# *Recursion and Applications*

*Xiaoyi Cui*

*April 9, 2018*

This is just a brief summary of the contents of the lectures. Please note: most of the calculations and demonstrations are neglected. The part on formal languages and finite state machines is neglected as we shall closely follow our textbook. I claim no originality of those contents.

## *Contents*



## *Recursion method*

**Definition 1** • *Recursion is a process of expressing the solution to a problem in terms of a simpler version of the same problem.*

- *A recursive algorithm is an algorithm that invokes itself during execution.*

**Example 1** *Pascal's Triangle and factorials.*

Guides for creating a recursive algorithm:
Step 1: Identify how to reduce the problem into smaller versions of itself.
Step 2: Identify one or more instances of the problem that can be directly solved.
Step 3: Determine how the solution can be obtained by combining the solutions to one or more smaller versions.
Step 4: Verify that the invocations in step 3 are within bounds.
Step 5: Assemble the algorithm.

**Example 2** *Determine the number of* $1$*'s in the binary representation of the decimal integer* $n$*.*

**Example 3 (Sierpinski Curve)**

## *When should recursion be avoided*

Recursion involves calling the program itself multiple times; each time a new region of memory is needed, so the “spacial" and “time" complexities are affected.

Tail-End recursion: the only recursive invocation it makes occurs at the last line of the algorithm.

**Example 4** *Compare the following two codes that compute the factorials.*

```
integer factorial(integer n)
  if n = 1
    return 1
  else
    return n*factorial(n-1)
end factorial
```

*and*

```
integer nfact(integer n)
  nf= n
  i= n
  while i > 1
    i = i - 1
    nf = nf*i
  return nf
end nfact
```

Redundant recursion: when an algorithm directly or indirectly invokes multiple instances of the same smaller version.

**Example 5** *Compare the following two codes which computes exponential of 2.*

```
integer tn(integer n)
  if n = 0
    return 1
  else
    return tn(n-1) + tn(n-1)
end tn
```

*The above code is even worse than tail-end. The correct way:*

```
integer twoExpn(integer n)
  tn= 1
  i = 1
  while i <= n
    tn - 2*tn
    i - i + 1
  return tn
end twoExpn
```

## *Recursive relations*

**Definition 2** *A recursively defined function is a function, f, whose domain is the set of non-negative integers and for which $f(0)$ is known and $f(n)$ is defined in terms of some subset of $\{f(0), f(1).....f(n-1)\}$.*

**Example 6** *Recursive definition of* $n!$ *and geometric/arithmetic progression.*

**Definition 3** *Let* $\{a_n | n = 0, 1, 2 \cdots\}$ *be a sequence. A recurrence relation for* $\{a_n\}$ *is a formula that expresses an in terms of some subset of* $\{a_0, a_1, \cdots a_{n-1}\}$*. The recurrence relation must also specify one or more base conditions. Given a recurrence relation, the sequence it generates is called the solution of the recurrence relation.*

**Example 7** *The Fibonacci sequence.*

**Exercise 1** *Solve the recurrence relation* $a_0 = 1, a_n = N \cdot a_{n-1} + 1$.

**Definition 4** *A recurrence relation for the sequence* $\{a_n\}$ *is called homogeneous if every term on the right-hand side of the recurrence contains a factor of the form* $a_j$*, for some integer* $j$*. The recurrence relation has constant coefficients if* $n$ *does not appear in any term involving some* $a_j$ *except in subscripts. A recurrence relation is called linear if no term contains more than one factor of the form* $a_j$ *(even with different values of* $j$*), and no factor of the form* $a_j$ *appears in a denominator, as an exponent, or as part of a more complex function.*

**Definition 5** *A linear homogeneous recurrence relation with constant coefficients of degree* $k$ *is a recurrence relation that can be written in the form*

$$a_n = c_l a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

*for some* $k$ *with* $1 < k$ *and* $c_k \neq 0$*. The constant* $k$ *is called the degree of the recurrence relation. The constants,* $c_j$*, are called the coefficients of the recurrence relations.*

*The characteristic equation of the recurrence relation*

$$a_n = c_l a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

*is*

$$X^k - c_1 X^{k-l} - c_2 X^{k-2} - \cdots - c_{k-l} X - c_k = 0.$$

**Theorem 1** *Suppose the sequence* $\{a_n\}$ *is generated by the recurrence relation*

$$a_n = c_l a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

*If* $a_n = \theta r^n$ *also generates this sequence, then* $r$ *is a root of the characteristic equation. Conversely, if* $r$ *is a root of the characteristic equation, then any expression of the form* $\theta r^n$ *generates a sequence that is a solution to the recurrence relation.*

