**Documentation laboratory work no. 2**

* **Breadth-first search:**

**def** bfs\_shortest\_path(self, start, goal):  
 explored = [] *#contains all the explored nodes  
 # keep track of all the paths to be checked* queue = [[start]] *#contains all the checked paths  
  
 # returns the path if start is goal* **if** start == goal:  
 msg = **"That was easy! Start = goal"  
 return** msg  
  
 *# keeps looping until all possible paths have been checked* **while** queue:  
 path = queue.pop(0) *#pop the first path from the queue* node = path[-1] *#get the last node from the path* **if** node **not in** explored:  
 neighbours = self.get\_outbound(node) *# go through all neighbour nodes,  
 # construct a new path and push it into the queue* **for** neighbour **in** neighbours:  
 new\_path = list(path)  
 new\_path.append(neighbour)  
 queue.append(new\_path)  
 *# return path if neighbour is goal* **if** neighbour == goal:  
 **return** new\_path  
  
 *# mark node as explored* explored.append(node)  
  
 *# in case there's no path between the 2 nodes* msg = **"A connecting path doesn't exist!"  
 return** msg  
  
**def** bfs(self):  
 start = input(**"Please input the first vertex: "**)  
 end = input(**"Please input the second vertex: "**)  
 self.validator.is\_integer(start) *#validates the input for the first veretx* self.validator.is\_integer(end) *#validates the input for the second vertex* l = self.bfs\_shortest\_path(int(start), int(end))  
 print(l) *#prints the list* **if** (l != **"A connecting path doesn't exist!" and** l != **"That was easy! Start = goal"**):  
 print(**"The length is: "**, len(l) - 1)