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. В. Тимофеев
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №2

Задача: Вариант №02

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.

Вариант структуры данных: Красно-чёрное дерево.

Формат входных данных: 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.

1 Описание

Как сказано в [1]:«Красно-чёрное 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 подразумевает 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 красночерного дерева, которая использует не более трех "поворотов". Сложность удаления оценивается как O(lg(n)).

4. Сохранение в файл.
Сначала происходит рекурсивный обход в глубину. От корня спускаемся до листа, при каждом переходе к дочернему узлу печатается в выходной файл направление, по которому мы переходим, L - влево от родительского, узла R - вправо от родительского. Когда доходим до листа рекурсия сворачивается, и при этом мы записываем данные (цвет: 0 - черный, 1 - красный; value; строку) в выходной файл.

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 TNode. Корень дерева вынесем в отдельную структуру TTree.

main.c	
main()	Основная функция, в ней происходит ввод и распознавание данных, определяется, соответствуют ли данные требованиям, также происходит управление классом TTree.
main.c	
struct TNode *MakeTNode (const char * data, TVal val)	Функция, которая создает объект.
main.c	
void DeleteTree (struct TNode *root)	Функция, которая разрушает объект.
main.c	
struct TNode *RRemove (struct TNode *root, char * data, int *done)	Функция рекурсивного удаления узла со строкой data.
main.c	
short int TRemove (struct TTree *tree, char * data)	Функция, запускающая рекурсивное удаление.
main.c	
struct TNode *RemoveBalance (struct TNode *root, int dir, int *done)	Функция балансировки дерева после операции удаления.
main.c	
short int IsRed (struct TNode *root);	Функция проверяет цвет узла.
main.c	
struct TNode *RotDouble (struct TNode *root, int dir);	Функция двойного поворота.
main.c	
struct TNode *RotSingle (struct TNode *root, int dir)	Функция одиночного поворота.
main.c	
short int TInsert (struct TTree *tree, const char * data, TVal val)	Функция, запускающая рекурсивную вставку.
main.c	
struct TNode *RInsert (struct TNode *root, const char * data, TVal val)	Функция рекурсивной вставки.

main.c	
void TSearch (struct TNode *root, const char *data)	Функция рекурсивного поиска.
main.c	
short int MyIsAlpha(const char *str)	Функция проверяет, являются ли элементы массива буквами.
main.c	
short int DeSerialize(struct TNode *root, FILE *f)	Функция десерелизации.
main.c	
void Serialize(struct TNode *root, FILE *f)	Функция серелизации.
main.c	
unsigned short int SaveTree(struct TTree* tree, const char* path)	Функция запуска серелизации.
main.c	
short int LoadTree(struct TTree* tree, const char* path)	Функция запуска десерелизации.
main.c	
struct TNode *MakeTNodeDes (const char * data, TVal val, int red)	Функция создания узла при десерелизации.

Стоит отметить, что все проходы по дереву осуществляются при помощи рекурсии. Также левый и правый поворот были несколько модернезированы для более компактного кода на основе варианта, представленного в [2].

```
1  #include <stdlib.h>
2  #include <time.h>
3  #include <string.h>
4  #include <math.h>
5
6  const int DATA_STR_LEN = 257;
7  const int RED = 1;
8  const int VAL_LEN = 21;
9  typedef unsigned long long int TVal;
10
11 struct TNode {
12     TVal val;
13     int red;
14     char* data;
15     struct TNode *link [2];
16 };
17 struct TTree {
18     struct TNode *root;
```

```
};
struct TNode *RRemove (struct TNode *root, char * data, int *done);
short int TRemove (struct TTree *tree, char * data);
struct TNode *RemoveBalance (struct TNode *root, int dir, int *done);
short int IsRed (struct TNode *root);
struct TNode *RotDouble (struct TNode *root, int dir);
struct TNode *RotSingle (struct TNode *root, int dir)
short int TInsert (struct TTree *tree, const char * data, TVal val);
struct TNode *RInsert (struct TNode *root, const char * data, TVal val);
void TSearch (struct TNode *root, const char *data);
short int MyIsAlpha(const char *str);
void DeleteTree (struct TNode *root);
short int DeSerialize(struct TNode *root, FILE *f);
void Serialize(struct TNode *root, FILE *f);
struct TNode *MakeTNode (const char * data, TVal val);
struct TNode *MakeTNodeDes (const char * data, TVal val, int red);
unsigned short int SaveTree(struct TTree* tree, const char* path);
short int LoadTree(struct TTree* tree, const char* path);
```

