《算法分析与设计》

课程设计报告

学院（系）：软件工程系
班级：　116030801
学生姓名：　周翔辉
学号：　11603080122
指导老师：　刘祥

时间：从 2018 年 12 月 17 日到 2018 年 12 月 27 日

目录

1 基本的递归算法

1.1 二项式的计算

1.1.1 问题描述

完成二项式公式计算, 即 $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$ 公式解释为了从 n 个不同元素中抓取 k 个元素 (C_n^k), 可以这样考虑, 如果第一个元素一定在结果中, 那么就需要从剩下的 $n-1$ 个元素中抓取 $k-1$ 个元素 (C_{n-1}^{k-1}); 如果第一个元素不在结果中, 就需要从剩下的 $n-1$ 个元素中抓取 k 个元素 (C_{n-1}^k)。要求分别采用以下方法计算, 并进行三种方法所需时间的经验分析。

1.1.2 解决问题所用的算法设计方法及基本思路

此问题可以通过下面的算法实现:

1. **递归算法** 考虑计算 C_n^k 的情况，$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, 则可以递归的调用此方法或者函数，直到 n 和 k 有一个为 1 就返回 1, $k = 1$ 时就返回 n 。

2. **备忘录方法** 需要借助上面的**递归算法**, 当 C_{n-i}^{k-j} 不等于 0 时，就计算这个值，然后保存到一个数组中，其它公式如果需要它的值则直接从数组中调用即可，不需要再次计算。

3. **迭代算法** 通过一个结果二维数组保存从 C_1^1 到 C_n^k 的所有值，从小到大计算，逐行填表。

1.1.3 采用的数据结构描述

在**递归算法**中，系统底层采用栈存储递归的数据，然后逐层返回；在**备忘录方法**中采用了一个额外的二维数组来保存中间过程的值；在**迭代算法**中，也是通过一个二维数组实现逐行填表，计算二项式公式的值。

1.1.4 算法描述

1. 递归算法

算法 CalculateBinomialRecursion (k, n)

```
// 计算二项式公式使用递归
// 输入：二项式公式的 k,n
// 输出：二项式公式的值
if k > 0 && n > 0
    if k = 1 and n = 1
        return 1
    else if k = 1
        return n
    else
```

return CalculateBinomialRecursion $(k-1, n-1)$
+ CalculateBinomialRecursion $(k, n-1)$

return 0

2. 备忘录算法

算法 CalculateBinomialMemo (k, n)

// 计算二项式公式使用备忘录方法
// 输入：二项式公式的 k, n
// 输出：二项式公式的值
// 注意：$C[n, k]$ 用于保存中间过程的值，减少重复计算
if $k > 0$ && $n > 0$
if $k = 1$ and $n = 1$
return 1
else if $k = 1$
return n
else
$c[k, n]$ = CalculateBinomialRecursion $(k-1, n-1)$
+ CalculateBinomialRecursion $(k, n-1)$
return $c[k, n]$
return 0

3. 迭代算法

算法 CalculateBinomialIteration (k, n)

// 计算二项式公式使用迭代方法
// 输入：二项式公式的 k, n
// 输出：二项式公式的值
kns ← array **for** $i = 0$ to n
for $j = 0$ to k
if $i < j$
$kns[i][j] = 0$
else if $j == 1$
$kns[i][j] = i$
else if $j == j$
$kns[i][j] = 1$
else
$kns[i][j] = kns[i-1][j-2] + kns[i-1][j-1]$
return kns[n-1][k-1]

