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

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Бинарное дерево поиска. AVL дерево»** | |
|  | |
| Выполнил студент группы ИКБО-02-19 | Макаров Д. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись руководителя)* |

Москва 2020

# **Цель работы**

Освоение применения бинарных деревьев и операций с ними.

# **Постановка задачи**

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом.
Каждую операцию оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.
Провести полное тестирование операций над программой. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.
Сделать выводы о проделанной работе, основанные на полученных результатах. Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

Вариант 21, сделать размер тестового дерева n=10, тип значения узла строка - имя, тип дерева АВЛ-дерево, вставка элемента (и балансировка), симметричный обход, найти сумму значений листьев,обход в ширину найти высоту дерева.

# **Решение**

АВЛ-дерево – структура данных, изобретенная в 1968 году двумя советскими математиками: Евгением Михайловичем Ландисом и Георгием Максимовичем Адельсон-Вельским. Сбалансированным называется такое двоичное дерево поиска, в котором высота каждого из поддеревьев, имеющих общий корень, отличается не более чем на некоторую константу k, и при этом выполняются условия характерные для двоичного дерева поиска.
АВЛ-дерево – сбалансированное двоичное дерево поиска с k=1. Для его узлов определен коэффициент сбалансированности (balance factor). Balance factor – это разность высот правого и левого поддеревьев, принимающая одно значение из множества {-1, 0, 1}.
Реализация дерева.

Реализация структуры по теме - Бинарное дерево поиска. AVL дерево

|  |
| --- |
| struct Node{ int data; Node \*left; Node \*right;};class Tree {public: Tree() : root\_(nullptr) {} int GetHeight(Node \*root); int Diff(Node \*root); Node \*RightRight\_Rotation(Node \*root); Node \*LeftLeft\_Rotation(Node \*root); Node \*LeftRight\_Rotation(Node \*root); Node \*RightLeft\_Rotation(Node \*root); Node \*Balance(Node \*root); Node \*Insert(Node \*root, int value); void Display(Node \*root, int level); void CountSum(Node \*root); void Sum(Node \*root); void width(Node \*root); void Simmetric(Node \*root); Node \*root\_;private: int sum = 0;}; |

Интерфейс программы:

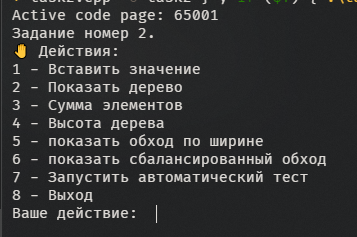


Рис. 1 Интерфейс программы

# **Тестирование:**

Запуск автоматического теста, где мы выводим дерево, обходим его по ширине и сбалансированно обходим, находим высоту дерева и сумму листьев.

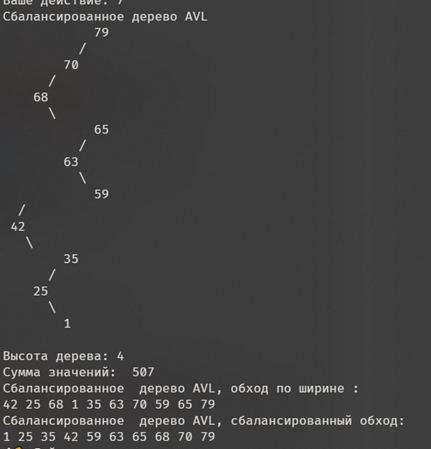


Рис. 2 Тест программы, Запуск автоматического теста, где мы выводим дерево, обходим его по ширине и сбалансированно обходим, находим высоту дерева и сумму листьев.

# **Вывод**

В результате выполнения работы я:

1. Освоил бинарное дерево поиска AVL и её реализацию на языке программирования C++
2. Научился делать симметричный обход и обход по ширине на дереве, вставке элемента в дерево,

