

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования
информационных систем

Средства анализа нагрузки на структуры хранения данных

Дипломная работа

Специальность 010503 Математическое обеспечение
и администрирование информационных систем

Специализация «Информационные системы»

Допущено к защите в ГЭК __.__.2015

Зав. кафедрой _____ д. ф.-м.н., проф. М.А. Артемов

Обучающийся _____ Д. Р. Потапов

Руководитель _____ преп. К. Е. Селезнев

Воронеж 2015

Содержание

Введение.....	4
1. Постановка задачи.....	6
2. Анализ задачи	7
2.1. Модель нагрузки	7
2.2. Примеры моделирования нагрузки	9
2.3. Методы индексации данных	10
2.4. Способ оценки метода индексации данных	11
2.5. Выбор оптимального способа хранения	12
2.6. Способ визуализации нагрузки на контейнер.....	13
2.6.1. Общая схема	13
2.6.2. Преобразование координат	15
2.6.3. Вычисление количества простейших операций	17
2.6.4. Определение цвета пикселя	17
2.7. Классификация	17
3. Средства реализации.....	20
4. Требования к аппаратному и программному обеспечению	21
5. Интерфейс пользователя	22
6. Реализация.....	28
6.1. Общая схема работы программы.	
Форматы входных и выходных данных	28
6.2. Общая архитектура программы.....	29
6.3. Модуль визуализации	31
6.4. Модуль генерации данных	33
6.5. Модуль перевода координат	35

6.6. Внешний модуль для тестирования нагрузки	36
6.7. Модуль классификации	37
6.8. Вспомогательные классы.	39
7. Тестирование	41
Заключение	48
Список литературы	49
Приложения	51
Приложение 1. Модуль визуализации и основной класс для запуска программы.....	51
Приложение 2. Модуль генерации данных	59
Приложение 3. Модуль перевода координат	64
Приложение 4. Модуль классификации	66
Приложение 5. Листинг. Вспомогательные классы	69

Введение

Сегодня много говорится об «информационном взрыве», суть которого состоит в постоянном увеличении скорости и объёмов информации в масштабах планеты. Это подтверждают цифры: в 2012 году человечеством было произведено информации $18 \cdot 10^{18}$ байт (18 Эксабайт). За пять предыдущих лет человечеством было произведено информации больше, чем за всю предшествующую историю. Объем информации в мире возрастает ежегодно на 30%. В среднем на человека в год в мире производится $2,5 \cdot 10^8$ байт. Следовательно, необходимо эффективно организовывать хранение и обработку информации.

На сегодняшний день существует много различных решений по хранению данных. У каждого из них есть свои плюсы и минусы. Кроме того, контейнеры (модули хранения информации) можно комбинировать различными способами. Таким образом, количество различных контейнеров, предназначенных для хранения информации и поддерживающих необходимые для работы с этими данными операции, огромно.

Выводы об их эффективности можно сделать в ходе анализа их работы при различной нагрузке и параметрах самих контейнеров. Для каждого конкретного случая на основе этих выводов можно сделать выбор оптимального из всего многообразия, в зависимости от нагрузки и других факторов, возникающих при работе с контейнерами.

Помимо математического и логического анализа преимуществ одних контейнеров над другими, большой помощью при выборе контейнера может служить графическое представление нагрузки. Визуализация одного модуля хранения информации поможет наглядно показать слабые и сильные стороны данного модуля, а визуализация сравнения нескольких сразу даст представление о том, какой способ хранения подходит лучше для каких операций.

Кроме того, рассмотрим необходимость выбора оптимального контейнера на примере индекса [5]. Если структура индекса хорошо оптимизирована под поиск, то происходит сильное ускорение работы. В основном для построения индекса используются такие структуры данных, как хеш-таблицы и сбалансированные деревья [4]. Сбалансированные деревья идеально подходят под данную задачу, но имеют очень большой недостаток – сложность и ресурсоемкость перестроения. Помимо этого, существуют системы с ограниченными ресурсами. В рассмотренных случаях должны применяться наиболее подходящие структуры хранения данных. Разумеется, вывод об оптимальном контейнере данных можно сделать на основе анализа работы системы. Но в таком случае встает вопрос, что будет, если возникнет ситуация, при которой данный контейнер окажется не оптимальным?

Данное исследование является актуальным, так как существует большое количество контейнеров для хранения данных и их комбинаций, при этом выбрать из них оптимальный для каждого конкретного случая можно только на основе статистики запросов к контейнеру данных.

1. Постановка задачи

Разработать программное обеспечение, в котором должны быть реализованы:

1. Визуализация различных параметров нагрузки на один контейнер.
2. Визуализация сравнения нескольких контейнеров.
3. Алгоритм получения линейного классификатора для адаптивного выбора оптимального способа хранения данных в зависимости от нагрузки.

Входные данные представляют собой параметры изображения (размер и название) и тестируемые контейнеры. Эти данные вводятся пользователем с консоли. Выходные данные представлены изображением, содержащим результаты анализа нагрузки на указанные структуры данных.

2. Анализ задачи

2.1. Модель нагрузки

Нагрузка представляет собой различные последовательности операций вставки, выборки и удаления [6][8]. Существует 6 видов нагрузки (рис. 2.1):



Рис. 2.1. Схема видов нагрузки

1. Последовательность (Sequence). Данная нагрузка означает, что все внутренние нагрузки идут в строгой последовательности.
2. Блок (Block). Данная нагрузка означает, что все внутренние нагрузки должны быть перемешаны в случайной последовательности.
3. Вставка (Insert). Нагрузка представляет собой набор операций вставки данных в контейнер.

4. Поиск (Select). Нагрузка представляет собой набор операций выборки данных.
5. Удаление (Remove). Нагрузка представляет собой набор операций удаления данных.

Для каждой нагрузки могут быть заданы дополнительные параметры:

1. Количество операций (count). Обязательный параметр для каждой нагрузки. По умолчанию равен 1.
2. Диапазон значений. Данный параметр является составным и описывается двумя параметрами:
 - минимальное значение (min). Минимальное возможное значение сгенерированного ключа;
 - максимальное значение (max). Максимальное значение такого ключа.
3. Псевдоним (alias). Показывает, что значения, сгенерированные данной нагрузкой, должны быть сохранены, т.к. могут быть использованы при генерировании другой нагрузки.
4. Ссылка на другую нагрузку (from). Показывает, что при генерации нагрузки должны быть использованы значения, сгенерированные в другой нагрузке.
5. Метка (label). Используется для выделения значений нагрузки в журнале, а также при подсчете операций, которые произвел контейнер, при выполнении нагрузки помеченной меткой. Используется только в случае, когда родительской нагрузкой является sequence.

Все базовые нагрузки обязательно должны содержаться в какой-либо структурной нагрузке. Вложенность структурных нагрузок неограниченна [7].

2.2. Примеры моделирования нагрузки

Пример 1. Обычная работа с заданным соотношением типов операций (рис. 2.2).



Рис. 2.2. Нагрузка для обычной работы

Пример 2. Обычная работа с первоначальным накоплением и заданным соотношением типов операций (рис. 2.3).

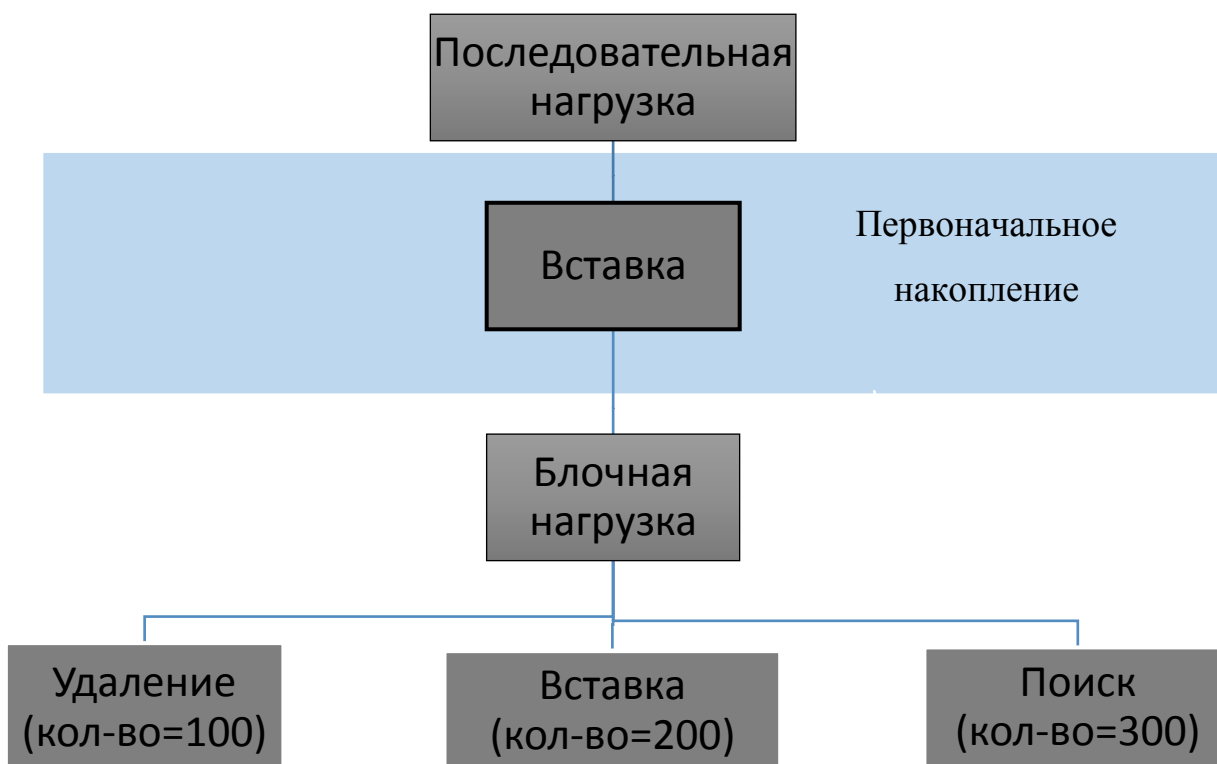


Рис. 2.3. Нагрузка для первоначального накопления и обычной работы

Пример 3. Чтение с заданным соотношением удачного и неудачного поиска (рис. 2.4).

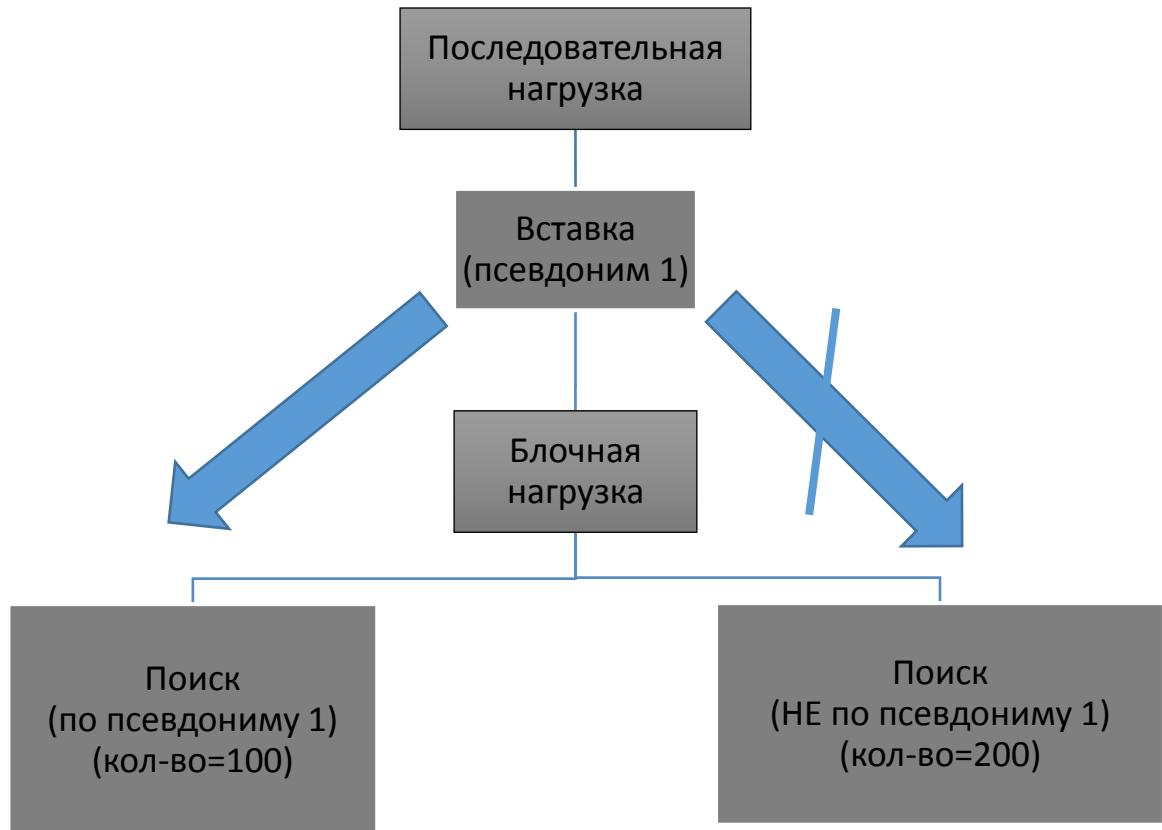


Рис. 2.4. Нагрузка для чтения с успешным и неуспешным поиском

2.3. Методы индексации данных

В качестве исследуемых методов индексации, выбраны такие распространенные структуры, как массивы, списки, деревья (сбалансированные бинарные деревья и В-деревья), хеш-таблицы [1][2][3].

Каждый метод индексации данных должен предоставлять базовый интерфейс для работы с контейнером. Такой интерфейс должен содержать три базовых операции:

1. Поиск (Select). Операция чтения данных из контейнера.
2. Вставка (Insert). Операция вставки данных.
3. Удаление (Remove). Операция удаления данных.

Каждая операция получает на вход данные, которые необходимо вставить в контейнер: в данном случае числовой идентификатор.

2.4. Способ оценки метода индексации данных

Внутри каждой операции, кроме операции вставки существующих данных, происходит подсчет количества простейших операций с контейнером. Поэтому каждый метод индексации данных должен также предоставлять методы для хранения и получения информации о количестве таких операций.

Для того чтобы оценить какой-либо из методов индексации данных, необходимо понять, что происходит при операциях (запись, выборка и удаление данных) с контейнером.

В наиболее простом случае, внутри контейнера каждую операцию можно разложить на композицию некоторого количества двух простейших операций: сравнения и присваивания (рис. 2.5). Более сложные варианты сравнения, учитывающие ввод-вывод и многопоточную параллельную обработку в данной работе не рассматриваются.

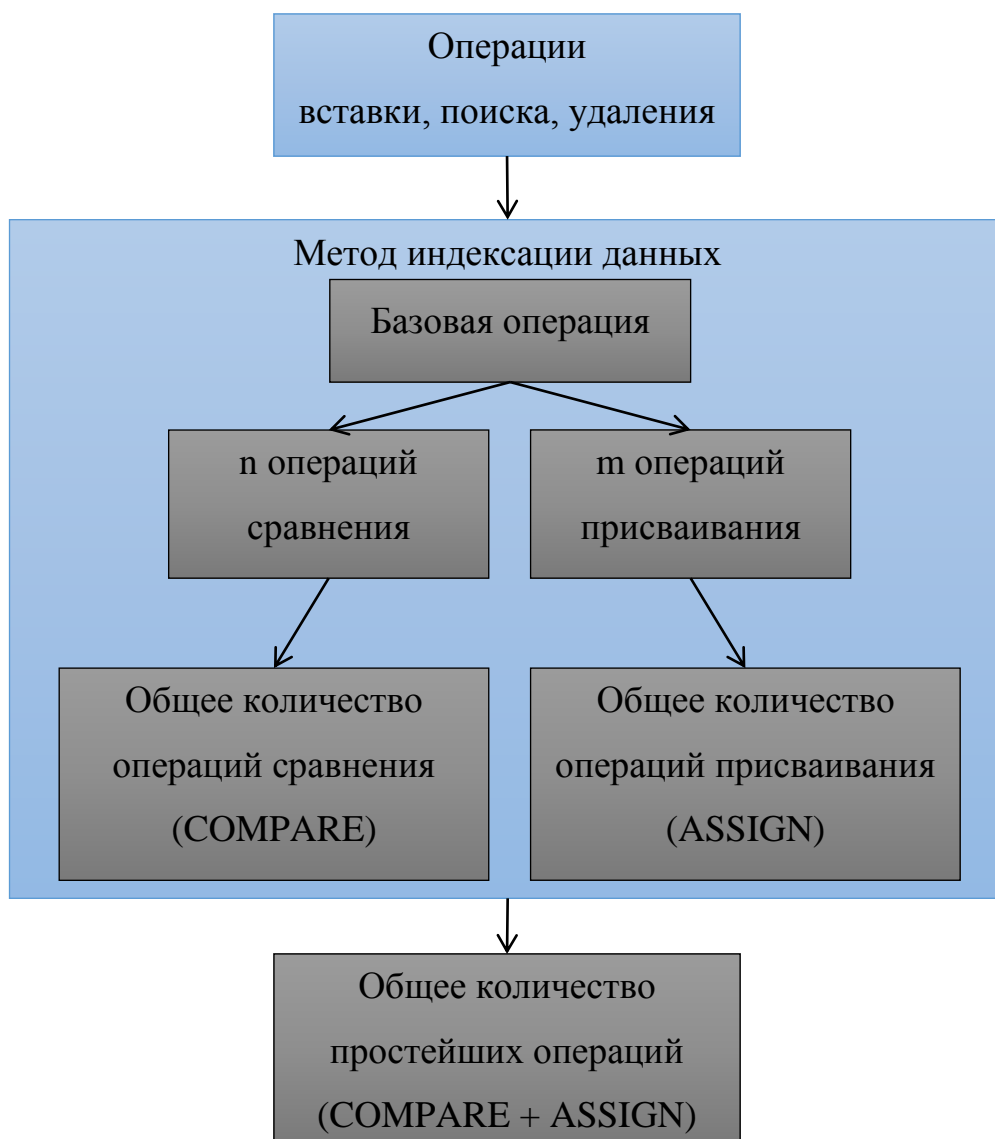


Рис. 2.5. Схема разложения базовых операций

Для каждого из описанных выше методов индексации подсчет таких операций производится по определенному алгоритму, отличающемуся для каждой структуры данных.

Результатом применения всех запросов (всех операций вставки, поиска и удаления) к контейнеру будет являться общее количество простейших операций.

2.5. Выбор оптимального способа хранения

В качестве критерия сравнения способов хранения данных стоит выбрать оценку суммарного количества операций сравнения и присваивания.

Оптимальной структурой хранения данных в таком случае будет являться та структура, у которой сумма таких операций будет минимальной.

$$\min_i \{a * Compare_i + b * Assign_i\}, \quad (2.1)$$

где i – идентификатор контейнера,

a – весовой коэффициент для операции сравнения,

b – весовой коэффициент для операции присваивания,

$Compare$ – количество операций сравнения,

$Assign$ – количество операций присваивания.

2.6. Способ визуализации нагрузки на контейнер

2.6.1. Общая схема

Поставленная задача предполагает два варианта визуализации:

1. Оценка одного контейнера.
2. Оценка нескольких контейнеров.

Общий алгоритм для них одинаков, но реализация некоторых пунктов различна.

Для решения задачи рассматривается трехмерная система координат по базовым операциям Insert, Remove, Select (IRS).

