Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

МЕТОДИЧНІ ВКАЗІВКИ
ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ
з дисципліни

"ТЕОРІЯ АЛГОРИТМІВ"

для студентів
спеціальності 126 - Інформаційні системи та технології

2023

Методичні вказівки до виконання комп’ютерних практикумів з дисципліни “Теорія алгоритмів” для студентів спеціальності 126 - Інформаційні системи та технології /Солдатова М. О.., – Київ: КПІ, 2023. – 64 с.

Укладач: Солдатова Марія Олександрівна,
кандидат технічних наук.

Рецензент: Гавриленко О. В.. доц. кафедри ІСТ ФІОТ, к.т.н

Затверджено вченою радою ФІОТ	Затверджено на засіданні кафедри ІСТ
Протокол № від	Протокол № від

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Дані методичні вказівки до виконання комп'ютерних практикумів з дисципліни «Теорія алгоритмів» дозволяють закріпити теоретичні відомості отримані під час вивчення даної дисципліни та отримати практичні навички проектування та дослідження алгоритмів студентам напряму підготовки спеціальності 126 - Інформаційні системи та технології.

Комп'ютерні практикуми виконують за допомогою мов високого рівня та застосуванням додаткового програмного забезпечення для оформлення звітів, зокрема MS Word, програми для креслень. Платформа та мова програмування обираються студентом самостійно, але обов'язково вказуються у звітах..

Мета цих комп'ютерних практикумів полягає у навчанні студентів дослідженню поставлених задач, застосуванню відомих алгоритмів або самостійному проектуванню алгоритмів для отримання рішення таких задач. Окремо приділяється увага дослідженню структури, характеристик та особливостей використання алгоритмів, що застосовуються під час виконання практикумів.

Тематика даних практикумів охоплює всі теми та розділи, які висвітлюють на теоретичних заняттях. Зокрема це стосується визначення основних характеристик алгоритмів, абстрактних типів даних та роботи з ними, методів хешування, алгоритмів сортування.

На протязі семестру студенти виконують індивідуальне завдання, яке підсумовує практичний матеріал, засвоєний на практикумах.

ОФОРМЛЕННЯ ЗВІТУ ТА ПОРЯДОК ЙОГО ПОДАННЯ.

Відповідно до графіку студенти перед виконанням комп'ютерних практикумів повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку по кожному практикуму студент здає викладачу цілком оформлений звіт згідно визначеним вимогам.

Звіт має містити:

- титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид і тему роботи, виконавця та особу, що приймає звіт, рік);
- мету, варіант і завдання практикуму;
- лаконічний опис теоретичних відомостей;
- текст програми, що обов'язково містить коментарі;
- вхідні та вихідні дані програми;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 х 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок. Для набору тексту звіту використовують редактор MS Word : шрифт Times New Roman, 12 пунктів. Міжрядковий інтервал: полуторний – для тексту звіту, одинарний – для лістингів програм, таблиць і роздруківок даних.

Під час захисту виконаного практикуму студент повинен виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу практикуму, по змісту основних розділів розробленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і здачі практикуму студент повинен відповісти на контрольні питання, що наведені наприкінці опису відповідної роботи. Загальний залік студент одержує після виконання і здачі останнього практикуму.

Комп’ютерний практикум 1

Тема: Побудова і аналіз алгоритмів.

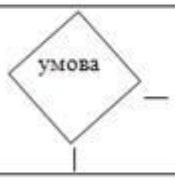
Мета роботи: ознайомитись з роботами, що виконує програміст на кожному з етапів розв’язку задачі.

Теоретичні дані

Будь-який алгоритм треба розглядати як опис процедури, яка обробляє вхідні дані й отримує вихідні дані. Вхідні дані ще називають входом алгоритму, або його аргументами, а вихідні дані - виходом алгоритму, або результатом.

Алгоритм вважається правильним, якщо при будь-якому допустимому для даної задачі наборі вхідних даних він завершує свою роботу і отримує результат, що задовольняє вимоги задачі. У цьому випадку кажуть, що алгоритм розв'язує дану задачу. Неправильний алгоритм може або не зупинитися, або дати неправильний результат.

При створенні блок-схеми алгоритму застосовуються наступні позначення:

Блоки	Назва та призначення
	Початок або кінець алгоритму
	Блок введення даних
	Блок виведення даних на друк
	Арифметичний блок-використовується при обчисленні виразів; процес, присвоєння.
	Логічний блок – використовується для перевірки умови
	Блок модифікації – використовується для зміни в залежності від попередніх значень
	Блок звернення до підпрограм

Правила формування оцінки O( ):

1. При оцінці за функцію береться кількість операцій, що зростає найшвидше.
   Тобто, якщо в програмі одна функція, наприклад, множення, виконується O(n) разів, а додавання - O(n^2) разів, то загальна складність програми - O(n^2), оскільки при збільшенні n додавання стануть виконуватися настільки часто, що будуть впливати на швидкодію куди більше, ніж множення.
2. При оцінці O( ) константи не враховуються.

Завдання

1. Виконати всі етапи розробки програми.
   1) Постановка задачі – чітко вказати, що дано і що треба знайти.
   2) Побудова моделі – які структури даних та які математичні залежності використані.

3) Розробка алгоритму – опис алгоритму у вигляді блок-схеми.
4) Правильність алгоритму – довести покроково правильність розробленого алгоритму.
5) Аналіз алгоритму та його складності – оцінити використовуючи О-символіку час виконання алгоритму в найгіршому або/і в середньому.
6) Реалізація алгоритму – навести текст програми.
7) Перевірка програми – описати тестові дані для перевірки програми на правильність, ефективність реалізації та обчислювальну складність (для цього використайте блок-схему алгоритму та профілі виконання програми).
8) Скласти таблицю тестування.
9) Навести скрин-шоти роботи програми.

2. Відповіді на контрольні питання

Варіанти завдань

1. Пошук мінімального елемента (індекс і значення) в одномірному масиві з n елементів.
2. Пошук мінімального елемента (індекс і значення) в двомірному масиві з $n \times m$ елементів.
3. В одномірному масиві з n елементів поміняти місцями максимальний та мінімальний елементи.
4. В одномірному масиві з n елементів обчислити кількість елементів більше заданого числа.
5. В двомірному масиві з $n \times m$ елементів знайти кількість нулів.
6. В двомірному масиві з $n \times n$ елементів знайти суму діагональних елементів.

Масиви заповнити випадковими числами автоматично. В програмі передбачити наступні варіанти отримати вхідні дані для обчислень: ввести вручну, згенерувати автоматично (і записати ці дані у файл); прочитати з файлу.

Склад звіту практичної роботи

- постановка задачі;
- блок-схема, на якій виконано аналіз складності алгоритмів;
- обґрунтування правильності алгоритму;
- текст програми;
- опис тестових даних (якого характеру дані і для якої перевірки використані);
- результати дослідження у вигляді графіків або діаграм;
- таблиця тестування;
- скрин-шоти роботи програми;
- відповіді на контрольні питання;
- висновки.

Контрольні питання

1. З яких етапів складається процес створення комп'ютерної програми для вирішення довільної практичної задачі?
2. Що саме має з'ясувати розробник програми на етапі постановки задачі?
3. Що робить розробник програми на етапі побудови моделі? Які фактори впливають на вибір структури моделі?
4. Якими міркуваннями має керуватися розробник програми на етапі розробки алгоритму? Чи потрібно перевіряти або доводити правильність алгоритму, якщо так, то з якою метою?
5. Навіщо виконується аналіз алгоритму та його складності?
6. Існує три аспекти перевірки програми: на правильність, на ефективність реалізації, на обчислювальну складність. Розкрийте суть кожної з перевірок.
7. Навіщо виконується вимірювання часу виконання програми? Які чинники на нього впливають?

Комп’ютерний практикум 2

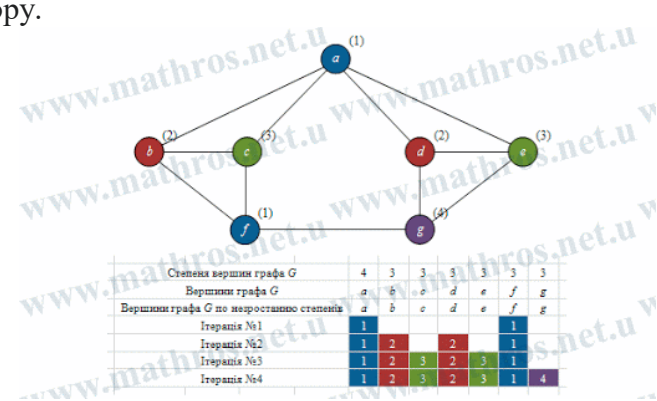
Тема: Методи розробки алгоритмів.

Мета роботи: порівняння алгоритмів розв’язку задачі, побудованих різними методами.

Теоретичні дані

Розфарбовування простого графа G

Розфарбовування простого графа G —це таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору.



Степени вершин графа G	4	3	3	3	3	3	3
Вершини графа G	a	b	c	d	e	f	g
Вершини графа G по незростанню степенів	a	b	c	d	e	f	g
Ітерація №1	1					1	
Ітерація №2	1	2		2		1	
Ітерація №3	1	2	3	2	3	1	
Ітерація №4	1	2	3	2	3	1	4

Алгоритмом послідовного розфарбування:

на першому кроці, вершини графа G розташовуються в порядку незростання їх степенів. Перша вершина фарбується в колір *«1»*. Після цього, список вершин проглядається зліва на право і в колір *«1»* фарбується всяка вершина, яка не є суміжною з іншою, вже пофарбованою в цей колір вершиною.

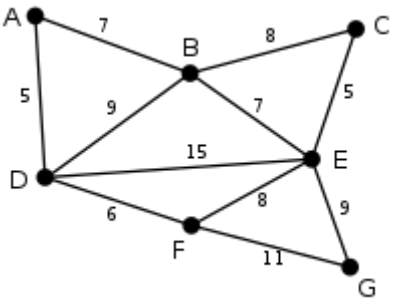
На наступному кроці, повертаємося до першої не пофарбованої у списку вершини, фарбуємо її в колір *«2»* і знову переглядаємо список вершин зліва на право, фарбуючи в колір *«2»* будь-яку не пофарбовану вершину, яка не поєднана ребром з іншою, вже пофарбованої в колір *«2»* вершиною. Аналогічно діємо з кольорами *«3»*, *«4»* і так далі, і робимо це до тих пір поки не будуть пофарбовані всі вершини графа.

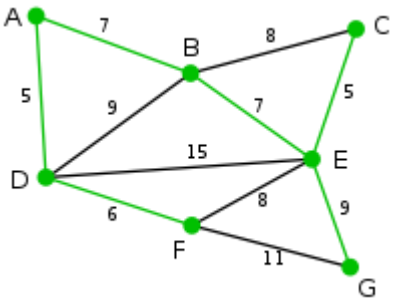
Алгоритм Уелша-Пауелла

1. Вершини графу впорядковуються за спаданням степенів.
2. i=1.
3. Вершина, що перша у списку, фарбується у колір c_i.
4. У той же колір c_i фарбуються в порядку за списком усі вершини, несуміжні з вершинами, вже пофарбованими на цьому кроці.
5. Пофарбовані вершини викреслюють із списку.
6. i=i+1.
7. Повторюємо пункти 2-4, поки у списку є непофарбовані вершини.

Побудова мінімального кістякового дерева.

Мінімальне кістякове дерево у зв'язаному, зваженому, неорієнтованому графі — це кістяк цього графа, що має мінімальну можливу вагу, де під вагою дерева розуміється сума ваг його ребер.





Алгоритм Прима

1. Спочатку ребра сортують за зростанням ваги.
2. Додають найменше ребро в дерево.
3. Зі списку ребер із найменшою вагою вибирають таке нове ребро, щоб одна з його вершин належала дереву, а інша — ні.
4. Це ребро додають у дерево і знову переходять до кроку 3.
5. Робота закінчується, коли всі вершини будуть у дереві.

Алгоритм Крускала

1. Побудова виродженого лісу, що містить V дерев, кожне з яких складається з однієї вершини.
2. Об'єднання двох дерев, для чого використовуються найкоротші можливі ребра, поки не утвориться єдине дерево.
3. Це дерево і буде мінімальним кістяковим деревом.

Алгоритм Борувки

Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до кістякового лісу графа, доки ліс не перетвориться на дерево, (тобто, ліс, що складається з однієї компоненти зв'язності).

Пошук найкоротшого шляху

Алгоритм Террі

У зв’язаному графі завжди можна знайти такий маршрут, що зв’язує дві задані вершини *v* та *u*, якщо, виходячи з вершини *v* і здійснюючи послідовний перехід від кожної досягнутої вершини до суміжної з нею, керуватися такими правилами:

1) йдучи по довільному ребру, кожний раз відмічати напрямок, в якому воно було пройдене;

2) виходячи з деякої вершини v_i, завжди рухатися тільки по тому ребру, яке не було пройдене або було пройдене у зворотному напрямку;

3) для кожної вершини v_i, відмінної від *v*, відмічати те ребро, яке першим заходить у v_i, якщо вершина v_i, зустрічається вперше;

4) виходячи з деякої вершини v_i, відмінної від *v*, по першому ребру, яке заходить у v_i, рухатися лише тоді, коли немає інших можливостей.

Алгоритм хвильовий

1. Позначаємо вершину *v* індексом 0, а вершини, що належать образу вершини *v*, – індексом 1. Множину вершин з індексом *k* позначаємо $F_k(v)$. Вважаємо $k = 1$.
2. Якщо $F_k(v) = \emptyset$ або виконується $k = n - 1$ і u не належить $F_k(v)$, то вершина u є незв’язаною з v_i робота алгоритму на цьому завершується. В іншому випадку перейти до пункту 3.
3. Якщо u не належить $F_k(v)$, то переходимо до пункту 4. В іншому випадку існує шлях із *v* до *u* завдовжки *k*, причому цей шлях є мінімальним. Послідовність вершин $v, u_1, u_2, \ldots, u_{k-1}, u$, де

$$u_{k-1} \in F_{k-1}(v) \cap D^{-1}(u),$$
$$u_{k-2} \in F_{k-2}(v) \cap D^{-1}(u_1),$$
$$\ldots\ldots\ldots\ldots$$
$$u_1 \in F_1(v) \cap D^{-1}(u_2)$$

і є шуканим мінімальним шляхом з *v* у *w*. На цьому робота алгоритму завершується.

4. Позначаємо індексом $k + 1$ всі непозначені вершини, які належать образу множини вершин з індексом *k*. Множину вершин з індексом $k + 1$ позначаємо $F_{k+1}(v)$. Збільшуємо індекс *k* на 1 і переходимо до пункту 2.

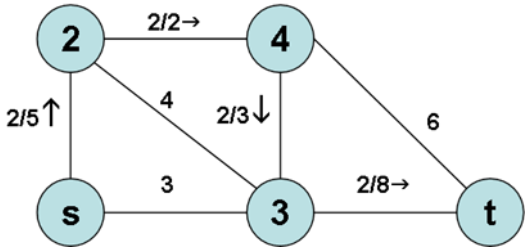
Алгоритм Дейкстри

1. Присвоювання початкових значень. Виконати $l(a) = 0$ та вважати цю мітку постійною. Виконати $l(v) = \infty$ для всіх $v \neq a$ й вважати ці мітки тимчасовими.
Виконати $x = a$, $M = \{a\}$.

2. Оновлення міток. Для кожної вершини v належить $\Gamma(x) \setminus M$ замінити мітки: $l(v) = \min\{ l(v), l(x) + w(x, v) \}$, тобто оновлювати тимчасові мітки вершин, у які з вершини x іде дуга.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v^* з умови $l(v^*) = \min\{l(v)\}$,
З v належить T, де $T = V \setminus M$.
4. Вважати мітку вершини v^* постійною й виконати $M = M \vee\{v^*\}$; $x = v^*$ (вершину v^* включено в множину M).
5. а) Для пошуку шляху від a до z: якщо $x = z$, то $l(z)$ – довжина найкоротшого шляху від a до z, зупинитись; якщо $a \neq z$, то перейти до кроку 2.
б) Для пошуку шляхів від a до всіх вершин: якщо всі вершини отримали постійні мітки (включені в множину M), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

Пошук найбільшої течії в мережі між заданими вершинами

Течією в мережі G називається дійсна функція f:V×V→R, яка має такі властивості:
а) обмеження пропускною здатністю: для будь-яких вершин i та j виконується нерівність $f[i, j] \le c(i, j)$;
б) асиметричність для будь-яких вершин i та j виконується рівність f[I,j]=-f[i,j];
в) збереження течії: для будь-якої проміжної вершини Vвиконується рівність $\sum_{u \in V} f[v,u] = 0$, тобто сума всіх течій що входять до вершини V дорівнює сумі всіх течій, що виходять з неї.



Алгоритм Форда – Фалкерсона

1. Спочатку всі течії дорівнюють 0. Залишкова мережа співпадає з початковою мережею.
2. Знаходимо аргументальний ланцюг P у залишковій за течією f мережі.
3. За ланцюгом P збільшуємо течію f на максимально можливу величину, перераховуємо залишкову мережу та переходимо до кроку 1.
4. Алгоритм закінчує свою роботу у випадку, коли крок 1 виконати неможливо, тобто аргументального ланцюга у залишковій мережі більше не існує. Одержана таким чином течія буде максимальною.

Алгоритм Едмонса – Карпа

1. Спочатку всі течії дорівнюють 0. Залишкова мережа співпадає з початковою мережею.
2. Знаходимо найкоротший аргументальний ланцюг P у залишковій за течією f мережі.
3. За ланцюгом P збільшуємо течію f на максимально можливу величину, перераховуємо залишкову мережу та переходимо до кроку 1.
4. Алгоритм закінчує свою роботу у випадку, коли крок 1 виконати неможливо, тобто аргументального ланцюга у залишковій мережі більше не існує. Одержана таким чином течія буде максимальною.

Граф називається **сильно зв'язним**, якщо існує шлях з будь-якої вершини графу до кожної з інших вершин. Зокрема, це означає наявність шляхів в обох напрямках: з a в b і з b в a.

В орієнтованих графах розрізняють декілька понять зв'язності.

1. Орієнтований граф називається **сильно-зв'язним**, якщо в ньому існує (орієнтований) шлях з будь-якої вершини до будь-якої іншої, або, що тотожно, граф містить рівно одну компоненту сильної зв'язності.

2. Орієнтований граф називається **односторонньо-зв'язним**, якщо для будь-яких двох його вершин u і v існує хоча б один з маршрутів від v до u чи від u до v.
3. Орієнтований граф називається **слабко-зв'язним**, якщо є зв'язним неорієнтований граф, отриманий з нього заміною орієнтованих ребер на неорієнтовані.

Компонента сильної зв'язності орієнтованого графу G — це найбільший сильно зв'язаний підграф. Якщо кожну компонента сильної зв'язності стягнути до однієї вершини, отримаємо орієнтований ациклічний граф, ущільнення G. Орієнтований граф є ациклічним тоді і лише тоді, коли він не має компонент сильної зв'язності з більш як однією вершиною, бо орієнтований цикл є сильно зв'язним і кожна нетривіальна компонента сильної зв'язності містить щонайменше один орієнтований цикл.

Алгоритми, що знаходять компоненти сильної зв’язності:

1. алгоритм Косараджу,
2. алгоритм Тар’яана,
3. алгоритм компонент сильної зв'язності по шляхах.

Алгоритм Косараджу

Алгоритм Косарадж— алгоритм для знаходження компонент сильної зв’язності орієнтованого графу. Він використовує факт, що транспонований граф (той самий граф з оберненими напрямками ребер) має ті самі компоненти сильної зв'язності, що й початковий граф.

Алгоритм Косараджу працює так:

1. Нехай G орієнтований граф і S порожній стек.
2. Допоки S не містить всі вершини:
3. Вибрати довільну вершину v не з S. Виконати пошук в глибину від v. По завершені пошуку в глибину для кожної вершини u, заштовхуємо u на S.
4. Обернути всі ребра для отримання транспонованого графу.
5. Доки S непорожній (містить вершину в порядку завершення, на верхівці стеку — останнє завершення пошуку в глибину):
6. Виштовхнути вершину v з S. Виконати пошук в глибину від v. Набір відвіданих вершин дасть компоненту сильної зв'язності, що містить v; записати це і видалити всі ці вершини з графу G і стеку S. Так само можна використати пошук в ширину.

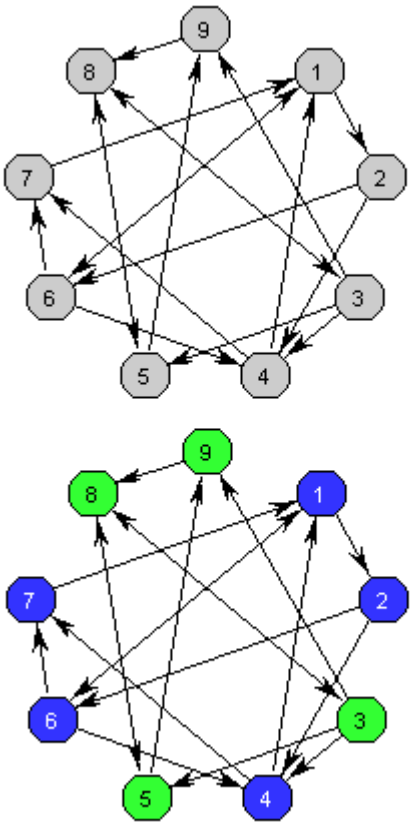
Алгоритм Тар'яна

Алгоритм Тар'я - алгоритм пошуку компонент сильної зв'язності в орієнтованому графі, що працює за лінійний час.

Цей алгоритм заснований на тому, що:

1. Вершини розглядаються в зворотному топологічному порядку, тому в кінці рекурсивної функції для вихідної вершини не буде виявлено жодної вершини з тією же сильною компонентою, так як всі вершини, досяжні з вихідної, вже оброблені.
2. Зворотні зв'язки в дереві дають другий шлях з однієї вершини в іншу і пов'язують сильні компоненти.

Неформальний опис алгоритму:



	id	pre	low
1	1	0	10
2	1	1	10
3	2	5	10
4	1	2	10
5	2	6	10
6	1	4	10
7	1	3	10
8	2	7	10
9	2	8	10

Алгоритм Тар'яна можна розуміти як варіацію алгоритму пошуку в глибину, в якому при відвідуванні вершини і закінченні обробки вершини виконуються додаткові дії. Відвідування вершини відбувається при русі від кореня до листя, а закінчення обробки вершини - на зворотному шляху. При відвідуванні вершини вона проштовхується у допоміжний стек, а виштовхується при закінченні обробки .

