3. Recurrence Relation

Dr. Swati Agarwal

Agenda

1. Types of Algorithm
2. Recurrence Relation
3. Solving Recurrence
   - Substitution Method
   - Recursion-Tree Method
   - Master's Theorem Method

Types of Algorithm

- Iterative
  - Function with a loop
  - Uses frequency count method to analyze time function
- **Recursive**
  - **Function calling itself**
  - **Uses recursive equations to analyze time function**
- Not recursive or iterative
  - No dependency of running time on input size.
  - Time will be constant

Recurrence Relation

- An equation that defines function in terms of its values on smaller input
- An equation which is defined in terms of itself (breaking into smaller inputs)

Example: Recurrence Relation

- **function** $\mathrm{A}(n)$
  if condition **then**
  return $\mathrm{A}(\mathsf{n}/2) + \mathrm{A}(\mathsf{n}/2)$

  Time to analyze the algorithm
  $T(n) = c + 2T(n/2)$

- **function** $\mathrm{A}(n)$
  if $n > 1$ **then**
  return $\mathrm{A}(\mathsf{n}\text{-}1)$

  Time to analyze the algorithm
  $T(n) = c + T(n-1)$

Solving Recurrence

1. **Substitution Method**

2. **Recursion-Tree Method**

3. **Master's Theorem Method**

Solving Recurrence

1. **Substitution Method**
   - Guess a bound (or the behavior of the function)
   - Use mathematical induction method to prove the guess correct.

2. **Recursion-Tree Method**

3. **Master's Theorem Method**

Solving Recurrence

1. **Substitution Method**
   - Guess a bound (or the behavior of the function)
   - Use mathematical induction method to prove the guess correct.
2. **Recursion-Tree Method**
   - Convert the recurrence equation into a tree where nodes represent the cost incurred at various levels of recursion
   - Summation of all the costs (till last level- stability condition) to solve the recurrence
3. **Master's Theorem Method**

Solving Recurrence

1. **Substitution Method**
   - Guess a bound (or the behavior of the function)
   - Use mathematical induction method to prove the guess correct.

2. **Recursion-Tree Method**
   - Convert the recurrence equation into a tree where nodes represent the cost incurred at various levels of recursion
   - Summation of all the costs (till last level- stability condition) to solve the recurrence

3. **Master's Theorem Method**
   - provides a cook-book or bounds to solve recurrence of the following form:
     $T(n) = aT(n/b) + f(n)$

Substitution Method

- Substitute the guessed answer when mathematical induction hypothesis is applied to smaller values.

- Powerful method but slow.

- It can be applied only when it is easy to guess the form of the solution/function.
  - unfortunately, there is no general way to guess the correct form/solution. It takes experience, practice and creativity.

- Function is represented as $T(n)$ instead of $f(n)$. (Time taken by the algorithm for input size n)

Example: Substitution Method

```
function A(n)
    if (n > 0) then
        print a statement          //1
        return A(n-1)
```

Time taken by the algorithm A

$$T(n) = \begin{cases} 1 + T(n-1) & \text{if } n \geq 1 \\ 1 & \text{if } n = 0 \end{cases}$$

Solving $T(n) = T(n-1) + 1$

$$T(n) = T(n-1) + 1 \tag{1}$$

Solving $T(n) = T(n-1) + 1$

$$T(n) = T(n-1) + 1 \tag{1}$$

Divide the task further.

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

Solving $T(n) = T(n-1) + 1$

$$T(n) = T(n-1) + 1 \tag{1}$$

Divide the task further.

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

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

Solving $T(n) = T(n-1) + 1$

$$T(n) = T(n-1) + 1 \tag{1}$$

Divide the task further.

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

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

Substitute Eq 2 in Eq 1

$$T(n) = T(n-2) + 2 \tag{4}$$

Solving $T(n) = T(n-1) + 1$

$$T(n) = T(n-1) + 1 \tag{1}$$

Divide the task further.

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

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

Substitute Eq 2 in Eq 1

$$T(n) = T(n-2) + 2 \tag{4}$$

Substitute Eq 3 in Eq 4

$$T(n) = T(n-3) + 3 \tag{5}$$

Solving $T(n) = T(n-1) + 1$

after some more iterations

$$T(n) = T(n-k) + k \tag{6}$$

Solving $T(n) = T(n-1) + 1$

after some more iterations

$$T(n) = T(n-k) + k \tag{6}$$

Termination/Stability condition $(n-k) = 0 \implies n = k$

Solving $T(n) = T(n-1) + 1$

after some more iterations

$$T(n) = T(n-k) + k \tag{6}$$

Termination/Stability condition $(n-k) = 0 \implies n = k$

$$T(n) = 1 + n \tag{7}$$

$T(n) = \mathcal{O}(n)$

Exercise: Substitution Method

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

$\mathcal{O}(n^2)$

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

$\mathcal{O}(n^2)$

3.
$$T(n) = \begin{cases} 2T(n/2) + c & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

$\mathcal{O}(n)$

*hint: you can ignore the constants here

Exercise: Substitution Method

4.

