Primer Parcial de Programación 3 (30/09/2010)

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 (13 puntos)

Un grafo no-dirigido se dice que es *bipartito completo* si el grafo es bipartito(*) y además sus aristas coinciden con todas las posibles que conectan todo par de vértices entre los dos conjuntos. Es decir, un grafo $G = (V = V_1 \cup V_2, A)$ es bipartito completo si para todo $v_1, v_2 \in V$ tal que $v_1 \in V_1$ y $v_2 \in V_2$ se cumple que $(v_1, v_2) \in A$ y esas son todas las aristas.

Dado un grafo no-dirigido $G(V, A)$ determinar si es bipartito completo mediante la aplicación de BFS:

a) (**3 puntos**) Describa brevemente como se podría resolver el problema.
b) (**10 puntos**) Implemente un algoritmo en C* que lo resuelva según a).

Nota: Asuma que los vértices de G están identificados por enteros $\in$ [0..N-1] donde N=|V|. Puede utilizar los TAD Grafo y Lista de Enteros provistos.

Sugerencia: Sea $T(V, A_T)$ el grafo resultante del recorrido BFS, analizar la existencia de aristas del mismo nivel en el conjunto $A - A_T$.

(*) Un Grafo es bipartito si sus vértices se pueden particionar en dos conjuntos y las aristas siempre unen vértices de un conjunto con vértices del otro conjunto.

TAD Grafo

```
Grafo construirGrafo (int cantNodos);
// Construye un grafo con 'cantNodos' nodos.
void agregarAristaGrafo (Grafo& grafo, int nodoOrigen, int nodoDestino, int costo);
// Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
//  Se agrega una arista a 'grafo'.
int costoAristaGrafo (Grafo & grafo, int nodoOrigen, int nodoDestino);
// Precondicion: existe la arista [nodoOrigen, nodoDestino] en el grafo.
//  Devuelve el costo de la arista [nodoOrigen, nodoDestino].
bool existeAristaGrafo (Grafo grafo, int nodoOrigen, int nodoDestino);
// Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
// Retorna true sii existe una arista desde 'nodoOrigen' hacia
// 'nodoDestino' en el grafo.
int cantNodosGrafo (Grafo grafo);
// Retorna la cantidad de nodos del grafo.
Lista adyacentesNodoGrafo (Grafo grafo, int nodo);
// Precondicion: 0 <= nodo < cantNodosGrafo (grafo)
// Retorna la lista id de nodos adyacentes a 'nodo' del grafo.
void destruirGrafo (Grafo & grafo);
// Libera la memoria asociada a la estructura.
```

TAD Lista de Enteros

```
Lista crearLista();
// Crea la lista vacia.
void listaInsertar (Lista l, int n);
// Agrega el entero n al principio l.
void listaInsertarUltimo (Lista l, int n);
// Agrega el entero n al final de l.
bool listaEstaVacia (Lista l);
// Retorna true sii l está vacía.
bool listaPerteneceElem (Lista l, int n);
// Retorna true sii el elemento n pertenece a l.
void listaSacar (Lista l);
// Saca el primer elemento de l.
// Precondicion: !ListaEstaVacia (l).
void listaSacarUltimo (Lista l);
// Saca el ultimo elemento de l.
// Precondicion: !ListaEstaVacia (l).
int listaPrimero (Lista l);
// Retorna el primer elemento de l.
// Precondicion: !ListaEstaVacia (l).
int listaUltimo(Lista l);
// Retorna el ultimo elemento de l.
// Precondicion: !ListaEstaVacia (l).
void listaDestruir (Lista l);
// Libera la memoria de l.
```

Ejercicio 2 (13 puntos)

Una policlínica maneja información acerca de las reservas de las consultas de los pacientes a atenderse con los distintos médicos que trabajan en la misma, según el **año actual**.

De cada médico se conoce su especialidad y código de médico. Un médico se identifica por su código de médico, y la policlínica tiene como máximo **M** médicos.

De cada paciente se conoce su nombre, cédula de identidad y edad. Un paciente se identifica por su cédula de identidad, y la policlínica tiene como máximo **P** pacientes.

Dado un día, mes y médico se registra el conjunto de pacientes que éste tiene para atender. Asuma que las reservas de las consultas de los pacientes son ingresadas en el mismo orden histórico en que suceden. Al momento que un paciente reserva una consulta para un día, mes y médico, se ingresa el motivo de la consulta. Dado un día, mes del año y un médico, el médico como máximo atiende **A** pacientes siendo $A \leq P$.

Sea *E* la estructura de datos utilizada para representar la realidad anterior. Se desea realizar las siguientes operaciones:

1. ***void imprimirInformacionMedico(string codigoMedico, Estructura E)***
   Dada la estructura *E* y el código de médico *codigoMedico*, imprime la especialidad del médico. Esta operación debe realizarse en ***O(1)*** **promedio**.
   Pre-condición: existe un médico con código de médico *codigoMedico* perteneciente a la estructura *E*.

2. ***void imprimirInformacionPaciente(string ciPaciente, Estructura E)***
   Dada la estructura *E* y la cédula de identidad *ciPaciente* de un paciente, imprime el nombre y edad del paciente. Esta operación debe realizarse en ***O(1)*** **promedio**.
   Pre-condición: el paciente de cédula de identidad *ciPaciente* pertenece a la estructura *E*.