3 Консоль

Тесты для этой лабораторной работы я делал как вручную, так и с помощью самописного генератора тестов. Вручную я проверял конкретные нетривиальные случаи, а генератором создавал большие однообразные тесты для оценки производительности. Пример урезанного теста, полученного с помощью генератора, приведен ниже.

```
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/Laba2$ cat t.txt
+ IZHJTVHEEB 17974477166857275917
+ NPTMMDEDQY 370355204217877396
+ AYYPAFZGRT 13285239387186300850
+ HFQVNGTCCX 43437915520765232 99
+ KTOYEEXUMA 84489720095477430 29
+ OLiNRUFUYZU 137319735407887137
+ ABDYiGINKHY 25315355260188804037
+ XYUGOiQVSPI 143219096909913183 74
+ ZCKKiUCLBRV 4931705949798879309
+ REFiNCBSPBJ 7373797892414157279
+ BQKCiFBJTQP 576365749937499333
+ PILCiCKAALW 165822937204099610 90
+ EVEMUDiZMSB 181153828556631242 54
+ AIRUiFXYQLM 114565883862519047 44
+ YJWiNPAAEPF 2472807705417724924
+ BEWXGJPiHLN 2878489296747235313
+ BiYQZZEDRGX 8023386416937237979
+ PMIONIiFUKT 3963041103399177625
+ FLKiSTQSNXU 7196245667501823038
+ TOFiQSRNCME 3740304964053012568
+ ARSFKFiYVZH 144465958094846467 42
+ IFQiRGPQRVZ 170360957980941148
+ WSGiIHZBTIZ 1829937667655070276
+ IOOQMiQYWQH 111583196395363464 51
! Save savef
PMIONIiFUKT
BEWXGJPiHLN
-BEWXGJPiHLN
BEWXGJPiHLN
-IFQiRGPQRVZ
! Load savef
BEWXGJPiHLN
IFQiRGPQRVZ
```

```
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/Laba2$ ls
makefile   savef   solution   solution.c   t.txt  'отчет лаба2'
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/Laba2$ ./solution <t.txt
>rep.txt
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/Laba2$ ls
makefile   rep.txt   savef   solution   solution.c   t.txt  'отчет лаба2'
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/Laba2$ cat rep.txt
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK: 3963041103399177625
OK: 2878489296747235313
OK
NoSuchWord
OK
OK
OK: 2878489296747235313
OK: 170360957980941148
```

4 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я познакомился с структурой данных красно-чёрное дерево. Эта структура оказалась сложна в реализации, но достаточно эффективна по времени, поэтому ее до сих пор используют в реализации некоторых инструментов, например, std::map.

Для меня было достаточно непривычно сравнивать элементы дерева не по целочисленному id, а по строковому, также трудности вызвала серелизация и десерелизация дерева, и его балансировка.

Программа соответствует требованиям, поставленным перед ней, однако не является идеальной. Улучшить её можно придумав более эффективный ввод данных и десерелизацию.

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] *Абстрактные типы данных — Красно-чёрные деревья (Red black trees) — rfLinux.*
URL: `http://rflinux.blogspot.com/2011/10/red-black-trees.html` (дата обращения: 10.10.2021).

[3] *Красно-черное дерево — neerc.ifmo.ru.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Красно-черное_дерево` (дата обращения: 30.09.2021).