Lezione_22_DeA

6.6.3. Implementazione dei metodi della Mappa: `get`

Metodo `get(k)`

```
w <- MWTreeSearch(k, T.root());
if(T.isExternal(w)) then{
     return null;
}
else{
     trova e∈w tale che e.getKey() = k;
     return e.getValue();
}
```

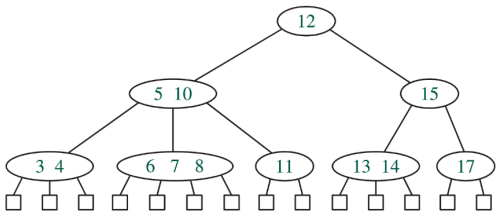
La *complessità* è dominata da quella di `MWTreeSearch`, ed è quindi $O(d_{max}h)$, dove d_{max} è il massimo numero di figli di un nodo e h è l'altezza dell'albero.

6.6.4. (2,4)-Tree

> Un *(2,4)-Tree* è un MWS-Tree tale che ogni nodo interno è un d-node con $2 \leq d \leq 4$ (quindi ha d figli e $d-1$ entry) e tale che tutte le foglie hanno la stessa profondità.

Osservazione

Si può definire un (2,3)-Tree con $2 \leq d \leq 3$ per il quale si applicano gli stessi algoritmi e le stesse complessità. Il vantaggio del (2,4)-Tree è che si generalizza al B-Tree, che è una struttura dati molti usata per la realizzazione di indici in memoria secondaria (ad esempio nelle basi di dati).



6.6.4.1. Altezza

Proposizione

Un (2,4)-Tree con $n > 0$ entry ha altezza $\Theta(\log n)$.

Il seguente corollario è una conseguenza immediata della proposizione e di quanto visto prima.

Corollario

In un (2,4)-Tree con n entry, la complessità di `MWTreeSearch` e del metodo `get` della mappa sono $\Theta(\log n)$

6.6.4.1.1. Dimostrazione

Sia T un (2,4)-Tree con n entry e altezza h, sia m_i il numero di nodi al livello i, per $0 \leq i \leq h$. Si ha che: $m_0 = 1$, $2 \leq m_1 \leq 4$, $2^2 \leq m_2 \leq 4^2$, $\ldots$, $2^i \leq m_i \leq 4^i$, $\ldots$, $2^h \leq m_h \leq 4^h$.

- So che, per definizione, le foglie sono tutti i nodi del livello h;
- Poiché il (2,4)-Tree è un MWS-Tree, so che il numero di foglie è $n+1$.
  $\implies 2^h \leq m_h = n+1 \leq 4^h \implies h \leq \log_2(n+1) \leq \log_2(4^h) = 2h \implies h \leq \log_2(n+1)$
  e $h \geq \frac{\log_2(n+1)}{2}$
  $\implies h \in \Theta(\log n)$
  Questo conclude la prova della Proposizione.

Il corollario segue immediatamente dall'analisi fatta per il MWS-Tree e dal fatto che per il (2,4)-Tree vale che $h \in \Theta(\log n)$ e $d_{max} \leq 4$.

6.6.5. Implementazione dei metodi della Mappa: `put` e `remove`

6.6.5.1. `put`

L'idea per realizzare il metodo `put(k,x)` consiste nei seguenti punti:

- Se la chiave non è presente, inserisci la nuova entry $e = (k, x)$ in un nodo giusto per k ad altezza 1.
- Se il nodo in cui è stata inserita e va in *overflow* (ovvero ha quattro entry e cinque figli), invoca il metodo *Split*, che *ripristina le proprietà* del (2,4)-Tree sfruttando la flessibilità sul numero di entry ammissibili in un nodo e *propagando*, se

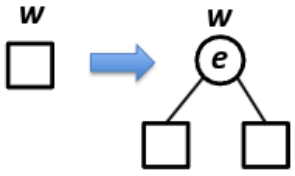
necessario, *l'overflow verso l'alto* che, se arriva alla radice, fa crescere di uno l'altezza dell'albero.

