Capítulo 4

Factor de selectividad y operadores físicos

Operación de selección

- Definimos fs(P, r) para distintos tipos de propiedades P y tabla r.
- Desde ahora *r* tabla, *A*, *A'* atributos de *r*, *c* y *c'* constantes.
- **Regla 1:** Asumiendo uniformidad:
  - fs(A = c, r) = 1/V(A, r), donde V(A, r) número de distintos valores que aparecen en *r* para *A*.
- **Regla 2**: Asumiendo uniformidad, A con valor numérico:
  - fs(A $\geq$ c, r) = (max(A, r) - c) / (max(A, r) – min(A, r))
- **Regla 3**: Asumiendo uniformidad, A con valor numérico:
  - fs(A < c, r) = (c – min(A)) / (max(A) – min(A) + 1)

Operación de selección

- **Regla 4:** asumiendo uniformidad, A con valor numérico:
  - fs ($c \leq A < c'$, r) = $(c' - c)/ (\max(A, r) - \min(A, r))$
- **Regla 5**: asumiendo independencia:
  - fs($P1 \wedge P2 \wedge \ldots \wedge Pn$, r) = fs($P_1$, r) fs($P_2$, r) …. fs($P_n$, r)
- **Regla 6**: usando propiedad de probabilidades:
  - fs($\neg P$, r) = 1 – fs(P, r)
- **Regla 7**: asumiendo independencia
  - fs($P \vee Q$, r) = fs($\neg (\neg P \wedge \neg Q$, r)) = 1 – fs($\neg P \wedge \neg Q$, r) = 1 – (fs($\neg P$, r) * fs($\neg Q$, r))
    = 1 – ((1 – fs(P, r)) * (1 – fs(Q, r)))

Operación de selección

- **Algoritmo de búsqueda lineal**
  - Escanear cada bloque del archivo y testear todos los registros para ver si satisfacen la condición de selección.
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
    - b_r denota el número de bloques conteniendo registros de la tabla *r*
    - Otra fórmula para el costo: $t_S + b_r * t_T$
  - Si se hace búsqueda lineal y se selecciona según igualdad para clave:
    - costo = (b_r /2) transferencias de bloques + 1 acceso a bloque
    - Este costo es para el caso promedio.

Operación de selección

- **Algoritmo para índice primario usando árbol B+ con igualdad en clave candidata**
  - Recorrer la altura del árbol más una E/S para recoger el registro.
  - Cada una de estas operaciones requiere un acceso a bloque y una transferencia de bloque.
  - $Costo = (h_i + 1) * (t_T + t_S)$,
  - donde h_i denota la altura del índice.

Operación de selección

- **Algoritmo para índice primario usando árbol B+ igualdad para no clave candidata**
  - Hay un acceso a bloque para cada nivel del árbol y un acceso a bloque para el primer bloque.
  - Los registros van a estar en bloques consecutivos.
  - Sea b el número de bloques conteniendo registros con la clave de búsqueda especificada, todos los cuales son leídos.
  - Estos bloques son bloques hoja asumiendo que están almacenados secuencialmente y no requieren accesos adicionales a bloque.
  - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

Operación de selección

- **Algoritmo para índice secundario usando árbol B+, igualdad en clave candidata**
  - Este caso es similar al índice primario.
  - *Costo* = $(h_i + 1) * (t_T + t_S)$,
  - donde h_i denota la altura del índice.
- **Algoritmo para índice secundario usando árbol B+, igualdad en no clave**
  - Aquí el costo de recorrido del índice es el mismo. Sin embargo, cada registro puede estar en un bloque diferente, requiriendo un acceso a bloque por registro.
  - Sea *n* el número de registros recogidos.
  - Costo = $(h_i + n) * (t_T + t_S)$
  - Esto puede ser muy costoso.

