Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА ИУ9

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Визуализация древовидных структур в динамике.

Студент	ИУ9-52Б		А.В. Никитин
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			А.В. Брагин
		(подпись, дата)	(И.О. Фамилия)

2024 г.

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ..... 2

ВВЕДЕНИЕ..... 3

ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ 5

ОСНОВНЫЕ ОСОБЕННОСТИ ДРЕВОВИДНЫХ СТРУКТУР 7

ПРАКТИЧЕСКИЙ РАЗДЕЛ 15

1.1 Выбор стека технологий.....15

1.2 Алгоритмы 16

1.3 Визуализация объектов 20

1.4 Тестирование.....23

ЗАКЛЮЧЕНИЕ 28

СПИСОК ЛИТЕРАТУРЫ..... 30

ВВЕДЕНИЕ

Задача визуализации объектов является актуальной, так как позволяет зрительно анализировать структуру этих объектов, их свойства, состояния и поведение. Визуализация данных широко используется в сфере научных и статистических исследований, а также применяется в современных образовательных материалах.

Учёные используют визуализацию как инструмент для представления результатов экспериментов или наблюдений. Визуализация позволяет преобразовать хаотичные числовые данные в графики, диаграммы и иные форматы, которые улучшают восприятие информации. Благодаря этому ученые могут выявлять закономерности, тенденции, аномалии, выдвигать и опровергать научные гипотезы, что помогает им оперативно достигать поставленных целей в науке. Зачастую в физике [1] используется демонстрация моделей и симуляций, что позволяет лучше понять динамику систем. В математике [2] визуализация используется для демонстрации всевозможных отображений между алгебраическими структурами. Часто в сфере информационных [3] технологий изображения объектов являются абстракциями, позволяющими более точно моделировать сложные системы, такие как, например, смартфон, компьютер или веб-сервис.

Отслеживание технических и продуктовых метрик в IT-компаниях происходит посредством динамической визуализации данных, полученных в результате сбора статистики сервисов или систем. В фармацевтических компаниях на основе визуализированных статистических данных производится оценка успешности проводимой работы в рамках разработки новых препаратов. Власти России используют визуализированную статистику для планирования и оценки социальных программ, а также анализа демографической ситуации в стране.

Образовательные учреждения в современные материалы подготовки обучающихся интегрировали изображения и анимации для упрощения сложных теоретических знаний и улучшения понимания. Графики, диаграммы и анимации помогают учащимся лучше усваивать материал и позволяют использовать зрительную память для последующего запоминания, что в свою очередь повышает эффективность обучения.

В рамках данной курсовой работы была рассмотрена задача визуализации древовидных структур в динамическом формате. Визуализация позволяет ознакомиться с процессом балансировки структуры данных, являющейся одной

из основных в мире компьютерных наук. Древовидные структуры являются важной частью информатики, они применяются в широком спектре технологий, обеспечивая продуктивное взаимодействие с данными посредством эффективного хранения и быстрого поиска.

ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

Предметной областью данной работы является изучение структур данных, представляющих собой важный аспект в области информационных технологий, и алгоритмов визуализации. Интеграция в образовательные программы, посредством демонстрации этой работы в общеобразовательных учреждениях, таких как лицеи и колледжи, может повысить эффективность развития профессиональных навыков у обучающихся при помощи объяснения фундаментальных понятий в области компьютерных технологий. В условиях стремительного развития информационных технологий знание структур данных становится не просто конкурентным преимуществом, а необходимым минимумом для будущего трудоустройства специалистов. Без этих знаний молодые специалисты столкнутся с серьезными сложностями при поиске работы ввиду того, что работодатели все чаще ищут кандидатов, обладающих навыками работы со структурами данных.

Современные стандарты работодателей требуют от программистов как теоретических знаний, так и практических навыков работы со структурами данных. Представленная визуализация структур данных помогает будущим специалистам лучше понять схему устройства деревьев, а также принцип их работы, что способствует более качественному усвоению преподаваемых материалов, что в свою очередь является толчком к получению фундаментальных теоретических знаний. Визуализация будет не менее полезна для преподавателей, которые смогут интегрировать демонстрацию в образовательный процесс, что позволит показывать учащимся сложные теоретические данные на простом примере.

Целью данной работы является систематизация существующих знаний о древовидных структурах данных, а также разработка и реализация алгоритма визуализации древовидных структур в динамике.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) Провести изучение предметной области, чтобы получить полное представление о различных типах древовидных структур, таких как AVL-деревья, красно-черные деревья, B-деревья, R-деревья и декартовы деревья.
2) Подобрать соответствующий стек технологий, необходимый для реализации алгоритмов визуализации.

3) Разработать и реализовать алгоритм визуализации, который позволит динамически отображать изменения в древовидной структуре в реальном времени.
4) Провести тестирование разработанного программного обеспечения.

ОСНОВНЫЕ ОСОБЕННОСТИ ДРЕВОВИДНЫХ СТРУКТУР

Для начала требуется ввести основные определения, такие как дерево, узел, ребро. Дерево — это не пустой набор ребер и вершин, которые соответствуют некоторому критерию. Узел (вершина) – объект, быть может, содержащий некоторую информацию. Ребро – связующий элемент двух вершин. Набор вершин и ребер может называться деревом тогда и только тогда, когда существует единственный список элементов дерева, соединяющих любые две вершины. Также древовидную структуру можно охарактеризовать как связный ациклический граф с неориентированными связями. Корнем дерева называют вершину, которая является для каждой вершины потомком, однако сама потомков не имеет. Бинарным деревом называют дерево, в котором у любой вершины есть связь, располагающаяся ближе к корню, а также, быть может, еще две связи, удаленные на один порядок дальше от корня.

Структура данных “бинарное дерево” или древовидная структура данных – это представление двоичного дерева в памяти компьютера. Двоичное дерево поиска – это дерево, в котором у любого узла есть значение, и это значение больше или равно значениям в левом поддереве и меньше любого из значений правого поддерева.

Рассмотрим основных представителей древовидных структур. Каждый из представителей класса деревьев имеет как преимущества, так и недостатки в зависимости от поставленных задач и условий применения.

AVL-дерево – сбалансированное бинарное дерево поиска, в котором высоты поддеревьев любого узла отличаются менее чем на 2. Данная структура является разработкой советских ученых Адельсона, Вельского и Ландиса. Ученые в 1962 опубликовали в научном журнале концепцию самобалансирующегося бинарного дерева поиска, которое могло автоматически поддерживать сбалансированность после операций вставки и удаления. Деревья используют операции вращения (левое и правое вращение) для восстановления сбалансированности дерева. Это позволяет поддерживать логарифмическую сложность операций поиска, вставки и удаления. Аббревиатура АВЛ соответствует первым буквам фамилий этих ученых. Первоначально структура данных была придумана для организации перебора в шахматных программах, однако сыскала свою популярность в образовательных материалах. Использование AVL-дерева в учебных курсах по алгоритмам и структурам данных помогло распространить знания об этой структуре, что впоследствии стало основой для многих исследований области

самобалансирующихся деревьев. Именно создание AVL-дерева можно считать толчком к созданию аналогичных структур данных, таких как, например, красно-черное дерево. Рисунок № 1 иллюстрирует описанную выше структуру данных с 11 вершинами и 10 ребрами.

