Stacks and DFS revisited

Topics for this week:

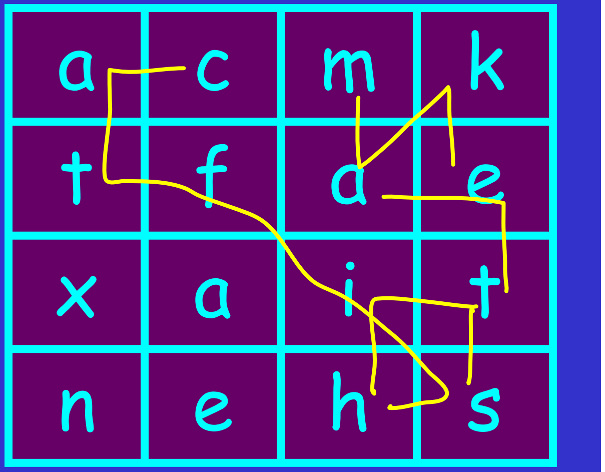
- stacks (and queues)
- non-recursive depth-first search implementation
- (breadth-first search)
- working with “configuration graphs”

Problem: BOGGLE

[Week 2]

Boggle game:

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



a	c	m	k
t	f	a	e
x	a	i	t
n	e	h	s

Some words from our example:

tea, hits, make, catfish, etc.

for now, assume:

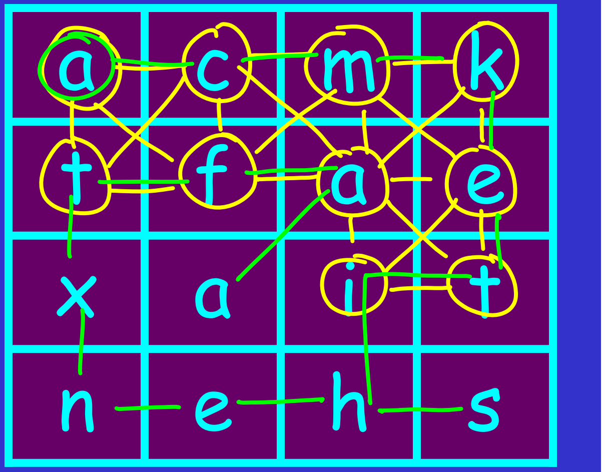
given the letters in the board and a file with valid words, list all possible valid words that can be formed by the given letters

Score by word lengths:

length	3 or 4	5	6	7	>=8
points	1	2	3	5	11

Problem: BOGGLE, configuration graph

[Week 2]

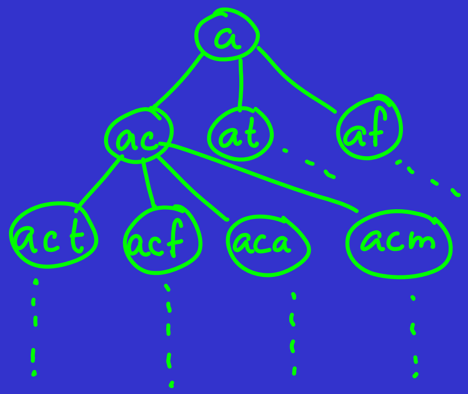


a	c	m	k
t	f	a	e
x	a	i	t
n	e	h	s

How to solve ?

"Configuration graph" :

-enumerate all words starting with 'a' in the upper-left corner



a config. tree

→ we want to list all words that are in the dictionary

e.g. 'act'

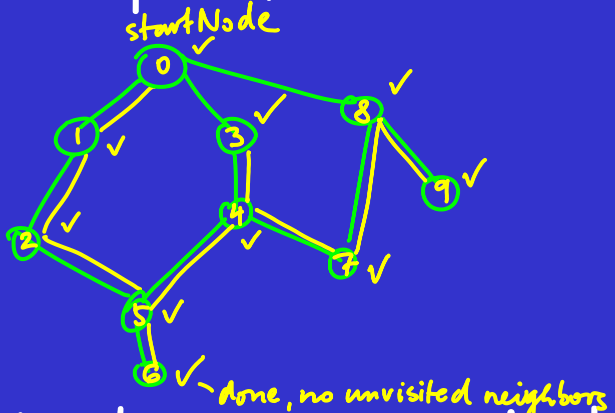
see the notes for a completely worked out config. tree for an upper left start in a 2x2 board

