Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2009-09-17]

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)]

a) **[lista (20pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()/iterator::operator++(int)`, `list()`, `begin()`, `end()`.

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] [Programación (total = 50pt)]

a) **[intersect-map (30pt)]** Implemente una función `void intersect_map(const map< string, list<int> > &A, const map< string, list<int> > &B, map< string, list<int> > &C)` que a partir de los diccionarios `A` y `B` construya un diccionario `C` de acuerdo a las siguientes reglas:

- Si una clave `key` esta contenida en A o B pero **no en ambos**, entonces C **no debe contener** dicha clave.
- Si una clave `key` esta contenida en A y B a la vez, entonces C debe contener dicha clave y su valor asociado debe ser una lista que contenga todos los **elementos comunes y sin repetición** de las listas asociadas a `key` en A y B (es decir, la intersección como conjunto, $C[key] \leftarrow A[key] \bigcap B[key]$).

Por ejemplo, dados:

```
A = { 'XX' : [3,3,1,2,2,7],     B = { 'YY' : [3,3,4,5,8,1],
      'YY' : [7,1,5,5,4,1] }           'ZZ' : [1,1,9]       }
```

se debe obtener:

```
C = { 'YY' : [1,5,4]}
```

Sugerencia: implementar una función auxiliar `bool contains(list<int>&L, int x)` que devuelve `true` si la lista `L` contiene el valor `x`; y `false` en caso contrario. Utilize esta funciń como ayuda para contruir la lista de elementos comunes y sin repetición $C[key] \leftarrow A[key] \bigcap B[key]$

b) **[are-inverses (10pt)]** Dos correspondencias `M1` y `M2` son **inversas** una de la otra si tienen el mismo número de asignaciones y para cada par de asignación `x->y` en `M1` existe el par `y->x` en `M2`.
Consigna: Escribir una función **predicado**
`bool areinverse(map<int,int> &M1,map<int,int> &M2);` que determina si las correspondencias `M1`, `M2` son una la inversa de la otra o no.

c) **[map2list (10pt)]** Escribir las funciones `map2list()` y `list2map()` de acuerdo a las siguientes especificaciones.

- `void map2list(map<int,int> &M,list<int> &keys,list<int> &vals);` dado un map `M` retorna las listas de claves y valores. Es decir, si `M={(1->2,3->5,8->20)}`, entonces debe retornar `keys=(1,3,8)`, `vals=(2,5,20)`.
  Nota: `keys` debe quedar ordenada de menor a mayor. Asumir que la implementación de `map<>` es STL, es decir, tal que al recorrer las asignaciones las claves resultan ordenadas.
- `void list2map(list<int> &keys,list<int> &vals,map<int,int> &M);` dadas las listas de claves `(k1,k2,k3...)` y valores `(v1,v2,v3...)` retorna el map `M` con las asignaciones correspondientes `{(k1,v1),(k2,v2),(k3,v3),...}`. (*Nota:* Si hay **claves repetidas**, sólo debe quedar la asignación correspondiente a la **última** clave en la lista. Si hay menos valores que claves, utilizar cero como valor. Si hay más valores que claves, ignorarlos).

- *Pregunta 1:* Si ejecutamos el siguiente código

```
list2map(keys,vals,M);
map2list(M,keys2,vals2);
```

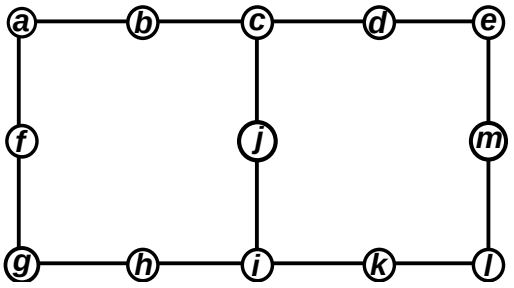
¿Cuáles son las condiciones sobre `keys` y `vals` para que resulte ser `keys2==keys`, `vals2==vals`? Justifique.
Ayuda: Tener en cuenta qué ocurre cuando 1) hay claves repetidas, 2) si `keys` y `vals` están ordenadas o no, y 3) las longitudes de ambas son iguales o no.

- *Pregunta 1:* Si ejecutamos el siguiente código

```
map2list(M,keys,vals);
list2map(keys,vals,M2);
```

¿Resulta ser `M2==M`? ¿Depende la respuesta de la implementación de `map<>`? Justifique.

[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. (*Nota:* debe recorrer los vértices **en el orden que se indica**, *a*, *b*, *c*, ...) ¿Puede indicar si la coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15pt, 3pt por pregunta)]**

a) Ordenar las siguientes funciones por tiempo de ejecución:

$$\begin{aligned} T_1 &= \sqrt{n} + 2\log_4 n + 2\,n^3 + 2\,n^5 \\ T_2 &= n^3 + 3\cdot 4^n + 2^{20} \\ T_3 &= 4^3 + 10 + 2\log_{10} n + 4\log_2 n \\ T_4 &= 2\cdot 4^n + 5\,n! + 3n^2 \\ T_5 &= \log_{16} n + \sqrt{5}\cdot n + 5\cdot 100^n \end{aligned} \tag{1}$$

b) ¿Porqué se dice que la pila es una estructura *FIFO ("First in, First Out")*? ¿Porqué se dice que la cola es una estructura *LIFO ("Last In, First Out")*?

c) Si tenemos la correspondencia `M={(5->6),(9->12)}` y ejecutamos el código `int x= M[9]`. ¿Que ocurre? ¿Que valores toman `x` y `M`? ¿Y si hacemos `x = M[7]`?

d) Cual es la complejidad algorítmica (mejor/promedio/peor) de las función `lower_bound()` para correspondencias implementadas por

1) listas ordenadas,
2) vectores ordenados.

e) Considerando la implementación de pilas con **listas simplemente enlazadas**. ¿Cuál es la diferencia entre elegir como tope de la pila el comienzo o fin de la lista?