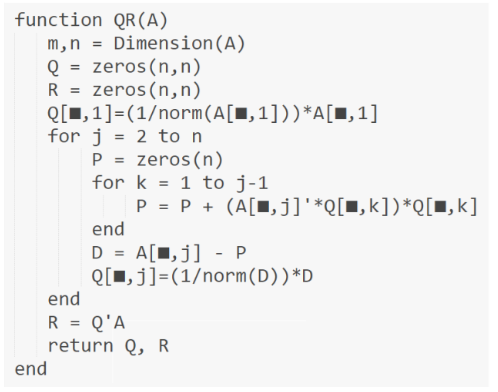
Realizar un programa Java serial y paralelo (*usando hilos explícitos creados e instanciados en el programa*) para solucionar el problema de la descomposición matricial QR de una matriz cuadrada NxN (). Verificar que los resultados sean equivalentes y comparar los tiempos de ejecución.

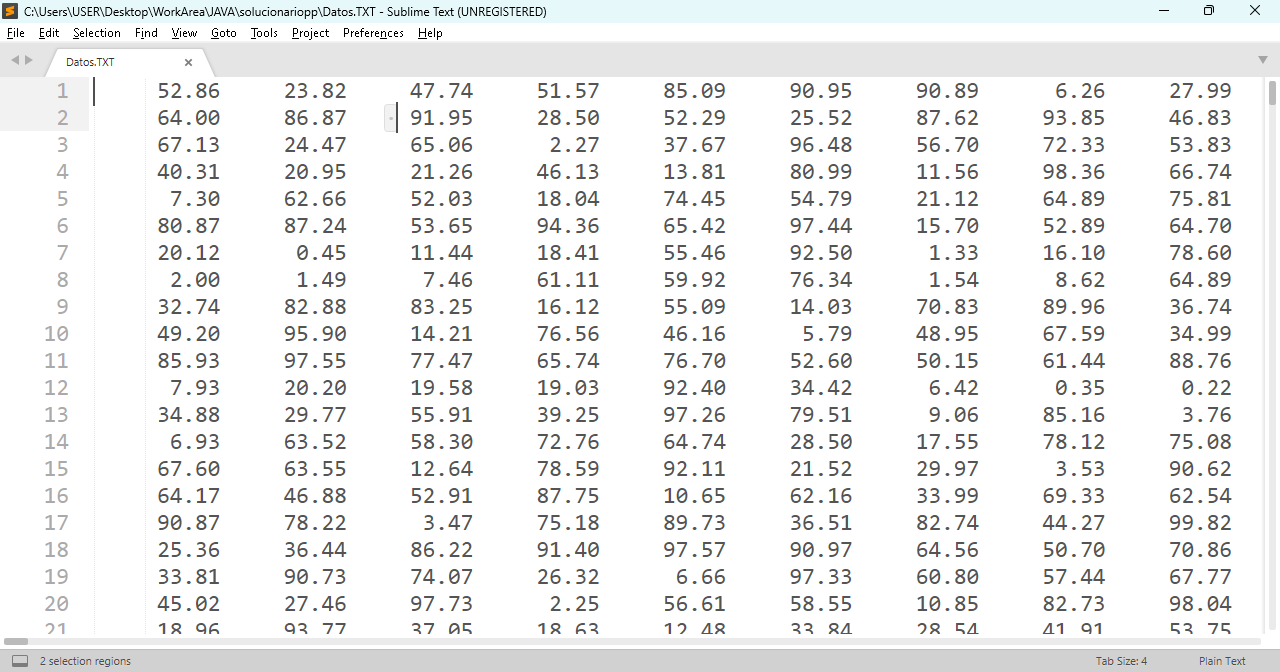


CLASE BASE MATRIZ.JAVA

| public class Matrix {  private int ROWS;  private int COLS;  private double[][] M;  //------------------------------------------------------  //------------------------------------------------------  public Matrix(double[][] M) {  this.M = M;  this.ROWS = M.length;  this.COLS = M[0].length;  }  //------------------------------------------------------  //------------------------------------------------------  public double GetCell(int i, int j){  return M[i][j];  }  //------------------------------------------------------  //------------------------------------------------------  public void SetCell(int i, int j, double val){  M[i][j] = val;  }  //------------------------------------------------------  //------------------------------------------------------  public int getRows() {  return ROWS;  }  //------------------------------------------------------  //------------------------------------------------------  public int getCols() {  return COLS;  }  //------------------------------------------------------  //------------------------------------------------------  void imprimir() {  for (int i = 0; i < ROWS; i++) {  for (int j = 0; j < COLS; j++) {  System.out.printf("%12.2f", M[i][j]);  }  System.out.println();  }  }  //------------------------------------------------------  //------------------------------------------------------  public synchronized void incrementar(int fil, int col, double cant) {  M[fil][col] = M[fil][col] + cant;  }  //------------------------------------------------------  //------------------------------------------------------  public double prodEsc(int col1, int col2, int filIni, int filFin) {  double resp = 0;  for (int k = filIni; k <= filFin; k++) {  resp += (M[k][col1] \* M[k][col2]);  }  return resp;  }  //------------------------------------------------------  //------------------------------------------------------  public double prodEsc(int col1, int col2) {  double resp = 0;  for (int k = 0; k < ROWS; k++) {  resp += (M[k][col1] \* M[k][col2]);  }  return resp;  }  //------------------------------------------------------  //------------------------------------------------------  Matrix prod(Matrix B) {  int col = B.getCols();  Matrix C = new Matrix(new double[ROWS][col]);  for (int i = 0; i < ROWS; i++) {  for (int j = 0; j < col; j++) {  for (int k = 0; k < COLS; k++) {  double x = GetCell(i, k) \* B.GetCell(k, j);  C.SetCell(i, j, C.GetCell(i, j) + x);  }  }  }  return C;  }  } |
| --- |

