Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Алтайский государственный технический университет
им. И.И. Ползунова»
Факультет (институт) Информационных технологий
Кафедра Прикладная математика

Отчет защищен с оценкой_____
_____А.И.Потупчик
(подпись преподавателя) (инициалы, фамилия)
“____”_____ 2024 г.

Отчет
по расчётному заданию
Связные списки. Библиотека стандартных шаблонов (STL)
(название расётного задания)

по дисциплине_____Типы и структуры данных_____
(наименование дисциплины)

_____ РЗ_09.03.04.13.000 ОТ_____
(обозначение документа)

Студент группы ПИ-21_____А.А.Лихтинфельд_____
(инициалы, фамилия)
Преподаватель доцент, доцент_____А.И.Потупчик_____
(должность, ученое звание) (инициалы, фамилия)

Барнаул 2024

Вариант №13

Удалить вершину с заданным значением ключа из бинарного дерева поиска. Предусмотреть вывод дерева сразу после его формирования и после удаления элемента.

Задание

1. Разработать и отладить программу на языке C++, реализующую работу с деревом в соответствии с вариантом. Выполнить оценку временной и емкостной сложности программы.
2. Исходные данные поместить в файл input.dat
3. Результаты вывести на экран. Исходные данные и результаты вывести также в выходной файл output.dat
4. Предусмотреть вывод исходного и выходного деревьев в табличной форме или в виде рисунка. Вывод в табличной форме означает вывод в прямом порядке, т.е. Корень, Левое поддерево, Правое поддерево

