CS 4823: Homework #2

Due on February 2, 2018

Christopher Tse

Problem 1

Compute $gcd(269, 35)$ and find its representation (i.e. x and y such that $269x + 35y = gcd(269, 35)$). Show step-by-step calculations.

Solution

Finding GCD
Using Euclidean Division Algorithm:

$$\begin{aligned} 269 &= 35(7) + 24 \\ 35 &= 24(1) + 11 \\ 24 &= 11(2) + 2 \\ 11 &= 2(5) + 1 \\ 2 &= 1(2) + 0 \end{aligned}$$

Last non-zero remainder is 1, therefore $gcd(269, 35) = 1$.

Finding Linear Combination
Using the Extended Euclidean Algorithm:

$$\begin{aligned} 1 &= 11 - 2(5) \\ &= 11 + 2(-5) \\ &= 11 + (24 + 11(-2))(-5) \\ &= 11(11) + 24(-5) \\ &= (35 + 24(-1))(11) + 24(-5) \\ &= 35(11) + 24(-16) \\ &= 35(11) + (269 + 35(-7))(-16) \\ 1 &= 35(123) + 269(-16) \end{aligned}$$

The linear combination can be represented as $269x + 35y = gcd(269, 35)$ where $x = -16$ and $y = 123$.

Problem 2

Consider the following Sage/Python program to compute the gcd of two integers a and b (assume that $a > b > 0$):

```
for i in range(b,0,-1):
    if a % i == 0 and b % i == 0:
        print i
        break
```

Is the algorithm correct? Is it efficient? Justify your answer.

Solution

Since $a > b$, the gcd can only be at most b. By starting to test every value from b to 1, the algorithm will be correct by breaking once it encounters the largest number that both a and b are divisible by, in other words the gcd of a and b. However, this is not an efficient algorithm since for very large b (and subsequently very large a) it will have to test every single number from b through 0 in a brute force manner, with the worst case being that it tests all numbers until it reaches 1, making the worst case time complexity $O(b)$.

Problem 3

Consider the following Sage/Python program to compute the gcd of two integers a and b (assume that $a > b > 0$):

```
while b != 0:
    a = a - b
    if a < b:
        a, b = b, a
print a
```

Is the algorithm correct? Is it efficient? Justify your answer.

Solution

This algorithm is correct as proven in Euclidean Algorithm. This is based on the principle that the gcd does not change if a is replaced by the difference $a - b$ (i.e., $gcd(a, b) = gcd(a - b, b)$). Therefore, by executing this algorithm until a and b are the same (causing b to become 0 in the final iteration), it will return the gcd of the two numbers. While this method can be efficient and greatly reduce the number of steps for many cases, in the worst case the time complexity can still be $O(a)$ when $b = 1$ (i.e., $gcd(a, 1)$).