МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

«Алгоритми і структури даних»

з курсу «Алгоритми і структури даних»

для студентів спеціальностей
122 «Комп’ютерні науки»,

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 28.06.2023 р.

Харків
НТУ «ХПІ»
2023

Методичні вказівки до виконання лабораторних робіт з курсу «Алгоритми і структури даних /уклад. К. В. Ягуп. – Харків : НТУ «ХПІ». – 29 с.

Укладач К. В. Ягуп

Рецензент А. М. Копп

Кафедра програмної інженерії та інтелектуальних систем управління

Вступ

Ефективне вирішення задач по програмуванню зумовлене не тільки вдалим алгоритмом вирішення, але і правильним вибором структури даних, що дозволить реалізувати основні операції алгоритму: додаток, видалення та пошук елементів, послідовність їх обробки. Алгоритм і структура даних нерозривно пов'язані між собою, адже рішення про структурування даних не можна прийняти без знання алгоритмів, що будуть застосовуватися, і навпаки, вибір алгоритму суттєво залежить від структури даних

Методичні вказівки по лабораторним роботам розглядають такі теми: стек та черга і їх програмна організація, динамічне програмування, жадібні алгоритми, алгоритми для формування прсевдовипадкових чисел, основні методи сортування, а саме сортування бульбашкою, сортування вибором, сортування вставками, швидке сортування, сортування Шелла, сортування злиттям, сортування купою.

Для всіх робіт складені докладні інструкції, щодо програмної реалізації відповідних алгоритмів і структур даних, надані основні теоретичні відомості та наведені приклади із ілюстраціями.

Для виконання лабораторних робіт застосовується мова програмування C++. При виконанні лабораторних робіт необхідно написати відповідну програму із назвами змінних і функцій, які запропоновані в методичних вказівках і скласти звіт. Звіт повинен містити:

1. Опис програми
2. Значення змінних, протягом виконання всієї програми
3. Блок схеми
4. Відповідні пояснення

Також в кінці кожної лабораторної роботи наведені контрольні питання до відповідної теми для закріплення знань студентами.

Лабораторна робота №1.

Структура даних стек

Загальна інформація

Стек – це структура даних, у якій елементи підтримують принцип LIFO (“Last in – first out”): останнім зайшов – першим вийшов. Або першим зайшов – останнім вийшов.

Стек дозволяє зберігати елементи і підтримує зазвичай дві базові операції:

• PUSH – кладе елемент на вершину стека

• POP – знімає елемент з вершини стека, переміщуючи вершину до наступного елемента

Також часто зустрічається операція PEEK, яка отримує елемент на вершині стека, але не знімає його звідти.

Стек є однією з базових структур даних і використовується не тільки у програмуванні, а й у схемотехніці, і просто у виробництві, для реалізації технологічних процесів тощо; Стек використовується як допоміжна структура даних у багатьох алгоритмах та в інших більш складних структурах.

Приклад

Нехай ми маємо стек, для зберігання чисел. Виконаємо кілька команд. Спочатку стек порожній. Вершина стека – покажчик на перший елемент, що нікуди не вказує. У разі використання мови С вона може дорівнювати NULL.

Push 3

Тепер стек складається з одного елемента числа 3. Вершина стека вказує на число 3.

Push 5

Стек складається з двох елементів, 5 та 3, при цьому вершина стека вказує на 5.

Push 7

Стек складається із трьох елементів, вершина стека вказує на 7.

Pop

Поверне значення 7, у стеку залишиться 5 і 3. Вершина вказуватиме на наступний елемент – 5.

Pop

Поверне 5, в стеку залишиться лише один елемент, 3, який буде вказувати вершина стека.

Pop

Поверне 3, стек стане порожнім.

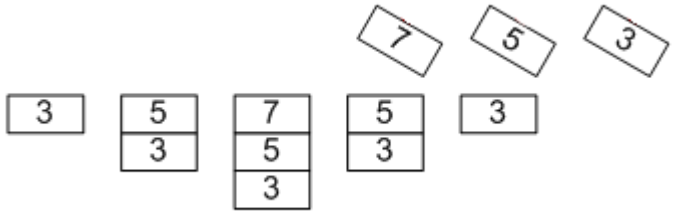

Рис. 1.1 – Послідовне виконання операцій
push 3, push 5, push 7, pop, pop, pop

Часто порівнюють стек зі стопкою тарілок. Щоб дістати наступну тарілку, потрібно зняти попередні. Вершина стека – це вершина стопки тарілок.

Коли ми будемо працювати зі стеком, можливі дві основні помилки, що часто зустрічаються:

• 1. Stack Underflow: Спроба зняти елемент із порожнього стека

• 2. Stack Overflow: Спроба покласти новий елемент на стек, який не може більше зростати (наприклад, не вистачає оперативної пам'яті)

Програмна реалізація стеку фіксованого розміру, побудований на масиві

Поняття і навики, необхідні для програмної реалізації: структура, покажчик, посилання, функція, передача параметрів функції у вигляді покажчиків, операція ->.

Відмінна риса – простота реалізації та максимальна швидкість виконання. Такий стек може застосовуватися в тому випадку, коли його максимальний розмір відомий заздалегідь або відомо, що він малий.

Порядок дій виконання програмної реалізації

1. Необхідно задати константу, що задає максимальний розмір статичного масива max_size.
2. Задаємо структуру даних Stack_tag з двома полями: масив data – із максимальним розміром, що буде представляти собою стек і змінна size покажчик на вершину стеку, яка водночас відображає кількість елементів. Вершина буде вказувати на наступний елемент масиву, в який буде занесене значення.
3. Виконати реалізацію функції pushSt, що виконує вставку елементу у стек. Параметрами функції мають бути покажчик типу структура Stack_tag на стек st та значення, яке буде заноситися в стек value.

Доступ до полів стеку виконується за допомогою оператора ->. В функції виконуються дві операції: операція занесення значення value в масив data та збільшення значення покажчика size на одиницю.

Також необхідно передбачити ситуацію виходу величини size за межі максимального розміру масиву. В цьому випадку скористуйтесь оператором exit(0). Подивитися, як буде поводити програма, при виході розміру масиву за межі максимального розміру. Перевірити, як буде робити програма в цьому випадку без використання команди exit(0).

