



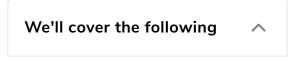






Solution Review: Implement Breadth First Search

This review provides a detailed analysis of the different ways to solve the breadth first search challenge.



- Solution: Using queues
- Time complexity

Solution: Using queues#

```
main.py

Graph.py

LinkedList.py

Node.py

Queue.py

Stack.py

1 from Graph import Graph
2 from Queue import MyQueue
3 from Stack import MyStack
4
5
6 def bfs_traversal_helper(g, source, visited):
7 result = ""
```

```
# Create Queue(implemented in previous lesson) for Breadth
                                                                            {}
9
        # and enqueue source in it
10
        queue = MyQueue()
11
        queue.enqueue(source)
        visited[source] = True # Mark as visited
12
        # Traverse while queue is not empty
13
        while not queue.is_empty():
14
15
            # Dequeue a vertex/node from queue and add it to result
16
            current_node = queue.dequeue()
            result += str(current_node)
17
            # Get adjacent vertices to the current_node from the list,
18
            # and if they are not already visited then enqueue them in th
19
20
            temp = g.array[current_node].head_node
21
            while temp is not None:
22
                if not visited[temp.data]:
23
                    queue.enqueue(temp.data)
24
                    visited[temp.data] = True # Visit the current Node
25
                temp = temp.next_element
26
        return result, visited
27
28
    def bfs_traversal(g, source):
```

For this solution, we will use the queue data structure that we studied earlier. bfs_traversal calls the helper function bfs_traversal_helper on every vertex which is not visited. Starting from the source vertex which is **0**, we insert the vertex into the gueue (line **11**). To keep track of where we have already traversed, every vertex inserted into the queue is marked visited in the visited list (line 12 and 24).

The result string will be our returning variable. The value of a node is appended to result (line 17) when it is dequeued from the queue (line 16). For each node that is dequeued, its adjacent nodes are added to the queue if they have not been visited (lines 21 to 25).

The **First In First Out** (FIFO) structure of the queue ensures that the graph is traversed one level at a time.

Time complexity#







Since this algorithm traverses the whole graph once, its time complexity is O(V + E).

Interviewing soon? We've partnered with Hired so that $$\times$$ companies apply to you instead of you applying to them. See $\underline{\text{how}}$ ①



Next

Next →

Challenge 1: Implement Breadth First ...

Challenge 2: Implement Depth First Se...



Mark as Completed



Report an Issue

