[**几道有意思的DP题**](http://www.cppblog.com/guyuecanhui/articles/76380.html)

POJ 1243 One Person  
这是一道蛮有意思的题目，想到了就觉得很简单，下面给出思路：  
    1.先考虑L=0的情况，由于一次都不能猜错，因此只能从1开始，逐个向上猜，总数不能超过G；  
    2.若G<=L，等价于G=L的情况，因为此时允许猜的次数限制了最大的数，总数为2^G  
    3.一般的情况，比如L=1时，可以设想，只能猜错一次的话，这个数不能太大，否则猜错了一点作用也没有，因此要确保即使本次猜错也能在以后的G-1次猜中，这就转化成了(G-1,0)的情况，也就是说必胜的策略应当为先猜G，如果错，则从1~G-1逐次的猜，如果猜对了就转化成了(G-1,1)的情况，他应当再猜G+G-1……对于G=i,L=j的情况也是一样的考虑，可以列出状态转移方程：  
opt[i][0]= i;  
opt[i][i]= 2^i;  
opt[i][j]= opt[i-1][j]+ opt[i-1][j-1]+ 1;  
    搞定！

**POJ 3132 Sum of Different Primes**  
      这题思路很清晰，从小到大考虑每个素数与已经有了分解的数相加，列出个式子感觉就能搞定：  
      比如：   2            3                 5                   7                 ...  
                                2+3             2+5               2+7             ...  
                                                   3+5               3+7             ...  
                                                   2+3+5           5+7              ...  
                                                                        2+3+7          ...  
                                                                        2+5+7          ...  
                                                                        3+5+7          ...  
                                                                        2+3+5+7      ...  
                                                                                            ...  
      可是实现起来发现不是那么简单，想到不能从前往后加，因为可能刚生成的opt[i][j]在后面被重复计算，只有从后向前算。假定在素数prime[i]之前的分解数都已经计算正确，那么每个opt[j][k]+= opt[j-prime[i]][k-1];  
      列出式子如下（为了处理边界且不至于重复计算，把opt[0][0]设为1）：  
 opt[0][0]= 1;  
 for( i= 0; i< n; i++ )  
   for( j= 1120; j>= p[i]; j-- )  
      for( k= 14; k>= 1; k-- )  
         opt[j][k]+= opt[j-p[i]][k-1];  
      搞定！  
  
**POJ 1548 Robots**  
      做这个题第一感觉就是有点像LCS的程序，关键在于看出最大机器人数是由行数较小，列数较大的路径上最多机器人数决定，即类似于以下这种情况：  
                     \*\*\*\*\*\*\*\*\*\*\*G\*\*  
                     \*\*\*\*\*\*\*G\*\*\*\*\*\*  
                     \*\*\*\*\*G\*\*\*\*\*\*\*\*  
                     \*\*\*G\*\*\*\*\*\*\*\*\*\*  
                     G\*\*\*\*\*\*\*\*\*\*\*\*\*  
      做的时候我用的是动态规划，opt[i][j]表示从左下角计i\*j的方阵内的最多机器人数，则:  
      opt[i][j]=Max(opt[i+1][j],opt[i][j-1],opt[i+1][j-1]+ga[i+1][j-1]);   //ga[i][j]=1表示此处有G，为0没有  
      最终只要输出opt[0][ymax+1];   //ymax表示输入中最大的列号  
      搞定！  
  
**POJ 1850 Code**        这题应该属于计数的问题，但可以使用DP简化过程。主要的思路是用a[i][j]表示以i开头的长度为j的序列有多少个，其中i=a~z，映射为1~26。则列出前几项可以很快找到递推式：  
        a[i][1]= 1;      a[i][j]= a[k][j-1], k=i~26;  
        接下来可以用sum[i][j]来预先计算在以i开头的长度为j的序列之前共有多少个序列，则最终计算结果为：

http://www.cppblog.com/Images/OutliningIndicators/None.gifif( num[0]=='a' )    total= sum[26][n-1];  
http://www.cppblog.com/Images/OutliningIndicators/None.gif        else    total= sum[num[0]-97][n];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif        for( i= 1; i< n; i++ ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            for( j= num[i-1]-95; j< num[i]-96; j++ )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                total+= a[j][n-i];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif        }

**POJ 2033 Alphacode**  
         又是一个简单的一维DP，关键是对0的处理。  
         从后往前处理，opt[i]表示字符串第i位到最后一位有多少种解码方式。  
               如果当前位为0，则opt[i]= 0;  
               如果上一位为0，则opt[i]= opt[i+2];  
               如果当前位与下一位能表示26内的数，则opt[i]= opt[i+1]+ opt[i+2];  
               否则opt[i]= opt[i+1];  
         Done!

