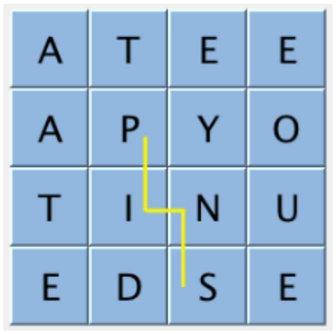
CS360 – Spring 2015
Project 5: Boggle

Using either Java (6/7) or C++ 1998 specification, write a program that solves the following problem.

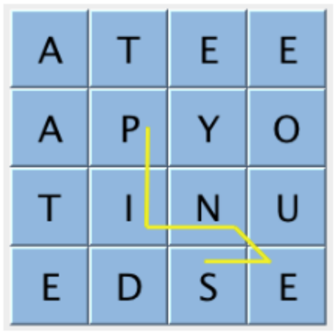
The Boggle game. Boggle is a word game designed by Allan Turoff and distributed by Hasbro. It involves a board made up of 16 cubic dice, where each die has a letter printed on each of its sides. At the beginning of the game, the 16 dice are shaken and randomly distributed into a 4-by-4 tray, with only the top sides of the dice visible. The players compete to accumulate points by building *valid* words out of the dice according to the following rules:

- A valid word must be composed by following a sequence of *adjacent dice*—two dice are adjacent if they are horizontal, vertical, or diagonal neighbors.
- A valid word can use each die at most once.
- A valid word must contain at least 3 letters.
- A valid word must be in the dictionary (which typically does not contain proper nouns).

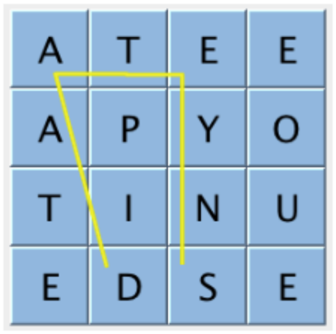
Here are some examples of valid and invalid words:



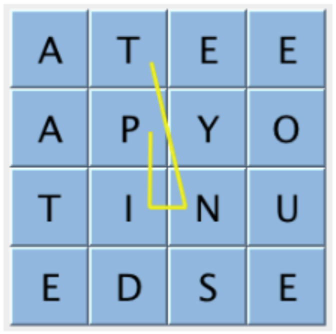
PINS
(*valid*)



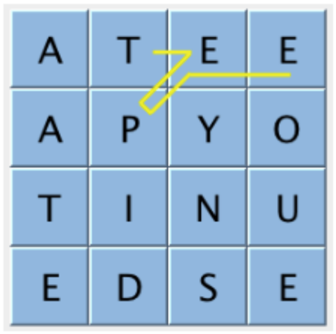
PINES
(*valid*)



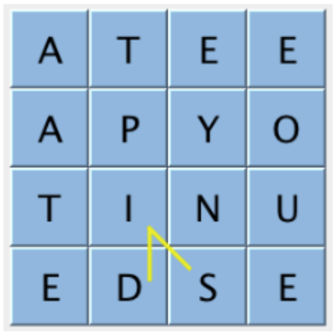
DATES
(*invalid—dice not adjacent*)



PINT
(*invalid—path not sequential*)



TEPEE
(*invalid—die used more than once*)



SID
(*invalid—word not in dictionary*)

Scoring. Words are scored according to their length, using this table:

word length	*points*
0–2	0
3–4	1
5	2
6	3
7	5
8+	11

The *Qu* special case. In the English language, the letter `Q` is almost always followed by the letter `U`. Consequently, the side of one die is printed with the two-letter sequence `Qu` instead of `Q` (and this two-letter sequence must be used together when forming words). When scoring, `Qu` counts as two letters; for example, the word `QuEUE` scores as a 5-letter word even though it is formed by following a sequence of 4 dice.

Input:
There are two input files: 1) board file and 2) dictionary file

Board file
Filename: *board.txt*
Format:
Line 1: #_of_rows #_of_cols //board dimensions , space delimited
Remaining lines: characters on board, space delimited
Example:

```
4 4
T  A  S  L
R  S  N  G
L  A  I  D
G  U  H  O
```

Dictionary file
Filename: *dict.txt*
Format: valid words, one per line, all in upper case, not necessarily sorted
Example:

```
AIR
HIT
EGG
```

Output
Output valid words to standard out. They can be in any order. They should be valid based on the dictionary file.

```
ANT
END
TEA
```

Other parameters:

- The program should be designed to put out as many words as quickly as possible. It is possible that your program will be given a limited amount of time to run and produce a list of valid words.
- You can use Collections or STL classes. Make sure that you choose the appropriate data structures and algorithms by reading the documentation.
- You must utilize breadth or depth first search to identify valid words.

