项目说明

在最后一次作业中，您需要运用本学期的知识来实现一个综合性的C++项目—**Copy-On-Write Trie**。

该作业改编自CMU 15-445 2023Fall Project0，在卡内基梅隆大学，学生必须以满分完成该项目，然后才能继续学习数据库系统基础课程。如果您对助教翻译的文档不满意，我们推荐您可以在下面的链接中阅读英文原版的Task1和Task2部分。

> **项目 #0 - C++ 入门 |CMU 15-445/645 ： ： 数据库系统简介（2023 年秋季）**

为了确保您了解一些最基础的C++并发编程知识，我们建议您在进入Task2前完成第四次家庭作业中的《并发控制》。

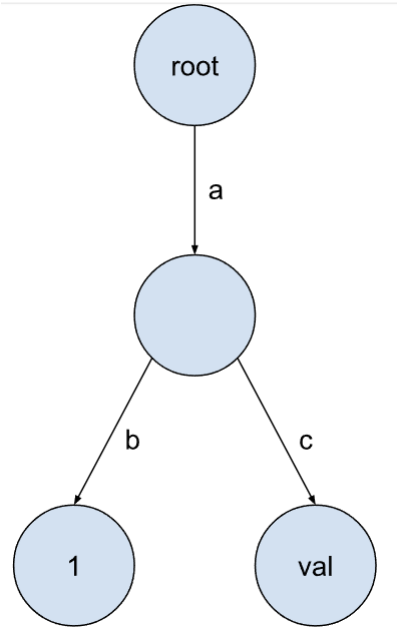
作业的框架代码已经公布在课程QQ群中。对于本题，您可以在自己的IDE中编写代码，请确保您在本地配置好**C++17**的开发环境。

项目描述

在本项目中，您将实现一个基于写时复制字典树（copy-on-write **trie**）的键值（key-value）存储容器。Tries 是高效的有序树型数据结构，用于检索给定键的值。为了简化解释，我们将假设键是可变长度的字符串，但实际上它们可以是任意类型。

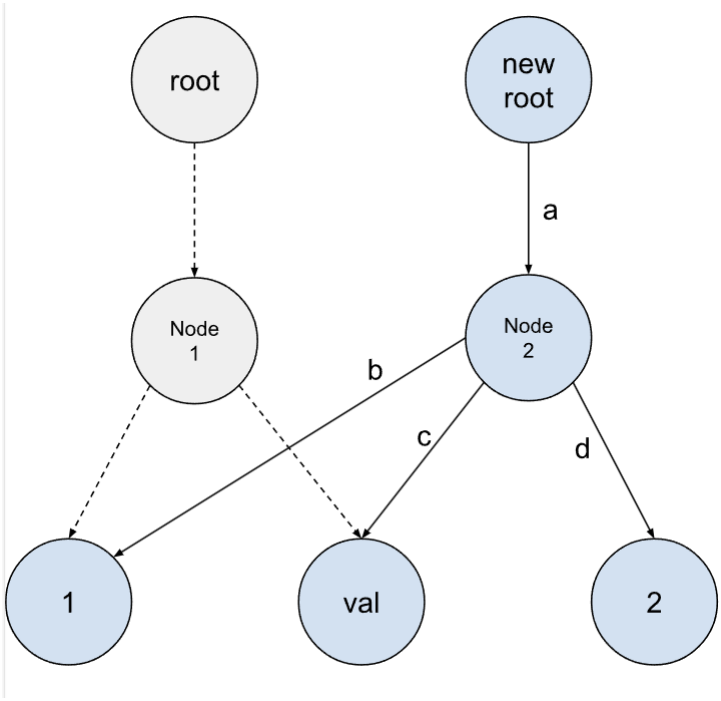
trie 中的每个节点都可以有多个子节点，表示不同的可能的 next 字符。

您将实现的键值存储可以存储映射到任何类型值的字符串键。键的值存储在表示该键的最后一个字符的节点（又名终端节点）中。例如，考虑将 kv 对 `("ab", 1)` 和 `("ac", "val")` 插入到 trie 中。

