1.- Descripción general hoy vamos a repasar:

• La reducción entre clasificación y colas de prioridad.
• Una encuesta de tipo
• Secuencias bitónicas
• Operación de fusión logarítmica
• Clasificación empacada
• Algoritmo de orden de firma

La reducción y la encuesta servirán como motivación para el tipo de firma. Las secuencias bitónicas se utilizarán para crear una fusión logarítmica que, a su vez, se utilizará para crear la clasificación por paquetes, que finalmente se utilizará para crear la clasificación por firma (Signature Sort).

2.- Clasificación reducida a colas de prioridad

Thorup [7] mostró que si podemos ordenar n enteros de w-bit en O (nS (n, w)), entonces tenemos una prioridad cola que puede admitir la inserción, eliminación y encontrar operaciones mínimas en O (S (n, w)). Para obtener una cola de prioridad de tiempo constante, necesitamos ordenación lineal de tiempo.

ABIERTO: tiempo lineal de clasificación

3.- Clasificación reducida a las colas de prioridad

A continuación se muestra una lista de los resultados que describen el progreso actual en este problema.

• Modelo de comparación: O (n . lg n)

• Ordenación de conteo (Counting Sort) : O (n + u) , u tamaño de la tabla

• Clase de radix (Radix sort): O (n .(w / lg n))

• Van Emde Boas: O (n lg w), mejorado a O (n .lg(w / lg n)) (ver [6]).

• Tipo de firma: lineal cuando w = Ω ((lg n)º(2+ ε)) (ver [2]).

• Han [4]: O (n lg lg n) determinista, ACº RAM.

• Han y Thorup: O (n ((lg lg n)º1/2)) aleatorizado, mejorado a O (n (lg(w/lg n))º1/2) (ver [5] y [6])).

Hoy, nos centraremos completamente en los detalles del tipo de firma. Este algoritmo funciona siempre que w = Ω ((lg n)º(2+ ε))). La clasificación de radix (Radix sort), que ya deberíamos saber, funciona para valores más pequeños de w, es decir, cuando w = O (log n). Para todos los demás valores de w y n, está abierto si podemos clasificar tiempo lineal. Anteriormente cubrimos el árbol de van Emde Boas, que permite la ordenación O (n log log n) siempre que w = (log n)º O (n) . Lo mejor que hemos hecho en el caso general es un algoritmo aleatorio en tiempo O (n (lg(w/lg n))º1/2) por Han, Thorup, Kirkpatrick y Reisch.

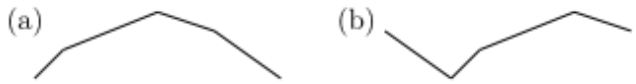
ABIERTO: O (n) clasificación de tiempo ∀ w.

4.- Clasificación para w = Ω ((lg n)º(2+ ε)))

El tipo de firma fue desarrollado en 1998 por Andersson, Hagerup, Nilsson y Raman [2].
Se ordena n enteros w-bit en tiempo O (n) cuando w = Ω ((lg n)º(2+ ε))) para algunos ε > 0. Esto es bastante complicado
ordenar, por lo que vamos a construir el algoritmo desde el principio Primero, proporcionamos un algoritmo para clasificar secuencias bitónicas utilizando métodos de computación paralela. En segundo lugar, mostramos como fusionar dos palabras de k ≤ log n log log n elementos en O (log k) tiempo. En tercer lugar, utilizando este algoritmo de fusión, creamos
una variante de MERGE SORT llamada ordenamiento empaquetado, que ordena n enteros de b-bit en el tiempo O (n) cuando
w ≥ 2 (b + 1) log n log log n. En cuarto lugar, usamos nuestro algoritmo de clasificación empaquetado para construir una clasificación de firma(signature sort).

4.1 Secuencias bitónicas

Una secuencia bitónica es un cambio cíclico de una secuencia que aumenta monotónicamente seguida de una secuencia que disminuye monótonamente. Cuando se examina cíclicamente, tiene solo un mínimo local y un local máximo.



Los cambios cíclicos preservan la binoticidad de una secuencia.

Para ordenar una secuencia bitónica btcseq, ejecutamos el algoritmo que se muestra a continuación en la figura siguiente. Asumir n = len (btcseq) es par.
La ordenación de secuencias bitónicas mantiene dos invariantes: (a) la secuencia btcseq contiene múltiples secciones de secuencias bitónicas (cada nivel de btcsort divide una secuencia bitónica en dos secuencias bitónicas), y (b) cuando se consideran dos secciones de secuencias bitónicas, un elemento en la sección izquierda siempre es más pequeño que un elemento en la sección derecha.

