第六章　动态规划法

历史及研究问题

- 动态规划是运筹学的一个分支，20世纪50年代初美国数学家Bellman等人在研究多阶段决策过程的优化问题时，提出了著名的最优性原理，创立了解决这类过程优化问题的新方法——动态规划法。

- 多阶段决策问题：求解的问题可以划分为一系列相互联系的阶段，每个阶段都需要做出决策，而且一个阶段决策的选择会影响下一个阶段的决策，从而影响整个过程的活动路线，求解的目标是选择各个阶段的决策使整个过程达到最优。

历史及研究问题

- 动态规划主要用于求解以时间划分阶段的动态过程的优化问题，但是一些与时间无关的静态规划（如线性规划、非线性规划），可以人为地引进时间因素，把它视为多阶段决策过程，也可以用动态规划方法方便地求解。

- 动态规划问世以来，在经济管理、生产调度、工程技术和最优控制等方面得到了广泛的应用。例如最短路线、库存管理、资源分配、设备更新、排序、装载等问题，用动态规划方法比其它方法求解更为方便。

概 述——多阶段决策过程

当某问题有n个输入，问题的解由这n个输入的一个子集组成，这个子集必须满足某些事先给定的条件，这些条件称为约束条件，满足约束条件的解称为问题的可行解。

满足约束条件的可行解可能不止一个，为了衡量这些可行解的优劣，通常以函数的形式给出一定的标准，这些标准函数称为目标函数（或评价函数），使目标函数取得极值（极大或极小）的可行解称为最优解，这类问题称为最优化问题 。

概 述——多阶段决策过程

例如，在0/1背包问题中，物品i或者被装入背包，或者不被装入背包。设x_i表示物品i装入背包的情况，则当$x_i=0$时，表示物品i没有被装入背包；$x_i=1$时，表示物品i被装入背包。根据问题的要求，有如下约束条件和目标函数：

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\} \quad (1 \le i \le n) \end{cases}$$

$$\max \sum_{i=1}^{n} v_i x_i$$

最优化问题

例如，付款问题：超市的自动柜员机（POS机）要找给顾客数量最少的现金。

假定POS机中有n张面值为$p_i(1\le i\le n)$的货币，用集合$P=\{p_1, p_2, \ldots, p_n\}$表示，若POS机需支付的现金为$A$，则必须从$P$中选取一个最小子集$S$，使得

$$p_i \in S \quad , \quad \sum_{i=1}^{m} p_i = A \quad (m = |S|) \qquad \text{(式6.1)}$$

如果用向量$X=(x_1, x_2, \ldots, x_n)$表示$S$中所选取的货币，则

$$x_i = \begin{cases} 1 & p_i \in S \\ 0 & p_i \notin S \end{cases} \qquad \text{(式6.2)}$$

最优化问题

则，POS机支付的现金必须满足

$$\sum_{i=1}^{n} x_i p_i = A \qquad \text{(式6.3)}$$

并且

$$d = \min \sum_{i=1}^{n} x_i \qquad \text{(式6.4)}$$

集合*P*是该问题的输入，满足式6.1的解称为可行解，式6.2是解的表现形式，因向量*X*中有*n*个元素，每个元素的取值为0或1，所有这些向量的全体构成该问题的解空间

式6.3是该问题的约束条件，式6.4是该问题的目标函数，使式6.4取得极小值的解称为该问题的最优解。

最优性原理

① 状态：表示每个阶段开始时，问题或系统所处的客观状况。状态既是该阶段的某个起点，又是前一个阶段的某个终点。通常一个阶段有若干个状态。

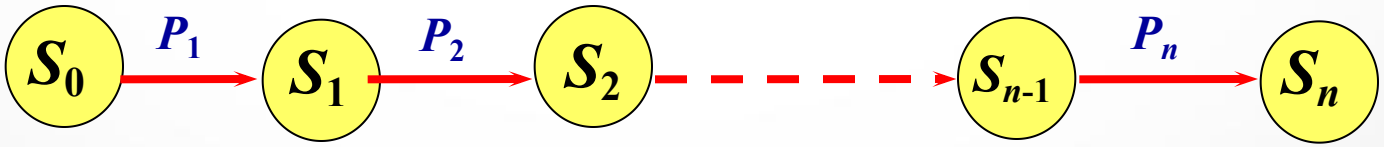
- 状态无后效性：如果某个阶段状态给定后，则该阶段以后过程的发展不受该阶段以前各阶段状态的影响，也就是说状态具有马尔科夫性。
- 适于动态规划法求解的问题具有状态的无后效性

② 策略：各个阶段决策确定后，就组成了一个决策序列，该序列称之为一个策略。由某个阶段开始到终止阶段的过程称为子过程，其对应的某个策略称为子策略。

最优性原理

具有n个输入的最优化问题，其求解过程划分为若干个阶段，每一阶段的决策仅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，成为下一阶段决策的依据。

一个决策序列在不断变化的状态中产生。这个决策序列产生的过程称为多阶段决策过程。

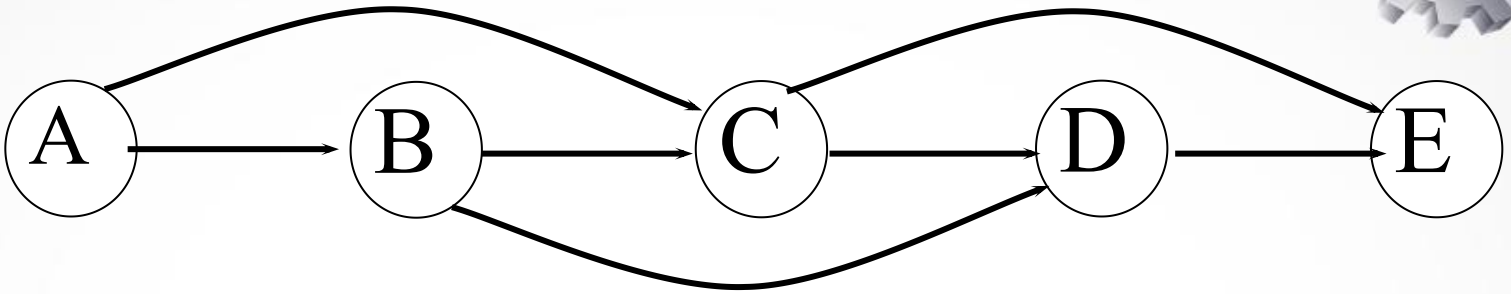


最优性原理

最优性原理：无论决策过程的初始状态和初始决策如何，其余的决策都必须相对于初始决策所产生的当前状态，构成一个最优决策序列。

换言之，在多阶段决策中，各子问题的解只与它前面的子问题的解相关，而且各子问题的解都是相对于当前状态的最优解，整个问题的最优解是由各个子问题的最优解构成。如果一个问题满足最优性原理，通常称此问题具有最优子结构性质。

最优性原理



在上图所示的决策过程中，原问题**E**的解依赖于子问题**C**和**D**的解，子问题**D**的解依赖于子问题**C**和**B**的解，子问题**C**的解依赖于子问题**A**和**B**的解，子问题**B**的解依赖于子问题**A**的解

