УДК 681.3.06

ВДОСКОНАЛЕННЯ ОРГАНІЗАЦІЇ ДАНИХ ОБ'ЄКТНИХ ПРИКЛАДНИХ ПРОГРАМНИХ СИСТЕМ ЯК МЕТОД ПІДВИЩЕННЯ ЇХ ЖИТТЄЗДАТНОСТІ

І.В. Федоров

Інститут програмних систем НАН України
03187, Київ, проспект Академіка Глушкова, 40,
тел.: +380 (44) 266 1540; Факс: 380 (44) 266 6263;
e-mail: fedorov@rgdata.com.ua

У роботі розглянуто актуальну проблему забезпечення життєздатності прикладних програмних систем. Для більш економного і збалансованого використання зовнішньої пам'яті пропонується нетрадиційна структура даних, яка дозволяє підвищити продуктивність дерева, а, отже, забезпечити більш ефективне індексування об'єктів та ефективну форму збереження даних при можливості роботи з ними.

In the report the actual problem of maintenance of viability of applied program systems is considered. For more economical and balanced use of external memory the nonconventional structure of the data which allows increasing efficiency of a tree is offered, and, means, to provide more effective indexing objects and the effective form of preservation of the data at an opportunity of job with them.

Вступ

На протязі останніх років у галузі створення і супроводження прикладних програмних систем спостерігається великий інтерес до методів забезпечення життєздатності програмних систем, адже однією з найбільш актуальних проблем, які виникають при створенні та супроводженні автоматизованих систем (АС) є проблема забезпечення стійкості цих систем до змін навколишнього середовища та змін вимог їх користувача, тобто їх життєздатності [1].

Методи забезпечення життєздатності програмних систем включають такі класи методів:

- забезпечення адаптивності програмних систем;
- забезпечення ефективного реінжинірингу програмних систем;
- забезпечення рефакторингу програмних систем.

На сьогоднішній день адаптивність являється чи не одним з найголовніших критеріїв життєздатності програмних систем. На сьогодні програмні системи не просто впливають на діяльність підприємств, прискорюючи і оптимізуючи бізнес-процеси, вони стають невід'ємною частиною цих процесів. Для того, щоб програмні системи давали максимальну віддачу від вкладених у них коштів, розвиток бізнесу і вдосконалення інфраструктури, на яку він опирається, повинні відбуватися паралельно.

Однак на практиці бізнес-процеси, як правило, швидко змінюються, тоді як перегляд концепції побудови програмних систем відбувається більш інертно. Степінь застосування програмних систем до поточних задач багато в чому визначається не реальними потребами бізнесу, а досягненим рівнем розвитку технологій. Результатом цього є неоптимальні інвестиції, неефективне використання ресурсів програмних систем, складність перенастроювання систем у відповідності з новими вимогами, збільшення витрат на експлуатацію і т.д. Поряд з високою вартістю і низькою ступінню використання наявних IT-ресурсів однією із ключових проблем є неспроможність інфраструктури інформаційних технологій більшості компаній до швидких змін. Не рідко доводиться відкладати впровадження нових застосувань із-за проблем з продуктивністю мережі. За останні роки в багатьох компаніях кількість програмного забезпечення досягла критичної маси і перетворилася у погано керовані об'єкти. Між тим бізнес ряду компаній зв'язаний з використанням критично важливих застосувань, зупинка яких може завершитися фатально. В цих умовах вищеописане направлення розвитку програмних систем стає одним із визначаючих, як для компаній, так і для тих, хто розроблює її елементи.

Реінжиніринг програмної системи проводиться з метою дослідження і зміни початкової системи для відновлення її у новій формі та наступної реалізації нової форми. Зміну програмної системи доцільно проводити у випадку зміни вимог до системи, добавленні функціональності. Забезпечення реінжинірингу програмної системи дозволить підвищити в цілому зрозумілість системи, як для супроводження, так і для нової розробки. Окрім цього це дасть можливість справитися зі складністю системи, створити альтернативні представлення системи, відновити загублену інформацію про систему, розкрити побічні ефекти, спростити перевикористання.

Забезпечення реінжинірингу програмної системи вимагає організації процесу аналізу існуючої системи для визначення компонент системи і їх взаємодій, створення представлення про систему на високому рівні абстракції. Далі необхідно виконати перетворення системи від одного представлення до іншого, на тому ж рівні абстракції, зберігаючи зовнішню поведінку системи.

Рефакторинг представляє собою процес такої зміни програмної системи, при якому не міняється зовнішня поведінка коду, але поліпшується його внутрішня структура. Забезпечення рефакторингу програмної системи дозволить поліпшити композицію програмного забезпечення та полегшити розуміння програмного забезпечення. По мірі розвитку програми весь час вносяться зміни, які обумовлені поточною необхідністю. Адже часто зміни вносять програмісти, які не до кінця розуміють архітектуру програмного забезпечення, у цілому. Тому постійно код становиться менш структурованим і розбиратися у ньому все важче. Рефакторинг приводить до більш глибшого розуміння того, як працює програма. Таке розуміння значно прискорює процес програмування, а отже дозволить набагато швидше розробляти код і знаходити помилки.

Ефективність названих класів методів багато в чому залежить від організації даних в автоматизованих системах. Вдосконалення методів організації даних дозволить забезпечити ефективну застосовність до автоматизованих систем названих класів методів для забезпечення їх життєздатності, досягти необхідних показників ефективності функціонування автоматизованих систем та досягти мінімальних втрат при їх створенні та супроводженні.

Автоматизована система (АС) – система, що реалізує інформаційну технологію у сфері управління за спільної роботи управлінського персоналу і комплексу технічних засобів. Вона призначена для автоматизованого збирання, реєстрації, збереження, пошуку, оброблення та видачі інформації за запитами користувачів (управлінського персоналу). Це відбувається на основі використання економіко-математичних методів, моделей, ЕОМ і засобів комунікації. Автоматизована система реалізує принципово нову платформу управління, що ґрунтується на інтеграції управлінської інформації за допомогою механізму загального інформаційного зв’язку даних, які включають в оброблення з метою здобуття інформації для управління.

Автоматизована система має забезпечувати:

1) постійне спостереження за поточним станом об’єкта управління та його характеристик;

2) адаптації, тобто пристосування до прийнятої практики бізнесу та модифікації, якщо така практика змінюється;

3) підтримку професійної діяльності управлінських працівників;

4) взаємодію з управлінським персоналом;

5) здійснення збирання та аналізу даних для управління й автоматичного виконання програмних засобів при настанні заданого часу з формуванням необхідної звітності;

6) реалізацію системи підказок і рекомендацій для користувачів;

7) ефективне збереження даних у базі даних і можливість доступу до них будь-якого кінцевого користувача зі свого робочого місця;

8) взаємодію користувачів між собою на основі безперервної технології.

Вся сукупність операцій оброблення інформації, що включає збирання, введення, запис, реєстрацію, перетворення, зчитування, збереження, знищення, коригування, обмін по каналах зв’язку, в автоматизованих системах здійснюється за допомогою технічних і програмних засобів.

