Bardas Alexandru-Cristian, 321

In this program, i will demonstrate 3 algorithms that compute the greatest common divisor between two natural numbers.

First method : Stein's Algorithm

The algorithm found by Stein in 1967 reduces the problem of calculcating the GCD of 2 nonnegative numbers by repeatedly applying the following identities:

1. $\gcd(0, v) = v$, because everything divides 0, and v is the largest number that divides v (similarly $\gcd(0, u) = u$)
2. $\gcd(2u, 2v) = 2\gcd(u, v)$
3. $\gcd(2u, v) = \gcd(u, v)$ if v is odd.Similarly, $\gcd(u, 2v) = \gcd(u, v)$ if u is odd.
4. $\gcd(u, v) = \gcd(|u - v|, \min(u, v))$, if u, v are both odd .

```
<<Stein Algorithm>>=
def steinGCD(u, v) :
        #Checking the base case u = v
        if u == v:
                return u
        #Checking the first identity
        if u == 0:
                return v
        if v == 0:
                return u
        #We are looking for factors of 2, checking the second and third identities
        if u % 2 == 0:
                if v % 2 == 0: #both even
                        return 2*steinGCD(u // 2, v // 2)
                else: #u is even, v is odd
                        return steinGCD(u // 2, v)
        if v % 2 == 0: #Checking if u is odd and v even
                return steinGCD(u , v // 2)
        #We will now reduce the larger argument
        if u > v:
                return steinGCD(u - v, v)
        return steinGCD(v - u, u)

@
```

Second method : The Euclidean Algorithm

Procedure : The Euclidean algorithm proceeds in a series of steps such that the output of each step is used as an input for the next one. Let k be an integer that counts the steps of the algorithm, starting with zero. Thus, the initial step corresponds to $k = 0$, the next step corresponds to $k = 1$, and so on. Each step begins with two nonnegative remainders r_{k-1} and r_{k-2}. Since the algorithm ensures that the remainders decrease steadily with every step, r_{k-1} is less than its predecessor r_{k-2}. The goal of the k-th step is to find a quotient q_k and remainder r_k that satisfy the equation : $r_{k-2} = q_k * r_{k-1} + r_k$ and that have $0 \leq r_k \leq r_{k-1}$. In other words, multiples of the smaller number r_{k-1} are subtracted from the larger number r_{k-2} until the remainder r_k is smaller than r_{k-1}. In the initial step ($k = 0$), the remainders r_{-2} and r_{-1} equal a and b, the numbers for which the GCD is sought. In the next step ($k = 1$), the remainders equal b and the remainder r_0 of the initial step, and so on. Thus, the algorithm can be written as a sequence of equations :

$a = q_0 * b + r_0$

$b = q_1 * r_0 + r_1$

$r_0 = q_2 * r_1 + r_2$

$r_1 = q_3 * r_2 + r_3$

$\vdots$

If a is smaller than b, the first step of the algorithm swaps the numbers. For example, if $a < b$, the initial quotient q_0 equals 0, and the remainder r_0 is a. Thus, r_k is smaller than its predecessor $r_{k-1}, \forall k \geq 0$. Since the remainders decrease with every step but can never be negative, a remainder r_N must eventually equal 0, at which point the algorithm stops. The final nonzero remainder r_{N-1} is the greatest common divisor of a and b. The number N cannot be infinite because there are only a finite number of nonnegative integers between the initial remainder r_0 and 0.

```
<<Euclidean Algorithm>>=
def euclidGCD(a, b) :
        #checking base case
        if a == b:
                return a
        #checking the first identity from Stein's Algorithm
        if a == 0:
                return b
        #We checked it only for it,because the loop that will do the calcultions for the euc
        while b != 0:
                t = b #Temporary variable used to store the value of b
                b = a % b # Calculating the reminder
                a = t # a gets the value of b, which is the last reminder
        return a
@
```

