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 системы данных»

ТЕМА: N-АРНАЯ КУЧА

Студент гр. 9381 _____ Любимов В.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить понятие n-арной кучи и реализовать её.

Задание.

Вариант 29

Дан массив чисел и число n (n=1, 2, 3, …). Предполагая, что массив является n-арной кучей:

- Вывести его в виде n-арной кучи.

- Получить путь от корня до листа такой, что при каждом шаге вниз выбирается наибольший сын.

Основные теоретические положения.

n-арной куча – структура данных, представляемая в виде n-арного дерева и имеющая следующие свойства:

1. Значение каждого узла не меньше любого его потомка.
2. Глубина всех листьев отличается не более, чем на 1 слой.
3. Слои заполняются слева направо и без «дырок»

Из этих свойств следует, что реализацией кучи является массив A, в котором в A[0] хранится корень кучи, а сыновьями элемента A[i] являются элементы A[i*n+1], A[i*n+2], … , A[i*n+n]. Очевидно, что при n = 1 куча превращается в массив, отсортированный по убыванию. n – натуральное число.

Так как потомками A[i] являются элементы A[i*n+1], A[i*n+2], … , A[i*n+n], то если обозначить индекс j-го потомка, как x, то x = n*i+j. Из этого получаем, что i = x-j/n. Это выражение принимает нецелое значение при любом j отличном от нуля и меньшим n. В этом случае его

необходимо округлить вниз до ближайшего целого числа, иначе получим, что x = x+α, где α – добавка при округлении. Такое значение x невозможно. Значит нужно округлять вниз. Тогда можно заменить j на 1 для любого допустимого j. Итого индекс предка любого потомка можно выразить, как i = floor((x-1)/n), где x – индекс рассматриваемого потомка, i – индекс предка, floor(число) – округление вниз до ближайшего целого числа.

Описание алгоритма.

Во входном файле содержится число n и отделённый от него пробелом массив целых чисел, представляющий кучу. Выводит эту кучу, как дерево. После этого проходит от корня до листа, выбирает наибольших потомков, последовательно перебрав всех для этого, и сохраняет их значения в специальный массив размером в высоту дерева. Выводит индексы элементов, составляющих найденный путь.

Чтобы при выводе кучи, как дерева, гарантировать корректность вывода, то есть все сыновья умещаются и не накладываются друг на друга, необходимо рассчитать расстояние между началом строки и первым потомком в этом слое и между двумя соседними потомками. Расстояние будет измеряться в символах. Во-первых, узел должен находится ровно посередине между крайними потомками. Во-вторых, на любом уровне между потомками должен быть хотя бы один символ. В-третьих, первые два пункта должны выполняться при любой высоте дерева и размерности кучи. Итак, каждый уровень дерева будет занимать одинаковое количество символов, и каждый узел – тоже. Для узла количество символов, отображающих его значение равно 4 (если вывод значения узла занимает менее 4 символов, то лишние символы заполняются пробелами). Для n-арного дерева количество узлов на i-ом ($0 \leq i <$ количество уровней) уровне равно n^i. Для вывода каждого узла потребуется 2*step + 4, где step – размер

отступа от поля вывода значения узла, то есть между двумя узлами 2*step пробелов. Значит ширина уровня равна S = (2*step + 4)*n^i и такова для любого i. Найдём минимальную $step_i$, то есть step для i-го уровня. $step_i = \frac{S}{2 \cdot n^i} - 2$. Так необходимо минимальное S и step больший нуля, то пускай на самом нижнем уровне, то есть tree_height – 1, step равен 1. Из этого следует, что $S = 3 \cdot 2 \cdot n^{tree\_height-1}$. Тогда на i-ом (0 ≤ i < количество уровней) уровне $step_i = \frac{3 \cdot 2 \cdot n^{tree\_height-1}}{2 \cdot n^i} - 2$. Также из этой формулы видно, что для ширина вывода бинарной кучи из пяти уровней равна 96 символов, для 3-арной кучи из 4 уровней – 162 символа, а для 8-арной кучи из 3 уровней - 384

Описание реализованных классов

Класс class Dheap реализует n-арную кучу и методы её обработки.

