背包问题算法

题目描述：有n个物品，第i个物品价值为vi，重量为wi，其中vi和wi均为非负数，背包的容量为W，W为非负数。现需要考虑如何选择装入背包的物品，使装入背包的物品总价值最大。

最优解的结构特征

对于最优解，如果包含第i个物品，则最优解一定为，剩下（i - 1）个物品的最优解 + vi；如果不包含第i个物品，则最优解一定为剩下（i - 1）个物品的最优解。

定义最优解

$$c[i,w]=\begin{cases}0 & w<w_i\\ \max\{c[i-1,w-w_i]+v_i, c[i-1,w]\}\ i>0 & w_i\le w\end{cases}$$

计算最优解

```
int knapsack(vector<int> &weight, vector<int> &value, vector<vector<int>> &dp, int capacity,
int num){
    for (int  i = 1; i < num + 1; i++){
        for (int j = 1; j < capacity + 1; j++){
            if(weight[i] <= j){
                dp[i][j] = max(dp[i - 1][j] , dp[i - 1][j - weight[i]] + value[i]);
            }
            else{
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
    return dp[num][capacity];
}
```

构造最优解

得到了价值矩阵之后，需要回溯得到最优解。如果在某个容量下，如果选择了某个物品i，则总价值会和上一个总价值不同，接下去在容量为capacity - wi下，再继续找使总价值发生改变的物品，直到capcity = 0或者所有总价值为0则停止搜索。

```
vector<int> find_solution(vector<int> &weight, vector<int> &value, vector<vector<int>> &dp, int
capacity, int num){
    vector<int>  inbag;
```

```
    int flag;
    while (capacity != 0 | capacity < *min_element(weight.begin(), weight.end())) {
        flag = 0;
        for (int i = num; i > 1; i--) {
            if (dp[i][capacity] != dp[i - 1][capacity]) {
                capacity = capacity - weight[i];
                inbag.push_back(i);
                flag = 1;
            }
        }
        if (flag == 0) {
            capacity = capacity - 1; // if not found element add in this capacity, search
capacity - 1
        }
    }
    return inbag;
}
```