详细讲解，由浅入深

lebudeling 发布于 4 个月前 19.8k 阅读 动态规划 记忆化 C++ Python

一、热身

第一步，我们面对不想正面回答，如果是两个普通的字符串的比较，如何进行匹配？我想这个简单应该谁都会写：

C++

```
bool isMatch(string text, string pattern) {
    if (text.size() != pattern.size())
        return false;
    for (int i = 0; i < pattern.size(); i++) {
        if (pattern[i] != text[i])
            return false;
    }
    return true;
}
```

然后，我稍微改造一下上面的代码，略微复杂了一点，但是思路是一样的，容容易理解吧：

C++

```
bool isMatch(string text, string pattern) {

    // 判断 pattern 和 text 是否一样的
    return i == text.size();
}
```

好上百写，是为了稍这个算法改造成递归算法（伪码）：

Python

```
def isMatch(text, pattern) -> bool:
    if pattern is empty return text is empty
    first_match = (text not empty) and pattern[0] == text[0]
    return first_match and isMatch(text[1:], pattern[1:])
}
```

如果你能够理解这段代码，恭喜你，你的递归思维已经到位，正则表达式解决条件有点复杂，其实是基于这段递归代码进步改造而成的。

二、处理点号 '.' 通配符

点号可以匹配任意一个字符，方法简单，其实是基源单的，修改上面的伪码，再稍加改造即可：

Python

```
def isMatch(text, pattern) -> bool:
    if not pattern return not text
```

```
Python

def isMatch(text, pattern) -> bool:
    if not pattern return not text
    first_match = bool(text) and pattern[0] in {text[0], '.'}
    if len(pattern) >= 2 and pattern[1] == '.':
        # 发现 * 通配符
        return first_match and isMatch(text[1:], pattern)
    else:
        return first_match and isMatch(text[1:], pattern[1:])
```

现在我要的是个字符回答更复杂了点呢？这需要计算机最大的毕来算，假设重复 N 次吧，都没有 N 次呢，它是的所有的可能都存在了，之前的事情都搞定了，接下来的事情就很简单，不管 N 是多少，当前的选择只有两个：匹配 0 次，匹配 1 次，所以可以这样处理：

Python

```
if len(pattern) >= 2 and pattern[1] == '*':
    return isMatch(text, pattern[2:]) or \
           first_match and isMatch(text[1:], pattern)
# 解释：如果发现有字符 '*' 出现，
# 将模式指定字符 0 次，然后递归调用剩下的
# 将模型与 pattern[0] 和 text[0] 匹配成，继续 text
```

可以看到，我们要通过保留 pattern 中的 '*'，同时找回指增等 text，来实现 '*' 所字符重复匹配多次的问题，毕竟需要的

至此，正则表达式解决就基本完成了。

四、动态规划

我选择使用 "备忘录" 进行的方法来解决重复问题，有了最小解法，现代此问题马其实很多，我直使用两个变量 `i`，`j` 记录当前匹配到的位置，从而避免使用字符串切片，并且将 `i`，`j` 作为 备忘录，避免重复计算即可。

我将最大解决现代解决思路一起，方便可行，你可以发现优化解决手段是把最大解法「翻译」了一遍，加了个 memo 作为备忘录，优化时间。

Python

```
# 带备忘录的递归
def isMatch(text, pattern) -> bool:
    memo = dict() # 备忘录
    def dp(i, j):
        if (i, j) in memo: return memo[i, j]
        if j == len(pattern): return i == len(text)

        first = i < len(text) and pattern[j] in {text[i], '.'}

    return dp(0, 0)

# 最大解法
def isMatch(text, pattern) -> bool:
    if not pattern: return not text

    first = bool(text) and pattern[0] in {text[0], '.'}

    if len(pattern) >= 2 and pattern[1] == '*':
        return isMatch(text, pattern[2:]) or \
               first and isMatch(text[1:], pattern)
    else:
        return first and isMatch(text[1:], pattern[1:])
```

有的读者也许会问，**你怎么知道这个问题是个动态规划问题呢，你怎么知道它就符合「最优子问题」呢，这玩字不信有最优解吗？**

解答这个问题，最直观的应该是随便把这一个输入，然后暴力测，我定义这个问题可怎的，这属于定量分析，其实不用这么麻烦，下面我来教你定性分析，一眼就能看出「最优子问题」性质。

先来看简单的最优解法的步骤，我们抽象出通用解法的模板：

Python

```
def fib(n):
    ...
    ...
    ...
```