МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з курсу «Алгоритми і структури даних»

для студентів, які навчаються за спеціальністю
121 «Інженерія програмного забезпечення»

Затверджено
редакційно-видавничою радою університету, протокол № 3
від 22.12.2016 р.

Харків
НТУ «ХПІ»
2017

Методичні вказівки до виконання лабораторних робіт з курсу «Алгоритми та структури даних» для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» / укл. Н.К. Стратієнко, І.О. Бородіна. – Харків : НТУ «ХПІ», 2017. – 36 с.

Укладачі: Н.К. Стратієнко
І.О. Бородіна

Рецензент В.О. Гужва

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

ВСТУП 6
Загальні рекомендації щодо виконання лабораторних робіт 6
Лабораторна робота 1. Базові структури даних 9
1.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 1 9
1.2. Завдання до лабораторної роботи 1 10
1.3. Варіанти завдань до лабораторної роботи 1 10
Лабораторна робота 2. Базові структури даних. Хеш-таблиці 11
2.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 2 11
2.2. Завдання до лабораторної роботи 2 12
2.3. Варіанти завдань до лабораторної роботи 2 13
Лабораторна робота 3. Базові структури даних: червоно-чорні дерева 14
3.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 3 14
3.2. Завдання до лабораторної роботи 3 15
3.3. Варіанти завдань до лабораторної роботи 3 16
Лабораторна робота 4. Алгоритми сортування 17
4.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 4 17
4.2. Завдання до лабораторної роботи 4 19
4.3. Варіанти завдань до лабораторної роботи 4 20
Лабораторна робота 5. Комбінаторні алгоритми 21

5.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 5 21
5.2. Завдання до лабораторної роботи 5 23
5.3. Варіанти завдань до лабораторної роботи 5 24
Лабораторна робота 6. Фундаментальні алгоритми на графах і деревах 25
6.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 6 25
6.2. Завдання до лабораторної роботи 6 25
6.3 Варіанти завдань до лабораторної роботи 6 26
Лабораторна робота 7. Геометричні алгоритми 27
7.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 7 27
7.2. Завдання до лабораторної роботи 7 27
7.3. Варіанти завдань до лабораторної роботи 7 27
Лабораторна робота 8. Математичні основи теорії алгоритмів 29
8.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 8 29
8.2. Завдання до лабораторної роботи 8 29
Лабораторна робота 9. Рекурсія 30
9.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 9 30
9.2. Завдання до лабораторної роботи 9 30
9.3. Варіанти завдань до лабораторної роботи 9 31
Лабораторна робота 10. Динамічне програмування 32

10.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 10 32
10.2. Завдання до лабораторної роботи 10 32
10.3. Варіанти завдань до лабораторної роботи 10 32
Лабораторна робота 11. Жадібні алгоритми 34
11.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 11 34
11.2. Завдання до лабораторної роботи 10 34
11.3. Варіанти завдань до лабораторної роботи 10 34
Список літератури 35

ВСТУП

Методичні вказівки до виконання лабораторних робіт з курсу «Алгоритми та структури даних» є частиною загального методичного забезпечення курсу «Алгоритми та структури даних», який викладається для студентів спеціальності 121 «Інженерія програмного забезпечення» в 2-му семестрі.

Мета виконання лабораторних робіт – закріплення та поглиблення теоретичних знань, отриманих при вивченні лекційної частини курсу, набуття стійких практичних навичок організації та використання динамічних структур даних для розв’язання задач при створенні програмного забезпечення, вивчення найбільш поширених алгоритмів розв’язання задач з використанням складних структур даних.

Методичні вказівки містять завдання для 11-ти лабораторних робіт та необхідний для їх виконання теоретичний матеріал, а також варіанти завдань до кожної роботи.

ЗАГАЛЬНІ РЕКОМЕНДАЦІЇ ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

На початку навчального семестру кожний студент академічної групи отримує від викладача завдання на виконання лабораторних робіт.

За результатом виконання кожної лабораторної роботи студент формує звіт відповідно до вимог НТУ «ХПІ» щодо оформлення навчальної документації.

Здача лабораторних робіт відбувається протягом усього навчального семестру відповідно до графіка їх проведення. Строк здачі комплексу лабораторних робіт – до останнього тижня семестру.

План проведення лабораторних робіт подано в таблиці 1.1.

Таблиця 1.1 – План проведення лабораторних робіт

Назва лабораторної роботи	Кількість годин
Лабораторна робота 1. Базові структури даних	2
Лабораторна робота 2. Базові структури даних. Хеш-таблиці	2
Лабораторна робота 3.Базові структури даних: червоно-чорні дерева	4
Лабораторна робота 4. Алгоритми сортування	4
Лабораторна робота 5. Комбінаторні алгоритми	2
Лабораторна робота 6. Фундаментальні алгоритми на графах і деревах	4
Лабораторна робота 7. Геометричні алгоритми	2
Лабораторна робота 8. Математичні основи теорії алгоритмів	4
Лабораторна робота 9. Рекурсія	2
Лабораторна робота 10. Динамічне програмування	4
Лабораторна робота 11. Жадібні алгоритми	2

Завдання можна виконувати мовою програмування на вибір:

– C;

– C++;

– Pascal (зараз мало використовується);

– Java;

– C#;

– Python;

– Ruby.

Програму можна демонструвати у будь-якій операційній системі. Рекомендовано розробляти застосунок так, щоб його можна було зібрати та виконувати на широкому класі систем (цього елементарно можна досягнути переліченими мовами). Рекомендовано використовувати будь-який дистрибутив GNU/Linux або FreeBSD.

Вибір середовища розробки може бути продиктований вподобаннями студента. Доцільно використати декілька середовищ для різних робіт.

