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

2.do parcialito – 19/05/2025

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 para el ABB `func (arbol *abb[K, V]) AncestroComun(clave1, clave2 K) K` que reciba 2 claves y devuelva el último ancestro en común entre ambas claves. Dicho ancestro en común podría ser incluso alguna de estas claves. Si alguna clave pasada no se encuentra en el árbol, finalizar con `panic`. Indicar y justificar la complejidad de la primitiva implementada.

   Mostramos ejemplos de resultados esperados de invocar la primitiva al **árbol del dorso**:

   ```
   arbol.AncestroComun(1, 4) --> 2
   arbol.AncestroComun(2, 4) --> 2
   arbol.AncestroComun(9, 1) --> 5
   ```

2. Implementar una función `func minimoExcluido(arr []int) int` que dado un arreglo de valores enteros (mayores o iguales a 0), obtenga el mínimo valor que **no** se encuentre en el arreglo. Indicar y justificar la complejidad del algoritmo (explicar en detalle este paso, porque es fácil que se te puedan pasar detalles importantes a explicar).

   Por ejemplo:

   ```
   minimoExcluido([]int{0, 5, 1}) --> 2
   minimoExcluido([]int{3, 5, 1}) --> 0
   minimoExcluido([]int{0, 5, 1, 3, 4, 1, 2}) --> 6
   minimoExcluido([]int{0, 5, 1, 3, 4, 1, 2, 12345675433221345}) --> 6
   ```

3. Realizar el seguimiento de las siguientes operaciones sobre un heap de máximos:

   a. Constuir el heap a partir del arreglo `[5, 3, 6, 1, 4, 7, 8, 10]`

   b. Sobre el heap resultante del punto (a): `Encolar 7`, `Desencolar`, `Desencolar`, `Encolar 13`, `Desencolar`.

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

2.do parcialito – 19/05/2025

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 para el ABB `func (arbol *abb[K, V]) AncestroComun(clave1, clave2 K) K` que reciba 2 claves y devuelva el último ancestro en común entre ambas claves. Dicho ancestro en común podría ser incluso alguna de estas claves. Si alguna clave pasada no se encuentra en el árbol, finalizar con `panic`. Indicar y justificar la complejidad de la primitiva implementada.

   Mostramos ejemplos de resultados esperados de invocar la primitiva al **árbol del dorso**:

   ```
   arbol.AncestroComun(1, 4) --> 2
   arbol.AncestroComun(2, 4) --> 2
   arbol.AncestroComun(9, 1) --> 5
   ```

2. Implementar una función `func minimoExcluido(arr []int) int` que dado un arreglo de valores enteros (mayores o iguales a 0), obtenga el mínimo valor que **no** se encuentre en el arreglo. Indicar y justificar la complejidad del algoritmo (explicar en detalle este paso, porque es fácil que se te puedan pasar detalles importantes a explicar).

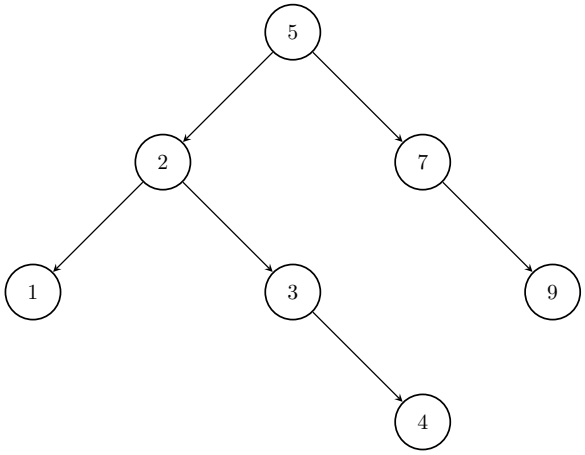
   Por ejemplo:

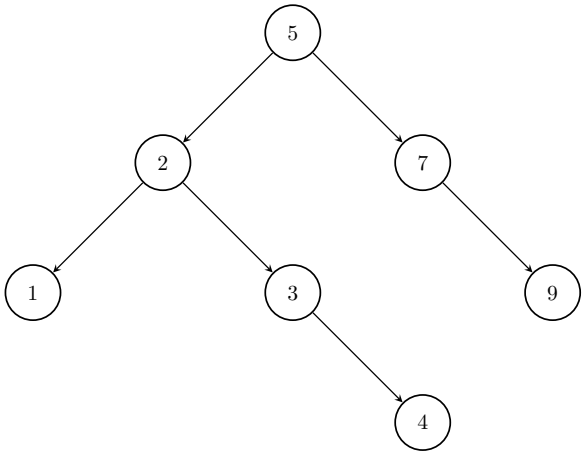
   ```
   minimoExcluido([]int{0, 5, 1}) --> 2
   minimoExcluido([]int{3, 5, 1}) --> 0
   minimoExcluido([]int{0, 5, 1, 3, 4, 1, 2}) --> 6
   minimoExcluido([]int{0, 5, 1, 3, 4, 1, 2, 12345675433221345}) --> 6
   ```

3. Realizar el seguimiento de las siguientes operaciones sobre un heap de máximos:

   a. Constuir el heap a partir del arreglo `[5, 3, 6, 1, 4, 7, 8, 10]`

   b. Sobre el heap resultante del punto (a): `Encolar 7`, `Desencolar`, `Desencolar`, `Encolar 13`, `Desencolar`.


5
2
7
1
3
9
4


5
2
7
1
3
9
4