Gli Alberi Splay
Gestione ed Analisi Ammortizzata

Simone Faro

Università di Catania
Corso di Algoritmi e Complessità
faro@dmi.unict.it

1 Introduzione agli alberi splay

Gli alberi splay sono alberi binari di ricerca con la proprietà che gli elementi cui si è acceduto più di recente tendono a trovarsi più vicini alla radice. In questo modo la loro ricerca è più efficiente che in un normale albero binario e talvolta anche più che in un albero bilanciato.

A differenza di altre possibili realizzazioni di alberi binari di ricerca, gli alberi autobilanciati 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.

Queste rotazioni, oltre a portare il nodo alla radice, accorciano di circa la metà la distanza tra la radice e tutti i nodi visitati durante l'accesso. Tuttavia, esse non garantiscono che l'albero risultante sia bilanciato, e nel caso peggiore un accesso a un nodo può richiedere di visitare tutti i nodi dell'albero, con una complessità totale lineare.

Tuttavia mostreremo come, mediante tali rotazioni, si riescano ad ottenere dei tempi di esecuzione ammortizzati di tipo logaritmico su una sequenza di operazioni. In particolare si dimostra come il costo ammortizzato di una sequenza di m operazioni di inserimento, cancellazione e ricerca, di cui n operazioni sono di inserimento, ha un costo pari a $O(m \log n)$.

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. Per questo motivo gli alberi splay sono preferiti per l'implementazione di cache, in cui le informazioni non sono accedute uniformemente, ma una parte degli elementi vengono acceduti pi frequentemente di altri.

2 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.

Per brevità, denoteremo con y il nodo $p[x]$, padre di x nell'albero. Chiameremo inoltre nonno di x, e lo indicheremo con z, il nodo $p[p[x]]$, il padre di $p[x]$. In ciascun passo di splay, le rotazioni da eseguire vengono scelte tra tre diversi casi, come segue:

– **Operazione Zig.**
se x è figlio della radice, ruotiamo su y in modo che il nodo x diventi la nuova radice (Figura 1);

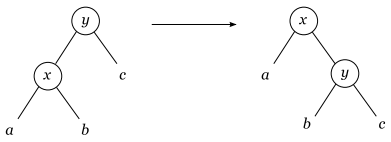

Figura 1. Operazione Zig sul nodo x. Il padre del nodo x è la radice dell'albero T.

– **Operazione ZigZig.**
se x ha un nonno, z, ed x e y sono entrambi figli destri o sinistri del proprio genitore, ruotiamo prima su z e poi su y (Figura 2);

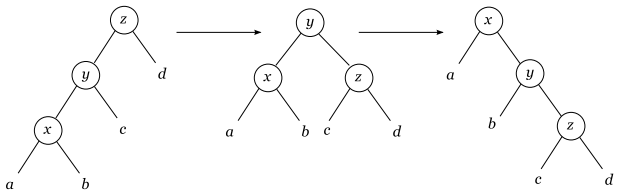

Figura 2. Operazione ZigZig sul nodo x. Il nodo x e il nodo y sono entrambi figli sinistri dei rispettivi genitori.

– **Operazione ZigZag.**
se x ha un nonno, z, ed x è figlio sinistro/destro mentre y è figlio destro/sinistro del proprio genitore, ruotiamo prima su y e poi sul nuovo padre di x (ovvero su z prima della rotazione) (Figura 3).

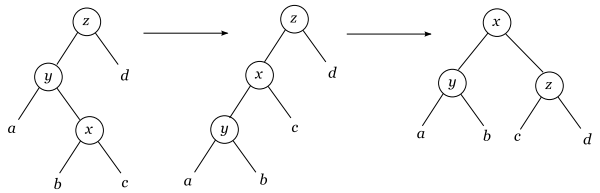

Figura 3. Operazione ZigZag sul nodo x. Il nodo x è figlio destro di y e y è figlio sinistro di z.

