

А. В. Шостак

Національний аерокосмічний університет «Харківський авіаційний інститут», Харків, Україна

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МЕТОДІВ ПОШУКУ В АВЛ-ДЕРЕВІ З УРАХУВАННЯМ ІЄРАРХІЇ ПАМ'ЯТІ

Анотація. Актуальність. Дослідження впливу просторової локальності даних, передбачення розгалужень та програмного попереднього завантаження даних на продуктивність пошуку становить значний інтерес при розробці високопродуктивних структур даних. **Предмет дослідження:** методи пошуку в АВЛ-дереві, представленому у вигляді масиву з використанням розміщення Ейтцингера, та вплив ієрархії пам'яті процесора на їхню швидкість. **Мета дослідження:** експериментально оцінити ефективність різних методів пошуку в АВЛ-дереві з поданням на масивах залежно від розміру масиву та положення робочого набору даних щодо рівнів ієрархії пам'яті процесора. **Завдання дослідження.** Реалізувати та експериментально дослідити алгоритми пошуку Binary, AVL, Eytzinger, EytzingerB та EytzingerBP; визначити діапазони розмірів масивів, які відповідають різним рівням ієрархії пам'яті; порівняти швидкість досліджуваних алгоритмів; встановити вплив просторової локальності даних, помилок передбачення розгалужень та програмного попереднього завантаження даних на ефективність пошуку. **Методи дослідження.** Використані методи алгоритмічного аналізу, експериментального дослідження продуктивності програмного забезпечення та статистичної обробки результатів вимірів. Експериментальна оцінка виконана засобами BenchmarkDotNet на платформі .NET 8 з використанням апаратних лічильників продуктивності процесора. **Результати дослідження.** Встановлено, що ефективність алгоритмів пошуку визначається як їх обчислювальна складність, а й особливостями взаємодії з ієрархією пам'яті процесора. Показано, що при розміщенні робочих даних у кешах L1/L2 найменший час пошуку забезпечує алгоритм EytzingerB, тоді як при переході до даних, розміщених у кешах L3 та оперативної пам'яті, найкращі результати демонструє алгоритм EytzingerBP, який використовує програмне попереднє завантаження даних. Отримано експериментальні залежності часу пошуку від розміру масиву, визначено галузі ефективного застосування досліджуваних алгоритмів та кількісно оцінено вираз найефективніших методів пошуку. Експериментально встановлено зміна найефективнішого алгоритму пошуку при переході робочих даних між рівнями ієрархії пам'яті процесора: у діапазоні L1/L2 мінімальний час пошуку забезпечує алгоритм EytzingerB, тоді як у діапазоні L3 та оперативної пам'яті лідирує алгоритм EytzingerBP. **Висновки.** Подання АВЛ-дерева в масиві Ейтцингера забезпечує істотне підвищення швидкості пошуку в порівнянні з класичною реалізацією дерева на покажчиках. Експериментально підтверджено, що вибір оптимального методу пошуку визначається рівнем ієрархії пам'яті, в якому знаходиться робочий набір даних. Отримані результати можуть бути використані для проектування високопродуктивних пошукових структур даних та оптимізації алгоритмів пошуку для сучасних багаторівневих підсистем пам'яті.

Ключові слова: АВЛ-дерево, розміщення Ейтцингера, алгоритми пошуку, програмне попереднє завантаження даних, кеш-пам'ять, локальність даних, BenchmarkDotNet, експериментальна оцінка продуктивності.

1. Вступ

1.1 Актуальність тематики

Пошук елементів у впорядкованих структурах даних є однією з фундаментальних операцій сучасної інформатики та широко використовується при побудові систем керування базами даних, інформаційно-пошукових систем, компіляторів, файлових систем, систем підтримки прийняття рішень та інших програмних комплексів [1–3]. Ефективність виконання операцій пошуку багато в чому визначає загальну продуктивність подібних систем, що зумовлює постійний інтерес до розробки та вдосконалення алгоритмів пошуку.

Класичні методи аналізу алгоритмів оцінюють ефективність пошуку переважно за асимптотичною обчислювальною складністю. Для бінарного пошуку, пошуку в АВЛ-дереві та інших збалансованих структур даних складність пошуку становить $O(\log n)$ [3, 4]. Проте розвиток архітектури сучасних процесорів призвело до того, що однакова асимптотична складність алгоритмів не гарантує однакового швидкодії за її практичної реалізації.

Сучасні обчислювальні системи використовують багаторівневу ієрархію пам'яті, що включає кеші першого (L1), другого (L2) та третього (L3) рівнів, а

також оперативну пам'ять. Час звернення до різних рівнів пам'яті відрізняється на порядки, внаслідок чого ефективність алгоритму все більшою мірою визначається просторовою та тимчасовою локальністю доступу до даних, кількістю промахів кешу, ефективністю апаратного прогнозування розгалужень та можливістю попереднього завантаження даних. За таких умов дослідження взаємодії алгоритмів пошуку з підсистемою пам'яті стає не менш важливим, ніж аналіз їх обчислювальної складності.

Одним із перспективних напрямів підвищення продуктивності пошуку є представлення дерев пошуку у вигляді масиву з використанням розміщення Ейтцингера. Таке уявлення забезпечує послідовне розміщення вузлів дерева в пам'яті, покращує просторову локальність доступу до даних і дозволяє ефективніше використовувати механізми кешування та апаратної попередньої вибірки процесора. Додаткове підвищення швидкодії може бути досягнуто за рахунок застосування алгоритмів пошуку без умовних переходів та програмного попереднього завантаження даних.

У зв'язку з цим експериментальне дослідження впливу різних способів організації пошуку у масивному уявленні АВЛ-дерева на сучасних процесорах є актуальним науково-практичним завданням.

