DAA Assignment 1

Priyal Gupta(IHM2017004), Aditya Gupta(IHM2017005),

Pratham Prakash Gupta(IHM2017003) and Rohit Ranjan(IHM2017001)

January 27,2019

1 Abstract

The problem is to find the smallest prime number in a given array. There are many algorithms to solve the above problem. We have to select the one which has least time complexity. For this we use the algorithm named Sieve of Eratosthenes. In this algorithm we precompute the prime numbers up to the maximum number of the array. Precomputation includes creating a boolean array and then the array block corresponding to a prime index is set to true. Then we check each number of the array whether it is prime or not and then find the minimum of the prime numbers. The time complexity of the above code is max(log(log(max)), where max is the largest number in the array.

2 Keywords

Prime: A number is said to be prime, if it is divisible only by 1 and itself.

3 Introduction

3.1 Question

Given an array of randomly generated positive integers of size 1000 such that, elements of the array range from 10000 to 1000000. Locate the smallest prime number.

3.2 Literature Survey

We have to develop an algorithm to find the smallest prime number in an array. There can be several ways to find the same. Our goal is to develop the most efficient algorithm with the least time complexity. We can either first check every number whether it is prime or not and then find the minimum number among them. This is the brute force solution of the above problem. But this algorithm will take much time if the numbers in the array are large. So, we precompute the prime numbers up to the maximum number of the array. The latter method is more efficient in terms of time complexity.

3.3 Idea

Here we have used the popular algorithm 'Sieve of Eratosthenes' for finding smallest prime number in a given array. A number is prime, if none of the smaller prime numbers divides it. Since we iterate over the prime numbers in order, we already marked all numbers, which are divisible by at least one of the prime numbers. Hence if we reach a cell and it is not marked, then it isn't divisible by any

smaller prime number and therefore has to be prime. We have first calculated the maximum number in the input array and used the algorithm for marking prime numbers in array of size equal to that number. Then among the prime numbers we found the minimum number.

4 Algorithm Design

4.1 Part a: Brute force algorithm

4.1.1 Parameters:

n: Size of the given array.
a[]: Array of size n.

4.1.2 Variables

flag: set when a number is composite.
min: stores the minimum number
i: used for storing array index
j: used for iterating over numbers up to n

4.1.3 Pseudo Code

```
min← 1000000
for   i ← 0  to  n − 1  do
    flag← 0
    for    j← 2  to  n − 1  do
        if(a[i]%j=0) then
            flag← 1
            break
    if(flag=0 and min≥ a[i])  then
    min← a[i]
print(min)
end
```

4.2 Part b: Using Sieve

4.2.1 Parameters:

n: Size of the given array.
a[]: Array of size n.

4.2.2 Variables

max: stores the maximum number in the array
min: stores the smallest prime number int the array(if exists)
i: used for storing array index
j: used for iterating over numbers up to n
prime[]: boolean array which stores true if the number is prime
ans: stores the final answer

4.2.3 Pseudo Code

```
sieve(a[],max,n)
    for    i← 0  to  max  do
        prime[i]← true
    prime[0]← false
    prime[1]← false
    for    i← 2  to  sqrt(max)  do
        j← 2 ∗ i
        if(prime[i]← true)  then
            while j≤ max  do
                prime[j]← false
                j← j + i
    min← 1000000
    for    i← 1  to  n  do
        if(prime(a[i])=true) then
            if(a[i]≥ min)then
                min← a[i]
    return min
main()
    max← 0
    for    i← 1  to  n  do
        if(max≤ a[i])then
            max← a[i]
    ans← sieve(a, max, n)
    if(ans=1000000) then
        print(Prime not found)
    else
        print(ans)
end
```

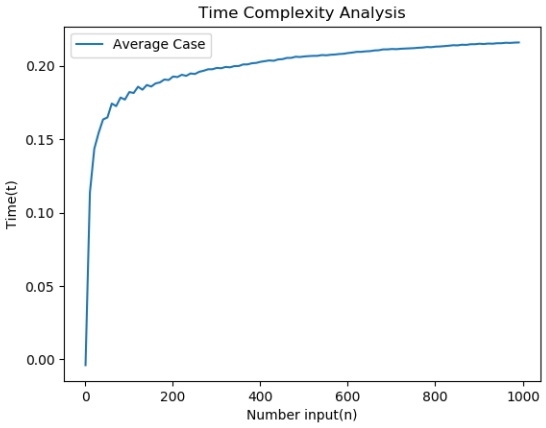
5 Analysis and Discussion

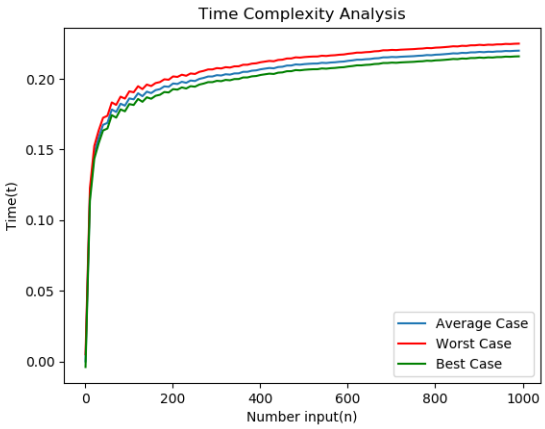
5.1 Time Complexity

The number of instructions executed in precomputation are of the order of max(log(log(max)), where max is the maximum number in the given array. This is the time complexity of sieve. Precomputation includes creating a boolean array and then the array block corresponding to a prime index is set to true. Further there are a number of instructions required for the above code. Though precomputation takes some extra time as it checks all the number less than the maximum number. But the overall time is much less than any other algorithm.

Best Case
Occurs when there is no prime number in the given array. This case will take 2*n operations less, as minimum value will not be assigned to ans.

Worst Case
This is the case when every number is prime and the largest number in the array is 999999. In this case the time complexity is 999999+999999(log(log(999999)))+3*n, where n is the number of elements in the array.





5.2 Space Complexity

In the solution space is used to store the 'prime' array which gives information about whether the index is prime or not. 'prime' array is a 1-D array of size equal to the largest number in input array. The space complexity of sieves generally grows in the order O(n).

6 Experimental Setup

Language Used :- C
Plotting tool :- matplotlib
Report Making Tool :- TextPortable(Latex)

7 Conclusion

Smallest prime number in a given array can be found by using sieve. In this method we first compute the maximum number in the given array. Then we create a boolean array of size equal to the maximum number and set the elements corresponding to prime index to 1. Then from this array we check whether a given number is prime or not and then find the minimum of the prime numbers.

8 References

Latex :- https://www.sharelatex.com/learn
Gnu plot :- https://matplotlib.org