Handwritten Notes

Data Structures & Algorithms

SAVITRIBAI PHULE
PUNE UNIVERSITY

1. Introduction to Algorithm & Datastructure.

* **Data**: Data is nothing but the collection of facts & figures or data is value or group of values which is in perticular Format.

The data must be stored in a systematic format so that one can easily perform different operations on it.

All the programming languages provides a set of built in data types stored the data such as int, float, char etc.

* **Data Structure**: It is a method of gathering as well as organizing data in such a manner that several operations can be performed on the data in an effective way.

A data structure is a logical model of a perticular organization of data.

* **Problem**: problem is defined as a situation or issue or condition which needs to solve to achive the goals.

* **Algorithm**: Algorithm is a set of ordered instructions which are written in simple English language.

Algorithm defines the step by step logic

of a program to solve the specific problem.

* Information : Information can be defined as a structured or classified data which delivers some meaningful things to the recipient.

Information is nothing but the processed data on which decisions & actions can be taken.

Information can have following characteristics:

(I) Accuracy.

(II) Completeness.

(III) Timely.

(I) Accuracy : Information must preserve the accuracy.

(II) Completeness : Information must be in a complete format.

(III) Timely : Information should be available 24/7.

* Knowledge : knowlede can be defined as knowing the things before those can be experienced.

It can be alternate alertness, or understanding of something such as data, information, details, or ability, which is acquired through practice or learning the things in academics.

Data object: A data object is a region of storage that contains a value or group of values.

Each value can be accessed using its identifier or a more complex expression that refers to the object.

It is also used in any type checking operations. Both the identifier & data type of an object are established in the object declaration.

* Need of data structure:

(I) Storage huge data: The basic data elements provided by programming languages like variable, array, list & dictionaries are not sufficient to store & manage such huge data.

Data structures provide a way to store & manage huge data.

(II) Stores in systematic way: After storing data variety of operations have to be performed on data of various types like numeric, string etc.

Data structure help to store the data in systematic way to ease the operations.

It makes easy to store & manipulate data

(III) Retains logical relationship: Data structure help to retain the logical relationship between the data elements.

(V) Provides various structures: Data structures provides various structures such as stack, Queue, linked list etc. to store the data as per the need of applications.

(VI) Static & Dynamic Format: Data structures provides static as well as dynamic formats to store the data.

(VII) Better algorithm: Data structures provides better algorithms to apply on organized data to improve efficiency of program.

* Abstract Data Types: (ADT): An ADT is a mathematical model for data types, where a data types is defined by it's behaviour from the point of view of a user of a user of the data, specifically in terms of possible values, possible operations on data of this type & the behaviour of these operations.

An ADT may be defined as a "class of objects whose logical behaviour is defined by a set of values & set of operations."

ADT refers to defining our own data types.

ADT is useful tool which helps to specify logical properties of data types without the need of going into details.

Data structure classification : There are two types of Data structure:

(I) Primitive Data structure.

(II) Non-Primitive Data structure.