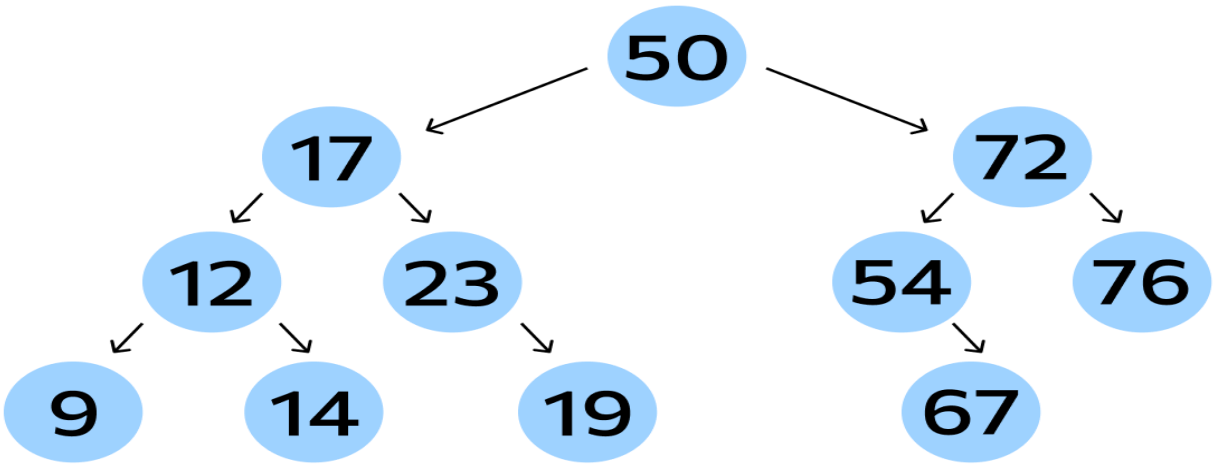

Рисунок №1 — AVL-дерево.

Спустя десятилетие немецкий ученый Рудольф Байер на основе AVL-дерева создает Красно-черное дерево – бинарное дерево поиска, в котором для выполнения балансировки все вершины окрашены либо в черный, либо красный цвет. Все листья дерева являются фиктивными и не содержат данных, но относятся к дереву и являются чёрными. Для экономии памяти фиктивные листья делают одним общим фиктивным листом. Красно-черные деревья получили более широкое распространение по сравнению с AVL-деревом: стандартные контейнеры в C++ STL, например std::map, ядро Linux. В красно-чёрном дереве обязательно должны выполняться следующие свойства:

- каждый узел промаркирован красным или чёрным цветом;
- корень и конечные узлы (листья) дерева — чёрные;
- у красного узла родительский узел — чёрный;
- все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов.

На рисунке № 2 изображено красно-черного дерево с 10 обычными вершинами, 11 фиктивными вершинами и 20 ребрами.

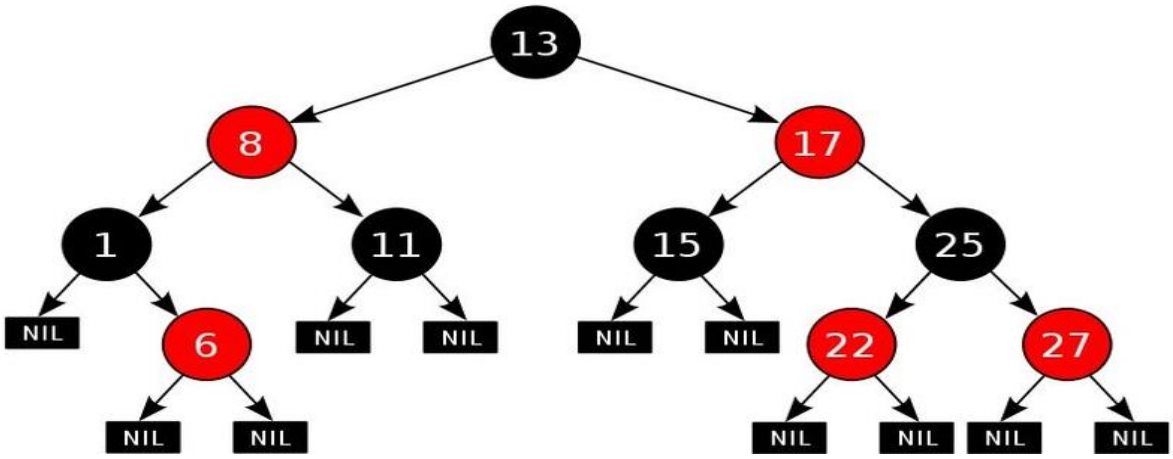

Рисунок № 2 — Красно-чёрное дерево.

Сравнение AVL-деревьев и красно-черных деревьев показывает:

- AVL-деревья более строго сбалансированы, быстрее выполняют поиск элемента. Максимальная высота ~1.44*$\log_2 n$;
- красно-черные деревья быстрее выполняют вставку и удаление элемента, возможно именно поэтому более популярны в качестве технического решения;
- AVL-деревья хранят в узлах значение баланса или высоту узла, нужно тратить память на целочисленную переменную.
- красно-черные деревья хранят в узлах еще цвет.

На рисунке № 3 изображена оценка сверху высот AVL-дерева и красно-черного дерева.

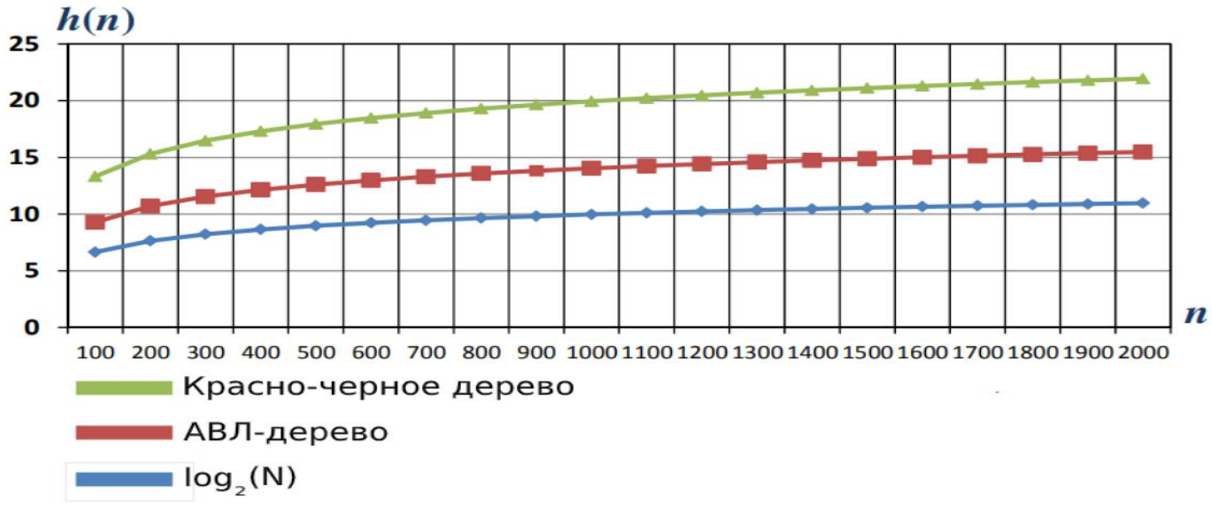

Рисунок № 3 — Сравнение AVL и красно-чёрного деревьев.

Еще одним популярным представителем класса древовидных структур является B-дерево. Использование B-деревьев впервые было предложено Р. Байером и Э. МакКрейтом в 1970 году, за пару лет до создания красно-черного дерева. Эта структура оптимизирует работу с диском, минимизируя число операций чтения и записи. На данный момент многие БД хранят данные в B-деревьях.

B-дерево может применяться для структурирования информации на жёстком диске. Время доступа к произвольному блоку на жёстком диске очень велико, порядка миллисекунд, поскольку оно определяется скоростью вращения диска и перемещения головок. Поэтому важно уменьшить количество узлов, просматриваемых при каждой операции. Использование поиска по списку каждый раз для нахождения случайного блока могло бы привести к чрезмерному количеству обращений к диску вследствие необходимости последовательного прохода по всем его элементам, предшествующим заданному, тогда как поиск в B-дереве, благодаря свойствам сбалансированности и высокой ветвистости, позволяет значительно сократить количество таких операций. Относительно простая реализация алгоритмов и существование готовых библиотек, в том числе для C, для работы со структурой B-дерева обеспечивают популярность применения такой организации памяти в самых разнообразных программах, работающих с большими объёмами данных, например PostgreSQL и MySQL.

