

CS 106B

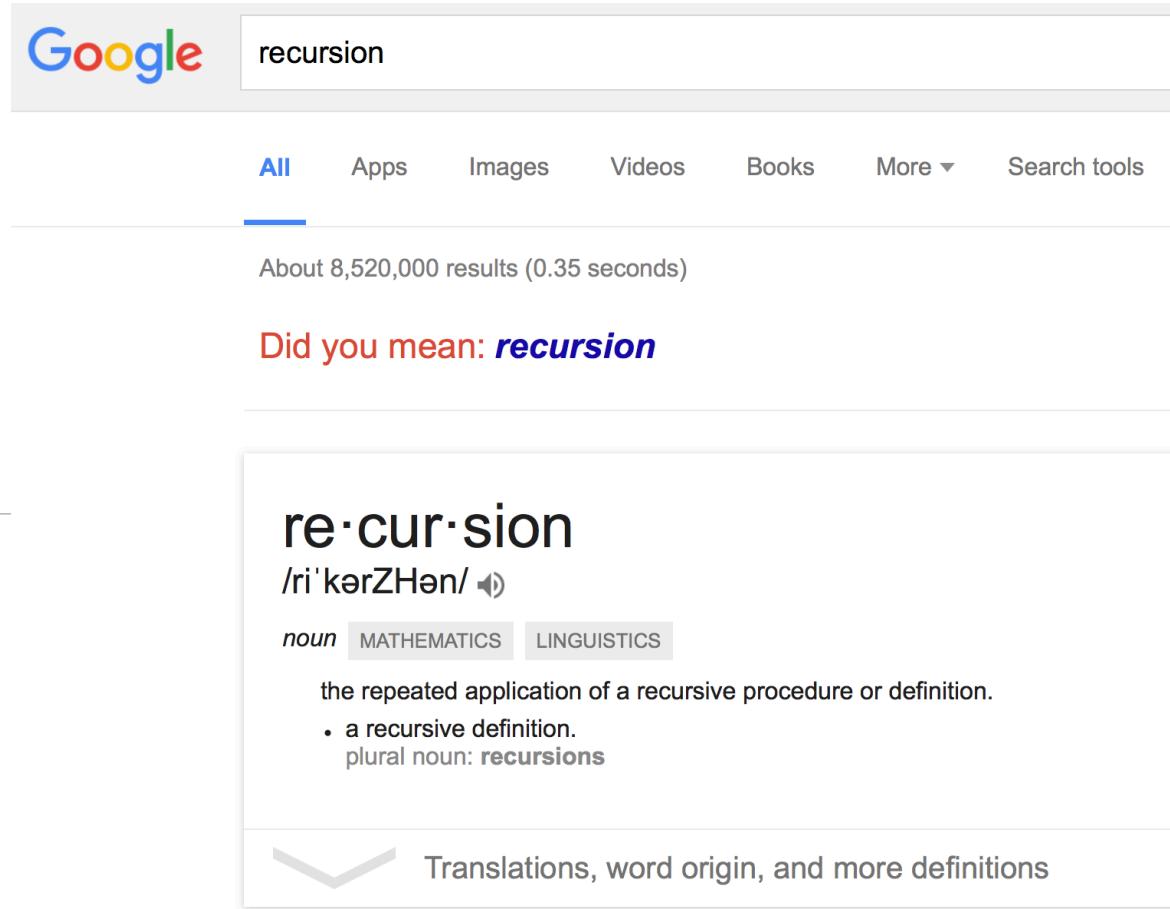
Lecture 10: Exhaustive Search and Recursive Backtracking

Monday, October 17, 2016

Programming Abstractions
Fall 2016
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 9



A screenshot of a Google search results page. The search query "recursion" is entered in the search bar. The results are filtered by "All". Below the search bar, it says "About 8,520,000 results (0.35 seconds)". A "Did you mean: recursion" suggestion is shown in red. The main result is a dictionary definition for "recursion".

re·cur·sion
/ri'kərZHən/ 
noun MATHEMATICS LINGUISTICS
the repeated application of a recursive procedure or definition.
• a recursive definition.
plural noun: **recursions**

 Translations, word origin, and more definitions



Today's Topics

- Logistics:
 - YEAH Hours for MetaAcademy today! 7-8pm, 420-041; Will be recorded
 - All midterm accommodations: let us know **by Friday at Noon.**
 - Midterm is Thursday, November 3rd!
- Cutting Edge Fauxtoshop
- More on Recursion, Trees, and "Exhaustive Search"
- Permutations
- Maze
- Recursive Backtracking



Cutting Edge Fauxtoshop

Chris Piech and Chris Gregg went to a CS Faculty Retreat this weekend. One of the speakers was Stanford Professor (emeritus) Mark Levoy, who now works at Google Research on cameras and photography.

Professor Levoy gave a talk about a side-project of his, called "Extreme imaging using cell phones."

This video demonstrates an amazing phone app he wrote.

Almost his entire talk involved manipulating images, in a more advanced but very similar way to your Fauxtoshop project — it is *amazing* what you can do with software these days! The camera was a normal Android camera, and all the manipulation was in software.

The following three slides show what he has accomplished — really amazing!





1/4 lux, iPhone 6S+





HDR+





SeeInTheDark, ~50 frames, handheld, real-time



Where We Are (Borrowed from Chris P)



Jumble

- Since 1954, the *JUMBLE* has been a staple in newspapers.
- The basic idea is to unscramble the anagrams for the words on the left, and then use the letters in the circles as another anagram to unscramble to answer the pun in the comic.
- As a kid, I played the puzzle every day, but some days I just couldn't descramble the words. Six letter words have $6! = 720$ combinations, which can be tricky!
- I figured I would write a computer program to print out all the permutations!

JUMBLE

Unscramble these four Jumbles,
one letter to each square,
to form four ordinary words.

KNIDY



©2015 Tribune Content Agency, LLC
All Rights Reserved.

LEGIA



CRONEE



TUVEDO



THAT SCRAMBLED WORD GAME

by David L. Hoyt and Jeff Knurek



Print answer here:



(Answers tomorrow)

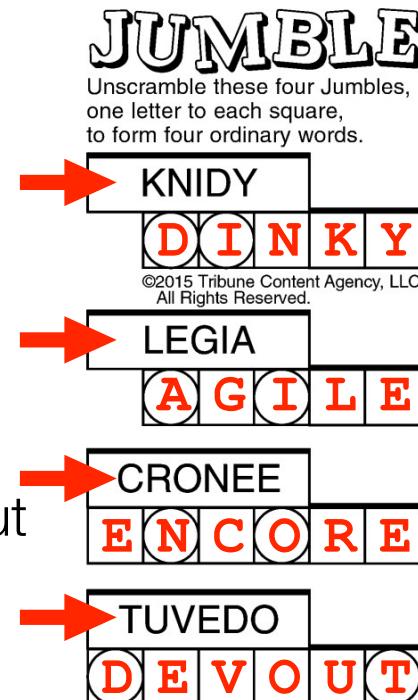
Saturday's

Jumbles: ELUDE JOINT AGENCY EASILY
Answer: The cyclops' son wanted an action figure for his birthday, so they bought him a — G- "EYE" JOE



Jumble

- Since 1954, the *JUMBLE* has been a staple in newspapers.
- The basic idea is to unscramble the anagrams for the words on the left, and then use the letters in the circles as another anagram to unscramble to answer the pun in the comic.
- As a kid, I played the puzzle every day, but some days I just couldn't descramble the words. Six letter words have $6! = 720$ combinations, which can be tricky!
- I figured I would write a computer program to print out all the permutations!



