**武汉大学计算机学院**

**本科生实验报告**

**数据结构与算法实验设计与实现**

专 业 名 称 ：软件工程

课 程 名 称 ：数据结构与算法实验

指 导 教 师 一：xxx

学 生 学 号 ：xxx

学 生 姓 名 ：xxx

二○一八年十二月

**郑 重 声 明**

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 日期：

摘 要

数据结构与算法的实验选取典型题目，涵盖线性表、堆栈、迷宫、哈夫曼树/编码、树的遍历、排序等为主要知识点的六个题目，目的是进一步巩固知识点，锻炼、提高学生的编码能力，能完成从理论到实践的转变。

实验内容主要包括：单个与多个线性表的排序，判断字符串左右括号是否相等，用两种方法求解迷宫问题，构造哈夫曼树和哈夫曼编码，二叉树先序遍历的递归算法与非递归算法，以及排序的多种算法。

最终程序能正常运行，且代码精简，效率较高。

**关键词：**数据结构；算法；实验

目 录

[1 实验目的和意义](#_Toc534210751)

[1.1 实验目的 5](#_Toc534210752)

[1.2 实验意义 5](#_Toc534210753)

[2 实验设计](#_Toc534210754)

[2.1 概述 5](#_Toc534210755)

[2.2 实验原理 6](#_Toc534210756)

[2.2.1 线性表 6](#_Toc534210757)

[2.2.2 堆栈 6](#_Toc534210758)

[2.2.3 迷宫问题 6](#_Toc534210759)

[2.2.4 哈夫曼树、哈夫曼编码 7](#_Toc534210760)

[2.2.5 二叉树的先序遍历 8](#_Toc534210761)

[2.2.6 排序问题 8](#_Toc534210762)

[2.3 实验方案 9](#_Toc534210763)

[2.3.1 线性表 10](#_Toc534210764)

[2.3.2 堆栈 10](#_Toc534210765)

[2.3.3 迷宫 11](#_Toc534210766)

[2.3.4 哈夫曼树、哈夫曼编码 13](#_Toc534210767)

[2.3.5 二叉树的先序遍历 14](#_Toc534210768)

[2.3.6 排序问题 16](#_Toc534210769)

[3 运行调试与结果分析](#_Toc534210770)

[3.1 空指针异常 19](#_Toc534210771)

[3.2 运行结果与预期不一致 19](#_Toc534210772)

[3.3 排序性能分析 19](#_Toc534210773)

[4 设计体会与小结](#_Toc534210774)

# 1 实验目的和意义

## 1.1 实验目的

本实验通过六个小实验，从不同的知识点出发，使学生对线性表、堆栈、队列、树、排序等相关算法有一个更深的认识，理论结合实际，锻炼学生的代码能力，对数据结构更加熟悉，掌握度更高。

## 1.2 实验意义

理论知识固然重要，但是仅有理论知识并不够。作为软件专业的学生，更需要动手写代码的能力，需要掌握数据结构的基本知识，对基本的概念，如线性表、堆栈、队列、树等需要有熟练的掌握程度。而选取典型的实验，就是为了在实践过程中让学生更熟练的掌握基本知识，学以致用。

# 2 实验设计

## 2.1 概述

第一个题目是有关线性表的算法，根据两个数组A, B建立两个飞递件有序单向链表，然后合并成一个非递减链表；第二个题目是堆栈，判断输入的字符串是否匹配；第三个题目是迷宫问题，用栈方法、队列方法、图搜索方法和递归回溯方法中的任意两种求解；第四个题是给定序列，构造哈夫曼树，输出哈夫曼编码；第五个题目是对采用二叉链表存储的二叉树实现先序遍历的递归算法和非递归算法；第六个题目是至少采用三种不同类型的排序算法，在排序数为100时输出结果，为10,000、100,000时输出排序时间，比较性能。

