Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів та алгоритмів роботи з повнотекстовими індексами

(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-5

Сорокін В. В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник д.ф.-м.н., проф. Смеляков С.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сорокіну Володимиру Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів та алгоритмів роботи з повнотекстовими індексами»

Затверджена наказом по університету від «29» березня 2024 р. №250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17.06.2024 р

3. Вихідні дані до роботи дослідження методів та алгоритмів роботи з повнотекстовими індексами

4. Перелік питань, що потрібно опрацювати в роботі

метою роботи – аналіз предметної галузі, дослідження існуючих алгоритмів прогнозування, підготовка даних та встановлення метрик, проведення експериментального дослідження, проєктування архітектури ансамблю моделей, побудова індексів, розробка програмної системи, висновки, перелік джерел посилань

КАЛЕНДАРНИЙ ПЛАН

№	Назви етапів курсової роботи	Термін виконання етапів роботи	Примітка
1	Видача теми, узгодження і затвердження	02.04.2024	виконано
2	Аналіз предметної галузі	02.04.2024	виконано
3	Огляд існуючих методів	09.04.2024	виконано
4	Оформлення пояснювальної записки	05.05.2024	виконано
5	Здача готового проекту	17.06.2024	виконано

Дата видачі завдання 02 квітня 2024 р.

Студент _____ Сорокін В.В
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ д.ф.-м.н., проф. Смеляков С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Практична робота магістра містить: 55 с., 10 рис., 10 джерел.

АНАЛІТИКА, ПАТЕРН, ПРОГРАМНА СИСТЕМА, BOOLEAN SEARCHING.

Об’єктом дослідження – методи роботи із повнотекстовими індексами.

Метою роботи – є дослідження предметної області методів роботи із повнотекстовими індексами

В результаті буде знайдено оптимальні методи пошуку та розміщення повнотекстових індексів.

ANALYTICS, PATTERN, SOFTWARE SYSTEM, BOOLEAN SEARCHING.

The object of the study is methods of working with full-text indexes.

The purpose of the work is to research the subject area of methods of working with full-text indexes

As a result, optimal methods of searching and placing full-text indexes will be found.

Я, Сорокін Володимир Віталійович, студент гр. ІПЗм-22-5, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя робота на тему «Дослідження методів та алгоритмів роботи з повнотекстовими індексами», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В сучасному світі дані мають дуже велику вагу в нашому житті, вони надають нам різноманітну інформацію різної направленості та форми, тому зараз об’єми даних все зростають а взаємодія із ними стає все складнішою.

Щоб ефективно взаємодіяти з великими об’ємами даних, розробники використовують індексування, воно використовується для всіх типів даних, та повнотекстові індекси є най-поширенішими але мають великий обсяг та складну структуру.

Однак, з усіма перевагами, які надають повнотекстові індекси, виникає ряд викликів та проблем, особливо коли мова йде про вибір оптимального методу пошуку.

З огляду на різноманіття та обсяги інформації, традиційні методи пошуку можуть бути неефективними або недостатньо гнучкими для задоволення сучасних інформаційних потреб. Водночас, зростання обсягів даних і складність запитів користувачів вимагають від систем більш розвинутих можливостей пошуку, що включають контекстуальний аналіз, обробку складних запитів та ефективне використання ресурсів.

Ці вимоги ставлять перед дослідниками та розробниками завдання розробки більш сучасних, швидких та точних методів пошуку, здатних оптимізувати процес пошуку в повнотекстових індексах.

Таким чином, предметна область нашого дослідження охоплює розробку та аналіз методів пошуку в повнотекстових індексах, з особливим акцентом на їх застосування у великих дата-сетах. Мета полягає в тому, щоб забезпечити ефективний доступ до великих обсягів текстової інформації, що є критично важливим у багатьох сферах сучасного цифрового світу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області індексування.

Індексування є фундаментальним процесом у управлінні даними, який відіграє ключову роль у покращенні ефективності пошуку та доступу до інформації у великих базах даних. Воно полягає у створенні індексів, які діють як карти або покажчики на дані, збережені у базах даних, файлових системах чи інших сховищах інформації. Індекси уможливлюють швидке знаходження необхідної інформації без необхідності перегляду всієї бази даних, що значно збільшує швидкість пошуку та знижує час відгуку системи.

Основна функція індексування полягає у підвищенні швидкості обробки запитів, особливо в системах з великими обсягами даних. Без індексів кожен запит до бази даних потребував би послідовного перегляду всіх записів, що могло б бути надзвичайно часо-витратним. Індекси дозволяють прямо звертатися до потрібних даних, оминаючи непотрібні записи, що робить процес пошуку значно швидшим і ефективнішим.

Індексування також має велике значення у контексті повнотекстового пошуку, де обробляються великі обсяги текстової інформації, такі як документи, статті, блоги та інші текстові дані. У повнотекстовому індексуванні створюються індекси, які мапають ключові слова або фрази до місць їх зустрічі у тексті. Це дозволяє користувачам швидко знаходити інформацію за допомогою ключових слів або запитів, витрачаючи значно менше часу, ніж при ручному пошуку.

Крім того, індексування важливе для оптимізації сховищ даних і баз даних, оскільки дозволяє більш ефективно організувати та управляти даними. Індекси можуть бути реалізовані різними способами в залежності від конкретних вимог і особливостей системи. Наприклад, у реляційних базах даних часто використовуються B-дерева або хеш-таблиці для індексування.

Таким чином, індексування є не тільки технічним інструментом, але й ключовим компонентом у розробці ефективних систем обробки даних, який дозволяє забезпечити швидкий, точний та гнучкий доступ до інформації.

Повнотекстові індекси – це особливий тип індексації даних, який орієнтований на зберігання, організацію та швидкий доступ до великих обсягів текстової інформації. Цей метод індексації з'явився як відповідь на зростаючу потребу в швидкому та ефективному пошуку в масивах текстових даних, які стали широко поширеними з розвитком інтернету та цифрових технологій.

В основі повнотекстових індексів лежить зберігання великих обсягів тексту, таких як книги, статті, веб-сторінки, електронні документи тощо. Вони дозволяють зберігати ці тексти у структурованому та легкодоступному форматі.

Одна з основних функцій повнотекстових індексів - надання здатності швидко знаходити інформацію в тексті. Це досягається за допомогою створення індексів, які вказують на місцезнаходження певних слів або фраз у великих текстових файлах.

Повнотекстові індекси дозволяють не тільки знаходити певні слова, але й аналізувати контекст, у якому вони використовуються. Це дозволяє користувачам отримувати більш точні та релевантні результати.

Із зростанням кількості цифрового контенту, особливо текстового, виникла необхідність у більш ефективних методах організації та пошуку інформації.

Сучасний світ характеризується високою швидкістю отримання та обробки інформації. Повнотекстові індекси забезпечують швидкий доступ до необхідної інформації, що є критично важливим у багатьох сферах, від наукових досліджень до повсякденного використання.

Повнотекстові індекси використовуються пошуковими системами для індексації та пошуку веб-сторінок, блогів, форумів та інших ресурсів в Інтернеті. Це дозволяє користувачам швидко знаходити інформацію на основі їхніх запитів.

У наукових та академічних базах даних повнотекстові індекси використовуються для пошуку статей, публікацій та інших дослідницьких матеріалів. Це дозволяє дослідникам та студентам ефективно проводити літературні огляди та пошук даних для своїх проектів. Великі компанії використовують повнотекстові індекси для управління та пошуку в корпоративних базах даних, документації, звітах та інших ділових документах.

Юридичний Пошук: У юридичній сфері повнотекстові індекси незамінні для пошуку великих обсягів законодавчих та судових документів, що дозволяє юристам і суддям швидко знаходити необхідні прецеденти та законодавчі акти.

Медіа і Розваги: Повнотекстові індекси також застосовуються для організації та пошуку великих баз даних мультимедійного контенту, таких як фільми, музика, електронні книги та інші.

У підсумку, повнотекстові індекси є надзвичайно важливим інструментом в ері цифрових технологій. Вони дозволяють швидко та ефективно обробляти величезні обсяги текстової інформації, забезпечуючи користувачам доступ до потрібних даних. Розвиток цих систем продовжує відкривати нові можливості для пошуку, аналізу та використання інформації у різних сферах життя.

1.2 Актуальність проблеми

Проблема вибору ефективного методу пошуку за повнотекстовими індексами є актуальною з кількох причин. По-перше, із зростанням обсягу цифрових даних, особливо текстової інформації, що генерується користувачами в інтернеті, соціальних мережах, в наукових базах даних, архівах та корпоративних системах, виникає величезна потреба у швидкому та точному доступі до цієї інформації. Повнотекстові індекси дозволяють організувати великі набори даних таким чином, щоб користувачі могли ефективно здійснювати пошук, знаходячи не тільки документи, що містять певні слова або фрази, але й оцінюючи релевантність цих документів до запиту.

Крім того, збільшення обсягу даних створює виклики, пов'язані з обробкою запитів та видачею результатів пошуку. Традиційні методи, такі як простий лінійний пошук або навіть складніші алгоритми, можуть бути неефективними або надмірно ресурсозатратними при роботі з великими дата-сетами. Це призводить до необхідності вибору методу пошуку, який би оптимізував процес індексації та пошуку, забезпечуючи при цьому високу точність та швидкість обробки.

