Recurrence Relation

By: Ashok Basnet

Recurrence Relation

- A *recurrence relation* (R.R., or just *recurrence*) for a sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more previous elements $a_0, \ldots, a_{n-1}$ of the sequence, for all $n \geq n_0$.
- A particular sequence (described non-recursively) is said to *solve* the given recurrence relation if it is consistent with the definition of the recurrence.
  - A given recurrence relation may have many solutions.
- Consider the recurrence relation
  - $a_n = 2a_{n-1} - a_{n-2} \quad (n \geq 2)$.
- Which of the following are solutions?

$$a_n = 3n$$
$$a_n = 2^n$$
$$a_n = 5$$

Recurrence Relation

- E.g.: $S = \{2 , 2^2 , 2^3 , ….. 2^n \}$

  $S = \{ S_n \} = \{ 2^n \}$

- Here Initial Condition is $S_1 = 2$. So Recurrence relation can be expressed as:

  $\mathbf{S_n = 2S_{n-1} , n \geq 2}$

- E.g.: $S = \{1, 1, 2, 3, 5, ….. \}$
  - $S_n = S_{n-1} + S_{n-2}$, $\mathbf{n \geq 3}$ **with** $\mathbf{S_1 = S_2 = 1}$ (This is Fibonacci Sequence)

Linear Recurrence Relation with Constant Time Coefficient

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots\ldots.. + C_ka_{n-k} = f(n)$]
  where C_0 C_1 ,C_2 , C_k are constants.
- E.g.:
  - $2a_n + 3a_{n-1} = 3$
  - $a_n - 7a_{n-1} + 12a_{n-2} = n.4^n$
- Order: It is the difference between the highest and lowest subscript of a.
- Degree: It is defined as the highest power of a_n.

Linear Recurrence Relation with Constant Time Coefficient

- Iteration Method
- Method of characteristic roots
- Generating functions

Linear Recurrence Relation with Constant Time Coefficient

- The method of characteristic roots with constant coefficient can solve both
    - Homogeneous Recurrence Relation [$f(n) = 0$]
    - Non-homogeneous Recurrence Relation [$f(n) != 0$]

Homogeneous Recurrence Relation

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots\ldots. + C_ka_{n-k} = 0$] where C_0 C_1 ,C_2 , C_k are constants.
- Put $a_n = r_n$, $a_{n-1} = r_{n-1}$ and so on.
- Find an equation in terms of r. This is called as characteristic equation or auxiliary equation.
- Solve characteristic equation and find characteristic roots.
- If characteristic roots are:
  - $r = r_1, r_2, r_3$ (distinct), then general solution is $a_n = b_1r_1^n + b_2r_2^n + b_3r_3^n$ where b_1, b_2, b_3 are constants.
  - $r = r_1, r_1$ then $a_n = (b_1 + nb_2)\, r_1^n$
  - $r = r_1, r_1, r_2, r_3$ then $a_n = (b_1 + nb_2)r_1^n + b_3r_2^n + b_4r_3^n$
  - $r = r_1, r_1, r_1, r_2$ then $a_n = (b_1 + nb_2 + n^2b_3)r_1^n + b_4r_2^n$

Homogeneous Recurrence Relation

- Solve $a_n = a_{n-1} + 2a_{n-2}$, $n \geq 2$ with the initial conditions $a_0 = 0$, $a_1 = 1$.
- Solve $a_n = 4(a_{n-1} - a_{n-2})$, $n \geq 2$ with the initial conditions $a_0 = a_1 = 1$.
- Solve $a_n = 8a_{n-1} - 21a_{n-2} + 18a_{n-3}$
- Solve $a_n = -a_{n-1} + 4a_{n-2} + 4a_{n-3}$, with the initial conditions $a_0 = 8$, $a_1 = 6$, $a_2 = 26$.

Case I: Non-Homogeneous Recurrence Relation with Const. Coeff.

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots\ldots\ldots + C_ka_{n-k} = f(n)$] where $f(n) != 0$.
- General solution: $a_n = a_n^h + a_n^p$ where h is for homogeneous solution and p is for particular solution.
- Case 1: Equate equation to zero and find homogeneous solution
- Case 2:
  - Let $a_n = A$
  - Put $a_n = a_{n-1} = \ldots\ldots\ldots\ldots = A$ in the given recurrence relation
  - Find A
  - $a_n^{(p)} = A$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- Solve $a_{n+2} = 5a_{n+1} - 6a_n + 2$, with the initial conditions $a_0 = 1$, $a_1 = -1$.
- Solve $a_n = 6a_{n-1} - 8a_{n-2} + 3$.
- Solve $a_n = 5a_{n-1} - 6a_{n-2} + 1$

Case II: Non-Homogeneous Recurrence Relation with Const. Coeff.

