【动态规划】01背包问题（通俗易懂，超基础讲解）

原创 Yngz_Miao 2018-08-24 22:29:29 229393 已收藏 1461 版权
分类专栏：《算法》算法导论初识 文章标签：算法 动态规划 01背包 代码 详解

问题描述

有n个物品，它们有各自的体积和价值，现有给定容量的背包，如何让背包里装入的物品具有最大的价值总和？

为方便讲解和理解，下面讲述的例子均先用具体的数字代入，即：eg：number＝4，capacity＝8

i（物品编号）	1	2	3	4
w（体积）	2	3	4	5
v（价值）	3	4	5	6

总体思路

根据动态规划解题步骤（问题抽象化、建立模型、寻找约束条件、判断是否满足最优性原理、找大问题与小问题的递推关系式、填表、寻找解组成）找出01背包问题的最优解以及解组成，然后编写代码实现。

1问题抽象化；
2建立模型；
3寻找约束条件；
4判断是否满足最优性原理；
5找大问题与小问题的递推关系式；
6填表；
7寻找解组成；

动态规划的原理

动态规划与分治法类似，都是把大问题拆分成小问题，通过寻找大问题与小问题的递推关系，解决一个个小问题，最终达到解决原问题的效果。但不同的是，分治法在子问题和子子问题等上被重复计算了很多次，而动态规划则具有记忆性，**通过填写表把所有已经解决的子问题答案纪录下来，在新问题里需要用到的子问题可以直接提取，避免了重复计算，从而节约了时间，所以在问题满足最优性原理之后，用动态规划解决问题的核心就在于填表，表填写完毕，最优解也就找到。**

最优性原理是动态规划的基础，最优性原理是指"多阶段决策过程的最优决策序列具有这样的性质：**不论初始状态和初始决策如何，对于前面决策所造成的某一状态而言，其后各阶段的决策序列必须构成最优策略**"。

背包问题的解决过程

在解决问题之前，为描述方便，首先定义一些变量：**Vi表示第 i 个物品的价值，Wi表示第 i 个物品的体积，定义V(i,j)：当前背包容量 j，前 i 个物品最佳组合对应的价值**，同时背包问题抽象化（X1，X2，…，Xn，其中 Xi 取0或1，表示第 i 个物品选或不选）。

1、建立模型，即求max(V1X1+V2X2+…+VnXn)；

2、寻找约束条件，W1X1+W2X2+…+WnXn<capacity；

3、**寻找递推关系式**，面对当前商品有两种可能性：

- **包的容量比该商品体积小，装不下，此时的价值与前i-1个的价值是一样的，即V(i,j)=V(i-1,j)；**
- **还有足够的容量可以装该商品，但装了也不一定达到当前最优价值，所以在装与不装之间选择最优的一个，即V(i,j)=**

1问题抽象化；
2建立模型；
3寻找约束条件；
4判断是否满足最优性原理；
5找大问题与小问题的递推关系式；
6填表；
7寻找解组成；

其中V(i-1,j)表示不装，V(i-1,j-w(i))+v(i) 表示装了第i个商品，背包容量减少w(i)，但价值增加了v(i)；

由此可以得出递推关系式：

- j<w(i)　　V(i,j)=V(i-1,j)
- j>=w(i)　　V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝

这里需要解释一下，为什么能装的情况下，需要这样求解（这才是本问题的关键所在！）：

可以这么理解，如果要到达V(i,j)这一个状态有几种方式?

肯定是两种，第一种是第i件商品没有装进去，第二种是第i件商品装进去了。没有装进去很好理解，就是V(i-1,j)；装进去了怎么理解呢？如果装进去第i件商品，那么装入之前是什么状态，肯定是V(i-1,j-w(i))。由于最优性原理（上文讲到），V(i-1,j-w(i))就是前面决策造成的一种状态，后面的决策就要构成最优策略。两种情况进行比较，得出最优。

4、填表，首先初始化边界条件，V(0,j)=V(i,0)=0；

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								

然后一行一行的填表：

- 如，i=1，j=1，w(1)=2，v(1)=3，有j<w(1)，故V(1,1)=V(1-1,1)=0；
- 又如i=1，j=2，w(1)=2，v(1)=3，有j=w(1),故V(1,2)=max｛V(1-1,2)，V(1-1,2-w(1))+v(1)｝=max｛0，0+3｝=3；
- 如此下去，填到最后一个，i=4，j=8，w(4)=5，v(4)=6，有j>w(4)，故V(4,8)=max｛V(4-1,8)，V(4-1,8-w(4))+v(4)｝=max｛9，4+6｝=10……

