Лабораторная работа № 2 по курсу дискретного анализа: Сбаласированные деревья

Выполнил студент группы М8О-208Б-20 МАИ *Каширин Кирилл.*

Условие

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.

+ 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».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный параметром команды. В случае успеха, программа должна вывести «OK».

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа).

Вариант задания: B-дерево.

Метод решения

Для создания программной библиотеки была создана структура dictPair, состоящая из строки и числа, которая хранила в себе пару «ключ-значение». Также был создан отдельный класс BNode для узла B-дерево, который состоит из массивая ключей и массива дочерних узлов, булевской переменной, указывающей является ли узел листом дерева и числом size, который обознал количество ключей в узле. Был создан класс дерева. У класса B-дерева были реализованы: функции поиска, вставки, слияния узлов дерева, удаления узла и самого дерева, сохранения дерева в бинарном формате и загрузка дерева из файла.

Методы класса B-дерева:

- *void Insert(DictPair value)* - вставка нового элемента.
- *void InsertNode(BNode* node, DictPair value)* - вспомогательная функция, позволяющая вставлять элемент в конкретной ноде.
- *int BinarySearchInNode(BNode* node, DictPair value)* - - бинарный поиск для вставки элементе в ноде. item *void SplitChild(BNode* node, int idx)* - метод для разбиения вершины в случае насыщения.
- *void DeleteFromNode(BNode* node, int pos)* - удаление элемента из конкретной ноды.
- *void Deleting(BNode* node,string value)* - удаление элемента из дерева.
- *void SearchNode(BNode* node, string cmd, BNode* result, int pos)* - метод для поиска элемента в дереве, возвращает ноду и позицию элемента в этой ноде.
- *void SaveFile(BNode *node, ofstream file)* - сохранение дерева в бинарный файл.
- *void LoadFile(BNode* node, std::ifstream file)* - загрузка дерева из бинарного файла.
- *void DeleteTree(TNode* cur)* - удаление дерева.

Описание программы

Программа состоит из одного файла, но разделена на три части: описание структуры dictPair и перегрузка операторов сравнения, описание узла дерева и само дерева и его основные функции, в главной функции описан интерфейс взаимодействия пользователя с библиотекой согласно условию задания.

Дневник отладки

1. Когда первый раз была загружена программа на чекер, она получила Runtime Error. Позже выяснилось, что это из-за неправильной функции удаления, а именно неправильное слияние узлов при удалении. Неверно был обработан случай, когда мы должны сделать слияние корня и его дочерних узлов, терялся указатель на одного из дочерних узлов.
2. После долгих попыток, программа дошла до 7 теста. Пробелма заключалась в некорректной сохранения в файл и загрузки из файла типа string. Она решалась тем, что перед тем, как записывать в файл строку, я преобразовывал её в массив типа char*.
3. После этого, программа дошла до 11 теста, в котором был неправильной ответ из-за неверного преобразования массива в строку при загрузки дерева из файла.

Тест производительности

Для анализа производительности была написала программа на основе контейнера std::map стандартной библиотеки (представляет собой словарь основанный на красно-черном дереве). Для сравнения производительности я подготовил 4 файла в который 1000, 10000, 100000, 500000 строк с разными командами, в основе которых лежит операции добавления, поиска и удаления.
Получились следующие результаты (B-tree || std::map) :

Добавление:

1000 входных данных - 20 ms || 41 ms
10000 входных данных - 147 ms || 463 ms
100000 входных данных - 5428 ms || 7359 ms
500000 входных данных - 6125 ms || 9878 ms

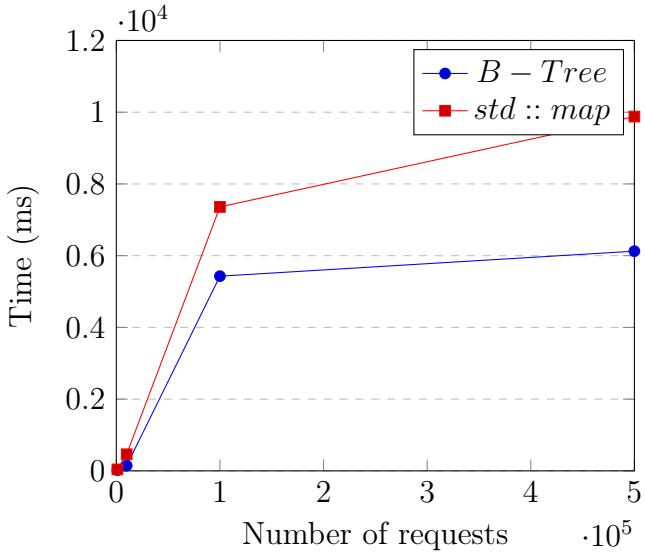
Поиск:

1000 входных данных - 23 ms || 35 ms
10000 входных данных - 567 ms || 2390 ms
100000 входных данных - 5207 ms || 8511 ms
500000 входных данных - 6310 ms || 47180 ms

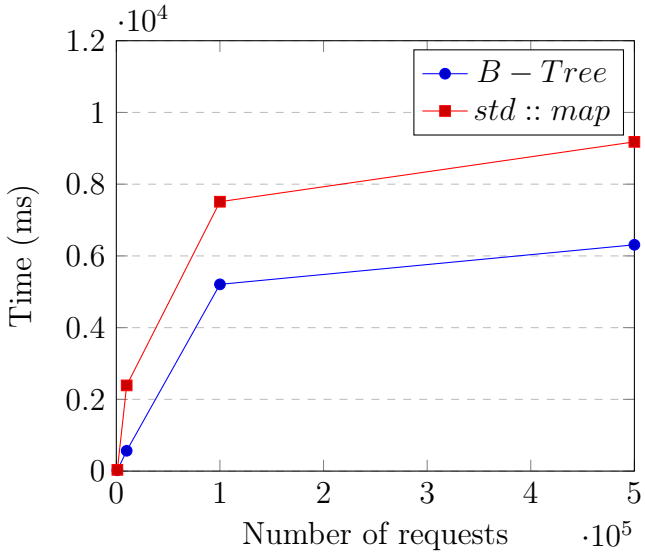
Удаление:

1000 входных данных - 19 ms || 19 ms
10000 входных данных - 933 ms || 618 ms
100000 входных данных - 6995 ms || 4239 ms
500000 входных данных - 7145 ms || 25625 ms
По оси Y отложено время выполнения (в миллисекундах), по оси X — количество входных данных.

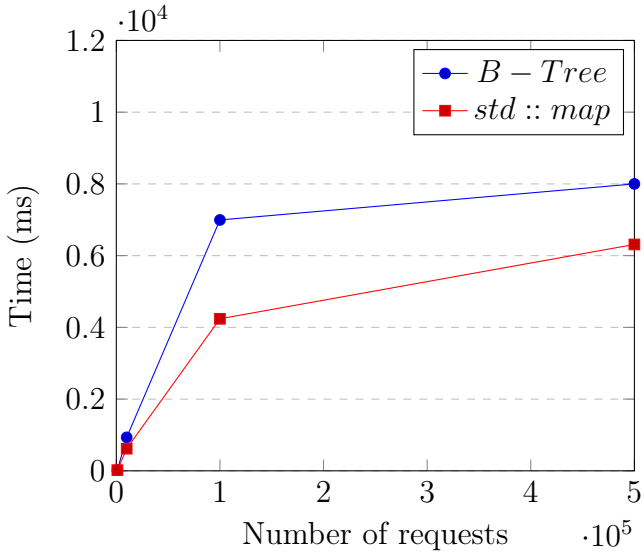
Добавление



Поиск



Удаление



Таким образом, можно заметить, что при операции вставка и поиск работают быстрее у B-дерева быстрее в 1,5 раза, однако операция удаления быстрее происходит у std::map. Это может быть связано с тем, что степень узла у B-дерева выбрана 2 и постоянно приходится делать слияние узлов B-дерева.

Недочёты

Основным недочетом заключается во времени исполнения программы, поскольку в некоторых случаях удаления дерева, происходит лишний проход по дереву при поиске элемента, а также возможны ненужные рекурсивные вызовы в некоторых вариантов удаления.

Выводы

В результате этой лабораторной работы я реализовал программную библиотеку, реализующую структуру данных, такую как B-дерево. Этот вид стркутуры данных имеет существенные преимущества перед альтернативными реализациями в случае, если время доступа к узлу превышает время, которое затрачивается на обработку этот узла. Это происходит, если данные находятся на дисковом пространстве. В таком случае B-дерево идеально подходит для обработки этих данных. За счет того, что увеличивается количество ключей в каждом внутреннем узле, уменьшается высота дерева, количество дорогостоящих обращений к узлам и реже происходит перебалансировка дерева.