CSE 100: PA4, BOGGLE
Announcements

• PA4 writeup and skeleton are now available
• Please read the writeup thoroughly
• Challenge: SETUP and DESIGN is harder than previous Pas
• We have given you a basic design to get you started (and finished with the checkpoint) quickly
• Checkpoint due next week, Wed May 20\textsuperscript{th} at 11:00PM
  • Start today
The game of Boggle:
• Two players, one board (MxN) dies
• Each die face contains a string
• Player that constructs the maximum number of unique words is the winner

Rules of the game:
• Words are constructed by traversing a sequence of adjacent dies (an acyclic path starting at any dice)
• Two dice are adjacent if they are next to each other horizontally, vertically, or diagonally.
• Die can only be used once in a word (acyclic path)

Checkpoint:
Implement
• setBoard(): Given a 2D array of strings populate the board
• isOnBoard(): Given a word, check if it is on the board
Graphs in PA4 (HINT!!)

This is the boggle board… how would you represent it in your code?
A. As a linked list
B. As a 2D array
C. As a tree
D. Graph
E. A vector
Graphs in PA4 (HINT!!)

This is also a graph… where are the edges? (fill them in)
Graphs in PA4 (HINT!!)

/**
 * Vector representing the boggle board.
 */
std::vector<BoardPos> board;
//Elements are in row major order

If row is the number of rows and col is the number of columns, how would you access the element in the ith row and jth column of the 2D array using the vector representation
A. board[i+j]
B. board[i*col+j]
C. board[j*col+i]
D. board[i*row+j]
E. board[j*row+i]
Why do we have the visited field?
PA4: BogglePlayer

Which of the following algorithms best applies to our problem i.e searching for a word on the board?
A. BFS  
B. DFS  
C. Dijkstra’s  
D. Prims

std::vector<int> BogglePlayer::isOnBoard(const std::string &word_to_check)
What would we be finding by running (vanilla) DFS on this graph, starting with source node $C_5$ (assume the index of each visited node is stored in the order in which nodes are visited)?

A. All sequences of words on the board starting with C  
B. A sequence that consists of all letters on the board starting with C  
C. A sequence that consists of some of the letters on the board starting with C  
D. None of the above
What is the simplest change you can make to DFS to get us one step closer to our solution?

DFS(G,v) ( v is the vertex where the search starts )
Stack S := {}; ( start with an empty stack )
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
    u := pop S;
    if (NOT u.visited) then
        set u.visited:=true;
        for each unvisited neighbour w of u
            push S, w;
    end if
end while
END DFS()
What is the simplest change you can make to DFS to get us one step closer to our solution?

- Pass in the word that we are searching for
- In our traversal only go down paths that match with the next letter in the word that we are searching for

```plaintext
DFS(G,v) ( v is the vertex where the search starts )
Stack S := {}; ( start with an empty stack )
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
    u := pop S;
    if (NOT u.visited) then
        set u.visited:=true;
        for each unvisited neighbour w of u
            push S, w;
    end if
end while
END DFS()
```
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 visited[u] := 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()

What is the sequence of visited vertices if the word that I am trying to find is CAP and the source vertex is C₅
A. C₅ A₁ P₄
B. C₅ P₄ A₁
C. Cannot say
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 visited[u] := 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()

What is the sequence of visited vertices if the word that I am trying to find is CAP and the source vertex is C_5
A. C_5 A_1 P_4
B. C_5 P_4 A_1
C. Cannot say

Am I doing unnecessary extra work at any point?
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 visited[u] := 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
    for each unvisited neighbour w of u
        push S, w;
end if
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
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 visited[u] := 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
        for each unvisited neighbour w of u
            push S, w;
    end if
end while
END DFS()

If none of the neighbors of A₁ match, need to set A₁ as unvisited- how?
Need Backtracking!

```
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 visited[u] := 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()
```