МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»**

**УТВЕРЖДАЮ**

**Институт** ИКН

**Кафедра** Инженерной кибернетики **Зав. Кафедрой**

**Направление** Прикладная математика « 25 » декабря 2024г.

**ЗАДАНИЕ**

## НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ БАКАЛАВРА

**Студенту группы** БПМ-21-3 Ибрагимову Петеру Ильгизовичу

(ф.и.о. полностью)

1. Тема работы: Система анализа производительности пространственных индексов в задаче геопоиска.
2. Цель работы: Разработка программного обеспечения для практического анализа популярных алгоритмов геопоиска и пространственных индексов, которые потенциально могут использоваться в высоконагруженных системах, а также математический (теоретический) анализ указанных методов и алгоритмов,
3. Исходные данные: Нет.
4. Основная литература, в том числе:
   1. Монография, учебники и т.п.: Mehta D.P., Sahni, S. Handbook of data structures and applications (1st ed.). - New York.: Chapman and Hall/CRC, 2004.– P. 1392.
5. Перечень основных этапов исследования и форма промежуточной отчетности по каждому этапу.
   1. Аналитический обзор литературы.
   2. Формулировка содержательной постановки задачи.
   3. Формулировка математической постановки задачи.
   4. Реализация программного обеспечения.
   5. Анализ результатов проведенных тестов.
   6. Подготовка текста выпускной квалификационной работы, доклада и презентации.
6. Аппаратура и методики, которые должны быть использованы в работе: математический анализ аксиоматической сложности алгоритмов по времени и затратам памяти, анализ результатов экспериментов (тестов) сложности алгоритмов.
7. Использование ЭВМ: Языки программирования: Golang (реализации алгоримтов, самой системы) и Python (визуализация и анализ результатов).
8. Перечень (примерный) основных вопросов, которые должны быть рассмотрены и проанализированы в литературном обзоре.
   1. Актуальность задачи,
   2. Обзор популярных алгоритмов и структур данных для работы с геоданными (далее алгоритмов),
   3. Обзор алгоритмов, используемых в высоконагруженных системах,
   4. Обзор ранее проведенных тестов и исследований приведенных алгоритмов,
   5. Выводы по аналитическому обзору литературы,
9. Перечень (примерный) графического и иллюстрированного материала.
   1. Актуальность поставленной задачи,
   2. Описание предметной области,
   3. Математическая постановка задачи,
   4. Содержательная постановка задачи,
   5. Краткие схемы работы каждого из описанных алгоритмов,
   6. Функциональная схема ПО,
   7. Полученные результаты и их анализ,
   8. Выводы,
10. Руководитель работы: cт. преп. Тагиев Э. Р

(Должность, звание, ф.и.о.)

(подпись)

Дата выдачи задания: 25 декабря 2024

**Задание принял к исполнению студент**

(подпись)

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#_bookmark0) 10

1. [АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ](#_bookmark0) 12
   1. [Расстояние между двумя точками](#_bookmark1) 13
      1. [Евклидово расстояние](#_bookmark2) 13
      2. [Расстояние гаверсинуса](#_bookmark3) 14
      3. [Геодезическое расстояние](#_bookmark4) 14
   2. [Пространственные индексы](#_bookmark5) 16
      1. [K-d tree](#_bookmark6) 16
      2. [KNN поиск по KD-tree](#_bookmark7) 17
      3. [Поиск в круге по KD-tree](#_bookmark8) 18
      4. [Quadtree](#_bookmark9) 19
      5. [R-tree](#_bookmark10) 20
      6. [R\*-tree](#_bookmark11) 21
      7. [Bruteforce](#_bookmark11) 22
      8. [VP-tree](#_bookmark12) 22
      9. [BSP-tree](#_bookmark13) 24
      10. [Иные индексы](#_bookmark14) 25
   3. [Иные структуры данных](#_bookmark14) 26
      1. [Geohash](#_bookmark15) 26
      2. [Uber H3](#_bookmark16) 27
      3. [S2 geometry](#_bookmark17) 29
   4. [Сравнительный анализ](#_bookmark18) 30
      1. [Качественный сравнительный анализ](#_bookmark19) 30
2. [СПЕЦИАЛЬНАЯ ЧАСТЬ](#_bookmark19) 32
   1. [Содержательная постановка задачи](#_bookmark20) 32
   2. [Математическая постановка](#_bookmark21) 34
   3. [Разработка](#_bookmark22) 35
      1. [Система тестирования](#_bookmark23) 35
      2. [Порядок работы с программным обеспечением](#_bookmark24) 36
   4. [Разработанные индексы и алгоритмы](#_bookmark25) 38
      1. [Geohash B-tree](#_bookmark26) 38
3. [ПРОВЕДЕННЫЕ ИССЛЕДОВАНИЯ. РЕЗУЛЬТАТЫ](#_bookmark27) 42
   1. [Результаты тестирования](#_bookmark28) 42
   2. [Выводы](#_bookmark29) 44
   3. [Выводы по практической части](#_bookmark30) 44

[ЗАКЛЮЧЕНИЕ](#_bookmark31) 45

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_bookmark32) 46

[ПРИЛОЖЕНИЕ А](#_bookmark53) 49

[Основные фрагменты кода](#_bookmark54) 49

[ПРИЛОЖЕНИЕ Б](#_bookmark55) 59

[Дополнительные результаты тестирований](#_bookmark56) 59

**РЕФЕРАТ**

ВКР 60 с., 24 рис., 4 ф-л, 1 табл., 20 источн., 2 прил.

ДРЕВОВИДНЫЕ СТРУКТУРЫ ДАННЫХ, АБСТРАКТНЫЕ СТРУК­ ТУРЫ ДАННЫХ, ИНДЕКСЫ, ПРОСТРАНСТВЕННЫЕ ИНДЕКСЫ, R-TREE, KD-TREE, GEOHASH, H3, B-TREE, ГЕОПОИСК

Объект исследования: пространственные индексы и алгоритмы поис­ ка по геоданным.

Цель работы: разработать программное обеспечение для практиче­ ского анализа популярных алгоритмов геопоиска и пространственных ин­ дексов, которые потенциально могут использоваться в высоконагруженных системах, а также математический (теоретический) анализ указанных ме­ тодов и алгоритмов. Доработка существующих и разработка новых алго­ ритмов под указанные задачи.

Методы проведения работы: дискретная математика, теория графов, анализ производительности индексов.

Результат работы: система нагрузочного тестирования индексов. Но­ вые индексы, оптимизация уже существующих индексов. Оценка качества индексов.

Область применения результатов: бекенд разработка, СУБД.

# ABSTRACT

FQW 60 p., 24 fig., 4 f-l, 1 tabl., 20 sources, 2 app.

TREE-LIKE DATA STRUCTURES, ABSTRACT DATA TYPES, INDEXES, SPACIAL INDEXES, R-TREE, KD-TREE, GEOHASH, H3, B-TREE, GEOSEARCH

Object of research: spacial indexes and search algorithms on geo data.

The purpose of the work: develop software for practical analysis of popular geo algorithms and spacial indexes, which could potentially be used in high loaded systems. Mathematical (theoretical) analysis of the reported methods and algorithms. Modification of the existing algorithms, development of new algorithms.

Methods of work: discrete mathematics, graph theory, performance analysis of indexes.

The result of the work: loadtesting system for indexes. New indexes, optimizations of existing indexes. Performance scoring of indexes.

Scope of the results: backend development, DBMS

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе применяют следу­ ющие термины с соответствующими определениями.

Индекс - абстрактная структура данных, создаваемая с

целью повышения операций поиска по заданно­ му массиву за счёт повышения затрат на хране­ ние. Примеры: в СУБД: b-tree, в Python: dict, в C++: map

Пространственный индекс

- индекс, построенный для ускорения поиска по геоданным, например, для решения задачи KNN или поиска в прямоугольнике. Примеры: R-Tree, KD-Tree и т.д.

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе применяют следу­ ющие сокращения и обозначения.

|  |  |  |
| --- | --- | --- |
| БД | - | база данных |
| СУБД | - | система управления базами данных |
| ЭВМ | - | электронная вычислительная машина |
| BBox | - | bounding box (задача поиска точек в прямоуголь­  нике) |
| KNN | - | k-nearest neighbors (K-ближайших соседей) |
| No-SQL | - | No Structured Query Language (Тип СУБД, кото­  рые не используют SQL) |
| RPS | - | Requests per second (Запросов за секунду) |
| SQL | - | Structured Query Language (Язык структурирован­  ных запросов) |

# ВВЕДЕНИЕ

В настоящее время геоданные являются неотъемлемой частью мно­ гих высоко нагруженных систем, таких как поисковые системы, социаль­ ные сети, картографические сервисы и другие. Однако обработка и хране­ ние большого объема геоданных может стать проблемой для разработчи­ ков. Для решения данной проблемы были разработаны методы геопоиска и пространственные индексы, которые позволяют эффективно работать с геоданными в высоко нагруженных системах.

Цель данной дипломной работы — провести сравнительный анализ применения алгоритмов геопоиска и пространственных индексов в высоко нагруженных системах. В работе будут рассмотрены основные принципы работы данных методов, их преимущества и недостатки. Также будет прове­ дено сравнение производительности данных методов на различных наборах геоданных.

Под высоконагруженной системой подразумевается система, в кото­ рой нет возможности бесконечно масштабировать систему вертикально и разработчикам приходится оптимизировать существующие алгоритмы и подходы. Часто, такая система имеет большое количество запросов как на чтение, так и на запись, а общее количество запросов к системе в секунду (RPS) превышает несколько тысяч.

Основной проблемой работы с геоданными в высоко нагруженных си­ стемах является неочевидность в выборе структур и методов хранения дан­ ных. Так, например, классический индекс B-Tree нельзя использовать при обращении к кортежам геоточек, потому что B-Tree умеет работать только со скалярными данными. При этом для работы с геоданным существует объемное количество индексов, например, R-Tree, KD-Tree, Quadtree, каж­ дый из которых имеет свои плюсы и минусы. Помимо этого существует проблема сериализации и десериализации данных, которая заключается в

том, что результаты анализа поиска, запросы на поиск и сами данных точек необходимо передавать по сети в как можно меньшем объеме и размере (в данном случае под объемом подразумевается количество передаваем дан­ ных, а размер — вес в байтах).

Результаты данной работы могут быть полезны для разработчиков высоко нагруженных систем, которые работают с геоданными, а также для специалистов в области геоинформатики.

## АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ

В данной работе будут анализироваться только 3 задачи поиска по геоданным. Задачи можно сформулировать следующим образом: Пусть есть множество точек, представляющих собой пару широта-долгота, отоб­ ражающих некий объект на поверхности земли.

Необходимо произвести операции:

а) BBox(bottomLeft Point, upperRight Point) Points - поиск всех объ­ ектов, которые входят в прямоугольник, определенный нижним левым углом (bottomLeft) и правым верхним углом (upperRight). Возвращает­ ся массив найденных точек;

б) KNN(p Point, k int) Points - поиск k ближайших объектов от точки p, возвращается массив найденных точек;

в) Insert(p Point) - вставка нового элемента (точки) в индекс. Важно отметить, что, зачастую, для прикладных задач также требу­

ется решить задачу поиска в кругу: Найти все точки X, находящееся на заданном расстояние R от заданной точки Y. Данная задача не рассмат­ ривается в данной работе из-за того, что на части индексов она не может быть решена оптимально. Например, на индексе R-tree. При этом её все равно можно решить через задачу BBox:

а) решить задачу BBox для квадрата, описывающего заданный круг; б) простым перебором отсечь точки, что не входят в заданный круг. Таким образом в данной работе данная задача не рассматривается.

При этом задача BBox может быть также сведена к задаче поиска в кругу: а) решить задачу поиска в кругу для круга, описывающего квадрат; б) простым перебором отсечь точки, что не входят в заданный пря­

моугольник.

## Расстояние между двумя точками

Указанные выше задачи требуют обусловить понятие расстояния между двумя точками на сфере земли. Для этого введем понятие геоточки: это точка, представляющая собой кортеж A(x, y), где x и y - широта и дол­ гота соответственно. Расстоянием между двумя точками есть наименьшая прямая на плоскости земного шара, при этом ее вычисление может быть разным в зависимости от положенной задачи.

Поставим задачу:

Даны геоточки 𝐴(𝑥1, 𝑦1), 𝐵(𝑥2, 𝑦2). Требуется найти расстояние

𝑑(𝐴, 𝐵)

## Евклидово расстояние

В некоторых системах, например, расширение Postgis для СУБД Postgres для типа Geometry, используется евклидово расстояние:

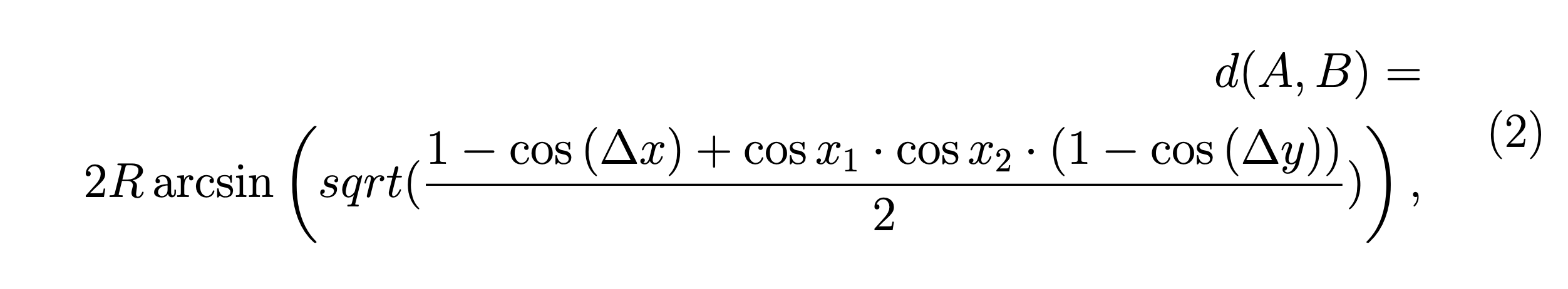
𝑑(𝐴, 𝐵) = 𝑠𝑞𝑟𝑡((𝑥1 *−* 𝑥2)2 + (𝑦1 *−* 𝑦2)2). (1)

Данный подход работает корректно только в части случаев, так, например, он выдаст правильный результат с точностью до 5 метров в случае поиска расстояния в пределах Москвы, но, расстояние в Африке будет отличаться в 2-3 раза относительно более строгих подходов. При этом важно отметить, что данная дистанция отвечает аксиомам метрики, поэтому ее корректно использовать при сравнении расстояний между двумя точками, например, при решении задаче KNN(К-ближайших соседей).

