**南京航空航天大学**

数据结构课程设计报告

指导教师：孙涵

班级：1619402

学号：161940219

学生姓名：阴俊晖

日期：2021.1.6

目 录

**一.必做题...................................................................................................2**

**二.选做题.................................................................................................17**

**三.代码行数统计.....................................................................................27**

**四.结束语.................................................................................................28**

# 必做题

* 1. 系统进程设计
     1. 问题描述

设计一个程序，每秒统计一次当前系统的进程状况，并按照内存使用自多到少排序打印输出相关信息。对已经结束的进程，另外给出一个列表，并显示该进程的结束时间和持续时间。

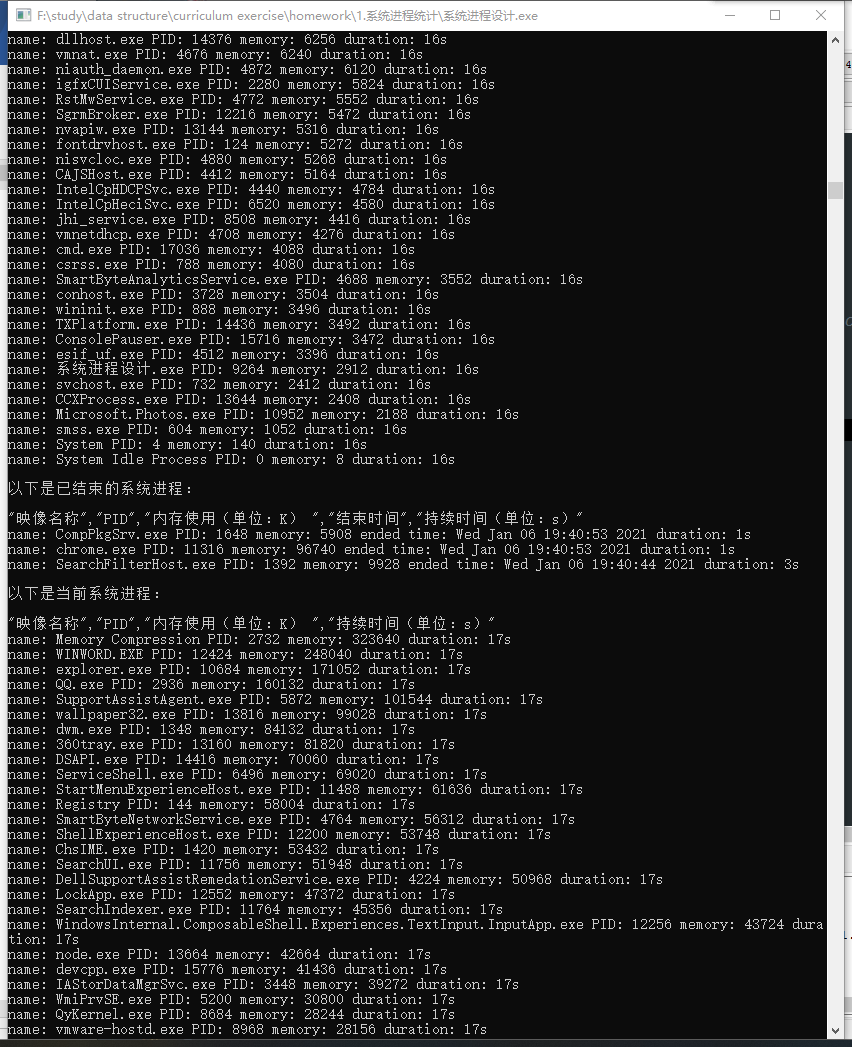
* + 1. 基本要求
* 该题目要求使用两个链式线性表。一个链表存储当前活动进程，要求使用单向链表，并按照内存使用自多到少排序。另外一个链表存储已结束进程，要求使用双向链表，按照持续时间自少到多排序。
* 每秒在窗口内更新一次当前系统进程情况，输出内容包括：进程名，持续时间，内存使用情况。
* 每秒在窗口内更新一次已结束进程情况，输出内容包括：进程名，持续时间，结束时间。
* 注意进程在这两个链表中的切换，一个进程既可被结束，也可以过一段时间后再被运行。
  + 1. 采用的数据结构：单向链表与双向链表
    2. 算法设计思想

Main函数：程序开始时先将所有进程读入当前进程链表中，之后每隔一秒更新一次当前进程链表与已结束进程链表。

链表更新函数：每隔一秒获得一次系统进程，之后依次进行如下操作：1.更新持续运行进程的时间。2.增加新进程（先判断该进程是否是在已结束进程链表中，若在，应将其删去）3.删除已结束的进程。

排序：采用插入排序，创建链表的同时排序。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O(n)

