UNIVERSITÀ DEGLI STUDI DI CATANIA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN INFORMATICA, PRIMO LIVELLO

SPATARO FABRIZIO

DAI BTREE AI BTREE+
SVILUPPO E ANALISI DELLE PRESTAZIONI

RELAZIONE PROGETTO FINALE

TUTOR UNIVERSITARIO:
Prof. Domenico Cantone

ANNO ACCADEMICO 2004 - 2005

INDICE

PREFAZIONE

In questa relazione verranno trattati i BTree, strutture dati che permettono di definire ed eseguire operazioni su sistemi dinamici.
Oltre ai BTree verranno trattate due sue ottimizzazioni che, successivamente, saranno analizzate sia dal punto di vista illustrativo sia mediante un testing per comprenderne le potenzialità.

I BTree sono alberi bilanciati di ricerca progettati per essere memorizzati su memorie esterne tipicamente più lente rispetto a quelle interne. Quindi la misura di efficienza (principale) non è il tempo necessario per accedere ad un determinato dato, ma la quantità di operazioni di I/O effettuate. In un BTree esistono strategie per mantenere bassa l'altezza dell'albero dato che, in tutte le operazioni effettuate, il numero di I/O aumenta con l'aumentare dell'altezza.

Le strutture dati inserite come ottimizzazioni dei BTree sono gli BTree+ e i BTree++.

Per ogni struttura saranno trattate le tipiche operazioni di inserimento e cancellazione dei dati analizzando il meccanismo per ritardare il più possibile la crescita in altezza dell'albero.

In chiusura verranno presentati dati sperimentali che consentono di confrontare le prestazioni dei BTree rispetto a quelle dei BTree+.

INTRODUZIONE: IMMAGAZZINAMENTO DEI DATI SU MEMORIE ESTERNE

In informatica quando si cerca di risolvere un problema l'analisi è una parte vitale dell'intero processo di progettazione. I due aspetti da considerare principalmente sono:

- i tipi di dati sui quali lavorare
- le operazioni da eseguire su quei dati per ottenere il risultato voluto.

I due aspetti sono strettamente legati fra loro e la scelta della struttura dati influenza l'algoritmo in modo significativo. In molti casi sono possibili più strutture dati alternative tutte apparentemente idonee a risolvere ugualmente bene il problema.
L'analisi preliminare dei costi in termini di operazioni necessarie, consente di preferire una struttura rispetto alle altre: infatti nel caso in cui viene scelta una struttura dati non ottimale potrebbe accadere di dover scrivere degli algoritmi estremamente complessi in casi in cui questo è evitabile.

Tra le operazioni che i calcolatori sono in grado di svolgere, la memorizzazione delle informazioni comporta l'utilizzo di una unità esterna (oltre la RAM che viene intesa memoria interna).
La memoria esterna più comune è l'Hard Disk la cui struttura è costituita da uno o più piatti ricoperti da materiale magnetico su entrambe le facce. Le unità possono avere dieci o più piatti disposti l'uno sull'altro su un asse passante per il centro del piatto.
Ciascuna delle superfici è fornita di una propria testina di lettura e tutte le testine si muovono insieme verso l'interno o verso l'esterno, per leggere l'informazione che si trova a differenti distanze dal centro.

Per leggere o scrivere un determinato byte su un disco, dobbiamo:

1) posizionare le testine in modo che una di esse stia sopra la traccia del byte prescelto;
2) aspettare che, per la rotazione del disco, il byte prescelto si posizioni sotto la testina.

Ciascuna operazione può impiegare, in media, qualche centesimo di secondo; il tempo, in realtà, dipende non solo dalle caratteristiche dell'unità a disco, ma anche dal numero di tracce che le testine devono attraversare e dalla posizione del byte rispetto alle testine, nel momento in cui queste raggiungono la traccia. [3]
Spesso il tempo necessario per accedere e leggere una informazione memorizzata in un disco magnetico è superiore al tempo necessario all'elaboratore per esaminare tutta l'informazione letta!!! [5]

Proprio per questo motivo, quando occorre lavorare con grandi quantità di dati è spesso impossibile (o non opportuno) mantenere l'intera struttura dati all'interno della RAM. Invece è fortemente consigliato tenere in RAM una piccola porzione di struttura dati e quando necessario recuperarne altre porzioni dalla memoria di massa.
I dati inseriti in RAM possono venire manipolati con una maggior velocità grazie ai tempi di accesso 10^5 volte superiori a quelli della memoria secondaria. [5]
In conclusione possiamo dedurre che la performance globale dipende non più dalla velocità di accesso di un "singolo dato" ma dalla quantità di operazioni di Input/Ouput che vengono eseguite per risolvere l'intero problema.
[Per approfondimenti si veda 1, 2]

PERCHÈ BTREE?

I BTree sono alberi bilanciati di ricerca progettati per le operazioni sui dischi magnetici o altri dispositivi di memoria secondaria ad accesso diretto. Grazie all'uso della struttura dati ad albero e la caratteristica di poter memorizzare più chiavi in un nodo, il BTree (o Balanced Tree) è la struttura dati migliore per la gestione sia delle memorie interne che di quelle esterne. Infatti il BTree è ottimizzato per mantenere più elementi in un singolo nodo (che può corrispondere ad una pagina della memoria virtuale) così che vengono minimizzati i tempi di accesso al disco per trovare la chiave richiesta.

Il numero di elementi in un nodo è legato al "grado di ramificazione" dell'albero. Un grado di ramificazione elevato riduce sensibilmente sia l'altezza dell'albero, che il numero di accessi necessari a trovare una chiave qualsiasi.

PERCHÈ BTREE+?

Il BTree+ è una delle varianti dei BTree con l'obiettivo di diminuire il tempo di accesso per letture sequenziali dei dati, al fine di incrementare le prestazioni in quelle applicazioni che richiedono sia accesso random che accesso sequenziale (per es. streaming Video/Audio o grandi database).

1. BTREE

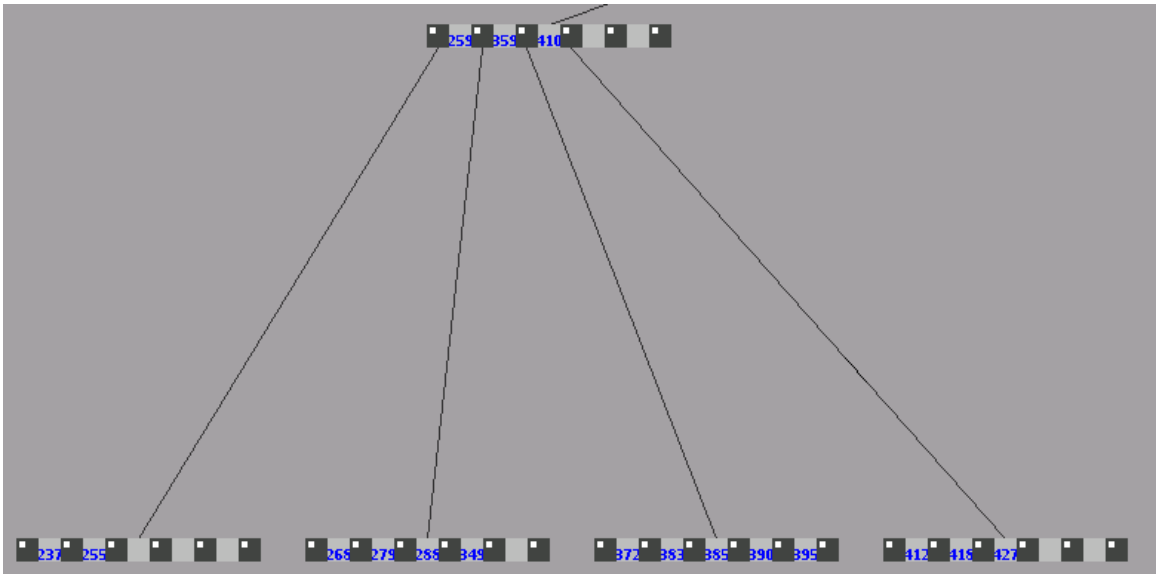
1.1 Caratteristiche

Dati memorizzati	in qualunque nodo
In inserimento, split	1 x 1 → 2 x ½
In cancellazione, split	2 x ½ → 1 x 1

I BTree sono alberi bilanciati di ricerca, i cui nodi possono avere anche parecchi figli, da poche decine a diverse centinaia, a seconda dal valore del *grado minimo*, o *fattore di ramificazione* dell'albero che può variare da 3 a molte centinaia.

Un nodo che contiene n chiavi avrà n+1 figli.

Le chiavi sono sostanzialmente punti di divisione: dividono l'intervallo di chiavi gestite dal nodo in n+1 sotto-intervalli, dove ciascun sotto-intervallo è gestito da un figlio.



Il grado minimo *t*, o fattore di ramificazione, definisce i limiti superiori e inferiori del numero di chiavi presenti in un nodo, con la seguente regola:

Tipo Nodo	***Limite inferiore***	***Limite superiore***
Radice	1	2 * grado - 1
Ogni nodo diverso dalla radice	grado - 1	2 * grado - 1

Nel caso in cui il numero di elementi in un nodo è uguale al limite inferiore/superiore il nodo stesso si dice magro/pieno.

I nodi a profondità massima vengono chiamati *foglie*, tutte le foglie sono alla stessa profondità che coincide con l'altezza dell'albero.

In un BTree con *n* chiavi il più lungo percorso dalla radice ad una foglia qualsiasi è al più:

$$log_t\left(\frac{n+1}{2}\right)$$

Il bilanciamento è garantito dal metodo di inserimento e cancellazione delle chiavi ed è molto importante. Infatti se consideriamo che un nodo può essere contenuto nella memoria primaria e che per leggere un altro nodo bisogna effettuare una lettura in memoria esterna, il bilanciamento aiuta ad ottimizzare lo spazio contenuto in un nodo risparmiando il costo di un'eventuale lettura su un albero sbilanciato.