## Расстояние гаверсинуса

Расстояние гаверсинуса — это способ определения расстояния меж­ ду двумя точками на поверхности Земли, учитывающий кривизну Земли. Оно используется в геопоиске для определения расстояния между заданной точкой и объектами в заданном радиусе.

Формула расстояния гаверсинуса выглядит следующим образом



где d - расстояние между двумя точками в километрах, R - радиус Земли (приблизительно 6371 км), 𝑥1 и 𝑥2 - широты двух точек в радианах,

𝑦1 и 𝑦2 - долготы двух точек в радианах.

Эта формула позволяет определить расстояние между точками с точ­ ностью до нескольких метров, что делает ее очень полезной для геопоиска. Однако она может быть достаточно ресурсоёмкой при работе с большими объемами геоданных, что делает ее медленнее по сравнению с Евклидовым расстоянием.

## Геодезическое расстояние

Геодезическое расстояние — это расстояние между двумя точками на поверхности Земли, измеренное вдоль кратчайшей линии (геодезической линии) между этими точками. Геодезическая линия — это кривая на по­ верхности Земли, которая имеет наименьшую длину между двумя точка­ ми.

Геодезическое расстояние учитывает кривизну Земли и может отли­ чаться от расстояния гаверсинуса, особенно на больших расстояниях и при использовании разных моделей формы Земли.

Для расчета геодезического расстояния используются различные ме­ тоды, такие как метод Винсента, метод Гаусса-Крюгера и метод Хаверсина. Эти методы учитывают форму Земли и позволяют получить более точные результаты, чем простые формулы для расчета расстояния на плоскости.

Геодезическое расстояние широко используется в геопозиционирова­ нии, навигации, картографии и других областях, где требуется точное опре­ деление расстояния между двумя точками на поверхности Земли.

Формула вычисления геодезии

𝑑(𝐴, 𝐵) = 𝑅 *·* arccos(sin(𝑥1) *·* sin(𝑥1) + cos(𝑥1) *·* cos(𝑥2) *·* cos(𝑦2 *−* 𝑦1). (3)

Таким образом, при выборе формулы поиска расстояния надо учитывать требования.

Например, при разработке приложения кикшеринга, то есть аренды электронных самокатов, их можно комбинировать. На запросах, которые возвращают клиентам ближайшие самокаты от их текущей геопозиции по радиусу можно использовать метрику Евклида, так как ошибку даже на 20-30 метров можно нивелировать поиском по большему радиусу. При этом, в ситуации, когда требуется построить аналитику данных, лучше исполь­ зовать расстояние геодезии, потому что в данных задачах крайне важна точность результата, а разница в скорости решение не так важна.

## Пространственные индексы

Как и с задачами поиска по *плоскому* массиву скаляров, для поис­ ка по геоточкам используются специальные структуры данных, которые позволяют оптимизировать операции поиска.

Указанные структуры можно разбить на 2 типа: древовидные и хэши со скалярным индексом. К первому виду относятся: R-tree, VP-tree, BSP- tree, Quadtree, KD-tree. К хэшам относятся: Geohash, S2 Geometry и Uber H3.

Рассмотрим принцип работы древовидных структур на примере KD

tree, самого *легкого* для понимания человеком пространственного индекса.

## K-d tree

K-d tree представляет собой дерево, позволяющее производить операции поиска в N-мерном пространстве. Рассмотрим двухмерное K-d tree, которое также можно назвать K-2 tree. Сама структура K-d tree является обычным бинарным деревом, как показано на рисурнке 1.

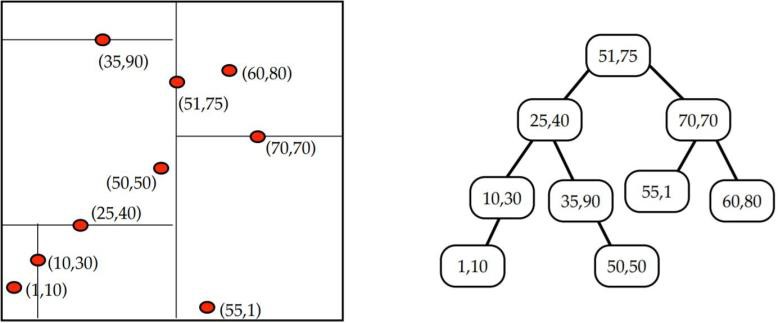


Рисунок 1 — Слева — пример визуализации разбиения плоскости через

KD-tree. Справа - результат построения дерева

Алгоритм построения K-d tree довольно [прост[1]:](#_bookmark33)

а) происходит поиск центральной точки, то есть той точки, которая будет находиться на суммарно меньшем расстоянии от всех точек. Дан­ ная точка ставиться в корень дерева k-d tree;

б) далее плоскость *разбивается* на 2 части по вертикальной оси;

в) *слева* ищется *средняя точка*, то есть та точка, которая по оси абс­ цисс (широте) находится на суммарно меньшем расстоянии до осталь­ ных. Данная точка записывает в левого ребенка корня дерева;

г) аналогичная процедура повторяется справа;

д) аналогичная процедура повторяется для вновь созданных поло­ тен, но уже с осью ординат (долготой);

е) данные процедуры повторяются со всеми точками.

## KNN поиск по KD-tree

Процесс поиска K ближайших соседей по KD-tree заключается в сле­ дующих шагах:

а) строим KD-tree из набора точек;

б) находим ближайшую к заданной точке точку в дереве. Для этого спускаемся по дереву, сравнивая координаты заданной точки и текущей точки в узле дерева. Если координата текущей точки больше или рав­ на координате заданной точки, то спускаемся в левое поддерево, иначе

- в правое. При этом сохраняем расстояние между текущей точкой и заданной точкой;

в) добавляем найденную точку в список ближайших соседей;

г) проверяем, есть ли еще точки в дереве, которые могут быть бли­ же к заданной точке, чем уже найденные соседи. Для этого проверяем расстояние между заданной точкой и границей текущего поддерева (это можно сделать, используя формулу расстояния между точками). Если

это расстояние меньше, чем расстояние до самого дальнего найденного соседа, то нужно проверить и другое поддерево;

д) повторяем шаги 2-4 для всех точек в дереве, пока не найдем K

ближайших соседей или не пройдем по всему дереву.

В результате получаем список K ближайших соседей заданной точки. Этот алгоритм позволяет быстро находить ближайшие соседи в больших наборах данных и широко используется в геоинформационных системах для поиска ближайших объектов на карте.

## Поиск в круге по KD-tree

Процесс поиска K ближайших точек и всех точек в заданном радиусе очень поход на процесс поиска по бинарному дереву за тем исключением, что при сравнении по четным нодам идет по широте, а по нечетным — по долготе.

Процесс заключается в следующих шагах:

а) строим KD-дерево на основе набора геоданных;

б) ищем листовой узел дерева, который содержит заданную точку. Для этого начинаем с корневого узла и спускаемся по дереву, выбирая каждый раз ту часть пространства, которая содержит заданную точку; в) находим все точки, которые находятся в заданном круге с центром

в заданной точке и радиусом R. Для этого проверяем каждую точку в листовом узле и всех его родительских узлах на расстояние до задан­ ной точки. Если расстояние меньше или равно R, то добавляем точку в список найденных точек;

г) если листовой узел не содержит достаточного количества точек, то расширяем круг до тех пор, пока не найдем достаточное количество точек;

д) возвращаем список точек, которые находятся в заданном круге.

## Quadtree

Quadtree - это пространственный индекс, который представляет пространство в виде дерева, где каждый узел представляет собой четыре дочерних узла. Этот индекс позволяет быстро искать точки в заданном круге, разбивая пространство на более мелкие части и проверяя только те части, которые находятся внутри круга. Представление дерева на плоскости и в памяти ЭВМ показано на рисунке 2.

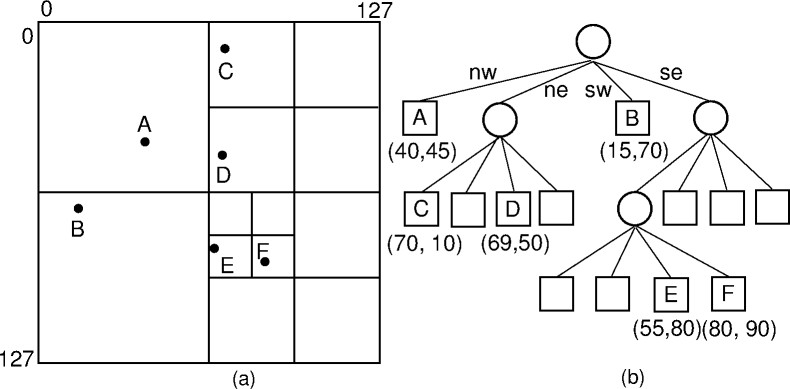


Рисунок 2 — Слева - пример визуализации разбиения плоскости через

Quadtree. Справа — результат построения дерева

Процесс поиска точек в заданном круге по Quadtree заключается в следующих шагах:

а) конвертируем заданные координаты в координаты узла Quadtree. Это делается путем разбиения пространства на сетку и назначения каж­ дой ячейке уникальных координат узла Quadtree;

б) определяем узлы Quadtree, которые находятся внутри заданного круга. Для этого используем формулу Хаверсина для вычисления рас­ стояния между заданными координатами и каждым узлом Quadtree. Если расстояние меньше или равно радиусу круга, то добавляем этот узел в список;

в) ищем все точки, которые соответствуют найденным узлам Quadtree. Для этого используем индекс, который связывает каждый узел Quadtree с набором геоданных. Ищем все точки в наборе, которые соответствуют найденным узлам Quadtree;

г) возвращаем список точек, которые находятся в заданном круге.

## R-tree

R-tree - это пространственный индекс, который используется для хранения и поиска объектов в [пространстве[2].](#_bookmark34) Как показано на рисунке 3, R-tree представляет собой дерево, где каждый узел представляет собой прямоугольник, содержащий объекты. Этот индекс позволяет быстро находить объекты, которые находятся в заданном прямоугольнике или близко к нему.

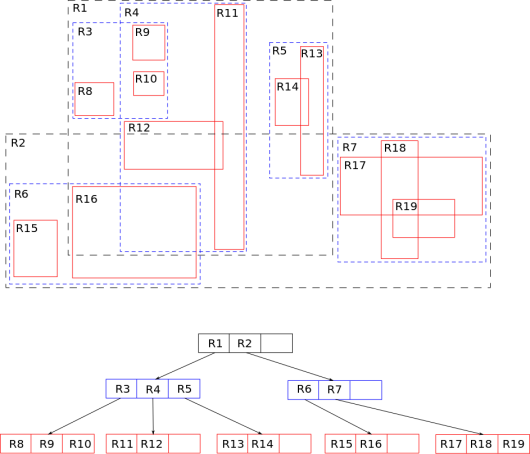


Рисунок 3 — Сверху — пример визуализации разбиения плоскости через

R-tree. Снизу - результат построения дерева

Процесс поиска объектов в заданном прямоугольнике по R-tree за­ ключается в следующих шагах:

а) конвертируем заданный прямоугольник в формат R-tree. Это де­ лается путем определения минимального и максимального значения ко­ ординат для каждого измерения;

б) ищем узлы R-tree, которые пересекаются с заданным прямоуголь­ ником. Для этого используем алгоритм пересечения прямоугольников, который позволяет быстро определить, какие узлы R-tree находятся внутри или пересекаются с заданным прямоугольником;

в) ищем все объекты, которые соответствуют найденным узлам R-tree. Для этого используем индекс, который связывает каждый узел R-tree с набором объектов. Ищем все объекты в наборе, которые соот­ ветствуют найденным узлам R-tree;

г) возвращаем список объектов, которые находятся в заданном пря­ моугольнике.

Важно отметить, что данная структуру используется в большом ко­ личестве популярных СУБД:

а) PostGIS - Расширение для СУБД PostgreSQL б) Redis - Key-value база данных

в) Tile38 - Аналог Redis с углублением в работу с геоданными Таким образом данный индекс выигрывает в популярности у разработчи­ ков.

