# 背包问题（The Knapsack Problem）简介

- 背包问题的应用领域
  - 信息密码学
  - 数论研究
  - 工业优化设计
- 背包问题的分类
  - 分数背包
  - **0-1背包**
  - 二维背包
  - 多重背包
  - 完全背包

- 基于背包公钥密码的数字图像加密


- 材料切割

# 3.4 0-1背包问题

- 给定n种物品和一背包。物品i的重量是$w_i$，其价值为$v_i$，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
- 应注意的问题：

  - 在选择装入背包的物品时，对每种物品i只有两种选择：装入和不装入{0,1}
  - 不能将物品i装入背包多次，也不能只装入部分的物品

■问题的形式化描述：

■C>0 ， $w_i$>0, $v_i$>0, 1<= i <=n, 要求找出一个n元0-1向量（$x_1,x_2,x_3,\ldots,x_n$）,xi∈{0，1}

■约束条件：
$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

■目标函数：
$$\max \sum_{i=1}^{n} v_i x_i$$

# 复习：动态规划基本步骤

- 找出最优解的性质，并刻划其结构特征。
- 递归地定义最优值。
- 以自底向上的方式计算出最优值。

基本步骤

- 根据计算最优值时得到的信息，构造最优解。

需要保存
更多信息

# 1、是否有最优子结构性质？

- 设（$y_1,y_2,\ldots,y_n$）是0-1背包问题的一个**最优解**，则（$y_2,y_3,\ldots,y_n$）是其相应一个子问题的**最优解**。

$$\max \sum_{i=2}^{n} v_i x_i, \quad \begin{cases} \sum_{i=2}^{n} w_i x_i \le C - w_1 y_1 \\ x_i \in \{0,1\}, 2 \le i \le n \end{cases}$$

- **反证法证明：**

假设（$z_2,z_3,\ldots,z_n$）是上述子问题的一个最优解，而（$y_2,y_3,\ldots,y_n$）不是其最优解。

$$\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i, \qquad w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$$

（$y_1,z_2,z_3,\ldots,z_n$）是原问题的一个解。

# 1、是否有最优子结构性质？

（$y_1,z_2,z_3,\ldots,z_n$）是原问题的一个解。

- 由：

$$\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i, \qquad w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$$