1.2 Costo delle operazioni

Poichè la visita di un nodo in un BTree richiede un accesso alla memoria secondaria, il numero di nodi visitati durante un'operazione forniscono una misura dei costi che è quasi sempre proporzionale all'altezza.

1.3 Altezza di un BTree

Considerando un albero minimo, ad eccezione della radice, ogni nodo in un BTree ha al più *t-1* discendenti diretti, mentre la radice ne ha almeno 2. Così che il numero di nodi è in relazione con l'altezza nel seguente modo:

Altezza <u>h</u>	***Numero di nodi***
0	1
1	$2t^0$
2	$2t^1$
3	$2t^2$
....	....
h	$2t^{h-1}$

Quindi il numero totale di chiavi è al più:

$$n = 1 + \sum_{i=1}^{h} 2\,t^{i-1}(t - 1)$$

$$\left(\frac{n+1}{2}\right) \geqslant t^h$$

da cui: $$h \leqslant log_t\left(\frac{n+1}{2}\right)$$

Da ciò possiamo dedurre che in un BTree con grado minimo t=50 e circa 1.000.000 di record, una chiave può essere cercata con al più 4 accessi al disco! [1]

1.4 Operazioni Base

Le operazioni effettuate in un BTree rispecchiano quelle di un qualunque albero binario con, in aggiunta, la gestione dei nodi magri e pieni.

Si possono schematizzare nel seguente modo:

1. Creazione di un BTree
2. Inserimento di una chiave
   - Gestione dei nodi pieni (operazione di SplitChild)
3. Eliminazione di una chiave
   - Gestione dei nodi magri (operazione di Merge)
4. Ricerca di una chiave
5. Visita del BTree

1.5 Divisione di un nodo – SplitChild

La procedura divide un nodo *x* pieno, all'altezza della mediana, in due nodi magri.

La chiave mediana viene spostata nel padre del nodo *x*, che naturalmente non deve essere pieno, in modo tale si possa identificare il punto di divisione dei due nuovi sotto-alberi.

Se il nodo *x* è la radice, allora l'altezza dell'albero aumenta di una unità.

N.B. L'operazione di Split è un'operazione di crescita degli alberi.

1.6 Inserimento

Quando si vuole inserire una chiave in un BTree il concetto è semplice. Se durante lo scorrimento dell'albero per trovare il nodo interessato, viene incontrato un nodo pieno avviene uno split (divisione).

Fasi:

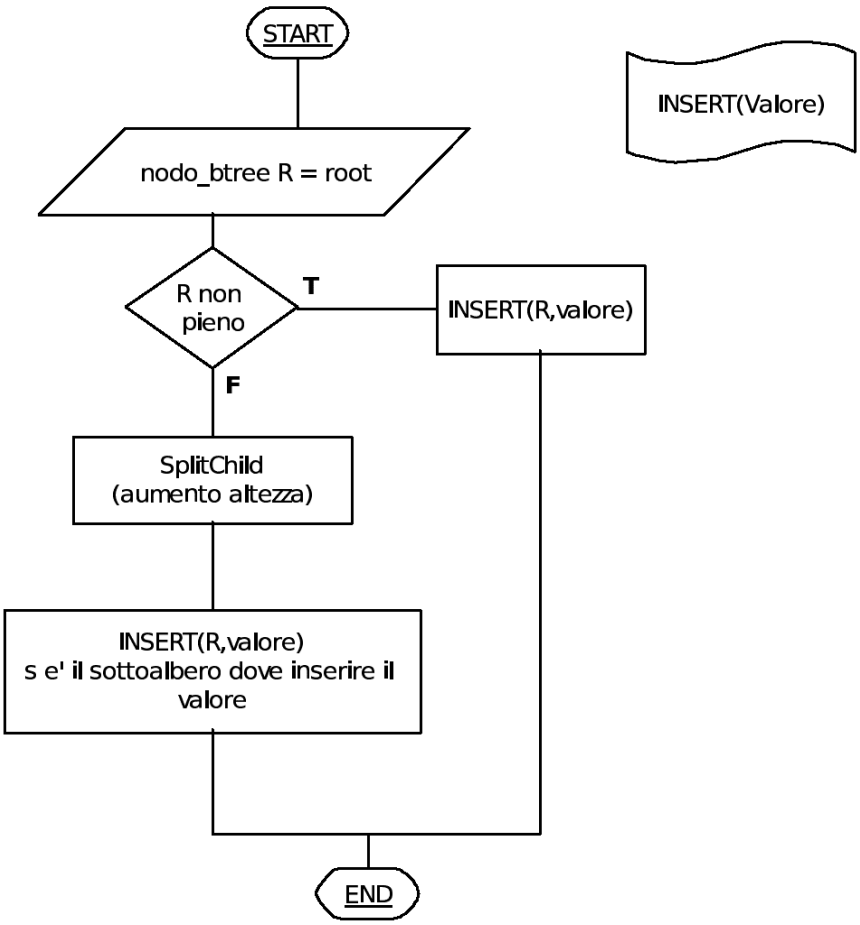
1) Si comincia scorrendo l'albero dalla radice, se è piena allora avviene uno split con un nuovo nodo (vuoto) e viene chiamata ricorsivamente la procedura di inserimento facendola partire dal sottoalbero dove dovrebbe essere inserita la chiave.

   N.B. In questa fase viene incrementata l'altezza dell'albero.
2) Viene controllato se il nodo di partenza è foglia, se lo è viene inserita la chiave all'interno del nodo.
3) Altrimenti, se il valore non è presente nel nodo e la radice del corrispondente sottoalbero è piena, viene eseguito lo Spit della radice del sottoalbero.
4) Viene chiamata ricorsivamente la funzione di inserimento facendola partire dal sottoalbero corrispondente.

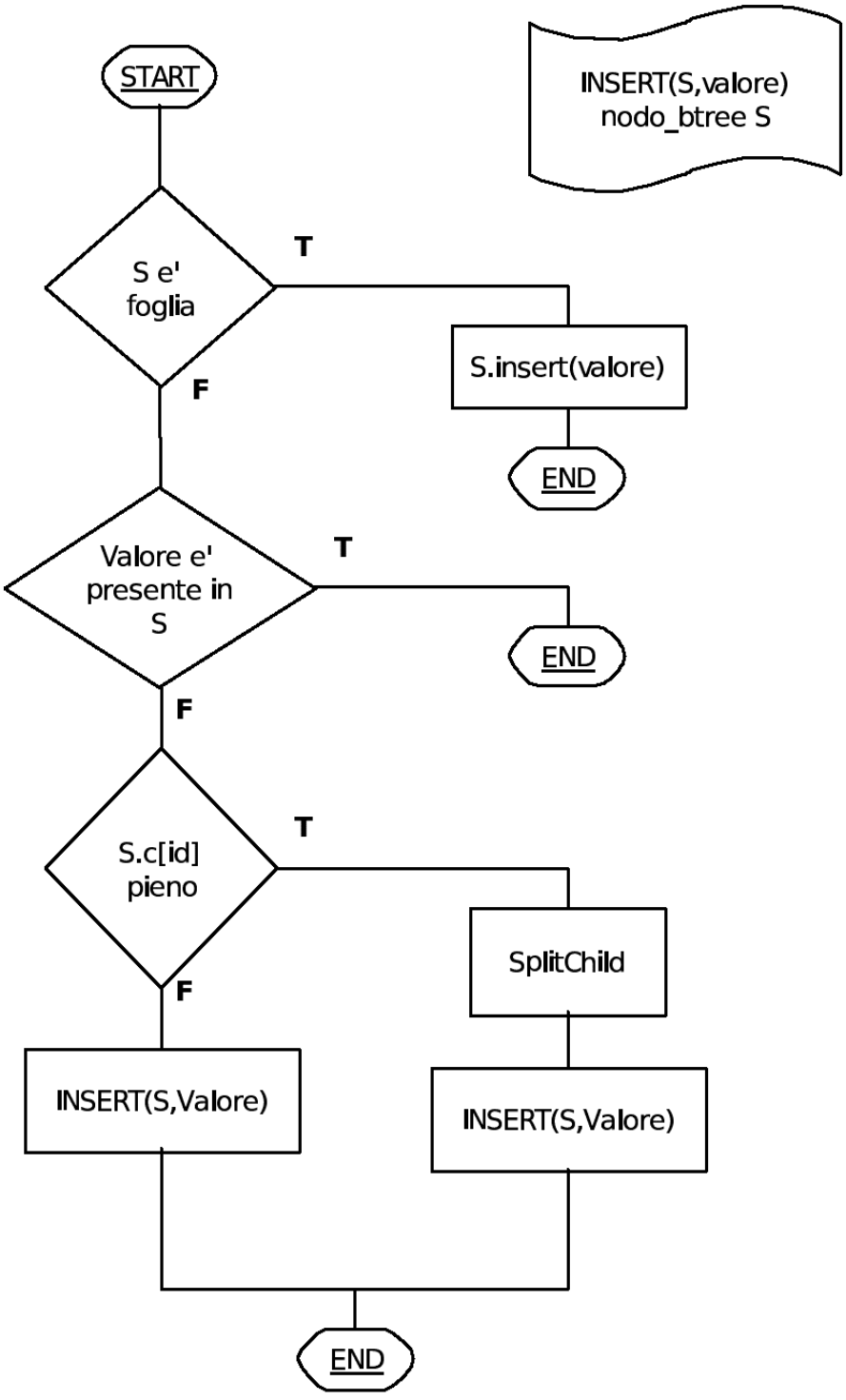
Al più la funzione di inserimento viene richiamata h volte.

~~~~~~~~~~~~~~~~~~~~ Procedura di inserimento ~~~~~~~~~~~~~~~~~~~~

(parte 1)



(parte 2)



~~~~~~~~~~~~~~~~~ Fine Procedura di inserimento ~~~~~~~~~~~~~~~~

1.7 Fusione dei nodi – Merge

La fusione tra due nodi avviene quando i nodi interessati sono magri, così che il nodo ottenuto dalla fusione risulti pieno.

