МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: Программная инженерия

Отчет по лабораторной работе №2

«Алгебра полиномов »

Выполнили: студенты группы 3821Б1ПР2
Винокуров Иван Дмитриевич

Карагодин Андрей Романович

Проверила:

ассистент кафедры МОСТ, Усова М.А

Нижний Новгород
2022

Оглавление

Введение

Многочле́н (или полино́м, от греч. πολυ- «много» + лат. nomen «имя») — фундаментальное понятие в алгебре и математическом анализе. В простейшем случае многочленом называется функция вещественной или комплексной переменной следующего вида:

$P(x)=c_0+c_1x^1+...+c_nx^n$, где c_i - фиксированные коэффициенты

Максимальная степень n среди слагаемых называется степенью многочлена. Слагаемые в многочлене (члены многочлена) называются одночленами. Степенью одночлена называется сумма степеней входящих в него переменных. Максимальная степень среди слагаемых-одночленов называется степенью многочлена от нескольких переменных, а коэффициент при этом одночлене называется старшим коэффициентом.

Постановка задачи

Разработать программную систему для выполнения алгебраических операций над полиномами от трех переменных.

Условия/требования:

1. полиномы хранятся в виде списка;
2. полиномы хранятся во всех таблицах одновременно, ключом является имя;
3. таблиц должно быть 6 видов: линейная на массиве, линейная на списке, упорядоченная на массиве, дерево (авл или красно-черное), две хэш-таблицы;
4. операции над отдельным полиномом: вычисление в точке, умножение на константу, производная.
5. операции в выражениях из полиномов: сложение, вычитание, умножение на константу, умножение полиномов, деление полиномов. операции должны выполняться, используя постфиксную форму;
6. операции над таблицами: добавление полинома (во все сразу), удаление полинома (во всех сразу), поиск (только в активной таблице, выполняется в процессе вычисления выражений, составленных из имен полиномов);
7. активная (выбранная пользователем) таблица должна выводиться на экран в формате, как минимум, двух столбцов: 1) имя полинома, 2) строковое представление полинома.

Необходимые классы и минимально необходимые методы.

Классы CMonomial и CPolynomial для реализации мономов и полиномов. В CPolynomial операторы реализуются через соответствующие операторы из Cmonomial. Оба класса имеют операторы сложения, вычитания, деления, умножения, нахождения производной, сравнения, присвоения. Класс CPolynomial имеет метод findResult, который находит корень уравнения по трём переменным. Также он имеет метод Parse, который, соответственно, является парсером, превращающим строку в полином.

Для реализации таблиц используется интерфейс Itable, оъявляющий основные методы внутри таблиц. Классы CHashTableList, CHashTableMix, CLinearTableArray, CLinearTableList, COrderedTableArray, CTreeTable представляют из себя классы, в которых реализованы хэш таблица на списке, хэш таблица с перемешиванием, линейная таблица на массиве, линейная таблица на списке, упорядоченная таблица на массиве и авл-дерево соответственно. Для функционирования класса авл-дерева реализован класс CNodeTree, представляющий из себя вершину дерева.

Используемые структуры данных: список.

Руководство пользователя

Программный интерфейс программы можно увидеть на рисунке 1. В программе присутствует возможность создания либо мономов, либо полиномов. Верхние 4 поля отвечают за коэффициент и степени переменных в мономе. Кнопкой «Создать Моном» создаётся моном, с введёнными выше параметрами. Ниже находится поле для введения полинома. Кнопкой «создать» этот полином заносится во все таблицы сразу. Кнопка «найти» выделяет этот полином в списке справа. Кнопка «удалить» удаляет полином из всех таблиц. Активную таблицу можно выбрать в выпадающем списке снизу, содержимое таблицы будет отображаться в списке справа. Для того, чтобы вычислить значение полинома в конкретной точке, нужно выделить требуемый полином в правом списке левой кнопкой мыши, ввести координаты точки и нажать кнопку «вычислить в точке». Результат будет отображён в открывшемся диалоговом окне.