## R\*-tree

R\*-tree представляет собой оптимизацию балансировки [R-tree[3],](#_bookmark35) бла­ годаря которой операции поиска становятся быстрее, но при этом операция вставки замедляется за счет необходимости перестройки дерева. Необходи­ мо подчеркнуть, что в указанных выше СУБД используется именно *ори­ гинальная* версия дерева - [R-tree[4].](#_bookmark36) Данный факт обязует на практике

протестировать разницу между деревьями и проанализировать причины такого [выбора[5].](#_bookmark37)

## Bruteforce

Для наиболее честного анализа качества работы индексов, требует­ ся использовать контрольный индекс, который можно именовать как *пере­ бор* (Bruteforce). Данный индекс представляет собой простой динамический массив, в который сохраняются все заданные точки. Данный массив можно использовать как самое *плохое* решение *в лоб*, иными словами если какой­ то алгоритм или индекс работает хуже, чем *перебор* - значит данный индекс либо разработан неправильно, либо имеет слишком критические недостат­ ки.

Процесс поиска ближайших соседей:

а) вычислить матрицу расстояний от заданной точки до каждой точ­ ки массива;

б) осортировать массив по расстояниям используя указанную матри­ цу;

в) взять первые K элементов. Процесс поиска в прямоугольнике:

а) последовательно итерироваться по всем элементам массива и до­ бавлять в результирующий массив все точки, чьи широты и долготы входят в прямоугольник.

## VP-tree

VP-tree (Vantage Point tree) - это пространственный индекс, который используется для быстрого поиска ближайших соседей в многомерном пространстве. VP-tree представляет собой бинарное дерево, где каждый узел представляет собой точку-центр (вантажный пункт, как показано на

рисунке 4), от которой строятся два поддерева: левое поддерево содержит все точки, которые находятся внутри заданного радиуса от центра, а правое поддерево содержит все точки, которые находятся за пределами этого радиуса.

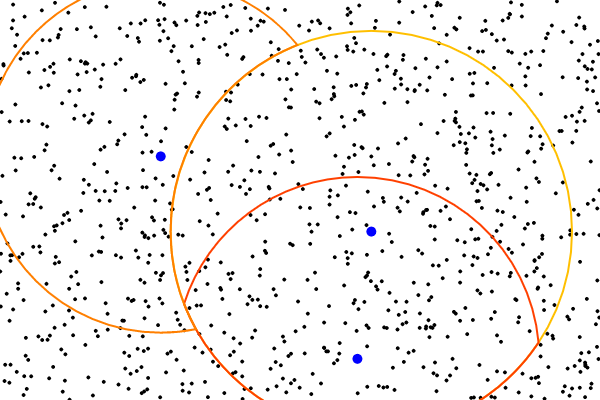


Рисунок 4 — Разбиения пространства на 3 подпространства через VP-Tree

Процесс поиска ближайших соседей по VP-tree заключается в следу­ ющих шагах:

а) строим VP-tree из набора точек. Для этого выбираем случайную точку-центр и разбиваем набор точек на две группы: которые находятся ближе к центру, и которые находятся дальше от него;

б) ищем ближайшую точку к заданной точке. Для этого начинаем с корня VP-tree и рекурсивно спускаемся по дереву, выбирая поддере­ во, которое содержит более близкие к заданной точке точки. Если рас­ стояние от центра поддерева до заданной точки меньше, чем текущее расстояние до ближайшей точки, то ищем ближайшую точку в этом поддереве;

в) ищем k ближайших соседей к заданной точке. Для этого начинаем с корня VP-tree и рекурсивно спускаемся по дереву, выбирая поддере­ во, которое содержит более близкие к заданной точке точки. Если рас­ стояние от центра поддерева до заданной точки больше, чем текущее расстояние до k-го ближайшего соседа, то не ищем в этом поддереве. Иначе ищем в этом поддереве и добавляем найденные точки в список ближайших соседей. После этого проверяем, есть ли еще поддеревья, которые могут содержать более близкие точки, и продолжаем поиск.

## BSP-tree

BSP-tree (Binary Space Partitioning tree) - это пространственный ин­ декс, который используется для разбиения пространства на бинарные подпро[странства[6].](#_bookmark38) BSP-tree представляет собой бинарное дерево, где каж­ дый узел представляет собой гиперплоскость, которая разделяет простран­ ство на две части: левую и правую.

Процесс поиска ближайших соседей по BSP-tree заключается в сле­ дующих шагах:

а) cтроим BSP-tree из набора точек. Для этого выбираем случайную гиперплоскость и разбиваем набор точек на две группы: те, которые находятся по одну сторону от гиперплоскости, и те, которые находятся по другую сторону;

б) ищем ближайшую точку к заданной точке. Для этого начинаем с корня BSP-tree и рекурсивно спускаемся по дереву, выбирая подде­ рево, которое содержит более близкие к заданной точке точки. Если расстояние от гиперплоскости до заданной точки меньше, чем текущее расстояние до ближайшей точки, то ищем ближайшую точку в этом поддереве;

в) ищем k ближайших соседей к заданной точке. Для этого начина­ ем с корня BSP-tree и рекурсивно спускаемся по дереву, выбирая под­ дерево, которое содержит более близкие к заданной точке точки. Если расстояние от гиперплоскости до заданной точки больше, чем текущее расстояние до k-го ближайшего соседа, то не ищем в этом поддереве. Иначе ищем в этом поддереве и добавляем найденные точки в список ближайших соседей. После этого проверяем, есть ли еще поддеревья, которые могут содержать более близкие точки, и продолжаем поиск.

## Иные индексы

В научном сообществе каждый год публикуются статьи с новыми ин­ дексами, например, MPTrie [[7],](#_bookmark39) S[THash[8]](#_bookmark40) и другие. Большинство из них решают довольно конкретные задачи, либо улучшают уже созданные ин­ дексы под определенные наборы данных. В данной работе указанные ин­ дексы не рассматриваются из-за того, что их некорректно было бы срав­ нивать с R-Tree, KD-Tree и тд. из-за того, что, как уже было сказано, они реализованы под конкретную задачу.

## Иные структуры данных

## Geohash

Geohash (далее также геохеш) - представляет собой бинарной представление координат (широта-долгот[а)[9].](#_bookmark41) Сам геохеш не реализует алгоритмов поиска, поэтому для поиска по геохеше дополнительно ис­ пользуются такие структуры как B-tree, trie, Radix-trie и т[д[10].](#_bookmark42) Самим геохешом называется строка, закодированная 32 разрядным алфавитом, перевод из 10-ой системы в указанный алфавит указан в таблице 1.

Таблица 1 — алфавит геохеша по основанию 10 и 32

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Основание | 10 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Основание | 32 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Основание | 10 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Основание | 32 | 8 | 9 | b | c | d | e | f | g |
| Основание | 10 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Основание | 32 | h | j | k | m | n | p | q | r |
| Основание | 10 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Основание | 32 | s | t | u | v | w | x | y | z |

Данная строка однозначно декодируется в кортеж геокоординат с точно­ стью, зависящей от количества символов в строке. Примеры:

а) строка ucft дегодируется в прямоугольник площадью примерно

800 𝑘𝑚2 и центром в (55.81, 37.44);

б) строка ucft943 имеет тот же центр - (55.8236, 37.3116), но мень­ шую площадь 23409𝑚2

Как можно наблюдать, чем выше количество символов, используемых в геохеше, чем выше точность получаемых координат, но при этом выше

затрачиваемая [память[11].](#_bookmark43) В данной работе не будет детально описываться процесс формирования геохеша за исключением базового принципа:

Сфера земли разбивается на практически равные прямоугольники, после чего каждому прямоугольнику присваивается номер в 32х-ричной си­ стеме координат, номера присваиваются в порядке *змейкой*, сначала самый левый-верхний, далее ниже от него, далее справа от самого левого-верхнего и тд, пример разбиения виден на рисунке 5.

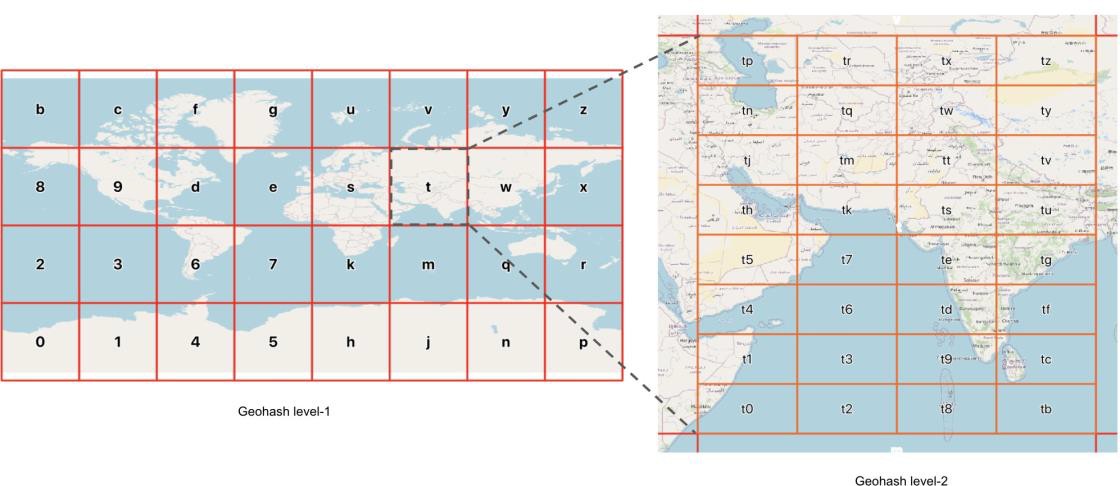


Рисунок 5 — Пример разбиения сферы земли через Geohash на первые 2 префикса

Сам по себе Geohash, как и ниже описанные структуры Uber H3 и S2 - не являются индек[сами[12],](#_bookmark44) а лишь методами сериализации геоточек. Реали­ зации алгоритмов поиска на этих методах производится в связке с такими структурами как B-tree, Trie и тд.

## Uber H3

Uber H3 - это сетка гексагональных ячеек (см. рисунок 6), которая используется сериализации и хранения геоданных в приложениях компа­ нии Uber, которая, собственно, и разработала данную систему. Каждая

ячейка имеет уникальный идентификатор и может быть использована для определения местоположения объектов. Логика работы Uber H3 аналогич­ на логики Geohash за основным исключением, что в Uber H3 используются шестиугольники, а в Geohash - прямоугольники.

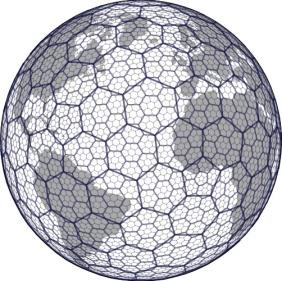


Рисунок 6 — Разбиение сферы земли на шестиугольники при использовании H3

Принципиальных отличий между H3 и [Geohash[13],](#_bookmark45) помимо, конечно, раз­ ницы в форме - нет. И тот и тот формат в конечном итоге кодирует данные в бинарном виде, имеется возможность перевести данные в строковое пред­ ставление.

Важно отметить, что между алгоритмами имееются чисто практиче­ ские отличия:

а) geohash более прост в реализации;

б) geohash более просто в понимание его человеком;

в) geohash поддерживается такими СУБД, как: Postgis, Redis, [MongoDB[14];](#_bookmark46)

г) H3 имеет очень обширную библиотеку с дополнительными мето­ дами;

д) H3 поддерживается такими СУБД, как: Clickhouse

## S2 geometry

S2 Geometry - это библиотека для работы с геометрическими объек­ тами на сфере, разработанная компанией Google.

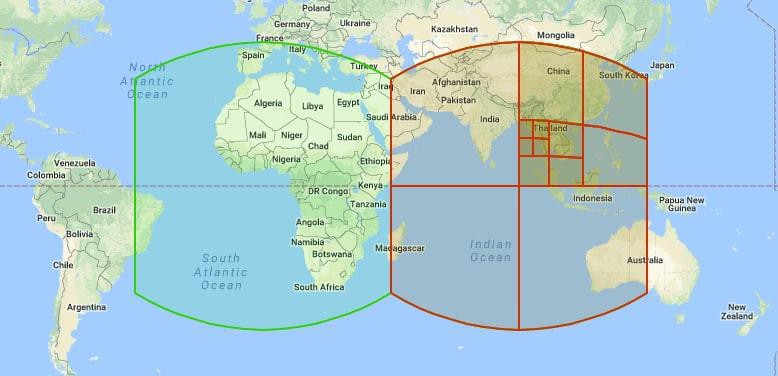


Рисунок 7 — Пример разбиения пространства на области с использованием S2 Geometry

Основой S2 Geometry является иерархическая структура данных, называ­ емая S2 Cell. Как показано на рисунке 7, каждая ячейка S2 Cell представ­ ляет собой квадрат на сфере, который может быть разбит на более мелкие квадраты более высокого уровня. Уровень ячейки определяется числом n, которое указывает на количество разбиений квадрата на подквадраты. Чем больше значение n, тем меньше размер каждой ячейки.

S2 не имеет значимых качественных или количественных преиму­ ществ перед Geohash и H3. Также, данный алгоритм имеет довольно скуд­ ное представление в СУБД, а также в подключаемых библиотеках в различ­ ных языках программирования. Из-за перечисленных выше причин дан­ ный алгоритм не будет далее рассматриваться в этой работе.

## Сравнительный анализ

Чтобы дать корректный ответ на вопрос о том, какую структуру дан­ ных стоит использовать, и какие алгоритмы к ней применять стоит доба­ вить следующие ограничения:

а) не будет учитываться время записи на диск. Предполагается, что все операции производятся при использовании оперативной памяти. Данное ограничение обусловлено тем, что операции сериализации и де­ сиарилизации индексов на постоянное запоминающие устройство в об­ щем случае идентичны для разных индексов;

б) требуется учитывать как теоретические показатели, то есть

O-нотацию, так и практические результаты.

