K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Course: Data Structure

(Course Code : ENCS205)

UNIT-1: Foundations of Data Structures

School of Engineering & Technology
K.R. Mangalam University

Unit-1 Foundations of Data Structures

SESSION 1	**Introduction to Data Structures**
SESSION 2	Abstract Data Types (ADTs)
SESSION 3	Static & Dynamic Implementations
SESSION 4	Algorithm Efficiency Basics
SESSION 5	Asymptotic Notations (Big-O, Theta, Omega)
SESSION 6	Recurrence Relation
SESSION 7	Analysis of Iterative and Recurrence Relations
SESSION 8	Practice
SESSION 9	Array Data Structure Basics

Unit-1 Foundation of Data Structure

SESSION 10 Array Operations (Insertion and Deletion)

SESSION 11 Multidimensional Arrays

SESSION 12 Representation of Arrays: Row-Major Order and Column-Major Order

SESSION 13 Applications of Arrays and Introduction to Sparse Matrices

SESSION 14 Sparse Matrix Representation and Operations

SESSION 15 Revision

Applications and Use cases of Data Structure (Cont..)

- **Array (Data Structure)- Book Titles in Library Management System**

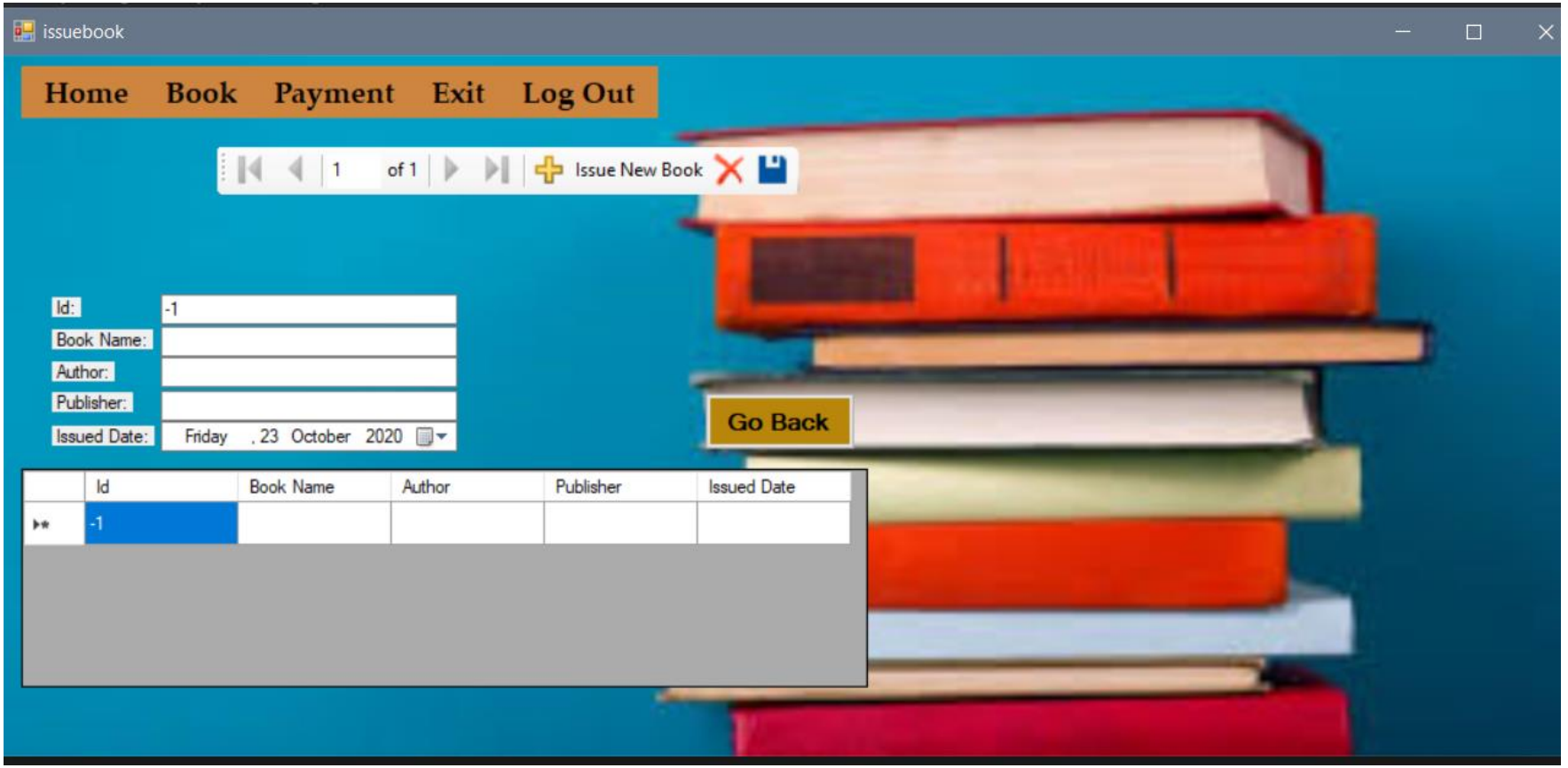


Fig.1 : Book Titles in Library Management System

Applications and Use cases of Data Structure (Cont..)

- **Queue (Data Structure)- Online Ticket Booking, In Escalators, vehicles at toll center**

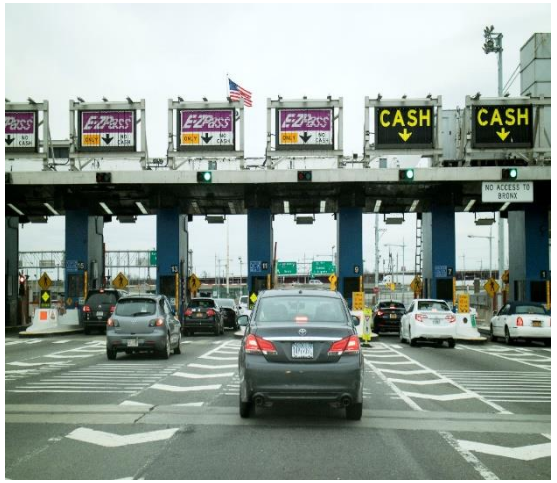



Fig.2 : Examples of Queue Data Structure

Applications and Use cases of Data Structure (Cont..)

- **Associative Array (Data Structure)- Contacts in a Phone**

Fig. 3 : Example of Associative Array

Applications and Use cases of Data Structure (Cont..)

- **Linked List (Data Structure)- Accessing pages using previous and next button**

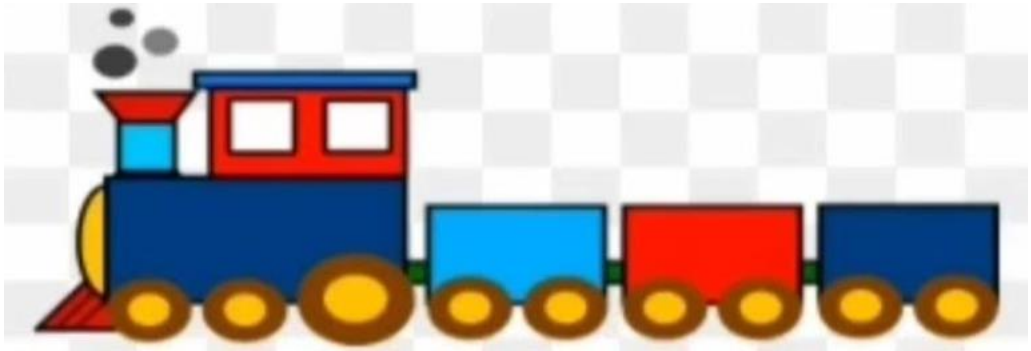


Fig. 4 : Examples of Linked List

Applications and Use cases of Data Structure (Cont..)

- **Stack (Data Structure) - First Hired Last Fired, Wearing/Removing Bangles, Pile of Dinner Plates, Stacked chairs- Stack (Data Structure)**

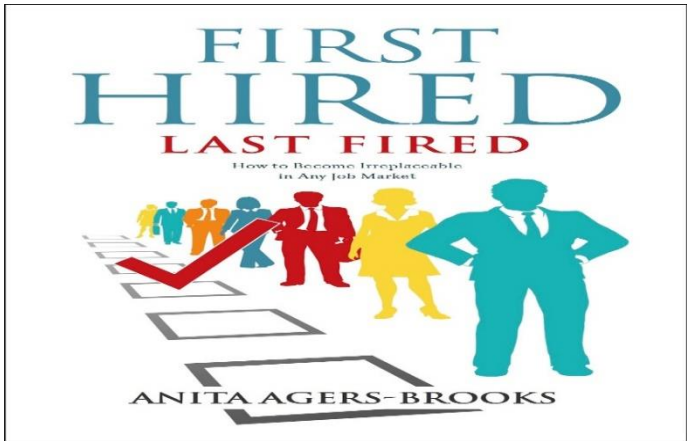


Fig. 5 : Examples of Stack Data Structure

Applications and Use cases of Data Structure (Cont..)

