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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема:Иерархические списки

Студент гр. 9304 _____ Краев Д.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Изучить, что такое иерархический список. Написать программу, использующую иерархически список для своей работы.

Основные теоретические положения.

Иерархическое содержание состоит из записей, каждая запись имеет иерархический номер, название исодержит либо несколько подчинённых записей ("организующая запись"), либо текст ("текстовая запись"). В текстовой записи "3.3.4 Главы 3-4 [текст глав]" "3.3.4" -это иерархический номер записи, "4" -относительный номер записи, "Главы 3-4" -название записи, а в квадратных скобках приведён текст записи.Иерархическое содержание может быть представлено иерархическим списком, каждый элемент которого имеет название и данные -список подчинённых записей либо текст. Позиция записи в иерархическом списке определяется её номером. В элементе иерархического списка следует хранить относительный номер записи, а не её полный иерархический номер. Относительный номер каждого последующего элемента в списке должен быть больше, чем у предыдущего.Запись с номером a.b.c обязательно должна быть в списке подчинённых записей элемента a.b (не в списке элемента a).

Задание.

Разработать программу, создающую иерархический список, соответствующий иерархическому содержанию, записанному в файле в приведённом выше формате. Записи могут быть в файле перепутаны, но все имеют различные номера. Программа должна позволять выводить иерархическое содержание на экран или в файл (в правильном порядке), добавлять новые записи (если запись с заданным номером уже существует, то сообщать об ошибке), а также выполнять дополнительные действия, в зависимости от варианта:28) -Удалить элемент из списка по номеру (удаляются также все подчинённые элементы);-Удалить элемент из списка по названию (если элементов с таким названием несколько, то среди них удалить

элемент с наименьшим номером);-Редактировать название элемента;-Изменить иерархический номер записи (если новый номер X.b уже существует или если отсутствует запись X,то сообщается об ошибке, иначе номер записи меняется на X.b и при необходимости выполняется перенос (здесь X -последовательность целых чисел (возможно, пустая), разделённых точками)).

Выполнение работы.

Была разработана программа, которая считывает иерархическое содержание из файла. После считывания она составляет иерархический список и может выполнить с ним 1 из 5 действий: добавление элемента, удаление элемента по номеру, удаление элемента по названию, изменение номера и изменение названия.

1) Классы

1.1) Класс Node

Класс Node содержит 5 полей: vector<int> number, Node* next, string name, bool isKnot, variant<Node*, string> value.

Поле number содержит массив, состоящий из номеров иерархического номера. Поле next содержит адрес на следующий элемент списка. Поле name содержит название элемента. Поле isKnot содержит информацию о том, является ли элемент узлом. Поле value содержит либо текст, либо адрес на другой элемент списка.

Класс Node также имеет метод print, который печатает содержимое класса на экран.

Конструктор класса Node, принимает на вход строку и записывает ее содержание в соответствующие поля.

1.2) Класс List

Класс List содержит 1 поле Node* firstElem, которое содержит адрес на первый элемент. Также класс List имеет 5 методов: print, addElem,deleteElem, changeName и changeNumber.

Метод print печатает содержимое списка на экран.

Метод deleteElem удаляет элемент списка, либо по номер, либо по названию, в зависимости от входных данных. Если подан вектор чисел, то он будет подавать удалять элемент, иерархический номер которого соответствует элементам данного вектора. Если подана строка, то удаляет элемент с названием, соответствующим поданной строке.

Метод addElem принимает на вход строку, содержащую иерархическое содержание элемента. Далее добавляет его в список в место, соответствующее его иерархическому номеру

Метод changeElem принимает на вход номер элемента, который нужно удалить, и новое название. Метод ищет соответствующий элемент и изменяет его название на новое.

Метод changeNumber принимает на вход номер элемента, номер которого нужно изменить, и номер, на который нужно изменить. Метод ищет элемент с соответствующим номером, изменяет номер и перемещает его в место, соответствующее новому номеру.

Метож changeName принимает на вход номер элемента, который нужно изменить и строку, содержащее новое название. Метод сначала ищет соответствующий элемент и далее меняет его название.

Конструктор принимает на вход название файла, в котором содержится иерархическое содержание. Сначала он открывает этот файл, считывает с него все иерархическое содержание и далее создает иерархический список.

1.3) Функции

Также для реализации программы были написаны несколько функций.

