UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI STUDIO IN
Informatica

TESI DI LAUREA IN
Algoritmi e Strutture Dati

TITOLO
La capacità di adattarsi:
Analisi Teorica e Sperimentale degli Splay Tree

Relatore:
Chiar.mo Prof.
Luciano Gualà

Laureando:
matricola: *0306676*
Matteo Cipolletta

Anno Accademico 2023/2024

La misura dell'intelligenza è data
dalla capacità di cambiare
quando è necessario.

– A. Einstein

Indice

Introduzione

Nell'ambito delle strutture dati, gli alberi di ricerca binari svolgono un ruolo fondamentale in numerosi algoritmi e applicazioni informatiche. Ne esistono numerose versioni ma, tra queste, gli Splay Tree si distinguono grazie ad una particolare caratteristica: la capacità di adattarsi dinamicamente in base alla sequenza di accessi effettuati.

A differenza degli alberi bilanciati classici, come i BST statici o gli AVL Tree, gli Splay Tree non mantengono esplicitamente il bilanciamento, ma si ristrutturano attraverso un processo chiamato Splaying. Questa operazione avviene ogni volta che si effettua un accesso e permette di avere le informazioni più utilizzate, a disposizione vicino alla cima dell'albero. Questo comportamento consente agli Splay Tree di avere delle prestazioni particolarmente favorevoli rispetto ad altre strutture, in termini di costi ammortizzati.

L'obiettivo di questa tesi è esplorare nel dettaglio le proprietà teoriche degli Splay Tree, evidenziando la loro capacità di adattamento ai pattern di accesso. Inoltre, si mettono in mostra i benefici che ne derivano in termini computazionali in diversi scenari applicativi. In base a questa descrizione, il lavoro si suddivide in due parti complementari.

Nella prima parte, quella teorica, vengono analizzati gli Splay Tree partendo dalla definizione formale della struttura e mostrando successivamente le operazioni fondamentali, con i relativi costi ammortizzati, e i teoremi principali, accompagnati dalle rispettive dimostrazioni.

La parte pratica, invece, è dedicata all'analisi sperimentale dei risultati teorici, evidenziati nella sezione precedente. In particolare, vengono effettuati dei confronti concreti tra i costi della struttura analizzata e i costi dei BST statici. In questo modo, vengono mostrati gli scenari in cui l'auto-bilanciamento adattivo degli Splay Tree si traduce in un vero e proprio vantaggio prestazionale.

Questa tesi offre dunque una visione completa e articolata degli Splay Tree, mettendone in luce sia i fondamenti teorici, sia le implicazioni pratiche, con particolare attenzione alla loro caratteristica principale: la capacità di adattarsi dinamicamente.

Capitolo 1

Fondamenti Teorici sugli Splay Tree

1.1 Introduzione agli Splay Tree

Gli Splay Tree sono stati introdotti per la prima volta da Daniel Sleator e Robert Tarjan nel 1985 e sono una particolare variante dei Binary Search Tree (BST). Queste strutture dati sono dette self-adjusting in quanto si adattano in automatico in base al pattern di accesso.

A differenza di altre strutture basate sui BST, come gli AVL Tree, gli Splay Tree non mantengono un bilanciamento esplicito attraverso fattori di bilanciamento ma utilizzano un particolare meccanismo chiamato Splaying. Questa operazione permette di ristrutturare l'albero in modo da avvicinare i nodi utilizzati più frequentemente alla radice dell'albero. In questo modo, le operazioni successive su quei particolari nodi risultano essere più efficienti.

In particolare, gli Splay Tree sono spesso utilizzati per implementare strutture dati come i dizionari dinamici, in cui è necessario mantenere un insieme di chiavi ordinate. In questo contesto, le principali operazioni supportate sono:

- **access:** prende in input un albero e una chiave. Restituisce il nodo associato alla chiave, se presente nell'albero;
- **insert:** prende in input un albero e una chiave (eventualmente accompagnata da altre informazioni aggiuntive). Aggiunge nell'albero un nuovo nodo con la chiave specificata;
- **findMax:** prende in input un albero e trova il nodo con chiave associata massima tra tutte quelle presenti nell'albero;
- **join:** prende in input due alberi e restituisce l'albero risultante dall'unione dei due;
- **split:** prende in input un albero e una chiave. Restituisce due alberi separati in corrispondenza del nodo in input;
- **delete:** prende in input un albero e una particolare chiave. Elimina il nodo associato alla chiave e restituisce l'albero rimanente.

Grazie alla procedura di Splaying, le operazioni sugli Splay Tree godono di una complessità temporale ammortizzata di $O(\log\ n)$, rendendoli ottimi candidati per molte applicazioni pratiche.

1.2 Rappresentazione e Struttura

1.2.1 Struttura dei nodi

Uno Splay Tree segue la stessa filosofia di un BST. Infatti, ogni nodo ha:

- una chiave (o valore) numerica confrontabile;
- un figlio sinistro e un figlio destro (eventualmente nulli) che rispettano le proprietà dei BST:
  - I nodi nel sotto-albero sinistro contengono solo valori minori della chiave del nodo padre;
  - I nodi nel sotto-albero destro contengono solo valori maggiori della chiave del nodo padre;
- (facoltativo) un puntatore al nodo padre, per semplificare l'implementazione delle varie operazioni;
- (facoltativo) informazioni associate al nodo[1].

1.2.2 Struttura dell'albero

Uno Splay Tree è rappresentato come un BST classico, strutturando i nodi in base al valore della loro chiave. In seguito a un'operazione di Splaying, la posizione dei nodi può variare, ma viene comunque garantito il loro ordinamento generale rispettando la regola dei BST.

1.3 Operazione di Splaying

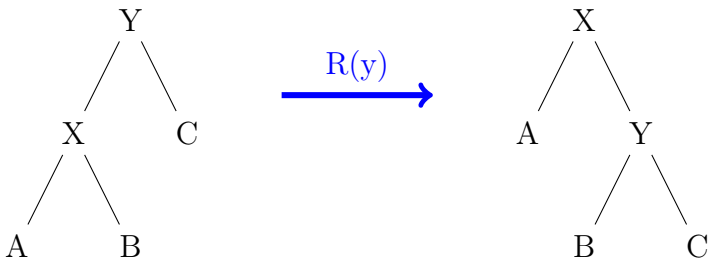
L'operazione di Splaying è il cuore del funzionamento degli Splay Tree. Questa procedura prende in input un nodo target e ha come obiettivo quello di portarlo in cima all'albero, rendendolo la radice e, nel frattempo, ristrutturare l'albero lungo il cammino di accesso al nodo.

[1] Per semplicità, nel documento non vengono memorizzate informazioni aggiuntive nei nodi

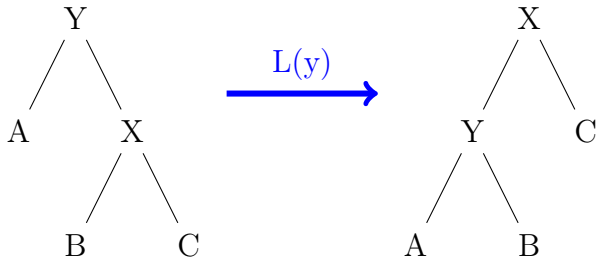
In questo contesto, andiamo a indicare con x il nodo target, con y il padre di x e con z il nonno di x (se esiste) prima delle rotazioni. Inoltre con i simboli A, B, C e D indichiamo dei sotto-alberi qualsiasi (eventualmente nulli).

Questa particolare operazione viene eseguita applicando una sequenza di rotazioni. In particolare, esistono due tipi di rotazioni fondamentali:

- **Right Rotation R:** viene eseguita quando x è figlio sinistro di y. Quindi, x viene spostato in alto mentre y viene fatto scendere, diventando figlio destro di x.

