This code defines a class "B" that contains a method "totalNQueens" which takes an input "n" and returns the number of possible ways to place "n" queens on an "n x n" chessboard such that no two queens threaten each other. The method uses a backtracking algorithm to solve the problem.

The "count" variable is used to keep track of the number of solutions found, which is initially set to zero. The "backtrack" method is a private helper function that is called to find all the possible solutions. The function takes 5 parameters "row", "ld", "col", "rd" and "n".

* "row" is an integer variable that represents the rows occupied by the queens.
* "ld" is an integer variable that represents the left diagonals occupied by the queens.
* "col" is an integer variable that represents the columns occupied by the queens.
* "rd" is an integer variable that represents the right diagonals occupied by the queens.

The base case of the recursion is when the number of occupied rows is equal to (1 << n) - 1, which is the maximum number of rows that can be occupied on an n x n chessboard. At this point, the count variable is incremented by 1.

The next step is to find the next available position on the chessboard where a queen can be placed. This is done by using bit manipulation to check the availability of positions. The expression "~(row | ld | col | rd)) & ((1 << n) - 1)" is used to find the next available position.

"~(row | ld | col | rd))" returns a number that has all bits set to 1 except the bits that are set to 1 in row, ld, col and rd. "& ((1 << n) - 1)" is used to mask the bits that are outside the range of the chessboard.

After finding the next available position, the function recursively calls itself with updated values of "row", "ld", "col", and "rd" variables. The values of these variables are updated by ORing with "p", which is the next available position. Then the function continues to find the next available position and recursively calls itself until all possible solutions are found.

Finally, the totalNQueens method returns the total number of solutions found which is stored in the count variable.

Please explain to me below code like I am 12 int pos = (~(row | ld | col | rd)) & ((1 << n) - 1);

Okay, so this code is trying to find a place on the chessboard where a queen can be placed. It does this by looking at the rows, columns, and diagonals that are already occupied by queens (represented by the variables "row", "ld", "col", and "rd").

First, it uses the "~" symbol to flip all the bits in the number made by adding together the occupied rows, columns and diagonals. This gives us a number with 1's in all the spots that are not occupied by a queen, and 0's in the spots that are occupied.

Then, it uses "&" symbol to compare this new number with a number that represents all the spots on the chessboard. This is done by using (1 << n) - 1 which means to shift 1 by n bits to left. This gives us a number with 1's in all the spots on the chessboard, and 0's everywhere else.

By using the "&" symbol, we are able to find the spots on the chessboard that are not occupied by a queen and also inside the chessboard range, which is stored in the variable "pos".

So in simple words, this line of code is **trying to find a blank spot on the chessboard where a queen can be placed** and store that position in the variable "pos".

What is bits and shift in below code?

Explain it to me like I am 12

"Bits" are like tiny switches that can be turned on or off, and "shift" is a way to move these bits around. In this code, the bits are being used to keep track of which rows and columns have queens on them, so that the program doesn't accidentally put two queens in the same spot.

The code is trying to find all the ways to put n queens on an n by n chess board, so that no two queens are attacking each other. And it's using bits and shift to keep track of which rows, columns and diagonals have queens on them.

The int pos is a number that has all the bits turned off, except for the bits that represent the rows,columns and diagonals that don't have queens. And the while loop is used to go through each of those bits, one at a time, and see if it's a safe place to put a queen, and then it will move on to the next bit if it's not a safe place. And if it finds all the safe places for queens, it will increment the count variable.

What do you mean by ""shift" is a way to move these bits around" ?

When we "shift" bits, it means we are moving the positions of the bits to the left or to the right.

For example, let's say we have a number **1101** (in binary) and we want to shift the bits to left by 2 positions, it will become **110100**.

If we shift left, it's like moving all the bits to the left, and adding 0s to the right.

Similarly, if we shift right, it's like moving all the bits to the right, and adding 0s to the left.

In the code you provided, the ">>" and "<<" operators are used to shift the bits to the right and left. The ">>" operator is used to shift the bits to the right and the "<<" operator is used to shift the bits to the left.

These operators are used to move the bits around so that the program can keep track of which rows, columns, and diagonals have queens on them.

So in conclusion bit is use to increase the length for program to check?

In this code, bits are being used to represent the rows, columns, and diagonals on the chess board. Each bit in the number represents a single row, column or diagonal. By using bits, the program can keep track of which rows, columns, and diagonals have queens on them in a very efficient way, as it only need to check a single number instead of an entire chess board.

In short, bits are used to represent and keep track of the state of the chess board in a compact and efficient way. It allows the program to check for safe places for the queens more quickly by using bitwise operations.