Examen. 22 de enero de 2016

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

1. Queremos definir formalmente un TAD **Array[T]**, para representar arrays de elementos de tipo **T**. Los arrays se crean con cierto tamaño, *n*, que será un número natural; las posiciones válidas del array irán desde 1 hasta *n*. Incluimos las siguientes operaciones en el tipo:

   - **crear**: dado un tamaño (natural), crea un nuevo array de ese tamaño. Las posiciones del array no se inicializan.
   - **escribir**: dado un array, una posición (natural) y un valor de tipo **T**, inserta ese valor en la posición del array indicada.
   - **verificar**: dado un array, comprueba si se ha realizado alguna escritura que esté fuera del rango del array. Devuelve un booleano, que será *true* si todas las escrituras son correctas, o *false* en caso contrario.
   - **leer**: dado un array y una posición (natural), devuelve el valor del array en dicha posición. Esta operación puede devolver tres tipos de error: "lectura fuera del array", "posición no inicializada", o "escritura fuera del array" (si alguna escritura previa se sale del array).
   - **borrar**: dado un array, elimina todas las escrituras, dejándolo con valores no inicializados.

   Escribir una especificación formal axiomática del TAD **Array[T]**, incluyendo las cuatro partes de la especificación: nombre, conjuntos, sintaxis y semántica. Si se considera necesario se pueden añadir otras operaciones, que deberán ser especificadas.

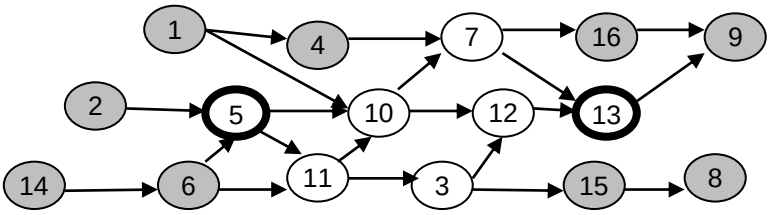
2. Las tablas de dispersión *cuckoo* (cucas) son tablas de dispersión cerradas en las que el mecanismo de resolución de colisiones consiste en disponer de dos tablas y dos funciones hashing distintas que permitan encontrar posiciones alternativas para los elementos a almacenar. La idea, entonces, es trabajar con dos tablas T_1 y T_2 de M posiciones cada una, y dos funciones de dispersión h_1 y h_2, cada una vinculada a una de las tablas. Al insertar un nuevo elemento x, se siguen los siguientes pasos:

   - Si $T_1[h_1(x)]$ está libre, poner x en esa posición y terminar.
   - De lo contrario, sea x' el elemento que hay en $T_1[h_1(x)]$.
     - Poner x en lugar de x' en dicha posición.
     - Ahora hay que recolocar x'. Esto lo logramos ubicándolo en $T_2[h_2(x')]$.
     - Si esa posición de la segunda tabla estuviera ocupada por un elemento x'', se repetiría el proceso ubicando este último en $T_1[h_1(x'')]$, y así sucesivamente.
   - Este proceso puede producir ciclos de reubicaciones, con lo que no terminaría nunca. En tal caso, se hace una reestructuración de las tablas.

   a) Dadas dos tablas de dispersión *cuckoo* de tamaño $M = 10$, con $h_1(x) = x \bmod M$ y $h_2(x) = x \operatorname{div} M$, siendo *div* la división entera, insertar los siguientes elementos describiendo y justificando la evolución de la tablas: 13, 67, 47, 93, 15, 5.
   b) Describir cómo sería el proceso de búsqueda de un elemento en tablas de dispersión *cuckoo* de tamaño M. Comparar el tiempo de búsqueda en el peor caso en tablas de dispersión *cuckoo* frente al tiempo de búsqueda en el peor caso en una tabla de dispersión cerrada tradicional. ¿Cuál resulta más eficiente para la operación de búsqueda?

3. Dibujar un árbol AVL que contenga los números naturales del 1 al 12, ambos incluidos, y que tenga la mayor profundidad posible. Sobre el árbol resultante, indicar qué elemento (o elementos) provocan el mayor número de rotaciones al ser borrados, de qué tipo son y sobre qué nodos se producen, y mostrar la evolución del árbol en el proceso de borrado. Explicar bien todo el proceso y razonar sobre el tiempo de ejecución de la operación de eliminación en estos casos.

4. En un grafo dirigido acíclico tenemos definidos dos nodos especiales: un nodo inicio y un nodo fin. Estamos interesados en todos los caminos que pueden ir del inicio hasta el fin, de manera que queremos encontrar todos los nodos por los que no puedan pasar esos caminos. Por ejemplo, en la figura de abajo el inicio es el nodo 5, el fin es el nodo 13, y los nodos por lo que no pueden pasar los caminos de inicio a fin están señalados en gris.



Escribir un algoritmo que encuentre de forma eficiente todos los nodos del grafo por los cuales no pueda pasar ningún camino entre el nodo inicio y el fin. Suponer que el grafo tiene *N* nodos y puede estar almacenado con matrices o listas de adyacencia (como se prefiera).