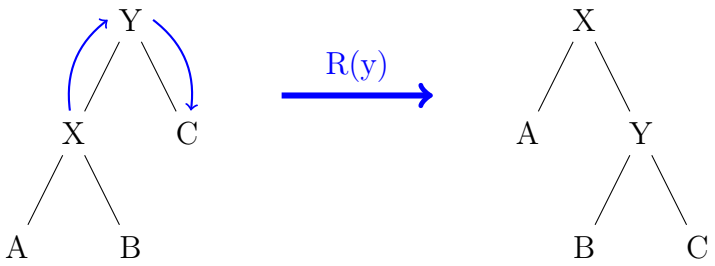


- **Left Rotation L:** viene eseguita quando x è figlio destro di y. Quindi, x viene spostato in alto mentre y viene fatto scendere, diventando figlio sinistro di x.

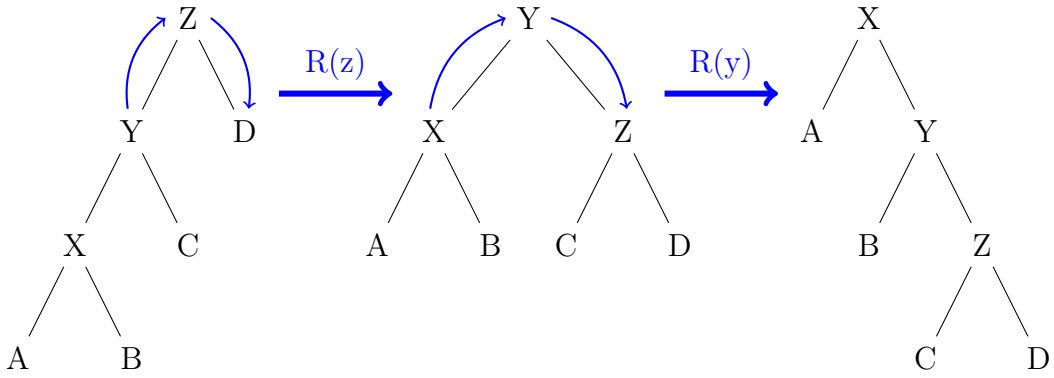


Le rotazioni appena definite vengono eseguite a seconda della posizione di x rispetto a y e z. Infatti, possiamo classificare tre casi fondamentali:

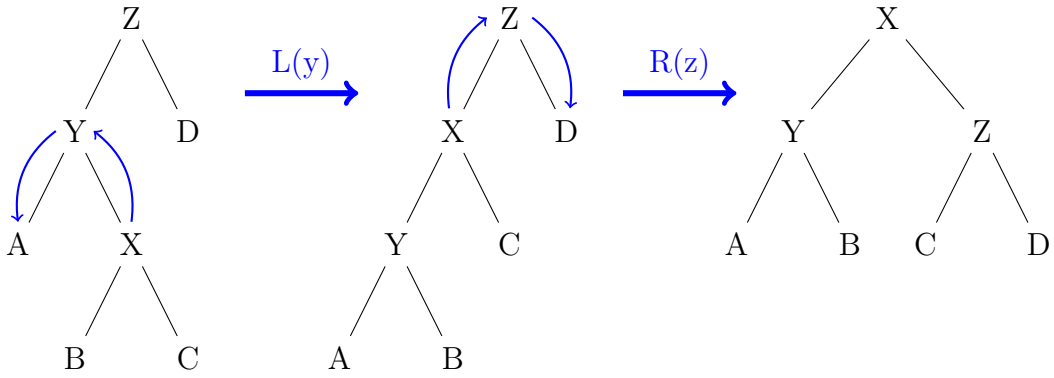
- **Caso 1: Zig**
  Viene eseguito quando y è la radice dell'albero e x è figlio sinistro di y. Si effettua una rotazione verso destra su y spostando i rispettivi sotto-alberi come si può osservare in figura.



- **Caso 2: Zig-Zig**
  Viene eseguito quando x e y sono entrambi figli sinistri. Si effettua una rotazione verso destra su z e, in seguito, un'ulteriore rotazione nella stessa direzione su y come si può osservare in figura.



- **Caso 3: Zag-Zig**
  Viene eseguito quando x è figlio destro di y e y è figlio sinistro di z. Si effettua una rotazione verso sinistra su y, portando x in alto e rendendolo figlio di z. In seguito, si effettua un'ulteriore rotazione verso destra su z.



In generale, con il termine zig si intende una rotazione verso destra, mentre con il termine zag si intende una rotazione verso sinistra. Quindi, in base a questa classificazione, possono esistere anche i corrispondenti casi zag, zag-zag e zig-zag.

Questa casistica viene applicata nella procedura di Splaying finché il nodo target non diventa la radice dell'albero.

1.3.1 Analisi ammortizzata e metodo del potenziale

Il metodo dell'analisi ammortizzata rappresenta un approccio più flessibile rispetto all'analisi di ogni singola operazione nel caso peggiore. Questo metodo permette di distribuire il costo di ogni singola operazione su tutta la sequenza di aggiornamento, tenendo conto del fatto che alcune operazioni possono essere più molto più economiche, compensando quelle più costose.

Per applicare questo concetto, si utilizza l'analisi della **funzione potenziale**. L'idea è quella di associare, a ogni possibile configurazione della struttura dati, un numero reale, chiamato appunto **potential**. Questo valore rappresenta una sorta di "energia accumulata" dell'albero e la sua variazione permette di stimare il costo ammortizzato effettivo di ciascuna operazione.

1.3.2 Analisi della complessità asintotica

L'operazione di Splaying ha un costo massimo di $O(n)$ nel caso peggiore in cui l'albero sia completamente sbilanciato. Tuttavia, si dimostra che il costo ammortizzato di questa operazione è $O(\log\ n)$.

Prima di procedere con l'analisi formale del costo dell'operazione di Splaying, dobbiamo dare alcune definizioni rigorose che ci torneranno utili ai fini della dimostrazione. Per ogni nodo $x \in T$ abbiamo:

- **Weight** $w(x)$**:** è un peso arbitrario ma fissato;
- **Size** $s(x)$**:** è la somma dei pesi dei nodi nel sotto-albero radicato in x;
- **Rank** $r(x)$**:** è il valore $\log s(x)$ [2].

Inoltre definiamo per T il valore seguente:

- **Potential** $\Phi(T)$**:** è il valore $\sum_{x \in T} r(x)$ e rappresenta la "disorganizzazione" dell'albero.

In particolare, tutte queste misure (tranne $w(x)$) cambiano dopo un'operazione di Splaying. Per questo motivo, andiamo a indicare con $s(x), r(x)$ e $\Phi(T)$ i rispettivi valori prima delle modifiche di uno Splaying e, con $s'(x), r'(x)$ e $\Phi'(T)$ i rispettivi valori dopo le modifiche di uno Splaying.

[2]Nel documento vengono utilizzati logaritmi in base 2

L'idea è quella di definire il tempo ammortizzato a di un'operazione come $a = t + \Phi'(T) - \Phi(T)$ dove t è il tempo reale di un'operazione. Con questa definizione, possiamo stimare il tempo totale di una sequenza di m operazioni come:

$$\sum_{j=1}^{m} t_j = \sum_{j=1}^{m} (a_j + \Phi_{j-1} - \Phi_j) = \sum_{j=1}^{m} a_j + \Phi_0 - \Phi_m$$

dove t_j e a_j sono rispettivamente il tempo reale e il tempo ammortizzato dell'operazione j, Φ_0 è il potenziale iniziale e Φ_j, per $j \geq 1$, è il potenziale dopo l'operazione j.
In altre parole, il tempo reale è equivalente a quello ammortizzato a cui viene aggiunta la diminuzione di potenziale dalla configurazione iniziale a quella finale.

Per misurare il tempo di esecuzione dell'operazione di Splaying, utilizziamo il numero di rotazione effettuate: quindi, contiamo 1 per ogni singola rotazione effettuata, mentre nel caso in cui non viene eseguita nessuna rotazione, contiamo 1 per il tempo totale dello Splaying.

Lemma 1 (**Access Lemma**). *Il tempo ammortizzato per effettuare un'operazione di Splaying al nodo x su un albero con radice t è al massimo* $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$.