4. Виконати реалізацію функції popSt, що повертає значення стеку з вершини. Функція має єдиний параметр – покажчик типу структура Stack_tag.

Функція виконує повернення значення з вершини, тобто останнього елементу масиву data, і зменшує значення покажчика size на одиницю (тобто переходить до наступного елементу). Також необхідно передбачити ситуацію, коли size дорівнює нулю.

5. Виконати реалізацію peekSt, що має повертати поточний елемент вершини. Функція має параметр покажчик типу структура на стек st. При досягненні покажчика нульового значення виконати команду exit(0).

6. Створити допоміжну функцію виводу стека printStackValue, що виводить окремий символ на консоль і в кінці символ '|'. Параметром функції є символ, який необхідно вивести.

7. Створити функцію printStack, яка виводить весь вміст утвореного стеку. Параметрами функції мають бути посилання на структуру стек і посилання на функцію printStackValue. Функція має за допомогою циклу for викликати функцію printStackValue параметром якої має бути зміст data із номером змінної в циклі.

Контрольні питання

1. Що називається стеком?
2. Які операції можна виконувати над стеком?
3. За яким принципом функціонує стек?
4. Що називають вершиною стека?
5. Яким чином можна реалізувати структуру стек в С++?
6. Яким чином можна реалізувати вставку нового елементу в стек в С++?
7. Яким чином можна реалізувати вилучення елементу із стека в С++?

Лабораторна робота №2.

Структура даних черга

Загальна інформація

Черга це такий послідовний список, із змінною довжиною, який підтримує принцип FIFO (“First in – first out” першим зайшов, першим вийшов). В черзі включення елементів здійснюється з одного боку – хвості черги, а виключення елементів з іншого – голови черги. Над чергою можна здійснювати такі основні операції: включення нового елементу, виключення елементу, визначення розміру, не руйнуюче читання, очищення.

Черга в програмуванні використовується, як і в реальному житті, коли потрібно вчинити якісь дії в порядку надходження, виконавши їх послідовно. Прикладом може бути організація подій у Windows. Коли користувач робить якусь дію на додаток, то в додатку не викликається відповідна процедура (адже в цей момент додаток може вчиняти інші дії), а йому надсилається повідомлення, що містить інформацію про чинність, це повідомлення ставиться в чергу, і тільки коли будуть оброблені повідомлення, що прийшли раніше, програма виконає необхідну дію.

Існує кілька способів реалізації черги:

за допомогою одновимірного масиву;

за допомогою зв'язаного списку;

за допомогою класу об'єктно-орієнтованого програмування.

Найпростіші операції з чергою:

init() – ініціалізація черги.

insert (q, x) – переміщення елемента x на кінець черги q (q - покажчик на чергу);

x = remove (q) – видалення елемента x з черги q;

isempty(q) – повертає 1, якщо черга порожня і 0 в іншому випадку;

print(q) – виведення елементів черги q.

Програмна реалізація черги на основі масиву фіксованого розміру

Розглянемо реалізацію черги з урахуванням масиву. Використовуємо масив та дві змінні:

head – позиція першого елемента у черзі;

tail – позиція останнього елемента у черзі

Спочатку head=1 та tail=0.

Черга порожня, якщо tail<head.

Число елементів у черзі можна визначити як

n = tail-head+1

Порядок дій виконання програмної реалізації черги

1. Для реалізації черги використовувати структуру Queue, яка має такі поля: que – масив, що зберігатиме значення всіх елементів (розмір масиву дорівнює константі max), і два покажчики head – покажчик, що вказує на початок черги, за допомогою якого здійснюється видалення елементів, tail – покажчик, що вказує на кінець черги, куди додаються елементи.
2. Створити функцію init ініціалізації черги, що присвоює початкові значення покажчикам head=1, tail = 0.
3. Створити функцію додавання елементу в чергу insert. Параметрами функції є покажчик на структуру Queue, та значення, яке додається до структури value. Функція має збільшувати на одиницю покажчик tail і додавати значення у масив que. Передбачити ситуацію, коли значення покажчика перевищує максимальне значення масиву.
4. Створити функцію isempty, яка перевіряє, чи є пустою черга. Параметром функції є покажчик на структуру Queue. Функція повертає 1, якщо виконується умова tail< head, якщо умова не виконується, то функція повертає 0.
5. Створити функцію виводу елементів черги print. Параметр функції – покажчик на структуру Queue. Вивід елементів здійснюється за допомогою циклу for.
6. Створити функцію видалення елемента remove. Параметр функції покажчик на структуру Queue. Функція повертає значення елементу, на котрий вказує покажчик head і збільшує значення покажчика на одиницю.

Контрольні питання

1. Що називається чергою? Який принцип роботи підтримує черга?
2. Які операції можна виконувати над чергою?
3. За яким принципом функціонує черга?
4. Скільки покажчиків використовують у структурі черга і для чого?
5. Яким чином можна реалізувати структуру черга в С++?
6. Яким чином можна реалізувати вставку нового елементу в чергу в С++?
7. Яким чином можна реалізувати вилучення елементу із черги в С++?

Лабораторна робота №3.

Методи сортування

Загальна інофрмація

Сортування (англ. Sorting – класифікація, впорядкування) називається процес упорядкування множини об'єктів за якоюсь ознакою. Сортування розподіляє елементи в порядку, зручному для роботи. Якщо відсортувати масив чисел у порядку зменшення, то перший елемент завжди буде найбільшим, а останній найменшим. Тому бажано зберігати інформацію впорядковано, щоб було легше проводити над нею операції.

Існує такі основні види сортування:

1. Бульбашкове сортування (Bubble sort);
2. Сортування вибором (Selection sort);
3. Сортування вставками (Insertion sort);
4. Швидке сортування (Quick sort);
5. Сортування злиттям (Merge sort);
6. Сортування Шелла (Shell sort);
7. Сортування купою (Heap sort).

1. Сортування бульбашкою

У сортуванні бульбашкою кожен елемент порівнюється з наступним. Якщо два таких елементи не стоять у потрібному порядку, то вони змінюються між собою місцями. Наприкінці кожної ітерації (далі називаємо їх проходами) найбільший/найменший елемент ставиться до кінця списку.

