МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7.1

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема «Балансировка дерева поиска»

Выполнил студент группы ИКБО-13-23 Русаков М.Ю.

Принял преподаватель Сорокин А.В.

Москва 2024 г.

Оглавление

Цель работы: изучить абстрактную структуру данных АВЛ-дерево

1. Выполнение работы

1.1. Формулировка задачи

Выполнить построение АВЛ-дерева. Реализовать алгоритм вставки узла с балансировкой, симметричный вывод значений элементов, а так же подсчет суммы и среднего арифметического значений элементов АВЛ-дерева.

1.2. Математическая модель

АВЛ-дерево — это сбалансированное двоичное дерево поиска, в котором разница высот поддеревьев любого узла не превышает 1. АВЛ-дерево поддерживает балансировку за счёт операций поворотов (малых и больших) при добавлении или удалении узлов, чтобы поддерживать условие сбалансированности. Это позволяет сохранить временную сложность операций поиска, вставки и удаления на уровне $O(\log n)$.

1. Симметричный обход АВЛ-дерева

Симметричный обход АВЛ-дерева означает посещение всех узлов дерева в порядке возрастания их значений. Этот обход начинается с самого левого узла дерева и завершается правым. Для этого:

- Рекурсивно посещаем все узлы в левом поддереве текущего узла;
- Посещаем сам текущий узел и выводим его на экран;
- Рекурсивно посещаем все узлы в правом поддереве текущего узла.

2. Алгоритм нахождения суммы элементов АВЛ-дерева

Чтобы найти сумму всех элементов АВЛ-дерева, выполняем модифицированный симметричный обход, в котором к общей сумме добавляется значение каждого посещенного узла. Процесс можно описать так:

Начинаем с корневого узла и идем в самое левое поддерево. Для каждого узла рекурсивно суммируем его значение и значения всех узлов его поддеревьев. Повторяем эти действия для каждого узла дерева, пока не посетим все элементы. После обхода всех узлов получаем итоговую сумму.

3. Алгоритм нахождения среднего арифметического значений элементов дерева

Чтобы вычислить среднее арифметическое значений узлов, нужны две вещи: общая сумма всех элементов и общее количество узлов дерева. Это можно описать следующим образом:

Во время симметричного обхода находим сумму значений узлов (как в предыдущем алгоритме). Параллельно или отдельным обходом подсчитываем количество узлов дерева. В конце делим общую сумму на количество узлов, получая среднее арифметическое.

1.3. Код программы

АВЛ-дерево и вышеописанные алгоритмы были реализованы на языке программирования Go (см. ниже)

```
1  package avl
2
3  import "fmt"
4
5  type avlTree struct {
6    value       float64
7    left, right *avlTree
8    height      int
9  }
10
11 func NewFromSlice(src []float64) *avlTree {
12   var tree *avlTree
13   for _, value := range src {
14     tree = Insert(tree, value)
15   }
16   return tree
17 }
18
19 func (tree *avlTree) PrintInOrder() {
20   if tree != nil {
21     tree.left.PrintInOrder()
22     fmt.Printf("%f ", tree.value)
23     tree.right.PrintInOrder()
24   }
25 }
26
27 func (tree *avlTree) Sum() float64 {
28   if tree == nil {
29     return 0.0
30   }
31   return tree.value + tree.left.Sum() + tree.right.Sum()
32 }
33
34 func (tree *avlTree) ArithmeticMean() float64 {
35   if tree == nil {
36     return 0.0
37   }
```

```
  return tree.Sum() / float64(countTreeNodes(tree))
}

func Insert(tree *avlTree, value float64) *avlTree {
  if tree == nil {
    return &avlTree{value: value, height: 1}
  }

  if value < tree.value {
    tree.left = Insert(tree.left, value)
  } else if value > tree.value {
    tree.right = Insert(tree.right, value)
  } else {
    fmt.Printf("Значение %f уже существует в дереве\n", value)
    return tree
  }

  tree.height = max(getHeight(tree.left), getHeight(tree.right)) + 1

  balance := getBalance(tree)

  if balance > 1 && value < tree.left.value {
    return rightRotation(tree)
  }

  if balance < -1 && value > tree.right.value {
    return leftRotation(tree)
  }

  if balance > 1 && value > tree.left.value {
    tree.left = leftRotation(tree.left)
    return rightRotation(tree)
  }

  if balance < -1 && value < tree.right.value {
    tree.right = rightRotation(tree.right)
    return leftRotation(tree)
  }

  return tree
}

func getHeight(tree *avlTree) int {
  if tree == nil {
    return 0
  }
  return tree.height
}

func getBalance(tree *avlTree) int {
  if tree == nil {
    return 0
  }
  return getHeight(tree.left) - getHeight(tree.right)
```

```
}

func max(a, b int) int {
  if a > b {
    return a
  }
  return b
}

func leftRotation(tree *avlTree) *avlTree {
  y := tree.right
  T2 := y.left

  y.left = tree
  tree.right = T2

  tree.height = max(getHeight(tree.left), getHeight(tree.right)) + 1
  y.height = max(getHeight(y.left), getHeight(y.right)) + 1

  return y
}

func rightRotation(tree *avlTree) *avlTree {
  x := tree.left
  T2 := x.right

  x.right = tree
  tree.left = T2

  tree.height = max(getHeight(tree.left), getHeight(tree.right)) + 1
  x.height = max(getHeight(x.left), getHeight(x.right)) + 1

  return x
}

func countTreeNodes(tree *avlTree) int {
  if tree == nil {
    return 0
  }
  return 1 + countTreeNodes(tree.left) + countTreeNodes(tree.right)
}
```