Print answer here: **A D D I T I O N**
D I A I N O D T

Saturday's | Jumbles: EL
Answer: The cyclops' son wanted an action figure for his birthday, so they bought him a — G- "EYE" JOE

THAT SCRAMBLED WORD GAME
by David L. Hoyt and Jeff Knurek



Permutations

My original function to print out all permutations of four letters:

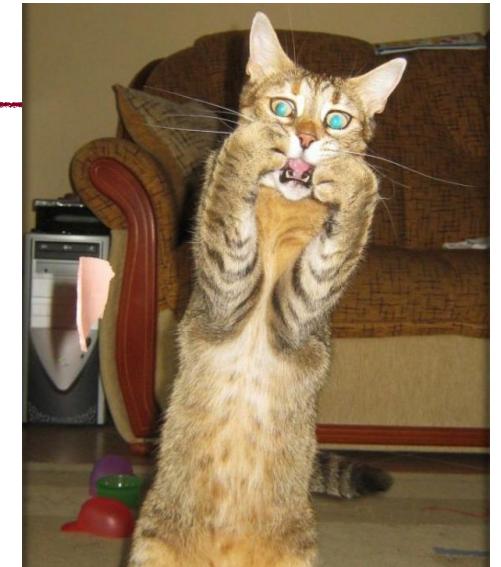
```
void permute4(string s) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4 ; j++) {
            if (j == i) {
                continue; // ignore
            }
            for (int k = 0; k < 4; k++) {
                if (k == j || k == i) {
                    continue; // ignore
                }
                for (int w = 0; w < 4; w++) {
                    if (w == k || w == j || w == i) {
                        continue; // ignore
                    }
                    cout << s[i] << s[j] << s[k] << s[w] << endl;
                }
            }
        }
    }
}
```



Permutations

I also had a permute5() function...

```
void permute5(string s) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5 ; j++) {
            if (j == i) {
                continue; // ignore
            }
            for (int k = 0; k < 5; k++) {
                if (k == j || k == i) {
                    continue; // ignore
                }
                for (int w = 0; w < 5; w++) {
                    if (w == k || w == j || w == i) {
                        continue; // ignore
                    }
                    for (int x = 0; x < 5; x++) {
                        if (x == k || x == j || x == i || x == w) {
                            continue;
                        }
                        cout << " " << s[i] << s[j] << s[k] << s[w] << s[x] << endl;
                    }
                }
            }
        }
    }
}
```



Permutations

And a permute6() function...

```
void permute6(string s) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5 ; j++) {
            if (j == i) {
                continue; // ignore
            }
            for (int k = 0; k < 5; k++) {
                if (k == j || k == i) {
                    continue; // ignore
                }
                for (int w = 0; w < 5; w++) {
                    if (w == k || w == j || w == i) {
                        continue; // ignore
                    }
                    for (int x = 0; x < 5; x++) {
                        if (x == k || x == j || x == i || x == w) {
                            continue;
                        }
                        for (int y = 0; y < 6; y++) {
                            if (y == k || y == j || y == i || y == w || y == x) {
                                continue;
                            }
                            cout << " " << s[i] << s[j] << s[k] << s[w] << s[x] << s[y] << endl;
                        }
                    }
                }
            }
        }
    }
}
```



What has been seen
cannot be un-seen

This is not tenable!



Tree Framework – Permutations

- Permutations do not lend themselves well to iterative looping because we are really *rearranging* the letters, which doesn't follow an iterative pattern.
- Instead, we can look at a recursive method to do the rearranging, called an *exhaustive algorithm*. We want to investigate all possible solutions. We don't need to know how many letters there are in advance!
- In pseudocode:

```
If you have no more characters left to rearrange, print current permutation
for (every possible choice among the characters left to rearrange) {
    Make a choice and add that character to the permutation so far
    Use recursion to rearrange the remaining letters
}
```
- In English:
 - The permutation starts with zero characters, as we have all the letters in the original string to arrange. The base case is that there are no more letters to arrange.
 - Take one letter from the letters left, add it to the current permutation, and recursively continue the process, decreasing the characters left by one.



Tree Framework – Permutations

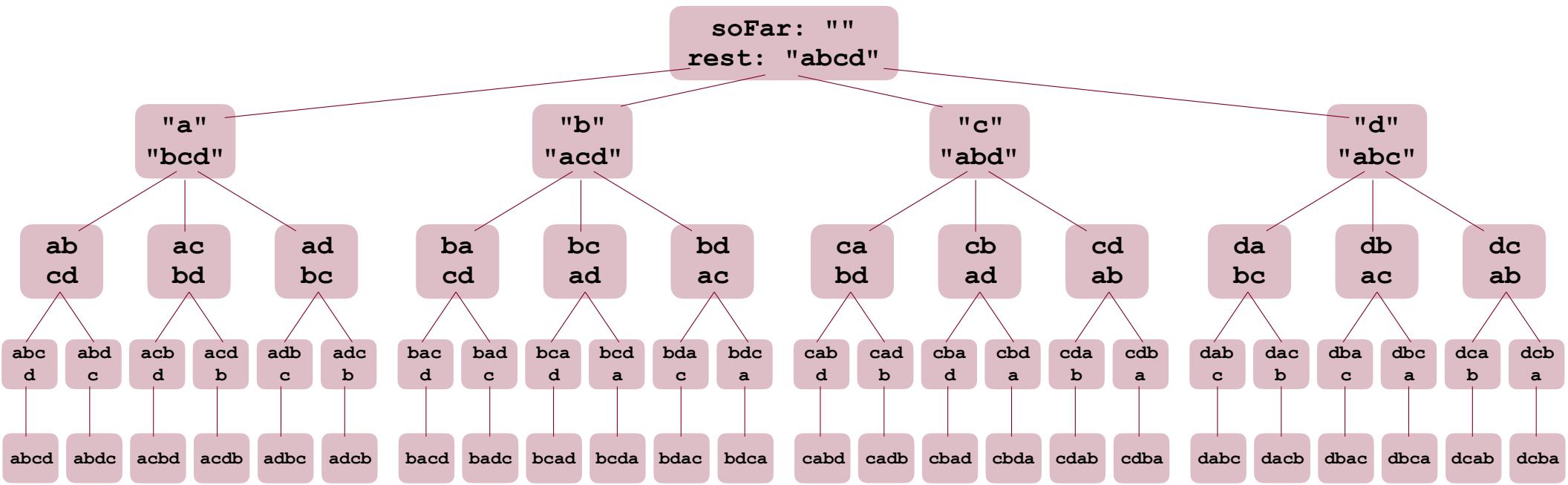
- The algorithm in C++:

```
void permute(string soFar, string rest) {  
    if (rest == "") {  
        cout << soFar << endl;  
    } else {  
        for (int i = 0; i < rest.length(); i++) {  
            string remaining = rest.substr(0, i) + rest.substr(i+1);  
            permute(soFar + rest[i], remaining);  
        }  
    }  
}
```

- Example call:
 - **recPermute ("", "abcd") ;**



Tree Framework – Permutations



This is a tree!

- ✓ Exhaustive
- ✓ Works for any length string
- ✓ $N!$ different results
- ✓ Can think of this as a "call tree" or a "decision tree"



Tree Framework – Helper functions

- Here is the algorithm again:

```
void permute(string soFar, string rest) {  
    if (rest == "") {  
        cout << soFar << endl;  
    } else {  
        for (int i = 0; i < rest.length(); i++) {  
            string remaining = rest.substr(0, i) + rest.substr(i+1);  
            permute(soFar + rest[i], remaining);  
        }  
    }  
}
```

- Some might argue that this isn't a particularly good function, because it requires the user to always start the algorithm with the empty string for the **soFar** parameter. It's ugly, and it exposes our internal parameter.
- What we really want is a **permute(string s)** function that is cleaner.
- We can re-name the function above **permuteHelper()** (and change the inner call, as well!), and have a cleaner permute function that calls this one.



Tree Framework – Helper functions

- The cleaner interface:

```
void permuteHelper(string soFar, string rest) {  
    if (rest == "") {  
        cout << soFar << endl;  
    } else {  
        for (int i = 0; i < rest.length(); i++) {  
            string remaining = rest.substr(0, i) + rest.substr(i+1);  
            permuteHelper(soFar + rest[i], remaining);  
        }  
    }  
}  
  
void permute(string s) {  
    permuteHelper("", s);  
}
```

- Now, a user only has to call **permute("tuvedo")**, which hides the helper recursion parameter.



Backtracking

- *Backtracking* is a method for trying partial solutions to some search and abandoning them if they aren't suitable or successful.
- A classic example is solving a maze: if you go down one path and it isn't the correct path, then you backtrack to your last decision point to try an alternate path.
- If you are using an object passed by reference you need to either *undo* (or "un-choose") paths that fail, or somehow mark them in your object.
- For a maze, for instance, you don't want to try and traverse the same path twice, so you need to mark whether you have been down that path before.



