# 

## 《算法分析理论及应用》课程实验报告

**班级：软工182 姓名：邓棋 学号：2018081062**

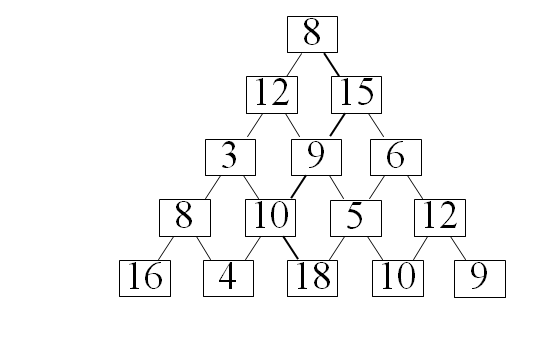
**一、实验题目**

1. 动态规划解决数塔问题。
2. 动态规划解决0/1背包问题。

**二、实验内容**

**1. 数塔问题。（完成实验代码、伪代码）**

从数塔的顶层出发，在每一个结点可以选择向左走或向右走，一直走到最底层，要求找出一条路径，使得路径上的数值和最大。（请完成算法的时间复杂度、伪代码以及代码的编写）。



**2. 0/1背包问题。（完成实验代码、伪代码）**

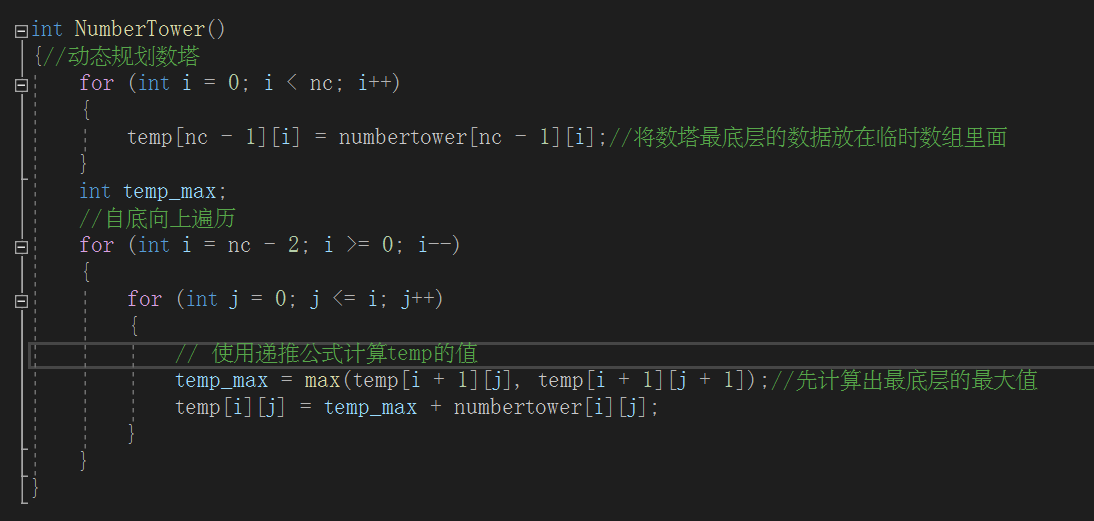
给定n种物品和一个背包，物品i的重量是wi，其价值为vi，背包的容量为C。背包问题是如何选择装入背包的物品，使得装入背包中物品的总价值最大?如果在选择装入背包的物品时，对每种物品i只有两种选择：装入背包或不装入背包，即不能将物品i装入背包多次，也不能只装入物品i的一部分。例如：有5个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为10，动态规划法求解0/1背包问题。