Объекты класса имеют следующие приватные поля:

- m_arr - массив, содержащей кучу
- m_root - корень кучи
- m_size - размер кучи
- m_arr_size - размер массива
- m_mem_size - размер массива в памяти
- m_d - порядок кучи, то есть n

Были реализованы следующие методы:

- Dheap(int* arr = nullptr, int root = 0, int size = 0, int d = 2) - конструктор, из полученного массива arr копирует элементы в массив m_arr, предварительно выделив под него необходимый объём памяти.

- bool readHeapFromFile(ifstream &fin) - ссылку на файл fin, содержащий входные данные. Записывает первое число из файла в поле m_d, а остальные числа - в полученный массив m_arr. Если заполнен весь массив, то увеличивает количество выделенной под него памяти. Если размер считанного массива нуль или m_d меньше единицы, то возвращает false. Иначе возвращает true.
- int calcHeight() - вычисляет и возвращает высоту дерева спускаясь по левым сыновьям пока они существуют.
- int findMaxLeaf(int root) - находит индекс максимального потомка полученного узла root. Последовательного перебирает значения всех потомков узла root, сравнивает их с текущим максимальным значением и запоминает индекс потомка с максимальным значением, который и возвращает.
- int findMax(int root) - находит индекс максимального потомка полученного узла root. Последовательного перебирает значения всех потомков узла root и самого узла, сравнивает их с текущим максимальным значением и запоминает индекс элемента с максимальным значением, который и возвращает.
- void printNode(int node_value, int step, bool is_col)- выводит узел с полученным значением node_value в консоль на определённом месте задающимся отступом step. В зависимости от значения is_col выделяет узел цветом. Сначала выводит step пробелов, включает выделение цветом при помощи управляющей эскейп-последовательности, если is_col == true. Выводит node_value в поле шириной четыре символа и выравниванием по левому краю. В завершение выводит ещё step пробелов.
- void printAsArr(bool is_col_first) - выводит кучу как массив. Ту часть массива, которая является кучей выделяет зелёным,

отсортированную часть – белым, а первый элемент отсортированной части – голубым, если is_col_first == true.

- void printHeap(int* color_nodes, int col_size) - выводит кучу, как дерево. Если размер кучи равен нуль, то выводит сообщение о том, что куча пуста. Принимает на вход массив color_nodes, содержащий индексы узлов, которые необходимо раскрасить, и их количество col_size. Высчитывает высоту кучи методом calcHeight(). Для каждого узла высчитывает отступ по описанной выше формуле. После чего выводит этот узел при помощи метода printNode(). Если индекс выводимого узла совпадает с текущим элементом в массиве узлов для раскраски, то раскрашивает его и переходит к следующему элементу в этом массиве. Если индекс текущего элемента равен максимальному индексу, допустимого на этом уровне, то увеличивает уровень на единицу и переносит строку.
- int goToMaxLeaf(int* &way) - спускается от корня до листа, переходя по максимальным потомкам для каждого узла на пути. Находит максимального потомка для текущего узла методом findMaxLeaf() и переходит в него. Записывает индексы получаемого пути в полученный массив way. Возвращает длину этого пути.

Описание функций.

1. int main() – реализован простейший функционал взаимодействия с пользователем. Пользователю предлагают выбор из нескольких опций: завершить выполнение программы или начать работу. В последнем случае предлагается ввести путь до файла с входными данными. Создаётся объект класса Dheap и методом

readHeapFromFile() создаётся куча. Эта куча выводится на экран в виде дерева. После выполняется поиск пути до листа, переходя по максимальным потомкам. Этот путь выводится в наглядном виде и в виде последовательности индексов узлов. Выделенная память освобождается. После чего пользователю снова предлагается выбор из двух выше указанных опций.

Тестирование.

Тестирование программы представлено в таблице 1.

Таблица 1.

