TP Bases de Datos II

Parte 5

(ENTREGA 5)
Límite de tiempo: 2 semanas

Introducción

En esta parte del trabajo práctico vamos a encargarnos del split de los nodos tipo hoja.

Para hacer el split vamos a necesitar crear o actualizar un nodo interno. Aún no habíamos definido la estructura de un nodo interno así que la describiremos a continuación. Luego presentaremos los algoritmos de split.

Nodo interno

El header de un nodo interno será muy similar al del nodo hoja, con la diferencia de que guardaremos el número de claves contenidas en el nodo, en vez del número de pares clave valor, y el puntero (número de página) del hijo más a la derecha.
El cuerpo de un nodo interno contendrá una lista de pares puntero-clave, donde el puntero será simplemente el número de página hijo al que se está referenciando, y la clave será la clave de mayor valor contenida en el hijo del anterior puntero (o sea el hijo a la izquierda)

Formato Nodo Interno

- Header
  - **node_type** (1 byte)
    - 0 para nodos internos
    - 1 para nodos hoja
  - **is_root** (1 byte)
    - 0 para false
    - 1 para true
  - **parent_pointer** (4 bytes)
    - número de página en la que está el nodo padre (ignorada si es un nodo raíz)
  - **num_keys** (4 bytes)
  - **right_child_ptr** (4 bytes)
- Body
  - *Lista de pares puntero-clave se alojarán hasta 510 pares puntero-clave en el nodo)*
    - **ptr #** (4 bytes)
      - el número de página del hijo a la izquierda de la clave
    - **key #** (291 bytes)

- el valor de la columna id, del registro

(4080 bytes = 510 * 8)

byte 0	byte 1	bytes 2-5	bytes 6-9
node_type	is_root	parent_pointer	num keys
bytes 6-9		bytes 10-13	bytes 14-17
num keys		right child pointer	child pointer 0
bytes 14-17		bytes 18-21	bytes 22-25
child pointer 0		key 0	child pointer 1
bytes 22-25		bytes 26-29	...
child pointer 1		key 1	...
...			bytes 4086-4089
...			child pointer 509
bytes 4086-4089		bytes 4090-4093	bytes 4094-4095
child pointer 509		key 509	desperdicio

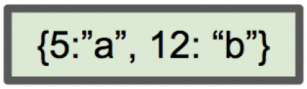
Nota al márgen:
El branching factor de los nodos internos, o sea, la cantidad de ramificaciones que puede contener es muy grande (511 hijos) ya que los pares puntero-clave ocupan muy poco espacio. Esto deriva en que cuando queramos buscar un valor vamos a tener que atravesar pocos niveles del árbol, haciendo la búsqueda muy eficiente en términos de lecturas a disco. Veamos algunos números

cant niveles de nodos internos	máximo número de nodos hoja	tamaño total de nodos hoja
0	511 ^ 0 = 1	4 kb
1	511 ^ 1 = 511	~2 mb
2	511 ^ 2 = 261.121	~1 gb
3	511 ^ 3 = 133.432.831	~550 gb

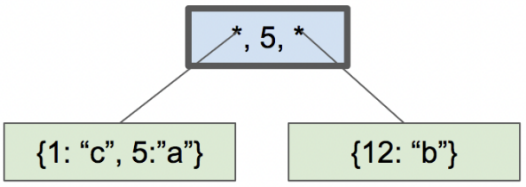
Esto quiere decir que podemos buscar un registro por su id, en una tabla de 500 gb, cargando en memoria solo 4 páginas del disco.
Esto es lo que hace que el B-Tree sea una estructura de datos tan útil para bases de datos

Algoritmo de split del nodo hoja

Hasta ahora, cuando intentamos insertar un registro en una hoja que ya está completa presentamos un mensaje de error.
Ahora vamos a cambiar esto para distribuir su contenido en dos nodos hoja, y luego crearemos un nodo padre, que será de tipo nodo interno.
El objetivo es ir de esto (*considerando un arbol de orden 3 para simplificar el gráfico*):



a esto:



Cada uno de estos nuevos nodos ocupará una nueva página en el archivo. Por otro lado, el nodo padre reemplazará la página del nodo original, de esta manera garantizamos que la primera página del archivo (offset 0) contendrá al nodo raíz.

Si miramos el archivo antes y después del split debería quedar de la siguiente manera:

Estado inicial *(nodo hoja-raíz completo)*:

página	0	**type**	**is_root**	**parent**				**# recs**			
offset	0	Leaf	TRUE	*garbage*				13			
		pair 1 (key,val)		**pair 2**		**pair 3**		**pair 4**		**pair 5**	
		1	v1	2	v2	6	v3	7	v4	10	v5
		pair 6		**pair 7**		**pair 8**		**pair 9**		**pair 10**	
		17	v6	18	v7	25	v8	29	v9	34	v10
		pair 11		**pair 12**		**pair 13**					
		39	v11	42	v12	48	v13				