У даний час багато підприємств відчувають необхідність поліпшення своєї автоматизованої системи. У першу чергу це зв'язано з незадоволеністю керівників якістю одержуваної ними інформації і швидкістю її одержання. Тому перед керівництвом постають такі питання: "Яким же чином організувати дані у залежності від цілей бізнесу, різних етапів розвитку компанії і поточного стану її автоматизації? Як вдосконалити організацію даних таким чином, щоб врахувати найбільш актуальні напрямки розвитку автоматизованої системи?

Особливо потребують вдосконалення даних системи з часовими обмеженнями - темпоральні системи, побудовані на основі темпоральних систем керування базами даних (СКБД) [2]. Розробка систем з жорсткими часовими обмеженнями вимагає розробки більш складних структур даних. Адже для правильного функціонування таких систем потрібно забезпечити ефективну форму збереження даних та роботу з даними, що прив'язані до часу. Звичайні бази даних зберігають миттєвий знімок моделі предметної області. Будь-яка зміна в момент часу t деякого елемента даних призводить до недоступності стану цього елемента даних у попередній момент часу. Водночас, у більшості відомих СКБД попередній стан елемента даних зберігається у часопису змін, але можливість доступу до цих даних для користувачів закрита.

Отже однією з проблем в таких системах є організація даних у зовнішній пам'яті. Нагадаємо, що основна теза темпоральних систем полягає у тому, що для будь-якого елемента даних, створеного в момент часу t1 і знищеного в момент часу t2, у базі даних зберігаються, і є доступними для користувачів, усі його стани в часовому інтервалі [t1, t2]. Дослідження і побудови прототипів темпоральних СКБД звичайно виконуються на основі деякої реляційної СКБД. Темпоральна СКБД — це надбудова над реляційною системою. Звичайно, це не кращий спосіб реалізації з погляду ефективності, але він простий і дозволяє виконувати досить глибокі дослідження.

Великі можливості для вдосконалення організації даних у програмних системах надає використання об'єктно-орієнтованого підходу, котрий радикально змінив сферу розробки програмного забезпечення вже в середині 1990-х років. Прихід об'єктно-орієнтованої парадигми додав гнучкості в моделюванні інформаційних систем, дозволив відображати ієрархічність елементів даних, представляти наслідування одних типів даних від інших з успадкуванням їх основних властивостей та поведінки з використанням інкапсуляції [3]. Об'єктно-орієнтований підхід дозволяє упакувати дані та код для його обробки разом. Таким чином практично знімається обмеження на типи даних, дозволяючи розробникам працювати на будь-якому рівні абстракції. Завдяки цьому

з'являється набагато більше можливостей досягнути мінімальних втрат при створенні та супроводженні програмних систем, призначених для обробки інформації, зберігаємої у рамках певних баз даних.

1. Визначення нових індексних структур даних з метою покращення їх властивостей

Сьогодні в сучасних застосуваннях баз даних все більш актуальною проблемою є зберігання й маніпулювання досить складними об'єктами, і здійснення над ними операцій, відсутніх у визначеному наборі. Традиційні типи даних вже не задовольняють досить високим вимогам, які пред'являються до сучасних застосувань баз даних. Основна проблема полягає у тому, що застосування мають досить суттєві відмінності у цих вимогах. Один із варіантів вирішення проблеми полягає у тому, щоб дозволити користувачам визначати власні типи даних.

Однією з потреб для створення нових структур даних є організація більш ефективного індексування об’єктів.

Як відомо, для прискорення доступу до інформації застосовуються індекси [4]. При послідовному методі доступу до даних для виконання запиту до таблиці бази даних проглядаються всі записи таблиці, від першої до останнього запису. Немає сенсу переглядати тисячі записів таблиці, коли запитові, можливо, задовольнять тільки декілька. У випадку індексно-послідовного методу доступу до даних, для виконання запиту до таблиці баз даних вказівник в індексі встановлюється на перший рядок, що задовольняє умові запиту (або його частини), і зчитується запис з таблиці по вказівнику, що зберігається на неї в індексі. Потім вказівник в індексі переміщається на наступний рядок і т. д. Як тільки рядок в індексі перестане задовольняти запитові, вибірка даних припиняється. Видно, що при індексуванні даних ми зчитуємо тільки ті дані, що задовольняють запитові. Індексування стовпця таблиці прискорює всі операції, зв'язані з пошуком значень цього стовпця. Наприклад, прискорюється видалення (коректування), якщо пошук записів, що видаляються (коректуються) здійснюється за умовами, накладеним на індексовану колонку. При цьому, однак, сповільнюється процедура додавання запису, тому що потрібно не тільки додати інформацію у файл даних, але й модифікувати індекси. Таким чином, чим більше індексів у таблиці, тим повільніше буде працювати операція модифікації. Щоб уникнути зайвого уповільнення роботи системи, рекомендується зупинитися на "золотій середині". Створення індексних структур даних вимагає врахування цих правил з метою забезпечення якості й продуктивності індексування.

Основне завдання, що виникає при проектуванні нової індексної структури даних є вказування на місцезнаходження об'єкта, маючи дані його ключа. Взагалі, обмежень на індексований тип даних накладати не бажано, оскільки це суттєво звужує область застосування описаного механізму індексації. Тому, єдиною вимогою для предметної області є те, щоб на множині значень ключа, по якому ідентифікуються індексовані об'єкти можна було визначити відношення часткового порядку. Це, власне, виражається в домені значення ключа. Використання об'єктного типу даних для представлення ключового поля (чи полів) при цьому не забороняється (звісно, при виконанні вищезазначеної умови), але є небажаним, оскільки виконання операції порівняння для екземплярів об'єктного типу є відносно дорогим за часом.

Якою б не була організація індексів в конкретній СКБД, їх основне призначення у забезпеченні ефективного прямого доступу до кортежу відношення по ключу. Взагалі індекс визначається для одного відношення, і ключем є значення атрибута (можливо, складового). Якщо ключем індексу є можливий ключ відношення, то індекс повинен володіти властивістю унікальності, інакше кажучи, не вміщувати дублікатів ключа. На практиці ситуація звичайно має вигляд зовсім протилежний: при об’явленні первинного ключа відношення автоматично заводиться унікальний індекс, а єдиним способом об’явлення можливого ключа, відмінного від первинного, є явне створення унікального індексу. Це зв’язано з тим, що для перевірки збереження властивості унікальності можливого ключа, так чи інакше потрібна індексна підтримка.

Оскільки при виконанні багатьох операцій язикового рівня потрібне сортування відношень відповідно до значень деяких атрибутів, корисною властивістю індексу є забезпечення послідовного перегляду кортежів відношення у діапазоні значень ключа в порядку зростання чи спадання значень ключа.

Нарешті, одним із засобів оптимізації виконання еквіз’єднання відношень (найбільш розповсюджена з числа дорогих операцій) є організація так званих мультиіндексів [5] для декількох відношень, які мають спільні атрибути. Будь-який з цих атрибутів (чи їх набір) може виступати в якості ключа мультиіндексу. Значенню ключа зіставляється набір кортежів усіх зв’язаних мультиіндексом відношень, значення виділених атрибутів яких співпадають зі значенням ключа.

