An Exercise in Program Explanation

JAYADEV MISRA
The University of Texas at Austin

A combination of program-proving ideas and stepwise refinement is used to develop and explain an algorithm which uses a variation of the sieve method for computing primes.

Key Words and Phrases: primes, algorithms, proofs of programs
CR Categories: 5.24, 5.25, 5.29

1. INTRODUCTION

An algorithm that computes the set of prime numbers less than or equal to some given $n \geq 2$ is developed and explained in this paper. The algorithm employs a variation of the sieve technique; however, every nonprime is generated and removed precisely once from the set, resulting in an algorithm with running time linear in n. This efficiency is achieved at the expense of maintaining a more complex data structure and assuming that multiplication (of positive integers smaller than n) requires unit time. The algorithm is simpler to explain and prove than a linear algorithm appearing in [1].

The purpose of this note is to show how a combination of program-proving ideas and stepwise refinement can be used to describe and explain an algorithm completely. The explanation is given by first postulating a suitable invariant. Hypothesizing an invariant is one of the most creative tasks in program construction; however, program construction becomes almost purely mechanical given a suitable invariant. The explanation given here is adequate for a reader to construct his own formal proof.

For this problem Pritchard [4] reports an asymptotically sublinear algorithm which uses no multiplication.

2. PROBLEM DESCRIPTION AND THE INVARIANT

The problem is to construct

$$S = \{x \mid 2 \leq x \leq n, x \text{ prime}\} \qquad \text{for any given} \quad n \geq 2. \tag{2.1}$$

Author's address: Department of Computer Sciences, College of Natural Sciences, The University of Texas at Austin, Austin, TX 78712.

We actually construct the following set from which the set in (2.1) may be deduced.

$$S = \{x \mid 1 \le x \le n,\ x = 1 \text{ or } x \text{ prime}\}. \tag{2.2}$$

In order to describe the invariant, we define certain functions on any set of positive integers Z.

$\in$ ::	Denotes the usual membership test.	(2.3)
succ ::	For any $t \in Z$, $succ(t)$ is defined to be the next larger number than t in Z; $succ(t)$ is undefined if t is the largest in Z.	(2.4)
pred ::	For any $t \in Z$, $pred(t)$ is the next smaller number than t in Z; $pred(t)$ is undefined when t is the smallest in Z.	(2.5)
remove ::	For any $t \in Z$, $remove(t)$ sets Z to $Z - \{t\}$.	(2.6)

In addition, we define a function on integers greater than 1:

sd ::	For any integer $x > 1$, $sd(x)$ is the smallest integer greater than 1 that evenly divides x.	(2.7)

OBSERVATION 1. *$sd(x) = x$ iff x prime.*

The intuitive idea behind the algorithm is as follows. Starting with $p = 2$, every iteration removes all remaining multiples of p, a prime number, from the set S. p is then increased to the next prime number and the iterations are continued as long as p has some multiple in S. In order to remove all remaining multiples of p, we maintain r such that $p \cdot r \le n$ and r is the largest number in S having this property. We will show (1) that p has a multiple in S if and only if $p \le r$ and (2) that a multiple of p, say $p \cdot t$, is in S if and only if $p \le t \le r$ and $t \in S$.

The program is built around the invariant

$$\begin{aligned} S = \{x \mid 1 \le x \le n,\ x = 1 \text{ or } x \text{ prime or } sd(x) \ge p\} \\ \textbf{and } p \ge 2 \textbf{ and } p \in S \\ \textbf{and } r \text{ is the largest number in } S \text{ such that } p \cdot r \le n. \end{aligned} \tag{2.8}$$

We first show that this invariant is equivalent to another invariant which is easier to manipulate in proofs.

LEMMA 2. *Given the invariant* (2.8), *$succ(r)$ is defined and $p \cdot succ(r) > n$.*

PROOF. We use a rather deep theorem due to Chebyshev [2] which states that for any positive integer $i > 1$ there is a prime v, $i < v < 2i$. From the invariant, $p \ge 2$ and $p \cdot r \le n$. Hence $r \le \lfloor n/2 \rfloor$. If $\lfloor n/2 \rfloor = 1$, $r = 1$ and $succ(r) = 2$. Otherwise, according to Chebyshev's theorem there exists a prime y, $\lfloor n/2 \rfloor < y < 2 \cdot \lfloor n/2 \rfloor$, $y \in S$ and $y > r$. Therefore $succ(r)$ is defined. Since r is the largest number in S for which $p \cdot r \le n$, $p \cdot succ(r) > n$. □

Using Lemma 2 we may rewrite the invariant (2.8) as follows:

$$\begin{aligned} I :: \quad S = \{x \mid 1 \le x \le n,\ x = 1 \text{ or } x \text{ prime or } sd(x) \ge p\} \\ \textbf{and } p \ge 2 \textbf{ and } p \in S \\ \textbf{and } r \in S \textbf{ and } p \cdot r \le n < p \cdot succ(r). \end{aligned} \tag{2.9}$$

OBSERVATION 3. *Given I, p is a prime.*

PROOF. Since $p \geq 2$, $p \in S$, either p is prime or $sd(p) \geq p$. Hence p is prime in either case. □

3. TOWARD SYNTHESIS OF A PROGRAM

We postulate the following program structure:

initialize;
while B **do** loop body **od**; (3.1)

Initialization is easy. Setting

$$p := 2; \tag{3.2}$$

forces us to set

$$S := \{x \mid 1 \leq x < n\}; \tag{3.3}$$

$$r := \lfloor n/2 \rfloor; \tag{3.4}$$

We next consider the loop body portion, which must remove all the multiples of p in each iteration. To this end, we give a characterization of multiples of p in S, from which we derive the loop body and the condition B.

