JavaScript Regular Expression Matching

Challenge

Given an input string `s` and a pattern `p`, implement regular expression matching with support for `'.'` and `'*'` where:

- `'.'` Matches any single character.
- `'*'` Matches zero or more of the preceding element.

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 = 'a*'
Output: true
Explanation: '*' means zero or more of the preceding
             element, 'a'. Therefore, by repeating 'a'
             once, it becomes 'aa'.
```

3rd Example

```
Input: s = 'ab', p = '.*'
Output: true
Explanation: '.*' means 'zero or more (*) of any
             character (.)'.
```

Constraints

- `1 <= s.length <= 20`
- `1 <= p.length <= 20`
- `s` contains only lowercase English letters.
- `p` contains only lowercase English letters, `'.'`, and `'*'`.
- It is guaranteed for each appearance of the character `'*'`, there will be a previous valid character to match.

Solution

```
const isMatch = (s, p) => {
       const recurse = (i, j) => {
              const matchesChar = i < s.length &&
                          (p[j] === '.' || s[i] === p[j]);

              if (i === s.length && j === p.length) {
                     return true;
              }
```

Solution continues on next page...

```
            if (j + 1 < p.length && p[j + 1] === '*') {
                return recurse(i, j + 2) ||
                    matchesChar &&
                    recurse(i + 1, j);
            } else {
                return matchesChar &&
                    recurse(i + 1, j + 1);
            }
        };

        return recurse(0, 0);
};
```

Explanation

I've written a function called `isMatch` that takes in two string parameters, `s` and `p`, and returns a boolean value. The function uses recursion to check if the string `s` matches the pattern `p` using certain rules.

Inside the function, there is a nested function called `recurse` that takes in two integer parameters, `i` and `j`, representing the indices of the current characters being compared in `s` and `p` respectively.

Within the `recurse` function, it checks if the current character in `s` matches the current character in `p` or if the current character in `p` is `.` (which can match any character in `s`). If there is a match, the function continues to compare the next characters in `s` and `p` using recursion.

If the current character in `p` is `*`, the function checks if the preceding character in `p` matches the current character in `s` (or if the preceding character is `.`). If there is a match, the function can either skip over the current character in `p` (which represents `*` and can match zero characters in `s`) or continue to compare the next character in `s` with the current character in `p` (which represents `*` and can match one or more characters in `s`).

If there is no match between the current characters in `s` and `p`, or if the end of both strings has been reached, the function returns `false`.

Finally, the `isMatch` function calls the `recurse` function with the starting indices of `0` for both `s` and `p` and returns the boolean value returned by the `recurse` function.

In summary, the `isMatch` function uses recursion to check if a string `s` matches a pattern `p` based on certain rules. It compares characters from `s` and `p` one by one, taking into account special characters like `.`, which can match any character, and `*`, which can match zero or more of the preceding character.

Author: Trevor Morin