1.2 Аналіз літературних джерел і визначення проблеми

Теоретичні основи алгоритмів пошуку та збалансованих дерев докладно викладені у фундаментальних роботах [1-4], де розглядаються властивості бінарного пошуку, бінарних дерев пошуку та AVL-дерев, аналізується їх обчислювальна складність та доводяться основні оцінки ефективності.

Сучасні підходи до реалізації AVL-дерев та особливостей їх практичного застосування представлені також у роботі [5]. Більшість класичних досліджень розглядає ефективність алгоритмів переважно з позицій кількості операцій, що виконуються, припускаючи однакову вартість звернення до пам'яті. Проте це припущення не відповідає архітектурі сучасних обчислювальних систем, у яких час доступу до різних рівнів пам'яті істотно відрізняється. Вплив архітектури пам'яті на продуктивність програм детально розглядається у працях [6, 7], де показано, що локальність доступу до даних та організація кеш-пам'яті мають визначальний вплив на практичну продуктивність алгоритмів. Аналогічні питання оптимізації програм для сучасних процесорів детально висвітлено в посібниках Intel [8–10], а також у працях [11–14], присвячених особливостям мікроархітектур сучасних процесорів, механізмів передбачення розгалужень та оптимізації доступу до пам'яті. Одним із найважливіших досліджень, присвячених впливу розміщення даних на ефективність пошуку, є робота [15]. Автори показали, що представлення бінарного дерева пошуку в масиві з використанням розміщення Ейтцингера дозволяє значно підвищити продуктивність пошуку порівняно з традиційним бінарним пошуком завдяки поліпшенню просторової локальності даних та використанню програмного попереднього завантаження. Однак отримані результати стосуються переважно ідеального бінарного дерева пошуку і не враховують особливості застосування цих підходів до AVL-дерев. У попередніх дослідженнях автора проведено порівняльний аналіз класичної реалізації AVL-дерев на показниках та масивного представлення AVL-дерев [16]. Отримані результати підтвердили істотну перевагу масивного представлення за швидкістю пошуку.

Незважаючи на значну кількість робіт у цій галузі, питання порівняльної ефективності різних методів пошуку в масивному представленні AVL-дерев при зміні положення робочого набору даних відносно рівнів ієрархії пам'яті процесора залишається недостатньо дослідженим. Практично відсутні експериментальні дослідження, що дозволяють визначити діапазони ефективного застосування класичного пошуку Eytzinger, його branchless-модифікації та алгоритмів із програмним попереднім завантаженням даних. Розв'язання цієї задачі представляє інтерес як з теоретичної точки зору, так і для практичного проектування високопродуктивних структур даних.

1.3 Мета і завдання дослідження

Метою дослідження є експериментальна оцінка ефективності методів пошуку в AVL-дереві,

представленому у вигляді масиву з використанням розміщення Ейтцингера, з урахуванням впливу ієрархії пам'яті сучасних процесорів. Для досягнення поставленої мети було сформульовано такі завдання дослідження:

1. Реалізувати методи пошуку Binary, AVL, Eytzinger, Eytzinger Branchless (EytzingerB) та Eytzinger Prefetch (EytzingerBP) у єдиному програмному середовищі.

2. Виконати експериментальне дослідження швидкодії зазначених методів для різних розмірів масивів, що відповідають розміщенню робочих даних у кешах L1, L2, L3 та оперативної пам'яті.

3. Провести порівняльний аналіз часу пошуку у сценаріях успішного (Hit) та неуспішного (Miss) пошуку.

4. Дослідити вплив просторової локальності даних, кількості промахів кешу та помилок передбачення розгалужень на продуктивність аналізованих алгоритмів пошуку з використанням апаратних лічильників продуктивності процесора.

5. Визначити сфери ефективного застосування досліджуваних методів пошуку в залежності від положення робочого набору даних щодо рівнів ієрархії пам'яті процесора.

2. Матеріали і методи дослідження

Об'єктом дослідження було AVL-дерево, представлене у вигляді одновимірного масиву з використанням розміщення Ейтцингера. На відміну від класичної реалізації AVL-дерев на основі динамічно пов'язаних вузлів, при такому представленні вершини дерева розташовуються послідовно в масиві у порядку обходу за рівнями. Для вершини з індексом i лівий нащадок розташовується за індексом $(2i+1)$, а правий — за індексом $(2i+2)$. Таке розміщення забезпечує високу просторову локальність даних і дозволяє ефективно використовувати механізми кешування сучасних процесорів.

Для формування досліджуваних структур використовувалося канонічне AVL-дерево, побудоване з попередньо відсортованого масиву унікальних цілих чисел. Побудова здійснювалася шляхом рекурсивного вибору медіанного елемента кожного підмасиву, що забезпечувало мінімально можливу висоту дерева та відповідало класичній структурі збалансованого бінарного дерева пошуку. Після побудови дерева виконувалося його перетворення в масив Ейтцингера шляхом обходу по рівнях.

У роботі досліджено п'ять методів пошуку:

- Binary – класичний бінарний пошук у відсортованому масиві [1, 2];

- AVL – пошук у класичному AVL-дереві, реалізованому на показниках [3–5];

- Eytzinger – пошук у масивному представленні AVL-дерев з розміщенням Ейтцингера [15];

- EytzingerB – модифікація пошуку Eytzinger, що мінімізує кількість умовних переходів за рахунок використання арифметичних операцій під час обчислення напрямку переходу [17–19];

- EytzingerBP – модифікація методу EytzingerB, що використовує програмне попереднє завантаження

даних для зменшення латентності звернень до пам'яті [20, 21].