Тобто обходимо масив від початку до кінця, принагідно міняючи місцями невідсортовані сусідні елементи. В результаті першого проходу на останнє місце "випливе" максимальний елемент. Тепер знову обходимо невідсортовану частину масиву (від першого елемента до передостаннього) і міняємо на шляху невідсортованих сусідів. Другий за величиною елемент опиниться на передостанньому місці. Продовжуючи в тому ж дусі, будемо обходити всю невідсортовану частину масиву, що зменшується, переставляя знайдені максимуми в кінець.

Порядок дій виконання програмної реалізації сортування бульбашкою

Неупорядковані елементи зберігаються у одномірному масиві list.

1. Сортування має здійснювати спеціальна функція bubbleSort, яка приймає два параметри – вказівник на масив list та його розмір listLength.
2. Функція має використовувати стяг, який приймає значення true, у разі того, якщо пройшла перестановка елементів масиву. Якщо стяг залишається

рівним false, то функція достроково закінчує своє виконання за допомогою оператора break.

3. В функції циклічно (цикл for) проводиться порівняння двох сусідніх елементів.
4. Якщо попередній елемент з номером і більший за елемент із номером і+1, то елементи міняються місцями. Для цього використовується стандартна функція swap.
5. Після такої ітерації найбільший елемент опиниться в кінці масиву і надалі можна застосовувати цикл for для масиву, на одиницю меншу за попередній. Для цих цілей цикл for можна заключити в цикл while, який зменшує на одиницю довжину масиву listLength.
6. В блоці main здійснити вивід невпорядкованого масиву, виконати функцію сортування bubbleSort і вивести на консоль впорядкований масив.

Контрольний приклад

Розглянемо сортування бульбашкою на прикладі із п’яти елементів. Елементи, які необхідно на поточному етапі поміняти місцями виділимо жирними лініями, а ті, які не треба – пунктиром.

Прохід 1

Найбільший елемент 20 тепер в кінці списку. Тепер можна пройтися лише до четвертої позиції

Прохід 2

Прохід 3

Прохід 4

Сортування вибором

Шукаємо найменше значення в масиві та ставимо його на позицію, звідки розпочали прохід. Потім рухаємось на наступну позицію. Тобто проходимо по масиву в пошуках мінімального елемента. Знайдений мінімум міняємо місцями із першим елементом. Невідсортована частина масиву зменшилася на один елемент (не включає перший елемент, куди ми переставили знайдений мінімум). До цієї невідсортованої частини застосовуємо ті ж дії – знаходимо мінімум і ставимо його на останнє місце у невідсортованій частині масиву. І так продовжуємо доти, доки невідсортована частина масиву не зменшиться до одного елемента.

Порядок дій виконання програмної реалізації сортування вибором

1. Необхідно реалізувати функцію вибору найменшого елемента findSmallestPosition, параметрами якої є показник на масив list[ ], його розмір listLength та номер початкової позиції елемента startPosition.
2. В функції задається номер позиції з найменшим елементом smallestPosition, якій присвоюється значення startPosition.
3. Далі використовується цикл for, початкове значення якого дорівнює startPosition. В циклі здійснюється порівняння поточного елемента з найменшим. Якщо поточний елемент найменший, то змінній smallestPosition номеру найменшого елемента присвоюється значення i. Функція повертає номер позиції найменшого елемента.
4. Ця функція використовується в іншій функції selectonSort, яка здійснює сортування вибором. Параметрами цієї функції є посилання на масив та його довжина.
5. Функція застосовує цикл for, в якому задається змінна smallestPosition, якій присвоюється значення функції findSmallestPosition. Після чого виконується стандартна функція swap, яка міняє місцями поточний елемент із найменшим елементом.

Контрольний приклад

Шукаємо найменше значення в масиві і міняємо цей елемент місцями із першим елементом. Потім переходимо на наступну позицію і виконуємо пошук в масиві скоротивши його довжину на одиницю.

3. Сортування вставками

Загальна суть сортувань вставками така:

Перебираються елементи у невідсортованій частині масиву. Кожен елемент вставляється у відсортовану частину масиву те місце, де він повинен перебувати.

Тобто, сортування вставками завжди ділять масив на 2 частини – відсортовану та невідсортовану. З невідсортованої частини витягується будь-який елемент. Оскільки інша частина масиву відсортована, то досить швидко можна знайти своє місце для цього вилученого елемента. Елемент вставляється куди потрібно, у результаті відсортована частина масиву збільшується, а невідсортована зменшується. За таким принципом працюють усі сортування вставками. Найслабше місце в цьому підході – вставка елемента у відсортовану частину масиву.

В результаті такого сортування масив проходимо зліва направо і обробляємо по черзі кожен елемент. Зліва від чергового елемента нарощуємо відсортовану частину масиву, праворуч у міру процесу потихеньку випаровується невідсортована. У відсортованій частині масиву шукається точка вставки чергового елемента. Сам елемент відправляється в буфер, в результаті чого в масиві з'являється вільний осередок – це дозволяє зрушити елементи та звільнити точку вставки.

На прикладі простих вставок показово виглядає головна перевага більшості сортувань вставками, а саме дуже швидка обробка майже впорядкованих масивів.

Порядок дій виконання програмної реалізації сортування вставками

У сортуванні вставками починаємо з другого елемента. Перевіряємо між собою другий елемент із першим і, якщо треба, міняємо місцями. Порівнюємо наступну пару елементів та перевіряємо всі пари до неї. Для реалізації такого сортування необхідно:

1. Створити функцію сортування вставками selectionSort, параметрами якої є масив, який необхідно відсортувати, та його розмір.
2. В функції прохід по масиву здійснюється за допомогою оператора for. Прохід починається з першого елементу.
3. Для розділення масиву на відсортовану та невідсортовану частини вводиться допоміжна змінна j = i - 1 – значення початкового елементу невідсортованної частини.
4. Змінна j використовується в циклі while, де циклічно порівнюються сусідні елементи невідсортованної частини масиву. В кінці кожного циклу змінна j зменшується на одиницю, якщо поточний елемент більший наступного.
5. Якщо поточний елемент більший наступного, то вони міняються містами за допомогою стандартної функції swap.

