LECTURE 15

RECURSION WITH BACKTRACKING

MCS 275 Spring 2021

Emily Dumas

LECTURE 15: RECURSION WITH BACKTRACKING

Course bulletins:

- Project 2 description available.
- Project 2 due 6pm CST Friday, February 26.
- Check out the recursion sample code.

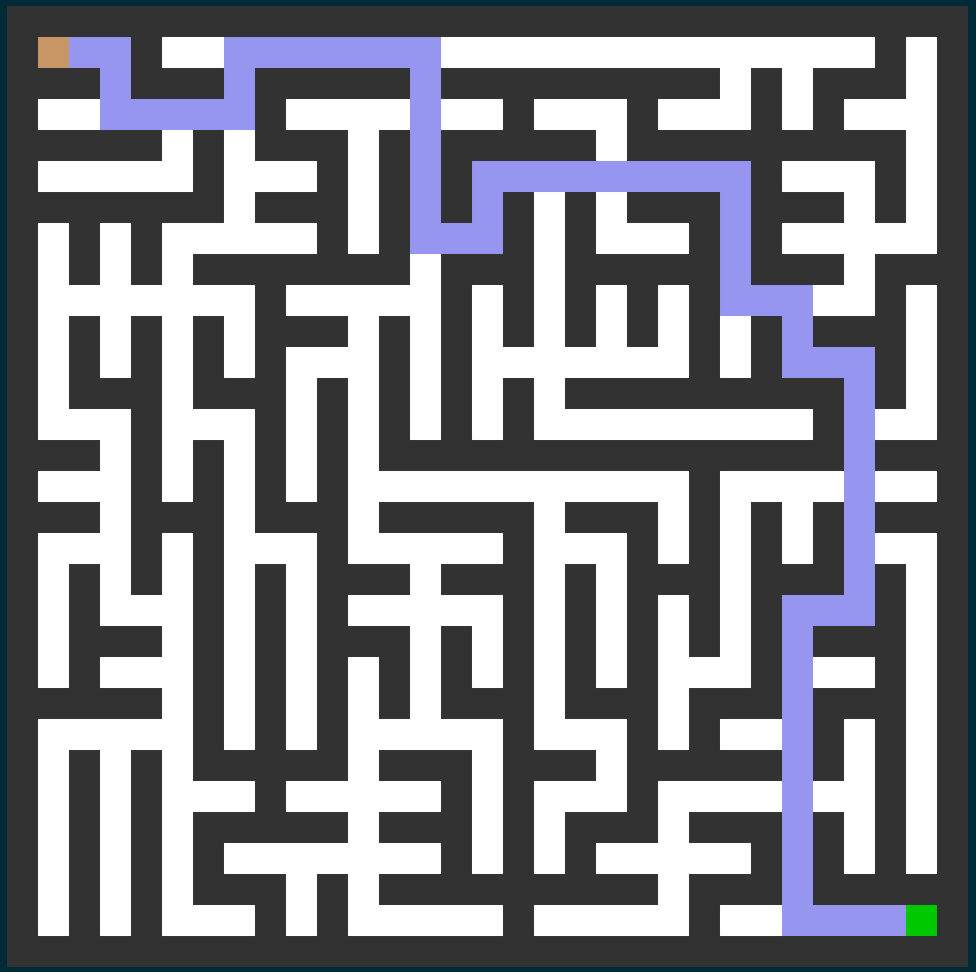
PLAN

- Talk about solving mazes with recursion
- Present a module for working with mazes
- Implement the maze solver

RECURSION WITH BACKTRACKING

How do you solve a maze?

RECURSION WITH BACKTRACKING



How do you solve a maze?

My guess at your mental algorithm:

- Try something (move around but don't return to anywhere you've visited).
- If you reach a dead end, go back a bit and reconsider which way to go at a recent intersection.

An algorithm that formalizes this is **recursion with backtracking**.

We make a function that takes:

- The maze
- The path so far

Its goal is to add one more step to the path, never backtracking, and call itself to finish the rest of the path.

But if it hits a dead end, it needs to notice that and **backtrack**.

BACKTRACKING

Backtracking is implemented through the return value of a recursive call.

Recursive call may return:

- A solution, or
- `None`, indicating that only dead ends were found.

Algorithm `depth_first_maze_solution`:

Input: a *maze* and a *path* under consideration (partial progress toward solution). Initially, the path is just the starting point.

