Today’s topics:

- Previous lecture:
  › Loops + recursion for *generating sequences and combinations*
- Today:
  › Loops + recursion for *recursive backtracking*

#MeToo
Backtracking
Maze solving
Backtracking

A particular behavior in recursive code where you tentatively explore many options, and recover to the nearest junction when you hit a “dead end,” so you can try a different path from that junction.

The easiest way to understand this is probably to see literal exploration and dead ends.
Maze-solving
Maze-solving

Thinking through the pseudo-code:

- Return true if there is a way to win from where we’re standing.
- Return false if there isn’t.

- From the start position, this amounts to saying, return true if there a way to solve the whole maze, otherwise false.
Backtracking template

- `bool backtrackingRecursiveFunction(args) {
  › Base case test for success: return true
  › Base case test for failure: return false
  › Loop over several options for “what to do next”:
    • Tentatively “do” one option
    • if (recursiveFunction() returns true) return true
    • else That tentative idea didn’t work, so “undo” that option
  › None of the options we tried in the loop worked, so return false
}
bool solveMaze(Maze & maze, Point start) {
    if (maze.isOutside(start)) return true;
    if (maze.isMarked(start)) return false;
    maze.markSquare(start);
    pause(200);
    for (Direction dir = NORTH; dir <= WEST; dir++) {
        if (!maze.wallExists(start, dir)) {
            if (solveMaze(maze, adjacentPoint(start, dir))) {
                return true;
            }
        }
    }
    maze.unmarkSquare(start);
    return false;
}
Maze-solving

In what order do we visit these spaces?

A. x1, x2, x3
B. x2, x3, x1
C. x1, x3, x2
D. We don’t visit all three
E. Other/none/more

//order of for loop:
enum Direction =
{NORTH, EAST, SOUTH, WEST};
The stack

What is the deepest the Stack gets (number of stack frames) during the solving of this maze?

A. Less than 5
B. 5-10
C. 11-20
D. More than 20
E. Other/none/more
Contrast: Recursive maze-solving vs. Word ladder

- With word ladder, you did **breadth-first search**
- This problem uses **depth-first search**

- Both are possible for maze-solving!

- The contrast between these approaches is a theme that you’ll see again and again in your CS career
Contrast: Recursive maze-solving vs. Word ladder

- With word ladder, you did **breadth-first search (BFS)**
  - Used a QUEUE of ladders

- This problem uses **depth-first search (DFS)**
  - Uses a STACK of locations (implicitly)
  - Can also do DFS with an explicit stack

- Both are possible for maze-solving!

- The contrast between these approaches is a theme that you’ll see again and again in your CS career