- **Graph (Data Structure)- Google's knowledge graph and GPS Navigation System**

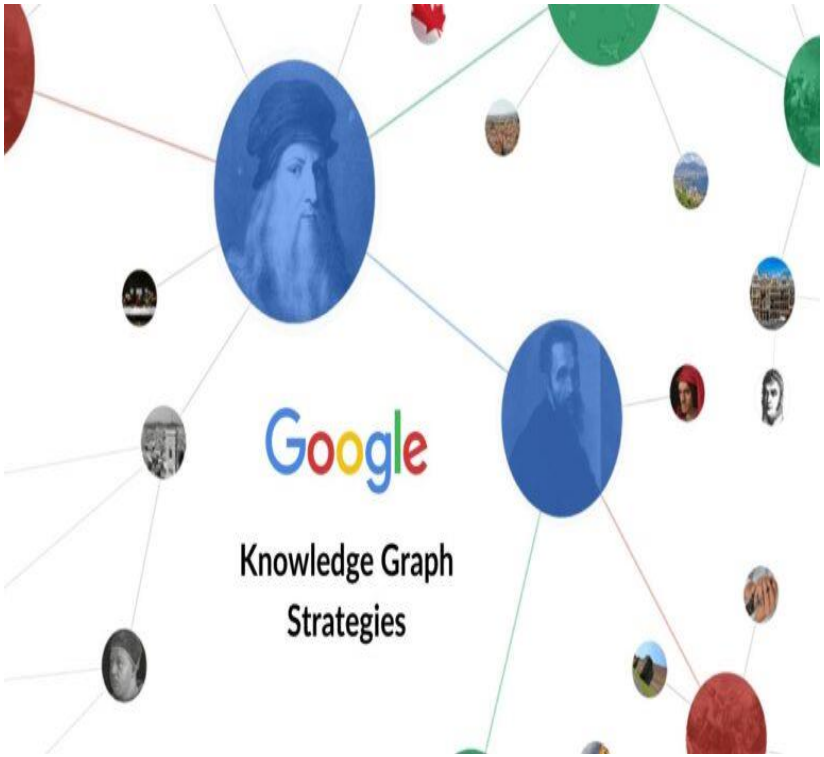
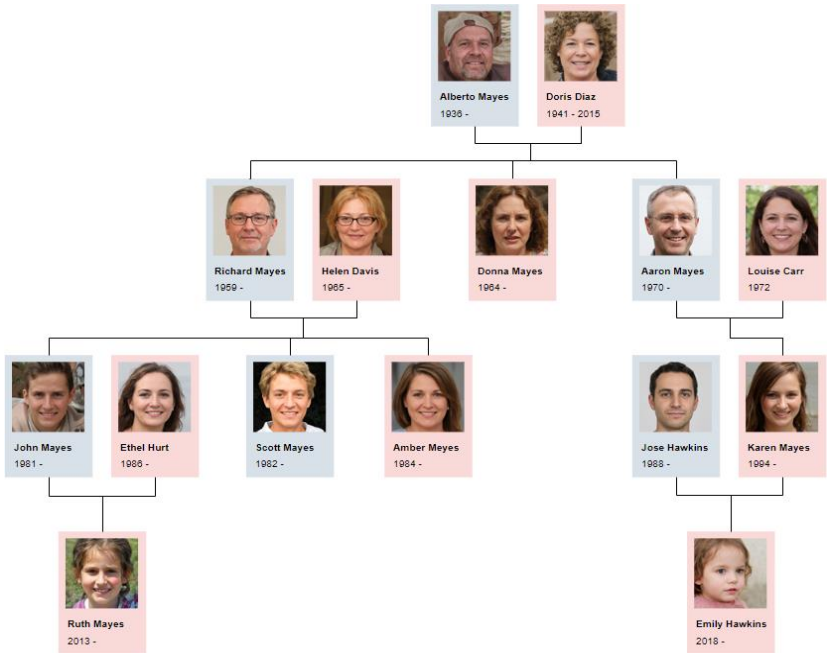


Fig. 6 : Examples of Graph Data Structure

Applications and Use cases of Data Structure (Cont..)

- **Tree (Data Structure)- Family Tree Structure and Domain Name Server**



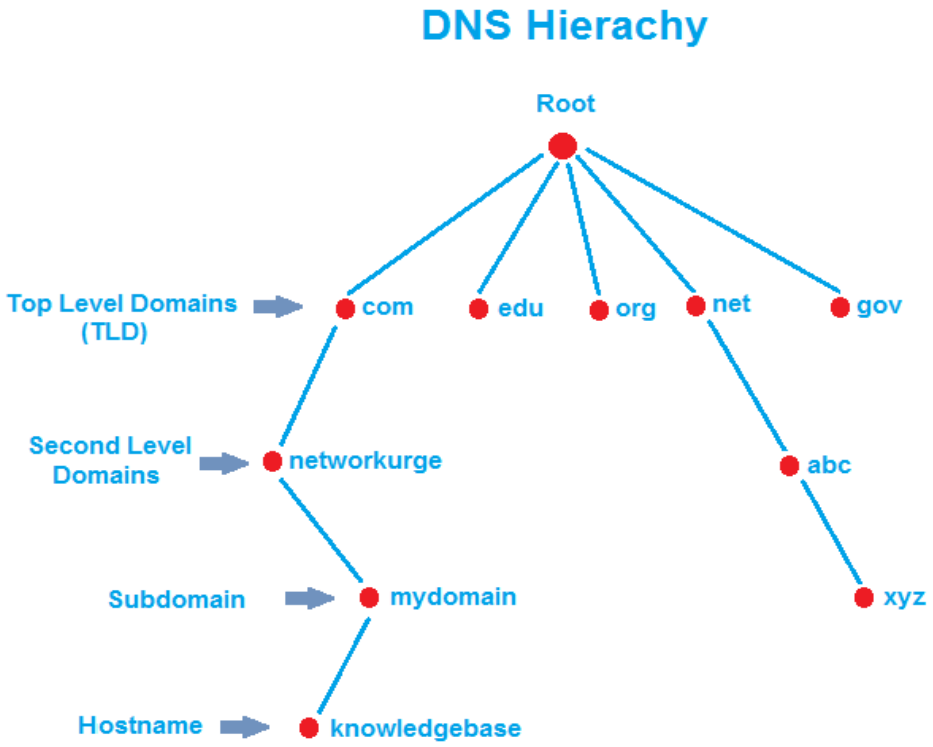
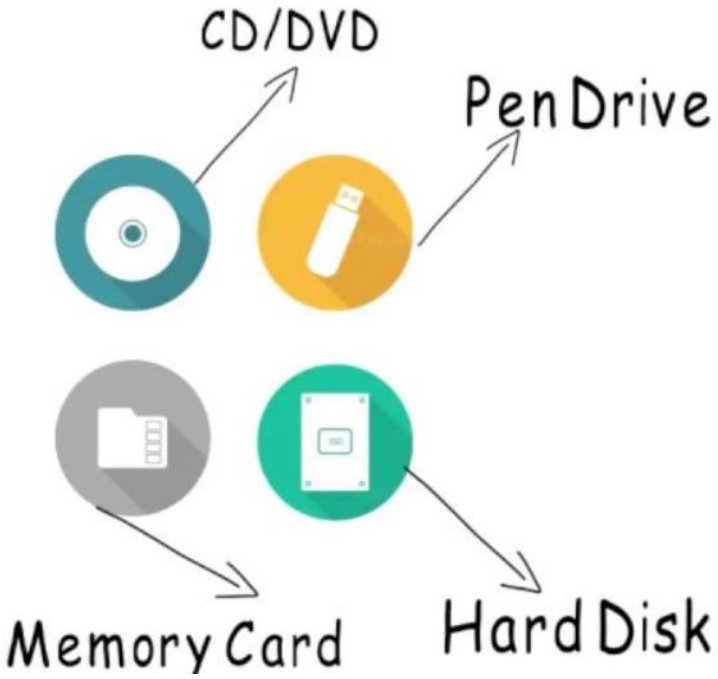