Для графического представления поведения контейнера в зависимости от количества операций был выбран следующий алгоритм:

1. Берется фиксированное число, равное сумме операций.
2. На основе этого числа строится плоскость, в которой при отсечении положительными осями координат образуется треугольник. Этот треугольник и попадет на конечное изображение.
3. Для каждой точки этого треугольника определяются координаты в трехмерном пространстве IRS (см. пункт 2.6.2).

4. По полученным координатам строится нагрузка, в которой количество каждой базовой операции совпадает с координатой по соответствующей оси (рис. 2.6).
5. Полученная нагрузка применяется к контейнеру (или контейнерам) (см. пункт 2.6.3).
6. На основе полученных значений количества простейших операций определяется цвет пикселя (см. пункт 2.6.4).

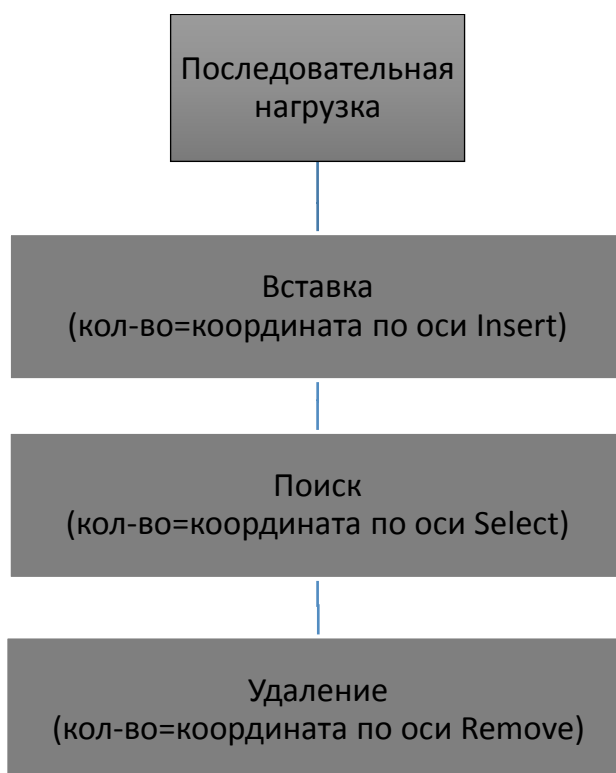


Рис. 2.6. Нагрузка для каждой точки изображения

По итогам этого алгоритма получаем на выходе треугольник, для каждого пикселя которого определен цвет.

При нахождении цвета пикселя создается нагрузка и тестируется на контейнерах. Это происходит для каждого пикселя, следовательно, при большом количестве пикселей время работы программы будет очень большим.

Для того чтобы его уменьшить, было решено использовать возможность многопоточности. Для этого все пиксели делятся на группы, и обработка каждой группы происходит в отдельном потоке.

2.6.2. Преобразование координат

Для визуализации нагрузки рассмотрим плоскость в пространстве IRS. Для удобства расчётов и наглядности построим плоскость так, что сумма координат всегда будет равна константе, т.е. $\text{Insert} + \text{Remove} + \text{Select} = \text{const}$, и будем рассматривать только положительные значения. В итоге получается треугольник, который представлен на рис. 2.7. Этот треугольник и будет отображаться пользователю.

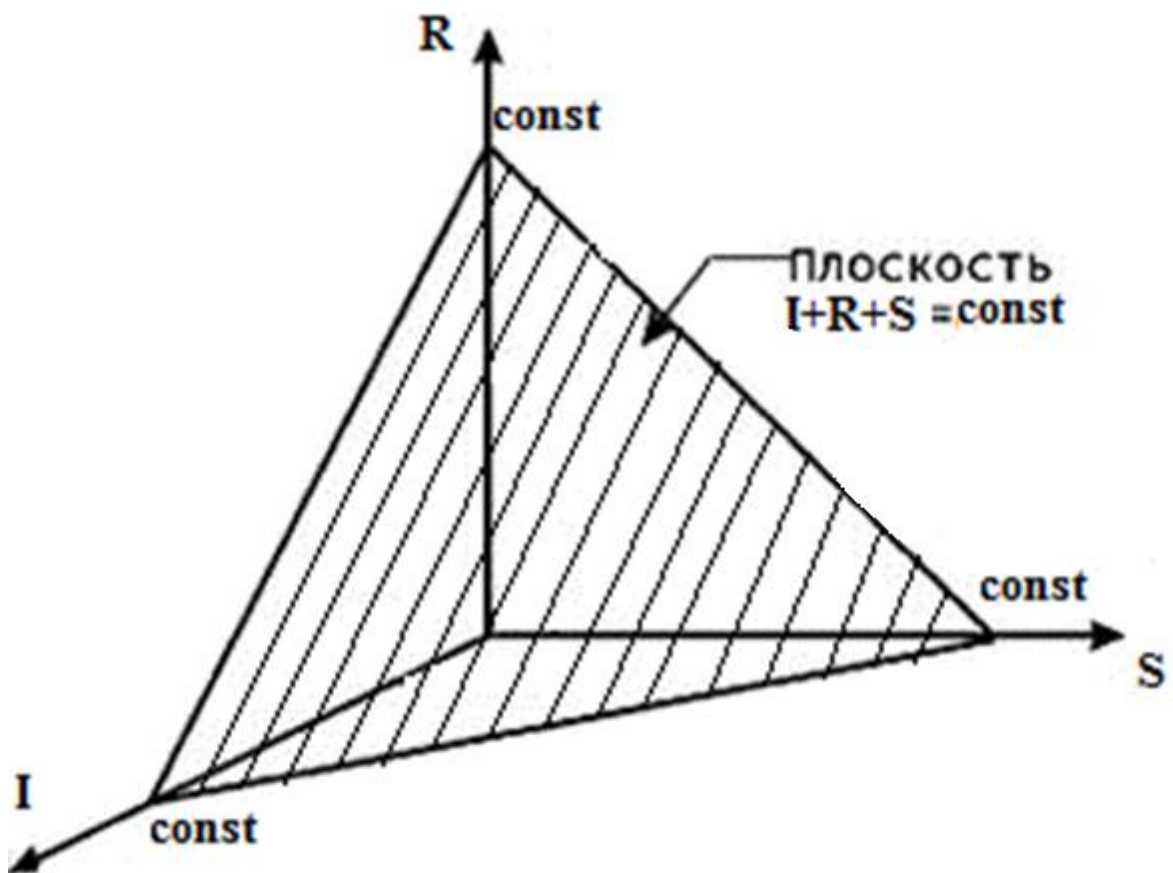


Рис. 2.7. Изображение плоскости.

Необходимо получить изображение нагрузки, для этого для каждой точки треугольника необходимо знать его координаты в пространстве IRS. Следовательно, необходимо преобразование экранных (двухмерных) координат треугольника в координаты трехмерные. Для решения этой задачи необходимо найти матрицу перехода.

Рассмотрим переход от системы экранных координат $d=(x, y, z)$ к системе координат $IRS=(insert,remove,select)$. Третья координата в системе d всегда будет равна 0. Пусть X_{IRS} – координаты вектора в системе IRS , а X_d – координаты того же вектора X в экранной системе. Тогда справедливо следующее $X_{IRS} = M \cdot X_d$, где M – матрица перехода.

Матрицу перехода можно получить как произведение простейших матриц преобразования координат (поворота и сдвига). $M=M_1*M_2*\dots*M_n$.

В данном случае необходимо использовать три матрицы: матрицу поворота по оси R (рис. 2.8), матрицу поворота по оси S (рис. 2.9), матрицу сдвига по оси I (рис. 2.10).

$$[R_y] = \begin{bmatrix} \cos \psi & 0 & -\sin \psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \psi & 0 & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Рис. 2.8. Матрица поворота вокруг оси y на угол альфа

$$[R_z] = \begin{bmatrix} \cos \chi & \sin \chi & 0 & 0 \\ -\sin \chi & \cos \chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Рис. 2.9. Матрица поворота вокруг оси z на угол альфа

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{bmatrix}$$

Рис. 2.10. Матрица сдвига

После проведения вычислений было определено, что необходим поворот вокруг оси $Remove$ на угол $35,264^\circ$, по оси $Select$ на угол -45° , и сдвиг по оси $Insert$ на величину $\text{const} * \sqrt{2} / 2$.

Окончательно получаем матрицу перехода $M = M_{\text{Remove}}(35, 264) * M_{\text{Select}}(-45) * T(\text{const} * \sqrt{2} / 2, 0, 0)$.

2.6.3. Вычисление количества простейших операций

Для определения количества простейших операций сравнения и присваивания было решено использовать существующую программу. Для этого необходимо создать нагрузку по количеству вставок, удалений и поисков (найденных из экранных координат текущего пикселя).

После этого необходимо указать контейнеры, на которых проводится тестирование и вызвать необходимый метод внешнего модуля.

2.6.4. Определение цвета пикселя

После того, как для каждого пикселя получен набор чисел (количество простейших операций для каждого хранилища), необходимо определить цвет пикселя. Для этого используются два алгоритма:

1. Если тестируется одно хранилище (для каждого пикселя одно число), то используется метод линейного градиента. Сначала определяется минимальное и максимальное значение среди всех пикселей. Потом получаем коэффициент пикселя из диапазона $[0; 1]$ по формуле (2.2):

$$Ratio = (value - min) / (max - min) \quad (2.2)$$

В итоге цвет определяется по формуле (2.3):

$$Color = colorMax * ratio + colorMin (1 - ratio) \quad (2.3)$$

2. Если тестируется больше одного хранилища, то для каждого хранилища задается свой цвет. Среди набора чисел для каждого пикселя ищется минимальный, и выбирается цвет соответствующего хранилища.

2.7. Классификация

В случае двух контейнеров необходимо провести классификацию. Под классификацией будем понимать признак, по которому можно сказать, что

определенная точка в пространстве IRS лучше подходит для одного контейнера, чем для другого.

Задача рассматривается в плоскости треугольника $\text{Insert}+\text{Remove}+\text{Select}=\text{const}$.

Простейшим примером классификатора является прямая, по одну сторону от которой оптимальным будет являться один контейнер, а по другую – второй. Следовательно, для решения поставленной задачи необходимо найти коэффициенты a, b, c уравнения (2.4):

$$ax + by + c = 0 \quad (2.4)$$

Для того чтобы решить данную задачу был выбран метод опорных векторов (англ. SVM, support vector machine) [15]. В качестве ядра SVM была выбрана линейная функция. Основной идеей этого метода является нахождение разделяющей гиперплоскости таким образом, что расстояние от гиперплоскости до ближайшей точки максимально среди всех таких плоскостей. Это приводит к более точной классификации и к меньшей средней ошибке классификации.

Для решения этой задачи была выбрана существующая реализация – библиотека `libsvm`. На вход она принимает список точек с уже известными классами, а на выходе будут коэффициенты a, b, c уравнения (4).

При большом размере выходного изображения точек для классификации получается слишком много, а на точность классификации такое количество сильно не влияет. Следовательно, для ускорения программы можно выбирать для классификации не все точки, а лишь определенные. Было решено брать точки на границе треугольника, а также точки расположенные в узлах сетки определенного размера, как изображено на рис. 2.11. На изображении зеленым цветом помечены те точки, для которых оптимальным будет одно хранилище, а красным – другое. Помимо этого, на изображении присутствует разделяющая линия, полученная в результате работы алгоритма SVM. При использовании

адаптивного хранилища для всех точек выше этой линии будет использоваться один контейнер, для всех ниже – другой.

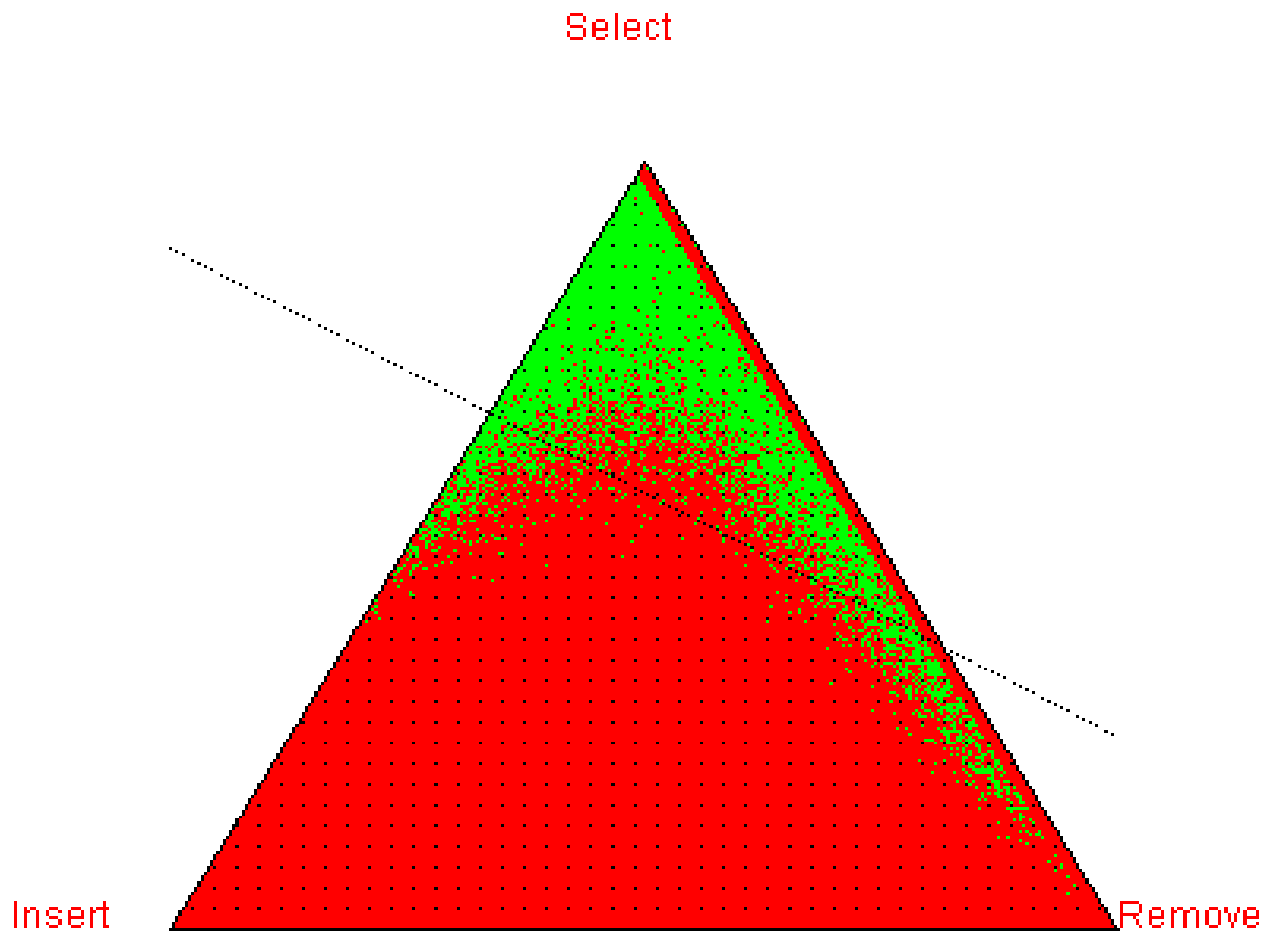


Рис. 2.11. Точки, выбираемые для классификации (все черные точки, за исключением разделяющей прямой)

3. Средства реализации

Для реализации поставленной задачи был выбран объектно-ориентированный язык программирования Java и использованы следующие инструментальные средства и технологии:

- среда разработки Eclipse Luna Service Release 2 (4.4.2) [12][13];
- язык программирования Java 8 [10];
- библиотека для работы с xml-файлами jdom2;
- библиотека логирования log4j;
- библиотека для решения задачи SVM libsvm [14].

4. Требования к аппаратному и программному обеспечению

Для функционирования программного обеспечения необходимо выполнение следующих требований к аппаратному и программному обеспечению:

- процессор не ниже Pentium IV 2.6 GHz;
- оперативная память размером не менее 1024 Мб;
- не менее 1 Гб свободного дискового пространства;
- операционная система Windows XP/Vista/Seven/Server;
- виртуальная машина Java версии 1.8.

5. Интерфейс пользователя

Программный комплекс предоставляет консольный интерфейс для пользователя. Все действия сопровождаются подсказками. При запуске программы пользователь увидит консоль, изображенную на рисунке 5.1.

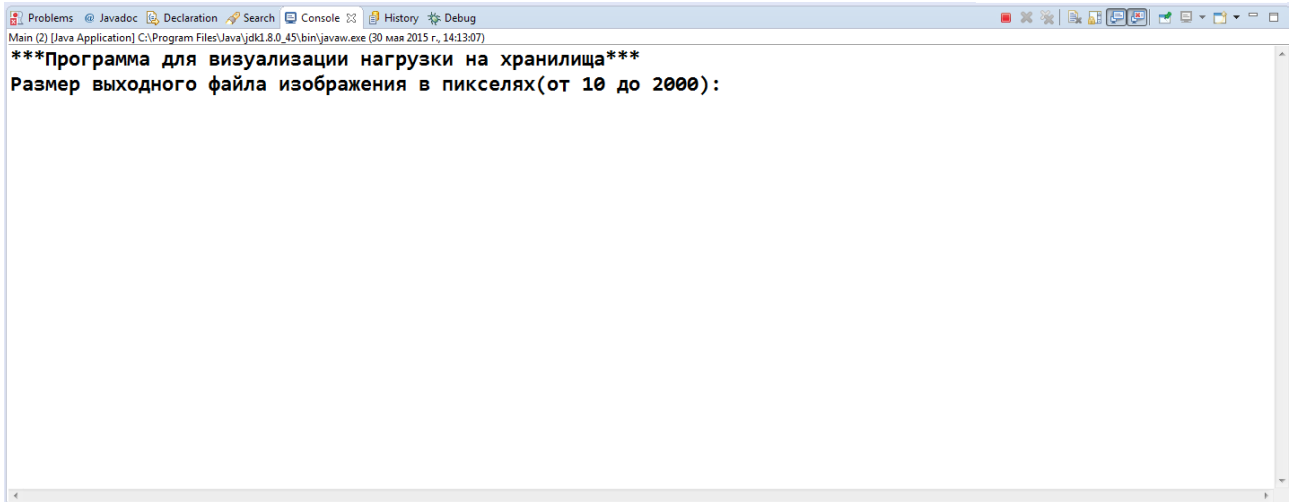


Рис. 5.1. Вид консоли при запуске

После этого пользователь должен ввести размер изображения в пикселях. Это должно быть целое число от 10 до 2000. В случае некорректного ввода пользователь увидит консоль, изображенную на рисунке 5.2. Данное сообщение будет возникать до тех пор, пока не будет введено корректное значение.

