МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ №2**

**«**Алгоритмы и абстрактные структуры данных**»**

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

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Мартынова Е.Д.*  «13» ноября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г.А.*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

Оглавление

[Цели и задачи работы 3](#_Toc182439895)

[Задание к работе 3](#_Toc182439896)

[Методика выполнения работы: 3](#_Toc182439897)

[Задание 1 3](#_Toc182439898)

[Задание 2 3](#_Toc182439899)

[Задание 3 4](#_Toc182439900)

[Задание 4 4](#_Toc182439901)

[Задание 5 4](#_Toc182439902)

[Задание 6 4](#_Toc182439903)

[Структура программы 5](#_Toc182439904)

[main.cpp 5](#_Toc182439905)

[header.h 11](#_Toc182439906)

[HashTable.cpp 14](#_Toc182439907)

[Tree.cpp 17](#_Toc182439908)

[Array.cpp 19](#_Toc182439909)

[Set.cpp 21](#_Toc182439910)

[Stack.cpp 25](#_Toc182439911)

[Функциональная характеристика программы 27](#_Toc182439912)

[Примеры работы программы 27](#_Toc182439913)

[Схемы работы программы 29](#_Toc182439914)

[Сложность использования структур с позиции BigO нотации 32](#_Toc182439915)

[Ссылки на репозитории 33](#_Toc182439916)

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

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

# 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.

Задание 1

Задано числовое выражение. Необходимо вычислить его значение или установить, что оно содержит ошибку. В выражении могут встречаться знаки сложения, вычитания, умножения, скобки и пробелы (пробелов внутри чисел быть не должно). Приоритет операций стандартный. Все числа в выражении целые и по модулю не превосходят 2×10^9. Также гарантируется, что все промежуточные вычисления также не превосходят 2×10^9.

# Задание 2

Реализовать основные операции со множеством: добавление элемента, удаление элемента, проверка наличия элемента в множестве.

*Таблица операций*

|  |  |  |  |
| --- | --- | --- | --- |
| Тип контейнера | Добавление | Удаление | Проверка, является ли элемент частью  множества |
| Множество | SETADD | SETDEL | SET\_AT |

# Задание 3

Необходимо реализовать алгоритм, который должен разбить множество натуральных чисел на непересекающиеся подмножества, разница между суммами которых была бы минимальна. Вывести получившиеся подмножества и разницу их сумм.

Пример:

множество S = {5, 8, 1, 14, 7}.

Получим два подмножества {5, 14} и {8, 1, 7}.

Разница между их суммами - 3.

# Задание 4

Необходимо реализовать алгоритм, который находит подмассив, сумма элементов которого равна заданному числу.

Пример: Массив [4, -7, 1, 5, -4, 0, -3, 2, 4, 1], цель 5.

Подмассивы: [5] [4, 1] [5, -4, 0, -3, 2, 4, 1] [1, 5, -4, 0, -3, 2, 4]

# Задание 5

Дано бинарное дерево поиска, в котором хранятся уникальные целые числа. Найдите вершину с заданным значением и удалите её из дерева так, чтобы дерево осталось корректным бинарным деревом поиска.

# Задание 6

Необходимо реализовать алгоритм, который определяет, являются ли строки изоморфными. Две строки a и b считаются изоморфными, если символы в a можно заменить на b. Символ сопоставляется только одному другому, порядок символов должен сохраняться. Пример: "fall" и "redd" изоморфны. "mad" и "odd" не изоморфны.

# Структура программы

## main.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

#include "header.h"

#include "HashTable.cpp"

#include "Tree.cpp"

#include "Array.cpp"

#include "Set.cpp"

#include "Stack.cpp"

int main()

{

cout << endl << "HELLO!" << endl;

while (true) {

Stack stackOper;

Stack stackNumb;

SetNode set(15);

Array array;

Tree tree;

cout << endl << "Select a task option -->> ";

string numberTask;

cin >> numberTask;

if (numberTask == "EXIT") {

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

break;

}

else if (numberTask == "1") {

cout << endl << "LETS GO -->> ";

cin.ignore();

string expression;

getline(cin, expression);

string expressionTemp = expression;

stringstream ss(expression);

string part;

int countBrackets = 0;;

int countOperators = 0;

int countNumbers = 0;

bool isWrong = false;

string prev = "";

while (getline(ss, part, ' ')) {

if (part == "(") {

if (prev != "oper" && prev != "" && prev != "bracket") {

isWrong = true;

break;

