МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №7

Тема: «НЕЛИНЕЙНЫЕ СТРУКТУРЫ»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Кузнецов Л. А.
группа ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЧАСТЬ 7.1

Условие

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Вариант

В данной работе представлен вариант 18, в котором требуется работать с АВЛ-деревом. По мере выполнения работы мне необходимо написать следующие функции: вставка элемента и балансировка дерева, симметричный обход, среднее арифметическое всех узлов.

Метод решения

Для начала стоит реализовать само АВЛ-дерево, главное правило которого гласит – все элементы справа от узла больше значения текущего листа, а слева – меньше. На рис.1 представлена реализация данного дерева ср всеми необходимыми методами и полями.

Теперь же стоит заняться описанием действия каждого метода, использованного в текущей работе. Начнём, пожалуй, с сеттеров (рис. 2). На данном рисунке всё очевидно: сеттеры возвращают значения, прописанные в их названии.

Далее затронем геттеры (рис.3). С ними всё аналогично – возвращают искомое, а зачем они нужны? Геттеры, как и сеттеры, нужны для безопасности элементов, чьё непосредственное изменение может быть губительно для структуры программы: ошибка в этих полях способна нарушить работу всей программы, поэтому данные меры предосторожности чрезмерно желательны к исполнению.

Стоит обратиться и к ядру нашего класса, а именно к конструктору (рис. 4). Он также не примечателен, ведь для создания объекта класса нашего дерева не нужно мудрить и делать лишних действий.

Рассмотрим метод поиска ячейки по значению (рис. 5). Подаём искомое значение и тривиально проходимся по элементам соответственно следующей методике: искомое значение больше текущего? Нет? Тогда идём налево. Меньше или равно? Нет? Тогда идём направо. При не нахождении элемента будем оповещать об этом пользователя, но это уже позже.

```
#include<iostream>
using namespace std;

class Leaf {
private:
    Leaf* right = nullptr;
    Leaf* left = nullptr;
    Leaf* head = nullptr;
    double value;

public:
    bool checked;
    int* rightLeftCount = new int[2] {0, 0};
    int rightLeft = 0;

    //сеттеры
    void setHead(Leaf* _head);
    void setRight(Leaf* _right);
    void setLeft(Leaf* _left);
    void setValue(double _value);

    //геттеры
    Leaf* getHead();
    Leaf* getRight();
    Leaf* getLeft();
    double getValue();

    //конструктор
    Leaf(double _value, Leaf* _head);

    // удалить элемент
    Leaf* deleteLeaf(double key);

    // добавить элемент
    void addValue(double _value);

    //найти сумму дерева
    double findTotalSum();

    //поменять корень
    Leaf* changeRoot();

    // поиск ячейки по значению
    Leaf* search(double _value);

    //повороты дерева
    Leaf* spinLeft();
    Leaf* spinRight();
};
```

Рисунок 1 – Вид АВЛ-дерева в коде

```
//сеттеры
void Leaf::setHead(Leaf* _head) { this->head = _head; }
void Leaf::setLeft(Leaf* _left) { this->left = _left; }
void Leaf::setRight(Leaf* _right) { this->right = _right; }
void Leaf::setValue(double _value) { this->value = _value; }
```

Рисунок 2 – Сеттеры

```
//геттеры
Leaf* Leaf::getHead() { return this->head; }
Leaf* Leaf::getLeft() { return this->left; }
Leaf* Leaf::getRight() { return this->right; }
double Leaf::getValue() { return this->value; }
```

Рисунок 3 – Геттеры

```
//конструктор
Leaf::Leaf(double _value,Leaf* _head) {
    setHead(_head);
    setValue(_value);
    this->checked = false;
}
```

Рисунок 4 – Конструктор

```
// поиск ячейки по зачению
Leaf* Leaf::search(double _value) {
    Leaf* tempLeaf = this;
    while (tempLeaf != nullptr) {
        if (_value == tempLeaf->getValue()) {
            return tempLeaf;
        }
        if (_value > tempLeaf->getValue()) {
            if (tempLeaf->getRight() == nullptr)
                return nullptr;
            tempLeaf = tempLeaf->getRight();
        }
        else
        {
            if (tempLeaf->getLeft() == nullptr)
                return nullptr;
            tempLeaf = tempLeaf->getLeft();
        }
    }
}
```

Рисунок 5 – Метод поиска ячейки по значению

На рис. 6 представлена работа метода по добавлению элемента в наше дерево. Нахождение свободной ячейки аналогично поиску ячейки по значению, однако же в данном случае на пустое место добавляется поданное значение.

