МИНОБРНАУКИ РОССИИ
САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 7381 _____ Дорох С.В.

Преподаватель _____ Фирсов М.А.

Санкт–Петербург

2018

Задание

Вариант 1(а-в):

Задано бинарное дерево *b* типа *BT* с типом элементов *Elem*. Для введенной пользователем величины *E* (**var** *E*: *Elem*):

а) определить, входит ли элемент *E* в дерево *b*;

б) определить число вхождений элемента *E* в дерево *b*;

в) найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E*;

Реализация дерева должна быть основана на массиве.

Пояснение задания

Наиболее важным типом деревьев являются бинарные деревья. Удобно дать следующее формальное определение. Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Например, бинарное дерево, изображенное на рис. 1, имеет скобочное представление:

(*a* (*b* (*d* # (*h* ##)) (*e* ##)) (*c* (*f* (*i* ##) (*j* ##)) (*g* # (*k* (*l* ##) #)))).

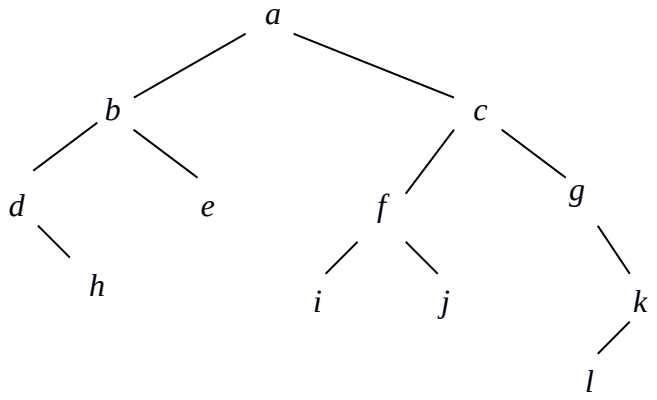

Рис. 1 – Бинарное дерево

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= #,

< непустое БД > ::= (< корень > < БД > < БД >).

Задача каждого из пунктов сводится к рекурсивному обходу дерева, заданного скобочным выражением, и выполнение определенных действий.

Описание алгоритмов

а) Пока узел не является искомым элементом происходит рекурсивный переход в левое и правое поддеревья и там проверяется является ли узел дерева нужным нам элементом. Если узел дерева совпадает с нужным нам значением, то выводится соответствующее сообщение.

б) Одновременно при выполнении пункта а выполняется проверка количества вхождений элемента в дереве, если узел является искомым элементом, то увеличивается значение счётчика, после проверки всего дерева возвращается значение счётчика и в соответствии с его значением выводится сообщение о количестве вхождений элемента в дерево.

в) Длина минимального пути для корня равна 0. При переходе в левое или правое поддерево, длина пути увеличивается на 1. При дальнейшем переходе к поддеревьям текущих корней, глубина увеличивается на 1. И это происходит до тех пор, пока мы не натыкаемся на лист – узел дерева, не имеющий детей или на элемент, для которого мы ищем минимальный путь.

При каждом вызове метода подсчета глубины производится проверка на наличие поддеревьев, и если поддеревья есть, то рекурсивно вызывается этот же метод для поддеревьев с прибавлением единицы к промежуточному значению длины пути деревьев, являющихся поддеревьями заданного. После этого выбирается наименьшее значение глубины левого или правого поддерева, которое возвращается, как результат выполнения метода, и также будет сравнен со значением, полученным в другом поддереве, и также будет сравнен, только на предыдущем шаге рекурсии

Описание функций

`void print_tabs(int count);` – функция, печатающая необходимое количество символов табуляции.

`int count` – количество выводимых символов табуляции.

Возвращаемое значение: функция ничего не возвращает.

`int lvl(int count);` -- функция, вычисляющая уровень, на котором находится узел дерева.

`int count` – индекс узла в дереве.

Возвращаемое значение: уровень соответствующий элементу.

`int leftChild(int count);` - функция, возвращающая индекс левого сына узла;

`int count` – индекс текущего узла дерева.

Возвращаемое значение: индекс левого сына.

`int rightChild(int count);` - функция, возвращающая индекс правого сына узла;

`int count` – индекс текущего узла дерева.

Возвращаемое значение: индекс правого сына.

`int readEmptyBT(char *buf, int count, std::istream &list);` - функция, считывающая корень бинарного дерева;

`char* buf` – массив, хранящий узлы бинарного дерева;

`int count` – индекс текущего узла дерева;

`std::istream &list` – ссылка на потоковый ввод list.

Возвращаемое значение: функция чтения бинарного дерева.

