背包问题

李淳风

长郡中学

2024 年 9 月 20 日

前言

背包问题是线性 DP 中一类重要而特殊的模型。

01 背包

01 背包问题

有 n 个物品和一个容量为 W 的背包，每个物品有体积 V_i 和价值 W_i 两种属性，要求选若干物品放入背包，使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

01 背包

01 背包问题

有 n 个物品和一个容量为 W 的背包，每个物品有体积 V_i 和价值 W_i 两种属性，要求选若干物品放入背包，使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

状态的设计比较基础，大家应该很容易就能想到，依次考虑每个物品是否放入背包，按照“已经考虑过的物品数”来划分阶段。题目中还有背包容量的限制，我们就再增加一个维度，记录“背包里已经放入的物品总体积”。
这样我们就用 $f[i][j]$ 表示从前 i 个物品中选出了总体积为 j 的物品放入背包，物品的最大价值。

$$f[i][j] = \max \begin{cases} f[i-1][j] & \text{不选第 } i \text{ 个物品} \\ f[i-1][j-V_i] + W_i \quad if\ j \geq V_i & \text{选第 } i \text{ 个物品} \end{cases}$$

初值为 $f[0][0] = 0$，其余为负无穷。目标：$\max_{0 \leq j \leq M}\{f[N][j]\}$。

01 背包

```
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++)
        f[i][j]=f[i-1][j];
    for(int j=v[i];j<=m;j++)
        f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]);
}
```

01 背包

```
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++)
        f[i][j]=f[i-1][j];
    for(int j=v[i];j<=m;j++)
        f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]);
}
```

通过转移方程，我们可以看出，每一阶段 i 的所有状态只与上一阶段 $i-1$ 的状态有关。在这种情况下，可以使用滚动数组来降低空间。

```
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++)
        f[i&1][j]=f[(i-1)&1][j];
    for(int j=v[i];j<=m;j++)
        f[i&1][j]=max(f[i&1][j],f[(i-1)&1][j-v[i]]+w[i]);
}
```

01 背包

进一步分析代码可以发现，我们在每个阶段开始时，都执行了一次从 $f[i-1][\ ]$ 到 $f[i][\ ]$ 的拷贝操作。所以我们可以进一步，省略掉 f 数组的第一维，当外层循环到第 i 个物品时，$f[j]$ 表示背包中放入总体积为 j 的物品的最大价值。

```
for(int i=1;i<=n;i++){
    for(int j=m;j>=v[i];j--)
        f[j]=max(f[j],f[j-v[i]]+w[i]);
}
```

01 背包

进一步分析代码可以发现，我们在每个阶段开始时，都执行了一次从 $f[i-1][\ ]$ 到 $f[i][\ ]$ 的拷贝操作。所以我们可以进一步，省略掉 f 数组的第一维，当外层循环到第 i 个物品时，$f[j]$ 表示背包中放入总体积为 j 的物品的最大价值。

```
for(int i=1;i<=n;i++){
    for(int j=m;j>=v[i];j--)
        f[j]=max(f[j],f[j-v[i]]+w[i]);
}
```

需要特别注意的是，上述代码中对 j 的枚举是倒序进行的。这样一来，当我们循环到 j 时，f 的后半部分 $f[j\sim M]$ 的值已经被更新了，处于“第 i 个阶段”；前半部分 $f[0\sim j-1]$ 则处于“第 $i-1$ 个阶段”。注意我们每次用 $f[j-v[i]]$ 来更新 $f[j]$，也就是使用“第 $i-1$ 个阶段”的状态向“第 i 个阶段”的状态进行转移，从而保证了每个物品只会被放入背包一次。

而如果在这里使用正序循环，$f[j]$ 有可能会被 $f[j-V_i]+W_i$ 更新，接下来 j 增加到 $j+V_i$ 的时候，$f[j+V_i]$ 又有可能被 $f[j]+W_i$ 更新。此时，两个都处于“第 i 个阶段”的状态之间发生了转移，相当于第 i 个物品被使用了两次，继续枚举下去会导致一个物品多次使用。