Algorithm 1 Bitonic sequence sorting algorithm

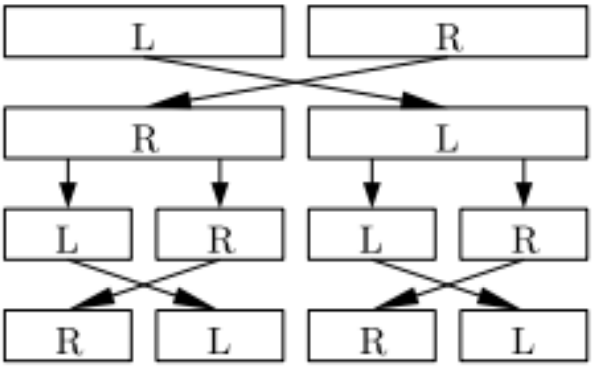
```
btcsort(btcseq):
  for i from 0 to n/2-1:
    if btcseq[i]>btcseq[i+n/2]:
      swap(btcseq[i], btcseq[i+n/2])
  btcsort(btcseq[0:n/2-1])
  btcsort(btcseq[n/2:n-1])
```

Luego de las rondas de O (log n), btcseq se divide en n secciones. Dado que la sección izquierda tiene elementos más pequeños que la sección de la derecha, la clasificación está completa.

Para obtener más información sobre la clasificación de secuencias bitónicas, incluida una prueba de la exactitud de este algoritmo, ver [3, Sección 27.3].

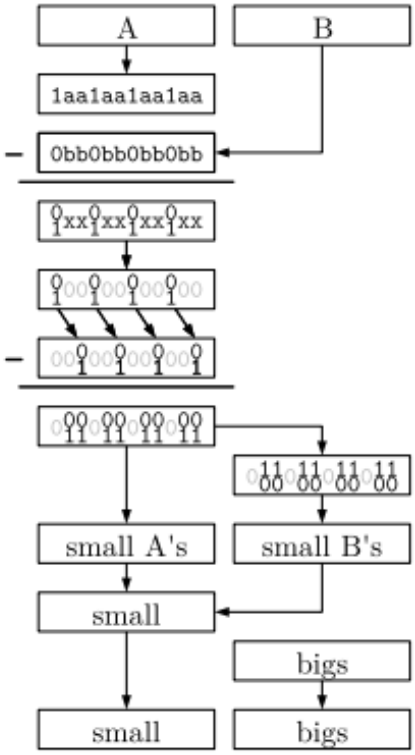
4.2 Operación de fusión logarítmica

El siguiente paso es combinar dos palabras ordenadas, cada una con k elementos de b-bits. En primer lugar, concatenamos la primera palabra con el reverso de la segunda palabra, obteniendo una secuencia bitónica. Para revertir eficientemente una palabra, enmascaramos a la izquierda k/2 elementos y desplazarlos hacia la derecha por (k/2).b ,
luego enmascara los elementos k / 2 que se encuentran más a la derecha y muévalos hacia la izquierda en (k / 2).b .Tomando el OR de las dos palabras resultantes nos deja con la palabra original con las mitades izquierda y derecha intercambiadas. Ahora podemos repetir en las mitades izquierda y derecha de la palabra, dándonos la recursión
T (n) = T (k/2) + O (1), por lo que todo el algoritmo toma T (k) = O (log k) tiempo. Las dos palabras ahora pueden ser concatenadas desplazando la primera palabra dejada por kb y tomando su OR con la segunda palabra. La clave aquí es realizar cada nivel de la recursión en paralelo, para que cada nivel tome la misma cantidad de tiempo. Esta reversión de la lista se ilustra en el la siguiente figura.



Los dos primeros pasos en la recursión para revertir una lista.

Todo lo que queda es ejecutar el algoritmo de clasificación bitónico en los elementos de nuestra nueva palabra. Para hacerlo, debemos dividir los elementos en dos mitades e intercambiar los pares correspondientes de elementos que están fuera de orden. Luego podemos repetir en la primera y la segunda mitad en paralelo, una vez más dándonos la recursión T (k) = 2T (k/2) + O (1) => T (k) = O (log k) tiempo. Por lo tanto necesitamos una operación de tiempo constante que realice el intercambio deseado. Supongamos que tenemos un bit 0 adicional antes de cada elemento empaquetado en la palabra. Utilizamos este bit de repuesto para ayudar a enmascarar nuestra palabra. Enmascararemos la mitad izquierda de los elementos y estableceremos este bit adicional en 1 para cada elemento, luego enmascare la mitad derecha de los elementos y muévalos hacia la izquierda con (k/2).b . Después restamos la segunda palabra de la primera, un 1 aparecerá en el bit extra si el elemento en el la posición correspondiente de la mitad izquierda es mayor que el elemento en la mitad derecha. Así podemos enmascarar los bits extra, desplace la palabra a la derecha por b - 1 bits, y réstela de sí misma, lo que da como resultado una palabra que enmascara todos los elementos de la mitad derecha que pertenecen a la mitad izquierda y viceversa. De esto, usemos cambios de bit, OR y negación para obtener nuestra lista ordenada. Vea el diagrama de abajo para aclaración; el proceso es bastante sencillo.