"aca": positions

[(0,0), (0,1), (1,2)]

Non-recursive DFS

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



DFS (graph, startNode)

idea:

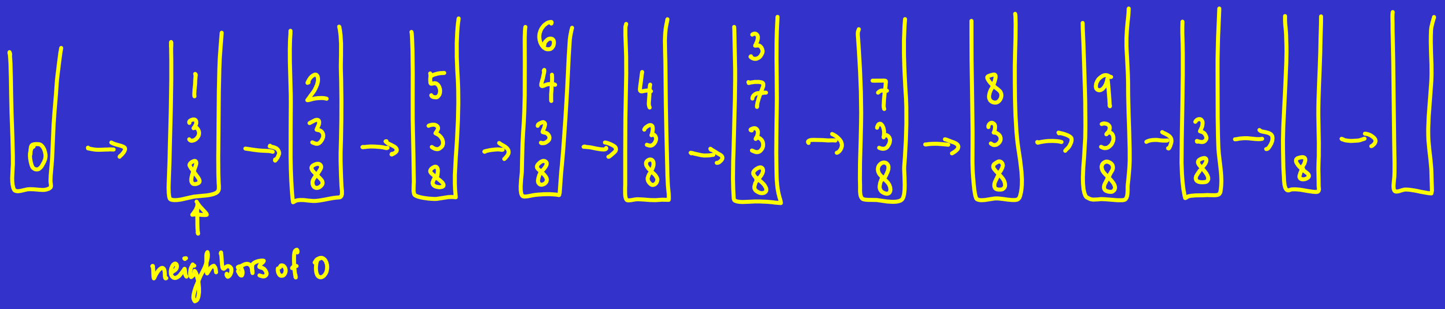
1) start w. empty stack

2)

we'll come back, let's check the next slide

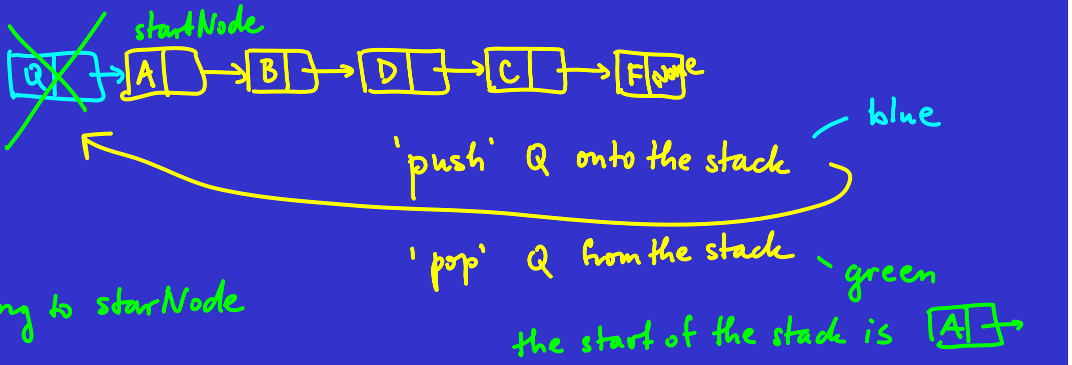
How to do non-recursively ?

seen so far: 0, 1, 2, 5, 6, 4, 3, 7, 8
[visited from week 9]



