Лабораторная работа № 2 по курсу дискртного анализа: словарь

Выполнил студент группы 08-208 МАИ *Куликов Алексей*.

Условие

Кратко описывается задача:

1. Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64} - 1$. При этом разным словам может быть поставлен в соответствие один и тот же номер.

2. Вариант задания: 1. Реализация с использованием АВЛ-дерева.

Метод решения

АВЛ-дерево – это бинарное дерево поиска, близкое к идеально сбалансированному из-за того, что для любого узла выполнятся условие: модуль разности высот левого и правого поддеревьев не боьльше единицы.

Для того чтобы дерево было сбалансированым его нужно балансировать. В АВЛ-дереве это происходит на основе баланс-фактора.

Баланс фактор узла – это разница высот правого и левого педдеревьев этого узла. Возможны два пути: хранить сам баланс-фактор либо хранить высоты поддеревьев и вычислять баланс-фактор по мере надобности.

Для балансировки применяется две вспомогательных процедуры: левый и правый повороты дерева. Через них можно выразить так же и большие левый и правый повороты дерева, описанные в литературе.

Вставка осуществляется так же как и в простое бинарное дерева, но после этого происходит процедура балансировка.

Удаление как и в обычном бинарном дереве, но далее следует балансировка дерева.

Поиск аналогичен поиску в бинарном дереве.

Далее реализованное АВЛ-дерево используется в качетве словаря, над которым производятся операции добавления, удаления, поиска, записи в файл или чтения из файла в зависимости от команд пользователя.

Словарь хранит пары ключ-значение. Ключом выступает строка (не более 256 значащих символов), значением беззнаковое 8-ми байтовое целое.

Для удобства ввод/вывода и обработки реализовано нечто, похожее на std::string.

Используемые источники:

Книги:

1. Д. Кнут, Искусство программирования, том 3, Сортировка и поиск.
2. Н. Вирт, Алгоритмы + структуры данных = программы.

Электронные источники:

1. https://habr.com/post/150732/
2. https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево
3. https://rsdn.org/article/alg/bintree/avl.xml

Общее описание алгоритма решения задачи, архитектуры программы и т. п. Полностью расписывать алгоритмы необязательно, но в общих чертах описать нужно. Приветствуются ссылки на внешние источники, использованные при подготовке (книги, интернет-ресурсы).

Описание программы

Программа состоит из 3-х файлов.

В первом (avl.hpp) реализован класс АВЛ-дерева и присущие ему методы вставки, удаления, поиска, вспомогательные скрытые методы для балансировки, поиска наименьшего элемента и т д. Также реализованы дополнительные методы: для ввода и вывода дерева в поток и из потока.

Название методов класса говорит само за себя. Все функции еализованы согласно определениям и алгоритмам, описанным в литературе по алгоритмам.

Во втором (custom_string.hpp) самодельный класс строк, кое-как реализующий функциональность, похожую на std::string.

В начале работы программы выделяется буффер, через который будет проходить весь ввод строк из потока. В данной задаче нам нужно всего 256 символов. Далее символы по одному считываются в буфер. Считав все символы до символа-разделителя функция «знает» размер будущей строки. Далее если строке уже была приписана память, она освобождается, выделяется блок памяти с новым размером и туда копируются считанные в буфер символы. В конец принудительно вставляется нулевой символ.

В конце работы программы, выделенный для работы со строками буффер освобождается.

Так же реализованы операторы конкатенации, сравнения и т. д. Где это возможно использована move-семантика. Остальные методы строк представляют меньший интерес. (См. код custom_string.hpp).

В третьем (main.cpp) описана основная логика работы программы: «общение» с пользователем, чтение/запись словаря в файл и обработка ошибок.

Дневник отладки

1. 13.10 16:00. В процессе тестирования АВЛ-дерева возникла проблема с удалением элементов из него и его последующей балансировкой. Почему-то казалось что балансировать нужно только родителя удаленного узла. РЕШЕНИЕ: Возвращение к изучению литературы. Корректировка кода для балансированиявсего дерева после удаления узла.(См. код avl.hpp)

2. 14.10 10:00. Т.к. необходимо подтверждать успешность операций вставки, удаления, и поиска были предприняты попытки «пробросить» результат операции наверх при обратном ходе рекурсии. Они не увенчались успехом. РЕШЕНИЕ: Было принято решение сначала искать элемент и уже на основе информации о наличии или отсутствии осуществлять вставку/удаление.(См. код avl.hpp)

