Dynamic Programming

韦俊林 (201928016029023)

2019 年 11 月 14 日

目录

1 Money Robbing

1.1 Problem Description

A robber is planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken

into on the same night.

1. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
2. What if all houses are arranged in a circle?

1.2 Optimal Substructure and DP Equation

1. Optimal Substructure:
   假设总共有 N 个房子, 令 $OPT(n,t)$，表示盗窃前 n 个房子最优收益，其中 $t=0,1$。

$t=0$: 表示不偷窃，因此 $OPT(n,0)$ 表示在不盗窃第 n 个房屋的情况下，盗窃第 1 到 n 个房屋获得的最优收益。

$t=1$: 表示盗窃，因此 $OPT(n,1)$ 表示在盗窃第 n 个房屋的情况下，盗窃第 1 到 n 个房屋获得的最有收益。

2. DP Equation:
   用 V_n 表示盗窃第 n 个房屋获得的收益，由上得：

$$OPT(n,t)=MAX\begin{cases}0 & n\leqslant 0\\ MAX(OPT(n-1,0),OPT(n-1,1)) & n\geqslant 1,t=0\\ OPT(n-1,0)+V_n & n\geqslant 1,t=1\end{cases}$$

1.3 Daily Language Description

1. 若房屋不是环形排列
   a. 开辟一个 $(N+1)*2$ 的表格（将 $[i:j]$ 记作行号为 i 列号为 j 的表项），行号为 n 表示盗窃前 n 个房屋，相对应的列表示第 n 个房屋盗窃与否（0 表示不偷窃，1 表示偷窃），表项数值表示相应情况的最大收益。其中，行号：$0,1,2,..,N$，列号：$0,1$。
   b. 将行号为 0 的表项全填 0, 即为盗窃前 0 个房屋获得的最大收益。
   c. 根据上述递推表达式，从行号为 1 至行号为 N 依次填写表格。
   d. 取行号为 N 中的较大值。
2. 若房屋是环形排列
   a. 开辟一个 $(N-1)*4$ 的表格（将 $[i:j]$ 记作行号为 i 列号为 j 的表项），表项数值表示相应情况的最大收益。其中，行号：$2,..,N$，列号：$0,1,2,3$。行号为 n 表示盗窃前 n 个房屋，相对应的列号:
      0: 在不偷窃 1 号房屋的情况下，不偷窃第 n 号房屋。
      1: 在不偷窃 1 号房屋的情况下，偷窃第 n 号房屋。
      2: 在偷窃 1 号房屋的情况下，不偷窃第 n 号房屋。
      3: 在偷窃 1 号房屋的情况下，偷窃第 n 号房屋。

b. 将 $[2:0]$,$[2:1]$,$[3:2]$,$[3:3]$ 分别填写 $0, V_2, V_1, (V_1+V_3)$。

c. 根据上述递推表达式，第 $1,2$ 列从行号为 3 至行号 N 依次填写表格，第 $3,4$ 列从行号为 4 至行号 $N-1$ 依次填写表格。

d. 取 $[n:0]$,$[n:1]$,$[(n-1):2]$,$[(n-1):3]$ 中的较大值。

1.4 pseudo-code

algorithm 1.

algorithm 1 Money Robbing

INPUT: A list of non-negative integers and length of list

OUTPUT: The maximum amount of money

```
1: function MoneyRobbing(LIST,N)
2:     OPT[0][0] ← 0
3:     OPT[0][1] ← 0
4:     for i = 1 to N do
5:         OPT[i][0] ← MAX(OPT[i − 1][0], OPT[i − 1][1])
6:         OPT[i][1] ← OPT[i − 1][0] + LIST[i]
7:     end for
8:     return MAX(OPT[N][0], OPT[N][1])
9: end function
```

1.5 Prove of Correctness

初始化： 一开始 $i=1$，

$OPT[1][0]$：前 1 个，即第 1 个房屋不盗窃，最优收益为 0。

$OPT[1][1]$：前 1 个，即第 1 个房屋盗窃，最优收益为盗窃第 1 个房屋的价值 V_1。

盗窃前一个房屋的最优收益为 V_1，所以循环不变量成立。

保持： 当 $i=k$ $(2 \leqslant k \leqslant N)$，此时已经计算出盗窃前 $k-1$ 个房屋各种情况下的最优收益的策略，则盗窃前 k 个房屋的最优收益为：

- 如果不盗窃第 k 个房屋，这种情况下的最优收益为盗窃前 $k-1$ 个房屋的最优收益，不论第 $k-1$ 个房屋是否盗窃，即伪代码第 5 行。
- 如果要盗窃第 k 个房屋，则想获得最优收益必须是在第 $k-1$ 个房屋不盗窃的情况下，盗窃前 $k-1$ 个房屋的最优收益加上盗窃第 k 个房屋的收益，即伪代码第 6 行。

因此，循环不变量成立。

终止： 房屋只有 N 个，当 $i=N+1$ 时，超出范围循环结束，循环不变量成立。

因此，由循环不变量原理，算法正确。

1.6 Analyse of Complexity

根据伪代码，显然时间复杂度为 $O(n)$。

2

2.1 Problem Description

2.2 Optimal Substructure and DP Equation

2.3 Daily Language Description

2.4 pseudo-code

2.5 Prove of Correctness

2.6 Analyse of Complexity

3 Coin Change

3.1 Problem Description

You are given coins of different denominations and a total amount of money amount. Write a function to compute the total number of ways to make up that amount using some of those coins.ionProblem Description.
Note: You may assume that you have an infinite number of each kind of coin.

