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: Н. А. Рудаков
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата: 01.05.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №2

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.

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

Вид дерева: Красно-чёрное дерево.

1 Описание

Как сказано в [1]: «Красно-черное дерево представляет собой бинарное дерево поиска с одним дополнительным битом цвета в каждом узле. Цвет узла может быть либо красным(RED), либо черным(BLACK). В соответствии с накладываемыми на узлы дерева ограничениями ни один простой путь от корня в красно-черном дереве не отличается от другого по длине не более чем в два раза, так что красно-черные деревья являются приближенно сбалансированными».

Дерево называется красно-черным, если удовлетворяет следующим условиям:

1. Каждый узел бывает либо красным, либо черным.
2. Корень дерва черный.
3. Каждый лист дерева (NIL) является черным узлом.
4. Если узел красный, то оба его дочених узла черные.
5. Для каждого узла все простые пути от него до листьев-потомков данного узла, содержат одинаковое количество черных узлов.

Операции красно-черного дерева и их свойства:

1. Поиск. Идентичен поиску в обычном бинарном дереве
2. Вставка. Первый этап повторяет вставку в обычное бинарное дерево. Второй этап подразумевет балланcировку. Сложность вставки оценивается как O(lg(n)).
3. Удаление. Первый этап соответствует удалению в обычном бинарном дереве. Второй этап подразумевает балланcировку для востановления свойств красночерного дерева. Сложность удаления оценивается как O(lg(n)).

2 Структура ноды и таблица с реализованными функциями

Каждый узел красно-черного дерева должен содержать в себе ключ, значение, цвет узла, указатели на своего правого и левого сына, а также указатель на родителя. Для хранения основной информации о дереве создадим вспомогательную структуру RBTNode.

```
1 struct RBTNode {
2     std::string key;
3     unsigned long long value{};
4     RBTNode *parent{};
5     RBTNode *left{};
6     RBTNode *right{};
7     Color color{};
8 };
```

Функции, которые были реализованы:

main.cpp	
RBTNode *searchNodeHelper(RBTNode *node, const std::string &key)	Функция, в которой записана логика поиска.
RBTNode *searchTree(const std::string &key)	Функция запуска поиска.
void insertNode(const std::string &key, const unsigned long long value)	Функция, в которой записана логика вставки и запуск вставки.
void deleteNodeHelper(RBTNode *node, const std::string &key)	Функция, в которой записана логика удаления.
void deleteNode(const std::string &key)	Функция запуска удаления.
void saveNode(RBTNode *temp, std::ofstream &output)	Функция сохранения ноды.
void saveTree(std::ofstream &output)	Функция сохранения дерева.
void recursiveDeleter(RBTNode *temp)	Функция, рекурсивно удаляющая все ноды (логика удаления дерева).
void removeTree()	Функция запуска удаления дерева.
void removeTree()	Функция запуска удаления дерева.
bool loadTree(std::ifstream &input)	Функция загрузки дерева и замена текущего словаря.
void leftRotate(RBTNode *node)	Функция, выполняющая левое вращение дерева.

void rightRotate(RBTNode *node)	Функция, выполняющая правое вращение дерева.
void fixedInsert(RBTNode *node)	Функция проверки баланса дерева после вставки.
void fixedDelete(RBTNode *node)	Функция проверки баланса дерева после удаления.
void transplant(RBTNode *destNode, RBTNode *sourceNode)	Функция, меняющая поддеревья.
RBTNode *searchMinimum(RBTNode *node)	Функция, которая находит минимум в поддереве.
void toLower(std::string &sequence)	Функция перевода строки в нижний регистр.
int main()	Функция запуска программы.

3 Консоль

```
ruda@ruda-lp:~/CLionProjects/lab2$ cat test.txt
+ p 17030326885634999690
+ e 14649173155356222787
+ z 2489607144041589600
+ o 17129694238019720037
+ b 2574559976823088878
p
e
z
o
b
-p
-e
-z
-o
-b
ruda@ruda-lp:~/CLionProjects/lab2$ ./main <test.txt
OK
OK
OK
OK
OK
OK: 17030326885634999690
OK: 14649173155356222787
OK: 2489607144041589600
OK: 17129694238019720037
OK: 2574559976823088878
OK
OK
OK
OK
OK
ruda@ruda-lp:~/CLionProjects/lab2$
```

4 Тест производительности

Я решил сравнить время работы RBTree и std::map. Вначале я сгенерировал тест на вставку, поиск и удаление. Суммарно в тесте 3000000 строк. Затем, я замерил время работы своего дерева. Получилось 7 секунд. Полсе я изменил весь main под std::map, чтобы замерить время работы этих же операций. Удивительно, но он работал на 2 секунды дольше. В этот раз у меня получилось обогнать встроенную вещь.

```
ruda@ruda-lp:~/CLionProjects/lab2$ g++ main.cpp -o main
ruda@ruda-lp:~/CLionProjects/lab2$ ./main <test.txt
7 sec
ruda@ruda-lp:~/CLionProjects/lab2$ rm main
ruda@ruda-lp:~/CLionProjects/lab2$ g++ main.cpp -o main
ruda@ruda-lp:~/CLionProjects/lab2$ ./main <test.txt
9 sec
```

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я познакомился с тяжелой для меня в реализацией структуры данных, как красно-черное дерево.

Также при выполнении задания пришлось столкнуться с проблемой Segmentation Fault, которая возникала на очень больших тестах. Изначально, я сделал дерево по книге Кормена, разобрался с реализацией (это было довольно трудно), но потом, получил на 5 тесте re. Продолжил разбираться -> *произошло разыменование нулевого поинтера* -> добавил кучу костылей в delete и rotate -> получил TL -> переписал все заново -> profit!

Вероятно, из-за большого количества внесенных дополнительных правок для больших тестов, моя реализация работает немного быстрее, чем std::map, потому что иначе трудно представить, почему это дерево написано лучше. Подозреваю, что существуют тесты, с которыми данная реализация не справится.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008