LeetCode 10

https://leetcode.com/problems/regular-expression-matching/description/

Yifeng Zeng

Description

10. Regular Expression Matching

Implement regular expression matching with support for '.' and '*'.

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

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

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

Idea Report

（Similar to edit distance）We use boolean[][] f to represent the DP solution space, where f[i][j] means boolean representation to match for the first i characters of s and first j characters of p. For the given example "aab" and "c*a*b", we can have:

	0 ' '	1 c	2 *	3 a	4 *	5 b

0 ' '	T	F	T	F	T	F
1 a	F	F	F	T	T	F
2 a	F	F	F	F	T	F
3 b	F	F	F	F	F	T

The first row is the case when s is "", so as long as p has can match an empty string, the cell is true. The first column is the case when p is "", so as long as s is not "", the cell is false, these are the base cases we need to cover. For any f[i][j] we need to discuss the situations:

- If s.charAt(i - 1) == p.charAt(j - 1), then for f[i][j], we need to see f[i - 1][j - 1], e.g. in the case f[1][3] = f[0][2] = true.
- If p.charAt(j - 1) == '.', it is the same situation as the above one, we check f[i - 1][j - 1].
- If p.charAt(i - 1) == '*', there may be a few different cases:
  - We need to check the character before '*' in p (p[j - 2]) and see if it is the same with the character in s[i - 1]. If p[j - 2] != '.' && p[j - 2] != s[i - 1], that means we cannot use this '*', which means '*' = 0, so we need to check f[i][j - 2].
  - Otherwise, we are able to use '*', but we don't necessary have to, so there may be three different situations:
  - We do not use '*' at all, then we check f[i][j - 2], at f[2][4] we check f[2][2]
  - We use '*' as 1, so we check f[i][j - 1], at f[2][4] we check f[2][3]
  - We use '*' multiple times, we check f[i - 1][j], at f[2][4] we check f[1][4].
- Other cases f[i][j] are all false.

Code:

```
class Solution {
    public boolean isMatch(String s, String p) {
        int len1 = s.length();
        int len2 = p.length();
        boolean[][] f = new boolean[len1 + 1][len2 + 1];
        f[0][0] = true;
        for (int i = 1; i <= len2; i++) {
            f[0][i] = p.charAt(i - 1) == '*' && f[0][i - 2];
        }

        for (int i = 1; i <= len1; i++) {
            for (int j = 1; j <= len2; j++) {
                if (s.charAt(i - 1) == p.charAt(j - 1)
                    || p.charAt(j - 1) == '.') {
```

```
                    f[i][j] = f[i - 1][j - 1];
                } else if (p.charAt(j - 1) == '*') {
                    if (s.charAt(i - 1) != p.charAt(j - 2)
                        && p.charAt(j - 2) != '.') {
                        f[i][j] = f[i][j - 2];
                    } else {
                        f[i][j] = f[i - 1][j] || f[i][j - 1] || f[i][j - 2];
                    }
                }
            }
        }

        return f[len1][len2];
    }
}
```

Summary

- List the transition matrix really helps analyzing the state transfer.