Algorithms Design Homework Assignment

Mateescu Corina-Elena

First year

CEN1.2A

Problems statement

We have the task of developing an algorithm for an advanced code editor that automatically corrects syntax errors in programming languages. 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 minimum 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 assume 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 required to correct the code fragment so that it matches the pattern defined by the rule. These operations may include, for example, reversing the characters ”n” and ”u” to obtain ”func”, then inserting the missing characters ”t” and ”)”, so that we obtain ”func(myFunction)” according to the given rule.

Algorithms

To solve this problem, I used a dynamic programming approach. First, I chose an example fragment to be corrected and a syntax rule to follow. Then I initialized a matrix of size (x+1)*(y+1) (x is the length of the fragment given as example and y is the length of the corrected syntax) to store the minimum edit distances and fill it conforming to the rules (if the characters in both strings are the same no operation to compute the minimum edit distance is needed, else the algorithm checks whether a substitution, insertion or deletion results the minimum edit distance). After completing the matrix, the result will be found in the last cell. The memory is then freed before returning said result.

- The algorithm for solving the problem:

```
Minimum-of-three(a, b, c)
    if a < b and a < c
        return a
    else
        if b < a and b < c
            return b
    else
        return c

Min-edit-distance(fragment, correctsyntax)
    x= length of fragment
    y= length of syntax
Let dp [0...x+1] [0...y+1] be new matrix
for i=0,x do
    dp[i][0] = i
for j =0,y do:
    dp[0][j] = j
for i=1, x do:
    for j=1,y do:
        if fragment[i-1]=correctsyntax[j-1]
            dp[i][j] = dp[i-1][j-1]
```

```
        else:
            dp[i][j] = Minimum-of-three(dp[i-1][j-1] + 1,
                                dp[i][j-1] + 1,
                                dp[i-1][j] + 1)
result = dp[x][y]
return result

main()

fragment = "fnuc(myFuncction"
correctsyntax = "func(myFunction)"
Min-operations=Min-edit-distance(fragment, correctsyntax)
print  Min-operations
```

- Memory and Computational complexity of the algorithm:

Because of the dynamic programming approach, the memory and the computational complexity are the same for the best and worst cases. In the best case, when the fragment matches the syntax rule exactly, the matrix still needs to be filled to confirm the match, and in the worst case, when every character needs to be modified, the matrix needs to be filled to determine the minimum number of operations.

The matrix “dp” has the dimensions (x+1)*(y+1)

Initializing the first row and column takes O(x+y)

The two “for” loops have a time complexity of O(x*y)

Each computation of a cell inside the “for” loops takes O(1)

=> The computational complexity is O(x*y)

The memory complexity is O(x*y) because that is the size of the “dp” matrix, and all the additional space used to store the strings and indices is negligible compared to it.

The same algorithm in pyton:

```
def min_edit_distance(fragment, correctsyntax):
    x = len(fragment)
    y = len(correctsyntax)
    # Create a DP table
    dp = [[0] * (y+1) for _ in range(x+1)]
```

```
    # Initialize the table
    for i in range(x+1):
        dp[i][0]=i  # deletions
    for j in range(y+1):
        dp[0][j] =j  # insertions
    # Fill the DP table
    for i in range(1,x+1):
        for j in range(1,y+1):
            if fragment[i-1]==correctsyntax[j-1]:
                dp[i][j]=dp[i-1][j-1]  # no operation needed
            else:
                dp[i][j]=min(dp[i-1][j-1]+1,  # substitution
                dp[i][j-1]+1,     # insertion
                dp[i-1][j]+1)     # deletion
    return dp[x][y]
if __name__ == "__main__":
    fragment = "fnuc(myFuncction"
    correctsyntax = "func(myFunction)"
    min_operations = min_edit_distance(fragment, correctsyntax)
    print(f"Minimum operations required: {min_operations}")
```

For the testing of the application and experimental results I needed to add to the algorithm a function that randomly generates non-trivial data (strings of characters) of a specified length. The function randomly chooses between lowercase and uppercase letters of the alphabet, as well as the ten digits and adds them to a string.

- The function that generates the non-trivial data:

generate-random-string(length):

1. >Allocate memory for string str[length + 1]
2. >Define character set “charset” containing lowercase, uppercase letters, and digits
3. charset-size = size of “charset”-1
4. For i=0,length-1 do:
5. key= >Pick a random character from charset
6. str[i] = charset[key]
7. str[length] = '\0'

8. Return str

- Memory and Computational complexity of the algorithm with the added function:

The memory and the computational complexity are the same as before, for both the best and worst cases.

