МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7.1

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема. Балансировка дерева поиска.

Выполнил студент группы ИКБО-41-23 Гольд Д.В.

Принял старший преподаватель Рысин М.Л.

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

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

Вариант

Вариант	Тип значения узла	Тип дерева	Реализовать алгоритмы								
			Вставка элемента	Прямой обход	Обратный обход	Симметричный обход	Обход в ширину	Найти сумму значений листьев	Найти среднее арифметическое всех узлов	Найти длину пути от корня до заданного значения	Найти высоту дерева
7	Строка – имя	Бинарное дерево поиска	+		+	+				+	+
8	Символ	Бинарное дерево поиска	+			+		+			+
9	Целое	АВЛ-дерево	+ (и балансировка)		+	+			+	+	

КОД ПРОГРАММЫ

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

using namespace std;

struct Node {//структура узла
    int key;//ключ узла
    unsigned char height;//высота узла
    Node* left;//указатель на левое поддерево
    Node* right;//указатель на правое поддерево

    Node(int k) {
        key = k;
        left = right = nullptr;
        height = 1;//начальная высота нового узла
    }
};

unsigned char height(Node* p) {//функция для получения высоты узла
    return p ? p->height : 0;
}

int balanceFactor(Node* p) {//функция для вычисления фактора баланса
    return height(p->right) - height(p->left);
}

void fixHeight(Node* p) {//функция для восстановления высоты узла
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;//обновляем высоту узла p -
    //выбираем максимальную высоту из левого и правого поддерева и добавляем 1
}

Node* rotateRight(Node* p) {//малое правое вращение
    Node* q = p->left;//сохраняем указатель на левое поддерево узла p в переменной q
    p->left = q->right;//левое поддерево p теперь указывает на правое поддерево q
```

```
38      p->left = q->right;//левое поддерево p теперь указывает на правое поддерево q
39      q->right = p;//правое поддерево q теперь указывает на p - выполняем правое вращение
40      fixHeight(p);
41      fixHeight(q);
42      return q;
43  }
44
45  Node* rotateLeft(Node* q) {//малое левое вращение
46      Node* p = q->right;
47      q->right = p->left;
48      p->left = q;
49      fixHeight(q);
50      fixHeight(p);
51      return p;
52  }
53
54  Node* rotateRightLeft(Node* p) {//большое правое вращение
55      p->right = rotateRight(p->right);
56      return rotateLeft(p);
57  }
58
59  Node* rotateLeftRight(Node* p) {//большое левое вращение
60      p->left = rotateLeft(p->left);
61      return rotateRight(p);
62  }
63
64  Node* balance(Node* p) {//балансировка дерева
65      fixHeight(p);
66      if (balanceFactor(p) == 2) {
67          if (balanceFactor(p->right) < 0)
68              return rotateRightLeft(p);
69          return rotateLeft(p);
70      }
71      if (balanceFactor(p) == -2) {
72          if (balanceFactor(p->left) > 0)
73              return rotateLeftRight(p);
74          return rotateRight(p);
```

```
            return rotateLeftRight(p);
        return rotateRight(p);
    }
    return p;
}

Node* insert(Node* p, int key) {//вставка элемента с балансировкой
    if (!p) return new Node(key);

    if (key < p->key)
        p->left = insert(p->left, key);//рекурсивно вставляем ключ в левое поддерево p
    else
        p->right = insert(p->right, key);

    return balance(p);
}

void inorder(Node* p, vector<int>& result) {//симметричный обход дерева (в отсортированном порядке)
    if (p) {
        inorder(p->left, result);//выполняем симметричный обход левого поддерева и добавляем элементы в результат
        result.push_back(p->key);//добавляем ключ текущего узла p в результат обхода
        inorder(p->right, result);//выполняем симметричный обход правого поддерева и добавляем элементы в результат
    }
}

void reverseOrder(Node* p) {//обратный обход дерева
    if (p) {
        reverseOrder(p->right);
        cout << p->key << " ";
        reverseOrder(p->left);
    }
}

double findAverage(Node* root) {//нахождение среднего арифметического всех узлов
    if (!root) return 0;

    vector<int> result;
    inorder(root, result);
```

```
110     inorder(root, result);
111
112     double sum = 0;
113     for (int key : result) {
114         sum += key;
115     }
116
117     return sum / result.size();
118 }
119
120 int findPathLength(Node* root, int value, int length = 0) {//нахождение длины пути от корня до заданного значения
121     if (!root) return -1;//значение не найдено
122
123     if (root->key == value) return length;
124
125     if (value < root->key)
126         return findPathLength(root->left, value, length + 1);
127     else
128         return findPathLength(root->right, value, length + 1);
129 }
130
131
132 void printTree(Node* root, string indent = "", bool last = true) {//функция для отображения дерева в текстовом виде
133     if (root) {
134         cout << indent;
135         if (last) {
136             cout << "R----";
137             indent += "   ";
138         }
139         else {
140             cout << "L----";
141             indent += "|  ";
142         }
143
144         cout << root->key << endl;
145         printTree(root->left, indent, false);
146         printTree(root->right, indent, true);
```

