

**函数式编程原理课程报告**

|  |  |
| --- | --- |
| 姓 名： | 刘文博 |
| 班 级： | 计算机本硕博2101班 |
| 学 号： | U202115666 |
| 指导教师： | 郑然 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2023 年 10 月 20 日

目 录

[一、 Heapify求解 1](#_Toc148682896)

[1.1 问题需求 1](#_Toc148682897)

[1.2 解题思路与代码 2](#_Toc148682898)

[1.3 遇到的问题及运行结果 3](#_Toc148682899)

[1.4 性能分析 5](#_Toc148682900)

[二、 课程总结和建议 6](#_Toc148682896)

# 一、 Heapify求解

## 问题需求

一棵minheap树定义为：

1. t is Empty;
2. t is a Node(L, x, R), where R, L are minheaps and values(L), value(R) >= x (value(T)函数用于获取树T的根节点的值）
3. 编写函数treecompare, SwapDown 和heapify：

**treecompare**函数：

* 描述：给定两棵树，该函数返回一个类型为**order**的值，该值基于哪棵树具有较大值的根节点。
* 参数：两个树作为输入。
* 返回值：类型为**order**的值，可以表示哪棵树的根节点值较大。
* 这个函数的目的是比较两个树的根节点的值，根据它们的大小关系返回一个**order**值，可能是**Greater**、**Less**或**Equal**，表示第一个树的根节点值大、小或等于第二个树的根节点值。

**SwapDown**函数：

* 描述：给定一个树，该函数确保返回一个minheap，其中包含与输入树中相同的元素，同时保持minheap属性。
* 参数：一个树作为输入，要求输入树的子树都是minheaps。
* 返回值：一个minheap，其中包含与输入树相同的元素。
* 这个函数的目的是确保给定的树 **t** 满足 minheap 的性质。它要检查树的根节点和其子节点之间的大小关系，并进行必要的交换，以确保树仍然是一个 minheap。

**heapify**函数:

* 描述：给定任意形状和结构的树 **t**，该函数将这个树转换成一个 minheap，这个 minheap 包含了与输入树 **t** 中相同的元素
* 参数：一个任意的树 **t**。
* 返回值：返回一个 minheap 树，包含与输入树相同的元素。
* 这个函数的目的是将任意给定的树转换为一个 minheap 树，保持树的结构不变，只是根据 minheap 性质对节点进行必要的交换。

1. 在作业中分析SwapDown 和heapify两个函数的work和span。

## 解题思路与代码

* 首先定义树类型，这里的 **Br** 是一个构造函数，它用于创建包含一个值的树节点，这个节点由左子树、整数值和右子树组成。**Empty**代表一个空树，也就是没有节点的树。

图形用户界面, 应用程序

描述已自动生成

* **treecompare**函数：

函数接受两个参数，分别是两棵树，即 **(t1, t2)，**该函数思路较为简单，我们默认**非空树>空树，**将所有可能的结果列举出来，并分情况给出其相应的order即可。

文本

描述已自动生成

* **SwapDown**函数：

根据函数定义的要求，可知输入树的子树为minheap树，最后输出的树仍是一个minheap树。

我们仍分情况讨论：

1. 如果输入的树是空树（**Empty**），则直接返回空树。这是**SwapDown**函数的**base case**之一，表示没有子树可以继续进行交换操作。



1. 如果输入的树是一个单节点的树（**Br(Empty, x, Empty)**），那么这个树已经是一个最小堆，因为它只有一个节点，直接返回。同样是**base case**。



1. 如果输入的树是一个根节点和非空子树（minheap树）组成的树，那么我们比较根节点**y**和两个子树的根节点**x**、**z**，根据**x**、**y**、**z**大小关系进行交换，使其满足根节点<左子树的根节点且根节点<右子树的根节点。分别对左右子树进行SwapDown操作。以SwapDown左子树为例，对于左子树来说,虽然其根节点经过交换，但的左右子树仍满足minheap性质，满足SwapDown函数的输入要求，根据定义SwapDown 的输出是一个minheap，且其根节点的值小于等于SwapDown之前的根节点的值，右子树同理。又由于经过了交换，左子树的根节点和右子树的根节点均大于根节点，故整棵树是一个minheap树，至此通过递归调用Swap Down函数实现了SwapDown操作。

