- I like that u decided to implement a trie instead of a hash function.
- I would say some of your variable names could be a little clearer.For example:
- For example: checkPre could be a little clearer. Also, the function makeLower could also be a bit clearer when defining.

I cleaned up the spacing and tried to add more comments to help make the program more clear.

```
/**
* Given a Boggle board and a dictionary, returns a list of
available words in
* the dictionary present inside of the Boggle board.
* @param {string[][]} grid - The Boggle game board.
* @param {string[]} dictionary - The list of available words.
* @returns {string[]} solutions - Possible solutions to the Boggle
board.
*/
exports.findAllSolutions = function(grid, dictionary) {
 let solutions = [];

 //declaring
 makeLower(grid,dictionary);
 let trie = MakeTrie(dictionary);

 if (grid == null || dictionary == null) {
   return solutions;
 }

 let len_of_grid = grid.length;
 //cross checks
 if (len_of_grid == 0) {
   return solutions
```

```
 }

 for(let x = 0; x < len_of_grid; x++)
 {
   if (grid[x].length != len_of_grid)
   {
     return solutions
   }
 }

 let AllSolutions = new Set();
   for (let b = 0; b < len_of_grid; b++)
   {
     for (let a = 0; a < len_of_grid; a++)
     {
       let word = "";
       let given = new Array(len_of_grid).fill(false).map(() => new
Array(len_of_grid).fill(false));

       checkWord(word,a,b,grid,given,trie,AllSolutions);

     }
   }

 solutions = Array.from(AllSolutions);

 return solutions;
}

var tNodes = function(value)
{
 this.value = value;
 this.newArr = new Array();
```

```
 this.wordChecker = false;
}

var MakeTrie = function(dict) {
 var root = new tNodes('');

 if(dict.length == 0) {
   return;
 }

 for(let words of dict)
 {
   var node = root;
   for(let x = 0; x < words.length; x++)
   {
     var character = words[x];
     var numberAt = character.charCodeAt(0) - 97;

     var currNode = node.newArr[numberAt];
     if (node.newArr[numberAt] == undefined)
     {
       var currNode = new tNodes(character);
       node.newArr[numberAt] = currNode;

     }

     node = currNode;
   }
   node.wordChecker = true;
 }
 return root;
}
```

```
checkWord = function(word,a,b,grid,given,trie,AllSolutions)
{
   // makes sure it is in bounds
 let coords =
[[0,1],[1,0],[0,-1],[-1,0],[1,1],[-1,1],[1,-1],[-1,-1]];

 if(b < 0 || a < 0 || b >= grid.length || a >= grid.length ||
given[a][b] == true)
 {
   return;
 }

 word += grid[a][b];

 if(checkPre(word, trie))
 {
   given[a][b] = true;
   if (wordChecker(word, trie))
   {
     if( word.length >= 2)
     {
       AllSolutions.add(word);

     }
   }
 for( let t = 0; t < 8; t++)
 {
   checkWord(word,a + coords[t][0], b + coords[t][1], grid, given,
trie, AllSolutions)
 }

 }
```

```
 given[a][b] = false;
}

checkPre = function(word, trie)
{
 //checking the grid
 let newWord = '';
 let currNode = trie;

 for(let t = 0; t < word.length; t++)
 {
   if(currNode != undefined)
   {
     for(let node of currNode.newArr)
     {
       if(node != undefined && node.value == word[t])
       {
         newWord += word[t];
         currNode = node;
         break;
       }
     }
   }
 }
 if(word == newWord)
 {
   return true;
 }
 return false;
}
//Check if its a word

wordChecker = function(word,trie) {
```

```
// makes sure characters match up
let newWord = '';
let currNode = trie;
for( let t = 0; t < word.length; t++)
{
 if(currNode != undefined)
 {
   for(let node of currNode.newArr)
   {
     if(node != undefined && node.value == word[t])
     {
       newWord += word[t];
       currNode = node;
       break;
     }
   }
 }
}
if(word == newWord && currNode.wordChecker == true)
{
 return true;
}
return false;
}

makeLower = function(grid,dict) {
 for(let m = 0; m < grid.length; m++)
 {
   for(let n = 0; n < grid.length; n++)
   {
     if(grid[m][n])
     {
```

```
        grid[m][n] = grid[m][n].toLowerCase();
      }
    }
  }

  for(let n = 0; n < dict.length; n++)
  {
    dict[n] = dict[n].toLowerCase();
  }
}

var grid = [['T', 'W', 'Y', 'R'],
            ['E', 'N', 'P', 'H'],
            ['G', 'Z', 'Qu', 'R'],
            ['St', 'N', 'T', 'A']];
var dictionary = ['art', 'ego', 'gent', 'get', 'net', 'new', 'newt',
'prat',
                  'pry', 'qua', 'quart', 'quartz', 'rat', 'tar',
'tarp',
                  'ten', 'went', 'wet', 'arty', 'egg', 'not',
'quar'];
//console.log(exports.findAllSolutions(grid, dictionary));
```