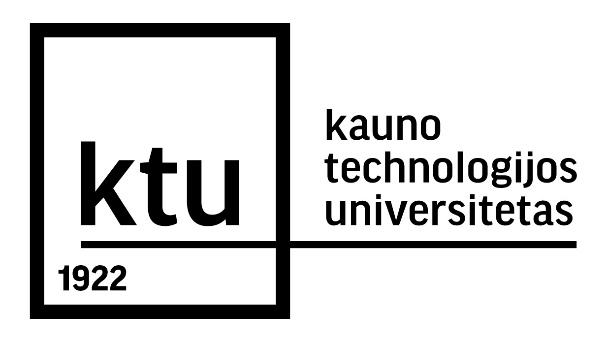
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P175B014 „Duomenų struktūros“**

Projektas: duomenų struktūrų kūrimas pasirinktam ADT

ataskaita

|  |  |
| --- | --- |
|  | **Dėstytojas**  lekt. BUDNIKAS Aurelijus |
|  |
| **Studentas**  Matas Vaitkevičius IFF-8/8 gr. |

**KAUNAS, 2019**

Turinys

[Turinys 2](#_Toc26991217)

[1. Darbo eiga: 3](#_Toc26991218)

[2. Atlikimas 4](#_Toc26991219)

[2.1 Pasirinkau realizuoti PriorityQueue duomenų struktūrą. Išsirinkau konkretų gyvenimišką pavyzdį, kuriame stojant į eilę prioritetai labai svarbūs. Tai pagalbos centro skambučių registravimas į eilę pagal jų svarbą. Skambučius lyginame pagal du parametrus: 4](#_Toc26991220)

[2.2 Projekto struktūra: 4](#_Toc26991221)

[2.2.1 Grafinė vartotojo sąsaja: 28](#_Toc26991222)

[2.2.2 Greitaveikos rezultatai: 29](#_Toc26991223)

[2.2.3 Išvados apie greitaveiką: 29](#_Toc26991224)

[3. IŠVADOS. 30](#_Toc26991225)

# ****Darbo eiga:****

**Darbo tikslai:**

1. Išmokti kurti naujas bendro naudojimo duomenų struktūrų klases pagal duotą ADT aprašą.

2. Įtvirtinti projektavimo, programavimo, testavimo ir tyrimo procesų žinias;

**Darbo eiga**

**1.** Pasirenkamas paskaitos metu nagrinėtas ar literatūroje aprašytas linijinės duomenų struktūros ADT.

Galimų duomenų struktūrų sąrašas:

•    Stack - realizuota pavyzdyje  
•    Queue- maksimalus balas 5  
•    Deque- maksimalus balas 5  
•    Doubly linked list - max 7  
•    Priority queue (8)  
•    Bit array (8)         •    Dynamic array (8)      •    Circular buffer  
•    Sparse array     •    Sparse matrix      
•    Set - įvairūs varijantai     •    Multiset  
•    Unrolled linked list         •    VList         •    Skip list

Pasirinkus dominančią ar aktualią duomenų struktūrą, reikia:   
- apsibrėžti jos sąsają (interfeisą);  
- suprogramuoti sąsają realizuojančią klasę (-es)  
- sudaryti testo klases;  
- atlikti efektyvumo tyrimą (palyginti su kita DS)  
- \* pateikti demonstracinį panaudojimo variantą.

# Atlikimas

### Pasirinkau realizuoti PriorityQueue duomenų struktūrą. Išsirinkau konkretų gyvenimišką pavyzdį, kuriame stojant į eilę prioritetai labai svarbūs. Tai pagalbos centro skambučių registravimas į eilę pagal jų svarbą. Skambučius lyginame pagal du parametrus:

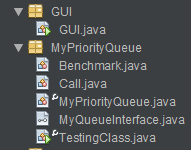
#### Grėsmę gyvybei – dangerProperty (galimos reikšmės true arba false)

#### Pavojaus lygmenį – dangerousLevel (int 0..100)

Pirmiausia atsižvelgiama į grėsmės gyvybei pavojų, jeigu jis egzistuoja (yra true), tada jis pagal prioritetą dedamas į eilės priekį, tačiau taip pat prieš įterpiant tokį įrašą, jis palyginamas su kitais įrašais jau esančiais eilėje ir taip pat turinčiais grėsmės gyvybei požymį, tokiu atveju elementai lyginami pagal antrąjį požymį – pavojaus lygmenį.

