北京理工大學
BEIJING INSTITUTEOF TECHNOLOGY

第8章 动态规划

动态规划

- 动态规划（dynamic programming）是一种算法设计技术，是一种多阶段决策过程最优的通用方法。
- 如果问题是由交叠的子问题所构成的，就可以用动态规划技术。
  - 这样的子问题出现在对给定问题求解的递推关系中，这个递推关系中包含了相同类型的更小子问题的解
- 思想：与其对交叠的子问题一次又一次的求解，不如对每个较小的子问题只求解一次并把结果记录在表中。这样就可以从表中得出原始问题的解。

动态规划解决问题的思想

北京理工大学

动态规划的解决该类问题的思想：

- 对较小子问题进行一次求解，并把结果记录下来，然后利用较小问题的解，求解出较大问题的解，直到求解出最大问题的解。

➢ Fibonacci sequence: 0，1，1，2，3，5，8，13，21，…

$$F_i = i \qquad if \ \ i \leq 1$$

$$F_i = F_{i-1} + F_{i-2} \quad if \ \ i \geq 2$$

最优性原理(Principle of Optimality)

北京理工大学

最优性原理(Principle of Optimality)

- 无论过程的初始状态和初始决策是什么，其余的决策都必须相对于初始决策所产生的状态构成一个最优决策序列。
- 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的。
- 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。

动态规划算法的基本步骤

北京理工大学

- **划分阶段：**
  - 按照问题的时间或空间特征，把问题分为若干个阶段。
- **选择状态：**
  - 将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。
- **确定决策并写出状态转移方程：**
  - 状态转移就是根据上一阶段的状态和决策来导出本阶段的状态。
- **写出规划方程（包括边界条件）：**
  - 动态规划的基本方程是规划方程的通用形式化表达式。

动态规划-基本例子

北京理工大学

- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

动态规划-基本例子

北京理工大学

- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

币值最大化问题

北京理工大学

- 给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。

币值最大化问题

北京理工大学

- 给定一排n个硬币，其面值均为正整数$c_1, c_2, \cdots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。
  - 最大可选金额用$F(n)$表示。

币值最大化问题

北京理工大学

- **给定一排n个硬币，其面值均为正整数$c_1, c_2, \cdots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。**
  - 最大可选金额用$F(n)$表示。
  - 将所有可行的选择划分为两组：

币值最大化问题

北京理工大学

- **给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。**
  - 最大可选金额用$F(n)$表示。
  - 将所有可行的选择划分为两组：
    - 第1组：包含最后一枚硬币，$F(n) = c_n + F(n-2)$

币值最大化问题

北京理工大学

- **给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。**
  - 最大可选金额用$F(n)$表示。
  - 将所有可行的选择划分为两组：
    - 第1组：包含最后一枚硬币，$F(n) = c_n + F(n-2)$
    - 第2组：不包含最后一枚硬币，$F(n) = F(n-1)$

币值最大化问题

北京理工大学

- **给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。**
  - 最大可选金额用$F(n)$表示。
  - 将所有可行的选择划分为两组：
    - 第1组：包含最后一枚硬币，$F(n) = c_n + F(n-2)$
    - 第2组：不包含最后一枚硬币，$F(n) = F(n-1)$
  - 递推式：

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, n > 1$$

币值最大化问题

北京理工大学

- **给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。**
  - 最大可选金额用$F(n)$表示。
  - 将所有可行的选择划分为两组：
    - 第1组：包含最后一枚硬币，$F(n) = c_n + F(n-2)$
    - 第2组：不包含最后一枚硬币，$F(n) = F(n-1)$
  - 递推式：

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = c_1$$

币值最大化问题

北京理工大学

- 给定一排n个硬币，其面值均为正整数$c_1, c_2, \ldots, c_n$，这些整数并不一定两两相同。请问如何选择硬币，使得在原始位置互不相邻的条件下，所选硬币的总金额最大。

算法 CoinRow($C[1..n]$)

