Provide a concise answer to each of the following questions. Your answers should **NOT** include code. It should be an English explanation, and each question can be answered in a few sentences. If your answer to one of the questions includes the use of a set, map, or symbol table, include an explanation of why/how you are using it, but do **NOT** explain what the data structure itself does or how it works. For example, if you use a BinarySearchST, do not explain to me how binary search works, how insertion works, etc. Instead, explain, what are the keys, what are the values, and why/how you using the symbol table in your solution. Each question is worth 5 points.

- Breadth-first search is done on a graph. What is the graph being searched in your solution? What are the vertices of the graph and when is there edge between two vertices?

Upon construction of the Solver() Object, I create a solution-graph called edgeTo starting from the Solved board where the previous edge it NULL. The vertices of the graph are every possible board position, and the edges connecting the boards are moves to get between those boards.

- During breadth first search, you need to loop over all neighbors of a given vertex. Explain how your code does this. How does your code know who the neighbors of a given vertex are?

My implementation of solver stores a string of 20 characters called ***moves*** of all the possible moves a single board has available to compute. I create a loop, starting from index 0 of the ***moves*** string to Index N-1, to move (transform) a copy of the current board by the character at index N.

- Breadth-first search requires that you mark vertices as you encounter them to keep track of the vertices you have already seen. The book's code uses an array to do this. How did you keep track of marked vertices? If you used a data structure, don't just mentioned the data structure. Explain the invariant. (What is the relationship between the data structure and the marked vertices? How do you use the data structure to mark a vertex? How do you use the data structure to check if a vertex is marked?)

I use a HashMap named edgeTo with a key of type Board, and value of type Board to keep track of the vertices I have already seen. I took advantage of HashMap's containsKey() method to check if a board was already inserted as a key to the edgeTo hashmap. I do not explicitly state that a board has been visited but rather check if the board in question has already been inserted as a key to the map, which implies that the board has already been visited; thereby not adding it to the map again and continuing the Breadth-first Search.

- To be able to recover the solution path, breadth-first search requires that for each vertex, you keep track of what vertex you came from when you first encountered it (edgeTo array). How do you keep track of that in your solution? Again, if you use a data structure, make sure to explain what is stored in it and how it is updated/used.

I use a HashMap named edgeTo with a key of type Board, and value of type Board to keep track of vertices I came from when I first encountered it. If my edgeTo hashmap does not contain the board as a key, I put it in the hashmap with it's value as the board that it came from when I first encountered it.

- Assuming your breadth-first search has already been performed, how is the solution itself recovered? Again, avoiding explaining line by line. Instead, explain what has been stored in the various fields/variables by the BFS, and how that information is used to construct a solution. (Note that there could be some overlap between this question and previous question.)

Using my edgeTo hashMap, I create a loop that starts with the current Board in question and Iterate through every previous Board (calculating the move on every iteration), until it reaches the solution Board (which has a value of NULL). More specifically, my Board class contains a getMove() method that takes a Board object as a parameter and iterates through every move until it finds the move it took to get from the previous board (passed in) to the current Board(***this*** or caller) and returns that character. On every iteration, I append the character returned by getMove() to a StringBuilder object called ***answer***. Once the Solution has been reached (denoted by NULL value), the ***answer*** is converted to a string object that is converted to a character array and returned.

If your solution does only a single breadth-first search at the beginning, regardless of the number of times solve is called, how did you do it? Focus on how your search and solution recovery differ from the more straightforward BFS described in the book and in class.

My solution does only a single breadth-first search at the beginning, regardless of the number of times solve is called by generating a graph with every possible combination of a board that is rooted at the solution board, and only uses solve to search and construct the path to the solution. During construction of the Solver object, I start at the solution Board and find every possible Board variation and put it in a HashMap called edgeTo. When solve is called, I construct a Board object from the 2D Boolean array that was passed in, and I find it in within my edgeTo Hashmap, and then I reconstruct the path to the solution. My search and solution recovery differs from the more straightforward BFS described in the book and in class because you would typically start constructing the graph from the passed in Board position, work towards the solution, and then construct the path once it's found. I did the opposite because the cost of constructing the graph and the path per-basis with a finite set of data was significantly slower than the initial cost of constructing the entire graph from the solution at construction and searching for the path when creating the board.