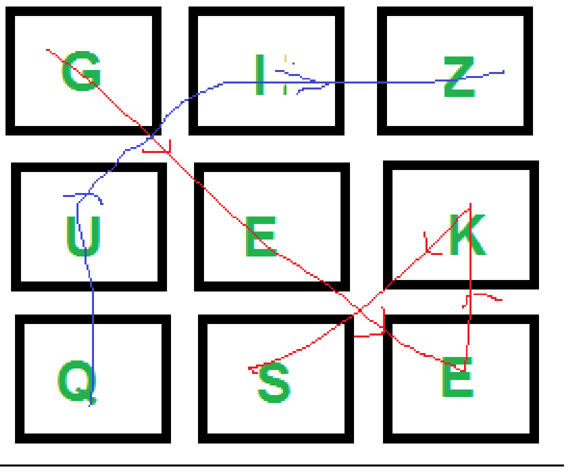
Boggle (Find all possible words in a board of characters)

Given a dictionary, a method to do lookup in dictionary and a M x N board where every cell has one character. Find all possible words that can be formed by a sequence of adjacent charactersNote that we can move to any of 8 adjacent characters, but a word should not have multiple instances of same cell.

Example:

```
Input: dictionary[] = {"GEEKS", "FOR", "QUIZ", "GO"};
       boggle[][]   = {{'G','I','Z'},
                       {'U','E','K'},
                       {'Q','S','E'}};
      isWord(str): returns true if str is present in dictionary
                   else false.

Output:  Following words of dictionary are present
         GEEKS
         QUIZ
```



We strongly recommend to minimize your browser and try this yourself first.

The idea is to consider every character as a starting character and find all words starting with it. All words starting from a character can be found using Depth First Traversal. We do depth first traversal starting from every cell. We keep track of visited cells to make sure that a cell is considered only once in a word.

```
// C++ program for Boggle game
#include<iostream>
#include<cstring>
using namespace std;

#define M 3
#define N 3

// Let the given dictionary be following
string dictionary[] = {"GEEKS", "FOR", "QUIZ", "GO"};
int n = sizeof(dictionary)/sizeof(dictionary[0]);

// A given function to check if a given string is present in
// dictionary. The implementation is naive for simplicity. As
// per the question dictionary is givem to us.
bool isWord(string &str)
```

```
{
    // Linearly search all words
    for (int i=0; i<n; i++)
        if (str.compare(dictionary[i]) == 0)
          return true;
    return false;
}

// A recursive function to print all words present on boggle
void findWordsUtil(char boggle[M][N], bool visited[M][N], int i,
                   int j, string &str)
{
    // Mark current cell as visited and append current character
    // to str
    visited[i][j] = true;
    str = str + boggle[i][j];

    // If str is present in dictionary, then print it
    if (isWord(str))
        cout << str << endl;

    // Traverse 8 adjacent cells of boggle[i][j]
    for (int row=i-1; row<=i+1 && row<M; row++)
      for (int col=j-1; col<=j+1 && col<N; col++)
        if (row>=0 && col>=0 && !visited[row][col])
          findWordsUtil(boggle,visited, row, col, str);

    // Erase current character from string and mark visited
    // of current cell as false
    str.erase(str.length()-1);
    visited[i][j] = false;
}

// Prints all words present in dictionary.
void findWords(char boggle[M][N])
{
    // Mark all characters as not visited
    bool visited[M][N] = {{false}};

    // Initialize current string
    string str = "";

    // Consider every character and look for all words
    // starting with this character
    for (int i=0; i<M; i++)
       for (int j=0; j<N; j++)
             findWordsUtil(boggle, visited, i, j, str);
}

// Driver program to test above function
int main()
{
    char boggle[M][N] = {{'G','I','Z'},
                         {'U','E','K'},
                         {'Q','S','E'}};

    cout << "Following words of dictionary are present\n";
    findWords(boggle);
    return 0;
}
```

Run on IDE

Output:

```
Following words of dictionary are present
GEEKS
```

```
QUIZ
```

Note that the above solution may print same word multiple times. For example, if we add “SEEK” to dictionary, it is printed multiple times. To avoid this, we can use hashing to keep track of all printed words.