BOGGLE

Spec FAQ Project Submit

If a valid word can be formed in more than one way, how many times should I return it? Only once. For example, you should return the word `EQUATION` only once in board-q.txt even though there are two ways to form it.

Do In need to return the words in alphabetical order? No, that is not required.

Which data structure(s) should I use to store the dictionary? It is up to you to decide. A trie is a natural starting point.

My program isn't fast enough to get 100%. What can I do to improve performance? See the *Possible Optimizations* section below for some ideas. It may be a challenge.

The zip file boggle.zip contains some sample dictionaries and boards for testing.

Dictionaries. Below are some dictionaries for testing. Each dictionary consists of a sequence of words containing only the uppercase letters `A` through `Z`, separated by whitespace, and in alphabetical order.

dictionary-nursery.txt	1,647	words that appear in several popular nursery rhymes
dictionary-algs4.txt	6,013	words that appear in Algorithms 4/e
dictionary-common.txt	20,068	a list of common English words
dictionary-shakespeare.txt	23,688	words that appear in the complete works of William Shakespeare
dictionary-enable2k.txt	173,528	Enhanced North American Benchmark Lexicon
dictionary-twl06.txt	178,691	Tournament Word List
dictionary-yawl.txt	264,061	Yet Another Word List
dictionary-sowpods.txt	267,751	the SOWPODS list of words
dictionary-zingarelli2005.txt	584,983	Italian Scrabble Dictionary

Boards. We provide a number of boards for testing. The boards named `boards-points[xxxx].txt` are Boggle board that results in a maximum score of `xxxx` points using the dictionary dictionary-yawl.txt. The other boards are designed to test various corner cases, including dealing with the two-letter sequence `Qu` and boards of dimensions other than 4-by-4.

These are purely suggestions for how you might make progress. You do not have to follow these steps.

- Familiarize yourself with the BoggleBoard.java data type.
- Use a standard set data type to represent the dictionary, e.g., a `SET<String>`, a `TreeSet<String>`, or a `HashSet<String>`.
- Create the data type `BoggleSolver`. Write a method based on depth-first search to enumerate all strings that can be composed by following sequences of adjacent dice. That is, enumerate all simple paths in the Boggle graph (but there is no need to explicitly form the graph). For now, ignore the special two-letter sequence `Qu`.
- Now, implement the following critical backtracking optimization: *when the current path corresponds to a string that is not a prefix of any word in the dictionary, there is no need to expand the path further*. To do this, you will need to create a data structure for the dictionary that supports the *prefix query* operation: given a prefix, is there any word in the dictionary that starts with that prefix?
- Deal with the special two-letter sequence `Qu`.

You will likely need to optimize some aspects of your program to pass all of the performance points (which are, intentionally, more challenging on this assignment). Here are a few ideas:

- Make sure that you have implemented the critical backtracking optimization described above. This is, by far, the most important step—several orders of magnitude!
- Think about whether you can implement the dictionary in a more efficient manner. Recall that the alphabet consists of only the 26 letters `A` through `Z`.
- Exploit that fact that when you perform a *prefix query* operation, it is usually almost identical to the previous *prefix query*, except that it is one letter longer.
- Consider a nonrecursive implementation of the *prefix query* operation.
- Precompute the Boggle graph, i.e., the set of cubes adjacent to each cube. But don't necessarily use a heavyweight `Graph` object.