B-дерево является идеально сбалансированным, то есть глубина всех его листьев одинакова. B-дерево имеет следующие свойства (t - параметр дерева, называемый минимальной степенью B-дерева, неменьший 2.):

- каждый узел, кроме корня, содержит не менее $t-1$ ключей, и каждый внутренний узел имеет по меньшей мере t дочерних узлов. Если дерево не является пустым, корень должен содержать как минимум один ключ;
- каждый узел, кроме корня, содержит не более $2t-1$ ключей и не более чем 2t сыновей во внутренних узлах;
- корень содержит от 1 до $2t-1$ ключей, если дерево не пусто и от 2 до 2t детей при высоте большей 0;
- каждый узел дерева, кроме листьев, содержащий ключи k_1, ..., k_n, имеет $n+1$ сына. i-й сын содержит ключи из отрезка $[k_{i-1}; k_i]$, $k_0 = -\infty$, $k_{n+1} = \infty$;
- ключи в каждом узле упорядочены по не убыванию;
- все листья находятся на одном уровне.

На рисунке № 3 изображено b-дерево глубины 3.

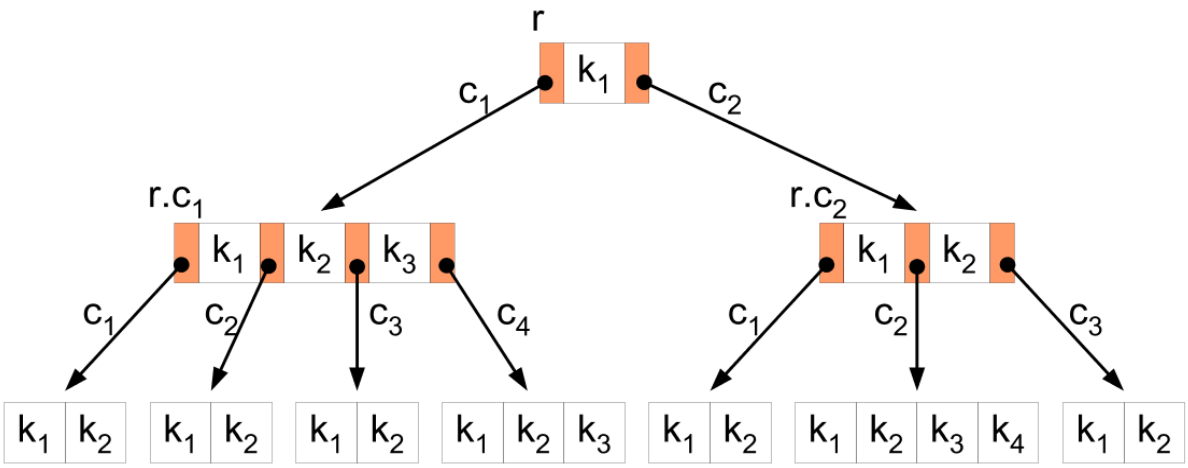

Рисунок № 4 — B-дерево.

Менее популярным, но не менее эффективным в своей сфере является R-дерево — древовидная структура данных, предложенная в 1984 году Антонином Гуттманом. Эта структура данных разработана специально для организации доступа к пространственным данным, это делает ее особенно полезной в контексте использования многомерных данных. В отличие от традиционных деревьев, таких как B-деревья или AVL-деревья, которые в основном используются для индексирования одномерных данных, R-деревья оптимизированы для работы с многомерными данными, которые могут иметь от двух координат. Например, R-дерево может включать в себя данные, связанные с широтой и долготой, что делает эту структуру идеальной для работы с информацией карт и другими пространственными данными. Пример такого дерева можно увидеть на рисунке № 5.

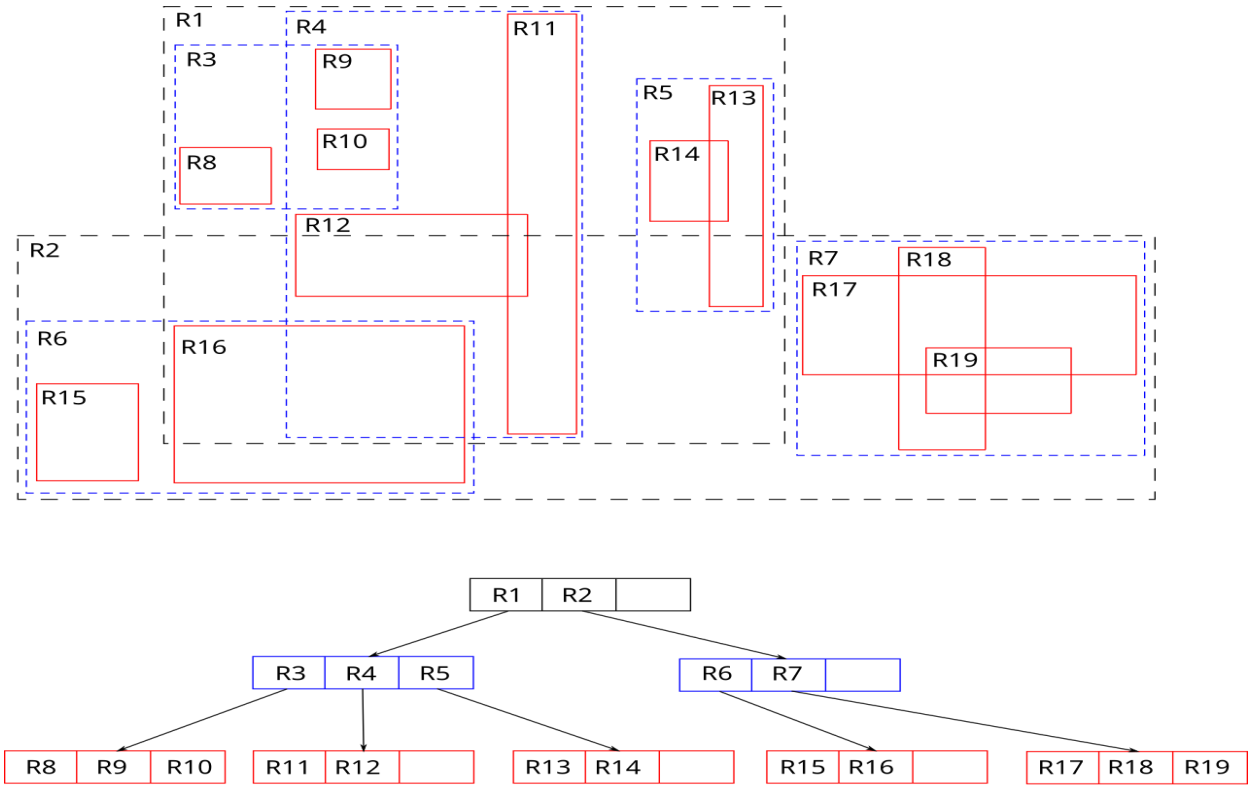

Рисунок № 5 — R-дерево.

R-дерево целесообразно использовать в приложениях, подобных “Яндекс Go”, где необходимо обрабатывать множество элементов, которые перемещаются с течением времени. Это может включать в себя информацию о такси, доставке еды и товаров или других сервисах, где важно отслеживать местоположение объектов в реальном времени. В таких случаях R-дерево позволяет быстро находить объекты, находящиеся в определенном диапазоне координат, что как следствие ускоряет процесс поиска и обработки многомерных данных.

Вместе с тем R-деревья находят широкое применение в компьютерных играх формата мультиплеер, в которых необходимо отображать множество движущихся объектов на экране. В таких играх игроки взаимодействуют как со статическими, так и с динамическими элементами, и R-дерево эффективно управляет пространственными запросами, такими отбор объектов или игроков, находящихся в пределах видимости главного игрока. Это позволяет обеспечить эффективное использование ресурсов сервера для корректной обработки отображения объектов.

Существует также подвид R-дерева — R*-дерево, которое было разработано для улучшения эффективности поиска по пространственным данным. В отличие от стандартного R-дерева, R*-дерево использует более сложные алгоритмы для балансировки географических данных, что позволяет

значительно оптимизировать только операции поиска. Это особенно полезно в случаях, когда количество операций чтения данных превосходит количество операций записи.

