webir

Clase 4

webir - Temas

- Repaso: recuperación tolerante a errores de ortografía y otras inconsistencias

- Algoritmo Soundex
- Construcción de índices
  - cuando no se puede hacer en memoria
  - algoritmos

webir - Búsquedas con ”Comodines”

- mon* - árbol de búsqueda para las palabras del vocabulario
- *mon - árbol de búsqueda invertido
- se*mon - mediante ambos árboles

webir - Correcciones Ortográficas

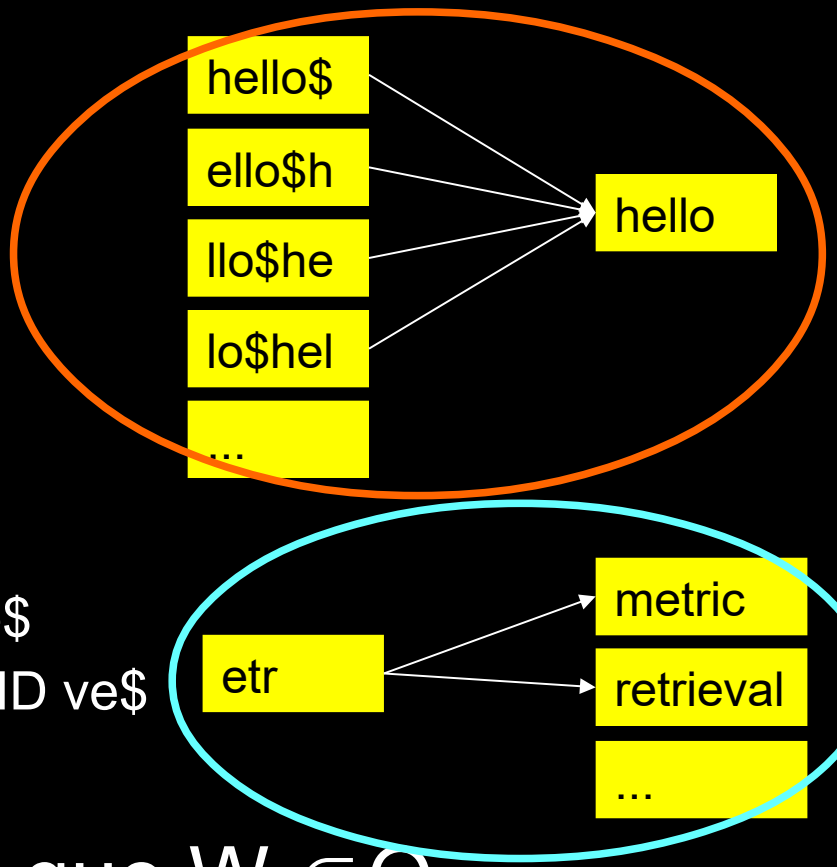
- Términos aislados
- Corrección Sensible al Contexto
  - Corregir las palabras de la consulta en forma individual
  - Buscar como conceptos o frases con y sin las correcciones
  - Usar frecuencias para acotar las búsquedas
    - En el corpus
    - En las consultas

webir - Correcciones Ortográficas – Distancia de Edición

- Distancia de edición entre s_1 y s_2 = mínimo número de operaciones de edición necesarias para transformar s_1 en s_2
  - Insertar, borrar o reemplazar un caracter
- Se puede asignar pesos a las operaciones
  - Probabilidades de reemplazo por el teclado
- Algoritmo de programación dinámica
- Ejercicio
  - Explicar porqué la distancia en edición entre s1 y s2 nunca es mayor que max{|s1|, |s2|}

webir - Búsquedas con ”Comodines”

- Caso general
  - Indice Permuterm
    - Consulta m*n → buscar n$m*
  - Indice k-gram
    - $castle$: $ca, cas, ast, stl, tle, le$
    - Consulta re*ve → buscar $re AND ve$