因此，动态规划的求解过程从初始子问题**A**开始，逐步求解并记录各子问题的解，直至得到原问题**E**的解。

最优性原理

◆最优性原理表明，一个最优问题的任何实例的最优解是由该实例的子实例的最优解组成。

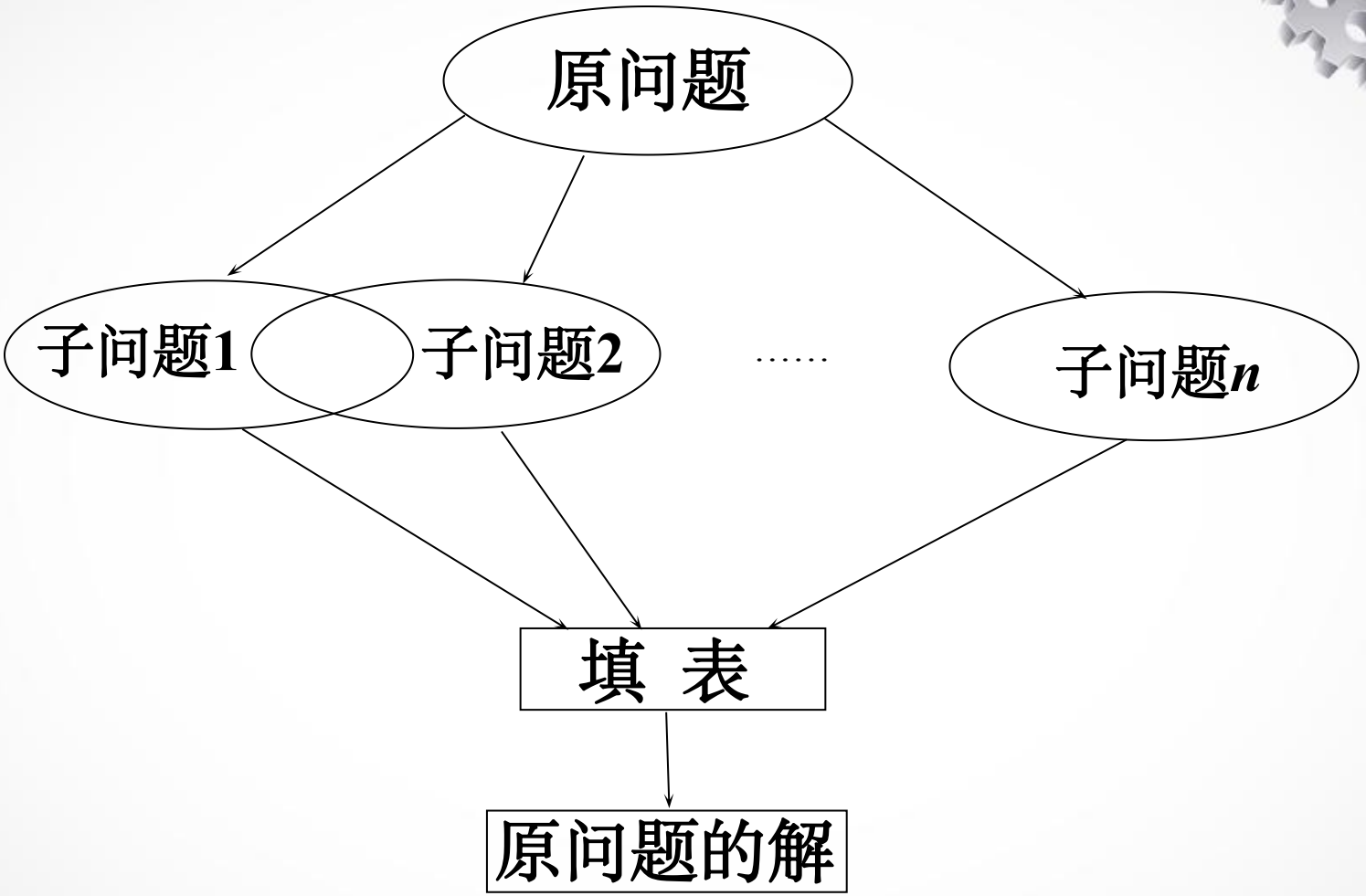
◆一般来说，如果所求解问题对于最优性原理成立，则说明用动态规划方法有可能解决该问题。而解决问题的关键在于获取各阶段间的递推关系式。

动态规划法的设计思想

动态规划法将待求解问题分解成若干个相互重叠的子问题，每个子问题对应决策过程的一个阶段，一般来说，子问题的重叠关系表现在对给定问题求解的递推关系（也就是动态规划函数）中

将子问题的解求解一次并填入表中，当需要再次求解此子问题时，可以通过查表获得该子问题的解而不用再次求解，从而避免了大量重复计算。为了达到这个目的，可以用一个表来记录所有已解决的子问题的解，这就是动态规划法的设计思想。具体的动态规划法是多种多样的，但他们具有相同的填表形式。

动态规划法的设计思想



动态规划法的求解过程

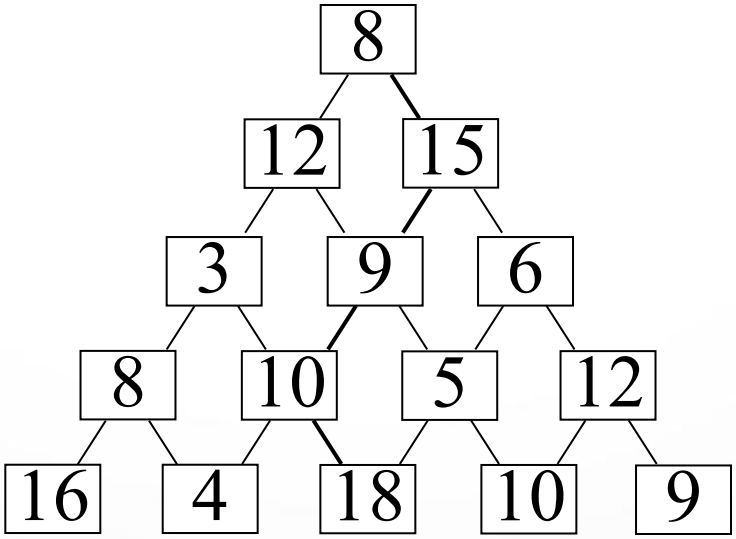
（1）划分子问题：将原问题分解为若干个子问题，每个子问题对应一个决策阶段，并且子问题之间具有重叠关系；

（2）确定动态规划函数：根据子问题之间的重叠关系找到子问题满足的递推关系式（即动态规划函数），这是动态规划法的关键；

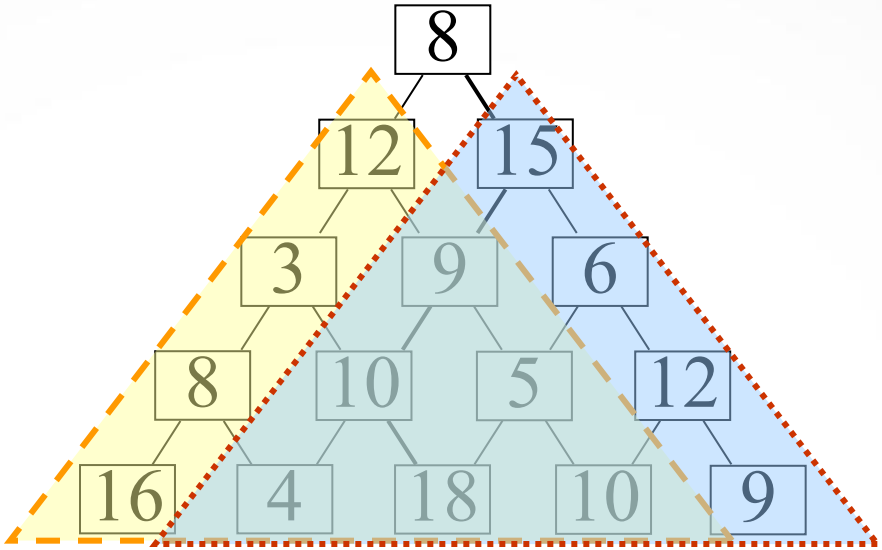
（3）填写表格：设计表格，以自底向上的方式计算各个子问题的解并填表，实现动态规划过程。

