
[§5.6-5.8] Recurrence Relations

1 Recursive Algorithms

- A recursive algorithm is an algorithm which **invokes itself**.
- A recursive algorithm looks at a problem **backward** -- the solution for the current value *n* is a modification of the solutions for the previous values (e.g. *n-1*).
- Basic structure of recursive algorithms:

```
procedure foo(n)
1.  if n satisfy some condition then  // base case
2.    return (possibly a value)
3.  else
4.    (possibly do some processing)
5.    return (possibly some processing using)
6.           foo(n-1)                // recursive call
```

- *Examples:*
  1. **Factorial (n!)**

```
0! = 1
1! = 1
2! = 1 * 2                      = 1! * 2
3! = 1 * 2 * 3                  = 2! * 3
n! = 1 * 2 * 3 * .. * (n-1) * n = (n-1)! * n
```

   - Recursive algorithm for factorial:

     Input: A positive integer *n* >= **0**
     Output: *n!*

```
procedure factorial(n)
1. if n = 0 then
2.   return 1                 // base case, 0! = 1
3. return n * factorial(n-1)  // recursive call
```

   - Trace of procedure calls when n = 3

     factorial(3)
       factorial(2)
         factorial(1)
           factorial(0)
           return 1
         return 1 * 1 = **1**
       return 2 * **1** = ***2***
     return 3 * ***2*** = 6

2. **Compound interest**

   With an initial amount of $100 and 22 % annual interest, the amount at the end of the nth year, A_n, is expressed recursively as:

$$A_n = \begin{cases} (1.22) \cdot A_{n-1}, & x \geq 1 \\ 100.0, & x = 0 \end{cases}$$

   - A general <u>recursive algorithm</u> for compound interest after *n* years (where n >= 0) with initial amount *a* and interest rate *r* is

```
procedure compound_interest(n, a, r)
1. if (n == 0) then
2.   return a
3. return ((1 + r) * compound_interest(n-1, a, r))
end compound_interest
```

3. **Greatest Common Divisor (gcd)**

   gcd(a, b) = gcd(b, a mod b), b > 0

   <u>Recursive algorithm for gcd:</u>

   Input: Integers a, b
   Output: gcd(a, b)

```
procedure recursive_gcd(a, b)
1. if a < b then
2.   swap(a, b)
3. if b = 0 then                      // base case
4.   return a
5. return recursive_gcd(b, a mod b)  // recursive call
```

<u>Proof of Correctness using Induction</u>

- Correctness of a recursive algorithm is often shown using mathematical induction, due to the structural similarity -- **base case** and **recursive/inductive case**.
- *Example:* Factorial

```
procedure factorial(n)
1. if n = 0 then
2.   return 1                 // base case, 0! = 1
 2. return n * factorial(n-1)  // recursive call
```

  Theorem: The algorithm *factorial(n)* outputs the value of n!, where n >= 0.

  <u>Proof:</u>

  Basic Step (n = 0): In line 2, 0! = 1 is correctly outputted.

  Inductive Step: Assume the algorithm correctly computes (n-1)! for *factorial(n-1)*. Then for *factorial(n)*, line 3 computes n * (n-1)! = n!, which is the correct value for *factorial(n)*.

- <u>NOTE:</u> The proof above uses "**strong mathematical induction**", which is defined as follows.

Principle of Strong Mathematical Induction: Let *P(n)* be a predicate that is defined for integers *n*, and let *a* and *b* be fixed integers with *a* <= *b*. Suppose the following two statements are true:

1. *P(a), P(a+1),..,P(b)* are all true.
2. For all integers *k* >= *b*, if *P(i)* is true for all integers *i* with *a* <= i < *k*, then *P(k)* is true.

Then the statement "For all integers *n* >= *a*, *P(n)*" is true.