1.8 Cancellazione

Analogamente all'operazione di inserimento, quando viene incontrato un nodo magro si cerca di effettuare un merge con il fratello oppure una donazione dal fratello.

Naturalmente la donazione tra fratelli nodi interni avviene coinvolgendo anche i sottoalberi interessati.

Casi:

Nei casi 2a, 2b, 2c la chiave da cancellare è in un nodo interno magro.

- case_2a:
  - Il nodo ha un fratello dx non magro. Avviene quindi uno scambio di chiavi che porta l'aggiornamento della chiave nel padre e l'incremento del numero di chiavi nel nodo interessato.
- case_2b:
  - Analogo al case_2a con la differenza che il nodo ha un fratello sx non magro.
- case_2c
  - Avviene quando entrambi i nodi sono magri (viene richiamato un merge)

Nei casi 3a, 3b la chiave non è nel nodo preso in considerazione durante la cancellazione e la radice del sottoalbero interessato è magra.

- case_3a
  - avviene una donazione da dx rendendo così la radice del sottoalbero non magra.
- case_3b
  - analogo al case_3a considerando la radice del sottoalbero sx.

Fasi:

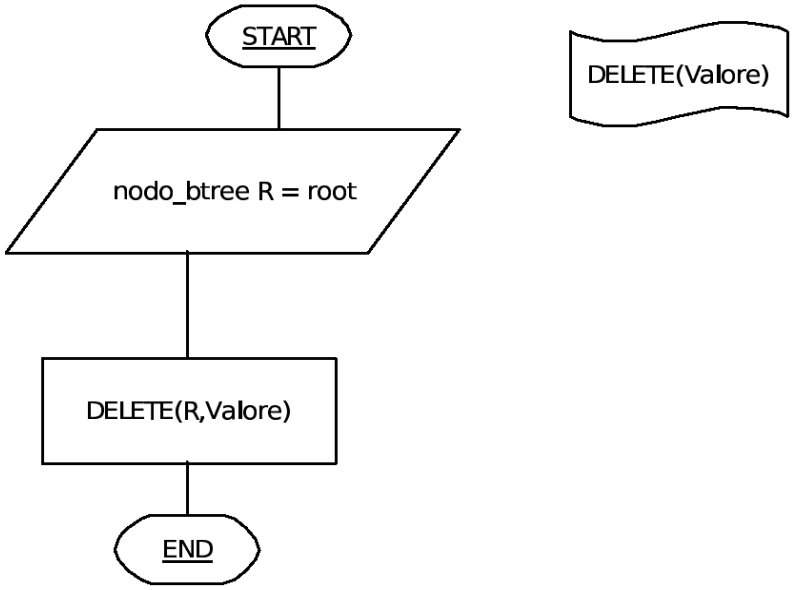
1) Si controlla se il valore da cancellare è presente nel nodo preso in considerazione (nel primo caso è la radice).
2) Se è presente avvengono ulteriori controlli per individuare la funzione da utilizzare (*case_2a*, *case_2b*, *case_2c*)
    1. Nel caso in cui è foglia, non magra e uguale alla radice si effettua una cancellazione del valore in quel nodo.
    2. Altrimenti se la foglia è magra ed il sottoalbero fratello dx non è magro siamo nel *case_2a.*
    3. Invece se il sottoalbero fratello sx non è magro siamo nel *case_2b.*
    4. Altrimenti entriamo nel *case_2c.*
3) Se invece il valore non è presente nel nodo preso in considerazione:
    1. Se non siamo in foglia viene controllato il sottoalbero relativo e se non è magro viene richiamata la funzione di cancellazione facendola partire dal quel sottoalbero.
    2. Altrimenti se il sottoalbero corrispondente ha fratello dx o sx, viene chiamato rispettivamente il *case_3a* o *case_3b* e poi la funzione di cancellazione (ricorsivamente).

    N.B. Se il sottoalbero ha solo il fratello sx siamo in *case_3b* altrimenti se ha solamente il fratello dx siamo in *case_3a*.

