Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 2

по курсу «Дискретный анализ»

Студент:	Королев И.М.
Группа:	М8О-208Б-19
Преподаватель:	Капралов Н.С.
Дата:	
Оценка:	

Москва 2020

1. Постановка задачи

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

! 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.

Вариант структуры данных: Красно-чёрное дерево

2. Описание программы

Для выполнения лабораторной работы был создан класс узла дерева Node, в котором хранились ключ, значение и указатели на левый и правый дочерние узлы, а также на родительский узел. Ключ представляет собой регистронезависимую последовательность букв английского алфавита длиной не более 256 символов. Значение представляет собой число от 0 до $2^{64} - 1$. Был создан класс дерева RBTree, который хранит в себе указатель на вершину дерева и указатель на ограничитель TNull. Этот ограничитель является родительским узлом для вершины дерева, а также он является дочерним узлом для тех узлов, у которых либо нет потомков, либо он один.

Выполняются все свойства красно-чёрного дерева:

1. Каждый узел является либо красным, либо чёрным.
2. Корень дерева является чёрным узлом.
3. Каждый лист дерева (NIL) является чёрным узлом.
4. Если узел красный, то оба его дочерних узла чёрные.
5. Для каждого узла все простые пути от него до листьев, являющихся его потомками, содержат одно и то же количество чёрных узлов.

Для работы с красно-чёрным деревом были реализованы методы:

1. Вставка
2. Удаление
3. Поиск

Вставка узла в красно-чёрное дерево с n узлами может быть выполнена за время $O(\lg(n))$. При вставке сравниваются два ключа, если ключ вставляемого узла меньше, чем у рассматриваемого ключа в дереве, то программа опускается в левый дочерний узел, иначе – в правый. Если оказывается, что ключи сравниваемых узлов равны, то программа выводит сообщение, что элемент с указанным ключом в дереве уже существует. После вставки нового узла в дерево, он перекрашивается в красный цвет. После вставки выполняется функция проверки дерева на соответствие его свойствам. Если функция находит несоответствие, она его исправляет.

Удаление как и вставка в красно-чёрном дереве выполняется за $O(\lg(n))$. Удаление оказывается более сложной задачей, чем вставка. При удалении узла также дерево проверяется на соответствие свойствам его типа.

Поиск ничем не отличается от поиска в бинарном дереве. Он также выполняется за время $O(\lg(n))$.

3. Исходный код

```
typedef enum { BLACK, RED } NodeColor;

template<class T1, class T2>
class Node {
private:
    T1 Key;
    T2 Value;
    Node* Left;
    Node* Right;
    Node* Parent;
    NodeColor Color;
public:
    Node();
    T1 FindKey();
    void GetValue(T2 value);
    T2 FindValue();
    void GetLeft(Node* node);
    Node* FindLeft();
    void GetRight(Node* node);
    Node* FindRight();
    void GetParent(Node* parent);
    Node* FindParent();
    void GetColor(NodeColor color);
    NodeColor FindColor();
    ~Node();

template<class T1, class T2>
class Tree {
private:
    Node<T1,T2>* Root;
    Node<T1,T2>* TNull;
public:
    Tree();
    Node<T1,T2>* FindRoot();
    Node<T1,T2>* FindTNull();
    void GetKey(T1 key);
    void GetValue(T2 value);
    void AllTreeDelete();
    Node<T1,T2>* Search(T1 key);
    void LeftRotation(Node<T1,T2>* x);
    void RightRotation(Node<T1,T2>* x);
    void Fixup(Node<T1,T2>* x);
    bool Insert(Node<T1, T2>* insertable_root);
    Node<T1,T2> *TreeMinimum(Node<T1,T2> *x);
    void Transplant(Node<T1,T2> *u, Node<T1,T2> *v);
    void DeleteFixup(Node<T1,T2> *x);
    void Delete(Node<T1,T2>* z);
    ~Tree();
```

functions.hpp	
int Equal_strings(const char *string1, const char *string2);	Функция сравнения строк. Возвращает значение в зависимости от того какая из них больше, а также, если они равны.
void Str_copy(const char *line, char *string);	Функция копирования строки. Также она преобразовывает верхний регистр символа строки в нижний.
work_with_files.hpp	
void Load_tree(std::ifstream *File, Tree<T1,T2> *tree)	Функция загрузки дерева из файла. Из бинарного файла считываются строка и значение, записываются в узел, который добавляется в дерево. При успешной загрузке нового дерева прошлое дерево удаляется.
void Tree_save(std::ostream &File, Tree<T1,T2> *tree, Node<T1,T2> *node)	Функция сохранения дерева в файл. Выполняется рекурсивный обход дерева от наименьшего ключа к наибольшему. Ключ и значение текущего рассматриваемого ключа записываются в бинарный файл.
menu.hpp	
int Menu()	Функция обработки запросов.
main.cpp	
int main()	Функция выполнения программы.

