МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ №1**

**«**Абстрактные структуры данных**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Мартынова Е.Д.*  «9» ноября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г.А.*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

Оглавление

[Цели и задачи работы: 3](#_Toc182168286)

[Методика выполнения работы 3](#_Toc182168287)

[Реализация абстрактных структур данных. 3](#_Toc182168288)

[Структура программы 5](#_Toc182168289)

[main.cpp 5](#_Toc182168290)

[header.h 5](#_Toc182168291)

[Массив (Array.cpp) 5](#_Toc182168292)

[Очередь (Queue.cpp) 5](#_Toc182168293)

[Стек (Stack.cpp) 5](#_Toc182168294)

[Односвязный список (OneList.cpp) 5](#_Toc182168295)

[Двусвязный список (TwoList.cpp) 5](#_Toc182168296)

[AVL-дерево (AVLtree.cpp) 5](#_Toc182168297)

[Хэш-таблица (HashTable.cpp) 5](#_Toc182168298)

[Перечень операций (INFO.txt) 5](#_Toc182168299)

[Функциональная характеристика программы 5](#_Toc182168300)

[Примеры работы программы 5](#_Toc182168301)

[Схемы работы программы 5](#_Toc182168302)

[Сложность выполнения каждой операции с позиции BigO нотации 5](#_Toc182168303)

[Ссылки на репозиторий 5](#_Toc182168304)

[GitHub 5](#_Toc182168305)

Цели и задачи работы: изучение алгоритмов формирования и работы с абстрактными структурами данных.

**Задание к работе**: Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Языки программирования**: C, C++, Rust, Go

# Методика выполнения работы

1. Разработать алгоритм решения задачи по индивидуальному заданию.
2. Написать и отладить программу решения задачи (С, Go или

Rust).

1. Протестировать работу программы на различных исходных данных.
2. По запросу преподавателя быть готовым модифицировать алгоритм и добавить операцию работы с данными.

# Реализация абстрактных структур данных.

1. Реализовать структуры данных с базовым набором операций:
   1. Массив. Операции: добавление элемента (по индексу и в конец массива), получение элемента по индексу, удаление элемента по индексу, замена элемента по индексу, длина массива, чтение.
   2. Список (односвязный, двусвязный). Операции: добавление элемента в голову и в хвост, удаление элемента с головы и с хвоста, удаление элемента по значению, поиск элемента по значению, чтение.
   3. Очередь. Операции: добавление и удаление элемента (push и

pop), чтение.

* 1. Стек. Операции: добавление и удаление элемента (push и pop), чтение.
  2. Хеш таблицы. Операции: предусмотреть работу с коллизиями добавление элемента (ключ-значение), получение значения элемента по ключу, удаление элемента по ключу.
  3. Деревья (вариант 11 Complete Binary Tree, вариант 2 Сбалансированное двоичное дерево (АВЛ-дерево), вариант 3 Full Binary Tree). Операции: добавление элемента, поиск элемента, удаление элемента с сохранением структуры дерева, чтение. Если Full Binary Tree или Complete Binary Tree, то вместо удаления элемента добавить операцию проверки на Full или Complete соответственно.

1. Реализовать интерфейс работы со структурами данных. Предусмотреть считывание из файла и запись в файл при внесении изменений в состав элементов.

Часть операций для реализации представлено в таблице. Дополнить список операций согласно п.1 и сохраняя стиль написания команды (первая буква

– принадлежность структуре данных). **Исключение: Операция PRINT выводит любую структуру данных на экран.**

1. Обосновать сложность выполнения каждой операции с позиции BigO

нотации. Составить таблицу BigO по операциям и структурам в отчете.

1. В отчете в контрольном примере представить визуальные формы, подтверждающие соответствие результатам работы программы.

# Структура программы

## main.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <algorithm>

#include <cstring>

using namespace std;

#include "header.h"

#include "Array.cpp"

#include "Queue.cpp"

#include "Stack.cpp"

#include "OneList.cpp"

#include "TwoList.cpp"

#include "AVLtree.cpp"

#include "HashTable.cpp"

template <typename T>

void WritingFromFileToStructure(T& structure, void (T::\* pushMethod)(string&), const string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

(structure.\*pushMethod)(element);

}

}

FileRead.close();

}

template <typename T>

void WritingFromStructureToFile(T& structure, const string& filename ) {

ofstream FileWrite(filename);

SingleNode\* current = structure.head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

void inputArguments(int argc, char\* argv[], string &filename, string &query) {

for (int i = 1; i < argc; i++) {

if (strcmp(argv[i], "--file") == 0) {

if (i + 1 < argc) {

filename = argv[i + 1];

i++;

} else {

cerr << "ERROR!!!!! Missing --file" << endl;

exit(1);

}

} else if (strcmp(argv[i], "--query") == 0) {

if (i + 1 < argc) {

query = argv[i + 1];

i++;

} else {

cerr << "ERROR!!!!! Missing --query" << endl;

exit(1);

}

}

}

}

int main(int argc, char\* argv[])

{

string filenameConsole = "";

string queryConsole = "";

string command;

inputArguments(argc, argv, filenameConsole, queryConsole);

if (queryConsole.empty()) {

string command;

cout << "Enter the command: ";

getline(cin, command);

queryConsole = command;

}

try {

string commandWord;

string name;

string element;

string index;

command = queryConsole;

Array array;

Queue queue;

Stack stack;

OneList onelist;

TwoList twolist;

AVL avl;

HashTable hashtable;

stringstream ss(command);

ss >> commandWord;

ss >> name;

ss >> element;

ss >> index;

string filename = name + ".txt";

if (command.find("INFO") != string::npos) {

ifstream File("INFO.txt");

string temp;

cout << endl;

while (getline(File, temp)) {

cout << temp << endl;

}

}

else if (command.find("EXIT") != string::npos) {

cout << endl << "GOOD BYE!" << endl;

return 0;

}

else if (command.at(0) == 'M') {

WritingFromFileToStructure(array, &Array::MPUSHend, filename);

;

if (command.find("MPUSH") != string::npos) {

if (index == "") {

array.MPUSHend(element);

}

else {

array.MPUSHindex(stoi(index), element);

