Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre Enero-Marzo 2017

Ordenamiento en tiempo lineal

1. Descripción de la actividad

El objetivo del laboratorio es el de implementar ordenamientos en tiempo lineal. En específico se quiere que implemente los algoritmos `Counting sort` y `Radix sort`

Debe usar como código base el usado en los laboratorios de las semanas pasadas, el cual está compuesto por los siguientes archivos:

- `arrayT.py`: Módulo con la definición de arreglos a utilizar.
- `ordenamiento.py`: Módulo que debe contener la implementación de los algoritmos de ordenamientos.
- `cliente_ordenamiento.py`: Cliente que ejecuta todos los algoritmos de ordenamiento.

El archivo `ordenamiento.py` debe contener solamente las implementaciones, de los algoritmos de `Mergesort`, `Heapsort` y `Quicksort` hechas en los laboratorios pasados y las implementaciones de `Counting sort` y `Radix sort` requeridas en este laboratorio.

1.1. Actividad 1: Máximo entero a generar

Se quiere que modifique los argumentos de entrada por la línea de comando del programa `cliente_ordenamiento.py`, para que reciba el número máximo de tipo entero, que pueden tener los arreglos a ordenar. Esto es, los arreglos a ordenar deben ser llenados con números en el intervalo $[0, M]$, para algún número M. Observe que actualmente en el programa `cliente_ordenamiento.py` el máximo número a generar es 500. Debe agregar a las opciones de la entrada por línea de comandos, la opción `-m` seguido del entero que corresponde al máximo entero a generar.

Por ejemplo, si se ejecuta:

```
>./cliente_ordenamiento.py -m 66 -t 3 100 300
```

Se deben ordenan arreglos de tamaño 100 300, cuyos elementos son generados aleatoriamente en el intervalo $[0, 66]$ y se debe ejecutar tres prueba por cada algoritmo de ordenamiento.

1.2. Actividad 2: Implementación de `Counting sort`

El algoritmo de `Counting sort` que debe implementar es el que se presenta en la página 194 de Cormen et al. [1] y que fue visto en la clase de teoría. En el archivo `ordenamiento.py` debe crear un procedimiento llamado `counting_sort(A, B, k)`, que recibe los arreglo A y B de tipo `arrayT` y un entero k.

1.3. Actividad 3: Implementación de `Counting sort`

Se quiere que implemente el algoritmo de `Radix sort` que está descrito en la página 197 de Cormen et al. [1] y que fue visto en la clase de teoría. En el archivo `ordenamiento.py` debe crear un procedimiento llamado `radix_sort(A, d)`, que recibe el arreglo A de tipo `arrayT` y un entero d.

1.4. Actividad 4: Experimentos a realizar

Se quiere comparar el rendimiento de los dos algoritmos de ordenamiento lineal con respecto a los algoritmos de Mergesort, Heapsort y Quicksort. Para ello debe realizar dos pruebas. La primera prueba consiste en ejecutar los algoritmos de ordenamiento en arreglos que contienen 1000, 2000, 3000 y 4000 elementos de tipo entero, cuyo máximo elemento es el número 100. La segunda prueba consiste en ordenar arreglos que contienen la misma cantidad de elementos que la prueba anterior, pero en este caso el máximo elemento es el número 5000. Debe generar una gráfica de comparación de los algoritmos por cada una de las pruebas. También debe hacer un reporte breve en donde analice los resultados obtenidos y de sus conclusiones sobre el rendimiento de los algoritmos probados. El reporte debe estar en formato `txt`.

2. Condiciones de entrega

El trabajo es por equipos de laboratorio. Debe entregar los códigos fuentes de sus programas, las gráficas con los resultados de los algoritmos de ordenamiento y el reporte, en un archivo comprimido llamado `LabSem7-X-Y.tar.gz` donde X y Y son los números de carné de los integrantes del grupo. La entrega se realizará por medio del aula virtual **antes** de la 4:30 pm del jueves 23 de Febrero de 2017.

Referencias

[1] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to algorithms*, 3rd ed. MIT press, 2009.

Guillermo Palma / gvpalma@usb.ve / Febrero 2017