6. Реалізувати виконання функції у частині програми main для масиву, розміром не менше 8 елементів. Підрахувати кількість разів обміну містами елементів.

Контрольний приклад

Сортування починається із другого елементу.

Порівнюємо значення першого і другого елементу і якщо значення першого елементу більше, то міняємо елементи місцями.

Прохід 1. Починаємо з другої позиції

Прохід 2. Перевіряємо другу і третю позиції. Потім першу і другу.

Прохід 4. Починаємо з четвертої позиції. Перевіряємо третю і четверту позиції. Потім другу і третю, а потім першу і другу.

Прохід 5.

4. Сортування Шелла

Вдосконаленим варіантом сортування вставками є сортування Шелла. В цьому методі порівнюються не тільки елементи, що стоять поряд, але й елементи, що знаходяться на заданій відстані один від одного. При цьому відстань циклічно зменшується в два рази. Коли крок стає рівним 1, то просто здійснюється сортування вставками.

1. Сортування масиву необхідно виконувати спеціальною функцією shellSort параметрами якої є масив list[] та його довжина listlength.
2. В функції потрібно застосувати два цикли for. Перший цикл використовується для того щоб поступово зменшувати крок step вдвічі, поки він більший за нуль.
3. Другий, вкладений в нього цикл for застосовується для того, щоб при заданому кроці порівнювати пари елементів масиву і після порівняння зміщуватися вправо, поки не буде досягнуто кінця масиву. Таке зміщення здійснюється за допомогою допоміжної змінної *i* початкове значення якої дорівнює значенню step, а потім збільшується на одиницю.
4. В середині другого циклу for необхідно організувати порівняння двох елементів, які знаходяться один від одного на відстані step і у випадку, якщо первший елемент більший за другий, поміяти їх місцями. Для цього можна застосувати оператор while, який виконується до тих пір, поки виконуються дві умови: допоміжна змінна j більша за step і перший елемент більший за другий. Якщо обидві умови виконуються, то елементи міняються місцями і змінна j зменшується на величину кроку step.

Контрольний приклад

Зменшуємо крок в два рази. Він стає рівним 2.

5. Швидке сортування

В основі швидкого сортування лежить стратегія поділу завдання на більш дрібні підзавдання. Підзавдання вирішуються окремо, а потім рішення об'єднують. У випадку із масивом, він поділяється на підмасиви, які сортуються і потім зливаються в один.

1. В першу чергу вибирається опорний елемент pivot, з яким у циклі будуть порівнюватися всі інші елементи. В якості такого елементу можна вибрати останній елемент масиву.
2. Реалізуємо функцію partition, яка буде реалізовувати порівняння елементів масиву з опорним елементом partition. Функція матиме три параметри масив list[], номер першого елементу списку start і номер опорного елементу pivot.
3. Функція має розміщати елементи, які менші за опорний зліва, а більші за опорний – справа. Рухатися треба від початку масиву. Для цього можна використовувати цикл while, який контролює те, що номер поточного елементу менший за номер опорного елементу.
4. Якщо поточний елемент більший за опорний, то цей елемент переставляється слідом за опорним, а на його місце ставиться елемент, порядковий номер якого на одиницю менший за номер опорного елементу. Може виникнути ситуація, коли номер поточного елементу на одиницю менший за номер опорного елементу. В цьому випадку поточний елемент з поточним просто міняються місцями. В обох випадках порядковий номер поточного елемента не треба збільшувати на одиницю. Але так, як в результаті операцій, опорний елемент зміщується на одну позицію вліво, його порядковий номер треба зменшити на одиницю.
5. Якщо поточний елемент менший за опорний, то порядковий номер елементу збільшується на одиницю в кінці циклу while.
6. Після виконання циклу while функція має повертати значення порядкового номеру опорного елементу.
7. Після виконання функції partition масив буде розділений на дві частини. Для сортування утворених частин масиву можна застосувати рекурсивну функцію, яка застосовує функцію partition. Умовою виходу за рекурсії має бути умова, що порядкові першого елементу і останнього в параметрах функції partition стають рівними.

Контрольний приклад

Тут подвійною лінією позначаємо опорний елемент. Всі елементи більші за нього перенесемо в праву частину масиву від нього, а менші в ліву.

Масив розділяється за опорним елементом на два масиви. В цих масивах обираємо також опорні елементи. Опорними елементами можуть бути будь-які. В даному прикладі ми обираємо останній в масиві або підмасиві.

Розглянемо, яким чином ми будемо розташовувати елементи відносно опорного елементу. Якщо значення елементу більше опорного, то воно ставиться на позицію опорного елементу, а опорний елемент перед ним.

Пунктирною лінією виділимо елемент із значенням меншим за опорний елемент, а жирною лінією – більшим за опорний елемент.

Контрольні питання

1. Що представляє собою сортування?
2. Яким чином можна сортувати об’єкти?
3. Які існують основні види сортування?
4. Яким чином здійснюються сортування бульбашкою?
5. Яким чином можна скорочувати довжину невідсортованого масиву при сортуванні бульбашкою?
6. Яким чином здійснюється сортування вибором?
7. Яким чином здійснюється сортування вставками?
8. Які оператори циклів бажано використовувати для реалізації сортування вставками?
9. Яким чином здійснюється сортування Шелла? Які оператори застосовуються для сортування таким чином?
10. Яка стратегія лежить в основі швидкого сортування?
11. Яким чином розташовуються елементи відносно опорного елементу в сортуванні вставками?
12. Яким чином можна застосувати рекурсію у методі швидкого сортування?

Лабораторна робота №4
Сортування злиттям (Merge sort)

Сортування злиттям також слідує за стратегією «розділяй і володарюй». Поділяємо вихідний масив на два рівні підмасиви. Повторюємо сортування злиттям цих двох підмасивів і об'єднуємо назад.

Рис. 1.1 – Сортування злиттям на прикладі

Цикл розподілу повторюється, поки залишиться по одному елементу в масиві. Потім об'єднуємо, доки не утворимо повний список.

Алгоритм сортування складається з чотирьох етапів:

1. Знайти середину масиву.
2. Сортувати масив від початку до середини.
3. Сортувати масив від середини до кінця.
4. Об'єднати масив.

