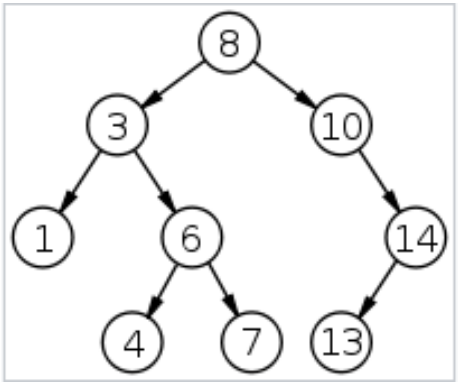
Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Estructuras de Datos y Algoritmos - Algoritmos y Estructuras de Datos II
Primer Cuatrimestre de 2025

Práctica Nº 9

Para los siguientes ejercicios proveer test para chequear el correcto funcionamiento de las implementaciones. Acceder al código de base de los ejercicios de esta práctica aceptando el siguiente assignment: `https://classroom.github.com/a/JtpZZVNL`.

1. Dado el árbol binario de la siguiente figura:



   - Marcar el nodo Raíz
   - Cuántos y cuáles son los nodos Hoja?
   - Cuales nodos son ancestros del nodo 4?
   - Cuál es la altura del árbol?
   - Cuántos y cuáles son los nodos del nivel 3?
   - Cual es la profundidad del nodo 6?
   - Imprimir el camino del nodo 8 al nodo 4

2. Demuestre por inducción las siguientes propiedades de árboles binarios y defina las funciones correspondientes en Haskell.

   2.1 Para todo árbol t: $alt.t \leq size.t$, en donde alt devuelve la altura y $size$ devuelve su tamaño (definir estas operaciones en Haskell).

   2.2 Para todo árbol t: $espejo.espejo.t = t$, en donde $espejo$ es la función que da vuelta los hijos de un árbol recursivamente (definirla en Haskell).

   2.3 Definir la función $mapTree : (a \to b) \to (Tree\ a) \to (Tree\ b)$, que dado un árbol, aplica una función dada a cada elemento del árbol.

   2.4 Para todo árbol t, donde $alt.t$ es su altura, $size.t$ su tamaño y $full.t$ dice si el árbol está lleno (tiene todos los nodos en todos los niveles), demostrar las siguientes propiedades:

   - $size.t \leq 2^{alt.t} - 1$
   - $log_2(size.t + 1) \leq alt.t$
   - $full.t \Rightarrow size.t = 2^{alt.t} - 1$
   - $full.t \Rightarrow alt.t = log_2(size.t + 1)$

3. Tenemos un árbol binario t, su recorrido preorden es HDBACFGLJIKNM y en inorden es ABCDFGHIJKLMN, dibujar el árbol, y dar su recorrido postorden.

4. Considere la Interface *SortedSet* provista en el repositorio correspondiente a esta práctica y agregue:

   - una Implementación usando *Binary Search trees*
   - una Implementación usando *AVLs*

5. Especifique el TAD *SortedMap* y agregue una Implementación usando *Binary Search trees* con el siguiente subconjunto de operaciones

   - *put*
   - $removeKey$
   - *get*
   - *size*
   - $isEmpty$
   - *min*
   - $removeMin$
   - $keySet$
   - $containsKey$

   Ayuda: Notar que la implementación es muy similar a *SortedSet*, pero se requiere almacenar pares (key,value) en los nodos.

6. Considere la clase *WordCounter* provista en el repositorio, que se encarga de almacenar conjuntos de palabras, y contar la cantidad de veces que cada palabra fue observada. Complete la implementación de esta clase utilizando SortedMap y SortedSet definidos por usted en los ejercicios previos y siguiendo la especificación de los diferentes métodos.

7. Implemente la clase Heap con las operaciones insertar, remover, esVacio y repOk. Para cada método calcule su tiempo de ejecución en el peor caso.

   7.1 Utilizando su clase Heap implemente el algoritmo HeapSort. La idea del algoritmo es construir un heap con los elementos del arreglo a ordenar, luego eliminar repetidamente el elemento más grande / más pequeño del heap e insertarlo en el arreglo resultante. Compare este algoritmo con los algoritmos de sorting del repositorio de la materia.

   7.2 Construir un algoritmo que, dado un arreglo de enteros, decida si representa o no un min-heap. Por ejemplo para el arreglo: $[2, 3, 4, 10, 15]$ debería retornar *True*, mientras que para el arreglo: $[2, 10, 4, 5, 3, 15]$ debería retornar *False*.