第5章：动态规划

动态规划的基本思想

- 动态规划主要用于组合优化问题，即求一个离散问题在某种意义下的最优解，有时也用于组合计数问题。
- 如果各个子问题不是独立的，不同的子问题的个数只是多项式量级，如果我们能够保存已经解决的子问题的答案，而在需要的时候再找出已求得的答案，这样就可以避免大量的重复计算。
- 由此而来的基本思路是，用一个表记录所有已解决的子问题的答案，不管该问题以后是否被用到，只要它被计算过，就将其结果填入表中。

动态规划的基本要素

- 那么，什么样的问题适合用动态规划求解呢？
- 适合用动态规划求解的问题的两个基本要素：
- (1) 最优子结构
  - 问题的最优解包含其子问题的最优解。

动态规划的基本要素

- (2)子问题重叠
  - 动态规划所针对的问题还有另外一个显著的特征，即它所对应的子问题呈现大量的重复，称为**子问题重叠性质**。
  - 在应用动态规划时，对于重复出现的子问题，只需在第一次遇到时加以求解，并把答案保存起来，以便以后再遇到时直接引用，不必重新求解，从而大大地提高解题的效率。
  - 相比之下，一般的搜索技术，对于某个子问题，不管是否已经求解过，只要遇上，就会再次对它求解，因而影响了解题的效率。

实例一、数字三角形问题

- 1. 问题描述
  - 给定一个具有N层的数字三角形，从顶至底有多条路径，每一步可沿左斜线向下或沿右斜线向下，路径所经过的数字之和为路径得分，请求出最小路径得分。

```
         2
      6     2
   1     8     4
1     5     6     8
```

数字三角形

用暴力的方法，可以吗？

- 这道题如果用枚举法（暴力思想），在数塔层数稍大的情况下（如31），则需要列举出的路径条数将是一个非常庞大的数目（2^30= 1024^3 > 10^9=10亿）。

- 2．解题思路
  - 这道题可以用动态规划成功地解决，但是，如果对问题的最优结构刻画得不恰当(即状态表示不合适)，则无法使用动态规划。
- 状态表示法一：
  - 用一元组D(X)描述问题，D(X)表示从顶层到达第X层的最小路径得分。因此，此问题就是求出D(N)(若需要，还应求出最优路径)。这是一种很自然的想法和表示方法。遗憾的是，这种描述方式并不能满足最优子结构性质。因为D(X)的最优解(即最优路径)可能不包含子问题例如D(X-1)的最优解。如图所示：

2

6　2

1　8　4

1　5　6　8

数字三角形

显然，D(4)＝2+6+1+1＝10，其最优解(路径)为2-6-1-1。而D(3)＝2+2+4＝8，最优解(路径)为2-2-4。故D(4)的最优解不包含子问题D(3)的最优解。由于不满足最优子结构性质，因而无法建立子问题最优值之间的递归关系，也即无法使用动态规划。

- 状态表示法二：
  - 用二元组D(X,y)描述问题，D(X,y)表示从顶层到达第X层第y个位置的最小路径得分。
  - 最优子结构性质：容易看出，D(X,y)的最优路径Path(X,y)一定包含子问题D(X-1,y)或D(X-1,y-1)的最优路径。
  - 否则，取D(X-1,y)和D(X-1,y-1)的最优路径中得分小的那条路径加上第X层第y个位置构成的路径得分必然小于Path(X,y)的得分，这与Path(X,y)的最优性是矛盾的。

2

6 2

1 8 4

1 5 6 8

数字三角形

- 如图所示，D(4,2)的最优路径为2-6-1-5，它包含D(3,1)最优路径2-6-1。因此，用二元组D(X,y)描述的计算D(X,y)的问题具有最优子结构性质。

- 递归关系：

$$
\begin{cases}
D(X,y)=\min\{D(X-1,y),\ D(X-1,y-1)\}+a(X,y) \\
D(1,1)=a(1,1)
\end{cases}
$$

- 其中，a(X,y)为第X层第y个位置的数值。
- 原问题的最小路径得分可以通过比较D(N,i)获得，其中i＝1，2，…，N。
- 在上述递归关系中，求D(X,y)的时候，先计算D(X-1,y)和D(X-1,y-1)，下一步求D(X，y+1)时需要D(X-1，y+1)和D(X-1，y)，但其中D(X-1，y)在前面已经计算过了。于是，子问题重叠性质成立。
- 因此，采用状态表示法二描述的问题具备了用动态规划求解的基本要素，可以用动态规划进行求解。