Jeigu pirmasis požymis – pavojus gyvybei nurodytas, kaip false, tada naujas skambutis į eilę dedamas praleidžiant visus turinčius true reikšmę, ir atsižvelgiant tik į tuos elementus, kurie turi false požymį ir tada atliekama paieška pagal antrąjį požymį, reikšmingumo lygmenį.

### Projekto struktūra:



**GUI.java:**

package GUI;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JButton;

import java.awt.Color;

import java.awt.Container;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Locale;

import javax.swing.BoxLayout;

import javax.swing.JOptionPane;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.WindowConstants;

import javax.swing.border.TitledBorder;

import MyPriorityQueue.Call;

import MyPriorityQueue.Benchmark;

import MyPriorityQueue.MyPriorityQueue;

public class GUI extends JFrame implements Serializable, ActionListener {

private static final int RECORDS\_NUMBER = 15;

ArrayList<Call> primaryData = new ArrayList<>();

MyPriorityQueue mainQueue = new MyPriorityQueue();

JButton addBtn = new JButton("Add to queue");

JButton removeBtn = new JButton("Remove from queue");

JButton getMostImportantBtn = new JButton("Get most important element");

JButton generateQueueBtn = new JButton("Generate random queue");

JButton speedTestBtn = new JButton("Speedtest - Main");

JButton speedTestMiddleBtn = new JButton("Speedtest - Add middle");

JPanel dataBlock = new JPanel();

JPanel buttonsBlock = new JPanel();

JTextArea leftTextarea = new JTextArea(16,30);

JTextArea rightTextarea = new JTextArea(16,30);

JScrollPane addScrollForLeft = new JScrollPane(leftTextarea);

JScrollPane addScrollForRight = new JScrollPane(rightTextarea);

public GUI()

{

Container inside = getContentPane();

inside.setLayout(new BoxLayout(inside, BoxLayout.Y\_AXIS));

inside.add(dataBlock);

inside.add(buttonsBlock);

addBtn.addActionListener(this);

removeBtn.addActionListener(this);

getMostImportantBtn.addActionListener(this);

generateQueueBtn.addActionListener(this);

speedTestBtn.addActionListener(this);

speedTestMiddleBtn.addActionListener(this);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

arrangement();

appereance();

setVisible(true);

pack();

}

private void arrangement() {

setLocation(0, 0);

dataBlock.add(addScrollForLeft);

dataBlock.add(addScrollForRight);

buttonsBlock.setLayout(new BoxLayout(buttonsBlock, BoxLayout.X\_AXIS));

buttonsBlock.add(addBtn);

buttonsBlock.add(removeBtn);

buttonsBlock.setLayout(new BoxLayout(buttonsBlock, BoxLayout.X\_AXIS));

buttonsBlock.add(getMostImportantBtn);

buttonsBlock.add(generateQueueBtn);

buttonsBlock.add(speedTestBtn);

buttonsBlock.add(speedTestMiddleBtn);

}

private void appereance() {

dataBlock.setBorder(new TitledBorder("Queue Control Window"));

dataBlock.setBackground(Color.lightGray);

buttonsBlock.setBackground(Color.lightGray);

}

@Override

public void actionPerformed(ActionEvent e) {

Object clickedBtn = e.getSource();

if(clickedBtn == addBtn) {

this.addElementToQueue();

}

else if(clickedBtn == removeBtn) {

this.removeQeueElement();

}

else if(clickedBtn == getMostImportantBtn) {

int elementId = 8;

this.getMostImportantElement();

