Algoritmos y Estructuras de Datos II

TALLER - 30 de marzo 2021

Laboratorio 2: Ordenación

- Revisión 2021: Marco Rocchietti
- Revisión 2018: Sergio Canchi
- Original 2017: Daniel Fridlender

Ejercicio 1: Insertion Sort

Dentro de la carpeta **`ej1`** vas a encontrar los siguientes archivos

Archivo	*Descripción*
`array_helpers.h`	Contiene prototipos y descripciones de funciones auxiliares para manipular arreglos.
`array_helpers.c`	Contiene implementaciones de dichas funciones.
`sort_helpers.h`	Contiene descripciones de las funciones `goes_before()`, `swap()` y `array_is_sorted()`
`sort_helpers.o`	Contiene implementaciones ilegibles de esas funciones (código compilado para la arquitectura **`x86-64`**)
`sort.h`	Contiene descripción de la función `insertion_sort()`
`sort.c`	Contiene una implementación incompleta de `insertion_sort()`, falta implementar `insert()`
`main.c`	Contiene el programa principal que carga un *array* de números, luego lo ordena con la función `insertion_sort()` y finalmente comprueba que el arreglo sea permutación ordenada del que se cargó inicialmente.

Si usted está trabajando en una computadora con arquitectura distinta a **`x86-64`***, entonces seleccione y renombre uno de los siguientes archivos,* `sort_helpers.o_32` *o* `sort_helpers.o_macos` *según la arquitectura de su máquina.*

Parte A: Ordenación por Inserción

Vas a hacer una implementación del algoritmo de ordenación por inserción. Para esta parte es necesario que abras el archivo **`sort.c`** e implementes el "procedimiento" `insert()`. Para guiarte, no dudes en examinar el resto del archivo **`sort.c`** y la definición del algoritmo de ordenación por inserción que hemos visto en clase. El algoritmo debe ordenar con respecto a la relación `goes_before()`, provista por **`sort_helpers.h`**.

Parte B: Chequeo de Invariante

Aquí será necesario que modifiques el "procedimiento" `insertion_sort()` agregando la verificación de cumplimiento de la invariante del ciclo `for` que se vio en el teórico. Por simplicidad solo verificá la siguiente parte de la Invariante:

- el segmento inicial `a[0,i)` del arreglo está ordenado.

Para esto debes hacer uso de las funciones `assert()` y `array_is_sorted()`.

Una vez implementado los incisos *a)* y *b)*, compilá ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando:

```
/sorter ../input/example-unsorted.in
```

Si anda bien (o sea, si no reporta error) probá con otros archivos de la carpeta **`../input`** no te olvides de probar con el archivo **`../input/empty.in`**

¿Te das cuenta qué relación implementa la función `goes_before()`?

Ejercicio 2: Quick Sort I

En este ejercicio vas a hacer una implementación *top-down* del algoritmo de ordenación rápida vista en el teórico. En la carpeta **`ej2`** se encuentran los siguientes archivos:

Archivo	*Descripción*
`array_helpers.h`	Es el mismo que en el ejercicio anterior.
`array_helpers.c`	Es el mismo que en el ejercicio anterior.
`sort_helpers.h`	Contiene además la declaración y descripción de `partition()`
`sort_helpers.o`	Contiene implementaciones ilegibles de esas funciones (código compilado para la arquitectura **`x86-64`**)
`sort.h`	Contiene descripción de la función `quick_sort()`
`sort.c`	Contiene una implementación muy incompleta de `quick_sort()`, además falta implementar `quick_sort_rec()`
`main.c`	Contiene el programa principal que carga un arreglo de números, luego lo ordena con la función `quick_sort()` y finalmente comprueba que el arreglo sea una permutación ordenada del que se cargó inicialmente.