Функция min принимает на вход ссылку на вектор указателей на Node и иерархический номер узла N. Функция ищет минимальный элемент типа N.x, где N — номер узла подаваемый функции, а x — любое число.

Функция printList принимает на вход указатель на элемент списка, с которого нужно начать печатать список, и число, содержащаее глубину.

Если какой-то элемент является узлом, то программа вызывает саму себя и начинает печатать элементы списка, указатель на первый элемент которого содержит узел.

Функция createList принимает на вход ссылку на вектор указателей на Node и вектор, содержащий иерархический номер, функция создает список, элементы которого расположены по возрастанию их иерархического номера. Если элемент является узлом, то вызывает саму себя и создает список, состоящий из подэлементов этого узла.

Функция clean очищает элементы списка, принимает на вход указатель на Node, с которого нужно начать очистку. Если какой-то элемент является узлом и в поле значения содержит другой список, то функция начинает вызывать саму себя чтоб очистить этот список.

1.4) Command line interface

Для вызова работы функций программы был создан CLI.

Команды CLI:

./lab -a “<иерархическое содержание элемента>“ - добавление элемента

./lab -b “<номер элемента>“ - удаление элемента по номеру

./lab -c “<название элемента>“ - удаление элемента по названию

./lab -d “<номер элемента>“ “<новое название>“ - изменение названия элемента

./lab -e “<номер элемента>“ “<новый номер>“ - изменение номера элемента

Тестирование

Тестирование проводится с помощью скрипта, написанном на языке Python. Скрипт использует библиотеки unittest и subprocess. Скрипт проводит 5 тестов на использование каждой функции. Выходные данные сравниваются с корректным выводом программы, содержащимся в файлах test1-5.txt. Библиотека subprocess нужна для запуска программы с нужными входными данными, а библиотека unittest для проведения тестирования.

Скрипт можно запустить при помощи команды «make run_tests». Результаты тестирования можно посмотреть в приложении В.

Визуализация иерархического списка

1 part 1

1.1 chapter 1 [jgfyuj]

1.2 chapter 2 [sdffdb]

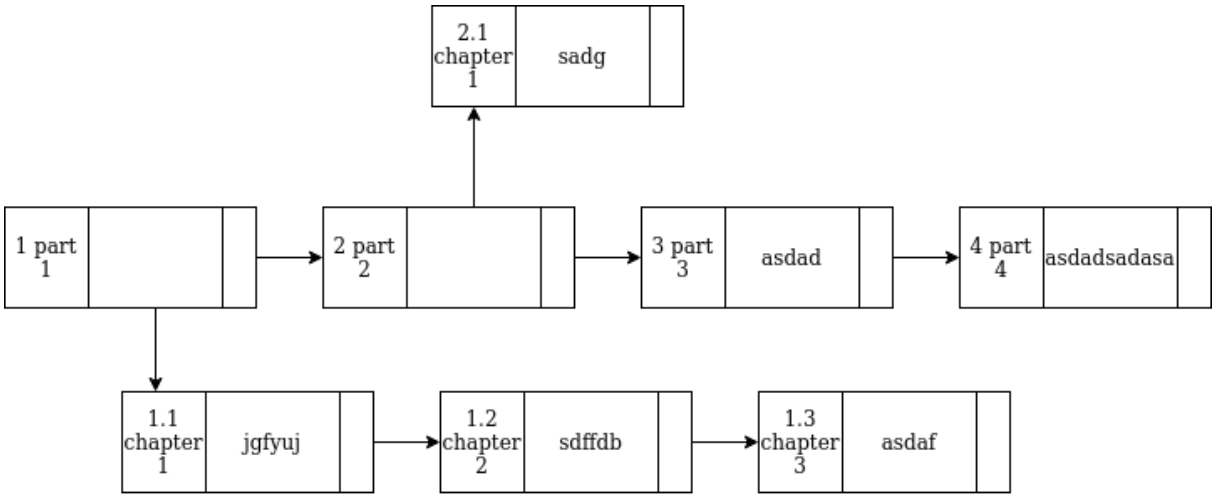
1.3 chapter 3 [asdaf]

2 part 2

2.1 chapter 1 [sadg]

3 part 3 [asdad]

4 part 4 [asdadsadasa]