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots\ldots\ldots + C_ka_{n-k} = f(n)$] where $f(n) != 0$ and $f(n)$ is a polynomial of degree s.
- General solution: $a_n = a_n^h + a_n^p$ where h is for homogeneous solution and p is for particular solution.
- Case 1: Equate equation to zero and find homogeneous solution
- Case 2:
    - Let $a_n = A_0 + A_1n + A_2n^2 + \ldots\ldots + A_sn^s$
    - Put a_n , a_{n-1} and so on in the given recurrence relation
    - Compare the coefficients of like power on n.
    - Find $A_0, A_1, A_2, \ldots. A_s$
    - $A_n^{(p)} = A_0 + A_1n + A_2n^2 + \ldots\ldots\ldots\ldots + A_sn^s$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- Solve $y_{n+2} - y_{n+1} - 2y_n = n^2$
- For $y_n^{(h)}$: $y_{n+2} – y_{n+1} – 2y_n = 0$
- Put $y_n = r_n$. Then characteristics equation can be written as:
  - $r^2 – r – 2 = 0$
  - Solving we get $r = -1, 2$
  - $y_n^{(h)} = b_1 (-1)^n + b_2 (2)^n$

To find $y_n^{(p)}$:

put $y_n = A_0 + A_1n + A_2n^2$

$\{A_0 + A_1(n+2) + A_2(n+2)^2 \} - \{ A_0 + A_1(n+1) + A_2(n+1)^2 \} - 2\{A_0 + A_1n + A_2n^2 \} = n^2$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- To find $y_n^{(p)}$:

put $y_n = A_0 + A_1 n + A_2 n^2$

$\{A_0 + A_1(n+2) + A_2(n+2)^2\} - \{A_0 + A_1(n+1) + A_2(n+1)^2\} - 2\{A_0 + A_1 n + A_2 n^2\} = n^2$

$\{A_1 - 2A_0 + 3A_2\} + \{2A_2 - 2A_1\}n + \{-2A_2\}n^2 = n^2$

Comparing coefficients,

$A_1 - 2A_0 + 3A_2 = 0$

$2A_2 - 2A_1 = 0$

$-2A_2 = 1$

Solving $A_0 = -1$, $A_1 = -½$ and $A_2 = -1/2$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- Solve $a_{n+2} + 2a_{n+1} - 15a_n = 6n + 10$, with the initial conditions $a_0 = 1$, $a_1 = -1/2$.
- Solve $a_{n+2} - 5a_{n+1} + 6a_n = n^2$
- Solve $a_n - 2a_{n-1} = 6n$, with $a_1 = 2$

Case III: Non-Homogeneous Recurrence Relation with Const. Coeff.

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \dots\dots + C_ka_{n-k} = f(n)$] where f(n) != 0 and $f(n) = \beta^nP(n)$ where β is not a characteristics root and P(n) is a polynomial.
- General solution: $a_n = a_n^h + a_n^p$ where h is for homogeneous solution and p is for particular solution.
- Case 1: Equate equation to zero and find homogeneous solution
- Case 2:
  - Let $a_n = \beta^n (A_0 + A_1n + A_2n^2 + \dots\dots + A_sn^s)$
  - Put a_n , a_{n-1} and so on in the given recurrence relation
  - Compare the coefficients of like power on n.
  - Find $A_0, A_1, A_2, \dots. A_s$
  - $a_n^{(p)} = \beta^n (A_0 + A_1n + A_2n^2 + \dots\dots + A_sn^s)$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- Solve $a_n - 7a_{n-1} + 12a_{n-2} = 2^n$,
- Solve $a_n - 7a_{n-1} + 12a_{n-2} = n.2^n$,

Case IV: Non-Homogeneous Recurrence Relation with Const. Coeff.

- Equation: [$C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots\ldots.. + C_ka_{n-k} = f(n)$] where $f(n) != 0$ and $f(n) = \beta^n P(n)$ where β is characteristics root of multiplicity t and P(n) is a polynomial.
- General solution: $a_n = a_n^h + a_n^p$ where h is for homogeneous solution and p is for particular solution.
- Case 1: Equate equation to zero and find homogeneous solution
- Case 2:
    - Let $a_n = n^t\beta^n (A_0 + A_1n + A_2n^2 + \ldots\ldots + A_sn^s)$
    - Put a_n , a_{n-1} and so on in the given recurrence relation
    - Compare the coefficients of like power on n.
    - Find $A_0, A_1, A_2, \ldots. A_s$
  - $a_n^{(p)} = n^t\beta^n (A_0 + A_1n + A_2n^2 + \ldots\ldots + A_sn^s)$

Non-Homogeneous Recurrence Relation with Const. Coeff.