}

prev = "bracket";

countBrackets++;

}

else if (part == ")") {

if (prev != "number" && prev != "bracket") {

isWrong = true;

break;

}

prev = "bracket";

countBrackets++;

}

else if (part == "+" || part == "-" || part == "\*") {

if (prev != "number" && prev != "bracket") {

isWrong = true;

break;

}

prev = "oper";

countOperators++;

}

else if (isdigit(part.at(0)) || part.at(0) == '-') {

if (part.at(0) == '0') {

isWrong = true;

break;

}

int i = 1;

while (i != part.size()) {

if (isdigit(part.at(i)) == false) {

isWrong = true;

break;

}

i++;

}

if (prev != "oper" && prev != "bracket") {

if (prev != "") {

isWrong = true;

break;

}

}

prev = "number";

countNumbers++;

}

else {

isWrong = true;

break;

}

}

if (isWrong) {

cout << endl << "Incorrect input!!!" << endl;

continue;

}

if (countBrackets % 2 != 0 || countNumbers - countOperators != 1) {

cout << endl << "Incorrect input!!!" << endl;

continue;

}

stringstream sss(expressionTemp);

string current = "";

while (getline(sss, current, ' ')) {

if (current == "+" || current == "-" || current == "\*") {

if (stackOper.tail == nullptr) {

stackOper.AddTail(current);

continue;

}

if (current == "+" || current == "-") {

if (stackOper.tail->cell == "\*") {

ProcessingExpression("\*", stackOper, stackNumb, current, 0);

}

else {

if (stackOper.tail->cell == "+" && current == "+") {

ProcessingExpression("+", stackOper, stackNumb, current, 0);

}

else {

stackOper.AddTail(current);

continue;

}

}

}

else if (current == "\*") {

if (stackOper.tail->cell == "\*") {

ProcessingExpression("\*", stackOper, stackNumb, current, 0);

}

else {

stackOper.AddTail(current);

}

}

}

else if (current == "(") {

stackOper.AddTail(current);

}

else if (current.at(0) == '-') {

if (stackOper.tail != nullptr) {

if (stackOper.tail->cell == "-") {

int temp = stoi(current);

temp = temp \* (-1);

string strTemp = to\_string(temp);

stackNumb.AddTail(strTemp);

stackOper.tail->cell = "+";

}

}

stackNumb.AddTail(current);

continue;

}

else if (isdigit(current.at(0))) {

if (stackOper.tail != nullptr) {

if (stackOper.tail->cell == "-") {

int temp = stoi(current);

temp = temp \* (-1);

string strTemp = to\_string(temp);

stackNumb.AddTail(strTemp);

stackOper.tail->cell = "+";

}

else {

stackNumb.AddTail(current);

}

}

else {

stackNumb.AddTail(current);

}

}

else {

while (stackOper.tail->cell != "(") {

string tempOper = stackOper.tail->cell;

int B = stoi(stackNumb.tail->cell);

stackNumb.DeleteTail();

int A = stoi(stackNumb.tail->cell);

stackNumb.DeleteTail();

int result = 0;

if (tempOper == "\*") result = A \* B;

else if (tempOper == "+") result = A + B;

else if (tempOper == "-") result = A - B;

string temp = to\_string(result);

stackNumb.AddTail(temp);

stackOper.DeleteTail();

}

stackOper.DeleteTail();

if (stackOper.tail != nullptr) {

if (stackOper.tail->cell == "-") {

int temp = stoi(stackNumb.tail->cell);

temp = temp \* (-1);

string strTemp = to\_string(temp);

stackNumb.tail->cell = strTemp;

stackOper.tail->cell = "+";

}

}

}

}

while (stackOper.head != nullptr) {

if (stackOper.head->next != nullptr) {

if ((stackOper.head->cell == "-" && stackOper.head->next->cell == "\*") || (

(stackOper.head->cell == "+" && stackOper.head->next->cell == "\*"))) {

string oper = stackOper.head->next->cell;

int b = stoi(stackNumb.head->next->next->cell);

int a = stoi(stackNumb.head->next->cell);

int result;

result = a \* b;

string resultStr = to\_string(result);

stackNumb.DelElementValue(stackNumb.head->next);

stackOper.DelElementValue(stackOper.head->next);

stackNumb.head->next->cell = resultStr;