Нарешті, з урахуванням різноманітності запитів користувачів та складності інформаційних потреб, методи пошуку повинні бути гнучкими та вміти

адаптуватися до різних типів даних та запитів. Це означає, що методи пошуку мають враховувати не тільки ключові слова, але й контекст, в якому ці слова використовуються, а також здатні обробляти складні запити, які включають логічні оператори або регулярні вирази.

У підсумку, вибір методу пошуку за повнотекстовими індексами є критично важливим у світі, де інформація стає все більш доступною, але водночас розпорошеною та об'ємною.

1.3 Постановка задачі

У ході даної роботи нами буде досліджено предметну область повнотекстових індексі, методів пошуку за ними та методи модифікації роботи із ними. Ми проведемо порівняльний аналіз та аналіз ефективності кожного методу пошуку.

Ця робота спрямована створення дорожньої карти для розробників що до знаходження найефективнішого методу пошуку за повнотекстовими індексами, також мета роботи – модернізувати деякі підходи для досягнення кращого результату у пошуку.

Реалізація наукового дослідження складається з наступних етапів:

- аналіз предметної області;
- аналіз методів пошуку за повнотекстовими індексами;
- розгляд особливостей кожного методу;
- порівняльний аналіз та виведення критеріїв порівняння;
- модифікація пошуку за файловими системами;
- розробка бенчмарку для методів;
- розробка розбиття на підфайли;

- формування висновків.

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ

Повнотекстові індекси організовують дані таким чином, що забезпечує швидкий доступ до інформації. Вони дозволяють системам пошуку не просто знаходити документи, що містять певні слова або фрази, але й оцінювати релевантність цих документів до запиту пошуку, враховуючи різні параметри, такі як частота слів, їх взаємне розташування та контекст.

Використання цих методів пошуку дозволяє користувачам ефективно обробляти запити, що включають складні комбінації ключових слів, фраз, логічних умов, а також виконувати специфічні запити, такі як пошук у певних полях документу або в межах певного діапазону дат чи чисел.

Таким чином, методи пошуку у повнотекстових індексах є фундаментальними у сферах, де потрібно швидко обробляти великі обсяги інформації, таких як інтернет-пошук, наукові дослідження, юридична обробка документів та корпоративні інформаційні системи. Далі ми будемо розглядати їх більш детально.

2.1 Методи роботи із повнотекстовими індексами

З початку ми розберемо методи пошуку за повнотекстовими індексами, спочатку будемо обирати із популярних та відносно компактних варіантів.

Простий лінійний пошук. Простий лінійний пошук, який також відомий як послідовний пошук, є одним із найпростіших алгоритмів пошуку. Цей метод полягає у послідовному перегляді кожного елемента у наборі даних (наприклад, у списку або масиві) для знаходження елемента, який відповідає заданим критеріям пошуку.

Метод включає в себе перевірку кожного документа у колекції для визначення, чи містить він запитувану інформацію. Алгоритм починає перевірку з першого документа та продовжує послідовно до тих пір, поки не знайде відповідний документ або не перегляне всі документи в наборі.

Плюси простого лінійного пошуку:

- алгоритм простий у розумінні та легкий у реалізації, що робить його зручним для використання в простих задачах або як основу для більш складних алгоритмів;
- для простого лінійного пошуку не потрібні додаткові структури даних, що зменшує вимоги до пам'яті;
- у ситуаціях, коли обсяг даних малий, цей метод може бути досить швидким і ефективним.

Мінуси простого лінійного пошуку:

- простий лінійний пошук стає надзвичайно неефективним при роботі з великими наборами даних, оскільки кожен елемент (або документ) перевіряється по черзі;
- час, необхідний для пошуку, пропорційний кількості елементів у наборі даних, що може бути неприйнятно повільним для великих баз даних;
- у цьому методі відсутні механізми оптимізації, які могли б покращити швидкість пошуку, такі як індексування або поділ на під-набори.

У підсумку, простий лінійний пошук є базовим інструментом, який може бути корисним у деяких сценаріях, особливо коли потрібно розробити просту пошукову систему або для навчальних цілей. Однак, у більш складних або об'ємних задачах пошуку, рекомендується використовувати більш просунуті методи та алгоритми, які забезпечують кращу швидкість та точність пошуку.

Булевий пошук. Булевий пошук – це метод пошуку в повнотекстових індексах, який використовує булеву логіку для комбінування словникових запитів. Цей метод базується на трьох основних логічних операторах: AND, OR та NOT

- AND використовується для знаходження документів, які містять усі вказані терміни. Наприклад, запит "кіт AND собака" поверне документи, які містять обидва ці слова;
- OR дозволяє знаходити документи, що містять будь-який із вказаних термінів. Наприклад, "кіт OR собака" поверне документи, які містять слово "кіт", "собака" або обидва;
- NOT використовується для виключення документів, що містять певний термін. Наприклад, "кіт NOT собака" поверне документи, що містять слово "кіт", але не містять слово "собака".

Плюси булевого пошуку:

- дозволяє точно вказати, які слова або фрази повинні або не повинні бути присутніми у результаті;
- можна комбінувати різні умови пошуку для створення складних запитів;
- булеві оператори є простими і інтуїтивно зрозумілими, що робить метод доступним навіть для неспеціалізованих користувачів.

Мінуси булевого пошуку:

- не дозволяє враховувати відтінки значень чи релевантність запитів;
- строгі умови пошуку можуть призвести до виключення релевантних документів, якщо вони не відповідають точним булевим критеріям;
- користувач мусить точно знати, які слова використовувати в запиті, щоб отримати бажані результати[1].

Булевий пошук чудово підходить для ситуацій, де необхідно точно визначити критерії пошуку, але може бути менш ефективним у випадках, коли

потрібно врахувати контекст або відтінки значень. Особливо це актуально у великих базах даних або коли необхідно знайти інформацію за менш конкретними або нечіткими запитами.

Пошук за допомогою регулярних виразів. Пошук за допомогою регулярних виразів є потужним і гнучким інструментом у системах повнотекстового пошуку. Регулярні вирази дозволяють визначати складні шаблони у тексті, які можуть включати різні комбінації символів, слова, послідовності символів або навіть цілі текстові структури.

Регулярні вирази використовують спеціальний синтаксис для опису шаблонів у тексті. Наприклад, символ . може відповідати будь-якому символу, а конструкції, такі як * або +, дозволяють вказувати на повторення певних елементів. Вони можуть бути використані для виявлення слів, які починаються з певних букв, мають певну довжину, або відповідають певним комбінаціям символів.

Плюси регулярного виразу пошуку:

– регулярні вирази дозволяють точно визначати шаблони пошуку, що робить їх ідеальними для складних або дуже специфічних запитів;
– регулярні вирази ефективні для пошуку складних текстових структур, таких як електронні адреси, телефонні номери, дати тощо;
– використовуються у багатьох областях, включаючи обробку тексту, програмування, аналіз даних.

Мінуси регулярного виразу пошуку:

- для ефективного використання регулярних виразів необхідно добре розуміти їх синтаксис, що може бути складним для новачків;
- складні регулярні вирази можуть бути повільними при виконанні, особливо на великих обсягах даних;
- існує ризик отримати невірні або неповні результати пошуку, якщо регулярний вираз неправильно сформульовано.

У цілому, регулярні вирази є надзвичайно потужним інструментом для проведення складних текстових пошуків. Вони дають можливість реалізувати дуже точний і гнучкий пошук, але вимагають від користувача високого рівня знань і уваги до деталей при формулюванні запитів. Цей метод особливо корисний в ситуаціях, де потрібно ідентифікувати специфічні шаблони у тексті, які складно визначити за допомогою простіших методів пошуку.

Проте, важливо зауважити, що регулярні вирази можуть не завжди бути найкращим вибором для простих пошукових задач або коли потрібно швидко обробляти великі обсяги даних. У таких випадках можуть бути більш підходящими інші методи пошуку, що оптимізовані для швидкості та простоти використання.

Метод перетину. Метод перетину (Intersection) є ключовим алгоритмом для визначення документів, які відповідають декільком критеріям пошуку одночасно. Цей метод зазвичай використовується в ситуаціях, де потрібно знайти документи, які містять більше ніж один пошуковий термін.

При використанні методу перетину, система спочатку ідентифікує набори документів, які відповідають кожному окремому терміну пошуку. Наприклад, якщо користувач шукає документи, що містять слова "економіка" та "технології", система спочатку знаходить усі документи зі словом "економіка" та окремо всі документи зі словом "технології". Потім вона виконує перетин цих двох наборів, щоб знайти лише ті документи, які містять обидва ці слова.

Плюси методу перетину:

- метод перетину забезпечує високу точність, оскільки він знаходить лише ті документи, які точно відповідають всім критеріям пошуку;
- цей метод є ефективним для виявлення найбільш релевантних документів, коли користувач шукає специфічну комбінацію термінів.

