УДК 004.62

ЗАСТОСУВАННЯ БІНАРНОГО ДЕРЕВА ПОШУКУ З ФІКСОВАНОЮ ВИСОТОЮ ДЛЯ ПРИСКОРЕННЯ ОБРОБКИ ОДНОВИМІРНИХ МАСИВІВ

Шпортько О. В. – канд. техн. наук, доцент, доцент кафедри інформаційних систем та обчислювальних методів ПВНЗ «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука», Рівне, Україна.

Бомба А. Я. – д-р техн. наук, професор, професор кафедри комп'ютерних наук та прикладної математики Національного університету водного господарства та природокористування, Рівне, Україна.

АНОТАЦІЯ

Актуальність. На сьогодні для прискорення пошуку, сортування та відбору елементів масивів широко використовуються бінарні дерева пошуку. Але обчислювальна складність пошуку з використанням бінарного дерева пропорційна його висоті, яка, в свою чергу, залежить від послідовності опрацювання елементів масиву. Для зменшення висоти дерева періодично виконують його балансування, яке є тривалим процесом, тому розробка альтернативних способів контролю за висотою бінарного дерева є на сьогодні актуальним науковим завданням.

Мета. Розробка принципів та відповідних алгоритмів формування та використання бінарного дерева з фіксованою висотою для прискорення пошуку елемента в масиві та визначення довільної *i*-ї порядкової статистики, зокрема, медіани масиву.

Метод. В дослідженні запропоновано встановлювати фіксовану висоту бінарного дерева пошуку на одиницю більшою від мінімально можливої висоти бінарного дерева для розміщення всіх елементів масиву, адже збільшення фіксованої висоти призводить до зайвих витрат оперативної пам'яті, а зменшення – сповільнює модифікації дерева. Формування таких дерев подібне до балансування дерев, але, на відміну від нього, рекурсивне переміщення вузлів у них виконується лише тоді, коли відповідне піддерево заповнене повністю. Для бінарного дерева пошуку з фіксованою висотою оперативна пам'ять виділяється один раз при його створенні – відразу під всі можливі вузли бінарного дерева заданої висоти. Це дає змогу уникнути операцій виділення та звільнення пам'яті під кожен вузол дерева та зберігати значення вузлів в одновимірному масиві без використання вказівок.

Результати. Наші експерименти показали, що для прискорення пошуку елементів та визначення *i*-тих порядкових статистик часто змінюваних невпорядкованих масивів доцільно додатково формувати бінарне дерево пошуку з фіксованою висотою. Для ініціалізації цього дерева доцільно використати відсортовану копію ключів елементів масиву, а не вставляти їх почергово. Використання бінарного дерева з фіксованою висотою прискорює, наприклад, пошук медіан таких масивів більш ніж в 7 разів відносно методу двох бінарних пірамід та додатково прискорює перерозподіл стиснутих даних між модифікованими DEFLATE-блоками в процесі прогресуючого ієрархічного стиснення без втрат зображень набору ACT в середньому на 2,92%.

Висновки. Для визначення медіан чи *i*-тих порядкових статистик окремих непов'язаних масивів та підмасивів замість відомих методів сортування доцільно використовувати розбиття Hoare з обміном на великих відстанях, оскільки воно переставляє лише окремі елементи, а не впорядковує весь масив повністю. З метою визначення медіан послідовності вкладених підмасивів, впорядкованих за зростанням їх довжини, варто застосовувати метод двох бінарних пірамід, адже вони орієнтовані на швидке доповнення новими елементами. Для знаходження медіан чи *i*-тих порядкових статистик після змін чи вилучень елементів невпорядкованого масиву доцільно використати бінарне дерево пошуку ключів елементів масиву з фіксованою висотою, оскільки таке фіксування запобігає неконтрольованому зростанню кількостей операцій порівняння та дає змогу обробляти дерево без використання вказівок.

КЛЮЧОВІ СЛОВА: сортування масивів, медіани масиву, метод двох бінарних пірамід, бінарне дерево пошуку з фіксованою висотою.

АБРЕВІАТУРИ

ACT – Archive Comparison Test, тестовий набір зображень;

BST – Binary Search Tree, бінарне дерево пошуку;

RAM – Random Access Memory, оперативна пам'ять;

БД – База Даних.

НОМЕНКЛАТУРА

countHBlock – початкова кількість результуючих блоків кожного модифікованого DEFLATE-блоку;

countLevelTree – кількість рівнів BST з фіксованою висотою;

$countLeft_i$ – кількість заповнених вузлів зліва для *i*-го вузла BST;

$countRight_i$ – кількість заповнених вузлів справа для *i*-го вузла BST;

$countSub_i$ – кількість створених підпорядкованих вузлів зліва і справа для *i*-го вузла BST;

indexArray – індекс елемента в масиві для вузла BST;

indexDel – індекс елемента для вилучення з відсортованого масиву ключів;

indexInsert – індекс елемента для вставки у відсортований масив ключів;

$\log(N)$ –логарифм числа N за основою 2;

N – кількість елементів вхідного масиву;

$tree_i$ – *i*-й вузол BST з фіксованою висотою;

DOI 10.15588/1607-3274-2025-1-18

indexLeft$_i$ – індекс підпорядкованого вузла зліва для *i*-го вузла BST;

valueArray – значення елемента з масиву для вузла BST;

valueNode – структура вузла BST;

X –вхідний масив для сортування;

x_i – *i*-й елемент масиву X для сортування;

Y_i – *i*-й елемент послідовності неперервних підмасивів, впорядкованих за зростанням довжини, який містить елементи з x_0 по елемент x_i;

Z_i – *i*-й елемент послідовності неперервних підмасивів, впорядкованих за спаданням довжини, який містить елементи з x_0 по елемент x_i.

ВСТУП

Як відомо, на сьогодні для прискорення пошуку, сортування та відбору елементів масивів широко використовуються бінарні дерева пошуку (BST) [1]. Нагадаємо, що таке дерево – це пов'язаний ациклічний орієнтований граф, який складається з кореневого вузла та лівого і правого піддерев, що не перетинаються між собою [2]. В кожному вузлі дерева міститься ключ, який складається зі значення одного з елементів масиву та індекса цього елемента в масиві. Зліва від кожного вузла BST містяться вузли з меншими, а справа – з більшими значеннями ключів. Таке дерево дає змогу реалізувати аналог бінарного пошуку замість лінійного, адже після порівняння ключа пошуку з ключем чергового вузла цей процес рекурсивно продовжується в одному з двох напрямків (зліва чи справа), а вузли в іншому напрямку відкидаються [2]. Принципи обробки елементів масиву за допомогою BST широко використовуються в індексах таблиць БД [3; 4].

Обчислювальна складність пошуку з використанням BST пропорційна його висоті [1], яка, в свою чергу, залежить від послідовності опрацювання елементів масиву при формуванні і вставці вузлів. Тому для зменшення висоти дерева періодично виконують його балансування [2], яке є тривалим процесом. Отже, розробка алгоритмів для автоматичного підтримування висоти дерева, близької до мінімальної, є на сьогодні актуальним науковим завданням.

