1812. Determine Color of a Chessboard Square

Easy Math String Leetcode Link

Problem Description

In the given LeetCode problem, you are presented with a string that represents the coordinates of a square on a standard chessboard. A chessboard is an 8×8 grid, with squares that alternate in color between white and black. The `coordinates` string consists of a letter and a number (for example, "a1" or "h8"), where the letter represents the column (ranging from 'a' to 'h') and the number represents the row (ranging from 1 to 8) on the chessboard.

The task is to determine whether a given square, based on these coordinates, is white or black. You are required to return `true` if the square is white and `false` if it is black. The challenge assumes that the coordinates provided will always be valid and that they will follow the classic chessboard layout where 'a1' is a black square, and 'a2' is a white square, and so on.

Intuition

The pattern on the chessboard is such that every consecutive square along a row or column alternates in color. Observing the pattern, we can see that moving from one square to the next -- in any direction -- always changes the color. That is, if we take a step horizontally, vertically, or diagonally from any square, the color of the square will change from white to black or black to white.

Because of this pattern, we know that if a square is white, the square directly adjacent to it (either to the right, left, above, or below) will be black, and vice versa. In terms of the provided coordinates, it implies that for every step we take in either the alphabetical direction (from 'a' to 'h') or the numerical direction (from '1' to '8'), the sum of ordinal values of the characters would change in terms of evenness (whether the sum is even or odd).

By convention, let's consider 'a1' (which is a black square) as our reference point. Since the letter 'a' and number '1' consecutively alternate the colors across the board from this starting point, we can conclude that when the sum of the ASCII values of the letter and number of any coordinate is even, we land on a square of the same color as 'a1', which is black. Conversely, when the sum is odd, we land on a white square.

Hence, the simple check `(ord(coordinates[0]) + ord(coordinates[1])) % 2 == 1` in the solution uses the ordinal values (ASCII) of the characters in the coordinate and checks whether the sum is odd (returns `true`) or even (`false`). If it's odd, the square is white; if it's even, the square is black.

Solution Approach

The implementation of the solution for this problem is quite straightforward and doesn't require complex algorithms or data structures. It relies on ASCII values and a simple mathematical trick that directly stems from the nature of a chessboard. Here's a step-by-step walkthrough of the solution approach:

- The solution involves examining the `coordinates` string, which contains two characters: a letter and a number.
- The ASCII value (or `ord` value in Python) of a character is a numeric representation of that character in the ASCII table. For instance, `ord('a')` will yield `97`, which is the ASCII value for the lowercase letter 'a'.
- The trick used in this solution is to sum the ASCII values of both characters in the `coordinates` string.

Why does adding ASCII values work?

- Since we know that a step horizontally or vertically on the chessboard changes the color of a square, we can think of the colors as being represented by "even" and "odd" categories.
- The mathematical basis for the solution is that the addition of two even numbers or two odd numbers yields an even number, while the addition of one even and one odd number yields an odd number.

Checking the parity (evenness or oddness):

- By taking the sum of the ASCII values of the letter and the number in the coordinates, we are effectively getting a number that represents the steps needed to reach that square from the origin ('a1').
- Using the modulo operator `%` with `2`, we can determine the parity of this sum. If the sum is even (`sum % 2 == 0`), the square shares the same color as 'a1', which is black. If the sum is odd (`sum % 2 == 1`), it means the square is white.

The final code snippet is just a one-liner:

```python
class Solution:
    def squareIsWhite(self, coordinates: str) -> bool:
        return (ord(coordinates[0]) + ord(coordinates[1])) % 2 == 1
```

- To fully understand why this works, let's consider an example: `coordinates = "c3"`.
- ASCII value of 'c' is 99 and '3' is 51.
- Adding them up gives us 150, which is an even number, meaning the square is black. However, since the problem asks us to return `true` if the square is white, we check if the sum is odd using `(99 + 51) % 2 == 1`, which yields `false`, so `c3` is indeed black.

This approach works for all valid squares on the chessboard and does not require additional validation since the problem states that the input will always be a valid chessboard square.

Example Walkthrough

Let's illustrate the solution approach with an example:

Suppose we are given the coordinates: `"f5"`.

