Avancerade algoritmer DD2440

Homework B

Jim Holmström - 890503-7571

November 7, 2011

Exercise 1. Using the 0/1-principle for sorting networks, either show that the following sorting network correctly sorts 4 inputs, or find an input where it fails.

Solution. 0/1-principle states that it will correctly sort any input if it can sort all possible sequences of $0, 1$. This reduces the combination of input to check in an n-wire network from $n!$ to 2^n.
In our case we will need to validate $2^4 = 16$ combinations.

The comparators will only swap values if they are misplaced, thus all combinations already sorted will come out the same way (sorted).

The comparators only swaps values so $\sum x = \sum x'$ (where $\bar{x}$ =input and $\bar{x}'$ =output)

$\forall x : \sum x = 0, 4$
Trivially $\bar{x} = 0000$ and $\bar{x} = 1111$ will still be sorted.
$\forall x : \sum x = 1$
The one set input-value will "bubble" down from position 1,2 and 3 (and already be sorted for position 4).
$\forall x : \sum x = 3$
By symmetry this also works in the same way as above but with the one unset input-value "bubble" up. $\forall x : \sum x = 2$
$\bar{x} = 0011$ is already sorted. Those 5 left is put in an matrix for batch calculations, if some rows become equal after a pass only one of them is considered since they will all act the same from there on.

$$\tilde{X}_2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix}0&0&1&1&1\\1&1&0&0&1\\0&1&0&1&0\\1&0&1&0&0\end{pmatrix} = \{\text{1-pass: compare 1-2 and 3-4}\} = \begin{pmatrix}0&0&0&0&1\\1&1&1&1&1\\0&0&0&0&0\\1&1&1&1&0\end{pmatrix} = \begin{pmatrix}0&1\\1&1\\0&0\\1&0\end{pmatrix} =$$

$$= \{\text{2-pass: compare 2-3}\} = \begin{pmatrix}0&1\\0&0\\1&1\\1&0\end{pmatrix} = \{\text{3-pass: compare 1-2 and 3-4}\} = \begin{pmatrix}0&0\\0&1\\1&0\\1&1\end{pmatrix} =$$

$$= \{\text{4-pass: compare 2-3}\} = \begin{pmatrix}0&0\\0&0\\1&1\\1&1\end{pmatrix}$$

And this proves that all input with $\sum x = 2$ will be sorted.

Now we have proven that all 16 combinations of binary input will be sort and by the 0/1-principle the net will be a valid sorting net for all comparable input. □

Exercise 2. Crack RSA! Given the public RSA key $(n, e) = (5192042249411, 3419183406883)$**, find the decryption exponent** d **and find a number** x **that encrypts to your ten-digit personal number.**

Solution. Firstly we factor the semiprime $n = pq$ using a simple implementation of Pollard-rho with $f(x) = x^2 + c$ picking $c = 1$ and $x_0 = (2, 2)$

```
function pollard_rho(n):
    d = 1,x = (f(x0[0])%n,f(f(x0[1])%n)
    while d==1
        x = (f(x[0])%n,f(f(x[1])%n)
        d = gcd(abs(x[1]-x[0]),n)
    return (d,n/d)
```

We get $n = 5192042249411 = 1532371 \cdot 3388241 = pq$

$$\varphi(n) = \varphi(pq) = \{\text{p,q primes}\} = (p-1)(q-1) = 5192037328800$$

And the private key d is:

$$d = e^{-1} \ (mod\ \varphi(n))$$
$$ed = 1 \ (mod\ \varphi(n))$$
$$ed - K\varphi(n) = 1$$

d is then calculated by the extended euclidean algorithm to be $\underline{d = 1456182194347}$ and one can verify that it is indeed a modular inverse to e. (K is irrelevant)

$$c = m^e \ (mod\ n) \tag{1}$$

$$m = c^d \ (mod\ n) \tag{2}$$

Since we are dealing with large exponents one cannot use the naive exponentiation instead we use modular exponentiation by repeated squaring to bring down the complexity. The main idea is this:

$$a^{2n} \equiv (a^n)^2 \ (mod\ m) \tag{3}$$

$$a^{2n+1} \equiv a^1(a^n)^2 \ (mod\ m) \tag{4}$$

An implementation:

```
function superpower(x,n,m):
    ans = 1
    while n: //int2bin
        n_bits.append(n%2)
        n>>=1
    n_bits.reverse()

    for bit in n_bits:
        ans = (ans*ans)%m
        if bit: #"odd"
            ans = (ans*x)%m
    return ans
```

To find the number x that encrypts to my personal number (8905037571) we need to go the other way by decrypting.
$m = c^d \ (mod\ n) = \texttt{superpower}(c, d, n) = \{c = 8905037571\} = \underline{2534986786188}$

Exercise 3. Multiply the following polynomials using Karatsuba's algorithm. $x^3 + 3x^2 + x$ and $2x^3 - x^2 + 3$.
You only need to show the top-most call, what recursive calls are made from the top-most level, what those calls return and how the final result is computed.

