Clue Paths

Movement Rules

- Players may move up, down, left and right but not diagonally.
- After the human player rolls the die, the program will highlight all possible target locations.
- Targets are all the possible grid squares that are on a path that has exactly that number of steps, unless you can enter a room (e.g., you can't roll a 3 but only move 2 squares, unless you are 2 squares from a doorway to a room).
- A cell may not occur more than once on a path (e.g., I can't move from (0,0) to (1,0) and then back to (0,0))

First step, understand the task

WHAT ARE WE TRYING TO DO?

The Challenge

Highlight all the squares that can be reached in # of steps shown on one die
Move only up/down/left/right (no diagonal)
Location should not appear more than once on a path
Highlight only the final squares, not the paths

Room	Room	Walk	Walk	Room
Room	Door	Walk	Walk	Room
Walk	Walk	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Room	Room	Walk	Walk	Room
Room	Door	Walk	Walk	Room
Walk	Walk	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Roll a 1

The Challenge

Room	Room	Walk	Walk	Room
Room	Room	Walk	Walk	Room
Walk	Door	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Room	Room	Walk	Walk	Room
Room	Room	Walk	Walk	Room
Walk	**Door**	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Roll a 2

Note that the Door is included. We always enter a door if possible, even if don't use full roll.

No Backtracking

Room	Room	Walk	Walk	Room
Room	Door	Walk	Walk	Room
Walk	Walk	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Room	**Room**	**Walk**	**Walk**	**Room**
Room	Door	Walk	Walk	Room
Walk	Walk	Walk	Walk	Room
Walk	Player	Walk	Walk	Walk
Walk	Walk	Walk	Walk	Walk

Roll a 4. (this is just one target)
Note that the person CANNOT backtrack… that is, visit the same cell twice. But the person is NOT required to go in a straight line. So you CANNOT just include all squares that are n away from the initial point.

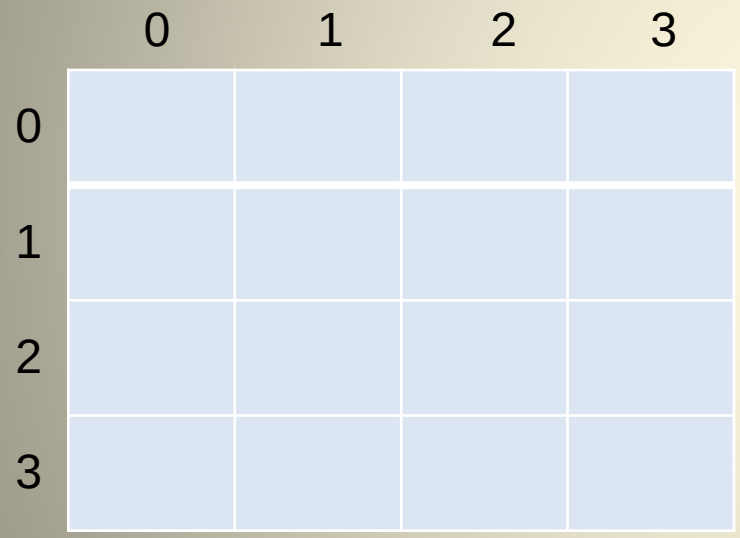
don’t do it all at once

START OFF WITH A SIMPLER PROBLEM

Simplify for now

Forget about doors and rooms. Let's do a simple 4x4 grid.

Number the rows and columns. Assume a 2D grid, each containing a board cell. Create an adjacency list.



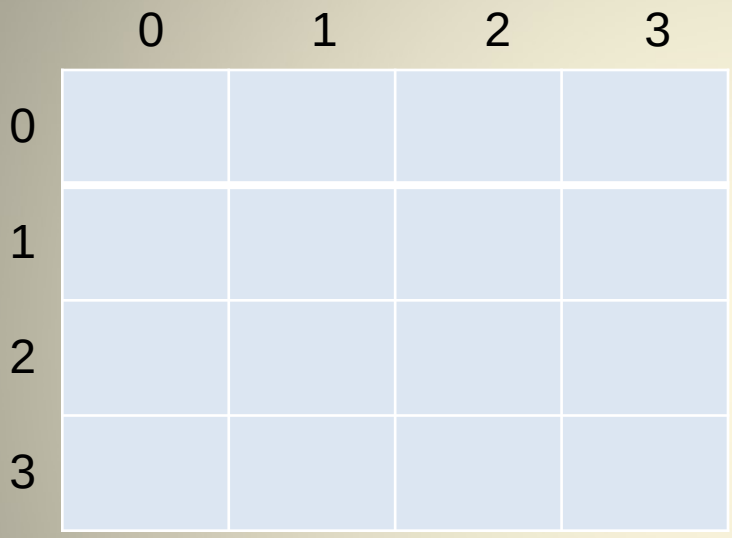
Adjacency List (sample entries)
[0][0] => { [0][1], [1][0] }

[1][1] => { [0][1], [1][0], [1][2], [2][1] }

NOTE: adjacency lists will be even more handy in the clue layout, because a square is not necessarily attached to each of its neighbors. By creating the adjacency list when the board is loaded, we can simplify the code for the target-finding algorithm.

