Algoritmos y Estructuras de Datos II

Laboratorio - 03/04/2025

Laboratorio 2: Divide y vencerás

- Revisión 2025: Franco Luque

Código

lab02-kickstart.tar.gz

Objetivos

1. Implementar algoritmos iterativos y recursivos que siguen la estrategia de **divide y vencerás**.
2. Hacer nuestras primeras experiencias de **Testing de Unidad (*Unit Testing*)**.
3. Analizar y comparar experimentalmente la complejidad de distintos algoritmos que solucionan un mismo problema.

Requerimientos

1. Compilar con los flags de la materia:
   ```
   $ gcc -Wall -Wextra -pedantic -std=c99 …
   ```
2. Seguir las guías de estilo
3. Prohibido usar break, continue y goto!!
4. Prohibido usar return a la mitad de una función.

Recursos

Recursos generales:

- Videos del Laboratorio en el aula virtual
- Documentación en el aula virtual
- Estilo de codificación:
  - Guía de estilo para la programación en C
  - Consejos de Estilo de Programación en C

Recursos específicos:

- Teóricos:
  - Ordenación avanzada
- Prácticos:
  - Práctico 1 - Parte 2
  - Práctico 1 - Parte 3
- Unit testing
- Documentación de código con formato Doxygen
- **Laboratorio 2: Algunas soluciones:** lab02-solved

Ejercicio 1: k-ésimo elemento más chico

Parte A: Resolver el ejercicio 5 del práctico 1 parte 2: Escribí un algoritmo que dado un arreglo `a : array[1..n]` of int y un número natural k ≤ n devuelve el elemento de a que quedaría en la celda `a[k]` si el arreglo estuviera ordenado. Está permitido realizar intercambios en el arreglo, pero no ordenarlo totalmente. La idea es explotar el hecho de que el procedimiento partition del quick sort deja al pivot en su lugar correcto.

Parte B: Implementar en C la solución al ejercicio anterior dentro de la función `k_esimo()` en el módulo **`k_esimo.c`**. Recordar que se debe adaptar a arreglos indexados desde cero (0) en lugar de uno (1).

Parte C: Para el ejemplo dado en **`main.c`**, imprimir arreglo y variables en pasos intermedios. Verificar que ocurren los mismos pasos explicados en este video de youtube.

Compilar y ejecutar con:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c k_esimo.c main.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 k_esimo.o main.o -o main
$ ./main
```

Parte D: Testing: Completar los tests de **`tests.c`** con al menos estos 10 (diez) casos de test:

- arreglo de un elemento (k=0 obligatoriamente).
- arreglo de dos elementos ordenados, con k=0.
- arreglo de dos elementos ordenados, con k=1.
- arreglo de dos elementos desordenados, con k=0.
- arreglo de dos elementos desordenados, con k=1.
- el arreglo de ejemplo dado {8, -2, 9, 0, 13} con todos los k posibles.

Compilar, ejecutar los tests y asegurarse de que todos pasen:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c k_esimo.c tests.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 k_esimo.o tests.o -o tests
$ ./tests
```

Ayudas:

- Algoritmos 2: Solución Ejercicio 1.2.3: k-ésimo elemento más chico

Ejercicio 2: Cima con búsqueda secuencial

Ejercicio 2 - práctico 1 parte 3: Un arreglo ***tiene cima*** si existe una posición k tal que el arreglo es estrictamente creciente hasta la posición k y estrictamente decreciente desde la posición k. La ***cima*** es justamente el elemento que se encuentra en la posición k.

Parte A: Resolver los ejercicios 2a y 2b del práctico 1 parte 3:

- Escribí un algoritmo que determine si un arreglo dado tiene cima.
- Escribí un algoritmo que encuentre la cima de un arreglo dado (asumiendo que efectivamente
- tiene una cima) utilizando una búsqueda secuencial, desde el comienzo del arreglo hacia el final.

Parte B: Implementar en C la solución al ejercicio anterior dentro de las funciones `tiene_cima()` y `cima()` en el módulo **`cima.c`**. Recordar que se debe adaptar a arreglos indexados desde cero (0) en lugar de uno (1).

Parte C: Para el ejemplo dado en **`main.c`**, imprimir arreglo y variables en pasos intermedios. Verificar que ocurren los mismos pasos explicados en este video de youtube.

Compilar y ejecutar con:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c cima.c main.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 cima.o main.o -o main
$ ./main
```

Parte D: Testing: Completar los tests de **`tests.c`** con al menos 10 (diez) casos de test para cada una de las funciones (`tiene_cima()` y `cima()`).

Compilar, ejecutar los tests y asegurarse de que todos pasen:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c cima.c tests.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 cima.o tests.o -o tests
$ ./tests
```

Ayudas:

- Algoritmos 2: Solución Ejercicio 1.3.2: cima de un arreglo
- `tiene_cima()` es más difícil que `cima()`.

Ejercicio 3: Cima con divide y vencerás

Parte A: Resolver el ejercicio 2c del práctico 1 parte 3: Escribí un algoritmo que resuelva el mismo problema del inciso anterior utilizando la idea de búsqueda binaria.

Parte B: Implementar en C la solución al ejercicio anterior dentro de la función **`cima_log()`** en el módulo **`cima_log.c`**. Recordar que se debe adaptar a arreglos indexados desde cero (0) en lugar de uno (1).

Parte C: Para el ejemplo dado en **`main.c`**, imprimir arreglo y variables en pasos intermedios. Verificar que ocurren los mismos pasos explicados en este video de youtube.

Parte D: Testing: Completar los tests de **`tests.c`** con al menos 10 (diez) casos de test para la función **`cima_log()`**. Compilar, ejecutar los tests y asegurarse de que todos pasen.

Ayudas:

- Algoritmos 2: Solución Ejercicio 1.3.2: cima de un arreglo

Ejercicio 4: Comparación

Parte A: Resolver el ejercicio 2d del práctico 1 parte 3: Calculá y compará el orden de complejidad de ambos algoritmos.

Parte B: Utilizar el archivo **`main.c`** provisto para ejecutar pruebas y medir tiempos de ejecución de las funciones **`cima()`** y **`cima_log()`** de los ejercicios anteriores. Para ello, copiar y pegar los módulos **`cima.*`** y **`cima_log.*`** implementados en los ejercicios anteriores.

Compilar y ejecutar con:

```
$ gcc -Wall -Wextra -pedantic -std=c99 -c cima.c cima_log.c main.c
$ gcc -Wall -Wextra -pedantic -std=c99 cima.o cima_log.o main.o -o main
$ ./main
```

Guardar el resultado en un archivo de texto. Esto se puede lograr ejecutando:

```
$ ./main > resultados.txt
```

Parte C: ¡No es de programar!: Leer y entender las pruebas realizadas en el código de **`main.c`** y la tabla de números que se imprime como resultado. Graficar los resultados usando una planilla de cálculos (Google Sheets por ejemplo). El gráfico debe tener las siguientes características:

- Eje X: tamaño del arreglo (1ra columna).
- Eje Y: tiempo en milisegundos.
- Dos líneas: una para `cima()` y otra para `cima_log()`.