FLAT_STABLE_SORT

Nuevo algoritmo de ordenación estable

1.- INTRODUCCIÓN

flat_stable_sort es un nuevo algoritmo estable de ordenación, que usa una diminuta memoria adicional (aproximadamente 1% de la memoria usada por los datos).

La memoria adicional necesaria es : Tamaño de los datos / 256 + 8K

Tamaño Datos	**Memoria adicional**	**Porcentaje**
1M	12 K	1.2 %
1G	4M	0.4 %

El algoritmo es eficiente ordenando cualquier tipo de datos desordenados, pero ha sido diseñado para ser extremadamente eficiente cuando los datos están casi ordenados, como por ejemplo:

- Elementos ordenados y se añaden elementos no ordenados al principio y al final.
- Elementos ordenados, y se insertan elementos en posiciones intermedias, o se hacen modificaciones en datos, alterando la secuencia ordenada.
- Elementos inversamente ordenados
- Combinación de los 3 anteriores.

Los resultados obtenidos en la ordenación de 100 000 000 números mas un porcentaje de números añadidos al final o en posiciones intermedias, fueron.

```
random                        |10.78 |
                              |      |
sorted                        | 0.07 |
sorted + 0.1% end             | 0.36 |
sorted +   1% end             | 0.49 |
sorted +  10% end             | 1.39 |
                              |      |
sorted + 0.1% middle          | 2.47 |
sorted +   1% middle          | 3.06 |
sorted +  10% middle          | 5.46 |
                              |      |
reverse sorted                | 0.14 |
reverse sorted + 0.1% end     | 0.41 |
reverse sorted +   1% end     | 0.55 |
reverse sorted +  10% end     | 1.46 |
                              |      |
reverse sorted + 0.1% middle| 2.46 |
reverse sorted +   1% middle| 3.16 |
reverse sorted +  10% middle| 5.46 |
```

Los resultados obtenidos con strings y objetos de diferente tamaño, y con diferente coste de la operación de comparación, fueron igualmente satisfactorios y acordes a lo esperado.

2.- DESCRIPCION DEL ALGORITMO

2.1.- INTRODUCCIÓN

El problema de los algoritmos de mezcla es ¿y donde coloco lo que voy ordenado?. Esa es la justificación de la memoria adicional.

Este algoritmo sigue una estrategia diferente. Los datos se agrupan en bloques de un tamaño fijo. Supongamos, inicialmente, que el numero de elementos a ordenar es un múltiplo del tamaño de un bloque. Mas adelante veremos que se hace cuando no es así.

El otro concepto importante es la secuencia. Una secuencia es un conjunto de bloques, no contiguos cuyos datos están ordenados. Los bloques que forman la secuencia, se definen mediante una posición inicial, y una posición final en un vector de enteros.

Por ejemplo

5	*6*	*7*	*8*
10	12	8	6

Esta secuencia, definida entre las posiciones 5 a 8 de un vector, significa que los bloques que forman la secuencia, están en las posiciones 10, 12, 8 y 6. Los datos en esta secuencia de bloques no contiguos, están ordenados.

La idea del algoritmo es similar a la de otros algoritmos de mezcla. Se ordenan los N bloques, y tenemos N secuencias de un bloque.

Mezclamos la secuencia 0 con la , la 1 con la 2 ….y obtenemos N/2 secuencias.

Volvemos a mezclar las nuevas secuencias, la 0 , con la 1 , la 1 con la 2 …. y obtenemos N/4 secuencias.

Al final de este proceso obtenemos 1 sola secuencia.

Mediante un sencillo algoritmo, y con el vector de las posiciones (index), movemos los bloques para que su posición lógica sea su posición física, y ya tenemos los datos ordenados.

El movimiento de los bloques se hace una sola vez, y es al última operación del algoritmo.

2.1.1.- Mezcla de Bloques

Para mezclar los bloques, utilizo una "mezcladora", que es un buffer circular, muy simple, con el tamaño de dos bloques.

La idea es ir mezclando los elementos de 2 bloques, hasta que uno de ellos queda vacío.
Los elementos mezclados están dentro de la mezcladora

2.1.2.-Mezcla de Secuencias

Tenemos dos secuencias, definidas como 2 rangos de posiciones en un vector de enteros. Los dos rangos pueden ser del mismo o distinto vector.

	0	*1*	*2*	*3*
Secuencia 1	1	2	0	3

	4	*5*	*6*	*7*
Secuencia 2	6	5	4	7

Empezamos mezclando el bloque 1 y el 6. Si por ejemplo se vacía el bloque 1, se rellena con los primeros datos de la mezcladora, y se apunta el 1 a la lista de salida. Repetimos la operación con el siguiente bloque de la secuencia 1, que sería el bloque 2, con lo que queda del bloque 6.