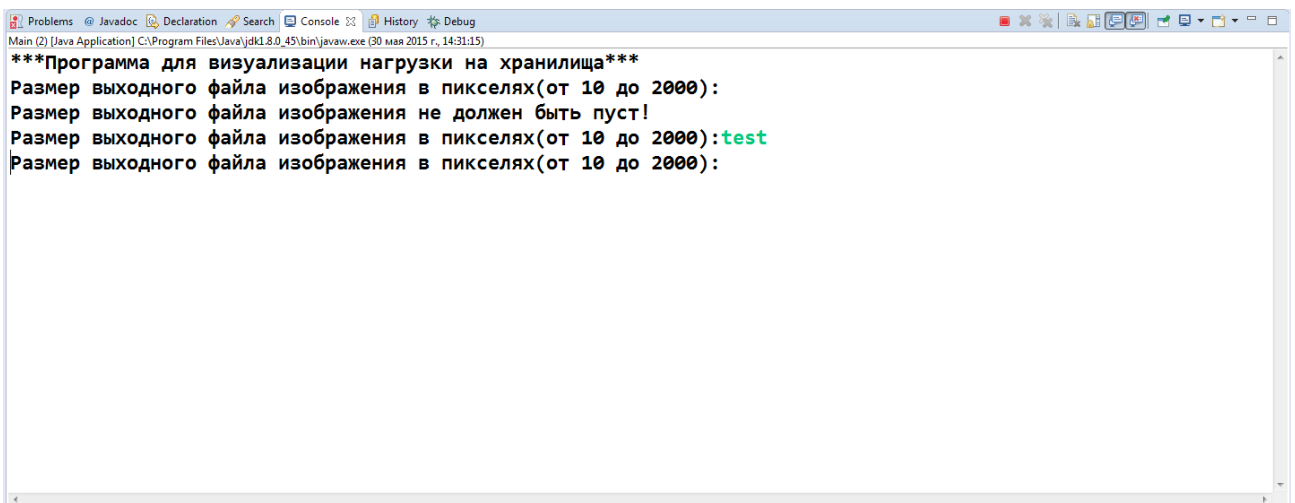


Рис. 5.2. Вид консоли при некорректном вводе размера изображения

После корректного ввода пользователь увидит консоль, изображенную на рисунке 5.3. Здесь пользователь должен ввести название выходного файла изображения.

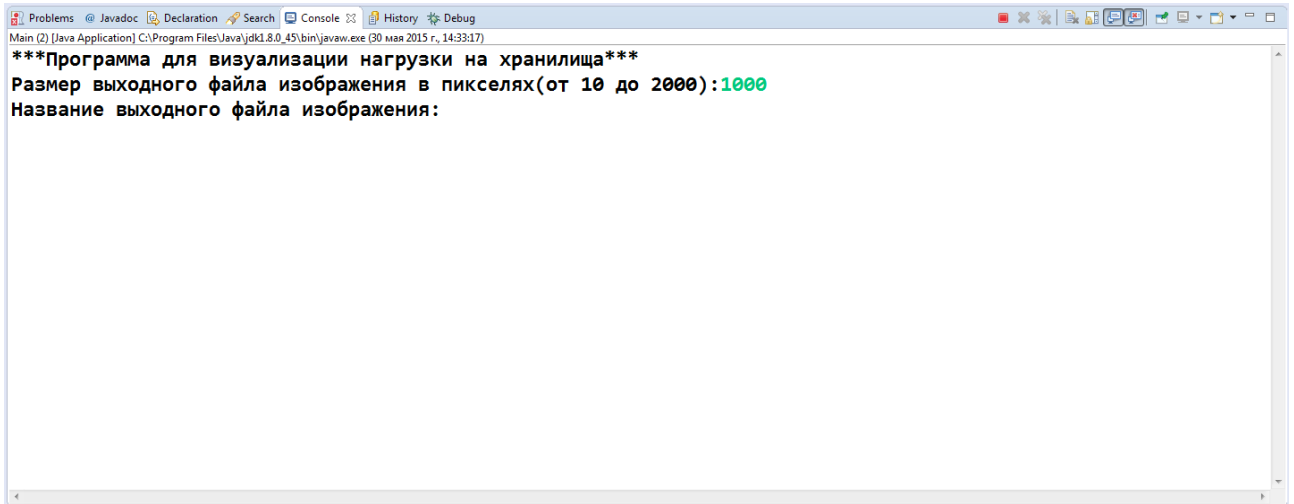


Рис. 5.3. Вид консоли после ввода размера изображения

Далее пользователю будет предложено ввести название класса контейнера, который будет тестироваться. Вид консоли при этом будет выглядеть как показано на рисунке 5.4.

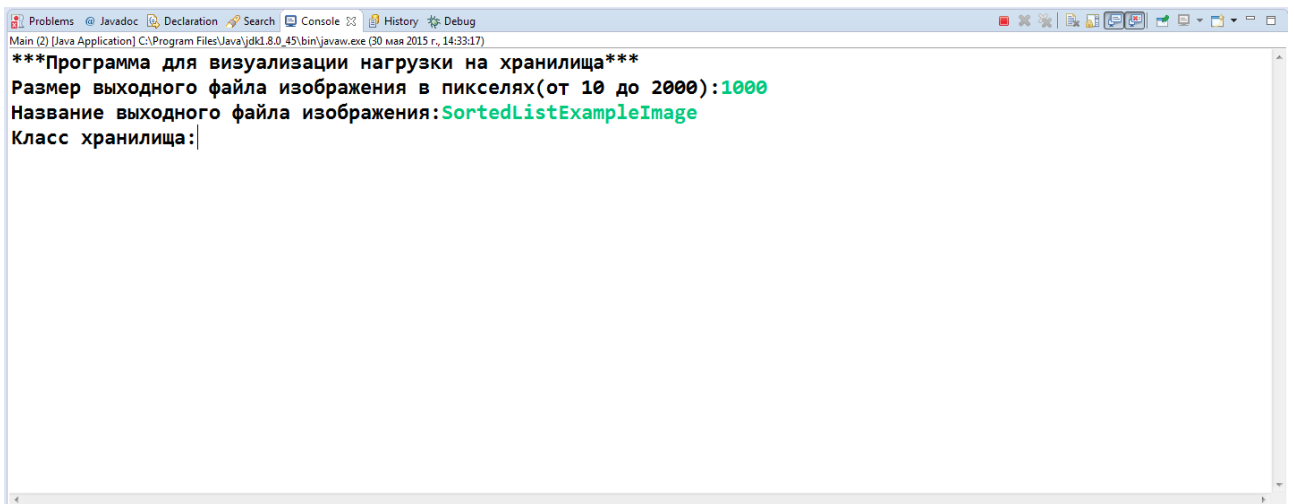


Рис. 5.4. Вид консоли после ввода названия изображения

Здесь необходимо ввести класс контейнера, при этом для контейнеров, располагающихся в пакете `com.vsu.amm.data.storage` можно вводить только названия классов, указанных в таблице 1, для остальных же классов необходимо вводить полное название, включающее пакет, например,

`com.vsu.amm.data.storage.SortedList`. После ввода класса контейнера консоль будет выглядеть как на рисунке 5.5.

Таблица 1. Список встроенных контейнеров

Название контейнера	Название класса
Простой список	<code>SimpleList</code>
Сортированный список	<code>SortedList</code>
Простой массив	<code>SimpleArray</code>
Сортированный массив	<code>SortedListArray</code>
В-дерево	<code>BTree</code>
Двоичное дерево	<code>BinaryTree</code>
Хеш-таблица	<code>HashTable</code>

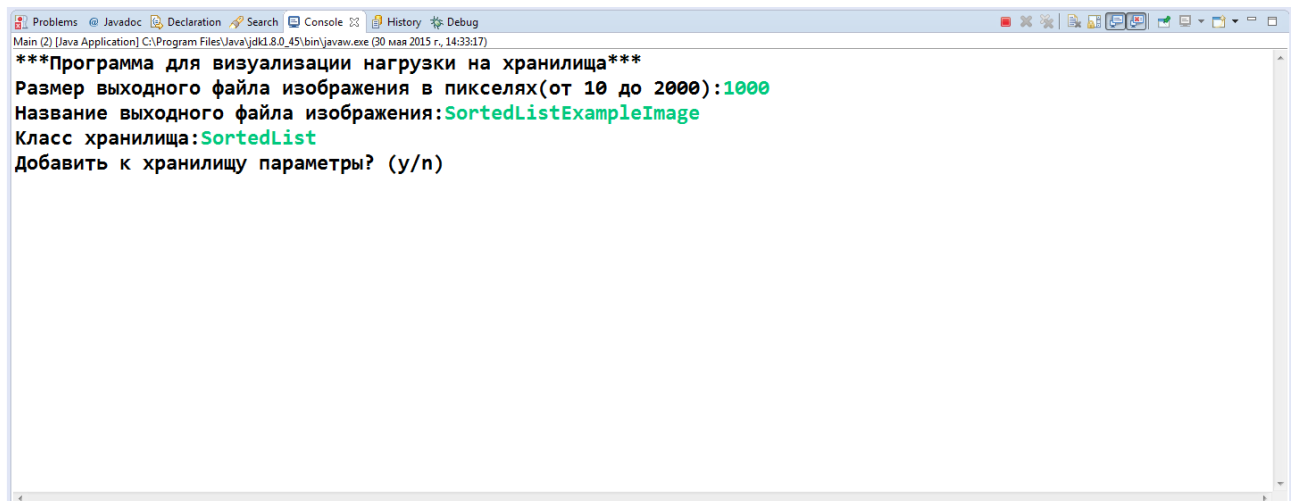


Рис. 5.5. Вид консоли после ввода названия класса контейнера

Здесь будет предложено добавить к контейнеру параметры, а после этого добавить еще один контейнер. После ввода всех контейнеров и их параметров, появится сообщение о начале тестирования контейнеров и создания файла с указанным размером и названием, как показано на рисунке 5.6.

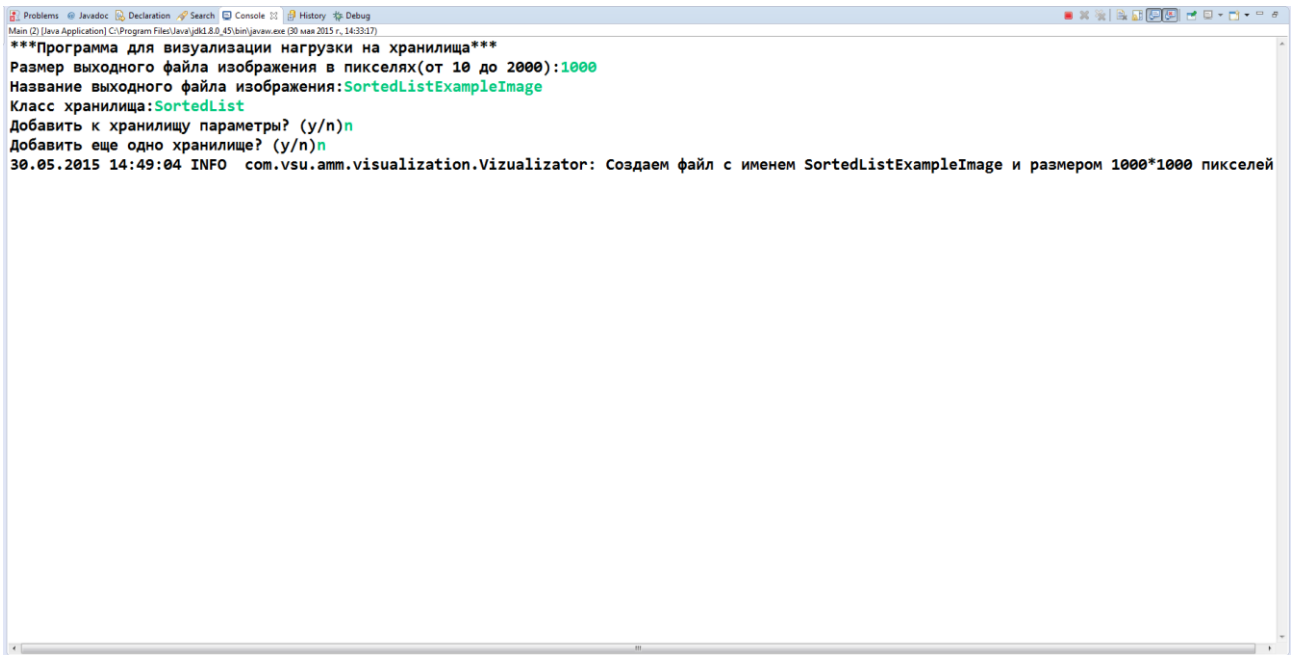


Рис. 5.6. Вид консоли после ввода всех контейнеров

Спустя некоторое время, которое зависит от производительности компьютера пользователя и введенного размера изображения, появится сообщение об окончании создания изображения. Окончательный вид консоли изображен на рисунке 5.7.

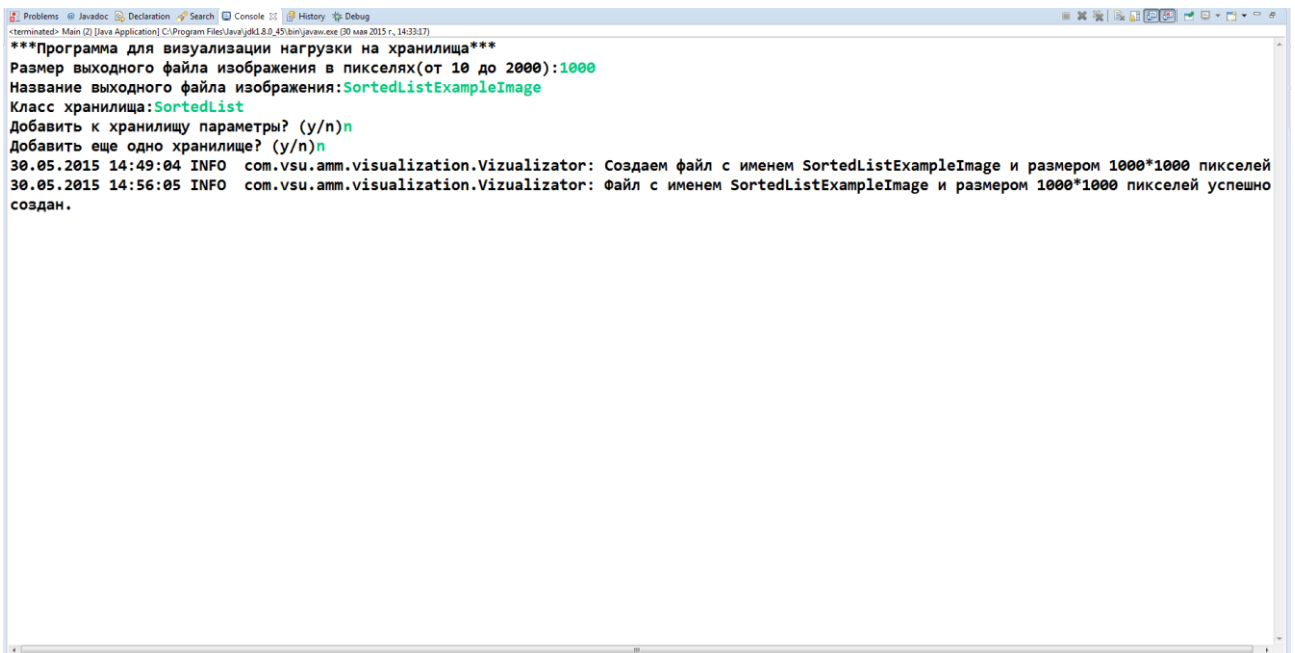
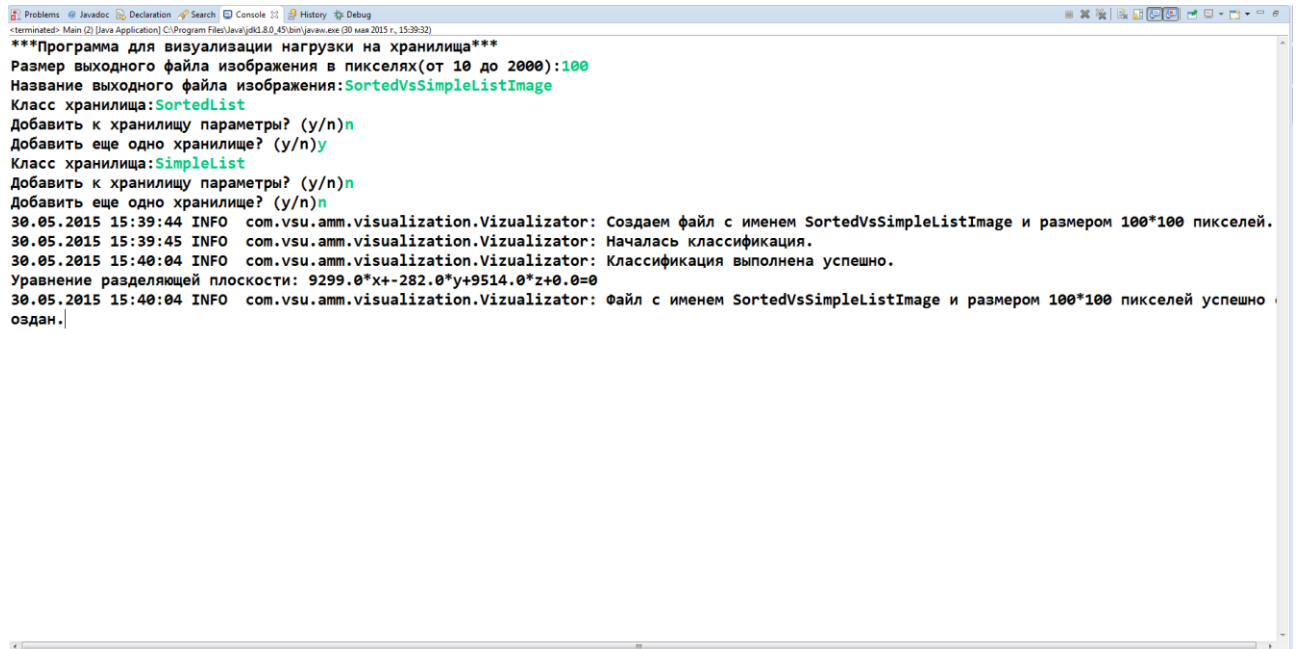


Рис. 5.7. Вид консоли после создания изображения

В случае если пользователь ввел два контейнера, на консоли помимо стандартных сообщений будут сообщения о проведении классификации и уравнение разделяющей плоскости. Пример изображен на рисунке 5.8.



```

***Программа для визуализации нагрузки на хранилища***
Размер выходного файла изображения в пикселях(от 10 до 2000):100
Название выходного файла изображения:SortedVsSimpleListImage
Класс хранилища:SortedList
Добавить к хранилищу параметры? (y/n)n
Добавить еще одно хранилище? (y/n)y
Класс хранилища:SimpleList
Добавить к хранилищу параметры? (y/n)n
Добавить еще одно хранилище? (y/n)n
30.05.2015 15:39:44 INFO com.vsu.amm.visualization.Vizualizator: Создаем файл с именем SortedVsSimpleListImage и размером 100*100 пикселей.
30.05.2015 15:39:45 INFO com.vsu.amm.visualization.Vizualizator: Началась классификация.
30.05.2015 15:40:04 INFO com.vsu.amm.visualization.Vizualizator: Классификация выполнена успешно.
Уравнение разделяющей плоскости: 9299.0*x+-282.0*y+9514.0*z+0.0=0
30.05.2015 15:40:04 INFO com.vsu.amm.visualization.Vizualizator: Файл с именем SortedVsSimpleListImage и размером 100*100 пикселей успешно
создан.

```

Рис. 5.8. Вид консоли после создания изображения в случае двух контейнеров

После появления этого сообщения на консоли, в директории, откуда производился запуск программы, появится файл с названием введенным пользователем и расширением png. Пример такого файла изображен на рисунке 5.9.