Усі алгоритми було реалізовано мовою C# із використанням платформи .NET 8. Експериментальні дослідження проводилися за допомогою бібліотеки BenchmarkDotNet v0.15.8, яка забезпечує автоматичну організацію вимірювань, попереднє прогрівання середовища виконання, багаторазове повторення експериментів та статистичну обробку результатів.

Вимірювання проводилися на персональному комп'ютері з такими характеристиками: процесор Intel Core i5-12400; оперативна пам'ять – 32 ГБ; операційна система Microsoft Windows 10 x64.

Для оцінки впливу ієрархії пам'яті використовувалися масиви 32-бітних ключів різних розмірів, що дозволяли послідовно переносити робочий набір даних із кеш-пам'яті першого рівня (L1) у кеші L2 та L3, а потім в оперативну пам'ять. Розміри досліджуваних масивів обиралися з урахуванням характеристик кеш-пам'яті використовуваного процесора та наведені в табл. 1 [7, 8, 13]. Для кожного розміру масиву виконувалися два сценарії пошуку: - Hit – пошук елемента, який є в дереві; - Miss – пошук елемента, якого немає.

Таблиця 1 – Відповідність розмірів досліджуваних масивів рівням ієрархії пам'яті

№	Розмірність масиву	Куди поміщається
1	500	L1
2	16000	L1/L2
3	500000	L1/L2/L3
4	2000000	L1/L2/L3
5	8000000	L1/L2/L3/RAM
6	16000000	L1/L2/L3/RAM
7	32000000	L1/L2/L3/RAM
8	64000000	L1/L2/L3/RAM

Для підвищення статистичної достовірності використовували 10 ітерацій прогрівання середовища виконання та 30 вимірювальних ітерацій BenchmarkDotNet. Підсумкові результати подавалися у вигляді середнього часу виконання операції пошуку, стандартного відхилення та довірчого інтервалу. Окрім часу пошуку, аналізувалися апаратні характеристики виконання алгоритмів із використанням апаратних лічильників продуктивності процесора. Для кожного алгоритму реєструвалися такі показники: – загальна кількість процесорних циклів на одну операцію пошуку; – кількість промахів кеш-пам'яті на одну операцію (CacheMisses/Op); – кількість помилок передбачення розгалужень на одну операцію (BranchMispredictions/Op). Використання апаратних лічильників дозволило зіставити експериментально вимірний час пошуку з особливостями роботи підсистеми пам'яті та механізму передбачення розгалужень процесора, що дало змогу пояснити спостережувані відмінності в продуктивності досліджуваних методів.

Порівняння методів здійснювалося за середнім часом виконання операції пошуку, а також за

відносним прискоренням порівняно з класичним бінарним пошуком і класичною реалізацією АВЛ-дерева на покажчиках. Додатково аналізувалася зміна ефективності алгоритмів при збільшенні розміру масиву та перенесенні робочих даних між рівнями ієрархії пам'яті процесора.

У роботі перевірялося припущення про те, що різні методи пошуку в масивному представленні АВЛ-дерева мають різну чутливість до характеристик ієрархії пам'яті процесора. Внаслідок цього при зміні обсягу робочого набору даних найбільш ефективний метод пошуку може змінюватися. Визначення областей ефективного застосування кожного з досліджуваних методів було одним із основних завдань даного дослідження.

3. Результати дослідження

Для оцінки ефективності досліджуваних методів пошуку було проведено серію обчислювальних експериментів для масивів різного обсягу. Для кожного розміру масиву вимірювався середній час пошуку існуючого елемента (сценарій Hit) та відсутнього елемента (сценарій Miss).

На рис. 1, 2 представлено залежність середнього часу пошуку від розміру масиву в логарифмічному масштабі для досліджуваних алгоритмів. У табл. 2, 3 наведено чисельні результати експериментів.

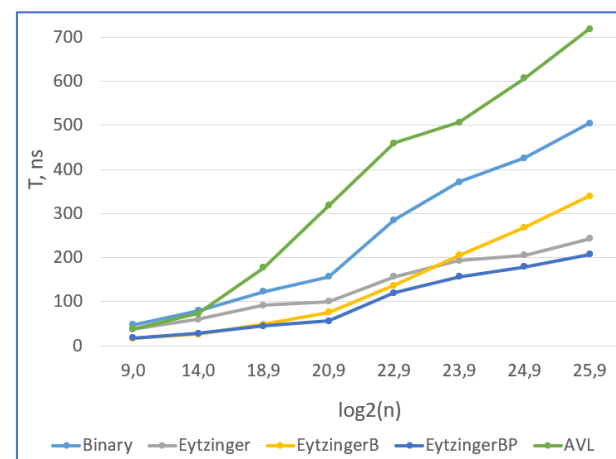


Рис. 1. Залежність часу пошуку T від розміру масиву n (сценарій Hit)

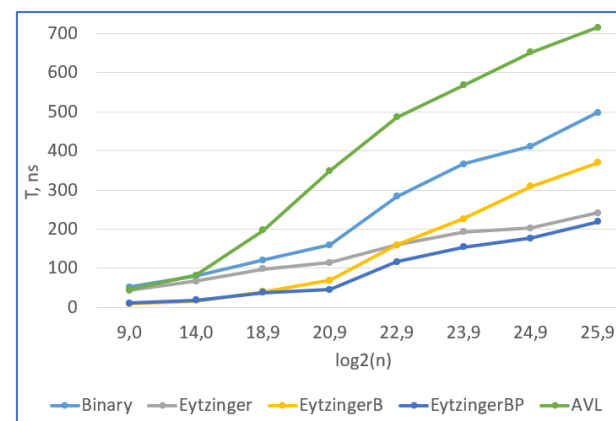


Рис. 2. Залежність часу пошуку T від розміру масиву n (сценарій Miss)

На рис. 3 наведено залежність часу пошуку досліджуваних алгоритмів пошуку відносно часу пошуку алгоритму Binary від розмірності масивів у логарифмічному масштабі для сценарію Hit.

