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: А. М. Голев
Преподаватель: С. А. Михайлова
Группа: М8О-201Б

Дата: 18.04.25
Оценка:
Подпись:

Москва, 2025

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 264 - 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.

Структура данных: `Красно-чёрное дерево`

1 Описание

Требуется написать реализацию структуры данных красное-черное дерево.

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 более чем в два раза, так что красно-черные деревья являются приближенно сбалансированными. [1]

КЧ-деревья обладает следующими свойствами [2] :

1. Узел может быть либо красным, либо чёрным и имеет двух потомков;
2. Корень — чёрный. Это правило слабо влияет на работоспособность модели, так как цвет корня всегда можно изменить с чёрного на красный;
3. Все листья, не содержащие данных — чёрные.
4. Оба потомка каждого красного узла — чёрные.
5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.

Работа процедур для кч-деревьев заключается в выполнении требуемых операций (поиск, вставка и удаление) таким образом, чтобы сохранялись указанные свойства.

Заметим, что поиск в кч-дереве идентичен поиску в BST.

Операция вставки на первом этапе идентична операции вставки в BST. При этом вставляемый узел красный, поскольку вставка красного узла может нарушить меньшее количество свойств, чем вставка черного. После вставки нарушенными могут оказаться свойства 2 и 4. Задача второго этапа операции вставки состоит в восстановлении этих свойств, посредством применения операций перекраски и поворотов, не нарушающих свойств дерева, как BST. На этом этапе окружение узла, для которого происходит поправка, классифицируется на 3 (с точностью до симметрии) случая:

1. «Дядя - красный» - родитель и дядя узла перекрашиваются в черный, дед - в красный. Дед становится текущим узлом.
2. «Текущий узел - правый ребенок» - выполняется левый поворот относительно родителя. Текущим становится узел, который до этого был родителем. Таким образом случай 2 сводится к случаю 3.

3. «Дядя - черный, текущий узел - левый ребенок» - цвета родителя и деда меняются местами и выполняется правый поворот относительно деда.

В конце корень безусловно окрашивается в черный для восстановления свойства 2.

Операция удаления на первом этапе идентична операции удаления из BST, за исключением того, что фиксируется цвет удаляемого узла. При этом второй этап, заключающийся в восстановлении нарушенных свойств, требуется только если был удален черный узел, поскольку удаление красного не нарушает ни одного из свойств. При удалении черного узла очевидно нарушается свойство 5. Для конпенсации этого единственному потомку удаленного узла присваивается дополнительная чернота (т.е. красный узел считается красно-черным, а черный - дважды черным), что исправляет нарушение свойства 5, но нарушает свойство 1. Задача второго этапа устранить дополнительную черноту текущего узла. Для этого окружение узла классифицируется на 4 (с точностью до симметрии) случая:

1. «Брат - красный» - цвета родителя и брата меняются местами и выполняется левый поворот относительно родителя. Братом становится прежний черный сын красного брат. Таким образом случай 1 сводится к случаям 2, 3 или 4.

2. «Дети брата - черные» - Брат перекрашивается в красный, а текущим узлом становится родитель.

3. «Правый ребенок брата - черный» - Цвета брата и его левого ребенка меняются местами, выполняется правый поворот относительно относительно брата. Таким образом случай 3 сводится к случаю 4.

4. «Правый ребенок брата - красный» - Брат получает цвет родителя, а родитель и правый сын брата становятся черными, выполняется левый поворот относительно родителя.

Такой подход позволяет свести текущий узел к корню или красно-черному узлу, в конце текущий узел безусловно перекрашивается в черный, что восстанавливает свойства 1 и 2 и не приводит к нарушению других (идентично удалению красного узла).

2 Исходный код

Код программы можно логически разделить на 2 части: реализацию кч-дерева и реализацию интерфейса.

Реализация кч-дерева начинается с определения типов ключа, значения и вспомогательного типа Color для представления цвета узла. Затем определяется тип для представления ошибки раборты с файлами.