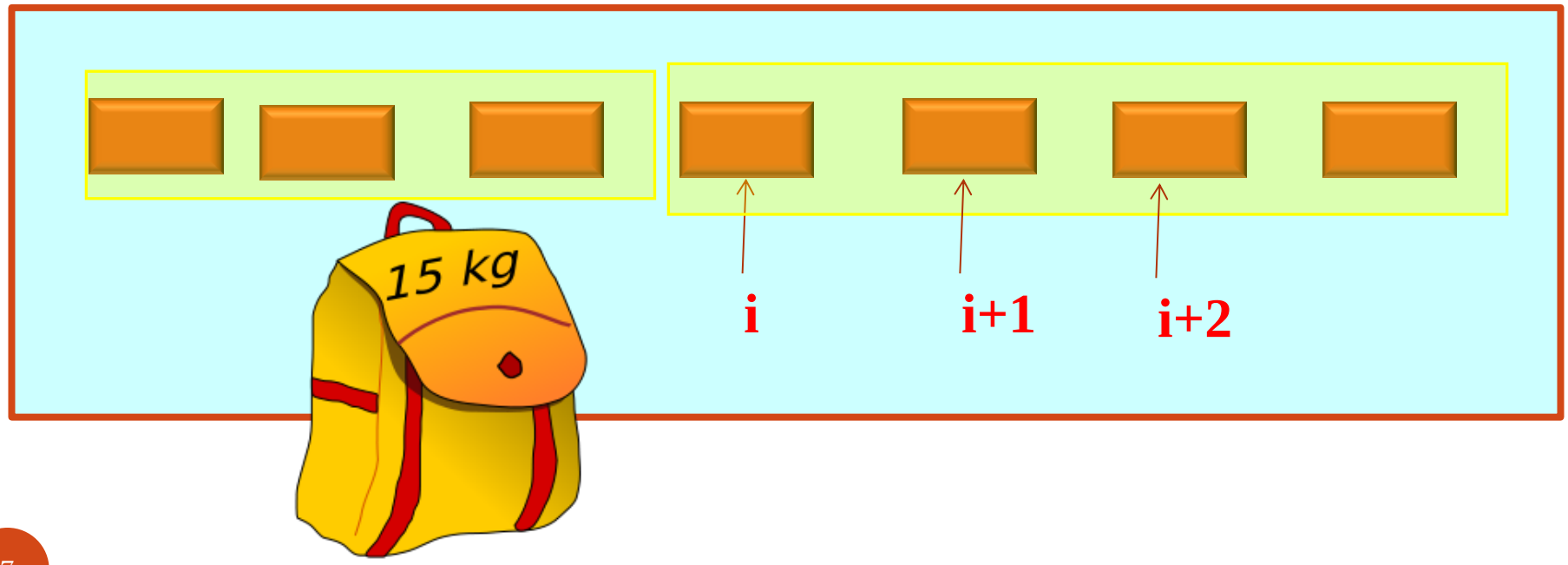
- 推得：

$$v_1 y_1 + \sum_{i=2}^{n} v_i z_i > \sum_{i=1}^{n} v_i y_i \qquad w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$$

- 这说明（$y_1,z_2,z_3,\ldots,z_n$）是更优的解，这与（$y_1,y_2,\ldots,y_n$）是最优解的假设矛盾。
- 因此， 0-1背包问题有最优子结构性质。

# 2、刻画最优值的递归关系

- **设所给0-1背包问题的子问题：**
  - 已选择完是否放入i-1个物品后，从第i个物品开始，对剩余的n-i+1个物品在背包**剩余容量**为j进行选择，使得装入背包中的物品价值**和最大**。



# 2、刻画最优值的

- 设所给0-1背包问题的子问题
  - 已选择完是否放入i-1个物
    对剩余的n-i+1个物品在背
    使得装入背包中的物品价

$$\max \sum_{k=i}^{n} v_k x_k \qquad \begin{cases} \sum_{k=i} v_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

- 思考题：
  - 考虑这两个参数与矩阵链相乘问题及最长公共子序列问题里的参数有什么区别？

- **子问题的界定：** 由i和j界定
  - i：考虑对物品i，i+1,i+2，……，n的选择
  - j：背包的容量为j。

# 2、刻画最优值的递归关系

- 最优值为m(i，j)
  - 即m(i，j)是背包容量为j，可选择物品为i，i+1，…，n时0-1背包问题的最优值，即最大价值和。
- 原问题的最优值为m（1，C）

$$m(i,j)=\begin{cases}\end{cases}$$

- 对物品的选择：i，i+1,i+2，……，n

# 2、刻画最优值的递归关系

- 由0-1背包问题的最优子结构性质，可以建立计算m(i，j)的递归式如下。

$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

$$m(n,j)=\begin{cases}$$

3、

w[]：记录

c：

n：物

m[][]：最优值

```
Void knapsack(int *v, int *w,int c,int n,int **m)
{ int jMax=min(w[n]-1,c);
  for(j=0;j<=jMax; j++)  m[n][j]=0;
  for(j=w[n]; j<=c; j++)  m[n][j]=v[n];
  for(i=n-1;i>1;i--){
   jMax=min(w[i]-1,c);
   for(j=0;j<= jMax;j++)  m[i][j]=m[i+1][j];
   for(j= w[i];j<=c;j++)  m[i][j]=max(m[i+1][j], m[i+1][j- w[i]]+v[i]);
   }
  m[1][c]=m[2][c];
  if(c>=w[1]) m[1][c]=max(m[1][c], m[2][c- w[1]]+v[1]);
}
```

