

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №4  
з дисципліни «Якість програмного забезпечення та тестування»**

**на тему:**

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

Виконав:

студент гр. ПЗ2011

Кулик С.В.

Прийняла:

Куроп’ятник О.С.

Дніпро, 2024

**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.

**Завдання**

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.

**Текст програм**

***Двозв’язний впорядкований список:***

package lr3;  
  
public class DoublyOrderedLinkedList {  
 private static class Node {  
 int data;  
 Node next;  
 Node prev;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
  
 private Node head;  
  
 public DoublyOrderedLinkedList() {  
 this.head = null;  
 }  
  
 public void add(int data) {  
 Node newNode = new Node(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 remove(int key) {  
 Node current = head;  
 Node prev = null;  
 while (current != null && current.data != key) {  
 prev = current;  
 current = current.next;  
 }  
 if (current != null) {  
 if (prev != null) {  
 prev.next = current.next;  
 } else {  
 head = current.next;  
 }  
 }  
 }  
  
 public boolean search(int key) {  
 Node current = head;  
 while (current != null && current.data != key) {  
 current = current.next;  
 }  
 return current != null;  
 }  
}

***Одномірний динамічний розширюваний масив:***

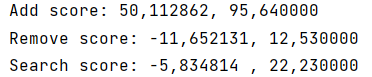
package lr3;  
  
import java.util.Arrays;  
  
public class DynamicArray {  
 private int[] array;  
 private int size;  
  
 public DynamicArray() {  
 this.array = new int[100];  
 this.size = 0;  
 }  
  
 public void add(int value) {  
 if (size == array.length) {  
 int newCapacity = array.length \* 2;  
 array = Arrays.copyOf(array, newCapacity);  
 }  
 array[size] = value;  
 size++;  
 }  
  
 public void remove(int key) {  
 int index = -1;  
 for (int i = 0; i < size; i++) {  
 if (array[i] == key) {  
 index = i;  
 }  
 }  
  
 if (index != -1) {  
 for (int i = index; i < size - 1; i++) {  
 array[i] = array[i + 1];  
 }  
 size--;  
 }  
 }  
  
 public boolean search(int key) {  
 for (int i = 0; i < size; i++) {  
 if (array[i] == key) {  
 return true;  
 }  
 }  
 return false;  
 }  
}

**Розроблена програма для тестування**

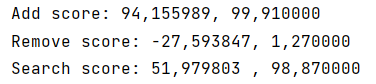
package lr4;  
  
import lr3.DoublyOrderedLinkedList;  
import lr3.DynamicArray;  
  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.Random;  
  
public class SpeedTest {  
 private static void addTest(int size, int countTest){  
 double sumS = 0, sumR = 0;  
 for (int i = 0; i < countTest; i++) {  
 DynamicArray array = new DynamicArray();  
 DoublyOrderedLinkedList list = new DoublyOrderedLinkedList();  
  
 int[] randomNumbers = new int[size];  
 for (int j = 0; j < size; j++) {  
 randomNumbers[j] = (int) (Math.*random*() \* 10000);  
 }  
  
 long start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 array.add(randomNumbers[j]);  
 }  
 long timeArray = System.*nanoTime*() - start;  
  
 start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 list.add(randomNumbers[j]);  
 }  
 long timeList = System.*nanoTime*() - start;  
  
 sumS += (timeList - timeArray) \* 1.0 / Math.*max*(timeArray, timeList);  
 if (timeList > timeArray) {  
 sumR++;  
 }  
 }  
 double scoreS = (sumS / countTest) \* 100.0;  
 double scoreR = (sumR / countTest) \* 100.0;  
  
 System.*out*.printf("Add score: %f, %f \n", scoreS, scoreR);  
 }  
  
 private static void removeTest(int size, int countTest){  
 double sumS = 0, sumR = 0;  
 for (int i = 0; i < countTest; i++) {  
 DynamicArray array = new DynamicArray();  
 DoublyOrderedLinkedList list = new DoublyOrderedLinkedList();  
  
 int[] randomNumbers = new int[size];  
 for (int j = 0; j < size; j++) {  
 randomNumbers[j] = (int) (Math.*random*() \* 10000);  
  
 array.add(randomNumbers[j]);  
 list.add(randomNumbers[j]);  
 }  
 *shuffle*(randomNumbers);  
  
 long start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 array.remove(randomNumbers[j]);  
 }  
 long timeArray = System.*nanoTime*() - start;  
  
 start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 list.remove(randomNumbers[j]);  
 }  
 long timeList = System.*nanoTime*() - start;  
  
 sumS += (timeList - timeArray) \* 1.0 / Math.*max*(timeArray, timeList);  
 if (timeList > timeArray) {  
 sumR++;  
 }  
 }  
 double scoreS = (sumS / countTest) \* 100.0;  
 double scoreR = (sumR / countTest) \* 100.0;  
  