//应用公式(8.3)，自底向上求最大金额

//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币

//输入：数组 $C[1..n]$保存 n 个硬币的面值

//输出：可选硬币的最大金额

$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$

for $i \leftarrow 2$ **to** n **do**

    $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$

return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
    $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

$F[6] = \max\{2 + 15, 15\} = 17$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, n > 1$$
$$F(0) = 0, F(1) = c_1$$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0;\ F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, n > 1$$
$$F(0) = 0, F(1) = c_1$$

为了求出最大金额的那些硬币，需要回溯计算过程来确定在递推方程中是$c_n + F(n-1)$还是$F(n-1)$，产生了最大的金额。

1. 最后一次引用方程：$c_6 + F(4) = 17$，$\boldsymbol{c_6 = 2}$
2. 回溯$F(4)$，由$c_4 + F(2)$给出，$c_4 = 10$
3. F(2)最大值由F(1)产生，不需要c_2，选择$c_1 = 5$

币值最大化问题

北京理工大学

- 应用上述算法求解一排硬币5,1,2,10,6,2

$F[0] = 0,\ F[1] = c_1 = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

下标	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//在满足所选硬币不相邻的条件下，从一排硬币中选择最大金额的硬币
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, n > 1$$
$$F(0) = 0, F(1) = c_1$$

为了求出最大金额的那些硬币，需要回溯计算过程来确定在递推方程中是$c_n + F(n-1)$还是$F(n-1)$，产生了最大的金额。

1. 最后一次引用方程：$c_6 + F(4) = 17$，$\boldsymbol{c_6 = 2}$
2. 回溯$F(4)$，由$c_4 + F(2)$给出，$c_4 = 10$
3. F(2)最大值由F(1)产生，不需要c_2，选择$c_1 = 5$

为了避免回溯时的重复计算，关于递推方程两个项哪个最大的信息也保存在一个额外数组中。

动态规划-基本例子

北京理工大学

- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

找零问题

北京理工大学

- 找零问题：考虑著名找零问题的一般情形，需找零金额为n，最少用多少面值为$d_1 < d_2 < \cdots < d_m$的硬币？也就是说，假设有m种面值为$d_1 < d_2 < \cdots < d_m$的硬币，其中$d_0 = 1$，且每种面值硬币数量无限可得。

找零问题

北京理工大学

- **$F(n)$为总金额为n的数量，$F(0)=0$。获得n的途径只能是：在总金额为$n-d_j$的一堆硬币上加入一个面值为d_j的硬币，其中$j=1,2,\ldots,m$，并且$n \geq d_j$。**
  - 只需要考虑所有满足上述要求的d_j并选择使得$F(n-d_j)+1$最小的d_j即可。
  - 由于1是常量，可以先找出最小的$F(n-d_j)$，加1即可。

找零问题

北京理工大学

- 找零问题：考虑著名找零问题的一般情形，需找零金额为n，最少要多少面值为$d_1 < d_2 < \cdots < d_m$的硬币？也就是说，假设有m种面值为$d_1 < d_2 < \cdots < d_m$的硬币，其中$d_0 = 1$，且每种面值硬币数量无限可得。
- 递推方程：

$$\begin{cases} \text{当} n > 0, \quad F(n) = \min\limits_{j:n \geq d_j} \{F(n - d_j)\} + 1 \\ F(0) = 0 \end{cases}$$

找零问题

北京理工大学

- 找零问题：考虑著名找零问题的一般情形，需找零金额为n，最少要多少面值为$d_1 < d_2 < \cdots < d_m$的硬币？也就是说，假设有m种面值为$d_1 < d_2 < \cdots < d_m$的硬币，其中$d_0 = 1$，且每种面值硬币数量无限可得。

```
算法 ChangeMaking(D[1..m], n)
    //应用动态规划算法求解找零问题，找出使硬币加起来等于 n 时所需最少的硬币数目
    //其中币值为 d1 < d2 < … < dm，d1 = 1
    //输入：正整数 n，以及用于表示币值的递增整数数组 D[1...m]，D[1] = 1
    //输出：总金额等于 n 的硬币最少的数目
    F[0]←0
    for i ←1 to n do
        temp←∞; j ←1
        while j ≤ m and i ≥ D[j ] do
            temp ←min(F [i − D[j ]], temp)
            j ←j + 1
        F[i]←temp + 1
    return F[n]
```