Operación de intercambio paralelo en la clasificación bitónica.

Usando estos trucos de bits, terminamos con nuestra operación deseada de intercambio de tiempo constante. Esto lleva al siguiente teorema:

Teorema 1. Suponga que dos palabras contienen k elementos de b-bits. Entonces podemos fusionar estas palabras en O (log k) tiempo.

Prueba. Como ya se señaló, podemos concatenar la primera palabra con el reverso de la segunda palabra en tiempo O (log k). Luego podemos aplicar el algoritmo de clasificación bitónica a estas palabras concatenadas. Dado que este algoritmo tiene una profundidad de recursión de O (log k), y podemos realizar los swaps necesarios en tiempo constante, nuestra operación de fusión termina con complejidad de tiempo O (log k).

4.3 Clasificación empacada (Packed Sorting)

En esta sección presentaremos la clasificación empaquetada [1], que ordena n enteros de b-bits en tiempo O (n) para un tamaño de palabra de w ≥ 2 (b + 1) log n log log n. Este límite para w nos permite empaquetar $k = \log n \log \log n$ elementos en una palabra, dejando un bit cero delante de cada entero, y w/2 bits cero al principio de

la palabra.

Estructura para empaquetar enteros de b-bit en una palabra de w-bit.

Para ordenar estos elementos utilizando la clasificación empaquetada, construimos las operaciones de fusión de la siguiente manera:

1. Combine dos palabras ordenadas en tiempo O (lg k), como se muestra con la ordenación bitónica en la sección anterior.

2. Combine la ordenación en k elementos con (1) como la operación de fusión(merge). Este tipo tiene la recursión $T (k) = 2T (k /2) + O (\lg k) \Rightarrow T (k) = O (k)$.

3. Combine dos listas ordenadas de r palabras ordenadas en una lista ordenada de 2r palabras ordenadas. Esto es hecho de la misma manera que en una ordenación de fusión estándar, excepto que usamos (1) para acelerar el operación. Comenzamos por fusionar las palabras más a la izquierda en las dos listas usando (1). La primera palabra después de esta fusión debe constar de los k elementos más pequeños en general, por lo que generamos esta palabra. Sin embargo, no podemos estar seguros de la posición relativa de los elementos en la segunda palabra. Por lo tanto, examinamos el elemento máximo en la

segunda palabra, y agregamos la segunda palabra al comienzo de la lista que contenía este elemento máximo. Seguimos fusionando las listas de esta manera, dando como resultado O (r . log k) en general.

4. Combine y clasifique todas las palabras con (3) como la operación de fusión y (2) como el caso base.

Teorema 2. Deje que el tamaño de la palabra $w \geq 2 (b + 1) \log n \log \log n$. Luego, la ordenación empaquetada clasifica n enteros de b bits en tiempo O (n).

Prueba. La clasificación definida en el Paso 4 del algoritmo anterior tiene la recursión $T (n) = 2 T (n/ 2) + O ((n/k) \log k)$ y el caso base $T (k) = O (k)$. La profundidad de recursión es $O (\log n/k)$, y cada nivel toma $O ((n/k) \lg k) = O (n/ \lg n)$ tiempo, por lo que en conjunto contribuyen con un costo de O (n). También hay n/k casos de base, cada caso toma O (k) tiempo, dando un tiempo de ejecución total de O(n), según se desee.

