```
//Perform breadth-first search from initial state, using defined
 "is_goal_state"
//and "find_successors" functions
//Returns: null if no goal state found
//Returns: object with two members, "actions" and "states", where:
//  actions: Sequence(Array) of action ids required to reach the goal state
 from the initial state
//  states: Sequence(Array) of states that are moved through, ending with the
 reached goal state (and EXCLUDING the initial state)
//  The actions and states arrays should both have the same length.

/**
 * AUTHORS: John Choi and Austin Schall
 */

function breadth_first_search(initial_state) {
  let open = []; //See push()/pop() and unshift()/shift() to operate like
   stack or queue
                //https://developer.mozilla
                 .org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
  let closed = new Set();
   //https://developer.mozilla
   .org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

  // save trail
  var nodeHistory = []
  var actionHistory = []

  let currentNode = {
    currentState: initial_state,
    children: find_successors(initial_state)
  }

  while (!is_goal_state(currentNode.currentState)) {
    // add it to open
    open.push(state_to_uniqueid(currentNode.currentState))

    var moveToNext = false // boolean used to terminate for loop
    // iterate through children
    for (var i = 0; i < currentNode.children.length && !moveToNext; i++) {
      // find first child that is not visited yet
      if (!closed.has(state_to_uniqueid(currentNode.children[i].resultState))
       &&
       !open.includes(state_to_uniqueid(currentNode.children[i].resultState)))
       {

        // child not visited yet
        let nextNode = Object.assign({}, currentNode) // copy current node
        // save action ID
        actionHistory.push(currentNode.children[i].actionID)
        currentNode = {
```

```
        currentState: nextNode.children[i].resultState,
        children: find_successors(nextNode.children[i].resultState)
      }
      nodeHistory.push(currentNode)
      moveToNext = true
      break
    }
  }
  if (moveToNext) {
    continue
  }
  // at this point, no available child was selected
  // add current node to closed and go up
  let id = state_to_uniqueid(currentNode.currentState)
  closed.add(state_to_uniqueid(currentNode.currentState))
  open.pop()
  actionHistory.pop()
  let prevNode = nodeHistory.pop()
  currentNode = {
    currentState: prevNode.currentState,
    children: find_successors(prevNode.currentState)
  }
}
nodeHistory.push(currentNode)

/*
  Hint: In order to generate the solution path, you will need to augment
    the states to store the predecessor/parent state they were generated from
    and the action that generates the child state from the predecessor state.

    For example, make a wrapper object that stores the state, predecessor
     and action.
    Javascript objects are easy to make:
      let object={
          member_name1 : value1,
          member_name2 : value2
      };

  Hint: Because of the way Javascript Set objects handle Javascript objects,
   you
    will need to insert (and check for) a representative value instead of
     the state
    object itself. The state_to_uniqueid function has been provided to help
     you with
    this. For example
      let state=...;
      closed.add(state_to_uniqueid(state)); //Add state to closed set
      if(closed.has(state_to_uniqueid(state))) { ... } //Check if state is
       in closed set
*/
```

```
  /***Your code to generate solution path here***/

  var actionsToGoal = []
  var statesToGoal = []

  for (var i = 0; i < nodeHistory.length; i++) {
    let state = nodeHistory[i].currentState
    statesToGoal.push(state)
  }
  for (var i = 0; i < actionHistory.length; i++) {
    let actionID = actionHistory[i]
    actionsToGoal.push(actionID)
  }

  if (actionsToGoal.length == 0) {
    return null
  }

  return {
    actions : actionsToGoal,
    states : statesToGoal
  }
}
```