Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2013-11-28]

ATENCIÓN (1): Escribir cada sección **hojas independientes**, poniendo claramente el nombre en la parte superior derecha de la misma.
ATENCIÓN (2): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases.
- **50 %** en programación.
- **50 %** en operativos.
- **60 %** sobre las preguntas de teoría.

ATENCIÓN (3): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [CLAS1 (W=20)]

a) 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)`** (postfijo), **`list()`**, **`begin()`**, **`end()`**.
b) 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.
c) **AOO:** declarar las clases **`tree`**, **`cell`**, **`iterator`**, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
`tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

[Ej. 2] [CLAS2 (W=20)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **AB:** declarar las clases **`btree`**, **`cell`**, **`iterator`**, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
`btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`
b) Implementar **`bool bst_erase(btree<T> t,T x,bool (*less)(T,T))`** que elimina el elemento **`x`** del ABB T, retornando true si la eliminación fue exitosa, y false caso contrario.
c) Implementar una función **`pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)`** que inserta el elemento **`x`** en la tabla de dispersión abierta **`table`** utilizando la función de dispersión **`hashfunc`** y la relación de equivalencia **`equal`** retornando un par de **`int`** (que indica la cubeta) y **`bool`** que indica si la inserción fue exitosa.
d) Implementar una función **`void vecbit_diffsym(vector<bool>&, A, vector<bool>& B, vector<bool>& C);`** que devuelve la **diferencia simétrica** definida como $C = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$ con **`A,B,C`** vectores de bits que representan conjuntos de un rango contiguo de enteros **`[0, N)`**, donde **`N`** es el tamaño de los vectores **`A,B,C`** (asumir el mismo tamaño para los tres, es decir, haciendo **`int N=A.size();`** es suficiente).

[Ej. 3] [PROG1 (W=40)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **[are-inverses (W=20)]** 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.

b) **[ord-nodo (W=20)]**. Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

[Ej. 4] [PROG2 (W=40)]

Insistimos: **deben usar la interfaz avanzada (STL)**.

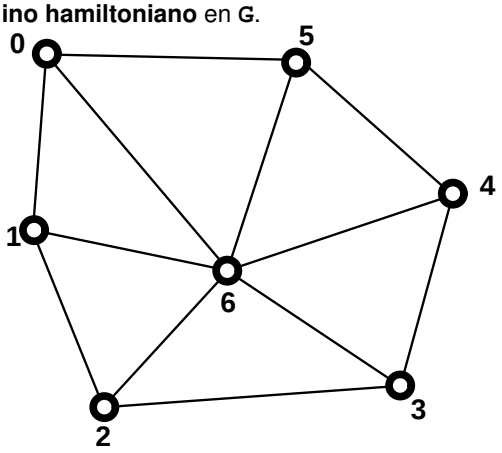
a) **[cumsum (W=20)]** *Consigna:* Escribir una función `void cumsum(btree<int> &T);` (por **suma acumulada**) que dado un **árbol binario (AB)** `T` modifica el valor de los nodos interiores, de manera que resulta ser la suma de los valores de sus hijos **más** el valor que había en el nodo antes de llamar a `cumsum()`. Los valores de las hojas no son modificados, y los valores de los nodos interiores resultan ser la suma de todos los valores del subárbol del nodo **antes** de llamar a `cumsum`.

Ejemplo: Si `T=(1 5 (2 3 7 (11 4 2)))` entonces después de llamar `cumsum(T)` debe quedar `T=(35 5 (29 3 7 (17 4 2)))`.

Ayuda:

- Si el nodo es Λ no debe hacer nada.
- Si no es Λ debe primero aplicar recursivamente la función a todos los hijos. Luego poner en el nodo `n` la suma de su valor más la de todos sus hijos.

b) **[is-hamilt (W=20)]** Dado un grafo `map<int, set<int> >G` y una lista de vértices `list<int> L` determinar si `L` es un **camino hamiltoniano** en `G`.



Consigna: Escribir una función
`bool ishamilt(map<int, set<int> > &G,list<int> &L);` que realiza la tarea solicitada.