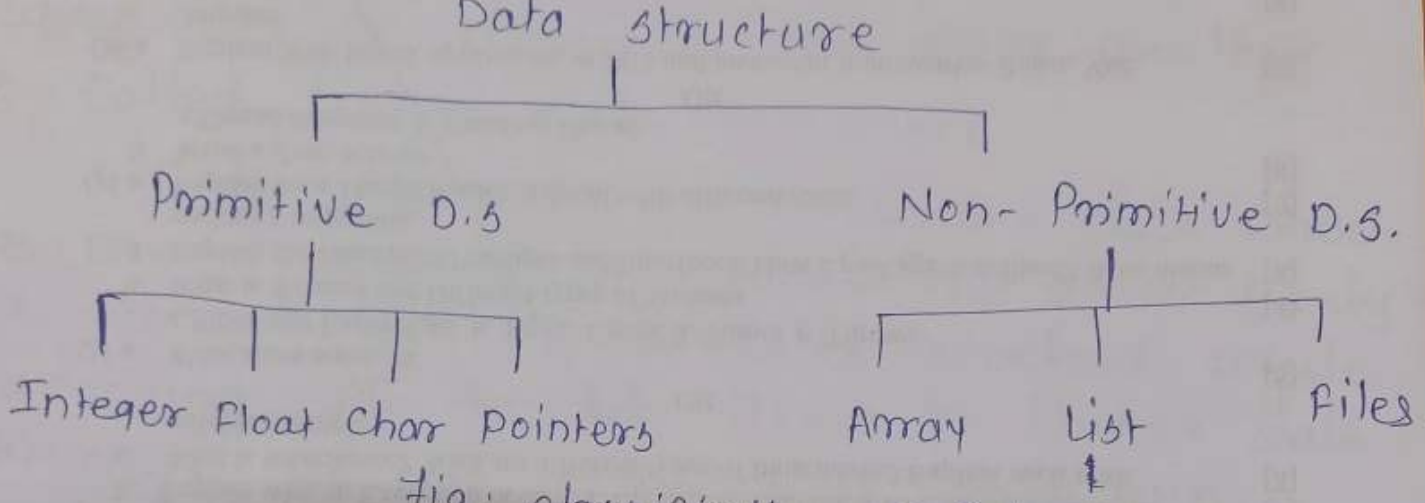


fig: classification of Data structure:

(I) Primitive Data Structure: Primitive data structure are the basic data structure that directly operate upon the machine instructions.

These are the basic data types provided by various programming language.

Examples: Integer, floating point numbers, character, string, pointer etc.

(II) Non-Primitive Data structures: Non-primitive D.S are more complicated data structures & are derived from primitive Data structure.

They emphasize on grouping same or different data items with relationship between each data item.

examples: Arrays, Lists & files.

* Linear Data Structure: The data structure where data items are arranged or organized sequentially or linearly one after another is called linear data structure.

- Data elements in a linear data structure are traversed one after the other & only one element can be directly reached while traversing. All the data items in linear data structure can be traversed in single run.

Examples: Stack & Queue, Array, Linked list.

* Non-linear Data structure: A data structure in which the data items are not organized sequentially or in linear fashion is called as Non linear data structure.

In other words, a data element of the non-linear data structure could be connected to more than one element to reflect a special relationship among them.

examples: Tree & graph.

Static Data Structure: In static data structure size of the structure is fixed.

It is possible to modify the content of the data structure but without making the content of the data structure but without making changes in the memory space allocated to it.

example : Array.

* Dynamic Data Structure: Dynamic data structure are designed to facilitate change of data structure at runtime.

It is possible to change the assigned values of elements dynamically.

In this data structure, the initially allocated memory size is not a problem. It is possible to add new elements, remove existing elements or do any kind of operations on data set without considering the memory space allocated initially.

examples: Linked list.

* Persistent Data Structure: A persistent data structure is a data structure that always preserve the previous version of itself when it is modified.

A data structure is partially persistent if all versions can be accessed but only the newest version

an be modified.

The D.S. is fully persistent if every version can be both accessed & modified.

* Ephemeral Data Structure: An ephemeral data structure is one for which only one version is available at a time. After an update operation, the structure as it existed before update operation is lost.

* Algorithm - Problem Solving:

* Problem: Problem is defined as a situation or issue or condition which needs to solve to achieve the goal.

* Steps in Problem Solving:

① Define the problem: A situation or issue or condition which needs to solve achieve the goal is called as a problem.

② Data gathering: It means that to involves the process to create list of required data to solve the problem.

gathering of data may contains key, terms, some assumptions, previous experience & also opinions of other peoples towards the problem.

Decide Effective Solution: After getting the details about the problem & their synonyms, finding the list of solutions is the main goal.

Among all the solutions, choose a solution which is effective & efficient. & has lowest side effect.

④ Implement & Evaluate the solutions: The selected solution must be implemented & then evaluated & check whether the expected output is generated or not.

The implementation of solution involves planning & execution of that task. It is an iterative process until desired solution is not getting.

Evaluation is process of checking whether the solution is as per expectations or not. The solution is evaluated by all means.

⑤ Review the Results: It is important to cross verify the result by applying variant inputs to know the success of the problem solution. It also helps you to improve long-term problem solving skills & keeps you from re-inventing the wheels.

Problem Solving Techniques:

① Algorithm: Algorithm is set of ordered instructions which are written in simple English language.

Algorithm defines the step by step logic for a program to solve specific problem.

(2) Flowchart: Flowchart is good problem solving technique.

Flowchart is graphical representation of an algorithm.

The sequence of steps in algorithm is maintained & represented in the Flowchart by using some standard symbols such as rectangles & directed lines.

* Some other problem solving techniques:

① Trial & error technique: Trial & error is also known as generate & test. It is a way of solving problem through repeated attempts, trying varient data on sequence steps until you are successful.

② Brain storming technique: It is a group activity. Group members are gathering together to discuss a problem & give soln to it.

③ Divide & Conquer technique:

* Introduction to Algorithms:

* Characteristics of Algorithm:

① Input: An algo. may or may not accept i/p.

In some problem definitions, algo. can have zero i/p.

