Programa 1

Estructuras de Datos y Algoritmos 2, Grupo 6

José Mauricio Matamoros de Maria y Campos

Entrega: Viernes 7 de Febrero, 2020

1. Introducción

El algoritmo de ordenamiento por inserción (*Insertion Sort*) se considera un buen algoritmo de ordenamiento para un número relativamente pequeño de elementos y funciona de manera muy similar a la forma en que una persona ordenaría una mano de cartas.

- Comience con la mano izquierda vacía y las cartas hacia abajo en la mesa.
- Retire la primera carta de la mesa y colóquela en su mano izquierda.
- Retire una carta de la mesa a la vez e insértela en la posición correcta en su mano izquierda.
- Para encontrar la posición correcta de una carta, compárela con cada una de las cartas que ya tiene en la mano, de derecha a izquierda.
- Note que las cartas que tiene en la mano izquierda están ordenadas en todo momento y que estas cartas estaban originalmente en la mesa, encima de las otras.

La complejidad (en tiempo) del algoritmo es $O(n^2)$, mientras que su uso de memoria es bastante eficiente al requerir un solo elemento adicional para almacenar la llave de ordenamiento.

El algoritmo es el siguiente:

Algorithm 1 Insertion Sort

```
1:  procedure INSERTION-SORT(A)                    ▷ A: conjunto de elementos a ordenar
2:      for j ← 2 to n do
3:          key ← A[j]
4:          i ← j − 1
5:          while i > 0 and A[i] > key do          ▷ Insertar A[j] en el conjunto ordenado A[1..j − 1]
6:              A[i + 1] ← A[i]
7:              i ← i − 1
8:          end while
9:          A[i + 1] ← key
10:     end for
11: end procedure
```

2. Instrucciones

Implemente el algoritmo de ordenamiento por inserción (*Insertion Sort*) en el lenguaje de programación de su preferencia (se recomienda Python), de acuerdo a las especificaciones del Apartado 2.1.

Ejecute el programa con los conjuntos de datos mostrados a continuación. Compare y discuta sus resultados con sus compañeros de clase.

- 7, 4, 2, 9, 8, 1, 3, 0, 5, 6
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- 9, 8, 7, 6, 5, 4, 3, 2, 1, 1
- 0, 1, 2, 3, ..., 98, 99, 100
- 100, 99, 98, ..., 3, 2, 1, 0

2.1. Especificaciones del programa

1. **Entrada**: El programa lee automáticamente los números a ordenar del archivo `numbers.txt`
2. **Salida:** El programa imprime en pantalla:
   a) El conjunto de números A obtenidos de `numbers.txt`
   b) El conjunto $S = A$ una vez ordenado
   c) El número de pasos u operaciones requeridos para ordenar el conjunto A

   Por ejemplo:

   ```
   A = [5, 2, 4, 6, 1, 3]
   S = [1, 2, 3, 4, 5, 6]
   N = 14
   ```

   Importante: No imprima nada más.
3. Verifique que su algoritmo funcione con números reales en notación decimal, tanto positivos como negativos.
4. Su programa deberá leer sólo la primera línea del archivo `numbers.txt`
5. Su programa deberá emitir una advertencia en caso de que el archivo `numbers.txt` no se encuentre en el mismo directorio o no pueda ser leído, y terminar normalmente (robustez).
6. Su programa deberá emitir una advertencia en caso de que el archivo `numbers.txt` contenga en la primera línea datos no numéricos o bien números en una notación diferente a la decimal. El programa debe terminar normalmente (robustez).

Importante: Tenga en cuenta que su programa será evaluado por otro programa, por lo que si incumple con la especificación o si éste produce una salida diferente a la esperada, podría no obtenerse una calificación aprobatoria.

2.2. `numbers.txt`

El archivo `numbers.txt` estará formado únicamente por números reales separados por comas, sin espacios.

Para validar la robustez de su programa, éste tendrá que ser capaz de leer números separados por comas y caracteres de espacio (espacios y tabulador).

Anexo 1: Cómo leer números de un archivo con Python

Para leer números separados por comas de un archivo de texto, introduciéndolos en un arreglo utilizando Python 3.0 o posterior, emplee el siguiente código:

```
def read_numbers(file_path):
        with open(file_path, 'r') as f:
                return f.readline().split(',')
```