Бінарне дерево пошуку також дає змогу швидко віднайти найменше чи найбільше значення в масиві, на які вказують, відповідно, найлівіший чи найправіший вузол дерева відносно його кореня. Поряд з цим, бінарне дерево пошуку несуттєво прискорює пошук довільної *i*-ї порядкової статистики, зокрема медіани масиву [5], адже це дерево впорядковує елементи, але не вказує місце кожного елемента у відсортованому масиві. Значення зазначеного недоліку BST не варто недооцінювати, адже на сьогодні в статистиці та соціології для аналізу ряду величин замість середнього арифметичного, мінімального чи максимального значення серед всіх його елементів все ширше використовують саме його медіану (величину, що розташована всередині ряду величин, розміщених у зростаючому або спадному порядку). Медіана ділить ряд значень ознаки на дві рівні частини [6; 7], тому для масивів з непарною кількістю елементів – це значення центрального елемента, а з парною – середнє арифметичне двох центральних елементів, якщо елементи масиву відсортувати. Елементи масиву можуть змінюватися з часом і щоразу формувати відсортовану копію масиву чи проходитися по частині його елементів для знаходження медіани або *i*-ї порядкової статистики недоцільно. Саме тому прискорення визначень *i*-ї порядкової статистики загалом і медіани масиву зокрема, в тому числі за допомогою BST, теж є на сьогодні актуальним науковим завданням.

Отже, **об'єктом** цього **дослідження** є механізми формування та використання бінарного дерева пошуку для прискорення обробки одновимірних масивів. **Предметом дослідження** є засоби реалізації та використання бінарного дерева пошуку з фіксованою висотою. **Мета дослідження** – розробка принципів та відповідних алгоритмів формування та використання бінарного дерева з фіксованою висотою для прискорення пошуку елемента в масиві та визначення *i*-ї порядкової статистики, зокрема, медіани масиву.

1 ПОСТАНОВКА ЗАДАЧІ

Нехай потрібно віднайти медіану вхідного масиву $X = \langle x_0, x_1, \ldots, x_{N-1} \rangle$ з N елементів. Очевидно, що в процесі визначення медіани доведеться використовувати додаткову пам'ять для сортування елементів копії масиву чи зберігання динамічних структур (пірамід, бінарного дерева) щоб не спотворити початковий масив. Зрозуміло також, що для визначення медіани масиву можливо використати стандартний метод сортування оболонки мови програмування (наприклад, для C# [8] – це метод Array.Sort()) чи один з швидких методів сортування [9] та обрати елементи, які після цього опиняться посередині. Але таке визначення буде тривалим, оскільки під час сортування будуть впорядковуватися всі елементи вхідного масиву, а нам потрібно лише дізнатися, які значення опиняться після сортування посередині. Ми проаналізуємо ефективність різних методів визначення медіани масиву не лише однократно, а й після послідовної зміни кожного елемента.

Дослідимо також ефективність різних методів для визначення медіан впорядкованої за зростанням довжини послідовності неперервних підмасивів $Y_i = \langle x_0, x_1, \ldots, x_{i-1}, x_i \rangle$, $i = \overline{0, N-1}$, де наступний підмасив отримується з попереднього підмасиву за допомогою доповнення його новим елементом. Очевидно, що визначення медіани чергового підмасиву прискориться, якщо використати впорядковані дані попереднього підмасиву, доповнивши їх новим елементом, і тут ефективними можуть виявитися інші методи, ніж для визначення медіани масиву X.

Крім цього, проаналізуємо ефективність різних методів прискорення обробки елементів масивів для

DOI 10.15588/1607-3274-2025-1-18

подібної до Y_i послідовності вкладених підмасивів $Z_i = \langle x_0, x_1, \ldots, x_{i-1}, x_i \rangle$, $i = \overline{N-1, 0}$, впорядкованої за спаданням їх довжини. Тут наступний підмасив отримується з попереднього за допомогою вилучення його останнього елемента. Такі послідовності використовуються нами в процесі прогресуючого ієрархічного стиснення зображень без втрат [10], під час якого після кожного вилучення максимального елемента масиву ще й змінюються значення двох інших його елементів, що призводить до модифікації пов'язаних динамічних структур.

2 ОГЛЯД ЛІТЕРАТУРИ

На практиці для визначення медіани масиву найчастіше застосовують розбиття C. A. R. Hoare [11]. Цей метод діє за принципом «Розділяй і володарюй»: серед елементів копії масиву обирається опорний елемент і всі елементи, не більші за нього, переміщуються в масиві лівіше від цього елемента, а не менші – правіше. Якщо після перестановок опорний елемент опинився посередині масиву (для масивів з непарною довжиною) чи в одній з центральних позицій (для масивів з парною довжиною), то обчислюють медіану масиву з участю цього опорного елемента. Інакше, якщо опорний елемент виявився правіше від центру, то аналогічно продовжують розбиття лівіше від нього. В протилежному випадку виконують розбиття елементів, розміщених справа від опорного елемента [1]. Елементи, розміщені з іншого боку від опорного елемента, на відміну від швидких алгоритмів сортування [11], надалі не аналізуються, тому середня обчислювальна складність такого алгоритму пошуку медіан менше $O(N \times \log(N))$. На сьогодні також розроблені інші методи пошуку медіани масиву, які мають теоретичну обчислювальну складність порядку $O(N)$ [1], але на практиці їх реалізації виявляються набагато складнішими від розбиття Hoare. Всі ці методи орієнтовані на однократний пошук медіани і результати їх роботи несуттєво впливають на прискорення наступних пошуків медіан після зміни значень елементів масиву.

З іншого боку, розбиття Hoare не завжди є найшвидшим варіантом для пошуку медіан підмасивів. Наприклад, для впорядкованої за зростанням довжини послідовності неперервних підмасивів Y_i, які на початку містять однакові елементи, найефективнішим є метод двох бінарних пірамід [12]. Цей метод послідовно формує дві піраміди однакового розміру, причому перша з них – незростаюча, містить менші елементи вхідного масиву, а друга – неспадна, містить більші елементи. Тому медіани вкладених підмасивів обчислюються тут лише за допомогою вершин цих пірамід і, можливо, чергового елемента, якщо він ще не має пари і не включений в піраміди. Метод двох бінарних пірамід орієнтований суто на пошук медіан підмасивів з послідовним збільшенням кількості їх елементів і неефективний при частих змінах значень елементів масиву чи для визначення довільної i-ї порядкової статистики.

Для визначення послідовності медіан чи i-тих порядкових статистики вхідного масиву X після кожної зміни значень його елементів на практиці використовують відсортований масив ключів елементів масиву, в який синхронно вносять зміни з використанням бінарного пошуку [9] за таким алгоритмом [12]:

1. Віднайти у відсортованому масиві ключів бінарним пошуком індекс ключа попереднього значення елемента вхідного масиву та записати його у змінну *indexDel*;

2. Віднайти у відсортованому масиві ключів бінарним пошуком індекс ключа, більшого за ключ нового значення елемента вхідного масиву та записати його у змінну *indexInsert*;

3. Якщо *indexInsert* > *indexDel*, то перемістити ключі з позиції *indexDel*+1 по позицію *indexInsert*-1 на один елемент вліво та вставити ключ нового значення елемента у відсортований масив ключів в позицію *indexInsert*-1;

4. Інакше перемістити ключі з позиції *indexDel*-1 по позицію *indexInsert* на один елемент вправо та вставити ключ нового значення елемента у відсортований масив ключів в позицію *indexInsert*.

