УДК 004.658

В. Мухін, Я. Корнага, Л. Снєгірєв
Національний технічний університет України
“Київський політехнічний інститут”,
кафедра обчислювальної техніки

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ МЕХАНІЗМІВ ПОШУКУ В БАЗАХ ДАНИХ НА ОСНОВІ К-ДЕРЕВ

Запропоновано новий вид дерев для побудови індексів таблиць баз даних: К-дерева. Проведено експериментальне порівняння з В+-деревами з дослідження ефективності використання К-дерев для пошуку в базах даних.

Ключові слова: пошук даних, індекси, В+-дерева, К-дерева.

There are suggested the new type of trees for database tables indexes forming: K-trees. The experimental researches for the B-trees and K-trees features are performed. The researches show that the K-trees ensure the improvement of the data search.

Key words: data search, indexes, B+-trees, K-trees.

Вступ

У базах даних важливим елементом є пошук даних у таблицях, які можуть мати велику кількість рядків, що зберігаються в довільному порядку, та їх пошук за заданим критерієм шляхом послідовного перегляду таблиці рядок за рядком може займати багато часу. Індекс формується зі значень одного чи декількох стовпців таблиці і вказівників на відповідні рядки таблиці та дає змогу шукати рядки, що задовольняють критерій пошуку. Робота з використання індексів прискорюється насамперед за рахунок того, що індекс має структуру, оптимізовану під пошук: наприклад, збалансованого дерева [1, 2].

Зазвичай, що більше індексів, то більша продуктивність запитів до бази даних. Однак за надмірного збільшення кількості індексів падає продуктивність операцій зміни даних (запис, модифікація, видалення), збільшується розмір БД, тому до додавання індексів слід ставитися обережно [1–3].

Базовим апаратом для пошуку даних у БД є В-дерева. В основу цього механізму покладено такі ідеї. По-перше, оскільки йдеться про структури даних у зовнішній пам'яті, загальний час доступу до якої визначається переважно не обсягом послідовно розташованих даних, а часом підведення магнітних головок, то вигідно отримувати за одне звернення до зовнішньої пам'яті якмога більше інформації, враховуючи при цьому необхідність економного використання основної пам'яті. За сформованого підходу до організації основної пам'яті у вигляді набору сторінок однакового розміру природно вважати саме сторінку одиницею обміну із зовнішньою пам'яттю. По-друге, бажано забезпечити таку пошукову структуру у зовнішній пам'яті, з використанням якої пошук інформації за будь-яким ключем вимагає заздалегідь відомого числа обмінів із зовнішньою пам'яттю [4, 5].

Для створення індексів у сучасних базах даних використовуються В+-дерева, але під час роботи з ними виникають проблеми з компактністю, адже вузли у них заповнені не менше ніж на 1/2. Потрібно вирішувати проблему з компактнішим заповненням та відповідно з меншою кількістю рівнів дерева [6, 7].

Опис нового дерева та його порівняння з В+-деревом

Створимо новий тип дерева та назвемо їх К-дерево. Воно містить всі характеристики аналогічно В+-дереву та відрізнялося лише стратегією розщеплення та об'єднання вузлів.

Опишемо властивості К-дерева порівняно з B+-деревом, як зображено в табл. 1, для кращого розуміння побудови пошуку та механізмів зчеплення та розчеплення вузлів.

Таблиця 1

Властивості дерев пошуку, де n – це параметр дерева, що набуває значення не менше 2, а переважно від 50 до 2000

Властивість	B+-дерево	К-дерево
Кількість елементів, що містить корінь дерева	від 1 до 2n елементів	від 1 до 3n елементів
Кількість елементів, що містять вузли дерева (крім кореня)	від n до 2n елементів	від 3/2n до 2n елементів
Кількість нащадків у вузлів (крім листових), які містять m елементів	m+1 нащадків	m+1 нащадків
Листові сторінки дерева знаходяться	на одному рівні	на одному рівні
Розчеплення вузлів відбувається шляхом	розбиття вузла навпіл на два нові вузли	розбиття вузла з двома його сусідами на чотири нових вузла
Зчеплення вузлів відбувається шляхом	з'єднання двох вузлів в один новий вузол	з'єднання трьох вузлів у два нові вузли

Отже, відмінність між B+-деревом та К-деревом полягає в відмінностях пунктів 1, 2, 5 та 6 характерних властивостей. В К-дереві, відповідно до першого пункту характерних особливостей, більша кількість елементів в корені вузла, що потрібно для проведення операцій з'єднання та роз'єднання вузлів в К-дереві. А за другим пунктом різниця в мінімальному ступені наповненості вузла: в К-дереві, на відміну від B+-дерева, вона становить 3/4 вузла. Це дає змогу економити місце на жорсткому диску та збільшити швидкість доступу до інформації за рахунок того, що зчитування вузлів К-дерева, як і B+-дерева, відбувається за блоками, і відповідно від наповненості блока залежить кількість інформації, яку ми отримаємо.