}

}

else if (command.find("MDEL") != string::npos) {

index = element;

array.MDEL(stoi(index));

}

else if (command.find("MGETL") != string::npos) {

index = element;

string cell = array.MGETL(stoi(index));

cout << endl << cell << endl;

return 0;

}

else if (command.find("MREPL") != string::npos) {

array.MREPL(stoi(index), element);

}

else if (command.find("MSIZE") != string::npos) {

int cell = array.MSIZE();

cout << endl << cell << endl;

return 0;

}

else if (command.find("MREAD") != string::npos) {

array.MREAD();

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

ofstream FileWrite(name + ".txt");

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

FileWrite << array.MGETL(i);

}

FileWrite.close();

}

else if (command.at(0) == 'Q') {

WritingFromFileToStructure(queue,& Queue::QPUSH, filename);

if (command.find("QPUSH") != string::npos) {

queue.QPUSH(element);

}

else if (command.find("QPOP") != string::npos) {

queue.QPOP();

}

else if (command.find("QREAD") != string::npos) {

queue.QREAD();

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

WritingFromStructureToFile(queue, filename);

}

else if (command.at(0) == 'S') {

WritingFromFileToStructure(stack,& Stack::SPUSH, filename);

if (command.find("SPUSH") != string::npos) {

stack.SPUSH(element);

}

else if (command.find("SPOP") != string::npos) {

stack.SPOP();

}

else if (command.find("SREAD") != string::npos) {

stack.SREAD();

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

WritingFromStructureToFile(stack, filename);

}

else if (command.substr(0, 2) == "LO") {

WritingFromFileToStructure(onelist,& OneList::LONEPUSHtail, filename);

if (command.find("LONEPUSHhead") != string::npos) {

onelist.LONEPUSHhead(element);

}

else if (command.find("LONEPUSHtail") != string::npos) {

onelist.LONEPUSHtail(element);

}

else if (command.find("LONEDELhead") != string::npos) {

onelist.LONEDELhead();

}

else if (command.find("LONEDELtail") != string::npos) {

onelist.LONEDELtail();

}

else if (command.find("LONEDELvalue") != string::npos) {

onelist.LONEDELvalue(element);

}

else if (command.find("LONESEARCH") != string::npos) {

onelist.LONESEARCH(element);

return 0;

}

else if (command.find("LONEREAD") != string::npos) {

onelist.LONEREAD();

return 0;

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

WritingFromStructureToFile(onelist, filename);

}

else if (command.substr(0, 2) == "LT") {

WritingFromFileToStructure(twolist,& TwoList::LTWOPUSHtail, filename);

if (command.find("LTWOPUSHhead") != string::npos) {

twolist.LTWOPUSHhead(element);

}

else if (command.find("LTWOPUSHtail") != string::npos) {

twolist.LTWOPUSHtail(element);

}

else if (command.find("LTWODELhead") != string::npos) {

twolist.LTWODELhead();

}

else if (command.find("LTWODELtail") != string::npos) {

twolist.LTWODELtail();

}

else if (command.find("LTWODELvalue") != string::npos) {

twolist.LTWODELvalue(element);

}

else if (command.find("LTWOSEARCH") != string::npos) {

twolist.LTWOSEARCH(element);

return 0;

}

else if (command.find("LTWOREAD") != string::npos) {

twolist.LTWOREAD();

return 0;

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

ofstream FileWrite(name + ".txt");

DoubleNode\* current = twolist.head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

else if (command.at(0) == 'T') {

WritingFromFileToStructure(avl, &AVL::TPUSH, filename);

if (command.find("TPUSH") != string::npos) {

avl.TPUSH(element);

}

else if (command.find("TDEL") != string::npos) {

avl.TDEL(element);

}

else if (command.find("TSEARCH") != string::npos) {

if (avl.TSEARCH(element)) {

cout << endl << "The element was found!!!" << endl;

}

else {

cout << endl << "The element was not found" << endl;

}

}

else if (command.find("TREAD") != string::npos) {

avl.TREAD();

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

ofstream FileWrite(name + ".txt");

avl.TSaveToFile(avl.root, FileWrite);

FileWrite.close();

}

else if (command.at(0) == 'H') {

string key = element;

string value = index;

ifstream FileRead(filename);

if (FileRead.is\_open()) {

string key, value;

while (FileRead >> key >> value) {

hashtable.HPUSH(key, value);

}

FileRead.close();

}

if (command.find("HPUSH") != string::npos) {

hashtable.HPUSH(key, value);

}

else if (command.find("HDEL") != string::npos) {

hashtable.HDEL(key);

}

else if (command.find("HGET") != string::npos) {

string temp = hashtable.HGET(key);

cout << endl << temp << endl;

}

else if (command.find("HREAD") != string::npos) {

hashtable.HREAD();

}

else {

cout << endl << "Command not found!" << endl;

return 0;

}

ofstream FileWrite(filename);

if (FileWrite.is\_open()) {

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

NodeHash\* current = hashtable.table[i];

while (current != nullptr) {

FileWrite << current->key << " " << current->value << endl;

current = current->next;

}

}

FileWrite.close();

}

}

else {

cout << endl << "The command was not found!" << endl;

return 0;

}

}

catch (...) {

cerr << endl << "UNKNOWN ERROR!!!" << endl;

}

}

## header.h

#pragma once

using namespace std;

#include <iostream>

struct SingleNode {

string cell;

SingleNode\* next;

};

struct DoubleNode {

string cell;

DoubleNode\* next;

DoubleNode\* prev;

};

struct AVLnode {

string key;

AVLnode\* left;

AVLnode\* right;

int height;

AVLnode(string key) : key(key), left(nullptr), right(nullptr), height(1) {}

};

struct NodeHash {

string key;

string value;

NodeHash\* next;

NodeHash(string& k, string& v) : key(k), value(v), next(nullptr) {}

};

struct Array {

string\* data;

int size;

int maxSize;

Array();

~Array();

void MPUSHend(string& value);

void MPUSHindex(const int& index, string& value);

string MGETL(const int& index);

void MDEL(const int& index);

void MREPL(const int& index, string& value);

int MSIZE();

void MREAD();

};

