Examen final 2021-03-22

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Se dispone de los archivos `ej1.py`, `ej2.py`, `ej3.py`, `ej4.py` y `ej5.c` correspondientes a los 5 ejercicios del examen.

Cada uno tiene un lugar para escribir la implementación del ejercicio, y una función de pruebas para verificar que la solución es correcta.

El examen se aprueba con al menos 3 ejercicios correctamente resueltos. Un ejercicio se considera correctamente resuelto si:

- El programa `ej<n>` no tiene errores de sintaxis y puede ser ejecutado
- La implementación cumple con lo pedido en el enunciado

En algunos ejercicios se incluye un ejemplo de uno o dos casos de prueba y queda a cargo del alumno agregar más casos de prueba, para los que se provee sugerencias. En otros ejercicios se provee únicamente sugerencias. La implementación de las pruebas adicionales es **opcional**, pero se recomienda hacerlo ya que permite asegurar que la resolución del ejercicio es correcta.

Ejercicios en lenguaje Python

Al ejecutar cada uno de los ejercicios (`python3 ej<n>.py`), se ejecutan todas las pruebas presentes en la función `pruebas`.

Si alguna de las verificaciones falla, se imprime un mensaje de error y el programa termina su ejecución. Por ejemplo:

```
$ python3 ej1.py
Traceback (most recent call last):
File "ej1.py", line 148, in pruebas
    assert p != None
AssertionError
```

Cuando todas las pruebas pasan correctamente, se imprime `OK`:

```
$ python3 ej1.py
ej1.py: OK
```

Pruebas

Se recomienda usar la instrucción `assert` de la biblioteca estándar para verificar condiciones en las pruebas. Ejemplo de uso:

```python
# función a probar
def sumar(a, b):
    return a + b

# pruebas
```

```python
def pruebas():
    assert sumar(0, 0) == 0
    assert sumar(2, 3) == 5
    assert sumar(2, -2) == 0

    from os import path
    print(f"{path.basename(__file__)}: OK")

pruebas()
```

Nota: A veces para depurar un error en las pruebas es útil imprimir valores; se permite el uso de `print()` para ello.

Nota: A veces para implementar las pruebas es útil utilizar números aleatorios. Se permite el uso de la biblioteca `random` para ello. En ese caso, se recomienda ejecutar `random.seed(0)` al inicio del programa para asegurar que la secuencia de números aleatorios sea siempre la misma, y así facilitar la depuración.

Ejercicios en lenguaje C

Para compilar y ejecutar el ejercicio `ej5.c`:

```
$ gcc -Wall -pedantic -std=c99 ej5.c -o ej5
$ ./ej5
ej5.c: OK
```

Pruebas

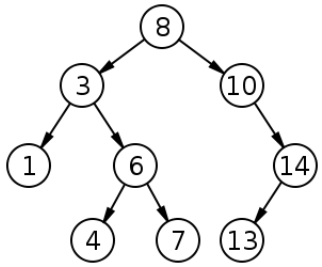
Se recomienda usar la función `assert` de la biblioteca estándar para verificar condiciones en las pruebas. Ejemplo de uso:

```c
#include <stdio.h>
#include <assert.h>

// funcion a probar
int sumar(int a, int b) {
    return a + b;
}

// pruebas
int main(void) {
    assert(sumar(0, 0) == 0);
    assert(sumar(2, 3) == 5);
    assert(sumar(2, -2) == 0);

    printf("%s: OK\n", __FILE__);
    return 0;
}
```

Ejercicios

Ejercicio 1 Un *árbol binario* es una estructura enlazada en la que cada nodo contiene referencias a otros dos nodos, llamados *hijo izquierdo* y *derecho* (pudiendo cualquiera de ellos ser una referencia nula).

Un *árbol binario de búsqueda* (ABB) es un árbol binario en el que el dato en cada nodo es mayor o igual que el dato de cualquier nodo del sub-árbol izquierdo, y menor o igual que el dato de cualquier nodo del sub-árbol derecho.



Si queremos buscar un elemento en el árbol (por ejemplo el número 6 en el árbol de arriba), no hace falta recorrer todos los nodos, ya que al visitar cada nodo podemos comparar el elemento buscado con el del nodo y así decidir si el elemento podría estar en el subárbol derecho o izquierdo. Como $6 < 8$, no hace falta visitar los nodos del subárbol derecho. Esta operación se repite en cada nodo visitado hasta encontrar el elemento, o determinar que no existe.

Dada la clase `NodoABB` que representa un nodo del ABB, implementar el método `buscar(dato)`, que busca el `dato` en el nodo y sus hijos, devolviendo el nodo que contiene el dato, o `None` si no se encuentra.

Recomendación: pensar la función en forma recursiva. Puede ser necesario hacer una función auxiliar.

Ejercicio 2 Implementar la clase `RegistroDeDesplazamiento`, que representa un valor de `n` bits. Se considera que el bit menos significativo es el que está en el extremo derecho, y el más significativo en el extremo izquierdo. Implementar los métodos:

- `__init__(n)`: crea un registro de `n` bits. Todos los bits se inicializan en 0.
- `rshift(bit)`: desplaza a la derecha el registro, moviendo todos los bits una posición en el sentido del bit menos significativo. El bit más significativo quedará con el valor `bit`. El valor previo del bit menos significativo es devuelto.
- `lshift(bit)`: desplaza a la izquierda el registro. El bit menos significativo quedará con el valor `bit`. El valor previo del bit más significativo es devuelto.
- `__str__()`: devuelve el valor en representación binaria (el bit menos significativo a la derecha).

Ejercicio 3 Implementar las funciones:

- `matriz_guardar(ruta, matriz)`, que recibe una matriz de `n x m` números enteros y la guarda en el archivo indicado.
- `matriz_cargar(ruta)` que lee del archivo la matriz y la devuelve.

El formato del archivo queda a libre criterio.

Ejercicio 4 Sea la clase `Libro` que representa un libro con un `titulo`, un `autor` y un `año` de publicación. Escribir las funciones:

- `__eq__`: (método de `Libro`) determina si dos libros son equivalentes (es decir, si todas sus propiedades son iguales.
- `ordenar_libros(libros)`: ordena la lista de instancias de `Libro` en forma creciente según el valor del `titulo`.
- `buscar_libro(libros, titulo)`: recibe una lista de libros ordenados por título, y devuelve el libro con el título indicado, o `None` si no existe, usando búsqueda binaria.

Ejercicio 5 Implementar en C la función `void join(char destino[], char delimitador, char *cadenas[], int n)`, que a partir de un arreglo de cadenas guarda en `destino` la cadena formada por todas las cadenas separadas por el delimitador. Asumir que `destino` tiene espacio suficiente. Ejemplo:

```c
char *cadenas[] = { "2021", "04", "05" };
char s[16];
join(s, '-', cadenas, 3);
// s contiene "2021-04-05"
```