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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студент гр. 9304 Шуняев А.В.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с пирамидальной сортировкой, реализовать её.

Задание.

Реализовать пирамидальную сортировку.

Описание алгоритма работы программы.

На вход программы подается строка чисел и указание сортировать по убыванию или по возрастанию. Алгоритм строить из полеченных данных либо max-heap, либо min-heap. В max-heap значение в родительском узле больше, чем значения в дочерних узах, в min-heap наоборот. Построение осуществляется путем сравнения родительского корня с дочерними. В случае если дочерний корень не удовлетворяет условию кучи они меняются местами с родительским корнем. После построения кучи первый и последний элементы меняются местами, затем уменьшается размер кучи, и куча строиться заново. Данные действия повторяются пока размер кучи не стане меньше 0.

Формат входных и выходных данных

С консоли считывается строка чисел, разделенных пробелом.

Описание основных структур данных и функций (кратко).

- Класс Data – хранит данные.
- Класс HeapSort – реализует сортировку.
- Метод CreateMaxHeap – создает Max-Heap.
- Метод CreateMinHeap – создает Min-Heap.
- Метод PrintData – выводит данные в консоль.

Тестирование.

```
Enter data:
1 8 2 3 9 4 5 0 7
Enter sort mode (min, max):
max

Making Max-Heap...1 8 2 7 9 4 5 0 3
1 8 5 7 9 4 2 0 3
1 9 5 7 8 4 2 0 3
9 8 5 7 1 4 2 0 3


Start extraction of elements from the heap...

Swap first and last elements - 9 3
Making Max-Heap from a reduced array...
8 7 5 3 1 4 2 0 9

Swap first and last elements - 8 0
Making Max-Heap from a reduced array...
7 3 5 0 1 4 2 8 9

Swap first and last elements - 7 2
Making Max-Heap from a reduced array...
5 3 4 0 1 2 7 8 9

Swap first and last elements - 5 2
Making Max-Heap from a reduced array...
4 3 2 0 1 5 7 8 9

Swap first and last elements - 4 1
Making Max-Heap from a reduced array...
3 1 2 0 4 5 7 8 9

Swap first and last elements - 3 0
Making Max-Heap from a reduced array...
2 1 0 3 4 5 7 8 9

Swap first and last elements - 2 0
Making Max-Heap from a reduced array...
1 0 2 3 4 5 7 8 9

Swap first and last elements - 1 0
Making Max-Heap from a reduced array...
0 1 2 3 4 5 7 8 9

Swap first and last elements - 0 0
Making Max-Heap from a reduced array...
0 1 2 3 4 5 7 8 9

Test result
HeapSort result:  0 1 2 3 4 5 7 8 9
std::sort result: 0 1 2 3 4 5 7 8 9

Do you want to continue (yes, no):
```

Выводы.

Было изучена пирамидальная сортировка. Была реализована данная сортировка на языке программирования C++.

В ходе работы были отмечены следующие моменты.
Плюсы пирамидальной сортировки:

- Сортировка имеет оценку худшего случая O(n * log(n));
- Сортировка не требует дополнительной памяти при выполнении.

Минусы:

- Алгоритм неустойчив;
- На почти отсортированных массивах работает столь же долго, как и на хаотических данных;
- Не работает на структурах данных с последовательным доступом к памяти.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ.

Файл mail.cpp:

```
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#include <iostream>
#include <algorithm>
#include "HeapSort.h"
#include "Data.h"


bool descending_comp(int a, int b) {
   return a > b;
}

// Управляющая программа
int main()
{
   bool exit = false;
   std::string str;

   while (!exit) {
      std::cout << "Enter data:\n";
      Data data;

      std::cout << "Enter sort mode (min, max):\n";
      std::getline(std::cin, str);

      if (str == "min") {
         HeapSort hsort(data, SortMode::MinHeap);
      }
      else if (str == "max") {
         HeapSort hsort(data, SortMode::MaxHeap);
      }

      std::cout << "\n\nTest result\n";
      std::cout << "HeapSort result:  ";
      data.PrintData(data.array_);

      if (str == "min") {
         std::sort(data.test_array.begin(), data.test_array.end(), descending_comp);
      }
      else if (str == "max") {
         std::sort(data.test_array.begin(), data.test_array.end());
```

```
        }

        std::cout << "\nstd::sort result: ";
        data.PrintData(data.test_array);

        std::cout << "\n\nDo you want to continue (yes, no):\n";
        std::getline(std::cin, str);

        if (str == "no") {
            exit = true;
        }

        std::cout << "\n\n";
    }

    return 0;
    _CrtDumpMemoryLeaks();
}
```

t

Файл Data.h:

```
#pragma once
#include <vector>
#include <fstream>
#include <iostream>
#include <sstream>


class Data
{
public:
        std::vector<int> array_;
        std::vector<int> test_array;

        Data();
        ~Data();

        void PrintData(std::vector<int>& data);
};
```

