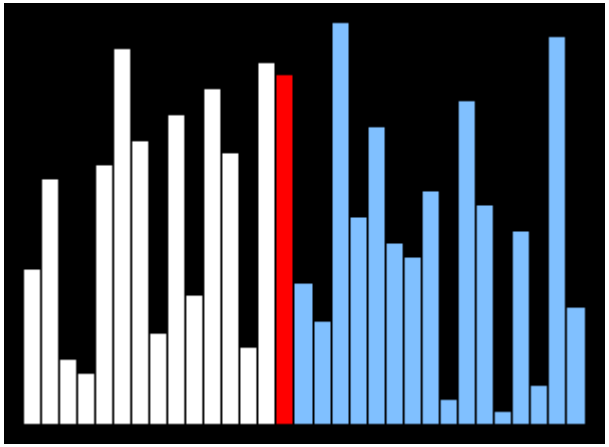
Algoritmos y Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán

2024

UNIVERSIDAD NACIONAL DE TUCUMÁN
facet
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍA

Ordenación(4)



METODOS LINEALES

Todos los métodos de ordenación presentados sirven para clasificar ítems con cualquier tipo de campo clave, siempre que en el mismo esté definido una relación de orden. Los algoritmos se definen en términos de operaciones básicas de comparaciones y movimientos.

Se ha demostrado que son necesarios $O(n\log_2 n)$ pasos para ordenar n elementos siempre que sea un método de propósito general para cualquier tipo de claves.

Los métodos con costo lineal no son aplicables a cualquier caso sino a un rango de problemas bien definidos. Sirven para un tipo particular de clave tal que tengan condiciones especiales como: ser números enteros y pertenecer a un cierto rango.

Los métodos de ordenación que aprovechan estas propiedades de las claves numéricas se llaman cuenta de distribuciones y ordenación por residuos.

METODOS LINEALES

Un problema de ordenación en un caso trivial es si se conocen las siguientes 2 condiciones en el arreglo que hay que ordenar:

1) Los valores de las claves son únicamente 1 o 2.

2) Los ítems a ordenar están formados solo por la clave y no tienen información asociada a la misma.

- Se podrá ordenar este arreglo con costo lineal?
- Como será el algoritmo?

Método Simple

Algoritmo Simple(A,n)
Entrada: A: arreglo de (1.. MAX) elementos enteros con valor: 1 o 2
n: nro. de datos a ordenar
Salida: A(1..n): arreglo ordenado
Auxiliares: i,cont1 $\in$ entero

P1. cont1 ← 0 → ϵ O(1)
P2. PARA i=1,n HACER → nit =n
SI A(i) = 1 ENTONCES → ϵ O(1)
cont1 ← cont1+1 → ϵ O(1)
ϵ O(n)
P3. PARA i=1,cont1 HACER → nit =n
A(i) ← 1 → ϵ O(1)
ϵ O(n)
P4. PARA i= cont1+1,n HACER → nit =n
A(i) ← 2 → ϵ O(1)
ϵ O(n)

T(n) ϵ O(n)

FIN

Método Simple

El tiempo de ejecución del método Simple crece linealmente con **n**:

$$T(n) \in O(n)$$

Donde: n: numero de datos a ordenar

- Muy importante destacar que el algoritmo Simple si bien compara el valor de cada entrada del arreglo con el valor 1, no compara los elementos del arreglo entre si como lo hacen todos los métodos generales de ordenación.
- De esta manera se está ordenando sin comparar de a dos las claves almacenadas en el arreglo.
- Se podrá generalizar al caso en que las claves tengan otros valores enteros, por ejemplo de 1 a K ?

Método cuenta de distribuciones

Un problema de ordenación también muy especial consiste en ordenar **n** ítems cuyas claves son números enteros en un rango entre **1** y **k**. Si k no es demasiado grande, se puede utilizar para resolver este problema un algoritmo denominado **cuenta de distribuciones**.

La idea es contar el número de claves de cada valor entre 1 y k en el arreglo dado A(1..n). La cuenta se lleva en un arreglo auxiliar **Cont(1..k)** de elementos enteros

Después se utilizan los números que se han contado en B para situar los ítems en su posición durante un segundo recorrido en el que se vuelve a armar el arreglo A en orden.

Una de las versiones se muestra en el Algoritmo Casillero.

Método Casillero

Algoritmo Casillero(A,n,k)
Entrada: A: arreglo de (1.. MAX) elementos ***enteros*** en el rango 1..k
n: nro. de datos a ordenar
k: los enteros a ordenar están en el intervalo [1..k]
Salida: A(1..n): arreglo ordenado
Auxiliares: Cont(1..k): arreglo de numeros enteros; i,j $\in$ entero

P1. PARA j=1,k HACER
Cont(j) ←0
P2. PARA i=1,n HACER
Cont(A(i)) ← Cont(A(i))+1
P3. i ← 0
P4. PARA j=1,k HACER
MIENTRAS Cont(j)≠0 HACER
i ← i+1
A(i) ← j
Cont(j) ← Cont(j)-1
FIN

Método Casillero

Ejemplo Algoritmo Casillero

Entrada:
A=(3, 4, 5, 4, 3, 2, 5, 3, 3, 5)
n= 10
k=5, las claves a ordenar son enteros que están en el intervalo [1..k]

Solución:
P1. → Cont=(0,0,0,0,0)
P2. → Cont=(0,1, 4, 2, 3)
P4. → A=(2,3,3,3,3,4,4,5,5,5)

Salida:
A=(2,3,3,3,3,4,4,5,5,5) **ordenado**

```
P1. PARA j=1,k HACER
        Cont(j) ←0
P2. PARA i=1,n HACER
      Cont(A(i)) ← Cont(A(i))+1
P3. i ← 0
P4. PARA j=1,k HACER
      MIENTRAS Cont(j)≠0 HACER
          i ← i+1
          A(i) ← j
          Cont(j) ← Cont(j)-1
```