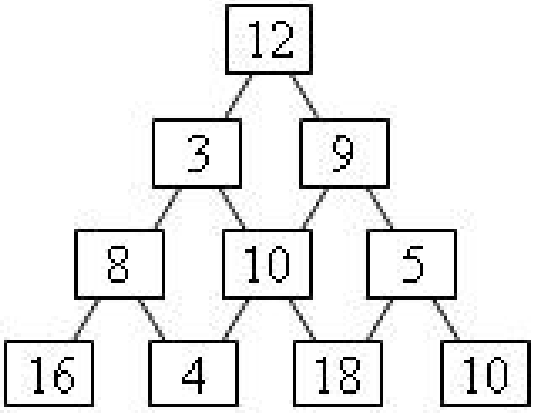
一个简单的例子——数塔问题

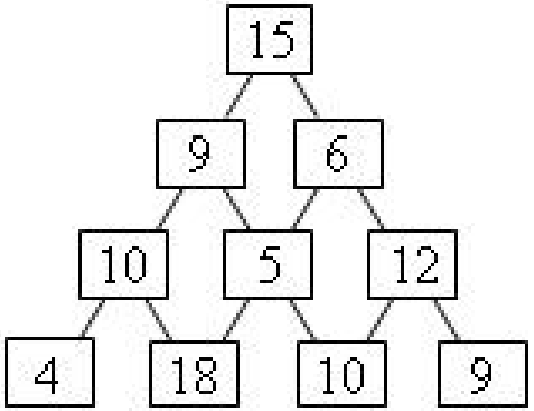
问题描述：从数塔的顶层出发，在每一个结点可以选择向左走或向右走，一直走到最底层，要求找出一条路径，使得路径上的数值和最大。



数塔问题——想法


8
12
15
3
9
6
8
10
5
12
16
4
18
10
9


12
3
9
8
10
5
16
4
18
10


15
9
6
10
5
12
4
18
10
9

数塔问题——想法

第 1 层的决策	8+max{49,52} = 60				
第 2 层的决策	12+max{31,37} = 49	15+max{37,29} = 52			
第 3 层的决策	3+max{24, 28} = 31	9+max{28,23} = 37	6+max{23,22} = 29		
第 4 层的决策	8+max{16,4} = 24	10+max{4,18} = 28	5+max{18,10} = 23	12+ max{10,9} = 22	
初始化	16	4	18	10	9

自底向上填写

求解初始子问题：底层的每个数字可以看作1层数塔，则最大数值和就是其自身；

再求解下一阶段的子问题：第4层的决策是在底层决策的基础上进行求解，可以看作4个2层数塔，对每个数塔进行求解；

再求解下一阶段的子问题：第3层的决策是在第4层决策的基础上进行求解，可以看作3个2层的数塔，对每个数塔进行求解；

以此类推，直到最后一个阶段：第1层的决策结果就是数塔问题的整体最优解。

数塔问题——算法

1. 初始化数组maxAdd的最后一行为数塔的底层数据:

```
for (j = 0; j < n; j++)
    maxAdd[n-1][j] = d[n-1][j];
```

2．从第n-1层开始直到第 1 层对下三角元素maxAdd[i][j]执行下述操作:

2.1 maxAdd[i][j] = d[i][j] + max{maxAdd[i+1][j], maxAdd[i+1][j+1]};

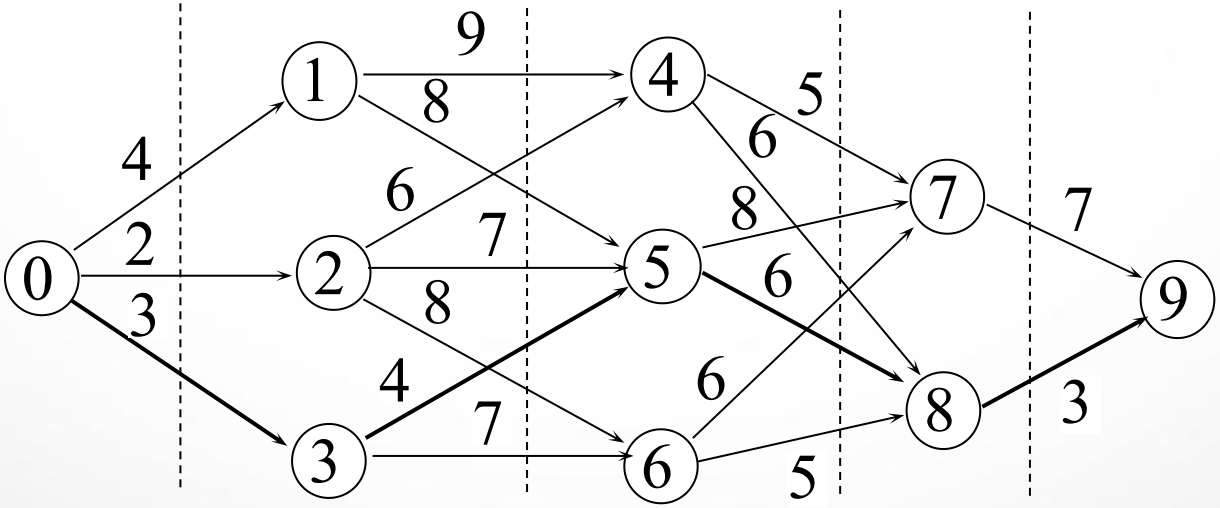
2.2 如果选择下标j的元素，则path[i][j] = j,
否则path[i][j] = j+1;

3. 输出最大数值和maxAdd[0][0];

4. 根据path数组确定每一层决策的列下标，输出路径信息;

