

Algoritmos de Ordenamiento Parte IV

1. Introducción

El objetivo de este laboratorio es el de agregar nuevos algoritmos de ordenamientos y extender la librería `ordenamiento.py`. En específico se quiere agregar a la librería de los algoritmos de ordenamiento *Counting sort* y *Bucket sort*.

2. Algoritmos de ordenamiento

El módulo `ordenamiento` debe contener los siguientes algoritmos de ordenamiento, implementados siguiendo el pseudo código usado en las referencias:

- `Insertion-Sort`; página 18 de [3].
- `Selection-Sort`; en las páginas 256-257 de [1].
- `Shell-Sort`; página 290 de [1].
- `Bubble-Sort`; página 40 de [3].
- `Mergesort`; página 34 de [3].
- `Mergesort-Insertion`; página 229 de [2]. Cuando el número de elementos a ordenar es menor o igual a 100, entonces se ordena el arreglo con `Insertion-Sort`.
- `Mergesort-Iterativo`; página 183 de [4].
- `Heapsort`; página 159 de [3].
- `Quicksort-iterativo`; página 179 de [4].
- `Quicksort`; página 171 de [3], con la modificación de que si el arreglo o subarreglo de entrada tiene un número de elementos a ordenar que es menor o igual a 100, entonces se ordena el mismo con `Insertion-Sort`.
- `Randomized-quicksort`; página 179 de [3], con la modificación de que si el arreglo o subarreglo de entrada tiene un número de elementos a ordenar que es menor o igual a 100, entonces se ordena el mismo con `Insertion-Sort`.
- `Counting-sort`; página 195 de [3].
- `Bucket-sort`; página 197 de [3].

Para los algoritmos `QUICKSORT` y `RANDOMIZED-QUICKSORT`, debe crear una versión modificada de `INSERTION-SORT`, llamada `INSERTION-SORT-INDEX`, que permita ordenar un arreglo o subarreglo, dados el índice inicial y final del arreglo. Es decir, la entrada del `INSERTION-SORT-INDEX`, es igual a la de `QUICKSORT` y `RANDOMIZED-QUICKSORT`.

3. Programa de pruebas

Se quiere que implemente un programa, llamado `prueba_ord.py` para probar la librería `ordenamiento.py`. El programa debe ejecutar varias pruebas sobre todos los algoritmos de ordenamiento de la librería. Las pruebas tienen diferentes tipos de secuencias que van a recibir los algoritmos de ordenamiento y se describen como sigue:

1. Secuencia de números reales en el intervalo $[0, 1)$ generados aleatoriamente.
2. Secuencia de números enteros en el intervalo $[0, N]$ generados aleatoriamente, donde N es el tamaño de la secuencia a ordenar.
3. Una secuencia de tipo 1, pero en donde todos los elementos están ordenados.
4. Una secuencia de tipo 2, pero en donde todos los elementos están ordenados.
5. Una secuencia de tipo 1, pero en donde todos los elementos están ordenados de forma inversa.
6. Una secuencia de tipo 2, pero en donde todos los elementos están ordenados de forma inversa.
7. Una secuencia que contiene ceros y unos generados aleatoriamente.

Para ordenar las secuencias de tipo 3, 4, 5 y 6, que van a ser las entradas de los algoritmos de ordenamiento, se puede usar los métodos `sort` y `sorted` de Python.

El usuario puede seleccionar la prueba a aplicar y el número de veces, o intentos, que se va a aplicar esa prueba sobre los algoritmos de ordenamiento. También el usuario **puede indicar una lista con el tamaño de la secuencias de entrada** a ordenar. El resultado del programa es el **tiempo promedio de ejecución y la desviación estándar, de cada uno de los algoritmos** sobre las secuencias generadas de un tipo específico. Es importante notar que una vez escogida un tipo de secuencia, **en cada nuevo intento se debe generar una nueva secuencia, y esta misma nueva secuencia debe ser ordenada por todos los algoritmos de ordenamientos seleccionados**. De esta forma se puede comparar el tiempo de cada uno de los algoritmos de ordenamiento, sobre la misma secuencia de entrada. Se quiere que el programa de pruebas **pueda graficar los tiempos de ejecución de los algoritmos de ordenamiento, dada la lista de tamaños de secuencias de entrada**. Para graficar los resultados de los algoritmos de ordenamiento debe hacer uso de la librería `graficar_puntos.py`. Esta librería permite tener un en solo gráfico, las curvas del tiempo de ejecución de varios algoritmos de ordenamiento. De esta manera es posible ver gráficamente el desempeño de los algoritmos de ordenamiento, y determinar cual es más rápido para un tipo de entrada específica.

El algoritmo `Counting-sort` solamente será válido ejecutarlo con secuencias de tipo 2, 4, 6 y 7. Con respecto al algoritmo `Bucket-sort`, podrá ejecutarse en secuencias de tipo 1, 3 y 5. En caso de que para `Counting-sort` y `Bucket-sort` sea indicado otro tipo de secuencia diferente a las permitidas, entonces el programa debe abortar y dar un mensaje de error al usuario.

La ejecución de `prueba_ord.py` se hace por medio de la siguiente línea de comando:

```
>./prueba_ord.py [-i #num] [-t #num] [--all|--nlgn|--On2|--cs|--bs] [-g] tamaño_1 .. tamaño_N
```

La semántica de los parámetros de entrada es la siguiente:

- **-t** Prueba a realizar. Los valores posibles están en el intervalo $[1, 7]$. Cada valor corresponde a una de las pruebas descrita anteriormente.
- **-i**: Número de pruebas o intentos a realizar de la prueba seleccionada.
- **--all**: Indica que se van a ejecutar todos algoritmos del módulo `ordenamiento`.
- **--nlgn**: Indica que se van a ejecutar los algoritmos con tiempo de ejecución $O(n \log n)$, para el peor caso, del módulo `ordenamiento`.
- **--On2**: Indica que se van a ejecutar los algoritmos con tiempo de ejecución $O(n^2)$, para el peor caso, del módulo `ordenamiento`.

