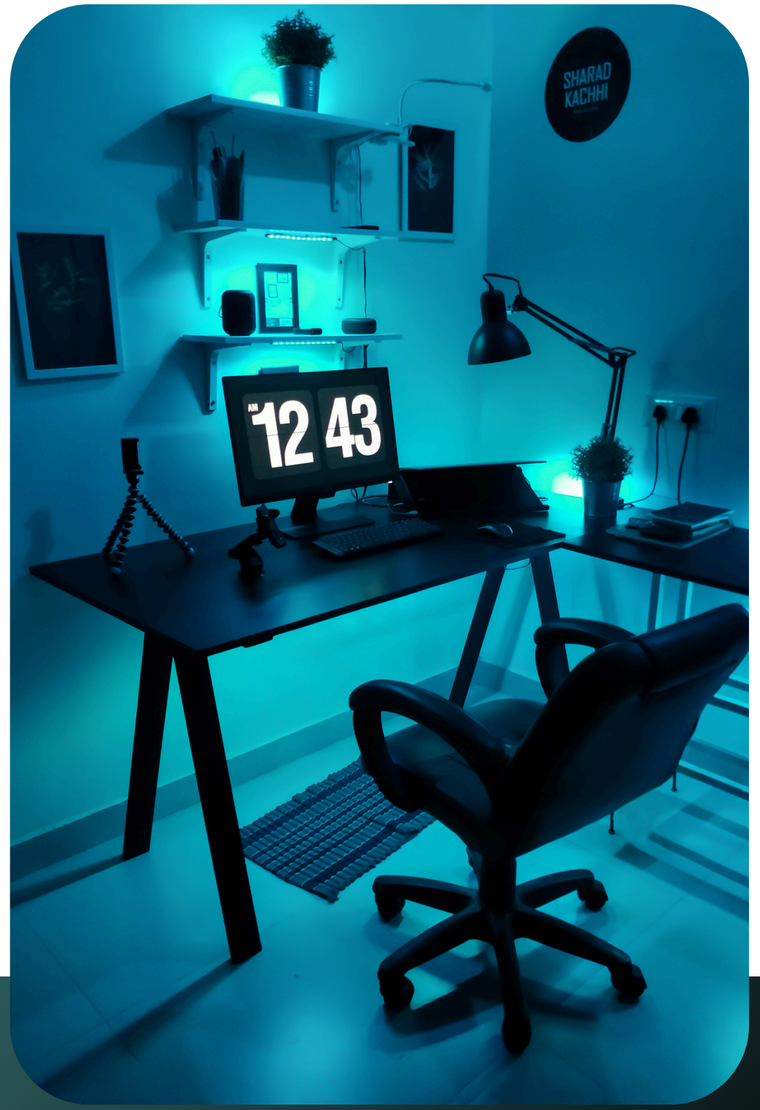
SOLVING THE SHORTEST PATH PROBLEM WITH DFS AND BFS

DATA STRUCTURES & ALGHORITMS PROJECT

FIRSTLY WHAT IS DFS AND BFS?

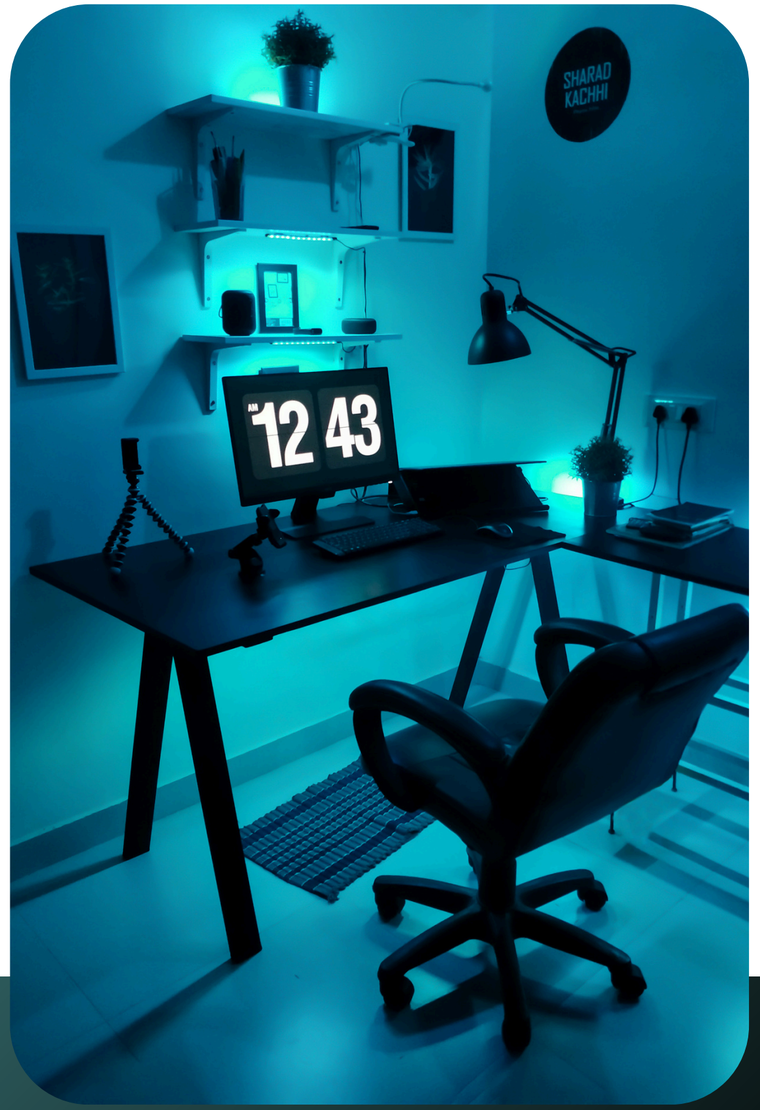
DEPTH-FIRST SEARCH(DFS)

- DFS is an algorithm used to traverse or search through the nodes of a graph or tree. It starts at a node and explores as far as possible along each branch before backtracking to explore other branches.