Разберём метод удаления элемента (рис. 7-8). Я решил, что мы будем способны удалить даже корень, поэтому данный метод разделён на две части.

Возможны 4 различные ситуации, от которых будет зависеть поведение нашей программы, что и показано в нашем коде за счёт установления значения ячейки при помощи сеттера.

```cpp
//добавить элемент
void Leaf::addValue(double _value){
    if (_value > this->getValue()) {
        this->rightLeftCount[1]++;
        if (this->getRight() == nullptr)
        {
            this->setRight(new Leaf(_value, this));
            this->getRight()->rightLeft = 1;
        }
        else
            this->getRight()->addValue(_value);
    }
    else
    {
        this->rightLeftCount[0]++;
        if (this->getLeft() == nullptr){
            this->setLeft(new Leaf(_value, this));
            this->getLeft()->rightLeft = 0;
        }
        else
            this->getLeft()->addValue(_value);
    }
}
```

Рисунок 6 – Метод добавления элемента в дерево

```cpp
// удалить элемент
Leaf* Leaf::deleteLeaf(double _value) {
    Leaf* tempLeaf = this->search(_value);

    if (tempLeaf == nullptr)
        return nullptr;

    if (tempLeaf->getHead() == nullptr) { // мы изменим значение root так, что 0 - root переходит к левому элементу, 1 - root переходит к правому элементу, 2 - root не переходить ни к кому
        if (tempLeaf->getLeft() == nullptr && tempLeaf->getRight() == nullptr)
        {
            tempLeaf->setValue(2);
        }
        else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() == nullptr)
        {
            tempLeaf->setValue(0);
            tempLeaf->getLeft()->setHead(nullptr);
        }
        else if (tempLeaf->getLeft() == nullptr && tempLeaf->getRight() != nullptr)
        {
            tempLeaf->setValue(1);
            tempLeaf->getRight()->setHead(nullptr);
        }
        else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() != nullptr)
        {
            tempLeaf->setValue(1);
            Leaf* tempRightLeaf = tempLeaf->getRight();
            tempRightLeaf->rightLeftCount[0] += tempLeaf->rightLeftCount[0];
            while (tempRightLeaf->getLeft() != nullptr) {
                tempRightLeaf = tempRightLeaf->getLeft();
                tempRightLeaf->rightLeftCount[0] += tempLeaf->rightLeftCount[0];
            }

            tempRightLeaf->setLeft(tempLeaf->getLeft());
            tempLeaf->getLeft()->setHead(tempRightLeaf);
            tempLeaf->setLeft(nullptr);
            tempLeaf->getRight()->setHead(nullptr);
        }
        return tempLeaf;
    }
```

