SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- **A readable, dynamic, pleasant, flexible, fast and powerful language**

Recap:

- Notations (pseudocode, flow chart, programming language).
- Flowcharts are a graphical means of representing an algorithm
- Flowchart is a diagrammatic representation of sequence of logical steps of a program.
- A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.
- Programming languages generally consist of instructions for a computer.
- Eg : C, C++, COBAL, JAVA, Python ... Etc

1.6 Algorithmic problem solving:

- An algorithm is **a defined set of step-by-step procedures** that provides the correct answer to a particular problem.
- Algorithmic problem solving is solving problem that require the formulation of an algorithm for their solution.
- The formulation of algorithm is always been an important element of problem solving.
- We can consider algorithms to be procedural solutions to problems.

1.6 Algorithmic problem solving:

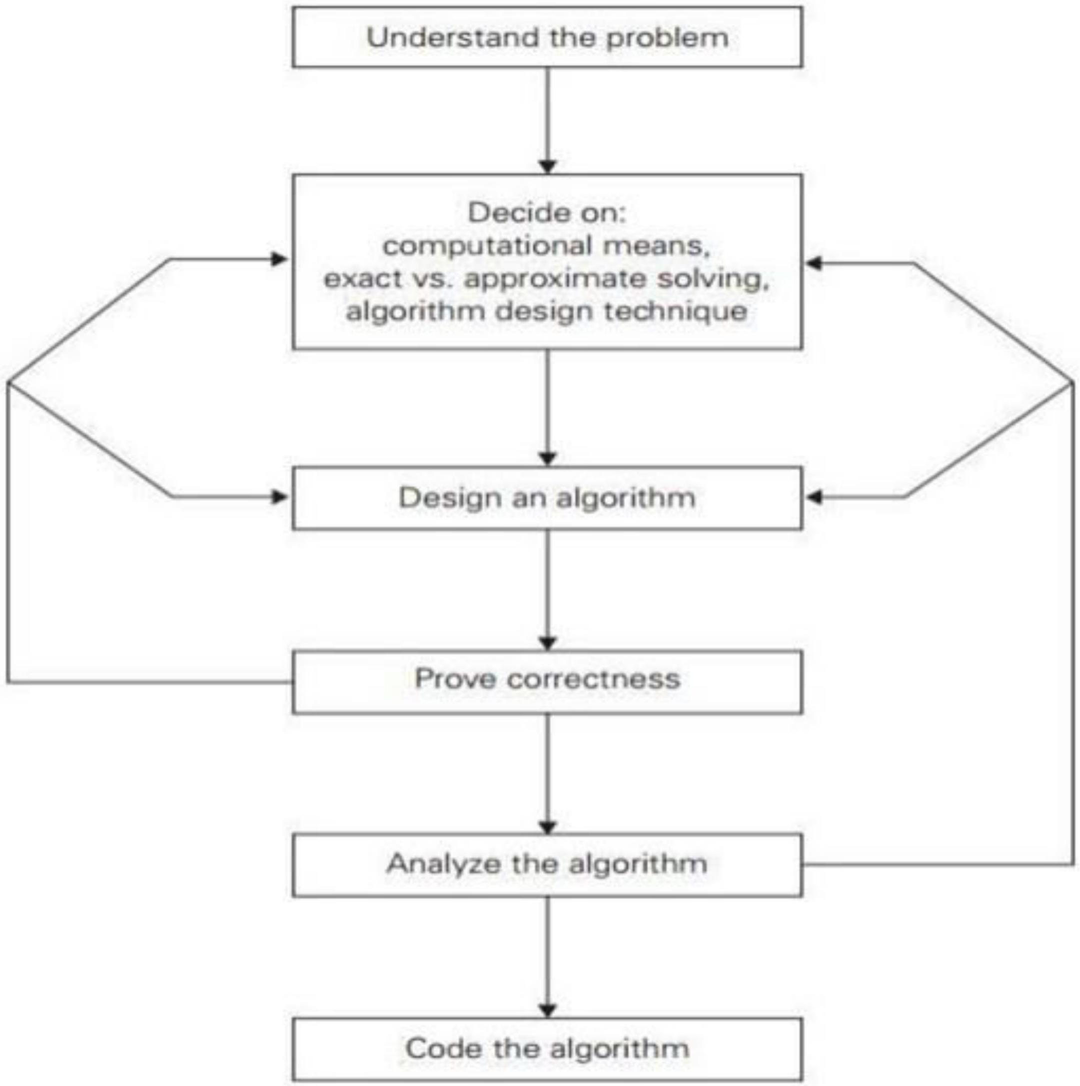

Figure 1: Algorithm design and analysis process

1.6 Algorithmic problem solving:

<u>The fundamental steps are:</u>

- Understanding the problem
- Ascertaining the capabilities of computational device
- Choose between exact and approximate problem solving
- Decide on appropriate data structures
- Algorithm design techniques
- Methods for specifying the algorithm
- Proving an algorithm's correctness
- Analyzing an algorithm
- Coding an algorithm

1.6 Algorithmic problem solving:

1. **Understanding the problem :**

- The first thing we need to do before designing an algorithm is **to understand completely the problem given**.

- **Read the problem's description carefully and ask questions if you have any doubts about the problem.**

- An input to an algorithm specifies an instance of the problem the algorithm solves.

1.6 Algorithmic problem solving:

1. **<u>Understanding the problem :</u>**

- It is very important to **specify exactly the range of instances** the algorithm needs to handle.

