Lab1: 树

Lab1的主题是树，在本Lab你需要完成三个任务：

- 掌握并实现一个课堂上尚未学习过的树形数据结构：基数树（Radix Tree）。
- 对基础的基数树进行节点压缩（Node Compression）优化。
- 实现YCSB测试，在不同工作负载下比较测试基数树、优化后的基数树以及红黑树的性能。

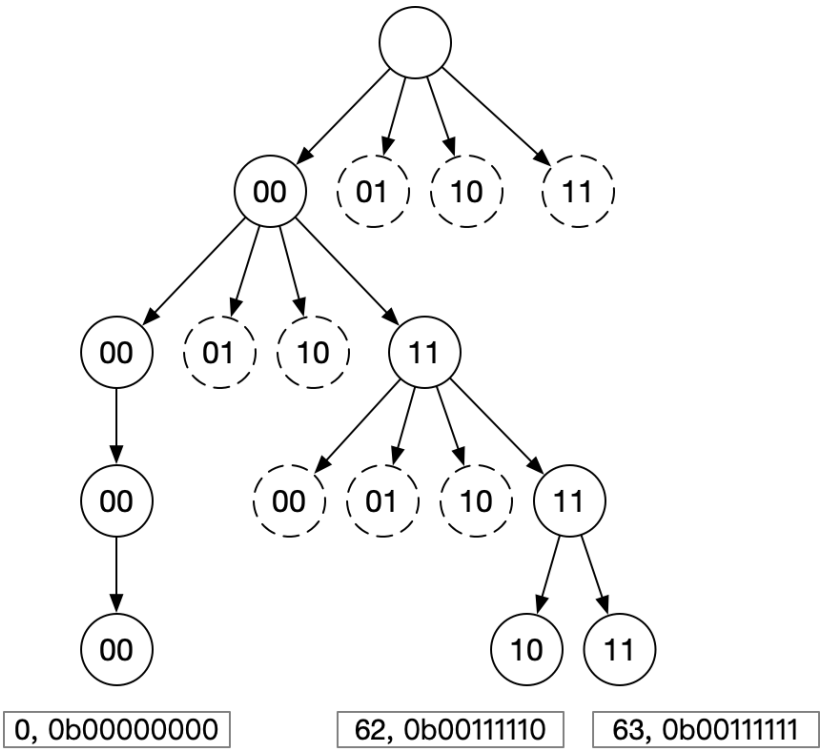
1 基数树

基数树是一种多叉搜索树，能够有效地存储和检索集合（set）或映射（map）的数据。基数树通过共享相同的前缀来节省空间，并提供快速的查找操作。由于该特点，基数树被广泛运用在IP路由、信息检索等领域中。在Linux中基数树被广泛应用于IDR机制、page cache索引、路由查找等实现中。在本节你需要掌握基数树的原理，并实现一个存储`int32_t`（32位整型）类型集合（Set）的4叉基数树。

1.1 结构

在基数树中，同一子树下的节点具有共同的前缀（存储字符串时有相同的字符前缀；存储整型时有共同的整型的二进制前缀）。

如下图所示为一个存储`int8_t`（8位整型，此处为了示例简短以`int8_t`为例，实现时需要存储`int32_t`）集合的4叉基数树。树的根节点有四个指针分别指向四个子节点，分别为00，01，10，11，代表这棵子树下的数字的二进制表示的前两位（其中01,10,11下没有存储数值，因此根节点维护的是空指针，图中用虚线表示这三个节点）。00节点同样有四个子节点，每个节点代表该节点下子树存储的数值有不相同的第3、4位（00,01,10,11）。根据这样的规律，将根节点到一个叶子节点路径上的4个节点的2比特拼接起来，即可得到叶子节点所代表的数字。例如，树中存储三个`int8_t`，分别是0，62和63。其中，从根节点到代表62的节点的路径包括00,11,11,10，拼接之后就是62的二进制表示0b00111110。