`bool readEmpty(char *buf, int count, std::istream &list);` - функция, считывающая узлы бинарного дерева;

`char* buf` – массив, хранящий узлы бинарного дерева;

`int count` – индекс текущего узла дерева;

`std::istream &list` – ссылка на потоковый ввод list.

Возвращаемое значение: true – если считывание прошло успешно, иначе false.

`int numberOfElem(char* buf, size_t count, char elem, int &number);` - функция, проверяющая есть ли элемент в узлах дерева, а также считает количество вхождений элемента в дерево;

`char* buf` – массив, хранящий узлы бинарного дерева;

`size_t count` – индекс текущего узла дерева;

`char elem` – элемент для проверок;

`int &number` – ссылка на счётчик вхождений;

Возвращаемое значение: количество вхождений элемента в дерево.

`int depthOfBT(char* buf, size_t count, char elem);` - функция, вычисляющая минимальную длину пути до элемента в дереве;

`char* buf` – массив, хранящий узлы бинарного дерева;

`size_t count` – индекс текущего узла дерева;

`char elem` – элемент для проверок;

Возвращаемое значение: минимальная длина пути до элемента в дереве.

Тестирование

Для проверки работоспособности программы был создан скрипт(см. ПРИЛОЖЕНИЕ) для автоматического ввода и вывода тестовых данных:

Корректные тесты:

Test 1: (a(b(d(g##)(e##))(e(d##)(p##)))(c#(g(e##)#)))

Test 2: (a(b##)#)

Test 3: (a (b# #)(c# (k##)))

Некорректные тесты:

Test 4: ()

Test 5: (abc)

Test 6: (a(((b##))))

Test 7:

Результаты тестирования сохраняются в файл result.txt.

Ниже представлены результаты тестирования.

Скобочная запись дерева	Результат тестирования
(a(b(d(g##)(e##))(e(d##)(p##)))(c#(g(e##)#)))	Entered tree: (a(b(d(g##)(e##))(e(d##)(p##)))(c#(g(e##)#))) Tree was read: abcde#ggedp##e######### Enter a character to check >> e call: Failure! Element 'a' not 'e' call: Failure! Element 'b' not 'e' call: Failure! Element 'd' not 'e' call: Failure! Element 'g' not 'e' llac call: Success! Element 'e' found at 9 position llac llac call: Success! Element 'e' found at 5 position call: Failure! Element 'd' not 'e' llac call: Failure! Element 'p' not 'e' llac llac llac call: Failure! Element 'c' not 'e' call: Failure! Element 'g' not 'e' call: Success! Element 'e' found at 14 position llac llac llac llac The number of occurences of 'e': 3

	Minimal path length for this step: 1 Not a leaf or disered element 'e' Minimal path length for this step: 1 Not a leaf or disered element 'e' Minimal path length for this step: 1 Not a leaf or disered element 'e' Minimal path length for this step: 1 Minimal path length for this step: 2 Minimal path length for this step: 2 Path length to symbol 'e' equally 2
(a(b##)#)	Entered tree: (a(b##)#) Tree was read: ab### Enter a character to check >> a The number of occurences of 'a': 1 Path length to symbol 'a' equally 0
(a (b# #)(c# (k##)))	Entered tree: (a(b##)(c#(k##))) Tree was read: abc###k Enter a character to check >> k The number of occurences of 'k': 1 Path length to symbol 'k' equally 2
()	Entered tree: () Error! Subtree can start only with '(' or '#'.
(abc)	Entered tree: (abc) Error! Subtree can start only with '(' or '#'.
(a(((b##))))	Entered tree: (a(((b##)))) Error! Subtree can start only with '(' or '#'.
	Entered tree: Error! Incorrect element at first leaf!

Выводы

В процессе выполнения лабораторной работы были поли получены знания и навыки по ООП в C++. Были изучены бинарные деревья, повторена работа с рекурсивным функциям, bash–скриптам и автоматизации тестирования. Работа была написана на C/C++.