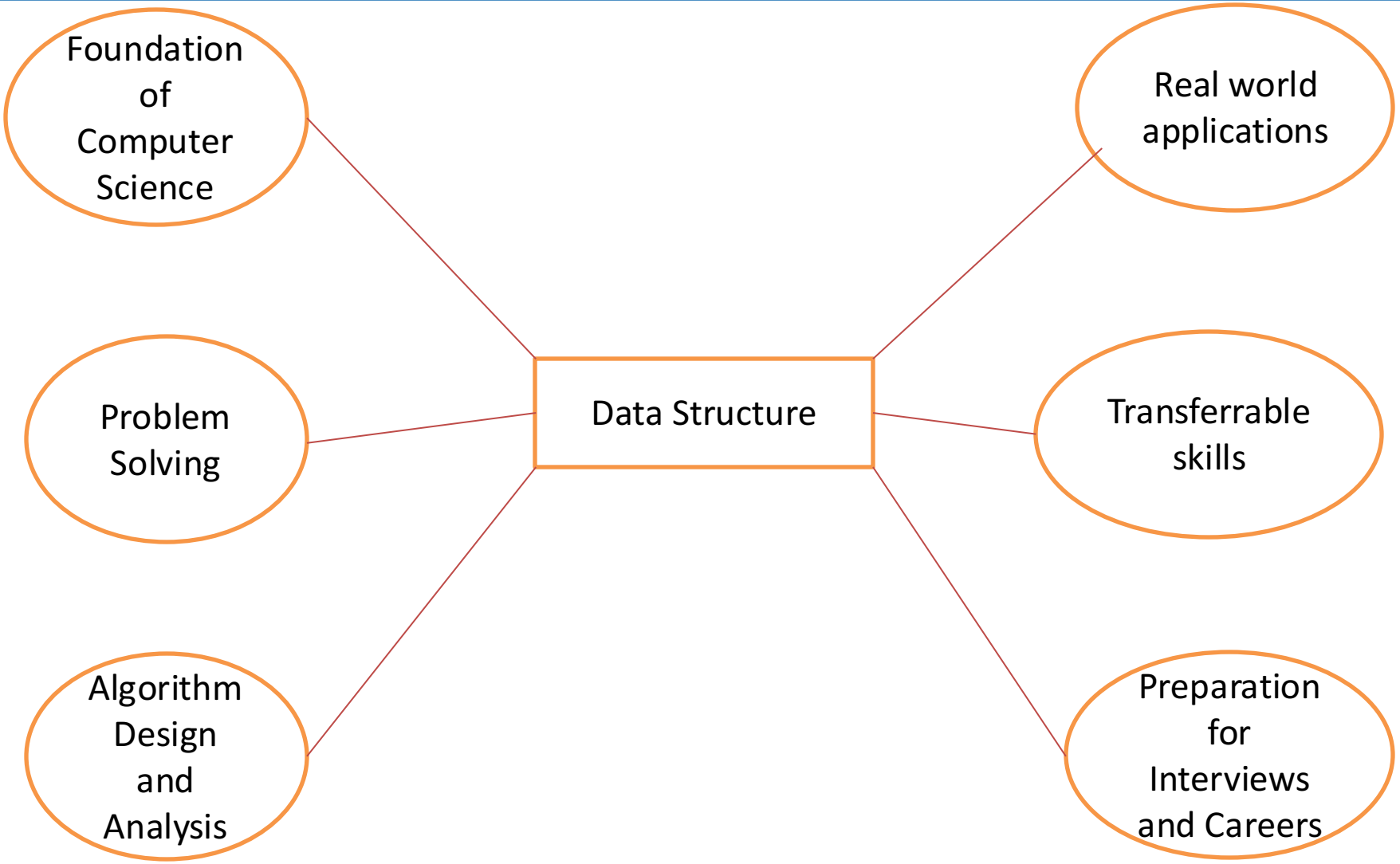
Fig. 7 : Examples of Tree Data Structure

Data Structure

- Specific way of storing and organizing data
- Uses computer memory
- Data used in efficient manner



Why Data Structure?



Definition of Data Structure

- "A data structure is a way of organizing information in a computer so that it can be used effectively." - Donald Knuth
- "A data structure is a systematic way of organizing and accessing data." - Clifford A. Shaffer
- "A data structure is a way to store and organize data in order to facilitate access and modifications." - Cormen, Leiserson, Rivest, and Stein
- "A data structure is a systematic way to organize and access data." - Robert Sedgewick
- "A data structure is a systematic way of organizing and accessing data." - Mark Allen Weiss

Basic Terminologies

- **Data**: as an elementary value or a collection of values. For example, the Employee's name and ID are the data related to the Employee.
- **Data Items**: A Single unit of value is known as Data Item. For example, age of an employee.
- **Group Items:** Data Items that have subordinate data items are known as Group Items. For example, an employee's name can have a first, middle, and last name.
- **Elementary Items**: Data Items that are unable to divide into sub-items are known as Elementary Items. For example, the ID of an Employee.