У табл. 2 наведено мінімальний час пошуку для сценаріїв Hit і Miss серед усіх розглянутих алгоритмів пошуку.

Для масивів, що повністю розміщуються в кешах L1 і L2, мінімальний час пошуку забезпечує алгоритм EytzingerB (рис. 1–3, табл. 2). Перевага алгоритму EytzingerB для розмірів $n=500$, 16000 пов'язана з тим, що в цьому випадку масив з даними вже повністю міститься в кешах L1 або L2, вартість доступу до пам'яті є незначною, тому програмне попереднє завантаження не встигає принести користь і лише збільшує кількість виконуваних інструкцій.

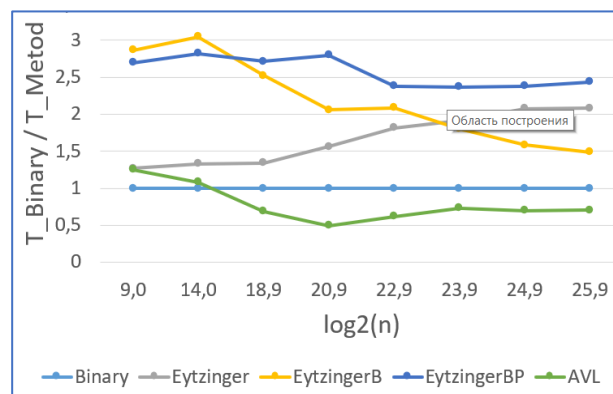


Рис. 3. Відносна залежність часу пошуку (Binary=1) від розміру масиву n

Таблиця 2 – Мінімальний час пошуку T залежно від розміру масивів n

n	Метод	Сценарій	T, ns
500	EytzingerB	Hit	16,684
		Miss	8,367
16000	EytzingerB	Hit	25,998
		Miss	16,024
500000	EytzingerBP	Hit	45,14
		Miss	37,25
2000000	EytzingerBP	Hit	55,97
		Miss	45,38
8000000	EytzingerBP	Hit	119,70
		Miss	116,70
16000000	EytzingerBP	Hit	156,80
		Miss	154,40
32000000	EytzingerBP	Hit	178,80
		Miss	177,10
64000000	EytzingerBP	Hit	207,20
		Miss	219,20

У табл. 3 наведено фрагмент результатів дослідження алгоритмів пошуку. Стовець 1 таблиці 3 відповідає алгоритму пошуку, стовець 2 – розміру масиву n, стовець 3 – усередненому часу пошуку T (ns), стовець 4 – значенню апаратного лічильника кількості помилок передбачення розгалуження на операцію, стовець 5 – значенню апаратного лічильника промахів кешу на операцію.

Таблиця 3 – Витяг із результатів дослідження алгоритмів пошуку

1	2	3	4	5
Binary_Hit	2000000	156,56	11	0
Binary_Miss	2000000	159,93	12	0
AVL_Hit	2000000	317,82	10	17
AVL_Miss	2000000	349,01	11	19
Eytzinger_Hit	2000000	100,23	10	0
Eytzinger_Miss	2000000	113,63	11	0
EytzingerB_Hit	2000000	75,99	1	0
EytzingerB_Miss	2000000	69,29	0	0
EytzingerBP_Hit	2000000	55,97	1	0
EytzingerBP_Miss	2000000	45,38	0	0
Binary_Hit	8000000	285,08	12	33
Binary_Miss	8000000	283,58	12	33
AVL_Hit	8000000	460,07	11	26
AVL_Miss	8000000	486,22	12	28
Eytzinger_Hit	8000000	156,94	11	5
Eytzinger_Miss	8000000	159,94	12	5
EytzingerB_Hit	8000000	136,36	1	1
EytzingerB_Miss	8000000	159,62	0	2
EytzingerBP_Hit	8000000	119,70	5	1
EytzingerBP_Miss	8000000	116,70	6	0
Binary_Hit	16000000	371,13	12	65
Binary_Miss	16000000	365,93	12	65
AVL_Hit	16000000	507,00	12	30
AVL_Miss	16000000	568,13	13	33
Eytzinger_Hit	16000000	193,52	11	11
Eytzinger_Miss	16000000	192,70	12	11
EytzingerB_Hit	16000000	205,53	1	2
EytzingerB_Miss	16000000	225,79	0	3
EytzingerBP_Hit	16000000	156,80	7	1
EytzingerBP_Miss	16000000	154,40	10	0

Для масивів, що розміщуються в кешах L1/L2/L3 та в L1/L2/L3/RAM, мінімальний час пошуку забезпечує алгоритм EytzingerBP (рис. 1–3, табл. 2, 3). При переході до масивів, розміри яких перевищують обсяг кеш-пам'яті L2 та L3, вартість звернення до пам'яті суттєво зростає. У цих умовах алгоритм EytzingerBP дозволяє заздалегідь ініціювати завантаження необхідних кеш-ліній, завдяки чому затримка доступу частково перекривається обчисленнями, що призводить до зменшення загального часу пошуку.

