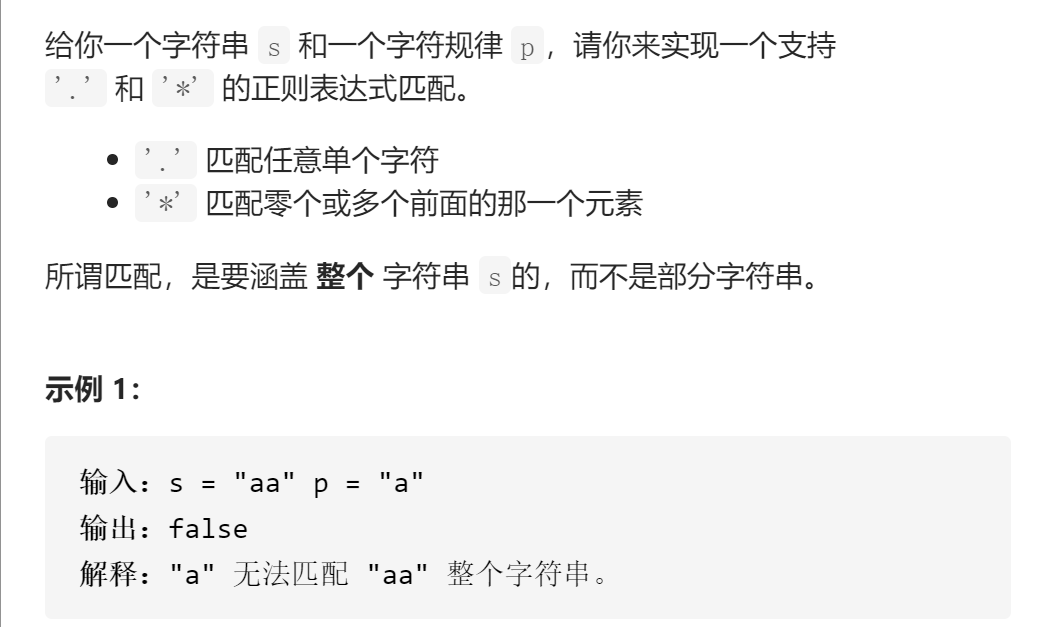
# 动态规划-[10. 正则表达式匹配](https://leetcode-cn.com/problems/regular-expression-matching/)

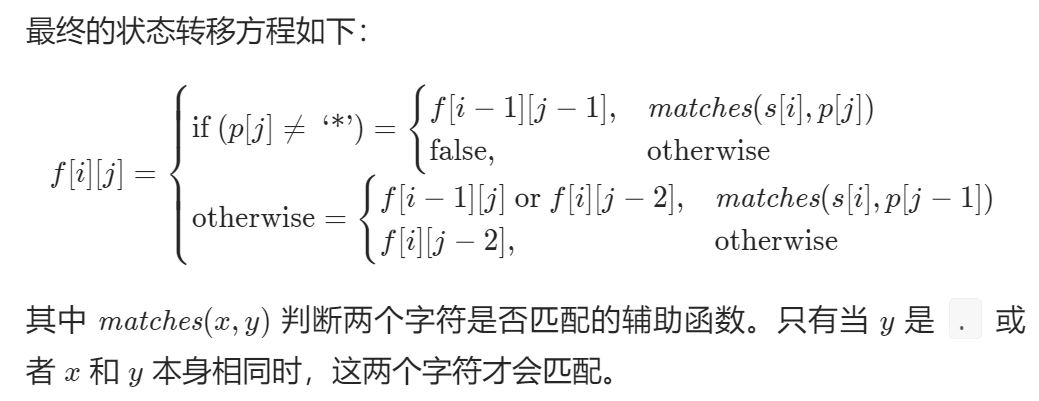
## 题目描述：



## 题目分析：

1. **经验分析：对于两个字符串的动态规划问题，一般都是dp[i] [j],表示s的前i个字符串和p的前j个字符串的匹配情况**
2. **动态规划的问题一般是需要进心一个比较严谨的分析来分析其动态的转移：寻找最优子结构**
3. **对于问题分析的思维：学会知识迁移和问题转换（从另外一个角度来看问题）**
4. **动态规划的问题一般都可以用记忆化搜索来解决（深度优先+记忆化）；**
5. **对于这样的问题：其思维性一般比较的强；我一般不能够轻易的ak这样的问题；但是要尽力的去尝试，提升自己思维的能力！！！**
6. **吃透+每日一题+每日精进！！！**

