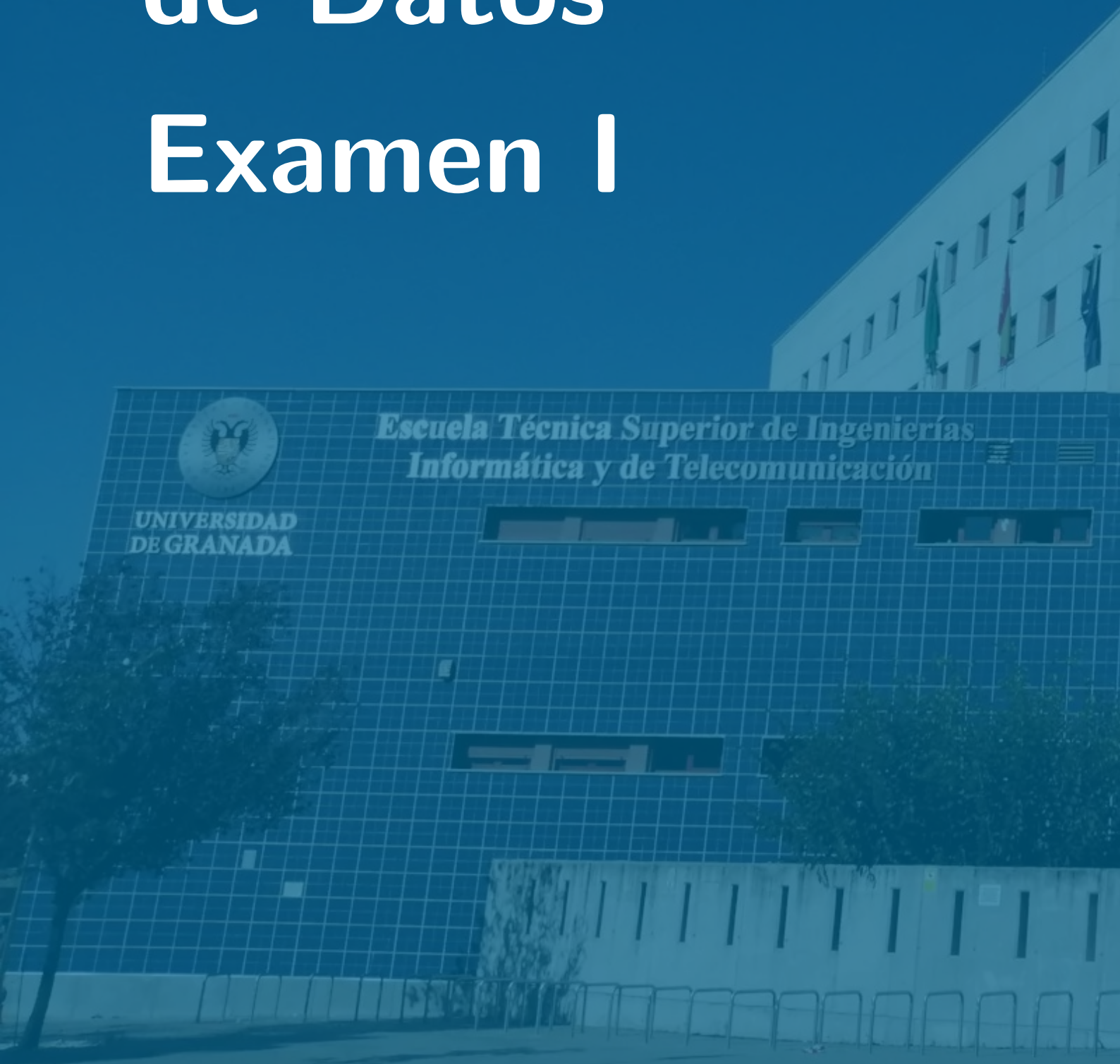
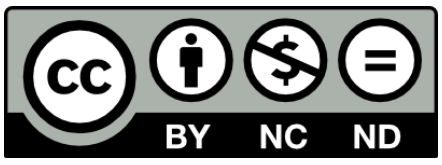
Estructura de Datos Examen I



Los Del DGIIM, losdeldgiim.github.io

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Estructura de Datos Examen I

Los Del DGIIM, `losdeldgiim.github.io`

José Juan Urrutia Milán

Granada, 2023-2024

Asignatura Estructura de Datos.