4. Консоль

```
root@Harry:~/work/MAI/DA/da_lab2# cat tests/test_01.txt
+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
- A
a
root@Harry:~/work/MAI/DA/da_lab2# ./solution < tests/test_01.txt
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
root@Harry:~/work/MAI/DA/da_lab2# cat tests/test_02.txt
+ hello 4534
+ hefrs 6546
+ sdfds 765
+ fgdsf 56645
+ fdgd 6754
+ a 655476
a
+ hello 6546547
- ardsfs
+ sfdtgrt 56746
sfdtgrt
+ dsfsd 4324
+ fdsdsd 5654765
+ hello 453456
+ jet 55435
+ koll 6546
- jet
+ hello 65456
+ jedai 5454
+ drive 3423
! Save hello
! Load hello
+ hello 4324
+ rudi 5435
root@Harry:~/work/MAI/DA/da_lab2# ./solution < tests/test_02.txt
OK
OK
OK
OK
OK
OK
OK: 655476
Exist
NoSuchWord
OK
OK: 56746
OK
OK
```

```
Exist
OK
OK
OK
Exist
OK
OK
OK
OK
Exist
OK
root@Harry:~/work/MAI/DA/da_lab2# cat tests/test_03.txt
+ harry 46365
+ liker 645654
+ garry 767657657
+ dima 564645
+ lesha 54656
+ daniil 67657
+ gaver 65778678452
+ rudi 75342
+ stint 6546
lesha
! Save load
+ igor 53436
+ rainbow 577834
+ bob 76554
+ masha 432675
+ maria 877645
rainbow
! Load load
+ marry 543536
+ dima 124234
- lesha
- masha
root@Harry:~/work/MAI/DA/da_lab2# ./solution < tests/test_03.txt
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK: 54656
OK
OK
OK
OK
OK
OK
OK: 577834
OK
OK
Exist
OK
NoSuchWord
```

5. Тест производительности

Сравнение красно-чёрного дерева производилось *с std::map*. В нём также ключи представляли собой массивы символов, а значения типа *unsigned long long*. Запись и чтение были опущены для сравнения только функций самого дерева.
Были проведены тесты на 1000, 10000, 100000 строк.

```
root@Harry:~/work/MAI/DA/da_lab2# g++ benchmark.cpp -03 -o benchmark
root@Harry:~/work/MAI/DA/da_lab2# ./benchmark < test_1000.txt
Red-Black Tree: 1.050 ms
std::map: 0.667 ms
root@Harry:~/work/MAI/DA/da_lab2# ./benchmark < test_10000.txt
Red-Black Tree: 11.878 ms
std::map: 8.646 ms
root@Harry:~/work/MAI/DA/da_lab2# ./benchmark < test_100000.txt
Red-Black Tree: 131.895 ms
std::map: 111.765 ms
```

На тестах с 1000 и 10000 std::map оказался быстрее примерно в 1.5 раза, чем алгоритм красно-чёрного дерева. На тесте 100000 оказался быстрее примерно в 1.18 раза. Контейнер std::map реализован с помощью красно-чёрного дерева. Из-за того, что в моей версии дерева происходит много копирований, это занимает лишнее время, поэтому std::map может выигрывать, так как при вставке и удалении элементов происходит меньше копирований, чем в моём дереве.

6. Выводы

Была написана программа, в которой реализована структура данных красно-чёрное дерево. Выполняя лабораторную работу, я узнал что из себя представляет красно-чёрное дерево, как его реализовать и как с ним работать. Также были получены навыки работы с выделением памяти. При выделении памяти нужно следить за тем, чтобы в конце блок выделенной памяти был освобождён. Если не освободить память, то произойдёт утечка памяти, что может повлечь проблемы. Были получены знания о работе с бинарными файлами, записи и чтения из них. С помощью этих знаний были реализованы запись красно-чёрного дерева в бинарный файл, а также чтение элементов красно-чёрного дерева из файла.

Список используемых источников

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. Роберт Лафоре. *Объектно-ориентированное программирование в C++*. Классика Computer Science, 4-е издание. — Издательский дом «Питер», 2018. Перевод с английского: А. Кузнецов, М. Назаров, В.Шрага. — 928 с. (ISBN 978-5-596-00353-7 (рус.))
3. *Красно-чёрное дерево – Википедия.*
URL: https://ru.wikipedia.org/wiki/Красно-чёрное_дерево (дата обращения: 16.11.2020).
4. *Информация о красно-чёрном дереве* – [Электронный ресурс]. – URL: https://habr.com/ru/post/330644/ (дата обращения: 16.11.2020).
5. *Информация о красно-чёрном дереве* – [Электронный ресурс]. – URL: https://habr.com/ru/post/330644/ (дата обращения: 16.11.2020).
6. *Информация о красно-чёрном дереве* – [Электронный ресурс]. – URL: http://www.codenet.ru/progr/alg/sort_search/rbt.php (дата обращения: 17.11.2020).
6. *Работа с бинарными файлами* – [Электронный ресурс]. – URL: http://ci-plus-plus-snachala.ru/?p=86 (дата обращения: 17.11.2020).