最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再做出最好选择。由此就导出许多互相重叠的子问题。这正是该问题可用动态规划算法求解的另一重要特征。实际上也是如此，动态规划算法的确可以有效地解0-1背包问题。

4.3 最优装载

有一批集装箱要装上一艘载重量为 c 的轮船。其中集装箱 i 的重量为 w_i。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

该问题可形式化描述为

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} w_i x_i \leqslant c$$

$$x_i \in \{0,1\}, \quad 1 \leqslant i \leqslant n$$

其中，变量 $x_i=0$ 表示不装入集装箱 i，$x_i=1$ 表示装入集装箱 i。

1. 算法描述

最优装载问题可用贪心算法求解。采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解。具体算法描述如下：

```
public static float loading(float c, float [] w, int [] x)
{
    int n=w.length;
    Element [] d=new Element [n];
    for (int i=0; i<n; i++)
      d[i]=new Element(w[i],i);
    MergeSort.mergeSort(d);
    float opt=0;
    for (int i=0; i<n; i++) x[i]=0;
    for (int i=0; i<n && d[i].w <=c; i++)
    {
       x[d[i].i]=1;
       opt+=d[i].w;
       c-=d[i].w;
    }
    return opt;
}
```

其中，Element 类说明如下：

```
public static class Element implements Comparable
{
    float w;
    int   i;
```

```
    public Element(float ww, int ii)
    {
        w=ww;
        i=ii;
    }

    public int compareTo(Object x)
    {
        float xw=((Element) x).w;
        if (w<xw) return-1;
        if (w==xw) return 0;
        return 1;
    }
}
```

2. 贪心选择性质

设集装箱已依其重量从小到大排序，$(x_1,x_2,\cdots,x_n)$是最优装载问题的一个最优解。又设 $k=\min\limits_{1\leqslant i\leqslant n}\{i\mid x_i=1\}$。易知，如果给定的最优装载问题有解，则 $1\leqslant k\leqslant n$。

(1) 当 $k=1$ 时，$(x_1,x_2,\cdots,x_n)$是一个满足贪心选择性质的最优解。

(2) 当 $k>1$ 时，取 $y_1=1,y_k=0,y_i=x_i,1<i\leqslant n,i\neq k$，则

$$\sum_{i=1}^{n}w_iy_i=w_1-w_k+\sum_{i=1}^{n}w_ix_i\leqslant\sum_{i=1}^{n}w_ix_i\leqslant c$$

因此，$(y_1,y_2,\cdots,y_n)$是所给最优装载问题的可行解。

另一方面，由 $\sum\limits_{i=1}^{n}y_i=\sum\limits_{i=1}^{n}x_i$ 知，$(y_1,y_2,\cdots,y_n)$ 是满足贪心选择性质的最优解。

所以，最优装载问题具有贪心选择性质。

3. 最优子结构性质

设$(x_1,x_2,\cdots,x_n)$是最优装载问题的满足贪心选择性质的最优解，则容易知道，$x_1=1$，且$(x_2,\cdots,x_n)$是轮船载重量为 $c-w_1$，待装船集装箱为$\{2,3,\cdots,n\}$时相应最优装载问题的最优解。也就是说，最优装载问题具有最优子结构性质。

由最优装载问题的贪心选择性质和最优子结构性质，容易证明算法 loading 的正确性。

算法 loading 的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为 $O(n\log n)$。

4.4 哈夫曼编码

哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在20%～90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用0,1串表示各字符的最优表示方式。假设有一个数据文件包含100 000个字符，要用压缩的方式存储它。该文件中各字符出现的频率如表4-1所示。文件中共有6个不同字符出现。字符a出现45 000次，字符b出现13 000次等。