- Buscar un conjunto Q tal que W ⊂ Q
- Controlar los términos que cumplen las condiciones

webir - Correcciones Ortográficas - Soundex Clásico

- Hash fonético
  - Principalmente nombres propios
- Algoritmos Soundex
  - Reducir cada término a 4 caracteres
  - Construir un índice invertido de los términos de la colección reducidos a los originales – índice Soundex
  - Reducir los términos de la consulta de la misma forma que los originales
  - Buscar en el índice Soundex

webir - Correcciones Ortográficas – Soundex Clásico

- Variaciones en la forma de reducir los términos
- Original = término → código de largo 4
- 1 letra y 3 dígitos (entre 0 y 9)
  - Dejar la primera letra del término
  - Llevar a 0 las letras ”a”, ”e”, ”i”, ”o”, ”u”, ”h”, ”w”, ”y”
  - Llevar a 1 las letras ”b”, ”f”, ”p”, ”v”
  - Llevar a 2 las letras ”c”, ”g”, ”j”, ”k”, ”q”, ”s”, ”x”, ”z”
  - Llevar a 3 las letras ”d”, ”t”
  - Llevar a 4 la letra ”l”
  - Llevar a 5 las letras ”m”, ”n”
  - Llevar a 6 la letra ”r”
  - Eliminar digítos repetidos consecutivos, dejando sólo uno de ellos
  - Eliminar los 0s
  - Completar las posiciones del final con 0s

webir - Correcciones Ortográficas - Soundex Clásico

- Ejercicio
  - Dar dos nombres propios diferentes cuyos códigos soundex sea idénticos

webir - Correcciones Ortográficas - Soundex Clásico

- Dar dos nombres propios diferentes cuyos códigos soundex sea idénticos
  - "Robert" y "Rupert" → "R163"
- Suenan igual pero tienen diferente código
  - Huff (H100) y Hough (H200)
- Parecidos pero tienen códigos diferentes
  - Carrigan (C625) y Kerrigan (K625)

- Búsquedas en bases de datos genealógicas http://www.searchforancestors.com/utility/soundex.html

webir - Construcción de Índices – no alcanza la memoria

- Indexación basada en ordenamiento por bloques
- Indexación de un sólo pasaje en memoria – más eficiente
- Indexación distribuida – para grandes colecciones
- Indexación dinámica
- Seguridad
- Indexación para recuperación ordenada

webir - Construcción de Índices

- Depende del hardware
- Accesos a memoria son más rápidos que a disco
- Nomenclatura
  - Mantener en memoria los datos más usados – "**caching**"
  - Tiempo que lleva mover el cabezal de lectura de disco – "**seek**"
  - Tiempo de transferencia de disco a memoria por byte cuando el cabezal está en la pos. correcta – "**transfer rate**"
  - Bloques de memoria contiguos leídos a memoria – "**chunks**" (8, 16, 32, and 64 kilobytes)
  - Bloques de memoria a donde se transfiere de disco – "**buffer**"

webir - Construcción de Índices

Símbolo	Estadística	Valor
s	Promedio de tiempo de seek	5 ms = 5×10^{-3} s (SSD 0.16 ms)
b	Tiempo transferencia/byte	0.02 μs = 2×10^{-8} s
	Velocidad - Tiempo ciclo reloj del procesador	10^{-9} s
p	Tiempo operaciones bajo nivel (compare, swap)	0.01 μs = 10^{-8} s
	Tamaño memoria principal	varios GB
	Tamaño del espacio en disco	varios TB

- 0.2 s transferencia de **10 MB contiguos** de disco a memoria (10×10^6 bytes)
- $0.2 + 100 \times (5 \times 10^{-3}) = 0.7$ s si se almacenan en 100 porciones no contiguas

webir - Construcción de Índices

- Transferencias de disco a memoria son llevadas a cabo por el “bus”
- Procesador libre durante las transferencias
- Aprovechar
  - Guardar datos comprimidos
  - Transferir y descomprimir
  - En general es más eficiente

webir - Construcción de Índices - Proceso

- Colección de documentos a indexar
- Separar en palabras (tokenize)
- Procesamiento lingüístico para normalizar las palabras
- Ordenar las palabras en orden alfabético
  - pares de términos (clave primaria) y docID (clave secundaria)
- Unificar ocurrencias repetidas de palabras
- Ordenar postings por docID (clave secundaria)

Indexación Basada en Ordenamiento por Bloques (BSBI)

