CS 2500: Algorithms

Lecture 6: Introduction to Recurrence Equations

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

September 5, 2024

What is a Recurrence Equation?

- **Definition:** A recurrence equation is a compact notation that defines a sequence by relating each term to one or more preceding terms.
- **Purpose:** Used to analyze recursive algorithms by expressing the running time of a problem in terms of its subproblems.

Why Recurrence Equations Matter

- Recursive algorithms often call themselves with smaller input sizes.
- To determine the algorithm's efficiency, we analyze how the work is divided and combined, which is captured by a recurrence relation.
- Example applications: Fibonacci sequence, factorial computation, dynamic programming problems.

Example: Factorial Recurrence Relation

$$n! = n \times (n-1)!$$

- **Base condition:** $0! = 1$
- This defines how $n!$ is calculated recursively.

Recurrence Equation vs. Recurrence Relation

- **Recurrence Equation:** A compact way to express a sequence based on previous terms.
- **Recurrence Relation:** A broader term that can include more complex expressions, equivalent to differential equations in discrete settings.
- **Example:**
  - **Recurrence Relation (Factorial):**

    $$n! = n \times (n-1)!$$

    This expresses the factorial as a recursive relationship, defining $n!$ in terms of $(n-1)!$.
  - **Recurrence Equation (Factorial):**

    $$t_n = t_{n-1} + 1$$

    This expresses the factorial sequence directly as a recurrence equation.

Example: Simple Recurrence Equation

Consider the recurrence equation:

$$t_n = t_{n-1} + 2$$

- **Initial Condition:** $t_0 = 0$
- This generates the sequence: 2, 4, 6, 8, ...
- Changing the initial condition alters the sequence (e.g., $a_0 = 1$ gives 1, 3, 5, 7, ...)

Importance of Initial Conditions

- Initial conditions are necessary to uniquely define the sequence.
- Changing the initial condition can completely alter the sequence generated by a recurrence equation.

Classification of Recurrence Equations

- **Linear Recurrences:** The next term is a linear combination of previous terms.
- **Non-linear Recurrences:** The next term is a non-linear combination of previous terms.

Linear Recurrences

- A linear recurrence equation for a sequence $t_0, t_1, \ldots t_n$ expresses the final term t_n as a linear combination of its previous terms in a polynomial form.
- Example: The recurrence equation of a Fibonacci series can be expressed as:

$$t_n = t_{n-1} + t_{n-2}$$

.

- In general, linear recurrences are of the form:

$$a_0 \cdot t_n + a_1 \cdot t_{n-1} + \ldots a_k \cdot t_{n-k} = f(n)$$

where k and a_i are constants, k being the order of the recurrence equation.

Types of Linear Recurrences

- **Based on order:**
  - First-order recurrence equations
  - Second-order recurrence equations
  - Higher-order recurrence equations
- **Based on homogeneity:**
  - Homogeneous recurrence equations
  - Non-homogeneous recurrence equations

Order of Linear Recurrences

- The number of preceding terms used for computing the present term of a sequence is called the **order of recurrence equations**.
- In other words, the order is the difference between the highest and the lowest subscripts of the dependent variable in a recurrence equation.
- Example: For the recurrence equation: $t_n = t_{n-1} + t_{n-2}$, the order is $n - (n-2) = 2$.

Order of Linear Recurrences

- A first-order linear recurrence equation uses one previous term:

$$t_n = c_1 t_{n-1}$$

- A second-order recurrence equation uses two previous terms:

$$t_n = c_1 t_{n-1} + c_2 t_{n-2}$$

Uniqueness of the Solution

- To uniquely define a sequence, the number of initial conditions must match the order of the recurrence equation.
- **Why?**
  - Each initial condition is required to determine the starting points for the sequence.
  - Without these starting points, the sequence isn't fully determined, leading to multiple possible sequences.

Uniqueness of the Solution: Example

- Consider the Fibonacci sequence:

$$t_n = t_{n-1} + t_{n-2}$$

- This is a second-order recurrence relation.
- To uniquely determine the sequence, we need two initial conditions:

$$t_0 = 0, \quad t_1 = 1$$

- Without these initial conditions, the sequence could start with any two numbers, leading to different sequences that all satisfy the same recurrence relation.

