Algoritm pentru Corectarea Automată a Erorilor de Sintaxă în Fragmente de Cod

[Punec Antonio Razvan]

[Anul 1]

Coperta

- **Titlul:** Algoritm pentru Corectarea Automată a Erorilor de Sintaxă în Fragmente de Cod
- **Student:** [Punec Antonio Razvan]
- **Grupa:** [Grupa CR1.3A]
- **Anul:** [Anul 1]
- **Secțiunea:** [Secțiunea Calculatoare Romana]

Enunțul problemei

Obiectivul nostru este de a dezvolta un algoritm pentru un editor de cod avansat care să corecteze automat erorile de sintaxă din limbajele de programare. Primim o specificație clară a sintaxei valide sub forma unei "reguli" și un fragment de cod care conține erori de sintaxă, adică nu se conformează acelei reguli. Obiectivul este de a găsi numărul minim de operații necesare pentru a transforma fragmentul de cod într-unul care respectă regula dată. Aceste operații pot include substituiri de caractere, inserții sau ștergeri.

Exemplu de regulă: "Fiecare funcție trebuie să înceapă cu cuvântul cheie 'func', urmat de numele funcției închis în paranteze."

Fragment de cod dat: "fnuc(myFuncion"

Forma corectă conform regulii: "func(myFunction)"

Algoritmi

Pentru a rezolva această problemă, vom utiliza distanța de editare (Levenshtein distance), care calculează numărul minim de operații necesare pentru a transforma un șir de caractere într-altul.

Pseudo-codul algoritmului

```
function levenshtein_distance(s1, s2):
    m = length of s1
    n = length of s2
    create a matrix dp of size (m+1) x (n+1)

    for i from 0 to m:
        dp[i][0] = i
    for j from 0 to n:
        dp[0][j] = j

    for i from 1 to m:
        for j from 1 to n:
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + minimum(dp[i-1][j], dp[i][j-1], dp[i-1]

    return dp[m][n]

function correct_code(fragment, correct_form):
    return levenshtein_distance(fragment, correct_form)
```

Analiza complexității

- Timp de execuție: $O(m \times n)$, unde ‘m‘ și ‘n‘ sunt lungimile celor două șiruri comparate. - Complexitate spațială: $O(m \times n)$ pentru matricea ‘dp‘.

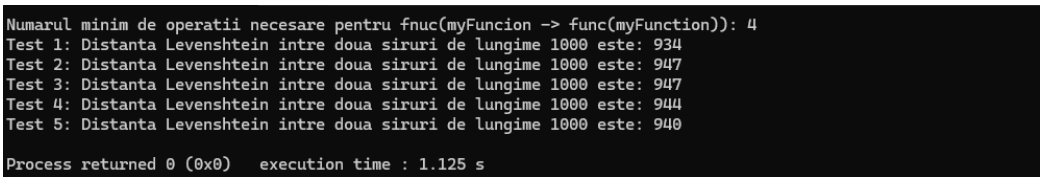

Figure 1: Enter Caption

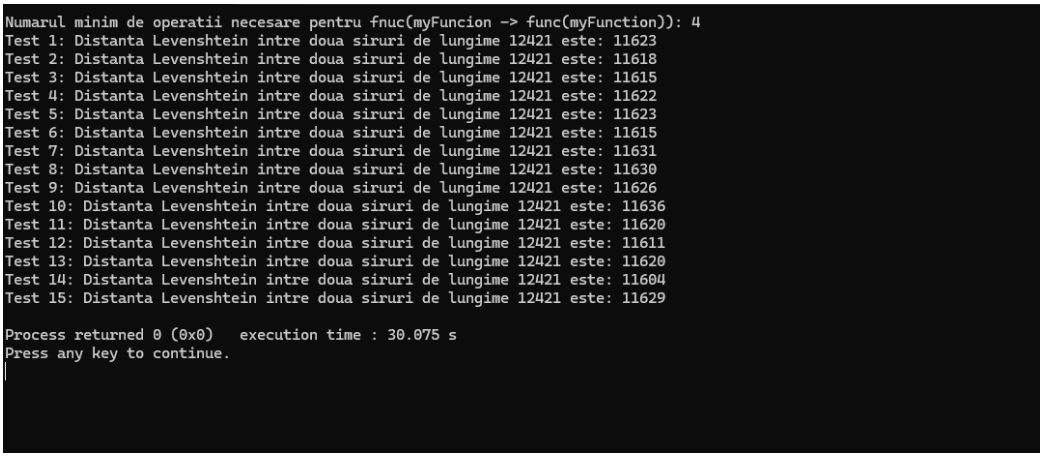

Figure 2: Enter Caption

Date experimentale

Proiectarea experimentală a aplicației

Vom implementa functia de calculare a distantei Levenshtein in C pentru a determina numarul minim de operatii necesare pentru a corecta fiecare fragment de cod.

Rezultate & concluzii

În urma rulării aplicației, vom colecta rezultatele pentru fiecare set de date de intrare. Rezultatele vor include numărul minim de operații necesare pentru fiecare fragment de cod, precum și timpul de execuție pentru fiecare set de date.

Concluziile vor fi bazate pe analiza datelor experimentale, demonstrând eficiența algoritmului în corectarea erorilor de sintaxă.

https://github.com/Razvan12123/Tema-de-casa