* + 1. 改进方法

统计每个程序所有子进程的情况，包括：进程名，持续时间，结束时间，内存使用情况。

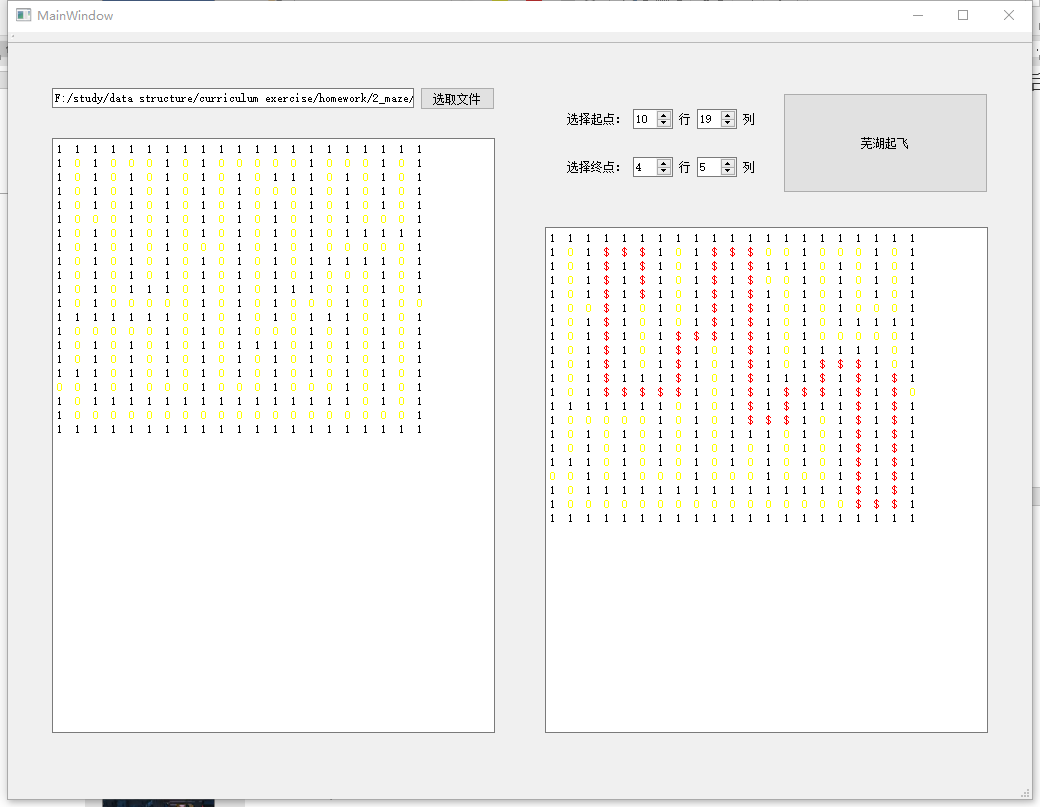
* 1. 迷宫问题
     1. 问题描述

利用栈操作实现迷宫问题求解。

* + 1. 基本要求
* 从文件中读取数据，生成模拟迷宫地图，不少于20行20列。
* 给出任意入口和出口，显示输出迷宫路线。
  + 1. 采用的数据结构：栈、队列
    2. 算法设计思想

核心即利用栈实现DFS的非递归写法遍历迷宫路径即可。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O()

* + 1. 改进方法

当迷宫路径不唯一时，显示最短路径。

* 1. 家谱管理系统
     1. 问题描述

实现具有下列功能的家谱管理系统。

* + 1. 基本要求
* 输入文件以存放最初家谱中各成员的信息，成员的信息中均应包含以下内容：姓名、出生日期、婚否、地址、健在否、死亡日期（若其已死亡），也可附加其它信息、但不是必需的。
* 实现数据的文件存储和读取。
* 以图形方式显示家谱。
* 显示第n 代所有人的信息。
* 按照姓名查询，输出成员信息（包括其本人、父亲、孩子的信息）。
* 按照出生日期查询成员名单。
* 输入两人姓名，确定其关系。
* 某成员添加孩子。
* 删除某成员（若其还有后代，则一并删除）。
* 修改某成员信息。
* 要求建立至少40个成员的数据，以较为直观的方式显示结果，并提供文稿形式以便检查。
* 界面要求：有合理的提示，每个功能可以设立菜单，根据提示，可以完成相关的功能要求。
* 存储结构：根据系统功能要求自行设计，但是要求相关数据要存储在数据文件中。测试数据：要求使用1、全部合法数据；2、局部非法数据。进行程序测试，以保证程序的稳定。
  + 1. 采用的数据结构：树
    2. 算法设计思想

将祖先结点作为家族类的成员变量，每一个结点记录一对夫妻的信息，以及存放后代夫妻结点的数组。

* + 1. 测试数据及运行结果



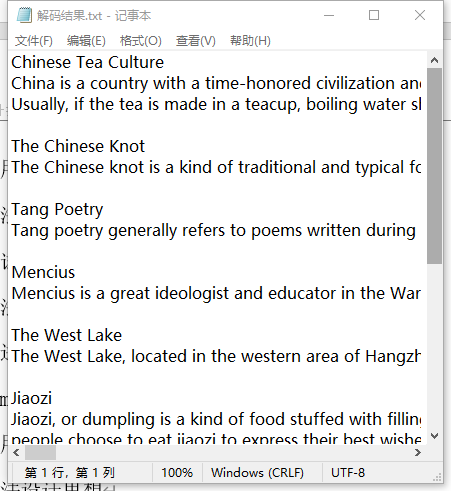
* + 1. 改进方法

可在界面设计上加以优化，力争用图形方式显示家族树。另外由于中国的亲缘关系错综复杂，在查询两个人关系的算法方面也有很大的改进空间。

* 1. Huffman编码与解码
     1. 问题描述
* 对一篇不少于5000字符的英文文章（source.txt），统计各字符出现的次数，实现Huffman编码(code.dat)，以及对编码结果的解码(recode.txt)。
  + 1. 基本要求
* 输出每个字符出现的次数和编码,并存储文件(Huffman.txt)。
* 在Huffman编码后，英文文章编码结果保存到文件中(code.dat)，编码结果必须是二进制形式，即0 1的信息用比特位表示，不能用字符’0’和’1’表示。
* 实现解码功能。
  + 1. 采用的数据结构：Huffman树
    2. 算法设计思想