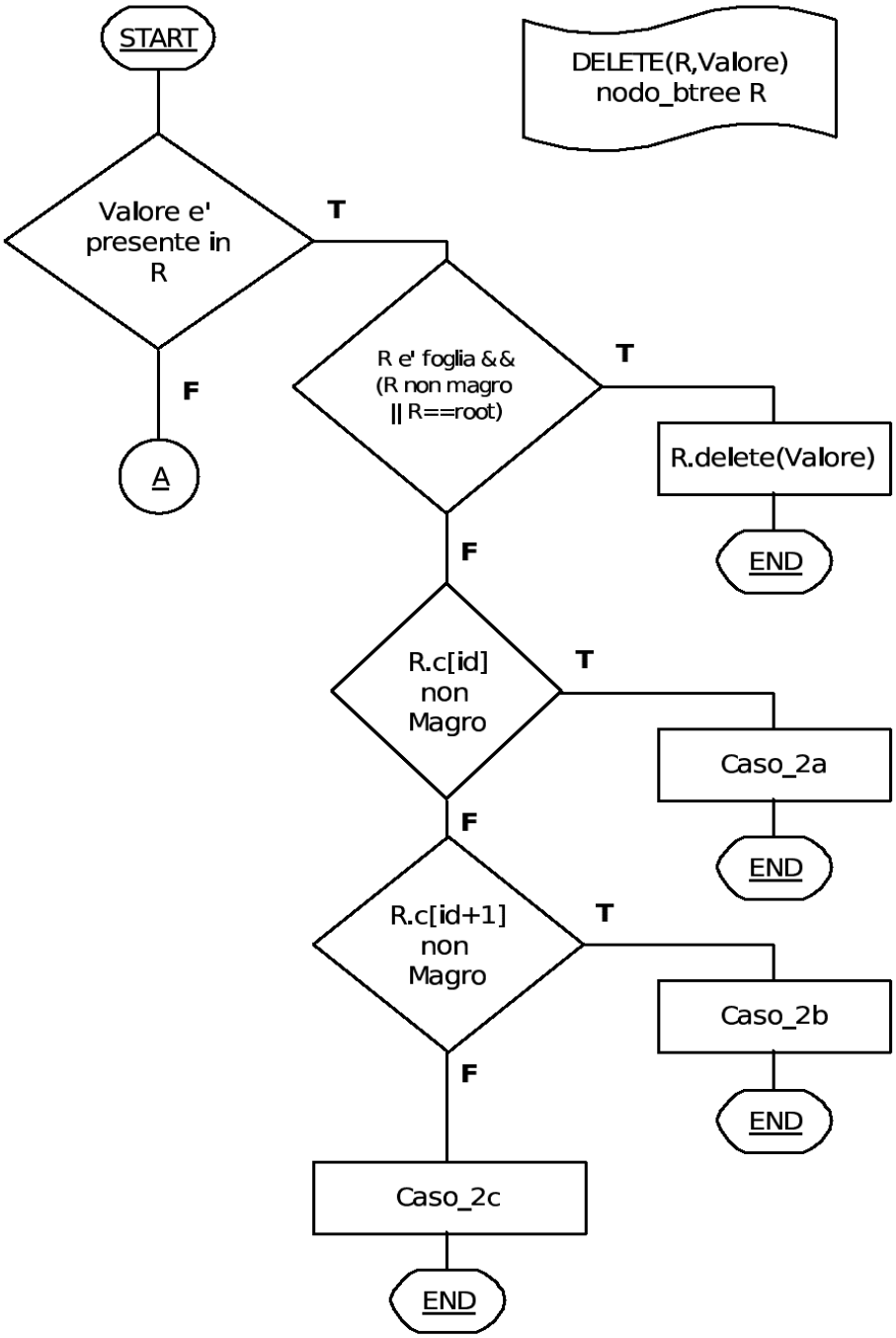
[Per approfondimenti si veda 4]

~~~~~~~~~~~~~~~~~~~ Procedura di Cancellazione ~~~~~~~~~~~~~~~~~~

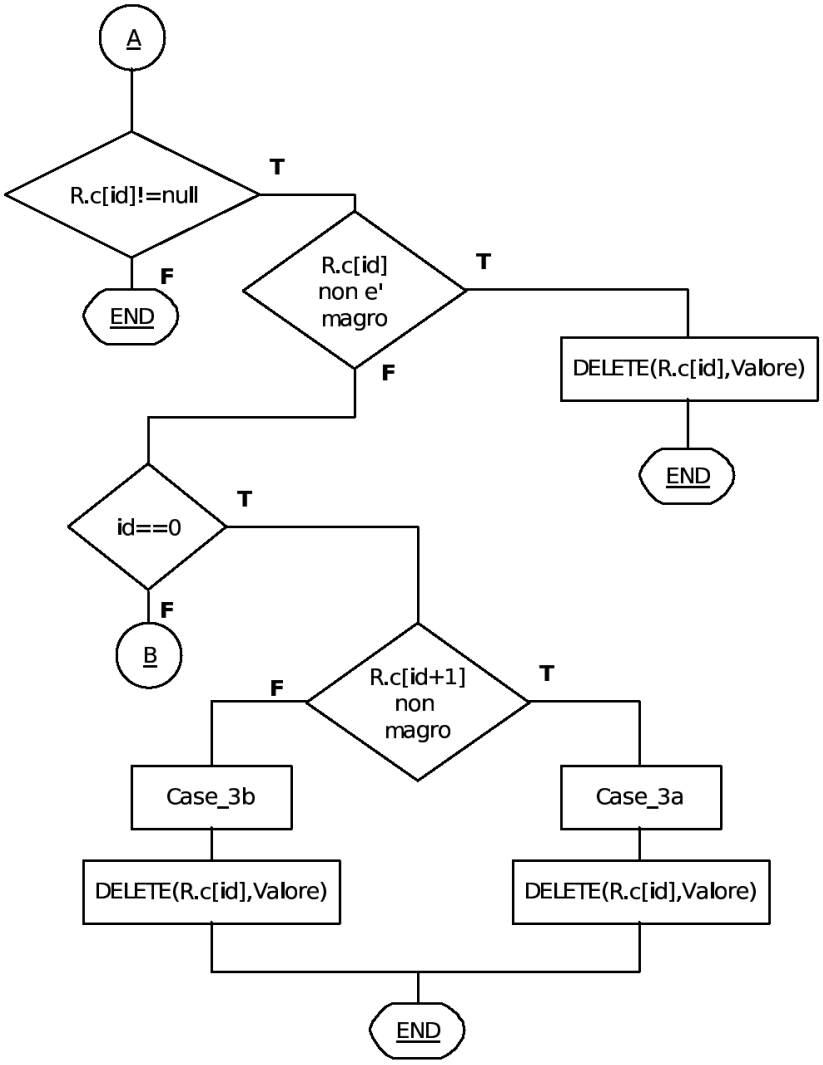
(parte 1)



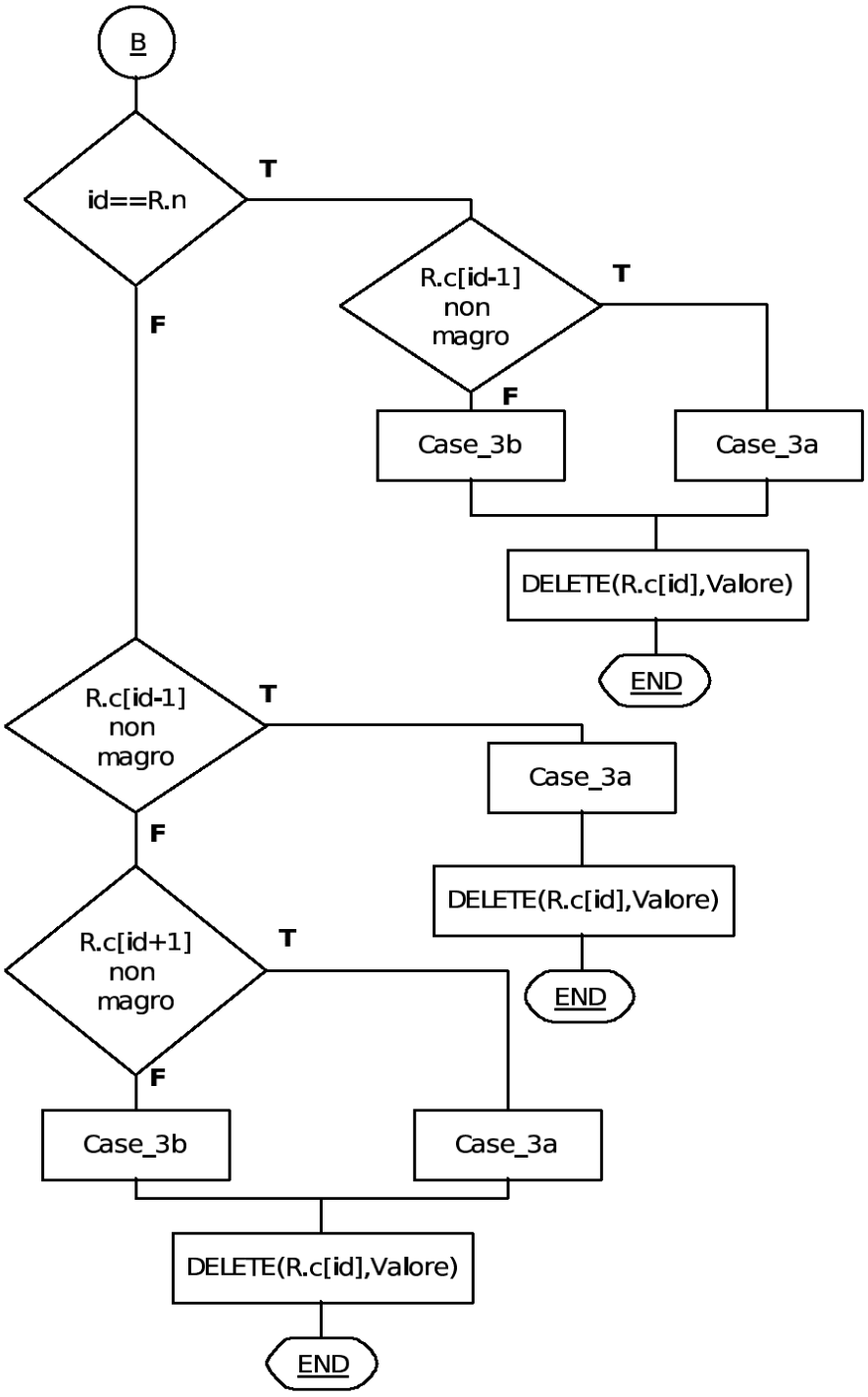
(parte 2)


START
DELETE(R,Valore)
nodo_btree R
Valore e' presente in R
T
F
A
R e' foglia && (R non magro || R==root)
T
R.delete(Valore)
END
F
R.c[id] non Magro
T
Caso_2a
END
F
R.c[id+1] non Magro
T
Caso_2b
END
F
Caso_2c
END

(parte 3)


A
R.c[id]!=null
T
F
END
R.c[id]
non e'
magro
T
F
DELETE(R.c[id],Valore)
END
id==0
T
F
B
R.c[id+1]
non
magro
F
T
Case_3b
Case_3a
DELETE(R.c[id],Valore)
DELETE(R.c[id],Valore)
END

(parte 4)



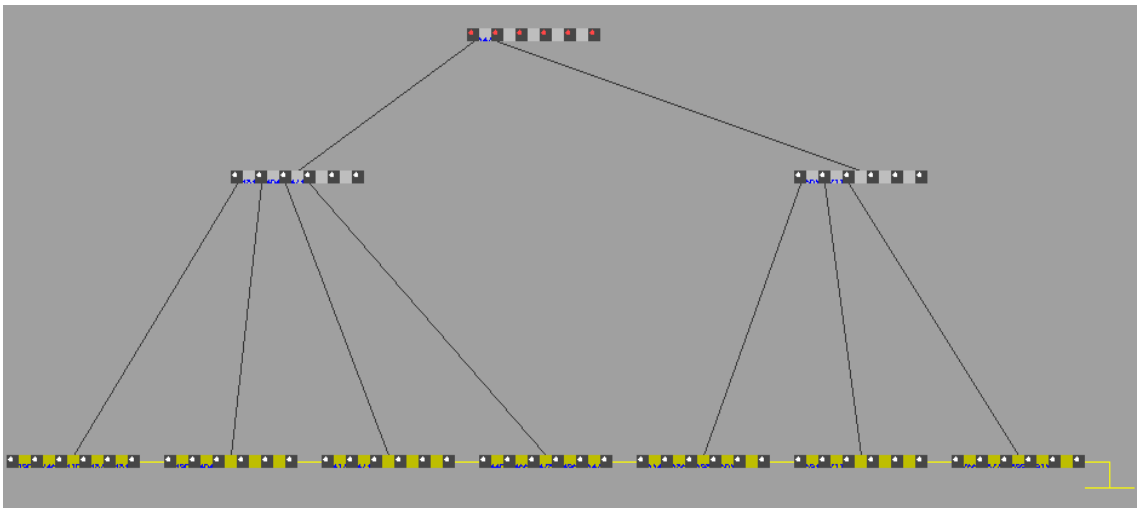
~~~~~~~~~~~~~~~~~ Fine Procedura di Cancellazione ~~~~~~~~~~~~~~~~

2. UN PASSO AVANTI: BTREE+

2.1 Caratteristiche

Dati memorizzati	solamente in foglia
In inserimento, split	1 x 1 → 2 x ½
In cancellazione, split	2 x ½ → 1 x 1

Le caratteristiche sono uguali a quelle dei BTree, ad eccezione del fatto che l'inserimento delle chiavi avviene solamente nei nodi foglia.
I valori inseriti nelle chiavi interne conservano la struttura ad albero del BTree e vengono creati mediante l'operazione di Split.
Ogni nodo foglia possiede un puntatore alla foglia successiva per diminuire il tempo di accesso sequenziale. Naturalmente il puntatore dell'ultima foglia dell'albero punterà a NULL.
Avremo quindi una struttura ad albero bilanciato che permette di effettuare la ricerca in tempo $O(h)$,con h altezza del BTree+, e con la possibilità di accedere alla foglia successiva con 1 solo accesso.



2.2 Altezza di un BTree+

Un UpperBound sull'altezza di un BTree+ avente *n* foglie può essere calcolato nel modo seguente:

$$n \geqslant 2\,t^{h-1}$$

$$\frac{n}{2} \geqslant t^{h-1}$$

$$\log_t \frac{n}{2} \geqslant h-1$$

$$h \leqslant \lfloor \log_t (\frac{n}{2}) + 1 \rfloor$$

$$h = O\left(\log_t \frac{n}{2}\right)$$

dove *t* è il grado minimo del BTree+.

Il numero di nodi interni *m* sarà al più:

$$m \leqslant \sum_{i=0}^{h-1} (2\,t)^i$$

$$m \leqslant \frac{(2\,t)^h - 1}{2\,t-1}$$

2.3 Operazioni Base

Le operazioni effettuate in un BTree+ rispecchiano quelle del BTree ad eccezione della diversa procedura di divisione e unione dei nodi.