Billy Mays Maize Maze



Maze Solving

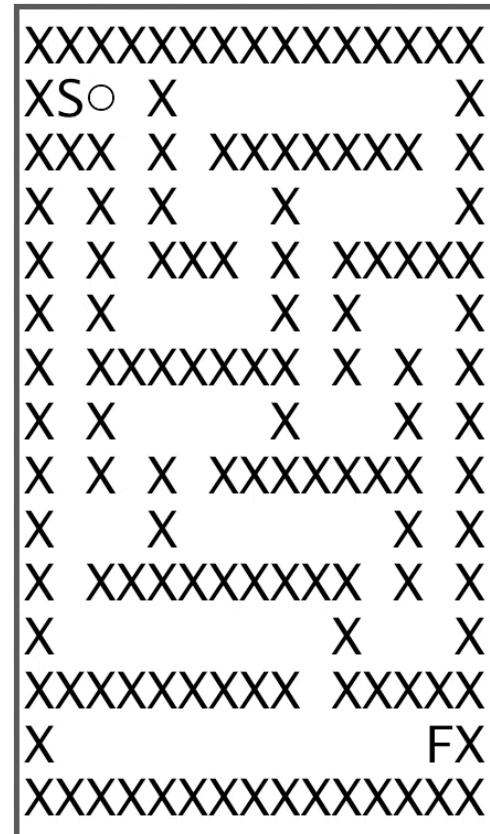
- The code for today's class includes a text-based recursive maze creator and solver.
- The mazes look like the one to the right
 - There is a Start (marked with an "S") and a Finish (marked with an "F").
 - The Xs represent walls, and the spaces represent paths to walk through the maze.

XXXXXXXXXXXXXXXXXXXX
XS X X
XXX X XXXXXXXX X
X X X X X
X X XXX X XXXXX
X X X X X
X XXXXXXXX X X X
X X X X X
X X X XXXXXXXX X
X X X X
X XXXXXXXXXX X X
X X X
XXXXXX XXXX XXXXX
X FX
XXXXXXXXXXXXXXXXXXXX

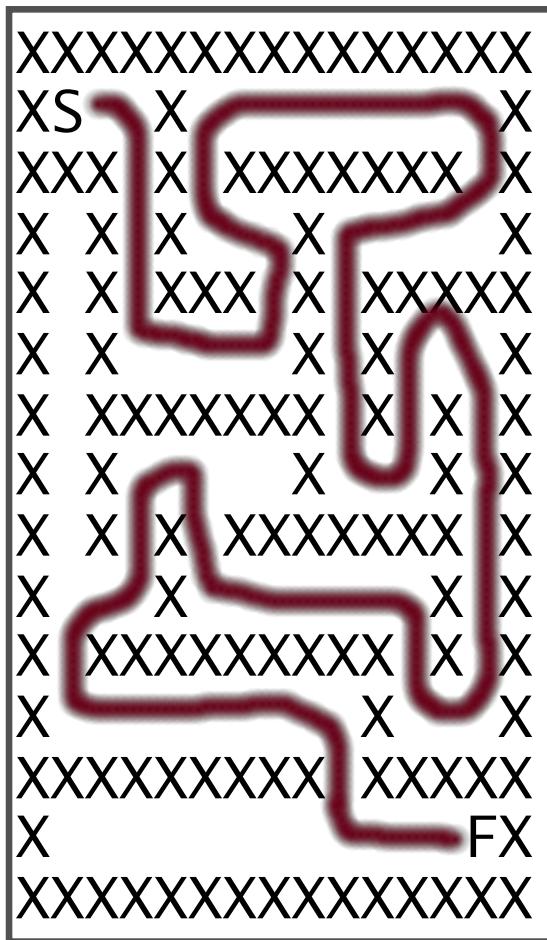


Maze Solving

- The solution to the maze is shown here (video):



Maze Solving

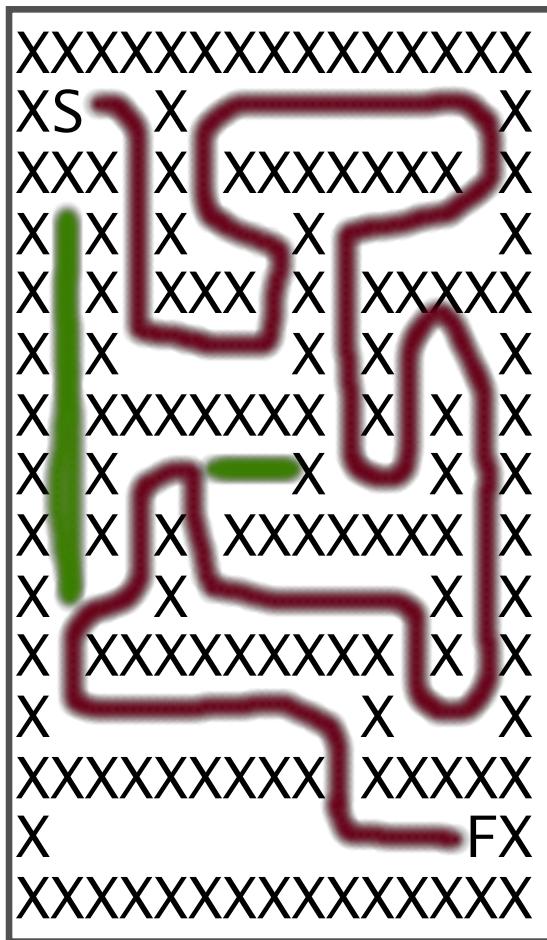


- The program will put dots in the correct positions.

XXXXXXXXXXXXXXXXXX																
XS	.	X	X	
XXX	.	X	.	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	



Maze Solving



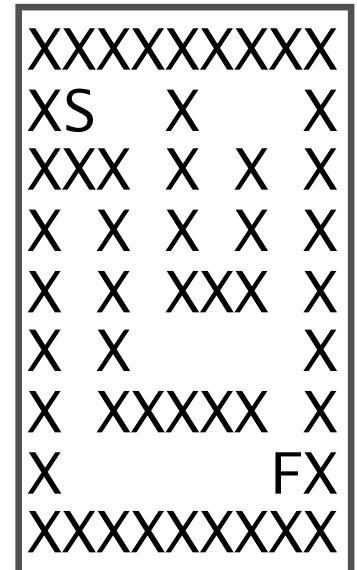
- The program will put dots in the correct positions.
 - But, it will also put lowercase b's when it goes in the wrong direction and has to backtrack.

XXXXXXXXXXXXXXXXXXXX
XS . X X
XXX . X . XXXXXXXX . X
XbX . X X X
XbX . XXX . X . XXXXX
XbX X . X . . . X
XbXXXXXXXX . X . X . X
XbX . . . bbX . . . X . X
XbX . X . XXXXXXXX . X
X . . . X X . X
X . XXXXXXXXXX . X . X
X X . . . X
XXXXXXXXXXXX . XXXXX
X FX
XXXXXXXXXXXXXX



Maze Solving

- What are some actual methods for solving a maze?
 - "Hand on a wall" -- put one hand on a wall at the start and keep following. Eventually you will reach the finish (circular paths may disrupt this method).
 - Break through walls (best for Corn Mazes)
 - Backtracking! Keep track of where you've been, and **systematically test all solutions.** Pick compass directions in order (e.g., N/E/S/W), returning to check other paths when you hit dead ends and have tried all combinations.
- Let's use the backtracking method to solve the maze to the right -- we will go N/E/S/W, from the Start.



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.

