SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2024

Topic Notes: Counting Operations

Counting Basic Operations: Bubble Sort

A sorting algorithm so simplistic, you might not have even studied it previously is called *bubble sort*. Here's a version.

ALGORITHM BUBBLESORT(A)
  //Input: an array $A[0..n-1]$
  for $i \leftarrow 0..n-2$ **do**
    for $j \leftarrow 0..n-2$ **do**
      if $A[j+1] < A[j]$ **then**
        swap $A[j+1]$ and $A[j]$

Let's simulate it! We will need someone to be each of the 5 array elements, someone to be `i`, someone to be `j`, someone to count comparison operations, and someone to count swap operations.

The comparison and swap operations are examples of *basic operations* of the algorithm. We often wish to count the number of times these basic operations occur to help us analyze the cost of an algorithm. We will focus here on the comparison, since it happens every time the inner loop iterates.

(To be computed in class: each `for` loop becomes a summation!)

Improving Bubble Sort

When we simulated the version above, there were some comparisons made that we know never would result in any swaps.

Let's think about how we can change the bounds of the inner loop to avoid making those comparisons:

ALGORITHM IMPROVEDBUBBLESORT(A)
  //Input: an array $A[0..n-1]$
  for $i \leftarrow 0..n-2$ **do**
    for $j \leftarrow 0..$ ☐ **do**
      if $A[j+1] < A[j]$ **then**
        swap $A[j+1]$ and $A[j]$

In a lab exercise, you will analyze the improved version.

Counting Basics

In class, we will work through the summations practice handout.

Example: Greatest Common Denominator

We next consider a very simple but surprisingly interesting example: computing a greatest common denominator (or divisor) (GCD).

Recall the definition of the GCD:

The gcd of m and n is the largest integer that divides both m and n evenly.

For example: gcd(60,24) = 12, gcd(17,13) = 1, gcd(60,0) = 60.

One common approach to finding the gcd is *Euclid's Algorithm*, specified in the third century B.C. by Euclid of Alexandria.

Euclid's algorithm is based on repeated application of the equality:

$$\text{gcd}(m,n) = \text{gcd}(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

More precisely, application of Euclid's Algorithm follows these steps:

Step 1 If $n = 0$, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value of the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

As pseudocode:

ALGORITHM EUCLID(m, n)
    // m and n non-negative, not both 0
    while $n \neq 0$ **do**
        $r \leftarrow m \bmod n$
        $m \leftarrow n$
        $n \leftarrow r$

Tracing through for our example of gcd(60,24) (using the steps 1, 2, 3 specification rather than the pseudocode):

Step 1 $n \neq 0$, go to Step 2

Step 2 $r \leftarrow 60 \bmod 24$ which is 12. Proceed to Step 3

Step 3 $n \leftarrow 24$, $m \leftarrow 12$. Proceed to Step 1

Step 1 $n \neq 0$, go to Step 2

Step 2 $r \leftarrow 24 \bmod 12$ which is 0. Proceed to Step 3

Step 3 $n \leftarrow 12$, $m \leftarrow 0$. Proceed to Step 1

Step 1 $n = 0$, return 12

It may not be obvious at first that this algorithm must terminate.

How can we convince ourselves that it does?

- the second number (n) gets smaller with each iteration and can never become negative
- so the second number in the pair eventually becomes 0, at which point the algorithm stops.

Euclid's Algorithm is just one way to compute a GCD. Let's look at a few others:

Consecutive integer checking algorithm: check all of the integers, in decreasing order, starting with the smaller of the two input numbers, for common divisibilty.

Step 1 Assign the value of min$\{m,n\}$ to t

Step 2 Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4

Step 4 Decrease t by 1 and go to Step 2

This algorithm will work. It always stops because every time around, Step 4 is performed, which decreases t. It will eventually become t=1, which is always a common divisor.

Let's run through the computation of gcd(60,24):

Step 1 Set t=24

Step 2 Divide m=60 by t=24 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=23, proceed to Step 2

Step 2 Divide m=60 by t=23 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=22, proceed to Step 2

Step 2 Divide m=60 by t=22 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=21, proceed to Step 2

Step 2 Divide m=60 by t=21 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=20, proceed to Step 2

Step 2 Divide m=60 by t=20 and check the remainder. It is 0, so we proceed to Step 3

Step 3 Divide n=24 by t=20 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=19, proceed to Step 2

Step 2 Divide m=60 by t=19 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=18, proceed to Step 2

Step 2 Divide m=60 by t=18 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=17, proceed to Step 2

Step 2 Divide m=60 by t=17 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=16, proceed to Step 2

Step 2 Divide m=60 by t=16 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=15, proceed to Step 2

Step 2 Divide m=60 by t=15 and check the remainder. It is 0, so we proceed to Step 3

Step 3 Divide n=24 by t=15 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=14, proceed to Step 2

Step 2 Divide m=60 by t=14 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=13, proceed to Step 2

Step 2 Divide m=60 by t=13 and check the remainder. It is not 0, so we proceed to Step 4

Step 4 Set t=12, proceed to Step 2

Step 2 Divide m=60 by t=12 and check the remainder. It is 0, so we proceed to Step 3

Step 3 Divide n=24 by t=12 and check the remainder. It is 0, so we return t=12 as our gcd

However, it does not work if one of our input numbers is 0 (unlike Euclid's Algorithm). This is a good example of why we need to be careful to specify valid inputs to our algorithms.

Another method is one you probably learned in around 7th grade.

Step 1 Find the prime factorization of m

Step 2 Find the prime factorization of n

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd(m,n)

So for our example to compute gcd(60,24):

Step 1 Compute prime factorization of 60: 2, 2, 3, 5

Step 2 Compute prime factorization of 24: 2, 2, 2, 3

Step 3 Common prime factors: 2, 2, 3

Step 4 Multiply to get our answer: 12

While this took only a total of 4 steps, the first two steps are quite complex. Even the third is not completely obvious. The description lacks an important characteristic of a good algorithm: *precision*.

We could not easily write a program for this without doing more work. Once we work through these, it seems that this is going to be a more complicated method.

We can accomplish the prime factorization in a number of ways. We will consider one known as the *sieve of Eratosthenes* (see the algorithm in Levitin Chapter 1, p. 6–7.

Given this procedure to determine the primes up to a given value, we can use those as our candidate prime factors in steps 1 and 2 of the middle school gcd algorithm. Note that each prime may be used multiple times.

So in this case, the seemingly simple middle school procedure ends up being quite complex, since we need to fill in the vague portions.