UNIT 4 SOLVING RECURRENCES

Structure **Page Nos.**

4.0 INTRODUCTION

Complexity analysis of iteration algorithms is much easier as compared to recursive algorithm but once the recurrence relation/equation is defined for recursive algorithm, which is not difficult task it become easier task to obtain for asymptotic bounds (θ, O(big oh)) for the recursive solution. In this unit we discuss three techniques for solving recurrence form methods three form techniques for solving recurrence relation. These are: (i) Substitution method (ii) Recursion Tree Method and (iii) Master Method. In the substitution method, we first guess an asymptotic bound and then we prove our guess is correct or not. In the recursion tree method a recurrences equation is converted into a recursion tree comprising several levels. Calculating time complexity requires taking a total sum of cost of all the levels. The master method requires memorization of three different types of **cases** which help to obtain asymptotic bounds of many simple recurrence relations.
The structure of the unit is done as follows.

4.1 OBJECTIVES

After going through this unit you will be able to:

- Define recurrence relation
- Construct recurrence relations of simple recursive algorithms
- List the techniques used to solve recurrence relation

Solve the recurrence relation through Substitution, Recurrence tree & Master methods

4.2 RECURRENCE RELATION

Like all recursive functions, a recurrence also consists of two steps:

Hence a recurrence has one or more initial conditions and a recursive formula, known as ***recurrence relation***.

For example: A Fibonacci sequence $f_0, f_1, f_2, \cdots$ can be defined by the recurrencerelation

$$f_n = \begin{cases} 0 & if\ n = 0 \\ 1 & if\ n = 1 \\ f_{n-1} + f_{n-2} & if\ n \geq 2 \end{cases}$$

Solving Recurrences

1. (**BasicStep**)Thegivenrecurrencesaysthatifn=0then$f_0 = 0$and if n=1 then $f_1 = 1$. These two conditions (or values) where recursion does not call itself is called a**initial conditions** (or **Baseconditions**).
2. (**Recursive step**): This step is used to findnewterms $f_2, f_3, \cdots,$ from the existing (preceding) terms, by using theformula $f_n = f_{n-1} + f_{n-2}$ for n≥ 2

   This formula says that "by adding two previous sequence (or term) we can get the next term".

   Forexample$f_2 = f_1 + f_0 = 1 + 0 = 1;$

Let us consider some recursive algorithms and try to write their recurrence relation. Then later we will learn some method to solve these recurrence relations to analyze the running time of these algorithms.

Example 1: Consider a Factorial function, defined as:

/* Algorithm for computing n!
Input:$n \in N$
Output:$n!$

Algorithm: fact(n)

1: $if\ n = 0\ then$
2: $return\ 1$
3: $else$
4: $return\ n * fact(n-1)$
5: $end\ if$

$f_3 = f_2 + f_1 = 1 + 1 = 2;\ \ f_4 = f_3 + f_2 = 2 + 1 = 3$ **and so on**

Let us try to understand the efficiency of the algorithm in terms of the number of multiplication operations required for each value of n

Let $T(n)$ denoted the number of multiplication required to execute the n!.,

that is $T(n)$ denotes the number of times the line 4 is executed in factorial algorithm.

- We have the initial condition T(0) = 1;when n = 0, the fact simply returns the number of multiplication is1.

- When $n > 1$, the line 4 performs 1 multiplication plus fact is recursively called with input (n -1). It means, by the definitionof $T(n)$, additional $T(n-1)$ number of multiplications arerequired.

Thus we can write a recurrence relation for the algorithm fact as:

$T(1) = 0$ (**base case**)

$T(n) = 1 + T(n-1)$ (**Recursive step**)

(We can also write some constant value instead of writing 0 and 1, that is

$$T(n) = \begin{cases} b & \text{if } n = 1 \quad (\textbf{base case}) \\ c + T(n-1) & (\textbf{Recursive step}) \end{cases}$$

Where b and c are constants

The following algorithm calculates x^n $(i.e. exponentiation)$.

Algorithm3: $\boldsymbol{Power\ (x, n)}$

1: **if**$(n == 0)$

2: $return\ 1$

3: ***else***

4: $return\ x * Power(x, n-1);$

5: $endif$

- The base case isreachedwhenn= = 0

The algorithm 3 performs one comparison and one returnstatement. Therefore,

When n > 1; the **algorithm 3** performs one recursive call with inputparameter(n – 1) at line 4, and some constantnumber of basic operations. Thus we obtain the recurrence relationas:

$$T(n) = \begin{cases} T(1) = a & (\boldsymbol{base\ case}) \\ T(n-1) + b & (\boldsymbol{Recursive\ step}) \end{cases}$$

Example 4: A customer makes an investment of Rs. 5000 at 15% annual compound interest. If T_n denotes the amount earned at the end of n years, define a recurrent relation and initial conditions.

At the end of n-1 years, the amount is T_{n-1}. After one more year , the amount will be T_{n-1}+ the interest amount.

Therefore $T_n = T_{n-1} + (15\%)T_{n-1} = (0.15)T_{n-1} = (1.15)T_{n-1}$; n≥ 1

To find out the recurrence relation when n=1 (base value) we have to find the value of T_0. Since T_0 refers to the initial amount, i.e.,5000 .With the above definitions we can calculate the value of T_n for any value of n. For example: $T3 = (1.15)T_2 = (1.15)(1.15)T_1 = (1.15)(1.15)(1.15)T_0 = (1.15)^3(5000)$ The above computation can be extended to any arbitrary value of n.

