动态规划

——无17 高天鸿

- 是一类算法思想
- 问题种类多、形式灵活
- 并没有统一的套路

- 空间换时间
- 巧妙的计算流程设计
- 充分利用问题的结构

去除无效的遍历和重复计算

化为***一系列小问题***的***依次***求解

这些小问题的规模逐渐变大，每个较大的问题的解决都可以由之前解决过的小问题的解***通过简单计算***得到

最终问题增大为整个问题之后，自然就得到了整个问题的解

背包问题

背包总重量$W = 5$

序号(n)	1	2	3	4	5
重量(w_n)	2	1	3	2	1
价值(v_n)	12	10	20	15	8

- 给定N个重量为wi，价值为vi的物品和一个承重量为W的背包
- 如果选择一些物品放到背包中，求使得背包中物品价值最大的方案

背包总重量$W = 5$

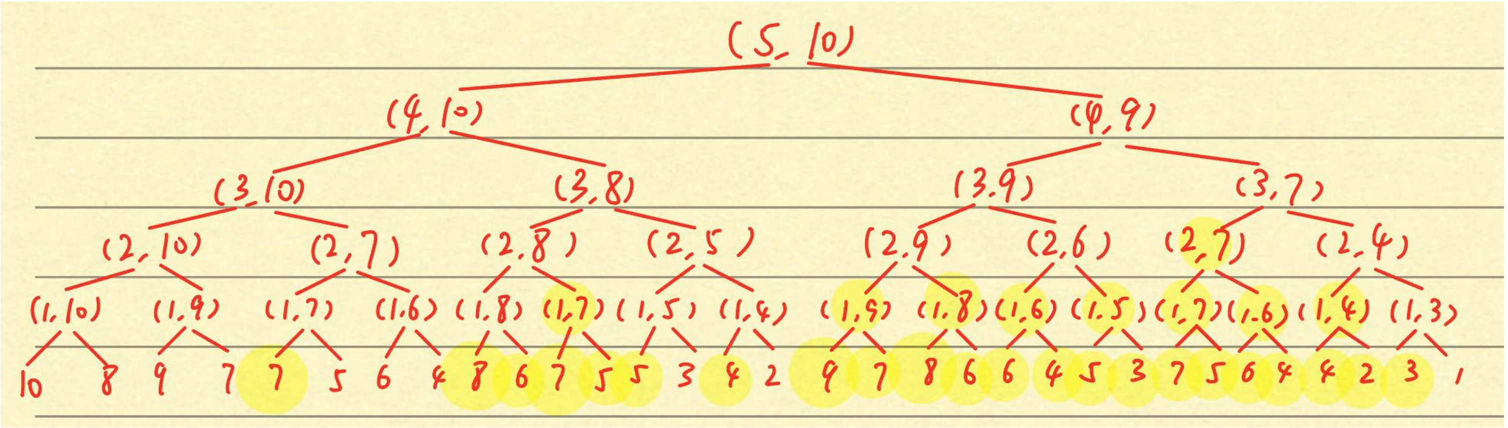
序号(n)	1	2	3	4	5
重量(w_n)	2	1	3	2	1
价值(v_n)	12	10	20	15	8

