Лабораторная работы № 2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы 08-210 МАИ *Некрасов Константин*

Условие

Реализовать декартово дерево с возможностью поиска, добавления и удаления элементов.

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 $2^{64} - 1$. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

Команд **! Save** и **! Load** в тестах нет.

Метод решения

Декартово дерево – структура данных, объединяющая в себе свойства дерева поиска и кучи. В каждом узле декартова дерева содержатся ключ и приоритет, при этом ключи узлов удовлетворяют условию двоичного дерева поиска, а приоритеты узлов формируют двоичную кучу. В основе реализации операций вставки и удаления лежат операции `Merge` и `Split`.

`Merge` предназначен для слияния двух деревьев, и это слияния производится рекурсивно по сравнению приоритетов узлов.

`Split` предназначен для «разрезания» дерева по конкретному ключу. В декартовом дереве ключ и приоритет обычно обозначают как x и y соответственно, и если представить дерево изображенным на координатной оси, то `Split` проведёт линию по значению x и разделит исходное дерево на два новых с сохранением свойств.

Поиск элемента в дереве реализован в операции `Search`, но запрашиваемый вывод осуществляется в `Find`. Поиск реализован также, как в двоичном дереве поиска, то есть сравниваются значения ключей. Поскольку ключом в случае с заданием является строка, сравнение производится с помощью функции `strcmp`.

Вставка элемента реализована в `Insert`. Дерево разрезается на два по ключу вставляемого узла и в правом полученном дерево смотрим равняется ли минимальный ключ вставляемому. Если равняется, выводим сообщение о существовании узла и восстанавливаем дерево слиянием. Если не равняется, сливаем вставляемый узел с ключом с левым поддеревом, и полученное дерево сливаем с правым.

Удаление элемента реализовано в `Remove`. Дерево разрезается на два по ключу удаляемого узла, после этого значение ключа модифицируется увеличением на 1 и правое дерево разрезается по модифицированному значению ключа. В итоге, если такой узел существовал, то он является всем левым поддеревом разрезанного правого, поэтому оно просто удаляется, а левое поддерево изначального сливается с правым поддеревом правого. Если такой узел не существовал, дерево восстанавливается и выводится соответствующее сообщение.

Описание программ

Помимо описания структуры декартова дерева и основной функции main, также была сделана функция `ToLower`, которая понижает регистры латинских букв до нижнего.

Дневник отладки

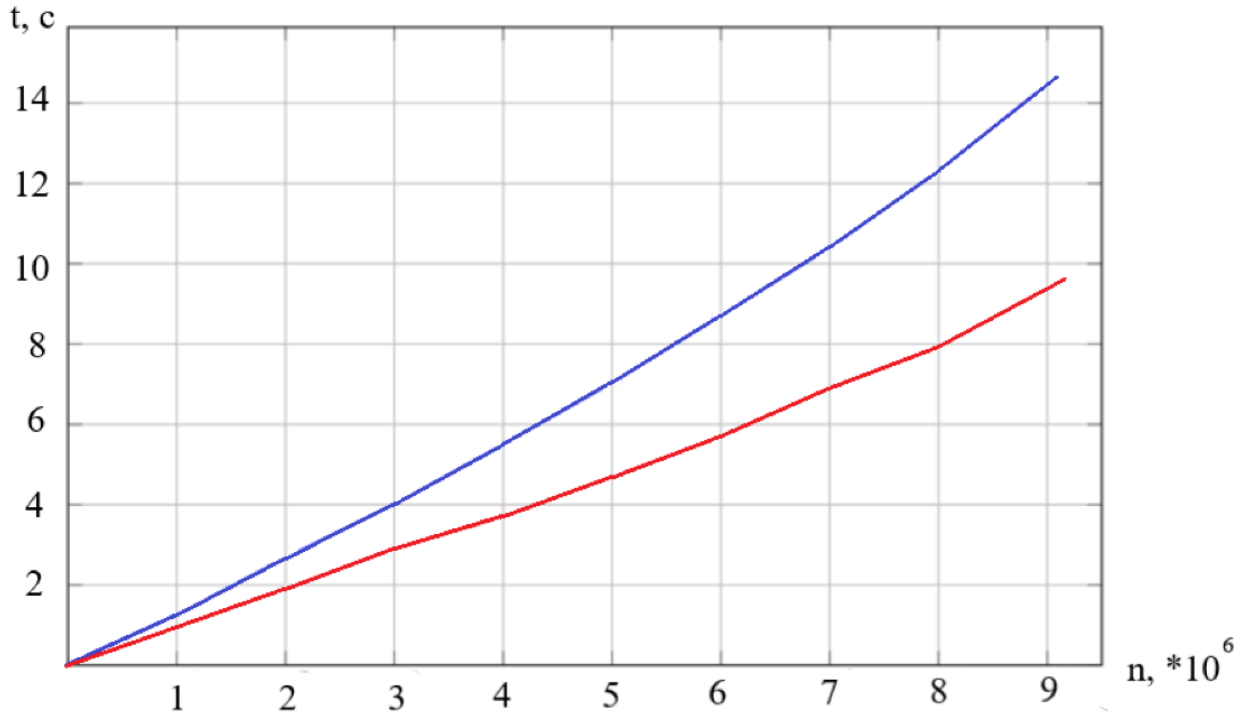
1. В первой посылке была глупая ошибка – случайно был выбран компилятор C, а не C++.
2. Вторая посылка превысила лимит времени на 16 тесте.
3. На третьей посылке было добавлено отключение синхронизации с `stdio` и оптимизация ввода за счёт отключения связности с выводом посредством `cin.tie(nullptr)`. Это не помогло пройти 16 тест.

4. В процессе исправления ошибок кода, код, который проходил 15 тестов, просто перестал работать, по этой причине он был полностью переписан с нуля, с заменой класса на структуру, изменением поиска минимального элемента, присвоением приоритету псевдослучайного значения `rand()` и изменением порядка некоторых операций. В результате, код прошёл все тесты.
5. Посылка также полностью успешна, но была добавлена оптимизация как в посылке 3 просто ради интереса.

Тест производительности

Оценка сложности поиска, вставки и удаления – O(h), где h – высота дерева. Дерево является сбалансированным, поэтому сложности операций можно представить как $O(\log n)$, где n – количество элементов в дереве. Поэтому сложность всей программы оценивается как $O(n * \log n)$. Для сравнения используем `std::map`, который реализован на красно-чёрном дереве.

Синим обозначено декартово дерево, красным – красно-чёрное.



Недочёты

Не удалось понять почему прежде работающий код просто перестал работать без внесения каких-либо изменений, но в остальном задача была выполнена.

Выводы

В данной работе было реализовано декартово дерево с операциями поиска, вставки и удаления, вместе с основополагающими операциями декартова дерева в лице `merge` и `split`.