}

}

string oper = stackOper.head->cell;

stackOper.SPOP();

int b = stoi(stackNumb.head->next->cell);

int a = stoi(stackNumb.head->cell);

stackNumb.SPOP();

stackNumb.SPOP();

int result;

if (oper == "+") result = a + b;

else if (oper == "-") result = a - b;

else if (oper == "\*") result = a \* b;

string resultStr = to\_string(result);

stackNumb.SPUSH(resultStr);//////////

}

cout << endl << "RESULT: " << stackNumb.tail->cell << endl;

}

else if (numberTask == "2") {

string command;

string commandWord;

cout << endl << "Enter the command -->> ";

cin.ignore();

getline(cin, command);

stringstream ss(command);

string fileName;

string element;

ss >> commandWord;

ss >> fileName;

ss >> element;

ifstream FileRead(fileName + ".txt");

string token;

while (getline(FileRead, token, ' ')) {

set.SETADD(token);

}

FileRead.close();

if (commandWord == "SETADD") {

set.SETADD(element);

}

else if (commandWord == "SETDEL") {

set.SETDEL(element);

}

else if (commandWord == "SET\_AT") {

set.SET\_AT(element);

continue;

}

else {

cout << endl << "INCORRECT INPUT!!!" << endl;

continue;

}

ofstream FileWrite(fileName + ".txt");

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

if (set.plenty[i] != nullptr && !set.plenty[i]->value.empty()) {

FileWrite << set.plenty[i]->value << " ";

}

}

FileWrite.close();

}

else if (numberTask == "3") {

cout << endl << "WARNING!!! The set should not contain duplicate elements!!!" << endl;

cout << endl << "LETS GO -->> ";

cin.ignore();

string elements;

getline(cin, elements);

string temp;

stringstream ss(elements);

while (getline(ss, temp, ' ')) {

set.SETADD(temp);

}

FindSubsets(set);

}

else if (numberTask == "4") {

cout << endl << "LETS GO -->> ";

cin.ignore();

string elements;

string sumStr;

getline(cin, elements);

string temp;

stringstream ss(elements);

cout << endl << "Sum -> ";

cin >> sumStr;

int sum = stoi(sumStr);

while (getline(ss, temp, ' ')) {

array.MPUSHend(temp);

}

cout << endl;

Array sums;

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

string subArray = array.data[i];

string tempArray1;

stringstream ss1(subArray);

int sumValues1 = 0;

while(getline(ss1, tempArray1, ' ')) {

sumValues1 += stoi(tempArray1);

}

if(sumValues1 == sum) {

sums.MPUSHend(subArray);

}

for(int j = i+1; j < array.size; j++) {

subArray+= " " + array.data[j];

string tempArray2;

stringstream ss2 (subArray);

int sumValues2 = 0;

while(getline(ss2, tempArray2, ' ')) {

sumValues2 += stoi(tempArray2);

}

if(sumValues2 == sum) {

sums.MPUSHend(subArray);

}

}

}

sums.MREAD();

}

else if (numberTask == "5") {

tree.Insert(15);

tree.Insert(17);

tree.Insert(23);

tree.Insert(13);

tree.Insert(19);

tree.Insert(135);

tree.Insert(5);

tree.Print("", tree.root, false);

int key;

cout << endl << "Choose the element to delete -->> ";

cin.ignore();

cin >> key;

tree.DeleteNode(tree.root, key);

cout << endl;

tree.Print("", tree.root, false);

}

else if (numberTask == "6") {

string elements;

string strA;

string strB;

cout << endl << "LETS GO -->> ";

cin >> strA >> strB;

if (isIsomorphic(strA, strB)) {

cout << endl << "They are isomorphic!!!" << endl;

}

else {

cout << endl << "They are NOT isomorphic!!!" << endl;

}

}

else if (numberTask == "EXIT") {

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

return 0;

}

else {

cout << endl << "INCORRECT INPUT!!!" << endl;

continue;

}

}

}

## header.h

#pragma once

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

struct SingleNode {

string cell;

SingleNode\* next;

};

struct Set {

string value;

};

struct TreeNode {

int value;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

struct NodeHash {

string key;

string value;

NodeHash\* next;

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

};

struct Stack {

int size;

string cell;

~Stack();

SingleNode\* head;

SingleNode\* tail;

void SPUSH(string& cell);

void SPOP();

void SREAD();

void Clear();

void DeleteTail();

void DelElementValue(SingleNode\* target);

void AddTail(string& cell);

Stack() : size(0), head(nullptr), tail(nullptr) {

}

};

Stack::~Stack() {

Clear();

