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: М. А. Бронников
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
Дата:
Оценка:
Подпись:

Москва, 2019

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, начинающуюуся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант структуры: Красно-черное 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 баллансировку, которая выполняет максимум два "поворота". Сложность вставки оценивается как $O(lg(n))$.
3. **Удаление**. *Первый этап* соответствует удалению в обычном бинарном дереве. *Второй этап* подразумевает баллансировку для востановления свойств красно-черного дерева, которая использует не более трех "поворотов". Сложность удаления оценивается как $O(lg(n))$.
4. **Сохранение в файл**. Первым в файл печатается *высота дерева*. Далее при проходе в глубину для кажого узла записывается *положение относительно предыдущего элемента:* h - если узел корневой, n - если нужно "подняться"вверх по дереву на один шаг(может следовать последовательность таких симолов), r - правый "сын"узла, l - левый "сын"узла. Далее записывается *значение , ключ, бит(на деле байт) цвета* и так пока не завершится проход в глубину символом n.
5. **Загрузка из фала**. Создает *новое дерево* согласно последовательности действий, заложенной в файл, проходом в глубину. Если при создании дерева "успех то уничтожаем старое дерево и заменяем на новое.

2 Исходный код

Каждый узел **красно-черного дерева** должен содержать в себе *ключ и значение* добавляемой пары, *цвет* узла, *указатели* на своего правого и левого "сына". Для этого создадим структуру $BRTree$. Для хранения основной информации о дереве построим вспомогательную структуру $Node$, содержащую *указатель* на корень дерева и *количество* элементов в дереве для более экономичной по памяти реализации прохода в глубину. //

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define BUFSIZE 257 // 256 + '\0'

typedef unsigned long long int TVal;

typedef struct BRTree{
  TVal value;
  char* key;
  bool color;
  struct BRTree* right;
  struct BRTree* left;
}BRTree;

typedef struct Node{
  BRTree* head;
  unsigned int size;
}Node;

BRTree* Search_Tree(Node* t, char* k);
void Create_Tree(Node* t);
void Error(short unsigned int e);
short unsigned int Tree_Insert(Node* t, char* k, short unsigned int si, TVal val);
short unsigned int Tree_Delete(Node* t, char* k);
short unsigned int Load_Tree(Node* t, char* path);
short unsigned int Save_Tree(Node* t, char* path);
void Destroy_Tree(Node* t);
BRTree* Left_Rotate(BRTree *h);
BRTree* Right_Rotate(BRTree *h);

}
```

Основные методы и функции:

main.c	
void Create_Tree(Node* t)	Метод инициализации дерева(он есть, но не используется)
BRTree* Search_Tree(Node* t, char* k)	Функция поиска в дереве по ключу, возвращает узел
void Error(short unsigned int e)	Примитивная функция обработки ошибок с выводом на экран
short unsigned int Tree_Insert(Node* t, char* k, short unsigned int si, TVal val)	Функция вставки нового элемента в дерево, дополнительно принимает в качестве аргументов указатель на ключ-строку, её длину и значение
short unsigned int Tree_Delte(Node* t, char* k)	Функция удаления узла из дерева по ключу
short unsigned Load_Tree(Node* t, char* path)	Функция загрузки дерева из файла
short unsigned Save_Tree(Node* t, char* path)	Функция сохранения дерева в файл
void Destroy_Tree(Node* t)	Метод уничтожения дерева
BRTree* Left_Rotate(BRTree* h)	Вспомогательная функция для балансировки, выполняющая левый "поворот"
BRTree* Right_Rotate(BRTree* h)	Вспомогательная функция для балансировки, выполняющая правый "поворот"

Стоит отметить, что все проходы по дереву осуществляются при помощи *стековой* модели без использования *рекурсии.*
Также левый и правый поворот были несколько модернезированы на основе варианта, представленного[3], и поэтому эти функции сразу перекрашивают узлы после поворота.

Помимо самой программы из *main.c* у меня есть генератор тестов *test.cpp*, который генерирует рандомные ключи и добавляет их в файл вместе со значениями, а позже сначала находит и затем удаляет их в произвольном порядке.

3 Консоль

