|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 2.4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Бинарное дерево поиска. AVL дерево»** | |
|  | |
| Выполнил студент группы ИКБО-30-22 | Сенькевич Г.Д. |
| Принял преподаватель | Красников С.А. |

Москва 2023

# **Цель работы**

Получить знания и навыки реализации деревьев поиска.

1. **Ход работы**

# **. Формулировка задачи**

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.

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 №20. Условие задания:

|  |  |
| --- | --- |
| Тип значения узла | Символ |
| Тип дерева | Бинарное дерево поиска |
| Вставка элемента | + |
| Прямой обход | + |
| Симметричный обход | + |
| Найти среднее арифметическое всех узлов | + |
| Найти длину пути от корня до заданного значения | + |

* 1. **. Описание модели (подход к решению)**

Двоичные деревья поиска (Binary Search Trees, BST) представляют собой структуру данных, используемую в информатике для хранения и организации данных. Они обладают важными свойствами, которые делают их полезными для операций поиска, вставки и удаления элементов.

Двоичное дерево поиска - это бинарное дерево, в котором каждый узел содержит значение (или ключ), и ключи в левом поддереве меньше ключа корня, а ключи в правом поддереве больше ключа корня. Это свойство называется свойством двоичного поиска.

Двоичные деревья поиска обеспечивают эффективный поиск элементов. Начиная с корня, мы сравниваем ключ, который мы ищем, с ключом корня. Если ключ, который мы ищем, меньше, чем ключ корня, мы переходим в левое поддерево; если больше - в правое. Процесс повторяется, пока мы не находим ключ или не достигаем конца дерева.

Вставка элемента в двоичное дерево поиска начинается с корня. Сначала сравнивается ключ, который нужно вставить, с ключом корня. Если ключ меньше, переходим в левое поддерево; если больше, в правое. Процесс повторяется до тех пор, пока не достигнем листа (пустого поддерева), в котором производим вставку нового узла с данным ключом. Важно поддерживать свойство двоичного поиска при вставке: ключи слева должны быть меньше ключа корня, а справа - больше. Если ключ уже существует в дереве, он может быть либо проигнорирован, либо обработан специальным образом в зависимости от требований задачи.

Существуют три варианта обхода деревьев:

* Прямой обход (КЛП): корень → левое поддерево → правое поддерево
* Симметричный обход (ЛКП): левое поддерево → корень → правое поддерево
* Обратный обход (ЛПК): левое поддерево → правое поддерево → корень
  1. **. Код программы**

Для решения поставленных задач был написан класс bin\_tree\_node (см. листинг 1), реализующий вершину бинарного дерева поиска, и класс bin\_tree (см. листинг 1), реализующий само бинарное дерево поиска.

Класс bin\_tree содержит следующие (открытые) методы:

* insert – вставка нового значения в дерево;
* print\_preorder – вывод прямого обхода дерева;
* print\_inorder – вывод симметричного обхода дерева;
* depth – длина пути от корня до вершины с заданным значением;
* height – высота дерева;
* average – среднее арифметическое значений всех вершин дерева;
* print – форматированный вывод дерева

Листинг 1 ­— Объявление классов bin\_tree\_node и bin\_tree.