Failure to follow directions will result in point deductions or a zero. There are 67 students in this class. It is unreasonable to expect that any exceptions to the procedure will be made.

Late assignments will not be accepted unless you have a doctor's note covering the entire period from Mar 24th through Apr 2nd. Java programs should not require any machine specific code. The C++ programs will be tested on bama.ua.edu. You can gain access to this system for testing. Develop C++ on other platforms at your own risk. Programs must only include constructs made available by the language specifications (Java 6 and C++11/Posix- https://en.wikipedia.org/wiki/C%2B%2B98) and no other extensions. You also must comply with any prohibitions on data structures in the program description.

Blackboard submission by Apr 2 @11:59pm:

- Program code in ASCII file with .java or .cpp extension. Pdfs of code will not be graded. Class files or executables will not be graded
- Test results in pdf format. You should generate test data such that it shows that your program works.
- If your solution utilizes more than one source file (and there is good reason why it would), provide a Unix batch (bash or csh), make or ant file to compile and run the test cases. File should run without any machine specific requirements other than make, bash/csh and ant.
- Any relevant notes in text file README
- The program must compile and run as submitted. In an effort to speed grading, I will not have time to seek input on programs that do not work.

In class printout submission Apr 6 @ 9am:

- Program printout that is easy to read (line wrapping etc).
- Short version of test results (save a tree...)
- The source code (.java or .cpp files) should also be turned in via Blackboard (not emailed to me or the TA). I do not use a Windows computer. Test result submissions of any type other than pdf will not be graded.
- ***No packages or directories used (default package running current directory)***
- The source code and test results should be printed and brought to class Apr 6th at 9am to get credit. Any assignments that are not submitted via printout on Apr 6th will receive a 50% deduction.
- I will do some initial automated testing based on checking the produced output. Adhere to the output format closely.

Cheating Policy: This is an individual assignment. The program must represent your own work. You can discuss high-level concepts. Do not show your code to anyone. I reserve the right to ask you about your program to discern if you indeed wrote the code. I also run software that identifies solutions that contain similar code. If you cannot explain your code and choices verbally, you may be turned in for academic misconduct. All submissions will be analyzed to identify possible cases of cheating. Any cases of suspected collaboration will be referred to the College of Engineering Dean. A zero or low grade is always better than having an academic misconduct on your academic record.

******** You can receive a zero or a greatly reduced grade based on not following these requirements. This is the 5th assignment. Students should understand the assignment coding and submission criteria.**

- **Do not include links to any online documentation. You are to write this completely on your own. If your code contains any code that can be found online, you will receive a zero.**
- **Your printout must match the code that is submitted on blackboard. If they do not match, you will receive a zero.**
- **No packages or directories used (default package running current directory)**
- **Filenames are correct and open files in the current directory**
- **Program does not compile on bama.ua.edu**
- **Programs that use approaches that are asymptotically slower than the optimal, contain extra loops or any code that hurts efficiency will receive a zero regardless of functionality. This assignment should showcase all that you have learned**

Here is an overview of evaluation criteria. It is subject to change within 10% at my discretion. The top submissions that contain particularly elegant and clear code may earn extra credit.

50% - Functionality
• Program contains the correct code to produce a valid word list
• Program meets the requirements (correct selection of built in or programmed data structures and algorithms, etc)
• Uses correct file name
• No artificial limits on word list or board size
• Output from running the program is included that shows that the program works
• Output in correct format
• Appropriate sort used

40% - Design
• Program uses appropriate language constructs
• Overall design should be reasonably consistent with OO design principles
• Program displays a reasonably fast method for producing the output
• Program exhibits defensible design choices in algorithms and data structures (if you add any)

10% - Style (Checked vigorously)
• Program must use appropriate and consistent style for naming of elements (C style or c++ style)
• Program must include reasonable whitespace and appropriate indentation
• Program must include comments, especially in areas where you need to support your choices or where the purpose of the code is unclear.

** Clarifications on the assignment will be posted to blackboard. You will be responsible for consulting blackboard frequently and remaining aware of clarifications.