Перелічимо кілька важливих вільних або безкоштовних редакторів та середовищ розробки:

– Vim;

– Emacs;

– Eclipse;

– NetBeans;

– IntelliJ IDEA;

– KDevelop;

– MonoDevelop;

– Lazarus (вільне середовище, подібне до Delphi).

Лабораторна робота 1. БАЗОВІ СТРУКТУРИ ДАНИХ

Мета роботи: познайомитися з базовими структурами даних (список, черга, стек) та отримати навички програмування алгоритмів, що їх обробляють.

1.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 1

Стеки і черги – це динамічні множини, в яких елемент видаляється з множини операцією *Delete*, яка не задається довільно, а визначається структурою множини. Саме зі стеку (*stack*) можна видалити тільки той елемент, який був у нього доданий останнім: стек працює за принципом «останнім прийшов – першим пішов» (*last-in, first-out – LIFO*). З черги (*queue*), навпаки, можна видалити тільки той елемент, який знаходився в черзі довше за всіх: працює принцип «першим прийшов – першим пішов» (*first-in, first-out – FIFO*).

У зв'язаному списку (або просто списку; *linked list*) елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а вказівниками, що входять до складу елементів списку. Списки є зручним способом реалізації динамічних множин.

Якщо кожен, хто стоїть у черзі, запам'ятає, хто за ним стоїть, після чого всі безладно розсядуться, то вийде однобічно зв'язаний список; якщо він запам'ятає ще й того, хто попереду, буде двобічно зв'язаний список.

Іншими словами, елемент двобічно зв'язаного списку (*doubly linked list*) – це запис, що містить три поля: *key* (ключ) і два вказівники *next* (наступний) і *prev* (попередній). Крім цього, елементи списку можуть містити додаткові дані.

У різних ситуаціях використовуються різні види списків. В односторонньому незв'язаному (*singly linked*) списку відсутні поля *prev*. У впорядкованому (*sorted*) списку елементи розташовані в порядку зростання ключів таким

чином, що у голови списку ключ найменший, а у хвоста списку – найбільший, на відміну від неврегульованого (*unsorted*) списку. У кільцевому списку (*circular list*) поле *prev* голови списку вказує на хвіст списку, а поле *next* хвоста списку вказує на голову списку.

1.2. Завдання до лабораторної роботи 1

Розробити програму, яка читає з клавіатури послідовність N цілих чисел ($1 < N < 256$), жодне з яких не повторюється, зберігає їх до структури даних (згідно з завданням) та видає на екран такі характеристики:

– кількість елементів;

– середнє арифметичне збережених елементів;

– мінімальний та максимальний елемент;

– четвертий елемент послідовності;

– елемент, що йде перед мінімальним елементом.

Наголосимо, що всі характеристики потрібно визначити із заповненої структури даних. Дозволено використовувати лише ті операції, що притаманні заданій структурі, наприклад, заборонено отримувати доступ до елемента із довільною позицією у черзі, яку реалізовано на базі масиву.

Використовувати готові реалізації структур даних (наприклад, STL) заборонено.

1.3. Варіанти завдань до лабораторної роботи 1

Для виконання завдання до лабораторної роботи потрібно використати такі структури даних:

Варіант 1: черга.

Варіант 2: стек.

Варіант 3: однобічно зв’язаний список.

Варіант 4: двобічно зв’язаний список.

Варіант 5: кільцевий список.

Варіант 6: масив із довільним доступом.

Лабораторна робота 2. БАЗОВІ СТРУКТУРИ ДАНИХ. ХЕШ-ТАБЛИЦІ

Мета роботи: познайомитися з хеш-функціями і хеш-таблицями та отримати навички програмування алгоритмів, що їх обробляють.

2.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 2

Часто виникає потреба використання динамічної множини, що підтримує тільки «словникові операції» додавання, пошуку і видалення елемента. У цьому випадку часто застосовують так зване хешування; відповідна структура даних називається хеш-таблицею. У гіршому випадку пошук у хеш-таблиці може займати стільки ж часу, скільки пошук у списку $\left(O(n)\right)$, але на практиці хешування вельми ефективне.

Тоді як при прямій адресації елементу з ключем k відводиться позиція номер k, при хешуванні цей елемент записується в позицію номер $h(k)$ в хеш-таблиці (*hashtable*) $T[0 \ldots m-1]$, де $h: U \to 0, 1, \ldots m-1$ – деяка функція, звана хеш-функцією (*hash function*).

Хороша хеш-функція повинна (наближено) задовольняти припущенням рівномірного хешування: для чергового ключа всі m хеш-значень повинні бути рівноймовірні.

Тривіальне хешування полягає у використанні хеш-функції, що повертає власний аргумент.

Побудова хеш-функції методом ділення із залишком (division method) полягає в тому, що ключу k ставиться у відповідність залишок від ділення k на m, де – m число можливих хеш-значень: $h(k) = k \; mod \; m$.

Наприклад, якщо розмір хеш-таблиці $m = 12$ і ключ дорівнює 100, то хеш-значення дорівнює 4.

Хороші результати зазвичай виходять, якщо вибрати за m просте число, яке є далеким від ступенів двійки.

Побудова хеш-функції методом множення (*multiplication method*) полягає в такому.

Нехай кількість хеш-значень дорівнює m. Зафіксуємо константу A в інтервалі $(0, 1)$ і покладемо $h(k) = [\, m(kA \bmod 1)]$, де $kA \bmod 1$ – дрібна частина kA.

Перевага методу множення полягає в тому, що якість хеш-функції мало залежить від вибору m. Зазвичай за m вибирають ступінь двійки, оскільки в більшості комп'ютерів множення на таке m реалізується як зсув слова.