Ciascun passo che costituisce l'operazione di splay richiede tempo $O(1)$. L'effetto della strategia di ristrutturazione splay è quello di far risalire il nodo x alla radice, riarrangiando i sottoalberi incotrati lungo il cammino. Come vedremo in seguito, per rendere un albero binario di ricerca autobilanciato, eseguiamo l'operazione splay in corrispondenza di ciascuna operazione di accesso o di modifica dell'albero.

Di seguito è mostrato lo pseudocodice di una singola operazione Zig sul nodo x e lo pseudocodice della procedura splay.

```
Zig(T, x)
1.    y ← p[x]
2.    if x = left[y] then
3.         RightRotate(T, y)
6.    else LeftRotate(T, y)
```

```
Splay(T, x)
1.    while root[T] ≠ x do
2.         y ← p[x]
3.         if root[T] = y then
4.              Zig(T, y)    /* operazione Zig */
5.         else
6.              z ← p[y]
7.              if (x = left[y] and y = left[z]) or
8.                 (x = right[y] and y = right[z]) then
9.                   Zig(T, y)    /* operazione ZigZig */
10.                  Zig(T, x)
11.             else
12.                  Zig(T, x)    /* operazione ZigZag */
13.                  Zig(T, x)
```

3 Gestione di un albero Splay

Accesso ad una chiave dell'albero

L'operazione di accesso ad una chiave dell'albero consiste nella ricerca della chiave, partendo dalla root, e procedendo verso le foglie dell'albero secondo il meccaniosmo standard di ricerca in un albero di binario di ricerca. Dopo aver cercato una chiave, viene eseguito 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

```
Access(T, k)
 1.  x ← root[T]
 2.  while x ≠ nil and k ≠ key[x] do
 3.        last ← x
 4.        if k < key[x] then
 5.              x ← left[x]
 6.        else x ← right[x]
 7.  if x ≠ nil then
 8.        Splay(T, x)
 9.  else
10.        if root[T] ≠ nil then
11.              Splay(T, last)
12.  return x
```

Unione di due alberi

L'operazione Join unisce due alberi binari di ricarca, T_1 e T_2, in un'unico albero binario di ricerca. L'applicazione dell'operazione Join su T_1 e T_2 presuppone che se $x \in T_1$ e $y \in T_2$ allora $key[x] < key[y]$. In altre parole le chiavi di T_1 sono più piccole delle chiavi di T_2.

L'operazione Join effettua uno splay sulla chiave più grande di T_1. Dopo tale operazione la radice di T_1 avrà il solo figlio sinistro. In seguito si collega T_2 come figlio destro della radice di T_1.

```
Join(T1, T2)
1.  Access(T1, max(T1))
2.  right[root[T1]] ← root[T2]
3.  p[root[T2]] ← root[T1]
4.  return T1
```

Divisione di un albero

L'operazione Split viene chiamata su un'albero T e prende come parametro una chiave k. Questa operazione restitutisce in output due alberi binari di ricerca, T_1 e T_2, dove T_1 contiene le chiavi minori di k e T_2 contiene quelle maggiori o uguali.

In particolare l'operazione Split, con parametro k, effettua un'accesso della chiave k all'interno dell'albero. In seguito controlla la chiave della radice, t, dell'albero. Se $key[t] < k$ allora viene tagliato il figlio destro e si ritorna tale nodo come radice di T_2 mentre si ritorna t stesso come radice dell'albero T_1. In caso contrario viene tagliato il figlio sinistro di t e si ritorna tale nodo come radice dell'albero T_1 mentre si ritorna il nodo t come radice di T_2.

```
Split(T, k)
 1.  Access(T, k)
 2.  t ← root[T]
 3.  if key[t] < k then
 4.        root[T_1] ← t
 5.        root[T_2] ← right[t]
 6.        p[right[t]] ← nil
 7.        right[t] ← nil
 8.  else
 9.        root[T_1] ← left[t]
10.        root[T_2] ← t
11.        p[left[t]] ← nil
12.        left[t] ← nil
13.  return T_1 and T_2
```