Выводы

Была изучена такая структура данных, как иерархический списос. Была написана программа, использующая иерархический список.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

```
#include "Node.h"
#include "List.h"
#include <iostream>
#include <memory>
#include <vector>
#include <fstream>
#include <unistd.h>

struct globalArgs_t {
    int noIndex;
    char *langCode;
    const char *outFileName;
    FILE *outFile;
    int verbosity;
    int numInputFiles;
    char **inputFiles;
} globalArgs;


int main(int argc, char** argv){
        List list("text.txt");

        int opt =0;
        globalArgs.noIndex = 0;
    globalArgs.langCode = NULL;
    globalArgs.outFileName = NULL;
```

```
globalArgs.outFile = NULL;
globalArgs.verbosity = 0;
globalArgs.inputFiles = NULL;
globalArgs.numInputFiles = 0;

    std::vector<int> number, number2;
std::string num;

    char* optString = "abcde";
    opt = getopt( argc, argv, optString );

    while( opt != -1 ) {
   switch( opt ) {
      case 'a':             // добавление элемента
                            list.addElem(argv[2]);
                            break;

      case 'b':             // удаление элемента по номеру
                            while(*argv[2] != '\0' ){
                                    if(*argv[2] != '.'){
                                            num.push_back(*argv[2]);
                                    }else{
                                            number.push_back(stoi(num));
                                            num.clear();
                                    }
                                    argv[2]++;
                            }
                            number.push_back(stoi(num));
                            list.deleteElem(number);
```

```
                    break;

case 'c':           // удаление элемента по названию
   list.deleteElem(argv[2]);
                    break;

case 'd':           // изменение названия
   while(*argv[2] != '\0' ){
      if(*argv[2] != '.'){
         num.push_back(*argv[2]);
      }else{
         number.push_back(stoi(num));
         num.clear();
      }
      argv[2]++;
   }
                    number.push_back(stoi(num));
                    list.changeName(number, argv[3]);
                    break;

case 'e':           // изменение номера
                    while(*argv[2] != '\0' ){
      if(*argv[2] != '.'){
         num.push_back(*argv[2]);
      }else{
         number.push_back(stoi(num));
         num.clear();
      }
      argv[2]++;
```

```
            }
                        number.push_back(stoi(num));
                        num.clear();
                        while(*argv[3] != '\0' ){
               if(*argv[3] != '.'){
                  num.push_back(*argv[3]);
               }else{
                  number2.push_back(stoi(num));
                  num.clear();
               }
               argv[3]++;
            }
            number2.push_back(stoi(num));

                        list.changeNumber(number, number2);
            break;

         default:
            break;
      }

      opt = getopt( argc, argv, optString );
   }
      list.print();
      return 0;
}
```

Файл: Node.h

```
#pragma once
#include <string>
#include <variant>
#include <vector>
#include <iterator>
#include <iostream>

class Node{
public:
	Node* next;

	std::vector<int> number;

	std::string name;

	bool isKnot;

	std::variant<Node*, std::string> value;

	Node(std::string &str);

	void print();

	~Node();
};
```

Файл: Node.cpp

```
#include "Node.h"



Node::Node(std::string &str){
	std::string::const_iterator iter = str.begin();
	int n=0;

	while(*iter == ' ' || *iter == '\n' || *iter == '\t'){
		iter++;
	}
	if(*iter == '\0'){
		this->value = nullptr;
		this->next = nullptr;
	}else{
		try{
			if(*iter < 48 || *iter > 57)
				throw 1;
		}catch(int){
			std::cerr << "ERROR: the hierarchical content is set incorrectly\
n";
			exit(0);
		}
		std::string num;
		while(*iter != ' '){
			try{
		if((*iter < 48 || *iter > 57) && *iter != '.')
			throw 1;
		}catch(int){
```

```
std::cerr << "ERROR: the hierarchical content is set incorrectly\n";
exit(0);
}
        if(*iter == '.'){
                this->number.push_back(stoi(num));
                iter++;
                num.clear();
        }else{
                num.push_back(*iter);
                iter++;
        }
}
this->number.push_back(stoi(num));
std::string text;
try{
        while(*iter != '['){
                if(*iter == '\0')
                        break;
                this->name.push_back(*iter);
                iter++;
        }
        if(*iter == '['){
                while(*iter != ']'){
                        if(*iter == '\0')
                                throw 1;
                        text.push_back(*iter);
                        iter++;
                }
                text.push_back(*iter);
```

```
                iter++;
                this->value = text;
                this->isKnot = false;
            }else{
                this->isKnot = true;
            }
        }catch(int){
            std::cerr << "ERROR:the hierarchical content is set incorrectly\
n";
            exit(0);
        }
        this->value = text;
    }
}



void Node::print(){
    if(this->number.size() != 0 && this->name.size() != 0){
        int last=0;
        for(int i=0;i < this->number.size()-1;i++){
            std::cout << this->number[i] << '.';
            last++;
        }
        std::cout << this->number[last];
        std::cout << this->name;

        if(std::get_if<std::string>(&this->value))
            std::cout << std::get<std::string>(this->value);
```

```
            std::cout << '\n';
        }
}


Node::~Node(){};
```

