UNIVERSITY OF LIVERPOOL

Department of Computer Science
Sebastian Wild

Date: 2020-03-11
Version: 2020-03-11 17:38

Tutorial 5 for COMP 526 – Applied Algorithmics, Winter 2020

—including solutions—

It is highly recommended that you first try to solve the problems on your own before consulting the sample solutions provided below.

Problem 1 (Periodicity lemma)

Prove the periodicity lemma:

If string $S = S[0..n-1]$ has periods p and q with $p + q \le n$, then it has also period $\gcd(p, q)$.

Solutions for Problem 1 (Periodicity lemma)

Euclid's algorithm for computing the greatest common divisor of two positive integers is famous example in algorithmic number theory. The main idea behind Euclid's algorithm is a recursive principle, where for two integers $p \ge q$, we have

$$\gcd(p,q) \quad = \quad \begin{cases} 1, & \text{for } q = 1; \\ p, & \text{for } q = p; \\ \gcd(p-q, q), & \text{otherwise.} \end{cases} \tag{1}$$

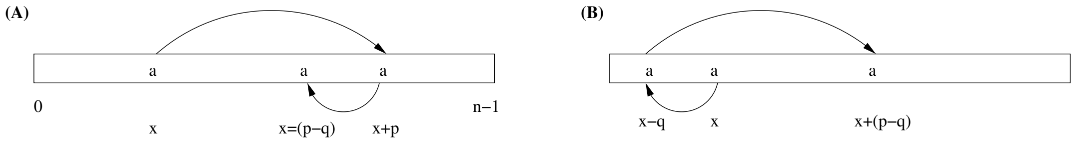
Extra material: proof of (1)
The first two cases are immediate. The third case can be seen as follows. Let $x = \gcd(p,q)$*, where* $p > q$*. Then* $p = p' \cdot x$ *and* $q = q' \cdot x$ *for some integers* p' *and* q'*. But note that* $p - q = p' \cdot x - q' \cdot x = (p' - q') \cdot x$*, i.e.,* x *is also a divisor of* $p - q$*. We still have to prove that* x *is also a* greatest *common divisor. Assume towards a contradiction that there is an integer* $y > x$ *s. t.* $y = \gcd(p, p-q)$*. This means that* $q = q'' \cdot y$ *and* $p - q = r \cdot y$*, for some integers* q'' *and* r*. But then we can also show that*

$p = q + (p - q) = q'' \cdot y + r \cdot y = (q'' + r) \cdot y$ meaning that y is also a divisor of p. We obtain a contradiction to the assumption that $x < y$ is the greatest common divisor of p and q.

We use same recursive observation (1) to prove the statement of the periodicity lemma, following the inductive proof of correctness of Euclid's algorithm, i.e., we show that if a string S has two periods $p > q > 1$, where $p + q \le n$, it also has the smaller period $p - q$.

From the definition of periods p and q we know that $S[i] = S[i+p]$, for all $i = 0, \dots, n - p - 1$ (or alternatively $S[i-p] = S[i]$ for all $i = p, \dots, n$) and $S[j] = S[j+q]$, for all $j = 0, \dots, n - q - 1$ (or alternatively $S[j-q] = S[j]$, for all $i = q, \dots, n$).

We have to prove that $S[x] = S[x + (p-q)]$, for all $x = 0, \dots, n - (p-q) - 1$. We consider two regimes for x:



(A) Consider first indices $x = 0, \dots, n - p - 1$.

From the definition of the period p we know that $S[x] = S[x+p]$ within this range. Note also that for all x in this range $S[x+p] = S[x+p-q]$ due to the alternative definition of the period q and the fact that $x + p > q$. Now since $S[x+p] = S[x]$ we conclude that $S[x] = S[x + (p-q)]$, for all $x = 0, .., n - p - 1$.

(B) Now consider indices $x = n - p, \dots, n - (p-q) - 1$.

From the alternative definition of q and from the assumption that $p + q \le n$ we learn that $S[x-q] = S[x]$ within this range (i.e., index $x - q$ never goes below 0). Also the value of $x - q + p \le n$, for all $x = n - p, \dots, n - (p-q) - 1$. Thus from the definition of the period p we get $S[x-q] = S[x-q+p]$ in this range. Thus we obtain $S[x] = S[x + (p-q)]$ also in this range.

Problem 2 (Parallel And)

We consider the problem of computing the logical *and* of an array $B[0..n-1]$ of n Boolean values (n bits), i.e., the result should be *true* if and only if all n entries are true. (We assume here that each bit is stored as a full word.)

a) Design a CREW-PRAM parallel algorithm for computing the "logical and" of $B[0..n-1]$. Your algorithm should have $\mathcal{O}(\log n)$ time (span) and $\mathcal{O}(n \log n)$ work.

b) Can you make the algorithm work-efficient?

c) Now consider a CR**C**W-PRAM; you can choose a write-conflict resolution rule that is convenient for your purposes. Design a *constant-time* parallel algorithm for computing the logical and.

Solutions for Problem 2 (Parallel And)

a) The key observation is that we can use the parallel prefix sum algorithm, and simply replace the summation (in each step) by a logical and.

   This approach indeed generalizes to any *associative* binary operation.

   Note that the prefix sum algorithm actually computes more than we asked for; it also computes the logical and of all prefixes of B.

b) There are two easy ways to obtain a work-efficient algorithm. First of all, notice that the sequential problem has (worst-case) complexity $\Theta(n)$ since we have to read the entire input, so we aim for linear work.

   The simplest way is to use a work-efficient prefix sum algorithm.

   Another way is to use the fact that we only need the and of the entire array; we can therefore simulate a single complete binary tree, where in round k only $n/2^k$ PEs are active. The total work is hence linear (geometric sum).

c) On a CREW-PRAM, we cannot improve beyond the logarithmic complexity of information collection / dissemination.

   If concurrent writes are allowed, though, the following simple algorithm solves the problem in constant time, and using any of our discussed write-conflict rules (in particular the weakest one, "common"):

```
CRCWparallelAnd(B[0..n − 1])
    o := true
    for i = 0, ..., n − 1 do in parallel
        if B[i] == false
            o := false
    return o
```

   Note that this trick is much less general, but it can be used, e.g., to compute in constant time whether two strings are equal.