```
146         printTree(root->right, indent, true);
147     }
148 }
149
150 int main() {
151     Node* root = nullptr;
152     int initial_values[] = { 30, 20, 40, 10, 5, 3, 35, 50, 45, 60 };
153
154     for (int val : initial_values) {//вставляем все элементы в дерево
155         root = insert(root, val);
156     }
157
158
159     cout << "AVL tree:\n";
160     printTree(root);
161
162     int choice, value;
163     int length = 0;//инициализируем переменную length
164
165     do {
166         cout << "\nchoose task:\n";
167         cout << "1 - reverse order traversal\n";
168         cout << "2 - inorder traversal (sorted)\n";
169         cout << "3 - find the average of all nodes\n";
170         cout << "4 - find the path length from root to a value\n";
171         cout << "5 - exit\n";
172         cout << "enter ur choice: ";
173         cin >> choice;
174
175         switch (choice) {
176         case 1:
177             cout << "reverse order traversal: ";
178             reverseOrder(root);
179             cout << endl;
180             break;
181
182         case 2:
```

```
182         case 2:
183         {
184             vector<int> result;
185             inorder(root, result);
186             cout << "inorder traversal (sorted): ";
187             for (int key : result) {
188                 cout << key << " ";
189             }
190             cout << endl;
191         }
192         break;
193
194         case 3:
195         {
196             double average = findAverage(root);
197             cout << "average all nodes: " << average << endl;
198         }
199         break;
200
201         case 4:
202             cout << "enter the value to find the path length: ";
203             cin >> value;
204             length = findPathLength(root, value);
205             if (length == -1)
206                 cout << "value not found in the tree\n";
207             else
208                 cout << "path length from root to " << value << ": " << length << endl;
209             break;
210
211         case 5:
212             cout << "oh no...\n";
213             break;
214
215         default:
216             cout << "error!\n";
217         }
218     } while (choice != 5);
```

```
218     } while (choice != 5);
219
220     return 0;
221 }
222
```

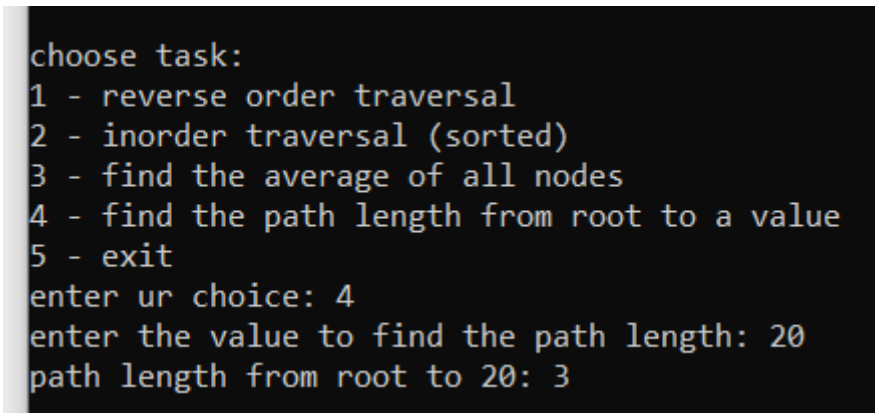
ТЕСТИРОВАНИЕ

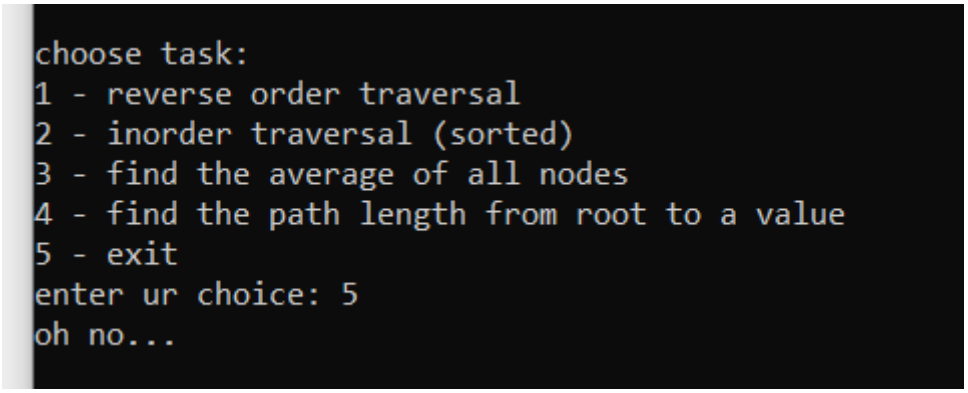
```
AVL tree:
R----10
   L----5
   |  L----3
   R----40
      L----30
      |  L----20
      |  R----35
      R----50
         L----45
         R----60

choose task:
1 - reverse order traversal
2 - inorder traversal (sorted)
3 - find the average of all nodes
4 - find the path length from root to a value
5 - exit
enter ur choice:
```

```
choose task:
1 - reverse order traversal
2 - inorder traversal (sorted)
3 - find the average of all nodes
4 - find the path length from root to a value
5 - exit
enter ur choice: 1
reverse order traversal: 60 50 45 40 35 30 20 10 5 3
```

```
choose task:
1 - reverse order traversal
2 - inorder traversal (sorted)
3 - find the average of all nodes
4 - find the path length from root to a value
5 - exit
enter ur choice: 2
inorder traversal (sorted): 3 5 10 20 30 35 40 45 50 60
```

```
choose task:
1 - reverse order traversal
2 - inorder traversal (sorted)
3 - find the average of all nodes
4 - find the path length from root to a value
5 - exit
enter ur choice: 3
average all nodes: 29.8
```





ВЫВОД

Составил программу для создания двоичного дерева поиска и реализовал процедуры для работы с деревом согласно варианту.