- Cada término se representa mediante un termID – eficiencia almacenamiento
  - Se pueden asignar los termID a medida que se construye el índice o
  - Se pueden asignar mediante un proceso en 2 pasadas – en la primera pasada se extraen los términos y se le asigna un termID y en la segunda pasada se construye el índice

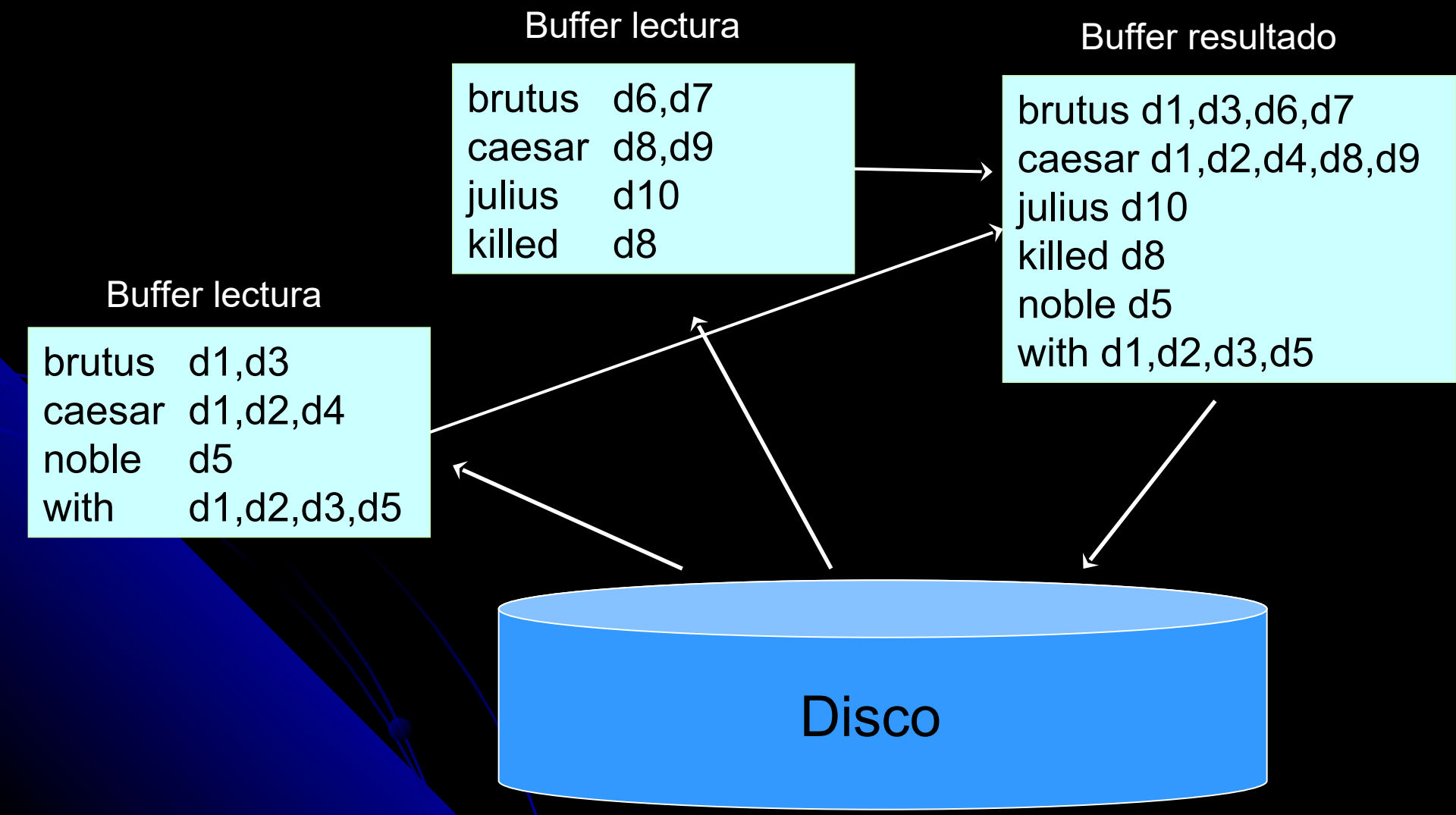
- Cuando la memoria es insuficiente es necesario usar algoritmos de ordenamiento externos (usan disco). Minimizar accesos a disco.

