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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных »
Тема: Иерархические списки

Студент гр. 8304 _____ Бутко А.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Изучить динамические структуры данных, иерархический список, бинарное коромысло. Попробовать реализовать специальный иерархический список.

Задание.

4 вариант.

Говорят, что бинарное коромысло сбалансировано, если момент вращения, действующий на его левое плечо, равен моменту вращения, действующему на правое плечо (то есть длина левого стержня, умноженная на вес груза, свисающего с него, равна соответствующему произведению для правой стороны), и если все подкоромысла, свисающие с его плеч, также сбалансированы. Написать рекурсивную функцию или процедуру Balanced, которая проверяет заданное коромысло на сбалансированность (выдает значение true, если коромысло сбалансировано, и false в противном случае).

Основные теоретические положения.

Бинарное коромысло устроено так, что у него есть два плеча: левое и правое. Каждое плечо представляет собой (невесомый) стержень определенной длины, с которого свисает либо гирька, либо еще одно бинарное коромысло, устроенное таким же образом.

В соответствии с данным выше рекурсивным определением бинарного коромысла представим бинарное коромысло списком из двух элементов

БинКор ::= (Плечо Плечо),

где первое плечо является левым, а второе – правым. В свою очередь Плечо будет представляться списком из двух элементов

Плечо ::= (Длина Груз),

где Длина есть натуральное число, а Груз представляется вариантами

Груз ::= Гирька | БинКор,

где в свою очередь Гирька есть натуральное число. Таким образом, БинКор есть специального вида иерархический список из натуральных чисел.

Описание основных данных.

Программа считывает информацию из пользовательского файла и выводит результат в файл и консоль.

Первая строка файла имеет такой набор данных:

N L R,

где N – номер “развилки”, L – номер левого “ребенка” (груз или “развилку”), R – номер правого “ребенка”.

Остальные строки содержат в себе следующий набор данных, имеющий вид:

N L R M H,

где N имеет такой же смысл, как и в первой строке, L и R – номера элементов (0, если груз, номер, если развилка) но, M – масса груза (0, если “развилка”), H – длина плеча.

В структуре элемента Node хранятся данные о предыдущем элементе (“родителе”), об следующих элементах справа и слева, если они есть, масса, длина плеча, номера следующих элементов (логика адресации изображена на рисунке 3).

Рассмотрим ввод и получившиеся иерархическое дерево на примере одного из тестов:

0 1 4

1 2 3 0 3

2 0 0 2 1

3 0 0 1 2

4 0 0 1 9

Иерархия элементов изображена на рисунке 1.

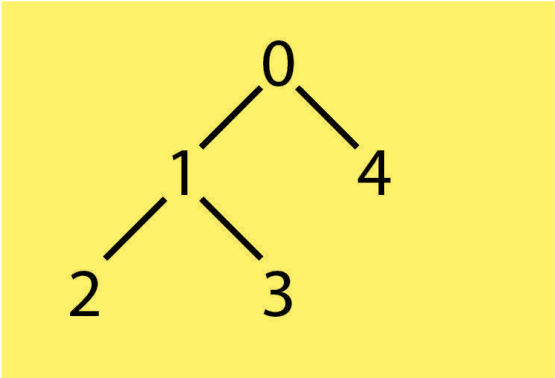

рис.1

Получившееся бинарное коромысло изображено на рисунке 2.

