LAB10

Objetivos:

- Realización de operaciones sobre matrices
- Realización de operaciones sobre listas estáticas
- Realización de operaciones sobre tipos mixtos

Lab 10. Programas Ada con Listas estáticas, Matrices y Tipos mixtos

10.1. Lista de ejercicios

Varios de los ejercicios que aquí se proponen se han propuesto previamente. En este laboratorio cambian el tipo de datos en el que se almacenan los datos y/o los resultados. Así, la resolución del ejercicio no radicará en los cálculos a realizar sino en la manera de acceder a los datos y/o resultados.

10.1.1. Insertar en medio

Implementa el procedimiento *Insertar_En_Medio* que, a partir de una lista estática y un número natural N, lo inserte en la mitad de la lista, si el número de elementos de la lista es par. Si el número de elementos de la lista es impar o la lista está llena se deja la lista como está. Por ejemplo, si L= [4, 5, 3, 6] y N= 9 debería quedar [4, 5, 9, 3, 6]. Si L= [4, 6, 3] y N= 9 debería quedar [4, 6, 3]. Si L= [] y N= 3 debería quedar [3].

10.1.2. Borrar intermedio

Implementa el procedimiento *Borrar_Intermedio* que, dada una lista estática de enteros L, la modifique eliminando el elemento intermedio (aquel que tiene el mismo número de nodos antes y después de dicho elemento). Esto significa que el subprograma debe eliminar un elemento SOLO cuando L tiene un número impar de elementos. Si L tiene un número par de elementos, entonces no se hace nada a la lista de partida.

10.1.3. Encriptar

Implementa el procedimiento *Encriptar* que, dados dos vectores de enteros *N*, *Clave,* devuelva otro vector *Num* con los dígitos de *N* recolocados usando la clave. N representa los dígitos de un número entero y clave tiene los números de 1 a N'length.

Para encriptar N, se coloca cada uno de los dígitos de N en la posición indicada en V_Clave, suponiendo siempre que el número N se rellena con tantos ceros a la izquierda como sea necesario. Por ejemplo, si N es (8,7,5,3,9) y Clave es (2,4,1,5,3), el resultado es (5,8,9,7,3), ya que el 8 va a la segunda posición, el 7 va a la cuarta posición, el 5 va a la primera posición, el 3 va a la quinta posición y el 9 va a la tercera posición. Si N es(0,0,5,7,6) y Clave es (2,4,1,5,3), el resultado es (5,0,6,0,7) ya que el primer 0 va a la segunda posición, el segundo 0 va a la cuarta posición, el 5 va a la primera posición, el 7 va a la quinta posición y el 6 va a la tercera posición.

10.1.4. Intersección

Implementa una función llamada *comunes* que, dados dos vectores de enteros cuyos valores están ordenados crecientemente, devuelva cuántos elementos son comunes a los dos vectores. Como ejemplo, con (3, 4, 5, 6, 7, 8, **9**, 10, **11**, 12) y (**9**, **11**, 13, 15, 17, 19, 21, 23, 25, 27), devuelve **2** (el 9 y el 11 son comunes a los dos). Nota: El hecho de que los vectores estén ordenados debe influir en el algoritmo a utilizar y una solución con dos bucles anidados no es aceptable.

10.1.5. Calcular el número de vecinos por vivienda del edificio

Implementa en Ada el procedimiento *Obtener_Num_Vecinos_Por_Vivienda* que, dados los datos de toda la comunidad, obtenga la distribución de los vecinos por vivienda en el rascacielos. Tenemos datos de los 3.546 vecinos de una comunidad de vecinos. Por cada vecino tenemos información de su nombre, número de piso y mano donde vive. La comunidad vive en un rascacielos de 100 pisos (numerados del 1 al 100) y en cada piso hay 10 manos (nombradas desde la 'A' a la 'J'). Para acceder más fácilmente al número de habitantes por cada vivienda (piso y mano concretos), queremos transformar los datos en otra estructura de datos (matriz T_Rascacielos).

Por ejemplo, si en el 1º A viven 5 personas en el vector de la comunidad habrá 5 posiciones en las que aparecerá información de vecinos del piso 1 y mano A. Tras el proceso de ir revisando el vector de la comunidad de vecinos, en R (de tipo T_Rascacielos) aparecerá el valor 5 en la primera celda, es decir, R(1, A)=5. NOTA sobre eficiencia: el procedimiento debe resolver el problema con un único recorrido del vector de la Comunidad.

Jon 1 A	Aiora 3 B	Koldo 2 A	Miren 2 A	...	Mireia 1 A	Leire 99 A	...	Iker 1 A	...	...	Aritz 1 A	...	...	Ane 1 A	...	...
1	2			...	32		...	145		...	13455		...	2500		...