Хешування Пірсона (*Pearson hashing*) – алгоритм, запропонований Пітером Пірсоном для процесорів з 8-бітними регістрами, завданням якого є швидке обчислення хеш-коду для рядка довільної довжини. На вході функція отримує слово W, що складається з n символів, кожен розміром 1 байт, і повертає значення в діапазоні від 0 до 255. Значення хеш-коду залежить від кожного символу вхідного слова.

Алгоритм, який отримує на вхід рядок w і використовує таблицю перестановок T, можна описати таким кодом мовою Python:

```
def hash_pearson(w):
h = 0
for c in w:
index = h ^ ord(c)
h = T[index]
return h
```

2.2. Завдання до лабораторної роботи 2

Розробити програму, яка читає з клавіатури цілі числа N, M $(1 < N, M < 256)$, N пар $<$ ключ, значення $>$ (при цьому ключ – ціле, дійсне число або рядок залежно від варіанта завдання; значення – рядок; усі рядки розміром до 255 символів, жодний з яких не повторюється) та ще M ключів. Всі рядки розділяються пропуском або новим рядком. Програма зберігає N пар рядків до

хеш-таблиці та видає на екран значення, що відповідають переліченим ключам.

Приклад входу для ключів-рядків:

```
3 2
abc x
gh yq
io qw
gh
io
```

Вихід:

```
yq
qw
```

Використовувати готові реалізації структур даних (наприклад, STL) заборонено, але можна використати реалізацію рядків (наприклад, std::string в C++).

2.3. Варіанти завдань до лабораторної роботи 2

Варіант 1: ключ – ціле число; тривіальне хешування.

Варіант 2: ключ – ціле число; хешування за допомогою ділення.

Варіант 3: ключ – ціле число; мультиплікативне хешування.

Варіант 4: ключ – дійсне число; мультиплікативне хешування.

Варіант 5: ключ – рядок; хешування за остачею суми символів.

Варіант 6: ключ – рядок; хешування Пірсона.

Лабораторна робота 3. БАЗОВІ СТРУКТУРИ ДАНИХ: ЧЕРВОНО-ЧОРНІ ДЕРЕВА

Мета роботи: ознайомитися з червоно-чорними деревами та отримати навички програмування алгоритмів, що їх обробляють.

3.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 3

Дерева як засіб реалізації словників ефективні, якщо їх висота мала, але мала висота не гарантується, і в гіршому випадку дерева не більш ефективні, ніж списки. Червоно-чорні дерева – це один з типів збалансованих дерев пошуку, в яких передбачені операції балансування гарантують, що висота дерева не перевищить $O\ (lg\ n)$.

Червоно-чорне дерево (*red-black tree*) – це двійкове дерево пошуку, вершини якого розділені на червоні (*red*) і чорні (*black*). Таким чином, кожна вершина зберігає один додатковий біт – її колір.

При цьому повинні виконуватися певні вимоги, які гарантують, що глибина будь-яких двох листків дерева відрізняється не більше, ніж у два рази, тому дерево можна назвати збалансованим (*balanced*).

Кожна вершина червоно-чорного дерева має поля *color* (колір), *key* (ключ), *left* (лівий нащадок), *right* (правий нащадок) і *p* (предок). Якщо у вершини відсутній нащадок або предок, відповідне поле містить значення *nil*. Для зручності ми будемо вважати, що значення *nil*, які зберігаються в полях *left* і *right*, є посиланнями на додаткові (фіктивні) листки дерева. При такому заповненні дерева кожна стара вершина, що містить ключ, має двох нащадків.

Двійкове дерево пошуку називається червоно-чорним деревом, якщо воно має такі властивості (будемо називати їх RB-властивостями, *red-black properties*):

1) кожна вершина або червона, або чорна;

2) кожен листок (*nil*) – чорний;

3) якщо вершина червона, обидва її нащадки – чорні;

4) всі шляхи, що йдуть вниз від кореня дерева до його листків, містять однакову кількість чорних вершин.

Операції *Tree-Insert* і *Tree-Delete* виконуються на червоно-чорному дереві за час O $(\lg n)$, але вони змінюють дерево, і результат може не відповідати RB-властивостям. Щоб відновити ці властивості, треба перефарбувати деякі вершини і змінити структуру дерева.

Детально процес повороту та модифікації червоно-чорного дерева описано у лекційному курсі.

3.2. Завдання до лабораторної роботи 3

Розробити програму, яка читає з клавіатури:

5) числа $N,\ M$, при цьому $(1 < N,\ M < 256)$;

6) послідовність N ключів (цілих, дійсних чисел або рядків (до 255 символів) залежно від варіанта завдання);

7) послідовність M ключів.

Програма зберігає першу послідовність до червоно-чорного дерева.

Кожного разу, коли до дерева додається новий елемент, потрібно вивести статистику (згідно з варіантом завдання):

1. Мінімальний елемент та його колір.

2. Максимальний елемент та його колір.

Після побудови дерева для кожного елемента x другої послідовності потрібно вивести результати наступних операцій над деревом (згідно з варіантом завдання):

1. Чи є елемент у дереві та його колір?

2. $Successor(x)$ та його колір.

3. $Predecessor(x)$та його колір.

Використовувати готові реалізації структур даних (наприклад, STL) заборонено, але можна використати реалізацію рядків (наприклад, std::string у C++).

3.3. Варіанти завдань до лабораторної роботи 3

Варіант 1: мінімальний елемент та його колір; чи є елемент у дереві та його колір?

Варіант 2: мінімальний елемент та його колір; $Successor(x)$ та його колір.