② Output: An algo. must generate some o/p.
The aim of algo. is to provide sequence of steps to achieve goal. So every algo. must provide o/p at the end.

③ Definiteness: Each instruction should be clear & unambiguous.

④ Effectiveness: Each instruction should have proper meaning & effectiveness to produce the final result.

⑤ Finiteness: The algorithm must terminate after fixed no. of steps.

* Steps in Algorithm Development:

① Define the problem.
② Identifying input.
③ Identifying output.
④ Identifying the sequence of steps to be processed.

* Different Approaches to Designing ~~dd~~ An algorithm:

There are two main approaches used in designing an algorithm:

Top - Down Approach:

(II) Bottom - up Approach.

(I) Top - Down Approach: In this approach initially the main basic requirement is understood. The problem is then futher divided into sub tasks so as to ease the overall process.

Every sub task is again tried to divide in further subtask.

This process is continued until each & every tasks become simple enough to handle easily.

A top-down approach is essentially the breaking down of a system to gain insight into its compositional sub-systems in a reverse engineering fashion.

(II) Bottom up Approach: This approach is considered as completely opposite to top down approach.

In this approach, initially most basic & primitive components are designed & then proceed to higher level components.

That means in a bottom-up approach the individual base elements of the systems are first specified in great details.

Basic Conventions while writing an algorithm:

① **Algorithm Name**: Each algo. should have a name so that user/programmer will know for which program the algo. is. That's why the header contains the name of that algo.

② **Comments**: The content followed by "\\" symbol are considered as comments. comments are necessary to provide extra information about the statement.

③ **i/p & o/p**: i/p & ~~o/p~~ expected o/p of an algo. should be provided.

④ **Steps**: Provide no. to all the steps in algo. Numbering will help to refer specific statement again.

⑤ **Variables**: All the variable name should be written in uppercase.

⑥ **Assignment statement**: For assignment use "←" symbol to assign the value of right hand side to left hand side.

example : A ← 20

The RHS of ← symbol may be a value, another variable or an expression. whereas the left hand side of symbol ← symbol should be an individual variable.

IF Statement: This Statement will be written in following Format.

Syntax: if Condition 1:
    Statement 1;
else:
    Statement 2.

(8) Repeat Statement: This Statement will be written in following Format.

Syntax: Repeat the step until condition 1
    Statement 1.

(9) End of algo: Every algo. last line should be END

* Keywords Used in an algorithm:

(1) Steps in Algorithm: Each effective Statement of an algo. should be declared as a separate step.
Each step starts with the keyword "Step" followed by its no. & colon (:).

(2) Starting of algo: First step of every algo. should be written as follows with the "Start" keyword.

example: Step 1: Start.

Variable Declaration in Algorithm: variables are used to hold the data entered by users & use it for further operations.

So it is necessary in algo. to declare variables. while declaring variables it is good to use the keyword "~~Declare~~ Declare" followed by list of variables.

Example: Step 4: Declare variables NUM1, NUM2.

(4) Variable Initialization in Algorithm: while initializing variables, use the keyword "Initialize" followed by the variable & the value to be initialized to it.

Example: Step 4: Initialize COUNT ← 1

(5) Accepting i/p from User in algorithm: In some problems the i/p of an algo. is given by users. To indicate the process of accepting i/p from the user for variables we use "Read" keyword.

Example: Step 7: Read values NUM1 & NUM2.
~~SUM ← NUM1 + NUM2.~~

(6) Processing statement in algo: Processing statements may include some mathematical expressions & Formulas.

Example: Step 7: SUM ← NUM1 + NUM2.

Conditional Statements in algo: Conditional Statements are the statements which lead to change the sequence of Flow of an algo. according to the condition satisfied.

Use "IF" & "Else" keyword to define the condition & their operations.

Step 5: Example: Step 5: IF Condition 1.
Display condition 1 is True.
Else
Display condition 1 is False.

(8) Repeat - Until Statements in algo: The set of statements will be executed repeatedly until given condition is satisfying.

To specify the repeat - until statement in an algo. use "Repeat" & "until" keywords.

Example: Step 5: Repeat step 6 until condition 1.

(9) Displaying o/p of algo: Every algo produces at least one o/p. To display that o/p use of "Display" keyword in an algo. is a good practice.

Example: Step 8: Display SUM.

(10) Displaying o/p of algo: Every algo. should end after finite no. of steps. To indicate the end of an algo. use keyword "Stop"

Example: Step 15: Stop.

* Algorithm Design Tools: There are different ways to represent an algo. such as writing pseudo codes &

rawing Flowcharts.