Indexación Basada en Ordenamiento por Bloques (BSBI)

1. Dividir la colección en porciones de igual tamaño (de tal forma que quede holgura en memoria)
2. Generar índice invertido de cada porción
   1. Obtener conjunto de pares (termID, docID)
   2. Ordenar los pares (termID, docID)
   3. Unificar ocurrencias repetidas de los términos
   4. Hasta que se llena un bloque de largo fijo
3. Guardar resultados parciales en disco
4. Unificar los resultados parciales (merge)
   - Abrir todos los bloques en simultáneo
   - Tener un pequeño buffer de lectura para cada bloque y uno para la escritura del resultado
   - En cada iteración elegir el menor termID, usar PQ o similar

Bloques contiguos para minimizar tiempos de seek.

Indexación Basada en Ordenamiento por Bloques (BSBI)



Indexación Basada en Ordenamiento por Bloques (BSBI)

- Ejemplo Reuters-RCV1
  - 1 GB de texto – 800 000 noticias
  - Noticias de un año 1996-1997
  - 100 millones de palabras no normalizadas (391,523 términos)
  - 4 bytes para cada termID y docID, total 0.8 GB todos los pares
    - Cuando la memoria es insuficiente es necesario usar algoritmos de ordenamiento externos (usan disco). Minimizar accesos a disco.

- BSBI para Reuters-RCV1
  - Si entran 10 millión de parejas termID–docID en memoria
  - Quedan 10 bloques para componer el índice invertido

Indexación Basada en Ordenamiento por Bloques (BSBI)

- O(T log T) debido al ordenamiento de los pares (termID, docID)
- En general dominado por
  - El tiempo para separar en palabras (tokenize) y el procesamiento lingüístico de cada bloque
  - El tiempo para unificar los resultados parciales

Indexación Basada en Ordenamiento por Bloques (BSBI)

- ¿Cómo se pueden asignar los termID a medida que se construye el índice en el BSIB?

Indexación Basada en Ordenamiento por Bloques (BSBI)

- ¿Cómo se pueden asignar los termID a medida que se construye el índice en el BSIB?
- Se necesita estructura auxiliar de memoria
- ¿Qué pasa si no cabe en memoria?

Indexación de un Sólo Pasaje en Memoria (SPIMI)

- En lugar de termID se usan los términos
- Algoritmo
  1. Dividir la colección en porciones de igual tamaño
  2. Generar índice invertido de cada porción procesando cada término en orden
     1. Si es la primera ocurrencia del término, agregarlo al diccionario parcial (hash)
     2. Agregar directamente docID a la lista de postings del término, si es necesario agrandar el espacio para la lista (no se guardan las (term, docID) como en BSBI)
  3. Guardar resultados parciales en disco – en orden lexicográfico → ordenar
  4. Unificar los resultados parciales (merge) igual que BSBI

Indexación de un Sólo Pasaje en Memoria (SPIMI)

- Diferencias entre BSBI y SPIMI
  - SPIMI agrega directamente el docID a la lista de postigs del término sin agruparlos todos y luego ordenarlos
  - SPIMI no requiere ordenamiento – más rápido
  - No es necesario guardar pares (termID, docID), sólo docID – menos memoria
  - SPIMI es O(T)

Indexación Distribuida

- Si la colección es muy grande no se puede llevar a cabo la indexación en una sola computadora
- Internet
- Clusters de computadoras
- Motores de búsquedas en Internet usan algoritmos de indexación distribuida
- Los resultados son también índices distribuidos, fraccionado por
  - Términos
  - Documentos (más frecuente)