Array::Array() : size(0), maxSize(1000) {

data = new string[maxSize];

}

Array::~Array() {

delete[] data;

}

struct Queue {

string cell;

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

~Queue();

void QPUSH(string& cell);

void QPOP();

void QREAD();

void Clear();

};

Queue::~Queue() {

Clear();

}

struct Stack {

string cell;

~Stack();

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

void SPUSH(string& cell);

void SPOP();

void SREAD();

void Clear();

};

Stack::~Stack() {

Clear();

}

struct OneList {

~OneList();

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

void LONEPUSHhead(string& cell);

void LONEPUSHtail(string& cell);

void LONEDELhead();

void LONEDELtail();

void LONEDELvalue(string& value);

void LONESEARCH(string& value);

void LONEREAD();

void Clear();

};

OneList::~OneList() {

Clear();

}

struct TwoList {

~TwoList();

DoubleNode\* head = nullptr;

DoubleNode\* tail = nullptr;

void LTWOPUSHhead(string& element);

void LTWOPUSHtail(string& element);

void LTWODELhead();

void LTWODELtail();

void LTWODELvalue(string& value);

void LTWOSEARCH(string& value);

void LTWOREAD();

void Clear();

};

TwoList::~TwoList() {

Clear();

}

struct AVL {

AVLnode\* root;

AVL() : root(nullptr) {};

~AVL();

void TDestroyAVL(AVLnode\* node);

string key;

unsigned char height;

SingleNode\* left;

SingleNode\* right;

int THeight(AVLnode\* node);

int TBalance(AVLnode\* node);

AVLnode\* TRightRotate(AVLnode\* y);

AVLnode\* TLeftRotate(AVLnode\* x);

AVLnode\* TInsert(AVLnode\* node, string & key);

AVLnode\* TMinValueLeftNode(AVLnode\* node);

AVLnode\* TDeleteNode(AVLnode\* root, string& key);

AVLnode\* TSearch(AVLnode\* node, string& key);

void TDisplay(AVLnode\* node);

void TPUSH(string& key);

void TDEL(string& key);

bool TSEARCH(string& key);

void TREAD();

void TSaveToFile(AVLnode\* node, ofstream& FileWrite);

};

AVL::~AVL() {

TDestroyAVL(root);

}

struct HashTable {

NodeHash\*\* table; // массив указателей на узлы хеш-таблицы

int size;

int count;

HashTable(int initialSize = 100);

int HashFunc(string& str);

void HPUSH( string& key, string& value);

string HGET( string& key);

void HDEL( string& key);

void HREAD();

void HDestroyTable(NodeHash\* node);

~HashTable();

};

HashTable::~HashTable() {

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

HDestroyTable(table[i]);

}

delete[] table;

}

## Массив (Array.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void Array::MPUSHend(string& value) {

data[size++] = value + " ";

}

void Array::MPUSHindex(const int& index, string& value) {

for (int i = size; i > index; i--) {

data[i] = data[i - 1];

}

data[index] = value + " ";

size++;

}

string Array::MGETL(const int& index) {

return data[index];

}

void Array::MDEL(const int& index) {

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

data[i] = data[i + 1];

}

size--;

}

void Array::MREPL(const int& index, string& value) {

data[index] = + " " + value + + " ";

}

int Array::MSIZE() {

return size;

}

void Array::MREAD() {

cout << endl;

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

cout << data[i];

}

cout << endl;

}

## Очередь (Queue.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void Queue::Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void Queue::QPUSH(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (head == nullptr) {

head = node;

tail = node;

}

else {

tail->next = node;

tail = node;

}

}

void Queue::QPOP() {

SingleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

}

}

void Queue::QREAD() {

SingleNode\* current = head;

cout << endl;

while (current != nullptr) {

cout << current->cell << " ";

current = current->next;

}

cout << endl;

}

## Стек (Stack.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void Stack::Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void Stack::SPUSH(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (head == nullptr) {

head = node;

tail = node;

}

else {

tail->next = node;

tail = node;

}

}

void Stack::SPOP() {

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

SingleNode\* current = head;

while (current->next != tail) {

current = current->next;

}

delete tail;

tail = current;

tail->next = nullptr;

}

void Stack::SREAD() {

SingleNode\* current = head;

cout << endl;

while (current != nullptr) {

cout << current->cell << " ";

current = current->next;

}

cout << endl;

}

## Односвязный список (OneList.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void OneList::Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void OneList::LONEPUSHhead(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (head == nullptr) {

head = tail = node;

}

else {

node->next = head;

head = node;

}

}

void OneList::LONEPUSHtail(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (tail == nullptr) {

head = tail = node;

}

else {

tail->next = node;

tail = node;

}

}

void OneList::LONEDELhead() {

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

SingleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

}

}

void OneList::LONEDELtail() {

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

SingleNode\* current = head;

while (current->next != tail) {

current = current->next;

}

delete tail;

tail = current;

tail->next = nullptr;

}

void OneList::LONEDELvalue(string&value) {

SingleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

if (head->cell == value) {

LONEDELhead();

return;

}

while (node != nullptr && node->next != nullptr && node->next->cell != value ) {

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

node = node->next;

}

SingleNode\* toDelete = node->next;

node->next = toDelete->next;

if (toDelete == tail) {

tail = toDelete;

}

delete toDelete;

}

void OneList::LONESEARCH(string& value) {

SingleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

int index = 0;

if (head->cell == value) {

cout << "\nThe element is found! Index: " << index << endl;

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

cout << "\nThe element is found! Index: " << index << endl;

}

void OneList::LONEREAD() {

SingleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

cout << endl;

while (node != nullptr) {

cout << node->cell << " ";

node = node->next;

}

cout << endl;

}

## Двусвязный список (TwoList.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void TwoList::Clear() {/////

while (head != nullptr) {

DoubleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void TwoList:: LTWOPUSHhead(string& element) {

DoubleNode\* node = new DoubleNode;

node->cell = element;

node->next = nullptr;

node->prev = nullptr;

if (head == nullptr) {

head = tail = node;

}

else {

node->next = head;

head->prev = node;

head = node;

}

}

void TwoList::LTWOPUSHtail(string& element) {

DoubleNode\* node = new DoubleNode;

node->cell = element;

node->next = nullptr;

node->prev = nullptr;

if (tail == nullptr) {

head = tail = node;