Proof. Se non vengono effettuate rotazioni, il limite è immediato in quanto implica che $t = x$ e il tempo di accesso è costante. Supponiamo quindi di effettuare almeno una rotazione. Come abbiamo fatto in precedenza, mettiamo in evidenza i tre casi fondamentali dell'operazione di Splaying e dimostriamo i seguenti limiti:

- **Caso 1 (Zig):** il tempo ammortizzato è al massimo $3(r'(x) - r(x)) + 1$.
  In questo passaggio, viene effettuata solo una rotazione e dato che, solo i nodi x e y possono cambiare il loro rank, il tempo ammortizzato è:

  $$a = 1 + r'(x) + r'(y) - r(x) - r(y)$$

  Dato che $r(y) \geq r'(y)$, risulta che $a \leq 1 + r'(x) - r(x)$. Inoltre, dato che $r'(x) \geq r(x)$, risulta che $r'(x) - r(x) \geq 0$. Per cui possiamo scrivere:

  $$a \leq 3(r'(x) - r(x)) + 1$$

  In questo modo, abbiamo completato la dimostrazione di questo caso;

- **Caso 2 (Zig-Zig):** il tempo ammortizzato è al massimo $3(r'(x) - r(x))$.
  In questo passaggio, vengono effettuate due rotazioni e dato che, solo i nodi x, y e z possono cambiare il loro rank, il tempo ammortizzato è:

$$a = 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

  Dato che $r'(x) = r(z)$, risulta che $a = 2 + r'(y) + r'(z) - r(x) - r(y)$. Inoltre, dato che $r'(x) \geq r'(y)$, risulta che $a \leq 2 + r'(x) + r'(z) - r(x) - r(y)$. Infine, dato che $r(y) \geq r(x)$, risulta che $a \leq 2 + r'(x) + r'(z) - 2r(x)$.

  Tornando al risultato che vogliamo dimostrare, possiamo scrivere la relazione

$$2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$$

  Con una semplice manipolazione matematica, la disequazione precedente risulta essere equivalente a $r'(z) + r(x) - 2r'(x) \leq -2$. Quindi, concentriamoci sul dimostrare questo risultato.
  Sfruttando la definizione di rank e le proprietà dei logaritmi, possiamo dire che $r'(z) + r(x) - 2r'(x) = \log((s'(z)/s'(x)) \cdot (s(x)/s'(x))) = \log(A \cdot B)$ dove, per semplicità, abbiamo definito $A = s'(z)/s'(x)$ e $B = s(x)/s'(x)$. A questo punto, possiamo notare che $s'(x) \geq s'(z) + s(x)$ e quindi vale l'espressione $A + B = (s'(z) + s(x))/s'(x) \leq 1$. Con dei semplici calcoli, è possibile far vedere che, data l'ipotesi $A + B \leq 1$, il prodotto $A \cdot B$ è massimo quando $A = B = \frac{1}{2}$.
  Quindi, tornando al risultato che volevamo dimostrare, risulta:

$$r'(z) + r(x) - 2r'(x) = \log(A \cdot B) \leq \log(\frac{1}{2} \cdot \frac{1}{2}) = -2$$

  In questo modo, abbiamo completato la dimostrazione di questo caso;

- **Caso 3 (Zag-Zig):** il tempo ammortizzato è al massimo $3(r'(x) - r(x))$.
  La dimostrazione di questo passaggio è equivalente a quello precedente.
  In questo passaggio, vengono effettuate due rotazioni e dato che, solo i nodi x, y e z possono cambiare il loro rank, il tempo ammortizzato è:

$$a = 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

Dato che $r'(x) = r(z)$, risulta che $a = 2 + r'(y) + r'(z) - r(x) - r(y)$. Inoltre, dato che $r'(x) \geq r'(y)$, risulta che $a \leq 2 + r'(x) + r'(z) - r(x) - r(y)$. Infine, dato che $r(y) \geq r(x)$, risulta che $a \leq 2 + r'(x) + r'(z) - 2r(x)$.

Tornando al risultato che vogliamo dimostrare, possiamo scrivere la relazione

$$2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$$

Con una semplice manipolazione matematica, la disequazione precedente risulta essere equivalente a $r'(z) + r(x) - 2r'(x) \leq -2$. Quindi, concentriamoci sul dimostrare questo risultato.
Sfruttando la definizione di rank e le proprietà dei logaritmi, possiamo dire che $r'(z) + r(x) - 2r'(x) = \log((s'(z)/s'(x)) \cdot (s(x)/s'(x))) = \log(A \cdot B)$ dove, per semplicità, abbiamo definito $A = s'(z)/s'(x)$ e $B = s(x)/s'(x)$. A questo punto, possiamo notare che $s'(x) \geq s'(z) + s(x)$ e quindi vale l'espressione $A + B = (s'(z) + s(x))/s'(x) \leq 1$. Con dei semplici calcoli, è possibile far vedere che, data l'ipotesi $A + B \leq 1$, il prodotto $A \cdot B$ è massimo quando $A = B = \frac{1}{2}$.
Quindi, tornando al risultato che volevamo dimostrare, risulta:

$$r'(z) + r(x) - 2r'(x) = \log(A \cdot B) \leq \log(\frac{1}{2} \cdot \frac{1}{2}) = -2$$

In questo modo, abbiamo completato la dimostrazione di questo caso.

A questo punto, analizziamo un'operazione di Splaying completa che richiede almeno k passi, con $k \geq 1$. In questo contesto, nel peggiore dei casi, sappiamo che il passo k sarà un caso Zig e che tutti gli altri passi sono casi Zig-Zig o casi Zag-Zig. Quindi, la somma telescopica risulta essere:

$$3r_k(x) - 3r_{k-1}(x) + 1 + \sum_{j=1}^{k-1}(3r_j(x) - 3r_{j-1}(x)) = 3r_k(x) - 3r_0(x) + 1$$

Questo risultato è esattamente quello che volevamo dimostrare: dopo k passi il nodo x si trova al posto di t e quindi $r_k(x) = r(t)$. Infine, utilizzando le notazioni asintotiche, possiamo riscrivere questo risultato come $O(\log(s(t)/s(x))$.

□

Osservazione 1. *Con un ulteriore raffinamento è possibile far vedere che il tempo ammortizzato del caso 3 (Zag-Zig) è al massimo* $2(r'(x) - r(x))$.
Per cui risulta che il caso 2 (Zig-Zig) è quello più costoso.

Osservazione 2. *Se imponiamo* $w(x) = 1\ \forall x \in T$, *allora* $s(t) = n$ *e* $s(x)$ *è almeno uguale a 1. Quindi risulta che* $O(\log(s(t)/s(x)) = O(\log\ n)$.

Osservazione 3. *La diminuzione massima del potenziale su una sequenza di accessi è al massimo* $\sum_{x=1}^{n} \log(W/w(x))$ *con* $W = \sum_{x=1}^{n} w(x)$, *dato che* $\forall x \in T$ *la size* $s(x)$ *è al massimo* W *e almeno* $w(x)$.

1.4 Teoremi Fondamentali sugli Splay Tree

Il Lemma 1 dimostrato nella sezione precedente è valido per ogni valore positivo assegnato a $w(x)$. In questa sezione continuiamo lo studio sugli Splay Tree osservando quali risultati sorprendenti possiamo ottenere cambiando i pesi $w(x)$.
In generale, consideriamo una sequenza di m accessi su uno Splay Tree con n nodi. Vediamo quindi i risultati più importanti:

Teorema 1 (**Balance Theorem**). *Il tempo totale di accesso su tutta la sequenza di operazioni è*

$$O((m+n)\log\ n + m)$$

Proof. Assegniamo a ogni elemento il peso di $1/n$. Allora $W = 1$ e, grazie al Lemma 1, possiamo dire che il tempo di accesso ammortizzato è $3log\ n + 1$ per ogni elemento. Quindi su tutta la sequenza, il tempo di accesso è $3mlog\ n + m$. Inoltre, basandoci sull'Osservazione 3, possiamo dire che il calo netto di potenziale è al massimo $O(n \log\ n)$. Mettendo insieme questi risultati in notazione asintotica, otteniamo l'enunciato del teorema.

