COOL 语言二叉搜索树的设计与实现

数据科学与大数据技术
学生姓名: [蒋雯丽]
学号: [20238131061]

2025 年 9 月 30 日

GitHub 仓库链接:

https://github.com/Lilian325/homework1

摘要

本报告详细阐述了使用 COOL (Classroom Object Oriented Language) 语言实现二叉搜索树 (Binary Search Tree, BST) 的过程。该项目旨在通过构建复杂数据结构，深入理解 COOL 语言的面向对象特性、递归机制以及静态类型系统的约束。实验结果表明，所实现的代码成功完成了节点的插入、查找以及中序遍历功能，验证了算法的正确性和鲁棒性。

1 设计与实现

1.1 设计理念

本此实现遵循模块化的面向对象设计原则。由于 COOL 是一种强类型语言，我们将数据存储逻辑（Node）与数据结构接口（BST）分离。该设计大量依赖**递归**来处理树形结构，这与 COOL 语言对递归调用的良好支持相契合。

1.2 类结构设计

本实现主要包含三个核心类：

1.2.1 1. Node 类 (节点)

这是树的基本构建单元，负责存储数据和维持树的拓扑结构。

继承关系: 继承自 `IO` 类（便于调试输出）。

属性:

`val : Int` - 存储整型数据。

`left : Node` - 指向左子节点的引用。

`right : Node` - 指向右子节点的引用。

核心方法:

`init(v : Int) : Node` - 初始化节点值。

`insert(new_val : Int) : Object` - 递归插入新值。

`search(target : Int) : Bool` - 递归查找目标值。

`print_in_order() : Object` - 递归执行左-根-右遍历打印。

1.2.2 2. BST 类 (树包装器)

该类作为对外的接口封装，管理树的根节点，并处理空树等边界情况。

属性: `root : Node` - 指向树的根节点（初始为 void）。

方法: 将 `insert`, `search`, `print` 等操作委托给根节点处理，在此之前先进行非空检查。

1.2.3 3. Main 类

程序的入口点，负责实例化 BST 对象并运行预设的测试用例。

1.3 核心算法实现

核心逻辑位于 Node.insert 方法中。由于 COOL 不支持像 C 语言那样的显式指针操作，我们通过修改对象的 left 或 right 属性来构建树。

```
        insert(new_val : Int) : Object {
            if new_val < val then
            -- 如果左子节点为空(void)，则在此处创建新节点
            if (isvoid left) then
            left <- (new Node).init(new_val)
            else
            -- 否则，向左子树递归
            left.insert(new_val)
            fi
            else if val < new_val then
            -- 右子树的对称逻辑
            if (isvoid right) then
            right <- (new Node).init(new_val)
            else
            right.insert(new_val)
            fi
            else
            self -- 忽略重复值
            fi fi
        };
```

Listing 1: COOL 中的递归插入逻辑

1.4 遇到的挑战与解决方案

挑战：处理 void 引用

在 COOL 中，未初始化的对象默认为 void。如果尝试对 void 对象调用方法，会触发运行时错误 ("Dispatch to void")。这是与 Java (NullPointerException) 类似但处理方式不同的机制。

解决方案：

采用了防御性编程策略，使用 isvoid 谓词进行严格检查。例如，在 BST 类中，必须先检查 root 是否为空，才能调用其方法：

```
        insert(val : Int) : Object {
            if (isvoid root) then
            root <- (new Node).init(val)
            else
```

```
            root.insert(val)
            fi
        };
```

2 测试与结果

2.1 测试代码 (Main 类)

Main 类构建了一棵树，并以乱序 (50, 30, 20, 40, 70, 60, 80) 插入数据，以测试二叉树的自动排序特性。

```
        out_string("Inserting: 50, 30, 20, 40, 70, 60, 80\n");
        tree.insert(50); tree.insert(30); tree.insert(20);
        tree.insert(40); tree.insert(70); tree.insert(60); tree.
           insert(80);

        out_string("In-order Traversal: ");
        tree.print_in_order();
```

Listing 2: Main 类中的测试序列

2.2 运行结果

程序使用 coolc 编译并在 spim MIPS 模拟器上运行。输出结果显示，中序遍历成功输出了有序序列，验证了 BST 的性质。

```
lilian@lilian-virtual-machine:~$ gedit bst.cl
lilian@lilian-virtual-machine:~$ coolc bst.cl
lilian@lilian-virtual-machine:~$ spim bst.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
=== BST Challenge Implementation ===
Inserting: 50, 30, 20, 40, 70, 60, 80
In-order Traversal: 20 30 40 50 60 70 80
Searching for 40: Found!
Searching for 90: Not Found.
Searching for 20: Found!
=== Test Finished ===
COOL program successfully executed
lilian@lilian-virtual-machine:~$
```

图 1: COOL 程序运行结果截图

2.3 结果分析

1. **排序正确性**: 尽管输入是乱序的，输出序列 `20 30 40 50 60 70 80` 是严格递增的。这证明了 `insert` 方法正确地维护了二叉搜索树的左小右大性质。

2. **查找准确性**: 对存在节点 (40, 20) 的查找返回了 `true`，对不存在节点 (90) 的查找返回了 `false`，验证了递归查找逻辑的正确性。

3 结论

我在做这个作业的时候，首先要通过查找一些详细的资料去充分地认识到 cool 语言的语法特征。cool 语言是为了体现“编译”特性而产生的语言，与以往学过的主流语言有很大的区别：

1. 类型系统采用严格的静态强类型且必须显式声明，不支持隐式转换；

2. 继承机制为单继承且所有类隐式继承自 Object 类，不支持接口或抽象类；

3. 空值处理使用独特的 void 关键字和 isvoid 运行时检查，而非 null；

4. 语法设计刻意简化，如使用 if-then-else-fi 和 loop-pool 结构，且赋值操作使用 <-符号；

5. 语言特性缺失较多，如不提供数组、异常处理、运算符重载等常见特性；

6. 执行环境依赖编译为 MIPS 汇编并在 SPIM 模拟器中运行，而非直接生成机器码或字节码。

而对于本次作业，实现二叉搜索树这样的递归数据结构，凸显了在无显式空指针语言中进行防御性编程（检查 `void`）的重要性。本项目成功满足了所有功能要求，包括挑战项中的二叉搜索树实现。