IS 520: Final Project – Complete Boggle Solution by Trie Structure

Executive Summary

Solving the Boggle puzzle can be fun, but it will be nearly impossible to find all words from any given Boggle board. This program takes any 5x5 Boggle board and returns all valid solution words, the total number of words found, and the total number of words of length four or greater.

The valid Scrabble Words dictionary and the 10,000 most commonly searched words on Google were used to create a dictionary of valid words. The words were loaded into a data structure called a trie which allows for extremely quick computation as the board is completely solved. The trie will stop the search as a possible word becomes invalid. This means that the program does not need to search every possible combination of paths beginning with every letter on the board.

In addition to a complete solution, the program allows the user to input words which are then checked for validity – both validity of the English language and validity for the given Boggle board.

Instructions

Place the .xlsm and the two .txt files in the same directory. Open the .xlsm document and click the "Build Trie" button. After the trie has finished running (can take 0–2 minutes), the "Solve Boggle" and the "User Submit" functions can be run by clicking those buttons. In order to run the "Verify Words" function, the "Solve Boggle" must have been previously run. To reset the board, click the "Clear Words" button. The Boggle board can be reset to various combinations of letters by clicking the "Shuffle Tab" button.

Implementation Documentation.

- *Build trie*

  The trie structure applied to this problem is a graph where the nodes are a Boolean True or False and the edges are the letters as strings. From the head node with value False there are 26 edges stemming that each have a letter of the alphabet as an edge.

  At the end of a valid English word, the node will have value True. Until a valid English word is found, previous letters' nodes that together form a word will have node value False. For example, to add the word "the" to the trie we have head node with value False, joined to edge "t" connecting to node with value False, joined to edge value "h" connecting to node with value False, joined to edge value "e" connecting to node value True (because we have arrived at the end of a valid English word). However it is important to note that the search will append valid words to the list upon discovering a True valued node and continue traversing the trie until there is no longer an edge corresponding to the next letter in the word. (This allows the search to return "the' and "there" when originally searching for "there").

  This allows for quick searching through the tree to identify valid words, because if there is no edge with the letter value corresponding to the next letter to verify in the proposed word, there is nowhere else to search.

  This structure also allows for similar words to be easily connected. For example, the words the, they, them, those, their, there, etc. All begin with "th." This means that traversing the tree, all of these words can be found by traveling down the "t" edge and then the "h" edge.

From the "h" edge of the tree, many words can be accessed. Especially with words like "the", "there", and "them" all begin with "the" which allows for quick lookup.

The Collections object in VBA made this construction possible. The trie was initialized with a head node set to Boolean value False. To start the first level, there were 26 collections appended to the head node, each with value false and edge value set to 1 through 26.

The edges in the trie were integer values 1 through 26 which were converted to ASCII values by adding 95 from which it was easy to convert them into English letters.

The word lists used to build the dictionaries were the "sowpods.txt" and "google10000.txt" files (respectively, the list of valid Scrabble words and the 10,000 most commonly searched words on Google).

As the words were read in one-at-a-time into the program, they were added to the tree to form the structure mentioned above. In order to save memory, all nodes were initialized with value "Nothing." As a letter was added to increase the trie's depth, the node value was set to True or False corresponding to the validity of the word submission. As the length of words increased, so did the number of levels of the trie structure. Upon adding a word with length longer than any previous word, the last letter of the new word was included with 26 edges and nodes all with value "Nothing."

- *Complete board solution*

  The function to solve the Boggle board is recursively called to search the board starting from every letter on the board. To avoid searching the same combination of letters more than once, the program calculates a unique binary number through an OR operation between a unique number assigned to the previous and current letter used in the search. If the binary OR returns the same result as the previous step, the program exits.

  As described in the structure of the trie, the search program will exit as it finds a word is no longer valid. If searching through the trie, the program finds that the next letter in the potential word has no corresponding edge stemming from the current node in the trie, the program exits and continues to the next word to verify. When the program finds a valid word, it is appended to a globally declared word list which is used to generate outputs.