□

Teorema 2 (**Static Optimality Theorem**). *Per ogni elemento x, sia* $q(x)$ *la frequenza di accesso all'elemento x, ovvero il numero di volte in cui x è stato acceduto. Se ogni elemento viene acceduto almeno una volta, allora il tempo totale di accesso su tutta la sequenza è*

$$O(m + \sum_{x=1}^{n} q(x) \log(\frac{m}{q(x)}))$$

Proof. Assegniamo a ogni elemento il peso di $q(x)/m$. Allora $W = 1$ e, grazie al Lemma 1, diciamo che il tempo di accesso ammortizzato è $3log(m/q(x))+1$ per ogni elemento. Quindi sulla sequenza, il tempo di accesso è $3\sum_{x=1}^{n} q(x)\log(m/q(x)) + m$. Inoltre, basandoci sull'Osservazione 3, possiamo dire che il calo netto di potenziale è al massimo $\sum_{x=1}^{n}\log(m/q(x))$. Mettendo insieme questi risultati otteniamo l'enunciato del teorema.

□

Teorema 3 (**Static Finger Theorem**). *Assumiamo che gli elementi dell'albero siano numerati da 1 a n nell'ordine in cui appaiono nella visita simmetrica. Siano* $x_1,\ x_2,\ \ldots\ , x_m$ *gli elementi ordinati acceduti nella sequenza di accessi. Se* f *è un qualsiasi elemento fissato, il tempo totale di accesso su tutta la sequenza è*

$$O(n\log\ n\ +\ m\ +\ \sum_{i=1}^{m}\log(|x_i\ -\ f|\ +\ 1))$$

Proof. Assegniamo a ogni elemento il peso di $1/(|x\ -\ f|\ +\ 1)^2$.
Allora $W = \sum_{x=1}^{n}\frac{1}{(|x\ -\ f|\ +\ 1)^2}$. Questo valore può essere stimato con l'utilizzo delle serie p, con p=2, da cui otteniamo $W \approx \sum_{k=1}^{\infty}\frac{1}{k^2} = O(1)$. Grazie al Lemma 1, possiamo dire che il tempo ammortizzato per l'accesso i è $6log(|x_i\ -\ f|\ +\ 1)\ +\ 1$. Quindi su tutta la sequenza, il tempo di accesso è $6\sum_{i=1}^{m}\log(|x_i\ -\ f|\ +\ 1)\ +\ m$. Inoltre, basandoci sull'Osservazione 3 e dato che il peso di ogni elemento è almeno $1/n^2$, possiamo dire che il calo netto di potenziale è al massimo $O(n\log\ n)$. Mettendo insieme questi risultati otteniamo l'enunciato del teorema.

□

Teorema 4 (**Working Set Theorem**). *Numeriamo gli accessi da 1 a m. Per ogni accesso i, sia* $t(i)$ *il numero di elementi differenti acceduti prima dell'accesso i dall'ultima occorrenza dell'elemento* x_i*, o dall'inizio della sequenza se l'elemento x viene acceduto per la prima volta. Il tempo totale di accesso su tutta la sequenza è*

$$O(n\log\ n\ +\ m\ +\ \sum_{i=1}^{m}\log(t(i)\ +\ 1))$$

Proof. Assegniamo a ogni elemento il peso di $1/(t(i)\ +\ 1)^2$ dove i fa riferimento all'accesso i in cui viene acceduto x. Allora, i pesi possibili sono $1,\ 1/4,\ 1/9,\ \ldots\ ,\ 1/n^2$ e vengono assegnati in ordine decrescente: quindi l'elemento più recente avrà peso massimo 1 mentre gli elementi mai acceduti avranno un peso basso.

Inoltre, supponiamo che il peso di x_i durante l'accesso i è $1/k^2$; allora dopo l'accesso, assegniamo il peso 1 a x_i e, ad ogni altro elemento x avente peso $1/(k')^2$, con $k' < k$, assegniamo il peso $1/(k' + 1)^2$. In questo contesto, $W = \sum_{x=1}^{n} \frac{1}{(t(i) + 1)^2}$. Questo valore può essere stimato con l'utilizzo delle serie p, con p=2, da cui otteniamo $W \approx \sum_{k=1}^{\infty} \frac{1}{k^2} = O(1)$. Grazie al Lemma 1, possiamo dire che il tempo ammortizzato per l'accesso i-esimo è $6log(t(i) + 1) + 1$. Quindi su tutta la sequenza, il tempo di accesso è $6 \sum_{i=1}^{m} \log(t(i) + 1) + m$. Inoltre, basandoci sull'Osservazione 3 e dato che il peso di ogni elemento è almeno $1/n^2$, possiamo dire che il calo netto di potenziale è al massimo $O(n \log\ n)$. Mettendo insieme questi risultati otteniamo l'enunciato del teorema.

□

1.4.1 Implicazioni dei Teoremi Presentati

A questo punto, analizziamo i significati dei vari teoremi e le loro implicazioni:

- **Balance Theorem:** questo teorema afferma che, su una sequenza sufficientemente lunga di accessi, uno Splay Tree è efficiente quanto un qualunque albero bilanciato;
- **Static Optimality Theorem:** questo teorema afferma che uno Splay Tree si adatta in modo ottimale in seguito ai vari accessi. Infatti, se un nodo x viene richiamato molte volte, il termine $m/q(x)$ risulta piccolo, riducendo il suo peso sul tempo totale di accesso;
- **Static Finger Theorem:** questo teorema afferma che, se gli accessi sono concentrati attorno a un nodo fissato f, il costo totale è basso perché la distanza tra i nodi $|x_i - f|$ è piccola, mentre se gli accessi sono sparsi in tutto l'albero, il costo sarà più elevato.
- **Working Set Theorem:** questo teorema afferma che il tempo di accesso a un elemento può essere stimato come $\log((t(j) + 1)$, ovvero come il logaritmo di uno più il numero di elementi diversi acceduti dall'ultima volta che l'elemento considerato è stato richiamato. In altre parole: gli elementi a cui si accede più di recente (Working Set), sono i più facili da accedere.

1.5 Operazioni sugli Splay Tree

Utilizzando la procedura di Splaying, possiamo implementare tutte le operazioni classiche che vengono effettuate sui BST standard. In particolare, siano T, T1 e T2 tre Splay Tree. Allora, possiamo effettuare le seguenti operazioni:

- ***access*(x, T) :** partiamo dalla radice di T cercando x come in BST normale. Se troviamo il nodo x, completiamo l'operazione eseguendo uno Splaying del nodo e restituendo il puntatore a x. Se raggiungiamo un nodo NULL (x non è presente in T),completiamo l'operazione eseguendo uno Splaying dell'ultimo nodo non NULL visitato e restituiamo un puntatore a NULL;

- ***insert*(x, T) :** assumiamo che x non sia in T. Allora effettuiamo l'operazione di ricerca di x, sostituiamo il puntatore NULL con l'elemento che vogliamo inserire e infine, effettuiamo l'operazione di Splaying su questo nodo;

- ***findMax*(T) :** partiamo dalla radice di T e, come in BST, cerchiamo l'elemento più grande in T (elemento più a destra nell'albero) e effettuiamo lo Splaying di questo nodo;

- ***join*$(T1, T2)$:** assumiamo che tutti gli elementi di $T1$ siano più piccoli di quelli di $T2$. Effettuiamo l'operazione $findMax(T1)$ e sia $x1$ la nuova radice di $T1$ dopo questa procedura. Allora $x1$ non avrà un sotto-albero destro, in quanto è l'elemento più grande in $T1$. Completiamo l'operazione di join rendendo l'intero albero $T2$ figlio destro di $x1$ e restituendo l'albero risultante;

- ***split*(x, T) :** effettuiamo l'operazione di $access(x, T)$. A questo punto, rompiamo un arco tra la radice di T (elemento x) e uno dei suoi figli. Se la radice del nuovo sotto-albero è minore di x, allora il sotto-albero diventa $T1$ e il rimanente diventa $T2$; altrimenti viceversa. Infine, restituiamo i due alberi risultanti;

- ***delete*(x, T) :** assumiamo che x sia presente in T. Allora effettuiamo l'operazione di $access(x, T)$ e, una volta che x è la nuova radice, lo eliminiamo. Così facendo, vengono generati due alberi, diciamo $T1$ e $T2$, i cui elementi sono tutti rispettivamente minori di x e maggiori x. Quindi, completiamo l'operazione effettuando la procedura di $join(T1, T2)$ e restituendo il risultato.