}

else {

tail->next = node;

node->prev = tail;

tail = node;

}

}

void TwoList::LTWODELhead() {

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

DoubleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

}

}

void TwoList::LTWODELtail() {

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

DoubleNode\* current = head;

while (current->next != tail) {

current = current->next;

}

delete tail;

tail = current;

tail->next = nullptr;

tail->prev = current->prev;

}

void TwoList::LTWODELvalue(string& value) {

DoubleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

if (head->cell == value) {

LTWODELhead();

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

if(node->prev) {

node->prev->next = node->next;

}

else {

head = node->prev;

}

if (node->next) {

node->next->prev = node->prev;

}

else {

tail = node->next;

}

delete node;

}

void TwoList::LTWOSEARCH(string& value) {

DoubleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

int index = 0;

if (head->cell == value) {

cout << "\nThe element is found! Index: " << index << endl;

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

cout << "\nThe element is found! Index: " << index << endl;

}

void TwoList::LTWOREAD() {

DoubleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

cout << endl;

while (node != nullptr) {

cout << node->cell << " ";

node = node->next;

}

cout << endl; }

## AVL-дерево (AVLtree.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void AVL::TDestroyAVL(AVLnode\* node) {

if (node == nullptr) {

return;

}

TDestroyAVL(node->left);

TDestroyAVL(node->right);

delete node;

}

int AVL::THeight(AVLnode\* node) {

if (node == nullptr) {

return 0;

}

return node->height;

}

int AVL::TBalance(AVLnode\* node) {

if (node == nullptr) return 0;

return THeight(node->left) - THeight(node->right);

}

AVLnode\* AVL::TRightRotate(AVLnode\* node) {

AVLnode\* A = node->left; // левый потомок узла становится А

AVLnode\* B = A->right; // правый потомок узла А становится В

A->right = node; // А становится корнем, ноде становится правым потомком

node->left = B; // левый потомок ноде становится В

node->height = max(THeight(node->left), THeight(node->right)) + 1; // ноде и А становятся двумя разными поддеревьями

A->height = max(THeight(A->left), THeight(A->right)) + 1;

return A; // корень

}

AVLnode\* AVL::TLeftRotate(AVLnode\* node) { //симметрия

AVLnode\* A = node->right;

AVLnode\* B = A->left;

A->left = node;

node->right = B;

node->height = max(THeight(node->left), THeight(node->right)) + 1;

A->height = max(THeight(A->left), THeight(A->right)) + 1;

return A;

}

AVLnode\* AVL::TInsert(AVLnode\* node, string&key) {

// если текущий узел пустой, создаем новый узел с переданным ключом

if (node == nullptr) {

return new AVLnode(key);

}

//рекурсивно вставляем в левое поддерево

if (key < node->key) {

node->left = TInsert(node->left, key);

}

// рекурсивно вставляем в правое поддерево

else if (key > node->key) {

node->right = TInsert(node->right, key);

}

// если ключ уже существует, возвращаем текущий узел

else {

return node; //(без дубликатов)

}

node->height = 1 + max(THeight(node->left), THeight(node->right)); // обновляем высоту текущего узла

// вычисляем баланстекущего узла

int balance = TBalance(node);

// левый-левый

if (balance > 1 && key < node->left->key)

return TRightRotate(node);

// левый-правый

if (balance > 1 && key > node->left->key) {

node->left = TLeftRotate(node->left); //меняет местами

return TRightRotate(node);

}

// правый-правый

if (balance < -1 && key > node->right->key)

return TLeftRotate(node);

// правый-левый

if (balance < -1 && key < node->right->key) {

node->right = TRightRotate(node->right); // меняет местами

return TLeftRotate(node);

}

return node; // возвращаем корень поддерева

}

AVLnode\* AVL::TMinValueLeftNode(AVLnode\* node) {

AVLnode\* current = node;

while (current->left != nullptr) // идем по левому поддереву до самого левого узла

current = current->left;

return current;

}

AVLnode\* AVL::TDeleteNode(AVLnode\* root, string &key) {

if (root == nullptr) {

return root; // если дерево пустое

}

// рекурсивно удаляем в левом поддереве

if (key < root->key)

root->left = TDeleteNode(root->left, key);

// рекурсивно удаляем в правом поддереве

else if (key > root->key)

root->right = TDeleteNode(root->right, key);

// если ключ совпадает с ключом текущего узла, удаляем текущий узел

else {

// если у текущего узла нет одного из потомков или оба потомка отсутствуют

if (root->left == nullptr || root->right == nullptr) {

// выбираем непустого потомка или nullptr, если оба потомка отсутствуют

AVLnode\* temp = root->left ? root->left : root->right;

// если непустой потомок отсутствует, удаляем текущий узел

if (temp == nullptr) {

temp = root; //переходим к этому потомку

root = nullptr; //обнуляем его

}

// если есть непустой потомок, копируем его данные в текущий узел

else

\*root = \*temp;

delete temp; // освобождаем память удаляемого узла

}

// если у текущего узла есть оба потомка

else {

AVLnode\* temp = TMinValueLeftNode(root->right); // находим узел с минимальным ключом в правом поддереве

root->key = temp->key; // копируем ключ минимального узла в текущий узел

root->right = TDeleteNode(root->right, temp->key); // рекурсивно удаляем минимальный узел из правого поддерева

}

}

// если дерево стало пустым после удаления, возвращаем его

if (root == nullptr) {

return root;

}

// обновляем высоту текущего узла

root->height = 1 + max(THeight(root->left), THeight(root->right));

// вычисляем баланс текущего узла

int balance = TBalance(root);

// левый-левый случай: баланс-фактор > 1 и баланс-фактор левого поддерева >= 0

if (balance > 1 && TBalance(root->left) >= 0)

return TRightRotate(root);

// левый-правый случай: баланс-фактор > 1 и баланс-фактор левого поддерева < 0

if (balance > 1 && TBalance(root->left) < 0) {

root->left = TLeftRotate(root->left);

return TRightRotate(root);

}

// правый-правый случай: баланс-фактор < -1 и баланс-фактор правого поддерева <= 0

if (balance < -1 && TBalance(root->right) <= 0)

return TLeftRotate(root);