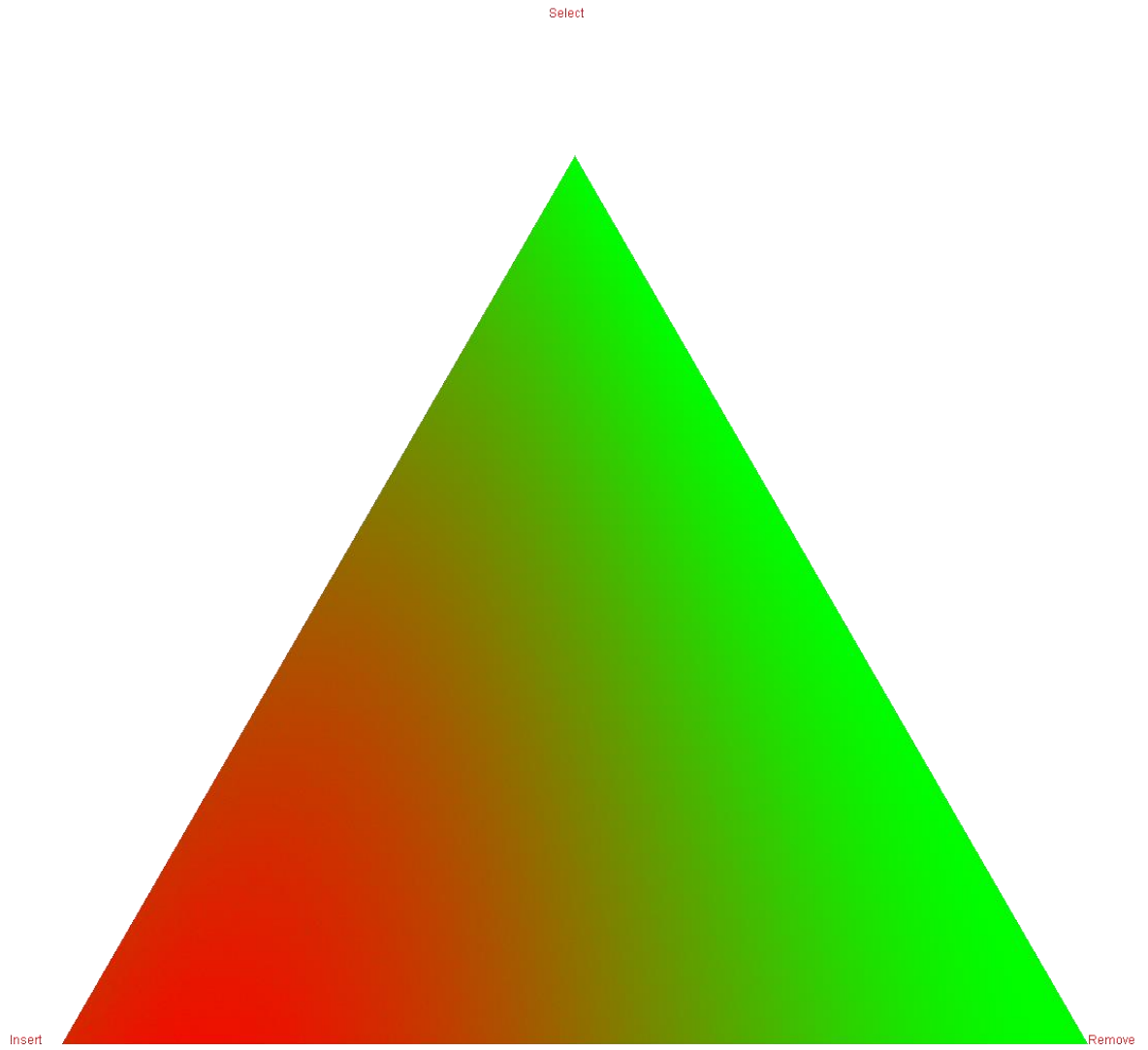


Рис. 5.9. Пример выходного файла для контейнера SortedList

6. Реализация

6.1. Общая схема работы программы. Форматы входных и выходных данных

Общая схема работы программы представлена на рис. 6.1. Входные данные вводятся с консоли. Входные данные содержат:

1. Размер выходного файла изображения в пикселях.
2. Название выходного файла изображения.
3. Список названий классов контейнеров.
4. Список параметров контейнеров.

Выходные данные содержат:

1. Png-файл содержащий изображение.
2. В случае двух контейнеров – уравнение разделяющей плоскости.



Рис. 6.1. Общая схема работы программы

6.2. Общая архитектура программы

Внутренняя схема работы программы представлена на рисунке 6.2. Здесь отображены основные модули программы и связи между ними.

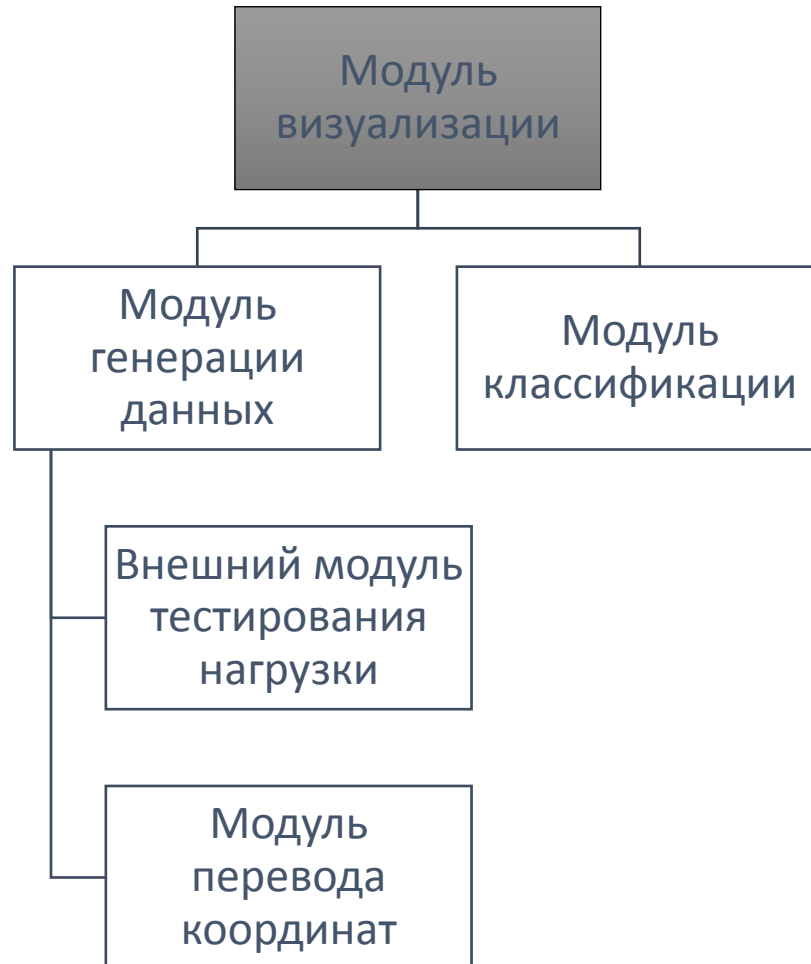


Рис. 6.2. Общая схема работы программы

Эти модули реализованы в виде java-классов [9]. Диаграмму этих классов можно увидеть на рисунке 6.3. Основные модули и соответствующие им классы:

1. Модуль визуализации (класс `Vizualizator`).
2. Модуль генерации данных (класс `DataGenerator`).
3. Модуль классификации (класс `Classifier`).
4. Модуль перевода координат (класс `CoordinateTranslator`).
5. Внешний модуль тестирования нагрузки (класс `DataSetPlayer`).

Далее рассмотрим эти модули подробнее.

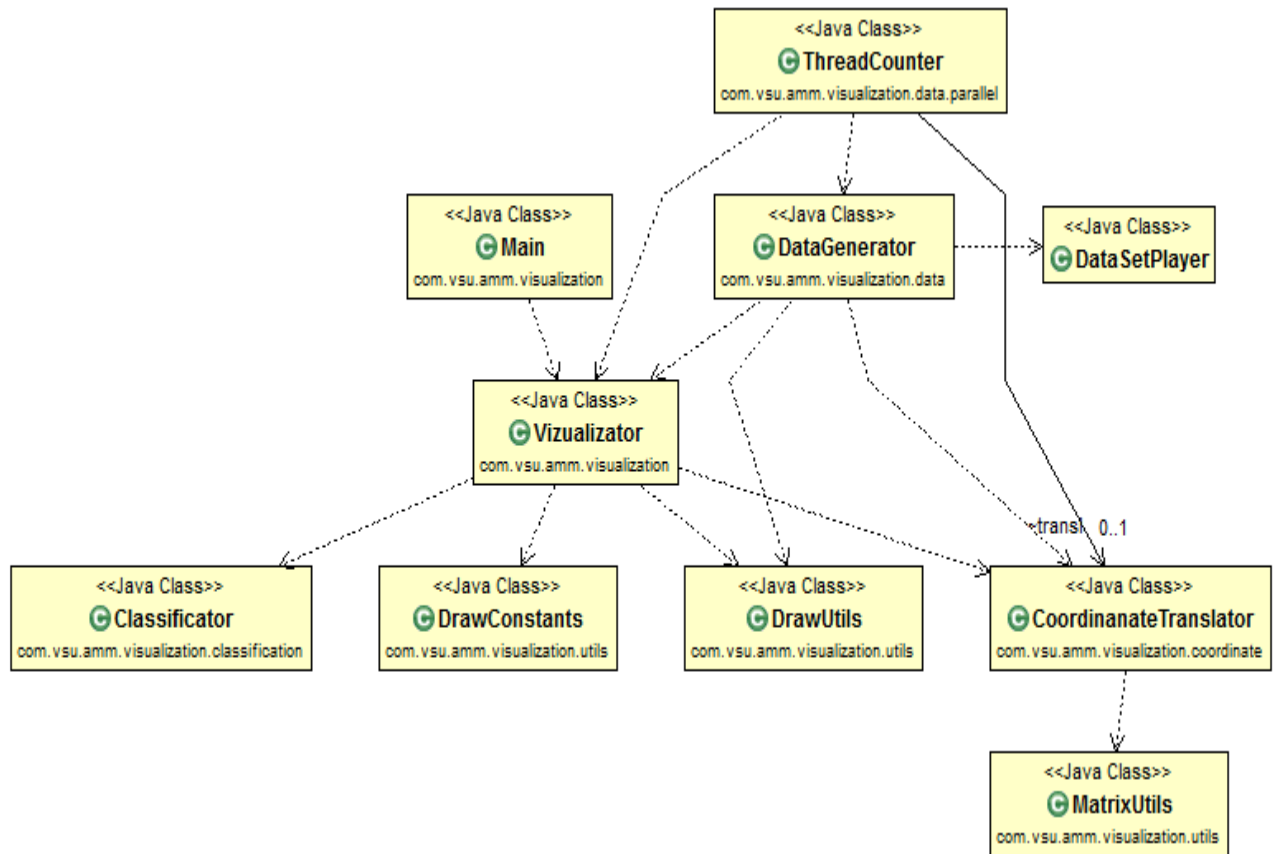


Рис. 6.3. Диаграмма основных классов

6.3. Модуль визуализации

Модуль реализует логику визуализации нагрузки на контейнер.

Реализация – класс `Vizualizator`. Основной алгоритм таков:

1. Получаем список координат и значений из модуля генерации данных.
2. Для каждой точки определяем цвет.
3. В случае двух контейнеров добавляем точку к данным для классификации и производим вызов модуля классификации.
4. Наносим на изображении треугольник, оси и разделяющую линию.

Основные методы модуля:

1. `public static void Draw(int size, String filename, List<IDataStorage> storages)` – метод производит валидацию входных данных, создает изображение.
2. `private static void drawImage(BufferedImage image, int size, List<IDataStorage> storages)` – метод производит получение точек из модуля генерации данных, отрисовку этих точек и если необходимо вызов классификатора.
3. `private static void drawAxis(BufferedImage image, int size)` – метод наносит название осей на изображение.
4. `private static Color getPointColor(Point point, Integer mi, double curRange)` – метод определяет цвет точки исходя из ее значения.
5. `private static void addPointForClassification(List<Point> classificationPoints, Point point, Integer size)` – метод определяет необходимо ли добавлять точку к данным для классификации.

6. `private static void drawClassifierLine(List<Point> classificationPoints, BufferedImage image, Integer size)` – метод вызывает модуль классификации и наносит разделяющую линию на изображение.

Диаграмма используемых модулей и вспомогательных классов представлена на рисунке 6.4.

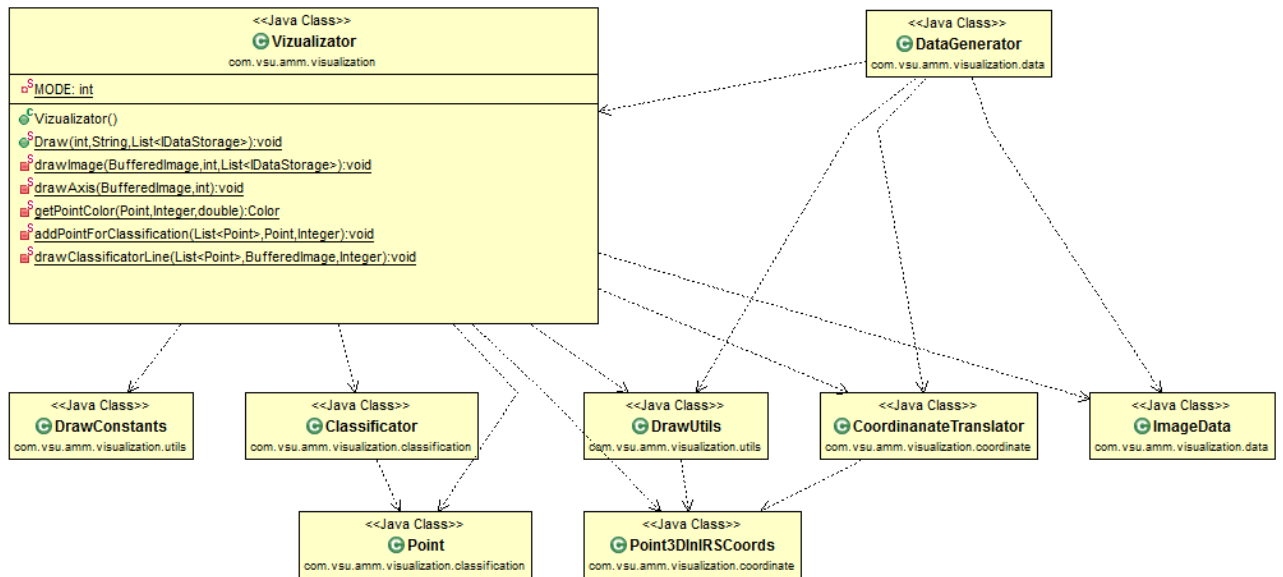


Рис. 6.4. Диаграмма используемых классов

6.4. Модуль генерации данных

Модуль реализует логику получения точек для отрисовки и количества простейших операций в каждой точке. Реализация – класс `DataGenerator`.

Основной алгоритм таков:

1. Проходим по всем точкам изображения и определяем лежит ли она в треугольнике.
2. Из всех точек, лежащих в треугольнике, формируем задания по количеству процессоров (для поддержки многопоточности).
3. В каждом задании для каждой точки получаем координаты соответствующей точки в пространстве `InsertSelectRemove` (посредством вызова модуля перевода координат), формируем нагрузку с таким же количеством операций.
4. Для каждой точки в задании прогоняем полученную нагрузку.
5. Ищем минимум и максимум для случая одного контейнера.
6. Возвращаем все точки со значениями и минимум, максимум.

Основные методы модуля:

1. `public static List<Integer> getContersForStorages(List<IDataStorage> storages, Integer insertCount, Integer selectCount, Integer removeCount)` – метод создает нагрузку и производит вызов внешнего модуля для ее прогонки. После этого производит подсчет количества простейших операций.
2. `public static ImageData getCoeffs(int size, List<IDataStorage> storages)` – метод, координирующий работу заданий по обработке точек и формирующий конечный список с точками и минимум с максимумом.
3. `private static List<Future<ThreadCounterResult>> generateTasks(int size, List<IDataStorage> storages, ExecutorService executor, int PROCESSORS_COUNT)` – метод,

который определяет принадлежность точки треугольнику и формирующий задания по их обработке.

Диаграмма используемых модулей и вспомогательных классов представлена на рисунке 6.5.

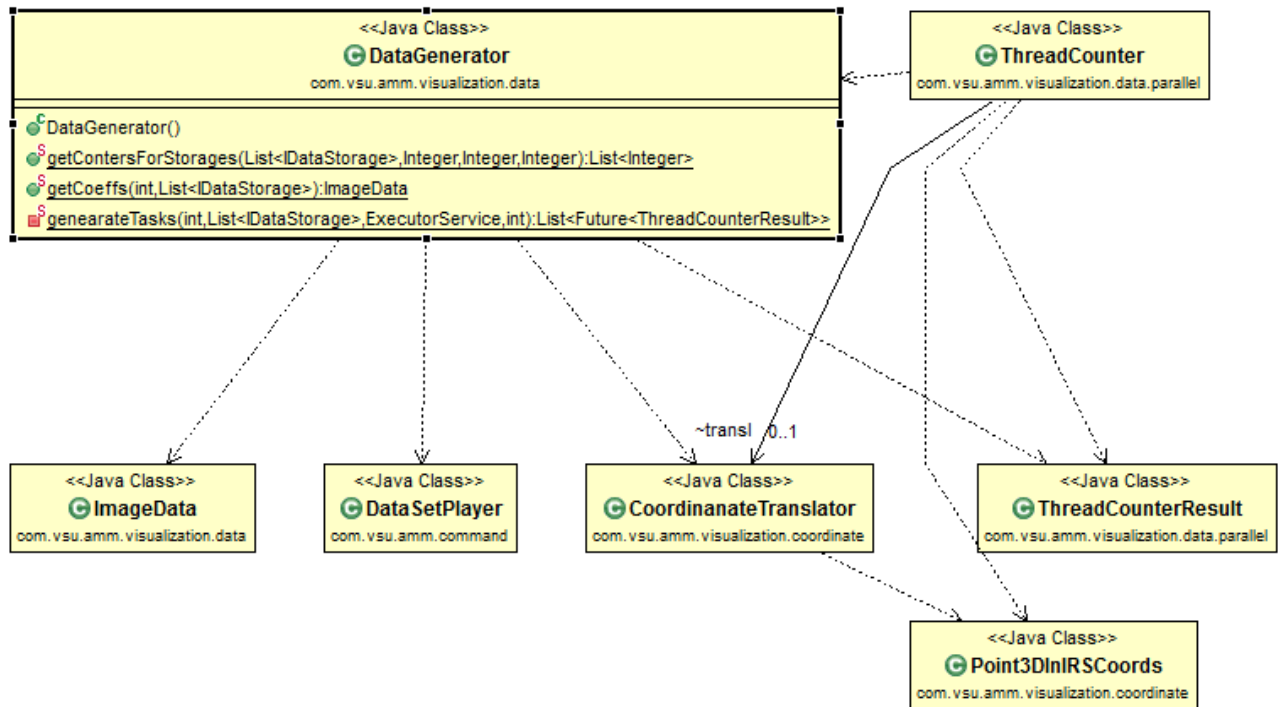


Рис. 6.5. Диаграмма используемых классов

Для реализации многопоточности был выбран стандартный механизм языка java [11]. Он предоставляет планировщик потоков `ExecutorService`, интерфейс `Callable`, на основе которого реализуется задание, и интерфейс `Future`, реализация которого является результатом асинхронного выполнения `Callable`.

