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: С. В. Кудинов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-18
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. Необходимо реализовать следующие функции:

- Добавление в словарь пары из ключа и значения
- Удаление из словаря элемента по ключу
- Поиск элемента в словаре по ключу
- Сохранение словаря в бинарный файл
- Загрузка словаря из бинарного файла

Используемая структура данных: AVL-дерево

1 Описание

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

2 Исходный код

1 Описание программы

Программа предоставляет интерфейс для работы с вышеописанной структурой данных, принимая во входной поток строки с командами. Существует 5 возможных команд.

- *+ KEY VALUE* — Добавляет в дерево пару из ключа KEY и значения VALUE, если такого элемента ещё нет в дереве
- *- KEY* — Удаляет из дерева элемент с ключом KEY, если он существует
- *KEY* — Осуществляет поиск элемента с ключом KEY в дереве
- *! Save PATH_TO_FILE* — Сохраняет словаря в файл в компактном бинарном представлении
- *! Load PATH_TO_FILE* — Загружает словарь из бинарного файла, заменяя им текущий

Само дерево реализовано с помощью структуры TNode, BalanceTree, в которой хранится какая-либо вершина дерева, а также с помощью методов RemoveElement, FindByKey, FindMinimum, RemoveMinimum,AddElement, строка из 256 английских символов считается числом в 26-ричной системе, переводится в 10-ричную и так записывается в ключи дерева.
Рассмотрим реализации вышеописанных функций.

1.1 Поиск

Пусть в дереве мы ищем элемент с ключом *Key*. Сначала мы переводим *Key* из слова в число. Начиная с корневого элемента дерева, будем спускаться по дереву. Если ключ меньше ключа корневого элемента, нужно спуститься в левое поддерево и продолжить поиск, в ином случае, нужно продолжить спуск в правое поддерево.

1.2 Вставка

Стратегия для вставки элемента в дерево такова:

- Попытаться найти Key в дереве. Если он уже есть, вывести сообщение об ошибке.
- С помощью поиска Key найти пустой лист в дереве.
- Создать новый узел, вставив в него Key и $Value$. Рекурсивно вернуться в корень, балансируя при этом вершины.

1.3 Удаление

Для удаления существует всего 3 случая.

- Если в дереве корневой элемент не содержит потомков, то его просто удаляют. Рекурсивно вернуться, балансируя вершины.
- Если удаляемый элемент p содержит только левый/правый потомок, то удалить этот элемент, вставив родителю элемента ссылку на этот потомок вместо ссылки на удаленный элемент. Рекурсивно вернуться, балансируя вершины.
- Если удаляемый элемент содержит левого и правого потомка, тогда найти самый минимальный элемент в правом потомке – зайдя в правый потомок, всегда идти в левые потомки, пока не будет достигнут лист. Удалить этот лист и вставить его вместо того элемента, что требуется удалить. Сбалансировать дерево.[4]

1.4 Сохранение в файл

Рекурсивным обходом записать дерево в файл. Если узел равен null, записать в файл пару нулей. Иначе записать пару ключ-значение вершины, а после рекурсивно вызвать функции записи вершины у левого и правого потомка. Так как ключ и значения два числа, то запись пройдет быстро.

1.5 Считывание из файла

Благодаря тем данным, которые мы сохранили в файл, из них можно быстро сконструировать дерево. Достаточно считывать файл до его конца.

2 Таблица функций и методов

main.cpp, вспомогательная функция	
long long GetHeight(TNode* n)	Возвращает высоту узла
long long GetBalance(TNode* n)	Считает баланс узла
TNode* RotateLeft(TNode* q)	Выполняет поворот влево
TNode* RotateRight(TNode* q)	Выполняет поворот вправо
void WriteTree()	Записывает дерево в бинарном представлении в файл
void RemoveTree(Node* node)	Рекурсивно удаляет дерево
TNode* BalanceTree(TNode* p)	Балансирует узел
TNode* AddElement(TNode* p, unsigned long long k,unsigned long long value)	Добавляет элемент в дерево
TNode* RemoveElement(TNode* p, unsigned long long k)	Удаляет элемент из дерева
TNode* FindByKey(TNode* p, unsigned long long key)	Удаляет элемент с помощью ключа
void RemoveTree(Node* node)	Рекурсивно удаляет дерево
TNode* BalanceTree(TNode* p)	Балансирует узел
TNode* AddElement(TNode* p, unsigned long long k,unsigned long long value)	Добавляет элемент в дерево
TNode* RemoveElement(TNode* p, unsigned long long k)	Удаляет элемент из дерева
TNode* FindByKey(TNode* p, unsigned long long key)	Удаляет элемент с помощью ключа

