src/Algorithms/PartC_IDS.java

```
 1 packageimportimportimportimportimportimportimportimport/**
 2  * Implements the Iterative Deepening Search (IDS) algorithm, which
combines the
 3  * strengths of both Breadth-First Search and Depth-First Search by
repeatedly
 4  * executing a depth-limited search and increasing the depth limit with
each
 5  * iteration until the goal is found or the depth limit exceeds the
specified
 6  * maximum.
 7  */publicclassPartC_IDS/**
 8      * Performs a depth-limited search from a given node up to a specified
depth.
 9      *
10      * @param@param@param@param@param@param@returnprivatestatic
depthLimitedSearch(Node current, Node goal, intintif0returnelseif0returnnull
forif1ifnullreturnreturnnull/**
11      * Executes the Iterative Deepening Search algorithm using a starting
node and a
12      * goal node, iteratively deepening the depth of the search until the
goal is
13      * found or the depth exceeds the size of the planet.
14      *
15      * @param@param@param@returnpublicstaticiterativeDeepeningSearch(Node
start, Node goal, intforintdepth=0newHashMapnullnewHashSetifnullreturn"fail"
returnnull Algorithms;
16
17  java.util.Collections;
18  java.util.HashMap;
19  java.util.HashSet;
20  java.util.List;
21  java.util.Map;
22  java.util.Set;
23
24  General.Node;
25  General.Utility;
26
27
28    {
29      current    The current node from which to explore.
30      *  goal       The goal node to be reached.
31      *  depth      The maximum depth limit for this search iteration.
32      *  parentMap  A map to track the path from each node to its parent.
33      *  visited    A set of nodes that have already been visited in this
34      *                  search iteration.
35      *  planetSize The size of the planet, influencing node expansion
rules.
36      *  A list of nodes representing the path from the current node to the
37      *         goal if found within the depth limit; otherwise, null.
38      */</span>
39       List<Node>  depth, Map<Node, Node> parentMap,
40            Set<Node> visited,  planetSize)</span> {
41
42         (depth ==  && current.equals(goal)) {
43             Utility.constructPath(current, parentMap);
44        }   (depth == ) {
45             ;
46        }
```

```
47          visited.add(current);
48
49          List<Node> successors = current.getSuccessors(planetSize, goal);
50          Collections.sort(successors);
51
52           (Node node : successors) {
53               (!visited.contains(node)) {
54                  parentMap.put(node, current);
55                  List<Node> result = depthLimitedSearch(node, goal, depth
- , parentMap, visited, planetSize);
56                   (result != )
57                       result;
58              }
59          }
60           ;
61      }
62
63       start      The starting node of the search.
64       *  goal       The target goal node.
65       *  planetSize The size of the planet, used to cap the depth of search.
66       *  A list of nodes representing the path from the start to the goal if
67       *         found; otherwise, null.
68       */</span>
69        List<Node>  planetSize)</span> {
70           (   ; depth <= planetSize; depth++) {
71              Map<Node, Node> parentMap =  <>();
72              parentMap.put(start, );
73              Set<Node> visited =  <>();
74
75              List<Node> path = depthLimitedSearch(start, goal, depth,
parentMap, visited, planetSize);
76               (path != ) {
77                  System.out.println(visited);
78                  Utility.printPath(path, visited.size());
79                   path;
80              }
81          }
82          System.out.println();
83           ;
84      }
85 }
```