利用最小堆建树，为每个结点编码时采用层序遍历的方式编码。为每个字符编码时只需在对应的Huffman树上查找其对应的叶结点即可。文件读写时，二进制文件中前四个字节存放数据的有效位数，第五个字节开始存放数据，解码时先读入前四字节确定数据的总位数，再进行解码。解码过程即编码逆过程。

* + 1. 测试数据及运行结果



* + 1. 改进方法

采用其他编码方法如LZW编码。

* 1. 地铁修建
     1. 问题描述

A市有n个交通枢纽，其中1号和n号非常重要，为了加强运输能力，A市决定在1号到n号枢纽间修建一条地铁。

　　地铁由很多段隧道组成，每段隧道连接两个交通枢纽。经过勘探，有m段隧道作为候选，两个交通枢纽之间最多只有一条候选的隧道，没有隧道两端连接着同一个交通枢纽。

　　现在有n家隧道施工的公司，每段候选的隧道只能由一个公司施工，每家公司施工需要的天数一致。而每家公司最多只能修建一条候选隧道。所有公司同时开始施工。

　　作为项目负责人，你获得了候选隧道的信息，现在你可以按自己的想法选择一部分隧道进行施工，请问修建整条地铁最少需要多少天。

输入格式

　　输入的第一行包含两个整数n, m，用一个空格分隔，分别表示交通枢纽的数量和候选隧道的数量。

第2行到第m+1行，每行包含三个整数a, b, c，表示枢纽a和枢纽b之间可以修建一条隧道，需要的时间为c天。

* + 1. 基本要求

输出格式

　　输出一个整数，修建整条地铁线路最少需要的天数。

样例输入

6 6

1 2 4

2 3 4

3 6 7

1 4 2

4 5 5

5 6 6

样例输出

6

样例说明

　　可以修建的线路有两种。

　　第一种经过的枢纽依次为1, 2, 3, 6，所需要的时间分别是4, 4, 7，则整条地铁线需要7天修完；

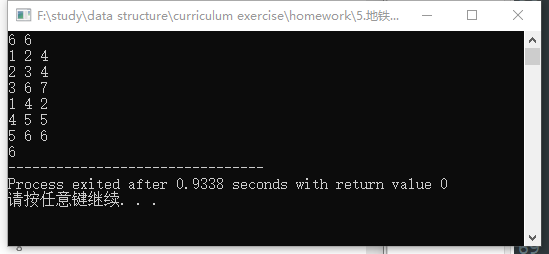
　　第二种经过的枢纽依次为1, 4, 5, 6，所需要的时间分别是2, 5, 6，则整条地铁线需要6天修完。

第二种方案所用的天数更少。

* + 1. 采用的数据结构：无向图、并查集
    2. 算法设计思想

先读入所有的边，并按权值从小到大排序。之后按权从小到大归并与边相连的两个顶点，并通过并查集检查顶点是否相连，当顶点1与顶点N连通时，当前边的权值即为最少的施工天数中最大的天数。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O(n)

* + 1. 其他方法

本题有多种解法，还可以采用dijkstra算法的思想求解。

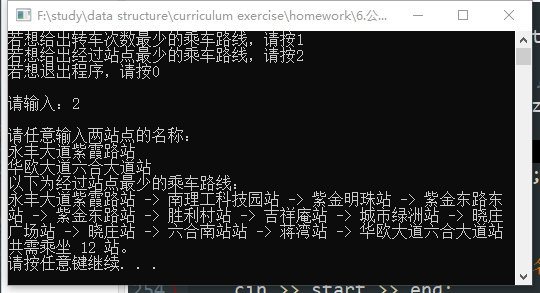
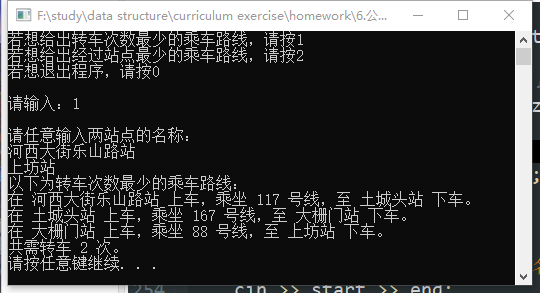
* 1. 公交线路提示
     1. 问题描述

上网下载真实南京公交线路图，建立南京主要公交线路图的存储结构。

* + 1. 基本要求
* 输入任意两站点，给出转车次数最少的乘车路线。
* 输入任意两站点，给出经过站点最少的乘车路线。
  + 1. 采用的数据结构：无向图
    2. 算法设计思想

求无权图的最短路径采用BFS的思想。求转乘次数最少的乘车路线时，将每路列车看作顶点，有公共站点的路线看作有边相连，然后再BFS。求经过站点最少的乘车路线时直接BFS即可。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O()