}

struct SetNode {

int size;

Set\*\* plenty;

SetNode(int cap);

~SetNode();

int HashSet(string itemKey);

bool isFull();

bool FindValue(string value, int& znach);

void SETADD(string value);

void SETDEL(string value);

void SET\_AT(string value);

};

SetNode::SetNode(int cap) : size(cap) {

plenty = new Set \* [size];

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

plenty[i] = nullptr;

}

}

SetNode::~SetNode() {

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

delete plenty[i];

}

delete[] plenty;

}

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(100) {

data = new string[maxSize];

}

Array::~Array() {

delete[] data;

}

struct Tree {

TreeNode\* root;

Tree() : root(nullptr) {};

~Tree();

void Destroy(TreeNode\* node);

TreeNode\* FindMin(TreeNode\* node);

TreeNode\* DeleteNode(TreeNode\* root, int key);

TreeNode\* InsertRecursive(TreeNode\* node, int value);

void Insert(int value);

void Print(const string& prefix, TreeNode\* node, bool isLeft);

};

Tree::~Tree() {

Destroy(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;

}

void ProcessingExpression(const string& oper, Stack& stackOper, Stack& stackNumb, string current);

void findSumsSet(SetNode& set, int index, int currentSum, string currentElements, SetNode& sums);

void FindSubsets(SetNode& set);

## HashTable.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

#include "header.h"

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 "";

}

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;

}

}

bool isIsomorphic(string& A,string& B) {

// если разной длины

if (A.size() != B.size()) {

return false;

}

HashTable matchA\_B(100);

HashTable matchB\_A(100);

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

char charA = A[i];

char charB = B[i];

string keyA(1, charA);

string keyB(1, charB);

string accord\_B = matchA\_B.HGET(keyA);

if (!accord\_B.empty() && accord\_B != keyB) {

return false;

}

if (accord\_B.empty()) {

matchA\_B.HPUSH(keyA, keyB); //новое соответствие

}

string accord\_A = matchB\_A.HGET(keyB);

if (!accord\_A.empty() && accord\_A != keyA) {

return false;

}

if (accord\_A.empty()) {

matchB\_A.HPUSH(keyB, keyA); //новое соответствие

}

}

return true;

}

## Tree.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

#include "header.h"

void Tree::Destroy(TreeNode\* node) {

if (node == nullptr) {

return;

}

Destroy(node->left);

Destroy(node->right);

delete node;

}

TreeNode\* Tree::FindMin(TreeNode\* node) {

while (node && node->left) {

node = node->left;

}

return node;

}

void Tree::Insert(int value) {

root = InsertRecursive(root, value);

}

TreeNode\* Tree::InsertRecursive(TreeNode\* node, int value) {

if (node == nullptr) {

return new TreeNode(value);

}

if (value < node->value) {

node->left = InsertRecursive(node->left, value);

}

else if (value > node->value) {

node->right = InsertRecursive(node->right, value);

}

return node;

}

TreeNode\* Tree::DeleteNode(TreeNode\* root, int key) {

if (!root) {

return root;

}

// идем влево

if (key < root->value) {

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

}

// идем вправо

else if (key > root->value) {

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

}

else {// найден узел для удаления

// нет правого поддерева

if (!root->right) {

TreeNode\* temp = root->left;

delete root;

return temp;

}

// нет левого поддерева

else if (!root->left) {

TreeNode\* temp = root->right;

delete root;

return temp;

}

// есть оба поддерева

else {

TreeNode\* temp = FindMin(root->right); //находим мин узел в правом поддереве

root->value = temp->value; //копируем его значение и удаляем

root->right = DeleteNode(root->right, temp->value);

}

}

return root;

}

void Tree::Print(const string& prefix,TreeNode\* node, bool isLeft) {

if (node != nullptr)

{

cout << prefix;

cout << (isLeft ? "|--" : "L--");

cout << node->value << endl;

Print(prefix + (isLeft ? "| " : " "), node->right, true);

Print(prefix + (isLeft ? "| " : " "), node->left, false);

}

}

## 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) {

if (index < 0 || index >= size ) {

cout << endl << "INCORRECT INPUT!!!" << endl; //

return;

}

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

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

}

data[index] = value + " ";

size++;

}

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

if (index < 0 || index >= size ) {

cout << endl << "INCORRECT INPUT!!!" << endl; //

return "ERROR";

}

return data[index];

