Regular Expression Matching

Difficulty: Hard

这是一道正则表达式匹配的问题。和普通的字符串匹配的区别在于，由于引入了通配符，就从一个DFA变成了NFA。理论上是可以通过将NFA转换为DFA来做的，判定算法的复杂度应该是$O(ns^2)$，这里n是字符串长度，s为NFA状态数目。

这里采用的是dp的思想。首先将模式串p分段为x或x*的形式，这里的x表示任意字母或"."。例如，样例5中的

> *mis*is*p*.*

被分成

> *"m", "i", "s*", "i", "s*", "p*", "."*

然后用两个数组来记录上面分段的结果。数组np记录所有的分段中的字母，即

> *np = ["m", "i", "s", "i", "s", "p", "."]*

用数组star记录对应位置的字母是否跟有星号*。

> *star = [false, false, true, false, true, true, false]*

接下来就是dp的部分了。用一个矩阵d来记录匹配的结果。d_{ij}表示从$s[0,\cdots,j-1]$与$p[0,\cdots,i-1]$匹配。初始d_{00}为true，其余元素均为false. 这里的d_{00}作为哨兵，可以简化边界条件的判断。

然后就可以写d_{ij}的递推式了。下面所有的情况都是在字符p_i与s_j 的条件下讨论的，否则d_{ij}一定为false.

- Case 1: d_{ij}为true，那么$d_{i+1,j+1}$也为true。这是因为，$s[0,\cdots,j-1]$与$p[0,\cdots,i-1]$匹配，p_i又与s_j匹配，所以$s[0,\cdots,j]$与$p[0,\cdots,i]$匹配。
- Case 2: $star[i]$为true，同时$d_{i+1,j}$为true，那么$d_{i+1,j+1}$也为true。这是因为，$p[0,\cdots,i]$与$s[0,\cdots,j-1]$匹配，而$p[i]$又含有星号，也就是可以继续向后匹配，所以$p[0,\cdots i]$与$s[0,\cdots,j]$匹配。
- Case 3: 其它情况，依次遍历d_{ij}, $d_{i-1,j}$, $\cdots$, d_{kj}，其中$0 \le k \le i$，满足$\forall x, k \le x \le i, star[x-1]$为true, 同时$\forall x, k < x \le i, d_{xj}$为false，$d_{kj}$为true。这种情况对应了$p_i$前面有数个连续的带星号的字符$p[k-1,\cdots,i-1]$，这些字符都没有和$s$中的任何字符匹配。例如，$p$ ="a*b*c"，s ="c"，这种情况下，"a*", "b*"都没有匹配到任何字符，p中的"c"与s中的"c"匹配。

当计算完整个矩阵d后，显然如果d最右下角的元素为true，就意味着p与s全部匹配了。除此以外，类似于上述Case 3，如果p的末尾有一连串连续的带星号的字符$p[i, \cdots, m]$，并且这些字符都没有匹配，那么如果$d_{i-1,n}$为true，同样是可以完全匹配的。例如$p =$"ab*c*"，$s =$"a"。

最后放上样例4对应的矩阵d。

$s =$"aab", $p =$"c*a*b"

		A	A	B
	✓			
c*				
a*		✓	✓	
b				✓