THEOREM 4. *Given I, $p \cdot t \in S$ and $t \geq 2$ if and only if $t \in S$ and $p \leq t \leq r$.*

PROOF. Suppose $p \cdot t \in S$ and $t \geq 2$. Since $p \cdot t$ is not prime, exceeds 1, and is in S, $sd(p \cdot t) \geq p$. Since $sd(p) = p$ (Observation 3), $sd(t) \geq p$.

Furthermore, since $p \cdot t \leq n$ and $t \geq 2$, $2 \leq t \leq n$. Therefore, $t \in S$. Also, $p \cdot t \leq n < p \cdot succ(r)$. Hence $t < succ(r)$; that is, $t \leq r$. $sd(t) \geq p$ means $t \geq p$. Therefore $t \in S$ and $p \leq t \leq r$.

Conversely, suppose $t \in S$ and $p \leq t \leq r$. $t \geq p$ means $t \geq 2$. Since $t \in S$, either t is a prime ($sd(t) = t \geq p$) or $sd(t) \geq p$. In either case, $sd(t) \geq p$. $sd(p) = p$; therefore $sd(p \cdot t) = p$. $p \cdot t \leq p \cdot r \leq n$; therefore $p \cdot t \in S$. □

This is the central theorem around which the sieve method works; by properly enumerating t—the elements of S between p and r—and removing $p \cdot t$ from S, we can guarantee that every nonprime will be generated and removed precisely once.

Theorem 4 implies that S has at least one nonprime if $p \leq r$. We prove a stronger result below which allows us to derive the condition B for execution of the loop.

THEOREM 5. *Given I, S has a nonprime >1 if and only if $p \leq r$.*

PROOF. If $p \leq r$, the result follows from Theorem 4. Conversely, we show if $p > r$, then S has no nonprime >1. Since $p \in S$, $r \in S$, and $p > r$, then $p \geq succ(r)$. Since $p \cdot succ(r) > n$, we have $p \cdot p > n$. Using this in conjunction with the definition of S in I, the result follows from an elementary result in number theory. □

We thus derive from Theorem 5 that condition B for the continuation of the loop is

$$B :: \quad p \leq r. \tag{3.5}$$

The partial program at this stage looks as follows.

Program : :
$p := 2$; $S := \{x \mid 1 \le x \le n\}$; $r := \lfloor n/2 \rfloor$;
while $p \le r$ **do** loop body **od**; (3.6)

4. SYNTHESIZING THE LOOP BODY

We postulate the following structure of the loop body.

loop body : :
remove all multiples of p from S;
reestablish I.

Note. As long as p increases in the loop body in each iteration, since $p \cdot r \le n$ the loop is bound to terminate.

4.1 Removing All Multiples of p from S

It follows from Theorem 4 that we can enumerate all $t \in S, p \le t \le r$, and remove $p \cdot t$ for every such t enumerated. However, the procedure would be incorrect if we enumerated t in *increasing order* from p to r and removed every $p \cdot t$. Consider, for instance, the situation in which $t' \in S$ **and** $p \le t' \le r$ **and** $p \cdot t' \le r$ **and** $p^2 \cdot t' \le n$. When t' is enumerated, $p \cdot t'$ will be removed, and hence $p^2 \cdot t'$ will never be removed. Gries and Misra [1] suggest the following solution.

> Enumerate $t \in S$ **and** $t \ge p$ in ascending order. For each t enumerated, remove all $p^k \cdot t \le n$, $k \ge 1$. The advantage of this strategy is that r need not be maintained; we simply stop the process when $p \cdot t > n$. Furthermore, there is a symmetry in enumeration: For fixed values of p and t we first enumerate k until $p^k \cdot t > n$; then t is increased to its next value in S and the above step is repeated until $p \cdot t > n$; then p is increased to its next value in S and the above steps are repeated until $p \cdot p > n$.

We propose the following simpler strategy since r is available to us.

> Enumerate $t \in S$ in decreasing order from r to p and remove $p \cdot t$ when t is enumerated.

Thus, the corresponding program for removing multiples of p looks like the following:

```
t := r;
while p ≤ t do
  remove(p·t);
  t := pred(t)
od                                   (4.1)
```

Correctness of (4.1) may be established by using the invariant

$$p \cdot q \in S, q \ge 2 \quad \text{iff} \quad q \in S, p \le q \le t. \tag{4.2}$$

Note. The only instance in which the presence of 1 in S is useful is when $p = 2$. The final iteration is started with $t = 2$, 4 is removed from S, and t is set to 1.

