

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной
математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Г. А. Ермаков
Преподаватель: С. А. Михайлова
Группа: М8О-201Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №1

Задача: 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.

Структура данных: В-дерево.

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 параметром 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, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

1 Описание

Основная идея алгоритма решения задачи состоит в использовании В-дерева для эффективного хранения и управления словарём, где ключами являются регистроне-зависимые слова, а значениями — соответствующие номера. В-дерево обеспечивает быстрый поиск, вставку и удаление элементов за счёт сбалансированной структуры и оптимизации операций ввода-вывода, что особенно важно при работе с большими объёмами данных. Для реализации необходимо разработать В-дерево с поддержкой операций добавления, удаления и поиска, а также методов Save и Load для сохранения и загрузки словаря в/из бинарного файла.

2 Исходный код

btree.cpp	
string to_lower(const string &o)	Преобразование входной строки к нижнему регистру.
BNode(bool leaf_)	Конструктор узла В-дерева, устанавливает флаг листа и нулевое число ключей.
void BNode::insertNotNull(const string &k, uint64_t v)	Вставка пары «ключ–значение» в узел (листьевой или внутренний) без проверки корня на переполнение.
bool BNode::remove(const string &key)	Удаление ключа из поддерева с последующей балансировкой (слияние, перераспределение).
pair<bool, uint64_t> BNode::search(const string &k)	Бинарный поиск ключа в узле и рекурсивный спуск в потомки при необходимости.
class BTree	Внешний интерфейс В-дерева: методы add, remove, search, dump, load.
int main()	Цикл чтения команд из stdin, разбор и выполнение операций над деревом.

```

1  const int T = 64;
2
3  class BNode {
4  public:
5      bool leaf;
6      int c;
7      string keys[2 * T - 1];
8      uint64_t values[2 * T - 1];
9      BNode* children[2 * T];
10
11     BNode(bool leaf_);
12     ~BNode();
13
14     pair<bool, uint64_t> search(const string &k);
15     void insertNotNull(const string &k, uint64_t v);
16     void split(int index);
17     bool remove(const string &key);
18 };
19
20 class BTree {
21 public:
22     BNode* root;

```

```

23
24     BTree();
25     ~BTree();
26
27     bool add(const string &word, uint64_t val);
28     bool remove(const string &v);
29     pair<bool, uint64_t> search(const string &word);
30     bool dump(const string &filename, string &errmsg);
31     bool load(const string &fname, string &errmsg);
32 };
33
34 int main();

```

3 Консоль

```
george@GEORGE-PC:/mnt/c/Users/george/Projects/MaiLabs/MAI_labs_Discran/src$  
./main  
+ a 1  
+ A 2  
+ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
18446744073709551615  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
A  
-A  
a  
OK  
Exist  
OK  
OK: 18446744073709551615  
OK: 1  
OK  
NoSuchWord
```

4 Тест производительности

Тест производительности представляет из себя следующее: в В-дерево и в `std::map` последовательно вставляются 1.000.000 пар «ключ–значение», затем выполняется 100.000 операций поиска случайных ключей и 100.000 операций удаления. Все измерения проводятся в одной программе, выводятся отдельно для каждой фазы.

Benchmarking B-Tree implementation:

```
george@GEORGE-PC:/home/george/Projects/MaiLabs/MAI_labs_Discran/src$ g++ btree.cpp
main.cpp -std=c++17 -O2 -o bench_btree
george@GEORGE-PC:/home/george/Projects/MaiLabs/MAI_labs_Discran/src$ ./bench_btree
B-Tree insertion time: 0.752341 sec
B-Tree search time:    0.048912 sec
B-Tree deletion time: 0.603127 sec
```

Benchmarking `std::map`:

```
george@GEORGE-PC:/home/george/Projects/MaiLabs/MAI_labs_Discran/src$ g++ main_map.cpp
-std=c++17 -O2 -o bench_map
george@GEORGE-PC:/home/george/Projects/MaiLabs/MAI_labs_Discran/src$ ./bench_map
std::map insertion time: 1.127589 sec
std::map search time:    0.081204 sec
std::map deletion time: 0.898432 sec
```

Как видно, В-дерево опережает ‘`std::map`’ по времени вставки (0.75с против 1.13с) и удаления (0.60с против 0.90с), а также чуть быстрее при поиске (0.05с против 0.08с), что демонстрирует преимущество сбалансированной структуры при больших объёмах данных.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучил В-деревья и научился использовать их для хранения больших объемов данных.

Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание*. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))
- [2] *В-дерево* — *Википедия*.
URL: <https://ru.wikipedia.org/wiki/В-дерево> (дата обращения: 20.05.2025).
- [3] *Структура данных В-дерево* — *Хабр*.
URL: <https://habr.com/ru/companies/otus/articles/459216/> (дата обращения: 20.05.2025).