Classification of Data Structure

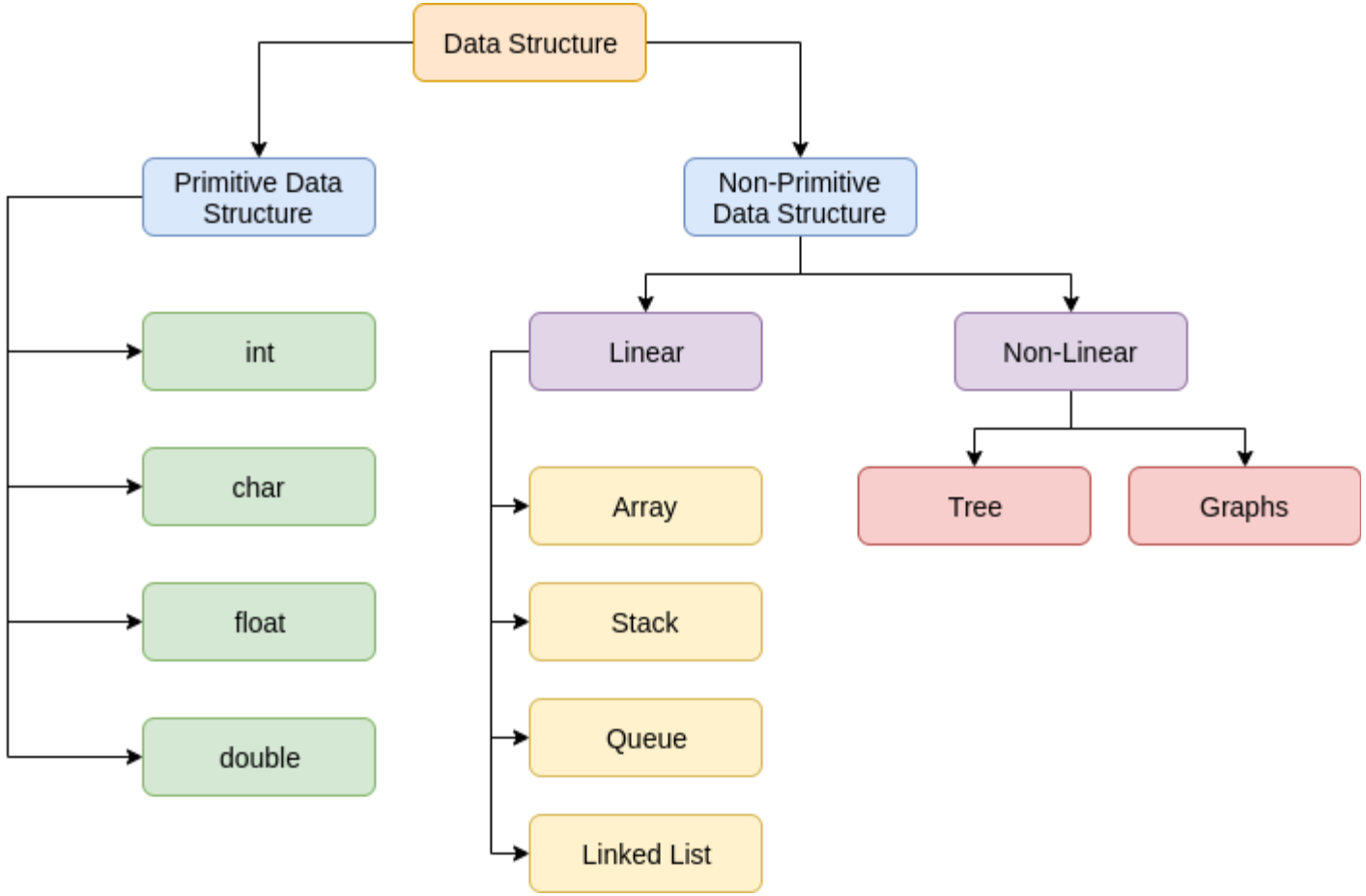


Fig. 8 : Classification of Data Structure

Primitive Data Structure

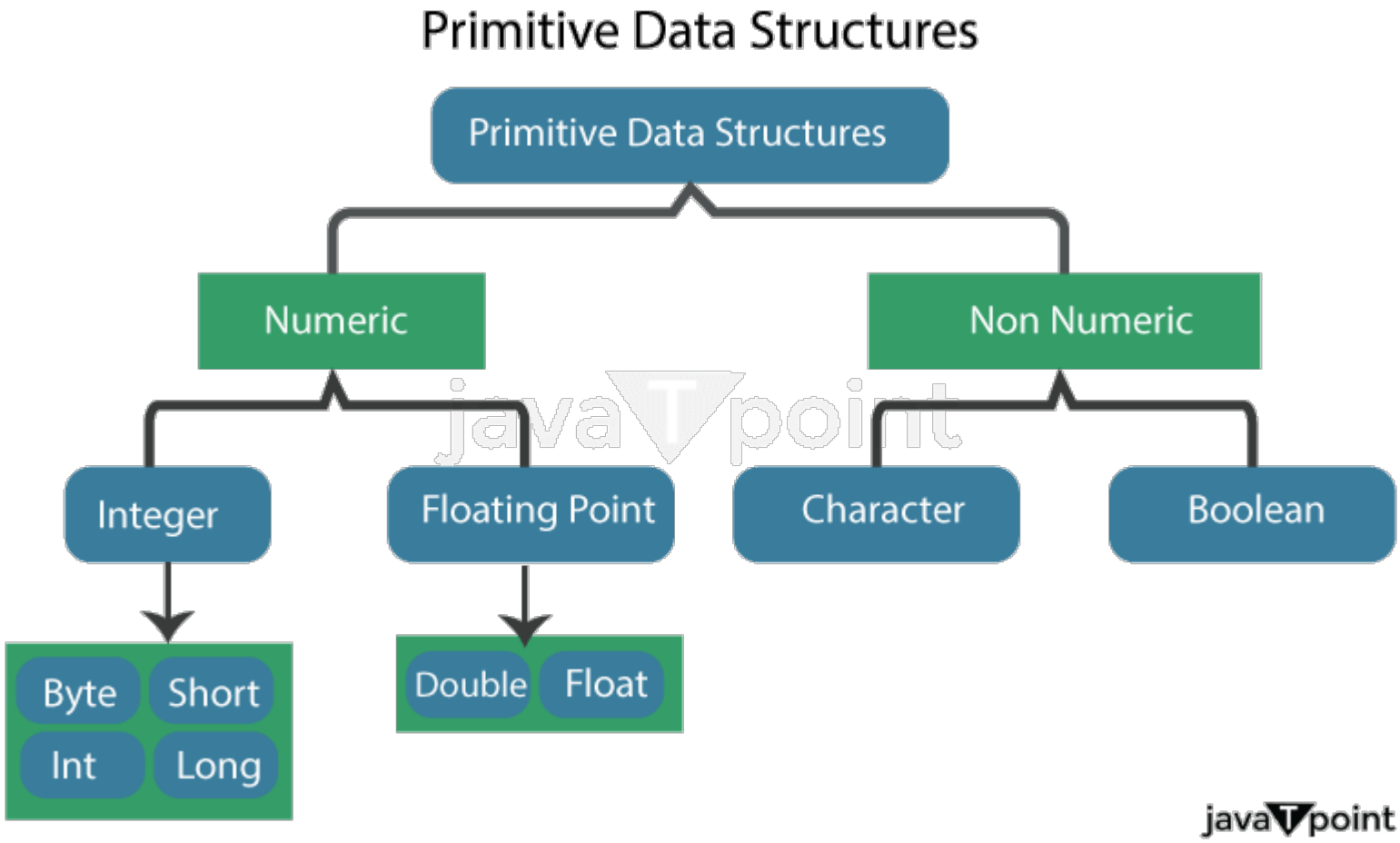


Fig. 9 : Classification of Primitive Data Structure

Types of Primitive Data Structures

1. **Bool-**

   -Represents logical value, either ‘true’ or ’false’.

   -Fixed size of 1 byte.

Syntax in C++:

```
bool a = true;
```

Syntax in Java:

```
boolean a = false;
```

Syntax in Python:

```
a = True
```

Types of Primitive Data Structures (Cont..)

2. **Byte-**

-Represents an 8-bit signed integer.
-use keyword ‘byte’ and size of 1 byte.
-Range of values from -128 to 127 (inclusive).

Syntax in Java:

```
byte myByteVariable = 127;
```

Syntax in C++:

```
byte myByteVariable = 127;
```

Types of Primitive Data Structures (Cont..)

3. Char-

-Represents single character.

-Example- a letter, a digit or symbol

-Fixed size of 2 bytes.

Syntax in C++:

```
char myChar = 'A';
```

Syntax in Java:

```
char myChar = 'C';
```

Types of Primitive Data Structures (Cont..)

4. Int-

-Represents integer values.
-Store both negative and positive whole numbers.
-Fixed size of 4 bytes in memory.

Syntax in C++:

```
int myInt = 15;
```

Syntax in Java:

```
int myInt = 25;
```

Syntax in Python:

```
myInt = 30
```

Types of Primitive Data Structures (Cont..)

5. **Float-**

-Represents floating-point numbers.
-Store real numbers (decimal values).
-Fixed size of 4 bytes in memory.

Syntax in C++:

```
float myFloat = 3.14;
```

Syntax in Java:

```
float myFloat = 5.20f;
```

Types of Primitive Data Structures (Cont..)

6. **Double-**

-Store larger decimal values with large precision.

-Fixed size of 8 bytes in memory.

Syntax in C++:

```
double myDouble = 3.14159;
```

Syntax in Java:

```
double myDouble = 3.14159d;
```

Operations on Primitive Data Structures

1. **Assignment Operations-**

-Act of putting a value in a variable.

Operator	Example	Equivalent Expression
=	$m = 10$	$m = 10$
+=	$m += 10$	$m = m + 10$
−=	$m -= 10$	$m = m - 10$
*=	$m *= 10$	$m = m * 10$
/ =	$m / =$	$m = m/10$
% =	$m \% = 10$	$m = m\%10$
<<=	$a <<= b$	$a = a << b$
>>=	$a >>= b$	$a = a >> b$
>>>=	$a >>>= b$	$a = a >>> b$
& =	$a \& = b$	$a = a \& b$
^ =	$a \hat{} = b$	$a = a \hat{} b$
| =	$a \mid = b$	$a = a \mid b$

Operations on Primitive Data Structures

2. Arithmetic Operations-

-Mathematical calculations on numeric data type.

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
– –	Decrement	A = 10; A– –	9

Operations on Primitive Data Structures (Cont..)

3. Comparison Operations-

-Compares two values and returns Boolean values.

Operator	Priority	Meaning
>	1	Greater than
>=	1	Greater than or Equal to
<	1	Less than
<=	1	Less than or Equal to
==	2	Equal to
!=	2	Not Equal to

Operations on Primitive Data Structures (Cont..)

4. Logical Operations-

-Perform Boolean operations on Boolean data types.

&& AND

Expression	Is
true && true	true
true && false	false
false && true	false
false && false	false

|| OR

Expression	Is
true || true	true
true || false	true
false || true	true
false || false	false

! NOT

Expression	Is
!true	false
!false	true

Key Features of Primitive Data Structure

1. **Fixed Size-** Example Integer takes up 4 bytes of memory.

Data Type	Size in 64-bit compilers	Range
Bool	1 byte	true or false
Byte	1 byte	-128 to 127
Char	2 bytes	0 to $2^{16} - 1$
Int	4 bytes	-2^{31} to $2^{31} - 1$
Float	4 bytes	-3.4E+38 to +3.4E+38
Double	8 bytes	-1.7E+308 to +1.7E+308

Key Features of Primitive Data Structure (Cont..)

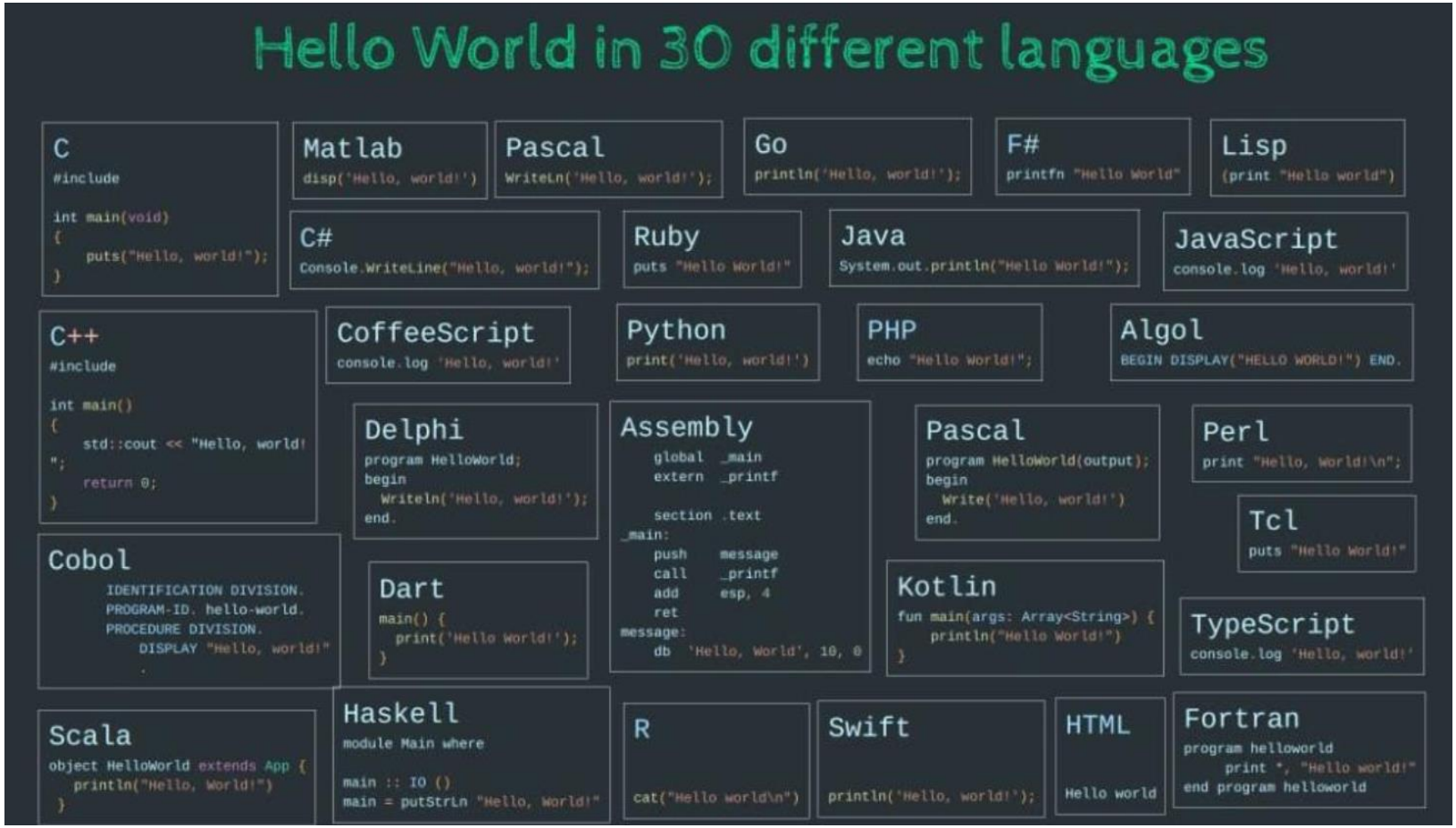
2. **High Speed-** Perform calculations easy because of fixed size and format.