Приклад синхронних змін масиву ключів для вхідного масиву X наведено на рис. 1. Для спрощення в масиві ключів тут наведені лише відсортовані значення елементів без їх індексів у вхідному масиві, оскільки вони унікальні. У випадку заміни елемента 18 на 60 у відсортованому масиві ключів спочатку шукається індекс попереднього значення ключа (змінна *indexDel*), потім – індекс більшого ключа від ключа нового значення (змінна *indexInsert*), далі ключі з позиції *indexDel*+1 по позицію *indexInsert*-1 переміщуються на один елемент вліво, після чого в позицію *indexInsert*-1 вставляється нове значення ключа (60).

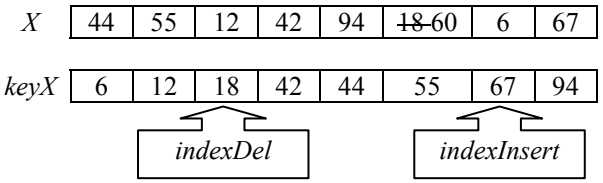

Рисунок 1 – Зміни у відсортованому масиві ключів для масиву X=<44, 55, 12, 42, 94, 18, 6, 67> після заміни елемента 18 на 60

Відсортований масив ключів дає змогу відразу визначити будь-яку i-ту порядкову статистику, хоча й вимагає повсякчасного переміщення ключів між позиціями видалення та вставки. Як зазначалося в [12] і буде підтверджено в цій статті, переміщення елементів лише між цими позиціями, а не двічі від кожної з цих позицій до кінця масиву, суттєво прискорює визначення порядкових статистик.

У цій статті ми деталізуємо новий метод визначення i-х порядкових статистик масивів та підмасивів за допомогою BST з фіксованою висотою та досліди-

DOI 10.15588/1607-3274-2025-1-18

мо його ефективність для визначення медіан. Вперше цей метод був представлений на конференції [13]. В цій праці ми використаємо даний метод також для прискорення обробки елементів одновимірних вкладених впорядкованих за спаданням довжини масивів Z_i в процесі прогресуючого ієрархічного стиснення зображень без втрат [10]: в процесі перерозподілу стиснутих даних між модифікованими DEFLATE-блоками [14] виконується ітеративний вибір та поєднання двох суміжних результуючих блоків, які максимально зменшують довжину стиснутих даних. Після поєднання результуючих блоків до кожного з них окремо застосовується контекстно-незалежне кодування [15]. Чим більше результуючих блоків згенеровано на початку – тим триваліше буде ітеративне поєднання. Під час кожної такої ітерації необхідно:

1. Віднайти два суміжні результуючі блоки, які максимально зменшують довжину стиснутих даних (максимально економлять біти для зберігання стиснутих даних);
2. Поєднати ці блоки в єдиний результуючий блок;
3. Перерахувати економії бітів від поєднання отриманого блоку з попереднім та наступним блоками.

Розглянемо, наприклад, характеристики фрагменту результуючих блоків для умовного вхідного блоку на початку і в кінці чергової ітерації (рис. 2, 3). Зменшення від поєднання суміжних блоків на цих рисунках схематично виведено між суміжними блоками, до яких воно стосується. На початку чергової ітерації максимальне зменшення розміру стиснутих даних (максимальна економія) від поєднання суміжних результуючих блоків (94 біти, виведені на рис. 2 на темно-сірому фоні) може бути досягнуто від поєднання третього та четвертого блоків цього фрагменту (виведені на світло-сірому фоні). Після поєднання цих результуючих блоків (рис. 3) їх загальна кількість зменшилася на один, розмір блоку поєднання зменшився на 94 біти відносно суми розмірів суміжних поєднаних блоків (10766=6020+4840–94, виведений на світло-сірому фоні), але ще й змінилися значення економій від поєднань блоку поєднання з суміжними блоками (42 на 62 та 18 на 8). Причому, як бачимо, значення економій бітів може як збільшитися, якщо суміжні результуючі блоки мають близькі нерівномірності розподілів, так і зменшитися, якщо ці нерівномірності розподілів суттєво різняться.

Розміри результуючих блоків до поєднання	5007	6580	6020	4840	3000	7000
Економії від поєднання суміжних блоків	…	12	42	94	18	6	…

Рисунок 2 – Фрагмент розмірів результуючих блоків та прогнозованих зменшень (економій) від поєднання суміжних блоків умовного вхідного блоку на початку чергової ітерації, бітів

Розміри результуючих блоків після поєднання	5007	6580	10766	3000	7000
Економії від поєднання суміжних блоків	…	12	62	8	6	…

Рисунок 3 – Фрагмент розмірів результуючих блоків та прогнозованих зменшень (економій) від поєднання суміжних блоків умовного вхідного блоку в кінці чергової ітерації, бітів

Після кожної ітерації кількість результуючих блоків зменшується на 1 за умови, що існує ще хоча б одна невід'ємна економія бітів від поєднання суміжних блоків. Нехай початкова кількість результуючих блоків Z_i рівна *countHBlock*. Тоді максимальна кількість таких ітерацій становить *countHBlock* – 1. Якщо під час кожної ітерації виконується лінійний пошук максимальної економії від поєднання суміжних блоків, то загальна кількість порівнянь для таких пошуків у найгіршому випадку становитиме (*countHBlock* – 2) × (*countHBlock* – 1) / 2, тобто маємо квадратичну обчислювальну складність. Чим менші результуючі блоки генеруються під час початкового розбиття вхідного блоку – тим більша їх однорідність, але й тим більше їх буде, і тому довше триватиме їх ітеративне поєднання. Отже, в цій статті розглянемо також варіанти прискорення пошуку максимального зменшення розміру коду від поєднання суміжних результуючих блоків та модифікації цих зменшень внаслідок такого поєднання за допомогою як відсортованого масиву ключів і бінарних пошуків в ньому, так і BST з фіксованою висотою.

3 МАТЕРІАЛИ І МЕТОДИ

Розглянемо особливості реалізації операцій вставки та видалення для BST з фіксованою висотою, пов'язаного з одновимірним масивом, адже пошук та сортування з використанням цього дерева виконуються так само, як і для класичного BST [1]. В кожному вузлі бінарного дерева для його зв'язку з масивом збережемо не лише значення одного з елементів масиву *valueArray*, а й поле *indexArray*, яке вказує на індекс цього елемента в масиві. Ці два поля формують структуру складеного ключа вузла дерева:

```
struct valueNode
 {int valueArray;
  int indexArray; };
```

Відсутність значення в корені дерева позначимо індексом –1. Порівняння ключів двох вузлів виконаємо за відповідними значеннями елементів масиву, а коли вони однакові – за індексами цих елементів в масиві. Мовою C# [8] така функція порівняння ключів може бути записана так:

DOI 10.15588/1607-3274-2025-1-18

```
int cmpNode(valueNode a, valueNode b)
 {if (a.valueArray < b.valueArray ||
     (a.valueArray == b.valueArray &&
      a.indexArray<b.indexArray)) return -1;
  if (a.valueArray > b.valueArray ||
     (a.valueArray == b.valueArray &&
      a.indexArray>b.indexArray)) return 1;
  return 0; }
```

Оскільки в ключ вузла входить його індекс в масиві, то різні елементи масиву ніколи не будуть породжувати однакові ключі, що дає змогу однозначно ідентифікувати їх по вузлах дерева.