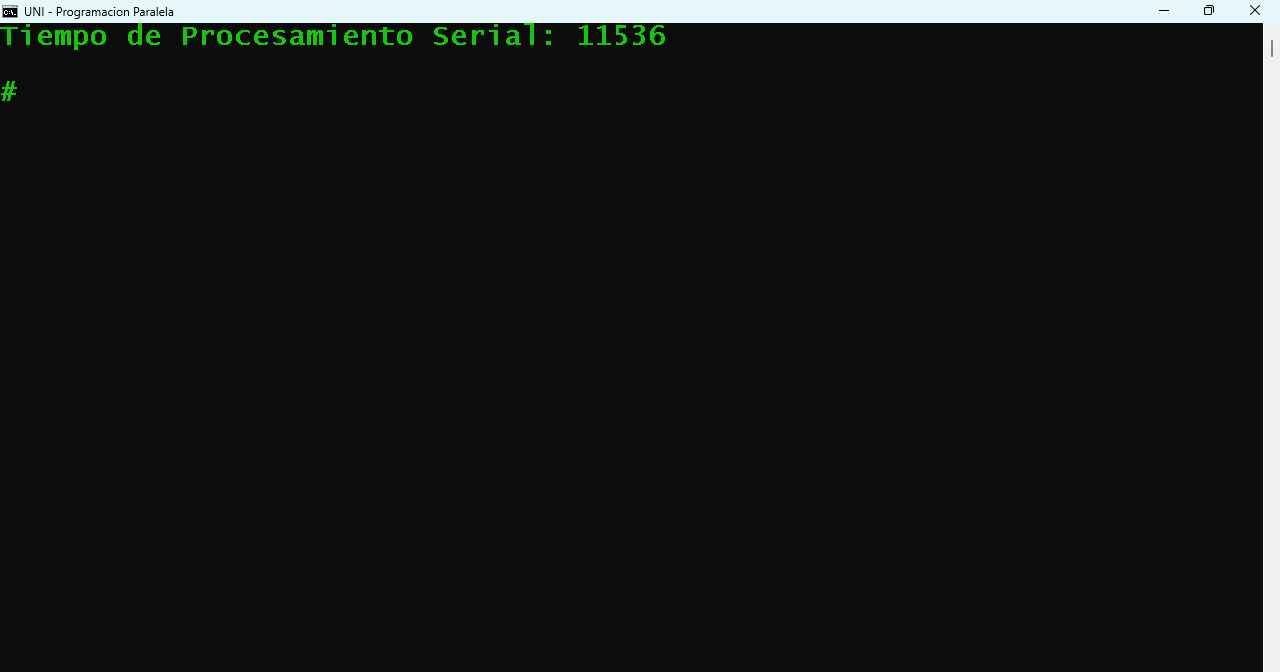
PROGRAMA QUE GENERA LOS DATOS

| import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.PrintWriter;  import java.nio.file.Path;  import java.nio.file.Paths;  import java.util.Random;  import java.util.Scanner;  //------------------------------------------------------  public class DataSet {  static int filas = 1000;  static int columnas = 1000;  static String DATAFILE = "Datos.TXT";  //------------------------------------------------------  public static void CreateFile(int M, int N) {  Random random = new Random();  PrintWriter pw = null;  try {  pw = new PrintWriter(DATAFILE);  for (int i = 0; i < M; i++) {  for (int j = 0; j < N; j++) {  double valor = random.nextDouble() \* 100;  pw.printf("%10.2f", valor);  }  pw.println("");  }  } catch (FileNotFoundException ex) {  System.out.println(ex.getMessage());  } finally {  pw.close();  }  }  //------------------------------------------------------  public static double[][] ReadFile(int M, int N) {  double[][] MTX = new double[M][N];  Path filePath = Paths.get(DATAFILE);  Scanner scanner = null;  try {  scanner = new Scanner(filePath);  }  catch (IOException ex) {  System.out.println(ex.getMessage());  }  int i = 0, j = 0;  while (scanner.hasNext()) {  if (scanner.hasNextDouble()) {  MTX[i][j] = scanner.nextDouble();  j++;  if (j == N) {  j = 0;  i++;  }  }  else {  scanner.next();  }  }  return MTX;  }  //------------------------------------------------------  public static void WriteFile(Matrix M, String archivo){  int mm = M.getRows();  int nn = M.getCols();  PrintWriter pw = null;  try {  pw = new PrintWriter(archivo);  for (int i = 0; i < mm; i++) {  for (int j = 0; j < nn; j++) {  double valor = M.GetCell(i, j);  pw.printf("%10.2f", valor);  }  pw.println("");  }  }  catch (FileNotFoundException ex) {  System.out.println(ex.getMessage());  }  finally {  pw.close();  }  }  //------------------------------------------------------  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  CreateFile(filas, columnas);  }  } |
| --- |



