Лабораторная работа № 2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы М8О-207Б-21 Меркулов Фёдор.

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. 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».
! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).
! 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); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Различия вариантов заключаются только в используемых структурах данных:

B-дерево

Метод решения

Для решения задачи была создана структура Element для хранения пары «ключ-значение» (строка и число). Для реализации B-дерева были созданы классы TreeNode и Tree. Класс TreeNode представляет узел дерева, который состоит из булевской переменной, указывающей является ли узел листом дерева, числа - количества ключей в узле, указателей на «ключ-значение» и дочерние узлы, минимальную степень дерева. Класс Tree представляет структуру дерева, состоящую из

корня и минимальной степени дерева. Также для решения задачи потребовалось реализовать собственный тип данных string, а также встроенные функции strlen, strcpy, strcmp.

Описание программы

Программа состоит из одного файла ДА2.cpp. В файле хранится реализация типа данных string, реализация необходимых структур, основная часть кода.

Основные использованные функции:

Название функции	Описание функции	Временная сложность
char* Copy(char* string_left, const char* string_right)	Собственная реализация функции strcpy (копирование строки) из библиотеки cstring	O(n), где n – длина строки
int Length(const char* line)	Собственная реализация функции strlen (определение длины строки) из библиотеки cstring	O(n), где n – длина строки
static int Lexis(const char* string_left, const char* string_right)	Собственная реализация функции strcmp (лексикографическое сравнение строк) из библиотеки cstring	O(n), где n – минимальная длина строки
Element* Search(const String& string)	Поиск элемента	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
void DeleteNode()	Удаление узла	O(t * n), где t - минимальная степень B-дерева, n – количество узлов дерева
void InsertForNoFull (const Element& element)	Вставка ключа в дерево с незаполненным корнем	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l –

		минимальная длина строки
void SplitChild(int index)	Разделение узла при равенстве его размера критическому (2 * value – 1)	O(t), где t – минимальная степень B-дерева
void Delete_leaf(int index)	Удаление листа	O(n), где n – количество ключей узла
void DeleteInnerNode (int index)	Удаление внутреннего узла	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
void Balance(int index)	Выполнение преобразований для увеличения размера (слияние или перемещение вершин), если при удалении встретилась вершина с минимальным размером	O(t), где t – минимальная степень B-дерева
void Delete(const String &string)	Удаление по заданному ключу	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
void Merge(int index)	Слияние узлов	O(t), где t – минимальная степень B-дерева
void GetFromLeftChild(int index)	Перемещение из левого ребенка при балансировке дерева	O(t), где t – минимальная степень B-дерева
void GetFromRightChild(int index)	Перемещение из правого ребенка при балансировке дерева	O(t), где t – минимальная степень B-дерева

TreeNode* FindSuccessor(int index)	Поиск минимального ребенка в левом поддереве	O(h), где h – высота дерева
TreeNode* FindPredecessor(int index)	Поиск максимального ребенка в правом поддереве	O(h), где h – высота дерева
bool Read(FILE* file)	Считывание дерева из бинарного файла	O(n), где n – количество узлов дерева
bool Write(FILE* file)	Загрузка дерева в бинарный файл	O(n), где n – количество узлов дерева
Element* Search(const String& string)	Поиск элемента в дереве по ключу	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
void Delete(const String& string)	Удаление элемента в дереве по ключу	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
void Insert(Element& element)	Вставка элемента в дерево	O(t * h * l) = O(t * $\log_t n * l$), где t - минимальная степень B-дерева, h – высота дерева, n – количество узлов дерева, l – минимальная длина строки
bool Read(const char* path)	Считывание дерева из бинарного файла	O(n), где n – количество узлов дерева
bool Write(const char* path)	Загрузка дерева в бинарный файл	O(n), где n – количество узлов дерева
void DeleteTree()	Удаление дерева	O(t * h) = O(t * $\log_t n$), где t - минимальная