Мінуси методу перетину[2]:

- метод перетину може пропустити релевантні документи, які містять лише один з пошукових термінів або використовують синоніми;
- ефективність цього методу безпосередньо залежить від якості індексації документів та від того, наскільки добре система може ідентифікувати та індексувати всі релевантні терміни.

На рисунку 3.1 можна побачити архітектурну взаємодію складових частин реалізації методу перетину.

На діаграмі можна побачити наступні елементи та їхні взаємодії:

- запит користувача: точка входу, де користувач вводить критерії для пошуку;
- модуль пошуку 1 і 2: компоненти, які виконують пошук в різних базах даних або індексах;
- база даних 1 і 2: місця зберігання даних, де виконується пошук;
- процесор запитів: компонент, який обробляє вхідні запити, виконує пошук через модулі пошуку та збирає результати;
- результати перетину: вихідний компонент, який представляє об'єднання (перетин) результатів обох пошукових запитів.

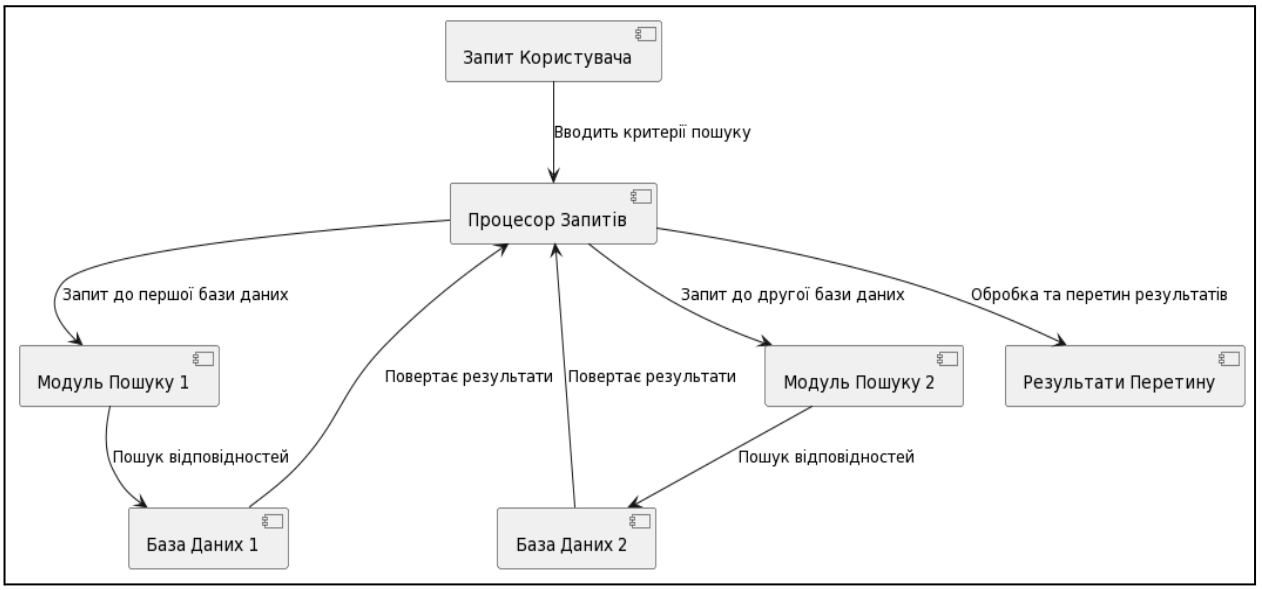


Рисунок 3.1 – Взаємодія елементів методу перетину

У цілому, метод перетину є важливим інструментом у повнотекстовому пошуку, особливо коли потрібно знайти документи, які точно відповідають кільком конкретним критеріям. Він ідеально підходить для детальних та специфічних пошукових запитів, але може бути обмежений у ситуаціях, де потрібен більш гнучкий або всеосяжний пошук.

Метод пошуку. Метод пошуку позиції відноситься до групи методів визначення точної локації (або позиції) слова або фрази всередині документа. Цей метод використовується, наприклад, для забезпечення функціоналу підсвічування тексту у пошукових системах, де слова або фрази, що відповідають запиту, виділяються у тексті документа.

У рамках повнотекстового індексу, цей метод включає зберігання інформації не лише про наявність слова у документі, а й про його точне розташування, наприклад, номер рядка або позиція у рядку. Коли виконується пошуковий запит, система може не тільки знайти всі документи, які містять запитувані слова, але й точно показати, де ці слова знаходяться у тексті.

Плюси методу пошуку позиції:

- метод забезпечує високу точність, вказуючи не лише на наявність слова, але й на його точне розташування в документі;

- цей метод може бути корисним для аналітичних цілей, наприклад, для визначення частоти та розподілу певних слів або фраз у тексті;
- підсвічування знайдених термінів у тексті полегшує користувачам орієнтування у великих обсягах інформації.

Мінуси методу пошуку позиції:

- зберігання інформації про позиції займає більше місця в індексі, що може призвести до збільшення вимог до дискового простору та пам'яті;
- залежно від оптимізації індексу, визначення позицій може додати додатковий час обробки запиту.

Загалом, метод пошуку позиції є важливим інструментом в повнотекстових пошукових системах, особливо коли потрібна висока точність і деталізація результатів пошуку. Він особливо корисний в системах, де важливий швидкий та ефективний доступ до інформації, а також у сценаріях, де необхідний глибокий аналіз текстових даних[3].

Метод фільтрування документів. Метод фільтрування документів у результатах пошуку є важливим інструментом у сучасних пошукових системах, зокрема у контексті повнотекстових пошуків. Цей метод використовується для уточнення та оптимізації результатів пошуку, забезпечуючи користувачам можливість знайти найбільш релевантну інформацію.

Фільтрування документів у результатах пошуку включає в себе застосування різних критеріїв для відсіювання несуттєвих або менш релевантних документів із загального набору результатів. Фільтри можуть бути засновані на різних атрибутах, таких як дата публікації, автор, тип документа, теги, географічне розташування та інше.

Плюси методу фільтрування документів:

- фільтрація допомагає забезпечити вищу релевантність результатів, відкидаючи ті, що не відповідають певним критеріям;
- користувачі можуть швидше знайти потрібну інформацію, задавши специфічні параметри пошуку;
- фільтри зазвичай мають зрозумілий та легкий у використанні інтерфейс, що полегшує навігацію по результатам пошуку.

Мінуси методу фільтрування документів:

- існує ризик пропустити важливі документи, які не відповідають точно встановленим критеріям фільтрування;
- ефективність фільтрації залежить від точності та повноти метаданих, що асоціюються з документами;
- користувачам необхідно мати чітке уявлення про те, що вони шукають, щоб ефективно використовувати фільтри.

На рисунку 3.2 можна побачити архітектурну взаємодію складових реалізації цього методу.

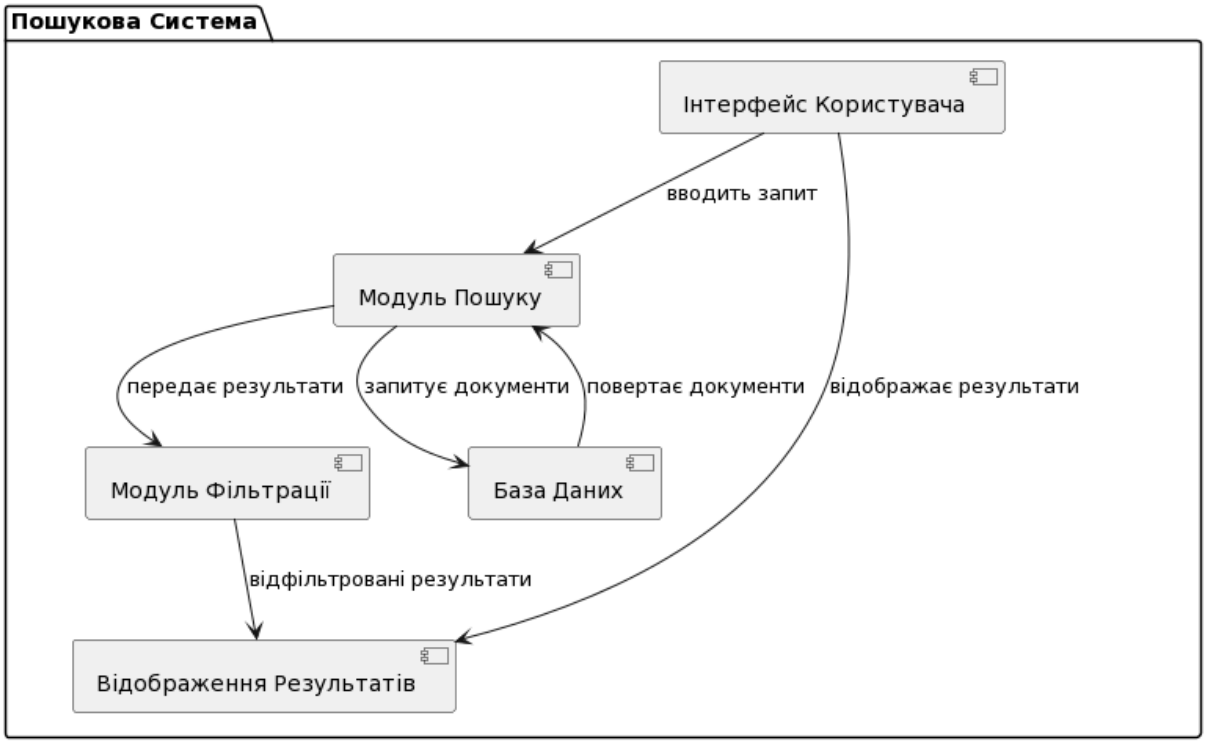


Рисунок 3.2 – Взаємодія елементів методу фільтрації документів

Діаграма відображає наступні компоненти та їхні взаємодії:

- інтерфейс користувача (ui): місце, де користувачі вводять свої пошукові запити;
- модуль пошуку (search): компонент, який обробляє пошукові запити та звертається до бази даних для отримання результатів;
- база даних (db): зберігає інформацію та документи, які можуть бути знайдені пошуковою системою;
- модуль фільтрації (filter): фільтрує отримані від модуля пошуку результати на основі заданих користувачем критеріїв;
- відображення результатів (display): показує кінцеві відфільтровані результати користувачам.