// правый-левый случай: баланс-фактор < -1 и баланс-фактор правого поддерева > 0

if (balance < -1 && TBalance(root->right) > 0) {

root->right = TRightRotate(root->right);

return TLeftRotate(root);

}

return root; // возвращаем корень поддерева

}

AVLnode\* AVL::TSearch(AVLnode\* node, string & key) {

if (node == nullptr || node->key == key) {// если узел пустой или ключ найден, возвращаем узел

return node;

}

if (key < node->key)

return TSearch(node->left, key);

else

return TSearch(node->right, key);

}

void AVL::TDisplay(AVLnode\* node) {

if (node != nullptr) {

cout << node->key << " "; // выводим ключ текущего узла

TDisplay(node->left); // рекурсивно обходим левое поддерево

TDisplay(node->right); // рекурсивно обходим правое поддерево

}

}

void AVL::TSaveToFile(AVLnode\* node, ofstream& FileWrite) {

if (node == nullptr) {

return;

}

FileWrite << node->key << " "; // записываем ключ текущего узла

// рекурсивно записываем левое и правое поддеревья

TSaveToFile(node->left, FileWrite);

TSaveToFile(node->right, FileWrite);

}

void AVL::TPUSH(string &key) {

root = TInsert(root, key);

}

void AVL::TDEL(string& key) {

root = TDeleteNode(root, key);

}

bool AVL::TSEARCH(string&key) {

return TSearch(root, key) != nullptr;

}

void AVL::TREAD() {

TDisplay(root);

cout << endl;

}

## Хэш-таблица (HashTable.cpp)

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

void HashTable::HDestroyTable(NodeHash\* node) {

while (node != nullptr) {

NodeHash\* temp = node;

node = node->next;

delete temp;

}

}

HashTable::HashTable(int initialSize) {

size = initialSize;

count = 0;

table = new NodeHash \* [size];

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

table[i] = nullptr;

}

}

int HashTable::HashFunc( string& str) {

int hash = 0;

for (char c : str) {

hash += c; //ASCII

}

return hash % size;

}

void HashTable::HPUSH( string& key, string& value) {

int index = HashFunc(key);

NodeHash\* current = table[index];

while (current != nullptr) {

if (current->key == key) {

current->value = value;

return;

}

current = current->next;

}

NodeHash\* newNode = new NodeHash(key, value); // создание нового узла

newNode->next = table[index]; // новый узел указывает на текущий узел по индексу

table[index] = newNode; // новый узел становится текущим узлом по индексу

count++;

}

void HashTable::HDEL( string& key) {

int index = HashFunc(key);

NodeHash\* current = table[index];

NodeHash\* prev = nullptr;

while (current != nullptr) {

if (current->key == key) {

if (prev == nullptr) { // если это первый узел в списке

table[index] = current->next; // следующий узел становится текущим

}

else {

prev->next = current->next; // предыдущий узел указывает на следующий узел

}

delete current;

count--;

return;

}

prev = current; // переход к следующему узлу

current = current->next;

}

}

string HashTable::HGET(string& key) {

int index = HashFunc(key);

NodeHash\* current = table[index];

while (current != nullptr) {

if (current->key == key) { // если ключ найден

return current->value;

}

current = current->next;

}

return "\n\nThe key was not found\n\n";

}

void HashTable::HREAD() {

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

cout << "[" << i << "]: ";

NodeHash\* current = table[i];

if (current == nullptr) {

cout << "empty" << endl;

continue;

}

while (current != nullptr) {

cout << "(" << current->key << ", " << current->value << ") ";

current = current->next;

}

cout << endl;

}

}

## Перечень операций (INFO.txt)

----------------КОМАНДЫ----------------

**\*\*\* МАССИВ \*\*\***

MPUSH <название файла> <элемент> - добавление элемента в конец

MPUSH <название файла> <элемент> <индекс> - добавление элемента по индексу

MDEL <название файла> <индекс> - удаление элемента по индексу

MGETL <название файла> <индекс> - получение элемента по индексу

MREPL <название файла> <индекс> <элемент> - замена элемента по индексу

MSIZE <название файла> - получение длины массива

MREAD <название файла> - вывод элементов

**\*\*\* ОЧЕРЕДЬ \*\*\***

QPUSH <название файла> <элемент> - добавление элемента в конец

QPOP <название файла> - удаление первого элемента

QREAD <название файла> - вывод элементов

**\*\*\* СТЕК \*\*\***

SPUSH <название файла> <элемент> - добавление элемента в конец

SPOP <название файла> - удаление последнего элемента

SREAD <название файла> - вывод элементов

**\*\*\* ОДНОСВЯЗНЫЙ СПИСОК \*\*\***

LONEPUSHhead <название файла> <элемент> - добавление элемента в начало списка

LONEPUSHtail <название файла> <элемент> - добавление элемента в конец списка

LONEDELhead <название файла> - удаление первого элемента из списка

LONEDELtail <название файла> - удаление последнего элемента из списка

LONEDELvalue <название файла> <элемент> - удаление элемента с заданным значением из списка

LONESEARCH <название файла> <элемент> - поиск элемента в списке по значению, выводит индекс найденного элемента

LONEREAD <название файла> - вывод всех элементов списка на экран

**\*\*\* ДВУСВЯЗНЫЙ СПИСОК \*\*\***

LTWOPUSHhead <название файла> <элемент> - добавление элемента в начало двусвязного списка

LTWOPUSHtail <название файла> <элемент> - добавление элемента в конец двусвязного списка

LTWODELhead <название файла> - удаление первого элемента из двусвязного списка

LTWODELtail <название файла> - удаление последнего элемента из двусвязного списка

LTWODELvalue <название файла> <элемент> - удаление элемента с заданным значением из двусвязного списка

LTWOSEARCH <название файла> <элемент> - поиск элемента в двусвязном списке по значению, выводит индекс

LTWOREAD <название файла> - вывод всех элементов двусвязного списка на экран

**\*\*\* AVL-ДЕРЕВО \*\*\***

TPUSH <название файла> <элемент> - добавление элемента

TDEL <название файла> <элемент> - удаление элемента

TSEARCH <название файла> <элемент> - поиск элемента (вывод Да/Нет)