	0	1	2	3	4	5	6	7	8
0	XXXXXXXXXX								
1	X S	X							
2	X X X	X	X	X					
3	X	X	X	X	X				
4	X	X	XXX	X					
5	X	X							
6	X	XXXXX	X						
7	X						F X		
8	XXXXXXXXXX								



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- **Start: row=1 and col=1, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	XXXXXXXXXX								
1	X .	X							
2	XXX	X	X	X					
3	X	X	X	X	X				
4	X	X	XXX	X					
5	X	X							
6	X	XXXXX	X						
7	X						FX		
8	XXXXXXXXXX								



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- **We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.**
- **Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,**
- **Trying east, row=1 and col=2, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	.	X	X	X	X	X	X	X	X
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							X
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- **We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.**
- **Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,**
- **Trying east, row=1 and col=2, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X	X	X	X	X	X	X
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							X
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.
- Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,
- Trying east, row=1 and col=2, Marking with period (.)
- **Trying north, row=0 and col=2, Hit wall! Back at row=1 and col=2,**
- **Trying east, row=1 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X	X	X	X	X	X	X
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.
- Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,
- Trying east, row=1 and col=2, Marking with period (.)
- **Trying north, row=0 and col=2, Hit wall! Back at row=1 and col=2,**
- **Trying east, row=1 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X						
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.
- Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,
- Trying east, row=1 and col=2, Marking with period (.)
- Trying north, row=0 and col=2, Hit wall! Back at row=1 and col=2,
- Trying east, row=1 and col=3, Marking with period (.)
- **Trying north, row=0 and col=3, Hit wall! Back at row=1 and col=3,**
- **Trying east, row=1 and col=4, Hit wall! Back at row=1 and col=3,**
- **Trying south, row=2 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X					X
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							X
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- We will mark positions we have seen with a period ('.'), and mark backtracking with 'b'.
- Start: row=1 and col=1, Marking with period (.)
- We have to try all paths, N/E/S/W, and if we hit a wall ('X'), we can't go that direction.
- Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,
- Trying east, row=1 and col=2, Marking with period (.)
- Trying north, row=0 and col=2, Hit wall! Back at row=1 and col=2,
- Trying east, row=1 and col=3, Marking with period (.)
- **Trying north, row=0 and col=3, Hit wall! Back at row=1 and col=3,**
- **Trying east, row=1 and col=4, Hit wall! Back at row=1 and col=3,**
- **Trying south, row=2 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X					X
2	X	X	.	X	X	X	X		X
3	X	X	X	X	X	X	X		X
4	X	X	X	X	X	X	X		X
5	X	X							X
6	X	X	X	X	X	X	X		X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X	X	X	X	X	X	X
2	X	X	.	X	X	X	X	X	X
3	X	X	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X	X	X	X	X	X	X	X
6	X	X	X	X	X	X	X	X	X
7	X								FX
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- **Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,**
- **Trying south, row=3 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X						X
2	X	X	.	X		X		X	X
3	X	X	X	X	X				X
4	X	X	XXX						X
5	X	X							X
6	X	XXXXX							X
7	X							FX	
8	X	XXXXX							X



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- **Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,**
- **Trying south, row=3 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X						X
2	X	X	.	X		X		X	X
3	X	X	.	X	X	X	X	X	X
4	X	X	X	XXX	X				X
5	X	X							X
6	X	XXXXX	X						
7	X							FX	
8	X	XXXXX	XXX						



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,
- Trying south, row=3 and col=3, Marking with period (.)
- **Trying north, row=2 and col=3, We came from here! Back at row=3 and col=3,**
- **Trying east, row=3 and col=4, Hit wall! Back at row=3 and col=3,**
- **Trying south, row=4 and col=3, Marking with period (.)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X					X
2	X	X	.	X	X		X	X	X
3	X	X	.	X	X	X	X	X	X
4	X	X	X	X	X	X	X	X	X
5	X	X							X
6	X	X	X	X	X	X	X	X	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,
- Trying south, row=3 and col=3, Marking with period (.)
- **Trying north, row=2 and col=3, We came from here! Back at row=3 and col=3,**
- **Trying east, row=3 and col=4, Hit wall! Back at row=3 and col=3,**
- **Trying south, row=4 and col=3, Marking with period (.)**

...
(continues)

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X					X
2	X	X	.	X	X	X	X		X
3	X	X	.	X	X	X	X		X
4	X	X	.	XXX	X				X
5	X	X							X
6	X		XXXXX	X					
7	X							F	X
8	X	XXXXXX	XXXXXX	X					



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,
- Trying south, row=3 and col=3, Marking with period (.)
- Trying north, row=2 and col=3, We came from here! Back at row=3 and col=3,
- Trying east, row=3 and col=4, Hit wall! Back at row=3 and col=3,
- Trying south, row=4 and col=3, Marking with period (.)

...

(continues)

What happens here?

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X						X
2	X	X	.	X	X	X	X	X	X
3	X	X	.	X	X	X	X	X	X
4	X	X	.	XXX	X				
5	X	X	X				X
6	X	X	XXXXX	X					
7	X							F	X
8	X	XXXXXXXXXX							



Maze Solving

- Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,
- Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,
- Trying south, row=3 and col=3, Marking with period (.)
- Trying north, row=2 and col=3, We came from here! Back at row=3 and col=3,
- Trying east, row=3 and col=4, Hit wall! Back at row=3 and col=3,
- Trying south, row=4 and col=3, Marking with period (.)

...

(continues)

What happens here?

Bummer. We check North first, so we start going up.

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X					X
2	X	X	.	X	X	X	X		X
3	X	X	.	X	X	X	X		X
4	X	X	.	XXX	X				
5	X	X					X
6	X	X	XXXXX	X					
7	X							F	X
8	X	XXXXXX	XXXXXX	X					



Maze Solving

Now what?

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	.	.	.	X	.	X	.	X	.
2	X	X	X	.	X	.	X	.	X
3	X	X	.	X	X	.	X	X	.
4	X	X	.	X	X	X	X	.	X
5	X	X	X	.
6	X	X	X	X	X	X	X	.	X
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X	.	.	X		
2	X	X	.	X	.	X	.	X	
3	X	X	.	X	X	.	X		
4	X	X	.	XXX	.	X			
5	X	X	.	.	.	X			
6	X	XXXXX	X						
7	X							FX	
8	X	XXXXX	XXX	X					



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

What is next?

How did we get here? From the North, meaning we **checked South to get here.**

So, **we now check West (remember, we are checking N/E/S/W)**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	.	.	.	X	.	.	X	.	X
2	X	X	X	.	X	.	X	.	X
3	X	X	.	X	b	X	.	X	X
4	X	X	.	XXX	.	X			
5	X	X	X		
6	X	X	XXXXX	X					
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

What is next?

How did we get here? From the North, meaning we checked South to get here.

So, we now check West (remember, we are checking N/E/S/W)

- Trying west, row=2 and col=4, Hit wall! Back at row=2 and col=5,
- Failed. Marking bad path with b. Back at row=1 and col=5,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	X	.	X	X	X	X	X
2	X	X	.	X	.	X	X	X	X
3	X	X	.	b	X	X	X	X	X
4	X	X	.	XXX	.	X	X	X	X
5	X	X	.	.	.	X	X	.	X
6	X	XXXXX	X						
7	X								FX
8	X	XXXXXX	XXX						



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

What is next?

How did we get here? From the North, meaning we checked South to get here.

So, we now check West (remember, we are checking N/E/S/W)

- Trying west, row=2 and col=4, Hit wall! Back at row=2 and col=5,
- Failed. Marking bad path with b. Back at row=1 and col=5,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X	.	.	X	.	X
2	X	X	.	X	b	X	.	X	X
3	X	X	.	X	b	X	.	X	X
4	X	X	.	XXX	.	X			X
5	X	X	.	.	.	X			X
6	X		XXXXX	X					
7	X								FX
8	X	XXXXXX	X						



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

What is next?

How did we get here? From the North, meaning we checked South to get here.

So, we now check West (remember, we are checking N/E/S/W)

Trying west, row=2 and col=4, Hit wall! Back at row=2 and col=5,
Failed. Marking bad path with b. Back at row=1 and col=5,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X	.	.	X	.	X
2	X	X	.	X	b	X	.	X	X
3	X	X	.	X	b	X	.	X	X
4	X	X	.	XXX	.	X	X	.	X
5	X	X	.	.	.	X	X	.	X
6	X	XXXXX	X						
7	X								FX
8	X	XXXXXX	XXX						

Now, we are "remembering" where we have been because we've been keeping track of our positions and what we last checked at a given position -- we will use recursion to do this!



Maze Solving

- Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,
- Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,
- Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,
- Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,
- Failed. Marking bad path with b. Back at row=2 and col=5,

What is next?