Si possono schematizzare nel seguente modo:

1. Creazione di un BTree
2. Inserimento di una chiave
   - Gestione dei nodi pieni (operazione di SplitChild)
   - Gestione dei nodi foglia pieni (operazione di SplitChildLeaf)
3. Eliminazione di una chiave
   - Gestione dei nodi magri (operazione di Merge)
   - Gestione dei nodi foglia magri (operazione di MergeLeaf)
4. Ricerca di una chiave
5. Visita del BTree

2.4 Divisione dei nodi – SplitChild & SplitChildLeaf

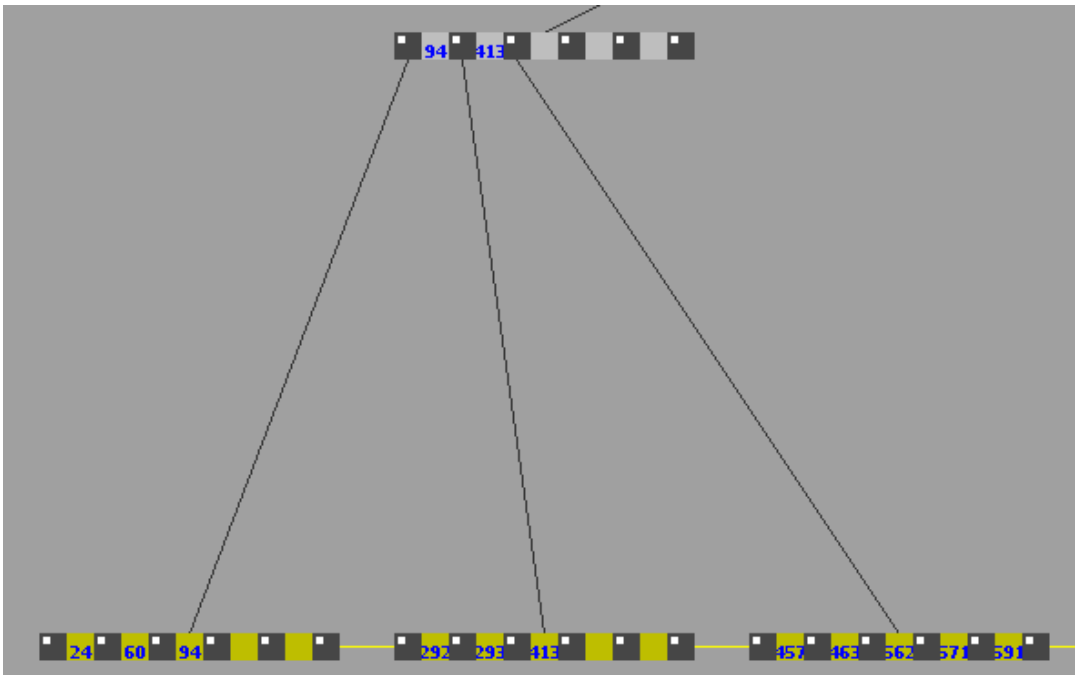
L'operazione (SplitChild) prevede la divisione di un nodo interno pieno in due nodi interni magri. È ereditata da BTree.

L'operazione (SplitChildLeaf) prevede la divisione di un nodo foglia pieno in due nodi foglia. Utilizza SplitChild e poi copia la chiave "mediana" nel figlio di sx.

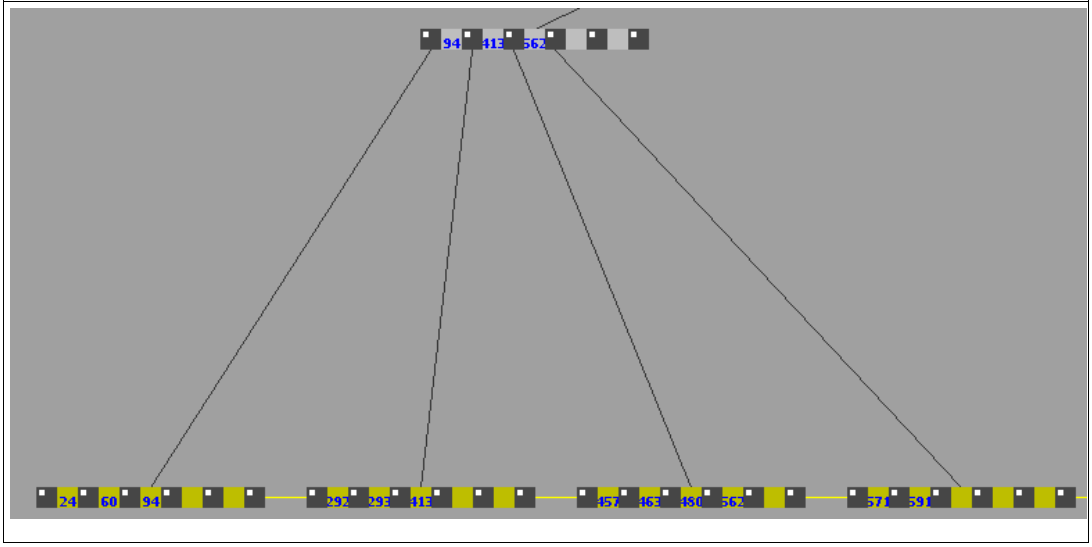
N.B. L'operazione di SplitChildLeaf può non produrre due figli magri.

(In questo esempio il fattore di ramificazione è uguale a 3)

Prima dello SplitChildLeaf



Dopo l'inserimento del valore 480



2.5 Inserimento

Dato che tutte le chiavi devono essere inserite in foglia, viene percorso l'albero dalla radice fino alla foglia.

Se durante lo scorrimento dell'albero troviamo:

1. Un nodo interno pieno, viene eseguito lo SplitChild
2. Se invece il nodo pieno è anche la radice dell'albero, viene eseguito lo SplitChild con un nuovo nodo (vuoto), questa operazione aumenta l'altezza dell'albero.

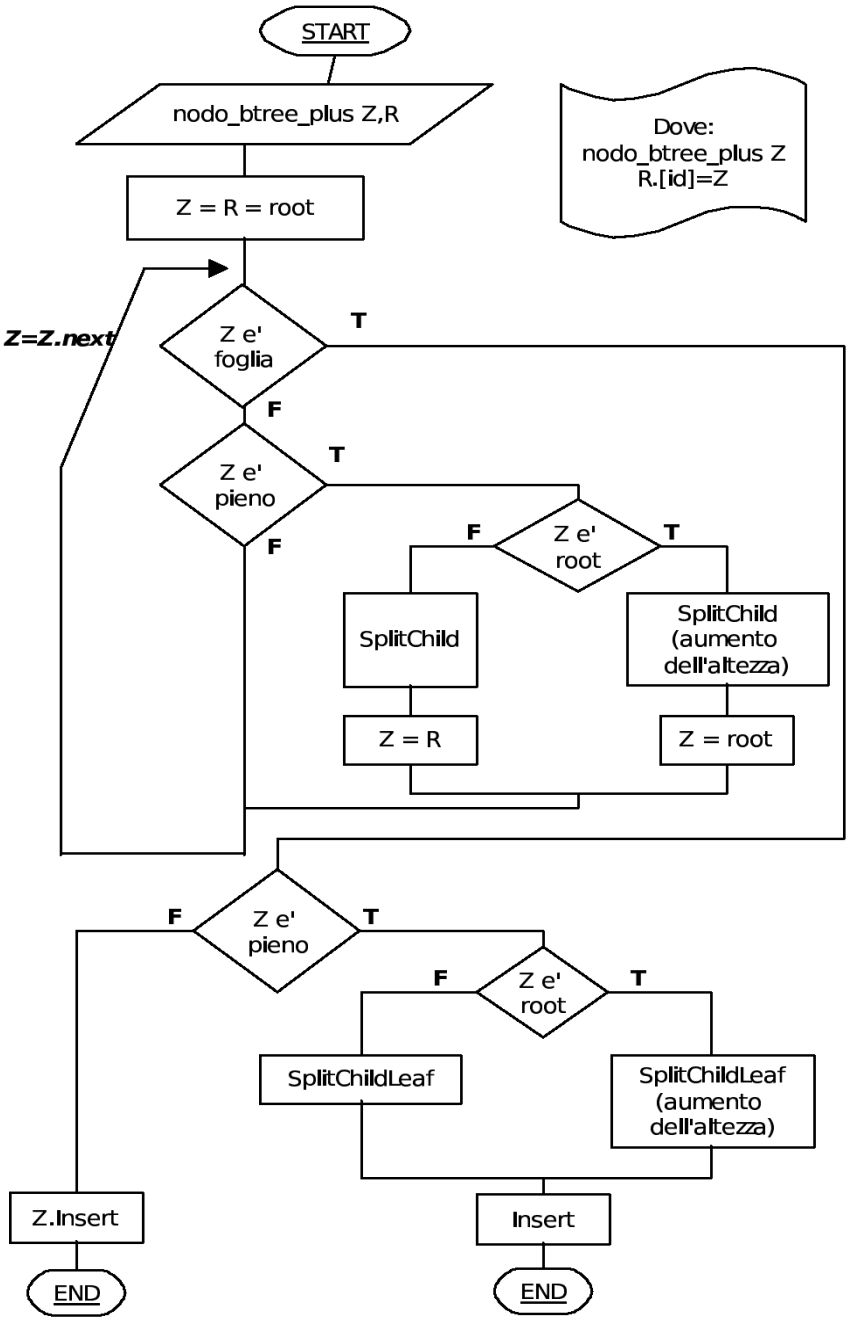
Alla fine dello scorrimento arriviamo certamente ad un nodo foglia.

Se la foglia non è piena la chiave viene inserita, altrimenti:

1. Se il nodo foglia è radice, viene eseguito lo SplitChildLeaf con un nuovo nodo (aumentando anche qui l'altezza dell'albero);
2. Altrimenti viene eseguito lo SplitChildLeaf tra il genitore del nodo foglia e la foglia stessa.