Costo algoritmo Casillero

Algoritmo Casillero(A,n,k)

P1. PARA j=1,k HACER → nit = k

Cont(j) ←0 → ϵ O(1)

} ϵ O(k)

P2. PARA i=1,n HACER → nit = n

Cont(A(i)) ← Cont(A(i))+1 → ϵ O(1)

} ϵ O(n)

P3. i ← 0 → ϵ O(1)

P4. PARA j=1,k HACER

MIENTRAS Cont(j)≠0 HACER } nit =n

i ← i+1 → ϵ O(1)

A(i) ← j → ϵ O(1)

Cont(j) ← Cont(j) -1 → ϵ O(1)

} ϵ O(n)

FIN

T(n,k) ϵ O(k+n)

Método Casillero

El tiempo de ejecución del metodo Casillero crece con **k** y con **n**:

$$T(n,k) \in O(k+n)$$

Donde:

n: numero de datos a ordenar
k: los enteros a ordenar están en el intervalo [1..k]

Análisis del tiempo de ejecución:

Si $k < n$ → $T \in O(n)$

Si $k = n, 2n, 3n$ → $T \in O(n)$

Si $k = n^2$ → $T \in O(n^2)$

ORDENACIÓN EXTERNA

Muchas importantes aplicaciones de clasificación deben procesar archivos muy grandes, demasiado como para tenerlos en memoria principal de cualquier computadora. Los métodos adaptados a estas aplicaciones se denominan ***métodos externos***, pues hacen el procesamiento usando dispositivos externos para realizar la clasificación.

Hay dos factores determinantes que hacen que los algoritmos externos sean diferentes de los que se han visto hasta ahora.

- El primero es que el *costo de acceso* a un elemento es infinitamente más grande que el de cualquier actualización o cálculo.
- El segundo, dependiendo del medio de almacenamiento externo utilizado, es que en algunos casos existen *restricciones de acceso* que permiten tener acceso solamente de forma secuencial.

ORDENACIÓN EXTERNA

La gran variedad de dispositivos de almacenamiento externo y de costos hace que el desarrollo de los métodos de ordenación externos sean muy dependientes de la tecnología actual.

En los métodos de ordenación externos el costo principal se debe a la entrada y salida de los datos, por lo tanto se trata de limitar el número de veces que una parte de los datos se mueve del almacenamiento externo a memoria principal, y asegurarse que las transferencias se hagan tan eficientemente como lo permita el hardware disponible.

METODO de MEZCLA O FUSION

La mayoría de los métodos de ordenación externa utilizan la siguiente estrategia general:

- Primero hacen una pasada a lo largo del archivo a ordenar, dividiendo a éste en bloques del tamaño de la memoria interna.
- Se guardan y se ordenan en memoria interna cada uno de estos bloques que vuelven a memoria externa.
- Luego fusionan entre sí los bloques ordenados haciendo varias pasadas a través del archivo, creando sucesivamente archivos ordenados más grandes hasta que el archivo completo esté ordenado.
- El acceso a los datos es siempre secuencial.

METODO de MEZCLA O FUSION

El método se basa en la mezcla o fusión o intercalación que es la combinación de dos o mas secuencias ordenadas en una única secuencia ordenada, obtenida por la selección por comparación entre los componentes accesibles en cada momento.

La mezcla es una operación mucho más simple que la ordenación y se usa como auxiliar en procesos más complejos de ordenación secuencial.

METODO de MEZCLA O FUSION

Puesto que la mayor parte del costo de un método de ordenación externa se debe a la entrada/salida, es posible tener una medida aproximada del costo de una ordenación contando el número de veces que se lee o escribe cada registro (o dato) del archivo.

Los métodos más usados son:

- Mezcla natural o simple
- Mezcla balanceada
- Mezcla polifase
- Mezcla de cascada

METODO de Mezcla Natural

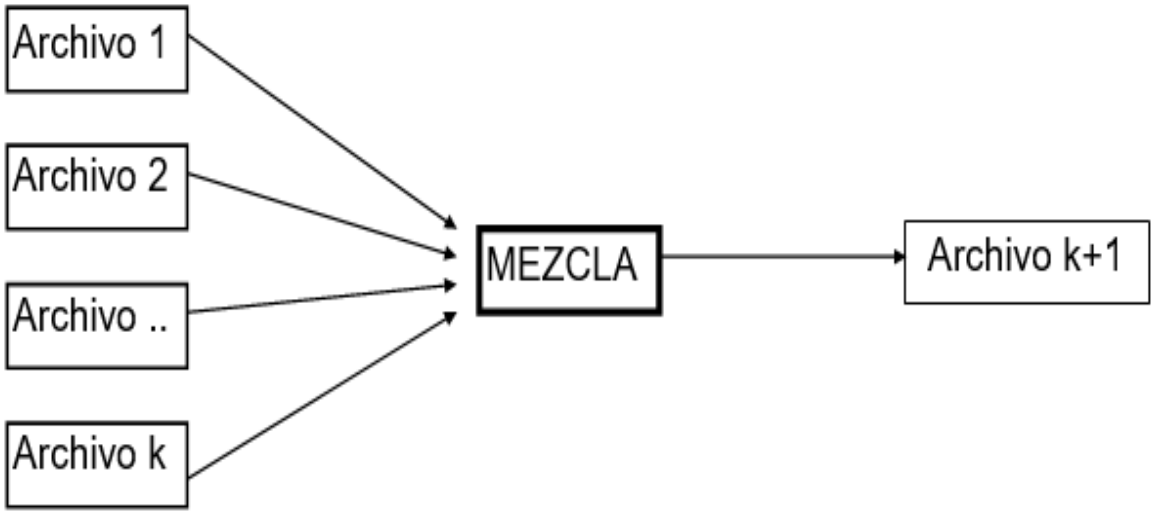
Mezcla natural 2 vías:

- Una mezcla que toma dos archivos de entrada se llama mezcla de ***doble vía***. Usa 3 archivos.