- 状态表示法三：
  - 采用状态表示法二的方法是从顶层开始，逐步向下至底层来求出原问题的解。事实上，还可以从相反的方向考虑。仍用二元组D(X,y)描述问题，D(X,y)表示从第X层第y个位置到达底层的最小路径得分。原问题的最小路径得分即为D(1,1)。
  - 最优子结构性质：显然，D(X,y)的最优路径Path(X,y)一定包含子问题D(X+1,y)或D(X+1,y+1)的最优路径，否则，取D(X+1,y)和D(X+1,y+1)的最优路径中得分小的那条路径加上第X层第y个位置构成的路径得分必然小于Path(X,y)的得分，这与Path(X,y)的最优性矛盾。

2

6 2

1 8 4

1 5 6 8

数字三角形

- 如图所示，D(1，1)的最优路径为2-6-1-1，它包含D(2,1)的最优路径6-1-1。因此，这种状态表示描述的计算D(X，y)的问题同样具有最优子结构性质。

- 递归关系：
- $\begin{cases} D(X,y)=\min\{D(X+1,y),D(X+1,y+1)\}+a(X,y) \\ D(N,k)=a(N,k)，k=1，\ldots，N \end{cases}$

- 其中，a(X，y)为第X层第y个位置的数值。
- D(X,y)表示从第X层第y个位置到达底层的最小路径得分。原问题的最小路径得分即为D(1,1)。

结论：自顶向下的分析，自底向上的计算。

算法设计

采用状态表示法三的算法的主要过程如下：

```
for ( i = n - 2; i >= 0; --i )
{
    for ( j = 0; j <= i; ++j )
    {
            tmp = d[i + 1][j];
            if ( d[i + 1][j + 1] < tmp )
            {
                    tmp = d[i + 1][j + 1];
            }
            d[i][j] += tmp;
    }
}
printf( "%d\n", d[0][0] );
```

动态规划算法步骤

- (1)选择适当的问题状态表示，并分析最优解的性质；
- (2)递归地定义最优值(即建立递归关系)；
- (3)以自底向上的方式计算出最优值；
- (4)根据计算最优值时得到的信息，构造一个最优解。

- 步骤(1)～(3)是动态规划的基本步骤。在只需要求出最优值的情形，步骤(4)可以省略。
- 若需要求出问题的一个最优解，则必须执行步骤(4)。此时，在步骤(3)中计算最优值时，通常需记录更多的信息，以便在步骤(4)中，根据所记录的信息，快速地构造出一个最优解。
- 注意事项：
  - 在进一步探讨动态规划设计方法及应用之前，有两点需要注意：

- (1)问题的状态表示对能否用动态规划进行求解是至关重要的，不恰当的状态表示将使问题的描述不具有最优子结构性质，从而无法建立最优值的递归关系，动态规划的应用也就无从谈起。因此，上面步骤(1)，即状态表示和最优子结构性质的分析，是最关键的一步。
- (2)在算法的程序设计中，应充分利用子问题重叠性质来提高解题效率。更具体地说，应采用递推(迭代)的方法来编程计算由递归式定义的最优值，而不采用直接递归的方法。

实例二、编辑距离问题

- 1. 问题描述
  - 设A和B是2个字符串。要用最少的字符操作，将字符串A转换为字符串B。这里所说的字符操作包括：
  - (1)删除一个字符；
  - (2)插入一个字符；
  - (3)将一个字符改为另一个字符。
  - 将字符串A转换为字符串B所用的最少字符操作数称为字符串A到B的编辑距离，记为$\delta(A,B)$。请求出$\delta(A,B)$。

2. 解题思路

- 设所给的2个字符串分别为A＝A[1..m]和B=B[1..n]。
- 状态表示：考虑从字符子串A[1..i](按序)变换到字符子串B[1..j]的最少字符操作问题，有d(i，j)＝δ(A[1..i]，B[1..j])。
- 显然，2个单字符a，b之间的编辑距离:当a≠b时，为δ(a，b)＝1；当a=b时，为δ(a，b)＝0。
- 问题的解变为求：d(m，n)。

