Método de Ordenamiento Shell Sort

Matías Astorga
Maximiliano Meza
Romina Navarrete

Julio 26, 2013

Análisis de Algoritmos
Universidad Tecnológica Metropolitana

Índice

Índice

Índice

Índice

Índice

Índice

Objetivos

General

Exponer sobre el método de Shell Sort

Secundarios

- Listar las características más importantes para comprender este método de ordenamiento.
- Mostrar en detalle el funcionamiento de este método con un programa y video explicativo.

Introducción

- Shell Sort es un algoritmo de ordenamiento, propuesto en 1959 por Donald Shell.
- Basado en comparaciones e intercambios.
- Algoritmo sencillo, y con mejores resultados que otros algoritmos de ordenamiento.

Introducción

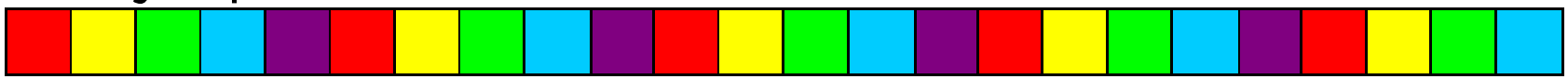
- Shell Sort es un algoritmo de ordenamiento, propuesto en 1959 por Donald Shell.
- Basado en comparaciones e intercambios.
- Algoritmo sencillo, y con mejores resultados que otros algoritmos de ordenamiento.

Introducción

- Shell Sort es un algoritmo de ordenamiento, propuesto en 1959 por Donald Shell.
- Basado en comparaciones e intercambios.
- Algoritmo sencillo, y con mejores resultados que otros algoritmos de ordenamiento.

El Algoritmo

- Shell Sort mejora el ordenamiento por inserción, comparando los elementos separados por varias posiciones (Saltos o **gap**).
- Por ejemplo, Salto = 5.



¿Cuál es el criterio para calcular los saltos?

Existen varios, pero el usado por Donald Shell fue:

$$k = \frac{N}{2}$$

N = Largo del arreglo

Ejemplo

- Entonces si tenemos un arreglo de 10 casillas, nuestros saltos serian:
- $k = \frac{10}{2} = 5$
- Las primeras comparaciones serán de saltos de 5.

Ejemplo

- Entonces si tenemos un arreglo de 10 casillas, nuestros saltos serian:
- $k = \frac{10}{2} = 5$
- Las primeras comparaciones serán de saltos de 5.

Ejemplo

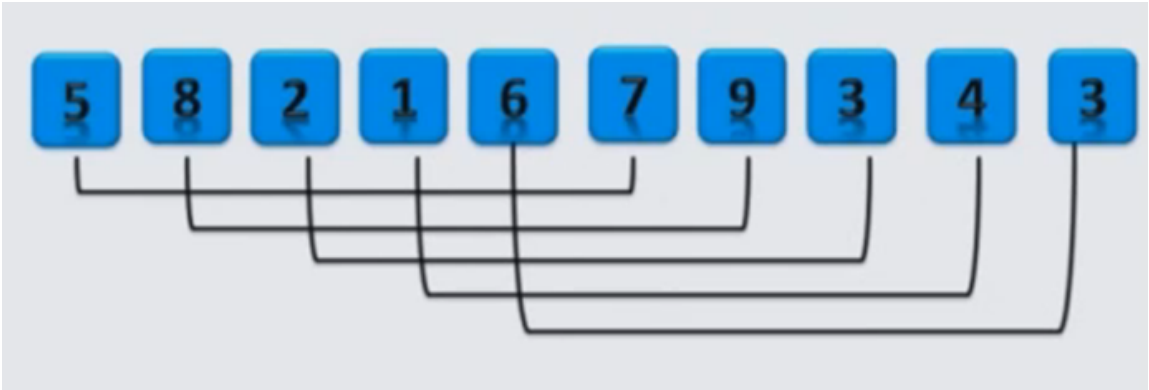
- Entonces si tenemos un arreglo de 10 casillas, nuestros saltos serian:
- $k = \frac{10}{2} = 5$
- Las primeras comparaciones serán de saltos de 5.

Ejemplo

- Entonces si tenemos un arreglo de 10 casillas, nuestros saltos serian:
- $k = \frac{10}{2} = 5$
- Las primeras comparaciones serán de saltos de 5.