1.1.5 算法的时间空间复杂度分析

下面是几种算法的运行时间的时间和 k 曲线图

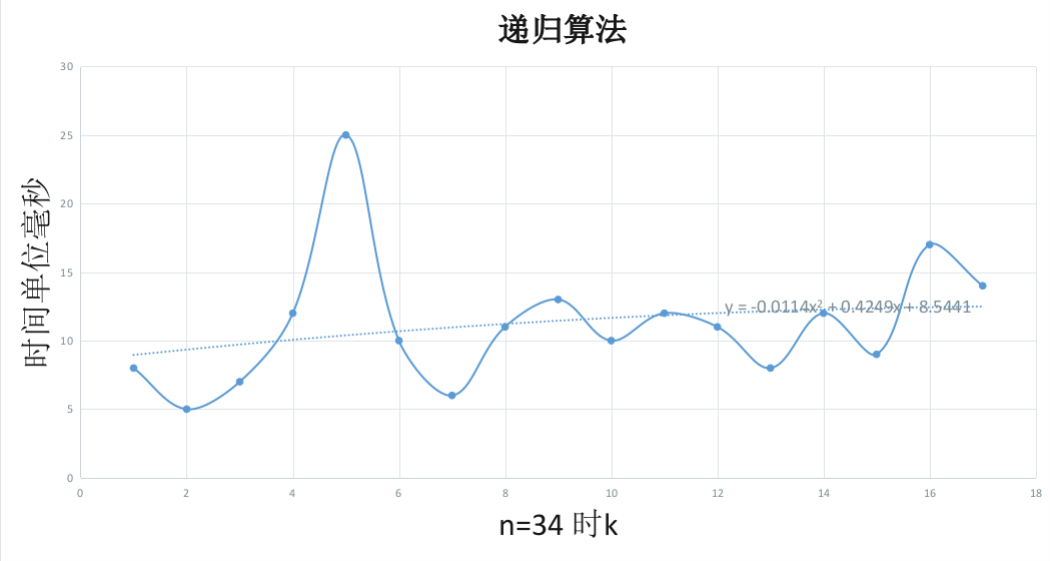

图 1: 递归算法

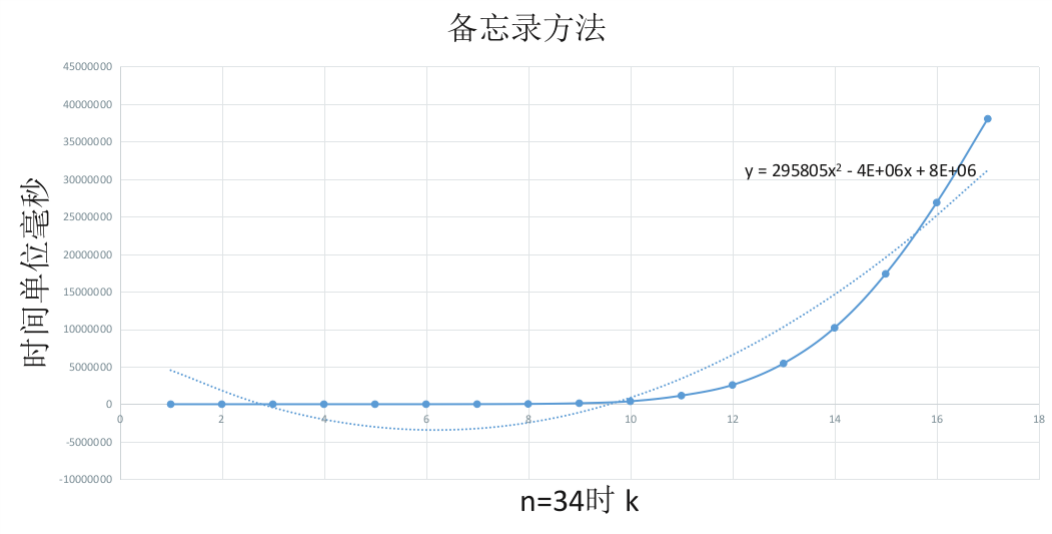

图 2: 备忘录方法

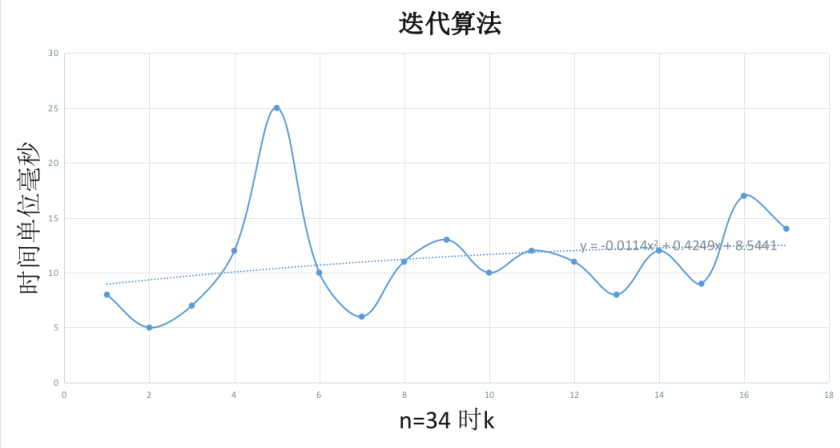

图 3: 迭代算法

通过分析，可以得出**递归算法**的时间复杂度 $\Theta(n) = n^2$，空间复杂度为 $O(1)$；**备忘录方法**的时间复杂度 $\Theta(n) = n^2$ 空间复杂度为 $O(n)$；**迭代算法**的时间复杂度 $\Theta(n) = n^2$，空间复杂度为 $O(n^2)$。

1.1.6 算法实例

输入：

```
n=16
k=7
```

输出:

```
=== RUN   TestCalculateBinomialRecursion
2018/12/25 16:23:54 C(7, 16)=CalculateBinomialBrute=11440, CalculateBinomialRecursion=11440
--- PASS: TestCalculateBinomialRecursion (0.47s)
=== RUN   TestCalculateBinomialMemo
2018/12/25 16:23:54 C(7, 16)=CalculateBinomialBrute=11440, CalculateBinomialMemo=11440
--- PASS: TestCalculateBinomialMemo (0.00s)
=== RUN   TestCalculateBinomialIteration
2018/12/25 16:23:54 C(7, 16)=CalculateBinomialBrute=11440, CalculateBinomialIteration =11440
--- PASS: TestCalculateBinomialIteration (0.00s)
```

