函数式编程原理
课程报告

姓　　名：　　廖翔
学　　号：**U201915116**
班　　级：　**CS1906**
指导教师：　　顾琳

计算机科学与技术学院

2022　年　5 月 1 日

一、上机实验心得体会

题目要求：该题需要我们编写三个函数，treecompare，swapdown 和 heapify，最后通过调用 heapify 函数来判断实验结果。

treecompare 函数接受两颗树为输入，根据两颗树在根结点上的值，返回 GREATER，EQUAL 或 LESS。swapdown 函数则是接受一棵树为输入，返回该树的最小堆。而 heapify 函数的功能就是给定一个任意的树 t，转换为一个 minheap，而其元素正是 t 的元素。

在解题之前，首先要明确以下定义

数据结构定义: datatype tree = Empty | Br of tree * int * tree;

最小堆定义: 树为空。或者对于任意一个结点，其左右子树为空且其根节点小于等于左右子树的根节点。

思路：通过分析题目要求之后，显然本题的重中之重就是 swapdown 函数。显然，递归是我们考虑该问题的方式。首先列出几种特殊情况.

(1) 对于传入一个空树，显然直接返回 empty 即可。

(2) 而对于只有一个结点的树，也是直接返回该树即可。

(3) 对于左子树或者右子树为空的树，在这里以右子树为空举例，左子树同理。如果根节点比左子树的根节点小或相等，保持该树不变即可，反之如果更大，则需要将根节点与左子树的根节点互换，同时对于左子树递归调用 swapdown 函数。

(4) 列出了上述的特殊情况后，对于一般情况的分析就比较轻车熟路

了，即左右子树均不为空的情况。同样采用分类讨论的思路。如果左子树的根节点小于等于右子树的根节点，且根节点小于等于左子树根节点，则当前树已经是一个最小堆树，保持不变即可，如果根节点比左子树根节点大，则需要进行交换，并对交换后的树继续调用 heapify 函数。对于左子树根节点大于右子树根节点的情况同理。基于上述分析，可以直接写出 swapdown 函数的代码：

```
fun Swapdown Empty = Empty
  | Swapdown (Br(Empty, x, Empty)) = Br(Empty, x, Empty)
  | Swapdown (Br(Empty, x, Br(a,b,c))) = if x<=b then Br(Empty, x, Br(a,b,c))
                                         else Br(Empty, b, Swapdown(Br(a,x,c)))
  | Swapdown (Br(Br(a,b,c),x,Empty)) = if x<=b then Br(Br(a,b,c),x,Empty)
                                       else Br(Swapdown(Br(a,x,c)),b,Empty)
  | Swapdown (Br(Br(a,b,c),x,Br(d,e,f))) = if b<=e then
                                           if x<=b then Br(Br(a,b,c),x,Br(d,e,f))
                                           else Br(Swapdown(Br(a,x,c)),b,Br(d,e,f))
                                         else
                                           if x<=e then Br(Br(a,b,c),x,Br(d,e,f))
                                           else Br(Br(a,b,c),e,Swapdown(Br(d,x,f)));
```

对于 treecompare 函数，也非常简单，如下:

```
fun treecompare(Empty, Empty) = EQUAL
  | treecompare(Empty, Br(a,b,c)) = GREATER
  | treecompare(Br(a,b,c), Empty) = LESS
  | treecompare(Br(a,b,c), Br(d,e,f)) = Int.compare(b,e);
```

最后的 heapify 函数，也只需要对左右子树分别调用 heapify 函数之后，再调用 swapdown 函数即可:

```
fun heapify Empty = Empty
  | heapify (Br(Empty, x, Empty)) = Br(Empty, x, Empty)
```

```
| heapify (Br(a,x,b)) = Swapdown(Br(heapify a,x,heapify b));
```

该实验的完整代码会放在后面的附录中，最后的运行结果如下:

可以看到，经过处理后，传入的树变成了一个以 1 为根节点，且左右子树均为最小堆的最小堆树，可以说明我们的代码编写功能正确。

对于函数的性能分析如下:

Work of SwapDown is W d

Span of SwapDown is S d

Work of heapify is W 2^d*d^2

Span of heapify is S d^2

在整个解题过程中遇到的问题就是如果确保 swapdown 函数中保持不变的树已经是一个最小堆了，这个问题通过 heapify 函数解决，因为 heapify 函数会对左右子树递归调用 heapify，这样就能确保在最外面的一层的 swapdown 函数中接受的参数已经都是变换好的最小堆了。