## 2.2 实验原理

### 2.2.1 线性表

第一个题目，首先建立线性表，这里采取链式存储结构。然后对其进行递增的排序。首先创建结点 p 指向头结点指针 L的第二个结点，构造一个只含有一个数据结点的有序单链表，且用结点 q 来保存 p 后面的结点。然后，将pre指向前驱结点。如果第一个结点（pre->next）的值比 p 的小，那么将 pre 指向下一个结点的前驱结点，即 pre = pre->next; 并且，将 pre 指向的结点之后插入 p 结点，即 pre->next = p->next; pre->next = p; 最后，将 p 后移，即 p = q; 扫描余下的结点，直到链表为空。

合并两链表时，分三种情况，第一种，两个有序单链表 A, B 的指针 pa, pb 都不为空，这时再分两种情况依次比较两链表的结点的值，将若 A 的结点 pa 的值比 B 的 pb 所指向的值小，则将这个值存入新的链表 C 的结点 pc 中，并将 pa, pc 指针后移，依次循环；第二种，pa 为空，则将 pb 的值依次赋给 pc，因为是有序单链表，所以无需排序；第三种，pb 为空，同理第二种。这样，链表 C 就是合并后的有序递增单链表。

### 2.2.2 堆栈

第二题，判断左右括号是否匹配。原理为栈。依次扫描表达式的字符，分为两种情况，第一种，如果是左括号‘(’，则将其进栈，继续扫描；第二种，如果是右括号‘)’，则需得到栈顶元素，根据“先进后出”原则，如果栈顶元素是‘(’，则出栈，继续扫描，反之，则不匹配。此外，需要记录左括号‘(’和右括号‘)’的数量，在扫描完后，栈如果不为空或者两者数量不相等，则都说明不匹配。

### 2.2.3 迷宫问题

第三题，迷宫问题。这里采用栈和队列两种方式。建立二维数组，用“0”表示可行，用“1”表示不可走。

①第一种，栈，深度优先。将起始位置 (xi, yi) 设置为 -1, 并将其进栈，表示已经走过，避免重复走到该方块。从此时开始，栈不为空的情况下，取栈顶元素，判断其是否是出口坐标。若是，则将其所有元素出栈并存入一个一维数组中，再将一维数组元素输出。这是根据栈“先进后出”的原理，使得输出的顺序是从起始方块到出口方块的路径。如果此栈顶元素不是出口，则设置寻找路径的顺序，假设依次从右、下、左、上来寻找路径，那么用 di (初始值为 -1，每次寻找可走方位时 + 1，因此 0 <= di <= 3) 来标记这个方位。找到一个可走方块后，将其进栈，且标记该点为 -1。依此循环，直到终点或没有可走方块。对于后者，找不到可走方块时，退栈，再取栈顶元素，寻找新的路径。这里需要注意的是，由于在进栈时已标记过 di 的值，这时再次寻找方位时 di 在之前的基础上增 1，因此避免了走相同路径的可能。按上述步骤循环，找到出口时则可以输出路径，找不到时则说明路径不存在。

②队列，广度优先。根据队列“先进先出”的原则，首先将起始坐标入队，同样地，将坐标标记为 -1。而这里不同的是，采用 pre 来记录上一个方块在队列中的下标，并将起始方块 pre 标记为 0。和前一种方法类似，也是在队列不为空的情况下，出队列元素，判断是否是终点，如果是，则输出路径；反之，寻找此出队元素四个方位中所有可走方块，依次进队，同时将队头指针赋给 pre，记录下前一个方块（由于进队方块是由于出队方块寻找的相邻可走的路径，那么方块出队后 队头指针后移，其实指向的就是出队方块，也就是当前进队方块的上一个路径），并将此点的值赋为 -1，避免重复进队。循环以上过程，直至找到出口。找到后，回溯 pre，找到其上一个方块，直至为起始方块。这样，路径就可以输出了。若不能输出，则说明无可通路径。