- Notice here
  - k >= b >= a. So when b != a, the inductive step starts from some number bigger than a.
  - P(k) -- **not** P(k+1) in this definition – P(k) depends on all of P(a), P(a+1),.., P(b), P(b+1),..P(k-1). In other words, the inductive hypothesis assumes PREVIOUS terms.

More examples of recursive algorithms

- **Usefulness** of recursive algorithms
  - For some problems, it is difficult to derive explicit formulas directly.
  - Many computer programs have recursive functions because of the reason above.

- *Example 1:* **Fibonacci sequence**

$$f_n = \begin{cases} f_{n-1} + f_{n-2}, \text{when } n \geq 2 \\ 1, \text{when } n = 1 \\ 1, \text{when } n = 0 \end{cases}$$

f_0	1
f_1	1
f_2	$f_1 + f_0 =$
f_3	$f_2 + f_1 =$

Recursive algorithm for Fibonocci:

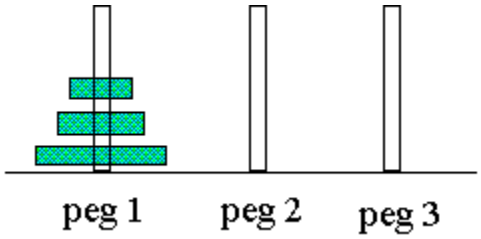
Input: A positive integer *n* >= 0
Output: f_n

```
procedure fibonocci(n)

        (will do in the class)
```

- *Example 2:* **Tower of Hanoi problem**

There are 3 pegs, and *n* number of disks of various size are stacked in a peg, from the smallest disk at the top to the largest disk at the bottom. How many moves are required to transfer those disks to another peg, if a disk can only be moved one at a time and a smaller disk can only be placed on a bigger disk???



A good web app on Hanoi Puzzle
(**http://www.softschools.com/games/logic_games/tower_of_hanoi/**)

C_n, the number of moves required to transfer *n* disks is expressed by a recursive relation:

$$C_n = \begin{cases} 2 \cdot C_{n-1} + 1, & n > 1 \\ 1, & n = 1 \end{cases}$$

1) C_{n-1} moves to transfer the top *n-1* disks to the intermediate peg (say peg 2);
2) 1 move to transfer the bottom disk to the destination peg (say peg 3);
3) C_{n-1} moves to transfer *n-1* disks from the intermediate peg to the destination peg.

==========

2 Recurrence Relations

2.1 Intro

- Consider the following sequence of numbers:

  a_1= 5, a_2= 8, a_3= 11, a_4= 14, a_5= 17, ...

  It seems there is a relation between any two adjacent terms a_n and a_{n+1}, namely **$a_{n+1} = a_n + 3$**, for any n (where n >= 1).

  There are two ways to express the relation above:

  1. **Recursively**

     $$a_n = \begin{cases} a_{n-1} + 3, & n \geq 2 \\ 5, & n = 1 \end{cases}$$

  2. Directly

     $\forall n \geq 1. a_n = 3n + 2$

- A recurrence relation is defined by expressing the n^{th} term (a_n) in terms of its **predecessor**(s) (a_{n-1}). A recurrence relation also includes initial conditions (as base case(s)) where the sequence starts.

- *Example 1:* Compound interest

With an initial amount M and r% annual interest, the amount at the end of the nth year, A_n, is expressed recursively as:

$A_n = (1 + r)*A_{n-1}$, for all n >= 1

With an initial condition

$A_0 = M$

- *Example 2:* Fibonacci numbers

  $f_n = f_{n-1} + f_{n-2}$, for all n >= 2

  with initial conditions

  $f_0 = 1$ and $f_1 = 1$

2.2 Solving Recurrence Relations

- **Closed form** of a recurrence relation
  - We are also interested in knowing a direct form, since the recurrence relation does not show it explicitly.
  - Direct forms do not involve recurrence -- **closed forms**.
  - Closed forms allows us to plug in a value for *n* and get the result immediately.