Pseudo Code: Pseudo code is a method which is used to describe computer algo. through a combination of natural language. & programing language.

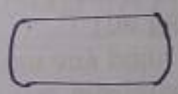
It is considered as an intermittent step before developing actual code.

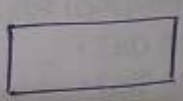
Pseudo code is in general frequently used but still there are no set of rules for its exact implementation.

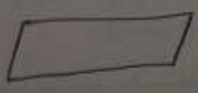
Flowchart: Flowchart is graphical representation of algo. The sequence of steps in algo is maintained & represented in the flowchart by using some standard symbols such as rectangles & directed lines.

The flowcharts allows breaking the complex problems into parts & at last connects them to represent solution for whole problem.

* Symbols of Flowcharts: It is constructed using standard shapes commonly known as flowcharts symbols.

(1)  Start / End terminal.

(2)  Process

(3)  i/p & o/p.

(4)  Flow line

Decision.

⑥ 

On-page Connector.

⑦ 

OFF page Connector.

⑧ 

Predefined Process (Function).

- Advantages of Flowchart:

① It is suitable for method of Communication betⁿ users/ programmers.
② it becomes easier to understand what actually is going on.
③ It becomes easier to trace the faults & correct them.
④ it can be used for documentation of systems.

* Disadvantages:

① Flowcharts are not suitable for commⁿ betⁿ human & a computer.
② If modification has to be done then the Flowchart must be redrawn which is wastage of time.
③ Flowcharts are waste of time & slow down the process of slw development.

Analysis of Algorithm:

① **Worst Case Analysis:** In the analysis of worst case, upper bound is calculated on running time of an algo.

One should have knowledge about case that leads to maximum no. of operations

② **Average Case Analysis:** In average case analysis, all possible i/p's are taken & computing time for all those i/p's is calculated.

- All the calculated values are added in sum & this sum is divided into total no. of i/p's.

One should know or predict the distribution of cases.

③ **Best Case Analysis:** In the analysis of Best Case lower bound is calculated on running time of an algo.

- one should have knowledge about the case that leads to maximum no. of operations to be executed.

* **Complexity of algorithm:** Complexity of an algo. F(N) provides the running time ~~f(n)~~ and/or storage space needed by the algo. with respect of N as the size of i/p data.

① **Space Complexity:** The space complexity depend on two components.

e byte code, variable space, constant space etc. This part is not dependent on characteristics of i/p & o/p.

(b) Variable part: instance of i/p & o/p data. This part consists of space necessary for variables, size of which depends upon specific problem instance being solved, space required for reference variable & recursion stack space.

(II) Time Complexity: The time complexity which is required for analysis of given problem of particular size is known as the time complexity.

Time complexity depends upon two components:

(a) Fixed Part: Compile time.

(b) Variable Part: Run time which depends upon problem instance.

* Asymptotic Notation: Asymptotic notations are used to get the rate of growth. Rather than dealing with exact expressions, it is possible to deal with asymptotic behaviour.

(I) Big-O (O) Notation: The Big-O analysis provides programmer some basis regarding computing & measuring the efficiency of a particular algorithm.

The Big-O help helps to identify the time & space complexity of the algorithm.

That means using Big-O notation, the time en by the algo. & the space required to run the algo. is determined.

Big-O notation is used to define the upper bound of an algo. in terms of Time Complexity.

Consider function f(n) the time complexity of an algo. & g(n) is the most significant term. : If $f(n) <= C\ g(n)$ for all $n >= n_0$, $C > 0$ & $n_0 >= 1$ then we can represent f(n) as O(g(n)).

* Orders of growth functions:

① O(1): it describes an algorithm that will always execute in the same time regardless of the size of the i/p data set.

② O(N): it describes an algo. whose performance will grow linearly & in direct proportion to the size of the i/p data set.

③ $O(N)^2$: it represents an algo. whose performance is directly proportional to the same square of the size of the i/p data set.

④ ~~O(N)~~ $O(2^N)$: $O(2^N)$ denotes an algo. whose growth doubles with each addition to the i/p data set.

Theta (Θ) notation: Theta notation is used to describe both upper bound as well as lower bound of an algo. hence it can be said that this notation can define exact asymptotic behaviour.

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2 & n_0 such that $0 <= c_1 * g(n) <= f(n) <= c_2 * g(n)$ for all $n >= n_0\}$.

(III) Omega (Ω) notation: Big-omega notation (Ω) is used to define the lower bound of an algo. in terms of time complexity.

