Orasi Monthly Code Challenge

Gold League

July 2013

Challenge Submitted by:

Matt Watson

Challenge Winner:

Ethan Bell

July 2013 Orasi Code Challenge - Gold

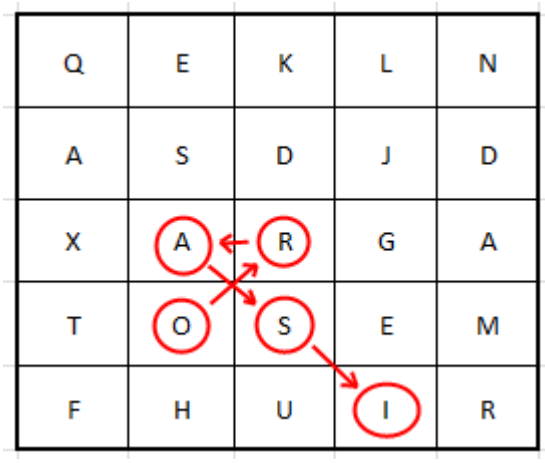
Challenge Description

In the board game Boggle players are asked to find as many words as they can from a 4x4 grid of random letters. To create words, players may start at any letter in the grid and can move to any connecting letter (horizontally, vertically, and diagonally). The player that finds the most words wins. By extending the Boggle board to an N x N size board with random letters and searching for all the possible word combinations, we arrive at an interesting computational problem. The Gold League challenge for July 2013 is to find all the possible Boggle style word combinations in an N x N array.

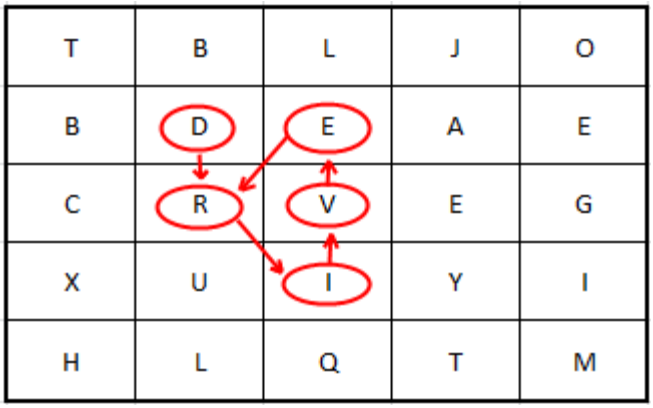
Boggle Rules

In Boggle, the goal is to find as many words as possible given a square board of random letters. Words can be created by starting at any position on the board and moving horizontally, vertically, or diagonally. Each letter on the board can only be used once for a given word. Only words of three letters or more are considered valid.

A Legal Word Combination



An Illegal Word Combination



The definitive word list for this challenge can be found at the following location.

http://web.mit.edu/kilroi/Public/bot/wordlist

Your script will need to access this word list, from this location, and use it to determine the possible word combinations. Your script may save a local copy of the list, but the copy must be removed before the script completes. **Use of this word list is required**

Submission Requirements:

Your submission must be in the form of a VBScript function that can run inside QTP. The function should accept one argument (an N x N array) and return a semicolon delimited string containing all the possible word combinations in alphabetical order. The function name should be your name. The function must comply with Orasi Coding Standards. You will be allowed 15 minutes of execution time to accomplish this task. Anyone who successfully returns the correct word list will be considered to have completed the challenge. The code that completes the task the fastest and meets coding standards will be the winner of the challenge.

