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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Слабые кучи

Студент гр. 9381 _____ Щеглов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Щеглов Д.А.

Группа 9381

Тема работы: Сортировка слабой кучей. Демонстрация.

Исходные данные:

На вход программе подаётся количество элементов массива и сам целочисленный массив, элементы массива разделены пробелом.

Содержание пояснительной записки:

«Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 03.03.2021

Дата защиты реферата: 03.03.2021

Студент _____ Щеглов Д.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В курсовой работе происходит сортировка массива. Программа демонстрирует процесс сортировки массива слабой кучей при помощи вывода на экран состояния элементов на каждом шаге. Результатом будет являться отсортированный массив.

SUMMARY

IN THE COURSE WORK, THE ARRAY IS SORTED. THE PROGRAM DEMONSTRATES THE PROCESS OF SORTING A WEAK HEAP BY DISPLAYING THE STATUS OF ITEMS AT EACH STEP. THE RESULT WILL BE A SORTED ARRAY.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является изучение сортировки методом слабой кучи и написание программы, которая будет сортировать входной массив. Для этого потребовалось изучить её структуру, алгоритм построения, алгоритм сортировки с помощью неё, а также придумать визуализацию работы алгоритма. Результатом является программа, которая считывает и сортирует исходный целочисленный массив, визуализируя работу алгоритма.

1. ЗАДАНИЕ

1. Сортировка слабой кучей. Демонстрация.

ВАРИАНТ №31.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

Класс слабой кучи WeakHeap

Для работы со слабой кучей был создан класс слабой кучи WeakHeap. Публичными полями класса являются:

vector <int> wheap — хранятся элементы введённого массива, это вектор, встроенная возможность языка программирования, хранится в библиотеке <vector>

int size_array — хранитсяколичество элементов wheap,

int bit_array* — массив для хранения информации об обмене поддеревьями слабой кучи,

int size_array — размер массива бит.

Для класса реализованы публичные методы для работы со слабой кучей.

Описание метода *void DisplayArray()*

Метод выводит элементы массива, который хранится в поле класса *vector <int> wheap.*

Описание метода *void DisplayHeap()*

Записываем элементы в кучув порядке, удобном для вывода на экран. Затем вычисляем глубину кучи при помощи функции log(2, size). Затем проходимся по элементам, выводим их в наглядном виде, в виде бинарного дерева. Счётчик k нужен для случая, когда количество элементов в слабой куче массива

не будет равно степени двойки, без него вместо отсутствующих элементов на последнемуровне выведутся нули.

Описание метода *void WeakHeapMerge(int *bit_array, inti, int j*)

Если суперродитель меньше потомка, то для потомка переопределяем, порядок его потомков, затем меняем значения суперродителя и потомка при помощи swap(), выводим слабую кучу на экран для демонстрации, какие элементы могли поменяться.

Описание *void WeakHeapSort()*

Метод, в котором происходит сортировка.

Описание метода *WeakHeap* InputHeap()*

Данный метод создаёт новый объект слабо кучи при вводе с консоли.

Описание деструктора ~*WeakHeap()*

Деструктор класса, очищает вектор *wheap*, в котором хранятся элементы.

Описание *void weakHeapSort()*

Метод, в котором происходит сортировка.

Описание функции *double log(int a, int b)*

Функция, которая считает логарифм по основанию a от b.

Описание алгоритма сортировки

Сложность алгоритма по времени — O(n*log n).

Вначале формируем из массива слабую кучу: перебираем элементы массива слева-направо, для текущего элемента поднимаемся вверх по родительской ветке до ближайшего «правого» родителя, сравниваем текущий элемент и ближайшего

правого родителя, если ближайшийправый родитель меньше текущего элемента, то: меняем местами поддеревья с потомками для узла, в котором находится текущий элемент, меняем значениями ближайший «правый» родитель и узел с текущим элементом.

Затем из корня кучи текущий максимальный элемент перемещаем в конец не отсортированной части массива, после чего восстанавливаем слабую кучу: в корне кучи находится текущий максимальный элемент для не отсортированной части массива, меняем местами максимум из корня кучи и последний элемент в не отсортированной части массива. Последний элемент с максимумом перестаётбыть узлом слабой кучи. После этого обмена дерево перестало быть слабой кучей, так как в корне оказался не максимальный элемент. Поэтому делаем просейку: опускаемся из корня кучи по левым потомкам как можно ниже. Поднимаемся по левым потомкам обратно к корню кучи, сравнивая каждый левый потомок с корнем. Если элемент в корне меньше, чем очередной левый потомок, то: меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Меняем значениями корень кучи и узел с текущим левым потомком. В корне слабой кучи снова находится максимальный элемент для оставшейся не отсортированной части массива. Затем снова из корня кучи текущий максимальный элемент перемещаем в конец не отсортированной части массива, восстанавливааем слабую кучу, повторяем процесс, пока не будут отсортированы все элементы.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была изучена сортировка методом слабой кучи. Была изучена структура слабой кучи, а также алгоритм её построения. Реализован алгоритм сортировки с помощью слабой кучи, а также визуализирована его работа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Habr. URL: https://habr.com/en/company/edison/blog/499786/