Dopo aver visto come è possibile implementare queste operazioni, analizziamo il loro costo temporale. In particolare, per ogni elemento x nell'albero T, siano x^- e x^+ rispettivamente l'elemento che precede e l'elemento che segue x in T in ordine di visita simmetrica. Se x^- non è definito, allora diciamo che $w(x^-) = \infty$; inoltre, se x^+ non è definito, allora diciamo che $w(x^+) = \infty$. Allora, vale il seguente lemma:

Lemma 2 (**Update Lemma**). *Sia W la somma dei pesi degli elementi presenti nell'albero o negli alberi. Allora i tempi ammortizzati delle operazioni negli Splay Tree hanno i seguenti upper bounds:*

$$access(x,T): \quad \begin{cases} 3\log(\frac{W}{w(x)}) \ + \ 1, & \textit{se } x \textit{ è in } T \\ 3\log(\frac{W}{\min\{w(x^-), w(x^+)\}}) \ + \ 1, & \textit{se } x \textit{ non è in } T \end{cases}$$

$$findMax(T): \quad 3\log(\frac{W}{w(x)}) \ + \ 1, \quad \textit{dove } x \textit{ è il più grande elemento in } T$$

$$join(T1,T2): \quad 3\log(\frac{W}{w(x)}) \ + \ O(1), \quad \textit{dove } x \textit{ è il più grande elemento in } T$$

$$split(x,T): \quad \begin{cases} 3\log(\frac{W}{w(x)}) \ + \ O(1), & \textit{se } x \textit{ è in } T \\ 3\log(\frac{W}{\min\{w(x^-), w(x^+)\}}) \ + \ O(1), & \textit{se } x \textit{ non è in } T \end{cases}$$

$$insert(x,T): \quad 3\log(\frac{W \ - \ w(x)}{\min\{w(x^-), w(x^+)\}}) \ + \ \log(\frac{W}{w(x)}) + \ O(1),$$

$$delete(x,T): \quad 3\log(\frac{W}{w(x)}) \ + \ 3\log(\frac{W \ - \ w(x)}{w(x^-)}) \ + \ O(1),$$

Proof. L'operazione principale da analizzare è $\boldsymbol{access(x,T)}$. Quindi, partendo dal Lemma 1, sappiamo che il costo per questa operazione è $3\log(s(t)/s(y)) \ + \ 1$ dove y è il nodo sul quale viene applicata la procedura di Splaying.
In questo contesto, dobbiamo analizzare separatamente il caso in cui l'elemento x che stiamo cercando è presente nell'albero e il caso in cui questo elemento non è presente. In particolare, se x è presente nell'albero, è anche in y, e quindi risulta che $s(y) \geq w(x)$.

Se invece x non è presente nell'albero, allora o x^- o x^+ sono sicuramente nel sotto-albero radicato in y, e quindi risulta che $s(y) \geq \min\{w(x^-), w(x^+)\}$.
Con queste osservazioni, abbiamo dimostrato il costo di questa operazione.

L'analisi appena completata è equivalente a quella per l'operazione di ***split*(x, T)** in quanto consiste semplicemente in una procedura di $access(x, T)$ con alcune operazioni aggiuntive di costo costante.

Anche l'operazione di ***findMax*(T)** ha un limite equivalente a quello dell'operazione di $access(x, T)$ con x presente in T.

Per quanto riguarda l'operazione di ***join*$(T1, T2)$**, il limite presente nell'enunciato del lemma deriva direttamente dal limite della procedura di $findMax(T1)$, per cui possiamo scriverlo come $3\log(s(t1)/w(x)) + 1$, dove $t1$ è la radice di $T1$.
Inoltre, in questo contesto, l'incremento di potenziale causato dall'unione degli alberi $T1$ e $T2$ è dato da $\Phi_{dopo} - \Phi_{prima} = \log(W) - \log(s(t1)) \leq 3\log(W/s(t1))$ dove $W = s(t1) + s(t2)$.
Sommando il costo della prima operazione e l'incremento di potenziale otteniamo:

$$3\log(\frac{s(t1)}{w(x)}) + 3\log(\frac{W}{s(t1)}) + 1 = 3\log(\frac{s(t1)}{w(x)} \cdot \frac{W}{s(t1)}) + 1$$

da cui deriva il limite enunciato nel lemma.

Il limite per l'operazione di ***insert*(x, T)** deriva direttamente dal limite definito per l'operazione di $access(x, T)$ con x non presente in T. E' importante notare che il valore W, presente nella formula, si riferisce al peso totale dell'albero dopo l'inserimento; quindi per il costo di questa operazione andiamo a togliere il peso del nodo da inserire $w(x)$.
Inoltre, per lo stesso ragionamento della procedura di $join(T1, T2)$, l'incremento del potenziale è $\Phi_{dopo} - \Phi_{prima} = \log(W) - \log(w(x))$ con $W = s(t) + w(x)$.
Sommando questi termini e il costo dell'operazione di salvataggio delle informazioni del nodo da inserire, otteniamo il limite enunciato nel lemma.

Infine, l'operazione di ***delete*(x, T)** si articola in due fasi: la prima consiste nell'operazione $access(x, T)$ con x presente nell'albero; la seconda consiste nell'operazione di $join(T1, T2)$. Tuttavia in questa seconda fase bisogna tenere conto che il peso totale dell'albero non è più W perché abbiamo rimosso x; inoltre, il nodo con valore massimo da cercare nell'operazione di $findMax(T)$ non è x ma x^-.
Tenendo conto di queste accortezze, il limite dell'enunciato è immediato.

□

Capitolo 2

Analisi comparativa delle prestazioni degli Splay Tree

2.1 Introduzione allo studio

In questo capitolo viene condotta un'analisi comparativa tra i costi degli Splay Tree e dei BST statici. L'obiettivo di questo confronto consiste nel misurare le prestazioni delle due strutture dati in diversi scenari.
I risultati ottenuti forniranno un quadro chiaro delle situazioni in cui gli Splay Tree sono preferibili o meno rispetto ai BST tradizionali. Per rendere più concreto questo confronto, all'inizio di ogni scenario, le strutture iniziali delle varianti di alberi saranno uguali tra di loro, pur potendo variare da uno scenario all'altro. In questo contesto, il numero di nodi n resta invariato, con $n = 2^{14} - 1$ (16383).

A supporto di questa analisi, presentiamo anche dei grafici esplicativi che illustreranno il comportamento delle due strutture in vari scenari e in funzione di diversi parametri. In particolare, ogni grafico presentato sarà strutturato come segue:

- **Asse X:** variabile a seconda degli scenari
- **Asse Y:** operazioni eseguite (archi attraversati + rotazioni). In particolare, come nella teoria, ogni rotazione effettuata è contata come 1 operazione.

Un lettore attento potrebbe chiedersi come mai è stato utilizzato il numero di operazioni e non il tempo reale per effettuarle. In generale, il tempo impiegato può essere soggetto a numerosi fluttuazioni che potrebbero alterare il risultato dello studio (overhead del linguaggio utilizzato, processi attivi sul computer, implementazioni diverse...); per questo si è preferita un'unità di misura indipendente.

Inoltre, al fine di evitare risultati troppo "artificiali" o troppo soggetti al "rumore" degli altri processi presenti sul computer, i dati mostrati rappresentano una media dei valori ottenuti, ripetendo i casi di studio più volte.

Prima di procedere con l'analisi dei vari scenari di studio è importante fare una precisazione. Nel seguente studio, ci si concentra sull'operazione di $access(x, T)$ che, come visto nel capitolo precedente, è alla base di tutte le altre procedure.

