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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарное дерево

Студент гр. 7383 _____ Бергалиев М.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

1. Цель работы

Цель работы: познакомиться со структурой и реализацией бинарного дерева на языке программирования C++.

Формулировка задачи: Для заданного бинарного дерева *b* типа *BT* с произвольным типом элементов:

а) определить максимальную глубину дерева *b*, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

б) вычислить длину внутреннего пути дерева *b*, т. е. сумму по всем узлам длин путей от корня до узла;

в) напечатать элементы из всех листьев дерева *b*;

г) подсчитать число узлов на заданном уровне *n* дерева *b* (корень считать узлом 1-го уровня);

д) определить, есть ли в дереве *b* хотя бы два одинаковых элемента.

2. Реализация задачи

В функции main выводится предложение выбрать способ ввода данных либо выйти из программы. В случае выбора файла, программа предлагает ввести имя файла. Создается объект типа std::istream, из которого считывается строка. В случае ввода информации с консоли, то она считывается из std::cin. После считывания данных вызывается функция from_string, создающая бинарное дерево чисел, которой передается считанная строка. Далее вызываются методы, решающие каждую подзадачу, и выводятся результаты на экран и процесс вновь повторяется с приглашения для выбора способа ввода данных. Если в ходе процесса были введены некорректные данные, то выводится сообщение о неправильных данных и процесс продолжается сначала.

Переменные, используемые в функции main:

- command — строка, содержащая номер команды, отвечающий за выбор способа ввода данных или выхода из программы.
- file — буффер потока файла, участвующий в создании объекта fin

типа istream, передаваемого в функцию input_parser.

- filename — имя файла, из которого берутся входные данные.
- input — считанная строка исходных данных.
- level — уровень бинарного дерева, для которого надо посчитать количество одинаковых элементов.
- tree — бинарное дерево, созданное из полученной строки.

Функция to_string принимает на вход итератор на сторку. Сначала проверяется, присутствует ли скобка на месте, куда указывает итератор. Если ее нет, то выбрасывается исключение, поскольку строка некорректна. Далее считывается число и создается корень дерева. Если его нет, то выбрасывается соответствующее исключение. После считывается произвольное число пробелов. Если далее идет скобка и после нее сразу не идет закрывающая, то вызывается функция to_string от итератора и созданный объект передается в метод push_left, иначе объект не создается и левого поддерева нет. Если скобки нет, то в метод передается узел, созданный из считанного числа. Тоже самое для правого поддерева. После проверяется, идет ли дальше скобка. Если ее нет, то выбрасывается исключение.

Переменные, используемые в функции to_string:

- num — считанное число.
- count — число считанных цифр, если их нет, то числа нет.
- res — созданное бинарное дерево.

Шаблонный класс BinTree с шаблонным параметром T (тип хранимых элементов) представляет из себя бинарное дерево на базе указателей. Класс содержит следующие поля:

- value — хранимое в узле значение.
- left — указатель на левое поддерево.
- right — указатель на правое поддерево.

Методы класса BinTree:

- Конструктор, принимающий объект класса T, создающий узел с

хранимым переданным значением.

- Конструктор копирования, принимающий константный объект класса BinTree.
- push_left — принимает бинарное дерево. Удаляется левое поддерево из данного дерева и вместо него создается копия переданного дерева.
- push_right — принимает бинарное дерево. Удаляется правое поддерево из данного дерева и вместо него создается копия переданного дерева.
- depht — рекурсивно находит глубину дерева. Выбирается максимум из глубин поддеревьев с добавленной единицей.
- node_number — возвращает число узлов в дереве. Суммирует число узлов в левом и правом поддереве и добавляет 1.
- internal_lenght — находит длину внутреннего пути. Для этого складываются длины внутренних путй левого и правого поддеревьев, а также число узлов в поддеревьях.
- print_leafs — выводит листья в переданный поток вывода.
- nlvl_node_number — находит число узлов на данном уровне дерева, вызывая данный метод для поддеревьев.
- get — возвращает значение в узле с номером n. Узлы нумеруются с 0 в порядке КЛП.
- equal-elem — перебирает все пары вершин в дереве. Если элементы совпадают, то возвращается true. Если одинаковых элементов нет, то возвращается false.
- Деструктор, высвобождающий память и вызывающий деструкторы левого и правого поддеревьев.

