背包问题

dbywsc

2025/8

目录

背包问题是线性动态规划中较为特殊的一种，并且变式较多，在算法竞赛中经常出现。

P1048 [NOIP 2005 普及组] 采药
给定一个容量为 m 的背包，并且有 n 件物品，每件物品有价值 w_i 和重量 v_i 。在**每件物品只能选一次**的情况下问如何选择物品能够在容量不超过 m 的情况下价值最大。这样的问题就叫做**01 背包**。

先考虑设置二维状态：
设 $f_{i,j}$ 为选择第 i 种物品，容量为 j 时的最大价值，那么每次我们都可以从前一件物品中转移过来，所以状态转移方程为：

$$f_{i,j} = \max(f_{i-1,j}, f_{i-1,j-v_i} + w_i)$$

接下来考虑边界，显然有 $f_{0,j} = 0, j \in [0, m]$ ，最后的答案应该为 $f_{n,m}$ 。由于需要枚举每一个物品，并且对于每一个物品要枚举全部的状态，所以这种做法的时空复杂度都为 $O(nm)$ 。

```
void solve(void) {
    for(int i = 1; i <= n; i++) {
        for(int j = 0; j <= m; j++) {
            f[i][j] = f[i - 1][j];
            if(j >= v[i])
                f[i][j] = std::max(f[i][j], f[i - 1][j - v[i]] + w[i]);
        }
    }
    std::cout << f[n][m];
}
```

这样做时空复杂度皆为 $O(nm)$ 。
对于朴素做法，时间复杂度已经无法优化，但是可以优化空间复杂度。

注意到每次转移状态时，$f_{i,j}$ 只会从 $f_{i-1,j-v_i}$ / j 转移过来，也就是说每次只需要和前一个 i 进行比较就可以了，所以考虑删去第一维 i，变成 $f_j = \max(f_j, f_{j-v_i} + w_i)$ 。然而，直接删掉第一维后维持原本的遍历顺序并不与二维状态等价：

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

在上面的代码中，$max(f_j, f_{j-v_i} + w_i)$ 其实是 (i, j) 和 $(i, j - v_i)$ 进行比较，然而我们应该拿 $(i - 1, j - v_i)$ 进行比较。
要想解决也非常简单，只需要改变一下第二层的循环顺序就可以了。

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

因此，最终时间复杂度为 $O(nm)$ ，空间复杂度为 $O(m)$ 。

下面来解释一下为什么可以这样做：
由于 $j - v_i$ 一定是小于等于 j 的，如果顺序进行遍历，f_{j-v_i} 一定会在 f_j 前被更新，我们知道 $i-1$ 层是会更新到第 i 层的，所以等轮到 f_j 的时候，我们会拿第 i 层的 f_j 更新它，因此会造成错误；如果倒序遍历，我们会先拿 f_j 更新，由于 $j - v_i$ 排在它后面，所以此时还没有被更新到第 i 层，因此我们其实是在拿 $i-1$ 层的 $j - v_i$ 更新 j，就不会造成错误了。
因此最后的答案就应该为 f_m。
随着算法竞赛的发展，一维的 01 背包状态转移方程已经成为了最基本的公式。绝大部分题目都会卡掉二维的做法，并且这个公式是其他背包变形的基础。

由于本节内容较多，因此每个小节后都额外给出一些练习题：
P1049 装箱问题
P1060 开心的金明
P1734 最大约数和
P1510 精卫填海
P1466 [USACO2.2] 集合 Subset Sums

P1616 疯狂的采药

给定一个容量为 m 的背包，并且有 n 件物品，每件物品有价值 w_i 和重量 v_i 。在**每件物品可以无限取**的情况下问如何选择物品能够在容量不超过 m 的情况下价值最大。这样的问题就叫做**完全背包**。

我们同样设 $f_{i,j}$ 为选第 i 种物品，容量为 j 时的最大价值，与 01 背包的选或不选的状态不同的是，由于物品是无限的，因此我们对与第 i 件物品，可以选 $1, 2, 3, ..., k$ 件物品，显然选择的上界 k 应该是 $\lfloor \frac{m}{v_i} \rfloor$ 。
于是我们可以列出状态转移方程：

$$f_{i,j} = \max_{k=1}^{\lfloor \frac{m}{v_i} \rfloor}(f_{i-1,j}, f_{i-1,j-k\times v_i} + k \times w_i)$$

```
void solve(void) {
    for(int i = 1; i <= n; i++) {
        for(int j = 0; j <= m; j++) {
            f[i][j] = f[i - 1][j];
            for(int k = 1; k <= m / v[i]; k++) {
                if(k * v[i] <= j)
                    f[i][j] = std::max(f[i][j],
                        f[i - 1][j - k * v[i]] + k * w[i]);
            }
        }
    }
    std::cout << f[n][m];
}
```

