DESIGN & ANALYSIS OF ALGORITHMS (21IS402)

Mr. Deepak D

Assistant Professor

Department of Information Science and Engineering

NMAM Institute of Technology,

NITTE (Deemed to be University),

Nitte, Karkala - 574110

Course Learning Objectives

1. Understand the notion of algorithms, Algorithm design and analysis process, and asymptotic notations, and Analyze the non-recursive and recursive algorithms and represent the efficiency of these algorithms in terms of the standard asymptotic notations.

2. Devise the Brute Force and Divide and Conquer techniques to design the algorithms and apply these methods in designing algorithms to solve a given problem.

3. Apply the Decrease and Conquer, Transform and Conquer algorithm design techniques to solve a given problem

4. Get an idea of Time versus Space Trade-offs and Apply and Analyse dynamic programming methods in designing algorithms to solve a given problem.

5. Describe and illustrate the idea of the Greedy method, Backtracking, and Branch and Bound algorithm design techniques to solve a given problem and to describe P, NP, and NP-Complete problems

CONTENT

UNIT -I

1. INTRODUCTION
2. FUNDAMENTALS OF THE ALGORITHMS EFFICIENCY
3. BRUTE FORCE
4. DIVIDE AND CONQUER

UNIT – II

1. DECREASE & CONQUER
2. TRANSFORM AND CONQUER
3. TIME AND SPACE TRADEOFFS
4. DYNAMIC PROGRAMMING

UNIT – III

1. GREEDY TECHNIQUE
2. BACKTRACKING
3. BRANCH AND BOUND

REFERENCES

Text Books

- Anany Levitin, “Introduction to the Design & Analysis of Algorithms”, 2nd Edition, Pearson Education, 2011.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “Introduction to Algorithms”, 3rd Edition, PHI, 2014

CONTENT

UNIT -1

1. Introduction
   - What is an Algorithm?
   - Fundamentals of Algorithmic Problem Solving

Why study Algorithms ?

Theoretical importance

- It is the core of computer science.

Practical importance

- To know a standard set of important algorithms from different areas of computing
- Should be able to design new algorithms and analyze their efficiency

Usefulness in developing analytical skills

- Algorithms can be seen as special kinds of solutions to problems
- Specific algorithm design techniques can be interpreted as problem-solving strategies

1.1 - What is an Algorithm ?

What is an 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.