Варіант 3: мінімальний елемент та його колір; $Predecessor(x)$ та його колір.

Варіант 4: максимальний елемент та його колір; чи є елемент у дереві та його колір.

Варіант 5: максимальний елемент та його колір; $Successor(x)$ та його колір.

Варіант 6: максимальний елемент та його колір; $Predecessor(x)$ та його колір.

Лабораторна робота 4. АЛГОРИТМИ СОРТУВАННЯ

Мета роботи: познайомитися з алгоритмами сортування та бінарним пошуком.

4.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 4

Багато алгоритмів використовують сортування як проміжний крок. Є багато різних алгоритмів сортування; в конкретній ситуації вибір алгоритму сортування залежить від довжини послідовності, яку необхідно відсортувати, та від того, якою мірою вона вже відсортована, а також від типу наявної пам'яті.

Сортування бульбашкою (*bubble sort*) складається з повторюваних проходів по сортованому масиву. За кожен прохід елементи послідовно порівнюються попарно, і якщо порядок у парі неправильний, виконується обмін елементів. Проходи по масиву повторюються $N - 1$ раз або доти, поки на черговому проході не з'ясується, що обміни більше не потрібні, що означає – масив відсортований. При кожному проході алгоритму по внутрішньому циклу черговий найбільший елемент масиву ставиться на своє місце в кінці масиву поруч з попереднім найбільшим елементом, а найменший елемент переміщається на одну позицію до початку масиву («плине» до потрібної позиції, як бульбашка у воді, звідси і назва алгоритму).

Сортування включенням (*insertion sort*) зручне для сортування коротких послідовностей. Саме в такий спосіб зазвичай сортують карти: тримаючи в лівій руці вже впорядковані карти і взявши правою рукою чергову карту, ми вставляємо її в потрібне місце, порівнюючи з наявними і йдучи справа наліво.

Швидке сортування основане на принципі «розділяй і володарюй». Сортування ділянки $A[p..r]$ відбувається так:

1. Елементи масиву A переставляються так, щоб будь-який з елементів $A[p]; \ldots; A[q]$ був не більший будь-якого з елементів $A[q+1]; \ldots; A[r]$, де q –

деяке число в інтервалі. Цю операцію ми будемо називати поділом (*partition*).

2. Процедура сортування рекурсивно викликається для масивів $A[p..q]$ і $A[q+1..r]$.

3. Після цього масив відсортований $A[p..r]$.

Сортування злиттям основане на принципі «розділяй і володарюй». Спочатку ми розбиваємо масив на дві половини меншого розміру. Потім ми сортуємо кожну з половин окремо. Після цього нам залишається з'єднати два впорядкованих масиви половинного розміру в один. Рекурсивне розбиття задачі на менші відбувається доти, поки розмір масиву не дійде до одиниці (будь-який масив довжини 1 можна вважати впорядкованим).

Нетривіальною частиною є з'єднання двох упорядкованих масивів в один. Воно виконується за допомогою допоміжної процедури $Merge\,(A, p, q, r)$. Параметрами цієї процедури є масив A і числа p, q, r, що вказують на межі об'єднуваних ділянок. Процедура передбачає, що $p < q < r$ та що ділянки $A[p..q]$ і $A[q+1..r]$ вже відсортовані, і зливає (*merges*) їх до однієї ділянки $A[p..r]$.

Сортування купою (пірамідальне сортування) базується на використанні сортувального дерева (бінарної купи). Сортувальне дерево – це таке двійкове дерево, у якого виконані такі умови.

1. Кожен листок має глибину або d, або $d-1$, де d – максимальна глибина дерева.

2. Значення в будь вершини не менше (інший варіант – не більше) від значення її нащадків.

Зручна структура даних для сортувального дерева – такий масив A, що $A[0]$ – елемент в корені, а нащадки елемента $A[i]$є $A[2i+1]$ і $A[2i+2]$.

Алгоритм сортування буде складатися з двох основних кроків:

1. Побудова сортувального дерева, такого, щоб кожний нащадок був меншим за предка. Цей крок вимагає $O(n)$ операцій.

2. Видалення елементів з кореня по одному за раз і перебудова дерева. Тобто на першому кроці обмінюємо $A[0]$ і $A[n-1]$, перетворюємо $A[0], A[1], \dots, A[n-2]$ в сортувальне дерево. Потім переставляємо $A[0]$ і $A[n-2]$, перетворюємо $A[0], A[1], \dots, A[n-3]$ в сортувальне дерево. Процес продовжується доти, поки в сортувальному дереві не залишиться один елемент. Тоді $A[0], A[1], \dots, A[n-1]$ – упорядкована послідовність. Цей крок вимагає $O(n \log n)$ операцій.

Алгоритм сортування підрахунком можна застосувати, якщо кожен з елементів сортованої послідовності – ціле позитивне число у відомому діапазоні (що не перевершує заздалегідь відомого k). Якщо $k = O(n)$, то алгоритм сортування підрахунком працює за час $O(n)$.

Ідея цього алгоритму полягає в тому, щоб для кожного елемента x заздалегідь підрахувати, скільки елементів вхідної послідовності менше x, після чого записати x безпосередньо у вихідний масив відповідно за цим числом (якщо, скажімо, 17 елементів вхідного масиву менше x, то у вихідному масиві x повинен бути записаний за номером 18). Якщо в послідовності, яку необхідно відсортувати, є однакові числа, то цю схему необхідно модифікувати, щоб не записати кілька чисел на одне місце.

4.2. Завдання до лабораторної роботи 4

Розробити програму, яка читає з клавіатури:

1) числа N, M $(1 < N, M < 256)$;

2) послідовність N ключів (цілих або дійсних чисел залежно від варіанту завдання);

3) послідовність M ключів.