Значне скорочення часу пошуку в діапазонах пам'яті L1/L2/L3 та L1/L2/L3/RAM алгоритму EytzingerBP порівняно з іншими алгоритмами забезпечується програмним попереднім завантаженням даних на 4–5 рівнів дерева вперед відносно поточного вузла. Таке попереднє завантаження даних у зазначеному діапазоні пам'яті мінімізує час доступу до пам'яті. Ефективність такого завантаження при реалізації алгоритму EytzingerBP підтверджується практично повною відсутністю промахів кешу (стовпець 5 табл. 3) та відсутністю або незначною кількістю помилок передбачення розгалужень (стовпець 4 табл. 3).

У всіх досліджених діапазонах розмірів масивів класичні алгоритми AVL і Binary демонструють найбільший час виконання серед розглянутих методів.

Для виявлення причин відмінностей у продуктивності досліджуваних методів поряд із часом виконання пошуку були використані апаратні лічильники продуктивності процесора.

На рис. 4 представлено залежність середньої кількості промахів кеш-пам'яті від розміру масиву для сценарію Hit для досліджуваних методів пошуку.

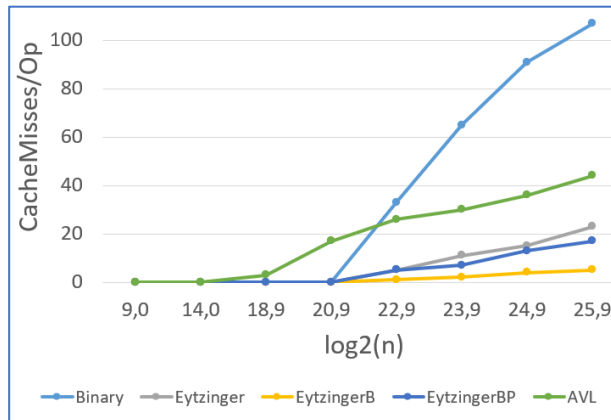


Рис. 4. Залежність середньої кількості промахів кеш-пам'яті від розміру масиву n

Аналіз отриманих результатів показує, що зі збільшенням обсягу масиву кількість промахів кеш-пам'яті закономірно зростає для всіх досліджуваних методів. Найбільш інтенсивне зростання спостерігається після виходу робочого набору даних за межі кеш-пам'яті верхніх рівнів, що відповідає переходу до обслуговування частини звернень із кеш-пам'яті L3 та оперативної пам'яті.

Для алгоритмів AVL і Binary характерна найбільша кількість промахів кеш-пам'яті практично в усьому досліджуваному діапазоні розмірів масивів. Для алгоритму Binary це пояснюється особливостями послідовності звернень до елементів масиву, за якої сусідні за часом звернення відповідають значно віддаленим адресам пам'яті. У результаті просторова локальність використовується недостатньо ефективно, а ймовірність потрапляння потрібних даних у кеш виявляється нижчою.

Розміщення вузлів дерева за схемою Ейтцингера забезпечує більш компактне розташування елементів, що належать до верхніх рівнів дерева, завдяки чому зменшується кількість промахів кеш-пам'яті (рис. 4). Цей ефект найбільш помітний під час обробки масивів великого обсягу, коли вартість звернення до пам'яті починає суттєво перевищувати вартість виконання арифметичних операцій та умовних переходів.

На рис. 5 представлені залежності середньої кількості помилок передбачення розгалужень від розміру масиву для сценарію Hit для досліджуваних методів пошуку (графіки для алгоритмів Eytzinger і EytzingerB збігаються).

Для алгоритмів EytzingerB та EytzingerBP кількість помилкових прогнозів розгалужень зменшується практично на порядок порівняно з іншими реалізаціями пошуку (рис. 5). Отримані результати підтверджують, що усунення умовних

переходів дозволяє істотно знизити вплив механізму прогнозування розгалужень процесора. Разом з тим зменшення кількості помилкових прогнозів не завжди супроводжується пропорційним скороченням часу пошуку, оскільки під час роботи з великими масивами визначальним фактором стає латентність доступу до пам'яті.

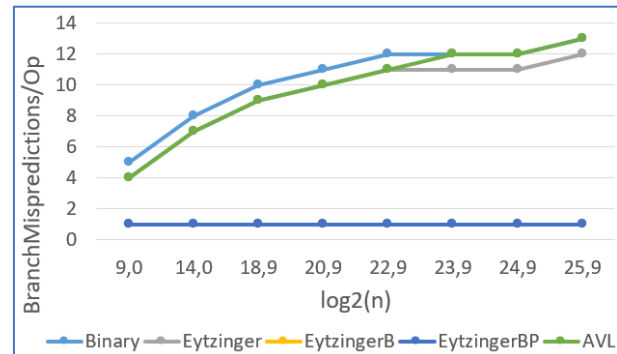


Рис. 5. Залежність середньої кількості помилок прогнозування розгалужень від розміру масиву n

Дослідження методу EytzingerBP із програмним попереднім завантаженням показує, що збільшення кількості звернень до пам'яті, спричинене виконанням інструкцій попереднього завантаження, не призводить до погіршення продуктивності. Навпаки, під час обробки великих масивів спостерігається скорочення загального часу пошуку. Це свідчить про те, що частина затримок звернень до пам'яті успішно перекривається виконанням обчислень, а необхідні дані виявляються завантаженими в кеш до моменту їх використання.

Таким чином, аналіз апаратних лічильників підтверджує, що відмінності в продуктивності досліджуваних методів зумовлені не лише їхньою алгоритмічною організацією, а й особливостями взаємодії з ієрархією пам'яті сучасного процесора. Отримані результати слугують експериментальним підтвердженням того, що просторова локальність розміщення даних, зменшення кількості умовних переходів та використання програмного попереднього завантаження по-різному впливають на продуктивність пошуку залежно від розміру робочого набору даних.