Selecciones involucrando comparaciones

- Podemos implementar selecciones de la forma $\sigma_{A \leq V}(r)$ o $\sigma_{A \geq V}(r)$ usando:
  - un escaneo de archive lineal,
  - o usando índices
- **Algoritmo para índice primario:**
  - La tabla está ordenada en *A*.
  - para $\sigma_{A \geq V}(r)$ usar el índice para encontrar el primer registro $\geq v$ y escanar la tabla secuencialmente desde allí.
    - Costo = hi $*$ (tT + tS) + b $*$ tT
    - b el número de bloques conteniendo registros con A $\geq v$.
  - para $\sigma_{A \leq V}(r)$ escanear la tabla secuencialmente hasta la primera tupla > *v;* no usar el índice.

Selecciones involucrando comparaciones

- **Algoritmo para índice secundario**
  - para $\sigma_{A \geq V}(r)$ usar el índice para encontrar la primera entrada del índice $\geq v$ y escanear el índice secuencialmente desde allí, para encontrar los punteros a los registros.
  - Costo = $(hi + n) * (t_T + t_S)$,
    - donde *n* número de registros con A >= v.
  - para $\sigma_{A \leq V}(r)$ escanear hojas del índice encontrando punteros a los registros, hasta la primera entrada > *v.*
  - En ambos casos retornar los registros que son apuntados
    - Requiere una E/S para cada registro
    - búsqueda lineal de la table puede ser más barata.

Selecciones involucrando comparaciones

- Leer implementación de selecciones complejas:
  - Sección 12.3.3 del Silberschatz.

Operación de proyección

- **Algoritmo para proyección**
  - Requiere recorrer todos los registros y realizar una proyección en cada uno.
  - Se recorren todos los bloques de la tabla.
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
    - b_r denota el número de bloques conteniendo registros de la tabla *r*
  - Proyección generalizada puede ser implementada de la misma manera que proyección.

Operación de ordenamiento

- Estudiamos el caso de ordenamiento donde las tablas son mayores que la memoria principal.
  - Ordenar tablas que no caben en memoria se llama **ordenamiento externo**.
  - El algoritmo de ordenamiento externo más usado se llama **ordenamiento-combinación externo (external merge sort)**.

Operación de ordenamiento

- Sea *M* la cantidad de bloques en búfer de memoria principal disponibles para ordenación.
  1. **Crear corridas ordenadas**

     i = 0

     repeat

     Leer *M* bloques de la tabla en memoria

     ordenar esos bloques en memoria

     Escribir los datos ordenados al archivo de ejecución R_i,

     i = i+1

     until el fin de la tabla

     - Sea N el valor final de i
  2. **Combinar las corridas**

Operación de ordenamiento

- Luego las corridas son combinadas. Suponemos que *N* <*M*.

  Leer un bloque de cada uno de los *N* archivos Ri en un bloque del bufer de memoria.

  repeat

    Seleccionar el primer registro en el orden de ordenamiento entre todos los bloques del bufer.

    Escribir el registro al bufer de salida y borrarlo del bloque del búfer

    if bufer de salida esta lleno **then** escribirlo a disco.

    Borrar el registro de su página de bufer de input.