Програма зберігає першу послідовність до масиву та виконує сортування. Потім програма виводить відсортовану послідовність на екран та виконує бінарний пошук кожного елемента другої послідовності x: для кожного x повідомити, чи є він у першій послідовності, а якщо є, то на якому місці.

4.3. Варіанти завдань до лабораторної роботи 4

Варіант 1: сортування бульбашкою.

Варіант 2: сортування включенням.

Варіант 3: швидке сортування.

Варіант 4: сортування злиттям.

Варіант 5: сортування купою.

Варіант 6: сортування підрахунком (для цілих чисел).

Лабораторна робота 5. КОМБІНАТОРНІ АЛГОРИТМИ

Мета роботи: познайомитися з генераторами випадкових чисел та методами перевірки випадковості.

5.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 5

Лінійний конгруентний метод застосовується в простих випадках і не має криптографічної стійкості.

Лінійний конгруентний метод полягає в обчисленні членів лінійної рекурентної послідовності за модулем деякого натурального числа , що задається такою формулою:

$$X_{k+1} = (aX_k + c)\ mod\ m,$$

де a і c – деякі коефіцієнти, цілі числа.

Отримана послідовність залежить від вибору стартового числа, і при різних його значеннях виходять різні послідовності випадкових чисел. Разом з тим багато властивостей цієї послідовності визначаються вибором коефіцієнтів у формулі і не залежать від вибору стартового числа.

Особливості розподілу випадкових чисел, що генеруються лінійним конгруентним алгоритмом, роблять неможливим їх використання в статистичних алгоритмах, що вимагають високого ступеня захищеності.

Один з найбільш поширених датчиків Фібоначчі оснований на наступній формулі:

$$X_k = \begin{cases} X_{k-a} - X_{k-b}, \text{якщо } X_{k-a} \geq X_{k-b}, \\ X_{k-a} - X_{k-b} + 1, \text{якщо } X_{k-a} < X_{k-b}, \end{cases}$$

де X_k – дійсні числа з діапазону $[0, 1)$;

a, b – цілі позитивні числа, які називаються лагами.

При реалізації через цілі числа досить формули $X_k = X_{k-a} - X_{k-b}$ (при цьому будуть відбуватися арифметичні переповнення). Для роботи датчика

Фібоначчі потрібно знати $max\{a, b\}$ попередніх згенерованих випадкових чисел, які можуть бути згенеровані простим конгруентним датчиком.

Лаги a і b – «магічні» і їх не слід вибирати довільно. Рекомендуються наступні значення лагів: $(a, b) = (55,24), (17,5)$ або $(97,33)$.

Статистичні тести видають числову характеристику послідовності і дозволяють однозначно сказати, чи пройдено тест. Розглянемо кілька тестів пакета NIST.

Частотний побітовий тест полягає у визначенні співвідношення між нулями і одиницями у всій двійковій послідовності. Мета – з'ясувати, чи дійсно число нулів і одиниць у послідовності приблизно однакові, як це можна було б припустити у випадку істинно випадкової бінарної послідовності. Тест оцінює, наскільки близька частка одиниць до $0,5$. Таким чином, число нулів і одиниць має бути приблизно однаковим. Якщо обчислене в ході тесту значення ймовірності $p < 0,01$, то дана двійкова послідовність не є істинно випадковою. В іншому випадку послідовність має випадковий характер. Потрібно відзначити, що всі наступні тести проводяться за умови, що пройдено даний тест.

Тест на послідовність однакових бітів полягає в підрахунку повного числа рядів у вихідній послідовності, де під словом «ряд» мають на увазі безперервну підпослідовність однакових бітів. Ряд довжиною k біт складається з k абсолютно ідентичних бітів, починається і закінчується з біта, що містить протилежне значення. Мета даного тесту – зробити висновок про те, чи дійсно кількість рядів, що складаються з одиниць і нулів з різними довжинами, відповідає їх кількості у довільній послідовності. Зокрема, визначається швидко чи повільно чергуються одиниці і нулі у вихідній послідовності. Якщо обчислене в ході тесту значення ймовірності $p < 0,01$, то дана двійкова послідовність не є істинно випадковою. В іншому випадку вона має випадковий характер.

Тест на найдовшу послідовність одиниць полягає у визначенні найдовшого ряду одиниць всередині блока довжиною m біт. Мета – з'ясувати,

чи дійсно довжина такого ряду відповідає очікуванням довжини найдовшого ряду одиниць у випадку абсолютно довільної послідовності. Якщо обчислене в результаті виконання тесту значення ймовірності $p < 0{,}01$, то вихідна послідовність не є випадковою. В іншому випадку робиться висновок про її випадковість. Слід зауважити, що з припущення про приблизно однакову частоту появи одиниць і нулів (тест №1) випливає, що точно такі ж результати даного тесту будуть отримані при розгляді самого довгого ряду нулів. Тому вимірювання можна проводити тільки з одиницями.

5.2. Завдання до лабораторної роботи 5

Розробити програму, яка читає з клавіатури число N ($1 < N < 256$) та параметри генератора випадкових чисел і виводить на екран послідовність з N згенерованих чисел. Програма зберігає до файлу графічну характеристику послідовності згідно з завданням та виводить на екран результат одного з тестів NIST (згідно з варіантом завдання).

Варіанти генераторів випадкових чисел:

1) лінійний конгруентний метод;

2) метод Фібоначчі із затримуванням.

Варіанти графічних характеристик:

1) гістограма розподілу елементів послідовності;

2) розподіл на площині (елементи попарно обробляються як координати точок (x, y));

3) автокореляція (користувач задає зсув для копії послідовності).

