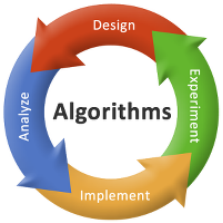
Computational Thinking

Course: Algorithms

Faculty: Dr. Rajendra Prasath



Autumn 2018

Computational Thinking

This class covers many important aspects of **Computational Thinking** to be considered while solving a specific problem. This lecture illustrates the various facets of algorithms design principles

Recap: Scalable Solutions

- Can you How to prepare the solution to work for the given problem with sufficiently large input size ?
- Whether the Solutions (Algorithms) you proposed could work for sufficiently large input size ??
- What are the issues in scaling up the solution?
- The given problem has to be solved **with the resources in hand** (!!)

Recap: Algorithm Pattern

- **Nomenclature**
  - *Name* (Descriptive Name of the algorithm)
  - *Context* (illustrating an essential part of the Algo)
  - *Facts* (Properties that could cause anyone to choose the algorithm specifically)
  - *Consequences* (Advantages and Disadvantages)
  - **Analysis of the algorithm (understanding the behavior of the algorithm)**
    - **why does an algorithm behave like this!!**
    - **Can we come up with lemmas and proofs to explain the behavior of the algorithm?**
  - *Alternatives* (find and compare other competitive variations of solutions to the same problem)

Recap: Take-HOME Assign

- **Problem:**
  - Write an algorithm to insert n randomly generated integers into a complete binary tree without using either an array or a queue or the extra space to store all n integers?
  - Solutions:
    - You could use a little space for storing the traverse in the complete binary tree
    - Bitwise operators

Compute Sum(N): Solved?

- Given: N – an integer
- **Task:**
  - How to compute the sum of first n natural numbers?

- **Solutions:**
  A) Simply use a for loop
  B) Have two pointers at the end point of the list of first n natural numbers and do the sum until they cross over each
  C) Gauss Technique
  D) Arithmetic Progression $(s + (N-1)d)/2$

Example -2 (contd)

- Task: Find the largest number in a list of numbers of random order
- Solution: look at every number in the list

- **High-level description (Linear Scan):**
  - If there are no numbers in the list then there is no highest number
  - Assume the first number in the list is the largest number
  - For each remaining number in the list: if this number is larger than the current largest number, consider this number to be the largest number in the list
  - When there are no numbers left in the list to scan through, consider the current largest number to be the largest number of the list

Recap: Find the Largest

Task: Find the largest element in a given list L

Algorithm **getLargest**

Input: A list of numbers L having n elements

Output: The largest number in the list L

```
begin
    if n = 0 return null
        largest ← L[0]
    for each item in L, do
        if item > largest, then
            largest ← item
    return largest
end
```

Algorithms ?

- **Definition:**
  - **An algorithm can be defined as a collection of unambiguous (precise) executable instructions, whose step by step execution leads to a predefined goal, within a finite number of steps**

  OR
  - **a set of rules that precisely defines a sequence of operations to reach the solution in finite no. of steps**

- Algorithmic Thinking
  - Enables a step by step method of solving a problem
  - Helps to design efficient algorithms
  - A complex intellectual process of thinking, combining facts, skills, and so on
  - Effective algorithms: Finite **time** and **space**

Good or Efficient Algorithms?

- **Easy to understand**
- **Easy to Implement**

- Efficient Algorithms
  - **Provides an added insight to the problem**
  - **Solutions must be Simple and Elegant**

- Needs a model of computation
  - Develop a denotational definition of complexity
- Of course Too much details will **Obscure the most beautiful** Algorithms

First Machine Model?

- **Turing Machine** – The First **computing machine**
  - **consists of a finite state control**
  - **a two-way infinite memory tape divided into squares, each of which can hold one symbol and**
  - **a read / write head**
  - **In one step, the machine can**
    - **Read the contents of one tape square**
    - **Write a new symbol in that square**
    - **Move the head one square left or right and**
    - **Change the state of the control**
- Such Turing Machine are useful in high-level theoretical studies of computational complexity
- But not realistic to follow the accurate analysis of algorithms

Example: GCD

- **What is the GCD of 10 and 15?**
  - Ans = 5
- **What is the GCD of 1599 and 650?**

