Proiectarea Algoritmilor
Tema de casa
Distanta intre doua siruri

Chiriac Adrian-Florin

Mai 2024

1 Introducere

1.1 Enuntul problemei

We have the task of developing an algorithm for an advanced code editor that automatically corrects syntax errors in programming lan- guages. It is assumed that we receive a clear specification of the valid syntax of the programming language in the form of a "rule" and a code fragment that contains syntax errors, i.e., does not conform to that rule.

Our objective is to build an algorithm that determines the mini- mum number of operations required to transform the code fragment

into one that complies with the given rule. These operations may include character substitutions, insertions, or deletions.

Let's take a concrete example to illustrate the problem: let's as- sume we have the following syntax rule for function declarations in

the programming language: "Every function must start with the keyword "func", followed by the function name enclosed in parentheses." An example of a valid function declaration would be "func(myFunction)". Here's how the situation looks: Given code fragment: "fnuc(myFuncion"

Our objective is to find the minimum number of operations re- quired to correct the code fragment so that it matches the pattern

defined by the rule. These operations may include, for example, re- versing the characters "n" and "u" to obtain "func", then inserting

the missing characters "t" and ")", so that we obtain "func(myFunc)" according to the given rule.

1.2 Descrierea problemei

We're given two strings and we have to determine the minimum number of operations that it takes to transform the second string into the rule string (valid one). The operations we're able to make are:

- Substitute a character
- Delete a character
- Insert a character
- Reverse two characters

2 Algoritmi

2.1 Pseudocod

```
DECLARE rule AS STRING
DECLARE code_to_change AS STRING
DECLARE n AS INTEGER
DECLARE m AS INTEGER
```

```
DECLARE dp[105][105] AS INTEGER
FUNCTION MinOfTwo(integer a, integer b)
IF a>b THEN:
RETURN a
else
RETURN b
END IF
END FUNCTION

FUNCTION MinimumNoOperations( string rule , string code_to_change, integer dp[105][105], integer n, int
FOR i = 0 TO n+1 DO
dp[i][0]=i
END FOR
FOR j = 0 TO m+1 DO
dp[0][j]=j
END FOR
FOR i = 0 TO n DO
    FOR j=0 TO m DO
IF rule[i]=code_to_change[j] THEN:
dp[i+1][j+1]=dp[i][j]
ELSE
integer insert_cost=dp[i+1]+1
integer delete_cost=dp[i][j+1]+1
integer replace_cost=dp[i][j]+1
IF rule[i-1]==code_to_change[j] and rule[i]==code_to_change[j-1] THEN:
swap_cost=dp[i-1][j-1]+1
replace_cost=MinOfTwo(replace_cost,swap_cost)
END IF
dp[i+1][j+1]=MinOfTwo(MinOfTwo(replace_cost,insert_cost),delete_cost)
END IF
END FOR
END FOR
RETURN dp[n][m]
END FUNCTION

READ rule
READ code_to_change

FOR i = 0 TO 100 DO
FOR j=0 TO 100 DO
dp[i][j]=0
END FOR
END FOR


n = length(rule)
m = length(code_to_change)
integer nr = MinimumNoOperations(rule,code_to_change,dp,n,m)
PRINT "Minimum number of operations required to transform the second string to a valid one is equal to:
PRINT nr
```

3 Descrierea aplicatiei

3.1 Utilizare

The application functions in 2 working regimes:

- User Input
- Between 2 and 5 randomly chosen words from a list

1) For the first regime, the user is prompted with the question "Input the rule string:" and after that "Input the fragment code:". (For example, the rule string is "func(myFunction)" and the fragment code is "fnuc(myFuncion"). The program is going to output "3", because there are 3 steps to accomplish this which are explained in the document:

- Reversing the characters "n" and "u"
- Adding the letter "t"
- Adding the character ")"

Therefore there are 3 operations.

