Algoritmos y Estructuras de Datos.
3er Parcial. [21 de Noviembre de 2008]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (30pt), mínimo 50 %]

a) Escribir una función `btree<int>::iterator abb_insert(btree<int> &T,int x);` que realiza la inserción de `x` en un ABB `T`. La función retorna el iterator donde está el elemento después de la inserción (sea esta exitosa o no).

b) Escribir una función `bool open_hash_erase(vector<list<int>> &H,int (*h)(int),int x);` que elimina el elemento `x` de la tabla de dispersión abierta `H`, representada como un vector de listas **ordenadas**. El valor de retorno indica si la eliminación fue exitosa.

c) 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.

[Ej. 2] [programacion (total = 30pt, mínimo 40 %)]

a) **[connected-to (15 pt)]** Recordemos que dado un grafo no dirigido $G = (V, E)$ donde V son los vértices y E las aristas del grafo, una componente conexa de G es un subconjunto V_i de V tal que cualquier par de nodos en V están conectados entre sí, es decir existe un camino entre los dos vértices.
Consigna: Escribir una función `void connected_to(map<int,set<int>> &G,int x,set<int> &W);` que calcula la componente conexa `W` del nodo `x`, es decir el conjunto de nodos `y`, para los cuales existe un camino en `G` desde `x` hasta `y`. *Por ejemplo,* si `G={1->{2},2->{1},3->{4},4->{3}}`, entonces si `x=1`, `W={1,2}` y si `x=4` entonces `W={3,4}`.
Sugerencia: Mantendremos dos conjuntos `W` (los vértices ya visitados) y `F` el frente de `W` que esta avanzando. Inicialmente `W=F={x}`. Ahora para calcular el siguiente frente hacemos

$$
\begin{aligned}
\mathtt{Q} &= \bigcup_{n \in \mathtt{F}} G[n], && \text{(vecinos de } \mathtt{F}\text{)} \\
\mathtt{F} &= \mathtt{Q} - \mathtt{W}, && \text{(nuevo frente)} \\
\mathtt{W} &= \mathtt{W} \cup \mathtt{F}, && \text{(actualiza } \mathtt{Q}\text{)}
\end{aligned}
\tag{1}
$$

El algoritmo termina cuando `F` es el conjunto vacío.

b) **[eq-class (15 pt)]**
Recordemos que dada una relación de order estricta y débil $<$ en un conjunto universal U, dos elementos $a, b \in U$ son equivalentes ($a \equiv b$) si no es verdad que $a < b$ ni $b < a$. Esto separa a cualquier conjunto $S \subset U$ en conjuntos disjuntos $E_i \subset U$ llamados *"clases de*

equivalencia" tales que a, b son equivalentes entre sí, si y solo si a, b pertencen al mismo E_i.
Consigna: Escribir una función
`void eqclass(set<int> &S, bool (*comp)(int x,int y), list<set<int>> &L);` que dado un conjunto `S` y una relación de orden `comp()` devuelve sus clases de equivalencia en `L`. *Por ejemplo*, si `S={-2,-1,0,1,2}` y `comp()` es la relación *"menor en valor absoluto"* entonces debe retornar `L=({0},{-1,1},{-2,2})`.
Sugerencia: Para cada elemento `x` de `S`, recorrer los conjuntos en `L` hasta encontrar la posición donde está o debería estar su clase de equivalencia. (Este algoritmo es similar a `lower_bound(x)`). Si la clase de equivalencia ya está, entonces simplemente se agrega `x` a esta clase. Si no, se inserta en la lista un nuevo conjunto con el elemento `x`.

[Ej. 3] [operativos (total 20 pt), mínimo 25 %]

a) **[abb (5 pts)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 1, 12\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 13, 5 y 2 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $3, 16, 26, 9, 8, 36, 18, 5, 28$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x \bmod 9$ y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{1, 5, 8, 2, 3, 13, 10\}$ 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 $\{5, 9, 4, 1, 5, 10, 8, 3, 3, 2, 11, 6\}$ 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 = 20 pt, mínimo 60 %)]

a) ¿Cuáles son las 3 etapas básicas en el ordenamiento por fusión (o *"merge sort"* o por intercalamiento)?. Explique de manera breve y simple cada una de ellas.
b) Explique una estrategia para elegir el pivot en quicksort.
c) ¿Cuál es el tiempo de ejecución para `insert(x)` en el TAD conjunto por tabla de dispersión cerrada, en el caso promedio?. ¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?
d) ¿Cuál es la función del procedimiento *"re-heap"* en montículos?.
e) Explique el concepto de estabilidad para algoritmos de ordenamiento. De ejemplos de algoritmos de ordenamiento estables y no estables.