

## Algoritmos y Estructuras de Datos.

### 1er Parcial. Tema: 1A. [21 de abril de 2005]

**[Ej. 1] [Clases (30 puntos)]** Escribir la implementación en C++ del TAD LISTA (clase `list`) implementado por punteros ó cursores ó arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `next()/iterator::operator++(int)`, `list()`, `clear()`. Observaciones:

- En caso de optar por escribir la interfase “básica”, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase “avanzada”, debe declarar e implementar completamente las partes privadas de la clase `list` e `iterator`.

**[Ej. 2] [Programación (total = 50 puntos)]**

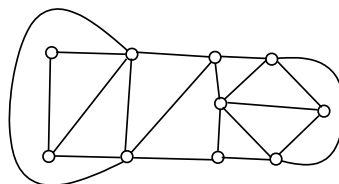
a) **[es-permutacion (25 puntos)]**

Una correspondencia es una “*permutacion*” si el conjunto de los elementos del dominio (las claves) es igual al del contradominio (los valores). De esta manera se puede interpretar a la correspondencia como una operación de permutación de las claves (de ahí su nombre). Por ejemplo  $M = \{1 \rightarrow 5, 3 \rightarrow 7, 9 \rightarrow 9, 7 \rightarrow 1, 5 \rightarrow 3\}$  es una permutación ya que tanto las claves como los valores son el conjunto  $\{1, 3, 5, 7, 9\}$ . Para determinar si una correspondencia es una permutación (y sin usar contenedores de tipo “*conjunto*”, que serán vistos más adelante), se puede utilizar el siguiente procedimiento. Construir una correspondencia  $M2$  que mapea los valores del contradominio a las claves, es decir, para cada par de asignación  $(k \rightarrow v)$  tal que  $M[k] = v$ , entonces se debe asignar el par  $(v \rightarrow k)$  en  $M2$ . Notar que si la correspondencia no es biunívoca, es decir si varias claves son asignadas a un mismo valor, entonces a  $v$  se le asignará alguna de esas claves. Cual de ellas es asignada dependerá del orden en que se recorren las asignaciones de  $M$ . Notar que si  $M$  es biunívoca, entonces  $M2$  es la inversa de  $M$ . Entonces, si la composición de  $M$  con  $M2$  es la identidad es decir para cada  $k$  en las claves de  $M$ , si  $M2[M[k]] = k$ , entonces  $M$  es una permutación. Consigna: escribir una función predicado `bool es_permutacion(map<int,int> &M)` que retorna `true` si  $M$  es una permutación y `false` si no lo es.

b) **[list-sort (25 puntos)]**

Escribir una función `void sort(list<int> &L);`, que ordena los elementos de  $L$  de menor a mayor. Para ello utilizar el siguiente algoritmo simple, utilizando una lista auxiliar  $L2$ : ir tomando el menor elemento de  $L$ , eliminarlo de  $L$  e insertarlo al final de  $L2$  hasta que  $L$  este vacía. Cual es el tiempo de ejecución en función de la cantidad de elementos  $n$  de  $L$ ?

**[Ej. 3] [color-grafo (5 pts)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. ¿La coloración obtenida es óptima? Justifique.



**[Ej. 4] [Preguntas (total = 15 puntos, 5 puntos por pregunta)]** Responder según el sistema “multiple choice”, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente “descabelladas” y tienen puntajes **negativos!!**

Apellido y Nombre: \_\_\_\_\_

Carrera: \_\_\_\_\_ DNI: \_\_\_\_\_

[Llenar con letra mayúscula de imprenta GRANDE]

a) Considere la función:

```
bool is_mapped(map<int,int> &M,int key,int val) {  
    return /* Esta key -> val en M? ... */; }
```

que debe retornar `true` si el par de asignación `(key,val)` está en la correspondencia `M`. ¿Cuál es la expresión correcta que refina el pseudocódigo?

- ☐ ... `M[key]==val`  
☐ ... `(M.find(key)!=M.end()) && M[key]==val`  
☐ ... `M[key]==val && (M.find(key)!=M.end())`  
☐ ... `M.find(key)->second==val`

b) El tiempo de ejecución de la función `find(key)` para correspondencias implementadas por vectores ordenados es ... ( $n$  es el número de asignaciones en la correspondencia)

- ☐ ...  $O(1)$   
☐ ...  $O(\log n)$   
☐ ...  $O(n^2)$   
☐ ...  $O(n)$

c) Dadas las funciones

- $T_1(n) = 3n^3 + 2n! + \log n$ ,
- $T_2(n) = 3 \cdot 2^3 + n^2 + n^{1.5}$ ,
- $T_3(n) = 5! + 6 \cdot 2^n + 20 \cdot n^2$  y
- $T_4(n) = 2^{10} + 20n + \log_2 40$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$