Для ділення масиву на дві частини необхідно створити рекурсивну функцію mergeSort, яка має три параметри – масив, який треба відсортувати list[], відповідно початковий та кінцевий елементи – start і end. Функція рекурсивно поділяє масив на дві частини, доки частини не будуть містити один елемент. Для контролю такої умови можна скористатися оператором if (start < end).

Після розділення виконується об’єднання частин масивів за допомогою спеціально створеної функції merge().

Алгоритм функції об’єднання merge() складається з таких етапів:

1. Циклічно проходимо по двох масивах.
2. У масив, що об'єднується, ставимо той елемент, що менший.
3. Рухаємось далі, доки не дійдемо до кінця обох масивів.

Функція merge має чотири параметри – масив list[] і відповідно перший start, кінцевий end і середній mid елементи.

В функції використовується допоміжний масив mergedList розмірність якого співпадає з розмірністю масиву list. В цей масив будемо вставляти елементи, що об’єднуються.

Також в роботу функції вводяться допоміжні змінні i= start, k = start, j = mid + 1.

В функції має здійснюватися циклічне порівняння елементів першої частини масиву із другою (використовувати цикл while доки i <mid і j < end).

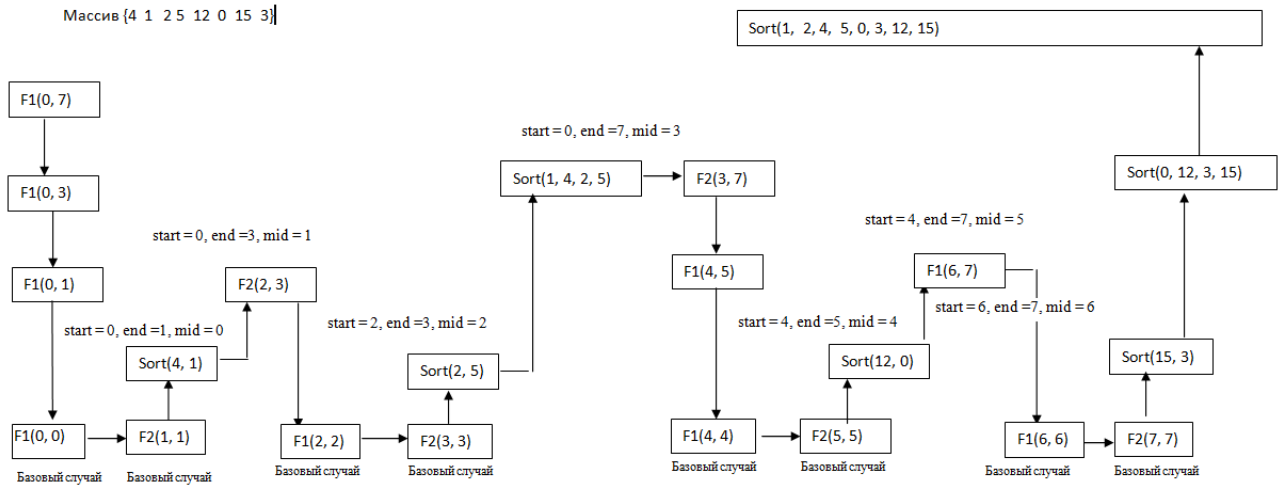

Рис. 1.2 – Послідовність роботи рекурсивних функцій і зміни їх параметрів в сортуванні злиттям для заданого масиву

Інструкція створення програми

1. Створіть функцію mergeSort, яка обчислює середину масиву mid, а потім викликає сама себе два рази. В першому випадку її параметрами мають бути початок масиву і обчислена середина , в другому виклику параметри – середина масиву і його кінець. Організуйте перед обчисленням середини масиву вивід елементів масиву на консоль. Дослідить як працює рекурсія на масиві з восьми елементів застосовуючи налагоджування програми. Подивиться скільки разів буде виконуватися функція mergeSort з параметрами початок-середина масиву, і коли програма буде виконувати функцію mergeSort за параметрами середина-кінець масиву.

2. В функції mergeSort після виконання двох функцій mergeSort, необхідно добавити функцію, яка виконує сортування і поєднання двох роздроблених масивів merge. Функція має наступні параметри: масив list[], порядковий номер початкового елементу масива start, кінцевий порядковий номер масиву end, номер середнього елементу масиву mid.

Функція має включати наступні допоміжні змінні:

Масив mergedList[], розмір якого дорівнює розміру основного масиву. В цей масив будуть записуватися елементи двох масивів у відсортованому вигляді. Лічильник k, який відслідковує порядковий номер елементу масиву. Початкове значення цієї змінної має дорівнювати start.

Змінна i, яка використовується для порядкових номерів першого масиву. Початкове значення цієї змінної має дорівнювати start.

Змінна j, яка використовується для порядкових номерів другого масиву. Початкове значення цієї змінної має дорівнювати mid + 1.

Функція має виконувати такі дії:

1. Доки і не досяг середнього значення і j не досяг кінцевого значення, будемо порівнювати два елементи двох масивів і присвоювати елементу масиву mergedList менше значення. Лічильники массиву mergedList і масиву, елементу, що менший, після цього збільшимо на одиницю.
2. Може виникнути ситуація, коли в одному із масивів лічильник не досягне свого кінцевого значення і необхідно розглянути дві ситуації: коли порядковий номер елементу першого масиву не досяг середнього значення. Тоді ці значення присвоюємо поточному елементу допоміжного масиву, і збільшуємо порядкові номери елементів обох масивів на одиницю. Аналогічну дію виконуємо і для випадку, коли порядковий номер елементу другого масиву не досяг кінцевого значення.
3. В кінці функції присвоюємо основному масиву list значення допоміжного mergedList в циклі цикл for.

Контрольні питання

1. В чому полягає основний принцип сортування злиттям?
2. Для чого в алгоритмі сортування злиттям використовується рекурсія?
3. Яким чином в програмі здійснюється поділ масиву?
4. Яким чином в програмі здійснюється злиття окремих частин масиву?

Лабораторна робота 5
Сортування купою
Загальні відомості

В цьому методі вихідний масив представляємо у вигляді структури даних купа. Купа – це один із типів бінарного дерева, яка має такі властивості:

1. Батьківський вузол завжди більше дочірніх;
2. На і-му шарі розташовано 2^i вузлів, починаючи з нуля. Тобто на нульовому шарі 1 вузол, на першому – 2 вузла, на другому – 4, тощо. Правило виконується для всіх шарів, крім останнього;
3. Шари заповнюються зліва направо.

