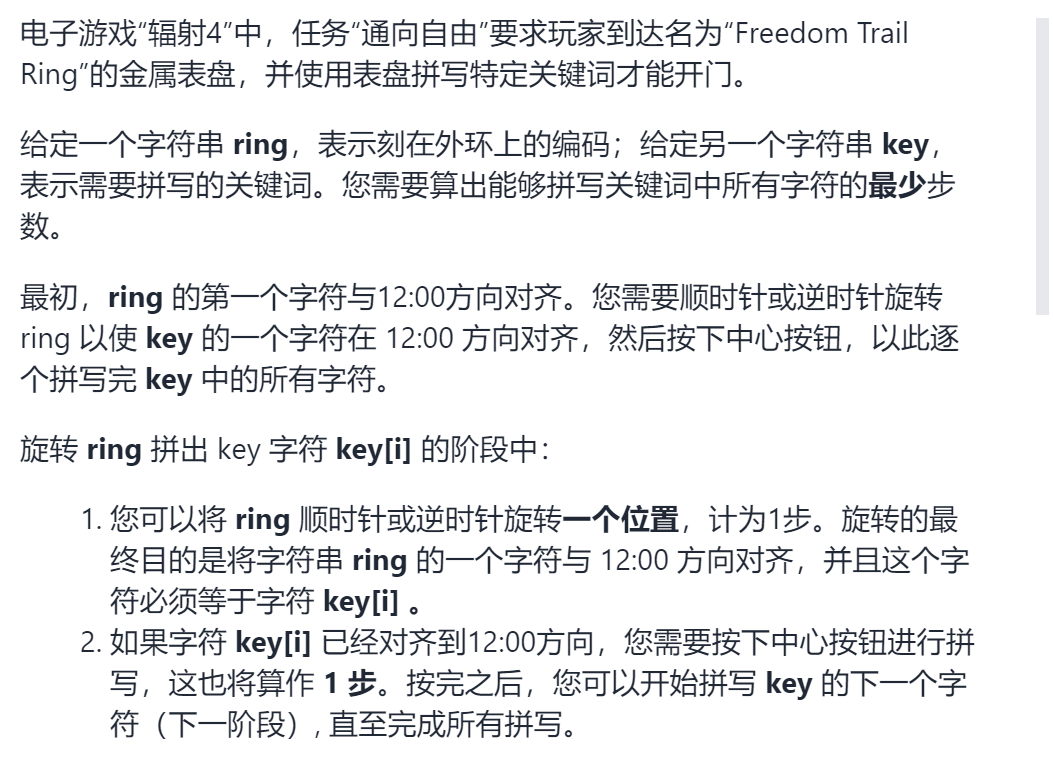
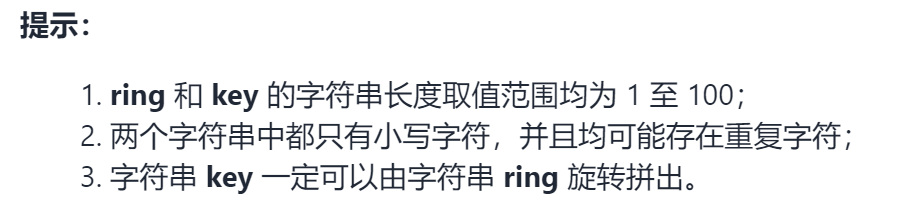
# 力扣每日一题（困难题）

## 2021/10/12:（深度优先搜索+记忆化）

### 题目描述：[514. 自由之路](https://leetcode-cn.com/problems/freedom-trail/)







### 题目分析：

* **这道题目虽然长，但是题意还是比较容易搞懂的。**
* **很容易想到深度优先搜索，但是考虑到字符的长度可能为100，应该意识到深度优先搜索很容易爆时间**

#### 深度优先+剪枝（爆时间）

class Solution {  
private:  
 //记录到key【i】的最小步数  
 vector<int>res;  
 //存储各个字符到其他字符之间的距离--用下标  
 vector<vector<int>>myvec;  
 //各个字符的位置  
 unordered\_map<char,vector<int>>mymap;  
  
 void dfs(string&ring,string&key,int k,int sum,int now){  
 if(k>=key.size())return;  
 bool flag=false;//字符在ring中是否唯一  
 if(mymap[key[k]].size()==1)flag=true;  
 for(auto&tag:mymap[key[k]]){  
 //tag为ring[tag]为key[k]的下标  
 if(flag){  
 if(sum+myvec[now][tag]<res[k]){  
 dfs(ring,key,k+1,sum+myvec[now][tag],tag);  
 //res[k]=sum+myvec[now][tag];  
 }  
 }else dfs(ring,key,k+1,sum+myvec[now][tag],tag);  
  
 res[k]=min(res[k],sum+myvec[now][tag]);  
 }  
 }  
public:  
 int findRotateSteps(string ring, string Key) {  
 int size=Key.size();  
 //减枝条：所见key的长度，相邻字符串消除  
 string key=Key.substr(0,1);  
 for(int i=1;i<Key.size();i++){  
 if(Key[i]==Key[i-1])continue;  
 key.push\_back(Key[i]);  
 }  
 //深度优先搜索--在唯一处取最短(剪枝)  
 int n=ring.size(),nk=key.size();  
 //记录到key【i】的最小步数  
 res.resize(nk,INT\_MAX);  
 myvec.resize(n,vector<int>(n,0));  
 for(int i=0;i<n;i++){  
 //循环取余操作  
 for(int j=0;j<n;j++){  
 myvec[i][j]=min(abs(j-i),n-abs(j-i));  
 }  
 }  
 for(int i=0;i<n;i++){  
 mymap[ring[i]].push\_back(i);  
 }  
 dfs(ring,key,0,0,0);  
 return res[nk-1]+size;  
   
   
 }  
};

#### 深度优先+记忆化搜索==性能==动态规划

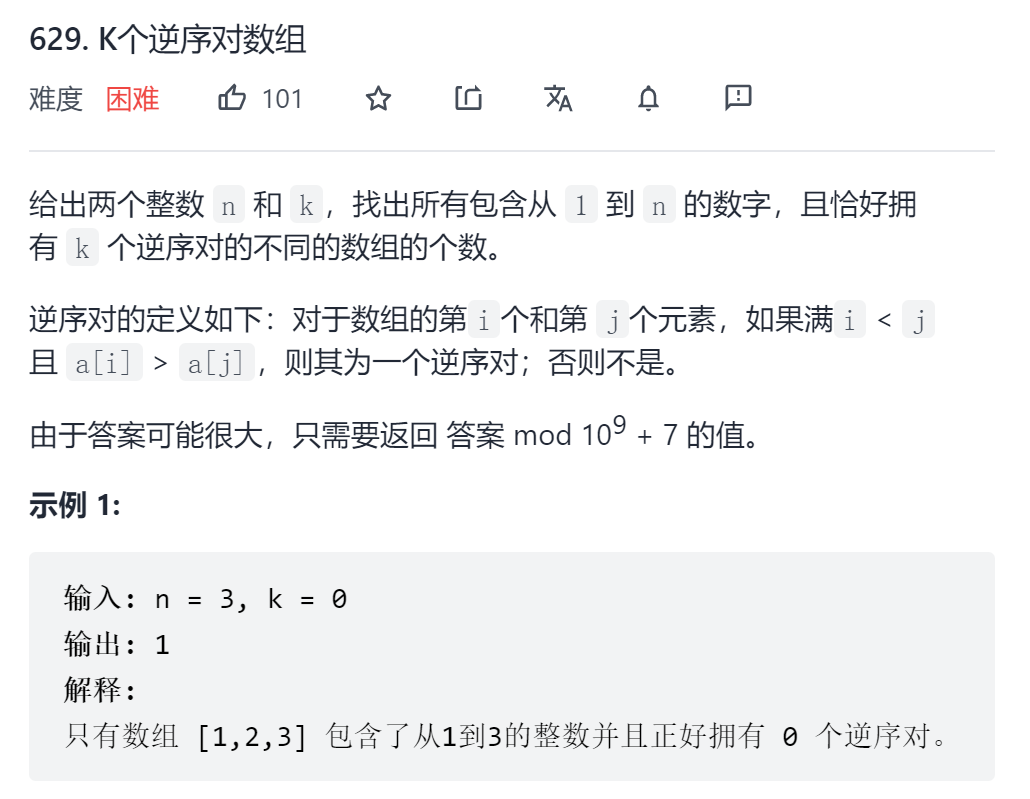
class Solution {  
private:  
 //优化-添加记忆化搜索  
 //memo[i][j]标识从ring字符串12：00处于i位置；key从搜索到结束  
 //的最短距离  
 vector<vector<long long int>>memo;  
 //记录到key【i】的最小步数  
 vector<int>res;  
 //存储各个字符到其他字符之间的距离--用下标  
 vector<vector<int>>myvec;  
 //各个字符的位置  
 unordered\_map<char,vector<int>>mymap;  
  
