Algoritmos 2023/24

Grado en Inteligencia Artificial

Práctica 2

Fecha límite de entrega: viernes 20 de octubre, a las 23:59

Ordenación por inserción y ordenación *shell*: El problema consiste en ordenar ascendentemente un vector de n números enteros. Para ello se utilizarán la *ordenación por inserción* y la *ordenación Shell*:

```
procedimiento Ordenación por inserción (var v[1..n])
  para i := 2 hasta n hacer
    x := v[i] ;
    j := i-1 ;
    mientras j > 0 y v[j] > x hacer
      v[j+1] := v[j] ;
      j := j-1
    fin mientras ;
    v[j+1] := x
  fin para
fin procedimiento

procedimiento Ordenación shell {Hibbard} (var v[1..n])
  incrementos := [ 2^k-1 , ... , 15, 7, 3, 1 ] (k = mayor entero tal que 2^k-1 < n)
  para cada incremento en incrementos hacer
    para i := incremento+1 hasta n hacer
      tmp := v[i];
      j := i;
      seguir := cierto;
      mientras j-incremento > 0 y seguir hacer
        si tmp < v[j-incremento] entonces
          v[j] := v[j-incremento];
          j := j-incremento
        sino  seguir := falso
        fin si
      fin mientras;
      v[j] := tmp
    fin para
  fin para cada
fin procedimiento
```

1. Implemente en PYTHON los algoritmos de ordenación por inserción y ordenación Shell con incrementos de Hibbard. La figura 1 proporciona el código para implementar los incrementos.
2. Valide el correcto funcionamiento de la implementación. La salida debería ser como sigue:

```
Inicializacion aleatoria
3, -3, 0, 17, -5, 2, 11, 13, 6, 1, 7, 14, 1, -2, 5, -14, -2
ordenado? 0
Ordenacion por Insercion
-14, -5, -3, -2, -2, 0, 1, 1, 2, 3, 5, 6, 7, 11, 13, 14, 17
ordenado? 1

Inicializacion descendente
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
ordenado? 0
Ordenacion Shell
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
ordenado? 1
```

```
def hibbard_increments(array_length):
    increments = []
    k = 1
    gap = 2**k - 1
    while gap < array_length:
        increments.insert(0, gap)
        k += 1
        gap = 2**k - 1
    return increments

def shell_sort_hibbard(v):
   increments = hibbard_increments(len(v))
   return shell_sort_aux(v,increments)

def shell_sort_aux(v,increments):
""" Escribe el resto del codigo de Shell aqui, sabiendo que increments es un vector de incrementos.
Para usar el algoritmo de ordenación se llamará a la función shell_sort_hibbard. """

def ins_sort(v):
""" Escribe el codigo de Ordenacion por Insercion aqui """
```

Figura 1: Código para las cabeceras de los algoritmos y para generar los incrementos de Hibbard.

```
Ordenacion por insercion con inicializacion aleatoria
+-------+--------------------+-----------------+---------------+-----------------+
|   n   |      t(n) (ns)     | t(n) / (n**1.8) | t(n) / (n**2) | t(n) / (n**2.2) |
+-------+--------------------+-----------------+---------------+-----------------+
|  128  |  916720.640000(*)  |    147.658708   |    55.952188  |     21.201914   |
|  256  |   3998208.000000   |    184.941104   |    61.007812  |     20.125073   |
|  512  |  14960640.000000   |    198.730371   |    57.070312  |     16.389144   |
| 1024  |  58483456.000000   |    223.096680   |    55.774170  |     13.943542   |
| 2048  |  244634624.000000  |    267.993345   |    58.325439  |     12.693811   |
| 4096  | 1020544768.000000  |    321.058487   |    60.829208  |     11.524980   |
| 8192  | 4235260160.000000  |    382.629275   |    63.110294  |     10.409317   |
| 16384 | 16822118912.000000 |    436.440262   |    62.667276  |      8.998225   |
+-------+--------------------+-----------------+---------------+-----------------+
```

Figura 2: Parte de la posible salida por pantalla de la ejecución del programa principal

3. Determine los tiempos de ejecución para distintos tamaños del vector y para tres diferentes situaciones iniciales: (a) el vector ya está ordenado en orden ascendente, (b) el vector ya está ordenado pero en orden descendente, y (c) el vector está inicialmente desordenado (inicialización aleatoria). Para las inicializaciones (a) y (b), se recomienda consultar el funcionamiento de la función `numpy.arange`.

4. Calcule empíricamente las complejidades de los algoritmos para cada una de las diferentes situaciones iniciales del vector (i.e., 6 tablas) (figura 2).

5. Entregue los ficheros de código PYTHON y el fichero `.txt` con el informe usando la tarea *Entrega Práctica 2* en la página de Algoritmos en `https://campusvirtual.udc.gal`. Se recuerda que el límite para completar la tarea es el viernes 20 de octubre a las 23:59, y una vez subidos los archivos no se podrán cambiar. **Todos los compañeros que forman un equipo deben entregar el trabajo.**