Варіанти тестів NIST:

1) частотний побітовий тест;

2) тест на послідовність однакових бітів;

3) тест на найдовшу послідовність одиниць.

5.3. Варіанти завдань до лабораторної роботи 5

В таблиці 5.1 наведені варіанти завдань до даної лабораторної роботи.

Таблиця 5.1 – Варіанти завдань до лабораторної роботи

	Генератор випадкових чисел	**Графічна характеристика**	**Тест NIST**
Варіант 1	лінійний конгруентний метод	гістограма розподілу елементів послідовності	частотний побітовий тест
Варіант 2	лінійний конгруентний метод	гістограма розподілу елементів послідовності	тест на послідовність однакових бітів
Варіант 3	лінійний конгруентний метод	гістограма розподілу елементів послідовності	тест на найдовшу послідовність одиниць
Варіант 4	лінійний конгруентний метод	розподіл на площині	частотний побітовий тест
Варіант 5	лінійний конгруентний метод	розподіл на площині	тест на послідовність однакових бітів
Варіант 6	лінійний конгруентний метод	розподіл на площині	тест на найдовшу послідовність одиниць
Варіант 7	лінійний конгруентний метод	автокореляція	частотний побітовий тест
Варіант 8	лінійний конгруентний метод	автокореляція	тест на послідовність однакових бітів
Варіант 9	лінійний конгруентний метод	автокореляція	тест на найдовшу послідовність одиниць
Варіант 10	метод Фібоначчі із затримуванням	гістограма розподілу елементів послідовності	частотний побітовий тест
Варіант 11	метод Фібоначчі із затримуванням	гістограма розподілу елементів послідовності	тест на послідовність однакових бітів
Варіант 12	метод Фібоначчі із затримуванням	гістограма розподілу елементів послідовності	тест на найдовшу послідовність одиниць
Варіант 13	метод Фібоначчі із затримуванням	розподіл на площині	частотний побітовий тест
Варіант 14	метод Фібоначчі із затримуванням	розподіл на площині	тест на послідовність однакових бітів
Варіант 15	метод Фібоначчі із затримуванням	розподіл на площині	тест на найдовшу послідовність одиниць
Варіант 16	метод Фібоначчі із затримуванням	автокореляція	частотний побітовий тест
Варіант 17	метод Фібоначчі із затримуванням	автокореляція	тест на послідовність однакових бітів
Варіант 18	метод Фібоначчі із затримуванням	автокореляція	тест на найдовшу послідовність одиниць

Лабораторна робота 6. ФУНДАМЕНТАЛЬНІ АЛГОРИТМИ НА ГРАФАХ І ДЕРЕВАХ

Мета роботи: ознайомитися зі способами подання графів та отримати навички програмування алгоритмів, що їх обробляють.

6.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 6

Структури даних та алгоритми, які потрібно використати, розглянуто в лекційному курсі, а також в запропонованій літературі.

6.2. Завдання до лабораторної роботи 6

Розробити програму, яка читає з клавіатури:

1) числа $N,\ M$ $(1 < N, M < 256)$ – кількість вершин та ребер графа;

2) послідовність M пар цілих чисел – ребра графа.

Програма зберігає граф та виконує над ним алгоритм згідно з варіантом.

Варіанти подання графів:

1. Матриця суміжності.

2. Список суміжності.

Варіанти алгоритмів:

1. Пошук в ширину. На екран потрібно вивести вершини в порядку обходу. Для кожної вказати час прибуття та предка в дереві обходу.

2. Пошук у глибину. На екран потрібно вивести вершини в порядку обходу. Для кожної вказати час початку розгляду, кінця розгляду та предка у дереві обходу.

1. Топологічне сортування. На екран потрібно вивести ті ж дані, що і для пошуку в глибину, а також результат сортування.

2. Визначити, чи є заданий граф деревом або лісом.

3. Побудувати кістяк дерева алгоритмом Прима.

4. Побудувати кістяк дерева алгоритмом Крускала.

6.3 Варіанти завдань до лабораторної роботи 6

В таблиці 6.1 наведені варіанти завдань до даної лабораторної роботи.

Таблиця 6.1 – Варіанти завдань до лабораторної роботи 6

	Подання графа	**Алгоритм**
Варіант 1	Матриця суміжності	Пошук у ширину
Варіант 2	Матриця суміжності	Пошук у глибину
Варіант 3	Матриця суміжності	Топологічне сортування
Варіант 4	Матриця суміжності	Чи є заданий граф деревом або лісом?
Варіант 5	Матриця суміжності	Алгоритм Прима
Варіант 6	Матриця суміжності	Алгоритмом Крускала
Варіант 7	Список суміжності	Пошук у ширину
Варіант 8	Список суміжності	Пошук у глибину
Варіант 9	Список суміжності	Топологічне сортування
Варіант 10	Список суміжності	Чи є заданий граф деревом або лісом?
Варіант 11	Список суміжності	Алгоритм Прима
Варіант 12	Список суміжності	Алгоритмом Крускала

Лабораторна робота 7. ГЕОМЕТРИЧНІ АЛГОРИТМИ

Мета роботи: познайомитися із основними геометричними алгоритмами.

7.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 7

Основний засіб багатьох геометричних алгоритмів – поняття векторного добутку.

Нехай дано вектори p_1 і p_2. Нас цікавлять тільки вектори, що лежать в одній площині, тому під векторним добутком $p_1 \times p_2$ можна розуміти площу паралелограма (з урахуванням знака), утвореного точками $(0;0)$, $p_1 p_2 p_1 + p_2 = (x_1 + x_2;\ y_1 + y_2)$.

Для обчислень більш зручне визначення векторного добутку як визначника матриці $p_1 \times p_2 = det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$.

