**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №6**

**з навчальної дисципліни “****Рекурсивні алгоритми”**

**Тема:**

**Розріджені матриці.**

**Виконав студент групи ТР–12**

Каркушевський Владислав

Лабораторну роботу захищено

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Перевірив доцент кафедри**

\_\_\_\_\_\_\_\_\_\_\_ Андрій ОНИСЬКО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 року

**Київ 2022**

**Мета:** Метою лабораторної роботи є створення економного способу зберігання в пам’яті розріджених матриць

**Теоретична частина:**

Розріджена матриця - матриця, в якій кількість ненульових елементів менше ніж нульових.

Зберігати цілу матрицю у пам'яті комп'ютера є неефективно по відношенню до пам'яті, тому є альтернативні способи збереження розріджених матриць.

**• Зберігання ненульових елементів в масиві**

Одним з таких способів полягає в зберіганні ненульових елементів та їх координат. Цей спосіб є економний для пам'яті але для виконання дій з матрицями (додавання, множення) він є неефективний, оскільки кожного разу потрібно перебирати всі елементи для пошуку відповідного елемента.

**• Зберігання ненульових елементів в списках**

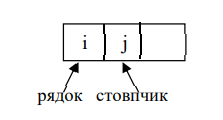
У цьому способі збереження кожен ненульовий елемент зберігається у вигляді значення, номера рядка та стовпця і вказівника на наступний елемент в рядку і стовпчику. Для цього методу збереження потрібно також зберігати рамку, яка складається з таких самих елементів, до якої ми будемо прив'язувати всі елементи вказівниками. Цей спосіб потребує більше пам'яті, але при цьому збільшується швидкість виконання дій над матрицями.

Всі подання, у тому числі і подання розріджених матриць складаються з дескриптора і тіла

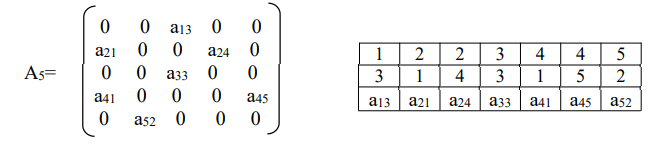
Дескриптор містить загальні властивості і додаткові відомості для оптимальної реалізації об'єкту.

У лабораторній буде використовуватися один із багатьох методів зберігання розріджених матриць.

Спосіб 1: Упакований спосіб зберігання розріджених матриць. Зберігаються всі ненульові елементи



Так, наприклад, значення елементів матриці A5 можна зберігати у тривимірному масиві. 1-й рядок масиву зберігає номер рядка ненульового елемента, 2-й рядок зберігає номер стовпчика, а 3-й – саме значення



Якщо τ - кількість ненульові символів, то необхідно 3τ комірок. При збереженні в упакованому вигляді розрідженої матриці необхідно передбачити можливість додавання нових ненульових елементів.

**Завдання**

Виконати індивідуальне завдання над стисненою матрицею.

Вивести матрицю до та після обробки у стисненому та розгорнутому вигляді.