Рисунок 7 – 1-ая часть метода удаления элемента по значению

Во второй же части представлен общий случай, когда происходит удаление листа дерева, что провоцирует сортировку дочерних элементов данного четырьмя различными способами.

```
    }
    else {
        Leaf* tempHead = tempLeaf->getHead();
        int right = tempLeaf->rightLeft;
        while (tempHead != nullptr) {
            tempHead->rightLeftCount[right]--;
            right = tempHead->rightLeft;
            tempHead = tempHead->getHead();
        }
        if (tempLeaf->getLeft() == nullptr && tempLeaf->getRight() == nullptr)
        {
            if (tempLeaf->rightLeft == 0)
                tempLeaf->getHead()->setLeft(nullptr);
            else
                tempLeaf->getHead()->setRight(nullptr);
        }
        else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() == nullptr)
        {
            if (tempLeaf->rightLeft == 0)
                tempLeaf->getHead()->setLeft(tempLeaf->getLeft());
            else
                tempLeaf->getHead()->setRight(tempLeaf->getLeft());
            tempLeaf->getLeft()->setHead(tempLeaf->getHead());
        }
        else if (tempLeaf->getLeft() == nullptr && tempLeaf->getRight() != nullptr)
        {
            if (tempLeaf->rightLeft == 0)
                tempLeaf->getHead()->setLeft(tempLeaf->getRight());
            else
                tempLeaf->getHead()->setRight(tempLeaf->getRight());
            tempLeaf->getRight()->setHead(tempLeaf->getHead());
        }
        else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() != nullptr)
        {
            if (tempLeaf->rightLeft == 0)
                tempLeaf->getHead()->setLeft(tempLeaf->getRight());
            else
                tempLeaf->getHead()->setRight(tempLeaf->getRight());

            Leaf* tempRightLeaf = tempLeaf->getRight();
            while (tempRightLeaf->getLeft() != nullptr)
            {
                tempRightLeaf->rightLeftCount[0] += tempLeaf->rightLeftCount[0];
                tempRightLeaf = tempRightLeaf->getLeft();
            }
            tempLeaf->getRight()->setHead(tempLeaf->getHead());
            tempLeaf->getLeft()->setHead(tempRightLeaf);
            tempRightLeaf->setLeft(tempLeaf->getLeft());

        }
        tempLeaf->setRight(nullptr);
        tempLeaf->setLeft(nullptr);
        return tempLeaf;
    }
}
```

Рисунок 8 - 2-ая часть метода удаления элемента по значению

Далее идёт метод нахождения суммы дерева симметричным обходом (рис. 9-10). Да уж, это был самый трудно реализуемый метод, который будет представлен в данной работе, так как пришлось метаться между рекурсией и обходом поэлементно при определённых условиях. Я выбрал второй метод, так как при работе с суммой рекурсия довольно нестабильна.

```
// найти сумму дерева
double Leaf::findTotalSum() {
    double sum = 0;

    int state = 0; // список сотояний: 0 - идём налево, 1 - идём направо, 2 - идём вверх, -1 - прекращение работы программы

    Leaf* tempLeaf = this;
    while (tempLeaf!=nullptr)
    {
        if (state == 0)
        {
            if (tempLeaf->getLeft() == nullptr) {
                if (tempLeaf->checked == false){
                    sum += tempLeaf->getValue();
                    tempLeaf->checked = true;
                }
                state = 1;
            }
            else {
                if (tempLeaf->getLeft()->checked) {
                    if (tempLeaf->getRight() == nullptr) {
                        if (tempLeaf->checked == false){
                            sum += tempLeaf->getValue();
                            tempLeaf->checked = true;
                        }
                        state = 2;
                    }
                    else{
                        if (tempLeaf->getRight()->checked == false)
                        {
                            if (tempLeaf->checked == false) {
                                sum += tempLeaf->getValue();
                                tempLeaf->checked = true;
                            }
                            state = 1;
                        }
                        else {
                            state = 2;
                        }
                    }
                }
                else {
                    tempLeaf = tempLeaf->getLeft();
                }
            }
        }
        else if (state == 1) {
            if (tempLeaf->getRight() == nullptr) {
                state = 2;
            }
            else {
                if (tempLeaf->getRight()->checked == false){
                    tempLeaf = tempLeaf->getRight();
                    state = 0;
                }
                else {
                    state = 2;
                }
            }
        }
```

