第8章 动态规划

*Steven*

## 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：

将待求解问题分解成若干个子问题，不过子问题之间通常有重叠部分，故使用数组或其他类似结构保存子问题的答案。当需要使用该子问题的解时可以直接调用而不是重复计算，提高了效率。

### 基本步骤：

1. 找出最优解的性质，并刻画其结构特征。
2. 递归地定义最优值。
3. 自底向上计算出最优值并保存在数组中，在需要子问题最优值时候直接调用。
4. 根据计算最优值时得到的信息，构造最优解。

### 适用场景：

求解最值问题的方案，例如最小费用、最短路径、最高价值等

## 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？**

1. **这两种技术之间有什么主要的不同点？**

动态规划和分治法都是使用分治思想，将待求解问题分解成若干子问题。

分治法是自顶向下地构造递推关系，再自底向上递归求解。这种求解方式下，父问题的解来自哪几个子问题是确定的，也就是静态的。

动态规划解决的是当子问题有重叠时分治法效率低下的问题，通过数组存储子问题解，当父问题需要某一子问题的解时不必重复计算，提高效率。

## 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。

|  |
| --- |
| 代码 |
| Adj = np.array([[0, 2, inf, 1, 8],                  [6, 0, 3, 2, inf],                  [inf, inf, 0, 4, inf],                  [inf, inf, 2, 0, 3],                  [3, inf, inf, inf, 0]])  def Floyd(Adj):      dist = Adj  # dist是最短路径长度矩阵      for k in range(len(Adj)):          for i in range(len(Adj)):              for j in range(len(Adj)):                  if dist[i][k] + dist[k][j] < dist[i][j]:                      dist[i][j] = dist[i][k] + dist[k][j]                      print("在求解从", i, "到", j, "的最短路径时，发现dist[", i, ",", k, "] + dist[", k, ",", j, "]=", dist[i][k] + dist[k][j], " < 当前的dist[", i, ",", j, "]=", dist[i][j], sep='')                      print("  故将dist[", i, "][", j, "]从", dist[i][j], "转为dist[", i, ",", k, "] + dist[", k, ",", j, "]=", dist[i][k] + dist[k][j], sep="")      return dist |
| 结果 |
| 在求解从1到4的最短路径时，发现dist[1,0] + dist[0,4]=14.0 < 当前的dist[1,4]=14.0  故将dist[1][4]从14.0转为dist[1,0] + dist[0,4]=14.0  在求解从4到1的最短路径时，发现dist[4,0] + dist[0,1]=5.0 < 当前的dist[4,1]=5.0  故将dist[4][1]从5.0转为dist[4,0] + dist[0,1]=5.0  在求解从4到3的最短路径时，发现dist[4,0] + dist[0,3]=4.0 < 当前的dist[4,3]=4.0  故将dist[4][3]从4.0转为dist[4,0] + dist[0,3]=4.0  在求解从0到2的最短路径时，发现dist[0,1] + dist[1,2]=5.0 < 当前的dist[0,2]=5.0  故将dist[0][2]从5.0转为dist[0,1] + dist[1,2]=5.0  在求解从4到2的最短路径时，发现dist[4,1] + dist[1,2]=8.0 < 当前的dist[4,2]=8.0  故将dist[4][2]从8.0转为dist[4,1] + dist[1,2]=8.0  在求解从0到2的最短路径时，发现dist[0,3] + dist[3,2]=3.0 < 当前的dist[0,2]=3.0  故将dist[0][2]从3.0转为dist[0,3] + dist[3,2]=3.0  在求解从0到4的最短路径时，发现dist[0,3] + dist[3,4]=4.0 < 当前的dist[0,4]=4.0  故将dist[0][4]从4.0转为dist[0,3] + dist[3,4]=4.0  在求解从1到4的最短路径时，发现dist[1,3] + dist[3,4]=5.0 < 当前的dist[1,4]=5.0  故将dist[1][4]从5.0转为dist[1,3] + dist[3,4]=5.0  在求解从2到4的最短路径时，发现dist[2,3] + dist[3,4]=7.0 < 当前的dist[2,4]=7.0  故将dist[2][4]从7.0转为dist[2,3] + dist[3,4]=7.0  在求解从4到2的最短路径时，发现dist[4,3] + dist[3,2]=6.0 < 当前的dist[4,2]=6.0  故将dist[4][2]从6.0转为dist[4,3] + dist[3,2]=6.0  在求解从2到0的最短路径时，发现dist[2,4] + dist[4,0]=10.0 < 当前的dist[2,0]=10.0  故将dist[2][0]从10.0转为dist[2,4] + dist[4,0]=10.0  在求解从2到1的最短路径时，发现dist[2,4] + dist[4,1]=12.0 < 当前的dist[2,1]=12.0  故将dist[2][1]从12.0转为dist[2,4] + dist[4,1]=12.0  在求解从3到0的最短路径时，发现dist[3,4] + dist[4,0]=6.0 < 当前的dist[3,0]=6.0  故将dist[3][0]从6.0转为dist[3,4] + dist[4,0]=6.0  在求解从3到1的最短路径时，发现dist[3,4] + dist[4,1]=8.0 < 当前的dist[3,1]=8.0  故将dist[3][1]从8.0转为dist[3,4] + dist[4,1]=8.0  [[ 0. 2. 3. 1. 4.]  [ 6. 0. 3. 2. 5.]  [10. 12. 0. 4. 7.]  [ 6. 8. 2. 0. 3.]  [ 3. 5. 6. 4. 0.]] |