3. 24.10 17:30. После долгого и упорного тестирования и поиска проблемы из-за который не проходился 4-й тест она была найдена. Из-за неполного удаления кода, реализуещего идею «проброса» получалось так, что после удаления любого элемента корень дерева становился nullptr. Таким образом любое удаление удалаяло все дерево. РЕШЕНИЕ: Удаление остатков кода для «проброса» наверх результата операции.(См. код avl.hpp)

4. 25.10 17:30. Проблемы с производительностью программы. Не проходится 14 тест с превышением временем выполнения. РЕШЕНИЕ: были оптимизированы некоторые участки кода, но остались медленные участки кода, которые не поддались.

5. 25.10 17:30. Утечки памяти при удалении элементов по одному. Оказалось что была перепутана последовательность операций удаления и возврата значения из функции удаления. РЕШЕНИЕ: Замена метсами строчек кода(См. код avl.hpp)

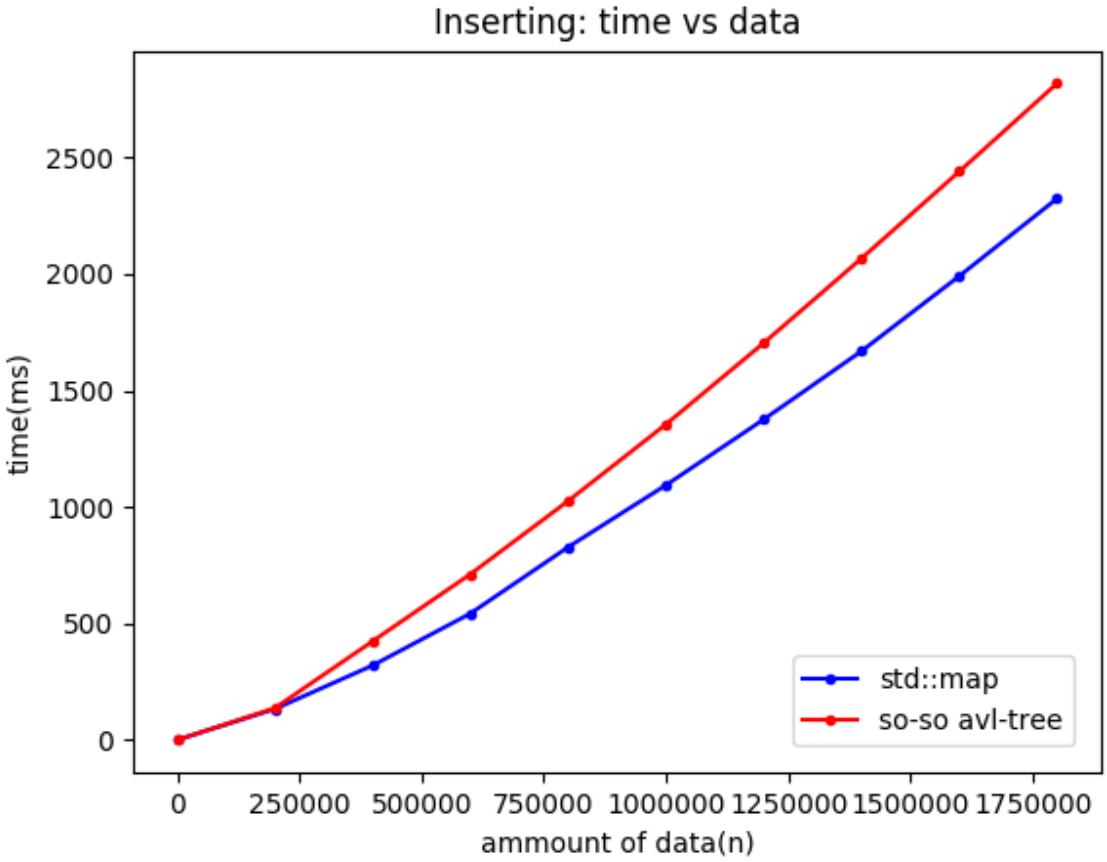
Тест производительности

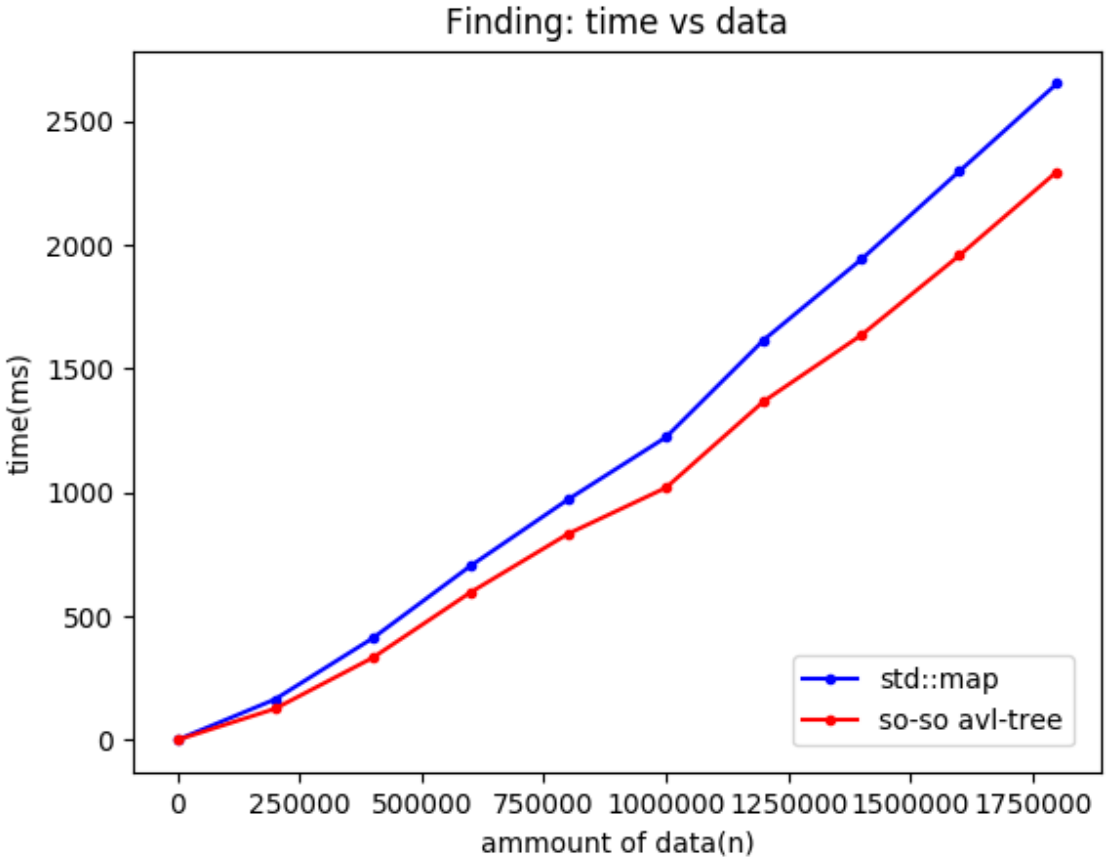
Созданное в результате выполнения АВЛ-дерево сравнивалось со стандартным контейнером `std::map`. Это имеет смысл так как в его основе лежит красно-черное дерево, которое тоже является сбалансированным. Замеры производительности проводились на операциях вставки/поиска/удаления 4-х байтовых целочисленных значений.

