Shor's Factoring Algorithm

Carsten Urbach

In order to break RSA cryptography one needs to be able to factorise a large integer n, which is known to be the product of two prime numbers $n = pq$.

Factoring Algorithm

Given an integer n, the factoring algorithm determines p, q such that $n = pq$. We assume $p, q \neq 1$. The following is Shor's algorithm (Shor 1997) for factoring:

1. Choose $m, 1 \leq m \leq n$ uniformnly random with m co-prime to n.
2. Find the order r of m modulo n.
3. If r is even, compute $l = \gcd(m^{r/2} - 1, n)$
4. If $l > 1$ then l is a factor of n. Otherwise, or if r is odd start with 1 for another value of m.

Greatest common divisor

Euclid described a classical algorithm for finding the greatest common divisor (gcd) of two positive integers $m > n$. It may be implemented recursively as follows:

```
gcd <- function(m, n) {
  if(m < n) {
    return(gcd(m=n, n=m))
  }
  r <- m %% n
  cat(r, m, n, "\n")
  if(r == 0) return(n)
  return(gcd(m=n, n=r))
}
```

Order finding

Another ingredient is the order finding algorithm, which we are also going to solve classically here, actually with the most naive algorithm

```
findOrder <- function(x, n) {
  stopifnot(x < n && x > 0)
  tmp <- x %% n
  x <- tmp
  for(r in c(1:n)) {
    if(tmp == 1) return(r)
    tmp <- (tmp*(x %% n)) %% n
  }
  if(tmp == 1) return(r)
  return(NA)
}
```

Factoring

Shor's algorithms can be implemented as follows

```
factoring <- function(n) {
  for(i in c(1:20)){
    ## generate random number
    m <- sample.int(n=n, size=1)
    cat("m=", m, "\n")
    ## Check, whether m, n are co-prime
    g <- gcd(n,m)
    if(g != 1 ) return(g)
    else {
      ## find the order of m modulo n
      r <- findOrder(x=m, n=n)
      cat("r=", r, "\n")
      if(!is.na(r)) {
        if((r %% 2) == 0) {
          l <- gcd(m^(r/2)-1, n)
          if(l > 1 && l < n) return(l)
        }
      }
    }
  }
  cat("could not find a factor!\n")
  return(NA)
}
```

And we can test whether it works

```
set.seed(81)  ## for reproducibility
factoring(65)
```

```
m= 25
15 65 25
10 25 15
5 15 10
0 10 5
```

```
[1] 5
```

```
factoring(91)
```

```
m= 86
5 91 86
1 86 5
0 5 1
r= 12
63 404567235135 91
28 91 63
7 63 28
0 28 7
```

```
[1] 7
```

```
factoring(511)
```

```
m= 504
```

```
7 511 504
0 504 7
```

```
[1] 7
```

Note that this computation is a bit tricky in `R` because of the integer arithmetic with large integers. However, for our example here, the code is sufficient.

References

Shor, Peter W. 1997. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." *SIAM Journal on Computing* 26 (5): 1484–1509. https://doi.org/10.1137/s0097539795293172.