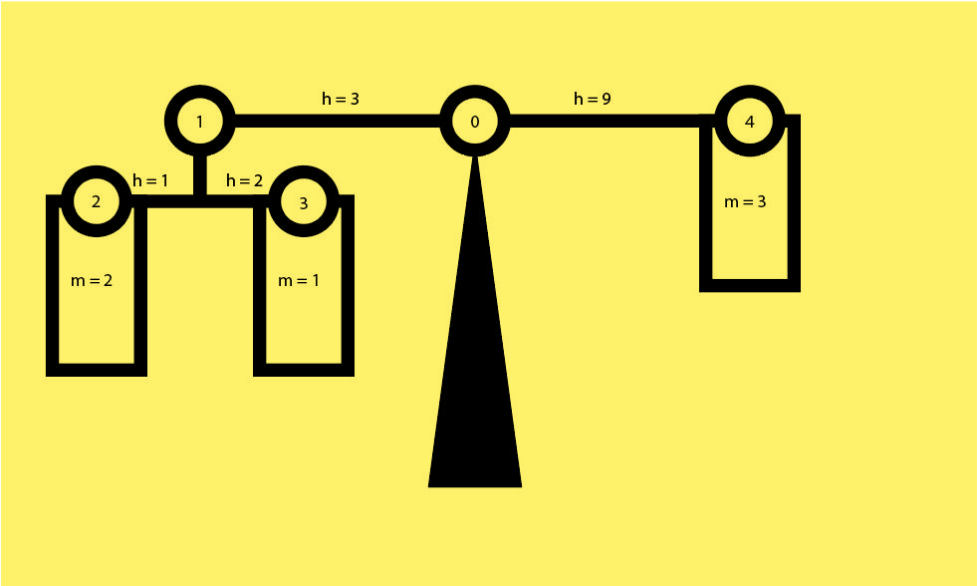

рис. 2

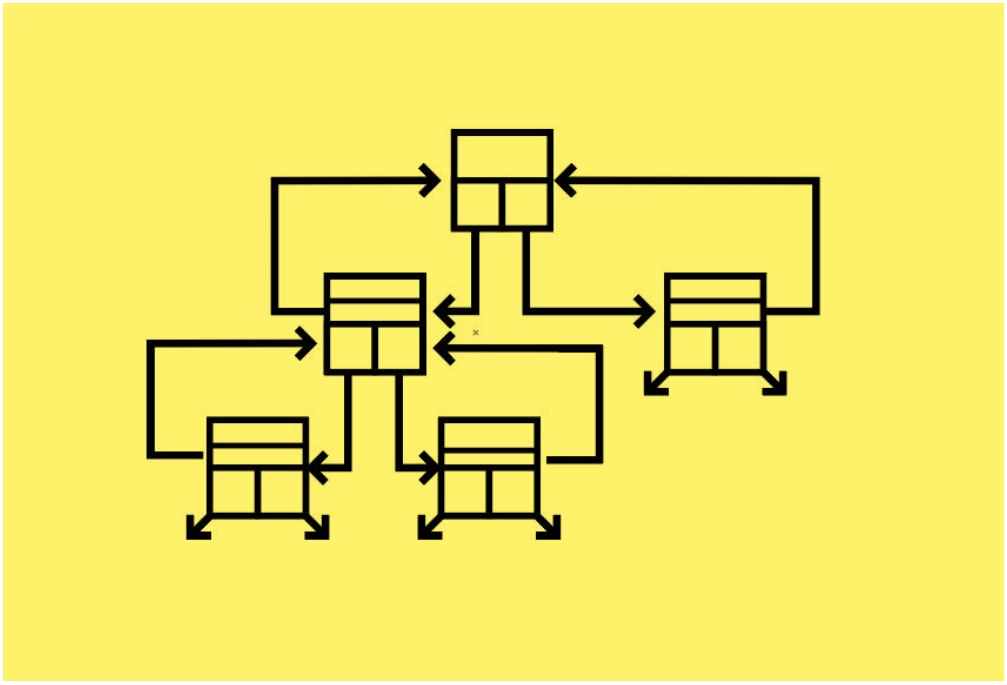
рис. 3

Описание основных функций.

1. add()

Добавление элемента в общую иерархию.

2. balance()

Функция проверяет заданное коромысло на сбалансированность (выдает значение true, если коромысло сбалансировано, и false в противном случае).

3. back()

Функция возвращает указатель на исходное место (элемент под 0 номером).

4. init()

Инициализация 0-го элемента.

5. fromFile()

Функция считывает полный путь до файла, где расположены тесты, и путь до файла, куда сохранить результаты тестов.

Тестирование программы.

TEST:	RESULT:
0 1 2 1 0 0 1 4 2 0 0 5 1	NOT BALANCED
0 1 6 1 2 5 0 1 2 3 4 0 1 3 0 0 1 2 4 0 0 1 2 5 0 0 2 2 6 7 8 0 1 7 0 0 1 2 8 0 0 2 2	NOT BALANCED
0 1 6 1 2 5 0 1 2 3 4 0 1 3 0 0 1 2 4 0 0 2 1 5 0 0 1 3 6 7 8 0 1 7 0 0 2 1 8 0 0 2 1	BALANCED
0 1 2 1 0 0 2 6 2 0 0 3 4	BALANCED
1 2 3 3 4 5 6 7	ERROR: Wrong data!

3 2 4 5 1 3 3 3 1 1	
0 1 2 1 0 3 4 5 2 0 0 5 6 3 0 0 2 5	ERROR: Wrong hierarchical list!

Выводы.