}

else if(clickedBtn == generateQueueBtn) {

this.generateData();

}

else if(clickedBtn == speedTestBtn) {

this.speedTest();

}

else if(clickedBtn == speedTestMiddleBtn) {

this.speedMiddleTest();

}

}

public static void main(String[] args) {

Locale.setDefault(Locale.US);

GUI gui = new GUI();

}

private void generateData() {

primaryData.clear();

mainQueue.clear();

leftTextarea.setText("");

rightTextarea.setText("");

Benchmark speedTest = new Benchmark();

primaryData = speedTest.generateData(RECORDS\_NUMBER);

for(int i = 0; i < RECORDS\_NUMBER; i++) {

leftTextarea.append(primaryData.get(i).toString());

}

}

private void removeQeueElement() {

if(mainQueue.getLength() != 0) {

mainQueue.poll();

this.reprintLeftTextArea();

this.reprintRightTextArea();

}

else {

JOptionPane.showMessageDialog(null, "There are no elements in queue!");

}

}

private void getMostImportantElement() {

if(mainQueue.getLength() != 0) {

JOptionPane.showMessageDialog(null, mainQueue.peek().toString());

}

else {

JOptionPane.showMessageDialog(null, "There are no elements in queue!");

}

}

private void addElementToQueue() {

if (!primaryData.isEmpty()) {

mainQueue.add(primaryData.get(0));

primaryData.remove(0);

this.reprintLeftTextArea();

this.reprintRightTextArea();

}

else {

JOptionPane.showMessageDialog(null, "Data list is empty");

}

}

private void speedTest() {

primaryData.clear();

mainQueue.clear();

this.reprintLeftTextArea();

this.reprintRightTextArea();

new Benchmark().SystemicTest(leftTextarea, false);

}

private void speedMiddleTest() {

primaryData.clear();

mainQueue.clear();

this.reprintLeftTextArea();

this.reprintRightTextArea();

new Benchmark().SystemicTest(rightTextarea, true);

}

private void reprintLeftTextArea() {

leftTextarea.setText("");

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

leftTextarea.append(primaryData.get(i).toString());

}

}

private void reprintRightTextArea() {

rightTextarea.setText("");

for(int i = 0; i < mainQueue.getLength(); i++) {

rightTextarea.append(mainQueue.getCallObject(i).toString());

}

}

}

Benchmark.java:

package MyPriorityQueue;

import SourceClasses.Timekeeper;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.PriorityQueue;

import java.util.Random;

import javax.swing.JTextArea;

public class Benchmark extends Thread {

static int[] testingQuantitiesMiddle = {1000, 2000, 3000, 4000};

static int[] testingQuantities = {10000, 20000, 40000, 50000};

JTextArea outputSource;

static ArrayList<Call> listOfCalls = new ArrayList<>();

static MyPriorityQueue queue = new MyPriorityQueue();

static PriorityQueue<Call> pQueue = new PriorityQueue();

Call[] baseObject;

Random ag = new Random();

public ArrayList generateData(int quantity){

String[] titlesArray = {"Gaisras", "Nuzudymas", "Apiplesimas", "Avarija", "Triuksmas", "Mustynes"};

Boolean[] lifeImportanceArray = {true, false};

baseObject= new Call[quantity];

ag.setSeed(2017);

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

int titleValueIndex = ag.nextInt(titlesArray.length);

int lifeValueIndex = ag.nextInt(lifeImportanceArray.length);

baseObject[i]= new Call(

titlesArray[titleValueIndex],

lifeImportanceArray[lifeValueIndex],

1 + ag.nextInt(100)

);

}

Collections.shuffle(Arrays.asList(baseObject));

listOfCalls.clear();

for(Call a: baseObject) listOfCalls.add(a);

return listOfCalls;