**Код програми:**

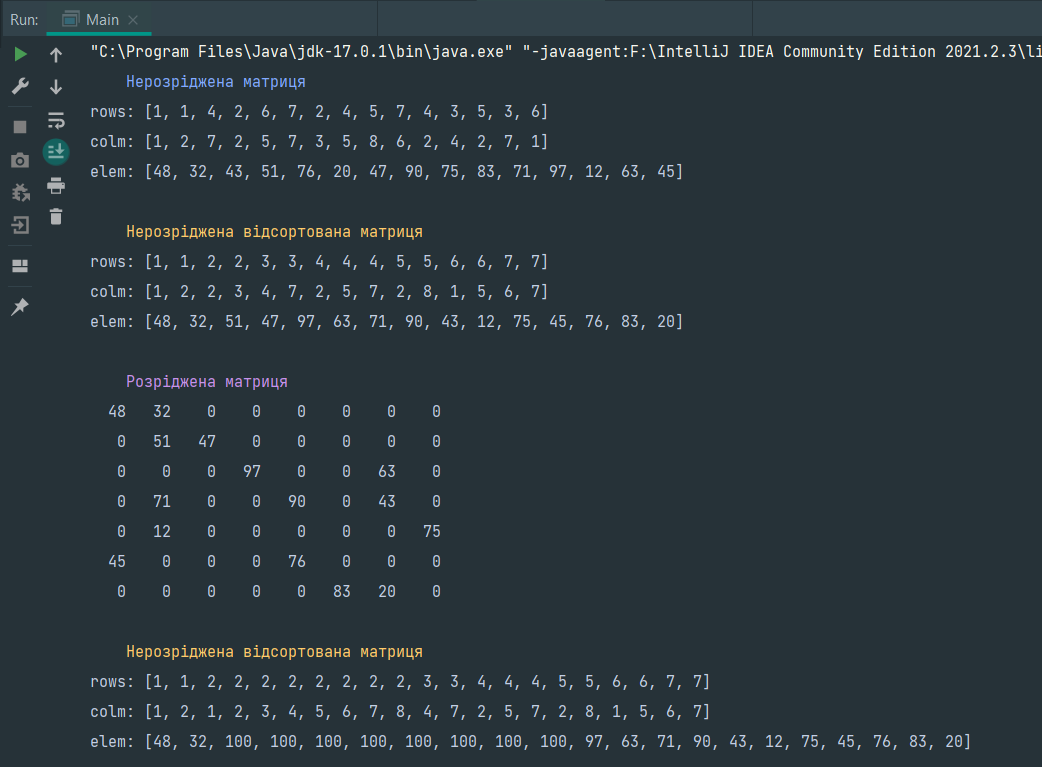
**DilutedMatrix**

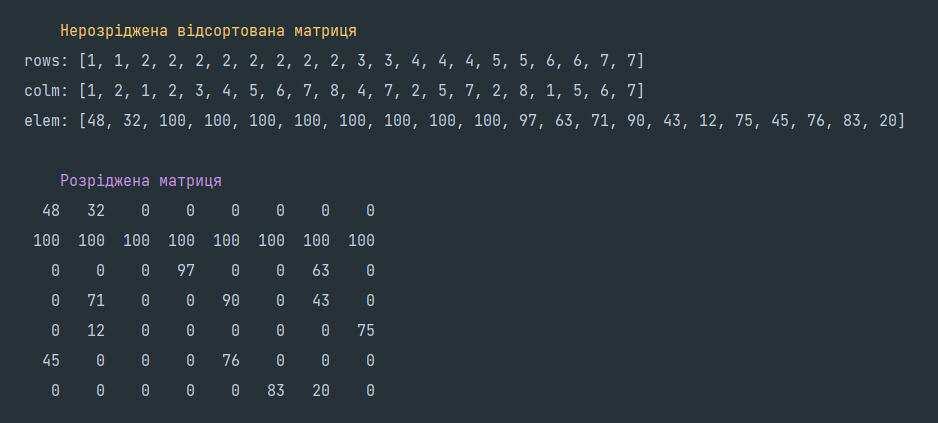
*public class* DilutedMatrix {  
  
 *private int* size;  
 *private int* k = 0;  
  
 *private int*[][] dilutedMatrix;*//нерозріджена матриця  
 private int*[][] res;*//розріджена матриця  
  
  
 public* DilutedMatrix(*int* size) {*//створимо конструктор(в параметр будемо передавати кількість ненульових елементів)  
 this*.size = size;  
 dilutedMatrix = *new int*[3][size];  
 }  
  
 *public void* addElem(*int* i, *int* j, *int* elem) {*//створимо метод для додавання елемента до нерозрідженої матриці  
 int* l = 0;  
 dilutedMatrix[l][k] = i;  
 l++;  
 dilutedMatrix[l][k] = j;  
 l++;  
 dilutedMatrix[l][k] = elem;  
 *this*.k++;  
 }  
  