Cuando se vacía un bloque , se rellena con los primeros elementos de la mezcladora, y su número se apunta a la lista de salida.

Para explicar que pasa cuando una secuenciase vacía, y no tiene mas bloques, lo hacemos con un ejemplo. Supongamos que la secuencia 1 está vacía

	0	*1*	*2*	*3*
Secuencia 1				

	4	*5*	*6*	*7*
Secuencia 2			4	7

En la mezcladora tenemos datos ya mezclados y el bloque 4 está parcialmente lleno. Los datos que le faltan al bloque 4 para estar lleno, son los datos de la mezcladora, por lo que los copiamos de la mezcladora al bloque 4, y en la lista de salida apuntamos el 4 y el 7.

Por lo que tendríamos una nueva secuencia, que podría ser algo parecido a ésto.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Secuencia Salida	1	6	2	0	5	3	4	7

Los elementos, en los bloques de una secuencia, están ordenados

2.1.3.- Ejemplo

Tenemos 4 bloques de 4 elementos cada uno. Hemos ordenado cada bloque, por lo que dentro del bloque los elementos están ordenados.

	0	*1*	*2*	*3*
Index	0	1	2	3

	0	*1*	*2*	*3*
Data	5, 7, 12, 23	4, 6, 11, 15	1, 9, 13, 34	8, 14, 17,19

Tenemos 4 secuencias de 1 bloque cada una. Vamos a mezclar las secuencias 0 y 1 , y 2 y 3. Con lo que obtendremos dos secuencias de dos bloques cada una.

Después mezclamos las dos secuencias de dos bloques y obtendremos una secuencia de 4 bloques, que sería la secuencia final.

En la mezcla de bloques 0 y 1

Mezcladora
4,5,6,7,11,12,15

Bloque 0	*Bloque 1*
23	

El bloque 1 está vacío. Lo rellenamos con los 4 primero elementos de la mezcladora, y los que quedan dentro los insertamos por la parte de adelante en el bloque 0 y tendríamos. La nueva secuencia sería el bloque 1 y después el bloque 0. Y esas secuencias las vemos en el Index

	0	*1*	*2*	*3*
Index	1	0	2	3

	0	*1*	*2*	*3*
Data	11, 12, 15, 23	4,5,6, 7	1, 9, 13, 34	8, 14, 17,19

Haciendo lo mismo con los bloques 2 y 3, el resultado sería

	0	*1*	*2*	*3*
Index	1	0	3	2

	0	*1*	*2*	*3*
Data	11, 12, 15, 23	4, 5, 6, 7	14, 17, 19, 34	1, 8, 9, 13

Tenemos ahora dos secuencias de dos bloques. Las secuencias están definidas mediante el índice. La primera secuencia son las posiciones 0y 1 del índice y la segunda secuencia son las posiciones 2 y 3 del índice.

Para hacer la mezcla, cogemos el primer bloque de la primera secuencia (el 1) con el primer bloque de la segunda secuencia (el 3) y empezamos la mezcla.

El bloque que primero se vacía es el 1, por lo que lo rellenamos desde la mezcladora y es el primero de la secuencia. Para sustituir al bloque 1, cogemos el siguiente de la primera secuencia que es el 0 y continuamos con la mezcla.

Ahora el primer bloque que se vacía es el 3. Lo rellenamos desde la mezcladora y lo apuntamos en la lista. Para sustituir al 3 cogemos el siguiente bloque de su secuencia que es el 2, y continuamos con la mezclas.

El siguiente bloque que se vacía es el 0. Lo rellenamos desde la mezcladora y lo añadimos a la lista. Ya solo queda el el bloque 2, porque la primera secuencia se ha agotado. Lo rellenamos desde la mezcladora y lo añadimos a la lista.

Al acabar tenemos

	0	*1*	*2*	*3*
Index	1	3	0	2

	0	*1*	*2*	*3*
Data	12, 13, 14, 15	1, 4, 5, 6	17, 19, 23, 34	7, 8, 9, 11

Ahora tenemos que hacer el movimiento de los bloques. Con un sencillo algoritmo, movemos los bloques y pasamos de la ordenación mediante un índice a la ordenación física. Esto se hace una sola vez, y es el último paso del algoritmo.

	0	*1*	*2*	*3*
Index	0	1	2	3

	0	*1*	*2*	*3*
Data	1, 4, 5, 6	7, 8, 9, 11	12, 13, 14, 15	17, 19, 23, 34

2.1.4.- Detalles internos

La memoria adicional que necesita este algoritmo son los 2 bloques de la mezcladora, y la lista con las posiciones de los bloques.