Repository link:https://github.com/CorinaMateescu/Algorithms-Design-Homework-Assignment.git

Experimental data

To check the correctness of the output I manually computed the results for the first 3 experiments, each having 100 characters. Subsequent experiments have from 1000 to 10000 characters.

- Experiment 1:

Input:

Fragment:
bIcsCZ39xPDbykftuDnW6qYaMY4HHGjwgGzmBcwE3La5goMOdktEZjxwIshmYq1yKE43XxE7FWs4vAaGuSSwHSFsxgEPbVS4XRi0

Syntax:
mKP7xIEVtTaWnQnicpCnhnNSDTgyAjgc28DPCemVzkjafHk6xfTOTEjIeGOGdS5vV9x9hvdKiwm9lH06rnMxFenx49sRPbjjyfbK

Output:

Minimum operations required: 96

execution time : 4.316 s

- Experiment 2:

Input:

Fragment:
2CIQeBFg9X406EZF2AUzL3LDijr0DQ8iWHSjHJCik4zBbSA9rFOcdDlVYNFk8pyGlDK1jAFTNrIevUgRxELohcIny38NDLdJJqjd

Syntax:
qDnVqC8fAi3reaALPWvsYRznSQliifhPMLQjg9aemSWpsOUD1RLXOCWcYrK8kiD138FmpJVsBRwfnpAYtN9IPO1aUaHXhQDaMgsR

Output:

Minimum operations required: 98

execution time : 0.012 s

- Experiment 3:

Input:

Fragment:
fFpTeXM5bArPEzCdvV6SgTmeEgnGYeFs9jaxARFhH6m75ZsqLA4Daw2pi9TH1xSTx6NHCk0nbBq92Riw4ZIdo4z4JWeMnQ0PfyW0

Syntax:
GYs5BW9BHNplZwZWDeuONeVZfzsaAXMuI4ToKQ04m1lt0oPD7OzTgmsWxFnBIry9RWsSim4DZglDYbGvhaQmLySGfgGtGtXuNtCW

Output:

Minimum operations required: 100

execution time : 0.213 s

- Experiment 4(1000 characters):

Input:

Fragment (first 100 chars):
3MXNK6u4mrh5AEJVuzkFHb98JlSu0eoNgJZyuCP36EPZjTmuw5FaOhizEPp1oKLdZrPgBbQf
HmvJxV2EF5RqaeXaT4aOnpxaLV6v...

Syntax (first 100 chars):
8miJt86fqqnSxxtw33K2IfaGDuHMRdV9b3u7mwJQ75fHszCYj1CXJXHDGQizAd5QPsI7lV8d
h7OEe4V5e0lATB0oXj9hcEJiGjIz...

Output:

Minimum operations required: 953

execution time : 10.674 s

- Experiment 5(2000 characters):

Input:

Fragment (first 100 chars):
tVsHzdrfS5JgZAZTLK4nHc1Vlfimbvr7rZUCESLAG7xforyr70l9fKU7u4KgDG6lYpnh6oauCKQz
qfaHCcYXTgZPXeANtTA2OIRl...

Syntax (first 100 chars):
KVRYpOiWv9sIepx6koNhjKZ5mvIAya9bIRvW5QAP0CkOgo7wPQXuLxSJUprA8KASHnNM9d
KlLpSmqs6ERKbbMUTCqlQNFhX2ZT62...

Output:

Minimum operations required: 1913

execution time : 8.483 s

- Experiment 6(3000 characters):

Input:

Fragment (first 100 chars):
npLFdQZbrA6PNL2Bc7DYhukup3w2w6xBKUWzXzvzd71L2LSCMWM7p4mKp2k5yvaBOYVJ
tAjRcrCBWsK9vS8wVpKZlT8wSH70oQxI...

Syntax (first 100 chars):
5c5BUDMd3HjTxpAarYBXamQ06iJkn9yym013JqnlvMbt3kIkI3Or4XrrFVkd0wGsziFPF5Khrlj
DbGUKQcZchF3eD6v5DDyqPOtj...

Minimum operations required: 2865

Output:

execution time : 8.517 s

- Experiment 7(4000 characters):

Input:

Fragment (first 100 chars):
RFHHeAVB62kH33mlFJU3BH2Y0ksZ80B0aozoeuvtgA7wLV3pPOSnUvuAhohFXZDJqmGHi9
4EFqlZMzs9Z6ARt1jrUVG49DV0rmXA...