Файл: List.h

```
#pragma once
#include "Node.h"
#include <string>
#include <vector>
#include <iostream>
#include <iterator>
#include <variant>
#include <fstream>

Node* min(std::vector<Node*> &nodes, std::vector<int> N);
void print(Node* node,  int N);
Node* createList(std::vector<Node*> &nodes, std::vector<int> N={});
void clean(Node* node);

class List{
private:
public:
        Node* firstElem;
        friend class Node;
        List(std::string text);
        void print();
```

```
        void addElem(std::string newElem);
        void deleteElem(std::vector<int> elem);
        void deleteElem(std::string elem, Node* cur = nullptr);
        void changeName(std::vector<int> elem, std::string newName);
        void changeNumber(std::vector<int> number, std::vector<int> newNumber);
        ~List();
};
```

Файл: List.cpp

```
#include "List.h"


Node* createList(std::vector<Node*> &nodes, std::vector<int> N){
        Node* list = min(nodes, N);
        Node* cur = list;
        int size = nodes.size();
        for(int i = 0;i < size;i++){
                cur->next = min(nodes, N);
                if(cur->next == nullptr)
                        break;
                cur = cur->next;
        }
        cur = list;
        while(cur != nullptr){
                if(cur->isKnot == true){
          cur->value = createList(nodes,cur->number);
        }
                cur = cur->next;
        }
```

```
        return list;
}

Node* min(std::vector<Node*> &nodes, std::vector<int> N){
        Node* min=nullptr;
        int m=-1;
        int count = 0;
        for(int i = 0;i < nodes.size();i++){
                if(nodes[i]->number.size() == N.size()+1){
                        for(int j = 0;j < N.size();j++){
                    if(nodes[i]->number[j] == N[j]){
                        count++;
                    }
                 }
                        if(count == N.size()){
                                min = nodes[i];
                                m = i;
                                count =0;
                                break;
                        }
                }
                count =0;
        }
        if(min == nullptr){
                return min;
        }
        for(int i = 0;i < nodes.size();i++){
                if(nodes[i]->number.size() == N.size()+1){
                        for(int j = 0;j < N.size();j++){
```

```
                        if(nodes[i]->number[j] == N[j]){
                            count++;
                        }
                    }
                    if(count == N.size() && nodes[i]->number[nodes[i]->number.size()-1] < min->number[min->number.size()-1]){
                        min = nodes[i];
                        m = i;
                    }
                    count = 0;
                    }
                }
                if(m != -1){
                    for(int i = m;i < nodes.size()-1;i++){
                        nodes[i] = nodes[i+1];
                    }
                    nodes.pop_back();
                }
                return min;
}

void printList(Node* node, int N){
                Node* cur = node;
                Node* oldCur;
                while(cur){
                    for(int i=0;i<N;i++)
                        std::cout << '\t';
                    cur->print();
                    if(std::get_if<Node*>(&cur->value)){
```

```
                oldCur = cur;
                cur = std::get<Node*>(cur->value);
                printList(cur, N+1);
                cur = oldCur;
            }
            cur = cur->next;
        }
}


List::List(std::string file){
        std::vector<Node*> nodes;
        std::ifstream f(file);
        std::string str;
        if(f.is_open()){
        while(getline(f,str)){
                nodes.push_back(new Node(str));
            }
        }
        if(nodes[nodes.size() -1]->number.size()==0){
            nodes.pop_back();
        }
        this->firstElem = createList(nodes);
        for(int i = 0;i<nodes.size();i++)
            delete nodes[i];

}

void List::addElem(std::string newElem){
```

```
        Node* node = new Node(newElem);
        Node* cur = this->firstElem;
        Node* next;
        for(int i = 0;i < node->number.size()-1;i++){
                while(node->number[i] != cur->number[i]){
                        cur = cur->next;
                }
                try{
                        if(cur->isKnot!=true)
                                throw 1;
                        else
                                cur = std::get<Node*>(cur->value);
                }catch(int){
                        std::cout << "ERROR: an element isn't a knot\n";
                        exit(0);
                }
        }
        while(node->number[node->number.size()-1]   >=   cur->next->number[cur-
>next->number.size()-1]){
                try{
                        cur = cur->next;
                        if(cur->next == nullptr)
                                break;
                        if(node->number[node->number.size()-1]  ==  cur->number[cur-
>number.size()-1])
                                throw 1;
                }catch(int){
                        std::cerr << "ERROR: an element with this number exists\n";
                        exit(0);
```

```
                }
        }
        if(cur->next != nullptr){
        next = cur->next;
        cur->next = node;
        node->next = next;
    }
    if(cur->next == nullptr){
                cur->next = node;
        node->next = nullptr;
        }
}

void List::deleteElem(std::vector<int> elem){
        Node* cur = this->firstElem;
        Node* knot;
        for(int i = 0;i < elem.size()-1;i++){
        while(elem[i] != cur->number[i]){
            cur = cur->next;
        }

                try{
            if(cur->isKnot!=true)
                throw 1;
            else
                knot = cur;
                                cur = std::get<Node*>(cur->value);
        }catch(int){
            std::cout << "ERROR: an element isn't a knot\n";
```

```
        exit(0);
    }
     }