TREAD <название файла> - вывод всех элементов на экран

**\*\*\* ХЭШ-ТАБЛИЦА \*\*\***

HPUSH <название файла> <ключ> <значение> - добавление элемента

HGET <название файла> <ключ> - получение значения по ключу

HDEL <название файла> <ключ> - удаление ячейки по ключу

HREAD <название файла> - вывод всех элементов на экран

-------------------------------------

# Функциональная характеристика программы

## Примеры работы программы

1. Массив

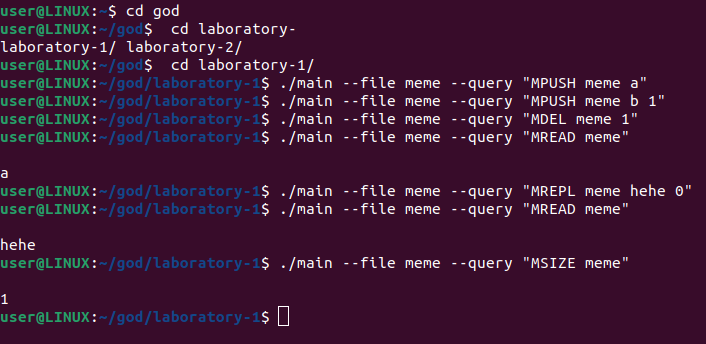


Рисунок 1. Пример работы программы для массива.

1. Очередь

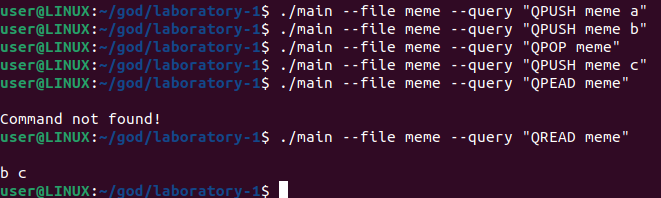


Рисунок 2. Пример работы программы для очереди.

1. Стек

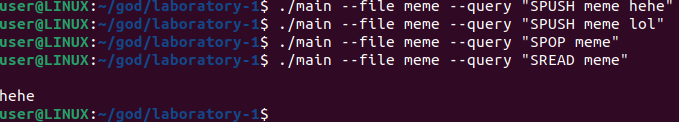


Рисунок 3. Пример работы программы для стека.

1. Односвязный список

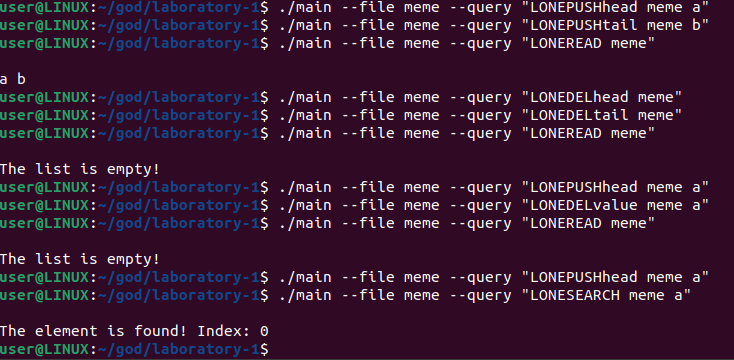


Рисунок 4. Пример работы программы для односвязного списка.

1. Двусвязный список

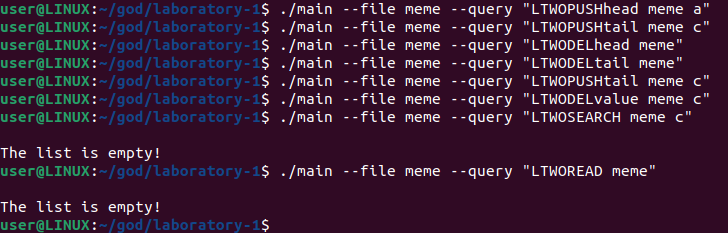


Рисунок 5. Пример работы программы для двусвязного списка.

1. AVL-дерево

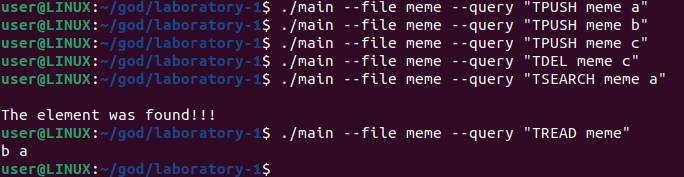


Рисунок 6. Пример работы программы для AVL-дерева.

1. Хэш-таблица

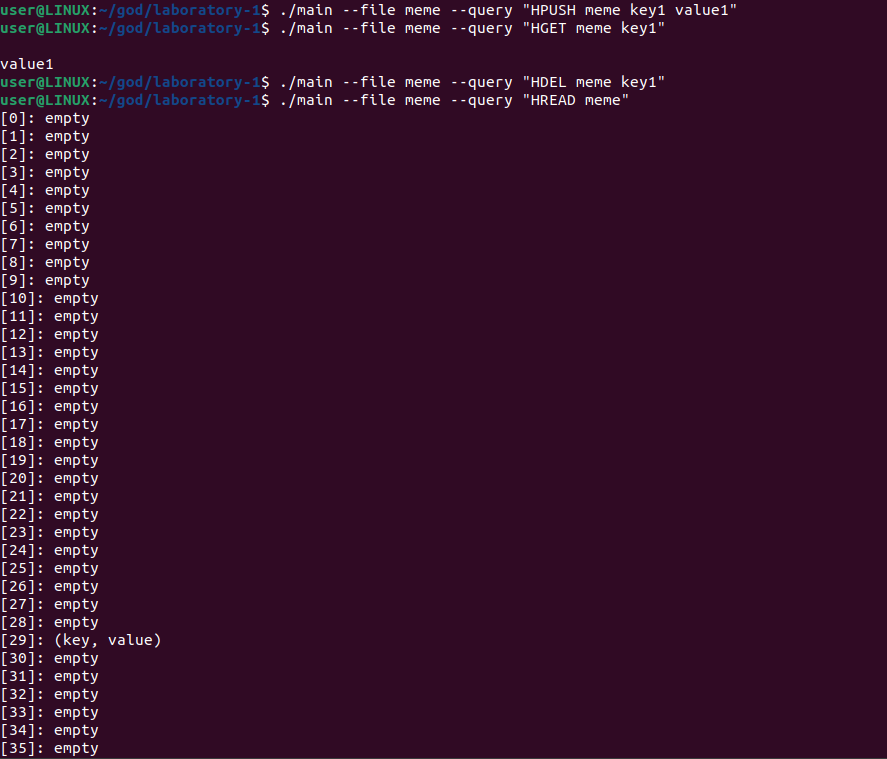


Рисунок 7. Пример работы программы для хэш-таблицы (начало).