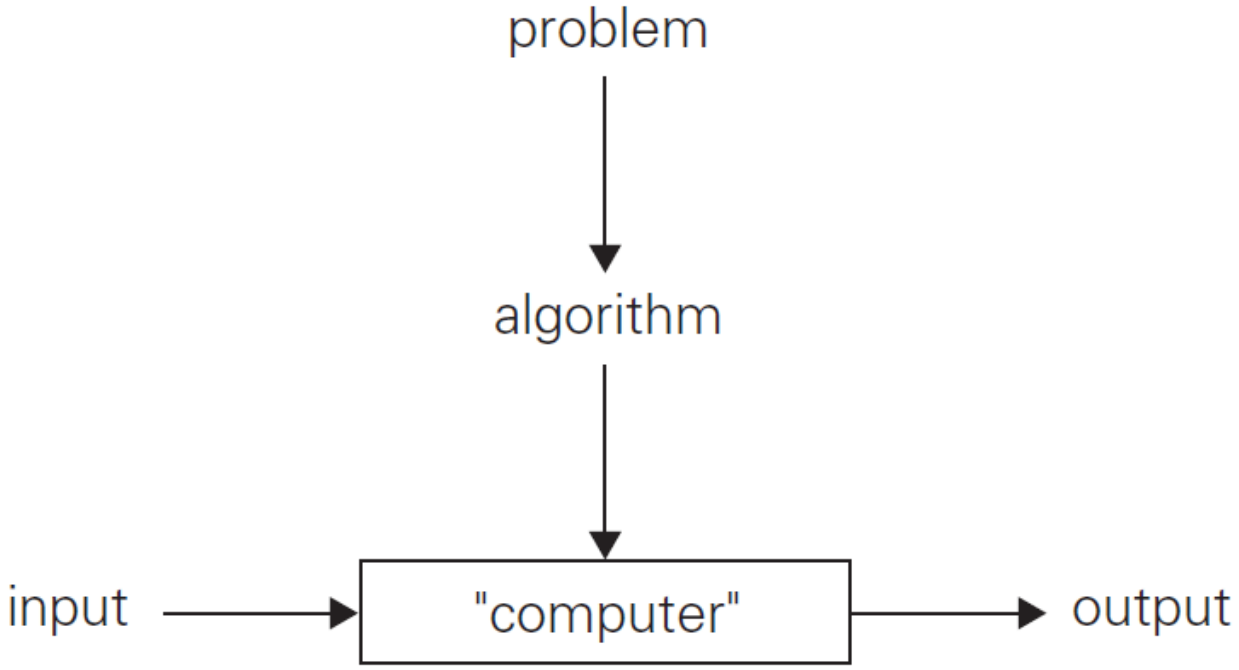


Fig: The notion of the algorithm.

1.1 - What is an Algorithm ?

All algorithms must satisfy the following criteria:

1. **Input:** Zero or more quantities are externally supplied.
2. **Output:** At least one quantity is produced.
3. **Definiteness:** Each instruction is clear and unambiguous.
4. **Finiteness:** If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. **Effectiveness:** Every instruction must be very basic so that it can be carried out by a person using a only pencil and paper. It is not enough that each operation is definite as in criterion 3 but it also must be feasible (possible to do easily or conveniently.)

1.1 - What is an Algorithm ?

Methods for solving the problem: Computing the greatest common divisor of two integers.

The greatest common divisor of two nonnegative, not-both-zero integers m and n, denoted

gcd (m, n)

is defined as the largest integer that divides both m and n evenly, i.e., with a remainder of zero.

Example:

	15	33
Divisors	1, 3, 5, 15	1, 3, 11, 33
Common Divisors	1, 3	
Greatest Common Divisors (GCD)	3	

GCD(15,33) = 3

1.1 - What is an Algorithm ?

1. Euclid's algorithm

Euclid of Alexandria (third century B.C.) outlined an algorithm for solving the GCD problem

Euclid's algorithm is based on applying repeatedly the equality

gcd(m, n) = gcd(n, m mod n)

Where, *m mod n* is the remainder of the division of *m by n*, until *m mod n* is equal to 0.

Since gcd(m, 0) = m, the last value of m is also the greatest common divisor of the initial m and n.

Example:

GCD (33,12) = 3

m	n	r
33	12	9
12	9	3
9	3	0
3	0	-

1.1 - What is an Algorithm ?

Structured description of an algorithm

Euclid's algorithm for computing gcd*(m, n)*

Step 1: If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.

Step 2: Divide m by n and assign the value of the remainder to r.

Step 3: Assign the value of n to m and the value of r to n. Go to Step 1.

Algorithm in pseudocode

ALGORITHM *Euclid(m, n)*

//Computes gcd*(m, n)* by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

while $n \neq 0$ **do**

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

1.1 - What is an Algorithm ?

2. Consecutive integer checking algorithm

Step 1: Assign the value of min{*m*, *n*} to *t*.

Step 2: Divide *m* by *t*. If the remainder of this division is 0, go to Step 3; otherwise, go to Step 4.

Step 3: Divide *n* by *t*. If the remainder of this division is 0, return the value of *t* as the answer and stop; otherwise, proceed to Step 4.

Step 4: Decrease the value of *t* by 1. Go to Step 2

1. **t ← min (m ,n)**
2. **if m % t = 0 goto 3,**
   else goto 4
3. **if n % t = 0 return t,**
   else goto 4
4. **t ← t – 1**
5. **goto 2**

1.1 - What is an Algorithm ?