Загальною ідеєю будь-якої організації індексу, який підтримує прямий доступ по ключу й послідовний перегляд у порядку зростання чи спадання значень ключа є зберігання впорядкованого списку значень ключа з прив’язкою до кожного значення ключа списку ідентифікаторів кортежів. Одна організація індексу відрізняється від іншої головним чином способом пошуку ключа з заданим значенням.

Б+-дерева [6] найбільш інтенсивно використовуються для організації індексів у базах даних. В основному це визначається двома властивостями цих дерев: передбачуваністю числа обмінів з зовнішньою пам’яттю для пошуку будь-якого ключа і тим, що це число обмінів за причиною сильної розгалуженості дерев не дуже велике при індексуванні навіть дуже великих таблиць. Хоча Б+-дерева зберігають надлишкову інформацію (один ключ може зберігатися у двох вузлах), вони володіють меншою глибиною, ніж класичні Б-дерева, а для пошуку будь-якого ключа потрібно одне й теж число обмінів із зовнішньою пам’яттю.

Для організації й управління індексами великої розмірності доцільно використовувати структуру даних, створену на основі принципів побудови Б+-дерев. Потрібно пам'ятати, що кожний перехід по дереву означає звернення до диску, а, отже, обходиться досить дорого. Операція доступу до диску означає посекторний обмін. Для зменшення кількості операцій обміну розмір вузла прирівнюється з розміром сектора й групуються разом декілька ключів у кожному вузлі. Загальний час доступу до зовнішньої пам'яті визначається в основному не об'ємом послідовно розташованих даних, а часом підводу магнітних голівок, то більш вигідно отримувати за одне звернення до зовнішньої пам'яті якомога більше інформації, враховуючи при цьому необхідність економного використання основної пам'яті.

Для більш економного і збалансованого використання зовнішньої пам'яті пропонується відмінний від класичного спосіб розщеплення та об'єднання вузлів дерева. Для застосування розщеплення і об'єднання вузлів використовуються критерії, які базуються на рівні наповненості відповідного вузла. В структурах даних, які використовують техніку Б+-дерев, пошук ключа завжди доходить до листового вузла. Аналогічно операції вставки і видалення також починаються з листового вузла. При вставці нового ключа зміст підходящого, але заповненого вузла перерозподіляється між братами. Якщо брати також заповнені, створюється новий вузол і половина ключів нащадка пересилаються у нього. Під час видалення наполовину заповненні нащадки є першими кандидатами на добавлення ключів із сусідніх вузлів. Якщо самі сусідні вузли заповнені тільки наполовину, вони об'єднуються так, щоб вийшов повний вузол. Але для підвищення продуктивності дерева можна застосувати інший метод розщеплення та об'єднання вузлів. Ми збираємо три вузли, а потім розщеплюємо їх. При вставці, коли нам потрібен додатковий вузол, розщеплюємо на чотири вузли. Коли вузол потрібно видалити, розщеплюємо на два вузли. Симетрія операцій дозволяє використовувати при реалізації вставки та видалення одні й ті функції для збору та розщеплення вузлів дерева.

2. Особливості створення деревовидних структур даних для підвищення швидкості реорганізації індексів

Типові деревовидні структури даних для організації індексів мають досить суттєві недоліки. Основний недолік - незбалансованість роботи структур після видалення і вставки деяких елементів. Оскільки під час пошуку при кожному проходженні елементів структури виконується звертання до диску, загальна тривалість пошуку в незбалансованій деревовидній структурі може виявитися зовсім непередбаченою. Вирішення даної проблеми дасть змогу оптимізувати індексні структури даних для застосувань, у яких швидкість оновлення даних, а також швидкість роботи функції пошуку об'єкта є характеристиками, які визначають якість роботи застосування. Наприклад, в просторово-часових базах даних індексування має складну структуру. Зв'язано це зі специфічністю зберігаємих у цих базах даних характеристик об'єктів.

Застосування такого роду потребують засобу керування великою кількістю інформації, у тому числі про реальне місцерозташування об'єктів, а також відображаючому переміщення цих об'єктів на протязі часу. Наприклад, застосування відслідковування руху може потребувати сховища інформації про засіб пересування (машин, автобусів, мотоциклів і т.д.). Аналогічно, танки, літаки та одиниці військової техніки при комп'ютерному моделюванні військових дій також потребують спеціального представлення. Зміна властивостей об'єктів є неперервним. Звичайно, це повинно бути відображено в засобі управління даними застосування. Постійне переміщення об'єктів робить недопустимим використання традиційних СКБД у задачах такого характеру, призначених в основному для робіт зі статичними даними. Зміна характеристик об'єкта може бути представлена в базі даних за допомогою параметрів сутності, що змінюються та функції місцерозташування, яка визначає положення об'єкта в просторі в будь-який заданий час.

У словниках-довідниках як підсистемах інформаційних ресурсів сучасних СКБД індекси, засновані на деревоподібних структурах, при пошуку елементів таблиць, доменів і списків витиснули такі класичні методи доступу, як індексно-послідовний і хешування. Основні різновиди деревоподібних структур: бінарне дерево пошуку і Б-дерево - забезпечують великий обсяг відновлення даних у БД. Для обробки звітів і пакетних транзакцій запису даних чи кортежі доменів організують як упорядковані послідовні файли (списки). Швидкий пошук і наявність у кожній вершині дерева покажчика на відповідний запис полегшують довільну вибірку даних. Операції включення і видалення після пошуку виконуються швидко, оскільки змінюються тільки значення покажчиків, а перерозподіляти пам'ять під дані не треба.

З підвищенням швидкості комп'ютерної обробки й ростом обсягів доступної пам'яті змінилися вимоги до організації індексів [7]. Якщо в СКБД перших поколінь головною характеристикою була швидкість доступу до даних, то сьогодні на перший план вийшла швидкість реорганізації індексу через суттєву мінливість БД і неминуче часту перебудову індексу на велику глибину. Тому ще однією вимогою до індексів є наявність потенційно великого числа вільних місць, що дозволить досягнути рідкої потреби у перебудові індексу.

3. Проектування індексу

При проектуванні індексу у вигляді Б-дерева його ступінь значно впливає на час пошуку, корисне застосування кожної вершини й продуктивність модифікації даних. Загалом конфігурація Б-дерева та правила балансу пристосовані до інтенсивного відновлення даних, тобто до мінливості даних у базах даних великого розміру.

Б+-дерево [6] - це модифікований варіант Б-дерева, у якого існує два типи вузлів. Перший тип вузлів це не листковий вузол або так званий проміжний (індексний) вузол, аналогічний вузлові звичайного Б-дерева.

Другий тип вузлів - кінцевий вузол (вузол даних) даних, що зберігає, замість посилань на послідовників, посилання на дані. Всі вузли містять у собі множини пар (ключ, показник), які впорядковані за значенням ключа в порядку зростання. Показники проміжних вузлів служать для посилання на інші показники проміжних або кінцевих вузлів. Кінцеві вузли знаходяться тільки на нижньому рівні дерева. Ключі кінцевих вузлів (листків) впорядковані всередині дерева, що дозволяє зменшити час сканування, оскільки цей процес виконується не послідовно. Ключ застосовується для організації інформації всередині Б+-дерева. В базі даних кожний запис має поле ключа, за значенням якого розрізняють записи одного й того ж типу. Б+-дерева використовують ключ для побудови індексу записів баз даних, який дозволяє скоротити час доступу до них. Порівнюючи текучий ключ з ключем вузла, що шукається, програма знаходить потрібну інформацію. Це дерево є збалансованим за висотою. Тому час пошуку об'єкта в такому дереві залежить від його висоти.

