МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Московский государственный университет геодезии и картографии”

(МИИГАИК)

Факультет геоинформатики и информационной безопасности

Кафедра геоинформационных систем и технологий

**Лабораторная работа №1**

**"Библиотечные реализации алгоритмов и их скорость"**

Проверил: Выполнил:

Лебедев Евгений Денисович Студент группы: 2024-ФГИИБ-ПИ-1б

Журавлев Андрей Дмитриевич

Москва 2025

**Вариант 9**

Формулировка задания:

Создать алгоритм по теме:

Оценка вычислительной сложности и потребления памяти: поиск элемента в Стэке

Определение структуры данных, с которой работаю:

Структура данных, с которой я работаю – стек.

Стек - это структура данных, организованная по принципу LIFO (last in, first out - "последним пришел, первым вышел")

Операция, которую я произвожу – поиск элемента.

Произвожу я её с помощью метода “ searchInStack“.

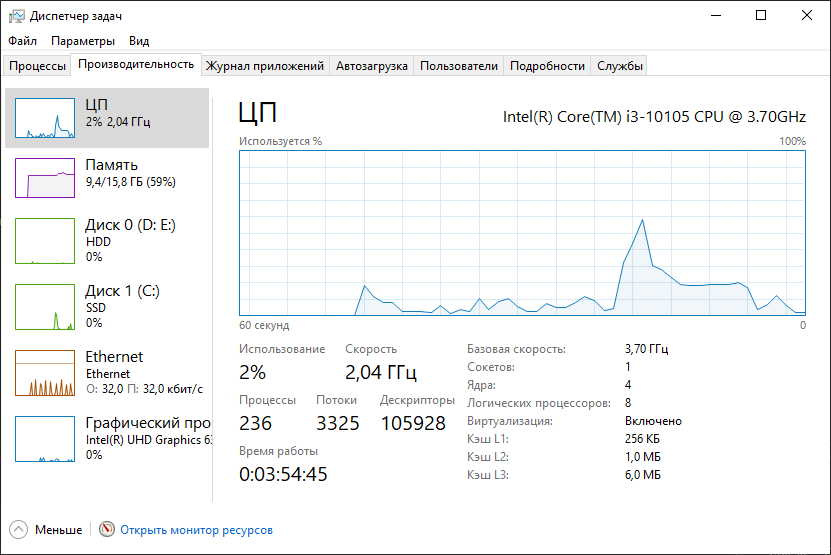
Описание ожидаемого результата:

Алгоритм должен корректно работать и считать время своей работы.

Я думаю что скорость нахождения будет зависить от места хранения элемента в стеке

Справка о процессоре на моём ПК:

Intel Core i3-10105 производится по стандарту 14-нм техпроцесса, имеет 4 ядра, которые работают в 8 потоков со штатной тактовой частотой 3.7 ГГц, 4.4 ГГц в режиме Turbo Boost. Объем кэш-памяти 3 уровня равен 6 МБ. Имеет 2-х канальный контроллер памяти DDR4.



Листинги кода реализации:

Основной код программы :  
  
#include <iostream>

#include <stack>

#include <chrono>

#include <unordered\_map>

bool searchInStack(std::stack<int> stack, int target) {

while (!stack.empty()) {

if (stack.top() == target) {

return true;

}

stack.pop();

}

return false;

}

void evaluateComplexity(std::stack<int> stack, int target) {

auto start = std::chrono::high\_resolution\_clock::now();

bool found = searchInStack(stack, target);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

std::cout << "Элемент " << (found ? "найден" : "не найден") << " в стеке." << std::endl;

std::cout << "Время выполнения: " << elapsed.count() << " секунд." << std::endl;

size\_t memoryUsage = sizeof(stack) + stack.size() \* sizeof(int);

std::cout << "Использование памяти: " << memoryUsage << " байт." << std::endl;

}

int main() {

std::stack<int> stack;

for (int i = 0; i < 10000000; ++i) {

stack.push(i);

}

int target = 1;

evaluateComplexity(stack, target);

return 0;

}

===================================================

Код для юнит-тестов :

#include "pch.h"

#include <stack>

#include <chrono>

#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(StackSearchTest)

{

public:

bool searchInStack(std::stack<int> stack, int target) {

while (!stack.empty()) {

if (stack.top() == target) {

return true;

}

stack.pop();

}

return false;

}

TEST\_METHOD(ElementFound)