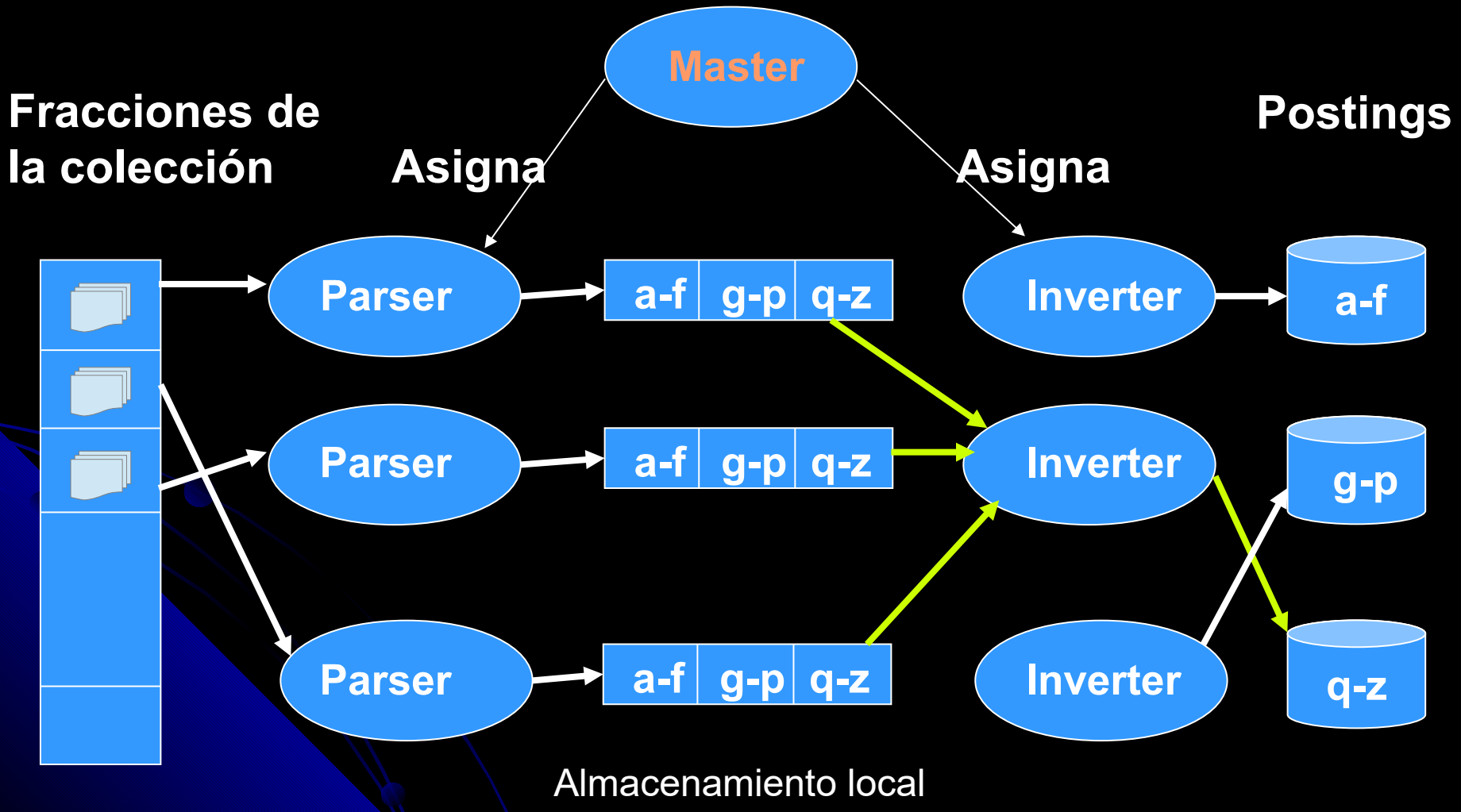
Indexación Distribuida – Términos

- Índices distribuidos, fraccionado por términos
- Arquitectura distribuida MapReduce
  - Varias computadoras estándar - nodos
  - Un “master node” que dirige el proceso, asignando y reasignando tareas
  - Google “MapReduce: Simplified Data Processing on Large Clusters ”
  - Procesamiento de parejas (clave, valor)