Proof of validity :
The validity of the Euclidean algorithm can be proven by a two-step argument. In the first step, the final nonzero remainder r_{N-1} is shown to divide both a and b. Since it is a common divisor, it must be less than or equal to the greatest

common divisor g. In the second step, it is shown that any common divisor of a and b, including g, must divide r_{N-1}; therefore, g must be less than or equal to r_{N-1}. These two conclusions are inconsistent unless $r_{N-1} = g$. To demonstrate that r_{N-1} divides both a and b (the first step), r_{N-1} divides its predecessor $r_{N-2} : r_{N-2} = q_N * r_{N-1}$ since the final remainder r_N is 0. r_{N-1} also divides its next predecessor $r_{N-3} : r_{N-3} = q_{N-1} * r_{N-2} + r_{N-1}$ because it divides both terms on the right-hand side of the equation. Iterating the same argument, r_{N-1} divides all the preceding remainders, including a and b. None of the preceding remainders r_{N-2}, r_{N-3}, etc... divide a and b, since they leave a remainder. Since r_{N-1} is a common divisor of a and b, $r_{N-1} \leq g$.

In the second step, any natural number c that divides both a and b (in other words, any common divisor of a and b) divides the remainders r_k. By definition, a and b can be written as multiples of c : $a = m * c$ and $b = n * c$, where m and n are natural numbers. Therefore, c divides the initial remainder r_0, since $r_0 = a - q_0 * b = m * c - q_0 * n * c = (m - q_0 * n) * c$. An analogous argument shows that c also divides the subsequent remainders r_1, r_2, etc.... Therefore, the greatest common divisor g must divide r_{N-1}, which implies that $g \leq r_{N-1}$. Since the first part of the argument showed the reverse ($r_{N-1} \leq g$), it follows that $g = r_{N-1}$. Thus, g is the greatest common divisor of all the succeeding pairs: $g = gcd(a, b) = gcd(b, r_0) = gcd(r_0, r_1) = \cdots = gcd(r_{N-2}, r_{N-1}) = r_{N-1}$.

Third method : Dijkstra's Algorithm

This algorithm is developed by Dijkstra who is a Dutch mathematician and a computer scientist. The idea of this algorithm is : if $m > n$, GCD(m, n) is the same as GCD$(m - n, n)$.

Proof : If m/d and n/d leave no reminder (i.e. $m = p*d$ and $n = q*d$ for some integers p, q) then $(m-n)/d$ leaves no reminder ,i.e. $(m-n)/d = (p*d-q*d)/d = d*(p-q)/d = p-q$.

```
<<Dijkstra Algorithm>>=
def dijkstraGCD(m, n):
        #Check base case m = n
        if m == n:
                return m
        #Do the check indicated by Dijkstra
        if m > n:
                return dijkstraGCD(m - n, n)
        else:
                return dijkstraGCD(m, n - m)
@
```

Last part : Testing the algorithms

```
<<Testing units>>=
import time
def test():
        #Test inputs
        A = [ [1, 2],[4, 6], [12, 16], [63, 128] , [384, 2048], [8192, 65536], [729, 128142]
         [1221, 1234567891011121314151617181920212223242526272829], [53667, 25527]]
        #Result vector for the test inputs
        B = [1, 2, 4, 1, 128, 8192,  81, 6, 3, 201]
        steinTime = []
        euclidTime = []
        dijkstraTime = []
        for i in range(0, 10):
                start = time.time()
                rez = steinGCD(A[i][0], A[i][1])
                end = time.time()
                assert(rez == B[i])
                steinTime.append(end - start)

                start = time.time()
                rez = euclidGCD(A[i][0], A[i][1])
                end = time.time()
                assert(rez == B[i])
                euclidTime.append(end - start)

                if i < 6: # because dijkstra's algorithm runs out of stack calls due to rec
                    start = time.time()
                    rez = dijkstraGCD(A[i][0], A[i][1])
                    end = time.time()
                    assert(rez == B[i])
                    dijkstraTime.append(end - start)

        #Print results
        print("Stein's Algorithm Time for the given inputs : \n")
        print(*steinTime, sep = ',')
        print("\n Euclid's Algorithm Time for the given inputs : \n")
        print(*euclidTime, sep = ',')
        print("\n Dijkstra's Algorithm Time for the given inputs : \n")
        print(*dijkstraTime, sep = ',')

#Run tests when program is ran from main module
if __name__ == "__main__":
        test()

#End of program
```

```
@

<<*>>=
<<Stein Algorithm>>
<<Euclidean Algorithm>>
<<Dijkstra Algorithm>>
<<Testing units>>

@
```