Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Abril-Julio 2021

Quicksort

1. Introducción

El objetivo de este laboratorio es el de agregar a la librería de ordenamiento `Sortlib.kt` varias versiones del algoritmo Quicksort, y hacer una estudio experimental con los algoritmos implementados hasta ahora en la librería.

2. Actividades a realizar

La primera actividad consiste en generación tres nuevas clases de secuencias, que se aplican con la opción *-s* de la línea de comandos. A continuación se presentan con su indicador y se describen:

cerouno: secuencia de N elementos de ceros y unos generados aleatoriamente.

mitad: secuencia de N elementos de tipo entero con la forma $1, 2, \ldots, N/2, N/2, \ldots, 2, 1$ si N es par, y con la forma $1, 2, \ldots, \lceil N/2 \rceil, \ldots, 2, 1$ si N es impar.

repetido: secuencia de N elementos de tipo entero, generados aleatoriamente en el intervalo $[0, \lceil N^{2/3} \rceil]$.

Como segunda actividad, debe agregar varias versiones del algoritmo Quicksort a la librería `Sortlib.kt`, que van desde la versión básica, hasta versiones de alto rendimiento. Todos estos algoritmos debe ejecutarse cuando se escoge en la familia de algoritmos `nlgn`, de la opción *-a* de la línea de comando. A continuación se introducen las variantes de Quicksort:

Quicksort: versión de Quicksort presentada en la página 171 de [2].

Randomized-quicksort: versión de Quicksort introducida en la página 179 de [2].

***Quicksort with 3-way partitioning*:** Esta es una versión de Quicksort presentada por Sedgewick y Bentley en una charla [5]. Este Quicksort utiliza el procedimiento de partición de un arreglo propuesto por Bentley y McIlroy [1]. En teoría este algoritmo es eficiente en la mayoría de los casos, incluyendo secuencias que están casi ordenados y con muchas repeticiones de elementos. La figura 1 muestra esta versión de Quicksort.

Introsort: Este algoritmo fue presentado por David Musser [4], y es una versión de Quicksort que limita la profundidad de la recursión. Cuando la profundidad de la recursión excede el límite establecido, entonces el algoritmo procede a ordenar los elementos restantes usando Heapsort. Esto hace que este algoritmo tenga un tiempo de ejecución en el peor caso de $O(n \log(n))$. Este algoritmo es ampliamente usado en la práctica, siendo el algoritmo de ordenamiento de la GNU C Library (glibc) [1], de la SGI C++

[1] https://www.gnu.org/software/libc/

```
void quicksort(Item a[], int l, int r)
{ int i = l-1, j = r, p = l-1, q = r; Item v = a[r];
  if (r <= l) return;
  for (;;)
    {
      while (a[++i] < v) ;
      while (v < a[--j]) if (j == l) break;
      if (i >= j) break;
      exch(a[i], a[j]);
      if (a[i] == v) { p++; exch(a[p], a[i]); }
      if (v == a[j]) { q--; exch(a[j], a[q]); }
    }
  exch(a[i], a[r]); j = i-1; i = i+1;
  for (k = l; k < p; k++, j--) exch(a[k], a[j]);
  for (k = r-1; k > q; k--, i++) exch(a[i], a[k]);
  quicksort(a, l, j);
  quicksort(a, i, r);
}
```

Figura 1: Pseudocódigo del algoritmo *Quicksort with 3-way partitioning*. Fuente [5].

Standard Template Library [2] y de la Microsoft .NET Framework Class Library [3]. La Figura 2 muestra el pseudocódigo del algoritmo INTROSORT. El procedimiento PARTITION es el que realiza la división de arreglo, y para ello debe hacer uso del procedimiento de partición propuesto por Hoare para el algoritmo Quicksort original [3], y que se muestra en la Figura 3. Se usará un valor de SIZE_THRESHOLD de 32. El valor del límite de profundidad $2 * \text{FLOOR\_LG(B-F)}$ corresponde a $2\lfloor \log_2 B - F \rfloor$, donde $B - F$ es el número de elementos por ordenar del arreglo. Como puede observar en este algoritmo, cuando se tiene que ordenar un número de elementos menor o igual a SIZE_THRESHOLD, entonces se usa el algoritmo INSERTION_SORT. Debe crear una versión modificada de `Insertionsort` para que realice el ordenamiento de una sección del arreglo de entrada.

