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

1. En los hospitales, las listas de espera quirúrgicas son muy largas. Pero, ¿qué ocurre cuando llega un caso realmente urgente? En esas situaciones no es necesario que el paciente espere a que le llegue su turno, ya que solo tendría que esperar si hay más casos urgentes que llegaron antes que él. Se pide realizar la especificación algebraica completa de un TAD *THospital*, de tal modo que, si un paciente es un caso urgente, se sitúa por delante de los pacientes normales y detrás de los urgentes (si los hay). En el caso de que dos pacientes tengan la misma urgencia, se atenderá en primer lugar al que llegó primero (entenderemos que un paciente consta de un DNI y de un campo que indica si es un caso es urgente o no).

   Las operaciones que debe incluir el TAD *THospital* son:

   - **crear**: crea un elemento de tipo *THospital* cuya lista de espera está vacía.
   - **insertar**: dado un elemento de tipo *THospital* y un nuevo paciente (es decir, un número de DNI y un booleano que indica si es urgente o no), lo inserta en la lista de espera.
   - **esVacío**: dado un elemento de tipo *THospital*, nos indica si quedan o no pacientes que atender.
   - **urgentes**: dado un elemento de tipo *THospital*, nos dice cuántos pacientes son urgentes.
   - **primero**: dado un elemento de tipo *THospital*, devuelve el DNI del paciente que será atendido en primer lugar.
   - **quitar**: dado un elemento de tipo *THospital*, elimina el paciente cuya urgencia es más elevada.

2. En una aplicación utilizamos tablas de dispersión para almacenar ciertos datos. Tenemos la siguiente función: h(k, a, b) = ($a \cdot k + b$) mod B, donde k son las claves que se almacenan en la tabla; a y b son constantes dadas; y B es el tamaño de la tabla.

   Definimos la siguiente estrategia de dispersión: dada una clave k, se intenta almacenar el elemento en la posición h(k, 3, 1); si esa posición está ocupada, se intenta meter en la cubeta h(k, 5, 0); si también está ocupada, se intenta meter en h(k, 9, 3); si se produce otra nueva colisión, se mete en una lista especial en la que se encuentran todos los elementos que han provocado 3 colisiones. Dicha lista es ordenada.

   Suponer que tenemos una tabla con B = 11, y que queremos meter los siguientes elementos en orden: 52, 35, 30, 63, 90, 5, 14, 41.

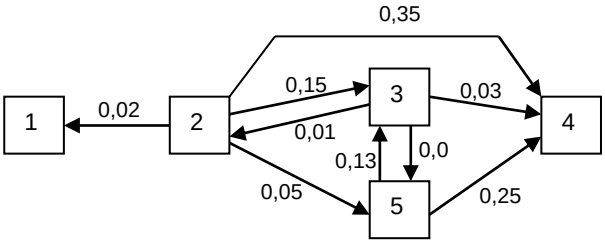
   a) Mostrar las secuencias de búsqueda de los elementos y la tabla de dispersión resultante. Hacer una estimación del uso de memoria y de la longitud total de las secuencias de búsqueda para este ejemplo. Suponer que los enteros y los punteros ocupan 4 bytes.
   b) Repetir la pregunta a) suponiendo que los mismos datos se meten en una tabla de dispersión abierta. En este caso se considera que la función de dispersión es h(k, 3, 1).
   c) Repetir la pregunta a) suponiendo que los mismos datos se meten en una tabla de dispersión cerrada con redispersión lineal. En este caso también se considera que la función de dispersión es h(k, 3, 1).
   d) Comparar los resultados de los tres tipos de dispersión.

3. En una aplicación tenemos implementados árboles B de orden $p = n+1$. El tipo ArbolB es básicamente un puntero a un nodo. Y los nodos del árbol son del siguiente tipo:

```
tipo NodoB = registro
                numClaves: entero       // Indica el número de claves usadas
                claves: array [1..n] de T
                valores: array [[1..n] de Tvalor
                punteros: array [0..n] de Puntero[NodoB]
        finregistro
```

   Escribir un algoritmo que recorra de forma ordenada el árbol B y devuelva los valores almacenados en él, ordenados según sus claves.

4. Queremos trasferir dinero desde un banco A hasta otro banco B en Suiza. Por cada trasferencia nos cobran un porcentaje de la cantidad traspasada, más una cantidad fija de 10 euros. El porcentaje depende del banco de origen y el de destino. En consecuencia, para pasar el dinero de A hasta B, puede ser más rentable pasar por otros bancos intermedios. Por ejemplo, en el caso de abajo suponer que queremos transferir dinero desde el banco 2 al 4. Según la cantidad inicial, interesará más uno u otro camino.



   Escribir un algoritmo eficiente que calcule la forma óptima de trasferir X euros desde el banco A hasta el B, y la cantidad que nos queda al final. Los bancos están numerados desde 1 hasta N; en la matriz C[i, j] se almacena el porcentaje (entre 0 y 1) de comisión que nos cobran al hacer una trasferencia de i hasta j. C[i, j] = 0, significa que no nos aplican un porcentaje (pero siempre nos cobran los 10 euros fijos); y C[i, j] = 1, significa que no está permitida esa trasferencia (pues nos lo quitarían todo).