找零问题

北京理工大学

- 假设有m种面值为$d_1 < d_2 < \cdots < d_m$的硬币，其中$d_0 = 1$。
- $n = 6$，币值为1,3,4的硬币算法过程如下：

$F[0] = 0$

n	0	1	2	3	4	5	6
F	0						

$F[1] = \min\{F[1-1]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1					

$F[2] = \min\{F[2-1]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2				

$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	2

$$\begin{cases} \text{当} n>0, \ F(n)=\min\limits_{j:n\geq d_j}\{F(n-d_j)\}+1 \\ F(0)=0 \end{cases}$$

- 假设有m种面值为

动态规划-基本例子

北京理工大学

- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

硬币收集问题

硬币收集问题:

- 在$n \times m$格木板中放有一些硬币，每格的硬币数目最多为1个。
- 在木板左上方的机器人需要收集尽可能多的硬币并把它们带到右下方的单元格。
- 每一步，机器人可以从当前的位置向右移动一格或向下移动一格。
- 当机器人遇到一格有硬币的单元格时，就会将这些硬币收集起来。
- 设计一个算法找出机器人能找到的最大硬币数并给出相应的路径。

硬币收集问题

北京理工大学

- **硬币收集问题**：令$F(i,j)$为机器人截止到第i行第j列单元格(i,j)能够收集到的最大硬币数。单元格(i,j)可以经由上方相邻单元格$(i-1,j)$或者左边相邻单元格$(i,j-1)$到达。
- 单元格$(i-1,j)$和单元格$(i,j-1)$中最大的硬币数目分别为$F(i-1,j)$和$F(i,j-1)$。
- 第一行单元格没有上方相邻单元格，第一列单元格没有左边相邻单元格，此时假定，$F(i-1,j)$或$F(i,j-1)$为0.
- 递推方程：

硬币收集问题

北京理工大学

- **硬币收集问题**：令$F(i,j)$为机器人截止到第i行第j列单元格(i,j)能够收集到的最大硬币数。单元格(i,j)可以经由上方相邻单元格$(i-1,j)$或者左边相邻单元格$(i,j-1)$到达。
- 单元格$(i-1,j)$和单元格$(i,j-1)$中最大的硬币数目分别为$F(i-1,j)$和$F(i,j-1)$。
- 第一行单元格没有上方相邻单元格，第一列单元格没有左边相邻单元格，此时假定，$F(i-1,j)$或$F(i,j-1)$为0.
- 递推方程：

$$\begin{cases} F(i,j)=\max\{F(i-1,j),F(i,j-1)\}+c_{ij}, & 1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m \\ F(0,j)=0, \quad 1\leqslant j\leqslant m; \quad F(i,0)=0, \quad 1\leqslant i\leqslant n \end{cases}$$

若单元格(i,j)中有硬币存在，则 c_{ij} 为 1，否则为 0。

硬币收集问题

北京理工大学

- **硬币收集问题**：令$F(i,j)$为机器人截止到第i行第j列单元格(i,j)能够收集到的最大硬币数。单元格(i,j)可以经由上方相邻单元格$(i-1,j)$或者左边相邻单元格$(i,j-1)$到达。

算法 RobotCoinCollection($C[1..n, 1..m]$)

