Red-Black Trees - Rotations

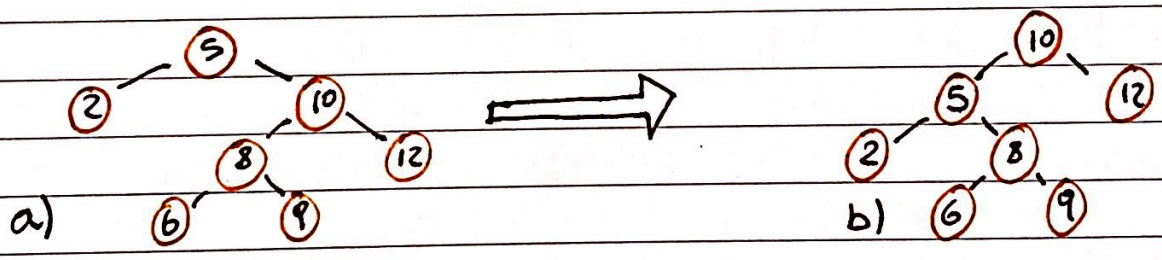
La Rotazione è un'operazione importante usata nelle operazioni di inserimento ed eliminazione degli elementi.

Rotazione

1) Altera la struttura di un albero riordinando i sottoalberi.
2) L'obiettivo è diminuire l'altezza dell'albero.
   - Negli alberi R-B l'altezza massima è di $O(\log n)$
   - Possiamo diminuire questa altezza muovendo gli alberi più grandi in alto e quelli più piccoli in basso.
3) Le rotazioni non modificano l'ordine degli elementi

Tipi di rotazione Left - Right

1) Left Rotation



2) Right Rotation

La Rotazione a destra è l'opposto della sinistra, infatti partendo dall'albero b) otteniamo a).

Complessità temporale $O(1)$

Questo perchè stiamo solo cambiando alcuni puntatori nel nostro albero.

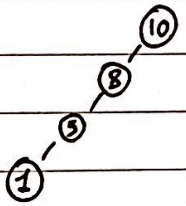
Alberi di ricerca 2-3 (Red-black)

Prima di parlare degli alberi 2-3, dobbiamo ricordare come sono composti gli alberi binari di ricerca.

BST

I BST sono semplicemente alberi binari ordinati. Ogni nodo ha due sottoalberi, dove gli elementi a sinistra di un determinato nodo sono più piccoli, e gli el. a destra sono più grandi.

Il problema principale di questa struttura è il fatto che una non è bilanciata, e potrebbe assumere questa forma:



Questa forma non è niente più che una lista! Ne per accedere ad 1, ovvero l'elemento più piccolo, dovremmo visitare ogni nodo.

La risposta a questo problema sono gli alberi di ricerca bilanciati, che garantiscono un'altezza di $O(\log n)$ per n elementi.

Albero Red-black

Un nodo è rosso oppure nero. La Root e le foglie sono sempre nere. Se un nodo è rosso, allora ~~anche~~ i suoi figli sono neri. Infine, tutti i percorsi da un nodo ai suoi discendenti foglia contiene lo stesso numero di nodi neri.

Note extra

- I nodi richiedono un bit di memoria per tenere traccia del colore.
- Il cammino più lungo non è più lungo del doppio della lunghezza del cammino più corto.
  - Cammino più lungo: Alternare rosso e nero
  - Cammino più corto: Tutti nodi neri

Operazioni

Search: L'operazione di ricerca non ha differenze

Insert e remove: Queste operazioni sono diverse, questo perché vanno a modificare la struttura.

Complessità temporale

- Search() $O(\log n)$
- insert() $O(\log n)$
- remore() $O(\log n)$

Complessità temporale

Siccome abbiamo bisogno di un solo bit per tenere traccia del colore, la complessità è $O(h)$

Red-Black Trees — Insertion

Quando inseriamo un nuovo elemento, dobbiamo essere sicuri che esso rimanga ordinato.

Strategia

1) Inseriamo z e lo coloriamo di ROSSO
2) Ricoloriamo e ruotiamo i nodi per risolvere le violazioni.

Perché rosso?

Colorando un nodo rosso, potremmo violare le proprietà 2 e 3, ma queste violazioni sono semplici da risolvere.

Pass 2: 4 Scenari

0: z potrebbe essere la radice
1: lo zio di z potrebbe essere ~~nero~~ ROSSO
2: z.uncle = black(triangle)
3: z.uncle = black(line)

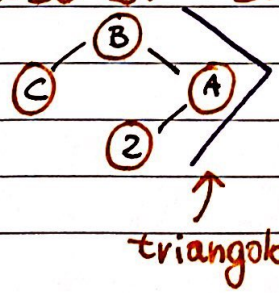
Caso 0: z = root

Tutto ciò che dobbiamo fare è colorare z di nero.

Caso 1: z.uncle = red

Per questa situazione, ricoloriamo sia genitore, zio e nonno del nodo. In questo caso il 'nonno' diventerà rosso.

Caso 2: z.uncle = black (triangle)



Quando incontriamo questo caso, effettuiamo una rotazione.