Ejemplo

- Entonces si tenemos un arreglo de 10 casillas, nuestros saltos serian:
- $k = \frac{10}{2} = 5$
- Las primeras comparaciones serán de saltos de 5.



Ejemplo

- Una vez que los datos están ordenados, volvemos a calcular el nuevo salto a usar:
- $K = \frac{5}{2} = 2,5 = 2$
- Ahora las comparaciones serán de saltos de 2.

Ejemplo

- Una vez que los datos están ordenados, volvemos a calcular el nuevo salto a usar:
- $K = \frac{5}{2} = 2,5 = 2$
- Ahora las comparaciones serán de saltos de 2.

Ejemplo

- Una vez que los datos están ordenados, volvemos a calcular el nuevo salto a usar:
- $K = \frac{5}{2} = 2,5 = 2$
- Ahora las comparaciones serán de saltos de 2.

Ejemplo

- Una vez que los datos están ordenados, volvemos a calcular el nuevo salto a usar:
- $K = \frac{5}{2} = 2,5 = 2$
- Ahora las comparaciones serán de saltos de 2.

Ejemplo

- Una vez que los datos están ordenados, volvemos a calcular el nuevo salto a usar:
- $K = \frac{5}{2} = 2,5 = 2$
- Ahora las comparaciones serán de saltos de 2.

Ejemplo

- Finalmente calculamos el último salto:
- $k = \frac{2}{2} = 1$
- Acá, vemos como el algoritmo se transforma en un algoritmo de inserción directa.

Ejemplo

- Finalmente calculamos el último salto:
- $k = \frac{2}{2} = 1$
- Acá, vemos como el algoritmo se transforma en un algoritmo de inserción directa.

Ejemplo

- Finalmente calculamos el último salto:
- $k = \frac{2}{2} = 1$
- Acá, vemos como el algoritmo se transforma en un algoritmo de inserción directa.

Ejemplo

- Finalmente calculamos el último salto:
- $k = \frac{2}{2} = 1$
- Acá, vemos como el algoritmo se transforma en un algoritmo de inserción directa.

Código

```
for(inc = n/2; inc > 0; inc /= 2) {
        for(i = inc; i<n; i++) {
             temp = arreglo[i];
             for(j = i; j >= inc ; j -= inc) {
                  if(temp < arreglo[j-inc])
                  arreglo[j] = arreglo[j-inc];
                  else
                  break;
              }
             arreglo[j] = temp;
         }
}
```

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- La idea principal es disminuir el movimiento de datos.
- Algoritmo de Ordenación Interna.
- Shell Sort inicia una comparación de los elementos más alejados(que distan entre sí un intervalo h_1).
- Luego se van disminuyendo los intervalos, para poder comparar elementos más cercanos.
- Hasta reducirse al método de ordenación por insercion.
- En sus primeros saltos trata de desplazar los elementos más chicos al inicio y los más grandes al final.

Características

- También llamado **Ordenación por disminución de Intervalos**
- Los intervalos escogidos $h_1, h_2, h_4, ...,$ se denominan **secuencia de incrementos**.
- $h_k - ordenado$, elementos ordenados y que distan entre sí un intervalo h_k
- Un array $h_k - ordenado$ lo sigue estando después de $h_{k-1} - ordenacion$.

Características

- También llamado **Ordenación por disminución de Intervalos**
- Los intervalos escogidos $h_1, h_2, h_4, ...,$ se denominan **secuencia de incrementos**.
- $h_k - ordenado$, elementos ordenados y que distan entre sí un intervalo h_k
- Un array $h_k - ordenado$ lo sigue estando después de $h_{k-1} - ordenacion$.

Características

- También llamado **Ordenación por disminución de Intervalos**
- Los intervalos escogidos $h_1, h_2, h_4, ...,$ se denominan **secuencia de incrementos**.
- $h_k - ordenado$, elementos ordenados y que distan entre sí un intervalo h_k
- Un array $h_k - ordenado$ lo sigue estando después de $h_{k-1} - ordenacion$.

Características

- También llamado **Ordenación por disminución de Intervalos**
- Los intervalos escogidos $h_1, h_2, h_4, ...,$ se denominan **secuencia de incrementos**.
- $h_k - ordenado$, elementos ordenados y que distan entre sí un intervalo h_k
- Un array $h_k - ordenado$ lo sigue estando después de $h_{k-1} - ordenacion$.