Асинхронное задание реализовано в виде класса `public class ThreadCounter implements Callable<ThreadCounterResult>`, его главный метод `public ThreadCounterResult call() throws Exception` — в асинхронном режиме для всех точек задания получает координаты соответствующей точки в пространстве `InsertSelectRemove` (посредством вызова модуля перевода координат), формирует нагрузку с таким же количеством операций и прогоняет ее на всех контейнерах. Результат

асинхронного выполнения задания хранится в виде списка точек и их значений, а также минимума и максимума по значениям. Реализован в классе `public class ThreadCounterResult`.

6.5. Модуль перевода координат

Модуль реализует логику перевода из экранных (двухмерных) координат в координаты пространства InsertRemoveSelect (трехмерные). Реализация – класс `CoordinateTranslator`.

Основные методы модуля:

1. `public CoordinateTranslator(double size)` – метод, который создает матрицу преобразования.
2. `public Point3DInIRSCoords translate(Point2D point)` – метод преобразования двухмерных в трехмерные координаты. Применяет матрицу преобразований к указанной точке.

Диаграмма используемых модулей и вспомогательных классов представлена на рисунке 6.6.

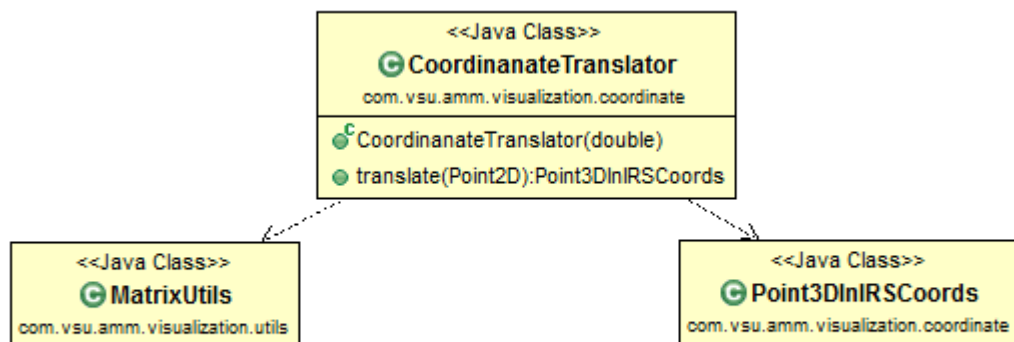


Рис. 6.6. Диаграмма используемых классов

6.6. Внешний модуль для тестирования нагрузки

Модуль используется для создания нагрузки и ее прогонки на контейнерах. Используются два класса: `SourceGenerator` и `DataSetPlayer`.

Основные методы классов модуля:

1. `public static ICommandSource createInsertSelectRemoveSource(Integer insertCount, Integer selectCount, Integer removeCount)` – метод, который создает нагрузку с указанным числом вставок, удалений и поисков.
2. `public void play(ICommandSource commandSource)` – метод, который прогоняет указанную нагрузку и подсчитывает число простейших операций сравнения и присваивания.

6.7. Модуль классификации

Модуль предназначен для проведения классификации. Реализация – класс `Classifier`. Основная цель модуля – получить коэффициенты a, b, c уравнения $ax+by+c$ для разделяющей прямой. Эту задачу можно решить методом опорных векторов (SVM). Для решения этой задачи была выбрана библиотека `libsvm`.

Основной алгоритм таков:

1. Проходим по всем точкам изображения и определяем нужно ли ее брать для классификации. Для этого проверяется лежит ли она на стороне треугольника или на пересечении сетки определенного размера. Для всех этих точек находится к какому классу(контейнеру) она относится. Например, для первого контейнера класс будет иметь метку 0, а для второго 1. Все точки, на основе которых проводится классификация передаются методу `public static double[] classification(List<Point> classificationPoints)`.
2. Создаем параметры классификации – объект класса `svm_parameter`. Устанавливаем тип классификации – линейный, а также остальные настройки.
3. Создаем проблему, которую будем решать – объект класса `svm_problem`. Заполняем его точками классификации и их метками.
4. Создаем модель решения: `svm_model model = svm.svm_train(problem, param)`.
5. Извлекаем из модели коэффициенты a, b, c и возвращаем их в виде массива.

Диаграмма используемых модулей и вспомогательных классов представлена на рисунке 6.7.

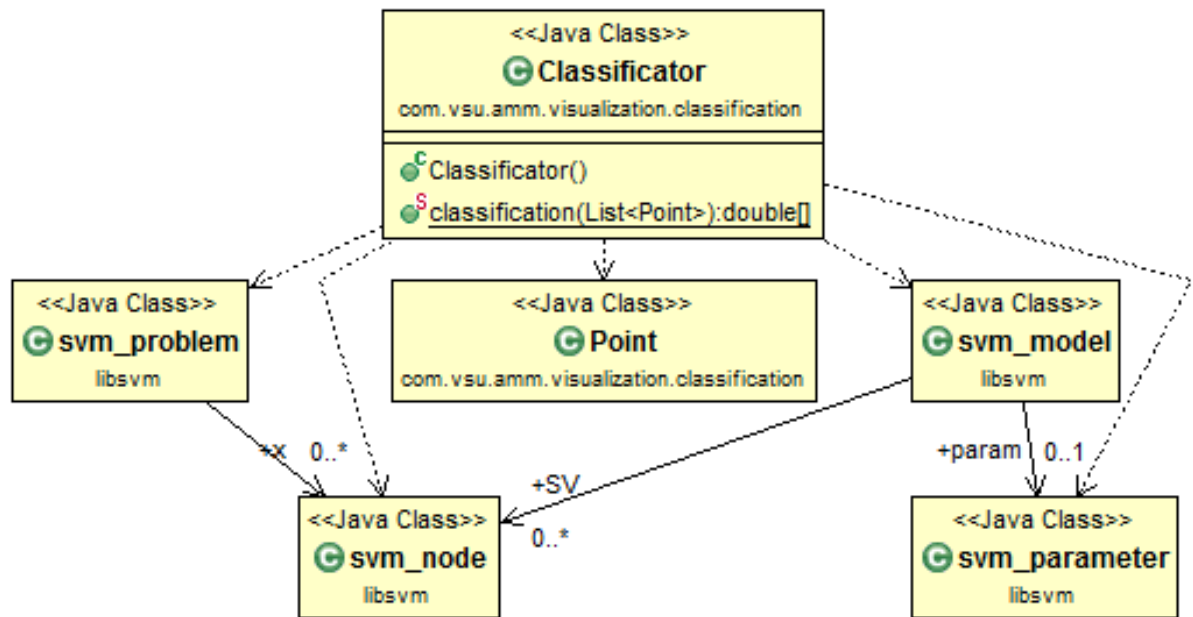


Рис. 6.7. Диаграмма используемых классов

6.8. Вспомогательные классы.

Кроме основных классов, есть классы, используемые для переноса информации между модулями и классы-утилиты, содержащие вспомогательные методы.

Классы для хранения информации:

1. `Point` – содержит координаты x и y , а также значение в этой точке.
2. `Point3DInIRSCoords` – содержит целочисленные трехмерные координаты в пространстве (`insert`, `remove`, `select`).
3. `ImageData` – класс для хранения списка точек и значений количества операций для нанесения на изображение, а также минимум и максимум по количеству операций.

Классы-утилиты:

1. `DrawUtils` – содержит вспомогательные методы для визуальной части:
 - а) `public static float side(float startX, float startY, float finishX, float finishY, float x, float y)` – метод для определения по какую сторону от прямой расположена точка;
 - б) `public static Boolean pointInTriangle(float xA, float yA, float xB, float yB, float xC, float yC, float x, float y)` – метод для определения принадлежности точки треугольнику с указанными координатами;
 - в) `public static Boolean pointInTriangleSide(float xA, float yA, float xB, float yB, float xC, float yC, float x, float y)` – метод для определения принадлежности точки стороне треугольника с указанными координатами;
 - г) `public static Color getColorByLinearGradient(double ratio, Color colorMin, Color colorMax)` – метод для определения цвета пикселя методом линейного градиента по цвету минимального и максимального значения и коэффициенту текущей точки по формуле $\text{colorMax} * \text{ratio} + \text{colorMin} (1 - \text{ratio})$;

д) `public static double evalLineFunction(double[] w, double x, double y)` – метод для вычисления значения функции $f(x,y)=ax+by+c$;

е) `public static double evalLineFunctionYValue(double[] w, double x)` – метод для вычисления значения функции $f(x)=ax+b$;

ж) `public static double[] getPlaneCoeffsByThreePoints(Point3DInIRSCoords r1, Point3DInIRSCoords r2, Point3DInIRSCoords r3)` – метод для нахождения коэффициентов плоскости по трем точкам.

2. `MatrixUtils` – содержит вспомогательные методы для перевода координат, методы работы с матрицей:

а) `public static double[][] multiplyByMatrix(double[][] firstMatrix, double[][] secondMatrix)` – метод для умножения матрицы на матрицу;

б) `public static double[][] rotateOnAngleByZ(double alpha)` – метод для получения матрицы поворота на угол α по оси Z;

в) `public static double[][] rotateOnAngleByY(double alpha)` – метод для получения матрицы поворота на угол α по оси Y;

г) `public static double[][] moveOnXYZ(double x, double y, double z)` – метод для получения матрицы сдвига системы координат на (x,y,z) .

7. Тестирование

Эксперимент 1. Простой список.

Описание: исследование зависимости суммарного количества операций сравнения и присваивания от соотношения количества операций вставки, поиска и удаления на простом списке.

Суммарное число операций вставки, поиска и удаления: 1000.

Класс исследуемого контейнера: `SimpleList`.

Результат: результат работы программы представлен на рис. 7.1. На изображении видно, что контейнер хорошо подходит для большого количества операций поиска и удаления при малом количестве вставок, а также при большом количестве вставок и почти полном отсутствии операций удаления и поиска.

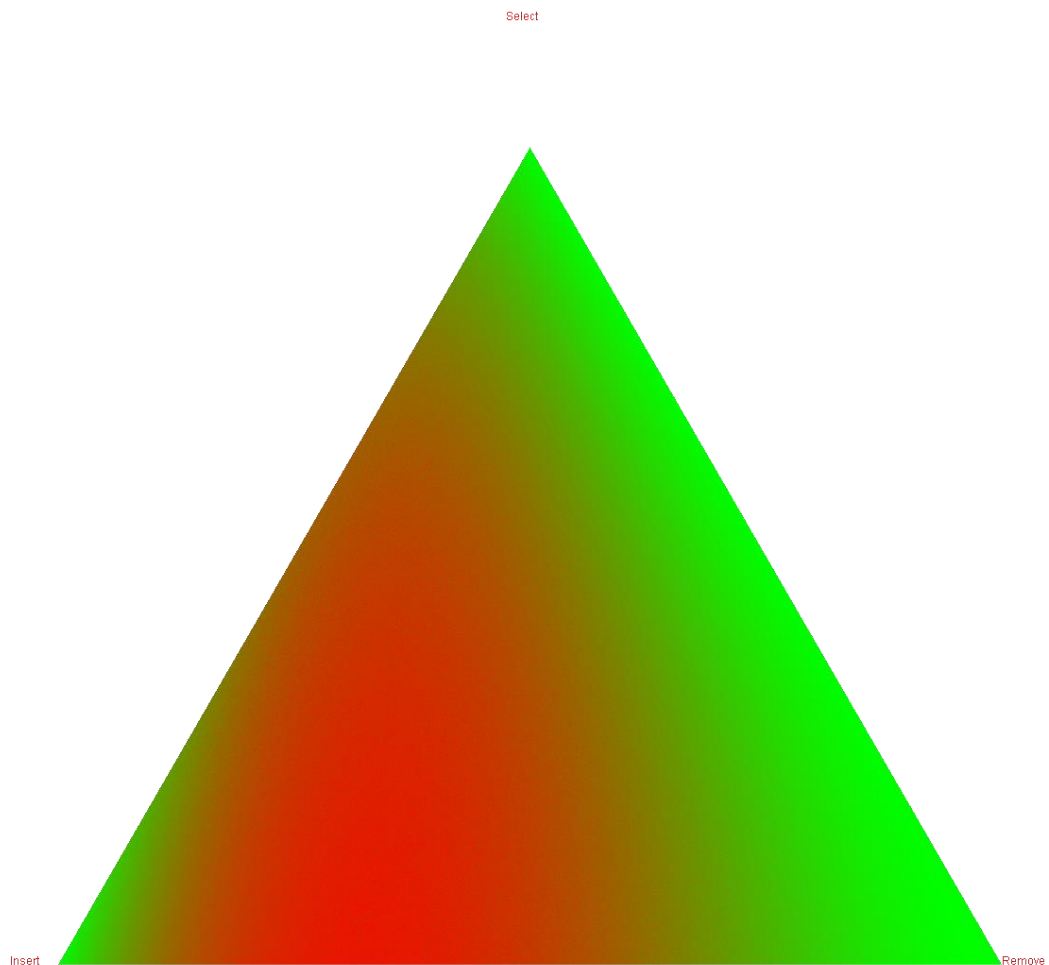


Рис. 7.1. Результат исследования простого списка

Эксперимент 2. Сортированный список.

Описание: исследование зависимости суммарного количества операций сравнения и присваивания от соотношения количества операций вставки, поиска и удаления на сортированном списке.

Суммарное число операций вставки, поиска и удаления: 1000.

Класс исследуемого контейнера: `SortedList`.

Результат: результат работы программы представлен на рис. 7.2. На изображении видно, что контейнер очень хорошо подходит для большого и среднего количества операций поиска и удаления при малом количестве вставок.

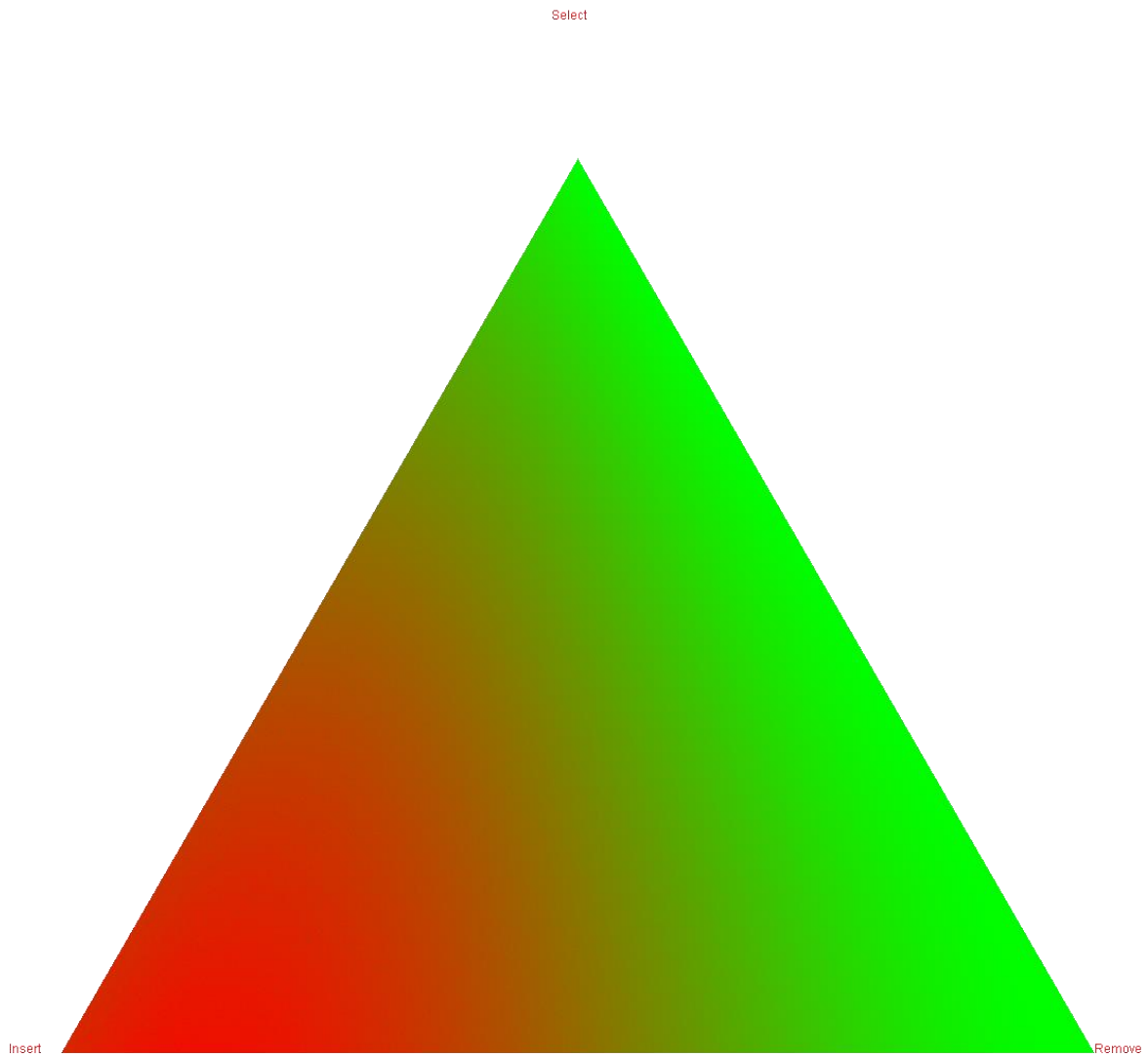


Рис. 7.2. Результат исследования сортированного списка

Эксперимент 3. В-дерево.

Описание: исследование зависимости суммарного количества операций сравнения и присваивания от соотношения количества операций вставки, поиска и удаления на В-дереве.

Суммарное число операций вставки, поиска и удаления: 1000.

Класс исследуемого контейнера: BTree.

Результат: результат работы программы представлен на рис. 7.3. На изображении видно, что контейнер плохо подходит для количества операций вставки примерно равном количеству операций поиска при малом количестве удалений. В остальных случаях контейнер хорошо подходит для различных соотношений операций вставки, удаления и поиска.