Winning Solution:

```
Public Function EthanBell(aBoard)

	'*********************************************************************
	'Variable Declaration
	'*********************************************************************

	'String that will be built from answers as they're found, then rebuilt after array is sorted.
	Dim answers
	'Array of words found in boggle board. Put into this array for sorting purposes.
	Dim aAnswers
	'Reference to a browser object. Used to call the page and status bar objects.
	Dim browserDesc
	'Regular expression object to be sure a given word has the letters we're interested in.
	Dim checkLettersRegex
	'Counter used when iterating over columns of the 2D array from input.
	Dim column
	'currentPosition is used in sorting, when iterating down the currently sorted part of the list to see where to insert.
	Dim currentPosition
	'Used to hold each word in the matches collection when looping to add words to dictionary.
	Dim match
	'The number of elements expected in each row.
	Dim numCols
	'The number of rows found in the array.
	Dim numRows
	'Trie that will be used for checks against the dictionary of valid words.
	Dim oDictionary
	'Description and reference to the page object, used to pull text from the word list.
	Dim pageDesc
	'Counter used when iterating over rows of the board.
	Dim row
	'When sorting, indicates the end of the sorted list
	Dim sortedLength
	'Description of Internet Explorer's status bar; used to determine when loading is done.
	Dim statusDesc
	'temp is used purely as temporary storage to swap values while sorting.
	Dim temp
	'String that will end up containing one instance of each letter found on the boggle board.
	Dim validLetters
	'Will reference a collection of match objects the regular expression will return after testing against the word list.
	Dim validWords

	'*********************************************************************
	'Variable Assignment
	'*********************************************************************
	answers = ""
	Set checkLettersRegex = New RegExp
```

```
	checkLettersRegex.IgnoreCase = True
	checkLettersRegex.Global = True
	numCols = uBound(aBoard, 2)
	numRows = uBound(aBoard, 1)
	validLetters = ""
	Set oDictionary = New Trie

	'**********************************************************************
	'Business Process
	'**********************************************************************
	'Use letters present in Boggle board to setup regular expression.
	For row=0 To numRows
		For column=0 To numCols
			If InStr(validLetters, aBoard(row,column)) = 0 Then
				validLetters = validLetters & aBoard(row,column)
			End If
		Next
	Next

	'Regular expression looking for words with valid letters, length 3 and up.
	'Determines if something is a word or not by having either newlines or beginning or end of input at edges.
	checkLettersRegex.Pattern = "\b["&validLetters&"]{3,}\b"&"|^["&validLetters&"]{3,}\b"&"|\b["&validLetters&"]{3,}$"

	'Open browser to wordlist.
SystemUtil.Run "iexplore.exe", "http://web.mit.edu/kilroi/Public/bot/wordlist"

	'Create empty description of browser; depends on having just one browser available.
	Set browserDesc = Description.Create
	browserDesc("title").Value = "http://web.mit.edu/kilroi/Public/bot/wordlist"
	browserDesc("index").Value = "0"

	'Describe IEstatus bar.
	Set statusDesc = Description.Create
	statusDesc("nativeclass").Value = "msctls_statusbar32"

	'Describe page object
	Set pageDesc = Description.Create
	pageDesc("url").Value = "http://web.mit.edu/kilroi/Public/bot/wordlist"

	Do While Not WinStatusBar(statusDesc).getROProperty("text")="Done"
		'Waits until page loading is done.
	Loop

	'Use regular expression to pull out words that have only letters found in the boggle board. Very helpful for small boards, or those
missing any common letters and vowels.
	Set validWords = checkLettersRegex.Execute(Browser(browserDesc).Page(pageDesc).Object.DocumentElement.innertext)

	'Browser objects are no longer useful.
	Browser(browserDesc).Close
	Set browserDesc = Nothing
	Set statusDesc = Nothing
	Set pageDesc = Nothing
	Set checkLettersRegex = Nothing

	'Store each match found in a trie. This allows for fast checking, and finds prefixes that lead to nothing early.
	For Each match In validWords
		oDictionary.AddWord oDictionary.root, match.Value
	Next

	'For each letter on board, explore possible paths. Assumes rectangular grid, and at least one row.
	For row=0 To numRows
		For column=0 To numCols
			Expand aBoard, row, column, "", "", oDictionary, answers
		Next
	Next
```

```
	Set oDictionary = Nothing