}

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

if (index < 0 || index >= size ) {

cout << endl << "INCORRECT INPUT!!!" << endl; //

return;

}

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

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

}

size--;

}

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

if (index < 0 || index >= size ) {

cout << endl << "INCORRECT INPUT!!!" << endl; //

return;

}

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;

}

## Set.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#include <cmath>

using namespace std;

#include "header.h"

int SetNode::HashSet(string itemKey) {

int result = 0;

for (char i : itemKey) {

result += static\_cast<int>(i);

}

return abs(result) % size;

}

bool SetNode::isFull() {

int count = 0;

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

if (plenty[i] != nullptr) {

count++;

}

}

return count >= size;

}

bool SetNode::FindValue(string value, int &result) {

int index = HashSet(value);

int newIndex = index;

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

if (plenty[index] == nullptr) {// элемент не найден

result = -1;

return false;

}

if (plenty[index]->value == value) { //найден

result = index;

return true;

}

// если не найден, пробуем следующую ячейку с помощью второй хеш-функции

index = (newIndex + i \* HashSet(value)) % size;

}

result = -1;

return false;

}

void SetNode::SETADD(string value) {

int index = HashSet(value);

// наличие уже ТАКОГО же значения

if (plenty[index] != nullptr && plenty[index]->value == value) {

return;

}

// наличие уже ТАКОГО ключа, если колизия его передвинула

int result = 0;

bool isExist = FindValue(value, result);

if (isExist) {

return;

}

if (isFull()) {

cout << endl << "The set is crowded!!!" << endl;

return;

}

//если есть место - используем

if (plenty[index] == nullptr) {

plenty[index] = new Set{value};

}

else { //иначе ищем другое свободное место

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

// Используем метод пробирования с другой хеш-функцией или методом

int newIndex = (index + i \* HashSet(value)) % size; // Пробирование с шагом через вторичную хеш-функцию

if (plenty[newIndex] == nullptr) {

plenty[newIndex] = new Set{value};

break;

}

}

}

}

void SetNode::SETDEL(string value) {

int index = HashSet(value);

if (plenty[index] == nullptr) {

cout << endl << "The element does not exist!!!" << endl;

return;

}

if (plenty[index]->value == value) { //Если было без коллизий

delete plenty[index];

plenty[index] = nullptr;

return;

}

bool isExist = FindValue(value, index);

if (isExist) {

delete plenty[index];

plenty[index] = nullptr;

return;

}

}

void SetNode::SET\_AT(string value) {

int index = HashSet(value);

if (plenty[index] == nullptr) {

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

return;

}

if (plenty[index]->value == value) { //Если было без коллизий

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

return;

}

bool isExist = FindValue(value, index);

if (isExist) {

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

return;

}

}

void findSumsSet(SetNode& set, int index, int currentSum, string currentElements, SetNode& sums) { // 2^10 - 1 (пустое множество)

// Если мы дошли до конца множества

if (index == set.size) {

if (sums.plenty[currentSum] == nullptr) {

sums.SETADD(currentElements);

//cout << endl << "Sum: " << currentSum << " Elements: " << currentElements;

}

else {

sums.SETADD(currentElements);

//cout << endl << "Sum: " << currentSum << " Elements: " << currentElements;

}

return;

}

findSumsSet(set, index + 1, currentSum, currentElements, sums);

if (set.plenty[index] != nullptr) {

findSumsSet(set, index + 1, currentSum + stoi(set.plenty[index]->value), currentElements + (currentElements.empty() ? "" : " ") + set.plenty[index]->value, sums);

}

}

void FindSubsets(SetNode& set) {

int countElements = 0;

int totalSum = 0;

string allElements = "";

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

string valueStr = "";

if (set.plenty[i] != nullptr) {

valueStr = set.plenty[i]->value;

int value = stoi(valueStr);

totalSum += value;

allElements += " " + valueStr;

countElements++;

}

}

totalSum = totalSum/2;

SetNode values(1000);

stringstream ss(allElements);

string token;

while (getline(ss, token, ' ')) {

values.SETADD(token);

}

int i = 0;

SetNode sums(1000);

findSumsSet(set, 0, 0, "", sums); //поиск всех вариаций сумм

/\*

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

if (sums.plenty[i] != nullptr && !sums.plenty[i]->value.empty()) {

cout << "\t" << i << "\t" << sums.plenty[i]->value << endl;

}

}

\*/

int minSum = pow(10,9);

;

SetNode sumVariants(1000);