* + 1. 其他方法

也可以利用邻接矩阵存储该图，比该方法更易于理解。

* 1. 平衡二叉树操作演示
     1. 问题描述

利用平衡二叉树实现一个动态查找表。

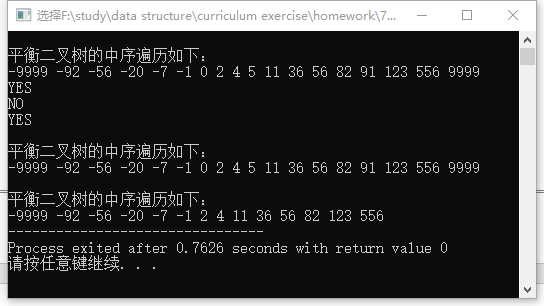
* + 1. 基本要求
* 从文件读取数据
* 实现动态查找表的三种基本功能：查找、插入和删除。
* 以可验证的方式输出结果
  + 1. 采用的数据结构：平衡二叉排序树
    2. 算法设计思想

插入：

1. 寻找数据插入位置。若小于当前根结点，将其递归插入至左子树，若大于，将其递归插入至右子树，若等于则插入。
2. 向父结点回溯。若当前二叉树不平衡，回溯至最小子树并调整其至平衡。

删除：

1. 寻找应当删除的结点。若小于当前根结点，则在左子树中寻找，若大于，将在右子树中寻找，直至找到，执行步骤2。
2. 若该结点为叶结点，执行步骤4。若非叶节点，执行步骤3。
3. 若其左子树比右子树高，将该结点数据与左子树最大结点的数据交换，再删除左子树的最大结点。若右子树比左子树高同理。
4. 删除该叶结点。
5. 向父结点回溯。若当前二叉树不平衡，回溯至最小子树并调整其至平衡。
   * 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O()

* + 1. 改进方法

将递归函数改为非递归函数，进一步加深理解。

* 1. 排序算法的比较
     1. 问题描述

利用随机函数产生10个样本，每个样本有50000个随机整数（并使第一个样本是正序，第二个样本是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。

* + 1. 基本要求
* 原始数据存在文件中，用相同样本对不同算法进行测试；
* 屏幕显示每种排序算法对不同样本所花的时间；
  + 1. 采用的数据结构：顺序表
    2. 算法设计思想

希尔排序：

事先选择一个合适的增量序列，对于每个子序列进行直接插入排序。

冒泡排序：

1. 第i趟冒泡从前往后遍历数组，若前一元素大于后一元素则交换两元素，最终使得[0,n-1)中最大元素放入n-i的位置上。
2. 重复上述操作n-1次。

快速排序：

每次将主元放在数列中间，递归调用函数，分而治之。

选择排序：

1. 选出无序序列最大值
2. 将无序序列最大值与无序序列最后位置元素互换
3. 重复上述操作n-1次。

堆排序：

1. 建堆

可以理解为递归思想，要想把root调整为堆，要先把root的左右子树都调整为堆，再把root向下过滤，直至找到合适的位置为止。

1. 排序

每次将使大顶堆出队一个元素，将其与堆最后一个元素互换，使得堆末尾后为有序序列，再将root调整为堆，重复上述操作n-1次即可。

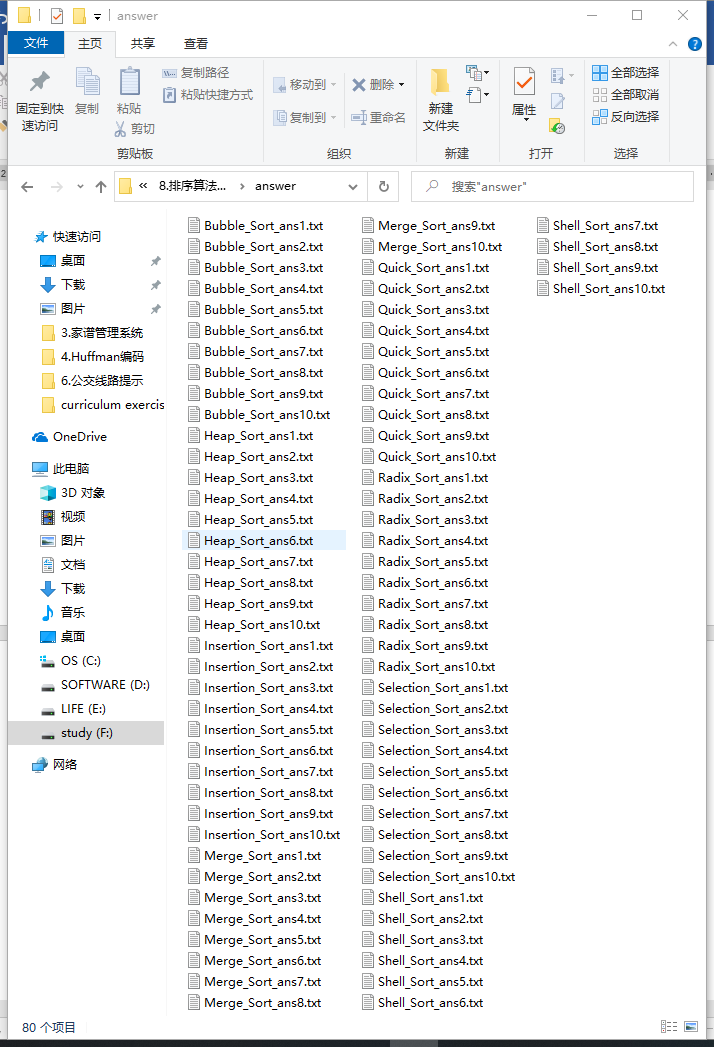
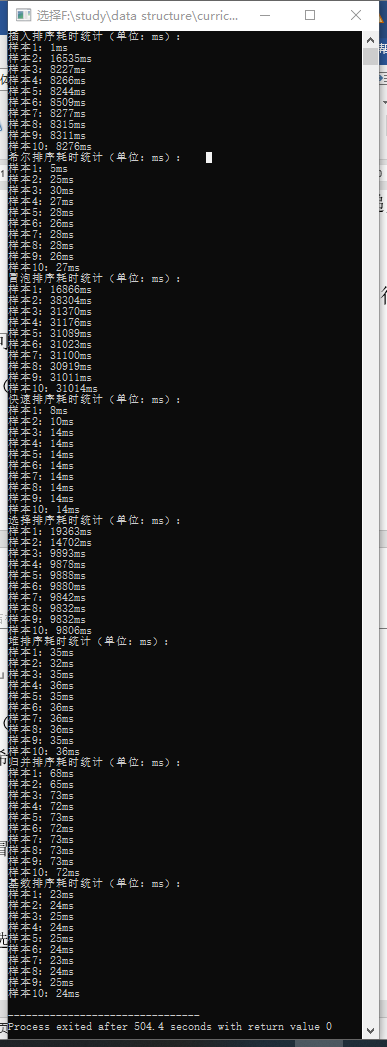
归并排序：

