**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 А, в котором в А[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 ≤ i < количество уровней) уровне равно ni. 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)\*ni и такова для любого i. Найдём минимальную stepi, то есть step для i-го уровня. . 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 . Тогда на i-ом (0 ≤ i < количество уровней) уровне . 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. Некоторые выходные данные будут частично представлены в виде скриншотов из-за своей очень большой «ширины».

Таблица 1.

|  |  |  |
| --- | --- | --- |
| № | Входные данные: | Выходные данные: |
| 1 | -8 6 -0.0001 7 -1 2 3 0.7 4 5 6 7 8 -9 0 | -9 -8 -1 -0.0001 0 0.7 2 3 4 5 6 6 7 7 8 |
| 2 | 1 | 1 |
| 3 | 1 2 3 10 7 8 | 1 2 3 7 8 10 |
| 4 | 63 28 483 595 724 949 98 | 28 63 98 483 595 724 949 |
| 5 | 36 8.8 -37163 0.0000067 -0.73 178361 | -37163 -0.73 6.7e-06 8.8 36 178361 |
| 6 | 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 | -9 -8 -7 -6 -5 -4 -3 -2 -1 0 |
| 7 | 1 2 3 4 5 6 7 8 9 0 | 0 1 2 3 4 5 6 7 8 9 |
| 8 | -1 1 -2 2 -3 3 -4.5 4.5 9.1 -9.1 0 | -9.1 -4.5 -3 -2 -1 0 1 2 3 4.5 9.1 |

Файлы с этими входными данными лежат в папке test.

**Выводы.**

Была изучена, проанализирована и реализована сортировка простыми вставками в массиве и списке.

**ПРИЛОЖЕНИЕ А**  
**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файлов: main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

using namespace std;

template <class T>

int getData(T\* &arr, ifstream& fin){

    int size = 0;

    int mem\_size = 0;

    while(1){

        if(size == mem\_size){

            mem\_size += 10;

            T\* new\_arr = new T[mem\_size];

            for(int i = 0; i < size; i++){

                new\_arr[i] = arr[i];

            }

            delete[] arr;

            arr = new\_arr;

        }

        if(fin.eof()){

            return size;

        }

        fin >> arr[size];

        size += 1;

    }

    return size;

}

template <class T>

void sort(T\* &arr, int size, ofstream& fout){

    if(size == 1){

        cout << "Only one elememt in array! Sort doesn't need.\n";

        fout << "Only one elememt in array! Sort doesn't need.\n";

    }

    T cur = 0;

    int j = 0;

    for(int i = 1; i < size; i++){

        cur = arr[i];

        j = i-1;

        cout << "Sorted part of array: ";

        fout << "Sorted part of array: ";

        for(int k = 0; k <= j; k++){

            cout << "\033[32m" << arr[k] << ' ';

        }

        cout << "\033[0mCurent elemrnt is " << "\033[33m" << cur << ' ' << "\033[0mUnsorted part of array: ";

        fout << "Curent elemrnt is " << cur << ' ' << "Unsorted part of array: ";

        for(int k = i+1; k < size; k++){

            cout << "\033[36m" << arr[k] << ' ';

            fout << arr[k] << ' ';

        }

        cout << "\033[0m\nCurrent index is " << i << '\n';

        fout << "\nCurrent index is " << i << '\n';

        while(arr[j] > cur && j >= 0){

            cout << "\033[32m" << arr[j] << "\033[0m at positon " << j << " is more than \033[33m" << cur << "\033[0m\n";

            fout << arr[j] << " at positon " << j << " is more than " << cur << "\n";

            arr[j+1] = arr[j];

            j -= 1;

        }

        if(j >= 0){

            cout << "\033[32m" << arr[j] << "\033[0m at positon " << j << " is less or equal than \033[33m" << cur << "\033[0m. Element insert into position with index " << j+1 << '\n';

            fout << arr[j] << " at positon " << j << " is less or equal than " << cur << ". Element insert into position with index " << j+1 << '\n';

