Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Enero-Marzo 2020

Algoritmos de Ordenamiento Parte II

1. Introducción

El objetivo de este laboratorio es el agregar nuevos algoritmos de ordenamientos, y extender la librería ORDENAMIENTO.PY. En específico se quiere agregar variantes del algoritmo de ordenamiento *mergesort*.

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].

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 aplicar esa prueba sobre los algoritmos de ordenamiento. También el usuario puede indicar el tamaño de la secuencia de entrada. 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.

La ejecución de PRUEBA_ORD.PY se hace por medio de la siguiente línea de comando:

```
>./prueba_ord.py [-n <valor>] [-i <valor>] [-t <valor>] [--all|--nlgn|--On2]
```

La semántica de los parámetros de entrada es la siguiente:

- **-n**: Número de elementos de la secuencia de entrada.
- **-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.

Los parámetros –**all**, –**nlgn** y –**On2**, 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. Todos los 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. Los valores por de defecto son:

- **-n**: 1000, es decir, la secuencia por defecto tienen mil elementos.
- **-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.

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

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

El programa usa los valores por defecto.

```
>./prueba_ord.py -n 1500 -t 6 -all
```

Se ejecutará la prueba 1, con seis secuencias diferentes, y las secuencias tienen 1500 elementos. Se ejecutarán todos los algoritmos de ordenamiento del módulo ORDENAMIENTO sobre las seis secuencias generadas.

La salida del programa es tiempo promedio y desviación estándar, de cada uno de los algoritmos de ordenamiento ejecutados.

4. Estudio experimental

Se quiere graficar los tiempos de ejecución de los algoritmos de ordenamiento, dados diferentes 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. Adicionalmente, se proveerá del programa PRUEBA_GRAFICAR_PUNTOS.PY que muestra como trabaja el módulo GRAFICAR_PUNTOS.

Debe ejecutar todos los algoritmos de ordenamiento, sobre seis secuencias de tipo 1, con diferentes tamaño, y cada secuencia debe ser ejecutada cuatro veces. Es decir, el programa debe ser ejecutado con la siguiente línea de comando:

```
>./prueba_ord.py -all -n X -t 1 -i 4
```

Donde X se sustituye por el tamaño de la secuencia a usar. El número de elementos de la secuencia va a ser escogido por usted. El objetivo es poder observar claramente en una gráfica el crecimiento del tiempo de ejecución de los algoritmos, tal que, sea posible distinguir la tendencia de crecimiento $O(n \log n)$ y $O(n^2)$.

En específico, debe presentar un reporte 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. Una tabla que contenga los seis tamaño de secuencia utilizados, y el tiempo promedio todos los algoritmos de ordenamiento para cada tamaño de secuencia.
3. La gráfica correspondiente a la tabla de resultados generada por el módulo GRAFICAR_PUNTOS.

El informe deberá estar en formato PDF.

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 *LabSem3_X.tar.xz*, donde X es el número de carné del estudiante. La entrega del archivo *LabSem3_X.tar.xz*, debe hacerse al profesor del laboratorio por email, antes de las 11:50 pm del día domingo 26 de enero 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.

Guillermo Palma / gvpalma@usb.ve / enero de 2020