            if(elem[elem.size()-1] == cur->number[cur->number.size()-1]){
                if(cur == this->firstElem)
                    this->firstElem = cur->next;
                knot->value = cur->next;
                if(cur->isKnot == true){
          clean(std::get<Node*>(cur->next->value));
       }
                delete cur;
            }else{
                try{
                while(elem[elem.size()-1]  !=  cur->next->number[cur->next-
>number.size()-1]){
                    cur = cur->next;
                    if(cur->next == nullptr){
                        throw 1;
                        break;
                    }
                }

                }catch(int a){
            std::cerr << "ERROR: an element doesn't exist\n";
            exit(0);
            }

                Node* prev = cur;
```

```
                Node* next = cur->next->next;
                if(cur->next->isKnot == true){
                        clean(std::get<Node*>(cur->next->value));
                }
                delete cur->next;
                prev->next = next;
        }
}

void List::deleteElem(std::string elem, Node* cur){
        if(cur == nullptr)
                cur = this->firstElem;
        Node* next;
        while(cur->next != nullptr){
                if(std::get_if<Node*>(&cur->value)){
                        if(std::get<Node*>(cur->value)->name == elem){
                                next = std::get<Node*>(cur->value);
                                cur->value = std::get<Node*>(cur->value)->next;
                                delete next;
                                break;
                        }
                }
                if(cur->next->name == elem){
                        next = cur->next;
                        cur->next = cur->next->next;
                        delete next;
                        break;
                }
                if(cur->isKnot == true && std::get<Node*>(cur->value) != nullptr){
```

```
                deleteElem(elem, std::get<Node*>(cur->value));
            }
            cur = cur->next;
        }
}

void List::changeName(std::vector<int> elem, std::string newName){
        Node* cur = this->firstElem;
   for(int i = 0;i < elem.size()-1;i++){
      while(elem[i] != cur->number[i]){
         cur = cur->next;
      }
      try{
         if(cur->isKnot!=true)
            throw 1;
         else
            cur = std::get<Node*>(cur->value);
      }catch(int){
         std::cout << "ERROR: an element isn't a knot\n";
         exit(0);
      }
   }
        if(elem[elem.size()-1] == cur->number[cur->number.size()-1]){
            cur->name = newName;
   }else{
        try{
            while(elem[elem.size()-1]  !=  cur->number[cur->next->number.size()-
1]){
            cur = cur->next;
```

```
                    if(cur->next == nullptr){
                throw 1;
                break;
            }
                    }

        }catch(int a){
            std::cerr << "ERROR: an element doesn't exist\n";
          exit(0);
          }
                    cur->name = newName;
          }
}

void List::changeNumber(std::vector<int> number, std::vector<int> newNumber){
          Node* cur = this->firstElem;
          Node* newNode;
          Node* knot;
          Node* next;
          for(int i = 0;i < number.size()-1;i++){
        while(number[i] != cur->number[i]){
                    cur = cur->next;
        }
        try{
            if(cur->isKnot!=true)
                throw 1;
            else
                knot = cur;
                cur = std::get<Node*>(cur->value);
```

```
    }catch(int){
        std::cout << "ERROR: an element isn't a knot\n";
        exit(0);
    }
  }

  if(number[number.size()-1] == cur->number[cur->number.size()-1]){
    cur->number = newNumber;
            newNode = cur;
            knot->value = cur->next;
  }else{
     try{
            while(number[number.size()-1]  !=  cur->next->number[cur->next-
>number.size()-1]){
                cur = cur->next;
        if(cur->next == nullptr){
            throw 1;
            break;
        }
            }

    }catch(int a){
        std::cerr << "ERROR: an element doesn't exist\n";
        exit(0);
    }


            cur->next->number = newNumber;
     newNode = cur->next;
```

```
        next = cur->next->next;
        cur->next = next;
        }
        cur = this->firstElem;
    next = nullptr;
    for(int i = 0;i < newNode->number.size()-1;i++){
       while(newNode->number[i] != cur->number[i]){
                        cur = cur->next;
       }

       try{
          if(cur->isKnot!=true)
             throw 1;
          else
             cur = std::get<Node*>(cur->value);
       }catch(int){
          std::cout << "ERROR: an element isn't a knot\n";
          exit(0);
       }
    }

        while(newNode->number[newNode->number.size()-1] >= cur->number[cur-
>number.size()-1]){

                try{

          if(cur->next == nullptr)
             break;
                        else
```

```
                cur = cur->next;
            if(newNode->number[newNode->number.size()-1] == cur->number[cur-
>number.size()-1])
            throw 1;
        }catch(int){
          std::cerr << "ERROR: an element with this number exists\n";
          exit(0);
        }
    }

    if(cur->next != nullptr){
        next = cur->next;
        cur->next = newNode;
        newNode->next = next;
    }
    if(cur->next == nullptr){
        cur->next = newNode;
        newNode->next = nullptr;
    }
         cur = cur->next;
}

void clean(Node* node){
        if(node != nullptr){
                Node* cur = node;
                Node* prev;
                while(cur->next != nullptr){
                        if(cur->isKnot == true){
                                clean(std::get<Node*>(cur->value));
```

```
                }
                prev = cur;
                cur = cur->next;
                delete prev;
            }
            delete cur;
        }
}

void List::print(){
        int N=0;
        printList(this->firstElem, N);
}

List::~List(){
        clean(this->firstElem);
};
```