Вставляти вузли з елементів масиву чи підмасиву та їх індексів в бінарне дерево будемо, як правило, традиційно [2] – перший вузол записуємо в корінь дерева, а всі інші опрацьовуємо послідовно та рекурсивно: якщо ключ чергового вузла більший значення ключа з поточного вузла дерева, то вставку рекурсивно продовжуємо в праве піддерево, коли ж ключ чергового вузла менший за ключ поточного вузла, то вставку рекурсивно продовжуємо в ліве піддерево. Рекурсивні виклики виконуємо до тих пір, поки поточний вузол дерева не виявиться порожнім, куди й включаємо черговий вузол. Приклад бінарного дерева, сформованого для вхідного масиву X=<44, 55, 12, 42, 94, 18, 6, 67>, наведено на рис. 4. У вузлах дерева тут і надалі наводяться лише значення елементів масиву, оскільки кожне з них є унікальним і дає змогу однозначно визначити індекс цього елемента в масиві. Бачимо, що перший елемент масиву обов'язково потрапляє у вершину дерева, а положення інших елементів залежить від значень попередніх елементів масиву. З елементів цього масиву сформувалося чотирьохрівневе дерево, хоча, якби елементи масиву були впорядковані за зростанням чи спаданням, то кількість рівнів сягнула б восьми.

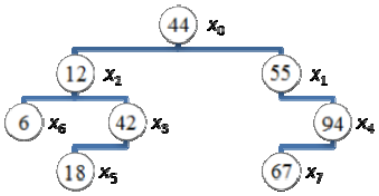

Рисунок 4 – Бінарне дерево, послідовно заповнене елементами масиву X=<44, 55, 12, 42, 94, 18, 6, 67>

Оскільки обчислювальна складність основних операцій над BST пропорційна його висоті [1], яка, в свою чергу, залежить від послідовності опрацювання елементів масиву при формуванні і вставці вузлів, то для прискорення пошуку, вставки і видалення вузлів з бінарного дерева додатково обмежимо його висоту значенням *countLevelTree* (саме тому таке дерево називається бінарним деревом пошуку з фіксованою висотою). Більші значення висоти дерева прискорюють вставку елементів в нього (оскільки з'являється більше вільних вузлів), хоча й збільшують обсяги оперативної пам'яті для його зберігання.

Змінну *countLevelTree* визначатимемо перед використанням BST з фіксованою висотою. Оскільки в бінарне дерево з висотою *countLevelTree* можна помістити максимум $2^{countLevelTree}-1$ вузли, то значення цієї змінної підберемо так, щоб загальна кількість вузлів дерева була неменшою кількості елементів масиву ($2^{countLevelTree}-1 \ge N$). Ми встановлювали змінну *countLevelTree* на одиницю більшою від мінімально можливого значення:

$$countLevelTree = \lceil \log(N+1) \rceil + 1. \qquad (1)$$

З метою уникнення зайвих операцій з виділення та звільнення пам'яті для окремих елементів відразу відведемо пам'ять під *всі можливі вузли* бінарного дерева висоти *countLevelTree*, тобто під всі $2^{countLevelTree}-1$ вузли, більшість з яких залишатимуться вільними. Тоді зберігати значення вузлів бінарного дерева можна в масиві: зліва від елемента $tree_i$ завжди буде міститися елемент $tree_{2i+1}$, а справа – $tree_{2i+2}$. Зрозуміло, що ключ вершини дерева буде міститися в $tree_0$. Змінну *countLevelTree* та кількість вузлів дерева можна збільшувати в процесі експлуатації, якщо BST виявиться заповненим. Для прискорення обробки дерева створимо також масив *indexLeft*, в якому для індекса кожного вузла збережемо індекс підпорядкованого вузла зліва: $indexLeft_i = 2*i+1$. Використовуючи цей масив, індекс підпорядкованого вузла справа буде більшим від індекса підпорядкованого вузла зліва на одиницю.

Бінарне дерево пошуку з фіксованою висотою подібне до збалансованого дерева [2], але в ньому відразу зарезервовано місце під всі можливі вузли, а операція балансування (обернення дерева) зводиться до переміщення значень вузлів без модифікацій вказівок на піддерева.

Приклад чотирьохрівневого BST з фіксованою висотою, заповненого елементами вхідного масиву X=<44, 55, 12, 42, 94, 18, 6, 67>, наведено на рис. 5.

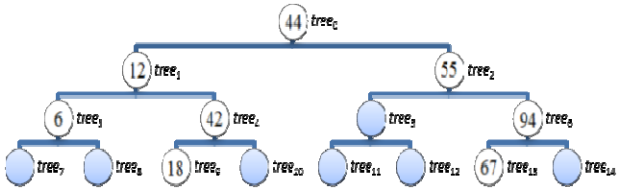

Рисунок 5 – Чотирьохрівневе бінарне дерево з фіксованою висотою, послідовно заповнене елементами масиву X=<44, 55, 12, 42, 94, 18, 6, 67>

Вставка нових значень в бінарне дерево з фіксованою висотою сповільнюється, якщо в обраному напрямку (вліво чи вправо) відсутні вільні вузли. Таке буває, якщо ключ нового значення дерева з чергового елемента масиву менший ключа з поточного вузла дерева, а зліва вільні вузли відсутні, або коли ключ нового значення більший ключа з поточного вузла, а

DOI 10.15588/1607-3274-2025-1-18

справа немає незайнятих вузлів. Тоді у першому з цих випадків звільнимо поточний вузол, вставивши його значення вправо, та порівняємо ключ нового значення з найбільшим значенням ключа зліва (міститься зліва найправіше). Якщо ключ нового значення більший від найбільшого значення ключа зліва, то вставимо в поточний вузол черговий елемент, інакше ж перенесемо в поточний вузол елемент з максимальним значення ключа зліва та рекурсивно продовжимо вставку чергового елемента відносно вузла зліва (це можливо, бо в цьому напрямку ми тільки що звільнили один вузол). Для другого випадку аналогічні дії виконуються дзеркально. Припустимо, наприклад, що в масиві X наступним елементом (під індексом 8) є число 17 і його потрібно вставити в бінарне дерево з фіксованою висотою, наведене на рис. 5. Тоді, рухаючись від вершини бінарного дерева, цей ключ буде послідовно порівнюватися зі значеннями вузлів 44, 12 та 42. Число 17 відносно 42 менше, тому мало б рекурсивно вставлятися відносно вузла зліва (там міститься 18), але зліва вільні вузли відсутні. Тому значення 42 рекурсивно вставляється відносно вузла справа, тим самим звільнюючи поточний вузол, в поточний вузол вставляється найбільше значення зліва (тобто, 18), а зліва рекурсивно вставляється черговий елемент 17 (рис. 6).

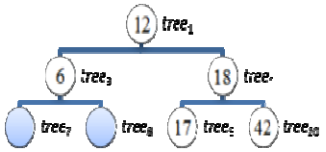

Рисунок 6 – Ліва частина чотирьохрівневого бінарного дерева, сформованого для масиву X=<44, 55, 12, 42, 94, 18, 6, 67> після доповнення його наступним елементом 17

Такі переміщення значень вузлів бінарного дерева забезпечують дотримання основного принципу його побудови: вузли справа від поточного вузла містять більші, а зліва – менші значення ключів.

