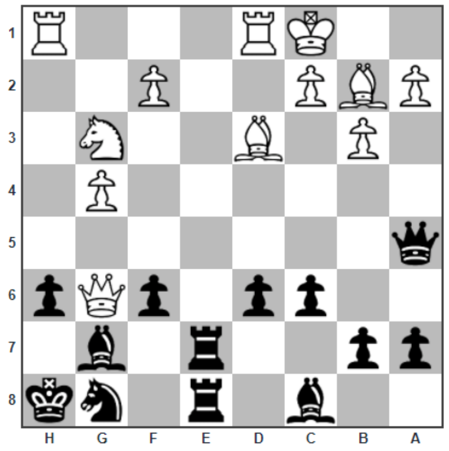
The n-Queens Problem

Suppose you have an $n \times n$ chessboard and n queens. That is, you have one queen for each row of the chessboard. The queen is a chess piece that threatens any piece on the same row, column, or diagonal she is on, unless another piece is in between the two. Your goal is to place all n queens on the board so that no two threaten one another. For example, in the diagram below, the queen on g6 in the following diagram threatens four enemy pieces (although there is a better move if it is her turn, that is not the point of this exercise).



Question 1. Here are two placements of $n = 4$ queens; how would you judge the quality of these solutions?

		Q	
	Q		
			Q
Q			

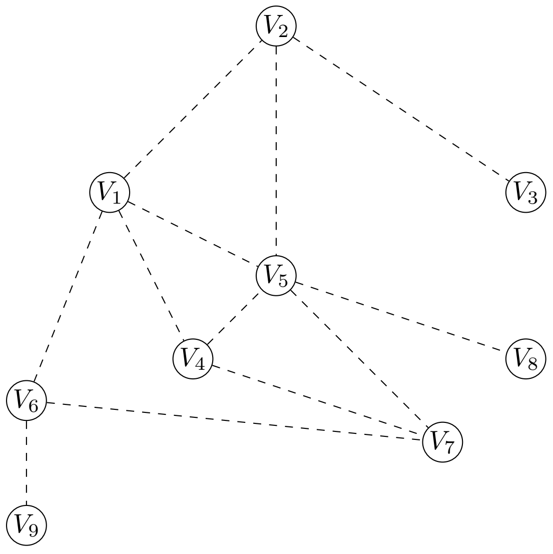
	Q		
			Q
Q			
		Q	

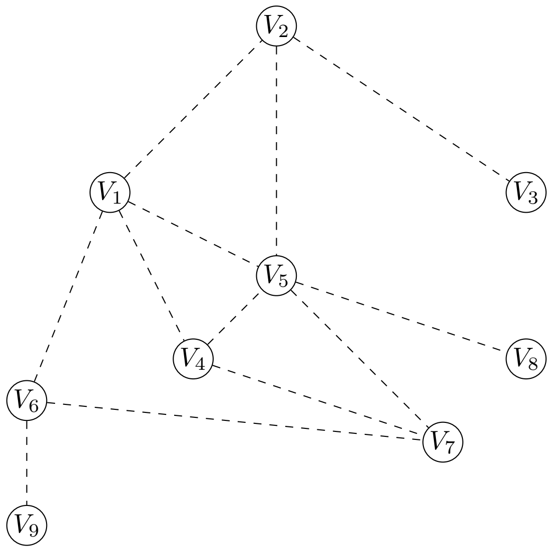
Question 2. How would we find such a solution methodically?

Graph Search

A *graph* (in this context) is a collection of vertices and edges. The former typically denote objects or concepts, while the latter typically indicate relations between concepts. There is more to a graph than that, but this will suffice for today.

There are many ways to *explore* a graph. We often do not have a graph drawn on a piece of paper, but rather are examining it in computer memory. The first two *graph search algorithms* we will discuss are *breadth-first search (BFS)* and *depth-first search (DFS).*





Breadth-First Search(s: starting vertex)

```
Set all $v$ undiscovered
Mark $s$ as discovered
L[0] $\leftarrow \{s\}$
$i \leftarrow 0$
**while** L[$i$] is not empty **do**
  Make L[$i+1$] as empty list
  **for all** vertices $u \in$ L[i] **do**
    **for all** edges $(u, v)$ **do**
      **if** $v$ not discovered **then**
        Mark $v$ as discovered
        Add $v$ to list L[$i+1$]
  Increase value of $i$ by one
```

Depth-First Search from a vertex u:

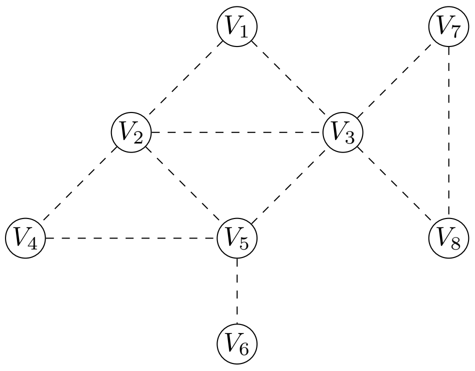
- Before running, set all undiscovered
- Start the search at u = start