Індекси компонент зв'язності всіх вершин зберігаються у векторі id, індексованих номерами вершин. Вектор low відстежує вершину з найменшим номером в прямому порядку обходу, досяжну з кожного вузла через послідовність прямих зв'язків, за якими слідує один висхідний зв'язок. Скориставшись пошуком в глибину таким чином, щоб розглядати вершини в зворотному топологічному порядку, обчислюється для кожної вершини v максимальну точку, досяжну через зворотний зв'язок з попередника (low [v]). Коли для вершини v виконується pre [v] = low [v], її виштовхують зі стека, а також всі вершини вище її і всім їм присвоюється номер наступного компонента.

Завдання

1. Для свого варіанту зробити наступні дії:
   1) Сформулювати постановку задачі
   2) Обрати відповідні 2 алгоритми з теоретичної частини практикуму або на свій розсуд
   3) Накреслити блок-схеми, на яких виконано аналіз складності алгоритмів
   4) Написати програмний код
   5) Провести дослідження продуктивності роботи алгоритмів, зробити результати дослідження у вигляді графіків або діаграм;
   6) Зробити висновки про доцільність використання кожного з алгоритмів для типових вхідних даних та про відповідність результатів експериментального дослідження аналітичним оцінкам складності.
   7) Скласти таблицю тестування.
   8) Навести скрин-шоти роботи програми. для заданої задачі;ю
   9) Дати відповіді на контрольні питання

Варіанти завдань

В Києві є декілька цікавих місць, а саме:

1. Червоний університет
2. Андріївська церква
3. Михайлівський собор
4. Золоті ворота
5. Лядські ворота
6. Фунікулер
7. Київська політехніка
8. Фонтан на Хрещатику
9. Софія київська
10. Національна філармонія
11. Музей однієї вулиці

Необхідно створити графічне зображення сполучень між ними, враховуючи наступні умови:

- Сполучення є, якщо місця поєднані однією вулицею, проспектом, провулком або унікальним транспортом.
- Визначити приблизно відстань в км.
- Врахувати односторонній рух між наступними місцями: Андріївська церква- Михайлівський собор, Музей однієї вулиці- Фунікулер, Михайлівський собор- Софія київська, Софія київська- Лядські ворота, Лядські ворота- Михайлівський собор, Золоті ворота- Фонтан на Хрещатику.

Враховуючи надану інформацію розв'язати наступні задачі:

1. Перша частина

   Варіанти завдань. Варіант обирається згідно номеру бригади або залишку від ділення номера бригади на 4.

1) Студент, що проживає в гуртожитку НТУУ «КПІ ім. Ігоря Сікорського» вирішив відвідати всі вищеперераховані місця. Він полюбляє ходити пішки, але обмежений в часі, тому за раз буде відвідувати лише одне місце. Допоможіть йому скласти список найкоротших маршрутів його екскурсій.
2) Група допитливих студентів НТУУ «КПІ ім. Ігоря Сікорського» вирішила розрахувати максимальну кількість автомобілів, яка може досягнути Музею однієї вулиці, якщо від їх університету буде прямувати максимальна кількість авто. При своїх розрахунках вони вважали, що за годину по одній смузі шляху може проїхати до 400 автомобілів. Допоможіть їм розрахувати цю кількість.
3) Не всі студенти НТУУ «КПІ ім. Ігоря Сікорського» полюбляють гуляти пішки, тому компанія хлопців і дівчат з університету вирішила прокласти транспортні маршрути від свого навчального закладу до інших вищеперерахованих місць за умови найменшої вартості такого проєкту. Допоможіть їм прокласти такі маршрути.
4) Прогулюючись пішки до всіх вищеперерахованих місць студент зацікавився, а скільки необхідно розробити типів сувенірних крамничок, щоб розташувати їх між цими місцями. Крамнички однакового типу не повинні бути розташовані на сполученнях, що прилягають до одного і того ж місця. Допоможіть йому визначити кількість типів крамничок.

2. Друга частина

Визначити зв'язність та компоненти зв'язності для графічного подання умов задачі.

Для розв'язку першої та другої частини завдання можна використовувати алгоритми з теоретичної частини.

Склад звіту

- постановка задачі ;
- обґрунтування вибору алгоритмів;
- блок-схеми, на яких виконано аналіз складності алгоритмів;
- результати дослідження у вигляді графіків або діаграм;

- висновки про доцільність використання кожного з алгоритмів для типових вхідних даних та про відповідність результатів експериментального дослідження аналітичним оцінкам складності;
- таблиця тестування;
- скрин-шоти роботи програми для заданої задачі;
- відповіді на контрольні питання;

Контрольні питання

1. Перерахуйте відомі вам методи розробки алгоритмів. Докладніше розкажіть про один з них.
2. Перерахуйте переваги та недоліки наступних методів розробки алгоритмів: методу часткових (проміжних) цілей, методу підйому (локального пошуку), методу відпрацювання назад.
3. Який тип алгоритмів називають «жадібними» і чому? В чому полягає різниця між жадібним та скупим алгоритмами?
4. Дайте характеристику евристичним алгоритмам. В яких випадках доцільно використовувати цей тип алгоритмів? Опишіть загальний підхід до побудови евристичних алгоритмів.
5. Проаналізуйте, що спільного мають та чим відрізняються алгоритми, що використовують пошук з поверненням, та алгоритми, що використовують метод гілок та границь.
6. Поясніть, для чого можна використовувати метод альфа-бета відсікання.
7. Поясніть термін «структурне програмування». Для чого воно застосовується?
8. В чому полягають особливості динамічного програмування при його застосуванні в програмуванні?
9. Що таке метод сходження? Які переваги та недоліки він має?

Комп'ютерний практикум 3

Тема: Абстрактні типи даних

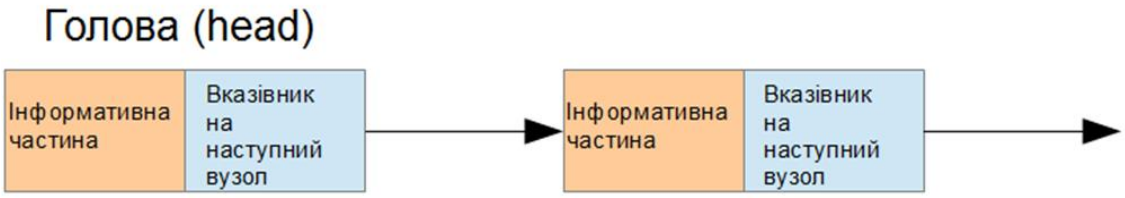
Мета роботи: порівняння реалізацій абстрактних типів даних, побудованих на базі різних структур даних.

Теоретичні дані

Абстрактний тип даних (АТД) — це математична модель для типів даних, де тип даних визначається поведінкою (семантикою) з точки зору користувача даних, а саме в термінах можливих значень, можливих операцій над даними цього типу і поведінки цих операцій. Вся внутрішня структура такого типу захована від розробника програмного забезпечення — в цьому і полягає суть абстракції. Абстрактний тип даних визначає набір функцій, незалежних від конкретної реалізації типу, для оперування його значеннями. Конкретні реалізації АТД називаються структурами даних.

Однозв'язний список

Однозв'язний список складається з вузлів, кожен з яких містить у собі данні (інформаційну частину) та посилання на наступний вузол.



Кожен вузол містить 1 поле покажчика на наступний вузол. Поле покажчика останнього вузла містить нульове значення (вказує на NULL).

В такому списку дозволяється:

1. додавати елементи
    1) з голови (початку),
    2) в кінець (хвіст) списку,
    3) між будь-якими двома іншими;
2. видаляти будь-який елемент.

Стек

Стек - структура даних, яка працює за принципом (дисципліною) «останнім прийшов — першим пішов» (LIFO, англ. last in, first out). Всі операції (наприклад, видалення елементу) в стеку можна проводити тільки з одним елементом, який знаходиться на верхівці стека та був введений в стек останнім.

Стек можна розглядати як певну аналогію до стопки тарілок, з якої можна взяти верхню, і на яку можна покласти верхню тарілку (інша назва стека — «магазин», за аналогією з принципом роботи магазину в автоматичній зброї).

Зі стеком можна виконувати наступні дії:

1. **push** («заштовхнути елемент»): елемент додається в стек та розміщується в його верхівці. Розмір стека збільшується на одиницю. При перевищенні розміру стека граничної величини, відбувається переповнення стека (англ. stack overflow).
2. **pop** («виштовхнути елемент»): отримує елемент з верхівки стека. При цьому він видаляється зі стека і його місце в верхівці стека займає наступний за ним відповідно до правила LIFO, а розмір стека зменшується на одиницю. При намаганні «виштовхнути» елемент з вже пустого стека, відбувається ситуація «незаповненість» стека (англ. stack underflow).

Кожна з цих операцій зі стеком виконується за фіксований час O(1) і не залежить від розміру стеку.

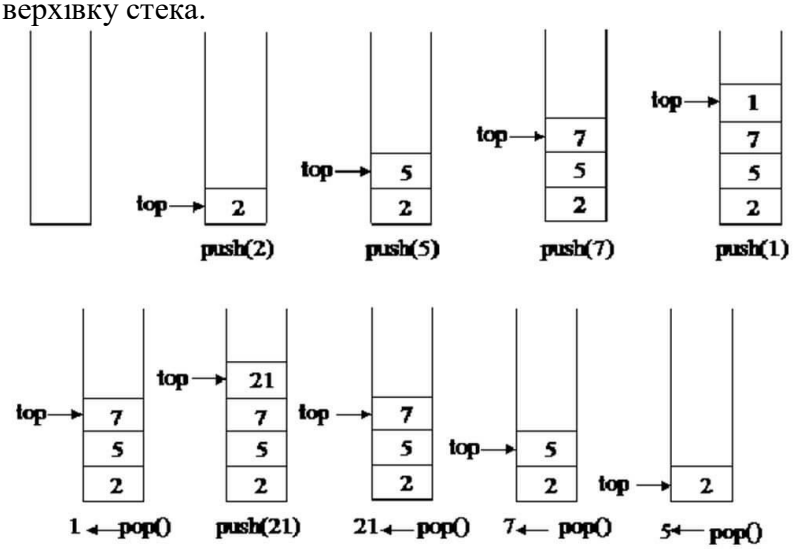
Додаткові операції (присутні не у всіх реалізаціях стека):

1. **isEmpty**: перевірка наявності елементів в стека; результат: істина (true), коли стек порожній.

2. **isFull**: перевірка заповненості стека. Результат: істина, коли додавання нового елементу неможливе.
3. **clear**: звільнити стек (видалити усі елементи).
4. **top**: отримати верхній елемент (без виштовхування).
5. **size**: отримати розмір (кількість елементів) стека.
6. **swap**: поміняти два верхніх елементи місцями.

Організація в пам'яті комп'ютера

1. масив
2. множина комірок в певній області комп'ютера з додатковим зберіганням ще й вказівника на верхівку стека.



Черга

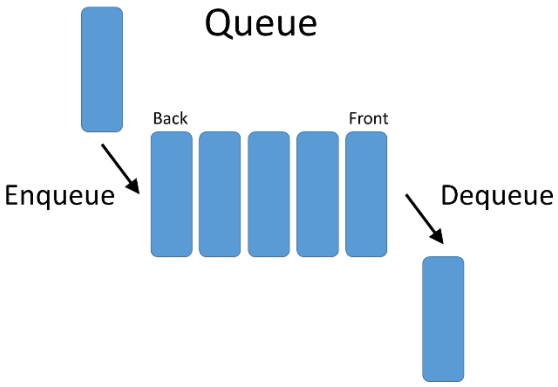
Черга (англ. queue)— динамічна структура даних, що працює за принципом «перший прийшов — перший пішов» (англ. FIFO — first in, first out).

Черга - однозв'язний циклічний список, тобто кожен вузол містить одне поле покажчика на наступний вузол, а поле покажчика останнього вузла містить адресу першого вузла (кореня списку).

У черги є голова (англ. head) та хвіст (англ. tail). Елемент, що додається до черги, опиняється в її хвості. Елемент, що видаляється з черги, знаходиться в її голові.

Основні операції з чергою:

1. ***enqueue*** — "поставити в чергу". Операція додавання елемента в "хвіст" черги. При цьому довжина черги збільшується на одиницю. Якщо відбувається намагання додати елемент у вже заповнену чергу, відбувається її переповнення (англ. *queue overflow*).
2. ***dequeue*** — "отримання з черги". Операція, яка повертає елемент з голови та видаляє його з черги, таким чином встановлюючи голову на наступний за видаленим елемент та зменшуючи довжину на одиницю. При намаганні видалити елемент з пустої черги, виникає ситуація "незаповненість" (англ. *queue underflow*).



Двобічно зв'язаний список

Двобічно зв'язаний список — вид зв'язаного списку, у якому посилання в кожному вузлі вказують на попередній і на подальший вузол у списку.

Вузол двобічно зв'язаного списку складається з трьох полів: вказівника на попередній елемент, поля даних та вказівника на наступний елемент.

Якщо в списку після останнього елемента йде перший, то такий список називається кільцевим двобічно зв'язаним списком. Тобто, поле голови списку вказує на хвіст списку, а поле хвоста списку вказує на голову списку.

По двобічно зв'язаному списку можна пересуватися в будь-якому напрямку — як від початку до кінця, так і навпаки. Для нього простіше проводити видалення і перестановку елементів, оскільки завжди відомі адреси тих елементів списку, вказівник яких спрямований на змінюваний елемент.

Стандартні функції для двобічно зв'язного списку

1. Функція **push** додати новий елемент у і-ту позицію списку. Вона виконує наступні дії:
   1) створює новий елемент;
   2) копіює значення інформаційного поля;
   3) якщо даний елемент є єдиним:
      - покажчики попереднього та наступного елемента посилаються на цей елемент
      - покажчик першого вказує на єдиний елемент в списку.
   4) інакше зсувається покажчик на і-ий елемент і додається новий елемент перед і-им.
2. Функція **pop** виштовхує і-ий елемент зі списку. Вона виконує наступні дії:
   1) якщо список порожній, вихід з функції;
   2) якщо в список містить єдиний елемент;
      - копіюється значення інформаційного поля;
      - видаляється елемент зі списку;
      - присвоюється заголовку пусто.
   3) інакше - зсувається покажчик на і-ий елемент;
      - якщо заголовок вказує на і-ий елемент (first == t), тоді переміщається заголовок на наступний елемент (First = t-> next)
      - копіюється значення інформаційного поля;
      - видаляється і-ий елемент зі списку.
      - повертається значення інформаційного поля.
3. Функція показати всі пробігає по всіх елементах і виводить в консоль значення інформаційного поля. Перегляд елементів здійснюється зліва направо, але легко можна переписати функцію і змінити перегляд елементів справа наліво.
4. Функція очистити видаляє всі елементи в списку і привласнює заголовку пусто Після цієї операції список повертається в початковий стан.

Завдання

1. Ознайомтесь з реалізаціями абстрактних типів даних за допомогою масивів та покажчиків.
2. Для свого варіанту побудуйте блок-схеми алгоритмів для додавання та видалення елементів, використовуючи дві різні реалізації (за допомогою масивів та покажчиків).
3. Напишіть програмні реалізації алгоритмів. Порівняйте результати експериментів по оцінці ефективності реалізації з аналітичною оцінкою складності алгоритмів.
4. Проаналізуйте ефективність та складність алгоритмів і вкажіть переваги та недоліки кожної з реалізацій.
5. Зробіть висновки.

Варіанти АТД для розв’язку завдань

- АТД «черга»
- АТД «однозв’язний лінійний список»
- АТД «двозв’язний лінійний список»
- АТД «стек»

За допомого попередніх варіантів АТД зробіть раціональний розв’язок наступних завдань:

Варіанти завдань.

1. Викладач проводить іспит в 2 етапи, спочатку теоретичне опитування, потім практичні завдання, співбесіда проводиться з кожним студентом окремо і відповідно номеру в заліковій, при чому частина умови завдання передається наступному студенту. Послідовність студентів в першій та другій частині однакова. Спочатку йому необхідно отримати інформацію про студентів, що складають іспит, а потім вже проводити опитування. Допоможіть викладачу раціонально організувати процес проведення іспиту.
2. Викладач проводить іспит в 2 етапи, спочатку теоретичне опитування, потім практичні завдання, співбесіда проводиться з кожним студентом окремо і відповідно номеру в заліковій, при чому частина умови завдання отримується у попереднього студента і передається наступному. Послідовність студентів в першій та другій частині однакова. Спочатку йому необхідно отримати інформацію про студентів, що складають іспит, а потім вже проводити опитування. Допоможіть викладачу раціонально організувати процес проведення іспиту.
3. Викладач проводив за час сесії іспити в декількох групах. Коли він перевіряв роботи він складав їх в стопки в порядку перевірки. В кампусі він повинен занести оцінки груп в тій послідовності, як проходили іспити, при чому поки перша відомість не буде закрита,то наступна не відкривається. Допоможіть викладачу раціонально організувати процес виставлення оцінок.
4. Викладач проводив за час сесії іспити в декількох групах. Коли він перевіряв роботи він складав їх в стопки в порядку перевірки. В кампусі він повинен занести оцінки груп в тій послідовності, як проходили іспити для цього йому необхідно зробити певний перелік. Допоможіть викладачу раціонально організувати процес виставлення оцінок.

5. Викладач проводив за час сесії іспити в декількох групах. Коли він перевіряв роботи він складав їх в стопки в порядку перевірки. Після перевірки він заносить бали у свою звітну таблицю. Заради аналізу успішності студентів за останні 6 років він порівнює результат трьох підряд років і досліджує динаміку, для цього йому необхідно зробити певний перелік. Допоможіть викладачу раціонально організувати процес аналізу успішності.

Склад звіту практичної роботи

- постановка задачі (вказати, який АТД досліджується, та які реалізації обрано);
- блок-схеми реалізацій, на яких виконано аналіз складності алгоритмів (розглянути тільки операції додавання та видалення елемента);
- опис тестових даних (якого характеру дані і для якої перевірки використані);
- скрин-шоти роботи програми для вирішення поставленої задачі;
- результати дослідження у вигляді графіків або діаграм;
- висновки про доцільність використання кожної з реалізацій для типових вхідних даних та про відповідність результатів експериментального дослідження аналітичним оцінкам складності;
- відповіді на контрольні питання.

Контрольні запитання

1. Поясніть різницю між термінами «тип даних», «абстрактний тип даних» «структура даних». Для чого вони застосовується?
2. Проаналізуйте АТД «зв’язний лінійний список»; перерахуйте, які різновиди списків бувають.
3. Проаналізуйте АТД «стек»; перерахуйте, які операції можна виконувати з цим АТД.
4. Перерахуйте послідовність дій при вставці елемента до стеку.
5. Перерахуйте послідовність дій при видаленні елемента зі стеку.
6. Проаналізуйте АТД «черга»; перерахуйте, які операції можна виконувати з цим АТД.
7. Перерахуйте послідовність дій при вставці елемента до черги.

8. Проаналізуйте АТД «однозв’язний лінійний список»; перерахуйте, які операції можна виконувати з цим АТД.
9. Перерахуйте послідовність дій при вставці елемента в середину списку.
10. Перерахуйте послідовність дій при видаленні елемента з середини списку.
11. Перерахуйте послідовність дій при видаленні елемента з кінця списку.

Комп’ютерний практикум 4

Тема: Пошук у масиві. Хешування

Мета роботи: дослідження схем відкритого хешування.

Теоретичні дані

Хеш-функція, або **геш-функція** — функція, що перетворює вхідні дані будь-якого (як правило великого) розміру в дані фіксованого розміру.

Хешування (гешування, англ. *hashing*) — перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються **хеш-функціями**, або **функціями згортання**, а їхні результати називають хешем, **хеш-кодом**, **хеш-сумою**, або **дайджестом повідомлення**

Хеш-таблиця — структура даних, що реалізує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення за ключем.

Основні хеш-функції

1. **Метод ділення**

Метод ділення досить простий – використовується залишок від ділення на M:

h (K) = K mod M

В якості M краще використовувати просте число.

1. **Метод множення**

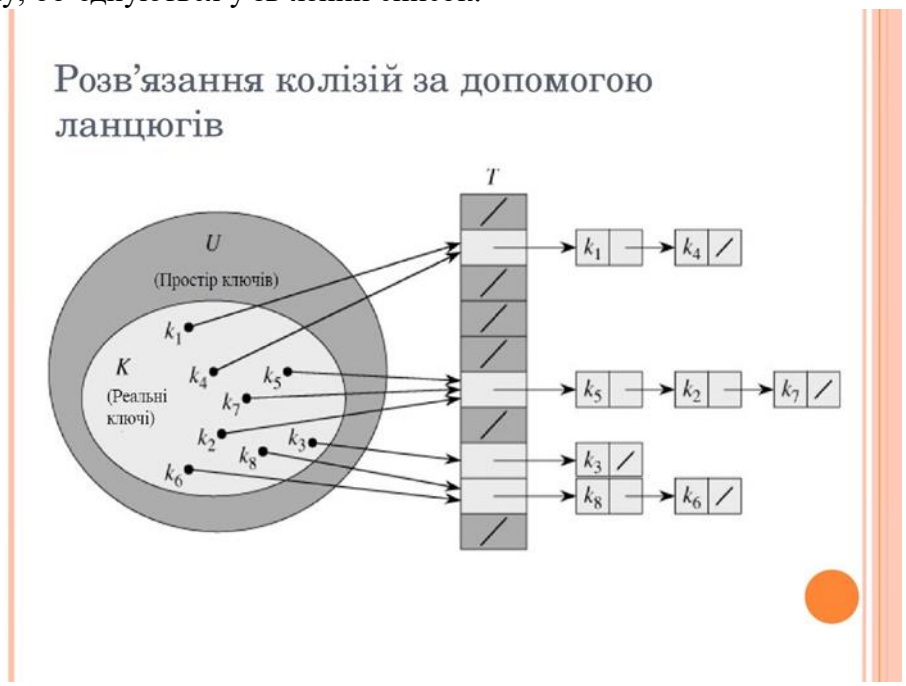
Для мультиплікативного хешування використовується наступна формула [3]:

h (K) = [M * ((C * K) mod 1)]

Тут відбувається множення ключа на константу C, що лежить в інтервалі [0 .. 1]. Після цього береться дробова частина цього виразу і множиться на деяку константу M, обрану таким чином, щоб результат не вийшов за межі хеш-таблиці. Оператор [] повертає найбільше ціле, яке менше аргументу.

Ситуація, коли для різних ключів отримується одне й те саме хеш-значення, називається **колізією**

При **відкритому хешуванні** для вирішення колізій усі елементи, що хешуються в одну комірку, об’єднуються у зв’язний список.



Кожна комірка масиву H є вказівником на зв'язаний список (ланцюжок) пар ключ-значення, відповідних одному і тому самому хеш-значенню ключа. Колізії просто призводять до того, що з'являються ланцюжки довжиною більше одного елемента.

Операції пошуку або видалення елемента вимагають перегляду всіх елементів відповідного ланцюжка, щоб знайти в ньому елемент з заданим ключем. Для додавання нового елемента необхідно додати елемент в кінець або початок відповідного списку, і, у випадку якщо коефіцієнт заповнення стане занадто великим, збільшити розмір масиву H і перебудувати таблицю.