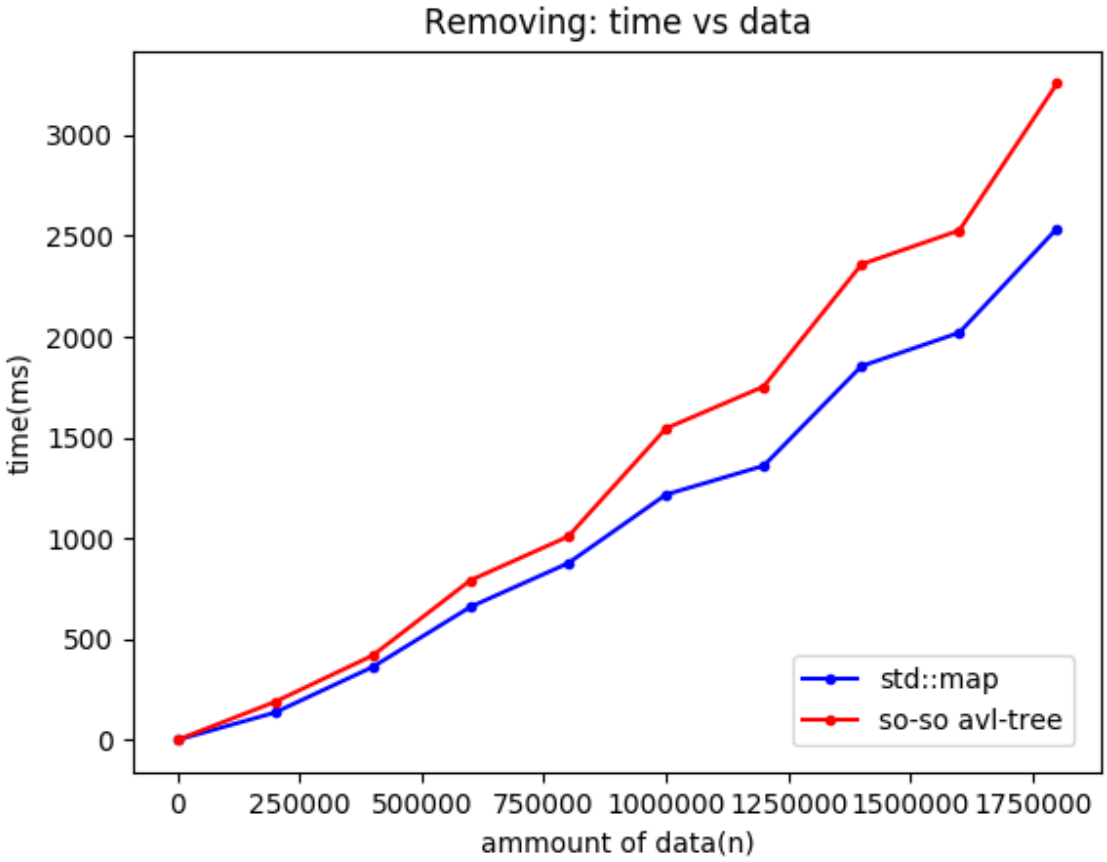
Из графиков видно, что, как и ожидалось, АВЛ-дерево проигрывает красно-черному в операциях вставки/удаления и выигрывает на операции поиска.

Это происходит в силу различий в алгоритмах балансировки: в красно-черных деревьях она происходит гораздо реже, чем в АВЛ. Как следствие вставка и удаление происходит быстрее Но при этом и высота красно-черного дерева больше высоты соответствующего АВЛ дерева. Отсюда вытекает большее время поиска.

Также, несомненно, повлияло то, что дерево было написано мной, а не нормальным программистом.


Inserting: time vs data
time(ms)
2500
2000
1500
1000
500
0
0
250000
500000
750000
1000000
1250000
1500000
1750000
ammount of data(n)
std::map
so-so avl-tree


Finding: time vs data
time(ms)
2500
2000
1500
1000
500
0
0
250000
500000
750000
1000000
1250000
1500000
1750000
ammount of data(n)
std::map
so-so avl-tree



Не логарифмическая зависимость на графиках, по моему мнению, получается из-за довольно частых выделений/освобождений памяти на кучи, а это, как известно, не дешевая в плане времени операция. К тому же графики имеют очень похожую форму, что говорит об асимптотически одинаковой скорости роста времени выполнения операций. А уж в `stl` сомневаться не приходится.

Недочёты

Программа работает не так быстро, как хотелось бы, хотя и прошла тестирование на чекере.

Узким местом, как выясняется, является пользовательский класс строк, что не удивительно. А именно считывание строки из потока. Из-за лишнних операций копирования и системных вызовов для выделения/возврата памяти программа замедляется.

Как решение можно предложить «подглядеть» как работают более удачные реализации строк и переделать/доделать свои строки с подобных методов. Исправление нецелесообразно потому что лабораторная работа немного не про это.

Выводы

Данная стркутура данных может быть применена как вспомогательная для решения большей задачи, например, в качестве словаря, множества, мультимножества(с доработками) и подобных этим абстрактным типам данных. Так же, весьма вероятно, в реализации простейших БД.

Первый раз реализовывать данную структуру было довольно сложно. Возможно, при повторном написании, получится сделать это быстрее и алгоритм получится оптимальнее.

Основные проблемы при написании возникли при леализации удаления из AVL-дерева из-за неполного понимания. Почему-то думалось, что балансировать нужно только дерево-родитель удаляемого узла, а, как оказалось, это не так.

Описать область применения реализованного алгоритма. Указать типовые задачи, решаемые им. Оценить сложность программирования, кратко описать возникшие проблемы при решении задачи.