4. Обговорення результатів

Отримані результати підтверджують висунуту в роботі гіпотезу про те, що ступінь впливу різних мікроархітектурних факторів змінюється під час переходу між рівнями ієрархії пам'яті процесора. Відповідно, ефективність конкретного алгоритму визначається не лише його структурою, а й обсягом робочого набору даних відносно розмірів кеш-пам'яті.

Першим чинником, що визначає відмінності в продуктивності, є просторова локальність розміщення даних. У бінарному пошуку послідовні звернення здійснюються до елементів масиву, розташованих на значній відстані один від одного. У міру збільшення розміру масиву це призводить до

зростання кількості промахів кеш-пам'яті та збільшення часу доступу до даних.

Розміщення елементів за схемою Ейтцингера суттєво змінює характер звернень до пам'яті. Вузли верхніх рівнів дерева розташовуються в пам'яті послідовно, завдяки чому елементи, до яких найчастіше звертаються, ефективно розміщуються в кеш-пам'яті процесора. Результати експериментів показують, що саме цей ефект забезпечує стійку перевагу алгоритму Eytzinger порівняно з бінарним пошуком практично на всьому досліджуваному діапазоні розмірів масивів (рис. 1–3).

Другим фактором є вплив механізму передбачення розгалужень. Використання branchless-реалізації практично усуває помилки передбачувача переходів, що підтверджується вимірами апаратного лічильника BranchMispredictions/Op. Якщо для класичного пошуку кількість помилкових передбачень сягає 11–13 на одну операцію пошуку, то для branchless-варіантів цей показник зменшується до одного або практично до нуля (рис. 5).

Однак проведені експерименти показують, що зменшення кількості помилкових прогнозів розгалужень саме по собі не гарантує максимальної продуктивності. При роботі з масивами невеликого обсягу виграш від виключення умовних переходів виявляється обмеженим, оскільки вартість звернень до пам'яті невелика, а основний час визначається виконанням обчислювальних операцій. При подальшому збільшенні обсягу масиву вплив латентності пам'яті стає значно суттєвішим, ніж вартість помилкового передбачення переходу.

Третім фактором є можливість перекриття латентності пам'яті за допомогою програмного попереднього завантаження даних. Експериментальні результати показують, що використання інструкцій попереднього завантаження не забезпечує переваг при розміщенні даних у кеш-пам'яті верхніх рівнів, а в ряді випадків призводить до невеликого збільшення часу пошуку внаслідок появи додаткових інструкцій (табл. 2).

Зовсім інша ситуація спостерігається під час обробки масивів, розміри яких значно перевищують обсяг кеш-пам'яті процесора. У цьому випадку програмне попереднє завантаження дозволяє розпочати перенесення даних у кеш до моменту їх безпосереднього використання. Завдяки перекриттю обчислень з операціями звернення до пам'яті зменшується ефективна латентність завантаження даних, що призводить до скорочення загального часу виконання пошуку.

Для алгоритмів без програмного попереднього завантаження відмінності між сценаріями Hit і Miss виявляються відносно невеликими й визначаються головним чином довжиною пройденого шляху пошуку. Однак для алгоритмів, що використовують програмне попереднє завантаження даних, відмінності стають більш вираженими. Ефективність попереднього завантаження залежить від того, чи будуть попередньо завантажені дані дійсно використані під час наступних звернень. У разі

успішного пошуку частина заздалегідь завантажених вузлів може виявитися непотрібною через дострокове завершення алгоритму, тоді як у разі невдалого пошуку практично весь шлях пошуку проходить до листового рівня, що дозволяє ефективніше використовувати результати попереднього завантаження (рис. 1–3).

Отримані в ході роботи експериментальні дані підтверджують цю закономірність. Для невеликих масивів відмінності між сценаріями Hit і Miss є незначними, оскільки затримки звернення до пам'яті невеликі й практично повністю компенсуються кеш-пам'яттю процесора. У міру збільшення обсягу даних вплив латентності пам'яті зростає, внаслідок чого ефективність різних стратегій організації пошуку починає істотно залежати від характеру пошукового запиту.

Цю обставину необхідно враховувати під час вибору алгоритму для прикладних систем, у яких співвідношення успішних і неуспішних пошукових запитів може істотно відрізнятися.

Проведені експерименти показують, що кількісна величина виграшу визначається характеристиками конкретної апаратної платформи, зокрема розмірами кеш-пам'яті, латентністю різних рівнів пам'яті та особливостями реалізації апаратного попереднього відбору процесора.

Таким чином, результати дослідження свідчать про те, що вибір найбільш ефективного методу пошуку повинен здійснюватися з урахуванням не тільки алгоритмічних характеристик, але й особливостей архітектури пам'яті обчислювальної системи.

Для сучасних процесорів визначальними факторами продуктивності стають локальність розміщення даних, ефективність використання кеш-пам'яті, зменшення кількості помилкових передбачень розгалужень та можливість приховування латентності пам'яті за допомогою програмного попереднього завантаження даних.

5. Висновки

У роботі проведено експериментальне дослідження методів пошуку в AVL-дереві, представленому у масиві з використанням розміщення Ейтцингера, з урахуванням особливостей ієрархії пам'яті процесора. Отримані результати дають змогу зробити такі висновки.

1. Експериментально підтверджено, що при однаковій асимптотичній складності $O(\log n)$ продуктивність алгоритмів пошуку суттєво визначається особливостями організації пам'яті процесора. У міру збільшення обсягу масиву вплив латентності пам'яті стає визначальним фактором швидкодії, тоді як вплив обчислювальної складності алгоритму поступово зменшується.

2. Показано, що розміщення вузлів AVL-дерева в масиві за схемою Ейтцингера забезпечує більш ефективне використання просторової локальності даних порівняно з бінарним пошуком і представленням дерева на покажчиках, що призводить до стійкого скорочення часу пошуку

практично в усьому дослідженому діапазоні розмірів масивів.

