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. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
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.

Различия вариантов заключаются только в используемых структурах данных:

Вариант 1:AVL-дерево.

Примеры:

Входные данные:

+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
− A
a

Результат работы:

OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord

1 Описание

Требуется написать реализацию структуры AVL дерева, которое будет хранить введённые значения,а так же запрограммировать интерфейс для взаимодействия пользователя с программой-словарём в соответствии с заданием.
Основная идея AVL дерева — это самобалансирующееся двоичное дерево поиска, в котором для каждого узла хранится разница высот его поддеревьев (баланс).
При этом: Каждый узел хранит свою высоту, а баланс вычисляется как разница высот правого и левого поддеревьев. Дерево поддерживает баланс таким образом, что для любого узла разница высот его поддеревьев не превышает 1 ($|\text{баланс}| \leq 1$).

Операции вставки, удаления и поиска выполняются за O(log n) в худшем случае, поскольку дерево всегда остается сбалансированным.

Балансировка достигается за счет поворотов (левых, правых, а также комбинированных), которые выполняются при нарушении условия AVL.

2 Исходный код

Программа считывает значения из файла, подающегося на поток, и выполняет действие в соответствии с заданием т. е. добавляет элемент в словарь или удаляет его оттуда, или находит элемент по ключу и выводит его значение, так же программа может сохранить содержимое словаря в файл или загрузить из файла словарь(предполагается что файл взаранее сохранён программой).

На каждой непустой строке входного файла располагается тройка «команда-ключ-значение» или ключ по которому надо найти элемент в словаре поэтому создадим новую структуру $TNode$, в которой будем хранить ключ и значение и параметры нужные программе для словаря. Для хранения пар создадим структуру (дерево) $TAVL$. Для взаимодействия пользователя со словарём напишем интерфейс в $main$. После выполнения каждой команды программа выводит сообщение в соответствии с заданием.

main.cpp:

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <fstream>

const int KEY_SIZE = 256;
using TKey = char;
using TValue = unsigned long long;

struct TAVL {



   struct TNode { //
       TNode() {
           Left = Right = nullptr;
           Height = 1;
       }
       TNode(TKey* key, TValue value) {
           Key = StyleKey(key);
           Value = value;
           Left = Right = nullptr;
           Height = 1;
       }
       ~TNode() {
           delete[] Key;
       }
       TKey* StyleKey(TKey* key) { //
           if (key == nullptr) {
               return nullptr;
           }
```

```
32              char* sub = new TKey[KEY_SIZE + 1];
33              memcpy(sub, key, strlen(key));
34              memset(sub + strlen(key), '\0', KEY_SIZE + 1 - strlen(key));
35              return sub;
36          }
37          TKey* Key = nullptr;
38          TValue Value = 0;
39          unsigned char Height;
40          TNode* Left = nullptr;
41          TNode* Right = nullptr;
42      };
43
44      TNode* Root; //
45
46
47      TAVL() {
48          Root = nullptr;
49      }
50
51      ~TAVL() {
52          DestroyTree(Root);
53      }
54
55      void DestroyTree(TNode* root) { //
56          if (root != nullptr) {
57              DestroyTree(root->Left);
58              DestroyTree(root->Right);
59              delete root;
60          }
61      }
62
63      int CompareKeys(TKey* lhs, TKey* rhs) { //
64          return strcmp(lhs, rhs);
65      }
66
67      unsigned char GetHeight(TNode* p) {
68          return p ? p->Height : 0;
69      }
70
71      int CheckHight(TNode* p) {
72          return GetHeight(p->Right) - GetHeight(p->Left);
73      }
74
75      void ReHeight(TNode* p) {
76          unsigned char hl = GetHeight(p->Left);
77          unsigned char hr = GetHeight(p->Right);
78          p->Height = (hl > hr ? hl : hr) + 1;
79      }
80
```