Далее следует определение структуры узла $_{N}ode$. Каждый узел содержит ключ key, значение $value$, указатели $left$, $right$ и p на левого, правого потомков и родителя соответственно и цвет $color$. В этой же конструкции определяется тип самого дерева $RBTree$, как указатель на узел.

Далее следуют заголовки публичных методов структуры. В конце объявляется вспомогательная функция $printTree$, использующаяся в целях отладки.

Далее определяется внутренний тип $_{L}oadError$. Он используется приватным методом $_{l}oadTraverse$ для представления типа ошибки при загрузке.

Затем определяется глобальная переменная $_{N}IL$, представляющая из себя указатель на узел, хранящийся в глобальной переменной. Этот узел представляет все узлы-пустышки в дереве. Такой подход позволяет экономить память на листовых узлах, при этом позволяет рассматривать их, как любые другие узлы.

Далее определяются вспомогательные функции $_{r}eadByte$ и $_{w}riteByte$ для чтения/-записи одного байта из файла/в файл.

Затем объявляются заголовки приватных методов структуры.

Реализация интерфейса начинается с определения констант, определяющих размеры буферов для команды, ключа и пути, а также константы строк-ответов.

Затем объявляется заголовок вспомогательной процедуры $readKey$, использующейся для считывания ключа и приведения его к нижнему регистру.

Далее объявляется заголовок функции UI, реализующей непосредственно логику обработки команд словаря и взаимодействия со структурой с их помощью.

После определяется точка входа в программу. В самом начале создается пустое дерево, после чего запускается интерфейс с ней. После завершения работы с интерфейсом происходит освобождение памяти, выделенной под структуру и программа завершается.

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

typedef enum Color { RED, BLACK } Color;
typedef char* Key;
typedef unsigned long int Value;

typedef enum FileError {
  NO_ERRORS,
  FE_CAN_NOT_WRITE,
  FE_CAN_NOT_READ,
  FE_PARSING_ERROR,
  FE_OUT_OF_MEMORY
} FileError;

typedef struct _Node {
  Key key;
  Value value;

  Color color;
  struct _Node *p;
  struct _Node *left;
  struct _Node *right;
} _Node, *RBTree;

RBTree createRBTree();
void deleteRBTree(RBTree tree);
bool getRBTree(RBTree tree, Key key, Value *res);
bool insertRBTree(RBTree *treeP, Key key, Value value);
bool removeRBTree(RBTree *treeP, Key key);
FileError saveRBTree(RBTree *treeP, char *path);
FileError loadRBTree(RBTree *treeP, char *path);
void printTree(RBTree tree, int offset);

typedef enum _LoadError {
  LOAD_OK,
  PARSING_ERROR,
  OUT_OF_MEMORY
} _LoadError;

_Node _nilStruct = {NULL, 0, BLACK, NULL, NULL, NULL};
RBTree _NIL = &_nilStruct;

void _writeByte(FILE *file, unsigned char byte);
unsigned char _readByte(FILE *file);

```