METODO de Mezcla Natural

Mezcla natural k vías:

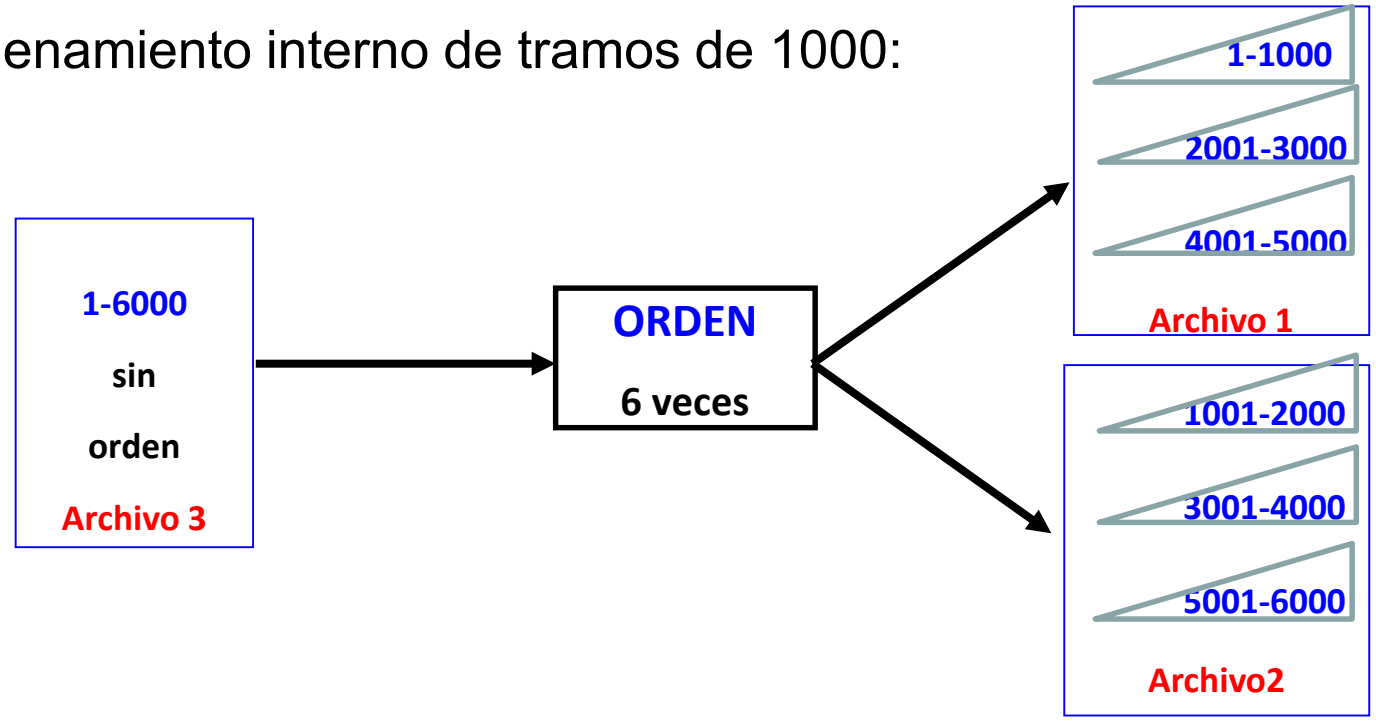
- Una mezcla que recibe ***k archivos de entrada*** a la vez y uno solo de salida se llama ***mezcla de k vías*** (k-way). Usa k+1 archivos.



Mezcla natural de 2 vías

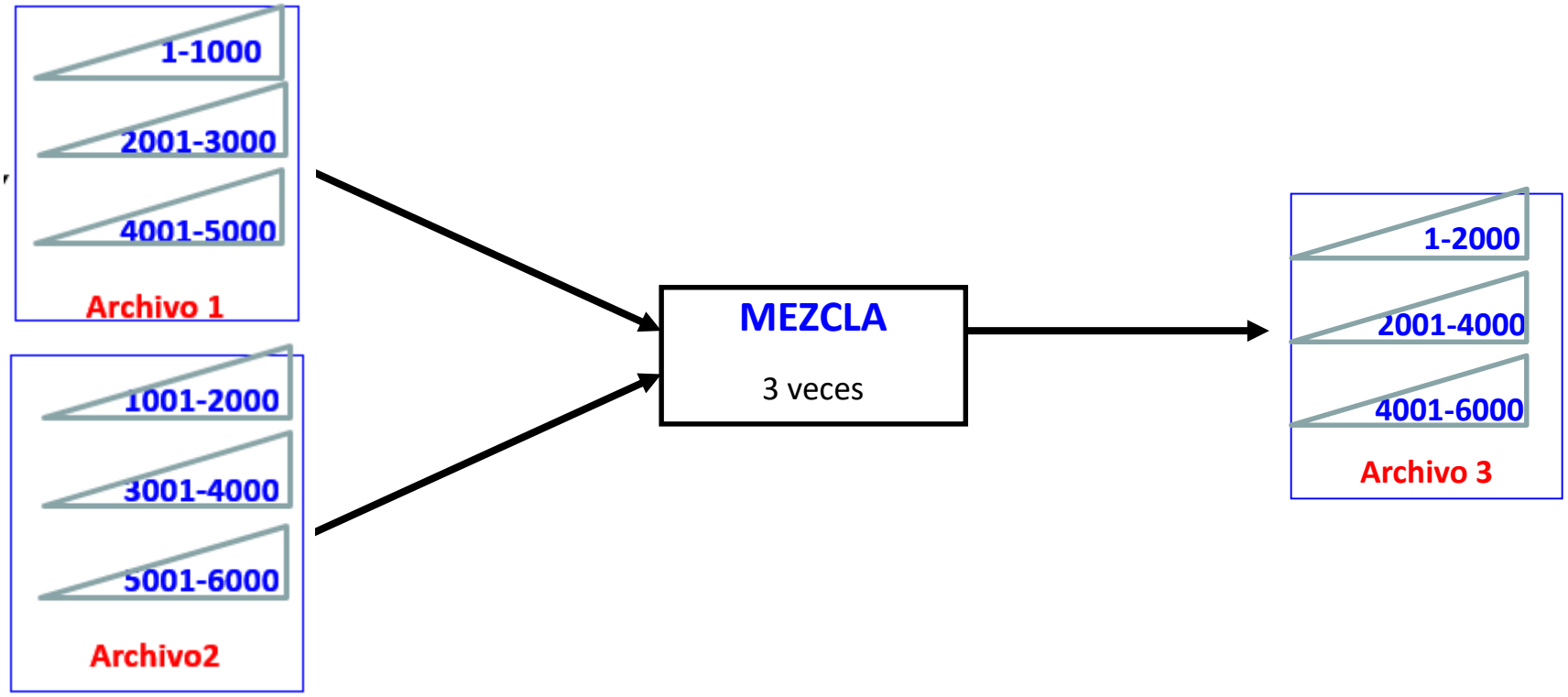
Ejemplo: Ordenar 6000 registros, de los cuales 1000 caben en memoria usando 2 vías.