$$OPT(i, w)$$

- 递归方案：构造上面的函数递归解决；
  其中i代表只能拿1~i这些物品，w代表背包剩下的空间。

- 给出递归公式：

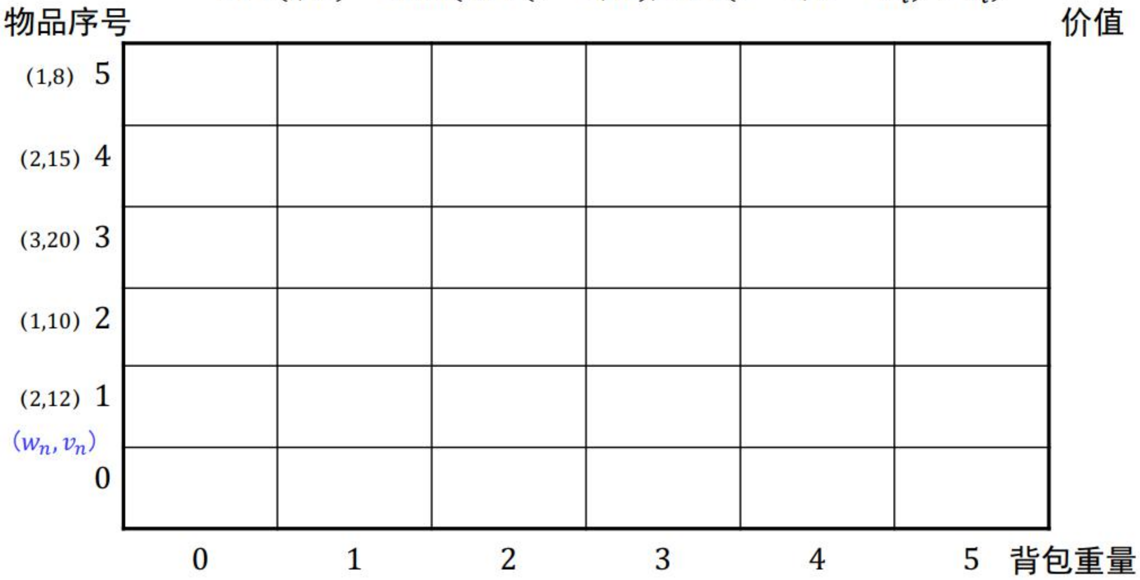
$$OPT(i, w) = \max\{OPT(i-1, w), OPT(i-1, w-w_i) + v_i\}$$


(5,10)
(4,10) (4,9)
(3,10) (3,8) (3,9) (3,7)
(2,10) (2,7) (2,8) (2,5) (2,9) (2,6) (2,7) (2,4)
(1,10) (1,9) (1,7) (1,6) (1,8) (1,7) (1,5) (1,4) (1,9) (1,8) (1,6) (1,5) (1,7) (1,6) (1,4) (1,3)
10 8 9 7 7 5 6 4 8 6 7 5 5 3 4 2 9 7 8 6 6 4 5 3 7 5 6 4 4 2 3 1

直接递归***过于耗时***。

从下往上计算并储存计算结果

$$\mathrm{OPT}(i,w)=\max(\mathrm{OPT}(i-1,w),\mathrm{OPT}(i-1,w-w_i)+v_i)$$



这时复杂度至多为n*w

- ✓ 化为***一系列小问题***的***依次***求解
- ✓ 这些小问题的规模逐渐变大，每个较大的问题的解决都可以由之前解决过的小问题的解***通过简单计算***得到
- ✓ 最终问题增大为整个问题之后，自然就得到了整个问题的解

$$\mathrm{OPT}(i,w)=\max(\mathrm{OPT}(i-1,w),\mathrm{OPT}(i-1,w-w_i)+v_i)$$

价值

(w_n, v_n)	物品序号						
(1,8)	5						
(2,15)	4						
(3,20)	3						
(1,10)	2						
(2,12)	1						
	0						
	背包重量	0	1	2	3	4	5

代码逻辑非常简单；效率也已经比较高。

当然还存在许多可以优化的空间，但是这里略去。

```
void FindMax(int capacity)
{
    /*capacity为背包容量、w数组为物品的重量、v数组为物品的价值，V为"表格"*/
    for (int i = 1; i <= numberOfObjects; i++)
    {
        for (int j = 1; j <= capacity; j++)
        {
            if (j < w[i])
            {
                V[i][j] = V[i - 1][j];
            }
            else
            {
                if (V[i - 1][j] > V[i - 1][j - w[i]] + v[i])
                {
                    V[i][j] = V[i - 1][j];
                }
                else
                {
                    V[i][j] = V[i - 1][j - w[i]] + v[i];
                }
            }
        }
    }
}
```