### 2.2.4 哈夫曼树、哈夫曼编码

首先需要构造哈夫曼树。其原理是，选取当前无双亲的结点中，权重最小的两个结点，分别作为左右结点，并将两个结点的权重之和作为其双亲结点。依此循环，直至构建完毕。在书写代码时，总结点数为叶子节点数 n0 的两倍减一（即 (2 \* n0 - 1)，n0 可为数组元素的个数）。并将所有结点的双亲结点、孩子结点赋初值为 -1。共有两层循环，第一层循环为构造双亲结点，范围为(n0 <= 双亲结点 < 2 \* n0 – 2, 共 (n0 – 1) 个)。第二层循环为找到权重最小的两个元素，将其作为左右结点。用 min1, min2 分别标记当前左右结点的权重，若下一个无双亲结点的权值比 min1 小，则将它作为左结点，其权值赋给 min1，并将前一个左结点变为右结点，min1 赋给 min2。同理，若结点权值比 min2 小，则将其作为右节点，并把权值赋给 min2。找到最小的两个结点（即第二层循环完成）后，再构造他们的双亲结点，其权值为两者之和，并分别给左右孩子结点、双亲结点赋值。接着进入下一次循环，直至哈夫曼树构建完毕。

哈夫曼编码时，从最开始的叶子结点（即初始给定数组）顺序依次遍历，在其有双亲结点的情况下，找到其双亲结点并判断此结点是左孩子还是右孩子。若是左孩子，则编码为“0”，若是右孩子，则编码为“1”。依此循环，一层一层找到双亲结点并判断左右孩子，分别赋“0”或“1”。值得注意的是，由于编码是逆向的，因而顺序是反的。这里采用 start 标记。初始时 start 为 n0，然后每增加一个编码，start 减一，这样就使得哈夫曼码里的顺序是正向的了，且哈夫曼码为 [start … n0]。

### 2.2.5 二叉树的先序遍历

第五题，给定一个字符串，假设为“A(B(D(, G)), C(E, F))”。首先创建二叉树。依次遍历字符，若为，则说明接下来的是左孩子结点，用 k 标记为 0，并将前一个结点进栈；若为‘,’，则将要处理右孩子，k = 1；若为‘)’，则说明已遍历完右孩子结点，出栈；其他情况，全为字符，则创建新结点，赋好初值后，根据 k 的值判断其是左孩子还是右孩子，并将栈顶结点的左孩子或右孩子指向此结点。需要注意的是，刚开始时，栈顶元素为空，需要将新建的结点（此情况为‘A’）作为根节点，并接着遍历下一个字符。

先序遍历的递归算法，先遍历根结点，再分别遍历左孩子结点、右孩子节点。

递归算法，用栈进行存储，在栈不空的情况下，先将根结点进栈。再出栈，根据“先进后出”的原则，若有右孩子，则先将右孩子进栈，若有左孩子，再将左孩子进栈。这样，出栈顺序即为先序遍历的顺序了。

### 2.2.6 排序问题

对数组 R[] 进行递增排序。

① 直接插入排序：取待排序的数组元素 R[i]，用 temp 记录当前 R[i]的值，一般从 i = 1 开始，依次与前面的元素进行比较，并将前面的元素从 i - 1依次往后挪一个，直到 R[i] 的值比待挪动的元素的值小。最后将 temp 放到此处。一直循环到数组元素为空为止。

② 折半插入排序：元素分为有序区 [0…(i - 1)] 和无序区 [i…(n - 1)]，选取 R[i]，将其与有序区中的值进行比较。比较方法为：取 low 初始值为 R[0] 的值，high 为 R[i-1]，mid 为 (high - low) / 2。若 R[i] 的值比 mid 大，则 low = mid；反之，high = mid，再进行 mid 与 R[i] 的比较。直到找到插入位置。找到后，与直接插入排序类似，有序区中插入位置之后的元素依次后挪，最后将此 R[i] 放到插入位置。进行下一轮循环。

③ 希尔排序：将数组元素分为 d 组，一般为 (总数n / 2)，每隔 d 个元素的元素为一组进行比较。比较方法与直接插入排序类似，差别在于，一组中的间隔不为 1，而是 d。依此循环，d = n / 2；直到 d = 0。

④ 冒泡排序：两层循环，外层循环从第一个元素到倒数第二个元素，内层循环为，从最后一个元素到比 i 大。从后往前，依次比较 R[i] 与 R[j] 的值，若 R[i] 大于 R[j]，则交换。