  
 *public void* task() {*//створимо метод з завданням  
  
 for* (*int* i = 0; i < res.length; i++) {*//спочатку додамо до розрідженої матриці до другого рядку елементи зі значенням 100  
 for* (*int* j = 0; j < res[0].length; j++) {  
 *if* (i == 1) {  
 res[i][j] = 100;  
 }  
 }  
 }  
  
  
 *int* p = 0;  
 *for* (*int* i = 0; i < dilutedMatrix[0].length; i++) {  
 *if* (dilutedMatrix[0][i] == 2) {  
 p++;  
 }  
  
 }  
  
 *this*.size = size + res[0].length - p;*//змінимо розмір нерозрідженої матриці* dilutedMatrix = *new int*[3][size];  
  
 *this*.k = 0;  
  
 *for* (*int* i = 0; i < res.length; i++) {  
 *for* (*int* j = 0; j < res[0].length; j++) {  
 *if* (res[i][j] != 0) {  
 addElem(i + 1, j + 1, res[i][j]);  
 }  
 }  
 }  
  
 sort();  
  
 }  
  
 *public void* sort() {*//метод для сортування нерозрідженої матриці  
  
 boolean* needIteration = *true*;  
 *while* (needIteration) {  
 needIteration = *false*;  
 *for* (*int* i = 0; i < dilutedMatrix[0].length - 1; i++) {  
 *if* (dilutedMatrix[0][i] > dilutedMatrix[0][i + 1]) {  
  
 set(i, i + 1);  
 needIteration = *true*;  
  
 }  
 }  
 }  
  
 needIteration = *true*;  
 *while* (needIteration) {  
 needIteration = *false*;  
 *for* (*int* i = 0; i < dilutedMatrix[0].length - 1; i++) {  
 *if* (dilutedMatrix[0][i] == dilutedMatrix[0][i + 1]) {  
 *if* (dilutedMatrix[1][i] > dilutedMatrix[1][i + 1]) {  
  
 set(i, i + 1);  
 needIteration = *true*;  
  
 }  
 }  
 }  
 }  
  
  
 }  
  
 *public void* set(*int* i, *int* j) {*//метод для зміни елементів нерозрідженої матриці  
  
 for* (*int* l = 0; l < 3; l++) {  
  
 *int* pr = dilutedMatrix[l][i];  
 dilutedMatrix[l][i] = dilutedMatrix[l][j];  
 dilutedMatrix[l][j] = pr;  
 }  
  
  
 }  
  
 *public int*[][] makeMatrix() {*//метод для створення розрідженої матриці* sort();  
  
 *int* rowMax = dilutedMatrix[0][0];  
 *for* (*int* i = 1; i < dilutedMatrix[0].length; i++) {  
  
 *if* (dilutedMatrix[0][i] > rowMax) {  
 rowMax = dilutedMatrix[0][i];  
 }  
 }  
  
 *int* colmMax = dilutedMatrix[1][0];  
 *for* (*int* i = 1; i < dilutedMatrix[0].length; i++) {  
 *if* (dilutedMatrix[1][i] > colmMax) {  
 colmMax = dilutedMatrix[1][i];  
 }  
 }  
  
 *int*[][] res = *new int*[rowMax][colmMax];  
  
  
 *int* row = dilutedMatrix[0][0] - 1;  
 *int* colm = dilutedMatrix[1][0] - 1;  
 *int* k = 0;  
 *for* (*int* i = 0; i < res.length; i++) {  
  
 *for* (*int* j = 0; j < res[0].length; j++) {  
  
 *if* (k != size) {  
 row = dilutedMatrix[0][k] - 1;  
 colm = dilutedMatrix[1][k] - 1;  
 }  
  
 *if* (i == row && j == colm) {  
 res[i][j] = dilutedMatrix[2][k];  
 k++;  
 } *else* {  
 res[i][j] = 0;  
 }  
  
 }  
 }  
  
 *this*.res = res;  
  
 *return* res;  
  
