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**

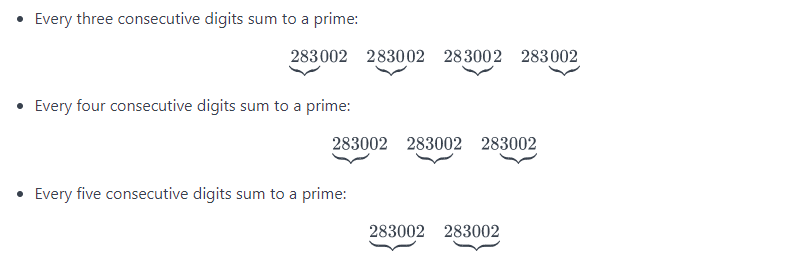
In this paper, we describe the dynamic programming and memoization algorithm to find prime digit sum.

**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 match with prime numbers. This rule is called **Chloe’s rules**. [2] Under *q* number of queries, each occurrence of 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**. For example, according to the following picture, if the given n is 6, then, we are supposed to figure out whether 6 digits of all positive integers have the following property. To be specific, we need to check whether sum of each three, four, and five consecutive digits is prime or not.



# Background

As this problem’s solution results from several different cases of functions, we should break down this problem into several small sub functions serving their own unique purposes, known as Dynamic Programming approach[3]. In addition, as sub functions share the recurrent feature of calculating the sum of consecutive digits, we can also utilize the method called Memoization, [4]an optimization technique used primarily to speed up computer programs by storing the results of repeated functions calls and returning cached result when the same inputs occur again.

# Algorithm

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

**3.1 is\_prime(n)**

This function returns whether the given number n is a prime or not. Due to this problem’s constraint that we only need to calculate from three digit sum to five digit sum, leading the minimum value from 0+0+0 = 0 to 9+9+9+9+9 = 45, this function has a repository array of Boolean values of True or False from number 0 to 45. For instance, is\_prime(37) returns True, and is\_prime(44) returns False.

**3.2 sum3\_prime(num)**

This function takes an integer n and calculate every three consecutive digits sum. Each sum will be stored at the temp vector in a global environment. It returns True if every sum is a prime, and returns False otherwise.

//input: integer value n

//output: return True if all three consecutive digits sum is prime, and return False otherwise.

//example: sum3\_prime(133) → True, sum3\_prime(1332) → False

**3.3 sum4\_prime(n)**

This function takes an integer n and updates the previous three digits sum elements in the temp vector by adding one more subsequent value of index. It returns True if every sum is a prime, and returns False otherwise.

//input: integer value n

//output: return True if all four consecutive digits sum is prime, False otherwise.

//example: sum4\_prime(101101) → True, sum4\_prime(101122) → False

**3.4 sum5\_prime(n)**

This function takes an integer n and updates the previous four digits sum elements in the temp vector by adding one more subsequent value of index. It returns True if every sum is a prime, and returns False otherwise.

//input: integer value n

//output: return True if all five consecutive digits sum is prime, False otherwise.

//example: sum5\_prime(101101) → True, sum5\_prime(101122) → False

**3.5 Chloe’s rules(d)**

This function takes the number of digits from the user’s input(d), and generate all possible integers following. Then, this function passes all possible integers to **sum3\_prime(n), sum4\_prime(n),** and **sum5\_prime(n)**. Finally, this function combines the Boolean results of all of them and determines how many integers satisfy **Chole’s rules**.

//input: integer value d which stands for the number of digits that are to be tested

//output: return True if the number of d digits satisfies is prime, False otherwise.