3. Тестирование

Программа была собрана в компиляторе G++ с ключом -std=c++14 в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования ошибки не были найдены.

Некорректный случай представлен в табл. 1, в котором пропущено число. В данном случае строка не является бинарным коромыслом, потому найти число гирек невозможно.

Таблица 1 - Некорректный случай с пропущенным числом

Входные данные	Результат
(1 2)	Missing number: (1 2) ^

Корректные тестовые случаи представлены в приложении А.

4. Вывод

В ходе работы были получены навыки работы с бинарным деревом. Был реализован шаблонный класс, представляющий из себя бинарное дерево на базе указателей. Бинарное дерево является рекурсивной структурой, для которой легко определяются рекурсивные методы.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
(1 2 (3 (4 5 6) 2)) 2	Depth: 3 Internal way lenght: 12 Leafs: 2 5 6 2 2-level's number of nodes: 2 There is equal elements
(1 (3 (4 6 2) (7 0 9)) (8 5 (10 11 (19 17 18)))) 3	Depth: 4 Internal way lenght: 36 Leafs: 6 2 0 9 5 11 17 18 3-level's number of nodes: 4 There is no equal elements
(3 () ()) 1	Enter file name: test3 Depth: 0 Internal way lenght: 0 Leafs: 3 1-level's number of nodes: 1 There is no equal elements
(1 (2 () (3 (4 5 ()) ())) ()) 4	Depth: 4 Internal way lenght: 10 Leafs: 5 4-level's number of nodes: 1 There is no equal elements

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile:

```
all: main.cpp BinTree.hpp
      g++ main.cpp -std=c++14 -o BinTree
```

main.cpp:

```
#include <iostream>
#include <fstream>
#include "BinTree.hpp"


BinTree<int> from_string(std::string::iterator &it){
    while(isspace(*it))
      ++it;
    if(*it != '(')
        throw std::invalid_argument("Missing opening bracket");
    ++it;
    while(isspace(*it))
      ++it;
    char* num = &(*it);
    int count = 0;
    while(isdigit(*it)) {
        ++count;
        ++it;
    }
    if(count == 0)
        throw std::invalid_argument("Missing number");
    BinTree<int> res(atoi(num));
    while(isspace(*it))
        ++it;
    if(*it == '('){
      if(*(it+1) != ')')
          res.push_left(from_string(it));
      else it += 2;
    }
    else{
      num = &(*it);
      count = 0;
        while(isdigit(*it)) {
            ++count;
            ++it;
        }
        if(count == 0)
            throw std::invalid_argument("Missing number");
      res.push_left(BinTree<int>(atoi(num)));
    }
    while(isspace(*it))
      ++it;
    if(*it == '('){
      if(*(it+1) != ')')
          res.push_right(from_string(it));
```

```
      else it += 2;
    }
    else{
      num = &(*it);
      count = 0;
        while(isdigit(*it)) {
            ++count;
            ++it;
        }
        if(count == 0)
            throw std::invalid_argument("Missing number");
      res.push_right(BinTree<int>(atoi(num)));
    }
    if(*it != ')')
        throw std::invalid_argument("Missing closing bracket");
    ++it;
    return res;
}

int main(){
    std::string command;
    std::filebuf file;
    std::string filename;
    std::string input;
    while(true){
      std::cout << "Enter 0 to read input from consol or 1 to read from file
or 2 to exit: ";
      getline(std::cin, command);
      try{
          if(std::stoi(command) == 2)
            break;
      }
      catch(std::exception &e){
          std::cout << "Invalid command, try again" << std::endl;
          continue;
      }
      switch(std::stoi(command)){
          case 0:{
            std::cout << "Enter binary tree of numbers:" << std::endl;
            getline(std::cin, input);
            break;
          }
          case 1:{
            std::cout << "Enter file name: ";
            getline(std::cin, filename);
            if(file.open(filename, std::ios::in)){
                std::istream fin(&file);
                getline(fin, input);
                file.close();
            }
            else{
                std::cout << "Incorrect filename" << std::endl;
                file.close();
                continue;
```

```
            }
            break;
          }
          default:{
            std::cout << "Incorrect command, try again" << std::endl;
            continue;
          }
      }
        std::string::iterator it = input.begin();
      try{
            BinTree<int> tree = from_string(it);
            std::cout << "Depth: " << tree.depth() << std::endl;
          std::cout << "Internal way lenght: " << tree.internal_lenght() <<
std::endl;
          std::cout << "Leafs:" << std::endl;
          tree.print_leafs(std::cout);
          std::cout << std::endl;
          std::string level;
          std::cout << "Enter level of tree: ";
          getline(std::cin, level);
          try{
            std::cout << level << "-level's number of nodes: " <<
tree.nlvl_node_number(std::stoi(level)) << std::endl;
          }
          catch(std::exception &e){
            std::cout << "It isn't number" << std::endl;
          }
          if(tree.equal_elem())
            std::cout << "There is equal elements" << std::endl;
          else
            std::cout << "There is no equal elements" << std::endl;
      }
      catch(std::exception &e){
          std::cout << e.what() << ":" << std::endl;
          std::cout << input << std::endl;
          std::cout << std::string(distance(input.begin(), it), ' ') << '^'
<< std::endl;
      }
    }
    return 0;
}
```

