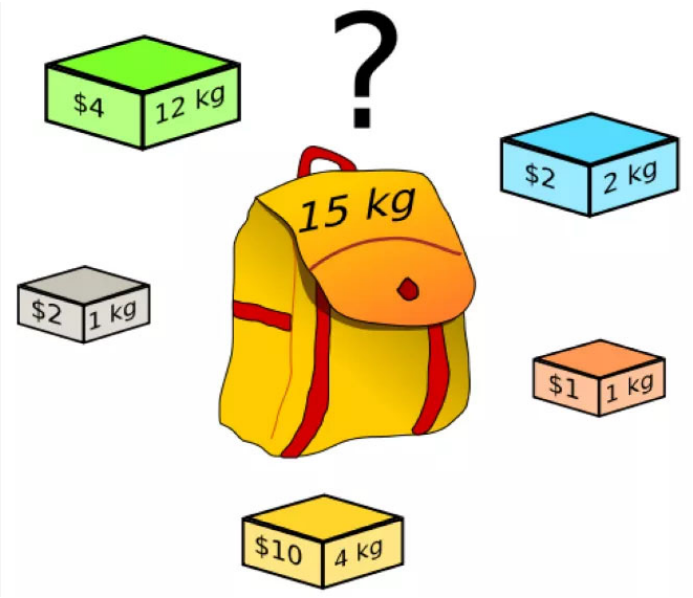
背包问题——动态规划

原创 李然 LOA算法学习笔记 2021-01-12 15:21

01 问题描述



有n个物品，它们有各自的重量和价值，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?

02 问题分析

1) 用V[i]表示物品的价值，w[i]表示物品的重量。定义状态dp[i][j]以j为容量放入前i个物品得到的最大价值。
2) 初始化边界条件，V(0,j)=V (i,0)=0。
3) 对于每个物品，有两种选择方法，装或者不装。

- 第一，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)；
- 第二，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)；

4) 得出递推关系式

$$j < w(i) \qquad V(i,j) = V(i-1,j)$$

$$j \geq w(i) \qquad V(i,j) = \max\left(V(i-1,j),\ V\left(i-1,j-w(i)\right)+v(i)\right)$$

03 DP图解

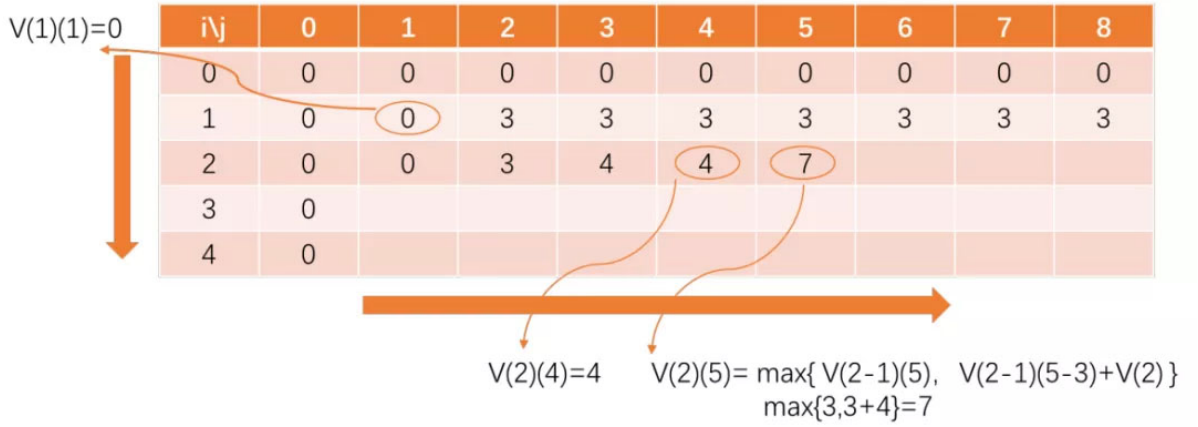
为了便于理解，设定如下参数：重量限制为8，能装4个物品。

i	1	2	3	4
W重量	2	3	4	5
V价值	3	4	5	6

1) 初始状态将边界初始化为0。j表示背包的的重量，i表示第i个物品，填表方式为一行一行的填，每次填写的时候取V(i-1,j)和V(i-1,j-w(i))+V(i)中的较大值。

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								

2) 按照一行一行，从左至右进行填表。随便取一个举例子，比如第三行第五个的7，它是由v(2-1,5)=3和v(2-1)(5-3)+v(2) =7中的较大值比较出来的。即表示为容量为5时，可以不装第二个物品即为3或者装下第二个物品即为7。



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

3) 将数组更新完则是这样的情况。即得到在容量限制为8时，能装下的最大价值为10。

04 算法设计

```cpp
#include <iostream>

using namespace std;

int w[105], val[105];
int dp[105][1005];

int main()
{
    int t, m, res=-1;
    cin >> t >> m;
    for(int i=1; i<=m; i++)
        cin >> w[i] >> val[i];

    for(int i=1; i<=m; i++) //物品
        for(int j=t; j>=0; j--) //容量
        {
            if(j >= w[i])
                dp[i][j] = max(dp[i-1][j-w[i]]+val[i], dp[i-1][j]);
            else       //只是为了好理解
                dp[i][j] = dp[i-1][j];
        }
    cout << dp[m][t] << endl;
    return 0;
}
```

05 空间优化

由上面的图解可以看出来，每一次改变的值只与V(i-1)(x)有关，V(i-1)(x)是前一次i循环保存下来的值(其中x(1，j))。因此，可以将二维数组V缩减成一维数组，从而达到优化空间的目的，状态转移方程转换为：dp(j)=max{dp(j),dp(j-w(i))+v(i)}

并且，状态转移方程，每一次推导V(i)(j)是通过V(i-1)(j-w(i))来推导的，所以一维数组中j的扫描顺序应该从大到小(最大重量到0)，否则前一次循环保存下来的值将会被修改，从而造成错误。

06 优化算法

```c
//求解将哪些物品装入背包可使这些物品的重量总和不超过背包承重量t，且价值总和最大。
#include <stdio.h>
#include <string.h>

int f[1010],w[1010],v[1010];
//f记录不同承重量背包的总价值，w记录不同物品的重量，v记录不同物品的价值

int max(int x,int y){//返回x,y的最大值
    if(x>y) return x;
    return y;
}

int main(){
    int t,m,i,j;
    memset(f,0,sizeof(f));  //总价值初始化为0
    scanf("%d %d",&t,&m);  //输入背包承重量t、物品的数目m
    for(i=1;i<=m;i++)
        scanf("%d %d",&w[i],&v[i]);  //输入m组物品的重量w[i]和价值v[i]
    for(i=1;i<=m;i++){  //尝试放置每一个物品
        for(j=t;j>=w[i];j--){//倒叙是为了保证每个物品都使用一次
            f[j]=max(f[j-w[i]]+v[i],f[j]);

        }
    }
    printf("%d",f[t]);  //输出承重量为t的背包的总价值
    printf("\n");
    getch();
    return 0;
}
```

07 作者感悟

动态规划的实质就是把大问题拆成许多个小问题，通过寻找大问题和小问题之间的递推关系，解决每个小问题，最终达到解决原问题的结果。动态规划可以描述成通过填写表把所有已经解决的子问题答案记录下来，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。