src/General/Utility.java

```java
packageimportimportimportimportimport/**
 * A utility class providing common functionalities such as printing the
 * frontier,
 * constructing paths, and handling failure cases for graph traversal
 * algorithms.
 */publicclassUtility/**
     * Logs a failure message and the number of visited nodes when an
algorithm
     * cannot find a path.
     *
     * @parampublicstaticvoidalgorithmFails(int"fail"/**
     * Prints the current state of the frontier.
     *
     * @parampublicstaticvoidprintFrontier(Collection<Node> frontier)if
StringBuilderresult=newStringBuilderforif0","[""]"/**
     * Constructs the path from the goal node back to the start node using
the
     * parent map.
     *
     * @param@param@returnpublicstaticconstructPath(Node goal, Map<Node,
Node> parentMap)newArrayListforNodecurrent=nullreturn/**
     * Prints the path found by the BFS.
     *
     * @param@parampublicstaticvoidprintPath(List<Node> path, intifnull
"fail"elseNodelastNode=1"\n%.3f\n%d\n" General;

 java.util.ArrayList;
 java.util.Collection;
 java.util.Collections;
 java.util.List;
 java.util.Map;


   {

     visited The number of nodes that were visited during the execution of
     *                the algorithm.
     */</span>
        visited)</span> {
        System.out.println();
        System.out.println(visited);
    }

     frontier The collection of nodes currently in the frontier.
     */</span>
        {
         (!frontier.isEmpty()) {
                ();
             (Node node : frontier) {
                 (result.length() > )
                    result.append();
                result.append(node.toString());
            }
            System.out.println( + result + );
        }
    }

     goal      The goal node where the path ends.
```

```
     *  parentMap A map of child nodes to their parent nodes as discovered
by
     *                  the BFS.
     *  A list of nodes representing the path from the start to the goal.
     */</span>
      List<Node>  {
        List<Node> path =  <>();
         (   goal; current != ; current = parentMap.get(current)) {
            path.add(current);
        }
        Collections.reverse(path);
         path;
    }

     path         The list of nodes constituting the path.
     *  visitedCount The number of nodes visited during the search.
     */</span>
        visitedCount)</span> {
         (path ==  || path.isEmpty()) {
            System.out.println();
        }  {
            path.forEach(node -> System.out.print(node));
               path.get(path.size() - );
            System.out.printf(, lastNode.getCost(), visitedCount);
        }
    }

}

```