При припущенні, що кожний елемент може потрапити в будь-яку позицію таблиці H з однаковою ймовірністю і незалежно від того, куди потрапив будь-який елемент, пересічний час роботи операції пошуку елемента складає $\Theta(1 + \alpha)$, де α — коефіцієнт заповнення таблиці.

Ситуація, коли для різних ключів отримується одне й те саме хеш-значення, називається **колізією**

В методі, що має назву відкритої адресації, всі записи зберігаються в самому масиві H. Коли додається новий запис, масив перевіряється в певному порядку, доки не буде знайдена вільна комірка, куди буде доданий елемент. У випадку пошуку елемента, масив сканується в тій самій послідовності, доки потрібний запис або порожня комірка не буде знайдена. Останнє — означає відсутність елемента. Назва «відкрита адресація» показує, що положення («адреса») елемента не визначається його хеш-значенням. Цей метод також називають **закритим хешуванням.** Загалом, послідовність, у якій переглядаються комірки хеш-таблиці, залежить лише від ключа елемента.

Варіанти переборів (нумерація елементів послідовності спроб і комірок хеш-таблиці при переборі ведеться від нуля, а N — розмір хеш-таблиці.

1. Лінійне зондування: комірки хеш-таблиці послідовно переглядаються з деяким фіксованим інтервалом k між комірками (зазвичай,), тобто i-й елемент послідовності — це комірка з номером (hash(x) + ik) mod N. Для того, щоб всі комірки виявились перебраними по одному разу, необхідно, щоб k було взаємно-простим з розміром геш-таблиці.
2. Квадратичне зондування: інтервал між комірками з кожним кроком збільшується на константу, що задається в загальному вигляді з допомогою деякого квадратичного поліному k1i+k2i2. У найпростішому випадку k1 = 0, k2 = 1 для i = 0,1,2,3,… отримаємо:

   h0 = hash(x) mod N, h1 = (hash(x) + 12) mod N, h2 = (hash(x) + 22) mod N, h3 = (hash(x) + 32) mod N, …
3. Подвійне хешування: інтервал між комірками фіксований, як при лінійному переборі, але, на відміну від нього, розмір інтервалу обчислюється другою, допоміжною хеш-функцією, тобто може бути різним для різним ключів. Значення цієї хеш-функції мають бути ненульовими і взаємно простими з розміром хеш-таблиці, що найпростіше всього зробити, якщо взяти просте число як розмір, і вимагати, щоб допоміжна хеш-функція приймала значення від 1 до N − 1.

Недоліком усіх схем відкритої адресації є те, що кількість елементів, які можуть зберігатися в таблиці може досягти кількість комірок в масиві.

Схеми відкритої адресації також накладають суворіші вимоги до хеш-функції: окрім рівномірного розподілу значень по всьому масиву, функція має також мінімізувати скупчення хеш-значень, що слідують одна за одною в послідовності спроб.

Хешування зозулею

Альтернатива відкритої адресації — англ. Cuckoo hashing,— забезпечує постійний час пошуку і сталий амортизований час для вставок і вилучень. Цей метод використовує дві чи більше хеш-функції, а це означає, що будь-яка пара ключ/значення може знаходитись в двох або більше місцях. Для пошуку використовується перша хеш-функція; якщо ключ/значення не знайдено, то використовується друга хеш-функція, і так далі. Якщо колізія відбувається під час вставки, то ключ повторно хешується другою хеш-функцією, щоб відобразити його в інший бакет. Якщо всі функції хешування використані, а колізія не розв'язана, то старий ключ на місці колізії видаляється, щоб звільнити місце для нового ключа, і цей попередній старий ключ повторно хешується однією з наступних хеш-функцій, які відображають його в інший бакет. Якщо це також призводить до колізії, то процес повторюється, поки колізія не зникне

або процес пройде через всі бакети в таблиці. Таким шляхом оптимізується використання пам'яті.

Hopscotch хешування

Інший альтернативний метод, поєднує в собі хешування зозулею та лінійне з але таким чином, щоб обійти обмеження цих методів. Зокрема, він добре працює навіть тоді, коли коефіцієнт завантаження таблиці зростає до 0,9

Завдання

1. Ознайомитися з теоретичними даними стосовно основних типів хеш-функцій.
2. Обрати для досліджень дві хещ-функції з вказаних в теоретичній складовій практикуму або за власними міркуваннями.
3. Зробити постановку задачі з описом обраних хеш-функцій та методу вирішення колізій при відкритому хешуванні
4. Написати програмні реалізації поставленої задачі з використанням двох хеш-функцій для відкритого хешування.
5. Обрати хеш-функцію та методи вирішення колізій для закритого хешування.
6. Зробити постановку задачі з описом обраної хеш-функції та методів вирішення колізій при закритому хешуванні
7. Написати програмні реалізації двох методів вирішення колізій для закритого хешування.
8. Відпрацювати роботу програмного забезпечення для додавання, видалення елемента, вирішення колізій для поставленої задачі і для даних загального типу.
9. Проаналізувати їх ефективність та складність і вказати переваги та недоліки кожної з реалізацій.
10. Зробити висновки.
11. Дати відповіді на контрольні питання.

Склад звіту

- формулювання постановки задачі (вказати обов’язково які хеш-функції обрано, методи уникнення колізій для обох типів хешування);
- розробити програмне забезпечення для розв’язку поставленої задачі;
- протестувати програмне забезпечення для поставленої задачі з додаванням відповідних скрін-шотів;
- відображення розглянутих операції вдалого пошуку та невдалого пошуку (додавання) елемента, видалення елемента. Результати дослідження у вигляді графіків окремо для вдалого та невдалого пошуку, видалення – дві хеш-функції в одній координатній сітці, по осі абсцис – поточна відносна кількість елементів у таблиці (відсоток заповнення таблиці), по осі ординат – середня довжина списків (або середній час на виконання операції). Вхідні дані можна генерувати за допомогою випадкових чисел, але в одному досліді один і той самий набір використати для обох хеш-функцій. Обов’язково провести 3-5 серій експериментів з різними вхідними даними і результати усереднити.
- Зроблені висновки про те, яка з розглянутих хеш-функцій краща, та при якому відсотку заповнення таблиці потрібно проводити її реструктуризацію.
- Відповіді на контрольні питання.

Варіанти завдань

1. Скласти список дат народження дівчат потоку для можливості в подальшому вручення їм квітів на день народження, а для того щоб зберегти це в таємниці згенерувати відповідну хеш-таблицю.
2. Скласти список дат народження хлопців потоку для можливості в подальшому вручення їм смаколиків на день Козацтва, а для того щоб зберегти це в таємниці згенерувати відповідну хеш-таблицю.

3. Скласти список групи із середнім рейтинговим балом за 1 семестр, для можливості в подальшому доступу до цих даних виключно старости згенерувати відповідну хеш-таблицю.

Контрольні питання

1. Сформулюйте та поясніть термін «задача пошуку».
2. Розкрийте суть абстрактного типу даних «Словник».
3. Поясніть, виходячи з чого обирається конкретна реалізація абстрактного типу даних «Словник» для розв'язку задачі та перерахуйте можливі варіанти реалізації.
4. Перерахуйте та прокоментуйте переваги та недоліки реалізації словника, що були визначені у відповіді на попередні питання.
5. Що таке хеш-функція?
6. Що таке хеш-таблиця?
7. Що таке відкрите хешування?
8. Перерахуйте порядок дій при закритому хешуванні.
9. Яку ситуацію називають колізією при використанні закритого хешування?
10. Що таке відкрита адресація?
11. Що таке лінійне зондування?
12. Що таке подвійне зондування?
13. Що таке хешування зозулею7
14. Які альтернативні методи хешування відносно відкритої адресації Ви знаєте?

Комп'ютерний практикум 5

Тема: Двійкові дерева пошуку.

Мета роботи: дослідження характеристик продуктивності двійкових дерев пошуку.

Теоретичні дані

Двійкове (або Бінарне) де́рево по́шуку (англ. binary search tree, BST) в інформатиці — двійкове дерево, в якому кожній вершині x зіставлене певне значення val[x]. При цьому такі значення повинні задовольняти умові впорядкованості:

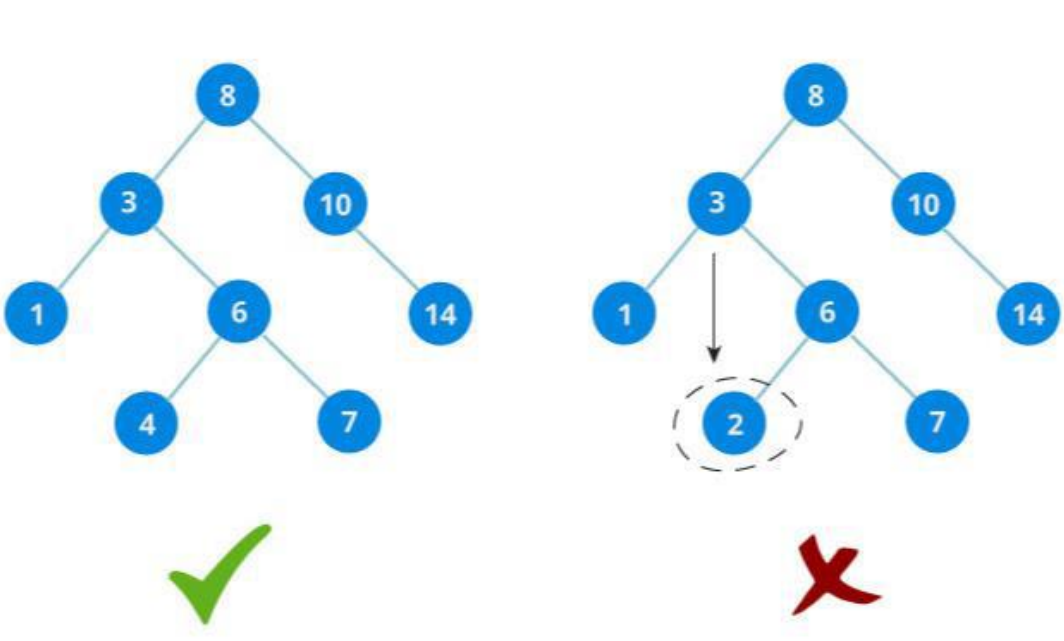
– нехай x — довільна вершина двійкового дерева пошуку.
– Якщо вершина y знаходиться в лівому піддереві вершини x, то val[y] ≤ val[x].
– Якщо y знаходиться у правому піддереві x, то val[y] ≥ val[x].

Таке структурування дозволяє надрукувати усі значення у зростаючому порядку за допомогою простого алгоритму центрованого обходу дерева.

Представляється таке дерево вузлами наступного вигляду:
*Node = (element, key, left, right, parent). Доступ до дерева T здійснюється за допомогою посилання root.

Властивості, які відрізняють двійкове дерево пошуку від звичайного двійкового дерева:

– Всі вузли лівого піддерева менше кореневого вузла.
– Всі вузли правого піддерева більше кореневого вузла.
– Обидва піддерева кожного вузла також є BST, тобто вони мають дві вищевказані властивості.



Двійкове дерево праворуч не є двійковим деревом пошуку, тому що праве піддерево вузла "3" містить значення, яке менше за нього.

Існують основні операції, які можна виконувати в двійковому дереві пошуку. Перші дві можна вважати основними.

1. Перевірка, чи присутнє число в двійковому дереві пошуку.

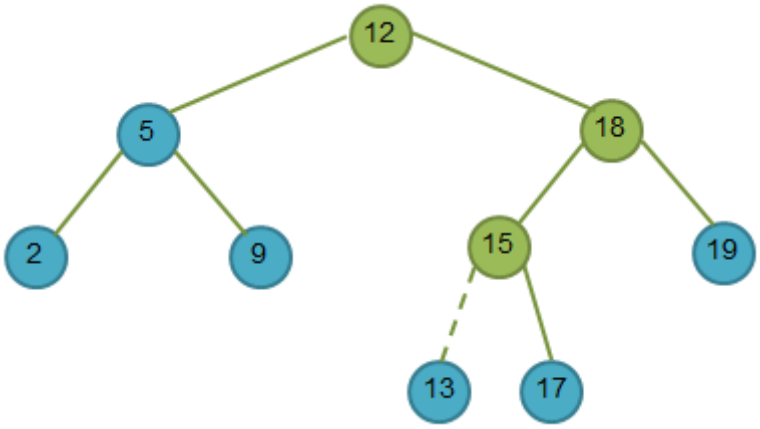
Доречно використовувати рекурсію, яка дозволяє через пошук та процедуру root повернути остаточний результат. При чому при знаходженні результату необхідно його поширювати на кожному кроці рекурсії.

2. Вставка значення в двійкове дерево пошуку (BST)

Вставка значення в правильну позицію аналогічна пошуку, тому що при цьому необхідно слідувати правилу, згідно з яким ліве піддерево менше кореневого, а праве піддерево більше кореневого.

Після під'єднання вузла потрібно вийти з функції, не завдаючи шкоди іншій частині дерева. Тут доречно також використовувати рекурсію в кінці для повертання до гори дерева без змін.

Схема додавання елементу 13



3. Пошук мінімуму і максимуму

Щоб знайти мінімальний/максимальний елемент в бінарному дереві пошуку, необхідно просто дотримуватися вказівниками від кореня дерева, поки не зустрінеться шукане значення

4. Пошук наступного і попереднього елемента

1.) Реалізація з використанням інформації про батька.

Якщо у вузла є праве піддерево, то наступний за ним елемент буде мінімальним елементом в цьому піддереві. Якщо у нього немає правого піддерева, то потрібно слідувати вгору, поки не зустріниться вузол, який є лівим дочірнім вузлом свого батька. Пошук попереднього виконується аналогічно. Якщо у вузла є ліве піддерево, то попередній йому елемент буде максимальним елементом в цьому піддереві. Якщо у нього немає лівого піддерева, то потрібно слідувати вгору, поки не зустрінеться вузол, який є правим дочірнім вузлом свого батька.

2.) Реалізація без використання інформації про батька

В цьоиу випадку розглядається пошук наступного елемента для деякого ключа. Пошук починається з кореня дерева, зберігаючи поточний і останній відвіданий вузли, ключ якого більший за заданий. Далі відбувається спуск вниз по дереву, як в алгоритмі пошуку вузла. При цьому розглядається ключ поточного вузла. Якщо поточний ключ меньший від заданого, то наступний вузол знаходиться в правому піддереві (в лівому піддереві все ключі менші за поточний). В іншому випадку поточний ключ може бути наступним для заданого ключа або наступний вузол міститься в лівому піддереві. Далі відбувається перехід до потрібного піддерева і повторення тих же самих дій.

Аналогічно реалізується операція пошуку попереднього елемента.

5. Видалення

1.) Нерекурсивна реалізація

Для видалення вузла з бінарного дерева пошуку потрібно розглянути три можливі ситуації.

1. Якщо у вузла немає дочірніх вузлів, то у його батька потрібно просто замінити покажчик на *null*.
2. Якщо у вузла є тільки один дочірній вузол, то потрібно створити новий зв'язок між батьком вузла, що видаляється та його дочірнім вузлом.

3. Якщо у вузла два дочірніх вузла, то потрібно знайти наступний за ним елемент (у цього елемента не буде лівого нащадка), його правого нащадка підвісити на місце знайденого елемента, а вузол, що видаляється замінити знайденим вузлом.

Таким чином, властивість бінарного дерева пошуку не буде порушено. Дана реалізація видалення не збільшує висоту дерева.

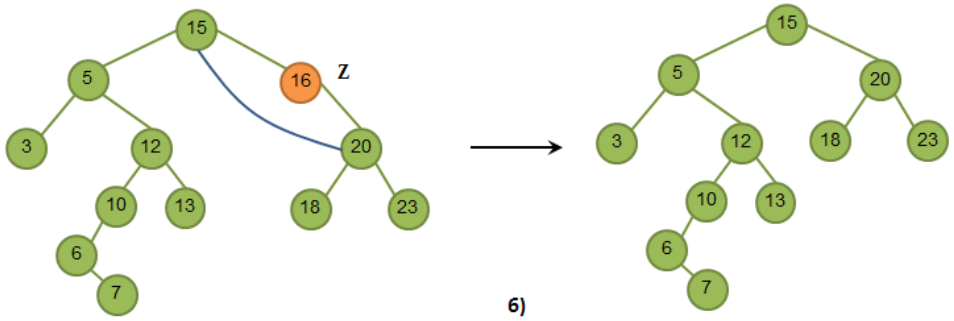
2.)Рекурсивна реалізація

При рекурсивном видаленні вузла з бінарного дерева потрібно розглянути три випадки:

1. видаляється елемент знаходиться в лівому поддереве поточного поддерева,
2. видаляється елемент знаходиться в правому піддереві або
3. видаляється елемент знаходиться в корені.

У двох перших випадках потрібно рекурсивно видалити елемент з потрібного піддерева. Якщо елемент, що видаляється знаходиться в корені поточного піддерева і має два дочірніх вузла, то потрібно замінити його мінімальним елементом з правого піддерева і рекурсивно видалити цей мінімальний елемент з правого піддерева. В іншому випадку, якщо елемент, щоьвидаляється має один дочірній вузол, потрібно замінити його нащадком.

Схема видалення елементу 6



Збалансоване дерево

Дерево пошуку з мінімальною висотою називається **збалансованим**, тобто таким, в якому висота лівого і правого піддерев відрізняються не більше ніж на одиницю.

Збалансоване воно тому, що в такому дереві в середньому потрібно витратити найменшу кількість операцій для пошуку.

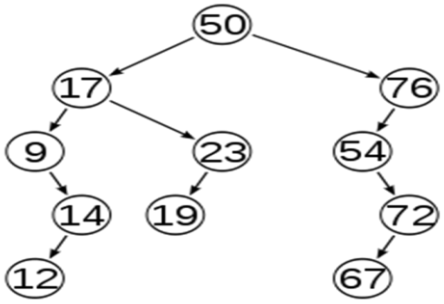
Щоб дерево мало мінімальну висоту, кількість вузлів лівого і правого піддерев повинні максимально наближатися один до одного.

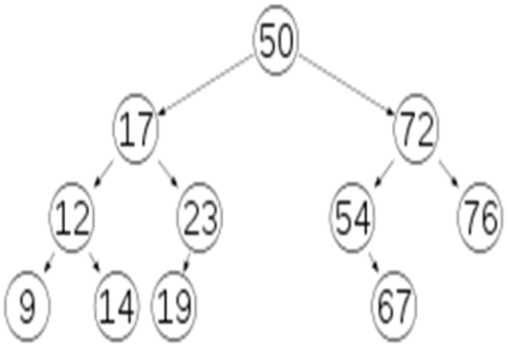
Для побудови такого дерева можна використати наступний принцип:

– отримання відсортованого масиву;

– середина кожного підрозділу масиву стає кореневим вузлом, а ліва і права частини - відповідними для нього піддеревами;

– оскільки масив відсортований, то отримане дерево відповідає визначенню бінарного дерева пошуку.





незбалансоване дерево

дерево після балансування

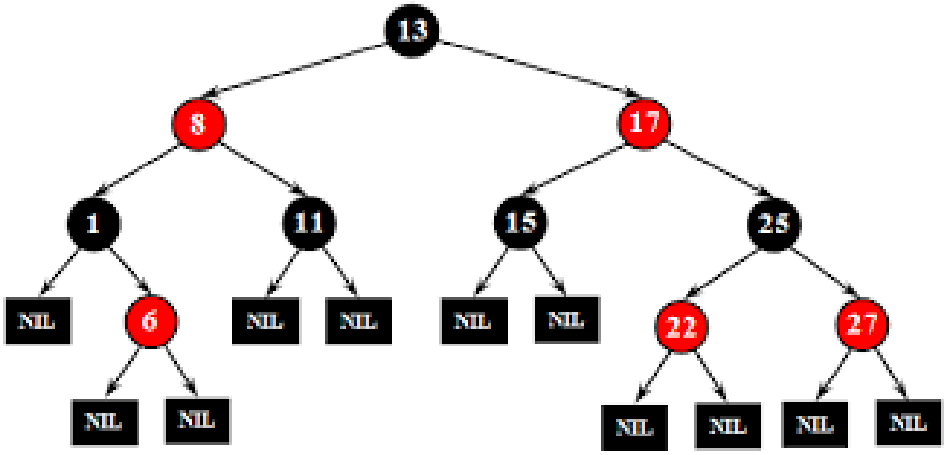
Червоно-чорні дерева

Червоно-чорні дерева — різновид збалансованих дерев, в яких за допомогою спеціальних трансформацій гарантується, що висота дерева h не буде перевищувати O (log n). Зважаючи на те, що час виконання основних операцій на бінарних деревах (пошук, видалення, додавання елементу) є O(h), ці структури даних на практиці є набагато ефективнішими, аніж звичайні бінарні дерева пошуку.

Бінарне дерево називається червоно-чорним, якщо воно має такі властивості:

- кожна вершина або червона, або чорна;
- корінь дерева — чорний;
- ;кожний лист (NIL) — чорний
- якщо вершина червона, обидві її дочірні вершини чорні (інакше, батько червоної вершини — чорний);;
- усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин.

Такі властивості надають червоно-чорному дереву додаткового обмеження: найдовший шлях з кореня до будь-якого листа перевищує найкоротший шлях не більше ніж вдвічі. В цьому сенсі таке дерево можна назвати збалансованим. Зважаючи на те, що час виконання основних операцій з бінарними деревами пошуку залежить від висоти, таке обмеження гарантує їхню ефективність в найгіршому випадку, чого звичайні бінарні дерева гарантувати не можуть.



Завдання

1. Проаналізуйте завдання свого варіанту
2. Сформулюйте постановку задачі.
3. Створіть базовий граф.
4. Напишіть програмну реалізацію розв'язку задачі.
5. Проведіть серію експериментів для оцінки продуктивності роботи алгоритму при вирішенні задачі.
4. Збалансуйте базовий граф.
5. Сформулюйте, які події (зміни) відбуваються з учасниками задачі або/і умовами при балансуванні.
6. Напишіть програмну реалізацію розв'язку задачі з використанням збалансованого або червоно-чорного дерев.
7. Проведіть серію експериментів для оцінки продуктивності роботи алгоритму при вирішенні задачі, зробіть висновки.
8. Зробіть порівняння розв'язку задачі через збалансоване дерево або червоно-чорне дерево та звичайне дерево пошуку.
9. Виявіть фактори, що впливають на продуктивність найбільше.
10. Зробіть висновки
11. Дайте відповіді на контрольні питання.