 }  
  
 @Override  
 *public* String toString() {*//Перевизначимо метод toString для виведення нерозрідженої матриці  
  
  
 if* (size == 0) *return* "matrix is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("rows: [");  
 *for* (*int* i = 0; i < dilutedMatrix[0].length - 1; i++) {  
 sb.append(dilutedMatrix[0][i]).append(", ");  
 }  
 sb.append(dilutedMatrix[0][dilutedMatrix[0].length - 1]).append("]\n");  
  
 sb.append("colm: [");  
 *for* (*int* i = 0; i < dilutedMatrix[0].length - 1; i++) {  
 sb.append(dilutedMatrix[1][i]).append(", ");  
 }  
 sb.append(dilutedMatrix[1][dilutedMatrix[0].length - 1]).append("]\n");  
  
 sb.append("elem: [");  
 *for* (*int* i = 0; i < dilutedMatrix[0].length - 1; i++) {  
 sb.append(dilutedMatrix[2][i]).append(", ");  
 }  
 sb.append(dilutedMatrix[2][dilutedMatrix[0].length - 1]).append("]\n");  
  
  
 *return* sb.toString();  
 }  
}

**Main**

*public class* Main {  
  
 *public static void* main(String[] args) {  
   
 DilutedMatrix table = *new* DilutedMatrix(15);*//створимо нерозріджену матрицю з 11 ненульовими елементами* table.addElem(1, 1, 48);*//додамо елементи* table.addElem(1, 2, 32);  
 table.addElem(4, 7, 43);  
 table.addElem(2, 2, 51);  
 table.addElem(6, 5, 76);  
 table.addElem(7, 7, 20);  
 table.addElem(2, 3, 47);  
 table.addElem(4, 5, 90);  
 table.addElem(5, 8, 75);  
 table.addElem(7, 6, 83);  
 table.addElem(4, 2, 71);  
 table.addElem(3, 4, 97);  
 table.addElem(5, 2, 12);  
 table.addElem(3, 7, 63);  
 table.addElem(6, 1, 45);  
  
 System.***out***.print("\u001B[34m");  
 System.***out***.println("\tНерозріджена матриця");  
 System.***out***.print("\u001B[0m");  
 System.***out***.println(table);*//виведемо нерозріджену матрицю* table.sort();*//відсортуємо нерозріджену матрицю* System.***out***.print("\u001B[33m");  
 System.***out***.println("\tНерозріджена відсортована матриця");  
 System.***out***.print("\u001B[0m");  
 System.***out***.println(table);*//виведемо нерозріджену матрицю  
  
 int*[][] res = table.makeMatrix();*//створимо розріджену матрицю* System.***out***.print("\u001B[35m");  
 System.***out***.println("\tРозріджена матриця");  
 System.***out***.print("\u001B[0m");  
 *outputMatrix*(res);*//виведемо розріджену матрицю* table.task();*//викличемо метод із завданням згідно з варіантом* System.***out***.print("\u001B[33m");  
 System.***out***.println("\tНерозріджена відсортована матриця");  
 System.***out***.print("\u001B[0m");  
 System.***out***.println(table);*//виведемо нерозріджену матрицю* System.***out***.print("\u001B[35m");  
 System.***out***.println("\tРозріджена матриця");  
 System.***out***.print("\u001B[0m");  
 *outputMatrix*(res);*//виведемо розріджену матрицю* }  
  
 *public static void* outputMatrix(*int*[][] b) {*//створюємо метод для виведення матриці  
  
 for* (*int*[] ints : b) {  
 *for* (*int* j = 0; j < b[0].length; j++) {  
 System.***out***.printf("%4d ", ints[j]);  
 }  
 System.***out***.println();  
 }  
 System.***out***.println();  
 }  
  
}

**Результат виконання програми:**

****

****

**Висновок:**

На цій лабораторній роботі було розроблено спосіб економного зберігання в пам’яті розріджених матриць. Написана структура для економного зберігання в пам’яті розрідженої матриці на основі масиву. Виконано завдання згідно варіанту.. Досліджено результат на правильність.