1. Kod źródłowy

```
for i in range(len(lstOfHash)):
  if lstOfHash[i] == value:
      flag = True
      if lstFH[i][0] != pattern[0]:
          flag = False
      if lstFH[i][1] != pattern[1]:
          flag = False
      if lstFH[i][2] != pattern[2]:
          flag = False
      if lstFH[i][3] != pattern[3]:
          flag = False
      if lstFH[i][4] != pattern[4]:
          flag = False
      if flag:
          counter += 1
```

Wybrałem powyższy fragment kodu, ponieważ jest on idealną ilustracją tego jak działa algorytm Karpa-Rabina. Najpierw porównujemy ze sobą wartości otrzymane w wyniku użycia funkcji haszującej, a dopiero w przypadku kiedy są one sobie równe sprawdzamy czy na danej pozycji rzeczywiście znajduje się szukany wzorzec. W tym laboratorium po raz pierwszy zetknąłem się z zastosowaniem funkcji haszującej oraz z wykrywaniem wzorca w tekście. Implementacja powyższego kodu w znacznym stopniu przybliżyła mi idę tych dwóch aspektów programistycznych, a także przekonała mnie do stosowania funkcji haszujących w praktyce.

2. Wyniki i podsumowanie

Podstawowym celem tego ćwiczenia laboratoryjnego było napisanie programu, który prawidłowo znajduje liczbę wystąpień danego wzorca dwuwymiarowego w tekście. Wykonałem to zadanie przy pomocy dwóch algorytmów; algorytmu naiwnego oraz algorytmu Karpa-Rabina. Doświadczenie przeprowadziłem na dołączonych do instrukcji macierzach o rozmiarach: 1000x1000, 2000x2000, 3000x3000, 4000x4000, 5000x5000, 8000x8000. W obu przypadkach otrzymałem identyczne rezultaty:

macierz	wystąpienia wzorca
1000	6
2000	40
3000	56
4000	97
5000	161
8000	393

Kiedy już wiemy, że oba algorytmy dają na wyjściu takie same wyniki, istotnym aspektem staje się to w jakim czasie się wykonują. W przypadku algorytmu Karpa-Rabina musimy najpierw dokonać haszowania wzorca oraz odpowiednich elementów w tekście, aby móc je ze sobą wydajnie porównywać. Może to być operacja stosunkowo kosztowna czasowo, w zależności od tego jakiej funkcji haszującej używamy. Jednak kiedy już zostanie to zrobione, możemy dostrzec znaczną przewagę czasową przy wyszukiwaniu wzorca w porównaniu do algorytmu naiwnego.

Zastosowana przeze mnie funkcja haszująca całkowicie opiera się na wprowadzonych danych oraz do wyliczenia wartości haszującej wykorzystuje każdy element listy. Funkcja wykorzystuje wartości znaków w tablicy ASCII, sumując je ze sobą, a następnie na przemian dodaje i odejmuje od klucza reszty z dzielenia modulo kolejnych elementów przez kolejne liczby pierwsze. To zapewnia możliwie unikalne wartości dla poszczególnych ciągów znaków.

Zastanówmy się teraz skąd wynika przewaga czasowa algorytmu Karpa-Rabina nad algorytmem naiwnym. W przypadku tego drugiego musimy dla każdego elementu w danym tekście porównać ze sobą tyle elementów ile posiada wzorzec. To daje nam złożoność czasową wynoszącą O(n*m), gdzie n- długoś teksu, m- długość wzorca. W przypadku bardzo dużych m, zbliżonych do wartości n otrzymujemy zależność kwadratową O(n^2) . Natomiast dla algorytmu Karpa-Rabina porównujemy czy dany ciąg zgadza się ze wzorcem dopiero wtedy gdy jego wartość otrzymana przez haszowanie odpowiada wartości wzorca. To daje nam złożoność czasową postaci O(n+m) w średnim oraz w najlepszym przypadku. Tylko w najgorszym przypadku (czyli takim, w którym wartości funkcji skrótu wszystkich badanych ciągów znakowych są równe wartości wzorca) otrzymujemy złożoność O(n*m). Ogólnie możemy przyjąć, że jest to zależność liniowa.

Jeśli chodzi o otrzymane przeze mnie pomiary czasowe to algorytm Karpa-Rabina zwykle oblicza ilość wystąpień wzorca od 5 do 15 razy szybciej niż algorytm naiwny. Średnie czasy dla poszczególnych macierzy przedstawiam w poniższej tabeli. (Dla macierzy 8000x8000 haszowanie wszystkich elementów zajmowało zbyt dużo czasu, aby wyznaczyć wartość średnią.)

macierz	algorytm naiwny	Karp-Rabin
1000	0.3198	0.0469
2000	1.1874	0.2042
3000	2.7149	0.4841
4000	4.7397	0.7422
5000	7.4215	1.2058
8000	19.1671	---------

Na koniec prezentuję na jednym wykresie zależności czasu od ilości rozmiaru macierzy. Jak widać algorytm naiwny daje dam zależność potęgową, podczas kiedy dla algorytmu Karpa-Rabina w przybliżeniu otrzymujemy zależność liniową.