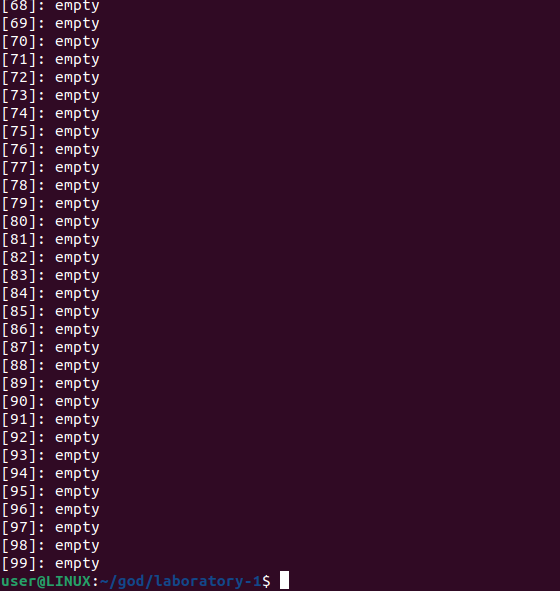


Рисунок 8. Пример работы программы для хэш-таблицы (конец).

## Схемы работы программы

1. Массив.

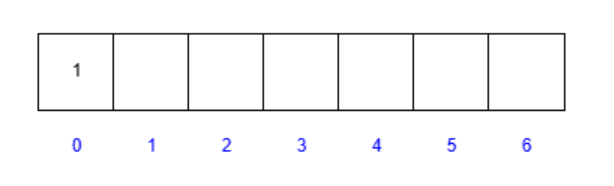
****

Рисунок 1. Массив после MPUSH 1.

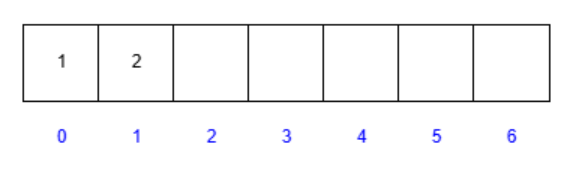
****

Рисунок 2. Массив после MPUSH 2 1.

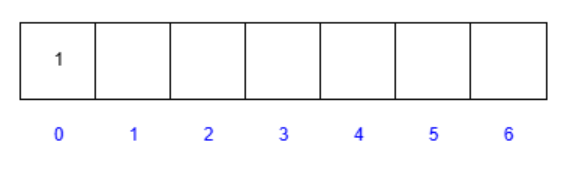
****

Рисунок 3. Массив после MDEL 1.

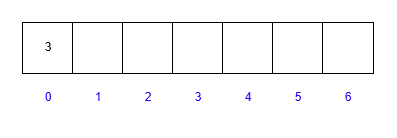


Рисунок 4. Массив после MREPL 3 0.

1. Очередь.

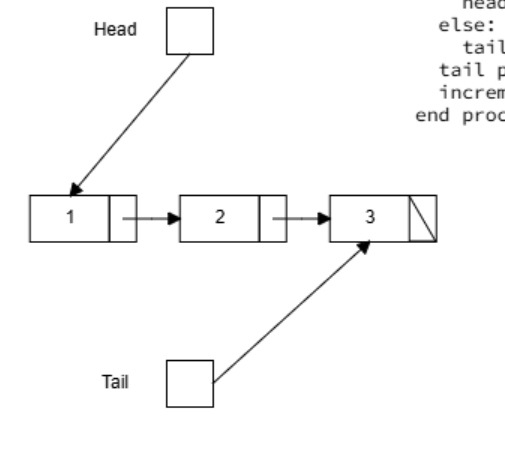


Рисунок 5. Очередь после QPUSH 1; QPUSH 2; QPUSH 3.

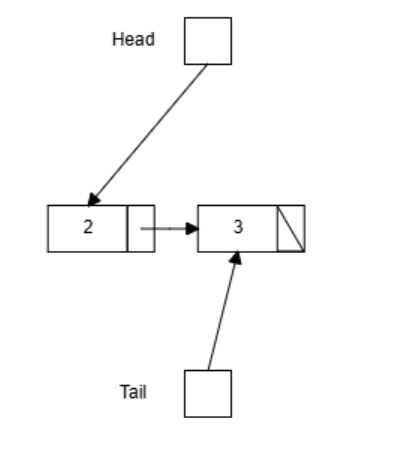


Рисунок 6. Очередь после QPOP.

1. Стек.

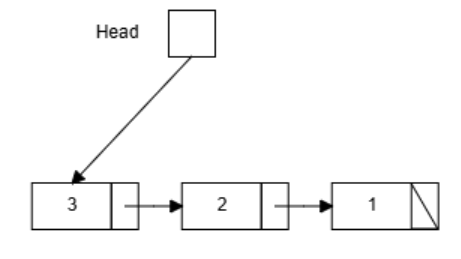


Рисунок 7. Стек после SPUSH 1; SPUSH 2; SPUSH 3.

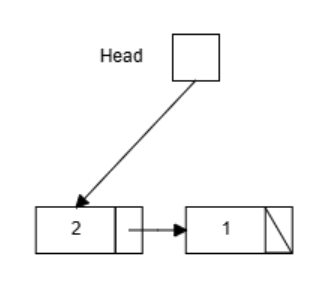


Рисунок 8. Стек после SPOP.

1. Односвязный список.