- Solve $a_n - 7a_{n-1} + 12a_{n-2} = 4^n$,
- Solve $a_n - 7a_{n-1} + 12a_{n-2} = n.4^n$,

Solving Recurrences

- Establishing a recurrence relation from a program involves identifying the relationship between different instances of the problem size and how the solution to one instance relates to the solution of a smaller instance.
- Let's go through an example to illustrate this process: Consider a simple recursive function for calculating the *n-th* Fibonacci number.

```
def fibonacci(n):
  if n <= 1:
    return n
  else:
    return fibonacci(n-1) + fibonacci(n-2)
```

Steps to Establish the Recurrence Relation

- **Identify the base cases:** Determine the simplest cases where the function does not call itself.

```
if n <= 1:
    return n
```

- This means $F(0) = 0$ and $F(1) = 1$
- **Identify the recursive case(s):** Look at the part of the function where it calls itself with smaller inputs.

```
else:
    return fibonacci (n-1) + fibonacci (n-2)
```

- This translates to the recurrence relation: $F(n) = F(n-1) + F(n-2)$ *for* $n > 1$
- Combine the base case and the recursive case to form the complete recurrence relation.

Complete Recurrence Relation for Fibonacci

- Combining the above steps, the recurrence relation for the Fibonacci sequence is:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

Another Example: Factorial

- Consider a recursive function to calculate the factorial of n:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

- Steps to Establish the recurrence relation for factorial
    - Base Case

      if n == 0:

      return 1

  This means ***F(0) = 1***

Another Example: Factorial

- Recursive case:

    else:

    *return n * factorial(n-1)*

  This translates to F(n) = n . F(n-1) for n > 0

- Steps to Establish the recurrence relation for factorial
    - Base Case:

      if n == 0:

      return 1

  This means ***F(0) = 1***

Complete Recurrence Relation for Factorial

- Combining the above steps, the recurrence relation for the Factorial is:

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \,.\, F(n-1) & \text{if } n > 0 \end{cases}$$

Example: Binary Search

- Consider a recursive function to calculate the factorial of n:

```
def binary_search(arr, target, low, high):
    if low > high:
        return  -1
    mid = (low + high) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] > target:
        return binary_search(arr, target, low, mid - 1)
    else:
        return binary_search(arr, target, mid + 1, high)
```

Example: Binary Search

- Steps to Establish the recurrence relation for factorial
- Define the problem size: The size of the problem is the range [low, high]
  - Base Case:

    if low > high:

    return -1

  This means ***T(0) = constant time = O(1)***
- **Recursive case:** elif and else in above program. Since Each recursive call reduces the problem size by half and there are other constant time factor involved, we have
  - ***T(n) = T(n/2) + O(1)***
- Complete Recurrence Relation:

$$F(n) = \begin{cases} O(1) & \text{if } n = 0 \\ T(n/2) + O(1) & \text{if } n > 0 \end{cases}$$

Solution Recurrence Relation for:

- Merge Sort
- Tower of Hanoi
- Quick Sort

Find Recurrence Relation for:

- Merge Sort

$$F(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ 2T(n/2) + O(1) & \text{if } n > 1 \end{cases}$$

- Tower of Hanoi

$$F(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n - 1) + O(1) & \text{if } n > 1 \end{cases}$$

- Quick Sort

$$F(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

General Steps to Derive Recurrence Relations

1. **Define the problem size:** Determine what n represents (e.g., the size of the input).
2. **Identify base case(s):** Find the simplest cases where the function terminates without recursion.
3. **Identify recursive case(s):** Determine how the function reduces the problem size and combines the results of smaller subproblems.
4. **Formulate the recurrence:** Write the mathematical expression that represents the function's behavior in terms of smaller problem sizes.

