DP 实验性讲稿 (蓝桥)

foreverlasting

目录

1 引言

DP (Dynamic Programming，动态规划) 是 CP (Competitive programming，算法竞赛) 中最核心的算法之一，基本能在各种题目中都触及到 dp。

传统的运筹学中将 dp 笼统地概括成需要三大条件：最优子结构，无后效性，子问题重叠性 (百度百科)。实际上从当代 CP 来看，本质上真正重要的只是最后一个性质。所谓**最优子结构**其实是**子问题重叠性**的自然导出结果，而无后效性在 CP 和 Math 的发展中，一些特殊问题中可以利用消元解方程的思想消除。

子问题重叠性在一定程度上完全可以被看作是 dp 的本质，只不过我们在真实运用 dp 中可能不会这样去考虑。本文将从搜索开始，逐渐理解 dp 并应用。

2 搜索与状态树

我们将从一道例题开始，这里选取的是最经典的 01 背包。

Problem 1 (01 背包)**.** *给出 n 种物品，每种物品有体积 w_i 和价值 v_i，每种物品只有一件。现你有一个容量为 V 的背包，在所装物品的体积之和不超过 V 的前提下，要求所装物品的价值之和最大。要求时间复杂度 $\mathcal{O}(nV)$。*

首先我们能数学化地抽象这个问题，即将题目改写成：

有 n 个二元组 (w_i, v_i)，要求取出一个 $[n]$ 的子集 S，使得 $\sum_{i\in S} w_i \le V$ 且 $\arg\max \sum_{i\in S} v_i$。

面对这种问题，我们有一个很经典的搜索思路，即 (这里偷了下懒，直接使用 C++ 代码了):

```cpp
std::function<void(int, int, int)> dfs = [&](int now, int nSw, int nSv) {
    if (now == n + 1) {
        ans = std::max(ans, nSv);
        return;
    }
    dfs(now + 1, nSw, nSv);
    if (nSw + w[now] <= V)dfs(now + 1, nSw + w[now], nSv + v[now]);
};
```

图 1: Dfs code

Dfs 的思路还是明确的，即对每个二元组考虑是否选择。如果不选，则递归到 $\text{dfs}(now+1, nSw, nSv)$，否则递归到 $\text{dfs}(now+1, nSw+w_{now}, nSv+v_{now})$。这样最后取 max 即可获得答案。

可以发现这样等价于枚举了所有的子集 S，时间复杂度为 $\mathcal{O}(2^n)$。

更进一步，实际上我们知道 dfs 就是对状态树的搜索，此时我们画出状态树 (例子为 $n = 4, V = 4, (w_1, v_1) = (1, 2), (w_2, v_2) = (2, 1), (w_3, v_3) = (3, 4), (w_4, v_4) = (1, 5)$)。