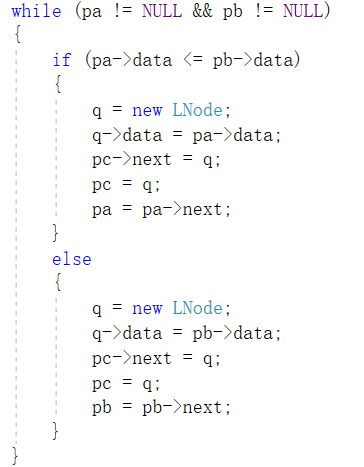
⑤ 快速排序：选择一个基准 i（一般为第一个），记为 temp，将其他元素依次与 temp 比较，区间为 s 到 t，从左到右扫描，找到比 temp 大的元素 R[s]，插入到 R[t] 处，将 s 加一；从右到左扫描，找到比 temp 小的元素，插入到 R[s] 处，将 t 减一；依次循环，直到 s 不小于 t。最后，在 R[s] 处插入 temp。此时元素被划分为两部分，左边的比 R[i] 小，右边的比 R[i] 大。再进行下一轮循环，左边和右边，分别选取基准，进行排序，直到 s 不小于 t。

⑥ 简单选择排序：将元素划分为有序区和无序区两部分，在无序区中找到最小的元素，将其与无序区第一个元素进行交换，这样，就可以排好序了。

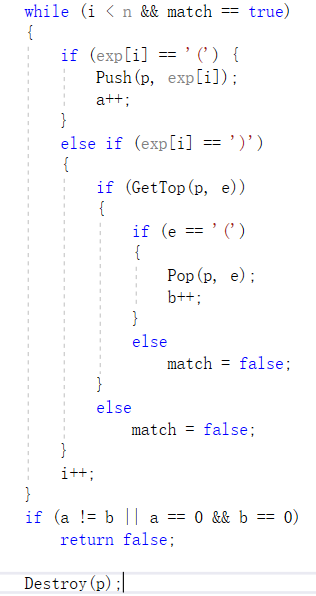
## 2.3 实验方案

以下是具体实验的核心代码：

### 2.3.1 线性表

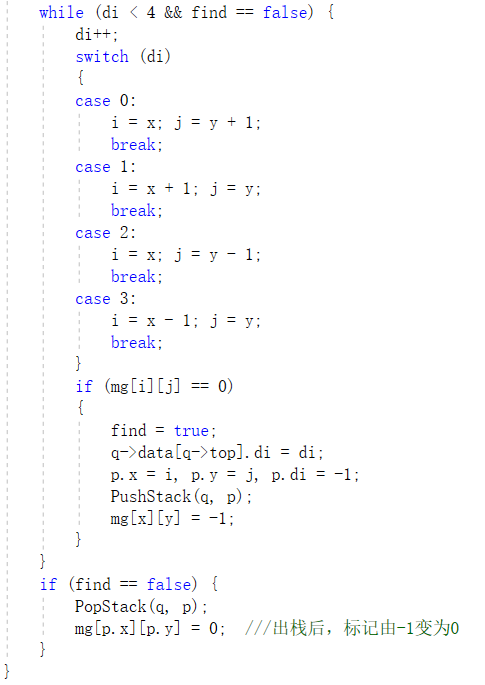
两有序表合并（两表元素都还不为空的情况）：

### 2.3.2 堆栈

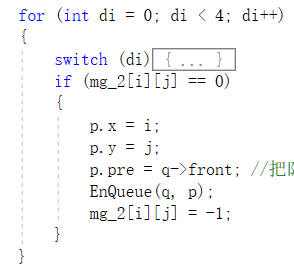


### 2.3.3 迷宫

① 用栈求解

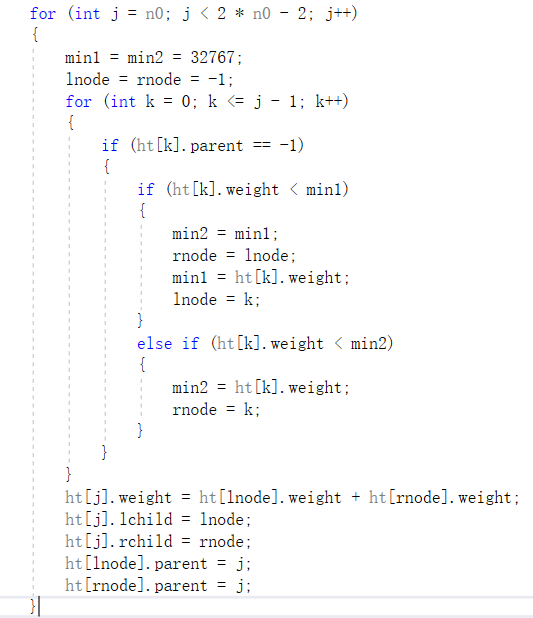


② 用队列求解

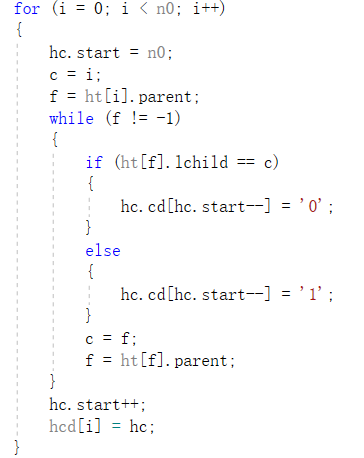


### 2.3.4 哈夫曼树、哈夫曼编码

构建二叉树：

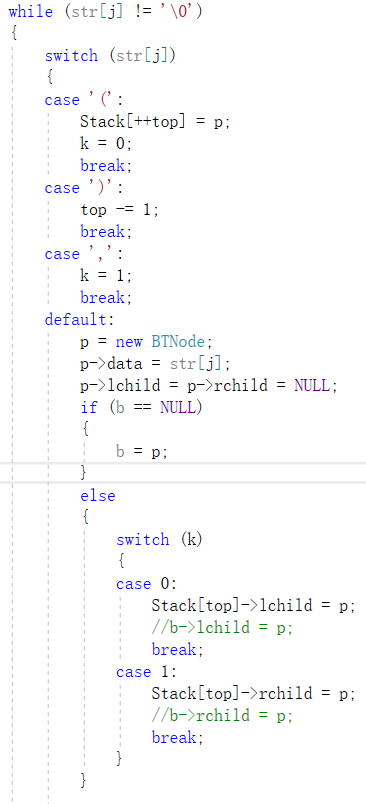


哈夫曼编码：

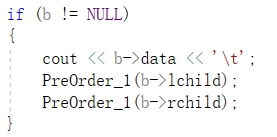