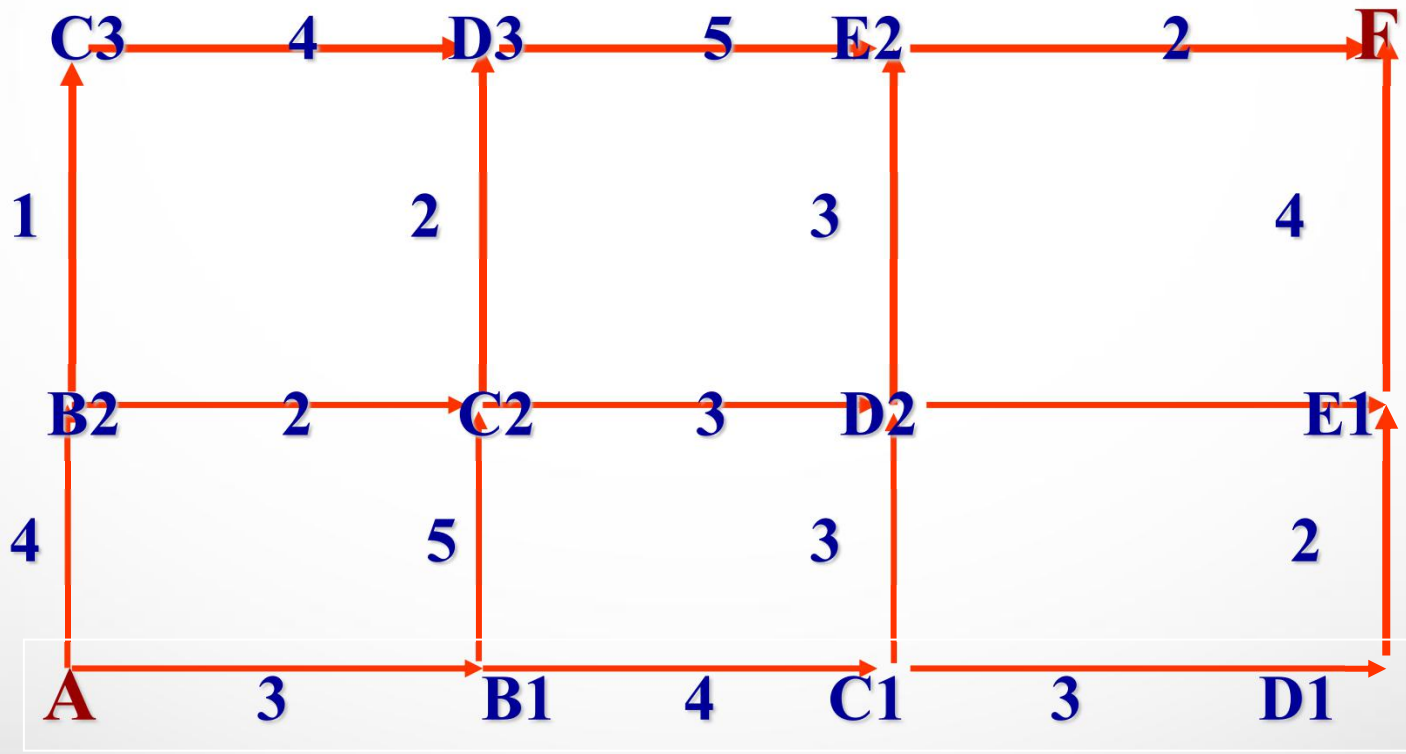
图问题中的动态规划法——多段图最短路径

问题描述：设图$G=(V, E)$是一个带权有向图，如果把顶点集合V划分成k个互不相交的子集V_i $(2\le k\le n, 1\le i\le k)$，使得$E$中的任何一条边$(u, v)$，必有$u\in V_i$，$v\in V_{i+m}$ $(1\le i\le k, 1<i+m\le k)$，则称图$G$为多段图，称$s\in V_1$为源点，$t\in V_k$为终点。多段图的最短路径问题求从源点到终点的最小代价路径。



多段图最短路径——实例

W先生每天驾车去公司上班。W先生的住所位于A，公司位于F，图中的直线段代表公路，交叉点代表路口，直线段上的数字代表两路口之间的平均行驶时间。现在W先生的问题是要确定一条最省时的上班路线。



多段图最短路径——想法

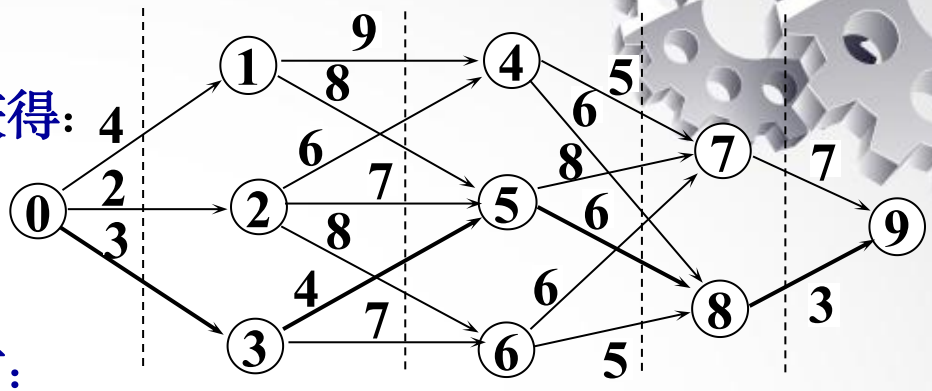
证明多段图的最短路径问题满足最优性原理。

设c_{uv}表示多段图的有向边$<u, v>$上的权值，将从源点s到终点t的最短路径长度记为$d(s, t)$，考虑原问题的部分解$d(s, v)$，显然有下式成立：

$$d(s, v) = c_{sv} \qquad (<s, v> \in E)$$

$$d(s, v) = \min\{d(s, u) + c_{uv}\} \qquad (<u, v> \in E)$$

多段图最短路径实例求解过程



首先求解初始子问题，可直接获得：

$d(0, 1)=c_{01}=4(0\rightarrow1)$

$d(0, 2)=c_{02}=2(0\rightarrow2)$

$d(0, 3)=c_{03}=3(0\rightarrow3)$

再求解下一个阶段的子问题，有：

$d(0, 4)=\min\{d(0, 1)+c_{14}, d(0, 2)+c_{24}\}=\min\{4+9, 2+6\}=8(2\rightarrow4)$

$d(0, 5)=\min\{d(0, 1)+c_{15}, d(0, 2)+c_{25}, d(0, 3)+c_{35}\}=\min\{4+8, 2+7, 3+4\}$
$=7(3\rightarrow5)$

$d(0, 6)=\min\{d(0, 2)+c_{26}, d(0, 3)+c_{36}\}=\min\{2+8, 3+7\}=10(2\rightarrow6)$

再求解下一个阶段的子问题，有：

$d(0, 7)=\min\{d(0, 4)+c_{47}, d(0, 5)+c_{57}, d(0, 6)+c_{67}\}=\min\{8+5, 7+8, 10+6\}$
$=13(4\rightarrow7)$

$d(0, 8)=\min\{d(0, 4)+c48, d(0, 5)+c_{58}, d(0, 6)+c_{68}\}=\min\{8+6, 7+6, 10+5\}$
$=13(5\rightarrow8)$

直到最后一个阶段，有：

$d(0, 9)=\min\{d(0, 7)+c_{79}, d(0, 8)+c_{89}\}=\min\{13+7, 13+3\}=16(8\rightarrow9)$

再将状态进行回溯，得到最短路径0→3→5→8→9，最短路径长度16。

多段图最短路径实例求解过程

多段图最短路径问题的填表过程

下标	1	2	3	4	5	6	7	8	9
元素值	4	2	3	8	7	10	13	13	16
状态转移	0→1	0→2	0→3	2→4	3→5	2→6	4→7	5→8	8→9

回溯求路径

cost

path

下标	1	2	3	4	5	6	7	8	9
距离	4	2	3	8	7	10	13	13	16
路径	0	0	0	2	3	2	4	5	8

多段图最短路径——算法

算法6.2：多段图的最短路径问题 2^(n-1)

输入：多段图的代价矩阵

输出：最短路径长度及路径

1. 循环变量j从1~n-1重复下述操作，执行填表工作：
   1.1 考察顶点j的所有入边，对于边$(i, j) \in E$：
      1.1.1 cost[j]=min{cost[i]+c_{ij}};
      1.1.2 path[j]=使cost[i]+c_{ij}最小的i;
   1.2 j++;
2. 输出最短路径长度cost[n-1];
3. 循环变量i=path[n-1]，循环直到path[i]=0：
   3.1 输出path[i];
   3.2 i=path[i];

多段图最短路径——算法

动态规划法，为什么快？

因为最小的子问题先计算，然后保存起来。

总的时间：子问题的个数

动态规划表：因为子问题有重叠，而现在所有的子问题只计算一次。所以说，空间换时间

$\sum_{i=0}^{n-2}\sum_{j=0}^{i}1=\sum_{i=0}^{n-2}(i+1)$　　1,1,1,…,1 （有i-0+1)个 “1”