La chiave viene inserita quando si giunge al nodo foglia corrispondente.

~~~~~~~~~~~~~~~~~~~~ Procedura di inserimento ~~~~~~~~~~~~~~~~~~~~



~~~~~~~~~~~~~~~~~~ Fine procedura di inserimento ~~~~~~~~~~~~~~~~~

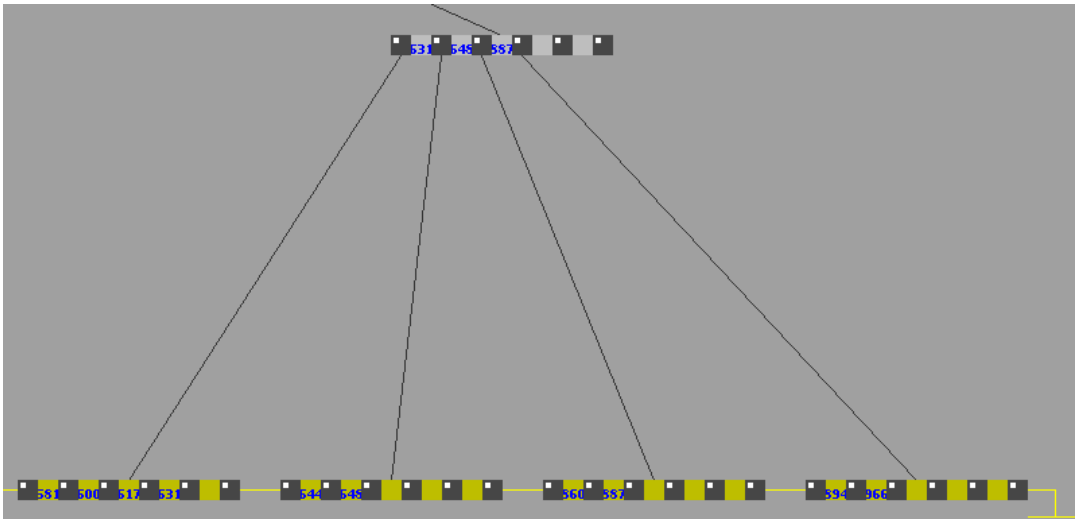
2.6 Fusione dei nodi – Merge & MergeLeaf

La procedura di fusione dei nodi interni(Merge) è ereditata dal BTree.

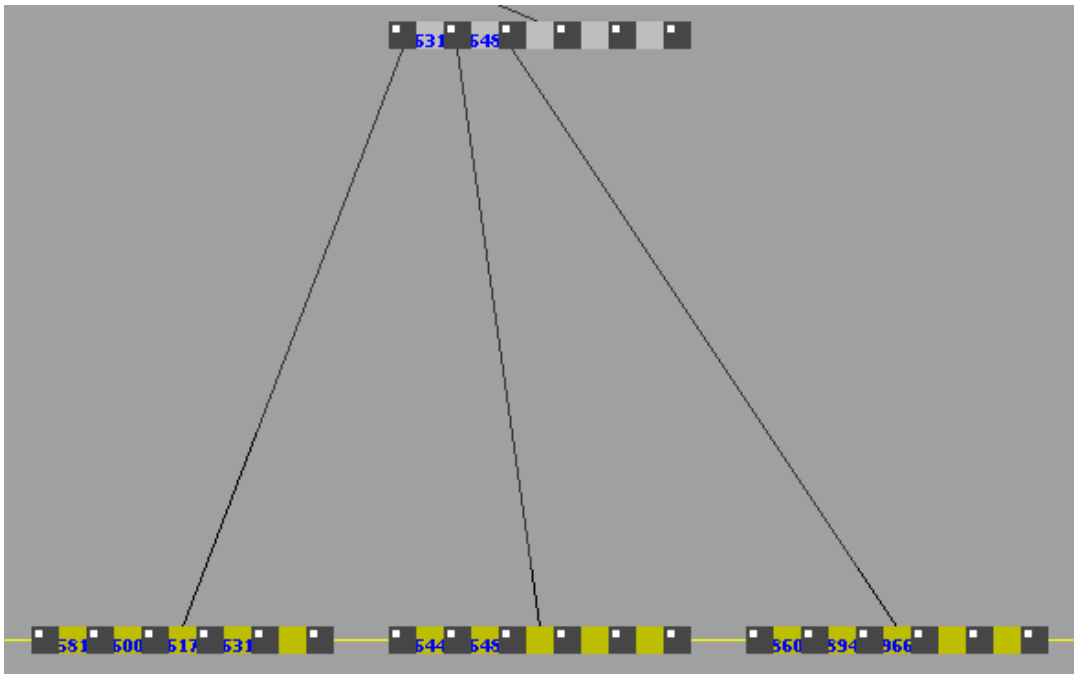
La fusione dei nodi foglia(MergeLeaf) consiste nel comprimere gli elementi inseriti nei due nodi foglia magri in una foglia.

(In questo esempio il fattore di ramificazione è uguale a 3)

Prima del MergeLeaf



Dopo la cancellazione del valore 887



2.7 Cancellazione

Come nell'inserimento viene percorso l'albero fino a trovare la foglia che dovrebbe contenere la chiave.

Durante lo scorrimento dell'albero possiamo trovare un nodo interno magro:

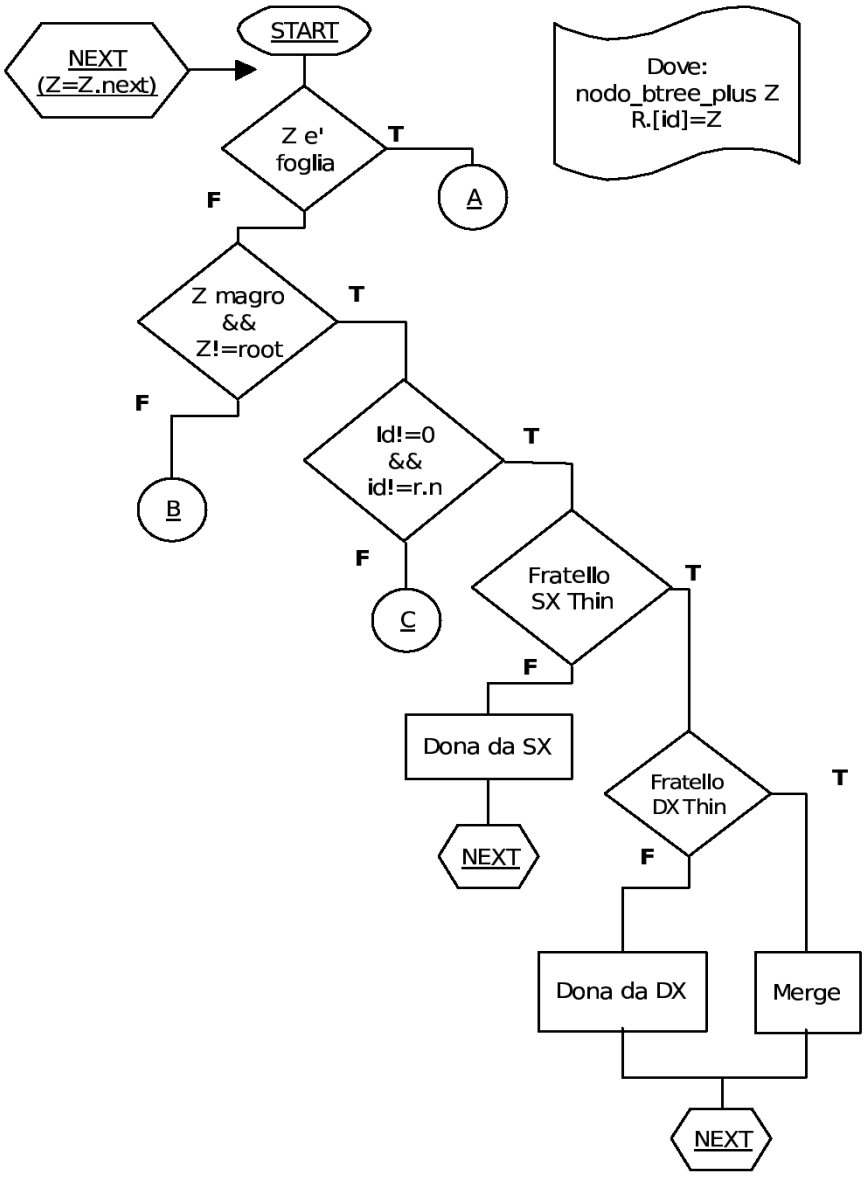
1. Se uno dei fratelli del nodo non è magro allora viene donato un elemento (con i corrispettivi sottoalberi).
2. Se entrambi i fratelli del nodo sono magri avviene una fusione (Merge).

Quando viene raggiunto il nodo foglia, se è magro e:

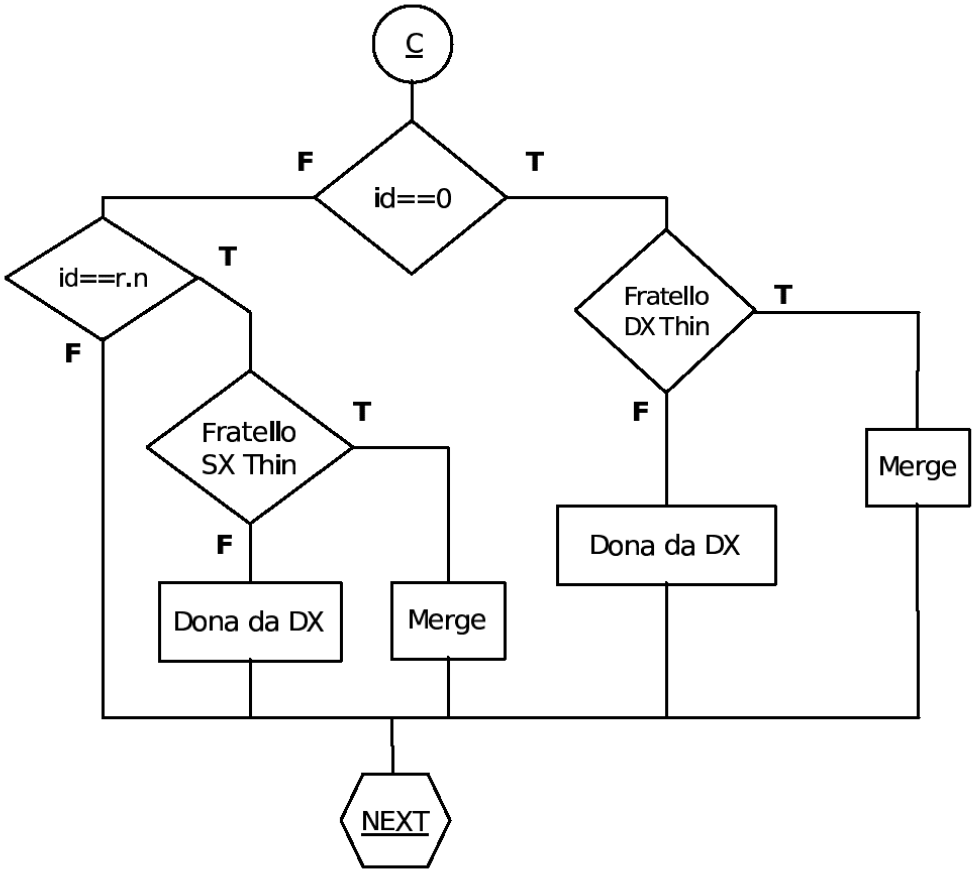
1. Se è la radice si elimina la chiave (albero vuoto!).
2. Se il nodo foglia fratello dx non è magro, viene donata una chiave alla foglia rendendola così non magra e poter quindi effettuare la cancellazione della chiave.
3. Lo stesso comportamento avviene se il nodo foglia fratello dx è magro ma il nodo foglia sx non lo è.
4. Se sia il fratello sx che quello dx sono magri avviene una fusione tra le foglie (MergeLeaf).

~~~~~~~~~~~~~~~~~~ Procedura di Cancellazione ~~~~~~~~~~~~~~~~~~

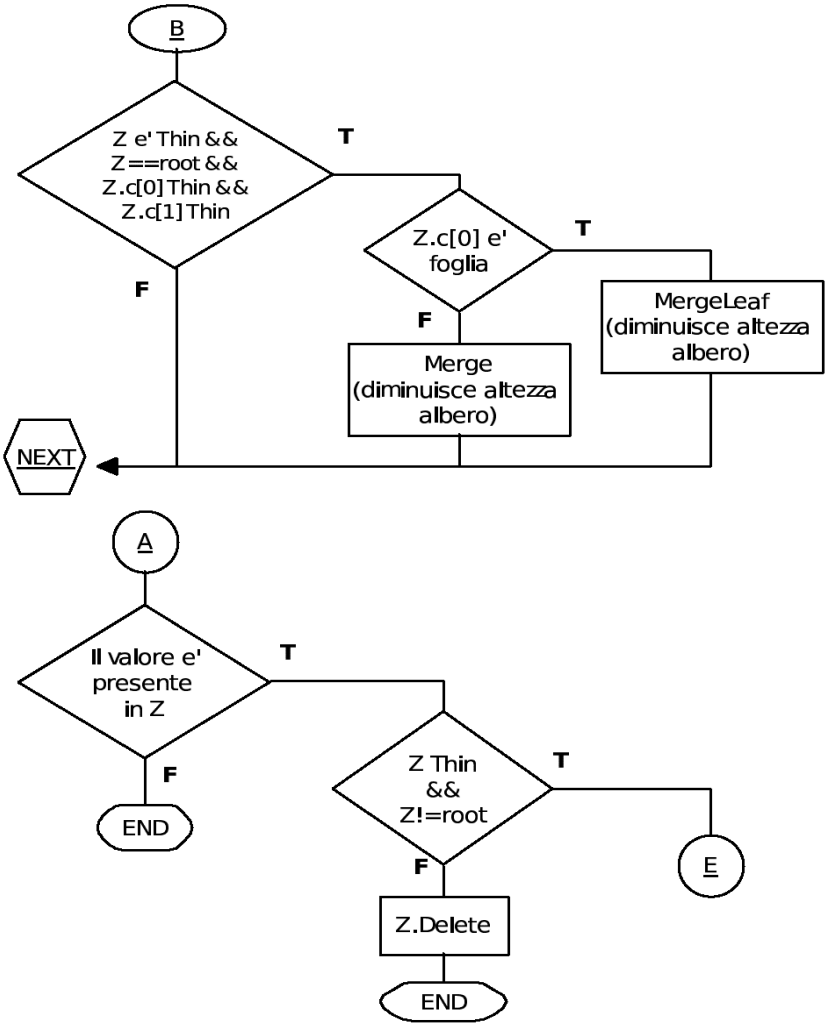
(parte 1)



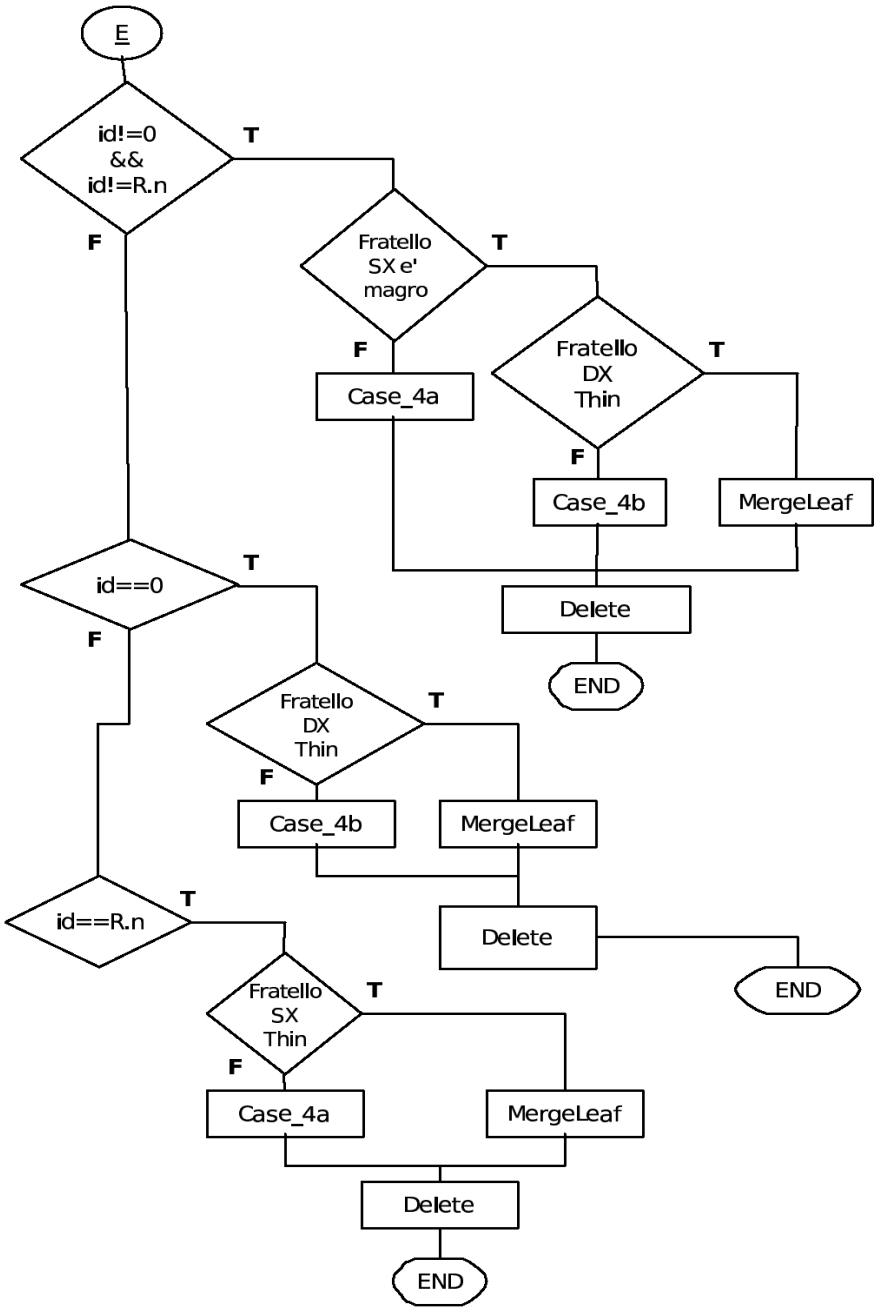
(parte 2)


C
F
id==0
T
id==r.n
T
F
Fratello
SX Thin
T
F
Dona da DX
Merge
Fratello
DX Thin
T
F
Dona da DX
Merge
NEXT

(parte 3)


B
Z e' Thin &&
Z==root &&
Z.c[0] Thin &&
Z.c[1] Thin
T
F
Z.c[0] e'
foglia
T
F
MergeLeaf
(diminuisce altezza
albero)
Merge
(diminuisce altezza
albero)
NEXT
A
Il valore e'
presente
in Z
T
F
END
Z Thin
&&
Z!=root
T
F
E
Z.Delete
END

(parte 4)



~~~~~~~~~~~~~~~~~ Fine Procedura di Cancellazione ~~~~~~~~~~~~~~~~

3. ANALISI E CONFRONTO TRA BTREE e BTREE+

Per ottenere uno studio completo tra le due varianti qui sopra citate, le strutture dati sono state testate con 1.000.000 di inserimenti, utilizzando fattori di ramificazione 50, 70 e 100.