## Качественный сравнительный анализ

Методы геопоиска и пространственных индексов могут использовать­ ся в различных сценариях, включая навигационные приложения, сервисы доставки еды и товаров, системы мониторинга транспорта и т[.д.[15]](#_bookmark47)

В навигационных приложениях методы геопоиска могут использо­ ваться для определения местоположения пользователя и построения марш­ рута до заданной точки. В этом случае наиболее эффективным методом может быть использование алгоритмов S2 geometry или Geohash, которые позволяют быстро находить ближайшие объекты на карте.

В сервисах доставки еды и товаров методы геопоиска могут использо­ ваться для определения ближайшего ресторана или магазина, а простран­ ственные индексы - для поиска объектов в заданном радиусе от точки доставки. В этом случае наиболее эффективными могут быть R-tree или Quadtree, которые позволяют быстро искать объекты в заданном радиусе.

В системах мониторинга транспорта методы геопоиска и простран­ ственные индексы могут использоваться для отслеживания местоположе­ ния транспортных средств и определения ближайшего транспорта для пользователя. В этом случае наиболее эффективным методом может быть использование R-tree, который позволяет быстро искать объекты в задан­ ном радиусе, а также определять их геометрические свойства (например, форму и размер).

Таким образом, выбор методов геопоиска и пространственные индек­ сы зависит от конкретной задачи и сценария использования. Необходимо учитывать требования к производительности, точности и эффективности поиска объектов на карте.

## СПЕЦИАЛЬНАЯ ЧАСТЬ

## Содержательная постановка задачи

Целью данной работы является разработка программного обеспече­ ния для анализа пространственных индексов в высоконагруженных систе­ мах. Данное ПО должно уметь анализировать как минимум предложен­ ные пространственные индексы: R-tree, R\*-tree, KD-tree, Geohash+btree, H3+btree и другие.

Пусть G - это множество всех точек в индексе, и каждая точка обозначает­

ся символом 𝐺𝑖, где 𝑖 *∈* N, широта и долгота каждой точки обозначается как 𝑋𝑖 и 𝑌𝑖, где 𝑋𝑖 *∈* R и 𝑌𝑖 *∈* R

Анализ должен проводится в разрезе трех операций: а) Insert(p Point) - вставка нового точки p в индекс;

б) KNN(p Point, k int) Points - поиск множества точек S размера

k, находящихся на наименьшем расстояние от точки p. 𝑆 = *{*𝐺𝑖*|*𝐺𝑖 *∈*

𝐺, 𝑑𝑖𝑠𝑡(𝑝, 𝐺𝑖) ⩽ 𝑑𝑖𝑠𝑡(𝑝, 𝐺𝑗), *∀*𝐺𝑗 *∈* 𝐺, 𝐺𝑗 *∈*/ 𝑆*}*, *|*𝑆*|* = 𝑘;

в) BBox(bottomLeft Point, upperRight Point) Points - поиск множе­ ства точек S, которых входят в прямоугольник, определяющийся ле­ вым нижним углом (bottomLeft) и правым верхнем углом (upperRight).

𝑆 = *{*𝐺𝑖*|*𝐺𝑖 *∈* 𝐺, 𝑋𝑖 ⩾ 𝑋𝑏𝑜𝑡𝑡𝑜𝑚𝐿𝑒𝑓 𝑡, 𝑌𝑖 ⩾ 𝑌𝑏𝑜𝑡𝑡𝑜𝑚𝐿𝑒𝑓 𝑡, 𝑋𝑖 ⩽ 𝑋𝑢𝑝𝑝𝑒𝑟𝑅𝑖𝑔ℎ𝑡, 𝑌𝑖 ⩽

𝑌𝑢𝑝𝑝𝑒𝑟𝑅𝑖𝑔ℎ𝑡)*}*.

Где 𝑑𝑖𝑠𝑡 - функция расстояния между двумя точками. В данном работе применяется расстояние гаверсинуса.

Важно отметить, что операция удаления точки не рассматривает­ ся из-за того, что в основном она не сильно отличается от операции вст[авки[16],](#_bookmark48) а также из-за того, что в высоко нагруженных системах, в основном, применяется подход soft-detele(от англ. мягкое удаление), в ко­

тором данные не удаляются, а лишь помечаются в СУБД как неактивные. Также, при проведении анализа данных, в основном, работа происходит с read-only данными, то есть теми данными, которые не меняются.

Для решения поставленных целей необходимо решить следующие за­

дачи:

а) Изучить существующие пространственные индексы

б) Разработать программное обеспечение для тестирования простран­

ственных индексов.

в) Реализовать в программном коде уже существующие индексы

г) Доработать существующие индексы и разработать новые индексы. д) Протестировать и отладить разработанное программное обеспече­

ние.

е) Проанализировать результаты работы программного обеспечения.

Найти наиболее подходящие структуры под анализируемые задачи.

## Математическая постановка

В работе будут проверять 3 основные операции над пространственны­ ми индексами, которые были подробно описаны в главе «Содержательная постановка задачи»: Insert, KNN, BBox.

Дано:

а) 𝑛 *∈* N, 𝑛 - количество элементов в индексе;

б) 𝑓 (𝑛) = 𝑥, 𝑥 *∈* Q, 𝑥 > 0, где 𝑓 (𝑛) - время выполнения операции в микросекундах.

Для каждой выше указанной операции и для каждой тестируемой струк­ туры требуется вычислить аксиоматическую сложность алгоритма:

𝑓𝑎𝑥(𝑛) = 𝑂(𝑓 (𝑛)). (4)

Реализуемое программное обеспечение должно уметь находить 𝑓 (𝑛) для заданных n и строить график зависимости 𝑓 (𝑛) от n. Разработанные и доработанные индексы должны корректно решать поставленые задачи.

## Разработка

## Система тестирования

Конечный продукт должен представлять собой сервис, который поз­ воляет протестировать реализацию пространственных структур с соответ­ ствующими операциями под нагрузкой, а также сами алгоритмы, как взя­ тые готовые решения, так и решения, разработанные в ходе данной работы: а) для реализации структур и алгоритмов был выбран язык програм­

мирования Golang;

б) для клиентского взаимодействия с сервисом используется HTML и JavaScript;

в) для визуализации результатов тестирований используется библио­ тека ChartsJS;

г) для визуализации работы алгоритмов используется библиотека

Mapbox.

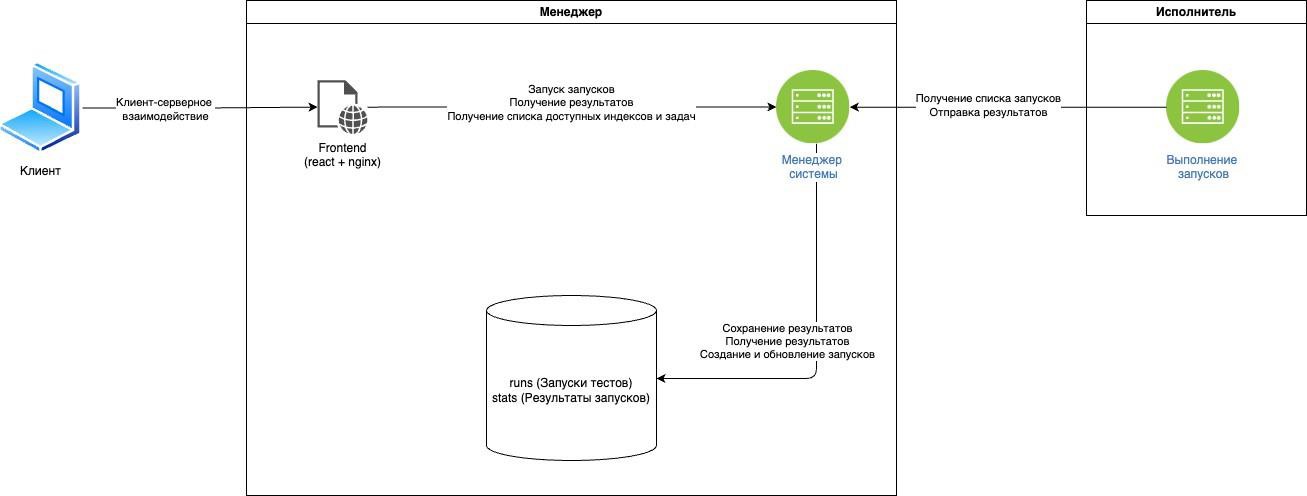


Рисунок 8 — Архитектура системы тестирования

На рисунке 8 визуализирована архитектура решения. Компонент менеджер отвечает за управление и запуск тестов. Компонент *Исполнитель* находит­ ся на отдельном от компонента *менеджер* сервере и отвечает за запуск конкретных задач. Сделано это для того, чтобы как можно сильнее изоли­

ровать среду запуска и среду анализа и не влиять на результаты запуска тестов какими-либо другими задачами.

## Порядок работы с программным обеспечением



Рисунок 9 — Функциональная схема

Процесс использования системы показан на рисунке 9: При откры­ тии страницы с тестами (рисунок 10), пользователь может ознакомиться с доступными индексами и задачами, а также задать количество точек для тестирования. После нажатия кнопки «Старт» начнется тестирование, ко­ торое сразу создаст запись с тестом и результатами в виде графиков. Поль­ зователь по своему усмотрению может остановить тестирование. Например, если оно выполняется слишком долго. Также он может открыть график с результатами в полном экране, а также скачать его. Также имеется стра­ ница «Визуализация данных» (рисунок 11), она может быть полезна для разработчиков во время разработки самих индексов, а именно для отладки индексов, визуализации их работы и сравнения с другими индексами.

С подробными результатами тестов можно ознакомится в приложение

Б.

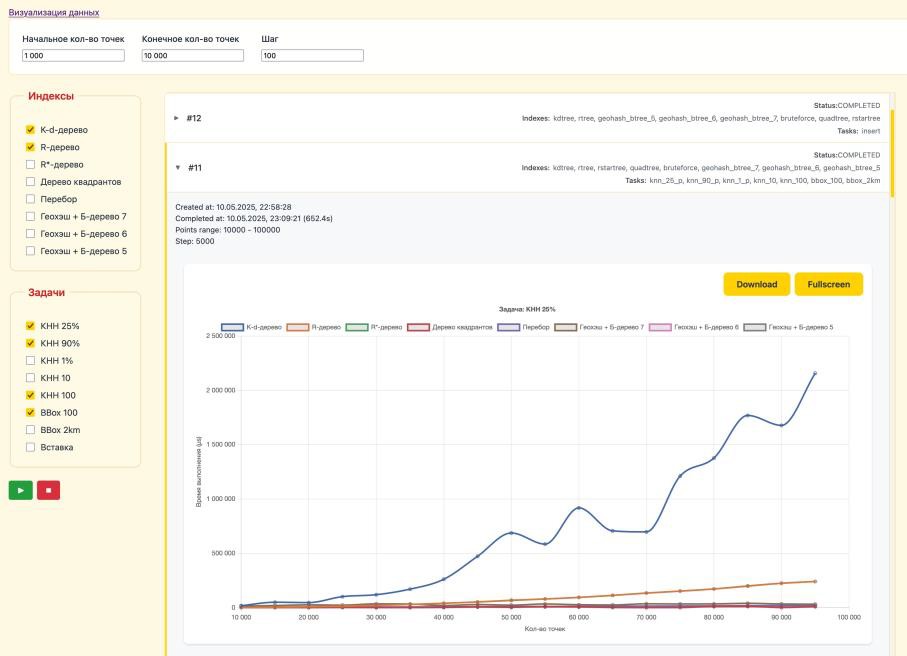


Рисунок 10 — Главный экран

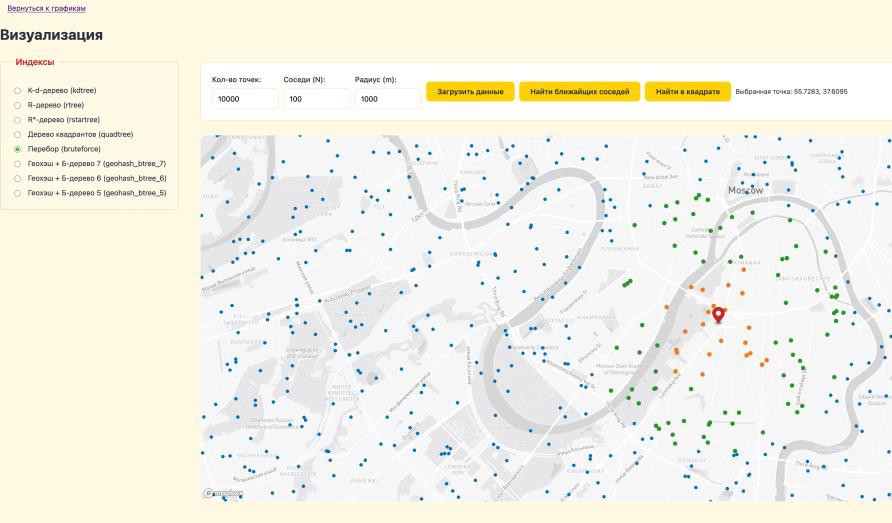


Рисунок 11 — Экран визуализации

## Разработанные индексы и алгоритмы

## Geohash B-tree

Во время разработки системы тестирования (рисунок 12), было заме­ чено, что алгоритм *перебора* достаточно хорошо работает на задаче «КНН 25%». Данная задача требует от алгоритма вернуть K ближайших точек, где 𝐾 = 𝑁4, где N - общее кол-во точек массива.

