第四章　动态规划

学习要点:

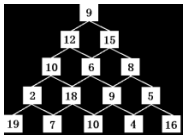
- ✓ 理解动态规划算法的概念。
- ✓ 掌握动态规划算法的基本要素
- ✓ 掌握设计动态规划算法的步骤。
- ✓ 学习动态规划算法设计实例

4.1 动态规划

- 动态规划的方法在工程技术、企业管理、工农业生产及军事等部门中都有广泛的应用, 并且取得了显著的效果。
- 在企业管理方面, 动态规划可以用来解决最优途径问题、资源分配问题、生产调度问题、库存问题、装载问题、排序问题、设备更新问题, 等等, 所以它是现代企业管理中的一种重要决策方法。

数塔问题

■ 有形如右图的一个数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一直走到底层，要求找出一条路径，使路径上的数值和最大。



■ 问题分析

9
12 15
10 6 8
2 18 9 5
19 7 10 4 16

■ 口算结果？ 9->12->10->18->10

贪心算法

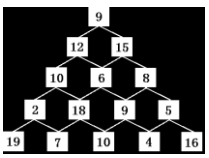
贪心策略？

■ 是否满足贪婪选择性质？

■ 是否满足最优子结构性质？

9
12 15
10 6 8
2 18 9 5
19 7 10 4 16

枚　举



■ 最保险的思路，列举出所有可能的路径再比较，得出和最大的路径。

■ 重复工作：循环、递归。

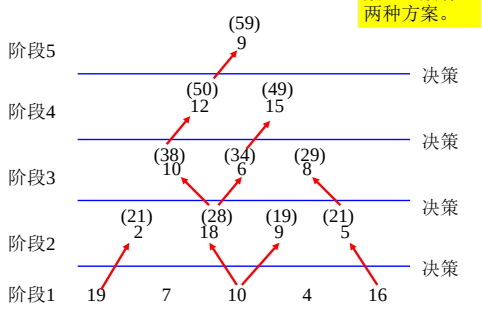
■ 如何循环？

■ 递归如何？

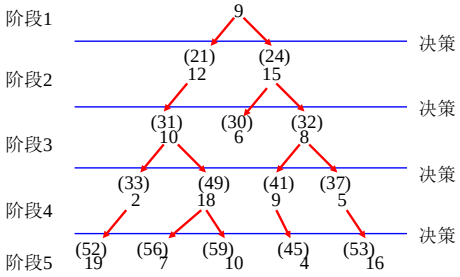
- 递归边界条件；
- 使问题向边界条件转化的规则(递归定义)。

动态规划的手工计算（逆序）

取第*i*行第*j*个数，一般有两种方案。



动态规划的手工计算（顺序）



动态规划的手工计算

- 顺序与逆序解法本质上无区别；
- 一般当初始状态唯一给定时可用逆序解法；
- 如需比较到达不同终点状态的各个路径及最大结果时，使用顺序法比较简便；
- 如需知道塔中每一点到最下层的最大值和路径时，使用逆序法比较简便。

动态规划的算法实现

逆序法

```
          59
       50   49
    38   34   29
  21   28   19   21
19   7   10   4   16
```

- 原始信息存储
  - 层数用整型变量*n*存储；
  - 数塔中的数据用二维数组data[ ][ ]存储，下三角阵。
- 动态规划过程存储
  - 必须用二维数组d[ ][ ]存储各阶段的决策结果。二维数组d的存储内容如下：
  - d[n][j]=data[n][j]，其中j=1,2,……,n；
  - d[i][j]=max(d[i+1][j]，d[i+1][j+1])+data[i][j]，其中i=n-1,n-2,……1，j=1,2,……,i；
- 最后d[1][1]存储的就是问题的最大值。
- 可以通过分析d，得到路径。

动态规划的算法实现

数组data

```
9
12  15
10  6   8
 2  18  9  5
19  7  10  4  16
```

数组d

```
59
50  49
38  34  29
21  28  19  21
19   7  10   4  16
```

- 输出data[1][1]“9”；
- b=d[1][1]-data[1][1]=59-9=50　b与d[2][1],d[2][2] 比较b与d[2][1]相等，输出data[2][1]“12”；
- b=d[2][1]-data[2][1]=50-12=38　b与d[3][1],d[3][2] 比较b与d[3][1]相等，输出data[3][1]“10”；
- b=a[3][1]-data[3][1]=38-10=28　b与d[4][1],d[4][2] 比较b与d[4][2]相等，输出data[4][2]“18”；
- b=d[4][2]-data[4][2]=28-18=10　b与d[5][2],d[5][3] 比较b与d[5][3]相等，输出data[5][3]“10”。

动态规划的算法实现

- 为了设计简洁的算法，可以用三维数组a[50][50][3]存储以上确定的三个数组的信息。
  - a[50][50][1]代替数组data，
  - a[50][50][2]代替数组d，
  - a[50][50][3]记录解路径。

```
for (i=n-1 ; i>=1;i--)
  for (j=1 ;j>= i ;j++)
    if (a[i+1][j][2]>a[i+1][j+1][2]) {
       a[i][j][2]=a[i][j][2]+a[i+1][j][2];
       a[i][j][3]=0;}
     else {
       a[i][j][2]=a[i][j][2]+a[i+1][j+1][2];
       a[i][j][3]=1;}
print('max=',a[1][1][2]);
```

```
for( i=1 ;i<=n;i++)
   for (j=1;j<=i;j++){
      input(a[i][j][1]);
      a[i][j][2]=a[i][j][1];
      a[i][j][3]=0;}
```

```
for( i=1 ;i<= n-1;i++){
   print(a[i][j][1],'->');
   j=j+a[i][j][3];
}
print (a[n][j][1]);
```

4.2 动态规划的思想和概念

- 动态规划的基本思想
  - 动态规划方法的基本思想是，把求解的问题分成许多阶段或多个子问题，然后按顺序求解各子问题。最后一个子问题就是初始问题的解。
    - 由于动态规划的问题有重叠子问题的特点，为了减少重复计算，对每一个子问题只解一次，将其不同阶段的不同状态保存在一个二维数组中。
- **动态规划=贪婪策略+递推(降阶)+存储递推结果**