 long long int dfs(string&ring,string&key,int k,int now){  
 if(k>=key.size())return 0;  
 if(memo[now][k]!=INT\_MAX)return memo[now][k];  
 for(auto&tag:mymap[key[k]]){  
 memo[now][k]=min(myvec[now][tag]+dfs(ring,key,k+1,tag),memo[now][k]);  
 }  
 return memo[now][k];  
 }  
public:  
 int findRotateSteps(string ring, string key) {  
 //深度优先搜索--在唯一处取最短(剪枝)  
 int n=ring.size(),nk=key.size();  
 //记录到key【i】的最小步数  
 res.resize(nk,INT\_MAX);  
 myvec.resize(n,vector<int>(n,0));  
 memo.resize(n,vector<long long int>(nk,INT\_MAX));  
 for(int i=0;i<n;i++){  
 //循环取余操作  
 for(int j=0;j<n;j++){  
 myvec[i][j]=min(abs(j-i),n-abs(j-i));  
 }  
 }  
 for(int i=0;i<n;i++){  
 mymap[ring[i]].push\_back(i);  
 }  
 //审题——起初，第一个字符对用12：00方向  
 /\*  
 for(int i=0;i<n;i++){  
 if(ring[i]==key[0]){  
 memo[i][0]=dfs(ring,key,0,i);  
 }  
 }  
 \*/  
 memo[0][0]=dfs(ring,key,0,0);  
 return memo[0][0]+nk;  
   
   
 }  
};

### 反思总结：

1. **对于可以用深度优先搜索的题，如果爆时间，或者一开始注意到输入可能很大，就应该意识到用记忆化搜索**
2. **为什么记忆化搜索可以有效的降低时间消耗？**
   * **深度优先搜索会重复进行很多相同的子搜索**
   * **记忆化搜索记忆的应该是这些子搜索的答案**
   * **比如这道题的子搜索就是，字符ring的下标处于位置i ;从key的位置开始到结尾的操作步数。**
3. **对于一道困难题，有了大概的思路之后，应该在纸上大概分析出简要流程和伪代码后再动手写代码；这样可以使得代码更加清晰和有序；也不容易出bug**

## 2021/10/15:(动态规划爆时间+状态转移方程的化简优化)

### 题目描述：[629. K个逆序对数组](https://leetcode-cn.com/problems/k-inverse-pairs-array/)



### 题目分析：

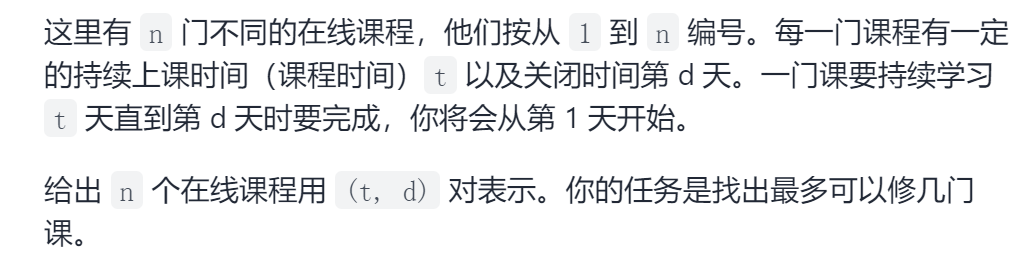
* **初次看到这个题目，暴力的做法肯定是深度优先搜索，遍历每一种可能并检测其逆序列是否为k；但是看到n<=1000且k<=1000；时间复杂度O（n!)肯定会爆时间**
* **优化：深度优先搜索+记忆化==动态规划；如何的去记忆化？同归归纳推理可以发现：每次处理前n个数，dp[i] [j]表示前i个数中符合逆序列为j的情况数；对于新加入的数i+1，其在数组中可以有i中可能的位置。放在最后不会增加逆序对，放在倒数第二个位置会增加逆序对一个，依次类推。则可以得出状态转移方程：dp[i] [j]=dp[i-1] [j]+dp[i-1] [j-1] +dp[i-1] [j-2]+....+dp[i-1] [j-i+1]**
* **从记忆化的角度导出动态规划？一定要养成写出状态转移方程的习惯**
* class Solution {  
  public:  
   int kInversePairs(int n, int k) {  
   //动态规划  
   int ans=0;  
   int mod=1e9+7;  
   vector<vector<int>>dp(n+1,vector<int>(k+1,0));  
   //初始化  
   //for(int i=0;i<=n;i++)dp[i][0]=1;  
   dp[1][0]=1;  
   for(int i=2;i<=n;i++){  
   //i表示前i各元素集合  
   for(int pos=1;pos<=i;pos++){  
   //pos表示数字i在数组中的位置  
   //基础逆序对  
   int base=i-pos;  
   //遍历前i-1个元素含各种逆序对情况下的个数  
   for(int j=0;j<=k;j++){  
   //dp[i-1][j]表示前i-1个数中逆序对为j个的情况有几种   
   if(base+j>k)continue;  
   dp[i][base+j]+=dp[i-1][j];  
   dp[i][base+j]%=mod;  
   }  
     
   }  
   }  
   return dp[n][k];  
   }  
  };
* **但依然爆时间：这里设计到一个通过化简状态转移方程来优化程序的技巧**
* **f(i, j) = f(i - 1, j) + f(i - 1, j - 1) + ... + f(i - 1, j - i + 1)**
* **f(i, j - 1) = f(i - 1, j - 1) + f(i - 1, j - 2) + ... + f(i - 1, j - i)**
* **f(i, j) - f(i - 1, j) = f(i, j - 1) - f(i - 1, j - i)**  
  **==> f(i, j) = f(i, j - 1) + f(i - 1, j) - f(i - 1, j - i)**
* **使用时注意边界条件和初始化**
* class Solution {  
  public:  
   int kInversePairs(int n, int k) {  
   int ans=0;  
   int mod=1e9+7;  
   vector<vector<int>>dp(n+1,vector<int>(k+1,0));  
   //初始化  
   for(int i=1;i<=n;i++)dp[i][0]=1;  
   for(int i=2;i<=n;i++){  
   for(int j=1;j<=k;j++){  
   int temp=(dp[i-1][j]+dp[i][j-1])%mod;  
   //这里涉及到一个编程技巧，涉及到减的地方需要及一个mod确保相见之后得到一个正数  
   dp[i][j]=(mod+temp-(j>=i?dp[i-1][j-i]:0))%mod;  
     
   }  
     
   }  
   return dp[n][k];  
   }  
  };

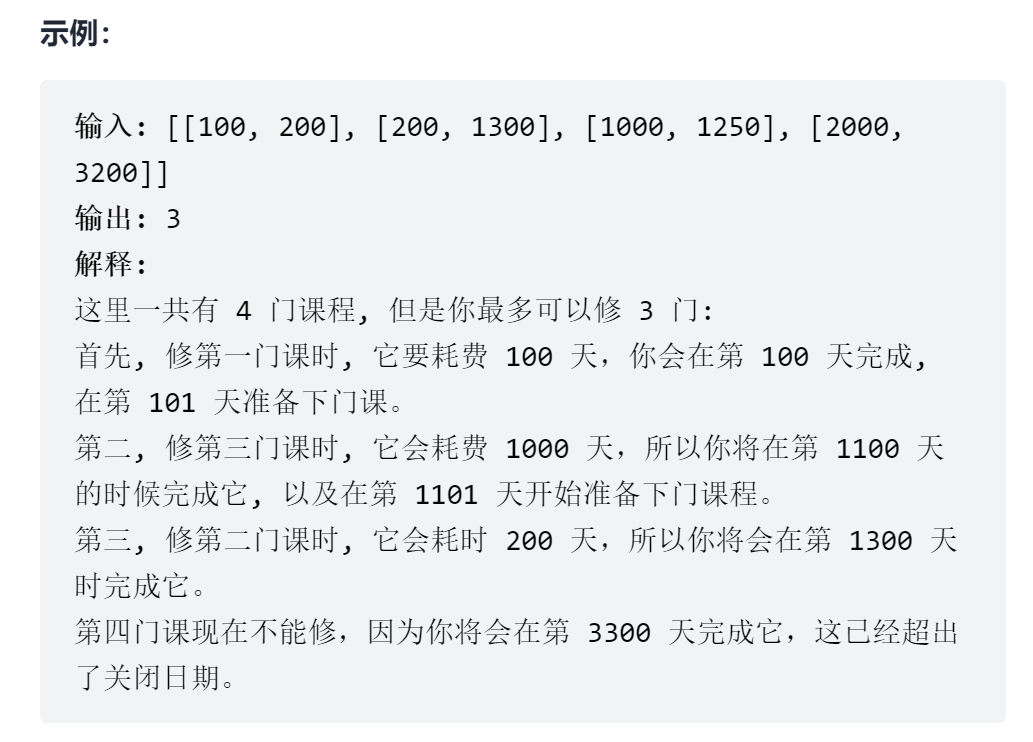
### 反思总结：

* 1. **由深度优先搜索+记忆化==导出动态规划**
  2. **动态规划一定要明晰的写出状态转移方程**
  3. **养成化简复杂动态规划方程的习惯**
  4. **注意状态转移方程使用中的初始化和边界条件**

