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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Слабая куча

Студент гр. 9381 Щеглов Д.А.

Преподаватель Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить структуру слабой кучи и написать программу, которая выводит её на экран в наглядном виде.

Задание.

31. Дан массив пар типа «число – бит». Предполагая, что этот массив представляет слабую кучу, вывести её на экран в наглядном виде.

Основные теоретические положения.

Обычная куча представляет собой сортирующее дерево, в котором любой родитель больше (или равен) чем любой из его потомков. В слабой куче это требование ослаблено — любой родитель больше (или равен) любого потомка только из своего правого поддерева. В левом поддереве потомки могут быть и меньше и больше родителя. Такой подход позволяет значительно сократить издержки по поддержанию набора данных в состоянии кучи. Ведь нужно обеспечить принцип «потомок не больше родителя» не для всей структуры, а только её половины. При этом слабая куча, не являясь на 100% сортирующим деревом, сортирует не хуже обычной кучи, а в чём-то даже и лучше. Поскольку в корне кучи, даже слабой, нужен именно максимальный по величине элемент, у корня левого поддерева нет. Также, в работе нужен будет дополнительный битовый массив (назовём его BIT), в котором для i-го элемента отмечено, был ли обмен местами между его левым и правым поддеревьями. Если значение для элемента равно 0, то значит обмена не было. Если значение равно 1, значит, левый и правый потомок идут в обратном порядке. А формулы при этом вот такие:

Левый потомок: $2 \times i + BIT[i]$

Правый потомок: $2 \times i + 1 - BIT[i]$

Описание алгоритма.

Сначала вводится массив, который согласно условию является слабой кучей. Числам, которые не имеют потомков, присваивается условный бит -1.

Далее, по формулам, приведённым в основных теоретических положениях, формируется новый массив, в котором числа расставлены в правильном порядке. После этого вычисляем, сколько чисел будет на каждой строке, и выводим их.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	1 79 -1	79
2.	2 -7 -1 -10 -1	-7 -10
3.	4 500 -1 20 1 34 -1 55 -1	500 20 55 34
4	16 47 -1 15 0 19 0 23 0 7 0 56 1 49 1 32 1 8 0 9 0 77 -1 23 -1 18 -1	47 15 19 23 7 56 49 32 8 9 23 77 50 18 10 12

	50 -1 12 -1 10 -1	

Выводы.

Была изучена структура слабой кучи и написана программа, которая выводит её на экран в наглядном виде.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Source5.cpp

```
#include<iostream>
#include<cstdlib>
#include<cmath>
#include<fstream>

using namespace std;

double logarifm(int a, int b) //вычисляем логарифм по основанию a
от
{
    return log(b) / log(a);
}


typedef struct elem { //структура для записи элементов массива
типа число-бит
    int number;
    int bit;
} elem;

class WeakHeap { //класс слабой кучи
    elem* heap; // массив элементов кучи
    int start_of_heap = 0;
    int end_of_heap = 0;
    int size_of_heap;
public:
    WeakHeap(int size) { //конструктор для создания новой слабой
кучи
        size_of_heap = size;
        heap = new elem[size_of_heap]; //выделяем память под
массив элементов кучи
        for (int i = 0; i < size_of_heap; i++)
            heap[i].number = 0;
    }
    void print_heap();
    void push(elem x);
    void pop();
    ~WeakHeap() {//деструктор для освобождения памяти массива
элементов кучи
        delete[] heap;
    }
};

void WeakHeap::print_heap() {
    cout << " " << heap[0].number;
    int depth = (int)logarifm(2, size_of_heap);//вычисляем глубину
дерева
    if ((int)logarifm(2, size_of_heap) != pow(2, depth))
```

```
       depth += 1;
    cout << "                  добавляем в кучу следующий
элемент\n";
    int k = 0;
    double idt = depth * 2;

   for (int i = 0; i < depth-1; i++) { //цикл для вывода всей кучи

       for (int iter = 0; iter < idt; iter++)
           cout << " ";
       idt = idt / 2;

        for (int j = 0; j < pow(2, i); j++) { //цикл для вывода
строки
                cout << heap[k + 1].number;
               for (int it = 0; it < idt * 4 - 1; it++)
                cout << " ";

           k++;
            }
        cout << "\n";

    }
}


void WeakHeap::push(elem x) { //функция помещения элемента в кучу
    heap[end_of_heap].number = x.number;
    heap[end_of_heap].bit = x.bit;
    end_of_heap++; //так как теперь в куче на один элемент больше
}

void WeakHeap::pop() {   //функуция извлечения элемента из кучи
    if (start_of_heap == end_of_heap) {
        cout << "куча пустая\n";
        return;
    }
    start_of_heap++;
}

int main() {
    setlocale(LC_ALL, "rus"); //смена кодировки консоли
    cout << "Введите количество пар:\n";
    int count;
    cin >> count; // считываем количество пар
    elem* bit_array = new elem[count]; // выделяем динамическую
память под массив бит
    cout << "Введите пары в формате число бит:\n";
    for (int i = 0; i < count; i++) { //считываем массив, который
является слабой кучей
        cin >> bit_array[i].number >> bit_array[i].bit;
    }
    cout << "\n\n";
```

```
    cout << "В конструктуре инициализируем массив для кучи
нулями:\n\n";
    WeakHeap result_heap(count); // Создаётся новая слабая куча
    //Инициализируем её нулями, чтобы не выводился мусор
    result_heap.print_heap();
    cout << "_____________\n";
    result_heap.push(bit_array[0]);
    result_heap.print_heap();
    cout << "_________________\n";
    result_heap.push(bit_array[1]);

    result_heap.print_heap();
    cout << "_________________\n";


    int count_of_bit = count / 2; // количество элементов с не
фиктивным дополнительным битом


    for (int i = 1; i < count_of_bit; i++) {//записываем элементы
в кучу в порядке, удобном для вывода на экран
        if (bit_array[i].bit == 0)
            cout << "Так как бит предка равен 0, оставляем на
месте его потомков:\n";
        else
            cout << "Так как бит предка равен 1, переставляем
местами его потомков:\n";
        result_heap.push(bit_array[2 * i + bit_array[i].bit]); //
Левый потомок: 2 * i + BIT[i]
        result_heap.print_heap();
        cout << "_________________\n";
        result_heap.push(bit_array[2 * i + 1 -
bit_array[i].bit]);// Правый потомок: 2 * i + 1 - BIT[i]
        result_heap.print_heap();
        cout << "_________________\n";
    }


    cout << "\n^^^^^^^^^^^^^^^^ Получившаяся слабая куча выведена
на экран!\n";
    delete[] bit_array; //освобождаем память, чтобы избежать
утечки
    return 0;
     }
```