《大学计算机基础》（常规班）实验指导书

* 注意：每个实验任务需要创建单独创建Python文件，命名规则是：f\_实验任务尾号.py
* 例：本次实验的任务6-1~6-3应该创建3个py文件，文件名分别是：f\_1.py~ f\_3.py
* 每次实验后，需按照模板完成实验报告，并提交（本次实验报告命名规则是：

“实验6\_学号姓名.docx”,例：“实验6\_12051211王一.docx”）。

# 实验6 算法设计与实现（2）

## 1、实验目的

（1）了解较复杂算法的设计

（2）掌握用Python语言进行算法实现

## 2、实验任务

### 实验任务6-1 排序

**问题描述：**

高考成绩在排序时，优先以总分排序，遇到总分相同的情况，就再比较单科成绩。

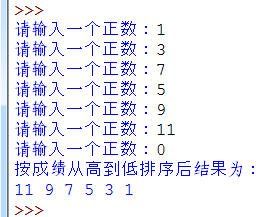
请你设计算法，编写**两个程序**，分别能够实现：

（1）输入多个正整数，（0表示输入结束），将其排序，在同一行中从大到小输出。

（2）输入姓名、总成绩、数学成绩，输入多组数据（姓名为""（空字符串）表示输入结束），将其排序。优先以总成绩排序，总成绩相同时以数学成绩为依据。按成绩从高到低排序，在同一行中依次输出姓名。

**要求：采用伪代码描述你的算法。多次输入数据，将程序运行结果截图后，粘贴到实验报告中。**

程序运行结果如下：



### 实验任务6-2 编写程序实现给定区间二分查找

**问题描述：**

（1）构造一个有序整数序列，例如[1,2,3,5,7,9]，不少于6个元素。

（2）用户输入一个数，在该有序整数序列中用二分搜索查找该数，若找到，则输出其在整数序列中的位置编号；若未找到，则输出“NO FOUND!”

**实验指导：**

设R[low..high]是当前的查找区间

（1）首先确定该区间的中点位置：R[mid]

（2）然后将待查的K值与R[mid]比较：若相等，则查找成功并返回此位置，否则须确定新的查找区间，继续二分查找，具体方法如下：

①若R[mid] >K，则由表的有序性可知R[mid..n]均大于K，因此若表中存在关键字等于K的结点，则该结点必定是在位置mid左边的子表R[1..mid-1]中，故新的查找区间是左子表R[1..mid-1]。

②类似地，若R[mid] <K，则要查找的K必在mid的右子表R[mid+1..n]中，即新的查找区间是右子表R[mid+1..n]。下一次查找是针对新的查找区间进行的。

### 实验任务6-3 动态规划（文科选做）

**问题描述：**

如图1所示是一个捡金币的游戏。图中每个圆圈代表一个位置，数字代表了这个位置的金币数。从最上面一层（第0层），可以任意选择一个点为起点。你只能往下一层继续前进，不能在同一层走过两个位置。你只能沿着图中的线走。请问：当到达最下面一层（第3层）后，可以获得的金币数量最大为多少？

输入：先输入层数，再输入各个层各个位置的金币数。（位置的数目与层数应该对应，例如第0层有4个位置，第1层有3个位置等）。

输出：当到达最下面一层（第3层）后，可以获得的最大金币数。

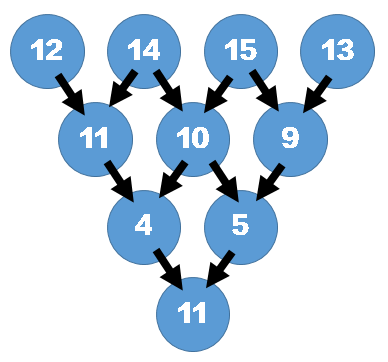


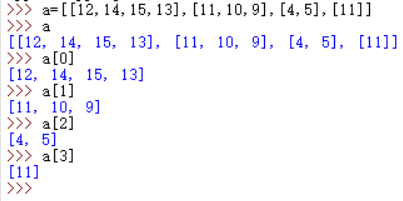
图1 捡金币的游戏示意图

**要求：采用伪代码描述你的算法。将程序运行结果截图后，粘贴到实验报告中。**

**实验指导：**

**（1）储存金币的数据结构**

首先要考虑恰当的数据结构来存储这些数据。可以采用二维列表来存储，例如上图用列表存储即为：



**（2）动态规划**

为了计算走到最后一层（即第3层）的最大金币数，只需要计算走到倒数第二层（即第2层）的两个位置的最大金币数；

为了计算走到倒数第二层（即第2层）的两个位置的最大金币数，只需要计算走到倒数第三层（即第1层）的三个位置的最大金币数；

……

为了计算到走第k层的各个位置的最大金币数，只需计算第k-1层的各个位置的最大金币数；

……

逆向思考，根据走到k层的各个位置的最大金币数，可以计算到走第k+1层各个位置的最大金币数。

根据到第0层各个位置的最大金币数（即各个位置的金币数本身），结合第1层的各个位置金币数，可以计算到第1层各个位置的最大金币数；

……

由此即算出到最后一层那个位置的最大金币数。

**（3）解题时的数据结构**

本题的关键是确定动态规划函数。

需要使用一个变量来记录走到某层各个位置时的最大金币数，仍然可以采用二维列表。例如：scores[i][j]表示走到第i层第j个位置的最大金币数。

仔细分析图1，可以发现，从第1层开始，各个位置都可以从上一层的相邻两个位置到达，而这两个位置的金币数目的多少，将决定其下一层对应位置的最大金币数是多少。例如，如果已经计算到scores[1][1]和scores[1][2]，那么第2层位置1的最大金币数scores[2][1]等于scores[1][1]与scores[1][2]中的最大值加上a[2][1]。

由此可以得到动态规划函数为：

scores[i][j] = max([scores[i - 1][j], scores[i - 1][j + 1]]) + a[i][j]，

其中i=1~n-1，j=0~ n-i-1

可以采用两重for循环，计算scores[i][j]。scores[n-1][0]即到达最下面一层时获得的最大金币数。

程序运行结果如下。

**说明**：为了观察输入过程和动态规划的决策过程，建议打印出中间结果。

