

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №4**

**з дисципліни «Якість програмного забезпечення»**

**на тему:**

**«**Проектування структур даних з урахуванням програмно-апаратних середовищ їх використання»

Виконала:

студентка гр. ПЗ2011

Костюк Анастасія Сергіївна

Прийняв:

Куроп’ятник О.С.

Дніпро, 2023

# Лабораторна робота № 4

**Тема.** Проектування структур даних з урахуванням програмно-апаратних середовищ їх використання

**Мета роботи.** Вивчити статистичні методи визначення показників часової ефективності алгоритмів та набути практичних навичок застосування цих методів при вирішенні задач розробки структур даних

## Зміст роботи:

1. вивчення методів визначення показників часової ефективності алгоритмів;
2. вивчення методів вибору структур даних та алгоритмів їх обробки з урахуванням показників часової ефективності у визначених програмно-апаратних середовищах;
3. розробка програми для виміру часової ефективності операцій обробки структур даних;
4. підготовка та виконання комп’ютерних експериментів щодо формування та використання альтернативних структур даних (за підготовленими в попередній роботі алгоритмами обробки даних).
5. розробка програми, що визначає часову ефективність окремих алгоритмів обробки даних та алгоритмів обробки даних у сукупності.
6. обґрунтований вибір структур даних для лексичного аналізатору з урахуванням показників часовою ефективністю.

## Зміст звіту:

1) вид і номер роботи, дисципліна, інформація про виконавця (група, П. І. Б.);

2) тема, мета лабораторної роботи;

3) постановка задачі з визначеними кращими структурами даних за комбінаторно-імовірнісних методами (попередня лабораторна робота).

4) текст програми, що реалізує альтернативні структури даних, які досліджуються, та операції над даними структурами;

5) опис методів дослідження (з приведеними програмами дослідження).

6) результати комп’ютерних експериментів (часові значення) та визначені показники часової ефективності.

7) аналіз результатів та висновки (обґрунтування вибору однієї з альтернативних структур даних)

## Постановка задачі:

На основі результатів лабораторної роботи №3, вибрати дві структури даних для зберігання лексем програми. Виконати операції обробки даних з виміром часу,

Виконати оцінювання часової ефективності алгоритмів та провести аналіз результатів, визначити, яка з структур є найбільш ефективною.

1. **Текст програми, що реалізує альтернативні структури даних** 
   1. **LinkedList.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace лаба4

{

public class LinkedList {

public class Node {

public int Data;

public Node Next;

public Node(int value) {

Data = value;

Next = null;

}

}

private Node head;

public void AddElement(int data)

{

Node newNode = new(data);

if (head == null || data < head.Data) {

newNode.Next = head;

head = newNode;

}

Else {

Node current = head;

while (current.Next != null && current.Next.Data < data) {

current = current.Next;

}

newNode.Next = current.Next;

current.Next = newNode;

}

}

public void RemoveElement(int data)

{

Node current = head;

if (current != null && current.Data == data) {

head = current.Next;

return;

}

while (current.Next != null && current.Next.Data != data) {

current = current.Next;

}

if (current.Next != null) {

current.Next = current.Next.Next;

}

}

public bool SearchElement(int data)

{

Node current = head;

while (current != null) {

if (current.Data == data) {

return true;

}

current = current.Next;

}

return false;

}

}

}

* 1. **SequentialStack.cs**

using System;

namespace лаба4

{

public class SequentialStack

{

private const int MaxSize = 100; // Максимальный размер стека

private readonly int[] elements; // Массив для хранения элементов

private int top; // Индекс вершины стека

public SequentialStack()

{

top = -1; // Инициализация вершины стека -1, что указывает на пустой стек

elements = new int[MaxSize]; // Инициализация массива элементов

}

public int Push(int data)

{

int res = -1;

if (top < MaxSize - 1) // Проверка, не полон ли стек

{

elements[++top] = data;

return 0;

