Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 22

Review Lectures 1-21

Lecture No 1: Model of Computation

- Analysis independent of the variations in
  - machine, operating system, language, compiler,
- We did not consider the low-level details
- We supposed our model to be an abstraction of a standard generic single-processor machine, called a random access machine RAM, an idealized machine
  - infinitely large random-access memory,
  - instructions execute sequentially
- Every instruction is in fact a basic operation on two values in machine's memory which takes unit time.

Drawbacks in Model of Computation

- We assumed each basic operation takes constant time i.e. adding, multiplying, comparing etc. of two numbers of any length in constants time
- Addition of two numbers takes a unit time!
  - not good because numbers may be arbitrarily
- Addition and multiplication both take unit time!
  - Again very bad assumption

Finally what about Our Model?

- But with all these weaknesses, our model is not so bad because we have to give the comparison not the absolute analysis of any algorithm.

Lecture No 2: Mathematical Tools

A Sequence of Mathematical Tools

- Sets
- Sequences
- Order pairs
- Cross Product
- Relation
- Functions
- Operators over above structures

Lecture No 3: Logic and Proving Techniques

- Propositional Logic
- Predicate Logic
- Proofs using
  - Truth Tables
  - Logical Equivalences
  - Counter Example
  - Contradiction
  - Rule of Inference

Lecture No 4 & 5: Mathematical Induction

Claim: P(n) is true for all $n \in Z^+$, for $n \geq n_0$

1. Basis
   - Show formula is true when $n = n_0$
2. Inductive hypothesis
   - Assume formula is true for an arbitrary $n = k$

     where, $k \in Z^+$ and $k \geq n_0$
3. To Prove Claim
   - Show that formula is then true for $k+1$

Note: In fact we have to prove

1) $P(n_0)$ and
2) $P(k) \Rightarrow P(k+1)$

Mathematical Way of Expressing Induction

- *Basis step*.

  Show that proposition $P(1)$ is true.

- *Inductive step*.

  Show that for every positive integer n, the implication $P(n) \rightarrow P(n+1)$ is true.

  $P(n)$ for a fixed n is called **inductive hypothesis.**

- $[P(1) \wedge \forall n, (P(n) \rightarrow P(n+1))] \rightarrow \forall n, P(n)$

Well Ordering and Modus Ponens Principal

Definition (Well-Ordering Principle)

- The Well-ordering Principle is the following statement "*every nonempty set of positive integers contains a least element*"
- In a mathematical way we can define this Principle as:

  there is a in S such that $a \leq b$ for all b in S i.e.

  $\exists\ a \in S$, such that $a \leq b$, $\forall\ b \in S$
- And we say that set S is *well-ordered* with respect to $\leq$.

Modus Ponens Principal

$$p \Rightarrow q$$
$$p$$
$$\text{Hence, } q$$

Why Mathematical Induction is Valid?

- Let's suppose that P(1) is true, and that
  $\forall k\,(P(k) \rightarrow P(k+1))$ is also true,
- Claim: $\forall n\, P(n)$ is true
  - Assume proposition $\forall\ n$, $P(n)$ is false, i. e, there are some positive integers for which $P(n)$ false.
  - Let S be the set of those n's. By well-ordering property, S has a least element, suppose, k.
  - As $1 \notin S$, *so* $1 < k$, so k-1 is a positive
  - Since k-1 < k, hence k-1$\notin S$. So $P(k\text{-}1)$ is true.
  - By modus ponens, $P((k\text{-}1) + 1) = P(k)$ is true.
  - Contradiction, hence $\forall n$, $P(n)$

Another Reason for Validity?

Basis Step

First suppose that we have a proof of P(0).

Inductive Hypothesis

$\forall\ k > 0, \qquad P(k) \Rightarrow P(k + 1)$

How it is proved $\forall\ n > 0$,?

$P(0) \Rightarrow P(1)$

$P(1) \Rightarrow P(2)$

$P(2) \Rightarrow P(3)$

. . .

Iterating gives a proof of $\forall$ n, P(n). This is another way of proving validity of mathematical Induction.

Strong Mathematical Induction

- Let P(n) be a predicate defined for integers n, and *a* and *b* are fixed integers with $a \leq b$.
- Suppose the following statements are true:

  1. P(a), P(a + 1), … , P(b) are all true

     (basis step)

  2. For any integer k > b,

     if P(i) is true for all integers i with $a \leq i < k$,

     then P(k) is true. (inductive step)
- Then P(n) is true for all integers $n \geq a$.

Lecture No 6: Fibonacci Sequences

- Start with a pair of rabbits, one male and one female, born on January 1. Assume that all months are of equal length and that rabbits begin to produce two months after their own birth. After reaching age of two months, each pair produces another mixed pair, one male and one female, then another mixed pair each month, and no rabbit dies.

  How many pairs of rabbits will there be after one year?
- Construction of Mathematical Model
- Explicit Formula Computing Fibonacci Numbers
- Recursive Algorithms, Generalizations of Rabbits Problem and Constructing its Mathematical Models
- Applications of Fibonacci Sequences

Lecture 7, 8, & 9: Recursion

- Recursion? Recursive Mathematical Models
- Solving Recurrence Relations
- First and second Order Linear Homogenous Recurrences with Constant Coefficients its Characteristics and Solution
- General Homogenous Recurrences, Characteristics and solution
- Solution: General Homogenous Recurrence when
  - Roots distinct, repeated, multiplicity of root is k
  - many roots with different multiplicities
- Non-homogenous Recurrence Relations
- Characteristics and solution of various type of non-homogenous recurrence relations

Lecture 10 & 11: Asymptotic Notations

- Major Factors in Algorithms Design
- Complexity Analysis
- Growth of Functions
- Asymptotic Notations
- Usefulness of Notations
- Reflexivity, Symmetry, Transitivity Relations over Θ, Ω, O, ω and o
- Relation between Θ, Ω and O
- Various Examples Explaining each concept

Big-Oh Notation (O)

If f, g: N $\rightarrow$ R^+, then we can define Big-Oh as

For a given function $g(n) \geq 0$, denoted by $O(g(n))$ the set of functions,

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_o such that

$0 \leq f(n) \leq cg(n)$, for all $n \geq n_o \}$

$f(n) = O(g(n))$ means function $g(n)$ is an asymptotically upper bound for $f(n)$.