动态决策问题的特点：

系统所处的状态和时刻是进行决策的重要因素；即在系统发展的不同时刻（或阶段）根据系统所处的状态，不断地做出决策；找到不同时刻的最优决策以及整个过程的最优策略。

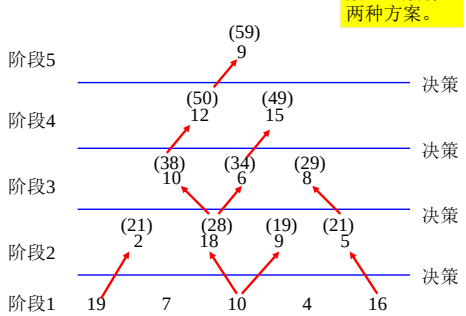
多阶段决策问题：

在多阶段决策过程中，系统的动态过程可以按照时间进程分为状态相互联系而又相互区别的各个阶段；

每个阶段都要进行决策，目的是使整个过程的决策达到最优效果。

动态规划的手工计算（逆序）

取第*i*行第*j*个数，一般有两种方案。

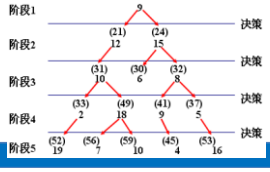


空间换取时间

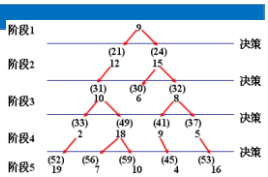
- 递归算法效率低的主要原因是因为进行了大量的重复计算。而动态规划的基本动机就是充分利用重叠子问题(Overlapping subproblems)。
- 因为动态规划将以前（子问题）计算过的结果都记录下来，遇到使用子问题结果的时候只需查表。
- 动态规划是一种用空间换取时间的方法。
- 因此，动态规划常常因为空间消耗太大而难以实现。

动态规划的主要概念

- 阶段：把问题分成几个相互联系的有顺序的几个环节，这些环节即称为阶段。
- 状态：某一阶段的出发位置称为状态。通俗地说状态是对问题在某一时刻的进展情况的数学描述。
- 决策：从某阶段的一个状态演变到下一个阶段某状态的选择。
- 状态转移方程：根据上一阶段的状态和决策导出本阶段的状态。这就像是“递推”。



主要概念



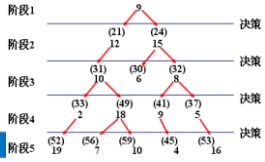
- 阶段：每行就是一个阶段；
- 状态：d[i][j]，即取第*i*行，第*j*个数能够达到的最大值；
- 决策：取第*i*行第*j*个数，则可以有两种方案：取第*i*-1行第*j*-1个数或取第*i*-1行第*j*个数后再取第*i*行第*j*个数；
- 状态转移方程：
  - d[i][j] = max (d[i+1][j]，d[i+1][j+1]) + data[i][j]；
  - 表示取第*i*行第*j*个数所能达到的最大和；

适合解决的问题的性质

- 动态规划算法的问题及决策应该具有两个性质：最优化原理、无后向性。
  - 1) 最优化原理(或称为最佳原则、最优子结构)；
  - 2) 无后向性(无后效性)：某阶段状态一旦确定以后，就不受这个状态以后决策的影响。即某状态以后的过程不会影响以前的状态，只与当前状态有关。
- 能够体现动态规划优点的性质：
  - 子问题重叠性质；
  - 动态规划用空间换取时间，在有大量重叠子问题的时候其优势才能充分体现出来。

适合解决的问题的性质-数塔问题说明

- 最优化原理(最优子结构)
  - 9->12->10->18->10
  - 显然12->10->18->10也是12到最后一层的最大和.....
- 无后效性
  - 如，计算到12的最大和只要考虑到10的最大和与到6的最大和哪个更大，而不要考虑到10的最大和或者到6的最大和具体是哪几个数构成的。



设计动态规划算法的基本步骤

- 设计一个标准的动态规划算法的步骤：
  - 1) 划分阶段；
  - 2) 选择状态；
  - 3) 确定决策并写出状态转移方程。

- 实际应用当中的简化步骤：
  - 1) 分析最优解的性质，并刻划其结构特征。
  - 2) 递推地定义最优值。
  - 3) 以自底向上的方式或自顶向下的记忆化方法(备忘录法)计算出最优值。
  - 4) 根据计算最优值时得到的信息，构造问题的最优解。

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。
- 与贪心法不同的是，动态规划分解得到的子问题，往往是相互不独立的。

- 贪心法在求解过程中，每一步仅对当前状态下进行局部的选择，这种选择依赖于过去所做的选择，而不考虑以后要做的选择。在很多情况下可以得到全局最优解，这主要取决于当前所做的选择是正确的。
- 动态规划法对问题进行全面的规划处理，可以弥补贪心法的不足。

4.3 动态规划的最优决策原理

- 问题的给出：

  对于具有n个输入的最优解问题，它们的活动过程往往划分为若干个阶段，每一阶段的决策依赖于前一阶段的状态，由决策所采取的动作使状态发生转移，称为下一阶段的决策依据。

  S_0是*初始状态*，依据此状态做出决策P_1，按照P_1所采取的动作，使得状态转换为S_1，经过一系列的决策和转移，到达最终状态S_n。于是，一个决策序列就在不断变化的状态中产生。

$$S_0 \xrightarrow{P_1} S_1 \xrightarrow{P_2} S_2 \cdots\cdots S_{n-1} \xrightarrow{P_n} S_n$$

- 由于每一阶段的决策，仅与前一阶段所产生的状态有关，而与如何达到这种状态的方式无关，因此，可以把每一阶段作为一个子问题来处理。
- 决策过程的每一阶段，都可能有多种决策可以选取，其中只有一个决策是对全局最优的。

25

- 为了说明问题，假定对一种状态，可以做出多种决策，而每一种决策可以产生一个新的状态。

对于初始状态S_0，$P_1=\{p_{1,1}, p_{1,2}, \ldots\ldots, p_{1,r_1}\}$是可能的决策值集合，由它们所产生的状态$S_1=\{S_{1,1}, S_{1,2}, \ldots\ldots, S_{1,r_1}\}$，其中是$S_{1,k}$对应于决策$p_{1,k}$所产生的状态。但此时尚无法判断哪一个决策是最优的，于是，把这些决策值集合作为这一阶段的子问题的解保存起来。

26

- 依次类推，在状态集合S_1上做出的决策值集合P_2，产生了状态S_2集合。
- 最后，对状态$S_{n-1}=\{S_{n-1,1}, S_{n-1,2}, \ldots\ldots, S_{n-1,r_{n-1}}\}$，
- $P_n=\{p_{n,11}, \ldots\ldots, p_{n,1k_1}, p_{n,21}, \ldots\ldots, p_{n,2k_2}, \ldots\ldots, p_{n,r_{n-1}1}, \ldots\ldots, p_{n,r_{n-1}k_{n-1}}\}$是可能的决策值集合，其中决策值集合$\{p_{n,t1}, \ldots\ldots, p_{n,tt_i}\}$是依据状态$S_{n-1,t}$所做出的可能的决策。