Фільтрування документів у результатах пошуку є важливим інструментом у багатьох пошукових системах, особливо в тих, де обсяги даних є великими. Воно допомагає користувачам зосередитись на найбільш релевантних документах, зменшуючи час, необхідний для знаходження потрібної інформації.

3 ЗНАХОДЖЕННЯ ЄФЕКТИВНОГО МЕТОДУ

Аналізуючи предметну область по використанню та роботі із повнотекстовими індексами – можна зрозуміти що зазвичай це є певні збірки, будь то музики, текстових матеріалів таких як книги чи статті, також повнотекстові індекси часто використовуються для аналізу новин та навісних статей яких може буде доволі багато у вибірці, отже виходячи з цього можна зрозуміти що варто розглядати методи пошуку які в першу чергу зосереджені на роботі із великими обсягами даних[4].

Виходячи із цього надалі нами буде розглянуто та порівняно методи у контексті ефективності роботи із великими обсягами даних.

3.1 Програмна система тестування

Для початку варто розглянути програмну систему яка фактично представляє із себе бенчмарк у якій буде реалізовано та проведено усі порівняння та тести. На рисунку 3.1 зображено файлову структуру документу.

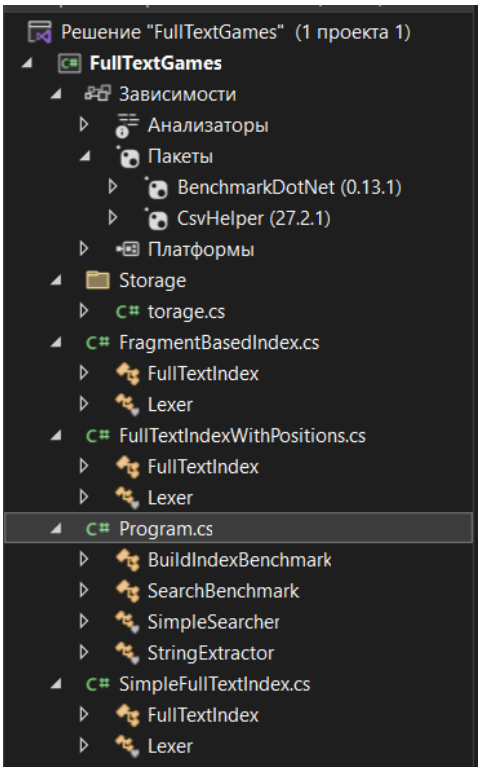


Рисунок 3.1 – Файлова система

Нижче розглянемо структуру.

Механізми індексування:

- simplefulltextindex.cs: реалізує базовий повнотекстовий індекс, який відображає токени до ідентифікаторів документів, де з'являється токен. він використовує просту хеш-таблицю для зберігання відповідностей;
- fulltextindexwithpositions.cs: цей індекс додатково зберігає позиції кожного токена в документах. це корисно для функцій, які потребують точного розташування токенів, наприклад, для підсвічування пошукових термінів у контексті;
- fragmentbasedindex.cs: використовує підхід, заснований на фрагментах, для індексування, де кожен фрагмент тексту вважається окремим елементом для індексування. це може підвищити ефективність пошуку за рахунок зменшення кількості даних, які потрібно аналізувати під час пошуку;
- torage.cs: цей файл включає інтерфейси та реалізації для зберігання вмісту, включно з бінарним зберіганням документів і індексуванням. це дозволяє ефективно зберігати і швидко відновлювати документи з диску.
- Програма використовує об'єкти типу SimpleSearcher для виконання пошуку тексту в індексованих даних. SimpleSearcher має методи для пошуку строк у збережених документах і виводу інформації у зручному форматі.

Загальний Потік Програми:

- програма спершу індексує документи, використовуючи один із доступних методів індексування;

- індексовані дані зберігаються у відповідних сховищах, в залежності від конфігурації і потреб;
- пошук виконується через виклики до simplesearcher, який шукає через індекси і повертає результати на основі запитів користувача.

3.2 Тестування продуктивності

Дла початку ми порівняємо та протестуємо продуктивність звичайного пошуку перебором. Перед тестуванням різних підходів до замірів швидкості пошуку по повнотекстовим індексам ми вирішили спочатку перевірити швидкість звичайного перебору значень. Цей підхід здається базовим і на перший погляд неефективним, особливо для великих наборів даних. Однак, враховуючи потужність сучасних комп'ютерів, навіть простий лінійний пошук може показати високу продуктивність на відносно невеликих датасетах.

Комп'ютери сьогодні оснащені потужними процесорами та великим об'ємом оперативної пам'яті, що дозволяє їм швидко виконувати великі обчислювальні завдання. Завдяки цьому простий лінійний пошук може бути досить ефективним для первинної оцінки швидкості та продуктивності системи. Перевірка швидкості звичайного перебору значень дозволяє нам отримати базові показники продуктивності, з якими можна порівнювати більш складні методи пошуку. Це допомагає визначити, чи варто впроваджувати складніші алгоритми або ж базовий підхід вже забезпечує необхідну швидкість і ефективність для конкретного обсягу даних[5].

До того ж, простий лінійний пошук не вимагає складних структур даних або додаткових витрат на побудову індексів, що може бути корисним у початкових етапах розробки або для систем з невеликим обсягом даних. Таким чином, тестування швидкості звичайного перебору значень є важливим кроком у нашому

дослідженні, що дозволяє оцінити базову продуктивність системи та прийняти обґрунтовані рішення щодо подальших оптимізацій та впровадження більш складних методів пошуку.

Отже у нас є приклад дата-сету на 10 000 слів(див. рис. 3.1).

Матеріал було взято із відкритого ресурсу який нараховує 10 000 строк, тип – ключ значення(див. рис. 3.2).