Варіанти завдань

1. Допоможіть лепрікону якомога швидше знайти скарб зі злитків золота в одній з печер. Кожна печера містить різну кількість злитків, печери з'єднані тунелями. Для цього треба побудувати каскад тунелів і печер. Визначте найзручнішу печеру для початку пошуку та покажіть шлях до заданого скарбу. Кількість шуканих злитків задається з клавіатури. Якщо такої печери не існує, то треба її викопати та з'єднати тунелем і там чарівним чином з'явиться золото. Знову покажіть лепрікону шлях.
2. Допоможіть армаді з найменшими втратами перемогти піратів. Армада має кораблі з різною кількістю моряків. Адміральський корвет має середню кількість моряків в порівнянні з іншими. Перемогти піратське судно може лише корабель з неменшим числом моряків, але зайві задіяні люди не допустимі. Кількість піратів на судні задається з клавіатури. Сигнал для атаки подає адміральський корвет, йому необхідно обрати відповідний корабель. За допомогою прапорців кожен з кораблів може зв'язатися лише з двома кораблями та отримати дані про кількість матросів. Команда атаки передається ланцюжком між кораблями. Якщо кількість матросів не відповідає кількості піратів з'являється новий корабель з відповідною кількістю і сигнал на атаку повторюється. Для обох випадків показати шлях проходження сигналу.
3. Є певна кількість приладів. Кожний прилад має свою унікальну потужність. Необхідно швидко знайти той прилад, що точно відповідає заданій потужності заряду робота. Потужність задається з клавіатури. Від кожного приладу є лише два шляхи для робота, але він точно знає де потужність буде збільшуватися та зменшуватися в порівнянні з поточним. Якщо для робота не буде знайдено відповідний пристрій, то для його порятунку треба терміново додати такий пристрій. Для обох випадків побудувати шлях знаходження пристрою.

Для кожного варіанту кількість елементів змінюється декілька разів починаючи з 15 та йде на збільшення.

Склад звіту практичної роботи

- постановка задачі;
- базовий граф;
- блок-схеми реалізацій, на яких виконано аналіз складності алгоритмів (розглянути тільки операції вдалого пошуку та невдалого пошуку (додавання) елемента);

- збалансоване або червоно-чорне дерева, які створені на основі базового графу;
- блок-схеми реалізацій, на яких виконано аналіз складності алгоритмів (розглянути тільки операції вдалого пошуку та невдалого пошуку (додавання) елемента, видалення елемента);
- результати дослідження у вигляді графіків для вдалого та невдалого пошуку, видалення – дві криві в одній координатній сітці для збалансоване дерево або червоно-чорне дерево та звичайне дерево пошуку, по осі абсцис – поточна кількість елементів, по осі ординат – середня довжина списків (або середній час на виконання операції);
- висновки про результати досліджень;
- висновки про порівняння розв'язку задачі через збалансоване дерево або червоно-чорне дерево та звичайне дерево пошуку ;

- відповіді на контрольні питання.

Контрольні питання

1. Чим відрізняються дерева бінарного пошуку та для чого вони використовуються?
2. Перерахуйте та проілюструйте послідовність дій при додаванні вузла в бінарне дерево пошуку.
3. Перерахуйте та проілюструйте послідовність дій при видаленні вузла з бінарного дерева пошуку.
4. Прокоментуйте характеристики продуктивності дерев бінарного пошуку на конкретних прикладах.
5. Що таке дерево рішень? Які особливості його застосування?
6. Перерахуйте та прокоментуйте переваги та недоліки 2-3-4-дерев. Поясніть основні принципи їх побудови.
7. Перерахуйте та прокоментуйте переваги та недоліки червоно-чорних дерев. Поясніть основні принципи їх побудови.
8. Перерахуйте та прокоментуйте переваги та недоліки навантажених дерев. В яких випадках їх використання найефективніше?

Комп'ютерний практикум 6

Тема: Алгоритми сортування. Методи сортування малих обсягів даних**.**

Мета роботи: Дослідження та порівняння алгоритмів сортування.

Теоретичні дані

Сортування вибором

Сортування вибором — простий алгоритм сортування лінійного масиву, на основі вставок. Неефективний при сортуванні великих масивів, і в цілому, менш ефективним за подібний алгоритм сортування включенням. Сортування вибором вирізняється більшою простотою, ніж інші алгоритми, і в деяких випадках, вищою продуктивністю.

Алгоритм працює таким чином:

1. Знаходить у списку найменше значення
2. Міняє його місцями із першим значеннями у списку
3. Повторює два попередніх кроки, доки список не завершиться (починаючи з наступної позиції)

Фактично, таким чином розділяється список на дві частини: перша (ліва) — повністю відсортована, а друга (права) — ні.

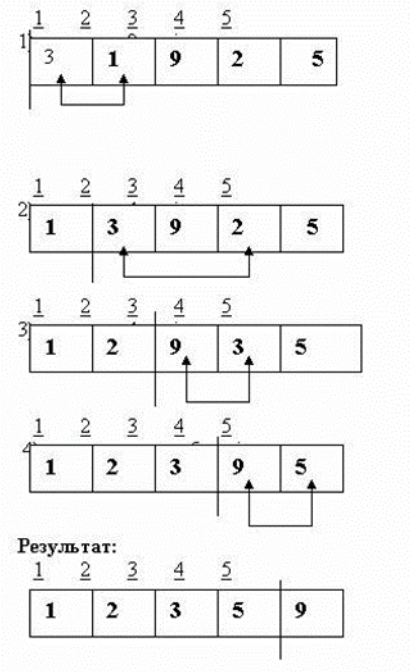
Сортування вставками (включенням)

Сортування включенням — простий алгоритм сортування на основі порівнянь. На великих масивах мало ефективний.

Алгоритм працює таким чином:

1. порівнюються другий та перший елемент вихідного масиву. Якщо порядок між ними, в залежності від типу сортування (за зростанням чи за спаданням) порушений, то перший елемент пересувається на одну позицію вправо. Тепер відсортований масив складається з двох елементів.
2. наступний (третій, четвертий і так далі) елемент черзі порівнюється, починаючи з кінця, з іншими елементами в уже відсортованому масиві. Якщо порядок між порівнюваними елементами порушений, то вони міняються місцями, якщо ні, то вставка нового елемента закінчена. Відбувається перехід до п.1. ітерації.

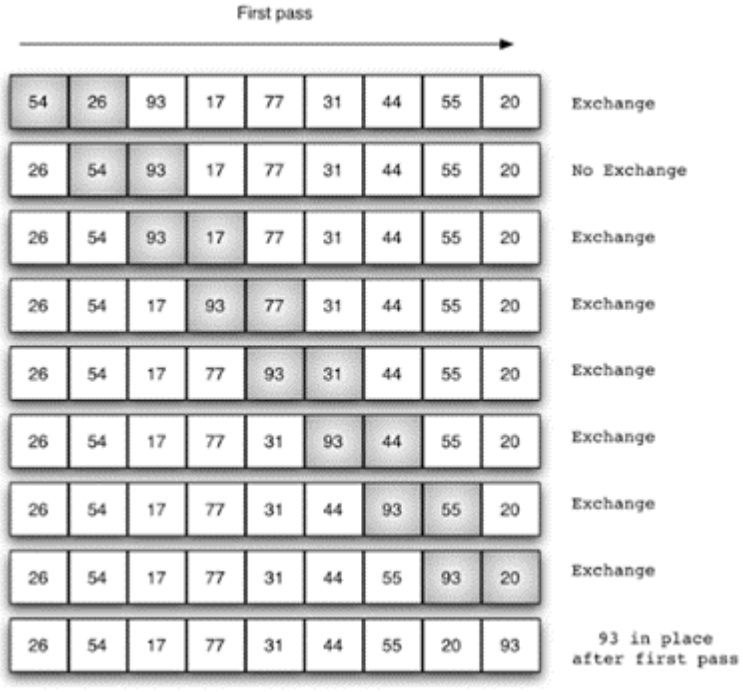
Метод вибору чергового елементу з початкового масиву довільний; може використовуватися практично будь-який алгоритм вибору. Зазвичай (і з метою отримання стійкого алгоритму сортування), елементи вставляються за порядком їх появи у вхідному масиві.



Сортування бульбашкою

Сортування обміном або сортування бульбашкою є простим алгоритмом сортування. Алгоритм працює наступним чином:

1. у поданому наборі даних (списку чи масиві) порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями;
2. прохід по списку продовжується доти, доки дані не будуть відсортованими.



Сортування підрахунком

Сортування підрахунком (англ. *Counting sort*) — алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості *різних* елементів.

Алгоритм працює наступним чином:

1. підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві;

2. спираючись на ці дані вираховується на якому місці має стояти кожен елемент, а потім за один прохід всі елементи розташовуються на своїх місцях.

Сортування Шелла

Сортува́ння Ше́лла — це алгоритм сортування, що є узагальненням сортування включенням.

Алгоритм базується на принципах:

1. Сортування включенням ефективне для майже впорядкованих масивів.
2. Сортування включенням неефективне, тому що переміщує елемент тільки на одну позицію за раз.

Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що розташовані на різній відстані один від одного.

Сортування Шелла не є стабільним методом.

Принцип роботи алгоритму:

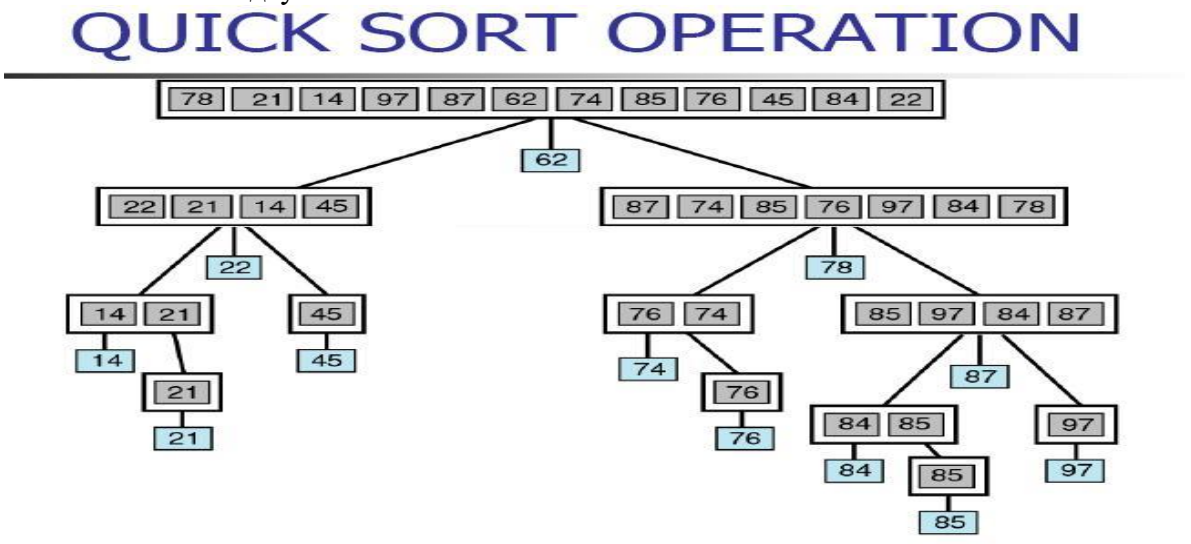
1. На початку обираються m-елементів: $d_1, d_2, \dots d_m$ причому $d_1 > d_2 > \cdots > d_m = 1,$.
2. Потім виконується m впорядкувань методом включення, спочатку для елементів, що стоять через d_1, потім для елементів через d_2 і т. д. до $d_m = 1$.
3. Ефективність досягається тим, що кожне наступне впорядкування вимагає меншої кількості перестановок, оскільки деякі елементи вже стали на свої місця.
4. Оскільки $d_m = 1$, то на останньому кроці виконується звичайне впорядкування включенням всього масиву, а отже кінцевий масив буде впорядкованим.

Швидке сортування

Швидке сортування (англ. *Quick Sort*) — алгоритм сортування, який не потребує додаткової пам'яті і виконує у середньому $\boldsymbol{O(nLog(n)}$ операцій. Однак, у найгіршому випадку робить $\boldsymbol{O(n^2)}$ порівнянь. Так як, алгоритм використовує дуже прості цикли і операції, він працює швидше за інші алгоритми, що мають таку ж асимптотичну оцінку складності. Наприклад, зазвичай більш ніж удвічі швидший порівняно з сортуванням злиттям.

Алгоритм складається з трьох кроків:

1. Вибрати елемент з масиву. Має назву опорний.
2. Розбиття: перерозподіл елементів в масиві таким чином, що елементи менше опорного розміщаються перед ним, а більше або рівні після.
3. Рекурсивно застосувати перші два кроки до двох підмасивів зліва і праворуч від опорного елемента. Рекурсія не застосовується до масиву, в якому тільки один елемент або відсутні елементи.



У ранніх реалізаціях, як правило, опорним вибирався перший елемент, що знижувало продуктивність на відсортованих масивах. Для поліпшення ефективності може вибиратися середній, випадковий елемент або (для великих масивів) медіана першого, середнього і останнього елементів. Медіана всієї послідовності є оптимальним опорним елементом, але її обчислення є занадто трудомістким для використання в сортуванні.

Для поліпшення продуктивності при великій кількості однакових елементів в масиві може бути застосована процедура розбиття масиву на три групи: елементи менші опорного, рівні йому і більше нього. Іноді його називають ще «товстим розбиттям».

Покращення

Покращення алгоритму спрямовані, в основному, на усунення або пом'якшення вищезазначених недоліків, внаслідок чого всі їх можна розділити на три групи:

1. надання алгоритму стійкості,
2. усунення зменшення продуктивності спеціальним вибором опорного елемента,
3. захист від переповнення стека викликів через велику глибину рекурсії при невдалих вхідних даних.

Проблема нестійкості вирішується шляхом розширення ключа вихідним індексом елемента в масиві. У разі рівного розподілу основних ключів порівняння проводиться за індексом, виключаючи, таким чином, можливість зміни взаємного положення рівних елементів. Ця модифікація вимагає додаткового обсягу O (n) пам'яті і одного повного проходу по масиву для збереження вихідних індексів.

Зменшення швидкості після невдалої спроби вхідних даних вирішується за двома різними напрямками:

1. зниження ймовірності виникнення гіршого випадку шляхом спеціального вибору опорного елемента;
2. застосування різних технічних прийомів, які забезпечують сталу роботу на невдалих вхідних даних.

Для першого напряму:

1. Вибір середнього елемента. Усуває зменшення якості алгоритму для попередньо відсортованих даних, але залишає можливість випадкової появи або навмисного підбору «поганого» масиву.
2. Вибір медіани з трьох елементів: першого, середнього і останнього. В порівнянні з вибором середнього елемента знижує ймовірність виникнення гіршого випадку,
3. Випадковий вибір. Імовірність випадкового виникнення гіршого випадку стає дуже малою, а навмисний підбір - практично нездійсненним. Очікуваний час виконання алгоритму сортування становить O (n lg n).

Недолік всіх ускладнених методів вибору опорного елемента -додаткові витрати ресурсів, але вони не дуже великі.

Для вирішення проблеми відмови програми через велику глибину рекурсії застосовують наступні методи:

1. При досягненні небажаної глибини рекурсії переходити на сортування іншими методами, що не вимагають рекурсії. Прикладом такого підходу є алгоритм Introsort або деякі реалізації швидкого сортування в бібліотеці STL. Можна помітити, що алгоритм дуже добре підходить для такого роду модифікацій, так як на кожному етапі дозволяє виділити безперервний відрізок вихідного масиву, призначений для сортування, і те, яким методом буде відсортований цей відрізок, ніяк не впливає на обробку інших частин масиву.
2. Модифікація алгоритму, що усуває одну гілку рекурсії: замість того, щоб після поділу масиву викликати рекурсивно процедуру поділу для обох знайдених підмасивів, рекурсивний виклик робиться тільки для меншого підмасиву, а більший обробляється в циклі в межах цього ж виклику процедури. З точки зору ефективності в середньому випадку різниці практично немає: витрати ресурсів на додатковий рекурсивний виклик і на організацію порівняння довжин підмасивів і циклу - приблизно одного порядку. Проте глибина рекурсії ні при яких обставинах

не перевищить log2n, а в гіршому випадку виродженого поділу вона взагалі буде не більше 2 - вся обробка пройде в циклі першого рівня рекурсії. Застосування цього методу не врятує від дуже великого зменшення продуктивності, але переповнення стека не буде.

3. Розбиття масиву не на дві, а на три частини.

Завдання

1. Ознайомтесь з різновидами простих методів сортування та особливостями швидкого метода сортування .
2. Напишіть програмні реалізації вирішення задачі, використовуючи прості алгоритми сортування, за своїм варіантом для кожного з трьох випадків. При обиранні алгоритму для 2 та 3 випадку звертайте увагу на особливості даних,
3. Напишіть програмну реалізацію вирішення задачі на основі швидкого методу сортування за своїм варіантом для кожного з трьох випадків, для 3 випадку зверніть увагу на особливості даних та застосуйте варіант покращення роботи алгоритму.
4. Проведіть серію експериментів для оцінки продуктивності реалізацій для всіх трьох випадків для простих алгоритмів та швидкого сортування. Порівняйте отримані результати .
5. За результатами експериментів зробіть висновки.
6. Зробіть висновки, чи отримано практичне підтвердження про продуктивність сортування, що визначено теоретично; для яких даних досліджувані алгоритми підходять якнайкраще, а у яких випадках використовувати їх недоцільно.
7. Дайте відповіді на контрольні питання.

Варіант 1

Жадібний леприкон зібрав всі можливі скарби в своїх печерах та вирішив їх розкласти по скриня в тій же кількості, що і знаходив.. Майстер по виготовленню скринь робить їх поступово від найменшої до найбільшої, тому вимагає надати йому відповідний перелік розмірів скарбів. Леприкон дав йому відповідний перелік. Але поки майстер розмірковував, гном пішов до сусіда та взяв частину його скарбів в мішечках та додав до своїх і знову пішов до майстра. Сформував новий список скарбів. Майстер прийняв замовлення. Але сусідам сподобалась така ідея і вони принесли свої скарби в мішечках леприкону, щоб розкласти по скринях. Він побачив, що скарбів в мішечках часто повторювалася. Але знову пішов до майстра з остаточним списком. Допоможіть йому дати майстру інформацію для кожного з трьох випадків.

Варіант 2

Адмірал флоту вирішив провести профілактичні ремонтні роботи флоту. Але працівники доку поставили умову прийняття кораблів в залежності від розміру (розмір залежить від кількості матросів). Штаб сформував відповідний перелік. Поки формувався список приплили ще кораблі, і виникла потреба переробити перелік. Список обновили. Але інші адмірали вирішили доєднатися до таких робіт. З'ясувалося, що серед всіх кораблів кількість матросів часто повторюється. Сформували остаточний перелік кораблів для ремонту. Допоможіть їм сформувати списки для кожного з трьох випадків.

Варіант 3

Мережу зарядних пристроїв вирішили оновити. Але це треба робити поступово, замінюючи пристрої в порядку збільшення їх потужності. Фахівці зробили перелік, але з'ясувалося, що після цього додалася додаткова частина мережі, яку необхідно включити до списку. Відбулося переформатування списку. Раптово компанія вирішила об'єднати всі мережі. Виявилося, що в цій купі пристроїв є багато з однаковою потужністю. Зробили остаточний перелік. Допоможіть сформувати перелік пристроїв для кожного з трьох випадків.

Склад звіту практичної роботи

- постановка задачі;
- опис швидкого алгоритму сортування;
- блок-схема реалізації, на якій виконано аналіз складності алгоритму;
- результати дослідження у вигляді графіків в координатах;
- висновки про доцільність використання кожного з алгоритмів для даних задачі та про відповідність результатів експериментального дослідження аналітичним оцінкам складності;
- відповіді на контрольні питання.

Контрольні питання

1. Сформулюйте задачу сортування.
2. Поясніть, виходячи з чого обирається алгоритм сортування для розв’язку конкретної задачі.
3. Поясніть чому задача сортування в програмуванні до сих пір повністю не визначена.
4. Що таке стійкість алгоритмів?
5. Які з простих алгоритмів стійкі, а які ні? Поясніть.
6. Зробіть порівняльну характеристику методів сортування вибором, сортування вставками, бульбашкового сортування. Вкажіть, на яких вхідних даних ці методи працюють найкраще.
7. Зробіть порівняльну характеристику методів сортування вибором та сортування методом підрахунку. Вкажіть, на яких вхідних даних ці методи працюють найкраще.
8. Порівняйте непряме (по індексам або покажчикам) та пряме сортування вибором. Вкажіть, на яких вхідних даних ці методи працюють найкраще.
4. Перерахуйте та прокоментуйте переваги та недоліки сортування методом швидкого сортування для різних послідовностей даних. Вкажіть, на яких вхідних даних цей метод працює найкраще.

Комп'ютерний практикум 7

Тема: Алгоритми сортування. Методи сортування великих обсягів даних

Мета роботи: Дослідження та порівняння алгоритмів сортування.

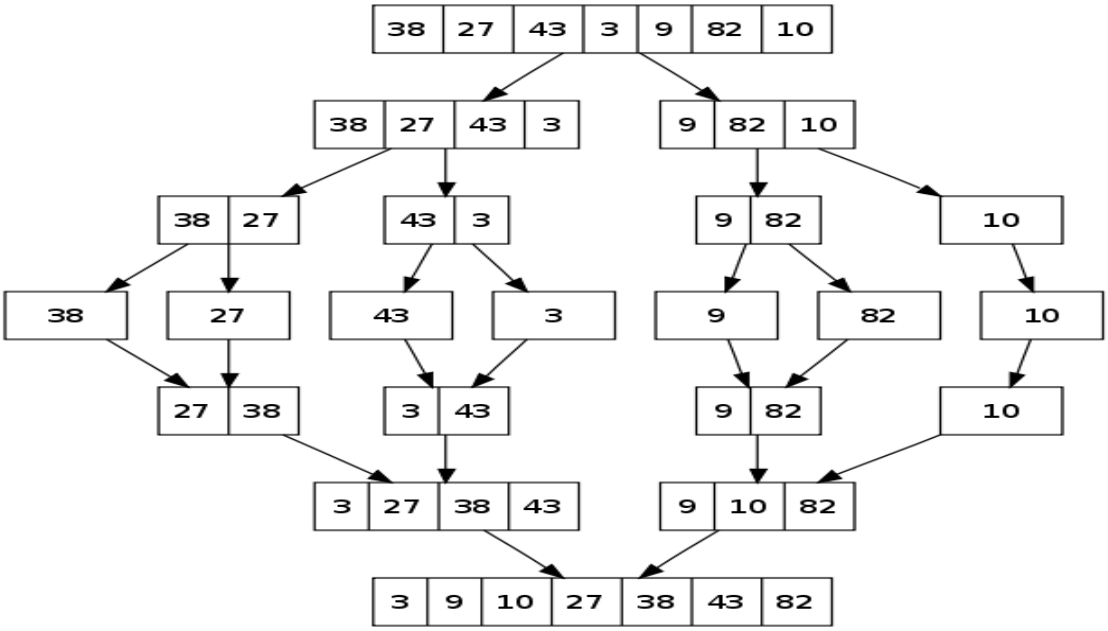
Теоретичні дані

Сортування злиттям

Сортування злиттям (англ. Merge sort) - алгоритм сортування, який впорядковує списки (або інші структури даних, доступ до елементів яких можна отримувати тільки послідовно, наприклад - потоки) в певному порядку. Це сортування - хороший приклад використання принципу «розділяй і володарюй». Спочатку завдання розбивається на декілька підзадач меншого розміру. Потім ці завдання вирішуються за допомогою рекурсивного виклику або безпосередньо, якщо їх розмір досить малий. Нарешті, їх рішення комбінуються, і отримується рішення вихідної задачі.

