// Your code has clarity and good readability

// The indentations look fine

/ The design of the program looks fine and easy to read. I would like to suggest creating a separate function that checks if the given grid is valid instead of checking the validity in the main function.

// The program looks optimized meaning the program is efficient in terms of speed and storage

// Variable name suggestions:

         1. solution\_Set to solution\_set

// The program has been divided into different components (functions) like convertToLowerCase, findWords, etc. which makes the code easy to modify or change

// Comments look consitent with code

exports.findAllSolutions = function (grid, dictionary) {  
  let solutions\_Set = new Array();  
  let solutions = [];

  // writing the following if statement in the first line in the body of the function will be better. That way you do not have to create variables if the grid or dictionary is empty and you will save memory resources  
  if (grid == null || dictionary == null) { //checks of grid or dict is empty

    // you can write the following as return [];

    return solutions;  
  }

  // checks if the grid is NxN matrix  
  for (let i = 0; i < grid.length; i++) {  
    if (grid[i].length != grid.length) {  
    return solutions;  
    }  
  }

  // good practice in naming the function  
  convertToLowerCase(grid, dictionary)

  // If your intention is to create a trie data structure use the following function to create the trie data structure from the given dictionary

// This function creates a trie data structure from the given dictionary

function create\_trie (dictionary) {

  var result = {};

  for (var i = 0; i < dictionary.length; i++) {

    result[dictionary[i]] = 1;

    var leng = dictionary[i].length;

    var str = dictionary[i];

    for (var j = leng; leng > 1; leng--) {

      str = str.substr(0, leng - 1);

      if (str in result) {

        if (str == 1) {

          result[str] = 1;

        }

      } else {

        result[str] = 0;

      }

    }

  }

  return result;

};

  //let trie = createTrie(dictionary)  
  let trie = new Set(dictionary);

  // to create NxN matrix from grid containing all elements as 0  
  for (let y = 0; y < grid.length; y++) {  
    for (let x = 0; x < grid.length; x++) {  
    let visited = new Array(grid.length).fill(false).map(() => new Array(grid.length).fill(false));  
    let word = [];//is empty to start  
  findWords(word, grid, trie, x, y, visited, solutions\_Set);  
    }  
  }  
  solutions = Array.from(solutions\_Set);  
  return solutions;  
}

function findWords(word, grid, trie, y, x, visited, solutions\_Set) { //this is the recursive call  
  const adjacent\_lookup = [[-1, -1],  
  [-1, 0],  
  [-1, 1],  
  [0, 1],  
  [1, 1],  
  [1, 0],  
  [1, -1],  
  [0, -1]];

  //given a wor and grid[y][x] and visited[y][x]  
  //Base cases:  
  // y/x out of bounds  
  // it already visited grid [y][x]

 // base case  
  if (y < 0 || y >= grid.length || x < 0 || x >= grid.length || visited[y][x] == true)  
    return;  
  //1. append grid [y][x] to the word  
  word += grid[y][x]; //word = word + grid

  //2. check if word is a prefix  
  if (isPrefix(trie, word)) {  
    visited[y][x] = true;  
    //2a. check if prefix is marked as visited  
    //2b. test if word is valid  
  if (isWord(trie, word)) {  
  //2b. if word is valid: word is added to solution\_Set