第n个物品无法装入

装入第n个物品

无法装入第i个物品

可以装入第i个物品

可以装入第1个物品

## 3、计算最优值算法描述

```
Void knapsack(v,w,c,n,m)
{ int jMax=min(w[n]-1,c);
  for(j=0;j<=jMax; j++)  m[n][j]=0;
  for(j=w[n]; j<=c; j++)  m[n][j]=v[n];
  for(i=n-1;i>1;i--){
    jMax=min(w[i]-1,c);
    for(j=0;j<= jMax;j++)  m[i][j]=m[i+1][j];
    for(j= w[i];j<=c;j++)  m[i][j]=max(m[i+1][j], m[i+1][j- w[i]]+v[i]);
    }
  m[1][c]=m[2][c];
  if(c>=w[1]) m[1][c]=max(m[1][c], m[2][c- w[1]]+v[1]);
}
```

- 课堂练习1：
- 已知：
  - n=5, c=10,
  - w={2, 2, 6, 5, 4},
  - V={6, 3, 5, 4, 6}
  - 求m。

# 0-1背包问题：构造最优解

- 已知：m[n][c]如下图所示，求出该问题的最优解。

| i \ j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | 15 | √ |
| 2 | 0 | 0 | 3 | 3 | 6 | 6 | 9 | 9 | 9 | 10 | 11 | √ |
| 3 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 11 | × |
| 4 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 10 | 10 | × |
| 5 | 0 | 0 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | √ |

- 若第1个物品重量为2
- 若第2个物品重量为2

- （1,1,0,0,1）

## 4、构造最优解

```
Traceback(m[][],w[],c,n,x[])
{
   for(i=1;i<n;i++)
     if(m[i][c]==m[i+1][c])   x[i]=0;
        else { x[i]=1;c-=w[i];  }
   x[n]=(m[n][c])?1:0;
}
```

**■算法复杂度分析：**

- 从m(i，j)的递归式容易看出，算法需要O(nc)计算时间。
- 当背包容量c很大时，算法需要的计算时间较多。
  - 例如，当$c>2^n$时，算法需要$\Omega(n2^n)$计算时间。

## 3、计算最优值算法描述

```
Void knapsack(v,w,c,n,m)
{ int jMax=min(w[n]-1,c);
  for(j=0;j<=jMax; j++)  m[n][j]=0;
  for(j=w[n]; j<=c; j++)  m[n][j]=v[n];
  for(i=n-1;i>1;i--){
    jMax=min(w[i]-1,c);
    for(j=0;j<= jMax;j++)  m[i][j]=m[i+1][j];
    for(j= w[i];j<=c;j++)  m[i][j]=max(m[i+1][j], m[i+1][j- w[i]]+v[i]);
    }
   m[1][c]=m[2][c];
   if(c>=w[1]) m[1][c]=max(m[1][c], m[2][c- w[1]]+v[1]);
}
```

- 课堂练习1：
- 已知：
  - n=5, c=10,
  - w={2, 2, 6, 5, 4},
  - V={6, 3, 5, 4, 6}
  - 求m和最优解。

# 0-1背包：课堂练习2

- 物品数量：　　n=4
- 物品体积集合：　　S={2, 3, 4, 5}
- 物品价值集合：　　V={3, 4, 5, 7}
- 背包体积：　　C=9
- 求m和该问题的最优解。

# 下节课问题：投资问题

- 设有投资公司，在3月份预计总投资额为m，共有n个项目，$G_i(x)$为向第i项工程投资费用为x时的预计收益，如何分配资源才能获得最大利润？

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $G_1(x)$ | 0 | 5 | 15 | 40 | 80 | 90 | 95 | 98 | 100 |
| $G_2(x)$ | 0 | 5 | 15 | 40 | 60 | 70 | 73 | 74 | 75 |
| $G_3(x)$ | 0 | 4 | 26 | 40 | 45 | 50 | 51 | 53 | 53 |

# 下节课问题：旅行商问题

- 旅行商问题又称货郎担问题，是指某售货员要到n个城市去推销商品，已知各城市之间的路程（或旅费）。

- 售货员要选定一条从驻地出发经过每个城市一次，最后回到驻地的路线，使总的路程（总旅费）最短（最小）。