2. 解题思路

- 最优子结构性质：假设E＝$e_1 \ldots e_{k-1} e_k$，k=d(i,j)为从字符串A[1..i]按序变换到字符串B[1..j]的一个最少字符操作序列(也称作d(i,j)的一个最优解)，那么最后一个操作e_k，属于下列3种操作之一：

2. 解题思路

- (1)将字符A[i]改为字符B[j](如果A[i]＝B[j]，则e_k为空操作,不参加计数)，此时E_1＝$e_1 \ldots e_{k-1}$为d(i-1，j-1)的一个最优解;
- (2)删除字符A[i],此时E_1＝$e_1 \ldots e_{k-1}$为d(i-1,j)的一个最优解;
- (3)插入字符B[j],此时E_1＝$e_1 \ldots e_{k-1}$为d(i,j-1)的一个最优解。

2. 解题思路

- 因此，问题具有最优子结构性质。根据最优子结构性质，建立递归关系如下：
- $d(i,j) = \min\{d(i-1,j-1)+\delta(A[i],B[j]), d(i-1,j)+1, d(i,j-1)+1\}$
- 初始条件：$d(i,0)=i$，$(i=0\sim m)$；
  $d(0,j)=j$，$(j=0\sim n)$。
- 问题的解为$d(m, n)$。

实例三、最长公共子序列问题

- 1. 问题描述
- 一个给定序列的子序列是在该序列中删去若干元素后得到的序列。
- 给定2个序列X和Y，当另一序列Z既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列。
- 公共子序列中长度最长的公共子序列叫做最长公共子序列。
- 最长公共子序列(LCS)问题可以叙述为：给定2个序列X={x1，...，xm}和Y={y1，...，yn}，要求找出X和Y的一个最长公共子序列的长度。

Sample Input

abcfbc abfcab
programming contest
abcd mnp

Sample Output

4
2
0

- 2．解题思路
  - 解最长公共子序列问题时最容易想到的算法是穷举搜索法，即对X的每一个子序列，检查它是否也是Y的子序列，从而确定它是否为X和Y的公共子序列，并且在检查过程中遴选出最长的公共子序列。X的所有子序列都检查过后即可求出X和Y的最长公共子序列。
  - X的一个子序列相应于下标序列{1，2，…，m}的一个子序列，故X共有2^m个不同子序列，穷举搜索法需要指数时间。为此，考虑能否用动态规划方法求解。

（一）状态表示：

- 用C[i,j]记录序列X(i)和Y(j)的最长公共子序列的长度，其中X(i)={x1，…，xi}，Y(j)={y1，…，yj }。

- 原问题最优解的长度为C[m,n]。

（二）最优子结构性质：

- 设序列**X={x1，…，xm}**和**Y={y1，…，yn}**的一个最长公共子序列为**Z={z1，…，zk}**。则下述结论成立：
  - (1)若xm = yn，则zk = xm = yn 且Z(k-1) = {z1，…，z_{k-1}}是X(m-1)和Y(n-1)的最长公共子序列。
  - (2)若xm ≠ yn 且 zk ≠ xm，则Z是X(m-1)和Y的最长公共子序列。
  - (3)若xm ≠ yn 且 zk ≠ yn，则Z是X和Y(n-1)的最长公共子序列。

- 证明：
  - (1)用反证法。若zk ≠ xm，则{z1，…，zk，xm}是X和Y的长度为k+1的公共子序列。这与Z是X和Y的一个最长公共子序列矛盾。因此，必有zk = xm = yn。由此可知Z(k-1)={z1，…，z_{k-1}}是X(m-1)和Y(n-1)的一个长度为k-1的公共子序列。若X(m-1)和Y(n-1)有一个长度大于k-1的公共子序列w，则将xm加在其尾部将产生X和Y的一个长度大于K的公共子序列。此为矛盾。故
  - Z(k-1) = {z1，…，z_{k-1}}是X(m-1)和Y(n-1)的一个最长公共子序列。

- (2)由于$z_k \neq x_m$，Z是X(m-1)和Y的一个公共子序列。若X(m-1)和Y有一个长度大于k的公共子序列w，则w也是X和Y的一个长度大于k的公共子序列。这与Z是X和y的一个最长公共子序列矛盾。由此可知，Z是X(m-1)和Y的最长公共子序列。
- (3)与(2)类似。
- 上述结论：**2**个序列的最长公共子序列包含了这**2**个序列前缀的最长公共子序列。因此，最长公共子序列问题具有最优子结构性质。

