

Algoritmos y Estructuras de Datos.

Parcial 2. [2024-11-21]

- [ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
- [ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo **CLAS1** en una o más hojas **separadas**, **OPER1** en una o más hojas **separadas**, **PREG1** en una más hojas **separadas**, etc...
- [ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre**,
ASI:

CLAS1, Hoja #2/3	TORVALDS, LINUS
------------------	-----------------

[Ej. 1] [CLAS2 (W=20pt)]

- (⇒ 1.a) **[AB]** Declarar las clases **btree**, **cell**, **iterator**, (preferentemente respetando el anidamiento). Implementar el método **btree<T>::iterator btree<T>::erase(btree<T>::iterator n);**
- (⇒ 1.b) **[set-abb]**: Dada la siguiente interfaz para un conjunto (**set**) por árbol binario de búsqueda:

```

1 template<class T>
2 class set {
3 private:
4     typedef btree<T> tree_t;
5     typedef typename tree_t::iterator node_t;
6     tree_t bstree;
7     ...
8 public:
9     class iterator {
10         node_t node;
11         tree_t &bstree;
12 private:
13         friend class set;
14         iterator(tree_t &t, node_t n) : bstree(t), node(n) {}
15         ...
16     };
17     ...
18     pair<iterator,bool> insert(T x);
19     iterator find(T x); //<- implementar
20     int erase(T x);
21     iterator begin(); //<- implementar
22     iterator end(); //<- implementar
23     node_t next(node_t ); //<- implementar
24     ...
25 };
  
```

implemente los métodos **begin**, **end**, **next** y **find** (si dentro de alguno de ellos utiliza otro método de **set**, no es necesario implementarlo).

[Ej. 2] [OPER2 (W=20pt)]

(⇒ 2.a) [huffman]

- (•2.a.1) Dados los símbolos {a, b, c, d, e, f} con pesos
 $P(a)=0.4$, $P(b)=0.15$, $P(c)=0.15$, $P(d)=0.1$, $P(e)=0.1$, $P(f)=0.1$, indique si los siguientes árboles corresponden a la aplicación del algoritmo de Huffman

- $T1 = (. (. a (. c d)) (. b (. e f)))$
- $T2 = (. a (. (. c d)(. b (. e f))))$,
- $T3 = (. (. (. b (. e f)) (. c d)) a)$

- (•2.a.2) Indicar el código correspondiente a cada símbolo en T1 y codificar la palabra **cabe**.

- (⇒ 2.b) **[abb]** Construir un árbol binario de búsqueda, indicando el árbol tras cada operación.

- (•2.b.1) Insertar sucesivamente los elementos: {4, 2, 7, 5, 6, 3, 1}.
- (•2.b.2) Eliminar 4.
- (•2.b.3) Volver a insertar 4.
- (⇒ 2.c) **[hash]** Considere una tabla de dispersión cerrada con estrategia de redispersión lineal y una TD abierta, ambas con **B=5** cubetas y función de dispersión $h(x) = x \bmod 5$. Indicar sus estados luego de cada grupo de operaciones que se listan a continuación:
 - (•2.c.1) Insertar 15, 22, 13, 7, 15.
 - (•2.c.2) Eliminar 22 e insertar 29, 24, 34.

[Ej. 3] [PREG2 (W=20pt)]

- (⇒ 3.a) ¿Porqué el árbol binario correspondiente a la **codificación binaria** de una serie de caracteres debe ser un árbol binario lleno **FBT (Full Binary Tree)**?
- (⇒ 3.b) ¿Cuales de los siguientes son **árboles binarios llenos (FBT)**?
 - (•3.b.1) (1 2 (3 . (4 5 6)))
 - (•3.b.2) (1 (2 3 4) 5)
 - (•3.b.3) (1 (2 3 4) .)
 - (•3.b.4) (1 (2 . 4) (3 5 6))
- (⇒ 3.c) ¿Es posible **insertar** en una posición **no-dereferenciable** (Δ) en un AB? ¿Y en una **dereferenciable**?
- (⇒ 3.d) Se quiere representar el conjunto de enteros que están en la unión de los rangos $[0, 9]$ y $[20, 29]$, es decir $U = [0, 1, 2, \dots, 9, 20, 21, \dots, 29]$ por **vectores de bits**.
 - (•3.d.1) ¿Cuál es el tamaño del conjunto universal?
 - (•3.d.2) Escribir las funciones **indx()** y **element()** correspondientes.
- (⇒ 3.e) Discuta el tiempo de ejecución (complejidad algorítmica) de las operaciones binarias
 - (•3.e.1) **set_union(A, B, C)**
 - (•3.e.2) **set_intersection(A, B, C)**, y
 - (•3.e.3) **set_difference(A, B, C)**para conjuntos implementados por vectores de bits, donde **A**, **B**, y **C** son subconjuntos de tamaño n_A , n_B , y n_C respectivamente, de un conjunto universal **U** de tamaño N .
- (⇒ 3.f) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión (merge-sort)**. ¿Es estable? ¿Bajo que condiciones?
- (⇒ 3.g) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido (quick-sort)**, en el caso promedio y en el peor caso? ¿Cuándo se produce el peor caso?
- (⇒ 3.h) ¿Qué dos condiciones cumple un **montículo (heap)**? Explique la utilidad de cada condición.