```
81      TNode* RotR(TNode* p) { //    p
82          TNode* q = p->Left;
83          p->Left = q->Right;
84          q->Right = p;
85          ReHeight(p);
86          ReHeight(q);
87          return q;
88      }
89
90      TNode* RotL(TNode* q) { //    q
91          TNode* p = q->Right;
92          q->Right = p->Left;
93          p->Left = q;
94          ReHeight(q);
95          ReHeight(p);
96          return p;
97      }
98
99      TNode* Balance(TNode* p) { //   p
100         ReHeight(p);
101         if (CheckHight(p) == 2)
102         {
103             if (CheckHight(p->Right) < 0)
104                 p->Right = RotR(p->Right);
105             return RotL(p);
106         }
107         if (CheckHight(p) == -2)
108         {
109             if (CheckHight(p->Left) > 0)
110                 p->Left = RotL(p->Left);
111             return RotR(p);
112         }
113         return p; //
114     }
115
116     TNode* Insert(TNode* p, TKey* key, TValue value) { //   k     p
117         if (p == nullptr) {
118             p = new TNode(key, value);
119             std::cout << "OK\n";
120             return p;
121         }
122         if (CompareKeys(key, p->Key) < 0) {
123             p->Left = Insert(p->Left, key, value);
124         }
125         else if (CompareKeys(key, p->Key) > 0) {
126             p->Right = Insert(p->Right, key, value);
127         }
128         else {
129             std::cout << "Exist\n";
```

```
130         }
131         return Balance(p);
132     }
133 
134     TNode* FindMin(TNode* p) { //
135         return p->Left ? FindMin(p->Left) : p;
136     }
137 
138     TNode* RemoveMin(TNode* p) { //        p
139         if (p->Left == 0)
140             return p->Right;
141         p->Left = RemoveMin(p->Left);
142         return Balance(p);
143     }
144 
145     TNode* Remove(TNode* p, TKey* key) { //   k   p
146         if (p == nullptr) {
147             std::cout << "NoSuchWord\n";
148             return nullptr;
149         }
150         if (CompareKeys(key, p->Key) < 0)
151             p->Left = Remove(p->Left, key);
152         else if (CompareKeys(key, p->Key) > 0)
153             p->Right = Remove(p->Right, key);
154         else // k == p->Key
155         {
156             TNode* q = p->Left;
157             TNode* r = p->Right;
158             delete p;
159             std::cout << "OK\n";
160             if (r == nullptr) return q;
161             TNode* min = FindMin(r);
162             min->Right = RemoveMin(r);
163             min->Left = q;
164             return Balance(min);
165         }
166         return Balance(p);
167     }
168 
169     TNode* Find(TNode* node, TKey* key) { //
170         if (node == nullptr) {
171             return nullptr;
172         }
173         if (CompareKeys(node->Key, key) > 0) {
174             return Find(node->Left,key);
175         }
176         else if (CompareKeys(node->Key, key) < 0) {
177             return Find(node->Right,key);
178         }
```

```
179         else {
180             return node;
181         }
182         return nullptr;
183     }
184 
185     void PrintFindRes(TKey* key) { //
186         TNode* res = Find(Root, key);
187         if (res != nullptr) {
188             std::cout << "OK: " << res->Value << std::endl;
189         }
190         else {
191             std::cout << "NoSuchWord\n";
192         }
193     }
194 
195     void Save(std::ostream& file, TNode* node) { //
196         if (node == nullptr) {
197             return;
198         }
199         int keySize = strlen(node->Key);
200         file.write((char*)&keySize, sizeof(int));
201         file.write(node->Key, keySize);
202         file.write((char*)&(node->Value), sizeof(TValue));
203 
204         bool hasL = (node->Left != nullptr);
205         bool hasR = (node->Right != nullptr);
206 
207         file.write((char*)&hasL, sizeof(bool));
208         file.write((char*)&hasR, sizeof(bool));
209 
210         if (hasL) {
211             Save(file, node->Left);
212         }
213         if (hasR) {
214             Save(file, node->Right);
215         }
216     }
217 
218     TNode* Load(std::istream& file, TNode* node) { //
219         TNode* root = nullptr;
220 
221         int keysize;
222         file.read((char*)&keysize, sizeof(int));
223 
224         if (file.gcount() == 0) {
225             return root;
226         }
227 
```

```
228         TKey* key = new char[keysize + 1];
229         key[keysize] = '\0';
230         file.read(key, keysize);
231 
232         TValue value;
233         file.read((char*)&value, sizeof(TValue));
234 
235         bool hasL = false;
236         bool hasR = false;
237         file.read((char*)&hasL, sizeof(bool));
238         file.read((char*)&hasR, sizeof(bool));
239 
240         root = new TNode(key,value);
241         if (hasL) {
242             root->Left = Load(file, root);
243         }
244         else {
245             root->Left = nullptr;
246         }
247 
248         if (hasR) {
249             root->Right = Load(file, root);
250         }
251         else {
252             root->Right = nullptr;
253         }
254         delete[] key;
255         return root;
256     }
257 };
258 
259 void MakeLower(TKey* key) {
260     int lengh = strlen(key);
261     for (int i = 0; i < lengh; i++) {
262         key[i] = tolower(key[i]);
263     }
264 }
265 
266 int main() {
267 
268     std::ios::sync_with_stdio(false);
269     std::cin.tie(nullptr);
270     std::cout.tie(nullptr);
271 
272     TAVL tree;
273     TKey key[KEY_SIZE + 1];
274     TValue value;
275 
276     std::ofstream ofstr;
```

