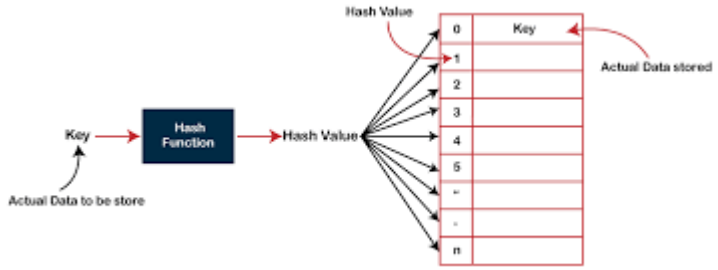
Data Structures

Stacks: Consider a table with a stack of plates on it. You build additional plates on top of the stack as you add them as needed. Similar to this, you always take the plate from the top of the stack when you want to remove one.

The core of a stack is its Last-In-First-Out (LIFO) behaviour. It adheres to the principle that the last thing you add to the stack is the first thing you remove from it. Managing function calls and enabling undo/redo features are just two common uses for stacks.

Queues: Imagine a line of people waiting to purchase tickets at a ticket desk. The first person in line gets to go first, and new people enter the line towards the back. The person in head of the line is the one who is served first when it is time to serve someone.

The core of a queue is its First-In-First-Out (FIFO) characteristic. It adheres to the principle that the first item you put in the queue will be the first one removed. Queues are frequently employed in situations when it is crucial to process jobs or items in a particular order.



Hash-Tables: Imagine you have a collection of students' names and their corresponding ages. You want to store this information in a way that allows for quick lookup of ages based on the students' names. This is where a hash table comes in handy.

In this case, think of a hash table as a set of labelled boxes, where each box represents a slot in the table. Each slot is labelled with a student's name, and inside each slot, you can store the corresponding age.
Let's consider the following student-age pairs:

Name: "Alice", Age: 15
Name: "Bob", Age: 16
Name: "Charlie", Age: 14
To store this information in a hash table, we perform the following steps:

Hashing: We apply a hash function to each student's name. The hash function calculates a numerical value based on the name. For example, the hash function could give us the following results:

Hash("Alice") = 2
Hash("Bob") = 1
Hash("Charlie") = 0
Storing: Using the hash values, we place each student's name and age into the respective slots in the hash table. The table now looks like this:

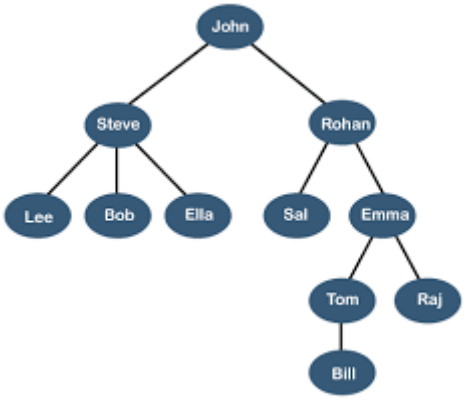
Slot 0: ("Charlie", 14)
Slot 1: ("Bob", 16)
Slot 2: ("Alice", 15)

Retrieval: Now, if we want to find the age of a specific student, let's say "Alice," we can use the same hash function to determine the slot where "Alice" is stored, which is Slot 2. By accessing Slot 2, we can retrieve the age, which is 15.

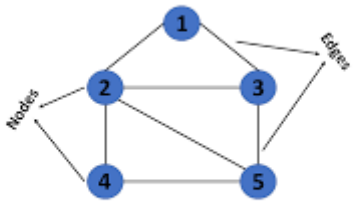
Hash tables are useful for efficient key-value pair lookups and duplicate detection. Instead of searching through the entire table, the hash function allows us to directly

determine the slot where the information is stored, providing quick access to the desired value.

In summary, think of a hash table as a storage system that uses a hash function to convert keys into numerical values. These values determine the slots where data is stored. Hash tables allow for efficient storage and retrieval of information based on the calculated index values from the keys.



Trees: Like family trees or organizational systems, trees indicate hierarchical patterns. The lines (or edges) depict the connections between the nodes, and each box (or node) contains some data (such as a person's name).
Imagine you have a tree-like structure made of boxes connected by lines. Each box represents a person, and the lines represent relationships between them, like parent-child relationships. The topmost box represents the oldest person, and the boxes below represent their children and grandchildren.



Graphs: Graphs represent connections between different elements. Each circle (or vertex) represents an element (like a city), and the lines (or edges) represent the connections between elements.Imagine you have a network of circles connected by

lines. Each circle represents a city, and the lines represent roads connecting them. You can travel from one city to another by following the roads.