2.2 Analisi degli scenari

Per comprendere meglio le prestazioni degli Splay Tree rispetto ai BST, è utile esplorare vari scenari di accesso che evidenziano i comportamenti delle due strutture. In particolare, i seguenti esperimenti si concentrano principalmente sulle conseguenze operative in contesti di località temporale (Working Set Theorem) e località spaziale (Static Finger Theorem), che rappresentano le situazioni più realistiche.

2.2.1 Applicazione del Working Set Theorem

In questo paragrafo viene mostrato il risultato teorico enunciato nel Working Set Theorem. In particolare, il tempo totale di una sequenza di m accessi su uno Splay Tree con n nodi è $O(n \log n + m + \sum_{i=1}^{m} \log(t(i) + 1))$ dove $t(i)$ indica il numero di elementi differenti acceduti prima dell'accesso i dall'ultima occorrenza dell'elemento x_i, o dall'inizio della sequenza se l'elemento x viene acceduto per la prima volta.

Nell'ambito di questo teorema, vengono presentati tre grafici che mettono in luce i vantaggi degli Splay Tree per contesti di località temporale.

Come è stato eseguito questo studio?
Tutti i grafici sono soggetti alle stesse condizioni iniziali: i nodi sono $n = 2^{14} - 1$ e gli alberi iniziali sono perfettamente bilanciati.

Inoltre, il numero di accessi m, eseguiti in tutti gli scenari, varia secondo una progressione di potenze di 2, partendo da $m = 2^{12}$ fino a $m = 2^{21}$, con un incremento costante della potenza di 2 ad ogni passaggio (come si può vedere nei grafici sottostanti). Questo implica che le curve degli alberi, che descrivono il numero di operazioni effettuato, cresce in modo esponenziale da un passaggio all'altro: questa condizione è stata scelta per osservare il comportamento degli alberi per un numero di accessi molto elevato.

La differenza tra i tre scenari si trova nella dimensione scelta del Working Set:

- Grafico 2.1: la dimensione del Working Set è pari allo $0,1\%$ di n (circa 16);
- Grafico 2.2: la dimensione del Working Set è pari all' 1% di n (circa 163);
- Grafico 2.3: la dimensione del Working Set è pari allo 10% di n (circa 1638).

In questo contesto, il Working Set è definito all'inizio dell'esperimento e i nodi a cui accedere sono scelti casualmente all'interno di questo insieme. In questo modo, è facile osservare fino a dove si spingono i vantaggi dello Splay Tree.

Quali sono i risultati di questo studio?

Osservando i 3 grafici, il primo risultato che appare evidente è il vantaggio dello Splay Tree in confronto al BST. Infatti, nonostante il numero di operazioni sia molto alto, la curva dello Splay Tree si mantiene sempre al di sotto della curva del BST e si può osservare che, più aumenta il numero di accessi effettuati e più il vantaggio dello Splay Tree diventa rilevante.

Inoltre, si può osservare come varia il numero di operazioni dello Splay Tree da uno scenario all'altro in relazione alla dimensione scelta per il Working Set. Per insiemi piccoli, il vantaggio degli Splay Tree è molto grande perché i nodi acceduti si trovano vicini alla cima dell'albero. Per insiemi grandi, il vantaggio degli Splay Tree si assottiglia perché i nodi si trovano a una distanza maggiore nell'albero.

Questi risultati sono in linea con quanto descritto nella teoria. Infatti, nel Working Set Theorem si può osservare che, se si accede sempre a pochi elementi ripetutamente, il termine $t(i)$ rimane piccolo nel corso degli accessi restando quasi irrilevante. Invece, se il Working Set è grande, la sommatoria diventa sempre più pesante.