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



'push' Q onto the stack

'pop' Q from the stack

the start of the stack is

Implementation:

```
def push ( startNode, value) :
    create a node with data = value, pointing to starNode
    return the new node

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

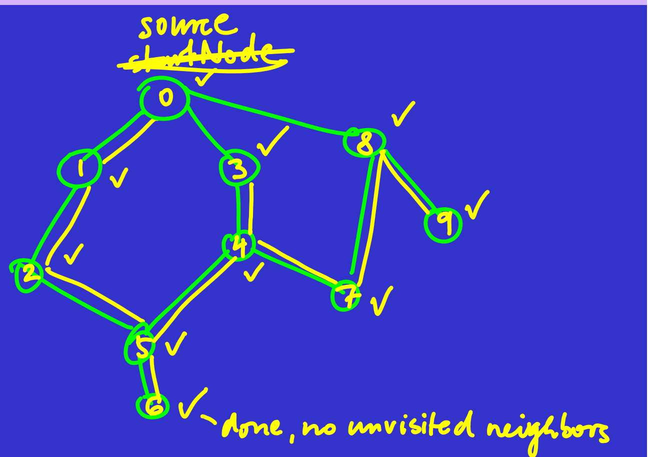
Running time complexity:

$O(1)$ for both push() and pop()

Non-recursive DFS, part II

Pseudo code:

```
def DFSnonrec ( graph, source ) :
    initialize the stack to empty stack
    create a dealtWith 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 dealtWith list :
                push the neighbor onto the stack
        add ts to the dealtWith list
```



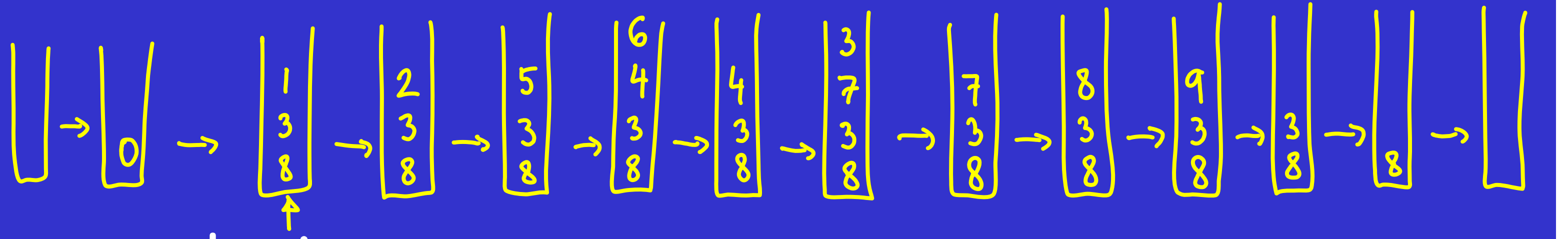
source

~~startNode~~

done, no unvisited neighbors

dealtWith list

seen so far: 0, 1, 2, 5, 6, 4, 3, 7, 8
[visited from week 9]



Running time complexity:

$O(n+m)$ or slightly weaker analysis: $O(n^2)$

Back to BOGGLE

[Week 2]

Pseudo code:

```
DFS boggle ( board ) :
    for every row :
        for every column :
            run DFSboggleSource ( board, [[(row, column)]] )
```

```
DFSboggleSource ( board, source ) :
    initialize an empty stack
    (create empty dealtWith list)   ← don't need in boggle
                                      bec. config. graph is
                                      a tree
    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 neighPos is not in ts :
                create a list neighborlist = ts with neighPos appended
                (if neighborlist is not in the dealtWith list :)
                    push neighborlist onto the stack
                    if the corresponding string is a valid word, then output it and add to a score
    (add ts to the dealtWith list)
```

```
def DFSnonrec ( graph, source ) :
    initialize the stack to empty stack
    create a dealtWith 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 dealtWith list
                push the neighbor onto the stack
        add ts to the dealtWith list
```

a node in the config. graph:

a list of positions in the board that form the string at the node

e.g. "aca" is $\underbrace{[(0,0),(0,1),(1,2)]}_{\text{eg. ts}}$ ← last

BOGGLE implementation details

[Week 2]

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.