Overview

• HW#8 Due Now. Pick up old HWs and Exams
• Review next Tuesday: Study HW8 questions.
• Last Lecture: Counting completed
  – Permutations with repeats
  – Combinations with repeats
  – Pascal’s identity
  – Binomial coefficients
  – Pascal’s triangle

• This Lecture
  – Recursion: sequence, function, sets (iterative vs recursive)
  – Fibonacci numbers, Ackerman’s function
  – Recursive algorithm
  – Deriving recurrence equation from a sequence
  – Solving recurrence equation (yielding iterative equation)
Recursion: one way to define sequences, sets, functions, and algorithms

- Recursive definition: an object is defined in terms of itself (object appears in both sides of equation).

Example: The sequence \( \{a_n\} \) of powers of 2

\[1, 2, 4, 8, 16, 32, 64, 128, 256, \ldots\]

Use iterative function as definition

\[a_n = 2^n \text{ for } n = 0, 1, 2, \ldots\]

Use recursive function as definition (defined recursively)

\[a_0 = 1\]
\[a_{n+1} = 2a_n \text{ for } n = 0, 1, 2, \ldots\]

- Recursion is a principle closely related to mathematical induction,

Example: Recursively define the factorial function

\[f(n) = n! = n \times (n-1) \times \ldots \times 1\], for \( n > 0 \) and \( 0! = 1 \).

\[f(0) = 1\]
\[f(n) = n \times f(n-1) \quad (**)\]

\[f(0) = 1; \quad f(1) = 1 \times 1 = 1;\]
\[f(2) = 2 \times f(1) = 2 \times 1 = 2; \quad f(3) = 3 \times f(2) = 3 \times 2 = 6;\]
\[f(4) = 4 \times f(3) = 4 \times 6 = 24;\]

A famous example: The Fibonacci numbers

\[f(0) = 0, f(1) = 1\]
\[f(n) = f(n-1) + f(n-2) \quad (**)*\]
\[f(0) = 0; \quad f(1) = 1;\]
\[f(2) = f(1) + f(0) = 1 + 0 = 1;\]
\[f(3) = f(2) + f(1) = 1 + 1 = 2;\]
\[f(4) = f(3) + f(2) = 2 + 1 = 3;\]
\[f(5) = f(4) + f(3) = 3 + 2 = 5;\]
\[f(6) = f(5) + f(4) = 5 + 3 = 8;\]
Another famous example: The Ackerman’s function

\[
A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0 \\
  A(m-1, 1) & \text{else if } n = 0 \\
  A(m-1, A(m, n-1)) & \text{otherwise} 
\end{cases}
\]

\[
A(0, n) = n + 1 \\
A(1, n) = n + 2 \\
A(2, n) = 2n + 3 \\
A(3, n) = 2^{n+3} - 3 \\
A(4, n) = 2^{2^n} - 3 \\
A(5, n) = 2^{2^{2^n}} - 3 \\
A(6, n) = 2^{2^{2^{2^n}}} - 3 \\
A(7, n) = 2^{2^{2^{2^{2^n}}}} - 3 \\
A(8, n) = 2^{2^{2^{2^{2^{2^n}}}}} - 3 \\
A(9, n) = 2^{2^{2^{2^{2^{2^{2^n}}}}}} - 3 \\
A(10, n) = 2^{2^{2^{2^{2^{2^{2^{2^n}}}}}}}-3
\]

This function value grows rapidly; even for small inputs, for example \(A(4,1) = 65533\) \(\text{ and } A(4,2) = 2^{65533} - 3\)

It can be shown that:

\[
A(0, n) = n + 1 \\
A(1, n) = n + 2 \\
A(2, n) = 2n + 3 \\
A(3, n) = 2^{n+3} - 3 \\
A(4, n) = 2^{2^n} - 3 \quad \text{// tower of exponents, } n+3 \text{ twos}
Recursive Algorithm

An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input. (Like tree traversal methods and pascal’s triangle and binary search)

Example: Recursive Algorithm for factorial(n)

integer function factorial(n: positive integer)
    if (n == 1) then return 1
    else return n*factorial(n-1)

Example: Recursive Euclidean Algorithm

integer function gcd(a, b: nonnegative integers with a < b)
    if (a == 0) then return b
    else return gcd(b mod a, a)

