Method-3 : Middle School Procedure ; GCD(m,n)

Step-1: Find the prime factors of m

Step-2 : Find the prime factors of n

Step-3 : Identify all common factors ~~of~~ found in Steps-1 and 2 . If 'p' is a common factor which occurs p_m in m and p_n in n respectively, then it should be repeated $\min(p_m, p_n)$ times.

Step-4 : Compute the product of all common factors and return it as the GCD.

eg) GCD(60, 24)

$$60 = \underline{2} \times \underline{2} \times \underline{3} \times 5$$

$$24 = \underline{2} \times \underline{3} \times \underline{2} \times 2$$

$$GCD = 2 \times 2 \times 3 = \underline{\underline{12}}$$

Design an algorithm to generate a list of consecutive prime numbers less than or equal to 'n'.

let 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

Iteration-1

Eliminates all multiples of 2 →

2 3 5 7 9 11 13 15 17 19 21 23 25

Iteration-2

Eliminates all multiples of 3

2 3 5 7 11 13 17 19 23 25

Iteration-3

Eliminates all multiples of 5

2 3 5 7 11 13 17 19 23

STOP when we cannot eliminate any more numbers.

$$p \cdot p \leq n$$

$$p^2 \leq n$$

$$p \leq \lfloor \sqrt{n} \rfloor$$

SIEVE OF ERATOSTHENES:-

ALGORITHM Sieve (n)

//Input : A positive number, $n > 1$

// 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

        if A[p] ≠ 0

            j ← p * p

            while j <= n do

                A[j] ← 0
                j ← j + p

i ← 0

for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i+1

return L
```

Fundamentals of Algorithmic Problem Solving :-

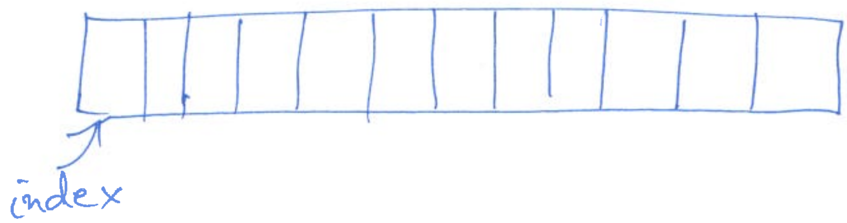
1. Understanding the problem
2. Ascertain the capabilities of the computational device :
   - RAM
   - speed & space
3. Choose between exact vs. approximate problem solving.
4. Design the algorithm & choose the appropriate data structure.
5. Prove the ~~accuracy~~ correctness of the algorithm.
6. Analyze the algorithm.
   - Time Efficiency
   - Space Efficiency
7. Coding the algorithm.

Problem Types

1. Searching
2. Sorting
3. Geometric
4. Combinatorial
5. Graph

Fundamental Data Structures

① Array

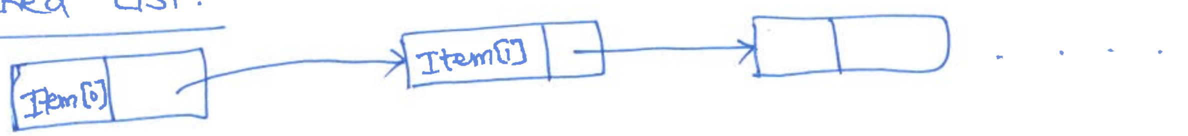


Advantages

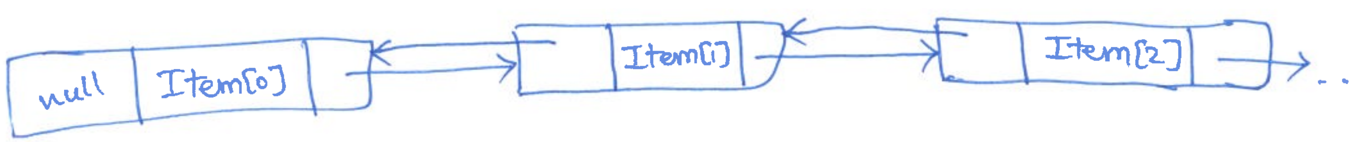
- same access time for any array element.
- stored contiguously in memory & values are accessed by its index

② Linked List

Single-linked List:



Double linked list:



Advantages

- → no need to allocate contiguous memory like that of an array.
- → Memory is allocated as values get added to a linked list.
- → easy to add / delete elements.

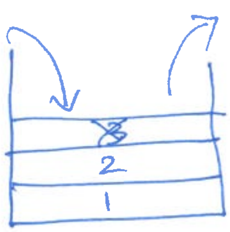
Disadvantages

- → Searching through a linked list is slower as it always begins at the head.

③ Stack

LIFO

Last-in-First-out order



Push(1)

Push(2)

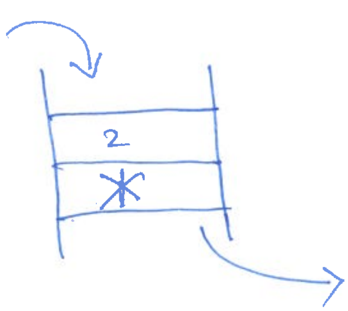
Push(3)

Pop() → deletes "3"

④ Queue

FIFO

First-in-First-out order



Enqueue(1)

Enqueue(2)

Dequeue() ⇒ deletes "1"

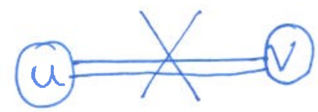
Graphs

$$G = \langle V, E \rangle$$

$V \rightarrow$ vertices

$E \rightarrow$ edges

Undirected Graph

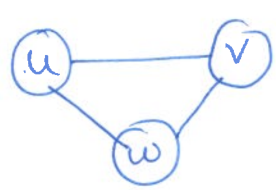


Loop

$\rightarrow$ Edges that connect vertices to themselves.

For an undirected graph, with no loops,

$$\boxed{0 \le |E| \le |V|(|V|-1)/2}$$



Sparse Graph

A graph with ~~a~~ relatively few edges when compared to the number of vertices.

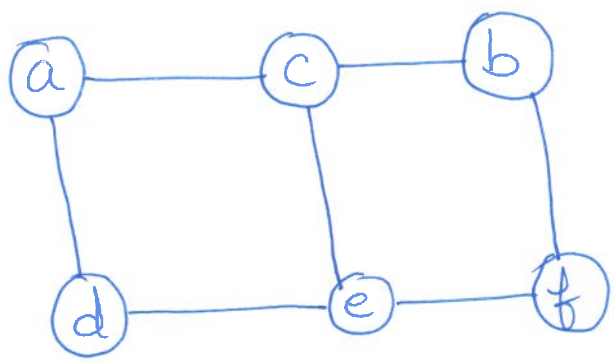
Dense Graph

A graph with only a few missing edges.

Complete Graph

A graph with every pair of vertices connected by an edge.

Graph Representation



Adjacency Matrix

$$
\begin{array}{c}
\\ a \\ b \\ c \\ d \\ e \\ f
\end{array}
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\left[\begin{array}{cccccc}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0
\end{array}\right]
\end{array}
$$