Рисунок 9 – 1-ая часть метода нахождения суммы дерева

Пришлось обработать множество различных ситуаций и предотвратить всевозможные ошибки программы, однако, несмотря на все трудности, желаемый результат был достигнут. И я несомненно доволен этим.

```
        }
        else {
            if (tempLeaf->getLeft() == nullptr && tempLeaf->getRight() == nullptr) {
                if (tempLeaf->getHead() == nullptr) {
                    tempLeaf->checked = false;
                    break;
                }
                tempLeaf = tempLeaf->getHead();
                if (tempLeaf->checked == false) {
                    sum += tempLeaf->getValue();
                    tempLeaf->checked = true;
                }
            }
            else if ( tempLeaf->getLeft() == nullptr && tempLeaf->getRight() != nullptr) {
                if (tempLeaf->getRight()->checked) {
                    tempLeaf->getRight()->checked = false;
                    tempLeaf = tempLeaf->getHead();
                    state = 0;
                }
                else {
                    state = 0;
                }
            }
            else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() == nullptr) {
                if (tempLeaf->getLeft()->checked == false) {
                    state = 0;
                }
                else {
                    tempLeaf->getLeft()->checked = false;
                    if (tempLeaf->getHead() == nullptr)
                        break;
                    tempLeaf = tempLeaf->getHead();
                    if (tempLeaf->checked == false) {
                        sum += tempLeaf->getValue();
                        tempLeaf->checked = true;
                    }
                    state = 0;
                }

            }
            else if (tempLeaf->getLeft() != nullptr && tempLeaf->getRight() != nullptr) {
                if (tempLeaf->getRight()->checked == false)
                    state = 1;
                else {
                    tempLeaf->getRight()->checked = false;
                    tempLeaf->getLeft()->checked = false;
                    tempLeaf = tempLeaf->getHead();
                    if (tempLeaf != nullptr && tempLeaf->checked == false) {
                        sum += tempLeaf->getValue();
                        tempLeaf->checked = true;
                    }
                }
            }
        }
    }

    this->checked = false;
    return sum;
}
```

Рисунок 10 - 2-ая часть метода нахождения суммы дерева

Одним из последних методов является метод балансировки дерева (рис. 11). Он невелик из-за того, что полагается на прочие методы, такие как поворот налево (рис.12) и поворот направо (рис.13).

```
// балансировка
Leaf* Leaf::changeRoot() {
    Leaf* tempLeaf = this;
    int* tempCount = tempLeaf->rightLeftCount;
    if (abs(tempLeaf->rightLeftCount[0] - tempLeaf->rightLeftCount[1]) < 2){
        cout << "Не нужна балансировка" << endl;
        return this;
    }

    if (tempLeaf->rightLeftCount[1] > tempLeaf->rightLeftCount[0])
    {
        return this->spinLeft();
    }
    else
    {
        return this->spinRight();
    }

    cout << "Не получилась балансировка" << endl;
}
```

Рисунок 11 – Метод балансировки дерева

```
// поворот налево
Leaf* Leaf::spinLeft() {
    Leaf* tempRoot = this->getRight();
    tempRoot->rightLeftCount[0] += this->rightLeftCount[0] + 1;
    Leaf* tempLeft = tempRoot;

    while (tempLeft->getLeft() != nullptr) {
        tempLeft = tempLeft->getLeft();
        tempLeft->rightLeftCount[0] += this->rightLeftCount[0] + 1;
    }

    tempLeft->setLeft(this);
    this->setHead(tempLeft);
    this->rightLeftCount[1] = 0;
    this->setRight(nullptr);
    tempRoot->setHead(nullptr);
    return tempRoot;
}
```

Рисунок 12 – Метод поворота дерева налево

```
// поворот направо
Leaf* Leaf::spinRight() {
    Leaf* tempRoot = this->getLeft();

    this->rightLeftCount[0] = tempRoot->rightLeftCount[1];
    this->setLeft(tempRoot->getRight());
    this->setHead(tempRoot);
    if (tempRoot->getRight() != nullptr) {
        tempRoot->getRight()->setHead(this);
    }

    tempRoot->setHead(nullptr);
    tempRoot->setRight(this);
    tempRoot->rightLeftCount[1] = this->rightLeftCount[0] + this->rightLeftCount[1] + 1;

    return tempRoot;
}
```

Рисунок 13 – Метод поворота дерева направо

Тестирование

Ниже будут представлены результаты тестирования различных методов на дереве, к которому будут представлены элементы 8.8, 10.2, 7.5, 7.3, 7.8, 9, 9.5, 10.5, 7.6, 7.55, 7.68, 7.57 соответственно.