# **Исходный код программы**

|  |
| --- |
| /\*\* \* @author Dmitriy Makarov \* @email DimaMakarov1941@yandex.ru \* @create date 2020-09-21 10:05:34 \* @modify date 2020-09-22 11:27:13 \* @desc [description] \*/#include <algorithm>#include <iomanip>#include <iostream>#include <queue>using namespace std;/\* █████╗ ██╗ ██╗██╗██╔══██╗██║ ██║██║███████║██║ ██║██║██╔══██║╚██╗ ██╔╝██║██║ ██║ ╚████╔╝ ███████╗╚═╝ ╚═╝ ╚═══╝ ╚══════╝████████╗██████╗ ███████╗███████╗╚══██╔══╝██╔══██╗██╔════╝██╔════╝ ██║ ██████╔╝█████╗ █████╗ ██║ ██╔══██╗██╔══╝ ██╔══╝ ██║ ██║ ██║███████╗███████╗ ╚═╝ ╚═╝ ╚═╝╚══════╝╚══════╝\* Вариант : 21\* Размер тестового дерева n : 10? Тип значения узла : Строка - имя\* Тип дерева : АВЛ-дерево-------------------------------------------------------------\* Вставка элемента : да (и балансировка)\* Симметричный обход : да\* Найти сумму значений листьев : да\* Обход в ширину : да\* Найти высоту дерева : да\*/struct Node{ int data; Node \*left; Node \*right;};class Tree{public: Tree() : root\_(nullptr) {} int GetHeight(Node \*root); int Diff(Node \*root); Node \*RightRight\_Rotation(Node \*root); Node \*LeftLeft\_Rotation(Node \*root); Node \*LeftRight\_Rotation(Node \*root); Node \*RightLeft\_Rotation(Node \*root); Node \*Balance(Node \*root); Node \*Insert(Node \*root, int value); void Display(Node \*root, int level); void CountSum(Node \*root); void Sum(Node \*root); void width(Node \*root); void Simmetric(Node \*root); Node \*root\_;private: int sum = 0;};int Tree::GetHeight(Node \*temp){ int h = 0; if (temp) { int l\_GetHeight = GetHeight(temp->left); int r\_GetHeight = GetHeight(temp->right); int max\_GetHeight = max(l\_GetHeight, r\_GetHeight); h = max\_GetHeight + 1; } return h;}int Tree::Diff(Node \*temp){ int l\_GetHeight = GetHeight(temp->left); int r\_GetHeight = GetHeight(temp->right); int b\_factor = (l\_GetHeight - r\_GetHeight); return b\_factor;}Node \*Tree::RightRight\_Rotation(Node \*parent){ Node \*temp; temp = parent->right; parent->right = temp->left; temp->left = parent; return temp;}Node \*Tree::LeftLeft\_Rotation(Node \*parent){ Node \*temp; temp = parent->left; parent->left = temp->right; temp->right = parent; return temp;}Node \*Tree::LeftRight\_Rotation(Node \*parent){ Node \*temp = parent->left; parent->left = RightRight\_Rotation(temp); return LeftLeft\_Rotation(parent);}Node \*Tree::RightLeft\_Rotation(Node \*parent){ Node \*temp = parent->right; parent->right = LeftLeft\_Rotation(temp); return RightRight\_Rotation(parent);}Node \*Tree::Balance(Node \*temp){ int balanceFactor = Diff(temp); if (balanceFactor > 1) if (Diff(temp->left) > 0) temp = LeftLeft\_Rotation(temp); else temp = LeftRight\_Rotation(temp); else if (balanceFactor < -1) if (Diff(temp->right) > 0) temp = RightLeft\_Rotation(temp); else temp = RightRight\_Rotation(temp); return temp;}Node \*Tree::Insert(Node \*root, int value){ if (!root) { root = new Node; root->data = value; root->left = NULL; root->right = NULL; return root; } else if (value < root->data) { root->left = Insert(root->left, value); root = Balance(root); } else if (value >= root->data) { root->right = Insert(root->right, value); root = Balance(root); } return root;}void Tree::Display(Node \*current, int level){ if (current != NULL) { if (current->right) Display(current->right, level + 4); if (level) cout << setw(level) << ' '; if (current->right) cout << " /\n" << setw(level) << ' '; cout << current->data << "\n "; if (current->left) { cout << setw(level) << ' ' << " \\\n"; Display(current->left, level + 4); } }}void Tree::CountSum(Node \*current){ if (!current) return; CountSum(current->right); CountSum(current->left); sum += current->data;}void Tree::Sum(Node \*current){ CountSum(current); printf("Сумма значений: %d\n", sum);}void Tree::width(Node \*root){ queue<Node \*> q; q.push(root); while (q.size() != 0) { Node \*tmp = (Node \*)q.front(); q.pop(); printf("%d ", tmp->data); if (tmp->left) q.push(tmp->left); if (tmp->right) q.push(tmp->right); }}void Tree::Simmetric(Node \*root){ if (!root) return; Simmetric(root->left); std::cout << root->data << " "; Simmetric(root->right);}/\*███╗ ███╗ █████╗ ██╗███╗ ██╗████╗ ████║██╔══██╗██║████╗ ██║██╔████╔██║███████║██║██╔██╗ ██║██║╚██╔╝██║██╔══██║██║██║╚██╗██║██║ ╚═╝ ██║██║ ██║██║██║ ╚████║╚═╝ ╚═╝╚═╝ ╚═╝╚═╝╚═╝ ╚═══╝\*/int main(){ system("chcp 65001"); printf("Задание номер 2.\n"); int input; Tree avl; while (input != 6) { printf("🤚 Действия:\n1 - Вставить значение\n2 - Показать дерево\n3 - Сумма элементов \n4 - Высота дерева\n5 - показать обход по ширине\n6 - показать сбалансированный обход\n7 - Запустить автоматический тест\n8 - Выход\nВаше действие: "); cin >> input; if (input == 1) { printf("Введите значение для вставки: \n"); int buff; scanf("%d", ); avl.root\_ = avl.Insert(avl.root\_, buff); printf("Значение вставлено успешно!\n"); } else if (input == 2) { if (!avl.root\_) printf("Дерево пусто!"); else printf("Сбалансированное дерево AVL\n"); avl.Display(avl.root\_, 0); printf("\n"); } else if (input == 3) avl.Sum(avl.root\_); else if (input == 4) printf("Высота дерева: %d\n", avl.GetHeight(avl.root\_)); else if (input == 5) avl.width(avl.root\_); else if (input == 6) avl.Simmetric(avl.root\_); else if (input == 7) { for (int i = 0; i 10; i++) avl.root\_ = avl.Insert(avl.root\_, rand() % 100 + 1); printf("Сбалансированное дерево AVL\n"); avl.Display(avl.root\_, 0); printf("\n"); printf("Высота дерева: %d\n", avl.GetHeight(avl.root\_)); avl.Sum(avl.root\_); printf("Сбалансированное дерево AVL, обход по ширине :\n"); avl.width(avl.root\_); printf("\n"); printf("Сбалансированное дерево AVL, сбалансированный обход:\n"); avl.Simmetric(avl.root\_); printf("\n"); } else if (input == 8) { printf("Выход...\n"); return 0; break; } else printf("Не правильная цифра! Введите значение заново:\n"); } return 0;} |
|  |