显然，这样做的空间复杂度是 $O(nm)$ ，时间复杂度是 $O(nm^2)$ 。接下来考虑优化。

先将朴素的状态转移方程拆开：

$$f_{i,j} = \max(\{f_{i-1,j}, f_{i-1,j-v_i} + w_i, f_{i-1,j-2v_i} + 2w_i, \dots, f_{i-1,j-kv_i} + kw_i\})$$

那么 $f_{i,j-v}$ 的状态就为：

$$f_{i,j-v_i} = \max(\{f_{i-1,j-v_i}, f_{i-1,j-2v_i} + w_i, \dots, f_{i-1,j-v_i} + (k-1)w_i\})$$

可以发现，$f_{i,j}$ 其实可以表示为 $f_{i,j} = \max(f_{i-1,j}, f_{i,j-v_i} + w_i)$ 。它和 01 背包的状态转移方程及其相似。

因此我们可以先把时间复杂度优化到 $O(nm)$：

```
void solve(void) {
    for(int i = 1; i <= n; i++) {
        for(int j = 0; j <= m; j++) {
            f[i][j] = f[i - 1][j];
            if(j >= v[i]) f[i][j] = std::max(f[i][j], f[i][j - v[i]] + w[i]);
        }
    }
    std::cout << f[n][m];
}
```

现在来回忆一下 01 背包的状态转移方程：

$$f_{i,j} = \max(f_{i-1,j}, f_{i-1,j-w_i} + v_i)$$

在 01 背包问题中，由于 f_i 永远只会从 f_{i-1} 层转移过来，所以我们可以用滚动数组的方式优化空间复杂度，将第一维删去，此时为了防止前面的项被提前更新，我们采用倒着遍历的方式。而在完全背包问题中，我们发现，f_i 只会从同一层转移过来：

$$f_{i,j} = \max(f_{i-1,j}, f_{i,j-v_i} + w_i)$$

所以我们同样可以是用滚动数组。由于是从同层转移的，因此我们也不需要考虑有些项被提前更新的情况，因此从前往后遍历就可以了。

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

因此，最终时间复杂度为 $O(nm)$ ，空间复杂度为 $O(m)$ 。

本小节习题:
P2722 [USACO3.1] 总分 Score Inflation
P1679 神奇的四次方数
P1832 A+B Problem(再升级)
P1853 投资的最大效益
P2918 [USACO08NOV] Buying Hay S

给定一个容量为 m 的背包，并且有 n 种物品，每种物品有 s_i 件，并且有重量 v_i 和价值 w_i ，问如何选能在总容量不超过 m 的情况下使物品的价值之和最大，这样的问题就叫做**多重背包**。

由于存在数量的限制，所以我们可以把这个问题看作是一个有 $\sum_{i=1}^{n} s_i$ 件物品的 01 背包问题，因此可以套用朴素的 01 背包的状态转移方程，只需要稍作修改即可：

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

显然，这种做法的时间复杂度是 $O(nms)$ ，空间复杂度是 $O(nm)$ 。在一定的规模下，这种做法非常实用：

P2347 [NOIP 1996 提高组] 砝码称重

但是我们已经知道，当数据规模增大时，这种做法在时间和空间上都会超出限制，所以考虑如何优化。
P1776 宝物筛选
众所周知，$2^0, 2^1, ..., 2^n$ 可以拼凑出 1 到 $2^{n+1} - 1$ 中的任何一个数。因此我们可以反过来，把一组 s 个物品拆分成 2^0 个、2^1 个，2^2 个.... 2^k 个共 $\log s$ 组，由于我们一定能够使用这 $\log s$ 组物品凭凑出 1 到 s 中的任意一个数，因此直接对它们做一次 01 背包就可以了。顺带地，我们还可以使用滚动数组优化空间复杂度。

```
int cnt = 0;
for(int i = 1; i <= n; i++) {
    int a, b, s; std::cin >> b >> a >> s;
    int k = 1;
    while(k <= s) {
        cnt++;
        v[cnt] = a * k;
        w[cnt] = b * k;
        s -= k;
        k *= 2;
    }
    if(s > 0) {
        cnt++;
        v[cnt] = a * s;
        w[cnt] = b * s;
    }
}
n = cnt;
for(int i = 1; i <= n; i++) {
    for(int j = m; j >= v[i]; j--) {
        f[j] = std::max(f[j], f[j - v[i]] + w[i]);
    }
}
```