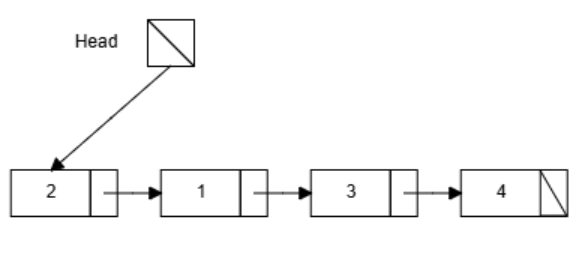


Рисунок 9. Односвязный список после LONEPUSHhead 4; LONEPUSHhead 3; LONEPUSHhead 1; LONEPUSHhead 2.

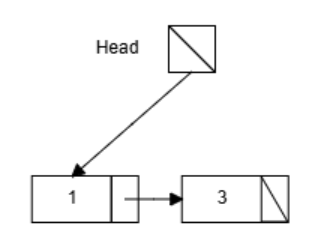


Рисунок 10. Односвязный список после LONEDELhead; LONEDELtail

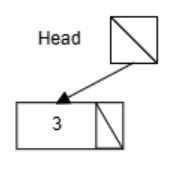


Рисунок 11. Односвязный список после LONEDELvalue 1.

1. Двусвязный список.

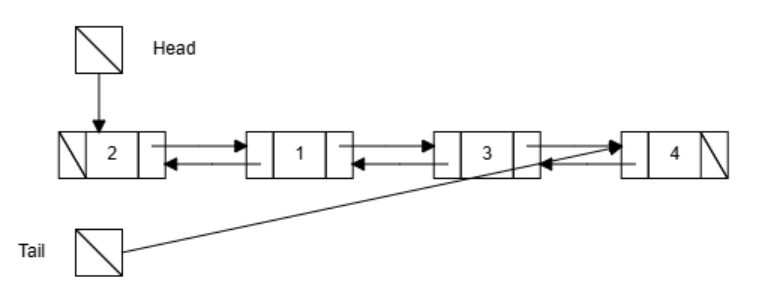


Рисунок 12. Двусвязный список после LTWOPUSHhead 4; LTWOPUSHhead 3; LTWOPUSHhead 1; LTWOPUSHhead 2.

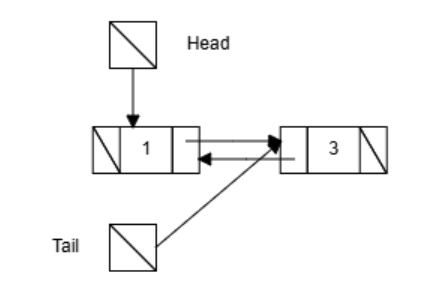


Рисунок 13. Двусвязный список после LTWODELhead; LTWODELtail.

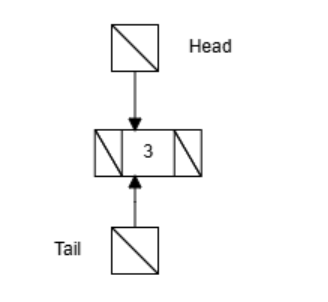


Рисунок 14. Двусвязный список после LTWODELvalue 1.

1. AVL-дерево.

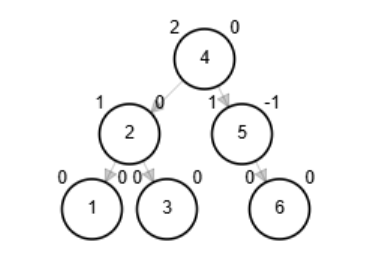


Рисунок 15. АВЛ-дерево после TPUSH 1; … ; TPUSH 6.

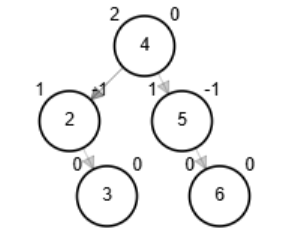


Рисунок 16. АВЛ-дерево после TDEL 1.

1. Хэш-таблица.

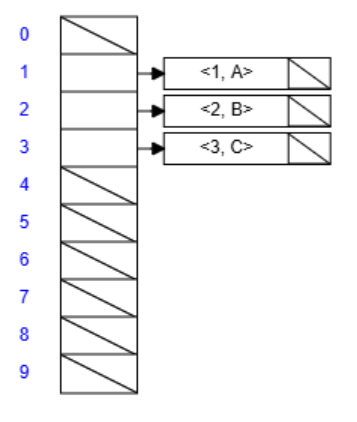


Рисунок 17. Хеш-таблица после HPUSH 1 A; HPUSH 2 B; HPUSH 3 C

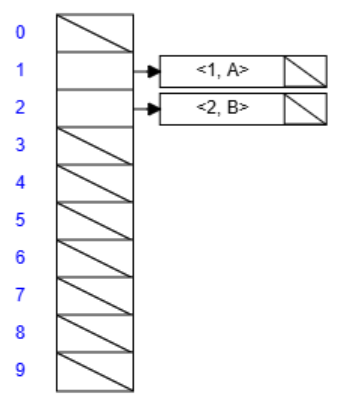


Рисунок 18. Хеш-таблица после HDEL 3

## Сложность выполнения каждой операции с позиции BigO нотации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип контейнера | Добавление | Удаление | Получение  (Поиск) | Замена | Чтение |
| Массив (M) | O(1)  O(n) | O(n) | O(1) | O(1) | O(n) |
| Односвязный список (L) | O(1)  O(1) | O(1)  O(n)  O(n) | O(n) | - | O(n) |
| Двусвязный список (D) | O(1)  O(1) | O(1)  O(1)  O(n) | O(n) | - | O(n) |
| Очередь (Q) | O(1) | O(1) | - | - | O(n) |
| Стек (S) | O(1) | O(1) | - | - | O(n) |
| Хеш-таблица (H) | O(1)-O(n) | O(1)-O(n) | O(1)-O(n) | - | O(n) |
| АВЛ-дерево (T) | O(log n) | O(log n) | O(log n) | - | O(n) |

Таблица 1. Сложность выполнения каждой операции с позиции BigO ноотации.

# Ссылки на репозиторий

## GitHub

https://github.com/mamam111a/laboratory-1