PARTE II

- Supponiamo di avere un file di 450.000 record. Ogni record occupa 85 byte, di cui 15 per il campo chiave. Ogni blocco contiene 2048 byte. Un puntatore a blocco occupa 5 byte. Non è richiesto l'allineamento dei campi ad indirizzi multipli di 4 e i record non sono puntati. Usiamo una organizzazione B^+-tree. La differenza con il B-tree normale è che ogni blocco del solo livello foglia punta al blocco successivo. Qual è il numero minimo di blocchi che dobbiamo utilizzare complessivamente per il file indice (totale dei blocchi di tutti i livelli) e per il file principale (livello foglia)?

- Sia dato il seguente scheduling S per l'insieme di transazioni T1, T2, T3

T1	T2	T3
		READ(Y)
	READ(X)	
READ(Y)		
Y = Y - 50		
		Y = Y - 20
	X = X + 50	
READ(X)		
X = X + Y		
	WRITE(X)	
WRITE(Y)		
		WRITE(Y)
WRITE(X)		

Il controllo della concorrenza è basato su timestamp, e le transazioni hanno i seguenti timestamp: TS(T1) = 120, TS(T2) = 110, TS(T3) = 100.

Descrivere come vengono modificate durante lo schedule le variabili READ_TIMESTAMP e WRITE_TIMESTAMP associate agli item X ed Y, e come procedono le tre transazioni. Fornire inoltre i valori finali dei due item X ed Y assumendo X_0 e Y_0 come valori iniziali.

SOLUZIONI PARTE II

Esercizio 3

- Il numero minimo di blocchi viene utilizzato quando i blocchi sono interamente pieni. Ovviamente i record del file principale hanno occupazioni diverse da quelli del file indice, che contengono solo chiave e puntatore al blocco del livello inferiore. Fa eccezione il primo record di ogni blocco indice che contiene solo il puntatore. Tale puntatore viene seguito in fase di ricerca in corrispondenza di valori minori rispetto alla chiave del record successivo. Nel caso del B^+-tree inoltre, nei blocchi del livello foglia, bisogna tenere conto dello spazio da allocare al puntatore al prossimo blocco. Questo non significa che i record dei blocchi foglia siano puntati, perché puntiamo genericamente all'inizio del prossimo blocco, senza tenere conto del valore della chiave del primo record in esso effettivamente contenuto (ovviamente questa chiave sarà maggiore di tutte quelle dei record del blocco precedente).

- Ricordiamo che per **comodità di calcolo** (non per definizione del B-tree) è consentito utilizzare come massimo numero di record contenuti in un blocco un numero dispari: tale approssimazione ha senso quando dobbiamo calcolare la metà della capienza del blocco (i record devono essere pieni almeno per metà) e approssimarla per eccesso (infatti nella espressione **capienza = 2d-1** il numero **d** rappresenta la parte intera superiore della metà di un numero dispari), oppure quando vogliamo calcolare in anticipo l'altezza massima che potrebbe raggiungere l'albero. In generale infatti un blocco andrebbe riempito interamente (eventualmente con un numero pari di record) e, nei casi di occupazione minima, andrebbe verificato che la metà di questo numero di record riempia effettivamente **almeno** la metà del blocco.

Calcoliamo prima di tutto il numero di record che possono essere memorizzati in un blocco del file principale:

NRF = ë(capacità blocco - taglia puntatore a blocco) / taglia recordû = ë2043 / 85û = ë24,03û = 24
Si prende la parte intera **inferiore** della divisione perché i record non devono essere spezzati, e la parte intera inferiore della divisione rappresenta appunto il massimo numero di record che entrano per intero in un blocco.

Possiamo ora calcolare quanti blocchi occorrono per memorizzare il file principale

NBF = é numero record / numero record per blocco ù = é450.000 / 241ù = é18.750ù = 18.750
Si prende la parte intera **superiore** della divisione perché i blocchi vanno allocati interamente, cioè non allochiamo frazioni di blocco.

Calcoliamo ora quanti record indice possono essere memorizzati in un blocco, tenendo presente che il primo record di ogni blocco contiene solo un puntatore. Posiamo effettuare il calcolo in due modi:

- se **N** e' il numero di record memorizzabili, dobbiamo calcolare **N** in modo che

(N-1)*(chiave) + N*(puntatore)≤ capacità blocco
(N-1) *15 + N*5 ≤2048 N*20 - 15 ≤2048 N*20≤2048+15 N≤2063/20
N ≤ 103,15 e poiché N deve rappresentare un numero intero di record otteniamo N=103

- possiamo considerare la capacità del blocco dopo aver già memorizzato il primo puntatore, e calcolare quanti record interi composti da chiave e puntatore possono essere ancora memorizzati; questo numero va poi incrementato di uno per tener conto del puntatore memorizzato prima

N = ë(capacità blocco - puntatore) / taglia record indice completoû + 1= ë(2048-5) / 20û + 1
N = ë2043 / 20û + 1 = ë102,15û + 1 = 102 + 1 = 103

Come al solito si prende la parte intera **inferiore** della divisione perché i record non devono essere spezzati.