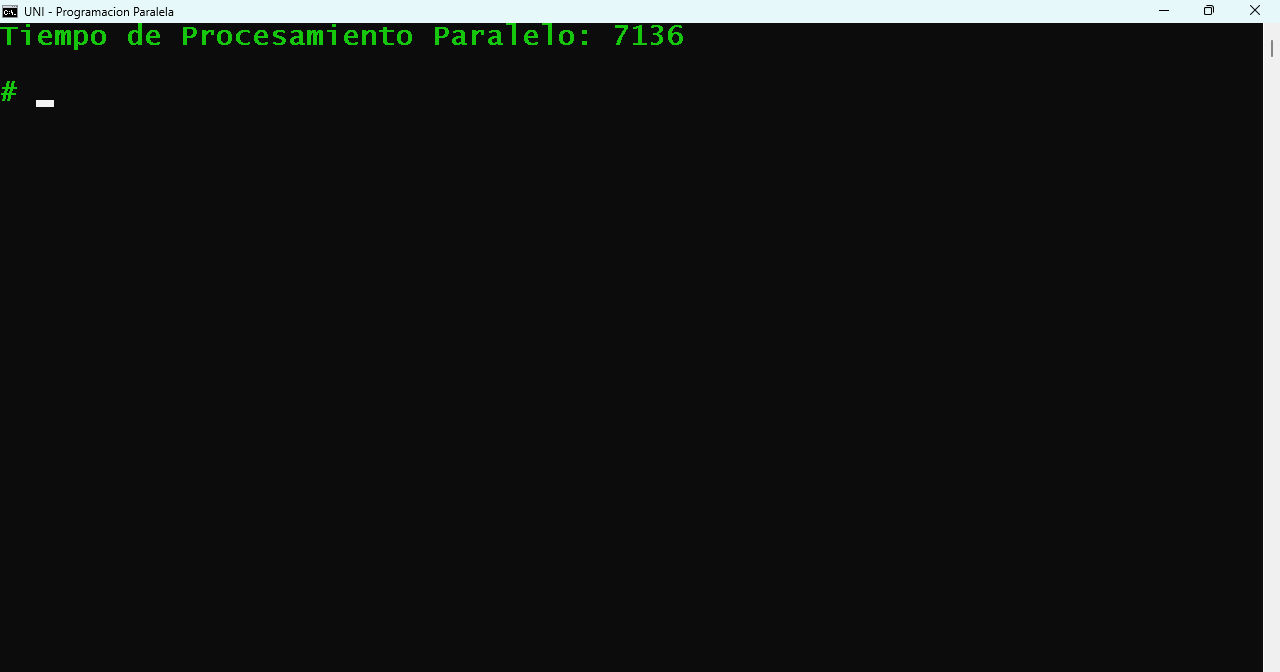
PROCESAMIENTO SERIAL

| public class QR\_Serial {  //--------------------------------------------------------------  //--------------------------------------------------------------  public static void Process\_Serial() {  long Time1,Time2;  double x;  double[][] A1;  A1 = DataSet.ReadFile(DataSet.filas, DataSet.columnas);  Time1 = System.currentTimeMillis();  Matrix A = new Matrix(A1);  int filas = A.getRows();  int columnas = A.getCols();  Matrix R = new Matrix(new double[columnas][columnas]);  for (int i = 0; i < columnas; i++) {  for (int j = i; j < columnas; j++) {  x = A.prodEsc(i, j) / Math.sqrt(A.prodEsc(i, i));  R.SetCell(i, j, x);  }  for (int k = 0; k < filas; k++) {  x = A.GetCell(k, i) / R.GetCell(i, i);  A.SetCell(k, i, x);  }  for (int j = i + 1; j < columnas; j++) {  for (int k = 0; k < filas; k++) {  x = A.GetCell(k, j) - R.GetCell(i, j) \* A.GetCell(k, i);  A.SetCell(k, j, x);  }  }  }  DataSet.WriteFile(A, "Matriz Q - Serial");  DataSet.WriteFile(R, "Matriz R - Serial");  Time2 = System.currentTimeMillis();  System.out.printf("Tiempo de Procesamiento Serial: %d%n", (Time2-Time1));  }  //--------------------------------------------------------------  //--------------------------------------------------------------  public static void main(String[] args) {  Process\_Serial();  }  //--------------------------------------------------------------  //--------------------------------------------------------------  }//CLASS |
| --- |