Использование R*-деревьев целесообразно в приложениях, подобных “Google Maps” или “Яндекс Карты”, где необходимо отображать статичные объекты с большим количеством деталей. В таких приложениях пользователи часто ищут информацию о различных местах, таких как кафе, рестораны, магазины, станции метро и другие объекты инфраструктуры города или страны. R*-дерево позволяет эффективно обрабатывать запросы пользователей, обеспечивая навигацию по картам.

Таким образом, R-деревья и их подвид — R*-дерево, используются в широком спектре областей современных технологий, от геоинформационных систем до компьютерных игр.

Еще одним представителем деревьев является декартово дерево или дерамида — это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. Это бинарное дерево, в узлах которого хранятся пары (x, y), где x — это ключ, а y — это приоритет. Также оно является двоичным деревом поиска по x и пирамидой по y. Предполагая, что все x и все y являются различными, получаем, что если некоторый элемент дерева содержит (x_0, y_0), то у всех элементов в левом поддереве $x < x_0$, у всех элементов в правом поддереве $x > x_0$, а также и в левом, и в правом поддереве имеем: $y < y_0$.

Это дерево может быть использовано в различных алгоритмах сжатия данных, хотя оно не являются наиболее распространённым выбором для этой задачи. Например, алгоритм Хаффмана. Хотя сам алгоритм Хаффмана обычно реализуется с использованием бинарных деревьев или куч, декартовые деревья могут быть использованы для хранения частот символов. После построения дерева Хаффмана декартово дерево может помочь в упрощении доступа к частотам и их обновлению.

Хотя реализация данной структуры обладает определенными преимуществами, такими как легкость исполнения, существуют и некоторые недостатки, которые следует учитывать:

- большие накладные расходы на хранение: вместе с каждым элементом хранятся два-три указателя и случайный ключ y;
- скорость доступа O(n) в худшем, хотя и маловероятном, случае. Поэтому декартово дерево недопустимо, например, в ядрах ОС.

На рисунке № 6 представлен пример декартового дерева с 10 узлами.

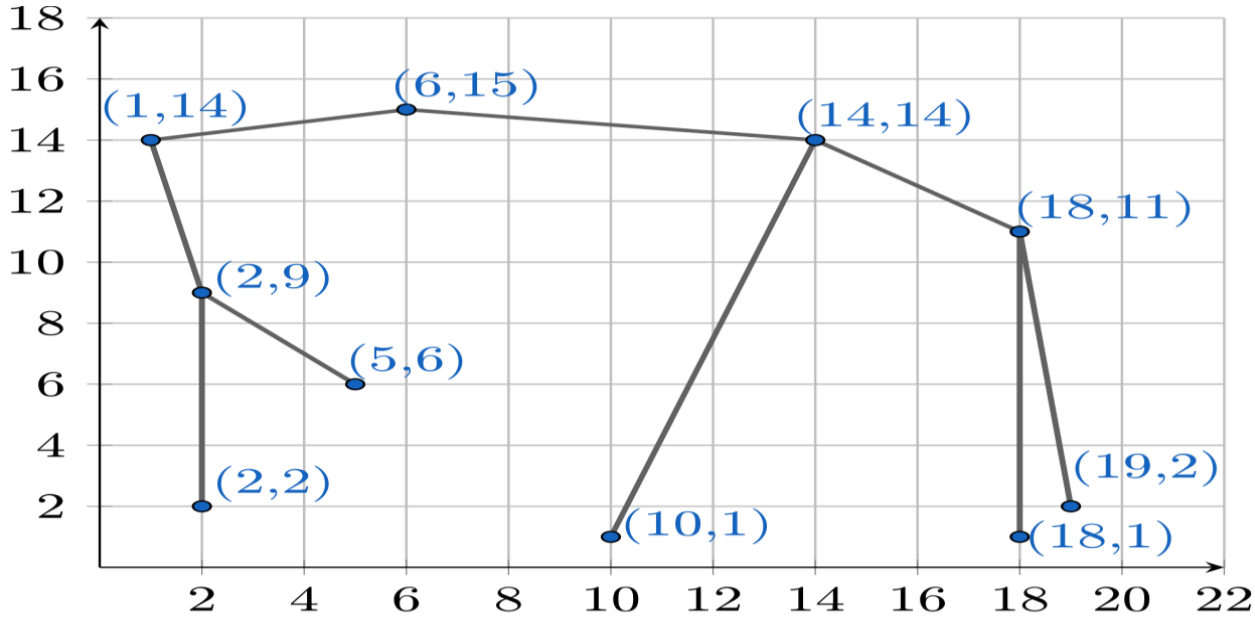

Рисунок № 6 — Декартово дерево.

ПРАКТИЧЕСКИЙ РАЗДЕЛ

1.1 Выбор стека технологий

Для реализации алгоритма визуализации был выбран язык программирования Python [4], который имеет обширное количество библиотек для графического представления данных.

В качестве основной библиотеки для компьютерной графики была выбрана библиотека Pygame [5]. Pygame — набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений, а также графических представлений. Pygame базируется на мультимедийной библиотеке SDL. Изначально Pygame был написан Питом Шиннерсом, однако начиная примерно с 2004/2005 года поддерживается и развивается сообществом свободного программного обеспечения. Одна из библиотек, предоставляющих доступ к API SDL. В то же время дает возможность написания более высокоуровневого кода. Pygame-приложения могут работать под Android на телефонах и планшетах с использованием подмножества Pygame для Android. На этой платформе поддерживаются звук, вибрация, клавиатура, акселерометр.

Simple Direct Media Layer (SDL) — это кроссплатформенная библиотека разработки, предназначенная для предоставления низкоуровневого доступа к аудио, клавиатуре, мыши, джойстику и графическому оборудованию через OpenGL и Direct3D. Она используется программным обеспечением для воспроизведения видео, эмуляторами и популярными играми, включая отмеченный наградами каталог Valve и множество игр Humble Bundle.

Стандарт OpenGL [6] (Open Graphics Library – графическая библиотека с открытым исходным кодом), который был создан IT-организациями, ведущими свою основную деятельность в области разработки программного обеспечения, в качестве универсального кроссплатформенного интерфейса. Эта библиотека предоставляет обширный набор функций и процедур, которые дают возможность работать с графическими примитивами при написании высокоуровневого программного обеспечения.

OpenGL впервые была представлена в 1992 году компанией Silicon Graphics Inc., быстро завоевав популярность в индустрии разработки компьютерных игр, а также программ, визуализирующих данные посредством работы с графическими примитивами. Суть работы OpenGL заключается в том, что библиотека предоставляет возможность работы с графическим процессором компьютера через библиотечные функции.

OpenGL включает более 300 функций для рисования сложных трёхмерных сцен из простых графических примитивов. Несмотря на значительный вклад в развитие графических технологий, OpenGL больше не развивается. Последняя версия OpenGL выпущена в 2017 году.

В ходе разработки алгоритма для упрощения запуска программного обеспечения использовалась технология Makefile. Makefile представляет собой текстовый файл, содержащий набор инструкций, которые описывают, как компилировать и связывать программы, а также как выполнять различные задачи, связанные с управлением проектом. Файл с названием Makefile с инструкциями последовательного запуска файлов, написанных на языке программирования Python, располагается в директории проекта для последующего использования. Makefile выполняется при помощи команды make, которая прописывается в командной строке, например, make [options] [target1 target2 ...]. По умолчанию, когда утилита make ищет make-файл и если имя make-файла не было указано в качестве параметра, то make пробует следующие имена по порядку: makefile и Makefile.

1.2 Алгоритмы

Одним из алгоритмов, представленных в данной курсовой работе, является функция, отвечающая за отображение круга с градиентной заливкой. Этот алгоритм создает визуальный эффект, придавая объем создаваемой вершине. В листинге № 1 представлен пример кода функции, которая принимает на вход экран для рисования, координаты центра круга и исходный цвет. Данная функция в точке, указанной как центр, создает круг радиусом 50 пикселей с заданным цветом.