З метою ідентифікації випадків відсутності вільних вузлів збережемо для кожного вузла BST з фіксованою висотою в цілочисельних масивах *countLeft* та *countRight* відповідно кількості заповнених вузлів зліва і справа, а в масиві *countSub* – кількість створених підпорядкованих вузлів.. Зрозуміло, що значення елементів цих масивів для листків дерева рівне нулю. Співставлення $countLeft_i$ та $countRight_i$ з $countSub_i$ відразу дає змогу визначити можливість вставки значень вузлів без переміщень в обраному напрямку.

Вилучення елементів з BST з фіксованою висотою виконується також традиційно – якщо ключ вузла для вилучення менше ключа чергового вузла, то вилучення виконується рекурсивно відносно підпорядкованого вузла зліва, якщо більше – то відносно вузла справа. Якщо ж вони рівні, то значення вузла замінюється найменшим значенням серед вузлів справа, а при його відсутності – найбільшим серед вузлів зліва.

Побудовані таким чином процедури вилучення і вставки вузлів дають змогу не перевищувати задану висоту BST, а підрахунок кількостей зайнятих вузлів зліва і справа дозволяють швидко не лише вставляти і вилучати елементи, а й визначати будь-яку його i-ту порядкову статистику [5]. Функцію для такого визначення (її перший аргумент – номер порядкової статистики, а другий не вказується) мовою C# подамо так:

```
valueNode valueTree(int index, int startIndex=0)
 {if (index < countLeft[startIndex]) // значення зліва
   return valueTree(index, indexLeft[startIndex]);
  index-=countLeft[startIndex]; // пропускаємо зліва
  if (index == 0) // якщо значення з поточного вузла
   return tree[startIndex];
  // переходимо до вузла справа
  return valueTree(index-1, indexLeft[startIndex]+1);}
```

Таким чином, для визначення довільної порядкової статистики, пошуку та вилучення елемента в BST з фіксованою висотою потрібно не більше 2**countLevelTree* (1) порівнянь, а кількість порівнянь при вставці залежить від розміщення вільних вузлів в дереві і може мати значно більшу обчислювальну складність. Крім цього, при вставці вузлів в дерево потрібно щоразу корегувати кількості зайнятих вузлів зліва чи справа. Отже, найтривалішим етапом при використанні такого дерева є його початкове заповнення. Тому у випадках, коли всі елементи початкового одновимірного масиву відомі до внесення у нього змін, відсортований масив ключів з елементів та індексів цього масиву доцільно відразу рекурсивно записати в дерево пошуку з фіксованою висотою за принципом: в поточний вузол дерева заносимо ключ, що міститься посередині масиву чи підмасиву; ключі підмасиву зліва від цього елемента рекурсивно записуємо в дерево відносно вузла зліва; ключі підмасиву справа від цього елемента рекурсивно записуємо в дерево відносно вузла справа. Такий підхід дає змогу відразу вказати кількості зайнятих вузлів дерева зліва і справа та не виконувати зайвих порівнянь, хоча й вимагає попереднього сортування ключів елементів початкового масиву. Для подальших корегувань початкового масиву потрібно буде лише дописувати та вилучати з дерева відповідні ключі. Зауважимо також, що вилучення попередніх та вставку нових ключів під час зміни елементів вхідного масиву можливо виконувати разом, адже якщо ці два ключі одночасно більші від ключа поточного вузла, то цю операцію можливо рекурсивно продовжити відносно вузла справа, якщо менші – то відносно вузла зліва і при цьому кількості зайнятих вузлів зліва і справа відносно поточного вузла змінювати не потрібно.

4 ЕКСПЕРИМЕНТИ

Перевірку ефективності різних методів однократного визначення медіан масивів ми виконували над масивами X з 10 млн. дійсних чисел. Ці експерименти

DOI 10.15588/1607-3274-2025-1-18

дають змогу оцінити швидкість **побудови** додаткових масивів чи динамічних структур (пірамід або дерев) для знаходження цього показника.

Для визначення послідовності медіан ми також послідовно обробляли 10 тис. вкладених підмасивів Y_i, впорядкованих за зростанням довжини. Такі тести дозволяють співставити швидкості **доповнення** додаткових структур при реалізації різних методів визначення медіан масивів.

Швидкості **модифікацій** додаткових масивів чи динамічних структур з метою визначення медіан масивів після частих змін їх елементів ми оцінювали на масивах X з 5 тис. елементів, послідовно вносячи в них 5 тис. змін. Тобто після ініціалізації вхідного масиву ми послідовно змінювали значення кожного його елемента і після кожної зміни визначали медіану масиву.

Для оцінки ефективності **видалення** та модифікацій додаткових масивів та дерев, що використовуються для прискорення обробки елементів одновимірних масивів, ми порівняли час поєднання результуючих блоків з використанням впорядкованих за спаданням довжини масивів економій бітів Z_i в процесі прогресуючого ієрархічного стиснення зображень відомого тестового набору ACT. Завантажити ці зображення можна, наприклад, з http://www.compression.ca/act/act-files.html. Даний набір містить як синтезовані (№№ 1, 2, 7) так і фотореалістичні (решта) зображення. Вибір саме цього тестового набору зумовлений різноплановістю його зображень та наявністю у відкритих джерелах результатів тестувань на ньому алгоритмів інших дослідників.

Алгоритми розглянутих методів ми реалізували в оболонці Microsoft Visual Studio мовою програмування C# [8]. Для реалізації швидкого сортування ми використали стандартний метод оболонки програмування *Array.Sort*(). Тестування проводилося на комп'ютері з процесором Intel Pentium 4 з тактовою частотою 3 GHz та розміром RAM 4Gb.

5 РЕЗУЛЬТАТИ

Результати тестування алгоритмів різних методів визначень медіан для різних варіантів масивів наведені в табл. 1–3. Тривалість визначення медіан з використанням принципів сортування для масивів, відсортованих за спаданням, як правило більша, ніж для масивів, відсортованих за зростанням, оскільки ці алгоритми орієнтовані на сортування за зростанням.