}

public void SystemicTest(JTextArea resultsArea, boolean addToMiddle) {

this.outputSource = resultsArea;

if(addToMiddle == true) {

Timekeeper tk = new Timekeeper(testingQuantitiesMiddle, outputSource);

for (int count : testingQuantitiesMiddle) {

generateData(count);

pQueue.clear();

queue.clear();

tk.start();

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

queue.add(listOfCalls.get(i));

}

tk.finish("OPQA+");

tk.seriesFinish();

}

}

else {

Timekeeper tk = new Timekeeper(testingQuantities, outputSource);

for (int count : testingQuantities) {

generateData(count);

pQueue.clear();

queue.clear();

//---------------------------------------------

tk.start();

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

queue.add2(listOfCalls.get(i));

}

tk.finish("OPQA");

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

pQueue.add(listOfCalls.get(i));

}

tk.finish("JPQA");

//---------------------------------------------

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

queue.remove();

}

tk.finish("OPQR");

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

pQueue.remove();

}

tk.finish("JPQR");

tk.seriesFinish();

}

}

}

}

**Call.java:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package MyPriorityQueue;

public class Call implements Comparable<Call>{

String event;

boolean dangerProperty;

int dangerLevel;

/\*\*

\* Constructor

\*

\* @param event

\* @param dangerProperty

\* @param dangerLevel

\*/

public Call(String event, boolean dangerProperty, int dangerLevel)

{

this.event = event;

this.dangerProperty = dangerProperty;

this.dangerLevel = dangerLevel;

}

public void setEvent(String event)

{

this.event = event;

}

public void setDangerProperty(boolean dangerProperty)

{

this.dangerProperty = dangerProperty;

}

public void setDangerProperty(int dangerLevel)

{

this.dangerLevel = dangerLevel;

}

public String getEvent()

{

return event;

}

public boolean getDangerPropery()

{

return dangerProperty;

}

public int getDangerLevel()

{

return dangerLevel;

}

/\*\*

\* Method that determines if one object is bigger

\* then another

\*

\* @param kl - object that is compared by

\* @return returns +1 if object is bigger then k1

\* returns -1 if object is smaller then k1

\* returns 0 if object is equal to k1

\*/

@Override

public int compareTo(Call kl) {

double newDangerLevel = kl.getDangerLevel();

if (dangerLevel < newDangerLevel) return -1;

if (dangerLevel > newDangerLevel) return +1;

return 0;

}

/\*\*

\* Formats data and returns string to print

\*

\* @return returns formatted data to string

\*/

@Override

public String toString()

{

return String.format("%-20s %-8s %15d%n",

event, dangerProperty, dangerLevel);

};

}

**MyPriorityQueue.java:**

package MyPriorityQueue;

import java.util.LinkedList;

public class MyPriorityQueue implements MyQueueInterface {

LinkedList<Call> queue = new LinkedList<>();

/\*\*

\* Returns length of queue

\*

\* @return returns true if operation was successful

\*/

public int getLength() {

return queue.size();

}

/\*\*

\* Returns object form queue by index

\*

\* @param i - the index to search object by

\* @return returns object from queue by index

\*/

public Call getCallObject(int i)

{

return queue.get(i);

}

/\*\*

\* Checks if queue is empty

\*

\* @return returns true if queue is empty

\*/

public boolean isEmpty() {

return queue.isEmpty();

}

/\*\*

\* Enqueues an object by evaluating on two parameters: dangerProperty

\* and dangerLevel and decides where to put it in queue

\*

\* @param e - object that is being enqueued

\* @return true if operation was successful

\*/

@Override

public boolean add(Call e) {

int queueLength = queue.size();

if(queueLength == 0)

{

return queue.add(e);

}

if(e.getDangerPropery() == true)

{

int index1 = 1;

Call current = queue.getFirst();

while(current.getDangerPropery() == true

&& (current.compareTo(e) == 0

|| current.compareTo(e) > 0)

&& index1 < queueLength)

{

current = queue.get(index1);

index1++;

}

queue.add(index1 - 1, e);

return true;