多段图的单源点的最短路径问题

1、定义子问题。2、找出初始子问题。0阶子问题。

3、怎么用0阶子问题的解，来求1阶子问题的解

$d(vi,vj,\{v0\})=\min\{d(vi,vj,\{\}),\ d(vi,v0,\{\})+d(v0,vj,\{\})\}$

多段图最短路径——算法

多段图的单源点的最短路径问题

4、一般化。怎么用n阶子问题的解，来求n+1阶子问题的解。动态规划递推方程。

$$d(vi,vj,\{v0,v1,\ldots,v_n\}) =$$

$$\min \{d(vi,vj,\{v0,v1,\ldots,v_{n-1}\})$$

$$d(vi,v_n,\{v0,v1,\ldots,v_{n-1}\})+d(v_n,vj,\{v0,v1,\ldots,v_{n-1}\})\}$$

5、怎么填表

6、根据填表过程，写出伪代码

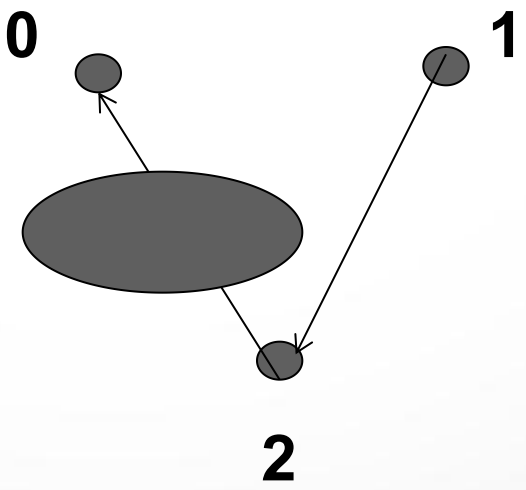
3、怎么用0阶子问题的解，来求1阶子问题的解

$d(vi,vj,\{v0\}) = \min \{d(vi,vj,\{\}),\ d(vi,v0,\{\})+d(v0,vj,\{\})\}$

图问题中的动态规划法——TSP问题

TSP问题是指旅行家要旅行*n*个城市，要求各个城市经历且仅经历一次然后回到出发城市，并要求所走的路程最短。

d(i,j,V')



TSP问题

1、定义子问题。d(j,V')，从顶点j，经过顶点集合V'中所有顶点仅一次，回到出发点0的最短距离

2、找出初始子问题。0阶子问题。 d(0, {1})

d(1,{ })，d(2,{ })，…，d(n-1,{ })，d(0,{})无意义

3、怎么用0阶子问题的解，来求1阶子问题的解

d(1,{1})无意义（因为d(1,{1}) = d(1,{ }) ），

d(1,{2})，d(1,{3})，…，d(1,{n-1})

d(2,{1})，d(2,{2})，…，d(2,{n-1})

d(n-1,{1})，d(n-1,{2})，…，d(n-1,{n-1})

d(1,{2}) = c12+d(2,{ })，…，d(1,{n-1}) = $c_{1,n-1}$+d(n-1,{})

d(1,V')= min {c1x+d(x, {V'-x})|x=2,3,…,n-1}

TSP问题

d(1,{2,3}) = min{c12+d(2,{3})，c13+d(3,{2})

d(1,V')= min {c1x+d(x, {V'-x})|x=2,3,…,n-1}

最后阶子问题

d(0,{1,2,3}) = min {c01+d(1,{2,3}) ， c02+d(2,{1,3} ， c03+d(3,{1,2}) }

TSP问题，蛮力法O(n!)，动态规划法O(2^{n-1})

TSP问题——想法

TSP问题满足最优性原理。

设s,s1,s2,…,sn,s是从s出发的一条最短路径长度的简单回路。

假设从s到下一个城市s1已经求出，则问题转化为求从s1到s的最短路径。

显然，s1,s2,…,sn,s一定构成一条从s1到s的最短路径。

如若不然，就会推出s,s1,s2,…,sn,s不是从s出发的一条最短路径。矛盾

TSP问题——想法

如何定义子问题?

对于图$G=(V, E)$，假设从顶点i出发，令$V'=V-i$，则$d(i, V')$表示从顶点i出发经过V'中各个顶点一次且仅一次，最后回到出发点i的最短路径长度。显然，初始子问题是$d(k, \{\})$，即从顶点i出发只经过顶点k回到顶点i。

现在考虑原问题的一部分，$d(k, V'-\{k\})$表示从顶点k出发经过$V'-\{k\}$中各个顶点一次且仅一次，最后回到出发点i的最短路径长度，则:

$$d(k, \{\})=c_{ki}$$

$$d(i, V')=\min\{c_{ik} + d(k, V'-\{k\})\} \quad (k \in V')$$

TSP问题——实例

∞	3	6	7
5	∞	2	3
6	4	∞	2
3	7	5	∞

首先计算初始子问题，可以直接获得：$d(1, \{\})= c_{10}$ =5(1→0);

$d(2, \{\})= c_{20}$ =6(2→0);　$d(3, \{\})= c_{30}$ =3(3→0)

再求解下一个阶段的子问题，有：

$d(1, \{2\})= c_{12}+d(2, \{\})$=2+6=8(1→2);　$d(1, \{3\})= c_{13}+d(3, \{\})$=3+3=6(1→3)

$d(2, \{1\})= c_{21}+d(1, \{\})$=4+5=9(2→1);　$d(2, \{3\})= c_{23}+d(3, \{\})$=2+3=5(2→3)

$d(3, \{1\})= c_{31}+d(1, \{\})$=7+5=12(3→1);　$d(3, \{2\})= c_{32}+d(2, \{\})$=5+6=11(3→2)

再求解下一个阶段的子问题，有：

$d(1, \{2, 3\})=\min\{c_{12}+d(2, \{3\}), c_{13}+ d(3, \{2\})\}=\min\{2+5, 3+11\}=7(1\rightarrow2)$

$d(2, \{1, 3\})=\min\{c_{21}+d(1, \{3\}), c_{23}+ d(3, \{1\})\}=\min\{4+6, 2+12\}=10(2\rightarrow1)$

$d(3, \{1, 2\})=\min\{c_{31}+d(1, \{2\}), c_{32}+ d(2, \{1\})\}=\min\{7+8, 5+9\}=14(3\rightarrow2)$

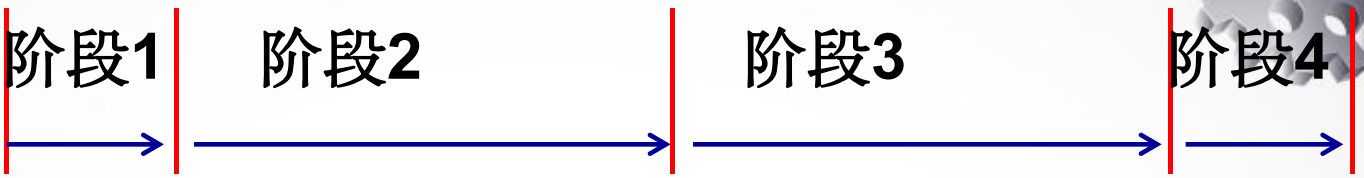
直到最后一个阶段，有：

$d(0, \{1, 2, 3\})=\min\{c_{01}+ d(1, \{2, 3\}), c_{02}+ d(2, \{1, 3\}), c_{03}+ d(3, \{1, 2\})\}$

