Chapter

Application of Discrete Mathematics for Programming Discrete Mathematics Calculations

Carlos Rodriguez Lucatero

Abstract

In the discrete mathematics courses, topics, such as the calculation of the element in any position of a sequence of numbers generated by some recurrence relation, calculation of multiplicative inverses in algebraic ring structures modulo a number *n*, obtaining the complete list of combinations without repetition, for which you can take advantage of the computing power of computers and perform such calculations using computer programs in some programming language. The implementations of these calculations can be carried out in many ways and therefore their algorithmic performance can be very varied. In this chapter, I propose to illustrate by means of some Matlab programs, how the use of results of the same discrete mathematics allows to improve the algorithmic performance of said computer programs. Another topic addressed in regular discrete mathematics courses where calculations arise that could become very expensive both in time and in occupied space, if the calculations are implemented directly from the definitions is modular arithmetic. Such calculations can be carried out much more efficiently by making use of results from discrete mathematics and number theory. The application of these ideas will be developed in the following sections of this chapter.

Keywords: recurrence relations, algorithms, generating functions, modular arithmetic, Matlab

1. Introduction

Discrete mathematics provides very useful calculation tools to enumerate mathematical objects of a certain type, such as the number of graphs that meet a certain particular property or how many regions will be formed within a circle as the number of points that grows, we will join by means of secant lines [1]. The methods and tools of discrete mathematics are of enormous relevance in the field of computer science when one wants to compare different algorithmic solutions to a problem. The goodness of an algorithmic solution must take into account, the use of runtime resources and memory space since they are limited. For this, it is necessary to carry out an analysis of the algorithm and count how many execution steps, such processing takes and how many memory locations it will occupy. When carrying out this analysis, mathematical expressions known as recurrence relationships are obtained, which reflect the behavior at execution time or the memory space occupied by the algorithm. Once the recurrence relationship is obtained, it can be evaluated to estimate, for example, how many executions steps an algorithm will take as the number of input data increases. This generates a succession of values that can eventually be graphed to give us an idea of the temporal complexity of such an algorithm. Such evaluation can be done by hand or it can be implemented, in some programming language available, a function. Normally the direct translation of the mentioned recurrences to a program produces compact and clear recursive routines. The evaluation of such routines is inefficient since many of the recursive calls repeat calculations and it is in this case that it is worth looking for more efficient iterative versions of these calculations. Even more, if possible, we can solve the recurrences to obtain a closed mathematical expression that describes the behavior of the growth in the time of the algorithm. That is where the tools of discrete mathematics acquire special relevance. In algorithm analysis books, one can find many techniques to solve certain types of recurrences and we can illustrate by giving some examples in this chapter. Either way, there are more general discrete mathematics tools for this purpose that we will also try to illustrate with examples. This process of solving recurrences can be seen as a generalization of the solution of classical problems. Some problems consist of finding the next element in a sequence of numbers and for which there are techniques, such as the method of divided differences. Sometimes obtaining the closed solution of a recurrence is not possible, however, discrete mathematics and mathematical analysis allow to define upper and lower bounds as well as approximate calculations of very good quality.

Another topic addressed in discrete mathematics courses, which requires the performance of many calculations, is that of modular arithmetic. Such calculations can be carried out much more efficiently by making use of results from discrete mathematics and number theory. The application of these ideas will be developed in the following sections of this chapter.

2. Body of the manuscript

The chapter will have the following structure. In Section 3, we will make a quick revision of some calculations problems in combinatorics. After that, in Section 4, we will talk about where the recurrence relations arise and some methods as well as mathematical tools that are frequently used to solve them. In each example, we will begin by obtaining the mathematical relationship that describes the behavior of the problem to be solved. Once the mathematical relationship is obtained, we will use it to give us an idea of the behavior it describes by evaluating it by means of some function programmed in Matlab. Later, we will obtain the associated mathematical expression or a good approximation function that we will evaluate implementing a Matlab function. This will allow us to compare the performance in the time of the evaluation of an expression and the direct application of a recurrence by implementing both in Matlab.

In Section 5, we will describe characteristics or properties of ring algebraic structures on the set of positive integers modulo some number n and directly from the definitions, we will obtain the tables of the two operations of the ring in question. Later, we will obtain the same tables more efficiently from the application of properties and theoretical results of modular arithmetic to illustrate the advantage of making use of the results of modular arithmetic to make calculations of modular arithmetic algorithmically more efficient both in time and space.

3. Combinatorics calculation

One of the most relevant topics in discrete mathematics is combinatorial analysis, where, among other things, it is important to calculate the number of all possible linear arrangements of a given size of objects of a set with repetitions or without repetitions. In some cases, the order of appearance of these objects must be taken into account. When the order of the elements is important and it is allowed to have a repetition of them, we are talking about permutations with repetition. If the repetition of elements is forbidden, then we are talking about permutations without repetition. To clarify ideas, suppose that we have a set $A = \{a, b, c, d\}$ and that we want to calculate all possible linear arrangements of 3-position elements allowing repetitions of elements in each of the positions. The cardinality of the set of objects A is 4 and as each of the three positions can be occupied by any of the elements of A then the total number of possible dispositions will be $4 \times 4 \times 4$ that is to say $4^3 = 64$. The resulting list of possibilities appear in **Table 1**.

This table was generated by the following Matlab function.

```
function y = enumeraPCR(A)
% Autor: Carlos Rodriguez Lucatero
[m,n]=size(A);
cuenta=0;
  for i=1:n
    for j=1:n
        for k=1:n
          fprintf('%s,%s,%s \n',A(1,i),A(1,j),A(1,k));
          cuenta=cuenta+1;
        end
    end
  end
y=cuenta;
end
```

If the repetitions of elements of $A = \{a, b, c, d\}$ are not allowed the linear dispositions of three positions of those elements have four possibilities for the first position, three possibilities for the second position, and two possibilities for the third position, giving as a total number of linear dispositions $4 \times 3 \times 2 = 24$. The listing of those dispositions can be seen in **Table 2**.

a,a,a	a,a,b	a,a,c	a,a,d	a,b,a	a,b,b	a,b,c	a,b,d
a,c,a	a,c,b	a,c,c	a,c,d	a,d,a	a,d,b	a,d,c	a,d,d
b,a,a	b,a,b	b,a,c	b,a,d	b,b,a	b,b,b	b,b,c	b,b,d
b,c,a	b,c,b	b,c,c	b,c,d	b,d,a	b,d,b	b,d,c	b,d,d
c,a,a	c,a,b	c,a,c	c,a,d	c,b,a	c,b,b	c,b,c	c,b,d
c,c,a	c,c,b	c,c,c	c,c,d	c,d,a	c,d,b	c,d,c	c,d,d
d,a,a	d,a,b	d,a,c	d,a,d	d,b,a	d,b,b	d,b,c	d,b,d
d,c,a	d,c,b	d,c,c	d,c,d	d,d,a	d,d,b	d,d,c	d,d,d

Table 1.
Complete listing of permutation with repetitions.

a,b,c	a,b,d	a,c,b	a,c,d	a,d,b	a,d,c
b,a,c	b,a,d	b,c,a	b,c,d	b,d,a	b,d,c
c,a,b	c,a,d	c,b,a	c,b,d	c,d,a	c,d,b
d,a,b	d,a,c	d,b,a	d,b,c	d,c,a	d,c,b

Table 2.
Complete listing of the permutations without repetitions.