Особенностью реализации является то, что цвет каждого последующего круга вычисляется путем уменьшения значений красного, зеленого и синего компонентов исходного цвета в зависимости от радиуса. Это создает эффект градиента, в котором внутренние круги имеют более насыщенный и яркий цвет, тогда как внешние круги становятся более темными и менее насыщенными. Таким образом, создаваемый градиент не только улучшает восприятие объекта, но и визуально создает эффект трехмерного изображения, который делает объект объемным и более реалистичным.

Листинг № 1 – Функция отображения вершины с градиентной заливкой.

```
def draw_gradient_circle(screen, center, color):
  def draw_circles_with_differences(start_radius, end_radius, step, factor):
    for r in range(start_radius, end_radius, step):
```

```
        difference_0 = color[0] - (color[0] // 50) * r * factor
        difference_1 = color[1] - (color[1] // 50) * r * factor
        difference_2 = color[2] - (color[2] // 50) * r * factor
        new_color = (
          max(0, difference_0),
          max(0, difference_1),
          max(0, difference_2)
        )
        pygame.draw.circle(screen, new_color, center, r)

draw_circles_with_differences(50, 48, -1, 0.4)
draw_circles_with_differences(48, 45, -1, 0.3)
draw_circles_with_differences(45, 40, -1, 0.2)
draw_circles_with_differences(40, 0, -1, 0.1)
```

Для визуализации множества вершин был разработан алгоритм, который отвечает за отображение всех вершин в соответствии с их позициями и номерами. Эти данные хранятся в глобальной переменной circle_positions, которая представляет собой массив массивов. Каждая запись в этом массиве содержит номер вершины, а также значения координат x и y, которые определяют её положение на экране. Это позволяет легко и эффективно управлять, и отображать множество вершин, обеспечивая их корректное расположение в зависимости от заданных координат.

Кроме того, для визуализации множества линий был реализован отдельный алгоритм, который отвечает за отображение каждой линии в соответствии с сохраненными данными, которые хранятся в глобальной переменной lines_positions. Эта переменная также представляет собой массив массивов, где каждая линия описывается набором координат, определяющих её начальную и конечную точки по осям x и y. Такой подход позволяет рисовать линии между заданными точками, обеспечивая визуальную целостность и правильное отображение древовидной структуры.

Код функций, реализующих алгоритмы, представлен в листинге № 2. В листинге содержатся две основные функции: draw_lines и draw_circle_positions.

Функция draw_lines принимает параметры, такие как экран для отображения, цвет линий и массив, содержащий линии. Цвет и массив являются необязательными параметрами, если данные параметры не переданы в функцию, то используются значения по умолчанию. Для массива линий значением по умолчанию является глобальная переменная lines_positions, для цвета линий по умолчанию используется черный. Внутри функции происходит

итерация по всем позициям линий, и для каждой линии, которая не находится в списке удаляемых линий, выполняется отрисовка.

Функция draw_circle_positions отвечает за отображение всех вершин, находящихся в глобальной переменной circle_positions. Внутри этой функции происходит перебор всех позиций кругов, и для каждой из них проверяется, находится ли она в списках для удаления. В случае, если номер вершины находится в списке delete_tree_pool, то вершина окрашивается в красный цвет перед полным удалением с экрана. В случае, если номер вершины находится в delete_circle_pool, то она вовсе не рисуется на экране. В остальных случаях происходит стандартная отрисовка вершины.

Таким образом, обе функции обеспечивают отображение вершин и линий на экране.

Листинг № 2 – Функции отображения всех вершин и всех линий.

```
def draw_lines(screen: pygame.Surface, color=None, lines_positions_func=None):
    if lines_positions_func is None:
        lines_positions_func = lines_positions
    if color is None:
        color = BLACK
    for i in range(len(lines_positions_func)):
        if i in delete_lines_pool:
            continue
        pygame.draw.line(screen, color, (lines_positions_func[i][0], lines_positions_func[i][1]),
                         (lines_positions_func[i][2], lines_positions_func[i][3]))

    return

def draw_circle_positions(screen: pygame.Surface):
    for pos in circle_positions:
        if pos[0] in delete_circle_pool:
            continue
        if pos[0] in delete_tree_pool:
            draw_circle_with_text(pos[0], int(pos[1]), int(pos[2]), screen, (255, 0, 0))
        else:
            draw_circle_with_text(pos[0], int(pos[1]), int(pos[2]), screen)
    return
```

В дополнение к ранее описанным функциям, в проекте также используется функция draw_first(), представленная в листинге № 3. Эта функция предназначена для анимации движения первой вершины, анимация

позволяет визуально отслеживать её перемещение по экрану. В функции реализован алгоритм, который управляет передвижением вершины с помощью переменной speed_x_1, определяющей скорость её перемещения по оси x.

Цикл анимации продолжается до тех пор, пока центр круга не достигнет указанной конечной точки. Внутри цикла происходит несколько ключевых действий. Сначала экран заполняется белым цветом для обновления визуализации. Далее вызываются функции draw_circle_positions() и draw_lines(), которые описаны ранее и которые отвечают за отрисовку всех вершин и линий на экране. После этого происходит смещение центра вершины на значение, заданное переменной speed_x_1, что в последствии приводит к движению вершины по экрану. Как только центр вершины достигает предопределенной точки на экране, информация о новой позиции вершины добавляется в массив circle_positions, который хранит все вершины, после чего функция завершается.

Кроме того, в функции предусмотрено отрисовка самой вершины с помощью draw_circle_with_text() до того, как будет достигнута финальная точка, отрисовка позволяет визуально представить её текущее положение. После этого обновляется экран с помощью pygame.display.flip(), и цикл повторяется, обеспечивая плавную анимацию.

Другие функции, которые используются в проекте и которые реализуют динамическую визуализацию передвижения вершин, используют аналогичные алгоритмы, но отличаются схемой движения, конечной точкой и дополнительными действиями. Например, по достижении нужной точки может происходить отрисовка линии между новой и родительской вершинами, а также добавление этой линии в массив, который хранит все ребра между вершинами.

Листинг № 3 – Функция отображения первой вершины в динамике.

```
def draw_first(screen: pygame.Surface, clock: pygame.time.Clock):
    running = True
    circle_pos_1 = ["1", 200, 100]
    speed_x_1 = 5

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        screen.fill(WHITE)
        draw_circle_positions(screen)
```

```
draw_lines(screen)
circle_pos_1[1] += speed_x_1
if circle_pos_1[1] == 750:
    circle_positions.append(circle_pos_1)
    return

draw_circle_with_text(circle_pos_1[0], int(circle_pos_1[1]), int(circle_pos_1[2]), screen)
pygame.display.flip()
clock.tick(60)
```

1.3 Визуализация объектов

После разработки и реализации алгоритмов визуализации графических примитивов, согласно поставленным в данной курсовой работе целям, необходимо перейти к этапу визуализации балансировки древовидной структуры в реальном времени. Для этого было выбрано AVL-дерево, которое является одной из наиболее распространенных и эффективных структур данных для хранения элементов. AVL-деревья обладают быстрой вставкой, удалением благодаря своим вращениям. Существуют малое правое, малое левое, а также большое правое, большое левое вращения.

В рамках данной работы требуется не только отобразить базовые операции, такие как вращения дерева, но и визуализировать произвольные добавления и удаления вершин с последующей балансировкой структуры, так как AVL-деревья должны поддерживать определенное соотношение высот между поддеревьями для обеспечения свойства сбалансированности.

На рисунке № 7 представлено несбалансированное бинарное дерево поиска до выполнения малого правого вращения. Визуализация этого дерева позволяет наблюдать за его структурой и узлами, а также понять, как именно происходит распределение значений и как они связаны друг с другом.