3. Middle-school procedure

Step 1 Find the prime factors of *m*.

Step 2 Find the prime factors of *n*.

Step 3 Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If *p* is a common factor occurring p_m and p_n times in *m* and *n*, respectively, it should be repeated $\min\{p_m, p_n\}$ times.)

Step 4 Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

Example: GCD (60, 24)

60 = 2 . 2 . 3 . 5

24 = 2 . 2 . 2 . 3

gcd(60, 24) = 2 . 2 . 3 = 12.

1.1 - What is an Algorithm ?

Sieve of Eratosthenes:

- The algorithm starts by initializing a list of prime candidates with consecutive integers from 2 to *n*.
- In the first iteration, the algorithm eliminates from the list all multiples of 2, i.e., 4, 6, and so on.
- Then it moves to the next item on the list, which is 3, and eliminates its multiples
- No pass for number 4 is needed: since 4 itself and all its multiples are also multiples of 2, they were already eliminated on a previous pass.
- The next remaining number on the list, which is used on the third pass, is 5. The algorithm continues in this fashion until no more numbers can be eliminated from the list. The remaining integers of the list are the primes needed.

Example: Consider the application of the algorithm to find the list of primes not exceeding n = 25

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	**3**		5		7				11		13				17		19				23		25
2	3		**5**		7				11		13				17		19				23		

1.1 - What is an Algorithm ?

Sieve of Eratosthenes:

Algorithm for generating consecutive primes not exceeding any given integer where n > 1.

ALGORITHM *Sieve*(n)

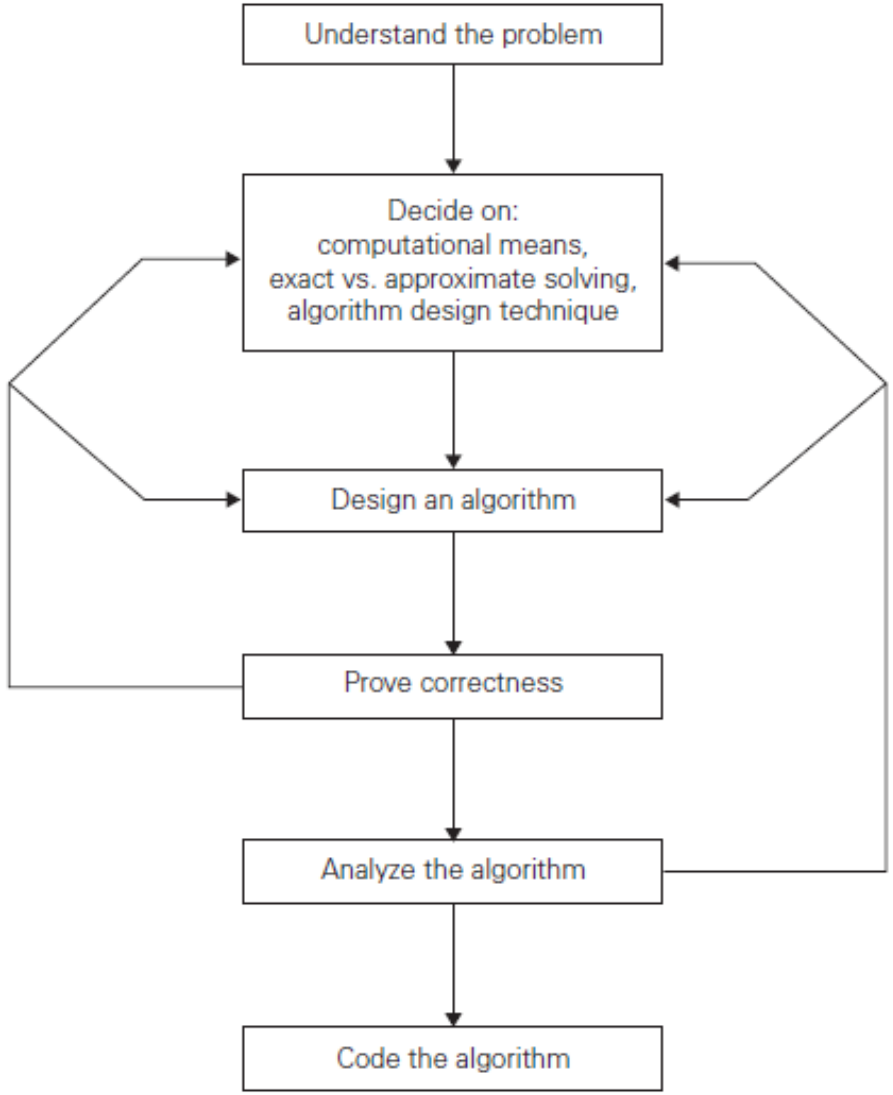
```
//Implements the sieve of Eratosthenes
//Input: A positive integer n > 1
//Output: Array L of all prime numbers less than or equal to n

for p ← 2 to n do A[p] ← p
for p ← 2 to ⌊√n⌋ do     //see note before pseudocode
    if A[p] ≠ 0          //p hasn't been eliminated on previous passes
        j ← p * p
        while j ≤ n do
            A[j] ← 0     //mark element as eliminated
            j ← j + p
//copy the remaining elements of A to array L of the primes
i ← 0
for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i + 1
return L
```