图 4: 测试计算二项式的算法

1.2 绘制简单的分形树

1.2.1 问题描述

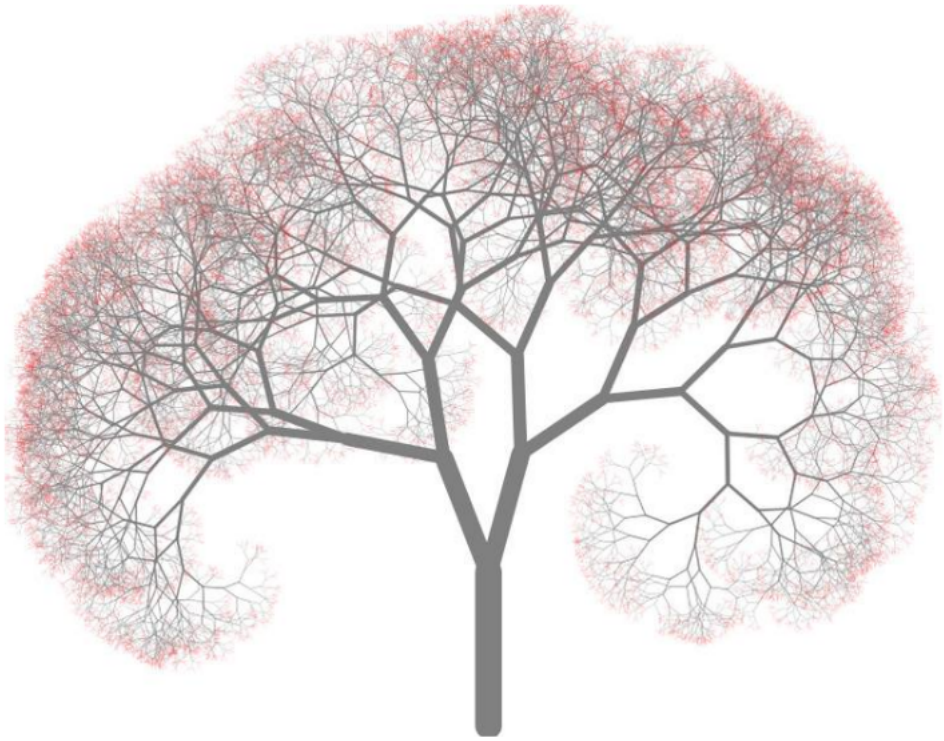

图 5: 分形树示例

先垂直绘制一根线段, 然后在线段顶端向右一定倾角绘制一根线段, 长度分别为原线段的 k 倍. 再同样的, 在线段左侧以固定倾角绘制一根线段, 如此反复, 直至线段长度小于某个较小的值。其中, 线条颜色以及长度, 夹角 (例如产生某个范围的随机数) 都可以自行进行微调。

1.2.2 解决问题所用的算法设计方法及基本思想

使用递归的思想，假定现在正在一个节点，那么随机生成两个角度和长度，调用画图的方法画出两个子节点，然后再递归的画出这两个子节点的图。

1.2.3 采用的数据结构描述

无。

1.2.4 算法描述

绘制简单的分形树算法

算法 ddraw $(x, y, length, angle)$

// 绘制简单的分形树递归算法
// 输入：节点的 x, y 表示坐标, $length$ 表示长度, $angle$ 表示偏的角度
// 输出：绘制出的分形树的图像
if $length < minilength$// 最短的边长度，退出条件
    return
x_2, y_2, x_3, y_3 = getRandomSon(x,y) // 获取子节点的坐标
draw$(x, y, length, angle)$// 画出当前节点
ddraw$(x_2, y_2, randomLength_2, randomAngle_2)$ // 画左边子节点
ddraw$(x_3, y_3, randomLength_3, randomAngle_3)$ // 画右边子节点

1.2.5 算法的时间空间复杂度分析

假定树的深度为 n ，那么此算法的时间复杂度为 $\Theta(n) = 2^n$，空间复杂度也为 2^n 。

1.2.6 算法实例

下面是一个随机生成的分形树的实例：

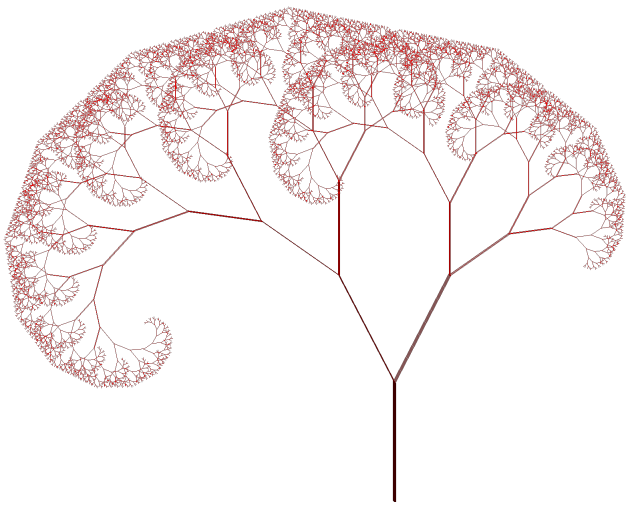

图 6: 生成的分形树的实例

2 遍历

2.1 8 品脱问题

2.1.1 问题描述

西蒙. 丹尼斯. 泊松是著名的法国数学家和物理学家。据说在他遇到某个古老的谜题之后, 就开始对数学感兴趣了, 这个谜题是这样的: 给定一个装满水的 8 品脱壶以及两个容量分别为 5 品脱和 3 品脱的空壶, 如何通过完全灌满或者到空这些壶从而使得某个壶精确地装有 4 品脱的水? 用广度优先查找来求解这个谜题。要求在输出结果中包含广度优先的遍历过程 (结点的遍历顺序)。

2.1.2 解决问题所用的算法设计方法及基本思想

采用广度有序一层一层遍历的思想，考虑某一层中的某个节点（这棵树中每个节点都保存着当前每个瓶子的信息，包括容量和已经使用了多少），这几个瓶子可以互相倒水，只有能够倒成功的才能进行下一步，并且倒水后的节点不能已经存在，这样一层一层的倒水，当某个节点满足添加时即可退出，并倒序回去得到倒水过程。