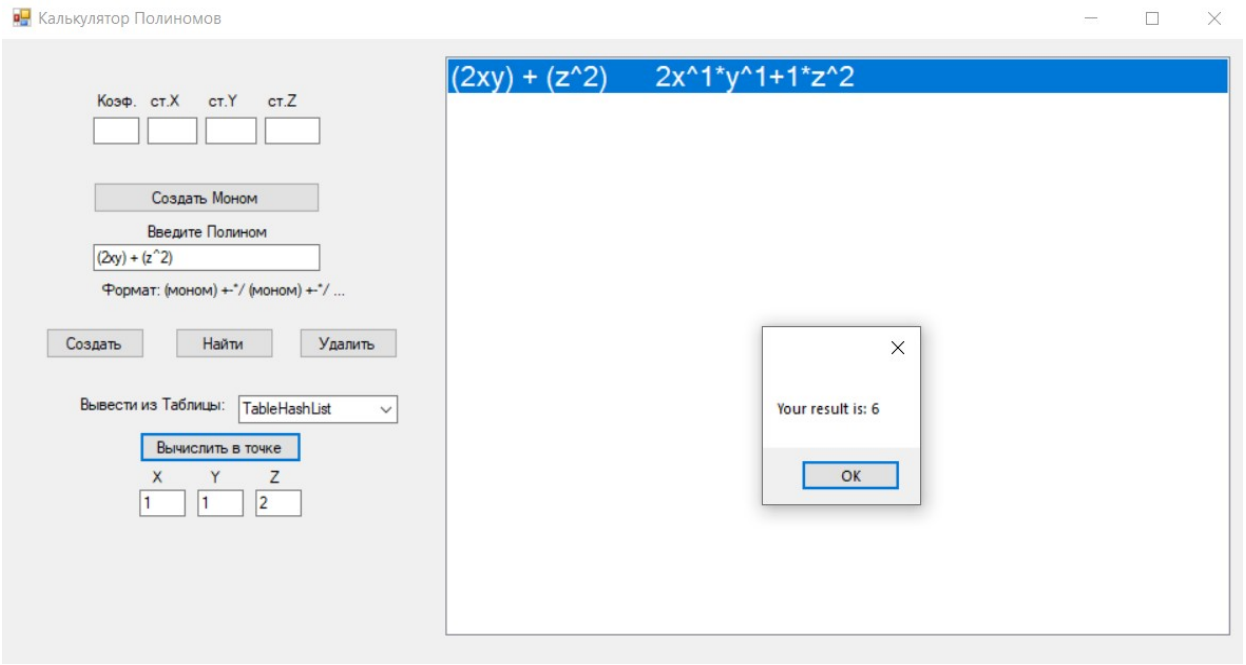


Рис. 1 Работа программы

Руководство программиста

Классы полиномов

CMonomial

Поля:
private:

double coefficient;

int degree[3];

bool checkequality(CMonomial m_monomial);

protected:

double getcoefficient();

Методы:

std::string toString();

Метод преобразующий моном в строку.

bool operator==(CMonomial _monomial);

Метод сравнения мономов. Если равны, возвращает true.

Параметры

CMonomial _monomial — моном, с которым необходимо сравнить.

bool operator!=(CMonomial _monomial);

Метод сравнения мономов. Если неравны, возвращает true.

Параметры

CMonomial _monomial — моном, с которым необходимо сравнить.

CMonomial& operator=(CMonomial& _monomial);

Метод присваивания мономов.

Параметры

CMonomial _monomial — моном, который необходимо присвоить.

CMonomial operator+(CMonomial _monomial);

Метод сложения мономов.

Параметры

CMonomial _monomial — моном, с которым необходимо сложить.

CMonomial operator-(CMonomial _monomial);

Метод вычитания мономов.

Параметры

CMonomial _monomial — моном, который необходимо вычесть.

CMonomial operator*(CMonomial _monomial);

Метод умножения мономов.

Параметры

CMonomial _monomial — моном, на который необходимо умножить.

CMonomial operator*(double _coefficient);

Метод умножения монома на скаляр.

Параметры

double _coefficient— скаляр, на который необходимо умножить.

