# 第8章 《动态规划法》习题

1. 简述动态规划算法的基本思想、使用动态规划算法求解问题的基本步骤。举例说明动态规划算法适用的条件或场景。

　　基本思想：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。与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的。

　　基本步骤：

（1）分析最优解的性质，并刻画其最优子结构特征

（2）确定状态表示S(x1,x2,...)和状态递推方程，递归地定义最优值；

（3）根据状态转移顺序，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，构造最优解。

适用条件：

问题必须满足最优化原理和无后效性才适用动态规划。

最优化原理（最优子结构性质）：一个最优化策略的子策略总是最优的。所以不管过去的过程如何，只从当前的状态和系统的最优化要求出发，作出下一步的最优决策。

无后效性：对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策。

2. 比较动态规划法和分治法：

（1）动态规划法和分治法有什么共同特点？

（2）这两种技术之间有什么主要的不同点？

（1）：二者都要求原问题具有最优子结构性质,都是将原问题分而治之,分解成若干个规模较小(小到很容易解决的程序)的子问题.然后将子问题的解合并,形成原问题的解.

（2）：分治法将分解后的子问题看成相互独立的，通过用递归来做。动态规划将分解后的子问题理解为相互间有联系,有重叠部分，需要记忆，通常用迭代来做。

3. 对于下面具有权重矩阵的有向图，使用Floyd算法求解所有点对之间的最短路径问题，给出求解过程。

**核心代码：**

for u ← 0：N

for v ← 0：N

for w ← 0：N

if(D[v][u] + D[u][w] < D[v][w])// 从v经u到w的一条路径更短

D[v][w] = D[v][u] + D[u][w];

end if

end for

end for

end for

4. 改进Floyd算法，使得该算法能够求出最短路径本身，而不仅仅是它们的长度。

**解决方法：增加一个P数组，用来存放前一个顶点的下标**

for u ← 0：N

for v ← 0：N

for w ← 0：N

if D[v][w]>D[v][u]+D[u][w]

D[v][w]←D[v][u]+D[u][w];

P[v][w]←P[v][u];

end if

end for

end for

end for

**初始化代码**：

for v ← 0: N

for w ← 0: N

D[v][w]←G.arc[v][w];

p[v][w]←w;

end for

end for

5.（1）对于下面背包问题的实例，背包容量*W*=6，使用动态规划算法求解，给出计算过程。

|  |  |  |
| --- | --- | --- |
| 物品 | 重量 | 价值（元） |
| 1 | 3 | 25 |
| 2 | 2 | 20 |
| 3 | 1 | 15 |
| 4 | 4 | 40 |
| 5 | 5 | 50 |

（2）（1）中的实例有多少个不同的最优子集？

（3）一般来说，如何从动态规划算法所生成的表中判断出背包问题的实例是不是具有不止一个最优子集？

（1）代码如下：

public static int[][] Solution(int[] weight, int[] value,int maxWeight){

int n = weight.length;//物品个数

int [][]dp = new int[n+1][maxWeight+1];

dp[0][0] = 0;

for(int i = 1; i < n+1; i++){

for(int j = 1; j < maxWeight+1; j++){

if( j - weight[i-1] >= 0 ){

dp[i][j] = max(dp[i-1][j],value[i-1]+dp[i-1][j-weight[i-1]]);

}else{

dp[i][j] = dp[i-1][j];

}

}

}

return dp;

}

过程讲解：

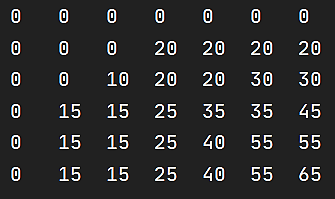
　　我们先考虑由前i个物品定义的实例，物品的重量是weight[1:i]，价值为value[1:i],背包承重量为j，dp(i,j)代表该实例的最优解。

１、在不包括第i个物品的子集中，最优子集的价值是dp(i-1,j)

2、在包括第i个物品的子集中（j – wi >= 0）,最优子集是由该物品和前i-1个物品中能够放进承重量为j-wi的背包的最优子集构成。这种最优子集的总价值为value[i] + dp(i-1,j-wi).所以得到递推式：

dp(i,j) =

（2）求得的dp数组如下:



所以，本实例只有一个最优子集

（3）对于dp[][]数组中的所有值，除了dp[5][6]本身，如果有和它数值相等的地方，那么可说明，该实例不只具有一个最优子集。

6. 为找零问题设计一个动态规划算法：给定金额*n*和各种面额*d*1, *d*2, …, *dm*的数量无限的硬币，求总金额等于*n*的硬币的最少个数，或者指出该问题无解。

public **static** **int** charge**(int[]** value**,** **int** target**)** **{**

**int** n **=** value**.**length**;**

**int[][]** dp **=** new **int[**n **+** 1**][**target **+** 1**];**

**//找零金额为 0 时，不需要找零**

**for** **(int** i **=** 0**;** i **<=** n**;** i**++)**

dp**[**i**][**0**]** **=** 0**;**

**for** **(int** i **=** 0**;** i **<=** target**;** i**++)**

dp**[**0**][**i**]** **=** Integer**.**MAX\_VALUE**;**

**for** **(int** money **=** 1**;** money **<=** target**;** money**++)** **{**

**for** **(int** i **=** 1**;** i **<=** n**;** i**++)** **{**

**//找零金额小于当前硬币面值**

**if** **(**money **<** value**[**i **-** 1**])** **{**

dp**[**i**][**money**]** **=** dp**[**i **-** 1**][**money**];**

**continue;**

**}**

**int** numberByi **=** dp**[**i **-** 1**][**money**];**

**int** numberNotByi **=** dp**[**i**][**money **-** value**[**i **-** 1**]]** **+** 1**;**

**//选较小的**

dp**[**i**][**money**]** **=** Math**.**min**(**numberByi**,** numberNotByi**);**

**}**

**}**

**return** dp**[**n**][**target**];**

**}**

7. 某国际航空公司在世界范围有个国际机场。第个国际机场到中心机场的距离为,。从国际机场到国际机场飞行费用为,为地面加油费用。从任何国际机场飞往中心机场的飞机可以在任一国际机场加油后继续飞行。飞机加油问题要求确定从距中心机场最远的国际机场飞到中心机场的最少费用。设计飞机加油问题的动态规划算法，并分析其时间复杂性。

**//初始化**

**for** i ← 1 **:** n

**for** j ← 1 **:** n

w**(**i**,**j**)** ← s **+** **(**d**(**j**)-**d**(**i**))^**2

end **for**

end **for**

**//两两间最短距离**

**for** k ← 1 **:** n

**for** i ← 1 **:** n

**for** j ← 1 **:** n

**if(**w**[**i**][**k**]** **+** w**[**k**][**j**]** **<** w**[**i**][**wj**])**

w**[**i**][**j**]** **=** w**[**i**][**k**]** **+** w**[**k**][**j**];**

end **if**

end **for**

end **for**

end **for**

**//计算到中心机场的最短距离**

min **=** MAX\_VALUE

**for** i ← 1 **:** n

**//设最远的机场为:d(m)**

**if(** d**(**m**,**i**)** **+** s **+** d**(**i**)^**2 **<** min**)**

min **=** d**(**m**,**i**)** **+** s **+** d**(**i**)^**2

end **if**

end **for**