Note. It follows that at the end of (4.1) we can assert $S = \{x \mid 1 \le x \le n, x = 1$ or x prime or $sd(x) > p\}$ **and** $p \in S, p \ge 2$ **and** $p \cdot r \le n$ **and** $p \cdot$ (the next number larger than r in S) $> n$.

Note. We *cannot* assert following (4.1) that $r \in S$, since r might have been a multiple of p and hence might have been removed.

4.2 Reestablishing I

In order to establish I, we consider each component assertion of I in turn. $S = \{x \mid 1 \le x \le n, x = 1$ or x prime or $sd(x) \ge p\}$ can be established from $S = \{x \mid 1 \le x \le n, x = 1$ or x prime or $sd(x) > p\}$ by setting

$$p := succ(p) \tag{4.3}$$

This also preserves $p \ge 2$ and $p \in S$.

In order to reestablish $r \in S$, we perform a linear search.

$$\textbf{while } r \notin S \textbf{ do } r := r - 1 \textbf{ od} \tag{4.4}$$

We can assert on termination of (4.4) that $r \in S$ **and** $p \cdot succ(r) > n$, since $p \cdot$ (next larger number than r in S) $> n$ prior to execution of (4.4).

Finally, in order to establish $p \cdot r \le n$, we can employ a linear search in which r decreases more rapidly than in (4.4):

$$\textbf{while } p \cdot r > n \textbf{ do } r := pred(r) \textbf{ od} \tag{4.5}$$

Note. (4.5) does not disturb the truth of propositions $r \in S$ **and** $p \cdot succ(r) > n$.

Note. (4.4) cannot employ $r := pred(r)$ since *pred* is applicable only for r in S.

Note. Termination proofs for both (4.4) and (4.5) are straightforward and hence left to the reader.

5. THE COMPLETE ALGORITHM

```
p := 2; S := {x | 1 ≤ x ≤ n}; r := ⌊n/2⌋;
while p ≤ r do
   t := r;
   while p ≤ t do
      remove(p·t);
      t := pred(t)
   od;
   p := succ(p);
   while r ∉ S do r := r − 1 od;
   while p·r > n do r := pred(r) od
od
```

6. DISCUSSION

6.1 Data Structure for S

Several different data structures for S have been proposed in [1], each of which is applicable in the algorithm proposed here. The simplest is representing S by a doubly linked list. Each elementary set operation can be performed in unit time on such a structure. See Misra [3] for a more careful choice of the data structure which takes the total number of bits of storage and array accessing time into account.

6.2 Running Time Estimation

We show that no statement in the program is executed more than n times. This is certainly true for $p := succ(p)$, which strictly increases p each time it is executed, and p cannot increase more than once beyond $\lceil\sqrt{n}\rceil$. Similarly, $r := r - 1$ and $r := pred(r)$ are executed at most n times in total. $remove(p \cdot t)$ removes one nonprime from S, and hence it and $t := pred(t)$ cannot be executed more than n times. Similar remarks apply to the tests in the loops.

6.3 A Conjecture

The algorithm could be simplified if the following conjecture were true.

Conjecture 6. Given I, r and $pred(r)$ cannot both be multiples of p.

We may thus initially save

$$r1 := pred(r) \tag{6.1}$$

After removal of multiples of p, we can reestablish $r \in S$ by the following, since both r and $r1$ could not have been removed:

$$\textbf{if } r \notin S \textbf{ then } r := r1 \textbf{ fi}; \tag{6.2}$$

Note that $p \cdot succ(r) > n$ after this step.

We may then reestablish $p \cdot r \leq n$ by

$$\textbf{while } p \cdot r > n \textbf{ do } r := pred(r) \textbf{ od}; \tag{6.3}$$

The complete algorithm then is the following.

```
p := 2; S = {x | 1 ≤ x ≤ n}; r = ⌊n/2⌋;
while p ≤ r do
   r1 := pred(r); t := r;
   while p ≤ t do
      remove(p·t);
      t := pred(t)
   od;
   p := succ(p);
   if r ∉ S then r := r1 fi;
   while p·r > n do r := pred(r) od
od                                                    (6.4)
```


REFERENCES

1. GRIES, D., AND MISRA, J. A linear sieve algorithm for finding prime numbers. *Commun. ACM 21*, 12 (Dec. 1978), 999–1003.
2. LEVEQUE, W.J. *Topics in Number Theory*, Vol. 1. Addison-Wesley, Reading, Mass., 1956.
3. MISRA, J. Space time trade-off in implementing certain set operations. *Inf. Process. Lett. 8*, 2 (Feb. 1979), 81–85.
4. PRITCHARD, P. A sublinear additive sieve for finding prime numbers. Tech. Rep. No. 10, Dep. Computer Science, Univ. Queensland, Australia, Dec. 1979.

Received October 1979; revised March and September 1980; accepted October 1980