3. Встановлено, що використання алгоритму пошуку без розгалужень EytzingerB практично усуває помилки прогнозування розгалужень, що підтверджується вимірами апаратного лічильника BranchMispredictions/Op. Однак експеримент показав, що зменшення кількості помилкових прогнозів саме по собі не гарантує максимальної швидкодії. При збільшенні обсягу даних і виходу робочого набору за межі кеш-пам'яті визначальним фактором стає латентність доступу до пам'яті.

4. Експериментально підтверджено, що застосування програмного попереднього завантаження даних дозволяє додатково зменшити час пошуку під час обробки великих масивів. Найбільш помітний ефект досягається при розміщенні робочого набору даних у кеш-пам'яті третього рівня та оперативній пам'яті, де програмне попереднє завантаження дозволяє частково приховати затримки звернення до пам'яті шляхом перекриття обчислень із завантаженням даних.

5. Аналіз апаратних лічильників CacheMisses/Op та BranchMispredictions/Op показав високий ступінь узгодженості з вимірними часовими характеристиками алгоритмів. Отримані результати підтверджують, що відмінності у швидкодії зумовлені насамперед особливостями взаємодії алгоритмів із підсистемою пам'яті процесора, а не лише відмінностями у кількості виконуваних операцій.

6. При розміщенні даних у кеш-пам'яті першого та другого рівнів найкращі результати демонструє алгоритм EytzingerB, який поєднує високу просторову локальність із практично повним усуненням помилок передбачення розгалужень. При збільшенні обсягу масиву та переході робочого набору даних в область кеш-пам'яті третього рівня, а потім оперативної пам'яті перевага переходить до алгоритму EytzingerBP, що забезпечує зменшення латентності доступу до пам'яті за рахунок програмного попереднього завантаження даних.

7. У порівнянні з бінарним пошуком розроблені методи пошуку в розміщенні Ейтцингера забезпечують скорочення часу до 3,05 разів (рис. 3, табл. 3). У порівнянні з традиційною реалізацією АВЛ-дерева на показниках виграш виявляється ще більш значним (до 5,67 разів) завдяки усуненню

великої кількості нерегулярних звернень до пам'яті та поліпшенню просторової локальності розміщення даних.

Таким чином, основним результатом дослідження є експериментальне підтвердження того, що ефективність пошуку в деревах визначається не лише алгоритмічною складністю $O(\log n)$, а й характером взаємодії алгоритму з ієрархією пам'яті процесора.

Отримані результати показують, що під час перенесення робочого набору даних між рівнями ієрархії пам'яті відбувається зміна домінуючих факторів продуктивності: якщо для невеликих масивів істотну роль відіграє вартість умовних переходів, то під час роботи з великими масивами основним обмежувальним фактором стає латентність доступу до пам'яті, яка ефективно компенсується програмним попереднім завантаженням даних. Отримані результати можуть бути використані при проектуванні високопродуктивних структур даних та алгоритмів пошуку для систем, що працюють із великими обсягами інформації. Перспективним напрямом подальших досліджень є детальний аналіз стратегій програмного попереднього завантаження даних для розміщення Ейтцингера, включаючи вибір оптимальної глибини попереднього завантаження, дослідження різних схем попереднього відбору та оцінку їхньої ефективності на процесорах різних архітектур.

Конфлікт інтересів

Автори декларують, що не мають конфлікту інтересів стосовно даного дослідження, в тому числі фінансового, особистісного характеру, авторства чи іншого характеру, що міг би вплинути на дослідження та його результати, представлені в даній статті.

Фінансування

Дослідження проводилося без фінансової підтримки.

Доступність даних

Рукопис не має пов'язаних даних.

Використання засобів штучного інтелекту

Автори підтверджують, що не використовували технології штучного інтелекту при створенні представленої роботи.

СПИСОК ЛІТЕРАТУРИ

1. Knuth, D. (1997), *The Art of Computer Programming. Sorting and Searching*, vol. 3, Addison-Wesley, 2nd edition, 780 p., available at: <https://www.scribd.com/document/438657664/The-Art-Of-Computer-Programming-Sorting-and-Searching-2nd-edition-Volume-3-pdf>
2. Knuth, D. (2010), *Selected papers on design of algorithms*, vol. 7, Stanford, California, 453 pp., available at: <https://press.uchicago.edu/ucp/books/book/distributed/S/bo8930293.html>
3. Sedgewick R., and Wayne K., (2024), *Algorithms*, 4nd edition, Princeton University, Addison-Wesley, 955 p. available at: <https://algs4.cs.princeton.edu/home/>
4. Cormen, H. Thomas, Leiserson, E. Charles, and Rivest, L. Ronald (2022), Clifford Stein. Introduction to algorithms, MIT Press, 1312 p., available at: <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/>
5. Wiener, R. (2022), "AVL Trees", *Generic Data Structures and Algorithms in Go: An Applied Approach Using Concurrency, Genericity and Heuristics*, Apress, Berkeley, CA, pp. 315–347, doi: https://doi.org/10.1007/978-1-4842-8191-8_10
6. Drepper, U. (2007), *What every programmer should know about memory*, 114 p., available at: <https://akkadia.org/drepper/cpumemory.pdf>