```
max@max-X550CC:~/DA/lab2$ g++ test.cpp -o test
max@max-X550CC:~/DA/lab2$ ./test
max@max-X550CC:~/DA/lab2$ ls
file  main.c  test  test.cpp
max@max-X550CC:~/DA/lab2$ gcc -std=c99 -Wall -g main.c -o prog
max@max-X550CC:~/DA/lab2$ cat file
+ fmcdphhkdfpeiitxesqhrmcbekjvxohdejivtpiywbeflxdqqvxkkclrmwnmowpuibqcqccodg
1640521021
+ gchaahklkydyvsmmpehxfbdwdhninlyvpjvqqjedkjdicpvtvdtceybjgotvctssfqlvcpamyfvdwtwswsue
1192649062
+ vpgqkljvjmljbmoyplsofshcyabocnbafhqpuanfobqsngrftntbgegfghviuxidhbudejkumbndjgigtfhc
2077424121
+ nssrcdmrgduqjfwfkgjvwuotcpoibmxqgsikvxcebywmfvtrddqcxhxawpkacirkd 1202300827
+ yccgdcdrjaldfqvjqdhtcwefcpyosbmuerdjvgcfgpknjjwamftrebwisvanymkfgnpcutjdlurueqvsvrmc
1649929416
+ icqkfqmvboyqvqbeqxhhwjingvaljqgtuwhaptyqiahfslljjttidevkbwxmpgjldqnskmmvmtdfhorsknbc
1048951652
+ sybpwgrmgqjinjwjvungxkeemfgbxkntjolhugvdyfnnqlwmimtgyxnoftpefdaotlxqttvtbjhsughdvdmu
1430414751
+ bgrscfqsanotqbdjvjtcfwqddqommseobvhebawbpouhpxsnkmrrmiwryoemhidigmohomjhdg
1649984314
+ hjityblnljflxjahtfrathjivupyegtlsefrfsfrekfevhlpogrindsjylkdtgqmkxeqqjkxvrcrbqjrwbcn
1818471452
+ yqlseapeykcbqirdfjiceourgrxendgnvtidwahvnkyguqjbdufhjabprbwigdvfagiygruvevcapodsjics
1447125600
icqkfqmvboyqvqbeqxhhwjingvaljqgtuwhaptyqiahfslljjttidevkbwxmpgjldqnskmmvmtdfhorsknbqrx
-icqkfqmvboyqvqbeqxhhwjingvaljqgtuwhaptyqiahfslljjttidevkbwxmpgjldqnskmmvmtdfhorsknbqr
bgrscfqsanotqbdjvjtcfwqddqommseobvhebawbpouhpxsnkmrrmiwryoemhidigmohomjhdg
-bgrscfqsanotqbdjvjtcfwqddqommseobvhebawbpouhpxsnkmrrmiwryoemhidigmohomjhdg
vpgqkljvjmljbmoyplsofshcyabocnbafhqpuanfobqsngrftntbgegfghviuxidhbudejkumbndjgigtfhcjr
-vpgqkljvjmljbmoyplsofshcyabocnbafhqpuanfobqsngrftntbgegfghviuxidhbudejkumbndjgigtfhcj
yccgdcdrjaldfqvjqdhtcwefcpyosbmuerdjvgcfgpknjjwamftrebwisvanymkfgnpcutjdlurueqvsvrmduj
-yccgdcdrjaldfqvjqdhtcwefcpyosbmuerdjvgcfgpknjjwamftrebwisvanymkfgnpcutjdlurueqvsvrmdu
gchaahklkydyvsmmpehxfbdwdhninlyvpjvqqjedkjdicpvtvdtceybjgotvctssfqlvcpamyfvdwtwswsuesv
-gchaahklkydyvsmmpehxfbdwdhninlyvpjvqqjedkjdicpvtvdtceybjgotvctssfqlvcpamyfvdwtwswsues
fmcdphhkdfpeiitxesqhrmcbekjvxohdejivtpiywbeflxdqqvxkkclrmwnmowpuibqcqccodg
-fmcdphhkdfpeiitxesqhrmcbekjvxohdejivtpiywbeflxdqqvxkkclrmwnmowpuibqcqccodg
nssrcdmrgduqjfwfkgjvwuotcpoibmxqgsikvxcebywmfvtrddqcxhxawpkacirkd
-nssrcdmrgduqjfwfkgjvwuotcpoibmxqgsikvxcebywmfvtrddqcxhxawpkacirkd
```

```
sybpwgrmgqjinjwjvungxkeemfgbxkntjolhugvdyfnnqlwmimtgyxnoftpefdaotlxqttvtbjhsughdvdmuda
-sybpwgrmgqjinjwjvungxkeemfgbxkntjolhugvdyfnnqlwmimtgyxnoftpefdaotlxqttvtbjhsughdvdmuc
hjityblnljflxjahtfrathjivupyegtlsefrfsfrekfevhlpogrindsjylkdtgqmkxeqqjkxvrcrbqjrwbcmhv
-hjityblnljflxjahtfrathjivupyegtlsefrfsfrekfevhlpogrindsjylkdtgqmkxeqqjkxvrcrbqjrwbcmh
yqlseapeykcbqirdfjiceourgrxendgnvtidwahvnkyguqjbdufhjabprbwigdvfagiygruvevcapodsjicsic
-yqlseapeykcbqirdfjiceourgrxendgnvtidwahvnkyguqjbdufhjabprbwigdvfagiygruvevcapodsjicsi
! Save loadfile
! Load loadfile
max@max-X550CC:~/DA/lab2$ ./prog <file
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK: 1048951652
OK
OK: 1649984314
OK
OK: 2077424121
OK
OK: 1649929416
OK
OK: 1192649062
OK
OK: 1640521021
OK
OK: 1202300827
OK
OK: 1430414751
OK
OK: 1818471452
OK
OK: 1447125600
OK
OK
OK
```

4 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я познакомился с такой тяжелой для меня в реализации структурой данных, как *красно-черное дерево*. Данное задание заставило меня по-новому взглянуть на вопрос о том как нужно программно хранить и использовать информацию, недаром красно-черное дерево легло в основу множества используемых и по сей день инструментов, таких как, например, *std::map*.

Также при выполнении задания пришлось столкнуться с такими новыми для себя проблемами, как полное переписывание целой функции из-за того, что старая реализация не подходит под требованиям к скорости работы. Эта задача осложнялась тем, что приходится писать функцию с новым принципом работы под старые, уже написанные и отлаженные программные элементы.

Однако вряд ли даже сейчас мою программу можно назвать идеальной, хоть она и сооответствует требованиям, поставленным перед ней. Из самых бросающихся в глаза недочетов можно отметить неудачный выбор названий для структур.
Но ведь в этом и заключалась цель этой работы- получить на своих ошибках бесценный опыт в программировании и в создании словарей в частности.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского: И. В. Красиков. — 1323 с. (ISBN 978-5-8459-1794-2 (рус.))

[2] *Красно-черное дерево — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Красно-чёрне_дерево` (дата обращения: 17.02.2019).

[3] www.ibm.com — Работа со структурами данных в языках Си и Python Часть 9. Красно-черные деревья