Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 2. [2022-11-17]

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 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 TORVALDS, LINUS`

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

a) **[set-abb]** A partir del siguiente bloque de código:

```
1  set<int> S;
2  S.insert(3);
3  S.insert(-1);
4  S.insert(2);
5  S.insert(6);
6  S.insert(4);
7  S.insert(7);
8  auto p = S.find(3);
```

Presente mediante un croquis las estructuras de datos presente en memoria luego de su ejecución. Considere la implementación de conjuntos por medio de árboles binarios de búsqueda.

b) **[ab]** Dado el siguiente resumen de la implementacicen de árbol binario

```
1  class btree{
2    class cell {
3      elem_t t;
4      cell *right,*left;
5      cell() : right(NULL), left(NULL) {}
6    };
7    class iterator {
8      private:
9      cell *ptr,*father;
10     side_t side;
11     iterator(cell *p,side_t side_a,cell *f_a)
12     : ptr(p), side(side_a), father(f_a) { }
13     public:
14     iterator(const iterator_t &q) { ... }
15     bool operator==(iterator_t q) { ... }
16     bool operator!=(iterator_t q) { ... }
17     iterator() : ptr(NULL), side(NONE),
18     father(NULL) { }
19     iterator left() { ... }
20     iterator right() { ... } //-->implementar
21   }
22   ...
23   iterator splice(iterator to, iterator from); //-->implementar
24   iterator insert(iterator it, elem_t e); //-->implementar
25   ...
26 }
```

implementar los métodos `btree::splice`, `iterator::right` y `btree::insert`.

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

a) **[huffman]**: Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario utilizando el algoritmo de Huffman y encodar la palabra **SCALONETA**, $P(S) = 0.07, P(C) = 0.06, P(A) = 0.07, P(L) = 0.20, P(O) = 0.10, P(N) = 0.05, P(E) = 0.05, P(T) = 0.1, P(Y) = 0.07, P(Z) = 0.23$. Calcular la longitud promedio del código obtenido.

b) **[hf-decode]** Utilizando la codificación correspondiente al árbol binario representado por el siguiente código Lisp
(. (. (. G ␣) (. (. (. B R) (. T H)) E)) (. O I))";
desencodar el mensaje:
0100011000001010000100110010100101101101001.

c) **[abb]** Dados los enteros $\{8, 14, 1, 19, 18, 11, 16, 17, 18, 9, 20\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 8, 14 y 19 en ese orden.

d) **[quick-sort]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *clasificación rápida (quick-sort)*. En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos. **Nota:** La partición se puede realizar con cualquier estrategia.

e) **[hash]:** Insertar los enteros **{3, 22, 28, 11, 10, 38, 20, 7}** en una tabla de dispersión cerrada con **B=10** cubetas, con función de dispersión **h(x) = x % B**. Luego eliminar el elemento **10** e insertar el elemento **27**, en ese orden. Mostrar la tabla resultante.

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

a) Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente árbol binario **T=(1 (3 . (2 10 20)) (5 7 .))**
b) Escriba la *"definición recursiva"* de la función **ALTURA** de un Ábrol Ordenado Orientado (AOO).
c) ¿Cuál es la altura de los nodos **A**, **B**, y **C** en el siguiente AOO: **(C (B (T X Y)) (Z A (D P)))**?
d) Explique porqué el método de Huffman produce una codificación que cumple con la *"condición de prefijos"*.
e) Cual es el **orden del tiempo de ejecución** (en promedio) en función del número de elementos (n) que posee el **conjunto por árbol binario de búsqueda (ABB)** de las siguientes operaciones/funciones/métodos. Asuma que el árbol se mantiene bien balanceado.

1) **insert(x)**
2) **erase(p)**
3) **erase(x)**
4) ***p**
5) **set_union(A,B,C)**

f) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol ordenado orientado (AOO)? ¿Y en una posición **dereferenciable**?
g) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol binario (AB)? ¿Y en una posición **dereferenciable**?