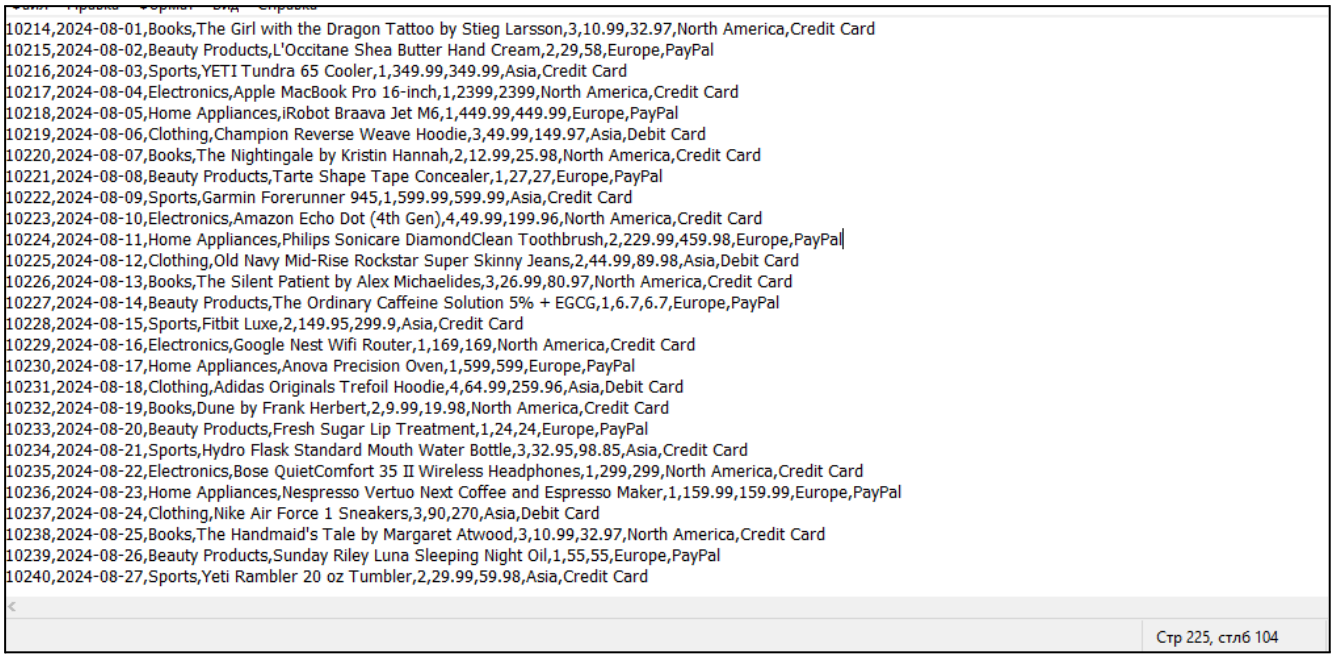


Рисунок 3.1 – Тестовий дата-сет

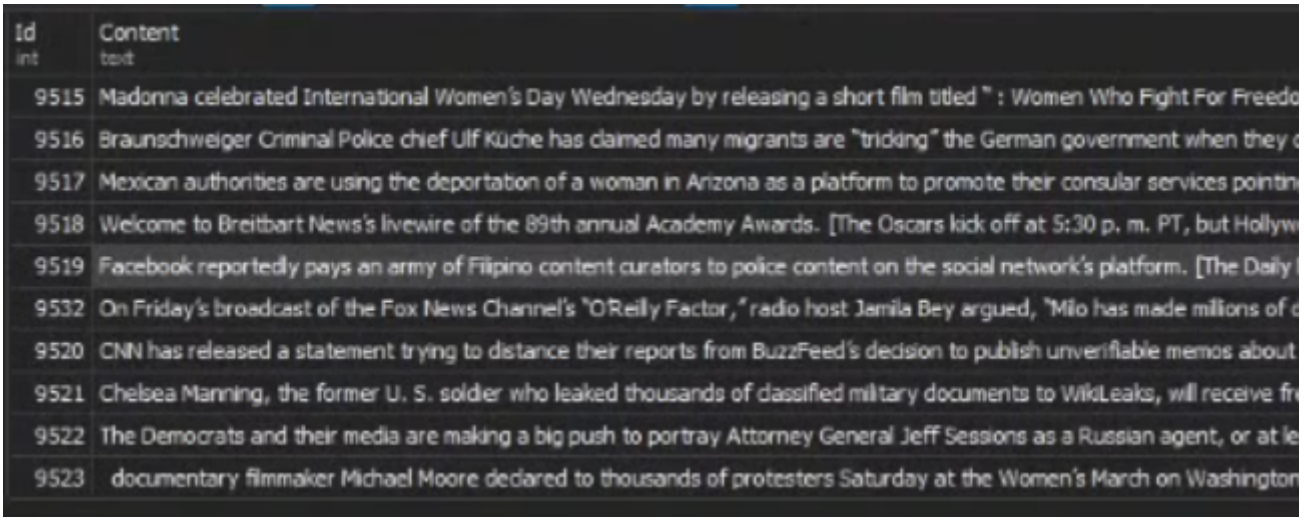


Рисунок 3.2 – Ключ-значення

Це гарний набір даних адже він не величезний як наприклад бібліотеки або не доволі малі як збірка відгуків, для більшої зручності та швидкості, дані описую збірку американських навісних каналів[6].

Отже було перебрано по всіх строках ключове слово, та ми отримали результат у 1.5с.

Отже лінійний перебір, або простий лінійний пошук, стає неефективним, коли кількість даних або користувачів значно зростає. Це пов'язано з тим, що час, необхідний для виконання лінійного перебору, збільшується прямо пропорційно до кількості елементів у наборі даних, тобто має лінійну залежність. У системах з великою кількістю користувачів або великими обсягами даних цей підхід стає вкрай неефективним через декілька причин:

При лінійному переборі кожен запит вимагає перевірки кожного елемента в наборі даних до знаходження потрібного або перегляду всього набору у випадку відсутності відповідного елемента.

У випадку з великими наборами даних, кожен запит може займати значну кількість часу, що призводить до затримок і зниження продуктивності.

Коли одночасно працює багато користувачів, кожен запит створює додаткове навантаження на процесор і пам'ять системи.

Лінійний перебір потребує значних обчислювальних ресурсів, особливо при великій кількості запитів, що може призвести до перевантаження системи.

Результати відпрацювання методу зображено на рисунку 3.3

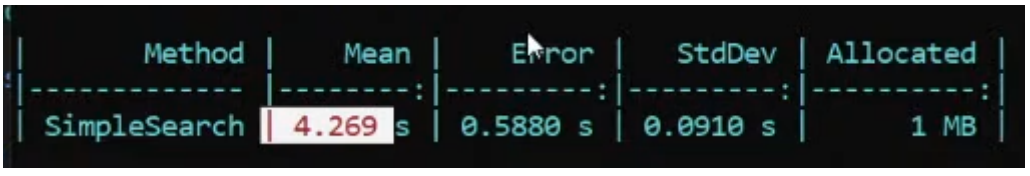


Рисунок 3.3 – Результат відпрацювання Simple search

В результаті як базове значення від якого ми будемо відштовхуватись – 4.269 секунди на наш дата-сет.

3.3 FragmentBasedIndex

Клас FragmentBasedIndex використовує підхід, який дещо відрізняється від традиційних методів індексування, де індексуються окремі слова. Замість цього, цей метод зосереджується на індексації фрагментів тексту. Це може бути корисно для оптимізації пошуку в документах, які містять багато тексту, або коли потрібно зберігати більше контексту навколо ключових слів.

FragmentBasedIndex відносить фрагменти тексту до індексу, який містить інформацію про початок та кінець кожного фрагмента, ідентифікатор документа, де цей фрагмент знаходиться, та ідентифікатори документів для швидкого доступу. Цей підхід може покращити точність пошуку, забезпечуючи більший контекст навколо запитів і допомагаючи знайти більш релевантні фрагменти тексту.

Текст розбивається на фрагменти фіксованого розміру (наприклад, кожні 64 токени). Це створює логічні блоки тексту, кожен з яких може бути індексований окремо.

Кожен фрагмент асоціюється з унікальним ідентифікатором документа і зберігається в індексі зі своїми позиціями (початок та кінець у документі). Це дозволяє швидко визначити, у якому документі та де саме знаходиться кожний фрагмент.

Під час виконання пошукового запиту, система спочатку знаходить усі фрагменти, які містять токени, пов'язані з запитом. Вона може потім відновити контекст, переглядаючи не тільки конкретні слова, що шукаються, але й навколишній текст, що допомагає користувачам краще зрозуміти, чому цей фрагмент був відібраний як релевантний.

Розглянемо переваги:

- контекстуальність: цей підхід може бути особливо корисним у контексті великих текстових документів, де потрібно зберегти контекст навколо ключових слів;

- ефективність: завдяки фрагментному підходу, можливо більш ефективно використовувати ресурси пам'яті та процесорний час, оскільки не потрібно вилучати великі обсяги тексту з бази даних при кожному пошуковому запиті;
- покращене відновлення даних: забезпечує можливість відновити не просто документи, але й специфічні частини документів, які найбільш релевантні до пошукового запиту.

FragmentBasedIndex може бути використаний у системах, де текст містить багато технічної або наукової інформації, у юридичних базах даних, наукових дослідженнях, або будь-яких інших застосуваннях, де контекст та точність пошуку мають велике значення.

Далі розглянемо код:

```
public class FragmentBasedIndex
{
    private readonly Dictionary<string, HashSet<int>> index = new
Dictionary<string, HashSet<int>>();
    private readonly List<string> content = new List<string>();
    private readonly List<Fragment> fragments = new List<Fragment>();
    public const int ChunkSize = 64;
    public void AddStringToIndex(string item)
    {
        int documentId = content.Count;
        content.Add(item);

        foreach (var tokenChunk in
Lexer.GetTokens(item).Chunk(ChunkSize))
        {
            var fragment = new Fragment() {
                DocumentId = documentId,
                Start = tokenChunk.First().Position,
                End = tokenChunk.Last().Position
            };

            foreach (var token in tokenChunk)
            {
                if (!index.TryGetValue(token.Token, out var set))
                {
                    set = new HashSet<int>();
                    index[token.Token] = set;
                }
                set.Add(fragments.Count);
            }
```

```
                fragments.Add(fragment);
            }
        }
        public IEnumerable<string> Search(string word)
        {
            if (!index.TryGetValue(word, out var fragmentIds))
                yield break;

            foreach (var id in fragmentIds)
            {
                var fragment = fragments[id];
                var text = content[fragment.DocumentId];
                yield return text.Substring(fragment.Start, fragment.End
- fragment.Start);
            }
        }
    }
```

Структура даних:

- index: словник, де ключем є токен, а значенням – множина ідентифікаторів фрагментів, де цей токен зустрічається;
- content: список, що зберігає весь текст кожного документа;
- fragments: список, що містить фрагменти тексту, кожен з яких має ідентифікатор документа, початок і кінець у тексті.

Додавання До Індексу:

- текст документа додається до content;
- текст розбивається на фрагменти розміром chunksize. кожен фрагмент індексується з інформацією про позиції токенів у документі;
- циклічно проходиться через кожен токен у фрагменті, додаючи ідентифікатор фрагмента до індексу під відповідним токеном.