CMonomial operator/(CMonomial _monomial);

Метод деления мономов.

Параметры

CMonomial _monomial — моном, на который необходимо поделить.

CMonomial getDerivative(CMonomial _monomial);

Метод нахождения производной.

Параметры

CMonomial _monomial — моном, производную которого необходимо вычислить.

CPolynomial

Поля:

CList<CMonomial> list;

Методы:

std::string toString();

Метод преобразующий моном в строку.

bool operator==(CPolynomial _polynomial);

Метод сравнения полиномов. Если равны, возвращает true.

Параметры

CPolynomial _polynomial— полином, с которым необходимо сравнить.

bool operator!=(CPolynomial _polynomial);

Метод сравнения полиномов. Если неравны, возвращает true.

Параметры

CPolynomial _polynomial— полином, с которым необходимо сравнить.

CPolynomial& operator=(CMonomial& _monomial);

Метод присваивания полиномов.

Параметры

CPolynomial _polynomial— полином, который необходимо присвоить.

CPolynomial operator+(CMonomial _monomial);

Метод сложения полинома с мономом.

Параметры

CMonomial _monomial — моном, с которым необходимо сложить.

CPolynomial operator-(CMonomial _monomial);

Метод вычитания мономов из полиномов.

Параметры

CMonomial _monomial — моном, который необходимо вычесть.

CPolynomial operator*(CMonomial _monomial);

Метод умножения полиномов на моном.

Параметры

CMonomial _monomial — моном, на который необходимо умножить.

CPolynomial operator*(double _coefficient);

Метод умножения полинома на скаляр.

Параметры

double _coefficient— скаляр, на который необходимо умножить.

CPolynomial operator/(CMonomial _monomial);

Метод деления полиномов на моном.

Параметры

CMonomial _monomial — моном, на который необходимо поделить.

CPolynomial l getDerivative(CPolynomial _polynomial);

Метод нахождения производной.

Параметры

CPolynomial _polynomial — полином, производную которого необходимо вычислить.

CPolynomial operator+(CPolynomial _polynomial);

Метод сложения полиномов.

Параметры

CPolynomial _polynomial — полином, с которым необходимо сложить.

CPolynomial operator-(CPolynomial _polynomial);

Метод вычитания полиномов.

Параметры

CPolynomial _polynomial — полином, который необходимо вычесть.

CPolynomial operator*(CPolynomial _polynomial);

Метод умножения полиномов

Параметры

CPolynomial _polynomial — полином, на который необходимо умножить.

CPolynomial operator/(CPolynomial _polynomial);

Метод деления полиномов.

Параметры

CPolynomial _polynomial — полином, на который необходимо поделить.

Классы таблиц

CHashTableList

Поля:

private:

CList<std::pair<CPolynomial, int>> arr[SIZE];

Методы:

private:

int hashFunction(std::pair<CPolynomial, int> _data)

Метод, генерирующая ключ. Хэш-функция.

Параметры

std::pair<CPolynomial, int> _data — данные, для которых нужно сгенерировать ключ.

Public:

void print()

Метод, выводящий на экран таблицу.

void insert(std::pair<CPolynomial, int> _data)

Метод вставки.

Параметры

std::pair<CPolynomial, int> _data — данные, которые нужно вставить

bool contains(std::pair<CPolynomial, int> _data)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

std::pair<CPolynomial, int> _data — данные, которые нужно обнаружить

void remove(std::pair<CPolynomial, int> _data)

Метод удаления из таблицы.

Параметры

std::pair<CPolynomial, int> _data — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

Параметры

std::string key — ключ, по которым нужно найти данные

CHashTableMix

Поля:

private:

size_t size;

std::vector<std::pair<std::string, Type>>* data;

Методы:

private:

size_t hashFunction(const std::string& obj)

Метод, генерирующая ключ. Хэш-функция.

Параметры

std::pair<CPolynomial, int> _data — данные, для которых нужно сгенерировать ключ.

Public:

void print()

Метод, выводящий на экран таблицу.

void insert(Type obj)

Метод вставки.

