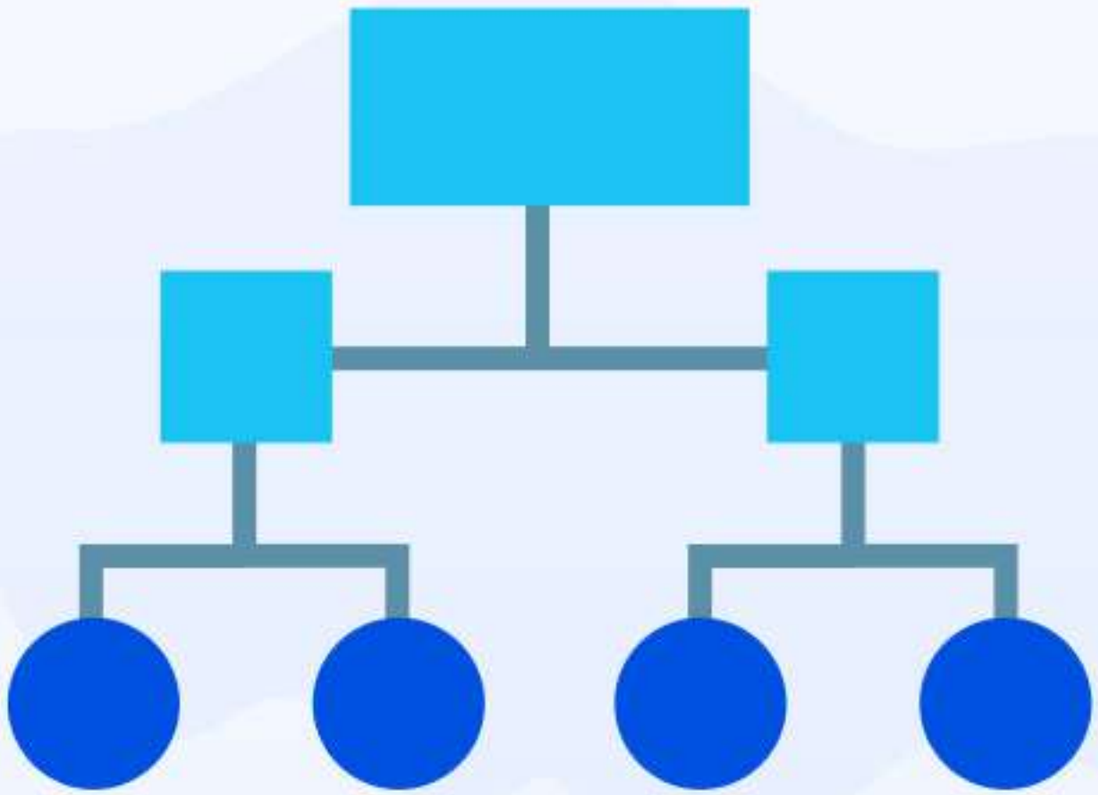
BOSSCODER ACADEMY

DATA STRUCTURE

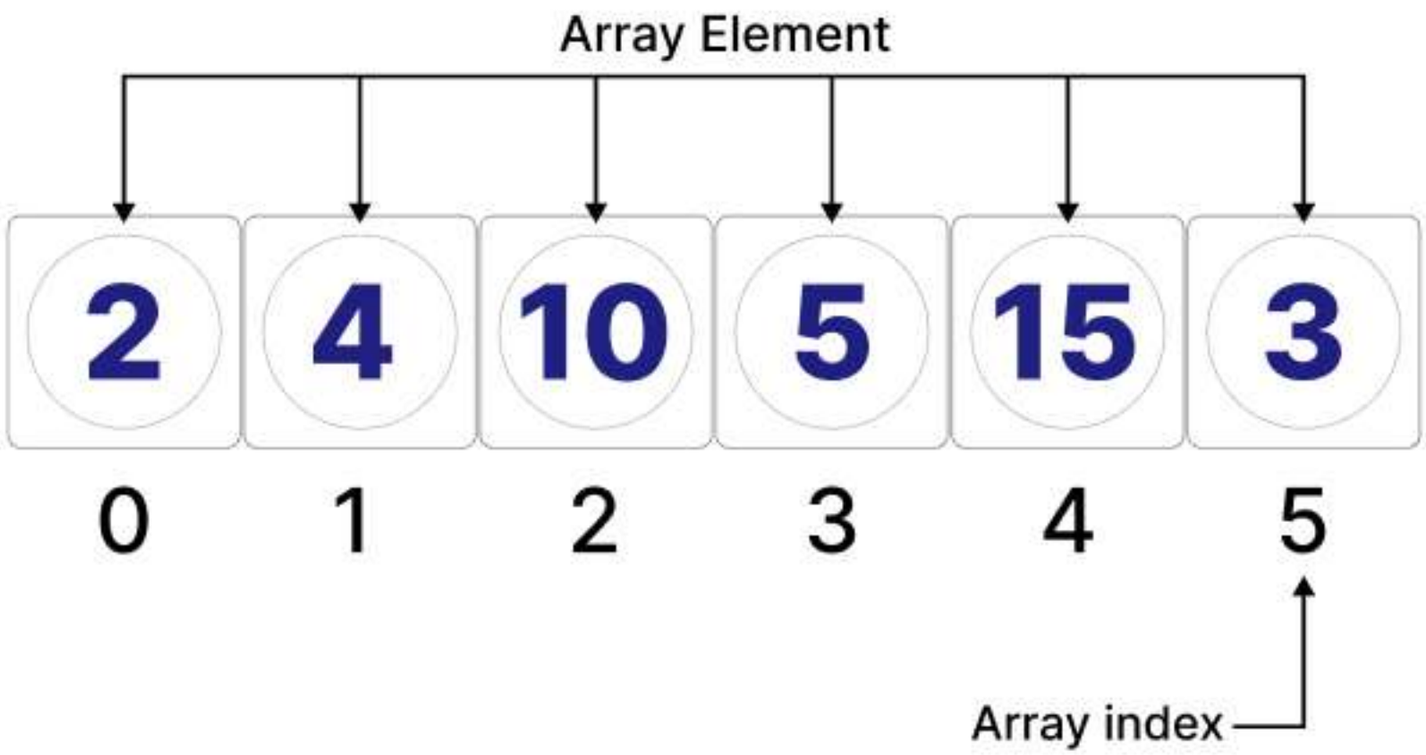
QUICK START

GUIDE



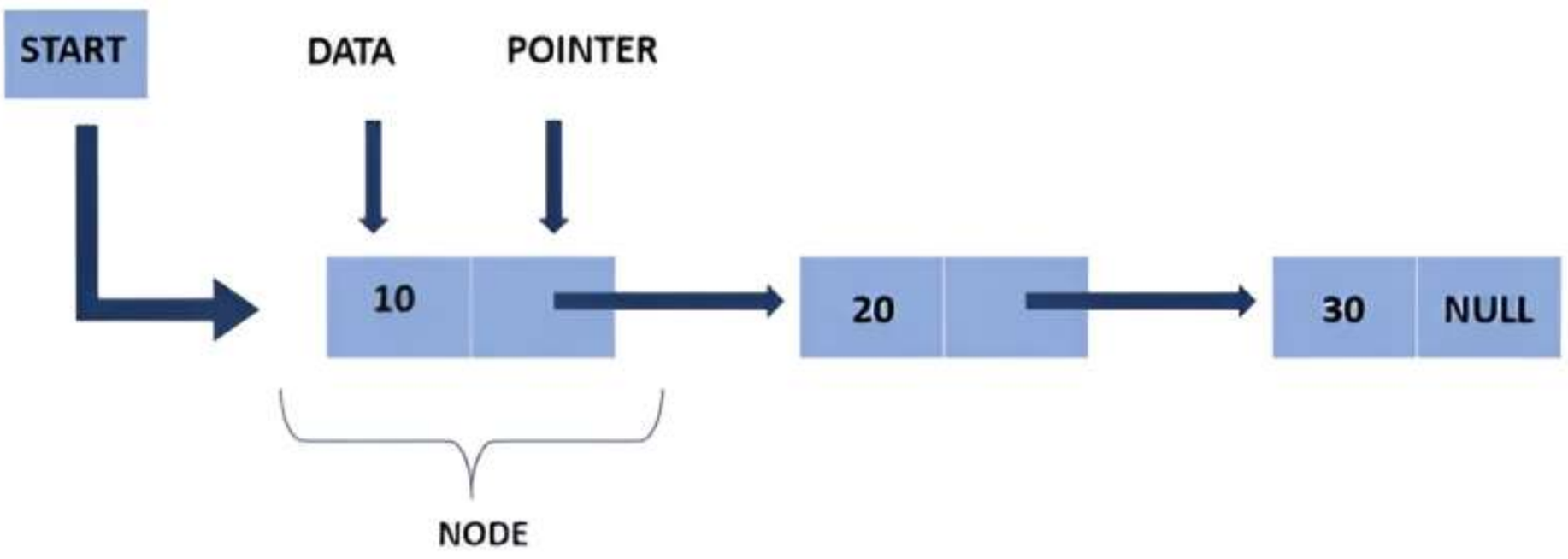
Arrays

- Arrays store multiple elements in contiguous memory.
- Elements are accessed by index.
- Arrays have a fixed or resizable size.
- Arrays can be iterated over using loops (***for***, ***while***) or array-specific