Metodo `put(k,x)`

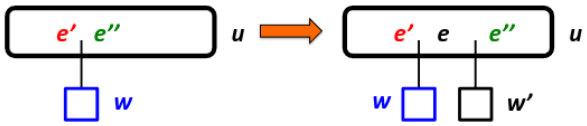
```
w <- MWTreeSearch(k, T.root());
if (T.isInternal(w)) then {
     e <- entry in w con chiave k;
     y <- e.getValue();
     sostituisci x a y in e;
     return y;
}
//Caso in cui w è foglia
e <- (k,x);
if (T.isRoot(w)) then{
     expandExternal(w,e);
}
else{
     u <- T.parent(w);
     inserisci e in u aggiungento una foglia w`;
     if (u è un 5-node) then { //overflow
          Split(u);
     }
}
incrementa il numero di entry in T di 1;
return null;
```

> **Osservazione**
>
> Se w è interno, vengono effettuate $\Theta(1)$ operazioni.

Nel caso in cui w è sia foglia che radice viene chiamato il metodo `expandExternal(w,e)`:

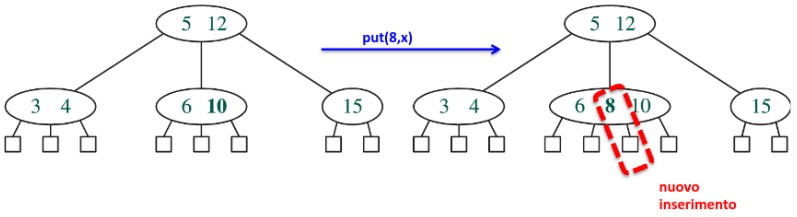


Nel caso in cui w è foglia ma non radice si inserisce e in u, aggiungendo una foglia w':



Nota bene

e' oppure e'' potrebbe non esistere.



6.6.5.1.1. Metodo `Split` (ricorsivo)

Metodo `Split(u)`
Input: 5-node $u \in T$, unica violazione delle proprietà di (2,4)-Tree.
Output: Ripristino delle proprietà di (2,4)-Tree.

Sia $u = (e_1, e_2, e_3, e_4)$ con figli u_1, u_2, u_3, u_4.

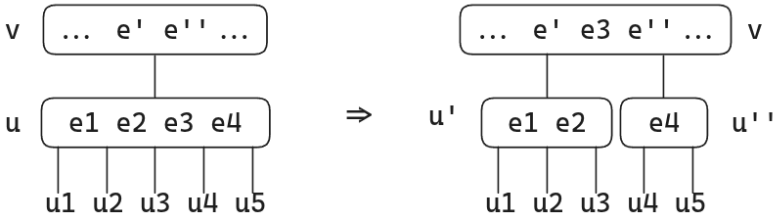
```
Crea due nuovi nodi u`=(e1, e2) con figli u1, u2, u3 e u`` = (e4) con
figli u4, u5;

if (!T.isRoot(u)) then{
     v <- T.parent(u);
     inserisci e3 in v con figlio sinistro u` e figlio destro u``;
     cancella u;
     if(v è un 5-node) then {
          Split(v);
     }
}
else{
     crea una nuova radice contenente e3 e con due figli u`, u`;
     cancella u;
}
```

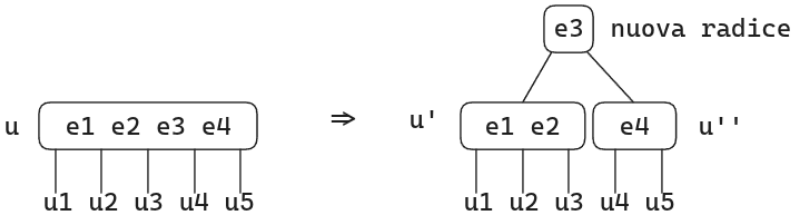
Osservazione

Il primo ramo `if` del merodo ha $\Theta(1)$ operazioni, esclusa l'eventuale chiamata ricorsiva, mentre il ramo `else` fa aumentare l'altezza di T di uno.