Indexación Distribuida – Términos

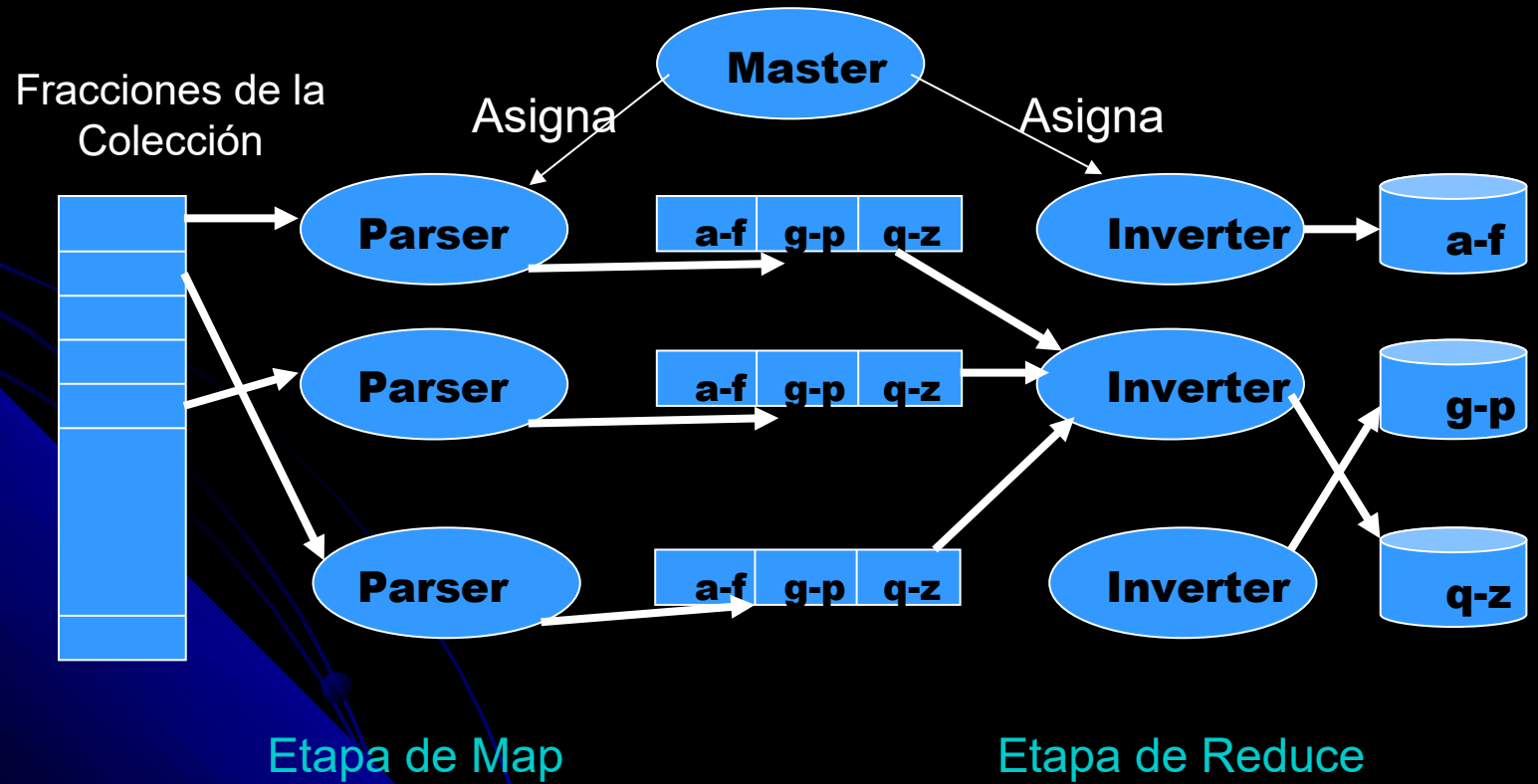
- Algoritmo
  - La colección de documentos es fraccionada
  - Las fracciones deben asegurar que el trabajo se pueda hacer de forma eficiente y pareja
  - Se debe llegar a un índice de (termID, docID)
  - La relación entre cada término y su termID también es distribuida
    - Lista de términos más frecuentes y su termID se copian a todos los nodos
    - Los demás términos se usan directamente en lugar de usar termID
  - La tarea inicial (Map) es de parsing, al igual que en BSBI y SPIMI, se genera un diccionario parcial en cada nodo, dividido en segmentos por ejemplo a-f, g-p y q-z
  - Luego (Reduce) se genera una sola lista de postings para cada termID (Invert), cada segmento es asignado a un nodo

