Challenge: Implement breadth-first search

Implement BFS

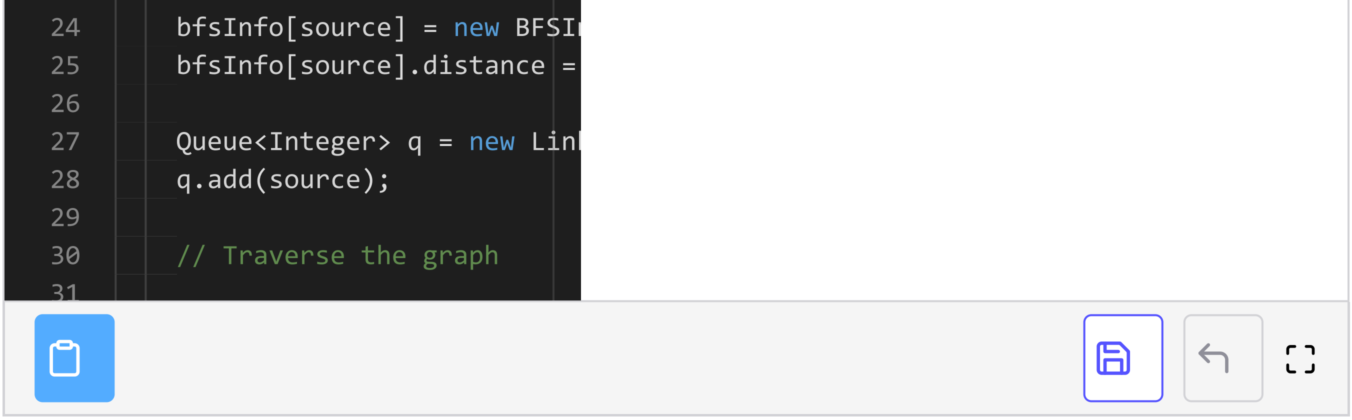
In this step, you'll finish implementing the doBFS function, which performs a breadth-first search on a graph and returns an array of objects describing each vertex.

For each vertex v, the object's distance property should be vertex v's distance from the source, and the predecessor property should be vertex v's predecessor on a shortest path from the source. If there is no path from the source to vertex v, then v's distance and predecessor should both be *null*. The source's predecessor should also be *null*.

In the starter code, the function initializes the distance and predecessor values to null, and then enqueues the source vertex. It is up to you to implement the rest of the algorithm, as described in the pseudocode.

Java | Python | C++ | JS

```
import java.util.LinkedList;
import java.util.Queue;

class BFSInfo {
  public BFSInfo() {
    this.distance = -1;
    this.predecessor = -1;
  }

  public BFSInfo(int distance,
    this.distance = distance;
    this.predecessor = predece
  }

  public int distance;
  public int predecessor;
};

class Solution {
  public static BFSInfo[] doBF
    System.out.println(graph.l
    BFSInfo[] bfsInfo = new BF

```



```
        bfsInfo[source] = new BFSI
        bfsInfo[source].distance =

        Queue<Integer> q = new Lin
        q.add(source);

        // Traverse the graph
```