 System.*out*.printf("Remove score: %f, %f \n", scoreS, scoreR);  
 }  
  
 public static void shuffle(int[] array) {  
 var random = new Random();  
 int count = array.length;  
 for (int i = count; i > 1; i--) {  
 *swap*(array, i - 1, random.nextInt(i));  
 }  
 }  
  
 private static void swap(int[] array, int i, int j) {  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 }  
  
 private static void searchTest(int size, int countTest){  
 double sumS = 0, sumR = 0;  
 for (int i = 0; i < countTest; i++) {  
 DynamicArray array = new DynamicArray();  
 DoublyOrderedLinkedList list = new DoublyOrderedLinkedList();  
  
 int[] randomNumbers = new int[size];  
 for (int j = 0; j < size; j++) {  
 randomNumbers[j] = (int) (Math.*random*() \* 10000);  
  
 array.add(randomNumbers[j]);  
 list.add(randomNumbers[j]);  
 }  
 *shuffle*(randomNumbers);  
  
 long start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 array.search(randomNumbers[j]);  
 }  
 long timeArray = System.*nanoTime*() - start;  
  
 start = System.*nanoTime*();  
 for (int j = 0; j < size; j++) {  
 list.search(randomNumbers[j]);  
 }  
 long timeList = System.*nanoTime*() - start;  
  
 sumS += (timeList - timeArray) \* 1.0 / Math.*max*(timeArray, timeList);  
 if (timeList > timeArray) {  
 sumR++;  
 }  
 }  
 double scoreS = (sumS / countTest) \* 100.0;  
 double scoreR = (sumR / countTest) \* 100.0;  
  
 System.*out*.printf("Search score: %f , %f \n", scoreS, scoreR);  
 }  
  
 public static void main(String[] args) {  
 int size = 1000, count = 10000;  
 *addTest*(size, count);  
 *removeTest*(size, count);  
 *searchTest*(size, count);  
 }  
}

**Аналіз результатів експериментів**

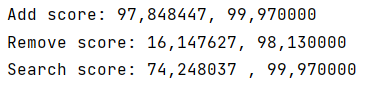
Тестування для розміру структур 10, та кількості повторень – 10000:



Тестування для розміру структур 100, та кількості повторень – 10000:



Тестування для розміру структур 1000, та кількості повторень – 10000:

****

Для оцінки параметрів S та R був використан метод із класу System.nanoTime(). Він дозволив розрахувати час виконання операцій у наносекундах для операцій додавання, пошуку та видалення у дані структури.

Оцінка S- демонструє перевагу масиву над списком. Оцінка R- демонструє співвідношення областей переваги. Якщо оцінка більша за 50% – це відсоток випробувань, що закінчились на користь другої структури. Якщо оцінка дорівнює 50% – області переваг рівнозначні. Якщо менша за 50% – це відсоток випробувань, у яких перша структура мала кращі часові показники.

Проаналізувавши отримані результати можна зробити наступні висновки:

* Операція додавання у всіх випадках краща для масиву, а не для списку, при чому зі збільшенням кількості елементів нічого не змінюється.
* Операція видалення краще працює для списку при малих розмірах, проте при збільшенні починає програвати у ефективності масиву, і при великих розмірах значно програє.
* Операція пошуку також краще працює для списку лише при малих розмірах, до 10-15 елементів, а після програє у ефективності масиву.

В цілому можна зробити висновки, що використовувати масив більш ефективно при більших розмірах структури, з якою буде йти робота, а ефективність списку проявляється лише при малих розмірах.

**Висновки**

Проектування структур даних є важливою складовою процесу розробки програмного забезпечення, особливо при урахуванні програмно-апаратних середовищ їх використання. У ході вивчення статистичних методів визначення показників часової ефективності алгоритмів було виявлено, що ці методи надають об'єктивні та кількісні критерії для оцінки ефективності розроблених структур даних.

Практичні навички, набуті під час роботи, дозволяють ефективно застосовувати статистичні методи при вирішенні завдань розробки структур даних. Аналіз часової складності алгоритмів дозволяє розробникам зрозуміти, як ефективно їхні структури даних працюють у конкретних програмно-апаратних середовищах.