Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

«Реализация таблиц для хранения арифметических полиномов»

**Выполнил:** студент ИИТММ гр. 0823-1

Романов Александр Анатольевич

**Проверил:** Ассистент каф. МОСТ

Пирова Анна Юрьевна

Нижний Новгород

2017 год

Оглавление

[Введение 3](#_Toc483793645)

[Формулировка задачи 4](#_Toc483793646)

[Руководство пользователя 5](#_Toc483793647)

[Руководство программиста 6](#_Toc483793648)

[Описание структуры программного комплекса 8](#_Toc483793649)

[Описание алгоритмов 9](#_Toc483793650)

[Заключение 11](#_Toc483793651)

[Литература 12](#_Toc483793652)

# Введение

В современном мире время слишком дорого, чтобы позволить себе использовать для решения задач только массивы – процедуры вставки и удаления весьма проблематичны. Поэтому были придуманы такие структуры данных, как хеш-таблицы и бинарные деревья поиска.

 **Хеш-таблица** — это [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), реализующая интерфейс [ассоциативного массива](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

**Двоичное дерево поиска** — это [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

* Оба поддерева — левое и правое — являются двоичными деревьями поиска.
* У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X.
* У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равно*, нежели значение ключа данных самого узла X.

**АВЛ-дерево** — сбалансированное по высоте [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0): для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

# Формулировка задачи

Лабораторная работа заключается в реализации программного комплекса для хранения арифметических полиномов в просматриваемых, сортируемых, хеш-таблицах и AVL-деревьях.

Необходимо реализовать классы, позволяющие искать, вставлять и удалять полиномы в структурах данных.

В просматриваемых таблицах поиск осуществляется за О(N), вставка и удаление -- за константу.

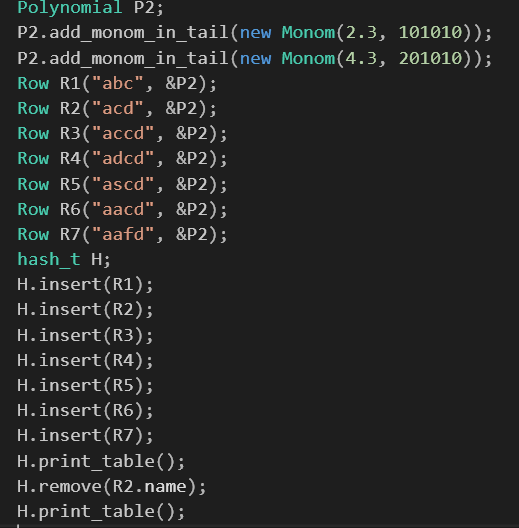
В сортируемых таблицах поиск осуществляется за О(logN), вставка и удаление -- за константу.

В AVL-деревьях все три операции реализуются за О(logN).