	If answers = "" Then
		EthanBell = ""
		Exit Function
	End If

	'Now all answers are in a string delimited by semicolons. Split for sorting.
	aAnswers = Split(Left(answers, Len(answers)-1), ";")

	'Insertion sort

	'Iterate over the positions of the array.
	For sortedLength=1 to uBound(aAnswers)
		currentPosition=sortedLength
		Do While currentPosition>0
			'See if the current word is smaller than the one in front of it. If so, fix that.
			If StrComp(aAnswers(currentPosition), aAnswers(currentPosition-1), 1)=-1 Then
				'Swap the words.
				temp = aAnswers(currentPosition)
				aAnswers(currentPosition)=aAnswers(currentPosition-1)
				aAnswers(currentPosition-1)=temp

				'Continue down the array, toward the front.
				currentPosition = currentPosition-1
			Else
				'If this item is already in the right place, skip to next.
				Exit Do
			End If

		Loop
	Next

	'Loop through now sorted array to build string return value.
	answers = ""
	For Each temp in aAnswers
		answers = answers & temp & ";"
	Next

	'**************************************************************************
	'Output assignment
	'**************************************************************************
	EthanBell = Left(answers, Len(answers)-1)
End Function



'**************************************************************************
'Assisting Classes
'**************************************************************************

'Trie class, used to efficiently check possibilities for matches from the word list.
Class Trie
  'Initial node
	Public root
	'Field that will be reused to refer to the chopped off first letter of a given input.
	Private firstLetter

	Private Sub Class_Initialize()
		Set root = new Node
		firstLetter = ""
	End Sub

	Private Sub Class_Terminate()
		Set root = Nothing
	End Sub
```

```
	'Puts a new word in the trie, allowing for validity check.
	'From outside, always call using [thisTrie].root for first param.
	Public Function AddWord(fromNode, ByVal word)

	  'If at the end of a word, increase parent node's word count to indicate word end.
		If word="" Then
			fromNode.words = fromNode.words+1

		'Otherwise, consider parent (and its path) as a prefix, and add the rest of the word.
		Else
			fromNode.prefixes = fromNode.prefixes+1
			firstLetter = Left(word, 1) 'Save first letter to chedk for edge.
			If isEmpty(fromNode.Child(firstLetter)) Then
				fromNode.ActivateChild(firstLetter)		'Activate edge to next letter if it is dormant.
			End If

			'Recursive call to add the rest of the word from the node just created.
			word = Right(word, Len(word)-1) 'Chop off the first letter of word for recursion.
			AddWord fromNode.Child(firstLetter), word
		End If
	End Function

	'Find out if a word is an entry in this trie/dictionary.
	'From outside, always call using [thisTrie].root for first param.
	Public Function CheckWord(fromNode, ByVal word)
		firstLetter = Left(word, 1)
		If word = "" Then
			CheckWord = CBool(fromNode.words)
			Exit Function
		ElseIf isEmpty(fromNode.Child(firstLetter)) Then
			CheckWord = False
			Exit Function
		Else
			word = Right(word, Len(word)-1)
			CheckWord = CheckWord(fromNode.Child(firstLetter), word)
		End If
	End Function

	'Check a given prefix, to see if it leads to any further words.
	Public Function CountWordsFromPrefix(fromNode, ByVal prefix)
		firstLetter = Left(prefix, 1)
		If firstLetter="" Then
			CountWordsFromPrefix = fromNode.prefixes
		ElseIf isEmpty(fromNode.Child(firstLetter)) Then
			CountWordsFromPrefix = 0
		Else
			prefix = Right(prefix, Len(prefix)-1)
			CountWordsFromPrefix = CountWordsFromPrefix(fromNode.Child(firstLetter), prefix)
		End If
	End Function

End Class

'A single node in the trie. Converted between letter and index into array of parent's children.
Class Node

	Private aChildren 'Links to child nodes.
	Public words 'Running count of words how many words end at this node. Should generally be 1 or 0.
	Public prefixes 'How many words this node is a prefix of.

	'Constructor to initialize values.
	Private Sub Class_Initialize()
		ReDim aChildren(25) 'One possible outgoing edge for each letter
		words = 0
		prefixes = 0
```

```
	End Sub

	'Property to work with the array of children/edges.
	Public Property Get Child(name)
		If isEmpty(aChildren(letterToNum(name))) Then
			Child = Empty
		Else
			Set Child = aChildren(letterToNum(name))
		End If
	End Property

	'Takes a letter as a string, enables child of the corresponding index.
	Public Sub ActivateChild(name)
		Set aChildren(letterToNum(name)) = New Node
	End Sub
End Class

'**********************************************************************