}

else

{

Call current = queue.getLast();

int index1 = queueLength;

while(current.getDangerPropery() == false

&& current.compareTo(e) < 0

&& index1 != 0)

{

index1--;

current = queue.get(index1);

}

if(index1 == queue.size())

{

queue.addLast(e);

return true;

}

else

{

queue.add(index1 + 1, e);

return true;

}

}

}

/\*\*

\* Enqueues object just by using default comparator

\*

\* @param e - object that is being enqueued

\*/

public void add2(Call e)

{

if(queue.size() == 0)

{

queue.add(e);

}

Call current = queue.getFirst();

if((current.compareTo(e) == 0 || current.compareTo(e) > 0))

queue.addFirst(e);

else

queue.addLast(e);

}

/\*\*

\* Dequeues all objects from queue

\*/

public void clear() {

queue.clear();

}

/\*\*

\* Dequeues first object from queue and returns it

\*

\* @return returns removed object

\*/

@Override

public Call remove() {

return queue.remove();

}

/\*\*

\* Enqueues object to the beginning of the queue

\*

\* @param callObject

\*/

@Override

public void addFirst(Call callObject) {

queue.addFirst(callObject);

}

/\*\*

\* Enqueues object to the end of the queue

\*

\* @param callObject

\*/

@Override

public void addLast(Call callObject) {

queue.addLast(callObject);

}

/\*\*

\* Dequeues and returns first object from queue

\*

\* @return returns first object from queue

\*/

@Override

public Call poll() {

return queue.poll();

}

/\*\*

\* Enqueues element to the end of the list

\*

\* @param callObject object that is being enqueued

\* @return returns true if operation was successful

\*/

@Override

public boolean offer(Call callObject)

{

return queue.offer(callObject);

}

/\*\*

\* Returns first object from queue

\*

\* @return returns first object from queue

\*/

@Override

public Call element()

{

return queue.element();

}

/\*\*

\* Retrieves but does not remove first element from queue

\*

\* @return returns first object from queue

\*/

@Override

public Call peek()

{

return queue.peek();

}

}

**MyQueueInterface.java:**

package MyPriorityQueue;

public interface MyQueueInterface {

/\*\*

\* Enqueues an object by evaluating on two parameters: dangerProperty

\* and dangerLevel and decides where to put it in queue

\*

\* @param callObject - object that is being enqueued

\* @return true if operation was successful

\*/

public boolean add(Call callObject);

/\*\*

\* Dequeues first object from queue and returns it

\*

\* @return returns removed object

\*/

public Call remove();

/\*\*

\* Enqueues object to the beginning of the queue

\*

\* @param callObject

\*/

public void addFirst(Call callObject);

/\*\*

\* Enqueues object to the end of the queue

\*

\* @param callObject

\*/

public void addLast(Call callObject);

/\*\*

\* Dequeues and returns first object from queue

\*

\* @return returns first object from queue

\*/

public Call poll();

/\*\*

\* Enqueues element to the end of the list

\*

\* @param callObject object that is being enqueued

\* @return returns true if operation was successful

\*/

public boolean offer(Call callObject);

/\*\*

\* Returns first object from queue

\*

\* @return returns first object from queue

\*/

public Call element();

/\*\*

\* Retrieves but does not remove first element from queue

\*

\* @return returns first object from queue

\*/

public Call peek();

