Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

2.do parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la menor complejidad posible dadas las características del problema planteado.**

1. Implementar una **primitiva** `Filter(f func(T) bool)` para el Heap, que deje al heap únicamente con los elementos para los que la función devuelva `true`. La primitiva debe funcionar en $\mathcal{O}(n)$, siendo n la cantidad de elementos inicialmente en el heap. Por supuesto, luego de aplicar la operación, el heap debe quedar en un estado válido para poder seguir operando. Justificar la complejidad de la primitiva implementada.

2. Implementar una función `eliminarRepetidos(arreglo []int) []int` que dado un arreglo de números, nos devuelva otro en el que estén los elementos del original **sin repetidos**. La primera aparición debe mantenerse, y las demás no ser consideradas. Indicar y justificar la complejidad del algoritmo implementado.

3. Se tiene un árbol binario que representa la fase eliminatoria del mundial. En cada nodo guarda el nombre del país, así como la cantidad de goles que convirtió en dicha fase (incluyendo la tanda de penales, si fuera necesario). El padre del nodo debe si o si tener al hijo que ganó (tuvo mayor cantidad de goles). Implementar una primitiva para el árbol donde solamente están los nombres de los equipos en las hojas (no en los internos), y deje el árbol completado con los ganadores en cada fase. Se puede asumir que el árbol es o bien completo, o que al menos todos los nodos internos tienen exactamente 2 hijos. La cantidad de goles en la raíz no es relevante. La estructura del árbol es:

   ```
   type Arbol struct {
       pais    string
       goles   int
       izq     *Arbol
       der     *Arbol
   }
   ```

   Tomando el ejempo del dorso, si invocamos para el árbol de la izquierda, debe quedar como el de la derecha.

Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

2.do parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la menor complejidad posible dadas las características del problema planteado.**

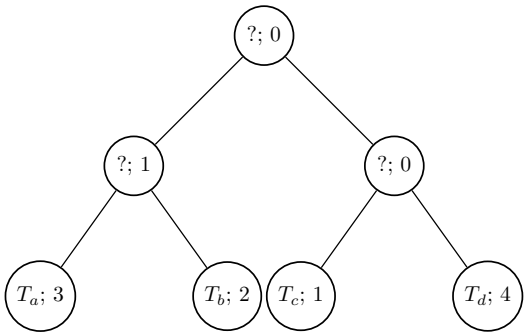
1. Implementar una **primitiva** `Filter(f func(T) bool)` para el Heap, que deje al heap únicamente con los elementos para los que la función devuelva `true`. La primitiva debe funcionar en $\mathcal{O}(n)$, siendo n la cantidad de elementos inicialmente en el heap. Por supuesto, luego de aplicar la operación, el heap debe quedar en un estado válido para poder seguir operando. Justificar la complejidad de la primitiva implementada.

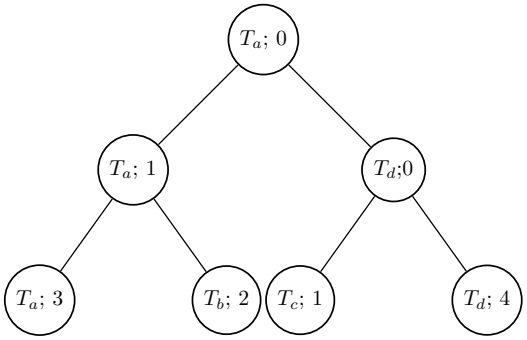
2. Implementar una función `eliminarRepetidos(arreglo []int) []int` que dado un arreglo de números, nos devuelva otro en el que estén los elementos del original **sin repetidos**. La primera aparición debe mantenerse, y las demás no ser consideradas. Indicar y justificar la complejidad del algoritmo implementado.

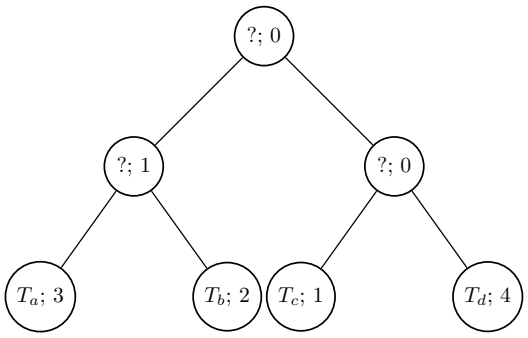
3. Se tiene un árbol binario que representa la fase eliminatoria del mundial. En cada nodo guarda el nombre del país, así como la cantidad de goles que convirtió en dicha fase (incluyendo la tanda de penales, si fuera necesario). El padre del nodo debe si o si tener al hijo que ganó (tuvo mayor cantidad de goles). Implementar una primitiva para el árbol donde solamente están los nombres de los equipos en las hojas (no en los internos), y deje el árbol completado con los ganadores en cada fase. Se puede asumir que el árbol es o bien completo, o que al menos todos los nodos internos tienen exactamente 2 hijos. La cantidad de goles en la raíz no es relevante. La estructura del árbol es:

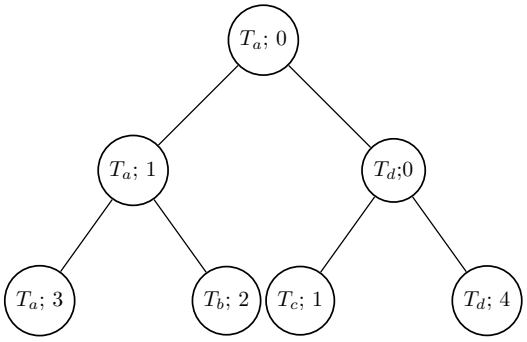
   ```
   type Arbol struct {
       pais    string
       goles   int
       izq     *Arbol
       der     *Arbol
   }
   ```

   Tomando el ejempo del dorso, si invocamos para el árbol de la izquierda, debe quedar como el de la derecha.


?; 0
?; 1
?; 0
T_a; 3
T_b; 2
T_c; 1
T_d; 4


T_a; 0
T_a; 1
T_d;0
T_a; 3
T_b; 2
T_c; 1
T_d; 4


?; 0
?; 1
?; 0
T_a; 3
T_b; 2
T_c; 1
T_d; 4


T_a; 0
T_a; 1
T_d;0
T_a; 3
T_b; 2
T_c; 1
T_d; 4