        }

        else{

            cout << "Start of array nave been reached. \033[33m" << cur << "\033[0m is insert into the start of array.\n";

            fout << "Start of array nave been reached. " << cur << " is insert into the start of array.\n";

        }

        arr[j+1] = cur;

        cout << "Current status of array: ";

        fout << "Current status of array: ";

        for(int k = 0; k < size; k++){

            cout << arr[k] << ' ';

            fout << arr[k] << ' ';

        }

        cout << "\n\n";

        fout << "\n\n";

    }

}

template <class T>

class Node{

    private:

        T m\_value;

        int m\_next;

    public:

        Node(T value = 0, int next = -1): m\_value(value), m\_next(next){

        }

        void setValue(T value){

            m\_value = value;

        }

        T getValue(){

            return m\_value;

        }

        void setNext(int next){

            m\_next = next;

        }

        int getNext(){

            return m\_next;

        }

};

template <class T>

class List{

    private:

        Node<T>\* m\_head = nullptr;

        int head = 0;

        int m\_size = 0;

        int m\_memory\_size = 0;

    public:

        List(){

            m\_head = nullptr;

        }

        void addNode(T value){

            if(!m\_head){

                head = 0;

                m\_head = new Node<T>[10];

                m\_head[0].setValue(value);

                m\_size += 1;

                m\_memory\_size = 10;

                return;

            }

            if(m\_size == m\_memory\_size){

                m\_memory\_size += 10;

                Node<T>\* new\_arr = new Node<T>[m\_memory\_size];

                for(int i = 0; i < m\_size; i++){

                    new\_arr[i] = m\_head[i];

                }

                delete[] m\_head;

                m\_head = new\_arr;

            }

            int cur = 0;

            while(m\_head[cur].getNext() != -1){

                cur += 1;

            }

            m\_head[cur].setNext(m\_size);;

            m\_head[m\_size].setValue(value);

            m\_size += 1;

        }

        void makeList(T\* &arr, int size){

            for(int i = 0; i < size; i++){

                addNode(arr[i]);

            }

        }

        void printList(ofstream& fout){

            int cur = head;

            while(cur != -1){

                cout << m\_head[cur].getValue() << ' ';

                fout << m\_head[cur].getValue() << ' ';

                cur = m\_head[cur].getNext();

            }

            cout << '\n';

            fout << '\n';

        }

        void sort(ofstream& fout){

            int precur = head;

            int cur = m\_head[precur].getNext();

            int comp = head;

            int precomp = head;

            while(cur != -1){

                m\_head[precur].setNext(m\_head[cur].getNext());

                m\_head[cur].setNext(-1);

                cout << "Current element is \033[33m" << m\_head[cur].getValue() << "\033[0m\n";

                cout << "This element is going to be compared with elements before its postion in list begining with the head of the list.\n";

                fout << "Current element is " << m\_head[cur].getValue() << '\n';

                fout << "This element is going to be compared with elements before its postion in list begining with the head of the list.\n";

                while(comp != -1 && m\_head[comp].getValue() < m\_head[cur].getValue() && comp != m\_head[precur].getNext()){

                    cout << "\033[32m" << m\_head[comp].getValue() << "\033[0m is less than \033[33m" << m\_head[cur].getValue() << "\033[0m\n";

                    fout << m\_head[comp].getValue() << " is less than " << m\_head[cur].getValue() << '\n';

                    precomp = comp;

                    comp = m\_head[comp].getNext();

                }

                cout << "\033[32m" << m\_head[comp].getValue() << "\033[0m is more or equal than \033[33m" << m\_head[cur].getValue() << "\033[0m. Also its possiblke that all elements before current have been viewed and current is bigger then all of them. ";

                fout << m\_head[comp].getValue() << " is more or equal than " << m\_head[cur].getValue() << ". Also its possiblke that all elements before current have been viewed and current is bigger then all of them. ";

                if(comp == head){

                    cout << "It is the head of th list, so \033[33m" << m\_head[cur].getValue() << "\033[0m is a new head of the list.\n";

                    fout << "It is the head of th list, so " << m\_head[cur].getValue() << " is a new head of the list.\n";

                    m\_head[cur].setNext(comp);

                    head = cur;