|  |
| --- |
| // Вершина бинарного дерева поиска  template <typename T>  struct bin\_tree\_node  {  // Значение вершины  T val\_ = T();  // Родительская вершина  bin\_tree\_node\* parent\_ = nullptr;  // Левая дочерняя вершина  bin\_tree\_node\* left\_child\_ = nullptr;  // Правая дочерняя вершина  bin\_tree\_node\* right\_child\_ = nullptr;  bin\_tree\_node(T val,  bin\_tree\_node\* parent = nullptr,  bin\_tree\_node\* left\_child = nullptr,  bin\_tree\_node\* right\_child = nullptr)  : val\_(val), parent\_(parent), left\_child\_(left\_child),  right\_child\_(right\_child) {}  };  // Бинарное дерево поиска  template <typename T>  class bin\_tree  {  private:  // Корень дерево  bin\_tree\_node<T>\* root\_ = nullptr;  // Функция, удаляющая поддерево рекурсивно (вызывается из деструктора)  void del\_tree(bin\_tree\_node<T>\* node);  // Вставка нового значения в поддерево  bin\_tree\_node<T>\* insert(bin\_tree\_node<T>\* parent, T val);  // Вывод прямого обхода поддерева  void print\_preorder(bin\_tree\_node<T>\* node);  // Вывод симметричного обхода поддерева  void print\_inorder(bin\_tree\_node<T>\* node);  // Длина пути от заданной вершины до заданного значения  int depth(bin\_tree\_node<T>\* node, T val, int current\_depth = 0);  // Высота поддерева (от заданной вершины)  size\_t height(bin\_tree\_node<T>\* node);  // Сумма значений всех вершин поддерева  T sum(bin\_tree\_node<T>\* node);  // Подсчёт вершин в поддереве  size\_t count(bin\_tree\_node<T>\* node);  public:  // Деструктор  ~bin\_tree();  // Вставка нового значения в дерево  bin\_tree\_node<T>\* insert(T val);  // Вывод прямого обхода дерева  void print\_preorder();  // Вывод симметричного обхода дерева  void print\_inorder();  // Длина пути от корня до заданного значения  int depth(T val);  // Высота дерева  size\_t height();  // Среднее арифметическое по всем вершинам  T average();  // Вывод дерева (форматированный)  void print(size\_t value\_length = 0);  }; |

Реализации функций, указанных в условии, представлены в отдельных листингах 2-5.

Листинг 2 – Вставка элемента.

|  |
| --- |
| // Вставка нового значения в поддерево  template <typename T>  bin\_tree\_node<T>\* bin\_tree<T>::insert(bin\_tree\_node<T>\* parent, T val)  {  if (parent == nullptr)  {  if (root\_ == nullptr)  {  root\_ = new bin\_tree\_node<T>(val);  return root\_;  }  return nullptr;  }  if (val < parent->val\_)  {  if (parent->left\_child\_ == nullptr)  {  bin\_tree\_node<T>\* new\_node = new bin\_tree\_node<T>(val, parent);  parent->left\_child\_ = new\_node;  return new\_node;  }  else  {  return insert(parent->left\_child\_, val);  }  }  else  {  if (parent->right\_child\_ == nullptr)  {  bin\_tree\_node<T>\* new\_node = new bin\_tree\_node<T>(val, parent);  parent->right\_child\_ = new\_node;  return new\_node;  }  else  {  return insert(parent->right\_child\_, val);  }  }  }  // Вставка нового значения в дерево  template <typename T>  bin\_tree\_node<T>\* bin\_tree<T>::insert(T val)  {  return insert(root\_, val);  } |

Листинг 3 – Прямой обход.

|  |
| --- |
| // Вывод прямого обхода поддерева  template <typename T>  void bin\_tree<T>::print\_preorder(bin\_tree\_node<T>\* node)  {  if (node == nullptr) return;  std::cout << node->val\_ << ' ';  print\_preorder(node->left\_child\_);  print\_preorder(node->right\_child\_);  }  // Вывод прямого обхода дерева  template <typename T>  void bin\_tree<T>::print\_preorder()  {  print\_preorder(root\_);  } |

Листинг 4 – Симметричный обход.

|  |
| --- |
| // Вывод симметричного обхода поддерева  template <typename T>  void bin\_tree<T>::print\_inorder(bin\_tree\_node<T>\* node)  {  if (node == nullptr) return;  print\_inorder(node->left\_child\_);  std::cout << node->val\_ << ' ';  print\_inorder(node->right\_child\_);  }  // Вывод симметричного обхода дерева  template <typename T>  void bin\_tree<T>::print\_inorder()  {  print\_inorder(root\_);  } |

Листинг 5 – Найти среднее арифметическое всех узлов.

