МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации ИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2.4: Бинарное дерево поиска. AVL дерево.
ПО ДИСЦИПЛИНЕ
« СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

Выполнил студент группы ИКБО-01-21 Маров Г.А.

Принял старший преподаватель Туманова М.Б.

Практическая работа выполнена «10» октября 2022 г. _____ *(подпись студента)*

«Зачтено» « » сентября 2022 г. _____ *(подпись руководителя)*

Москва 2022

Цель: Получить знания и навыки в построении бинарных деревьев. Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Ход работы

Индивидуальный вариант (24):

24	Строка – город	Бинарное дерево поиска	+	+		+			+	+	

Математическая модель решения:

Для хранения дерева используется структура с значением элемента, указателями на левого, правого и верхнего элемента. Параметризированный конструктор со значением добавляемого элемента, который устанавливает его для данного члена и создаёт пустые указатели в остальных полях.

Прямой обход (preorderTraversal) идет в следующем порядке: корень, левый потомок, правый потомок. Симметричный (inorderTraversal) — левый потомок, корень, правый потомок.

Операция вставки работает аналогично поиску элемента, только при обнаружении у элемента отсутствия ребенка нужно подвесить на него вставляемый элемент.

Для поиска элемента в бинарном дереве поиска можно воспользоваться следующей функцией, которая принимает в качестве параметров корень дерева и искомый ключ. Для каждого узла функция сравнивает значение его ключа с искомым ключом. Если ключи одинаковы, то функция возвращает текущий узел, в противном случае функция вызывается рекурсивно для левого или правого поддерева. Узлы, которые посещает функция образуют нисходящий путь от корня, так что время ее работы O(h), где h — высота дерева.

Среднее значение находится с помощью прямого обхода и суммирования всех ключей.

Код программы с комментариями:

practice4.cpp:

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

```
// nodes for tree
struct Node {
    string key; // node value
    // pointers for neighbors
    Node *left;
    Node *right;
    Node *parent;
    // constructor with key as arg
    Node(string key) : key(key), left(NULL), right(NULL), parent(NULL)  {};
};

void insert(Node *root, Node *node) {
    while (root) { // go through tree
        if (node->key > root->key) {  // in case insertable node key > current node key
            if (root->right)
                root = root->right; // right (bigger) node exists = moving into it
            else {
                node->parent = root; // right (bigger) node doesnt exist = turning it
into insertable node
                root->right = node;
                break;
            }
        }
        else if (node->key < root->key) { // in case insertable node key < current node
key
            if (root->left)
                root = root->left; // left (lower) node exists = moving into it
            else {
                node->parent = root; // left (lower) node doesnt exist = turning it into
insertable node
                root->left = node;
                break;
            }
        }
    }
}
void preorderTraversal(Node *x) {
    if (x) { // while we dont reach end of a branch moving deeper
        cout << x->key << " "; // printing current node
        preorderTraversal(x->left); // printing left node-tree
        preorderTraversal(x->right); // printing right node-tree
    }
    else {
        return; // end of a branch have been reached
    }
}
void inorderTraversal(Node *x) {
    if (x) { // while we dont reach end of a branch moving deeper
        inorderTraversal(x->left); // printing left node-tree
        cout << x->key << " "; // printing current node
        inorderTraversal(x->right); // printing right node-tree
    }
    else return;
}
int averageValue(No de *x, int sum) {
    if (x) {
        averageValue(x->left, sum); // sum up left node-tree
        for (char c : x->key) // sum up current node
            sum += c - '0';
        averageValue(x->right, sum);  // sum up right node-tree
    }
    else return sum; // returning sum
}
int search(Node *x, string key, int cnt) {
```

```
    if (!x || key == x->key) // end of a branch or required node have been reached
        return cnt;
    if (key < x->key)
        return search(x->left, key, cnt + 1); // key bigger than required = recursively
moving to left branch
    else
        return search(x->right, key, cnt + 1); // else recursively moving to right branch
}

int main()
{
    cout << "enter amount of nodes in binary search tree: ";
    int n;
    Node* root = new Node("0"); // root node
    cin >> n;

    // building tree from keyboard
    for (int i = 0; i < n; i++) {
        cout << "enter " << i + 1 << " town: ";
        string town;
        cin >> town;
        insert(root, new Node(town));
    }
    cout << "tree has been built. list of all commands\n";
    cout << "/inorder - to inorder traversal\n";
    cout << "/preorder - to preorder traversal\n";
    cout << "/search - to find depth of a node\n";
    cout << "/insert - to insert a node\n";
    cout << "/average - to find average value of all nodes\n";
    cout << "/stop - to stop processing\n";
    cout << "enter command to execute: ";

    // starting command processing
    string command;
    getline(cin, command);
    while (getline(cin, command)) {
        if (command == "/inorder") {
            cout << "inorder traversal: ";
            inorderTraversal(root);
            cout << endl;
            cout << "enter next command: ";
        }
        else if (command == "/preorder") {
            cout << "preorder traversal: ";
            preorderTraversal(root);
            cout << endl;
            cout << "enter next command: ";
        }
        else if (command == "/search") {
            string key;
            cout << "enter key to search for: ";
            cin >> key;
            getline(cin, command);
            cout << "depth of a node: ";
            cout << search(root, key, 0) << endl;
            cout << "enter next command: ";
        }
        else if (command == "/insert") {
            string node;
            cout << "enter town to insert: ";
            cin >> node;
            getline(cin, command);
            insert(root, new Node(node));
            cout << "town has been inserted\n";
```

```
            cout << "enter next command: ";
        }
        else if (command == "/average") {
            cout << "average value of all nodes: ";
            cout <<  double(averageValue(root, 0)) / n << endl;
            cout << "enter next command: ";
        }
        else if (command == "/stop") {
            cout << "programm stopped";
            exit(0);
        }
        else cout << "wrong command" << endl << "enter next command: ";
    }


}
```

Тестирование

Тестирование программы приведено на рисунке 1.

```
enter amount of nodes in binary search tree: 4
enter 1 town: moscow
enter 2 town: kyoto
enter 3 town: bryansk
enter 4 town: la
tree has been built. list of all commands
/inorder - to inorder traversal
/preorder - to preorder traversal
/search - to find depth of a node
/insert - to insert a node
/average - to find average value of all nodes
/stop - to stop processing
enter command to execute: /inorder
inorder traversal:  bryansk kyoto la moscow
enter next command: /preorder
preorder traversal:  moscow kyoto bryansk la
enter next command: /search
enter key to search for: moscow
depth of a node: 1
enter next command: /insert
enter town to insert: kyiv
town has been inserted
enter next command: /average
average value of all nodes: 94
enter next command: /preorder
preorder traversal:  moscow kyoto bryansk kyiv la
enter next command: /stop
programm stopped
```

Рисунок 1 - тестирование задачи 3.1

Таким образом, по результатам тестирования видно, что программы работают корректно.

Вывод

В результате выполнения данной работы мной были освоены навыки построения бинарных деревьев и работы с ними. Была реализована программа для построения бинарного дерева и функции для работы с ним.