En el proceso de mezcla intervienen 2 bloques , y los 2 bloques de la mezcladora. El tamaño está ideado para que estos 4 bloques quepan en la caché L1 del procesador. En un I7, un core tiene 32K para datos, pero maneja 2 hebras, por lo que tenemos la mitad 16K.

Son 4 bloques lo que manejamos , por lo que el tamaño de un bloque no puede superar los 4K.

Tamaño del objeto x Tamaño del bloque = 4096

2.2.- NUMERO DE DATOS NO MÚLTIPLO DEL TAMAÑO DE BLOQUE

Cuando el número de elementos a ordenar no es un múltiplo del tamaño del bloque, rellenamos bloques, pero tenemos un bloque final incompleto , al que llamaremos tail y siempre es el último.

Cuando mezclamos dos secuencias, si hay un bloque tail, está siempre en la segunda secuencia.

Para explicar la mezcla de secuencias con el bloque tail, lo vemos con un ejemplo

Cuando existe un bloque tail en la segunda secuencia, la mezcla la hacemos como hemos visto.
Si la secuencia que primero se vacía es la 1, entonces, el procedimiento es como el anteriormente descrito

Si la secuencia que primero se vacía es la segunda, la que tiene tail, hay que hacer un procedimiento diferente

Lo vemos con otro ejemplo

	0	1	2	3	4	5
Index	1	2	0	4	3	5

	0	1	2	3	4	5
Data	38, 42, 44, 46	1, 5, 13, 19	23, 29, 34, 36	22, 25, 27, 28	4, 14, 17, 20	31, 32

Queremos mezclar las secuencias del 0 al 2, con la secuencia del 3 al 5. El bloque 5 es un bloque incompleto o tail.

Empezamos la mezcla de secuencias como hemos visto antes, y llegamos a un punto en el que la segunda secuencia que contiene el tail se vaciá. En ese instante el estado de los datos es:

	0	1	2
Secuencia 1		2	0

El bloque 2 está parcialmente vacío.

	0	1	2
Secuencia 2			

	0	1	2	3	4	5
Secuencia de salida	1	4	3			

	0	1	2	3	4	5
Data	38, 42, 44, 46	1, 4, 5, 13	34, 36	22, 23, 25, 27	14, 17, 19, 20	

Mezcladora	28, 29, 31, 32

Veamos que los datos de secuencia 1 que faltan por mezclar. Y añadamos a esa secuencia el bloque tail vacío al final.

	2	0	5
Datos pendientes de la secuencia 1	34, 36	38, 42, 44, 46	

El bloque tail, en este ejemplo tiene un tamaño de 2. Desplazamos a la derecha los datos pendientes el tamaño del tail (2), e insertamos los datos de la mezcladora por la parte izquierda y obtenemos

	2	0	5
Datos pendientes de la secuencia 1	28, 29, 31, 32	34, 36, 38, 42	44, 46

Ya tenemos los bloques rellenos y ahora solo hay que apuntarlos en la secuencia de salida

	0	1	2	3	4	5
Secuencia de salida	1	4	3	2	0	5

	0	1	2	3	4	5
Data	34, 36, 38, 42	1, 4, 5, 13	28, 29, 31, 32	22, 23, 25, 27	14, 17, 19, 20	44, 46

Ahora solo queda el movimiento final de los bloques, y ya están los datos ordenados.

2.3.- CASOS ESPECIALES

El algoritmo ha sido diseñado para ser extremadamente eficiente cuando los datos están casi ordenados. Podemos clasificar estos casos especiales en 4 tipos:

- Elementos ordenados y se añaden elementos no ordenados al principio y al final.
- Elementos ordenados, y se insertan elementos en posiciones intermedias, o se hacen modificaciones en datos, alterando la secuencia ordenada.
- Elementos inversamente ordenados
- Combinación de los 3 anteriores.

Se empieza por el inicio buscando elementos ordenados o inversamente ordenados. Si el número de elementos ordenados o inversamente ordenados rebasa un valor umbral que suele ser la cuarta parte del tamaño a ordenar, aplicamos un tratamiento especial, si es menor, se aplica el tratamiento general descrito anteriormente.