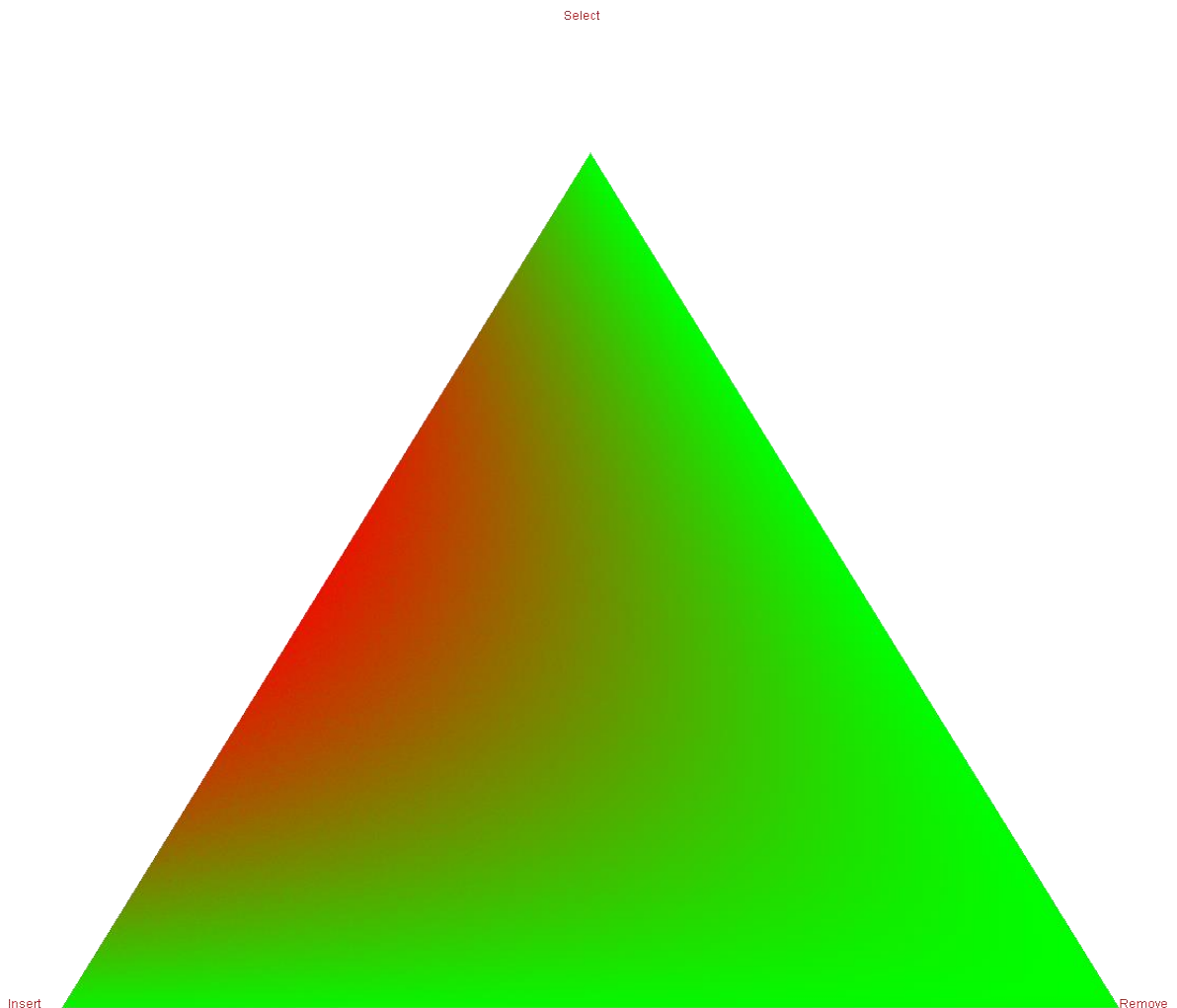


Рис. 7.3. Результат исследования В-дерева

Эксперимент 4. Простой и сортированный список.

Описание: сравнение суммарного количества операций сравнения и присваивания для простого и сортированного списков в зависимости соотношения количества операций вставки, поиска и удаления.

Суммарное число операций вставки, поиска и удаления: 1000.

Классы исследуемых контейнеров: `SortedList`, `SimpleList`.

Результат: результат работы программы представлен на рис. 7.4. На изображении видно, что сортированный список лучше простого только для очень большого количества удалений и поисков при очень малом количестве вставок. Также на рисунке можно увидеть разделяющую прямую, слева от которой при адаптивном выборе оптимального контейнера будет использоваться простой список, а справа сортированный.

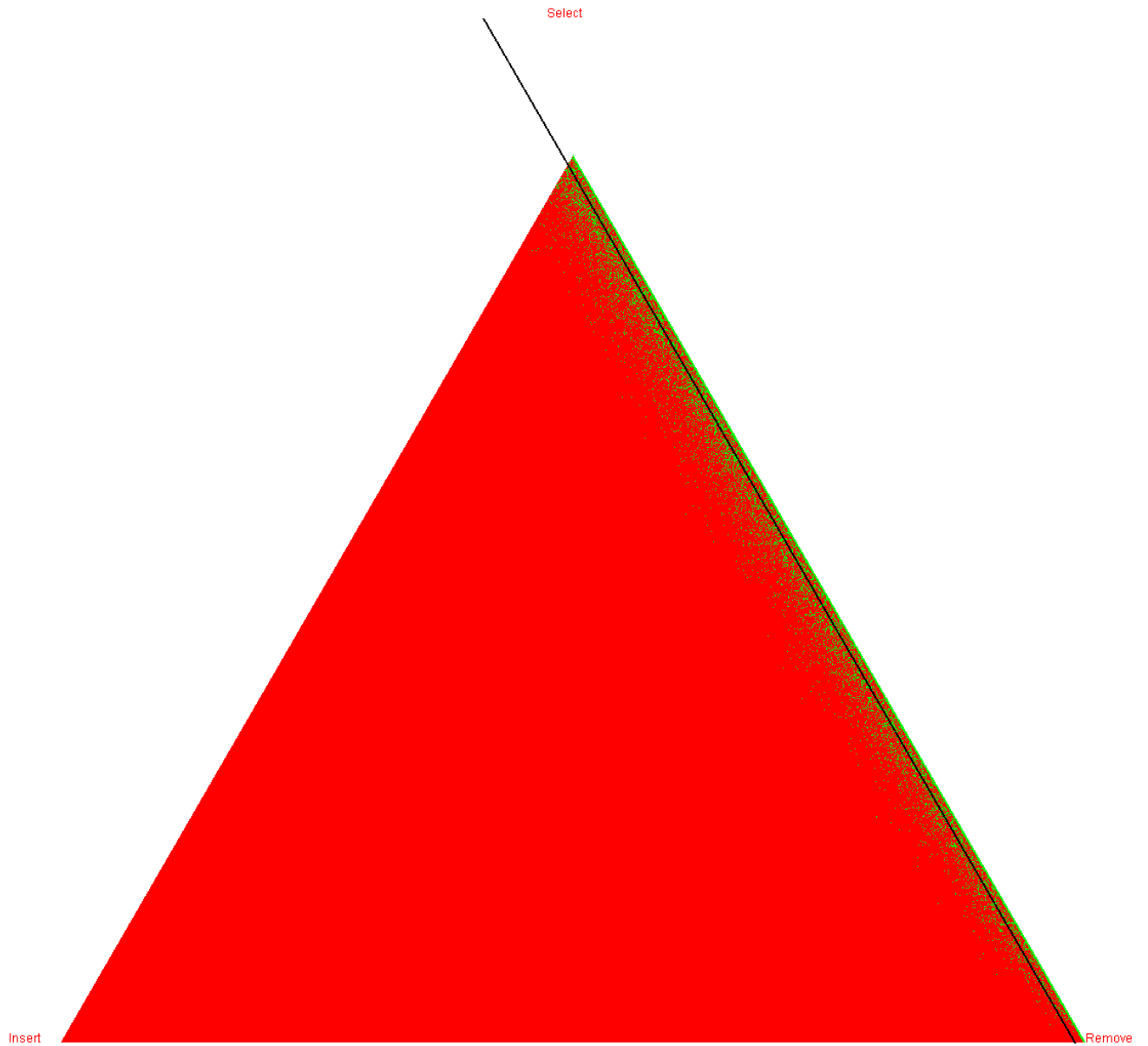


Рис. 7.4. Результат сравнения простого и сортированного списков

Эксперимент 5. Простой массив и В-дерева.

Описание: сравнение суммарного количества операций сравнения и присваивания для простого массива и В-дерева в зависимости соотношения количества операций вставки, поиска и удаления.

Суммарное число операций вставки, поиска и удаления: 1000.

Классы исследуемых контейнеров: `SimpleArray`, `BTree`.

Результат: результат работы программы представлен на рис. 7.5. На изображении видно, что простой хорошо использовать для тех случаев, когда количество операций поиска больше количества операций вставки и удаления. Также на рисунке можно увидеть разделяющую прямую, слева от которой при адаптивном выборе оптимального контейнера будет использоваться простой массив, а справа В-дерево.

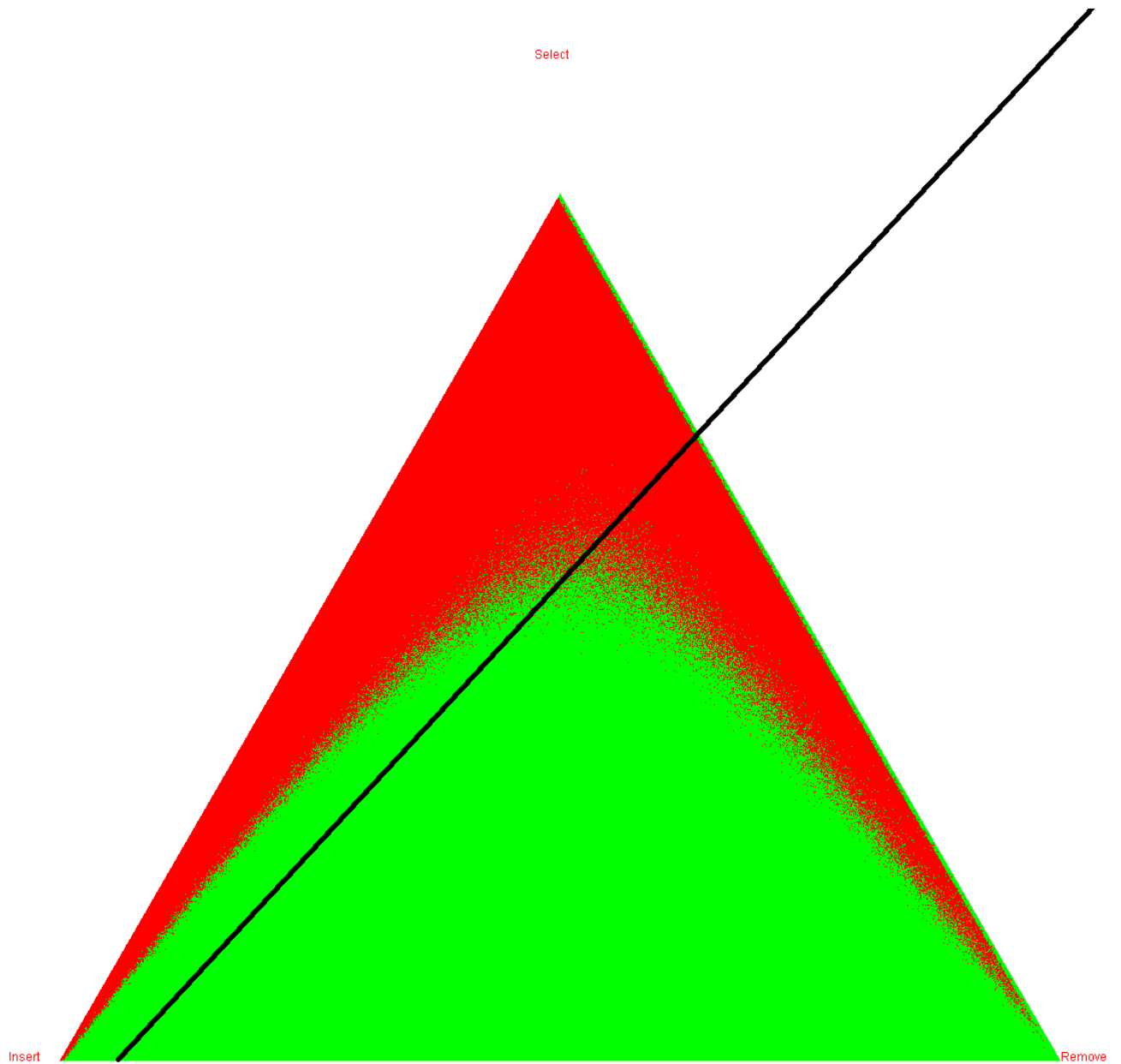


Рис. 7.4. Результат сравнения простого массива и В-дерева

Заключение

В ходе исследования был разработан специальный способ визуализации нагрузки на контейнеры данных. Такой способ помогает рассмотреть нагрузку практически любой структуры. Помимо этого, разработан способ получения коэффициентов плоскости, разделяющей пространство InsertRemoveSelect на две части, для одной из которых хорошо подходит один контейнер, а для другой – второй.

Основным результатом работы является программное обеспечение, в котором реализован следующий функционал:

1. Алгоритм визуализации нагрузки на контейнер хранения данных по параметру – количество операций.
2. Визуализация нагрузки на одном контейнере хранения данных.
3. Визуализация сравнения нескольких контейнеров хранения данных.
4. Алгоритм получения линейного классификатора для адаптивного выбора оптимального способа хранения данных в зависимости от нагрузки на основе оценки количества простейших операций с контейнером.

Результаты данного исследования могут быть применены при построении и/или перестроении индексов внутри баз данных, для любых структурированных и неструктурированных данных большого объема (Big Data) и других данных, требующих индексации, с целью добавить возможность адаптивного выбора оптимальной структуры таких индексов в зависимости от различных факторов.

Список литературы

1. Алгоритмы: построение и анализ / Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К; под ред. И. В. Красикова – 2-е изд. – Москва : Вильямс, 2005. – 1296 с.
2. Блох Д. Java. Эффективное программирование / Джошуа Блох – Москва : «Лори», 2002. – 220 с.
3. Вирт Н. Алгоритмы и структуры данных. / Н. Вирт – Москва : ДМК Пресс, 2010. – 272 с.
4. Гарсиа-Молина Г. Системы баз данных. Полный курс. / Гектор Гарсиа-Молина, Джеффри Д. Ульман, Дженнифер Уидом – Москва: Вильямс, 2004. – 1088 с.
5. Дейт Крис Дж. Введение в системы баз данных / Крис Дж. Дейт; пер. с англ. К. Птицин – Москва: Вильямс, 2006. – 1328 с.
6. Зобов В.В. Инструмент для моделирования нагрузки на контейнеры данных/ В.В. Зобов, К.Е. Селезнев // Материалы четырнадцатой научн.-метод. Конференции «Информатика: проблемы, методология, технологии», 10-11 февраля 2011 г. – Воронеж, 2014. – Т. 3. – с. 154-161.
7. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. / Д. Кнут – 3-е изд. – Москва : Вильямс, 2006. – 720 с.
8. Мартин Дж. Организация баз данных в вычислительных системах / Дж. Мартин – Москва : Мир, 1980. – 664 с.
9. Смит Дж. Мак-Колм Элементарные шаблоны проектирования / Джеймс Мак-Колм Смит – Москва : Вильямс, 2012. – 304 с.
10. Шилдт Г. Java. Полное руководство / Герберт Шилдт – 8-е изд. – Москва : Вильямс, 2012. – 1102 с.
11. Эккель Б. Философия Java / Брюс Эккель; пер. с англ. Е. Матвеев – Санкт-Петербург : Питер, 2009. – 640с.
12. Eclipse tutorial. – URL: <http://wiki.eclipse.org> (дата обращения: 01.06.2015).
13. Holzner S. Eclipse / S.Holzner. – O'Reilly Media, 2004. – 336p.

- 14.Libsvm. – URL: <http://www.csie.ntu.edu.tw/~cjlin/libsvm> (дата обращения: 01.06.2015).
- 15.SVM description. – URL: <http://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-and-machine-learning-on-documents-1.html> (дата обращения: 01.06.2015).

Приложения

Приложение 1. Модуль визуализации и основной класс для запуска программы

```

/**
 * Класс для запуска приложения визуализации
 *
 * @author Potapov Danila
 */
public class Main {
    /**
     * Логгер
     */
    static Logger log = Logger.getLogger(Main.class.getName());

    /**
     * Основная функция программы
     *
     * @param args
     *         аргументы запуска
     */
    public static void main(String[] args) {
        try {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            System.out
                .println("***Программа для визуализации нагрузки
на хранилища***");
            Integer size = getSize(br);
            String imageName = getImageName(br);
            List<IDataStorage> storages = getStorages(br);

            Vizualizator.Draw(size, imageName, storages);
        } catch (IOException e) {
            log.error("Во время выполнения программы произошла ошибка:",
e);
        }
    }

    /**
     * Функция получения хранилищ для тестирования
     *
     * @param br
     *         поток с консоли
     * @return список сформированных хранилищ
     * @throws IOException
     */
    private static List<IDataStorage> getStorages(BufferedReader br)
        throws IOException {
        String command = "y";
        String storageClass;
        String storageParamName;
        String storageParamValue;
        Integer storageParamIntValue = null;
        Map<String, Integer> params = null;
        List<IDataStorage> storages = new ArrayList<IDataStorage>();
        while ("y".equals(command.toLowerCase())) {

```

```

// Считываем название класса хранилища
do {
    System.out.print("Класс хранилища:");
    storageClass = br.readLine();
    if (storageClass == null || storageClass.equals("")) {
        System.out.println("Класс хранилища не должен быть
пуст!");
    }
} while (storageClass == null || storageClass.equals(""));

// Спрашиваем нужны ли параметры для хранилища
System.out.print("Добавить к хранилищу параметры? (y/n)");
command = br.readLine();
while ("y".equals(command)) {
    params = new HashMap<String, Integer>();
    // Считываем название параметра
    do {
        System.out.print("Название параметра хранилища:");
        storageParamName = br.readLine();
        if (storageParamName == null ||
storageParamName.equals("")) {
            System.out
                .println("Название параметра
хранилища не должно быть пусто!");
        }
    } while (storageParamName == null
        || storageParamName.equals(""));

    // Считываем значение параметра
    do {
        System.out
            .print("Значение параметра
хранилища (целочисленное):");
        storageParamValue = br.readLine();
        if (storageParamValue == null
            || storageParamValue.equals("")) {
            System.out
                .println("Значение параметра
хранилища не должно быть пусто!");
        }
        try {
            storageParamIntValue = Integer
                .parseInt(storageParamValue);
        } catch (NumberFormatException ex) {
        }
    } while (storageParamValue == null
        || storageParamValue.equals(""));
    if (storageParamIntValue != null)
        params.put(storageParamName,
storageParamIntValue);
    System.out
        .print("Добавить к хранилищу еще один
параметр? (y/n)");
    command = br.readLine();
}
IDataStorage storage = StorageGenerator.getDataStorage(
    storageClass, null, params);
if (storage != null) {
    storage.setCounterSet(new SimpleCounterSet());
    storages.add(storage);
} else
    System.out

```

```

        .println("Хранилище с таким классом не было
найдено. Оно не будет протестировано.");
        System.out.print("Добавить еще одно хранилище? (y/n)");
        command = br.readLine();
    }
    return storages;
}

/**
 * Функция получения с консоли размера выходного файла изображения в
 * пикселях
 *
 * @param br
 *      поток с консоли
 * @return размер выходного файла изображения в пикселях
 * @throws IOException
 */
private static Integer getSize(BufferedReader br) throws IOException {
    Integer size = null;
    do {
        System.out
            .print("Размер выходного файла изображения
в пикселях(от 10 до 2000):");
        String sizeStr = br.readLine();
        if (sizeStr == null || sizeStr.equals("")) {
            System.out
                .println("Размер выходного файла изображения
не должен быть пуст!");
        }
        try {
            size = Integer.parseInt(sizeStr);
            if (size < 10 || size > 2000) {
                System.out
                    .println("Размер выходного файла
изображения должен быть в диапазоне от 10 до 2000!");
            } else
                return size;
        } catch (NumberFormatException ex) {
        }
    } while (size == null || size < 10 || size > 2000);
    return null;
}

/**
 * Функция получения с консоли имени выходного файла изображения
 *
 * @param br
 *      поток с консоли
 * @return Имя выходного файла изображения
 * @throws IOException
 */
private static String getImageName(BufferedReader br) throws IOException {
    String imageName = null;
    do {
        System.out.print("Название выходного файла изображения:");
        imageName = br.readLine();

        if (imageName == null || imageName.equals("")) {
            System.out
                .println("Название выходного файла
изображения не может быть пусто!");
        }
    }

