Hanyu Xiong

CS 1501

Revised Project 1 User Guide

Assignment 4

Table of Contents

Introduction

How To Play

FAQ

About Boggle

Contact

Introduction

In the word game Boggle, players try to form words from letters printed on cubes. These cubes are arranged in a 2-D array as shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| F | R | O | O |
| Y | I | E | S |
| L | D | N | T |
| A | E | R | E |

To form a word, start at any letter and traverse over adjacent cubes vertically, horizontally or diagonally. Any cube may appear no more than once in a word. In the board above, FRIEND, FROST, and ALDER are valid words. However, ENDEAR is not valid since A and R are not adjacent. Also, DEAD is not valid since D needs to be used twice in that word. Only real words from the dictionary are valid, so a word like FROOSEI would not be valid. In addition, words need to be 3+ letters long.

A variation of Boggle can be created by inserting one or more wild characters onto the board. These characters can be any letter in the alphabet. For example, if we insert a wild character into the board above:

|  |  |  |  |
| --- | --- | --- | --- |
| F | R | O | O |
| Y | I | E | S |
| L | \* | N | T |
| A | E | R | E |

The new board contains words in addition to the words found in the old board, such as: RING, RIVER, TRAIL, etc.

The goal of the game is to find as many words as possible.

How To Play

1. Start game by entering the command “java BoggleTester”
2. Game asks “Run with MyDictionary (enter a) or DLB (enter b)”
3. Enter “a” to use Mydictionary and enter “b” to use DLB
4. Game displays the boggle board
5. Game asks “Enter your guess (enter 0 to end the game):”
6. Enter your guess, or enter 0 to finish playing the game
7. After you enter your guess the game will tell you if your guess was valid, invalid, or already used.
8. Game outputs “All the words you guessed:” in an alphabetized numbered list
9. Game outputs “All the words on the board:” in an alphabetized numbered list
10. Game outputs how many words you guessed correctly, and the percent of the total words that were guessed correctly
11. Game asks “Do you want to play next game? (1 for yes, 0 for no)”
12. Enter 1 to keep playing onto the next board, 0 to stop.
13. After playing all 10 boards then the game automatically stops.
14. The first 2 boards are 5x5, the next 2 boards are 6x6, and the rest are 4x4.
15. Must restart the game to use DLB if you chose MyDictionary

FAQ

1. Can I guess the same word twice?

No, if you enter the same word you will get a message saying that you’ve already guessed that word

1. Am I allowed to use a dictionary?

No, the game is not allowed to be played with a dictionary, because that is cheating.

1. How do I win?

Technically winning would be to guess every word on the board but that is almost impossible. So every word you guess correctly is a win.

1. Can I use words I used for previous boards with the new board?

Yes, every new board is a new game, so every word you guess only counts for that board.

About Boggle

1. Algorithm used
   1. Asks user if he/she wants to use DictInterface object MyDictionary or DLB.
   2. Reads in the dictionary file.
   3. Read data file into two arrays representing the Boggle boards.
   4. Solve for first board using recursion and backtracking with the earlier chosen DictInterface object.
      1. Loop through every letter on the board.
      2. If the letter is not beyond the edge of the board and hasn’t already been used yet then append to StringBuilder, otherwise return.
      3. If the letter is an “\*” wildcard then loop through an array of all the letters in the alphabet and add that to the StringBuilder.
      4. Search if the StringBuilder is in the dictionary.
         1. If it is not in dictionary then delete the last letter added and return.
         2. If it is a prefix then do recursion with the function on the next adjacent letter.
         3. If it is a prefix and a word then add it to list of found words and move onto the next adjacent letter.
         4. If there are not more available adjacent letters then remove the last letter of StringBuilder and return. This returns from the most recent recursion step and backtracks to the letter before the most recent one. This is illustrated in the picture below: if the StringBuilder has the value DER, and we find that all of R’s adjacent letters have either been used or do not make a word or prefix, then we backtrack back to E and remove R in the StringBuilder to DE. From there we move on to E’s next adjacent letters, which is A, and we continue the cycle, adding A to StringBuilder and checking if it is in the dictionary, until we finish solving for the board.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | F | R | O | O | | Y | I | E | S | | L | D | N | T | | A | E | R | E | | |  |  |  |  | | --- | --- | --- | --- | | F | R | O | O | | Y | I | E | S | | L | D | N | T | | A | E | R | E | |

* + - 1. If it is a word then add to list of found words, remove the last letter of StringBuilder and return
  1. Output the 4x4 boggle board
  2. Asks user to enter their guess or enter 0 to end this game
  3. Asks user if they want to continue the game, if so play the next board. If both boards have been used up read in next data file. If no more boards are left, end game.
  4. Using MyDictionary
     1. Add method works by adding the words into an ArrayList if the word is not already in the ArrayList, and sorting the list.
     2. Search method works by checking if the word is in the dictionary by iterating through the ArrayList until the end of until the key is passed up. If characters match and it is at the end of the key but not the end of the string then it is a prefix. If characters match and it is at the end of the key and end of the string then it is a word. Return if it is a prefix and a word, or just a prefix, or just a word, or not a match.
  5. Using DLB
     1. Add methods work by looping through each letter in the word and making a root node if the root is null, or tracing down the trie if the child node or if any of the child node’s right sibling node is equal to the letter, or adding it as child node if there is no child node, or adding a right sibling node if the child node is not equal to the letter.
     2. Search method works by looping through each letter in the word and tracing down each child node that matches the letter. We check if any child nodes match, if not then we trace to the right sibling nodes until a match is found. Once a match if found we try again to match the next letter to a child node. This is repeated throughout the tracing process.
        1. If the whole string is matched with nodes then the string is a prefix.
        2. If the whole string is matched with nodes, and an end-of-word key is found after the last node, then the string is a word.
        3. If the whole string is matched with nodes, an end-of-word key is found after the last node, and there are more nodes following the end-of-word key, then the string is a word and a prefix.
        4. If at any point no more child nodes or right sibling nodes match a letter in the string, then the string is not a match.
        5. Return if the string is a prefix, a word, a prefix and a word, or not a match.
     3. To illustrate: ABS is a word, AD is a word and prefix, ADS is a word, BAD is a word, BAB is not a match, and ABD is not a match.

B

A

A

D

B

S

D

/

S

/

/

/

1. Algorithm efficiency
   1. MyDictionary uses a linear search algorithm. Assuming that there are N values in our dictionary to search from, in the worst case scenario we will require O(N) time. For DLB, assume we have S valid characters possible in our "alphabet" and assume our key contains K characters. In worst case we can have up to Θ(KS) character comparisons required for a search, since we can have up to S comparisons to find the character on each level and we have K levels to get to the end of the key. But most of the levels will have very few characters, so practically speaking a dlB search will require Θ(K) time. DLB is a very efficient data structure because it gives us a search time that is independent of N.
   2. Implementing DictInterface using MyDictionary as the object is extremely inefficient, especially when compared to using DLB as the object. The difference in efficiency can clearly be seen when we used data3.txt and a timer in the program to test the difference in the run-times between MyDictionary and DLB. The time tested for is the time to find all of the words within a given board. MyDictionary had the three run-times of 117.47 s, 112.85s, and 109.47s, while DLB had 2.27s, 2.51s, and 2.35s. MyDictionary averaged at 113.26s and DLB averaged at 2.38s.

Contact

The author is Hanyu Xiong and can be reached at her email hax12@pitt.edu.