所以填完表如下图：

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7
3	0	0	3	4	5	7	8	9	9
4	0	0	3	4	5	7	8	9	10

5、表格填完，最优解即是V(number,capacity)=V(4,8)=10。

代码实现

为了和之前的动态规划图可以进行对比，尽管只有4个商品，但是我们创建的数组元素由5个。

```
#include<iostream>
```

```
using namespace std;
#include <algorithm>

int main()
{
    int w[5] = { 0 , 2 , 3 , 4 , 5 };           //商品的体积2、3、4、5
    int v[5] = { 0 , 3 , 4 , 5 , 6 };           //商品的价值3、4、5、6
    int bagV = 8;                          //背包大小
    int dp[5][9] = { { 0 } };                   //动态规划表

    for (int i = 1; i <= 4; i++) {
        for (int j = 1; j <= bagV; j++) {
            if (j < w[i])
                dp[i][j] = dp[i - 1][j];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
        }
    }

    //动态规划表的输出
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 9; j++) {
            cout << dp[i][j] << ' ';
        }
        cout << endl;
    }

    return 0;
}
```

背包问题最优解回溯

通过上面的方法可以求出背包问题的最优解，但还不知道这个最优解由哪些商品组成，故要根据最优解回溯找出解的组成，根据填表的原理可以有如下的寻解方式：

- **V(i,j)=V(i-1,j)时，说明没有选择第i 个商品，则回到V(i-1,j)；**
- **V(i,j)=V(i-1,j-w(i))+v(i)时，说明装了第i个商品，该商品是最优解组成的一部分，随后我们得回到装该商品之前，即回到V(i-1,j-w(i))；**
- **一直遍历到i = 0结束为止，所有解的组成都会找到。**

就拿上面的例子来说吧：

- 最优解为V(4,8)=10，而V(4,8)!=V(3,8)却有V(4,8)=V(3,8-w(4))+v(4)=V(3,3)+6=4+6=10，所以第4件商品被选中，并且回到V(3,8-w(4))=V(3,3)；
- 有V(3,3)=V(2,3)=4，所以第3件商品没被选择，回到V(2,3)；
- 而V(2,3)!=V(1,3)却有V(2,3)=V(1,3-w(2))+v(2)=V(1,0)+4=0+4=4，所以第2件商品被选中，并且回到V(1,3-w(2))=V(1,0)；
- 有V(1,0)=V(0,0)=0，所以第1件商品没被选择。

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7
3	0	0	3	4	5	7	8	9	9
4	0	0	3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

代码实现

背包问题最终版详细代码实现如下：

```
1  #include<iostream>
2  using namespace std;
3  #include <algorithm>
4  
5  int w[5] = { 0 , 2 , 3 , 4 , 5 };          //商品的体积2、3、4、5
6  int v[5] = { 0 , 3 , 4 , 5 , 6 };          //商品的价值3、4、5、6
7  int bagV = 8;                         //背包大小
8  int dp[5][9] = { { 0 } };                  //动态规划表
9  int item[5];                          //最优解情况
10 
11 void findMax() {                    //动态规划
12     for (int i = 1; i <= 4; i++) {
13         for (int j = 1; j <= bagV; j++) {
14             if (j < w[i])
15                 dp[i][j] = dp[i - 1][j];
16             else
17                 dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
18         }
19     }
20 }
21 
22 void findWhat(int i, int j) {              //最优解情况
23     if (i >= 0) {
24         if (dp[i][j] == dp[i - 1][j]) {
25             item[i] = 0;
26             findWhat(i - 1, j);
27         }
28 
     else if (j - w[i] >= 0 && dp[i][j] == dp[i - 1][j - w[i]] + v[i]) {
29             item[i] = 1;30 |             findWhat(i - 1, j - w[i]);
31         }
32     }
33 }
34 
35 void print() {
36     for (int i = 0; i < 5; i++) {          //动态规划表输出
37         for (int j = 0; j < 9; j++) {
38             cout << dp[i][j] << ' ';
39         }
40         cout << endl;
41     }
42     cout << endl;
43 
44     for (int i = 0; i < 5; i++)        //最优解输出
45         cout << item[i] << ' ';
46     cout << endl;
47 }
48 
49 int main()
50 {
51     findMax();
52     findWhat(4, 8);
53     print();
54 
55     return 0;
56 }
```

第30行