|  |
| --- |
| // Сумма значений всех вершин поддерева  template <typename T>  T bin\_tree<T>::sum(bin\_tree\_node<T>\* node)  {  if (node == nullptr)  {  return 0;  }  return node->val\_ + sum(node->left\_child\_) + sum(node->right\_child\_);  }  // Подсчёт вершин в поддереве  template <typename T>  size\_t bin\_tree<T>::count(bin\_tree\_node<T>\* node)  {  if (node == nullptr)  {  return 0;  }  return 1 + count(node->left\_child\_) + count(node->right\_child\_);  }  // Среднее арифметическое по всем вершинам  template <typename T>  T bin\_tree<T>::average()  {  return sum(root\_) / count(root\_);  } |

Листинг 6 – Найти длину пути от корня до заданного значения.

|  |
| --- |
| // Длина пути от заданной вершины до заданного значения  template <typename T>  int bin\_tree<T>::depth(bin\_tree\_node<T>\* node, T val, int current\_depth)  {  if (node == nullptr)  {  return -1;  }  if (val == node->val\_)  {  return current\_depth;  }  else if (val < node->val\_)  {  return depth(node->left\_child\_, val, current\_depth + 1);  }  else  {  return depth(node->right\_child\_, val, current\_depth + 1);  }  }  // Длина пути от корня до заданного значения  template <typename T>  int bin\_tree<T>::depth(T val)  {  return depth(root\_, val);  } |

* 1. **. Результаты тестирования**

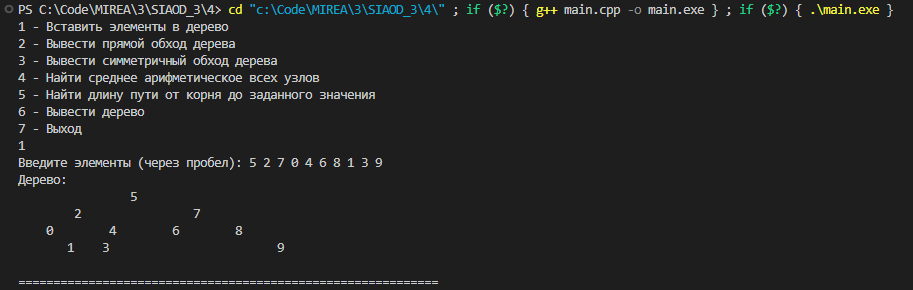


Рисунок 1 – Ввод дерева из 10 вершин с клавиатуры.

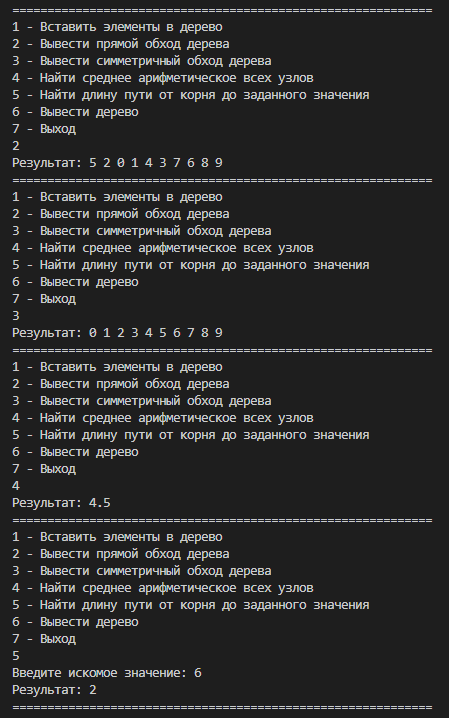


Рисунок 1 –Тестирование всех функций.

Тестирование показало, что программа работает правильно, корректно решая поставленные задачи.

1. **Выводы**

В результате выполнения работы я:

1. Изучил устройство двоичного дерева поиска и алгоритмы работы с ним;
2. Реализовал двоичное дерево поиска на C++.
3. **Список литературы**
4. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
5. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
6. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).