methods (like ***forEach***, ***map***, ***filter*** in JavaScript) to perform operations on each element efficiently.
- Arrays support a wide range of operations including adding/removing elements, searching for elements, sorting, and more, making them versatile structures for data manipulation.



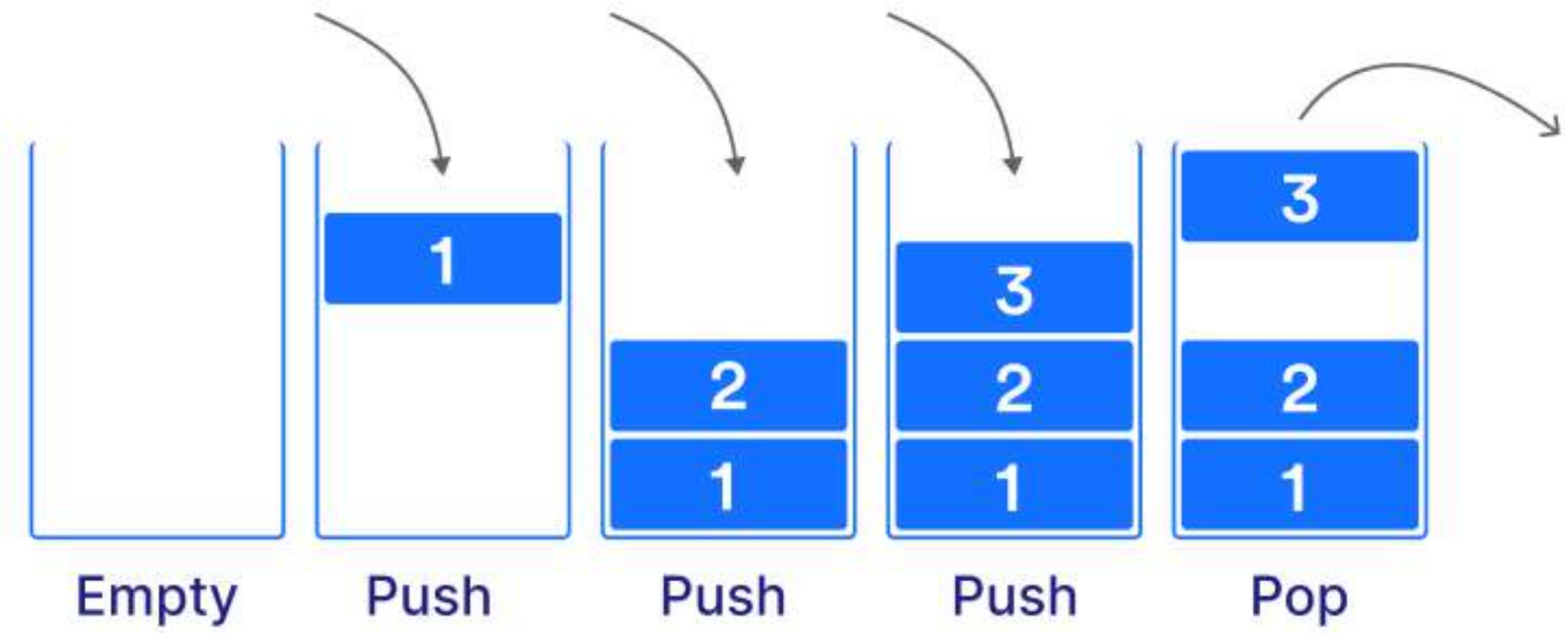
Linked List

- LinkedLists store elements with next node references.
- They support dynamic size adjustments.
- Efficient insertion and deletion operations.
- **No Need for Preallocation:** LinkedLists do not require an initial size declaration, growing as new nodes are added.
- **Bidirectional Traversal:** Doubly linked lists allow for traversal in both forward and backward directions, thanks to references to both next and previous nodes.