Si usted está trabajando en una computadora con arquitectura distinta a **`x86-64`***, entonces seleccione y renombre uno de los siguientes archivos,* `sort_helpers.o_32` *o* `sort_helpers.o_macos` *según la arquitectura de su máquina.*

Parte A: Implmentación de `quick_sort_rec()`

Implementá el "procedimiento" `quick_sort_rec()` en el archivo **`sort.c`**. Tené en cuenta que **no es necesario** que implementes la función `partition()` puesto que la misma ya está implementada (aunque no podés leer el código porque solo disponés de la versión compilada). Para saber cómo utilizarla, examiná su descripción en **`sort_helpers.h`**.

A modo de guía no dudes en revisar la presentación del algoritmo de ordenación rápida realizada en la clase del teórico.

Parte B: Función `main()`

Para esta parte es necesario que abras el archivo **`main.c`** y completes la función `main()` con una llamada al procedimiento `quick_sort()`. Para saber cómo utilizar este "procedimiento", examiná el archivo **`sort.h`**.

Compilación

Una vez completados las partes A y B, compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

```
./sorter ../input/example-unsorted.in
```

Ejercicio 3: Quick Sort II

En la carpeta **`ej3`** se encuentran los siguientes archivos

Archivo	*Descripción*
`sort_helpers.h`	Contiene descripciones de las funciones `goes_before()`, `swap()` y `array_is_sorted()`
`sort_helpers.o`	Contiene implementaciones ilegibles de todo lo descripto en `sort_helpers.h` (código compilado para la arquitectura **`x86-64`**)
`sort.h`	Contiene descripción de la función `quick_sort()`
`sort.c`	contiene una implementación incompleta de `quick_sort()`, falta implementar `quick_sort_rec()` y `partition()`.

Si usted está trabajando en una computadora con arquitectura distinta a **`x86-64`***, entonces seleccione y renombre uno de los siguientes archivos,* `sort_helpers.o_32` *o* `sort_helpers.o_macos` *según la arquitectura de su máquina.*

Ahora tenés que copiar los archivos **`array_helpers.h`**, **`array_helpers.c`** y **`main.c`** del *ejercicio 2*. Luego copiar el "procedimiento" `quick_sort_rec()`, también del *ejercicio 2*, en el archivo **`sort.c`** y **definir** la función `partition()`.

Para finalizar la implementación de `quick_sort()` tenés que abrir el archivo **`sort.c`**, copiar tu implementación de `quick_sort_rec()` del ejercicio anterior y, ahora sí, implementar la función `partition()` usando como guía la presentación que se dio del algoritmo de ordenación rápida en la clase del teórico. Una vez implementada la función, compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

```
./sorter ../input/example-unsorted.in
```

Ejercicio 4: Versus

Vas a realizar una comparación de todos los algoritmos de ordenación implementados en este laboratorio. En la carpeta **`ej4`** estarán los siguientes archivos:

Archivo	*Descripción*
`sort_helpers.h`	Se agregan nuevas declaraciones de funciones para manejo de contadores
`sort_helpers.o`	Contiene implementaciones ilegibles de todo lo descripto en `sort_helpers.h` (código compilado para la arquitectura **`x86-64`**)
`sort.h`	Contiene las declaraciones y descripciones de las funciones de ordenación implementadas
`sort.c`	Contiene una implementación incompleta de `insertion_sort()`, falta implementar `insert()`
`main.c`	Contiene el programa principal que carga un arreglo de números, luego lo ordena usando alguno de los algoritmos de ordenación implementados y muestra: • Tiempo de ejecución • Número de comparaciones • Intercambios realizados.

Si usted está trabajando en una computadora con arquitectura distinta a **`x86-64`***, entonces seleccione y renombre uno de los siguientes archivos,* `sort_helpers.o_32` *o* `sort_helpers.o_macos` *según la arquitectura de su máquina.*

Ahora tenés que copiar los archivos **`array_helpers.h`** y **`array_helpers.c`** que venís usando y luego:

1. Abrí el archivo **`sort.c`** y copiar el código de cada uno de los algoritmos de ordenación resueltos en los ejercicios anteriores.
2. Abrí el archivo **`main.c`** y completá la función `main()` siguiendo los pasos indicados en los comentarios.

Una vez completados *1)* y *2)*, compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

```
/sorter ../input/example-unsorted.in
```