}

else {

return res; // Возврат -1 для указания переполнения стека

}

}

public int Pop(int data\_to\_delete)

{

int res = -1;

if (top >= 0) {

return elements[top--]; // Возврат элемента и уменьшение вершины

}

else {

return res; // Возврат -1 для указания пустого стека

}

}

public bool Find(int data)

{

SequentialStack tempStack = new(); // Создание временного стека

bool found = false;

while (top >= 0) {

int current = Pop();

if (current == data) {

found = true;

}

tempStack.Push(current);

}

while (tempStack.top >= 0) {

Push(tempStack.Pop());

}

return found;

}

private int Pop(){

throw new NotImplementedException();

}

}

}

1. **Опис методів дослідження**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading;

using лаба4;

public class Program

{

// Функция для вычисления индекса Супремума

static double Sup(double i, double j)

{

double Sup;

if (i > j)

Sup = (j - i) / i;

else

Sup = (j - i) / j;

return Sup;

}

// Функция для вычисления индекса Регрессии

static double Reg(double i, double j)

{

double Reg;

Reg = j - i;

if (Reg > 0)

return 1;

return 0;

}

static void Main(string[] args)

{

// Создание объекта Random для генерации случайных чисел

Random rand = new Random();

// Создание объектов для каждого типа стека

//LinkedStack linkedStack = new LinkedStack();

LinkedList linkedList = new LinkedList();

SequentialStack sequentialStack = new SequentialStack();

int total\_repeat = 500; // Общее кол-во повторений эксперимента

int count\_of\_data = 1001; // Количество элементов данных

// Переменные для измерения времени вставки и поиска

double list\_time\_i, stack\_time\_i, list\_time\_s, stack\_time\_s, list\_time\_d, stack\_time\_d;

// Переменные для вычисления индексов Супремума и Регрессии

double sup\_i = 0;

double reg\_i = 0;

double sup\_d = 0;

double sup\_s = 0;

double reg\_s = 0;

double reg\_d = 0;

// Время задержки между экспериментами

int t = 10;

// Максимальное значение для генерации случайных чисел

int num = 100;

for (int i = 0; i < total\_repeat; i++)

{

linkedList = new LinkedList();

sequentialStack = new SequentialStack();

int tmp\_loading = rand.Next(count\_of\_data - 1);

var stopwatch = Stopwatch.StartNew();

for (int j = 0; j < tmp\_loading; j++)

{

int data\_to\_add = rand.Next(num);

linkedList.AddElement(data\_to\_add);

}

Thread.Sleep(t);

stopwatch.Stop();

list\_time\_i = stopwatch.Elapsed.TotalMilliseconds;

stopwatch.Restart();

for (int j = 0; j < tmp\_loading; j++)

{

int data\_to\_add = rand.Next(num);

sequentialStack.Push(data\_to\_add);

}

Thread.Sleep(t);

stopwatch.Stop();

stack\_time\_i = stopwatch.Elapsed.TotalMilliseconds;

sup\_i += Sup(list\_time\_i, stack\_time\_i);

reg\_i += Reg(list\_time\_i, stack\_time\_i);

// delete

stopwatch.Restart();

for (int j = 0; j < tmp\_loading; j++)

{

int data\_to\_delete = rand.Next(num);

linkedList.RemoveElement(data\_to\_delete);

}

Thread.Sleep(t);

stopwatch.Stop();

list\_time\_d = stopwatch.Elapsed.TotalMilliseconds;

stopwatch.Restart();

for (int j = 0; j < tmp\_loading; j++)

{

int data\_to\_delete = rand.Next(num);

sequentialStack.Pop(data\_to\_delete);

}

Thread.Sleep(t);

stopwatch.Stop();

stack\_time\_d = stopwatch.Elapsed.TotalMilliseconds;

sup\_d += Sup(list\_time\_d, stack\_time\_d);

reg\_d += Reg(list\_time\_d, stack\_time\_d);

// poisk

stopwatch.Restart();

for (int j = 0; j < tmp\_loading; j++)