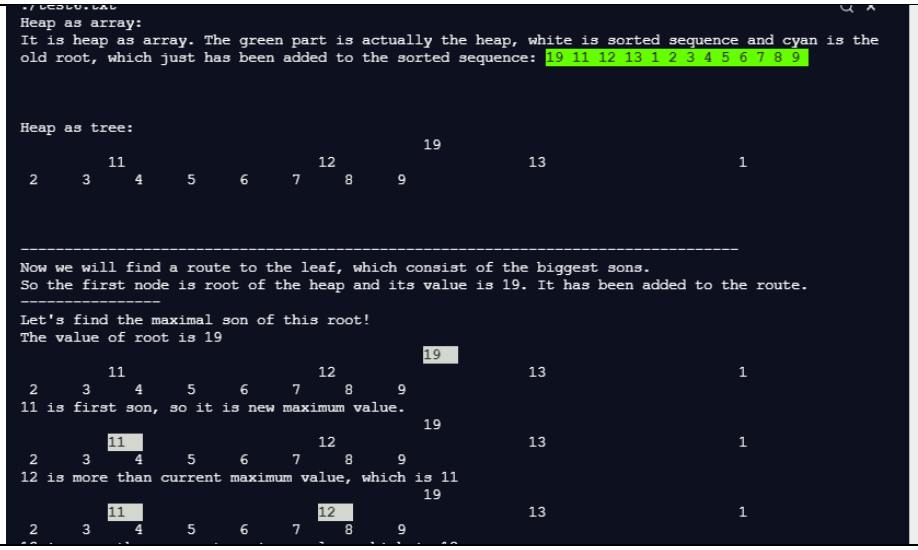
№	Входные данные:	Выходные данные:
1	2 9 8 5 -1 -9 1 2	Heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 9 8 5 -1 -9 1 2 Heap as tree: 9 8 5 -1 -9 1 2 ---------- Now we will find a route to the leaf, which consist of the biggest sons. So the first node is root of the heap and its value is 9. It has been added to the route. ---------- Let's find the maximal son of this root! The value of root is 9 9 8 5 -1 -9 1 2

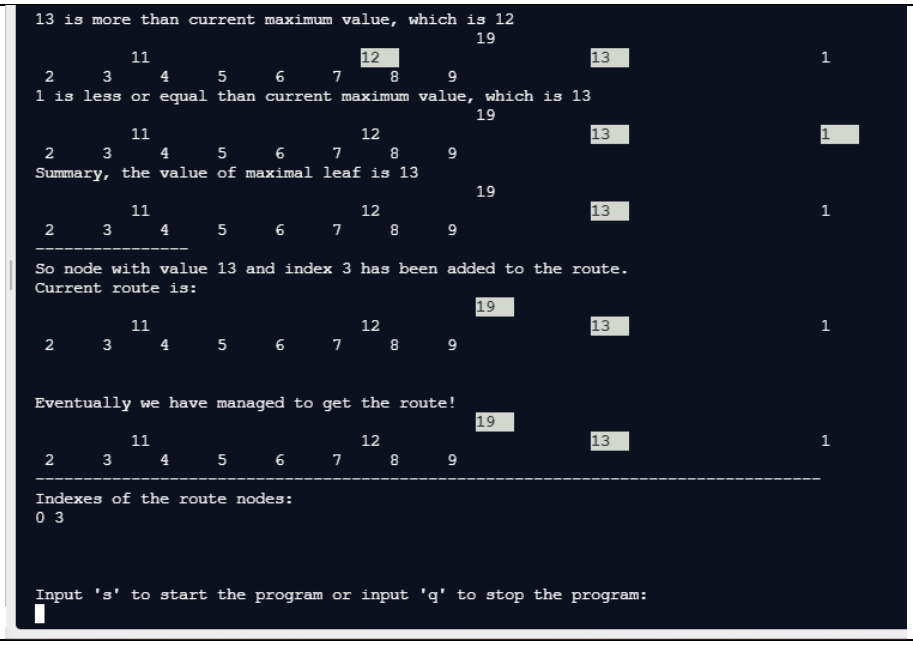
```
8 is first son, so it is new maximum value.
          9
    8           5
 -1    -9    1     2

5 is less or equal than current maximum value, which
is 8
          9
    8           5
 -1    -9    1     2

Summary, the value of maximal leaf is 8
          9
    8           5
 -1    -9    1     2

----------------
So node with value 8 and index 1 has been added to the
route.
Current route is:
          9
    8           5
 -1    -9    1     2



----------------
Let's find the maximal son of this root!
The value of root is 8
          9
    8           5
 -1    -9    1     2

-1 is first son, so it is new maximum value.
          9
    8           5
 -1    -9    1     2

-9 is less or equal than current maximum value, which
is -1
          9
    8           5
 -1    -9    1     2

Summary, the value of maximal leaf is -1
          9
    8           5
 -1    -9    1     2
```

