

Boggle Solver

Credit Todd Feldman for the original idea behind the assignment. Adapted from handouts of Julie Zelenski and Eric Roberts.

Boggle Solver

The Boggle board is a 4x4 grid onto which you shake and randomly distribute 16 dice. These 6-sided dice have letters rather than numbers, creating a grid of letters from which you can form words. In the original version, the players start simultaneously and write down all the words they can find by tracing by a path through adjoining letters. Two letters adjoin if they are next to each other horizontally, vertically, or diagonally. There are up to eight letters adjoining a cube. A grid position can only be used once in the word. When time is called, duplicates are removed from the players' lists and the players receive points for their remaining words based on the word lengths. In this assignment, you will be creating a program that will find all the words on a boggle board.

This assignment is broken into two parts. The first part of the program will be creating a dictionary that can be used to store and look up words. This dictionary implementation will use a special tree called a prefix tree.

```
C:\Users\Erickson\Documents\Visual Studio 2015\Projects\S
127142 words loaded.

Enter Board
-----
Row 0: a u c o
Row 1: n l n i
Row 2: o s a e
Row 3: m a i e

Show Board (y/n)?: y
Word: ala
Number of Words: 1
'a' u c o 1 0 0 0
n 'l' n i 0 2 0 0
o s 'a' e 0 0 3 0
m a i e 0 0 0 0

Word: alan
Number of Words: 2
'a' u c o 1 0 0 0
n 'l' 'n' i 0 2 4 0
o s 'a' e 0 0 3 0
m a i e 0 0 0 0

Word: alane
Number of Words: 3
'a' u c o 1 0 0 0
n 'l' 'n' i 0 2 4 0
o s 'a' 'e' 0 0 3 5
m a i e 0 0 0 0
```

```
C:\Users\Erickson\Documents\Visua
127142 words loaded.

Enter Board
-----
Row 0: a u c o
Row 1: n l n i
Row 2: o s a e
Row 3: m a i e

Show Board (y/n)?: n
1 ala
2 alan
3 alane
4 alans
5 alae
6 alas
7 als
8 also
9 anlas
10 ansa
11 ansae
12 anoa
13 anoas
14 unco
15 unci
16 uncia
17 unciae
18 uncial
19 uncials
20 unai
```

Part 4b: Boggle Solver

Once you have completed and tested your Dictionary class, your next task is creating the Boggle solver.

User Input

After creating and loading the dictionary, you should prompt the user for input:

```
C:\Users\Erickson\Documents\Visual Studio 2015\Projects\S
127142 words loaded.

Enter Board
-----
Row 0: a u c o
Row 1: n l n i
Row 2: o s a e
Row 3: m a i e
```

After initializing the board, you should ask the user whether or not to print the board when finding the solutions:

```
C:\Users\Erickson\Documents\Visual Studio 2015\Projects\S
127142 words loaded.

Enter Board
-----
Row 0: a u c o
Row 1: n l n i
Row 2: o s a e
Row 3: m a i e

Show Board (y/n)?: y
Word: ala
Number of Words: 1
'a' u c o 1 0 0 0
n 'l' n i 0 2 0 0
o s 'a' e 0 0 3 0
m a i e 0 0 0 0

Word: alan
Number of Words: 2
'a' u c o 1 0 0 0
n 'l' 'n' i 0 2 4 0
o s 'a' e 0 0 3 0
m a i e 0 0 0 0

Word: alane
Number of Words: 3
'a' u c o 1 0 0 0
n 'l' 'n' i 0 2 4 0
o s 'a' 'e' 0 0 3 5
m a i e 0 0 0 0
```

```
C:\Users\Erickson\Documents\Visua
127142 words loaded.

Enter Board
-----
Row 0: a u c o
Row 1: n l n i
Row 2: o s a e
Row 3: m a i e


Show Board (y/n)?: n
1 ala
2 alan
3 alane
4 alans
5 alae
6 alas
7 als
8 also
9 anlas
10 ansa
11 ansae
12 anoa
13 anoas
14 unco
15 unci
16 uncia
17 unciae
18 uncial
19 uncials
20 unai
```

Board Printing

One of the requirements of the program is that the solver should also show the path of the solution. For the example below, we can see how “alane” is found using the grid on the right.

