BCA 1ST Semester

BCA - 101: COMPUTER FUNDAMENTALS AND PROGRAMMING

UNIT: 3

(Programming Fundamentals)

Algorithm :

In mathematics and computer science, an algorithm (/ˈælgərɪðəm/) is a finite sequence of well-defined instructions, typically used to solve a class of specific problems or to perform a computation.

Algorithms are used as specifications for performing calculations, data processing, automated reasoning, automated decision-making and other tasks. In contrast, a heuristic is an approach to problem solving that may not be fully specified or may not guarantee correct or optimal results, especially in problem domains where there is no well-defined correct or optimal result.

An Algorithm Development Process

1. Obtain a description of the problem

This step is much more difficult than it appears. In the following discussion, the word *client* refers to someone who wants to find a solution to a problem, and the word *developer* refers to someone who finds a way to solve the problem. The developer must create an algorithm that will solve the client's problem.

2. Analyze the problem

The purpose of this step is to determine both the starting and ending points for solving the problem. This process is analogous to a mathematician determining what is given and what must be proven. A good problem description makes it easier to perform this step.

3. Develop a high-level algorithm

An algorithm is a plan for solving a problem, but plans come in several levels of detail. It's usually better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later. We can use an everyday example to demonstrate a high-level algorithm.

Problem: I need a send a birthday card to my brother, Mark.

Analysis: I don't have a card. I prefer to buy a card rather than make one myself.

4. Refine the algorithm by adding more detail

A high-level algorithm shows the major steps that need to be followed to solve a problem. Now we need to add details to these steps, but how much detail should we add? Unfortunately, the answer to this question depends on the situation. We have to consider who (or what) is going to implement the algorithm and how much that person (or thing) already knows how to do. If someone is going to purchase Mark's birthday card on my behalf, my instructions have to be adapted to whether or not that person is familiar with the stores in the community and how well the purchaser known my brother's taste in greeting cards.

5. **Review the algorithm**

   The final step is to review the algorithm. What are we looking for? First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem. Once we are satisfied that the algorithm does provide a solution to the problem, we start to look for other things. The following questions are typical of ones that should be asked whenever we review an algorithm. Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

Problem Solving Techniques :

1. Create the example of the algorithm

Before thinking about the logic of algorithm,We should understand the question clearly. Write atleast two examples , indicating input and output. This two minute initial work will remove the uncertainity of misunderstanding the question and thinking in wrong direction.

For example ,

Question is → **To find first non repeated character in the string**

Then, before writing algorithm follow this steps

Example 1 :

Input : A live Is Awesome

Output : l (small L)

Example 2 :

Input : Love Yourself

Output : v

2. Pattern Matching

We have to consider what problems the algorithm is similar to , we need to figure out if we can modify the solution to develop an algorithm for the given problem .

For example :

Question :A sorted array has been rotated so that the elements might appear in the order 3 4 5 6 7 1 2 . How would we find the minimum element.

Similar Problems:

1.Find the minimum element in an array.

2.Find a particular element in an array (eg. binary search).

3. Simplify and Generalize

Changing constraint (e.g size,length,data type) to simplify the problem.For example changing the data type from double to int , make the problem smaller. Write algorithm for int data type and then generalize for double .

For example ,

Question is to trim the whitespaces in the password string or squeeze the string

Solve like this

1. left trim (remove whitespaces from the extreme left)
2. right trim (remove whitespaces from the extreme right)
3. remove whitespaces in between the string.

4. Base case and Build

This approach is most widely used in the recursive algorithm .

Solve the algorithm first for a base case (e.g., just one element). Then, try to solve it for elements one and two, assuming that we have the answer for element one. Then, try to solve it for elements one, two and three, assuming that we have the answer to elements one and two.

For Example ,

Question is to find all possible permuations of a string , assuming all the characters in the string is unique.

Input string : "abcdef"

Solving procedure :

Take one character at a time from the input string

Base Case

a(first element of the input string) : a(possible permutation)

Build

ab : ab,ba

abc : abc,acb,bac,bca,cba,cab

abcd...

abcde...

abcdef...

5. Data Structure Brainstorm

This is tiresome method and based on hit and trial method . Simply , run through a list of data structures and try to apply each one.