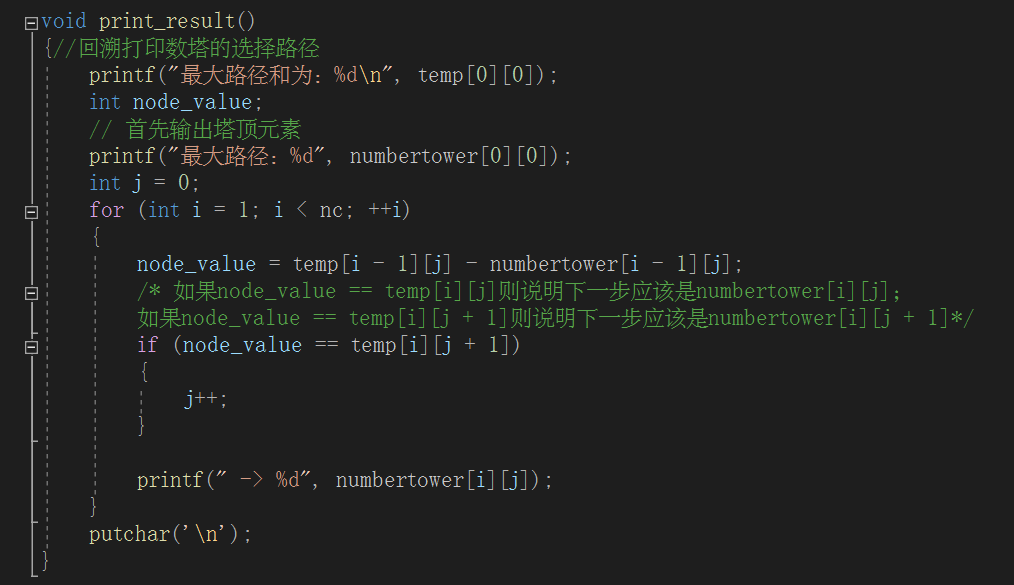
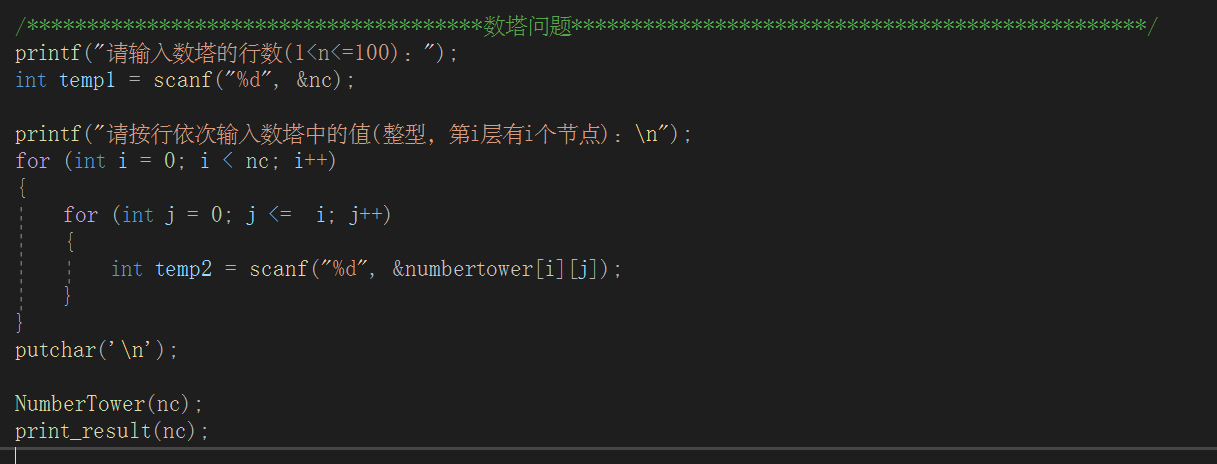
**三、实验目的**