## 2021/10/19：[630. 课程表 III](https://leetcode-cn.com/problems/course-schedule-iii/)

### 题目描述：





### 题目分析：（贪心+优先队列）

1. **这道题是一道关于贪心的题；这道题的贪心没有那么的显而易见，而是需要优化的贪心。**
2. **首先课程的终止时间表示一个课程的紧急程度；这里的一个贪心点就是：我们应该选择终止时间最早的课程**
3. **但是在这个贪心点上并不能保证最后得出的答案一定是最优的：因为终止时间早的课程也有可能花费更多的时间；**
4. **贪心的优化就是，如果已经选择的课程中有比现在要选的课程中更费时的课程，那么取代之！！！**

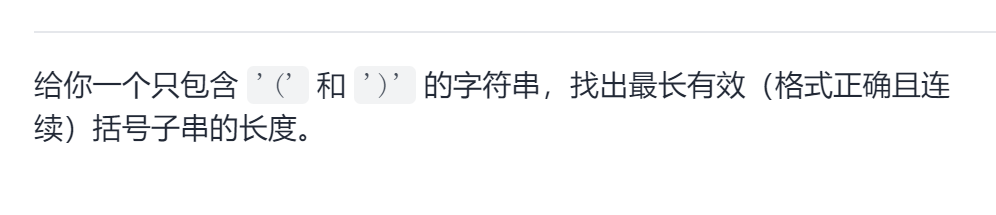
class Solution {  
private:  
 bool static cmp(vector<int>&a,vector<int>&b){  
 if(a[1]==b[1])return a[0]<b[0];  
 return a[1]<b[1];  
 }  
public:  
 int scheduleCourse(vector<vector<int>>& courses) {  
 //贪心算法+数据结构（优先队列）  
 //优先队列-自定义排序方法  
 //先学习结束时间最早的课，如过之后有课学习不了，判断是否可以将该可和最耗时的课程置换  
 sort(courses.begin(),courses.end(),cmp);  
 priority\_queue<int>que;  
 int time=0;//表示时间和  
 int numcour=0;//表示已经修的课程  
 int n=courses.size();  
 for(int i=0;i<n;i++){  
 if(courses[i][0]>courses[i][1])continue;  
 if(time+courses[i][0]<=courses[i][1]){  
 time+=courses[i][0];  
 ++numcour;  
 que.push(courses[i][0]);  
 }else{   
 //if(time-que.top()+courses[i][0]<=courses[i][1]){//这个条件并不能保证被替换掉的课程一定比该课程更费时间  
 if(!que.empty()&&que.top()>courses[i][0]){  
 time=time-que.top()+courses[i][0];  
 que.pop();  
 que.push(courses[i][0]);  
 }  
 }  
 }  
 return numcour;  
 }  
};

### 该题总结：

* **复杂一点的贪心往往需要在一个贪心点的基础之上进行优化！！！**
* **补充说明优先队列自定义排序：**
  1. //优先队列默认是从大到小排序的  
     priority\_queue<int> q1;//默认从大到小排序，整数中元素大的优先级高   
     //如果要改成从小到大排序，可以加上greater<类型>  
     priority\_queue<int,vector<int>,greater<int> >q1;  
       
     //自定义排序  
     struct cmp1{  
      bool operator()(int x,int y)  
      {  
      return x>y;//小的优先级高 ,从小到大排 //注意此处的比较与sort中数组排序中的比较相反  
      }  
     };   
     struct cmp2{  
      bool operator()(const int x,const int y)  
      {  
      return tmp[x]>tmp[y];  
      }  
     };   
     struct node{  
      int x,y;  
      friend bool operator<(node a,node b)  
      {  
      return a.x>b.x;//按x从小到大排   
      }  
     };  
     priority\_queue<int>q1;  
     priority\_queue<int,vector<int>,cmp1>q2;  
     priority\_queue<int,vector<int>,cmp2>q3;  
       
     ————————————————  
     版权声明：本文为CSDN博主「yanyanwenmeng」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。  
     原文链接：https://blog.csdn.net/yanyanwenmeng/article/details/78153192

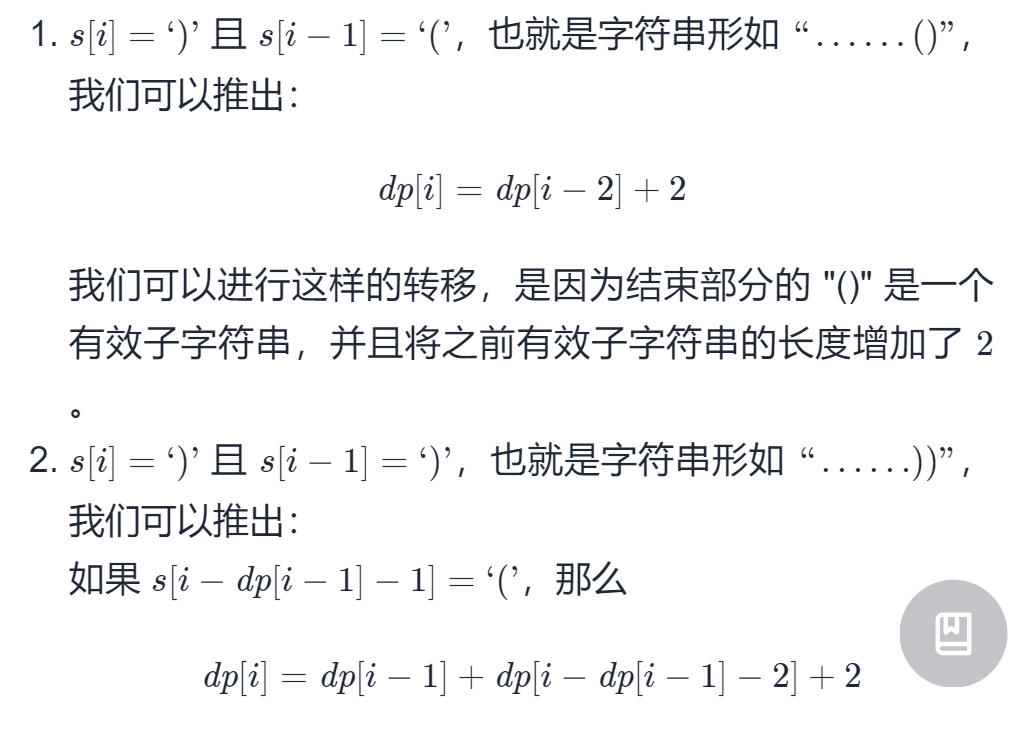
## 2021/10/22:[32. 最长有效括号](https://leetcode-cn.com/problems/longest-valid-parentheses/)

### 题目描述：



### 题目分析：

**一般来说，含有“最”字的题都可以用动态规划的思想来做，这题也一样。dp[i]可以认为是以s[i]字符结尾的最长有效括号的长度；不同的是，该题的动态规划一次以两个字符来分析。**



**我在思考这个题目的时候也想到了动态规划，以及dp【i】的含义，但是如何进行状态转移却没有想出来。**

**关于括号匹配的题目，很容易想到数据结构栈，这个题目也可以用数据结构栈来做，但是需要做一点优化！！！**

class Solution {  
public:  
 int longestValidParentheses(string s) {  
 //数据结构：单调栈  
 //匹配的最大长度  
 int res=0;  
 stack<int>stk;//存储下标  
 int n=s.size(),i=0;  
 vector<int>dp(n+1,0);  
 while(i<n){  
 if(stk.empty()||(s[stk.top()]==')')||s[i]=='('){  
 //当栈空、栈顶部为头括号、要进栈的为左括号  
 //字符进栈  
 stk.push(i);  
 ++i;  
 }else{  
 //此时要出栈-确定一次连续出栈的值  
 int end=i,begin=-1;  
 while(i<n&&!stk.empty()){  
 if(s[stk.top()]=='('&&s[i]==')'){  
 end=i;  
 begin=stk.top();  
 stk.pop();  
 ++i;  
 }else break;  
 }  
 dp[end+1]=dp[begin]+end+1-begin;  
 res=max(res,dp[end+1]);  
 }  
 }  
 return res;  
  