{

std::stack<int> stack;

for (int i = 0; i < 10; ++i) {

stack.push(i);

}

int target = 5;

bool result = searchInStack(stack, target);

Assert::IsTrue(result);

}

TEST\_METHOD(ElementNotFound)

{

std::stack<int> stack;

for (int i = 0; i < 10; ++i) {

stack.push(i);

}

int target = 15;

bool result = searchInStack(stack, target);

Assert::IsFalse(result);

}

TEST\_METHOD(EvaluateComplexity)

{

std::stack<int> stack;

for (int i = 0; i < 1000000; ++i) {

stack.push(i);

}

int target = 999999;

auto start = std::chrono::high\_resolution\_clock::now();

bool found = searchInStack(stack, target);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

Logger::WriteMessage(("Элемент " + std::string(found ? "найден" : "не найден") + " в стеке.").c\_str());

Logger::WriteMessage(("Время выполнения: " + std::to\_string(elapsed.count()) + " секунд.").c\_str());

size\_t memoryUsage = sizeof(stack) + stack.size() \* sizeof(int);

Logger::WriteMessage(("Использование памяти: " + std::to\_string(memoryUsage) + " байт.").c\_str());

Assert::IsTrue(found);

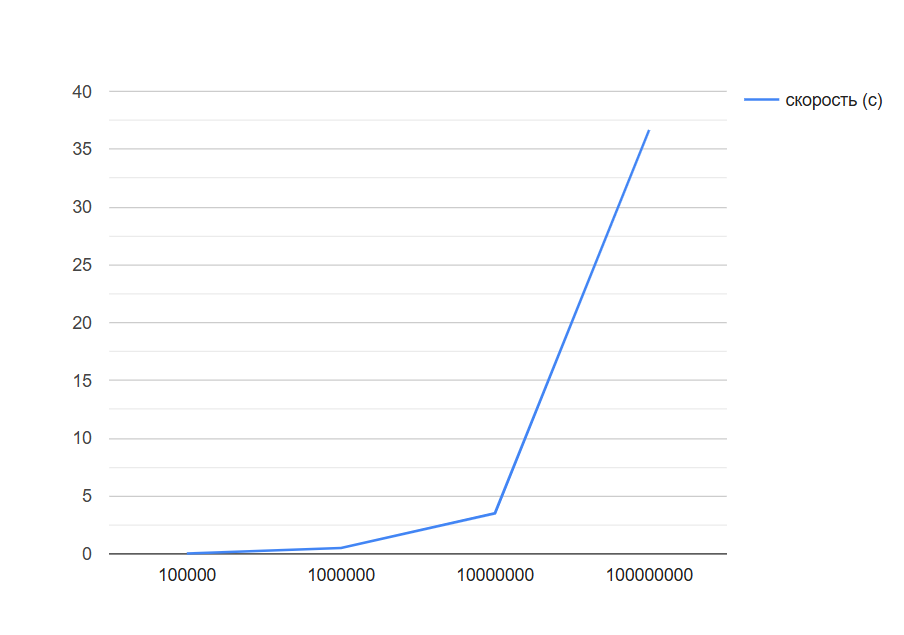
}

};

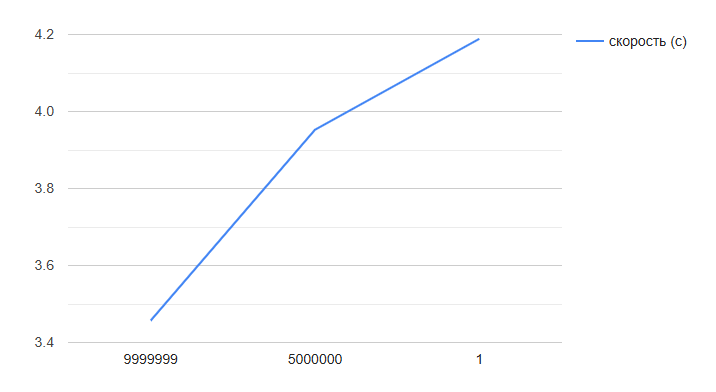
}

График с измерениями

График зависимости времени работы алгоритма от кол итераций



Измерение скорости работы алгоритма в зависимости от разной позиции нужного элемента (при количестве итериций = 10000000)



**Итак, скорость работы этого метода зависит не только от количества итериций, но и от положения нужного элемента в стеке. В свою очередь память зависит только от размера стека**