JavaScript Wildcard Matching

Challenge

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'` where:

- `'?'` Matches any single character.
- `'*'` Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

1st Example

```
Input: s = 'aa', p = 'a'
Output: false
Explanation: 'a' does not match the entire string 'aa'.
```

2nd Example

```
Input: s = 'aa', p = '*'
Output: true
Explanation: '*' matches any sequence.
```

3rd Example

```
Input: s = 'cb', p = '?a'
Output: false
Explanation: '?' matches 'c', but the second letter is
             'a', which does not match 'b'.
```

Constraints

- `0 <= s.length, p.length <= 2000`
- `s` contains only lowercase English letters.
- `p` contains only lowercase English letters, `'?'` or `'*'`.

Solution

```
const isMatch = (s, p) => {
    const m  = s.length,
          n  = p.length,
          dp = Array.from({length: m + 1},
               () => new Array(n + 1).fill(false));

    dp[0][0] = true;

    for (let j = 1; j <= n; j++) {
        if (p[j - 1] === '*') {
            dp[0][j] = dp[0][j - 1];
        }
    }
```

Solution continues on next page...

```
    for (let i = 1; i <= m; i++) {
        for (let j = 1; j <= n; j++) {
            if (p[j - 1] === '*') {
                dp[i][j] = dp[i][j - 1] || dp[i - 1][j];
            } else if (s[i - 1] === p[j - 1] ||
                       p[j - 1] === '?') {
                dp[i][j] = dp[i - 1][j - 1];
            }
        }
    }

    return dp[m][n];
};
```

Explanation

I've defined a function called `isMatch` that checks if a given string `s` matches a given pattern `p` which may contain wildcard characters (`*` and `?`). The function returns true if the string matches the pattern and false otherwise.

Inside the function, the lengths of the string `s` and pattern `p` are calculated and stored in variables `m` and `n` respectively.

A two-dimensional array called `dp` is created using the `Array.from()` method. The length of the array is `m + 1`, and each element of the array is an array of length `n + 1`. Initially, all elements of the array are set to false.

The value of `dp[0][0]` is set to true because an empty string matches an empty pattern.

A loop is then used to iterate through each character of the pattern

`p`, starting from index `1`. If the current character is a wildcard character (`*`), the value of `dp[0][j]` is set to the value of `dp[0][j-1]`. This is because `*` can match zero or more characters.

Next, two nested loops are used to iterate through each character of the string `s` and each character of the pattern `p`, starting from index `1`.

If the current character of the pattern `p` is a wildcard character (`*`), the value of `dp[i][j]` is set to `dp[i][j-1]` OR `dp[i-1][j]`. This accounts for the cases where `*` can match zero or more characters.

If the current character of the pattern `p` is not a wildcard character and matches the current character of the string `s`, OR the current character of the pattern `p` is a question mark (`?`), the value of `dp[i][j]` is set to `dp[i-1][j-1]`. This handles the case where the current character of the pattern `p` can only match the current character of the string `s`.

Finally, the function returns the value of `dp[m][n]`, which indicates whether the entire string `s` matches the entire pattern `p`.

In summary, the `isMatch` function checks if a string matches a pattern with wildcard characters, using dynamic programming to track the matching possibilities.

Author: Trevor Morin