Analysis and Design of Algorithms

UNIT-1

Recurrence Relations

Content

- Recurrence Relation
- Forming Recurrence Relation
- Solving Recurrence Relations
  - Iterative Method
  - Substitution Method
  - Recursion Tree Method
  - Master's Method

Definition

- A recurrence relation, $T(n)$, is a recursive function of integer variable n.
- Like all recursive functions, it has both recursive case and base case.
- Example:

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

- The portion of the definition that does not contain T is called the **base case** of the recurrence relation; the portion that contains T is called the **recurrent or recursive case**.
- Recurrence relations are useful for expressing the running times (i.e., the number of basic operations executed) of recursive algorithms

Recurrence Examples

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Examples of recurrence relations

- Example-1:
  - Initial condition $a_0 = 1$ (BASE CASE)
  - Recursive formula: $a_n = 1 + 2a_{n-1}$ for $n \geq 2$
  - First few terms are: 1, 3, 7, 15, 31, 63, …
- Example-2:
  - Initial conditions $a_0 = 1$, $a_1 = 2$ (BASE CASE)
  - Recursive formula: $a_n = 3(a_{n-1} + a_{n-2})$ for $n \geq 2$
  - First few terms are: 1, 2, 9, 33, 126, 477, 1809, 6858, 26001,…

Example-3: Fibonacci sequence

- Initial conditions: (BASE CASE)
  - $f_1 = 1, f_2 = 2$
- Recursive formula:
  - $f_{n+1} = f_{n-1} + f_n$ for $n \geq 3$
- First few terms:

n	1	2	3	4	5	6	7	8	9	10	11
f_n	1	2	3	5	8	13	21	34	55	89	144

Example-4: Compound interest

- Given
  - P = initial amount (principal)
  - n = number of years
  - r = annual interest rate
  - A = amount of money at the end of n years

At the end of:

- 1 year: $A = P + rP \quad = P(1+r)$
- 2 years: $A = P + rP(1+r) \quad = P(1+r)^2$
- 3 years: $A = P + rP(1+r)^2 \quad = P(1+r)^3$

...

- Obtain the formula $A = P(1 + r)^n$

Recurrence Relations: Terms

- Recurrence relations have two parts:
  - recursive terms and
  - non-recursive terms

$$T(n) = 2T(n-2) + n^2 - 10$$

- Recursive terms come from when an algorithms calls itself
- Non-recursive terms correspond to the non-recursive cost of the algorithm: work the algorithm performs within a function
- First, we need to know how to <u>solve</u> recurrences.

Forming Recurrence Relation

- For a given recursive method, the base case and the recursive case of its recurrence relation correspond directly to the base case and the recursive case of the method.
- Example 1: Write the recurrence relation for the following method.

```
void f(int n) {
   if (n > 0) {
       cout<<n;
       f(n-1);
   } }
```

- The base case is reached when n == 0. The method performs one comparison. Thus, the number of operations when n == 0, T(0), is some constant a.
- When n > 0, the method performs two basic operations and then calls itself, using ONE recursive call, with a parameter n – 1.
- Therefore the recurrence relation is:

$T(0) = a$ where a is constant
$T(n) = b + T(n-1)$ where b is constant, $n>0$

Forming Recurrence Relation

- Example 2: Write the recurrence relation for the following method.

```
 int g(int n) {
   if (n == 1)
      return 2;
   else
      return 3 * g(n / 2) + g( n / 2) + 5;
}
```

- The base case is reached when n == 1. The method performs one comparison and one return statement. Therefore, T(1), is constant **c**.
- When n>1, the method performs TWO recursive calls, each with the parameter n/2, and some constant # of basic operations.
- Hence, the recurrence relation is:

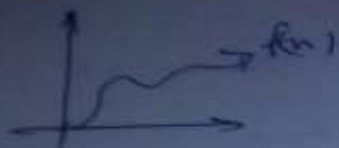
$$T(1) = c \qquad \text{for some constant } c$$
$$T(n) = b + 2T(n/2) \qquad \text{for a constant } b$$

Solving Recurrence Relations

- Iteration method
- Substitution method
- Recursion Tree
- Master method

Iteration method

Analysis of Iterative programming (Time Complexity) →



- f(n) increases for Algo The Program Time increases
- Approximation Technique
- Not Actual or Accurate (Real Time)

How to find Approximate time ?