Після формування купи витягуватимемо найстарший вузол (елемент із найбільшим значенням) і ставитимемо на кінець масиву.

В загальному вигляді алгоритм сортування купою може складатися із таких кроків:

1. Формуємо бінарне дерево із масиву.
2. Розставляємо вузли в дереві так, щоб вийшла купа (метод heapify ()).
3. Верхній елемент поміщаємо на кінець масиву.
4. Повертаємось на крок 2, поки купа не спорожніє.

Звертатися до дочірніх вузлів можна, знаючи, що дочірні елементи і-го елемента перебувають у позиціях 2*i + 1 (лівий вузол) і 2*i + 2 (правий вузол).

Приклад сортуванням купою

З наведеного масиву можна сформувати таке бінарне дерево.

Рис. 5. 1 Масив і сформоване з нього бінарне дерево

Індекс із нижнім лівим вузлом визначається за формулою n/2-1, де n – довжина масиву. Виходить 5/2 - 1 = 2 - 1 = 1. З цього індексу і починаємо операцію heapify(), яка розставляє вузли дерева. Порівняємо дочірні вузли 1-ї позиції, а саме 1 та 30.

Рис. 5.2 – Порівняння дочірніх вузлів 1-ої позиції

Так як значення дочірнього елементу виявилося більшим, то міняємо його місцем із батьківським.

Тепер проводимо перевірку батьківського вузла від позиції 1.

Рис. 5.3 – Обмін місцями дочірніх вузлів, і порівняння
дочірніх вузлів 0-ї позиції

Міняємо місцями елементи із значеннями 30 і 4.

Рис. 5.4 – Бінарне дерево після обміну місцями вузлі 30 та 4

Після зміни перевіряємо всі дочірні вузли опущеного елемента. Тобто для елемента, значення якого дорівнює 4 застосовуємо функцію heapify(). Оскільки 4 менше 21, міняємо ці два вузли місцями.

Рис. 5.5 – Бінарне дерево після обміну місцями вузлів 21 та 4

Тепер елемент з найбільшим значенням знаходиться зверху купи. Його необхідно поставити у кінець масиву на четверту позицію.

Рис. 5.6 – Зміщення елементу з найбільшим значенням на останнє місце

Тепер продовжимо сортувати купу, але ігноруємо останній елемент. Для цього можна зменшити довжину масиву на одиницю.

Рис. 5.7 – Бінарне дерево, розмір якого зменшений на одиницю
(без елементу з найбільшим значенням)

Повторюємо алгоритм сортування поки купа не стане пустою і отримаємо відсортований масив.

Рис. 5.8 – Відсортований масив і нове бінарне дерево, утворене для нього

Інструкція щодо створення програми

1. Створюємо функцію heapify(), яка виконує перевірку дочірніх позицій вибраного вузла. Функція має три параметри: масив list[], довжина масиву listlength, та обраний батьківський елемент root.

Вводимо допоміжні змінні: largest, якій присвоюємо значення root і змінні які позначають лівий і правий листи цієї гілки l і r. Значення цих двох змінних вираховуються відповідно за формулами 2*root + 1 і 2*root + 2 відповідно.

Далі необхідно порівняти дочірній лівий елемент із батьківським largest. Якщо значення дочірнього елементу більше за батьківське, то елементи треба поміняти місцями. Виконувати порівняння, якщо порядковий номер дочірнього елементу менший за довжину масиву listlength.

Далі аналогічне порівняння необхідно виконати для дочірнього правого елементу.

Далі при умові, що елемент largest не дорівнює root, необхідно поміняти ці елементи місцями та рекурсивно викликати функцію heapfy.

2. Створюємо функцію heapSort, яка буде циклічно викликати функцію створення купи heapfy. Функція heapSort має два параметри – масив list[] і довжину цього масиву listLength.

Функція містить два цикли for, які викликають функцію формування купи heapfy. Спочатку цикл виконується із вершини, що обчислюється за формулою listlength/2 -1 і зменшується циклічно на одиницю.

Другий цикл переносить найбільше значення у кінець массиву. Виконання циклу починається із вершини listLength – 1, а далі зменшується на одиницю, щоб обрізати кінець масиву. Переміщення елемента за найбільшим значенням, який знаходиться на вершині (нульовий елемент) в кінець масиву (елемент з номером і), здійснюється обміном місцями елементів з цими позиціями. Після обміну необхідно сформувати нове бінарне дерево з елементами меншими на одиницю.

Контрольні питання

1. Що представляє собою купа?
2. Яким чином формується купа?
3. Яким чином розміщуються елементи в купі?
4. Яким чином здійснюється сортування купою?
5. За якими формулами можна обчислити позиції дочірні елементи вузлів?
6. Яким чином застосовується рекурсія у сортування купою?

Лабораторна робота 6

Псевдовипадкові числа

Лінійний конгруентний метод

Генератор псевдовипадкових чисел (ГПВЧ, англ. pseudorandom number generator, PRNG) – алгоритм, що породжує послідовність чисел, елементи якої майже незалежні один від одного і підпорядковуються заданому розподілу (зазвичай дискретного рівномірного).

Для генерації випадкових чисел найчастіше застосовують такі методи: лінійний конгруентний метод, метод Фібоначчі із запізнюваннями, регістр зсуву з лінійним зворотним зв'язком, регістр зсуву з узагальненим зворотним зв'язком.

Важливими характеристиками ГПВЧ є:

Досить довгий період, що гарантує відсутність зациклювання послідовності в межах розв'язуваної задачі. Довжина періоду має бути математично доведена;

Ефективність – швидкість роботи алгоритму та малі витрати пам'яті;

Відтворюваність – можливість заново відтворити послідовність чисел, що була раніше згенерована, будь-яку кількість разів;

Портованість – однакове функціонування на різному устаткуванні та операційних системах;

Швидкість отримання наступного елемента послідовності чисел, при завданні поточного елемента, для будь-якої величини; це дозволяє розділяти послідовність на кілька потоків (послідовностей чисел).

