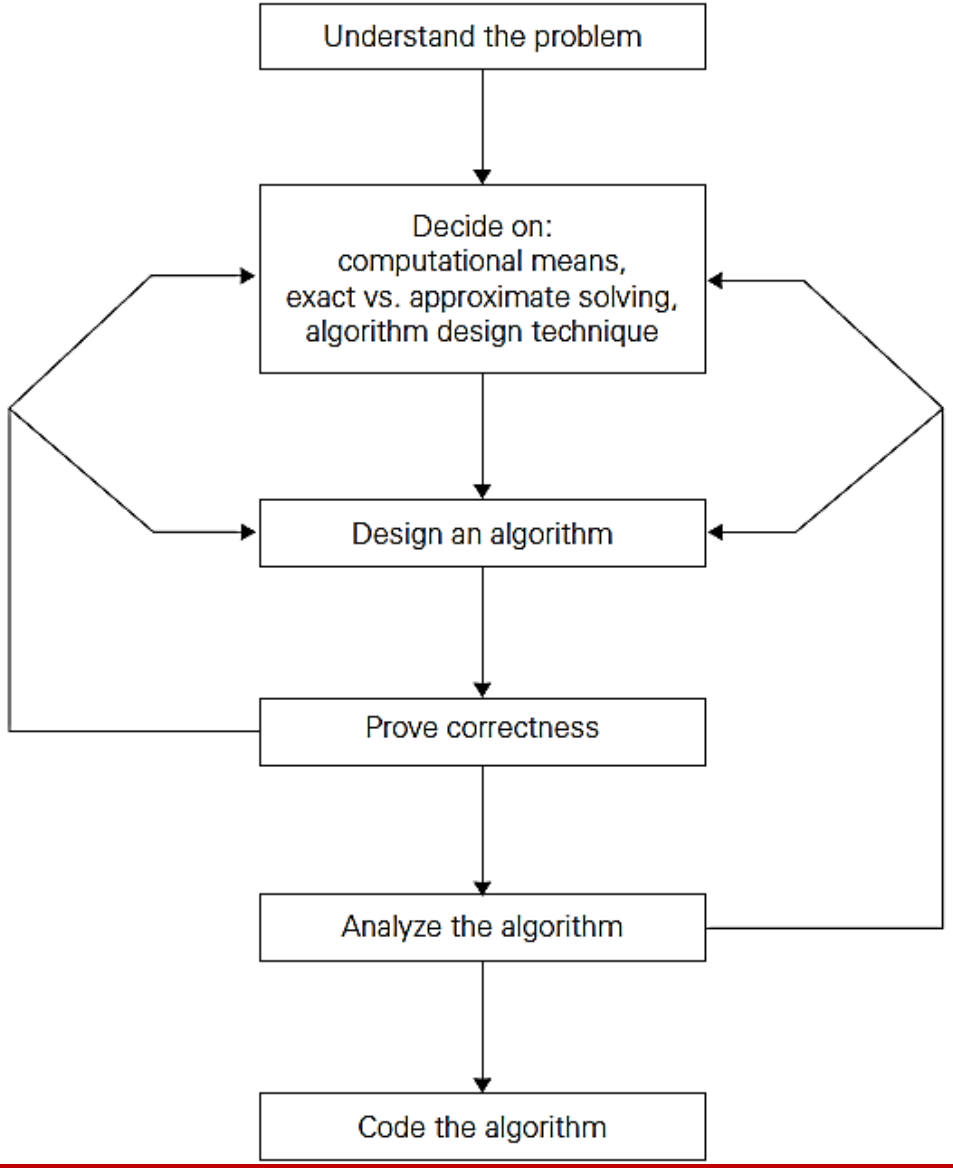
UNIT I INTRODUCTION:

Introduction to Algorithms – Fundamentals of Algorithmic Problem Solving – Fundamentals of the Analysis of Algorithmic Efficiency – Analysis Framework – Asymptotic Notations and Basic Efficiency Classes – Mathematical Analysis of Recursive Algorithms – Mathematical Analysis of Non-recursive Algorithms