Types of Linear Recurrences

Homogeneous vs Non-Homogeneous Linear Recurrences

- Consider the linear recurrence equation:

$$a_0 \cdot t_n + a_1 \cdot t_{n-1} + \ldots a_k \cdot t_{n-k} = f(n)$$

- If $f(n) = 0$, it is called a **homogeneous equation**. If $f(n) \neq 0$, the equation is called **non-homogeneous**.

Homogeneity Test. To determine whether a given linear recurrence is homogeneous or non-homogeneous:

- Substitute all orders of t_n are with zero.
- Check if LHS = RHS.

Examples

- $t_n = t_{n-1} + t_{n-2}$. Substituting t_n and all its factors with zero: $0 = 0 + 0 = 0$. So homogeneous equation.
- $t_n = t_{n-1} + (n-3)$. Substituting t_n and t_{n-1} with zero yields:$0 = (n-3)$. So non-homogeneous equation.

Non-Linear Recurrences

- The non-linear recurrence equation of a sequence $\{t_0, t_1, \ldots, t_n\}$ expresses t_n as a non-linear combination of its previous terms.
- In algorithm study, a unique form of non-linear recurrence equations, called **divide-and-conquer recurrences**, is often encountered.

Non-Linear Recurrences

Divide-and-Conquer Recurrences

- The divide-and-conquer recurrence equations are of the following form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

  - a is the number of subproblems,
  - n is the size of the problem,
  - $\frac{n}{b}$ is the size of the subproblem,
  - $f(n)$ is the cost of work done for non-recursive calls.

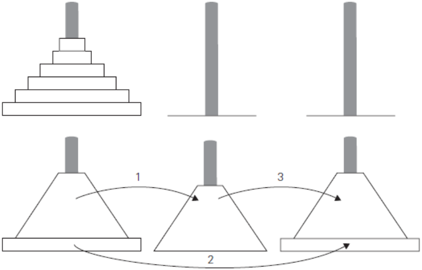
Example. The recurrence equation for merge sort is:

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

- Number of subproblems is 2 at every level.
- Problem size is reduced by a factor of 2.
- n amount of work has to be performed to combine the results.

Formulation of Recurrences: Tower of Hanoi

- Classic example of a recursive problem.
- n disks of different sizes placed on the first of three pegs.
- Objective: Move all disks from the first peg to the third peg, using the second peg as an auxiliary.
- The rules:
  - Only one disk can be moved at a time.
  - A larger disk cannot be placed on top of a smaller disk.



Formulation of Recurrences: Tower of Hanoi

Recursive Solution for Tower of Hanoi

- The problem has an elegant recursive solution.
- To move $n > 1$ disks from peg 1 to peg 3:
  1. Recursively move $n - 1$ disks from peg 1 to peg 2 (using peg 3 as auxiliary).
  2. Move the largest disk directly from peg 1 to peg 3.
  3. Recursively move $n - 1$ disks from peg 2 to peg 3 (using peg 1 as auxiliary).
- For $n = 1$, simply move the disk directly from peg 1 to peg 3.

Formulation of Recurrences: Tower of Hanoi

Recurrence Relation for Tower of Hanoi

- Let t_n represent the number of moves required to solve the puzzle with n disks.
- The recurrence equation for t_n can be derived as follows:

$$t_n = t_{n-1} + 1 + t_{n-1} \text{ for } n > 1$$

- This simplifies to:

$$t_n = 2t_{n-1} + 1 \text{ for } n > 1$$

- With the initial condition:

$$t_1 = 1$$

Formulation of Recurrences: Tower of Hanoi

Understanding the Recurrence Relation

- The relation $t_n = 2t_{n-1} + 1$ captures the recursive nature of the Tower of Hanoi problem.
- Each step involves:
  1. Moving $n - 1$ disks twice.
  2. Moving the largest disk once.
- As n increases, the number of moves required grows exponentially.