先将[left,center]和[center+1,right]归并为有序子列，再将[left,right] 归并为有序子列，递归调用以上函数。

基数排序：

事先已知待排数据的位数为n，再进行n次收集与分配即可。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

希尔排序：

T(n) = O()（Sedgewick增量序列）

S(n) = O(1)

冒泡排序：

T(n) = O()

S(n) = O(1)

选择排序：

T(n) = O()

S(n) = O(1)

归并排序：

T(n) = O()

S(n) = O(n)

快速排序：

T(n) = O()

S(n) = O()

堆排序：

T(n) = O()

S(n) = O(1)

基数排序：

T(n) = O(n)

S(n) = O(n)

* + 1. 改进方法

可继续练习其他排序算法的编程如：桶排序、表排序、计数排序等。

# 选做题

* 1. 数字排序【1分】
     1. 问题描述

给定n个整数，请统计出每个整数出现的次数，按出现次数从多到少的顺序输出。

* + 1. 基本要求

输入格式

　　输入的第一行包含一个整数n，表示给定数字的个数。

　　第二行包含n个整数，相邻的整数之间用一个空格分隔，表示所给定的整数。

输出格式

输出多行，每行包含两个整数，分别表示一个给定的整数和它出现的次数。按出现次数递减的顺序输出。如果两个整数出现的次数一样多，则先输出值较小的，然后输出值较大的。

样例输入

12

5 2 3 3 1 3 4 2 5 2 3 5

样例输出

3 4

2 3

5 3

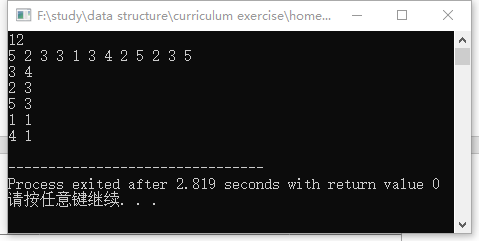
1 1

4 1

* + 1. 采用的数据结构：顺序表
    2. 算法设计思想

该题目可用一个数组，以下标作为数，数组内容存储该数出现次数来实现（这就相当于直接映射，是机试题目里面常用的一种解题法，很多看似非线性的题型最后其实都可以采取哈希或者映射的方法来巧解，体会哈希思想在机试题目中的巧用）。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O(n)

* 1. 社交网络图中结点的“重要性”计算【1】
     1. 问题描述

在社交网络中，个人或单位（结点）之间通过某些关系（边）联系起来。他们受到这些关系的影响，这种影响可以理解为网络中相互连接的结点之间蔓延的一种相互作用，可以增强也可以减弱。而结点根据其所处的位置不同，在网络中体现的重要性也不尽相同。

“紧密度中心性”是用来衡量一个结点达到其他结点的“快慢”的指标，即一个有较高中心性的结点比有较低中心性的结点能够更快地（平均意义下）到达网络中的其他结点，因而在该网络的传播过程中有更重要的价值。在有N个结点的网络中，结点vi的“紧密度中心性” 数学上定义为 到其余所有结点(j!=i)的最短距离的平均值的倒数：



对于非连通图，所有结点的紧密度中心性都是0。

本实验给定一个无权的无向图以及其中的一组结点，要求计算这组结点中每个结点的紧密度中心性。

* + 1. 基本要求

（1）输入说明：输入的第一行给出两个正整数N (N<=1000)和M，其中N是图中结点个数，结点编号从1到N； M（M<=10000）是边的数目。随后的M行中，每行给出一条边的信息，即该边连接的两个结点编号，中间用空格分隔。最后一行给出需要计算紧密度中心性的这组结点的个数K（K<=100），以及K个结点的编号，用空格分隔。

