1、01背包问题

定义：物品只有一个，只能取或者不取。

版本1 二维

（1）状态f[i][j]定义：前i个物品，背包容量j下的最优解（最大价值）：

当前的状态依赖于之前的状态，可以理解为从初始状态f[0][0] = 0开始决策，有 N件物品，则需要 N次决 策，每一次对第 i件物品的决策，状态f[i][j]不断由之前的状态更新而来。

（2）当前背包容量不够（j < v[i]），没得选，因此前 i个物品最优解即为前 i-1个物品最优解：

对应代码：f[i][j] = f[i - 1][j]。

（3）当前背包容量够，可以选，因此需要决策选与不选第 i个物品：

选：f[i][j] = f[i - 1][j - v[i]] + w[i]。

不选：f[i][j] = f[i - 1][j] 。

我们的决策是如何取到最大价值，因此以上两种情况取 max()

```
for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
        {
            //  当前背包容量装不进第i个物品，则价值等于前i-1个物品
            if(j < v[i])
                f[i][j] = f[i - 1][j];
            // 能装，需进行决策是否选择第i个物品
            else
                f[i][j] = max(f[i - 1][j], f[i - 1][j - v[i]] + w[i]);
        }
```

版本2 一维

将状态f[i][j]优化到一维f[j]，实际上只需要做一个等价变形。

为什么可以这样变形呢？我们定义的状态f[i][j]可以求得任意合法的i与j最优解，但题目只需要求得最终状态f[n][m]，因此我们只需要一维的空间来更新状态。

（1）状态f[j]定义：N件物品，背包容量j下的最优解。

（2）注意枚举背包容量j必须从m开始。

（3）为什么一维情况下枚举背包容量需要逆序？在二维情况下，状态f[i][j]是由上一轮i - 1的

状态得来的，f[i][j]与f[i - 1][j]是独立的。而优化到一维后，如果我们还是正序，则有f[较小体积]更新到f[较大体积]，则有可能本应该用第i-1轮的状态却用的是第i轮的状态。
状态转移方程为：f[j] = max(f[j], f[j - v[i]] + w[i])。
实际上，只有当枚举的背包容量 j>= v[i] 时才会更新状态，因此我们可以修改循环终止条件进一步优化。

```
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]);
}
```

2、完全背包问题

定义：物品有无限个，可以限取得

版本一：二维
每种物品从0到最多尝试全部取一遍

```
for(int j = 0 ; j<=m ;j++)
    {
        for(int k = 0 ; k*v[i]<=j ; k++) //取k个物品
            f[i][j] = max(f[i][j],f[i-1][j-k*v[i]]+k*w[i]);
}
f[i , j ] = max( f[i-1,j]   ,  f[i-1,j-v]+w ,
   f[i-1,j-2*v]+2*w ,    f[i-1,j-3*v]+3*w , .....)

f[i , j-v]= max( f[i-1,j-v] ,  f[i-1,j-2*v] + w ,
   f[i-1,j-3*v]+2*w ,    .....)
```

由上两式，可得出如下递推关系：
f[i][j]=max(f[i,j-v]+w , f[i-1][j])
有了上面的关系，那么其实k循环可以不要了，核心代码优化成这样：

```
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]>=0)
              f[i][j]=max(f[i][j],f[i][j-v[i]]+w[i]);
        }
```

版本二：一维

用小的更新大的

```
for(int i = 1 ; i<=n ;i++)
    for(int j = v[i] ; j<=m ;j++)//注意了，这里的j是从小到大枚举，和01背包不一样
    {
            f[j] = max(f[j],f[j-v[i]]+w[i]);
        }
```

3、多重背包问题

定义：每个物品有数量限制，每一个物品可以任意选。

```
for(int i = 1;i <= n;i ++)
    {
        int a,b,s;
        cin >> a >> b >> s;
        int k = 1; // 组别里面的个数
        while(k<=s)
        {
            cnt ++ ; //组别先增加
            v[cnt] = a * k ; //整体体积
            w[cnt] = b * k; // 整体价值
            s -= k; // s要减小
            k *= 2; // 组别里的个数增加
        }
        //剩余的一组
        if(s>0)
        {
            cnt ++ ;
            v[cnt] = a*s;
            w[cnt] = b*s;
        }
    }
    n = cnt ; //枚举次数正式由个数变成组别数
    //01背包一维优化   剩下做法和01背包一样
    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]);
```