//example: sum3\_prime(283002) → True, sum4\_prime(283002) → True, sum5\_prime(283002) → True

**3.6 main\_function()**

//input: number of queries, number of digits that are to be tested

//output: return number of cases which satisfy **Chole’s rules** on each query

## 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(*q*),

subsequent *q* lines <- the number of digits of input(*n*)

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

4 **is\_prime(n)**

5 Array Prime[46] = {false,false,true,true,false … false}

6 return Prime[n]

7 temp =[]

8 **sum3\_prime(n)**

9 for i = 0 to len(n-2)

10 seg <- int(n[i]) + int(n[i+1]) + int(n[i+2])

11 temp.append(seg)

12 if is\_prime(seg) = False:

13 return False

14 return True

15 **sum4\_prime(n)**

16for i = 0 to len(n-3)

17 temp[i] = temp[i] + int(n[i+3])

18 if is\_prime(temp[i]) = False

19 return False

20 return True

21 **sum5\_prime(n)**

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

23 temp[i] = temp[i] + int(n[i+4])

24 if is\_prime(temp[i]) = False

25 return False

26 return True

27 **Chloe’s rules(d)**

28 result = []

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

30 i = str(i)

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

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

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

34 if x,y,z are all True:

35 result.append(i)

36 temp.clear()

37 return len(lst)

38 **main\_function()**

39 q <- first\_line\_input\_value(“Enter the number of queries: ”)

40 for i = 0 to q-1:

41 n <- input (“Enter the number of digits : ”)

42 print(final\_digits(n))

## An Example

For example, **‘665002’**is the one that satisfies **Chloe’s rules.** First and foremost, as this number is 6 digits, we can apply this number into **sum3\_prime(n), sum4\_prime(n), and sum5\_prime(n)** which calculates whether three, four, and five consecutive digits are prime numbers or not.

We can get 6+6+5, 6+5+0, 5+0+0, and 0+0+2 sequentially, which are equivalent to 17, 11, 5, and 2 according to the **sum3\_prime(665002).** Now, **temp array** is currently stored as[17,11,5,2],which are all prime numbers.

According to **sum4\_prime(665002), temp array** will be updated into [17+0, 11+0, 5+2, 2] = [17, 11, 7, 2],which are all prime numbers.

According to **sum5\_prime(n), temp array** is to be updated as [17+0, 11+2, 7, 2] = [17, 13, 7, 2], which are all prime numbers.

Therefore, we store **665002** into our result vector.

## Time Complexity

When this program is prompted to run, six different functions start to interactively run together in the form of bottom-up implementation, meaning that main function at the bottom starts to call all sub functions designed above subsequently. To be specific, it takes **O(n),** a linear time for **is\_prime(n), sum3\_prime,(n) sum4\_prime(n)**, and **sum5\_prime(n).** 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 **Chloe’s rules(d)** function where the number of iteration is the difference between 10n-1 to 10n -1. Fortunately, since transitory elements in ***temp*** array saves unnecessary steps of repetitive addition process by memoization, we can prevent entire running time from growing exponential. The final time complexity would be **q times O(n2)** depending on the input size *n.*

# ConclusionS

Dynamic Programming is very powerful and systematic in terms of splitting one clustered schema into different roles of sub-functions. Memoization which temporarily stores data which was already calculated from previous implementation contributes to this entire algorithm’s process efficiently by diminishing unnecessary and repetitive addition. Brute Force algorithm approach from which two previous algorithm got derived suggests very straightforward and categorial strategy. However, in terms of space and time complexity, Brute Force algorithm in this case does not appear to be the best solution due to the length of implementation and space size taken from this approach.

# REFERENCES

1. Levitin Anany. 2012. *Introduction to The Design & Analysis of Algorithms*. 3rd ed. Boston: Pearson Education, 2012. Print.
2. Sogo, K. (n.d.). 2016. Prime Digit Sums. HackerRank. (September 2016). Retrieved May 10, 2018 from https://www.hackerrank.com/contests/world-codesprint-8/challenges/prime-digit-sums
3. 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
4. Saumont, Pierre-Yves. "Do it in Java 8: Automatic memoization - DZone Performance."Dzone.com. N.p., 30 Sept. 2014. Web. 16 Apr. 2018.