2) In the second regime, the program takes its data from a .txt file using file pointer. In the .txt file there are 1000 randomly generated words. The program then chooses a random number between 2 and 5 (this will be the number of words in the string). The individual words are then randomly chosen with an "index=rand()MOD1000" and concatened to the rule string. The code fragment is then generated by a "similar string generator" that we will explain in a different section to make sure all the details are clear.

2*) SimilarStringGenerator Function:

Explanation of the `similar_string_generator` Function

Summary:

The function creates a string similar to the input string but with a random number of errors (between 1 and the length of the string). These errors can be character swaps, deletions, or replacements. This is useful for testing automatic syntax error correction algorithms by simulating different types of errors that can occur in a character string.

Step-by-step Explanation:

1. `size_t length = strlen(source);`
   Calculate the length of the source string.
2. Allocate memory for the similar string:

```
    char *similarString = malloc(length + 1);
    if (!similarString) {
        return NULL;
    }
```

3. Copy the source string into the result string:

```
    strcpy(similarString, source);
```

4. Generate the number of errors:

```
    int num_errors = (rand() % length) + 1;
```

5. Introduce the errors:

```
    for (int i = 0; i < num_errors; i++) {
        int error_type = rand() % 3;
        int pos = rand() % length;
        if (error_type == 0 && pos < length - 1) {
            char temp = similarString[pos];
            similarString[pos] = similarString[pos + 1];
```

```
            similarString[pos + 1] = temp;
        } else if (error_type == 1 && length > 1) {
            memmove(&similarString[pos], &similarString[pos + 1], length
                - pos);
            similarString[length - 1] = '\0';
            length--;
        } else if (error_type == 2) {
            similarString[pos] = 'a' + (rand() % 26);
        }
    }
```

Iterate `num_errors` times to introduce errors into the similar string:

- **Swap**: If `error_type` is 0 and `pos` is less than `length-1`, swap two adjacent characters.
- **Deletion**: If `error_type` is 1 and the string length is greater than 1, delete the character at position `pos` and shift the characters to the right of it one position to the left, reducing the string length.
- **Replacement**: If `error_type` is 2, replace the character at position `pos` with a random character between 'a' and 'z'.

6. Return the modified string:

```
    return similarString;
```

4 Rezultate

```
Edit Distance w/ swap:

Enter your choice
1 for user input
2 for randomly chosen words
3 for exiting the program

1

Please type the rule string:          func(myFunction)
Please type the user input string:    fnuc(myFuncion
Minimum number of operations required: 3
```