This last table was generated by the following Matlab function.

```
function [y,lista] = ListaPSR(A,k)
% Autor: Carlos Rodriguez Lucatero
[m,n]=size(A);
r=factorial(n)/factorial(n-k);
y(1,1:k)=A(1,1:k);
s=1;
while r-1 > 0
     j=n-1;
     while((A(1,j) >= A(1,j+1)) & (j>0))
          j=j-1;
     end
     l=n;
     while ((A(1,j) >= A(1,l)) & (l>j))
          l=l-1;
     end
     temp=A(1,j);
     A(1,j)=A(1,l);
     A(1,l)=temp;
     t=j+1;
     l=n;
     while (t<l)
          temp=A(1,t);
          A(1,t)=A(1,l);
          A(1,l)=temp;
          t=t+1;
          l=l-1;
     end
     s=s+1;
     y(s,1:k)=A(1,1:k);
     r=r-1;
end
lista=y;
end
```

In general, if the cardinality of the set of objects is n and the number of possible positions within the linear arrangement is k then the total of possible permutations with repetition would be equal to n^k. In the event that repetitions of elements are not allowed in the arrangement, the first position can be occupied by any of the n elements of the set of objects. Once the element is assigned to position 1, the next position can be filled by any of the remaining $n - 1$ elements and so on until filling the k positions of the linear arrangement. In the general case, the calculation of the total of possible dispositions under the aforementioned conditions would be $n \times (n-1) \times \ldots \times (n-k+1)$. As you can see, this calculation can result in an excessive number of factors and for that reason the $n!$ symbol is introduced, which is equal to $n \times (n-1) \times (n-2) \ldots \times 3 \times 2 \times 1$. Thus, the calculation of the total number of permutations without repetition of n objects taken from k in k is calculated with the following formula

$$\frac{n!}{k!(n-k)!}. \tag{1}$$

The factorial can be defined as a recurrence relation. Every recurrence relationship has a stop condition and a recursive step. To do this, it is necessary to define both parts of the recurrence. For this, we agree that $0! = 1$ which constitutes the stop condition and since $n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 = n \times (n-1)!$. So, we can rewrite $n!$ as recurrence as follows

$$\begin{cases} 0! = 1 & \text{for } n = 0. \\ n! = n \times (n-1)! & \text{for } n > 0. \end{cases} \tag{2}$$

The recurrence 2 to calculate the factorial is one of the first recurrence relationships to appear in discrete mathematics. If we try to implement in Matlab a direct translation of the recurrence 2 this could be the following.

```
function y=fact(n)
if n<0
    disp('argument should be positive');
    y=-1;
elseif n==0
    y=1;
  else
    y=n*fact(n-1);
  end
end
```

Such an implementation is compact and elegant but has the disadvantage that it can easily overwhelm the system's recursive call stack, which is why it is worth looking for an equivalent iterative implementation. Fortunately, it is known that recursive routines whose last instruction is a recursive call can always be translated into an iterative version. Therefore, we propose the following iterative implementation.

```
function y= factI( n )
acum=1;
for i=1:n
    acum=acum*i;
end
y=acum;
end
```

Sometimes when calculations have to be made, it may be enough to use good approximations. The calculation of $n!$ can be done in an approximate and efficient way using the Stirling approximation formula whose proof can be consulted in ref. [2] and whose mathematical expression is the following

$$n! \approx \sqrt{2 \cdot \pi} \cdot n^{n+\frac{1}{2}} e^{-n}. \tag{3}$$

This approximation turns out to be quite good and this allows us to calculate $n!$ very efficiently with a computer, avoiding the problem of excessive recursive calls in the case that n is very large. A possible implementation of this approach using Matlab would be the following.

```
function y = factStirling(n)
% Stirling approximation of the factorial
% Feller Vol. I pag 70
$ Autor: Carlos Rodriguez Lucatero
y=sqrt(2*pi)*n^(n+1/2)*exp(-1*n);
end
```

We can compare the algorithmic complexities of these three different functions to calculate $n!$ and we can notice that both the recursive and iterative versions have a time complexity of $O(n)$ while the Stirling approximation version is of $O(1)$ which is more efficient but at the price of the calculation being approximate. This example of the different ways of calculating $n!$ is the first example of how we can benefit from mathematical results to improve the temporal performance of the programs that we implement for this purpose.

4. Recurrences calculation and analysis of algorithms

With regard to algorithm efficiency, it is in this topic that some recurrence relationships and methods of solving said recurrences can be found to determine the time behavior of the algorithms in terms of the number of input data. One of the design techniques that allows you to build efficient algorithms is known as *Divide and Conquer*. A well-known problem in computing is that of ordering in ascending order a set of numerical data in disorder. In regular algorithm analysis courses, various algorithmic solutions for the said problem are studied and some of the most efficient algorithmic solutions are designed using the *Divide and Conquer* approach. One of the most famous Divides and Conquer algorithms is the Merge-Sort. The operation of the Merge-Sort starts by dividing the arrangement into subarrays in half and each subarray divides it again, in the same way, proceeding recursively until it reaches a point where it is not possible to further subdivide subarrays, after which it begins to sort by interleaving the subarrays until merging the ordered subarrays of the first partition. A possible Matlab implementation of this algorithm is shown below.

```
function y = mergesort(A,p,r)
if (p < r)
    q=floor((p+r)/2);
    y1=mergesort(A,p,q);
    y2=mergesort(A,q+1,r);
    y=merge(A,y1,y2,p,q,r);
else
    y=A;
end
end
function y = merge(A,y1,y2,p,q,r)
L=y1(1,p:q);
[m1,n1]=size(L);
L(1,n1+1)=99999999;
R=y2(1,q+1:r);
```

```
[m2,n2]=size(R);
R(1,n2+1)=99999999;
i=1;
j=1;
for k=p:r
    if L(i) <= R(j)
       A(k)=L(i);
       i=i+1;
    else
       A(k)=R(j);
       j=j+1;
    end
end
y=A;
end
```

When analyzing the Merge-Sort, a recurrence of the time or number of steps in total that it takes depending on the size of the input is obtained, which we will denote as $T(n)$. The recurrence obtained as a product of the analysis of this algorithm can have the following form

$$\begin{cases} T(1) = 1 & \text{for } n = 1. \\ T(n) = 2T\left(\frac{n}{2}\right) + n & \text{for } n > 1. \end{cases} \tag{4}$$

To solve the recurrence relation, four several methods, such as substitution, recurrence tree, or iteration of recurrence can be applied [3]. Here we will use the iteration method of recurrence. To simplify the problem, we will assume that the size of the array is $n = 2^m$, that is, it is a power of 2. Taking into account the above, we can operate a variable change in recurrence 4 which is rewritten as follows

$$\begin{cases} T\left(2^0\right) = 1 & \text{for } m = 0. \\ T(2^m) = 2T\left(2^{m-1}\right) + 2^m & \text{for } m > 0. \end{cases} \tag{5}$$

We iterate over the recurrence 5 we obtain the following expression

$$T(2^m) = 2^m + 2^m + 2^m + \ldots + 2^m = m \times 2^m. \tag{6}$$

We know that $n = 2^m$ and therefore that $m = \log_2(n)$. Then we can make the change of variable in expression 5 and obtain the solution to recurrence 4 that would have the following form

$$T(n) = n \log_2(n). \tag{7}$$

Recurrence relationships are present in undergraduate courses in algorithmic analysis, mathematical thinking, or discrete mathematics. One of the topics that are usually addressed in the courses of mathematical thought is that of sequences of numbers and the detection of patterns of behavior of these sequences to illustrate the mathematical process of discovering the properties of sequences of numbers. Typical examples consist of presenting a sequence of integer values and inferring what is the next number in the sequence. In classic mathematical thinking, textbooks such as [4] systematic methods are exposed to solve this type of mathematical puzzles, such as the method of successive differences. Normally these types of exercises remain at the point of discovering the next element in the sequence, but you can go further in the problem and try to discover the mathematical expression that allows obtaining the term of a sequence of numbers in an arbitrary position. It is there where we can resort to the recurrence relations from which a recursive program can be implemented to obtain elements of the numerical sequence in any given position or even more use the tools of discrete mathematics to solve said recurrences to obtain in computationally efficient way elements in arbitrary positions in a sequence of numbers. Next, we will show these ideas with an example with numerical sequences in which we will obtain the following element of the sequence by the method of successive differences, then we will obtain some recurrence relation from which we will implement a function in Matlab that allows us to obtain any element of the sequence given the position and finally, we will solve the recurrence in question to obtain a mathematical formula that represents the form of the n-th term of the sequence from which a function will be implemented to Matlab to evaluate said mathematical expression giving the position to obtain the number that occupies that position within the given sequence of numbers.

4.1 Numerical sequences and recurrences

There are sequences of numbers known as figurative numbers since they are related to figures of a certain type that are formed by joining a certain number of points with lines. These points and lines can form, for example, triangles, squares, pentagons, or heptagons, and the associated sequences will be called triangular numbers, square numbers, pentagonal numbers, or heptagonal numbers, respectively. In **Figure 1**, we can see how pentagons can be formed from the number of points given in each case.

The sequence associated with the pentagonal numbers is shown in **Table 3**.

Suppose you want to obtain the next element of the numerical sequence, that is, the element a_5. For this, we can apply the method of successive differences that consists of taking the first difference between successive elements of the sequence, then the differences of the first successive differences are taken, and so on.

When the successive differences become constant, the process stops and we perform a backward calculation adding the last differences of each level until we reach the calculation of the last element of the original sequence.

