

Alberi auto-aggiustanti

Dispensa didattica per il corso di Algoritmi e Strutture Dati
a.a. 2014/2015 – ver. 1.4

Ing. Claudio Mazzariello, Prof. Carlo Sansone

20 Dicembre 2014

A differenza di altre possibili realizzazioni di alberi binari di ricerca, gli alberi auto-aggiustanti (*splay trees*) [1] non necessitano di alcuna condizione esplicita di bilanciamento. L'idea fondamentale degli *splay trees* è che l'albero viene automaticamente riaggiustato ogni volta che si esegue su di esso una qualunque operazione: in particolare, l'elemento cui si accede viene fatto risalire alla radice ed i sottoalberi incontrati lungo il cammino sono riposizionati tramite opportune rotazioni. Mostriamo come, mediante tali rotazioni, si riesca ad ottenere dei tempi di esecuzione ammortizzati di tipo logaritmico su una sequenza di n operazioni.

Si noti che, dal momento che gli elementi cui si accede più di frequente si muovono verso la radice, ad essi si potrà accedere in media più rapidamente rispetto ad altri tipi di alberi binari di ricerca; questo è un indubbio vantaggio in molte applicazioni pratiche.

1 L'operazione *splay*

La strategia di ristrutturazione proposta da Sleator e Tarjan è anche nota come operazione **splay**. Lo **splay** di un nodo x consiste nel partire da x e farlo risalire fino alla radice dell'albero eseguendo una sequenza di rotazioni.

Denoteremo con $x.p$ il padre di x nell'albero. Chiameremo inoltre *nonno* di x , e lo indicheremo con $x.p.p$, il padre di $x.p$. In ciascun passo di *splay*, la rotazione da eseguire viene scelta come segue:

- (i) se x è figlio della radice, ruotiamo su $x.p$ (Figura 1);
- (ii) se x ha un nonno, $x.p.p$, ed x e $x.p$ sono entrambi figlio destri o sinistri del proprio genitore, ruotiamo prima su $x.p.p$ e poi su $x.p$ (Figura 2);
- (iii) se x ha un nonno, $x.p.p$, ed x è figlio sinistro/destro mentre $x.p$ è figlio destro/sinistro del proprio genitore, ruotiamo prima su $x.p$ e poi sul nuovo padre di x (ovvero su $x.p.p$ prima della rotazione) (Figura 3).

Ciascun passo che costituisce l'operazione di **splay** richiede tempo $O(1)$. L'effetto della strategia di ristrutturazione **splay** è quello di far risalire x alla radice, riarrangiando i sottoalberi incontrati lungo il cammino. Per rendere un albero binario di ricerca auto-aggiustante, eseguiamo l'operazione **splay** in corrispondenza di ciascuna operazione di accesso o di modifica dell'albero. In particolare:

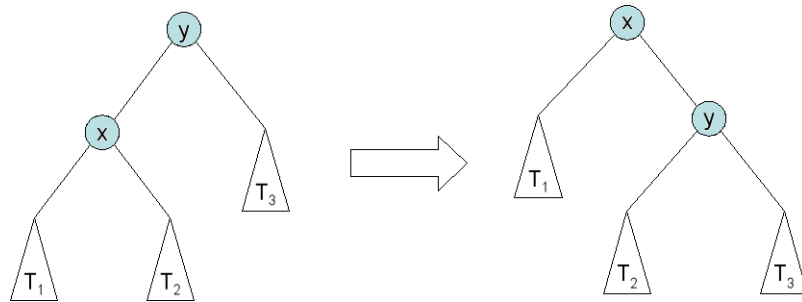


Figura 1: Operazione di **splay** sul nodo x — caso (i)

- dopo aver cercato una chiave, eseguiamo uno **splay** sul nodo che contiene la chiave, o sulla foglia su cui la ricerca è terminata, a meno che l'albero non sia vuoto, nel qual caso non viene ovviamente compiuta alcuna operazione;

SPLAYTREE-SEARCH(T, k)

```

1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and  $k \neq x.key$ 
3      do  $last = x$ 
4          if  $k < x.key$ 
5              then  $x = x.left$ 
6              else  $x = x.right$ 
7  if  $x \neq \text{NIL}$ 
8      then  $splay(T, x)$ 
9      else if  $T.root \neq \text{NIL}$ 
10         then  $splay(T, last)$ 
11 return  $x$ 

```

- dopo aver inserito un elemento, eseguiamo sempre uno **splay** sul nuovo nodo che contiene l'elemento;

