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.
Recuperatorio. [2014-11-21]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 3 y 6) 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, apellido, y nombre, ASI**:

CLAS1, Hoja #2/3 LOVELACE, ADA

[Ej. 1] [CLAS1 (W=30pt)]
Recordar que **deben usar la interfaz STL**.

a) **[list (10pt)]** Escribir la implementación en **C++** del TAD lista (clase **`list`**) implementado por punteros **ó** cursores. Los métodos a implementar son **`insert(p,x)`**, **`erase(p)`**, **`iterator::operator++(int)`**.

b) **[stack-queue (10pt)]** Escribir la implementación en **C++** de los métodos **push**, **pop**, **`front`**, y **top** de los TAD pila y cola (clases **`stack`** y **queue**), según corresponda.

c) **[map (10pt)]** Escribir la implementación en **C++** del TAD correspondencia (clase **`map`**) implementado con **vectores ordenados**. Métodos a implementar: **`find(key)`** (en caso de utilizar la función privada **`lower_bound()`** también implementarla).

[Ej. 2] [PROG1 (W=50pt)]

a) **[qsortl (20pt)]**
Escribir una función **`int qsortl(list<int> &L, bool (*comp)(int,int));`** que implementa el algoritmo de ordenamiento **quick-sort** para listas de la siguiente manera

- Toma como pivote **`v`** el primer elemento de la lista.
- Separa **`L`** en tres listas: **`L1,L2,L3`** los estrictamente menores a **`v`**, equivalentes a **`v`**, y mayores a **`v`**.
- Aplica recursivamente **`qsortl()`** a **`L1`** y **`L3`**.
- Concatena **`L1,L2,L3`** en **`L`**, en ese orden.

Nota: El concepto de particionamiento es el mismo que el que hemos visto en el capítulo de ordenamiento para vectores, sin embargo acá como es una lista no hace falta hacer intercambios de elementos, simplemente se van tomando los elementos de la lista y se van insertando en la lista auxiliar apropiada.

Restricciones: Usar sólo listas. Hacer la operación in place (usar sólo $n + O(1)$ celdas), donde n es el tamaño de la lista original.

b) **[apply-map (15pt)]**
Escribir una funcion **`void apply_map(list<int> &L, map<int,int> &M, list<int> &ML)`** que, dada una lista **`L`** y una correspondencia **`M`** retorna por **`ML`** una lista con los resultados de aplicar **`M`** a los elementos de **`L`**. Si algún elemento de **`L`** no está en el dominio de **`M`** entonces el elemento correspondiente de **`ML`** no es incluido. Por ejemplo, si

`L= (1,2,3,4,5,6,7,1,2,3)`, y `M = { (1,2),(2,3),(3,4),(4,5),(7,8) }` entonces después de hacer `apply_map(L,M,ML)`, debe quedar `ML = (2,3,4,5,8,2,3,4)`

Restricciones: No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$, donde n es el número de elementos en la lista (asumiendo que las operaciones usadas de correspondencia son $O(1)$).

c) **[gather (15pt)]**
Escribir una función `void gather(list<int> &L, int M);` que, dada una lista de enteros `L`, agrupa elementos consecutivos de tal manera que en la lista queden solo elementos mayores o iguales que `M`. El algoritmo recorre la lista y, cuando encuentra un elemento menor, empieza a agrupar el elemento con los siguientes hasta llegar a `M` o hasta que se acabe la lista. Por ejemplo, si `L=[3,4,2,4,1,4,4,3,2,2,4,1,4,1,4,4,1,4,4,2]`, entonces `gather(L,10);` da `L=[13,12,13,10,10]`. En la lista final NO deben quedar elementos menores que `M` salvo eventualmente el último elemento.
Restricciones: Usar sólo listas y hacerlo *in-place* es decir usar sólo $n + O(1)$ celdas.

[Ej. 3] [PREG1 (W=20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$
\begin{aligned}
T_1 &= n^2 + 5 \cdot 3^n + 4^{10}, \\
T_2 &= 2 \cdot 10^n + \sqrt{3} \cdot n + \log_8 n, \\
T_3 &= 1000 + 3 \log_4 n + 8^2 + 5 \log_{10} n, \\
T_4 &= 5 \cdot 2^n + 2n^3 + 3\, n! \\
T_5 &= 2.3 \log_8 n + \sqrt{n} + 5\, n^2 + 2\, n^5,
\end{aligned}
$$

b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `list<>` implementada con **celdas simplemente enlazadas** (por punteros o cursores) en el caso promedio? *Métodos:* `insert(p,x)`, `*p`, `erase(p)`, `clear()`, `begin()`, `end()`.

c) Explique la propiedad de **transitividad** de la notación asintótica $O(\,\cdot\,)$. De un ejemplo.

d) ¿Qué ocurre si ejecutamos el siguiente código? `S` es una pila de enteros que puede estar vacía o no.

```
S.push(3); S.push(5); S.top(); S.top(); x=S.top();
```

¿Puede dar un error porque `S` se queda vacía? ¿Que valor toma `x`?

e) Discuta ventajas y desventajas de usar contenedores lineales **ordenados** o **desordenados** para representar correspondencias.

