

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. Постнов  
Преподаватель: С. А. Михайлова  
Группа: М8О-201Б-21  
Дата:  
Оценка:  
Подпись:

Москва, 2023

## 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 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:

**AVL-дерево.**

# 1 Описание

АВЛ-дерево (англ. AVL-Tree) — сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ-деревья названы по первым буквам фамилий их изобретателей, Г. М. Адельсона-Вельского и Е. М. Ландиса, которые впервые предложили использовать АВЛ-деревья в 1962 году.

АВЛ-дерево — это прежде всего двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Это значит, что для поиска нужного ключа в АВЛ-дереве можно использовать стандартный алгоритм.

Особенностью АВЛ-дерева является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.

Традиционно, узлы АВЛ-дерева хранят не высоту, а разницу высот правого и левого поддеревьев (так называемый *balance factor*), которая может принимать только три значения -1, 0 и 1.

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда *balance factor* некоторых узлов оказывается равными 2 или -2, т.е. возникает расбалансировка поддерева. Для выправления ситуации применяются хорошо нам известные повороты вокруг тех или иных узлов дерева.

Простой поворот вправо (влево) производит следующую трансформацию дерева:

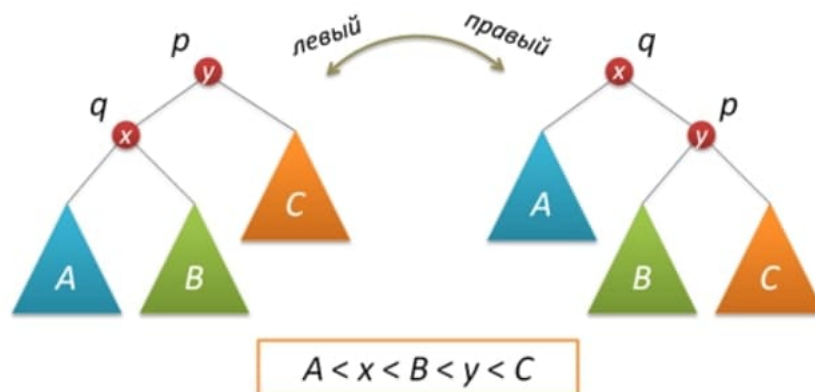


Рис. 1:

Рассмотрим теперь ситуацию дисбаланса, когда высота правого поддерева узла  $p$

на 2 больше высоты левого поддерева (обратный случай является симметричным и реализуется аналогично). Пусть  $q$  — правый дочерний узел узла  $p$ , а  $s$  — левый дочерний узел узла  $q$ .

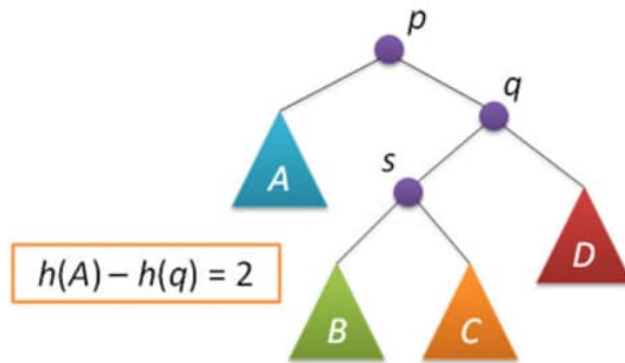


Рис. 2:

Анализ возможных случаев в рамках данной ситуации показывает, что для исправления расбалансировки в узле  $p$  достаточно выполнить либо простой поворот влево вокруг  $p$ , либо так называемый большой поворот влево вокруг того же  $p$ . Простой поворот выполняется при условии, что высота левого поддерева узла  $q$  больше высоты его правого поддерева:  $h(s) \leq h(D)$ .