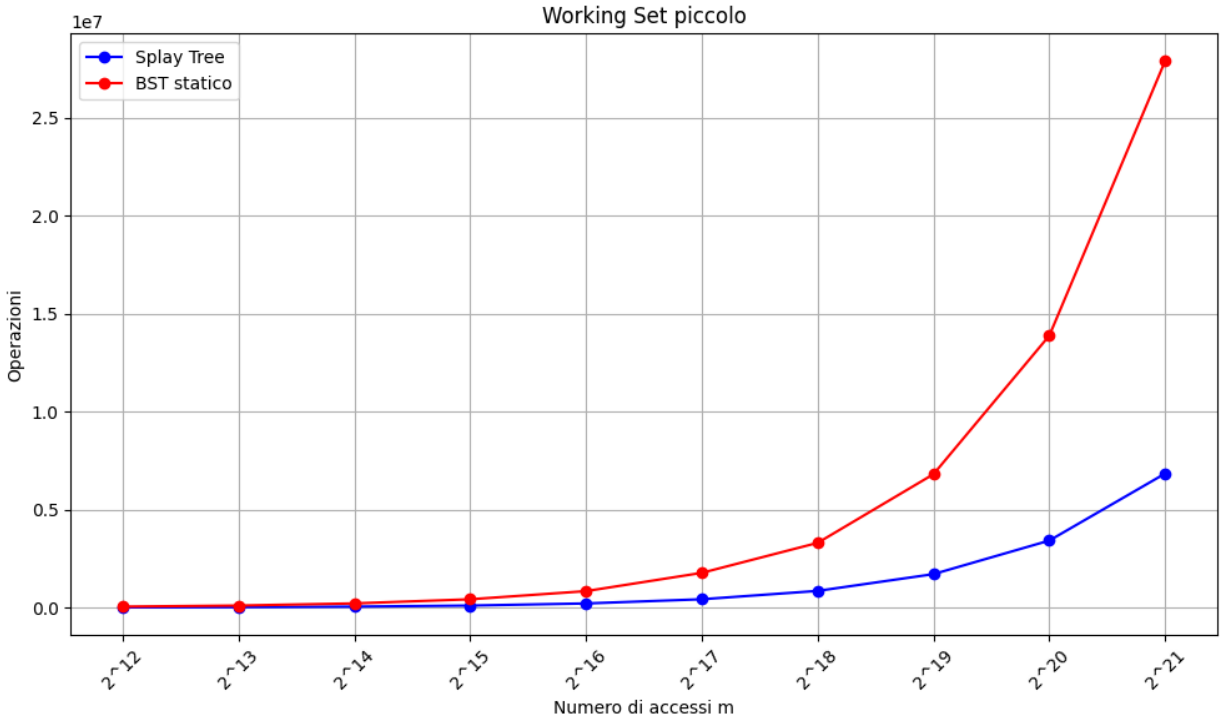


Grafico 2.1

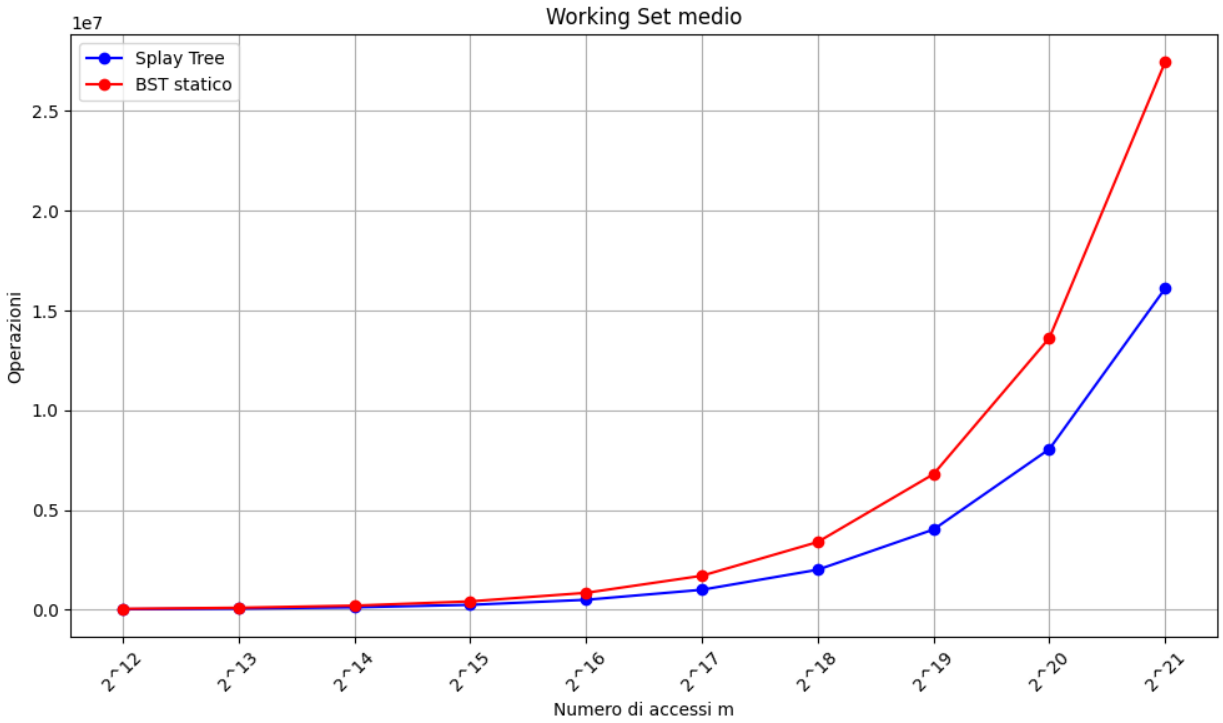


Grafico 2.2

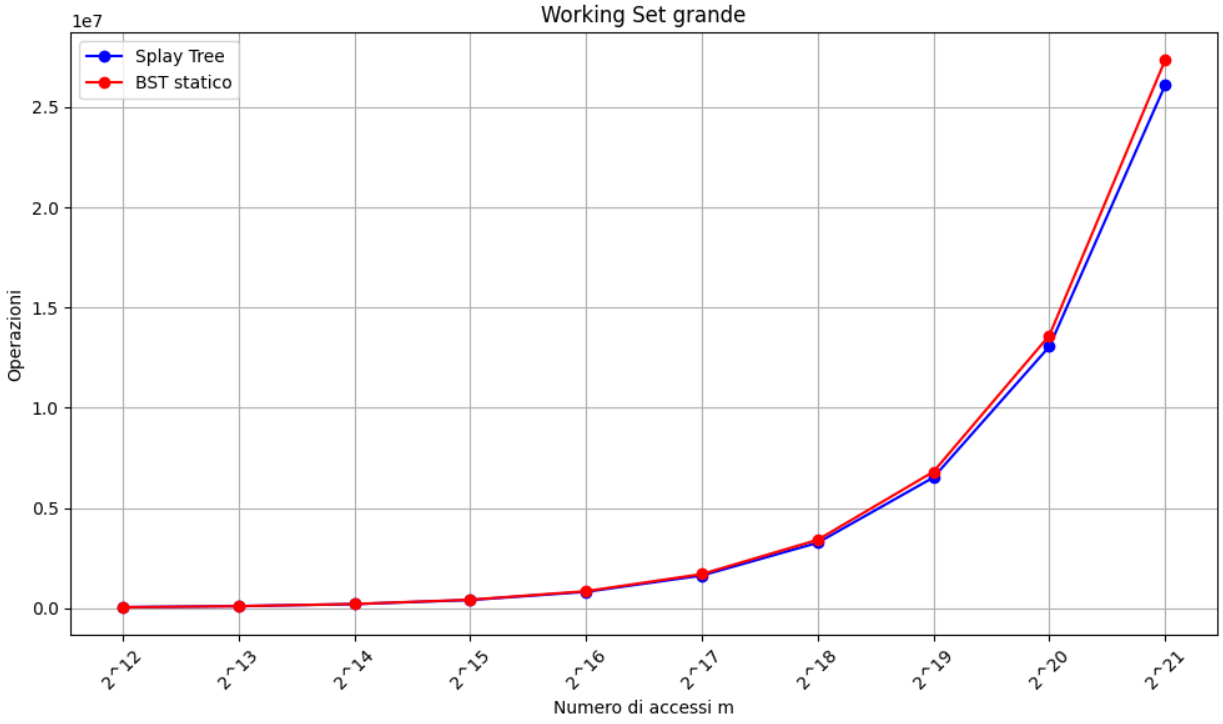


Grafico 2.3

2.2.2 Applicazione dello Static Finger Theorem

In questo pragrafo viene mostrato il risultato teorico enunciato nello Static Finger Theorem. In particolare, il tempo totale di una sequenza di m accessi su uno Splay Tree con n nodi è $O(n \log n + m + \sum_{i=1}^{m} \log(|x_i - f| + 1))$ dove f è un qualsiasi elemento fissato e x_i è l'elemento acceduto durante l'accesso i.
Nell'ambito di questo teorema, viene presentato un grafico che mette in luce i vantaggi degli Splay Tree per contesti di località spaziale.

Come è stato eseguito questo studio?
Questo grafico è soggetto alle stesse condizioni iniziali dello scenario precedente: i nodi sono $n = 2^{14} - 1$ e gli alberi iniziali sono perfettamente bilanciati.

Inoltre in questo caso, il numero di accessi m non varia ma rimane costante per tutto l'arco dell'esperimento e risulta $m = 2^{18}$. Questa scelta consente di osservare come cambia il comportamento dello Splay Tree al variare della posizione relativa dei nodi acceduti rispetto a un nodo Finger f, mantenendo fisso il carico di lavoro.

Infatti, l'elemento distintivo di questo studio è la distanza massima entro la quale i nodi da accedere vengono scelti rispetto a f: in particolare, si accede a nodi casuali in un intervallo centrato su f, con ampiezza variabile. Quindi, l'esperimento inizia con una distanza massima pari a 1200, che viene poi decrementata di 10 a ogni step, fino a raggiungere la distanza minima pari a 950. In questo modo, si riduce progressivamente l'intervallo di accesso intorno al Finger. Questi valori sono stati scelti per evidenziare il cambio di tendenza del grafico, sottolineato nella sezione successiva.

Quali sono i risultati di questo studio? Dal grafico risultante si può osservare un comportamento coerente con quanto previsto dallo Static Finger Theorem. Nella fase iniziale dell'esperimento, quando la distanza massima dal nodo Finger è molto alta, i nodi acceduti si distribuiscono su un ampio intervallo dell'albero. Man mano che la distanza massima dal nodo Finger si riduce, la selezione dei nodi acceduti si concentra progressivamente in un intorno più ristretto. Questo implica che i nodi sono, in media, più vicini a f e, di conseguenza, anche il numero di operazioni dello Splay Tree diminuisce visibilmente.

Questi risultati sono in linea con quanto descritto nella teoria. Infatti, nello Static Finger Theorem si può osservare che, se si accede sempre ai nodi contenuti in un intervallo limitato, la distanza $|x_i - f|$ si riduce, portando a un vantaggio crescente nello Splay Tree.