文本

描述已自动生成

注意尽量避免不必要的SwapDown，只有当子树的根节点和原本的根节点进行交换时，才对子树进行SwapDown操作。

* **heapify**函数：

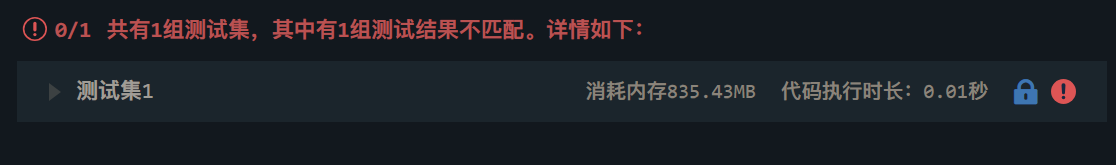
根据heapify函数的定义，它将一棵树变为minheap树。可以将原问题分解为两个较小的子问题进行求解，再将两部分解进行组合得到最终解。即对左右子树进行heapify操作得到两个minheap树，此时满足swapDown函数的定义，调用swapDown函数组合两子问题解得到最终的minheap树。

文本

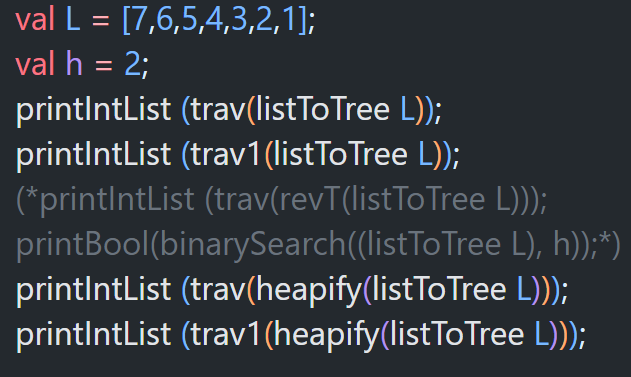
描述已自动生成

## 遇到的问题及运行结果

* 问题：未能通过头歌测试



* 运行结果：尽管未能通过头歌测试，为了检验代码的正确性，我通过分析heapify之后树的结构来判断其是否为minheap树，得到运行结果如下。



文本

描述已自动生成

其中listTOTree将int list类型的数据转化为tree类型，trav函数中序遍历一个tree，trav1先序遍历一个tree，将tree转化为int list。printIntList将int list打印出来。