Curso Académico 2023-24.

Grado Doble Grado en Ingeniería Informática y Matemáticas.

Grupo Único.

Profesor Joaquín Fernández Valdivia.

Descripción Convocatoria Ordinaria.

Duración 2 horas y media.

Ejercicio 1 (1 punto)**.** Elegir en cada caso la opción correcta, de forma justificada.

(a) Si inserto las claves $\{2, 4, 5, 6, 12, 1, 3\}$ en un **APO** de enteros:

(a1) Hay que hacer un solo intercambio padre-hijo.

(a2) Hay que hacer dos intercambios padre-hijo.

(a3) Hay que hacer tres intercambios padre-hijo.

(a4) Todo lo anterior es falso.

Mostrar el árbol final

(b) Dadas las siguientes 3 afirmaciones:

- Es correcto en un esquema de **hashing cerrado** el uso como función hash de:
$$h(k) = [k + 2k] \,\%M \qquad M \text{ primo}$$
- La declaración `map<list<int>, string> m;` es una declaración válida.
- El elemento de valor máximo en un `ABB<int>` se encuentra en el nodo de más profundidad.

(b1) Todas son falsas.

(b2) Hay 2 ciertas y 1 falsa.

(b3) Hay 1 cierta y 2 falsas.

(b4) Todas son ciertas.

(c) Dados los siguientes recorridos Preorden y Postorden:

$$Pre = \{A, Z, X, Q, V, Y, L, W, T, R\} \quad Post = \{Q, V, X, Y, L, Z, T, R, W, A\}$$

(c1) Hay exactamente 2 árboles binarios con esos recorridos.

(c2) No hay ningún árbol binario con esos recorridos.

(c3) Hay exactamente 1 árbol binario con esos recorridos.

(c4) Hay más de 2 árboles binarios con esos recorridos.

Razona la respuesta

(d) Dados dos nodos n_1 y n_2 en un árbol binario T y dadas las distancias (longitudes de los camino) m_1 y m_2 de ambos nodos a su antecesor común más cercano (nodo más profundo que tiene tanto a n_1 como a n_2 como descendientes):

(d1) Si $m_1 = m_2 = 1$ los nodos son el mismo nodo.

(d2) Si $m_1 = 0$ y $m_2 > 0$: n_2 es sucesor de n_1.

(d3) Si $m_1 = m_2 = 2$ los nodos no son hermanos.

(d4) Todo lo anterior es cierto.

Ejercicio 2 (1 punto)**.** Supongamos que respresentamos una lista usando un vector de la siguiente forma:

```
1  class listacursores{
2      private:
3      struct dato{char elem; int siguiente;};
4
5      vector<dato> elementos;
6      int primero;
7      int nelems;
8
9      public:
10     // ...
11 };
```

donde el campo `elem` es el elemento de cada posición de la lista, y `siguiente` indica la posición dentro del vector en que está el siguiente elemento de la lista. Ejemplo: El vector:

0	1	2	3	4
a,4	d,5	b,3	e,1	c,2

con: `nelems = 5` `primero = 0`, representa la lista (en orden) `L: <a, c, b, e, d>`.

Dada dicha representación de listas donde la posición de cada elemento viene determinada por un número entero, **construir una clase iteradora**, de forma que los elementos listados por el iterador deben aparecer en el orden en que están en la lista (independientemente de cómo estén almacenados en el vector). Para hacerlo correctamente, deben implementarse constructor, `*`, `==`, `!=`, `++`, junto con las funciones `begin()` y `end()` de la clase `listacursores`.