Файл Data.cpp:

```
#include "Data.h"

Data::Data()
{
```

```
	int digit;
	std::vector<int> temp;
	std::string str;

	std::getline(std::cin, str);
	std::istringstream iss(str);

	while (iss >> digit) {
		this->array_.push_back(digit);
		this->test_array.push_back(digit);
	}
}


Data::~Data()
{

}

void Data::PrintData(std::vector<int>& data)
{
	for (int i = 0; i < data.size(); i++) {
		std::cout << data[i] << ' ';
	}
}
```

Файл HeapSort.cpp:

```
#include "HeapSort.h"

#include "Data.h"


HeapSort::HeapSort(Data& data, SortMode mode)

{

   this->mode_ = mode;


   if (mode_ == SortMode::MaxHeap) {

      std::cout << "\nMaking Max-Heap...";

      for (int i = (data.array_.size() / 2 - 1); i >= 0; i--) {

         CreateMaxHeap(data.array_, data.array_.size(), i);

         data.PrintData(data.array_);

         std::cout << std::endl;

      }


      std::cout << "\n\nStart extraction of elements from the heap...";

      for (int i = (data.array_.size() - 1); i >= 0; i--)

      {

         std::cout << "\n\nSwap first and last elements - " << data.array_[0] << ' ' << data.array_[i] << '\n';

         std::swap(data.array_[0], data.array_[i]);


         std::cout << "Making Max-Heap from a reduced array...\n";

         CreateMaxHeap(data.array_, i, 0);


         data.PrintData(data.array_);

      }
```

```
    }
    else if (mode_ == SortMode::MinHeap) {
        std::cout << "\nMaking Min-Heap...";
        for (int i = (data.array_.size() / 2 - 1); i >= 0; i--) {
            CreateMinHeap(data.array_, data.array_.size(), i);
            data.PrintData(data.array_);
            std::cout << std::endl;
        }

        std::cout << "\n\nStart extraction of elements from the heap...";
        for (int i = (data.array_.size() - 1); i >= 0; i--)
        {
            std::cout << "\n\nSwap first and last elements - " << data.array_[0] << ' ' << data.array_[i] << '\n';
            std::swap(data.array_[0], data.array_[i]);

            std::cout << "Making Min-Heap from a reduced array...\n";
            CreateMinHeap(data.array_, i, 0);

            data.PrintData(data.array_);
        }
    }
}

HeapSort::~HeapSort()
{
```

```
}

void HeapSort::CreateMaxHeap(std::vector<int>& data, int heap_size, int root)
{
    int largest = root;
    int left = 2 * root + 1; // левый = 2*i + 1
    int right = 2 * root + 2; // правый = 2*i + 2

    if (left < heap_size && data[left] > data[largest]) {
        largest = left;
    }

    if (right < heap_size && data[right] > data[largest]) {
        largest = right;
    }

    if (largest != root)
    {
        std::swap(data[root], data[largest]);

        CreateMaxHeap(data, heap_size, largest);
    }
}

void HeapSort::CreateMinHeap(std::vector<int>& data, int heap_size, int root)
{
```

```
    int smallest = root;
    int left = 2 * root + 1; // левый = 2*i + 1
    int right = 2 * root + 2; // правый = 2*i + 2

    if (left < heap_size && data[left] < data[smallest]) {
        smallest = left;
    }

    if (right < heap_size && data[right] < data[smallest]) {
        smallest = right;
    }

    if (smallest != root)
    {
        std::swap(data[root], data[smallest]);

        CreateMinHeap(data, heap_size, smallest);
    }
}
```

Файл HeapSort.h:

```
#pragma once
#include <vector>
#include <iostream>

class Data;
```

```
enum class SortMode {
	MaxHeap,
	MinHeap,
};

class HeapSort
{
public:
	SortMode mode_;

	HeapSort(Data& data, SortMode mode);
	~HeapSort();

	void CreateMaxHeap(std::vector<int>& data, int heap_size, int root);
	void CreateMinHeap(std::vector<int>& data, int heap_size, int root);
};
```