Текст программы:

```
#include<iostream>
#include <fstream>
#include <string>
#include <Windows.h>
using namespace std;
class BST {//бинарное дерево поиска
	struct node {
		int data; //поле данных
		node* left; //указатель на левый потомок
		node* right; //указатель на правый потомок
	};
	node* root;
	void makeEmpty(node* t) {
		if (t->left)
			makeEmpty(t->left);
		if (t->left) // тут right?
			makeEmpty(t->right);
		delete t;
	}

	//добавление узла в бинарное дерево
	node* insert(int x, node* t)
	{
```

```
		// Если текущий узел равен NULL, это означает, что мы достигли конца ветви
		if (t == NULL)
		{
			// Создаем новый узел и присваиваем ему значение x
			t = new node;
			t->data = x;
			t->left = t->right = NULL;// Устанавливаем указатели на NULL
		}
		else if (x < t->data)
			// Если значение x меньше значения текущего узла, рекурсивно вызываем insert для левого поддерева
			t->left = insert(x, t->left);
		else if (x > t->data)
			// Если значение x больше значения текущего узла, рекурсивно вызываем insert для правого поддерева
			t->right = insert(x, t->right);
		return t;// Возвращаем указатель на корень дерева после добавления узла
	}
	// поиск в бинарном дереве min
	node* findMin(node* t)
	{
		// если текущий узел t равен NULL, то функция возвращает NULL, что означает, что в дереве нет элементов.
		if (t == NULL)
			return NULL;
		//роверяется, есть ли у текущего узла левый потомок.
		//  Если нет, то текущий узел содержит минимальное значение в дереве, и он возвращается.
		else if (t->left == NULL)
			return t;
		//Если у текущего узла есть левый потомок, то рекурсивно вызывается функция findMin
		//  для левого потомка. Это продолжается до тех пор, пока не будет найден узел без левого потомка,
		//  что будет означать, что это минимальное значение в дереве.
		else
			return findMin(t->left);
	}
	//удаление вершины из бинарного дерева
```

```
//Эта функция remove принимает значение x для удаления и указатель на ткорень дерева текущий узел t
node* remove(int x, node* t) {
    node* temp;
    //Если текущий узел t равен NULL, значит мы достигли конца дерева, поэтому возвращается NULL. не найден элемент
    if (t == NULL)
        return NULL;
    //Если значение x меньше значения в текущем узле t, мы вызываем функцию remove для левого поддерева.
    else if (x < t->data)
        t->left = remove(x, t->left);
    //Если значение x больше значения в текущем узле t, мы вызываем функцию remove для правого поддерева.
    else if (x > t->data)
        t->right = remove(x, t->right);
    /*Если элемент найден и у текущего узла есть оба потомка, мы находим минимальное значение в правом поддереве,
    заменяем значение текущего узла этим минимальным значением
    и рекурсивно вызываем remove для правого поддерева с этим новым значением.*/
    else if (t->left && t->right)
    {
        temp = findMin(t->right);
        t->data = temp->data;
        t->right = remove(t->data, t->right);
    }
    /*В остальных случаях (если текущий узел имеет только одного потомка или не имеет их вовсе),
    мы сохраняем указатель на текущий узел в temp, затем перенаправляем указатель на узел на его потомка
    (если потомок один) или NULL (если потомков нет) и удаляем temp.*/
    else
    {
        temp = t;
        if (t->left == NULL)
            t = t->right;
        else if (t->right == NULL)
            t = t->left;
        delete temp;
```

```
        }
        return t;
    }
    // Вспомогательный метод для рекурсивной записи данных из дерева в файл
    void writeToFileHelper(node* t, ofstream& outFile) {
        if (t != nullptr) {
            outFile << t->data << " "; // Записываем значение текущего узла в
файл
            writeToFileHelper(t->left, outFile); // Рекурсивно обходим левое
поддерево
            writeToFileHelper(t->right, outFile); // Рекурсивно обходим правое
поддерево
        }
    }
    // поиск в бинарном дереве
    node* find(node* t, int x) {
        if (t == NULL)
            return NULL;
        else if (x < t->data)
            return find(t->left, x);
        else if (x > t->data)
            return find(t->right, x);
        else
            return t;
    }
    //печать бинарного дерева в виде дерева повернутого на -90 градусов
    void print_tree(node* t, int l) {
        if (t == NULL)
            return;
        print_tree(t->right, l + 1);
        for (int i = 0; i < l; ++i)
            cout << "\t";
        cout << t->data << endl;
        print_tree(t->left, l + 1);
    }
public:
    BST() {
        root = NULL;
```

```
}
~BST() {
	makeEmpty(root);
	root = nullptr;
}
//добавление узла в бинарное дерево.
void insert(int x) {
	root = insert(x, root);
}
//удаление вершины из бинарного дерева
void remove(int x) {
	root = remove(x, root);
}
// поиск в бинарном дереве
bool search(int x) {
	if (find(root, x))
		return true;
	else
		return false;
}
//печать бинарного дерева в виде дерева повернутого на -90 градусов
void display_tree() {
	cout << endl << endl;
	print_tree(root, 0);
	cout << endl << endl;
}
// Функция для чтения данных из файла и вставки их в дерево
void buildBSTFromFile(const string& filename, BST& tree) {
	ifstream file(filename);
	if (!file.is_open()) {
		cout << "Ошибка при открытии файла " << filename << endl;
		return;
	}
	int num;
	while (file >> num) {
		tree.insert(num);
	}
```

```
            file.close();
        }
        // Метод для записи данных из дерева в файл в прямом порядке
        void writeToFile(const string& inputFilename, const string& outputFilename) {
            // Открытие файлов для чтения и записи
            ifstream inFile(inputFilename);
            ofstream outFile(outputFilename);
            if (!inFile || !outFile) {
                cerr << "Ошибка открытия файлов!" << endl;
                return;
            }
            // Чтение и запись данных из файла inputFilename в outputFilename
            char ch;
            while (inFile.get(ch)) {
                outFile << ch;
            }
            outFile << endl; // Добавляем символ новой строки между старыми и новыми
данными
            // Рекурсивно обходим дерево в порядке "корень, левое поддерево, правое
поддерево"
            writeToFileHelper(root, outFile);
            cout << "Данные из дерева записаны в файл '" << outputFilename << "'." <<
endl;
            inFile.close();
            outFile.close();
        }
};
int main() {
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        //исходные данные помещаем в файл input.dat
        ofstream outFile("input.dat"); // Создаем файл для записи чисел
        if (!outFile) {
            cerr << "Не удалось открыть файл для записи!" << endl;
            return 1;
        }
        cout << "Введите числа (введите 'end' для завершения):\n";
```

```
	string input;
	while (true) {
		cin >> input; // Вводим число или 'end'
		if (input == "end") {
			break; // Завершаем цикл, если введено 'end'
		}
		try {
			int number = stoi(input); // Преобразуем строку в число
			outFile << number << " "; // Записываем число в файл
		}
		catch (const std::invalid_argument& e) {
			cerr << "Ошибка: Некорректный ввод!" << endl;
			// Очищаем ввод, чтобы избежать зацикливания
			cin.clear(); // Сбрасываем флаги состояния потока ввода
			// Очистка ввода путем считывания и игнорирования символов до символа новой строки '\n'
			while (cin.get() != '\n') {}
		}
	}
	cout << "Исходные данные записаны в файл 'input.dat'." << endl;
	outFile.close(); // Закрываем файл
	// Создание объекта BST
	BST bst;
	// Вставка элементов в дерево
	bst.buildBSTFromFile("input.dat", bst);
	// Печать бинарного дерева в виде дерева, повернутого на -90 градусов
	cout << "Binary Tree:" << endl;
	bst.display_tree();
	int value;
	cout << "Введите вершину для удаления: ";
	// Проверка корректности ввода
	while (!(cin >> value)) {
		cin.clear(); // Сброс флагов ошибок ввода
		while (cin.get() != '\n') {}
		cout << "Некорректный ввод. Пожалуйста, введите целое число: ";
	}
	bst.remove(value);
```

```
    cout << "New Binary Tree:" << endl;
    bst.display_tree();
    //вывод в прямом порядке, т.е.Корень, Левое поддерево, Правое поддерево
    // Запись данных из дерева в файл output.dat с данными из input.dat и новыми данными
    bst.writeToFile("input.dat", "output.dat");
    return 0;
}
```

Ёмкостная оценка: O(n), где n - кол-во элементов

Временная оценка: O(n), где n - кол-во элементов

Выполнение программы:

```
Введите числа (введите 'end' для завершения):
4
7
3
8
5
4
2
34
12
end
Исходные данные записаны в файл 'input.dat'.
Binary Tree:

                    34
                            12
                8
        7
                5
4
        3
                2
```

```
Введите вершину для удаления: 8
New Binary Tree:

                34
                        12
        7
                5
4
        3
                2

Данные из дерева записаны в файл 'output.dat'.
```

input.dat – Блокнот

Файл Правка Формат Вид Справка

4 7 3 8 5 4 2 34 12

output.dat – Блокнот

Файл Правка Формат Вид Справка

4 7 3 8 5 4 2 34 12

4 3 2 7 5 34 12