1. If the path is a solution, just return it.
2. Otherwise, enumerate possible next steps (not returning to a location already in the path).
3. For each of the possible next steps:
    - Add it to the path under consideration.
    - Make a recursive call to attempt to complete this path to a solution.
    - If recursive call returns a solution, return it immediately.
    - Otherwise, remove the last step from the path and go to the next iteration.
4. If we get to this point, every continuation of the path is a dead end. Return `None`.

DEPTH FIRST

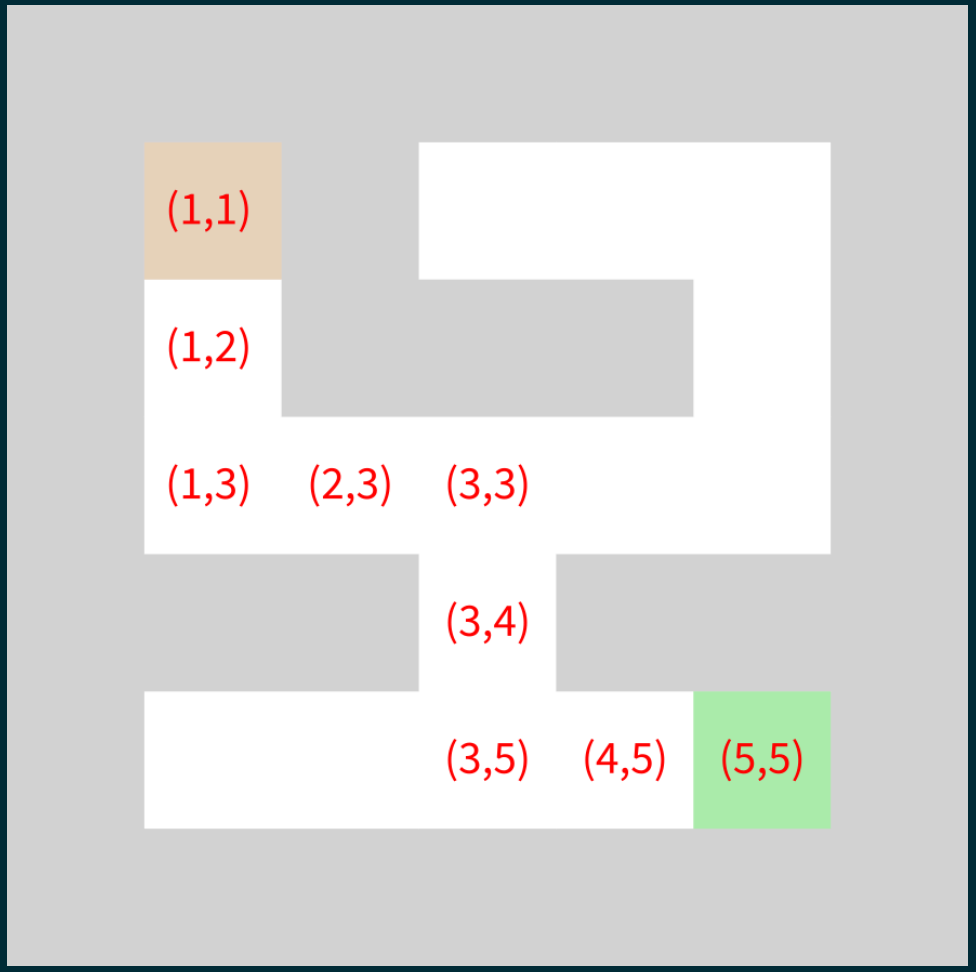
This method is also called a **depth first search** for a path through the maze.

Here, **depth first** means that we always add a new step to the path before considering any other changes (e.g. going back and modifying an earlier step).

MAZE COORDINATES

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)

MAZE COORDINATES



REFERENCES

No changes to the references from Lecture 13

- *Algorithms* by Jeff Erickson, Chapter 1.
- *Lutz* discusses recursive functions in Chapter 19 (pages 555-559 in the print edition).
- *Intro to Python for Computer Science and Data Science* by Deitel and Deitel, Chapter 11.
- *Think Python*, 2ed, by Allen B. Downey, Sections 5.8 to 5.10.
- *Computer Science: An Overview* by Brookshear and Brylow, Section 5.5.

REVISION HISTORY

- 2021-02-15 Initial publication