时间复杂度 $O(nm\log s)$ ，空间复杂度 $O(m)$ 。

本小节习题:
P6771 [USACO05MAR]Space Elevator 太空电梯

P1757 通天之分组背包

给定一个容量为 m 的背包，并且有 n 种物品，每种物品有重量 v_i 和价值 w_i ，并且每件物品属于不同的组别中。问在每组物品中只能选一个点情况下，如何选能在总容量不超过 m 的情况下使物品的价值之和最大，这样的问题就叫做**分组背包**。

设 $f_{k,j}$ 为选前 k 组物品，容量为 j 时的最大价值，此时转移分两种情况：要么不选第 k 组的物品，要么枚举第 k 组中物品，选最大值，因此状态转移方程为：

$$f_{k,j} = \max(f_{k-1,j}, \max_{i=1}^{s}(f_{k-1,j-v_i} + w_i))$$

本质上，分组背包是 01 背包的变形，所以我们同样使用滚动数组优化。

```cpp
int main(void) {
    int n, m; std::cin >> m >> n;
    std::vector<PII> s[N];
    std::vector<int> f(N, 0);
    int cnt = 0;
    for(int i = 1; i <= n; i++) {
        int a, b, c; std::cin >> a >> b >> c;
        s[c].emplace_back(a, b);
        cnt = std::max(cnt, c);
    }
    for(int i = 1; i <= cnt; i++) {
        auto last = f;
        for(auto ss : s[i]) {
            int v = ss.first, w = ss.second;
            for(int j = m; j >= v; j--) {
                f[j] = std::max(f[j], last[j - v] + w);
            }
        }
    }
    std::cout << f[m];
    return 0;
}
```

时间复杂度 $O(nm)$ ，空间复杂度 $O(n+m)$ 。

给定一个容量为 m 的背包，并且有 n 种物品。物品分为三类，第一类物品只能用一次，第二种物品可以用无限次，第三种物品可以用 s_i 次。问如何选能在总容量不超过 m 的情况下使物品的价值之和最大。这样的问题就叫做**混合背包**。

可以发现，第一类物品其实就是 01 背包问题，第二类物品就是完全背包问题，第三类物品就是多重背包问题。所以想要解决这个问题非常的简单，只需要对每一种物品进行分情况讨论，针对不同的物品使用不同的状态转移方程就可以了。

```
void solve(void) {
    int n, m; std::cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        int v, w, s;
        std::cin >> v >> w >> s;
        if(s == 0) {    //完全背包
            for(int j = v; j <= m; j++)
                f[j] = std::max(f[j], f[j - v] + w);
        } else {   //多重背包、01 背包都当作 01 背包处理。
            if(s == -1) s = 1;
            for(int k = 1; k <= s; k *= 2) {
                for(int j = m; j >= k * v; j--)
                    f[j] = std::max(f[j], f[j - k * v] + k * w);
                s -= k;
            }
            if(s) for(int j = m; j >= s * v; j--)
                    f[j] = std::max(f[j], f[j - s * v] + s * w);
        }
    }
    std::cout << f[m];
}
```

练习题：P1833 樱花

P1064 金明的预算方案

在这道题中，物品分为主件和附件，如果要买一个附件的话就必须要买它的主件。但是反过来说，买了主件不必购买全部的附件。在这种条件下，要求我们不超过背包容量的同时最大化价值。

我们可以把一个主件和他的全部附件归属于一个物品组中，此时就可以把这个问题当作分组背包处理。

```cpp
void solve(void) {
    int n, m; std::cin >> m >> n;
    for(int i = 1; i <= n; i++) {
        int v, w, q; std::cin >> v >> w >> q;
        if(!q) zhujian[i] = {v, v * w};
        else fujian[q].push_back({v, v * w});
    }
    for(int i = 1; i <= n; i++) {
        if(zhujian[i].first) {
            for(int j = m; j >= 0; j--) {
                for(int k = 0; k < (1 << fujian[i].size()); k++) {
                    int v = zhujian[i].first, w = zhujian[i].second;
                    for(int u = 0; u < fujian[i].size(); u++)
                        if(k >> u & 1) {
                            v += fujian[i][u].first;
                            w += fujian[i][u].second;
                        }
                    if(j >= v) f[j] = std::max(f[j], f[j - v] + w);
                }
            }
        }
    }
    std::cout << f[m];
}
```

相对于每小节的练习题，此处的题目更有挑战性

P2370 yyy2015c01 的 U 盘
P4141 消失之物
P1156 垃圾陷阱
P3985 不开心的金明
P1455 搭配购买
P2170 选学霸
P1858 多人背包
P5662 纪念品
P5020 [NOIP2018 提高组] 货币系统
P1941 飞扬的小鸟
P5365 [SNOI2017] 英雄联盟
P2851 [USACO06DEC]The Fewest Coins G
P5322 [BJOI2019] 排兵布阵
P1782 旅行商的背包
P2904 [USACO08MAR]River Crossing S