А тепер опишемо детальніше операції додавання і видалення елементів в К-дереві. Як видно з таблиці порівняння , в кожному вузлі К-дерева можуть зберігатися 2n ключів, а в корені їх може бути 3n. Перед тим, як під час вставки ми спустимося до нащадка, ми перевіряємо, чи він повний. Якщо це так, то ключі, що знаходяться в нащадку і двох суміжних до нього вузлах, об'єднуються і перерозподіляються. Якщо два суміжні вузли також заповнені, то додається новий вузол. Так, ми отримуємо вже чотири вузли, кожен з яких повний на 3/4.

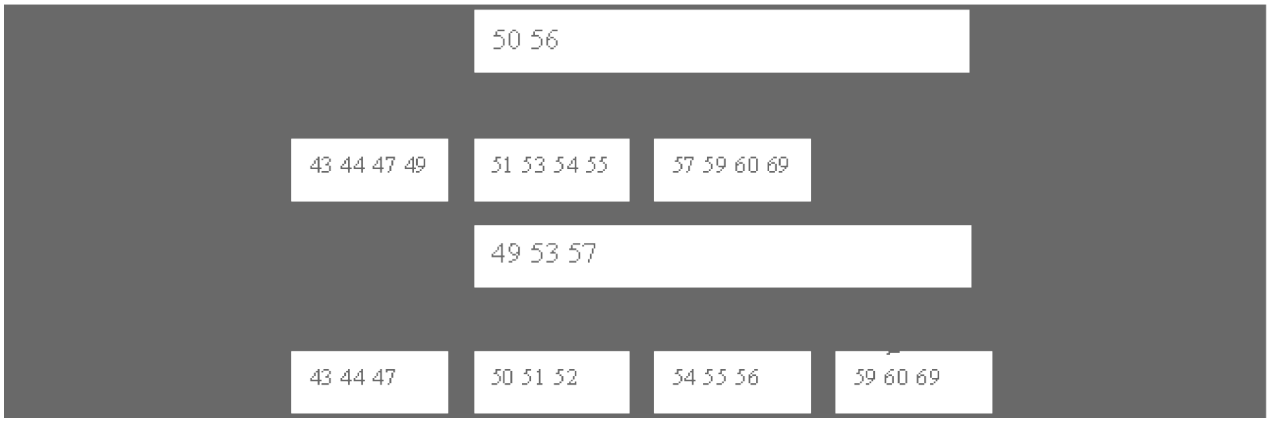

Рис. 1. Процедура розбиття вузлів К-дерева за повністю заповнених вузлів та вставки ключа 52

Перед тим, як під час видалення спуститися до нащадка, ми перевіряємо, чи не повний чи він на 1/2. Якщо це так, ключі нащадка і двох суміжних вузлів об'єднуються і перерозподіляються так, як показано на рис. 1.

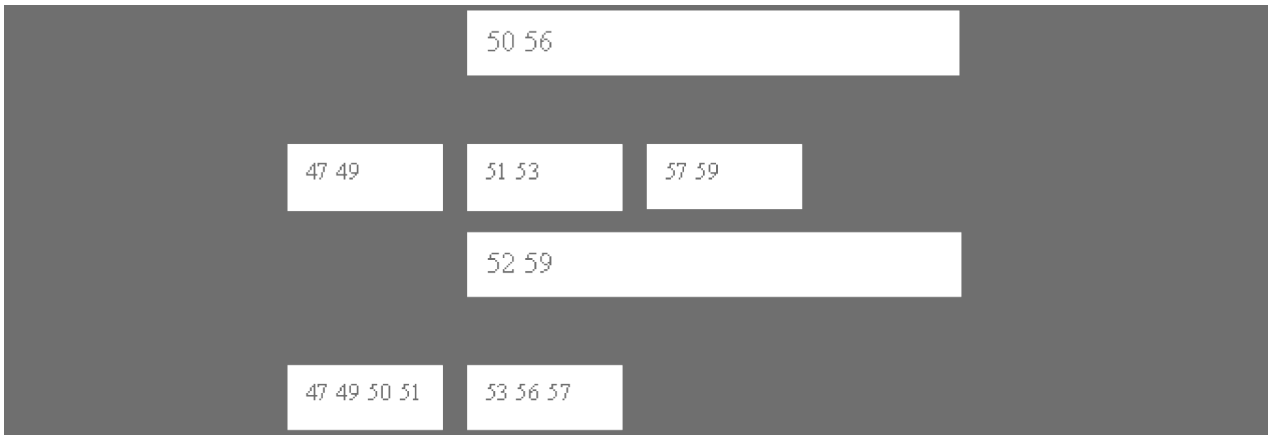

Рис. 2. Процедура з'єднання вузлів К-дерева у разі заповнення вузлів на 1/2 та вставки ключа 52