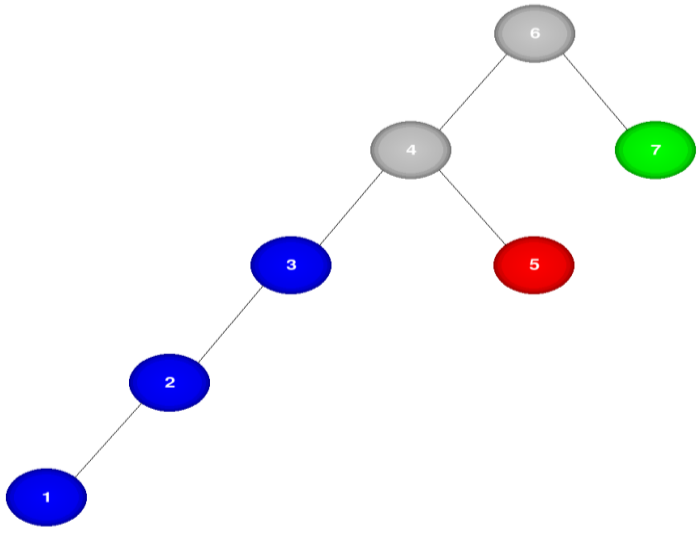

Рисунок № 7 — Визуализация дерева до малого правого вращения.

После выполнения вращения, в ходе визуализации, дерево,

представленное на рисунке № 7, становится сбалансированным AVL-деревом. Этот процесс вращения позволяет перераспределить узлы таким образом, чтобы разница высот поддеревьев не превосходила единицу, так как это является основным требованием для поддержания сбалансированности AVL-дерева. Результат этого процесса также демонстрируется на рисунке № 8, где можно наблюдать, как изменились связи между узлами, и как структура дерева стала оптимальной для выполнения операций поиска, вставки и удаления.

Таким образом, визуализация вращений в AVL-дереве позволяет наглядно увидеть, как происходит сам процесс балансировки, который является основным способом поддержания эффективной работы данной структуры.

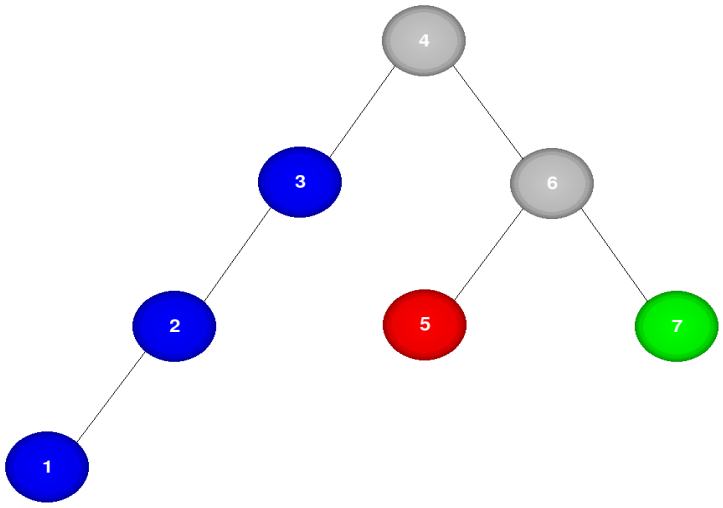

Рисунок № 8 — Визуализация дерева после малого правого вращения.

Для реализации произвольного добавления и удаления вершин было создано AVL-дерево с пятнадцатью вершинами, которое балансируется при добавлении каждой из пятнадцати вершин, а также балансируется после удаления любой из вершин. На рисунке № 9 изображено AVL-дерево после добавления всех пятнадцати вершин, рисунок демонстрирует его сбалансированную структуру и оптимальное распределение узлов. Каждое добавление вершины сопровождается проверкой высот поддеревьев, и, при необходимости, выполняются соответствующие вращения для поддержания баланса.

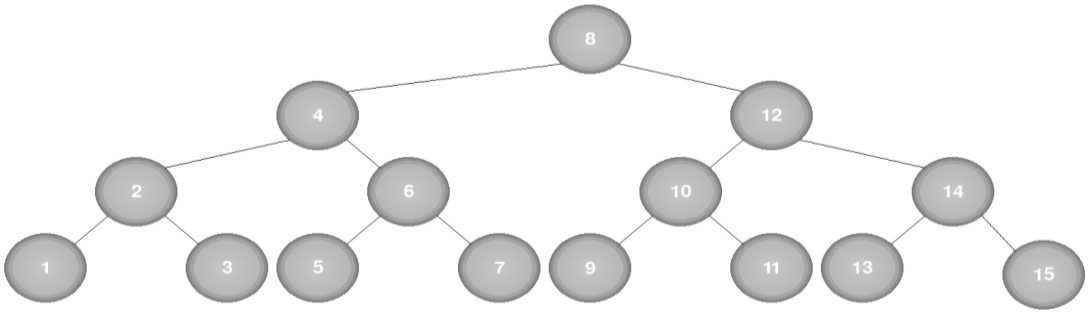

Рисунок № 9 — Визуализация AVL-дерева с пятнадцатью вершинами.

Далее происходит поочередное удаление узлов дерева с предварительной окраской удаляемой вершины в красный цвет. Это позволяет визуально выделить конкретные вершины, которые будут удалены, и облегчает понимание процесса удаления. Удаление узлов из AVL-дерева требует особого внимания, так как после каждого удаления свойство сбалансированности дерева может быть утрачено. Если высота одного из поддеревьев становится больше, чем разрешенные пределы, выполняются соответствующие вращения для восстановления баланса.

Результат удаления некоторого количества узлов из AVL-дерева, изображенного на рисунке № 9, представлен на рисунке № 10. На этом рисунке изображено как изменяется структура дерева после удаления узлов. Каждый раз, когда удаляется вершина, происходит перераспределение значений, и, в зависимости от того, какая вершина была удалена, могут потребоваться различные типы вращений для восстановления свойства сбалансированности дерева. Таким образом, визуализация этих изменений помогает лучше понять динамическое изменение AVL-деревьев и важность поддержания их сбалансированности для обеспечения эффективных операций поиска, вставки и удаления.

Кроме того, такой подход к визуализации предоставляет практическое понимание алгоритмических процессов, происходящих в динамически измененяемых древовидных структурах.

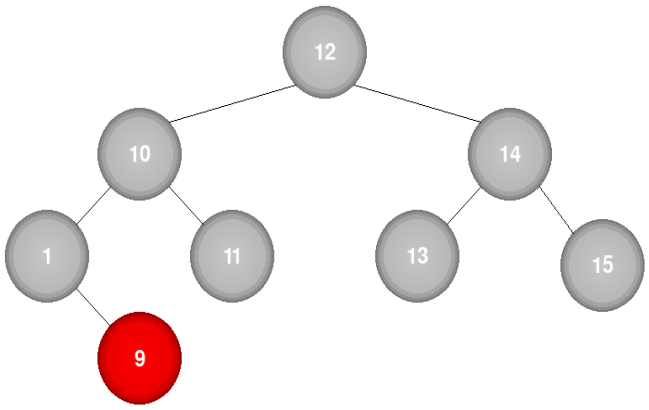

Рисунок № 9 — Визуализация AVL-дерева с пятнадцатью вершинами.

1.4 Тестирование

Тестирование кода является необходимым условием обеспечения качества разработки программного обеспечения. Без тестирования программного обеспечения не обходится ни один серьезный проект. Оно позволяет выявить ошибки и недочеты на ранних этапах, что в последствии экономит время. В основном тестирование делится на функциональное и нефункциональное. Функциональное тестирование делится на модульное тестирование, интеграционное тестирование и системное тестирование, в то время как нефункциональное тестирование оценивает характеристики, такие как производительность, безопасность и удобство использования.

Модульное тестирование фокусируется на отдельных компонентах или модулях программы, что позволяет выявить ошибки на раннем этапе разработки. Интеграционное тестирование проверяет взаимодействие между модулями, а системное тестирование охватывает всю систему в целом, проверяя ее соответствие функциональным и нефункциональным требованиям.