3 Консоль

```
sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/da2$ g++ main.cpp
sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/da2$ ./a.out
+ a 1
OK
+ a 1
Exist
+ aaa 2
OK
+ aaaa 3
OK
aaa
OK: 2
! Save file1.txt
OK
-aaa
OK
aaa
NoSuchWord
! Load file1.txt
OK
aaa
OK: 2
^C
sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/da2$ hexdump
file1.txt
0000000 02bf 0000 0000 0000 0002 0000 0000 0000
0000010 0001 0000 0000 0000 0001 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
*
0000040 4767 0000 0000 0000 0003 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
*
0000070
```

4 Тест производительности

Напишем простую программу для генерации тестов и замера времени выполнения данных тестов. Она генерирует заданное количество пар из строк случайной длины от 1 до 256, содержащих случайные символы латинского алфавита, и чисел типа unsigned long long.

```
#include <ctime>
#include <random>
#include <map>
#include <limits>
#include <tuple>
#include <string>
#include <cstdlib>
#include <iostream>
#include <chrono>
#include <iomanip>
#include <iostream>
#include <string.h>
#include <fstream>
#include <stdlib.h>
#include "profile.h"
#include <algorithm>
using namespace std;

default_random_engine rng;
const int KEY_SIZE = 260;
const char ADD = '+';
const char DELETE = '-';
const char FILE_OPERATION = '!';
const int PATH_SIZE = 1000;
const int ALPHABET_SIZE = 26;
struct TNode {
unsigned long long Key;
unsigned long long Value;
TNode* Left;
long long Height;
TNode* Right;
TNode(unsigned long long key1, unsigned long long value1) {
Value = value1;
Key = key1;
Height = 1;
Left = 0;
Right = 0;
}

};
TNode* p = 0;