Был создан специальный иерархический список, в котором использовались некоторые принципы двусвязных и односвязных списков. Было изучен принцип бинарного дерева.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

struct Node {
    int N, R, L, H, M;
    bool balance, b_check;
    Node *prev, *left, *right;
};

void init(int n, int l, int r, Node *&bin) {
    bin->left = bin->right = bin->prev = nullptr;
    bin->N = n;
    bin->L = l;
    bin->R = r;
    bin->M = bin->H = 0;
    bin->b_check = false;
    std::cout << n << std::endl;
}

void *back(Node *&bin) {
    if (bin->N == 0) return bin;
    else (back(bin->prev));
}

int add(int n, int l, int r, int m, int h, Node *&bin) {
    Node *tmp = new Node;
    tmp->N = n;
    tmp->R = r;
    tmp->L = l;
    tmp->M = m;
    tmp->H = h;
    tmp->right = tmp->left = nullptr;
    tmp->balance = true;
    /*std::cout << "TMP->N = " << tmp->N << std::endl;
    std::cout << "BIN->L = " << bin->L << std::endl;
    std::cout << "BIN->R = " << bin->R << std::endl;*/
    if (bin->L == tmp->N) {
        tmp->prev = bin;
        std::cout << "left move" << std::endl;
        std::cout << std::endl;
        bin->left = tmp;
        bin = bin->left;
        return 0;
    } else if (bin->R == tmp->N) {
        tmp->prev = bin;
        std::cout << "right move" << std::endl;
        std::cout << std::endl;
        bin->right = tmp;
        bin = bin->right;
        return 0;
    } else {
        bin = bin->prev;
        std::cout << "need to back" << std::endl;
        std::cout << std::endl;
        add(n, l, r, m, h, bin);
        return 0;
```

```
        }
    }

    bool balance(Node *&bin) {
        if (bin->left->left != nullptr && !bin->left->b_check) {
            std::cout << "from: " << bin->N << " to:" << bin->left->N <<
std::endl;
            bin = bin->left;
            balance(bin);
        } else if (bin->right->right != nullptr && !bin->right->b_check) {
            std::cout << "from: " << bin->N << " to:" << bin->right->N <<
std::endl;
            bin = bin->right;
            balance(bin);
        } else {
            bin->left->b_check = bin->right->b_check = true;
            int IL = bin->left->H * bin->left->M;
            int IR = bin->right->H * bin->right->M;
            bin->M = bin->left->M + bin->right->M;
            std::cout << "NL: " << bin->left->N << " IL = " << IL << " ML = "
<< bin->left->M << std::endl;
            std::cout << "NR: " << bin->right->N << " IR = " << IR << " MR = "
<< bin->right->M << std::endl;
            std::cout << "M = " << bin->M << std::endl;
            if (IL == IR && bin->right->balance && bin->left->balance) {
                bin->balance = true;
                std::cout << IL << "=" << IR << std::endl;
            } else {
                bin->balance = false;
                std::cout << IL << "!=" << IR << std::endl;
            }
            bin->b_check = true;
            if (bin->N == 0 && bin->left->b_check && bin->right->b_check) return
bin->balance;
            bin = bin->prev;
            balance(bin);
        }
    }

    int fromFile(std::string path) {
        std::string file_name = path, log_file;
        if (path.size() == 1) {
            std::cout << " Enter test-file location: " << std::endl;
            std::cin >> file_name;
        }
        std::ifstream file;
        file.open(file_name);
        if (!file.is_open()) {
            std::cout << "ERROR: File is not open" << std::endl;
            return 0;
        }
        std::cout << " Enter where to save results (location with <name>.txt):
" << std::endl;
        std::cin >> log_file;
        std::ofstream log(log_file);
        if (!log.is_open()) {
            std::cout << "ERROR: File is not open" << std::endl;
            return 0;
        }
        std::string str, numb;
        Node *bin = new Node;
        int counter = 0;
        while (!file.eof()) {
            getline(file, str);
```

```
            counter ++;
        }
        if ((counter - 1) % 2 != 0) {
            std::cout << "ERROR: Wrong hierarchical list!" << std::endl;
            return 0;
        }
        file.close();
        file.open(file_name);
        while (!file.eof()) {
            getline(file, str);
            std::cout << str << std::endl;
            std::istringstream iss(str);
            std::vector<int> array;
            while (iss >> numb) array.push_back(std::stoi(numb));
            for (int i : array) std::cout << i;
            std::cout << std::endl;
            if (array[0] == 0) {
                init(array[0], array[1], array[2], bin);
            } else if (array.size() == 5) {
                add(array[0], array[1], array[2], array[3], array[4], bin);
            } else {
                std::cout << "ERROR: Wrong data!" << std::endl;
                return 0;
            }
            log << str << std::endl;
        }
        bin = (Node *) back(bin);
        log << std::endl;
        if (balance(bin)) log << "BALANCED" << std::endl;
        else log << "NOT BALANCED" << std::endl;
        log.close();
        return 0;
    }

    int main(int argc, char *argv[]) {
        std::ifstream input;
        if (argc > 1) {
            std::string path = "Tests/";
            path += argv[1];
            std::cout << path << std::endl;
            fromFile(path);
        } else if (argc == 1) {
            fromFile(" ");
        }
        return 0;
    }
```