1) Ordenamiento interno de tramos de 1000:



Mezcla natural de 2 vías

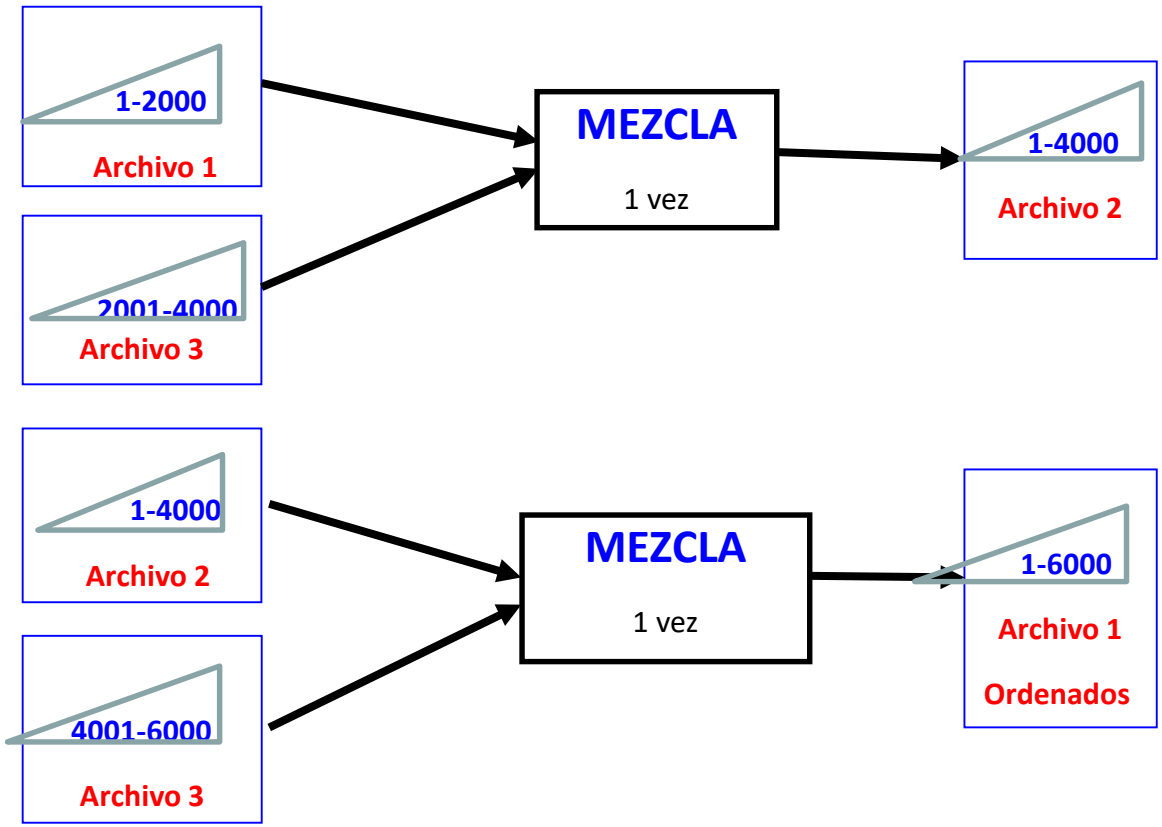
2) Se aplica Mezcla de 2 vías tres veces:



Mezcla natural de 2 vías

3) Copiar el tramo ordenado 1-2000 del archivo 3 al 1.

4) Se aplica Mezcla de 2 vías dos veces:

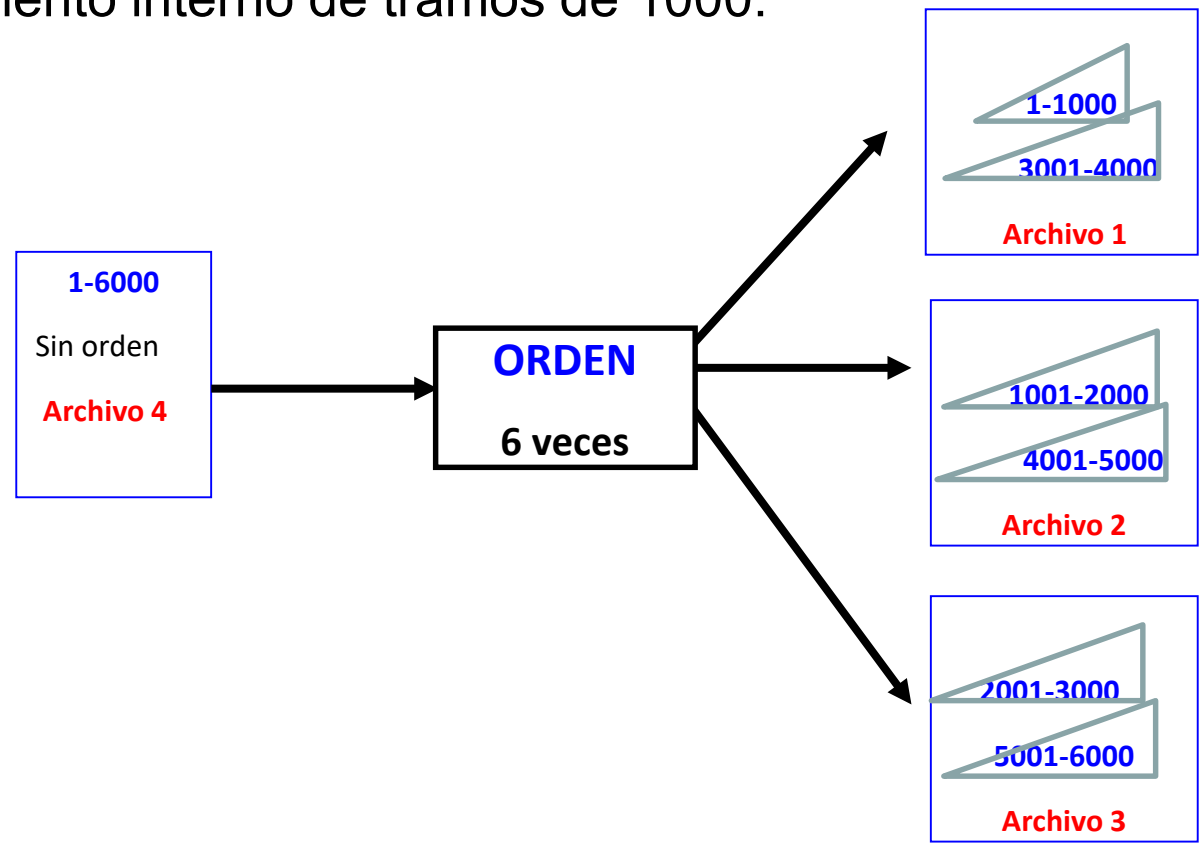


Se lee/escribe
5 veces el reg1

Mezcla natural de 3 vías

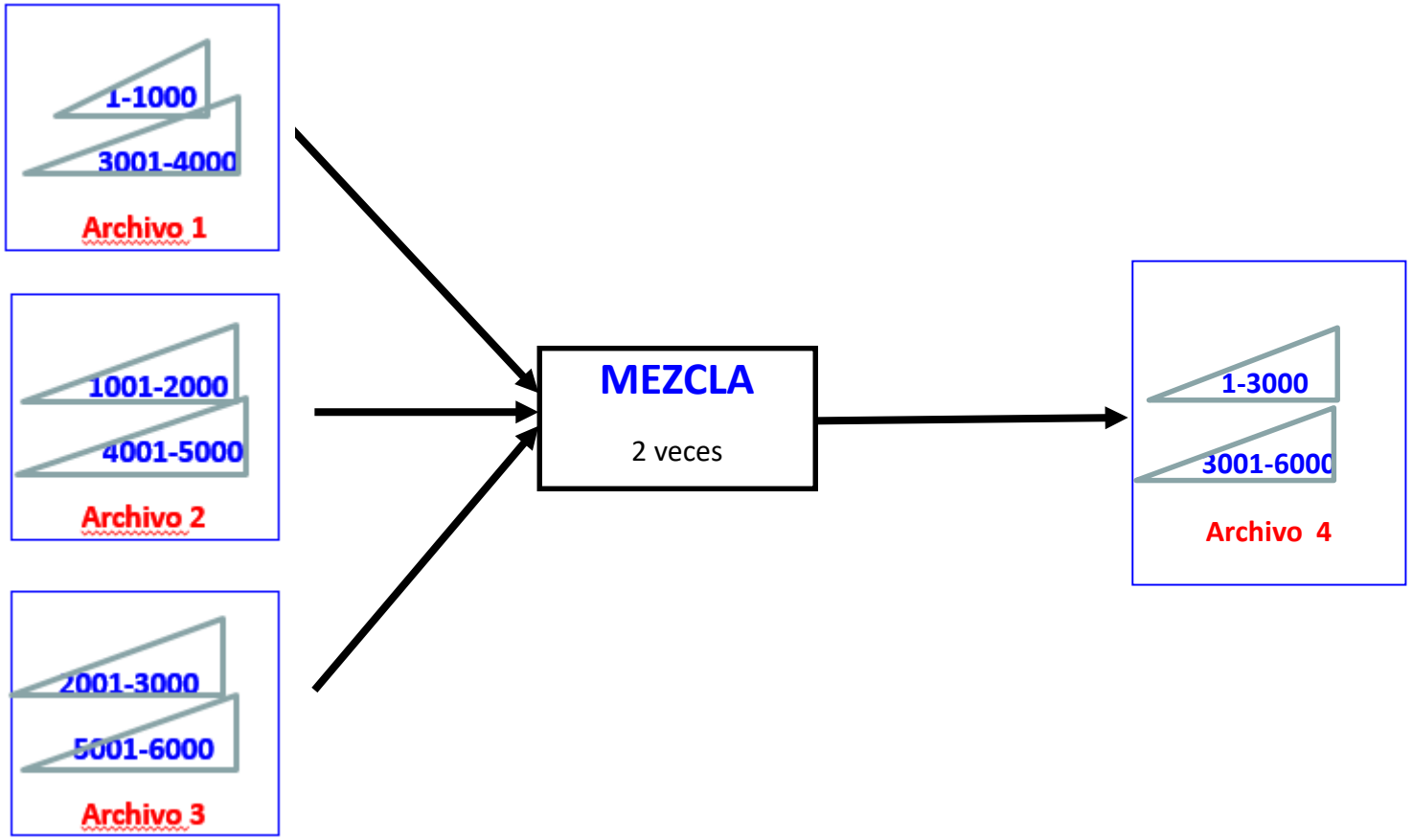
Ejemplo: Ordenar 6000 registros, de los cuales 1000 caben en memoria usando 3 vías.

