Chapter 10: Addition Machines

[Written with Robert W. Floyd. Originally published in SIAM Journal on Computing 19 (1990), 329-340.]

It is possible to compute $gcd(x,y)$ efficiently with only $O(logxy)$ additions and subtractions, when three arithmetic registers are available but not when there are only two. Several other functions, such as $x^y \ mod \ z$, are also efficiently computable in a small number of registers, using only addition, subtraction, and comparison.

Main Algorithms

An addition machine is a computing device with a finite number of registers, limited to the following six types of operations:

read x	$\{input\ to\ register\ x\}$
x $\leftarrow y$	$\{copy\ register\ y\ to\ register\ x\}$
x $\leftarrow x+y$	$\{add\ register\ y\ to\ register\ x\}$
if x $\geq y$	$\{compare\ register\ x\ to\ register\ y\}$
write x	$\{output\ from\ resister\ x\}$

The register contents are assumed to belong to a given set A, which is an additive subgroup of the real numbers. If A is the set of all integers, we say the device is an integer addition machine; if A is the set of all real numbers, we say the device is a real addition machine.

We will consider how efficiently an integer addition machine can do operations such as multiplication, division, greatest common divisor, exponentiation, and sorting. We will also show that any addition machine with at least six registers can compute the ternary operation x[y/z] with reasonable efficiency, given x, y, z $\in A$ *with* $z \neq 0$.

Remainder

As a first example, consider the calculation of

$$\text{x mod y} = \begin{cases} x - y[x/y], & if\ y \neq 0; \\ x, & if\ y = 0; \end{cases}$$

This binary operation is well defined on any additive subgroup A of the reals, and we can easily compute it on an addition machine as follows:

read x; **read** y; z $\leftarrow z - z$;
if y $\geq z$ **then**
  if z $\geq y$ **then** $\{y = 0, do\ nothing\}$
  else if x $\geq z$ **then while** $x \geq y$ **do** $x \leftarrow x - y$
    else repeat x $\leftarrow x + y$ **until** $x \geq z$
else if z $\geq x$ **then while** $y \geq x$ **do** $x \leftarrow x - y$
    else repeat x $\leftarrow x + y$ **until** $z \geq x$;
write x.

(There is implicitly a finite-state control. A pidgin Pascal program such as this one is easily convert ed to other formalisms).

A Fibonacci Method

Remainders can, however, be computed with the desired efficiency $O(log(x/y))$ if we implicitly use the Fibonacci representation of $[x/y]$ instead of the binary representation. Define Fibonacci numbers as usual by

$$F_0 = 0;\ F_1 = 1;\ F_{n-1} + F_{n-2}, \qquad for\ n \geq 2$$

Every non-negative integer n can be uniquely represented in the form

$$n = F_{l_1} + F_{l_2} + ... + F_{l_t}, \qquad l_1 \gg l_2 \gg ... \gg l_t \gg 0,$$

where $t \geq 0$ and $l \gg l'$ means that $l - l' \geq 2$. If $n > 0$, this representation can be found by choosing l_1 such that

$$F_{l_1} \leq n < F_{l_1+1},$$

so that $n - F_{l_1} < F_{l_1+1} - F_{l_1} = F_{l_1-1}$, and by writing

$$n = F_{l_1} + \text{(Fibonacci representation of } n - F_{l_1}).$$

We shall let

$$\lambda n = l_1 \qquad and \qquad \upsilon n = t$$

denote respectively the index of the leading term and the number of terms, in the Fibonacci representation of n. By convention, $\lambda 0 = 1$.

Fibonacci numbers are well suited to addition machines because we can go from the pair $\langle F_l, F_{l+1} \rangle$ up to the next pair $\langle F_{l+1}, F_{l+2} \rangle$ with a single addition, or down to the previous pair $\langle F_{l-1}, F_l \rangle$ with a single subtraction. Furthermore, Fibonacci numbers grow exponentially, about 69% as fast as powers of 2. They have been used as power-of-2 analogs in a variety of algorithms (see, for instance, "Fibonacci numbers" in the index to), and in Matijasevic's solution to Hilbert's tenth problem.