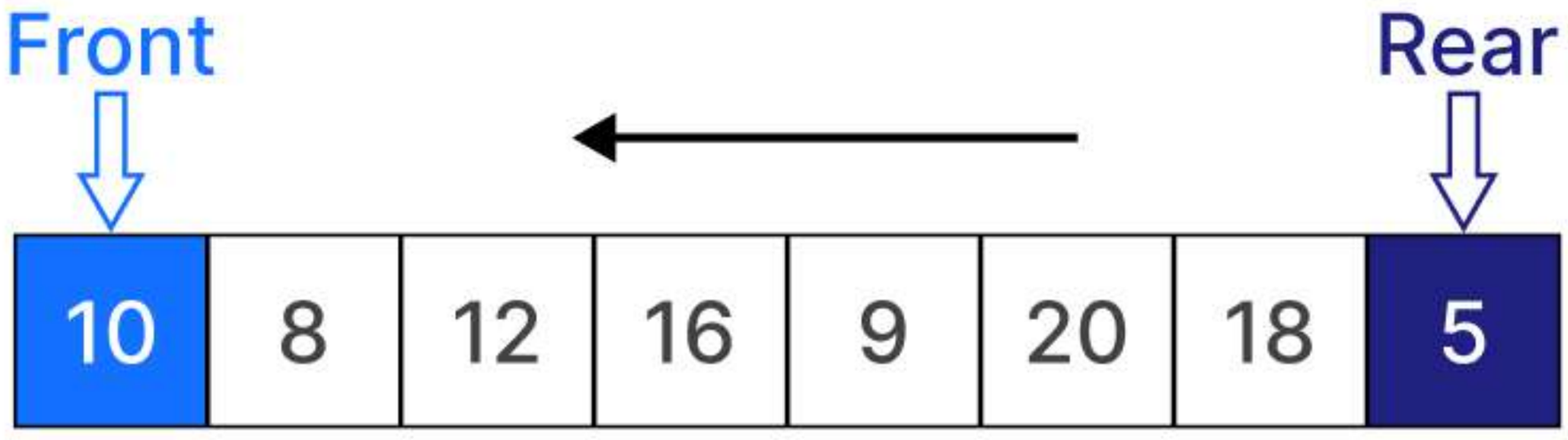
Stack

- Stacks follow last-in, first-out order.
- Efficient insertion and deletion at the top.
- Used for function calls, undo operations, etc.
- Only the top element is accessible at any time, preventing direct access to the interior elements of the stack.
- Stacks can be implemented using various data structures, such as linked lists or arrays, offering flexibility in terms of underlying data management and memory usage.