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

Название файла main.cpp

Номер теста	Входные данные	Выходные данные
1	4 5 3 9 3	Программа для визуализации сортировки слабой кучей. На примере данной программы, можно увидеть, как происходит сортировка слабой кучей. Введите количество элементов массива: 4 Через пробел введите элементы массива: 5 3 9 3 Построение первоначальной слабой кучи 5 3 9 3 _____ _____ _____ _____ 5 3 9 3 _____ _____

		СуперРодитель 5 меньше потомка 9, меняем их местами 9 3 5 3 _____ _____ 9 3 5 3 _____ _____ _____ _____ Слабая куча построена!!! 9 3 5 3 _____ _____ Переносим максимум из корня, применяя слабую просейку 9 3 5 3 _____ _____ Переместили корень 9 и элемент из конца неотсортированной части 3 3 3

		5 9 _____ _____ 3 3 5 9 _____ _____ СуперРодитель 3 меньше потомка 5, меняем их местами 5 3 3 9 _____ _____ 5 3 3 9 _____ _____ _____ _____ Переносим максимум из корня, применяя слабую просейку 5 3 3 9 _____ _____ Переместили корень 5 и элемент из конца неотсортированной части 3

		3 3 5 9 _____ _____ 3 3 5 9 _____ _____ _____ _____ 3 3 5 9 _____ _____ Меняем местами корень 3 и следующий за ним элемент 3 3 3 5 9 _____ _____ В результате сортировки массив примет вид: Итоговый массив: 3 3 5 9

2	2 57 40	Программа для визуализации сортировки слабой кучей. На примере данной программы, можно увидеть, как происходит сортировка слабой кучей. Введите количество элементов массива: 2 Через пробел введите элементы массива: 57 40 Построение первоначальной слабой кучи 57 40 _____ _____ _____ _____ Слабая куча построена!!! 57 40 _____ _____ 57 40 _____ _____ Меняем местами корень 57 и следующий за ним элемент 40 40 57

		_____ _____ В результате сортировки массив примет вид: Итоговый массив: 40 57
3	1 88	Программа для визуализации сортировки слабой кучей. На примере данной программы, можно увидеть, как происходит сортировка слабой кучей. Введите количество элементов массива: 1 Через пробел введите элементы массива: 88 В результате сортировки массив примет вид: Итоговый массив: 88

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ.

Название файла main.cpp

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include <cctype>
#include <iterator>
#include <fstream>
#include <cstring>
#include <cmath>
#include <limits>
#include <cstring>


using namespace std;

class WeakHeap {

public:
    vector <int> wheap;
    int size_of_heap = 0;
    int* bit_array = NULL;
    int size_array;
    WeakHeap* InputHeap();
    void DisplayArray();
    void DisplayHeap();
    void WeakHeapMerge(int* bit_array, int i, int j);
    void WeakHeapSort();

    ~WeakHeap() {
        if (!wheap.empty())
            wheap.clear();
    }
```

```
};

double logarifm(int a, int b) //вычисление логарифма b по
сонованию a
{
    return log(b) / log(a);
}

void WeakHeap::DisplayHeap() {

    vector <int> result_heap;
    result_heap.push_back(wheap[0]);
    result_heap.push_back(wheap[1]);
    for (int i = 0; i < size_array; i++) {//записываем элементы в
кучу в порядке, удобном для вывода на экран
        result_heap.push_back(wheap[2 * i + bit_array[i]]);
        result_heap.push_back(wheap[2 * i + 1 - bit_array[i]]);

    }
    cout << wheap[0];
    cout << "\n";

    int depth = (int)logarifm(2, size_of_heap); //вычисляем
глубину дерева
    if (depth == 3)
        depth++;
    int k = 0;
    double idt = depth * 2;
    for (int i = 0; i < depth; i++) {