Después de insertar el registro con id 50:

página	0	type	is_root	parent	# keys	right child ptr
offset	0	Internal	TRUE	*garbage*	1	2
		pair 1 (pag,key)				
		1	18			

página	1	type	is_root	parent		# recs			
offset	4096	Leaf	FALSE	*0*		7			
		pair 1 (key,val)		pair 2		pair 3		pair 4		pair 5	
		1	v1	2	v2	6	v3	7	v4	10	v5
		pair 6		pair 7							
		17	v6	18	v7						

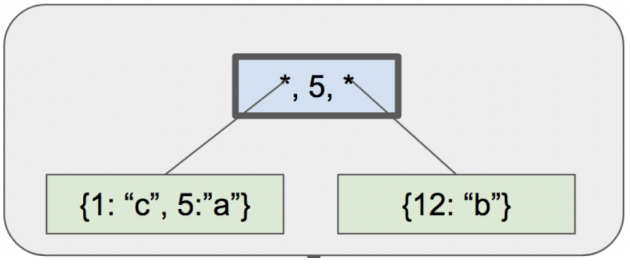
página	2	type	is_root	parent		# recs			
offset	8192	Leaf	FALSE	*0*		7			
		pair 1 (key,val)		pair 2		pair 3		pair 4		pair 5	
		25	v8	29	v9	34	v10	39	v11	42	v12
		pair 6		pair 7							
		48	v13	50	v14						

Como se puede ver en el header del nuevo nodo raíz (página 0), sobreescribimos la página, con el nodo padre que es de tipo interno, que tiene solo un par puntero-clave **pair 1=(1,18)** que determina que el hijo a la izquierda está en la página 1, y valor máximo para las claves bajo esa rama es 18. Por otro lado, en el header el campo **right child ptr** que determina que el hijo más a la derecha de ese nodo se encuentra en la página 2.

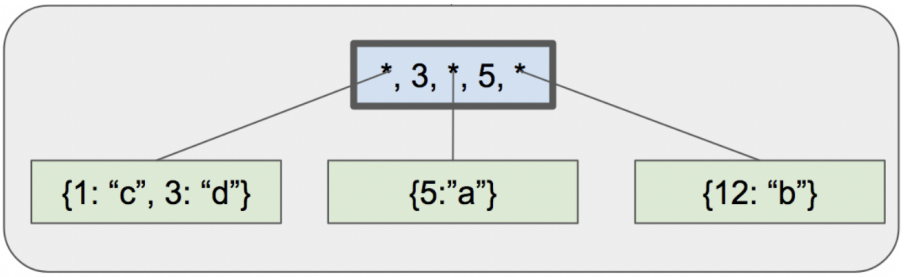
Actualización del nodo padre luego de un split

Cuando se necesite hacer un split de un nodo hoja que no es raíz habrá que escalar la actualización de las claves al nodo padre, que por definición, será un nodo interno.

Queremos pasar de esto:



a esto:



En este ejemplo, agregamos la clave 3, esto deriva en el split de la hoja izquierda. Luego del split, hay que actualizar el árbol haciendo lo siguiente:

1. actualizar la primera clave en el padre para que sea la clave máxima del hijo izquierdo
2. agregar el nuevo par puntero-clave a continuación de la clave recientemente actualizada. El puntero apuntará al hijo nuevo, y la clave será la clave máxima de ese nuevo hijo.