```

```
43 long long GetHeight(TNode* n) {
44 if (n!=NULL) {
45 return n->Height;
46 }
47 return 0;
48 }
49 long long GetBalance(TNode* n) {
50 return GetHeight(n->Right)-GetHeight(n->Left);
51 }
52
53 void CountHeight(TNode* n)
54 {
55 int hl = GetHeight(n->Left);
56 int hr = GetHeight(n->Right);
57 n->Height = (hl>hr?hl:hr)+1;
58 }
59 TNode* RotateLeft(TNode* q) {
60 TNode*p = q->Right;
61
62 q->Right = p->Left;
63 p->Left = q;
64 CountHeight(q);
65 CountHeight(p);
66 return p;
67
68 }
69 TNode* RotateRight(TNode* q) {
70 TNode* p = q->Left;
71 q->Left = p->Right;
72 p->Right = q;
73 CountHeight(q);
74 CountHeight(p);
75 return p;
76 }
77 TNode* BalanceTree(TNode* p) {
78 CountHeight(p);
79 if (GetBalance(p)==2) {
80 TNode* q = p->Right;
81 if (GetBalance(q)<0) {
82 p->Right = RotateRight(q);
83 }
84 return RotateLeft(p);
85 }
86 if (GetBalance(p)==-2) {
87 TNode* q = p->Left;
88 if (GetBalance(q)>0) {
89 p->Left=RotateLeft(q);
90 }
91 return RotateRight(p);
```

```
}
return p;
}

TNode* AddElement(TNode* p, unsigned long long k,unsigned long long value ) {
if( !p ) {
return new TNode(k,value);
}
if (k<p->Key) {
p->Left = AddElement(p->Left,k,value);
}
else if (k>p->Key) {
p->Right = AddElement(p->Right,k,value);
}
return BalanceTree(p);
}
TNode* FindMinimum(TNode* p) {
if (p->Left==0) {
return p;
}
return FindMinimum(p->Left);
}
TNode* RemoveMinimum(TNode* p)
{
if( p->Left==0 ) {
return p->Right;
}
p->Left = RemoveMinimum(p->Left);
return BalanceTree(p);
}
TNode* RemoveElement(TNode* p, unsigned long long k) {
if (!p) {
return 0;
}
if(k<p->Key) {
p->Left = RemoveElement(p->Left,k);
}
else if(k>p->Key) {
p->Right = RemoveElement(p->Right,k);
}
if (k==p->Key) {
TNode* q = p->Left;
TNode* r = p->Right;
delete p;
if (r==0) {
return q;
}
TNode* min = FindMinimum(r);
min->Right = RemoveMinimum(r);
```

```
141 min->Left = q;
142 return BalanceTree(min);
143 }
144 return BalanceTree(p);
145
146 }
147 void RemoveTree(TNode* p) {
148 if (!p) {
149 return;
150 }
151 RemoveTree(p->Left);
152 RemoveTree(p->Right);
153 delete p;
154 }
155 TNode* FindByKey(TNode* p, unsigned long long key) {
156 if (p==0) {
157 return 0;
158 }
159
160 if (key<p->Key) {
161 return FindByKey(p->Left,key);
162 }
163 if (key>p->Key) {
164 return FindByKey(p->Right,key);
165 }
166 return p;
167 }
168 std::ofstream fs;
169 std::ifstream input;
170 unsigned long long Key;
171 unsigned long long value;
172 void WriteTree(char path[PATH_SIZE], TNode* p) {
173
174 if (p==0) {
175 unsigned long long v = 0;
176 unsigned long long k = 0;
177
178 fs.write((char*)(&k), sizeof(unsigned long long ));
179 fs.write((char*)(&v), sizeof (unsigned long long ));
180 return;
181 }
182 fs.write((char*)(&p->Key), sizeof(unsigned long long));
183 fs.write((char*)(&p->Value), sizeof (unsigned long long));
184 WriteTree(path,p->Left);
185 WriteTree(path,p->Right);
186
187 }
188 TNode* ReadTree() {
189 TNode* q = 0;
```

```
190 if (input.read((char*)(&Key), sizeof(unsigned long long int))) {
191 input.read((char*)(&value), sizeof (unsigned long long int));
192 if (Key==0) {
193 return q;
194 }
195 q = new TNode(Key,value);
196 }
197 q->Left = ReadTree();
198 q->Right = ReadTree();
199 return q;
200 }
201 char command[KEY_SIZE];
202 char operation[KEY_SIZE];
203 unsigned long long GetHash(char key[KEY_SIZE]) {
204 int i = 0;
205 unsigned long long a=0;
206 while ((key[i]))
207 {
208 a*=ALPHABET_SIZE;
209 a+=(key[i]-('a'-1));
210 ++i;
211 }
212 return a;
213 }
214 uint64_t get_number(uint64_t min = 0,uint64_t max = numeric_limits<unsigned long long
        >::max()) {
215 uniform_int_distribution<unsigned long long> dist_ab(min, max);
216 return dist_ab(rng);
217 }
218
219 string get_string() {
220 size_t string_size = get_number(1,256);
221 string string;
222 string.resize(string_size);
223 for (size_t i = 0; i < string_size; ++i) {
224 string[i] = 'a' + get_number(0,25);
225 }
226 return string;
227 }
228
229 vector<unsigned char> convert_string(const string& s) {
230 vector<unsigned char> vector(s.size());
231 for (size_t i = 0; i < s.size(); ++i) {