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

```
        for (int j = 0; j < pow(2, i); j++) {
            cout << wheap[k+1]<< "  ";
            for (int it = 0; it < idt * 4 - 1; it++)
                cout << " ";
            k++;
        }
        cout << "\n";
    }

    for (int i = 0; i < 100; i++)
        cout << "_";
    cout << "\n";
}

void WeakHeap::WeakHeapMerge(int* bit_array, int i, int j) {


    if (wheap[i] < wheap[j]) { //Для потомка переопределяем,
порядок его потомков, кто левых, а кто правый
        bit_array[j>>3] ^= 1 << ((j) & 7);
        this->DisplayHeap();
        cout << "СуперРодитель " << wheap[i] << " меньше потомка "
<< wheap[j] << ", меняем их местами\n";
        swap(wheap[i], wheap[j]);//Меняем значения "суперродителя"
и потомка
        this->DisplayHeap();

    }
    else {
        this->DisplayHeap();
        for (int i = 0; i < 100; i++)
            cout << "_";
        cout << "\n";
    }
}
```

```
void WeakHeap::WeakHeapSort() {
    int n = size_of_heap;
    int lef;
    int per;
    if (n > 1){
    int i;
    int j;
    int x;
    int y;
    int GreatParent;
    size_array = (n + 7) / 8;
    bit_array = new int[size_array]; // массив для обозначения
левого и правого потомков элемента
    for (i = 0; i < n / 8; ++i)
            bit_array[i] = 0;
    cout << "Построение первоначальной слабой кучи\n";
            for (i = n - 1; i > 0; --i) { //Построение
первоначальной слабой кучи
            j = i;
            //Поднимаемся на сколько возможно вверх,
            lef = (bit_array[(j >> 1) >> 3] >> ((j >> 1) & 7)) &
1;  //если в качестве левого потомка родителя
            while ((j & 1) == lef) {
                j = j >> 1;
                lef = (bit_array[(j >> 1) >> 3] >> ((j >> 1) & 7))
& 1;
            }
            GreatParent = j >> 1;//И ещё на один уровень вверх как
правый потомок родителя
            WeakHeapMerge(bit_array, GreatParent, i); //Слияние
начального элемента, с которого начали восхождение до
СуперРодителя
        }
```

```
        cout << "Слабая куча построена!!!\n"; //переносим максимум
из корня в конец, потом слабая просейка
        this->DisplayHeap();

        for (i = n - 1; i >= 2; --i) {
            cout << "Переносим максимум из корня, применяя слабую
просейку\n";
            this->DisplayHeap();
            cout << "Переместили корень " << wheap[0] << " и
элемент из конца неотсортированной части " << wheap[i] << "\n";
            swap(wheap[0], wheap[i]);
            this->DisplayHeap();
            x = 1;
            lef = (bit_array[(j >> 1) >> 3] >> ((j >> 1) & 7)) &
1; //Опускаемся жадно вниз по левым веткам
            while ((y = 2 * x + lef) < i) {
                x = y;
                lef = (bit_array[(j >> 1) >> 3] >> ((j >> 1) & 7))
& 1;
            }

            while (x > 0) {   //Поднимаемся полевой ветке обратно
до самого вверха
                WeakHeapMerge(bit_array, 0, x);
                x >>= 1;
            }
        }
        this->DisplayHeap();
        cout << "Меняем местами корень " << wheap[0] << " и
следующий за ним элемент " << wheap[1] << "\n";
        swap(wheap[0], wheap[1]);
        this->DisplayHeap();
        delete[] bit_array;
    }
}
```

```
void WeakHeap::DisplayArray()
{
    for (int i = 0; i < size_of_heap; i++)
        cout << wheap[i] << " ";
    cout << "\n\n\n";
}

WeakHeap* WeakHeap::InputHeap() {
    int count;
    cin >> count;
    int elem;
    WeakHeap* wh = new WeakHeap();
    cout << "Через пробел введите элементы массива:\n";
    while (wh->size_of_heap != count) {
        cin >> elem;
        wh->wheap.push_back(elem);
        wh->size_of_heap++;
    }
    return wh;
}

int main()
{
    setlocale(LC_ALL, "rus");
    cout << "Программа для визуализации сортировки слабой
кучей.\n\n";
    cout << "На примере данной программы, можно увидеть, как
происходит сортировка слабой кучей.\n\n";
    WeakHeap* wh = NULL;
    cout << "Введите количество элементов массива:\n";
     wh = wh->InputHeap(); //вводим элементы в кучу
     wh->WeakHeapSort();
     cout << "\nВ результате сортировки массив примет вид:\n";
     cout << "Итоговый массив: ";
     wh->DisplayArray();
```

```
    delete wh;
    return 0;
}
```