У реалізації нової індексної структури даних потрібно забезпечити необхідність множинних посилань на одні й ті ж дані, дозволити здубльовані ключі [8]. У межах одного індексу всі ключі повинні бути однієї довжини. Для утримання вузлів у пам'яті до тих пір, доки вистачає пам'яті, потрібно реалізувати гнучку схему буферизації. Якщо очікується доступ до чого-небудь упорядкованого потрібно зменшити кількість звертань до диску. Але при цьому треба обов'язково дотримуватися стандартних властивостей Б+-дерева. Всі ключі мають зберігатися у листах, там також зберігається й інформаційна частина вузла. У внутрішніх вузлах зберігаються копії ключів - вони допомагають шукати потрібний лист. Лівий показник веде до ключів, які менше заданого значення, правий - ключам, які більші або рівні. Також необхідно врахувати особливості роботи з батьківськими вузлами під час операцій вставки та видалення. Коли модифікується перший ключ у листі, дерево проходить від листа до кореня. Останній з показників (більше або рівно), знайдений при спуску по дереву, і є тим, що потребує модифікації, щоб відобразити нове значення ключа. Оскільки всі ключі повторюються у листах, ми можемо зв'язувати їх для послідовного доступу.

При переповненні вузла в Б+-дереві, потрібно зробити розбивку цього вузла на два (рис. 1, а). Операція розподілу переміщає середину вузла А у свого попередника вузол В, де А є і-им нащадком вузла В. Новий вузол С створюється, і всі ключі вузла А, що знаходяться правіше центрального значення, переміщаються в С; ключі, що лежать лівіше медіани залишаються у вузлі А. Новий вузол С стає нащадком наступного за медіаною, переміщеною у вузол попередника, правого ключа, а змінений вузол А стає нащадком ключа, що стоїть лівіше переміщеної в попередника медіани. Операція розподілу повного вузла поділяє вузол з 2n-1 ключами на два вузли з n-1 ключем (де n – рівень вузла), один ключ переноситься в попередника (рис. 1, б).

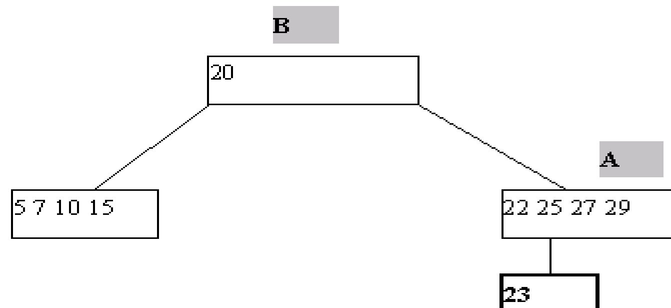


Рис.1а. Спроба вставити ключ 23 у вже заповнений вузол

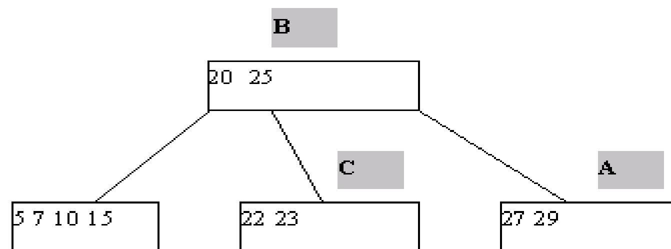


Рис. 1, б. Виконання включення ключа 23 шляхом розщеплення вузла А

Далі наведено блок-схему процедури розщеплення переповненого вузла при вставці у нього нового ключа (рис. 2).

Перед тим, як зробити вставку нового ключа в Б+-дерево, необхідно знайти той вузол, у який вставлятиметься новий ключ. Ця операція схожа з операцією пошуку ключа. Якщо знайдений вузол дерева не повний, то новий ключ просто вставляється відповідно до упорядкованості у вузлі. Якщо ж вузол повний, то спочатку потрібно розділити його на два, перемістити середину вузла в попередника, якщо попередник теж повний, то його поділити на два і т.д. Ця процедура може дійти до кореня дерева. Є й інший варіант, це перебудова сусідніх вузлів. Особливий випадок може виникнути при розподілі кореневого вузла, можливо, прийдеться поділяти його на два і створювати новий корінь.

Видалення ключа починається з пошуку вузла, що містить ключ. Далі ключ видаляється з вузла. Якщо кількість записів у вузлі стає меншою від мінімальної, то виконується переливання ключів. Може виявитися, що жоден з сусідніх вузлів не придатний для переливання, оскільки містить по n ключів. Тоді виконується

процедура злиття сусідніх листових вузлів. Далі наведено блок-схему процедури злиття вузлів при видаленні ключа (рис. 3).