Параметры

Type obj— данные, которые нужно вставить

bool contains(Type obj)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

Type obj— данные, которые нужно обнаружить

void remove(Type obj)

Метод удаления из таблицы.

Параметры

Type obj — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

Параметры

std::string key — ключ, по которым нужно найти данные

CLinearTableArray

Поля:

std::pair<CPolynomial, int>* data;

int size;

int capacity;

Методы:

public:

void print()

Метод, выводящий на экран таблицу.

void insert(Type obj)

Метод вставки.

Параметры

Type obj— данные, которые нужно вставить

bool contains(Type obj)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

Type obj— данные, которые нужно обнаружить

void remove(Type obj)

Метод удаления из таблицы.

Параметры

Type obj — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

Параметры

std::string key — ключ, по которым нужно найти данные

CLinearTableList

Поля:

```
CList<std::pair<std::string, Type>> data;
```

Методы:

public:

void print()

Метод, выводящий на экран таблицу.

void insert(Type obj)

Метод вставки.

Параметры

Type obj— данные, которые нужно вставить

bool contains(Type obj)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

Type obj— данные, которые нужно обнаружить

void remove(Type obj)

Метод удаления из таблицы.

Параметры

Type obj — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

Параметры

std::string key — ключ, по которым нужно найти данные

COrderedTableArray

Поля:

std::pair<CPolynomial, int>* data;

int size;

int capacity;

Методы:

private:

int stringToInt(std::string _str)

Служебный метод, превращающий строку в int.

public:

void print()

Метод, выводящий на экран таблицу.

void insert(Type obj)

Метод вставки.

Параметры

Type obj— данные, которые нужно вставить

bool contains(Type obj)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

Type obj— данные, которые нужно обнаружить

void remove(Type obj)

Метод удаления из таблицы.

Параметры

Type obj — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

Параметры

std::string key — ключ, по которым нужно найти данные

CNodeTree

Поля:

Type data;

CNodeTree* left;

CNodeTree* right;

int height;

Методы:

CTreeTable

Поля:

private:

CNodeTree<std::pair<CPolynomial, int>>* root;

Методы:

private:

int stringToInt(std::string _str)

Служебный метод, превращающий строку в int.

int getHeight(CNodeTree<Type>* node)

Служебный метод, возвращающий высоту вершины.

int getBalanceFactor(CNodeTree<Type>* node)

Служебный метод, возвращающий значение баланса, т.е в которую сторону он смещён.

int updateHeight(CNodeTree<Type>* node)

Служебный метод, обновляющий высоту вершины.

CNodeTree<Type>* rotateLeft(CNodeTree<Type>* node)

Служебный метод, поворачивающий дерево влево.

CNodeTree<Type>* rotateRight(CNodeTree<Type>* node)

Служебный метод, поворачивающий дерево вправо.

CNodeTree<Type>* balance(CNodeTree<Type>* node)

Служебный метод, балансирующий дерево. Использует в себе методы rotateLeft и rotateRight

CNodeTree<Type>* remove(CNodeTree<Type>* node)

Служебный метод, реализующий удаление вершины.

CNodeTree<Type>* removeMin(CNodeTree<Type>* node)

Служебный метод, удаляющий крайний лист.

CNodeTree<Type>* findMin(CNodeTree<Type>* node)

Служебный метод, находящий крайний лист слева.

CNodeTree<Type>* findParent(CNodeTree<Type>* node)

Служебный метод, находящий «отца» вершины.

void print(CNodeTree<Type>* node)

Служебный метод, выводящий на экран вершину.

public:

void print()

Метод, выводящий на экран таблицу.

void insert(Type obj)

Метод вставки.

Параметры

Type obj— данные, которые нужно вставить

bool contains(Type obj)

Метод нахождения в таблице. Если найдено, то возвращает true

Параметры

Type obj— данные, которые нужно обнаружить

void remove(Type obj)

Метод удаления из таблицы.

Параметры

Type obj — данные, которые нужно удалить.

