Primer Parcial de Programación 3 (28/09/2012)

Instituto de Computación, Facultad de Ingeniería

- Este parcial dura **4** horas y contiene 3 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 &, las sentencias *new*, *delete*, el uso de *cout* y *cin* y el tipo `bool` predefinido en `C++`.
- NO se puede utilizar ningún tipo de material de consulta. Salvo que se indique lo contrario puede 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 (12 puntos)

Un negocio de comida rápida ubicado en cierta ciudad, dispone de un sistema que le permite representar sus centros de envíos en un mapa que divide a la ciudad en zonas.

Para representar el mapa, se divide la ciudad en *n* zonas, estas se identifican por un número entre cero y $n - 1$. En cada zona puede haber o no un único centro de envío; el centro de envío se identifica con el identificador de la zona a la que pertenece.

Dados dos centros de envíos se desea saber cuales son las zonas equidistantes a ambos centros. En este contexto se entiende por distancia entre dos zonas al largo (en términos de saltos de adyacencia) del camino de menor largo entre ellas. Se asume que la distancia de una zona a si misma es cero, la distancia entre dos zonas adyacentes es uno, y no existe distancia para todo par de zonas si no hay camino entre ellas.

Dado el mapa de zonas, se requiere describir como se puede solucionar el problema e implementar un algoritmo que obtenga el conjunto de zonas equidistantes entre dos centros de envíos dados (que **se asume** pertenecen a distintas zonas).

Se pide:

Parte 1) (2 puntos) Modele la realidad dada en términos de grafos (detallando las características del grafo) e indique en este modelo en qué consiste el problema a resolver. Describa brevemente como se podría resolver el problema.
Parte 2) (10 puntos) Implemente su solución en C* de acuerdo a:

```
ListaEnteros ZonasEquidistan(Grafo g,int n,int idZona1,int idZona2)
```

Asumir: $0 \leq$ `idZona1, idZona2 <` **`n;`** `idZona1` es distinto a `idZona2; idZona1, idZona2` corresponden a centros de envío
Notas:
- Puede utilizar los TAD: Grafo, Lista y Cola de Enteros vistos en el curso.
- Toda otra operación debe ser implementada.

Ejercicio 2 (11 puntos)

Parte 1) (6 puntos)

1. Considere un algoritmo que tiene como entrada un arreglo **a** de enteros (los enteros son infinitos) de tamaño **n** sin elementos repetidos. Cualquier decisión que toma el algoritmo sólo depende de las *comparaciones* de *elementos* del arreglo y su operación básica es la *comparación* de *elementos* del arreglo. En el algoritmo las únicas modificaciones que se realizan al arreglo son intercambios de elementos del mismo.

   (a) Explicar por qué se pueden utilizar permutaciones (de la entrada) como representantes canónicos para calcular el costo del algoritmo. Muestre en un ejemplo lo explicado anteriormente.

   (b) ¿Qué utilidad tiene estudiar sólo las permutaciones en vez de todos los posibles arreglos a los efectos de estudiar el costo en un caso medio?

   (c) Definir inversión de una permutación. De un ejemplo de una permutación π de n = 8, con exactamente 3 inversiones.

2. De un ejemplo de una función $g : N \rightarrow R^*$ tal que $g \in O(1)$ y $g \notin \Theta(1)$.
Verifique, utilizando las definiciones, que efectivamente el ejemplo cumple dichas condiciones.

3. Enuncie la regla del límite. Para el caso de que el límite sea un real positivo pruebe, utilizando un contraejemplo, que el recíproco no necesariamente se cumple. Verifique, utilizando las definiciones, que efectivamente el contraejemplo cumple dichas condiciones.

Parte 2) (5 puntos)

Dado el siguiente algoritmo que recibe como parámetro un arreglo **a** de tamaño **n** de enteros distintos, interesa calcular el costo en el mejor ($T_B(n)$) y peor ($T_W(n)$) caso, considerando simultáneamente las dos siguientes operaciones elementales:

- La comparación entre elementos de **a** con costo *c1*.
- La asignación en el arreglo **a** con costo *c2*.

