Robert Bennion's "Hopping Sieve"

William F. Galway
Department of Mathematics
University of Illinois at Urbana-Champaign
Urbana, IL 61801

galway@math.uiuc.edu
http://www.math.uiuc.edu/~galway

DRAFT Fri Jun 19 15:30:07 CDT 1998

Introduction

Let

$$\mathcal{P}_Y = \{p : \ p \text{ is prime, } p \leq Y\} \ , \quad \text{and}$$
$$S = \#\mathcal{P}_Y = \pi(Y) \ ,$$

and suppose we want to sieve a range of numbers n, in the interval $n_0 \leq n < n_0 + L$, eliminating those n which are multiples of any prime $p \in \mathcal{P}_Y$.

If we think of the natural numbers as a sequence of numbered tiles, the "classical" sieve of Eratosthenes works by crossing out tiles (one bit per tile).

In Bennion's algorithm, we think of each tile as holding a prime from $\mathcal{P}_Y$ (one prime per tile). As the algorithm progresses, primes hop forward, preferably to the next multiple of p . . . .

DRAFT Fri Jun 19 15:30:07 CDT 1998

The Rules

To limit memory usage, we only consider a block of "tiles" of length S, running from $n \ldots n + S - 1$. Tile number n passes through the sieve $\iff$ the prime p at n divides n. This prime p becomes "the current prime".

Then, the current prime p hops forward, to the next multiple of p if that tile is within the block, and either

- it lands on the last tile of the block, and we have finished updating the sieve, or
- it lands somewhere else, displacing the prime previously stored there, which becomes the new "current prime", and we repeat...

DRAFT Fri Jun 19 15:30:07 CDT 1998

The Sieve in Action

$$n_0 = 15, \quad L = 11,$$
$$\mathcal{P}_Y = \{2, 3, 5\}, \quad S = 3.$$

m	p_m
15	5
16	2
17	3
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	

DRAFT Fri Jun 19 15:30:07 CDT 1998

Building the Sieve

$S = 1$

m	p_m
15	

$S = 2$

m	p_m
15	
16	

$S = 3$

m	p_m
15	
16	
17	

DRAFT Fri Jun 19 15:30:07 CDT 1998

Data Structures

```
typedef struct HoppingSieve {
  int n;      /* The "current number".  */
  int n0;     /* Starting value for n.  */
  int S;      /* Number of entries in pbuf.  */
  int *pbuf;
} HoppingSieve;

/* Given HoppingSieve *sv,
 * convert m to "the prime at m".
 */
#define P(sv,m) sv->pbuf[((m) - (sv->n0))%(sv->S)]
```

Code to Advance the Sieve

```
int AdvanceSieve(HoppingSieve *sv)
{
  int m, tmp, p;
  int rslt;

  m = sv->n;
  p = P(sv,m);
  if (m%p == 0)
    rslt = 0;
  else
    rslt = 1;

  while (1) {
    m += p - m%p;
    if (m >= sv->n + sv->S)
      break;
    /* p hops into place, displaces previous prime.  */
    tmp = p;  p = P(sv,m);  P(sv,m) = tmp;
  }
  P(sv, sv->n) = p;
  sv->n++;
  return rslt;
}
```


DRAFT Fri Jun 19 15:30:07 CDT 1998

Running Time

Inside the loop in `AdvanceSieve`, p lands at a multiple m of p, $m \in \mathcal{I}$, where $\mathcal{I} = [n_0, n_0 + L + S)$. The number of "hop and displace" operations inside the loop is bounded by

$$\sum_{p \le Y} \sum_{\substack{m \in \mathcal{I} \\ p|m}} 1 = \sum_{p \le Y} \left(O(1) + \frac{L+S}{p} \right)$$

$$= O(S) + (L+S) \sum_{p \le Y} \frac{1}{p}$$

$$= O(S+L) + (L+S) \ln \ln(Y) \ ,$$

since $S = \pi(Y)$, and $\sum_{p \le Y} \frac{1}{p} = O(1) + \ln \ln(Y)$.

If we assume $L \ge S$, this is $O(L \ln \ln(Y))$, i.e. $O(\ln \ln(Y))$ operations per number sieved.

DRAFT Fri Jun 19 15:30:07 CDT 1998

Code to Build the Sieve

```
/* Initialize a pre-allocated sieve structure
 * to start at n0.
 */
void InitSieve(HoppingSieve *sv, int n0,
               int size, int *primes)
{
  int  m, p, tmp;

  sv->n = sv->n0 = n0;
  for (sv->S = 1; sv->S <= size; sv->S++) {
    p = primes[sv->S - 1];
    m = n0 + (p-n0%p)%p;
    while (m < n0 + sv->S - 1) {
      /* Hop and displace.  */
      tmp = p;  p = P(sv,m);   P(sv,m) = tmp;
      m += p - m%p;
    }
    P(sv, n0 + sv->S - 1) = p;
  }
  sv->S = size;
}
```


DRAFT Fri Jun 19 15:30:07 CDT 1998

Time to Build Sieve

Given a precomputed set of primes $\mathcal{P}_Y$, the number of "hop and displace" operations performed in the inner loop of `InitSieve` is

$$O(S \ln \ln(Y)) = O\left(\frac{Y \ln \ln(Y)}{\ln(Y)}\right) \ .$$

(Normally this would be dominated by the time to find $\mathcal{P}_Y$.)

DRAFT Fri Jun 19 15:30:07 CDT 1998

Speeding Things Up

We can eliminate remaindering operations by storing the distance d to the next multiple of p along with p.

m	p	d
15	5	0
16	2	0
17	3	1
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		

DRAFT Fri Jun 19 15:30:07 CDT 1998

Factoring Consecutive Numbers

If each "tile" holds a list of of divisors and a list of leftovers, we can factor numbers using a "hopping factor sieve".

m	divisors	leftovers
15	3, 5	2
16		
17		
18		
19		
20		
21		
22		
23		

DRAFT Fri Jun 19 15:30:07 CDT 1998

Can we Beat the Competition?

To eliminate multiples of primes, or to factor numbers, the hopping sieve requires $O(Y)$ bits of storage, and $O(L \ln \ln(Y))$ arithmetic operations, which is of the same order as required by the "segmented sieve" of Bays and Hudson for finding primes.

We can also factor numbers using a sieve similar to the segmented sieve, with the same asymptotic cost as for the hopping sieve, but storage management appears to be more difficult.

For both factoring and finding primes, the hopping sieve *may* be better suited for a cached memory.

DRAFT Fri Jun 19 15:30:07 CDT 1998