INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

Basic Data Structures and Abstract Data Types (ADT)

Data Structures

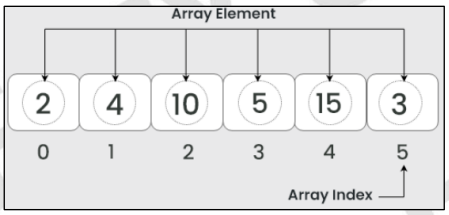
Data structures are a particular way of organizing data in a computer to be used efficiently. It is the arrangement of data in memory locations to represent values of the carrier set of an abstract data type. It provides a means to manage large amounts of data for users, such as in large databases and internet indexing services. This is key to designing efficient algorithms.

Additionally, data structures are based on the ability of a computer to fetch and store data at any place in its memory, which is specified by a **pointer** – a bit string representing a memory address that can be stored in memory and manipulated by the program.

Examples of Data Structures:

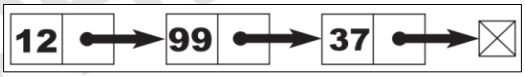
1. **Array**: A fixed-length, ordered collection of values of the same type stored in contiguous memory locations. The collection may be ordered in several dimensions. It consists of a collection of **elements** (values or variables), each identified by at least one (1) **array index** or **key**.

   An array can be considered as the simplest type of data structure and can be either a one-dimensional array or a two-dimensional array (**matrix**). They are used to implement tables, especially lookup tables, and lists and strings used by almost every program on a computer.



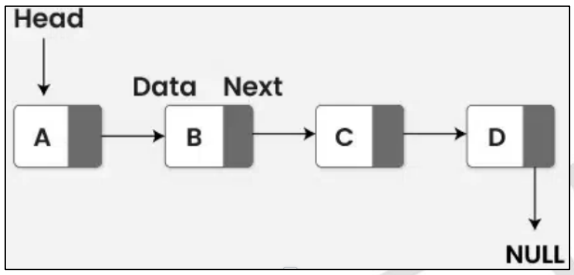
2. **List:** An abstract data type that represents a sequence of values, where the same value may occur more than once. An **instance** of a list is a computer representation of the mathematical concept of a finite sequence, while the (potentially) infinite analog of a list is a **stream**.

   Lists are a basic example of containers, as they contain other values. There are two (2) families of lists. Those that primarily expose an infrastructure for the first element, and the rest of the elements, and one whose primary interface is random access through an indexer.

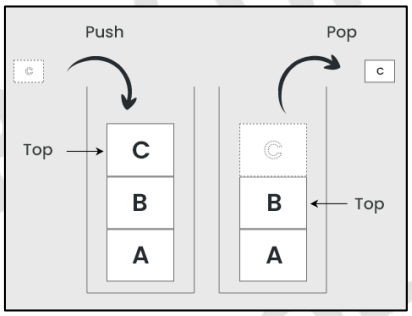


3. **Linked List**: Consists of **chains of nodes** where each node contains information such as **data** and a **pointer to the next node** in the chain. Each node is composed of data and a **reference** (a link) to the next node in the sequence; more complex variants add additional links.

   They are among the simplest and most common data structures that can be used to implement several other common abstract data types. They have the principal benefit over a conventional array in that the list elements can easily be inserted or removed. This can be done without reallocation or reorganization of the entire structure, as the data items need not be stored contiguously in memory or on disk.



4. **Stack:** A kind of abstract data type or collection in which the principal operations on the collection are the addition of an entity to the collection (**push**) and the removal of an entity (**pop**).

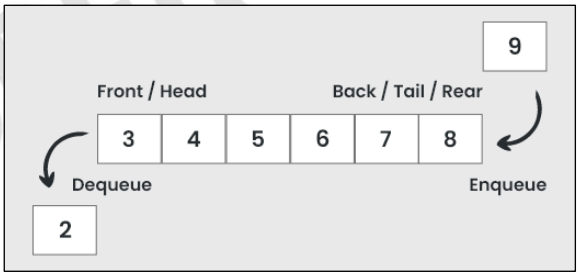


The stack is a **Last-In-First-Out (LIFO)** data structure that the last element added to the structure must be the first one to be removed. The push and pop operations occur only at one end of the structure, referred to as the **top of the stack**. A **peek** or top operation can also be implemented, returning the value of the top element without removing it.

