

Algoritmos y Estructuras de Datos. Recup Parciales. [2024-11-28]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo **CLAS2** en una o más hojas **separadas**, **OPER2** en una o más hojas **separadas**, **PREG2** en una o más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI:**

CLAS2, Hoja #2/3	TORNADOS, LINUS
------------------	-----------------

[Ej. 1] [CLAS1 (W=20pt)]

- a) **[list]** Escribir la declaración en C++ del TAD lista (clase **list**, simplemente enlazada) implementado por punteros. Los métodos que solo se deben implementar son **insert(p,x)**, **begin()** y **end()**.
- b) **[stack, queue]** Escribir la implementación en C++ de los métodos **push**, **pop**, **front**, **top**, **clear** y **size** de los TAD pila y cola (clases **stack** y **queue** implementadas con lista), según corresponda.

[Ej. 2] [OPER1 (W=20pt)]

a) [AOO]

- 1) Construir el AOO cuyos nodos, listados en orden previo y posterior, son:

```
1 ORD_PRE: {M,H,D,L,I,E,C,F,G}
2 ORD_POST: {L,I,D,E,H,F,G,C,M}
```

determine los **antecesores propios**, **descendientes propios**, **izquierda** y **derecha** del nodo **H**. ¿Son conjuntos disjuntos? Justifique.

- 2) **[hacer-arbol]** Utilizando sólo métodos **insert**, **lchild**, **n++**, e iteradores del **Arbol Ordenado Orientado AOO**, complete el siguiente código que arma el árbol **T=(7 (2 6 1) 5)**

```
1 tree<int> T;
2 tree<int>::iterator n = T.insert(T.begin(),7);
3 ...
4 // COMPLETAR
5 ...
```

- 3) **[Notación $O(\cdot)$]** Para cada una de las funciones **T1, T2, T3, T4, T5**, determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente:

- $T_1 = n^3 + 3n^2 \log_2 n + 5n + n + 2^{n/2}$
- $T_2 = 3n! + 5n^3 + 2^n$
- $T_3 = 5^n + n^2 \log n + n^4$
- $T_4 = 6n^2 + \log_2 n + 3! + 7^n$
- $T_5 = 50n + 2n^3 + \log_5(10n) + n!$

[Ej. 3] [PREG1 (W=20pt)]

- a) ¿Como se define la altura de un nodo en un árbol? ¿Cuál es la altura de los nodos **n**, **o**, y **p** en el AOO siguiente: (**j (n o (p (q s t u))) k l**)?
- b) Explique la propiedad de transitividad de la notación asintótica $O(\cdot)$. De un ejemplo.

- c) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `map<>` implementada con **listas ordenadas** y **vectores ordenados** en el caso promedio? Métodos: `find(key)`, `M[key]`, `erase(key)`, `erase(p)`, `begin()`, `end()`, `clear()`.
- d) Sea el árbol AOO (`c d r (a s (q u))`). Cuáles de los siguientes son **caminos**?
- (a c)
 - (c a)
 - (a q u)
 - (s a q)
- e) ¿Existe una relación biunívoca entre un Árbol Ordenado Orientado (AOO) y su **orden previo**? ¿Y con el **orden posterior**? ¿Y con ambos a la vez? De ejemplos.
- f) Explique qué se entiende por algoritmos de **búsqueda exhaustiva** y **heurístico**. Discuta cuales son las ventajas y desventajas.
- g) Expresé la **regla del producto** para la notación $O(\cdot)$. De un ejemplo.

[Ej. 4] [CLAS2 (W=20pt)]

- a) **[AB]**: declarar las clases `btree` y `cell`. Implementar el método `btree<T>::iterator btree<T>::insert(btree<T>::iterator n, T t)`;
- 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. 5] [OPER2 (W=20pt)]

- a) **[huff]**:
- 1) ¿El siguiente código cumple la condición de prefijos: `T=010`, `S=011`, `D=01`, `O=001`, `A=10`? Justifique.

- 2) Dado el árbol indicado por el siguiente código Lisp: $(. (. (. D A) T) (. O S))$, decodifique: **1110000001**.
 - 3) Considerando probabilidades (**T:15%, A:25%, D:50%, O:6%, %S:4%**), obtenga un árbol de huffman.
 - 4) Calcule la longitud media del código correspondiente a los árboles de los dos incisos anteriores.
 - 5) Codifique la palabra **DOS** con ambos árboles. Compare las longitudes de los códigos y justifique las diferencias.
- b) **[hash]**: Dadas una tabla de dispersión cerrada y una abierta con **B=5** cubetas, con función de dispersión $h(x) = x \% B$ y redispersión lineal, aplicar y explicar los efectos de las siguientes operaciones para cada una:
- 1) Insertar sucesivamente **{7, 5, 2, 12, 7, 9, 2}**
 - 2) Luego eliminar **7** y **5** (con redispersión).
 - 3) Insertar **8**.
- c) **[ABB]**
- 1) Dados la secuencia **{9, 8, 10, 1, 3, 4, 6, 15}**, construir un árbol binario de búsqueda insertando sucesivamente los elementos.
 - 2) Graficar el árbol tras eliminar **8**.
 - 3) Eliminar **13** del árbol obtenido en 2 y graficar el árbol resultante.
 - 4) Insertar **9** del árbol obtenido en 3 y graficar el árbol resultante.
 - 5) Hacer un nuevo árbol binario de búsqueda insertando los elementos de la secuencia inicial de manera tal que quede balanceado.

[Ej. 6] [PREG2 (W=20pt)]

- a) Si tenemos un iterator **q** dereferenciable en un **set<>** ¿es posible asignarle un valor? Es decir, dado el siguiente código, ¿puede fallar la línea (1)? ¿porqué?
- ```
set<string> S;
//... llena S
auto q = S.begin();
// (Asumir que q es dereferenciable)
*q = "Spiderman"; // (1)
```
- b) Escriba las instrucciones necesarias para crear el siguiente **árbol binario (AB)**: **T=(2 (3 . 4) .)**
- c) Explique que operaciones realizan (semántica) las dos versiones de **erase** para conjuntos.
- ```
1 void erase(iterator p); // (1)
2 int erase(key_t x); // (2)
```
- Explicar qué son los valores de retorno, y porqué una retorna un entero y la otra no. ¿Cuándo puede fallar la versión de iterator (1)? ¿Puede fallar la versión (2) si la clave **x** no tiene asignación?
- d) Discuta el número de **intercambios** que realizan los algoritmos de ordenamiento lentos en el peor caso.
- e) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condición de prefijo y que no cumplen para un conjuntos de 3 caracteres.