Алгоритм сортування злиттям полягає в наступному:

1. Сортований масив розбивається на дві частини приблизно однакового розміру;
2. Кожна з частин сортується окремо, наприклад - тим же самим алгоритмом;
3. Два упорядкованих масиви половинного розміру з'єднуються в один.



Пірамідальне сортування

Пірамідальне сортування (англ. Heapsort, «Сортування купою») — алгоритм сортування, що працює в найгіршому, в середньому і в найкращому випадку (тобто гарантовано) за Θ(n log n) операцій при сортуванні n елементів. Кількість застосовуваної службової пам'яті не залежить від розміру масиву (тобто, O (1)).

Сортування пірамідою використовує **бінарне сортувальне дерево**. Сортувальне дерево — це таке дерево, у якого виконані наступні умови:

1. кожен лист має глибину або d, або d-1, де d — максимальна глибина дерева;
2. значення в будь-якій вершині не менші (інший варіант — не більші) за значення їх нащадків.

Плавне сортування

Плавне сортування (англ. Smoothsort) — алгоритм сортування, різновид пірамідального сортування, розроблений Е. Дейкстрою 1981 року. Як і пірамідальне сортування, в найгіршому випадку має швидкодію O(nlogn). Перевагою плавного сортування є те, що його швидкодія

наближається до O(n), якщо вхідні дані частково відсортовано, в той час як швидкодія пірамідального сортування є незмінною та не залежить від стану вхідних даних.

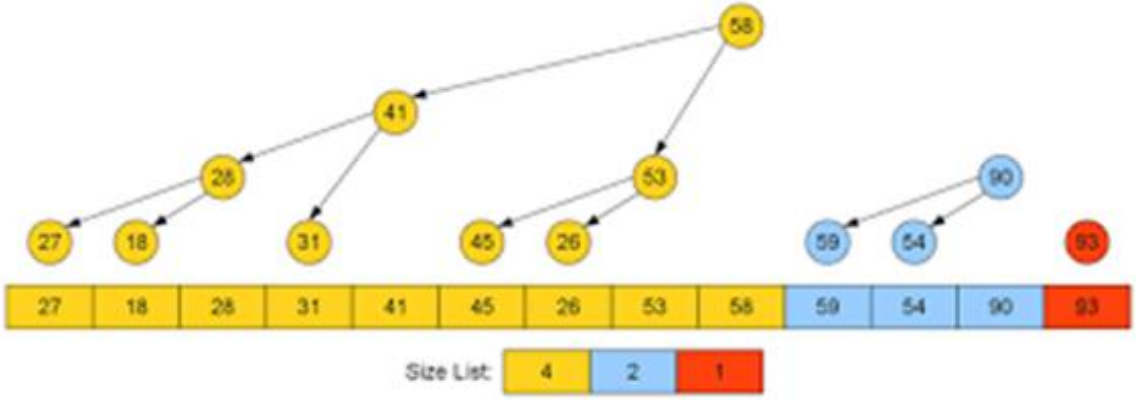
Алгоритм плавного сортування:

1. Як і в пірамідальному сортуванні, до масиву накопичується купа з даних, які потім сортуються шляхом безперервного видалення максимуму з купи.
2. На відміну від пірамідального сортування, тут використовується не двійкова купа, а спеціальна, отримана за допомогою чисел Леонардо.

$$L(n) := \begin{cases} 1 & n = 0; \\ 1 & n = 1; \\ L(n-1) + L(n-2) + 1 & n > 1. \end{cases}$$

1, 1, 3, 5 ,9, 15, 25, 41, 67, 109, 177…

3. Купа складається з послідовності куп, розміри яких дорівнюють одному з чисел Леонардо, а корені зберігаються в порядку зростання.



Завдання

1. Ознайомтесь з наступними методами сортування: сортування злиттям, пірамідальне сортування, плавне сортування.
2. Напишіть програмну реалізацію вирішення задачі на основі свого варіанту методів сортування.
3. Проведіть серію з 3-5 експериментів для оцінки продуктивності реалізацій для обраних алгоритмів, а також для алгоритму швидкого сортування. Доречно скористатися створеною програмною реалізацією з попереднього практикуму.
4. За результатами експериментів зробіть висновки.
5. Зробіть висновки, чи отримано практичне підтвердження про продуктивність сортування, що було визначено теоретично; для яких даних досліджувані алгоритми підходить якнайкраще, а у яких випадках використовувати його недоцільно.
6. Дати відповіді на контрольні питання.

Варіанти задач

Варіант 1

Король леприконів вирішив навести лад в своїх володіннях. Для цього він доручив помічникам відсортувати скарби в особистій та державній (там їх було дуже багато) скарбницях. Але на цьому він не зупинився та наказав зібрати дані про скарби всіх родин гномів та скласти впорядкований список. При збиранні таких даних з'ясувалося, що багато родин має частково однакові дані про свої скарби. Допоможіть скласти помічникам короля відповідні списки.

Варіант 2

Головнокомандувач армії вирішив провести ревізію флоту. Для цього штаб повинен скласти впорядкований за кількістю матросів список кораблів кожної армади та всього флоту. І з'ясувалося, що флот є великим. Альянс сусідніх держав теж вирішив провести ревізію, і під

час складання впорядкованого списку виявилося багато суден з однаковою кількістю матросів. Допоможіть скласти штабам відповідні списки

Варіант 3

Щоб запустити в роботу нових роботів необхідно розуміти, який впорядкований перелік пристроїв існує в невеликій частині даної мережі та в усій мережі (кількість пристроїв в мережі велика). Але роботи можуть ще пересуватися між мережами, тому склали ще один відповідний список всіх мереж. Такий список виявився великий і мав багато пристроїв з однаковою потужністю. Допоможіть скласти фахівцям відповідні списки.

Перелік алгоритмів:

1) сортування злиттям;
2) пірамідальне сортування;
3) плавне сортування;

У варіантах вказані номери алгоритмів із вищевказаного переліку.

Варіанти

1. 1), 2).
2. 1), 3).
3. 2), 3).

Склад звіту практичної роботи

- постановка задачі;
- опис методів сортування: сортування злиттям, пірамідальне сортування, плавне сортування;
- програмна реалізація вирішення задачі на основі свого варіанту методів сортування;
- 3-5 експериментів для оцінки продуктивності реалізацій для обраних алгоритмів, а також для алгоритму швидкого сортування, доречно скористатися створеною програмною реалізацією з попереднього практикуму.
- висновки за результатами експериментів;
  висновки про отримання практичного підтвердження або спростування про продуктивність сортування, що було визначено теоретично; для яких даних досліджувані алгоритми підходить якнайкраще, а у яких випадках використовувати його недоцільно;
- відповіді на контрольні питання.

Контрольні питання

1. Перерахуйте та прокоментуйте переваги та недоліки швидкого сортування.
2. Перерахуйте та прокоментуйте переваги та недоліки сортування злиттям. Які найпоширеніші модифікації цього методу сортування?
3. Перерахуйте та прокоментуйте переваги та недоліки пірамідального сортування.
4. Перерахуйте та прокоментуйте переваги та недоліки плавного сортування.
5. Зробіть порівняльну характеристику швидкого сортування та сортування злиттям.
6. Зробіть порівняльну характеристику пірамідального та плавного сортування.

Комп’ютерний практикум 8

Тема: Алгоритми сортування. Методи сортування даних з довгими ключами

Мета роботи: дослідження та порівняння алгоритмів сортування.

Завдання

1. Ознайомтесь з наступними методами сортування: порозрядне сортування, сортування кошиком.
2. Напишіть програмні реалізації методів сортування заданих за вашим варіантом та поставлено задачею.
3. Проведіть серію з 3-5 експериментів для оцінки продуктивності реалізацій для обраних алгоритмів.
4. За результатами експериментів зробіть висновки.
5. Зробіть висновки, чи отримано практичне підтвердження про продуктивність сортування, що було визначено теоретично; для яких даних досліджувані алгоритми підходить якнайкраще, а у яких випадках використовувати його недоцільно.
6. Дати відповіді на контрольні питання.

Варіанти задач

Варіант 1

Після того, як зібрали дані про скарби всіх родин гномів, король леприконів наказав скласти список сумарних скарбів родин. При збиранні таких даних з’ясувалося, що родини дуже багаті, тобто мають великі статки. Допоможіть скласти помічникам короля відповідні списки.

Варіант 2

Альянс сусідніх держав вирішив скласти впорядкований список кількості матросів флотів, і під час складання такого переліку виявилося, що кожна флотилія має дуже багато матросів. Допоможіть скласти штабам відповідний список.

Варіант 3

Корпорація мереж роботів вирішила з’ясувати яку сумарну потужність потребує кожна мережа, щоб під’єднати нові джерела живлення. Виявилося, що такі джерела будуть надавати потужність, що вимірюється великими числами. Допоможіть скласти фахівцям відповідний список.

Перелік алгоритмів:

1) LSD - угрупування;
2) MSD - угрупування;
3) сортування кошиком;

У варіантах вказані номери алгоритмів із вищевказаного переліку.

Варіанти

4. 1), 2).
5. 1), 3).
6. 2), 3).

Склад звіту практичної роботи

- постановка задачі;
- опи обраних алгоритмів;
- блок-схеми реалізацій, на яких виконано аналіз складності алгоритмів;
- результати дослідження у вигляді графіків в координатах: кількість елементів – час (елементи без повторень, без попереднього сортування); степінь відсортованості – час (при фіксованій загальній кількості елементів, без

повторень); кількість однакових елементів – час (при фіксованій загальній кількості елементів та низькій попередній відсортованості).
- висновки про порівняння продуктивності алгоритмів
- висновки про доцільність використання кожного з алгоритмів для типових вхідних даних та про відповідність результатів експериментального дослідження аналітичним оцінкам складності.
- відповіді на контрольні питання.

Контрольні питання

1. Перерахуйте та прокоментуйте переваги та недоліки порозрядного сортування.
2. Які найпоширеніші модифікації цього методу сортування? Вкажіть, на яких вхідних даних цей метод працює найкраще.
3. Перерахуйте та прокоментуйте переваги та недоліки сортування кошиком.

Індивідуальне завдання

1. Постановка задачі:

На основі карти України обрати частину території, яка включає в себе населений пункт проживання студента, населений пункт його рідних (не співпадає з попереднім) та ще декілька населених пунктів навколо них. На основі виділеної території створити базовий граф, вершини якого – населені пункти, ребра – дороги між ними. Кількість вершин повинна бути від 15 до 20, кількість ребер від 30 . Вагами ребер необхідно взяти:

a) довжину шляху між населеними пунктами в км,

b) тип дороги, ставлячи йому відповідний пріоритет.

Для кожного населеного пункту визначити його координати та кількість населення.

2. Завдання:

На основі базового графу необхідно розв'язати наступні завдання;

1. За допомогою одного з алгоритмів обходу графів обійти всі населені пункти на виділеній територій. Початком вважати населений пункт проживання студента, кінцевою точкою – пункт проживання його рідних. На довжину шляху та тип дороги не зважати.
2. За допомогою одного з алгоритмів знайти шлях від населеного пункту проживання студента до пункту проживання його рідних. На довжину шляху та тип дороги не зважати.
3. За допомогою одного з алгоритмів знайти найкоротший шлях для двох випадків обирання критерію проходження (вага a) та b)) від населеного пункту проживання студента до пункту проживання його рідних. Порівняти результати.
4. Зробити сортування населених пунктів на основі відомих алгоритмів за їх координатами із заходу на схід, та кількістю населення з більшого к меншому.
5. Для кожного попереднього пункту описати обраний алгоритм, його особливості, обґрунтувати вибір цього алгоритму та визначити його складність.
6. Для пунктів 1-4 повинне бути написане програмне забезпечення, що реалізує рішення завдання відповідного пункту.
7. Все програмне забезпечення повинно бути протестовано і надані відповідні результати.
8. Визначити час роботи створеного програмного забезпечення.

Зміст звіту.

Звіт по виконанню індивідуального завдання обов'язково повинен містити наступні пункти:

1. Титульний лист.
2. Зміст звіту
3. Постановку задачі з усіма даними.
4. Скрін-шот частини території України, що застосовується в роботі.
5. Базовий граф на основі пункту 4.
6. Розв'язок кожного поставленого вище завдання повинно бути відображено у звіті, враховуючи текст коду програмного забезпечення, таблиць і результатів тестування, висновки.

Вимоги до оформлення звіту.

1. Титульний лист містить назву закладу, кафедри, дисципліни, прізвище студента та викладача, його посаду.
2. Шрифт тексту – 14, тип – Times New Roman.
3. Платформа та мова програмування обираються студентом самостійно, але обов'язково вказуються у звіті.

Література

1. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы / М.: Вильямс, 2000 – 384с.
2. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов/М.: Мир, 1981 – 368 с.
3. Седжвик Р. Фундаментальные алгоритмы на С++ / К.: ДиаСофт, 2001 – 688с.
4. Ковалюк Т.В. Основи програмування / К.: BHV, 2005 – 384 с.
5. Ананій В. Левітін Алгоритми: введення в розробку й аналіз = Introduction to The Design and Analysis of Aigorithms. / М .: "Вильямс", 2006. - С. 275-284. - ISBN 5-8459-0987-2
6. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 6. Пірамідальна сортування / / Алгоритми: побудова й аналіз = Introduction to Algorithms / Под ред. І. В. Красикова. - 2-ге вид. / М .: Вільямс, 2005. - С. 182-188. - ISBN 5-8459-0857-4
7. https://www.toptal.com/developers/sorting-algorithms/

Додаток 1

Приклад 1 Оформлення звіту з комп'ютерного практикуму. Дослідження етапів розробки алгоритму.

Постановка задачі

В одномірному масиві з n елементів обчислити кількість елементів більше заданого числа.
Вхідні дані: число, з яким порівнюються елементи, масив чисел.
Вихідні елементи: кількість елементів, що більші за задане число.

Побудова моделі

Основні величини:

size	Цілий тип	Розмір масиву
array	Цілий тип	Масив чисел, у якому буде відбуватись обрахунок
number	Цілий тип	Число, відносно якого відбувається обрахунок
count	Цілий тип	Лічильник елементів

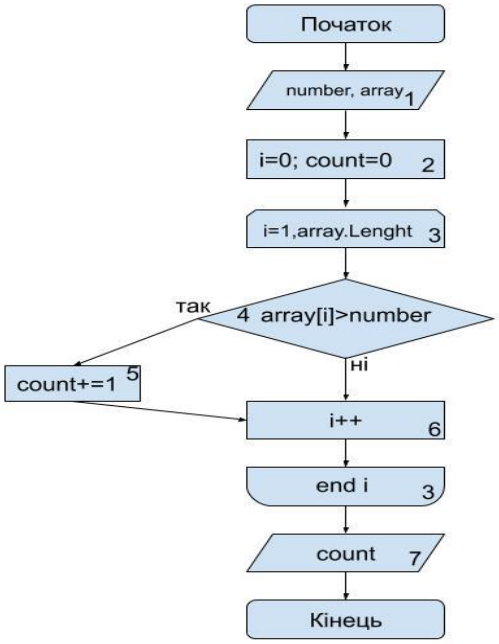
Допоміжні величини:

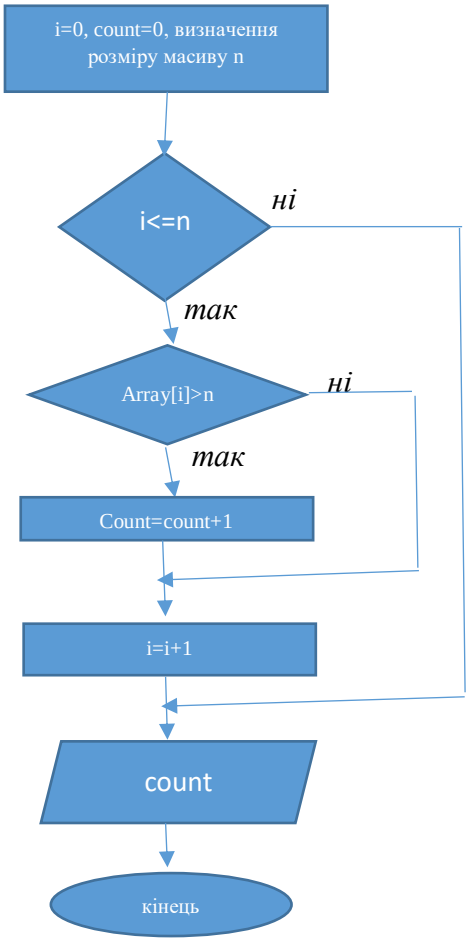
i	Цілий тип	Лічильник елементів масиву
path	Рядок	Назва файлу, у якому є елементи масиву
line	Рядок	Строка файлу

Інтерфейс користувача

Оскільки ми маємо дати змогу ввести число, відносно якого буде відбуватися рахунок, та шлях до файлу із масивом чисел, то було прийняте рішення додати простий інтерфейс користувача:
Користувач має число, відносно якого буде вестись обрахунок, і натиснути Enter.
Якщо користувач ввів не ціле число, то програма запропонує це зробити знову:
Після цього програма почне вести рахунок згідно алгоритму та виведе результат у консоль.

Розробка алгоритму

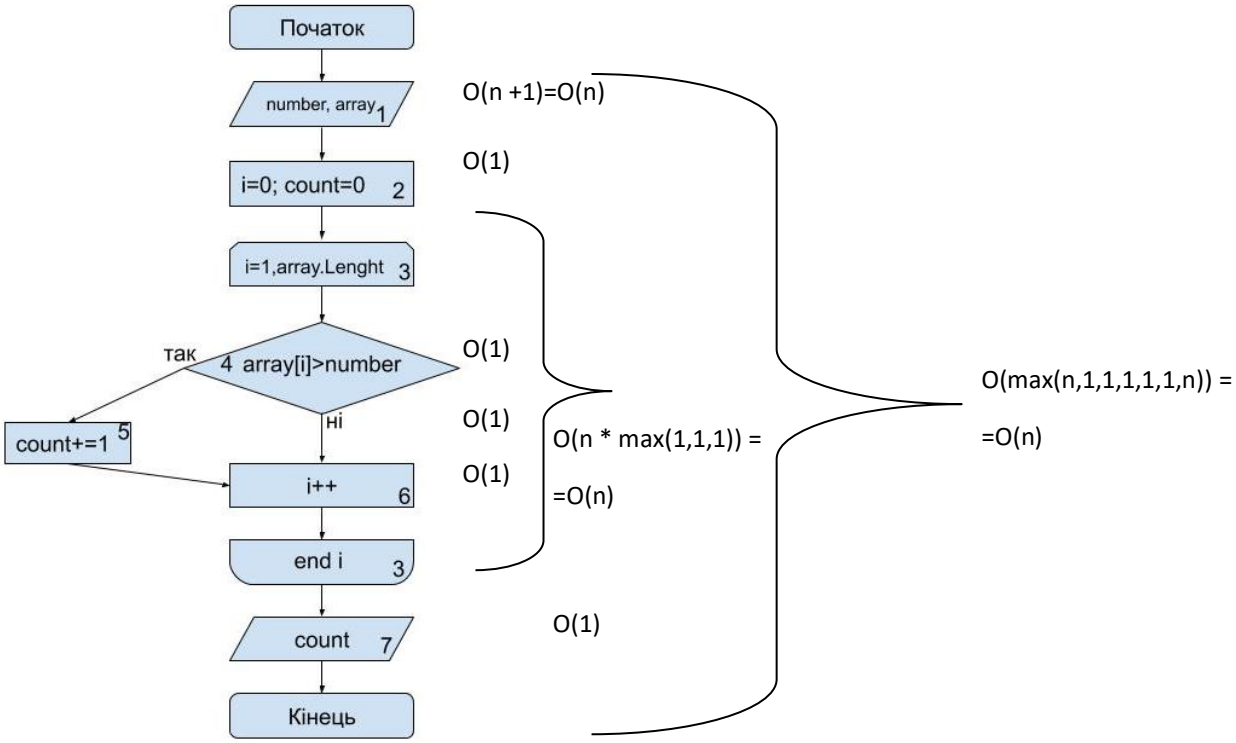




Правильність алгоритму

1. Обґрунтування правомірності кожного кроку: блок 1 відповідає за введення початкових даних – без них алгоритм не працюватиме; блок 2 відповідає за присвоєння початкових значень; блок 3 відповідає за обрахунок кількості елементів, більших за дане число із усього масиву; блок 4 – 6 забезпечують обчислення результатів; блок 7 виводить результати обчислення на екран користувача.
2. Доведення кінцевості алгоритму: кінцевість алгоритму залежить тільки від блоків 3,4, 5 і 6; початкове значення змінної “i” завжди менше ніж довжина масиву “array”, в тілі циклу ніякого впливу на змінну “i” немає, отже кількість ітерацій циклу кінцева і дорівнює array.Length - 1.

Аналіз складності алгоритму



Загальна складність алгоритму O(n).

Реалізація алгоритму та інтерфейсу користувача:

```
# Програма намагається отримати коректні вхідні дані та відкрити файл:
while True:
    try:
        number = int(input('Enter the number: ' + '\n'))
        path = input('Enter the path to the file: ' + '\n')
        file = open(path, 'r')
        break
    except:
        print('Wrong input. Try again:')

count = 0
# Починаємо читати файлу та рахувати кількість чисел, більших за дане:
try:
    for line in file:
        if int(line) > number:
            count += 1
except:
    print('Wrong data in file. Try again')
# Вивід результату:
file.close()
print(count)

buffer = input('Press Enter to exit')
```

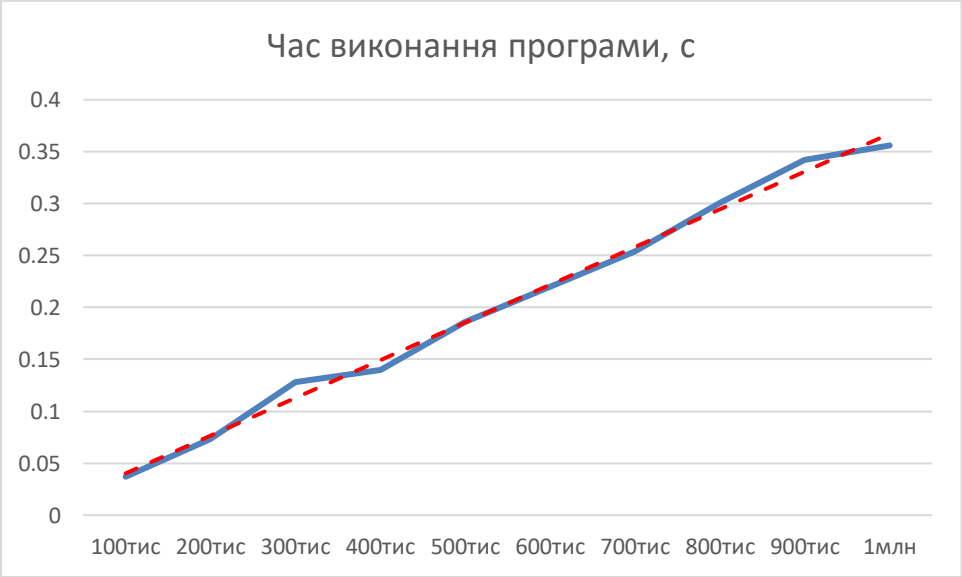
Перевірка правильності програми

Для перевірки використаємо три типи вхідних даних: детерміновані, випадкові та некоректні.