（2）输出说明：按照“Cc(i)=x.xx”的格式输出K个给定结点的紧密度中心性，每个输出占一行，结果精确到小数点后2位。

（3）测试用例：

输入 5 8

1 2

1 3

1 4

2 3

3 4

4 5

2 5

3 5

2 4 3

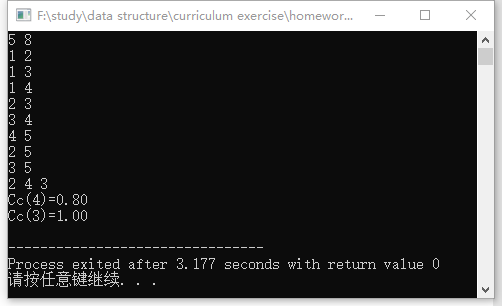
输出 Cc(4)=0.80

Cc(3)=1.00

* + 1. 采用的数据结构：无向图
    2. 算法设计思想

核心为求无权图的单源最短路径，方法依旧采用BFS，然后重复n次即可。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O()

S(n) = O()

* 1. 魔法优惠券【1】
     1. 问题描述

在火星上有个魔法商店，通过魔法优惠券。每个优惠券上印有一个整数面值K，表示若你在购买某商品使用这张优惠券，可以得到K倍该商品价值的回报。该商店还免费赠送一些有价值的商品，但如果你在领取免费赠品的时候使用面值为正的优惠券，则必须倒贴给商品K倍该商品价值的金额……但是不要紧，还有面值为负的优惠券可以用。

例如，给定一组优惠券，面值分别为1、2、4、-1；对应一组商品，价值为火星币7、6、-2、-3，其中负的价值表示该商品是免费赠品。我们可以将优惠券3（面值4）用在商品1（价值7）上，得到火星币28的回报。优惠券4（面值-1）用在商品4（价值-3）上，得到火星币3的回报。但是，如果一不小心把优惠券3（面值4）用到商品4（价值-3）上，你必须倒贴给商店火星币12个。同样，把优惠券4（面值-1）用到商品1（价值7）上，你必须倒贴给商店火星币7个。

规定每张优惠券和每件商品都只能最多被使用一次，求你可以得到的最大回报。

* + 1. 基本要求

输入说明：输入有两行。第一行首先给出优惠券的个数N，随后给出N个优惠券的整数面值。 第二行首先给出商品的个数M，随后给出M个商品的整数价值。N和M在[1,106]之间，所有的数据大小不超过230，数字间以空格分隔。

（2）输出说明：输出可以得到的最大回报。

（3）测试用例：

输入 4 1 2 4 -1

4 7 6 -2 -3

输出 43

输入：4 3 2 6 1

3 2 6 3

输出：49

输入： 7 3 36 -1 73 2 3 6

6 -1 -1 -1 -1 -1 -1

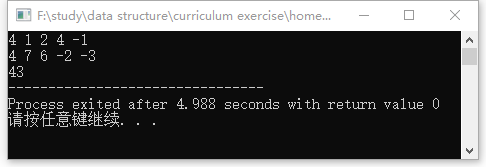
输出： 1

* + 1. 采用的数据结构：顺序表
    2. 算法设计思想

思路：1.最大正数对应相乘再相加，最小负数对应相乘再相加

2.正负数不交叉相乘

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O(n)

S(n) = O(n)

* 1. 点击窗口【1】
     1. 问题描述

在某图形操作系统中,有 N 个窗口,每个窗口都是一个两边与坐标轴分别平行的矩形区域。窗口的边界上的点也属于该窗口。窗口之间有层次的区别,在多于一个窗口重叠的区域里,只会显示位于顶层的窗口里的内容。  
　　当你点击屏幕上一个点的时候,你就选择了处于被点击位置的最顶层窗口,并且这个窗口就会被移到所有窗口的最顶层,而剩余的窗口的层次顺序不变。如果你点击的位置不属于任何窗口,则系统会忽略你这次点击。  
　　现在我们希望你写一个程序模拟点击窗口的过程。

* + 1. 基本要求

输入格式

输入的第一行有两个正整数,即 N 和 M。(1 ≤ N ≤ 10,1 ≤ M ≤ 10)  
　　接下来 N 行按照从最下层到最顶层的顺序给出 N 个窗口的位置。 每行包含四个非负整数 x1, y1, x2, y2,表示该窗口的一对顶点坐标分别为 (x1, y1) 和 (x2, y2)。保证 x1 < x2, y1 < y2。  
　　接下来 M 行每行包含两个非负整数 x, y,表示一次鼠标点击的坐标。  
　　题目中涉及到的所有点和矩形的顶点的 x, y 坐标分别不超过 2559 和　　1439。

**问题分析：**这个问题可以用链式线性表来实现。

输出格式：输出包括 M 行,每一行表示一次鼠标点击的结果。如果该次鼠标点击选择了一个窗口,则输出这个窗口的编号(窗口按照输入中的顺序从 1 编号到 N);如果没有,则输出"IGNORED"(不含双引号)。