Algo

Iterative	Recursive
fn c ( ) // function.	A(n)
{	{
single loop formula: for i 1 to n	If (——)
max (a, n); // No function calling itself so Iterative	A(n/2); // Recursion due to calling himself Again.
}	}

- Any Program written in Iteration can be converted into Recursion. & Recursion to Iteration method or vice versa. So both of these technique are just equivalent in manner.
- Both of the technique are power or workwise same but Analysiswise both are Different.

Ex: Analysis → (1) for i=1 to n → n

(2) A(n) { --- A(n/2) } → n & $\frac{n}{2}$ Analysiswise

- If any program does not contain Iteration or Rec
That means It has a "Constant time", means No dependency on running time on Input size.
whatever the input size the running time is constant. say $O(1) \to$ Constant time.

Examples $\to$ for Approximation of time.

(1.) A ( ) { int i ;
for (i = 1 to n) // loop n times Runs
printf (" Ravi "); $\to$ // n time Ravi prints.
}

So Complexity $\to$ $O(n)$.

(2) A ( ) { int i, j ;
for (i = 1 to n) // n times
{ for (j = 1 to n) // n times.
printf (" Ravi ") // n^2 times.

Complexity $\to$ $O(n^2)$

1. Iteration Method

Step-1: Expand the Recurrence.

Step-2: Express the expansion as a summation,

by plugging the Recurrence back into itself,

until you see a Pattern.

(Use algebra to express as a summation)

Step-3: Evaluate the summation.

- Also known as "Try back substituting until you know what is going on".

Solving Recurrence Relations - Iteration method

- In evaluating the summation one or more of the following summation formulae may be used:
- Arithmetic series:

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

- Geometric Series:

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$

$$\sum_{k=0}^{n-1} x^k = \frac{x^n - 1}{x - 1} (x \neq 1)$$

- Special Cases of Geometric Series:

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \qquad \text{if } x < 1$$

Solving Recurrence Relations - Iteration method

- Harmonic Series:

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$

- Others:

$$\sum_{k=1}^{n} \lg k \approx n \lg n$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=0}^{n-1} c = cn.$$

$$\sum_{k=0}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$$

$$\sum_{k=0}^{n-1} \frac{1}{2^k} = 2 - \frac{1}{2^{n-1}}$$

Example-1

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

s(n) = c + s(n-1)

c + c + s(n-2)

2c + s(n-2)

2c + c + s(n-3)

3c + s(n-3)

…

kc + s(n-k) = ck + s(n-k)

Example-1

- So far for $n >= k$ we have

  $s(n) = ck + s(n-k)$

- What if $k = n$?

  $s(n) = cn + s(0) = cn$

Example-1

- So far for n >= k we have

  s(n) = ck + s(n-k)

- What if k = n?

  s(n) = cn + s(0) = cn

- So

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- Thus in general

  s(n) = cn

$$s(n)=\begin{cases}0 & n=0\\ n+s(n-1) & n>0\end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$= \sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

- Thus in general

$$s(n) \quad = \quad n\frac{n+1}{2}$$

Example-2

- **Solve $T(n) = 2T(n/2) + n$.**

Solution: **Assume $n = 2^k$** (so $k = \log n$).

$$
\begin{aligned}
\mathbf{T(n)} &\mathbf{= 2T(n/2) + n} \\
&= 2\,\mathbf{(\,2T(n/2^2) + n/2\,)} + n && T(n/2) = 2T(n/2^2) + n/2 \\
&= 2^2\,\boldsymbol{T(n/2^2)} + 2n \\
&= 2^2\,\mathbf{(\,2T(n/2^3) + n/2^2\,)} + 2n && T(n/2^2) = 2T(n/2^3) + n/2^2 \\
&= 2^3T(n/2^3) + 3n \\
&= \ldots \\
&= \boldsymbol{2^kT(n/2^k) + k\,n} \\
&= n\,T(1) + n \log n \\
&= \Theta(n \log n)
\end{aligned}
$$