PARTE II

3- È dato un file di 250.000 record. I record sono a lunghezza fissa non puntati. Ogni record occupa 325 byte, di cui 75 per la chiave. Un puntatore a blocco occupa 5 byte. Un blocco di memoria contiene 2048 byte. Utilizziamo una organizzazione hash con una funzione che prende come argomento la chiave, e distribuisce uniformemente i record in 350 bucket.
- Quanti blocchi di memoria occupa la bucket directory?
- Quanti blocchi di memoria occupa ogni bucket?
- Qual e' il numero medio di accessi a blocco richiesti per ricercare un record?

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

T1	T2	T3
	READ(X)	
		READ(Y)
	X = X - 20	
READ(X)		
X = X + 10		
		Y = Y - 5
WRITE(X)		
		WRITE(Y)
	WRITE(X)	
READ(Y)		
Y = Y + 20		
WRITE(Y)		

Il controllo della concorrenza è basato su timestamp, e le transazioni hanno i seguenti timestamp: TS(T1) = 110, TS(T2) = 100, TS(T3) = 105.

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.

SOLUZIONI PARTE II

Esercizio 3

- Calcoliamo prima di tutti quanti blocchi occorrono per la bucket directory. Tale directory è un array con indici [0…(bucket-1)] che contiene i puntatori ai bucket. È necessario assicurarsi che le strutture che sono memorizzate in un blocco non siano frammentate. In particolare in questo caso bisogna assicurarsi che tutti i puntatori contenuti in un blocco siano interi. Calcoliamo allora il numero NP di puntatori che possono essere interamente memorizzati in un blocco

NP = ëcapacità blocco / taglia puntatoreû = ë2048 / 5û = ë409,6û = 409

Si prende la parte intera **inferiore** della divisione proprio perché i puntatori non devono essere spezzati, e la parte intera inferiore della divisione rappresenta il massimo numero di puntatori che entrano per intero in un blocco.

La bucket directory contiene in questo caso 350 puntatori, quindi un blocco è sufficiente a memorizzarla.

- Per calcolare il numero di blocchi necessario per memorizzare i singoli bucket, dobbiamo prima di tutto calcolare quanti record devono far parte di un bucket. Ricordiamo che, nel caso migliore di funzione hash che riempie i bucket in maniera uniforme, sono i **record** ad essere distribuiti uniformemente tra i bucket, e **non i blocchi di dati** in cui sono stati prima memorizzati i record. Si può dimostrare infatti che nel secondo caso in un blocco potrebbero essere memorizzati record appartenenti di fatto a bucket diversi.

Avendo assunto che la funzione hash distribuisca uniformemente i record, il numero di questi per bucket sarà

RB = numero record per bucket = énumero record / numero bucketù = é250.000 / 350ù = é714,28ù = 715
Si prende la parte intera **superiore** della divisione perché dobbiamo assicurarci che ogni record trovi collocazione in un bucket. Di fatto considerando al contrario la parte intera inferiore 714 ci ritroviamo alla fine a prevedere spazio per 714 X 350 = 249.900 record invece che per 250.000 record.

Calcoliamo ora quanti record interi possono essere memorizzati in un blocco, considerando che ci occorre anche memorizzare, in ogni blocco di un bucket, un puntatore al blocco successivo

R = numero record per blocco = ë(capacità blocco – taglia puntatore) / taglia recordû = ë2043 / 325û = ë6,28û = 6
La motivazione che ci porta a prendere la parte intera **inferiore** della divisione è anche in questo caso quella di dover memorizzare i record nella loro interezza.

Possiamo ora calcolare quanti blocchi occorrono per memorizzare un bucket

NB = énumero record in un bucket / numero record per blocco ù = é715 / 6ùm = é119,16ù = 120
Si prende la parte intera **superiore** della divisione perché i blocchi vanno allocati interamente, cioè non allochiamo frazioni di blocco.

- La funzione hash (e quindi anche la ricerca) prende come argomento la chiave dei record, che li identifica univocamente (due record non possono avere lo stesso valore di chiave), quindi nella ricerca possiamo fermarci al primo record che presenta il valore cercato. In questo caso, e comunque se il record con la chiave cercata è presente, accediamo in media alla metà dei blocchi di cui è composto il bucket.

MA = énumero blocchi in un bucket / 2 ù = é120 / 2ù = 60
A tale numero bisogna aggiungere eventualmente un ulteriore accesso nel caso in cui la bucket directory non sia in memoria principale.

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) = 110, TS(T2) = 100, TS(T3) = 105. 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	**T2** TS(T2)=100	READ(X)	100	0	0	0	x0	y0
2	**T3** TS(T3)=105	READ(Y)	100	105	0	0	x0	y0
3	**T2** TS(T2)=100	X=X-20	100	105	0	0	x0	y0
4	**T1** TS(T1)=110	READ(X)	110	105	0	0	x0	y0
5	**T1** TS(T1)=110	X=X+10	110	105	0	0	x0	y0
6	**T3** TS(T3)=105	Y=Y-5	110	105	0	0	x0	y0
7	**T1** TS(T1)=110	WRITE(X)	110	105	110	0	x0+10	y0
8	**T3** TS(T3)=105	WRITE(Y)	110	105	110	105	x0+10	y0-5
9	**T2** TS(T2)=100	WRITE(X) **ROLL BACK**	110	105	110	105	x0+10	y0-5
10	**T1** TS(T1)=110	READ(Y)	110	110	110	105	x0+10	y0-5
11	**T1** TS(T1)=110	Y=Y+20	110	110	110	105	x0+10	y0-5
12	**T1** TS(T1)=110	WRITE(Y)	110	110	110	110	x0+10	(y0-5)+20

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) = 110 > TS(T2) =100). 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.