На рис. 14 совершенно верно представлена сумма нашего дерева, пройдёмся же и по прочим методам.

```
----- "Программа для решения 7.1" - v1.0 -----
1 - Вывести на экран дерево | 2 - Удалить элемент по значению | 3 - Добавить элемент
4 - Вывести сумму дерева | 5 - Вывести среднее арифметическое дерева |  6 - Сделать балансировку корня
7 - Найти ячейку по значению | 8 - Очистить консоль | 9 - завершить работу программы
4
Сумма дерева равна 101
```

Рисунок 14 – Результат работы метода поиска суммы

Добавим-ка к нашему дереву новый элемент 10 и проверим результат работы метода методом поиска суммы.

```
1 - Вывести на экран дерево | 2 - Удалить элемент по значению | 3 - Добавить элемент
4 - Вывести сумму дерева | 5 - Вывести среднее арифметическое дерева |  6 - Сделать балансировку корня
7 - Найти ячейку по значению | 8 - Очистить консоль | 9 - завершить работу программы
3
Введите вещественное значение (пример: 0.0000...):
10
1 - Вывести на экран дерево | 2 - Удалить элемент по значению | 3 - Добавить элемент
4 - Вывести сумму дерева | 5 - Вывести среднее арифметическое дерева |  6 - Сделать балансировку корня
7 - Найти ячейку по значению | 8 - Очистить консоль | 9 - завершить работу программы
4
Сумма дерева равна 111
```

Рисунок 15 – Результат работы метода добавления элемента

Всё работает. Просто отлично!

Найти среднее арифметическое? На рис. 16 представлен результат работы метода.