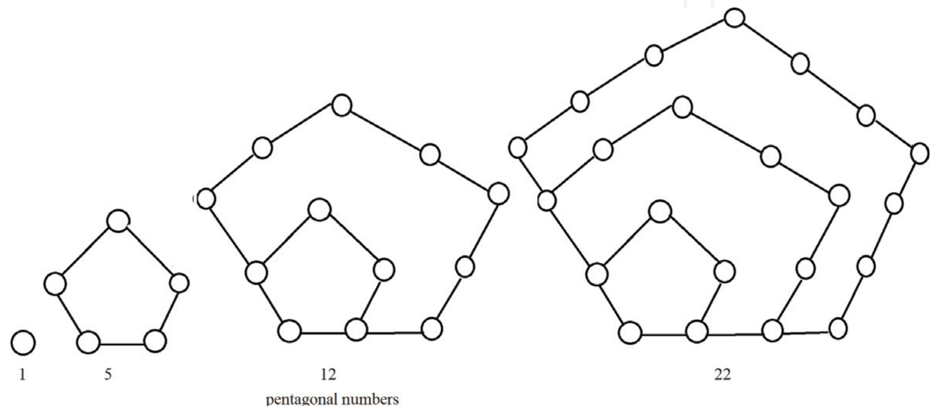

Figure 1.
Pentagonal numbers sequence.

a_0	a_1	a_2	a_3	a_4	a_5
1	5	12	22	35	

Table 3.
Pentagonal numeric sequence.

a_0	a_1	a_2	a_3	a_4	a_5
	5	12	22	35	**51**
4	7	10	13	**16**	
	3	3	3	**3**	

Table 4.
Successive differences on the pentagonal sequence.

The steps performed by this procedure appear in **Table 4**.

The numbers that appear in boldface are generated by the backward calculation mentioned before. This procedure, although correct, can become very cumbersome if instead of wanting to calculate the next element in the sequence, you want to know the value of the element in the sequence in position 30. It is in this case that it would be worthwhile to obtain a mathematical relationship that would allow us to implement a program in any programming language that is available to obtain any element of the numerical sequence in any given position. One way to achieve this goal is to try to discover the recurrence relationship that allows the elements of the sequence to be generated until the element of the sequence in the desired position is reached. This is what we will do next. From the calculations carried out in the successive differences procedure, we can obtain the following first difference relationships between elements of the sequence

$$a_1 - a_0 = 4. \tag{8}$$

$$a_2 - a_1 = 7. \tag{9}$$

$$a_3 - a_2 = 10. \tag{10}$$

$$a_4 - a_3 = 13. \tag{11}$$

$$a_5 - a_4 = 16. \tag{12}$$

From the first differences, we can obtain the following relationships corresponding to the second differences

$$(a_2 - a_1) - (a_1 - a_0) = 3. \tag{13}$$

$$(a_3 - a_2) - (a_2 - a_1) = 3. \tag{14}$$

$$(a_4 - a_3) - (a_3 - a_2) = 3. \tag{15}$$

$$(a_5 - a_4) - (a_4 - a_3) = 3. \tag{16}$$

We can observe the relations related to the second difference; a regular behavior that allows us to establish the following generalization

$$(a_n - a_{n-1}) - (a_{n-1} - a_{n-2}) = 3. \tag{17}$$

From Eq. (17), we can obtain the following difference equation

$$a_n - 2a_{n-1} + a_{n-2} = 3. \tag{18}$$

As can be seen, it is a difference equation, it is non-homogeneous linear second order and with constant coefficients, and for this reason, it requires two initial conditions that are taken from the numerical sequence in question. These initial conditions are $a_0 = 1$ and $a_1 = 5$. Thus, the equation in complete differences would be expressed in the following form

$$a_n - 2a_{n-1} + a_{n-2} = 3, a_0 = 1, a_1 = 5. \tag{19}$$

From Eq. (19), we can obtain the following recurrence

$$\begin{cases} a_0 = 1 & \text{for } n = 0. \\ a_1 = 5 & \text{for } n = 1. \\ a_n = 2a_{n-1} - a_{n-2} + 3 & \text{for } n > 1. \end{cases} \tag{20}$$

Recurrence allows us to directly implement the following recursive function in Matlab.

```
function y = recpentagR(n)
%Author: Carlos Rodriguez Lucatero
if (n==0)
    y=1;
else
    if (n==1)
       y=5;
    else
       y=2*recpentagR(n-1)-recpentagR(n-2)+3;
    end
end
end
```

We can test the routine from Matlab by first calculating an element of the sequence whose value we know, for example $a_5 = 51$. The result of calling the function from the Matlab prompt is the following.

```
>> z=recpentagR(5)
z =
51
```

Now let us try to calculate with this same routine the element of the numerical sequence at position 30, that is, a_{30}. The result is shown below.

```
>> z=recpentagR(30)
z =
    1426
```

If we wanted to calculate a_{50} with this recursive routine we would realize that the calculation time increases a lot because many of the calculations are recalculated and also there is a risk of overflowing the system stack due to a large amount of recursive calls. Fortunately, we can reprogram an iterative version of this function and it is the one shown below.

```
function y = recpentagI(n)
%Author: CRL
a0=1;
a1=5;
if (n==0)
    y=a0;
else
    if (n==1)
        y=a1;
    else
        an=0;
        for i=2:n
            an=2*a1-a0+3;
            a0=a1;
            a1=an;
        end
        y=an;
    end
end
end
```

We will test this routine by first executing it for the known value $a_5 = 51$, then for the value a_{30} whose value obtained with the recursive version was 1426 and finally, we will obtain the value a_{50} as well as the a_{100} value.

```
>> z=recpentagI(5)
z =
     51
>> z=recpentagI(30)
z =
     1426
>> z=recpentagI(50)
z =
     3876
>> z=recpentagI(100)
z =
     15251
```

The runtime improvement of the iterative version over the recursive version is truly impressive. However, the execution time of a routine can be further improved to carry out this calculation by resorting to discrete mathematics tools to solve the recurrence and implement a routine that the only thing that does is evaluate in the given position the mathematical formula obtained from the solution of the recurrence. Discrete mathematics provides us with many methods to solve non-homogeneous linear difference equations, such as the one we will solve, associated with recurrence relationships. This type of difference equations can be obtained by methods, such as the iteration of recurrence, the characteristic polynomial method, or the generating function method. In this subsection, we will resolve the recurrence by applying several of these methods for illustrative purposes. For more details on solving recurrences using these methods, we recommend consulting [3, 5–8].

4.2 Solving a recurrence by iteration

We start with the application of the iteration method of recurrence. For this purpose, we will use the third row of the recurrence relation (20). This equation would be the following

$$a_n = 2a_{n-1} - a_{n-2} + 3. \tag{21}$$

We apply Eq. (21) to the case of $n - 1$ which would give the following equation

$$a_{n-1} = 2a_{n-2} - a_{n-3} + 3. \tag{22}$$

We substitute 22 in 21 and obtain the following equation

$$a_n = 3a_{n-2} - 2a_{n-3} + 2 \cdot 3 + 3. \tag{23}$$

We apply Eq. (21) to the case of $n - 2$ which would give the following equation

$$a_{n-2} = 2a_{n-3} - a_{n-4} + 3. \tag{24}$$

We substitute 24 in 23 and obtain the following equation

$$a_n = 4a_{n-3} - 3a_{n-4} + 3 \cdot 3 + 2 \cdot 3 + 3. \tag{25}$$

We apply Eq. (21) to the case of $n - 3$ which would give the following equation

$$a_{n-3} = 2a_{n-4} - a_{n-5} + 3. \tag{26}$$

We substitute 26 in 25 and obtain the following equation

$$a_n = 5a_{n-4} - 4a_{n-5} + 4 \cdot 3 + 3 \cdot 3 + 2 \cdot 3 + 3. \tag{27}$$

Continuing with this procedure until reaching the base cases and noting that certain regularities appear, such as the presence of a sum of successive natural numbers, we arrive at the following expression

$$a_n = n \cdot a_1 - (n-1) \cdot a_0 + 3 \cdot \sum_{i=1}^{n-1} i. \tag{28}$$

Substituting $a_0 = 1, a_1 = 5$ and applying Gauss's formula to the summation in Eq. (28), we obtain the following solution

$$a_n = \frac{3}{2} \cdot n^2 + \frac{5}{2} \cdot n + 1. \tag{29}$$

4.3 Solving a recurrence by the characteristic polynomial method

We can try another method of solving the recurrence to illustrate another tool provided by discrete mathematics. In this case, we will use the characteristic polynomial method. The difference equation that we are going to solve is linear inhomogeneous with coefficients, which makes it capable of being solved by this method.

