations to correct the syntax
int min_operations_to_correct_syntax(const char *code_fragment, const char
*valid_syntax) {
    int m = strlen(code_fragment);
    int n = strlen(valid_syntax);

    // Allocate memory for the DP table
    int **dp = (int **)malloc((m + 1) * sizeof(int *));
    for (int i = 0; i <= m; i++) {
        dp[i] = (int *)malloc((n + 1) * sizeof(int));
    }
    // Initialize the DP table
    for (int i = 0; i <= m; i++) {
        dp[i][0] = i;  // Deletion
    }
    for (int j = 0; j <= n; j++) {
        dp[0][j] = j;  // Insertion
    }

    // Fill the DP table
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (code_fragment[i - 1] == valid_syntax[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                dp[i][j] = min(dp[i - 1][j] + 1,       // Deletion
                               dp[i][j - 1] + 1,       // Insertion
                               dp[i - 1][j - 1] + 1);  // Substitution
            }
        }
    }

    // Get the result from the DP table
    int result = dp[m][n];

    // Free the allocated memory
    for (int i = 0; i <= m; i++) {
        free(dp[i]);
    }
    free(dp);

    return result;
}

int main() {
    const char *code_fragment = "fnuc(myFuncion";
    const char *valid_syntax = "func(myFunction)";
    int result = min_operations_to_correct_syntax(code_fragment, valid_syntax);
    printf("Minimum number of operations: %d\n", result);
    return 0;
}
```

Explanation:

1. **Helper Function min:**

    - Returns the minimum of three integers.

2. **Function min_operations_to_correct_syntax:**

    - Takes two input strings: code_fragment and valid_syntax.
    - Initializes a 2D array dp to store the minimum number of operations required to transform substrings of code_fragment into substrings of valid_syntax.
    - Fills the dp array using the rules described above.
    - Returns the value at dp[m][n], which represents the minimum number of operations required.

3. **Main Function:**

    - Example usage with a given code fragment and valid syntax.
    - Prints the minimum number of operations required to correct the syntax.

Conclusion:

This report presents a robust algorithm to correct syntax errors in code fragments using a dynamic programming approach. The provided C implementation demonstrates the practical application of the algorithm, efficiently calculating the minimum number of operations required to transform an erroneous code fragment into a valid one according to a specified syntax rule. This technique can be extended to handle more complex syntax rules and integrated into advanced code editors to enhance their error-correcting capabilities.