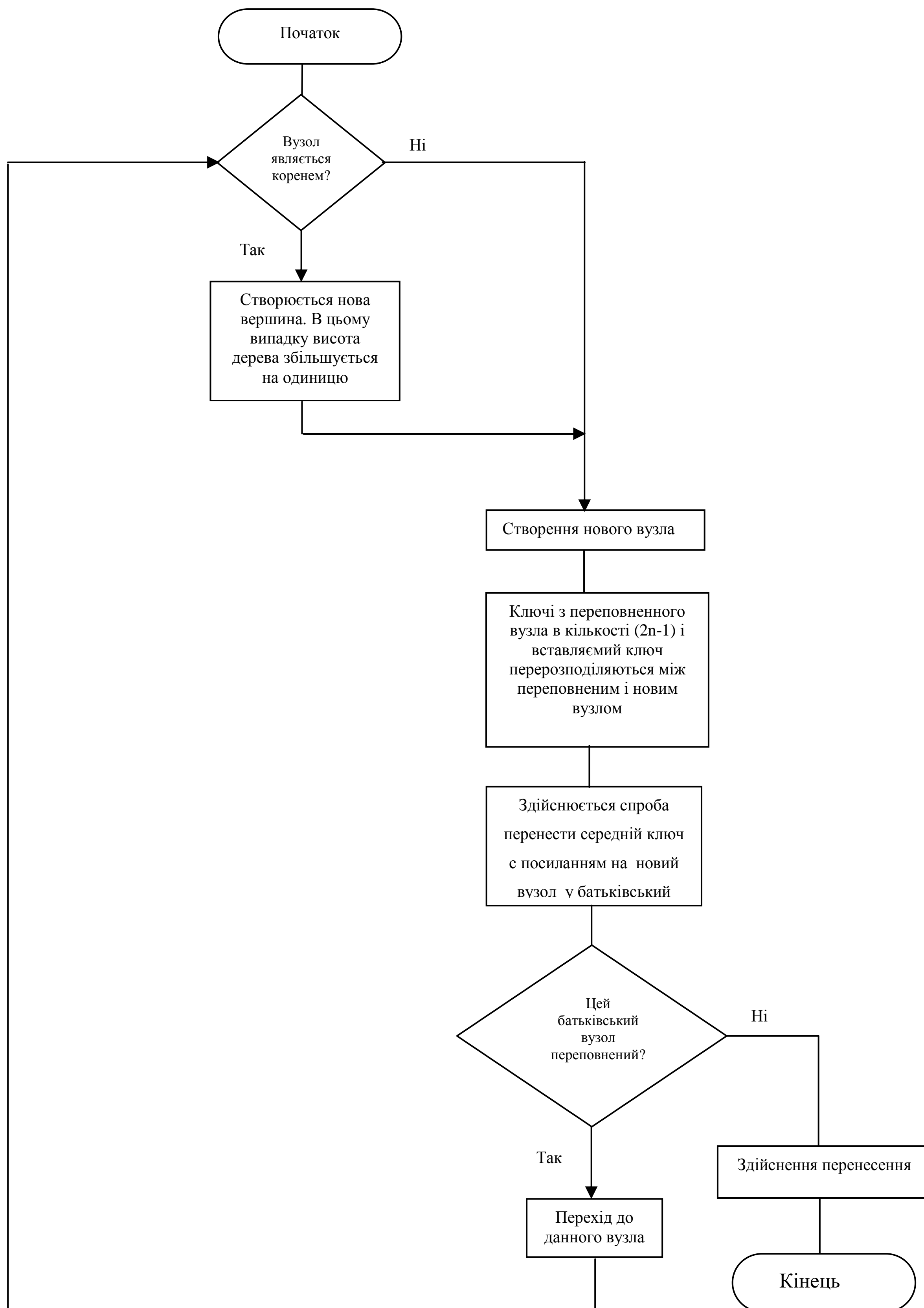


Рис. 2. Блок-схема розщеплення переповненного вузла при вставці ключа

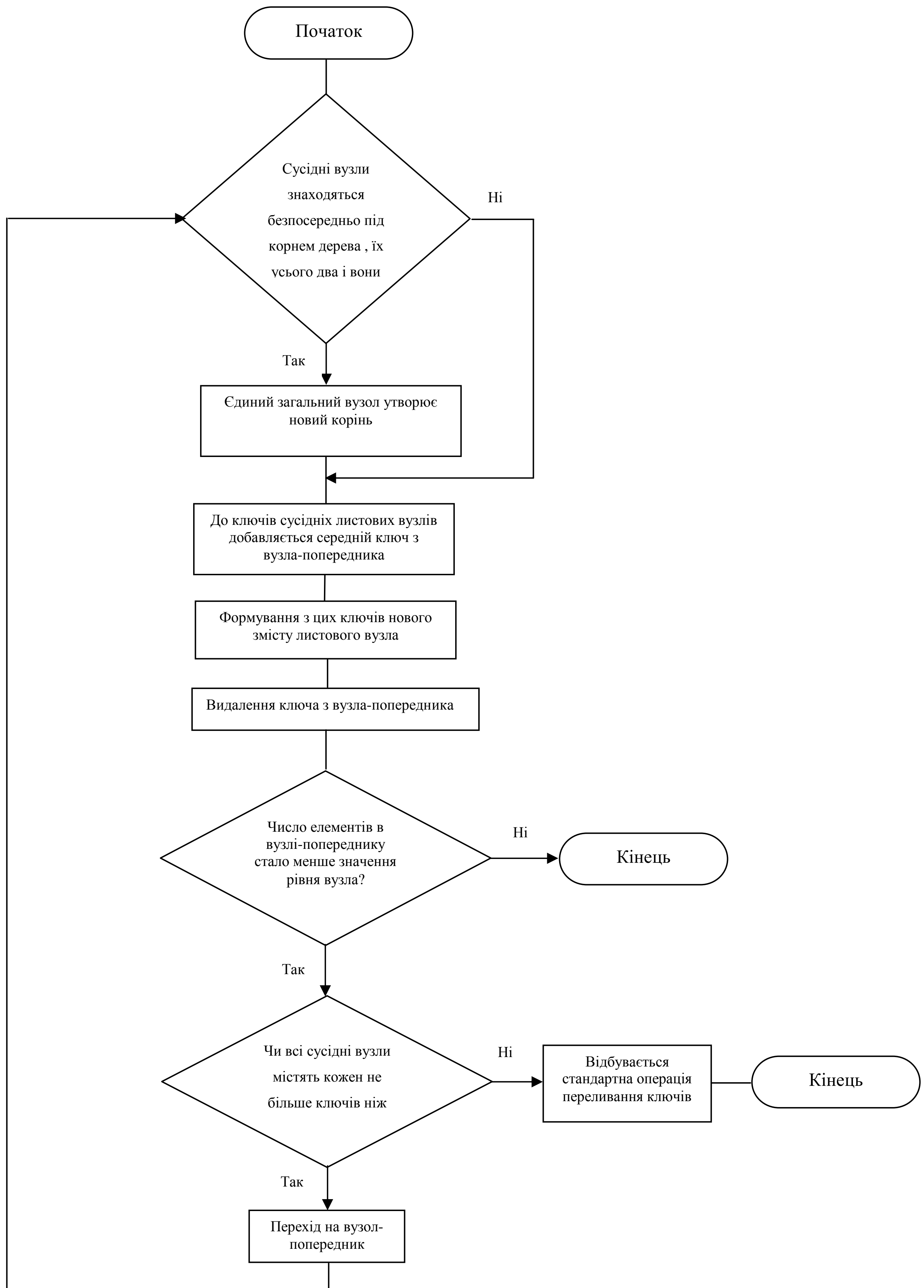
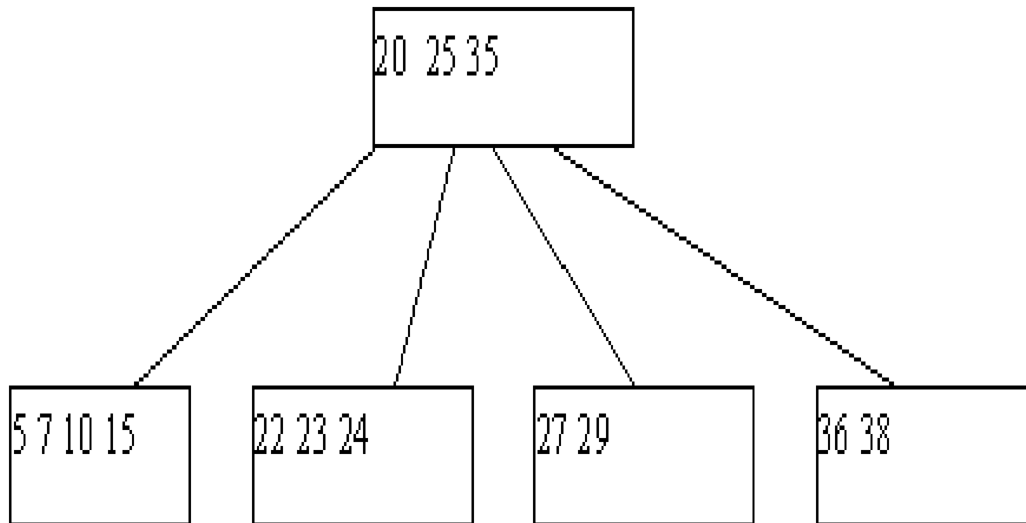


Рис. 3. Блок-схема злиття вузлів при видаленні ключів

До 2n-1 ключів сусідніх листових вузлів добавляється середній ключ з вузла-попередника (з вузла попередника він видаляється) і всі ці ключі формують новий зміст початкового листового вузла (рис. 4, а). Оскільки в вузлі-попереднику число ключів зменшилося на одиницю, може виявитися, що число елементів у ньому стало менше n (де n – рівень вузла), і тоді на цьому рівні виконується процедура переливання, а можливо і злиття (Рис.4б). Так може відбуватися до внутрішніх вузлів, які знаходяться безпосередньо під коренем B-дерева. Якщо таких вузлів усього два, і вони зливаються, то єдиний загальний вузол утворює новий корінь. Висота дерева зменшується на одиницю, але як і надалі довжина шляху до будь-якого листа одна й та ж.