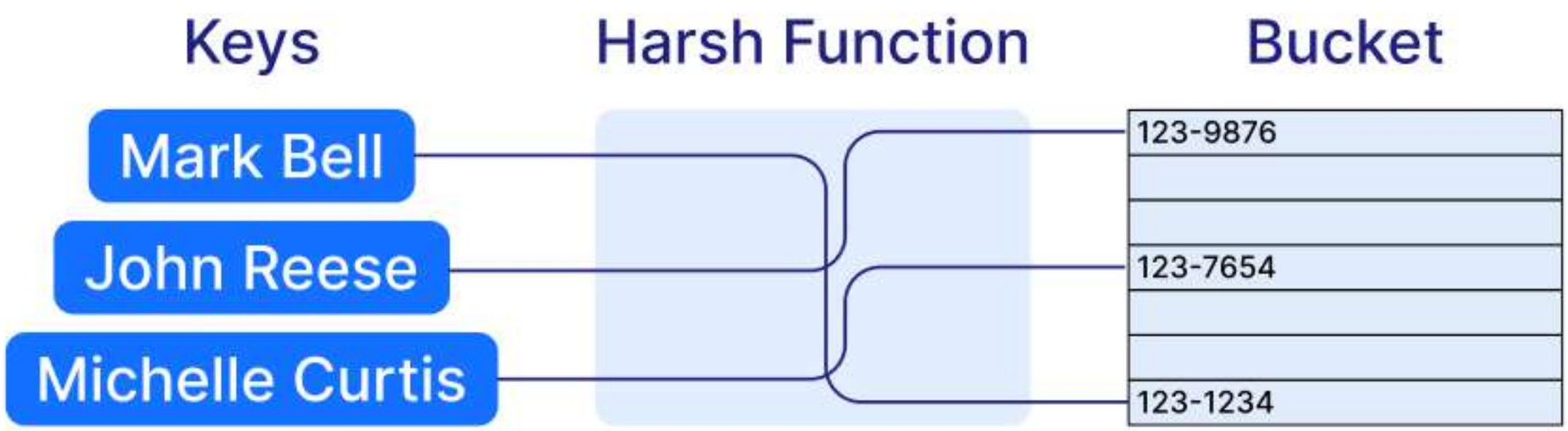
Queue

- Queues follow a first-in, first-out order.
- Efficient insertion at the rear and deletion at the front.
- Used for managing tasks, message passing, etc.
- **Concurrency Support:** Queues are fundamental in concurrent programming, allowing for safe data exchange between threads or managing tasks in multi-threaded applications.
- **Variants Offer Flexibility:** Specialized queue types, such as priority queues (which sort elements by priority) and circular queues (which optimize storage space), cater to specific needs, enhancing the utility of queues in diverse applications.