BinTree.hpp:

```
#pragma once

template <class T>

class BinTree{
public:
    BinTree(T value) : left(nullptr), right(nullptr), value(value){}
    BinTree(BinTree const& tree){
      value = tree.value;
      if(tree.left != nullptr)
          left = new BinTree(*tree.left);
      else left = nullptr;
```

```
  if(tree.right != nullptr)
      right = new BinTree(*tree.right);
  else right = nullptr;
}
void push_left(BinTree<T> tree){
  delete left;
  left = new BinTree<T>(tree);
}
void push_right(BinTree<T> tree){
  delete right;
  right = new BinTree<T>(tree);
}
unsigned int depth(){
  unsigned int lenght = 0;
  if(left != nullptr)
      lenght = left->depth() + 1;
  if(right != nullptr){
      unsigned int right_lenght = right->depth() + 1;
      if(right_lenght > lenght)
        lenght = right_lenght;
  }
  return lenght;
}
unsigned int node_number(){
  unsigned int res = 1;
  if(left != nullptr)
      res += left->node_number();
  if(right != nullptr)
      res += right->node_number();
  return res;
}

unsigned int internal_lenght(){
  unsigned int lenght = 0;
  if(left != nullptr)
      lenght += left->internal_lenght() + left->node_number();
  if(right != nullptr)
      lenght += right->internal_lenght() + right->node_number();
  return lenght;
}

void print_leafs(std::ostream &output){
  if(left == nullptr && right == nullptr)
      output << value << " ";
  if(left != nullptr)
      left->print_leafs(output);
  if(right != nullptr)
      right->print_leafs(output);
}

unsigned int nlvl_node_number(unsigned int n){
  if(n == 1)
      return 1;
  int res = 0;
```

```
      if(left != nullptr)
          res += left->nlvl_node_number(n-1);
      if(right != nullptr)
          res += right->nlvl_node_number(n-1);
      return res;
    }

    T get(unsigned int n){
      if(n == 0)
          return value;
      unsigned int left_num = 0;
      if(left != nullptr)
          left_num = left->node_number();
      if(n > left_num)
          return right->get(n - left_num - 1);
      return left->get(n-1);
    }

    bool equal_elem(){
      for(unsigned int i=0; i<node_number(); ++i)
          for(unsigned int j=0; j<node_number(); ++j)
            if(i != j && get(i) == get(j))
                return true;
      return false;
    }

    ~BinTree(){
      delete left;
      delete right;
    }

private:
    T value;
    BinTree<T>* left;
    BinTree<T>* right;
};
```