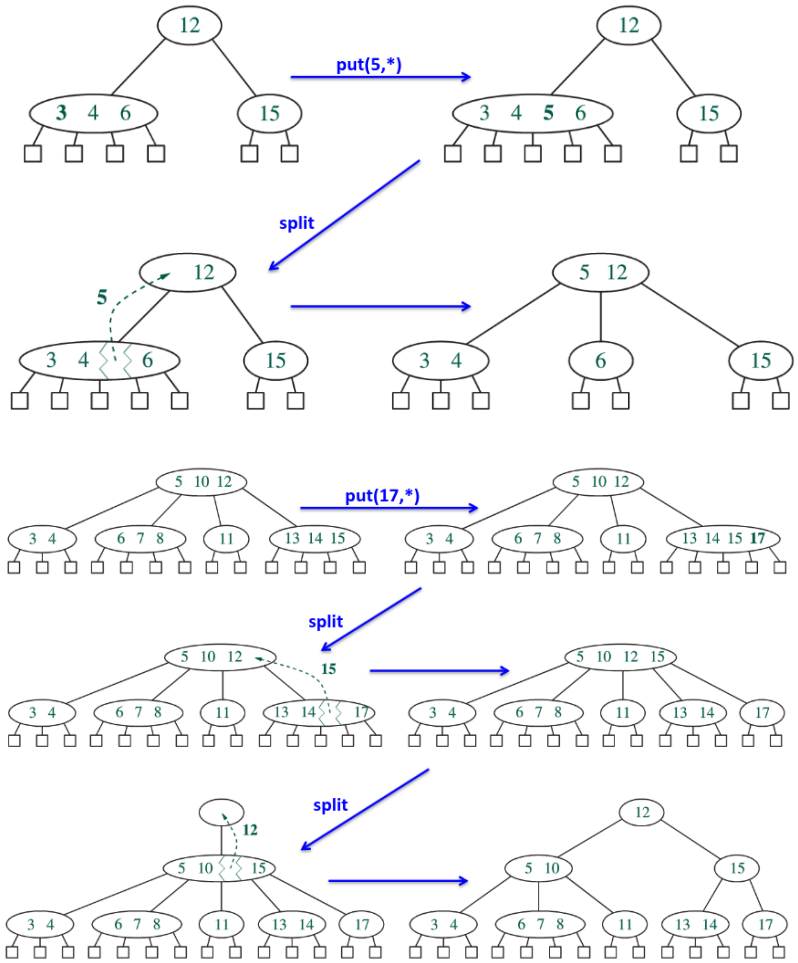
Graficamente, il caso in cui *u non è radice* si rappresenta così:



Mentre il caso in cui *u è radice* si rappresenta così:



6.6.5.1.2. Esempi con overflow



6.6.5.1.3. Complessità

Proposizione

Per una mappa con n entry implementata tramite un (2,4)-Tree, la complessità di `put` è $\Theta(\log n)$.

La complessità è dominata da `MWTreeSearch` e da `Split`.

- `MWTreeSearch` effettua $\Theta(\log n)$ operazioni;
- `Split` è un algoritmo ricorsivo.
  - La prima invocazione avviene su un nodo ad altezza 1;
  - Le successive invocazioni, una per livello sino ad arrivare eventualmente alla radice, eseguono $\Theta(1)$ operazioni per invocazione
    $\implies$ in totale per lo `Split` vengono effettuate $\Theta(\log n)$ operazioni.
    $\implies$ La complessità finale è $\Theta(\log n)$.

6.6.5.2. `remove`

L'idea per realizzare il metodo `remove(k)` consiste nei seguenti punti:

- Si rimuovono sono entry in nodi ad altezza 1;
- Se la entry e da rimuovere è in un nodo ad altezza maggiore di 1, si sostituisce con una entry e' ad altezza 1 e si rimuove quella;
- Se il nodo ad altezza 1 da cui è stata rimossa una entry va in *underflow* (ovvero contiene zero entry), *ripristina le proprietà* del (2,4)-Tree sfruttando la flessibilità sul numero di entry per nodo *propagando*, se necessario, *l'underflow verso l'alto* che, se arriva alla radice, fa diminuire di uno l'altezza dell'albero.

Nota bene

La rimozione di una entry da un nodo e la gestione dell'eventuale underflow sarà effettuata tramite il metodo `Delete`.