7. Pikus, F. G. (2021), *The Art of Writing Efficient Programs*, Packt Publishing, Birmingham, 445 p., available at: https://ivms.com.safespacenj.org/Tech_Books/writ%20eff%20pgms.pdf
8. (2024), *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, vol. 1, Order Number 248966-049, available at: <https://cdrdv2-public.intel.com/814198/248966-Optimization-Reference-Manual-V1-049.pdf>
9. (2026), *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined*, vol. 2, Instruction Set Reference Order Number 325383-092US, available at: <https://cdrdv2.intel.com/v1/dl/getContent/671110>
10. (2026), *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined*, vol. 3, System Programming Guide, 182 p., available at: <https://cdrdv2.intel.com/v1/dl/getContent/671447>
11. Fog, A. (2026), *Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms*, available at: https://www.agner.org/optimize/optimizing_cpp.pdf
12. Fog, A. (2025), *Optimizing subroutines in assembly language An optimization guide for x86 platforms*, available at: https://www.agner.org/optimize/optimizing_assembly.pdf
13. Fog, A. (2026), *The microarchitecture of Intel, AMD, and VIA CPUs*, available at: <https://www.agner.org/optimize/microarchitecture.pdf>
14. Fog, A. (2025), *Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*, available at: https://www.agner.org/optimize/instruction_tables.pdf
15. Khuong P.-V., and Morin, P. (2017), "Array Layouts for Comparison-Based Searching", *Journal of Experimental Algorithmics*, vol. 22, pp. 1–39, doi: <https://doi.org/10.1145/3053370>
16. Shostak, A. (2026), "Comparative experimental analysis of methods for implementing the AVL tree", *Control, Navigation and Communication Systems*, no. 1, pp. 131–134, doi: <https://doi.org/10.26906/SUNZ.2026.1.131>
17. Anderson, S.E. (2026), *Bit Twiddling Hacks*, available at: <https://graphics.stanford.edu/~seander/bithacks.html>
18. Warren H. S. (2013), *Hacker's Delight*, Addison-Wesley, 2nd ed., 512 p., available at: <https://dl.acm.org/doi/10.5555/2462741>
19. (2026), *Branchless Programming*, available at: <https://en.algorithmica.org/hpc/pipelining/branchless/>
20. Mowry, T.C., Lam, M. S., and Gupta A. (1992), "Design and Evaluation of a Compiler Algorithm for Prefetching", *Proceedings of ASPLOS-V*, pp. 62–73, available at: <https://suif.stanford.edu/papers/mowry92.pdf>
21. Luk C.-K., and Mowry, T.C. (1996), "Compiler-Based Prefetching for Recursive Data Structures", *Proceedings of ASPLOS-VII*, pp. 222–233, available at: <https://www.cs.tufts.edu/comp/150IPL/papers/luk96prefetch.pdf>

Received (Надійшла) 22.05.2026

Accepted for publication (Прийнята до друку) 14.08.2026

Publication date (Дата публікації) 25.09.2026

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Шостак Анатолій Васильович – кандидат технічних наук, доцент, доцент кафедри кібербезпеки та інтелектуальних інформаційних технологій Національного аерокосмічного університету України «ХАІ», Харків, Україна;
Anatolii Shostak – Candidate of Technical Sciences, Associate Professor, Associate Professor of a Department of Cybersecurity and Intelligent Information Technologies of the National Aerospace University of Ukraine "KhAI", Kharkiv, Ukraine;
 e-mail: a.shostak@csn.khai.edu; ORCID Author ID: <https://orcid.org/0009-0003-7687-3641>

Experimental Study of Search Methods in an AVL Tree Considering the Memory Hierarchy

Anatolii Shostak

Abstract. Background. Investigating the effects of data spatial locality, branch prediction, and software prefetching on search performance is of considerable interest in the development of high-performance data structures. **Subject of the study:** search methods in an AVL tree represented as an array using the Eytzinger layout, and the impact of the processor memory hierarchy on their performance. **Purpose of the study:** to experimentally evaluate the efficiency of various search methods in an array-based AVL tree depending on the array size and the location of the working data set relative to the levels of the processor memory hierarchy. **Research objectives.** To implement and experimentally investigate the Binary, AVL, Eytzinger, EytzingerB, and EytzingerBP search algorithms; to determine the ranges of array sizes corresponding to different levels of the memory hierarchy; to compare the performance of the investigated algorithms; and to determine the effects of data spatial locality, branch mispredictions, and software prefetching on search efficiency. **Research methods.** The study employs algorithmic analysis, experimental software performance evaluation, and statistical processing of measurement results. The experimental evaluation was performed using BenchmarkDotNet on the .NET 8 platform with processor hardware performance counters. **Research results.** It was established that the efficiency of search algorithms is determined not only by their computational complexity but also by the characteristics of their interaction with the processor memory hierarchy. The results show that when the working data are located in the L1/L2 caches, the EytzingerB algorithm provides the shortest search time, whereas when the data are located in the L3 cache and main memory, the EytzingerBP algorithm, which uses software prefetching, demonstrates the best performance. Experimental relationships between search time and array size were obtained, the ranges of effective application of the investigated algorithms were determined, and the performance gains achieved by the most efficient search methods were quantitatively evaluated. The experiments demonstrated a change in the most efficient search algorithm as the working data move between levels of the processor memory hierarchy: within the L1/L2 range, the EytzingerB algorithm provides the shortest search time, whereas within the L3 and main-memory range, the EytzingerBP algorithm is the most efficient. **Conclusions.** Representing an AVL tree using the Eytzinger array layout provides a substantial increase in search performance compared with the conventional pointer-based tree implementation. Experimental results confirm that the choice of the optimal search method is determined by the level of the memory hierarchy in which the working data set resides. The obtained results can be used in the design of high-performance search data structures and in the optimization of search algorithms for modern multi-level memory subsystems.

Keywords: AVL tree; Eytzinger layout; search algorithms; software prefetching; cache memory; data locality; BenchmarkDotNet; experimental performance evaluation.