Stacks and DFS revisited

Topics for this week:
- stacks (and queues)
- non-recursive depth-first search implementation
- (breadth-first search)
- working with “configuration graphs”
Problem: BOGGLE

Boggle game:
Given 4x4 grid with letters, find English words by starting at any location and move through unused neighboring cells

Some words from our example:
  tea, hits, make, catfish, etc.

Score by word lengths:

<table>
<thead>
<tr>
<th>length</th>
<th>3 or 4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>&gt;=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>11</td>
</tr>
</tbody>
</table>
Problem: BOGGLE, configuration graph

How to solve?

"Configuration graph":
- enumerate all words starting with 'a' in the upper-left corner

See the notes for a completely worked out config tree for an upper-left start in a $2 \times 2$ board

"aca": positions

$[(0,0), (0,1), (1,2)]$
Non-recursive DFS

Recall depth-first search from last quarter’s week 9:

How to do non-recursively?

Seen so far: 0, 1, 2, 5, 6, 4, 3, 7, 8

[visited from week 9]

idea:
1) start w. empty stack
2) we’ll come back, let’s check the next slide

neighbors of 0
Stack

- Linked list based data structure that supports
  - adding an element at the top of the list, and
  - removing an element from the top of the list

Implementation:

```python
def push(startNode, value):
    newNode = Node(value, startNode)
    return newNode

def pop(startNode):
    if startNode is None:
        return startNode.next, startNode.data
    return None
```

Running time complexity:

- $O(1)$ for both `push()` and `pop()`
Non-recursive DFS, part II

Pseudo code:

```python
def DFSnonrec ( graph, source ) :
    initialize the stack to empty stack
    create a dealWith list - initially empty
    push source onto the stack
    while stack is nonempty:
        let ts be the top of the stack (pop)
        go through all neighbors of ts:
        if a neighbor is not in the dealWith list:
            push the neighbor onto the stack
            add ts to the dealWith list
```

Running time complexity:

\[ O(n+m) \] or slightly weaker analysis: \( O(n^2) \)
Pseudo code:

def DFSboggle (board):
    for every row:
        for every column:
            run DFSboggleSource([row, column])

def DFSboggleSource(board, source):
    initialize an empty stack
    (create empty dealWith list) ← don’t need in boggle
    push source onto the stack
    while stack is nonempty:
        let ts be the top of the stack
        let last be the last position in ts
        go through the neighboring positions of last (← last →):
            if neighbors is not in ts:
                create a list neighborlist = ts with neighbors appended
                (if neighborlist is not in the dealWith list ;)
                push neighborlist onto the stack
                if the corresponding string is a valid word, then output it and add to a score
BOGGLE implementation details

Initialize:
- read dice from a file
- read legal English words from a file
- create board by randomly tossing the dice

Run depth-first traversal through the configurations, starting with each possible starting letter.