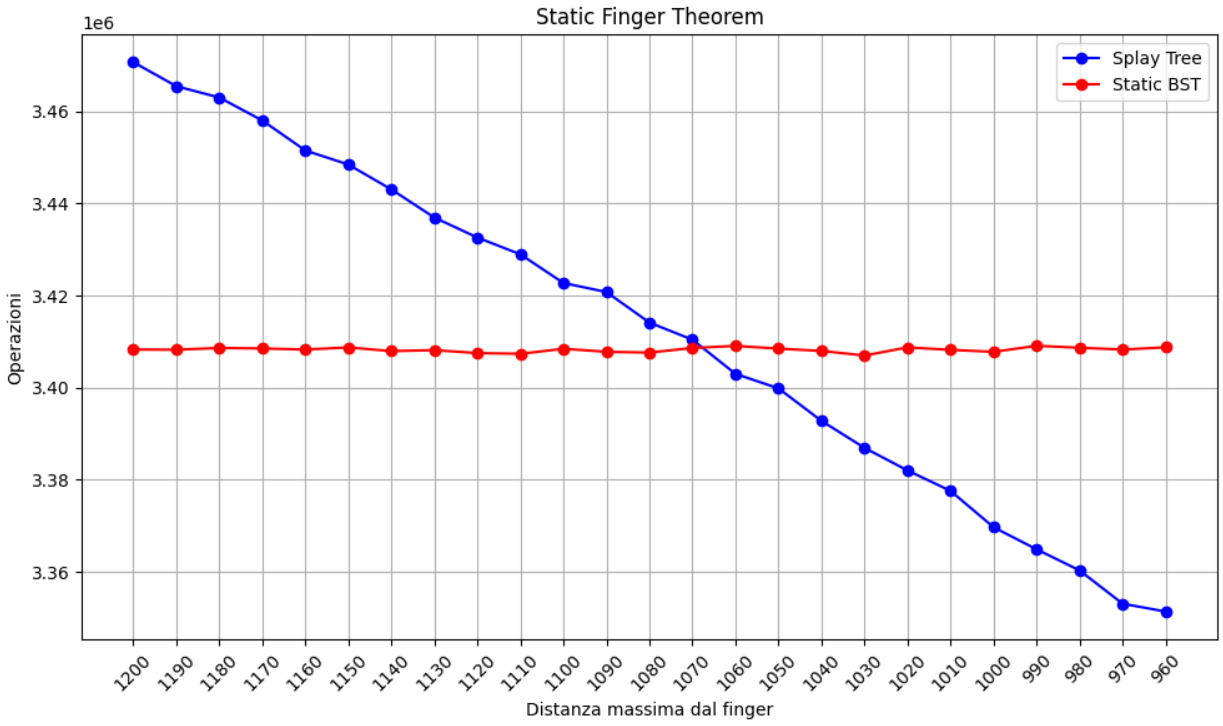

Static Finger Theorem
1e6
Splay Tree
Static BST
Operazioni
3.46
3.44
3.42
3.40
3.38
3.36
1200
1190
1180
1170
1160
1150
1140
1130
1120
1110
1100
1090
1080
1070
1060
1050
1040
1030
1020
1010
1000
990
980
970
960
Distanza massima dal finger

Grafico 2.4

Conclusioni

In questo progetto abbiamo analizzato in dettaglio gli Splay Tree, introdotti da Daniel Sleator e Robert Tarjan nel 1985. Ne abbiamo messo in evidenza le proprietà di auto-bilanciamento e i risultati teorici che ne garantiscono un costo ammortizzato ottimo per le operazioni fondamentali. A differenza di altri alberi di ricerca bilanciati, gli Splay Tree non mantengono dei fattori di bilanciamento espliciti, ma si affidano a una particolare ristrutturazione per ottimizzare le operazioni successive.

Nella parte teorica abbiamo presentato l'operazione fondamentale di Splaying che rappresenta il cuore di questa struttura. Inoltre, abbiamo approfondito i teoremi principali che ne caratterizzano il comportamento. Infine, tramite le relative dimostrazioni, abbiamo compreso come gli Splay Tree riescano a mantenere un costo logaritmico, nonostante la mancanza di informazioni sull'equilibrio dell'albero.

Nella parte sperimentale abbiamo proposto un confronto pratico con i BST statici per quanto riguarda l'operazione principale di accesso, sia da un punto di vista teorico sia tramite grafici esplicativi. Abbiamo condotto gli esperimenti variando il numero di accessi e utilizzando pattern ripetitivi per simulare scenari realistici. In particolare, l'efficacia degli Splay Tree è risultata evidente in contesti di località temporale (Working Set Theorem) e località spaziale (Static Finger Theorem). I risultati emersi confermano che gli Splay Tree rappresentano una soluzione ottima e spesso preferibile ad altre strutture dati. La loro capacità di auto-organizzarsi li rende non solo efficienti, ma anche "intelligenti" nella gestione di dati realistici.

In una prospettiva futura sarà interessante approfondire ulteriormente l'utilizzo degli Splay Tree in strutture dati più complesse come i Lexicographic Search Tree o i Link/Cut Tree dove l'operazione fondamentale di Splaying assume sempre più importanza. Inoltre, a distanza di decenni dalla loro introduzione, gli Splay Tree continua- no a suscitare interesse nella ricerca della "struttura perfetta". Alcune congetture fondamentali, come la Dynamic Optimality Conjecture, restano ancora aperte e sono dimostrate solo parzialmente. Queste ipotesi suggeriscono che gli Splay Tree sono in grado di eseguire qualunque sequenza di accessi con un costo vicino al minimo possibile, rendendole molto affascinanti nell'ambito delle strutture dati.

Esattamente come noi, gli Splay Tree imparano dal passato per ottimizzare il futuro, adattandosi continuamente al contesto in cui si trovano. Forse è questo il loro insegnamento più profondo: l'intelligenza non consiste nell'essere perfetti fin dall'inizio, ma sta nel sapersi evolvere ogni volta, adattandosi alla sfida che ci si pone davanti.

Bibliografia

[1] Goemans, M. X. (1994). *Advanced Algorithms*. Massachusetts Institute of Technology, Lecture Notes.

[2] Karger, D., & Zhang, X. (2005). *Splay Trees*. Appunti del corso Advanced Algorithms (6.854J), Massachusetts Institute of Technology.

[3] Mazzariello, C., & Sansone, C. (2024). *Alberi auto-aggiustanti*. Dispensa didattica per il corso di Algoritmi e Strutture Dati, Università degli Studi di Napoli Federico II.

[4] Nelson, J., & Tiffany, S. (2017). *Advanced Algorithms*. Appunti del corso CS 224, Harvard University.

[5] Sleator, D. D., & Tarjan, R. E. (1985). *Self-Adjusting Binary Search Trees*. Journal of the ACM, 32(3), 652–686.

[6] Stanford University. (2014). *Splay Trees*. Appunti del corso CS166, Department of Computer Science.

Sitografia

[1] University of San Francisco. *Splay Tree Visualization*. https://www.cs.usfca.edu/~galles/visualization/SplayTree.html

[2] TutorialsPoint. *Splay Trees in Data Structures*. https://www.tutorialspoint.com/data_structures_algorithms/splay_trees.htm

[3] Wikipedia. *Splay Tree*. https://en.wikipedia.org/wiki/Splay_tree

Ringraziamenti

Scrivo i ringraziamenti perché si sa, a parte pochissime persone costrette, nessuno legge il resto della tesi. Quindi ora che ho la vostra attenzione, ricominciamo: gli Splay Tree sono stati introdotti per la prima volta da Daniel Sleator e Robert Tarjan nel 1985 e...

No scherzo. Non vi romperò con altre rotazioni e simboli strani ma se non sapete neanche di cosa parlo e vi è venuta un po' di curiosità, fate sempre in tempo a tornare all'inizio.

Chi mi conosce veramente sa che questa è la parte più difficile da scrivere ma ci provo veramente: grazie a chi mi ha accompagnato in questo viaggio dentro a quello strano mondo che permette ai nostri telefoni di funzionare, ma che nessuno vuole veramente capire. Ora, per vostra fortuna, sono pronto a venire in vostro soccorso quando il computer non vi funzionerà e mi guarderete stupiti dopo che avrò semplicemente schiacciato qualche tasto. So che mi vorrete sempre a disposizione ma spero di non limitarmi a questo.

Grazie a chi ha finto di capire con entusiasmo mentre spiegavo per la terza volta la differenza tra uno Zig e uno Zag e a chi mi ha sopportato in quei momenti in cui fissavo lo schermo sparendo dal mondo (spesso facendo finta di fare qualcosa).

Grazie alla mia famiglia (e soprattutto a mia sorella sennò chi la sente) che mi ha supportato e ha imparato a decifrare le mie frasi piene di termini tecnici come fossero normali conversazioni.

Grazie a tutti i miei amici veri, che sono riusciti a distrarmi un po' da tutto questo, e ai miei compagni di viaggio con cui ho riso nonostante avessimo la testa piena di algoritmi nel tentativo di capire questi alberi ma anche la vita vera.

Grazie al Prof. Gualà per la disponibilità che mi ha dimostrato e la pazienza che ha avuto con me durante la scrittura di questo progetto.

Grazie a me, per tutte le volte in cui avrei potuto mollare ma semplicemente non l'ho fatto. In fondo dai, dietro questa tesi non c'è solo tanta fatica ma anche crescita, tante risate e cavolate.

E adesso potete veramente smettere di leggere. Sia che avete letto tutta la tesi o solo questa parte, grazie per esserci stati e per essere arrivati fin qui (anche solo per curiosità).