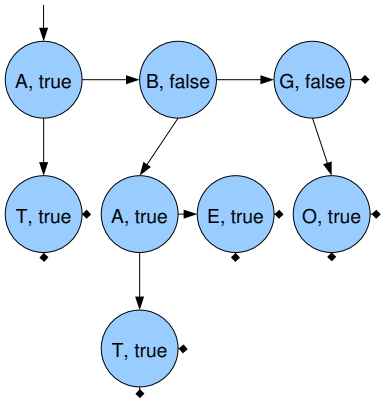
Programming Assignment D
Due on Saturday, March 16, 2019 at 11:59 pm

The goal of this assignment is to implement a Boggle™ player (see `http://www.boardgamegeek.com/boardgame/1293` for more information, if you have never played Boggle before) in Prolog. The basic idea is that a set of letters are placed in a 4×4 grid. You (or in this case, your program) must find as many words as possible in the grid. Words can be formed by starting at any letter in the grid and then moving to any other of the 8 adjacent letters and continuing in such a fashion. However, you may not repeat a letter. Therefore in the grid below, you can find "PUN" and "NODE" but neither "HEN" (E not adjacent to N) nor "DUDE" (repeats the D).

B	O	O	E
R	Y	E	H
P	U	D	L
I	O	N	C

For this assignment, I have implemented a dictionary data structure and the necessary code to load in a dictionary from disk. A dictionary is implemented as a tree. At each node in the tree there is a letter, a boolean, and two subtrees. The letter is the (potential) next letter in the word, and the boolean is whether the word ending with this letter is a word. The first subtree is the subtree of all suffixes to this stem. The second subtree is the subtree of alternative letters at this position. So, the first subtree is "down" and the second subtree is "sideways" in the representation below.



This tree holds the words "a," "at," "ba," "bat," "be," and "go." Note that it does not hold "g" because that node has false.

In terms of prolog, an empty tree is the term `null` and a node in the tree is represented by the term `node(C,B,D,L)` where `C` and `B` are the character and boolean of a node respectively, and `D` and `L` are the "down" and "left" subtrees (as in the diagram above). Usually you would keep the left-pointing linked list in sorted order. For this assignment, it will actually be simpler to assume that is not necessarily the case.

The supplied code includes predicates `loaddict` and `boggleboard` that load a dictionary and create a boogle board. There is also a predicate to draw a boggle board. You do not need to understand how these work, although it might be helpful to understand how the `indict` predicate works.

Supplied with the sample code is also a small set of words in the file `bogwords`. You should load your dictionary from this file. Larger dictionaries are certainly possible, but you'll find this one a good size for testing.

Question 1. [40 points]
Implement a predicate `isboggleword(+board,+dictionary,?word)` that will check whether the word given is a valid word on the board. If the word is not given (that is, it is unbound), it will generate all valid words on the board. For the moment, do not worry about repeating squares on the board.[1]

The supplied file predicate `boggleword` loads the dictionary and checks the word (using the predicate `isboggleword` that you wrote). As an example, here's how it could be used:

```
?- boggleboard(B), drawboard(B), boggleword(B,X).
SREU
CGHF
SNLT
ROLS

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['E', 'R', 'E'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['E', 'R', 'G'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['E', 'F', 'T'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['H', 'E', 'R'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['H', 'E', 'R', 'E'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['H', 'E', 'F', 'T'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['H', 'U', 'E'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['H', 'U', 'H'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['L', 'O', 'L', 'L'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['L', 'O', 'N', 'G'] ;

B = ['S', 'R', 'E', 'U', 'C', 'G', 'H', 'F', 'S'|...],
X = ['L', 'O', 'L', 'L']

Yes
```

At this point, it would have continued, but I pressed return instead of semicolon. Note that it generated "ere" even though that repeats the "e" and it generated "loll" two different ways (both incorrect, as it turns out). For this part of the assignment, that is fine.

[1] Also, treat a "Q" just as a "Q." In the real version of Boggle, the "Q" square is considered as having an automatic "U" after it. You should *not* make this special exception for this assignment. It just makes things harder.

Note, my solution is very short: 15 lines for all predicates (and that includes a long—8 lines—but simple helper predicate). You may have a longer solution, but if you are writing a lot of code, you are probably doing this incorrectly. Your recursion should walk down the dictionary tree at the same time that it walks around the board. You should not need the `indict` function.

Question 2. [20 points]
Now modify `isboggleword` so that it does not repeat grid cells within a single word. A word *can* repeat a letter, if each occurance of the same letter comes from a different cell. Furthermore, your solution can repeat entire words (if there are multiple paths that form the same word).

Question 3. [10 points]
Finally, you should write `removedup(+inlist,-outlist)` which succeeds if the second argument has all of the elements of the first, but without any duplicates. Then, the supplied `allbogglewords(+board,-words)` should work to print out all valid boggle words. As an example:

```
?- boggleboard(B), drawboard(B), allbogglewords(B,X), writeln(X).
MEQP
JOTP
OLRW
EZEV
[JET, JOLT, JOT, LEW, LOOM, LOOT, LORE, LOT, MET, METRO, MOLE, MOLT, MOO,
MORE, MOREL, MORT, MOT, OOZE, ORE, PRO, PROM, REV, ROE, ROLE, ROT, ROTE,
TOE, TOME, TOO, TOOL, TOR, TORE, WELT, WERT, WROTE, ZERO, ZOO, ZOOM]

B = ['M', 'E', 'Q', 'P', 'J', 'O', 'T', 'P', 'O'|...],
X = ['JET', 'JOLT', 'JOT', 'LEW', 'LOOM', 'LOOT', 'LORE', 'LOT', 'MET'|...]
```

Mine ends up with a sorted list. Yours may or may not. This is probably the easiest part of the assignment, so if you wish to do it first, you may.

Submission Instructions:
Place all of your code in a single `.pro` file **that also includes the supplied code.** Please comment your code and do not submit any other files (any additional information should be placed in comments in the .pro file). Submit through gradescope.