How would you write a program to create an adjacency list?

Calculating the Adjacency Lists

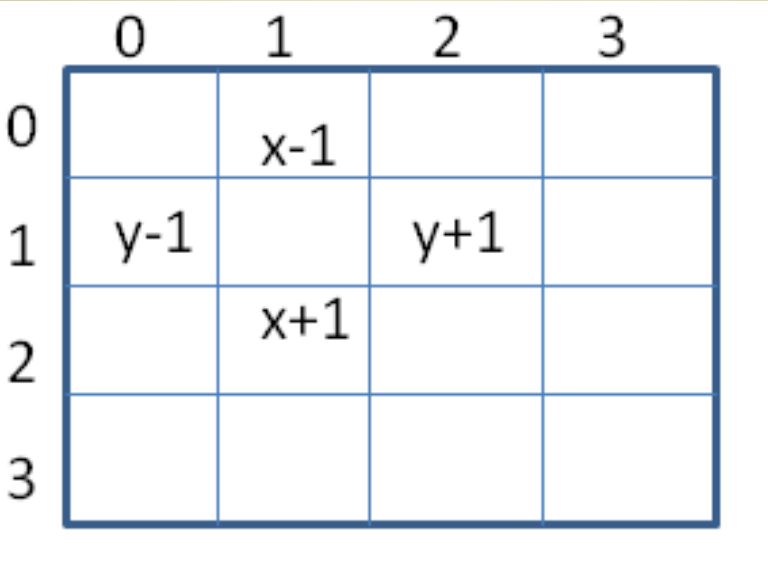


Notice that for each cell, you have at most 4 neighbors.

To calculate the adjacency list:
for each cell
  look at each neighbor
    if it is a valid index
      add it to the adjacency list

What data structure would you use for the adjacency list?
(we'll use a common structure – covered in a minute)

Calculating Neighbors



Choose Data Structures

Adjacency List (sample entries)

[0][0] => { [0][1], [1][0] }

[1][1] => { [0][1], [1][0], [1][2], [2][1] }

We want to be able to look up all the adjacencies for a given cell. This is a good application of a *map*:

```
private Map<BoardCell, Set<BoardCell>> adjMtx;
```

What is a **BoardCell**? For now, a simple class with a row and column. More will be added in the Clue game.

Note: Our adjacency list is now pointers (to BoardCell object), not x,y's.

What are some other alternatives?

- Set is a good choice for us, because it prevents duplicates (the algorithm can potentially identify the same target twice, but we don't need to worry about that)

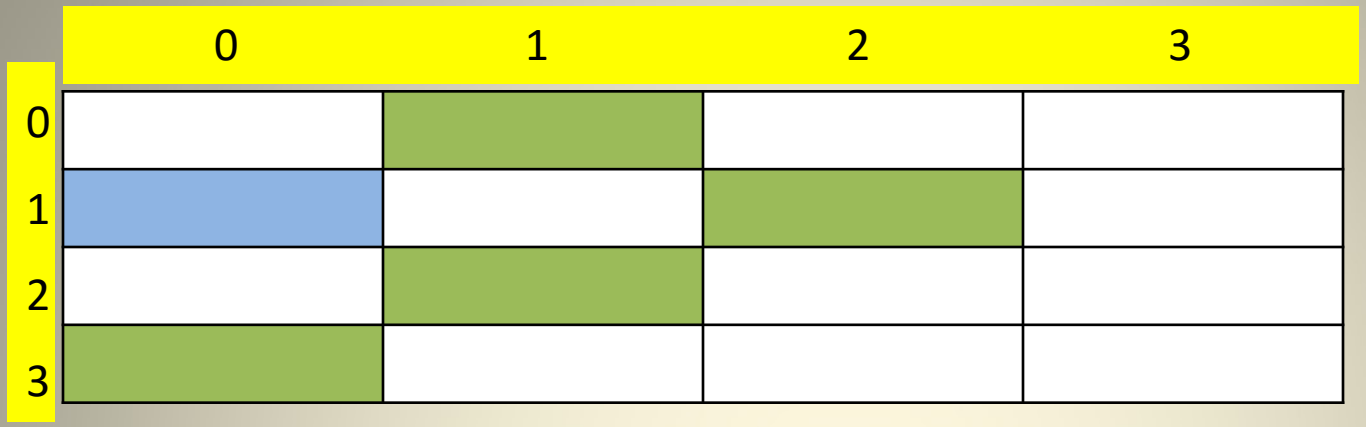
A walkthrough

THE ALGORITHM IN PSEUDOCODE

calcTargets

Goal: Targets with paths of length 2 from cell [1],[0]

Answer: Shown in green below



adjacency matrix should have been calculated prior to calling calcTargets!

calcTargets: Set up for recursive call

- **visited** list is used to avoid backtracking. Set to empty list.
- **targets** will ultimately contain the list of cells (e.g., [0][1], [1][2], [2][1], [3][0]) . Initially set to empty list.
- **add the start location to the visited list (so no cycle through this cell)**
- call recursive function
  - what will you name this function? I did **findAllTargets**
  - what parameters does it need? I used startCell and pathLength

findAllTargets

visited:
[4]

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

adjacencies
(in condensed form, 0 is [0][0] etc.)