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

$$\mathcal{O}(n \log n)$$

5.

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

$$\mathcal{O}(\log n)$$

Recursion-Tree Method

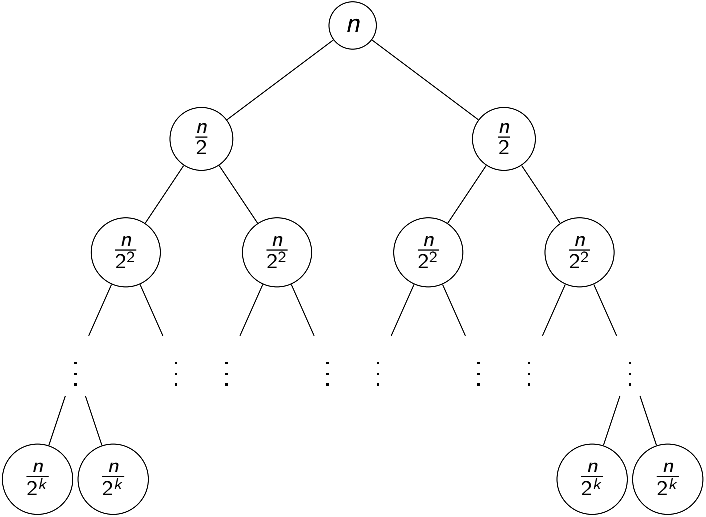
- Addresses the challenge of substitution method- guessing the correct solution
- Each node represents the amount of work/cost/time to solve single sub-problem.
- The total cost of solving the problem is the sum of cost at all the levels (sum of the cost at all sibling nodes defines the cost within a level)

Example: Recursion Tree

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

refer to class notes.

Example: Recursion Tree



Exercise: Recursion Tree

All examples used for substitution method (good for practice and comparison of solution)

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

   $\mathcal{O}(n \log n)$

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

   $\mathcal{O}(\log n)$

Master's Theorem Method

Before we study Master's Theorem, we need to become more familiar with comparison of Functions.
$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < ...2^n < 3^n < n^n$

Comparison of Functions

1. Substitute values and see the results (for very large value of input)
2. Cancel common terms. Repeat Step 1.
3. Take a log of both the functions. Repeat Step 2.

Exercise: Comparison of Functions[†]

1. $f(n) = n^2, \qquad g(n) = n^3$
2. $f(n) = n^2, \qquad g(n) = 2^n$
3. $f(n) = 3^n, \qquad g(n) = n^3$
4. $f(n) = n^2, \qquad g(n) = n \log n$
5. $f(n) = n, \qquad g(n) = (\log n)^{100}$
6. $f(n) = n^{\log n}, \qquad g(n) = n \log n$
7. $f(n) = \sqrt{\log n}, \qquad g(n) = \log \log n$
8. $f(n) = n^{\sqrt{n}}, \qquad g(n) = n^{\log n}$
9. $f_1(n) = 2^n, \quad f_2(n) = n^{3/2}, \quad f_3(n) = n \log n, \quad f_4(n) = n^{\log n}$

[†] refer to the class notes for solutions.

Exercise: Comparison of Functions

Functions behaving differently for different input size†

1.
$$f(n) = \begin{cases} n^3 & \text{if } 0 < n < 10,000 \\ n^2 & \text{if } n \geq 10,000 \end{cases}$$

$$g(n) = \begin{cases} n & \text{if } 0 < n < 100 \\ n^3 & \text{if } n > 100 \end{cases}$$

Master's Theorem Method

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

- If $f(n) = \mathcal{O}(n^{\log_b a-\epsilon})$ then
  Solution: $T(n) = \Theta(n^{\log_b a})$

- If $f(n) = \Theta(n^{\log_b a})$ then
  Solution: $T(n) = \Theta(n^{\log_b a} \log n)$

- If $f(n) = \Omega(n^{\log_b a+\epsilon})$ AND
  $af(n/b) \leq cf(n)$ then
  Solution: $T(n) = \Theta(f(n))$
  if the equation lies in case 3 but regularity condition does not meet then Master's theorem fails and we have to use some other method (substitution or recursion tree) to solve the equation.

Examples:

1. $T(n) = 9T(n/3) + n$
2. $T(n) = T(2n/3) + 1$
3. $T(n) = 2T(n/2) + n^4$
4. $T(n) = 3T(n/4) + n\log n$

check the tutorial sheet for more examples/equations on Recurrence Relation.