                }

                else{

                    cout << "So \033[33m" << m\_head[cur].getValue() << "\033[0m has been placed between \033[32m" << m\_head[precomp].getValue() << "\033[0m and \033[32m" << m\_head[comp].getValue() << "\033[0m\n";

                    fout << "So " << m\_head[cur].getValue() << " has been placed between " << m\_head[precomp].getValue() << " and " << m\_head[comp].getValue() << '\n';

                    m\_head[precomp].setNext(cur);

                    m\_head[cur].setNext(comp);

                }

                if(m\_head[precur].getNext() == cur){

                    precur = cur;

                    cur = m\_head[cur].getNext();

                }

                else{

                    cur = m\_head[precur].getNext();

                }

                comp = head;

                precomp = head;

                cout << "The current state of the list: ";

                fout << "The current state of the list: ";

                printList(fout);

                cout << '\n';

                fout << '\n';

            }

        }

        ~List(){

            delete[] m\_head;

        }

};

template <class T>

void printArr(T\* &arr, ofstream& fout, int size){

    for(int i = 0; i < size; i++){

        cout << arr[i] << ' ';

        fout << arr[i] << ' ';

    }

    cout << '\n';

    fout << '\n';

}

int main(){

    double\* arr = nullptr;

    string fname;

    char type;

    ifstream fin;

    List<double>\* list;

    int size;

    ofstream fout;

    while(1){

        cout << "Input the type of the data structure (l(for list)/a(for array)) or input 'q' to stop the program:\n";

        cin >> type;

        switch (type){

            case 'q':

                cout << "You choose to end the programm!\n";

                return 0;

            case 'l':

                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;

                }

                size = getData(arr, fin);

                fin.close();

                cout << "Input the path to result file:\n";

                cin >> fname;

                cout << "Do you want to rewrite it (y/n)?:\n";

                cin >> type;

                switch (type){

                    case 'y':

                        fout.open(fname, ofstream::trunc);

                        break;

                    case 'n':

                        fout.open(fname, ofstream::app);

                        break;

                    default:

                        cout << "Error command! File will be rewritten!\n";

                        fout.open(fname, ofstream::trunc);

                        break;

                }

                if(!fout.is\_open()){

                    cout << "Opening file for writing result data failed! Try again!\n";

                    break;

                }

                list = new List<double>;

                list->makeList(arr, size);

                cout<< "List before sort: ";

                fout<< "List before sort: ";

                list->printList(fout);

                if(size == 1){

                    cout << "Only one elememt in array! Sort doesn't need.\n";

                    fout << "Only one elememt in array! Sort doesn't need.\n";

                }

                list->sort(fout);

                cout<< "List after sort: ";

                fout<< "List after sort: ";

                list->printList(fout);

                cout<< "\n";

                fout<< "\n";

                fout.close();

                delete list;

                list = nullptr;

                break;

            case 'a':

                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;

                }

                size = getData(arr, fin);

                fin.close();

                cout << "Input the path to result file:\n";

                cin >> fname;

                cout << "Do you want to rewrite it (y/n)?:\n";

                cin >> type;

                switch (type){

                    case 'y':

                        fout.open(fname, ofstream::trunc);

                        break;

                    case 'n':

                        fout.open(fname, ofstream::app);

                        break;

                    default:

                        cout << "Error command! File will be rewritten!\n";

                        fout.open(fname, ofstream::trunc);

                        break;

                }

                if(!fout.is\_open()){

                    cout << "Opening file for writing result data failed! Try again!\n";

                    break;

                }

                cout<< "Array before sort: ";

                fout<< "Array before sort: ";

                printArr(arr, fout, size);

                sort(arr, size, fout);

                cout<< "Array after sort: ";

                fout<< "Array after sort: ";

                printArr(arr, fout, size);

                cout<< "\n";

                fout<< "\n";

                fout.close();

                delete[] arr;

                arr = nullptr;

                break;

            default:

                cout << "Error command! Try again!\n";

                break;

        }

    }

    return 0;

}