Розглянемо лінійний конгруентний метод. Його суть полягає в обчисленні послідовності випадкових чисел за формулою

(1)

де m – модуль, відносно якого обчислюють остаток від ділення, $m \geq 2$;

a – множник $0 \leq a < m$

c – приріст $0 \leq c < m$.

m має бути досить великим числом, бажано, щоб його величина дорівнювала , де – довжина машинного слова.

Програмна реалізація методу

В нашому випадку скористуємося більш вдосконаленою формулою, що визначають конгруентний метод генерації псевдовипадкових чисел:

(2)

(3)

де – множники, їх значення бути відповідно рівними таким значенням {1.1, 5.2, 9.3, 4.4, 3.5, 7.6, 0.7, 2.8, 8.9, 6.0};

– значення інкременту, рівне 65

– отримані випадкові числа, в програмі необхідно задати початкове значення рівним 7, і наступне значення рівним 0.

– випадкові числа на інтервалі [0, 1).

Також необхідно задати кількість величин, які необхідно згенерувати.

Необхідно створити програмку за таким псевдокодом:

```
функція  dGet_A(int iNumber) //вибору числа з масиву
    взяти залишок від ділення iNumber  на 10
    задати масив чисел {1.1, 5.2, 9.3, 4.4, 3.5, 7.6, 0.7, 2.8, 8.9, 6.0}
….повернути  елемент масиву з номером iNumber
```

головна програма

```
    задати змінні: початкове значення послідовності, значеня інкременту,
    кількість змінних, які необхідно згенерувати
    цикл генерації псевдовипадкових чисел
        реалізувати формулу 2, із використанням функції dGet_A
        реалізувати формулу 3
        вивід на консоль сгенерованого числа
закінчення циклу.
```

Контрольні питання

1. Що представляє собою генератор псевдовипадкових чисел?
2. Які методи найчастіше застосовують для генерації псевдовипадкових чисел?
3. Які важливі характеристики мають генератори псевдовипадкових чисел?
4. За якою формулою обчислюється послідовність випадкових чисел при застосуванні лінійного конгруентного методу?

Лабораторна робота 7
Динамічне програмування

Динамічне програмування – це розбиття задачі на дрібні підзадачі із застосуванням рекурентних виразів. Такий підхід дозволяє уникнути використання рекурсій них функцій, що значно уповільнюють роботу програми при збільшенні даних. Прикладами динамічного програмування можуть бути задача про маршрут на прямокутному полі та задача про пошук найбільшої загальної послідовності.

Задача про маршрут на прямокутному полі

Дано прямокутне поле розміром m×n, кожна комірка якого має певну вагу, що задається якимось натуральним числом. Можна здійснювати кроки вправо та униз. Необхідно попасти з верхньої лівої комірки в праву нижню і знайти маршрут із максимальною вагою. Вага маршруту обчислюється як сума чисел всіх комірок, які були відвідані.

При розробці алгоритму, треба вважити, що вага комірки першого рядка (стовпця), окрім першого елемента, буде розраховуватися як сума ваги цього елемента і попереднього, тобто тут застосовується рекурентний вираз

Вага комірок з усіма номерами обчислюється як сума ваги поточної комірки і комірки з максимальною вагою, що взята з двох комірок, а саме на один рядок вище за поточну, і на один стовпець лівіше від поточної комірки.

Рекурентний вираз буде мати такий вигляд:

Приклад задачі про маршрут на прямокутному полі

Дано прямокутне поле. Розв’язати задачу про маршрут на прямокутному полі.

3	2	7	4	12
11	8	9	1	5
8	6	7	13	10
2	9	4	7	3

В результаті виконаних дій отримаємо таку матрицю

3	5	12	16	28
14	22	31	32	37
22	28	38	51	61
24	37	42	58	64

Задача про найбільшу загальну послідовність

Розглянемо два рядки (або числові послідовності) – A і B. Нехай перший рядок складається із n символів , другий рядок складається із m символів . Підпослідовністю даного рядка (послідовності) називається деяке підмножина символів вихідного рядка, що прямують у тому порядку, в якому вони йдуть у вихідному рядку, але не обов'язково підряд. Якщо в рядку n символів, то він матимє різні підпослідовності: кожен із n символів рядка може або входити, або не входити в будь-яку вибрану підпослідовність. Порожня підпослідовність не містить жодного елемента і є підпослідовністю будь-якого рядка.

Розглянемо завдання — для двох даних рядків знайти такий рядок найбільшої довжини, який був би підпослідовністю кожної з них. Наприклад, якщо A=«abcabaac», B=«baccbca» то рядки A і B мають загальну підпослідовність довжини 4, наприклад, «acba» або «acbc».

Це завдання можна вирішити перебором – наприклад, перебравши всі підпослідовності першого рядка і для кожного з них перевіривши, чи є вона підпослідовністю другого рядка. Але за допомогою динамічного програмування це завдання можна вирішити за складність .

Розглянемо останні символи даних рядків та . Якщо ці символи збігаються, то вони обов'язково увійдуть останніми символами і найбільшу загальну підпослідовність даних рядків. Тоді можна звести завдання знаходження найбільшої загальної підпослідовності для рядків і до завдання знаходження найбільшої загальної підпослідовності для рядків, отриманих відкиданням від даних рядків останнього символу, тобто для і . Потім до відповіді для «укорочених» рядків додамо останні (рівні) символи вихідних рядків (або) та отримаємо відповідь для вихідних рядків.

Якщо останні символи вихідних рядків не збігаються, то ці символи (та) неспроможні одночасно входити у найбільшу загальну підпослідовність, тому можна одне із них відкинути. Тоді завдання зводиться до знаходження найбільшої загальної підпослідовності для одного з двох випадків – для рядків і або для рядків і .

Ми навчилися зводити завдання знаходження найбільшої загальної підпослідовності двох рядків до меншого завдання – знаходження найбільшої загальної підпослідовності для рядків, отриманих відкиданням останніх символів від вихідних рядків, тобто префіксів вихідних рядків. Для подальшого вирішення завдання слідуватимемо принципу побудови рішення за допомогою динамічного програмування.