3. **Memory Efficiency-** Use minimal amount of memory.

4. Efficient Access and Manipulation- Since they are directly supported by the hardware, operations on primitive data structures are highly efficient. This includes arithmetic operations on numbers, logical operations on booleans, and character manipulations.

Key Features of Primitive Data Structure (Cont..)

5. **Portability**- Code can be easily adapted to another platform using different programming language.



Limitations of Primitive Data Structures

- **Limited Functionality:** Because they are atomic and indivisible, so limited operations.
- **Limited size range:** For example, an integer data type in Java can store values between -2147483648 and 2147483647.
- **Lack of abstraction:** They don't provide a way to abstract or encapsulate data, making it more difficult to manage and maintain large programs.
- **Limited precision:** Sometimes produce unexpected results when used in calculations.

Non Primitive Data Structure (cont..)

- Derived from primitive data structures.

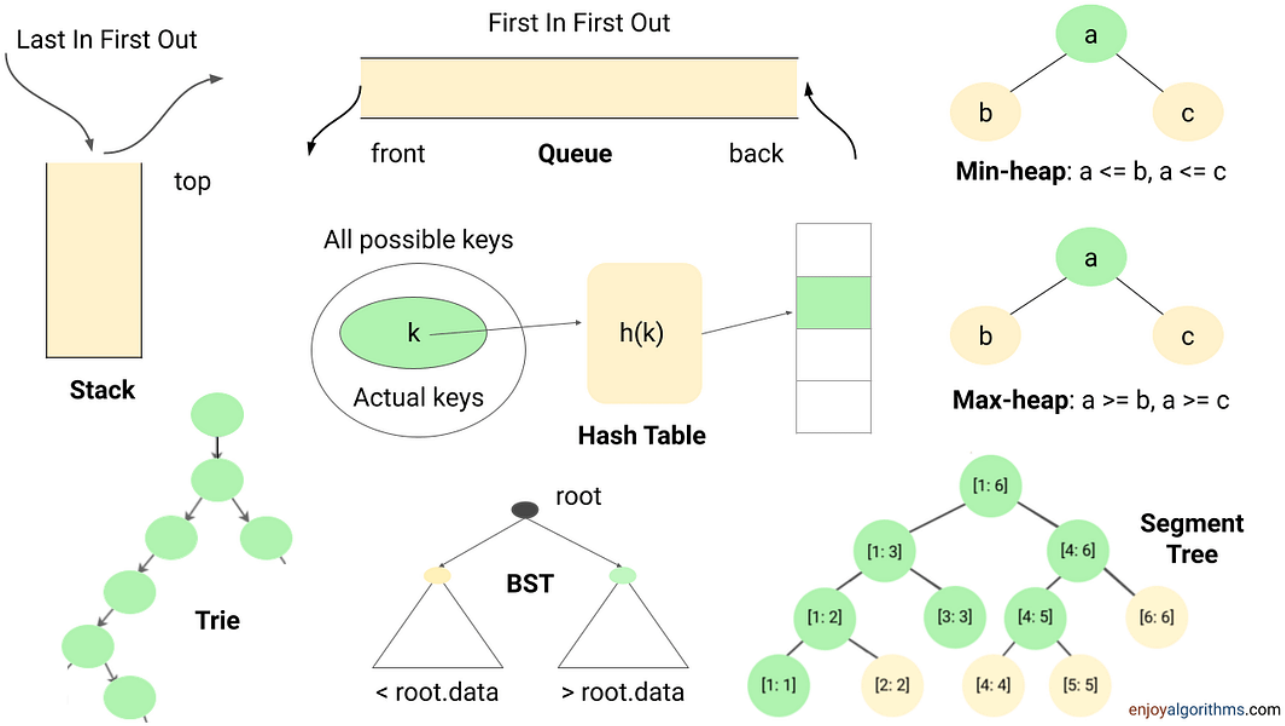


Fig. 10 : Examples of Non- Primitive Data Structure

Classification of Non Primitive Data Structure

Linear Non-Primitive Data Structures:

- Homogenous elements.
- Example- *Stack, Queue and Linked List*

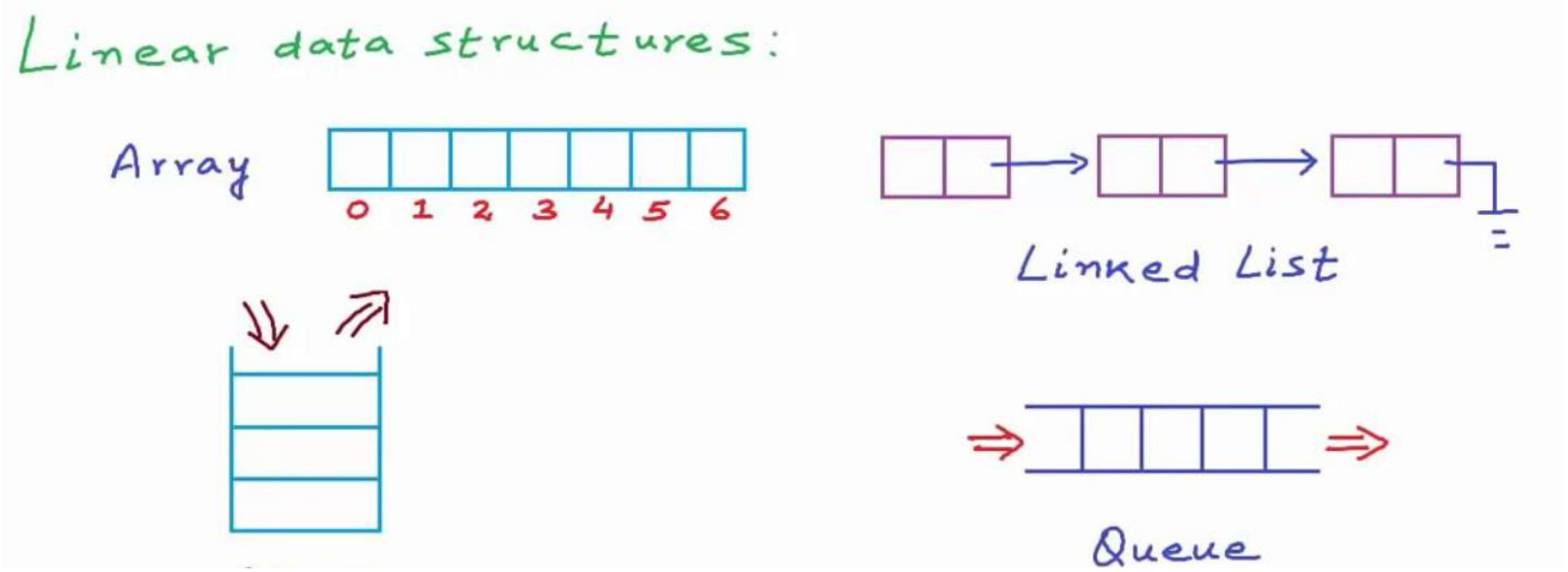
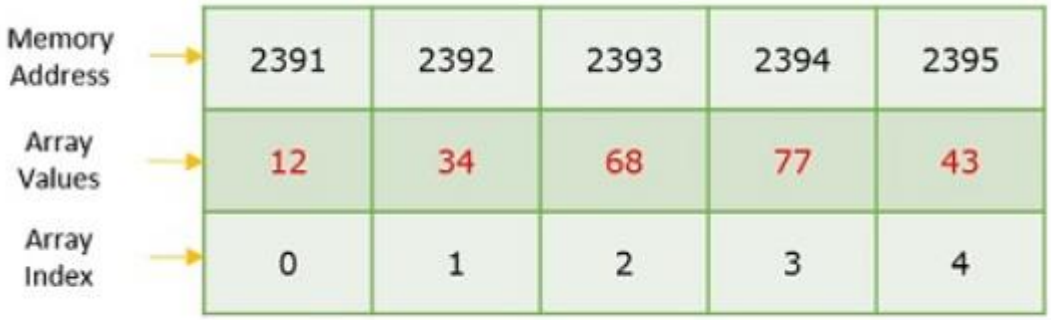


Fig. 11 : Examples of Non- Primitive (Linear) Data Structure

Types of Linear Non-primitive DS

Array:

- linear data structure
- Collection of elements with same or different data types.
- Exist in both single dimension and multiple dimensions.