	'A'	'B'		...	'J'
1	5				
2	2				
3		1			
4					
...					
99	1				
100					

10.1.6. Obtener información de consumos

Tenemos información de una comunidad de vecinos que vive distribuida en un rascacielos de 100 pisos (numerados del 1 al 100) y por cada piso hay 10 manos (nombradas de la 'A' a la 'J'). Por cada vivienda (piso y mano concretos) tenemos información del consumo de electricidad, consumo de gas y número de habitantes. Toda esa información la representamos con T_Edificio. Implementa el procedimiento **Obtener_Consumos** que, dada la información de todo el edificio, y **utilizando un único recorrido de la matriz**, obtenga,

teniendo en cuenta el número de habitantes en cada piso: (1) el consumo eléctrico medio por habitante de todo el rascacielos (un único valor para todo el edificio, el cálculo usa todos los habitantes del edificio), y (2) los consumos medios de gas por habitante de cada mano del rascacielos (10 valores distintos, uno para cada mano; usa T_Consumo_Medio_Manos para el resultado; en cada cálculo solo se tienen en cuenta los habitantes de esa mano en el edificio).

10.1.7. Lluvias

Implementa el procedimiento **Maximo** que, dados los datos pluviométricos del siglo XXI, y dos fechas de este siglo, indique cuál es el mes que más ha llovido de los que se encuentran incluidos entre esas dos fechas. En los extremos solo se cuentan los días del mes que se incluyen. Por ejemplo, Entre el 10-ENE-2011 y el 15-Ene-2011 el resultado debe ser ENERO-2001, porque solo está ese mes comprendido entre esas dos fechas. Si fuera entre 10-ENE-2011 y 15-FEB-2011, entonces se contarían las lluvias entre el 10 y el 31 de enero frente a las del 1 al 15 de febrero. En la fecha de resultado, el valor del día no tiene relevancia (solo importan los otros dos).

10.1.8. Simplificar lista de carreteras 1

Implementa el procedimiento **Simplificar** que, a partir de una lista estática de carreteras L de tipo T_Estatica_Carreteras, elimine de ella las cinco primeras que sean de peaje y las devuelva en una lista nueva en el mismo orden en que se encuentran. Se considera que una carretera es de peaje cuando tiene más de 0 km de peaje. Nota: Se penalizará el uso de listas auxiliares en la resolución del problema.

10.1.9. Simplificar lista de carreteras 2

Tenemos una representación de carreteras con demasiados puntos y queremos reducir el espacio de memoria necesario para representarlas aun perdiendo algo de información. Implementa en Ada el procedimiento **Simplificar** que, a partir de una lista de carreteras L de tipo T_Estatica_Carreteras, la modifique eliminando para cada carretera aquellos puntos cuyas diferencias en los valores X e Y con el punto anterior sean menores que 0.001 (en ambos valores). Los puntos que se queden en la lista deben mantener el orden de la lista original y los puntos inicial y final siempre se habrán de mantener. **Nota:** Se penalizará el uso de listas auxiliares en la resolución del problema.

10.1.10. Simplificar lista de puntos

Implementa en Ada el procedimiento **Simplificar** que, a partir de una lista de puntos no vacía L de tipo T_Estatica_Puntos, la modifique para que solo contenga un cuarto de los puntos, respetando uno sí y tres no de la lista original. Los puntos que se queden en la lista deben mantener el orden de la lista original y los puntos inicial y final siempre se habrán de mantener. También se devolverá el número de puntos restantes en la lista resultado. Nota: Se penalizará el uso de listas auxiliares. Para reducir la lista se añade al final un punto de coordenadas (-1,-1), A partir de ahí, los puntos se ignoran. Por ejemplo:

- Si L= <p1, p2> el resultado sería <p1, p2> y **2**
- Si L= <p1, p2, p3, p4> el resultado sería <p1, p4> y **2**
- Si L= <p1, p2, p3, p4**,** p5**,** p6**,** p7**,** p8, p9, p10**,** p11> el resultado sería <p1, p5, p9, p11> y **4**

10.1.11. Localizar máximo patrimonio

Implementa la función **dni_mayor_patrimonio** que, dada una urbanización U, devuelve el DNI de la persona de la urbanización que tiene un mayor patrimonio. Se considera que el patrimonio es la suma de los valores de los pisos que posee. Para resolver este ejercicio necesitarás guardar el patrimonio de cada uno de los diferentes propietarios que te vayas encontrando e ir actualizando su patrimonio. Se sugiere que definas una lista estática en la que guardar por cada elemento dos datos: el dni de un propietario y su patrimonio detectado. Para resolver el problema se tendrá que recorrer todas las propiedades de la urbanización. Para cada propiedad, si el propietario ya estaba registrado previamente, se actualiza su patrimonio con el valor de la propiedad actual y, si no, se añade a la lista con el valor de la propiedad actual como patrimonio. Finalmente, se recorre esta estructura auxiliar para localizar el propietario con mayor patrimonio.