How did we get here? From the North, meaning we checked South to get here.
So, we now check West (remember, we are checking N/E/S/W)

Trying west, row=2 and col=4, Hit wall! Back at row=2 and col=5,
Failed. Marking bad path with b. Back at row=1 and col=5,

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	.	.	.	X	.	b	b	b	X
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	XXX	b	X		
5	X	X			X
6	X	X	XXXXX	X					
7	X								FX
8	X	XXXXXX							

Now, we are "remembering" where we have been because we've been keeping track of our positions and what we last checked at a given position -- we will use recursion to do this!

We will arrive back at row=5, col=7 quickly.



Maze Solving

- Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,
- Trying south, row=6 and col=7, Marking with period (.)

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	X	X	X	X	X	X	X
7	X								FX
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,
- Trying south, row=6 and col=7, Marking with period (.)

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	.	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	XXXXX	.	X				
7	X								FX
8	X	XXXXXX	XXX	XX	XX	XX	XX	XX	X



Maze Solving

- Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,
- Trying south, row=6 and col=7, Marking with period (.)
- **Trying north, row=5 and col=7, We came from here! Back at row=6 and col=7,**
- **Trying east, row=6 and col=8, Hit wall! Back at row=6 and col=7,**
- **Trying south, row=7 and col=7, Found the Finish!**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	X	X	X	X	X	.	X
7	X								FX
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,
- Trying south, row=6 and col=7, Marking with period (.)
- **Trying north, row=5 and col=7, We came from here! Back at row=6 and col=7,**
- **Trying east, row=6 and col=8, Hit wall! Back at row=6 and col=7,**
- **Trying south, row=7 and col=7, Found the Finish!**

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	S	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	b	X	
5	X	X	X	
6	X	X	X	X	X	X	.	X	
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,
- Trying south, row=6 and col=7, Marking with period (.)
- Trying north, row=5 and col=7, We came from here! Back at row=6 and col=7,
- Trying east, row=6 and col=8, Hit wall! Back at row=6 and col=7,
- Trying south, row=7 and col=7, Found the Finish!

The total number of steps: 71!

That seems like a lot of steps to solve such a small maze, but remember, we are going through a methodical process that *must check all paths*.

(see extra slides for all steps for this maze)

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	S	.	X	b	b	b	X		
2	X	X	.	X	b	X	b	X	
3	X	X	X	.	X	b	X	b	X
4	X	X	X	X	.	X	b	X	
5	X	X	X
6	X	X	X	X	X	X	.	X	
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

- Our recursive backtracking method for solving mazes must follow the same rules for all recursion:
 - (1) have a case for all valid inputs,
 - (2) must have base cases,
 - (3) make forward progress towards the base case.

Let's start with the base cases. How many are there?

- This actually won't happen for our mazes, because we have surrounded all paths with walls.

(2) If we hit a backtracked position ('b')

 - Also won't happen, because once we mark as backtracked, we'll never get there again.

(3) If we hit a wall ('X')

(4) If we hit a position we have seen before ('.')

(5) If we find the finish ('F')



Maze Solving

Base cases:

Returning **true** means we have solved the maze!

Returning **false** means that this path does not solve the maze.

```
bool solveMazeRecursive(int row, int col, Grid<int> &maze) {  
  
    if (maze[row][col] == 'X') {  
        return false;  
    }  
  
    if (maze[row][col] == '.') {  
        return false;  
    }  
  
    if (maze[row][col] == 'F') {  
        return true;  
    }  
}
```

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	S	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	X	X	X	X	X	X	.
7	X								F
8	X	X	X	X	X	X	X	X	X



Maze Solving

Once we take care of our base cases, we'd better mark the position we are at!

```
bool solveMazeRecursive(int row, int col, Grid<int> &maze) {  
  
    if (maze[row][col] == 'X') {  
        return false;  
    }  
  
    if (maze[row][col] == '.') {  
        return false;  
    }  
  
    if (maze[row][col] == 'F') {  
        return true;  
    }  
  
    maze[row][col] = '.';  
}
```

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	S	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	X	X	X	X	X	X	.
7	X								F
8	X	X	X	X	X	X	X	X	X



Maze Solving

Now we can recurse -- we have to check all directions!

```
bool solveMazeRecursive(int row, int col, Grid<int> &maze) {  
    ...  
  
    maze[row][col] = '.';  
  
    // Recursively call solveMazeRecursivePrint(row,col)  
    // for north, east, south, and west  
    // If one of the positions returns true, then return true  
  
    // north  
    if (solveMazeRecursivePrint(row-1,col,maze) == true) {  
        return true;  
    }  
    ...  
}
```

	0	1	2	3	4	5	6	7	8
	12345678								
0	X	X	X	X	X	X	X	X	X
1	X	S	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	X	b	X
5	X	X	X
6	X	X	X	X	X	X	X	X	.
7	X								F
8	X	X	X	X	X	X	X	X	X



Maze Solving

All four recursions. If all four return, we have to backtrack!

```
bool solveMazeRecursive(int row, int col, Grid<int> &maze) {
    ...
    // north
    if (solveMazeRecursive(row-1,col,maze) == true) {
        return true;
    }

    // east
    if (solveMazeRecursive(row,col+1,maze) == true) {
        return true;
    }

    // south
    if (solveMazeRecursive(row+1,col,maze) == true) {
        return true;
    }

    // west
    if (solveMazeRecursive(row,col-1,maze) == true) {
        return true;
    }

    maze[row][col] = 'b';
    return false;
}
```

	0	1	2	3	4	5	6	7	8
0	X	X	X	X	X	X	X	X	X
1	X	S	.	X	b	b	b	X	
2	X	X	.	X	b	X	b	X	
3	X	X	.	X	b	X	b	X	
4	X	X	.	X	X	X	b	X	
5	X	X	X	
6	X	X	X	X	X	X	.	X	
7	X							F	X
8	X	X	X	X	X	X	X	X	X



Maze Solving

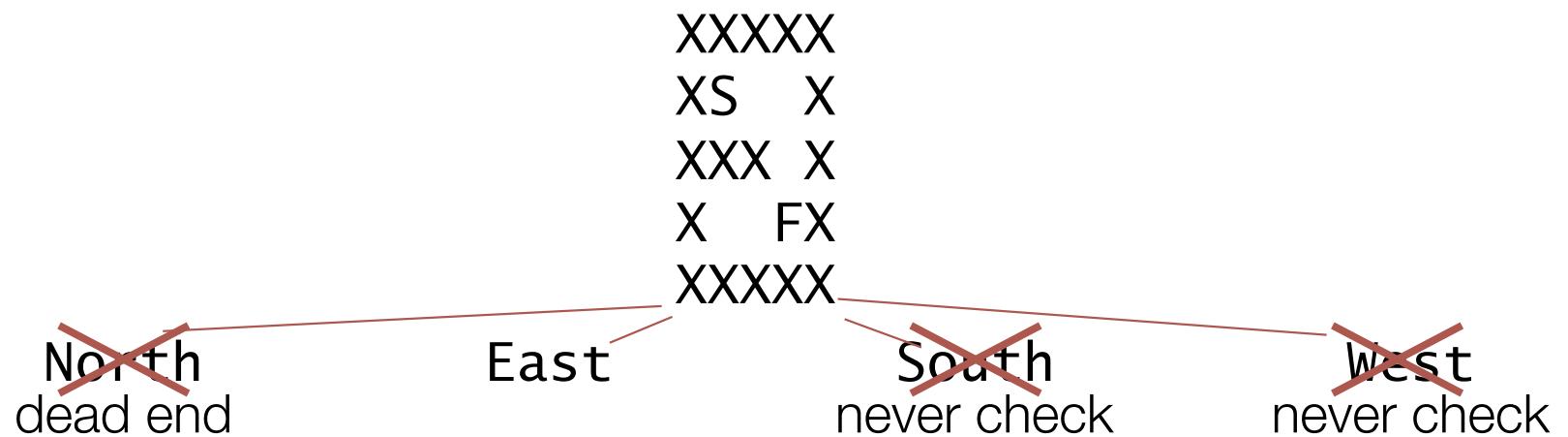
The entire recursive function (not too long!)

```
bool solveMazeRecursive(int row, int col, Grid<int> &maze) {  
    if (maze[row][col] == 'X') {  
        return false;  
    }  
  
    if (maze[row][col] == '.') {  
        return false;  
    }  
  
    if (maze[row][col] == 'F') {  
        return true;  
    }  
  
    maze[row][col] = '.';
```

