INTRODUCTION TO ALGORITHMS AND PROBLEM SOLVING

AGNI DATTA

May 20, 2021

Contents

Part I

THEORY

1 ALGORITHM:

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time. An Algorithm, therefore, corresponds to a solution to a problem that is independent of any programming language.

A well-defined computational procedure that takes some value or a set of values as input and procedures to find the output. It represents the solution to a problem in a step wise manner.

---

2 PROGRAM:

A set of instructions is known as a program. A program is an expression of an algorithm in a programming language.

---

3 CHARACTERISTICS OF AN ALGORITHM:

- ⋆ Finiteness i.e., the number of steps.
- ⋆ Definiteness i.e., the steps should be well defined.
- ⋆ Generality i.e., the program should be generic and be able to solve a certain class of problems.
- ⋆ Effectiveness i.e., the code should be basic enough to be easily understandable.
- ⋆ IO the code should be able to take inputs and produce outputs.

---

4 PROCEDURE OF WRITING AN ALGORITHM:

1. Breaking the problem into sub problems.
2. Choose a suitable data structure.
3. Constructions of loops if necessary.
4. Establishing the initial condition for loops.
5. Finding the iterative construct.
6. Termination of loops.

This is also known as the top-down design.

---

5 GENERAL PROBLEM-SOLVING METHODS:

⋆ Brute Force,

⋆ Divide and Conquer,

⋆ Decrease and Conquer,

⋆ Transform and Conquer,

⋆ Greedy Programming,

⋆ Backtracking,

⋆ Branch and Bound,

⋆ Approximation Algorithms,

⋆ Space and Time Trade-offs,

⋆ Dynamic Programming,

⋆ Exhaustive Search.

---

6 PSEUDO CODE:

Pseudo Code is a plain language description of the steps in an algorithm or another system. Pseudo Code often uses structural conventions of a normal programming language, but is intended for human reading rather than machine reading. The immediate representation of a problem between an algorithm and a program. Pseudo Code is a method of representing an algorithm. There are a lot of formats used for the representation of pseudo code and most of them borrow some of the structures from popular programming languages.

Some programming constructs used for pseudo code include:

⋆ `read`

⋆ `print`

⋆ `set`

⋆ `initialize`

⋆ `if-then-end if`

⋆ `if-then-else-end if`

⋆ `repeat-until`

⋆ `increment`

⋆ `goto`

Pseudo Code closely relates to a programming language.

---

7 FLOWCHART:

A schematic representation of a sequence of operations, as in a manufacturing process or computer program.

or

A graphical representation of the sequence of operations in an information system or program.

- ⋆ Information system flowcharts show how data flows from source documents through the computer to final distribution to users.
- ⋆ Program flowcharts show the sequence of instructions in a single program or subroutine.
- ⋆ Flow charts is a graph used to depict or show a step-by-step solution using symbols which represent a task.
- ⋆ The symbols used consists of geometrical shapes that are connected by flow lines. It is an alternative to pseudo coding; whereas a pseudo- code description is verbal, a flow chart is graphical.

---

8 PROGRAMMING LANGUAGES:

A programming language is a formal language comprising a set of strings that produce various kinds of machine code output.

It is a tool for developing executable models for a class of problem domains.

8.1 LEVELS OF PROGRAMMING LANGUAGE:

8.1.1 LOW-LEVEL LANGUAGE:

Machine Level Language this includes instructions in 1s and 0s i.e., in binary.

Example: Binary Language.

8.1.2 MIDDLE-LEVEL LANGUAGE:

Assembly Level Language uses mnemonics to create instructions.

Example: Assembly Level Language.

8.1.3 HIGH-LEVEL LANGUAGE:

Very similar to human-level language.

Examples: COBOL, FORTRAN, BASIC, C, JAVA

---

9 TYPES OF PROGRAMMING LANGUAGE:

- ⋆ Batch Programming (BASIC, FORTRAN, COBOL)
- ⋆ Structured Programming (PASCAL, C)
- ⋆ Object-Oriented Programming (C++, Java, C#, PYTHON)

- ⋆ Logic/Declarative
- ⋆ Programming (Prolog)
- ⋆ Functional/Applicative Programming (Lisp)

---

10 HIGH-LEVEL LANGUAGES:

10.1 C:

- ⋆ Developed by Bell Laboratories in the early 1970s.
- ⋆ Provides control and efficiency of assembly language while having third-generation language features.
- ⋆ Often used for system programs.
- ⋆ UNIX is written in C.

10.2 C++:

- ⋆ It is C language with additional features.
- ⋆ Widely used for developing system and application software.
- ⋆ Graphical user interfaces can be developed easily with visual programming tools.

10.3 HTML:

- ⋆ Hypertext Markup Language.
- ⋆ Used on the Internet and the World Wide Web
- ⋆ Web page developer puts brief codes called tags in the page to indicate how the page should be formatted.

10.4 XML:

- ⋆ Extensible Markup Language.
- ⋆ A language for defining other languages

10.5 JAVA:

- ⋆ An object-oriented language similar to C++ that eliminates lots of C++'s problematic features.
- ⋆ Allows a web page developer to create programs for applications.
- ⋆ The objective of JAVA developers is that it be a machine, platform and operating system independent.

---

11 DIFFERENCE BETWEEN A HIGH AND LOW-LEVEL PROGRAMMING LANGUAGE:

⋆ Low-level languages work more closely with hardware and do not require a compiler to be executed.

⋆ High-level languages are more understandable for the programmer in terms of the words in the code.

---

12 VERIFICATION AND VALIDATION:

The program being developed must be checked to ensure that it meets its specification and delivers the functionality expected by the people paying for the software.

12.1 VERIFICATION:

Are you building the product, right? The software must conform to its specification.

12.2 VALIDATION:

Are you building the right product? The software should do what the user requires.

---

13 EFFICIENCY OF ALGORITHMS:

Every algorithm must use up some of the computer resources to complete its tasks. The resources most relevant to efficiency are central processor time and internal memory. The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity, or volume of memory, known as space complexity. A good algorithm is correct, but a great algorithm is both correct and efficient. The most efficient algorithm is one that takes the least amount of execution time and memory usage possible while still yielding a correct answer. Algorithm efficiency is used to describe properties of an algorithm relating to how much of various types of resources it consumes. The run time complexity of an algorithm is the amount of time that it takes to complete once it has begun. The space complexity of an algorithm is the amount of storage space that it requires while running from start to completion.

---

14 PARAMETER:

A parameter is a piece of data provided as input to a function or procedure.

---

15 ANALYSIS OF ALGORITHMS:

Quantitative measures are valuable in that they can give us a way of directly predicting the performance of an algorithm and of comparing the relative performance of two or more algorithms that are intended to solve the same

problem. This can be important because the use of an algorithm that is more efficient means saving in computing resources which translates into a saving in time and memory.

We can have three cases to analyze an algorithm:

15.1 Worst Case:

n the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search( ) functions compares it with all the elements of $arr[\ ]$ one by one. Therefore, the worst case time complexity of linear search would be $\Theta(n)$.

15.2 Average Case:

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by $(n + 1)$. Following is the value of average case time complexity

15.3 Best Case:

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be $\Theta(1)$ Most of the times, we do worst case analysis to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is good information. The average case analysis is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know (or predict) the mathematical distribution of all possible inputs. The Best Case analysis is useless. Guaranteeing a lower bound on an algorithm does not provide any information as in the worst case, an algorithm may take years to run. For some algorithms, all the cases are asymptotically same, i.e., there are no worst and best cases.