- **tamaño_1 .. tamaño_N** lista con los números de elementos, o tamaño, de la secuencias de entradas que van a ser ordenadas.
- **-cs**: Indica que se van a ejecutar los algoritmos **Counting-sort** y **Quicksort**.
- **-bs**: Indica que se van a ejecutar los algoritmos **Bucket-sort** y **Quicksort**.
- **-g**: indica que van a ser graficados los resultados, de todas los diferentes tamaños de secuencias ejecutados. Esta opción solo es válida si hay dos o más tamaños de secuencia. En caso contrario, el programa aborta con mensaje de error.

Los parámetros **-all**, **-nlgn**, **-On2**, **-cs** y **-bs** son mutuamente exclusivos, es decir, solo puede haber uno de ellos en la línea de comandos. En caso de haber más de uno de ellos, el programa debe abortar con un mensaje de error al usuario. Cuando se aplique la opción **-g**, primero el programa debe mostrar todos los resultados por la salida estándar y luego, se debe mostrar la gráfica resultante. Es obligatorio chequear que debe haber al menos un valor de tamaño de la secuencia. Todos los demás argumentos de la línea de comandos son opcionales. Para manejar los argumentos de entrada del programa, debe hacer uso del módulo `argparse` o del módulo `getopt`. Si la opción

Los valores por de defecto son:

- **-t**: 1, es decir, el tipo de secuencia de entrada, es la generada por el tipo 1.
- **-i**: 3, es decir, se ejecutará tres veces sobre los algoritmos de ordenamiento la prueba seleccionada.
- **-all**: falso, es decir, no se ejecutarán todos los algoritmos de ordenamiento
- **-nlgn**: cierto, es decir, se ejecutarán los algoritmos de ordenamiento con tiempo de ejecución $O(n \log n)$ para el peor caso.
- **-On2**: falso, es decir, no se ejecutarán los algoritmos de ordenamiento con tiempo de ejecución $O(n^2)$ para el peor caso.
- **-cs**: falso, es decir, no se ejecutarán los algoritmos de ordenamiento **Counting-sort** y **Quicksort**.
- **-bs**: falso, es decir, no se ejecutarán los algoritmos de ordenamiento **Bucket-sort** y **Quicksort**.
- **-g**: falso, es decir, el programa no mostrará las gráficas del tiempo de ejecución.

Las siguientes líneas de comando son ejemplos de llamadas válidas del programa.

```
> ./prueba_ord.py 1000
```

El programa ordena una secuencia con 1000 elementos usa los valores de entrada por defecto.

```
> ./prueba_ord.py -i 6 --nlgn 1000 2000 3000
```

El programa ejecutará la prueba 1. En esta prueba se ejecutará con tres secuencias que contienen 1000, 2000 y 3000 elementos. Para cada tamaño de secuencia se generarán seis secuencias diferentes. Se ejecutarán todos los algoritmos de ordenamiento del módulo **ordenamiento**, con tiempo de ejecución $O(n \log n)$ para el peor caso, sobre las seis secuencias generadas, para cada uno de los tres tamaños de secuencia.

La salida del programa es tiempo **promedio y desviación estándar**, de cada uno de los algoritmos de ordenamiento ejecutados, **los cuales deben estar en una misma línea para su mejor lectura**.

4. Estudio experimental

Se desea comparar el rendimiento de **Counting-sort** y **Quicksort** sobre el mismo conjunto de datos. El programa de pruebas debe ser ejecutado con la siguiente línea de comandos:

```
>./prueba_ord.py -t 2 -i 3 --cs -g 10000 20000 40000 80000 160000
```

También se debe hacer un estudio experimental para comparar el tiempo de ejecución de los algoritmos **Bucket-sort** y **Quicksort**. Para ello el programa de pruebas debe ser ejecutado de la siguiente forma:

```
>./prueba_ord.py -t 1 -i 3 --bs -g 10000 20000 40000 80000 160000
```

Para mostrar los resultados del estudio experimental, debe presentar un reporte, en formato PDF, que conteniendo lo siguiente:

1. Descripción de la plataforma utilizada para correr los experimentos, es decir, sistema de operación, modelo de CPU y memoria RAM del computador.
2. Dos tablas, una tabla por cada experimento realizado. Cada tabla debe contener el tiempo promedio todos los algoritmos de ordenamiento para los tamaños de secuencia usados.
3. Para cada tabla de resultados, debe presentar la gráfica de tiempo de ejecución correspondiente.

5. Condiciones de entrega

La versión final del código del laboratorio y el informe deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem5_X.tar.xz*, donde *X* es el número de carné del estudiante. La entrega del archivo *LabSem5_X.tar.xz*, debe hacerse al profesor del laboratorio por email, antes de las 9:00 pm del día domingo 09 de febrero de 2020.

Referencias

- [1] AHO, A., HOPCROFT, J., AND ULLMAN, J. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1998.
- [2] BRASSARD, G., AND BRATLEY, P. *Fundamentals of Algorithmics*. Prentice Hall, 1996.
- [3] CORMEN, T., LEISERSON, C., RIVEST, R., AND STEIN, C. *Introduction to algorithms*, 3rd ed. MIT press, 2009.
- [4] KALDEWAIJ, A. *Programming: the derivation of algorithms*. Prentice-Hall, Inc., 1990.