а

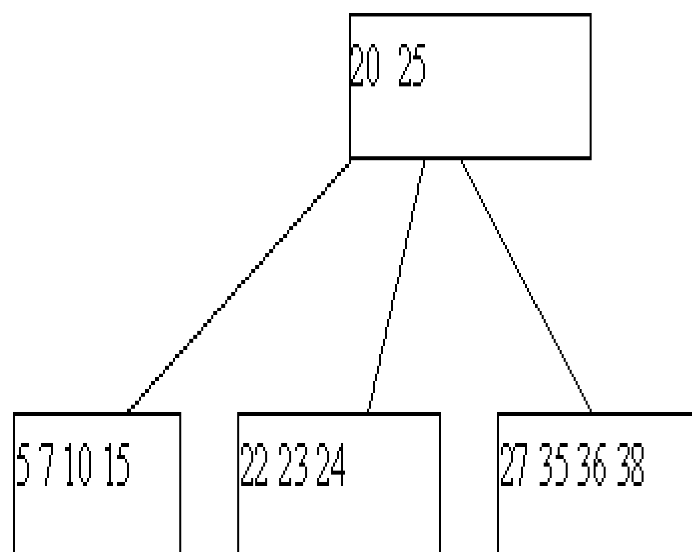


Рис. 4. а - Початковий вид дерева; б - Дерево після видалення ключа 29

Для більш економного і збалансованого використання зовнішньої пам'яті та для кращого використання місця, яке займає побудована на базі техніки Б+-дерев структура даних (а також для підвищення продуктивності дерева) можна запропонувати наступний спосіб розщеплення (рис. 5, а) та об'єднання вузлів дерева (рис. 5, б).