ПРИЛОЖЕНИЕ А

КОД MAIN.CPP

```
#include <iostream>
#include <sstream>
#include <cmath>
#include <algorithm>
#include <cstring>

int leftChild (int count) {
    return 2*count + 1;
}

int rightChild (int count) {
    return 2*count + 2;
}

bool readBT(char* buf, int count, std::istream &list) {
    char elem;
    list >> buf[count];
    list >> elem;
    if(elem != '(' && elem != '#') {
        std::cout << "Error! Subtree can start only with '(' or '#'." <<
std::endl;
        return false;
    }
    if(elem == '(') {
        if(!readBT(buf, leftChild(count), list))
            return false;
        list >> elem;
        if(elem != ')') {
            std::cout << "Error! Waiting ')', leaf cant have more then 2
sons" << std::endl;
            return false;
        }
    }
    if(elem == '#')
        buf[leftChild(count)] = elem;
    list >> elem;
    if(elem != '(' && elem != '#') {
        std::cout << "Error! Subtree can start only with '(' or '#'." <<
std::endl;
        return false;
    }
    if(elem == '(') {
        if(!readBT(buf, rightChild(count), list))
            return false;
        list >> elem;
```

```
        if(elem != ')') {
            std::cout << "Error! Waiting ')', leaf cant have more then 2
sons" << std::endl;
            return false;
        }
    }
    if(elem == '#')
        buf[rightChild(count)] = elem;
    return true;
}

int lvl(int count) {
    int lvl = 1;
    while((pow(2, lvl-1)-1) < count && count > (pow(2, lvl)-2))
        lvl++;
    return lvl;
}

void print_tabs(int count) {
    int lvl_s = lvl(count);
    for(int i = 0; i < lvl_s; i++)
        std::cout << '\t';
}

int numberOfElem(char *buf, size_t count, char elem, int &number) {
//Функция, проверяющая имеется ли заданный элемент в узлах дерева
    if(count <= strlen(buf)) {
//и количество вхождений данного элемента в дерево
        if(elem == buf[count]) {
            print_tabs(count);
            std::cout << "call: Success! Element '" << elem << "' found
at " << count+1 << " position" << std::endl;
            ++number;
        }
        else if(buf[count] != '#' && buf[count] != elem && buf[count] !=
'\0') {
            print_tabs(count);
            std::cout << "call: Failure! Element '" << buf[count] << "'
not '" << elem << "'" << std::endl;
        }
        numberOfElem(buf, leftChild(count), elem, number);
        numberOfElem(buf, rightChild(count), elem, number);
        if(buf[count] != '\0' && buf[count] != '#') {
            print_tabs(count);
            std::cout << "llac" << std::endl;
        }
    }
    return number;

}
```

```
int depthOfBT(char *buf, size_t count, char elem) {
//Функция находящая минимальную длину пути до заданного элемента
    size_t left_depth = 1;
    size_t right_depth = 1;
    if(buf[count] == elem || (2*count+1 > strlen(buf) && 2*count+2 >
strlen(buf))) {            //Если узел дерева не лист или искомый элемент
заканчиваем рекурсию
        if(buf[count] != '#' && (isalpha(buf[count]) ||
isdigit(buf[count])))
            std::cout << "\tNot a leaf or disered element '" <<
buf[count] << '\'' << std::endl;
        return 0;
    }
    else {
        left_depth += depthOfBT(buf, 2*count+1, elem);
        right_depth += depthOfBT(buf, 2*count+2, elem);
        if(buf[count] != '#')
            std::cout << "\tMinimal path length for this step: " <<
std::min(right_depth, left_depth) << std::endl;
        return std::min(right_depth, left_depth);
    }
}

int readEmptyBT(char *buf, int count, std::istream &list) {
//Функция считывания корня бинарного дерева
    char elem;
    list >> elem;
    if(elem != '(') {
        std::cout << "Error! Incorrect element at first leaf!" <<
std::endl;
        return 0;
    }
    return readBT(buf, count, list);
}

int main() {
    std::stringbuf buffer;
    std::string list;

    std::cout << "Enter a bracket representation of binary tree: ";
    getline(std::cin, list);
    list.erase(std::remove(list.begin(), list.end(), ' '), list.end());
    std::cout << "Entered tree: " << list << std::endl;

    std::istream str(&buffer);
    buffer.str(list);
    int size = 0, max = 0;

    for(unsigned int i = 0; i < list.size(); i++) {
```

```
        if(list[i] == '(')
            ++size;
        if(max < size)
            max = size;
        if(list[i] == ')')
            --size;
        }
    max = pow(2, max) - 1;
    char* buf = new char [max];
    for(int i = 0; i < max; i++)
        buf[i] = '#';

    size_t count = 0;
    if(!readEmptyBT(buf, count, str))
        return 0;
    std::cout << "Tree was read: " << buf << std::endl;
    char elem;
    std::cout << "Enter a character to check >> ";
    std::cin >> elem;
    if(elem != '#'){
        int number = 0;
        if(numberOfElem(buf, count, elem, number))
            std::cout << "The number of occurences of '" << elem << "': "
<< number << std::endl;
        else {
            std::cout << "Error! Symbol '" << elem << "' was not found."
<< std::endl;
            delete[] buf;
            return 0;
        }
        size_t depth = depthOfBT(buf, 0, elem);
        std::cout << "Path length to symbol '" << elem << "' equally " <<
depth << std::endl;
    }
    else std::cout << "Error! Incorrect character." << std::endl;
    delete[] buf;
}
```