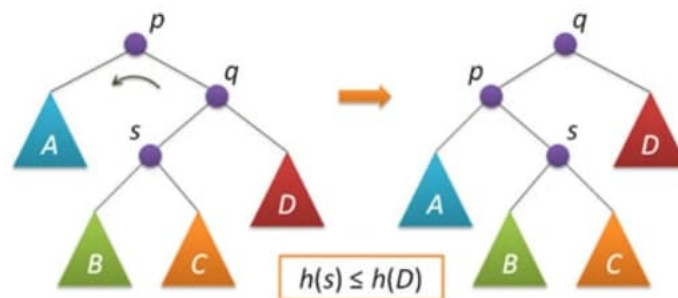


Рис. 3:

Большой поворот применяется при условии  $h(s) > h(D)$  и сводится в данном случае к двум простым — сначала правый поворот вокруг  $q$  и затем левый вокруг  $p$ .

Вставка нового ключа в АВЛ-дерево выполняется, по большому счету, так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое

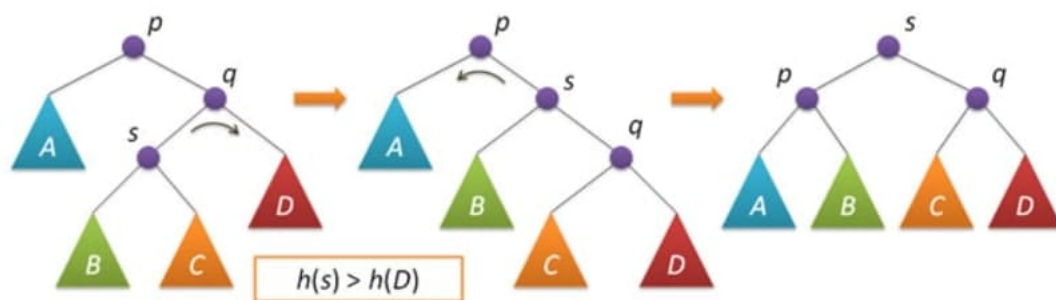


Рис. 4:

или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как ключ вставлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла.

Удаление. Идея следующая: находим узел  $p$  с заданным ключом  $k$  (если не находим, то делать ничего не надо), в правом поддереве находим узел  $\min$  с наименьшим ключом и заменяем удаляемый узел  $p$  на найденный узел  $\min$ . Вызываем балансировку узла.

## 2 Исходный код

Структура программы состоит из 2 файлов:

1. *avl.hpp* В нем содержится объявление и реализация класса АВЛ-дерева.
2. *main\_.cpp* Обработка команд.

```
1  #ifndef AVL_H
2  #define AVL_H
3
4
5  #include <algorithm>
6  #include <cstdint>
7  #include <fstream>
8  #include <iostream>
9  #include <stdexcept>
10 #include <string>
11 #include <utility>
12
13 class TAVLTree {
14     private:
15     struct TNode {
16         std::string key;
17         unsigned long long value;
18         int height;
19         TNode* left;
20         TNode* right;
21         TNode(std::string& key, unsigned long long value) {
22             this->key = std::move(key);
23             this->value = value;
24             left = nullptr;
25             right = nullptr;
26             height = 1;
27         }
28     };
29
30     TNode* root;
31
32     int Height(TNode* node) // 0(1) node->height
33
34     int BFactor(TNode* node); // balance factor 0(1) node->right->height - node->left->
        height
35
36     void FixHeight(TNode* node); // 0(1)
37
38     TNode* RotateRight(TNode* node); // 0(1)
39
40     TNode* RotateLeft(TNode* node); // 0(1)
41
42     TNode* Balance(TNode* node); // 0(1)
```

```

43 |
44 |     TNode* Insert(TNode* node, std::string& key, unsigned long long value); O(log n)
45 |
46 |     TNode* Find(TNode* node, std::string& key); // O(log n)
47 |
48 |     TNode* FindMin(TNode* node); // O(log n)
49 |
50 |     TNode* RemoveMin(TNode* node); // O(log n)
51 |
52 |     TNode* Erase(TNode* node, std::string& key); // O(log n)
53 |
54 |     void Destroy(TNode*& node); // O(n)
55 |
56 |     void Destroy(); // O(n)
57 |
58 |     size_t Size(TNode* node); // O(n)
59 |
60 |     size_t HeightTree(TNode* node); // O(1) node->height
61 |
62 |     void SaveToFile(std::ofstream& file, TNode* node); // O(n)
63 |
64 |     public:
65 |
66 |     TAVLTree();
67 |
68 |     ~TAVLTree();
69 |
70 |     void Insert(std::string& key, unsigned long long value); // adding a node O(log n)
71 |
72 |     void Erase(std::string& key); // deleting a node O(log n)
73 |
74 |     std::pair<unsigned long long, bool> Exist(std::string& key); // O(log n)
75 |
76 |     size_t Size(); // count of nodes O(n)
77 |
78 |     void Clear(); // clear tree O(n)
79 |
80 |     bool Empty(); // checking for emptiness O(1)
81 |
82 |     size_t HeightTree(); // height of tree O(1)
83 |
84 |     void SaveToFile(std::ofstream& file); // O(n)
85 |
86 |     void LoadFromFile(std::ifstream& file); // O(n * log n)
87 |
88 | }
89 | #endif

