CSE 100: PA4, B TREES (CONTD)
PA4 (Checkpoint due tonight at 11pm):
Where are you with PA4?

How far along are you with the checkpoint?
A. Haven’t started
B. I have implemented setBoard(), but I don’t know how to go about isOnBoard()
C. I have worked out isOnBoard(), plan to finish by tonight
D. Done
PA4: BogglePlayer

How can you represent the board?
A. `vector<Node> board`
B. `vector<vector<Node>> board`
C. `Node ** board`
D. `Node *** board`
E. Any of the above and other possibilities

The board is represented as a 2D array.

 Populate board at run time
PA4: BogglePlayer

How can you represent the board?
A. vector<Node> board
B. vector<vector<Node>> board
C. Node ** board
D. Node *** board
E. Any of the above and other possibilities

How do you design the class Node?
Depends on what you want to do.

std::vector<int> BogglePlayer::isOnBoard(const std::string &word_to_check)
What do we get from running (vanilla) DFS on this graph, starting with source node C₄?

A. All possible words (valid and invalid) starting with C
B. A sequence containing all the letters on the board starting with C
C. A sequence containing some of the letters on the board starting with C
D. Either B or C
What is the simplest change you can make to DFS to get closer to the solution?

0. Visited is the selected to be on the path that we want to construct.

1. Set vertex as visited only if it contains the node expected letter.
isOnBoard(G,v, word) (v is the vertex where the search starts, word is the word we are looking for)

Stack S := {}; (start with an empty stack), pos:=0
for each vertex u, set u.visited := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if (NOT u.visited AND u.letter == word[pos])
    pos:=pos+1
    u.visited := true;
    for each unvisited neighbour w of u
      push S, w;
  end if
end while
END DFS()

With the modified algorithm, can I find the word CAP (if the source vertex is C\textsubscript{5})
A. Yes
B. No
C. Not exactly, but I have made progress

Why?

- The graph is a portion of a chessboard with letters and numbers representing coordinates. The search is performed on this graph.
- The algorithm in the image is a depth-first search (DFS) algorithm for finding a path on a graph.
- The question asks whether, starting from vertex C\textsubscript{5}, it is possible to find the word CAP.

**Reasoning**

- The algorithm checks each vertex for a match with the word, incrementing the position in the word if a match is found.
- The search continues as long as there are unvisited neighbors, and it marks visited vertices to avoid cycles.

**Analysis**

- The graph shows that the word CAP cannot be found starting from C\textsubscript{5} because:
  - The word CAP is not on the board at C\textsubscript{5}.
  - The neighbors of C\textsubscript{5} (A\textsubscript{4}, D\textsubscript{5}, E\textsubscript{5}, F\textsubscript{5}) do not contain the letters A, C, or P.
  - The search algorithm stops when it finds a match, and since CAP is not found, the algorithm stops even before reaching a match.

**Conclusion**

- The correct answer is C. Not exactly, but I have made progress because the algo continues even after a word has been found.
isOnBoard(G, v, word) (v is the vertex where the search starts, word is the word we are looking for)

Stack S := {}; (start with an empty stack), pos := 0
for each vertex u, set u.visited := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if (NOT u.visited AND u.letter == word[pos])
    u.visited := true;
    pos := pos + 1
    if pos == length(word)
      return
  end if
for each unvisited neighbour w of u
  push S, w;
end while
END DFS()

Am I done? What happens if I search for “CANIBAL”?

A. We can always find it without any problem
B. We may or may not find it
We need a way to remember a vertex if the search through its neighbors is unsuccessful.

To remember a vertex, push it into the stack before visiting its neighbors.

A vertex will be popped out of the stack only if search via its neighbors is unsuccessful!
isOnBoard(G, v, word) (v is the vertex where the search starts, word is the word we are looking for)

Stack S := {}; (start with an empty stack), pos:=0

for each vertex u, set u.visited := false;
push S, v;

while (S is not empty) do
    u := pop S;
    if u.revisit == true
        u.revisit := false
        u.visited := false;
        pos:=pos-1
    else if (NOT u.visited AND u.letter ==word[pos])
        u.visited := true;
        pos:=pos+1
        if pos==length(word)
            return
        u.revisit := true;
        push S, u
    for each unvisited neighbour w of u
        push S, w;
end if
end while
END DFS()

Need Backtracking!