Using the approach described above, let's go through the steps to determine the color of the square at these coordinates:

- First, we take the ASCII value for the letter `'f'` which is 102.
- Then, we take a look at the number `'5'`, in ASCII, numbers start with the value 48 for `'0'`, so `'5'` would have the ASCII value of 53.
- Now, let's sum these two values: 102 (for 'f') + 53 (for '5') = 155.
- Then, to find out the color of the square, we check the sum's parity. We use the modulo operation with 2, i.e., `155 % 2`.
- The result of `155 % 2` is 1, which is odd.
- Since the sum is odd, according to the solution approach, the square is white.
- Thus, our solution would return `true` for the coordinates `"f5"` indicating that the square is white.

In conclusion, by performing an operation that checks the parity of the sum of the ASCII values of the input coordinates, we can efficiently determine whether a given square on a chessboard is white or black. This method leverages the alternating pattern of colors on a chessboard and provides a simple, yet elegant solution to the problem.

Python Solution

```python
class Solution:
    def squareIsWhite(self, coordinates: str) -> bool:
        # Extract the column (letter) and row (number) from the coordinates
        column, row = coordinates[0], coordinates[1]

        # Convert the column (a-h) to its numerical representation (1-8) using `ord`
        # Subtract the ASCII value for 'a' to align 'a' with 1, 'b' with 2, etc.
        column_number = ord(column) - ord('a') + 1

        # Convert the row (1-8) from str to int for numerical operations
        row_number = int(row)

        # If the sum of the column and row numbers is odd, the square is white
        # Use modulo operation to check for odd (sum % 2 == 1)
        # Use `return` to output the result of the check
        return (column_number + row_number) % 2 == 1

```

Java Solution

```java
class Solution {

    /**
     * Determines if a chessboard square is white based on its coordinates.
     * Chessboard squares are identified by a combination of a letter (column) and a number (row).
     * For example: "a1", "h3", etc.
     *
     * @param coordinates the chessboard coordinates in algebraic notation.
     * @return true if the square is white, false if the square is black.
     */
    public boolean squareIsWhite(String coordinates) {
        // The chessboard columns ('a' to 'h') have ASCII values from 97 to 104.
        // Even rows ('2', '4', '6', '8') and odd columns ('a', 'c', 'e', 'g') lead to white squares.
        // By adding the ASCII values of both the column and the row, we can determine the color of the square.
        // If the sum is odd, it's a white square; if it's even, it's a black square.

        // Extract the column character and row character from the input string.
        char columnChar = coordinates.charAt(0);
        char rowChar = coordinates.charAt(1);

        // Calculate the ASCII sum of the column and row characters.
        int sumAsciiValues = columnChar + rowChar;

        // The square is white if the sum of the ASCII values is an odd number.
        return sumAsciiValues % 2 == 1;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to determine if a chessboard square is white
    bool squareIsWhite(string coordinates) {
        // Extract the alphabetical and numerical parts of the coordinates
        char column = coordinates[0]; // 'a' to 'h'
        char row = coordinates[1];    // '1' to '8'

        // Calculate the color of the square using the ASCII values of the characters
        // If the sum of the ASCII values is odd, the square is white, otherwise, it's black.
        // Because 'a' and '1' represent a dark square, adding the ASCII values
        // for any given square will yield odd for white and even for black squares.
        bool isWhite = (column + row) % 2 != 0;

        // Return the result
        return isWhite;
    }
};

```

Typescript Solution

```typescript
// Function to determine if a chessboard square is white based on its coordinates
// The chessboard coordinates have an alphanumeric format, e.g., "a1", "b2", etc.
// Parameters:
// coordinates - a string representing the chessboard coordinates
// Returns:
// A boolean value indicating whether the square is white (true) or not (false)
function squareIsWhite(coordinates: string): boolean {
    // Calculate the sum of the ASCII values of the coordinate characters
    const charCodeSum: number = coordinates.charCodeAt(0) + coordinates.charCodeAt(1);

    // For the square to be white, the sum of the ASCII values of the ranks and files
    // should be odd since chessboard has alternating colors and starts with 'a1' as black.
    // Using bitwise AND ('&') to determine if the sum is odd (result will be 1).
    return (charCodeSum & 1) === 1;
}

```

Time and Space Complexity

The time complexity of the function `squareIsWhite` is `O(1)` because it consists of a constant number of operations: accessing the elements of the string `coordinates`, computing the ordinal values using `ord()`, addition, and modulus operation.

The space complexity of the function is also `O(1)` since it does not allocate any additional space that grows with the input size; it only uses a fixed amount of space for the input and constant space for the calculations.