Figure 1: The first example

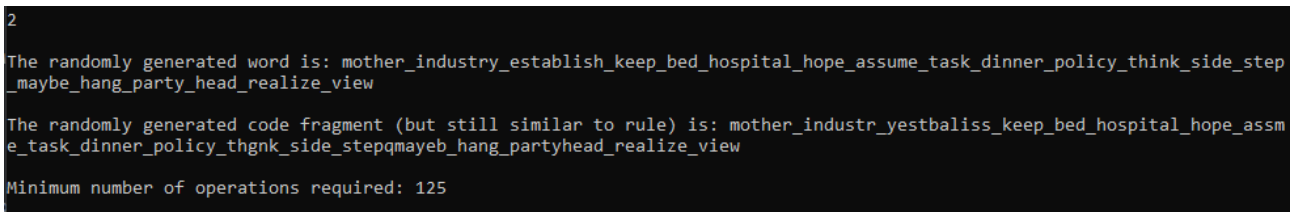

Figure 2: The second example

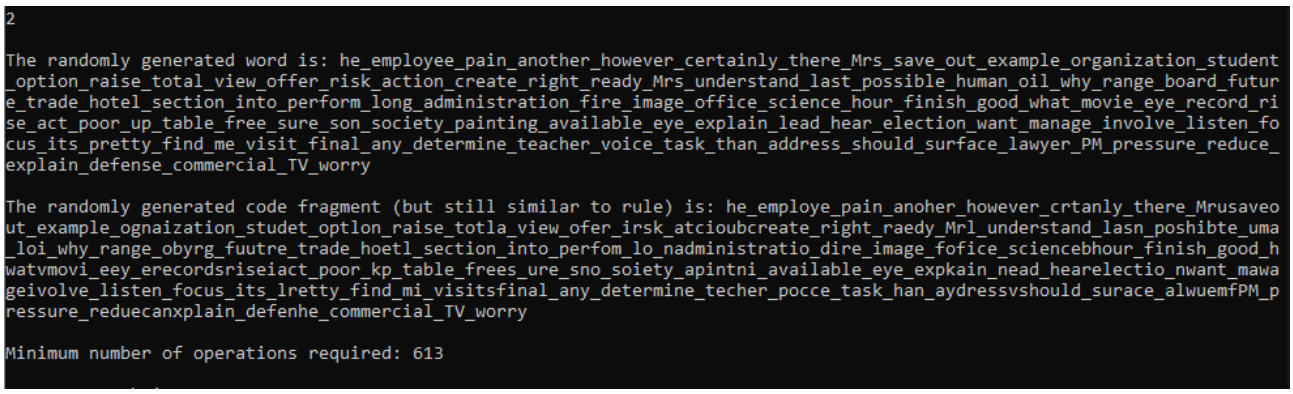

Figure 3: The third example

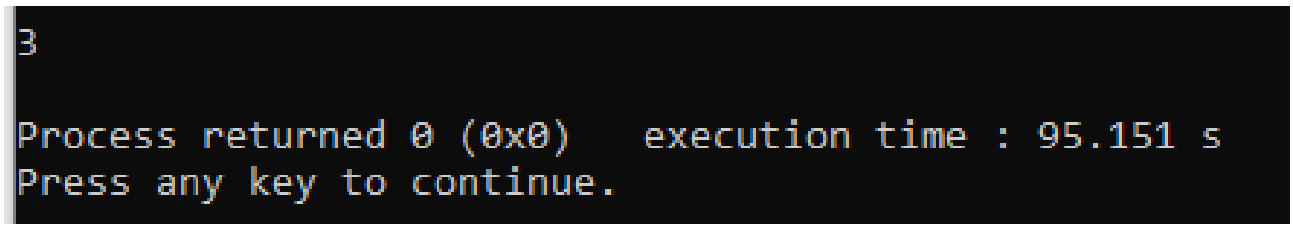

Figure 4: The 4th example (Making sure the program closes properly)

5 Algoritmul lui Levenshtein Personalizat

We will use the Levenshtein algorithm from the famous "Edit Distance" problem, where there are only three operations (the inversion operation does not exist). We will use a dp matrix (an idea based on dynamic programming) that works as follows:

- If two characters are the same, we do nothing.
- If the two characters are different, we have four operations: deletion, insertion, substitution, inversion.

Imagine that the rows represent the first string, and the columns represent the second string. If you want to delete a character, you go back to the previous column (`dp[i][j-1]`). If you want to add a letter, you go back to the previous row (`dp[i-1][j]`). If you want to substitute the last character in both strings, you go back to `dp[i-1][j-1]`. If you want to perform an inversion, you need to go to `dp[i-2][j-2]`.

In the end, to reach `dp[i][j]`, you need to perform the minimum of these four operations and add one.

Example: Customized Levenshtein Algorithm (with inversion operation)

Target string: `func(myFunction)`
Code fragment: `fnuc(myFuncion)`

To transform the code fragment into the target string, the following operations are needed:

1. Invert "n" and "u".
2. Add the letter "t".
3. Add the character ")".

So, in total, there are 3 operations.

Let n be the length of the target string, which is 16, and m be the length of the code fragment, which is 14. Our solution is `dp[n][m]`, which is `dp[16][14]`, and its value is 3. This verifies the algorithm.