A questo punto cominciamo a calcolare l'occupazione in blocchi del file indice, procedendo per livelli e ricordando che ad ogni livello sono è presente un record indice per ogni blocco del livello inferiore, quindi al primo livello di indice dobbiamo inserire un record per ogni blocco del file principale, al secondo livello un record per ogni blocco del primo, e così via fino ad arrivare alla radice composta da un solo blocco; inoltre i blocchi devono essere allocati:

BI1 = é18.750 / 103ù = é182,04ù = 183 blocchi
BI2 = é183 / 103ù = é1,78ù = 2 blocchi
BI3 = é2/ 103ù = 1 blocco

Occupazione totale indice 183+2+1 = 186

Esercizio 4

Assumiamo che all'inizio RTS(X) = 0 (READ_TIMESTAMP di X), RTS(Y) = 0 (READ_TIMESTAMP di Y), WTS(X) = 0 (WRITE_TIMESTAMP di X), WTS(Y) = 0 (WRITE_TIMESTAMP di Y), e che i valori iniziali degli item X ed Y nella memoria condivisa siano rispettivamente x0 ed y0 (ogni transazione esegue le operazioni in memoria locale e poi le riporta in memoria condivisa tramite la WRITE). Ricordiamo che i timestamp delle transazioni sono TS(T1) = 120, TS(T2) = 110, TS(T3) = 100. Consideriamo le operazioni dello schedule nell'ordine in cui vengono eseguite.

Passo	Transazione	Operazione	RTS(X)	RTS(Y)	WTS(X)	WTS(Y)	X	Y
1	**T3** TS(T3)=100	READ(Y)	0	100	0	0	x0	y0
2	**T2** TS(T2)=110	READ(X)	110	100	0	0	x0	y0
3	**T1** TS(T1)=120	READ(Y)	110	120	0	0	x0	y0
4	**T1** TS(T1)=120	Y=Y-50	110	120	0	0	x0	y0
5	**T3** TS(T3)=100	Y=Y-20	110	120	0	0	x0	y0
6	**T2** TS(T2)=110	X=X+50	110	120	0	0	x0	y0
7	**T1** TS(T1)=120	READ(X)	120	120	0	0	x0	y0
8	**T1** TS(T1)=120	X = X +Y	120	120	0	0	x0	y0
9	**T2** TS(T2)=110	WRITE(X) **ROLL BACK**	120	120	0	0	x0	y0
10	**T1** TS(T1)=120	WRITE(Y)	120	120	0	120	x0	y0-50
11	**T3** TS(T3)=100	WRITE(Y) ROLL BACK	120	120	0	120	x0	y0-50
12	**T1** TS(T1)=120	WRITE(X)	120	120	120	120	x0+ y0-50	y0-50

Al passo 9 la transazione T2 viene abortita. Dovrebbe eseguire la scrittura dell'item X, ma il suo timestamp è minore del timestamp della transazione più giovane (con timestamp più alto) che ha letto l'item X (RTS(X) = 120 > TS(T2) =110). Ciò significa che una transazione che ha iniziato le proprie operazioni dopo T2 ha già letto il valore dell'item X, mentre secondo l'ordine di esecuzione avrebbe dovuto leggere il valore di X già modificato da T2. Da qui la necessità di eseguire il roll back della transazione T2.
Al passo 10 la transazione T1 effettua con successo la scrittura (RTS(Y)=120 = TS(T1) - da notare che la relazione di uguaglianza deve essere ammessa, altrimenti nessuna transazione riuscirebbe a scrivere un item dopo averlo letto !).
Al passo 11 la transazione T3 viene abortita. Dovrebbe eseguire la scrittura dell'item Y, ma il suo timestamp è minore del timestamp della transazione più giovane (con timestamp più alto) che ha letto l'item X (RTS(X) = 120 > TS(T2) =100). Ciò significa che una transazione che ha iniziato le proprie operazioni dopo T3 ha già letto il valore dell'item X, mentre secondo l'ordine di esecuzione avrebbe dovuto leggere il valore di X già modificato da T3. Da qui la necessità di eseguire il roll back della transazione T3.
Per lo stesso motivo illustrato per il passo 10, la transazione T1 completa con successo la scrittura al passo 12.