C++

```
int arr[5];
```

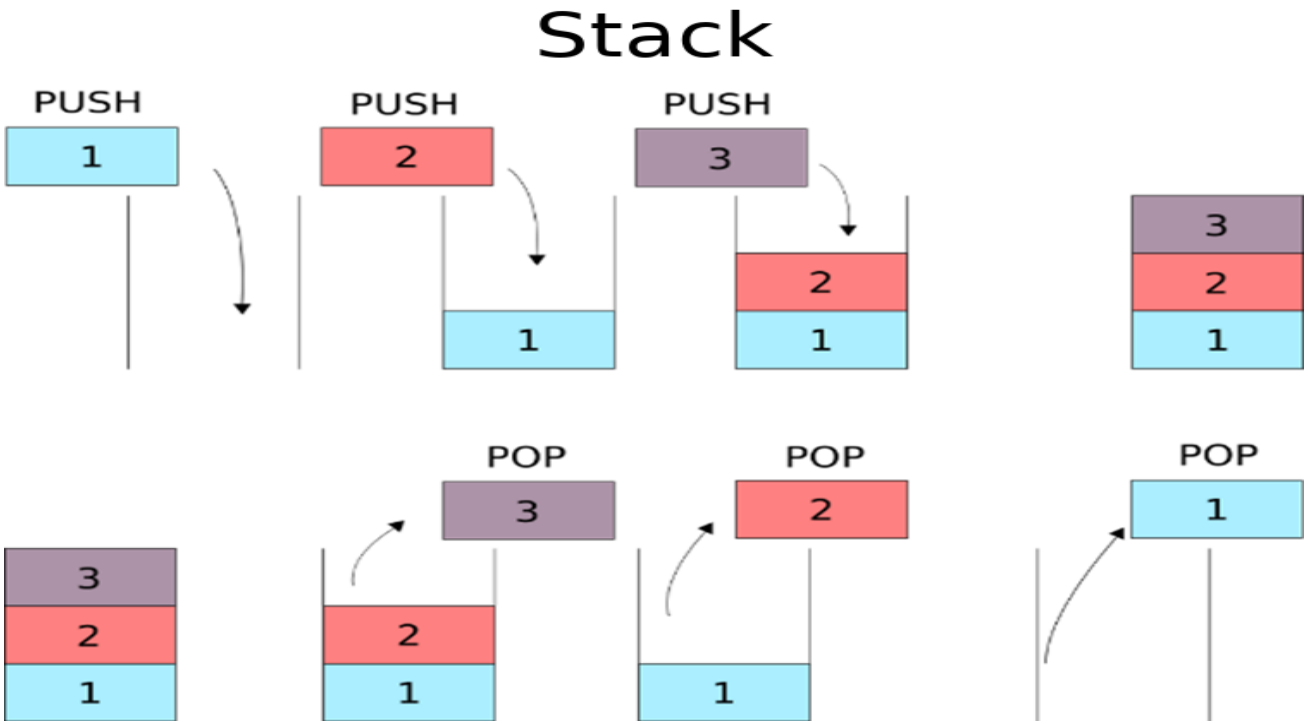
Java

```
int[] ar = new int[5];
```

Types of Linear Non-primitive DS (Cont..)

Stack:

- Linear data structure.
- Follows a particular order to perform operations.
- LIFO(Last In First Out) or FILO(First In Last Out).



Types of Linear Non-primitive DS (Cont..)

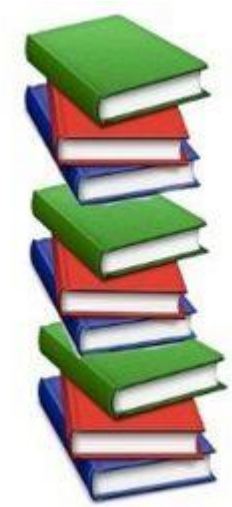


Fig. 12 : Examples of Linear Non- Primitive (Stack) Data Structure

Types of Linear Non-primitive DS (Cont..)

Queue:

- It is an ordered list.
- Insertion operations from REAR end.
- Deletion operations from FRONT end.

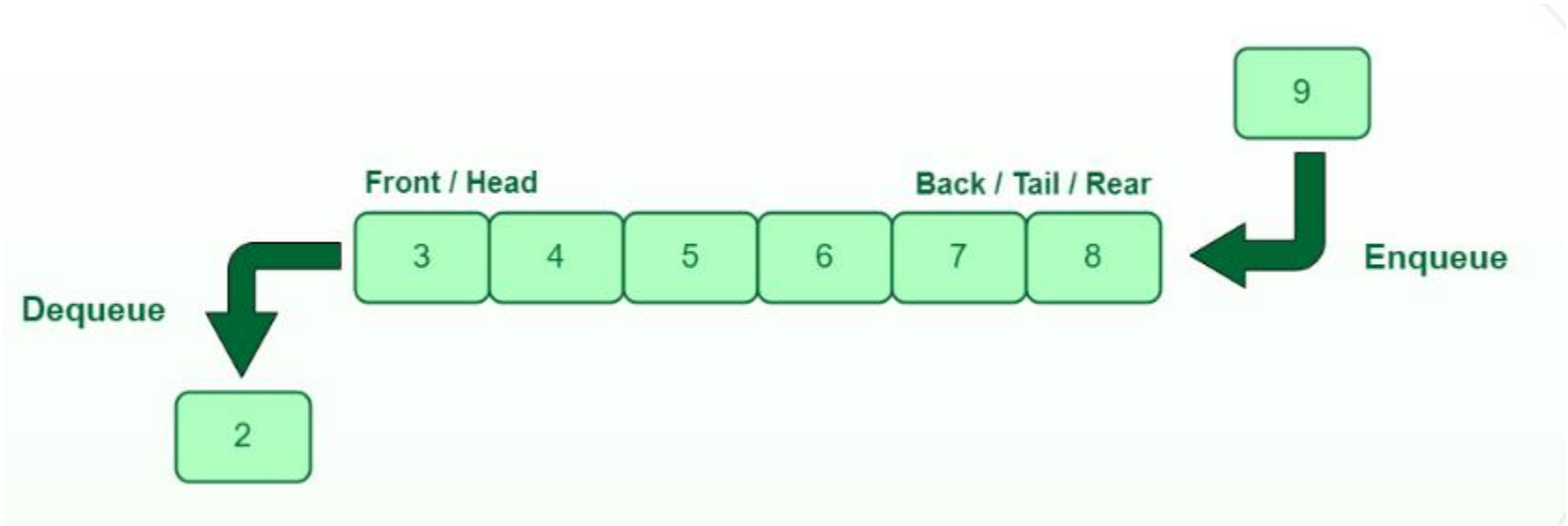


Fig. 13 : Examples of Linear Non- Primitive (Queue) Data Structure

Types of Linear Non-primitive DS (Cont..)

Queue:

- Queue is referred to be as First In First Out list (FIFO).

Fig. 13 : Example of Linear Non- Primitive (Queue) Data Structure

Types of Linear Non-primitive DS (Cont..)

Linked Lists:

- Linear data structure
- Collection of "nodes" connected via links i.e. pointers.
- Linked lists nodes are not stored at a contiguous location.
- They are linked using pointers to the different memory locations.

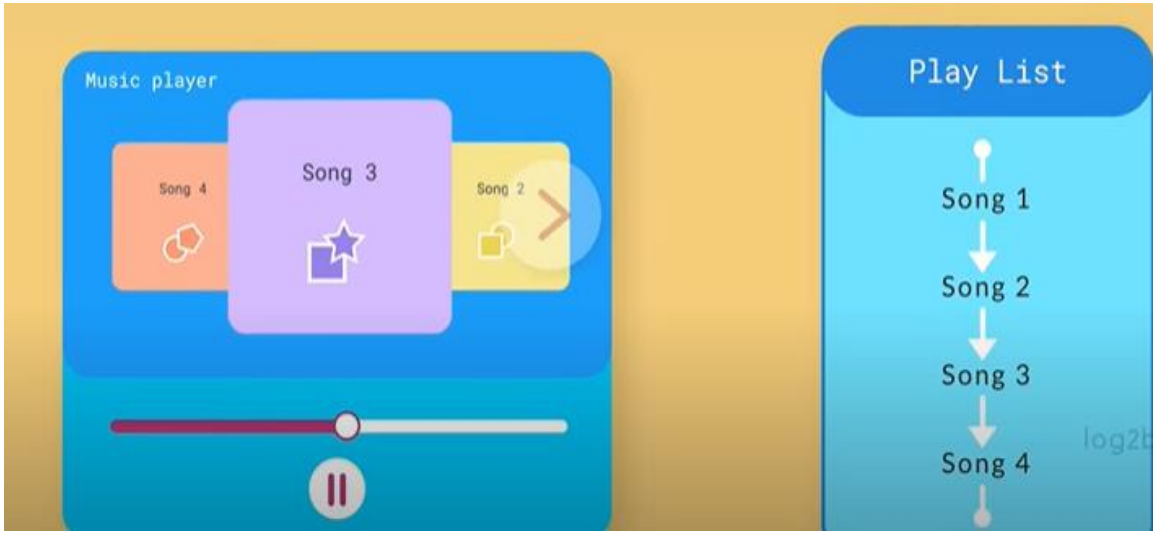


Fig. 13 : Example of Linear Non- Primitive (Linked List) Data Structure

Types of Linear Non-primitive DS (Cont..)

Linked Lists:

- A node consists of :
  -- a data value and
  -- a pointer to the address of the next node.