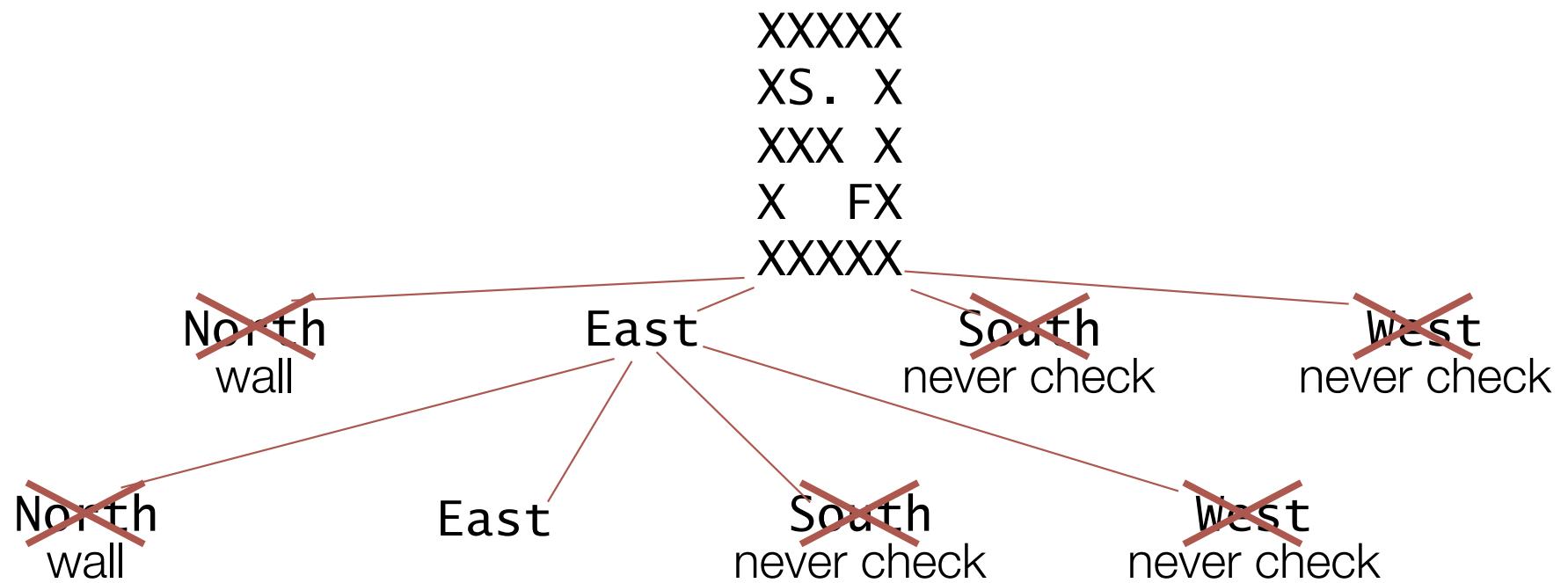
```
// north  
if (solveMazeRecursive(row-1,col,maze) == true) {  
    return true;  
}  
  
// east  
if (solveMazeRecursive(row,col+1,maze) == true) {  
    return true;  
}  
  
// south  
if (solveMazeRecursive(row+1,col,maze) == true) {  
    return true;  
}  
  
// west  
if (solveMazeRecursive(row,col-1,maze) == true) {  
    return true;  
}  
  
maze[row][col] = 'b';  
return false;  
}
```



