Prime Digit Sum

Tae Young Kevin Shin

Wofford College  
429 North Church Street  
Spartanburg, SC 29303-3663  
+1 (864) 540-1327

shint@email.wofford.edu

**ABSTRACT**

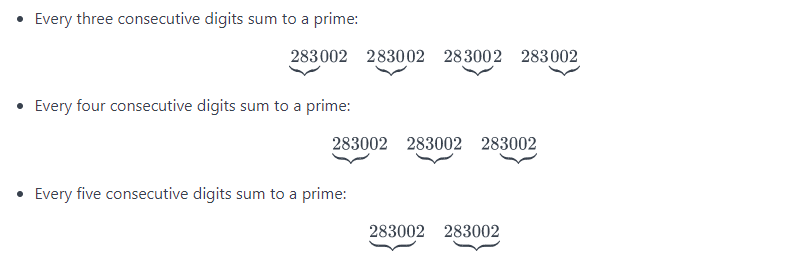
In this paper, we describe the algorithm property of Dynamic programming to find prime digit sum.

**Keywords**

Brute Force, Dynamic Programming, Memoization,

# Introduction

In this paper, we come across any random number from which we seek to find whether the sum of three, four, and five consecutive digits of the given number respectively accord with prime number. This rule is called Chloe’s rules. Under *q* number of queries, each query contains an integer value of n denoting the number of digits. With regard to constraints, 1≤ q ≤ 2 x 104, such that q represents the number of queries and 1≤ n ≤4 x 105, such that n represents the number of digits to be tested in each query. The final output from each query should be the number of units that satisfy Chloe’s rules.



# Background

**2.1 Problem Solution Synopsis**

As aforementioned above in Keywords section, we mainly focus on using Brute Force Algorithm, Dynamic Programming, and Memoization.

**2.2 Problem Solution Description**

Brute Force algorithm based on the definition of Prime number, iterative statement, and conditional statement is very clever and straightforward approach for this problem. When this program is implemented, the main function is supposed to call many other multiple sub-functions that are to be designed for each own purpose. This schema is closely intertwined with Dynamic Programming, and Memoization concepts.

# Algorithm

In general, there are three major blocks of functions that we need to construct.

**3.1 First sub function**

The first one is defining a function that checks whether the given input value is a prime number or not. This function is going to be continuously called as we need to check whether any instance of a number is a prime number or not.

**3.2 Second sub functions**

Second function is to identify whether all of three, four, and five consecutive digits sum of the given input n turn out to be prime numbers of not. If so, the output of this function is going to be usefully utilized in third function’s implementation.

**3.3 Third main function**

The last function should serve implementing the process of given queries by checking Chloe’s rules of given input determined by a user. Primarily, this function should generate all possibilities of variable-sized input given by a user and call all second functions defined above. This function is supposed to return the final output from all second functions and count how many numbers of each case belong to Chloe’s rules, leading to returning the final result.

## Pseudocode

1 // Find the number of units whose all three, four, and five consecutive digits sum are prime numbers

2 // Input: first line <- the number of queries(n),

subsequent n lines <- the number of digits of input

3 // Output: subsequent n lines <- the number of units that satisfy Chloe’s rules

4 first function: is\_prime(n)

5 for i = 2 to sqrt(n) + 1

6 if n % i = 0:

7 return False

8 return True

9 second function (1): is\_three\_digits\_sum\_prime(n):

10 for i = 0 to len(n-3)

11 sum <- n[i] + n[i+1] + n[i+2]

12 if is\_prime(sum) = False

13 return False

14 return True

15 second function (2): is\_four\_digits\_sum\_prime(n):

16 for i = 0 to len(n-4)

17 sum <-n[i] + n[i+1] + n[i+2] + n[i+3]

18 if is\_prime(sum) = False

19 return False

20 return True

21 second function (3): is\_five\_digits\_sum\_prime(n):

22 for i = 0 to len(n-5)

23 sum <- n[i] + n[i+1] + n[i+2] + n[i+3] + n[i+4]

24 if is\_prime(sum) = False

25 return False

26 return True

27 third function: final\_digits(n):

28 for i = 10n-1 to 10n -1

29 x <- is\_three\_digits\_sum\_prime(i)

30 y <- is\_four\_digits\_sum\_prime(i)

31 z <- is\_five\_digits\_sum\_prime(i)

32 if m,n,l are all True:

33 temp\_array <- i

34 return len(temp\_array)

35 fourth function: main\_function():

36 q <- first\_line\_input\_value (number of queries)

37 for i=0 to q-1:

38 n <- input(“number of digits)

39 return final\_digits(n)

40

## An Example

We can put 2 for q value which refers to the number of queries, and 6 as n value which represents the number of digits for first trial and 7 for the second trial.

When we put 6 for the first input trial as n denoting the number of digits, we get 95 as output. In the meantime, when we put 7 for the second input trial as n denoting the number of digits, we get 110 as output.

This means that we want to print out two cases of output resulting from first case of 6 digits number satisfying Chloe’s rules and second case of 7 digits number following all Chloe’s rules.

Once 6 is inserted as a parameter in final\_digits function, the size of iteration that for-loop should run is the difference between 105 to 106 -1, which would be 899999. On every number from 100000 to 999999, three linear for-loop checking whether they belong to three digits of them are prime numbers, four digits of them are prime numbers, and five digits of them are prime numbers. Subsequently, each group of digit units is also checked by first fibonnaci function.

This entire algorithm has a very similar characteristic of shells inside shells reminding us of divide and conquer algorithm.

## Time Complexity

The time complexity of this dynamic programming algorithm can be analyzed into separate pieces. Each small block of function from the number one, three of the number two, and the number three have influence on entire algorithm’s time complexity. Even though the time complexities of all of functions are O(n), a linear time since every function contains one single for-loop. However, the true time complexity is contingent on the input query size q and input digit size n since the major portion of iteration takes place inside final\_digits function where the number of iteration is the difference between 10n-1 to 10n -1. In addition, when every case of third function is constructed, it calls both three of second functions and the first functions sequentially in order to check consecutive number property and prime number characteristic. The final time complexity would be q times 10n-1 to 10n -1 times O(n) times O(n), which is about O(n2) times 10n time complexity.

# ConclusionS

Dynamic Programming is very powerful and systematic in terms of splitting each disparate schema into different roles. Memoization which largely contributes to this entire algorithm’s process by enabling temporary return data from each different function to be held is also very useful. Brute Force algorithm which incorporates two previous algorithm approaches was very straightforward and practical in terms of following the entire problem’s flow. However, in terms of space and time complexity, Brute Force algorithm in this case does not appear to be efficient.

# REFERENCES

1. Levitin Anany. 2012. *Introduction to The Design & Analysis of Algorithms*. 3rd ed. Boston: Pearson Education, 2012. Print.
2. Miller, Bradley M., and David L. Ranum. *4.12 Dynamic Programming – Problem Solving with Algorithms and Data Structures.* Franklin Beedle, 22 Sept. 2013. Web.16.Apr 2018.http://interactivepython.org/runestone/static/pythonds/Recursion/DynamicProgramming.html
3. Saumont, Pierre-Yves. "Do it in Java 8: Automatic memoization - DZone Performance."*Dzone.com*. N.p., 30 Sept. 2014. Web. 16 Apr. 2018.