1.2-Fundamentals of Algorithmic Problem Solving

The sequence of steps in designing and analyzing an algorithm

1. Understanding the Problem
2.1 Ascertaining the Capabilities of the Computational Device
2.2 Choosing between Exact and Approximate Problem Solving
2.3 Deciding on Appropriate Data Structures
2.4 Algorithm Design Techniques
3. Methods of Specifying an Algorithm
4. Proving an Algorithm's Correctness
5. Analyzing an Algorithm
6. Coding an Algorithm



1.2-Fundamentals of Algorithmic Problem Solving

1. Understanding the Problem

- The First thing that needs 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, do a few small examples by hand, think about special cases, and ask questions again if needed.
- Based on the types of problems that arise in computing applications, you might be able to use a known algorithm for solving them. It helps to understand how such an algorithm works and to know its strengths and weaknesses, especially if you have to choose among several available algorithms.

1.2-Fundamentals of Algorithmic Problem Solving

2. The Decision making is done on the following:

(a) Ascertaining the Capabilities of the Computational Device

- In a random-access machine (RAM), instructions are executed one after another (The central assumption is that one operation at a time). Accordingly, algorithms designed to be executed on such machines are called sequential algorithms.
- In some newer computers, operations are executed concurrently, i.e., in parallel. Algorithms that take advantage of this capability are called parallel algorithms.
- The choice of computational devices like Processors and memory is mainly based on space and time efficiency

(b) Choosing between Exact and Approximate Problem Solving

- The next principal decision is to choose between solving the problem ***exactly*** or ***solving it approximately***.
- An algorithm used to solve the problem exactly and produce the correct result is called an ***exact algorithm***.
- If the problem is so complex and not able to get the exact solution, then we have to choose an algorithm called an ***approximation algorithm.*** i.e., produces an approximate answer. E.g., extracting square roots, solving nonlinear equations, and evaluating definite integrals

1.2-Fundamentals of Algorithmic Problem Solving

2. The Decision making is done on the following:

(a) Ascertaining the Capabilities of the Computational Device

- In a random-access machine (RAM), instructions are executed one after another (The central assumption is that one operation at a time). Accordingly, algorithms designed to be executed on such machines are called sequential algorithms.
- In some newer computers, operations are executed concurrently, i.e., in parallel. Algorithms that take advantage of this capability are called parallel algorithms.
- The choice of computational devices like Processors and memory is mainly based on space and time efficiency