For Example ,

Question : Numbers are randomly generated and stored into an (expanding) array. How would we keep track of the median(odd number of elements : the middle number but if even number of elements then the average of the middle two elements)?

Data Structure Brainstorm:

1**. Linked list** : Linked lists do not perform very well with accessing and sorting numbers. Hence it is better to avoid it .

2. Array : Not sure, as we already have an array.It could be expensive to keep elements sorted in array . We will keep this option on hold and return to it if it's needed.

3. Binary tree : It can be a viable option, since binary trees do fairly well with ordering. As we know the top of the perfectly balanced binary search tree is median if there's an

odd number of elements present in the binary search tree. But if there's an even number of elements, the median is actually the average of the middle two elements. The middle two elements can't both be at the top. This is probably we need to look at, lets hold off to it.

4. Heap : A heap is really good at basic ordering and keeping track of max and mins. Suppose we had two heaps, we could keep track of the biggest half and the smallest half of the elements. The biggest half is kept in a min heap, such that the smallest element in the biggest half is at the root. The smallest half is kept in a max heap, such that the biggest element of the smallest half is at the root. Now, with these data structures, we have the potential median elements at the roots. If the heaps are no longer the same size, we can quickly "rebalance" the heaps by popping an element off the one heap and pushing it onto the other.

FLOW CHARTS

Flowcharting is the most popular and the oldest design tool system flowcharts detail the flow of data throughout an entire information system. Program flowcharts describe the processes taking place within an individual program in the system and the sequence in which they must be executed.

A flow chart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions.

The system flowchart is a graphic way of depicting all of the procedures that take input data and convert them to their final output form.

1. Shows the overall structure of the system.
2. Traces the flow of information and work.
3. Shows the physical media on which data are input, output and stored.
4. Highlights key processing and decision points.

System flowcharts can encompass different levels of details

 Quality Improvement Tool: Flow charts used specifically for a process.

 A flow chart is defined as a pictorial representation describing a process being studied or even used to plan stages of a project. Flow charts tend to provide people with a common language or reference point when dealing with a project or process.

 Four particular types of flow charts have proven useful when dealing with a process analysis: top-down flow chart, detailed flow chart, work flow diagrams, and a deployment chart. Each of the different types of flow charts tends to provide a different aspect to a process or a task. Flow charts provide an excellent form of documentation for a process, and quite often are useful when examining how various steps in a process work together.

 When dealing with a process flow chart, two separate stages of the process should be considered: the finished product and the making of the product. In order to analyze the finished product or how to operate the process, flow charts tend to use simple and easily recognizable symbols. The basic flow chart symbols below are used when analyzing how to operate a process.

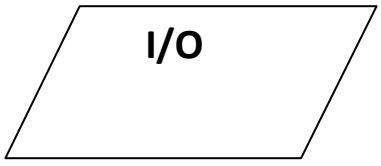
FLOWCHART SYMBOLS

In order to analyze the second condition for a flow process chart, one should use the ANSI standard symbols.

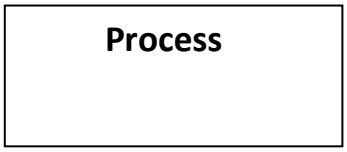
The American National Standard Institute (ANSI) has standardized the symbols:-

1. **TERMINAL** – The terminal symbol, as the name implies, is used to indicate the beginning (START), ending (STOP) and pauses (HALT) in the program logic flow. It is the first symbol and the last symbol in the program logic.

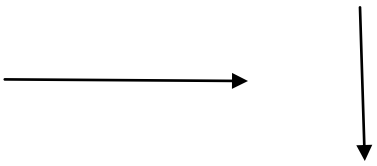
2. INPUT/OUTPUT – The I/O symbol is used to denote any function of an I/O device in the program.



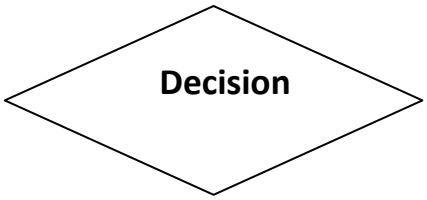
3. PROCESSING – A processing symbol is used in a flowchart to represent arithmetic and data movement instructions. Thus, all arithmetic processes of adding, subtracting, multiplying & dividing are shown by a processing symbol. The logical process of moving data from one location of the main memory to another is also denoted by this symbol.