## 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。

|  |
| --- |
| 代码 |
| Adj = np.array([[0, 2, inf, 1, 8],                  [6, 0, 3, 2, inf],                  [inf, inf, 0, 4, inf],                  [inf, inf, 2, 0, 3],                  [3, inf, inf, inf, 0]])  def Floyd(Adj):      dist = Adj  # dist是最短路径长度矩阵      path = np.zeros((len(Adj), len(Adj)))  # path是最短路径矩阵      for k in range(len(Adj)):          for i in range(len(Adj)):              for j in range(len(Adj)):                  if dist[i][k] + dist[k][j] < dist[i][j]:                      dist[i][j] = dist[i][k] + dist[k][j]                      path[i][j] = k      return dist, path  res = Floyd(Adj)  print(res[0])  print(res[1]) |
| 结果 |
| [[ 0. 2. 3. 1. 4.]  [ 6. 0. 3. 2. 5.]  [10. 12. 0. 4. 7.]  [ 6. 8. 2. 0. 3.]  [ 3. 5. 6. 4. 0.]]  [[0. 0. 3. 0. 3.]  [0. 0. 0. 0. 3.]  [4. 4. 0. 0. 3.]  [4. 4. 0. 0. 0.]  [0. 0. 3. 0. 0.]] |

## 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）以上（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？**

### 笔算结果就是逐行填表，结果如下：

**①第0行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

当考虑物品0，即不考虑任何物品时，无论背包容量多少，价值自然始终为0。

**②第1行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |

当背包容量为3的时候，能够容纳下物品1，则将物品1放入背包，价值由0变为25。

**③第2行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 2 | 0 | 0 | 20 | 25 | 25 | 45 | 45 |

当背包容量为2的时候，能够容纳下物品2。即此时,且，，所以。

当背包容量为3的时候，能够容纳下物品1。即此时，且，，所以。

当背包容量为5的时候，能够在容纳下物品1的基础上容纳下物品2。即此时，且，，所以。

**④第3行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 2 | 0 | 0 | 20 | 25 | 25 | 45 | 45 |
| 3 | 0 | 15 | 20 | 35 | 40 | 45 | 60 |

