

西安交通大学

数据结构与算法实验报告

专业班级：

姓名学号：

联系电话：

时 间：

一、约瑟夫环问题仿真

(1) 实验要求:

设编号为 1, 2, ..., n ($n > 0$) 个人按顺时针方向围坐一圈, 每人持有一个正整数密码。开始时任意给出一个报数上限 m , 从第一个人开始顺时针方向自 1 起顺序报数, 报到 m 时停止报数, 报 m 的人出列, 将他的密码作为新的 m 值, 从他在顺时针方向上的下一个人起重新自 1 报数; 如此下去直到所有人全部出列为止。

(2) 程序思想:

构造循环链表, 插入的最后一个元素的指针指向头结点

`void Jo(int m)` 循环 pass 后输出该元素, 将上一个元素的指针指向下一个元素, 然后删除该结点, 直到只剩最后一个结点。

(3) 输入输出数据:

```
Microsoft Visual Studio 调试控制台
请输入初始密码值m和人数n
20
7
请输入第1人的密码
3
请输入第2人的密码
1
请输入第3人的密码
7
请输入第4人的密码
2
请输入第5人的密码
4
请输入第6人的密码
3
请输入第7人的密码
4

#####
1的密码是 3
2的密码是 1
3的密码是 7
4的密码是 2
5的密码是 4
6的密码是 3
7的密码是 4

#####
第1个出局的人是6
第2个出局的人是1
第3个出局的人是4
第4个出局的人是7
第5个出局的人是2
第6个出局的人是3
第7个出局的人是5
```

(4) 简单评价：

算法实现比较简单，插入结点的时间复杂度为 $O(n)$, 每一次淘汰计算时间复杂度为 $O(1)$ ，不需要额外空间。

二、 二叉排序树与平衡二叉树的实现

(1) 实验要求：

(1) 分别用二叉链表和顺序表作存储结构实现二叉排序树。

1) 以回车符(‘\n’)为输入结束标志，输入数列 L，生成一棵二叉排序树 T；

- 2)对二叉排序树 T 作中序遍历, 输出结果;
- 3)计算二叉排序树 T 查找成功的平均查找长度, 输出结果;
- 4)输入元素 x, 查找二叉排序树 T, 若存在含 x 的结点, 则删除该结点, 并作中序遍历(执行操作 2); 否则, 输出信息 “无 x” ;

(2)用二叉链表作存储结构实现平衡的二叉排序树。

- 1)用数列 L, 生成平衡的二叉排序树 BT: 当插入新元素之后, 发现当前的二叉排序树 BT 不是平衡的二叉排序树, 则立即将它转换成新的平衡的二叉排序树 BT;
- 2)计算平衡的二叉排序树 BT 的平均查找长度, 输出结果。

(2) 程序思想:

1. 建树, 如果是空树, 插入后为根结点

若不为空树, 元素等于则返回, 大于插入右子树, 小于则插入左子树

2. `void in_order_(node<T>* n)`中序遍历 中序遍历首先遍历左子树，然后访问根结点，最后遍历右子树。

3. `double average_length()`平均查找长度，通过堆栈前序遍历，推入左右子树时查找长度将会加一，计算访问所有节点的长度之和并除以结点个数，得到平均查找长度

4. 删除 叶子结点时直接删除即可
根结点，先在左子树上找最大结点，若找到，将其父节点的指针指向其左子树，将该结点替换为删除的结点。若左子树为空，则在右子树上找最小结点，若找到，将其父节点的指针指向其右子树，将该结点替换为删除的结点。