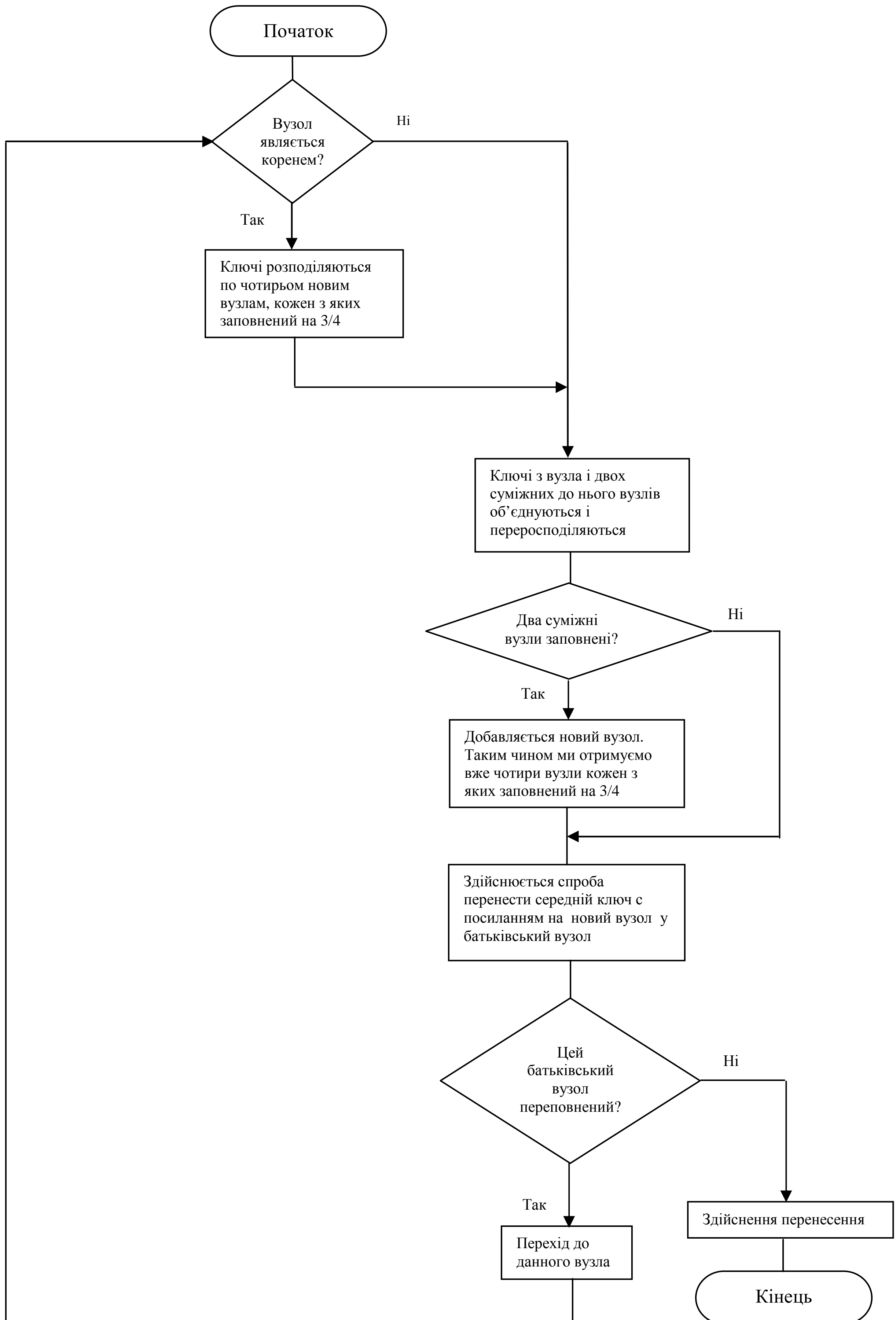


Рис. 5а. Метод розщеплення вузла дерева

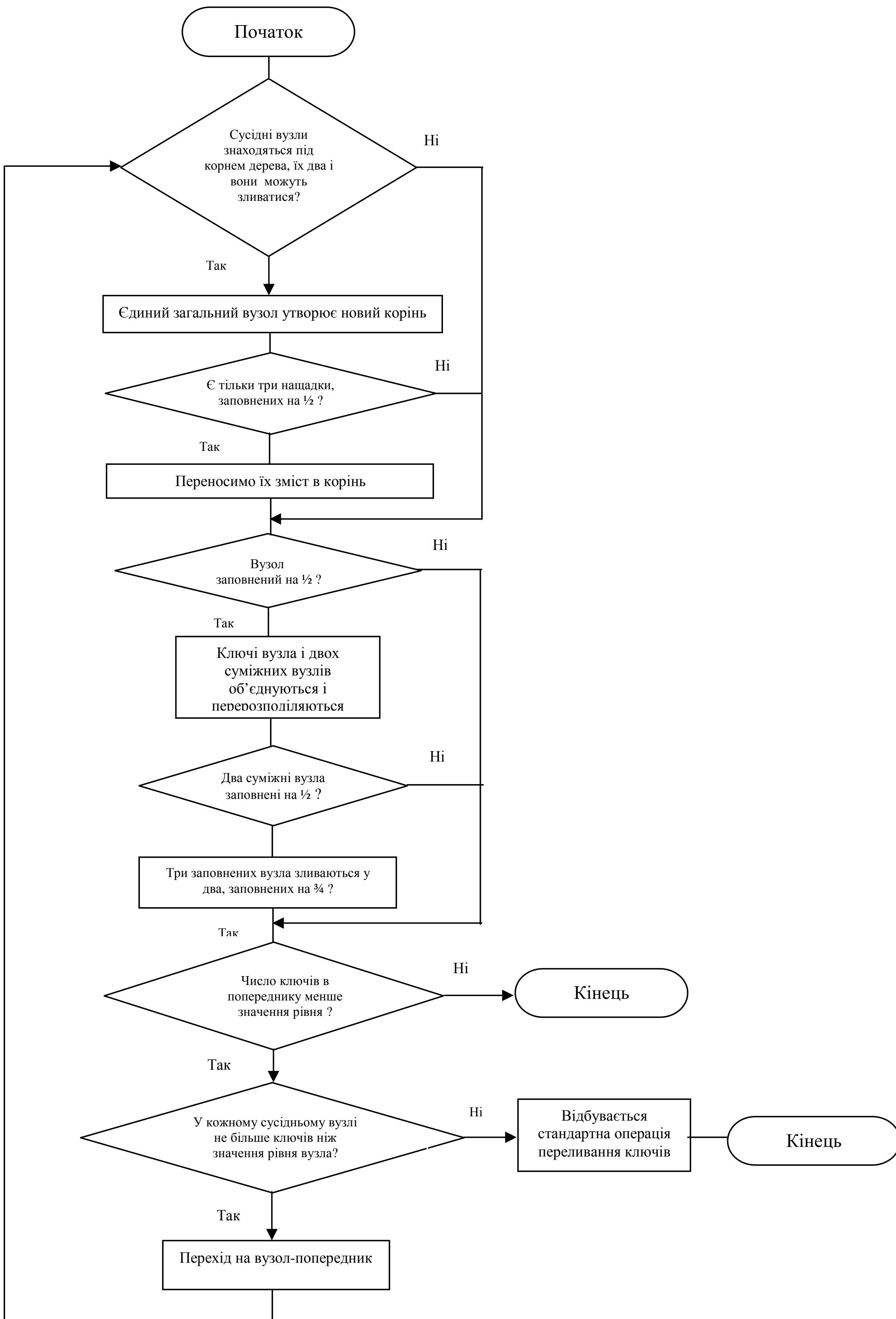


Рис.5б. Метод злиття вузлів дерева

Алгоритм забезпечує можливість зберігання у кожному вузлі n ключів, кількість ключів у корні
може бути 3n . Попередньо виконавши перевірку на порожність нащадка під час вставки, спускаємось до нього. Якщо нащадок пустий, ключі, які знаходяться у ньому та двох суміжних до нього вузлах, об'єднуються і перерозподіляються. Якщо два суміжних вузла також заповнені, то додається вузол. Таким чином, ми отримуємо вже чотири вузла, кожний з яких на ¾ заповнений. Під час видалення попередньо виконуємо перевірку на наповненість нащадка наполовину, далі опускаємося до нього. Якщо нащадок заповнений наполовину, ключі нащадка й двох суміжних вузлів об'єднуються і перерозподіляються. Якщо два суміжних вузла самі заповнені наполовину, вони зливаються у два вузли, кожний з яких повний на ¾. Таким образом, опиняємося посередині між наповненістю наполовину та повною наповненістю, що дозволяє нам сподіватися на однакове число вставок і видалень.

Якщо під час вставки виявиться, що корінь повний, ми розподіляємо ключі по чотирьом новим вузлам, кожний з яких заповнений на ¾. Ці дії призведуть до збільшення висоти дерева. Під час видалення алгоритм досліджує нащадків. Якщо є тільки три нащадки і вони заповнені наполовину, переносимо їх зміст у корінь, це призведе до зменшення висоти дерева.

Отже, збираємо три вузла, а потім розщеплюємо їх. При вставці, коли нам потрібен додатковий вузол, розщеплюємо на чотири вузла. При видаленні, коли вузол треба видалити, ми розщеплюємо на два вузли. Цей спосіб зберігає збалансованість дерева. Максимальна висота росте не швидше, ніж логарифм числа елементів.

Основною "родзинкою" Б-дерев є автоматична підтримка властивості збалансованості. Дослідимо, як це робиться при виконанні операцій занесення і видалення записів.

У основу сучасних формалізмів для опису алгоритмів покладений апарат комп'ютерної алгебри та логіки. Системи алгоритмічних алгебр (САА) [9] – надвисокого рівня формалізм для описання алгоритмів, який запропонований В.М. Глушковим у 1965 році. Алгоритм дій, які виконуються при занесенні нового запису, можна описати за допомогою САА-схеми.

СХЕМА
ВСТАВКА = "Занесення нового запису в Б+-дерево"
"СТАРТ" *
ПОКИ 'Пошук не досяг листового вузла'
ЦИКЛ (ВИКОНАТИ ("Читання кореневого вузла дерева з ключами та посиланнями на інші вузли в основну пам'ять") *
ВИКОНАТИ ("Послідовний пошук у вузлі потрібного ключа") *
ЯКЩО 'Ключ знайдено'
ТО 'Пошук завершено'
ІНАКШЕ ВИКОНАТИ ('Перехід по посиланню вузла'));
ЯКЩО 'В листі знаходиться 2n ключів (n – рівень вузла)'
ТО ВИКОНАТИ ('Процедуру розщеплення вузлів')
ІНАКШЕ ВИКОНАТИ ('Ключ поміщається на своє місце, яке визначене порядком сортування ключів') ;
КІНЕЦЬ;

Далі наведено опис операцій, представлених за допомогою САА-схеми.

Пошук листового вузла. Фактично, проводиться звичайний пошук по ключу. Якщо в Б-дереві не міститься ключ з заданим значенням, то буде отриманий номер вузла, в якому йому належить міститися, і відповідні координати в середині вузла.

Вставка запису на місце. Природно, що вся робота виконується в буферах оперативної пам'яті. Листовий вузол, у який потрібно занести запис, зчитується у буфер, і в ньому виконується операція уставки. Розмір буфера повинен перевищувати розмір вузла зовнішньої пам'яті.

Якщо після виконання вставки нового запису розмір частини буфера, що використовується, не перевищує розмір вузла, то на цьому виконання операції занесення запису закінчується. Буфер може бути негайно виштовхнутий у зовнішню пам'ять, або тимчасово збережений в оперативній пам'яті в залежності від політики управління буферами.