Пошук:

- при пошуку слова, ідентифікатори фрагментів, які містять слово, витягуються з index[7];

- для кожного ідентифікатора фрагмента відновлюється текст фрагмента з content, використовуючи збережені позиції початку та кінця.

Цей підхід особливо корисний для систем, де потрібно зберігати значний контекст навколо пошукових запитів, наприклад, у наукових дослідженнях, юридичних документах чи літературних аналізах. Завдяки фрагментації можливо швидко ідентифікувати і відображати релевантні частини тексту без необхідності аналізувати весь документ.

3.4 FullTextIndexWithPositions

Клас FullTextIndexWithPositions, який ми використовуємо в своєму проекті, є реалізацією повнотекстового індексу, який, на відміну від простіших індексів, зберігає не тільки інформацію про наявність токена (слова) в документі, але й точні позиції цих токенів у тексті документа. Це дозволяє реалізувати більш розширені функції пошуку, такі як підсвічування слова в контексті або пошук з урахуванням порядку слів.

Структура індексу:

- індекс: основою є словник (dictionary), де ключем є токен, а значенням є словник із ідентифікаторами документів і списками позицій, де цей токен зустрічається у відповідному документі. це дозволяє швидко знаходити не тільки документи, що містять певний токен, але й точні місця цих токенів у тексті;
- контент: окремий список (list), що зберігає весь текст кожного документа за його ідентифікатором, забезпечуючи доступ до повного тексту для подальшого аналізу або відображення.

Під час додавання тексту до індексу, текст спочатку обробляється лексичним аналізатором (Lexer), який розділяє текст на токени та визначає їхні позиції.

Для кожного токена перевіряється, чи вже існує відповідний запис у зовнішньому словнику індексу. Якщо так, то додається нова позиція або новий документ із позиціями. Якщо ні, створюється новий запис.

Для виконання пошуку по індексу використовуються методи, які можуть швидко визначити наявність та локацію токенів у документах. Це включає здатність виконувати як простий пошук одного слова, так і більш складні запити з декількома словами та фразами[8].

Збережені позиції дозволяють не тільки визначати, чи містить документ певне слово, але й точно вказувати, де саме в документі це слово знаходиться. Це особливо корисно для реалізації функцій підсвічування, аналізу контексту тощо.

FullTextIndexWithPositions ідеально підходить для застосувань, де необхідно забезпечити високу точність пошуку та багатий користувацький досвід, таких як системи юридичного документообігу, наукових досліджень, або будь-які інші області, де текстовий контент є складним і багатим на контекст. Цей метод забезпечує високу ефективність і точність, роблячи його важливим інструментом для сучасних систем пошуку. Нижче наведено код реалізації:

```
     public class FullTextIndex
     {
         private readonly Dictionary<string, Dictionary<int, List<int>>>
_index = new ();
         private readonly List<string> _content = new ();
         private readonly Lexer _lexer = new ();

         public void AddStringToIndex(string text)
         {
             int documentId = _content.Count;
             foreach (var token in _lexer.GetTokens(text))
             {
                 if (_index.TryGetValue(token.Token, out var set))
                 {
                     if (set.TryGetValue(documentId, out var positions))
                         positions.Add(token.Position);
                     else
                         set.Add(documentId, new List<int>() {
token.Position });
                 }
                 else
                 {
                     _index.Add(token.Token, new Dictionary<int,
List<int>>()
                     {
```

```
                    [documentId] = new List<int>() { token.Position }
                });
            }
        }
        _content.Add(text);
    }
}
```

3.5 Аналіз результатів

Нижче наведено результат відпрацювання нашого бенчмарку(див. рис. 3.5).

```
InvocationCount=5  IterationCount=5  UnrollFactor=1
WarmupCount=1

|          Method |     Mean |      Error |   StdDev | Ratio | RatioSD |    Gen 0 | Allocated |
|---------------- |---------:|-----------:|---------:|------:|--------:|---------:|----------:|
|     ReadDataset | 904.4 us | 127.46 us | 33.10 us |  1.00 |    0.00 |        - |    556 KB |
|     SimpleIndex | 390.6 us |  20.57 us |  5.34 us |  0.43 |    0.02 |        - |    419 KB |
| PositionedIndex | 824.5 us | 224.51 us | 58.30 us |  0.91 |    0.07 |        - |    662 KB |
|   FragmentIndex | 684.9 us | 322.34 us | 49.88 us |  0.75 |    0.08 | 200.0000 |  1,303 KB |
```

Рисунок 3.5 – Результати тестування методів текстового індексування

На зображенні відображено таблицю результатів бенчмаркінгу для чотирьох різних методів індексування документів, а також читання датасету. Результати включають виміри таких параметрів, як середній час виконання (Mean), похибка (Error), стандартне відхилення (StdDev), співвідношення до базового методу (Ratio), варіація співвідношення (RatioSD), кількість використань пам'яті у поколінні Gen 0 і загальний обсяг виділеної пам'яті (Allocated).

PositionedIndex має найбільший час виконання серед індексуючих методів і використовує більше пам'яті, що може бути пов'язано з додатковим навантаженням через збереження позиційних даних для кожного токена.

FragmentIndex показує добрі результати за часом, але використовує значно більше пам'яті, ніж інші методи. Це свідчить про те, що він зберігає більшу кількість даних, ймовірно через індексацію більших фрагментів тексту.

Підведем загальний аналіз:

- ефективність: simpleindex є найефективнішим з точки зору швидкості та використання пам'яті;

- функціональність: positionedindex і fragmentindex можуть забезпечувати більш деталізовані можливості для пошуку, але це йде на шкоду швидкості та використання пам'яті;
- обрання методу: вибір методу залежить від вимог до точності пошуку та доступних ресурсів. simpleindex підходить для швидкого і простого пошуку, тоді як fragmentindex і positionedindex краще використовувати, коли потрібна висока точність та контекстуальність пошуку.

Ці результати дають змогу зрозуміти, як різні методи індексування можуть впливати на продуктивність системи і вибір конкретного методу має базуватися на специфічних вимогах до пошукової системи.

3.6 Порівняння методів пошуку

З початку буде порівняно простий лінійний пошук, булевий пошук (Boolean Searching) та регулярний вираз пошуку[9].

Простий Лінійний Пошук – виконує послідовний перегляд кожного елемента датасету для знаходження відповідностей.

Цей метод стає надзвичайно неефективним у великих базах даних, оскільки вимагає перевірки кожного елемента на відповідність запиту, що призводить до значного збільшення часу обробки.

Булевий Пошук (Boolean Searching) – використовує логічні оператори (AND, OR, NOT) для формування запитів, що дозволяє комбінувати кілька умов пошуку. Метод дозволяє швидко фільтрувати великі обсяги даних, забезпечуючи точні та релевантні результати. Ефективно використовує індекси, що мінімізує кількість необхідних операцій.

Регулярний Вираз Пошуку (Regular Expression Search) – використовує спеціалізований синтаксис для створення складних шаблонів пошуку. Хоча це потужний інструмент для складних шаблонів пошуку, він може бути ресурсо

затратним та повільним на великих дата-сетах, особливо при використанні складних шаблонів.

У сценаріях з великими обсягами даних, Булевий Пошук виявляється найбільш підходящим через його здатність швидко і точно обробляти запити, використовуючи логічні операції. Він забезпечує оптимальне співвідношення між швидкістю, точністю та ефективністю. На відміну від простого лінійного пошуку, який є повільним і неефективним для великих даних, та регулярного виразу пошуку, який може бути занадто складним та ресурс затратним, булевий пошук пропонує збалансоване рішення для обробки великих наборів даних.

Також варто зазначити що сучасні обчислювальні можливості дозволяють на даних в оперативній пам’яті також ефективно працювати[10].

Також важливо розібрати метод перетину, адже у великих дата-сетах часто зустрічаються документи, які містять широкий спектр інформації. Метод перетину дозволяє точно ідентифікувати документи, які відповідають кільком умовам пошуку одночасно, забезпечуючи високу точність вибірки.

Це особливо важливо в сценаріях, де користувач шукає дуже специфічну інформацію, що вимагає відповідності кільком критеріям.

В цьому контексті варто розібрати роботу (sets) та їх перетину (intersect).

На рисунку 3.6 наведено діаграма яка відображає дві множини документів (Set 1 та Set 2) та їх перетин (Intersection). Кожен документ представлений окремим блоком, а блоки перетину вказують на документи, які знаходяться у обох множинах. Це візуалізує, як метод перетину використовується для ідентифікації спільних елементів між двома множинами даних.