Word: alane							
Number of Words: 3							
'a'	u	c	o	1	0	0	0
n	'l'	'n'	i	0	2	4	0
o	s	'a'	'e'	0	0	3	5
m	a	i	e	0	0	0	0

step



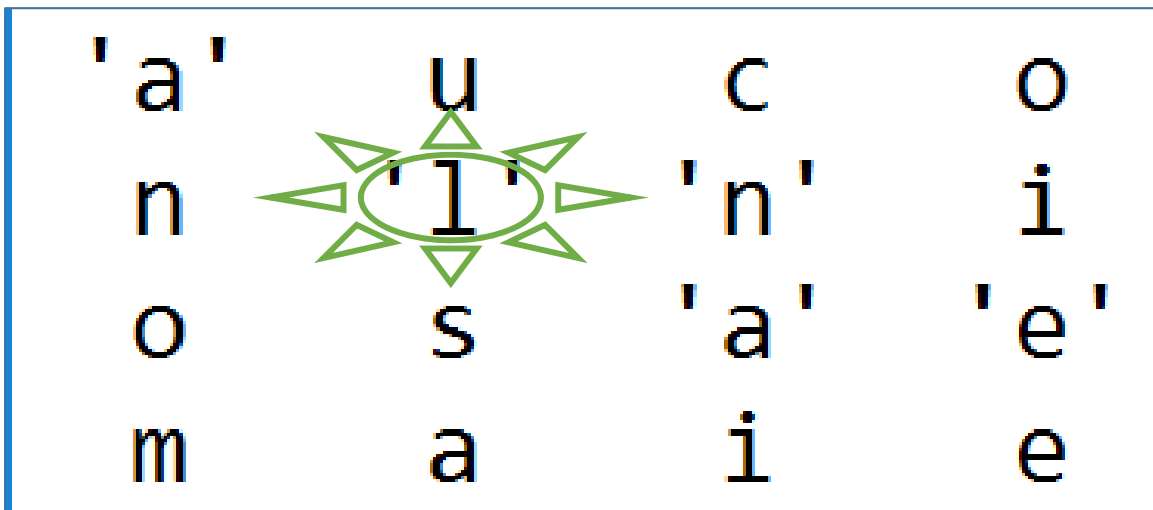
Recursion Strategy

A straightforward strategy to finding all the words on the board is to use recursive backtracking. The general idea is to start at each position and then explore all paths originating at that position being careful not to re-visit grid positions.

The solution will start with the SolveBoard function. SolveBoard will serve as a “wrapper” for another SearchForWord function that will do the heavy lifting of the recursion:

- For each position on the board
 - Call SearchForWord starting at the current board position

The job of SearchForWord is to **recursively** (hint hint...) check the surrounding 8 positions.



I suggest thinking about the base case(s) first. A solution using this strategy is very short (my SearchForWord is only about 40 lines) so if you find yourself writing lot of code, you may want to rethink your strategy. Be sure to make use of the isPrefix method to prevent exploration of paths that will not produce words.

Each call to SearchForWord function will require multiple variables in order to complete its task:

- What row and col to start
- What the current step is (see diagram for Board Printing on previous page)
- A string that stores the current progress of the path
- The board
- A dictionary to determine whether or not you found a word
- A way to remember all the previously visited spaces
- A way to remember words you have already found (Another dictionary would work well)

I strongly suggest you use a debugger or use a generous number of cout statements to help you debug this method.

Extra Credit

- 3% Create an option for the user to roll each of the 16 Boggle dice to create a random board to solve.
- 5% Create an option to play Boggle as a game (minus the egg timer).

Part 4a: Deliverables

- Dictionary (turn in again please)

Part 4b: Deliverables

- BoggleSolver and any other files used for the game
- Your test plan for BoggleSolver
- The output file from the test runs