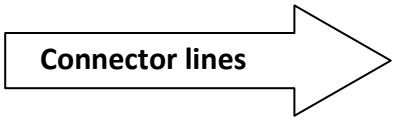
4. FLOW LINES – flow lines with arrowheads are used to indicate the flow of operation that is the exact sequence in which the instructions are to be executed. The normal flowchart is from top to bottom and left to right.



5. DECISION – the decision symbol is used in a flow chart to indicate a point at which a decision has to be made and a branch to one of two or more alternative points is possible.



6. CONNECTOR – if a flow chart becomes very long, the flow lines start cress crossing at many places that causes confusion and reduces understandability of the flowchart. Thus, whenever a flowchart becomes complex enough that the no and direction of flow lines is confusing or, it spreads over more than one page is useful to utilize the connector symbol as a substitute for flow lines.



Advantages of flowcharts :

The following benefits may be obtained when flowcharts are used for program planning:-

1. BETTER COMMUNICATION – the old saying that a picture is worth a thousand words holds true for flowcharts also.

2. EFFECTIVE ANALYSIS - A macro flowchart that charts the main line of logic of a software system becomes a system model, which can be broken down into detailed parts for study and further analysis of the system.

3. EFFECTIVE SYNTHESIS – Flowcharts are thus used as working models in the design of new programs and software system.

4. PROPER PROGRAM DOCUMENTATION – Program documentation involves collecting, organizing, storing and otherwise maintaining a complete historical record of programs.

5. EFFICIENT CODING – once a flowchart is ready programmers find it very easy to write the concerned program between the flowchart acts as a road map for them.

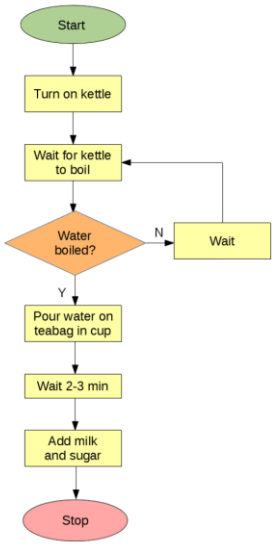
6. SYSTAMATIC DEBUGGING – Program errors are called bugs and the process of removing these errors is k/s DEBUGGING.

7. SYSTEMATIC TESTING – Testing is the process of confirming whether a program will successfully do all the jobs for which it has been designed under the specified constraints.

EXAMPLE

Process Flow Chart- Finding the best way home.

This is a simple case of processes and decisions in finding the best route home at the end of the working day.



(***Process Flow Chart- How a process works)***

Top down Design :

Top-down design specifies the solution to a problem in general terms and then divides the solution into finer and finer details until no more detail is necessary. Top-down design methodologies have had a profound impact on digital system design. They allow us to quickly and efficiently specify, design, synthesize and verify designs ready for fabrication. The key to these methodologies is synthesis, which relies on a mapping between the logical functions we use in a design and the physical circuits that realize the functions. Synthesis technology allows us to work at higher levels of abstraction and to delegate physical implementation details to automatic tools.

Characteristics of Top-Down Design :

- ***Top-down design can be perceived as visionary*.** Because top-down designs are not constrained by current work practice, they can turn out startlingly different and even futuristic designs, which shows how top-down design can be visionary.
- ***Top-down design is heavily driven by domain knowledge***. Designers need extensive domain knowledge to be able to abstract the nature of work in that domain. This usually translates to the need for designers to envision multiple work activity instances in that domain.
- ***Being a potential user in the domain is good for a designer***. An important factor in the success of Apple designers practicing top-down design is the fact that they could see themselves as users of the iPod and iPad, etc.

Debugging and testing of Programs :

Debugging :

In computer programming and software development, debugging is the process of finding and resolving bugs (defects or problems that prevent correct operation) within computer programs, software, or systems. *Debugging* is a cyclic activity involving execution testing and code correction.

Testing

Testing *means verifying correct behavior. Testing can be done at all stages of module development: requirements analysis, interface design, algorithm design, implementation, and integration with other modules. In the following, attention will be directed at implementation testing. Implementation testing is not restricted to execution testing. An implementation can also be tested using correctness proofs, code tracing, and peer reviews, as described below.*