Таблиця 1 – Тривалості визначень медіан масивів з 10 млн. дійсних чисел алгоритмами різних методів, мс

Метод визначення медіани	Варіант масиву			
	Згенерований випадковим чином	Відсортований за зростанням	Відсортований за спаданням	З однакових елементів
Швидке сортування	3906	1328	2153	2086
Розбиття Hoare	**625**	**273**	**328**	**602**
Бінарні включення по масиву ключів	> 22.8 млн.	6992	> 24 млн.	6953
Дві бінарні піраміди	5718	9521	8833	5030
Бінарне дерево з фіксованою висотою	> 59 млн.	> 345 млн.	> 345 млн.	> 148 млн.
Бінарне дерево по відсортованому масиву	5146	3151	3827	2669

Таблиця 2 – Тривалості знаходжень медіан 10 тис. вкладених підмасивів дійсних чисел алгоритмами різних методів, мс

Метод визначення медіани	Використання попередніх впорядкованих даних	Варіант масиву			
		Згенерований випадковим чином	Відсортований за зростанням	Відсортований за спаданням	З однакових елементів
Швидке сортування	Ні	11414	4714	6605	6145
	Так	4868	3877	6233	5498
Розбиття Hoare	Ні	4118	2678	2477	2516
	Так	1070	1063	1102	828
Бінарні включення по масиву ключів	Так	250	**16**	469	23
Дві бінарні піраміди	Так	**23**	**16**	**16**	**16**
Бінарне дерево з фіксованою висотою	Так	510	3420	3704	35

Таблиця 3 – Тривалості визначень медіан масивів дійсних чисел з 5 тис. елементів після послідовної модифікації кожного елемента алгоритмами різних методів, мс

Метод визначення медіани	Використання попередніх впорядкованих даних	Варіант масиву			
		Згенерований випадковим чином	Відсортований за зростанням	Відсортований за спаданням	З однакових елементів
Швидке сортування	Ні	6296	3882	4799	4540
	Так	3496	2792	2958	2606
Розбиття Hoare	Ні	2400	502	674	709
	Так	642	332	432	498
Бінарні включення по масиву ключів	Так	340	**235**	**329**	126
Дві бінарні піраміди	Так	268	469	470	14
Бінарне дерево з послідовними вилученнями та вставками елементів	Так	38	2198	2341	16
Бінарне дерево з одночасними вилученнями і вставками елементів	Так	**36**	2216	2331	**13**

DOI 10.15588/1607-3274-2025-1-18

Таблиця 4 – Час перерозподілу стиснутих даних між модифікованими DEFLATE-блоками в процесі прогресуючого ієрархічного стиснення зображень набору ACT з використанням різних варіантів алгоритмів прискорення цього процесу, мс

Варіант алгоритму прискорення перерозподілу стиснутих даних між модифікованими DEFLATE-блоками	№ файла								Середній час
	1	2	3	4	5	6	7	8	
Лінійний пошук максимальної економії бітів	330	120	330	470	340	640	30	480	343
Бінарні включення по масиву ключів змінених економій бітів з переміщенням елементів від позицій вилучення та вставки до кінця масиву	260	80	200	220	160	330	30	270	194
Бінарні включення по масиву ключів змінених економій бітів з переміщенням елементів між позиціями вилучення та вставки	240	80	170	220	140	300	20	230	175
Бінарне дерево пошуку з фіксованою висотою та початковою ініціалізацією вставками	250	80	190	250	130	330	30	220	185
Бінарне дерево пошуку з фіксованою висотою та початковою ініціалізацією з відсортованого масиву	**220**	**80**	**160**	**200**	**140**	**280**	**20**	**220**	**165**
Бінарне дерево пошуку з фіксованою висотою та поєднанням вилучень і вставок економій бітів	230	80	170	230	160	270	20	230	174

Час перерозподілу стиснутих даних між модифікованими DEFLATE-блоками в процесі прогресуючого ієрархічного стиснення зображень тестового набору ACT з використанням різних методів прискорення обробки одновимірних масивів наведено в табл. 4.

6 ОБГОВОРЕННЯ

Проаналізуємо спочатку тривалості визначень однократних медіан алгоритмами різних методів (табл. 1). Бачимо, що, як і передбачалося, для однократних визначень медіан окремих великих масивів доцільно використовувати розбиття Hoare. Воно швидше за алгоритми сортування, оскільки не впорядковує елементи масиву повністю, та швидше за алгоритми методів двох бінарних пірамід [12] і описаного в цій статті і в [13] бінарного дерева пошуку з фіксованою висотою, бо не формує ієрархічні структури, а лише переміщує окремі елементи. Формування описаного тут BST по невпорядкованому масиву з 10 млн. дійсних чисел взагалі триває понад 16 год., оскільки вимагає повсякчасних переміщень елементів для забезпечення його фіксованої висоти. Тому початкове бінарне дерево слід формувати по відсортованому масиву, а це не раціонально для однократного визначення медіани, адже після сортування масиву цю характеристику можна визначити відразу. Цікаво, що визначення медіани після бінарних включень ключів для масиву, відсортованого за зростання, триває в 3432 рази швидше, ніж для масиву, відсортованого за спаданням, адже для першого з цих масивів бінарне включення кожного ключа не призводить до переміщення всіх попередніх елементів масиву, а для другого – переміщує всі попередні елементи на один вправо.

Співставимо тепер тривалості визначень медіан 10 тис. підмасивів дійсних чисел, впорядкованих за зростанням їх довжини, алгоритмами різних методів (табл. 2). Бачимо, що для визначення медіан вкладених підмасивів Y_i доцільно використати впорядковані дані попередніх підмасивів, а не опрацьовувати їх спочатку. Формування описаного тут бінарного дерева пошуку для підмасивів, впорядкованих за зростанням чи спаданням триває довше, ніж для підмасивів, згенерованих випадковим чином, оскільки супроводжується постійними переміщеннями елементів для забезпечення його фіксованої висоти. Як і прогнозувалося, найефективнішим та найстабільнішим для визначення медіан таких підмасивів виявився метод двох бінарних пірамід, оскільки для визначення всіх медіан він в середньому використовує лише $N \log N$ порівнянь та стільки ж присвоєнь.

Розглянемо також тривалості знаходжень 5 тис. медіан масивів дійсних чисел такого ж розміру після модифікації кожного елемента (табл. 3). Як і для вкладених підмасивів бачимо, що для визначення медіани масиву після зміни чергового елемента варто відкоригувати попередні впорядковані дані (для алгоритмів сортування – його відсортований аналог, для розбиття Hoare – результати перестановок попереднього масиву, для двох бінарних пірамід та дерева пошуку з фіксованою висотою – ці самі ієрархічні структури для попередніх масивів), а не обробляти елементи щоразу спочатку. Алгоритм методу двох бінарних пірамід показує для невпорядкованих масивів не найкращі результати, оскільки при пошуку елемента для вилучення він може переглядати практично повністю одну з двох пірамід (половину масиву). Найефективнішим методом для визначення медіан після зміни кожного елемента масиву, згенерованого випадковим чином, як і для масиву з однаковими елементами, виявився метод BST з фіксованою висотою та одночасними вилученнями і вставками ключів елементів, адже він в середньому виконує лише $2 \times N \times countLevelTree$ порівнянь. Побудова такого дерева по відсортованому масиву ключів прискорює визначення медіан більш ніж в 7 разів відносно методу двох бінарних пірамід. З іншого боку, для масивів, впорядкованих за зростанням чи спадання, найшвидше визначення медіан продемонстрував метод бінарного включення по масиву ключів, адже він в середньому використовує лише $2 \times N \times \log N$ порівнянь і переміщує елементи між позиціями вилучення та вставки без складнішого корегування ієрархічних структур.