- 由最长公共子序列问题的最优子结构性质可知，要找出X={x1，…，xm}和Y={y1，…，yn}的一个最长公共子序列，可按以下方式递归地进行：
- 当 xm = yn时，找出X(m-1)和Y(n-1)的最长公共子序列，然后在其尾部加上xm(=yn)即可得X和Y的一个最长公共子序列。
- 当xm ≠ yn时，必须解2个子问题，即找出X(m-1)和Y的一个最长公共子序列及X和Y(n-1)的一个最长公共子序列。这2个公共子序列中较长者即为X和Y的一个最长公共子序列。

- 我们来建立子问题的最优值C[i,j]的递归关系。当i=0或j=0时，空序列是X(i)和Y(j)的最长公共子序列，故C[i,j]＝0。其他情况下，由最优子结构性质，可建立递归方程如下：

$$C[i,j]=\begin{cases} 0 & i=0 \text{ 或} j=0; \\ C[i-1,j-1]+1 & i,j>0 \text{ 且} x_i=y_j; \\ \max\{C[i-1,j],C[i,j-1]\} & i,j>0 \text{ 且} x_i \neq y_j; \end{cases}$$

由此递归方程容易看到最长公共子序列问题具有子问题重叠性质。

实例四、最少硬币找钱问题

- 1. 问题描述
- 设有n种不同面值的硬币，各硬币的面值存于数组T[1..n]中。现要用这些面值的硬币来找钱。可以使用的各种面值的硬币个数不限。请计算找出钱数 L 的最少硬币个数。
- 设0<T[1]<T[2]<…<T[n]。当只用这些面值的硬币找不出钱数j时，记∞

- 设C[j]，j=1，…，L，表示用T[1..n]的硬币可找出钱数j的最少硬币个数。最优子结构性质：
- 设S[k]，k=1, 2, …, n 是C[j]的一个最优找钱序列, 即

$$j = \boxed{\sum_{k=1}^{n} S[k]T[k]}, \quad \text{而且} \quad \boxed{\sum_{k=1}^{n} S[k] = C[j]}$$

例如：**T1=1,T2=5,T3=11**，找**19**块钱：**C[19]=4**，

S1=2,S2=1,S3=1

j=19=T1*S1+T2*S2+T3*S3

S1+S2+S3=C[19]

- 假设对某个i,S[i]>0，即最优找钱序列至少有一个T[i]，则在该最优找钱序列中去掉一个T[i]后的找钱序列应是用用T[1..n]的硬币可找出钱数j-T[i]的最优找钱序列，其个数为C[j-T[i]]。故计算C[j]的问题具有最优子结构性质。
- 根据最优子结构性质，对于任何1≤j≤L及1≤i≤n，若j-T[i]>0，则C[j-T[i]]所表示的找钱j-T[i]的最优找钱序列，再加上一枚面值为T[i]的硬币是一种找钱j的方法，且所用的硬币个数为C[j-T[i]]+1。因此，可建立如下的递归关系：

递归关系和初始条件:

$$C[j]=\begin{cases} 0, & j=0 \\ \infty, & 0<j<T[1] \\ 1, & j=T[1] \\ \min\limits_{1\le i\le n,\, j-T[i]\ge 0}\{C[j-T[i]]+1\}, & j>T[1] \end{cases}$$

- 问题的解:找出钱数 L 的最少硬币个数为C[L]。

还记得这题吗？

- 假设：钱币面值为1元、5元、11元，
  - 找15元？

$$
C[15]=\begin{cases} C[4]+1 \\ C[10]+1 \\ C[14]+1 \end{cases}
$$

$$C[4]=4$$

$$C[10]=2\begin{cases} C[5]+1=2 \\ C[9]+1=6 \end{cases}$$

$$C[5]=1\begin{cases} 1 \\ C[4]+1=5 \end{cases}$$

$$C[9]=5\begin{cases} C[4]+1=5 \\ C[8]+1=5 \end{cases}$$

$$C[8]=4\begin{cases} C[3]+1=4 \\ C[7]+1=\min(C[2]+1,C[6]+1)+1 \end{cases}$$

结果

- 假设：钱币面值为1元、5元、11元，
  - 找15元？

$$C[15]=\begin{cases} C[4]+1=4+1=5 \\ C[10]+1=2+1=3 \\ C[14]+1=4+1=5 \end{cases}$$