SPLAYTREE-INSERT(T, z)

```

1  TREE-INSERT( $T, z$ )
2   $splay(T, z)$ 

```

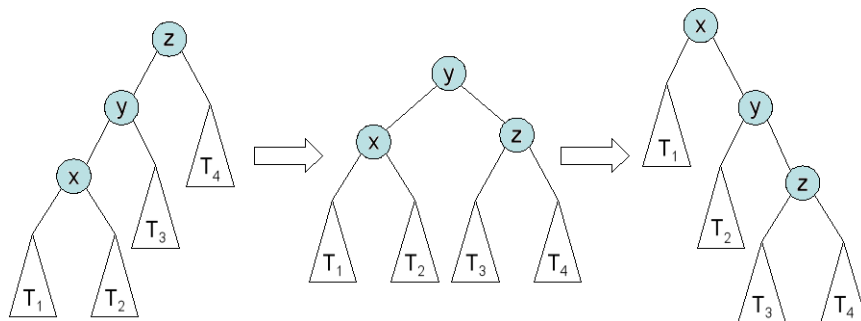


Figura 2: Operazione di **splay** sul nodo x — caso (ii)

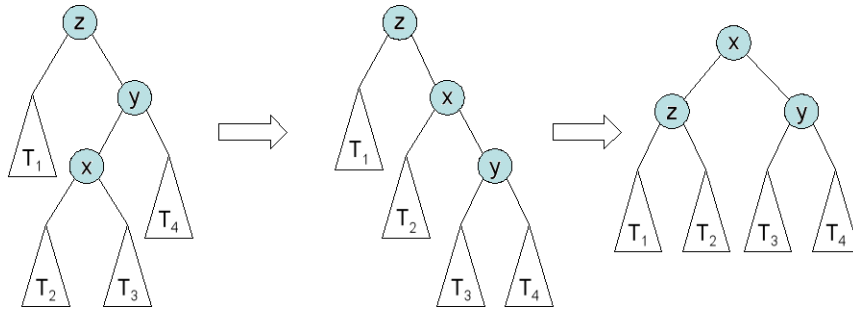


Figura 3: Operazione di **splay** sul nodo x — caso (iii)

- dopo aver eliminato un nodo z , eseguiamo uno **splay** sul padre di z appena prima dell'eliminazione, a meno che z non fosse la radice dell'albero prima della sua eliminazione (e quindi abbia NIL come padre).

SPLAYTREE-DELETE(T, z)

```

1  TREE-DELETE( $T, z$ )
2  if  $z.p \neq \text{NIL}$ 
3      then  $\text{splay}(T, z.p)$ 
```

Per lo pseudocodice delle funzioni

TREE-INSERT(T, z)

TREE-DELETE(T, z)

riferirsi a [2].

In ciascun caso, il costo dell'operazione è proporzionale alla lunghezza del cammino su cui lo **splay** procede.

2 Analisi basata sul potenziale

Mostriamo ora che il costo ammortizzato di una operazione di **splay** è $O(\lg n)$; utilizzeremo a tale scopo il metodo del potenziale.

Definizione 1. Sia T uno *splay tree*. Per ogni nodo x , sia $d(x)$ il numero di discendenti di x , incluso se stesso. Definiamo “*rango di x* ” il valore $r(x) = \lg d(x)$ e la funzione potenziale come

$$\Phi(T) = \sum_{x \in T} r(x) \quad (1)$$

È facile verificare che più l'albero è bilanciato, più basso è il valore del potenziale $\Phi(T)$.

Notiamo inoltre che un albero con un solo nodo ha un valore del potenziale pari a 0. Se consideriamo quindi di iniziare una sequenza di operazioni di **insert**, **search** e **delete** a partire da un albero T_0 con un solo nodo, avremo $\Phi(T_0) = 0$. Dal momento che non chiamiamo mai la funzione **splay** su di un albero vuoto, non sarà mai possibile avere un valore di $\Phi(T_i)$ minore di 0, quindi $\Phi(T_i) \geq \Phi(T_0)$ e possiamo usare la (1) per calcolare i costi ammortizzati.

