**ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ**

**“ТЕОРІЯ АЛГОРИТМІВ”**

**Лабораторна робота №1**

**Тема**: “Розробка алгоритмів. Сортування включенням”

**Мета:** створення алгоритму сортування включенням (insertion sort).

**Завдання:** Модифікувати алгоритм сортування включенням (insertion sort) для розв’язання наступної задачі.

Вхідні дані. Масив цілих чисел *A*. Числа в масиві можуть повторюватись.

Вихідні дані. Масив чисел *A*, відсортований наступним чином. Спочатку йдуть всі парні числа у порядку зростання, а потім всі непарні числа у порядку зменшення.

Приклад. Вхідний масив *A* = [30, 19, 9, 15, 55, 24, 3, 78, 46, 41]. Тоді вихідний масив повинен мати наступний вигляд: *A* = [24, 30, 46, 78, 55, 41, 19, 15, 9, 3].

Додатково звертаємо вашу увагу на те, що алгоритм розв’язку даної задачі повинен ґрунтуватись на методі сортування включенням.

**Лабораторна робота №2**

**Тема**: “Метод декомпозиції. Пошук інверсій”

**Мета:** за допомогою методу декомпозиції розробити алгоритм та знайдення числа степенів схожості вподобань (кількість інверсій).

**Завдання: За допомогою методу декомпозиції** розробити алгоритм, який буде розв’язувати наступну задачу.

Вхідні дані. Матриця *D* натуральних чисел розмірності *u**m,* де *u* — ці кількість користувачів, *m*

— кількість фільмів. Кожний елемент матриці *D*[*i*, *j*] вказує на позицію фільму *j* в списку вподобань користувача *i*. Іншим вхідним елементом є *x* — номер користувача, з яким будуть порівнюватись номери всіх інших користувачів.

Вихідні дані. Список з впорядкованих за зростанням другого елементу пар (*i*, *c*), де *i* — номер користувача, *c* — число, яке вказує на степінь схожості вподобань користувачів *x* та *c* (кількістьінверсій).

**Лабораторна робота №3**

**Тема**: “Метод швидкого сортування”

**Мета:** реалізація трьох модифікацій алгоритму швидкого сортування (Quick Sort) та порівняння їх швидкодії.

**Завдання:** Реалізувати три модифікації алгоритму швидкого сортування (Quick Sort) та порівняти їх швидкодію. Швидкість алгоритмів порівнюється на основі підрахунку кількості порівнянь елементів масиву під час роботи алгоритмів.

**Лабораторна робота №4**

**Тема**: “Піраміди”

**Мета:** визначення послідовності медіан для заданого вхідного масиву.

**Завдання:** В даній роботі необхідно розв'язати наступну задачу визначення послідовності медіан для заданого вхідного масиву. Нагадаємо, що медіаною для масиву називається елемент, який займає середнє положення у відсортованому масиві. Так, якщо кількість елементів у масиві непарна, то медіана одна та індекс її у відсортованому масиві визначається як [*n*/2] (де *n* — розмір вхідного масиву). Якщо кількість елементів у масиві парна, то медіан буде дві та їх індекси визначаються за формулами [*n*/2] та [*n*/2] + 1. Задача формулюється наступним чином. Нехай заданий вхідний масив *A* = [*x*1, ..., *xN*].

Припустимо, що елементи масиву поступають на вхід програми послідовно: в кожний момент часу розглядається новий елемент *xi*. Необхідно для кожного *i* (від 1 до *N*) визначити медіану підмасиву *A*' = [*x*1, ..., *xi*], тобто медіану для масиву елементів, які були отримані програмою на даний момент часу. Необхідно розв’язати цю задачу з використанням структури даних пірамід і таким чином, щоб кожна медіана визначалась за час O(log(*i*)).

**Лабораторна робота №5**

**Тема**: “Хеш-таблиці”

**Мета:** Хеш-таблиці (*hash-tables*) можуть використовуватись для збереження масивів даних, швидкого доступу, вставки та видалення елементів. За допомогою хеш-таблиць можна ефективно розв'язати наступну задачу. Нехай заданий масив чисел *A* та число *S*. Потрібно дізнатись, чи присутні в масиві *A* два числа, сума яких дорівнює *S*.