更多的例子

完全背包：每个物品不限个数

原来是用了第i个之后就不能再用，
现在是用了之后还可以再用！

➢ 直接修改递归公式：

背包总重量$W = 5$

序号(n)	1	2	3	4	5
重量(w_n)	2	1	3	2	1
价值(v_n)	12	10	20	15	8

$$OPT(i, w) = \max\{OPT(i-1, w), OPT(i, w-w_i) + v_i\}$$

找零钱问题

- 给定不同面额（ci）的硬币**无穷个**和一个总金额（t），求可以凑成总金额所需的最少的硬币个数（凑不成返回-1）
- 例：用c1=1,c2=5,c3=6凑t=20

找零钱问题

给定不同面额（ci）的硬币**无穷个**和一个总金额（t），求可以凑成总金额所需的最少的硬币个数（凑不成返回-1）

为叙述方便，令OPT(t)表示上面这个问题的答案

考虑从OPT(t)对应的硬币组合中去掉一个ci面额的硬币，则这个新的组合必为OPT(t-ci)对应的硬币组合！由此可以直接给出递归式：

$$OPT(t) = \min_i \{OPT(t - c_i) + 1\}$$

求连续子串和的最大值

- 对于一个整数数组，找出一个具有最大和的连续子数组（不能为空）返回其最大和

- 例：在[-2,1,-3,4,-1,2,1,-5,4]中，连续子串[4,-1,2,1]的和最大，为6

✓ **依次**求**以某一位为尾的**递增子串中最长的一个的长度：

在前面某一个比它小的数的“最大长度”上加一！

$$OPT(t) = \max\{OPT(t-1) + x[t], x[t]\}$$

最长递增子串

- 求一个序列的最长递增子序列的长度。注意子序列不一定连续！
- 例：在[10,9,2,5,3,7,23,18]中，子串[2,3,7,23]为最长递增子序列，长度为4

✓ **依次**求**以某一位为尾的**连续子串中和最大的一个的和：

要么是这一位，要么是”上一位的最大和”加上这一位！

$$OPT(t) = \max_i\{OPT(i) + 1\}, i < t \text{ 且 } x[i] < x[t]$$

全源最短路径的Floyd算法

- ✓ 依次地计算在“只允许经一部分点中转”的情况下的解，
- ✓ 并不断在这“一部分”这个集合中增加点直至全部加入

$$D_k[i][j] = \min\{D_{k-1}[i][j], D_{k-1}[i][k] + D_{k-1}[k][j]\}$$
$$(k = 0,1,\cdots,n-1)$$

```
void Floyd(Graph* graph, float* d)
{
    for (int v = 0; v < graph->Vcnt; v++)
    {
        for (int w = 0; w < graph->Vcnt; w++)
        {
            d[v * graph->Vcnt + w] = graph->adj[v * graph->Vcnt + w];
        }
    }
    for (int v = 0; v < graph->Vcnt; v++)
        d[v * graph->Vcnt + v] = 0.0;
    //开始依次计算!
    for (int i = 0; i < graph->Vcnt; i++)
    {
        for (int s = 0; s < graph->Vcnt; s++)
        {
            for (int t = 0; t < graph->Vcnt; t++)
            {
                if (d[s * graph->Vcnt + t] > d[s * graph->Vcnt + i] + d[i * graph->Vcnt + t])
                {
                    d[s * graph->Vcnt + t] = d[s * graph->Vcnt + i] + d[i * graph->Vcnt + t];
                    //按照递推关系更新
                }
            }
        }
    }
}
```

总结

巧妙地转化为***一系列***条件信息**由简单到复杂**的***同类型问题***，

然后从简单的开始***依次地计算***这些问题的解并***记录***下来，

较复杂的问题可以用较简单的问题的解非常容易地计算出来

数学抽象

- 状态
- 状态转移方程
- 无后效
- 具有最优子结构

谢谢大家！