Primer Parcial de Programación 3 (30/09/2011)

Instituto de Computación, Facultad de Ingeniería

- Este parcial dura **4** horas y contiene 5 carillas. El total de puntos es **40**.
- En los enunciados llamamos C* a la extensión de C al que se agrega el operador de pasaje por referencia &, y las sentencias *new*, *delete* y el uso de *cout* y *cin*.
- NO se puede utilizar ningún tipo de material de consulta. Salvo que se indique lo contrario usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.

Se requiere:

i. Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
ii. Utilizar las hojas de un sólo lado y escribir con lápiz.
iii. Iniciar cada ejercicio en hoja nueva.
iv. Poner en la carátula la cantidad de hojas entregadas, y un índice indicando en qué hojas respondieron cada problema.

Ejercicio 1 (14 puntos)

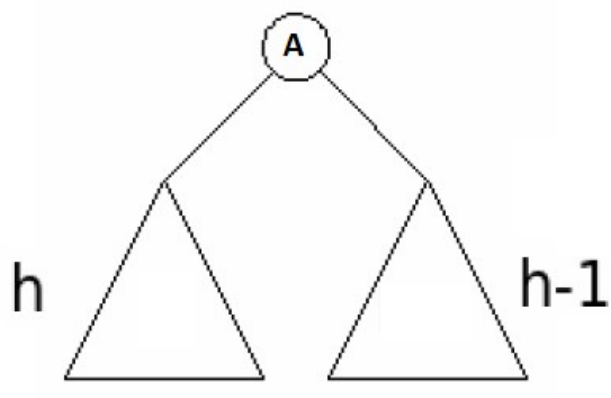
Parte 1) (5 puntos) Para el siguiente algoritmo calcule el costo en los casos: mejor y peor considerando las operaciones simples: suma, asignación y comparación con costo 1. Tenga en cuenta dichas operaciones tanto sobre las variables de control como los elementos del arreglo.

```
1     void algoritmo(int array[], int n)
2     {
3           int i,j;
4           for(i=0; i < n-1; i = i + 1)
5                 swap(array[i], array[n-1-i]);

6           for(i=0; i < n-1; i = i + 1)
7                 for(j=0; j < n-(i+1); j = j + 1)
8                       if(array[j] > array[j+1])
9                             swap(array[j], array[j+1]);
10    }

11    void swap(int &x, int &y)
12    {
13          int temp;
14          temp = x;
15          x = y;
16          y = temp;
17    }
```

Parte 2) (4 puntos)
Determine si son correctas las siguientes afirmaciones realizando una demostración **detallada** de la veracidad o falsedad de las mismas. Para esta parte puede utilizar propiedades vistas en el curso, enunciándolas correctamente antes de aplicarlas.

i) $n! \in \Theta((n!)!)$
ii) $n! \in O((n!)!)$
iii) $100n^5 + 23n^4 \in \Theta(3n^5 + (n+1)^2)$

Parte 3) (5 puntos)
Dado el siguiente árbol AVL, con *h >= 0*



discuta todos los casos en los cuales la inserción de un elemento requiera una rotación LR teniendo como nodo raíz de la rotación el vértice A. En cada caso indique como están los factores de balanceo antes y luego de la rotación / rebalanceo.

Nota: recordar que la altura de un árbol vacío es -1.

Ejercicio 2 (12 puntos)

Un video club maneja información acerca de las películas que el mismo tiene, de los socios que pertenecen al mismo, así como información relacionada a los alquileres de las películas.

De cada película se conoce su título, director, género y código. Una película se identifica por su código y el video club tiene **P** películas. A modo de simplificación, el video club cuenta con un único ejemplar de cada película.

De cada socio se conoce su código de socio del video club, su dirección, teléfono y conjunto de películas que tiene alquiladas en un momento dado. Un socio se identifica por su código de socio y el video club tiene **S** socios. En un momento dado, un socio puede tener alquiladas **A** películas.

A su vez, para cada película se registra la cantidad de veces que un mismo socio alquiló la misma.

Llamaremos *VideoClub* a la estructura de datos utilizada para representar la realidad anterior. Se desean realizar las siguientes operaciones:

1. *void imprimirPeliculasAlquiladas(VideoClub V)*
   Dada *V*, imprime para cada película que se encuentra alquilada, su título y género. La información de cada película debe ser impresa ordenada cronológicamente según cuando se alquiló la película. En caso de que no exista ninguna película alquilada al momento de invocar la operación se imprime el siguiente mensaje: “No hay películas alquiladas en este momento.”. Esta operación debe realizarse en O(P) promedio.

2. *void imprimirPeliculasAlquiladasDeUnSocio(String codigoSocio, VideoClub V)*
   Dada *V*, imprime la información de las películas alquiladas del socio de código *codigoSocio*. De cada película se imprime su código y director, ordenados en forma ascendente por el código de película. Esta operación debe realizarse en O(A) promedio.

   Pre-condición: el socio de código *códigoSocio* pertenece a *V*.

3. *void devolverPeliculaAlquilada(String codigoPelicula, String codigoSocio, VideoClub &V)*
   Dada *V*, el código de la película *codigoPelicula* y el código del socio *codigoSocio*, se devuelve la película al video club. Esta operación debe realizarse en O(logA) promedio.

   Pre-condiciones: la película de código *codigoPelicula* y el socio de código *codigoSocio* pertenecen a *V* y la película de código *codigoPelicula* la tiene alquilada el socio de código *codigoSocio* al momento de invocar la operación.