**Завдання:** В роботі необхідно реалізувати різні типи хеш-таблиць із використанням різних хеш-функцій для розв’язання наведеної вище задачі. При цьому потрібно порівняти ефективність різних підходів шляхом підрахунку кількості колізій для кожного типу хеш-функцій та хеш-таблиць.

**Лабораторна робота №6**

**Тема**: “Бінарні дерева пошуку”

**Мета:** перетворення вхідного бінарного дерева у бінарне дерево пошуку.Пошук сум послідовних вузлів в дереві.

Бінарні дерева пошуку (*binary search trees*) являють собою бінарні дерева, які мають наступну властивість: для кожного вузла *X* елементи, які знаходяться у лівому піддереві *X*, будуть мати значення менше за *Х*, а елементи у правому піддереві — більше за *X*.

**Завдання:**

1. Перетворити вхідне бінарне дерево у бінарне дерево пошуку.

На вхід подається деяке бінарне дерево, із фіксованою структурою (тобто зв’язками між вузлами, їх батьком та нащадками). Необхідно переписати значення вузлів дерева таким чином, щоби:

a) їх нові значення брались тільки з того набору, який присутній у вхідному дереві;

b) зберігалась внутрішня структура дерева (зв’язки між вузол-батько та вузол-нащадки).

2. Пошук сум послідовних вузлів в дереві

Після того, як вхідне дерево перетворене на бінарне дерево пошуку, необхідно розв’язати наступну задачу. Додатково задається деяке число S. В отриманому бінарному дереві пошуку необхідно знайти всі такі монотонні шляхи (які не обов'язково йдуть від кореня, але всі прямують згори вниз), що сума значень вузлів, які належать знайденим шляхам, дорівнює S.

**Лабораторна робота №7**

**Тема**: “Жадібні алгоритми”

**Мета:** реалізація жадібного алгоритму для задачі комівояжера.

Жадібні алгоритми дозволяють знаходити швидко розв’язок для більшості задач. Проте часто отриманий ними розв’язок не є оптимальним. Таким чином не для всіх задач гарантується існування коректного жадібного алгоритму, тобто такого, який буде знаходити оптимальний розв’язок задач.

**Завдання:** У даній роботі необхідно запропонувати жадібний алгоритм для задачі комівояжера.

Задача комівояжера формулюється для повного графу. Для зваженого повного графу *G* з *n* вершинами задані відстані між усіма парами вершин (*i*, *j*). Необхідно знайти найкоротший маршрут, який проходить через всі вершини графу та заходить в кожну вершину лише один раз.

В даній роботі розглядається симетричний варіант задачі, коли відстань від міста *i* до міста *j* дорівнює відстані від *j* до *і* (відстані між (*i*, *j*) та (*j*, *i*) рівні).

Задача комівояжера відноситься до NP-повних задач і для неї не існує оптимального алгоритму, який би працював за поліноміальний час. Тому для її розв’язання часто використовуються евристичні алгоритми, до яких відносяться і жадібні алгоритми.

Вам необхідно запропонувати ідею жадібного алгоритму та перевірити його роботу на декількох екземплярах задач.

**Лабораторна робота №8**

**Тема**: “Динамічне програмування”

**Мета:** реалізація алгоритму динамічного програмування для задачі про рюкзак.

Динамічне програмування — це досить потужний метод розробки алгоритмів для розв'язання задач, які можна розкласти на задачі меншої розмірності, з яких, зрештою, отримується розв'язок початкової задачі.

**Завдання:** Застосування динамічного програмування для задачі про рюкзак є одним з яскравих прикладів потужності цього підходу. Дана задача формулюється наступним чином:

Дано *n* різних предметів, про які відомі їх розмір, або вага, *wi* та вартість *vi*. Є рюкзак, в який необхідно покласти ці предмети. Для рюкзака відома його місткість (сумарний розмір, або вага, предметів, що можуть бути розміщені у рюкзаку) — *W*. Необхідно відібрати таку множину предметів *S* серед усіх заданих предметів, що (1) їх сумарна розмірність не перевищує місткість рюкзака *W* та (2) сумарна вартість предметів в множині *S* є максимально можливою серед усіх інших множин.