}

**TestingClass.java:**

**package MyPriorityQueue;**

**public class TestingClass {**

**public static void main(String args[])**

**{**

**MyPriorityQueue priorityQueue = new MyPriorityQueue();**

**Call call1 = new Call("Gaisras", true, 1);**

**Call call2 = new Call("Vagyste", false, 15);**

**Call call3 = new Call("Vagyste", true, 100);**

**Call call4 = new Call("Gaisras", false, 1);**

**Call call5 = new Call("Gaisras", true, 55);**

**Call call6 = new Call("Gaisras", false, 13);**

**Call call7 = new Call("Gaisras", true, 100);**

**Call call8 = new Call("Gaisras", false, 15);**

**Call call9 = new Call("Gaisras", true, 101);**

**priorityQueue.add(call1);**

**priorityQueue.add(call2);**

**priorityQueue.add(call3);**

**priorityQueue.add(call4);**

**priorityQueue.add(call5);**

**priorityQueue.add(call6);**

**priorityQueue.add(call7);**

**priorityQueue.add(call8);**

**priorityQueue.add(call9);**

**System.out.println("==============================Add===================================");**

**for(int i = 0; i < priorityQueue.getLength(); i++)**

**{**

**Call call = priorityQueue.getCallObject(i);**

**System.out.println(call.getEvent()+ " " + call.getDangerPropery()+ " " + call.getDangerLevel());**

**}**

**System.out.println("=================================Poll========================================");**

**Call removedCall = priorityQueue.poll();**

**System.out.println(removedCall.getEvent()+ " " + removedCall.getDangerPropery()+ " " + removedCall.getDangerLevel());**

**System.out.println("=================================Offer========================================");**

**priorityQueue.offer(new Call("OfferTest", true, 1));**

**for(int i = 0; i < priorityQueue.getLength(); i++)**

**{**

**Call call = priorityQueue.getCallObject(i);**

**System.out.println(call.getEvent()+ " " + call.getDangerPropery()+ " " + call.getDangerLevel());**

**}**

**System.out.println("=================================Clear=======================================");**

**priorityQueue.clear();**

**for(int i = 0; i < priorityQueue.getLength(); i++)**

**{**

**Call call = priorityQueue.getCallObject(i);**

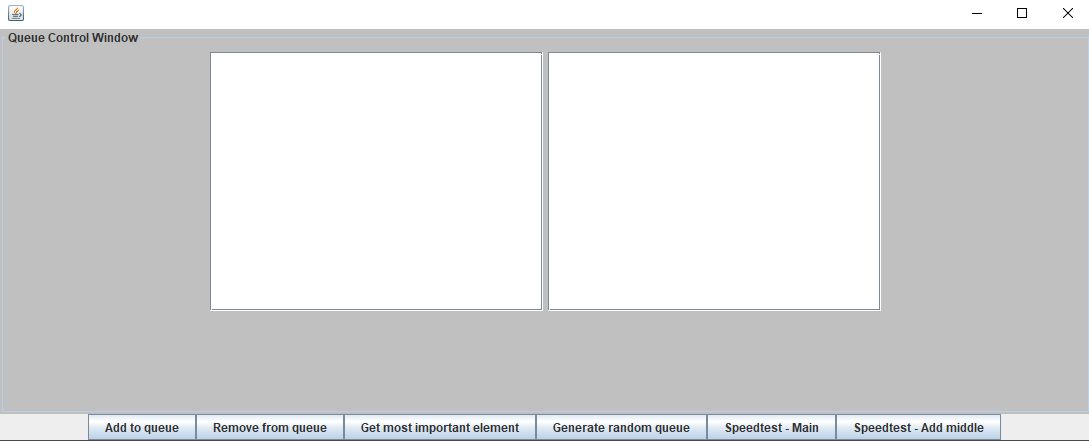
**System.out.println(call.getEvent()+ " " + call.getDangerPropery()+ " " + call.getDangerLevel());**

