LeetCode #10

作者：friedwm
原文链接：https://ld246.com/article/1536591949769
来源网站：链滴

问题：模拟正则表达式中的.和 *

'.' 匹配任意单个字符。

'*' 匹配零个或多个前面的元素。

匹配应该覆盖**整个**字符串 (s)，而不是部分字符串。

说明:

- s 可能为空，且只包含从 a-z 的小写字母。
- p 可能为空，且只包含从 a-z 的小写字母，以及字符 . 和 *。

解答：

这是比较困难的一道题。官方中文站的通过率只有不到1/4。其实想清楚正则的匹配过程也不难。主有以下几点：

0. 正则匹配以正则单元为单位，普通字符也算正则单元，特殊的，如果有*
则其与前一个字符结合为一个正则单元。

1. java的正则表达式是NFA，默认是贪婪匹配，也就是说每个正则单元(记为pi)会尽可能多的匹配目字符串(s)。

2. 当前正则单元存在*时，既可尝试匹配当前目标字符也可跳到下一个单元，
认是尝试匹配。

3. 如若当前单元不匹配，则回退到上一次存在匹配分岔的位置(状态)去尝试。

4. 如若模式和字符串都匹配完了，则匹配成功。

5. 如若字符串匹配完了，正则式无法匹配完则失败，反之亦然。

6. 如没有可退回的位置，则匹配失败。

7. 注意边界情况。

如果想系统学习正则表达式可找《精通正则表达式》这本书来看看，力荐。

这里用一个链表模拟栈，记录每次分岔的状态(下次可尝试匹配的状态)

```
package xyz.quxiao.play.lab.leetcode;

import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Map.Entry;

/**
 * @author 作者 :quxiao 创建时间：2018/9/3 22:39
 */public class Problem10 {

  public static void main(String[] args) {
    Problem10 problem10 = new Problem10();
//    System.out.println(problem10.isMatch("abcda", ".*a.*c*da"));
//    System.out.println(problem10.isMatch("aa", "a*"));
//    System.out.println(problem10.isMatch("aab", "c*a*b"));
```

```
//    System.out.println(problem10.isMatch("aab", ".*"));
//    System.out.println(problem10.isMatch("mississippi", "mis*is*p*."));
//    System.out.println(problem10.isMatch("abc", "abcc"));
//    System.out.println(problem10.isMatch("", "a*c*"));
//    System.out.println(problem10.isMatch("aac", ".*.*a*c"));
  System.out.println(problem10.isMatch("aacd", ".*.*a*c"));
    System.out.println(problem10.isMatch("", ""));
  }

  public boolean isMatch(String s, String p) {
    if (s == null) {
      s = "";
    }
    if (p == null) {
      p = "";
    }
    int star = "*".charAt(0);
    // 提取匹配单元
  List patternUnit = new ArrayList<>();
    for (int i = 0; i < p.length(); i++) {
      int j = i + 1;
      if (j < p.length() && p.charAt(j) == star) {
        patternUnit.add(p.substring(i, i + 2));
        i++;
      } else {
        patternUnit.add(p.substring(i, i + 1));
      }
    }
    //
  LinkedList> matchQueue = new LinkedList<>();
    int i = 0;
    int j = 0;
    // 当还没匹配完时
  while (true) {
      if (i == s.length() && j == patternUnit.size()) {
        return true;
      }

      if (j == patternUnit.size()) {
        if (matchQueue.isEmpty()) {
          return false;
        } else {
          Entry pop = matchQueue.pop();
          i = pop.getKey();
          j = pop.getValue();
          continue;
        }
      }

      String pattern = patternUnit.get(j);
      String str = getChar(s, i);
      if (pattern.contains("*")) {
        matchQueue.add(0, new SimpleEntry<>(i, j + 1));
        if (unitMatch(str, pattern)) {
```

```
        // 推进
i++;
        // 注意i可能因为向前推进导致超过边界，这里不分辨是否在边界上，超出时依然让j向前推进
if (i >= s.length()) {
          j++;
        }
        continue;
      } else {
        if (matchQueue.isEmpty()) {
          return false;
        } else {
          Entry pop = matchQueue.pop();
          i = pop.getKey();
          j = pop.getValue();
          continue;
        }
      }
    } else {
      if (unitMatch(str, pattern)) {
        // 推进
i++;
        j++;
        continue;
      } else {
        if (matchQueue.isEmpty()) {
          return false;
        } else {
          Entry pop = matchQueue.pop();
          i = pop.getKey();
          j = pop.getValue();
          continue;
        }
      }
    }
  }
}

private boolean unitMatch(String character, String unit) {
  if (unit.startsWith(".")) {
    return true;
  } else if (character == null && unit.contains("*")) {
    return true;
  } else if (character == null){
    return false;
  } else {
    return unit.startsWith(character);
  }
}

private String getChar(String s, int pos) {
  if (pos >= s.length()) {
    return null;
  }
  return s.substring(pos, pos + 1);
```

```
    }
}
```

原文链接：LeetCode #10