 }  
};

**还有一种比较妙的做法是通过贪心来做的！！！**

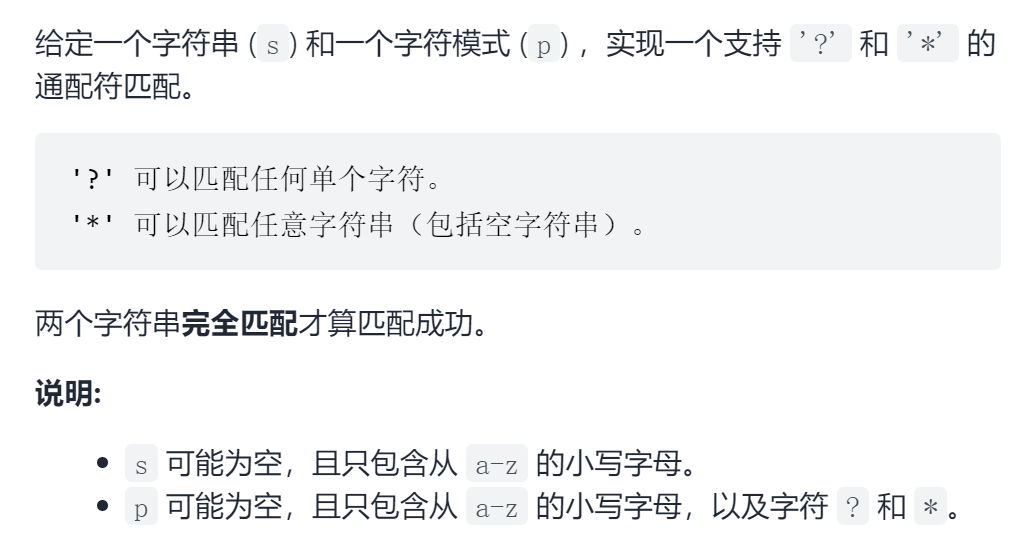
<https://leetcode-cn.com/problems/longest-valid-parentheses/solution/zui-chang-you-xiao-gua-hao-by-leetcode-solution/>

### 题目总结：

**不要被困难题吓到，困难题在思维上更有深度，做起来更有挑战性！！！但是从本质上来讲万变不离其宗，一是要理解题意，并在思考的基础上转换题意，选好数据结构和算法，加上一定代码力便可AC！**

## [44. 通配符匹配](https://leetcode-cn.com/problems/wildcard-matching/)【动态规划】

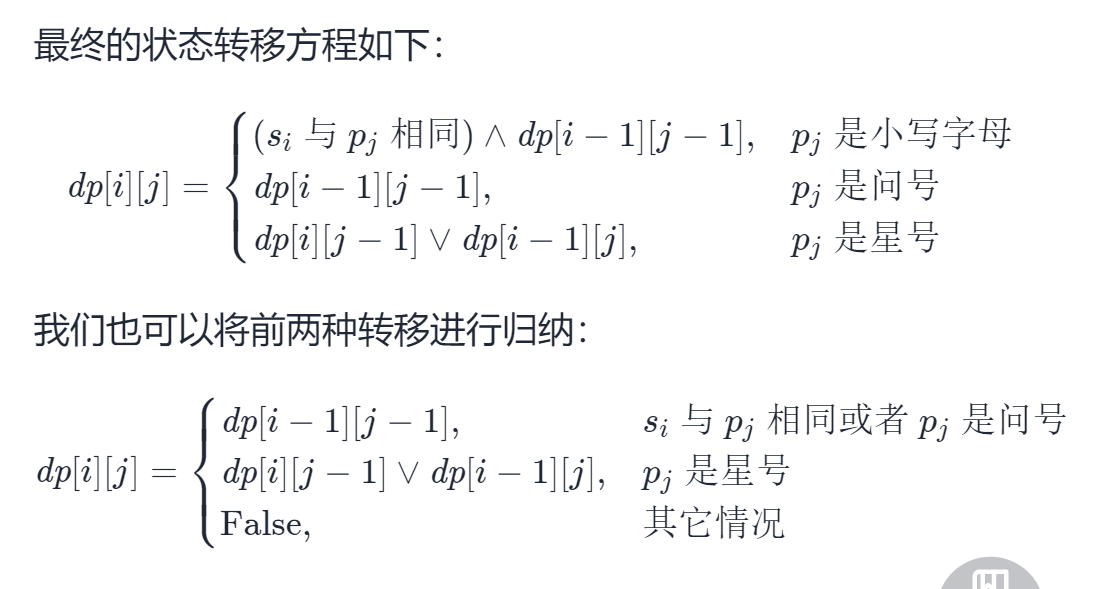
### 题目描述:



### 题目分析：

**字符串匹配类的题，很容易想到用动态规划。**

1. **dp[i] [j] 表示的是以i结尾的s[i]串与以j结尾的p[j]串是否匹配。**
2. **这样的题目需要根据通配符的？或 \* 分类讨论；但是一定要有一个状态转移的概念；**

* 

1. **这样的题建议画个矩阵进行模拟，以加强理解；当p[j]是星号的时候，星号要么为空，要么代表一个或多个字符。当代表一个或多个字符是使用dp[i] [j]=dp[i-1] [j]进行转移。**
2. **确定好状态转移方程之后初始化**
3. **不管遍历顺序是什么样，dp[i] [j] 都可以从dp[m] [n] 只要m<i和n<j其中有一个满足条件**

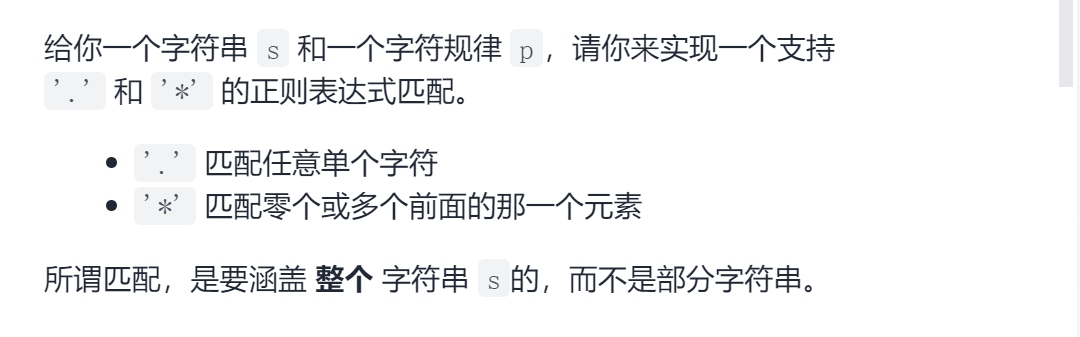
class Solution {  
public:  
 bool isMatch(string s, string p) {  
 //动态规划  
 int ns=s.size(),np=p.size();  
 vector<vector<bool>>dp(ns+1,vector<bool>(np+1,false));  
 dp[0][0]=true;  
 //特殊情况处理  
 /\*  
 ""  
 "\*\*\*\*\*\*"  
 \*/  
 bool flag=true;  
 for(int i=0;i<np;i++){  
 if(p[i]!='\*')flag=false;  
 }  
 if(flag&&np)return true;//应对全星的情况  
  
 //换一个遍历顺序  
 for(int j=1;j<=np;j++){  
 for(int i=1;i<=ns;i++){  
 if(s[i-1]==p[j-1]){  
 dp[i][j]=dp[i-1][j-1];  
 }else if(p[j-1]=='?'){  
 dp[i][j]=dp[i-1][j-1];  
 }else if(p[j-1]=='\*'){  
 dp[0][j]=dp[0][j-1];  
 dp[i][j]=(dp[i][j-1]|dp[i-1][j-1]|dp[i-1][j]);//\*代表空，\*代表一个，\*代表一堆  
 }  
 }  
 }  
 return dp[ns][np];  
 }  
};