Type find(std::string key)

Метод нахождения в таблице по ключу. Если найдено, то возвращает данные из ячейки таблицы

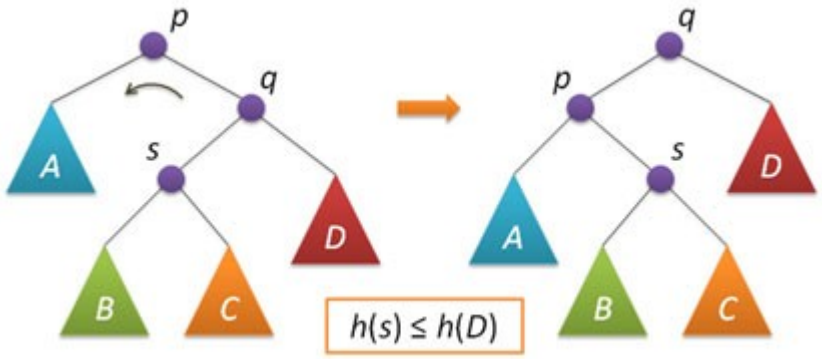
Параметры

std::string key — ключ, по которым нужно найти данные

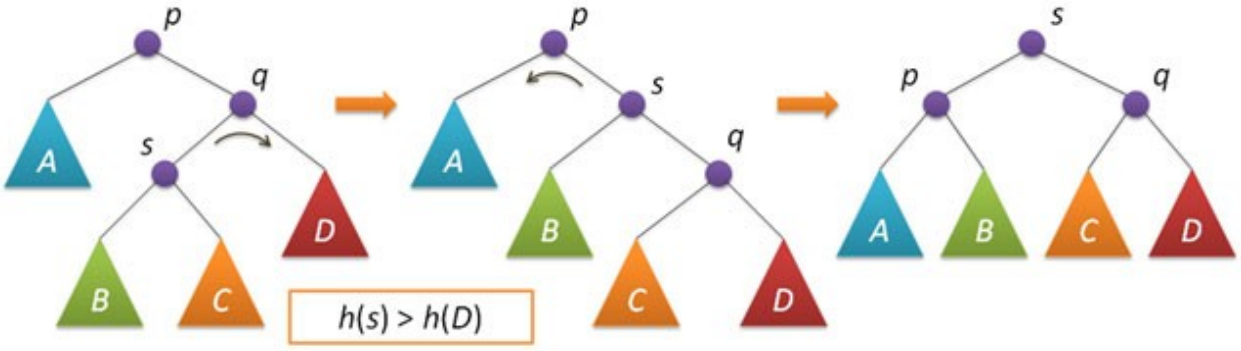
Описание алгоритмов

Балансировка АВЛ-дерева.

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда баланс некоторых узлов оказывается равными 2 или -2, т.е. возникает *расбалансировка* поддерева. 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)≤h(D).



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.



Хэш-таблица с открытым перемешиванием

Вставляем элемент

key = “pol”; k = ‘p’+’o’+’l’; // = 112+111+108=331

h(k) = 11 – адрес ячейки для вставки.

key = “ned”; k = ‘n’+’e’+’d’; // = 110+101+100=311

h(k) = 11 - пример коллизии при вставке, позиция уже занята.

Решение: повторное перемешивание с шагом перемешивания, удовлетворяющим $1 \le h < M$, НОД(h, M) = 1:

hh(k) = (h(k) + h) mod M

Значение шага перемешивания может быть любым, но обязательно взаимно простым с величиной, определяющей размер таблицы, чтобы обеспечить просмотр всех ее позиций.

hh(k) = (11 + 3) mod 20 =14

Если таблица еще не переполнена:

1. Вычислить значение функции хеширования для заданного ключа.

2. Проверить, статус найденной позиции для вставки:

- если позиция занята и ключ совпадает с ее значением, выбросить исключение (дублирование), выход;

- если позиция свободна, вставить элемент по указанной позиции, изменить статус на занята, выход;

- если позиция удалена, вставить элемент по найденной позиции, изменить статус на занята, выход;

3. Вызвать повторное перемешивание hh(k). Вернуться к шагу 2.

Заключение

В результате выполнения лабораторной работы мы реализовали рабочее приложение для работы с алгеброй полиномов и хранящее эти самые полиномы в 6 различных видах таблиц.