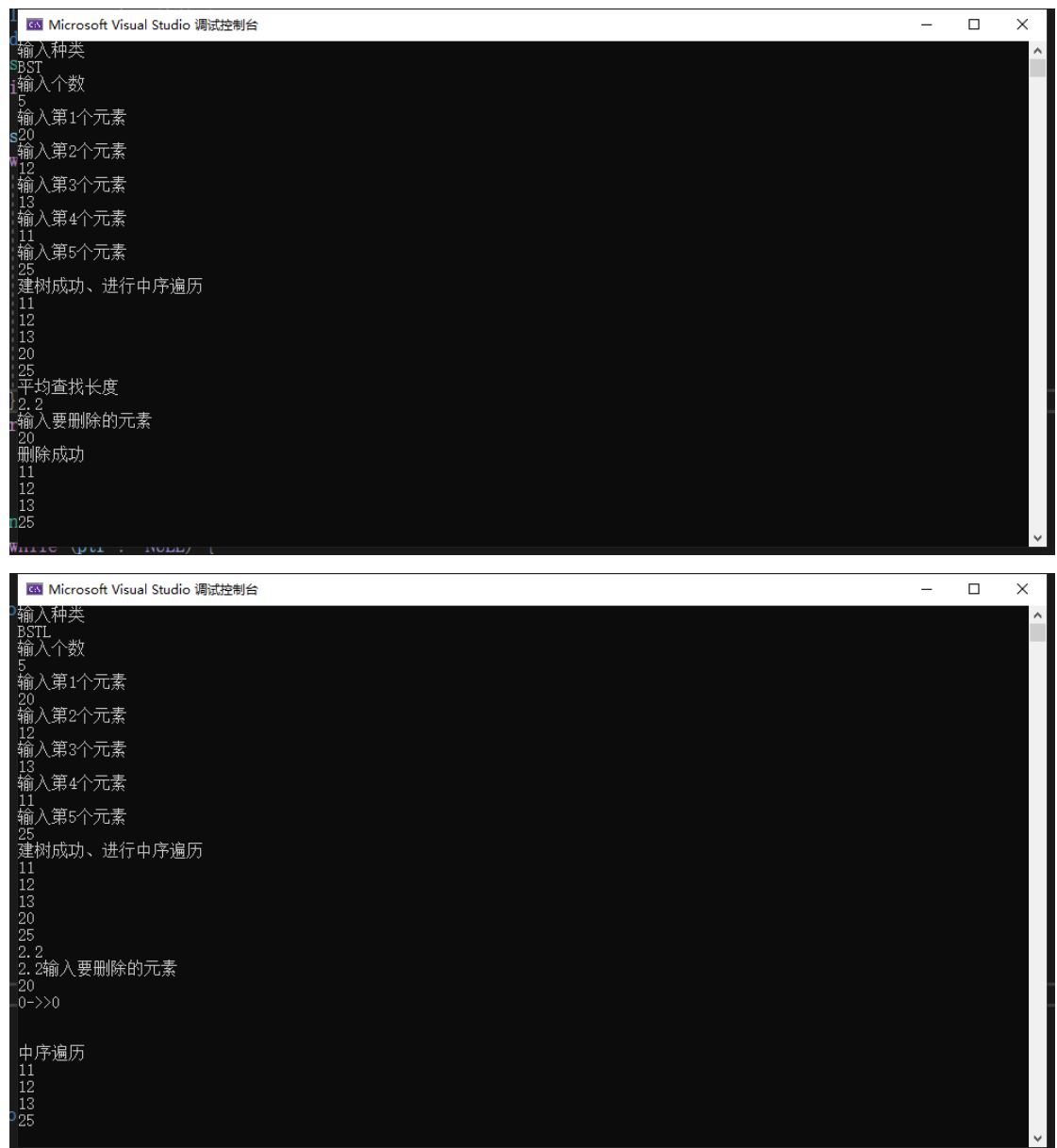
AVL 树平衡化旋转

插入结点后更新平衡因子，对树进行调整，新结点的平衡因子为0，统计插入后若失去平衡，则平衡二叉树。

旋转方式有左旋和右旋，不平衡共有 4 种情况，分别为 LL、RR、LR、RL，分别以右旋，左旋，左旋再右旋，右旋再左旋来进

行平衡化处理。

(3) 输入输出数据：



```
Microsoft Visual Studio 调试控制台
输入种类
BST
输入个数
5
输入第1个元素
20
输入第2个元素
12
输入第3个元素
13
输入第4个元素
11
输入第5个元素
25
建树成功、进行中序遍历
11
12
13
20
25
平均查找长度
2.2
输入要删除的元素
20
删除成功
11
12
13
25

Microsoft Visual Studio 调试控制台
输入种类
BSTL
输入个数
5
输入第1个元素
20
输入第2个元素
12
输入第3个元素
13
输入第4个元素
11
输入第5个元素
25
建树成功、进行中序遍历
11
12
13
20
25
2.2
2.2输入要删除的元素
20
0->>0

中序遍历
11
12
13
25
```

```
Microsoft Visual Studio 调试控制台
输入种类
AVL
输入个数
5
输入第1个元素
20
中序遍历
20
输入第2个元素
10
中序遍历
10
20
输入第3个元素
5
LL
中序遍历
5
10
20
输入第4个元素
15
中序遍历
5
10
15
20
输入第5个元素
11
RL
中序遍历
5
10
11
15
20
建树成功、进行中序遍历
平均查找长度
11 5
2.2
```