В случае реализации тестирования проекта данной курсовой работы целесообразным выбором будет модульное тестирование, которое позволит протестировать работу основных функций, изображающих графические примитивы. Модульное тестирование обеспечивает изолированную проверку каждого компонента, такой подход позволяет быстрее находить и исправлять ошибки. Кроме того, модульные тесты могут выполняться автоматически, что ускоряет процесс тестирования.

Листинг № 4 представляет собой класс для тестирования функций

проекта, использующего библиотеку Pygame для графической отрисовки. В этом классе определены несколько методов, каждый из которых предназначен для проверки конкретной функции, связанной с рисованием на экране.

Кроме того, в листинге № 4 методы вызывают функций отрисовки, после чего сохраняют изображения, которые позволяют визуально проверить корректность работы функций изображения. А также в тестах вызывается функция draw_first() для проверки работоспособности и визуального сравнения ожидаемого результата выполнения функции с фактическим. Это дает возможность не только удостовериться в корректности работы кода, но и оценить его визуальные аспекты.

Листинг № 4 – Класс для тестирования функций проекта

```
class TestDrawingFunctions(unittest.TestCase):
    def setUp(self):
        pygame.init()
        self.clock = pygame.time.Clock()
        display = (1500, 1000)
        self.screen = pygame.display.set_mode(display)

    def test_draw_circle_with_text(self):
        draw_circle_with_text("Test", 100, 100, self.screen, (255, 0, 0))
        draw_circle_with_text("Test", 300, 100, self.screen, (0, 255, 0))
        draw_circle_with_text("Test", 500, 100, self.screen, (0, 0, 255))
        pygame.image.save(self.screen, "red_green_blue_circle.png")

    def test_gradient_circle_without_text(self):
        draw_gradient_circle(self.screen, (100, 100), 50, (200, 200, 200))
        draw_gradient_circle(self.screen, (200, 100), 50, (255, 0, 0))
        draw_gradient_circle(self.screen, (300, 100), 50, (0, 255, 0))
        draw_gradient_circle(self.screen, (400, 100), 50, (0, 0, 255))
        pygame.image.save(self.screen, "gradient_without_text.png")

    def test_draw_lines(self):
        lines_positions = [[100, 100, 100, 200], [200, 100, 200, 200], [300, 100, 300, 200], [400,
100, 400, 200], [100, 100, 400, 200]]
        draw_lines(self.screen, (255, 255, 255), lines_positions)
        pygame.image.save(self.screen, "lines_positions.png")

    def test_draw_first(self):
        draw_first(self.screen, self.clock)

    def tearDown(self):
        pygame.quit()
```

В методе setUp инициализируется Pygame, создается объект Clock для управления частотой кадров, а также задается размер окна отображения через. Это обеспечивает необходимую среду для тестирования графических функций, которые будут вызываться в последующих тестах.

Метод test_draw_circle_with_text проверяет функцию, которая должна рисовать круги с текстом на экране. В этом тесте создаются три круга различных цветов: красный, зеленый и синий. Каждый из них располагается на заданных координатах, и в конце теста изображение сохраняется в файл с именем "red_green_blue_circle.png". Это позволяет визуально проверить, что функция работает, производя ожидаемый результат. На рисунке № 11 изображены три вершины с градиентной заливкой и с текстом, что соответствует ожидаемому результату выполнения функции draw_gradient_circle() с входными параметрами color=(255,0,0), (0,255,0),(0,0,255) и textInCircle =“Test” в тесте test_draw_circle_with_text().

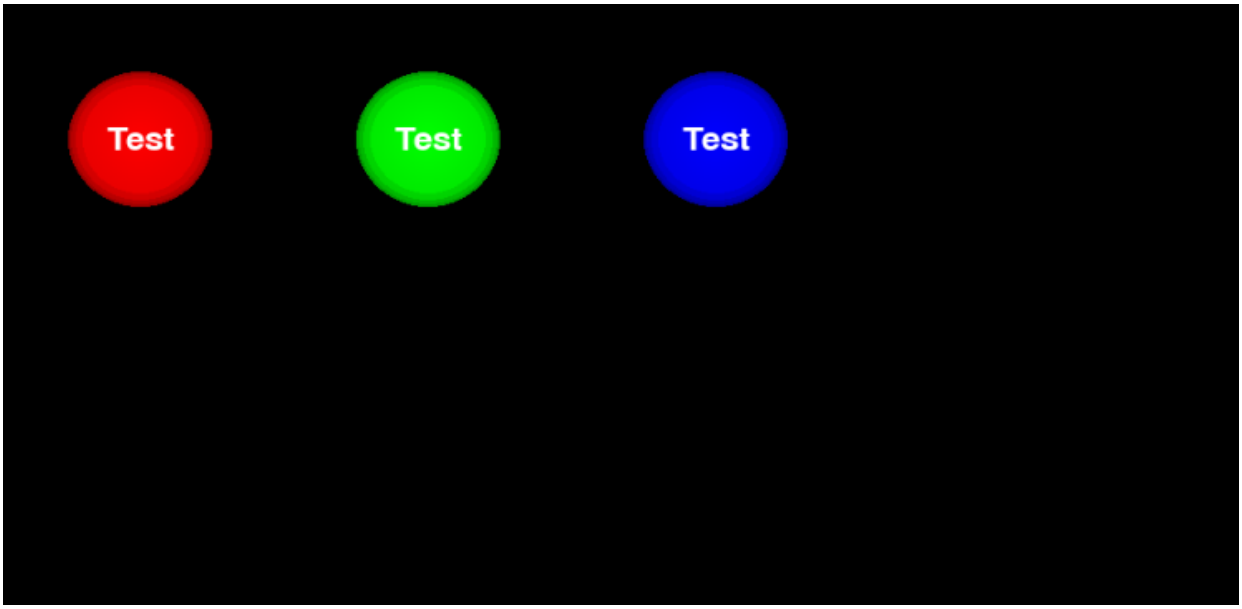

Рисунок № 11 — Содержание файла red_green_blue_circle.png.

Следующий тест — это test_gradient_circle_without_text, который проверяет функцию draw_gradient_circle, рисующую круги с градиентной заливкой, но без текста. В этом методе создаются четыре круга с различными цветами градиента: светло-серый, красный, зеленый и синий. Результат также сохраняется в изображение под названием "gradient_without_text.png". Этот тест позволяет убедиться, что функция правильно реализует градиентную заливку. На рисунке № 12 изображены четыре вершины без текста с градиентной заливкой, что соответствует ожидаемому результату выполнения функции draw_gradient_circle() с входными параметрами color=(200,200,200),(255,0,0),

(0,255,0),(0,0,255).

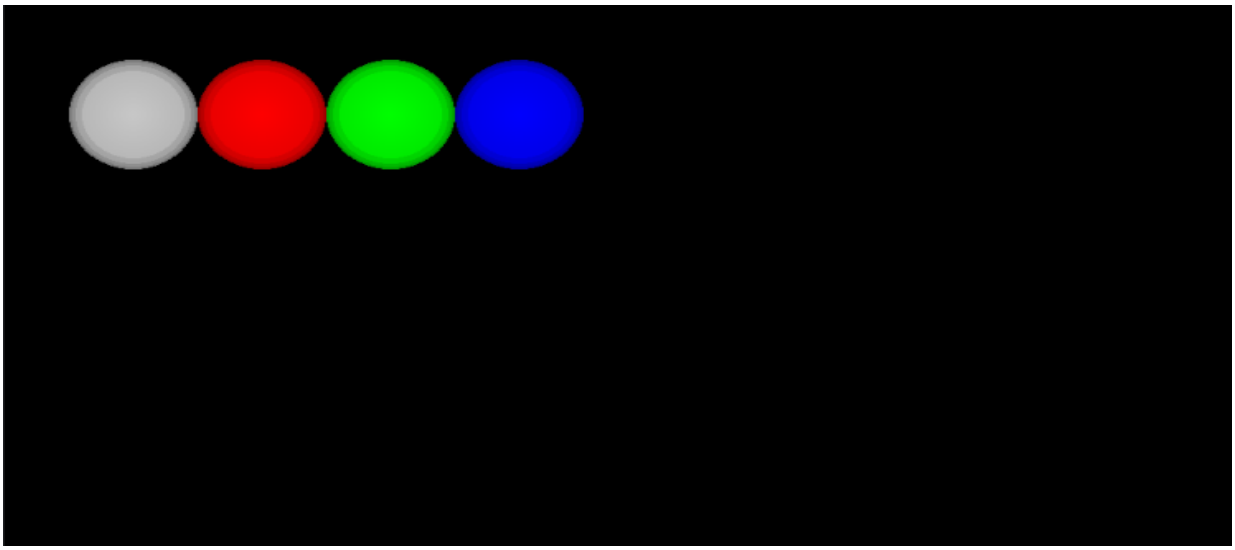

Рисунок № 12 — Содержание файла gradient_without_text.png.

