Programming Abstractions

CS106B

Cynthia Lee

Today’s topics:

- Previous lectures:
  - Introduction to recursion with Factorial
  - Mechanics of recursion: looking at the stack frames
  - Classic, widely-used CS algorithm example: Binary Search
  - Visual example: Boxy “snowflake” fractal
- Today:
  - New patterns of recursion application: adding loops
    - Loops + recursion for *generating combinations/permutations*
    - Loops + recursion for *recursive backtracking*

Generating combinations/permutations

Recursion pattern: generating permutations

- Example problems:
  - Given a deck of cards, output all possible distinct 5-card poker hands
  - Generate all strings of length N

- Pseudocode of the approach for “generate all strings of length N”:
  - In a loop, do:
    1. Choose a character to be the first letter, then recursively generate the rest of the string from the remaining characters
    2. Now let a different character be the first letter, …

Backtracking

Maze solving

Stanford University

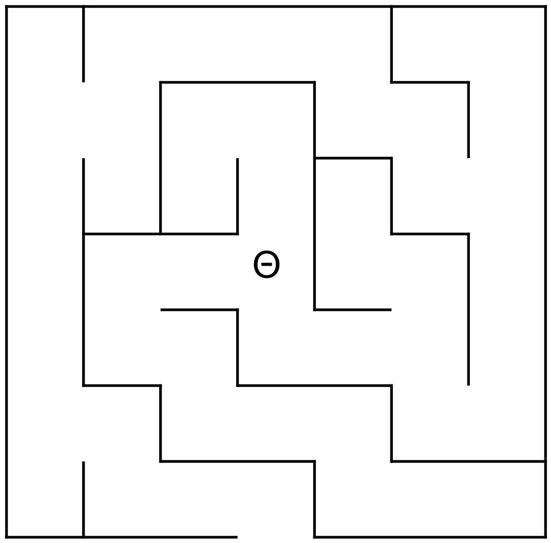
Backtracking

A particular behavior in recursive code where you tentatively explore many options, and recover to the nearest junction when you hit a "dead end"

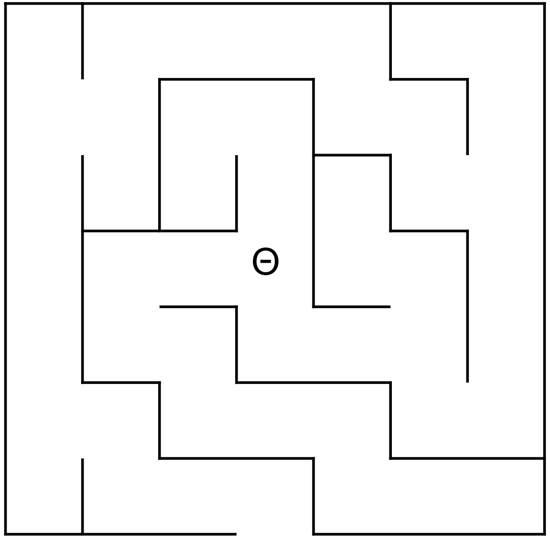
Similar to generating all permutations/combinations, but as you go along you evaluate each one to determine if it has potential to become successful in the future or not (and then "give up" early if not)

The easiest way to understand this is probably to see literal exploration and dead ends

Maze-solving



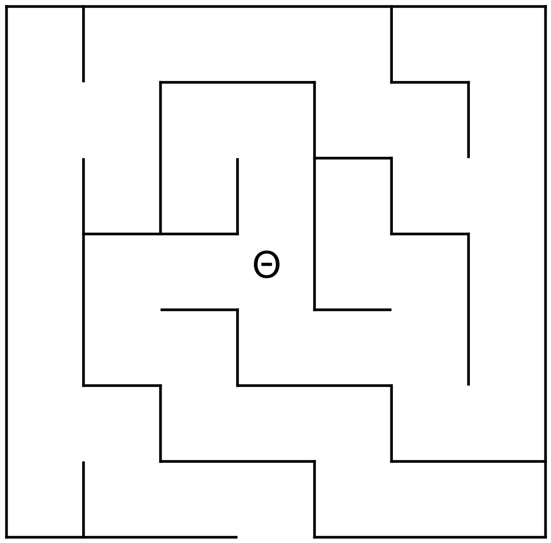
Maze-solving



Thinking through the pseudo-code:

- From position Θ, what does it mean for a step North to be a good idea?

