CS3510

PROGRAMMING ASSIGNMENT 2

REPORT

by: ***DEVRAJ (ES22BTECH11011)***

DESIGN OF THE PROGRAM

FINDING IF A NUMBER IS A VAMPIRE NUMBER OR NOT

The function ***isVampireNumber(x)*** checks if ***x*** is a vampire number.

It does this in the following manner:

- First it counts the number of digits in ***x*** and stores the number of times each digit (0 to 9) is occurring in ***x***.

- It then checks each pair of factors of ***x*** that give back ***x*** on multiplication, if both of them have half the number of digits that ***x*** has and if they have the same digits (0 to 9) that ***x*** has.

- If these conditions are met for any pair of factors (that it encounters first), the function returns ***true***.

- If the function can't find any pair of factors that satisfy these conditions, it returns ***false***.

Implementation of Threading:

In this program, we have to check "***N***" numbers to see if they are vampire numbers or not.

To do this, in each execution, the program creates "***M***" threads to handle "***N/M***" numbers and to check if they are vampire numbers or not.

In each thread, the ***runner()*** function receives a set of numbers of size "***N/M***" and calls ***isVampireNumber()*** to find which among them are vampire numbers.

Since I needed multiple arguments for the ***runner()*** function, I created a structure containing the arguments.

The ***runner()*** function prints these vampire numbers into a file and adds to the total number of them found.

It also stores these numbers in a local array and transfers them to a global one before completion.

For an efficient implementation of threading:

Dividing the set of "***N***" numbers like ***(0, N/M), (N/M+1 , 2N/M), …., ((M-1)N/M+1, N)*** into "***M***" equal sets is not efficient since we know that there aren't many vampire numbers in the first 100000 natural numbers (only 7). This implies that the variance in the number of vampire numbers found by each thread will be quite large, i.e., the work is not efficiently divided between the threads.

To make the threading more efficient, I have divided the set of "***N***" numbers into "***M***" sets in the following manner:

{1, 1+M, 1+2M, …, 1+k_1M}; (1+k_1M <= N)

{2, 2+M, 2+2M, …, 2+k_2M}; (2+k_2M <= N)

.

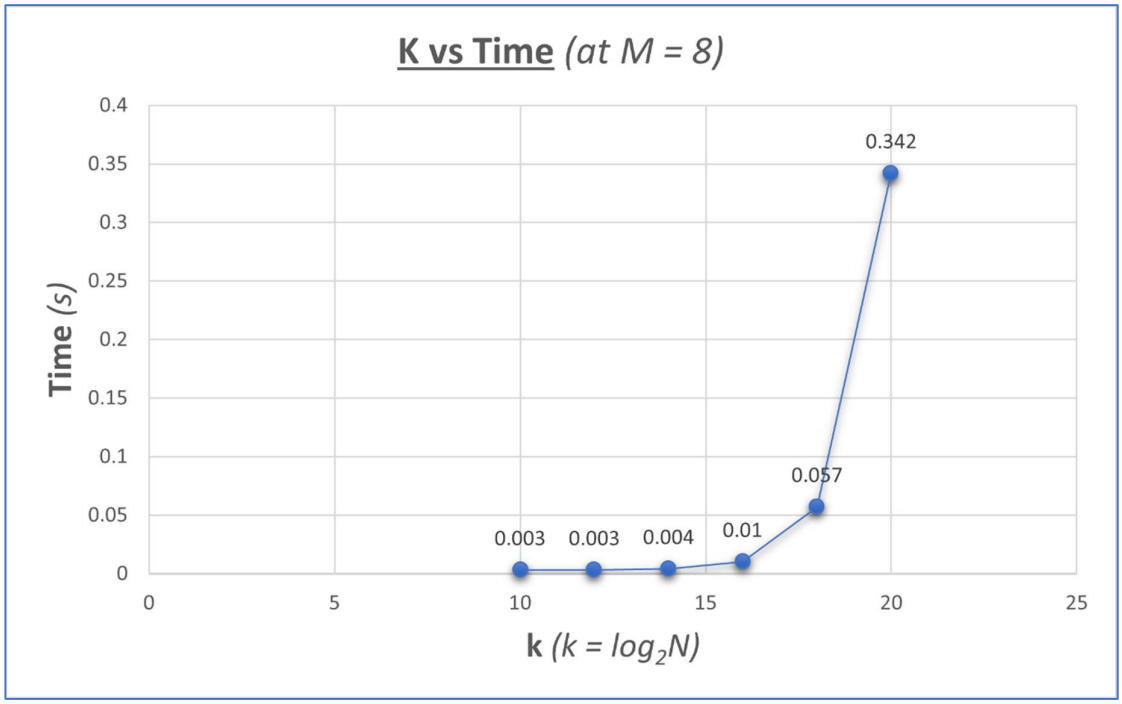
.