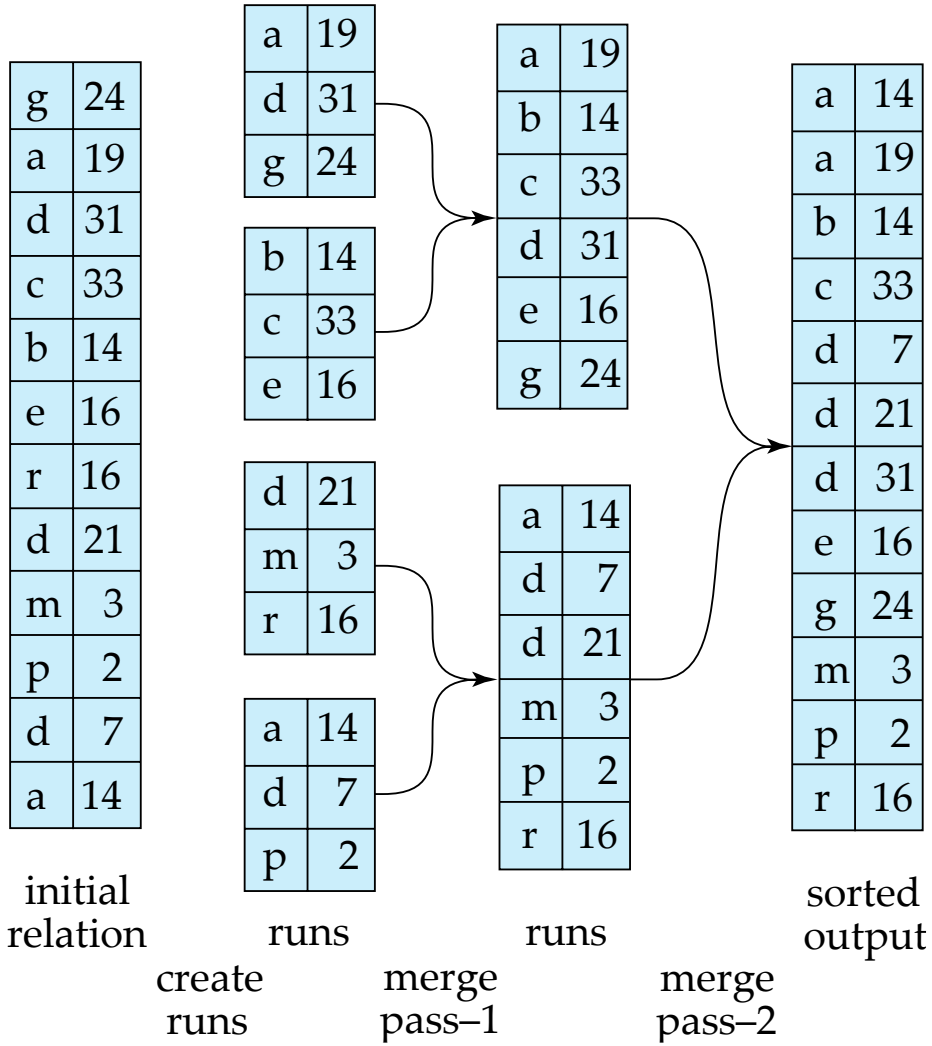
    if pagina del bufer de corrida Ri pasa a estar vacía **then** leer el próximo bloque de la corrida Ri en el bloque de bufer.

  until que todos los bloques del input búfer estén vacías.

Operación de ordenamiento

- Si $N \geq M$, varias pasadas de combinación son requeridas.
  - En cada pasada, grupos contiguous de M-1 corridas son combinados.
  - Una pasada reduce el número de corridas por un factor de M-1 y crrea corridas más largas por el mismo factor.
  - Pasadas repetidas son realizadas hasta que todas las corridas han sido combinadas en una.

Operación de ordenamiento



Operación de ordenamiento

- **Costo del ordenamiento:**
  - Leer justificación en el Silberschatz.
  - Número de transferencias de bloques para ordenamiento externo:
    $b_r (2 \lceil \log_{M-1} (b_r / M) \rceil + 1)$
  - Donde b_r es cantidad de bloques de la tabla
  - Cantidad de accesos a bloque:
    $2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{\lfloor M/bb \rfloor - 1} (b_r / M) \rceil - 1)$,
  - donde b_b bloques son alojados en cada corrida

Operación de eliminación de duplicados

- **Eliminación de duplicados**
  - Se ordena la tabla; al hacerlo, todos los duplicados van a ser adyacentes entre sí. Y todos los duplicados menos uno pueden ser eliminados.
  - **Optimización**: en ordenación externa duplicados pueden ser eliminados durante generación de corridas así como en pasos de combinación intermedios.
  - El peor caso de costo para eliminación de duplicados es el mismo que el peor caso de costo para ordenar una table.

Ejercicio

- Sea la consulta: $\Pi_{id}\, \sigma_{(edad > 40)}(persona)$
- Donde:
  - Persona ocupa 20 bloques de 10 registros cada uno.
  - La probabilidad de que un registro tenga edad mayor a 40 es 0.5.
  - Asumir que entran 50 id por bloque.
  - Hay 100 edades distintas
  - Entran 20 edades por nodo en el índice.
- Calcular cuántos bloques se leen y cuántos bloques se escriben para cada uno de los siguientes casos:
  1. No hay índice en edad
  2. Hay índice en edad usando árbol B+.