```

```

1 | #include <iostream>
2 | #include <string>

```

```

3 | #include <algorithm>
4 |
5 | #include "avl.hpp"
6 |
7 |
8 | int main() {
9 |     TAVlTree avl;
10 |    std::string command;
11 |    while (std::cin >> command) {
12 |        if (command == "+") {
13 |            std::string key;
14 |            unsigned long long value;
15 |            std::cin >> key >> value;
16 |            std::transform(key.begin(), key.end(), key.begin(),
17 |                [](unsigned char symb){ return std::tolower(symb); });
18 |            if (avl.Exist(key).second) {
19 |                std::cout << "Exist\n";
20 |            } else {
21 |                avl.Insert(key, value);
22 |                std::cout << "OK\n";
23 |            }
24 |        }
25 |        else if (command == "-") {
26 |            std::string key;
27 |            std::cin >> key;
28 |            std::transform(key.begin(), key.end(), key.begin(),
29 |                [](unsigned char symb){ return std::tolower(symb); });
30 |            if (!avl.Exist(key).second) {
31 |                std::cout << "NoSuchWord\n";
32 |            } else {
33 |                avl.Erase(key);
34 |                std::cout << "OK\n";
35 |            }
36 |        }
37 |        else if (command == "!") {
38 |            std::string action;
39 |            std::cin >> action;
40 |            std::string path;
41 |            std::cin >> path;
42 |            if (action == "Save") {
43 |                std::ofstream file(path, std::ios::binary | std::ios::trunc);
44 |                auto size = avl.Size();
45 |                file.write(reinterpret_cast<char*>(&size), sizeof(unsigned long long));
46 |                if (size > 0) {
47 |                    avl.SaveToFile(file);
48 |                }
49 |                std::cout << "OK\n";
50 |                file.close();
51 |            }

```



```

52     } else {
53         std::ifstream file(path, std::ios::binary);
54         avl.LoadFromFile(file);
55         std::cout << "OK\n";
56         file.close();
57     }
58 }
59
60 } else {
61     std::transform(command.begin(), command.end(), command.begin(),
62         [](unsigned char symb){ return std::tolower(symb); });
63     auto node = avl.Exist(command);
64     if (node.second) {
65         std::cout << "OK: " << node.first << "\n";
66     } else {
67         std::cout << "NoSuchWord\n";
68     }
69 }
70 }
71 }

```

### 3 Консоль

```
[alex@fedora mai-da-labs]$ cd build
[alex@fedora build]$ cmake ../
--Configuring done (0.3s)
--Generating done (0.0s)
--Build files have been written to: /home/alex/mai-da-labs/build
[alex@fedora build]$ cmake --build .
[2/2] Linking CXX executable lab2_avl/lab2_avl
[alex@fedora build]$ cd lab2_avl
[alex@fedora lab2_avl]$ ls
CMakeFiles  cmake_install.cmake  CTestTestfile.cmake  lab2_avl  test.txt
[alex@fedora lab2_avl]$ cat test.txt
+ a 1
+ A 2
+ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
18446744073709551615
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
A
-A
a
[alex@fedora lab2_avl]$ ./lab2_avl <test.txt
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

## 4 Тесты

Чтобы убедиться, что AVL реализовано корректно, я 5000 раз добавлял элемент и потом убирал их в случайном порядке, и сравнивал количество элементов дерева с ожидаемым количеством. А также, чтобы проверить, что дерево сбалансированное, я вывел высоту дерева из  $10^6$  элементов.

Производительность своей реализации AVL, буду сравнивать с `std::map`