Ricerca di una chiave

L'operazione di ricerca di una chiave all'interno di un'albero Splay si traduce in un'operazione di accesso alla chiave k. Al termine dell'operazione di accesso, se la chiave k è contenuta all'interno dell'albero, il nodo che la contiene sarà la radice dell'albero stesso.

```
Search(T, k)
1.  Access(T, k)
2.  if root[T] ≠ nil and key[root[T]] = k then
3.        return root[T]
4.  else return nil
```

Inserimento di una nuova chiave

L'operazione di inserimento di una nuova chiave k all'interno dell'albero T si divide in due passi. Il primo passo consiste nell'effettuare uno split dell'albero T utilizzando la chiave k come parametro, ottenendo in output due alberi T_1 e T_2. Il secondo passo crea un nuovo albero T la cui radice consiste nel nuovo nodo con chiave k. A tale nodo vengono collegate le radici degli alberi T_1 e T_2 come figlio sinistro e figlio destro, rispettivamente.

```
Insert(T, k)
1.  (T_1, T_2) ←Split(T, k)
2.  t ← NewNode(k)
3.  left[t] ← root[T_1]
4.  right[t] ← root[T_2]
5.  p[root[T_1]] ← p[root[T_2]] ← t
6.  root[T] ← t
7.  return T
```

Cancellazione di una chiave

L'operazione di cancellazione di una chiave k dall'albero T si divide ancora una volta in due passi. Il primo passo consiste nell'effettuare un'accesso al nodo con chiave k dell'albero T. In questo modo, se la chiave k si trova all'interno dell'albero, il nodo che la contiene diventerà la nuova radice dell'albero T. In seguito se la radice dell'albero contiene la chiave k si effettua l'operazione join dei sottoalberi radicati sul suo figlio sinistro e sul suo figlio destro. L'albero ottenuto dalla join sarà il nuovo albero T.

```
Delete(T, k)
1.  Access(T, k)
2.  if key[root[T]] = k then
3.      root[T_1] ← left[root[T]]
4.      root[T_2] ← right[root[T]]
5.      p[root[T_1]] ← p[root[T_2]] ← nil
6.      left[root[T]] ← right[root[T]] ← nil
7.      T ←Join(T_1, T_2)
8.  return T
```

4 Analisi ammortizzata

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

Sia T uno splay tree. Per ogni nodo $x \in T$, definiamo il valore $s(x)$ come il numero di discendenti di x, incluso se stesso. In altre parole $s(x)$ è il numero di nodi contenuti nel sottoalbero radicato in x. Per ogni nodo x definiamo anche il rango di x, indicato con il simbolo $r(x)$, il cui valore è posto pari a $r(x) = \log s(x)$. Infine la funzione potenziale associata all'albero T è definita come

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

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 inserimento ricerca e cancellazione a partire da un albero T_0 con un solo nodo,

avremo $\Phi(T_0) = 0$ (questa relazione risulta valida anche se iniziamo la sequenza di operazioni con un albero vuoto). Inoltre, 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 tale funzione per calcolare i costi ammortizzati.

Nel corso della nostra analisi faremo uso del seguente semplice risultato.

Lemma 1. *Siano a e b due numeri naturali positivi e sia $c \geq a + b$. Allora vale la relazione $2 \log c - \log a - \log b \geq 2$.*

Dimostrazione.
Al fine di dimostrare il presente lemma si faccia riferimento alle seguenti relazioni

$$
\begin{array}{ll}
(a+b)^2 \geq 0 & \\
a^2 - 2ab + b^2 \geq 0 & \\
a^2 + 2ab + b^2 \geq 4ab & \text{(sommando a entrambi i membri } 4ab\text{)} \\
(a+b)^2 \geq 4ab & \\
a + b \geq 2\sqrt{ab} & \text{(elevando entrambi i membri a } \tfrac{1}{2}\text{)} \\
c \geq 2\sqrt{ab} & \text{(perchè } c \geq a + b\text{)} \\
\log c \geq \log 2 + \log \sqrt{ab} & \text{(passando ai logaritmi)} \\
\log c \geq 1 + \frac{1}{2}(\log a + \log b) & \\
2 \log c - 2 \geq loga + logb &
\end{array}
$$