Methods to Solve Recurrence Relations

1. **Substitution Method or Backward Substitution Method**
2. **Recursion Tree Method**
3. **Master Method**

Substitution Method

$$F(n) = \begin{cases} 1 & \text{otherwise} \\ 3T(n - 1) & \text{if } n > 0 \end{cases}$$

- Solution

T(n) = 3T(n-1) -------------------- (1)

Put, n -> n-1

T(n-1) = 3T(n-2)

Put, n -> n-2

T(n-2) = 3T(n-3)

Substitute,

$T(n) = 3T(n-1) = 3 \{ 3T(n-2) \} = 3^2T(n-2) = 3^2\{3T(n-3)\} = 3^3T(n-3) \ldots = 3^kT(n-k)$

Assume, n - k = 0, then k = n. So, $T(n) = 3^kT(n-k) = 3^nT(0) = 3^n$

Complexity, $T(n) = O(3^n)$

Substitution Method

$$F(n) = \begin{cases} 1 & \text{otherwise} \\ 2T(n-1) - 1 & \text{if } n > 0 \end{cases}$$

T(n) = 2T(n-1) - 1 ------------------- (1)

Put, n -> n-1

T(n-1) = 2T(n-2) - 1

Put, n -> n-2

T(n-2) = 2T(n-3) - 1

Substitute,

$T(n) = 2T(n-1) -1 = 2 \{ 2T(n-2) -1 \} -1 = 2^2T(n-2) -2 -1$

$= 2^2\{2T(n-3) - 1\} -2 -1 = 2^3T(n-3) - 2^2 - 2 -1$

$= 2^KT(n-k) - 2^{k-1} - 2^{k-2} \ldots\ldots - 2^2 - 2 - 1$

Assume, n - k = 0, then k = n. So, $T(n) = 2^nT(0) - 2^{n-1} - 2^{n-2} \ldots\ldots -2^2 - 2 -1= 2^n - 2^{n-1}\ldots -2 -2^0$

$= 2n - (2^0 + 2^1 + \ldots\ldots + 2^{n-2} + 2^{n-1}) = 1$ So, Complexity, T(n) = O(1)

Substitution Method

$$F(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$$

Complexity, $T(n) = O(n^2)$

Tree Method

$$F(n) = \begin{cases} 1 & \text{if n =1} \\ 2T(n / 2) + n & \text{if } n > 1 \end{cases}$$

Complexity, $T(n) = O(n^2)$

Master Method

$T(n) = aT(n/b) + f(n)$,

where,

n = size of input

a = number of subproblems in the recursion

n/b = size of each subproblem. All subproblems are assumed to have the same size.

$f(n)$ = cost of the work done outside the recursive call, which includes the cost of dividing the problem and cost of merging the solutions

Here, $a \geq 1$ and $b > 1$ are constants, and $f(n)$ is an asymptotically positive function.

Master Theorem Method

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Master Method

- If the cost of solving the sub-problems at each level increases by a certain factor, the value of ***f(n)*** will become polynomially smaller than $\mathbf{n^{\log_b a}}$. Thus, the time complexity is oppressed by the cost of the last level ie. $\mathbf{n^{\log_b a}}$
- If the cost of solving the sub-problem at each level is nearly equal, then the value of f(n) will be $\mathbf{n^{\log_b a}}$**.** Thus, the time complexity will be f(n) times the total number of levels ie. $\mathbf{n^{\log_b a} * \log n}$
- If the cost of solving the subproblems at each level decreases by a certain factor, the value of **f(n)** will become polynomially larger than $\mathbf{n^{\log_b a}}$. Thus, the time complexity is oppressed by the cost of **f(n)**.

Master Theorem Method

Example 1: $T(n) = 9T(n/3) + n$. Here $a = 9$, $b = 3$, $f(n) = n$, and $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$ for $\epsilon = 1$, case 1 of the master theorem applies, and the solution is $T(n) = \Theta(n^2)$.

Example 2: $T(n) = T(2n/3) + 1$. Here $a = 1$, $b = 3/2$, $f(n) = 1$, and $n^{\log_b a} = n^0 = 1$. Since $f(n) = \Theta(n^{\log_b a})$, case 2 of the master theorem applies, so the solution is $T(n) = \Theta(\log n)$.

Master Theorem Method

$T(n) = 3T(n/2) + n^2$

Here,

$a = 3$

$n/b = n/2$

$f(n) = n^2$

$\log_b a = \log_2 3 \approx 1.58 < 2$

ie. $f(n) < n^{\log_b a + \epsilon}$, where, ϵ is a constant.

Case 3 implies here if $a f(n/b) \leq cf(n)$ and $c \leq 1$.

$3 (n^2 / 4) \leq c n^2$

Hence, $3/4 \leq 1$. Case 3 applies.

Thus, $T(n) = f(n) = \Theta(n^2)$

Thank you