На завершення співставимо тривалості перерозподілу стиснутих даних між модифікованими DEFLATE-блоками в процесі прогресуючого ієрархічного стиснення зображень без втрат (табл. 4). Бачимо, що використання бінарного включення ключів змінених економій бітів в масивах Z_i замість їх лінійного пошуку дало змогу прискорити перерозподіл в середньому на 48.98% (перший та третій рядки табл. 4). З них 5.54% досягається за допомогою переміщення ключів елементів між позиціями вилучення і

DOI 10.15588/1607-3274-2025-1-18

вставки, а не до кінця масиву (другий рядок цієї таблиці). Бінарне дерево пошуку з фіксованою висотою дає змогу додатково прискорити перерозподіл ще на 2.92%, але за умови його початкової ініціалізації з відсортованого масиву (п'ятий рядок табл. 4). Програмне поєднання операцій вилучень і вставки модифікованих економій бітів (шостий рядок даної таблиці) не принесло очікуваного ефекту, оскільки воно призводить до додаткових порівнянь, а відповідні значення найчастіше потрапляють в різні гілки дерева. Зважаючи на ефективність BST з фіксованою висотою, ми рекомендуємо використовувати на практиці саме цю динамічну структуру для визначення *i*-тих порядкових статистик після змін окремих елементів невпорядкованих масивів.

ВИСНОВКИ

1. Наші дослідження вчергове підтвердили недоцільність пошуку «універсальних методів» визначення медіан чи *i*-тих порядкових статистик, ефективних для всіх послідовностей масивів чи підмасивів. Зокрема, при виборі такого методу слід зважати на кількості елементів, якими відрізняються масиви.

2. Для визначення медіан чи *i*-тих порядкових статистик окремих непов'язаних масивів та підмасивів найефективнішим, як і очікувалося, виявилося розбиття Hoare з обміном на великих відстанях, оскільки воно переставляє лише окремі елементи, а не впорядковує масиви повністю.

3. З метою визначення медіан послідовності з N вкладених підмасивів, впорядкованих за зростанням їх довжин, варто застосовувати метод двох бінарних пірамід, адже його реалізації для всіх таких підмасивів в середньому виконують лише $N \log N$ порівнянь та стільки ж присвоєнь, хоча цей метод не придатний для визначення довільних *i*-тих порядкових статистик. Знаходження цих статистик для вкладених підмасивів варто виконувати за допомогою бінарних включень по масиву ключів.

4. Для знаходження медіан чи *i*-тих порядкових статистик після змін чи вилучення елементів невпорядкованого масиву доцільно використовувати бінарне дерево ключів з фіксованою висотою. З цією метою слід спочатку відсортувати ключі елементів масиву, побудувати по них це дерево і після цього послідовно виконувати пов'язані модифікації елементів вхідного масиву та дерева пошуку з фіксованою висотою.

Надалі ми плануємо дослідити ефективність застосування BST з фіксованою висотою в процесі прогресуючого ієрархічного стиснення зображень без втрат після застосування різницевих колірних моделей [16].

ЛІТЕРАТУРА

1. Introduction to Algorithms, Fourth Edition / [T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein]. – Cambridge : MIT Press, 2022. – 1312 p. – Access mode: http://mitpress.mit.edu/9780262046305/introduction-to-algorithms.
2. Шаховська Н. Б. Алгоритми і структури даних : Навчальний посібник / Н. Б. Шаховська, Р. О. Голощук. – Львів : Магнолія-2006, 2020. – 214 с.
3. Silberschatz A. Database system concepts, Seventh edition / A. Silberschatz, H. F. Korth, S. Sudarshan. – New York : McGraw-Hill Education, 2020. ISBN 978-1-260-08450-4.
4. Kerttu P-M. B+-trees [Electronic resource] / P-M. Kerttu. – The Department of Computer Science University of Helsinki. – 9 p. – Access mode: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf.
5. Gentle J. E. Computational Statistics / J. E. Gentle. – Springer, 2009. – 749 p. ISBN 9780387981444. DOI: 10.1007/978-0-387-98144-4.
6. Ageel M. I. The Mean-Median-Mode Inequality for Discrete Unimodal Probability Measure / M. I. Ageel // Far East Journal of Mathematical Sciences. – 2000. – № 2 (2). – P. 187–192.
7. Abadir K M. The mean-median-mode inequality: counter examples / K M. Abadir // Econometric Theory. – 2005. – № 21 (02). – P. 477–482. DOI: 10.1017/S0266466605050267.
8. C# 8.0 draft specification [Electronic resource]. – Microsoft Corporation, 2024. 3769 p. – Access mode: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction.
9. Knuth D. E. The Art of Computer Programming, Vol. 3. Sorting and Searching, Second edition / D. E. Knuth. – Massachusetts : Addison Wesley Longman, 1998. – 791 p.
10. Bomba A. Ya. Redistribution of the Compressed Data Between Modified DEFLATE-Blocks in the Image Compression Process Without Lossless / A. Ya. Bomba, A. V. Shportko, V. A. Postolatii // Computational Linguistics and Intelligent Systems (COLINS 2024) : Proceedings of the 8th International Conference (Lviv, 12–13 april, 2024). Volume II: Modeling, Optimization, and Controlling in Information and Technology Systems Workshop (MOCITSW). – CEUR Workshop Proceedings, 2024. – Vol. 2604. – P. 145–156. Access mode: https://ceur-ws.org/Vol-3668/paper11.pdf.
11. Hoare C. A. R. Quicksort / C. A. R. Hoare // Computer Journal. – 1962. – № 5 (1). – P. 10–16. DOI: 10.1093/comjnl/5.1.10.
12. Shportko A. The Acceleration of the Determination of the Median of Nested Subarrays Using Two Binary Pyramids / A. Shportko, V. Shportko // Computational Linguistics and Intelligent Systems (COLINS 2020) : Proceedings of the 4th International Conference (Lviv, Ukraine, 23–24 april, 2020). – CEUR Workshop Proceedings, 2020. – Vol. 2604. – P. 1102–1116. Access mode: http://ceur-ws.org/Vol-2604/paper72.pdf.
13. Shportko O. The Use of a Fixed Height Binary Tree to Accelerate the Calculation of the Medians of Subarrays / O. Shportko, L. Shportko // Computer Science and Information Technologies (CSIT 2020) : Proceedings of the XVth International Scientific and Technical Conference (Zbarazh, Ukraine, 23–26 septembr, 2020). – Springer Cham, 2020. – Vol. 2. – P. 46–49. Access mode: https://ieeexplore.ieee.org/document/9321921.
14. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 / P. Deutsch. – Alladin enterprises, 1996. – 15 p. Access mode: https://www.rfc-editor.org/rfc/rfc1951. DOI: 10.17487/rfc1951.
15. Kotha H. D. Review on Lossless Compression Techniques / H. D. Kotha, M. Tummanapally, V. K. Upadhyay // Journal of Physics. – 2019. – Vol. 1228. DOI: 10.1088/1742-6596/1228/1/012007.
16. Shportko A. V. Programming the Formation of Difference Color Models for Lossless Image Compression / A. V. Shportko, A. Ya. Bomba, V. A. Postolatii // Computational Linguistics and Intelligent Systems (COLINS 2023) : Proceedings of the 7th International Conference (Kharkiv, Ukraine, 20–21 april, 2023). – CEUR Workshop Proceedings, 2023. – Vol. 3. – P. 53–68. Access mode: http://ceur-ws.org/Vol-3403/paper5.pdf.

Received 12.08.2024.
Accepted 05.12.2024.