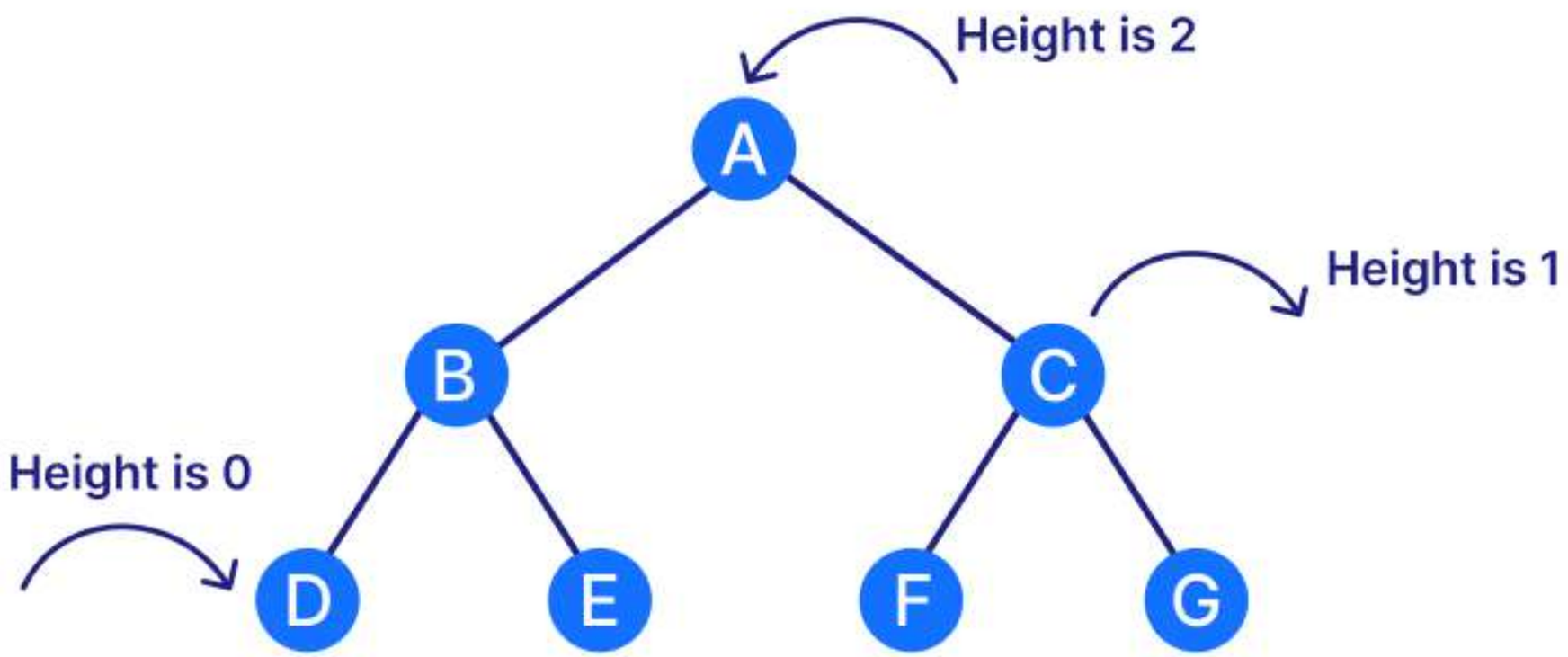
Hash table

- Hash tables store key-value pairs for quick lookup.
- They use a hash function for indexing.
- Efficient retrieval, insertion, and deletion operations.
- **Handling Collisions:** Hash tables address key collisions through methods like chaining (linking items with the same hash index) or open addressing (finding another slot using a probing sequence), ensuring that each key-value pair is accessible even when multiple keys hash to the same index.
- **Load Factor:** The performance of a hash table is closely linked to its load factor (the ratio of the number of stored entries to the number of slots).



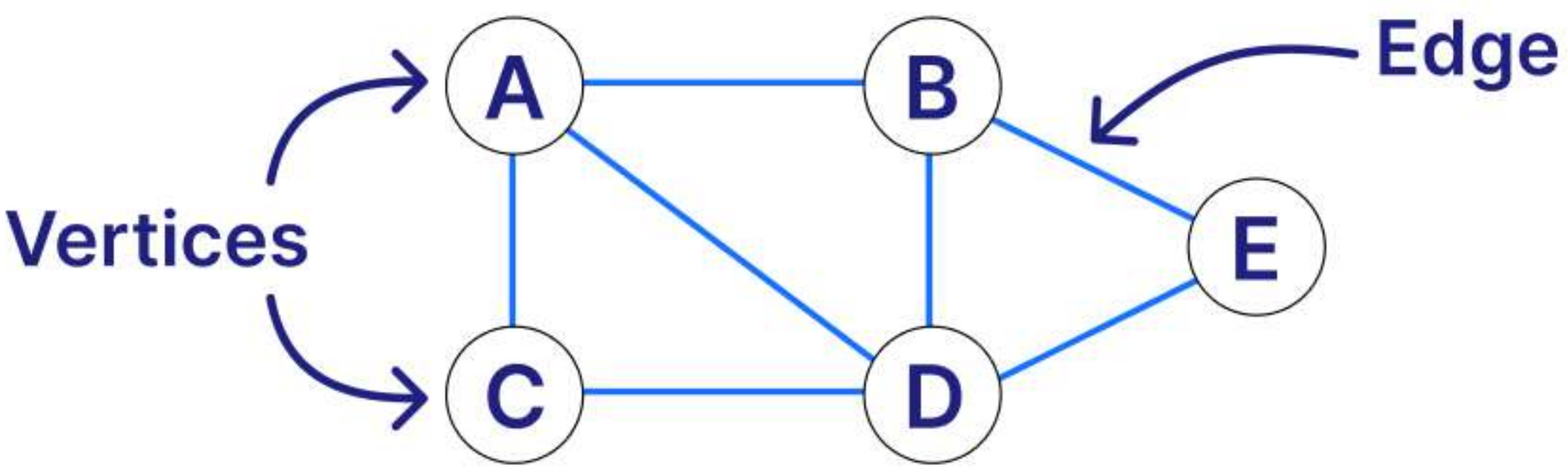
Tree

- Trees organize elements in a hierarchical structure.
- Elements have parent and child relationships.
- Used for hierarchy, searching, sorting, etc.
- **Balancing for Efficiency:** Many tree structures, such as AVL trees and red-black trees, incorporate balancing mechanisms to ensure that the tree remains as balanced as possible. This balancing significantly improves search, insert, and delete operation efficiencies by maintaining the tree's height as minimal.