Литература

1. Статьи на сайте habr.com: [https://habr.com/ru/articles/150732/], [https://habr.com/ru/articles/509220/] ,

2. Сайт wikipedia.org [https://ru.wikipedia.org/wiki/Многочлен]

Приложения

Приложение 1. Функция парсинга

```
std::stringstream ss(_string);
std::stack<char> st;


CMonomial m;


char c;


char prevc;


while (ss >> c) {
```

```
if (c == '(') {
    st.push(c);
}
else if (c == ')') {
    st.pop();
    if (st.empty()) {
        list.push_back(m);
        m = CMonomial();
    }
}
else if (isdigit(c)) {
    ss.putback(c);
    double coefficient;
    ss >> coefficient;
    m.coefficient = coefficient;
}
else if (isalpha(c)) {
    int index = 0;
    while (isalpha(c)) {
        switch (c) {
        case 'x':
```

```
                index = 0;
                prevc = c;
                break;
            case 'y':
                index = 1;
                prevc = c;
                break;
            case 'z':
                index = 2;
                prevc = c;
                break;
            }
            m.degree[index] = m.degree[index] + 1;
            ss >> c;
        }
        ss.putback(c);
    }
    else if (c == '^') {
        int degree = 0;
        ss >> degree;
        int index = 0;
```

```
            switch (prevc) {
            case 'x':
                index = 0;
                break;
            case 'y':
                index = 1;
                break;
            case 'z':
                index = 2;
                break;
            }
            m.degree[index] = degree;
        }
    }
```

Приложение 2. Оператор деления полинома на полином

```
    CPolynomial result;
    if (_polynomial.list.isEmpty())
        throw std::invalid_argument("Division by Zero.");
    if (_polynomial.list.size == 1 &&
        (_polynomial.list.pop_front().degree[0] == 0 &&
            _polynomial.list.pop_front().degree[1] == 0 &&
```

```
            _polynomial.list.pop_front().degree[2] == 0))
    {
        CMonomial divMon = _polynomial.list.pop_back();
        CList<CMonomial> cpylist;
        cpylist.cpy(list);
        while (!cpylist.isEmpty())
        {
            CMonomial monom1;
            CMonomial resultmonom;
            monom1 = cpylist.pop_back();
            resultmonom = monom1 / divMon;
            result.list.push_back(resultmonom);
        }
    }
    else
    {
        CPolynomial divident = *this;
            while (divident.list.size > 0 && (divident.list.getHead().degree[0] >=
_polynomial.list.getHead().degree[0] &&
                                              divident.list.getHead().degree[1]    >=
_polynomial.list.getHead().degree[1] &&
```

```
                                                    divident.list.getHead().degree[2]    >=
_polynomial.list.getHead().degree[2]))

        {

                                    CMonomial   chastnoe   =   divident.list.pop_back()   /
_polynomial.list.pop_back();

            result.list.push_back(chastnoe);

            CPolynomial product = _polynomial * chastnoe;

            divident = divident - product;

        }

    }

    return result;
```

Приложение 3. Вставка в упорядоченную таблицу на массиве

```
            if (size < capacity) {
      int i = size - 1;

      while (i >= 0 && stringToInt(data[i].first) > stringToInt(obj.first)) {

            data[i + 1].first = data[i].first;

            data[i + 1].second = data[i].second;

            i--;

      }

      data[i + 1].first = obj.first;

      data[i + 1].second = obj.second;

      size++;

}
```

```
else {
        int new_capacity = capacity * 2;
        Type* new_data = new Type[new_capacity];
        for (int i = 0; i < size; i++) {
                new_data[i].first = data[i].first;
                new_data[i].second = data[i].second;
        }
        delete[] data;
        data = new_data;
        capacity = new_capacity;
        int i = size - 1;
        while (i >= 0 && stringToInt(data[i].first) > stringToInt(obj.first)) {
                data[i + 1].first = data[i].first;
                data[i + 1].second = data[i].second;
                i--;
        }
        data[i + 1].first = obj.first;
        data[i + 1].second = obj.second;
        size++;
}
```