### 2.3.5 二叉树的先序遍历

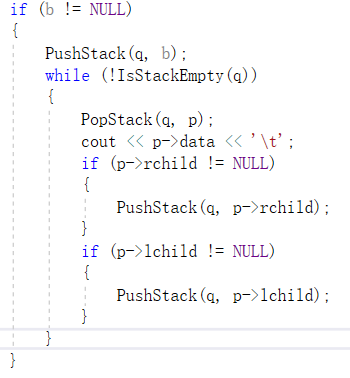
创建二叉树：



先序遍历的递归算法：



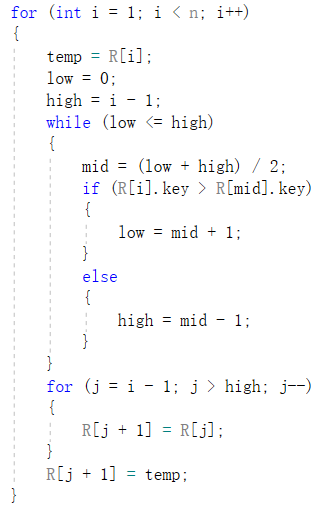
先序遍历的非递归算法：



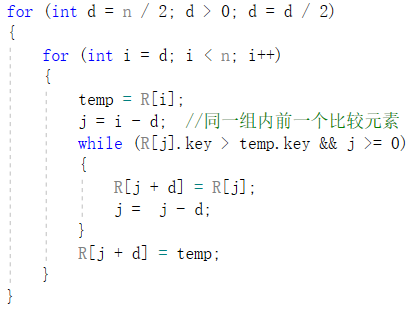
### 2.3.6 排序问题

① 直接插入排序

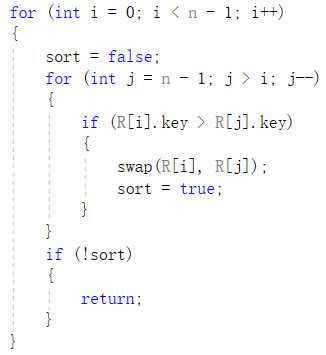
② 折半插入排序



③ 希尔排序

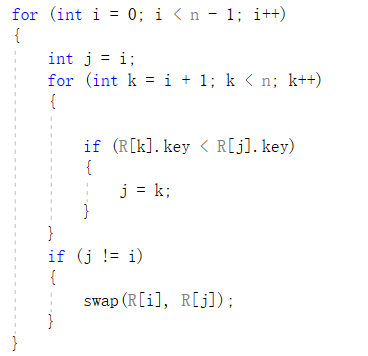


④ 冒泡排序



⑤ 快速排序

⑥ 简单选择排序



# 3 运行调试与结果分析

### 3.1 空指针异常

出此错误原因多是没赋初值，只定义了，但是没有 new 或者没给初值。在线性表、哈夫曼树和哈夫曼编码里都有遇到。

### 3.2 运行结果与预期不一致

比较常见了。多是出于逻辑错误，需要一行行加断点进行调试，在迷宫问题上格外突出。需要用纸笔跟着分析，判断出错的地方，再进行修改。

### 3.3 排序性能分析

此表为多次运行后的结果，‘/’表示两次不同运行结果之间的分隔（时间：ms）。

表3.3 不同排序算法在不同排序数据下的运行时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 排序算法 | 时间复杂度 | 10,000个 | 100,000个 | 1,000,000个 |
| 直接插入排序 | O(n²) | 78/94 | 7156/7328 | 777250 |
| 折半插入排序 | O(n²) | 47/62 | 6078/6172 |  |