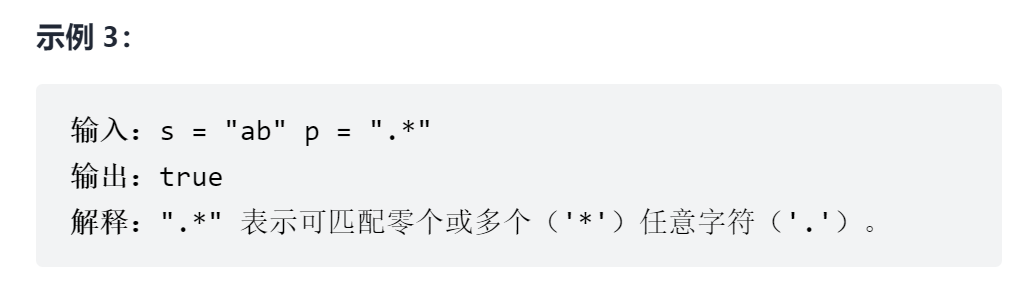
### 题目总结：

**面对动态规划的题，思路是有，但是思路还不是很顺畅，总是会卡壳！！！多多练习和总结！！**

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

### 题目描述：





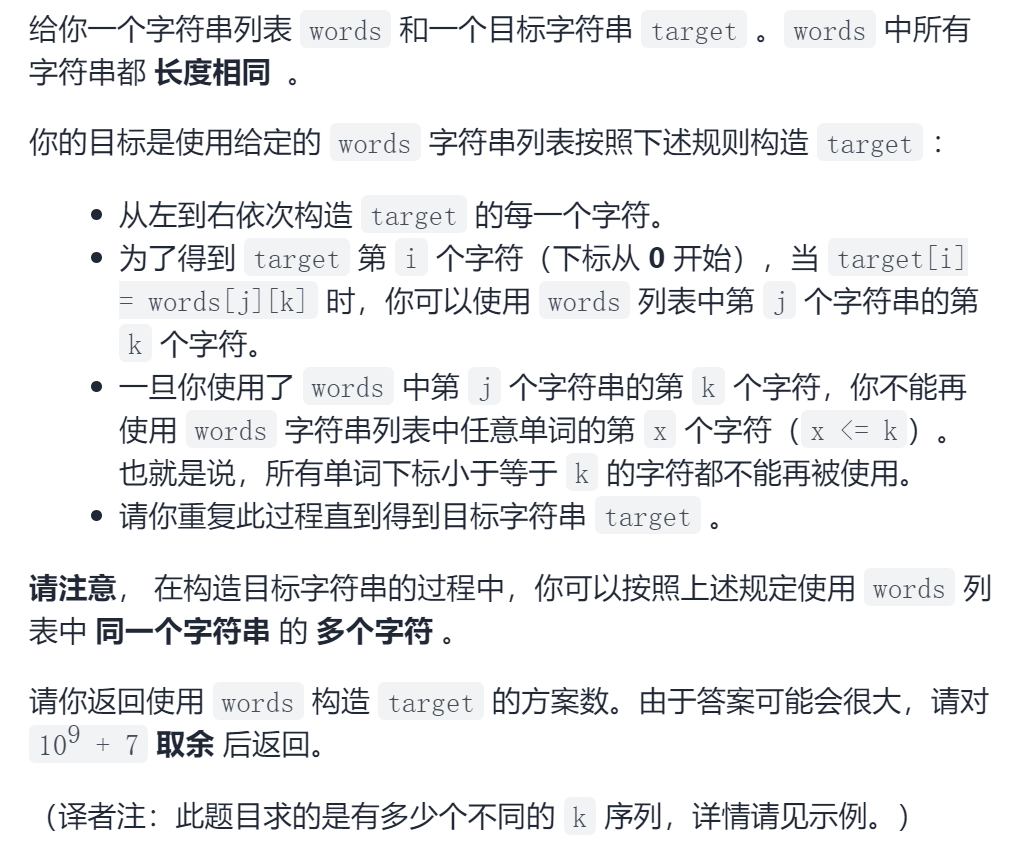
### 题目分析：

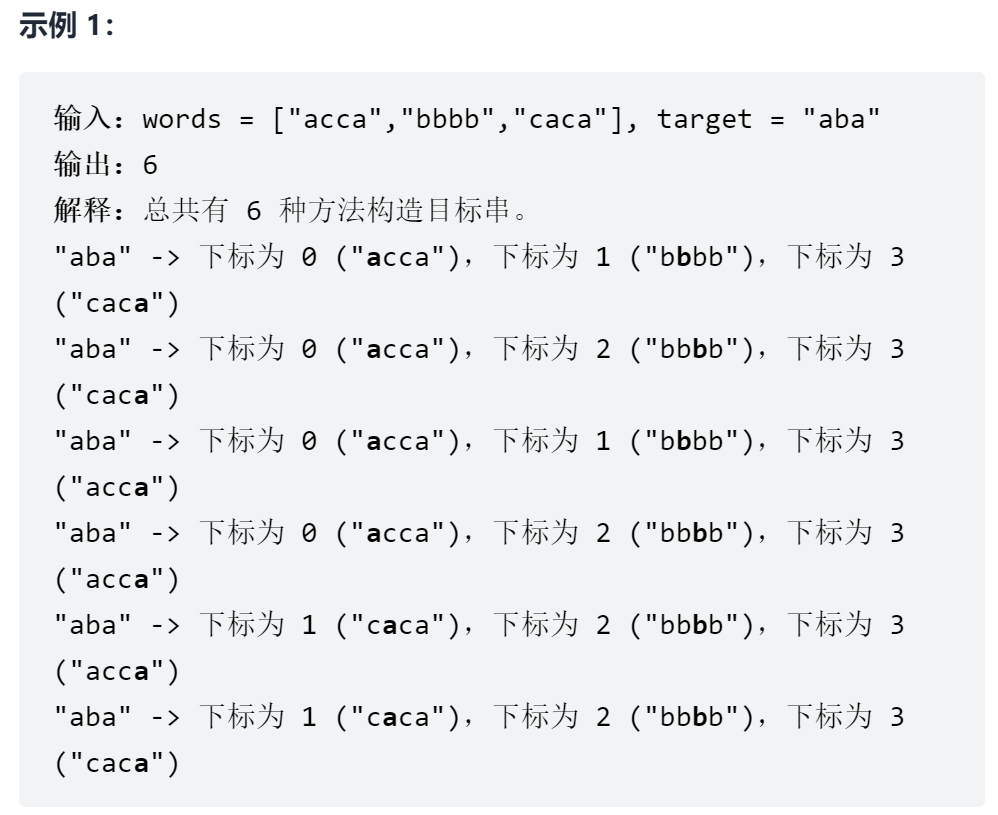
1. **当s[i]==p[j]或者p[j]=='.'时：dp[i] [j]=dp[i-1] [j-1];**
2. **当p[j]==\* 号时**
   * **若匹配0个字符：dp[i] [j]=dp[i] [j] | dp[i] [j-1]**
   * **若消除一个字符：dp[i] [j]=dp[i] [j] | dp[i] [j-2] (考虑"aab"和"c*a*b的情况"，消除一个字符可能导致与0的情况匹配**)\*\*
   * **考虑到消除一个字符后可能导致与s为空时的匹配:故dp[0] [j]=dp[0] [j-2];**
   * **若后缀为两个星号,dp【i】[j]=dp[i-1] [j-1] 状态转移就是这个星号可以不要**
   * **若后缀为点加星号：则表示可以代表任何字符，故：dp[i] [j]=dp[i] [j] | dp[i-1] [j];(此处状态转移，s串的前i-1个字符是否可以与p的前j个字符匹配)**
   * **若后缀为字符加星号：则要判断在s[i-1]==p[j-2]的前提下：dp[i] [j]=dp[i] [j] | dp[i-1] [j]**

class Solution {  
public:  
 bool isMatch(string s, string p) {  
 //动态规划  
 int m=s.size(),n=p.size();  
 vector<vector<bool>>dp(m+1,vector<bool>(n+1,false));  
 dp[0][0]=true;  
 for(int i=1;i<=m;i++){  
 for(int j=1;j<=n;j++){  
 if(s[i-1]==p[j-1]||p[j-1]=='.'){  
 dp[i][j]=dp[i-1][j-1];  
 }else if(p[j-1]=='\*'){  
 //匹配0个或消除一个  
 dp[i][j] = dp[i][j] | dp[i][j - 1]|dp[i][j-2];  
 //消除1个导致与s为空串时的匹配  
 dp[0][j] = dp[0][j-2];  
 if(p[j-2]=='\*')dp[i][j]=dp[i-1][j-1]|dp[i][j];  
 else if(p[j-2]=='.')dp[i][j]=dp[i-1][j]|dp[i][j];  
 else if(s[i-1]==p[j-2])dp[i][j]=dp[i-1][j]|dp[i][j];  
 }  
 }  
 }  
 return dp[m][n];  
 }  
};