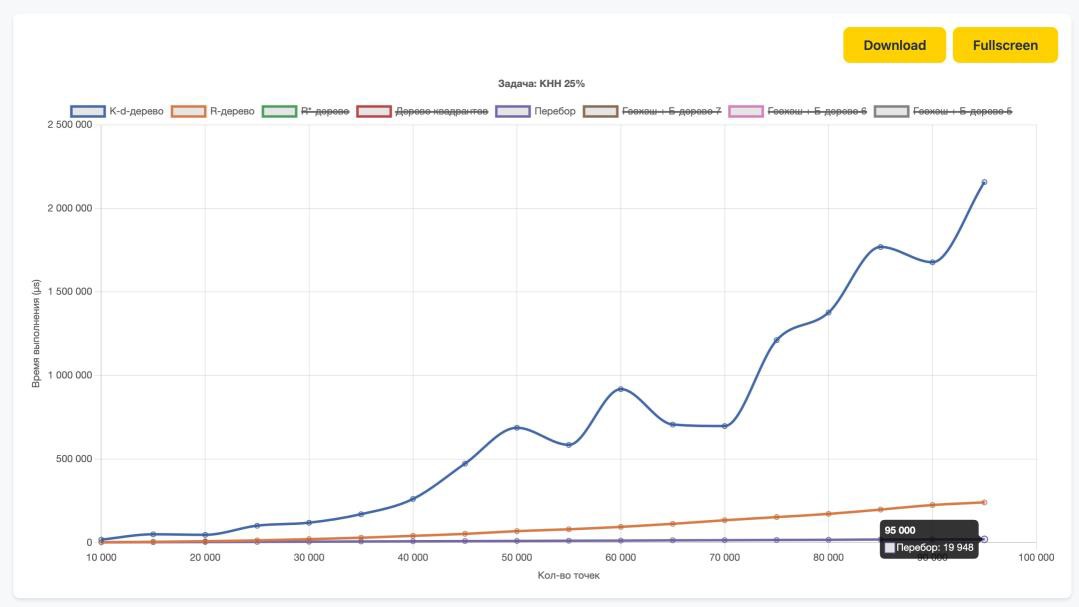


Рисунок 12 — Результаты тестирования

Иными словами, алгоритм простого перебора показывает хорошие резуль­

таты, когда K близко или сравнимо с N (𝐾 *̸≪* 𝑁), что было взято за основу алгоритма «Geohash B-tree».

Устройство разработанного «Geohash B-tree» просто, индекс пред­ ставляет собой B-T[ree[17],](#_bookmark49) ключами которого являются геохеши в типе данных uint64, а значениями - массив точек, которые находятся в указан­ ном хеше. Размерность является параметром индекса и фиксируется. Для реализации алгоритмов поиска применяется возможность геохеша класте­ ризовать пространство земли и находить кластеры, в которых уже далее *перебором* производятся операции поиска по KNN или BBo[x[18].](#_bookmark50)

Итерации поиска в прямоугольнике (рисунок 13):

а) получить геохеши углов, в примере на рисунке 13 - *5* и *y* (если опускать общий префикс);

б) получить все точки, что находятся на гранях прямоугольника (*5*,

*h*, *j*, *n* и тд.);

в) данные точки требуется также проверить на вхождение в прямо­ угольник через простой перебор;

г) все точки, что входят во внутрь (*k*, *s*, *m*, *t*) - проверять не надо, они гарантированно входят в прямоугольник.

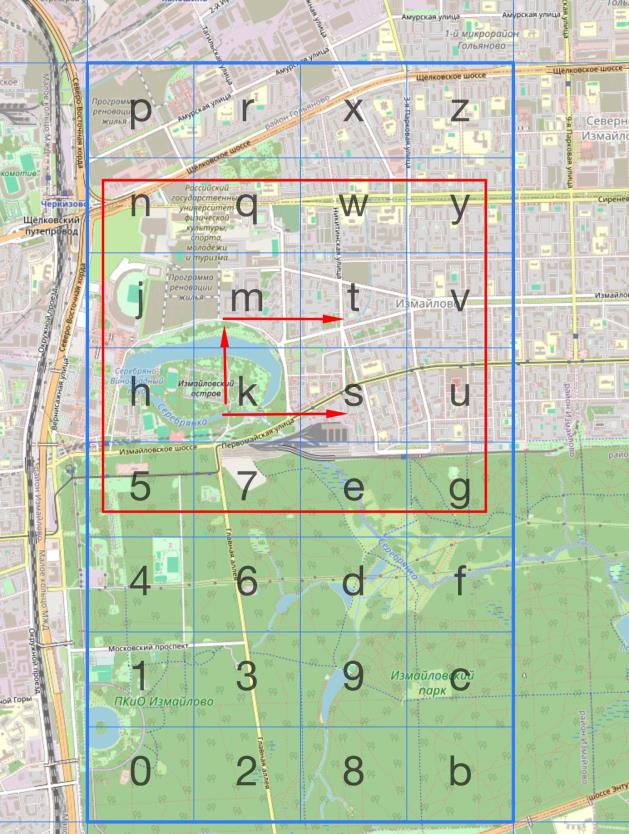


Рисунок 13 — Алгоритм поиска в прямоугольнике

Итерации поиска ближайшего соседа (рисунок 14):

а) найти геохеш точки P. В примере на рисунке 14 - *ucfv7m*;

б) получить все точки, в указанном геохеше;

в) по часовой стрелке итерироваться по соседям оригинального гео­ хеша и выгружать точки. (ucfv7m -> ucfv7q -> ucfv7w и тд);

г) завершить итерацию, когда кол-во найденных точек будет больше или равно K;

д) найти наиболее дальшестоящую точку из найденного массива от­ носительно заданной точки P. Пусть расстояние между указанными точ­ ками будет R;

е) найти все точки в прямоугольнике, с центром в P и сторонами 2R;

ж) точки в полученном массиве отсортировать по расстоянию и вы­ дать первые K точек.

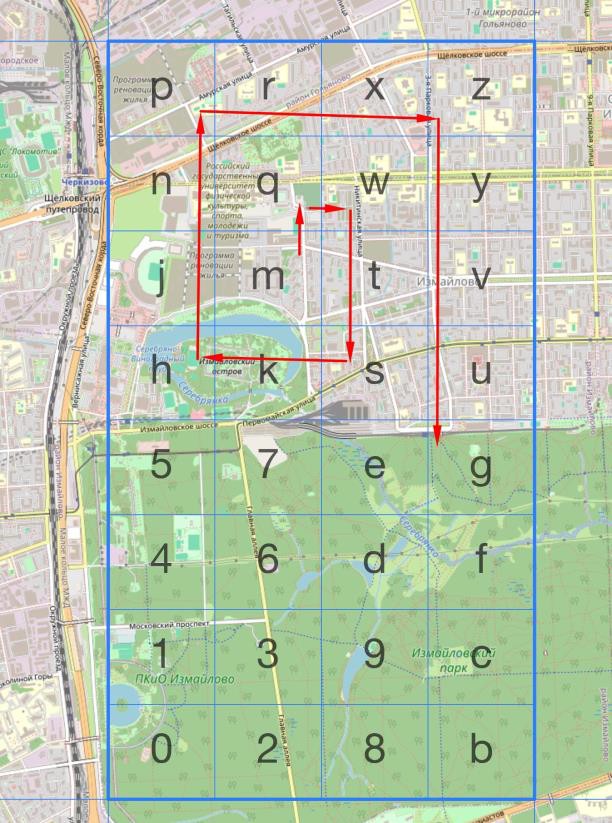


Рисунок 14 — Итеративный проход по соседям

У разработанного алгоритма крайне много преимуществ:

а) клиентский. Может быть полностью реализован на стороне клиен­ та, а не СУБД. Если СУБД не поддерживает какие-либо геоиндексы, как например AWS DynamoDB, данный индекс можно использовать для закрытия потребности в геоиндексах;

б) кластерезируемый. В отличие от R-Tree, данный индекс без про­ блем реализует кластеризацию. Также важно отметить, что B-Tree в

данном индексе использует только операции Get и Set, без сравнения и итераций, за счет чего данный индекс можно применять, например, в СУБД Redis, которая нативно поддерживает кластеризацию ключей;

в) комбинируемый. В случае необходимости, его можно комбиниро­ вать с R-Tree, например, для СУДБ Redis можно хранить в ключах, а в значениях - R-tree через операцию *GEOADD* [[19].](#_bookmark51)

Но также имеется много минусов:

а) параметризируемость. На создание индекса требуется выбрать раз­ мерность Geohash, она не может быть изменена. Если выбранная размер­ ность будет слишком большой или слишком малой, производительность индекса будет слишком низкой;

б) крайние случаи. Если клиент отправит запрос на задачу KNN, при этом в указанной точке и рядом не будет точек, индекс будет медленно итерироваться по всем ближайшим геохешам.

## ПРОВЕДЕННЫЕ ИССЛЕДОВАНИЯ. РЕЗУЛЬТАТЫ

## Результаты тестирования

Практически на всех задачах такие индексы как *перебор* и *дерево квадрантов* показали плохие результаты (рисунок 15), в свою очередь *R-Tree* и *KD-Tree* - основные финалисты.

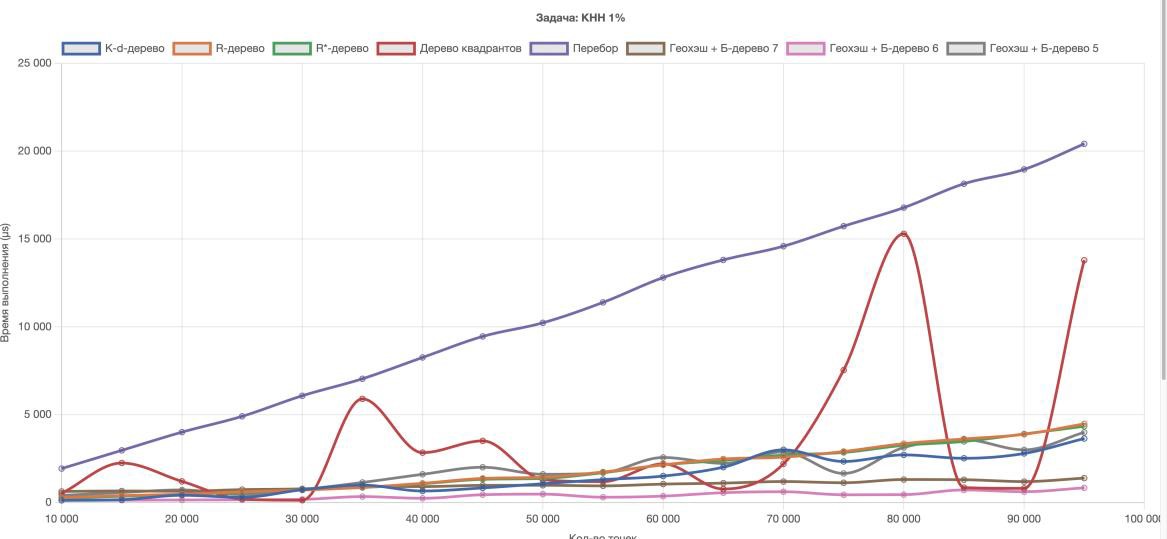


Рисунок 15 — Задача KNN на 1% точек

При этом разработанный индекс Geohash-B-Tree с правильной параметризацией всегда показывает крайне хорошие результаты (Рисунки 16 и 17)