6 Program C

The program is composed by 3 files:

- The main file
- The file that generates the strings
- The header file which calls the functions

6.1 Main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "choosewords.h"
#define DIM 200000
#include <string.h>
char rule[DIM], input[DIM];
char *fragment_cod;
int dp[DIM][DIM], distance;

int Min(int a, int b)
{
    return a>b ? b : a;
}

int MinimumNoOperations(char rule[DIM], char code_fragment[DIM], int dp[DIM
    ][DIM], int n, int m)
{
    for(int i=0; i<=n; i++)
```

```
    {
        dp[i][0]=i;
    }
    for(int j=0; j<=m; j++)
    {
        dp[0][j]=j;
    }

    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
        {
            if(rule[i]==code_fragment[j])
                dp[i+1][j+1]=dp[i][j];
            else
            {
                int insert_cost=dp[i+1][j]+1;
                int delete_cost=dp[i][j+1]+1;
                int replace_cost=dp[i][j]+1;

                if(rule[i-1]==code_fragment[j] && rule[i]==code_fragment[j
                    -1])
                {
                    int swap_cost=dp[i-1][j-1]+1; // bonus operation
                        compared to the original "Edit Distance" problem
                    replace_cost=Min(replace_cost,swap_cost);
                }

                dp[i+1][j+1]=Min(Min(insert_cost,delete_cost), replace_cost)
                    ;
            }
        }
    }
    return dp[n][m];
}

int main ()
{

    printf("Edit Distance w/ swap:\n");

    while(1)
    {
        printf("\nEnter your choice\n");
        printf("1 for user input\n");
        printf("2 for randomly chosen words\n");
        printf("3 for exiting the program\n\n");

        int choice;
        scanf("%d",&choice);

        switch (choice)
        {
        case 1:
            printf("\n");
            getchar();
            printf("Please type the rule string:        ");
            fgets(rule, DIM, stdin);
            rule[strlen(rule)-1]=0;

            printf("Please type the user input string:  ");
```

```
78              fgets(input, DIM, stdin);
79              input[strlen(input)-1]=0;
80
81              int len_input = strlen(input);
82              int len_rule = strlen(rule);
83
84              distance = MinimumNoOperations(rule, input, dp, len_rule,
                    len_input);
85
86              printf("Minimum number of operations required: %d\n", distance);
87              memset(input,0,DIM);
88              memset(rule,0,DIM);
89
90              break;
91
92          case 2:
93              getchar();
94              printf("\n");
95
96              char **words= allocate2DArray(1000, 50);
97              if(words==NULL)
98              {
99                  exit(4);
100             }
101
102             choose_random_word(words, rule);
103             printf("The randomly generated word is: ");
104             printf("%s",rule);
105             printf("\n\n");
106             int n=strlen(rule);
107             fragment_cod=similar_string_generator(rule);
108             printf("The randomly generated code fragment (but still similar
                    to rule) is: ");
109             printf("%s",fragment_cod);
110             printf("\n\n");
111             int m=strlen(fragment_cod);
112
113             distance = MinimumNoOperations(rule, input, dp, n, m);
114             printf("Minimum number of operations required: %d\n", distance);
115
116             free2DArray(words, 1000);
117             memset(input,0,DIM);
118             memset(rule,0,DIM);
119
120             break;
121
122         case 3:
123             return 0;
124             break;
125
126         default:
127             printf("\nInvalid choice!\n");
128             break;
129         }
130
131     }
132     return 0;
133 }
```

6.2 Header