Operación de reunión selectiva

- Si r tabla y A atributo de r, entonces V(A, r) número de distintos valores que aparecen en *r* para *A*.
- **Regla**: asumiendo uniformidad:
  - fs(r.A = s.B, r, s) = 1 /max(V(A, r), V(B, s))
  - la uniformidad significa: para cada valor de atributo A/B en una tabla hay la misma cantidad de tuplas por el atributo B/A que cazan en la otra tabla.
- **Observaciones**:
  - Si A clave candidata de r: fs(r.A = s.B, r, s) = 1 /max(|r|, V(B, s))
  - Si B clave foránea en s referenciando r: fs(r.A = s.B, r, s) = 1 /|r|
  - Si toda tupla en r produce tuplas en la reunión selectiva:
    - fs(r.A = s.B, r, s) = 1 /V(B,s)
  - Si se puede asumir independencia:
    - fs(r.A = s.B Λ r.A' = s.B', r, s) = fs(r.A = s.B , r,s) * fs(r.A' = s.B', r,s)

Operación de reunión selectiva

- Vamos a ver algoritmos para computar la reunión selectiva (y por ende la reunión natural)
- **Algoritmo de reunión selectiva de loop anidado**
- **for each** tuple t_r **in** r **do begin**
  for each tuple t_s **in** s **do begin**
  test pair (t_r, t_s) to see if they satisfy the join condition
  if they do, add $t_r \bullet t_s$ to the result.
  end
  end

Operación de reunión selectiva

- Nomenclatura: r se llama **tabla externa** y s se llama **tabla interna** de la reunión selectiva.
- El algoritmo anterior no requiere de índices.
- El algoritmo anterior es costoso porque examina todo par de tuplas en las dos tablas.
- En el peor caso, si hay suficiente memoria solo para sostener un bloque para cada tabla, el costo estimado es:
  - $n_r * b_s + b_r$ transferencias de bloques
  - $n_r + b_r$ accesos a bloques.
  - donde n_r es cantidad de registros en *r* , b_r es la cantidad de bloques en *r,* y b_s es la cantidad de bloques en *s.*

Operación de reunión selectiva

- **Algoritmo de reunión selectiva de loop anidado de bloques**

for each block B_r **of** r **do begin**
for each block B_s **of** s **do begin**
for each tuple t_r **in** B_r **do begin**
for each tuple t_s **in** B_s **do begin**
if (t_r, t_s) satisface condición de reunión selectiva
then add $t_r \bullet t_s$ to the result.
end
end
end
end

Operación de reunión selectiva

- **Estimación de peor caso:**
  - $b_r * b_s + b_r$ transferencias de bloque + 2 * b_r accesos a bloque
  - Cada bloque en la tabla interna *s* es leído una sola vez por cada bloque en la tabla externa.
- **Mejora del algoritmo anterior:**
  - M tamaño de memoria en bloques.
  - Usar M-2 bloques de disco para tabla externa, usar los restantes 2 bloques para tabla interna y salida.
  - Costo = $\lceil b_r / (M-2) \rceil * b_s + b_r$ transferencias de bloques + $2 \lceil b_r / (M-2) \rceil$ accesos a bloque

Operación de reunión selectiva

- **Algoritmo de reunión selectiva de loop anidado indexado**
  - Asumir que hay un índice en el atributo de la relación interna de la reunión.
  - Para cada tupla t_r en la tabla externa *r* usar el índice para buscar tuplas en *s* que satisfacen la condición de reunión con tupla t_r.
  - **Peor caso**: el búfer tiene espacio para solo un bloque de *r* y un bloque del índice y para cada tupla en *r* hacemos búsqueda en índice de *s*.
    - Costo de la reunión selectiva: $b_r (t_T + t_S) + n_r * c$
    - Para leer *r* en el peor caso hay b_r accesos a bloque y b_r transferencias de bloque.
    - Aquí *c* es el costo de recorrer el índice y recolectar todas las tuplas de *s* que cazan para una tupla de *r*.
    - El valor *c* puede estimarse como el costo de una selección en *s* usando la condición de la reunión.

