Session 57: Arithmetic with Base 2 Expansions

- Addition
- Multiplication
- Modular Exponentiation

Binary Addition of Integers

procedure *add*(a, b: positive integers)

{the binary expansions of a and b are $(a_{n-1}, a_{n-2},\ldots, a_0)_2$ and $(b_{n-1}, b_{n-2},\ldots, b_0)_2$, respectively}

$c := 0$

for $j := 0$ **to** $n - 1$

    $d := \lfloor (a_j + b_j + c)/2 \rfloor$

    $s_j := a_j + b_j + c - 2d$

    $c := d$

$s_n := c$

return$(s_0, s_1,\ldots, s_n)$

{the binary expansion of the sum is $(s_n, s_{n-1},\ldots, s_0)_2$}

The number of additions of bits used by the algorithm to add two n-bit integers is $O(n)$.

Example

Adding $a = (1110)_2$ and $b = (1011)_2$.

Binary Multiplication

Two observations

$a \cdot b = a \ \cdot (b_k 2^k + b_{k-1} 2^{k-1} + ... + b_1 2 + b_0) = ab_k 2^k + ab_{k-1} 2^{k-1} + ... + ab_1 2^1 + ab_0 2^0$

Multiplying a binary number by 2^j is corresponds to add j zeros at the end

Example: Multiply a = $(110)_2$ and b = $(101)_2$

Binary Multiplication of Integers

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}, a_{n-2},\ldots, a_0)_2$ and $(b_{n-1}, b_{n-2},\ldots, b_0)_2$, respectively}
for $j := 0$ **to** $n - 1$
    if $b_j = 1$ **then** $c_j = a$ with j zeros appended
    else $c_j := 0$
{$c_o, c_1,\ldots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ **to** $n - 1$
  $p := p + c_j$
return p

The number of additions of bits used by the algorithm to multiply two n-bit integers is $O(n^2)$.

Binary Modular Exponentiation

In cryptography, it is important to be able to find b^n **mod** m efficiently, where b, n, and m are large integers.

- Use the binary expansion of n, $n = (a_{k-1}, \ldots, a_1, a_o)_2$, to compute b^n .

  Note that:

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1\cdot 2} \cdot b^{a_0}.$$

- Therefore, to compute b^n, we need only compute the values of

$$b, b^2, (b^2)^2 = b^4, (b^4)^2 = b^8, \ldots, b^{2^{k-1}}$$

- and then multiply the terms b^{2^j} in this list, for all $a_j = 1$.

Example

Example: Compute 3^{11} using this method.

Note that $11 = (1011)_2$ so that

$3^{11} = 3^8\ 3^2\ 3^1 =$

$((3^2)^2\ )^2\ 3^2\ 3^1 =$

$(9^2\ )^2 \cdot 9 \cdot 3 =$

$(81)^2 \cdot 9 \cdot 3 =$

$6561 \cdot 9 \cdot 3 =$

$117{,}147.$

Binary Modular Exponentiation Algorithm

The algorithm successively finds

$$b \textbf{ mod } m,\ b^2 \textbf{ mod } m,\ b^4 \textbf{ mod } m,\ \ldots,\ b^{2^{k-1}} \textbf{ mod } m,$$

and multiplies together the terms b^{2^j} where $a_j = 1$.

```
procedure modular exponentiation(b: integer, n = (a_{k-1}a_{k-2}...a_1a_0)_2 , m: positive integers)
x := 1
power := b mod m
for i := 0 to k − 1
    if a_i = 1 then x := (x· power ) mod m
    power := (power· power ) mod m
return x
```

$O((\log m)^2 \log n)$ bit operations are used to find b^n **mod** m.

Summary

- Binary addition, multiplication, modular exponentiation