  $1599 = 650 \times 2 + 299$

  $650 = 299 \times 2 + 52$

  $299 = 52 \times 5 + 39$

  $52 = 39 \times 1 + 13$

  $39 = 13 \times 3 + 0$

  - Ans = 13
  - Using Least Common Multiple (LCM)
    - The smallest number that is a multiple of all the numbers

Formalization

- **How do the computers process the data**
  - **Essential ways to instruct the computers to do operations on the data for the given task**
  - **Example:**
    - **Preparing your CGPA per semester and also their average across all semesters**
    - **Essential parts: Data with the associated input and output sources.**
      - **This forces to focus on Data Structures and the operations on the chosen data structures.**
- The order or execution of always crucial to the functioning of the algorithm
  - Flow Control (Top – Down approach)

Expressing Algorithms

- **What are the ways to express Algorithms?**
  - **Pseudocode**
  - **Flowchats**
  - **Programming languages**
  - **State Transition**
  - **Control tables**
  - **Natural Language Description (in fact hard for machines to disambiguate)**
- Efficient Design Principles
  - High-level Description (Procedure)
  - Implementation Description (Data Structures)
  - Formal Description (State Transitions)

Algorithm Design

- A method or even a Mathematical process for problem solving
- Design an algorithm with efficient run time

- **Steps:**
  - **Problem Definition**
  - **Develop a Mathematical model of computation**
  - **Specification of the algorithm**
  - **Design the algorithm**
  - **Check the correctness of the algorithm**
  - **Analysis of the algorithm**
  - **Choice of proper data Structures and implementation**
  - **Perform Program Testing**
  - **Documentation of the above procedure**

- Good algorithm – simplicity and its elegance

Classification of Algorithms

- **Various Ways**
  - **By implementation**
  - **By Design Paradigm**
  - **By Domain Specific study**
  - **By Computational Complexity**
- There are different variations of algorithms and their purpose might be different from the classical theory of algorithms

By Implementation

- **Recursion**
  - Base case
  - Recursive case
- **Logical Deduction**
  - logic component expresses the axioms that may be used in the computation and the control component
- **Sequential and Parallel algorithms**
  - Whether one process or multiple processes work to solve a problem
- **Distributed Algorithms**
  - Multiple processes to work on the different tasks at the same time

By Implementation (contd)

- **Deterministic Algorithms**
  - Exact Decision at every step
- **Non-Deterministic Algorithms**
  - Solve problems via guessing
  - More accurate through the use of heuristics
- **Approximate Algorithms**
  - Seeking solutions closer to the original solution
  - Hard problems (Knapsack Problem)
    - You have a set of items and a knapsack
    - The goal of the problem is to pack the knapsack to get the maximum total value
- **Quantum Algorithms**
  - Uses spin of atoms … quantum computation

By Design Paradigm

- **Brute-Force / Exhaustive Search**
  - Trying every possible solution
- **Divide and Conquer**
  - Reduce the problem instance to one or more smaller instances and solve each of them easily
  - Merge Sort
- **Search and Enumeration**
  - Playing Chess
  - Graph Exploration Algorithm
  - Maps for finding the fastest route
- **Randomized algorithms**
  - Suitable where finding exact solutions can be impractical
    - Monte – Carlo algorithms return a correct answer with high probability

By Design Paradigm (contd)

- **Reduction**
  - Finding a reducing algorithm
    - Finding the median in the unsorted list (expensive)
    - Instead find the median in the sorted list (cheaper)

- **Backtracking**
  - Build multiple solutions

  - How to reach bottom left square to the top-right square following the specific movements (up and right)?
  - Abandoned when this can not give a valid solution

Optimization Problems

- **Linear Programming**
  - Inequality Constraints
    - Simplex algorithm
    - Maximum flow problem
    - Assignment Problem
  - To arrive at optimal solutions

- **Dynamic Programming**
  - For problems showing substructures
  - Same subproblem can be solved using different problem instances
    - Floyd – Warshall Agorithm (Shortest Path problem)
  - Reduces the exponential nature of many problems to polynomial complexity

Optimization Problems (contd)

- **The Greedy Methods**
  - Similar to Dynamic Programming
  - Many algo can find optimal but others struck at local optima
    - Minimal Spanning tree algorithms (Kruskal, Prim, Huffman Tree)

- **The Heuristic Method**
  - Can find solutions closer to the optimal solution in cases where finding the optimal solution is impractical.
  - Run for infinite amount of time (find the optimal solution)