完全背包

完全背包问题

有 n 种物品和一个容量为 W 的背包，第 i 种物品有体积 V_i 和价值 W_i 两种属性，每种物品都有无限个，要求选若干个物品放入背包，使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

可以先来考虑传统的二维线性 DP 的做法。设 $f[i][j]$ 表示从前 i 个物品中选出了总体积为 j 的物品放入背包，物品的最大价值。

$$f[i][j] = \max \begin{cases} f[i-1][j] & \text{不选第 } i \text{ 个物品} \\ f[i][j-V_i] + W_i \quad if\ j \geq V_i & \text{从第 } i \text{ 种物品中选一个} \end{cases}$$

初值为 $f[0][0] = 0$，其余为负无穷。目标：$\max_{0 \leq j \leq M}\{f[N][j]\}$。

完全背包

与 01 背包一样，我们也可以省略掉 f 数组的第一维，根据我们之前的分析，当采用正序循环时，就对应每种物品可以使用无限次，也对应着 $f[i,j] = f[i][j - V_i] + W_i$ 这个转移。

```
for(int i=1;i<=n;i++){
    for(int j=v[i];j<=m;j++)
        f[j]=max(f[j],f[j-v[i]]+w[i]);
}
```

例题

自然数拆分

给定一个自然数 N，要求把 N 拆分成若干个正整数相加的形式，参与加法运算的数可以重复。求拆分的方案数对 2147483648 取模的结果。$1 \leq n \leq 4000$。

例题

自然数拆分

给定一个自然数 N，要求把 N 拆分成若干个正整数相加的形式，参与加法运算的数可以重复。求拆分的方案数对 2147483648 取模的结果。$1 \leq n \leq 4000$。

这是一个典型的完全背包问题。$1 \sim N$ 这 N 个自然数构成 N 种物品，每种物品都可以无限使用，背包容积也是 N。由于本题要求方案数，所以用 $f[j]$ 表示自然数 j 的拆分方案数，把原本求 $\max$ 的函数改为求和即可。

例题

Jury Compromise

在一个遥远的国家，一名嫌疑犯是否有罪需要由陪审团来决定。陪审团是由法官从公民中挑选的。
法官先随机挑选 N 个人（编号 $1, 2, \cdots, N$）作为陪审团的候选人，然后再从这 N 个人中按照下列方法选出 M 人组成陪审团。
首先，参与诉讼的控方和辩方会给所有候选人打分，分值在 0 到 20 之间，第 i 个人的得分分别记为 p_i 和 d_i。
为了公平起见，法官选出的 M 个人必须满足：辩方总分 D 和控方总分 P 的差的绝对值 $|D-P|$ 最小。
如果选择方法不唯一，那么再从中选择辨控双方总分之和 $D+P$ 最大的方案。
求最终的陪审团获得的辩方总分 D、控方总分 P，以及陪审团人选的编号。
若陪审团的人选方案不唯一，则任意输出一组合法方案即可。

Jury Compromise

这是一道具有多个“体积维度”的 01 背包问题。把 N 个候选人看作 N 个物品，那么每个物品有如下三种“体积”：

- “人数”，每个人的“人数”都是 1，最终要填满容积为 M 的背包。
- “辩方得分”，辩方打的分数 d_i
- “控方得分”，控方打的分数 p_i

我们还是按照人数来划分阶段，当枚举完前 i 个候选人时，用 $f[j][d][p]$ 表示已经有 j 人被选入了评审团，当前辩方总分为 d，控方总分为 p 的状态是否可行。

$$f[j][d][p] = f[j][d][p] \ or \ f[i-1][d-d_i][p-p_i]$$

初始 $f[0][0][0] = True$，其余均为 $False$。
目标是找到一个状态 $f[M][D][P]$，满足 $f[M][D][P] = True$，且 $|D-P|$ 尽量小，$|D-P|$ 相同时 $D+P$ 尽量大。