//利用动态规划算法计算机器人在 $n \times m$ 木板上所能收集的最大硬币数

//机器人从(1, 1)出发，每次向右或向下移动，从左上方移动到右下方

//输入：矩阵 $C[1..n, 1..m]$，矩阵元素为 1 或者 0 分别表示单元格中有一枚硬币或者没有

//输出：机器人在单元格(n,m)中收集到的最大硬币数

$F[1, 1] \leftarrow C[1, 1]$; **for** $j \leftarrow 2$ **to** m **do** $F[1, j] \leftarrow F[1, j-1] + C[1, j]$

for $i \leftarrow 2$ **to** n **do**

  $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$

  for $j \leftarrow 2$ **to** m **do**

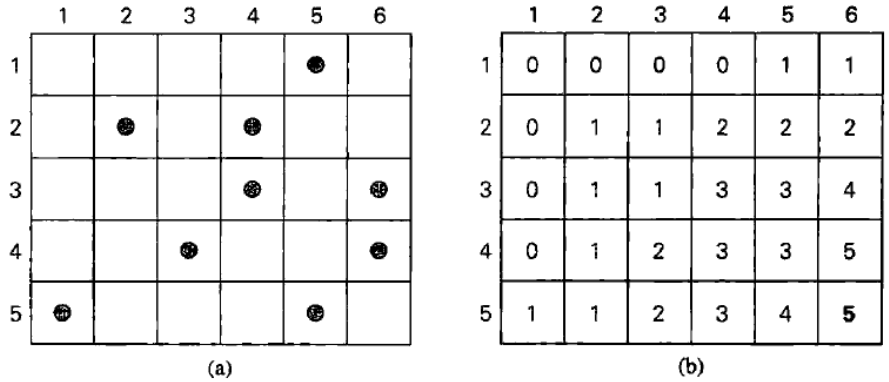
    $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$

return $F[n, m]$

硬币收集问题

北京理工大学

- **硬币收集问题**：令$F(i,j)$为机器人截止到第i行第j列单元格(i,j)能够收集到的最大硬币数。单元格(i,j)可以经由上方相邻单元格$(i-1,j)$或者左边相邻单元格$(i,j-1)$到达。



(a)木板上初始的硬币布局，(b)动态规划算法的计算结果，(c)收集到 5 个硬币(最大值)的两条路径

动态规划-基本例子

北京理工大学

- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

计算二项式系数

北京理工大学

➢ 二项式系数(Binomial Coefficient)

- 是n元素集合中k个元素组合的数量$(0 \le k \le n)$，记作$C(n,k)$或者C_n^k，来源于二项式公式：

$$(a+b)^n = C_n^0 a^n + \cdots + C_n^i a^{n-i} b^i + \cdots + C_n^n b^n$$

➢ 递推关系式：

当n>k>0时，$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$

其他，$C_n^0 = C_n^n = 1$

或者写成：C(n,k)=C(n-1,k-1)+C(n-1,k)，n>k>0

C(n,0)=C(n,n)=1

如：**C(2,1)=C(1,0)+C(1,1)=1+1=2**

C(3,1)=C(2,0)+C(2,1)=1+2=3 …

二项式系数(Binomial Coefficient)

北京理工大学

C(n,k)	0	1	2	…	k-1	k
0	1					
1	1	1				
2	1	2	1			
…						
k	1					1
…						
n-1	1				C(n-1,k-1)	C(n-1,k)
n	1					C(n,k)

C(n,k)=C(n-1,k-1)+C(n-1,k)，n>k>0

C(n,0)=C(n,n)=1

帕斯卡三角形

二项式系数算法

北京理工大学

算法 **Binomial(n,k)**

//用动态规划算法计算C(n,k)

//输入：一对非负整数n≥k ≥0

//输出：C(n,k)的值

```
for(i←0;i<=n;i++)
    for(j←0;j<=min(i,k);j++)
        if(j=0 || j=k)   C[i,j]=1;
        else
            C[i,j]←C[i-1,j-1]+C[i-1,j];
Return C[n,k];
```

$$A(n,k)=\sum_{i=1}^{k}\sum_{j=1}^{i-1}1+\sum_{i=k+1}^{n}\sum_{j=1}^{k}1=\sum_{i=1}^{k}(i-1)+\sum_{i=k+1}^{n}k$$

$$=\frac{(k-1)k}{2}+k(n-k)\in\Theta(nk)$$

P215 习题8.1-2

a. 应用动态规划算法计算C(6,3).

b. 为了计算C(n,k)，在填表时是否可以逐列填，而不是逐行填?

a.