Вхідні дані		Результат	Призначення тесту
number	array		

5	{1,3,2,9,6,1,7,4,8}	Відомий	Загальна перевірка працездатності алгоритму та коректності результату.
випадкове	випадкове	Невідомий, але легко обчислити	Перевірка ефективності та обчислювальної складності алгоритму із замірюванням часу виконання програми та побудовою профілю програми.
*		Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
5	{3,5,*,3,a}	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

Перевірка обчислювальної складності програми:



Червоний пунктирний графік – теоретична складність
Синій графік – реальна складність
Перевірка обчислювальної складності програми підтвердила аналітичну оцінку O(n).

Додаток 2
Приклад 2. Оформлення звіту з комп'ютерного практикуму. Хешування.

Завдання
1. Ознайомтесь з реалізаціями схем закритого хешування (дві методики вирішення колізій для закритого хешування)
2. Напишіть програмні реалізації для двох методик вирішення колізій для закритого хешування.
3. Проаналізуйте їх ефективність та складність і вкажіть переваги та недоліки кожної з реалізацій.

Хешування — перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями, або функціями згортання, а їхні результати називають хешем, хеш-кодом, хеш-сумою, або дайджестом повідомлення (англ. message digest).
При **закритому хешуванні** в таблиці сегментів зберігаються безпосередньо елементи словника, а не заголовки списків. Тому в кожному сегменті може зберігатися тільки один елемент словника. При закритому хешуванні застосовується методика повторного хешування. Якщо ми спробуємо помістити елемент x в сегмент з номером h (x), який вже зайнятий іншим елементом (така ситуація називається колізією), то відповідно до методики повторного хешування вибирається послідовність інших номерів сегментів h1 (x), h2 (x),. .., куди можна помістити елемент x. Кожне з цих місць розташування послідовно перевіряється, поки не буде знайдено вільне. Якщо вільних сегментів немає, то, отже, таблиця заповнена і елемент x вставити не можна.

Під час виконання даної роботи, були обрані наступні методи дослідження відкритого хешування:
- Метод подвійного хешування;
- Метод лінійного зондування;

Метод подвійного хешування

Згідно з даним методом, якщо для елемента A адреса п () = h (A), вказує на вже зайняту комірку, тобто в разі виникнення колізії, необхідно обчислити значення функції n1 = h1(A) і перевірити зайнятість комірки за адресою n1. Якщо і вона зайнята, то обчислюється значення h2(A). І так до тих пір, поки або не буде знайдена вільна комірка, або чергове значення hi(A) не співпаде з h(A). В останньому випадку вважається, що таблиця ідентифікаторів заповнена і місця в ній більше немає видається повідомлення про помилку розміщення ідентифікатора в таблиці.
Подвійне хешування аналогічно лінійному випробуванню, за винятком того, що тут використовується друга хеш-функція - для визначення кроку пошуку, що використовується після кожної колізії. Крок пошуку повинен бути ненульовим, а розмір таблиці і крок пошуку повинні бути взаємно простими числами. Функція remove для лінійного випробування не працює з подвійним хешування, оскільки будь-який ключ може бути присутнім в багатьох різних послідовностях проб.

Блок-схема:

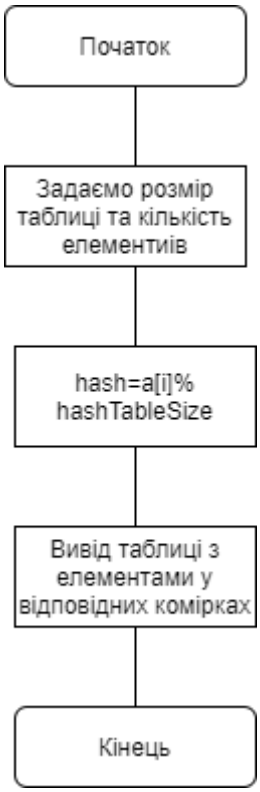
Складність алгоритму: лінійна складність пошуку даної хеш-функції методом ділення по модулю - O(n).

Метод лінійного зондування

Лінійне зондування є компонентом схем відкритої адресації для використання в хеш-таблицях для вирішення словникових завдань. У словникової задачі структура даних повинна працювати з набором пар ключ-значення і повинна забезпечувати можливість вставки і видалення пар, а також пошук значення, асоційованого з ключем.

Комірки хеш-таблиці послідовно переглядаються з деяким фіксованим інтервалом k між комірками (зазвичай,), тобто i-й елемент послідовності — це комірка з номером (hash(x) + ik) mod N. Для того, щоб всі комірки виявились перебраними по одному разу, необхідно, щоб k було взаємно-простим з розміром геш-таблиці.

Блок-схема:



Складність алгоритму: лінійна складність пошуку даної хеш-функції методом ділення по модулю - O(n).

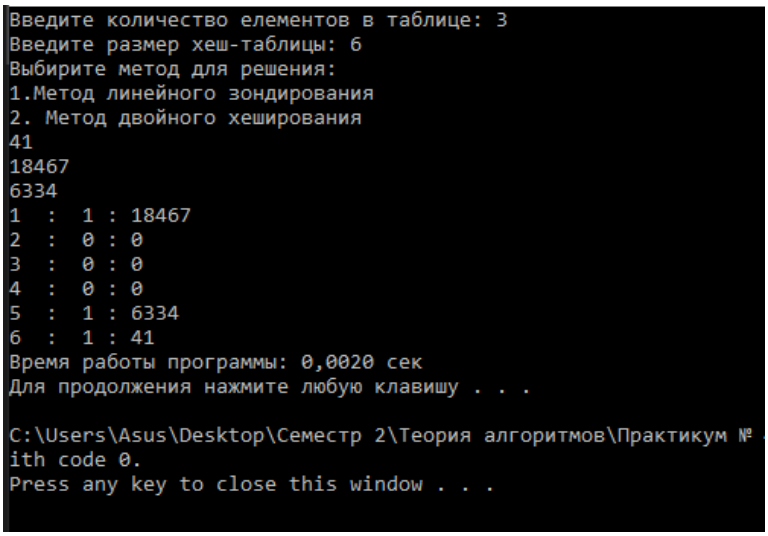
Реалізація програми на мові C++:

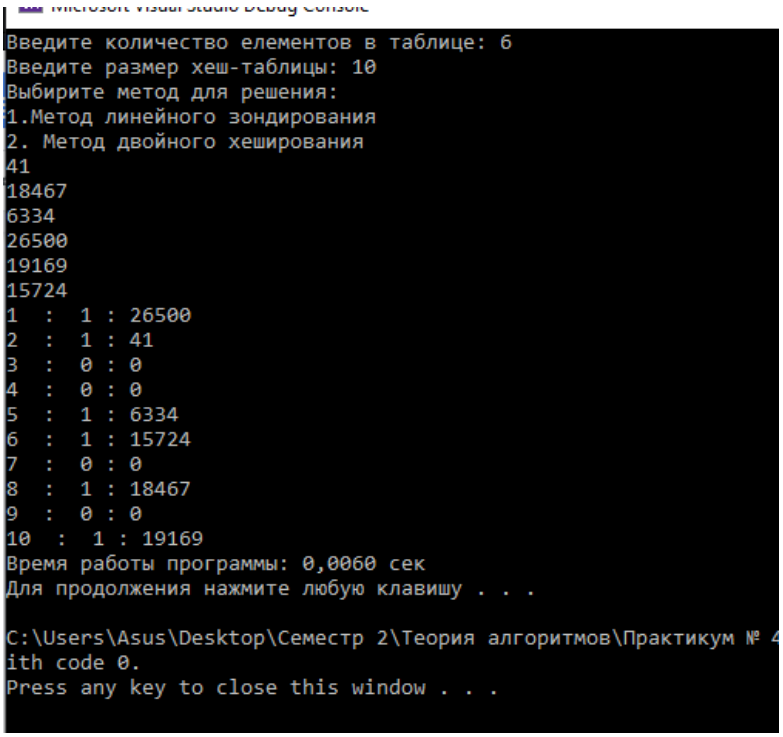
```
#include <iostream>
#include <conio.h>
#include <time.h>
#include <fstream>
using namespace std;

typedef int T;
typedef int index;
int hashTableSize;
int flag;
T* hashTable;
bool* used;

index lineZond(T data);
index doubleHash(T data);
void insertData(T data);
void deleteData(T data);
bool findData(T data);
int dist(index a, index b);

int main() {
   setlocale(LC_ALL, "rus");
   clock_t start, end;
   int i, * a, maxnum;
   cout << "Введите количество елементов в таблице: ";
   cin >> maxnum;
   cout << "Введите размер хеш-таблицы: ";
   cin >> hashTableSize;
```

```
    cout << "Выбирите метод для решения:\n1.Метод линейного зондирования\n2. Метод двойного хеширования" << endl;
    do { flag = _getch(); } while (flag != 49 && flag != 50);
    a = new int[maxnum];
    hashTable = new T[hashTableSize];
    used = new bool[hashTableSize];
    for (i = 0; i < hashTableSize; i++) {
        hashTable[i] = 0;
        used[i] = false;
    }
    for (i = 0; i < maxnum; i++)
        a[i] = rand();
    for (i = 0; i < maxnum; i++)
        insertData(a[i]);
    start = clock();
    for (i = 0; i < maxnum; i++) {
        cout << a[i];
        if (i < maxnum - 1) cout << '\n';
    }
    end = clock();
    cout << endl;
    for (i = 0; i < hashTableSize; i++) {
        cout << i + 1 << "  :  " << used[i] << " : " << hashTable[i] << endl;
    }
    printf("Время работы программы: %.4f сек\n", ((double)end - start) / ((double)CLOCKS_PER_SEC));
    system("pause");
    return 0;
}
//Метод лінійного зондування
index lineZond(T data) {
    return (data % hashTableSize);
}
//Метод подвійного хешування
index doubleHash(T data) {
    return (1 + (data) % (hashTableSize - 2));
}

void insertData(T data) {
    index bucket;
    if (flag == 49)bucket = lineZond(data);
    else bucket = doubleHash(data);
    while (used[bucket] && hashTable[bucket] != data)
        bucket = (bucket + 1) % hashTableSize;
    if (!used[bucket]) {
        used[bucket] = true;
        hashTable[bucket] = data;
    }
}
```

Результати роботи програми:

Метод лінійного зондування:



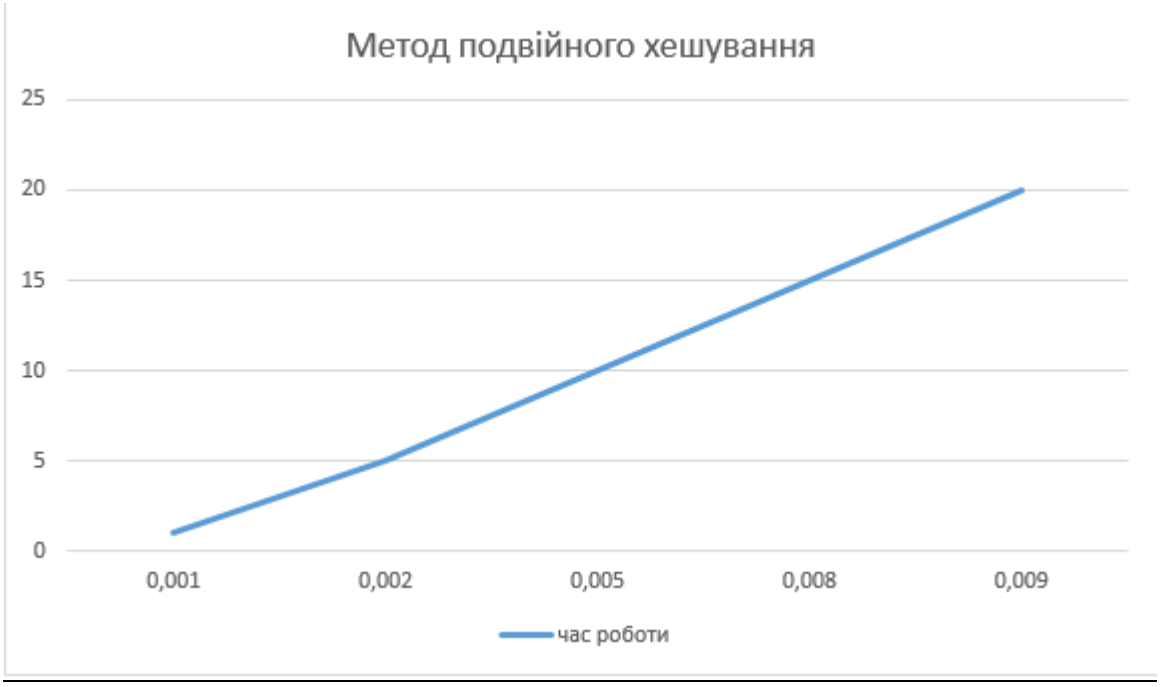


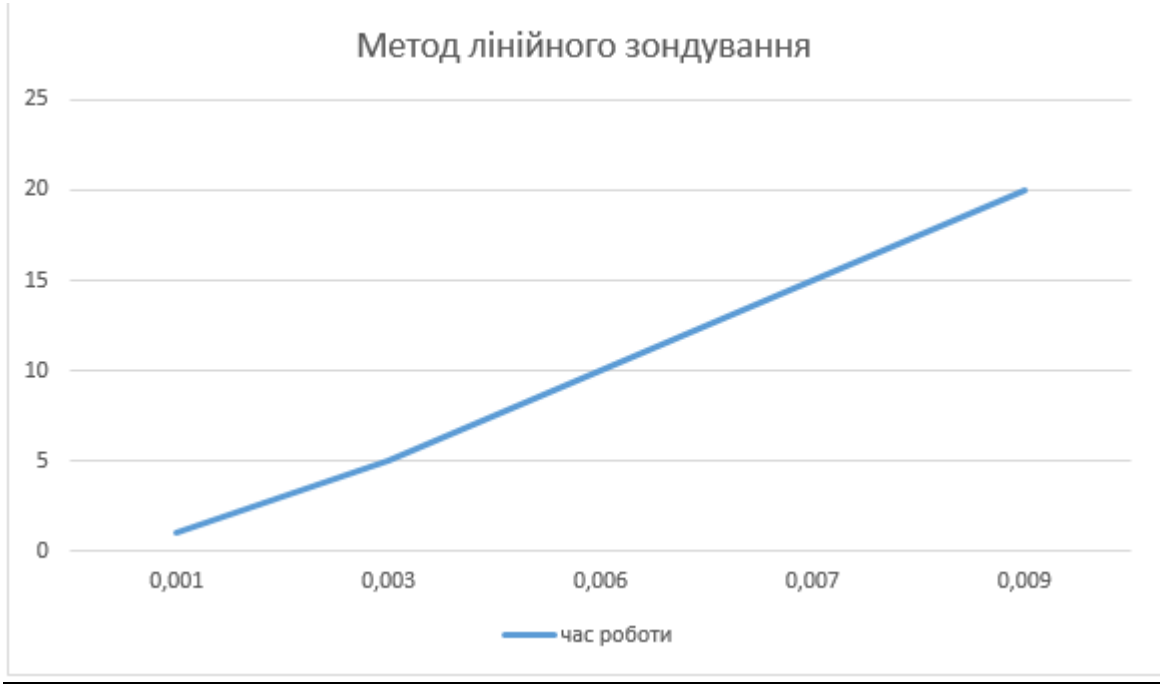
Метод подвійного хешування:

```
Введите количество елементов в таблице: 3
Введите размер хеш-таблицы: 7
Выбирите метод для решения:
1.Метод линейного зондирования
2. Метод двойного хеширования
41
18467
6334
1  :  0 : 0
2  :  0 : 0
3  :  1 : 41
4  :  1 : 18467
5  :  0 : 0
6  :  1 : 6334
7  :  0 : 0
Время работы программы: 0,0030 сек
Для продолжения нажмите любую клавишу . . .

C:\Users\Asus\Desktop\Семестр 2\Теория алгор
ith code 0.
```

```
Введите количество елементов в таблице: 10
Введите размер хеш-таблицы: 10
Выбирите метод для решения:
1.Метод линейного зондирования
2. Метод двойного хеширования
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
1  :  1 : 26962
2  :  1 : 24464
3  :  1 : 41
4  :  1 : 19169
5  :  1 : 18467
6  :  1 : 26500
7  :  1 : 15724
8  :  1 : 6334
9  :  1 : 11478
10  :  1 : 29358
Время работы программы: 0,0050 сек
Для продолжения нажмите любую клавишу
```

Графіки ефективності алгоритмів:
(залежність кількості елементів в таблиці від часу роботи)





Практичне застосування:

1. Протокол для створення ефективної інформаційної конструкції. Головне завдання - структурування інформації в хеш-таблиці. Така таблична структура робить можливим додавати / видаляти, знаходити потрібні дані з неймовірною швидкістю.

2. Криптографічний алгоритм. Впроваджується в якості захисту від несанкціонованого доступу (злому). Перевіряє систему на предмет спотворення інформації в момент передачі файлів по мережі. При таких обставинах «правильний» хеш має вільний доступ, а ключ, отриманого документа, обчислюється за допомогою різноманітних софтів (програм). Приклад - SHA-1, SHA-2, MD5, що відрізняються стабільністю і складним рівнянням пошуку колізій.
3. Протокол для аудиту цілісності інформації. В процесі передачі документів здійснюється обчислення хеш-коду. Результат передається разом з документами. Прийом інформації супроводжується повторним обчисленням хеш-коду з подальшим порівнянням, отриманим значенням. Розбіжність - помилка. Цей алгоритм має високі швидкісними

показниками при обчисленні, але відрізняється нестабільністю і малим значенням хеш-функції. Приклад - протокол CRC-32, який має тільки 232 значення хеш, що розрізняються між собою.

Висновок:

Головною превагою лінійного зондування є проста реалізація. Основний недолік – кластеризація. Також подвійне хешування суттєво зменшує ймовірність формування кластерів і зменшує кількість колізій. На практиці схема подвійного хешування виявляється найбільш вдалою з усіх альтернативних схем з відкритою адресацією. Очевидно, що цей варіант буде давати значно кращий розподіл і незалежні один від одного ланцюжки. Однак, він трохи повільніше через введення додаткової функції.

Додаток 3

Приклад 3. Оформлення звіту з комп'ютерного практикуму. Бінарні дерева.

Постановка задачі. Ми маємо таке завдання: Є певна кількість приладів. Кожний прилад має свою унікальну потужність. Необхідно швидко знайти той прилад, що точно відповідає заданій потужності заряду робота. Потужність задається з клавіатури. Від кожного приладу є лише два шляхи для робота, але він точно знає, де потужність буде збільшуватися та зменшуватися в порівнянні з поточним. Якщо для робота не буде знайдено відповідний пристрій, то для його порятунку треба терміново додати такий пристрій. Для обох випадків побудувати шлях знаходження пристрою. Якщо пристрій зламався, то він вже не має можливості допомогти роботу, тому видаляється з мережи приладів та надається повідомлення про це.

Отже ми маємо реалізувати такий розподіл приладів з відомою потужністю, який допомагатиме шукати потрібний елемент швидко та ефективно. З поставленої задачі можемо зробити висновок, що прилади мають розміщатись в залежності від спадання та зростання значень їхніх потужностей.

Ми маємо такі вхідні дані: кількість та список приладів (їхніх потужностей), потужність приладу, який ми маємо знайти, потужність приладу, який маємо видалити.

Для реалізації ми використовували двійкове дерево пошуку, що, на нашу думку, було простим для розуміння та програмної реалізації, тому найбільше підходило для вирішення цієї задачі на той момент. Цього разу використовуємо також збалансоване дерево АВЛ та червоно-чорне дерево, що мають бути ефективнішими. Перевіримо і порівняємо їхню роботу.

Двійкове дерево

Двійкове дерево пошуку – структура, що будується за певними правилами:

- дерево має один «корінь» (кореневий вузол), від якого йде розподіл елементів за певним алгоритмом, решта елементів знаходяться у «вузлах»;
- вузол, після якого є хоча б один елемент, є батьківським для того елементу (батько);
- вузол, який розміщується після якогось елементу, є дочірнім для того елементом (дитиною).
- у кожного вузла не більше двох дітей;
- будь-яке значення менше значення вузла стає лівою дитиною або дитиною лівої дитини;
- будь-яке значення більше або рівне значенню вузла стає правою дитиною або дитиною правої дитини;
- всі піддерева кожного вузла є двійковими деревами.

Методи додавання та пошуку елементів є дуже подібними, оскільки в обох випадках відбувається однаковий перебір елементів. Складність їхня в кращому випадку дорівнює O(log(n)), в гіршому - O(n), де n – кількість елементів. Це пояснюється тим, що під час виконання даних методів, через специфічні правила, під час кожної операції відбувається відкидання половини елементів, що залишились для перегляду. Оскільки не можна одночасно піти і праворуч, і ліворуч, даний етап відкидання є обов'язковим, що забезпечує логарифмічну складність.

Проте складність O(n) є більш імовірною, оскільки ми використовуємо цикли для перебору елементів, а також, двійкове дерево не є збалансованим, тому в гіршому випадку елементи можуть бути розташовані в порядку, схожому до звичайного списку чи масиву.

Збалансоване дерево пошуку – АВЛ-дерево

АВЛ-дерево – це збалансоване по висоті двійкове дерево пошуку: для кожної його вершини висота її двох піддерев відрізняється не більше ніж на 1.

Щодо АВЛ-дерева балансуванням вершини називається операція, яка у разі різниці висот лівого і правого піддерев = 2, змінює зв'язок предок-нащадок в піддереві даної вершини так, що різниця стає <= 1, інакше нічого не змінює. Зазначений результат здійснюється обертаннями піддерева даної вершини.