Якщо $p_1 \times p_2$ позитивне, то найкоротший поворот p_1 відносно $(0,0)$, що поєднує його з p_2, відбувається проти годинникової стрілки, а якщо негативне, то за нею.

Алгоритми побудови опуклої оболонки розглянуте у лекційному курсі, а також у запропонованій літературі.

7.2. Завдання до лабораторної роботи 7

Розробити програму, яка читає з клавіатури число N $(1 < N < 256)$ та N пар дійсних чисел-координат точок на площині. Програма виконує один з алгоритмів згідно з варіантом.

7.3. Варіанти завдань до лабораторної роботи 7

Варіант 1. Точки належать ламаній. Потрібно для кожної нової ланки вказати, направо чи наліво здійснено поворот.

Варіант 2. Точки належать ламаній. Потрібно для кожної нової ланки вказати, чи перетинає вона будь-яку з попередніх.

Варіант 3. Точки є координатами багатокутника в порядку обходу. Вивести площину багатокутника та повідомити, за годинниковою стрілкою чи проти годинникової стрілки здійснено обхід.

Варіант 4. Побудувати опуклу оболонку наданих точок алгоритмом Грехема.

Варіант 5. Побудувати опуклу оболонку наданих точок алгоритмом Джарвіса.

Лабораторна робота 8. МАТЕМАТИЧНІ ОСНОВИ ТЕОРІЇ АЛГОРИТМІВ

Мета роботи: навчитися визначати складність алгоритмів.

8.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 8

Аналізуючи алгоритм, можна намагатися знайти точне число виконуваних ним дій. Але в більшості випадків достатньо оцінити асимптотику зростання часу роботи алгоритму при прямуванні величини розмірності вхідних даних до нескінченності:

$$\theta\big(g(n)\big) = f(n): \exists \text{ позитивні константи } c_1, c_2 \text{ та } n_0, \text{щоб}$$
$$0 \le c_1 g(n) \le \quad f(n) \le c_2 g(n) \; \forall \; n \ge n_0.$$

8.2. Завдання до лабораторної роботи 8

Для кожного реалізованого алгоритму з попередніх робіт визначити складність. Для цього для кожного рядка алгоритму потрібно вказати кількість разів, що він виконується, залежно від розмірності вхідних даних. Результат записати у вигляді θ-позначення.

Для алгоритмів робіт «Базові структури даних» та «Фундаментальні алгоритми на графах і деревах» визначити, чи є обрані структури даних оптимальними, а якщо ні – запропонувати такі, що призводять до зменшення часу роботи алгоритму.

Лабораторна робота 9. РЕКУРСІЯ

Мета роботи: навчитися оцінювати складність рекурсивних алгоритмів.

9.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 9

Методи, які потрібно використати, розглянуто в лекційному курсі, а також у запропонованій літературі.

Наведемо базову теорему про рекурентні співвідношення.

Нехай $a \geq 1$ та $b > 1$ – константи, $f(n)$ – функція та $T(n)$ – рекурентно визначена для невід'ємних цілих чисел функція:

$$T(n) = at(n/b) + f(n),$$

де ми інтерпретуємо n/b як $\lfloor n/b \rfloor$ або $\lceil n/b \rceil$.

Тоді $T(n)$ має такі асимптотичні оцінки:

1) якщо $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ для деякої константи $\epsilon > 0$, тоді функція $T(n)$ має вигляд: $T(n) = \theta\left(n^{\log_b a}\right)$;

2) якщо $f(n) = \theta\left(n^{\log_b a}\right)$, тоді $T(n) = \theta\left(n^{\log_b a} \lg n\right)$;

3) якщо $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ для деякої константи $\epsilon > 0$ та якщо $af(n/b) \leq cf(n)$ для деякої константи $c < 1$ і всіх достатньо великих n, тоді $T(n) = \theta(f(n))$.

9.2. Завдання до лабораторної роботи 9

Для алгоритму, складність якого описано виданою згідно з варіантом рекурентною функцією вигляду $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, визначити складність його виконання, використовуючи такі методи:

1) метод підстановки;

2) перетворення до суми;

3) базова теорема про рекурентні співвідношення;

4) моделювання за допомогою програми (отримати значення $T(n)$ для різних n та підібрати формулу).

Отримані результати порівняти.

9.3. Варіанти завдань до лабораторної роботи 9

Варіант 1: $T(n) = 2T\left(\frac{n}{2}\right) + n.$

Варіант 2: $T(n) = 2T\left(\frac{n}{2} + 35\right) + n.$

Варіант 3: $T(n) = 4T\left(\frac{n}{5}\right) + 2n.$

Варіант 4: $T(n) = 4T\left(\frac{n}{5}\right) + n^3.$

Варіант 5: $T(n) = 2T\left(\frac{n}{2}\right) + \lg n.$

Варіант 6: $T(n) = 3T\left(\frac{n}{3}\right) + n \lg^2 n.$

Лабораторна робота 10. ДИНАМІЧНЕ ПРОГРАМУВАННЯ

Мета роботи: навчитися використовувати динамічне програмування та оцінювати його складність.

10.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 10

Структури даних та алгоритми, які потрібно використати, розглянуто в лекційному курсі, а також у запропонованій літературі.

10.2. Завдання до лабораторної роботи 10

Розробити програму, яка читає з клавіатури вхідні дані та розв'язує задачу методом динамічного програмування. Визначити складність алгоритму.

10.3. Варіанти завдань до лабораторної роботи 10

Варіант 1. Пошук маршруту на прямокутному полі таким чином, щоб сума чисел у клітинках, через які він проходить, була максимальною. Рух починається із клітинки верхнього лівого кута та завершується у правому нижньому. Кожний крок виконується на одну позицію праворуч або вниз.