Syntax (first 100 chars):
tlZY8fVjCSWF85g1Willb0lORPsa1VnVW1JjtdoypABwkHoh4fu7zyhp6kNffg6OQ9i6ZIK6CCv
FuPCDF3B9Wa8688QgRkYiuoLS...

Output:

Minimum operations required: 3819

execution time : 8.627 s

- Experiment 8(5000 characters):

Input: Fragment (first 100 chars):
YXjue1svCBg6Qgb6p80HBD41WzFP3rnALctKUOwHXmg1XZZ2fwvQ8rPQG4Tl9ixQwCt2bweERuWH12JefvPvxsewZ7Y3gp52IWLt...

Syntax (first 100 chars):
4w1NDdMWao8eQHZ0q1ki7J7FjJX9oM30DPP854RjZqY3m5ZJ96ZwZ80KBPXihRl2FhTnyyO52kNTbgn59ZfhhAj8fy3lI88trcWB...

Minimum operations required: 4772

Output:

execution time : 10.699 s

- Experiment 9(6000 characters):

Input:

Fragment (first 100 chars):
1UWbwfSy1hrO86NvWpBkSpN4gHoDkOt8ixpLntBSekfM9CE3JxsyemcG6jZGhmZoycNTXRqIUQsCuPk93Ri1Pb7TLbIdlH4bMvVR...

Syntax (first 100 chars):
OiKrYrewQGB5zzTbq8Ld23HRY8H8ZjGK7oYIJDFnhCF6w3XotuxmkaersVd7wPXojevJ1qmjfnrpciVxanBaYaynSqxw77JtyzG...

Minimum operations required: 5726

Output:

execution time : 8.752 s

- Experiment 10(7000 characters):

Input:

Fragment (first 100 chars):
Iky7FwQF4CrxSRdFTXtDXQv2u1G8lv1YUQ80jwiSViWMSO6syyCBViXOIWl8O3JycgiFxMICA
n1eZ5mWTKvLKZxnXVbttrOAlvYs...

Syntax (first 100 chars):
r5tzY79vBlWc7MAimi43u1gZXWqXRGxLUFkgBQh6xixmRI9pQsinLz9XDHJMTFyrU5rIVHvdP
R3KJz73yIrNPKQdT4YxpKV6kSDs...

Output:

Minimum operations required: 6674

execution time : 10.824 s

- Experiment 11(8000 characters):

Input:

Fragment (first 100 chars):
bPGBRstDT6vqlEThdNff4yyXCVMKHzuHgpvbDs507m8No6xvCvjG7gpmpy6J1W9v7fE8sKP
aKTNw5Yhlx5V3BXViXIH6oR5FiroC...

Syntax (first 100 chars):
ZnDP6Cx4gKhj6IYWOU4I8hpYncVsT7gjxxF0mL1J5LoyCn9Ht2Xxrp0e7a3n6xAOIKQk9Y5H5
CAJP6FvxBqquM9dVveaf9UhEFSe...

Output:

Minimum operations required: 7634

execution time : 10.972 s

- Experiment 12(9000 characters):

Input:

Fragment (first 100 chars):
ZjIliYLCNu74aDfk4wJ9mruxJnErU3VOxrnqdjv0lZTz2gENmF1SRDawAAhmWynXxukePargVY
opzpTjZIouMjVniqtKwm0vd4fN...

Syntax (first 100 chars):
S0JSLy47mH6rPILF7zKXJiXAxRUqLn41Qw1WhhpjvExLvRf4YFFgLPM8IOqxGhKV33xChzZlM
Ivb8ukRJSMOajyvsdkoZVzyT6ru...

Output:

Minimum operations required: 8578

execution time : 9.295 s

- Experiment 13(10000 characters):

Input:

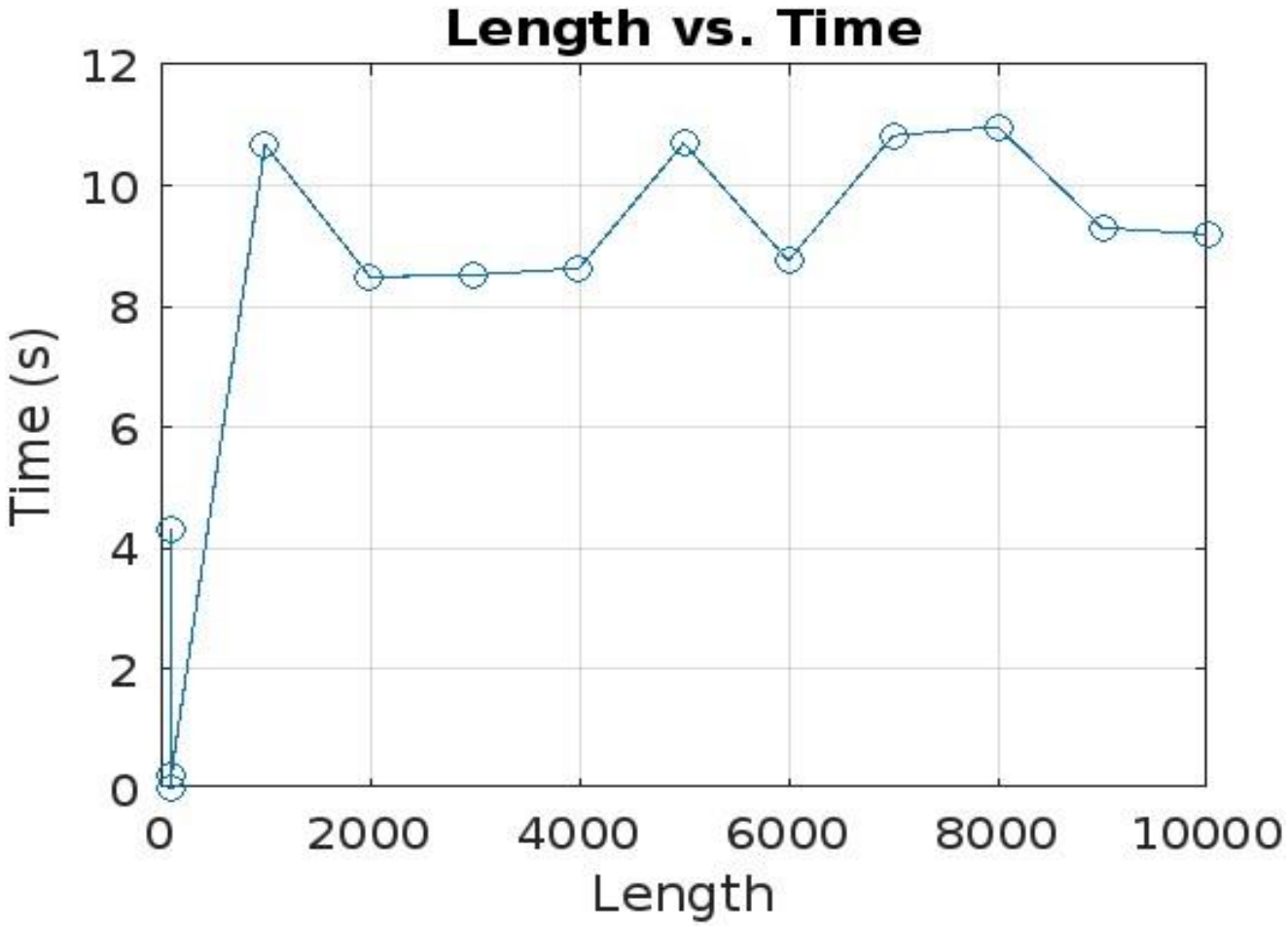
Fragment (first 100 chars):
PDn97PfowdRqxX01tpeufu5fSGaShkMPYNIjtRiCbe8zD91azldGm0hd80RkP7alUKiLCjnbm
aqLfVh88CaF57Nf5eR3uiFywPMj...

Syntax (first 100 chars):
ise55euk9xXGSFmoJkhISYdApOzLWifCe2Bo2cuABabwcM88ORVNj4w99Sv45c0tVwBvzQZhBzI6JB6p6xiUs2DS1v4JXfsl4XVV...

Output:

Minimum operations required: 9540

execution time : 9.192 s



In the graphical representations of the experiments, we can better analyze the variation of the execution times. It is noticeable that for the experiments with the length of 100 characters the times vary significantly, from which we can deduce that the inconsistency is probably caused by the randomness of the string generation. From the lengths of 2000 characters and beyond the times become more reflective of the

computational complexity, seemingly increasing with the length of the strings. However, we can notice some anomalies, probably determined by the randomness of the input.

Conclusion

Working on this project not only has allowed me to refresh my knowledge on basic (the insertion, deletion and substitution of characters in strings) and complex (random generation of large and very large data sets) programming concepts, but it has also pushed me to go out of my comfort zone and acquire new knowledge about subjects that I was either ill-educated about (coding in Python) or completely unfamiliar with (working in Matlab) . While some of these things proved quite challenging (at first, I could not figure out how to create a table in Matlab), perseverance and the looming deadline kept me straight on the path that ultimately led to the finalization of the project.