(4) 简单评价：

用二叉链表存储的二叉树结构，存储空间需求略大，删除操作只需改变指针，时间复杂度较小。

用数组存储的二叉树结构元素按物理顺序依次储存，在树形接近完全二叉树时存储空间较小。读取较为方便，但是删除时更新的元素较多

平衡化处理可以减少平均查找长度，提高查找效率

三、 迷宫问题

(1) 实验要求:

迷宫由 m 行 n 列的二维数组设置, 0 表示无障碍, 1 表示有障碍。

设入口为 $(1, 1)$, 出口为 (m, n) , 每次只能从一个无障碍单元移到周围四个方向上任一无障碍单元。编程实现对任意设定的迷宫, 求出一条从入口到出口的通路, 或得出没有通路的结论。

(2) 程序思想:

用 `getline()` 函数, 从文件中依次读取地图信息。存入二维数组中。

```
bool findway(int x, int y)
```

将当前格子标记为正在走的点, 若周围格子均为墙壁或不可达点则将自身标记为不可达点并返回 `false`, 若周围是终点或可达点则将自身也标记为可达点并返回 `true`, 递归调用不断试探, 直到起点四周全部被试探完。

从起点开始调用函数知道函数返回, 若返回值为真则说明可以到达

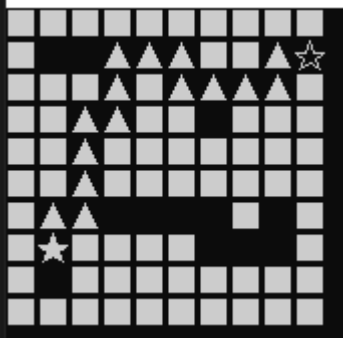
终点，输出到达路径。若返回值为假说明终点不可达，输出相关信息。

(3) 输入输出数据：

puzzle.txt - 记事本

文件(F) 编辑(E) 格式(O) 窗口(W) 帮助(H)

```
1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 1 0 7
1 1 1 0 1 0 0 0 0 1
1 1 0 0 1 1 0 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1 0 1
1 9 1 1 1 1 0 0 0 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```



```
1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 1 1 7
1 1 1 0 1 0 0 0 0 1
1 1 0 0 1 1 0 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1 0 1
1 9 1 1 1 1 0 0 0 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```

不可达

(4) 简单评价：

函数递归，空间复杂度较高。

程序设计比较简单，无法寻求最短路径

将函数栈中的格子设为不通过的点，避免了在小范围转圈的死循环

四、 哈夫曼压缩/解压缩算法

(1) 实验要求：

- 1) 通信内容可以是任意的多媒体文件；
- 2) 自己设定字符大小，统计该文件中不同字符的种类（字符集、个数）、出现频率（在该文件中）；
- 3) 构建相应的哈夫曼树，并给出个字符的哈夫曼编码；
- 4) 对源文件进行哈夫曼压缩编码形成新的压缩后文件（包括哈夫曼树）；
- 5) 编写解压缩文件对压缩后文件进行解码还原成源文件。

(2) 程序思想：

(1)压缩

1. 统计字符频率，通过下标索引，存入 char_num=[]

2. 建哈夫曼树

3. 编码，中序遍历，通过下标索引将每个字符的存入 huf_tab[]

4. 计算字符种类，最后一位的有效位数，将各个字符出现的频率

相加，求除以 16 的余数

5. 写文件头，包含垃圾位，字符种类，字符频率

6. 依次读取字符，根据 huf_tab[]中的编码，每 16 位以 int 写入

压缩文件，最后一位的

(2)解压

1. 读取文件头，统计字符频率，通过下标索引，存入 char_num

2. 建哈夫曼树

3. 通过操作树的指针，还原文件

当读入 1 时，树指针向左移动，读入 0 时，树指针像右移动。当指

针所指结点有字符时，输出字符并将指针置为根结点。

(3) 输入输出数据:

压缩		解压	
输入需要压缩的文件名2.txt		输入需要解压的文件名2	
输入压缩后的文件名2		输入解压后的文件名3.txt	
统计字符频率		统计字符频率	
2	2021/1/5 21:46	文件	15 KB
2.txt	2021/1/5 16:40	文本文档	26 KB
3.txt	2021/1/5 21:46	文本文档	26 KB

(4) 简单评价:

统计字符频率时和哈夫曼编码通过字符下标索引直接存入数组，时间复杂度为 $O(1)$

建哈夫曼树时使用了优先队列，空间复杂度较高。

压缩比例根据压缩文件字符频率而定，理论上讲，频率越接近压缩比率越小。比较适合压缩频率差距很大的文件。

算法还需进一步改进，例如暂时无法压缩字符全部相同的极端情况