  Example: Sequence 5, 8, 11, 14, 17, ...

  Closed form is $a_n = 3n + 2$, for all n >= 1. So,

  $a_1 = 3*1 + 2 = 5$
  $a_2 = 3*2 + 2 = 8$ etc.

- There are two ways to derive a closed form for a given recurrence relation.
  1. Iteration
  2. Characteristic functions

(1) Iteration Method

- Steps:
  1. Write the recurrence equation for a_n (This equation has terms in a_{n-1}, a_{n-2}, etc.)
  2. Replace each of a_{n-1}, a_{n-2} by its recursive expression
  3. Continue until you see a **pattern** developing
  4. Prove (usually by induction) that the pattern found is correct.

- *Example 1:*

  $a_n = a_{n-1} + 3$, for all n >= 2

  with initial condition $a_1 = 5$

$a_n = a_{n-1} + 3$

[$a_{n-1} = a_{n-2} + 3$ ... a side note

$= a_{n-2} + 3 + 3$ ... substitute $a_{n-2} + 3$ for a_{n-1}

$= a_{n-3} + 3 + 3 + 3$ ... substitute $a_{n-3} + 3$ for a_{n-2}

...

$= a_{n-k} + k*3$ ... **derive a <u>pattern</u> for an arbitrary kth iteration**

...

$= a_1 + (n-1)*3$ ... k = (n-1) since n-(n-1) = 1

Then, we can plug in $a_1 = 5$. We get

$a_n = 5 + (n-1)*3 = 3n - 3 + 5 = 3n + 2$ ... **closed form**

- *<u>Example 2:</u>*

  $S_n = 2*S_{n-1}$, for all n >= 1

  with initial condition $S_0 = 1$

  $S_n = 2*S_{n-1}$

  [$S_{n-1} = 2*S_{n-2}$ ... a side note

  =

  (will do in the class)

- *<u>Example 3:</u>* Tower of Hanoi

  Number of moves required for n disks is

  $C_n = 2C_{n-1} + 1$, for all n >= 2

  with initial condition $C_1 = 1$.

  $C_n = 2C_{n-1} + 1$

  [$C_{n-1} = 2C_{n-2} + 1$ ... a side note

  $= 2(2C_{n-2} + 1) + 1$ ... substitute $2C_{n-2} + 1$ for C_{n-1}

  $= 4C_{n-2} + 2 + 1$

  $= 4(2C_{n-3} + 1) + 2 + 1$ ... substitute $2C_{n-3} + 1$ for C_{n-2}

  $= 8C_{n-3} + 4 + 2 + 1$

  ....

  $= 2^k*C_{n-k} + 2^{k-1} + ... + 2^1 + 2^0$ ... **a general pattern!!**

  ....

  $= 2^{(n-1)}*C_1 + 2^{n-2} + 2^{n-3} + ... + 2^1 + 2^0$ ... k = (n-1) since n-(n-1) = 1

  Then we can plug in $C_1 = 1$ in the above, obtaining

  $$C_n = 2^{n-1} + 2^{n-2} + 2^{n-3} + \cdots + 2^1 + 2^0$$
  $$= \frac{2^n - 1}{2 - 1}$$ ... by summation of geometric sequence
  $$= 2^n - 1$$

- More examples:
  a. T(n) = -3*T(n-1), T(0) = 2
  b. T(n) = T(n-1) + 1, T(0) = 0
  c. T(n) = 2*T(n-1) + 3, T(0) = 0
  d. T(n) = T(n/2) + 1, T(1) = 0

Proving the correctness of the derived formula

- Now we know how to apply the iteration method to derive explicit formula (i.e., closed form) for recurrence relations of the form:
  - T(n) = T(n-c) + a
  - T(n) = T(n/c) + a

  where a is some number (constant or some expression which depends on n), and c is a constant.

- The next order of business is to **prove** the correctness of a formula (to verify that you didn't make any calculation mistake -- hopefully!!).