{M, 2M, …, k_MM}; (k_MM <= N)

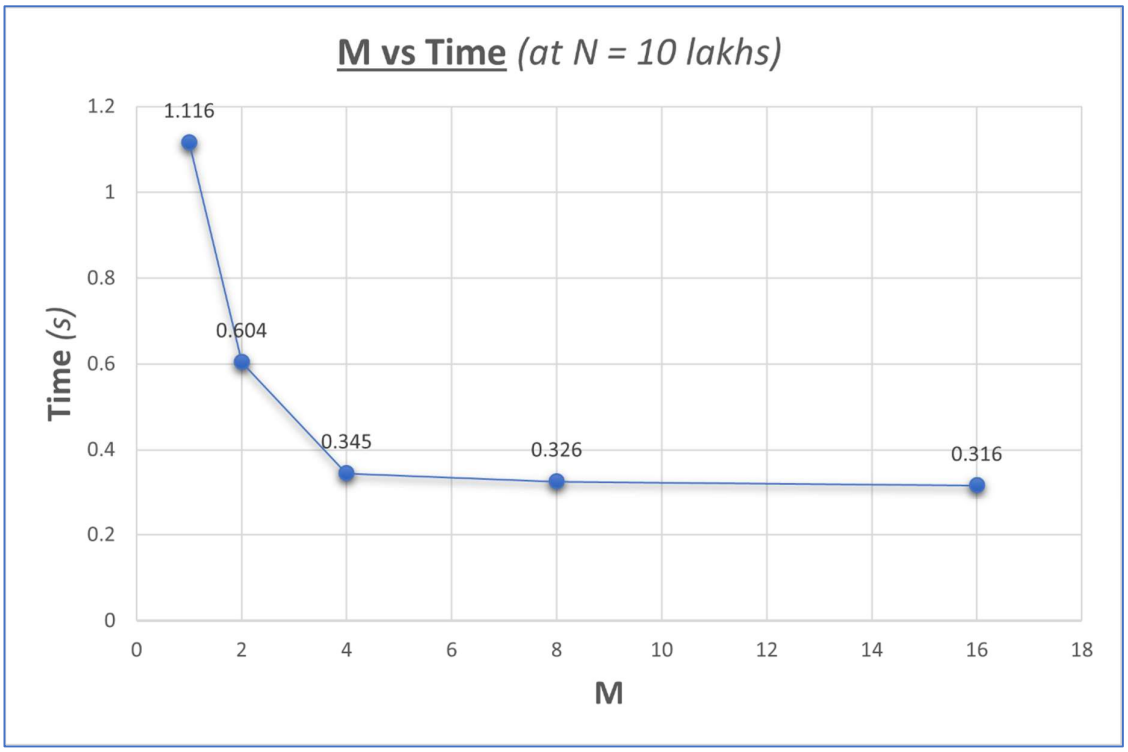
Each of these sets has size ~ "***N/M***".

In this method of division of numbers, each thread covers a set of numbers that is spread apart over the "***N***" numbers. So, the variance of the number of vampire numbers found by each thread is lower compared to the former method.

Performance of the program



The time of execution is well below a second for all values of ***k*** given and increases exponentially with increase in ***k*** (exponential increase in size "***N***").



Here, the time of execution is around a second and decreases exponentially with increase in the number of threads "***M***".