That means Big-omega (Ω) notation always indicates the minimum time required by an algo. for all i/p values. That means Big-omega notation describes the best case of an algo time complexity.

Consider function $f(n)$ the time complexity of an algo. & $g(n)$ consider is the most significant term. If $f(n) >= c \times g(n)$ for all $n >= n_0$, $c > 0$ & $n_0 >= 1$. Then we can represent $f(n)$ as $\Omega(g(n))$.

* Algorithm Strategies:

(1) Divide & conquer: D&C is considered as an algo. design paradigm which has been based on multi-branched recursion.

D&C technique works by the process of recursively breaking down a problem into two or more than two sub-problem of the similar ty unless & until these become easy sufficient to be solved directly.

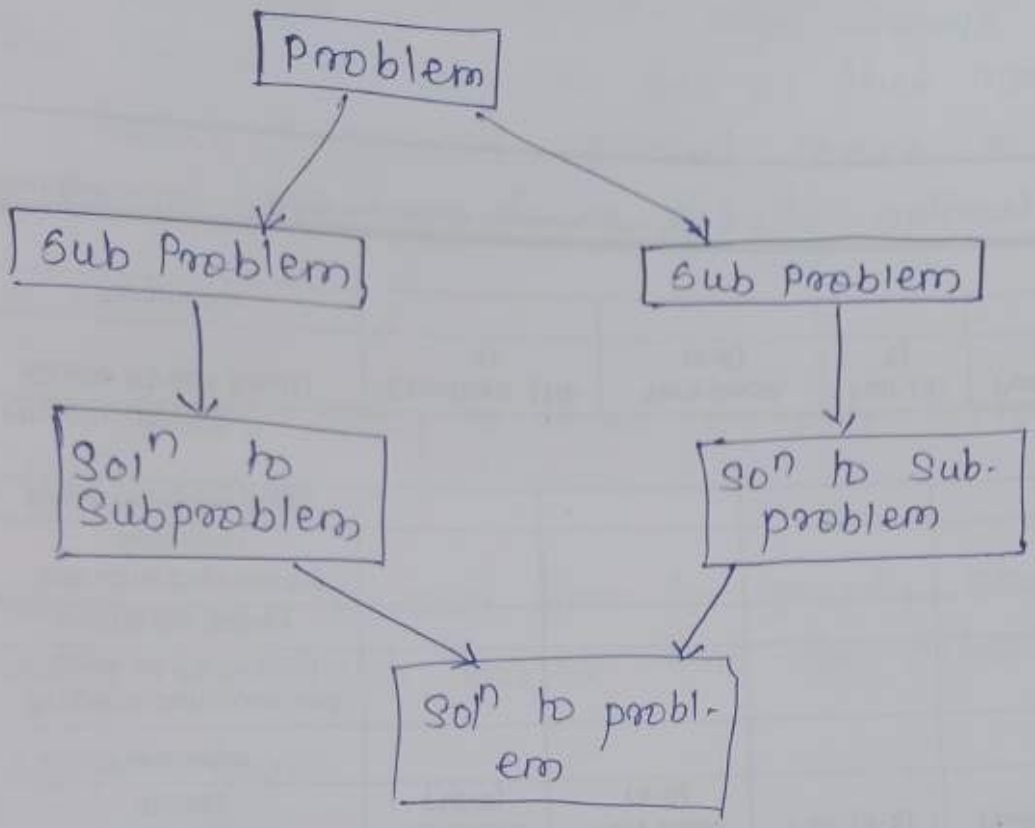


fig: Divide & Conquer.

All the respective soln of the sub-problems are further merged to form a soln to the original problem.

A typical D&C algo. follows three steps to solve any problem.

(1) Divide: Break the problem into sub-problems of same type.

(2) Conquer: Recursively solve these sub-problems.

(3) Combine: Combine the soln sub-problems.

The following algo. are based on D&C algo. design paradigm:

(1) Tower of Hanoi (2) Merge sort (3) Quick sort.

(4) Binary search.

Greedy Strategy: A greedy algo. always make the choice that seems to be the best at that moment. This means it makes a locally optimal choice in the hope that this choice will lead to a globally optimal soln.

* Advantages:

(1) It is quite easy to come up with greedy algo. for a problem.

(2) Analysizing the run time for greedy algo. will generally be much easier than for other techniques.

* Disadvantages:

(1) The difficult part is that for greedy algo. we have much harder to understand correctness issues.

* Applications:

(1) Travelling Salesman problem.
(2) Prim's Minimal spanning Tree algo.
(3) kruskal's Minimal spanning Tree algo.
(4) Graph - Coloring problem.