## [1639. 通过给定词典构造目标字符串的方案数](https://leetcode-cn.com/problems/number-of-ways-to-form-a-target-string-given-a-dictionary/)

### 题目描述：



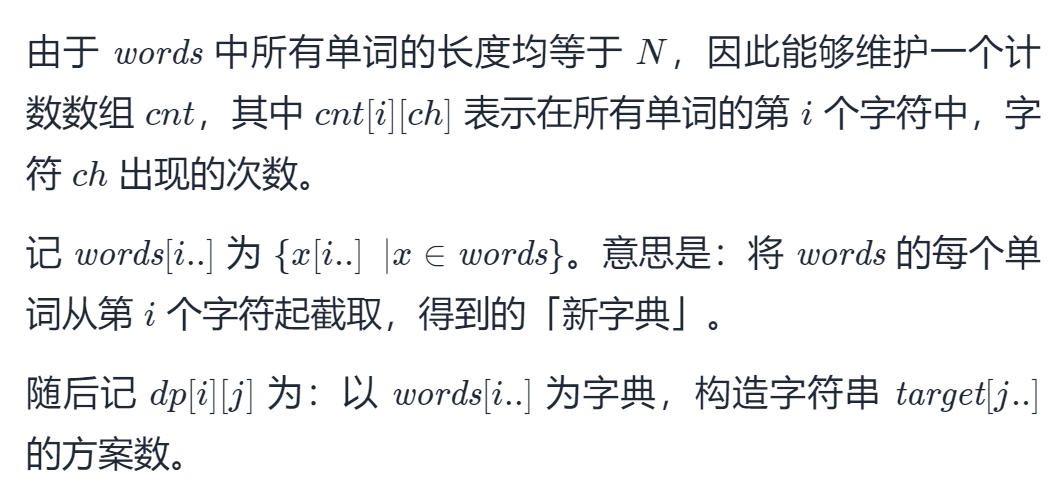


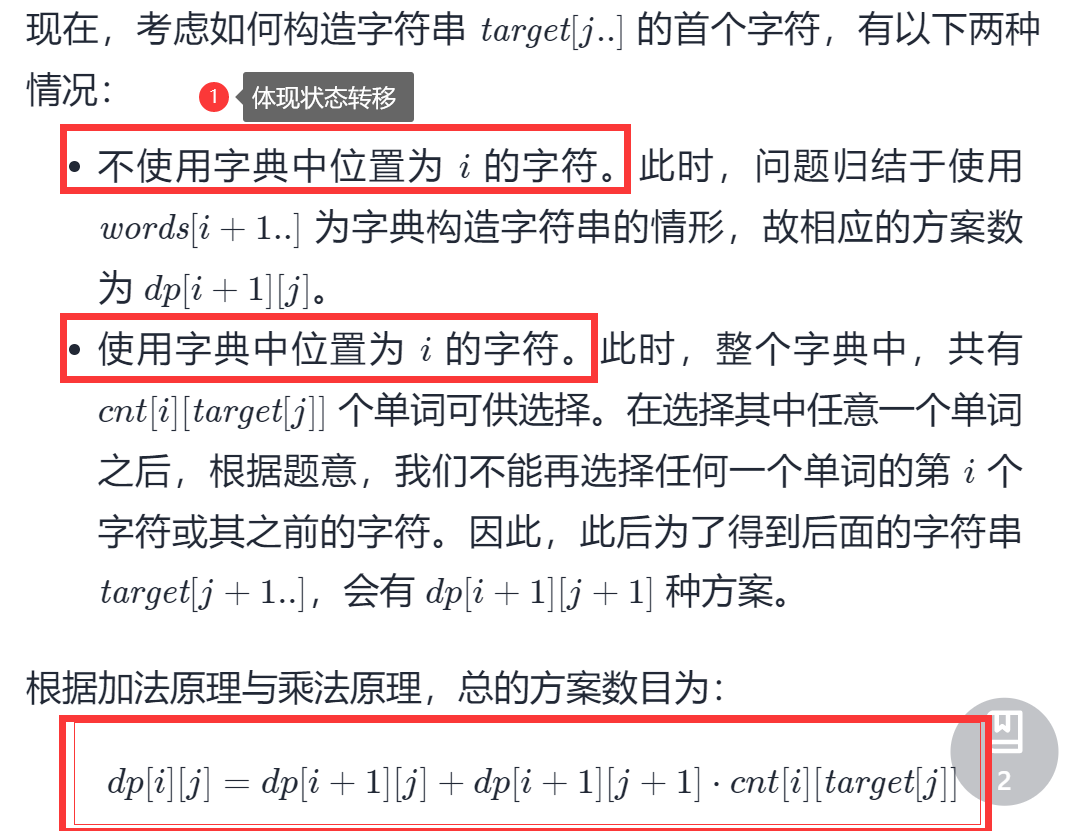
### 题目分析：(深度优先+记忆化)

1. **问题转换：构造一个“ 位置-字符-个数 ”的数组；表示所有word字符中位置i处字符c的个数；**
2. **这里有一个空间的优化：自己手动构建哈希函数，由于word.length<=1000,故pos<=1000;又小写字符个数为26个，故字符长度可设为26等；**
3. **这里我想的是深度优先搜索+记忆化**
4. **memo[i] [j] 表示的是从word[i...]开始到target[j...]字符匹配的个数**
5. **如何设计dfs函数：**
   * **返回类型表示的是memo[i] [j],存在直接返回，不存在求出而返回。**
   * **参数，word中的下标w,target中的下标t;表示两者的匹配**
   * **首先求出map[w] [target[t]].即word[w]与字符target[t]匹配的个数base；**
   * **然后分别枚举w+1，w+2，w+3...(此处w有范围)，与t+1匹配的个数，并求和（加法原理）**
   * **将上述所求和与base相乘（乘法原理）**
6. **特别注意边界条件：t==n-1时直接返回base；不允许t存在n的情况**
7. **存在除以mod的情况建议都用long long int**

class Solution {  
private:  
typedef long long int ll;  
 unordered\_map<int,unordered\_map<char,int>>map;  
 //memo表示word中位置i之后，target位置j之后可以匹配的示例  
 vector<vector<int>>memo;//记忆化矩阵  
 int ans,mod,m,n;  
  
 int dfs(int w,int t,string& target){  
 //if(m-w<n-t)return 0;//不可能完成匹配了  
 //if(t==n)return 1;//如何处理？  
 //每个t都有对应的一个最晚匹配字符  
 if(memo[w][t]!=-1)return memo[w][t];  
 ll res=map[w][target[t]],base=0;  
 if(t==n-1)return res;  
 if(res==0)return 0;//此处不能完成匹配  
 for(int i=w+1;i<=m-n+t+1;i++){  
 base+=dfs(i,t+1,target);  
 base%=mod;  
 }  
 res\*=base;  
 res%=mod;  
 memo[w][t]=res;  
 return res;  
 }  
public:  
 int numWays(vector<string>& words, string target) {  
 m=words[0].size(),n=target.size();  
 ans=0;mod=1e9+7;  
 memo.resize(m,vector<int>(n,-1));  
 //构建位置-字符-个数  
 for(auto&w:words){  
 for(int i=0;i<m;i++){  
 ++map[i][w[i]];  
 }  
 }  
 for(int i=0;i<=m-n;i++){  
 ans+=dfs(i,0,target);  
 ans%=mod;  
 }  
 return ans;  
  
 }  
};

