> 给你一个可装载重量为W的背包和N个物品，每个物品有重量和价值两个属性。其中第i个物品的重量为wt[i]，价值为val[i]，现在让你用这个背包装物品，最多能装的价值是多少?

动归标准套路

第一步： 明确两点，[状态]和[选择] **状态：** 如何描述一个问题的局面?

> 给定一组物品和一个背包容量限制就形成了背包问题。 **状态就有： 一组可选物品和背包容量**

选择：

> **对背包问题的选择只有两种，物品放入背包，不放入背包**

明确了状态和选择，剩下的套入入下框架：

```
for 状态1 in 状态1全集 {
  for 状态2 in 状态2全集{
    for ...
    dp[状态1][状态2][...] = 择优(选择,  不选择)
  }
}
```

第二步： 明确dp数组定义 dp数组是用来描述问题局面。即用dp数组将状态描述出来。

> dp[i][w]的定义如下：对于前i个物品，当前背包的容量为w，这种情况下可以装的最大价值是dp[i][w]。

框架演变为：

```
int dp[N+1][W+1]
dp[0][..] = 0
dp[..][0] = 0

for i in [1..N]:
    for w in [1..W]:
        dp[i][w] = max(
            把物品 i 装进背包,
            不把物品 i 装进背包
        )
return dp[N][W]
```

第三步： 根据「选择」，思考状态转移的逻辑。 当选择第i物品时，`dp[i][w] = dp[i-1][w-w[i]] + v[i-1]` 当不选择第i物品时，`dp[i][w] = dp[i-1][w]`

框架演变为：

```
int dp[N+1][W+1];
dp[...][0] = 0;
dp[0][...] = 0;

for i = 1 to w.size() {
  for j = 1 to W {
    // 放不下w[i]
    if(j - w[i-1] < 0){
      dp[i][j] = dp[i-1][j]
    }else {
      // 放得下w[i]
      dp[i][j] = std::max(dp[i-1][j], dp[i-1][j-w[i]] + v[i-1])
    }
  }
}

return dp[w.size()][W]
```

转化为C++代码为：

```
int 01Package(std::vector<int> &w, std::vector<int> &v, int weight) {
  int lenw = w.size();
  std::vector<std::vector<int>> dp(lenw+1, std::vector<int>(weight+1, 0));
  for(int i=1;i < lenw;i++) {
    for(int j = 1;j < weight; j++){
      if(j - w[i-1] < 0){
        // 无法放入
        dp[i][j] = dp[i-1][j];
      } else {
        dp[i][j] = std::max(dp[i-1][j], v[i-1]+dp[i-1][j-w[i]]);
      }
    }
  }
  return dp[lenw][weight];
}
```