Заповнимо таблицю (двомірний масив) із значеннями найдовших послідовностей і таблицю напрямків. При цьому розмірність таблиці напрямків буде на одиницю меншою, за розмірність таблиці значень найдовших послідовностей. Таблиця напрямків надасть потім змогу відновити відповідь – безпосередньо найбільшу загальну послідовність.

Перший рядок і перший стовпчик таблиці із значеннями найдовших послідовностей мають бути заповненні нульовими значеннями.

Для цього скористуємося рекурентними співвідношеннями, які залежать від того, чи збігаються останні символи рядків послідовностей A і B. Якщо , то тоді значення комірки таблиці дорівнюватиме +1. В таблицю напрямків треба занести значення, яке позначає напрямок на комірку в якої обидва індекси менші на одиницю (стрілка направлена до лівого верхнього кута).

Якщо , то значення комірки буде рівним більшому числу , а стрілка в таблиці напрямків буде направлена вгору.

Якщо , то значення комірки буде рівним більшому числу , а стрілка буде направлена вліво.

Остання комірка заповненої таблиці значень найдовших послідовностей міститиме значення найдовшої послідовності.

Для відновлення значень найдовшої послідовності треба скористатися матрицею напрямків і вектором із значеннями однієї із послідовностей. Починати алгоритм треба із останньої комірки матриці. Якщо ця комірка містить позначення, що відповідає стрілці вліво чи вправо, необхідно здійснити перехід у відповідну комірку. Якщо комірка містить позначення, що відповідає стрілці в лівий верхній кут, то в вектор відповідей необхідно занести значення із матриці A (або И) із відповідним індексом і здійснити перехід у комірку, в якої індекси на 1 менші. Здійснення алгоритму відновлення послідовності бажано здійснювати циклічно за допомогою циклу while.

Контрольний приклад

Знайти загальну найбільшу послідовність для двох послідовностей X=«ABCABAAC» і Y= «BACCBCA».

Тут довжина найбільшої послідовності дорівнюватиме 4, а сама послідовність буде містити символи “ACBA” для номерів j= 1, 4, 5, 6

	j		0	1	2	3	4	5	6
i	y_j / x_i		B	A	C	C	B	C	A
			0	0	0	0	0	0	0
0	A	0	0↑	1↖	1←	1←	1←	1←	1↖
1	B	0	1↖	1↑	1↑	1↑	2↖	2←	2←
2	C	0	1↑	1↑	2↖	2←	2↑	3↖	3←
3	A	0	1↑	2↖	2↑	2↑	2↑	3↑	4↖
4	B	0	1↖	2↑	2↑	2↑	3↖	3↑	4↑
5	A	0	1↑	2↖	2↑	2↑	3↑	3↑	4↖
6	A	0	1↑	2↖	2↑	2↑	3↑	3↑	4↖
7	C	0	1↑	2↑	3↖	3↖	3↑	4↖	4↑

Контрольні питання

1. В чому полягає сутність динамічного програмування?
2. Наведіть приклади задач, які можна вирішити із застосуванням динамічного програмування.
3. В чому полягає суть задачі маршрут на прямокутному полі?
4. Яким чином розраховується маршрут із максимальною вагою?
5. Яким чином можна вирішити задачу про найбільшу загальну послідовність за допомогою динамічного програмування?
6. Яким чином будується таблиця напрямків?

Лабораторна робота № 9
Жадібні алгоритми. Задача про заявки

Жадібні алгоритми знаходять на кожному кроці оптимальний вибір, вважаючи що кінцеве вирішення буде оптимальним. Однак на практиці таке кінцеве рішення не буває завжди оптимальним, тому жадібні алгоритми вважаються приблизними, а не повноцінними. Але для великої кількості задач, такий підхід дійсно дає оптимальне рішення.

Класичною задачею, для вирішення якої застосовуються жадібні алгоритми, є задача про заявки.

Подана деяка кількість N заявок на проведення занять із заданим проміжком часу (тобто заданий час початку і закінчення кожного заняття) в одній тій самій аудиторії. Необхідно вибрати максимально сумісну за часом кількість заявок, тобто потрібно щоб час проведення занять в цій аудиторії не перекривався.

Алгоритм розв'язання задачі про вибір заявок.

Необхідно:

1. Упорядкувати заявки за зростанням часу закінчення.
2. Надати множині заявок (відповіді) значення порожньої множини.
3. До множини заявок відповіді включити першу (після упорядкування) заявку.
4. Знайти у переліку першу заявку, що починається не раніше закінчення останньої, включеної у відповідь.
5. Включити знайдену заявку до множини заявок (відповіді).
6. Два попередні кроки повторювати доти, поки при пошуку чергової заявки не буде досягнуто кінця переліку заявок.

Контрольний приклад

Надана множина заходів. Кожний захід позначається двома величинами – умовним часом початку заходу і закінчення заходу таким чином

{[6; 7], [3; 6], [1; 3], [3; 5], [2; 6], [1; 5], [5; 6], [4; 7], [2; 4] }

. Необхідно розподілити заходи так, щоб в одній аудиторії провести як можна більше заходів.

Відсортуємо всі інтервали часу проведення заходів по правому кінцю, тобто по часу закінчення заходів і відобразимо їх на рисунку у вигляді відрізків.

{[1; 3], [2; 4], [1; 5], [3; 5], [2; 6]. [3; 6], [5; 6], [4; 7], [6; 7]}

Обираємо інтервал часу, який закінчується раніше за всіх, а заходи, які починаються до часу його закінчення викреслюємо із списку.

Наступним інтервалом часу буде [3; 5].

На наступному етапі викреслюємо інтервали [3; 6] і [4; 7].

Отримаємо відповідь [1; 3], [3; 5], [5; 6], [6; 7]

Контрольні питання

1. В чому полягає суть жадібних алгоритмів?
2. Яким чином можна сформувати задачу про заявки?
3. Який алгоритм розв’язання задачі про вибір заявок?

Навчальне видання

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

для студентів
спеціальності 122 – Комп’ютерні науки

Укладач:
ЯГУП Катерина Валеріївна

Відповідальний за випуск (завідувач кафедри)_____
Роботу рекомендував до друку (експерт РВР) _____
Комп’ютерна верстка _____
Редактор _____

План 2023 р., поз. 266

Підп. до друку (дата підпису проректора)_____.
Гарнітура Times New Roman.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2