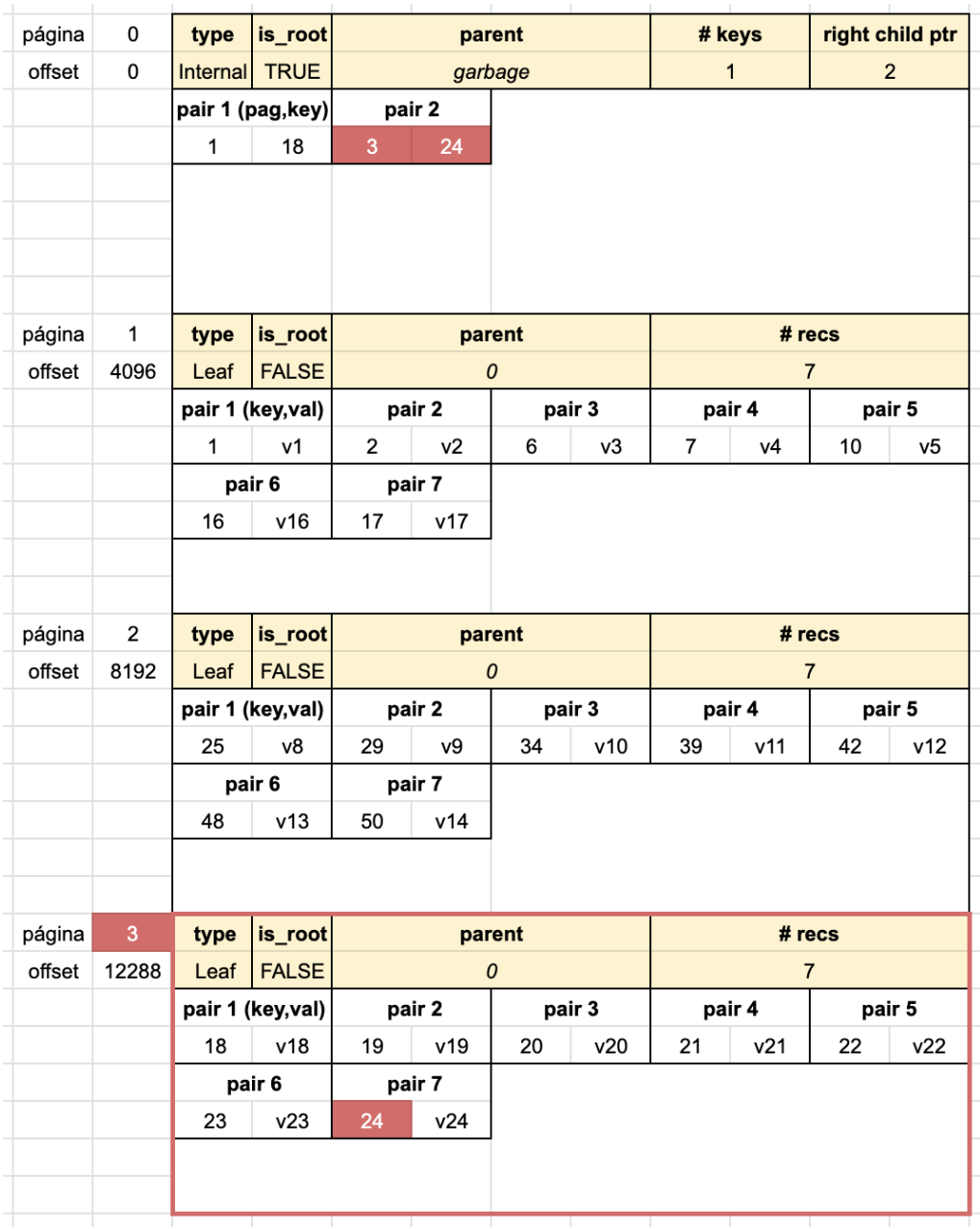
Continuando con el ejemplo anterior, asumiendo que vamos a insertar un registro con id 24, si miramos el archivo antes y después del split debería quedar de la siguiente manera:
Antes:

página	0	**type**	**is_root**	**parent**		**# keys**	**right child ptr**
offset	0	Internal	TRUE	*garbage*		1	2
		pair 1 (pag,key)					
		1	18				

página	1	**type**	**is_root**	**parent**		**# recs**	
offset	4096	Leaf	FALSE	*0*		7	
		pair 1 (key,val)		**pair 2**		**pair 3**	
		1	v1	2	v2	6	v3
		pair 4		**pair 5**			
		7	v4	10	v5		
		pair 6		**pair 7**		**pair 8**	
		16	v16	17	v17	18	v18
		pair 9		**pair 10**			
		19	v19	20	v20		
		pair 11		**pair 12**		**pair 13**	
		21	v21	22	v22	23	v23

página	2	**type**	**is_root**	**parent**		**# recs**	
offset	8192	Leaf	FALSE	*0*		7	
		pair 1 (key,val)		**pair 2**		**pair 3**	
		25	v8	29	v9	34	v10
		pair 4		**pair 5**			
		39	v11	42	v12		
		pair 6		**pair 7**			
		48	v13	50	v14		

Después de insertar el registro:



página	0	type	is_root	parent		# keys	right child ptr
offset	0	Internal	TRUE	*garbage*		1	2
		pair 1 (pag,key)		pair 2			
		1	18	3	24		

página	1	type	is_root	parent		# recs			
offset	4096	Leaf	FALSE	*0*		7			
		pair 1 (key,val)		pair 2		pair 3		pair 4		pair 5	
		1	v1	2	v2	6	v3	7	v4	10	v5
		pair 6		pair 7							
		16	v16	17	v17						

página	2	type	is_root	parent		# recs			
offset	8192	Leaf	FALSE	*0*		7			
		pair 1 (key,val)		pair 2		pair 3		pair 4		pair 5	
		25	v8	29	v9	34	v10	39	v11	42	v12
		pair 6		pair 7							
		48	v13	50	v14						

página	3	type	is_root	parent		# recs			
offset	12288	Leaf	FALSE	*0*		7			
		pair 1 (key,val)		pair 2		pair 3		pair 4		pair 5	
		18	v18	19	v19	20	v20	21	v21	22	v22
		pair 6		pair 7							
		23	v23	24	v24						

En el caso de que el split se haga sobre el hijo de más a la derecha, creando un nuevo nodo más a la derecha, entonces hay que actualizar también ese puntero (right child ptr) en el nodo padre.

Requerimientos

- Actualizar el `INSERT` para que implemente los dos casos de split presentados anteriormente.
    - Si se intenta insertar un nuevo registro superado el límite del nodo interno (el padre) se lanzará un error de "Nodo interno lleno, split no implementado"

- Actualizar el `SELECT` para que recorra el árbol y muestre todos los registros
- Actualizar el metacomando .table-metadata