```
#ifndef CHOOSEWORDS_H
#define CHOOSEWORDS_H

void choose_random_word(char **words, char rule[]);
char *similar_string_generator(char *source);
char** allocate2DArray(int rows, int cols);
void free2DArray(char **array, int rows);
#endif // CHOOSEWORDS_H
```

6.3 Generator File

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include "choosewords.h"
#define WORD_COUNT 10000

void choose_random_word(char **words, char rule[])
{


    FILE *samplewords = fopen("wordlist.txt","r");
    if( samplewords == NULL)
    {
        printf("the file could not be opened!");
        exit(1);
    }

    int i=0;
    while(fgets(words[i],40,samplewords)!= NULL)
    {
        words[i][strlen(words[i])-1]=0;
        i++;
    }

    int howmanywords=i;

    /*
    printf("iata cateva cuvinte!\n");
    for(i=0; i<howmanywords; i++)
    {
        printf("%s\n",words[i]);
    }
    */

    srand(time(NULL)*getpid());

    //0 sa aleg intre 2 si 5 cuvinte
    int sentencelength=rand()%100;

    for(int i=0; i<sentencelength; i++)
    {
       int random_index=rand()%1000;
       strcat(rule, words[random_index]);
       strcat(rule,"_");
    }
    rule[strlen(rule)-1]=0;//eliminates the last '_'
```

```
}

char *similar_string_generator(char *source)
{
    size_t length = strlen(source);

    char *similarString = malloc(length + 1);
    if (!similarString)
    {
        exit(1);
    }
    strcpy(similarString, source);

    int num_errors = (rand() % length) + 1;

    for (int i = 0; i < num_errors; i++)
    {
        int error_type = rand() % 3;
        int pos = rand() % length;

        if (error_type == 0 && pos < length - 1)
        {
            char temp = similarString[pos];
            similarString[pos] = similarString[pos + 1];
            similarString[pos + 1] = temp;
        }
        else if (error_type == 1 && length > 1)
        {
            memmove(&similarString[pos], &similarString[pos + 1], length -
                pos);
            similarString[length - 1] = '\0';
            length--;
        }
        else if (error_type == 2)
        {
            similarString[pos] = 'a' + (rand() % 26);
        }
    }

    return similarString;
}


char** allocate2DArray(int rows, int cols)
{
    char **array;
    int i;

    // Allocate memory for the array of pointers
    array = (char**) calloc(rows, sizeof(char*));

    if (array == NULL)
    {
        printf("Memory allocation failed.\n");
        exit(2);
    }

    // Allocate memory for each row
    for (i = 0; i < rows; i++)
    {
        array[i] = (char*) calloc(cols, sizeof(char));
```

```
109
110         if (array[i] == NULL)
111         {
112             printf("Memory allocation failed.\n");
113
114             // Free previously allocated rows
115             while (--i >= 0)
116                 free(array[i]);
117             free(array);
118             exit(3);
119         }
120     }
121
122     return array;
123 }
124
125
126 void free2DArray(char **array, int rows)
127 {
128     for (int i = 0; i < rows; i++) {
129         free(array[i]);
130     }
131     free(array);
132 }
```

7 Program Python

The program is based on 1 file.

```
1
2  def MinOfTwo(a,b):
3      if a<b:
4          return a
5      else:
6          return b
7  def MinimumNoOperations(rule,code_fragment):
8      n=len(rule)
9      m=len(code_fragment)
10     dp = [[0] * (m+1) for _ in range(n+1)]
11     for i in range(n+1):
12         dp[i][0]=i
13     for j in range (m+1):
14         dp[0][j]=j
15     for i in range (n):
16         for j in range(m):
17             if rule[i]==code_fragment[j]:
18                 dp[i+1][j+1]=dp[i][j]
19             else:
20                  insert_cost=dp[i+1][j]+1
21                  delete_cost=dp[i][j+1]+1
22                  replace_cost=dp[i][j]+1
23                  if rule[i-1]==code_fragment[j] and rule[i]==code_fragment[j
                        -1]:
24                      swap_cost=dp[i-1][j-1]+1
25                      replace_cost=MinOfTwo(replace_cost,swap_cost)
26                  dp[i+1][j+1]=MinOfTwo(MinOfTwo(insert_cost,delete_cost),
                        replace_cost)
27     return dp[n][m]
28 rule=input("Type the rule that will define the correct syntax: ")
29 code_fragment=input("Introduce the code fragment : ")
```

```
nr=MinimumNoOperations(rule,code_fragment)
print(nr)
```

8 Github + Useful links:

GitHub Profile

Edit Distance - Youtube

The VJudge Problem - Edit Distance

Useful Python Guide - Youtube

Levenshtein Algorithm (Explained In Details)

StackOverFlow