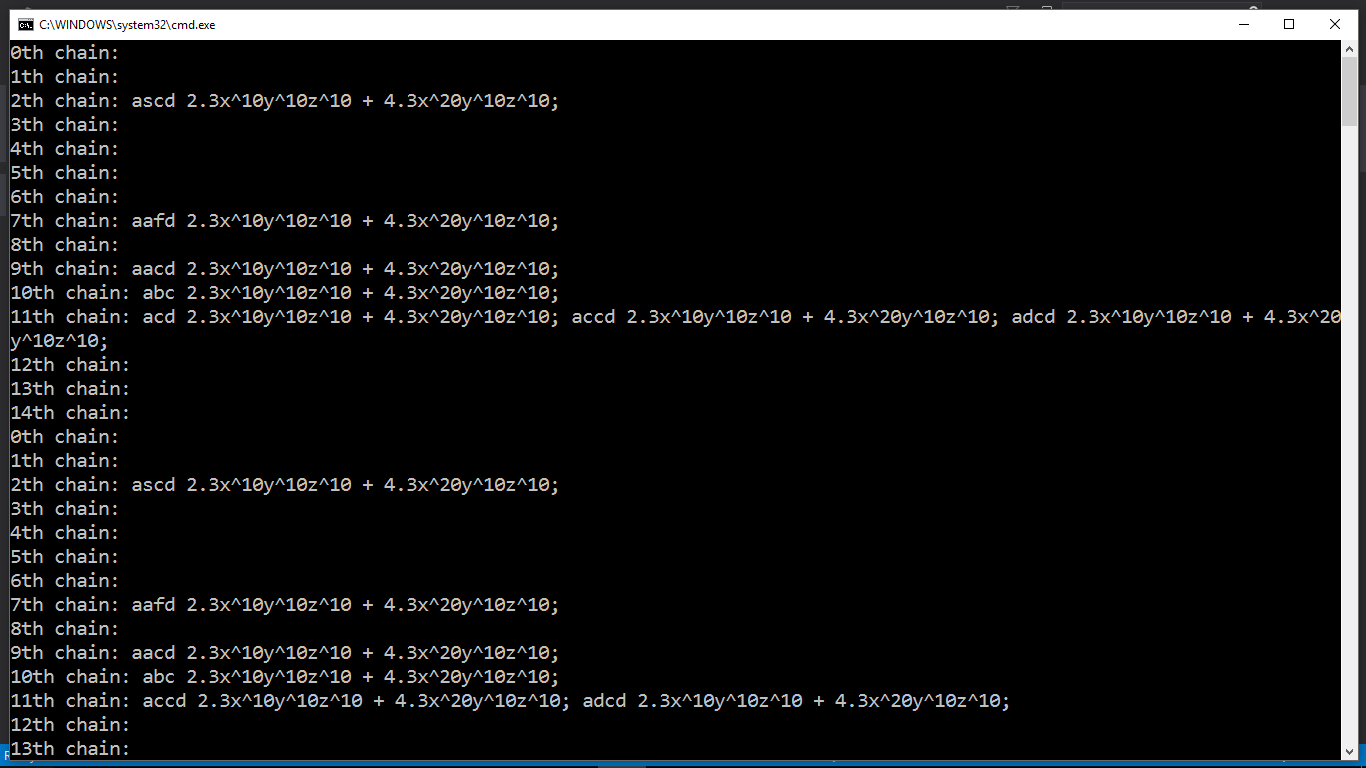
# Руководство пользователя

В файле main.cpp, находящемся в проекте main\_proj, представлены три примера работы программного комплекса.

Ниже представлены возможности касса hash\_t по размещению элементов в таблицу и избеганию коллизий с помощью метода цепочек:



Инициализация строк полиномами и именами



# Руководство программиста

**Описание структур данных:**

Структуры:

struct Monom. Содержит поля для хранения степени монома, коэффициента монома и указателя на последующий моном.

struct Row. Содержит поля с именем полинома, указатель на полином.

struct Node\_tree. Содержит указатель на Row, указатели на «левый» и «правый» элемент Node\_tree и поле, хранящее высоту дерева.

Классы:

class Polynomial. Содержит указатели на начало и конец полинома, количество мономов в нём и методы, позволяющие производить арифметические вычисления с полиномами.

class Table. Абстрактный класс, содержащий поле, хранящее количество строк в таблице и четыре виртуальных метода для поиска, вставки, удаления и печати строк таблицы.

Классы viewed\_tables, sorted\_t и hash\_t являются наследниками класса Table. В каждом классе-потомке переопределяются четыре функции базового класса.

class btree. Содержит указатель \*Node\_tree, являющийся корнем дерева. Также содержит методы поиска, вставки, удаления, балансировки дерева, правых и левых поворотов и проч.

Проекты:

gtest.

Ниже представлены декларации некоторых из вышеперечисленных структур данных:

struct Row {

public:

string name;

Polynomial\* ptr\_p;

bool is\_\_empty;

inline Row() { name = "emp", ptr\_p = nullptr; is\_\_empty = true; }

Row(string n, Polynomial\* p);

Row& operator =(const Row& n);

Row(const Row& r);

bool operator==(const Row& r);

friend ostream& operator << (ostream& os, const Row& row);

~Row();

};

class Table {

public:

int filled\_rows;

virtual void insert(Row& r) = 0;

virtual void remove(string name) = 0;

virtual Row\* search(string name) = 0;

virtual void print\_table() = 0;

};

class Polynomial {

private:

Monom\* head;

Monom\* tail;

int size; // counter of amount of monoms

public:

Monom\* get\_head() { return head; }

Monom\* get\_tail() { return tail; }

Polynomial();

Polynomial(const Polynomial& p);

void Input\_poly();

void bubble\_sort();

void swap\_(Monom\* m1, Monom\* m2);

void add\_monom\_in\_tail(Monom\* m);

void del\_monom(Monom \*ptr2);

const Polynomial& operator = (const Polynomial &p);

Polynomial operator + (Polynomial p);

Polynomial operator - (Polynomial m);

Polynomial operator \* (double d);

Polynomial operator \* (Monom m);

