Todas las preguntas tienen la misma ponderación (25%).

1. Suponer que tenemos definidos los TAD **Natural** y **Booleano**. Queremos definir el TAD genérico **Lista[T]**, para almacenar listas de elementos de tipo **T**. El tipo **T** debe tener definida una operación de comparación entre sus elementos, que tendrá el nombre **esMenor(*t1*, *t2*)**, que indica si ***t1*** es menor que ***t2***. Construir una especificación formal, usando el método axiomático, del TAD **Lista[T]**. El tipo deberá tener las siguientes operaciones:

   - **vacia**: crea una nueva lista vacía.
   - **insertar**(*elemento*, *lista*): inserta un nuevo elemento en una lista.
   - **tamaño**(*lista*): devuelve el número de elementos de la lista.
   - **contarMayores**(*valor*, *lista*): dada una lista y un *valor* de tipo **T**, devuelve un natural que indica el número de elementos de la lista que son mayores que *valor*.
   - **entreValores**(*valor1*, *valor2*, *lista*): devuelve una lista con los elementos de *lista* que sean mayores que *valor1* y menores que *valor2*, en el mismo orden que la lista original.

   Escribir las cuatro partes de la especificación axiomática (nombre, conjuntos, sintaxis y semántica) del tipo genérico **Lista[T]**.

2. En una aplicación necesitamos almacenar información sobre personas: el nombre, el DNI, el teléfono, la dirección, el email y la fecha de nacimiento. Algunos de esos campos pueden ser nulos si no los conocemos. Además, a cada persona queremos asignarle un número consecutivo y único, desde 1 hasta *n*, que se almacena junto con los datos de la persona. En la aplicación necesitamos que las consultas por nombre, por DNI y por teléfono sean muy rápidas.

   Describir cómo se utilizarían las tablas de dispersión para almacenar los datos, optimizando la búsqueda por los tres criterios indicados. Se deben indicar la tabla o tablas de dispersión que se usarían, proponer funciones de dispersión y de redispersión, en su caso, qué datos se almacenan, qué tamaños de tabla usar, etc. Hacer una estimación del uso de memoria de la aplicación y de la eficiencia conseguida para las operaciones de consulta.

3. Tenemos un conjunto de palabras almacenadas en un árbol trie. Escribir una operación que encuentre de forma eficiente todas las palabras del conjunto que tengan sus consonantes duplicadas; es decir, aquellas palabras tales que sus consonantes aparecen dos veces seguidas. Por ejemplo, serían ejemplos las palabras: mmississippi, ella, llaollao, erroll, ppmmbbss, ea (esta última, al no tener consonantes, también cumple la condición).

   No se debe suponer una representación concreta de los nodos del trie con listas o con arrays, sino que se debe suponer que tenemos unas operaciones genéricas sobre los nodos trie: *Consulta (n: trie, c: carácter): trie* (dado un nodo *n* del trie, devuelve el hijo de *n* para el carácter *c*, o NULO si no existe), y un iterador del tipo *para cada carácter c hijo del nodo n hacer*. Además, también se supone que existen funciones *esVocal(c: carácter): booleano* y *esConsonante(c: carácter): booleano*.

4. Una empresa ha diseñado el siguiente plan de evacuación de sus trabajadores en caso de incendio. Cada trabajador tiene una lista de cinco trabajadores, a los que debe avisar si se entera de un incendio. Cuando un trabajador ve un incendio, avisa a los cinco de su lista; estos avisan a los de sus listas, y así hasta que todos están avisados (decimos que el plan se ha completado). Se supone que cada trabajador tarda 1 minuto en encontrar su lista, y otro minuto por cada uno de los que avisa. Por lo tanto, si el trabajador A tiene en su lista: B, C, D, E, F, entonces B es avisado a los 2 minutos, C a los 3 minutos, y así sucesivamente.

   Un trabajador puede recibir el aviso desde distintas fuentes; con la que antes llegue a él, se dará por enterado y empezará a llamar a su lista. Como no sabe los que ya están avisados, llama siempre a todos los de su lista de cinco.

   Queremos saber cual es el trabajador tal que si se empieza por él, más tiempo tarda en completarse el plan de evacuación. Escribir un algoritmo que lo calcule de manera eficiente. Suponer que los trabajadores están numerados desde 1 hasta **n**, y la matriz **P[1..n][1..5]** contiene para cada trabajador **x** su lista de personas a las que debe avisar en: P[x][1], P[x][2], …, P[x][5].