{

int data = rand.Next(num);

linkedList.SearchElement(data);

}

Thread.Sleep(t);

stopwatch.Stop();

list\_time\_s = stopwatch.Elapsed.TotalMilliseconds;

stopwatch.Restart();

for (int j = 0; j < tmp\_loading; j++)

{

int data = rand.Next(num);

sequentialStack.Find(data);

}

Thread.Sleep(t);

stopwatch.Stop();

stack\_time\_s = stopwatch.Elapsed.TotalMilliseconds;

sup\_s += Sup(list\_time\_s, stack\_time\_s);

reg\_s += Reg(list\_time\_s, stack\_time\_s);

}

sup\_i = (sup\_i / total\_repeat) \* 100;

reg\_i = (reg\_i / total\_repeat) \* 100;

sup\_s = (sup\_s / total\_repeat) \* 100;

reg\_s = (reg\_s / total\_repeat) \* 100;

sup\_d = (sup\_d / total\_repeat) \* 100;

reg\_d = (reg\_d / total\_repeat) \* 100;

//Console.WriteLine($"supI(LL, SS) = {sup\_i}\n supD(LL, SS) = {sup\_d}\n supS(LL, SS) = {sup\_s}\n regI(LL, SS) = {reg\_i}\n regD(LL, SS) = {reg\_d}\n regS(LL, SS) = {reg\_s}");

//Console.WriteLine();

//Console.Read();

Console.WriteLine("Results of computational complexity metrics calculation LinkedList & SequentialStack:");

Console.WriteLine("Operation\t\tS-\t\t\tR-");

Console.WriteLine("Insertion\t\t{0:0.0}%\t\t{1:0.0}%", sup\_i, reg\_i);

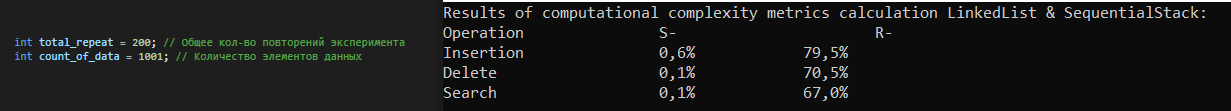
Console.WriteLine("Delete\t\t\t{0:0.0}%\t\t{1:0.0}%", sup\_d, reg\_d);

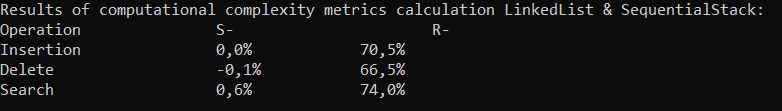
Console.WriteLine("Search\t\t\t{0:0.0}%\t\t{1:0.0}%", sup\_s, reg\_s);

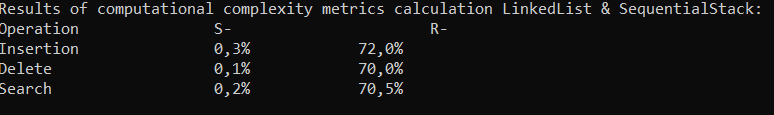
}

}

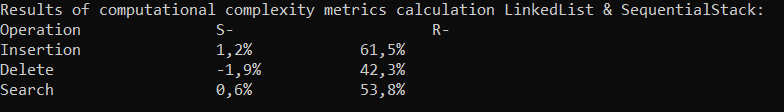
1. **Результати комп’ютерних експериментів**
   1. int total\_repeat = 200; // Общее кол-во повторений эксперимента  
      int count\_of\_data = 1001; // Количество элементов данных

