

函数式编程原理

课程报告

姓 名：

学 号：U2019

班 级：校交1902班

指导教师：郑然

**计算机科学与技术学院**

**2022 年 4 月 16 日**

1. 上机实验心得体会

**1.1 代码分析**

(1)value函数:

代码:

fun value(Empty) = 0

| value(Node(l,x,r)) = x;

功能:这个函数主要找到每个树的根结点.

函数类型:tree->int

输入输出关系:如果输入是空,那就返回0,如果输入是一个正常的结点,那就返回结点的根值即可.

1. swap函数:

代码:

fun swap (Empty,\_:int) = Empty

| swap (Node(l,x,r),y) = Node(l,y,r);

功能:将每个函数的根与一个特定的值交换.

函数类型:tree\*int->tree

输入输出关系:如果输入是空,那就返回Empty,如果输入是一个正常的结点,那就将这个结点的根植改为输入来的值返回即可.

1. treecompare函数:

代码:

fun treecompare (Empty,Empty) = EQUAL

| treecompare(Empty,Node(t1,x,t2)) = LESS

| treecompare(Node(t1,x,t2),Empty) = GREATER

| treecompare(Node(left1,x1,right1),Node(left2,x2,right2))=

if (x1 > x2) then GREATER

else if (x1 < x2) then LESS

else EQUAL;

功能:比较两个树,比较的结果是左边的树大于、小于或等于右边的树.

函数类型:tree\*tree->order

输入输出关系:如果输入有空值,则非空的一方比较大,如果不为空值那么就比较两个结点的根结点的值即可.

1. SwapDown函数:(这是没有使用treecompare的版本)

代码:

fun SwapDown Empty = Empty

| SwapDown (Node(Empty,x,Empty)) = Node(Empty,x,Empty)

| SwapDown (Node(left,x,right))=

if value(left)>x andalso value(right)>x then Node(left,x,right)

else if value(left)<value(right) then

Node(SwapDown(swap(left,x)),value(left),right)

else

Node(left,value(right),SwapDown(swap(right,x)));

功能:给定一个树,将树的根结点插入在合适的位置,满足小根堆.

函数类型:tree->tree

输入输出关系:递归插入,有两种停止的情况,第一是根的值小于左子树和右子树根的值,第二是左子树和右子树都为空,那么就可以停止了.如果不满足,x要插入到根的值比较小的那个子树中,递归调用Swapdown函数.

1. heapify函数:

代码:

fun heapify Empty = Empty

| heapify (Node(left,root,right))=

let

val temp\_t1 = heapify(left)

val temp\_t2 = heapify(right)

val temp\_t3 = Node(temp\_t1,root,temp\_t2);

in SwapDown(temp\_t3)

end;

功能:给定一个树,将其转化为大根堆.

函数类型:tree->tree

输入输出关系:递归调用,将左子树转化为大根堆,然后将右子树转化为大根堆,再把根结点插入到合适的位置即可.

**1.2 运行结果**

下面我们选取几个例子来运行一遍观察结果,当然在运行前我们首先需要构造几个函数来唯一确定树:

这个时候我们有check函数

代码:

fun check(Empty:tree):bool = true

| check(Node(Empty,x,Empty)) = true

| check(Node(Empty,x,Node(l,v,r))) = x < v andalso check(Node(l,v,r))

| check(Node(Node(l,v,r),x,Empty)) = x < v andalso check(Node(l,v,r))

| check(Node(Node(l1,v1,r1), x, Node(l2,v2,r2))) = x < v1 andalso x < v2 andalso check(Node(l1,v1,r1)) andalso check(Node(l2,v2,r2));

功能:给定一个树,判断满不满足小根堆.

函数类型:tree->bool

输入输出关系:判断当前结点与左右子树的值是不是满足要求,再递归判断左子树和右子树是否满足要求即可.

例子1:[7,6,5,4,3,2,1];

输出:true.

例子2:[1,2,3,4,5,6,7];

输出:true.

例子3:[]

输出:true.

通过三个例子可以初步判断函数的实现暂无问题.

**1.3 时间分析**

(1) 假设二叉树的层数是n:

对于Swapdown:由于不存在并行性质,所以说Work和Span是相等的.

W(n)=k+W(n-1)

由主方法定力,可以证明是Work=Span=O(n).

(2) 假设二叉树的结点数是n:

对于heapify的串行执行速率:

W(n)=k+2W(\frac{n}{2})

由主方法,答案是Work=O(n).

对于heapify的并行执行速率:

S(n)=k+S(\frac{n}{2})

由主方法,答案是Span=O(logn).

**1.4 出现的问题**

1. 出现了类型匹配的问题,一直都说Node(left,x,Empty)没考虑,所以说我更换了方法,避开了val x = Node(l,x,r)的讨论
2. 发现一直都不是小根堆.

原来发现Swapdown的时候只做了一层:Node(swap(left,x),value(left),right),这样子根植只会插入在第二行合适的位置,但是根植不一定就在第二行,所以说导致错误,解决办法是加上递归调用就好了.

二、课程建议和意见

**2.1 课程知识的总结**

1. 函数式语言的背景介绍:

包括函数式语言的数学背景、发展,还有函数式编程语言大家族.

1. 基本语法:
   1. 函数式语言的基本类型.
   2. 表达式和运算符.
   3. 声明及其作用域.
   4. 模式匹配以及结合律.
2. 程序的正确性分析:

包括基本的简单归纳法、完全归纳法、良基归纳法.

1. 程序的效率分析:
   1. 串行效率Work.
   2. 并行效率 Span.
2. 基本算法:
   1. 基本排序:插入&归并.
   2. 树算法以及基于树的排序.
3. 类型推导:
   1. 静态类型推导(编译时的推导).
   2. 动态类型推导(多态函数).
4. 高阶函数特性(本课程的重点和难点)
   1. 函数作为参数传入.
   2. 函数作为返回值.
   3. 函数的柯里化.((a\*b->c转化为a->b->c)).

**2.2 课程的建议**

1. 可以讲一讲其他函数式编程语言的特性,比如说Haskell等.
2. 我觉得不用花很长时间讲程序的正确性验证以及程序的性能计算,这些其实高中还有算法课上讲过,感觉着重去讲函数的高阶性质会更好,比如说Currying我就学得不是很明白.
3. 对于实验3的最后一题,还是希望能写一个判断是不是小根堆的函数最好.
4. 对于头歌实验,希望每个实验能设计多个测试用例最好.