		---------- So node with value -1 and index 3 has been added to the route. Current route is: 9 8 5 -1 -9 1 2 Eventually we have managed to get the route! 9 8 5 -1 -9 1 2 ---------- Indexes of the route nodes: 0 1 3
2	3 9 8 5 1 4 1 2	Heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 9 8 5 1 4 1 2 Heap as tree: 9 8 5 1 4 1 2 ---------- Now we will find a route to the leaf, which consist of the biggest sons. So the first node is root of the heap and its value is 9. It has been added to the route. ---------- Let's find the maximal son of this root! The value of root is 9 9 8 5 1 4 1 2 8 is first son, so it is new maximum value. 9

```
       8                   5                   1
 4     1     2
5 is less or equal than current maximum value, which
is 8
                           9
       8                   5                   1
 4     1     2
1 is less or equal than current maximum value, which
is 8
                           9
       8                   5                   1
 4     1     2
Summary, the value of maximal leaf is 8
                           9
       8                   5                   1
 4     1     2
----------------
So node with value 8 and index 1 has been added to the
route.
Current route is:
                           9
       8                   5                   1
 4     1     2


----------------
Let's find the maximal son of this root!
The value of root is 8
                           9
       8                   5                   1
 4     1     2
4 is first son, so it is new maximum value.
                           9
       8                   5                   1
 4     1     2
1 is less or equal than current maximum value, which
is 4
                           9
       8                   5                   1
 4     1     2
2 is less or equal than current maximum value, which
is 4
                           9
       8                   5                   1
 4     1     2
Summary, the value of maximal leaf is 4
                           9
       8                   5                   1
 4     1     2
```

		---------- So node with value 4 and index 4 has been added to the route. Current route is: 9 8 5 1 4 1 2 Eventually we have managed to get the route! 9 8 5 1 4 1 2 ---------- Indexes of the route nodes: 0 1 4
3	4 123 34 56 90 78	Heap as tree: 123 34 56 90 78 Route: 123 34 56 90 78 Indexes of the route nodes: 0 3
4	1 9 8 7 6 5 4 3 2 1 0	Heap as tree: 9 8 7 6 5 4 3 2 1 0 Route 9 8 7 6 5 4 3

		2 1 0 Indexes of the route nodes: 0 1 2 3 4 5 6 7 8 9
5	3 19 11 12 13 1 2 3 4 5 6 7 8 9	Heap as tree: 19 11 12 13 1 2 3 4 5 6 7 8 9 Route: 19 11 12 13 1 2 3 4 5 6 7 8 9 Indexes of the route nodes: 0 3 12
6	4 19 11 12 13 1 2 3 4 5 6 7 8 9	

		
7	2 89 81 42 43 61 2 37	Heap as tree: 89 81 42 43 61 2 37 Route: 89 81 42 43 61 2 37 Indexes of the route nodes: 0 1 4

Файлы с этими входными данными лежат в папке test.

Выводы.

Было изучено понятие n-арной кучи. Была разработана программа выводящая кучу в наглядном виде и находящая путь от корня до листа, проходящий через наибольших потомков.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

using namespace std;

class Dheap{
    private:
        int* m_arr = nullptr;//массив хранящий кучу
        int m_root = 0;//корень кучи
        int m_size = 0;//размер кучи
        int m_arr_size = 0;//размер массива
        int m_mem_size = 0;//размер кучи в памяти
        int m_d = 2;//порядок кучи; по умолчанию куча бинарная

    public:
        Dheap(int* arr = nullptr, int root = 0, int size = 0, int d = 2):
m_root(root), m_size(size), m_d(d){//конструктор копирует полученный массив в
массив вершин; пока в это ещё не d-арное дерево
            if(arr){
                m_arr = new int[size];
                m_arr_size = size;
                for(int i = 0; i < size; i++){
                    m_arr[i] = arr[i];
                }
            }
        }

        bool readHeapFromFile(ifstream &fin){//метод считывающий массив из
входного файла
            m_size = 0;
            m_mem_size = 0;
            m_root = 0;
            m_d = 0;
            fin >> m_d;

            if(m_d <= 0){
                cout << "Non natural value of amount of node sons!\n";
                return false;
            }
```