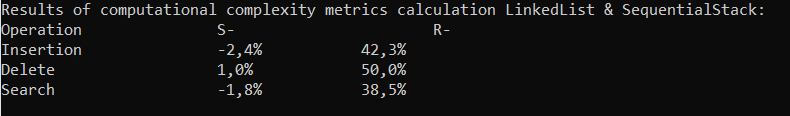
****

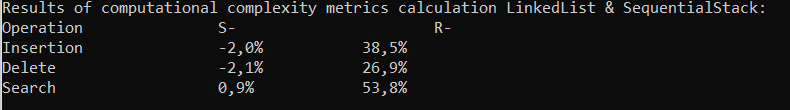
****

****

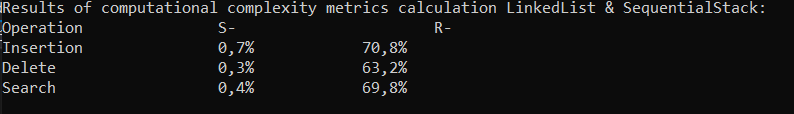
* 1. int total\_repeat = 26; // Общее кол-во повторений эксперимента  
     int count\_of\_data = 400; // Количество элементов данных

****

****

****

* 1. int total\_repeat = 500; // Общее кол-во повторений эксперимента  
     int count\_of\_data = 1001; // Количество элементов данных

****.

**Результати розрахунку показників обчислювальної складності однозв’язного впорядкованого списку та стека із послідовним представленням**

|  |  |  |
| --- | --- | --- |
| **Операція** | **Показники обчислювальної складності** | |
| Оцінка S- | Оцінка R- |
| Додавання до структури даних | 0,7% | 70,8% |
| Видалення у структурі даних | 0,3% | 63,2% |
| Пошук у структурі даних | 0,4% | 69,8% |

Оцінка S- демонструє перевагу списку над масивом. Якщо отримана оцінка додатня (операція додавання) – це перевага першої структури. Якщо оцінка від’ємна – це перевага другої структури. Оцінка R- демонструє співвідношення областей переваги. Якщо оцінка більша за 50% – це відсоток випробувань, що закінчились на користь першої структури. Якщо оцінка дорівнює 50% – області переваг рівнозначні. Якщо менша за 50% – це відсоток випробувань, у яких друга структура мала кращі часові показники.

1. **Аналіз результатів**

**Загалом**, індекси Супремуму та Регресії показують, що однозв’язний впорядкований список (LL) працює швидше, ніж послідовний стек (SS).

**Залежність від кількості повторень експерименту та даних:**Результати експерименту показують, що дані можуть відрізнятися залежно від кількості повторів експерименту. За невеликої кількості повторів експерименту стек із послідовним представленням може працювати трохи швидше, ніж зв’язний список.

Але при збільшенні кількості повторень експерименту результати мають усереднюватися, що дає змогу отримати більш стабільні та надійні оцінки.

У разі збільшення кількості елементів даних час виконання операцій вставки, видалення та пошуку збільшуватиметься. Це пов'язано з тим, що для обробки більшої кількості елементів даних потрібно більше часу.

**Однозв’язний впорядкований список:**

У моєму випадку результати експерименту показують, що при збільшенні кількості повторів експерименту та кількості даних однозв’язний впорядкований список працює швидше, ніж послідовний стек. Це пов'язано з тим, що однозв’язний впорядкований список підтримує більш ефективну структуру даних, яка може ефективно обробляти великі набори даних.

**Стек із послідовним представленням:**

Такий стек має простішу структуру даних, яка може бути ефективною для невеликих наборів даних. Однак, при збільшенні кількості елементів даних та збільшені кількості повторів експерименту перевага однозв’язного впорядкованого списку стає більш очевидною.

**Тому:**

* Індекси Супремуму та Регресії показують, що різниця в продуктивності між однозв’язним впорядкованим списком і стеком із зв’язним представленням змінюється залежно від кількості повторів експерименту та кількості елементів даних.

**У висновку,** результати експерименту показують, що впорядкований однозв’язний список є більш ефективною структурою даних, ніж стек із послідовним представленням.

**Висновки**

В ході виконання лабораторної роботи було визначено, що найкращою структурою є однозв’язний впорядкований список, так як такий список має найкращі значення показників в порівнянні з іншою протестованою структурою.