'Private Functions
'**********************************************************************

'Explore the grid, abadoning if no more words have a given prefix.
Private Function Expand(grid, ByVal rw, ByVal co, ByVal path, ByVal word, dict, answers)
  'Counters used in a for loop. Represent directions that can be travelled along x and y.
	Dim i
	Dim j

	'Build the addtions to word and path that come from visiting this point in the board/graph/grid.
	word = word & grid(rw,co)
	path = path & "(" & rw & "," & co & ")"


	'If the word built so far is in the dictionary, append it to answers.
	If dict.CheckWord(dict.root, word) Then

		'Regular expression object;
		Dim DuplicateCheck

		'First check to ensure that the word isn't a duplicate.
		Set DuplicateCheck = New RegExp

		'Look for word at beginning, or within semicolons.
		DuplicateCheck.Pattern = "^"&word&";|;"&word&";"
		DuplicateCheck.IgnoreCase = True

		'If a non-duplicate is found, add it to the string of answers.
		If Not DuplicateCheck.Test(answers) Then
			answers = answers & word & ";"
		End If

		Set DuplicateCheck = Nothing
	End If


	'See if continuing this path can lead to anything, if not stop.
	If dict.CountWordsFromPrefix(dict.root, word) = 0 Then
		Exit Function
	End If

	'Loop over possible neighboring directions.
	For i=-1 To 1
		For j=-1 To 1

			'Make sure the move we get ready to look at is in bounds.
			If rw+i>=0 AND rw+i<=uBound(grid,1) AND co+j>=0 AND co+j<=uBound(grid,2) Then
```

```
                                        'Make sure the node we're about to expand hasn't been visited on this path
                                        If InStr(path, "(" & (rw+i) & "," & (co+j) & ")")=0 Then

                                                'If the node is an in-bounds neighbor not yet visited, visit it.
                                                Expand grid, rw+i, co+j, path, word, dict, answers
                                        End If
                                End If
                        Next
                Next
End Function

'Pass in a letter, returns the corresponding number [0-25]
Private Function LetterToNum (letter)
  Select Case uCase(letter)
                        Case "A"
                                LetterToNum=0
                        Case "B"
                                LetterToNum=1
                        Case "C"
                                LetterToNum=2
                        Case "D"
                                LetterToNum=3
                        Case "E"
                                LetterToNum=4
                        Case "F"
                                LetterToNum=5
                        Case "G"
                                LetterToNum=6
                        Case "H"
                                LetterToNum=7
                        Case "I"
                                LetterToNum=8
                        Case "J"
                                LetterToNum=9
                        Case "K"
                                LetterToNum=10
                        Case "L"
                                LetterToNum=11
                        Case "M"
                                LetterToNum=12
                        Case "N"
                                LetterToNum=13
                        Case "O"
                                LetterToNum=14
                        Case "P"
                                LetterToNum=15
                        Case "Q"
                                LetterToNum=16
                        Case "R"
                                LetterToNum=17
                        Case "S"
                                LetterToNum=18
                        Case "T"
                                LetterToNum=19
                        Case "U"
                                LetterToNum=20
                        Case "V"
                                LetterToNum=21
                        Case "W"
                                LetterToNum=22
                        Case "X"
                                LetterToNum=23
                        Case "Y"
                                LetterToNum=24
                        Case "Z"
                                LetterToNum=25
```

```
                Case Else
                        MsgBox "Input must be character. [" & letter & "] is not a letter."
  End Select
End Function

'*********************************************************************
'End File
'*********************************************************************
```