| 希尔排序 | O(n^1.3) | 0 | 31/32 | 343 |
| 冒泡排序 | O(n²) | 0 | 265/328 | 4219 |
| 快速排序 | O(nlog2n) | 0/15/0 | 15/16/31 | 203 |
| 简单选择排序 | O(n²) | 141/125 | 10234/10328 | 1072938 |

# 4 设计体会与小结

六个题目很典型，做的途中出的问题也很多，主要还是对算法不熟悉，每开始做一个题目就需要翻阅书本，把这块的知识补上，研究算法，在纸上一点点画，懂了之后再开始写代码。调试也比较累，需要慢慢调。写这个报告时，也需要自己再次看一遍算法，达到能讲出来的效果后，再开始书写，也算是巩固了。

不过，好处总是大于坏处的，确实掌握了很多，以排序算法为例吧，以前用得最多的就是冒泡了，一想到排序就是冒泡，现在发现其实有很多别的算法更快、也更简洁，实用性较高。之前学离散数学时，有关图和树的概念已经接触过，但是只知道原理，不知道代码应如何书写，现在认真过了几遍后，也知道应该怎么做了。个人觉得代码得多练才能生巧，不然总是写的很慢，得想很久。也有一些体会吧，比如写同一类的，刚开始不会，写多了就越写越快，调试也很快，一下子就知道应该是哪儿做错了。也慢慢的发现了一些相通之处，算法都一样，只是代码有些差异。有时候写一个程序，调试的时候总是会忘掉时间，一下子就过了很长时间。数据结构也算是很重要的一门课吧，毕竟用的地方多，不局限，而且对程序性能上也有很大的影响，就像我现在在运行 1,000,000个数据的排序算法，已经好久了，还是没有出结果，我到底是等呢，还是不等呢。也慢慢开始注意到代码的整洁性，有关“一行不多，一行不少”的理念也慢慢开始在注意，也开始读《代码整洁之道》，有意识的让代码变得更整洁易懂。