27

- 由P_n所产生的状态$S_n=\{S_{n,11}, \ldots\ldots, S_{n,1k_1}, S_{n,21}, \ldots\ldots, S_{n,2k_2}, \ldots\ldots, S_{n,r_{n-1}1}, \ldots\ldots, S_{n,r_{n-1}k_{n-1}}\}$。$S_n$是最终状态集合，其中只有一个状态是最优的。
- 假定这个状态是$S_{n,k}$，它由决策p_{n,k_n}所产生的。由此可以确定p_{n,k_n}是最优决策。假定p_{n,k_n}是由依据状态$S_{n-1,k_{n-1}}$做出的，由此回溯，使状态达到$S_{n-1,k_{n-1}}$的决策$P_{n-1,k_{n-1}}$是最优策略。这种回溯一直到p_{1,k_1}，从而得到一个一个最优决策序列$\{p_{1,k_1}, p_{n,21}, \ldots\ldots, p_{2,k_2}, \ldots\ldots, p_{n,k_n}\}$，而这个决策序列导致了状态转移序列$=\{S_0, \ldots\ldots, S_{1,k_1}, S_{2,k_2}, \ldots\ldots, S_{n,k_n}\}$。

28

- 根据最优性原理，**上述决策序列，是根据初始状态S_0和初始决策$p_{1,k}$所产生的状态而构成的一个最优决策序列。**由这个决策序列导致了由初始状态S_0到最优状态S_{n,k_n}的转移。
- 上述决策过程中，有一个赖以决策的策略或目标，该策略或目标称之为***动态规划函数***。它由问题的性质和特点所确定，并应用于每一阶段的决策。整个决策过程，可以递归的进行。

29

- 最优决策是在最后阶段形成的，然后向前倒退，直到初始阶段，而决策的具体结果及所产生的状态转移，却是由初始阶段开始进行计算的，然后向后递归或迭代，直到最终结果。

30

动态规划实例

4.4货郎担问题

- 问题的提出：

  如果对于任意数目的n个城市，分别用1~n编号，则这个问题归结为在有向带权图中，寻找一条路径最短的哈密尔顿回路问题。

  这里，V表示城市顶点，$(i,j) \in E$ 表示城市之间的距离，用邻接矩阵C表示城市之间的距离。

令 $d=(i,\overline{V})$ 表示从顶点i出发，经过 $\overline{V}$ 中各顶点一次，最终回到顶点i的最短路径长度。

开始时，$\overline{V}=V-\{i\}$ ，于是，可以定义下面的动态规划函数：

$$d(i,V-\{i\})=d(i,\overline{V})=\min_{k\in V}\{c_{ik}+d(k,\overline{V}-\{k\})\} \quad (4.4.1)$$
$$d(k,\varphi)=c_{ki} \qquad k\neq i$$

设有四个城市，其中

$$C=(c_{ij})=\begin{bmatrix}\infty & 3 & 6 & 7\\ 5 & \infty & 2 & 3\\ 6 & 4 & \infty & 2\\ 3 & 7 & 5 & \infty\end{bmatrix}$$

开始时，$\overline{V}=V-\{i\}$ ，于是，可以定义下面的动态规划函数：

$$d(i,V-\{i\})=d(i,\overline{V})=\min_{k\in V}\{c_{ik}+d(k,\overline{V}-\{k\})\} \quad (4.4.1)$$
$$d(k,\varphi)=c_{ki} \qquad k\neq i$$

设有四个城市，其中

$$C=(c_{ij})=\begin{bmatrix}\infty & 3 & 6 & 7\\ 5 & \infty & 2 & 3\\ 6 & 4 & \infty & 2\\ 3 & 7 & 5 & \infty\end{bmatrix}$$

- 根据式（4.4.1），由城市1出发，经城市2,3,4然后返回1的最短路径长度为：

$$d(1,\{2,3,4\})=\min\{c_{12}+d(2,\{3,4\}),c_{13}+d(3,\{2,4\}),c_{14}+d(4,\{2,3\})\}$$

- 根据式（4.4.1），由城市1出发，经城市2,3,4然后返回1的最短路径长度为：

$$d(1,\{2,3,4\})=\min\{c_{12}+d(2,\{3,4\}),c_{13}+d(3,\{2,4\}),c_{14}+d(4,\{2,3\})\}$$

这是最后一个阶段的决策，它必须依据 d(2,{3,4}),d(3,{2,4}) 和 d(4,{2,3}) 的计算结果。于是有：

$$d(2,\{3,4\})=\min\{c_{23}+d(3,\{4\}),c_{24}+d(4,\{3\})\}$$
$$d(3,\{2,4\})=\min\{c_{32}+d(2,\{4\}),c_{34}+d(4,\{2\})\}$$
$$d(4,\{2,3\})=\min\{c_{42}+d(2,\{3\}),c_{43}+d(3,\{2\})\}$$

- 这一阶段的决策，又必须依据下面的计算结果：

$$d(3,\{4\}),d(4,\{3\}),d(2,\{4\}),d(4,\{2\}),d(2,\{3\}),d(3,\{2\})$$

再向前推，有：

$$d(3,\{4\})=c_{34}+d(4,\varphi)=c_{34}+c_{41}=2+3=5$$
$$d(4,\{3\})=c_{43}+d(3,\varphi)=c_{43}+c_{31}=5+6=11$$
$$d(2,\{4\})=c_{24}+d(4,\varphi)=c_{24}+c_{41}=3+3=6$$
$$d(4,\{2\})=c_{42}+d(2,\varphi)=c_{42}+c_{21}=7+5=12$$
$$d(2,\{3\})=c_{23}+d(3,\varphi)=c_{23}+c_{31}=2+6=8$$
$$d(3,\{2\})=c_{32}+d(2,\varphi)=c_{32}+c_{21}=4+5=9$$

- 有了这些结果，再向后计算，于是有：

$d(2,\{3,4\})=\min\{2+5,3+11\}=7$　　路径顺序是（2,3,4,1）

$d(3,\{2,4\})=\min\{4+6,2+12\}=10$　　路径顺序是（3,2,4,1）

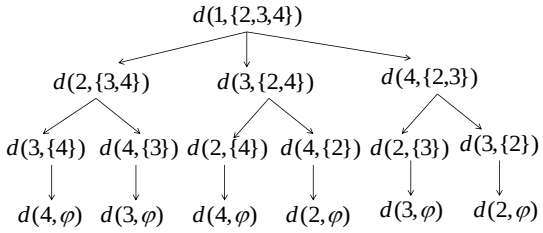
$d(4,\{2,3\})=\min\{7+8,5+9\}=14$　　路径顺序是（4,2,3,1）

最后，有：

$$d(1,\{2,3,4\})=\min\{3+7,6+10,7+14\}=10$$

路径顺序是：1,2,3,4,1

求解过程示意图



$$T_i=\sum_{j=0}^{n-1} j\cdot C_{n-1}^{j}<\sum_{j=0}^{n-1} n\cdot C_{n-1}^{j}=n\sum_{j=0}^{n-1} C_{n-1}^{j}=O(n^2 2^n)$$

