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. Нелинейные структуры
Балансировка дерева поиска

Выполнил студент группы ИКБО-43-23 Кощеев М. И.

Принял старший преподаватель Рысин М.Л.

Москва 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Освоить приёмы реализации алгоритмов балансиров

2 ХОД РАБОТЫ

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.
Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.
Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.
Вариант 7:

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	+		+	+				+	+

Алгоритм (menu)

1) Цикл: ввод с клавиатуры числа от 1 до 7
   - 1 – вставить элемент
     - Ввод с клавиатуры имени
   - 2 – Обратный обход
   - 3 – Симметричный обход
   - 4 – Нахождение длины пути от корня до заданного значения
     - Ввод имени
     - Если длинна не равна -1, вывод результат работы функции findPathLength
   - 5 – Вывод высоты дерева
   - 6 – Вывод дерева (данная функция не указана в задаче, на необходима только для тестирования)
   - 7 – выход из цикла
   - Другое значение – Вывод об ошибки
2) Удаление дерева

Код программы

```
91  void menu() {
92      TreeNode* root = nullptr;
93      int choice;
94      string name;
95
96      do {
97          cout << "1. Input element" << endl;
98          cout << "2. Reverse bypass" << endl;
99          cout << "3. Symmetric bypass" << endl;
100         cout << "4. Find the length of the path to the value" << endl;
101         cout << "5. Find tree height" << endl;
102         cout << "6. Print tree" << endl;
103         cout << "7. Exit" << endl;
104         cin >> choice;
105
106         switch (choice) {
107         case 1:
108             cout << "Input name: ";
109             cin >> name;
110             root = insertNode(root, name);
111             cout << "Element has been added" << endl;
112             break;
113         case 2:
114             cout << "Reverse bypass: ";
115             reverseOrder(root);
116             cout << endl;
117             break;
118         case 3:
119             cout << "Symmetric bypass: ";
120             inOrder(root);
121             cout << endl;
122             break;
123         case 4:
124             cout << "Input name for search: ";
125             cin >> name;
126             int length;
127             length = findPathLength(root, name);
128             if (length != -1) {
129                 cout << "The length of the path to " << name << ": " << length << endl;
130             } else {
131                 cout << "The value was not found in the tree" << endl;
132             }
133             break;
134         case 5:
135             cout << "Tree height: " << findHeight(root) << endl;
136             break;
```

```
137             case 6:
138                 cout << "Tree structure:" << endl;
139                 printTree(root);
140                 break;
141             case 7:
142                 break;
143             default:
144                 cout << "Error" << endl;
145             }
146         } while (choice != 7);
147 
148         deleteTree(root);
149     }
150 
151     int main() {
152         menu();
153         return 0;
154     }
155 
```

Алгоритм (insertNode):

1) Если дерево пустое, создается новый узел, который станет корнем
2) Если вставляемое значение меньше значения текущего узла, продолжаем искать место для вставки в левом поддереве
3) Если вставляемое значение больше, ищем в правом поддереве
4) Если значение уже существует, ничего не делаем, чтобы избежать дубликатов
5) Когда найдено подходящее место (свободный узел), создаем новый узел с этим значением

Код

```
16  TreeNode* insertNode(TreeNode* root, const string& name) {
17      if (root == nullptr) {
18          return new TreeNode(name);
19      }
20      if (name < root->name) {
21          root->left = insertNode(root->left, name);
22      } else if (name > root->name) {
23          root->right = insertNode(root->right, name);
24      }
25      return root;
26  }
27 
```

Алгоритм (reverseOrder)

1) Если текущий узел не пустой:
    a. Сначала рекурсивно обходим правое поддерево
    b. Вывод значение текущего узла
    c. рекурсивно обходим левое поддерево

Код

```
27
28  void reverseOrder(TreeNode* root) {
29      if (root != nullptr) {
30          reverseOrder(root->right);
31          cout << root->name << " ";
32          reverseOrder(root->left);
33      }
34  }
```

Алгоритм (inOrder)

1) Если текущий узел не пустой:
    a. Сначала рекурсивно обходим левое поддерево
    b. Вывод значение текущего узла
    c. рекурсивно обходим правое поддерево

Код

```
35
36  void inOrder(TreeNode* root) {
37      if (root != nullptr) {
38          inOrder(root->left);
39          cout << root->name << " ";
40          inOrder(root->right);
41      }
42  }
```

Алгоритм (findPathLength)

1) Начинаем с длины пути, равной нулю
2) Пока текущий узел не пуст
    a. Если искомое значение меньше значения текущего узла, переходим в левое поддерево и увеличиваем длину пути
    b. Если искомое значение больше значения текущего узла, переходим в правое поддерево и снова увеличиваем длину пути
    c. Если значения совпадают, возвращаем текущую длину пути
3) Если прошли по дереву до конца и не нашли значение, возвращаем -1

Код

```
int findPathLength(TreeNode* root, const string& target) {
    int length = 0;
    while (root != nullptr) {
        if (target < root->name) {
            root = root->left;
            length++;
        } else if (target > root->name) {
            root = root->right;
            length++;
        } else {
            return length;
        }
    }
    return -1;
}
```

Алгоритм (findHeight)

1) Если текущий узел пуст, возвращаем -1
2) Рекурсивно находим высоту левого поддерева
3) Рекурсивно находим высоту правого поддерева
4) Возвращаем максимальную высоту между левым и правым поддеревом, добавив 1, чтобы учесть текущий узел

Код

```
int findHeight(TreeNode* root) {
    if (root == nullptr) {
        return -1;
    }
    int leftHeight = findHeight(root->left);
    int rightHeight = findHeight(root->right);
    return max(a: leftHeight, b: rightHeight) + 1;
}
```

Тестирование (reverseOrder)

Заданные значения	Ожидаемый результат	Фактический результат
Cartman, Stan, Kyle, Kenny, Butters, Randy, Towelie, Mackey, Princess, Jesus	Towelie, Stan, Randy, Princess, Mackey, Kyle, Kenny, Jesus, Cartman, Butters	Towelie Stan Randy Princess Mackey Kyle Kenny Jesus Cartman Butters

```
Reverse bypass: Towelie Stan Randy Princess Mackey Kyle Kenny Jesus Cartman Butters
```

Тестирование (inOrder)

Заданные значения	Ожидаемый результат	Фактический результат
Cartman, Stan, Kyle, Kenny, Butters, Randy, Towelie, Mackey, Princess, Jesus	Butters, Cartman, Jesus, Kenny, Kyle, Mackey, Princess, Randy, Stan, Towelie	Butters Cartman Jesus Kenny Kyle Mackey Princess Randy Stan Towelie

```
Symmetric bypass: Butters Cartman Jesus Kenny Kyle Mackey Princess Randy Stan Towelie
```

Тестирование (findPathLength)

Заданные значения	Ожидаемый результат	Фактический результат
Kenny	3	3

```
Input name for search:Kenny
 The length of the path to Kenny: 3
```

Тестирование (findPathLength)

Заданные значения	Ожидаемый результат	Фактический результат
-	5	5

```
Tree height: 5
```

4 ВЫВОДЫ

В результате проделанной работы было разработано двоичное дерево поиска, которое реализует все необходимые операции для работы с дерево

5 ИСПОЛЬЗУЕМЫЙ ИСТОЧНИК

1. М.Л. РЫСИН, М.В. САРТАКОВ, М.Б. ТУМАНОВА Учебно-методическое пособие СиАОД часть 2