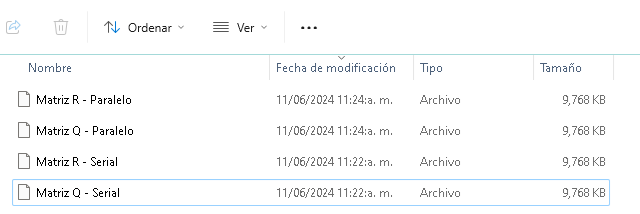


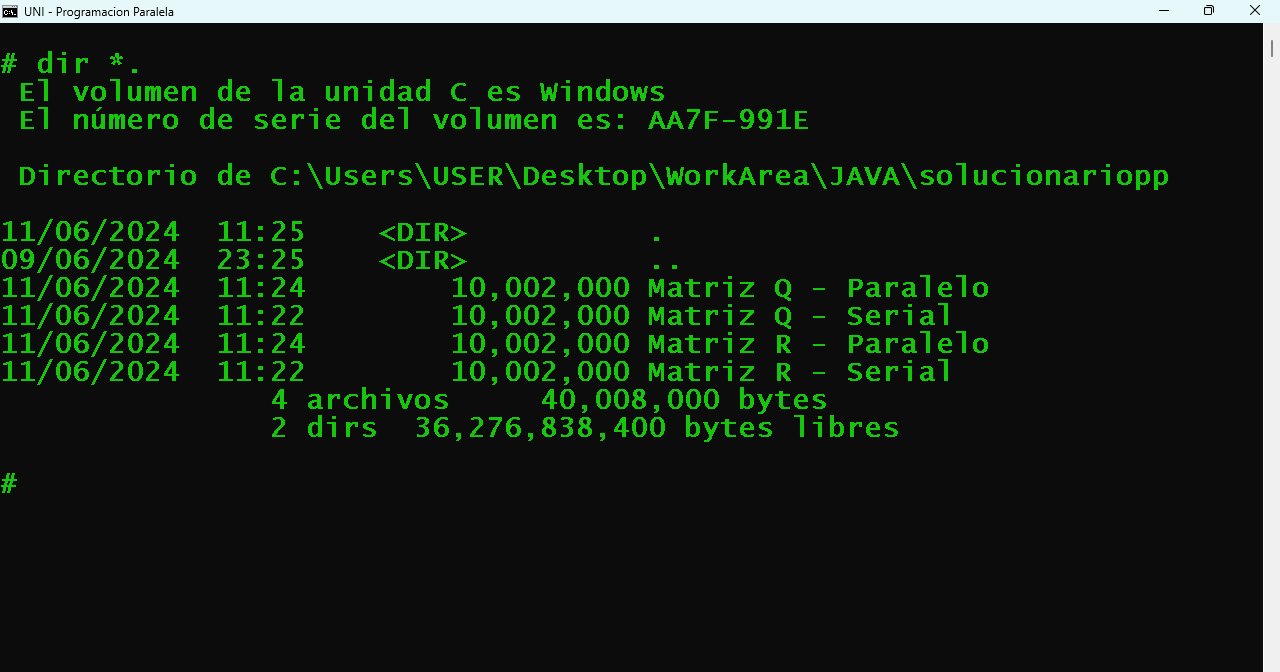
PROCESAMIENTO PARALELO

| public class QR\_Paralelo {  public static final int HILOS = 2;  //--------------------------------------------------------------  public static void main(String[] args) {  long Time1,Time2;  double[][] A1 = DataSet.ReadFile(DataSet.filas, DataSet.columnas);  Time1 = System.currentTimeMillis();  Matrix A = new Matrix(A1);  Matrix R = new Matrix(new double[A.getCols()][A.getCols()]);  for (int i = 0; i < A.getCols(); i++) {  Thread[] threads = new Thread[HILOS];  for (int t = 0; t < HILOS; t++) {  threads[t] = new Thread(new Oper1(t + 1, A, R, i));  threads[t].start();  }  try {  for (int t = 0; t < HILOS; t++) {  threads[t].join();  }  } catch (InterruptedException e) {  System.out.println(e.getMessage());  }  double x = Math.sqrt(R.GetCell(i, i));  for (int j = i; j < A.getCols(); j++) {  R.SetCell(i, j, R.GetCell(i, j) / x);  }  for (int t = 0; t < HILOS; t++) {  threads[t] = new Thread(new Oper2(t + 1, A, R, i));  threads[t].start();  }  try {  for (int t = 0; t < HILOS; t++) {  threads[t].join();  }  } catch (InterruptedException e) {  System.out.println(e.getMessage());  }  for (int t = 0; t < HILOS; t++) {  threads[t] = new Thread(new Oper3(t + 1, A, R, i));  threads[t].start();  }  try {  for (int t = 0; t < HILOS; t++) {  threads[t].join();  }  } catch (InterruptedException e) {  System.out.println(e.getMessage());  }  }  DataSet.WriteFile(A, "Matriz Q - Paralelo");  DataSet.WriteFile(R, "Matriz R - Paralelo");  Time2 = System.currentTimeMillis();  System.out.printf("Tiempo