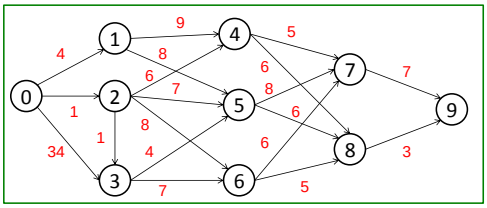
4.5多段图的最短路径问题

- **4.5.1 多段图的决策过程**

定义： 给定有向带权图$G(V,E,W)$，如果把顶点集合V划分成k个不相交的子集V_i，$1\le i\le k$，$k\ge 2$，使得E中的任何一条边(u,v)，必有$u\in V_i$，$v\in V_{i+m}$，$m\ge 1$，则称这样的图为多段图。 $|V_1|=|V_k|=1$，称$s\in V_1$为源点，$t\in V_k$为收点。

多段图的最短路径问题，是求从源点s到达收点t的最小花费的通路。根据多段图的k个不相交子集V_i，把多段图划分为k段，每一段包含顶点的一个子集。

为了便于决策，把顶点集中所有顶点按段的顺序进行编号。



✓ 决策的第一阶段

- 确定图中第k-1 段的所有顶点到达收点t的花费最小的通路。必须把这些信息保存起来，以便在最后形成最优决策时使用。用数组cost[i]存放顶点i到达收点t 的最小花费，用数组path[i]存放顶点i到达收点t 的最小花费通路上的前方顶点编号。

✓决策的第二阶段

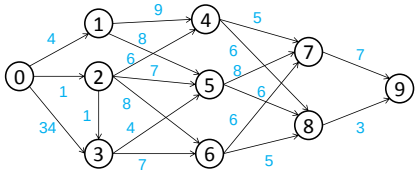
- 确定图中第k-2 段的所有顶点到收点t的花费最小的通路。这时，利用第一阶段所形成的信息来进行决策，并把决策的结果存放在数组cost和path的相应元素中，如此依次进行下去，直到最后确定源点s到收点t的花费最小的通路。
- 源点s的path数组中就是最优决策序列。

✓建立动态规划函数

$$cost[i]=\min_{i<j\le n}\{c_{ij}+cost[j]\} \quad (4.5.1)$$

$$path[i]=\text{使得}\quad c_{ij}+cost[j]\ \text{最小的}j,\quad i<j\le n$$

算法步骤：
（用route[n]存放从源点s出发到达收点t的最短通路上的顶点编号）
（1） 对所有的i, 0≤i<n，把cost[i] 初始化为最大值，path[i]初始化为-1，cost[n-1] 初始化为0;
（2）令i=n-2;
（3） 根据式（4.5.1）计算cost[i] 和path[i]；
（4）i=i-1; 若I ≥0，转到步骤（3），否则转到步骤（5）；
（5）令i=0，route[i]=0;
（6）若route[i]=n-1，算法结束，否则转到步骤（7）；
（7）i=i+1，route[i]=path[route[i-1]]，转到步骤（6）。



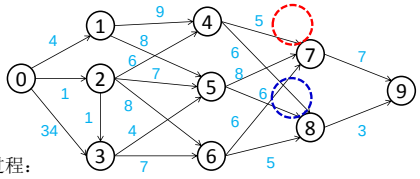
求解过程：

$$cost[i]=\min_{i<j\le n}\{c_{ij}+cost[j]\}$$

$$path[i]=\text{使得}\quad c_{ij}+cost[j]\ \text{最小的}j,\quad i<j\le n$$

i=8：cost[8]=c_{89}+cost[9]=3+0=3;
path[8]=9

i=7：cost[7]=c_{79}+cost[9]=7+0=7;
path[7]=9

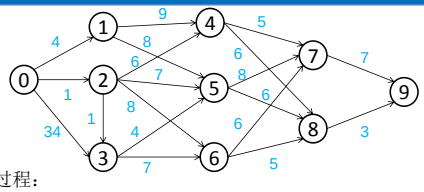


求解过程：

i=6：cost[6]=min{c_{67}+cost[7], c_{68}+cost[8]}=min{6+7, 5+3}=8;
path[6]=8

i=5：cost[5]=min{c_{57}+cost[7], c_{58}+cost[8]}=min{8+7, 6+3}=9;
path[6]=8

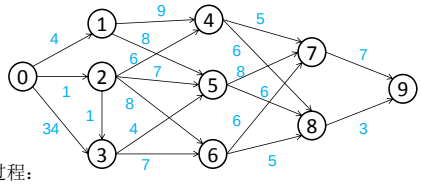
i=4：cost[4]=min{c_{47}+cost[7], c_{48}+cost[8]}=min{5+7, 6+3}=9;
path[4]=8



求解过程：

i=3：cost[3]=min{c_{35}+cost[5], c_{36}+cost[6]}=min{4+9, 7+8}=13;
path[3]=5

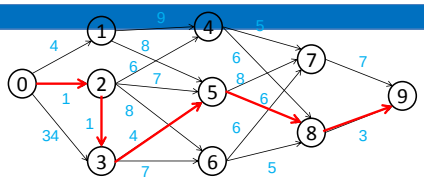
i=2：cost[2]=min{c_{23}+cost[3], c_{24}+cost[4], c_{25}+cost[5],
c_{26}+cost[6],}=min{1+13, 6+9,7+9,8+8}=14;
path[2]=3



求解过程：

i=1：cost[1]=min{c_{14}+cost[4], c_{15}+cost[5]}=min{9+9, 6+9}=15;
path[1]=5

i=0：cost[0]=min{c_{01}+cost[1], c_{02}+cost[2],
c_{03}+cost[3]}=min{4+15, 1+14,3+13}=15;
path[0]=2



route[0]=0;
route[1]=path[route[0]]=path[0]=2;
route[2]=path[route[1]]=path[2]=3;
route[3]=path[route[2]]=path[3]=5;
route[4]=path[route[3]]=path[5]=8;
route[5]=path[route[4]]=path[8]=9;

- **4.5.2 算法实现**

$O(n+m)$

```
Struct node{
   int v_num;
   type len;
   struct node *next;
}
```

```
局部决策：
For (i=n-2; i>=0;i--)
{
   pnode=node[i]->next;
   while (pnode!=NULL) {
     if (pnode->len+cost[pnode->v_num]<cost[i] {
        cost[i]=pnode->len+cost[pnode->v_num];
        path[i]=pnode->v_num;
     }
   pnode=pnode->next;
  }
}
```

```
初始化：
For (i=0;i<n;i++)
 {
    cost[i]=Maxtype;
    path[i]=-1;
    route[i]=0;
 }
 cost[n-1]=zero;
```

```
最优决策序列：
While ((route[i]!=n-1)&&(path[i]!=-1)) {
   i++;
   route[i]=path[route[i-1]];
}
```


