1. Initialize an empty queue and a visited set to keep track of visited nodes.
2. Enqueue the start node into the queue and mark it as visited.
3. While the queue is not empty, dequeue a node from the queue and process it (e.g., print its value).
4. For each neighbour of the current node, if the neighbour has not been visited yet, mark it as visited and enqueue it into the queue.

It begins by traversing the graph from the root node and **explores all of the nodes in the immediate vicinity**. It chooses the closest node and then visits all of the nodes that have yet to be visited.

```
class Graph {
    constructor () {
        this.adjacencyList = {}
    }

    addVertex (v) {
        if (!this.adjacencyList[v]) this.adjacencyList[v] = []
    }

    addEdge (v1, v2) {
        this.adjacencyList[v1].push(v2)
        this.adjacencyList[v2].push(v1)
    }

    bfs (v) {
        const queue = [v]
        const visited = {}
        const result = []

        visited[v] = true

        while (queue.length) {
            const currVertex = queue.shift()
            result.push(currVertex)

            this.adjacencyList[currVertex].forEach(neighbour => {
                if (!visited[neighbour]) {
                    visited[neighabour] = true
                    queue.push(neighbour)
                }
            })
        }
        return result
    }
}

// Example usage:
const graph = new Graph();

graph.addVertex("A");
graph.addVertex("B");
```

```
graph.addVertex("C");
graph.addVertex("D");
graph.addVertex("E");
graph.addVertex("F");

graph.addEdge("A", "B");
graph.addEdge("A", "C");
graph.addEdge("B", "D");
graph.addEdge("C", "E");
graph.addEdge("D", "E");
graph.addEdge("D", "F");
graph.addEdge("E", "F");

console.log("BFS result:", graph.bfs("A"));
```