01背包问题 图解+详细解析 （转载）

转载 Take^that 2019-08-10 17:06:13 120169 已收藏 1456 版权
分类专栏： 数据结构 算法

一、题目描述

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

二、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。

三、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**”。

四、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)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝。**

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。

五、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;
}
```

六、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	4	5	7	8	9	10

七、代码实现

背包问题最终版详细代码实现如下：

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

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
int item[5];                                //最优解情况

void findMax() {                            //动态规划
    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]);
        }
    }
}

void findWhat(int i, int j) {               //最优解情况
    if (i >= 0) {
        if (dp[i][j] == dp[i - 1][j]) {
            item[i] = 0;
            findWhat(i - 1, j);
        }
        else if (j - w[i] >= 0 && dp[i][j] == dp[i - 1][j - w[i]] + v[i]) {
            item[i] = 1;
            findWhat(i - 1, j - w[i]);
        }
    }
}

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

    for (int i = 0; i < 5; i++)             //最优解输出
        cout << item[i] << ' ';
    cout << endl;
}

int main()
{
    findMax();
    findWhat(4, 8);
    print();

    return 0;
}
```


请发表有价值的评论，博客评论不欢迎灌水，良好的社区氛围需大家一起维护。 评论

hunterzf：这个背包问题是我见过讲的最明白的，让我彻底明白了背包问题，牛 1 年前 回复 23

weixin_46250557：博主讲的很清晰，但是背包容量j改为背包总的容量会更好，之前一直以为是剩余容量。 3 月前 回复 8

Swithun Yang：博主，你最后，直接用findWhat(4,8)吗，不应该先去找出规划表里面哪个最大然后带入吗？ 1 年前 回复 5

sigedengpao 回复：最后的就是当前状态下的最优 1 年前 回复

tommorrow12：我想请问一下，在：四、背包问题的解决过程中当i=1，j=3时最大价值我觉得应该时物品2，所以应该是4，这怎么理解？ 1 年前 回复 3

m0_61764110 回复：一件物品只能放一次，不能放两次 26 天前 回复

Take^that 博主 回复：亲，定义V(i,j)指的是：当前背包容量 j，前 i 个物品最佳组合对应的价值，已经划定范围为必须从前i个物品中取了。即： 2

```
//dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
//也就是当i=1时，我们暂时能决定的是第一件物品装包和不装包，当然是装包的价值大
dp[1][3] = max(dp[1 - 1][3], dp[1 - 1][3 - w[1]] + v[1]) = max(dp[0][3], dp[0][1] + 3) = 3;
//当i=2时，我们又发现装第二件物品的价值比装第一件物品的价值大，当然，我们选择装第二件物品
dp[2][3] = max(dp[2 - 1][3], dp[2 - 1][3 - w[2]] + v[2]) = max(dp[1][3], dp[1][0] + 4) = max(3, 4) = 4;
```

1 年前 回复

zwzp：我想问一下：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))。为什么是V(i-1,j-w(i)) 7 月前 回复 2

小楚神 回复 liujiejin：你这样理解错的啊，楼主是对的，j表示背包总容量，不是剩余容量 1 月前 回复

liujiejin 回复：楼主可能在这部分表达的不是很好，或者是楼主表达的方式我们不是很好理解（仅代表个人观点，也可能是我理解能力不好，勿喷）。如果不看文字的话单看表达式就容易理解了，V（i-1,j-w(i)）的意思就是装了第i件物品以后背包容量变成了j-w(i),而此时i-1是前i-1件物品，所以要加上第i件物品的价值，也就是v(i)。所以，装了第i件物品后，当背包容量为j-w(i)（注：此时的j是在装（或不装）第i-1件物品后的的背包容量）的时候前j件物品的最优价值就是V(i-1,j-w(i))+V(I)。不知道这样你能不能理解 6 月前 回复 3

ELVE960520：楼主，我想问一下你这个代码片段背景怎么调的 1 年前 回复 2

科楠：请问这个最终代码的时间复杂度是多少呢？😄 8 月前 回复 1

公众号：八戒程序猿：可以看看这个更好理解：https://blog.csdn.net/XuNeely/article/details/112025393 10 月前 回复 1

qq_45944293 回复：你可别吹自己了，写的不清不楚的 2 月前 回复

梦想车宾利：感谢分享 是我目前看的博客讲的最清楚的了 15 天前 回复

湍水遭岩阻：为什么装了物品之后还是i-1呢？ 29 天前 回复

< 1 2 3 >