```

```

        return imageName;
    } while (imageName == null || imageName.equals(""));
}

/**
 * Класс для создания изображений, отображающих результаты сравнения хранилищ
 *
 * @author Potapov Danila
 */
public class Vizualizator {

    /**
     * Логгер
     */
    static Logger log = Logger.getLogger(Vizualizator.class.getName());

    /**
     * Переменная показывающая текущее количество хранилищ
     */
    private static int MODE;

    /**
     * Основной метод для создания изображения
     *
     * @param size
     *         размер выходного изображения
     * @param filename
     *         название выходного файла
     * @param storages
     *         список хранилищ для тестирования
     */
    public static void Draw(int size, String filename,
        List<IDataStorage> storages) {
        if (size < 0) {
            log.fatal("Был выбран некорректный размер файла.");
            return;
        }
        if (filename == null || filename.isEmpty()) {
            log.fatal("Было выбрано некорректное имя файла.");
            return;
        }
        if (storages == null || storages.isEmpty()) {
            log.fatal("Не были выбраны хранилища для тестирования");
            return;
        }
        switch (storages.size()) {
            case 1:
                MODE = 1;
                break;
            case 2:
                MODE = 2;
                break;
            default:
                MODE = 3;
                break;
        }
        log.info("Создаем файл с именем " + filename + " и размером " + size
            + " * " + size + " пикселей.");
        // Создаем файл изображения
        int type = BufferedImage.TYPE_INT_ARGB;
    }
}

```

```

        BufferedImage image = new BufferedImage(
            size + DrawConstants.OFFSET * 2, size +
DrawConstants.OFFSET
            * 2, type);
        File outputfile = new File(filename + "."
            + DrawConstants.IMAGE_EXTENSION);
        // рисуем на изображении
        drawImage(image, size, storages);
        // Сохраняем изображение
        try {
            ImageIO.write(image, DrawConstants.IMAGE_EXTENSION,
outputfile);
            log.info("Файл с именем " + filename + " и размером " + size +
""
                + size + " пикселей успешно создан.");
        } catch (IOException e) {
            log.fatal("Невозможно сохранить файл с именем " + filename
                + " и размером " + size + "" + size + "
пикселей.");
        }
    }

/**
 * Метод который наносит рисунок на изображение
 *
 * @param image
 *         изображение на котором рисуем
 * @param size
 *         размер изображения
 * @param storages
 *         список тестируемых хранилищ
 */
private static void drawImage(BufferedImage image, int size,
    List<IDataStorage> storages) {
    // Получаем список точек и их значений
    ImageData data = DataGenerator.getCoeffs(size, storages);
    Map<Point2D, List<Integer>> coeffs = data.getData();
    // Определяем тип изображения
    Color curColor = null;
    double curRange = 0;
    // Определяем диапазон значений(если 1 хранилище)
    if (MODE == 1) {
        curRange = data.getMax() - data.getMin();
    }
    // Список из точек для классификации
    List<Point> classificationPoints = new ArrayList<>();
    // Проходим по всем точкам, находим их цвета и отображаем
на изображении
    for (Entry<Point2D, List<Integer>> entry : coeffs.entrySet()) {
        Point2D point = entry.getKey();
        List<Integer> vals = entry.getValue();
        // считаем значение в точке
        Point p = new Point(point, vals);
        // определяем цвет
        curColor = getPointColor(p, data.getMin(), curRange);
        if (curColor != null)
            image.setRGB((int) point.getX() + DrawConstants.OFFSET,
size
                - (int) point.getY() + DrawConstants.OFFSET,
                curColor.getRGB());
        // Если 2 хранилища пытаемся добавить точку
        if (MODE == 2)

```

```

        addPointForClassification(classificationPoints, p,
size);

    }
    // Если 2 хранилища рисуем линию классификатора
    if (MODE == 2) {
        drawClassifierLine(classificationPoints, image, size);
    }
    // рисуем оси
    drawAxis(image, size);
}

/**
 * Метод для отрисовки осей координат
 *
 * @param image
 *         изображение на котором рисуем
 * @param size
 *         размер изображения
 */
private static void drawAxis(BufferedImage image, int size) {
    Graphics gr = image.getGraphics();
    gr.setColor(Color.RED);
    gr.drawString("Select", (size + DrawConstants.OFFSET) / 2,
        DrawConstants.OFFSET);
    gr.drawString("Insert", 0, size + DrawConstants.OFFSET);
    gr.drawString("Remove", size + DrawConstants.OFFSET, size
        + DrawConstants.OFFSET);
}

/**
 * Метод получения цвета для точки
 *
 * @param isSingle
 *         параметр для определения алгоритма определения
 *         цвета (количество сравниваемых хранилищ)
 * @param point
 *         точка для которой ищем цвет
 * @param vals
 *         значения этой точки
 * @param mi
 *         минимальное значение среди всех точек (Null если кол-во
 *         хранилищ>1)
 * @param curRange
 *         диапазон значений среди всех точек (0 если кол-во хранилищ>1)
 * @return цвет для точки
 */
private static Color getPointColor(Point point, Integer mi, double
curRange) {
    Color curColor = null;
    // Если одно хранилище, получаем коэффициент точки из диапазона [0,1]
и
    // цвет с использованием линейного градиента
    if (MODE == 1) {
        double ratio = (point.getValue() - mi) / curRange;
        curColor = DrawUtils.getColorByLinearGradient(ratio,
            DrawConstants.COLORS[0], DrawConstants.COLORS[1]);
        // Иначе ищем минимальное значение среди всех хранилищ
и выбираем
        // соответствующий данному хранилищу цвет
    } else {
        try {
            curColor = DrawConstants.COLORS[point.getValue()];

```



```

        } catch (ArrayIndexOutOfBoundsException ex) {
            log.error("Для хранилища №" + point.getValue()
                + "не задан цвет в точке(" + point.getX() +
", "
                + point.getY() + ").");
        }
    }
    return curColor;
}

/**
 * Функция которая добавляет точки по которым будет производиться
 * классификация
 *
 * @param classificationPoints
 *         точки по которым производится классификация
 * @param point
 *         точка для которой определяется будет ли она добавлена
к списку
 * @param size
 *         размер изображения
 */
private static void addPointForClassification(
    List<Point> classificationPoints, Point point, Integer size) {
    double x = point.getX();
    double y = point.getY();
    // Строим сетку из точек и берем точки лежащие на сторонах
треугольника
    if (((x % (size / 40) == 0 && y % (size / 40) == 0) || DrawUtils
        .pointInTriangleSide(0, 0, size / 2,
            (float) (size * Math.sqrt(3.0) / 2.0f),
size, 0,
            (float) x, (float) y))) {
        classificationPoints.add(point);
    }
}

/**
 * Функция которая производит классификацию и рисует разделяющую линию
 *
 * @param classificationPoints
 *         точки по которым производится классификация
 * @param image
 *         изображение на котором надо нарисовать линию
 * @param size
 *         размер изображения
 */
private static void drawClassifierLine(List<Point>
classificationPoints,
    BufferedImage image, Integer size) {
    log.info("Началась классификация.");
    double[] w = Classifier.classification(classificationPoints);
    log.info("Классификация выполнена успешно.");
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            if (Math.abs(DrawUtils.evalLineFunction(w, x, y)) <
0.001) {
                image.setRGB((int) x + DrawConstants.OFFSET, size
- (int) y
                    + DrawConstants.OFFSET,
Color.BLACK.getRGB());
            }
        }
    }
}

```

```

    }
    Point2D p1 = new Point2D.Double(0,
DrawUtils.evalLineFunctionYValue(w,
    0));
    Point2D p2 = new Point2D.Double(size,
DrawUtils.evalLineFunctionYValue(
    w, size));
    // Переводим координаты из 2х мерных в 3х мерные
    CoordinanateTranslator transl = new CoordinanateTranslator(size);
    Point3DInIRSCoords r1 = transl.translate(p1);
    Point3DInIRSCoords r2 = transl.translate(p2);
    Point3DInIRSCoords r3 = new Point3DInIRSCoords(0, 0, 0);
    double[] coeffs = DrawUtils.getPlaneCoeffsByThreePoints(r1, r2, r3);
    System.out.println("Уравнение разделяющей плоскости: " + coeffs[0]
        + "*x+" + coeffs[1] + "*y+" + coeffs[2] + "*z+" +
coeffs[3]
        + "=0");
    }
}

```

Приложение 2. Модуль генерации данных

```

/**
 * Класс для получения количества операций
 *
 * @author Potapov Danila
 *
 */
public class DataGenerator {

    /**
     * Логгер
     */
    static Logger log = Logger.getLogger(Vizualizator.class.getName());

    /**
     * Метод для получения количества операций
     *
     * @param storages
     *           хранилища, на которых тестируются нагрузки
     * @param insertCount
     *           количество insert
     * @param selectCount
     *           количество select
     * @param removeCount
     *           количество remove
     * @return список, который содержит сумму операций присваивания
и сравнения
     *           для каждого хранилища
     */
    public static List<Integer> getConterersForStorages(
        List<IDataStorage> storages, Integer insertCount,
        Integer selectCount, Integer removeCount) {
        // Создаем плеер для тестирования нагрузки
        DataSetPlayer dsp = new DataSetPlayer(storages, null);
        // Создаем нагрузку и тестируем ее
        dsp.play(SourceGenerator.createInsertSelectRemoveSource(insertCount,
            selectCount, removeCount));
        // Получаем список, который содержит сумму операций присваивания и
        // сравнения для каждого хранилища
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < storages.size(); i++) {
            ICounterSet set = storages.get(i).getCounterSet();
            Integer sumOfOperations = set.get(OperationType.ASSIGN)
                + set.get(OperationType.COMPARE);
            result.add(sumOfOperations);
        }
        return result;
    }

    /**
     * Метод который для каждой точки изображения получает количество операций
     *
     * @param size
     *           размер изображения
     * @param storages
     *           тестируемые хранилища
     * @return карту из точек и их значений, а также минимум и максимум (если
     *           одно хранилище)
     */
    public static ImageData getCoeffs(int size, List<IDataStorage> storages) {

```

```

        // Получаем количество доступных процессоров
        int PROCESSORS_COUNT = Runtime.getRuntime().availableProcessors();
        ExecutorService executor = Executors
            .newFixedThreadPool(PROCESSORS_COUNT + 1);
        Map<Point2D, List<Integer>> coeffs = new HashMap<Point2D,
List<Integer>>>();
        Integer min = null;
        Integer max = null;
        // Генерируем задания по обработке точек
        List<Future<ThreadCounterResult>> list = generateTasks(size,
storages,
            executor, PROCESSORS_COUNT);
        // Вызываем задания
        for (Future<ThreadCounterResult> fut : list) {
            ThreadCounterResult r = null;
            try {
                r = fut.get();
                // Если одно хранилище, то ищем минимум и максимум
                по всем
                // заданиям
                if (storages.size() == 1) {
                    if (min == null || r.getCurMin() < min)
                        min = r.getCurMin();
                    if (max == null || r.getCurMax() > max)
                        max = r.getCurMax();
                }
                // Добавляем результаты в одну карту точек - значений
                coeffs.putAll(r.getResult());
            } catch (InterruptedException | ExecutionException e) {
                ошибка. Результат может быть некорректным.");
            }
        }
        executor.shutdown();
        if (storages.size() == 1) {
            log.debug("Min=" + min + " ;max= " + max);
        }
        ImageData data = new ImageData(min, max, coeffs);
        return data;
    }

    /**
     * Метод который формирует задания по обработке точек изображения и
     * отправляет их на выполнение
     *
     * @param size
     *         размер изображения
     * @param storages
     *         хранилища для тестирования
     * @param executor
     *         класс для выполнения заданий
     * @param PROCESSORS_COUNT
     *         количество заданий
     * @return
     */
    private static List<Future<ThreadCounterResult>> generateTasks(int size,
        List<IDataStorage> storages, ExecutorService executor,
        int PROCESSORS_COUNT) {
        // Количество точек на поток
        Integer countInTask = (size * size / 2) / PROCESSORS_COUNT;
        // номер текущей точки
        Integer curPointNumber = 0;
        // Номер текущего задания

```

```

Integer curTaskNumber = 1;
List<Point2D> curPoints = new ArrayList<Point2D>();
List<Future<ThreadCounterResult>> list = new
ArrayList<Future<ThreadCounterResult>>();
// Создаем переводчик координат
CoordinanateTranslator transl = new CoordinanateTranslator(size);
// Проходим по всем точкам изображения
for (int x = 0; x < size; x++) {
    for (int y = 0; y < size; y++) {
        // Определяем лежит ли точка в треугольнике Insert(0,0)
        // Select(size/2,size*sqrt (3)/2) Remove(size,0)
        if (DrawUtils.pointInTriangle(0, 0, size / 2, (float)
(size
                                * Math.sqrt(3.0) / 2.0f), size, 0, x, size -
y)) {
            Point2D p = new Point2D.Double(x, size - y);
            curPoints.add(p);
            // Если набралось необходимое количество точек для
создания
            // задания
            if (curPointNumber > curTaskNumber * countInTask)
{
                // Создаем копии хранилищ
                List<IDataStorage> storagesCopy = new
ArrayList<IDataStorage> (
                                storages.size());
                for (int k = 0; k < storages.size(); k++) {
                    storagesCopy.add(storages.get(k).cloneDefault());
                }
                // Создаем задание и отправляем его
на выполнение
                Callable<ThreadCounterResult> callable = new
ThreadCounter (
                                curPoints, transl, storagesCopy);
                Future<ThreadCounterResult> future =
executor
                                .submit(callable);
                // add Future to the list, we can get return
value using
                // Future
                list.add(future);
                curTaskNumber++;
                curPoints = new ArrayList<Point2D>();
            }
            curPointNumber++;
        }
    }
}
// Если остались точки, то создаем еще одно задание
if (curPoints != null) {
    List<IDataStorage> storagesCopy = new ArrayList<IDataStorage> (
        storages.size());
    for (int k = 0; k < storages.size(); k++) {
        storagesCopy.add(storages.get(k).cloneDefault());
    }
    Callable<ThreadCounterResult> callable = new ThreadCounter (
        curPoints, transl, storagesCopy);
    Future<ThreadCounterResult> future =
executor.submit(callable);
    // add Future to the list, we can get return value using
Future
    list.add(future);
}

```

```

        }
        return list;
    }
}

/**
 * Класс реализующий вызов в отдельном потоке
 *
 * @author Potapov Danila
 *
 */
public class ThreadCounter implements Callable<ThreadCounterResult> {
    /**
     * Точки, которые необходимо обработать данному потоку
     */
    List<Point2D> points;
    /**
     * Ссылка на транслятор координат
     */
    CoordinanateTranslator transl;
    /**
     * Логгер
     */
    static Logger log = Logger.getLogger(Vizualizator.class.getName());
    /**
     * Список тестируемых хранилищ
     */
    List<IDataStorage> storages;

    /**
     * Конструктор
     *
     * @param points
     *         список точек для тестирования
     * @param transl
     *         ссылка на транслятор координат
     * @param storages
     *         список хранилищ для тестирования
     */
    public ThreadCounter(List<Point2D> points, CoordinanateTranslator transl,
        List<IDataStorage> storages) {
        this.points = points;
        this.transl = transl;
        this.storages = storages;
    }

    /**
     * Метод вызываемый в отдельном потоке для обработки
     */
    @Override
    public ThreadCounterResult call() throws Exception {
        Integer min = null;
        Integer max = null;
        Map<Point2D, List<Integer>> coeffs = new HashMap<Point2D,
List<Integer>>();
        // Проходим по всем точкам потока
        for (int i = 0; i < points.size(); i++) {
            Point2D p = points.get(i);
            // Переводим в координаты IRS
            Point3DInIRSCoords transfRes = transl.translate(p);
            // Получаем значения количества операций для точки IRS
            List<Integer> result = DataGenerator.getConetersForStorages(
                storages, transfRes.getInsertCoord(),

```

```

        transfRes.getSelectCoord(),
transfRes.getRemoveCoord());
        log.debug("Была обработана точка ("
            + p.getX()
            + ","
            + p.getY()
            + "). Новые координаты ISR ("
            + transfRes.getInsertCoord()
            + ","
            + transfRes.getSelectCoord()
            + ","
            + transfRes.getRemoveCoord()
            + "). Количество операций: "
            + result.stream().map(Object::toString)
                           .collect(Collectors.joining(", ")));
        // Если одно хранилище, то ищем минимум и максимум
по значениям
        // точек
        if (storages.size() == 1) {
            Integer cur = result.get(0);
            if (min == null || cur < min)
                min = cur;
            if (max == null || cur > max)
                max = cur;
        }
        // Очищаем хранилища
        for (int j = 0; j < storages.size(); j++) {
            storages.get(j).clear();
            storages.get(j).setCounterSet(new SimpleCounterSet());
        }
        coeffs.put(p, result);
    }

    ThreadCounterResult result = new ThreadCounterResult(coeffs, min,
max);
    return result;
}
}

```

Приложение 3. Модуль перевода координат

```

/**
 * Класс для перевода координат из 2х мерных координат изображения (x, y) в
 * целочисленные координаты 3х мерные (insert, select, remove)
 *
 * @author Potapov Danila
 *
 */
public class CoordinanateTranslator {
    /**
     * Угол поворота по оси Z
     */
    private static final double ANGLE_ROTATE_BY_Z = -45;
    /**
     * Угол поворота по оси Y
     */
    private static final double ANGLE_ROTATE_BY_Y = 35.26438968275464;

    private static final double COEFFICIENT_FOR_MOVE = Math.sqrt(2.0) / 2;
    /**
     * Матрица перевода координат
     */
    double[][] tranlationMatrix;

    /**
     * Конструктор для создания транслятора
     *
     * @param size
     *
     */
    public CoordinanateTranslator(double size) {
        // создаем матрицу поворота по оси Y
        double[][] rotateOnY =
MatrixUtils.rotateOnAngleByY(ANGLE_ROTATE_BY_Y);
        // создаем матрицу поворота по оси Z
        double[][] rotateOnZ =
MatrixUtils.rotateOnAngleByZ(ANGLE_ROTATE_BY_Z);
        // создаем матрицу сдвига по оси X (умножаем размер на синус 45 для
        // получения длины отрезка (0, maxInsertCoordinat))
        double[][] move = MatrixUtils.moveOnXYZ(size * COEFFICIENT_FOR_MOVE,
0,
0);
        double[][] tmp = MatrixUtils.multiplyByMatrix(rotateOnY, rotateOnZ);
        // Получаем матрицу перевода путем умножения
        tranlationMatrix = MatrixUtils.multiplyByMatrix(tmp, move);
    }

    /**
     * Функция перевода координат из 2х мерных координат изображения (x, y) в
     * целочисленные координаты 3х мерные (insert, select, remove)
     *
     * @param point
     *
     * точка (x, y)
     *
     * @return точку в 3х мерных координатах
     */
    public Point3DInIRSCoords translate(Point2D point) {
        // Создаем вектор в старых координатах
        double[][] param = { { 0, point.getX(), point.getY(), 1 } };
        // Получаем координаты в 3х мерном пространстве
        double[][] result = MatrixUtils.multiplyByMatrix(param,
        tranlationMatrix);
    }
}

```



```
int insertCount = (int) Math.round(result[0][0]);
int selectCount = (int) Math.round(result[0][1]);
int removeCount = (int) Math.round(result[0][2]);
Point3DInIRSCoords res = new Point3DInIRSCoords(insertCount,
    selectCount, removeCount);
return res;
    }
}
```