4.6 资源分配问题

- 资源分配问题，是考虑如何把有限的资源分配给若干个工程问题。假设资源总数为 r，工程个数为 n，给每个工程投入的资源不同，所得的利润也不同，要求把总数为 r 的资源分配给n个工程，以获得最大利润的分配方案。

✓ 资源分配的决策过程

- 把资源 r 划分为m个相同的部分，每份资源为 r/m，m为整数。假定利润函数为$G_i(x)$, $1\le i\le n$, $0\le x\le m$，表示把 x 份资源分配给第 i 个工程所得的利润，则分配m份资源给所有的工程，所得到的利润总额为：

$$G(m)=\sum_{i=1}^{n}G_i(x_i)$$

$$\sum_{i=1}^{n}x_i=m$$

于是，问题转换为把m份资源分配给n个工程，使得$G(m)$最大的问题。

- **首先，把各个工程按顺序编号，然后按下述方法进行划分：**

第一阶段：分别把x=0,1,…,m份资源分配给第**1**个工程，确定第一个工程在各种不同份额的资源下，能够得到最大的利润。

第二阶段：分别把x=0,1,…,m份资源分配给第**1**、**2**个工程，确定第**1**个工程在各种不同份额的资源下，这两个工程能够得到最大的利润。以及在此利润下，第二个工程的最优分配份额。依次类推，在第n个阶段，分别把x=0,1,…,m份资源分配给所有n个工程，确定能够得到最大的利润，以及在此利润下，第n个工程的最优分配份额。

考虑到把m份资源全部投入给所有n个工程部一定能够得到最大利润，***因此必须在各个阶段里，对不同的分配份额计算能够得到的最大利润，然后取其中的最大者，作为每个阶段能够取得的最大利润。***再取每个阶段的最大利润中的最大者，以及在该最大利润下的分配方案，即为整个资源分配的最优决策。

令$f_i(x)$表示把x份资源分配给前i个工程时，所得到的最大利润，$d_i(x)$表示使$f_i(x)$最大时，分配给第i个工程的资源份额。于是，在第一阶段，只把x份资源分配给第一个工程，有：

$$\begin{cases} f_1(x)=G_1(x) \\ d_1(x)=x \end{cases} \quad 0\le x\le m \qquad (4.6.1)$$

在第二阶段，只把x份资源分配给前面两个工程，有：

$$\begin{cases} f_2(x)=\max\limits_{z}\{G_2(z)+f_1(x-z)\} \\ d_2(x)=\text{使}\ f_2(x)\text{达到最大的}z \end{cases} \quad 0\le x\le m, 0\le z\le x$$

一般地，在第i个阶段，把x份资源分配给前面i个工程，有：

$$\begin{cases} f_i(x)=\max\limits_{z}\{G_i(z)+f_{i-1}(x-z)\} \\ d_i(x)=\text{使}\quad f_i(x)\text{达到最大的}z \end{cases} \quad 0\le x\le m, 0\le z\le x \qquad (4.6.2)$$

令第 i 阶段的最大利润为g_i，则：

$$g_i=\max\{f_i(1),\ldots,f_i(m)\} \qquad (4.6.3)$$

设q_i是使g_i达最大时，分配给前面i个工程的资源份额，则:

$$q_i=\text{使}\quad f_i(x)\text{达到最大的}x \qquad (4.6.4)$$

在每个阶段，把所得的所有局部决策值

$f_i(x)$，$d_i(x)$，g_i，q_i保存起来。最后，在第n阶段结束之后，令全局的最大利润为$optg$，则：

$$optg=\max\{g_1,g_2,\ldots,g_n\} \qquad (4.6.5)$$

在全局最大利润下，所分配工程项目的最大编号（即所分配工程项目的最大数目）为 k，则：

$$k=\text{使}g_i\text{最大的}i \qquad (4.6.6)$$

分配给前面k个工程的最优份额为：

$$optx=\text{与最大的}g_i\text{相对应的}q_i \qquad (4.6.7)$$

分配给第k个工程的最优份额为：

$$optq_k=d_k(optx)$$

分配给其余k-1个工程的剩余的最优份额为：

$$optx=optx-d_k(optx)$$

由此回溯，得到分配给前面各个工程的最优份额的递推关系式：

$$\begin{cases} optq_i=d_i(optx) \\ optx=optx-optq_i \end{cases} \quad i=k,k-1,\ldots,1 \qquad (4.6.8)$$

问题的求解步骤：

- 按照（4.6.1）、（4.6.2），对各个阶段i，各个不同份额x的资源，计算$f_i(x)$，$d_i(x)$。
- 按照（4.6.3）、（4.6.4），计算各个阶段的最大利润g_i，获得此最大利润的份额q_i。
- 按照（4.6.5）、（4.6.6）和（4.6.7），计算全局的最大利润$optg$、总的最优分配份额$optx$以及编号最大的工程项目k。
- 按照（4.6.8），递推计算各个工程的最优分配份额。

- 例题：有8个份额的资源，分配给3个工程，其利润如下：

x	0	1	2	3	4	5	6	7	8
$G_1(x)$	0	4	26	40	45	50	51	52	53
$G_2(x)$	0	5	15	40	60	70	73	74	75
$G_3(x)$	0	5	15	40	80	90	95	98	100

求资源的最优分配方案。

解：第一步，求各个阶段不同分配份额时的最优分配方案，即各个工程在此利润下的分配份额。

$$\begin{cases} f_1(x)=G_1(x) \\ d_1(x)=x \end{cases} \quad 0\le x\le m$$

在第一阶段，只把资源的份额分配给第一个工程，由式（4.6.1），有：

x	0	1	2	3	4	5	6	7	8
$f_1(x)$	0	4	26	40	45	50	51	52	53
$d_1(x)$	0	1	2	3	4	5	6	7	8

其次，把资源的份额分配给前面两个工程，当x=0，显然有：

$$f_2(0)=0,\quad d_2(0)=0$$

当x=1时，由式（4.6.2）有：

$$\begin{cases} f_i(x)=\max\limits_{z}\{G_i(z)+f_{i-1}(x-z)\} \\ d_i(x)=\text{使}\quad f_i(x)\text{达到最大的}z \end{cases} \quad 0\le x\le m, 0\le z\le x$$

$$f_2(1)=\max(G_2(0)+f_1(1),G_2(1)+f_1(0))=\max(4,5)=5$$
$$d_2(1)=1$$

当x=2时，由式（4.6.2）有：

$$f_2(2)=\max(G_2(0)+f_1(2),G_2(1)+f_1(1),G_2(2)+f_1(0))$$
$$=\max(26,9,15)=26$$
$$d_2(2)=0$$

类似的计算x=3,4,…,8时的$f_2(x)$及$d_2(x)$的值，有：

x	0	1	2	3	4	5	6	7	8
$f_2(x)$	0	5	26	40	60	70	86	100	110
$d_2(x)$	0	1	0	0	4	5	4	4	5