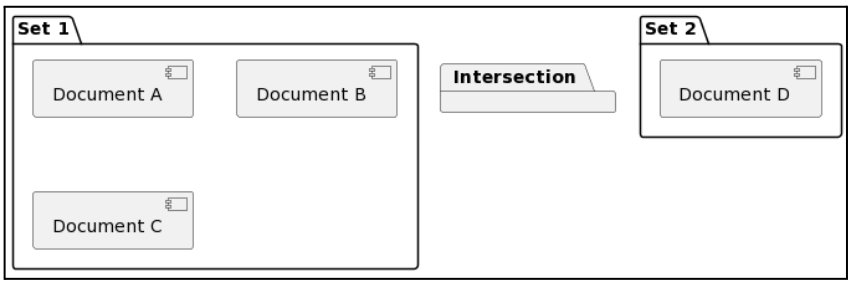


Рисунок 3.6 – Діаграма (Intersection)

3.7 Модифікація файлового пошуку

Припустимо в нас великий об’єм даних з яким не можливо працювати через оперативну пам’ять, тоді ми будемо використовувати метод розміщеня індексів у файлову систему на диску, тож тепер в нас постає проблема пошуку об’єкту по файловій системі.

Отже, однією з ключових проблем є забезпечення швидкого і точного доступу до інформації. Існуючі методи повнотекстового індексування як ми вже з’ясували є неефективними при роботі з великими обсягами даних, оскільки вони часто вимагають значного часу для пошуку та фільтрації інформації.

Цю проблему ми можемо вирішити лінійним пошуком або бінарним та навіть у випадку використання хеш таблиць – пошук буде лінійно сповільнюватись відносно зростаючого об’єму файлів, тож треба розробити більш ефективний метод для роботи із пошуком у файловій системі.

Метод використання префіксного дерева (trie) для модифікації сховища повнотекстових індексів пропонує елегантне рішення цієї проблеми. Префіксні дерева дозволяють швидко знаходити слова або фрази за їхніми початковими символами, що є ідеальним для пошуку у великих текстових базах даних.

Розглянемо більш детально процес імплементації та роботи методу.

Спочатку створюється префіксне дерево, де кожен вузол представляє окремий символ у слові або фразі. Вузли пов'язані таким чином, що кожен шлях від кореня до вузла відображає унікальне слово або частину слова.

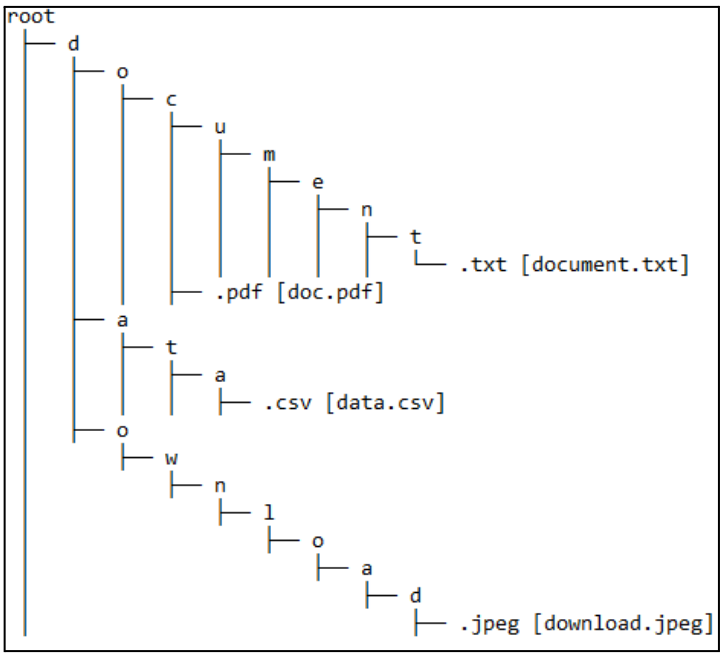
Це дерево інтегрується із системою сховища даних, що дозволяє індексувати великі обсяги текстової інформації.

Кожен документ або запис у повнотекстовому індексі проходить через процес індексації, де його текст розбивається на слова або фрази, які потім додаються у префіксне дерево. Індекси або підіндекси зберігаються в вузлах дерева для швидкого доступу до відповідних документів[11].

Інтеграція префіксного дерева з сховищем повнотекстових індексів пропонує значне покращення в швидкості та ефективності пошуку індексованих файлів, особливо при роботі з великими обсягами даних. Це рішення не тільки

оптимізує пошук та індексацію даних, але й забезпечує покращення швидкості знаходження інформації через обхід у н-етерацій.

На рисунку 3.7 зображене схематичний вигляд файлової системи після створення її за нашим методом.



Рисунку 3.7 – Схематичний вигляд файлової системи

Підсумовуючи, метод використання префіксного дерева для модифікації сховища повнотекстових індексів є незвичним але високоефективним підходом, який пропонує ряд переваг у порівнянні з іншими методами пошуку в файлових системах.

4 РЕАЛІЗАЦІЯ НАУКОВОЇ НОВИЗНИ

4.1 Аналіз методу фрагментації тексту

Великі текстові документи ставлять виклики перед системами повнотекстового пошуку, оскільки стандартне індексування кожного слова може бути неефективним та повільним. Потрібен метод, який зменшує кількість даних для аналізу при кожному пошуковому запиті і зберігає достатній контекст навколо кожного слова для високоякісного пошуку.

Щоб адресувати цю потребу, було запропоновано розділити текст на фрагменти. Кожен фрагмент містить групу слів, і такий підхід дозволяє зменшити обсяг тексту, який потрібно аналізувати при конкретному пошуковому запиті. Фрагменти можуть бути оптимізовані за розміром, залежно від типової довжини пошукових запитів.

Весь текст документа спочатку проходить через процес токенізації, де він розділяється на слова або фрази. Ці токени потім групуються у чанки фіксованого розміру, що формують фрагменти. Кожен фрагмент індексується з позначенням його початкової та кінцевої позиції у документі.

Індексовані фрагменти зберігаються у системі управління базою даних або в спеціалізованій структурі даних[12]. Це дозволяє швидко вилучати фрагменти, що містять конкретні слова або фрази відповідно до пошукових запитів.

При пошуку слова або фрази система спочатку визначає, які фрагменти містять ці слова. Завдяки тому, що кожен фрагмент має відомі початкову та кінцеву позиції, пошук може бути швидко зосереджений лише на відповідних частинах тексту. Загальна стратегія роботи зображена на рисунку 4.1.

Вона включає в себе:

- лексер (lexer): компонент, який відповідає за перетворення вхідного тексту на серію токенів (слів або фраз);
- індекс (index): компонент, який управляє індексацією документів, створенням фрагментів і їх зберіганням;

– сховище даних (storage): база даних або система зберігання, де тримаються всі токени, документи і фрагменти.

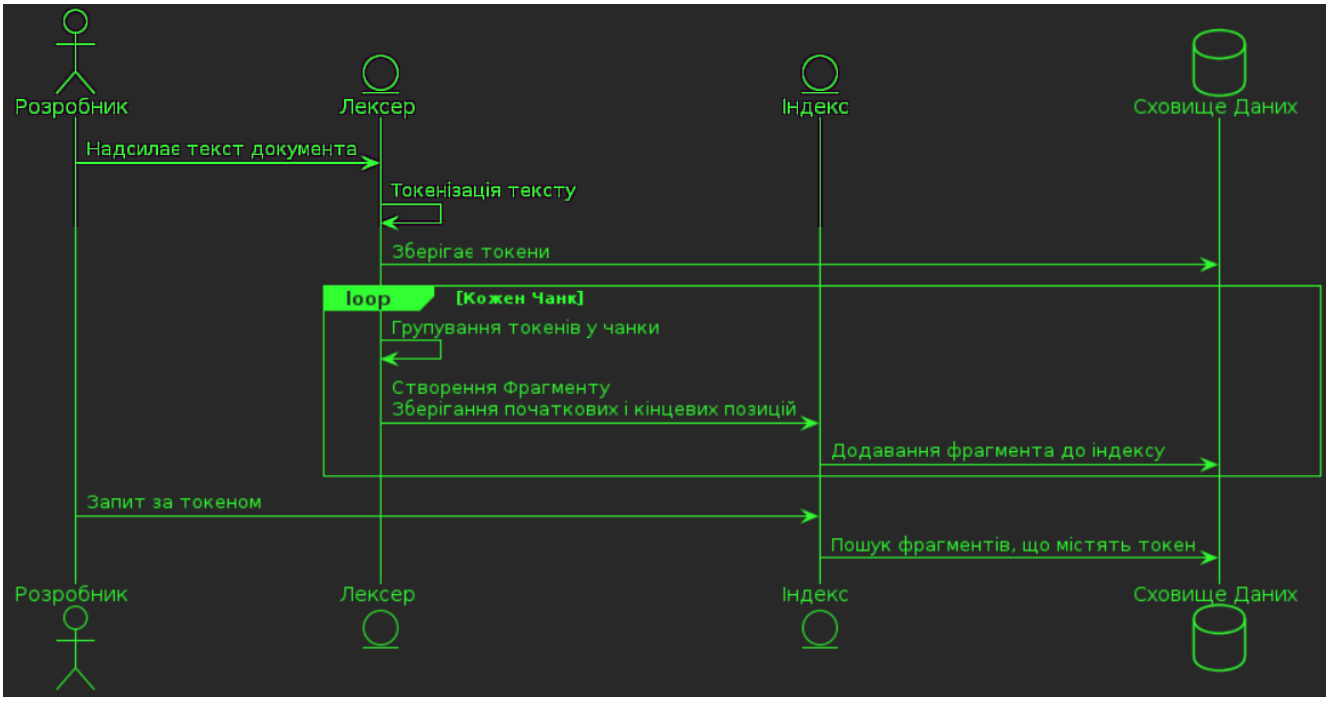


Рисунок 4.1 – Принцип роботи методу фрагментації тексту

4.2 Аналіз результатів фрагментації

Результати відпрацювання бенчмарку BuildIndexBenchmark по вимірюванню часу наведено на рисунку 5.2, у ньому ми більше зосередимось на метриках ресурсів.