Використовуються всього 4 типи обертань:

1. Мале ліве обертання;

2. Мале праве обертання;
3. Велике ліве обертання;
4. Велике праве обертання.

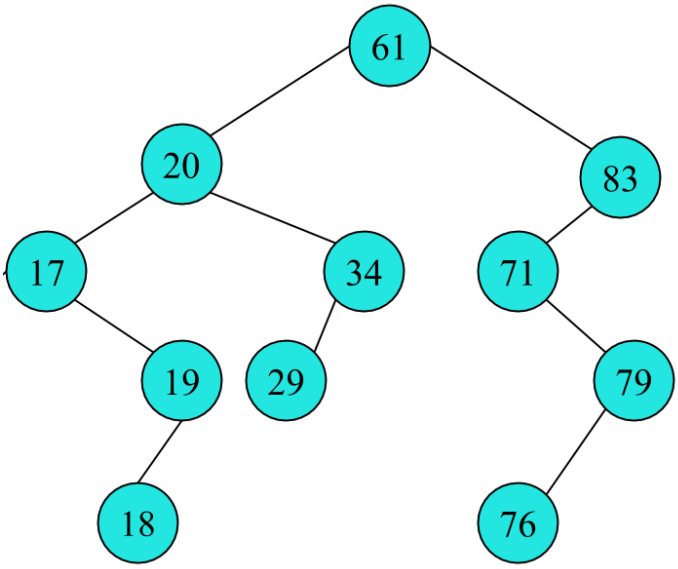
У кожному випадку досить просто довести те, що операція приводить до потрібного результату і що повна висота зменшується не більше ніж на 1 і не може збільшитися.
Через умови балансування висота дерева O (log (N)), де N-кількість вершин, тому додавання елемента вимагає O (log (N)) операцій.

Червоно-чорне дерево

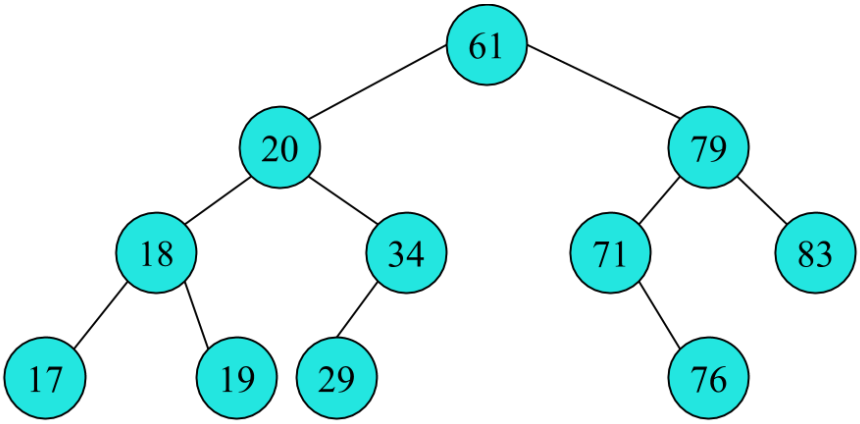
Червоно-чорне дерево — різновид самозбалансованого бінарного дерева пошуку, вершини якого мають додаткові властивості (RB-властивості), зокрема «колір» (червоний або чорний). Ці біти кольору використовуються для забезпечення того, щоб дерево залишалося приблизно збалансованим при виконанні операцій вставки та видалення.
Червоно-чорні дерева — різновид збалансованих дерев, в яких за допомогою спеціальних трансформацій гарантується, що висота дерева h не буде перевищувати O (log n). Зважаючи на те, що час виконання основних операцій на бінарних деревах (пошук, видалення, додавання елементу) є O(h), ці структури даних на практиці є набагато ефективнішими, аніж звичайні бінарні дерева пошуку. Бінарне дерево називається червоно-чорним, якщо воно має такі властивості:
1. кожна вершина або червона, або чорна
2. корінь дерева — чорний
3. кожний лист (NIL) — чорний
4. якщо вершина червона, обидві її дочірні вершини чорні (інакше, батько червоної вершини — чорний)
5. усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин

Приклад побудови та балансування двійкового дерева

Для початку отримуємо на вхід незбалансоване дерево:



Далі, отримуємо збалансоване дерево, яке має такий вигляд:



Аналіз ситуації після збалансування

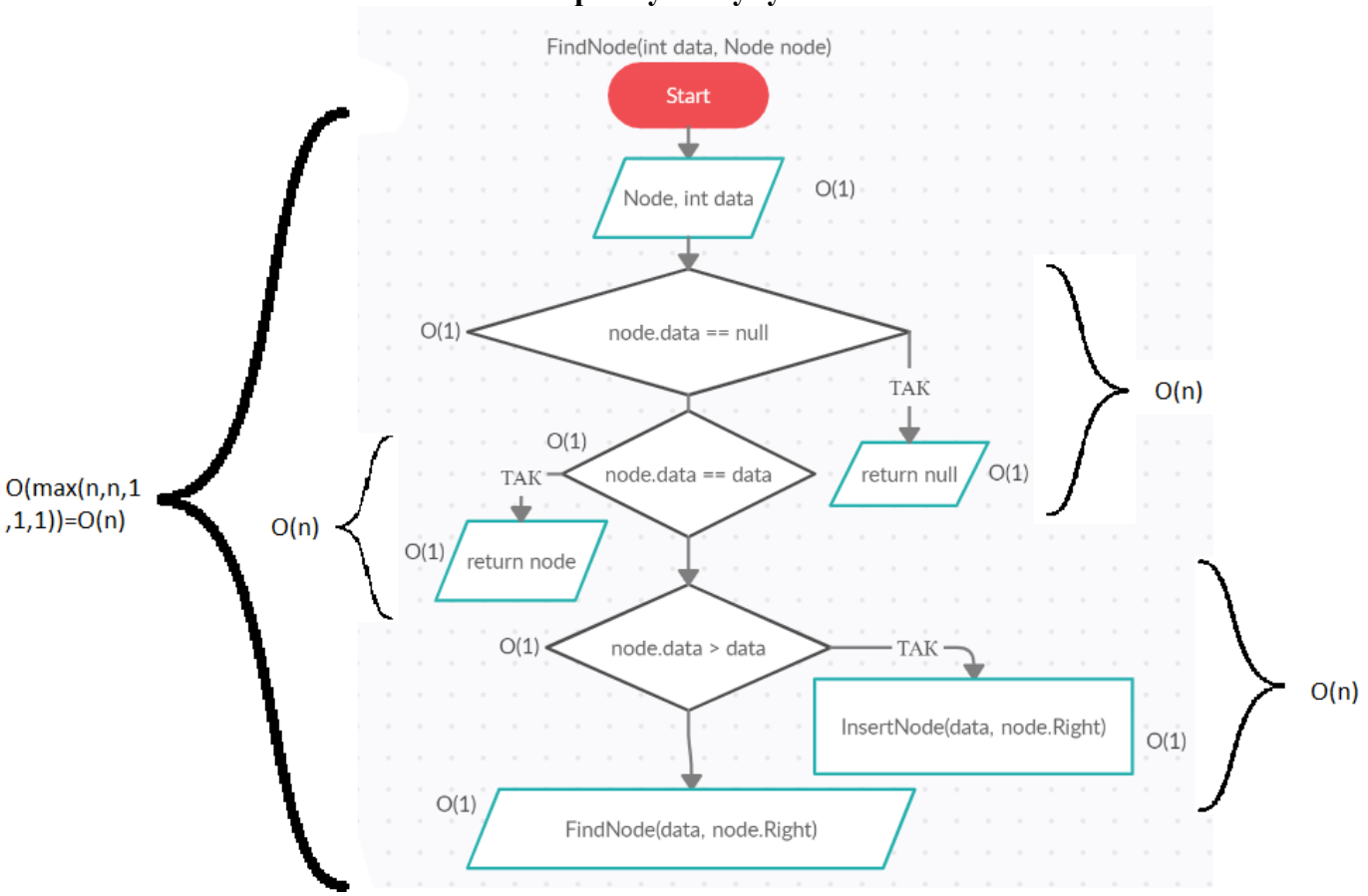
Оскільки ми маємо вирішити конкретну задачу: пошук приладів з потрібною потужністю, що здійснюється роботом, розглянемо попередні дії саме для цієї задачі.

Уявимо, що прилади знаходяться в певних точках простору, наприклад, подібного до лабіринту, системи коридорів тощо, по якому можна побудувати шляхи до приладів. Нехай є початкова точка, звідки робот почне свій пошук. Далі він зможе піти по одному з коридорів – праворуч або ліворуч – в залежності від потужності приладу, до якого він дійшов.

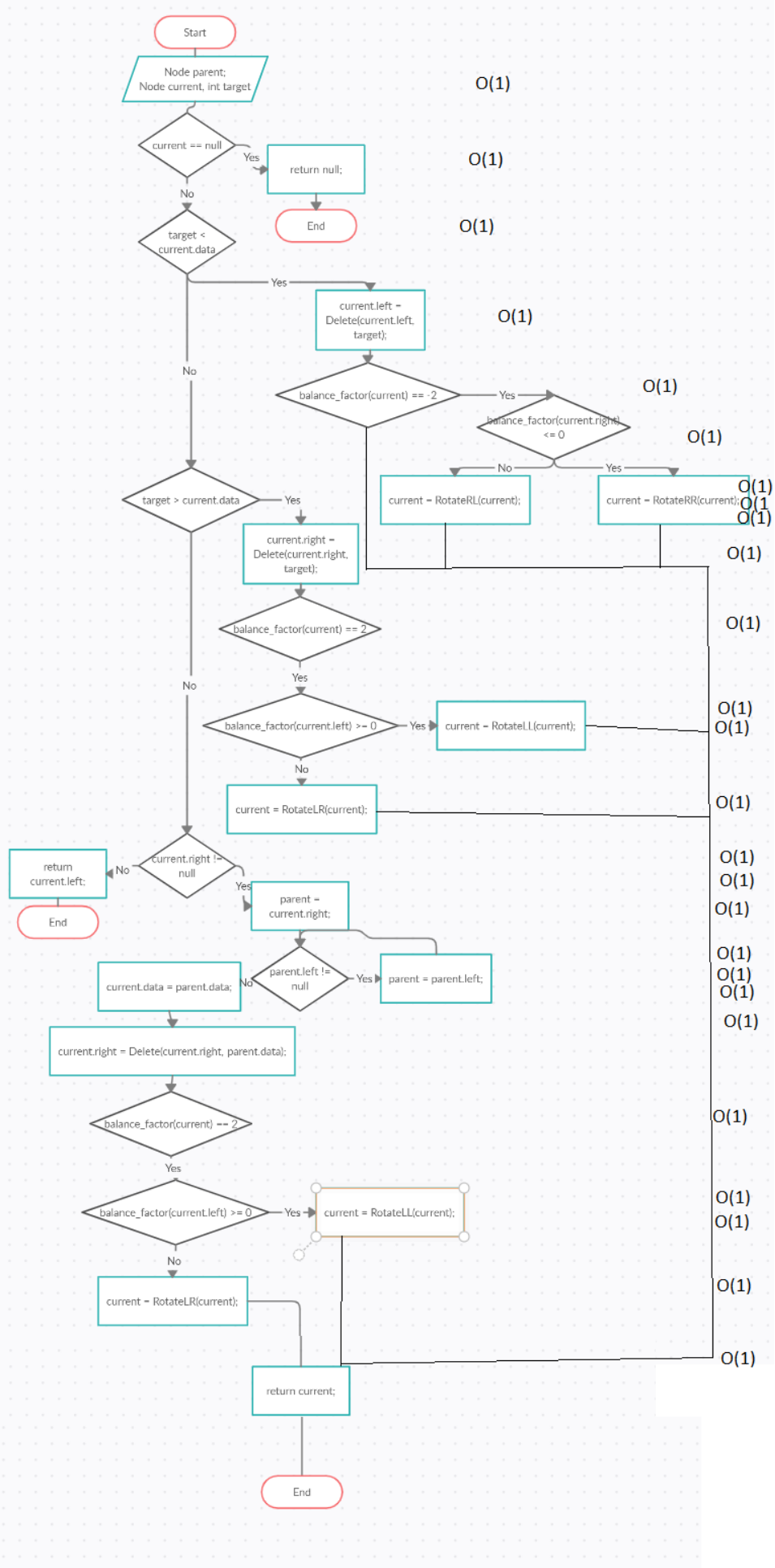
Перший граф (незбалансований) – це початковий шлях, в нашому випадку система коридорів. За умовою задачі нас не влаштовує те, що він незбалансований, тобто пошук може бути надто довгим. Тому аналітично можна його збалансувати, порівняти з початковим, та в залежності від отриманих результатів перебудувати систему коридорів.

Оскільки розглядається практична задача, то якщо незбалансований шлях є оптимальним для пошуку, було б логічніше залишити його, адже перебудовувати всю систему коридорів – це додаткова витрата робочих, часових та фінансових ресурсів, чого варто уникати. Проте якщо початковий шлях є поганим для пошуку, ми маємо перебудувати систему коридорів для робота, щоби в майбутньому у нього був оптимізований шлях.

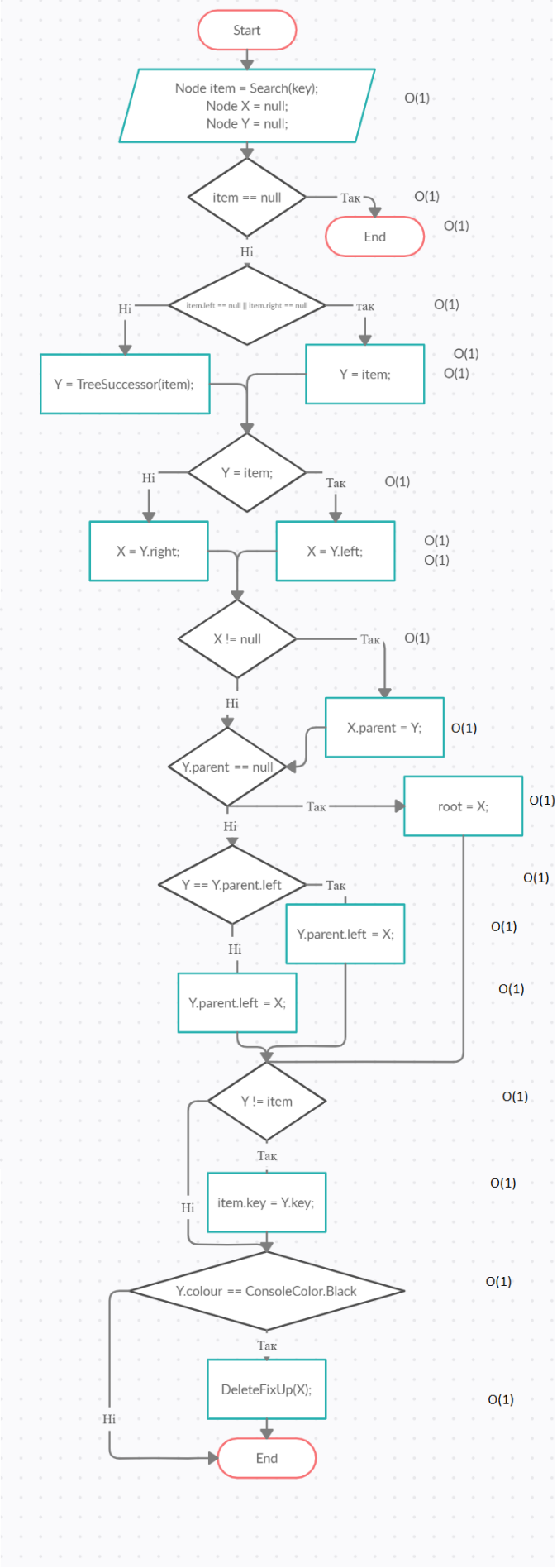
Блок-схема алгоритму пошуку елементів



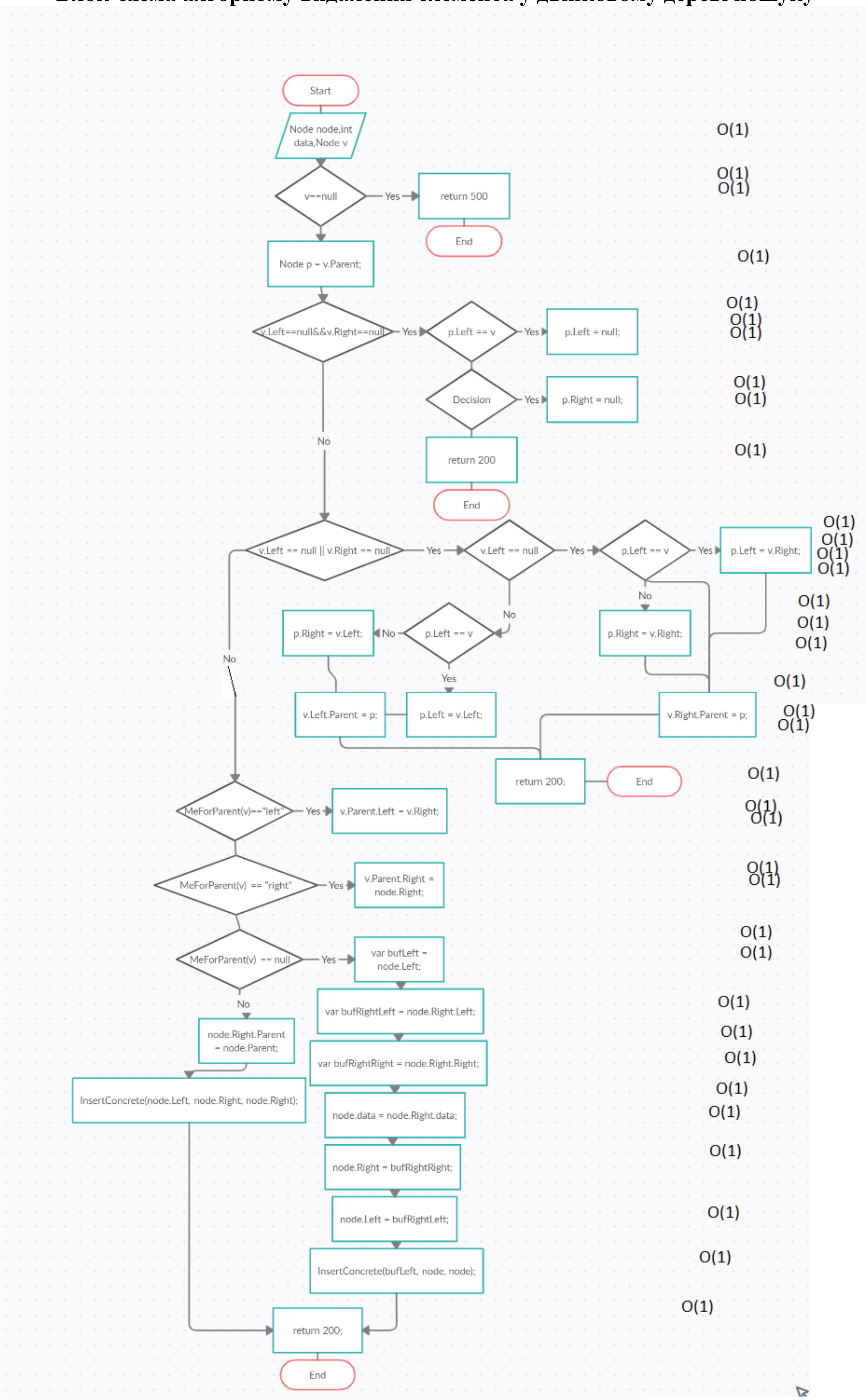
Блок-схема алгоритму видалення елемента в АВЛ-дереві пошуку



Блок-схема алгоритму видалення елемента у червоно-чорному дереві пошуку



Блок-схема алгоритму видалення елемента у двійковому дереві пошуку



Оцінка правильності алгоритмів

Коли програма починає свою роботу, відбувається створення потрібних класів для реалізації дерева. Користувач отримує повідомлення, де вказано, що потрібно ввести кількість вершин в дереві. Коли дана операція виконана, здійснюється ініціалізація допоміжного масиву та змінних, виконується побудова дерева з заданими елементами (значення вузлів генерується з файлу, доступ до якого вказаний в самій програмі). Далі відбувається збалансування дерева за допомогою алгоритму балансування.

Користувач отримує повідомлення про вибір пошуку, який він хоче здійснити: пошук багатьох елементів, заданих масивом, чи пошук конкретного елемента. В залежності від вибору користувача, здійснюється пошук (вдалий та ні), коли він завершується, користувач отримує повідомлення про час виконання операцій.

Далі користувач обирає, яку вершину він хоче видалити, відбувається видалення вершини та відлік часу його виконання, вивід повідомлення про час виконання.

Також користувач має можливість переглянути побудоване дерево, після чого програма завершує свою роботу.

Дані програми приймають вхідні дані – кількість елементів, які будуть розподілені в дереві. Надалі у відповідності з нашим завданням програма починає розподіл елементів та збалансування дерева, а потім виконує пошук елементів. Коли всі елементи розподілені, відлік часу починається, здійснюється пошук елементів, відлік часу зупиняється та виводиться повідомлення зі вказаною тривалістю роботи алгоритмів. Також здійснюється видалення заданого елемента та відлік часу виконання видалення. Далі програми завершують свою роботу.

Оцінка складності алгоритмів

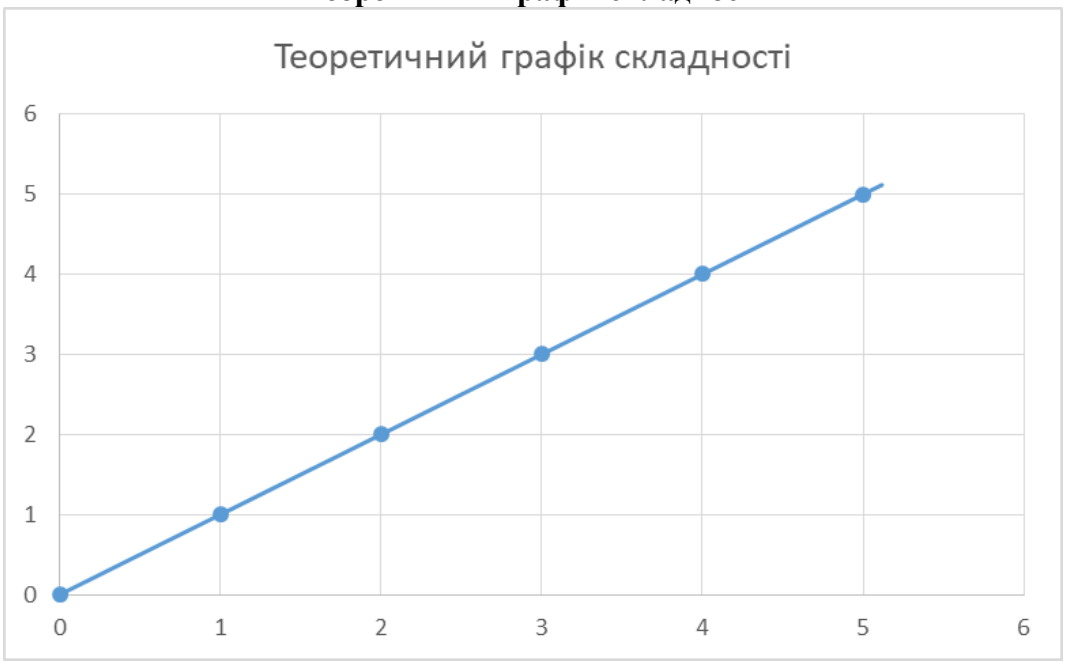
Як видно на наших блок-схемах, алгоритм пошуку є лінійним за підрахунком, тобто складність алгоритму – O(n) (для найгіршого випадку). Проте ми балансуємо дерево, тому випадок не може бути найгіршим та складність буде між O(log(n)) та O(1). Аналітична складність алгоритму видалення елемента - O(1).