We may write f(n) = O(g(n)) OR f(n) $\in$ O(g(n))

Intuitively:
Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

Big-Omega Notation (Ω)

If f, g: N $\rightarrow$ R^+, then we can define Big-Omega as

For a given function $g(n)$ denote by $\Omega(g(n))$ the set of functions,

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_o such that

$0 \leq cg(n) \leq f(n)$ for all $n \geq n_o \}$

$f(n) = \Omega(g(n))$, means that function $g(n)$ is an asymptotically lower bound for $f(n)$.

We may write f(n) = Ω(g(n)) OR f(n) ∈ Ω(g(n))

Intuitively:
Set of all functions whose *rate of growth* is the same as or higher than that of *g*(*n*).

Theta Notation (Θ)

If f, g: N $\rightarrow$ R^+, then we can define Big-Theta as

For a given function $g(n)$ denoted by $\Theta(g(n))$ the set of functions,

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2 and n_o such that

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_o \}$

$f(n) = \Theta(g(n))$ means function $f(n)$ is equal to $g(n)$ to within a constant factor, and $g(n)$ is an asymptotically tight bound for $f(n)$.

We may write f(n) = Θ(g(n)) OR f(n) ∈ Θ(g(n))

Intuitively: Set of all functions that have same *rate of growth* as $g(n)$.

Relations over Asymptotic Notations

Reflexivity

- All the relations, Θ, Ω, O, *are reflexive*
- Small o and small omega are not reflexive relations

Symmetry

- Θ is symmetric.
- Big O, big omega Ω, little o, and little ω, do not satisfy the symmetry property.

Transitivity

- All complexity measuring notations Θ, Ω, O, ω and o satisfy the transitive property.

Lecture 12 &13: Brute Force Approach

Brute Force Approach,

- Checking primality
- Sorting sequence of numbers
- Knapsack problem
- Closest pair in 2-D, 3-D and n-D
- Finding maximal points in n-D

Lecture 14: Divide and Conquer

Divide and Conquer?

- Merge Sort algorithm
- Finding Maxima in 1-D, and 2-D
- Finding Closest Pair in 2-D

Lecture 15 & 16: Dynamic Programming

- Optimizations problem?
- Steps in Development of Dynamic Algorithms
- Why dynamic in optimization problem?
- Chain-Matrix Multiplication
- Problem Analysis
  - Brute Force approach
  - Time Complexity
- Chain-Matrix Multiplication
- Using Dynamic
  - Notations
  - Dynamic Algorithm
  - Time Complexity

Lecture 15 & 16: Chain Matrix Multiplication

Statement: The chain-matrix multiplication problem can be stated as below:

- Given a chain of $[A_1, A_2, \ldots, A_n]$ of n matrices where for i = 1, 2, . . . , n, matrix A_i has dimension p_{i-1} x p_i, find the order of multiplication which minimizes the number of scalar multiplications.

Note:

- Order of A_1 is p_0 x p_1,
- Order of A_2 is p_1 x p_2,
- Order of A_3 is p_2 x p_3, etc.
- Order of A_1 x A_2 x A_3 is p_0 x p_3,
- Order of A_1 x A_2 x . . . x A_n is p_0 x p_n

Steps in Development of Dynamic Algorithms

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution in a bottom-up fashion
4. Construct an optimal solution from computed information

   Note: Steps 1-3 form the basis of a dynamic programming solution to a problem. Step 4 can be omitted only if the value of an optimal solution is required.

Lecture 17 & 18: Assembly Line Scheduling

- Assembly Line Scheduling Problem
- Problem Analysis
  - Notations, Brute Force and Dynamic Solutions
- Algorithm using Dynamic Programming
- Time Complexity
- n-Line Assembly Problem
- n-Line Assembly Algorithm using Dynamic Programming
- Time Complexity
- Applications

Lecture 17 & 18: Assembly-Line Scheduling

- There are two assembly lines each with n stations
- The jth station on line i is denoted by $S_{i, j}$
- The assembly time at that station is $a_{i,j}$.
- An auto enters factory, goes into line i taking time e_i
- After going through the jth station on a line i, the auto goes on to the (j+1)st station on either line
- There is no transfer cost if it stays on the same line
- It takes time $t_{i,j}$ to transfer to other line after station $S_{i,j}$
- After exiting the nth station on a line, it takes time x_i for the completed auto to exit the factory.
- Problem is to determine which stations to choose from lines 1 and 2 to minimize total time through the factory.

Lecture 19 & 20: 0-1 Knapsack Problem

- 0-1 Knapsack Problem
- Problem Analysis
  - Divide and Conquer
  - Dynamic Solution
- Algorithm using Dynamic Programming
- Time Complexity
- Generalization, Variations and Applications
- Optimal Weight Triangulation
  - Definitions, Problem Analysis
  - Dynamic Solution
  - Algorithm using Dynamic Programming
  - Time Complexity
- Conclusion

Lecture 21: Longest Common Subsequence

- Longest Common sub-sequence problem
- Problem Analysis
  - Brute Force approach
  - Dynamic Solution
- Algorithm using Dynamic Programming