Ejercicio 3 (1 punto)**.** Implementar una función:

```
void divide_por_signo(list<int> &L, vector<list<int> > &VL);
```

que dada una lista `L`, devuelve en el vector de listas `VL` las sublistas contiguas del mismo signo (el 0 se considera junto con los positivos). El algoritmo puede modificar a `L`.

Ejemplos:

$$L = \{4, -3, -5, -4, -5, -1, 4, -1, -5, -5\} \Rightarrow$$
$$\Rightarrow VL = [\{4\}, \{-3, -5, -4, -5, -1\}, \{4\}, \{-1, -5, -5\}]$$

$$L = \{0, 4, -2, 4, 1, -1, -4, -4, -3, -1, -4, 4, 1\} \Rightarrow$$
$$\Rightarrow VL = [\{0, 4\}, \{-2\}, \{4, 1\}, \{-1, -4, -4, -3, -1, -4\}, \{4, 1\}]$$

$$L = \{2, -1, 3, -3, 3, -3, 0, -1, 0\} \Rightarrow$$
$$\Rightarrow VL = [\{2\}, \{-1\}, \{3\}, \{-3\}, \{3\}, \{-3\}, \{0\}, \{-1\}, \{0\}]$$

Ejercicio 4 (1 punto)**.** Implementar la función

```
void Fibonacci_Trees(vector<bintree<int>> &v, int n);
```

que construye la sucesión de árboles binarios de Fibonacci y los almacena en un vector de árboles. La sucesión comienza con un árbol con 1 solo nodo (T_0) y un árbol con un solo hijo a la derecha (T_1). A partir de ellos, se construye la sucesión construyendo cada árbol binario T_i insertando T_{i-1} a la derecha y T_{i-2} a la izquierda (para $i = 2, \ldots, n$). La etiqueta de la raíz del nuevo árbol se obtiene como la suma de las etiquetas de las raíces de los árboles izquierdo y derecho.

Ejemplo:

Figura 1: T0

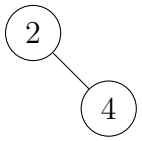


Figura 2: T1

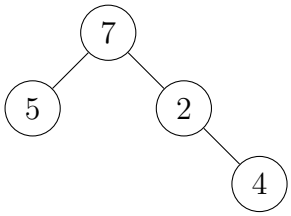


Figura 3: T2

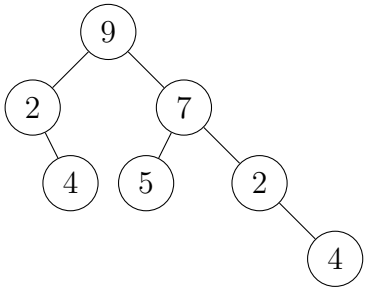


Figura 4: T3

Ejercicio 5 (1 punto)**.** Dados dos `map`, `M1` y `M2`, definidos como:

```
map<string,int> M1, M2;
```

con el primer campo representando el nombre de una **persona** (`string`) y el segundo campo su **número de seguidores** (`int`) en una red social, **implementar una función:**

```
map<string,int> Union (const map<string,int> &M1, const map<string,int> &M2);
```

que obtenga el `map` correspondiente a la unión de los dos `map` de entrada, en el que el número de seguidores será la suma de los seguidores en `M1` y los seguidores en `M2` para la misma persona que aparece en `M1` y `M2`. En el caso que solamente aparezca en uno de los dos se queda tal cual en el `map` resultado.

Ejercicio 6 (1 punto)**.**

(a) Insertar en el orden indicado (detallando los pasos) los siguientes claves en un **AVL**: $\{45, 30, 48, 65, 49, 51, 81, 37, 6, 62, 52, 73\}$. Borrar el elemento 49 del árbol.

(b) Insertar (detallando los pasos) las claves $\{8, 16, 12, 41, 10, 62, 27, 65, 13\}$ en una **Tabla Hash cerrada** de tamaño 13. A continuación, borrar el 10 y finalmente insertar el valor 51. Resolver las colisiones usando hashing doble.