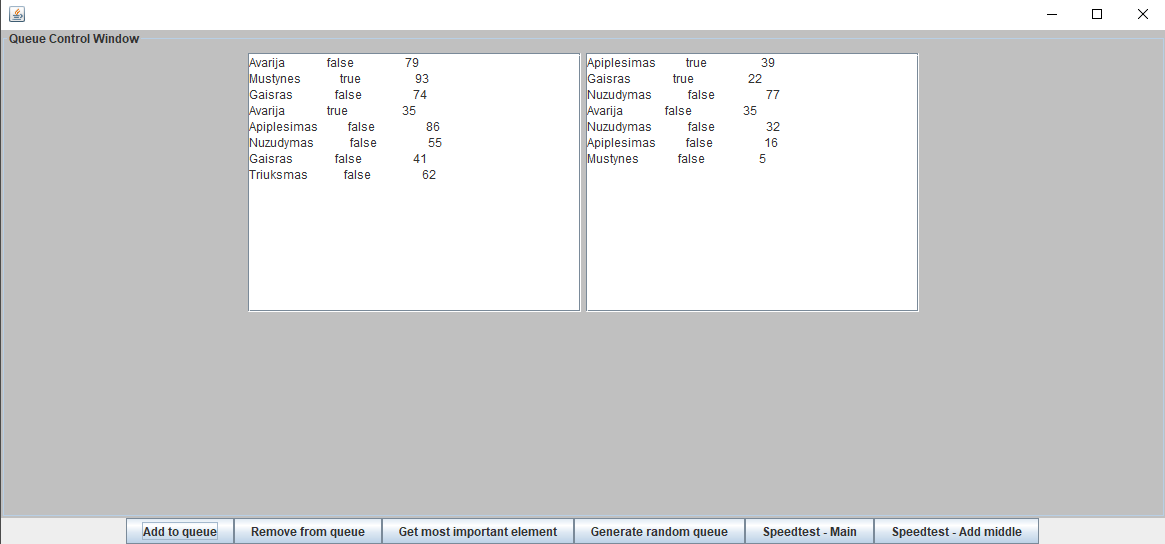
**}**

**}**

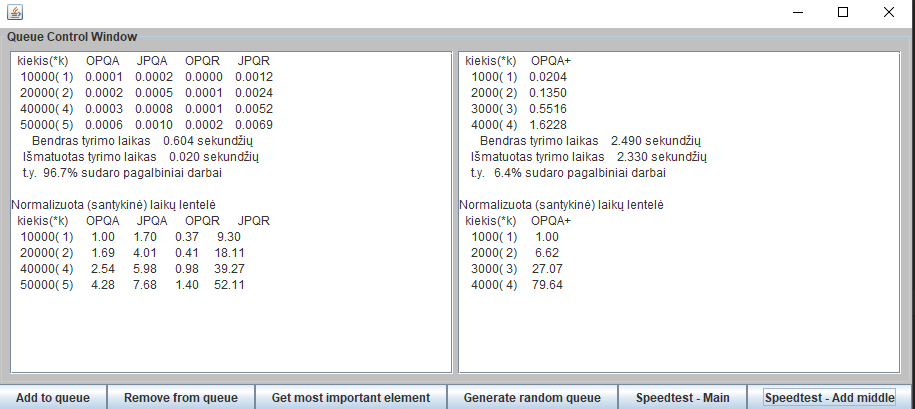
**}**

### Grafinė vartotojo sąsaja:





### Greitaveikos rezultatai:



### Išvados apie greitaveiką:

Atlikau greitaveikos testus, palyginau mano sukurtos eilės implementacijos metodų(OPQA) greitaveiką su Java PriorityQueue klase(JPQ). Mano sukurtas metodas elementus šalina(OPQR), kur kas greičiau negu java PriorityQueue klasė(JPQR). Taip pat pagal add metodo veikimo greičio palyginimą, matome, kad nors ir gana nežymiai, tačiau MyPriorityQueue klasės add metodas(OPQA) veikia šiek tiek greičiau negu pačios Java klasės(JPQA). Taip pat kitas add metodas(OPQA+), kuris pagal prioritetą, sugeba įdėti ir į eilės vidurį.

# IŠVADOS.

Išmokau kurti naujas bendro naudojimo duomenų struktūrų klases pagal pasirinktą priority queue ADT aprašą. Sukūriau savo priority queue eilės interfeisą bei jį implementuojančia klasę. Man pavyko pasiekti, kad Add bei Remove metodai veikai greičiau, nei įprastos Java PriorityQueue klasės. Taip pat įtvirtinau projektavimo, programavimo, testavimo ir tyrimo procesų žinias.