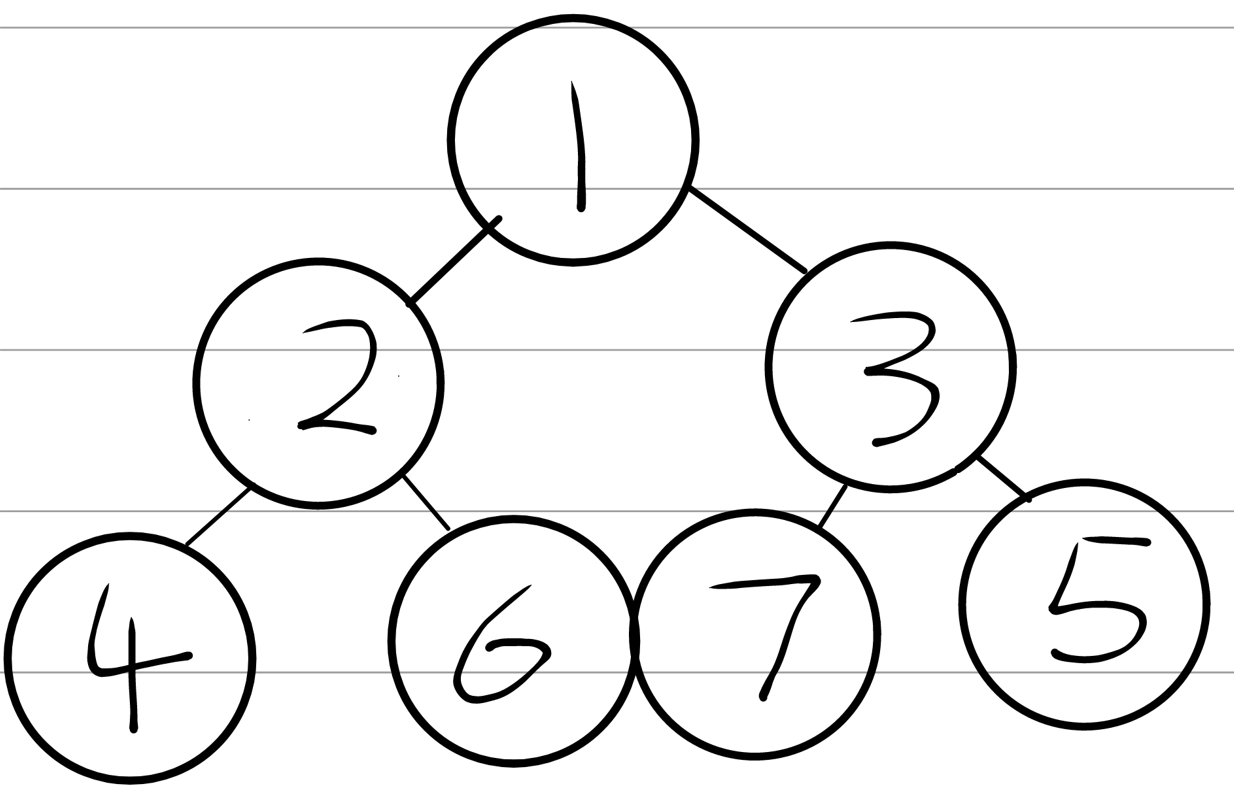
经过我的分析，listToTree得到的树的结构为

图片包含 形状

描述已自动生成

可以看出它并不是一个minheap树

Heapify之后树的结构为



此时是一个minheap树。

同时我通过自测运行，打印出了头歌平台上int List L的输入，虽然结果不一样，但经过类似分析，heapify之后得到的同样是一个minheap树。

## 性能分析

* **SwapDown** 函数：

Work： 在最坏情况下，SwapDown 会递归调用到叶子节点。因为SwapDown函数在树的每层的Work为常数级，之后递归调用对子树的SwapDown。故SwapDown 的工作量与树的高度成正比。

因为SwapDown函数只进行节点的值的交换，不会改变树的结构，故树高由ListToTree函数决定。

文本

描述已自动生成

又因为split函数将int list尽可能地进行均分，故左右子树的大小（所含节点的数目）

大体相近，故树高的期望值为O(log n),故SwapDown的Work为O(log n)数量级。当然，若假设原本的树高为h，则Span为O(h)数量级。

Span： SwapDown是一个递归函数，每次递归调用都必须等待其子调用完成。在最坏情况下，SwapDown 的并行性与树的高度成正比，为O(log n)数量级。当然，若假设原本的树高为h，则Span为O(h)数量级。

* **heapify** 函数：

Work：SwapDown 阶段work是O(log n)，其中 n 是树中的节点数。由于heapify递归调用heapify left ，heapify right，设heapify的Work为W(n),则平均状况下有递推表达式W(n)=O(log n)+2W(n/2),故W(n)=O(nlog n) 。当然，若假设原本的树高为h，则Work的分析类似，Work为O()数量级。

Span：SwapDown 阶段的span与树的高度成正比，其期望值为O(log n)。 heapify递归调用heapify left ，heapify right，两部分计算没有依赖关系，但SwapDown操作依赖递归调用的结果。故设heapify的Span为S(n),则平均状况下有递推表达式S(n)=O(log n)+S(n/2),故S(n)=O()。

若假设树高为h,则Span的分析类似，为O()数量级。

# 二、课程总结和建议

总结：这门课让我了解了函数式编程语言SML，增强了我的并行思维和函数式编程的思想。

建议：希望这门课开设在并行数据结构与算法（CMU-15210）课程之前，因为该课程的实验需要自学SML语言，且实验难度较大。个人认为函数式编程原理是并行数据结构与算法的先导课程，但教务的排课顺序不是很合理。