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [0, 5, 8]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

findAllTargets pseudocode

Parameters: **thisCell** and **numSteps**

for each **adjCell** in adjacentCells
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell = **4,** numSteps=2

for each adjCell in **adjacentCells [0,5,8]**
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

no

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell =4, numSteps=2

for each **adjCell** in adjacentCells [**0,** 5, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, **0**]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell =4, numSteps=2

for each **adjCell** in adjacentCells [0, 5, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, **0**]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell =4, numSteps=**2**

for each **adjCell** in adjacentCells [0, 5, 8]

-- if already in **visited** list, skip rest of this

no

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: (none yet)

findAllTargets call: thisCell =4, numSteps=2

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

for each **adjCell** in adjacentCells [0, 5, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell = **0,** numSteps=1

for each **adjCell** in adjacentCells
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

NOTE: the recursive call is like the player putting the piece on that cell

findAllTargets

visited:

[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell = **0,** numSteps=1

for each adjCell in adjacentCells **[4, 1]**
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: (none yet)

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

yes

findAllTargets call: thisCell =0, numSteps=1

for each **adjCell** in adjacentCells **[4, 1]**
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

no

findAllTargets call: thisCell =0, numSteps=1

for each **adjCell** in adjacentCells [4, **1**]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 0, 1]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: (none yet)

findAllTargets call: thisCell =0, numSteps=1

for each **adjCell** in adjacentCells [4, **1**]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, 0, 1]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

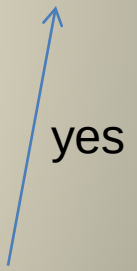
Targets: [1]

findAllTargets call: thisCell =0, **numSteps=1**

for each **adjCell** in adjacentCells [4, 1]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list



findAllTargets

visited:

[4, 0, ~~1~~]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: [1]

findAllTargets call: thisCell =0, **numSteps=1**

for each **adjCell** in adjacentCells [4, **1**]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

findAllTargets

visited:

[4, 0]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: [1]

findAllTargets call: thisCell =0, **numSteps=1**

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

for each **adjCell** in adjacentCells **[end of list, return]**
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, ~~0~~]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

for each **adjCell** in adjacentCells [**0,** 5, 8]

-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list

-- if **numSteps** == 1, add **adjCell** to Targets

-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

findAllTargets

visited:
[4]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

for each **adjCell** in adjacentCells [0, **5**, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

no

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

for each **adjCell** in adjacentCells [0, **5**, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 5]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

for each **adjCell** in adjacentCells [0, **5**, 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 5]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**8, 5, 0**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

no

for each **adjCell** in adjacentCells [0, **5,** 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:
[4, 5]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Targets: [1]

findAllTargets call: thisCell =4, **numSteps=2**

for each **adjCell** in adjacentCells [0, **5,** 8]
-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list
-- if **numSteps** == 1, add **adjCell** to Targets
-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

findAllTargets

visited:

[4, 5]

	0	1	2	3
0	**0**	1	2	3
1	4	**5**	6	7
2	**8**	9	10	11
3	12	13	14	15

Targets: **[1]**

findAllTargets call: thisCell = **5**, **numSteps=1**

for each **adjCell** in adjacentCells **[4, 9, 6, 1]**

-- if already in **visited** list, skip rest of this

-- add **adjCell** to **visited** list

-- if **numSteps** == 1, add **adjCell** to Targets

-- else call findAllTargets with **adjCell**, **numSteps-1**

-- remove **adjCell** from **visited** list

All adjacencies

0 => [4, 1]
1 => [0, 5, 2]
2 => [1, 6, 3]
3 => [2, 7]
4 => [**0, 5, 8**]
5 => [4, 9, 6, 1]
6 => [2, 5, 10, 7]
7 => [11, 3, 6]
8 => [12, 9, 4]
9 => [13, 10, 5, 8]
10 => [14, 11, 6, 9]
11 => [15, 7, 10]
12 => [13, 8]
13 => [14, 9, 12]
14 => [15, 10, 13]
15 => [11, 14]

Before you code

- Continue this trace (starting from the prior slide) on paper with your partner.
  - Currently one target has been identified: [1]
  - Trace enough of the code to add at least one more target
  - Remember that targets is a Set, so if you find the same target again, it just does not update the set
  - Make sure you understand when cells are added/removed from visited (big source of errors).
  - Be sure you understand that the algorithm has both iteration (through adjacency list) AND recursion.