3. ***void imprimirMotivoConsulta(int día, int mes, string codigoMedico, string ciPaciente, Estructura E)***
   Dada la estructura E, imprime el motivo de la consulta del paciente de cédula de identidad *ciPaciente* a atenderse el día *dia* del mes *mes* con el médico de código *codigoMedico*. Esta operación debe realizarse en ***O(1)*** **promedio**.
   Pre-condiciones:
   - el paciente de cédula de identidad *ciPaciente* pertenece a la estructura *E*.
   - existe un médico con código de médico *codigoMedico* perteneciente a la estructura E.
   - existe una reserva para el médico con código *codigoMedico* del paciente con cédula de identidad *ciPaciente* para el día *dia* y mes *mes*.

4. ***void imprimirPacientesAAtenderPorUnMedico(int día, int mes, string codigoMedico, Estructura E)***
   Dada la estructura *E*, el día *dia* del mes *mes* y el código de médico *códigoMedico*, imprime los nombres de los pacientes a atenderse por el médico en ese día y mes,

ordenados en forma ascendente por *orden de atención*. En caso de que no haya pacientes a atenderse para ese día, mes y médico se imprime: "*Consulta vacía"*. Esta operación debe realizarse en ***O(A)*** **promedio**.
Pre-condición: existe un médico con código de médico *codigoMedico* perteneciente a la estructura E.

5. ***void imprimirMedicoConMasPacienteAAtender(int día, int mes, Estructura E)***
Dada la estructura *E*, el día *dia* del mes *mes*, imprime el código de médico junto a su especialidad del médico que tiene más pacientes a atender ese día y mes. En caso de que no haya consultas para ese día y mes se imprime: "*No hay ningún paciente a atenderse por ningún médico"*. En caso de que haya más de un médico que tenga la cantidad máxima de pacientes a atender, se imprime la información del médico que alcanzó el máximo primero. Esta operación debe realizarse en ***O(1)*** **peor caso**.

6. ***void reservarDiaParaUnMedico(int día, int mes, string codigoMedico, string ciPaciente, string motivoConsulta, Estructura &E)***
Dada la estructura *E*, reserva para el día *dia* del mes *mes* una consulta para el médico de código *codigoMedico* para el paciente de cédula de identidad *ciPaciente* con el motivo de consulta *motivoConsulta*. Esta operación debe realizarse en ***O(1)*** **promedio**.
Pre-condiciones:

- el paciente de cédula de identidad *ciPaciente* pertenece a la estructura *E*.
- existe un médico con código de médico *codigoMedico* perteneciente a la estructura E.
- no existe una reserva para el médico con código *codigoMedico* del paciente con cédula de identidad *ciPaciente* para el día *dia* y mes *mes*.

Se pide:

a) **(11 puntos)** Diseñe una estructura (**haga un dibujo** con los comentarios pertinentes) que permita soportar las operaciones con los órdenes de ejecución pedidos. A su vez, justifique porque se cumple con los órdenes de ejecución pedidos **para cada operación**.

b) **(2 puntos)** Discuta los cambios a la solución de la parte a) para resolver eficientemente la siguiente nueva operación, sin afectar los órdenes de las operaciones anteriores, y especifique el orden de esta nueva operación en el **peor caso**. **Si alguna operación de la parte a) se ve afectada, indique porque se sigue respetando el orden pedido.**

void imprimirPacientesDeUnMedico(string codigoMedico, Estructura E)
Dada la estructura *E*, imprime el nombre de los pacientes que reservaron al menos una vez en el año una consulta para el médico de código *codigoMedico*.
Pre-condición: existe un médico con código de médico *codigoMedico* perteneciente a la estructura E.

Ejercicio 3 (14 puntos)

Parte 1) (7 puntos)

a) (3 puntos) Para el siguiente algoritmo calcule el costo en los casos: mejor, peor y promedio tomando como **operación básica** la invocación al procedimiento *calcular()*.

La función *random()* retorna aleatoriamente ***true*** ó ***false***, donde la probabilidad de que retorne *true* es *1/3*. El resultado de cualquier invocación de *random()* es **independiente** a los anteriores.

```
for (int i = 1; i <= n; i++){

   for (int j = 1; j <= i * i; j++){

        if (j % i == 0) {
             bool valor = random();
             if (valor) {
                  calcular();
             }
        }
   }
}
```

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

i) $n! \in \Theta((2n)!)$

ii) $n! + 3n^2 \in \Theta(n!)$

c) (2 puntos) Dada la siguiente función arbitraria $f : N \to R^*$ tal que $f \in O(n)$ indique si es verdadera o falsa la siguiente afirmación **partiendo de la definición de O**:

$(f^2 + n) \in O(2n)$. En caso de ser verdadero o falso, demuestre su afirmación.

Parte 2) (7 puntos)

a) (4 puntos) Sea la secuencia de enteros [100, 50, 60, 40, 55, 30, 45, 52, 110, 90, 51, 56] partiendo del árbol vacío muestre paso a paso el árbol AVL resultado de la inserción de cada elemento de la secuencia. Indique qué tipo de rotación realiza en cada paso en caso de sea necesario junto a los factores de balanceo del nodo que causa la rotación.

b) (3 puntos) Explique el concepto de "peor árbol AVL de altura *h*" y construya uno de los peores AVL de altura 5 cuyos nodos tengan como clave enteros.