Jury Compromise

但是这一做法并没有很好地利用背包“价值”这一维度。实际上，我们可以把每个人的双方的分差 $d_i - p_i$ 作为该物品的“体积”之一，把控、辩双方得分的和 $d_i + p_i$ 作为该物品的价值。
当枚举完前 i 个人时，用 $f[j][k]$ 表示已经选出了 j 个候选人，此时辩方总分与控方总分的差为 k 时，辩方总分与控方总分的和的最大值。

$$f[j][k] = \max(f[j][k], f[j-1][k-(d[i]-p[i])] + d[i] + p[i])$$

初始 $f[0][0] = 0$，其余为负无穷。目标是找到一个状态 $f[M][k]$，满足 $|k|$ 尽量小，当 $|k|$ 相同时 $f[M][k]$ 尽量大。
注意 k 这一维有可能为负数，可以给所有 k 都加上一个同样常数来解决这个问题。题目还需要输出方案，我们就新开一个数组 $g[i][j][k]$，记录当枚举完前 i 个人时，$f[j][k]$ 是从哪个状态转移过来的。当然为了方便处理，也可以把 f 开成三维数组。输出方案的时候根据 (i, j, k) 这个最终状态不断往前倒退即可。

多重背包

多重背包问题

有 n 种物品和一个容量为 W 的背包，第 i 种物品有体积 V_i 和价值 W_i 两种属性，并且有 C_i 个，要求选若干个物品放入背包，使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

多重背包

多重背包问题

有 n 种物品和一个容量为 W 的背包，第 i 种物品有体积 V_i 和价值 W_i 两种属性，并且有 C_i 个，要求选若干个物品放入背包，使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

我们当然可以把每种的 C_i 个物品都单独考虑，当作一个 01 背包问题来解决：

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=c[i];j++)
        for(int k=m;k>=v[i];k--)
            f[k]=max(f[k],f[k-v[i]]+w[i]);
}
```

但是这样没有利用到每种物品完全一样这一特点。

多重背包

众所周知，从 $2^0, 2^1, \cdots, 2^{k-1}$ 这 k 个 2 的整次幂中选出若干个数相加，可以表示出 $0 \sim 2^k - 1$ 之间的任何整数。进一步地，我们求出满足 $2^0 + 2^1 + \cdots + 2^p \leq C_i$ 的最大的整数 p，并且设 $R_i = C_i - 2^0 - 2^1 - \cdots - 2^p$。那么：

- 根据 p 的最大性，有 $2^0 + 2^1 + \cdots + 2^p + 2^{p+1} > C_i$，可以推出 $2^{p+1} > R_i$，因此从 $2^0, 2^1, \cdots, 2^p$ 中选出若干个数相加，可以表示出 $0 \sim 2^{p+1} - 1$ 之间的任何整数，包括了 $0 \sim R_i$ 之间的整数。
- 从 $2^0, 2^1, \cdots, 2^p$ 以及 R_i 中选出若干个数相加，可以表示出 $R_i \sim R_i + 2^{p+1} - 1$ 之间的任何整数。而根据 R_i 的定义，$R_i + 2^{p+1} - 1 = C_i$，因此从 $2^0, 2^1, \cdots, 2^p$ 以及 R_i 中选出若干个数相加可以表示出 $0 \sim C_i$ 之间的任何整数。

综上所述，我们可以把数量为 C_i 的第 i 种物品拆成体积为 $2^0 * V_i, 2^1 * V_i, \cdots, 2^p * V_i, R_i * V_i$ 的 $p+2$ 个物品，从这 $p+2$ 个物品中选若干个出来，可以凑出 $0 \sim C_i * V_i$ 之间所有被 V_i 整除的数。该方法仅把每种物品拆成了 $O(\log C_i)$ 个，效率较高。

例题

Coins

银国的人们使用硬币。他们有面值分别为 $A_1, A_2, \cdots, A_n$ 的硬币。有一天，托尼打开了他的储蓄罐，发现里面有一些硬币。他决定去附近的商店购买一块非常漂亮的手表。他想要支付准确的价格（不找零），而他知道手表的价格不会超过 m 。但他不知道手表的确切价格。
你需要编写一个程序，读取 n m $A_1, A_2, \cdots, A_n$ 以及对应的数量 $C_1, C_2, \cdots, C_n$（表示托尼拥有的每种面值的硬币数量），然后计算托尼可以用这些硬币支付的价格数量（从 1 到 m 的所有价格）。
$n \leq 100, m \leq 10^5, 1 \leq A_i \leq 10^5, 1 \leq C_i \leq 10^3$

例题

Coins

银国的人们使用硬币。他们有面值分别为 $A_1, A_2, \cdots, A_n$ 的硬币。有一天，托尼打开了他的储蓄罐，发现里面有一些硬币。他决定去附近的商店购买一块非常漂亮的手表。他想要支付准确的价格（不找零），而他知道手表的价格不会超过 m 。但他不知道手表的确切价格。
你需要编写一个程序，读取 n m $A_1, A_2, \cdots, A_n$ 以及对应的数量 $C_1, C_2, \cdots, C_n$（表示托尼拥有的每种面值的硬币数量），然后计算托尼可以用这些硬币支付的价格数量（从 1 到 m 的所有价格）。
$n \leq 100, m \leq 10^5, 1 \leq A_i \leq 10^5, 1 \leq C_i \leq 10^3$

我们当然可以枚举前 i 种硬币，$f[j]$ 表示现在能否拼出恰好为 j 的价格，使用多重背包的模板往上套。但对于这道题来说，哪怕使用了二进制分组，复杂度也比较勉强。当然可以使用 bitset，但是我们还可以继续优化算法。

Coins

由于本题只关注“可行性”，不关注“最优性”与“方案数”，因此对于前 i 种硬币能拼出面值 j，要么 j 能被前 $i-1$ 种硬币拼出来，要么使用了第 i 种硬币。那么我们可以考虑一种贪心策略，即尽可能少地使用第 i 种硬币，额外开一个数组 $used[j]$ 表示想要 $f[j]$ 为 $True$，至少需要使用多少枚第 i 种硬币。