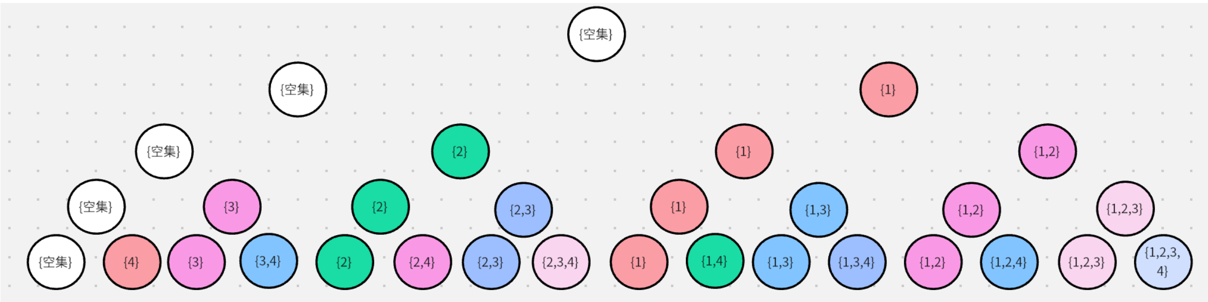

图 2: **状态树，我们要获得的最终答案就是状态树中的所有叶子。由于有限制的只有体积，我们将体积和相等的那些状态标成同一种颜色。**

3 压缩状态与 dp

从图 2 中可以发现一件重要的事情，对于同色的那些节点 (体积和相等)，在这个状态树的每一层中都只用存一个最大的价值和即可。比如第四层的 $\{3\}$ 和 $\{1,2\}$，它们这两个节点的后续转移是完全一致的，而前者的价值和为 4，后者为 3，此时无论如何后者都不太可能成为答案中的一部分。也就是，**对于 $\{1,2\}$，它在状态树上的后续部分都不可能成为答案**，因为它已经被 $\{3\}$ 的后续状态**完全以更优的方式替代**了。

这也就引申出了 dp 的思路。对于这棵状态树的每一层，对于所有体积和相等的节点，我们只用记录一个。于是就有设 $f_{i,j}$ 表示在状态树的第 i 层中，体积和为 j 的**所有节点**中价值和**最大是多少**。

注意，我们只用记录价值和是多少，不需要记录具体有哪些数。这是因为我们只想知道价值和是多少。于是就有了所谓的**转移方程**: $f_{i,j} = \max(f_{i-1,j}, f_{i-1,j-w_i} + v_i)$。从状态树上来看，这个转移方程就是对应节点的后续状态。

回头来看这个分析的过程，可以发现这是个**压缩状态**的办法。本身我们需要 $\mathcal{O}(2^n)$ 的量来记录每个状态的信息，现在由于只关注**体积和**以及最大价值和，所以只用了 $\mathcal{O}(V)$ 的量就表达了所有**有**

效状态的信息。

在平时的 dp 题，我们其实无需这样思考。我们只要知道某个状态可能表达出所要求答案的**限制**和**结果**，同时状态之间有符合**逻辑**的转移，同时能有**合理的转移顺序** (拓扑序或者消元) 即可。

还是这道题 1 举例。题目限制的是**体积和**，要求的是**最大价值和**。所以我们设 $f_{i,j}$ 表示选取了前 i 个数，**体积和**为 j 时的**最大价值和**。从逻辑上来看，我们需要获取当前物品是否选取。即若第 i 个物品没有选取，那么显然就是 $f_{i-1,j}$，即前 $i-1$ 个物品选取了体积和为 j 时的最大价值和；若选取了，则是 $f_{i-1,j-w_i}+v_i$，即前 $i-1$ 个物品选取了体积和为 $j-w_i$ 时的最大价值和，再加上选取的第 i 个物品。

可以发现这种转移是符合**逻辑**的，转移顺序也是容易得到的。注意到每次影响到 $f_{i,j}$ 的都是 i 更小的，所以我们只用 i 从小到大枚举，j 任意顺序枚举就可以得到**合理**的转移顺序。初始情况显然是 $f_{0,0}=0$，其余都不能转移，最终答案显然是 $\max_{j=0}^{V} f_{n,j}$。时间复杂度一下子优化到了 $\mathcal{O}(nV)$。

在空间上，实际上我们能观察到 f_i 只被 f_{i-1} 影响，所以能使用所谓**滚动数组**的思路，即当计算完 f_i 之后，我们就删去 f_{i-1} 的所有信息，这样空间复杂度就是 $\mathcal{O}(V)$ 了。

4 应用

参考课上的讲解，好好体会**逻辑**上正确是什么样的感觉。

实际上，在得到 dp 的状态和转移方程后，剩下来的内容与我们所谓的递推几乎没有两样。比如高中所学的斐波那契数列 $a_n=a_{n-1}+a_{n-2}$，dp 很多时候不过是更高维的递推。

彻底数学化后的好处是，我们很容易思考所谓**前缀和优化**，**数据结构优化**等方法，同时对无后效性的问题也有一些数学解法，由于本课程不会遇到，所以这里姑且不再讲解。

5 总结

dp 作为 CP 中最重要的内容，难点其实就是在于**设计状态**和**优化**。如何习得呢？最好的办法就是多做题，多思考，才可能让大脑更容易地抓住模型结构上的信息。