1.2 基本操作

在本节，你需要实现基数树的基本操作，包括查询（find），插入（insert）以及删除（erase）操作。以下会分别介绍这些接口的实现方式。在执行基本操作前，你需要初始化基数树：创建树的根节点，根节点的四个子节点均为空。

1.2.1 find

功能：查询一个`int32_t`的整型数字N是否在基数树维护的集合中。

实现：查询操作从基数树的根节点出发，从二进制表示最高位开始每次读取N的两个比特，并根据这两个比特决定继续在当前节点的哪个子节点查找。一旦对应的子节点不存在即可立即返回false，相反，如果能够完整遍历整个数字N并找到其对应的叶子节点，则返回true。

1.2.2 insert

功能：将一个`int32_t`的整型数字N插入到基数树中，如果N已在树中则不改动树的结构。

实现：插入操作首先对基数树的根节点进行插入。读取N的二进制表示最高位的两个比特，并查看这两个比特对应的子节点是否创建，若已经创建则可以继续对该子节点进行插入，若未创建则创建一个对应的新的子节点，接着使用N的第三、四位比特对该子节点继续插入。依此法即可遍历完整个数字N并插入对应数字N的叶子节点。

1.2.3 erase

功能：将一个`int32_t`的整型数字N从基数树中删除，若N原本就不存在于树中则改动树的结构。

实现：首先参照find操作的方法找到数字N对应的叶子节点，如果不存在，则直接返回，如果存在，则删除数字N对应的叶子节点。需要注意的是，如果删除一个叶子节点后，其父节点的非空子节点数目为0，则需要将该父节点删除，依此类推，直至根节点（根节点始终不删除，树中只有根节点，并且其四个子节点为空时，树即为空树）。

1.2.4 其它接口

除了上述基本操作以外，你还需要实现一些辅助接口，用来输出基数树的一些基本信息，测试程序会以此判断基数树实现的正确性。辅助接口包括：

- `size`：返回基数树中的节点个数（例图中基数树的节点个数为9，需要计入根节点。图中虚线圈出的节点表明该节点下没有保存数据，实现时父节点指向该节点的指针为空指针，图中仅为说明所以画出，不计入size）。
- `height`：返回基数树的高度（例图中基数树的高度为5，根节点为第1层）。

