МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Деревья

Студент гр. 9383 Ноздрин В.Я.

Преподаватель Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Научиться работать с бинарными деревьями, изучить способ их реализации, решить с их помощью рекурсивную задачу.

Задание.

Задано бинарное дерево b типа BT с произвольным типом элементов.
Используя очередь, напечатать все элементы дерева b по уровням: сначала из корня дерева,
затем из узлов, сыновних по отношению к корню, затем из узлов, сыновних по отношению к этим узлам, и т.д.

Теория.

Рекурсия — это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе. Рекурсия очень широко применяется в математике и программировании. Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m 0 попарно не пересекающихся множествах T 1 , T 2 , ..., T m , каждое из которых, в свою очередь, является деревом.

Бинарное дерево - это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Ход работы:

1. Анализ задания
2. Разработка программного кода.
3. Тестирования программы.

Выполнение работы:

В первую очередь для работы с деревьями была реализована структура **Node**, позволяющая реализовать структуру дерева через указатели. Также по заданию необходимо использовать очередь **Queue** для печати данных из вершин дерева.

Класс **Node** содержит три поля: **m_data**, **m_left**, **m_rigth** — для хранения данных и указателей на дочерние вершины. В случае когда правой или левой вершины нет, значение соответствующего указателя устанавливается в **nullptr**.

Класс **Queue** содержит в себе подкласс **Element,** хранящий данные в поле **m_data**, а также указатели на предыдущий (**m_prev**) и следующий (**m_next**) элементы цепочки элементов **Element** в очереди **Queue**. Для работы с очередью реализованы методы **top**, **pop**, **push**, которые соответственно возвращают элемент из очереди (первый), убирают элемент из очереди (первый), либо же добавляют элемент в очередь (первым).

Главная функция выполняющая задание **rLR_traversal** принимает на вход очередь вершин одного уровня, добавляет данные из этих вершин в очередь печати и рекурсивно вызывает себя же, передавая первым аргументом очередь с уже следующим уровнем бинарного дерева.

Также были написаны функции чтения бинарного дерева, возвращающие деревья, реализованные с помощью структуры **Node**, и читающие из потока ввода текстовое описание дерева в формате (**data left right**) в стиле **depth-first** алгоритмов.

В функции **main** с помощью функции **readStringTree** считывается дерево из соответствующего файла, создается очередь, состоящая из одного только корня дерева, а также очередь на печать, и вызывается функция **rLR_traversal.** После этого печатается на экран содержимое очереди печати **printQueue.**

Пример работы программы:

Входные данные (в файлах binTree1.txt, binTree2.txt, binTree3.txt):

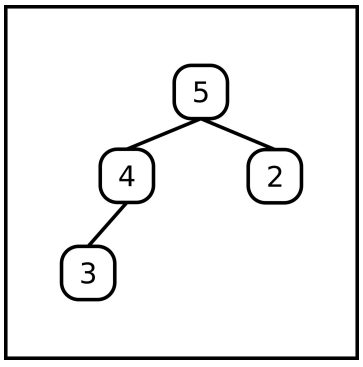

рис.1 Дерево из файла binTree1

binTree1:

5 1 1

4 1 0

3 0 0

2 0 0

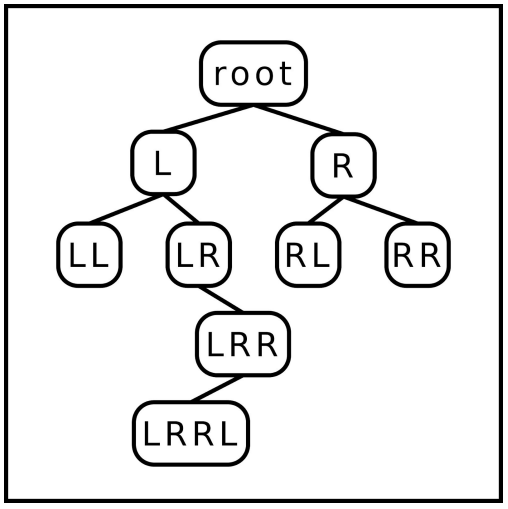

рис.2 Дерево из файла binTree2

binTree2:

root 1 1

L 1 1

LL 0 0

LR 0 1

LRR 1 0

LRRL 0 0

R 1 1

RL 0 0

RR 0 0

binTree3:

ROOT 1 1

I 0 1

A 1 1

1 0 0

2 0 0

II 1 1

B 0 0

C 1 1

3 0 0

4 0 0

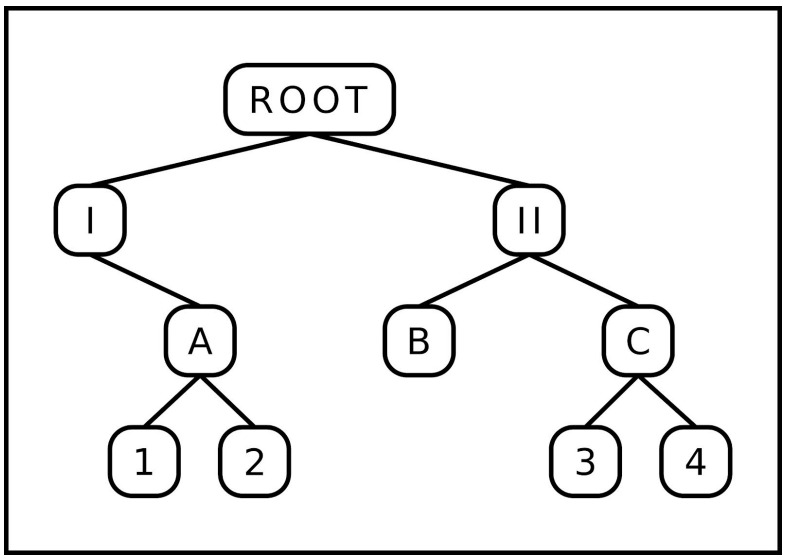

рис.3 Дерево из файла binTree3

Выходные данные:

5 4 2 3

root L R LL LR RL RR LRR LRRL

ROOT I II A B C 1 2 3 4

Разработанный программный код см. в приложении А.

Вывод.

Получены навыки работы с бинарными деревьями, изучен способ их реализации. Была решена рекурсивная задача с помощью бинарных деревьев на языке программирования C++.

Приложение А

Исходный код программы

Название файла: main.c

```
#include <iostream>
#include <fstream>
#include <string>

template <class T>
class Node {
 public:
  Node(T data)
    : m_data(data), m_left(nullptr), m_right(nullptr){};
  ~Node()=default;

  T m_data;
  Node<T>* m_left;
  Node<T>* m_right;
};

template <class T>
class Queue {
 public:
   /*--------Queue Element--------*/
  class Element {
   public:
    Element(T data)
    : m_data(data),  m_prev(nullptr), m_next(nullptr) {};

    T m_data;
    Element *m_prev;
    Element *m_next;
  };
  /*------------------------------*/

 public:
  Queue()
    : m_first(nullptr), m_last(nullptr) {};
  Element *top() {                                          // top
    return m_last;
  }

  void pop() {                                              // pop
    if (m_last) {
      m_last = m_last->m_prev;
      if (m_last) {
        delete m_last->m_next;
        m_last->m_next = nullptr;
      }
    }
  };

  void push(T data) {                                      // push
    auto *elem = new Element(data);
    if (m_first) {
      elem->m_next = m_first;
```

```
      m_first->m_prev = elem;
      m_first = elem;
    } else {
      m_first = elem;
      m_last = elem;
    }
  };

 private:
  Element *m_first;
  Element *m_last;
};

template <class T>
void rLR_traversal(Queue<Node<T>*> *currentLevel,
                   Queue<T> *printQueue) {
  auto nextLevel = new Queue<Node<T>*>();

  while (currentLevel->top()) {
    // add data to printQueue
    printQueue->push(currentLevel->top()->m_data->m_data);

    // add left node to nextLevel
    if (currentLevel->top()->m_data->m_left)
      nextLevel->push(
        currentLevel->top()->m_data->m_left
      );

    // add right node to nextLevel
    if (currentLevel->top()->m_data->m_right)
      nextLevel->push(
        currentLevel->top()->m_data->m_right
      );

    currentLevel->pop();
  };

  if (nextLevel->top())
    rLR_traversal(nextLevel, printQueue);

  delete nextLevel;
};

Node<int>* readIntTree (std::fstream *input) {
  std::string line;
  getline(*input, line);

  int size = line.size();
  auto node = new Node<int>(std::stoi(line));

  if (line[size-3] - '0' ==  1)
    node->m_left = readIntTree(input);

  if (line[size-1] - '0' ==  1)
    node->m_right = readIntTree(input);

  return node;
}
```

```
Node<std::string>* readStringTree (std::fstream *input) {
  std::string line;
  getline(*input, line);

  int size = line.size();
  auto node = new Node<std::string>(line.substr(0, size-4));

  if (line[size-3] - '0' ==  1)
    node->m_left = readStringTree(input);

  if (line[size-1] - '0' ==  1)
    node->m_right = readStringTree(input);

  return node;
}

int main() {

  auto input = new std::fstream("binTree3.txt");

  auto printQueue   = new Queue<std::string>();
  auto currentLevel = new Queue<Node<std::string>*>();
  auto root         = readStringTree(input);

  currentLevel->push(root);

  rLR_traversal<std::string>(currentLevel, printQueue);

  // print result
  while (printQueue->top()) {
    std::cout << printQueue->top()->m_data <<  " ";
    printQueue->pop();
  };

  return 0;
}
```