By Field Study

- **Every Field has its own problems and need efficient solutions**
  - Large Scale Sorting / Searching algorithms
  - Numerical Algorithms
  - Graph Algorithms (social network or small world graphs or complex networks)
  - Medical Algorithms
  - Machine Learning
  - Cryptography
  - Combinatorial algorithms
  - Data Compression
  - Natural Language / Parsing Algorithms
  - Geometric Algorithms and so on

By Complexity

- **The amount of time they need to complete compared to their input size:**
  - **Constant time:** if the time needed by the algorithm is the same, regardless of the input size
    - Accessing a specific element in the given array
  - **Linear time:** if the time is proportional to the input size
    - Traverse elements in a list
  - **Logarithmic time:** if the time is a logarithmic function of the input size
    - Binary search algorithm
  - **Polynomial time:** if the time is a power of the input size.
    - Bubble sort algorithm has quadratic time complexity.
  - **Exponential time:** if the time is an exponential function of the input size
    - Exhaustive search
  - Every Field has its own problems and need efficient solutions

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

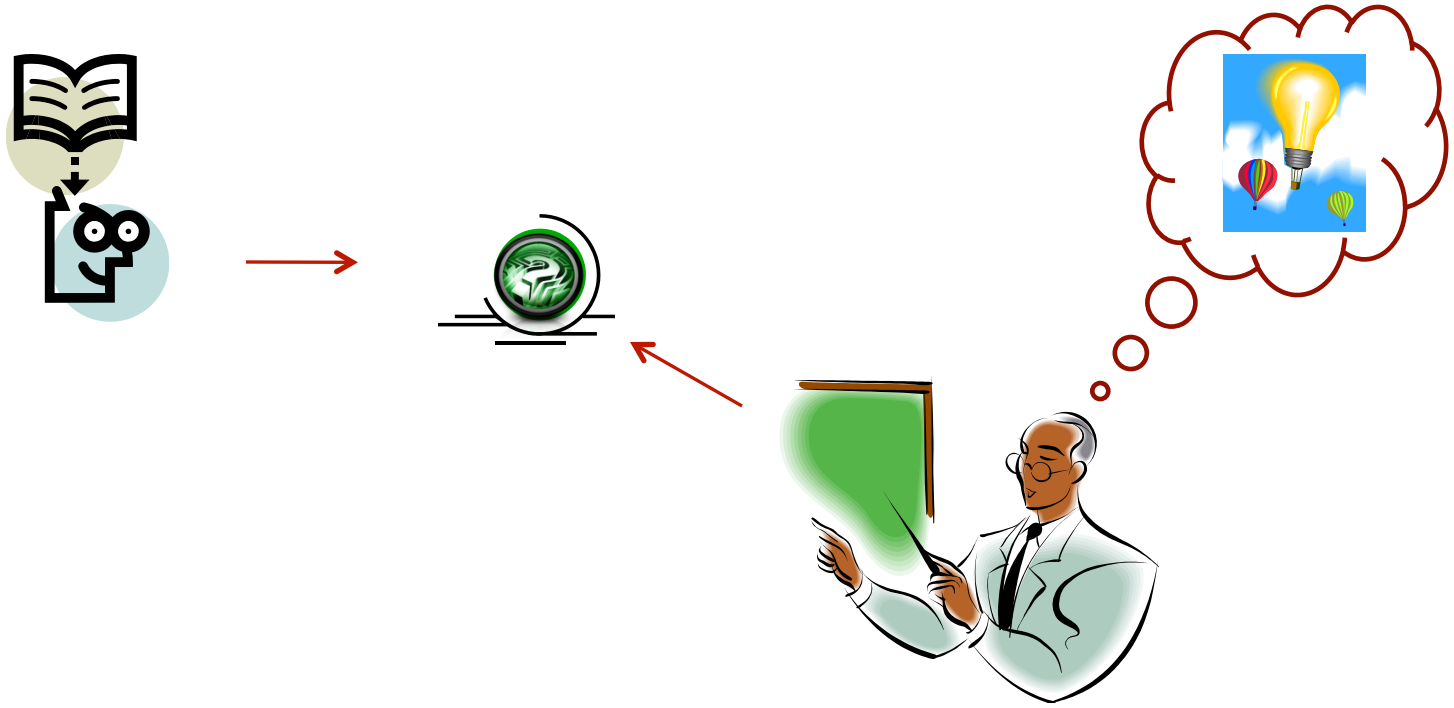
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???