同样计算$f_3(x)$及$d_3(x)$的值，有：

x	0	1	2	3	4	5	6	7	8
$f_3(x)$	0	5	26	40	80	90	106	120	140
$d_3(x)$	0	1	0	0	4	5	4	4	4

第二步，按照（4.6.3），（4.6.4），求各个阶段的最大利润，以及在此利润下的分配份额，有：

$$g_i = \max\{ f_i(1),..., f_i(m)\}$$

q_i = 使 $f_i(x)$达到最大的x

x	0	1	2	3	4	5	6	7	8
$f_1(x)$	0	4	26	40	45	50	51	52	53
$d_1(x)$	0	1	2	3	4	5	6	7	8

$$g_1 = 53$$
$$q_1 = 8$$

第二步，按照（4.6.3），（4.6.4），求各个阶段的最大利润，以及在此利润下的分配份额，有：

$$g_i = \max\{ f_i(1),..., f_i(m)\}$$

q_i = 使 $f_i(x)$达到最大的x

x	0	1	2	3	4	5	6	7	8
$f_2(x)$	0	5	26	40	60	70	86	100	110
$d_2(x)$	0	1	0	0	4	5	4	4	5

$$g_1 = 53 \quad g_2 = 110$$
$$q_1 = 8 \quad q_2 = 5$$

第二步，按照（4.6.3），（4.6.4），求各个阶段的最大利润，以及在此利润下的分配份额，有：

$$g_i = \max\{ f_i(1),..., f_i(m)\}$$

q_i = 使 $f_i(x)$达到最大的x

x	0	1	2	3	4	5	6	7	8
$f_3(x)$	0	5	26	40	80	90	106	120	140
$d_3(x)$	0	1	0	0	4	5	4	4	4

$$g_1 = 53 \quad g_2 = 110 \quad g_3 = 140$$
$$q_1 = 8 \quad q_2 = 5 \quad q_3 = 4$$

第三步，按照（4.6.5），（4.6.6），（4.6.7），计算全局的最大利润$optg$、最大的工程数目以及总的最优分配额度：

$$optg = \max\{ g_1, g_2,..., g_n\}$$

$optx$ = 与最大的g_i相对应的q_i

k = 使g_i最大的i

$$g_1 = 53 \quad g_2 = 110 \quad g_3 = 140$$
$$q_1 = 8 \quad q_2 = 5 \quad q_3 = 4$$

$$optg = 140, \quad optx = 8, \quad k = 3$$

第四步，按照（4.6.8），计算各个工程的最优分配额度：

$$\begin{cases} optq_i = d_i(optx) \\ optx = optx - optq_i \end{cases} \quad i = k, k-1,...,1$$

$$optg = 140, \quad optx = 8, \quad k = 3$$

x	0	1	2	3	4	5	6	7	8
$f_3(x)$	0	5	26	40	80	90	106	120	140
$d_3(x)$	0	1	0	0	4	5	4	4	4

$optq_3 = d_3(optx) = d_3(8) = 4, \quad optx = optx - optq_3 = 8 - 4 = 4$

$optq_2 = d_2(optx) = d_2(4) = 4, \quad optx = optx - optq_2 = 4 - 4 = 0$

$optq_1 = d_1(optx) = d_1(0) = 0$

最后的决策是：

分配给2,3工程各4个份额，可得最大利润140.

第四步，按照（4.6.8），计算各个工程的最优分配额度：

$$\begin{cases} optq_i = d_i(optx) \\ optx = optx - optq_i \end{cases} \quad i = k, k-1, \ldots, 1$$

$$optg = 140, \quad optx = 8, \quad k = 3$$

x	0	1	2	3	4	5	6	7	8
$f_2(x)$	0	5	26	40	60	70	86	100	110
$d_2(x)$	0	1	0	0	4	5	4	4	5

$$optq_3 = d_3(optx) = d_3(8) = 4, \quad optx = optx - optq_3 = 8-4 = 4$$
$$optq_2 = d_2(optx) = d_2(4) = 4, \quad optx = optx - optq_2 = 4-4 = 0$$
$$optq_1 = d_1(optx) = d_1(0) = 0$$

$$optq_3 = d_3(optx) = d_3(8) = 4, \quad optx = optx - optq_3 = 8-4 = 4$$
$$optq_2 = d_2(optx) = d_2(4) = 4, \quad optx = optx - optq_2 = 4-4 = 0$$
$$optq_1 = d_1(optx) = d_1(0) = 0$$

x	0	1	2	3	4	5	6	7	8
$G_1(x)$	0	4	26	40	45	50	51	52	53
$G_2(x)$	0	5	15	40	60	70	73	74	75
$G_3(x)$	0	5	15	40	80	90	95	98	100

最后的决策是：

分配给2,3工程各4个份额，可得最大利润140.

4.7 最长公共子序列问题

- 假定，$A=a_1a_2\ldots a_n$是字母表∑上的一个字符序列，如果存在∑上的另外一个字符序列$S=c_1c_2\ldots c_j$，使得对所有的k，$k=1,2,\ldots,j$，有$c_k=a_{ik}$（其中，$1\le ik\le n$），是字符序列A的一个下标递增序列，则称字符序列S是A的子序列。

- 如果∑={x,y,x}，∑上的字符序列是A=xyzyxzxz，则xxx是A的一个长度为3的子序列。该子序列中的字符，对应于A的下标是1,5,7。而$xzyzx$是A的长度为5的子序列，对应于A的下标是1,3,4,6,7.

- 给定两个字符序列A=xyzyxzxz和B=xzyxxyzx。则xxx是这两个字符序列的长度为3的公共子序列，$xzyz$是这两个字符序列的长度为4的公共子序列，而$xzyxxz$是这两个字符序列的长度为6的最长公共子序列。

- **因此，最长公共子序列的问题是：给定两个字符序列$A=a_1a_2\ldots a_n$和$B=b_1b_2\ldots b_m$，寻找A和B的一个公共子序列，使得它是A和B的最长公共子序列。**

✓ 搜索过程

- 令序列$A=a_1a_2\ldots a_n$和$B=b_1b_2\ldots b_m$，记$A=a_1a_2\ldots a_k$为A中最前面连续k个字符的子序列，记$B=b_1b_2\ldots b_k$为B中最前面连续k个字符的子序列，容易看出，序列A和序列B的最长公共子序列，有如下性质：

（1）若$a_n=b_m$，序列$S_k=c_1c_2\ldots c_k$是序列A和序列B的长度为k的最长公共子序列，必有$a_n=b_m=c_k$，且序列$S_{k-1}=c_1c_2\ldots c_{k-1}$是序列$A_{n-1}$和$B_{m-1}$的长度为$k$-1的最长公共子序列。