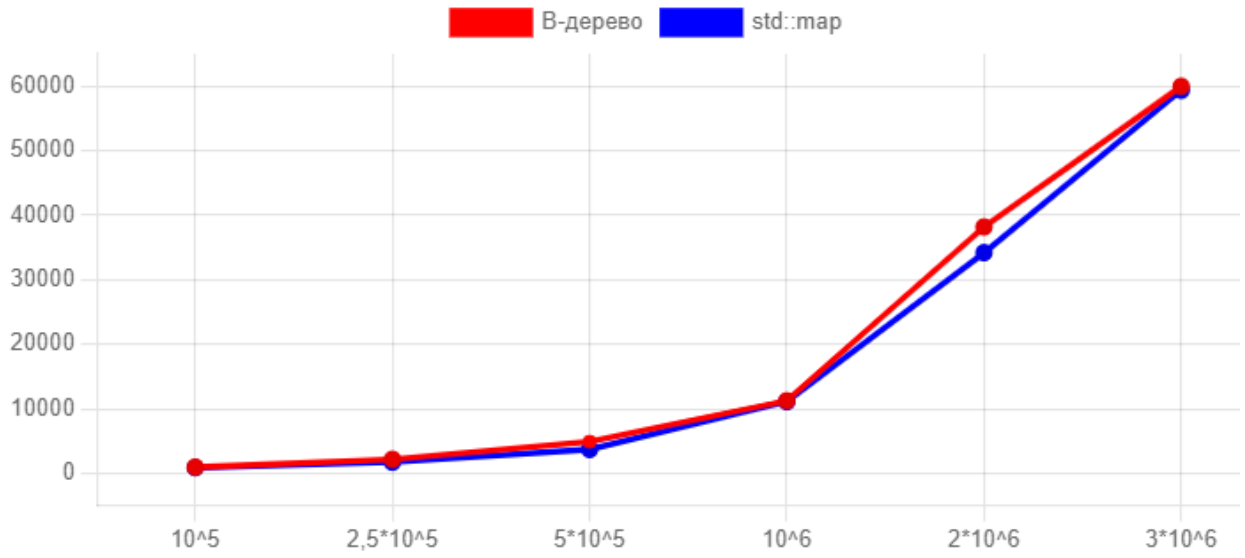
		степень B-дерева, h – высота дерева, n – количество узлов дерева

Дневник отладки

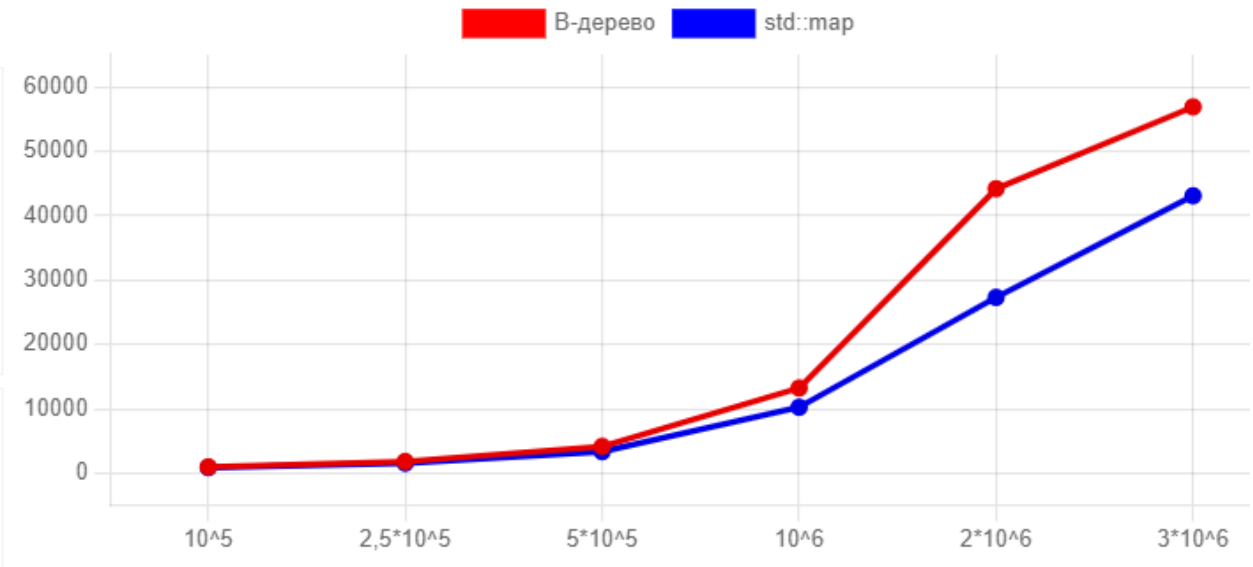
Основной проблемой была неправильная работа функций Read и Write. Я пытался соответствовать теоретическому заданию, но в итоге оказалось, что в случаях неудачи не стоит сообщать об ошибке, а также стоит выводить сообщение “OK\n”.