```
277     std::ifstream ifstr;
278
279     while (std::cin >> key) {
280         if (key[0] == '+') {
281             std::cin >> key >> value;
282             MakeLower(key);
283             tree.Root = tree.Insert(tree.Root, key, value);
284         }
285         else if (key[0] == '-') {
286             std::cin >> key;
287             MakeLower(key);
288             tree.Root = tree.Remove(tree.Root, key);
289         }
290         else if (key[0] == '!' && strlen(key) == 1) {
291             std::cin >> key;
292             if (key[0] == 'S') {
293                 std::cin >> key;
294                 ofstr.open(key, std::ios::out | std::ios::binary);
295                 if (ofstr) {
296                     tree.Save(ofstr, tree.Root);
297                     std::cout << "OK\n";
298                 }
299                 else {
300                     std::cout << "ERROR: can't open file\n";
301                 }
302                 ofstr.close();
303             }
304             else if (key[0] == 'L') {
305                 std::cin >> key;
306                 ifstr.open(key, std::ios::in | std::ios::binary);
307                 if (ifstr) {
308                     tree.DestroyTree(tree.Root);
309                     tree.Root = tree.Load(ifstr, nullptr);
310                     std::cout << "OK\n";
311                     ifstr.close();
312                 }
313                 else {
314                     std::cout << "ERROR: can't open file\n";
315                 }
316             }
317         }
318         else {
319             MakeLower(key);
320             tree.PrintFindRes(key);
321         }
322     }
323     return 0;
324 }
```

main.cpp	
struct TNode	Структура узла, так же используется для хранения значений
struct TAVL	Структура дерева, которое я использую для организации словаря и его хранения
void DestroyTree(TNode* root)	удаление дерева
TNode* Remove(TNode* p, TKey* key)	удаление элемента из словаря
TNode* Balance(TNode* p)	Балансировка после удаления
TNode* Insert(TNode* p, TKey* key, TValue value)	Вставка ключа k в дерево с корнем p
TNode* Find(TNode* node, TKey* key)	поиск элемента в словаре по ключу
TNode* RotR(TNode* p)	Правый поворот вокруг p
TNode* RotL(TNode* q)	Левый поворот вокруг q
void ReHeight(TNode* p)	подсчёт новой высоты в узле
int CheckHight(TNode* p)	проверка высоты в узле
unsigned char GetHeight(TNode* p)	получение высоты узла
int CompareKeys(TKey* lhs, TKey* rhs)	сравнение ключей
TNode* FindMin(TNode* p)	Поиск узла с минимальным ключом в дереве
TNode* RemoveMin(TNode* p)	Удаление узла с минимальным ключом из дерева p
void PrintFindRes(TKey* key)	Печать результатов поиска
void Save(std::ostream& file, TNode* node)	Сохранение в файл
TNode* Load(std::istream& file, TNode* node)	Загрузка из файла
void MakeLower(TKey* key)	приведение к нижнему регистру

Листинг структуры TTree:

```
struct TTree
{
    TNode* Root;
};
```

Листинг структуры TNode:

```
struct TNode
{
    TKey* Key = nullptr;
    TValue Value = 0;
    unsigned char Height;
    TNode* Left = nullptr;
    TNode* Right = nullptr;
};
```

3 Консоль

```
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ g++ -o p main.cpp
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ ./p <test2.txt
OK
OK: 155
OK
OK: 18446744073709551615
OK: 1
NoSuchWord
OK
OK: 155
OK
```

4 Тест производительности

Тест производительности представляет из себя следующее: Вставка большого количества значений из файла с тестовыми значениями и поиск по словарю затем удаление этих значений программа выполняется дважды первый раз с использованием стандартных функций второй - реализация из лабораторной работы