Якщо виникло переповнення буфера (тобто розмір його частини, що використовується, переверщує розмір вузла), то виконується розщеплення вузла. Для цього запитується новий вузол з зовнішньої пам'яті, частина буфера, що використовується, розбивається грубо кажучи навпіл (так, щоб друга половина також починалась з ключа), і друга половина записується у знову виділений вузол, а в старому вузлі модифікується значення розміру вільної пам'яті. Природно, модифікуються посилання за списком листових вузлів.

Щоб забезпечити доступ від кореня дерева до заново заведеного вузла, необхідно відповідним чином модифікувати внутрішній вузол, який є нащадком листового вузла, що вже існував раніше, тобто вставити в нього відповідне значення ключа й посилання на новий вузол. При виконанні цієї дії може знову відбутися переповнення тепер вже внутрішнього вузла, і він буде розщеплений на два. Зрештою знадобиться вставити значення ключа й посилання на новий вузол у внутрішній вузол-нащадок вище за ієрархією і т.д.

Граничним випадком є переповнення кореневого вузла Б-дерева. В цьому випадку він також розщеплюється на два, і заводиться новий кореневий вузол дерева, тобто його глибина збільшується на одиницю.

При видаленні запису виконуються такі дії:

СХЕМА
ВИДАЛЕННЯ = "Видалення ключа у Б+-дереві"
СТАРТ *
ПОКИ 'Пошук не досяг листового вузла'
ЦИКЛ (ВИКОНАТИ ("Читання кореневого вузла дерева з ключами та посиланнями на інші вузли в основну пам'ять")
ВИКОНАТИ ("Послідовний пошук у вузлі потрібного ключа") *
ЯКЩО 'Ключ знайдено'
ТО 'Пошук завершено'
ІНАКШЕ ВИКОНАТИ ("Перехід по посиланню вузла");
ВИКОНАТИ ("Видалення ключа в дереві") ;
ЯКЩО 'Хоч один з сусідніх вузлів придатний до переливання'
ТО ВИКОНАТИ ("Відбувається перехід на один з сусідніх листових вузлів (з спільним вузлом-попередником)")
ІНАКШЕ ВИКОНАТИ ("Процедури злиття вузлів дерева") ;
ВИКОНАТИ ("Ключі, які містяться у цих вузлах і середній ключ вузла-попередника розподілити між листами");
ВИКОНАТИ ("Новий середній ключ вставляється на місце ключа вузла-попередника");
КІНЕЦЬ;

Далі наведено опис операцій, представлених за допомогою САА-схеми.

Пошук запису по ключу. Якщо запис не знайдено, то значить видаляти нічого не потрібно.

Реальне видалення запису у буфері, в якій прочитано відповідний листовий вузол. Якщо після виконання цієї підоперації розмір занятої у буфері області виявляється таким, що його сума з розміром занятої області в листових вузлах, є лівим чи правим братом даного вузла, більше, ніж розмір вузла, операція завершується.

Інакше відбувається злиття з правим або лівим братом, тобто в буфері створюється новий образ вузла, яка вміщує загальну інформацію з даного вузла та його лівого чи правого брата. Листовий вузол, який став непотрібний, заноситься у список вільних вузлів. Відповідним чином, коректується список листових вузлів.

Щоб усунути можливість доступу від кореня до звільненого вузла, треба видалити відповідне значення ключа й посилання на звільнений вузол з внутрішнього вузла - його нащадка.

При цьому може виникнути потреба в злитті цього вузла з його лівим або правим братами і т.д.

Граничним випадком є повне спустошення кореневого вузла дерева, яке можливе після злиття останніх двох нащадків кореня. У цьому випадку кореневий вузол звільнюється, глибина дерева зменшується на одиницю.

Проблемою є те, що при виконанні операцій модифікації занадто часто можуть виникнути розщеплення і злиття. Тоді щоб добитися ефективного використання зовнішньої пам'яті з мінімізацією числа розщеплень і злиттів, можна використати такі прийоми:

1. Випереджаючі розщеплення, тобто розщеплення вузла не при його переповненні, а трохи раніше, коли степінь залежності вузла досягає деякого рівня.
2. Переливання, тобто підтримка рівномірного заповнення сусідніх вузлів.

Варто відзначити, що при організації мультидоступу до Б-дерев, характерного при їх використанні в СКБД, доводиться вирішувати ряд нетривіальних проблем. Найбільш правильним рішенням є монопольний захват Б-дерева на все виконання операції модифікації.

4. Результати та висновки

У роботі розглянуто проблему життєздатності програмних систем, досить актуальну на сьогоднішній день. Розглянуто класи методів забезпечення життєздатності програмних систем, а також можливості вдосконалення цих методів за рахунок зміни організації даних у програмних системах. Для більш економного і збалансованого використання зовнішньої пам'яті пропонується відмінний від класичного спосіб розщеплення та об'єднання вузлів дерева. Використання даного способу організації індексу дерева призводить до кращого використання місця, яке займає дерево та більш кращої продуктивності. Покращені властивості дерева дозволяють не втратити швидкості розщеплення вузлів. Для застосування розщеплення і об'єднання вузлів використовуються критерії, які базуються на рівні наповненості відповідного вузла. Особлива увага приділяється проблемі ліквідації таких недоліків ієрархічних структур даних, як незбалансованість роботи структур після видалення і вставки деяких елементів. У результаті таких змін структури елементів можуть опинитися на різних рівнях від кореневого елемента. Оскільки під час пошуку при кожному проходженні елементів структури виконується звертання до диску, загальна тривалість пошуку в незбалансованій деревовидній структурі може виявитися зовсім непередбаченою. Хоча на практиці для скорочення числа звернень до диска кореневий рівень (інколи інші) при роботі СКБД здебільшого зберігається в оперативній пам'яті, цей недолік досить суттєвий.

Продовження дослідження у цьому напрямку дозволить покращити властивості дерева, удосконалити алгоритми сортування об'єктів у разі вставки в незаповнений вузол, провести аналіз взаєморозташування об'єктів на одному рівні дерева, вести спеціальну метрику відстані між об'єктами. Це дасть змогу будувати індексну структуру оптимальніше з приводу пошуку.

1. *Ігнатенко П.П.* Проблеми забезпечення життєздатності програмних систем та підходи до їх вирішення // Проблеми програмування. - 2002. - № 3-4. - С. 58 - 73.
2. *Чічкань І.С.* Бази даних у темпоральних системах з жорсткими обмеженнями. – М.: Наука, 2003. - 178 с.
3. *Інтернет* ресурс "Центр Информационных Технологий" www.citforum.ru
4. *Дейт К. Дж.* Введение в системы баз данных. – М.: Вильямс, 2000, - 848 с.
5. *Кормен Т., Лейзерсон Ч., Ривест Р.*. Алгоритмы: построение и анализ. - М.: МЦНМО, 1990. - 960 с.
6. *Перевозчикова О.Л.* Інформаційні системи та структури даних. – К.: Видавничий дім "КМ-Академія", 2004. – 149 с.
7. *Інтернет* ресурс Object Management Group - www.omg.org.
8. *Ахо А., Хопкрофт Д., Ульман О.* Построение и анализ вычислительных алгоритмов. - М.: Наука, 1989. - 360 с.
9. *Цейтлин Г.Е.* Введение в алгоритмику. – К.: Сфера, 1998. - 310 с.