L'ultima espressione dimostra la nostra tesi. □

Al fine di calcolare il costo ammortizzato di un'operazione splay su un albero binario di ricerca, proviamo a calcolare il costo ammortizzato delle singole operazioni Zig, ZigZig e ZigZag. A tal fine, se consideriamo la rotazione di un nodo dell'albero come un'operazione unitaria, otteniamo i seguenti costi effettivi per le tre diverse operazioni.

$$
\begin{aligned}
c_{\text{Zig}} &= 1 \\
c_{\text{ZigZig}} &= 2 \\
c_{\text{ZigZag}} &= 2
\end{aligned}
$$

Nel tentativo di effettuare l'analisi dei costi ammortizzati delle diverse operazioni indicheremo con T l'albero prima della trasformazione e con T' l'albero ottenuto subito dopo la trasformazione. In questo contesto utilizzeremo anche i simboli $r(x)$ e $s(x)$ per indicare i valori di rango e size di un nodo x prima della trasformazione, mentre utilizzeremo i simboli $r'(x)$ e $s'(x)$ per indicare i medesimi valori dopo la trasformazione.

Di seguito analizziamo separatamente i costi ammortizzati delle tre singole operazioni.

Costo ammortizzato di una Zig. Per il calcolo del costo ammortizzato dell'operazione Zig sul nodo x si faccia riferimento alla Figura 1. In particolare si ottiene

$$\begin{aligned}
\hat{c}_{\text{Zig}} &= c_{\text{Zig}} + \Phi(T) - \Phi(T') \\
&= 1 + \sum_{x \in T} r(x) - \sum_{x \in T'} r'(x) \\
&= 1 + r'(x) + r'(y) - r(x) - r(y) && \text{(solo } x \text{ e } y \text{ cambiano rango)} \\
&\leq 1 + r'(x) - r(x) && \text{(poichè } r(y) \geq r'(y)\text{)} \\
&\leq 1 + 3(r'(x) - r(x)) && \text{(poichè } r'(x) \geq r(x)\text{)}
\end{aligned}$$

Costo ammortizzato di una ZigZig. Per il calcolo del costo ammortizzato dell'operazione ZigZig sul nodo x si faccia riferimento alla Figura 2. In particolare si ottiene

$$\begin{aligned}
\hat{c}_{\text{ZigZig}} &= c_{\text{ZigZig}} + \Phi(T) - \Phi(T') \\
&= 2 + \sum_{x \in T} r(x) - \sum_{x \in T'} r'(x) \\
&= 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) && \text{(solo } r(x), r(y) \text{ e } r(z) \text{ cambiano)} \\
&= 2 + r'(y) + r'(z) - r(x) - r(y) && \text{(poichè } r(z) = r'(x)\text{)} \\
&\leq 2 + r'(y) + r'(z) - 2r(x) && \text{(poichè } r(y) \geq r(x)\text{)} \\
&\leq 2 + r'(x) + r'(z) - 2r(x) && \text{(poichè } r'(x) \geq r'(y)\text{)}
\end{aligned}$$

Si osservi adesso che $s(x)$ ed $s'(z)$ sono due numeri naturali positivi e che inoltre $s'(x) \geq s(x) + s'(z)$. È quindi possibile applicare il Lemma 1 ottenendo la relazione $2 \log s'(x) - \log s(x) - \log s'(z) \geq 2$ da cui segue che

$$2 \leq 2r'(x) - r(x) - r'(z).$$

Applicando questa relazione alla precedente si ottiene

$$\begin{aligned}
\hat{c}_{\text{ZigZig}} &= 2 + r'(x) + r'(z) - 2r(x) \\
&\leq 2r'(x) - r(x) - r'(z) + r'(x) + r'(z) - 2r(x) \\
&= 3r'(x) - 3r(x) \\
&= 3(r'(x) - r(x))
\end{aligned}$$