10.1.12. Localizar espacio en blanco de sudoku

En un sudoku siempre hay 9 fichas, cada una con los 9 números (1, 2, …9). En este ejercicio se tiene una matriz NxN (9x9) para representar un sudoku a medio rellenar. En una ficha sin rellenar, en blanco, los números en blanco se representan con el valor 0. Aparte, se dispone de una ficha, representada por una matriz cuadrada MxM (3x3), que hay que colocar en el sudoku. Por ejemplo: Se dispone del siguiente sudoku, que tiene hueco para una ficha, y una ficha a colocar:

1	6	9	2	5	8	4	3	7
5	7	8	4	9	3	6	2	1
3	4	4	7	6	1	9	5	8
9	5	2	1	8	4	7	6	3
6	1	3	5	7	9	2	8	4
4	8	7	6	3	2	5	1	9
8	3	6	9	4	5	0	0	0
7	9	1	8	2	6	0	0	0
2	4	5	3	1	7	0	0	0

Ficha

8	3	1
9	4	7
6	5	2

Una ficha no encaja en el sudoku si algún número se repite en alguna columna, fila, o en la misma ficha. Por ejemplo, tal y como está la ficha, no encaja en el hueco del sudoku, pero también se quiere examinar si puede ser colocada girada 90, 180 o 270 grados:

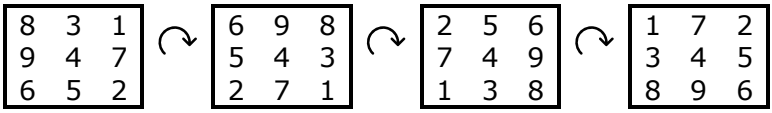
1	6	9	2	5	8	4	**3**	**7**
5	7	8	4	9	3	**6**	2	**1**
3	4	4	7	6	1	**9**	**5**	8
9	5	2	1	8	4	7	6	3
6	1	3	5	7	9	2	8	4
4	8	7	6	3	2	5	1	9
8	**3**	6	9	4	5	0	0	0
7	**9**	1	8	2	6	0	0	0
2	4	**5**	3	1	7	0	0	0

Ficha

8	3	1
9	4	7
6	5	2

En la fila 7, se repiten los números 8 y 3. En la fila 8 se repiten los números 7 y 9. En la fila 9, se repiten los números 2 y 5. En la columna 7, se repiten los números 6 y 9. En la columna 8 se repiten los números 3 y 5. En la columna 9, se repiten los números 1 y 7. La ficha, tal y como está, no se puede colocar en el hueco blanco del sudoku. Pero, la ficha se puede rotar hacia la

derecha para poder encontrar su colocación. Por ejemplo, si se rota la ficha hacia la derecha, se obtienen las siguientes fichas:



Si se rota la ficha 270º, entonces sí que encaja en el sudoku, ya que los números no se repiten en ninguna fila ni columna:

1	6	9	2	5	8	4	3	7
5	7	8	4	9	3	6	2	1
3	4	4	7	6	1	9	5	8
9	5	2	1	8	4	7	6	3
6	1	3	5	7	9	2	8	4
4	8	7	6	3	2	5	1	9
8	3	6	9	4	5	0	0	0
7	9	1	8	2	6	0	0	0
2	4	5	3	1	7	0	0	0

Ficha

1	7	2
3	4	5
8	9	6

Implementa el procedimiento **encontrar_espacio_blanco**. Un espacio en blanco es una submatriz MxM con todas las casillas con el valor 0. Se puede presuponer que, si un número tiene el valor 0, el resto de los números de esa ficha también lo son.

10.1.13. Rotar matriz derecha de sudoku

Implementa el procedimiento **rotar_matriz_derecha_90**, que, dada una ficha, la rote 90º a la derecha. Nota: Se busca una solución transportable a fichas de

cualquier tamaño, por lo que se penalizarán soluciones que no incluyan un bucle.

10.1.14. Comprobar filas correctas de sudoku

Implementa la función **filas_correctas**, que, dados un sudoku sin terminar (con un espacio para una ficha), una ficha, y la especificación de la casilla superior izquierda de un hueco del sudoku, compruebe que si el resultado de añadir la ficha a el sudoku con un espacio en blanco, todas las filas y las columnas cumplen los requisitos.