These methods are closely related to the methods of solving differential equations of the same type. The difference equation that we are going to solve is the following

$$a_n - 2a_{n-1} + a_{n-2} = 3, a_0 = 1, a_1 = 5. \tag{30}$$

The theory on the solution of equations in non-homogeneous linear differences says that the general solution is composed of a solution of the homogeneous equation plus a particular solution related to the non-homogeneous part. The homogeneous equation associated with Eq. (30) would be the following

$$a_n - 2a_{n-1} + a_{n-2} = 0. \tag{31}$$

A characteristic polynomial is associated with the homogeneous Eq. (31) that is obtained by substituting a possible solution in the difference equation. Said possible solution has the form $a_n = c \cdot r^n$ where c is an arbitrary constant if we substitute said possible solution in 31 we obtain the following polynomial

$$r^2 - 2r + 1 = 0. \tag{32}$$

The roots of the characteristic polynomial 32 are $r_1 = 1$ and $r_2 = 1$, that is, they are real and repeated roots and since the solutions of a difference equation must be linearly independent, the form of the solution of the homogeneous difference equation would be the following

$$a_n = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n. \tag{33}$$

Now we proceed to solve the particular equation whose solution must be linearly independent of the solutions of the associated homogeneous equation. We note that the right-hand side of the difference Eq. (30) is $f(n) = 3$ which is equivalent to $f(n) = 3 \cdot 1^n$. So the form of the particular solution would be the following

$$a_n = A \cdot n^2. \tag{34}$$

If we evaluate the equation in differences 30 the solution 34 we obtain the following expression

$$A \cdot n^2 - 2 \cdot A \cdot (n-1)^2 + A \cdot (n-2)^2 = 3 \cdot 1^n. \tag{35}$$

After algebraically simplifying Eq. 35 we deduce that $A = \frac{3}{2}$, so the general solution would have the form

$$a_n = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n + \frac{3}{2} \cdot n^2. \tag{36}$$

Applying on 37 $a_0 = 1$ we deduce that $c_1 = 1$. Applying on 37 $a_1 = 5$ we get $c_2 = \frac{5}{2}$ In this way we obtain that the general solution is

$$a_n = 1 + \frac{5}{2} \cdot n + \frac{3}{2} \cdot n^2. \tag{37}$$

We can verify that solution 37 coincides with solution 29. Finally, we can use this solution to implement a Matlab function to perform this calculation and obtain the value of an element of the sequence given the position. The Matlab function would be the following.

```
function y = recpentag(n)
% Author: Carlos Rodriguez Lucatero
y=1+(5/2)*n+(3/2)*(n^2);
end
```

For being convinced about the correctness of the solution, we can evaluate this Matlab function for the same values used before and we obtain the following results.

```
>> y = recpentag(5)
y =
     51
>> y = recpentag(10)
y =
     176
>> y = recpentag(30)
y =
     1426
>> y = recpentag(50)
y =
     3876
>> y = recpentag(100)
y =
     15251
```

4.4 Solving a recurrence by generating function method

Another powerful method of discrete mathematics to solve difference Eq. (19) associated with the recurrence 20 is that of ordinary generating functions. Said method consists, in the simplest case, in converting a numerical sequence into an infinite polynomial of a single variable whose coefficients are precisely the elements of the numerical sequence. The reason for doing this transformation is that the algebraic manipulation of these infinite polynomials is relatively simple. For this reason, we define below the concept of the ordinary generating function of a single variable.

Definition 1.1 Given a numerical sequence $a_0, a_1, a_2, \dots, a_k, \dots$ the function

$$A(z) = \sum_{k \geq 0} a_k z^k. \tag{38}$$

It is called the ordinary generating function (OGF) of the sequence. The notation $[z^k]A(z)$ will be used to refer to the coefficient a_k of the k-th term of the infinite polynomial.

By means of the generating functions, we can map sequences of numbers that often are integers to power series. The coefficient of the n-th adding will, therefore, be related to the n-th element of the associated numerical sequence. For example, in the generating function $\sum_{i=0}^{\infty} z^i = 1 + z + z^2 + z^3 + z^4 + \dots$, we observe that it is the geometric series that converges if $|z| < 1$ and can be expressed as $\frac{1}{1-z}$. We can also observe that all the terms of the sum have a coefficient of 1, so this generating function is associated with the numerical sequence $1, 1, 1, 1, \dots$. On the other hand, since the generating functions are infinite polynomials, it is easy to apply the derivative operation to them. Thus, the derivative of the geometric series would be expressible as follows

$$\frac{d}{dz}\left(\frac{1}{(1-z)}\right) = \frac{1}{(1-z)^2} = \frac{d}{dz}\left(1 + z + z^2 + z^3 + z^4 + \dots\right) = 1 + 2z + 3z^2 + 4z^3 + \dots \tag{39}$$

As can be seen from 39, the derivative of the geometric series can be related to de succession of natural numbers.

It is also possible to do certain types of algebraic operations on ordinary generating functions that have an impact on the sequence of numerical values associated with the given generating function. For example, if I multiply the generating function of $\frac{1}{(1-z)^2}$ by z, we obtain the following effect in the numerical sequence

$$\frac{z}{(1-z)^2} = 0 + z + 2z^2 + 3z^3 + 4z^4 + \dots \leftrightarrow 0, 1, 2, 3, 4, 5, \dots \tag{40}$$

that is to say that we obtain a shift to the right in the sequence of numbers.

If we apply the derivative to Eq. (40), we obtain the following relationship

$$\begin{aligned}\frac{d}{dz}\left(\frac{z}{(1-z)^2}\right) &= \frac{d}{dz}\left(0 + z + 2z^2 + 3z^3 + 4z^4 + \dots\right) = \frac{z+1}{(1-z)^3} \\ &= 1 + 2^2 z + 3^2 z^2 + 4^2 z^3 + 5^2 z^4 + \dots\end{aligned} \tag{41}$$

then we have

$$\frac{z+1}{(1-z)^3} = 1 + 2^2 z + 3^2 z^2 + 4^2 z^3 + 5^2 z^4 + \dots \leftrightarrow 1^2, 2^2, 3^2, 4^2, \dots \tag{42}$$

We have exemplified the following relationships between numerical sequences and generating functions

$$1, 1, 1, 1, \dots \leftrightarrow \frac{1}{1-z} \tag{43}$$

$$1, 2, 3, 4, \dots \leftrightarrow \frac{1}{(1-z)^2}. \tag{44}$$

$$0, 1, 2, 3, 4, \dots \leftrightarrow \frac{z}{(1-z)^2} \tag{45}$$

$$1^2, 2^2, 3^2, 4^2, \dots \leftrightarrow \frac{z+1}{(1-z)^3} \tag{46}$$

$$0^2, 1^2, 2^2, 3^2, 4^2, \dots \leftrightarrow \frac{z(z+1)}{(1-z)^3} \tag{47}$$

After all those examples, we can be convinced that it is possible to establish relationships between sequences of numbers and generating functions, as shown in **Table 5**.

$1, 1, 1, 1, 1, \dots$	$\frac{1}{1-z} = \sum_{n\geq 0} z^n$
$0, 1, 2, 3, 4, \dots, n, \dots$	$\frac{z}{(1-z)^2} = \sum_{n\geq 1} nz^n$
$0, 0, 1, 3, 6, , 10, \dots, \binom{n}{2}, \dots$	$\frac{z^2}{(1-z)^3} = \sum_{n\geq 2} \binom{n}{2} z^n$
$0, 0, \dots, 0, 1, m+1, , 10, \dots, \binom{n}{m}, \dots$	$\frac{z^m}{(1-z)^{m+}} = \sum_{n\geq m} \binom{n}{m} z^n$
$1, m, \binom{m}{2}, \dots, \binom{m}{n}, \dots, m, 1$	$(1+z)^m = \sum_{n\geq 0} \binom{m}{n} z^n$
$1, m+1, \binom{m+2}{2}, \dots, \binom{m+3}{3}, \dots$	$\frac{1}{(1-z)^{m+1}} = \sum_{n\geq 0} \binom{n+m}{n} z^n$
$1, 0, 1, 0, \dots, 1, 0, \dots$	$\frac{1}{(1-z)^2} = \sum_{n\geq 0} z^{2n}$
$1, c, c^2, c^3, c^4, \dots, c^n, \dots$	$\frac{1}{1-cz} = \sum_{n\geq} c^n z^n$
$1, 1, \frac{1}{2!}, \frac{1}{3!}, \frac{1}{4!}, \dots, \frac{1}{n!}, \dots$	$e^x = \sum_{n\geq 0} \frac{z^n}{n!}$
$0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots, \frac{1}{n}, \dots$	$\ln\left(\frac{1}{1-z}\right) = \sum_{n\geq 1} \frac{z^n}{n}$
$0, 1, 1+\frac{1}{2}, 1+\frac{1}{2}+\frac{1}{3}, \dots, H_n, \dots$	$\frac{1}{1-z} \ln\left(\frac{1}{1-z}\right) = \sum_{n\geq 1} H_n z^n$ (where H_n is the harmonic series)
$0, 0, 1, 3\left(\frac{1}{2}+\frac{1}{3}\right), 4\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}\right) \dots$	$\frac{z}{(1-z)^2} \ln\left(\frac{1}{1-z}\right) = \sum_{n\geq 0} n(H_n - 1) z^n$

Table 5.
Table of sequences and ordinary generating functions.