Приложение 4. Модуль классификации

```

/**
 * Класс для бинарной классификации методом SVM
 *
 * @author Potapov Danila
 *
 */
public class Classifier {

    /**
     * Метод линейной классификации
     *
     * @param classificationPoints
     *        точки по которым идет классификация
     * @return коэффициенты разделяющей линии  $ax+by+c=0$ 
     */
    public static double[] classification(List<Point> classificationPoints) {
        // Создаем параметры
        svm_parameter param = new svm_parameter();
        svm.svm_set_print_string_function(new libsvm.svm_print_interface() {
            @Override
            public void print(String s) {
                // Disables svm output
            }
        });
        // default values
        param.svm_type = svm_parameter.C_SVC;
        param.kernel_type = svm_parameter.LINEAR;
        param.degree = 3;
        param.gamma = 0;
        param.coef0 = 0;
        param.nu = 0.5;
        param.cache_size = 0;
        param.C = 5;
        param.eps = 1e-2;
        param.p = 0.1;
        param.shrinking = 0;
        param.probability = 0;
        param.nr_weight = 0;
        param.weight_label = new int[0];
        param.weight = new double[0];

        // Создаем проблему
        svm_problem problem = new svm_problem();

        problem.l = classificationPoints.size(); // number of training
examples
        problem.y = new double[problem.l];
        problem.x = new svm_node[problem.l][2];
        int i = 0;
        // Добавляем к проблеме точки (param.x) и значения их классов (0 и 1)
        List<Integer> vals = new ArrayList<>();
        for (Point point : classificationPoints) {
            problem.x[i][0] = new svm_node();
            problem.x[i][0].index = 1;
            problem.x[i][0].value = point.getX();
            problem.x[i][1] = new svm_node();
            problem.x[i][1].index = 2;
            problem.x[i][1].value = point.getY();
            problem.y[i] = point.value;
            vals.add(point.value);
            i++;
        }
    }
}

```

```

    }

    // создаем модель, содержащую решение(на основе настроек и проблемы)
    svm_model model = svm.svm_train(problem, param);
    double[] w = new double[3];
    for (i = 0; i < model.SV[0].length; i++) {
        for (int j = 0; j < model.SV.length; j++) {
            w[i] += model.SV[j][i].value * model.sv_coef[0][j];
        }
    }
    w[2] = -model.rho[0];
    // нормализуем решение
    double norma = Math.sqrt(w[0] * w[0] + w[1] * w[1] + w[2] * w[2]);
    if (model.label[1] == 0) {
        w[0] = -w[0];
        w[1] = -w[1];
        w[2] = -w[2];
    }
    w[0] /= norma;
    w[1] /= norma;
    w[2] /= norma;
    // do_cross_validation(problem,param);
    return w;
}

/**
 * Метод для проверки и поиска точности
 *
 * @param prob
 * @param param
 */
@SuppressWarnings("unused")
private static void do_cross_validation(svm_problem prob,
    svm_parameter param) {
    int i;
    int total_correct = 0;
    double total_error = 0;
    double sumv = 0, sumy = 0, sumvv = 0, sumyy = 0, sumvy = 0;
    double[] target = new double[prob.l];

    svm.svm_cross_validation(prob, param, 2, target);
    if (param.svm_type == svm_parameter.EPSILON_SVR
        || param.svm_type == svm_parameter.NU_SVR) {
        for (i = 0; i < prob.l; i++) {
            double y = prob.y[i];
            double v = target[i];
            total_error += (v - y) * (v - y);
            sumv += v;
            sumy += y;
            sumvv += v * v;
            sumyy += y * y;
            sumvy += v * y;
        }
        System.out.print("Cross Validation Mean squared error = "
            + total_error / prob.l + "\n");
        System.out
            .print("Cross Validation Squared correlation
coefficient = "
                                + ((prob.l * sumvy - sumv * sumy) *
(prob.l * sumvy - sumv
                                * sumy))

```

```

                                                                    / ((prob.l * sumvv - sumv * sumv) *
(prob.l * sumyy - sumy                                                                    * sumy)) + "\n");
    } else
        for (i = 0; i < prob.l; i++)
            if (target[i] == prob.y[i])
                ++total_correct;
    System.out.print("Cross Validation Accuracy = " + 100.0 *
total_correct
                    / prob.l + "%\n");

```

Приложение 5. Листинг. Вспомогательные классы

```

/**
 * Класс для хранения координат и значения в точке
 *
 * @author Potapov Danila
 *
 */
public class Point {
    // координата по оси x
    Double x;
    // координата по оси y
    Double y;
    // значение в точке
    Integer value;

    public Double getX() {
        return x;
    }

    public void setX(Double x) {
        this.x = x;
    }

    public Double getY() {
        return y;
    }

    public void setY(Double y) {
        this.y = y;
    }

    public Integer getValue() {
        return value;
    }

    public void setValue(Integer value) {
        this.value = value;
    }

    public Point(Double x, Double y, Integer value) {
        super();
        this.x = x;
        this.y = y;
        this.value = value;
    }

    @Override
    public String toString() {
        return "Point [x=" + x + ", y=" + y + ", value=" + value + "];"
    }

    /**
     * Метод получения точки по координатам и набору возможных значений
     *
     * @param point
     *            координаты точки
     * @param vals
     *            возможные значения
     * @return
     */
    public Point(Point2D point, List<Integer> vals) {

```

```

        Integer value = vals.get(0);
        if (vals.size() != 1) {
            int minInd = 0;
            int min = vals.get(minInd);
            for (int i = 0; i < vals.size(); i++) {
                if (vals.get(i) < min) {
                    minInd = i;
                    min = vals.get(i);
                }
            }
            value = minInd;
        }
        this.x=point.getX();
        this.y=point.getY();
        this.value=value;
    }

}

/**
 * Класс, который содержит 3х мерные координаты в пространстве
 * (insert,select,remove)
 *
 * @author Potapov Danila
 *
 */
public class Point3DInIRSCoords {
    /**
     * Координата по оси insert
     */
    Integer insertCoord;
    /**
     * Координата по оси remove
     */
    Integer removeCoord;
    /**
     * Координата по оси select
     */
    Integer selectCoord;

    /**
     * Конструктор
     *
     * @param insertCoord
     *            Координата по оси insert
     * @param removeCoord
     *            Координата по оси remove
     * @param selectCoord
     *            Координата по оси select
     */
    public Point3DInIRSCoords(Integer insertCoord, Integer removeCoord,
                               Integer selectCoord) {
        super();
        this.insertCoord = insertCoord;
        this.removeCoord = removeCoord;
        this.selectCoord = selectCoord;
    }

    public Integer getInsertCoord() {
        return insertCoord;
    }

    public void setInsertCoord(Integer insertCoord) {

```

```

        this.insertCoord = insertCoord;
    }

    public Integer getRemoveCoord() {
        return removeCoord;
    }

    public void setRemoveCoord(Integer removeCoord) {
        this.removeCoord = removeCoord;
    }

    public Integer getSelectCoord() {
        return selectCoord;
    }

    public void setSelectCoord(Integer selectCoord) {
        this.selectCoord = selectCoord;
    }
}

/**
 * Вспомогательный класс, на основе которого строится изображение
 *
 * @author Potapov Danila
 */
public class ImageData {
    /**
     * Минимальное количество операций
     */
    Integer min;
    /**
     * Максимальное количество операций
     */
    Integer max;
    /**
     * Карта, которая содержит координаты точки для отрисовки
     и соответствующие
     * ей значения количества операций
     */
    Map<Point2D, List<Integer>> data;

    /**
     * Конструктор
     *
     * @param min
     *             Минимальное количество операций
     * @param max
     *             Максимальное количество операций
     * @param data
     *             Карта, которая содержит координаты точки для отрисовки и
     соответствующие ей значения количества операций
     */
    public ImageData(Integer min, Integer max, Map<Point2D, List<Integer>>
data) {
        super();
        this.min = min;
        this.max = max;
        this.data = data;
    }

    public Integer getMin() {

```

```

        return min;
    }

    public void setMin(Integer min) {
        this.min = min;
    }

    public Integer getMax() {
        return max;
    }

    public void setMax(Integer max) {
        this.max = max;
    }

    public Map<Point2D, List<Integer>> getData() {
        return data;
    }

    public void setData(Map<Point2D, List<Integer>> data) {
        this.data = data;
    }
}

/**
 * Результат работы потока по подсчету количества операций
 *
 * @author Potapov Danila
 */
public class ThreadCounterResult {
    /**
     * Карта из точек и их значений посчитанная текущим потоком
     */
    Map<Point2D, List<Integer>> result;
    /**
     * Минимум найденный текущим потоком
     */
    Integer curMin;
    /**
     * Максимум найденный текущим потоком
     */
    Integer curMax;

    /**
     * Конструктор
     *
     * @param result
     *             Карта из точек и их значений посчитанная текущим потоком
     * @param curMin
     *             Минимум найденный текущим потоком
     * @param curMax
     *             Максимум найденный текущим потоком
     */
    public ThreadCounterResult(Map<Point2D, List<Integer>> result,
                               Integer curMin, Integer curMax) {
        super();
        this.result = result;
        this.curMin = curMin;
        this.curMax = curMax;
    }
}

```



```

    public Map<Point2D, List<Integer>> getResult() {
        return result;
    }

    public void setResult(Map<Point2D, List<Integer>> result) {
        this.result = result;
    }

    public Integer getCurMin() {
        return curMin;
    }

    public void setCurMin(Integer curMin) {
        this.curMin = curMin;
    }

    public Integer getCurMax() {
        return curMax;
    }

    public void setCurMax(Integer curMax) {
        this.curMax = curMax;
    }
}

/**
 * Константы используемые для отрисовки изображения
 *
 * @author Potapov Danila
 */
public class DrawConstants {
    /**
     * Используемые цвета
     */
    public static final Color[] COLORS = { Color.GREEN, Color.RED, Color.BLUE,
        Color.YELLOW };
    /**
     * Смещение для отображения координатных осей
     */
    public static final int OFFSET = 50;
    /**
     * Расширение для изображения
     */
    public static final String IMAGE_EXTENSION = "png";
}

/**
 * Класс, который содержит вспомогательные функции для построения изображения
 *
 * @author Potapov Danila
 */
public class DrawUtils {
    /**
     * Функция, для определения по какую сторону от прямой лежит точка
     *
     * @param startX
     *         x координата начала прямой
     * @param startY

```

```

*          у координата начала прямой
* @param finishX
*          х координата конца прямой
* @param finishY
*          у координата конца прямой
* @param x
*          х координата тестируемой точки
* @param y
*          у координата тестируемой точки
* @return число, по которому можно определить с какой стороны от прямой
*         расположена точка
*/
public static float side(float startX, float startY, float finishX,
                        float finishY, float x, float y) {
    return (finishY - startY) * (x - startX) + (-finishX + startX)
           * (y - startY);
}

/**
 * Функция для проверки принадлежности точки треугольнику ABC
 *
 * @param xA
 *          х координата точки A
 * @param yA
 *          у координата точки A
 * @param xB
 *          х координата точки B
 * @param yB
 *          у координата точки B
 * @param xC
 *          х координата точки C
 * @param yC
 *          у координата точки C
 * @param x
 *          х координата тестируемой точки
 * @param y
 *          у координата тестируемой точки
 * @return true если точка лежит в треугольнике
 */
public static Boolean pointInTriangle(float xA, float yA, float xB,
                                     float yB, float xC, float yC, float x, float y) {
    // Точка должна быть по одну сторону от всех прямых
    Boolean checkSide1 = side(xA, yA, xB, yB, x, y) >= 0;
    Boolean checkSide2 = side(xB, yB, xC, yC, x, y) >= 0;
    Boolean checkSide3 = side(xC, yC, xA, yA, x, y) >= 0;
    return checkSide1 && checkSide2 && checkSide3;
}

/**
 * Функция для проверки принадлежности точки сторонам треугольника ABC
 *
 * @param xA
 *          х координата точки A
 * @param yA
 *          у координата точки A
 * @param xB
 *          х координата точки B
 * @param yB
 *          у координата точки B
 * @param xC
 *          х координата точки C
 * @param yC
 *          у координата точки C

```

```

* @param x
*           x координата тестируемой точки
* @param y
*           y координата тестируемой точки
* @return true если точка лежит на стороне треугольника
*/
public static Boolean pointInTriangleSide(float xA, float yA, float xB,
    float yB, float xC, float yC, float x, float y) {
    // Точка должна быть на стороне
    Boolean checkSide1 = side(xA, yA, xB, yB, x, y) <= 50
        && side(xA, yA, xB, yB, x, y) >= 0;
    Boolean checkSide2 = side(xB, yB, xC, yC, x, y) <= 50
        && side(xB, yB, xC, yC, x, y) >= 0;
    return checkSide1 || checkSide2;
}

/**
* Функция получения цвета с использованием линейного градиента
*
* @param ratio
*           коэффициент для которого получаем цвет(должен быть
из диапазона
*           [0,1]))
* @param colorMin
*           цвет от которого идет заливка
* @param colorMax
*           цвет к которому идет заливка
* @return новый цвет полученный с использованием линейного градиента
*/
public static Color getColorByLinearGradient(double ratio, Color colorMin,
    Color colorMax) {
    if (ratio > 1)
        ratio = 1;
    if (ratio < 0)
        ratio = 0;
    // Получаем каждую компоненту нового цвета по формуле
    int red = (int) (colorMax.getRed() * ratio + colorMin.getRed()
        * (1 - ratio));
    int green = (int) (colorMax.getGreen() * ratio + colorMin.getGreen()
        * (1 - ratio));
    int blue = (int) (colorMax.getBlue() * ratio + colorMin.getBlue()
        * (1 - ratio));
    Color c = new Color(red, green, blue);
    return c;
}

/**
* Метод вычисления значения функции  $f(x,y)=ax+by+c$ 
*
* @param w
*           коэффициенты a,b,c
* @param x
*           значение x
* @param y
*           значение y
* @return значение функции в точке (x,y)
*/
public static double evalLineFunction(double[] w, double x, double y) {
    double res = w[0] * (x) + w[1] * (y) + w[2];
    return res;
}

/**

```

```

* Метод вычисления значения функции  $f(x)=ax+b$ 
*
* @param w
*         коэффициенты a,b
* @param x
*         значение x
* @return значение функции в точке x
*/
public static double evalLineFunctionYValue(double[] w, double x) {
    double res = w[1] != 0 ? (w[0] / w[1] * (x) + w[2] / w[1]) :
Double.NaN;
    return res;
}

/**
* Метод вычисления коэффициентов плоскости  $Ax+By+Cz+D=0$  по трем точкам
*
* @param p1
*         первая точка
* @param p2
*         вторая точка
* @param p3
*         третья точка
* @param size
*         размер изображения
* @return массив коэффициентов
*/
public static double[] getPlaneCoeffsByThreePoints(Point3DInIRSCoords r1,
    Point3DInIRSCoords r2, Point3DInIRSCoords r3) {
    double[] result = new double[4];
    // коэффициент при Insert
    result[0] = r2.getRemoveCoord() * r3.getSelectCoord()
        + r1.getRemoveCoord() * r2.getSelectCoord()
        + r1.getSelectCoord() * r3.getRemoveCoord()
        - r1.getSelectCoord() * r2.getRemoveCoord()
        - r1.getRemoveCoord() * r3.getSelectCoord()
        - r2.getSelectCoord() * r3.getRemoveCoord();
    // коэффициент при Remove
    result[1] = r2.getSelectCoord() * r3.getInsertCoord()
        + r1.getInsertCoord() * r3.getSelectCoord()
        + r1.getSelectCoord() * r2.getInsertCoord()
        - r1.getSelectCoord() * r3.getInsertCoord()
        - r2.getInsertCoord() * r3.getSelectCoord()
        - r1.getInsertCoord() * r2.getSelectCoord();
    // коэффициент при Select
    result[2] = r1.getInsertCoord() * r2.getRemoveCoord()
        + r1.getRemoveCoord() * r3.getInsertCoord()
        + r2.getInsertCoord() * r3.getRemoveCoord()
        - r2.getRemoveCoord() * r3.getInsertCoord()
        - r1.getRemoveCoord() * r2.getInsertCoord()
        - r1.getInsertCoord() * r3.getRemoveCoord();
    // свободный коэффициент
    result[3] = -(r1.getInsertCoord() * r2.getRemoveCoord()
        * r3.getSelectCoord() + r1.getRemoveCoord()
        * r3.getInsertCoord() * r2.getSelectCoord()
        + r2.getInsertCoord() * r3.getRemoveCoord()
        * r1.getSelectCoord() - r2.getRemoveCoord()
        * r1.getSelectCoord() * r3.getInsertCoord()
        - r1.getRemoveCoord() * r2.getInsertCoord()
        * r3.getSelectCoord() - r1.getInsertCoord()
        * r3.getRemoveCoord() * r2.getSelectCoord());
    return result;
}

```

```

}

/**
 * Вспомогательный класс для работы с матрицами
 *
 * @author Potapov Danila
 *
 */
public class MatrixUtils {

    /**
     * Метод для перемножения матриц
     *
     * @param firstMatrix
     *         матрица, которую умножают
     *
     * @param secondMatrix
     *         матрица, на которую умножают
     *
     * @return результат умножения матриц или null, если размерности матриц не
     *         совпадают
     */
    public static double[][] multiplyByMatrix(double[][] firstMatrix,
        double[][] secondMatrix) {
        int m1ColLength = firstMatrix[0].length; // m1 columns length
        int m2RowLength = secondMatrix.length; // m2 rows length
        if (m1ColLength != m2RowLength)
            return null; // matrix multiplication is not possible
        int mRRowLength = firstMatrix.length; // m result rows length
        int mRColLength = secondMatrix[0].length; // m result columns length
        double[][] mResult = new double[mRRowLength][mRColLength];
        for (int i = 0; i < mRRowLength; i++) { // rows from m1
            for (int j = 0; j < mRColLength; j++) { // columns from m2
                for (int k = 0; k < m1ColLength; k++) { // columns from
m1
                    mResult[i][j] += firstMatrix[i][k] *
secondMatrix[k][j];
                }
            }
        }
        return mResult;
    }

    /**
     * Метод для получения матрицы поворота на угол alpha по оси Z
     *
     * @param alpha
     *         угол, на который производится поворот
     * @return матрицу поворота
     */
    public static double[][] rotateOnAngleByZ(double alpha) {
        double cosA = (double) Math.cos(Math.toRadians(alpha));
        double sinA = (double) Math.sin(Math.toRadians(alpha));

        double[][] matr = { { cosA, -sinA, 0.0f, 0.0f },
            { sinA, cosA, 0.0f, 0.0f }, { 0.0f, 0.0f, 1.0f, 0.0f },
            { 0.0f, 0.0f, 0.0f, 1.0f } };
        return matr;
    }

    /**
     * Метод для получения матрицы поворота на угол alpha по оси Y

```

```

*
* @param alpha
*         угол, на который производится поворот
* @return матрицу поворота
*/
public static double[][] rotateOnAngleByY(double alpha) // radians
{
    double cosA = (double) Math.cos(Math.toRadians(alpha));
    double sinA = (double) Math.sin(Math.toRadians(alpha));
    double[][] matr = { { cosA, 0.0f, sinA, 0.0f },
                        { 0.0f, 1.0f, 0.0f, 0.0f }, { -sinA, 0.0f, cosA, 0.0f },
                        { 0.0f, 0.0f, 0.0f, 1.0f } };
    return matr;
}

/**
* Метод для получения матрицы сдвига системы координат на (x,y,z)
*
* @param x
*         сдвиг по оси X
* @param y
*         сдвиг по оси Y
* @param z
*         сдвиг по оси Z
* @return матрицу сдвига
*/
public static double[][] moveOnXYZ(double x, double y, double z) //
radians
{
    double[][] matr = new double[][] { { 1, 0, 0, 0 }, { 0, 1, 0, 0 },
                                         { 0, 0, 1, 0 }, { x, y, z, 1 } };
    return matr;
}
}

```