Multiplication and Division

We can use essentially the same idea to compute the ternary operation $x[y/z]$ efficiently on any addition machine. This time we accumulate multiples of x as we discover the Fibonacci representation of $[y/z]$:

read x; **read** y; **read** z; $\{assume\ that y \geq 0\ and\ z > 0\}$
$\omega \leftarrow 0$;
if $y \geq z$ **then**
  begin $l \leftarrow 1$;
  repeat $l \leftarrow l + 1$ **until** $y < zF_{l+1}$;
  repeat if $y \geq zF_{l+1}$ **then** $\langle \omega, y \rangle \leftarrow \langle \omega + xF_l, y - zF_l \rangle$;
    $l \leftarrow l - 1$;
  until $l = 1$;
  end;
write ω.

Greatest Common Divisors

We can therefore use our method for computing $x \ mod \ y$ to calculate $gcd(x,y)$ on an integer addition machine:

read x; **read** y; $\{assume\ that\ x > 0\ and\ y \geq 0\}$
$z \leftarrow y$; $z \leftarrow z + z$;
while not $y \geq z$ **do** $\{equivalently,\ y > 0,\ since\ z = 2y\}$
  begin while $x \geq z$ **do** $\langle y, z \rangle \leftarrow \langle z, y + z \rangle$;
  repeat if $x \geq y$ **then** x$\leftarrow x - y$;
    $\langle y, z \rangle \leftarrow \langle z - y, y \rangle$;
  until $y \geq z$;
  $\langle x, y \rangle \leftarrow \langle y, x \rangle$; $z \leftarrow y$; $z \leftarrow z + z$;
  end;
write x.

(Here the operation $\langle x,y \rangle \leftarrow \langle y,x \rangle$ should not really be performed; it means that the roles of registers x and y should be interchanged. The implementation jumps between six copies of this program, one for each permutation of the register names x, y, z.)

Stacks and Sorting

Euclid's algorithm defines a one-to-one correspondence between pairs of relatively prime positive integers $\langle x,y \rangle$ with $x > y$ and sequences of positive integers $\langle q_1, ..., q_m \rangle$ where each $q_1 \geq 1$ and $q_m \geq 2$. We can push a new integer q onto the front of such a sequence by setting $\langle x,y \rangle \leftarrow \langle qx + y, x \rangle$; we can pop $q_1 = \lfloor x/y \rfloor$ from the front by setting $\langle x,y \rangle \leftarrow \langle y, x \ mod \ y \rangle$.

Therefore an integer addition machine can represent a stack of arbitrary depth in two of its registers. The operation of pushing or popping a positive integer q can be done with $O(logq)$ operations, using a few auxiliary registers.

Here, for example, is the outline of an integer addition program that reads a sequence of positive integers followed by zero and writes out those positive integers in reverse order:

$\langle x,y \rangle \leftarrow \langle 2,1 \rangle$; $\{the\ empty\ stack\}$
repeat read q;
  if $q \geq 1$ **then** $\langle x,y \rangle \leftarrow \langle qx + y, x \rangle$;
until not q≥ 1;
repeat $\langle q,x,y \rangle \leftarrow \langle \lfloor x/y \rfloor, y, x \ mod \ y \rangle$;
  if $y \geq 1$ **then write** q;
until not $y \geq 1$.

This program uses the algorithms for multiplication and division shown earlier. The total running time to reverse the input $\langle q_1, q_2, ..., q_m, 0 \rangle$ is $O(m + logq_1q_2...q_m)$.

We can sort a given list of positive integers $\langle q_1, q_2, ..., q_m \rangle$ in a similar way, using the classical algorithms for merge sort ing with th ree or more magnetic tapes that can be "read backwards". The basic operations required are esseptially those of a stack; so we can sort in $O((m + logq_1q_2 \ ... \ q_m) \ log \ m)$ steps if there are at least 12 registers.