CSE143X Lecture Questions for Wednesday, 11/18/20

Time (e.g., 12:45)	Question	Answer
	Can I understand dead end in backtracking like the base case in recursion? I see. Can I say backtracking is a kind of recursion? thx.	It's similar in that the backtracking ends at a dead-end. No, backtracking is a technique that can be implemented recursively or iteratively.
32:15	For 3 digits, the decision tree has 4 levels. So for 4 queens, the tree will have 5 levels? Got it.	It depends on how you count levels. You are seeing four levels in the 3-digit case because at the top we have made no choices. If you include that top level before choices have been made, then yes, 4-queens would lead to 5 levels.
1:00	Can we also apply for the TA positions in later quarters? Thank you.	Yes.
9:38	When you say search space, do you mean like a blank _ u can fill any digit? I see thanks!	Yes, you can describe the digit search space that way (a blank to be filled in with a digit).
41:08	Why after the first column, I call the second, and then the third… column in the recursion. Intuitively, I think the first search has 8*8 choices, then 7*7, then 6*6...like that, but not column by column or row by row. Kind of. To meet the rule of unthreatening each other, it's 8*7*6… ?	We know that any solution to 8 queens has one queen in each of the 8 columns. That observation allows us to greatly reduce the number of choices to consider. We can pick the 8 queens column by column, which means the total number of possibilities is 8*8*8… (eight times, so 8^8).

	Will backtracking cover ALL cases? For example you find a solution in row2col1, you print it, but do you still back up to the top level and carry on finding other solutions?
	
	I see. I've encountered this before for sudoku solving, but that one only did one solution. I'm guessing we toggle a flag or something to stop once we find a solution?
	
	Ah, okay. Then the function itself becomes the conditional for whether we continue or not?
	
	If I do something like
	if(!explore(...)){
	}
	Does it run the method in the conditional statement?
	
	Interesting. Thanks! This is very cool.
		You can decide whether you want your backtracking solution to find all solutions or just one solution. The code I wrote for 8 queens finds all solutions.
		
		Typically you would return a boolean that indicates whether or not you found a solution and you would stop exploring once you had the value true returned.
		
		Yes, the return value of the method determines if you continue.
		
		You would tend to replace the call on explore with:
		`if (explore(b, col + 1)) {`
		`  return true;`
		`}`

47	I am still kind of confused with the choose, explore and unchoose flow. I understood after the Queens game visualization was showed such that it has to undo the previous choice when it is unable to place a Queen in that current column safely, but the code itself is telling me that it will be undo-ing every Queen placement before moving on to another Queen placement in the next column even though it has been placed safely? Why does it have to remove it each time? Oh! So the code is kind of placing each queen at each row, then removing, then trying to place another one at another row? Sorry, I mistakenly thought that you were removing previous safely placed queens from previous columns. Does this sound right? Awesome, thanks!	Yes, it always removes the queen after fully exploring the options with the queen placed there. If you don't remove, then you leave behind a queen that will threaten any other queens you would want to place in that column. I think you're understanding it now. Think about the first few calls. I ask a robot to explore column 1 and it places a queen in row 1. That one is safe and stays there. Then I ask a robot to explore column 2. It finds that rows 1 and 2 are dead ends, so it skips those. Then it places a queen in row 3 and recursively explores. After that is done, it removes the queen from row 3 so that the next time through the loop it will place a queen in row 4 of column 2. As you said, the queen in column 1 is not removed (not yet).

46	Suppose after every 1st recursive call on explore we have found the correct solution, then we will never call remove right? But if we've found the safe positions without ever having to change our previous decision, why will the execution reach remove? Or will that be once we place all our queens AND print them? The printing will happen backwards correct?	No, we end up calling remove when the recursion finishes. We always remove the queens we place. The recursion explores and explores, printing every solution it finds, but it will eventually run out of things to explore. So it will eventually get back to that part of the code and will remove the queen it had placed before exploring. I'm not sure what you mean by backwards. It prints as soon as it finds a solution.
32:04	Since there's 8 rows, wouldn't there be 4 more possible rows to put the queen? Cause you said you can put a queen in each of the first four columns, but you also put them in the 4th-8th column	I don't know what you mean by 4 more possible rows. There might have been some confusion about when I was doing the 4-queens problem versus the 8-queens problem. When you do 8-queens, there are 8 rows where you can place a queen in each column.
	Are dead ends where the value of the backtracking is not wanted? Or the point wanted. > solve (3, 2) is (3, 2) the dead end?	You reach a dead end when you find that a combination you are considering is not a solution.
	Why do we find and compute dead end values? Wouldn't it be better to avoid them? But if we know something is going to be a dead end, wouldn't it be better to avoid it completely instead of exploring it? I see, thank you.	It's more that we encounter dead ends as we explore. And when we encounter them, we stop exploring that part of the decision tree. Sometimes you avoid a recursive call because you know it will lead to a dead end. That's what we do in the 8-queens. Other times it can be more convenient to just let the call occur and then notice it's a dead end, as with the section problem with N, NE, and E.

	Boy it would be fun to do the same on Sudoku, do you think it would take a whole bunch of additional programming to get there?	Sudoku lends itself nicely to backtracking. I have the TAs go over that in the 143 sections, but we don't have time to fit it in for 143x. If you want to check it out, here is the solution: https://courses.cs.washington.edu/courses/cse143x/20au/lectures/sudoku.zip Run Sudoku.java and give it a file like sudoku1.txt. The animation is cool.