Ayuda: Mantener un `set<int> visited` con todos los vértices visitados. Para dos enteros `x`, `y` sucesivos en la lista `L`, verificar si son vértices de `G` y si son adyacentes. Verificar que el nuevo vértice `y` no fuera visitado previamente e insertarlo en `visited`. Al final, chequear que todos los vértices fueron visitados.

Nota: Recordamos que un camino Hamiltoniano en un grafo es un camino en el mismo que pasa por todos los nodos sin repetir ninguno. Por ejemplo en el grafo de arriba el camino `{0,1,2,3,4,5,6}` es Hamiltoniano.

[Ej. 5] [OPER1 (W=20)]

a) **[rec-arbol (W=10)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE={Z,X,R,T,Q,P,A,W}`
- `ORD_POST={X,T,W,A,P,Q,R,Z}`

b) **[particionar (W=10)]**. Considerando el árbol `(Z (T Q R) (A (B (D E (F Y ))))))` decir cuál son los nodos `descendientes(F)`, `antecesores(F)`, `izquierda(F)` y `derecha(F)`.
(**Nota:** se refiere a antecesores y descendientes **propios**).

[Ej. 6] [OPER2 (W=20)]

a) **[huffman (W=5)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`NALBANDIAN`**.
$P(A) = 0.10, P(N) = 0.10, P(L) = 0.10, P(I) = 0.30, P(D) = 0.05, P(B) = 0.3, (C) = 0.05$
Calcular la **longitud promedio** del código obtenido.

b) **[abb (W=5)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 3, 12, 1\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 13, 5 y 2 en ese orden.

c) **[hash-dict (W=5)]** Insertar los números $\{1, 14, 24, 7, 6, 34, 16, 3\}$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x$.

d) **[heap-sort (W=5)]** Dados los enteros $\{11, 14, 4, 3, 9, 6, 2\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 7] [PREG1 (W=20)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$
\begin{aligned}
T_1 &= 5n^4 + 2 \cdot 3^n + 25\, n!. \\
T_2 &= \sqrt{2} \cdot n + 5 \cdot 2^n + \log_{10} n, \\
T_3 &= 2^{20} + n^2 + 3 \cdot 5^n, \\
T_4 &= \sqrt{n} + 5 \log_8 n + 2\, n^2 + 3\, n^5, \\
T_5 &= 2 \log_2 n + 5^3 + 3 \log_4 n + 200,
\end{aligned}
$$

b) Si la correspondencia **`M={(4->3),(2->8)}`** y ejecutamos el código **`int x= M[4]`**. ¿Que valor toma **`x`**? ¿Cómo queda **`M`**? Si ahora hacemos **`x = M[5]`**, ¿cómo quedan **`x`** y **`M`**?

c) Defina que es un **camino** en un árbol. Dado el AOO **`(b c (d f (g h)) e)`**, cuáles de los siguientes son caminos?
1) **`(f,d,b)`**,
2) **`(b,d,g,h)`**,
3) **`(d,g,h)`**,
4) **`(f,d,g,h)`**.

d) Discuta si los algoritmos de **búsqueda exhaustiva** y **heurístico** para la coloración de grafos dan la solución **óptima** al problema. Señale la complejidad algorítmica de los mismos. ¿Cuál de los dos elegiría para un grafo con 100 vértices y 500 aristas? Justifique.

e) Defina en forma **recursiva** el listado en **orden previo** y el listado en **orden posterior** de un arbol ordenado orientado con raiz **`n`** y sub-árboles hijos **`h_1,h_2,...,h_n`**.

[Ej. 8] [PREG2 (W=20)]

a) ¿Es posible insertar en una posición **no-dereferenciable** (Λ) en un AB? ¿Y en una posición dereferenciable?

b) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.

c) Defina la propiedad de **transitividad** para las relaciones de orden. Si $f(x)$ es una función sobre los reales y defino $a <_f b$ sii $f(a) < f(b)$, ¿es $<_f$ transitiva?

d) Discuta el valor de retorno de **`insert(x)`** para conjuntos.

e) ¿Cuál es el costo de **inserción** en tablas de dispersión cerradas?

`mstorti@galileo/aed-2.0.5-17-gcd7e7ca/Mon Dec 2 09:05:11 2013 -0300`