Solution. Karatsuba's algorithm is used as base.

$$(ax+b)(cx+d) = \underbrace{(a\cdot c)}_{=u}x^2 + \underbrace{(a\cdot d + b\cdot c)}_{=w-u-v}x + \underbrace{(b\cdot d)}_{=v}$$

$$\begin{cases} u = a\cdot c \\ v = b\cdot d \\ w = (a+b)\cdot(c+d) = ac+bc+ad+bd \end{cases} \tag{5}$$

All higher order polynoms can be broken up this way:

$$\begin{aligned} p(x) &= \sum_{0}^{2t-1} p_i x^i = \underbrace{\sum_{0}^{t-1} p_i x^i}_{b(x)} + x^t \underbrace{\sum_{0}^{t-1} p_{t+i} x^i}_{a(x)} = a(x)x^t + b(x) \\ q(x) &= \{\text{The same way as above}\} = c(x)x^t + d(x) \end{aligned} \tag{6}$$

Then taking $p(x)q(x)$ yields:

$$\begin{aligned} p(x)q(x) =& (a(x)x^t + b(x))(c(x)x^t + d(x)) = \{x^t \leftrightarrow x\ ,\ \text{Karatsuba's algorithm}\} = \\ =& \underbrace{(a(x)c(x))}_{=u(x)}x^{2t} + \underbrace{(a(x)d(x) + b(x)c(x))}_{=w(x)-u(x)-v(x)}x^t + \underbrace{(b(x)d(x))}_{=v(x)} \end{aligned} \tag{7}$$

Our polynoms $(x^3+3x^2+x-1)(2x^3-x^2+3)$ is represented as $(1,3,1,-1)$ and $(2,-1,0,3)$ in memory, x^i's coefficient is the variable at position i. All operations like addition and subtraction can be done termwise.

First split the polynoms in the middle (trivial if one looks at the memory representation):

$$(x^3+3x^2+x-1)(2x^3-x^2+3) = \{(6)\} = \underline{((x+3)x^2 + (x-1))((2x-1)x^2 + 0x + 3)} \tag{8}$$

Then executing Karatsuba's algorithm

$$\begin{cases} u(x) = (x+3)(2x-1) = \{\text{Recursive call to Karatsuba's algorithm}\} = \underline{2x^2+5x-3} \\ v(x) = (x-1)(0x+3) = \{\text{Recursive call to Karatsuba's algorithm}\} = \underline{3x-3} \\ w(x) = (2x+2)(2x+2) = \{\text{Recursive call to Karatsuba's algorithm}\} = \underline{4x^2+8x+4} \end{cases}$$

Karatsuba's algorithm then returns
$(a(x)x^2 + b(x))(c(x)x^2 + d(x)) = \{(5)\} = u(x)x^4 + (w(x) - u(x) - v(x))x^2 + v(x)$
$= (2x^2 + 5x - 3)x^4 + (2x^2 + 10)x^2 + (3x - 3) = \underline{2x^6 + 5x^5 - x^4 + 10x^2 + 3x - 3}$

Exercise 4. A positive odd integer n is a Carmichael number if

(a) it is composite

(b) $\forall$ numbers $1 \leq a < n : gcd(a, n) = 1$, it holds that $a^{n-1} \equiv 1 \ (mod\ n)$

Show that it is easy to find a non-trivial factor of a Carmichael number. More precisely, there is a polynomial-time algorithm for finding a non-trivial factor with probability $> 1/2$.
Explain your algorithm, why it works, and analyze its complexity.

Solution. Miller-Rabin Probabilistic Primality test for n basically goes like:
`LOOP` :repeat k times:
$x = random(1, n-1)$
Let $n - 1 = 2^q m$ where $m \in 2\mathbb{Z} + 1$
If $x^m \equiv 1 \ (mod\ n)$ (not needed here) $\bigvee \exists i \in [0, q-1] : x^{2^i m} \equiv -1 \ (mod\ n)$
then "inconclusive" (do next `LOOP`) else return "n is composite"
return "prob. prime"

Knowing that n is Carmicheal number the only intresting part in Miller Rabin is the $x \leftarrow x^2 \equiv 1 \ (mod\ n)$ (equivalent to the $x^{2^i m}$-part but more cunning) wich is foreach 2 factor in $(n-1)$

The algorithm:
When Miller-Rabin hits then gives us that the base x that fullfills $x^2 \equiv 1 \ (mod\ n) \bigwedge x \not\equiv \pm 1$ (so let us use this)

$\Rightarrow (x+1)(x-1) \equiv 0 \ (mod\ n)$, that is n is a factor of $(x+1)(x-1)$ and thus n has a non-trivial common factor with at least one of $(x+1)$ and $(x-1)$ since neither is a multiple of n. Then you use Euclid's Algorithm to find this common factor by `euclid`$(n, x \pm 1)$ And thus must return a factor $\square$

Probability:
The Miller-Rabin algorithm is known to at most return inconclusive $1/4$ of the bases x. This holds for any composite number n including Carmichael numbers. And the fermat-test part won't add any information since n is guarenteed to be Carmichael (so it's not needed in this algorithm), therefore the algorithm will get it right $\underline{3/4 > 1/2}$

Complexity:
For one pass $(k = 1)$:
Firstly the search for x is done foreach 2-factor $(\exists i : x^{2^i m})$ and the worstcase number of 2-factors is when it's only 2-factors and thus $O(log(n))$ Euclidean is running in $O(log(n)^2)$ and since it's called once foreach 2-factor $(O(log(n)))$ giving us a total complexity of $\underline{O(log(n)^3)}$