2.1.3 采用的数据结构描述

使用了一个栈来保存树的节点，一个数组来保存已经产生的结果（用来判断这个节点是否已经产生过了），一个结构体来保存瓶子的容量和用量。

2.1.4 算法描述

找到一个倒水过程使其符合给定的结果

```
算法 find (nodes)
// 找到一个倒水过程使其符合给定的结果
// 输入：一个或多个起始状态节点数组 nodes
// 输出：到给定结果的倒水过程
if len(nodes) ← 0
    return null
cloneNode ← clone(nodes) // 复制当前所有节点信息
newNodes ← array // 创建一个新的数组保存生成的所有节点
for node in cloneNode do // 遍历这层的所有节点
    for bottle1 in node // 每个节点的瓶子互相倒水
    for bottle2 in node // 每个节点的瓶子互相倒水
        if bottle1 ≠ bottle2 do // 瓶子不同才倒水
        // 先克隆，后倒水
        newNode=pour(bottle1,bottle2,node)
```

if newNode isnot in nodeLists **do**
nodeList.add(newnode) // 全局保存的所有节点
newNodes.add(newnode)
newNode.parent = node // 使当前节点的父节点志向 node
if Has(newNode, number) // 判断当前节点是否满足退出条件
return newNode
find(newNodes)

2.1.5 算法的时间空间复杂度分析

2.1.6 算法实例

输入：

```
// 瓶子数组 容量、用量
[{122,99}, {50,25}, {12,1}, {2,0}, {5,4}]
40 // 需要的结果
```

输出：

```
bottles:[{112 99} {50 25} {12 1} {2 0} {5 4}]
bottles:[{112 74} {50 50} {12 1} {2 0} {5 4}]
bottles:[{112 72} {50 50} {12 1} {2 2} {5 4}]
bottles:[{112 72} {50 39} {12 12} {2 2} {5 4}]
bottles:[{112 72} {50 38} {12 12} {2 2} {5 5}]
bottles:[{112 72} {50 40} {12 12} {2 0} {5 5}]
```

图 7: 品脱倒水问题结果

2.2 24 点问题

2.2.1 问题描述

用户输入 4 位个位数 (1 9), 四个数之间只能通过 +,-,*,/运算进行连接, 请输出四则运算表达式, 其求值结果为 24。然后输出所有求值结果为 24 的组合。你能不能不通过四重循环来产生这 4 个参与运算的数。

2.2.2 解决问题所用的算法设计方法及基本思想

可以使用如下的不在求解：

1. 产生 4 个数的排列;
2. 产生运算符的排列 (4 种运算符, 任选 3 个);
3. 考虑加括号方式, 纸上推演一下, 有 5 种加括号方式, 如果对 4 个数的排列没有约束, 加括号方式也可以简化为三种 $< (a+b)+(c+d), ((a+b)+c)+d, (a+(b+c))+d >$ $a+((b+c)+d), a+(b+(c+d))$ 是被去掉的两种, 这里加号表示运算符。

然后循环数的排列、运算符的排列和三个加括号的方式，一一计算，即可以求出哪些可以满足 24 点。

2.2.3 采用的数据结构描述

使用数组来保存生成的数的排列和运算符的排列，使用字符串数组来保存满足 24 点的表达式。

2.2.4 算法描述

计算 24 点

算法 CalculatePoints(a, b, c, d)