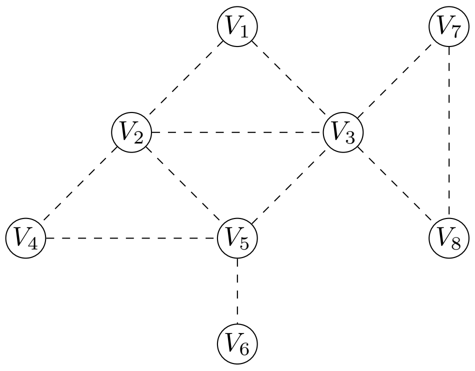
DFS(u: current vertex)

```
Mark $u$ as "discovered"
**for** each edge $(u, v)$ **do**
  **if** $v$ is not marked "discovered" **then**
    DFS from $v$
    When that is done, continue here.
```

Question 3. For the graph above, what is the order of vertices marked as discovered ("visited") by a Breadth-First Search? By a Depth-First-Search? What is the resulting "Breadth-First Search Tree?" The resulting "Depth-First Search Tree?"

Question 4. Repeat the previous question for the following graph.





Sliding Tile Puzzles

Consider the following puzzle; it has the first eight positive integers as "squares" and an empty square. You can slide any square adjacent to the empty square into the empty square to adjust it. Your goal is to get the puzzle to look like the one on the right hand side, when given it looking like the left hand side.

■	1	3
4	2	5
7	8	6

1	2	3
4	5	6
7	8	■

You can achieve this for that particular puzzle by sliding 1 left, then 2 up, then 5 left, then 6 up.

There are variations of this puzzle for different integers of the form $k^2 - 1$, where k is a positive integer. The 8-puzzle is $k = 3$, for example.

Question 5. Suppose we want to write a program to solve this; how do we use BFS and/or DFS to do so? To answer this question, we first need to define a graph, which will involve defining vertices and edges. We then need to decide where we want to start our search, which algorithm type to use (BFS or DFS), and if we want to modify our algorithm.

Question 6. How do BFS and DFS look for the graph from the Sliding Tile Puzzle?

▨	1	3
4	2	5
7	8	6

▨	1	3
4	2	5
7	8	6

Modeling Other Puzzles as Graphs

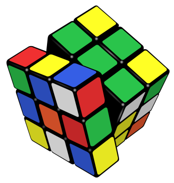
Question 7. We have two containers: one has a capacity of three gallons of water, the other five gallons. Both are initially empty, although we have access to a large water fountain. We can take a non-full container and fill it completely up with the water, or we can completely empty a container (with water in it) into the fountain, or we can pour the contents of one container into the other until either the first is empty or the second is full. Our goal is to find a sequence of actions we can take to end up with exactly four gallons of water in one container (and none in the other container). How can you use a graph modeling to solve this problem?

Question 8. A number maze is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board. Your goal is to find the minimum number of moves required to solve a given number maze, or to correctly report that the maze has no solution. How can you use a graph modeling to solve this problem?

For example, in the following maze, we can solve this with eight moves: right, down, left, right, up, left, right, down.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	

Question 9. How would you use graph modeling to find a way to solve a Rubik's Cube?



Additional Reading

There is a wealth of information about using search techniques to solve problems, not just puzzles. The textbook of Russell and Norvig has two chapters about this ; today's class has just barely scratched the surface.

I think the timing for today's class will be such that we might not get to the following topics:

- *Iterative Deepening Depth-First Search*, in which we modify DFS to give us the benefits of BFS without some of its drawbacks, at a small price.
- *Bidirectional Breadth-First Search*, in which we try to speed up BFS by searching both from the start and end positions.
- *Hill Climbing*, a search algorithm that has been described as "trying to find the top of Mount Everest in a thick fog while suffering from amnesia." That description is in the Russell and Norvig textbook.

A large topic we will not have the chance to discuss is heuristic search. To understand this, you will need to have done enough programming to understand what a *priority queue* is; once you have that, I would encourage you to read about an algorithm called A* ("A star"), as well as admissible heuristics and pattern databases for it, and an algorithm called recursive best-first search.

In nine lectures, you've seen a wide overview of techniques in Machine Learning and Artificial Intelligence. To briefly illustrate what a wide world there is available to you, let me tell you a little about classes available to you if you were an undergraduate Computer Science major at UCI (and maybe some of you will be!).

There are two ten-week "survey" (overview) courses, one diving into several major topics in Artificial Intelligence and the other doing the same for Machine Learning and Data Mining.

After you have taken those, there is a ten-week project course, constructing and evaluation a working AI system.

There are also four additional in-depth topic courses about neural networks and deep learning; the social impact of artificial intelligence; the uses of probability in Computer Science; the use of probabilistic and deterministic graphical models (we saw Bayesian and Markov networks as examples of this last one).

There is also a course about the uses of Artificial Intelligence in the fields of Biology and Medicine.

Like I said, there is a wide world of Artificial Intelligence and Machine Learning available for you to explore if you wish; you have learned the fundamentals and been introduced to it, but we have just scratched the surface.