Вхідні дані: натуральні числа N, M $(1 < N, M < 256)$ та послідовність $N \times M$ натуральних чисел – прямокутне поле рядок за рядком.

Вихідні дані: таблиця динамічного програмування ($A(i, j)$ = максимальна сума маршруту до клітинки (i, j)); прямокутне поле із маршрутом, що відмічено зірочками, та сума чисел у клітинках маршруту.

Варіант 2. Пошук найбільшої спільної підпослідовності.

Вхідні дані: натуральні числа N, M $(1 < N, M < 256)$ та дві послідовності X та Y натуральних чисел довжиною N та M відповідно.

Вихідні дані: динамічна таблиця ($A(i, j)$ = довжина НСП для префіксів X_i та Y_i) та НСП для X та Y.

Варіант 3. Пошук оптимального способу множення матриць.

Вхідні дані: натуральне число N $(1 < N < 256)$ – кількість матриць, натуральні числа $x_i,\ i = \overline{0, N}$ – розмірності матриць (матриця $B_i,\ i = \overline{1, N}$ має розмірність $x_{i-1} \times x_i$).

Вихідні дані: таблиця динамічного програмування ($A(i, j)$ = найменша кількість множень для обчислення добутку матриць $B_i, B_{i+1}, \dots, B_{j-1}, B_j$) та оптимальна розстановка дужок у виразі $A_1, A_2, \dots A_N$.

Лабораторна робота 11. ЖАДІБНІ АЛГОРИТМИ

Мета роботи: навчитися використовувати жадібні алгоритми та оцінювати їхню складність.

11.1. Теоретичний матеріал, необхідний для виконання лабораторної роботи 11

Структури даних та алгоритми, які потрібно використати, розглянуто в лекційному курсі, а також у запропонованій літературі.

11.2. Завдання до лабораторної роботи 10

Розробити програму, яка читає з клавіатури вхідні дані та розв'язує задачу жадібним алгоритмом. Визначити складність алгоритму.

11.3. Варіанти завдань до лабораторної роботи 10

Варіант 1. Розв'язати задачу про вибір найбільшої кількості заявок для аудиторії. *Вхідні дані:* кількість заявок N $(1 < N < 256)$, N пар натуральних чисел – початок та кінець заявок s_i, f_i. *Вихідні дані:* заявки, відсортовані за зростанням часу закінчення, та номери заявок, які потрібно обрати.

Варіант 2. Стиснути послідовність нулів та одиниць алгоритмом Хаффмана. *Вхідні дані:* натуральні числа N, M $(1 < N < 256, 1 < M < 9)$ та послідовність N груп по M нулів та одиниць. *Вихідні дані:* перелік кодів, які потрібно використати для кожної групи, та стиснута послідовність.

Варіант 3. Розв'язати неперервну задачу про рюкзак. *Вхідні дані:* натуральні числа N, W $(1 < N < 256, 1 < W < 1024)$ – кількість видів товару та місткість рюкзака, послідовність N пар $p[i], v[i]$ – вартість та вага товару номер і. *Вихідні дані:* набір дійсних чисел, які вказують, скільки кожного товару потрібно покласти до рюкзака, щоб сумарна вартість була максимальною.

СПИСОК ЛІТЕРАТУРИ

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона / Н. Вирт. – М. : ДМК Пресс, 2010. – 272 с.

2. Ахо А.В. Структуры данных и алгоритмы / А.В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. – М. : Вильямс, 2003. – 384 с.

3. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер. с англ. под ред. А. Шеня. – 3-е изд. – М. : ООО «И. Д. Вильямс», 2013. – 1398 с.

4. Игошин В.И. Математическая логика и теория алгоритмов : учеб. пособ. для студ. высш. учеб. заведен. / В.И. Игошин. – 2-е изд., стер. – М. : Издательский центр «Академия», 2008. – 448 с.

5. Игошин В.И. Задачи и упражнения по математической логике и теории алгоритмов / В.И. Игошин. – 3-е изд., стер. – М. : Издательский центр «Академия», 2007. – 304 с.

6. Романовский И.В. Вычислительная математика и структура алгоритмов / И.В. Романовский. – М. : МГУ, 2006. – 112 с.

7. Cormen T.H. Introduction to Algorithms / T.H. Cormen, C. E. Leiserson, R.L. Rivest, C. Stein. – 3rd ed. – Cambridge : MIT Press and McGraw-Hill, 2009 – 1292 p.

8. Wirth. N. Algorithms + Data Structures = Programs / N. Wirth. – Prentice-Hall, 1976. – 183 p.

9. Aho A.V. Data Structures and Algorithms / A.V. Aho, J.E. Hopcroft, J.D. Ullman. – Addison-Wesley, 1983. – 384 p.

Навчальне видання

Стратієнко Наталія Костянтинівна
Бородіна Інна Олександрівна

Методичні вказівки до виконання лабораторних робіт з курсу «Алгоритми і структури даних»

для студентів, які навчаються за спеціальністю
121 «Інженерія програмного забезпечення»

Відповідальний за випуск М.Д. Годлевський
Роботу до видання рекомендував О.В. Горілий

Редактор О.С. Самініна

План 2017 р., поз. 8

Підписано до друку 19.01.2017 р. Формат 60x84 1/16. Папір офсет.
Друк – ризографія. Гарнітура Times New Roman. Ум. друк. арк. 1,6.
Наклад 50 прим. Зам № _____. Ціна договірна.

Видавничий центр НТУ «ХПІ». 61002, Харків, вул. Фрунзе, 21.
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.

Електронна версія