- **Example1:** Tower of Hanoi

  Let T(n) be the number of moves required for n disks. It can be defined in two ways, by recurrence relation and by closed formula, as:

  - Recurrence relation

    T(n) = 2*T(n-1)+ 1, for all n >= 2, and
    T(1) = 1.

  - Closed form

    $T(n) = 2^n - 1$, for all n >= 1.

  So we need to prove that the two formulas are the same, for all n >= 1.

  Theorem: Show if T(1), T(2),.., T(n) is the sequence defined by

  T(n) = 2*T(n-1) + 1, for all n >= 2, and T(1) = 1

  then, $T(n) = 2^n - 1$, for all n >= 1.

  Proof: by strong mathematical induction (with one base case).

  Basic Step (n = 1):

  - Recurrence: T(1) = 1 ... by definition
  - Closed form: $T(1) = 2^1 - 1 = 1$

  Therefore, the closed formula holds for n = 1. ..... (A)

  Inductive Step:

Assume the closed formula holds for n-1, that is, $T(n-1) = 2^{n-1} - 1$, for all n >= 2.
Show the formula also holds for n, that is, $T(n) = 2^n - 1$.

T(n) = 2*T(n-1) + 1 ... by recursive definition
= $2*(2^{n-1} - 1) + 1$ ... by inductive hypothesis
= $2*(2^{n-1}) - 2 + 1$
= $2*(2^{n-1}) - 1$
= $2^n - 1$

Therefore, the closed formula holds for n >= 2. ..... (B)

By (A) and (B), the closed formula holds for all n >= 1. QED.

- **Example 2:** Show if T(1), T(2),.., T(n) is the sequence defined by

  T(n) = -3*T(n-1) for all n >= 1, and T(0) = 2

  then, $T(n) = 2*(-3)^n$

  Proof:

  (will do in the class)

(2) Characteristic Equation

- A recurrence relation that involves only the preceding term can be easily solved by iteration. Unfortunately, things tend to get a little messier if there is more than one preceding terms.

  Example: How can we derive a closed form for $f_n = f_{n-1} + f_{n-2}$?

- To solve those kinds of patterns, we need more powerful tools.

Second order, Linear Homogeneous recurrence with Constant Coefficients

- **Algebra review**

  The roots r of a quadratic equation $ax^2 + bx + c = 0$ are
  $$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- **Definition:** A second order linear homogeneous recurrence with constant coefficients is a recurrence relation of the form

  $a_n = c_1*a_{n-1} + c_2*a_{n-2}$ for all integers n >= some fixed integer,

  where c_1, c_2 are fixed real numbers with c_2 != 0.

- Notes on terminology:
  - second order -- expressions on the RHS contains the two previous terms (a_{n-1} and a_{n-2})
  - linear -- each term on the RHS is at the first power (i.e., $(a_{n-1})^1$ and $(a_{n-1})^1$)
  - homogeneous -- every term on the RHS involves a_i
  - constant coefficients -- c_1, c_2 are fixed constants, not involving n

- **Examples:**

  State whether each of the following is a second order linear homogeneous recurrence with constant coefficients.

$a_n = 3a_{n-1} - 2a_{n-2}$	
$a_n = 3a_{n-1} - 2a_{n-2} + 5a_{n-3}$	
$a_n = (1/3)*a_{n-1} - (1/2)*a_{n-2}$	
$a_n = 3(a_{n-1})^2 + 2a_{n-2}$	
$a_n = 2a_{n-2}$	
$a_n = 3a_{n-1} + 2$	
$a_n = (-1)^n*3a_{n-1} + 2a_{n-2}$	

- **Definition:** Given a recurrence relation $a_n = A*a_{n-1} + B*a_{n-2}$, the characteristic equation of the relation is

  $t^2 - At - B = 0$

Theorem: Consider a recurrence relation $a_n = A*a_{n-1} + B*a_{n-2}$. Let x and y be the **roots** of the characteristic equation for this relation. Then there are any numbers C, D such that

a) If x != y, then $a_n = C*x^n + D*y^n$ ... Distinct roots case

b) If x = y, then $a_n = C*x^n + D*n*y^n$ ... Single root case

Proof: See the proof on p. 320 for the distinct root case.

