|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра инструментального и прикладного программного обеспечения | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7.1** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Нелинейные структуры»** | |
|  | |
| Выполнил студент группы ИКБО-10-23 | Харитонов А.Н. |
| Принял преподаватель | Макеева О.В. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_ \_ \_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024

# **Цель работы**

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

# **Ход работы**

Для реализации дерева был написан класс Node.

*struct* Node {

*int* data;

    Node\* left;

    Node\* right;

    Node(*int* *val*) : data(*val*), left(nullptr), right(nullptr) {}

};

Дерево – структура, состоящая из таких нод, для добавления новой ноды написана функция insert.

Node\* insert(Node\* *root*, *int* *key*) {

    if (*root* == nullptr) {

        return new Node(*key*);

    }

    if (*key* < *root*->data) {

*root*->left = insert(*root*->left, *key*);

    } else {

*root*->right = insert(*root*->right, *key*);

    }

    return *root*;

}

Для реализации симметричного обхода написана функция

*void* inorder(Node\* *root*) {

    if (*root* == nullptr) return;

    inorder(*root*->left);

    cout << *root*->data << " ";

    inorder(*root*->right);

}

А для поиска в ширину – bfs

*void* bfs(Node\* *root*) {

    if (*root* == nullptr) return;

    queue<Node\*> q;

    q.push(*root*);

    while (!q.empty()) {

        Node\* current = q.front();

        q.pop();

        cout << current->data << " ";

        if (current->left != nullptr) q.push(current->left);

        if (current->right != nullptr) q.push(current->right);

    }

}

Для поиска суммы листьев – sumLeaves.

*int* sumLeaves(Node\* *root*) {

    if (*root* == nullptr) return 0;

    if (*root*->left == nullptr && *root*->right == nullptr) return *root*->data;

    return sumLeaves(*root*->left) + sumLeaves(*root*->right);

}

Для высоты – height

*int* height(Node\* *root*) {

    if (*root* == nullptr) return 0;

*int* leftHeight = height(*root*->left);

*int* rightHeight = height(*root*->right);

    return max(leftHeight, rightHeight) + 1;

}

Протестируем нашу программу, для удобства напишем menu

*void* menu() {

    Node\* root = nullptr;

*int* choice, val;

    while (true) {

        cout << "\n--- Menu ---\n";

        cout << "1. Add Node\n";

        cout << "2. In-order\n";

        cout << "3. BFS\n";

        cout << "4. Leaves summ\n";

        cout << "5. Height\n";

        cout << "6. Exit\n";

        cout << "choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "value: ";

                cin >> val;

                root = insert(root, val);

                break;

            case 2:

                cout << "In-order: ";

                inorder(root);

                cout << endl;

                break;

            case 3:

                cout << "BFS: ";

                bfs(root);

                cout << endl;

                break;

            case 4:

                cout << "summ: " << sumLeaves(root) << endl;

                break;

            case 5:

                cout << "height: " << height(root) << endl;

                break;

            case 6:

                exit(0);

            default:

                cout << "Choose again.\n";

        }

    }

}

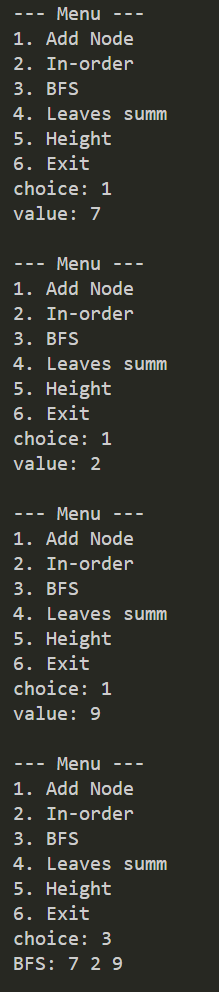


Рисунок 1. Тестирование программы

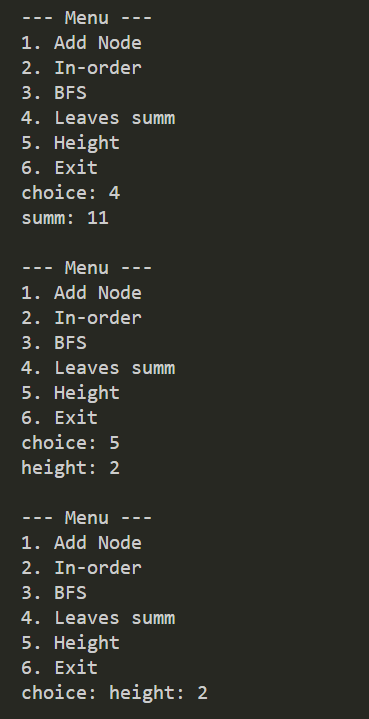


Рисунок 2. Тестирование программы

1. **Вывод**

В ходе работы я составил программу создания двоичного дерева поиска и реализовал процедуры для работы с деревом