心得与体会:

因为疫情的缘故，我不能到校参与理论课的讲授和机房上机的实验课。不过因为老师和学校的安排，这些都没有成为太大的问题，通过腾讯会议也能学到一样的东西，机房上机平常作业也可以直接通过头哥平台提交，所以还是顺利地完成了该课程的学习。

函数式编程原理这门课程，让我学会用不同的角度去审视编程这一艺术，在函数式中，函数代表的是数学概念里面的函数，描述映射(计算)关系(做什么)。对于函数式编程来说，显然也有他独特的一些特性，例如，因为没有可变状态，也没有 for,while 循环，所以函数式编程非常需要依赖递归，而这个概念也是围绕整个实验设计中的。并发性和确定性是函数式编程的一大优点，因为函数无副作用，所以原生并发友好，且可读性高，易于测试和调试。而这也给它带来了一个缺点就是处理可变状态的 IO 能力比较差。

Anyway，作为一门专选课，函数式编程这门课程扩大了我的编程知识面，也让我在以后的编写代码过程中对递归有了更深刻的认识，给我提供了一种全新的编程思想。

二、课程建议和意见

(1)实验中可以加入一些高阶函数的内容，并且加一些复杂度的限制，可以使得实验更加有挑战性。

(2)老师可以对每一个章节设计一个跟实验不一样的小一点的实践任务，让同学们将课件中提及的一些经典代码自己实践一遍，这样会更加便于理解一些抽象的问题。

三、附录

```
fun printInt (a:int) =
   print(Int.toString(a)^" ");

fun getInt () =
   Option.valOf (TextIO.scanStream (Int.scan StringCvt.DEC) TextIO.stdIn);

fun printIntList ( [] ) = ()
  | printIntList ( x::xs ) =
    let
  val tmp = printInt(x)
    in
```

```
 printIntList(xs)
   end;

fun getIntList ( 0 ) = []
 | getIntList ( N:int) = getInt()::getIntList(N-1);

fun split [ ]  = ([ ], [ ])
   | split [x] = ([ ], [x])
   | split (x::y::L) =
 let val (A, B) =split L
 in (x::A, y::B)
 end;

datatype tree = Empty | Br of tree * int * tree;

fun trav(Br(t1,a,t2)) = trav(t1)@(a::trav(t2))
   |trav empty = [];

fun listToTree ([] : int list) : tree = Empty
 | listToTree (x::l) = let val (l1, l2) = split l
    in Br(listToTree l1, x, listToTree l2)
 end;

fun treecompare(Empty, Empty) = EQUAL
 | treecompare(Empty, Br(a,b,c)) = GREATER
 | treecompare(Br(a,b,c), Empty) = LESS
 | treecompare(Br(a,b,c), Br(d,e,f)) = Int.compare(b,e);

fun Swapdown Empty = Empty
 | Swapdown (Br(Empty, x, Empty)) = Br(Empty, x, Empty)
 | Swapdown (Br(Empty, x, Br(a,b,c))) = if x<=b then Br(Empty, x, Br(a,b,c))
                                  else Br(Empty, b, Swapdown(Br(a,x,c)))
 | Swapdown (Br(Br(a,b,c),x,Empty)) = if x<=b then Br(Br(a,b,c),x,Empty)
                                 else Br(Swapdown(Br(a,x,c)),b,Empty)
 | Swapdown (Br(Br(a,b,c),x,Br(d,e,f))) = if b<=e then
                                    if x<=b then Br(Br(a,b,c),x,Br(d,e,f))
                                    else Br(Swapdown(Br(a,x,c)),b,Br(d,e,f))
                                  else
                                    if x<=e then Br(Br(a,b,c),x,Br(d,e,f))
                                    else Br(Br(a,b,c),e,Swapdown(Br(d,x,f)));

fun heapify Empty = Empty
 | heapify (Br(Empty, x, Empty)) = Br(Empty, x, Empty)
 | heapify (Br(a,x,b)) = Swapdown(Br(heapify a,x,heapify b));


val L = getIntList(7);
printIntList (trav(heapify(listToTree L)));
```