Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [2016-08-04]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en las restantes secciones.
[ATENCIÓN 2] Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo CLAS1 en una o más hojas, PROG1 en una o 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 LOVELACE, ADA

[Ej. 1] [Preguntas (mínimo 70 %)]

a) Sea L una lista conteniendo los elementos $(1, 3, 4, 2, 5, 6)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = L.erase(q);
p++;
q = p;
q++;
```

indique para cada uno de los iterators **p**, **q** si es válido o inválido y en el primer caso el valor que almacena (es decir `*p` y `*q`).

b) Explique que quiere decir la propiedad de "Transitividad" de $O()$.

c) Cuál es la "Regla del producto" para $O()$?.

d) Cual es la signatura de la función `insert()` en listas STL?. Diga que es cada una las variables y cual es el tipo de retorno?. En caso de que haya varias signaturas explique una de ellas.

e) Explique con un ejemplo la condición de "prefijo" de un código.

f) Cuál es el costo de inserción en tablas de dispersión abiertas con listas desordenadas en el peor caso?. Diga cual es ese caso.

g) Que condiciones debe satisfacer un árbol binario para ser "árbol binario de búsqueda"?

h) Que quiere decir que un algoritmo de ordenamiento sea "estable"?. Dé ejemplos de algoritmos estables y no estables.

i) Por qué se les dice "lentos" a ciertos algoritmos de ordenamiento?. Liste los algoritmos lentos que conoce. Liste los algoritmos rápidos que conoce.

j) Discuta el número de inserciones que requieren los algoritmos de ordenamiento lentos en el peor caso.

[Ej. 2] [Clases (mínimo 60 %)]

a) Escribir los métodos que se indican del TAD `list<>` `insert(x,p)`, `erase(p)`, `retrieve(p)/*p`, `next()/p++`, `list()` y `~list()` implementados por celdas enlazadas por punteros o cursores. Defina también los miembros privados de la clase.

b) Implementar `set::iterator set::find(T x)` para conjuntos implementados por ABB. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.

[Ej. 3] [Programación (mínimo 40 %)]

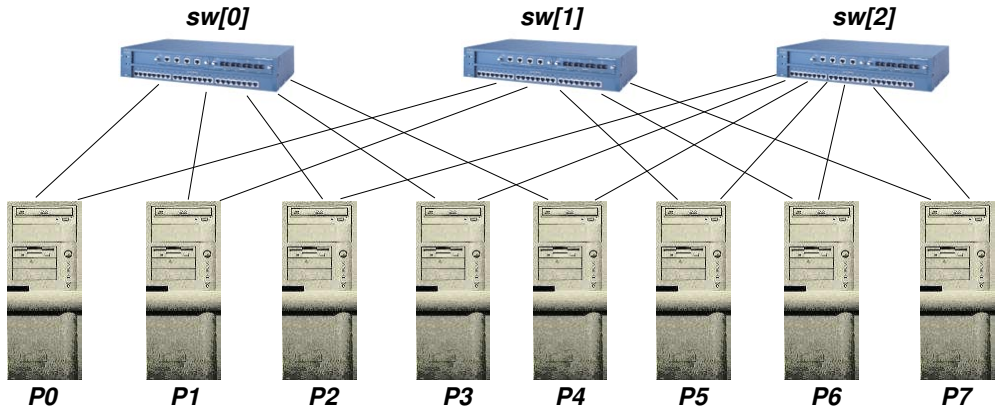
a) [max-dev-n] Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: max_dev$(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$. Escribir una función **`int max_dev_m(list<int> &l, int m);`** que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\text{max\_dev\_m(L)} = \max\{\text{max\_dev}(a_1, a_2, \ldots, a_m), \text{max\_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max\_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max\_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces **`max_dev_m(L,3)`** debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones.

b) **[ord-nodo]**
Escribir una función predicado **`bool ordnodo(tree<int> &A);`** que verifica si cada secuencia de hermanos del subárbol del nodo **`n`** (perteneciente al árbol ordenado orientado **`A`**) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol **`(3 5 (6 1 3) (7 4 5))`** debería retornar **`true`**, mientras que para **`(3 9 (6 1 3) (7 4 2))`** debería retornar **`false`**, ya que las secuencias de hermanos **`(9 6 7)`** y **`(4 2)`** NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

c) **[flat]** Se está diseñando una red inerconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es **`n`** y tenemos un **`vector< set<int> > sw`** que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir **`sw[j]`** es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Consigna: Escribir una función predicado **`bool flat(vector< set<int> > &sw, int n);`** que retorna verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjunto en **`sw[]`**.



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verficarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector **`sw`** sería

$$\texttt{sw[0]} = \{0, 1, 2, 3, 4\}, \quad \texttt{sw[1]} = \{0, 1, 5, 6, 7\}, \quad \texttt{sw[2]} = \{2, 3, 4, 5, 6, 7\} \quad (2)$$

Por lo tanto **`flat(sw,8)`** debe retornar **`true`**. Por otra parte si tenemos

$$\texttt{sw[0]} = \{0,2,3,4\},\ \ \texttt{sw[1]} = \{0,1,5,7\},\ \ \texttt{sw[2]} = \{2,3,5,6,7\} \tag{3}$$

entonces los pares $(0,6)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,6)$, $(4,5)$, $(4,6)$ y $(4,7)$ no están conectados en forma directa y **`flat(sw,8)`** debe retornar **`false`**.
Sugerencia 1: Recorrer todos los pares de valores (j,k) y para cada par recorrer todos los conjuntos en **`sw[]`** hasta encontrar uno que contenga al par.
Sugerencia 2: Puede ser de ayuda el escribir una función auxiliar
`bool estan_conectados(sw,j,k)`.

[Ej. 4] [operativos (mínimo 70 %)]

a) **[rec-arbol]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- ORD_PRE $=\{C,Z,Q,R,A,M,P,K,L,T\}$,
- ORD_POST $=\{Z,A,P,M,K,L,R,T,Q,C\}$.

b) **[make-tree]** Escribir la secuencia de sentencias necesarias para construir el arbol binario **`(5 (2 . 3) (7 6 8))`**. (**Restricciones:** No usar **`find()`**).

c) **[quick-sort]:** Dados los enteros 3, 7, 9, 11, 10, 12, 6, 5, 7, 4 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.

d) **[huffman]:** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario (para todos los caracteres) y encodar la palabra ARGENTINA: $P(A) = 0.24$, $P(R) = 0.12$, $P(G) = 0.10$, $P(E) = 0.10$, $P(N) = 0.10$, $P(T) = 0.14$, $P(I) = 0.08$, $P(S) = 0.06$, $P(V) = 0.06$. Calcular la longitud promedio del código obtenido.

e) **[hash-dict]** Insertar los números $2, 15, 25, 8, 7, 35, 17, 4, 27$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x \bmod 10$ y estrategia de redispersión lineal.