Operación de reunión selectiva

- **Algoritmo de reunión merge-sort:**
- Supongamos que hay un atributo de la reunión.
- Ordenar ambas tablas en el atributo de la reunión.
- Hacer merge de las relaciones ordenadas para hacer la reunión.
  - El paso de reunión es similar a la etapa de merge del algoritmo merge sort.
  - La principal diferencia es manejo de valores duplicados en el atributo de la reunión – cada par con el mismo valor de atributo del join debe ser juntado.
  - Leer detalles del algoritmo en el libro.
- Cada bloque necesita ser leído una sola vez asumiendo que todas las tuplas para un valor dado de los atributos del join entran en memoria.
- Entonces el costo de reunión merge-sort es:
  - $b_r + b_s$ transferencia de bloques $+\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ accesos a bloque
  - Donde b_b bloques de búfer son asignados a cada tabla.
  - + el costo de ordenar si las tablas no están ordenadas.

Operación de reunión selectiva

- **Algoritmo de reunión merge-sort híbrido:**
- **Suposición**: la primera tabla está ordenada y la segunda tabla está desordenada pero tiene un índice secundario árbol B+ en los atributos de la reunión.
- **Algoritmo**:
  1. El algoritmo combina la tabla ordenada con las entradas hoja del índice secundario B+.
  2. El archivo resultante contiene tuplas de la relación ordenada y direcciones para las tuplas de la relación no ordenada.
  3. Se ordena el resultado por las direcciones de las tuplas de la relación no ordenada, permitiendo el retorno eficiente de las tuplas correspondientes en el orden de almacenamiento físico para terminar la reunión.
- **Costo**:
  - Segundo paso: Costo de ordenar el archivo resultante.
  - Tercer paso: costo de acceso a tuplas en el orden de almacenamiento físico

Agregación

- **Algoritmo**:
  - **Agregación** puede ser implementada de una manera parecida a eliminación de duplicados.
  - Ordenación puede ser usada para juntar tuplas del mismo grupo y luego las funciones de agregación pueden ser aplicadas a cada grupo.
  - **Optimización**:
    - En lugar de recolectar todas las tuplas en un grupo y luego aplicar operadores de agregación, se puede implementar los operadores de agregación sum, min, max, count y avg al vuelo a medida que los grupos van siendo construidos.
    - Para el caso de sum, min, max, cuando dos tuplas en el mismo grupo son encontradas, el sistema las reemplaza por una sola tupla conteniendo el sum, min, o max respectivamente de las columnas siendo agregadas.
    - Para count se mantiene un count de cada grupo para el cual una tupla ha sido encontrada.
    - Para avg, se computan sum and count al vuelo y se divide sum por count al final.
- **Costo**: es el mismo que el costo de eliminación de duplicados.

Concatenación

- Concatenación requiere leer ambas tablas: primero la de la izquierda y luego la de la derecha.
  - A medida que se lee una tabla se genera el resultado.
- En el peor caso para computar r++s vamos a necesitar:
  - $b_r + b_s$ transferencias de bloques y accesos a bloques
- Si el resultado es un resultado intermedio: se escriben $b_r + b_s$ bloques en disco.

intersección

- **Algoritmo**:
  - Para calcular la intersección de dos tablas, podemos primero **ordenarlas** a ambas por la clave primaria.
  - Luego podemos escanear una vez cada tabla ordenada para producir el resultado.
- **Costo**: Suponiendo las tablas r, s ordenadas de ese modo, $r \cap s$ requiere:
  - Si un solo bloque en búfer se usa por tabla:
    - $b_r + b_s$ transferencias de bloques
    - $b_r + b_s$ accesos a bloque
  - El número de accesos a bloque puede ser reducido alojando bloques extra en búfer.

Resta

- **Algoritmo**:
  - Para calcular la resta de dos tablas, podemos primero **ordenarlas** a ambas por la clave primaria.
  - Luego podemos escanear una vez cada tabla ordenada para producir el resultado.
- **Costo**:
  - Suponiendo las tablas r, s ordenadas de ese modo.
  - Si un solo bloque en búfer se usa por tabla:
    - $b_r + b_s$ transferencias de bloques
    - $b_r + b_s$ accesos a bloque