4. *void imprimirHistorialDeUnaPeliculaParaUnSocio(String codigoPelicula, String codigoSocio, VideoClub V)*
   Dada *V*, el código de la película *codigoPelicula* y el código del socio *codigoSocio*, imprime la cantidad de veces que el socio alquiló la película. En caso de que el socio nunca haya alquilado la película se imprime 0. Esta operación debe realizarse en O(1) promedio.

   Pre-condiciones: la película de código *codigoPelicula*, y el socio de código *codigoSocio* pertenecen a *V*.

a) (**6 puntos**) Diseñe una estructura (haga un dibujo con los comentarios pertinentes) que permita soportar las operaciones mencionadas con los órdenes de ejecución indicados. Explique detalladamente su estructura y justifique los órdenes de ejecución para cada operación.

b) (**6 puntos**) Suponga ahora que el video club permite efectuar reservas de películas. De una reserva, interesa saber la fecha en que el socio realizó la reserva de la película.

   Discuta los cambios a la solución propuesta en la parte anterior para resolver eficientemente las siguientes nuevas operaciones, **sin afectar los órdenes de las operaciones anteriores**.

   b.1) Indique el orden en el **peor caso** para la siguiente operación:

void alquilarPeliculaAUnSocio(String codigoPelicula, String codigoSocio, VideoClub &V)
Dada *V*, el código de la película *codigoPelicula* y el código del socio *codigoSocio*, se alquila la película de código *codigoPelicula* al socio de código *codigoSocio*.

El alquiler de la película será exitoso, si la película no se encuentra alquilada al momento de invocar esta operación, y además, si la película a alquilar se encuentra reservada, el alquiler de la misma solo será exitoso, si el socio que intenta alquilar la película, tiene hecha una reserva y a su vez, es él que tiene la reserva más antigua para dicha película. Si el socio tenía una reserva, al momento de alquilar la película la misma deberá ser borrada.

Pre-condiciones: la película de código *codigoPelícula* y el socio de código *codigoSocio* pertenecen a *V*. Al momento de invocar esta operación asuma que el socio no tiene A películas alquiladas.

Debe indicar paso a paso como se resuelve la operación (pseudocódigo, NO implementar).

b.2) Indique el orden en el **caso promedio** para la siguiente operación:

void reservarPeliculaAUnSocio(String codigoPelicula, String codigoSocio, String fecha, VideoClub &V)
Dada *V*, el código de la película *codigoPelicula* y el código del socio *codigoSocio*, se reserva la película de código *codigoPelicula* al socio de código *codigoSocio*.

Dado un socio, en un momento dado, éste no puede tener más de una reserva a una misma película.

Pre-condiciones: la película de código *codigoPelícula* y el socio de código *codigoSocio* pertenecen a *V*. Al momento de invocar esta operación, asuma que la película está alquilada.

Ejercicio 3 (14 puntos)

Se desea implementar un algoritmo de búsqueda regional para una red P2P llamada "FingMule". Cuando un cliente (*peer*) de FingMule es iniciado, se obtiene la ubicación (IP) de otros K *peers* ya miembros de la red y localizados cerca de este; a los cuales se conectará para poder compartir y buscar contenidos en toda la red.

Cuando se realiza una búsqueda en un *peer* (e.g. título de un libro), este envía la cadena de caracteres a sus *peers* inmediatos (i.e. conectados directamente a él) de modo que cada uno de ellos busque si posee dicho contenido. En caso de existir, se enviará la IP del peer como respuesta al origen para ser desplegada. Luego de chequear sus contenidos, cada peer reenvía la consulta a sus peers inmediatos que no hayan sido consultados, de modo de repetir el procedimiento sucesivamente.

Para acotar la sobrecarga de la red, a cada búsqueda se le indica una distancia máxima H, que representa la cantidad de veces que puede ser reenviada a través de *peers*. Es decir, cuando una búsqueda con distancia H es reenviada a un *peer* (*hop*), el receptor recibirá H-1 como distancia máxima para reenviarla (i.e. *hops* disponibles), no pudiendo ser reenviada cuando ésta llegue a 0.

Una vez terminada la búsqueda, el cliente despliega la lista de *peers* que poseen el contenido deseado para que el usuario seleccione uno de ellos y comience la descarga. Al iniciarla, se utiliza algún camino de costo mínimo, pero en paralelo y a modo de prevención, el peer origen busca y almacena hasta C caminos (ignorando costos) desde él, al peer fuente de la descarga. De este modo contará con alternativas inmediatas en caso de pérdida de conexión.

Se pide:

1) Implementar un emulador de la búsqueda en FingMule que: dado un peer origen, una cadena y una distancia máxima, emula el comportamiento de la búsqueda retornando la IP de los *peers* acorde a lo establecido, ordenados ascendentemente por distancia desde el origen.
   ListaNodos BuscarContenido(int origen, char cadena, int hops)*
2) Implementar el método de *CaminosAlternativos(int origen, int fuente)*, que dados los peers *origen* y *fuente*, busca y guarda hasta C caminos simples entre ellos.

Notas:

- Puede utilizar los TAD: **Grafo, Lista y Cola** de Enteros vistos en el curso.
- Puede asumir que en el origen (donde se inicia la búsqueda) no hay contenidos disponibles.
- Asuma la existencia de:
  - Todas las **constantes** mencionadas.
  - **Grafo *red*** //variable global que modela la red P2P.
  - ***bool existeContenido(char* cadena, int peer)*** //indica la existencia de contenido correspondiente a *cadena* en *peer*.
  - ***int getIP(int peer)*** // dado el identificador de un nodo retorna su IP en tipo *int*.
  - ***void guardarCamino(Lista camino)*** //guarda el camino indicado en *camino*.
- ***Cualquier otra operación invocada deberá ser implementada.***