Costo ammortizzato di una ZigZag. Per il calcolo del costo ammortizzato dell'operazione ZigZag sul nodo x si faccia riferimento alla Figura 3. In particolare si ottiene

$$\begin{aligned}
\hat{c}_{\text{ZigZag}} &= c_{\text{ZigZag}} + \Phi(T) - \Phi(T') \\
&= 2 + \sum_{x \in T} r(x) - \sum_{x \in T'} r'(x) \\
&= 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) && \text{(solo } r(x), r(y) \text{ e } r(z) \text{ cambiano)} \\
&= 2 + r'(y) + r'(z) - r(x) - r(y) && \text{(poichè } r(z) = r'(x)\text{)} \\
&\leq 2 + r'(y) + r'(z) - 2r(x) && \text{(poichè } r(y) \geq r(x)\text{)}
\end{aligned}$$

Si osservi adesso che $s'(y)$ ed $s'(z)$ sono due numeri naturali positivi e che inoltre $s'(x) \geq s'(y) + s'(z)$. È quindi possibile applicare il Lemma 1 ottenendo la relazione $2\log s'(x) - \log s'(y) - \log s'(z) \geq 2$ da cui segue che

$$2 \leq 2r'(x) - r'(y) - r'(z).$$

Applicando questa relazione alla precedente si ottiene

$$\begin{aligned}
\hat{c}_{\mathsf{ZigZag}} &= 2 + r'(y) + r'(z) - 2r(x) \\
&\leq 2r'(x) - r'(y) - r'(z) + r'(y) + r'(z) - 2r(x) \\
&= 2r'(x) - 2r(x) \\
&\leq 3(r'(x) - r(x))
\end{aligned}$$

Costo ammortizzato di una Splay. Una singola applicazione dell'operazione splay su un nodo x consiste nell'applicare una sequenza di passi Zig, ZigZig e ZigZag fino a che il nodo x non diventi la radice dell'albero. Si noti tuttavia che il passo Zig può essere applicato solo come ultimo passo della sequenza.

Per l'analisi ammortizzata dell'operazione splay si supponga che essa venga applicata sul nodo x contenuto in un albero T con radice t. Si supponga inoltre che l'operazione splay consista di una sequenza di k passi (di tipo Zig, ZigZig e ZigZag). Partendo dalla configurazione iniziale dell'albero T si ottiene l'albero T^1 dopo l'applicazione del primo passo e, in generale, l'albero T^i dopo l'applicazione dell'i-esimo passo.

Se indichiamo con $r^i(x)$ il rango di un nodo $x \in T^i$ possiamo affermare che ogni singolo passo della sequenza ha un costo ammortizzato $\hat{c}_i \leq 3(r^i(x) - r^{i-1}(x))$. Per cui il costo ammortizzato dell'intera sequenza di passi è pari a

$$\begin{aligned}
\hat{c}_{\mathsf{Splay}} &= \sum_{i=1}^{k-1} \hat{c}_i + 3(r^k(x) - r^{k-1}(x)) + 1 \\
&\leq \sum_{i=1}^{k} 3(r^i(x) - r^{i-1}(x)) \\
&= 3(r^{k-1}(x) - r(x)) + 3(r^k(x) - r^{k-1}(x)) + 1 \\
&= 3(r^k(x) - r(x)) + 1 \\
&= 3(r(t) - r(x)) + 1
\end{aligned}$$

Nel caso pessimo si ha che x è una foglia a distanza massima dalla radice. Ne segue che $s(x) = 1$ che implica $r(x) = \log s(x) = 0$. Il costo ammortizzato di un'operazione splay sarà quindi pari a $3r(t) + 1 = 3\log s(t) + 1 = O(\log |T|)$.

Se analizziamo il comportamento di una sequenza di m operazioni di inserimento cancellazione e ricerca su un albero splay, di cui n di queste sono operazioni di inserimento, si avrà che $|T| \leq n$. Per cui una sequenza di m operazioni avrà costo ammortizzato pari a $O(m \log n)$ con un costo medio pari a $O(\log n)$ per ogni singola operazione.