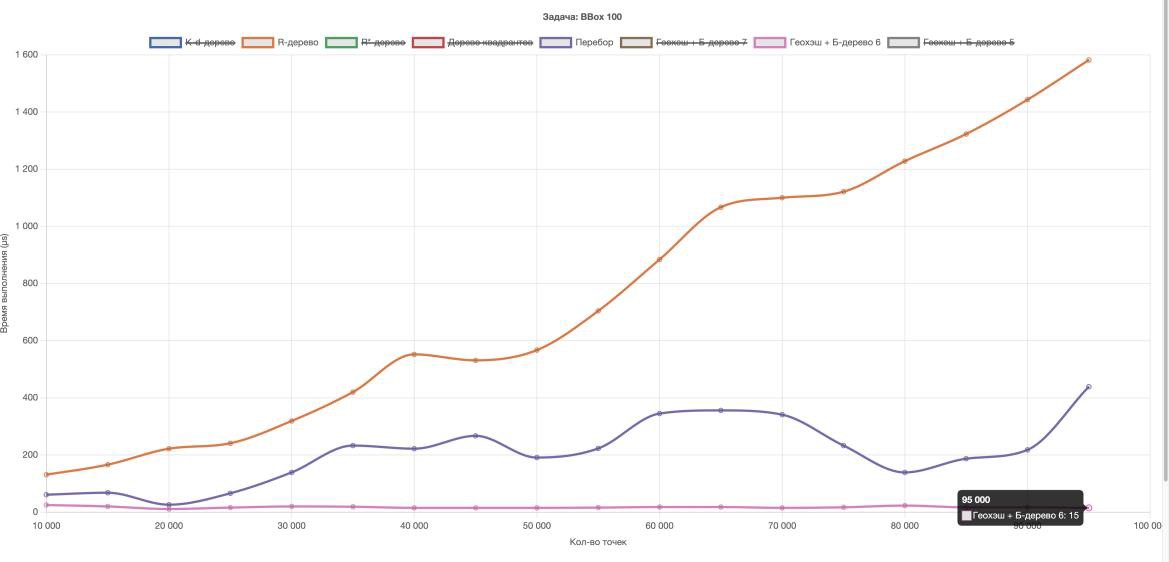


Рисунок 16 — Задача BBox на 100 случайных точках

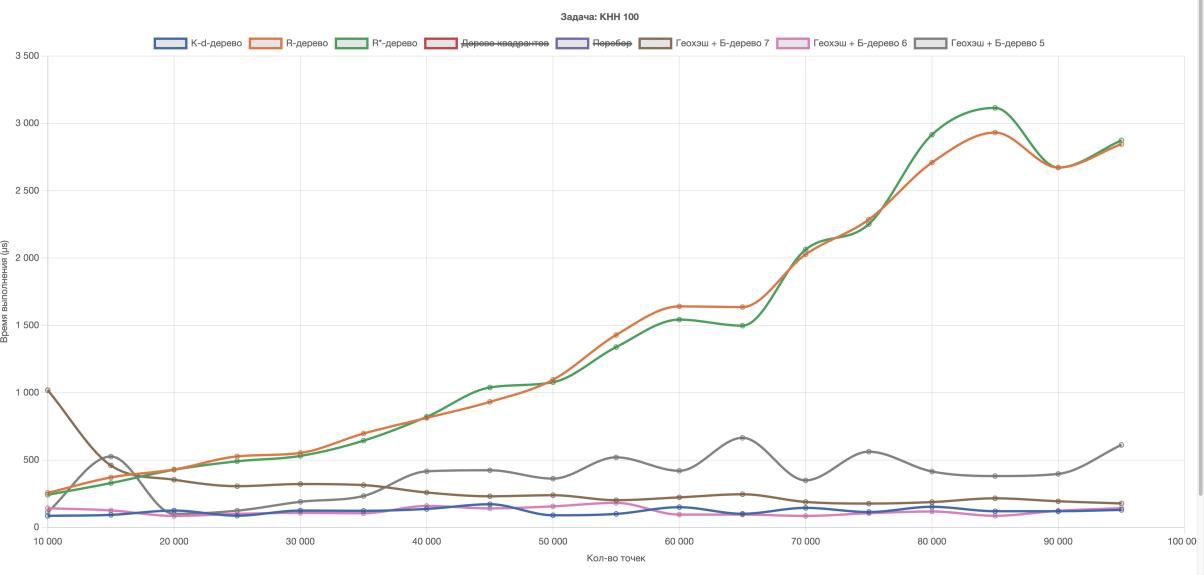


Рисунок 17 — Задача KNN на 100 случайных точках

Таким образом, можно сделать следующие выводы:

а) популярные индексы *R-Tree*, *KD-Tree* и другие - в общем случае показывают крайне хорошие результаты;

б) перебор часто работает быстрее, чем сложные индексы;

в) разработанные индексы также показывают крайне хорошие ре­ зультаты.

## Выводы

По результатам проведенных работ, можно сделать следующие реко­ мендации относительно выбора индексов:

а) если используемая СУБД поддерживает геоиндексы - используйте их, например, из Redis, MongoDB и PostGIS;

б) если есть возможность выбрать индекс - рекомендуется выбрать индекс R-Tree, он имеет крайне большое количество методов, реализа­ ций и в общем случае работает крайне хорошо;

в) если используемая СУБД не поддерживает геоиндексы - их можно реализовать через Geohash + B-Tree. Более сложные реализации (H3 + B-Tree, Geohash + R-Tree) тоже возможны, но не необходимы;

г) если требуется кластеризация, то для ключа кластерации можно использовать Geohash + B-Tree. Другие индексы не поддерживают кла­ стеризацию.

## Выводы по практической части

Разработанное в ходе практической части программное обеспечение позволяет протестировать качество работы тех или иных индексов на раз­ ных задачах. Важно отметить, что полученное ПО является легко расширя­ емым, что позволяет разработчикам его использовать во время разработки, отладки и улучшения указанных индексов.

Преимуществами данной работы являются:

а) изолированность. Сервер исполнителя задач физически отделен от сервера менеджера;

б) наглядность. Пользователь в реальном времени видит результаты, может самостоятельно запрашивать задачи.

# ЗАКЛЮЧЕНИЕ

Таким образом, в рамках выпускной квалификационной работы вы­ полнены следующие пункты:

а) проведен аналитический обзор литературы, в рамках которого бы­ ли расcмотрены такие индексы и методы как: R-Tree, R\*-Tree, KD-Tree, Geohash, H3 и другие;

б) сформулированы содержательная и математическая постановки задачи;

в) в качестве основных инструментариев были выбраны языки про­ граммирования Golang и JavaScript;

г) проработан клиентский путь, разработана функциональная схема программного обеспечения;

д) разработаны дополнительные индексы;

е) продемонстрирована система тестирования, а также результаты её работы. Сделаны выводы по указанным результатам;

ж) основные результаты работы были опубликованы в сборнике тези­ сов конференции «79-е Дни науки студентов [«МИСИС»[20];](#_bookmark52)

з) работа представлена на научной конференции кафедры Инженер­ ной кибернетики «79-е Дни науки «МИСИС».

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Roussopoulos N., Kelley S., Vincent F. Nearest Neighbor Queries // Proceedings ACM SIGMOD. — 1995. — V.24. — №2. — P. 71-79.
2. Guttman A. R-trees: A Dynamic Index Structure for Spatial Searching

// Proceedings of ACM SIGMOD. — 1984. — V.14. — №2. — P. 47-57.

1. Beckmann N., Kriegel H.P., Schneider R. [et al.]. The R\*-tree: An Efficient and Robust Access Method for Points and Rectangles // Proceedings of ACM SIGMOD. — 1990. — V.19. — №2. — P. 322–331.
2. Sun L., Jin B. Improving NoSQL Spatial-Query Processing with Server- Side In-Memory R\*-Tree Indexes for Spatial Vector Data // Sustainability. — 2023. — V.15, — №1. — P. 1-23.
3. Федотовский П. В. и др. Сортировать или нет: экспериментальное сравнение R-Tree и B+-Tree в транзакционной системе для упорядоченной выдачи // Труды Института системного программирования РАН. — 2014.

— Т.26. — №.4. — С.73-90.

1. Liu Q., Tan X., Huang F., [et al.]. GB-Tree: An efficient LBS location data indexing method // 2014 The Third International Conference on Agro- Geoinformatics, Beijing, China. — 2014. — №9. — P. 1-5.
2. Ganti R., Srivatsa M., Agrawal D., [et al.]. MP-trie: fast spatial queries on moving objects // Proceedings of the Industrial Track of the 17th International Middleware Conference. — 2016. — №12. — P. 1-6.
3. Guan X., Bo C., Li Z., [et al.]. ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database // 2017 25th International Conference on Geoinformatics, Buffalo, NY, USA. — 2017. — №8 — P. 1-7.
4. Liu J., Li H., Gao Y., [et al.]. A geohash-based index for spatial data management in distributed memory // IEEE 2014 22nd International Conference on Geoinformatics. — 2014. — №6. — P. 1-4.
5. Sahr K. Central Place Indexing: Hierarchical Linear Indexing Systems for Mixed-Aperture Hexagonal Discrete Global Grid Systems // Cartographica: The International Journal for Geographic Information and Geovisualization. — 2019. — V.54. — №3. — P.16-29.
6. Сидоров, И. Ю., Армяков, А. О., Байтин, А. А., [и др.]. Хранилище точечных геообъектов // Минцевские чтения. — 2015. — C. 234-247.
7. Balki´c Z., Sˇoˇs tari´c D., Horvat G. GeoHash and UUID identifier for multi-

agent systems // Proceedings of the 6th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications. — 2012. — №6. — P. 290–298.

1. Bohui, J., Weifeng, Z. Comparative Analysis of GeoHash, Google S2 and Uber H3 as Global Geographic Grid Coding Methods // Geography Geographic Information Science. — 2024. — V.40. — №2. — P. 19
2. Membrey P., Plugge E., Hawkins T. The definitive guide to MongoDB: the noSQL database for cloud and desktop computing.— New York.: Apress, 2010.— P. 328.
3. S. He S., Chu L., Li X. Spatial query processing for location based application on Hbase // 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China. — 2017. — №3. — P. 110-114.
4. Bayer R., McCreight E. Organization and maintenance of large ordered indices // Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description Access and Control. — 1970. — №11. — P. 107–141.
5. Comer, D. Ubiquitous B-tree // ACM Computing Surveys. — 1979. —

V.11 — №2. — P. 121-137.

1. Гулаков В.К. Многомерные структуры данных.— Брянск.: БГТУ,

2010.— 387 c.

1. Muradova G., Hematyar M., Jamalova J. Advantages of Redis in- memory database to efficiently search for healthcare medical supplies using geospatial data // 2022 IEEE 16th International Conference on Application of

Information and Communication Technologies. — 2022. — №10. — P. 1-5.

1. Ибрагимов П.И. Сравнительный анализ алгоритмов геопоиска и про­ странственных индексов в высоконагруженных системах // 79-е дни науки студентов МИСиС: международные, межвузовские и институтские научно­ технические конференции. Тезисы докладов. — М.: МИСиС, 2022.

# ПРИЛОЖЕНИЕ А

## Основные фрагменты кода

Листинг 1 — points.go

1. // Package geo
2. //
3. // Point , Points method
4. package geo 5
5. import (
6. " encoding / j s o n "
7. " geoindexing\_comparison / backend / h e l p e r s "
8. " github . com/ paulmach/ orb "
9. " github . com/ paulmach/ orb / ge o j s on "
10. " golang . org /x/ exp/ s l i c e s "
11. "math"
12. " s t r i n g s " 14

15 " github . com/ pkg/ e r r o r s " 16

17 " github . com/ mmcloughlin / geohash " 18

19 mapset " github . com/ deckarep / golang−s e t / v2 "

20 )

21

1. // Point r e p r e s e n t s a geographic c o o r d i n a te .
2. type Point s t r u c t {
3. ID s t r i n g ‘ j s o n : " id " ‘

25

1. Lat f l o a t 6 4 ‘ j s o n : " l a t " ‘
2. Lon f l o a t 6 4 ‘ j s o n : " lon " ‘

28 }

29 type Points [ ] Point 30

1. func NewPoint ( l a t f l o a t 6 4 , lng f l o a t 6 4 ) Point {
2. re turn Point {
3. ID : h e l p e r s . ID ( ) ,
4. Lat : l at ,

35

36

37 }

38

Lon : lng ,

}

1. func ( r Point ) Geohash ( b i t s uint ) uint 64 {
2. re turn geohash . Encode Int With Precision ( r . Lat , r . Lon , b i t s )

41 }

42

1. func ( r Point ) Geohash String ( chars uin t ) s t r i n g {
2. re turn geohash . Encode With Precision ( r . Lat , r . Lon , chars )

45 }

46

1. func ( r Point ) GeoJSON( ) ∗ ge o j s on . Feature {
2. f e a t u r e := ge o j s on . NewFeature ( orb . Point { r . Lon , r . Lat })
3. f e a t u r e . Pr o p e r t i e s [ " ID "] = r . ID 50

51 re turn f e a t u r e

52 }

53

1. func ( r Point ) Inside BBox ( bottom Left Point , upper Right Point ) bool {
2. re turn bottom Left . Lat < r . Lat && bottom Left . Lon < r . Lon && r . Lat < upper Right . Lat && r . Lon < upper Right . Lon

56 }

57

1. func ( r Point ) AddLatitude (dvKM f l o a t 6 4 ) Point {
2. r . Lat = r . Lat + (dvKM/ earthRadiusKm ) ∗( 180 / math . Pi )
3. re turn r

61 }

62

1. func ( r Point ) AddLongitude (dvKM f l o a t 6 4 ) Point {
2. r . Lon = r . Lon +

(dvKM/ earthRadiusKm ) ∗( 180 / math . Pi ) /math . Cos ( r . Lat∗math . Pi / 180 )

1. re turn r

66 }

67

1. func ( r ∗ Points ) GetRandomPoint ( ) Point {
2. re turn ( ∗ r ) [ h e l p e r s .RNG. IntN ( l e n ( ∗ r ) ) ] // n o l i n t : gosec // Allowed here

70 }

71

72 func ( r ∗ Points ) Find Corners ( ) ( Point , Point ) {

73

74

75

76

77

78

79

80

81

82

83

84

85 }

86

bottom Left , upper Right := ( ∗ r ) [ 0 ] , ( ∗ r ) [ 0 ] f o r \_, point := range ∗ r {

i f point . Lat < bottom Left . Lat && point . Lon < bottom Left . Lon { bottom Left = point

}

i f point . Lat > upper Right . Lat && point . Lon > upper Right . Lon { upper Right = point

}

}

re turn bottom Left , upper Right

1. func ( r ∗ Points ) Center ( ) ( f l o a t 6 4 , f l o a t 6 4 ) {
2. bottom Left , upper Right := r . Find Corners ( )
3. re turn ( bottom Left . Lat + upper Right . Lat ) / 2 . 0 , ( bottom Left . Lon + upper Right . Lon ) / 2 . 0

90 }

91

1. func ( r ∗ Points ) Str i n g ( ) s t r i n g {
2. byteArray , e r r := j s o n . Marshal ( r )
3. i f e r r != n i l {
4. panic ( e r r o r s . Wrap( err , " f a i l e d to marshal p o i n ts ") )

96 }

97

98 re turn s t r i n g ( byte Array )