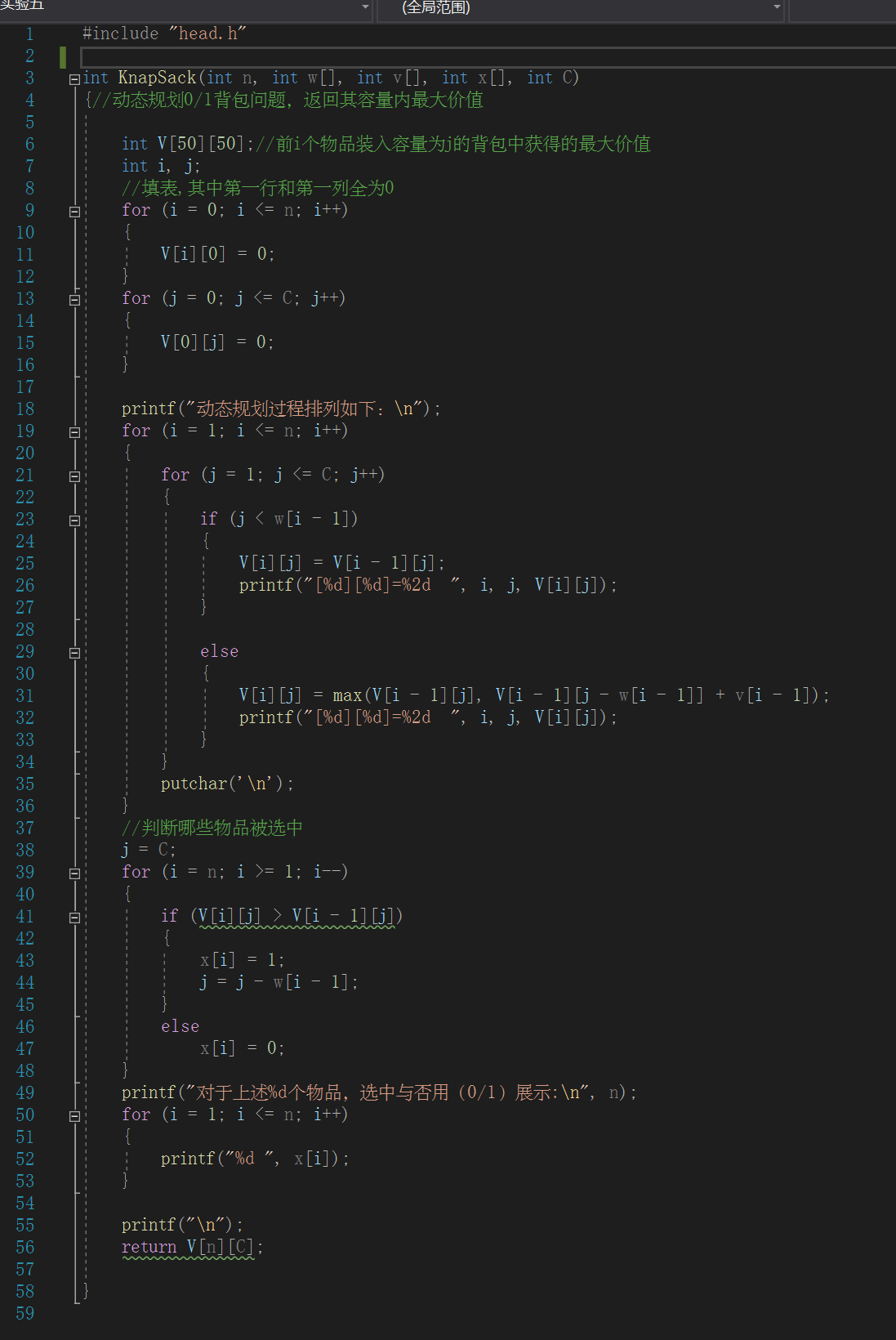
1. 理解动态规划问题的思想，算法策略。

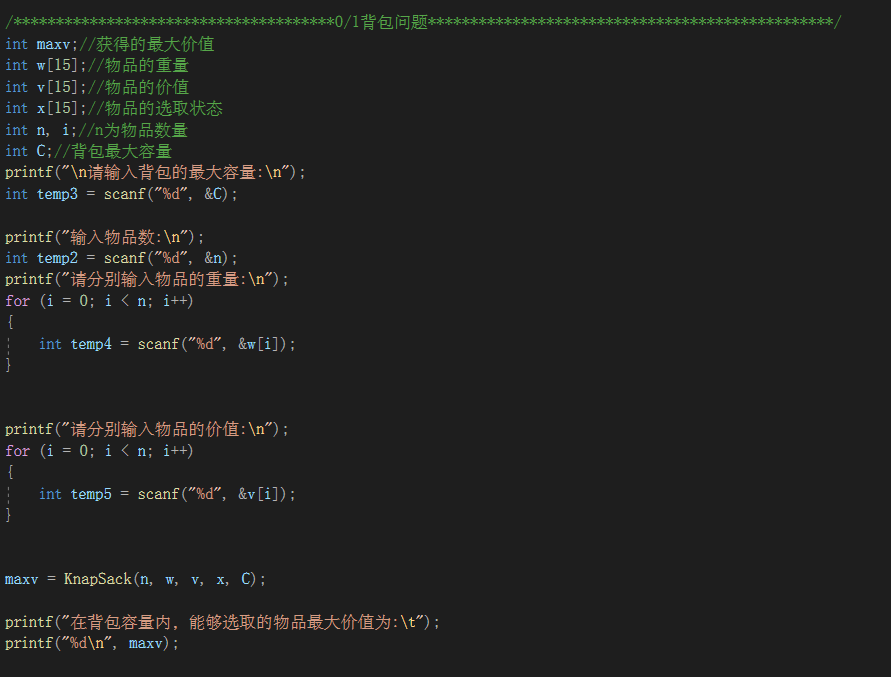
2. 掌握利用动态规划解决问题的基本思想，会用高级语言对算法进行描述，并对算法复杂度（时间和空间）进行分析。

**四、实验代码**

**数塔问题：**

****

**0/1背包问题：**

****

**五、实验总结**

书写以下实验总结：1、算法伪代码编写；2、算法设计策略描述；3、算法时空复杂度分析。4、遇到的问题及解决方法。

1. **算法伪代码编写**

**数塔问题：**

1. 初始化数组temp的最后一行为数塔的底层数据：

for (j = 0; j < n; j++)

temp[n-1][j] = numbertower[n-1][j];

2. 从第n-1层开始直到第 1 层对下三角元素temp [i][j]执行下述操作：

2.1 temp[i][j] = numbertower [i][j] + max{maxAdd[i+1][j], maxAdd[i+1][j+1]}；

2.2 如果选择下标j的元素，则path[i][j] = j，

否则path[i][j] = j+1；

3. 输出最大数值和temp[0][0]；

4. 根据path数组确定每一层决策的列下标，输出路径信息；