232 vector[i] = s[i];
233 }
234 return vector;
235 }
236
237 string convert_string(vector<unsigned char>& vec) {
```

```
string str;
str.resize(vec.size());
for (size_t i = 0; i < str.size(); ++i) {
str[i] = vec[i];
}
return str;
}
void add(char word[KEY_SIZE], unsigned long long num) {
int i = 0;
while (word[i]) {
word[i] = tolower(word[i]);
i++;
}

if (FindByKey(p,GetHash(word))==0) {
p = AddElement(p,GetHash(word),num);
} else {
}
}
void Delete(char word[KEY_SIZE]) {
int i = 0;

while (word[i]) {
word[i] = tolower(word[i]);
i++;
}
if (FindByKey(p,GetHash(word))==0) {
} else {
p = RemoveElement(p,GetHash(word));
}
}
int main() {
char cstr[KEY_SIZE];
rng.seed(std::chrono::system_clock::now().time_since_epoch().count());
size_t count;
cin >> count;

vector<pair<string, unsigned long long>> test_data(count);
ofstream test_data_log("file.test", ios::out);
{
LOG_DURATION("Generate")
for (size_t i = 0; i < count; ++i) {
test_data[i].first = (get_string());
test_data[i].second = get_number(0, numeric_limits<unsigned long long>::max());
test_data_log << test_data[i].first << " " << test_data[i].second << "\n";
}
}

cout << "AVL test\n";
```

```
287 |
288 |{
289 |LOG_DURATION("Insert time")
290 |for (size_t i = 0; i < count; ++i) {
291 |std::copy(test_data[i].first.begin(), test_data[i].first.end(), cstr);
292 |
293 |add(cstr, test_data[i].second);
294 |}
295 |}
296 |
297 |
298 |std::shuffle(test_data.begin(), test_data.end(), rng);
299 |{
300 |LOG_DURATION("Write and erase one element")
301 |for (size_t i = 0; i < count; ++i) {
302 |std::copy(test_data[i].first.begin(), test_data[i].first.end(), cstr);
303 |
304 |Delete(cstr);
305 |add(cstr, test_data[i].second);
306 |}
307 |}
308 |{
309 |LOG_DURATION("Erase time")
310 |for (size_t i = 0; i < count; ++i) {
311 |std::copy(test_data[i].first.begin(), test_data[i].first.end(), cstr);
312 |Delete(cstr);
313 |}
314 |}
315 |{
316 |LOG_DURATION("Erase empty time")
317 |for (size_t i = 0; i < count; ++i) {
318 |std::copy(test_data[i].first.begin(), test_data[i].first.end(), cstr);
319 |Delete(cstr); }
320 |}
321 |
322 |
323 |
324 |
325 |vector<pair<string, unsigned long long>> map_test(test_data.size());
326 |for (size_t i = 0; i < map_test.size(); ++i) {
327 |map_test[i] = {(test_data[i].first), test_data[i].second};
328 |}
329 |cout << "std::map test\n";
330 |map<string, unsigned long long> map;
331 |{
332 |LOG_DURATION("Insert time")
333 |for (size_t i = 0; i < count; ++i) {
334 |map[map_test[i].first] = map_test[i].second;
335 |}
```

```
}

std::shuffle(test_data.begin(), test_data.end(), rng);

{
LOG_DURATION("Write and erase one element")
for (size_t i = 0; i < count; ++i) {
map.erase(map_test[i].first);
map[map_test[i].first] = map_test[i].second;
}
}
{
LOG_DURATION("Erase time")
for (size_t i = 0; i < count; ++i) {
map.erase(map_test[i].first);
}
}
{
// LOG_DURATION("Erase empty time")
// for (size_t i = 0; i < count; ++i) {
// map.Erase(test_data[i].first);
// }
}

return 0;
}
```

1 Протокол тестирования производительности

```
sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/da2$ ./test1
1000000
Generate: 12222 ms
AVL test
Insert time: 4564 ms
Write and erase one element: 10648 ms
Erase time: 5066 ms
Erase empty time: 3521 ms
std::map test
Insert time: 3632 ms
Write and erase one element: 7176 ms
Erase time: 4811 ms
```

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я научился эффективнее использовать функции заголовочного файла chrono. Также я узнал о строении и тонкостях реализации структуры данных AVL-дерево,одной из первых бинарных деревьев, где вставка и удаление занимают $O(log(n))$ написав бенчмарк.

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] Роберт Седжвик. *Фундаментальные алгоритмы, 3-я редакция.* — Издательский дом «ДиаСофт», 2001. Перевод с английского: С. Н. Козлов, Ю. Н. Артеменко, О. А. Шадрин — 688 с. (ISBN 966-793-89-5(рус.))

[3] *Лекции по курсу «Дискретный анализ» МАИ.*

[4] Dinesh P. Mehta, Sartaj Sahni. *Handbook of Data Structures and Applications, 2nd Edition.* — Chapman and Hall/CRC, 2018. (ISBN 9781498701853).