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

if (sums.plenty[i] != nullptr && !sums.plenty[i]->value.empty()) {

stringstream tempSS1(sums.plenty[i]->value);

string tempToken;

int tempSum = 0;

while(getline(tempSS1, tempToken, ' ')) {

tempSum+= stoi(tempToken);

}

if (abs(totalSum - tempSum) <= minSum) { // -i так как мы ищем близость к общей сумме всех элементов,

string tempNumbers; // а i это ключ - сумма, где value это список элементов этой суммы

stringstream tempSS2(sums.plenty[i]->value);

int countNumbers = 0;

while(getline(tempSS2, tempNumbers, ' ')) {

countNumbers++;

}

minSum = abs(totalSum - tempSum);

if (countNumbers != countElements) {

if ((totalSum-tempSum )== minSum) {

sumVariants.SETADD(sums.plenty[i]->value);

}

}

}

}

else {

continue;

}

}

string subsetA = "";

string subsetB = "";

int minDiff = pow(10,9);

int sumA = 0;

int sumB = 0;

string finalSubSetA = "";

string finalSubSetB = "";

int finalDifference = 0;

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

sumB = 0;

sumA = 0;

subsetB = "";

if(sumVariants.plenty[i] != nullptr && !sumVariants.plenty[i]->value.empty()) {

subsetA = sumVariants.plenty[i]->value;

stringstream ssA(subsetA);

string tokenA;

while (getline(ssA, tokenA, ' ')) {

if (tokenA != "") {

sumA += stoi(tokenA);

}

}

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

if (values.plenty[i] != nullptr && !values.plenty[i]->value.empty()) {

if (subsetA.find(values.plenty[i]->value) == string::npos) {

subsetB += " " + values.plenty[i]->value;

sumB += stoi(values.plenty[i]->value);

}

}

}

if(abs(sumA-sumB) < minDiff) {

finalSubSetA = subsetA;

finalSubSetB = subsetB;

finalDifference = abs(sumA-sumB);

minDiff = abs(sumA-sumB);

}

}

}

cout << endl << "The maximum amount difference: " << finalDifference << endl;

cout << endl << "The first subset: " << finalSubSetA << endl;

cout << endl << "The second subset: " << finalSubSetB << endl;

}

## Stack.cpp

#include <sstream>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

#include "header.h"

void Stack::Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

size = 0;

}

void Stack::SPUSH(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 Stack::SPOP() {

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

SingleNode\* current = head; ///////////

head = head->next;

delete current;

}

void Stack::SREAD() {

SingleNode\* current = head;

cout << endl;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

void Stack::DeleteTail() {

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 Stack::AddTail(string&cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (tail == nullptr) {

head = tail = node;

tail->next = nullptr;

}

else {

tail->next = node;

tail = node;

tail->next = nullptr;

}

}

void Stack::DelElementValue(SingleNode\* target) {

if (head == nullptr) {

return;

}

if (head == target) {

size--;

SPOP();

return;

}

SingleNode\* current = head;

while (current != nullptr && current->next != target) {

current = current->next;

}

if (current != nullptr && current->next == target) {

SingleNode\* temp = current->next;

if (current->next != nullptr) {

current->next = current->next->next;

}

else {

current->next = nullptr;

}

delete temp;

}

size--;

}

void ProcessingExpression(const string& oper, Stack& stackOper, Stack& stackNumb, string current, bool parameter) {

stackOper.DeleteTail();

int A = 0;

int B = stoi(stackNumb.tail->cell);

stackNumb.DeleteTail();

if (parameter == 1) {

A = stoi(stackNumb.tail->cell);

A = A \* (-1);

}

else {

A = stoi(stackNumb.tail->cell);

}

stackNumb.DeleteTail();

int result;

if (oper == "\*") result = A \* B;

if (oper == "+") result = A + B;

if (oper == "-") result = A - B;

if (result < 0 && parameter == 1) {

result = result \* (-1);

}

string temp = to\_string(result);

stackNumb.AddTail(temp);

stackOper.AddTail(current);

}

# Функциональная характеристика программы

## Примеры работы программы

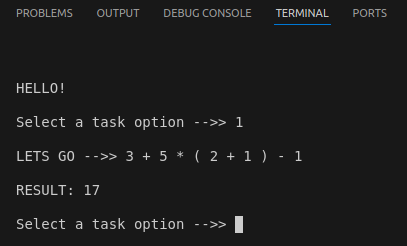


Рисунок 1. Пример работы программы для 1 задания.