$=\min\{3+7, 6+10, 7+14\}=10(0\rightarrow1)$

所以，从顶点0出发的TSP问题的最短路径长度为10，再将状态进行回溯，得到最短路径是0→1→2→3→0。

TSP问题——实例（填表过程）



出发点	{ }	{1}	{2}	{3}	{1, 2}	{1, 3}	{2, 3}	{1, 2, 3}
0								10 0→1
1	5 1→0		8 1→2	6 1→3			7 1→2	
2	6 2→0	9 2→1		5 2→3		10 2→1		
3	3 3→0	12 3→1	11 3→2		14 3→2			

TSP问题——算法

假设n个顶点分别用0~n-1的数字编号，顶点之间的代价存放在数组arc[n][n]中。
下面考虑从顶点0出发，求解TSP问题的填表形式。
1. 首先按1、2、…、n-1的顺序，生成1~n-1个元素的子集，存放在数组V[2^(n-1)]中。
例如，当n=4时，V[1]={1}，V[2]={2}，V[3]={3}，V[4]={1,2}，V[5]={1,3}，V[6]={2,3}，V[7]={1,2,3}
2. 设数组d[n][2^(n-1)]存放迭代结果。其中，d[i][j]表示从顶点i经过子集V[j]中的顶点一次且一次，最后回到出发点0的最短路径长度。

TSP问题——算法

TSP问题： Ω(2^n)
输入： 图的代价矩阵arc[n][n]
输出： 从顶点0出发经过所有顶点一次且仅一次再回到顶点0的最短路径长度

1. 初始化第0列：

```
for (i=1; i<n; i++)
    d[i][0]=c[i][0];
```

2. 依次处理每一个子集数组V[2^{n-1}]

```
for (i=1; i<n; i++)
  if (子集V[j]中不包含i)
    对V[j]中的每个元素k,
      计算d[i][j]=min{arc[i][k]+d[k][j-1]};
```

3. 输出最短路径长度d[0][$2^{n-1}-1$];

组合问题中的动态规划法

——最长递增子序列问题

问题描述：在数字序列$A=\{a_1, a_2, \ldots, a_n\}$中按递增下标序列$(i_1, i_2, \ldots, i_k)$（$1 \le i_1 < i_2 < \ldots < i_k \le n$）顺序选出一个子序列$B$，如果子序列$B$中的数字都是严格递增的，则子序列$B$称为序列$A$的递增子序列。最长递增子序列问题就是要找出序列$A$的一个最长的递增子序列。

最长递增子序列问题——想法

设序列$A=\{a_1, a_2, \ldots, a_n\}$
最长递增子序列是$B=\{b_1, b_2, \ldots, b_m\}$
最长递增子序列问题满足最优性原理。

设$L(n)$为数字序列$A=\{a_1, a_2, \ldots, a_n\}$的最长递增子序列的长度，显然，初始子问题是$\{a_1\}$，即$L(1)=1$。考虑原问题的一部分，设$L(i)$为子序列$A=\{a_1, a_2, \ldots, a_i\}$的最长递增子序列的长度，则满足如下递推式:

$$L(i)=\begin{cases}1 & i=1\text{或不存在}a_j<a_i\ (1\le j<i)\\ \max\{L(j)+1\} & \text{对于所有的}a_j<a_i\ (1\le j<i)\end{cases}$$

最长递增子序列问题——实例

于序列A={5, 2, 8, 6, 3, 6, 9, 7}，用动态规划法求解最长递增子序列。

首先计算初始子问题，可以直接获得：

L(1)=1({5})

然后依次求解下一个阶段的子问题，有：

L(2)=1({2})

L(3)=max{L(1)+1, L(2)+1}=2({5, 8}, {2, 8})

L(4)= max{L(1)+1, L(2)+1}=2({5, 6}, {2, 6})

L(5)=L(2)+1=2({2, 3})

L(6)=max{L(1)+1, L(2)+1, L(5)+1)}=3({2, 3, 6})

L(7)=max{L(1)+1, L(2)+1, L(3)+1, L(4)+1, L(5)+1, L(6)+1}=4({2, 3, 6, 9})

L(8)=max{L(1)+1, L(2)+1, L(4)+1, L(5)+1, L(6)+1}=4({2, 3, 6, 7})

序列A的最长递增子序列的长度为4，有两个最长递增子序列，分别是{2, 3, 6, 9}和{2, 3, 6, 7})。

最长递增子序列问题——实例（填表）

序号	1	2	3	4	5	6	7	8
序列元素	5	2	8	6	3	6	9	7
子序列长度	1	1	2	2	2	3	4	4
递增子序列	{5}	{2}	{5,8},{2,8}	{5,6},{2,6}	{2,3}	{2,3,6}	{2,3,6,9}	{2,3,6,7}

最长递增子序列问题——算法

设序列A存储在数组a[n]中，数组L[n]存储最长递增子序列的长度，其中L[i]表示元素序列a[0]~a[i]的最长递增子序列的长度

二维数组x[n][n]存储对应的最长递增子序列，其中x[i][n]存储a[0]~a[i]的最长递增子序列，注意到数组下标均从0开始，给出用编程语言描述的算法。

组合问题中的动态规划法

——最长公共子序列问题

问题描述：对给定序列$X=(x_1, x_2,\ldots, x_m)$和序列$Z=(z_1, z_2,\ldots, z_k)$，Z是X的子序列当且仅当存在一个递增下标序列$(i_1, i_2,\ldots, i_k)$，使得对于所有$j=1, 2, \ldots, k$，有$zj=xi_j$（$1\le i_j\le m$）

例如，X=(a,b,c,b,d,a,b)，Z=(b,c,d,b)。则Z是X的一个长度为4的子序列，相应的递增下标序列为(2,3,5,7)。

组合问题中的动态规划法

——最长公共子序列问题

给定两个序列*X*和*Y*，当序列*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。最长公共子序列问题就是在序列*X*和*Y*中查找最长的公共子序列。

例如，X=(a,b,c,b,d,b)，Y=(a,c,b,b,a,b,d,b,b)，Z=(a,c,b)。则Z是X和Y的一个长度为3的公共子序列。

组合问题中的动态规划法

满足最优性原理

设序列*X*=(x1,x2,…,xm)，*Y*=(y1,y2,…,yn)，*X*和*Y*的最长公共子序列为Z=(z1,z2,…,zk)。

记Xk为序列X中前k个连续字符组成的子序列，Yk为序列Y中前k个连续字符组成的子序列，Zk为序列Z中前k个连续字符组成的子序列。

则下式成立：

(1)若xm=yn，则zk=xm=yn，且Zk-1是Xm-1和Yn-1的最长公共子序列

(2)若xm≠yn，且zk≠xm，则Zk是Xm-1和Yn的最长公共子序列

(3)若xm≠yn，且zk≠yn，则Zk是Xm和Yn-1的最长公共子序列

最长公共子序列问题——想法

如何定义子问题?