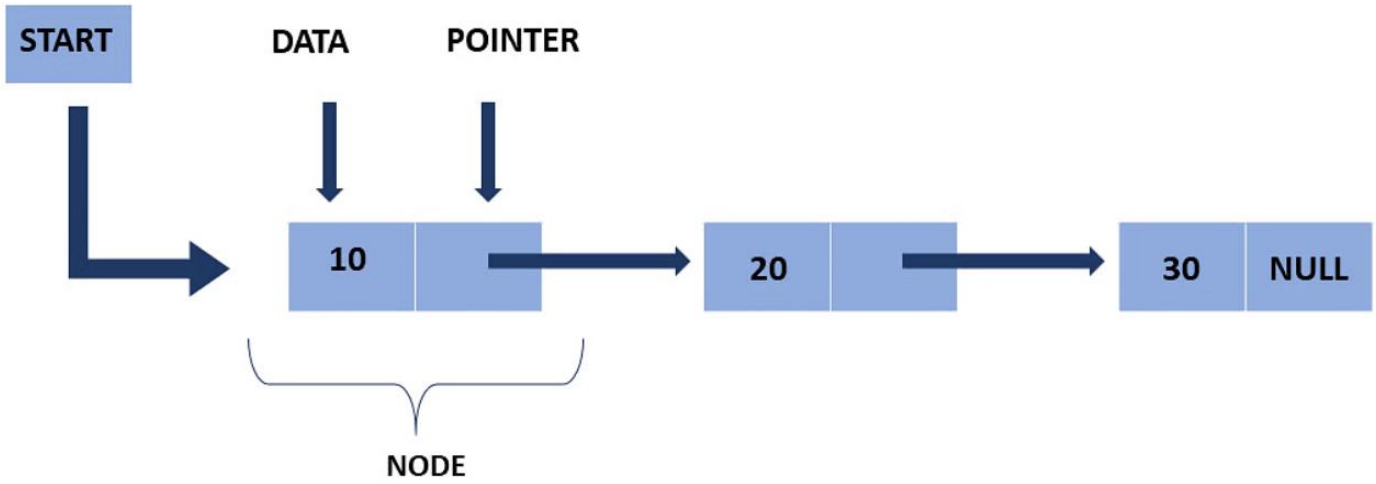
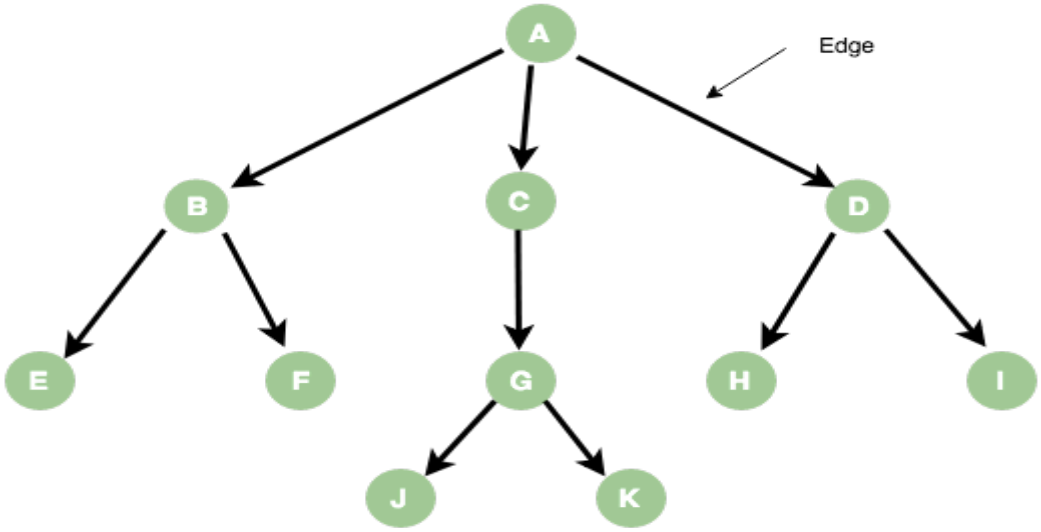


Fig. 14 : Representation of Linear Non- Primitive (Linked List) Data Structure

Classification of Non-Primitive Data Structure

Non- Linear Non Primitive Data Structure

- Data item connected to other data item.
- Hierarchical relationship or parent-child relationship.
- Example- *Trees and Graphs*



Non- Linear Non Primitive Data Structure

Trees:

- Hierarchical structure
- Used to represent and organize data in a way so that easy to navigate and search.
- Collection of nodes connected by edges.

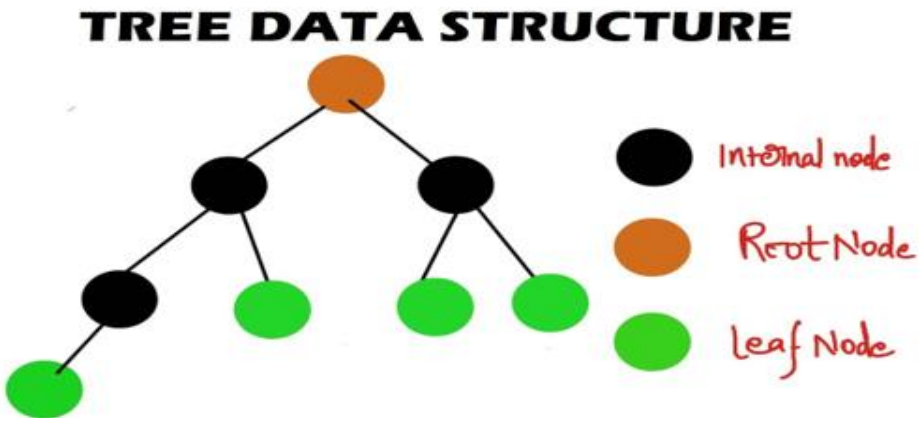


Fig. 15 : Representation of Non- Linear Non- Primitive (Tree) Data Structure

Non- Linear Non Primitive Data Structure

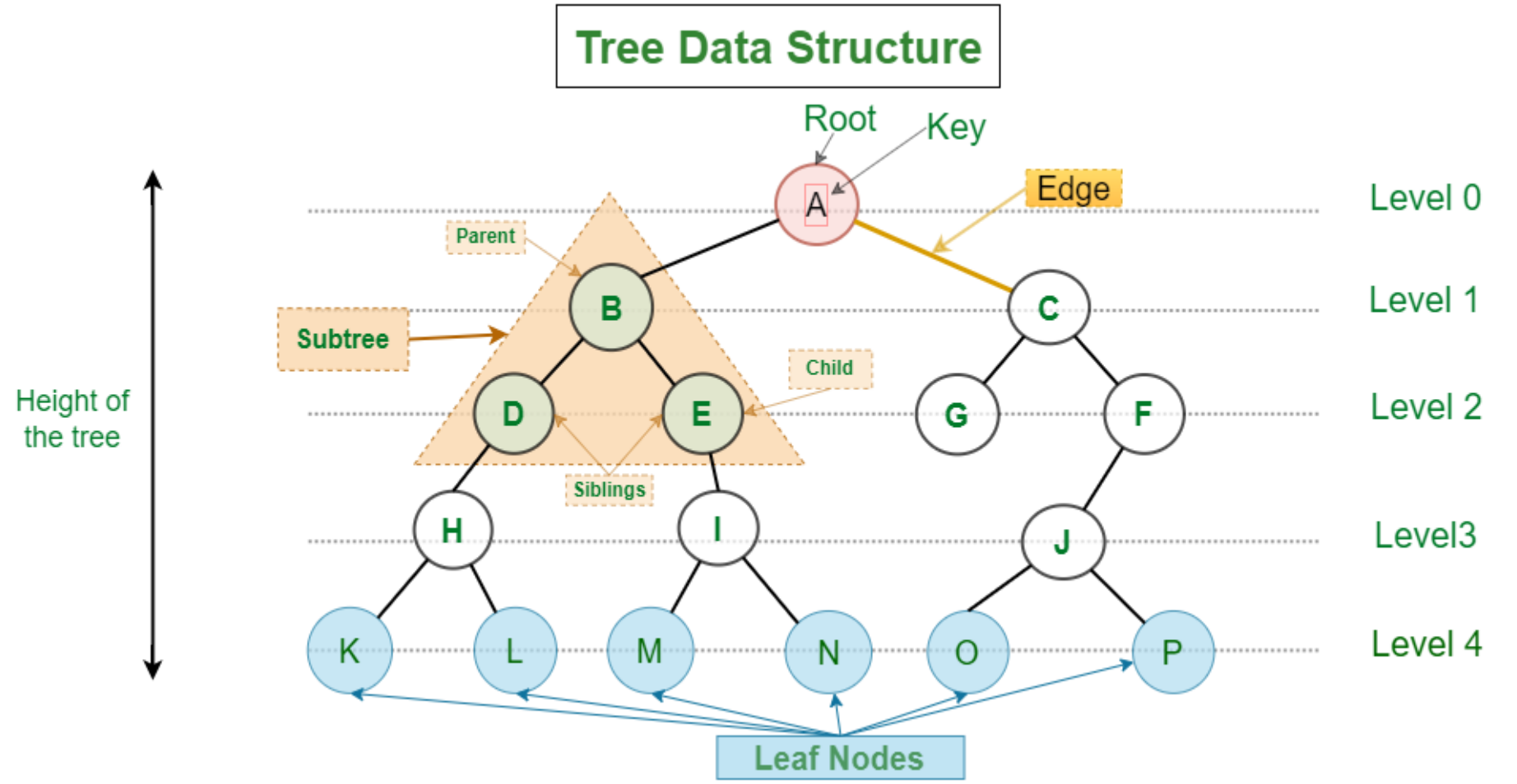


Fig. 16 : Representation of Non- Linear Non- Primitive (Tree) Data Structure

Non- Linear Non Primitive Data Structure

Graphs:

- Collection of nodes connected by edges.
- Used to represent relationships between different entities.
- Example- used to find the shortest path or detecting cycles.