```
50 RBTree _searchRBTree(RBTree tree, Key key);
51 void _insertFixupRBTree(RBTree *treeP, RBTree z);
52 void _deleteFixupRBTree(RBTree *treeP, RBTree x);
53 void _saveTraverse(RBTree tree, FILE *file);
54 _LoadError _loadTraverse(RBTree *treeP, FILE *file);
55 void _RotateLeft(RBTree *treeP, RBTree x);
56 void _RotateRight(RBTree *treeP, RBTree x);
57 void _Transplant(RBTree *treeP, RBTree u, RBTree v);
58
59 #define MAX_KEY_LENGTH 257
60 #define MAX_INPUT_LENGTH 280
61 #define MAX_PATH_LENGTH 257
62
63 #define RESULT_SUCCESS "OK"
64 #define RESULT_NOT_FOUND "NoSuchWord"
65 #define RESULT_ALREADY_EXISTS "Exist"
66 #define RESULT_ERROR "ERROR"
67
68 void readKey (Key dst, char *src, int keyLength);
69 void UI(RBTree *treeP);
70
71 int main() {
72     RBTree tree = createRBTree();
73     UI(&tree);
74     deleteRBTree(tree);
75
76     return 0;
77 }
```

main.c	
RBTree createRBTree()	Конструктор кч-дерево (только что созданное дерево - пустое).
void deleteRBTree(RBTree tree)	Деструктор кч-дерева.
bool getRBTree(RBTree tree, Key key, Value *res)	Метод получения значения по ключу. Возвращает false, если ключ не найден.
bool insertRBTree(RBTree *treeP, Key key, Value value)	Метод вставки пары ключ-значение в дерево. Возвращает false, если ключ уже есть.
bool removeRBTree(RBTree *treeP, Key key)	Метод удаления пары из дерева по ключу. Возвращает false, если ключ не найден.
FileError saveRBTree(RBTree *treeP, char *path)	Метод сохранения дерева в файл в компактном бинарном представлении. Возвращает тип ошибки.

FileError loadRBTree(RBTree *treeP, char *path)	Метод загрузки дерева из ранее сохраненного файла. Возвращает тип ошибки.
void printTree(RBTree tree, int offset)	Отладочная функция для печати дерева.
void _writeByte(FILE *file, unsigned char byte)	Вспомогательная функция для записи байта в файл.
unsigned char _readByte(FILE *file)	Вспомогательная функция для считывания байта из файла.
RBTree _searchRBTree(RBTree tree, Key key)	Приватный метод для поиска узла по ключу. Возвращает _NIL, если ключ не найден.
void _insertFixupRBTree(RBTree *treeP, RBTree z)	Приватный метод исправления нарушений свойств кч-дерева после вставки.
void _deleteFixupRBTree(RBTree *treeP, RBTree x)	Приватный метод исправления нарушений свойств кч-дерева после удаления.
void _saveTraverse(RBTree tree, FILE *file)	Приватный метод реализующий preorder обход для сохранения дерева.
_LoadError _loadTraverse(RBTree *treeP, FILE *file)	Приватный метод реализующий считывание файла и построение дерева по нему.
void _RotateLeft(RBTree *treeP, RBTree x)	Приватный метод реализующий операцию левого поворота.
void _RotateRight(RBTree *treeP, RBTree x)	Приватный метод реализующий операцию правого поворота.
void _Transplant(RBTree *treeP, RBTree u, RBTree v)	Приватный метод замены поддерева u поддеревом v
void readKey (Key dst, char *src, int keyLength)	Вспомогательная функция для считывания ключа. Приводит вводимый ключ к нижнему регистру.
void UI(RBTree *treeP)	Функция, реализующая пользовательский интерфейс.

3 Консоль

```
gcc -o app.out main.c
./app.out
+ a 1
OK
+ A 2
Exist
+ aaaaa 18446744073709551615
OK
aaaaa
OK: 18446744073709551615
A
OK: 1
-A
OK
a
NoSuchWord
+ b 2
OK
! Save file1
OK
-b
OK
b
NoSuchWord
! Load file1
OK
b
OK: 2
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнивается время обработки последовательности из 10^6 случайных команд программами, использующими реализованную структуру и std::map. Время на ввод и вывод не учитывается.

```
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab2/benchmark$ make
gcc rb.c -o rb.out
g++ map.cpp -o map.out
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab2/benchmark$ make test_rb
./rb.out <./in.txt | grep "time"
time: 1221.265000ms
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab2/benchmark$ make test_map
./map.out <./in.txt | grep "time"
time: 1469.249000ms
```

Как видно, реализованная структура несколько выигрывает у std::map из стандартной библиотеки, однако различие незначительно. Эти результаты согласуются с тем фактом, что обе программы используют кч-деревья для реализации словаря.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать структуру данных красно-черное дерево на языке программирования C, производить сериализацию и десериализацию структуры в компактное бинарное представление, использовать контейнер std::map из стандартной библиотеки языка C++. Я узнал об особенностях и тонкостях реализации красно-черных деревье.

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] *Красно-черные деревья — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Красно-чёрное_дерево` (дата обращения: 16.04.2025).