设$L(m, n)$表示序列$X=\{x_1, x_2, \ldots, x_m\}$和$Y=\{y_1, y_2, \ldots, y_n\}$的最长公共子序列的长度，显然，初始子问题是序列$X$和$Y$至少有一个空序列，即:

$L(0, 0)=L(0, j)=L(i, 0)=0 \qquad (1\le i\le m, 1\le j\le n)$

考虑原问题的一部分:

设$L(i, j)$表示子序列X_i和Y_j的最长公共子序列的长度，则有如下动态规划函数:

$$L(i,j)=\begin{cases} L(i-1,j-1)+1 & x_i=y_j, i\ge 1, j\ge 1 \\ \max\{L(i,j-1), L(i-1,j)\} & x_i\ne y_j, i\ge 1, j\ge 1 \end{cases}$$

最长公共子序列问题——实例

为了得到序列*X*m和*Y*n具体的最长公共子序列，设二维表*S*(*m*, *n*)记载求解过程中的状态变化，其中*S*(*i*, *j*)表示在计算*L*(*i*, *j*)时的搜索状态，并且有：

$$S(i,j)=\begin{cases}1 & x_i = y_j \\ 2 & x_i \neq y_j \text{且} L(i,j-1) \geq L(i-1,j) \\ 3 & x_i \neq y_j \text{且} L(i,j-1) < L(i-1,j)\end{cases}$$

若*S*(*i*, *j*)=1，表明$x_i=y_j$，则下一个搜索方向是*S*(*i*-1,*j*-1)；

若*S*(*i*, *j*)=2，表明$x_i \neq y_j$且*L*(i, *j*-1)≥*L*(*i*-1, *j*)，则下一个搜索方向是*S*(*i*, *j*-1)；

若*S*(*i*, *j*)=3，表明*xi*≠*yj*且*L*(*i*, *j*-1) < *L*(*i*-1, *j*)，则下一个搜索方向是*S*(*i*-1, *j*)。

最长公共子序列问题——实例（填表）

例如，序列$X=(a, b, c, b, d, b)$，$Y=(a, c, b, b, a, b, d, b, b)$，动态规划法求解最长公共子序列的过程如下。

			a	*c*	*b*	*b*	*a*	*b*	*d*	*b*	*b*
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
a	1	0	1	1	1	1	1	1	1	1	1
b	2	0	1	1	2	2	2	2	2	2	2
c	3	0	1	2	2	2	2	2	2	2	2
b	4	0	1	2	3	3	3	3	3	3	3
d	5	0	1	2	3	3	3	3	4	4	4
b	6	0	1	2	3	4	4	4	4	5	**5**

(a) 长度矩阵 L

			a	*c*	*b*	*b*	*a*	*b*	*d*	*b*	*b*
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
a	1	0	1	2	2	2	1	2	2	2	2
b	2	0	3	2	1	1	2	1	2	1	1
c	3	0	3	1	2	2	2	2	2	2	2
b	4	0	3	3	1	1	2	1	2	1	1
d	5	0	3	3	3	2	2	2	1	2	2
b	6	0	3	3	1	1	2	1	2	1	1

(b) 状态矩阵 S

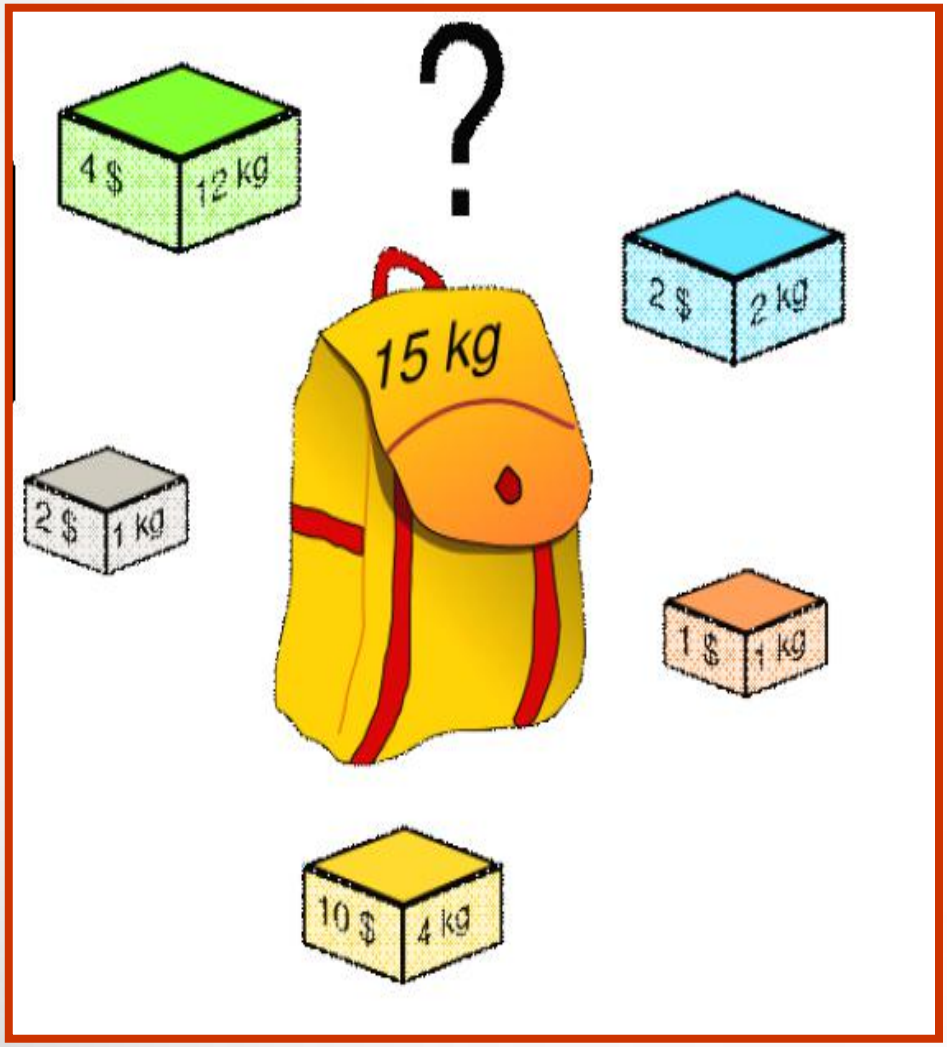
最长公共子序列问题——算法

```
int CommonOrder(char x[ ], int m, char y[ ], int n, char z[ ])
{
    int i, j, k;
    for (j = 0; j <= n; j++)
        L[0][j] = 0;
    for (i = 0; i <= m; i++)
        L[i][0]=0;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
            if (x[i] == y[j]) { L[i][j] = L[i-1][j-1] + 1; S[i][j] = 1; }
            else if (L[i][j-1] >= L[i-1][j]) {
                    L[i][j] = L[i][j-1]; S[i][j] = 2;
                }
                else {L[i][j] = L[i-1][j]; S[i][j] = 3; }
```

最长公共子序列问题——算法

```
    i = m; j = n; k = L[m][n];
    while (i > 0 && j > 0)
    {
        if (S[i][j] == 1) { z[k] = x[i]; k--; i--; j--; }
        else if (S[i][j] == 2) j--;
                else i--;
    }
    for (k = 0; k < L[m][n]; k++)
       cout<<z[k];
    return L[m][n];
}
```

组合问题中的动态规划法——0/1背包问题



给定n种物品和一个背包，物品i的重量是w_i，其价值为v_i，背包的容量为C。背包问题是如何选择装入背包的物品，使得装入背包中物品的总价值最大?如果在选择装入背包的物品时，对每种物品i只有两种选择：装入背包或不装入背包，即不能将物品i装入背包多次，也不能只装入物品i的一部分，则称为0/1背包问题。