```
1  #include <iostream>
2
3  #include <gtest/gtest.h>
4
5
6  #include <algorithm>
7
8  #include <chrono>
9  #include <random>
10
11
12  #include "avl.hpp"
13
14
15  std::mt19937 randomKey(std::chrono::steady_clock::now().time_since_epoch().count());
16
17
18  TEST(AvlTest, MyTest) {
19      TAVlTree avl;
20      avl.Insert(1);
21      avl.Insert(2);
22      EXPECT_EQ(2, avl.Size());
23      avl.Clear();
24      EXPECT_EQ(0, avl.Size());
25  }
26
27  TEST(AvlTest, CommonTest) {
28      size_t const countActions = 5000;
29      TAVlTree avl;
30      std::vector<int> keys;
31      for (size_t i = 0; i < countActions; ++i) {
32          int key = static_cast<int>(randomKey());
33          avl.Insert(key);
34          keys.push_back(key);
35          EXPECT_EQ(i + 1, avl.Size());
36      }
37      std::random_device randomDevice;
38      std::mt19937 perm(randomDevice());
39      std::shuffle(keys.begin(), keys.end(), perm);
40      for (size_t i = 0; i < countActions; ++i) {
```

```

41     int key = keys[i];
42     EXPECT_EQ(avl.Exist(key), true);
43     avl.Erase(key);
44     EXPECT_EQ(countActions - i - 1, avl.Size());
45 }
46 EXPECT_EQ(avl.Empty(), true);
47 }
48
49 TEST(AvlTest, HeightTest) {
50     size_t const countNodes = 1000000;
51     TAVlTree avl;
52     for (size_t i = 0; i < countNodes; ++i) {
53         int key = static_cast<int>(randomKey());
54         avl.Insert(key);
55     }
56     std::cout << "Height: " << avl.HeightTree() << std::endl;
57 }

1  #include <cstdint>
2  #include <iostream>
3
4  #include <gtest/gtest.h>
5
6
7  #include <algorithm>
8
9  #include <chrono>
10 #include <random>
11 #include <string>
12
13
14 #include "avl.hpp"
15
16
17 std::mt19937 randomKey(std::chrono::steady_clock::now().time_since_epoch().count())
18     ;
19
20 TEST(AvlTest, Benchmark) {
21     const size_t countOfActions = 1000000;
22     TAVlTree avl;
23     std::map<std::string, unsigned long long> dict;
24     std::vector<std::pair<std::string, unsigned long long>> dataInsert;
25     std::vector<std::pair<std::string, unsigned long long>> dataErase;
26     for (size_t i = 0; i < countOfActions; ++i) {
27         std::string key = std::to_string(randomKey());
28         unsigned long long value = randomKey();
29         dataInsert.emplace_back(key, value);
30         dataErase.emplace_back(key, value);
31     }

```