// 计算 24 点
// 输入：a, b, c, d 四个个位数
// 输出：所有满足 24 点的表达式
resultList ← array // 保存符号 24 点的表达式数组
expressionList ←getPerms(a, b, c, d) // 获取 a, b, c, d 构成的排列
operatorList ← getOperator("+","-","*","/") // 获取 $+, -, *, /$ 构成的排列
for $i \leftarrow 0$ to len(expressionList) **do**
    for $j \leftarrow 0$ to len(expressionList) **do**
        if calculate(expressionList[i], operatorList[j]) = 24 //$(a+b)+(c+d)$
        resultList.add(expressionList[i])
        if calculate(expressionList[i], operatorList[j]) = 24 // $((a+b)+c)+d$

resultList.add(expressionList[i])
if calculate(expressionList[i], operatorList[j]) = 24 //$(a + (b + c)) + d$
resultList.add(expressionList[i])

return resultList

2.2.5 算法的时间空间复杂度分析

输入有 4 个数字，可以产生 64 种组合，4 和运算符可以参数 64 种组合，还有 3 种情况，所以当输入梳子个数为 n 时，则有 $n! * C_n^{n-1} * (n-1)! * c = cn!^2$, 所以时间复杂度 $\Theta(n) = cn!^2$，空间复杂度为 $cn!^2 + k$ 。

2.2.6 算法实例

输入:

```
4,5,6,7 24
```

输出:

```
(4+(4/5))*6 (4-(4/5))*6 (4+(4/7))*6 (4-(4/7))*6 (6+6)*(7-5)
(6*6)-(7+5) (6*6)-(5+7) (6/(6-5))*4 ((6/6)+5)*4 (6+(6/7))*4
(6-(6/7))*4 (7-(7/5))*4 ((7/7)+5)*4 (7-7)+(4*6) ((7-7)+4)*6
(7-7)+(6*4) ((7-7)+6)*4 (7-(7/6))*4 (7*7)/(6-4) (5*5)-(7/4)
(5-5)+(6*4) ((5-5)+6)*4 (5*5)-(6/4) (5-5)+(4*6) ((5-5)+4)*6
(5-(5/4))*6 ((5*5)-7)+6 (5*5)-(7/6) (5*5)+(6-7) ((5*5)+6)-7
```

3 动态规划

3.1 最长回文子序列问题

3.1.1 问题描述

如果一个子序列从左向右和从右向左读都一样, 则称之为回文。例如, 序列 $ACGTGTCAAAATCG$ 有很多回文子序列, 比如 $ACGCA$ 和 $AAAA$ 。请给出一个算法, 求出最长的回文子序列。

3.1.2 解决问题所用的算法设计方法及基本思想

将字符串中的每一个字符最开始时都当成一个回文子序列，然后遍历整个字符串，再遍历回文子序列的数组，找到从字符 i 到这个字符的最大回文子序列，最后提取出最大的子序列。

3.1.3 采用的数据结构描述

使用一个 map 来保存子序列数组的下标，key 为 start，value 为 end。

3.1.4 算法描述

找出最长的回文子序列

```
算法 find(chars)
// 找出最长的回文子序列
// 输入：chars 需要寻找最长回文子序列的字符串
// 输出：chars 中最长的回文子序列
result ← array // 保存最长回文子序列的数组
resultMap ← map // 保存最长回文子序列的 start 和 end
// 初始化最长回文子序列为字符串第一个元素
for i ← 0 to len(chars) do
        resultMap[i] ← = i
for index ← 1 to len(chars) do
        for k, v range resultMap
        find(k, v, char, resultMap, index)
// 找到 resultMap 中最长的元素 value-key
longestLength ← findLongestLength(resultMap)
// 找到 chars 中的最长回文子序列，长度为 longestLength
result ← findLongestSubString(longestLength,resultMap,chars)
return result
```

3.1.5 算法的时间空间复杂度分析

首先需要一个外层字符串的遍历，然后是寻找最长的回文子序列，其时间复杂度为 $\Theta(n) = n^3$，空间复杂度为 n^2。

3.1.6 算法实例

下面是实验测试过程中的一些输入输出:

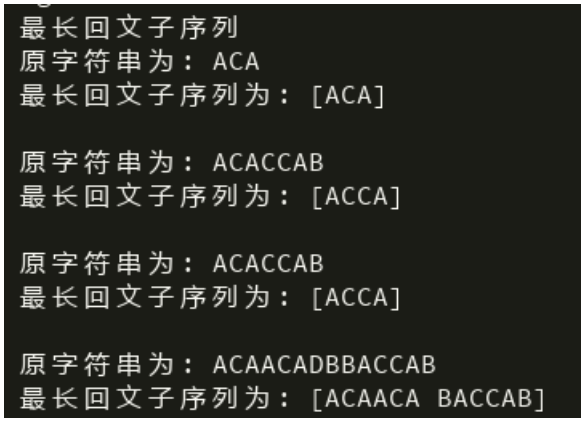

图 8: 最长回文子序列

3.2 小美购物问题

3.2.1 问题描述

小美近来疯狂购物, 信用卡上获得积分 100000。在积分商城中有许多物品可以选兑。例如食用油, 大米, 钢笔, 电烤炉, 研磨机, 热水壶等等。信用卡积分就快过期了, 小美想将这一万积分尽量用光, 请问小美应该怎么办?

小美在挑选物品的过程中, 发现有些积分和实际价格的比例大概是 100:1, 继而发现有些商品在积分商城里购买并不划算, 不如去京东购买。小美为了这 100000 积分也是拼了, 将所有商品在京东的价格罗列了出来, 同时罗列了所有商品的积分。此时小美又应该怎么挑选, 使得获得的价值最大?(或许你应该想想价值是什么?)

3.2.2 解决问题所用的算法设计方法及基本思想

此问题跟背包问题非常类似，都是价值最大化的问题，在一定的积分（重量）下，选择合适的商品（具有一定的积分和价值（重量和价值）），使用一层一层的填表，即

$$F(i,j) = \begin{cases} F(i,j)=maxF(i-1,j),v_i+F(i-1,j-w_j),j-w_j\geq 0 \\ F(i-1,j), \qquad\quad j-w_j<0 \end{cases}$$

3.2.3 采用的数据结构描述

使用一个二维数组保存填表的信息（$F(i,j)$）的值。

3.2.4 算法描述

最大价值化问题

算法 FindMaxValue1(i,j,prices, credits, totalCredit)

// 找出价值最大的商品集合
// 输入：prices 每个商品的价格
// 输入：credits 每个商品的积分
// 输入：totalCredit 一共拥有的积分
// 输出：最大价值商品的集合
// 注意：F 数组除了行 0 和列 0 用 0 初始化外，其它全部用-1 初始化

if $F(i,j) < 0$ **do**
    if $j < credits[i]$ **do**
        $value \leftarrow FindMaxValue(i-1, j, prices, credits, totalCredit)$
    else do
        $value \leftarrow max(FindMaxValue1(i-1, j)),$
$prices[i] + FindMaxValue(i-1, j-credits[j-prices[i]])$
    $F[i,i] \leftarrow value$
return $F[i,j]$

3.2.5 算法的时间空间复杂度分析

此算法基于填表的查询相加，时间复杂度为 $\Theta(n) = (n+1)^2$，空间复杂度也为 $(n+1)^2$。

3.2.6 算法实例

下面是实验测试时的一些输入输出：

```
考虑每种商品只能选择一次的情况
总积分为: 10000
商品价格列表为:  [200 400 500 250 500 60 300 200]
商品积分列表为:  [5000 1000 2500 500 3000 1000 2000 1200]
最大价值列表商品选择的为:  [6 5 4 3 2 1]
最大价值为:  2010
```

图 9: 最大价值问题实验输入输出

4 分支限界与回溯

4.1 n 个处理机和 k 个任务问题

4.1.1 问题描述

假设有 n 个任务由 k 个可并行工作的机器完成。完成任务 i 需要的时间为。试设计一个算法找出完成这 n 个任务的最佳调度, 使得完成全部任务的时间最早。例如, 输入为

```
7 3
2 2 14 4 16 6 5 3
```

表示有 7 个任务, 有 3 台可并行工作的机器。每个任务需要的完成时间在第二行。输出文件示例

```
1
17
```

4.1.2 解决问题所用的算法设计方法及基本思想

此问题采用的方法是分支限界，先然一个处理机处理所以任务，然后从这个处理机中分出一个任务分给其它的处理机，这样一次一次的分配，直到三个处理机中最大的为所有情况中最小时，返回，并且在此过程中记录每一次每个处理机中的处理情况。