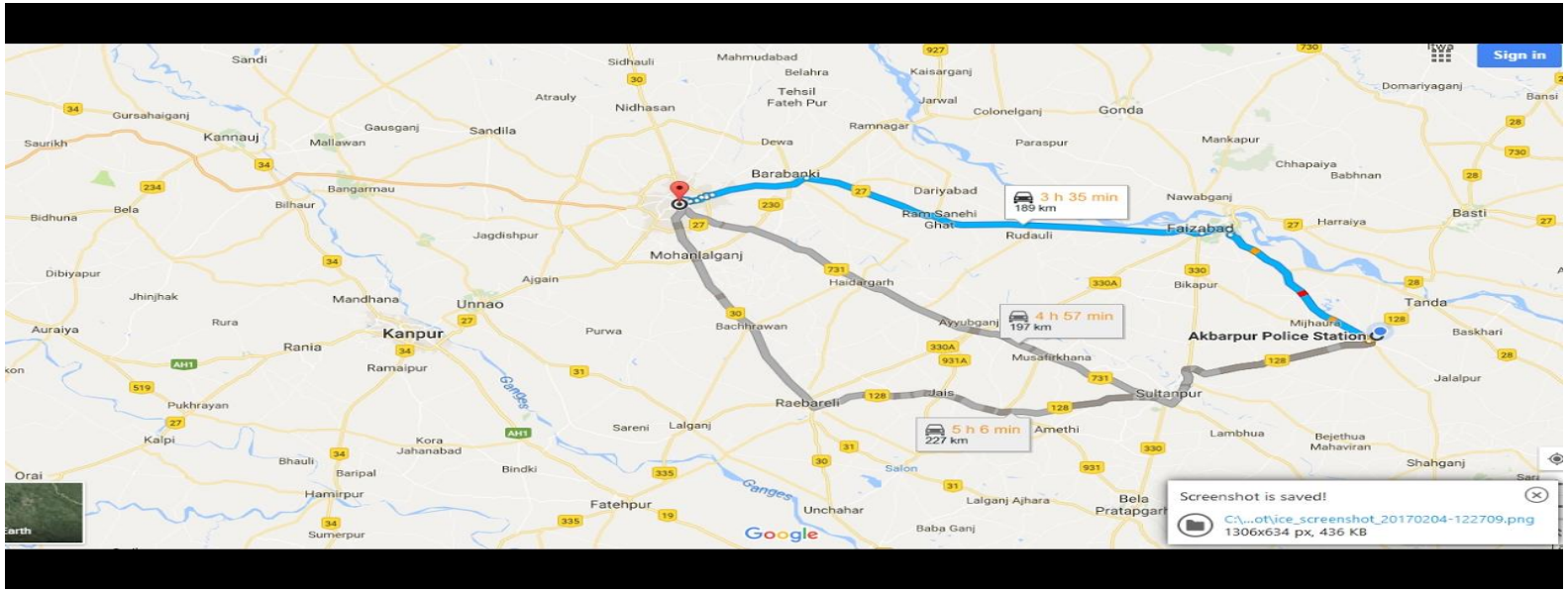


Fig. 17 : Representation of Non- Linear Non- Primitive (Graph) Data Structure

Non- Linear Non Primitive Data Structure

Types of Graphs:

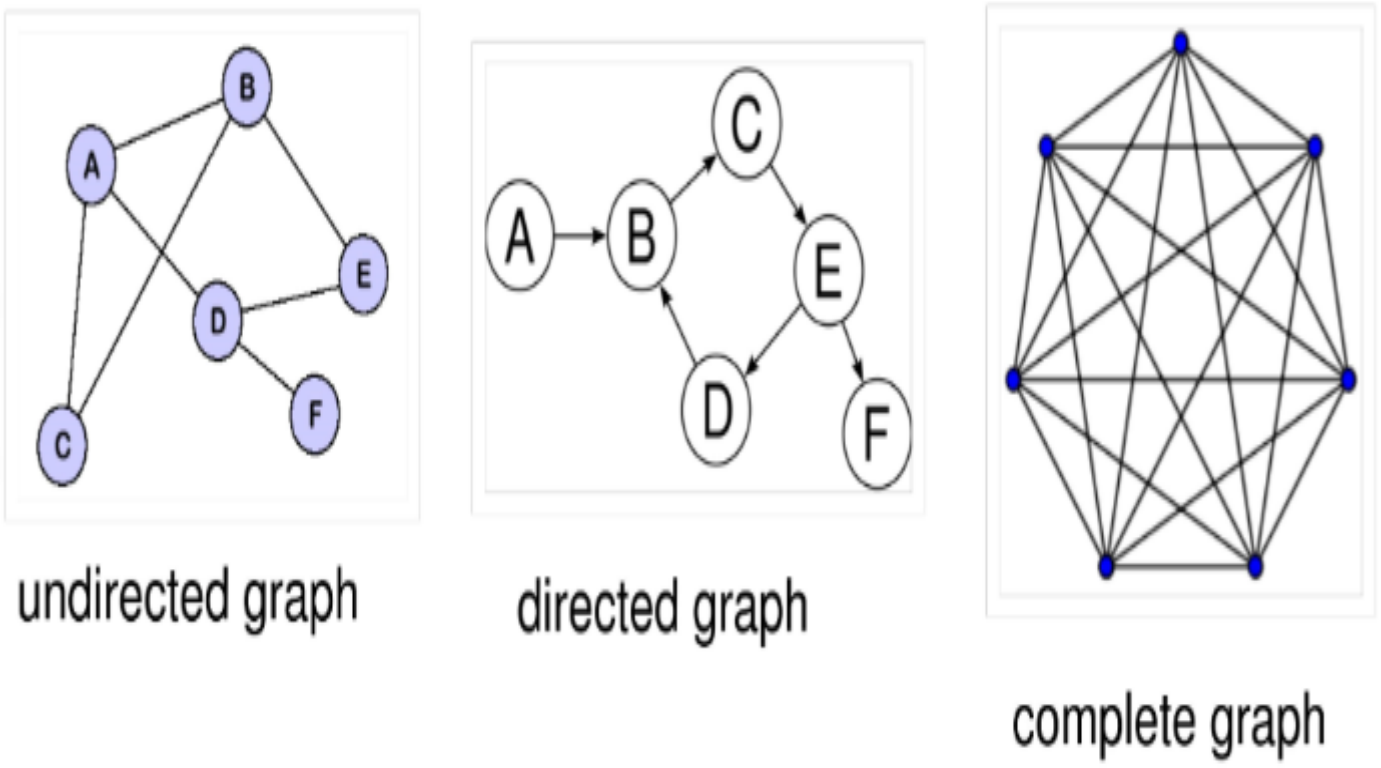


Fig. 18 : Types of Graph Data Structure

Test Your Knowledge

Give some examples of uses of data structure in our day to day life..

Situation-Based Q/A

Scenario 1 Description: Imagine you are developing a help desk ticketing system for a software company. When customers experience issues, they submit tickets to this system. Each ticket contains a unique ticket ID, the customer's details, and the problem description. The tickets need to be processed in the order they are received to ensure fairness and systematic handling.

Question: Given this scenario, which linear data structure would you choose to manage the tickets in the help desk system? Explain your choice and discuss its advantages and disadvantages in this context. Additionally, outline the key operations you would need to support, their time complexity, and how you would implement them.

Situation-Based Q/A- Solution

Scenario 1 Solution

Choice of Data Structure: Queue

Why Queue?

- **FIFO Structure:** A queue operates on a First In, First Out (FIFO) principle, which is ideal for managing tickets in the order they are received, ensuring that all customers are treated fairly and in a timely manner.
- **Simplicity:** Queues allow easy additions (enqueue) at the rear and removals (dequeue) from the front, which aligns well with processing tickets from the oldest to the newest.

Advantages

- **Order Preservation:** Ensures that tickets are handled in the exact order they are received.
- **Efficiency in Operations:** Enqueue and dequeue operations are very efficient, typically running in constant time, O(1).

Situation-Based Q/A (Cont..)

Scenario 2:

Scenario Description: Imagine you are tasked with developing a route planner for a city's public transportation network, which includes buses, subways, and trams. This system should provide the shortest path between any two stops, taking into account various factors such as transfer times, distances, and the number of stops. The transportation network can be quite complex, with multiple routes intersecting and various modes of transportation connecting at different points.

Question:

Given this scenario, which data structures would you use to model the transportation network and compute the shortest paths between stops? Explain your choice and discuss its advantages and disadvantages in this context. Additionally, outline the key operations and algorithms you would need to implement, their time complexity, and how they would function within your chosen data structures.

Situation-Based Q/A (Cont..)

Scenario 2: Solution

Choice of Data Structures: Graphs and Trees

Why Graphs and Trees?

- **Graphs for Network Representation:** A graph is ideal for representing a transportation network where stops are nodes and routes are edges connecting these nodes. Graphs can easily model the complexity of interconnected routes and various transportation modes.
- **Trees for Pathfinding Algorithms:** Specifically, trees generated by pathfinding algorithms (like Dijkstra's or A*) can help find the shortest path in a graph. These algorithms typically output a tree structure from the source to all reachable destinations, highlighting the shortest path to each.

THANK YOU