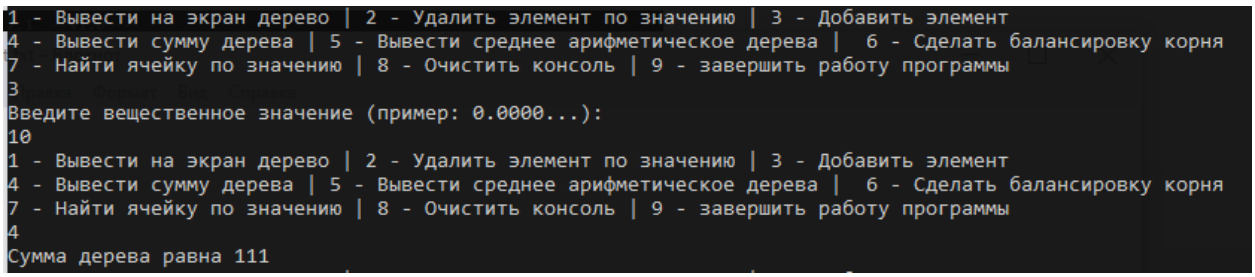

Рисунок 16 – Результат работы метода нахождения среднего арифметического

Замечательно. Теперь же сделаем балансировку корня

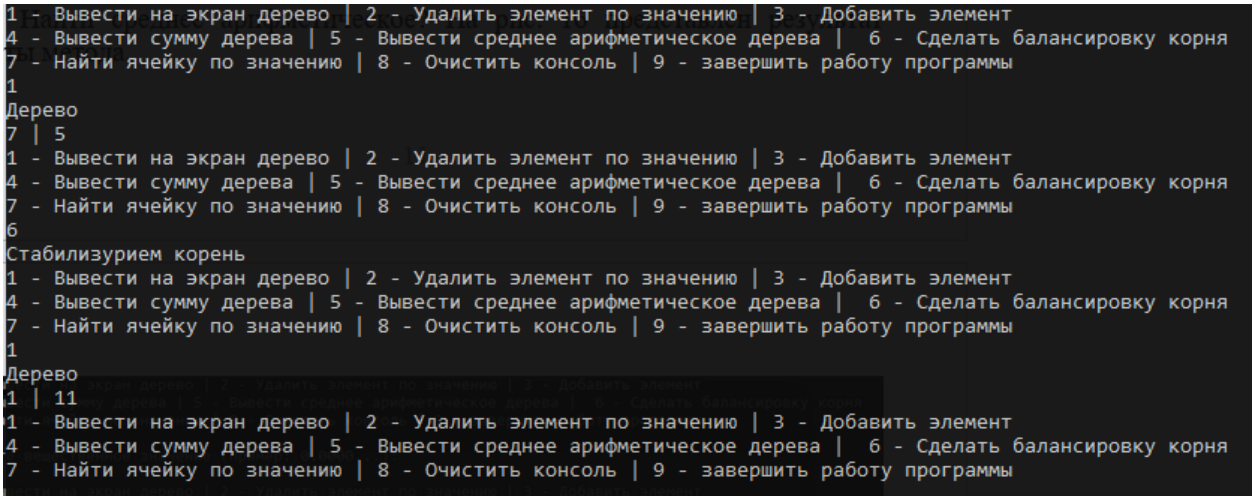

Рисунок 16 – Результат работы метода балансировки корня

Да уж, дерево сильно изменилось, но порой приходится несколько раз балансировать дерево, чтобы прийти к желаемому результату.

Найдём ячейку 8.8, а после удалим!

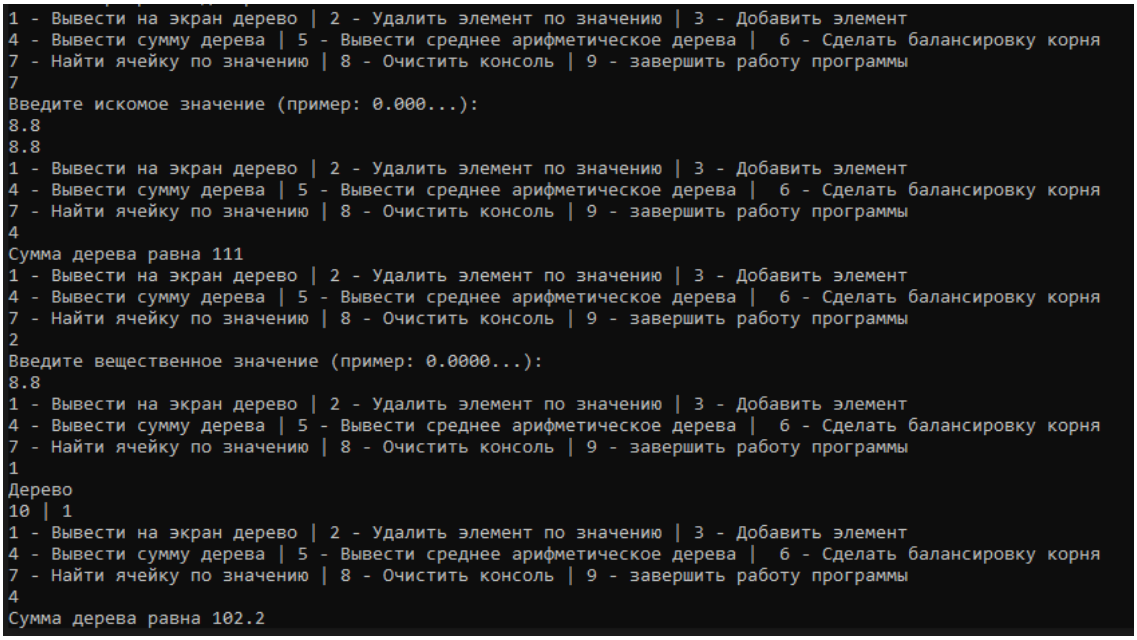

Рисунок 17 – Результаты работ методов поиска и удаления элементов

ВЫВОДЫ

Изучили различные виды деревьев и метода работы с ними, а также разработали программу взаимодействия с элементами этих деревьев.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).