Перевірка правильності програм

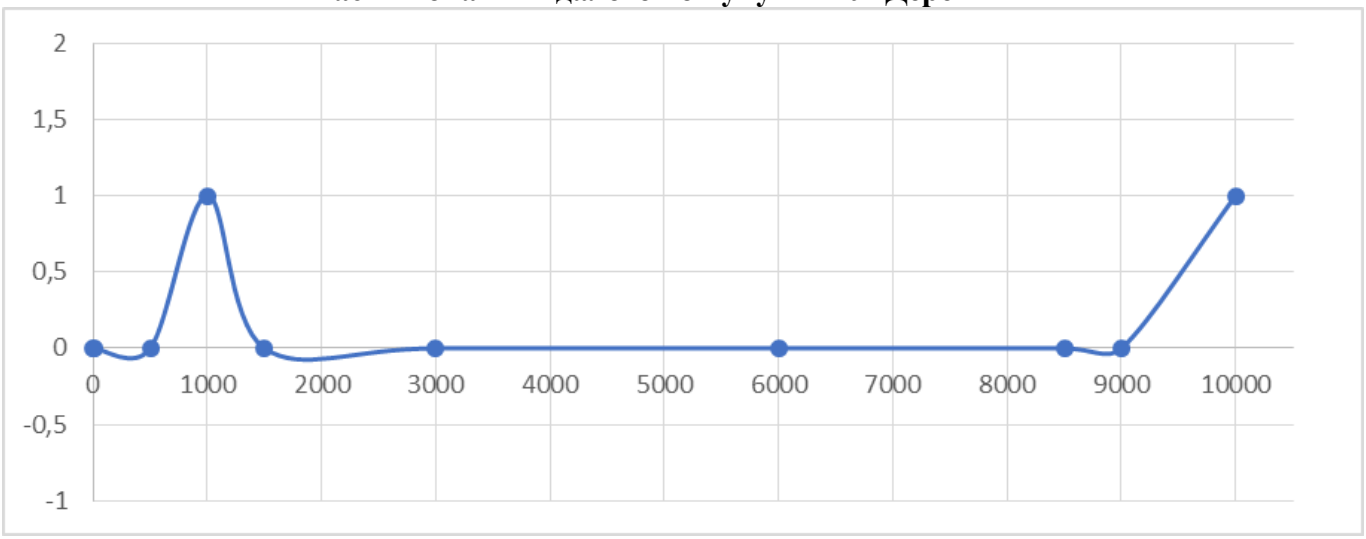
Вхідні дані countiterations, findNumber, shipDel	**Результат**	**Призначення тесту**
11, 71, 71	Відомий	Перевірка правильності роботи програми у випадку введення коректних даних
*	Відомий	Перевірка правильності роботи програми у випадку введення некоректних даних для кількості елементів
11, *	Відомий	Перевірка правильності роботи програми у випадку введення некоректних даних для пошуку елементів
11, 71, *	Відомий	Перевірка правильності роботи програми у випадку введення некоректних даних для видалення елементів
випадкове	Відомий	Перевірка правильності роботи програми у випадку автоматичної генерації випадкових чисел

Перевірка часу виконання роботи програм

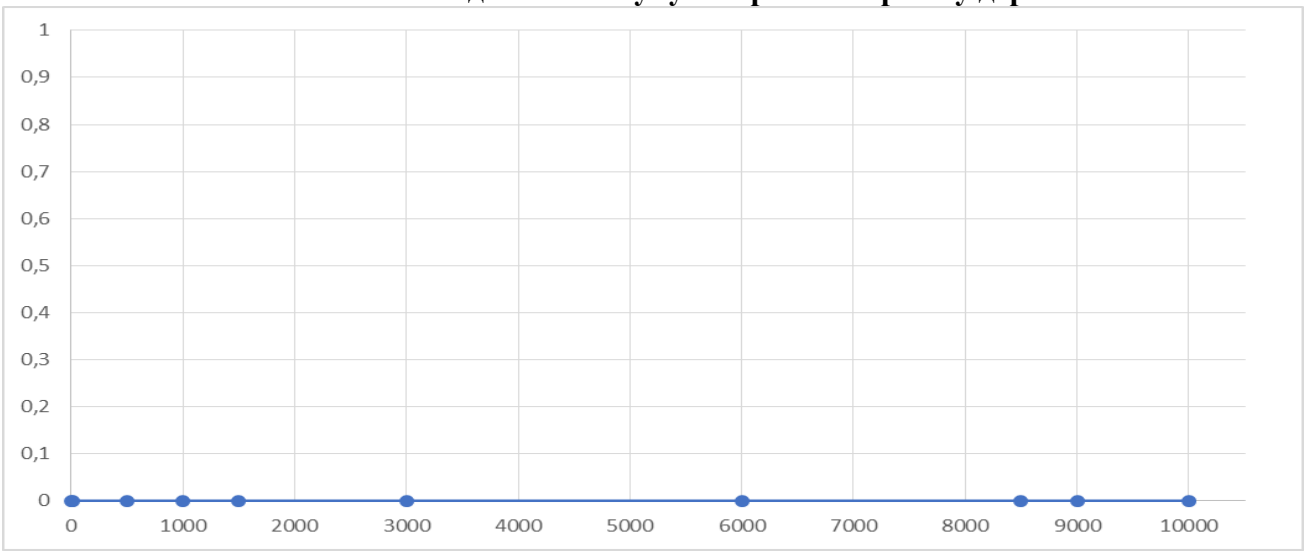
Теоретичний графік складності



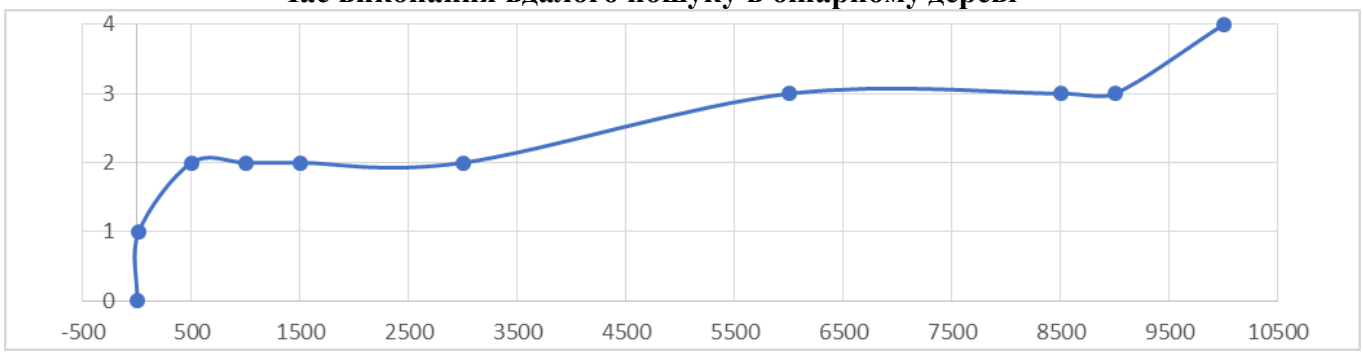
Час виконання вдалого пошуку в АВЛ-Дереві



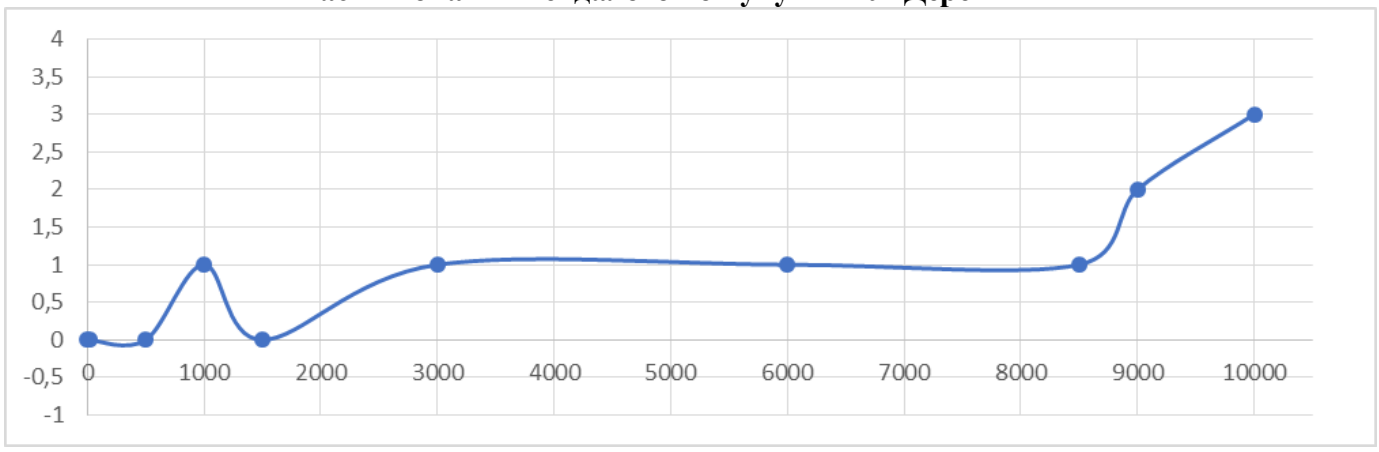
Час виконання вдалого пошуку в червоно-чорному дереві



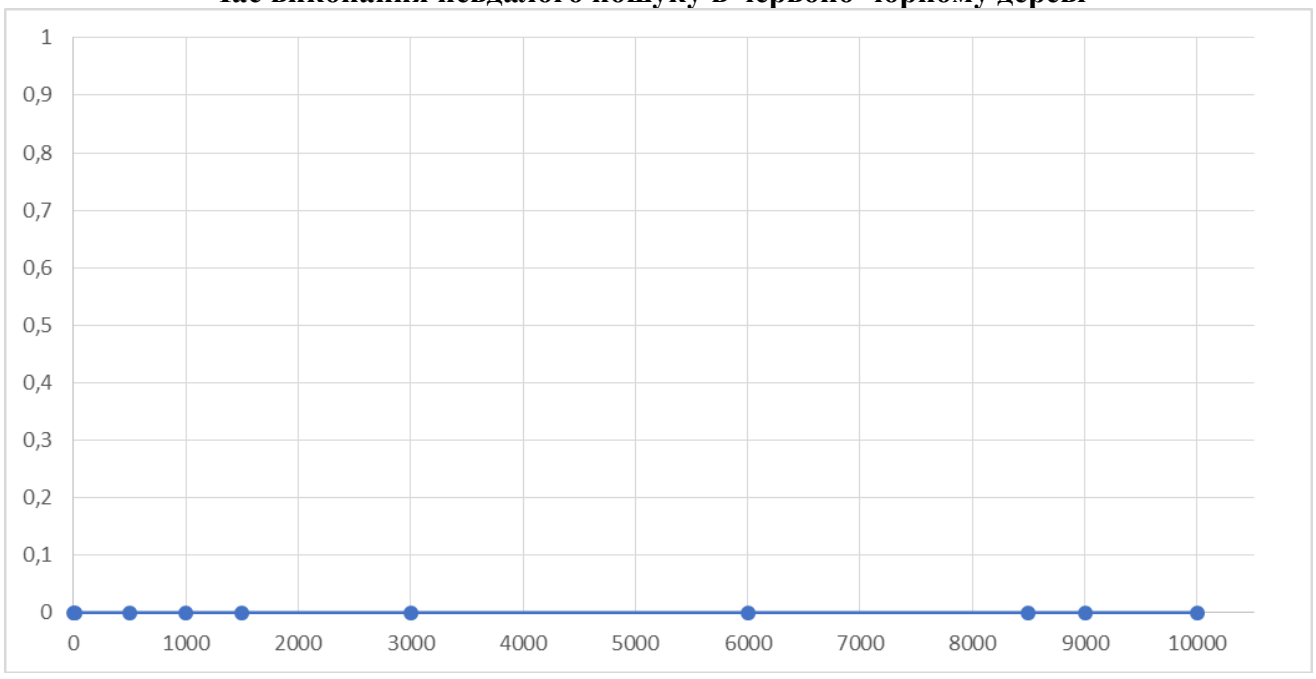
Час виконання вдалого пошуку в бінарному дереві



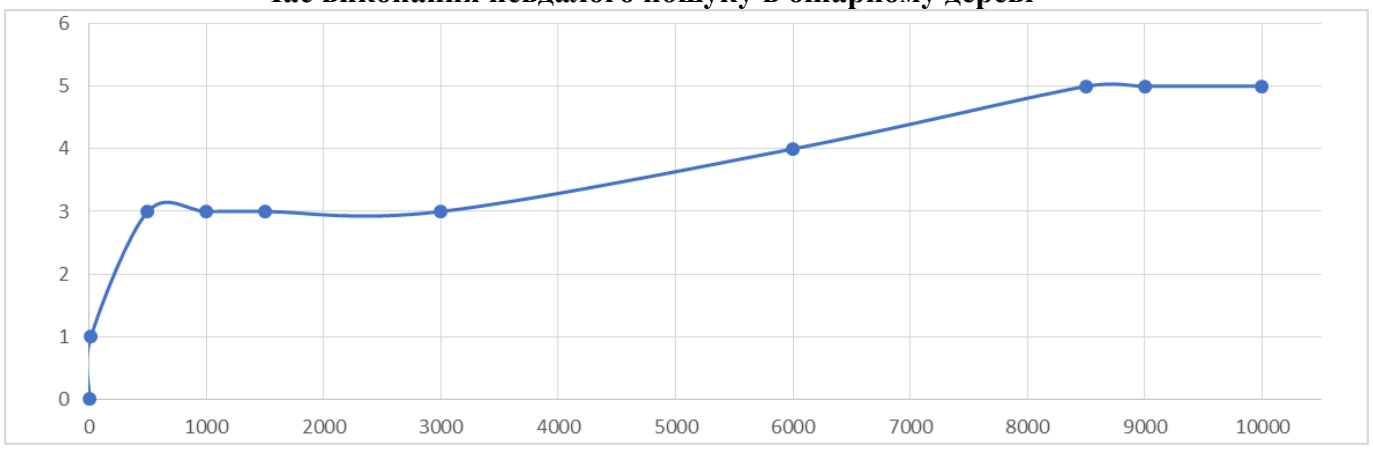
Час виконання невдалого пошуку в АВЛ-Дереві



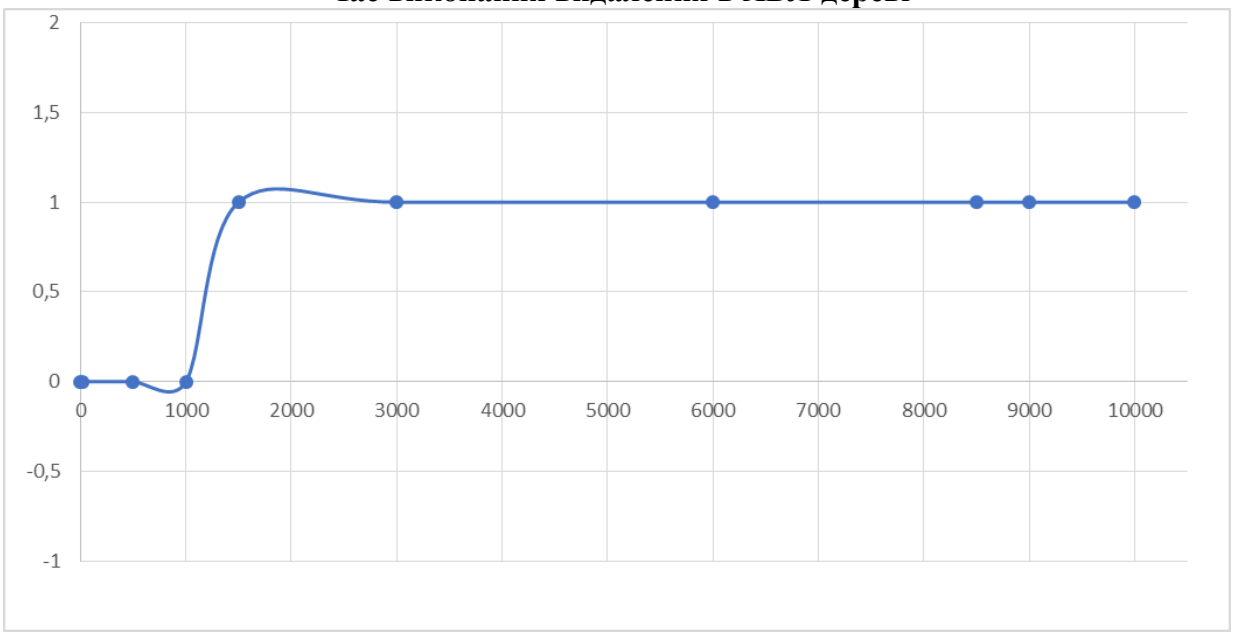
Час виконання невдалого пошуку в червоно-чорному дереві



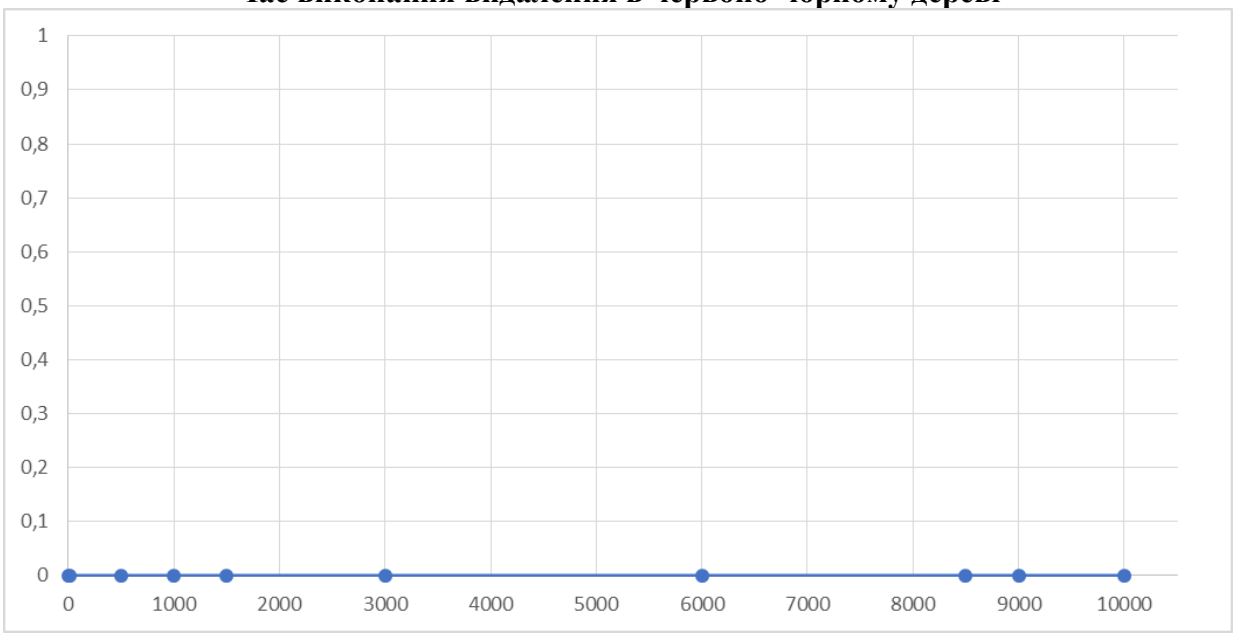
Час виконання невдалого пошуку в бінарному дереві



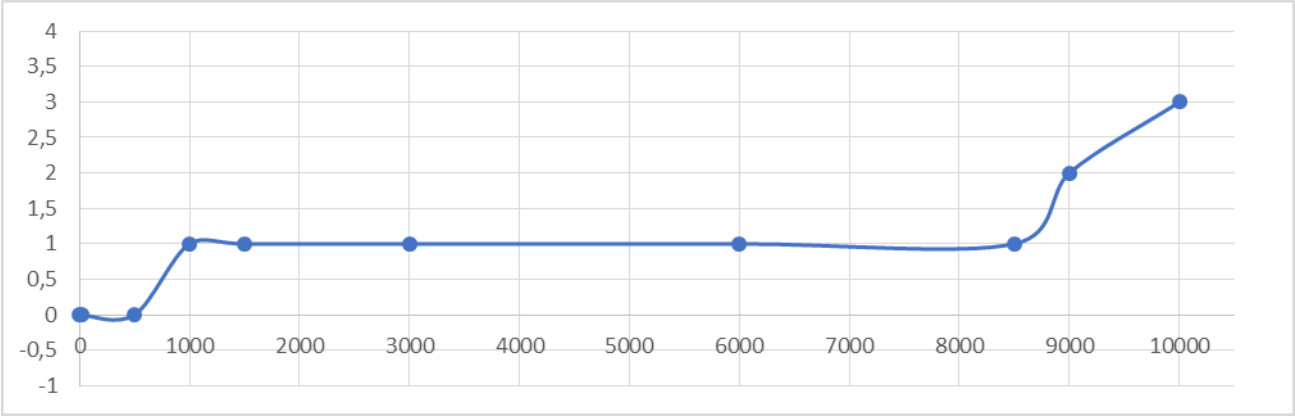
Час виконання видалення в АВЛ-дереві



Час виконання видалення в червоно-чорному дереві



Час виконання видалення в бінарному дереві



Доцільність використання алгоритмів

Як ми можемо побачити на графіках, для однієї кількості елементів є різна швидкість виконання пошуку та видалення елементів в залежності від алгоритму.

Провівши дослідження трьох видів дерев пошуку, ми дійшли висновку, що найкраще працюють саме збалансовані дерева, а не двійкове, адже в збалансованих дерев складність усіх трьох операцій: вдалий та невдалий пошук, видалення, подібна до одиничної, що є показником ефективності даного алгоритму.

Якщо порівнювати роботу АВЛ-дерева та червоно-чорного дерева, то ми дослідили, що червоно-чорне дерево є ефективнішим. Алгоритм балансування червоно-чорного дерева працює швидше, простіше, тому пошук і видалення відбуваються швидше.