3.2 Optimal Substructure and DP Equation

1. Optimal Substructure: 假设提供了 N 种类型的硬币，分别价值为 $V_1, V_2, ..., V_N$，要组合成的金钱为 SUM。则定义 $DP(i, sum)$ 为用前 i 枚硬币构成 sum 的所有组合数。

2. DP Equation:

$$DP(i, sum) = \begin{cases} 0 & i, sum \leqslant 0 \\ DP(i-1, sum) + 1 & i \geqslant 1, V_i = sum \\ DP(i-1, sum) + DP(i-1, sum - V_i) & i \geqslant 1, V_i \neq sum \end{cases}$$

3.3 Daily Language Description

a. 开辟一个 $(N+1) * (SUM+1)$ 的表格（将 $[i:j]$ 记作行号为 i 列号为 j 的表项），行号为 n 表示使用前 n 枚硬币（从小到大依次排列），相对应的列表示要组合的金钱数。

b. 假如硬币钟类或要组合的金钱总数不大于 0，则最多只有 0 种方案。

c. 假如提供了 $i(1 \leqslant i \leqslant N)$ 种硬币，则组合金钱 SUM 可以分两种情况：

   1. 不使用第 i 枚硬币，用前 $i-1$ 枚硬币组合 SUM 的所有组合数。

2. 使用第 i 枚硬币，若该枚硬币价值小于 SUM，则多出了利用 $i-1$ 枚硬币组合剩余 $SUM-V_i$ 金额的所有组合数；若该枚硬币等于 SUM，则只是多了 1 种方案。

将上述两种情况所得组合数相加，便是提供了 i 枚硬币组合 SUM 的最大组合数，即按照上述的递推式填写表格。

3.4 pseudo-code

algorithm 3.

algorithm 2 Coin Change

INPUT: COINS[N],N,SUM

OUTPUT: The total number of ways

```
1:  function CoinChange(COINS,N,SUM)
2:      for i = 0 to SUM do
3:          DP[0][i] ← 0
4:          DP[i][0] ← 0
5:      end for
6:      for i = 1 to N do
7:          for j = 1 to SUM do
8:              if j − V_i < 0 then
9:                  DP[i][j] ← DP[i − 1][j]
10:             else if j − V_i = 0 then
11:                 DP[i][j] ← DP[i − 1][j] + 1
12:             else
13:                 DP[i][j] ← DP[i − 1][j] + DP[i − 1][j − COINS[i]]
14:             end if
15:         end for
16:     end for
17:     return DP[N][SUM]
18: end function
```

3.5 Prove of Correctness

运用循环不变式原理，

初始化： 当硬币 1 枚也不提供，或要组合的面额为 0 时，组合数也为 0，即伪代码第 2, 3 行，正确。

保持： 当提供了 $i(1 \leqslant i \leqslant N)$ 枚硬币时，利用前 $i-1$ 枚硬币组合 $j(0 \leqslant j \leqslant SUM)$ 的组合数已计算完成后，只是多了利用第 i 枚硬币组合 j 的情况。因此，只需要将已有的利用前 $i-1$ 枚硬币组合 j 的组合数再加上利用第 i 枚硬币组合 j 的组合数。若第 i 枚硬币价值大于 j，此时第 k 枚硬币没法利用，伪代码第 8, 9 行。当第 i 枚硬币价值刚好等于要组合的面额时，只多了 1 种情况，伪代码第 10, 11 行；若第 i 枚硬币价值小于要组合的面额，则只多了利用前 $i-1$ 枚硬币组合剩余 $j-V_k$ 面额的组合数，伪代码 12, 13 行, 正确。

终止： 只提供 N 枚硬币，只需组合面额 SUM，当 $i > N$ 并且 $j > SUM$ 时循环跳出循环，此时已将 $DP[N][SUM]$，即利用 N 枚硬币组合 SUM 的组合数计算出来。

因此，算法正确。

3.6 Analyse of Complexity

算法主要由两个循环组成，第一个循环用于初始化代码第 2 到 5 行，时间复杂度为 $O(SUM)$; 第二个大循环是算法核心运算部分，即伪代码第 6 到 16 行，时间复杂度为 $O(N * SUM)$，显然整个算法时间复杂度为 $O(N * SUM)$。