A stack is said to be full if it does not contain enough space to accept an entity to be pushed, and it is then considered to be in an **overflow state**. The nature of the pop and push operations also means that stack elements have a natural order. Elements are removed from the stack in the reverse order of their addition.

5. **Queue**: A kind of abstract data type or collection in which the entities in the collection are kept in order, and the principal operations on the collection are the addition of entities to the rear terminal position and the removal of entities from the front terminal position**.**

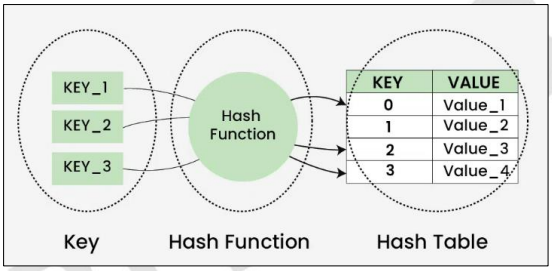
It is a **First-In-First-Out (FIFO)** data structure where the first element added to the queue will be the first one to be removed. It is an example of a linear data structure.



6. **Hashing**: A method for storing and retrieving records from a database. It allows one to insert, delete, and search for records based on a search key value. A hash system stores records in an array called a **hash table**. It works by performing a computation on a search key **K** in a way that is intended to identify the position in the hash table that contains the record with key K.

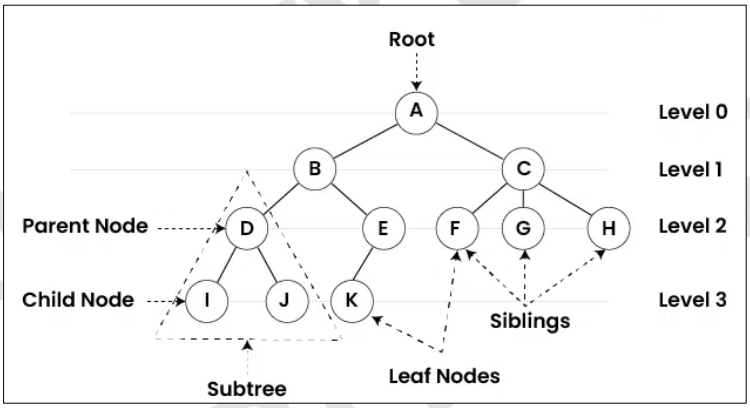
The calculations are done by a function called the **hash function**. Since hashing schemes place records on the table in whatever order satisfies the needs of the address calculation, records are not ordered by value.

A position in the hash table is known as a **slot**. The number of slots in the hash table will be denoted by the variable **M,** with slots numbered from 0 to M - 1.



7. **Trees**: A data structure made up of **nodes** or vertices and edges without having any cycles. A tree with no nodes is called the **null** or **empty tree**. A tree that is not empty consists of a **root node** and potentially many levels of additional nodes that form a hierarchy.

   A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.



Abstract Data Type

It is a mathematical model for a certain class of data structures that have similar behavior, or for certain data types of one or more programming languages that have identical semantics. An abstract data type is defined only by the operations that may be performed on it and by mathematical pre-conditions and constraints on the effects (and possibly cost) of those operations.

Abstract Data Type	Operations
Abstract Array	○ Adding elements ○ Sorting elements ○ Searching elements ○ Re-arranging the elements ○ Performing matrix operations ○ Pre-fix and post-fix operations

Abstract Data Type	Operations
Abstract List	○ Inserting ○ Searching ○ Deletion
Abstract Link	○ Checking whether the list is empty ○ Accessing a node to modify it or to obtain the information in it ○ Traversing the list to access all elements, such as to print them or to find a specific element ○ Determining the size (i.e., the number of elements) of the list ○ Inserting or removing a specific element ○ Creating a list by reading the elements from an input stream ○ Converting a list to and from an array, string, etc.
Abstract Stack	○ Push, which inserts some data item into the structure ○ Pop, which extracts an item from it ○ Peek or top, which allows data on top of the structure to be examined without removal.
Abstract Queue	○ Enqueue, to join the queue ○ Dequeue, to remove the first element from the queue ○ Front, to access and serve the first element in the queue
Abstract Hashing	○ Add (Insert) ○ Delete (Remove)
Abstract Tree	○ Searching ○ Insertion ○ Deletion ○ Traversal ○ Sort

Introduction to Algorithms

An **algorithm** is a finite sequence of steps for accomplishing a computational task. It must have steps that are simple and definite enough to be done by a computer, and it must terminate after a finite number of steps. Different algorithms may complete the same task with a different set of instructions in more or less time, space, or effort than others.