DOI 10.15588/1607-3274-2025-1-18

UDC 004.62

APPLICATION OF BINARY SEARCH TREE WITH FIXED HEIGHT TO ACCELERATE PROCESSING OF ONE-DIMENSIONAL ARRAYS

Shportko A. V. – PhD, Associate Professor, Associate Professor of the Department of Information Systems and Computing Methods of the Private Higher Educational Institution "International University of Economics and Humanities named after Academician Stepan Demianchuk", Rivne, Ukraine.

Bomba A. Ya. – Dr. Sc., Professor, Professor of the Department of Computer Sciences and Applied Mathematics of the National University of Water and Environmental Engineering, Rivne, Ukraine.

ABSTRACT

Topicality. Nowadays, binary search trees are widely used to speed up searching, sorting, and selecting array elements. But the computational complexity of searching using a binary tree is proportional to its height, which depends on the sequence of processing the elements of the array. In order to reduce the height of a tree, its balancing is periodically carried out, which is a long process,, thus, the development of alternative methods of controlling the height of a binary tree is currently an actual scientific task.

Objective. Development of algorithms for the formation and use of a binary tree with a fixed height to accelerate the search for an element in an array and to determine arbitrary *i*-th order statistics, in particular, the median of the array.

Method. In this study, it is proposed to set the fixed height of the binary search tree by one greater than the minimum possible height of the binary tree to accommodate all the elements of the array because increasing the fixed height leads to excessive RAM consumption, and decreasing it slows down tree modifications. The formation of such trees is similar to the balancing of trees but, unlike it, the recursive movement of nodes in them is performed only when the corresponding subtree is completely filled. For a binary search tree with a fixed height, RAM is allocated once when it is created, immediately under all possible nodes of a binary tree with a given height. This allows to avoid allocating and freeing memory for each node of the tree and store the values of the nodes in a one-dimensional array without using pointers.

The results. Our experiments showed that in order to speed up the search of elements and to determine the *i*-th order statistics of frequently changing unordered arrays, it is advisable to additionally form a binary search tree with a fixed height. To initialize this tree, it is advisable to use a sorted copy of the keys of the array elements, and not to insert them one by one. For example, the use of a binary tree with a fixed height accelerates the search of medians of such arrays by more than 7 times compared to the method of two binary pyramids and additionally accelerates the redistribution of compressed data between modified DEFLATE-blocks in the process of progressive hierarchical lossless compression of images of the ACT set by an average of 2.92%.

Conclusions. To determine medians or *i*-th order statistics of individual unrelated arrays and subarrays, instead of known sorting methods, it is advisable to use Hoare partitioning with exchange over long distances as it rearranges only individual elements and does not order the entire array completely. In order to determine the medians of the sequence of nested subarrays, ordered by the growth of their length, it is worth using the method of two binary pyramids because they are oriented to rapid addition of new elements. To find medians or *i*-th order statistics after changes or removal of elements of an unordered array, it is advisable to use a binary search tree for the keys of array elements with a fixed height as such fixing prevents uncontrolled growth of the number of comparison operations and makes it possible to process the tree without using instructions.

KEYWORDS: array sorting, array medians, method of two binary pyramids, binary search tree with fixed height.

REFERENCES

1. Cormen T. H. Leiserson C. E. , Rivest R. L. , Stein C. . Introduction to Algorithms, Fourth Edition. Cambridge, MIT Press, 2022, 1312 p. Access mode: http://mitpress.mit.edu/9780262046305/introduction-to-algorithms.
2. Shakhovska N. B., Holoshchuk R. O. Alhorytmy i struktury danykh : Navchalnyi posibnyk. Lviv, Mahnoliia-2006, 2020, 214 p.
3. Silberschatz A., Korth H. F., Sudarshan S. Database system concepts, Seventh edition. New York, McGraw-Hill Education, 2020. ISBN 978-1-260-08450-4.
4. Kerttu P-M. B+-trees [Electronic resource] / P-M. Kerttu. – The Department of Computer Science University of Helsinki, 9 p. Access mode: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf.
5. Gentle J. E. Computational Statistics. Springer, 2009, 749 p. ISBN 9780387981444. DOI: 10.1007/978-0-387-98144-4.
6. Ageel M. I. The Mean-Median-Mode Inequality for Discrete Unimodal Probability Measure, *Far East Journal of Mathematical Sciences*, 2000, № 2 (2), pp. 187–192.
7. Abadir K M. The mean-median-mode inequality: counter examples, *Econometric Theory,* 2005, № 21 (02), pp. 477–482. DOI: 10.1017/S0266466605050267.
8. C# 8.0 draft specification [Electronic resource]. – Microsoft Corporation, 2024, 3769 p. Access mode: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction.
9. Knuth D. E. The Art of Computer Programming, Vol. 3. Sorting and Searching, Second edition. Massachusetts, Addison Wesley Longman, 1998, 791 p.
10. Bomba A. Ya., Shportko A. V. , Postolatii V. A. Redistribution of the Compressed Data Between Modified DEFLATE-Blocks in the Image Compression Process Without Lossless, *Computational Linguistics and Intelligent Systems (COLINS 2024) : Proceedings of the 8th International Conference (Lviv, 12–13 april, 2024). Volume II: Modeling, Optimization, and Controlling in Information and Technology Systems Workshop (MOCITSW)*. CEUR Workshop Proceedings, 2024, Vol. 2604, pp. 145–156. Access mode: https://ceur-ws.org/Vol-3668/paper11.pdf.
11. Hoare C. A. R. Quicksort, *Computer Journal,* 1962, № 5 (1), pp. 10–16. DOI: 10.1093/comjnl/5.1.10.
12. Shportko A., Shportko V. The Acceleration of the Determination of the Median of Nested Subarrays Using Two Binary Pyramids, *Computational Linguistics and Intelligent Systems (COLINS 2020) : Proceedings of the 4th International Conference (Lviv, Ukraine, 23–24 april, 2020)*, CEUR Workshop Proceedings, 2020, Vol. 2604, pp. 1102–1116. Access mode: http://ceur-ws.org/Vol-2604/paper72.pdf.
13. Shportko O., Shportko L. The Use of a Fixed Height Binary Tree to Accelerate the Calculation of the Medians of Subarrays, *Computer Science and Information Technologies (CSIT 2020) : Proceedings of the XVth International Scientific and Technical Conference (Zbarazh, Ukraine, 23–26 septembr, 2020)*. Springer Cham, 2020, Vol. 2, pp. 46–49. Access mode: https://ieeexplore.ieee.org/document/9321921.
14. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. Alladin enterprises, 1996, 15 p. Access mode: https://www.rfc-editor.org/rfc/rfc1951. DOI: 10.17487/rfc1951.
15. Kotha H. D., Tummanapally M. , Upadhyay V. K. Review on Lossless Compression Techniques, *Journal of Physics*, 2019, Vol. 1228. DOI: 10.1088/1742-6596/1228/1/012007.
16. Shportko A. V., Bomba A. Ya , Postolatii V. A. Programming the Formation of Difference Color Models for Lossless Image Compression, *Computational Linguistics and Intelligent Systems (COLINS 2023) : Proceedings of the 7th International Conference (Kharkiv, Ukraine, 20–21 april, 2023)*. CEUR Workshop Proceedings, 2023, Vol. 3, pp. 53–68. Access mode: http://ceur-ws.org/Vol-3403/paper5.pdf.

DOI 10.15588/1607-3274-2025-1-18