	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10
6	1	6	15	**20**

b.可以逐列填

动态规划-基本例子

北京理工大学

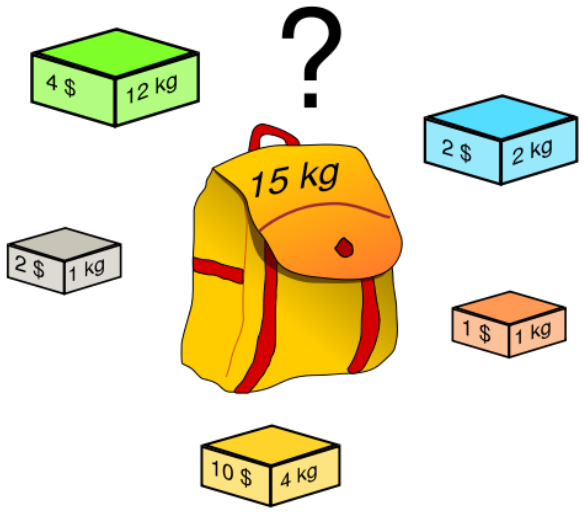
- 币值最大化问题
- 找零问题
- 硬币收集问题
- 计算二项式系数
- 背包问题

8.4 背包问题(The Knapsack Problem)

北京理工大学

重量	w_1	w_2	…	w_n
价值	v_1	v_2	…	v_n

n个物品，背包容量W



最大化： $\sum_{j=1}^{n} v_j x_j.$

限制条件：

$$\sum_{j=1}^{n} w_j x_j \le W, \quad 0 \le x_j \le 1, \quad j = 1, \ldots, n.$$

8.4 背包问题(The Knapsack Problem)

北京理工大学

➢ 动态规划分析

- 设$V[i,j]$为前i个物品放到背包容量为j的背包中时最优解的物品总价值。则目标是：$V[n,W]$。
- 对于n个物品，要得到$V[n,W]$，有两种情况：
  - a. 第n个物品不在背包中，则最优解物品总价值为$V[n-1,W]$
  - b. 第n个物品在背包中，则最优解为前$n-1$个物品的最优解总价值$V[n-1,W-\mathrm{w}_n]$与第n个物品价值v_n的和，

这里取两个中的最大值，即:

$$V[n,W]=Max\{V[n-1,W-w_n]+v_n,V[n-1,W]\}$$

背包问题(The Knapsack Problem)

北京理工大学

➢ 对于一般情况有：

- 设$V[i,j]$为前i个物品放到背包容量为j的背包中时最优解的物品总价值。有两种情况：
  - a. 第i个物品不在背包中，则最优解为$V[i-1,j]$
  - b. 第i个物品在背包中，则最优解物品总价值为前$i-1$个物品的最优解总价值$V[i-1,j-w_i]$与第i个物品价值的和，或者就是前 $i-1$ 个物品的最优解总价值，这里取两者最大值，即

$$V[i,j] = Max\{V[i-1,j-w_i]+v_i, V[i-1,j]\}$$

$$V[i,j]=\begin{cases}\max\{V[i-1,j], v_i+V[i-1,j-w_i]\} & \text{如果} j-w_i \geq 0 \\ V[i-1,j] & \text{如果} j-w_i < 0\end{cases}$$

当 $j \geq 0$ 时，$V[0,j]=0$；当 $i \geq 0$ 时，$V[i,0]=0$

物品	重量	价值
1	2	**12￥**
2	1	**10￥**
3	3	**20￥**
4	2	**15￥**

承重量W=5

$$V[i,j]=\begin{cases}\max\{V[i-1,j],v_i+V[i-1,j-w_i]\} & \text{如果} j-w_i\geq 0\\ V[i-1,j] & \text{如果} j-w_i<0\end{cases}$$

北京理工大学

	0	$j-w_i$	j	W
0	0	0	0	0
$i-1$	0	$V[i-1, j-w_i]$	$V[i-1, j]$	
w_i, v_i i	0		$V[i, j]$	
n	0			目标