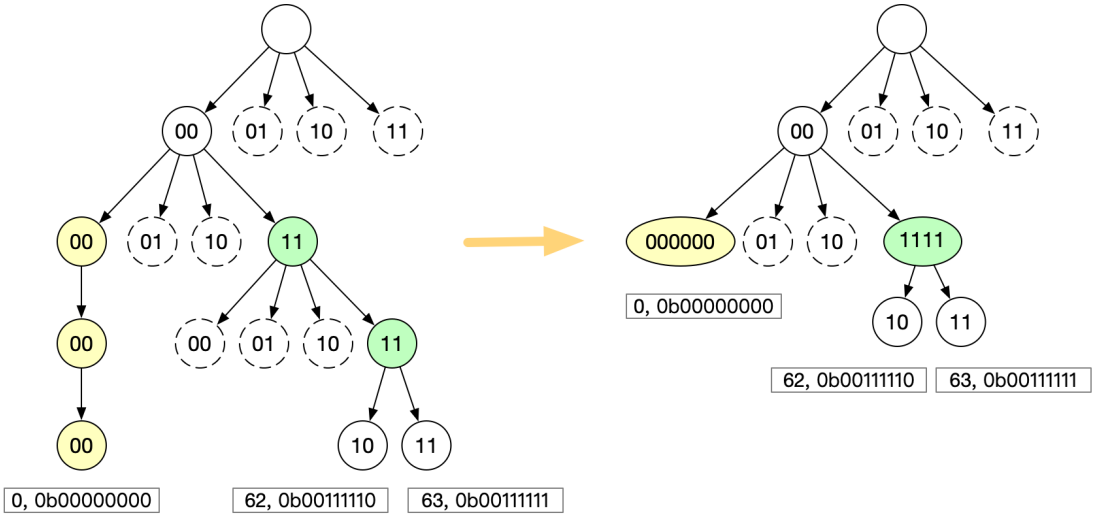
1.3 实现文件

- radix_tree.hpp
- radix_tree.cpp

2 节点压缩

在上一节的示例中，我们能够发现，目前的基数树的实现存在冗余的节点，这使得树的基本操作效率降低。具体来说，如果树中一个非根节点只有一个子节点，我们可以将该节点与其子节点进行合并，合并后的节点所代表的值为被合并节点代表的值拼接后的结果。由于节点数减少了，在执行基本操作时，就可以减少在树上进行遍历节点的节点跳转次数。如下图左侧所示，标有相同颜色的节点就可以合并成一个节点。基于这样

的观察我们可以对基数树进行节点压缩（node compression）：将只有一个子节点的父节点与其子节点进行合并，从而提升基本操作执行效率。节点压缩后的基数树如下图右侧所示，节点数为6，树高为4。



2.1 基础操作

本节介绍引入节点压缩优化后基础操作的变化。

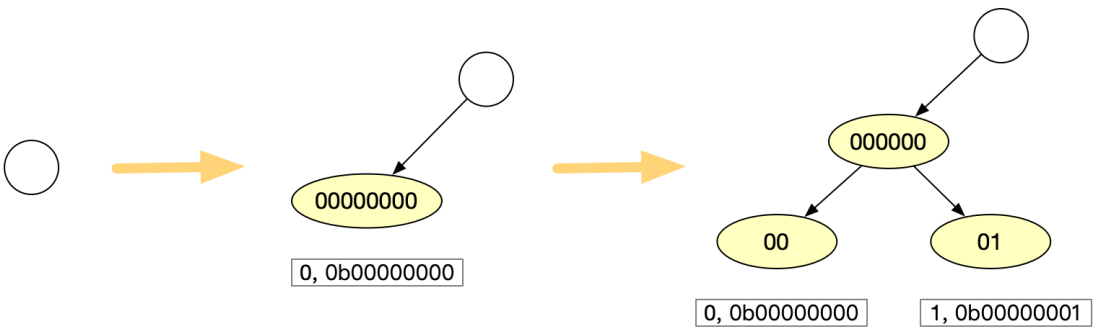
2.1.1 find

实现与基础基数树一致，但需要注意每个节点此时不仅可以代表默认的2比特组合，也可以代表任意2n个比特，因此需要额外维护每个子节点代表的前缀（包括前缀内容以及长度），并在查找时进行比对。

2.1.2 insert

在插入数字N时，首先在树上执行find操作直到以下三种情况之一发生：

- find操作找到了被插入的数字。则直接返回，不修改树的结构。
- find操作最终停在某个节点，该节点代表的前缀内容可以与数字N相匹配，数字N应当插入到该节点的某个子节点，但是该子节点尚未创建。该情况则创建一个新的节点，该节点保存数字N二进制表示已经遍历部分外的剩余部分。例如下图为存储`int8_t`的基数树的插入示例，在空树（树只有根节点）中插入0时，find操作最终停留在根节点，插入操作直接在根节点原本对应00的子节点插入保存0b00000000的节点。
- find操作最终停在某个节点，但是该节点代表的前缀内容不能与数字N相匹配。在该情况需要将该节点分裂，选取两个数字二进制表示的最长公共2n个比特前缀替代当前节点，并创建两个节点分别保存这两个数字后缀的不同部分。例如下图中，继续插入1时，find操作停留在0b00000000节点，0b00000000节点分裂为0b000000父节点以及0b00、0b01子节点。