1) Ordenamiento interno de tramos de 1000:



Mezcla natural de 3 vías

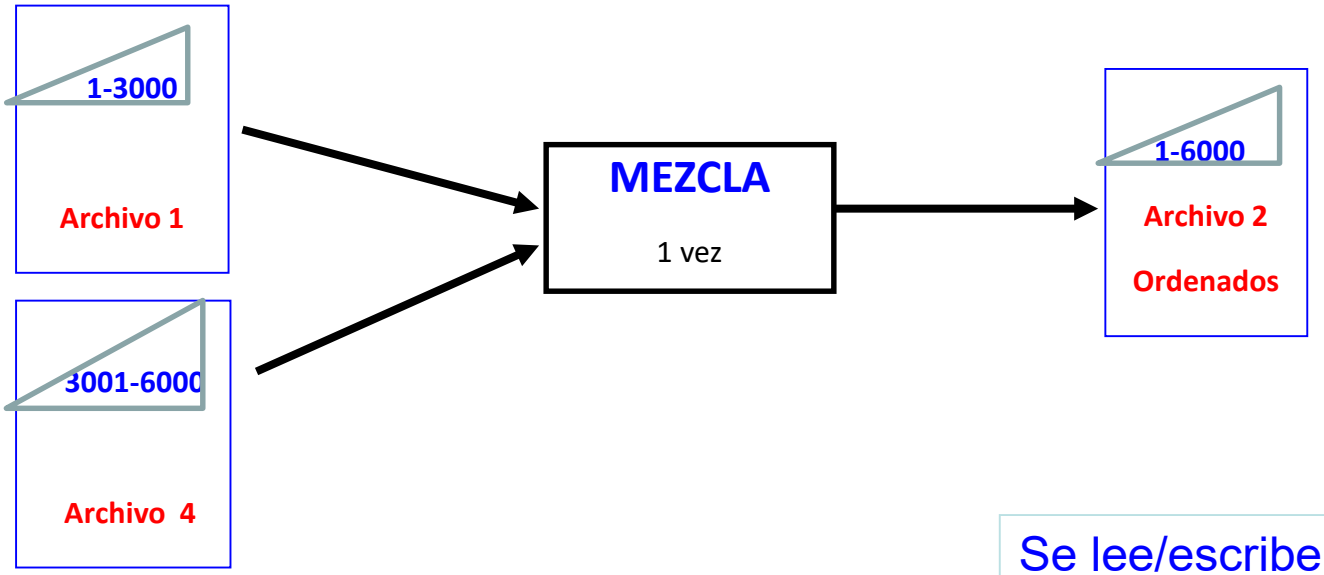
2) Se aplica Mezcla de 3 vías dos veces:



Mezcla natural de 3 vías

3) Copiar el tramo ordenado 1-3000 del archivo 4 al 1.

4) Se aplica Mezcla de 2 vías una vez:



Se lee/escribe
4 veces el reg1

Mezcla natural vs balanceada

Cerca de la mitad de las operaciones de Entrada/Salida en la mezcla natural son para copiar en archivos las sublistas intercaladas que se producen en un paso para usarlas en el siguiente.

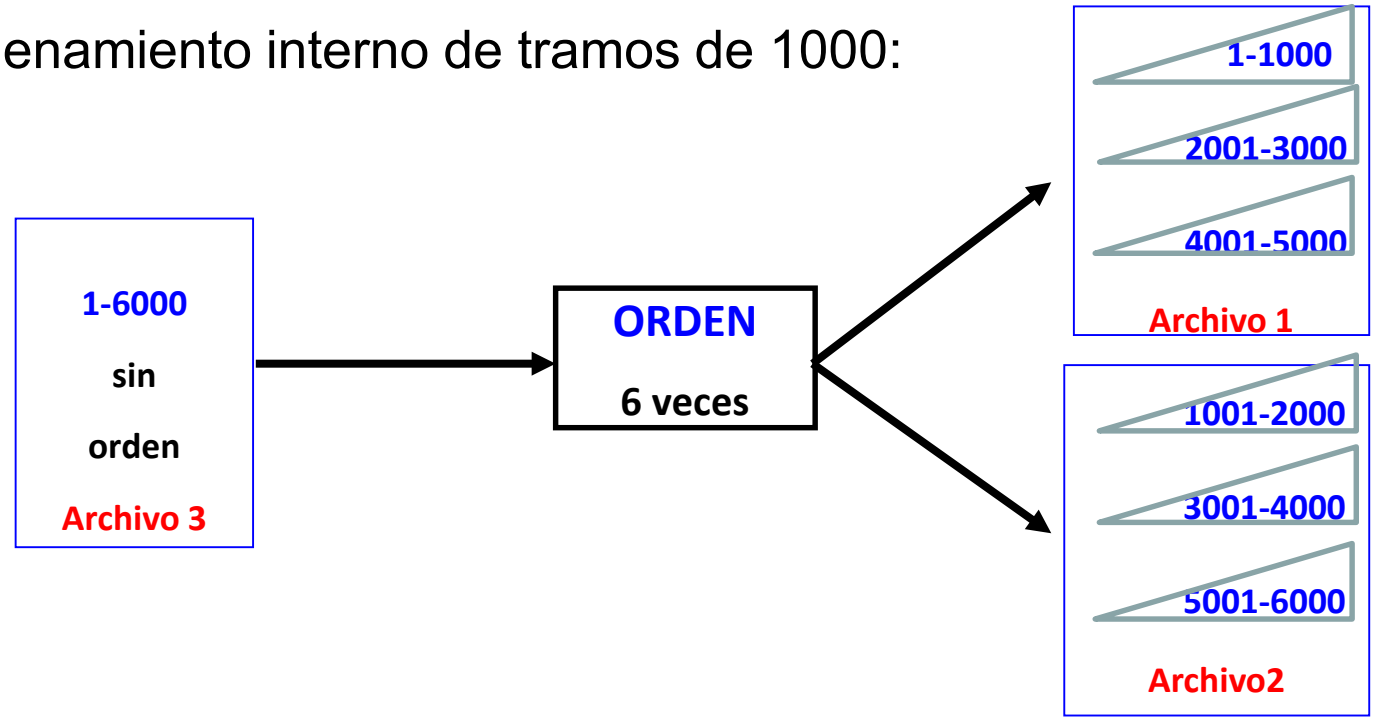
Los requerimientos de Entrada/Salida de una mezcla natural pueden reducirse usando mezcla balanceada. Esta evita el copiado de registros distribuyendo los resultados de la mezcla directamente en el número apropiado de archivos para la entrada al siguiente paso de mezcla, en contraste con la mezcla natural que pone el resultado de la mezcla en un solo archivo.