Тест производительности

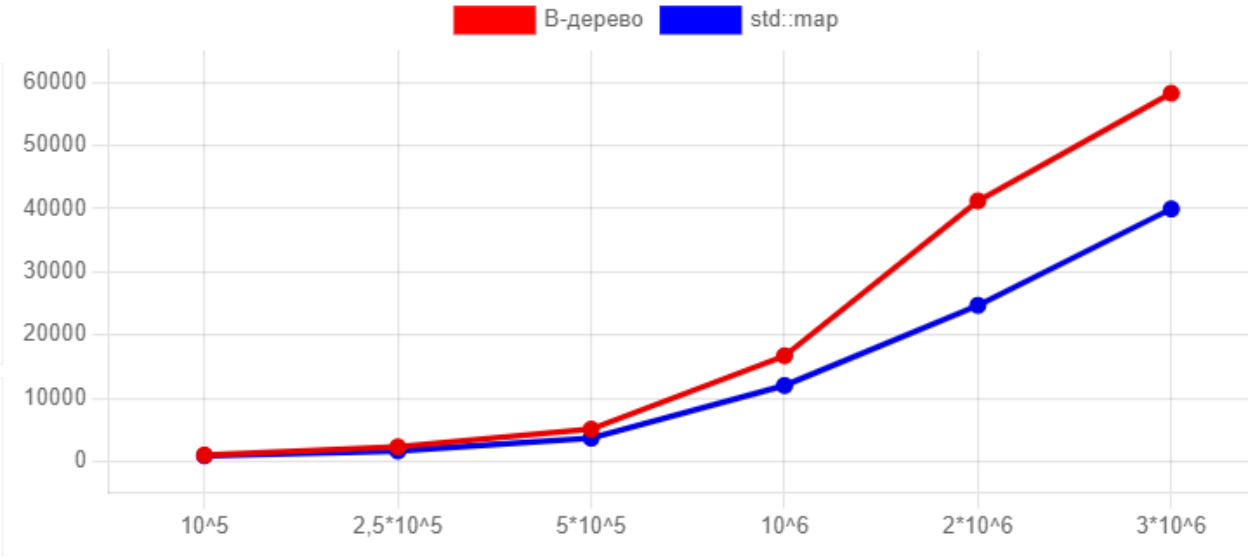
1. Вставка



2. Поиск



3. Удаление



Данные замеры включают также и ввод элементов, поэтому сложность этих действий $O(nlogn)$ (ввод + операции над деревом)

Выводы

В ходе выполнения лабораторной работы изучил и реализовал структуру данных B-дерево.

B-дерево достаточно удобно и полезно использовать при работе с жёстким диском, то есть для структурирования информации на нём. Данная структура позволяет уменьшить количество узлов, просматриваемых при каждой операции, что важно при работе с диском. Операции вставки, удаления и поиска элементов выполняются за $O(logn)$ (временная сложность).