For more detailed information on recurrences that appear in the analysis of algorithms as well as generating functions, we recommend the excellent book [5].

After this brief summary on ordinary generating functions, we can solve the difference Eq. (19) using generating functions and that is just what we will do next. The difference equation to solve is

$$a_n - 2a_{n-1} + a_{n-2} = 3, a_0 = 1, a_1 = 5. \tag{48}$$

We know from definition 38 that a generating function is expressed mathematically as

$$A(z) = \sum_{k\geq 0} a_k z^k. \tag{49}$$

and we apply this operator to the difference Eq. (48) keeping the same summation index for the terms of the difference Eq. (48) obtaining the following expression

$$\sum_{n\geq 2} a_n z^n - 2\sum_{n\geq 2} a_{n-1} z^n + \sum_{n\geq 2} a_{n-2} z^n = 3\sum_{n\geq 2} z^n. \tag{50}$$

Because of the index shift, the first summation is $A(z) - a_0 - a_1 z$. In the second term of the left hand of Eq. (50) the index of the coefficient and the power of z being different; so, we can factorize z and take into account the shift in the summation index and obtain the term $2z(A(z) - a_0$. The difference between the summation index and the power of z in the third term of the left hand of 50 can be arranged by factoring z^2 and in that case, we obtain the term $z^2 A(z)$. The right hand of Eq. (50) is a geometric series but given the shift on the index of the summation, it is necessary to subtract the two first terms of the geometric series giving, as a result, the term $3\left(\frac{1}{1-z} - 1 - z\right)$. Then taking into account these remarks, we obtain the next equation

$$A(z) - 1 - 5z - 2zA(z) - 2z + z^2A(z) = 3\left(\frac{1}{1-z} - 1 - z\right). \tag{51}$$

Simplifying Eq. (51) we get the next equation

$$A(z)\left(1 - 2z + z^2\right) - 1 - 3z = 3\left(\frac{z^2}{1-z}\right). \tag{52}$$

The left hand of 53 can be algebraically simplified obtaining the following expression

$$A(z)(1-z)^2 = 1 + 3z + 3\left(\frac{z^2}{1-z}\right). \tag{53}$$

Simplifying de left hand of Eq. (53) we get the equation

$$A(z) = \frac{1}{(1-z)^2} + 3\frac{z}{(1-z)^2} + 3\left(\frac{z^2}{(1-z)^3}\right). \tag{54}$$

We apply the $[z^n]$ operator to Eq. (54) with the purpose of obtaining the coefficient of the nth addend of the sum that will correspond to the element in the position n of the sequence of numbers studied, arriving at the equation

$$a_n = [z^n]A(z) = [z^n]\left(\frac{1}{(1-z)^2}\right) + 3[z^n]\left(\frac{z}{(1-z)^2}\right) + 3[z^n]\left(\frac{z^2}{(1-z)^3}\right). \tag{55}$$

and then to the final result

$$a_n = (n+1) + 3n + 3\binom{n}{2} = 4n + 1 + \frac{3n^2}{2} - \frac{3n}{2} = 1 + \frac{5n}{2} + \frac{3n^2}{2}. \tag{56}$$

As you can see the expressions (29), (37), and (56) are the same.

5. Modular arithmetical calculations

In discrete mathematics courses at the undergraduate level, topics of modern algebra are addressed where the most elementary algebraic structures are defined, such as the group structure and the ring structure. The algebraic structure to which we are going to devote our attention in this section is the ring structure in the context of modular arithmetic.

This algebraic structure is often used, informally, in arithmetic courses to learn to add and multiply. Formally speaking we can say that this structure is composed of a set of objects that could be in the case of arithmetic the set of integers as well as two operations which are addition and multiplication which is denoted as $(\mathbb{Z}, +, \cdot)$. The two operations must be closed, that is to say, that the results they give must belong to the same set from which the operands are taken. Additionally, the operations must respect certain properties that we will define below.

Definition 1.2 (Ring) Let be R nonempty set that has two closed binary operations denoted as $+$ y $\cdot$. Then $(R, +, \cdot)$ it is a ring if $\forall a, b, c \in R$ the following conditions are met:

a) $a + b = b + a$ (commutative law of $+$).
b) $a + (b + c) = (a + b) + c$ (Associative law of $+$).
c) $\exists z \in R$ such that $a + z = z + a = a, \ \forall a \in R$ (existence of identity element for $+$).
d) For each $a \in R, \ \exists \ b \in R$ such that $a + b = b + a = z$ (existence of inverse under $+$).
e) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (Associative law for $\cdot$).
f) $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(b + c) \cdot a = b \cdot a + c \cdot a$ (Distributive laws for $\cdot$ over $+$).

In order to simplify the notation, the operation $a \cdot b$ can be written as ab. The associative properties of both operations as well as the distributivity of one operation over the other can be generalized for the case of more than 2 operands.

The $+$ operation of the ring is commutative but the $\cdot$ operation is not commutative in the 1.2 definitions. We can apply the commutativity property to the $\cdot$ operation and we would obtain the commutative ring structure. Formally the definition of a commutative ring is as follows.

Definition 1.3 (Commutative ring) Let be R a nonempty set having two closed binary operations denoted as $+$ and $\cdot$. Then $(R, +, \cdot)$ is a ring if $\forall a, b, c \in R$ meet the following conditions:

a) If $ab = ba, \forall a, b \in R$ then R is a commutative ring.
b) It is said that the ring R has no proper divisors of zero if for any $a, b \in R, ab = z \Rightarrow a = z \vee b = z$.
c) If an element $u \in R$ such that $u \neq z$ and $ua = au = a, \forall a \in R$ we say that u is a unit or identity element of multiplication and that R is a ring with unity.

Another interesting property that can be added to the list of ring properties is the existence of multiplicative inverse. The definition is the following.

Definition 1.4 (Ring with multiplicative inverse) Let R be a ring with the unit u if $a, b \in R$ and $ab = ba = u$ then b is called inverse multiplicative of a and a is called a unit of R. (An element b is also a unit of R).

Taking into account the existence of the unit property, we can add it to the ring commutative ring and obtain a commutative ring with unity.

Definition 1.5 (Commutative ring with unity) Let be R a commutative ring with unity. Then:

a) R is said to have an integer domain if R has no proper divisors of zero.
b) R is said to be a field if each non-zero element of R is a unit.

Rings can have subsets that satisfy the properties of a ring as defined in 1.2 in which case we are talking about subring structures. These algebraic structures can have additive cancelation properties as well as multiplicative cancelation properties and therefore will have symmetric or inverse elements of addition as well as multiplication. Once we have understood the theoretical part of the rings we are better positioned to tackle the construction and use of finite rings specials and fields. We will start by presenting some results of the modular arithmetic needed for the de notion of operations in modular rings. These notions will be explained below.

5.1 Integers mod n

In this section, we will review some notions of modular arithmetic that usually appear in introductory courses of the Theory of Numbers and that allow us to understand important concepts, such as divisibility, primality, the greatest common divisor operation as well as the algorithm of Euclid to calculate it efficiently. Let us start with establishing the meaning of some mathematical symbols that appear in this context. The expression $d|a$ translates into words like d divides a to what is the same as $\exists k \in \mathbb{Z}$ such that $a = kd$ and d is called divisor of a. This means that if we divide a by d, the remainder will be 0. From here we can make the following properties and definitions.

Definition 1.6 Every integer divides 0. That is $\forall x \in \mathbb{Z}, x|0$.

Proposition 1.7 If $a > 0$ and $d|a$ then $|d| \leq |a|$.

Proposition 1.8 $d|a$ if and only if $-d|a$.

Definition 1.9 A number x is said to be composite if it has divisors other than 1 and x.

Example 1.10 The number 39 is composite since 3|39 is true and $3 \neq 1$ is true as like $3 \neq 39$.