4.1.3 采用的数据结构描述

使用一个二维数组来保存每个处理机处理任务的信息。

4.1.4 算法描述

处理机调度

算法 nkSchedeling (dep, lens, t, n, k, result)

```
// 找出价值最大的商品集合
// 输入：dep, lens, t, n, k, result
// 输出：result 包含处理机调度的数组
if dep = n do
        tmp ← comp(lens) // 找到一次中的最大时间
        if tmp < bestSoFar
                bestSoFar = tmp
        return
for i ← 0 to k do
        lens[i] += t[dep]
        result[dep] = i
```

if $lens[i] < bestSoFar$ **do**
    nkSchedeling(dep+1, lens, t, n, k, result)
lens[i] -= t[dep]
delete(result[dep])

4.1.5 算法的时间空间复杂度分析

时间复杂度 $\Theta(n) = n^k$，空间复杂度为 n^2

4.1.6 算法实例

输入：

```
处理机个数：n = 3
任务个数：k = 7
每个任务所花时间：t = {2, 14, 23, 11, 9, 3, 4, 16, 6, 5, 3}
```

输出:

```
最短作业时间为： 23
分配情况如下： [[3 4 2 14] [23] [11 9]]
```

图 10: 处理机调度

5 附加题目

5.1 学习超市选址问题

5.1.1 问题描述

学校超市选址问题 (带权有向图的中心点) 设计内容: 对于某一学校超市, 其他各单位到其的距离不同, 同时各单位人员去超市的频度也不同。请为超市选址, 要求实现总体最优。设计要求:

1. 设计该问题的核心算法;
2. 设计可视化的界面, 界面中能有效显示学校超市可设立的地点和各单位的位置以及它们之间的有效路径;
3. 程序能自动计算出最优设立点, 并最好以图示化方式演示。

5.1.2 解决问题所用的算法设计方法及基本思想

本题目采用 Floyd 最短路径算法，把每个点到其它点的距离全部算出来，然后比较，选出距离最短的点。

5.1.3 采用的数据结构描述

使用二维矩阵来保存点与点之间的距离。

5.1.4 算法描述

找出带权有向图的中心点

```
算法 findShortest (maxtrix)
// 找出带权有向图的中心点
// 输入：一个二维数组表示每个点之间的距离
// 输出：maxtrixS 表示最终的点与点之间的最终距离
// 输出：pPath 表示最终距离到最终距离的路径
maxtrixS ← matrix
pPath ← new array
for k ← 0 to len(matrix) do
        for i ← 0 to len(matrix) do
                for j ← 0 to len(matrix) do
                        min,po ← min(maxtrixS[i,j],maxtrixS[i,k]+maxtrixS[k,j])
                        maxtrixS[i,j] ← minP
                        append(path[i,j], k)
```

return maxtrix, pPath

5.1.5 算法的时间空间复杂度分析

本题的时间复杂度为 $\Theta(n) = n^3$，空间复杂度为 n^3。

5.1.6 算法实例

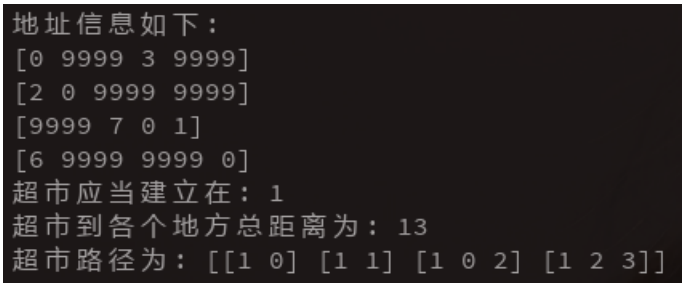

图 11: 超市选址实例

6 课程设计总结

本次的算法课程设计让我更加深刻的理解的一些基本的算法思想，包括蛮力法、分支限界、回溯法、动态规划等。在每完成一道算法题时，都会对该算法思想具有更深刻的认识和体会，即使有些比较困难，但最终都会得到很大的收获。