算法导论课程报告

——01 背包问题的求解与分析

赵伯远

211440128

2023 年 12 月 4 日

目录

1 问题描述

01 背包问题是组合优化问题中的一个经典问题，其背景可以追溯到物品的装载和资源的分配。具体来说，假设有一个背包，其承载的最大重量为 W。同时，有 n 个物品，每个物品有各自的重量 w_i 和价值 v_i。01 背包问题的目标是选择一些物品装入背包中，使得这些物品的总重量不超过背包的最大承载重量，而它们的总价值尽可能大。

形式化地，我们可以将问题描述如下：

目标：

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

受到约束：

$$\sum_{i=1}^{n} w_i \cdot x_i \leq W \tag{2}$$

其中：

- x_i 是一个二元变量，如果物品 i 被选中，则 $x_i = 1$；如果没有被选中，则 $x_i = 0$。
- v_i 是物品 i 的价值。
- w_i 是物品 i 的重量。
- W 是背包的最大承重。

2 算法分析

2.1 蛮力法

2.1.1 朴素方法

以下是使用朴素方法求解 01 背包问题的伪代码：

Algorithm 1 BruteForceKnapsack

Input: $weights$ - list of item weights, $values$ - list of item values, $capacity$ - knapsack capacity

Output: Maximum value achievable

```
Function BruteForceKnapsack(weights, values, capacity):
    maxValue ← 0  foreach combination do
        if Weight(combination) ≤ capacity then
            maxValue ← max(maxValue, Value(combination))
    return maxValue
```

2.1.2 位运算优化方法

以下是使用位运算优化方法求解 01 背包问题的伪代码：

Algorithm 2 BitwiseOptimizedKnapsack

Input: $weights$ - list of item weights, $values$ - list of item values, $capacity$ - knapsack capacity

Output: Maximum value achievable

Function `BitwiseOptimizedKnapsack`($weights, values, capacity$)**:**

- $maxValue \leftarrow 0$ **for** $i \leftarrow 0$ **to** $2^n - 1$ **do**
  - $weight \leftarrow 0, value \leftarrow 0$ **for** $j \leftarrow 0$ **to** $n - 1$ **do**
    - **if** $i \ \& \ (1 \ll j)$ **then**
      - $weight \leftarrow weight + weights[j]$ $value \leftarrow value + values[j]$
  - **if** $weight \leq capacity$ **then**
    - $maxValue \leftarrow \max(maxValue, value)$
- **return** $maxValue$

2.2 动态规划

2.2.1 二维数组方法

二维数组方法涉及创建一个表格，每个条目 dp[i][w] 表示在只考虑前 i 个物品且背包容量为 w 时所能达到的最大价值。

Algorithm 3 TwoDimensionalKnapsack

Input: $weights$ - list of item weights, $values$ - list of item values, $capacity$ - knapsack capacity

Output: Maximum value achievable

Function `TwoDimensionalKnapsack`($weights, values, capacity$)**:**

- $dp \leftarrow$ array[0..length($weights$)][0..$capacity$] initialized to 0 **for** $i \leftarrow 1$ **to** *length($weights$)* **do**
  - **for** $w \leftarrow 0$ **to** $capacity$ **do**
    - **if** $weights[i] \leq w$ **then**
      - $dp[i][w] \leftarrow \max(dp[i-1][w], dp[i-1][w - weights[i]] + values[i])$
    - **else**
      - $dp[i][w] \leftarrow dp[i-1][w]$
- **return** dp[length($weights$)][capacity]

2.2.2 滚动数组方法

滚动数组方法通过仅维护表格的两行来优化空间使用，每次迭代时交替更新这两行。该方法在时间复杂度上与二维数组方法相同，但在空间复杂度上更优。

Algorithm 4 OneDimensionalKnapsack

Input: $weights$ - list of item weights, $values$ - list of item values, $capacity$ - knapsack capacity

Output: Maximum value achievable

```
22 Function OneDimensionalKnapsack(weights, values, capacity):
23     dp ← array[0..capacity] initialized to 0  for i ← 0 to length(weights) - 1 do
24         for w ← capacity to weights[i] step -1 do
25             dp[w] ← max(dp[w], dp[w − weights[i]] + values[i])
26     return dp[capacity]
```