Formulation of Recurrences: Complete Graph

- Consider deriving a recurrence relation for a complete graph.
- A complete graph K_n with n vertices has an edge between every pair of vertices.
- From Table 1, we see that the sequence generated is: $0, 1, 3, 6, 10, 15, \ldots$
- This leads to the recurrence:

$$t_n = t_{n-1} + (n - 1)$$

Vertices	1	2	3	4	5	6
Edges	0	1	3	6	10	15

Table: Vertices and edges of a complete graph

Verification of the Recurrence Relation

- The sequence generated by this recurrence relation for the number of edges is:

$$0, 1, 3, 6, 10, 15, \ldots$$

- This sequence can be verified by substituting different values of n into the recurrence relation:

$$\begin{aligned}
t_1 &= t_0 + (1 - 1) = 0 + 0 = 0 \\
t_2 &= t_1 + (2 - 1) = 0 + 1 = 1 \\
t_3 &= t_2 + (3 - 1) = 1 + 2 = 3 \\
t_4 &= t_3 + (4 - 1) = 3 + 3 = 6
\end{aligned}$$

Techniques for Solving Recurrences

We are going to look at the following techniques for solving recurrence equations:

1. Guess-and-Verify method (called "method of substitution" by Cormen in his book "Introduction to Algorithms".)
2. Substitution/Iteration method
3. Recurrence-tree method
4. Difference/Telescoping method
5. Polynomial reduction method
6. Master theorem

Solving Recurrences: Guess-and-Verify

- This is one of the simplest methods for solving recurrence equations.
- The method involves two phases:
  1. **Guess**: Make an educated guess about the form of the solution.
  2. **Verify**: Use mathematical induction to verify that the guessed solution satisfies the recurrence relation.

Solving Recurrences: Guess-and-Verify

Example 1: Solve the recurrence equation $t_n = t_{n-1} + 2$ using the guess-and-verify method, with the initial condition $t_0 = 1$.
Solution:

1. **Guess**: Start by guessing the form of the solution by substituting different values of n into the recurrence equation:

$$
\begin{aligned}
t_1 &= t_0 + 2 = 1 + 2 = 3, \\
t_2 &= t_1 + 2 = 3 + 2 = 5, \\
t_3 &= t_2 + 2 = 5 + 2 = 7.
\end{aligned}
$$

   The sequence $1, 3, 5, 7, \ldots$ suggests that the solution is of the form $2n + 1$.

Solving Recurrences: Guess-and-Verify

Example 1: Solve the recurrence equation $t_n = t_{n-1} + 2$ using the guess-and-verify method, with the initial condition $t_0 = 1$.
Solution:

2. **Verify**: Use mathematical induction to verify the guessed solution:
    - Base case: $t_0 = 1 = 2(0) + 1$
    - Inductive hypothesis: Assume $t_n = 2n + 1$ for some n.
    - Inductive step: $t_{n+1} = t_n + 2 = (2n + 1) + 2 = 2(n + 1) + 1$

   Therefore, the solution is verified as $t_n = 2n + 1$.

Solving Recurrences: Guess-and-Verify

Example 2: Solve the recurrence equation $t_n = t_{n-1} + n^2$ using the guess-and-verify method, with the initial condition $t_1 = 1$.
Solution:

1. **Guess**: Start by substituting different values of n into the recurrence equation to identify a pattern:

$$\begin{aligned} t_1 &= 1 = 1^2, \\ t_2 &= t_1 + 2^2 = 1 + 4 = 5, \\ t_3 &= t_2 + 3^2 = 5 + 9 = 14, \\ t_4 &= t_3 + 4^2 = 14 + 16 = 30. \end{aligned}$$

This suggests that the solution could be

$$t_n = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Solving Recurrences: Guess-and-Verify

Example 2: Solve the recurrence equation $t_n = t_{n-1} + n^2$ using the guess-and-verify method, with the initial condition $t_1 = 1$.
Solution:

2. **Verify**: Prove using mathematical induction that

$$t_n = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

(**Homework)**

Solving Recurrences: Guess-and-Verify

Example 3: Use the guess-and-verify method to solve the following recurrence equation:

$$T(n) = 3T\left(\frac{n}{2}\right) \quad \text{with initial condition } T(1) = 1.$$

Note that $n > 1$ and $n = 2^k$.
Solution: Since n is a power of 2 (i.e., $n = 2, 4, 8, \ldots$), we can compute:

$$\begin{aligned}
T(1) &= 1, \\
T(2) &= 3T(1) = 3 \times 1 = 3, \\
T(4) &= 3T(2) = 3 \times 3 = 3^2, \\
T(8) &= 3T(4) = 3 \times 3^2 = 3^3
\end{aligned}$$

Solving Recurrences: Guess-and-Verify

Example 3: Use the guess-and-verify method to solve the following recurrence equation:

$$T(n) = 3T\left(\frac{n}{2}\right) \quad \text{with initial condition } T(1) = 1.$$

Note that $n > 1$ and $n = 2^k$.