http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif        for( i= n-2; i>= 0; i-- ){  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif            if( s[i]=='0' ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                opt[i]= 0;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                continue;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif            }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            else if( s[i+1]=='0' )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                opt[i]= opt[i+2];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif            else{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                opt[i]= opt[i+1];  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                if( s[i]=='1'|| s[i]=='2'&&s[i+1]<='6' )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    opt[i]+= opt[i+2];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif            }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/None.gif

**POJ 1692 Crossed Matchings**  
         一道类似于最长公共子序列的问题，略有变化，但思想是一样的。这题用到了两次DP，先来说下思路：  
         用opt[i][j]来表示第一个字符串S1前i位和第二个字符串S2前j位可能达到的最大匹配对数，则对于opt[i][j]有三种情况：一是无法匹配，二是可以匹配，但是匹配后总数减少或不变，即它打破了之前的匹配情况，此时，opt[i][j]= Max(opt[i][j-1],opt[i-1][j])，三是S1[i]S2[j]可以与S2[q]S1[p]匹配且总数增加，此时，opt[i][j]=opt[p-1][q-1]+2。而pq的取得要用到第二次DP，以S1为例，用ma[i][j]表示S1[i]与S2前j个字符中相同的最大位置，则列出状态方程：ma[i][j]=j-1，S1[i]==S2[j-1]；ma[i][j]=ma[i][j-1]，S1[i]!=S2[j-1]；  
         下面是主要的程序：

http://www.cppblog.com/Images/OutliningIndicators/None.gif        for( i= 1; i<= m; i++ )  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif            for( j= 2; j<= n; j++ ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                if( a[i]== b[j-1] )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    ma[i][j]= j-1;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                else  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    ma[i][j]= ma[i][j-1];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif            }  
http://www.cppblog.com/Images/OutliningIndicators/None.gif        for( i= 1; i<= n; i++ )  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif            for( j= 1; j<= m; j++ ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                if( b[i]== a[j-1] )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    mb[i][j]= j-1;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                else  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    mb[i][j]= mb[i][j-1];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif            }  
http://www.cppblog.com/Images/OutliningIndicators/None.gif        for( i= 2; i<= m; i++ )  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif            for( j= 2; j<= n; j++ ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                opt[i][j]= Max( opt[i-1][j], opt[i][j-1] );  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                p= mb[j][i]-1;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                q= ma[i][j]-1;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                if( p>=0 && q>=0 && a[i]!= b[j] && opt[p][q]+2> opt[i][j] )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    opt[i][j]= opt[p][q]+2;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif            }

**POJ 1141 Brackets Sequence**      这题很类似于矩阵连乘的问题，输出有点麻烦，还有一个空行的问题也很讨厌。  
      设opt[i][j]表示字符串i~j之间要加入的字符数，tag[i][j]记录取最小值的位置信息，初始的处理：  
         tag全置为-1  
         若i>j，opt[i][j]= 0  
         若i==j，opt[i][j]= 1  
         若i>j，opt[i][j]= 200  
      对于i> j的情况：若s[i]与s[j]组成一对括号，则opt[i][j]= opt[i+1][j-1]  
                              否则对于i<=k<=j中，找到最小的opt[i][k]+opt[k+1][j]与opt[i][j]比，如果较小，则替换，并把tag[i][j]置为k  
      输出用递归也很简单：Print(i,j)表示输出i~j之间的字符，则如果tag[i][j]==-1，说明取最小值时两边为匹配的括号，此时应当输出{Print(i+1,j-1)}，'{'表示'('或'['；如果tag>=0，则说明最小值为拆开的两部分之和，此时应当输出Print(i,tag[i][j]),Print(tag[i][j+1],j)，递归边界：i==j，则分情况输出{}。  
      还要注意的是有空行的情况，我把加入到Print()中处理的，直接返回即可，不用输出回车。  
      下面是DP部分的代码：

http://www.cppblog.com/Images/OutliningIndicators/None.gif    for( j= 1; j< n; j++ )  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif        for( i= j-1; i>= 0; i-- ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            opt[i][j]= 200;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if( Match(s[i],s[j]) )  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                opt[i][j]= opt[i+1][j-1];  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            for( k= i; k<= j; k++ )  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif                if( opt[i][j]> opt[i][k]+opt[k+1][j] ){  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    tag[i][j]= k;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                    opt[i][j]= opt[i][k]+opt[k+1][j];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif                }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif        }