```

32     std::random_device randomDevice;
33     std::mt19937 perm(randomDevice());
34     std::shuffle(dataErase.begin(), dataErase.end(), perm);
35
36
37     auto begin = std::chrono::high_resolution_clock::now();
38     for (size_t i = 0; i < countOfActions; ++i) {
39         auto key = dataInsert[i].first;
40         auto value = dataInsert[i].second;
41         avl.Insert(key, value);
42     }
43     for (size_t i = 0; i < countOfActions; ++i) {
44         auto key = dataInsert[i].first;
45         avl.Erase(key);
46     }
47     auto end = std::chrono::high_resolution_clock::now();
48     auto timeAVL = std::chrono::duration_cast<std::chrono::milliseconds>(end -
49         begin).count();
50
51     begin = std::chrono::high_resolution_clock::now();
52     for (size_t i = 0; i < countOfActions; ++i) {
53         auto key = dataInsert[i].first;
54         auto value = dataInsert[i].second;
55         dict[key] = value;
56     }
57     for (size_t i = 0; i < countOfActions; ++i) {
58         auto key = dataInsert[i].first;
59         dict.erase(key);
60     }
61     end = std::chrono::high_resolution_clock::now();
62     auto timeMAP = std::chrono::duration_cast<std::chrono::milliseconds>(end -
63         begin).count();
64
65     std::cout << "AVL time: " << timeAVL << " ms\n";
66     std::cout << "std::map time: " << timeMAP << " ms\n";
67
68 }

```

4: Test command: /home/alex/mai-da-labs/build/tests/avl\_test

4: Working Directory: /home/alex/mai-da-labs/build/tests

4: Test timeout computed to be: 10000000

4: Running main() from /home/alex/mai-da-labs/build/\_deps/googletest-src/googletest/s

4: [=====] Running 3 tests from 1 test suite.

```

4: [-----] Global test environment set-up.
4: [-----] 3 tests from AvlTest
4: [ RUN      ] AvlTest.MyTest
4: [          OK ] AvlTest.MyTest (0 ms)
4: [ RUN      ] AvlTest.CommonTest
4: [          OK ] AvlTest.CommonTest (149 ms)
4: [ RUN      ] AvlTest.HeightTest
4: Height: 37
4: [          OK ] AvlTest.HeightTest (1727 ms)
4: [-----] 3 tests from AvlTest (1877 ms total)
4:
4: [-----] Global test environment tear-down
4: [=====] 3 tests from 1 test suite ran. (1877 ms total)
4: [ PASSED   ] 3 tests.
1/1 Test #4: avl_test ..... Passed    1.88 sec

```

The following tests passed:  
avl\_test

100% tests passed,0 tests failed out of 1

```

Total Test time (real) = 1.89 sec
[alex@fedora build]$ ctest -V -R lab2_test
UpdateCTestConfiguration from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
UpdateCTestConfiguration from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
Test project /home/alex/mai-da-labs/build
Constructing a list of tests
Done constructing a list of tests
Updating test list for fixtures
Added 0 tests to meet fixture requirements
Checking test dependency graph...
Checking test dependency graph end
test 5
Start 5: lab2_test

5: Test command: /home/alex/mai-da-labs/build/tests/lab2_test
5: Working Directory: /home/alex/mai-da-labs/build/tests
5: Test timeout computed to be: 10000000
5: Running main() from /home/alex/mai-da-labs/build/_deps/googletest-src/googletest/s
5: [=====] Running 1 test from 1 test suite.
5: [-----] Global test environment set-up.

```

```

5: [-----] 1 test from AvlTest
5: [ RUN      ] AvlTest.Benchmark
5: AVL time: 4025 ms
5: std::map time: 2683 ms
5: [      OK   ] AvlTest.Benchmark (7091 ms)
5: [-----] 1 test from AvlTest (7091 ms total)
5:
5: [-----] Global test environment tear-down
5: [=====] 1 test from 1 test suite ran. (7091 ms total)
5: [  PASSED  ] 1 test.
1/1 Test #5: lab2_test ..... Passed    7.10 sec

```

The following tests passed:  
lab2\_test

100% tests passed,0 tests failed out of 1

Total Test time (real) = 7.10 sec

Как видно, *std :: map* выиграл у *AVL*, так как *std :: map* реализовано с помощью КЧД, и, возможно, операция балансировки вызывается реже.

## 5 Выводы

В ходе выполнения лабораторной работы по курсу «Дискретный анализ» я изучил сбалансированные деревья поиска и более подробно AVL дерево. Изучил как сохранять информацию в файл в бинарном формате. А также (снова) возникли ML и TL, TL я решил тем, что передавал строки по ссылке(избежал лишних копирований), а ML я получил из-за плохой подсказки линтера, он мне порекомендовал использовать `std::move`, что как выяснилось ошибочно.



## 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] *АВЛ-дерево — Вики-конспекты ИТМО*.  
URL: <https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево> (дата обращения: 23.04.2023).