```
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++) used[j]=0;
    for(int j=a[i];j<=m;j++)
        if(!f[j] && f[j-a[i]] && used[j-a[i]]<c[i])
            f[j]=True,used[j]=used[j-a[i]]+1;
}
```

虽然我们使用了完全背包模型中的枚举顺序，但通过 $used$ 数组来实现了多重背包中“物品个数”的限制。同时根据贪心策略，只有当 $f[j]$ 为 $false$，不得不利用第 i 种硬币时才会执行转移，保证不会漏掉可行解。

分组背包

分组背包问题

有 n 种物品和一个容量为 W 的背包，第 i 种物品中的第 j 个体积为 V_{ij} ，价值为 W_{ij}，要求选若干个物品放入背包，使得每组至多选择一个物品，求在总体积不超过 M 的情况下，背包中物品的最大价值。

分组背包

分组背包问题

有 n 种物品和一个容量为 W 的背包，第 i 种物品中的第 j 个体积为 V_{ij} ，价值为 W_{ij}，要求选若干个物品放入背包，使得每组至多选择一个物品，求在总体积不超过 M 的情况下，背包中物品的最大价值。

我们依然可以先考虑二维线性 DP 的做法。用 $f[i][j]$ 表示从前 i 种物品中选出了总体积为 j 的物品放入背包，物品的最大价值。

$$f[i][j] = \max \begin{cases} f[i-1][j] & \text{不选第 } i \text{ 种物品} \\ f[i-1][j-V_{ik}] + W_{ik} \quad if\ j \geq V_{ik} & \text{选第 } i \text{ 种物品中的第 } k \text{ 个} \end{cases}$$

分组背包

与前面几个背包模型一样，我们也可以省略掉 f 的第一维，用 j 的倒序枚举来控制“阶段 i”的状态只能从“阶段 $i-1$”转移过来。

```
for(int i=1;i<=n;i++)
    for(int j=m;j>=0;j--)
        for(int k=1;k<=c[i];k++)
            if(j>=v[i][k])
                f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);
```

大家还要注意到，对于每一组内 c_i 个物品的循环 k 应该放在 j 的内层。这是因为如果把 k 的循环放在 j 的循环外层，就可能会导致同一组内的物品选择多个。从动态规划的角度来看，i 是“阶段”，i, j 共同组成了“状态”，而 k 是“决策”，表示在第 i 中物品中选择哪一个放入背包，三者的顺序不能错。

此外，分组背包也是许多树上 DP 问题中状态转移的基本模型。