## 动态规划



class Solution {  
public:  
 bool isMatch(string s, string p) {  
 //动态规划  
 //dp[i][j]表示s的前i个字符和p的钱j个字符的匹配情况  
 int m=s.size();  
 int n=p.size();  
 vector<vector<bool>>dp(m+1,vector<bool>(n+1,false));  
 //细节问题  
 dp[0][0]=true;  
 //lambda表达式  
 auto matches=[&](int i,int j){  
 if(i==0)return false;  
 //此处有一个下标关系  
 if(p[j-1]=='.')return true;  
 //if(s[i-1]==s[j-1])return true;  
 return s[i-1]==p[j-1];  
 };  
  
 for(int i=0;i<=m;++i){  
 for(int j=1;j<=n;++j){  
 if(p[j-1]=='\*'){  
 //dp[i][j]|=dp[i][j-2];  
 dp[i][j]=dp[i][j]|dp[i][j-2];  
 if(matches(i,j-1))//dp[i][j] |= dp[i-1][j];   
 dp[i][j]=dp[i][j]|dp[i-1][j];  
 }else if(matches(i,j)){  
 // dp[i][j] |= dp[i-1][j-1];  
 dp[i][j]=dp[i][j]|dp[i-1][j-1];  
 }  
 }  
 }  
 return dp[m][n];  
 }  
};

## 记忆化搜索

class Solution {  
public:  
  
 int sn;  
 int pn;  
  
 vector<vector<bool>> memo; //记忆容器  
  
 bool backtracking(string&s,string&p,int sindx,int pindx) {  
  
 if(!memo[sindx][pindx]) return false;//记忆化剪枝  
  
 if(sindx==sn){   
 memo[sindx][pindx] = pindx==pn||(pindx+1<pn&&p[pindx+1]=='\*'&&backtracking(s,p,sindx,pindx+2));  
 return memo[sindx][pindx];  
 }  
 else if(pn==pindx) {  
 memo[sindx][pindx] = false;  
 return false;  
 }  
  
   
  
 if(s[sindx]==p[pindx]||p[pindx]=='.') { //当前位置成功匹配   
  
 if(pindx+1<pn&&p[pindx+1]=='\*') {  
 //复制+终结+越过  
 memo[sindx][pindx] = backtracking(s,p,sindx+1,pindx)||backtracking(s,p,sindx+1,pindx+2)||backtracking(s,p,sindx,pindx+2);  
 return memo[sindx][pindx];  
 }  
 else {  
 memo[sindx][pindx] = backtracking(s,p,sindx+1,pindx+1);  
 return memo[sindx][pindx];  
 }   
 }else{ //未成功匹配  
 //找p下一位\* 或者匹配失败  
 if(pindx+1<pn&&p[pindx+1]=='\*') {  
 memo[sindx][pindx] = backtracking(s,p,sindx,pindx+2);  
 return memo[sindx][pindx];  
 }  
 else {  
 memo[sindx][pindx] = false;  
 return false;  
 }  
 }  
  
 }  
  
 bool isMatch(string s, string p) {  
 sn = s.size();  
 pn = p.size();  
  
 memo = vector<vector<bool>>(sn+1,vector<bool>(pn+1,true));  
  
 return backtracking(s,p,0,0);  
  
 }  
};  
  
作者：randy\_woll  
链接：https://leetcode-cn.com/problems/regular-expression-matching/solution/c-hui-su-fa-dfsji-yi-hua-by-randy\_waler-ws8t/  
来源：力扣（LeetCode）  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。