```
        while(1){
            if(m_size == m_mem_size){
                m_mem_size += 10;
                int* new_arr = new int[m_mem_size];
                for(int i = 0; i < m_size; i++){
                    new_arr[i] = m_arr[i];
                }
                delete[] m_arr;
                m_arr = new_arr;
            }

            if(fin.eof()){
                break;
            }

            fin >> m_arr[m_size];
            m_size += 1;
        }

        m_arr_size = m_size;
        if(m_size == 0){
            cout << "Error! Empty heap has inputed!\n";
            return false;
        }
        return true;
    }

    int calcHeight(){//высчитывает количество уровней в дереве
        int i = m_root;
        int height = 0;
        while(i < m_size){
            height += 1;
            i = i*m_d + 1;
        }
        return height;
    }

    int findMaxLeaf(int root){//поиск индекса максимального элемента среди
потомков вершины
        cout << "----------------\n";
        cout << "Let's find the maximal son of this root!\n";
        int max = -1;
        int j = 0;
        int nodes[m_d+1];
        cout << "The value of root is " << m_arr[root] << "\n";
        nodes[j] = root;
        j += 1;
```

```
            printHeap(nodes, j);
            for(int i = root*m_d+1; i <= root*m_d + m_d && i < m_size; i++){
                if(max == -1){
                    max = i;
                    cout << m_arr[i] << " is first son, so it is new maximum
value.\n";
                    nodes[0] = i;
                    //j += 1;
                    printHeap(nodes, j);
                    continue;
                }

                if(m_arr[i] > m_arr[max]){
                    cout << m_arr[i] << " is more than current maximum value,
which is " << m_arr[max] << '\n';
                    nodes[0] = max;
                    max = i;
                }
                else{
                    cout << m_arr[i] << " is less or equal than current
maximum value, which is " << m_arr[max] << '\n';
                    nodes[0] = max;
                }
                nodes[j] = i;
                //j += 1;
                printHeap(nodes, j+1);
            }
            cout << "Summary, the value of maximal leaf is " << m_arr[max] <<
"\n";
            nodes[0] = max;
            printHeap(nodes, 1);
            cout << "----------------\n";
            return max;
        }

        int findMax(int root){//поиск индекса максимального элемента среди
вершины и потомков
            cout << "----------------\n";
            cout << "Let's find the maximal elememt in this root or its
sons!\n";
            int max = root;
            int j = 0;
            int nodes[m_d+1];
            cout << "The value of root is " << m_arr[root] << "\n";
            nodes[j] = root;
            j += 1;
            printHeap(nodes, j);
            for(int i = root*m_d+1; i <= root*m_d + m_d && i < m_size; i++){
```

```
                if(m_arr[i] > m_arr[max]){
                    cout << m_arr[i] << " is more than current maximum value,
which is " << m_arr[max] << '\n';
                    nodes[0] = max;
                    max = i;
                }
                else{
                    cout << m_arr[i] << " is less or equal than current
maximum value, which is " << m_arr[max] << '\n';
                    nodes[0] = max;
                }
                nodes[j] = i;
                //j += 1;
                printHeap(nodes, j+1);
            }
            cout << "Summary, the value of maximal element of root and its
leaf is " << m_arr[max] << "\n";
            nodes[0] = max;
            printHeap(nodes, 1);
            cout << "----------------\n";
            return max;
        }

        void siftUp(int leaf){//просейка снизу-вверх
            if(leaf == m_root || (leaf-1)/m_d < 0){
                return;
            }

            while(leaf != m_root && m_arr[leaf] > m_arr[(leaf-1)/m_d] &&
m_arr[m_root] < m_arr[leaf]){
                int c = m_arr[leaf];
                m_arr[leaf] = m_arr[m_root];
                m_arr[m_root] = c;
                leaf = (leaf-1)/m_d;
                if(leaf < 0){
                    return;
                }
            }
        }

        void printAsArr(bool is_col_first){//выводит кучу как массив
            cout << "It is heap as array. The green part is actually the heap,
white is sorted sequence and cyan is the old root, which just has been added
to the sorted sequence: ";
            for(int i = 0; i < m_arr_size; i++){
                cout << "\033[1;30;42m";
                if(i >= m_size){
                    if(i == m_size && is_col_first){
```