$T_n = (1.15)T_{n-1}$

.. ..

$= ((1.15)^n(5000))$

4.3 METHODS FOR SOLVING RECURRENCE RELTIONS

Three methods are discussed here to solve recurrence relations:

1. The Substitution Method
   The Recursion-tree method
2. Master method

We start with substitution method

4.3.1 SUBSTITUTIONMETHOD

A substitution method is one, in which we guess a bound and then use mathematical induction to prove our guess correct. It is basically two step process:

Step1: Guess the form of the Solution.

Step2: Prove your guess is correct by using Mathematical Induction.

Example 1. Solve the following recurrence by using substitution method.

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Solution*: step1:* The given recurrence is quite similar with that of MERGE- SORT, you guess the solutionis $T(n) = O(nlogn)$

Or $T(n) \leq c.nlogn$

Step2: Now we use mathematical Induction.

Here our guess does not hold for n=1because $T(1) \leq c.1log1$

i.e., T(1) <= 0 which is in contradiction with T(1) =1
The reason is that O(n log n) means the upper bound holds for n >= n_0 and n_0 could be 2

Now for n=2

$$T(2) \leq c.2log2$$

$$2T\left(\frac{2}{2}\right) + 2 \leq c.2$$

$$2T(1) + 2 \leq c.2$$

$$0 + 2 \leq c.2$$

2 ≤ c.2 which is true for c=1

w So$T(2) \leq c.nlogn$is True forn=2

(i) **Induction step:** Now assume it is true forn=n/2

$$i.e. T\left(\frac{n}{2}\right) \leq c.\left(\frac{n}{2}\right)\log\left(\frac{n}{2}\right) \; is \; true.$$

Now we have to show that it is true for $n = n$
$\lfloor \cdot \rfloor$
$i.e. T(n) \leq c.nlogn$

We known that $T(n) \leq 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$

$\leq c \text{ nlog } \lfloor \frac{n}{2} \rfloor + n \leq c \text{ nlogn } + cnlog2 + n$

$\leq cnlogn - cn + n$

$\leq cnlogn \quad \forall c \geq 1$

Thus $T(n) = O(nlogn)$

Remark: Making a good guess, which can be a solution of a given recurrence, requires experiences. So, in general, we are often not using this method to get a solution of the given recurrence.

4.3.2 RECURSION TREEMETHOD

A recursion tree is a convenient way to visualize what happens when a recurrence is iterated. It is a pictorial representation of a given recurrence relation, which shows how Recurrence is divided till Boundaryconditions.

Recursion tree method is especially used to solve a recurrence of theform:

$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad \ldots\ldots\ldots (1) \quad where \ a > 1, b \geq 1$

$\leq 2\left(c\left\lfloor \frac{n}{2} \right\rfloor log\left\lfloor \frac{n}{2} \right\rfloor\right) + n$

This recurrence (1) describe the running time of any divide-and-conquer algorithm.
Method (steps) for solving a recurrence $\boldsymbol{T(n)} = \boldsymbol{aT(\frac{n}{b})} + \boldsymbol{f(n)}$ using recursion tree:

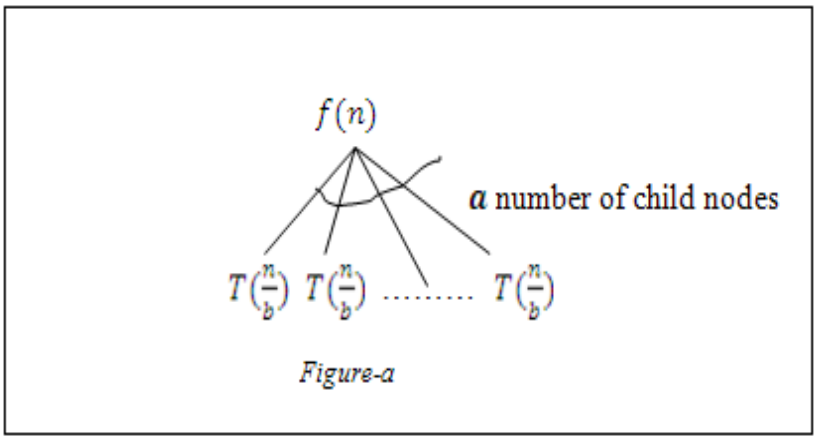

Figure-a

We make a recursion tree for a given recurrence as follows: a) To make a recursion tree of a given recurrence (1), First put the value of $\boldsymbol{f(n)}$ at root node of a tree and make a number of **child nodes** of this root value f(n). Now the tree will look like as:

[copy from the pdf file all sections from pdf file before summary. Use summary and solutions sections from this word fille which are modified]

4.4 SUMMARY

When an algorithm contains a recursive call to itself, its running time can often be described by a recurrence equation which describes a function in terms of its value on smaller inputs.

There are three basic methods of solving the recurrence relation:

1. The SubstitutionMethod
2. The Recursion-treeMethod
3. The MasterTheorem

Master method provides a "cookbook" method for solving recurrences of the form:$T(n) = aT\left(\frac{n}{b}\right) + f(n) where\ a\ \geq 1\ and\ b > 1$ are $f(n)$ constants

In master method you have to always compare the value of$f(n)$ with $n^{log_b a}$ to decide which case is applicable.

4.5 SOLUTIONS/ANSWERS

Copy from pdf file. No change was suggested by the editor