BTree (grado 50)

Numero chiavi	***Numero nodi***	***Altezza albero***
10000	142	3
50000	724	3
100000	1441	3
200000	2903	3
300000	4382	3
450000	6611	4
650000	9478	4
800000	11690	4
1000000	14731	4

BTree (grado 70)

Numero chiavi	***Numero nodi***	***Altezza albero***
10000	108	2
50000	528	3
100000	1034	3
200000	2088	3
300000	3137	3
450000	4671	3
650000	6776	3
800000	8350	3

Numero chiavi	*Numero nodi*	*Altezza albero*
1000000	10421	3

BTree (grado 100)

Numero chiavi	*Numero nodi*	*Altezza albero*
10000	65	2
50000	365	3
100000	722	3
200000	1447	3
300000	2124	3
450000	3398	3
650000	4531	3
800000	5813	3
1000000	7524	3

BTree+ (grado 50)

Numero chiavi in foglia	*Numero foglie*	*Numero nodi interni*	*Altezza albero*
10000	140	3	3
50000	719	9	3
100000	1437	17	3
200000	2892	33	3
300000	4350	65	3
450000	6574	114	4
650000	9427	131	4
800000	11658	141	4
1000000	14635	258	4

BTree+ (grado 70)

Numero chiavi in foglia	***Numero foglie***	***Numero nodi interni***	***Altezza albero***
10000	99	1	2
50000	524	5	3
100000	1037	10	3
200000	2061	19	3
300000	3102	33	3
450000	4698	49	3
650000	6739	65	3
800000	8288	82	3
1000000	10469	123	3

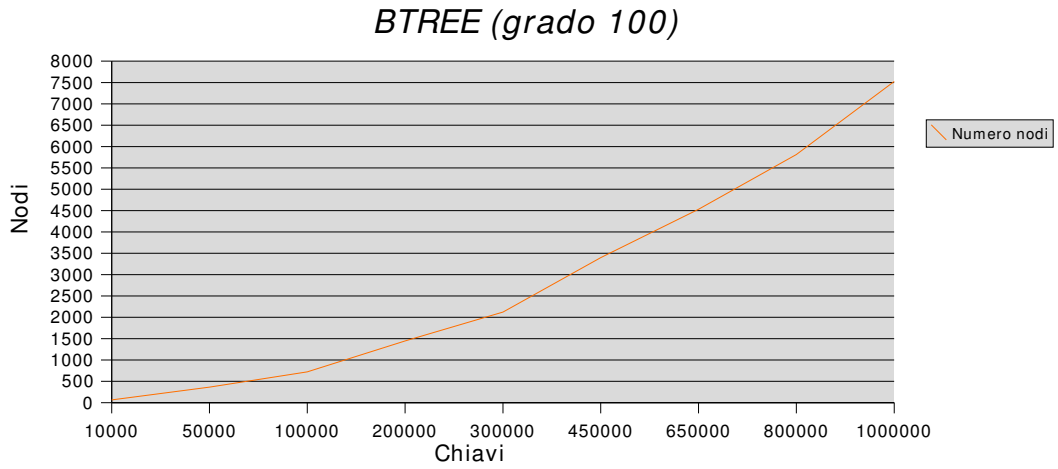
BTree+ (grado 100)

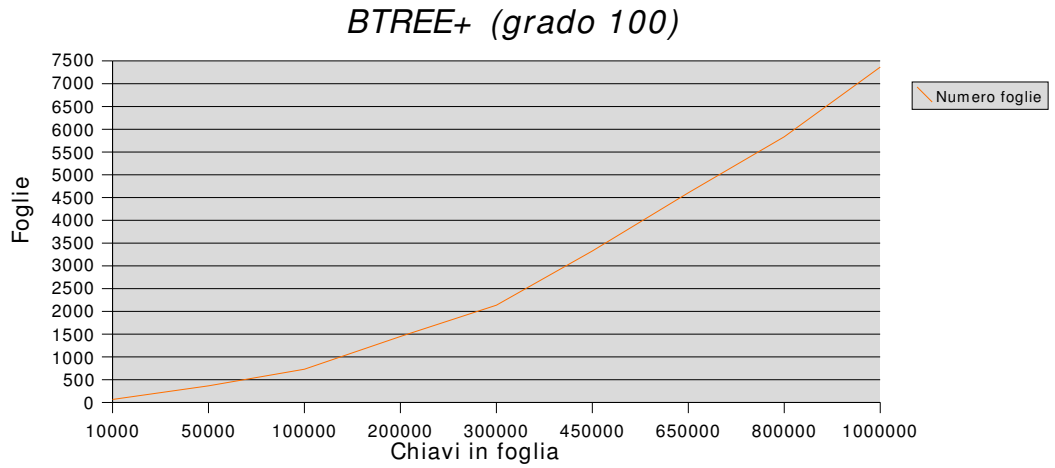
Numero chiavi in foglia	***Numero foglie***	***Numero nodi interni***	***Altezza albero***
10000	67	1	2
50000	364	3	3
100000	730	5	3
200000	1449	9	3
300000	2136	17	3
450000	3327	31	3
650000	4604	33	3
800000	5836	37	3
1000000	7365	65	3

Da queste tabelle risulta evidente che, utilizzando un BTree o un BTree+ con grado 50 in un database con 1.000.000 di chiavi, possiamo effettuare mediamente 4 accessi al disco per accedere alla chiave cercata.

Invece se aumentiamo il grado (per esempio 100) basteranno 3 accessi al disco.

Graficamente:





Risultati

Dai risultati ottenuti, nel caso del BTree+ viene creato un numero di foglie approssimativamente uguale al numero di nodi creati nel caso del BTree. C'è da precisare che al numero di foglie del BTree+ si deve aggiungere il numero di nodi interni, non trascurabile in termini di spazio occupato. Non dimentichiamo pero' che nel BTree+ l'accesso ai dati sequenziale ha un costo minimale, invece nel BTree costa molti accessi in più al disco.

4. UN ULTERIORE PASSO AVANTI BTREE++

4.1 Caratteristiche

Dati memorizzati	solamente in foglia
In inserimento, split	2 x 1 → 3 x 2/3 * (2 * grado - 1)
In cancellazione, split	3 x 2/3 * (2 * grado - 1) → 2 x 1

Il BTree++ nasce dall'unione delle varianti:

- BTree*, da cui eredita le regole di riempimento del nodo [si veda 6]
- BTree+, da cui eredita l'utilizzo dei puntatori tra le foglie.

Nel BTree* il limite inferiore del nodo è incrementato per diminuire il numero di Split e Merge.

I nuovi limiti per il nodo sono:

Tipo Nodo	***Limite inferiore***	***Limite superiore***
Radice	1	2 * grado - 1
Nodi ad altezza uguale ad 1	grado - 1	2 * grado - 1
Nodi ad altezza > 1	2/3 * (2 * grado - 1)	2* grado - 1

L'altezza di un BTree++ segue lo stesso UpperBound del BTree+.

4.2 Operazioni Base

Le operazioni effettuate in un BTree++ rispecchiano in linea di massima quelle del BTree* e del BTree+ ad eccezione della diversa procedura di divisione e unione dei nodi.

Si possono schematizzare nel seguente modo:

1. Creazione di un BTree
2. Inserimento di una chiave
   - Gestione nodo pieno root (operazione SplitChildRoot)
   - Gestione dei nodi interni pieni (operazione di SplitChild)
   - Gestione dei nodi foglia pieni (operazione di SplitChildPlusPlus)
3. Eliminazione di una chiave
   - Gestione dei nodi magri ad altezza 1 (operazione di Merge)
   - Gestione dei nodi magri (operazione di Merge_Star)
   - Gestione dei nodi magri foglia (operazione di MergeLeaf_p)
   - Gestione dei nodi magri foglia ad altezza 1(operazione di MergeLeaf)
4. Ricerca di una chiave
5. Visita del BTree

4.3 Divisione dei nodi – SplitChildRoot & SplitChild & SplitChildPlusPlus

- SplitChildRoot è identico a quello di SplitChild di BTree+.
- SplitChild è ereditato da BTree*, divide due nodi interni pieni creandone tre magri.
- SplitChildPlusPlus gestisce lo SplitChild di BTree* in modo che possa essere utilizzato tra foglie.

4.4 Inserimento

Poichè il BTree++ eredita la caratteristica principale del BTree+ (le chiavi devono essere inserite in foglia) il concetto dell'inserimento è identico a quello del BTree+: viene percorso l'albero dalla radice fino alla foglia (non piena) ove si inserisce la chiave.

L'unica differenza è nella gestione dei nodi interni con le diverse operazioni di Split, poichè si possono verificare i seguenti “casi speciali”:

1. Quando il nodo è sia radice che foglia piena. In tal caso viene richiamato lo SplitChildRoot ereditato da BTree+ che genera due foglie magre (oltre naturalmente alla radice con un elemento solamente).
2. Nel caso in cui viene incontrato un nodo interno magro cerchiamo di capire se i fratelli vicini possono donare qualche elemento in modo da ritardare lo split. Se i due fratelli piu vicini (dx o sx se esistono) sono magri allora avviene lo SplitChild ereditato da BTree*.
3. Nel caso in cui il nodo è una foglia, cerchiamo di capire se possono essere fatte delle donazioni dalle le foglie sorelle. Nel caso in cui sono magre allora viene effettuato lo SplitChildPlusPlus.

4.5 Fusione dei nodi – Merge & MergeStar & MergeLeaf_p & MergeLeaf

- Merge è ereditato da BTree, gestisce il caso speciale che avviene quando esistono due nodi interni magri ad altezza 1.
- Merge_Star gestisce il merge tra nodi interni. Crea due nodi pieni da tre nodi magri.
- MergeLeaf gestisce la fusione di tre foglie magre in due piene.
- MergeLeaf_p ereditato da BTree+, gestisce il caso speciale che avviene quando esistono due foglie magre ad altezza 1.

4.6 Cancellazione

La procedura di cancellazione è simile alla cancellazione del BTree+ con l'evidente differenza della gestione dei diversi “casi speciali” per il merge dei nodi interni/foglia:

1. Nel caso in cui la radice ha solamente due foglie magre, viene richiamato MergeLeaf_p (sostanzialmente il MergeLeaf di BTree+).
2. Invece se la radice ha solamente due nodi interni magri, viene eseguito il Merge ereditato da BTree.
3. Se durante lo scorrimento viene incontrato un nodo magro interno si tenta di donare, al nodo magro, qualche elemento dai vicini in modo da ritardare la fusione. Se questo non può avvenire allora viene effettuato il Merge_Star tra i nodi interni.
4. Nel caso in cui siamo in foglia valgono i tre casi precedenti utilizzando però MergeLeaf al posto di Merge_Star.

5. BIBLIOGRAFIA

[1] Douglas Comer – The Ubiquitous B-Tree – Computing Surveys 1979

[2] Peter Bishop – L'informatica seconda edizione – Jackson Libri 1998

[3] Alfred V.Aho, Jeffrey D.Ullman – Fondamenti di informatica – Zanichelli 1994

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest – Introduzione agli algoritmi vol. 2 – Jackson Libri 1994

[5] Peter Neubauer – B-Trees: Balanced Tree Data Structures
http://www.bluerwhite.org/BTree/

[6] Todaro Michelangelo – Dai BTree ai BTree, Sviluppo e Analisi delle Prestazioni – Relazione Progetto Finale, Università di Catania A.A. 2004-2005*