当背包容量为1的时候，能够容纳下物品3。即此时,且，，所以。

当背包容量为2的时候，能够容纳下物品2。即此时,且，，所以。

当背包容量为3的时候，能够在容纳下物品3的基础上容纳下物品2。即此时，且，，所以。

当背包容量为4的时候，能够在容纳下物品1的基础上容纳下物品3。即此时，且，，所以。

当背包容量为5的时候，能够在容纳下物品3的基础上容纳下物品4。即此时，且，，所以。

当背包容量为6的时候，能够在容纳下物品1和物品2的基础上容纳下物品3。即此时，且，，所以。

**⑤第4行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 2 | 0 | 0 | 20 | 25 | 25 | 45 | 45 |
| 3 | 0 | 15 | 20 | 35 | 40 | 45 | 60 |
| 4 | 0 | 15 | 20 | 35 | 40 | 55 | 60 |

当背包容量为1的时候，能够容纳下物品3。即此时,且，，所以。

当背包容量为2的时候，能够容纳下物品2。即此时,且，，所以。

当背包容量为3的时候，能够在容纳下物品3的基础上容纳下物品2。即此时，且，，所以。

当背包容量为4的时候，能够在容纳下物品1的基础上容纳下物品3。即此时，且，，所以。

当背包容量为5的时候，能够在容纳下物品3的基础上容纳下物品4。即此时，且，，所以。

当背包容量为6的时候，能够在容纳下物品2的基础上容纳下物品4。即此时，且，，所以。

**⑥第5行：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 背包容量  物品 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 25 | 25 | 25 | 25 |
| 2 | 0 | 0 | 20 | 25 | 25 | 45 | 45 |
| 3 | 0 | 15 | 20 | 35 | 40 | 45 | 60 |
| 4 | 0 | 15 | 20 | 35 | 40 | 55 | 60 |
| 5 | 0 | 15 | 20 | 35 | 40 | 55 | 65 |

当背包容量为1的时候，能够容纳下物品3。即此时,且，，所以。

当背包容量为2的时候，能够容纳下物品2。即此时,且，，所以。

当背包容量为3的时候，能够在容纳下物品3的基础上容纳下物品2。即此时，且，，所以。

当背包容量为4的时候，能够在容纳下物品1的基础上容纳下物品3。即此时，且，，所以。

当背包容量为5的时候，能够在容纳下物品3的基础上容纳下物品4。即此时，且，，所以。

当背包容量为6的时候，能够在容纳下物品3的基础上容纳下物品5。即此时，且，，所以。

### 15个

### 一般来说，可以通过判断表中最后一列的最大值个数来判断，因为背包问题的最优值的产生只会在最后一列产生。

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

|  |
| --- |
| 代码 |
| def get\_change(d, money):      """最小零钱数问题的动态规划解法      Args:          d (list): 零钱的面额，并且已经递增排序          money (int): 应找零的金额      """      n = zeros(1, money + 1)      n[1] = 1 # 初始化      for i in range(2, money):          j = 1          while d[j] < i:              n[i] = min(n[i], n[i-d[j]]+1) # 状态转移函数              j += 1      return n[money] |

## 7． 某国际航空公司在世界范围有国际机场。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。

可以看出，本题假设中心机场和国际机场均排列在一条直线上，且中心机场位于直线的一端。

|  |
| --- |
| def min\_cost(dist, n, s):      """最小费用的动态规划解法      Args:          dist (list):各国际机场离中心机场的距离，并假定已升序排列          n (int):国际机场数量          s (int):地面油费用      """      # 初始化      cost = [inf]\*(n+1)      cost[0] = 0      cost[1] = s+pow(dist[1], 2)      for i in range(2, n+1):          for j in range(1, i):              cost[i] = min(cost[i], cost[j]+s+pow(dist[i]-dist[j], 2))      return cost[n]  # 距离升序排列，那么cost[n]就是距离最远的机场的费用 |