Programación II. Parcial 2. 9 de mayo 2017 - Grupos de mañana

Apellidos (en mayúsculas):
_____Nombre:_____

Ejercicio 1 (2,5 puntos en total)

a) Dadas las estructuras de datos que hemos utilizado en clase para implementar un nodo y una lista enlazada:

En lista.c:

```
struct _Nodo {
      Elemento *info;
      struct _Nodo *next;
};
typedef struct _Nodo Nodo;

struct _Lista {
      Nodo *first;
};
```

En lista.h:

```
typedef struct _Lista Lista;
```

Implementa en C, **con control de errores**, la primitiva:

```
Status lista_cuenta_apariciones(Lista *pl, const Elemento *pe);
```

que cuenta todas las apariciones en la lista del elemento apuntado por pe. El elemento puede estar varias veces repetido en la lista de entrada, que **supondremos está siempre ordenada de menor a mayor valor**. Puedes usar la primitiva del TAD elemento:

```
int elemento_comparar(const Elemento *pe1, const Elemento *pe2)
```

que es 0 si los dos elementos son iguales, -1 si el primer elemento tiene menor valor que el segundo elemento, y 1 en caso contrario.

Utiliza el hecho de que la lista está ordenada para hacer la función lo más eficiente posible. ¿Qué aspectos son más eficientes respecto a una implementación en la que no aprovechas que la lista está ordenada?

b) Suponiendo que la lista esté implementada en un array estático, y dicha lista está ordenada, ¿es posible implementar la función anterior de una manera aún más eficiente? Justifica tu respuesta. ¿Cuál sería el orden de complejidad de esta función y cuál era el de la función que implementaste en el apartado a)?

Programación II. Parcial 2. 9 de mayo 2017 - Grupos de mañana

Apellidos (en mayúsculas):
_____Nombre:_____

Ejercicio 2 (2,5 puntos en total)

(a) Proporcione el pseudocódigo de un algoritmo que determine si un árbol binario dado es una lista (es decir, el árbol es vacío, o no es vacío pero cada nodo tiene como mucho un hijo).

(b) Proporcione el código C del algoritmo anterior haciendo uso de las siguientes macros y estructuras

```
#define ROOT(t) (t)->root
#define IZQ(pn) (pn)->izq
#define DER(pn) (pn)->der


typedef struct _Nodo {
   Elemento *info;
   struct _Nodo *izq;
   struct _Nodo *der;
} Nodo;


typedef struct _ARBOL ARBOL;
struct _ARBOL {
     Nodo *root;
}
```

Programación II. Parcial 2. 9 de mayo 2017 - Grupos de mañana

Apellidos (en mayúsculas):
_____Nombre:_____

Ejercicio 3 (2,5 puntos en total)

a) Define, en 2 o 3 líneas, qué es un árbol binario de expresión (AdE). Dibuja uno que tenga más de 3 nodos.

b) ¿Qué información se puede obtener de un AdE, según el modo en que se recorra?

-

c) ¿Se puede crear un árbol de expresión algorítmicamente a partir de una expresión sufijo (=posfijo)?

☐ No → ¿Por qué? ¿Cómo tendría que ser la expresión?

☐ Sí → Pon un ejemplo en el que se muestre la evolución del algoritmo, paso a paso, sobre una expresión sufijo que contenga 5 elementos más la marca de final de expresión.

d) Construye un árbol binario de búsqueda (ABdB) para mantener ordenados y poder buscar después los siguientes elementos. Sigue para ello el algoritmo estudiado en clase y muestra la evolución del mismo paso a paso.

6 8 2 7 4 9 1

e) Directamente sobre el árbol obtenido anteriormente, marca los nodos con los que se compararía para buscar el número 5, siguiendo el algoritmo de búsqueda en ABdB (rodea esos nodos con mayor grosor, p.ej.), y numéralos según el orden en que se realizarían las comparaciones.

f) Observa la estructura del árbol obtenido (nodos, nº hijos de cada nodo, nº nodos en cada nivel, etc.). ¿Qué tipo de árbol es? _____¿Los ABdB siempre tienen esta estructura?

☐ No → Dibuja uno que no la tenga.

☐ Sí → ¿Por qué?

g) Compara la utilización de un ABdB para ordenar y buscar elementos, frente a utilizar un array en el que se vayan colocando los elementos consecutivamente, de forma ordenada, en cuanto a:

- Eficiencia del procedimiento inicial hasta tener los elementos ordenados en la estructura.

- Eficiencia en la búsqueda

- Uso de memoria

Programación II. Parcial 2. 9 de mayo 2017 - Grupos de mañana

Apellidos (en mayúsculas):
_____Nombre:_____

Ejercicio 4 (2,5 puntos en total)

World of Warcraft, comúnmente conocido como WoW, es un videojuego de rol multijugador masivo en línea.

En él los usuarios (jugadores) desde sus ordenadores conectados a internet controlan personajes "avatares" dentro de un mundo virtual explorando el entorno, completando misiones, e interactuando y combatiendo con personajes no jugadores o con otros usuarios.

Para poder jugar en red los usuarios crean o se unen a una sesión de juego a través de un servidor de aplicaciones.

Un servidor mantiene un protocolo de comunicaciones que permite obtener y sincronizar los momentos (*timestamps*) en los que se realizan las acciones de los jugadores.

Suponer que en un intervalo de tiempo dado, un servidor recibe y procesa las siguientes peticiones de acciones de jugadores, ordenadas de izquierda a derecha según su llegada al servidor:

orden de llegada	1	2	3	4	5	6
jugador-acción	elfo1-A1	humano1-A2	orco1-A4	orco1-A2	goblin1-A3	elfo2-A3
timestamp	104	102	106	101	103	105

El servidor almacena las peticiones entrantes según el orden de llegada, y las procesa atendiendo a sus timestamps.

Así, en un momento dado, de las peticiones que tiene almacenadas, el servidor procesa la de menor timestamp.

Para ello, mantiene una cola de prioridad implementada con el TAD Heap.

A partir de las peticiones de la tabla de arriba:

- ¿Cómo queda el heap tras haber recibido las 6 peticiones? Razona tu respuesta dibujando el estado* del heap tras almacenar cada una de las peticiones. (1 punto)
- Tras haber almacenado las 6 peticiones, el servidor extrae del heap y procesa 2 peticiones. ¿Qué pares jugador-acción se ejecutan? Dibuja el estado* del heap tras cada una de esas extracciones. (1 punto)

(*) Para simplificar la visualización, representa cada nodo del heap con su timestamp.

En vez de un Heap, se plantea usar una Lista Enlazada ordenada como cola de prioridad para gestionar las peticiones del servidor. Para esta tarea:
Asumiendo que el heap se almacena en un array y que la lista enlazada se almacena usando nodos independientes con la estructura Nodo estudiada en la asignatura, ¿qué diferencias en coste computacional de inserción/extracción y memoria empleada tienen cada uno de los dos TADs? (0,5 puntos)