Una mezcla natural de **m** vías usa **m+1** archivos, mientras que una mezcla balanceada de **m** vías usa **2 m** archivos.

Mezcla balanceada de 2 vías

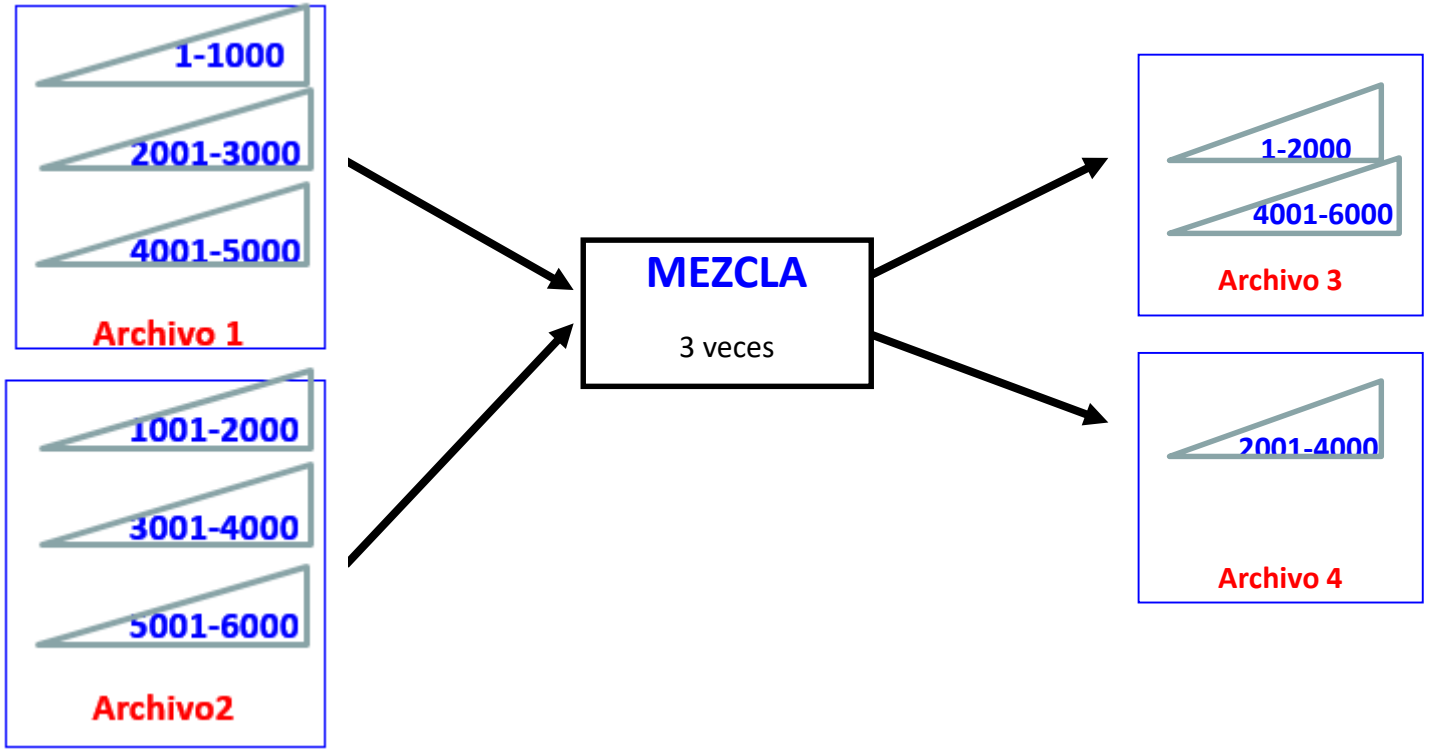
Ejemplo: Ordenar 6000 registros, de los cuales 1000 caben en memoria usando 2 vías.

1) Ordenamiento interno de tramos de 1000:



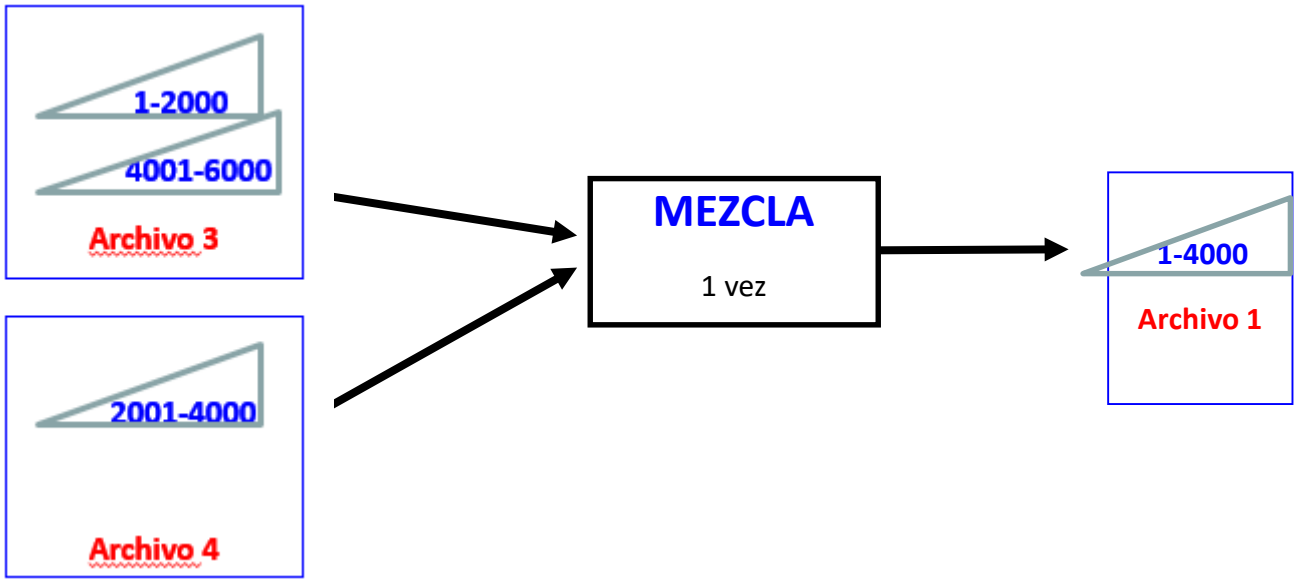
Mezcla balanceada de 2 vías

2) Aplicar Mezcla de 2 vías balanceada tres veces.



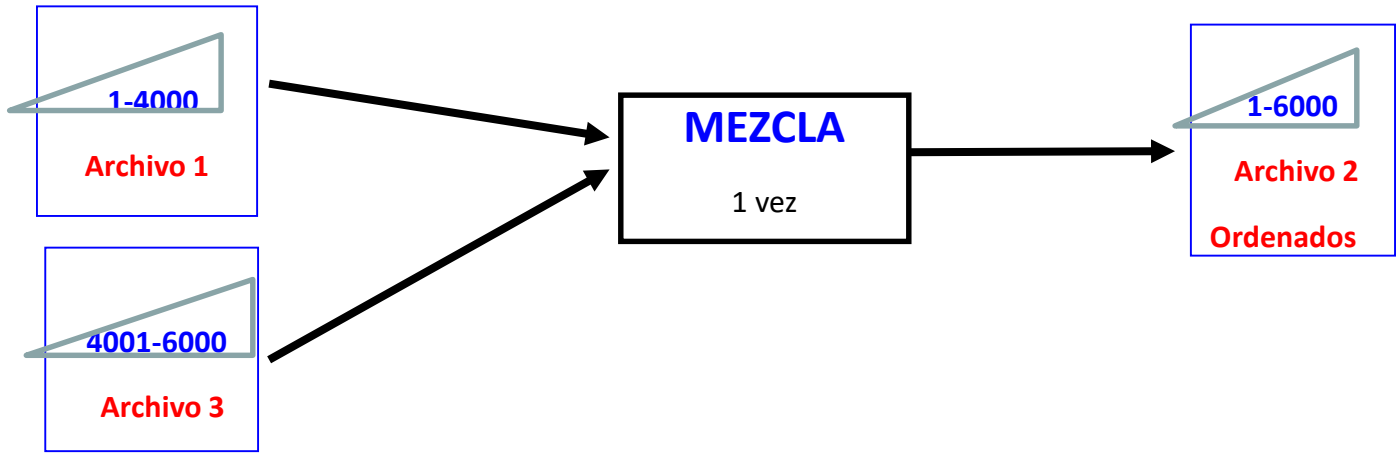
Mezcla balanceada de 2 vías

3) Aplicar Mezcla natural de 2 vías una vez:



Mezcla balanceada de 2 vías

4) Aplicar Mezcla natural de 2 vías una vez:



Se lee/escribe
4 veces el reg1

Mezcla balanceada de 2 vías

Análisis de la Mezcla balanceada de dos vías:

Si **n** el número total de registros a ordenar, y **m** el número de registros que entran en memoria.

Este algoritmo requiere $\mathbf{\log_2(n/m)}$ pasadas, más la pasada inicial, que es la pasada que construye las secuencias ordenadas.

Cada pasada es O(n), de modo que el método es:

$$\mathbf{O(n \log_2(n/m))}$$

Mezcla balanceada múltiple

Mezcla balanceada múltiple:

Si se dispone de varios archivos se puede reducir el número de pasadas para ordenar la entrada.

Si se tiene **2k** archivos la estrategia de mezcla es la misma que para 4 archivos.

El algoritmo se complica porque en el paso de mezcla hay que encontrar el mínimo entre k elementos, en lugar de entre 2.

Se lo puede hacer usando una ***cola de prioridad***. Para obtener el elemento para escribir en el archivo de salida se hace una operación que borre el mínimo de la cola de prioridad. Se avanza en el archivo apropiado y se inserta ese nuevo elemento en la cola de prioridad.

Mezcla múltiple

Analisis de la Mezcla múltiple

Después del primer paso de ordenación el número de pasadas de la clasificación de mezcal de **k vías** es :

$$\lceil \log_k(n/m) \rceil + 1$$

Cada pasada es O(n) de modo que el tiempo total es:

$$O(n \log_k(n/m))$$

MEMORIA VIRTUAL

Actualmente se dispone de la posibilidad de manejo de ***memoria virtual*** de gran capacidad que puede servir para ordenar archivo muy grandes. En este caso el programador puede direccionar mucha memoria dejando al sistema que maneje la memoria virtual, y será el encargado de transferir de memoria interna a externa cuando sea necesario.

Muchos métodos de ordenación presentan una localización cercana de referencias, esto implica pocas transferencias de memoria externa a la interna. Estos algoritmos serán los mas adecuados cuando se use memoria virtual.