(b) Choosing between Exact and Approximate Problem Solving

- The next principal decision is to choose between solving the problem ***exactly*** or ***solving it approximately***.
- An algorithm used to solve the problem exactly and produce the correct result is called an ***exact algorithm***.
- If the problem is so complex and not able to get the exact solution, then we have to choose an algorithm called an ***approximation algorithm.*** i.e., produces an approximate answer. E.g., extracting square roots, solving nonlinear equations, and evaluating definite integrals

1.2-Fundamentals of Algorithmic Problem Solving

(C) Deciding on the Appropriate Data Structures

- Close attention is required while choosing data structures appropriate for the operations performed by the algorithm.
- The algorithm design techniques depend on structuring and restructuring data specifying a problem's instance.

Algorithms+ Data Structures = Programs

- Though Algorithms and Data Structures are independent, they are combined to develop a program. Hence the choice of proper data structure is required before designing the algorithm.
- Implementation of algorithm is possible only with the help of Algorithms and Data Structures

(d) Algorithm Design Techniques

- An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Algorithmic technique is a general approach by which many problems can be solved algorithmically. E.g., Brute Force, Divide and Conquer, Dynamic Programming, Greedy Technique, and so on.
- Algorithm Design Techniques are important because they provide guidance for designing algorithms for new problems and it possible to classify algorithms according to an underlying design idea;

1.2-Fundamentals of Algorithmic Problem Solving

3. Methods of Specifying an Algorithm

There are three ways to specify an algorithm. They are:

- Natural language
- Pseudocode
- Flowchart

1. Natural language:

- It is very simple and easy to specify an algorithm using natural language. But many times specification of the algorithm using natural language is not clear and we get brief specifications.

2. Pseudocode

- Pseudocode is a mixture of a natural language and programming language constructs. Pseudocode is usually more precise than natural language.
- For the Assignment operation left arrow "←", for comments two slashes "//",if condition, for, while loops are used.

1.2-Fundamentals of Algorithmic Problem Solving

3. Flow Chart

- In the earlier days of computing, the dominant method for specifying algorithms was a ***flowchart***, this representation technique has proved to be inconvenient.

- ***Flowchart*** is a graphical representation of an algorithm. It expresses an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

1.2-Fundamentals of Algorithmic Problem Solving

4. Proving an Algorithm's Correctness

- Once an 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 example, the correctness of Euclid's algorithm for computing the greatest common divisor stems from the correctness of the equality ***gcd(m, n) = gcd(n, m mod n).***
- 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 algorithms is less straightforward than it is for exact algorithms. The error produced by the algorithm should not exceed a predefined limit.

1.2-Fundamentals of Algorithmic Problem Solving

5. Analyzing an Algorithm

For an algorithm the most important is efficiency. The efficiency of an algorithm is determined by measuring both time efficiency and space efficiency.

- Time efficiency- indicates how fast the algorithm runs
- Space efficiency- indicates how much extra memory it uses.

Other characteristics of an algorithm are:

- The simplicity of an algorithm
- The generality of an algorithm

1.2-Fundamentals of Algorithmic Problem Solving

6. Coding an Algorithm

- The coding/implementation of an algorithm is done by a suitable programming language like C, C++, or JAVA.
- The transition from an algorithm to a program can be done either incorrectly or very inefficiently. Implementing an algorithm correctly is necessary. The Algorithm's power should not be reduced by the inefficient implementation.
- Standard tricks like computing a loop's invariant (an expression that does not change its value) outside the loop, collecting common subexpressions, replacing expensive operations with cheap ones, selecting programming language, and so on should be known to the programmer.
- Typically, such improvements can speed up a program only by a constant factor, whereas a better algorithm can make a difference in running time by orders of magnitude. But once an algorithm is selected, a 10–50% speedup may be worth an effort.
- It is very essential to write an optimized code (efficient code) to reduce the burden of the compiler

REFERENCE

TEXT BOOK

1. Anany Levitin, “Introduction to the Design & Analysis of Algorithms”, 2nd Edition, Pearson Education, 2011