Fundamentals of Algorithmic Problem


Understand the problem
Decide on:
computational means,
exact vs. approximate solving,
algorithm design technique
Design an algorithm
Prove correctness
Analyze the algorithm
Code the algorithm

Fundamentals of Algorithmic problem solving

Understanding the problem:

- The problem given should be understood completely.
- Check if it is similar to some standard problems & if a Known algorithm exists. otherwise a new algorithm has to be devised.
- Creating an algorithm is an art which may never be fully automated.
- An important step in the design is to specify an in- stance of the problem.

Ascertain the capabilities of the computational device:

- Once a problem is understood we need to Know the capabilities of the computing device this can be done by Knowing the type of the architecture, speed & memory availability.

Exact /approximate solution

- Once algorithm is devised, it is necessary to show that it computes answer for all the possible legal inputs
- The solution is stated in two forms, Exact solution or approximate solution
- examples of problems where an exact solution cannot be obtained are
  - Finding a square root of number
  - Solutions of non linear equations

Fundamentals of Algorithmic problem solving

Algorithm design techniques

Creating an algorithm is an art which may never be fully automated. By mastering these design strategies, it will become easier for you to devise new and useful algorithms.

Dynamic programming is one such technique.

Some of the techniques are especially useful in fields other then computer science such as operation research and electric- al engineering.

Some important design techniques are linear, non linear and integer programming

Decide on the appropriate data structure

- Some algorithms do not demand any in- genuity in representing their inputs.
- Some others are in fact are predicted on ingenious data structures.
- A data type is a well-defined collection of data with a well-defined set of operations on it.
- A data structure is an actual implementation of a particular abstract data type.
- The Elementary Data Structures are Arrays.

Decide on the appropriate data structure

The Elementary Data Structures are Arrays.

- These let you access lots of data fast. **(good)**
- You can have arrays of any other data type. **(good)**
- However, you cannot make arrays bigger if your program decides it needs more space. **(bad)**
- Built-in sets are limited to a certain small size. (**bad**, but we can build our own set data type out of arrays to solve this problem if necessary)

Methods of specifying an algorithm:

- There are mainly two options for specifying an algorithm: use of **natural language** or **pseudocode** & **Flowcharts.**
- A Pseudo code is a mixture of natural language & programming language like constructs.
- A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes.

Proving an algorithms correctness:

- Once algorithm is devised, it is necessary to show that it computes answer for all the possible legal inputs .
- The process of validation is to assure us that this algorithm will work correctly independent of issues concerning programming language it will be written in.
- A proof of correctness requires that the solution be stated in two forms.

- One form is usually as a program which is annotated by a set of assertions about the input and output variables of a program. These assertions are often expressed in the predicate calculus.
- The second form is called a specification, and this may also be expressed in the predicate calculus.
- A proof consists of showing that these two forms are equivalent in that for every given legal input, they describe same out- put.
- A complete proof of program correctness requires that each statement of programming language be precisely defined and all basic operations be proved correct.

Analyzing algorithms:

- As an algorithm is executed, it uses the computers central processing unit to perform operation and its memory (both immediate and auxiliary) to hold the program and data.
- Analysis of algorithms and performance analysis refers to the task of determining how much computing time and storage an algorithm requires.
- An important result of this study is that it allows you to make quantitative judgments about the value of one algorithm over another.

Analyzing algorithms:

- In fact, there are two kinds of algorithm efficiency
- time efficiency
  - indicating how fast the algorithm runs
- space efficiency
  - indicating how much extra memory it uses

Coding an Algorithm

- Most algorithms are destined to be ultimately implemented as **computer programs**.
- Programming an algorithm presents both a **peril** and an **opportunity**.
- The peril lies in the possibility of making the **transition** from an **algorithm** to a **program** either incorrectly or very inefficiently.
- A working program provides an opportunity in allowing an **empirical analysis** of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analyzing the results obtained.


Thank You