Example 1.11 The divisors of 24 are $\{1, 2, 3, 4, 6, 8, 12, 24\}$. The 1 and 24 are known as trivial divisors.

Definition 1.12 The set of divisors of a number a that are neither 1 nor a are called factors of a.

Definition 1.13 A number prime is here the one that has no factors.

Example 1.14 The numbers $\{1, 3, 5, 7, 11, 13\}$ are prime.

Definition 1.15 A number that is not prime is said to be a composite number.

Theorem 1.16 (Division Theorem) For any $a, n \in \mathbb{Z}$ there exist $q, r \in \mathbb{Z}$ único such that $a = qn + r$ where $0 \leq r < n$.

$q = \lfloor \frac{a}{n} \rfloor$ is called the quotient and $r = a \bmod n$ is called the remainder. Then $n|a$ if $a \bmod n$ is equal to 0. From this, in conjunction with theorem 16, it can be stated that $a = \lfloor \frac{a}{n} \rfloor n + a \bmod n$ or equivalently that $a - \lfloor \frac{a}{n} \rfloor n = a \bmod n$.

Definition 1.17 If $(a \bmod n) = (b \bmod n)$ then we say that $a \equiv b \bmod n$ if they have the same remainder a and b when divided by n is to say that $a \equiv b \bmod n \Leftrightarrow n|(b - a)$.

When it is true that $(a \bmod n) = (b \bmod n)$ we will denote it as $a \equiv b \bmod n$ and the equivalence classes that this relation generates will be expressed as $[a]_n = \{a + kn | k \in \mathbb{Z}\}$. In general, we can say the congruence relation modulo some number n partitions to the set of integers in equivalence classes by the remainder left when divided by a number n which we express as $[\mathbb{Z}]_n = \{[a]_n : 0 \leq a \leq n - 1\}$.

5.2 Common divisors and greatest common divisor

Let us start by defining the concept of a common divisor.

Definition 1.18 d is a common divisor of the numbers a and b if $d|a$ and $d|b$ hold.

Common divisors have the following properties.

Proposition 1.19 $d|(a + b)$ and $d|(a - b)$.

Proposition [1.20] If $d|a$ and $d|b$ then $d|(ax + by)$ $\forall x, y \in \mathbb{Z}$. That is, d divides every linear combination of a and b.

Proposition 1.21 If $a|b$ then $|a| \leq |b|$ or $b = 0$.

Proposition 1.22 If $a|b$ and $b|a$ hold then $a = +b$ or $a = -b$.

Definition 1.23 The greatest common divisor of a and b denoted as $gcd(a, b)$. (greatest common divisor) is: $\gcd(a, b) = \max\{d : d|a\ y\ d|b\}$.

Example 1.24 $gcd(24, 30) = 6, gcd(5, 7) = 1, gcd(0, 9) = 0$.

Proposition 1.25 If a and b are not both equal to 0, then $gcd(a, b)$ is an integer between 1 and $\min(|a|, |b|)$.

Definition 1.26 $gcd(0, 0) = 0$.

Definition 1.27 a and b are relatively prime if $gcd(a, b) = 1$.

Proposition 1.28 $gcd(a, b) = gcd(b, a)$.

Proposition 1.29 $gcd(a, b) = gcd(-a, b)$.

Proposition 1.30 $gcd(a, b) = gcd(|a|, |b|)$.

Proposition 1.31 $gcd(a, 0) = |a|$.

Proposition 1.32 $gcd(a, ka) = a, \forall k \in \mathbb{Z}$.

Theorem 1.33 If a and b are any integers and are not both equal to 0 then $gcd(a, b)$ is the smallest positive element of the set $\{ax + by : x, y \in \mathbb{Z}\}$ of linear combinations of a and b.

Proof: Let s be the smallest positive element of the set of such linear combinations of a and b, then $s = ax + by$ for some $x, y \in \mathbb{Z}$. Let $q = \lfloor \frac{a}{s} \rfloor$. The equation $a \mathrm{mod} n = a - \lfloor \frac{a}{n} \rfloor$ implies $a \mathrm{mod} s = a - qs = aq(ax + by) = a(1 - qx) + b(-qy)$, so $s \mathrm{mod} s$ is a linear combination of a and b. Since $a \mathrm{mod} s < s$ then $a \mathrm{mod} s = 0$ since s is the smallest value that is a linear combination of a and b. The above means that $s|a$. Reasoning in a similar way we can prove that $s|b$. Then s is a common divisor of a and b, and it also holds that $gcd(a, b) \geq s$ and like $d|a$ and $d|b$ implies that $d|(ax + by)$, so as a consequence of all this $gcd(a, b)|s$ since $gcd(a, b)$ divides any linear combination of a with b and s is a linear combination of a with b. But $gcd(a, b)|s$ and $s > 0$ imply that $gcd(a, b) < s$. Combining that is satisfied simultaneously that $gcd(a, b) > s$ and that $gcd(a, b) < s$ we can conclude that $gcd(a, b) = s$.

Corollary 1 For any $a, b \in \mathbb{Z}$, if $d|a$ and $d|b$ then $d|gcd(a, b)$.

Proof: If $d|a$ and $d|b$ then $\forall x, y \in \mathbb{Z} d|(ax + by)$ and since $gcd(a, b)$ is a linear combination of a and b we can conclude. □.

Corollary 2 $\forall a, b, n \in \mathbb{Z}$ and $n \geq 0$ $gcd(an, bn) = ngcd(a, b)$.

Corollary 3 $\forall a, b, n \in \mathbb{Z}$ and $a, b, n \geq 0$ if $n|ab$ and $gcd(a, n) = 1$ then $n|b$.

Theorem 1.34 $\forall a, b \in \mathbb{Z}^+, gcd(a, b) = gcd(b, a \mathrm{mod} b)$.

Proof:

- (demo sketch).
- We first prove that $gcd(a, b)|gcd(b, a \mathrm{mod} b)$.
- Let $d = gcd(a, b)$ then $d|a$ and $d|b$.
- We know that $(a \mathrm{mod} b) = a - qb$ where $q = \lfloor \frac{a}{b} \rfloor$.
- Since $(a \mathrm{mod} b)$ is a linear combination of a and b.
- from the above and the fact that $d|a$ and $d|b$ this implies that $d|(ax + by)$ then $d|b$ and $d|(a \mathrm{mod} b)$ where $gcd(a, b)|gcd(b, a \mathrm{mod} b)$.
- As a second part, it is proved in a similar way that $gcd(b, a \mathrm{mod} b)|gcd(a, b)$.
- If both $gcd(a, b)|gcd(b, a \mathrm{mod} b)$ and $gcd(b, a \mathrm{mod} b)|gcd(a, b)$ hold we can conclude that $\forall a, b \in \mathbb{Z}^+, gcd(a, b) = gcd(b, a \mathrm{mod} b)$.

Theorem 1.34 provides us with the mathematical elements to be able to define a recursive algorithm for the calculation of the Greatest Common Divisor is the following.

```
Euclid(a,b)
1) if b=0
2) then return a
3) else Euclid(b, a mod b)
```

The following is an example of how Euclid's algorithm works.

Example [1.35] *We will calculate the* $gcd(30, 21)$ *applying the Euclidean algorithm that I have just explained. The execution steps are displayed in* **Table 6**.

5.3 Modular rings

Definition 1.36 Let $n \in \mathbb{Z}^+, n > 1$. For $a, b \in \mathbb{Z}$, tenths that a is congruent to b modulo n and is written as $a \equiv b(\text{mod} n)$, if $n|(ab)$, or equivalently $a = b + kn$ for some $k \ \ in\mathbb{Z}$.

Example 1.37 For example:

a) $17 \equiv 2(\text{mod}5)$.
b) $-7 \equiv -49(\text{mod}6)$.

Theorem 1.38 The congruence modulo n is an equivalence relation $\mathbb{Z}$.

Proof: (Do it as an exercise).

Since an equivalence relation on a set produces a partition of that set, then for $n \geq 2$ the congruence relation modulo n produces a partition of set $\mathbb{Z}$ in the following n equivalence classes:

$$[0] = \{ \dots, -2n, -n, 0, n, 2n, \dots \} = \{0 + nx | x \in \mathbb{Z}\}$$
$$[1] = \{ \dots, -2n + 1, -n + 1, 1, n + 1, 2n + 1, \dots \} = \{1 + nx | x \in \mathbb{Z}\}$$
$$[2] = \{ \dots, -2n + 2, -n + 2, 2, n + 2, 2n + 2, \dots \} = \{2 + nx | x \in \mathbb{Z}\}$$
$$\vdots$$
$$[n - 1] = \{ \dots, -n + 1, -1, n - 1, 2n - 1, 3n - 1 \dots \} = \{(n - 1) + nx | x \in \mathbb{Z}\}$$

By the division algorithm, we know that $\forall t \in \mathbb{Z}, t = qn + r$ where $0 \leq r < n$, so $t \in [r]$ or also means that $[t] = [r]$. We use $\mathbb{Z}$ to denote $\{[0], [1], [2], \dots [n - 1]\}$ or when there is no ambiguity we also use 'in $\mathbb{Z}_n = \{0, 1, 2, \dots, n - 1\}$.