Graph

- Graphs represent relationships between entities.
- They consist of vertices and edges.
- Used for modeling networks, social connections, etc
- Directed vs. Undirected: Graphs can be either directed, where edges have a direction indicating a one-way relationship, or undirected, with edges showing a two-way, reciprocal relationship. This distinction is crucial for applications like web navigation (directed) or undistorted social networks (undirected).



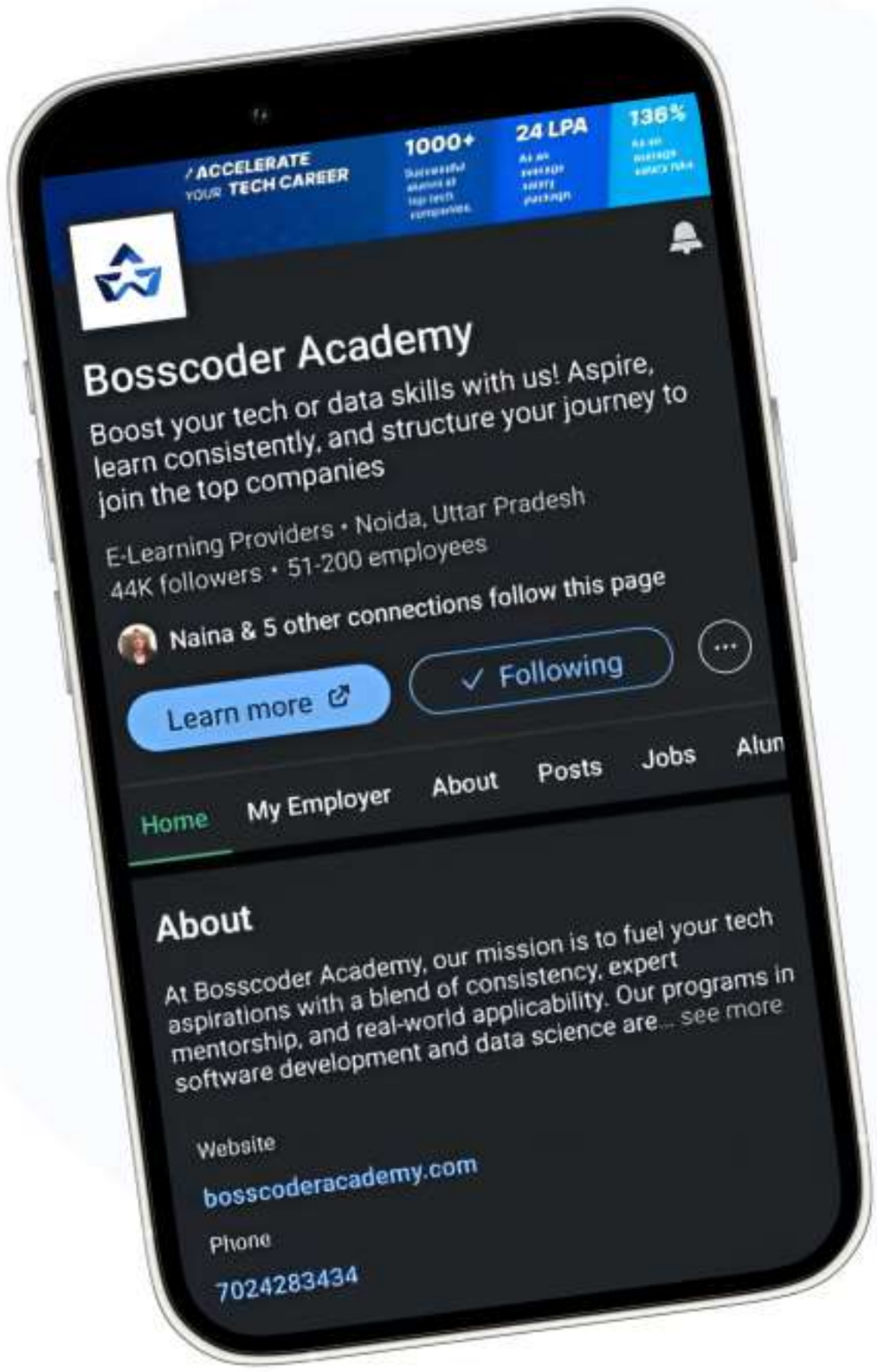
BOSSCODER ACADEMY

Help us reach

50K

Followers

Follow our profile
for more Informative
content



Save it
for later