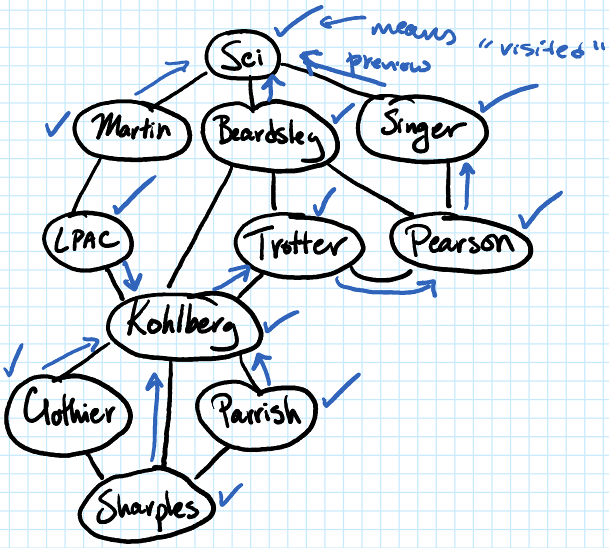
6.2 stacks, queues, and mazes

Tuesday, October 4, 2022 11:26

SEARCH ALGORITHM

```
- add start position to the data structure
- mark start as visited
- while data structure is not empty:
    ○ remove a position from data structure, call it current
    ○ if current is the goal:
        ▪ search is complete! break
    ○ otherwise, for each neighbor of current:
        ▪ if neighbor isn't visited
            □ mark neighbor as visited
            □ neighbor's "previous" recorded as current
            □ insert neighbor into data structure
```



Walkthrough: data structure is stack

current: ~~Sci~~
~~Singer~~
~~Pearson~~
~~Trotter~~
~~Kohlberg~~
~~LPAC~~
Sharples

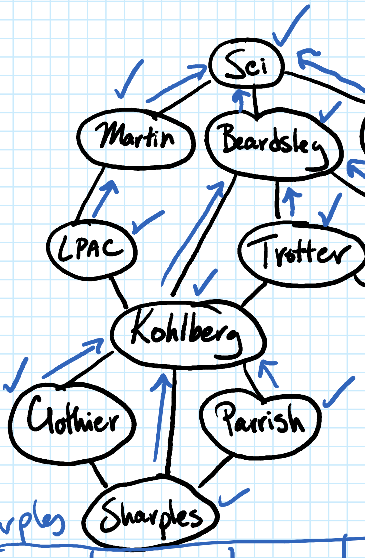
~~LPAC~~
~~Sharples~~
~~Clothier~~
~~Parrish~~
~~Kohlberg~~
~~Trotter~~
~~Pearson~~
~~Singer~~
Beardsley
Martin
~~Sci~~

final path:

Sci, Singer, Pearson, Trotter, Kohlberg, Sharples.

SEARCH ALGORITHM

```
- add start position to the data structure
- mark start as visited
- while data structure is not empty:
    ○ remove a position from data structure, call it current
    ○ if current is the goal:
        ▪ search is complete! break
    ○ otherwise, for each neighbor of current:
        ▪ if neighbor isn't visited
            □ mark neighbor as visited
            □ neighbor's "previous" recorded as current
            □ insert neighbor into data structure
```



walkthrough using queue

current: Sci ~~Martin~~ ~~Beardsley~~
~~Singer~~ ~~LPAC~~ ~~Kohlberg~~
~~trotter~~ ~~Pearson~~ ~~Clothier~~ Sharples

front | | | | | | | | | | back

~~Sci~~	~~Martin~~	~~Beardsley~~	~~Singer~~	~~LPAC~~	~~Kohlberg~~	~~Trotter~~	~~Pearson~~	~~Clothier~~	~~Sharples~~	Parrish

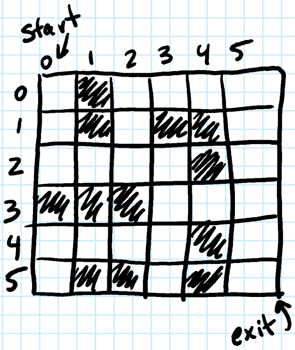
path: Sci Beardsley Kohlberg Sharples

Differences in how the search works

- when using a queue: breadth-first search, finds shortest (fewest hops) path
- when using a stack: depth-first search
  can be better for certain applications

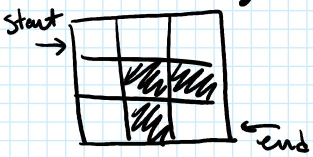
Searching a maze

- possible moves are N, S, E, W (no diagonals)
- walls block movement
- start is always (0,0)
- exit is always bottom right (height -1, width -1)



We are searching for a path:
(0,0) . . . (5,5)

Impossible maze:



What would the search algorithm find here?

REPRESENTING A MAZE

Position class

data	methods
int x	int getX()
int y	int getY()
bool wall	void setWall(), bool isWall()
bool visited	void setVisited(), bool isVisited()
Position* previous	void setPrevious(Position* p)
	Position* getPrevious()

MAZE CLASS

data

int width
int height
Position*** positions

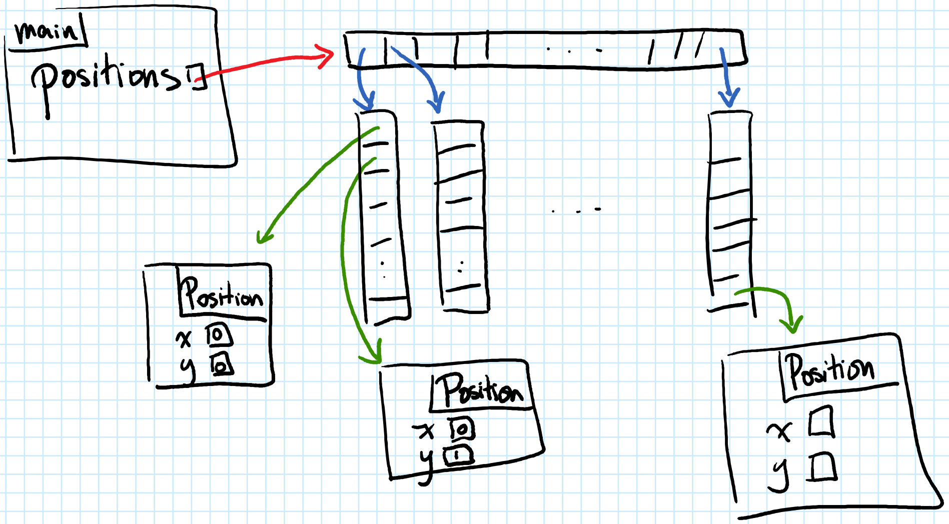
"positions" is a pointer to an array of pointers to arrays of pointers to Position objects

methods

int getWidth()
int getHeight()
void setWall(int x, int y)
List<Position*>* solveBreadthFirst()
List<Position*>* solveDepthFirst()

private method

List<Position*>* getNeighbors(Position* p)


main
positions
. . .
. . .
Position
x 0
y 0
Position
x 0
y 1
Position
x
y