- **Correct algorithm** is not one that works most of the time, but **one that works correctly for all legitimate inputs.**

- Do not skip on this first step of algorithmic problem-solving process; if we do, then we need to do unnecessary rework on it.

1.6 Algorithmic problem solving:

2. Ascertain the capabilities of computational device :

- Once you completely understand a problem, you need to ascertain the capabilities of the computational device the algorithm is intended for.
- If the instructions are executed one after another, one operation at a time. Algorithms designed to be executed on such machines are called ***sequential algorithm.***
- If the instructions are executed concurrently, it is called ***parallel algorithm.***

1.6 Algorithmic problem solving:

3. Choose between exact and approximate problem solving :

- Next principal decision is to Choose between solving the problem exactly or solving the problem approximately.
- Case 1: solving the problem exactly – an algorithm is called ***exact algorithm***
- Case 2: solving the problem approximately – an algorithm is called ***approximation algorithm.***
- First, some important problems cannot be solved exactly for most of their instances; example – extracting square roots solving nonlinear equations.
- Second, available algorithm for solving a problem exactly can be unacceptably slow because of the problem's intrinsic complexity

1.6 Algorithmic problem solving:

<u>4. Decide on appropriate data structures :</u>

- Data structure plays a vital role in designing and analysis the algorithms.
- Some of the algorithm design techniques also depend on the structuring or restructuring data specifying a problem's instance.
- **Algorithm+ Data structure=programs.**

1.6 Algorithmic problem solving:

5. Algorithm Design Techniques:

- An ***algorithm design technique*** (or “strategy” or “paradigm”) is ***a general approach to solving problems algorithmically*** that is applicable to a variety of problems from different areas of computing.
- Learning these techniques is of atmost importance for the following reasons:
  - First, they provide guidance for designing algorithms for new problems, ex : problems for which there is no known satisfactory algorithm.
  - Second, algorithms are the cornerstone of computer science.
- Algorithm design techniques make it possible to classify algorithms according to an underlying design idea.

1.6 Algorithmic problem solving:

6. Methods of Specifying an Algorithm:

- Three ways to specify an algorithm
  - Pseudocode
  - Flowchart
  - Programming language

1.6 Algorithmic problem solving:

6. Methods of Specifying an Algorithm:..

6.1 Pseudocode :

- ***Pseudocode*** is a mixture of a natural language and programming language-like constructs.
- Pseudocode is usually more precise than natural language, and its usage often yields more concise algorithm descriptions.

1.6 Algorithmic problem solving:

6. Methods of Specifying an Algorithm:..

6.2 Flowchart:

- In the earlier days of computing, the dominant vehicle for specifying algorithms was a ***flowchart.***
- ***A Flow chart*** is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

1.6 Algorithmic problem solving:

6. Methods of Specifying an Algorithm:..

6.3 Programming language:

- A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.
- Programming languages generally consist of ***instructions for a computer***.
- Programming languages can be used to create programs that implement specific algorithms.
- Eg : C, C++, COBAL, JAVA, Python ... Etc

1.6 Algorithmic problem solving:

7. Proving an Algorithm's correctness:

- Once the algorithm has been specified, then its ***correctness*** must be proved.
- An algorithm must yield a required result for every legitimate input in a finite amount of time.
- For some algorithm, a proof of correctness is quite easy; for others, it can be quite complex.

1.6 Algorithmic problem solving:

7. Proving an Algorithm’s correctness:..

- A common technique for proving correctness is to ***use mathematical induction*** because an algorithm’s iterations provide a natural sequence of steps needed for such proofs.

- The notion of correctness for approximation algorithm is less straightforward than it is for exact algorithms.

1.6 Algorithmic problem solving:

8. Analyzing an Algorithm:

- Our algorithms need to possess several qualities. After correctness, the most important one is efficiency.

- There are two kind of algorithm efficiency: i) **Time efficiency** ii) **Space efficiency**

- Time efficiency: Indicates **how fast the algorithm runs**.

1.6 Algorithmic problem solving:

8. Analyzing an Algorithm:..

- Space efficiency: indicates **how much extra memory** the algorithm needs.
- Another desirable characteristic's of an algorithm are ***simplicity and generality.***
- If you are not satisfied with the algorithm's ***efficiency, simplicity, or generality,*** you must return to the drawing board and redesign the algorithm.

1.6 Algorithmic problem solving:

9. Coding an Algorithm:

- Most algorithms are destined to be ultimately implemented as computer programs.

- The coding / implementation of an algorithm is done by a suitable programming language like C, C++, JAVA

- It is very essential to write an optimized code (efficient code) to reduce the burden of compiler.

1.6 Algorithmic problem solving:

- As a rule a good algorithm is a result of repeated effort and rework.

- Even if you have been fortunate enough to get an algorithmic idea that seems perfect, you should still try to see whether it can be improved.

1.6 Algorithmic problem solving:

- An important issue of algorithmic problem solving is the question of whether or not every problem can be solved by an algorithm.

- Fortunately, a vast majority of problems in practical computing can be solved by an algorithm.

Summary:

- An algorithm is a sequence of non ambiguous instructions for solving a problem in a finite amount of time.
- An input to an algorithm specifies an instance of the problem the algorithm solves.
- Algorithm can be specified in a natural language or a pseudocode; they can also be implemented as computer programs.

Summary:

- Algorithm design techniques are ***general approaches to solving problems algorithmically***, applicable to a verity of problems from different areas of computing.
- The same problem can often be solved by several algorithms.
- Algorithms operate on **data**. This makes the issue of data structuring critical for efficient algorithmic problem solving.


THANK YOU