[Ej. 4] [CLAS2 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[aoo (5pt)]**
Arbol Ordenado Orientado: Implementar (haciendo uso de la interfaz STL) los métodos

- `iterator lchild();`
- `iterator right ();`

- `iterator insert (iterator p, T t);`

considerando que se debe escribir la clase **`iterator`** (en el scope que corresponda) además de aquellas estructuras sobre las cuales esta se construye (a saber, la clase **`cell`**).

b) **[hash-dict (5pt)]**
Diccionario por tablas de dispersión abierta: Implementar la clase **`iterator_t`** en conjunto con aquello que considere pertinente dentro de la parte privada de la clase **`hash_set`**. Finalmente, implemente el método **`iterator_t find (key_t & x)`**

c) **[vbitset (5pt)]**
Conjuntos por vectores de bits: Implementar los métodos

- `pair_t insert (elem_t x);`
- `elem_t retrieve (iterator_t p);`
- `void set_difference (set & A, set & B, set & C);`

para ello deberá además implementar la parte privada de la clase.

d) **[abb (5pt)]**
Conjuntos por árbol binario de búsqueda: Implementar (haciendo uso de la interfaz STL) el método

- `void erase (iterator_m)`

definiendo para ello (y respetando el anidamiento pertinente) la clase iterator.

[Ej. 5] [OPER2 (W=20pt)]

a) **[rec-arbol (2.5pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- **`ORD-PRE=(Q,R,S,U,V,W,X,Y,T)`**,
- **`ORD-POST=(R,V,W,Y,X,U,S,T,Q)`**,

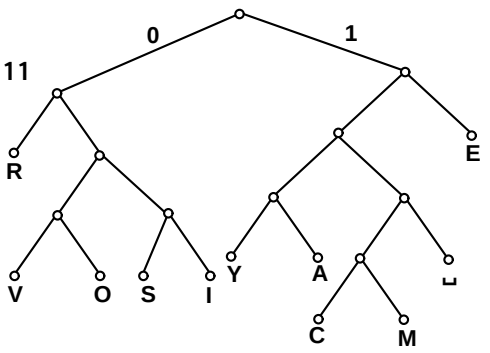
b) **[part-arbol (2.5pt)]** Dado el árbol ordenado orientado (AOO): **`(Z (Q T) (R W) (Y (Q A C)))`** determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo **`R`**. ¿Son **disjuntos**? Justifique.

c) **[huffman (2.5pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`LOLLAPALOOZA`**, $P(L) = 0.10$, $P(O) = 0.10$, $P(A) = 0.10$, $P(P) = 0.30$, $P(Z) = 0.10$, $P(Q) = 0.10$, $P(M) = 0.20$.

Calcular la longitud promedio del código obtenido.

d) **[hf-decode (2.5pt)]**
Utilizando el código de la derecha desencodar el mensaje

`01101010010010010001011101010101000011111`



e) **[abb (2.5pt)]** Dados los enteros $\{16, 10, 23, 5, 6, 13, 8, 7, 6, 15, 4\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 16, 10, y 5 en ese orden.

f) **[hash-dict (2.5pt)]** Insertar los números $\{9, 15, 25, 8, 7, 35, 17, 4\}$ en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión `h(x) = x`.

g) **[heap-sort (2.5pt)]** Dados los enteros $\{1, 10, 8, 2, 3, 13, 5\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

h) **[quick-sort (2.5pts)]** Dados los enteros $\{10, 4, 11, 6, 3, 5, 12, 4, 9, 7, 6, 2\}$ ordenarlos por el método de **clasficación 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.

[Ej. 6] [PREG2 (W=20pt, 4pt por pregunta)]

a) Si queremos generar un código binario de igual longitud para un conjunto de 26 caracteres. ¿Cuantos bits tendrá, como **mínimo**, cada caracter?

b) ¿Como se define la **altura** de un nodo en un árbol? ¿Cuál es la altura de los nodos `A`, `B`, y `C` en `(W X Y (A B (C (D  F G H))))`?

c) ¿Cómo se define la clase **iterator** para árboles binarios? Enumere sus campos dato y discuta la utilidad de cada uno de ellos. De ejemplos.

d) ¿Cómo se define en forma recursiva el **orden posterior** para un árbol ordenado orientado (AOO)?

e) ¿Cuál es el resultado de aplicar la función `l=particiona(w,j,k,v)`? (**Nota:** No se pide el algoritmo, sino cuál es el efecto de aplicar tal función, independientemende de como se programe). Explique los argumentos de la función. ¿Cuál debe ser el tiempo de ejecución para `particiona()` si queremos que `quick_sort()` sea un algoritmo **rápido**?