99 }

100

1. func ( r ∗ Points ) IDs ( ) s t r i n g {
2. var i d s [ ] s t r i n g
3. f o r \_, point := range ∗ r {
4. i d s = append ( ids , point . ID)

105 }

106

1. s l i c e s . Sort ( i d s )
2. return s t r i n g s . Join ( ids , " ,")

109 }

110

111 func ( r ∗ Points ) Delete ( point ID s t r i n g ) {

112

113

114

115

116

117

118 }

119

f o r idx , point := range ∗ r { i f point ID == point . ID {

∗ r = append ( ( ∗ r ) [ : idx ] , ( ∗ r ) [ idx + 1 : ] . . . ) re turn

}

}

1. func ( r ∗ Points ) ToSet ( ) mapset . Set [ s t r i n g ] {
2. r e s u l t := mapset . NewSet [ s t r i n g ] ( )
3. f o r \_, point := range ∗ r {
4. r e s u l t . Add( point . ID)

124 }

125

126 return r e s u l t

127 }

128

1. func ( r ∗ Points ) GeoJSON( ) ∗ ge o j s on . Fe a tu r e Co l l e c t i o n {
2. f e a t u r e C o l l e c t i o n := ge o j s on . New Feature Collection ( )
3. f o r \_, point := range ∗ r {

|  |  |  |  |
| --- | --- | --- | --- |
| 132 |  | f e a t u r e C o l l e c t i o n . Append ( point . GeoJSON( ) ) |  |
| 133 | } |  |  |
| 134 |  |  |  |
| 135 | return f e a t u r e C o l l e c t i o n | |  |
| 136 | } |  |  |
| 137 |  |  |  |
| 138 | func | ( r ∗ Points ) SortByID ( ) { |  |
| 139 | s l i c e s . SortFunc ( ∗ r , func ( a , b Point ) i n t { | |  |
| 140 |  | re tu rn s t r i n g s . Compare ( a . ID , b . ID) |  |
| 141 | }) | |  |
| 142 | } |  |  |
| 143 |  |  |  |
| 144 | func | ( r ∗ Points ) Sort By Distance ( o r i g i n Point ) { |  |
| 145 | s l i c e s . SortFunc ( ∗ r , func ( a , b Point ) i n t { | |  |
| 146 |  | i f a . Distance Haversine ( o r i g i n ) < b . Distance Haversine ( o r i g i n ) | { |
| 147 |  | re turn −1 |  |
| 148 |  | } |  |
| 149 |  | re tu rn 1 |  |
| 150 | }) | |  |
| 151 | } | |  |

152

|  |  |  |
| --- | --- | --- |
| 153 func | ( r ∗ Points ) Get Closest Via Sort ( o r i g i n Point , n i n t ) Points | { |
| 154 | i f n > l e n ( ∗ r ) { |  |
| 155 | re tu rn ∗ r |  |
| 156 | } |  |
| 157 |  |  |
| 158 | type d i s t s t r u c t { |  |
| 159 | idx i n t |  |
| 160 | d i s t f l o a t 6 4 |  |
| 161 | } |  |
| 162 |  |  |
| 163 | knnMatrix := make ( [ ] d i s t , 0 , l e n ( ∗ r ) ) |  |
| 164 | f o r idx , index Point := range ∗ r { |  |

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183 }

184

knnMatrix = append ( knnMatrix , d i s t { idx : idx , d i s t : index Point . Distance Haversine ( o r i g i n ) })

}

s l i c e s . SortFunc ( knnMatrix , func ( a , b d i s t ) i n t { i f a . d i s t < b . d i s t {

re turn −1

}

re tu rn 0

})

r e s u l t := make ( Points , n )

f o r idx := range n {

r e s u l t [ idx ] = ( ∗ r ) [ knnMatrix [ idx ] . idx ]

}

return r e s u l t

1. func ( r ∗ Points ) Equal ( other Points ) bool {
2. return r . ToSet ( ) . Equal ( other . ToSet ( ) )

187 }

188

1. func ( r ∗ Points ) EqualMany ( other [ ] Points ) bool {
2. f o r \_, other Point := range other {

191



return true

192

193

194

195

196

197

Листинг 2 — distance.go

// Package geo

//

// Distance f u n c t i o n s package geo

import (

" github . com/ t i d w a l l / g e o d e s i c " "math"

)

const earthRadiusKm = 6371 // r a d i u s o f the earth in k i l o m e te r s .

func d i s ta n c e Eu c l i d e a n ( lat 1 , lon 1 , lat 2 , lon 2 f l o a t 6 4 ) f l o a t 6 4 { re turn math . Sqrt ( math . Pow( lat 2 −lat 1 , 2 ) + math . Pow( lon 2 −lon 1 , 2 ) )

}

// degrees To Radians c o n ve r ts from d e g r e e s to r a d i a n s . func degrees To Radians ( d f l o a t 6 4 ) f l o a t 6 4 {

re turn d ∗ math . Pi / 180

}

func d i s ta n c e Ha v e r s i n e ( lat 1 , lon 1 , lat 2 , lon 2 f l o a t 6 4 ) f l o a t 6 4 { l a t 1 = degrees To Radians ( l a t 1 )

lon 1 = degrees To Radians ( lon 1 ) l a t 2 = degrees To Radians ( l a t 2 ) lon 2 = degrees To Radians ( lon 2 )

d i f f L a t := l a t 2 − l a t 1 d i f f Lo n := lon 2 − lon 1

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

|  |  |  |
| --- | --- | --- |
| 31 | Ц · Ц Ц := math . Pow( math . Sin ( d i f f L a t / 2 ) , 2 ) + | |
|  | math . Cos ( l a t 1 ) ∗math . Cos ( l a t 2 ) ∗ | |
| 32 | math . Pow( math . Sin ( d i f f Lo n / 2 ) , 2 ) | |
| 33 |  | |
| 34 | c := 2 ∗ math . Atan2 ( math . Sqrt ( Ц · Ц Ц ) , math . Sqrt(1− Ц · Ц Ц ) ) | |
| 35 |  | |
| 36 | re turn c ∗ earthRadiusKm | |
| 37 | } | |
| 38 |  | |
| 39 | func d i s ta n c e Ge o d e s i c ( lat 1 , lon 1 , lat 2 , lon 2 f l o a t 6 4 ) f l o a t 6 4 { | |
| 40 | var d i s t f l o a t 6 4 | |
| 41 | g e o d e s i c .WGS84 . In v e r s e ( lat 1 , lon 1 , lat 2 , lon 2 , &d i s t , n i l , n i l ) | |
| 42 |  | |
| 43 | re turn d i s t | |
| 44 | } | |
| 45 |  | |
| 46 | func ( r Point ) Distance Haversine ( other Point ) f l o a t 6 4 { | |
| 47 | re turn d i s ta n c e Ha v e r s i n e ( r . Lat , r . Lon , other . Lat , other . Lon ) | |
| 48 | } | |
| 49 |  | |
| 50 | func ( r Point ) Distance Geodesic ( other Point ) f l o a t 6 4 { | |
| 51 | re turn d i s ta n c e Ge o d e s i c ( r . Lat , r . Lon , other . Lat , other . Lon ) | |
| 52 | } | |
| 53 |  | |
| 54 | func ( r Point ) Distance Euclidean ( other Point ) f l o a t 6 4 | { |
| 55 | re turn d i s ta n c e Eu c l i d e a n ( r . Lat , r . Lon , other . Lat , | other . Lon ) |
| 56 | } |  |
|  | Листинг 3 — bbox.go |  |
| 1 | // Package geohash\_utils |  |
| 2 | // |  |
| 3 | // Geohash u t i l s , such as BBox |  |
| 4 | package geohash\_utils |  |
| 5 |  |  |
| 6 | import ( |  |
| 7 | " github . com/ mmcloughlin / geohash " |  |
| 8 | ) |  |
| 9 |  |  |
| 10 | type BBox s t r u c t { |  |
| 11 | b i t s uint |  |

12

13

14

15

16 }

17

l eft Bottom uint 64

h e i gh t i n t wight i n t

1. func NewBBox( bottom LeftLat , bottomLeftLon , upper RightLat , upperRightLon f l o a t 6 4 , b i t s uin t ) BBox {
2. var (
3. wight = 0
4. h e i gh t = 0 22
5. bottomLeftHash = geohash . Encode Int With Precision ( bottom LeftLat , bottomLeftLon , b i t s )
6. upper LeftHash = geohash . Encode Int With Precision ( upper RightLat , bottomLeftLon , b i t s )
7. bottomRightHash = geohash . Encode Int With Precision ( bottom LeftLat , upperRightLon , b i t s )

26 )

27

1. f o r {
2. i f bottomLeftHash == upper LeftHash {
3. break

31 }

32 upper LeftHash = Neighbor Int With Precision ( upper LeftHash , b i ts , geohash . South )

|  |  |  |  |
| --- | --- | --- | --- |
| 33 |  | h e i gh t++ |  |
| 34 | } |  |  |
| 35 |  |  |  |
| 36 | f o r | { |  |
| 37 |  | i f bottomLeftHash == bottomRightHash { | |
| 38 |  | break |  |
| 39 |  | } |  |
| 40 |  | bottomRightHash = Neighbor Int With Precision ( bottomRightHash , b i ts ,  geohash . West) | |
| 41 |  | wight++ |  |
| 42 | } |  |  |
| 43 |  |  |  |
| 44 | r := | BBox{ |  |
| 45 |  | b i t s : | b i ts , |

46

47

48

49 }

50

l eft Bottom : bottomLeftHash , h e i gh t : height ,

wight : wight ,

51

52 }

53

re turn r

1. func c o l l e c t Pe r i m e t e r ( hash uint64 , b i t s uint , height , wight i n t ) [ ] uint 64 {
2. perimeter := make ( [ ] uint 64 , 0 , h e i gh t ) 56
3. f o r range h e i gh t {
4. perimeter = append ( perimeter , hash )
5. hash = Neighbor Int With Precision ( hash , b i ts , geohash . North )

60 }

61

1. f o r range wight {
2. perimeter = append ( perimeter , hash )
3. hash = Neighbor Int With Precision ( hash , b i ts , geohash . East )

65 }

66

1. f o r range h e i gh t {
2. perimeter = append ( perimeter , hash )
3. hash = Neighbor Int With Precision ( hash , b i ts , geohash . South )

70 }

71

1. f o r range wight {
2. perimeter = append ( perimeter , hash )
3. hash = Neighbor Int With Precision ( hash , b i ts , geohash . West)

75 }

76

77 re turn perimeter

78 }

79

1. // Perimeter
2. // Returns outer part o f BBox
3. func ( r ∗BBox) Perimeter ( ) [ ] uint 64 {
4. re turn c o l l e c t P e r i m e t e r ( r . left Bottom , r . b i ts , r . height , r . wight )

84 }

85

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 86  87  88  89  90 | // Inner  // Retur func ( r  var | ns i n n e r  ∗BBox) In  (  h e i gh t | par ner  = | t o f BBox p o i n ts from Perimeter are not i n c lu d e d ( ) [ ] uint 64  r . h e i gh t − 1 |
| 91 |  | wight | = | r . wight − 1 |
| 92 |  | i n n e r | = | make ( [ ] uint 64 , 0 , r . h e i gh t ) |
| 93 |  | hash | = | Neighbor Int With Precision ( r . left Bottom , r . b i ts , |
| geohash . NorthEast ) | | | | |
| 94 |  | inner Hash = hash | | |
| 95 | ) |  |  |  |
| 96 |  |  |  |  |
| 97 | f o r | range h e i gh t { | | |
| 98 |  | inner Hash = hash | | |
| 99 |  | f o r range wight { | | |
| 100  101  102  103  104  105  106  107 | retu | i n n e r = append ( inner , inner Hash )  inner Hash = Neighbor Int With Precision ( inner Hash , r . b i ts ,    hash = Neighbor Int With Precision ( hash , r . b i ts , geohash . North )  rn i n n e r | | |

# ПРИЛОЖЕНИЕ Б

## Дополнительные результаты тестирований

В данном приложение на рисунках 18-24 показаны результаты тестов по всем имеющимся задачам и всем имеющимся индексам.