Si los elementos están inversamente ordenados, los movemos para dejarlos ordenarlos.
Por ejemplo, tenemos una tabla de 16 elementos con un tamaño de bloque de 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	4	5	7	8	10	13	14	17	21	6	12	9	11	2	16

Rango 1, 10 elementos ordenados Rango 2, 6 elementos no ordenados

Ordenamos el rango 2, y obtendríamos

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	4	5	7	8	10	13	14	17	21	2	6	9	11	12	16

Rango 1, 10 elementos ordenados Rango 2, 6 elementos ordenados

La idea es mezclar los dos rangos, y van a aparecer 2 casos diferentes:
1. Cuando el tamaño del Rango 2 es menor o igual que el doble del tamaño de bloque
2. Cuando el tamaño del Rango 2 es mayor que el doble del tamaño de bloque

2.3.1.- Número de elementos no ordenados menor o igual que el doble del tamaño de bloque

En este caso la idea es usar como memoria auxiliar, la memoria interna de la mezcladora que es el doble del tamaño de bloque.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	4	5	7	8	10	13	14	17	21	2	6	9	11	12	16

Copiamos el Rango 2 en la memoria auxiliar y tenemos

0	*1*	*2*	*3*	*4*	*5*
2	6	9	11	12	16

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
3	4	5	7	8	10	13	14	17	21						

1 posición *2 posiciones* *3 posi-ciones* *5 posiciones* *6 posiciones*

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
2	3	4	5	6	7	8	9	10	11	12	13	14	16	17	21

Con lo que tendríamos los elementos ordenados

2.3.2.- Número de elementos no ordenados mayor que el doble del tamaño de bloque

En el algoritmo tenemos una mezcla de rangos usada en el procedimiento general, pero tiene una limitación. El número de elementos del Rango 1 ha de ser múltiplo del tamaño de bloque. El Rango 2 no tiene esa limitación, ya que puede contener un último bloque incompleto (tail).

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
3	4	5	7	8	10	13	14	17	21	6	12	9	11	2	16

Rango 1, 10 elementos ordenados Rango 2, 6 elementos no ordenados

Si tenemos un tamaño de bloque de 4, el primer bloque contiene 10 elementos, lo recortamos al múltiplo del bloque mas cercano, en este caso 8, por lo que el Rango 1 quedaría de 8 elemento y el Rango 2 de 8 elementos tanbien que hemos de ordenar.

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
3	4	5	7	8	10	13	14	17	21	6	12	9	11	2	16

Nuevo Rango 1 de 8 elementos ordenados *Nuevo Rango 2 de 8 elementos no ordenados*

Ordenamos el Rango 2 y obtendríamos

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
3	4	5	7	8	10	13	14	2	6	9	11	12	16	17	21

Nuevo Rango 1 de 8 elementos ordenados *Nuevo Rango 2 de 8 elementos no ordenados*

Ahora el Rango 1 tiene un tamaño múltiplo del tamaño de bloque y ya podemos hacer la mezclas, como se describe en apartados anteriores.

2.3.3.- Búsqueda de elementos ordenados hacia atrás

De la misma forma que hemos empezado la búsqueda de las primeras posiciones a las últimas, vamos a hacerlo en sentido inverso desde las últimas posiciones hacia las primeras.

Los conceptos son idénticos a los descritos en la búsqueda hacia adelante.
Si el numero de elementos ordenados o inversamente ordenados es mayor del valor umbral, hacemos un tratamiento especial.

Si el número de elementos que faltan por ordenar es menor que el doble del tamaño de bloque, hacemos una inserción, similar a la anteriormente descrita, Y si es mayor, recortamos el ramaño del Rango 2 para que el numero de elementos que nos quedan por ordenar (Rango 1) sea múltiplo del tamaño de bloque, y poder hacer la mezcla como en el procedimiento general.

3.- BENCHMARKS

```
La memoria medida en el test de ordenar 100 000 000 números de 64 bits fué:

std::stable_sort        1176 MB
Boost spin_sort         1175 MB
Boost flat_stable_sort   787 MB

En el test de tiempos, los datos random, sorted y reverse sorted son 100 000 000 números
de 64 bits. A estos se les añaden el 0.1%, 1% y 10% de elementos rellenados de forma
aleatoria al final o en posiciones intermedias uniformemente espaciados.

[ 1 ] std::stable_sort    [ 2 ] Boost spin_sort  [ 3 ] Boost flat_stable_sort
```

	[1]	[2]	[3]
random	8.51	9.45	10.78
sorted	4.86	0.06	0.07
sorted + 0.1% end	4.89	0.41	0.36
sorted + 1% end	4.96	0.55	0.49
sorted + 10% end	5.71	1.31	1.39
sorted + 0.1% middle	6.51	1.85	2.47
sorted + 1% middle	7.03	2.07	3.06
sorted + 10% middle	9.42	3.92	5.46
reverse sorted	5.10	0.13	0.14
reverse sorted + 0.1% end	5.21	0.52	0.41
reverse sorted + 1% end	5.27	0.65	0.55
reverse sorted + 10% end	6.01	1.43	1.46
reverse sorted + 0.1% middle	6.51	1.85	2.46
reverse sorted + 1% middle	7.03	2.07	3.16
reverse sorted + 10% middle	9.42	3.92	5.46