de Procesamiento Paralelo: %d%n", (Time2-Time1));  }  }  class Oper1 implements Runnable {  private Matrix A;  private Matrix R;  private int filIni;  private int filFin;  private int i;  public Oper1(int nroHilo, Matrix A, Matrix R, int i) {  this.A = A;  this.R = R;  filIni = (nroHilo - 1) \* A.getRows() / QR\_Paralelo.HILOS;  filFin = nroHilo \* A.getRows() / QR\_Paralelo.HILOS - 1;  this.i = i;  }  @Override  public void run() {  for (int j = i; j < A.getCols(); j++) {  R.incrementar(i, j, A.prodEsc(i, j, filIni, filFin));  }  }  }  class Oper2 implements Runnable {  private Matrix A;  private Matrix R;  private int filIni;  private int filFin;  private int i;  public Oper2(int nroHilo, Matrix A, Matrix R, int i) {  this.A = A;  this.R = R;  filIni = (nroHilo - 1) \* A.getRows() / QR\_Paralelo.HILOS;  filFin = nroHilo \* A.getRows() / QR\_Paralelo.HILOS - 1;  this.i = i;  }  @Override  public void run() {  double x = R.GetCell(i, i);  for (int k = filIni; k <= filFin; k++) {  A.SetCell(k, i, A.GetCell(k, i) / x);  }  }  }  class Oper3 implements Runnable {  private Matrix A;  private Matrix R;  private int filIni;  private int filFin;  private int i;  public Oper3(int nroHilo, Matrix A, Matrix R, int i) {  this.A = A;  this.R = R;  filIni = (nroHilo - 1) \* A.getRows() / QR\_Paralelo.HILOS;  filFin = nroHilo \* A.getRows() / QR\_Paralelo.HILOS - 1;  this.i = i;  }  @Override  public void run() {  for (int j = i + 1; j < A.getCols(); j++) {  double x = R.GetCell(i, j);  for (int k = filIni; k <= filFin; k++) {  A.SetCell(k, j, A.GetCell(k, j) - x \* A.GetCell(k, i));  }  }  }  } |
| --- |