Якщо два суміжні вузли самі повні наполовину, вони зливаються в два вузли, кожен з яких повний на 3/4 – так, як це показано на рис. 2.

Тобто ми опиняємося посередині між наповненістю на 1/2 і повною наповненістю, що дає нам змогу очікувати однакового числа вставок і вилучень.

У кореневому вузлі зберігаються 3n ключів. Якщо під час вставки виявиться, що корінь повний, ми розподіляємо ключі за трьома новими вузлами, кожен з яких повний на 3/4. Це збільшує висоту дерева на одиницю.

Під час видалення ми досліджуємо нащадків. Якщо є тільки три нащадки і вони повні наполовину, переносимо їх вміст в корінь, у результаті чого висота дерева зменшується.

Іншими словами можна сказати, що ми збираємо три вузли, а потім розділяємо їх. Якщо нам потрібен додатковий вузол, ми ділимо на чотири вузли. Якщо вузол потрібно видалити, ми поділяємо на два вузли. Симетрія операцій дає змогу використовувати при реалізації вставки і видалення одні й ті самі об'єднавчі та роз'єднавчі функції під час програмної реалізації К-дерев, що дає змогу економити час на написання програмного коду та його редагування.

Подамо опис К-дерева мовою опису графів (дерев). Позначимо G = (V,E) як множину всіх дерев. Тоді V – це непуста множина всіх вершин дерева, а E – множина пар вершин, що називаються ребрами. Як V, так і E є скінченними множинами. До множини V входять вершини $v_1 \ldots v_n$, а до множини E – ребра $e_k=(v_i,v_j)$. Для прискорення пошуку за допомогою дерев потрібно створити матрицю, яка б описувала розміщення блоків з вузлами на жорсткому диску, тобто фактично показувала, яку наступну вершину зчитувати з нього. Для такої побудови використаємо матрицю списку дуг, тобто трирядковий масив, в якому в першому рядку зберігатимуться вершини v_i, з яких починається дуга, в другому рядку – вершини v_j, в яких закінчується дуга, а в третьому рядку – вага дуг. Вагу дуг визначають за номером по порядку та зліва направо.

Наведемо приклад матриці дуг для зберігання вершин дерева, як показано на рис. 3.

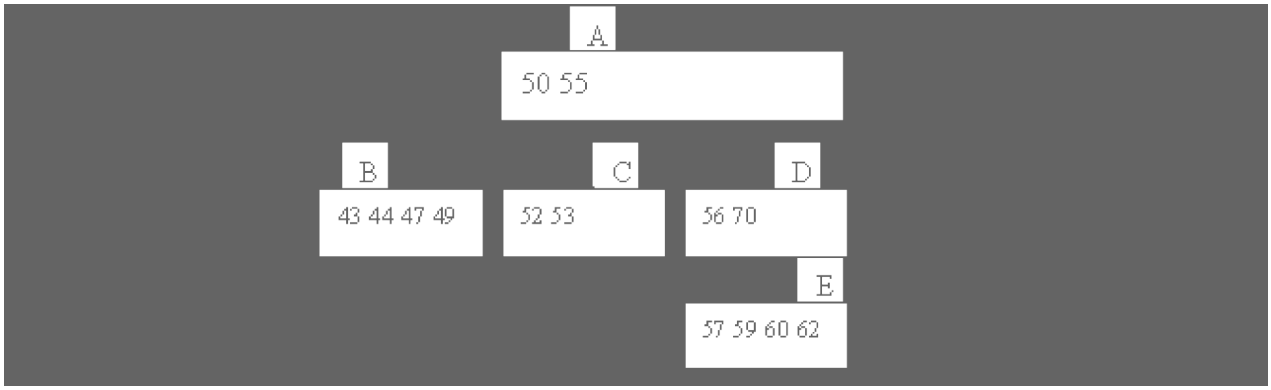

Рис. 3. Приклад К-дерева для опису матрицею дуг

Таблиця 2

Матриця дуг для зберігання опису розміщення дерев на жорсткому диску

Перша вершина дуги	A	A	A	D
Друга вершина дуги	B	C	D	E
Номер дуги	1	2	3	1

Отже, дуги з вершини A було пронумеровано так, як це потрібно для процесу пошуку, і чітко видно, що якщо ми шукаємо ключ 60, то перевіряємо два числа кореня дерева і йдемо по третій дузі з вершини A вправо донизу до вершини D, а звідти, своєю чергою, до вершини E, де і знаходимо цей ключ.