Метод test_draw_lines тестирует функцию draw_lines, которая рисует линии на экране. В этом тесте задаются координаты для нескольких линий, после чего они рисуются на экране белым цветом. Результат сохраняется в файл "lines_positions.png", что позволяет проверить, что линии рисуются корректно и соответствуют заданным координатам. На рисунке № 13 изображены пять линий, что соответствует ожидаемому результату выполнения функции draw_lines() с входными параметрами lines_positions = [[100, 100, 100, 200], [200, 100, 200, 200], [300, 100, 300, 200], [400, 100, 400, 200], [100, 100, 400, 200]] в тесте test_draw_lines().

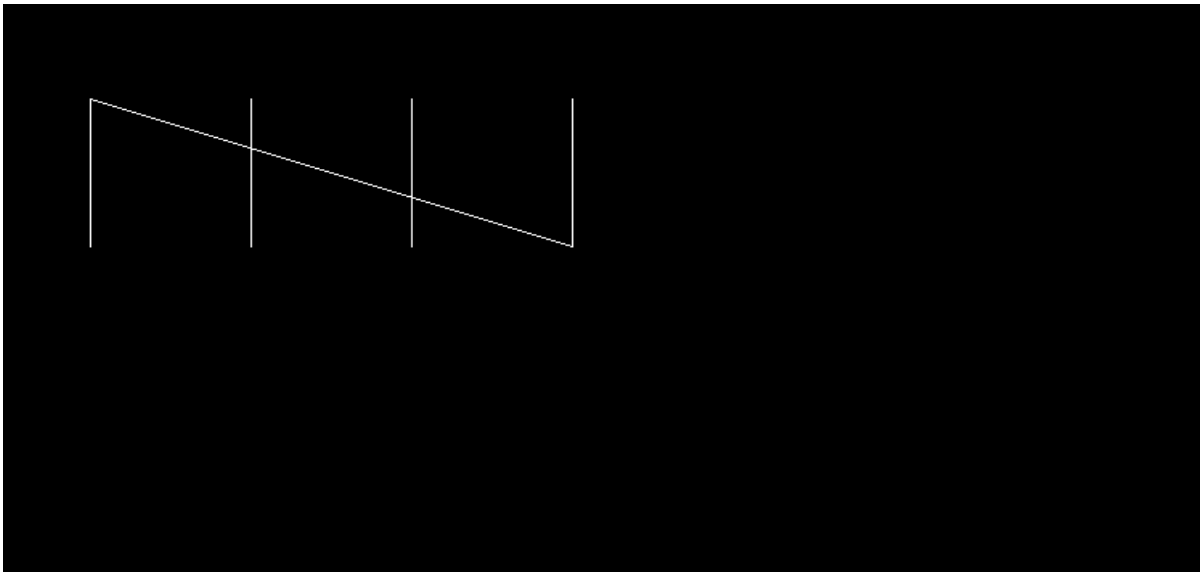

Рисунок № 13 — Содержание файла lines_positions.png.

Метод test_draw_first представляет собой вызов функции draw_first, которая, отвечает за отрисовку движения первой вершины дерева. Основной

целью данного теста является визуальное сравнение ожидаемого результата с фактическим результатом при выполнении функции.

В методе tearDown происходит завершение работы Pygame с помощью pygame.quit(), что освобождает ресурсы, используемые библиотекой, и завершает работу графического интерфейса.

Таким образом, на основе проведенного тестирования можно сделать вывод, что все основные функции отображения графических примитивов возвращают результат, полностью совпадающий с ожидаемым.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были проанализированы такие древовидные структуры, как AVL-деревья, красно-черные деревья, B-деревья, R-деревья и декартовы деревья. Приведены примеры применения данных древовидных структур в образовательных пособиях по структурам данных, а также в технологиях, используемых в индустрии информационных технологий.

После изучения предметной области древовидных структур был подобран следующий технологический стек: язык программирования Python, библиотека Pygame для реализации компьютерной графики.

В процессе выполнения курсовой работы были разработаны и реализованы алгоритмы визуализации:

- узлов, имеющих градиентную окраску;
- линий, соединяющих между собой вершины;
- заданного количества вершин;
- заданного количества линий;
- перемещения расположения вершин и линий.

Также было разработано программное обеспечение на языке Python, которое интегрирует вышеописанные алгоритмы, а также алгоритмы компьютерной графики, основанные на библиотеке Pygame, для визуализации как больших, так и малых вращений AVL-дерева, а также для демонстрации самобалансирующегося AVL-дерева после добавления и удаления узлов.

В рамках разработки было проведено модульное тестирование основного функционала программы, результаты которого подтвердили корректность работы программного обеспечения. Весь ожидаемый результат функций совпал с фактическими выходными данными, что свидетельствует о правильной реализации разработанных алгоритмов.

Разработанное в течение выполнения курсовой работы программное обеспечение может быть использовано в разных сферах, связанных с информационными технологиями напрямую или косвенно. Например, может использоваться в официальных образовательных учреждениях для обучения школьников или студентов колледжей основам структур данных и алгоритмов, а также в курсах по компьютерной графике и визуализации данных. Кроме того, может использоваться как интерактивное учебные пособие для онлайн-курсов повышения квалификации или обучения с нуля или онлайн-учебников,

благодаря которым в любое время обучающиеся могут визуализировать работу AVL-деревьев в реальном времени. Таким образом, визуализация AVL-деревьев может быть полезна в образовательных учреждениях, способствуя лучшему пониманию материалов обучающимися.

СПИСОК ЛИТЕРАТУРЫ

[1] Свидетельство о государственной регистрации программы для ЭВМ № 2021612876 Российская Федерация. Физический движок для имитации жидкости и газов: № 2020667054 : заявл. 21.12.2020: опубл. 26.02.2021 / В. В. Черноусенко, А. В. Иванова, И. С. Баранов, И. Ю. Квасов ; заявитель федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский ядерный университет «МИФИ».

[2] Свидетельство о государственной регистрации программы для ЭВМ № 2020667813 Российская Федерация. Программа для визуализации математических функций в декартовой системе координат с возможностью масштабирования: № 2020666025: заявл. 08.12.2020: опубл. 29.12.2020 / Е. А. Петрик, А. А. Шедльбауэр, К. К. Корнеев; заявитель Федеральное государственное бюджетное образовательное учреждение высшего образования «Юго-Западный государственный университет» (ЮЗГУ)

[3] Гребенкин, Д. А. Визуализация изображений с применением компьютерной графики для осуществления компьютерного моделирования в сфере информационных технологий / Д. А. Гребенкин // Моя профессиональная карьера. – 2020. – Т. 2, № 11. – С. 254–261.

[4] Документация Python [Электронный курс] – URL: https://docs.python.org/3/ (дата обращения: 6.10.2024)

[5] Документация Pygame [Электронный курс] – URL: https://www.pygame.org/docs/ (дата обращения: 6.10.2024)

[6] Документация OpenGL [Электронный курс] – URL:

https://www.opengl.org/ (дата обращения: 6.10.2024)