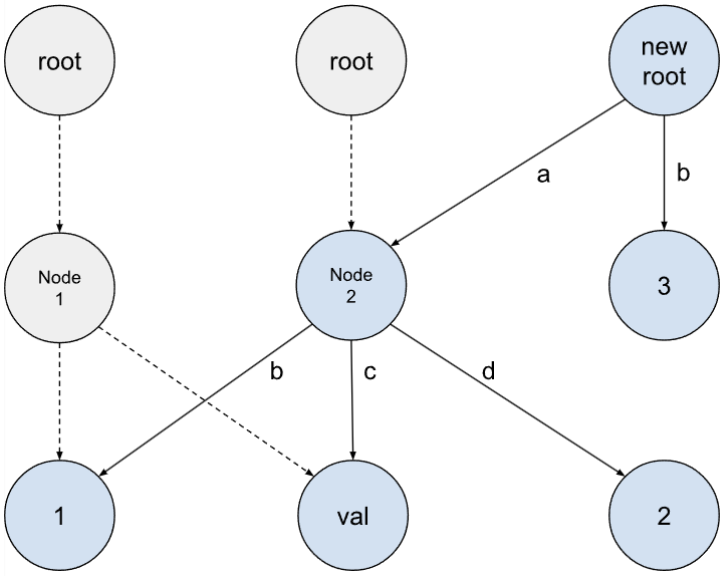


Task 1：实现Copy-On-Write Trie (71%)

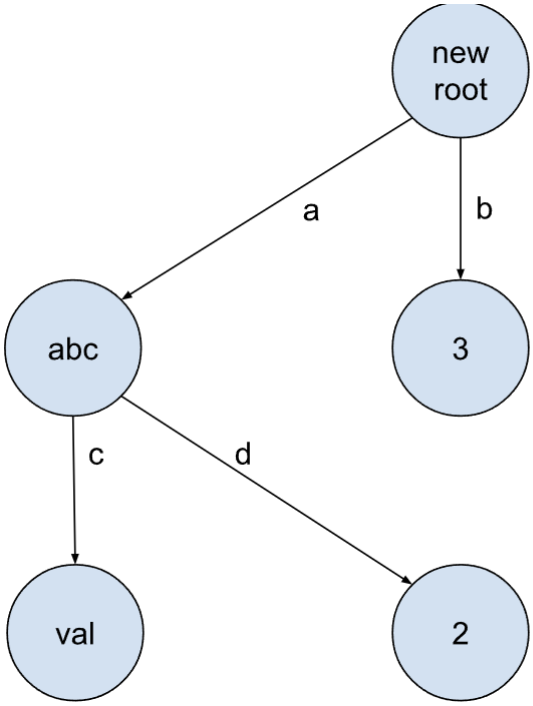
在Task1中，您需要修改 `trie.h` 和 `trie.cpp` 来实现 copy-on-write trie。在写时复制的trie中，操作不会直接修改原始trie的节点，而是会为修改后的数据创建新的节点，并为新修改的trie返回新的根节点。Copy-on-write 使我们能够在每次操作后随时以最小的开销访问 trie。考虑在上面的示例中插入 `("ad", 2)`，我们通过重用原始树中的两个子节点并创建一个新的 node 2 来创建新的节点 `Node2`。（见下图）



如果我们随后插入 `("b", 3)`，我们将创建一个新的根、一个新的节点并重用以前的节点。这样，我们可以在每次插入操作之前和之后获取 trie 的内容。只要我们有根对象 （ `Trie` 类），我们就可以在那个时候访问 trie 里面的数据。（见下图）



再举一个例子：如果我们插入 `("a", "abc")` 并删除 `("ab", 1)`，我们可以得到下面的trie。请注意，父节点可以具有值，**删除后需要清除所有不必要的节点**。空 trie 用 `nullptr` 表示。



您的 trie 必须支持三个操作：

- `Get(key)`：获取 key 对应的值。
- `Put(key, value)`：为 key 设置相应的值。如果键已存在，则覆盖现有值。请注意，值的类型可能是不可复制的（即 `std::unique_ptr`）。此方法返回一个新的 trie。
- `Delete(key)`：删除键的值。此方法返回一个新的 trie。

这些操作都不应直接在 trie 本身上执行。您应该创建新的 trie 节点并尽可能重用现有的 trie 节点。

要创建一个全新的节点（即一个没有子节点的新叶节点），你可以简单地使用 `TrieNodeWithValue` 构造函数构造对象，然后将其变成一个智能指针。要 copy-on-write 创建新节点，您应该使用 `TrieNode` 上的 `Clone` 方法。要在新 trie 中重用现有节点，您可以复制 `std::shared_ptr<TrieNode>`：复制共享指针不会复制底层数据。您**不应该**在此项目中使用 `new` 和 `delete` 手动分配内存。 `std::shared_ptr` 将在没有人引用底层对象时释放该对象。