Example: Recursive Fibonacci Algorithm

integer function fibo(n: nonnegative integer)
    if (n == 0) then return 0
    else if (n == 1) then return 1
    else return (fibo(n – 1) + fibo(n – 2))

fibo(4) = fibo(3) + fibo(2)
= [fibo(2)+fibo(1)] + fibo(2)
= [[fibo(1)+fibo(0)]+fibo(1)]+fibo(2)
= [[fibo(1)+ 0]+fibo(1)]+fibo(2)
= [[1+0]+1]+fibo(2)
= [1+1]+fibo(2)
= 2 + fibo(1) + fibo(0)
= 2 + 1 + 0
= 3
For every recursive algorithm, there is an equivalent iterative algorithm!!! (Solve recursion)

- Recursive algorithms are often shorter, more elegant, and easier to understand than their iterative counterparts.
- However, iterative algorithms may be more efficient in their use of space and time.
- Recursive definitions that specifies one or more initial terms and rules for determining subsequent terms is called recurrence equation/relation (=recursive function)

See (**) from previous pages in the last lecture

- Note: In computer science, recurrence equations can be used to define recursive algorithms and to analyze run-time of recursive programs

How to define sequence $A(n)$ recursively?

- List the sequence: $A(0), A(1), A(2), A(3), ...$
  1) Evaluate difference $(A(n)-A(n-1))$ or ratio $(A(n)/A(n-1))$ between successive terms to find a constant
  2) Write the equation: $F(A(n), A(n-1), ...)$ = constant
  3) Solve it for $A(n)$

How to define function $f(n)$ recursively?

- List the function outputs $f(n)$ for $n=0, 1, 2, ...$
- Do the same as above.
Example: write a recurrence relation for an **arithmetic**
progression with initial term \( a \) and common *difference* \( d \),
\( a, a+d, a+2d, \ldots \)

\[
A(n) = \begin{cases} 
  a & n = 1 \\
  A(n-1)+d & \forall \ n \geq 2 
\end{cases}
\]

Example: write a recurrence relation for an **geometric**
progression with initial term \( a \) and common *ratio* \( r \),
\( a, ar, ar^2, \ldots \)

\[
A(n) = \begin{cases} 
  a & n = 1 \\
  r \cdot A(n-1) & \forall \ n \geq 2 
\end{cases}
\]

Example: combination of an **arithmetic** and **geometric**
progression?

\[
A_1 = \begin{cases} 
  a = \frac{A_{n-1}^2}{A_{n-2}} + A_{n-1} \\
  A_1 = 1, \ A_2 = 2 
\end{cases}
\]
How to solve recurrence equation?

**Example:**

\[
A(n) = \begin{cases} 
2 \\
2A(n-1)
\end{cases} \quad \forall \ n \geq 2
\]

- \( A(n) = 2A(n-1) 
- = 2 \times 2 \times A(n-2) = 2^2 \times A(n-2) 
- = 2^2 \times 2 \times A(n-3) = 2^3 \times A(n-3) 
- \ldots 
- = 2^k \times A(n-k) 
- \ldots 
- = 2^n \times A(1)

What is the value of \( k \) when we have \( A(1) \)?

\[ \therefore n-k = 1 \ (k=n-1) \text{ so } A(n) = 2^{n-1} \times A(1) = 2^{n-1} \times 2 = 2^n \]

**Example:**

\[
T(n) = \begin{cases} 
1 \\
T(n-1)+3
\end{cases} \quad \forall \ n \geq 2
\]

- \( T(n) = T(n-1) + 3 
- = (T(n-2) + 3) + 3 = T(n-2) + 3 + 3 = T(n-2) + 2 \times 3 
- = (T(n-3) + 3) + 3 + 3 = T(n-3) + 3 + 3 + 3 = T(n-3) + 3 \times 3 
- \ldots 
- = T(n-k) + k \times 3

\[ \therefore \text{when } n-k = 1 \ (k=n-1), \quad T(n) = T(1) + (n-1) \times 3 = 1 + 3n - 3 = 3n-2 \]
e.g. \( T(n) = \begin{cases} 
1 & n = 1 \\
2T(n-1) + 1 & \forall \ n \geq 2
\end{cases} \)

\[
T(n) = 2T(n-1) + 1 \\
= 2(2T(n-2)+1) + 1 \\
= 2^2T(n-2)+2+1 \\
= 2^2(2T(n-3)+1)+2+1 \\
= 2^3T(n-3)+2^2+2^1+2^0 \\
\vdots \\
= 2^{n-k}T(n-k)+2^{k-1}+2^{k-2}+\ldots+2^1+2^0 \\
= 2^n + 2^{n-1} + \ldots + 2^1 + 2^0
\]