За цим методом можна пришвидшити знаходження вершин дерев та відповідно пошук загалом.

Аналітичне оцінювання параметрів пошуку на основі К-дерев

Для порівняння швидкості пошуку на основі B+-дерев та К-дерев визначимо кількість елементів, якими можна заповнити ніжній рівень дерева за максимального, середнього та мінімального заповнення. Якщо параметр n дорівнює 50, k – кількість елементів (записів) в дереві, m – рівень (висота) дерева, то формула розрахунку кількості елементів в дереві для відповідного рівня матиме вигляд:

– для B+-дерев

$$k_{Bmax}=(2n+1)^m 2n, \quad (1)$$

$$k_{Bave}=(n+1)(3/2n+1)^{m-1}3/2n, \quad (2)$$

$$k_{Bmin}=2(n+1)^{m-1}n, \quad (3)$$

– для К-дерев

$$k_{Kmax}=(3n+1)(2n+1)^{m-1}2n, \quad (4)$$

$$k_{Kave}=(3/2n+1)(7/4n+1)^{m-1}7/4n, \quad (5)$$

$$k_{Kmin}=2(3/2n+1)^{m-1}3/2n, \quad (6)$$

та формула знаходження висоти дерева при відомій кількості елементів k матиме вигляд:

– для B+-дерева

$$m_{Bmax}=\log_{(2n+1)}(k/2n), \quad (7)$$

$$m_{Bave}=\log_{(3/2n+1)}(k/(3/2n(n+1)))+1, \quad (8)$$

$$m_{Bmin}=\log_{(n+1)}(k/2n)+1, \quad (9)$$

– для К-дерева

$$m_{Kmax}=\log_{(2n+1)}(k/2n(3n+1)))+1, \quad (10)$$

$$m_{Kave}=\log_{(7/4n+1)}(k/(7/4n(3/2n+1)))+1, \quad (11)$$

$$m_{Kmin}=\log_{(3/2n+1)}(k/3n)+1. \quad (12)$$

Основною умовою знаходження висоти дерева є те, що m завжди ціле число, тому його завжди потрібно округляти до цілого, причому завжди у більший бік.

Після визначення параметрів m для дерев визначимо час пошуку за індексом. Позначимо час пошуку за B+-деревом – $T_{ПВ}$ та відповідно за К-деревом – $T_{ПК}$. Час пошуку ділимо на дві частини: час $T_Д$, за який відбувається пошук та зчитування відповідного вузла дерева на жорсткому диску, зчитування його в оперативну пам'ять та час T_B пошуку у цьому вузлі.

Для B+-дерев та К-дерев час зчитування вузла та час пошуку в вузлі є однаковими; відрізнятися може тільки пошук у корені К-дерева, але ця відмінність настільки мала, що нею можна знехтувати.

Отже, час пошуку даних за допомогою дерев визначають за формулами

$$T_{ПВ}=m_B(T_Д+T_B), \quad (13)$$

$$T_{ПК}=m_K(T_Д+T_B). \quad (14)$$

Відповідно до цих формул відношення часу пошуку даних за допомогою B+-дерев до часу пошуку даних за допомогою К-дерев відрізнятиметься за значенням параметра m. Позначимо коефіцієнт відмінності через s, а формула його знаходження виглядатиме так:

$$\frac{T_{ПВ}}{T_{ПК}}=\frac{m_B}{m_K}=s. \quad (15)$$

Знаючи коефіцієнт відмінності s ми можемо визначити, у скільки разів пошук за К-деревами буде швидший від пошуку за B+-деревами та наскільки зміниться час.

Розрахуємо конкретний час $T_П$ пошуку для індексів на основі дерев для таблиць з різною кількістю записів в них.

Таблиця 3

Час пошуку в мс для різного числа записів та середньої наповненості дерева

Число записів	B+-дерева	K-дерева
5000	24	12
10000	24	24
50000	24	24
100000	24	24
500000	36	24
1000000	36	36
5000000	36	36
10000000	36	36
50000000	48	36

Вважаємо, що час $T_Д$, за який відбувається пошук та зчитування відповідного вузла дерева на жорсткому диску, дорівнює 10 мс, а час $T_В$ пошуку у цьому вузлі – 2 мс, підставимо його до формул (13), (14) для розрахунку та запишемо отримані дані до табл. 3 та побудуємо графік.