> Note：在下学期的《软件系统设计》课程中，我们会学习相关的设计模式。感兴趣的同学可以提前了解"原型模式"。

有关这些操作的完整规范，请参阅起始代码中的**注释。您的实现应按照上面的例子存储数据。**不要将 C 字符串终止符 `\0` 存储在 trie 中。**禁止从类定义中删除任何 `const` 或使用 `mutable` / `const_cast` 来绕过 const 检查。**

正确完成上面的代码，你可以得到71%的分数。

Task 2：并发Key-Value存储 (29%)

在拥有可在单线程环境中使用的写入时复制 trie 后，为多线程环境实现并发键值存储。在此任务中，您需要修改 `trie_store.h` 和 `trie_store.cpp`。此键值存储还支持 3 种操作：

- `Get(key)`：返回对应 `key` 的值。
- `Put(key, value)`：无返回值。
- `Delete(key)`：无返回值。

对于原始的 Trie 类，每次修改 trie 时，都需要获取新的根来访问新的内容。但对于并发键值存储，`put` 和 `delete` 方法没有返回值。这要求您使用并发控制来同步读取和写入，以便在此过程中不会丢失任何数据。

您的并发键值存储应同时为**多个读者和单个写者**提供服务。也就是说，当有人修改 trie 时，仍然可以在旧根上执行读取操作。当有人正在读取时，仍然可以执行写入，而无需等待读取。

此外，如果我们从 trie 获得对值的引用，那么无论我们如何修改 trie，我们都应该能够访问它。`Trie` 的 `Get` 函数仅返回一个指针。如果存储此值的 trie 节点已被删除，则指针将悬空。因此，在 `TrieStore` 中，我们返回一个 `ValueGuard` ，它同时存储对值的引用和对应于 trie 结构根的 TrieNode，以便可以访问值在我们存储 `ValueGuard` 的时候。

为此，我们在 `trie_store.cpp` 为您提供了 `TrieStore::Get` 的伪代码。请仔细阅读并考虑如何实现 `TrieStore::Put` 和 `TrieStore::Remove` 。

正确完成上面的代码，你可以得到所有的分数🎉。

环境配置

我们推荐您在Linux环境下完成这道练习。您需要配置好C++17的环境，并安装必要的库：

```
sudo apt install libfmt-dev   # Ubuntu/Debian
sudo dnf install fmt-devel   # Fedora
```

如果您选择完成配置原版实验的环境并克隆了原版实验，请注意原版框架和CPP作业框架的差别。

测试

由于CPPOJ不支持使用GTest进行单元测试，我们编写了 `SimpleGTest.h` 来测试大家的代码。您可以在标准输入中输入对应的函数名来进行自测，例如：TrieTest_ConstructorTest。

同原版实验一样，在这个项目中没有隐藏的测试用例。但与原版项目不同的是，所有测试用例均编写在 `Main.cpp` 文件中。

格式化与内存泄漏

由于CPPOJ的评测限制，我们不会的大家的格式化和内存泄漏进行测试。但是你写的程序应该尽可能满足下列要求：

- 您的代码应当遵循 **Google C++ 样式指南**。您可以使用 **Clang** 自动检查您的源代码的质量。
- 您的代码不应该存在内存泄漏。您可以使用 **LLVM 地址清理器 （ASAN） 和泄漏清理器 （LSAN）** 来检查内存错误。

提交

你需要在CPPOJ提交的代码文件有： `trie.h` ， `trie.cpp` ， `trie_store.h` 和 `trie_store.cpp` 。其余文件OJ在测试时会自动填充。

其他要求

- 每个学生都必须单独完成此作业。
- 学生可以与他人讨论有关该项目的高级细节。
- 学生**不得**复制他人的解决方案。
- 在这个项目中，您可以在 Google 上搜索或询问 ChatGPT 高级问题，例如“什么是 trie”， “如何使用 `std::move` ”。
- 请**认真阅读框架代码中的注释**，对于题目的疑问请尽量在课程QQ群中提出，不要私发助教。
- 根据CMU15445的课程要求，**即使在课程结束后，你也不要将代码提交到任何公开的仓库中。**

> **警告：**此项目的所有代码都必须是您自己的代码。您不得从其他学生或您在 Web 上找到的其他来源复制源代码。剽窃是**不能**容忍的。 ——翻译自CMU15445项目组