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.
3er Parcial. [2012-11-22]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)] Insistimos: deben usar la interfaz STL.

a) **[set-vbit (6pt)]**
- Escribir una función `void vbset_difference(vector<bool> &A,vector<bool> &B,vector<bool> &C);` que implementa la operación binaria diferencia de conjuntos ($C = A - B$), para la representación por vectores de bits. Asuma que `A`, `B`, `C` tienen la misma longitud.
- Escribir una función `void vbset_clear(vector<bool> &A);` que elimina todos los elementos de un conjunto `A` representado con vectores de bits.

b) **[set-oht (7pt)]** Implementar una función `pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y la relación de equivalencia `equal` retornando un par de `int` (que indica la cubeta) y `bool` que indica si la inserción fue exitosa.

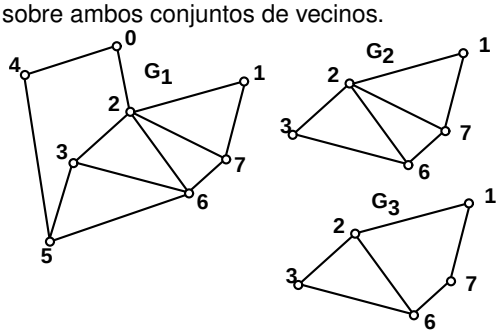
c) **[set-abb (7pt)]**
- Escribir la función `btree<int>::iterator abb_find(btree<int> &T,int x);` que devuelve el iterator en el ABB `T`, donde se encuentra el elemento `x`, o caso contrario `T.end()`. **Restricción:** La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.
- Escribir la función `pair<int,int> minmax(btree<int> &T);` que devuelve el mínimo y el máximo de el ABB `T`. Si el árbol está vacío debe devolver las constantes `(INT_MAX,INT_MIN)`. **Restricción:** La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.

[Ej. 2] [Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.

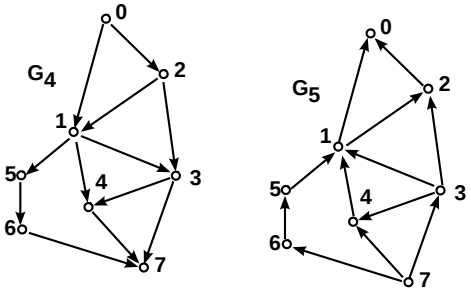
Atención: Hay 4 ejercicios cuya suma es **60**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,a+b+c+d)`).
Nota: En los ejercicios con grafos siguientes, los grafos no tienen bucles y se representan como un `map<int, set<int> >G`, donde `G[j]` es el conjunto de vértices adyacentes (esto es, conectados por una arista) con el vértice `j`. A continuación el tipo `graph_t` está definido como `typedef map<int, set<int> > graph_t`.

a) **[is-subgraph (20pt)]** Dados dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ determinar si G_2 es un subgrafo que se obtiene dejando un cierto conjunto de vértices $V_2 \subseteq V_1$ y **todas** las aristas de E_1 que conectan nodos de V_2.
Consigna: Escribir una función `bool issubgraph(graph_t &G1,graph_t &G2);` que realiza la tarea indicada.
Si lo aplicamos a los grafos de abajo entonces debemos tener `issubgraph(G1,G2) -> T` y `issubgraph(G1,G3) -> F` ya que la arista $(2, 7)$ está en `G1` pero no en `G3`.
Ayuda: Primero chequear que los vértices de `G2` están incluidos en `G1`. Después chequear que para vértice en `G2` sus vecinos son exactamente la intersección de sus vecinos en `G1` y el conjunto de vértices de `G2`. Esta

última operación se puede hacer o bien utilizando las operaciones binarias de conjuntos (con conjuntos auxiliares) o bien iterando sobre ambos conjuntos de vecinos.



b) **[dag-descendants (20pt)]** Dados un **Grafo Dirigido Acíclico (DAG)** $G = (V, E)$ y un subconjunto de vértices $W \subseteq V$, determinar el conjunto $D \subseteq V$ de **descendientes** de W, es decir $d \in D$ si y sólo si existe un camino que va de algún nodo $w \in W$ a d. En el grafo de abajo los descendientes de $W = \{5, 3\}$ en el grafo G_4 son $D = \{5, 3, 4, 6, 7\}$.
Consigna: Escribir una función `void dag_desc(graph_t &G,set<int> &W, set<int> &D);` que realiza la tarea indicada.



c) **[reverse-dag (10pt)]** Dados un **Grafo Dirigido Acíclico (DAG)** $G = (V, E)$ obtener el DAG $G' = (V, E')$ tal que si $(v, w) \in E$ entonces $(w, v) \in E'$. (Es decir, equivale a invertir el sentido de las flechas en el dibujo). Por ejemplo en los grafos de arriba aplicamos a G_4 debe dar G_5 y vicersa. **Consigna:** Escribir una función `void reverse_dag(graph_t &Gin,graph_t &Gout);` que realiza la tarea indicada.

d) **[is-mapped (10pt)]** Dados dos conjuntos `set<int> A,B` y una función **biyectiva** `int f(int)` determinar si es `B=f(A)` es decir **todos** los elementos de `B` se obtienen como imagen de `A` aplicando `f`.
Consigna: Escribir una función `bool ismapped(set<int> &A,set<int> &B,int (*f)(int));`
Restricción: el algoritmo debe ser **in-place** (no usar estructuras auxiliares) y **no-destructivo**, de hecho, no debe modificar ni `A` ni `B`.
Ayuda: Como `f()` es biyectiva, basta con determinar si el tamaño de ambos conjuntos es el mismo, y aplicar la función a los elementos de `A` y chequear que estén en `B`.

[Ej. 3] [operativos (total 20pt)]

a) **[abb (5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 2, 11\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 12, 6 y 1 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $\{0, 16, 26, 9, 8, 36, 16, 18, 5\}$ en una **tabla de dispersión cerrada** con $B = 8$ cubetas, con función de dispersión `h(x) = 2*x+3` y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{0, 4, 7, 1, 2, 12, 9\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ 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. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión** (`merge-sort`). ¿Es estable? ¿Bajo que condiciones?

b) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido** (`quick-sort`), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?

c) ¿Cuáles son las condiciones que debe satisfacer un predicado binario para ser una **relación de orden débil**?

d) Se quiere representar subconjuntos del conjunto universal de los cubos perfectos en $[-1000, 1000]$, o sea

$$\begin{aligned} U &= \{-1000, -729, -512, \ldots, -27, -8, 0, 8, 27, \ldots, 512, 729, 1000\}, \\ &= \{j \in \mathbb{Z} \,/\, \exists\, k \in \mathbb{Z} \text{ con } j = k^3\}, \end{aligned} \tag{1}$$

como un conjunto con **vectores de bits**.

1) ¿Cuál es el tamaño del conjunto universal?
2) Escribir las funciones `int indx(int)` y `int element(int)` correspondientes, que mapean el espacio de elementos del conjunto universal a índices y viceversa.

Ayuda: asumir la existencia de una función `int cbrt(int)` que calcula la raíz cúbica entera de un número entero (si es un cubo perfecto). Si el argumento no es un cubo perfecto, retorna un valor indefinido.
Nota: al escribir la función `index()` se debe chequear que el argumento sea un cubo perfecto. Es decir, si no es un cubo perfecto en el rango indicado debe dar un error (llamando a `abort()` o lanzando una excepción).

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**?