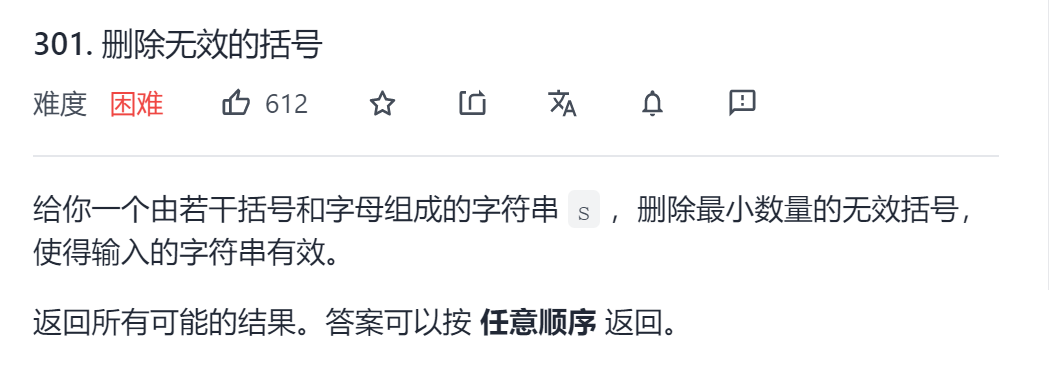
### 题目分析：动态规划（在记忆化的过程中体现状态的转移）

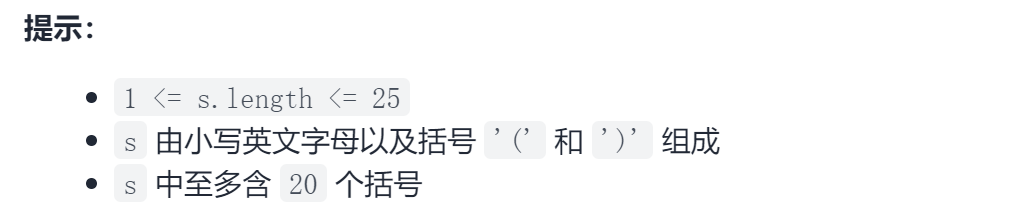




class Solution {  
public:  
 const int mod = 1e9+7;  
 long dfs(vector<vector<long>>& dp, vector<vector<int>>& cnt, string& target, int i, int j, int n, int m) {  
 if (j == m) return 1;  
 if (n - i < m - j) return 0;  
 if (dp[i][j] != -1) return dp[i][j];  
   
 long val = cnt[i][target[j] - 'a'] \* dfs(dp, cnt, target, i + 1, j + 1, n, m);  
 val += dfs(dp, cnt, target, i + 1, j, n, m);  
 val %= mod;  
 return dp[i][j] = val;;  
 }  
   
   
 int numWays(vector<string>& words, string target) {  
 int n = words[0].length();  
 vector<vector<int>> cnt(n, vector<int>(26, 0));  
 for (const auto& s: words) {  
 for (int i = 0; i < n; i++) {  
 cnt[i][s[i]-'a']++;  
 }  
 }  
   
 int m = target.length();  
 vector<vector<long>> dp(n, vector<long>(m, -1));  
 return dfs(dp, cnt, target, 0, 0, n, m);  
 }  
};  
  
作者：Arsenal-591  
链接：https://leetcode-cn.com/problems/number-of-ways-to-form-a-target-string-given-a-dictionary/solution/dong-tai-gui-hua-omn-jie-fa-by-arsenal-591/  
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。

## 301：删除无效括号





## 方法一：暴力枚举

### 题目分析：直接暴力枚举（超时）

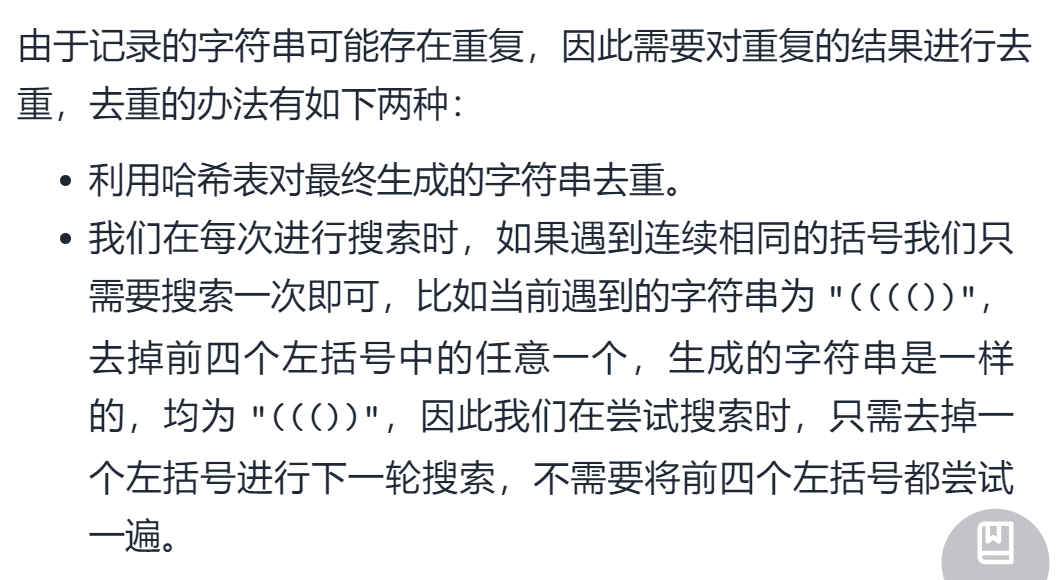
**一般来说，直接暴力枚举只适用于n<=15的情况，而此时的数据已经超过了25，当时间呈现处指数级增长时，必然会超时**

class Solution {  
private:  
 int dnum;//删除的最小数量  
 int lnum;//要删除的左括号数量  
 int rnum;//要删除的右括号的数量-用于剪枝  
 bool judge(string&s){  
 //判断是否属于有效字符串  
 stack<char>stk;  
 int n=s.size(),i=0;  
 while(i<n){  
 if(s[i]!='('&&s[i]!=')'){  
 //忽略小写字符  
 ++i;  
 continue;  
 }  
 if(stk.empty()||stk.top()==')'){  
 //此时不可能弹栈  
 stk.push(s[i]);  
 }else if(s[i]==')'){  
 stk.pop();//此时栈顶为(  
 }else stk.push(s[i]);//此时s[i]为(  
 ++i;  
 }  
 return stk.empty();  
 }  
public:  
 vector<string> removeInvalidParentheses(string s) {  
 //暴力枚举+剪枝条  
 //剪枝2：首先忽略掉所有字目，减小n值  
 //不可这样剪枝条，毕竟最后需要输出所有结果  
 //求删除的最小数量  
 int suml=0,sumr=0,sumc;//左右括号的总数以及字符的总数  
 stack<char>stk;  
 int n=s.size(),i=0;  
 while(i<n){  
 if(s[i]!='('&&s[i]!=')'){  
 //忽略小写字符  
 ++i;  
 continue;  
 }  
  
 if(s[i]=='(')++suml;  
 else if(s[i]==')')++sumr;  
 else ++sumc;  
  
 if(stk.empty()||stk.top()==')'){  
 //此时不可能弹栈  
 stk.push(s[i]);  
 }else if(s[i]==')'){  
 stk.pop();//此时栈顶为(  
 }else stk.push(s[i]);//此时s[i]为(  
 ++i;  
 }  
 //统计应该删除的左括号数量和右括号数量  
 dnum=stk.size();  
 while(!stk.empty()){  
 char c=stk.top();  
 if(c=='(')++lnum;  
 else ++rnum;  
 stk.pop();  
 }  
 //暴力枚举+剪枝条  
 //去重  
 unordered\_set<string>ans;  
 vector<string>res;  
 string temp;  
 //剪枝三：至多含20个括号  
 if(lnum>=20||rnum>=20)return res;  
  
 int ln=0,rn=0;//统计左括号和右括号的数量以及字符长度  
 for(int i=(1<<(n-dnum))-1;i<(1<<n);i++){//此处当测试数据的长度n=25时，数量级就相当庞大了，会导致超时  
 ln=rn=0;  
 temp.clear();  
 //剪枝1-从符合长度的最小值开始  
 for(int j=0;j<n;j++){  
 if((i>>j)&1){  
 temp.push\_back(s[j]);  
 if(s[j]=='(')++ln;  
 else if(s[j]==')')++rn;  
 }  
 }  
 //判断是否符合条件  
 if(temp.size()!=n-dnum)continue;//字符不可忽略  
 if(ln!=suml-lnum||rn!=sumr-rnum)continue;  
 if(judge(temp))ans.insert(temp);  
  
 }  
 for(auto s:ans)res.push\_back(s);  
 return res;  
  
 }  
};

### 暴力枚举优化（分成两个部分然后再合并两个部分）

class Solution {  
public:  
 bool checkValid(const string & str, int lmask, vector<int> & left, int rmask, vector<int> & right) {  
 int pos1 = 0;  
 int pos2 = 0;  
 int cnt = 0;  
  
 for (int i = 0; i < str.size(); i++) {  
 if (pos1 < left.size() && i == left[pos1]) {  
 if (!(lmask & (1 << pos1))) {  
 cnt++;  
 }  
 pos1++;  
 } else if (pos2 < right.size() && i == right[pos2]) {  
 if (!(rmask & (1 << pos2))) {  
 cnt--;  
 if (cnt < 0) {  
 return false;  
 }  
 }  
 pos2++;  
 }  
 }  
  
 return cnt == 0;  
 }  
  
 string recoverStr(const string & str, int lmask, vector<int> & left, int rmask, vector<int> & right){  
 string ans;  
 int pos1 = 0;  
 int pos2 = 0;  
  
 for (int i = 0; i < str.size(); i++) {  
 if (pos1 < left.size() && i == left[pos1]) {  
 if (!(lmask & (1 << pos1))){  
 ans.push\_back(str[i]);  
 }  
 pos1++;  
 } else if (pos2 < right.size() && i == right[pos2]) {  
 if (!(rmask & (1 << pos2))) {  
 ans.push\_back(str[i]);  
 }  
 pos2++;  
 } else {  
 ans.push\_back(str[i]);  
 }  
 }  
  
 return ans;  
 }  
  
 vector<string> removeInvalidParentheses(string s) {  
 int lremove = 0;//需要移除的左括号的数量  
 int rremove = 0;//需要移除的右括号的数量  
 vector<int> left;//所有左括号的下标位置集合  
 vector<int> right;//所有右括号的下标位置集合  
 vector<string> ans;//存储结果  
 unordered\_set<string> cnt;//哈希映射去重  
  