  The program can return multiple copies of the same word because on some boards there are multiple adjacent letter combinations. In order to only return the unique list, the program deletes all duplicate words and returns two sorted lists, one by alphabet and one by word length.

- *User submit*

  Upon initialization, the program will allow the user to input word selections into an input box which will immediately be verified to be valid English words. The program will also check to see if the submitted words are valid solutions for the given Boggle board.

In order to check if the user input word is a valid English word, the string must be converted into integers with values 1 through 26 so that the submission can be used to search the trie in the same way as in the complete solution mentioned above.

- *Validations / output*
  Because the trie must be built before anything else can be run, the program will prompt the user to run the trie if any other processes dependent upon the trie are requested. In order for the user submitted words to be verified as valid Boggle solutions, the complete solution must be run as well. The user will be prompted to do this too as necessary.

  In addition to sorted lists or solution words, the program outputs the count of unique words and the count of unique words with length greater than four.

  There is a ribbon modification that allows the user to shuffle the board from the Excel ribbon. The other features of the program are run by clicking the buttons on the worksheet.

- *Shuffle board*
  In order to generate multiple boards to solve, the program shuffles different valid combinations of Boggle boards (the equivalent of shaking up the Boggle pieces). This is the same assignment that the IS 520 class did earlier in the semester.

  The valid letter combinations are grouped and loaded into an array of length 25. The order was shuffled and the output letter was determined by a random number.

Discussion of learning and conceptual difficulties encountered

Upon completing this project, my knowledge of how VBA works had increased greatly. With significant programming experience in Python, understanding how to make the code work was my struggle. I knew what the algorithm was supposed to do, but not necessarily how to implement it. Building the trie was a large syntax challenge and it led me to learn more about the collections object in VBA. I learned that it was a much better fit than constructing a class module with associated functions and properties for building a trie. After seeing how the collections object worked in this application I was able to see that it could be used in several other settings as well.

After getting the algorithm to function and output correctly, one unexpected part of the task was getting the program to handle user error. I had to implement the above described checks to assure that the trie had been run before the user attempted to run the complete solver, etc. In addition to the checks and validations implemented to assure that with user input the program could still run correctly, I especially had to manage the aesthetics of the code in the solution module in the VBA developer as well as the aesthetics of the worksheet so that the user could understand how the various parts fit together intuitively.

One feature that I was not able to implement in the program was the ability to allow the user to build his or her word submissions by clicking on the Boggle board, concatenating a word as the user clicked. After extensive searches on-line I wasn't able to find the appropriate syntax, so I instead implemented an Input Box for the user to type his or her submissions. While my solution immediately verifies if the user input word is a valid English word, in order to give the user the option to play without first seeing the complete solution a separate function must be also run to verify if the words are valid submissions for the Boggle board. Verifying that the words are valid for the Boggle board also requires

that the complete Boggle solver be run first. I would have liked to have had both the clicking interaction and immediate feedback to return whether or not the submission was valid for the Boggle board.

Assistance

I am in the Applied Math program at BYU in which we do a significant amount of computer programming in Python. In this program I have learned data structures, coding implementation, and algorithm optimization and mathematical computing using the Numpy and Scipy modules.

At the start of the semester, one of the students in the program started to teach a unique algorithms class that teaches algorithm design, optimization, artificial intelligence, problem solving, and advanced data structures. In this class I learned the techniques used to develop the algorithm to solve the Boggle board and I first implemented the trie data structure in Python.

To learn the appropriate VBA syntax I leveraged several online sources including StackExchange, Mr.Excel, Microsoft Office documentation, and a few other forums.

Another significant source of help for the syntax and implementation of the trie in VBA was Dr. Gove Allen, Associate Professor of Information Systems. I spent many hours with him explaining what the data structure and the algorithm should do and he helped teach me the syntax associated with the Collections object to implement the trie in VBA and integrate it with my search algorithm for providing the complete solution for the Boggle board.