Algoritmos y Estructuras de Datos
UT9-PD4

Santiago Blanco

05-07-2025

Introducción

Secuencias de incrementos (**gaps**) para ShellSort. La elección de estos incrementos afecta directamente el rendimiento:

Sedgewick (1990s)

Propone secuencias con propiedades matemáticas que aseguran una complejidad de $O(n^{\frac{4}{3}}$. Basadas en combinaciones de potencias de 2 y 3, son buenas en teoría pero algo menos eficientes en la práctica.

- $h_k = 9 * 4^k - 9 * 2^k + 1$
- $h_k = 4^k - 3 * 2^k + 1$

$1, 5, 19, 41, 109, 209, 505, 929, \ldots$

La secuencia de Sedgewick ha demostrado funcionar bien en la práctica, especialmente con estructuras de datos más grandes:
- Reducen el número de comparaciones y asignaciones.
- Son simples de generar mediante fórmulas.
- Tienen mejor comportamiento en distribuciones aleatorias.

Referencias:
- R. Sedgewick, “Analysis of Shellsort and Related Algorithms”, PhD thesis, 1975.

Ciura (2001)

Propone una secuencia obtenida empíricamente:

$1, 4, 10, 23, 57, 132, 301, 701$

Se ha comprobado que es una de las más eficaces en promedio para arreglos pequeños y medianos. Para tamaños mayores, se puede extender con la fórmula $h_k = \lfloor 2.25 * h_{k-1} \rfloor$.

Referencias:
- V. Ciura, “Best Increments for the Average Case of Shellsort” (2001)

Tokuda (1992)

Utiliza una fórmula basada en una razón de crecimiento de aproximadamente 2.25:

$h_k = \left\lceil \left(\frac{9}{4}\right)^k - \frac{1}{\frac{9}{4}-1} \right\rceil$

Ofrece buen rendimiento en arreglos grandes, incluso compitiendo con Ciura en ciertas configuraciones.

Resultados de pruebas en Java:

```
HABEMUS RESULTADUS:
Ciura             : 0.526 ms
Tokuda            : 0.628 ms
Sedgewick         : 0.470 ms
```

Sedgewick es mi favorito

Criterio de selección de pivote en QuickSort

- **Elemento aleatorio**: reduce la probabilidad del peor caso ($O(n^2)$). Recomendado por su simplicidad y robustez en entradas adversas [1].
- **Mediana de tres**: considera los elementos primero, medio y último. Mejora el rendimiento en datos parcialmente ordenados [2].
- **Mediana de cinco o más**: mejora teórica en balance de partición, pero con mayor costo computacional. Usada en investigaciones como MQuickSort [3].
- **Median-of-medians**: garantiza O(n log n) en el peor caso. Su complejidad constante lo hace poco práctico en QuickSort estándar [4].

Implementaciones en lenguajes modernos

- **Java**: utiliza QuickSort con doble pivote (Dual-Pivot Quicksort), introducido por Yaroslavskiy. Mejora la eficiencia práctica respecto al QuickSort clásico [5].
- **C++ (STL)**: emplea Introsort, que comienza con QuickSort (mediana de tres), y cambia a Heapsort si la recursión se profundiza [6].
- **Python**: usa Timsort, una combinación adaptativa de Mergesort e Insertion Sort, por su estabilidad y rendimiento en datos parcialmente ordenados

Resultados experimentales

Se implementaron variantes de QuickSort con:

- pivote aleatorio,
- mediana de tres,
- primer elemento (como control).

Resultados

```
OJO:
Pivote Fijo                    Tiempo: 18.40 ms
Pivote Aleatorio               Tiempo: 13.11 ms
Mediana de Tres                Tiempo: 9.17 ms
```

Referencias

[1] Hoare, C. A. R. “Quicksort”, **The Computer Journal**, 1962. [2] Sedgewick, R. “Algorithms in C”, Addison-Wesley, 1998. [3] M. Weiss et al., “Enhancing QuickSort using dynamic pivot”, **ResearchGate**, 2012. [4] Blum et al., “Time bounds for selection”, **J. Computer and System Sciences**, 1973. [5] Yaroslavskiy, V. “Dual-Pivot Quicksort”, **OpenJDK Blog**, 2010. [6] Musser, D. R. “Introspective Sorting”, **Software Practice & Experience**, 1997.