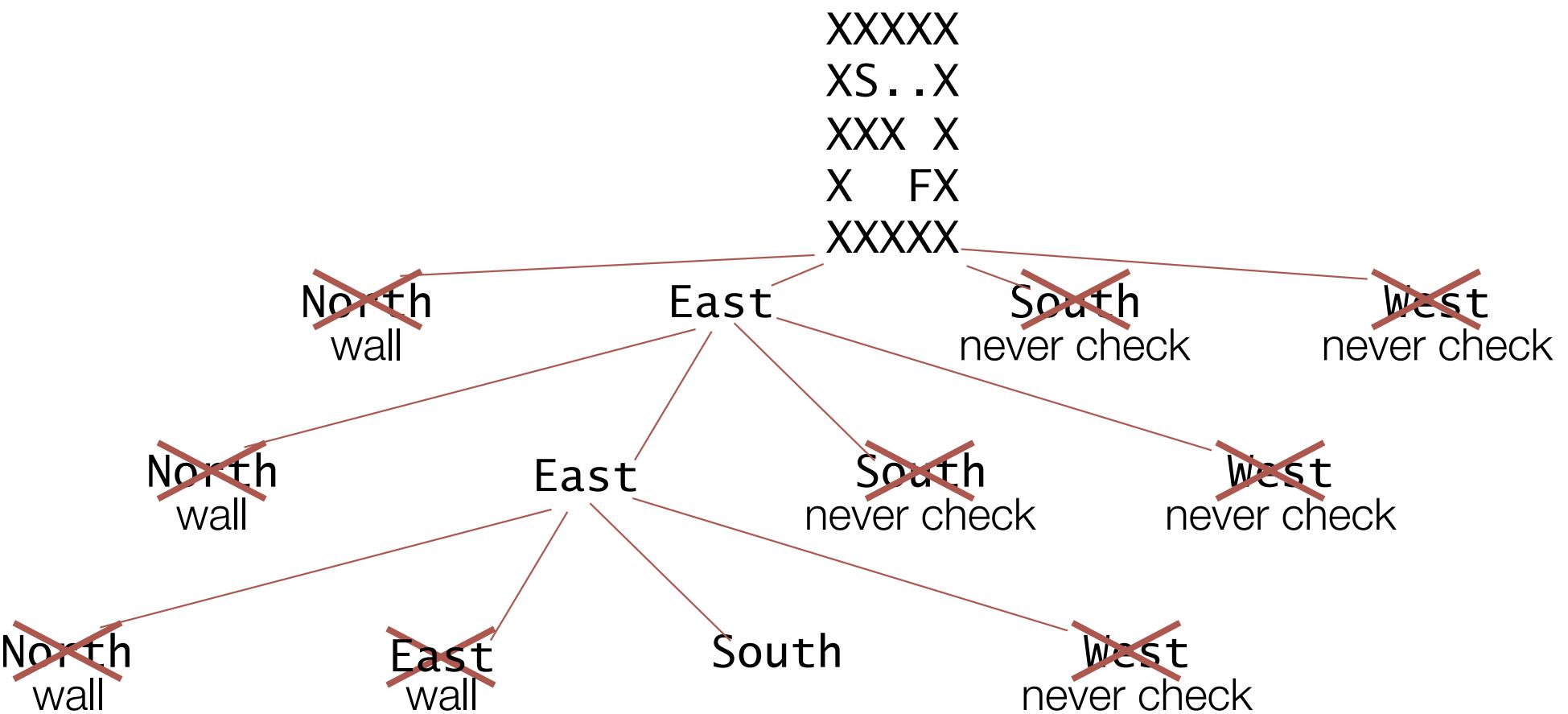
Maze Solving: A Decision Tree



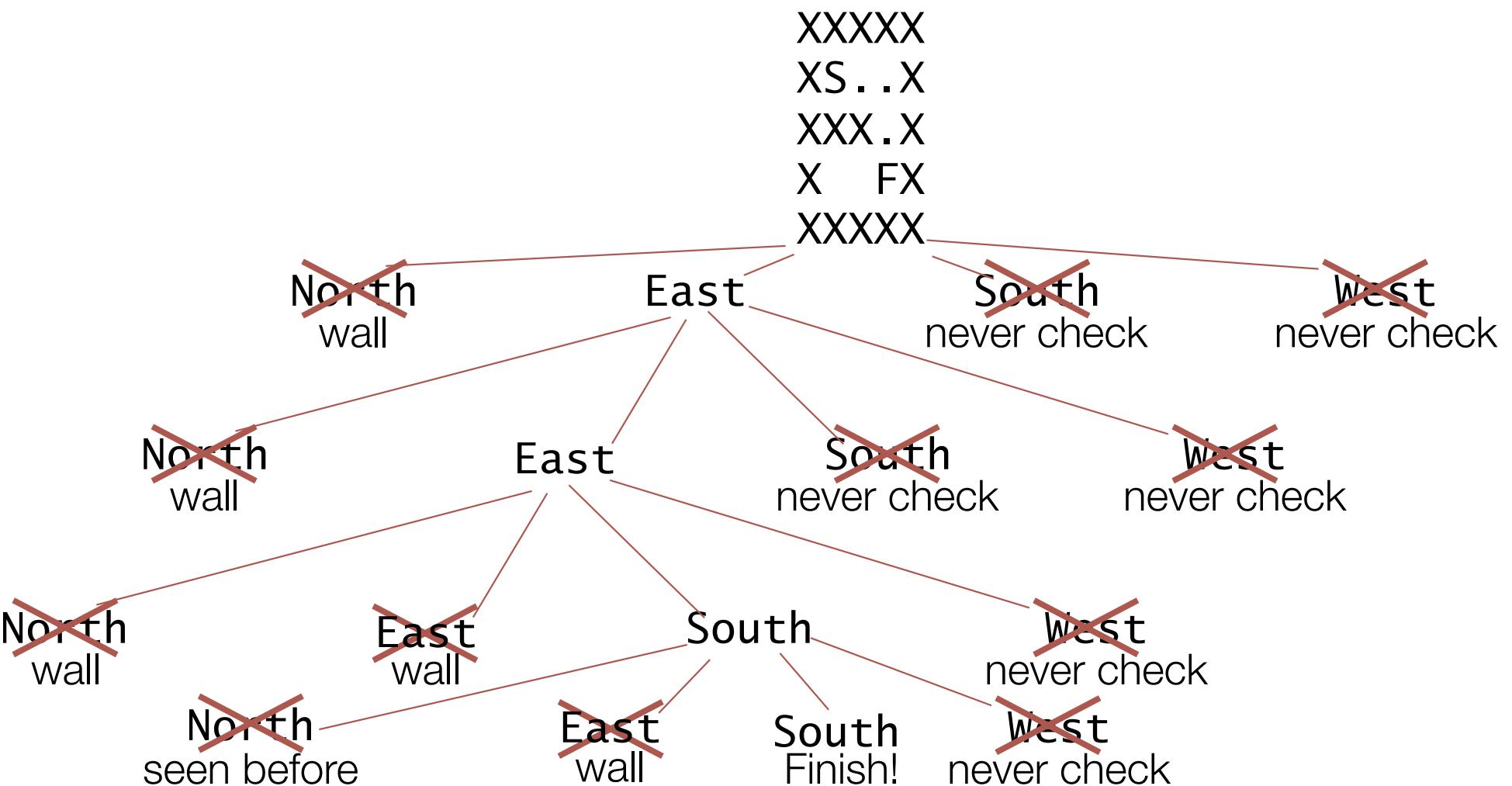
Maze Solving: A Decision Tree



Maze Solving: A Decision Tree

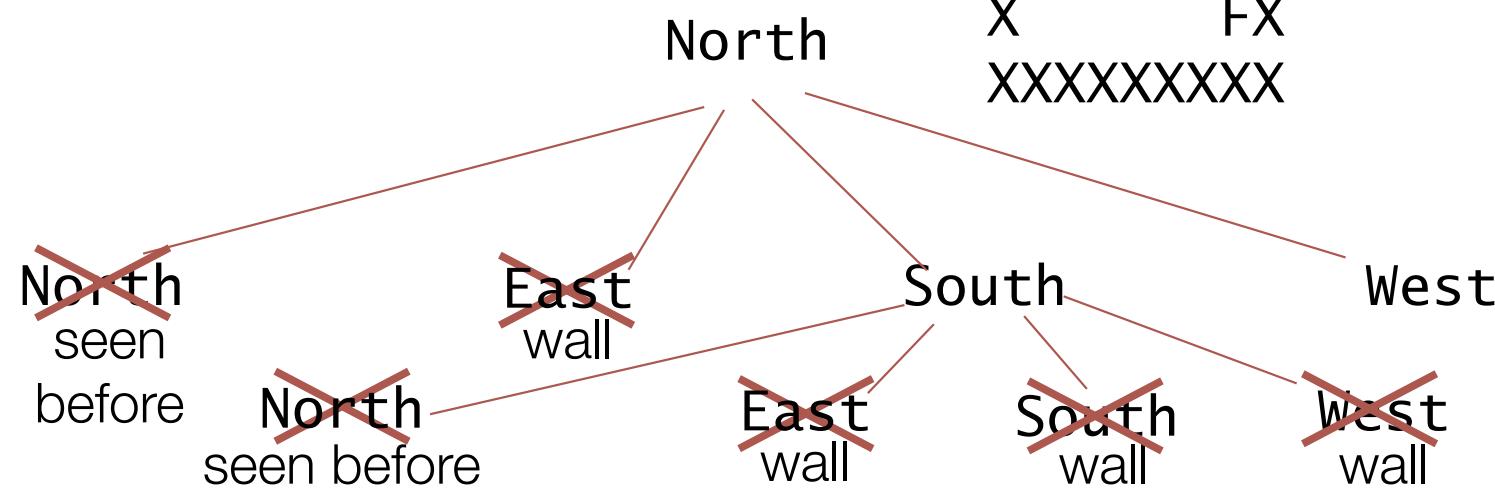


Maze Solving: A Decision Tree



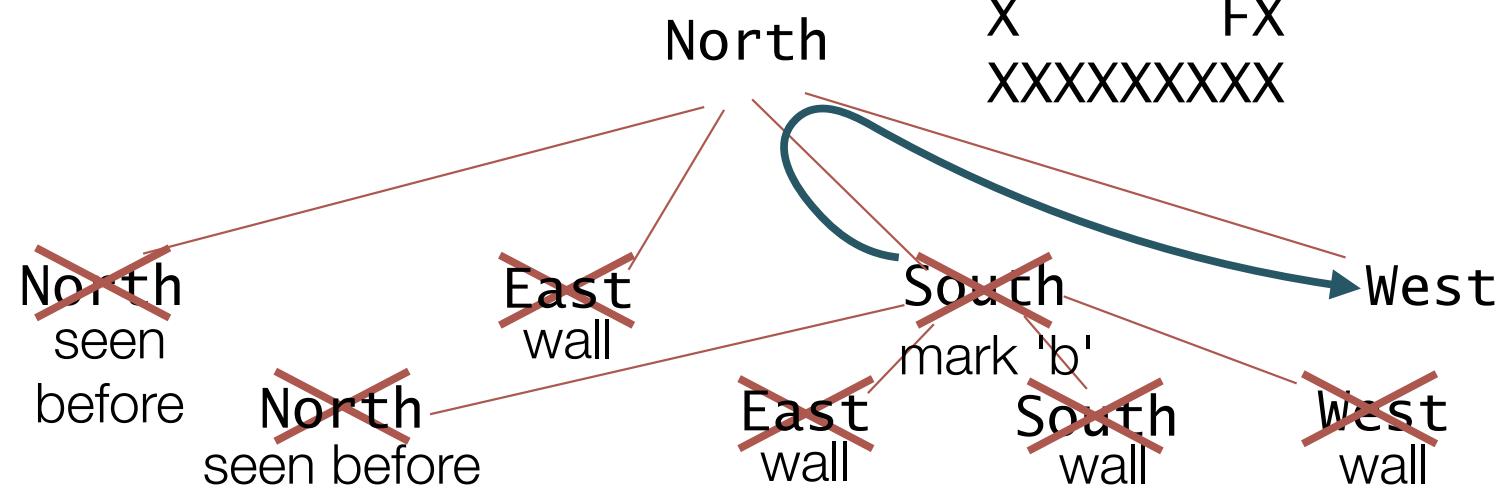
Maze Solving: A Decision Tree

Backtracking would involve traversing back up the tree. Because we have one shared grid / maze, we must "undo"!



Maze Solving: A Decision Tree

Backtracking would involve traversing back up the tree. Because we have one shared grid / maze, we must "undo"!