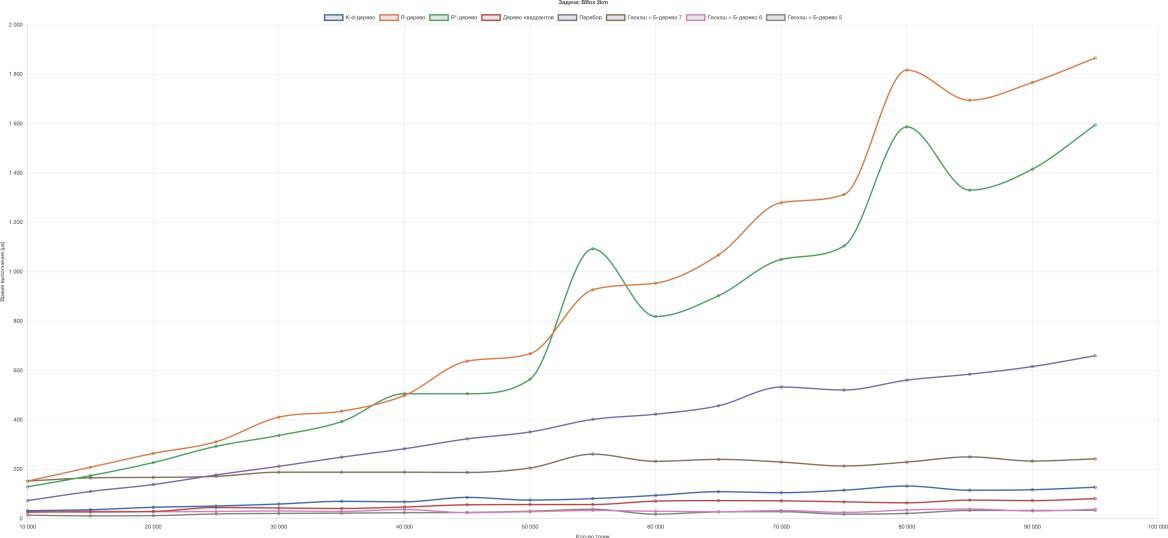


Рисунок 18 — Задача BBox на квадрате с ребром 2км

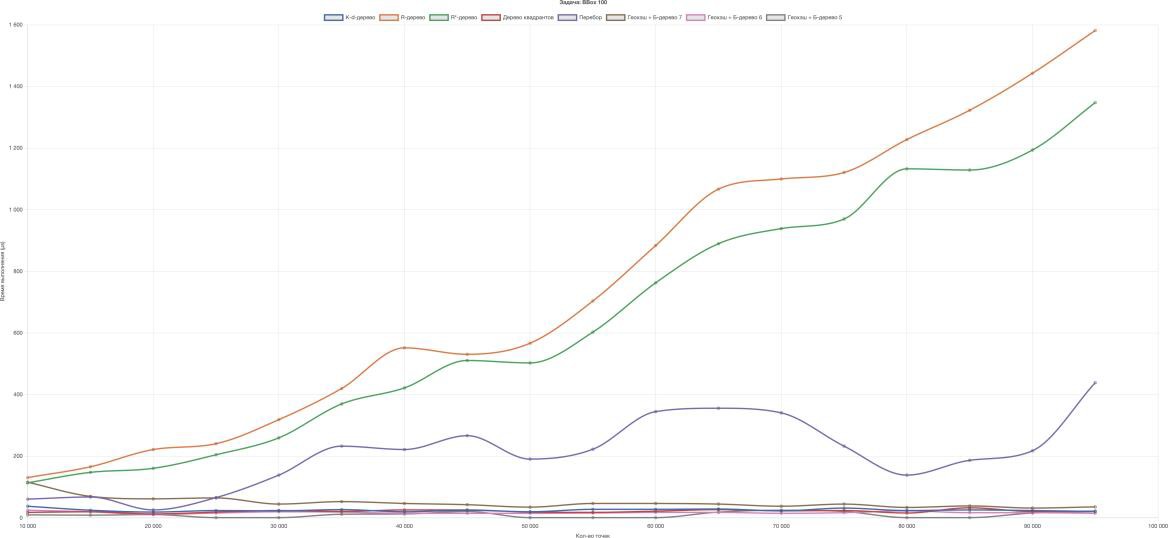


Рисунок 19 — Задача BBox на квадрате, в который входит не более 100

точек

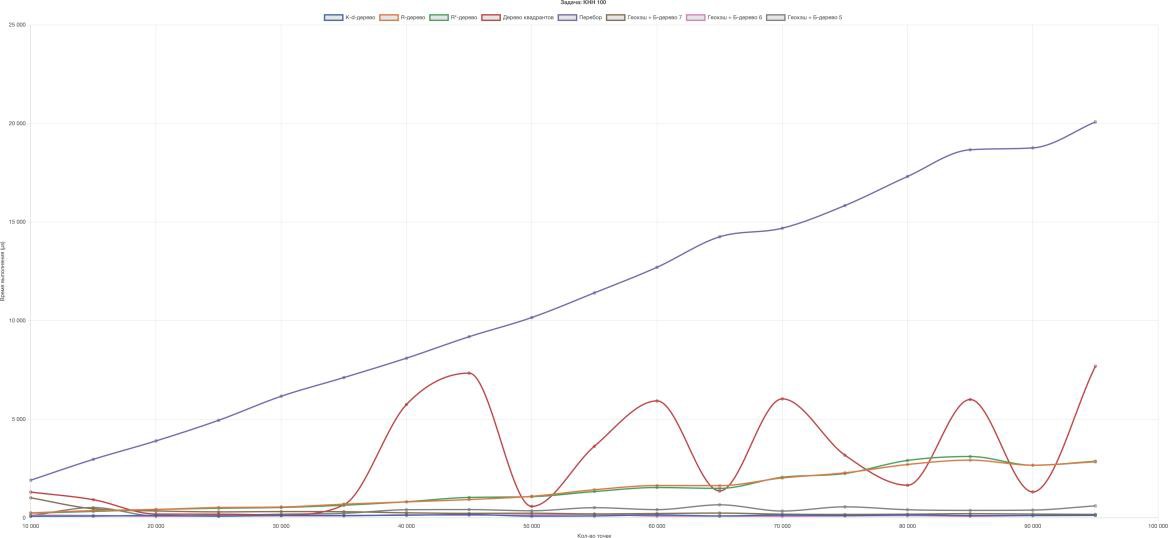


Рисунок 20 — Задача KNN на 100 точек

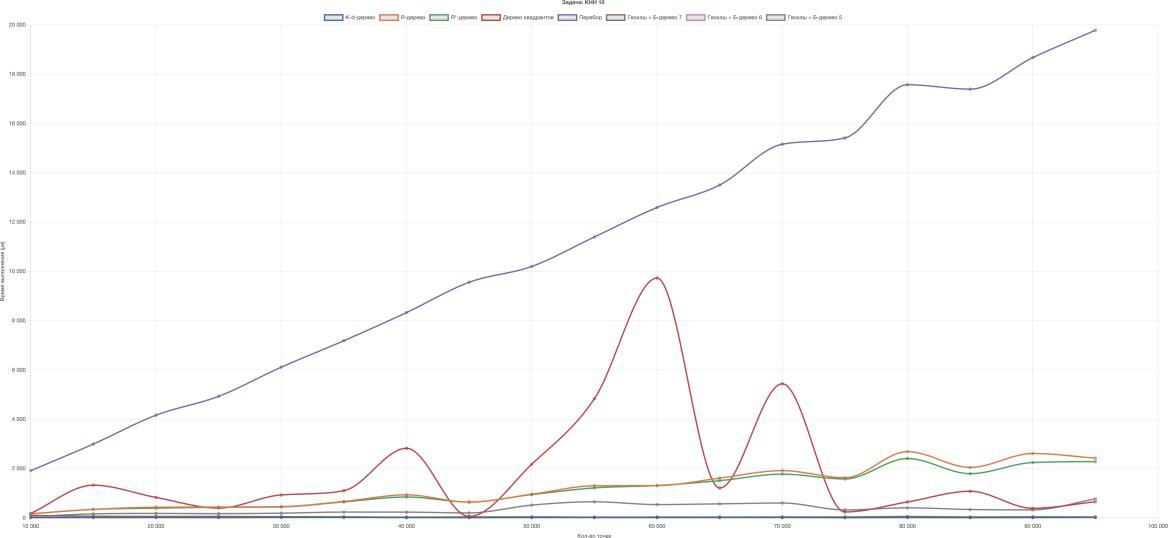


Рисунок 21 — Задача KNN на 10 точек

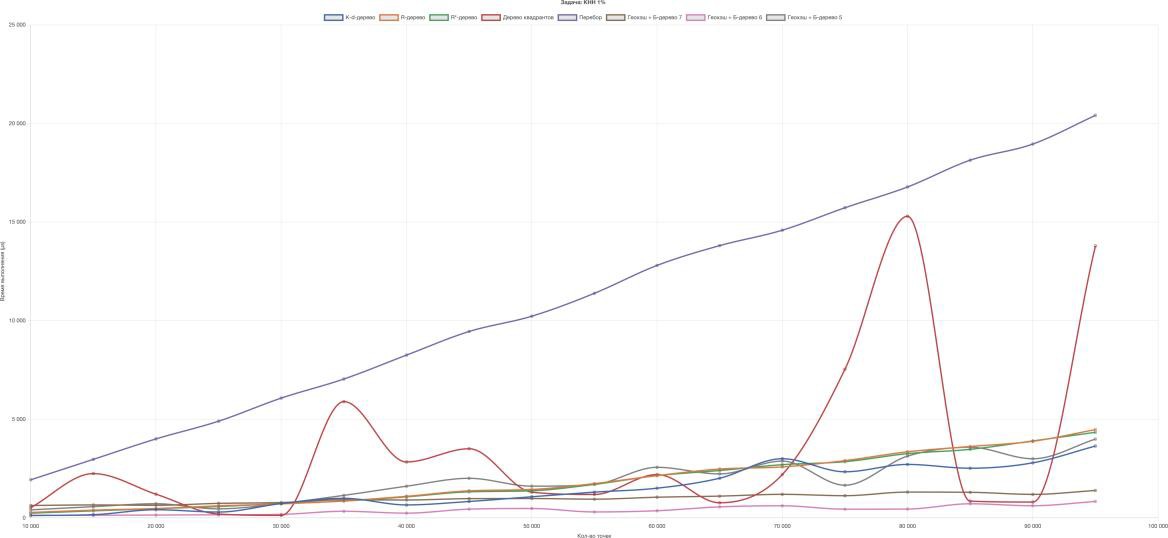


Рисунок 22 — Задача KNN на 1% точек

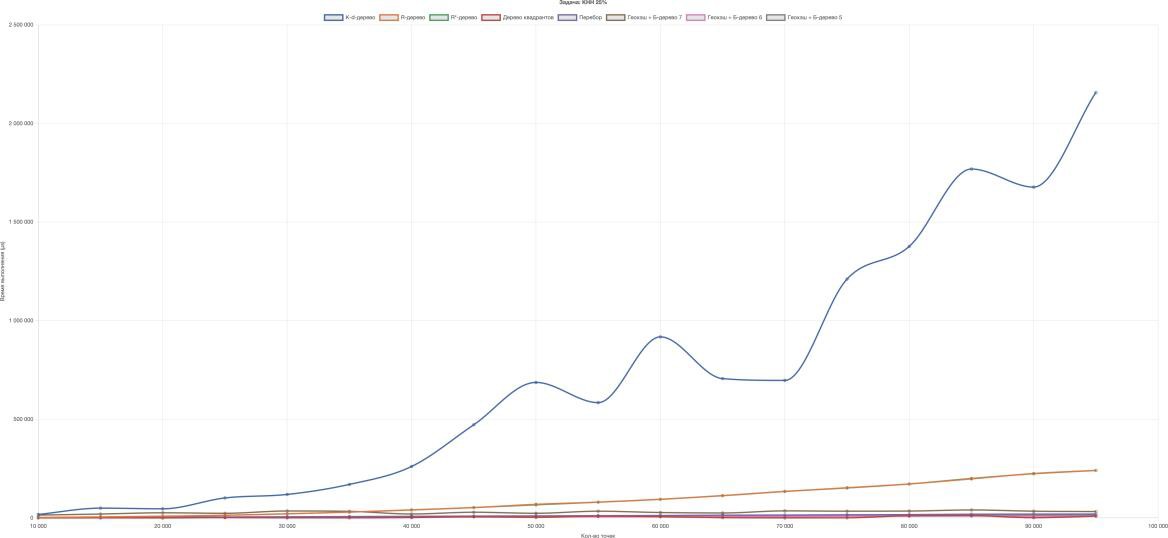


Рисунок 23 — Задача KNN на 25% точек

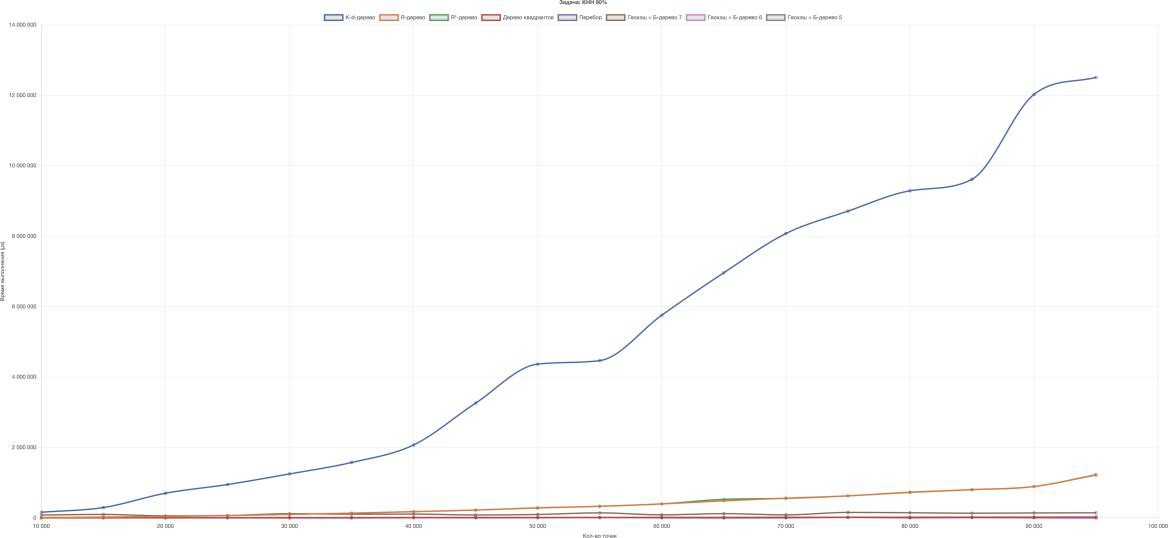


Рисунок 24 — Задача KNN на 90% точек