**0/1背包问题：**

V[0][] ← {0}

V[][0] ← {0}

for i←1 to n

for j←1 to C

if (j < w[i - 1])

V[i][j]=V[i-1][j]

打印信息

else

V[i][j] = max(V[i-1][j], V[i-1][j - w[i-1]] + v[i-1]);

打印信息

for i←n to 1

if (V[i][j] > V[i - 1][j])

x[i] = 1;

j = j - w[i - 1];

else x[i] = 0

打印x[i]

retuen V[n][C]

1. **算法设计策略描述**

**数塔问题：**

定义两个二维数组，一个存储数塔原始数据（i层有i个元素），一个存储动态规划过程中的数组（临时数组）。动态规划数塔的过程：先将数塔最底层的数据放在临时数组里面，自底向上遍历数组，取相邻的两个元素的最大值与上一层能够联系到它们的元素相加，这样两两两两与上层配对，选出一个最大的，再在当前层重复上述过程，遍历到塔顶元素时，就找到了最大路径和。回溯打印数塔的选择路径过程：最大路径和就是临时数组的首个元素，路径第一个元素自然是塔顶的首个元素，然后按层来遍历，用临时数组相同位置的变量减去数塔变量，得到的当前选择路径的值，然后与临时变量的下一层元素大小比较，即可知选择了左路径还是右路径。

