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.
Examen Final. [08-02-2018]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 70 % de la sección PREG. y 50 % en las restantes secciones.

[Ej. 1] [PREG (min 70 %)]

a) Explique qué quiere decir la propiedad de "Transitividad" de **`O()`**
b) ¿Cuál es la "Regla del producto" para **`O()`**?
c) Discuta las diferencias en la implementación y las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?
d) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **`map<T>`** implementada con listas ordenadas y vectores ordenados en el caso promedio? Métodos: **`find(key)`**, **`M[key]`**, **`erase(key)`**, **`erase(p)`**, **`begin()`**, **`end()`**, **`clear()`**.
e) Indique los tiempos de ejecución de los algoritmos de ordenamiento rápidos para los tres casos (peor, mejor y promedio). ¿Cuando es conveniente utilizar uno u otro?
f) Explique el concepto de "Estabilidad" de un algoritmo de ordenamiento. Mencione dos ejemplos: un algoritmo estable y otro no estable.
g) Explique que es la condición de prefijos. ¿Cómo se representa ésta condición en términos del árbol de la codificación?
h) ¿Es posible "insertar" en una posición "dereferenciable" (Λ) en un AB? ¿Y en un AOO? Discuta y de ejemplos.
i) ¿Cómo se define la tasa de ocupación de una tabla de dispersión cerrada? ¿Qué implicancias tiene en el funcionamiento de la estructura de datos el hecho de que dicha tasa sea cercana a cero? ¿y cercana a uno?.
j) Mencione las dos condiciones que debe cumplir un montículo y describa cual es la utilidad en términos algorítmicos y/o almacenamiento de datos de dichas estructuras.

[Ej. 2] [CLASES (min 50 %)]

a) **[dlist]** Implemente los métodos **`insert`** y **`erase`** para una lista doblemente enlazada:

```
class cell {
    cell *next, *prev;
    elem_t elem;
    cell() : next(NULL) {}
    friend class list;
}
typedef cell *iterator_t;
class list {
    cell *first;
    public:
    ...
    iterator_t insert(iterator_t p, elem_t e);
    iterator_t erase(iterator_t p);
    ...
};
```

- ¿Es necesario utilizar un iterador atrasado en este tipo de lista? ¿Por qué?

b) **[aoo]** En una implementación de Árbol Ordenado Orientado AOO en donde cada nodo del árbol es representado por una estructura como la siguiente:

```
template<class T> struct cell{
    cell * left_child, right;
    T elem;
}
```

- ¿Por qué es insuficiente utilizar iteradores que sean **`typedef cell* iterator`**? Presente un ejemplo que lo demuestre.
- Proponga una definición de iterador que solucione el/los problemas mencionados en el punto anterior.

c) **[ab]** Defina los structs/clases **`iterator`** y **`cell`** para un Árbol Binario, e implemente el método **`iterator btree<T>::find(T x)`**.

[Ej. 3] [PROG (min 50 %)]

a) **[remp-seq]** Dada una lista de enteros **`L`** y dos listas **`SEQ`** y **`REEMP`**, posiblemente de distintas longitudes, escribir una función **`void reemplaza(list<int> &L, list<int>& SEQ, list<int> &REEMP)`**, que busca todas las secuencias de **`SEQ`** en **`L`** y las reemplaza por **`REEMP`**. Por ejemplo, si **`L=1,2,3,4,5,1,2,3,4,5,1,2,3,4,5)`**, **`SEQ=(4,5,1)`** y **`REEMP=(9,7,3)`**, entonces después de llamar a **`reemplaza(L,SEQ,REEMP)`**, debe quedar **`L=(1,2,3,9,7,3,2,3,9,7,3,2,3,4,5)`**. Para implementar este algoritmo primero buscar desde el principio la secuencia **`SEQ`**, al encontrarla, reemplazar por **`REEMP`**, luego seguir buscando a partir del siguiente elemento al último de **`REEMP`**.

b) **[hay-camino]** Escriba una función **`bool hay_camino(map<int,list<int>> &G,int i,int j)`** que dado un Grafo dirigido **`G`**, y dos nodos del mismo, retorne verdadero si y solo si existe un camino dentro del grafo que vaya desde el primer al segundo nodo.

[Ej. 4] [OPER (min 50 %)]

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

- **`ORD-PRE`**=$\{P, F, H, S, B, N, Q, G, K, T, E, D, W, O\}$,
- **`ORD-POST`**=$\{Q, G, N, B, S, H, D, W, E, T, O, K, F, P\}$,

b) **[huffman]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`EXAMEN`**
$P(X) = 0.10, P(M) = 0.10, P(U) = 0.10, P(E) = 0.30, P(I) = 0.05, P(A) = 0.3, (N) = 0.05$
Calcular la longitud promedio del código obtenido.

c) **[hf-decode]** Utilizando el código del siguiente árbol binario **`(* (* (* U V) (* (* W X) Y)) Z)`**, desencodar el mensaje `010101111011`

d) **[abb]** Dados los enteros $\{15, 9, 22, 4, 5, 12, 7, 6, 3, 14\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 15, 7 y 4 en ese orden.

e) **[hash-dict]** Insertar los números $9, 2, 0, 19, 32, 14, 41, 58$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x \bmod 9$ y estrategia de redispersión lineal.

f) **[heap-sort]** Dados los enteros $\{10, 13, 3, 2, 8, 5, 1\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

mstorti@galileo/TPL-SUPREC-2015-15-ge3b3752/Thu Nov 19 12:34:07 2015 -0300