Práctica 7: Colas de prioridad

Ejercitación básica

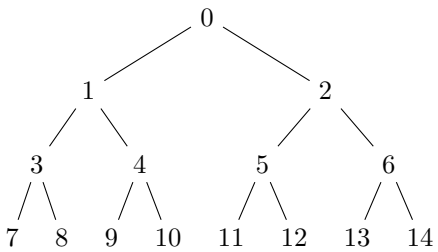
Ejercicio 1.

a) Implementar en Python una función `enBinario`(n) que dado un entero n devuelva su representación binaria, como una lista de 0s y 1s. Por ejemplo:

```
enBinario(0)  =  []
enBinario(1)  =  [1]
enBinario(2)  =  [1, 0]
enBinario(3)  =  [1, 1]
enBinario(4)  =  [1, 0, 0]
```

Observar que, por convención, tomamos `enBinario(0) = []` en lugar de `enBinario(0) = [0]`.

b) Implementar en Python una función `caminoHasta`(n) que dado un entero $n \geq 0$ devuelva el camino que conduce desde la raíz hasta el nodo n-ésimo en un árbol izquierdista. El camino está expresado como una lista de 0s y 1s, en la que 0 corresponde a la arista que baja desde un nodo hacia su hijo izquierdo, mientras que 1 corresponde a la arista que baja desde un nodo hacia su hijo derecho. Por ejemplo, `caminoHasta`$(13) = [1, 1, 0]$ porque para llegar hasta el decimotercer nodo se debe empezar en la raíz, bajar dos veces hacia la derecha y a continuación bajar una vez hacia la izquierda.



Ejercicio 2. Implementar en Python una función `esUnHeap`(A) que dado un arreglo A de n elementos devuelva un booleano indicando si es un heap. La complejidad temporal debe ser $O(n)$ en peor caso.

Se asume que el arreglo representa un árbol izquierdista con la representación vista en clase, de manera que los hijos izquierdo y derecho del nodo en la posición i se encuentran en las posiciones $2i + 1$ y $2i + 2$, mientras que su padre se encuentra en la posición $\lfloor \frac{i-1}{2} \rfloor$.

Ejercicio 3. Implementar en Python el algoritmo `heapsort`(A) que ordene un arreglo A *in-place*, con complejidad temporal $O(n \log n)$. Usar el algoritmo Heapify como primer paso, para convertir el arreglo A en un heap, y proceder eliminando sucesivamente el máximo elemento.

Ejercitación adicional

Ejercicio 4. Supongamos que se tiene un arreglo A ordenado de mayor a menor. ¿Cómo se puede construir un heap que contenga a los elementos de A?

Ejercicio 5. Dado un arreglo A de n enteros:

a) Diseñar un algoritmo para encontrar los k elementos más chicos de A en $O(n + k \log n)$.

b) Comparar el costo del algoritmo propuesto con el costo de ordenar el arreglo usando un algoritmo de ordenamiento "bueno" (p. ej. MERGESORT) y mirar los primeros k elementos del arreglo ordenado.

c) Comparar el costo del algoritmo propuesto con el costo de insertar los n elementos en un AVL y sacar k veces el elemento mínimo del AVL.

Ejercicio 6. En un heap, el máximo elemento (es decir, el primero en orden de prioridad) siempre se encuentra en la raíz.

a) ¿En qué posición del heap se puede encontrar el segundo elemento (en orden de prioridad)? Diseñar un algoritmo para encontrar el segundo elemento en $O(1)$.

b) ¿En qué posición del heap se puede encontrar el tercer elemento? Diseñar un algoritmo para encontrar el tercer elemento en $O(1)$.

c) Diseñar un algoritmo para encontrar y eliminar el k-ésimo elemento en orden de prioridad en $O(k \log n)$, donde n es el número total de elementos en el heap.

Ejercicio 7. Supongamos que se tiene una implementación de una cola de prioridad basada en heaps. La interfaz permite insertar elementos acompañados de un número que indica su prioridad.

a) Explicar cómo se puede implementar una *cola* (*first-in, first-out*) usando la cola de prioridad.

b) Explicar cómo se puede implementar una *pila* (*last-in, first-out*) usando la cola de prioridad.