На рис. 4 зображено графік залежності часу пошуку від числа записів у таблиці, за якою проводиться пошук та відповідно по осі x проставлено кількість записів в таблиці, за якими проводився пошук, а по осі y – час пошуку в мс.

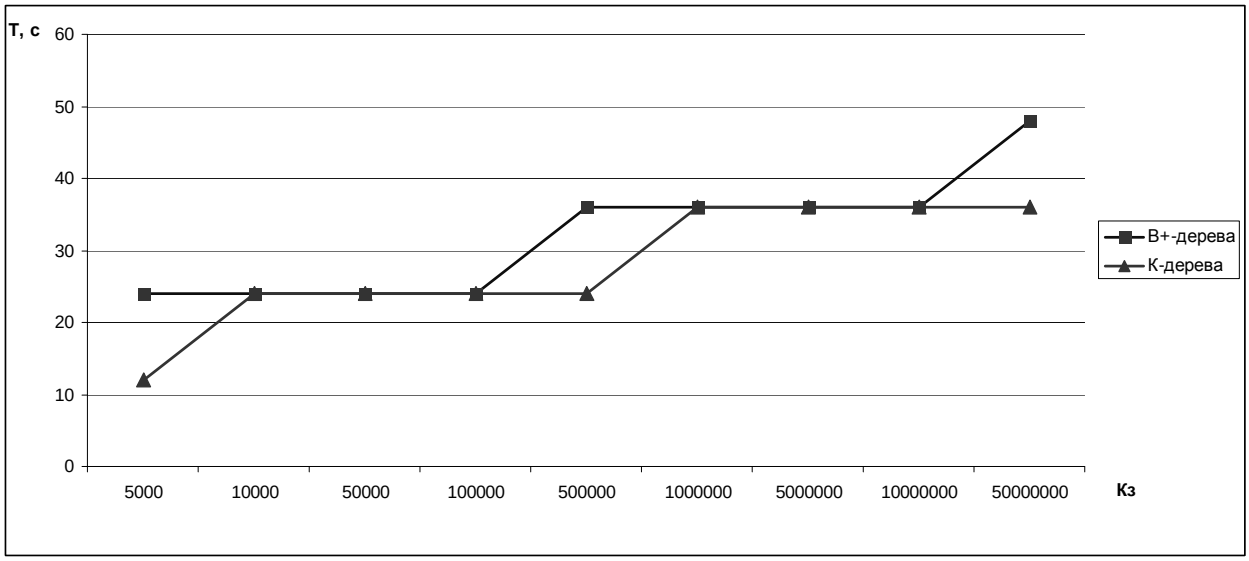

Рис. 4. Залежність часу пошуку від числа записів для різних методів

Висновок

Отже, на невеликій кількості записів К-дерева зменшують час пошуку на відповідних проміжках, яке становить в середньому 9 %, але зі збільшенням кількості записів час пошуку за допомогою індексів, основаних на К-деревах, значно зменшуватиметься порівняно з B+-деревами, що прискорить роботу самої бази даних та дасть змогу витрачати ресурси серверів, на яких розміщені БД, з більшою ефективністю.

1. Реймонд Ф. Базы данных. Проектирование и разработка / Ф. Реймонд, Д. Джон, В.С. Крейг. – М.: НТ Пресс, 2007. – 592 с. 2. Реймонд Ф. Проектирование и разработка баз данных. Визуальный подход / Ф. Реймонд, Д. Джон, В.С. Крейг. – М.: НТ Пресс, 2007. – 592 с. 3. Ролланд Ф. Основные концепции баз данных / Ф. Ролланд. – М.: Вильямс, 2008. – 256 с. 4. Мишра С. Секреты Oracle SQL / С. Мишра, А. Бьюли. – М.: Симбо, 2009. – 368 с. 5. Бьюли А. Изучаем SQL / А. Бьюли. – М.: Символ-Плюс, 2007. – 312 с. 6. Макдональд К. Oracle PL/SQL для профессионалов. Практические решение / К. Макдональд, Х. Кац. – М.: ДиаСофтЮп, 2005. – 560 с. 7. Шварц Б. Оптимизацыя производительности баз данных / Б. Шварц, П. Зайцев, В. Ткаченко. – М.: Символ-Плюс, 2010. – 832 с.