Maze-solving



Thinking through the pseudo-code:

- From position Θ, what does it mean for a step South to be a good idea?
- It means that from position one-step-South-of-Θ, there exists some step that is a good idea…
- …Recursion!

Backtracking template

- **bool recursiveFunction(){**
  - 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()) return true
    - That tentative idea didn’t work, so “undo” that option
  - None of the options we tried in the loop worked, so return false

SolveMaze code

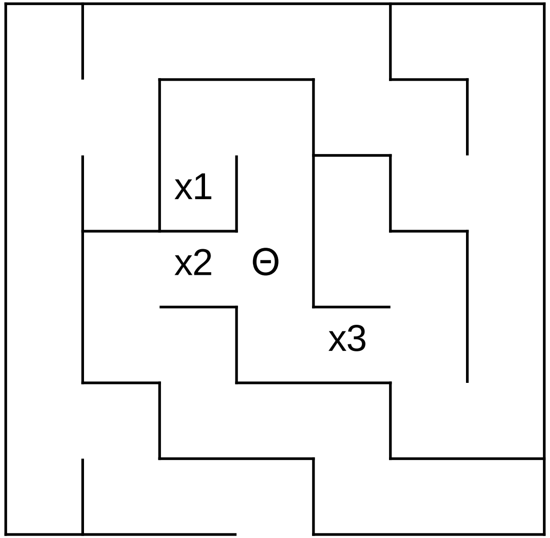
Adapted from the textbook by Eric Roberts

```
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;
}
```

```
enum Direction =
{NORTH, EAST, SOUTH,
WEST};
```

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

Maze-solving



In what order do we visit these spaces?

A. x1, x2, x3

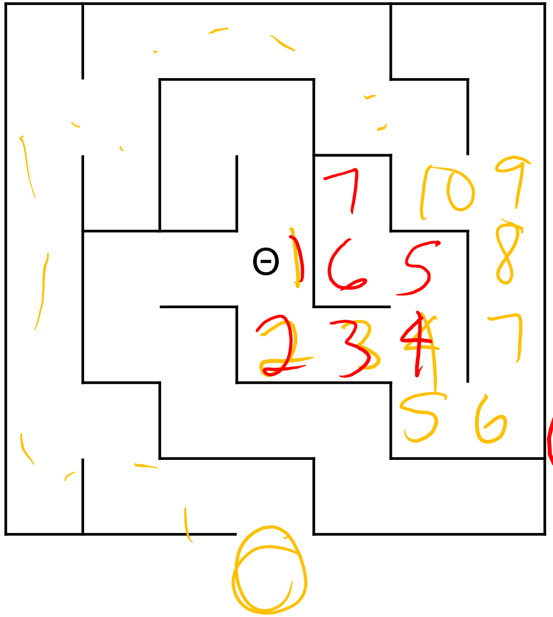
B. x2, x3, x1

C. x1, x3, x2

[✓ D. We don’t visit all three]

E. Other/none/more

The stack



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

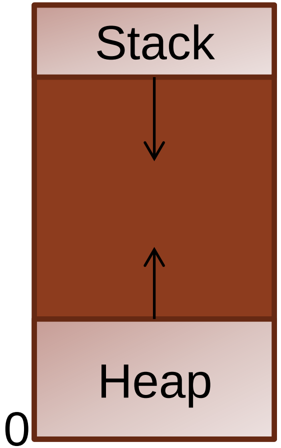
A. Less than 5

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