Polynomial operator \* (Polynomial m);

void Print\_poly();

void bring\_similar();

bool operator==(const Polynomial& rhs) const;

~Polynomial();

friend ostream& operator << (ostream& os, const Polynomial& p);

};

# Описание структуры программного комплекса

Решение Lab\_Tables состоит из шести проектов:

1. gtest. В проекте находятся google-тесты.
2. main\_proj. Создавался как главный main для демонстрации.
3. Monom. В проекте находится реализация структуры Monom.
4. Polynomials. В проекте находится реализация структуры Polynomial.
5. Tabl\_proj. В проекте находятся реализации структуры Row и абстрактного класса Table.
6. tests. В проекте находится набор тестов.

Декларации и реализации структур данных разделены и содержатся соответственно в файлах \_\_name\_\_ .h и \_\_name\_\_ .cpp.

# Описание алгоритмов

Нерекурсивный алгоритм поиска строки в AVL-дереве.

Row\* btree::search(string t)

{

Node\_tree\* tmp = root;

if (!tmp->row\_) { return NULL; }

if (!tmp)

{

while (tmp->row\_->name != t) //? && tmp

{

if (tmp->row\_->name > t)

{

if (tmp->left)

tmp = tmp->left;

else {tmp = NULL; break;}

}

else

{

if (tmp->right)

tmp = tmp->right;

else { tmp = NULL; break; }

}

}

}

if (tmp->row\_->name == t) {return tmp->row\_;}

else return nullptr;

}

Функция получает на вход имя полинома. После этого создаётся локальная переменная tmp для хранения текущего узла дерева. Далее, пока имя полинома в текущем узле не совпадёт с искомым именем, tmp передвигается влево или вправо в зависимости от результата сравнения искомого имени с текущим.

Функция возвращает указатель на строку с искомым именем или nullptr, если оно не найдено.

Алгоритм удаления строки из хеш-таблицы. (Метод устранения коллизий – метод цепочек).

void hash\_t::remove(string r) {

unsigned int hash = MurmurHash2((r.c\_str()), r.length());

//Polynomial\* tmp = find\_row(r);

list<Row>::const\_iterator it;

for (it = array[hash]->begin(); it != array[hash]->end(); it++) {

if (it->name == r) { array[hash]->remove(\*it);

break;

filled\_rows--;

}

}

}

Функция получает на вход имя полинома. В теле функции вычисляется хеш имени, создаётся итератор. После этого в цикле итератор проходит всю цепочку элементов с вычисленным кешем и, при нахождении в ней строки с нужным именем, удаляет эту строку и уменьшает счётчик числа строк на единицу.

Алгоритм бинарного поиска строки в отсортированной таблице:

int sorted\_t::binsearch(string name) {

if (curr\_pos == 0)

{

throw std::logic\_error(" Empty table ");

cout << " search error ";

return NULL;

}

else {

int left = -1;

int right = curr\_pos;

int mid;

while (left < right - 1)

{

mid = (left + right) / 2;

if (array[mid].name < name)

left = mid;

else

right = mid;

}

return right;

//return \*(array[right].ptr\_p);

}

}

Функция получает на вход имя полинома.

Если таблица пустая, выводится сообщение об ошибке, и результатом работы программы является NULL.

Иначе происходит поиск элемента в отсортированном массиве:

1. Определяется значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе - во второй.
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

# Заключение

При выполнении лабораторной работы по написанию программного комплекса для хранения данных в таблицах были разработаны необходимые алгоритмы, позволяющие решить поставленную задачу.

Из ошибок, совершённых во время разработки, был извлечён ценный опыт. Необходимо:

1. Проектировать программный комплекс до начала его разработки
2. Соблюдать иерархию файлов и проектов
3. Вовремя писать тесты
4. При необходимости безжалостно переписывать код
5. Уметь выбирать правильный вариант между написанием своего кода и рефакторингом готового
6. Грамотно использовать указатели
7. Использовать утилиты, автоматически собирающие проекты в решение

# Литература

1. Гергель В. П. – Рабочие материалы студента по общему курсу «Методы программированя» (часть 2)
2. <https://habrahabr.ru/post/150732/> -- AVL-дерево
3. <http://www.mkurnosov.net/teaching/uploads/DSA/dsa-fall-lecture3.pdf> -- AVL-дерево
4. https://ru.wikipedia.org/wiki/Двоичное\_дерево\_поиска
5. https://ru.wikipedia.org/wiki/Хеш-таблица