Parcialito Proyecto 4 - Tema A

Resolver los ejercicios listados abajo, en una hora y media (o menos). El código resultante **no** debe tener memory leaks **ni** accesos (read, write o free) inválidos a la memoria. La entrega es a través de Jaime.

Ejercicio 1

Modificar el TAD `heap` tal que éste provea una operación nueva cuya signatura es:

```
bool array_is_heap(heap_elem_t *elems, unsigned int size)
```

Esta operación retorna un booleano que indica si el array de aristas dado (`elems`) cumple la condición de heap o no. El parámetro `size` indica cuántas aristas hay en el array (notar que por la implementación particular del heap, el primer elemento no cuenta en el `size` y debe ser `NULL`).

Recordar que la definición formal de heap es la que se vio en el teórico:

$$\text{heap}(t) \stackrel{\text{def}}{=} \forall p \in \text{pos}(t). \begin{cases} p \lhd 0 \in \text{pos}(t) \Rightarrow t.(p \lhd 0) \leq t.p \\ p \lhd 1 \in \text{pos}(t) \Rightarrow t.(p \lhd 1) \leq t.p \end{cases}$$

IMPORTANTE: el símbolo $\leq$ compara prioridad entre elementos del heap, recordar que en el heap de aristas, la mayor prioridad la tiene las arista de menor peso.

Las PRE y POST de este método nuevo son las siguientes:

pre si el tamaño dado `size` es mayor que cero, el array `elems` no es nulo, y es un array de elementos del mismo tipo de los que maneja el heap (en este caso, aristas), y tiene exactamente `size` elementos no nulos.

post el resultado es `true` si para cada uno de los elementos del array, cada uno de sus hijos izquierdo y derecho (si están definidos), tienen menos prioridad que ese elemento (es decir, cada arista-hijo tiene más peso que el peso de la arista-padre).

Ejemplo

Supongamos el siguiente array, en donde cada elemento representa el peso de una arista (en el parcialito los elementos son aristas, no enteros):

`[NULL, 8, 10, 25, 23, 50, 27, 105, 93, 28, 74]`

Las llamada a `array_is_heap(array, 10)` retorna `true`. En cambio, si el array fuera:

`[NULL, 11, 10, 25, 23, 50, 27, 105, 93, 28, 74]`

o

`[NULL, 8, 10, 25, 23, 74, 27, 105, 93, 28, 50]` `array_is_heap(array, 10)` debe retornar `false`.

Ejercicio 2

Crear un archivo `parcialito.c` que provea una función `main` que haga lo siguiente:

1. Imprimir por pantalla si un array vacío cumple la condición de heap o no (**ver abajo** la función auxiliar de impresión para hacer esto).
2. Crear un array de aristas (de `heap_elem_t`) con el ejemplos de la sección anterior (elegir números al azar para los labels de los vértices), e imprimir por pantalla para cada uno de los tres casos, si el array es heap o no.

3. Crear otro array de `heap_elem_t` con las siguientes aristas, en este orden, e imprimir por pantalla el resultado de chequear si este array cumple la condición de heap o no:

   - (siempre el elemento 0 del array debería ser `NULL`)
   - `left:  5  right:  9  weight:     8`
   - `left: 15  right:  2  weight:  1010`
   - `left: 10  right: 15  weight:    29`
   - `left:  5  right:  3  weight:     1`
   - `left: 13  right: 11  weight: 65536`
   - `left:  1  right:  5  weight:   453`

4. Modificar el array del punto anterior (usando las mismas aristas), de manera tal que se garantice que el array es heap. Imprimir por pantalla el resultado de chequear si este nuevo array cumple la condición de heap o no.

IMPORTANTE Para mostrar el resultado de cada una de las llamadas arriba listadas, usar la siguiente función auxiliar:

```
void print_result(heap_elem_t * array, unsigned int size) {
    unsigned int i = 0;
    bool is_heap = array_is_heap(array, size);

    printf("El array [ ");
    for (i = 1; i <= size; i++) {
        printf("%u ", edge_weight(array[i]));
    }
    if (is_heap) {
        printf("] es heap.\n");
    } else {
        printf("] no es heap.\n");
    }
}
```