```
                    cout << "\033[1;30;46m";
                }
                else{
                    cout << "\033[1;30;47m";
                }
            }
            cout << m_arr[i] << ' ';
            cout << "\033[0m";
        }
        cout << '\n';
    }

    void printNode(int node_value, int step, bool is_col){//выводит узел в
консоль
        for(int i = 0; i < step; i++){
            cout << ' ';
        }

        if(is_col){
            cout << "\033[1;30;47m";
        }
        cout.setf(ios::left);
        cout.width(4);
        cout << node_value;
        cout.unsetf(ios::left);
        cout << "\033[0m";

        for(int i = 0; i < step; i++){
            cout << ' ';
        }
    }

    void printHeap(int* color_nodes, int col_size){//выводит кучу в
консоль, как дерево
        if(m_size == 0){
            cout << "Empty heap!\n";
            return;
        }
        int lev = 0;
        int height = calcHeight();
        bool is_col = false;
        int j = 0;
        int step = 0;
        for(int i = 0; i < m_size; i++){
            step = int(3*2*int(pow(double(m_d),double(height-
1))))/(2*int(pow(double(m_d),double(lev))))-2);

            is_col = false;
```

```
                if(j < col_size){
                    if(i == color_nodes[j]){
                        is_col = true;
                        j += 1;
                    }
                }

                if(lev == 0){
                    printNode(m_arr[i], step, is_col);
                    lev += 1;
                    cout << '\n';
                    continue;
                }
                printNode(m_arr[i], step, is_col);

                if(i%((int)(double(1.0-pow(double(m_d),
double(lev+1)))/double(1-m_d))-1) == 0 || m_d == 1){
                    lev += 1;
                    cout << '\n';
                }
            }
            cout << "\n";
        }

        int goToMaxLeaf(int* &way){//спускаемся до листа, для каждой вершины
выбирая максимального потомка
            cout << "-------------------------------------------------------
-------------------------\n";
            cout << "Now we will find a route to the leaf, which consist of
the biggest sons.\n";
            int root =  m_root;
            int length = 1;
            int i = 0;
            way[i] = root;
            cout << "So the first node is root of the heap and its value is "
<< m_arr[root] << ". It has been added to the route.\n";
            while(root*m_d+1 < m_size){
                length += 1;
                i += 1;
                root = findMaxLeaf(root);
                way[i] = root;
                cout << "So node with value " << m_arr[root] << " and index "
<< root << " has been added to the route.\n";
                cout << "Current route is:\n";
                printHeap(way, length);
                cout << "\n\n";
            }
```

```
            cout << "Eventually we have managed to get the route!\n";
            printHeap(way, length);
            cout << "-------------------------------------------------------------
-------------------------\n";
            return length;
        }

        int getHeight(){//возвращает высоту дерева
            return calcHeight();
        }

        ~Dheap(){//деструктор; очищает память выделенную под массив-кучу
            delete[] m_arr;
        }
};


int main(){
    ifstream fin;

    char command;
    string fname;
    Dheap* heap = nullptr;
    int* route = nullptr;
    int length = 0;
    bool isD = true;
    while(1){
        cout << "Input 's' to start the program or input 'q' to stop the
program:\n";
            cin >> command;
            switch (command){
                case 'q':
                    cout << "You choose to end the programm!\n";
                    return 0;

                case 's':
                    cout << "Input the path to data file:\n";
                    cin >> fname;
                    fin.open(fname, ifstream::in);
                    if(!fin.is_open()){
                        cout << "Opening file with test data failed! Try
again!\n";
                        break;
                    }

                    heap = new Dheap;
                    isD = heap->readHeapFromFile(fin);
                    fin.close();
```

```
                    if(!isD){
                        cout << "Error in input data";
                        delete heap;
                        break;
                    }

                    cout << "Heap as array:\n";
                    heap->printAsArr(false);
                    cout << "\n\n\n";

                    cout << "Heap as tree:\n";
                    heap->printHeap(nullptr, -1);
                    cout << "\n\n\n";

                    route = new int[heap->calcHeight()];
                    length = 0;
                    length = heap->goToMaxLeaf(route);
                    cout << "Indexes of the route nodes:\n";
                    for(int i = 0; i < length; i++){
                        cout << route[i] << ' ';
                    }
                    cout << "\n\n\n\n";

                    delete[] route;
                    delete heap;
                    break;

                default:
                    cout << "Error command! Try again!\n";
                    break;

            }
    }

    return 0;
}
```