Solution:

1. **Guess**: Every time n doubles (from 1 to 2, then 4, then 8, etc.), the value of $T(n)$ is multiplied by another factor of 3. Therefore, the power of 3 increases incrementally as n increases. From the pattern above, we observe that the result for $T(n)$ seems to be $3^{\log_2 n}$, because:
   - When $n = 1$, $\log_2 n = 0$ and $T(1) = 3^0 = 1$,
   - When $n = 2$, $\log_2 n = 1$ and $T(2) = 3^1 = 3$,
   - When $n = 4$, $\log_2 n = 2$ and $T(4) = 3^2$,
   - When $n = 8$, $\log_2 n = 3$ and $T(8) = 3^3$.

Solving Recurrences: Guess-and-Verify

Example 3: Use the guess-and-verify method to solve the following recurrence equation:

$$T(n) = 3T\left(\frac{n}{2}\right) \quad \text{with initial condition } T(1) = 1$$

Solution:

2. **Verify**: Use mathematical induction to verify the guessed solution:
   - **Basis Step**: $T(1) = 3^{\log_2 1} = 3^0 = 1$. Given $T(1) = 1$, so basis step is true.
   - **Induction Hypothesis**: Assume $T(k) = 3^{\log_2 k}$ holds for some $n = k$.
   - **Inductive Step**: Prove it holds for $n = 2k$:

     $$T(2k) = 3T(k) = 3 \times 3^{\log_2 k} = 3^{\log_2 k+1} = 3^{\log_2 k + \log_2 2} = 3^{\log_2 (2k)}$$

- Therefore, the guess is correct: $T(n) = 3^{\log_2 n}$.

Example 4: Use the guess-and-verify method to solve the following recurrence equation:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n \div 2) + n & \text{otherwise} \end{cases}$$

- Discontinuous functions such as the floor function (implicit in $n \div 2$) are hard to analyze.
- Our first step is to replace $n \div 2$ by the better behaved "$n/2$" with a suitable restriction on the set of values of n that we consider initially.

Solving Recurrences: Guess-and-Verify

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n \div 2) + n & \text{otherwise} \end{cases}$$

- It is tempting to restrict n to being even since in that case $n \div 2 = n/2$.
- BUT: Recursively dividing an even number by 2 might produce an odd number larger than 1.
  - Starting with an even n, each division by 2 reduces the size predictably until an odd number is produced.
  - For example, $n = 6$:

    $$6 \div 2 = 3 \quad \text{(odd number)}$$

  - Odd numbers introduce irregular behavior in the recurrence, leading to complexities in analysis.
  - Further division of odd numbers by 2 results in non-integer or discontinuous behavior (e.g., $3 \div 2 = 1$).
- So we restrict n to be exact powers of 2.

Tabulating Values of the Recurrence

- We tabulate the values of the recurrence for the first few powers of 2:

n	$T(n)$
1	1
2	5
4	19
8	65
16	211
32	665

- Each term is computed from the previous term, e.g., $T(16) = 3 \times T(8) + 16 = 3 \times 65 + 16 = 211$.
- No obvious pattern is visible in the sequence initially.

Finding the Pattern

n	$T(n)$
1	1
2	$3 \times 1 + 2$
2^2	$3^2 \times 1 + 3 \times 2 + 2^2$
2^3	$3^3 \times 1 + 3^2 \times 2 + 3 \times 2^2 + 2^3$
2^4	$3^4 \times 1 + 3^3 \times 2 + 3^2 \times 2^2 + 3 \times 2^3 + 2^4$
2^5	$3^5 \times 1 + 3^4 \times 2 + 3^3 \times 2^2 + 3^2 \times 2^3 + 3 \times 2^4 + 2^5$