价值
12￥
10￥
20￥
15￥

承重量W=5

承重量 j

i	0	1	2	3	4	5
0	0	0	0	0	0	0
$w_1=2, v_1=12$ 1	0	0	12	12	12	12
$w_2=1, v_2=10$ 2	0	10	12	22	22	22
$w_3=3, v_3=20$ 3	0	10	12	22	30	32
$w_4=2, v_4=15$ 4	0	10	15	25	30	37

时间效率： $\Theta(nW)$

V[4,5]≠V[3,5]，物品4包含在最优解中，前者由V[3,3]确定；V[3,3]＝V[2,3]，表明3不在最优解里，又因为V[2,3] ≠V[1,3]，物品2在最优解中，V[1,2] ≠V[0,2]，物品1在最优解中。最优解为＝{1,2,4}

P231 习题8.4-1 （课堂作业）

北京理工大学

a. 对于下列背包问题的实例，用自底向上的动态规划求解。

		capacity j						
	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
$w_1 = 3, v_1 = 25$	1	0	0	0	25	25	25	25
$w_2 = 2, v_2 = 20$	2	0	0	20	25	25	45	45
$w_3 = 1, v_3 = 15$	3	0	15	20	35	40	45	60
$w_4 = 4, v_4 = 40$	4	0	15	20	35	40	55	60
$w_5 = 5, v_5 = 50$	5	0	15	20	35	40	55	65

P231 习题8.4-2 （课堂作业）

北京理工大学

a. 为背包问题的自底向上的动态规划伪代码。

a. **Algorithm** *DPKnapsack*$(w[1..n], v[1..n], W)$
//Solves the knapsack problem by dynamic programming (bottom up)
//Input: Arrays $w[1..n]$ and $v[1..n]$ of weights and values of n items,
// knapsack capacity W
//Output: Table $V[0..n, 0..W]$ that contains the value of an optimal
// subset in $V[n, W]$ and from which the items of an optimal
// subset can be found
for $i \leftarrow 0$ **to** n **do** $V[i, 0] \leftarrow 0$
for $j \leftarrow 1$ **to** W **do** $V[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** W **do**
    if $j - w[i] \geq 0$
      $V[i, j] \leftarrow \max\{V[i-1, j], v[i] + V[i-1, j-w[i]]\}$
    else $V[i, j] \leftarrow V[i-1, j]$
return $V[n, W], V$

记忆功能

- **用自顶向下的方式对给定的问题求解，但需要维护一个类似自底向上动态规划算法使用的表格——目标是只对必要的子问题求解且只求解一次。**
  - 开始的时候，用“null”初始化表中的所有单元格，用来表示它们还没有被计算。
  - 一旦需要计算一个新的值，该方法首先检查表中相应的单元格，如果不是“null”就直接从中取值；否则计算它并记录在表中。

8.4.2 背包问题的记忆功能

北京理工大学

算法 *MFKnapsack* (*i*,*j*)

//对背包问题实现记忆功能方法

//输入：一个非负整数 *i* 指出先考虑的物品数量，一个非负整数 *j* 指出了背包的承重量

//输出：前 *i* 个物品的最优可行子集的价值

//注意：我们把输入数组 *Weights*[1..*n*]，*Values*[1..*n*]和表格 *V*[0..*n*,0..*W*]作为全局变量，除了行 0 和列 0 用 0 初始化以外，*V* 的所有单元都用-1 做初始化。

```
if V[i, j] < 0
    if j < Weights[i]
        value ← MFKnapsack(i − 1, j)
    else
        value ← max(MFKnapsack(i − 1), j),
                    Value[i] + MFKnapsack(i − 1, j − Weights[i]))
    V[i, j] ← value
return V[i, j]
```

		承重量 *j*					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	–	12	12
$w_2 = 1, v_2 = 10$	2	0	–	12	22	–	22
$w_3 = 3, v_3 = 20$	3	0	–	–	22	–	32
$w_4 = 2, v_4 = 15$	4	0	–	–	–	–	37