BREADTH-FIRST SEARCH(BFS)

BFS is an algorithm used to traverse or search through the nodes of a graph or tree by exploring all the nodes at the current depth level before moving to the next level. It starts from a specified source node, visiting all its immediate neighbors first, and then progresses outward level by level. This approach ensures that the shortest path from the source to any other node in an unweighted graph is found during traversal.



COMPARISON OF DFS AND BFS

DEPTH-FIRST SEARCH

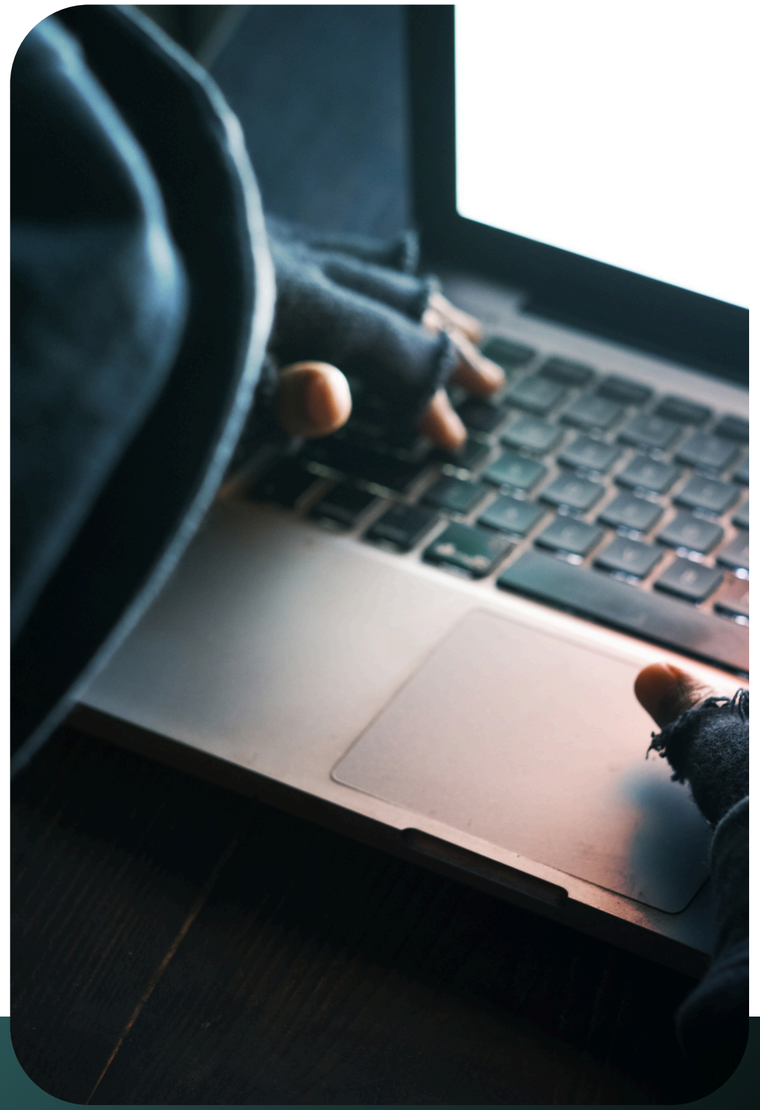
- Explores as deep as possible along each branch before backtracking.
- Uses a **stack**.
- Does not guarantee the shortest path in unweighted graphs.

BREADTH-FIRST SEARCH

- Explores all neighbors at the current level before moving to the next level.
- Uses a **queue**.
- Guarantees the shortest path in unweighted graphs.

CUSTOM STACK IMPLEMENTATION

We implemented a custom stack structure for the DFS algorithm, which requires a stack. A stack is a linear data structure that follows the LIFO (Last In, First Out) principle, where the last element added is the first to be removed. It can be visualized as a vertical collection of items, with elements added and removed only from the top.



Operations of Stacks:

- **Push**: Adds an element to the top of the stack.
- **Pop**: Removes and returns the element at the top of the stack.
- **Peek (or Top)**: Allows viewing the top element without removing it.
- **IsEmpty**: Checks whether the stack is empty.
- **Size**: Returns the number of elements in the stack.

CUSTOM QUEUE IMPLEMENTATION

- We implemented a custom queue structure for the BFS algorithm, which requires a queue. A queue is a linear data structure that follows the FIFO (First In, First Out) principle, where elements are added at the rear and removed from the front, ensuring an orderly sequence of operations.

Operations of Queue:

- **enqueue(element):** Adds an element to the rear of the queue.
- **dequeue():** Removes and returns the element at the front of the queue.
- **peek() (or front()):** Returns the element at the front of the queue without removing it.
- **isEmpty():** Checks whether the queue is empty.
- **size():** Returns the number of elements in the queue.

THANK YOU

NURBANU KARAKAYA 042201059
ATA SARGULLAR 042401175
KERİM ERCAN 042401167
ARDA TUNA KÜPÇÜ 042301093