 //统计需要移除的左右括号的数量以及所有左右括号的下标集合  
 for (int i = 0; i < s.size(); i++) {  
 if (s[i] == '(') {  
 left.push\_back(i);  
 lremove++;  
 } else if (s[i] == ')') {  
 right.push\_back(i);  
 if (lremove == 0) {  
 rremove++;  
 } else {  
 lremove--;  
 }  
 }  
 }  
  
 int m = left.size();  
 int n = right.size();  
 vector<int> maskArr1;//所有符合删除左括号数量的子集  
 vector<int> maskArr2;//所有符合删除右括号数量的子集  
 for (int i = 0; i < (1 << m); i++) {  
 if (\_\_builtin\_popcount(i) != lremove) {  
 //\_builtin\_popcount(i)--->用于求i的二进制数中1的个数  
 continue;  
 }  
 maskArr1.push\_back(i);  
 }  
 for (int j = 0; j < (1 << n); j++) {  
 if (\_\_builtin\_popcount(j) != rremove) {  
 continue;  
 }  
 maskArr2.push\_back(j);  
 }  
   
 //对于mask1和mask2中的数逐一的枚举  
 for (auto mask1 : maskArr1) {  
 for (auto mask2 : maskArr2) {  
 if (checkValid(s, mask1, left, mask2, right)) {  
 //checkValid函数检查删除mask1和mask2中位置所在字符后是否时有效字符串  
 //如果是有效字符串，那么将其插入哈希集合  
 cnt.insert(recoverStr(s, mask1, left, mask2, right));  
 }  
 }  
 }  
 for (auto v : cnt) {  
 ans.emplace\_back(v);  
 }  
  
 return ans;  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/remove-invalid-parentheses/solution/shan-chu-wu-xiao-de-gua-hao-by-leetcode-9w8au/  
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。

## 方法二：深度优先搜索+回溯剪枝（万物皆可搜索）

* **统计需要删除的左括号lremove和右括号rremove**
* **由于n<=25再加上有效的剪枝，可以有比较好的效果**
* **在深度优先的时候去重有一定的技巧**
* 

class Solution {  
public:  
 vector<string> res;  
 vector<string> removeInvalidParentheses(string s) {  
 int lremove = 0;//需要删除的左括号数量  
 int rremove = 0;//需要删除的右括号数量  
  
 for (char c : s) {  
 if (c == '(') {  
 lremove++;  
 } else if (c == ')') {  
 if (lremove == 0) {  
 rremove++;  
 } else {  
 lremove--;  
 }  
 }  
 }  
 helper(s, 0, lremove, rremove);//深度优先搜索  
 return res;  
 }  
  
 void helper(string str, int start, int lremove, int rremove) {  
 if (lremove == 0 && rremove == 0) {  
 if (isValid(str)) {  
 res.push\_back(str);  
 }  
 return;  
 }  
 for (int i = start; i < str.size(); i++) {  
 //此处用于去重---学习一下  
 if (i != start && str[i] == str[i - 1]) {  
 continue;  
 }  
 // 如果剩余的字符无法满足去掉的数量要求，直接返回  
 if (lremove + rremove > str.size() - i) {  
 return;  
 }   
 // 尝试去掉一个左括号  
 if (lremove > 0 && str[i] == '(') {  
 helper(str.substr(0, i) + str.substr(i + 1), i, lremove - 1, rremove);  
 }  
 // 尝试去掉一个右括号  
 if (rremove > 0 && str[i] == ')') {  
 helper(str.substr(0, i) + str.substr(i + 1), i, lremove, rremove - 1);  
 }  
 }  
 }  
  
 inline bool isValid(const string & str) {  
 int cnt = 0;  
  
 for (int i = 0; i < str.size(); i++) {  
 if (str[i] == '(') {  
 cnt++;  
 } else if (str[i] == ')') {  
 cnt--;  
 if (cnt < 0) {  
 return false;  
 }  
 }  
 }  
  
 return cnt == 0;  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/remove-invalid-parentheses/solution/shan-chu-wu-xiao-de-gua-hao-by-leetcode-9w8au/  
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。

## 方法三：广度优先搜索

* **一般来说，可以深度优先搜索，就可以进行一个广度的优先搜索**
* **此处由于是求最小删除括号的数目，又是求所有符合的结果**
* **那么我们每次删除一个括号，并在此基础上继续删除其他括号，最后当出现符合条件的字符串时必然可以求出所有答案**

class Solution {  
public:  
 bool isValid(string str) {  
 int count = 0;  
  
 for (char c : str) {  
 if (c == '(') {  
 count++;  
 } else if (c == ')') {  
 count--;  
 if (count < 0) {  
 return false;  
 }  
 }  
 }  
  
 return count == 0;//判断字符是否时有效的字符串  
 }  
  
 vector<string> removeInvalidParentheses(string s) {  
 vector<string> ans;  
 unordered\_set<string> currSet;  
  
 currSet.insert(s);  
 while (true) {  
 for (auto & str : currSet) {  
 if (isValid(str))  
 ans.emplace\_back(str);  
 }  
 if (ans.size() > 0) {  
 return ans;  
 }  
 unordered\_set<string> nextSet;  
 for (auto & str : currSet) {  
 for (int i = 0; i < str.size(); i++) {  
 //此处用于去重  
 if (i > 0 && str[i] == str[i - 1]) {  
 continue;  
 }  
 if (str[i] == '(' || str[i] == ')') {  
 nextSet.insert(str.substr(0, i) + str.substr(i + 1, str.size()));  
 }  
 }  
 }  
 currSet = nextSet;  
 }  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/remove-invalid-parentheses/solution/shan-chu-wu-xiao-de-gua-hao-by-leetcode-9w8au/  
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。

### 题目总结：**万物皆可搜索！！！**

## [407. 接雨水 II](https://leetcode-cn.com/problems/trapping-rain-water-ii/)

1. **广度优先搜索专题：这道题的广度优先搜索比较特殊，是"包围式""缩小式"的搜索，利用木桶原理不断的分析掉最外的一层**

//代码学习  
typedef pair<int,int> pii;  
  
class Solution {  
public:  
 int trapRainWater(vector<vector<int>>& heightMap) {   
 if (heightMap.size() <= 2 || heightMap[0].size() <= 2) {  
 return 0;  
 }   
 int m = heightMap.size();  
 int n = heightMap[0].size();  
 priority\_queue<pii, vector<pii>, greater<pii>> pq;//优先队列的排序  
 vector<vector<bool>> visit(m, vector<bool>(n, false));  
 for (int i = 0; i < m; ++i) {  
 for (int j = 0; j < n; ++j) {  
 if (i == 0 || i == m - 1 || j == 0 || j == n - 1) {  
 pq.push({heightMap[i][j], i \* n + j});  
 visit[i][j] = true;  
 }  
 }  
 }  
  
 int res = 0;  
 int dirs[] = {-1, 0, 1, 0, -1};//利用循环向四周搜索  
 while (!pq.empty()) {  
 pii curr = pq.top();  
 pq.pop();   
 for (int k = 0; k < 4; ++k) {  
 int nx = curr.second / n + dirs[k];  
 int ny = curr.second % n + dirs[k + 1];  
 if( nx >= 0 && nx < m && ny >= 0 && ny < n && !visit[nx][ny]) {  
 if (heightMap[nx][ny] < curr.first) {  
 res += curr.first - heightMap[nx][ny];   
 }  
 visit[nx][ny] = true;  
 pq.push({max(heightMap[nx][ny], curr.first), nx \* n + ny});  
 }  
 }  
 }  
   
 return res;  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/trapping-rain-water-ii/solution/jie-yu-shui-ii-by-leetcode-solution-vlj3/  
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。