Indexación Distribuida – Términos



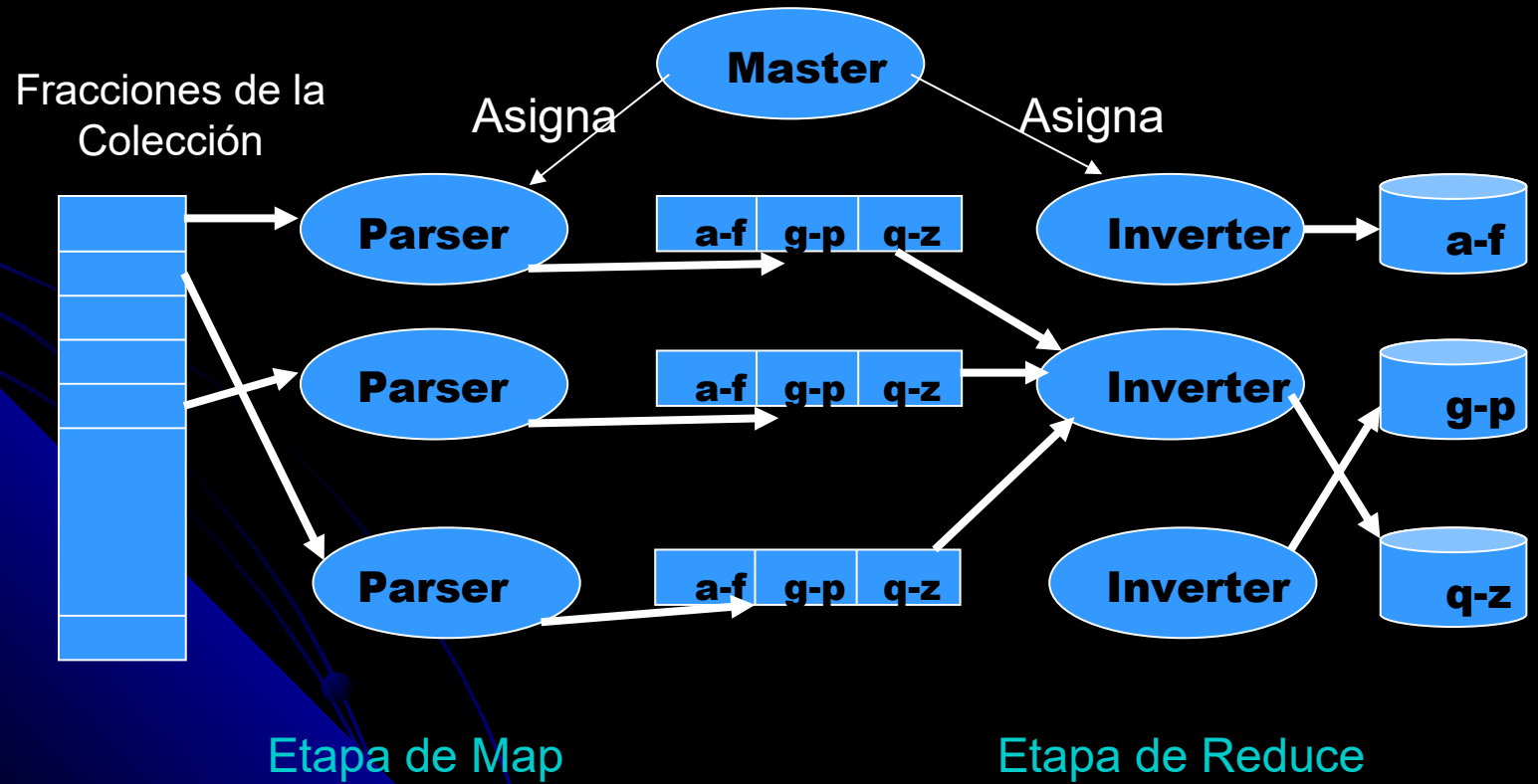
Ejercicios

- ¿Se puede obtener la cantidad de veces que cada término aparece en cada uno de los docs de la colección usando MapReduce? ¿Cómo?



Ejercicios

- ¿Se puede obtener la cantidad de veces que cada término aparece en cada uno de los docs de la colección usando MapReduce? ¿Cómo?



Ejercicios

- En la etapa de “invertir” en la indexación distribuida usando MapReduce, se debe distribuir de forma equitativa las fracciones del índice invertido a construir de modo tal que contengan aproximadamente la misma cantidad de postings. ¿Cómo?