**Theorem 2** *Suppose the characteristic equation of the degree* $k$ *recurrence relation*

$$a_n = c_l a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

*has k distinct roots, $r_1$, $r_2$ ... , $r_k$. Then for any choice of constants, $\theta_0$, $\theta_1$, ..., $\theta_k$ the closed-form expression*

$$a = \sum_{i=1}^{k} \theta_i r_i^n$$

*generates a solution to the recurrence relation. In addition, if the k initial values, $a_0$, $a_1$, ... , $a_{k-1}$, are specified, it is always possible to find unique values, $\theta_1$, $\theta_2$, .... $\theta_k$, so that the recurrence relation generates the solution that matches those initial values.*

## *Generating functions*

**Definition 6** *Let $\{a_0, a_1, a_2 \cdots\}$ be a sequence of real or complex numbers. The generating function, $G(z)$, is a function with single variable $z$, whose formal Taylor expansion at $z \to 0$ is given by the formal power series $a_0 + a_l z + a_2 z^2 + \cdots + a_k z^k \cdots$. For most of the time, we shall identify $G(z)$ with its formal Taylor expansion [1]. If the sequence is finite, then $G(z)$ is a polynomial.*

[1] Here we work with the ring of formal power series $\mathbb{R}[[z]]$ instead of the polynomial ring $\mathbb{R}[z]$

**Example 8** $\frac{1}{1-z}$ *and* $\frac{1}{1+z}$.

**Proposition 1 (Shifting Generating Functions)** *Let $G(z)$ be the generating function for the sequence $\{a_0, a_1, a_2 \cdots\}$. Then $\sum_{k=m}^{\infty} a_{k-m} z^k = z^m G(z)$.*

**Proposition 2 (Multiplying Generating Functions)** *Let $F(z) = \sum_{k=0}^{\infty} f_k z^k$ and $G(z) = \sum_{k=0}^{\infty} g_k z^k$ be two generating functions. The generating function that is the product of F and G is $P(z) = F(z)G(z) = \sum_{k=0}^{\infty} (\sum_{j=0}^{k} f_{k-j} g_j) z^k$.*

**Definition 7 (The Derivative of a Generating Function)** *Let $A(z) = \sum_{k=0}^{\infty} a_k z^k$ be a generating function. Then its derivative is denoted by $A'(z)$ and is defined by $A'(z) = \sum_{k=1}^{\infty} k a_k z^{k-1}$.*

Using the above results, sometimes we can recover the exact expression for $G$ from the information of $\{a_n\}$.

**Exercise 2** *Find the generating function for the following sequences: (1) $-1, 1, -1, 1, \cdots$, (2) $1, 0, 0, 1, 0, 0, 1, \cdots$, (3) $1, 2, 2^2, 2^3, 2^4, \cdots$, (4) $1, m, \begin{pmatrix} m+1 \\ 2 \end{pmatrix}, \begin{pmatrix} m+2 \\ 3 \end{pmatrix}, \begin{pmatrix} m+3 \\ 4 \end{pmatrix}, \cdots$.*

**Theorem 3 (Newton's Binomial Theorem)** *Let $w$ and $z$ be real numbers with $|\frac{z}{w}| < 1$. Then for any real number, $u$, $(w + Z)^u = \sum_{k=0}^{\infty} \begin{pmatrix} u \\ k \end{pmatrix} w^{u-k} z^k$.*

We can solve non-constant, non-homogeneous cursive relations using the generating function method.

**Example 9** $a_n = (n-1)a_{n-1} + 1$ *with* $a_0 = 1$.

**Example 10** $a_n = a_{n-1} + n$ *with* $a_0 = 3$.

**Exercise 3 (Tower of Hanoi)** *In 1883, the French mathematician Edouard Lucas created an interesting puzzle, called the Tower of Hanoi. The "game board" consists of eight disks of uniformly increasing size, together with a board containing three pegs or dowels, labeled A, B, and C, each able to hold all eight disks.*

*A recurrence relation creeps in if we ask the question: What is the minimum number of moves,* $H_n$*, it will take to solve a Tower of Hanoi puzzle with n disks? Try to solve the problem using generating functions.*

*Hint: Firstly show that the recursive relation for the Tower of Hanoi puzzle is* $Ho = 0, H_1 = 1$ *and* $H_n = 2H_{n-1} + 1, \forall n > 1$*. Then solve the relation to show that* $H_n = 2^n - 1$.