Министерство образования Республики Беларусь
Учреждение образования «Полоцкий государственный университет»

Кафедра технологий программирования

Алгоритмы и структуры данных

Отчет по лабораторной работе №5

Вариант 11

Выполнил — Ланцев Евгений Николаевич. 21-ИТ-1, ФИТ

Проверил — преподаватель Виноградова А.Д.

Полоцк
2022 г.

Лабораторная работа № 5

“Деревья. Двоичное дерево. Применение деревьев при разработке приложений: код Хаффмана”

Цель работы: ознакомиться с основными понятиями «Деревья», «Двоичное дерево», «код Хаффмана» и алгоритмами их обработки, научиться применять полученные знания на практике.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ (ответы на контрольные вопросы):

1. Определение понятия двоичное дерево (далее ДД).

Двоичное дерево – древовидная структура данных, в которой у родительских узлов не может быть больше двух детей.

2. Перечислите типы двоичных деревьев.

- Полное двоичное дерево
- Совершенное двоичное дерево
- Законченное двоичное дерево
- Вырожденное двоичное дерево
- Скошенное вырожденное дерево
- Сбалансированное двоичное дерево

3. Разница совершенного ДД и законченного ДД.

Совершенное двоичное дерево — особый тип бинарного дерева, в котором у каждого внутреннего узла по два ребенка, а листовые вершины находятся на одном уровне.

Законченное двоичное дерево похоже на совершенное, но есть три большие отличия.

1. Все уровни должны быть заполнены.
2. Все листовые вершины склоняются влево.
3. У последней листовой вершины может не быть правого собрата. Это значит, что завершенное дерево необязательно полное.

4. Определение понятия обход дерева.

Обход дерева – это способ последовательного посещения узлов дерева, при котором каждый узел посещается только один раз.

5. Какие типы обходов существуют?

- В ширину
- В глубину

6. Какие характеристики имеет дерево Хаффмана?

- Для того же набора весов доступное дерево Хаффмана не обязательно уникально.

- Левое и правое поддеревья дерева Хаффмана можно поменять местами, поскольку это не влияет на взвешенную длину пути дерева.
- Все узлы с весами - это конечные узлы, а узлы без весов - все корневые узлы суббинарного дерева.
- Узлы с большими весами находятся ближе к корневому узлу дерева Хаффмана, а узлы с меньшими весами находятся дальше от корневого узла дерева Хаффмана.
- В дереве Хаффмана есть только листовые узлы и узлы со степенью 2,

и нет узлов со степенью 1.

- Дерево Хаффмана с n листовыми узлами имеет 2n-1 узлов.

7. Этапы построения дерева Хаффмана.

1. Рассмотрим заданные n весов как n бинарных деревьев только с корневыми узлами (без левого и правого потомков), чтобы сформировать множество HT. Вес каждого дерева - это вес узла.
2. Выберите два двоичных дерева с наименьшими весами из набора HT, чтобы сформировать новое двоичное дерево, вес которого является суммой весов двух двоичных деревьев.
3. Удалите два двоичных дерева, выбранных на шаге 2, из набора HT, и добавьте вновь полученное двоичное дерево на шаге 2 в набор HT.
4. Повторяйте шаги 2 и 3 до тех пор, пока набор HT не будет содержать только одно дерево, которое является деревом Хаффмана.

Вариант 11

```
class Node {
  // узел бинарного дерева
  constructor(data) {
    this.data = data;
    this.children = [];
  }
}
```

Рисунок 1 - Класс узла бинарного дерева

```
class Tree {
  // класс дерева
  constructor(data) {
    this.root = new Node(data); // корневой узел
  }
  add(data, currentNode) {
    // произвольное заполнение
    if (currentNode.children.length == 0) {
      // если нет детей, то добавить слева или справа
      console.log(`Добавление узла ${data} к узлу ${currentNode.data}`);
      currentNode.children.push(new Node(data, currentNode));
      tasks();
    } else if (currentNode.children.length == 1) {
      // если ребенок один, то выбор между добавлением узла к ребенку или к корню
      rl.question(
        `Выберете к какому элементу добавить узел, к ${currentNode.children[0].data}
        (index) => {
          if (index == 0) {
            this.add(data, currentNode.children[index]);
          } else if (index == 1) {
            currentNode.children.push(new Node(data, currentNode));
            tasks();
          } else {
            tasks();
          }
        }
      );
    } else if (currentNode.children.length == 2) {
      // если два, то вызов этой же функции для элемента
      rl.question(
        `Выберете к какому элементу добавить узел, к ${currentNode.children[0].data}
        (index) => {
          if (index < 2) {
            this.add(data, currentNode.children[index]);
          } else {
            tasks();
          }
        }
      );
    }
  }
  display(currentNode) {
    console.log("-------");
    console.log("Корень : " + currentNode.data);
    for (let i = 0; i < currentNode.children.length; i++) {
      console.log("Ребенок : " + currentNode.children[i].data);
    }
    for (let i = 0; i < currentNode.children.length; i++) {
      this.display(currentNode.children[i]);
    }
  }
```

Рисунок 2 - Базовые методы для работы с бинарным деревом

```
preorderTraversal(currentNode) {
  if (currentNode == null) {
    return;
  }
  console.log(currentNode.data);
  this.preorderTraversal(currentNode.children[0]);
  this.preorderTraversal(currentNode.children[1]);
}
postorderTraversal(currentNode) {
  if (currentNode == null) {
    return;
  }
  this.preorderTraversal(currentNode.children[0]);
  this.preorderTraversal(currentNode.children[1]);
  console.log(currentNode.data);
}
orderTraversal(currentNode) {
  if (currentNode == null) {
    return;
  }
  this.preorderTraversal(currentNode.children[0]);
  console.log(currentNode.data);
  this.preorderTraversal(currentNode.children[1]);
}
```

Рисунок 3 - Типы обхода дерева

```
delete(data, currentNode) {
  /*

  i = эта переменая показывает слева или справа находится удаляемый элемент,
  это нужно что-бы потом, при удалении его из дерева, его родитель на его место
  количества детей у корня и у удаляемого элемента

  */
  for (let i = 0; i < currentNode.children.length; i++) {
    // поиск по всем детям
    if (currentNode.children[i].data ≠ data) {
      this.delete(data, currentNode.children[i]);
    } else {
      if (currentNode.children[i].children.length ≠ 0) {
        // если элемента есть дети, то
        if (currentNode.children.length == 1) {
          // если у корня только удаляемый элемент в детях, то меняем его на все
          for (let k = 0; k < currentNode.children[i].children.length; k++) {
            if (currentNode.children[k] == null) {
              currentNode.children.push(new Node(null));
            }
            currentNode.children[k] = currentNode.children[i].children[k];
          }
        } else {
          currentNode.children[i] = currentNode.children[i].children[0];
        }
      } else {
        currentNode.children.splice(i, 1);
      }
```

Рисунок 4 - Удаление из бинарного дерева

```
1-Добавить узел
2-Вывод дерева
3-Удалить элемент
4-Найти повторяющиеся элементы
5-Обход дерева
Выберите задание : 1
Добавить узел : 67

LEFT OR RIGHT : left

 CURRENT NODE : + 4

LEFT OR RIGHT : right

1-Добавить узел
2-Вывод дерева
3-Удалить элемент
4-Найти повторяющиеся элементы
5-Обход дерева
Выберите задание : 2

4 <- 1 -> NULL

CHOOSE WAY : 4

NULL <- 4 -> 67

 : 67

Конец дерева....

NULL <- 4 -> 67
```

Рисунок 5 - Результат работы программы