4.4 Algoritmo de orden de firma (Signature Sort)

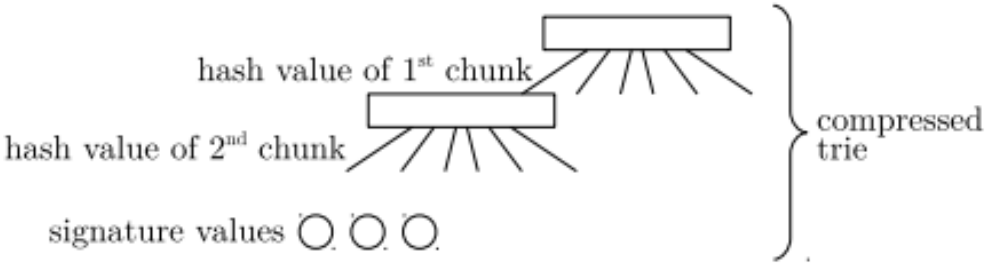
Utilizamos nuestro algoritmo de clasificación empaquetado para construir nuestra clasificación de firma en siete pasos. Asumimos que $w \geq (\log n)^{\varepsilon}(2+ \varepsilon) \log \log n$. La ordenación de la firma ordenará n enteros de w-bit en tiempo O (n). Comenzaremos dividiendo cada número entero en $(\lg n)^{\varepsilon}$ trozos de igual tamaño, que luego serán reemplazados por O (lg n)-bits de las firmas mediante hashing. Estas firmas más pequeñas se pueden clasificar con ordenamiento empaquetado en tiempo lineal. Debido a que el hashing no conserva el orden, construiremos un conjunto comprimido de estas firmas ordenadas, que luego ordenaremos de acuerdo con los valores de los fragmentos originales. Esto nos permitirá recuperar el orden ordenado de los enteros. La clasificación de la firma procederá de la siguiente manera:

1. Divida cada entero en $(\lg n)^{\varepsilon}$ trozos de igual tamaño. (Note la distinción de un árbol de fusión, que tiene trozos de tamaño $(\lg n)^{\varepsilon}$)

2. Reemplace cada fragmento por un hash O(lg n) -bit (el hashing perfecto está bien). Al hacer esto, terminamos con n . O ((lg n)º(1+ ε))-bits de firmas. Una forma en que podemos hacer hash es multiplicar por algún valor al azar x, y luego enmascarar las claves hash. Esto nos permitirá hacer hash en tiempo lineal. Ahora, nuestro hash no conserva el orden, pero lo importante es que preserva la identidad.

3. Ordene las firmas en tiempo lineal con clasificación empaquetada, como se muestra arriba.

4. Ahora queremos rescatar las identidades de las firmas. Construir un trío comprimido sobre las firmas, de modo que un recorrido ordenado del trío nos dé nuestras firmas en orden. El trío comprimido solo usa espacio O (n).



Un trío para el almacenamiento de firmas. Edge representaba valores hash de trozos; hojas representadas posibles valores de firma

Para hacer esto en tiempo lineal, agregamos las firmas en orden de izquierda a derecha. Ya que estamos en la palabra RAM, podemos calcular el LCP con la firma (i - 1)ºst tomando la más significativa 1 bit del XOR. Luego, subimos por el árbol hasta el nodo apropiado, y cargamos la caminata hasta la disminución en la longitud del camino más a la derecha del camino. La creación de la nueva rama es un tiempo constante, por lo que obtenemos un tiempo lineal total. Este proceso es similar a la creación de un árbol cartesiano.

5. Ordene recursivamente los bordes de cada nodo en el trie en función de sus valores reales. Esta es una recursión en (ID de nodo, fragmento real, índice de borde), que ocupa (O (log n), O (w/(log n)º ε), O (log n)) espacio . Los índices de borde están allí para realizar un seguimiento de la permutación. Después de una constante (1/ ε) + 1 niveles de recursión, tendremos b = O (log n + (w/(log n)º (1+ε))) = O (w/(log n log log n)), por lo que podemos usar la clasificación empaquetada como el caso base de la recursión.

6. Permuta los bordes del hijo.

7. Haga un recorrido del orden para obtener la lista ordenada deseada de las hojas.

Juntando estos pasos, obtenemos:

Teorema 3. Deje w ≥ (log n)º(2+ ε) log log n. Luego, la ordenación de firmas ordenará n enteros de w-bit en tiempo O (n).

Prueba. Romper los números enteros en trozos y hacerlos hash (Pasos 1 y 2) toma tiempo lineal. Clasificación estos trozos de hash que usan ordenación empaquetada (Paso 3) también toman tiempo lineal. La creación del conjunto comprimido sobre las firmas toma tiempo lineal (Paso 4), y la clasificación de los bordes de cada nodo (Paso 5) toma tiempo constante por nodo para un número lineal de nodos. Finalmente, escaneando y permutando los nodos (Paso 6) y el recorrido en orden de las hojas del trie (Paso 7) tomarán un tiempo lineal, completando la prueba.