**Algorytmy i struktury danych, Teleinformatyka, I rok**

**Raport z laboratorium nr: 8**

**Imię i nazwisko studenta: Patryk Kurek**

**nr indeksu: 414887**

*1. W pole poniżej wklej najważniejszy (według Ciebie) fragment kodu źródłowego z zajęć (maksymalnie 15 linii).*

def rabin\_karp(word, arr):

positions = []

for i in range(0,len(arr)-2):

rolling\_hash = RollingHash(arr[i], len(word))

word\_hash = RollingHash(word, len(word))

for j in range(len(arr[i]) - len(word) + 1):

if rolling\_hash.hash == word\_hash.hash:

if rolling\_hash.window\_text() == word:

temp = [arr[i][j],arr[i+1][j],arr[i+2][j]]

letters = RollingHash(temp,len(temp))

if letters.hash == word\_hash.hash:

if letters.window\_text() == word:

positions.append({"y": i,"x": j})

rolling\_hash.move\_window()

return positions

W tym kodzie funkcja rabin\_karp przyjmuje słowo word oraz dwuwymiarową tablicę arr. Iteruje po każdym wierszu tablicy arr, tworząc obiekt RollingHash dla bieżącego wiersza i słowa word. Następnie stosuje algorytm Rabin-Karpa, przesuwając okno przez wiersz i porównując hashe oraz tekst okna z word. Jeśli zostanie znalezione dopasowanie, sprawdza kolejne wiersze w tej samej pozycji kolumny, aby sprawdzić, czy wzorzec się powtarza. Jeśli spełnione są wszystkie warunki, pozycja zostaje dodana do listy positions. Ta funkcja pokazuje doskonale jak działa algorytm.

*Uzasadnij swój wybór.*

*2. Podsumuj wyniki uzyskane podczas wykonywania ćwiczenia. Co ciekawego zauważyłeś? Czego się nauczyłeś? Jeśli instrukcja zawierała pytania, odpowiedz na nie. Do sprawozdania możesz dodać wykresy jeśli*

*jest taka potrzeba.*

Algorytm naiwny o dziwo zadziałał lepiej. Typowo algorytm Karpa-Rabina pracuje w liniowej klasie złożoności obliczeniowej O(m+n), zatem dla długich tekstów ma on zdecydowaną przewagę. W naszym przypadku dla plików rzędu 8000 znaków algorytm naiwny ma przewagę. Algorytm Rabina-Karpa jest oparty na haszowaniu, który może generować kolizje (tzn. różne ciągi znaków mogą dać ten sam hash). W przypadku wystąpienia kolizji, algorytm Rabina-Karpa musi wykonać dodatkowe porównania, aby potwierdzić dopasowanie wzorca.

Algorytm naiwny, z drugiej strony, porównuje wzorzec z tekstem znak po znaku, bez żadnych operacji haszowania. Chociaż algorytm naiwny ma stałą złożoność czasową O(n \* m), gdzie n jest długością tekstu, a m jest długością wzorca, może być szybszy w praktyce, jeśli występuje duża liczba kolizji w algorytmie Rabin-Karpa. Warto jednak zauważyć, że algorytm Rabin-Karpa ma potencjał do osiągnięcia optymalnej złożoności czasowej O(n + m) w przypadku, gdy kolizje są rzadkie.

Wyniki:

1000 pattern naive: 0.21819496154785156

number of occurrences: 6

1000 pattern rabin karp: 0.9923007488250732

2000 pattern naive: 0.8897705078125

number of occurrences: 40

2000 pattern rabin karp: 3.9599506855010986

3000 pattern naive: 2.0058658123016357

number of occurrences: 57

3000 pattern rabin karp: 8.948881149291992

4000 pattern naive: 3.576605796813965

number of occurrences: 97

4000 pattern rabin karp: 16.527572631835938

5000 pattern naive: 5.9574010372161865

number of occurrences: 161

5000 pattern rabin karp: 25.101385354995728

8000 pattern naive: 14.43262791633606

number of occurrences: 393

8000 pattern rabin karp: 63.70729970932007

Dla naszych plików algorytm naiwny okazuje się szybszy, ponieważ być może występuję w nich dużo kolizji.