**0/1背包问题：**

0/1背包问题要抽象化成为动态规划表，其数据就是当前背包在承重范围内的最大价值，i表示在第i个物品之前选择，j表示当前的背包容量，i=0的行和j=0价值自然是0；寻找递推关系式，当前包的容量比该商品体积小，装不下，此时的价值与前i-1个的价值是一样的，即V(i,j)=V(i-1,j)；还有足够的容量可以装该商品，但装了也不一定达到当前最优价值，所以在装与不装之间选择最优的一个，即V(i,j)=max｛ V(i-1,j)，V(i-1,j-w(i))+v(i)｝。按照这个递推式，选择动态规划表中数据最大的元素（一定位于表的右下方），即是最优解，然后最优解回溯找出最优解的组成部分，V(i,j)=V(i-1,j)时，说明没有选择第i 个商品，则回到V(i-1,j)； V(i,j)=V(i-1,j-w(i))+v(i)实时，说明装了第i个商品，该商品是最优解组成的一部分，随后我们得回到装该商品之前，即回到V(i-1,j-w(i))；一直遍历到i＝0结束为止，所有解的组成都会找到。

1. **算法时空复杂度分析**

**数塔问题：**

**时间复杂度：**T(n) = 1 + 2 + 3 + … + n – 1 + n = n^2 / 2 + n /2

所以它的时间复杂度为O（n^2）

**空间复杂度：**内存空间的主要消耗是存放数塔每个元素的数组以及动态规划过程中使用的临时数组，n层的数塔，第i层有i个元素，总量也是一个等差数列，所以它的空间复杂度为O（n^2）。

**0/1背包问题：**

**时间复杂度：**主要的时间开销是在动态规划制表过程中，T（n）= nC，n为物品个数，C为背包容量是一个常量，所以它的时间复杂度为O（n）。

**空间复杂度：**规划0/1背包问题使用的数组大小都是预先设定的一个合理的范围，都是常数阶的，而只有物品的数量是一个不确定的未知数n，它的空间复杂度为O（n）。

1. **遇到的问题及解决方法**

一开始直接选择最底层最大的值，然后与它能够关联的两个上层元素做加法，这样来说在一些特定的情况下可以计算出最佳路径，但是如果存在一个特别大的数，是下层最大元素无法联系的，那么就不能直接取下层的最大，而应该和上层先做加，再取最值，这样层层遍历，才能得到最优解。

**六、算法策略的英文描述（字数>200）**

**Number tower problem**

Define two two-dimensional arrays, one to store the raw data of the tower (i-level has i elements), and one to store the array in the dynamic programming process (temporary array). The process of dynamically planning the number towers: first put the bottommost data of the number towers in the temporary array, traverse the array from the bottom up, take the maximum value of two adjacent elements and add the elements that can be connected to the previous layer, In this way, pair up with the upper layer one by one, select the largest one, and repeat the above process on the current layer. When you traverse to the element at the top of the tower, you find the maximum path sum. The process of selecting the path of the number tower is traced back: the largest path sum is the first element of the temporary array. The first element of the path is naturally the first element at the top of the tower, and then iterate by layers. Use the variable at the same position in the temporary array to subtract the number. Tower variable, get the value of the currently selected path, and then compare it with the element size of the next layer of the temporary variable to see whether the left path or the right path is selected.

**The 0/1 knapsack problem：**

The 0/1 knapsack problem should be abstracted into a dynamic planning table. The data is the maximum value of the current knapsack within the load-bearing range. I indicates the selection before the i-th item, j indicates the current knapsack capacity, i = 0, and j= 0 value is naturally 0; looking for a recurrence relationship, the current package's capacity is smaller than the product's volume and cannot be loaded. At this time, the value is the same as the value of the previous i-1, that is, V (i, j) = V (i-1, j); there is enough capacity to install the product, but it does not necessarily reach the current optimal value, so choose the best one between installed and not installed, that is, V (i, j) = max ｛V (i-1, j), V (i-1, j-w (i)) + v (i)｝.According to this recursive formula, select the element with the largest data in the dynamic programming table (must be located at the lower right of the table), which is the optimal solution, and then the optimal solution backtracks to find the components of the optimal solution, V (i, j) When V = i (i-1, j), it means that if the i-th product is not selected, V (i-1, j) is returned; v (i) is real-time, indicating that the i-th product is installed, which is part of the optimal solution composition. Then we have to go back to the product before loading, that is, return to V (i-1, j-w (i)); Iterate until the end of i = 0, and the composition of all solutions will be found.