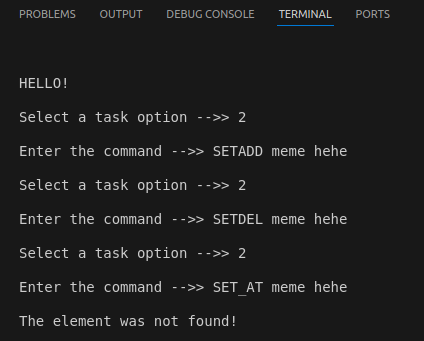


Рисунок 2. Пример работы программы для 2 задания.

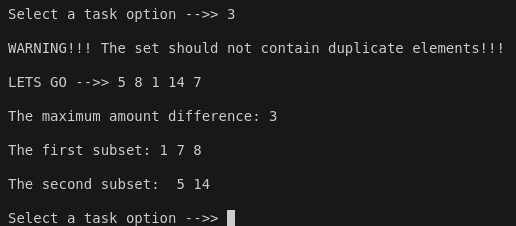


Рисунок 3. Пример работы программы для 3 задания.

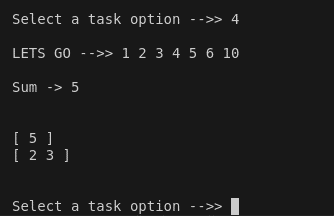


Рисунок 4. Пример работы программы для 4 задания.

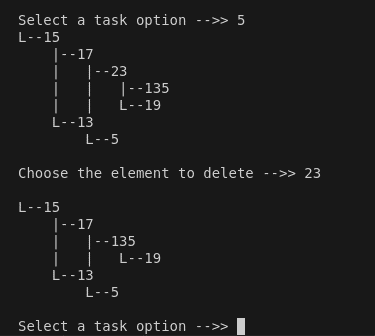


Рисунок 5. Пример работы программы для 5 задания.

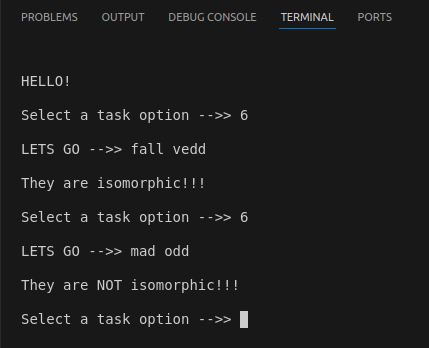


Рисунок 6. Пример работы программы для 6 задания.

## Схемы работы программы

1. Массив.

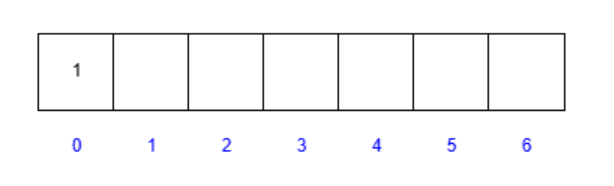
****

Рисунок 1. Массив после добавления элемента 1.

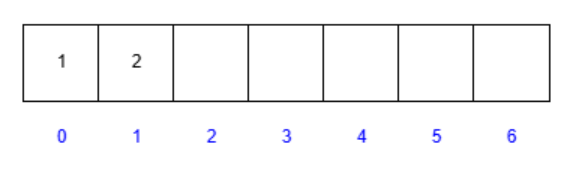
****

Рисунок 2. Массив после добавления элемента 1 по индексу 2.

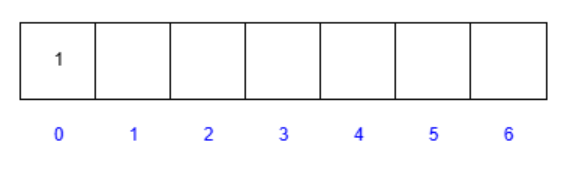
****

Рисунок 3. Массив после удаления элемента 1.

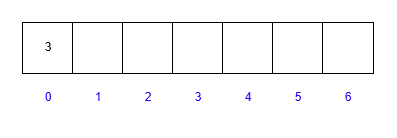


Рисунок 4. Массив после замены элемента 3 по индексу 0.

1. Стек.

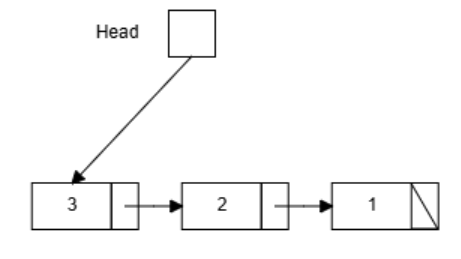


Рисунок 5. Стек после добавления элементов 3, 2, 1.