1.4. Тестирование

Произведем тестирование на 10 элементах

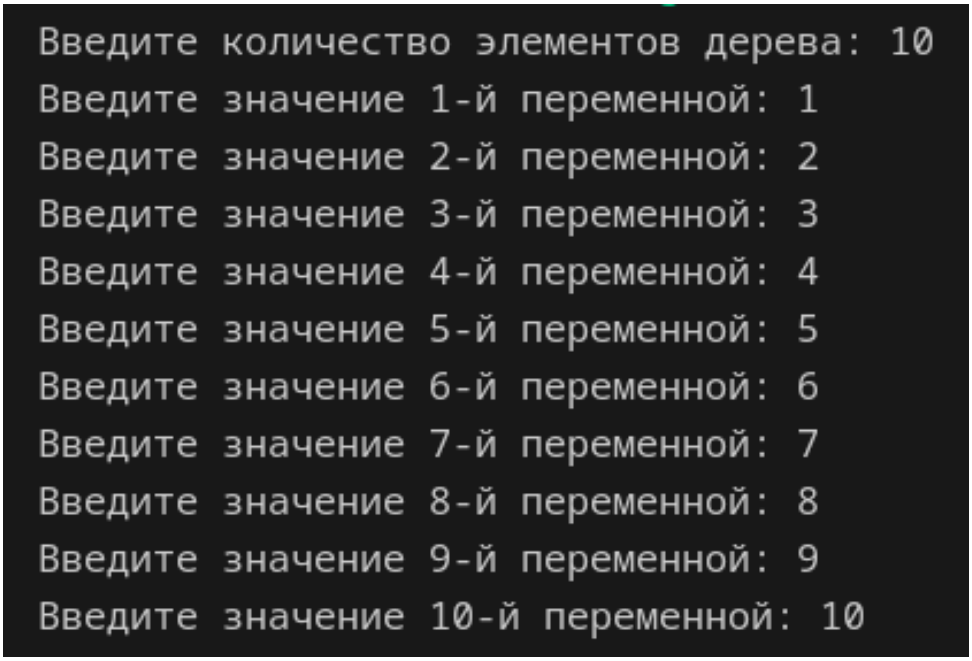

Рисунок 1 – Создание АВЛ-дерева из 10 элементов

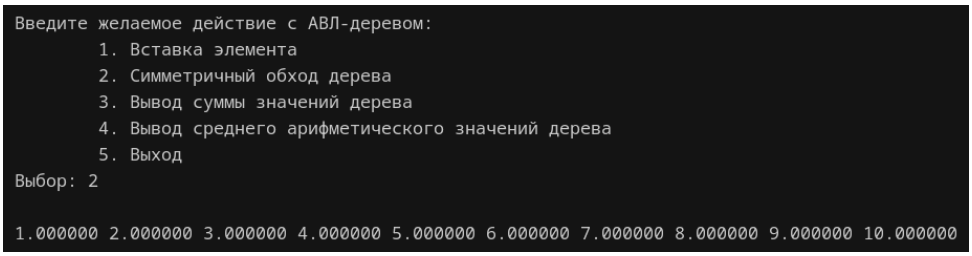

Рисунок 2 – Симметричный обход АВЛ-дерева

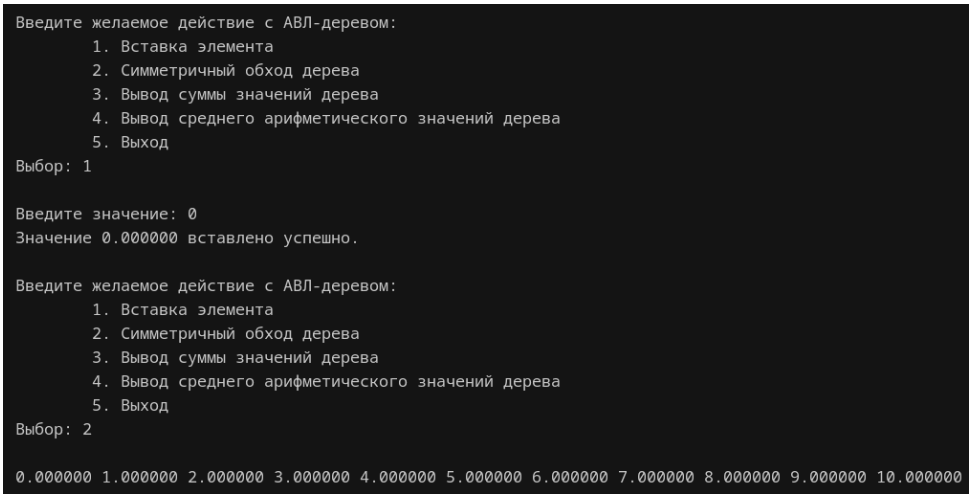

Рисунок 3 – Вставка элемента и симметричный обход АВЛ-дерева

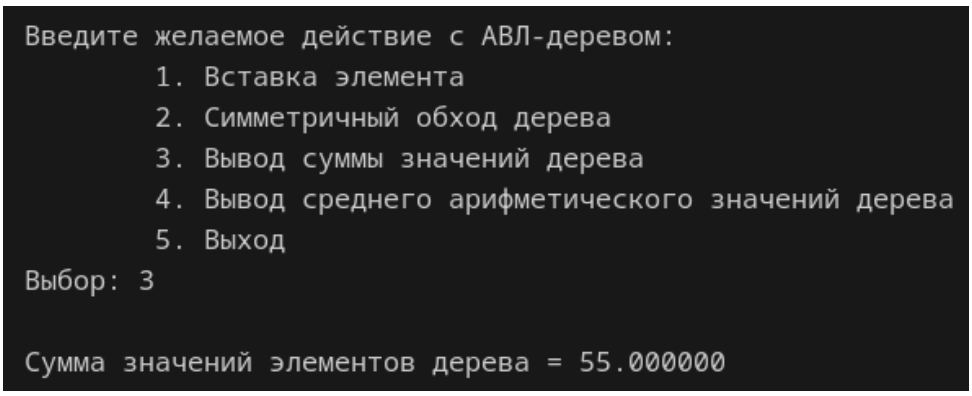

Рисунок 4 – Сумма значений элементов АВЛ-дерева

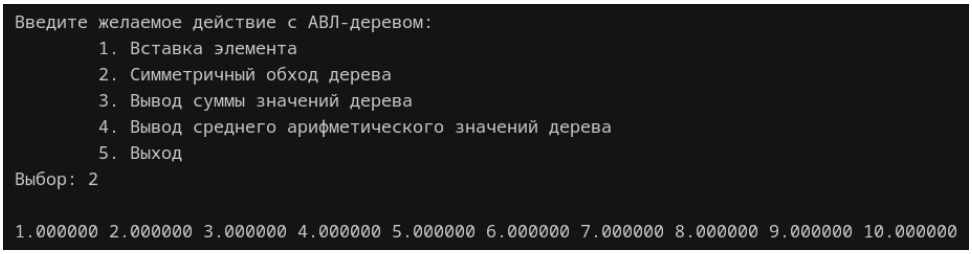

Рисунок 5 – Среднее арифметическое значений элементов АВЛ-дерева

2. Вывод

В ходе работы было изучено устройство AVL-дерева, его особенности и принципы сбалансированности, которые позволяют эффективно выполнять операции над элементами. Был рассмотрен алгоритм симметричного обхода дерева, позволяющий обрабатывать его узлы в порядке возрастания значений.

Также были реализованы алгоритмы для нахождения суммы и среднего арифметического значений всех элементов дерева, что позволяет проводить базовый анализ данных в дереве.