Se pide:

1. Indique en forma genérica las entradas de tamaño ***n*** que maximizan el costo y luego calcule el costo del peor caso.
2. Indique en forma genérica las entradas de tamaño ***n*** que minimizan el costo y luego calcule el costo del mejor caso.

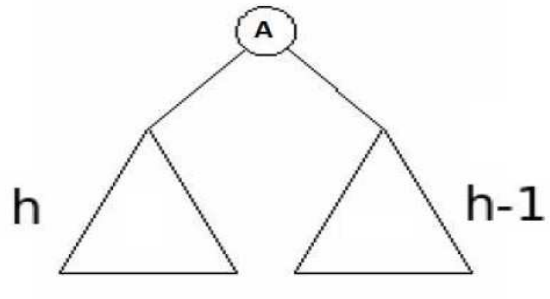
```
void algoritmo(int n, int* a) {

   for (int i=0; i<n-2; i=i+3) {
      swap(a, i, i+2);
   }
   for (int i = 0; i < n ; i++) {
      for (int j = i + 1; j <= n - 1; j++) {
         if (a[i] > a[j])
            swap(a, i, j);
      }
   }
}

void swap (int* a, int i, int j) {
   int aux = a[i];
   a[i] = a[j];
   a[j] = aux;
}
```

Ejercicio 3 (5 puntos)

Dado un AVL de altura h+1, con h ≥ 0, discuta todos los casos en los cuales luego de insertar un elemento en el subárbol izquierdo sea necesario realizar una rotación LL teniendo como vértice raíz de la rotación el vértice **A**. En todos los casos indique cómo están los factores de balanceo antes y luego de la rotación/rebalanceo.



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

Ejercicio 4 (12 puntos)

Considere un Diccionario acotado de enteros con las siguientes operaciones:

```
Crear     : entero → Diccionario
Insertar  : Diccionario × entero → Diccionario
Pertenece : Diccionario × entero → bool
Borrar    : Diccionario × entero → Diccionario
```

Sea ***N*** la cantidad máxima de elementos a insertar en el diccionario.
Considere que el diccionario está implementado con una tabla de hash (vector de tamaño *N*) con resolución de colisiones mediante listas encadenadas.

a. (2 puntos) Escriba, para el caso promedio y peor caso, el mejor orden del tiempo de ejecución que se puede lograr en cada operación.
b. (5 puntos) Proponga una extensión con una operación que retorne una lista con todas las claves que fueron ingresadas al diccionario, sin importar el orden relativo entre ellas. Esta operación debe poder ser implementada en O(1) peor caso. Se debe hacer un dibujo con los comentarios pertinentes y dar una explicación de cómo implementaría esta operación y cada una de las operaciones de la parte (a). En ningún caso, la extensión puede afectar los órdenes de las operaciones de la parte (a).
c. (5 puntos) Considere una extensión a la parte (b) con las operaciones:
   `InsertarM  : Diccionario × SetEnteros → Diccionario`
   `BorrarM    : Diccionario × SetEnteros → Diccionario`
   Ambas operaciones invocan *M* veces a `Insertar` y `Borrar` respectivamente, con cada uno de los elementos de un conjunto dado (`SetEnteros`). Los conjuntos tienen $M < N$ elementos.
   La extensión debe, además, contar con una operación `DevolverListaOrdenada` que retorne una lista con todas las claves que fueron ingresadas al diccionario, ordenadas de menor a mayor. Partiendo de un diccionario vacío se ejecuta `Devolver()` se compone de `Crear`,`InsertarM`, `DevolverListaOrdenada`. Se requiere resolver este grupo de operaciones en el mejor orden posible en el peor caso. Se debe hacer un dibujo con los comentarios pertinentes y para cada operación de las partes a, b y c se debe dar una explicación de cómo la implementaría, indicando su orden en el peor caso (con una breve justificación).