背包问题

$0-1$ 背包

给定 N 件物品和一个容量为 V 的背包，第 i 件物品有价值 w_i 和体积 v_i，求将物品放入背包可得到的最大价值。

朴素求解

设 $dp[i][j]$ 表示前 i 件物品放入容量为 j 的背包中的最大价值，有：

$$dp[i][j]=\max\{dp[i-1][j],dp[i-1][j-v_i]+w_i\}$$

边界条件：$dp[0][0...V]=0$.

时间复杂度：$O(VN)$，空间复杂度：$O(VN)$.

优化空间复杂度

第二维逆序遍历，保证计算前 i 个物品时调用的 dp 值是前 $i-1$ 个物品的值。

```
for(int j = 0; j <= V; j++) dp[j] = 0; // initialization
for(int i = 1; i <= n; i++)
    for(int j = V; j >= v[i]; j--)
        dp[j] = max(dp[j], dp[j-v[i]] + w[i]);
```

空间复杂度：$O(V)$.

> 注：初始化——若题目要求恰好装满，只应将 $dp[0]$ 置为 0，$dp[1...V]$ 应是 $-\mathrm{INF}$.

可行性 $0-1$ 背包：bitset 优化

可行性背包指只需要判断能否用给定的物品填满某容量为 V 的背包。

dp 是长度为最大容量的 bitset，第 i 位为 1 表示能够凑出 i.

```
dp.set(0);
for(int i = 1; i <= n; i++) dp |= dp << v[i];
```

时间复杂度：$O\left(\frac{NV}{32}\right)$，空间复杂度：$O\left(\frac{V}{32}\right)$.

完全背包

在 $0-1$ 背包的基础上补充：每种物品有无限多件。

朴素求解

设 $dp[i][j]$ 表示前 i 件物品放入容量为 j 的背包中的最大价值，有：

$$dp[i][j] = \max\{dp[i-1][j], dp[i][j-v_i] + w_i\}$$

边界条件：$dp[0][0...V] = 0$.

时间复杂度：$O(VN)$，空间复杂度：$O(VN)$.

优化空间复杂度

第二维正序遍历，允许计算前 i 个物品时调用的 dp 值还是前 i 个物品的值。

```
for(int j = 0; j <= V; j++) dp[j] = 0; // initialization
for(int i = 1; i <= n; i++)
    for(int j = v[i]; j <= V; j++)
        dp[j] = max(dp[j], dp[j-v[i]] + w[i]);
```

空间复杂度：$O(V)$.

多重背包

更改完全背包的条件为：第 i 种物品只有 k_i 件。

朴素求解

设 $dp[i][j]$ 表示前 i 件物品放入容量为 j 的背包中的最大价值，有：

$$dp[i][j] = \max_{0 \leqslant k \leqslant k_i} \{dp[i-1][j-k \cdot v_i] + k \cdot w_i\}$$

时间复杂度：$O(V\sum k_i)$.

二进制优化

每一种物品按照二进制拆分成多个物品，然后做 $0-1$ 背包即可。

时间复杂度：$O(V\sum \lg k_i)$.

```
for(int i = 1; i <= n; i++){
    for(int p = 1; p <= c[i]; c[i] -= p, p <<= 1)
        for(int j = V; j >= p * v[i]; j--)
            dp[j] = max(dp[j], dp[j-p*v[i]] + p * w[i]);
    for(int j = V; j >= c[i] * v[i]; j--)
        dp[j] = max(dp[j], dp[j-c[i]*v[i]] + c[i] * w[i]);
}
```

单调队列优化

设 $j=a\cdot v_i+b$，则转移方程变为：

$$dp[i][j]=\max_{a-k_i\leqslant k\leqslant a}\{dp[i-1][b+k\cdot v_i]-k\cdot w_i\}+a\cdot w_i$$

按 b 分类，每一类之中都用单调队列优化转移，时间复杂度 $O(VN)$.

可行性多重背包：$O(VN)$ 解法

设 $dp[i][j]$ 为用前 i 件物品恰好填满容量 j 之后，最多还剩下多少个第 i 件物品可用。$dp[i][j]$ 为 -1 表示不可行，$dp[i][j]\in[0,k_i]$ 表示可行。则：

```
for(int j = 1; j <= V; j++) dp[0][j] = -1;
dp[0][0] = 0; // initialization
for(int i = 1; i <= n; i++){
    for(int j = 0; j <= V; j++){
        if(dp[i-1][j] >= 0) dp[i][j] = k[i];
        else    dp[i][j] = -1;
    }
    for(int j = 0; j <= V - v[i]; j++)
        if(dp[i][j] > 0)
            dp[i][j+v[i]] = max(dp[i][j+v[i]], dp[i][j] - 1);
}
```

时间复杂度：$O(VN)$.