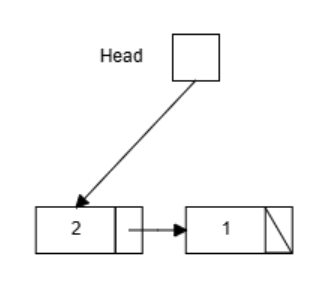


Рисунок 6. Стек после SPOP.

1. Хэш-таблица.

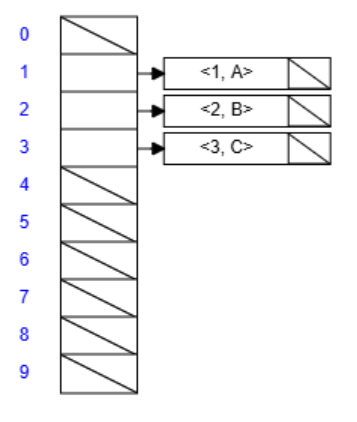


Рисунок 7. Хеш-таблица после добавления элементов в формате <ключ, значение>

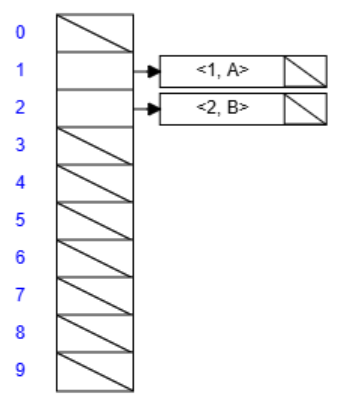


Рисунок 8. Хеш-таблица после удаления элемента.

1. Бинарное дерево поиска.

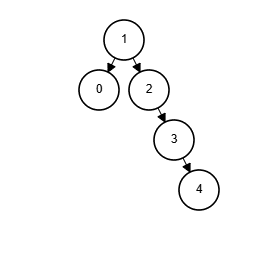


Рисунок 9. Бинарное дерево поиска после добавления элементов 1,2,3,0,4.

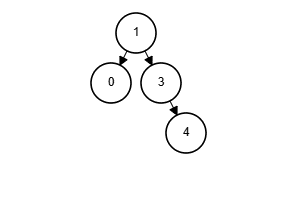


Рисунок 10. Бинарное дерево поиска после удаления элемента 2.

1. Сет ( хэш-таблица )

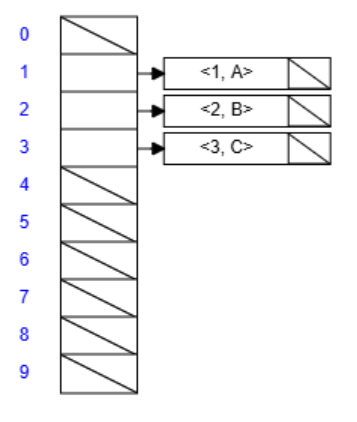


Рисунок 11. Хеш-таблица после добавления элементов в формате <ключ, значение>

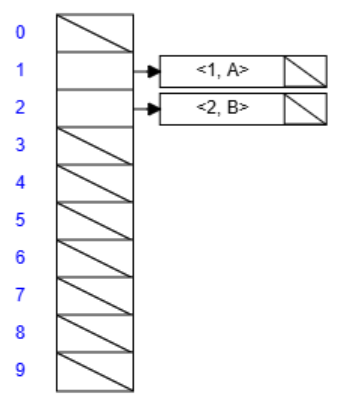


Рисунок 12. Хеш-таблица после удаления элемента.

## Сложность использования структур с позиции BigO нотации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип контейнера | Добавление | Удаление | Получение  (Поиск) | Замена | Чтение |
| Массив | O(1)  O(n) | O(n) | O(1) | O(1) | O(n) |
| Стек | O(1) | O(1) | - | - | O(n) |
| Хеш-таблица | O(1)-O(n) | O(1)-O(n) | O(1)-O(n) | - | O(n) |
| Бинарное дерево поиска | O(log n) | O(log n) | O(log n) | - | O(n) |
| Сет (хэш-таблица) | O(1)-O(n) | O(1)-O(n) | O(1)-O(n) | - | O(n) |

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

# Ссылки на репозитории

https://github.com/mamam111a/laboratory-2