Method	Query	Mean	Error	StdDev	Ratio	Gen 0	Allocated
SimpleSearch	Books	1,830,737.89 ns	815,112.256 ns	211,682.064 ns	1.000	15.6250	69,265 B
FragmentsSearch	Books	89.08 ns	15.276 ns	3.967 ns	0.000	0.0421	176 B
FullTextIndexedSearch	Books	98.54 ns	1.307 ns	0.202 ns	0.000	0.0497	208 B
WithPositionsIndexedSearch	Books	99.36 ns	1.501 ns	0.232 ns	0.000	0.0592	248 B
SimpleSearch	assessment	1,914,779.69 ns	28,848.025 ns	4,464.264 ns	1.000	15.6250	69,265 B
FragmentsSearch	assessment	88.80 ns	9.387 ns	1.453 ns	0.000	0.0421	176 B
FullTextIndexedSearch	assessment	87.05 ns	4.832 ns	1.255 ns	0.000	0.0421	176 B
WithPositionsIndexedSearch	assessment	88.88 ns	6.201 ns	1.610 ns	0.000	0.0516	216 B
SimpleSearch	where	1,623,642.62 ns	14,599.704 ns	3,791.497 ns	1.000	15.6250	69,265 B
FragmentsSearch	where	6,867.84 ns	150.968 ns	39.206 ns	0.004	0.1068	464 B
FullTextIndexedSearch	where	8,237.68 ns	302.196 ns	78.479 ns	0.005	0.0916	400 B
WithPositionsIndexedSearch	where	272.12 ns	12.966 ns	3.367 ns	0.000	0.1259	528 B

Рисунок 4.2 – Аналіз часу обробки

Після аналізу відмітимо що запити "Books" та "Assessment" обробляються надзвичайно швидко (менше 100 ns), що свідчить про ефективність фрагментації при обробці стандартних або коротких запитів. Навіть для більш складного запиту "Where" час виконання значно менший порівняно з іншими методами, крім методу з індексацією позицій.

Порівняно з іншими методами, такими як SimpleSearch та FullTextIndexedSearch, FragmentsSearch використовує значно менше пам'яті, що робить його оптимальним для систем з обмеженими ресурсами. Незважаючи на зростання часу виконання для складнішого запиту "Where", цей метод все ще забезпечує прийнятну продуктивність, що підтверджує його здатність масштабуватися до більш великих або складних наборів даних.

Також висока продуктивність FragmentsSearch у швидкості виконання та ефективності використання пам'яті демонструє, що фрагментація тексту на чанки є вдалим рішенням для оптимізації пошукових систем, особливо коли потрібна швидка відповідь на пошукові запити.

BuildIndexBenchmark спрямований на тестування процесу створення індексу. Цей бенчмарк оцінює, наскільки ефективно система може обробляти вхідні дані та створювати індекс, який буде використовуватися для швидкого пошуку.

Метрики:

- час побудови: час, необхідний для створення повного індексу з набору вхідних даних;
- оптимізація пам'яті: використання пам'яті під час побудови індексу;
- масштабованість: здатність системи ефективно масштабуватися зі збільшенням обсягу даних.

Отже засновуючись на результатах дослідження – варто розробити додатково і рекомендації для розробників що до використання кожного з метоідв.

Для систем з обмеженими ресурсами: FragmentsSearch є оптимальним вибором. Він забезпечує швидкий пошук з мінімальним навантаженням на пам'ять, ідеально підходячи для систем з обмеженою оперативною пам'яттю або дисковим простором[13].

Для задач зі складними пошуковими запитами: WithPositionsIndexedSearch рекомендується для систем, де необхідна висока точність і швидкість пошуку, зокрема для складних запитів. Він підходить для великих датацентрів або спеціалізованих додатків, де використання пам'яті не є критичним фактором.

Для загального використання і рівномірної продуктивності: FullTextIndexedSearch може слугувати золотою серединою, забезпечуючи швидкість і ефективність без надмірного споживання ресурсів. Цей метод підходить для широкого спектру застосунків, від корпоративних до веб-сервісів.

ВИСНОВКИ

У цій роботі ми провели глибокий аналіз предметної області повнотекстових методів, їх методів та їх використання. Дарі нами було розглянуто методи роботи із повнотекстовими індексами.

В рамках нашого дослідження, ми зосередилися на аналізі існуючих методів роботи з повнотекстовими індексами. Ми вивчили, як повнотекстові індекси організують дані для забезпечення швидкого доступу до інформації та як вони можуть бути використані для обробки складних запитів, що включають комбінації ключових слів, фраз та логічних умов.

Було порівняно різні методи, такі як простий лінійний пошук, булевий пошук та пошук за допомогою регулярних виразів. Кожен з цих методів має свої переваги та недоліки, проте ми зосередили увагу на їх ефективності при роботі з великими обсягами даних. Зокрема, ми виявили, що булевий пошук є найбільш підходящим для великих дата-сетів завдяки своїй швидкості, точності та ефективності.

Наш аналіз підтвердив, що у сучасних умовах, де обсяги даних постійно зростають, потребується комплексний та ефективний підхід до пошуку інформації. Методи, які ми розглядали, демонструють різні рівні ефективності, але всі вони мають обмеження при роботі з великими наборами даних.

Завершуючи наш аналіз, було зосереджено увагу на методі використання префіксних дерев для модифікації файлових систем. Цей метод виявився надзвичайно ефективним у покращенні швидкості пошуку індексованих файлів, особливо у великих обсягах даних.

Загалом щоб остаточно визначити найефективніший метод пошуку – та більш детально дослідити метод префіксних дерев – необхідно буде провести додаткові дослідження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Аналіз предметної області \ A Guide to Choosing the Right .NET Object Mapper? 2023р., URL: https://learn.microsoft.com/ru-ru/sql/relational-databases/search/full-text-search?view=sql-server-ver16 (дата звернення:05.04.2024).

2. Аналіз існуючих рішень, URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/full-text-queries.html (дата звернення: 05.04.2024).

3. What is Full-Text Search and How Does it Work?, URL: https://www.mongodb.com/resources/basics/full-text-search#:~:text=Full-text%20search%20involves%20reviewing,PDFs%2C%20and%20more. (дата звернення: 10.04.2024).

4. Enhancing Database Text Search., URL: https://medium.com/@havus.it/enhancing-database-search-full-text-search-fts-in-mysql-1bb548f4b9ba (дата звернення: 10.04.2024).

5. Full-Text Search (FTS) in MySQL, URL: https://medium.com/@havus.it/enhancing-database-search-full-text-search-fts-in-mysql-1bb548f4b9ba (дата звернення: 10.04.2024).

6. Аналіз Full-Text \ HigLabo.Mapper, Creating Fastest Object Mapper in the World with Expression Tree, 2024р., URL: https://www.codeproject.com/Articles/5275388/HigLabo-Mapper-Creating-Fastest-Object-Mapper-in-t (дата звернення: 10.04.2024).

7. Free Public Data Sets For Analysis, URL: https://www.tableau.com/learn/articles/free-public-data-sets (дата звернення: 17.04.2024).

8. Аналіз reflection ammit \ Comparison of Object Mapper Libraries, Jon Dodd, 2021р., URL: https://dev.to/jdinnovensa/comparison-of-object-mapper-libraries-gm2 (дата звернення: 17.04.2024).

9. FastCompileExpression бібліотека \ Fast Compiler for C# Expression Trees and the lightweight LightExpression alternative.? 2024р., URL: https://github.com/dadhi/FastExpressionCompiler (дата звернення: 17.04.2024).

10. Optimization Factors in Modeling and Testing Hardware and Semiconductor Defects by Dynamic Discrete Event Simulation \ Arabian, J. H., Видавництво: ХНУРЕ, 2009р., URL: https://openarchive.nure.ua/entities/publication/be918fba-8755-4d34-be6d-fc1464089d77 (дата звернення: 05.05.2024).

11. Falatiuk H., Shirokopetleva M., Dudar Z. Investigation of Architecture and Technology Stack for e-Archive System. 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv,Ukraine,8-11October2019.2019.,URL: https://doi.org/10.1109/picst47496.2019.9061407 (дата звернення: 05.05.2024).

12. DATA EXCHANGE MODEL IN THE INTERNET OF THINGS CONCEPT / I. Afanasieva et al. Telecommunications and Radio Engineering. 2019. Vol. 78,no. 10.P. 869–878. URL: https://doi.org/10.1615/telecomradeng.v78.i10.30 (date of access: 05.05.2024).

13. Аналіз результатів бенчмаркіунгу URL: https://github.com/dotnet/BenchmarkDotNet (date of access: 05.05.2024)