样例输入

3 4  
0 0 4 4  
1 1 5 5  
2 2 6 6  
1 1  
0 0  
4 4  
0 5

样例输出

2  
1  
1  
IGNORED

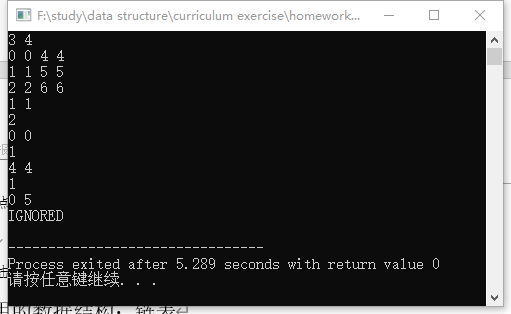
样例说明

第一次点击的位置同时属于第 1 和第 2 个窗口,但是由于第 2 个窗口在上面,它被选择并且被置于顶层。  
　　第二次点击的位置只属于第 1 个窗口,因此该次点击选择了此窗口并将其置于顶层。现在的三个窗口的层次关系与初始状态恰好相反了。  
　　第三次点击的位置同时属于三个窗口的范围,但是由于现在第 1 个窗口处于顶层,它被选择。  
　　最后点击的 (0, 5) 不属于任何窗口。

* + 1. 采用的数据结构：链表
    2. 算法设计思想

用链表存储n个窗口，若本次点击击中窗口，将该窗口从链表中删除并插入到表头，并输出窗口编号，若未击中输出IGNORED。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O(n)

S(n) = O(n)

* 1. 消除类游戏【2】
     1. 问题描述

消除类游戏是深受大众欢迎的一种游戏，游戏在一个包含有n行m列的游戏棋盘上进行，棋盘的每一行每一列的方格上放着一个有颜色的棋子，当一行或一列上有连续三个或更多的相同颜色的棋子时，这些棋子都被消除。当有多处可以被消除时，这些地方的棋子将同时被消除。

　　现在给你一个n行m列的棋盘，棋盘中的每一个方格上有一个棋子，请给出经过一次消除后的棋盘。

　　请注意：一个棋子可能在某一行和某一列同时被消除。

输入格式

　　输入第一行包含两个整数n, m，用空格分隔，分别表示棋盘的行数和列数。

　　接下来n行，每行m个整数，用空格分隔，分别表示每一个方格中的棋子的颜色。颜色使用1至9编号。

输出格式

　　输出n行，每行m个整数，相邻的整数之间使用一个空格分隔，表示经过一次消除后的棋盘。如果一个方格中的棋子被消除，则对应的方格输出0，否则输出棋子的颜色编号。

* + 1. 基本要求

样例输入

4 5

2 2 3 1 2

3 4 5 1 4

2 3 2 1 3

2 2 2 4 4

样例输出

2 2 3 0 2

3 4 5 0 4

2 3 2 0 3

0 0 0 4 4

样例说明

　　棋盘中第4列的1和第4行的2可以被消除，其他的方格中的棋子均保留。

样例输入

4 5

2 2 3 1 2

3 1 1 1 1

2 3 2 1 3

2 2 3 3 3

样例输出

2 2 3 0 2

3 0 0 0 0

2 3 2 0 3

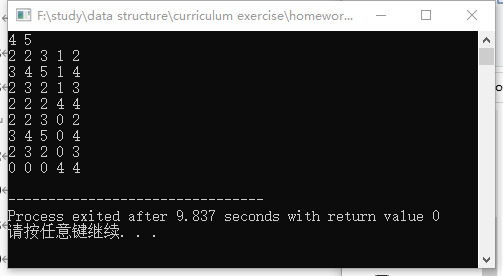
2 2 0 0 0

* + 1. 采用的数据结构：二维数组
    2. 算法设计思想

棋盘中所有的1以及最后一行的3可以被同时消除，其他的方格中的棋子均保留。

问题分析：这个问题与树无关，可以使用二维数组来存储，通过一遍遍历对符合条件的格子进行标记，然后第二遍遍历时消除符合条件的格子。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

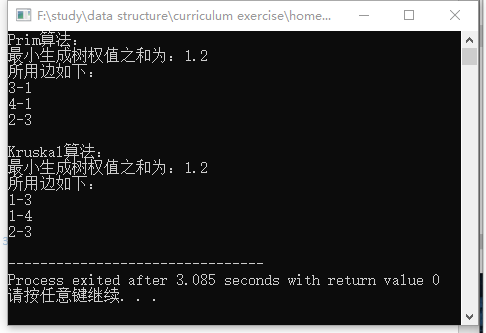
T(n) = O()

S(n) = O(1)

* 1. 最小生成树【3】
     1. 问题描述

利用普利姆算法和克鲁斯卡尔算法实现最小生成树问题。

* + 1. 基本要求
* 自行建立图的数据文件，第一行是顶点个数，然后依次是顶点名，接下来是边，用float表示边的权值；
* 以邻接表或者邻接矩阵表示图皆可；
* 分别利用Prim和Kruskal算法实现最小生成树；
* 输出最小生成树的权值之和，及所用的边。
  + 1. 采用的数据结构：无向有权图、并查集
    2. 算法设计思想
* Prim算法：利用dist[]数组存放结点到生成树的距离，并实时统计当前树的边数，每次找一个不在树上且dist值最小的结点，若找到该v，将对应的边收录入生成树上，若未找到，且当前收录的边数小于n-1，那么最小生成树不存在，若等于n-1，那么输出最小生成树权值之和以及所有边。
* Kruskal算法：先将所有的边按照权值从小到大排序，若不构成回路，每次将一个权值最小的边收录到生成树上，并利用并查集归并两棵子树。
  + 1. 测试数据及运行结果