（2）若$a_n\ne b_m$，且$a_n\ne c_k$，则序列$S_k=c_1c_2\ldots c_k$是序列A_{n-1}和序列B的长度为k的最长公共子序列。

（3）若$a_n\ne b_m$，且$b_m\ne c_k$，则序列$S_k=c_1c_2\ldots c_k$是序列A和序列B_{m-1}的长度为k的最长公共子序列。

- **若记$L_{n,m}$为序列A_n和B_m的最长公共子序列的长度，则为$L_{i,j}$为序列A_i和B_j的最长公共子序列的长度。根据最长公共子序列的性质，有：**

$$L_{0,0} = L_{i,0} = L_{0,j} = 0 \quad 1 \le i \le n, 1 \le j \le m \qquad (4.7.1)$$

$$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & a_i = b_j, i > 0, j > 0 \\ \max\{L_{i,j-1}, L_{i-1,j}\} & a_i \ne b_j, i > 0, j > 0 \end{cases} \qquad (4.7.2)$$

性质（1）和性质（2）(3)

$$L_{0,0}=L_{i,0}=L_{0,j}=0 \quad 1\le i\le n, 1\le j\le m \qquad (4.7.1)$$

$$L_{i,j}=\begin{cases} L_{i-1,j-1}+1 & a_i=b_j, i>0, j>0 \\ \max\{L_{i,j-1}, L_{i-1,j}\} & a_i\neq b_j, i>0, j>0 \end{cases} \qquad (4.7.2)$$

因此，对于最长公共子序列的搜索，分成n个阶段。第一阶段按照（4.7.1）和（4.7.2），计算A_1和B_j的最长公共子序列的长度$L_{1,j}$，j=1,2,…,m。第二阶段，按照$L_{1,j}$和（4.7.2）计算A_2和B_j的最长公共子序列的长度$L_{2,j}$，j=1,2,…,m。依次类推，最后，在第n阶段，按$L_{n-1,j}$和（4.7.2）计算A_n和B_j的最长公共子序列的长度$L_{n,j}$，j=1,2,…,m。于是，在第n阶段的$L_{n,m}$便是序列A_n和B_m的最长公共子序列的长度。

- 为了得到A_n和B_m的最长公共子序列，设置一个二维的状态数组$s_{i,j}$，在上述每一阶段计算$L_{i,j}$的过程中，根据公共子序列的三个性质，按如下方法把***搜索状态***记录于状态数组中：

$$\begin{aligned} s_{i,j}&=1 \quad 若 a_i=b_j \\ s_{i,j}&=2 \quad 若 a_i\neq b_j, 且 L_{i-1,j}\ge L_{i,j-1} \\ s_{i,j}&=3 \quad 若 a_i\neq b_j, 且 L_{i-1,j}< L_{i,j-1} \end{aligned} \qquad (4.7.3)$$

设$L_{n,m}=k$，$S_k=c_1c_2\ldots c_k$是序列A_n和序列B_m的长度为k的最长公共子序列，最长公共子序列的搜索过程从状态字$s_{n,m}$开始。搜索过程如下：

若$s_{n,m}=1$，表明$a_n=b_m$。根据最长公共子序列性质1，$c_k=a_n$是子序列的最后一个字符，且前一个字c_{k-1}符是序列A_{n-1}和序列B_{m-1}的长度为k-1的最长公共子序列的最后一个字符，下一个搜索方向是$s_{n-1,m-1}$。

若$s_{n,m}=2$，表明$a_n\neq b_m$，且$L_{n-1,m}\ge L_{n,m-1}$。根据最长公共子序列性质2，$c_k\neq a_n$，序列$S_k=c_1c_2\ldots c_k$是序列A_{n-1}和序列B_m的长度为k的最长 公共子序列，下一个搜索方向是$s_{n-1,m}$。

若$s_{n,m}=3$，表明$a_n\neq b_m$，且$L_{n-1,m}< L_{n,m-1}$。根据最长公共子序列性质3，$c_k\neq b_m$，序列$S_k=c_1c_2\ldots c_k$是序列A_n和序列B_{m-1}的长度为k的最长 公共子序列，下一个搜索方向是$s_{n,m-1}$。

由此，可以得到下面一般的递推关系：

$$\begin{aligned} &若 \quad s_{i,j}=1, 则 c_k=a_i, i=i-1, j=j-1, k=k-1 \\ &若 \quad s_{i,j}=2, 则 i=i-1 \\ &若 \quad s_{i,j}=3, 则 j=j-1 \end{aligned} \qquad (4.7.4)$$

从$i=n, j=m$开始搜索，直到i=0或j=0结束，即可得到A 和B的最长公共子序列。

例题：求A=*xyxzyxyzzy*和B=*xzyzxyzxyzxy*的最长公共子序列。

	0	1 x	2 z	3 y	4 z	5 x	6 y	7 z	8 x	9 y	10 z	11 x	12 y
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 x	0	1	1	1	1	1	1	1	1	1	1	1	1
2 y	0	1	1	2	2	2	2	2	2	2	2	2	2
3 x	0	1	1	2	2	3	3	3	3	3	3	3	3
4 z	0	1	2	2	3	3	3	4	4	4	4	4	4
5 y	0	1											
6 x	0	1											
7 y	0	1											
8 z	0	1											
9 z	0	1											
10 y	0	1	2	3	4	4	5	6	6	7	7	7	8

最长公共子序列长度求解过程：

i=1,j=1,$a_1=b_1$,$L_{1,1}$=1

i=1,j=2,$a_1\neq b_2$,
$L_{1,1}$=1，$L_{0,2}$=0
$L_{1,2}$=1

i=2,j=1,$a_2\neq b_1$,
$L_{2,0}$=0，$L_{1,1}$=1
$L_{2,1}$=1

$$L_{i,j}=\begin{cases} L_{i-1,j-1}+1 & a_i=b_j, i>0, j>0 \\ \max\{L_{i,j-1}, L_{i-1,j}\} & a_i\neq b_j, i>0, j>0 \end{cases}$$

例题：求A=xyxzyxyzzy和B=xzyzxyzxyzxy的最长公共子序列。

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	3										
2	0	2											
3	0												
4	0												
5	0												
6	0												
7	0												
8	0												
9	0												
10	0												

i=1,j=1,a_1=b_1,$L_{1,1}$=1

i=1,j=2,a_1≠b_2,
$L_{1,1}$=1，$L_{0,2}$=0
$L_{1,2}$=1

i=2,j=1,a_2≠b_1,
$L_{2,0}$=0，$L_{1,1}$=1
$L_{2,1}$=1

$$s_{i,j}=1 \quad 若a_i=b_j$$
$$s_{i,j}=2 \quad 若a_i\neq b_j,且L_{i-1,j}\geq L_{i,j-1}$$
$$s_{i,j}=3 \quad 若a_i\neq b_j,且L_{i-1,j}< L_{i,j-1}$$