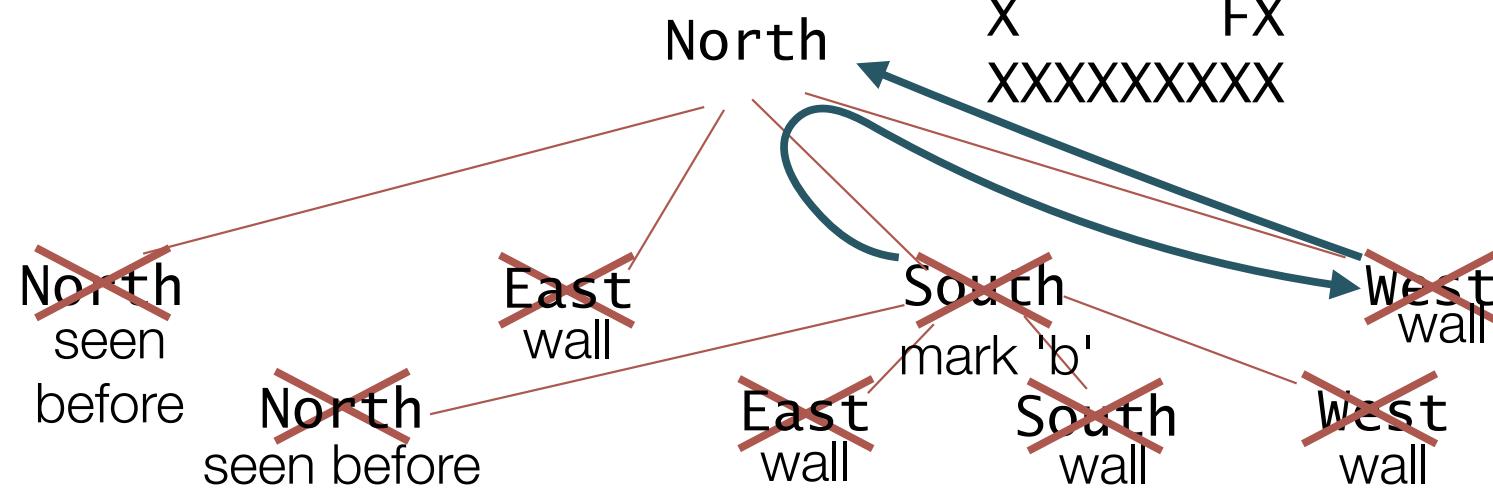


XXXXXXXXXX
XS..X...X
XXX.X.X.X
X X.XbX.X
X X.XXX.X
X X.....X
X XXXXX X
X FX
XXXXXXXXXX



Maze Solving: A Decision Tree

Backtracking would involve traversing back up the tree. Because we have one shared grid / maze, we must "undo"!



Recap

- Some algorithms just don't make sense iteratively, so we must use recursion (or stacks...but that's for another day!)
 - Permutations are a good example. When you solve them, you get a very nice tree representation.
- Recursive Backtracking can be used to search a system (e.g., a maze) -- you must either mark where you have been, or "undo" where you have been in order to progress in the algorithm.



References and Advanced Reading

- **References:**

- Understanding permutations: <http://stackoverflow.com/questions/7537791/understanding-recursion-to-generate-permutations>
- Maze algorithms: https://en.wikipedia.org/wiki/Maze_solving_algorithm

- **Advanced Reading:**

- Exhaustive recursive backtracking: <https://see.stanford.edu/materials/icspacs106b/h19-recbacktrackexamples.pdf>
- Backtracking: <https://en.wikipedia.org/wiki/Backtracking>



Extra Slides



All steps for Maze starting on slide 22

row=1 and col=1, Marking with period (.)

Trying north, row=0 and col=1, Hit wall! Back at row=1 and col=1,

Trying east, row=1 and col=2, Marking with period (.)

Trying north, row=0 and col=2, Hit wall! Back at row=1 and col=2,

Trying east, row=1 and col=3, Marking with period (.)

Trying north, row=0 and col=3, Hit wall! Back at row=1 and col=3,

Trying east, row=1 and col=4, Hit wall! Back at row=1 and col=3,

Trying south, row=2 and col=3, Marking with period (.)

Trying north, row=1 and col=3, We came from here! Back at row=2 and col=3,

Trying east, row=2 and col=4, Hit wall! Back at row=2 and col=3,

Trying south, row=3 and col=3, Marking with period (.)

Trying north, row=2 and col=3, We came from here! Back at row=3 and col=3,

Trying east, row=3 and col=4, Hit wall! Back at row=3 and col=3,

Trying south, row=4 and col=3, Marking with period (.)

Trying north, row=3 and col=3, We came from here! Back at row=4 and col=3,

Trying east, row=4 and col=4, Hit wall! Back at row=4 and col=3,

Trying south, row=5 and col=3, Marking with period (.)

Trying north, row=4 and col=3, We came from here! Back at row=5 and col=3,

Trying east, row=5 and col=4, Marking with period (.)

Trying north, row=4 and col=4, Hit wall! Back at row=5 and col=4,

Trying east, row=5 and col=5, Marking with period (.)

Trying north, row=4 and col=5, Hit wall! Back at row=5 and col=5,

Trying east, row=5 and col=6, Marking with period (.)

Trying north, row=4 and col=6, Hit wall! Back at row=5 and col=6,

Trying east, row=5 and col=7, Marking with period (.)

Trying north, row=4 and col=7, Marking with period (.)

Trying north, row=1 and col=7, Marking with period (.)

Trying north, row=0 and col=7, Hit wall! Back at row=1 and col=7,

Trying east, row=1 and col=8, Hit wall! Back at row=1 and col=7,

Trying south, row=2 and col=7, We came from here! Back at row=1 and col=7,

Trying west, row=1 and col=6, Marking with period (.)

Trying north, row=0 and col=6, Hit wall! Back at row=1 and col=6,

Trying east, row=1 and col=7, We came from here! Back at row=1 and col=6,

Trying south, row=2 and col=6, Hit wall! Back at row=1 and col=6,

Trying west, row=1 and col=5, Marking with period (.)

Trying north, row=0 and col=5, Hit wall! Back at row=1 and col=5,

Trying east, row=1 and col=6, We came from here! Back at row=1 and col=5,

Trying south, row=2 and col=5, Marking with period (.)

Trying north, row=1 and col=5, We came from here! Back at row=2 and col=5,

Trying east, row=2 and col=6, Hit wall! Back at row=2 and col=5,

Trying south, row=3 and col=5, Marking with period (.)

Trying north, row=2 and col=5, We came from here! Back at row=3 and col=5,

Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,

Trying south, row=4 and col=5, Hit wall! Back at row=3 and col=5,

Trying west, row=3 and col=4, Hit wall! Back at row=3 and col=5,

Failed. Marking bad path with b. Back at row=2 and col=5,

Trying west, row=2 and col=4, Hit wall! Back at row=2 and col=5,

Failed. Marking bad path with b. Back at row=1 and col=5,

Trying east, row=3 and col=6, Hit wall! Back at row=3 and col=5,

Failed. Marking bad path with b. Back at row=1 and col=6,

Trying east, row=2 and col=5, Hit wall! Back at row=1 and col=6,

Failed. Marking bad path with b. Back at row=1 and col=7,

Trying east, row=3 and col=7, Hit wall! Back at row=2 and col=7,

Failed. Marking bad path with b. Back at row=1 and col=7,

Trying east, row=2 and col=8, Hit wall! Back at row=2 and col=7,

Failed. Marking bad path with b. Back at row=1 and col=7,

Trying east, row=3 and col=8, Hit wall! Back at row=3 and col=7,

Failed. Marking bad path with b. Back at row=2 and col=7,

Trying east, row=2 and col=9, Hit wall! Back at row=3 and col=7,

Failed. Marking bad path with b. Back at row=1 and col=7,

Trying east, row=3 and col=9, Hit wall! Back at row=4 and col=7,

Failed. Marking bad path with b. Back at row=3 and col=7,

Trying east, row=4 and col=8, Hit wall! Back at row=4 and col=7,

Failed. Marking bad path with b. Back at row=3 and col=7,

Trying east, row=5 and col=8, Hit wall! Back at row=5 and col=7,

Failed. Marking bad path with b. Back at row=4 and col=7,

Trying east, row=6 and col=8, Hit wall! Back at row=6 and col=7,

Failed. Marking bad path with b. Back at row=5 and col=7,

Trying east, row=7 and col=8, Found the Finish!

