0-1 背包问题

- 0-1 背包问题：给定 n 种物品和一个容量为 C 的背包，物品 i 的重量是 wi，其价值为 vi 。
  问：应该如何选择装入背包的物品，使得装入背包中的物品的总价值最大？

> 分析：面对每个物品，我们只有拿或者不拿两种选择，不能选择装入某物品的一部分，也不能装入同一物品多次。

解决方法

声明一个 大小为 dp[n][c] 的二维数组，**dp[i][j] 表示 在面对第 i 件物品，且背包容量为 j 时所能获得的最大价值，**那么我们可以很容易分析得出 dp[i][j] 的计算方法

1. j < w[i] 的情况，这时候背包容量不足以放下第 i 件物品，只能选择不拿
   dp[i][j] = dp[i-1][j]
2. j >= w[i] 的情况，此时背包可以放下第 i 件物品，结果取 拿这件物品和不拿这件物品 的最大值
   dp[i][j] = max(dp[i-1][j-w[i]] + v[i], dp[i-1][j])
   这里的dp[i-1][j-w[i]]指的是只考虑了i-1件物品（即没放入第i物品），背包容量为 j-w[i] 时的最大价值，相当于为第 i 件物品腾出了w[i]的空间

- 状态转换方程

```
if(j>=w[i])
    dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);
else
    dp[i][j]=dp[i-1][j];
```

dp[i,j] 表示只在前 i 件物品中选择若干件放在承重为 j 的背包中，可以取得的最大价值。（ dp[i][j]考虑前 i 间物品，dp[i-1][j]考虑前 i-1 件物品 ）
v i表示第i件物品的价值。
w i表示第i件物品的重量。
决策：为了背包中物品总价值最大化，第 i 件物品应该放入背包中吗？

例题

0-1背包问题。在使用动态规划算法求解0-1背包问题时，使用二维数组dp[i][j]存储背包剩余容量为j，可选物品为i、i+1、……、n时0-1背包问题的最优值。
价值数组v = {8, 10, 6, 3, 7, 2}，
重量数组w = {4, 6, 2, 2, 5, 1}，
背包容量C = 12时对应的dp[i][j]数组。

物品编号/当前承重			1	2	3	4	5	6	7	8	9	10	11	12
1	4	8	0	0	0	8	8	8	8	8	8	8	8	8
2	6	10	0	0	0	8	8	10	10	10	10	18	18	18
3	2	6												
4	2	3												
5	5	7												
6	1	2												
	重量	价值												

填表的过程中，按行填，即先考虑只有前 i 件物品。

- dp[2][5]，j=5，w[i]=w[2]=6,即** j < w[i]**，所以dp[2[5] = dp[1][5],
  即 dp[i][j]=dp[i-1][j]
- dp[2][6]时，j=6，w[i]=w[2]=6，即**j >= w[i]**，dp[2][6] = max (dp[2-1][6-w[2]] + v[2], dp[2-1][6])，比较拿与不拿第i件物品时的最大值
  即dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]);

由此可解得最优解dp[6][12]，但不清楚具体选取了哪些物品。

最大价值求解代码实现

```
int pack(const int N, const int size, vector<int> &v, vector<int>& w){  //建立dp数组，返回最优的价值总和
 int ** dp = new int*[N+1];  //申请二维数组
 for (int i = 0; i <= N; i++)
  dp[i] = new int[size + 1];

 for (int i = 0; i <= N; i++) //初始化二维数组的值
  for (int j = 0; j <= size; j++)
   dp[i][j] = 0;

 for (int i = 1; i <= N; i++){
  for (int j = 1; j <= size; j++){
   if (j < w[i])
    dp[i][j] = dp[i - 1][j];
   else
    dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
  }
 }

//求解具体选取了哪些物品，res数组保存结果（实现思路在下方）
int n = N, sizee = size;
 vector<int> res(N + 1,0);
 for (int i = N; i >= 1; --i){
```

```
  if (dp[i][size] == dp[i - 1][size])
   res[i] = 0;
  else{
   res[i] = 1;
   sizee -= w[i];
  }
 }

 int res = dp[N][size];
 delete[]dp;
 return res;
}

int main(){
 int N, size;
 cin >> N >> size;
 vector<int> v(N + 1, 0);
 for (int i = 1; i <= N; i++)
  cin >> v[i];
 vector<int> w(N + 1, 0);
 for (int i = 1; i <= N; i++)
  cin >> w[i];

 cout << "best value is: " pack(N, size, v, w) << endl;

 return 0;
}
```

求解最优值选取了哪些物品的代码实现

另起一个 x[N] 数组，x[i]=0表示不拿，x[i]=1表示拿。

- 假设m[n][c]为最优值
  如果m[n][c]=m[n-1][c] ,说明有没有第n件物品都一样，则x[n]=0；否则 x[n]=1。
    - 当x[n]=0时，由x[n-1][c]继续构造最优解；
    - 当x[n]=1时，则由x[n-1][c-w[i]]继续构造最优解。以此类推，可构造出所有的最优解。

```
//求最优解选取了哪些物品,需要基于上面求解的dp二维数组，dp[N][size]的值为最优解
 int n = N, sizee = size;
 vector<int> res(N + 1,0);
 for (int i = N; i >= 1; --i){
  if (dp[i][size] == dp[i - 1][size])
   res[i] = 0; //没拿第i件物品
  else{
   res[i] = 1; //拿了第i件物品
   sizee -= w[i]; //减去第i件物品的重量
  }
 }

```

```
cout << dp[N][size] << endl;
for (int i = 1; i <= N; i++){
 cout << res[i] << " "; //0表示没拿，1表示拿了
}
```