An algorithm can be considered as a computational procedure that consists of a set of instructions that takes some value or set of values as **input** and produces some value or set of values as **output**. It can also be described as a procedure that accepts data, manipulates it following the prescribed steps, to eventually fills the required unknown with the desired value(s).

The example of a recipe best illustrates the concept of an algorithm, although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as logic or comparison) until the task is completed.

Correctly performing an algorithm will not solve a problem if the algorithm is flawed or not appropriate to the problem. They are essential to the way computers process information because a computer program is essentially an algorithm that tells the computer what specific steps to perform (in what exact order) to carry out a specified task.

Characteristics of an Algorithm:

- **Each step of an algorithm must be exact**. An algorithm must be precise and unambiguously described. This eliminates any uncertainty.
- **Algorithms must terminate**. Since the ultimate aim of an algorithm is to solve a problem, it must terminate; otherwise, there will not be a solution to the problem. This leads to the fact that an algorithm must have a finite number of steps in its execution. The presence of endless loops must be avoided.
- **It must be effective**. An algorithm must provide the correct answers at all times.
- **It must be general**. An algorithm must solve every instance of a problem.
- **It must be unique.** Results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- **Finiteness.** The algorithm stops after a finite number of instructions are executed.
- **Output.** The algorithm always produces output.

Algorithms can be expressed in the following ways:

1. **Human Language**: Expressing algorithms in human language means describing a sequence of steps or instructions using **plain, natural language** instead of formal programming code or mathematical notation. This method is typically used to make the logic of an algorithm understandable to people without requiring technical syntax or specific tools.

2. **Pseudocode**: An informal high-level description of the operating principle of a computer program or other algorithm. It is a procedure for solving a problem based on the actions to be executed and the order in which those actions are to be executed.

   It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. It usually omits details that are not essential for human understanding of the algorithm, such as variable declarations, system-specific code, and some subroutines.

   It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in the planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

   For example:
   1. Start the program
   2. Enter two numbers X, Y
   3. Multiply the two numbers together
   4. Print product
   5. End program

3. **Flowchart**: A type of diagram that represents an algorithm, workflow, or process. It shows the steps in the form of boxes of various kinds and their order by connecting them with arrows. The diagrammatic representation illustrates a solution model to a given problem.

   A flowchart can be used in the analysis, design, documenting, or managing of a process or program in various fields. They are also used in designing and documenting complex processes or programs.

Here are the standard symbols used in program flowcharts.

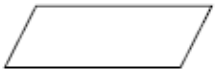
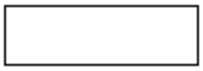
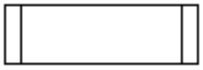
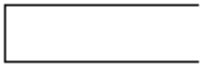
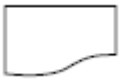
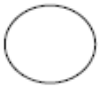
Symbol	Name	Description
	Terminal	Shows the start and end of a set of computer-related processes
	Input/Output	Shows any input/output operation
	Computer Processing	Shows any processing performed by a computer system
	Predefined Processing	Indicates any process not specially defined in the flowchart
	Comment	Used for writing any explanatory statement required to clarify something
	Flow line	Used for connecting the symbols
	Document Input/Output	Used when input comes from a document, and output goes to a document
	Decision	Shows any point in the process wherein a decision must be made to determine further action
	On-page Connector	Connects parts of a flowchart continued on the same page.
	Off-page Connector	Connects parts of a flowchart and continues to separate pages

Table 1. Flowchart symbols. Retrieved from Chaudhuri, A. (2020). Flowchart and algorithm basics: The art of programming. Mercury Learning and Information.

References:

Chaudhuri, A. (2020). *Flowchart and algorithm basics: The art of programming.* Mercury Learning and Information.

LibreTexts Engineering. (2023). *Databases and data structures.* Retrieved on June 19, 2025 from https://eng.libretexts.org/Bookshelves/Computer_Science/Databases_and_Data_Structures