Zadanie Trójki palindromów – LOGIA 19 SP (2018/19), etap 3

Treść zadania

Ania bada liczby. Ostatnio zaintrygowało ją, czy można daną liczbę zapisać w postaci sumy trzech różnych, dodatnich liczb palindromicznych, tj. takich, które czytane od początku i końca są identyczne. Pomóż Ani i napisz jednoparametrową funkcję pali, której parametrem jest liczba nie mniejsza niż 10 i nie większa niż 100 000. Wynikiem funkcji jest liczba różnych trójek liczb palindromicznych, których suma jest równa liczbie podanej jako parametr. Postaraj się tak rozwiązać zadanie, aby nie trzeba było długo czekać na wynik. Ocenie podlega też czas działania Twojego rozwiązania.

Przykłady:

Wynikiem **pali(100)** jest **7**. Trójki palindromów to: 1 11 88, 1 22 77, 1 33 66, 1 44 55, 3 9 88, 4 8 88, 5 7 88.

Wynikiem **pali(27)** jest **3**. Trójki palindromów to: 1 4 22, 2 3 22, 7 9 11.

Omówienie rozwiązania

Pomysł rozwiązania opiera się na realizacji algorytmu w dwóch krokach: najpierw generujemy uporządkowaną rosnąco listę wszystkich palindromów z rozpatrywanego przedziału, a potem sprawdzamy, ile sum trójek palindromów z tej listy jest równych parametrowi funkcji.

Sprawdzanie, czy dana liczba jest palindromem, można zaimplementować porównując czy napis utworzony z liczby jest równy napisowi będącego jego odwrotnością. Jeśli sprawdzamy trójki liczb palindromicznych warto zadbać o to, by nie sprawdzać tych samych trójek wielokrotnie. Dlatego iterujemy po pierwszej i drugiej liczbie z trójki, przy czym druga liczba jest zawsze większa niż pierwsza. Trzecią liczbę wyliczamy jako różnicę parametru funkcji i sumy liczb pierwszej oraz drugiej. Następnie sprawdzamy, czy znajduje się ona na wygenerowanej wcześniej liście liczb palindromicznych. Dodatkowo sprawdzamy, czy ta liczba jest większa od pozostałych, inaczej ta trójka była już rozpatrywana.

Lepsze pod względem złożoności czasowej jest rozwiązanie kombinatoryczne, ale opracowanie algorytmu jest bardziej skomplikowane, dlatego je pomijamy.

Rozwiązanie w języku Python

Funkcja `czy_pali(x)` sprawdza, czy danych napis jest palindromem. Zapis `napis[::-1]` wyodrębnia z napisu wszystkie znaki od początku do końca, ale w odwrotnej kolejności, czyli odwraca łańcuch znaków.

```
1.  def czy_pali(x):
2.      return str(x) == str(x)[::-1]
3.
4.  def pali(liczba):
5.      # tworzenie listy liczb palindromicznych
6.      pom = []
7.      for i in range(1, liczba+1):
8.          if czy_pali(i):
9.              pom.append(i)
10.
11.     # sprawdzanie trójek liczb
12.     ile = 0
13.     for i in range(len(pom)):
14.         for j in range(i+1, len(pom)):
15.             x = pom[i]
16.             y = pom[j]
17.             z = liczba - x - y
18.             if z > y and z in pom:
19.                 ile += 1
20.     return ile
```

Testy

Testy obejmują proste przypadki, które można przeliczyć ręcznie. Warto też zwrócić uwagę na przypadki brzegowe. W tym zadaniu pewną trudność stanowi opracowanie algorytmu o dobrej złożoności czasowej. Jeśli na przykład będziemy sprawdzać wszystkie możliwe trójki liczb, algorytm będzie działać dłużej. Dlatego warto uwzględnić w testach przypadki dla dużej wartości parametru.

Wywołanie – Python	**Wynik**
pali(17)	9
pali(24)	4
pali(383)	23
pali(384)	33
pali(706)	117
pali(707)	65
pali(10000)	135
pali(10002)	134
pali(98998)	890
pali(99998)	7966