```
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ g++ -o p bench2.cpp
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ ./p
Starting AVL Tree vs std::map Benchmark 1000
AVL Tree Insert: 0.000937669 s
AVL Tree Search: 0.000351823 s
AVL Tree Remove: 0.000626979 s
std::map Insert: 0.000944942 s
std::map Search: 0.000602891 s
std::map Remove: 0.00075356 s
Benchmark completed.
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ g++ -o p bench2.cpp
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ ./p
Starting AVL Tree vs std::map Benchmark 100000
AVL Tree Insert: 0.205597 s
AVL Tree Search: 0.100184 s
AVL Tree Remove: 0.161481 s
std::map Insert: 0.147301 s
std::map Search: 0.106993 s
std::map Remove: 0.13575 s
Benchmark completed.
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ g++ -o p bench2.cpp
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ ./p
Starting AVL Tree vs std::map Benchmark 1000000
AVL Tree Insert: 4.08867 s
AVL Tree Search: 2.23604 s
AVL Tree Remove: 3.40057 s
std::map Insert: 2.19488 s
std::map Search: 1.86108 s
std::map Remove: 2.31803 s
Benchmark completed.
```

5 Работа с профилировщиком и Valgrind

Работа с профилировщиком и Valgrind представляет из себя следующее: компилирование программы с ключом -pg запуск и создание файла профилировщика с последующим преобразованием его в текстовый файл. Gprof генерирует подробные отчёты, которые показывают количество времени, затраченного на каждую функцию, а также вызовы между функциями.
Затем проверка программы на наличие утечек памяти при помощи Valgrind
Вывод консоли:

```
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ g++ -pg -o p main.cpp
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ ./p <tests/test100_1000.txt
>out.txt
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ gprof p gmon.out
>anal.txt
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2$ valgrind ./p <tests/test100_
>out.txt
==875== Memcheck,a memory error detector
==875== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==875== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==875== Command: ./p
==875==
==875==
==875== Process terminating with default action of signal 27 (SIGPROF)
==875==    at 0x4C278B2: __open_nocancel (open64_nocancel.c:39)
==875==    by 0x4C3785F: write_gmon (gmon.c:393)
==875==    by 0x4C3820A: _mcleanup (gmon.c:467)
==875==    by 0x4B52381: __cxa_finalize (cxa_finalize.c:82)
==875==    by 0x10A556: ??? (in /mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025/lab2/p)
==875==    by 0x40010F1: _dl_call_fini (dl-call_fini.c:43)
==875==    by 0x4005577: _dl_fini (dl-fini.c:114)
==875==    by 0x4B52A75: __run_exit_handlers (exit.c:108)
==875==    by 0x4B52BBD: exit (exit.c:138)
==875==    by 0x4B351D0: (below main) (libc_start_call_main.h:74)
==875==
==875== HEAP SUMMARY:
==875==     in use at exit: 0 bytes in 0 blocks
==875==   total heap usage: 208 allocs,208 frees,251,260 bytes allocated
==875==
```

```
==875==    All heap blocks were freed --no leaks are possible
==875==
==875== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Profiling timer expired
```

6 Выводы

В результате выполнения лабораторной работы по курсу «Дискретный анализ», были получены навыки использования бинарных деревьев поиска, работы с утилитой valgrind и закреплены навыки использования классов и ввода(вывода) в(из) файла(ов). Так же в ходе лабораторной работы было проведено сравнение самописной реализации AVL-дерева и стандартного std::map (который обычно реализован как красно-чёрное дерево).

Основные результаты На малых данных (10 000–100 000 элементов):

Разница в скорости незначительна, иногда AVL-дерево даже немного быстрее из-за более строгой балансировки.

Это объясняется тем, что AVL-дерево гарантирует меньшую высоту, поэтому поиск может выполняться за меньшее число сравнений.

На больших данных (1 000 000+ элементов):

std::map оказался быстрее как при вставке, так и при удалении.

Причина в том, что:

Красно-чёрное дерево требует меньше перебалансировок при вставке и удалении.

STL (std::map) использует оптимизированные аллокаторы памяти, что уменьшает накладные расходы.

Поиск в AVL-дереве иногда быстрее

Поскольку AVL строже балансируется, теоретически поиск должен быть чуть быстрее, чем в std::map.

Однако на практике разница может нивелироваться из-за:

Накладных расходов на указатели (char* vs std::string).

Лучшей оптимизации std::map под конкретный процессор.

Список литературы

[1] *АВЛ-дерево - Википедия*
URL:https://ru.wikipedia.org/wiki/АВЛ-дерево
(дата обращения: 15.04.2025).

[2] *Лекция 33: Решение задач на динамические структуры данных*
URL: https://intuit.ru/studies/courses/648/504/lecture/11459?page=2
(дата обращения: 15.04.2025).

[3] *Хабр:АВЛ-деревья*
URL: https://habr.com/ru/post/330644/(дата обращения: 15.04.2025).

[4] *АВЛ-дерево*
URL: https://blog.skillfactory.ru/glossary/avl-derevo/?ysclid=mabc6yorcn251331466
(дата обращения: 15.04.2025).