可行性多重背包：**bitset**+二进制优化

下列代码中，$a[i]$ 是体积，$c[i]$ 是对应数量，dp 是长度为最大容量的 `bitset`，第 i 位为 1 表示能凑出容量 i.

```
dp.reset(), dp.set(0); // initialization
for(int i = 1; i <= n; i++){
    for(int p = 1; p <= c[i]; c[i] -= p, p <<= 1)
        dp |= dp << (p * a[i]);
    dp |= dp << (c[i] * a[i]);
}
```

时间复杂度：$O\left(\frac{V\sum \lg k_i}{32}\right)$，空间复杂度：$O\left(\frac{V}{32}\right)$.

混合背包

混合上述三种背包的问题。是哪种背包就用哪种背包的代码即可。

二维费用背包

每件物品具有两种费用，选择该物品必须同时支付这两种费用。每一种费用都有一个最大容量，求如何选择物品得到最大价值。

求解

费用增加一维，dp 状态也增加一维即可：设 $dp[i][j][k]$ 表示前 i 件物品放入容量为 j,k 的背包的最大价值，然后按 $0-1$ 或完全或多重背包转移即可。

如若优化空间，形如 $0-1$ 只需逆序循环，形如完全只需顺序循环，形如多重则二进制拆分。

变式

若题目是在 $0-1$ 背包基础上加了限制：最多取 U 件物品，则相当于多了一维费用，本质就是二维费用背包。

分组背包

在 $0-1$ 背包的基础上，物品被划分成了 K 组，组内物品相互冲突，即最多选一件出来。求最大价值。

求解

设 $dp[k][j]$ 表示前 k 组放入容量为 j 的背包的最大价值，则：

$$dp[k][j]=\max\{dp[k-1][j],dp[k-1][j-v_i]+w_i\mid \text{item } i\in \text{group } k\}$$

优化空间复杂度

```
for(int k = 1; k <= K; k++)
    for(int j = V; j >= 0; j--)
        for(auto &i : group[k])
            dp[j] = max(dp[j], dp[j-v[i]] + w[i]);
```

有依赖的背包

每个物品都依赖于最多一个物品（即依赖关系形成一个森林），选择一个物品，则必须选择它所依赖的物品。求最大价值。

求解

树形 dp。设 $dp[i][j]$ 表示以 i 为根的子树放入容量为 j 的背包的最大价值，则在选了 i 之后，还有 $j-v_i$ 的空间去分配给子树们。对与每一棵子树而言，我们已经知道了它占 $0\sim j-v_i$ 空间的最大价值（因为自底向上转移），所以这就相当于一个分组背包——每个儿子就是一组，组内是体积分别为 $0\sim j-v_i$ 的物品。

泛化物品

泛化物品的价值是费用的一个函数：$w=h(v)$，即费用为 v 是价值为 $h(v)$.

各背包转化为泛化物品

- $0-1$ 背包中的一件物品：$h(x)=\begin{cases} w & x=v \\ 0 & \text{otherwise} \end{cases}$.
- 完全背包中的一件物品：$h(x)=[v\mid x]\frac{x}{v}w$.
- 多重背包中的一件物品：$h(x)=\left[v\mid x \text{ 且 } \frac{x}{v}\leqslant k\right]\frac{x}{v}w$.
- 分组背包中的一组：$h(x)=\max\{w_i\mid v_i=x\}$.

泛化物品的和

两个泛化物品的和也是一个泛化物品，满足：

$$h(x)=\max_{0\leqslant i\leqslant x}\{f(i)+g(x-i)\}$$

即枚举 x 的容量如何进行分配。时间复杂度 $O(V^2)$.

容易知道，泛化物品的和运算满足交换律、结合律。于是乎，任何一个背包问题，其实就是求多个泛化物品的和，答案就是和物品的最大价值。

背包问题变形

输出方案

记录每个状态是从哪一个状态转移过来的即可。

求可行方案总数

求凑出容量 V 的方案总数，不要求价值最大：将状态转移方程中的 max 改为 sum 即可。

求最优方案总数

求凑出容量 V 且价值最大的方案总数：再设一个 dp 数组，随着 dp 过程进行转移即可。

求第 K 优解

每一个状态存储的是一个有序队列，从前往后是最优解、次优解、……、K 优解。把原来的 max 改为两个有序队列的并，复杂度多一个 $O(K)$.