2.1.3 erase

删除流程与基础基数树基本一致，除了在删除叶子节点后，若发现其父节点的子节点数变为1，需要将其与其子节点进行合并。

2.2 实现文件

- compressed_radix_tree.hpp
- compressed_radix_tree.cpp

3 测试

本次实验需要你学习编写测试程序并使用不同测试负载比较测试对象的性能。测试部分包括两项内容：

- 正确性测试：验证基数树及其节点压缩优化是否正确实现。
- YCSB测试：比较基数树、优化后基数树以及红黑树基本操作的性能。

3.1 正确性测试

该测试已实现在main函数中，并使用简单的数据样例进行检查。input文件夹为程序输入，expected文件夹为正确程序输出（带有compressed后缀的文件为优化后的基数树的输出文件），测试结果打印在output文件夹。测试程序比较你的输出结果与正确输出是否一致从而进行评判。

使用如下命令运行正确性测试：

```
# 编译链接正确性测试文件
make all
# 运行正确性测试
make grade
# 清理编译结果
make clean
```

3.2 YCSB测试

YCSB（Yahoo! Cloud Serving Benchmark）是一个常用的基准测试工具，它旨在模拟真实世界的工作负载，并提供一种标准化的方法来比较不同数据库管理系统（DBMS）的性能。

在本节你需要实现测试代码生成YCSB测试给出的几种工作负载模版，并比较在不同工作负载下基数树、优化后的基数树以及红黑树的基本操作的性能。其中红黑树的实现可以由你自己实现，可以从之前的作业中修改得到，也可以从网上自行查找选用（例如红黑树示例）。

3.2.1 测试对象

YCSB测试对象包括：

- 基数树
- 添加节点压缩优化后的基数树
- 红黑树

3.2.1 工作负载

你需要测试三种不同的工作负载，在所有工作负载下，测试程序不断循环对测试对象调用某一种基础操作（每种工作负载中会规定各种基础操作的占比），查询、插入、删除的数值均服从zipfian分布。在每轮测试开始前，你需要重新初始化测试对象，并加载1000个均匀随机分布的`int32_t`到测试对象中。

每组测试需要运行30~60s从而获得一个较为稳定的测试结果。三种不同的工作负载描述如下：

- workload-1: 在该负载下，每轮循环50%几率执行find操作,50%几率执行insert。
- workload-2: 在该负载下，每轮循环100%执行find操作。
- workload-3: 在该负载下，每轮循环50%执行find, 25%执行insert, 25%执行del。

注：zipfian分布是一种概率分布，通常用来描述自然语言中单词出现频率、Web 流量、文件大小、用户访问等现象。Zipfian分布得名于美国语言学家George Zipf，他发现自然语言中的单词频率与它们在排名上呈现出一种特定的关系。 **在util.hpp文件中实现了获取满足zipfian分布数字的工具函数，测试程序可以直接使用其进行编写。**

3.2.2 测试结果

该Lab仅要求记录基本操作的时延，不需要测试数据结构的吞吐量。对于不同工作负载，你需要对不同测试对象的相关基础操作记录以下数据并作图：

- 平均时延
- P50、P90、P99时延：P50是指一组数据中的中位数值（第50百分位数），相应的，P90时延是指一组数据中的第90百分位数，P99时延是指一组数据中的第99百分位数。

4 提交要求

你的提交内容应该包括：

- 所有相关代码
- 一份README：
  - 说明如何编译、运行测试、获取实验数据结果。
- 一份实验报告：
  - 简单列举实现过程中遇到的难点、或是印象较深的细节
  - 你的实验数据结果以及对实验结果的作图及简要分析
  - 如果测试使用的红黑树不是自己实现的，需要在报告中注明源码来源
  - 实验报告篇幅在4页以内，参考模版已有约两页

请将相关的代码和实验报告打包上传Canvas，命名使用“学号+姓名+Lab1”，如“5xxxxxxxxxx+张三+Lab1.zip”。

5 建议

- 在实现节点压缩优化时，需要进行许多位操作，你可以使用GCC一些内置的位操作工具函数辅助编写。例如 **__builtin_clz** 函数可以返回一个int最高位开始的连续的0的个数。
- YCSB测试可以基于基类Tree进行编写，使用基类指针指向不同的测试对象，可以更好地重用测试代码，保持代码简洁。