La tercera actividad es un estudio experimental. Debe ejecutar la familia de algoritmos `nlgn`, que viene dada con la opción *-a* de la línea de comando. Los algoritmos deben ordenar todas las clases de secuencias. Cada secuencia debe ser ejecutada tres veces, es decir, se debe aplicar la opción `-t` 3. El tamaño de las secuencias del estudio experimental son 10.000, 20.000, 30.0000, 40.000 y 50.0000.

Una vez efectuados los resultados experimentales, debe realizar un breve reporte con los resultados obtenidos y su análisis. El reporte debe contener las seis imágenes generadas por la ejecución de las seis clases de secuencias (`random`, `sorted`, `inv`, `cerouno`, `mitad` y `repetido`). También el reporte debe incluir un análisis de los resultados mostrados en las gráficas. Dentro de su análisis debe indicar si los resultados obtenidos se corresponden a lo esperado en la teoría. En el reporte debe incluir los datos de la plataforma donde se ejecutaron los algoritmos: sistema de operación, modelo de CPU, cantidad de memoria RAM del computador, versión del compilador Kotlin y versión de la JVM utilizada. **El reporte debe estar en formato PDF**.

[2] http://www.sgi.com/tech/stl/

[3] https://msdn.microsoft.com/en-us/library/6tf1f0bc(v=vs.110).aspx

```
Algorithm INTROSORT(A, f, b)
                                Inputs: A, a random access data structure containing the sequence
                                    of data to be sorted, in positions A[f], ..., A[b − 1];
                                  f, the first position of the sequence
                                  b, the first position beyond the end of the sequence
                                Output: A is permuted so that A[f] ≤ A[f+1] ≤ ... ≤ A[b − 1]
    INTROSORT_LOOP(A, f, b, 2 * FLOOR_LG(b − f))
    INSERTION_SORT(A, f, b)

Algorithm INTROSORT_LOOP(A, f, b, depth_limit)
                                Inputs: A, f, b as in INTROSORT;
                                  depth_limit, a nonnegative integer
                                Output: A is permuted so that A[i] ≤ A[j]
                                  for all i, j: f ≤ i < j < b and size_threshold < j − i
    while b − f > size_threshold
        do if depth_limit = 0
                then HEAPSORT(A, f, b)
                     return
           depth_limit := depth_limit − 1
           p := PARTITION(A, f, b, MEDIAN_OF_3(A[f], A[f+(b−f)/2], A[b−1]))
           INTROSORT_LOOP(A, p, b, depth_limit)
           b := p
```

Figura 2: Pseudocódigo del algoritmo INTROSORT. Fuente [4].

```
PARTITION(A, p, r, x)
 1  i = p − 1
 2  j = r
 3  while TRUE
 4      repeat
 5          j = j − 1
 6      until A[j] ≤ x
 7      repeat
 8          i = i + 1
 9      until A[i] ≥ x
10      if i < j
11          exchange A[i] with A[j]
12      else return j
```

Figura 3: Procedimiento de partición propuesto por Hoare para el algoritmo Quicksort.

3. Sobre la implementación

El código encargado de la manipulación de los argumentos de la línea de comandos, debe hacerse en por medio de funciones en Kotlin implementadas por usted, no se puede hacer uso de librerías externas. Su entrega debe contener los archivos `Sortlib.kt`, `runSortlib.sh`, `Makefile`, `Main.kt`, y los archivos necesarios de la librería *libPlotRuntime*. El archivo `runSortlib.sh`, es un *script* cuyo único objetivo es la ejecución del cliente

de `Main.kt`. Todo el código debe usar la guía de estilo Kotlin indicada en clase. Asimismo, el código debe estar debidamente documentado.

4. Condiciones de entrega

La versión final del código del laboratorio, el reporte y la declaración de autenticidad firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem4_X.tar.xz*, donde X es el número de carné del estudiante. La entrega del archivo *LabSem4_X.tar.xz* debe hacerse por la plataforma Classroom, antes de las 11:50 pm del día domingo 6 de junio de 2021.

Referencias

[1] Bentley, J. L., and McIlroy, M. D. Engineering a sort function. *Software: Practice and Experience 23*, 11 (1993), 1249–1265.

[2] Cormen, T., Leirserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*, 3ra ed. McGraw Hill, 2009.

[3] Hoare, C. A. Quicksort. *The Computer Journal 5*, 1 (1962), 10–16.

[4] Musser, D. R. Introspective sorting and selection algorithms. *Softw., Pract. Exper. 27*, 8 (1997), 983–993.

[5] Sedgewick, R., and Bentley, J. Quicksort is optimal. `https://www.cs.princeton.edu/~rs/talks/QuicksortIsOptimal.pdf`, Enero 2002.

Guillermo Palma / gvpalma@usb.ve / Junio 2021