We can define closed operations addition and pipeline on the set of equivalence classes of $\mathbb{Z}_n$ as $[a] + [b] = [a + b]$ and $[a] \ \ cdot[b] = [a][b] = [ab]$.

Euclid(30,21)	=Euclid(21,9)
	=Euclid(9,3)
	=Euclid(3,0)
	= 3

Table 6.
Execution steps of the Euclid's algorithm.

Example [1.39] If $n = 7$ then $[2] + [6] = [8] = [1]$ and $[2] \cdot [6] = [12] = [5]$.

Before accepting the definitions of the modular operations $[a] + [b] = [a + b]$ and $[a][b] = [ab]$ we must convince ourselves that they are well defined, that is, if $[a] = [c]$ and $[b] = [d]$ then $[a] + [b] = [c] + [d]$ and $[a][b] = [c][d]$. We will prove that these operations are independent of the choice of elements within a class. So, then $[a] = [c] \Rightarrow a = c + sn$ for some $s \in \mathbb{Z}$ and $[b] = [d] \Rightarrow b = d + tn$ for some $t \in \mathbb{Z}$. Then $a + b = (c + sn) + (d + tn) = (c + d) + (s + t)n$ so that $(a + b) \equiv (c + d) \text{mod} n$ that is $[a + b] = [c + d]$.

In the same way $ab = (c + sn)(d + tn) = cd + (sd + ct + stn)n$ so that $ab \equiv cd \text{mod} n$, that is $[ab] = [cd]$. This leads us to establish the following theorem.

Theorem 1.40 For $n \in \mathbb{Z}^+, n > 1$ under the closed binary operations just defined $\mathbb{Z}_n$ is a commutative ring with the unit [1].

Proof: The proof is left as an exercise for the student. What would have to be done for this is to verify that the ring properties hold for the definition of the addition and multiplication operations on $\mathbb{Z}n$ and the properties of the ring $(\mathbb{Z}, +, \cdot)$.

Before continuing with more theoretical results, let us see the tables of the operations of $+$ and $\cdot$ for $\mathbb{Z}_5$ and $\mathbb{Z}_6$. The $\mathbb{Z}_5$ operation tables are displayed in **Table 7** and operation tables corresponding to $\mathbb{Z}_6$ ring are shown in **Table 8**.

We can see in the tables of the $+$ and $\cdot$ operations of $\mathbb{Z}_5$ all elements other than 0 have an inverse element, which is why this is a field. In the case of the tables of $\mathbb{Z}_6$, as opposed to those of $\mathbb{Z}_5$, only 1 and 5 are elements with inverse (bf units) and 2, 3, 4 are proper divisors of 0. If we obtained the corresponding tables for $\mathbb{Z}_9$, we could see that $3 \cdot 3 = 3 \cdot 6 = 0$, that is there are also proper divisors of 0 which means that it is not enough for n to be an odd number for the set $\mathbb{Z}_n$ to be a field. The latter leads us to establish the following theorem.

Theorem 1.41 $\mathbb{Z}_n$ would be a field if and only if n is a prime number.

Proof: Let n be a prime, and suppose $0 < a < n$. Then the $\gcd(a, n) = 1$ and since the $\gcd(a, n)$ is a linear combination of a and n that is to say that $\exists s, t \in \mathbb{Z}_n, as + tn = 1$. Therefore $as \equiv 1(\text{mod} n)$ i.e. $[a][s] = [1]$, which implies that a is an element with inverse (unit) which makes $\mathbb{Z}_n$ a field. Conversely, if n is not a prime then it can be represented as a product of two numbers, that is $n = n_1 n_2$ where $1 < n_1, n_2 < n$ and if

+	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3
·	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Table 7.
Operations tables of the ring $(\mathbb{Z}_5, +, \cdot)$.

+	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	4	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4
·	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Table 8.
Operations tables of the ring $(\mathbb{Z}_6, +, \cdot)$.

$[n_1] \neq 0$ so as $[n_1] \neq 0$ but $[n_1][n_2] = 0$ means that Z_n is not an integer domain since it has proper divisors of 0 and therefore cannot be a field.

In $\mathbb{Z}_6$ the [5] has an inverse (it is a unit) and on the other hand, [3] is a proper divisor of 0. It is useful and necessary to be able to detect which elements have an inverse (ie they are units) when n is composite. For this, the following theorem is established.

Theorem 1.42 In $\mathbb{Z}_n$, $[a]$ has an inverse (it is a unit) if and only if $\gcd(a, n) = 1$.

Proof: If $\gcd(a, n) = 1$ the proof will be the same as the theorem??. In the reverse direction, let $[a] \in \mathbb{Z}_n$ and $[a]^{-1} = s$. So $[a][s] = 1$, so then $as \equiv 1(\text{mod} n)$ and $as = 1 + tn$ for some $t \in \mathbb{Z}$ but $1 = as + n(-t) \Rightarrow \gcd(a, n) = 1$.

In the following example, we illustrate the application of Euclid's algorithms to obtain multiplicative inverses of elements of a modular field-type group.

Example 1.43 Find $[25]^{-1}$ in $\mathbb{Z}_{72}$. Since $\gcd(25, 72) = 1$ Euclid's gcd algorithm we get the following:

$$72 = 2(25) + 22, 0 < 22 < 25$$

$$25 = 1(22) + 3, 0 < 3 < 22$$

$$22 = 7(3) + 1, 0 < 1 < 3$$

since 1 is the l$'$ast non-zero remainder, we have

$$1 = 22 - 7(3) = 22 - 7[25 - 22] = -7(25) + 8(22) = -7(25) + 8[72 - 2(25)] = 8(72) - 23(25)$$

but

$1 = 8(72) - 23(25) \Rightarrow 1 \equiv (-23 + 72)(25)(\text{mod} 72)$, so then $[1] = [49][25]$ then $[49] = [25]^{-1}$ in $\mathbb{Z}_{72}$.

For a more detailed exposition of the previous results in modular arithmetic, greatest common divisor and Euclid's algorithm, it is recommended to consult these bibliographical references [3, 6, 9, 10].

After this brief overview of the properties of the algebraic ring structure as well as how to work with it in the context of modular arithmetic, we are ready to discuss the advantage of applying results from discrete mathematics to perform calculations of modular arithmetic efficiently. As already explained in the previous paragraphs of this subsection, we can, from the tables of the modular ring in which we are working, obtain all the multiplicative inverses of the ring as well as all the proper divisors of zero in the event that the ring does not be a field. We can do this by generating the sum and product tables of the ring and going through the multiplication table detecting a box where the value is equal to 1 in the case of multiplicative inverses, or that the box is equal to 0 in the case of proper divisors of zero. This procedure for obtaining the proper divisors of zero could be carried out with the following Matlab program.

```
function Fz = FacPropZero(n)
% Modulo n [S, P] = TabOpMod (n)
% Search on the table P all those factors whose product is equal to 0
% CRL 29/sep/2021
[S,P] = TabOpMod(n);
[r,c] = size(P);
k=1;
for i=1:r
   for j=1:r
      if (P(i,j)==0)
         Fz{1,k}=[i-1,j-1];
         k=k+1;
      end
    end
end
end
```

The procedure for obtaining the ring elements that have multiplicative inverse could be carried out with the following Matlab program.

```
function U = unitsZn(n)
% Modulo n [S,P] = TabOpMod(n)
% Search on the table P all those factors whose product is equal to 1
% CRL 22/sep/2021
[S,P] = TabOpMod(n);
[r,c] = size(P);
k=1;
for i=1:r
    for j=1:r
        if (P(i,j)==1)
           U{1,k}=[i-1,j-1];
           k=k+1;
        end
    end
end
end
```