* + 1. 算法复杂度
* Prim算法：

T(n) = O()

S(n) = O(n)

* Kruskal算法：

T(n) = O()

S(n) = O(n)

* + 1. 改进方法

利用最小堆实现Kruskal算法，降低时间复杂度。

* 1. 电子小词典【3】
     1. 问题描述

利用键树结构，建立一个微型电子字典。

* + 1. 基本要求

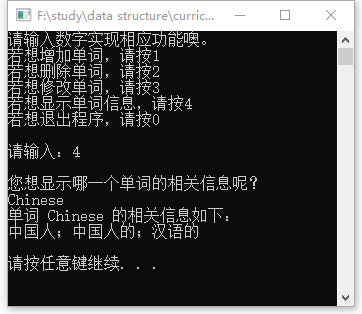
实现生词的加入，单词的查找、删除，修改等操作。

* + 1. 采用的数据结构：键树
    2. 算法设计思想

 假设字符的种数有m个，有若干个长度为n的字符串构成了一个Trie树，则每个节点的出度为m（即每个节点的可能子节点数量为m），Trie树的高度为n。很明显我们浪费了大量的空间来存储字符，此时Trie树的最坏空间复杂度为O(m^n)。也正由于每个节点的出度为m，所以我们能够沿着树的一个个分支高效的向下逐个字符的查询，而不是遍历所有的字符串来查询，此时Trie树的最坏时间复杂度为O(n)。这正是空间换时间的体现，也是利用公共前缀降低查询时间开销的体现。

每个结点里存放关键字与对应信息，再利用动态数组存放每个结点的子树，查找时逐个字符进行匹配，若匹配结束且当前结点中存有信息则匹配成功，反之失败。增添单词时先查找单词前半部分，该部分不需插入，进而将单词后半部分逐次增添结点。删除时，先查找到单词对应的结点，然后找到从当前结点到根的路径中，将第一个没有分支结点的结点，将该结点及其对应子树删除即可。

* + 1. 测试数据及运行结果



* + 1. 算法复杂度

T(n) = O(n)

S(n) = O(n)

* + 1. 改进方法

压缩结点以提高查找效率。

分支压缩：对于稳定的Trie树，基本上都是查找和读取操作，完全可以把一些分支进行压缩。例如，前图中最右侧分支的inn可以直接压缩成一个节点“inn”，而不需要作为一棵常规的子树存在。Radix树就是根据这个原理来解决Trie树过深问题的。  
节点映射表：这种方式也是在Trie树的节点可能已经几乎完全确定的情况下采用的，针对Trie树中节点的每一个状态，如果状态总数重复很多的话，通过一个元素为数字的多维数组（比如Triple Array Trie）来表示，这样存储Trie树本身的空间开销会小一些，虽说引入了一张额外的映射表。节点压缩。

# 三．**代码行数统计**

|  |  |
| --- | --- |
| 题目 | 代码行数 |
| 必做1 | 31+45+298+43+157=574 |
| 必做2 | 11+66+168+26+80=351 |
| 必做3 | 36+75+510=621 |
| 必做4 | 24+49+291+21+32=417 |
| 必做5 | 60 |
| 必做6 | 32+40+346=418 |
| 必做7 | 56+51+306=413 |
| 必做8 | 41+31+597=669 |
| 选做9 | 44 |
| 选做10 | 103 |
| 选做11 | 54 |
| 选做12 | 88 |
| 选做13 | 111 |
| 选做21 | 16+43+151=210 |
| 选做22 | 31+47+222=300 |
| 总计 | 4433 |

# 四．**结束语**

总体来说，本次课设难度较大，任务量也很大，但完成之后也收获满满。

我总共完成了八个必做题，七个选做题，这些选做题分别为9题、10题、11题、12题、13题、21题、22题，选做题分值总计12分，并且充分完成了每个题目及其要求的所有功能。

开始时，感觉必做题难度较大，因为每个题都会或多或少涉及一些课外的操作或技巧，比如第一题中如何获取系统进程、第四题中二进制文件按位读写、第五题中并查集的实现、第七题中平衡二叉排序树的删除操作等等，但通过与同学交流，探讨方法，以及在CSDN上参考其他博主的方法，这些问题最终都一一解决。

这次课设充分巩固了我在理论课上的所学知识，并得以实践。相比上机题，课设的很多题都更具有一定的实际意义，比如Huffman编码这道题，若真正实现将其按位存储，能够将文件压缩存储，又比如公交线路提示，利用真实南京公交线路图建立图的存储结构，并给出提示，充分说明了图结构的广泛应用价值。

这次课设也拓宽了我的知识面，学习到了很多实用的操作与技巧，在解决一个个问题的过程中也极大地锻炼了我的自学能力与知识迁移能力。

同时做题过程中当然免不了会有各种奇奇怪怪的bug，在给自己以及给同学调代码的过程中，自己的debug能力也有了进一步的提升。

总之这次课设虽然过程曲折艰辛，但完成之后还是很有成就感的。