Ejercicios del primer parcialito - Taller Abril 2018

Del primer recuperatorio primer parcialito, primer cuatrimestre de 2017

1. Implementar en C el TDA ComposiciónFunciones que emula la composición de funciones (i.e. $f(g(h(x)))$. Se debe definir la estructura del TDA, y las siguientes primitivas:

   - `composicion_t* composicion_crear()`
   - `void composicion_destruir(composicion_t*)`
   - `bool composicion_agregar_funcion(composicion_t*, double (*f)(double))`
   - `double composicion_aplicar(composicion_t*, double)`

   Considerar que primero se irán agregando las funciones como se *leen*, pero tener en cuenta el correcto orden de aplicación. Por ejemplo: para emular $f(g(x))$, se debe hacer:

   ```
   composicion_agregar_funcion(composicion, f);
   composicion_agregar_funcion(composicion, g);
   composicion_aplicar(composicion, x);
   ```

   Indicar el orden de las primitivas.

2. En un colegio de Villa General Belgrano, los alumnos deben formar de la siguiente manera: primero las niñas ordenadas por altura de menor a mayor. Luego los niños, también ordenados por altura de menor a mayor. Escribir una función en C que reciba una cola de enteros (representando las alturas de los alumnos en centímetros), y devuelva true si es posible que estén bien formados (considerando sólo las alturas) o false en caso contrario. Se puede vaciar la cola sin necesidad de dejarla como se la recibió. Por ejemplo:

   - Primero -> `[ 125, 128, 129, 124, 134, 138, 140 ]`: true
   - Primero -> `[ 125, 128, 129, 133, 134, 138, 140 ]`: true
   - Primero -> `[ 125, 120, 129, 133, 124, 138, 140 ]`: false

   Indicar el orden del algoritmo.

Del primer recuperatorio primer parcialito, segundo cuatrimestre de 2017

1. Dada una lista enlazada implementada con las siguientes estructuras:

   ```
   typedef struct nodo_lista {
       struct nodo_lista* prox;
       void* dato;
   } nodo_lista_t;
   ```

   ```
   typedef struct lista {
       nodo_lista_t* prim;
   } lista_t;
   ```

   Escribir una **primitiva** que reciba una lista y devuelva el elemento que esté a k posiciones del final (el *ante-k-último*), **recorriendo la lista una sola vez**. Considerar que k es siempre menor al largo de la lista.

   Por ejemplo, si se recibe la lista `[ 1, 5, 10, 3, 6, 8 ]`, y `k = 4`, debe devolver 10.

   Indicar el orden de complejidad de la función.

2. Dada una pila de enteros, escribir una **función** que determine si es piramidal. Una pila de enteros es *piramidal* si cada elemento es menor a su elemento inferior (en el sentido que va desde el tope de la pila hacia el otro extremo). La pila no debe ser modificada al terminar la función.

   Indicar el orden del algoritmo propuesto.

3. Implementar un algoritmo en C que reciba un arreglo de n enteros sin repetir y ordenado ascendentemente, y determine en $O(logn)$ si es mágico. Un arreglo es *mágico* si existe algún valor i (entre 0 y $n-1$) tal que `arr[i] = i`.

   Ejemplos:

   - `A = [ -3, 0, 1, 3, 7, 9 ]` es mágico porque `A[3] = 3`.
   - `B = [ 1, 2, 4, 6, 7, 9 ]` no es mágico porque `B[i] != i` para todo `i`.

   Justificar el orden del algoritmo.