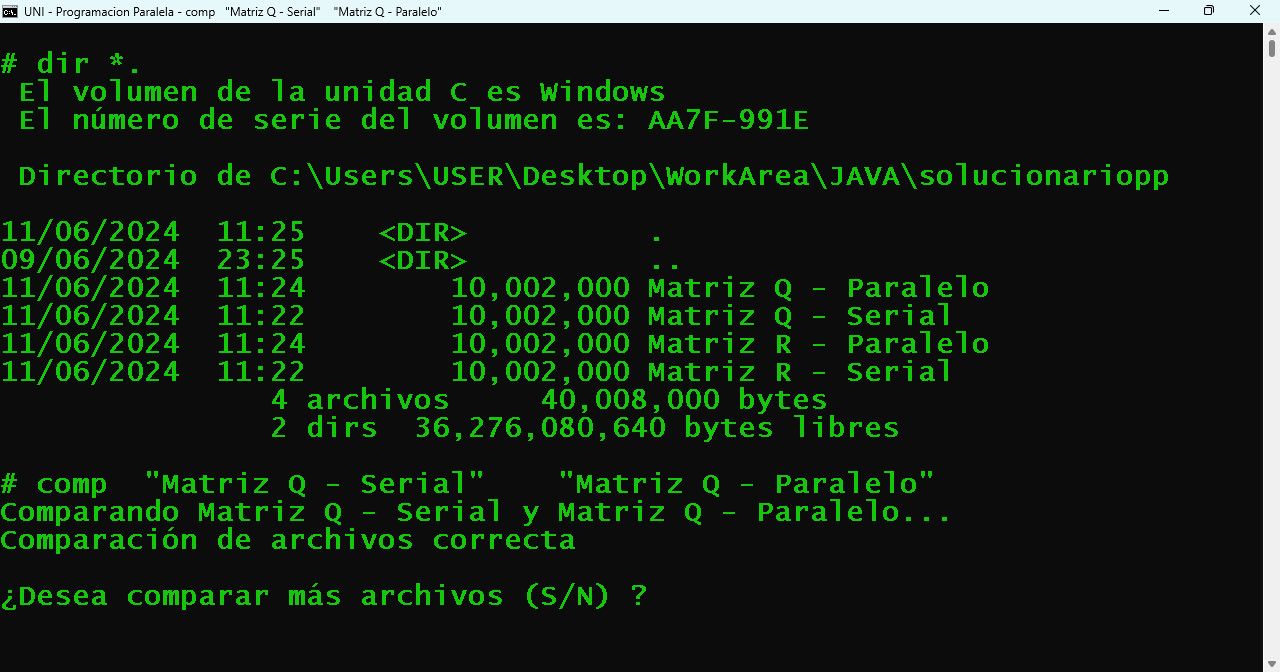


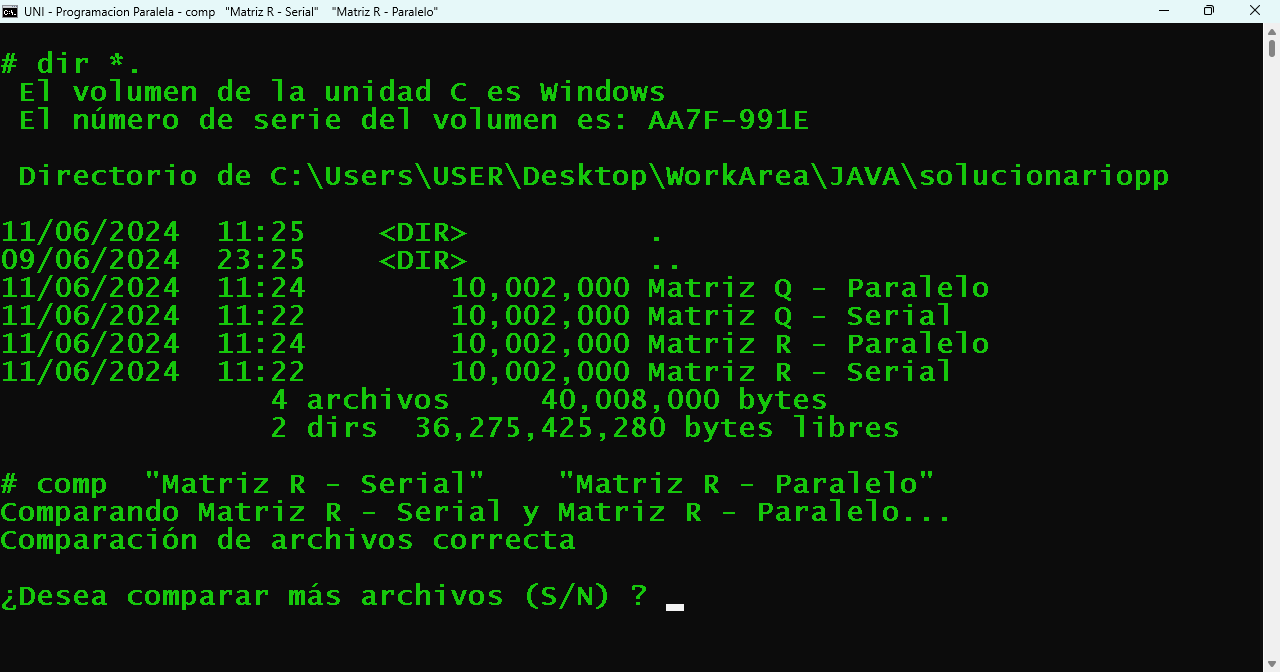
Al procesar los programas anteriores (Serial y Paralelo) se crearon los siguientes archivos





Comparando los archivos generados en modo serial y paralelo





Se verifica que son correctos