- Keep more "history" about the value of $T(n)$. For example:

$$T(2) = 3 \times 1 + 2$$

- This allows us to see a pattern:

$$T(4) = 3 \times T(2) + 4 = 3 \times (3 \times 1 + 2) + 4 = 3^2 \times 1 + 3 \times 2 + 4$$

- Continuing this way:

$$T(2^k) = 3^k \times 1 + 3^{k-1} \times 2 + \cdots + 3 \times 2^{k-1} + 2^k$$

Step-by-Step Calculation for $T(2^k)$

Given recurrence:

$$T(2^k) = 3^k \times 2^0 + 3^{k-1} \times 2^1 + 3^{k-2} \times 2^2 + \cdots + 3^1 \times 2^{k-1} + 3^0 \times 2^k$$

- The recurrence involves summing powers of 3 multiplied by powers of 2.

Step 1: Writing the Recurrence as a Summation

$$T(2^k) = \sum_{i=0}^{k} 3^{k-i} \times 2^i$$

- This summation is written as the sum of decreasing powers of 3 and increasing powers of 2.

Step 2: Factoring Out the Common Term

We can factor out 3^k from each term:

$$T(2^k) = 3^k \sum_{i=0}^{k} \left(\frac{2}{3}\right)^i$$

- This simplifies the expression and makes it easier to recognize the next step.

Step 3: Recognizing the Geometric Series

- The summation $\sum_{i=0}^{k} \left(\frac{2}{3}\right)^i$ is a geometric series.
- The sum of the first $k+1$ terms of a geometric series is given by:

$$\sum_{i=0}^{k} r^i = \frac{1 - r^{k+1}}{1 - r}$$

where $r = \frac{2}{3}$.

Step 4: Applying the Geometric Series Formula

$$\sum_{i=0}^{k} \left(\frac{2}{3}\right)^i = \frac{1 - \left(\frac{2}{3}\right)^{k+1}}{1 - \frac{2}{3}} = 3 \times \left(1 - \left(\frac{2}{3}\right)^{k+1}\right)$$

- We now have the sum of the series in a simplified form.

Step 5: Substituting Back into $T(2^k)$

Substitute the result of the geometric series back into the equation for $T(2^k)$:

$$T(2^k) = 3^k \times \left[3 \times \left(1 - \left(\frac{2}{3}\right)^{k+1}\right)\right]$$

Step 6: Simplifying the Expression

$$T(2^k) = 3^{k+1} \times \left(1 - \left(\frac{2}{3}\right)^{k+1}\right)$$

$$T(2^k) = 3^{k+1} - 2^{k+1}$$

Homework: Use mathematical induction to prove that the solution $T(2^k) = 3^{k+1} - 2^{k+1}$ holds for all k.

Handling n When It Is Not a Power of 2

- **Problem:** Solving the recurrence exactly when n is not a power of 2 can be difficult.
- **Solution:** Use asymptotic notation and express $T(n)$ in terms of $T(2^k)$, where $k = \log_2 n$.
- Rewriting the equation:

$$T(n) = T(2^{\log_2 n}) = 3^{1+\log_2 n} - 2^{1+\log_2 n} = 3 \cdot 3^{\log_2 n} - 2 \cdot 2^{\log_2 n}$$

- **Logarithmic Simplification:** Using the property of logarithms ($a^{\log_b x} = x^{\log_b a}, \log_a a = 1$)

$$3^{\log_2 n} = n^{\log_2 3} \quad 2^{\log_2 n} = n^{\log_2 2} = n$$

- Final form of the recurrence relation:

$$T(n) = 3n^{\log_2 3} - 2n$$

Asymptotic Conclusion

- Using conditional asymptotic notation, we conclude that:

$$T(n) \in \Theta(n^{\log_2 3}) \quad \text{(when } n \text{ is a power of 2)}$$

- Since $T(n)$ is non-decreasing, and $n^{\log_2 3}$ is smooth, we conclude:

$$T(n) \in \Theta(n^{\log_2 3}) \quad \text{unconditionally.}$$

Asymptotic Conclusion

Homework: $T(n) \in \Theta(n^{\log_2 3})$ holds when $T(n)$ is a non-decreasing function. Use mathematical induction to prove that $T(n) = 3n^{\log_2 3} - 2n$ is a non-decreasing function.