Файл: Makefile

```
all: lab

lab: main.o Node.o List.o
   g++ main.o Node.o List.o -std=c++17 -o lab

main.o: main.cpp
   g++ main.cpp -c -std=c++17

Node.o: Node.cpp
   g++ Node.cpp -c -std=c++17
```

```
List.o: List.cpp
    g++ List.cpp -c -std=c++17

run_tests:
    python test.py

clean:
    rm -rf *.o lab
```

ПРИЛОЖЕНИЕ В

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	-a "5 part 5 [qwerty] "	1 part 1 1.1 chapter 1 [jgfyuj] 1.2 chapter 2 [sdffdb] 1.3 chapter 3 [asdaf] 2 part 2 2.1 chapter 1 [sadg] 3 part 3 [asdad] 4 part 4 [asdadsadasa] 5 part 5 [qwerty]
2.	-c " chapter 1 "	1 part 1 1.2 chapter 2 [sdffdb] 1.3 chapter 3 [asdaf] 2 part 2 2.1 chapter 1 [sadg] 3 part 3 [asdad] 4 part 4 [asdadsadasa]
3.	-b "1.2"	1 part 1 1.1 chapter 1 [jgfyuj] 1.3 chapter 3 [asdaf] 2 part 2 2.1 chapter 1 [sadg] 3 part 3 [asdad] 4 part 4 [asdadsadasa]
4.	-d "1.1" " Hello "	1 part 1 1.1 Hello [jgfyuj] 1.2 chapter 2 [sdffdb] 1.3 chapter 3 [asdaf] 2 part 2

		2.1 chapter 1 [sadg] 3 part 3 [asdad] 4 part 4 [asdadsadasa]
5.	-e "1.1" "2.2"	1 part 1 1.2 chapter 2 [sdffdb] 1.3 chapter 3 [asdaf] 2 part 2 2.1 chapter 1 [sadg] 2.2 chapter 1 [jgfyuj] 3 part 3 [asdad] 4 part 4 [asdadsadasa]