The results of the execution of the Matlab function for obtaining the list of elements of the modular ring that have multiplicative inverse for the case of $n = 5$ and $n = 6$ are the following:

```
>> U=unitsZn(5)
U =
  1×4 cell array
    {1×2 double}   {1×2 double}   {1×2 double}   {1×2 double}
>> U{1,1}
ans =
      1     1
>> U{1,2}
ans =
    2     3
>> U{1,3}
ans =
    3     2
>> U{1,4}
ans =
    4     4
>> U=unitsZn(6)
U =
    1×2 cell array
        {1×2 double}   {1×2 double}
>> U=unitsZn(6)
U =
    1×2 cell array
        {1×2 double}   {1×2 double}
>> U{1,1}
ans =
    1     1
>> U{1,2}
ans =
    5     5
```

The above Matlab routines look good and bad from an algorithmic perspective. The good side is that since it is a direct translation of the operations table generation of a modular ring, the programming of the routines is relatively simple. The negative aspect is in the fact that the size of the arrays to generate the tables can become very large as the number n with respect to which the module is taken grows a lot, that is, the memory occupied will be of the order of $O(n^2)$. It is at this point that the properties and results of discrete mathematics in the subject of modular arithmetic come to the rescue and allow us to obtain more efficient algorithms for obtaining proper divisors of zero as well as the list of elements of the modular ring that have an inverse multiplicative. The efficient algorithm for obtaining the list of elements of a modular ring with multiplicative inverse and the respective Matlab implementation will make use of discrete mathematics results, such as Euclid's algorithm for obtaining the greatest common divisor, as well as the version extension of this, and that will be called by a routine that solves linear modular equations of the type $ax = b\text{mod}n$. The multiplicative inverse x of a number a modulo n will correspond to the solution of the modular linear equation $ax = 1\text{mod}n$. Based on theorem 34, we can implement the Euclid's algorithm in Matlab as follows.

```
function y = mcdEuclides(a,b)
% Recursive function for the GCD using Euclid's algorithm
% The first parameter musy be bigger than the second parameter
if a < b
   temp=a;
   a=b;
   b=temp;
end
if b == 0
   y=a;
else
      y=mcdEuclides(b, mod(a,b));
end
end
```

Euclid's algorithm can be extended to obtain the coefficients a and b of x and y in the relation $d = gcd(a, b) = ax + by$. This will be used for obtaining multiplicative inverses of a modular ring. The Matlab implementation of the extended Euclid algorithm is as follows.

```
Function [d,x,y] = mcdEuclidesExtend(a,b)
% Extended GCD Euclid's algorithm
if a < b
   temp=a;
   a=b;
   b=temp;
end
if b == 0
   d=a;
   x=1;
   y=0;
else
   [d1,x1,y1]=mcdEuclidesExtend(b, mod(a,b));
   d=d1;
   x=y1;
   y=x1-floor(a/b)*y1;
end
end
```

As already mentioned, obtaining the elements of a modular ring that have a multiplicative inverse can be reduced to the problem of calculating the elements x that satisfy the solution to the modular equation $ax = 1 \mathrm{mod} n$. The Matlab implementation of the linear modular equation solver will be the following.

```
Function S = ModLinEqSolv(a,b,n)
% Modular linear equation solver
% That have the form ax=b mod n
% Author: Carlos Rodriguez Lucatero 29/sep/2021
[d,x,y] = mcdEuclidesExtend(a,n);
if (mod(b,d)==0)
     x0=mod((y*(b/d)),n);
     for i=0:d-1
           S(i+1)=mod(x0+(i*(n/d)),n);
     end
```

```
else
    S=-1;
end
end
```

If we call this routine with the parameter $b = 1$, it would give us the result of the value of x, which is the inverse of a in the relation $ax = b \bmod n$, if it exists, or it would return -1 to indicate that the element a of the modular ring does not have a multiplicative inverse. The following routine calls the linear modular equation solver routine to get the list of elements of the modular ring that have an inverse.

```
Function L = unitsZnV3(n)
% Get the list of units of a modular ring
% mod n calling the routine that solves linear modular equations
% Autor: Carlos Rodriguez Lucatero 14/Ene/2022
k=1;
for i=1:n
    S = ModLinEqSolv(i,1,n);
    if (S ~=-1)
        L(1,k)=S;
    end
    k=k+1;
end
if (length(L)==0)
    L=-1;
end
end
```

The elements of the list in 0 correspond to the inverse of the element that does not have an inverse and therefore we would be talking about dividing elements proper to zero of the modular ring in question. The elements in the list that are not null are the inverse of the element of the modular ring represented by the position in the list. For instance, in $(\mathbb{Z}_5, +, \cdot)$, we have that $2 \cdot 3 = 1$ and $4 \cdot 4 = 1$.

```
>> L = unitsZnV3(5)
L =
    1    3    2    4
>> L = unitsZnV3(6)
L =
    1   0   0   0   5
```

6. Conclusions

In this chapter, we address some calculation problems that take place in discrete mathematics and how we can use the theoretical results provided by discrete mathematics itself to be able to carry out these calculations more efficiently. In this chapter, we were able to verify the great utility of the use of generating functions to carry out some calculations more efficiently. Some interesting applications of the use of generating functions to count graphs with some property can be found in refs. [11, 12]. These counting techniques can later be applied to the famous probabilistic method [13–15]. Generating functions are also very useful for counting the number of possible partitions of integers. An excellent text that addresses this interesting topic is [16].

It is true that computers have not stopped increasing their storage capacity as well as the speed of their processing units. However, these resources are not infinite and there are even calculation-intensive problems in topics, such as combinatorics or modular arithmetic that could quickly exhaust these calculation resources. That is why it is worth taking advantage, when possible, of the theoretical results that discrete mathematics itself provides, to efficiently carry out the calculations that it requires. For this, we take two specific topics of discrete mathematics where that is possible. One is the obtainment of elements in arbitrary positions of a numerical sequence and the other topic was the obtainment of elements with multiplicative inverse of the elements that are proper divisors of zero in a modular ring. We illustrate this by programming functions in Matlab. I hope that the chapter will convince you of the goodness of taking advantage of the mathematical results offered by discrete mathematics to perform efficient calculations in the area of discrete mathematics.

Acknowledgements

I would like to thank the Universidad Autonoma Metropolitana, Unidad Cuajimalpa for the support they gave me to make and publish this book chapter.

Author details

Carlos Rodriguez Lucatero
Universidad Autónoma Metropolitana Unidad Cuajimalpa, CDMX, Mexico

*Address all correspondence to: crodriguez@cua.uam.mx

References

[1] Rodríguez-Lucatero C. The Moser's formula for the division of the circle by chords problem revisited. 2017. Available from: https://arxiv.org/abs/1701.08155v1

[2] Feller W. An Introduction to Probability Theory and its Applications. Vol. I. New York, USA: Wiley and Sons Inc; 1968. pp. 52-53

[3] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 3d ed. Massachusetts, USA: The MIT Press; 2009

[4] Miller CD, Heeren VE, Hornsby J. Matematica, razonamiento y aplicaciones. USA: Pearson; 2013

[5] Sedgewick R, Flajolet P. An Introduction to the Analysis of Algorithms, Second Printing. USA: Addison-Wesley; 2001

[6] Grimaldi RP. Discrete and Combinatorial Mathematics: An Applied Introduction. 3rd ed. USA: Addison-Wesley; 1994

[7] Graham RL, Knuth DE, Patashnik O. Concretel Mathematics, 6th Printing. USA: Addison-Wesley; 1990

[8] Wilf HS. Generatingfunctionology. 3th ed. Massachusetts, USA: A. K. Peters Ltd.; 2006

[9] Hardy GH, Wright EM. Introduction to the Theory of Numbers. 5th ed. Oxford, UK: Oxford Science Publications, reprinted; 1998

[10] Vinográdov I. Fundamentos de la Teora de números. URSS: Moscu: Editorial MIR; 1977

[11] Harary F, Palmer EM. Graphical Enumeration. New York, NY, USA; London, UK: Academic Press; 1973

[12] Rodríguez-Lucatero C. Combinatorial Enumeration of Graphs. Rijeka: IntechOpen; 2019

[13] Erdös P. Graph theory and probability. Canadian Journal of Mathematics. 1959;**11**:34-38

[14] Alon N, Spencer JH. The Probabilistic Method. 2nd ed. New York Wiley-Interscience; 2000

[15] Rodríguez-Lucatero C, Alarcón L. Use of enumerative combinatorics for proving the applicability of an asymptotic stability result on discrete-time SIS epidemics in complex networks. MDPI Mathematics Open access Journal. 2019;**7**(1). DOI: 10.3390/math7010030

[16] Andrews GE. In: Rota GC, editor. The Theory of Partitions Encyclopedia of Mathematics and its Applications. Vol. 2. USA: Addison-Wesley; 1976