- How can we use this theorem to derive closed forms???
- **Example 1 (distinct roots):** Find the closed form for a recurrence relation

  $a_n = a_{n-1} + 2a_{n-2}$, for all integers n >= 2

  with initial conditions $a_0 = 1$ and $a_1 = 8$.

  Solution:

  The characteristic equation for this relation is $t^2 - t - 2 = 0$. Computing the roots of this equation,

  $t^2 - t - 2 = 0$, so $(t - 2)(t + 1) = 0$

  we get $t_1 = 2$ and $t_2 = -1$. Then from the theorem above, we know the closed form is written as:

$a_n = C*(2)^n + D*(-1)^n$ ..... **(*)**

Now, we must solve for C and D (at this point, they are unknown). To do so, we use the initial conditions -- by plugging in n = 0 (for a_0) and n = 1 (for a_1) in **(*)**, and solve the two equations.

$a_0 = C*(2)^0 + D*(-1)^0 = C + D = 1$ ..... **(1)**
$a_1 = C*(2)^1 + D*(-1)^1 = 2C - D = 8$ ..... **(2)**

To solve those equations, there are several ways. Here is one way using substitution.
We know C = 1 - D by **(1)**. Substituting this in **(2)**, we get

2(1 - D) - D = 8
2 - 2D - D = 8
3D = -6
D = -2, so C = 1 + 2 = 3

Then we have completely solved the closed form, which is now

$a_n = 3*(2)^n + (-2)*(-1)^n$ ..... **(**)**

- **Example 2 (single root):** Find the closed form for a recurrence relation

$a_n = 4a_{n-1} - 4a_{n-2}$, for all integers n >= 2

with initial conditions $a_0 = 1$ and $a_1 = 3$.

Solution:

The characteristic equation for this relation is $t^2 - 4t + 4 = 0$. Computing the roots of this equation,

$t^2 - 4t + 4 = 0$, so $(t - 2)^2 = 0$

we get $t = 2$. Then from the theorem above, we know the closed form is written as:

$a_n = C*(2)^n + D*n*(2)^n$ ..... **(*)**

Now, we must solve for C and D (at this point, they are unknown). To do so, we use the initial conditions -- by plugging in n = 0 (for a_0) and n = 1 (for a_1) in **(*)**, and solve the two equations.

$a_0 = C*(2)^0 + D*n*(2)^0 = C = 1$ ..... **(1)**
$a_1 = C*(2)^1 + D*n*(2)^1 = 2C + 2D = 3$ ..... **(2)**

We know C = 1 by **(1)**. Substituting this in **(2)**, we get

2(1) + 2 D = 3
2D = 1
D = 1/2

Then we have completely solved the closed form, which is now

$a_n = 1*(2)^n + (1/2)*n*(2)^n$

$= \mathbf{(1 + (1/2)*n) * (2)^n}$ ..... **(**)**

- More examples**:** Find a closed form and theta complexity for each of the following recurrence relations.
  a. $a_n = 4a_{n-2}$, for all integers n >= 2, and $a_0 = 1$, $a_1 = -1$.
  b. $a_n = 2a_{n-1} + a_{n-2}$, for all integers n >= 2, and $a_0 = 1$, $a_1 = 4$.
  c. $a_n = 10a_{n-1} - 25a_{n-2}$, for all integers n >= 2, and $a_0 = a_1 = 1$.
  d. $a_n = a_{n-1} + a_{n-2}$, for all integers n >= 2, and $a_0 = 1$, $a_1 = 2$.