0/1背包问题——想法

0/1背包问题满足最优性原理。

0/1背包问题可以看作是决策一个序列$(x_1, x_2, \ldots, x_n)$，对任一变量x_i的决策是决定$x_i=1$还是$x_i=0$。设$V(n, C)$表示将n个物品装入容量为C的背包获得的最大价值，显然，初始子问题是把前面i个物品装入容量为0的背包和把0个物品装入容量为j的背包，得到的价值均为0，即：

$$V(i, 0)= V(0, j)=0 \qquad (0\le i\le n, 0\le j\le C)$$

考虑原问题的一部分，设$V(i, j)$表示将前i（$1\le i\le n$）个物品装入容量为j（$1\le j\le C$）的背包获得的最大价值，在决策x_i时，可采用递推式：

$$V(i,j)=\begin{cases} V(i-1,j) & j<w_i \\ \max\{V(i-1,j),\ V(i-1,j-w_i)+v_i\} & j\ge w_i \end{cases}$$

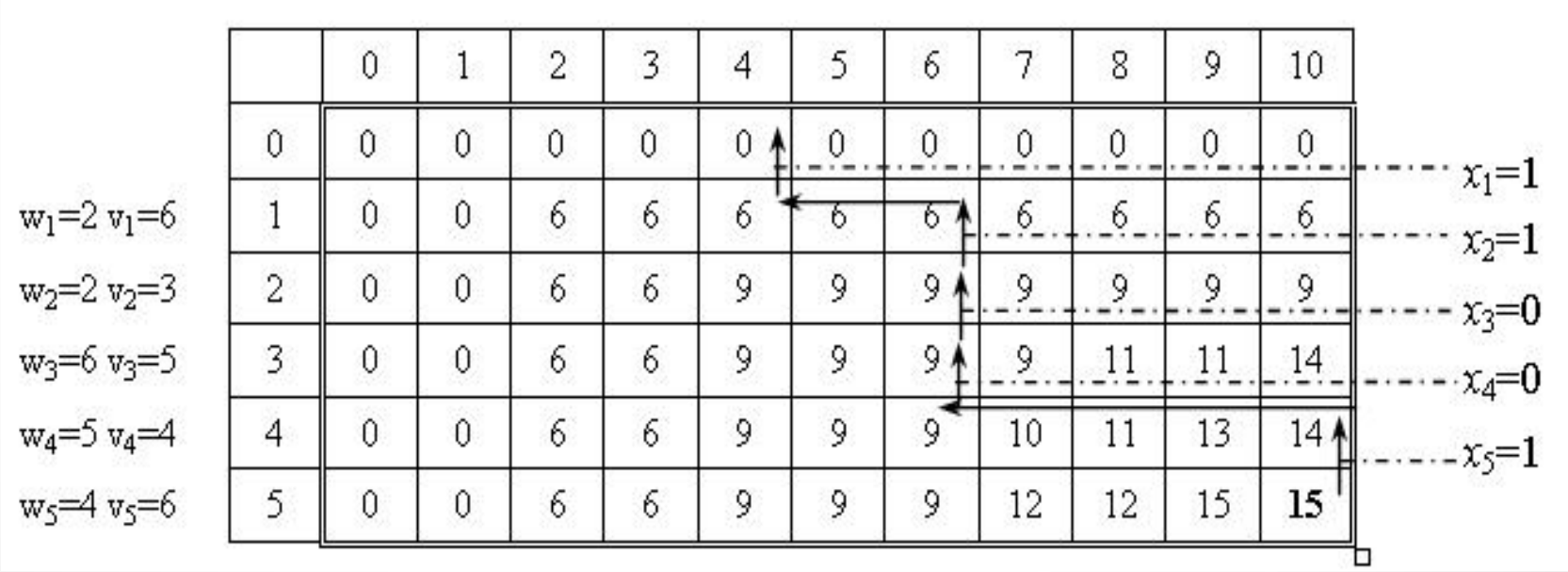
0/1背包问题——实例

为了确定装入背包的具体物品，从$V(n,C)$的值向前推，如果$V(n,C)>V(n\text{-}1,C)$，表明第n个物品被装入背包，前n-1个物品被装入容量为$C\text{-}w_n$的背包中；否则，第n个物品没有被装入背包，前n-1个物品被装入容量为C的背包中。依此类推，直到确定第1个物品是否被装入背包中为止。由此，得到如下函数：

$$x_i=\begin{cases}0 & V(i,j)=V(i-1,j)\\ 1,\quad j=j-w_i & V(i,j)>V(i-1,j)\end{cases}$$

0/1背包问题——实例

有5个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为10，动态规划法求解0/1背包问题。



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

查找问题中的动态规划法
——最优二叉查找树

设$\{r_1, r_2, \cdots, r_n\}$是$n$个记录的集合，其查找概率分别是$\{p_1, p_2, \cdots, p_n\}$，最优二叉查找树是以这$n$个记录构成的二叉查找树中具有最少平均比较次数的二叉查找树，即$\sum_{i=1}^{n} p_i \times c_i$最小，其中$p_i$是记录$r_i$的查找概率，$c_i$是在二叉查找树中查找$r_i$的比较次数。

最优二叉查找树——想法

最优二叉查找树是否满足最优性原理?

思考如何定义子问题?

动态规划函数?

自己描述实例、算法。

小结——适用条件

动态规划法的有效性依赖于问题本身所具有的两个重要的性质：

1.最优子结构。如果问题的最优解是由其子问题的最优解来构造，则称该问题具有最优子结构性质。

2.重叠子问题。在用递归算法自顶向下解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。

动态规划 算法正是利用了这种子问题的重叠性质，对每一个子问 题只解一次，而后将其解保存在一个表格中，在以后该子问题的求解时直接查表。

小结——与其他算法比较

动态规划的思想实质是分治思想和解决冗余。
与分治法类似的是将原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。

与分治法不同的是经分解的子问题往往不是互相独立的。若用分治法来 解，有些共同部分（子问题或子子问题）被重复计算。

如果能够保存已解决的子问题的答案，在需要时再查找，这样就可以避免重复计算、节省时间。动态规划法用一个表来记录所有已解的子问题的答案。这就是动态规划法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表方式。

小结——其他应用领域

生产与存储问题

某工厂每月需供应市场一定数量的产品。供应需求所剩余产品应存入仓库，一般地说，某月适当增加产量可降低生产成本，但超产部分存入仓库会增加库存费用，要确定一个每月的生产计划，在满足需求条件下，使一年的生产与存储费用之和最小。

小结——其他应用领域

投资决策问题

某公司现有资金Q亿元，在今后5年内考虑给A、B、C、D四个项目投资，这些项目的投资期限、回报率均不相同，问应如何确定这些项目每年的投资额，使到第五年末拥有资金的本利总额最大。

小结——其他应用领域

设备更新问题

企业使用设备都要考虑设备的更新问题，因为设备越陈旧所需的维修费用越多，但购买新设备则要一次性支出较大的费用。现在某企业要决定一台设备未来8年的更新计划，已预测到第j年购买设备的价格为K_j，G_j为设备经过j年后的残值，C_j为设备连续使用j−1年后在第j年的维修费用(j=1,2…8)，问应在哪年更新设备可使总费用最小。

- 作业
- P124-125：
- T3，T4，T7，T8
- T3：有5个物品，其重量分别为（3,2,1,4, 5），价值分别为（25,20,15,40,50）。背包容量为5。写出动态规划法求解的过程。
- T4：用动态规划法求解两个字符串A='xzyzzyx'，B='zxyyzxz'的最长公共子序列。写出求解过程。

- 作业
- P124-125：
- T3，T4，T7，T8
- T7：Ackermann函数A(m,n)的递归定义如下
- A(m,n)=n+1，当m=0
- A(m,n)=A(m-1,n)，当m>0，n=0
- A(m,n)=A(m-1,A(m,n-1))，当m>0，n>0
- 设计动态规划算法计算A(m,n)。