	0	1 x	2 z	3 y	4 z	5 x	6 y	7 z	8 x	9 y	10 z	11 x	12 y
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 x	0	1	3	3	3	1	3	3	1	3	3	1	3
2 y	0	2	2	1	3	3	1	3	3	1	3	3	1
3 x	0	1	2	2	2	1	3	3	1	3	3	1	3
4 z	0	2	1	2	1	2	2	1	3	3	1	3	3
5 y	0	2	2	1	2	2	1	2	2	1	3	3	1
6 x	0	1	2	2	2	1	2	2	1	2	2	1	3
7 y	0	2	2	1	2	2	1	3	2	1	3	2	1
8 z	0	2	1	2	1	2	2	1	3	2	1	3	2
9 z	0	2	1	2	1	2	2	1	2	2	1	2	2
10 y	0	2	2	1	2	2	1	2	2	1	2	2	1

s_{ij}=3,
j=j-1;

s_{ij}=2,
i=i-1;

i=n;j=m;
s_{ij}=1
i=i-1;j=j-1

若 $s_{i,j}=1$,则$c_k=a_i, i=i-1, j=j-1, k=k-1$
若 $s_{i,j}=2$,则$i=i-1$
若 $s_{i,j}=3$,则$j=j-1$

4.8 0/1 背包问题

- 在0/1背包问题中，物体或者被装入背包中，或者不被装入背包中，只有两种选择。
- 假设：x_i表示物体被装入背包的情况，则有当x_i=0时表示物体没有被装入背包，当x_i=1时表示物体被装入背包。
- 根据题目要求，有如下约束方程和目标函数：

$$\sum_{i=1}^{n} w_i x_i \le M, \quad optp=\max\sum_{i=1}^{n} p_i x_i$$

问题被归结为寻找一个满足上述约束方程并且使得目标函数为最大的解向量$X(x_1,x_2,\ldots,x_n)$。

假设背包的重量范围为0~m，类似于资源分配一样，令$optp_i(j)$表示在前i个物体中，能够装入载重量为j的背包中的物体的最大值，j=1,2,…,m。

显然，此时在前i个物体中，有些物体可以装入背包，有些物体不能装入背包。于是可以建立下来动态规划函数：

$$optp_i(0)=optp_0(j)=0 \qquad (4.8.1)$$

$$optp_i(j)=\begin{cases} optp_{i-1}(j) & j<w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i)+p_i\} & j\ge w_i \end{cases} \qquad (4.8.2)$$

(4.8.1)式说明，将前面i个物体装入重量为0的背包，或者把0个物体装入重量为j的背包，得到的价值都为0.

$$optp_i(j)=\begin{cases} optp_{i-1}(j) & j<w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i)+p_i\} & j\ge w_i \end{cases} \qquad (4.8.2)$$

（4.8.2）第一式说明，如果第i个物体的重量大于背包的载重量，则装入前面i个物体得到的最大值与装入前i-1个物体得到的最大值一样（第i个物体没有装入背包）。

（4.8.2）第二式中$optp_{i-1}(j-w_i)+p_i$说明，当第i个物体的重量小于背包的重量时，如果把第i个物体装入载重量为j的背包，则背包中物体的价值等于把前面i-1个物体状态载重量为j-w_i的背包所得到的价值加上第i个物体的价值p_i。

按照这样的定义，可以把求解划分为二个阶段：

第一个阶段， 只装入一个物体，确定在各种不同载重量的背包下能够得到的最大值。

第二阶段， 装入前两个物体，确定在各种不同的载重量的背包下等够得到的最大值。以此类推，直到第n个阶段。最后，$Opt\ p_n(m)$便是在载重量为m的背包下，装入n个物体时能够得到的最大价值。

为了确定装入背包的具体物体，从最大价值$Opt\,p_n(m)$向前倒推。如果$Opt\,p_n(m)$大于$Opt\,p_{n-1}(m)$，表面第n个物体被装入背包，则前n-1个物体被装入重量为m-w_i的背包中。如果$Opt\,p_n(m)$小于$Opt\,p_{n-1}(m)$，表面第n个物体未被装入背包中，则前n-1个物体被装入在载重量为m的背包中。以此类推，直到确定第一个物体是否被装入背包中为止。

若 $optp_i(j)=optp_{i-1}(j)$，则$x_i=0$

若 $optp_i(j)>optp_{i-1}(j)$，则$x_i=1, j=j-w_i$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，背包的重量为10.

求解：设定一个数组，存放前面i个物体装入载重量为j的背包时所获得的最大价值。

$$optp_i(0)=optp_0(j)=0$$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										
5	0										

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，背包的重量为10.

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	**0**	**6**								

i=1;j=1
wi=2,
j<wi
$optp_{i-1}(1)$=0

i=1;j=2
wi=2,
j=wi
$optp_{i-1}(2)$=0
$optp_{i-1}(2\text{-}2)+6$=6

$$optp_i(j)=\begin{cases}optp_{i-1}(j) & j<w_i\\ \max\{optp_{i-1}(j),optp_{i-1}(j-w_i)+p_i\} & j\geq w_i\end{cases}$$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，背包的重量为10.

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6
2	0	0	0	6	6	9	9	9	9	9	9
3	0	0	0	6	6	9	9	9	11	11	14
4	0	0	0	6	6	9	9	9	10	11	13	14
5	0	0	0	6	6	9	9	12	12	15	15	***15***

$$optp_i(j)=\begin{cases}optp_{i-1}(j) & j<w_i\\ \max\{optp_{i-1}(j),optp_{i-1}(j-w_i)+p_i\} & j\geq w_i\end{cases}$$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，背包的重量为10.

若 $optp_i(j)=optp_{i-1}(j)$，则$x_i=0$

若 $optp_i(j)>optp_{i-1}(j)$，则$x_i=1, j=j-w_i$

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6
2	0	0	0	6	6	9	9	9	9	9	9
3	0	0	0	6	6	9	9	9	11	11	14
4	0	0	0	6	6	9	9	9	10	11	13	14
5	0	0	0	6	6	9	9	12	12	15	15	***15***

i=5;j=10
$optp_i(10)$=15
$optp_{i-1}(10)$=14
$x5$=1;
$w5$=4,
j=10-4=6

i=4;j=6
$optp_i(6)$=9
$optp_{i-1}(6)$=9
$x4$=0;

i=3;j=6
$optp_i(6)$=9
$optp_{i-1}(6)$=9
$x3$=0;

i=2;j=6
$optp_i(6)$=9
$optp_{i-1}(6)$=6
$x2$=1;
$w2$=2,
j=6-2=4

i=1;j=4
$optp_i(4)$=6
$optp_{i-1}(4)$=0
$x1$=1;