Lemma 1. *Sia x un nodo in un albero, con figli y e z . Allora*

$$r(x) > 1 + \min\{r(y), r(z)\} \quad (2)$$

Dimostrazione. Chiaramente, $d(x) = d(y) + d(z) + 1$, ed inoltre $d(y) + d(z) \geq 2 \min\{d(y), d(z)\}$, per cui:

$$d(x) \geq 2 \min\{d(y), d(z)\} + 1$$

Prendendo il logaritmo di ambedue i membri della disuguaglianza, si ottiene

$$\lg d(x) \geq \lg(2 \min\{d(y), d(z)\} + 1)$$

ricordando la definizione 1 possiamo riscrivere

$$r(x) \geq \lg(2 \min\{d(y), d(z)\} + 1)$$

Osserviamo inoltre che il logaritmo è una funzione strettamente crescente, da cui

$$\begin{aligned} r(x) &> \lg(2 \min\{d(y), d(z)\}) \\ &= \lg 2 + \lg(\min\{d(y), d(z)\}) \\ &= 1 + \lg(\min\{d(y), d(z)\}) \end{aligned}$$

da cui, sempre per la proprietà di monotonicità della funzione logaritmo, otteniamo

$$r(x) > 1 + \min\{\lg d(y), \lg d(z)\}$$

richiamando ancora una volta la definizione 1:

$$r(x) > 1 + \min\{r(y), r(z)\}$$

□

Daremo ora una limitazione superiore alla variazione di potenziale causata da un singolo passo di splay. Questa limitazione sarà poi utile per calcolare il costo ammortizzato di una singola operazione di splay.

Lemma 2. *Sia x un nodo in uno splay tree. Siano $r_{i-1}(x)$ ed $r_i(x)$ i ranghi del sottoalbero radicato in x prima e dopo un passo di **splay** ad x , rispettivamente. In modo simile, siano T_{i-1} e T_i l'albero prima e dopo il passo di splay. Risulta:*

$$(i) \quad r_i(x) \geq r_{i-1}(x)$$

$$(ii) \quad \text{Se } p[x] \text{ è radice, allora } \Phi(T_i) - \Phi(T_{i-1}) < r_i(x) - r_{i-1}(x)$$

$$(iii) \quad \text{Se } p[x] \text{ non è radice, allora } \Phi(T_i) - \Phi(T_{i-1}) < 3(r_i(x) - r_{i-1}(x)) - 1$$

Dimostrazione. Il punto (i) è ovvio, poichè i discendenti di x aumentano quando x sale nell'albero. Per brevità, denoteremo d'ora in avanti

$$\Delta\Phi_i = \Phi(T_i) - \Phi(T_{i-1})$$

Relativamente al punto (ii), osservando la Figura 1 si vede che $r_i(x) = r_{i-1}(y)$. Poichè solo x e y cambiano rango in T_i , risulta:

$$\Delta\Phi_i = (r_i(x) - r_{i-1}(x)) + (r_i(y) - r_{i-1}(y)) = r_i(y) - r_{i-1}(x)$$

In seguito all'operazione di splay, x risulta essere il nuovo padre di y , nonchè radice dell'albero, quindi $r_i(x) > r_i(y)$, da cui otteniamo:

$$\Delta\Phi_i < r_i(x) - r_{i-1}(x)$$

Per il punto (iii), consideriamo solo il caso in Figura 2, essendo gli altri casi analoghi. In questo caso solo x , y e z cambiano rango nella transizione da T_{i-1} a T_i , e quindi risulta

$$\Delta\Phi_i = (r_i(x) - r_{i-1}(x)) + (r_i(y) - r_{i-1}(y)) + (r_i(z) - r_{i-1}(z))$$

Guardando gli alberi iniziali e finali, abbiamo $r_i(y) < r_i(x)$ e $r_{i-1}(x) < r_{i-1}(y)$, quindi, sommando:

$$r_i(y) - r_{i-1}(y) < r_i(x) - r_{i-1}(x)$$

che implica

$$\Delta\Phi_i < 2(r_i(x) - r_{i-1}(x)) + (r_i(z) - r_{i-1}(z)) \quad (3)$$

Per completare la dimostrazione, è sufficiente dimostrare che

$$r_i(z) - r_{i-1}(z) < r_i(x) - r_{i-1}(x) - 1$$

Denoteremo con r'_{i-1} il rango calcolato sull'albero intermedio T'_{i-1} , ottenuto dopo la prima rotazione. Grazie al Lemma 1:

$$r'_{i-1}(y) > 1 + \min\{r'_{i-1}(x), r'_{i-1}(z)\}$$

e guardando agli alberi iniziale T_{i-1} , intermedio T'_{i-1} e finale T_i , risulta $r'_{i-1}(x) = r_{i-1}(x)$, $r'_{i-1}(y) = r_i(x) = r_{i-1}(z)$, e $r'_{i-1}(z) = r_i(z)$, così che:

$$r_i(x) = r_{i-1}(z) > 1 + \min\{r_{i-1}(x), r_i(z)\}$$

Quindi, o abbiamo $r_i(x) > 1 + r_{i-1}(x)$ oppure $r_{i-1}(z) > 1 + r_i(z)$. Nel primo caso,

$$r_i(x) - r_{i-1}(x) > 1 \Rightarrow r_i(x) - r_{i-1}(x) - 1 > 0$$

e poichè $r_i(z) < r_{i-1}(z)$, otteniamo la seguente catena di disuguaglianze:

$$r_i(z) - r_{i-1}(z) < 0 < r_i(x) - r_{i-1}(x) - 1$$

Nel secondo caso,

$$r_{i-1}(z) > 1 + r_i(z) \Rightarrow r_i(z) - r_{i-1}(z) < -1$$

e, poichè dal Lemma 2

$$r_i(x) \geq r_{i-1}(x) \Rightarrow r_i(x) - r_{i-1}(x) \geq 0 \Rightarrow r_i(x) - r_{i-1}(x) - 1 \geq -1$$

otteniamo ancora

$$r_i(z) - r_{i-1}(z) < -1 \leq r_i(x) - r_{i-1}(x) - 1$$

Usando la disuguaglianza appena dimostrata nell'equazione 3 si ottiene l'asserto. $\square$

Teorema 1. *Il costo ammortizzato di una operazione completa di **splay** in un albero auto-aggiustante con n nodi è $O(\lg n)$.*

Dimostrazione. Usiamo il lemma 2 per calcolare il costo ammortizzato di una operazione di **splay** al nodo x , che consiste di una sequenza di passi di **splay** (rotazioni). Per ciascun passo di **splay**, siano T_{i-1} e T_i gli alberi prima e dopo la rotazione, rispettivamente. Il costo ammortizzato di un passo di **splay** è dato dal costo effettivo c_i più la variazione di potenziale:

$$\hat{c}_i = c_i + \Phi(T_i) - \Phi(T_{i-1})$$

Assumiamo che il costo reale c_i di un singolo passo costituente l'operazione di **splay** sia pari ad $O(1)$. Consideriamo i due casi dell'enunciato del Lemma 2.

- se x è figlio della radice, siamo all'ultimo passo di quelli costituenti lo **splay**, e quindi:

$$\begin{aligned}\hat{c}_i &= O(1) + \Phi(T_i) - \Phi(T_{i-1}) \\ &< O(1) + r_i(x) - r_{i-1}(x) \\ &= O(1) + \Delta r_i \\ &< O(1) + 3\Delta r_i\end{aligned}$$

- se x non è figlio della radice:

$$\hat{c}_i = O(1) + \Phi(T_i) - \Phi(T_{i-1}) < O(1) + 3(r_i(x) - r_{i-1}(x)) - O(1) < 3\Delta r_i$$

Per l'intera operazione di **splay**, sia $T = T_0, T_1, T_2, \dots, T_k$ la serie di alberi prodotta dalla sequenza di k passi di **splay**, e siano $r = r_0, r_1, r_2, \dots, r_k$ i corrispondenti valori delle funzioni di rango. Il costo ammortizzato totale è dato da

$$\sum_{i=1}^k \hat{c}_i < O(1) + 3\Delta r_k + \sum_{i=1}^{k-1} 3\Delta r_i = O(1) + \sum_{i=1}^k 3\Delta r_i$$

L'ultimo termine della disuguaglianza contiene una serie telescopica, e quindi il costo ammortizzato totale è pari a:

$$\sum_{i=1}^k \hat{c}_i < O(1) + 3(r_k(x) - r_0(x))$$

Poichè il rango finale di x è al più pari a $\lg n$, otteniamo il tempo desiderato. $\square$

Dal teorema appena dimostrato si può infine ricavare il seguente:

Teorema 2. *Il tempo ammortizzato di una sequenza di m operazioni di **insert**, **search** e **delete** in un albero auto-aggiustante è $O(m \cdot \lg n)$ dove n è il massimo numero di nodi che l'albero raggiunge durante la sequenza di operazioni.*

References

- [1] Daniel D. Sleator and Robert E. Tarjan. Self Adjusting Binary Search Trees.
In *Journal of the ACM*, vol. 32, n. 3, pp. 652-686, July 1985.
- [2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
Introduzione agli Algoritmi e Strutture Dati — III ed. Mc-Graw Hill Italia,
2010.