Secuencias de Incrementos

Es válida cualquier secuencia que cumpla:

- Los intervalos van disminuyendo
- El último intervalo es de tamaño 1

Shell

$$\frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \cdots \frac{N}{2^k}, \ldots, 1$$

Hibbard

$$1, 3, 7, \ldots, 2^k - 1$$

Secuencias de Incrementos

Es válida cualquier secuencia que cumpla:

- Los intervalos van disminuyendo
- El último intervalo es de tamaño 1

Shell

$$\frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \cdots \frac{N}{2^k}, \ldots, 1$$

Hibbard

$$1, 3, 7, \ldots, 2^k - 1$$

Secuencias de Incrementos

Knuth

$$1, 4, 13, ..., \frac{3^k - 1}{2}$$

Sedgewick

$$1, 5, 19, 41, 109, ..., 9 * 4^k - 9 * 2^k + 1$$

o

$$1, 5, 19, 41, 109, ..., 4^k - 3 * 2^k + 1$$

Complejidad

- La complejidad de este algoritmo es variable, ya que existen diferentes formas de calcular los saltos.
- Inicialmente, por el salto $K = \frac{N}{2}$ propuesto por Shell, su complejidad es:

Complejidad Shell

- Mejor Caso: $O(n \log n)$
- Caso Promedio: $O(n^2)$
- Peor Caso: $O(n^2)$

Complejidad

- La complejidad de este algoritmo es variable, ya que existen diferentes formas de calcular los saltos.
- Inicialmente, por el salto $K = \frac{N}{2}$ propuesto por Shell, su complejidad es:

Complejidad Shell

- Mejor Caso: $O(n \log n)$
- Caso Promedio: $O(n^2)$
- Peor Caso: $O(n^2)$

Complejidad - Mejoras

Caso Promedio

- Shell: $O(n^2)$
- Hibbard: $O(n^{5/4})$
- Sedgewick: $O(n^{7/6})$

Complejidad - Mejoras

Peor de los Casos

Cuando N es una potencia de 2 y además:

1. Los elementos grandes en posiciones pares
2. Los pequeños en posiciones impares

- Shell: $O(n^2)$
- Hibbard: $O(n^{4/3})$
- Sedgewick: $O(n \log^2 n)$

Complejidad Empírica

Tiempo de ejecución en ms para InsertionSort y ShellSort.

N	Ordenación por inserción	ShellSort (con distintas secuencias de intervalos)		
		Secuencia de shell	Sólo intervalos impares	Dividiendo por 2,2
1.000	122	11	11	9
2.000	483	26	21	23
4.000	1.936	61	59	54
8.000	7.950	153	141	114
16.000	32.560	358	322	269
32.000	131.911	869	752	575
64.000	520.000	2.091	1.705	1.249

- La ventaja de Shell Sort es que es eficiente en listas de tamaño medio. Para listas mas grandes, no es la mejor opción.
- Es hasta 5 veces más rápido que Bubble Sort y hasta 2 veces más rápido que Insertion Sort.
- Es menos eficiente que Merge, Heap y Quick Sorts.

- La ventaja de Shell Sort es que es eficiente en listas de tamaño medio. Para listas mas grandes, no es la mejor opción.
- Es hasta 5 veces más rápido que Bubble Sort y hasta 2 veces más rápido que Insertion Sort.
- Es menos eficiente que Merge, Heap y Quick Sorts.

- La ventaja de Shell Sort es que es eficiente en listas de tamaño medio. Para listas mas grandes, no es la mejor opción.
- Es hasta 5 veces más rápido que Bubble Sort y hasta 2 veces más rápido que Insertion Sort.
- Es menos eficiente que Merge, Heap y Quick Sorts.

Conclusiones

- Existen algoritmos más eficientes
- La gran ventaja de shellsort es que pertenece a los denominados algoritmos "in place".
- Actualmente existen mejoras de este algoritmo con una complejidad de $O(n \log^2 n)$

Conclusiones

- Existen algoritmos más eficientes
- La gran ventaja de shellsort es que pertenece a los denominados algoritmos "in place".
- Actualmente existen mejoras de este algoritmo con una complejidad de $O(n \log^2 n)$

Conclusiones

- Existen algoritmos más eficientes
- La gran ventaja de shellsort es que pertenece a los denominados algoritmos "in place".
- Actualmente existen mejoras de este algoritmo con una complejidad de $O(n \log^2 n)$