Specifica dell' albero Red-Black:
Dato che durante il corso non abbiamo affrontato esplicitamente questa struttura credo di dover aggiungere qualche parola in più sulla loro realizzazione.
I Red-Black sono degli alberi di ricerca binaria con alcune aggiunte:

1) Ogni nodo possiede un colore, o Rosso o Nero;
2) Le foglie (fittizie) sono nere;
3) I figli di un nodo Rosso devono essere entrambi Neri;
4) La radice deve essere sempre nera;
5) Per ogni nodo, tutti i percorsi semplici che vanno dal nodo alle foglie devono contenere lo stesso numero di nodi neri, questa proprietà è detta anche altezza nera.

Nella mia implementazione le foglie vengono rappresentate dal valore NULL.

Inserimento:
L'inserimento avviene nello stesso modo in cui avviene in BST "normale", ma successivamente occorre effettuare dei ribilanciamenti.
(NB: ogni nuovo nodo viene colorato di rosso per poter rilevare le violazioni nelle proprietà e ripristinarle).

In queste condizioni possiamo violare solamente la proprietà 3) e distinguere tra 3 casi particolari.
(NB: i casi da distinguere sono 6, ma "speculari" tra di loro, quindi una volta che se ne risolvono 3 gli altri si risolvo in modo analogo)

Indicherò con:
- Colore Rosso o Nero per indicare ovviamente la colorazione del nodo, utilizzerò il Blu per indicare un nodo il cui colore può essere sia rosso che nero;

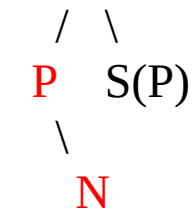
- N il nodo corrente;
- S (sibling) il fratello di N;
- S_sx e S_dx sono rispettivamente il figlio sinistro e destro di S;
- P (parent) il padre di N;
- G (grandparent) rappresenta il nonno di N;
- con X(node) una sorta di funzione che restituisce la X-esima relazione di parentela, X può essere (P, G, S).

(Indicherò le possibili violazioni nel sottoalbero sx)
Caso 1:
Sia N che che P (N) sono Rossi, indipendentemente che N sia figlio sx o dx, ed S(P(N)) è anche esso rosso.

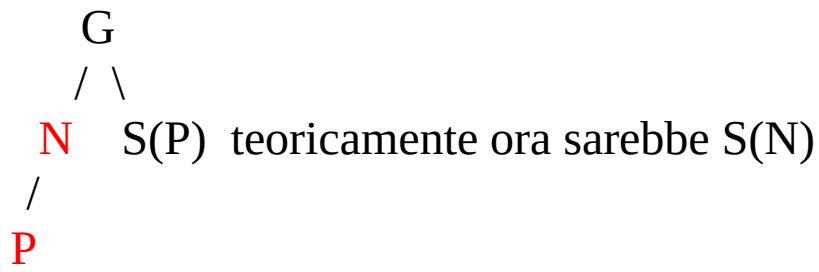
Risoluzione:
Coloro di nero P e S(P), mentre G diviene rosso, ciò ha l'effetto di propagare la violazione ai livelli superiori, inoltre se G è la radice la si lascia nera ed il ribilanciamento è completo.

Caso 2:
N e P ovviamente rossi, S(P) nero ed N è figlio dx di P.

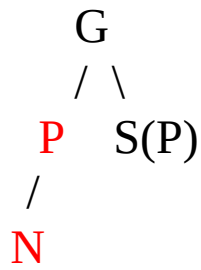
G



Risoluzione:
Effettuo una rotazione sinistra su P, in modo da “scambiarlo” con N e mantenere ugualmente la relazione d’ordine tra i nodi.
Questo caso non è risolutivo, ma “trasforma” la violazione da un caso 2 ad un caso 3!

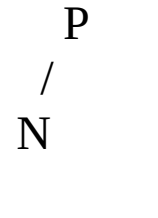


Caso 3:
N e P sono rossi, S(P) è nero ed inoltre N è figlio sx di P



Risoluzione:
Effettuo una rotazione destra su G, successivamente coloriamo G di rosso e la nuova radice di nero ora qualunque violazione prima presente nell’albero è stata rimossa.

Cancellazione:
Per la cancellazione nei Red-Black esistono varie strategie più o meno equivalenti, ma quella che ho implementato risulta la più semplice nel linguaggio C.
Iniziamo osservando che se occorre eliminare un nodo con due figli è possibile prendere il minimo nel sottoalbero destro (oppure il minimo nel sottoalbero sinistro) sostituire il valore del nodo con quello “recuperato” e successivamente eliminare il minimo (massimo), riconducendo il procedimento al successivo passaggio.
Nel caso occorre eliminare un nodo con meno di 2 figli occorre distinguere vari casi.
Se il nodo ha esattamente 1 figlio lo eliminiamo ed al suo posto e promuoviamo il figlio, colorandolo di nero.
In un caso del genere non può accadere che entrambi i nodi (padre e figlio) siano neri, e quindi non ci sono violazioni.



Non può accadere dato che supponiamo di eliminare nodi in un albero Red-Black e in un caso del genere vi sarebbe una violazione della proprietà 5) !

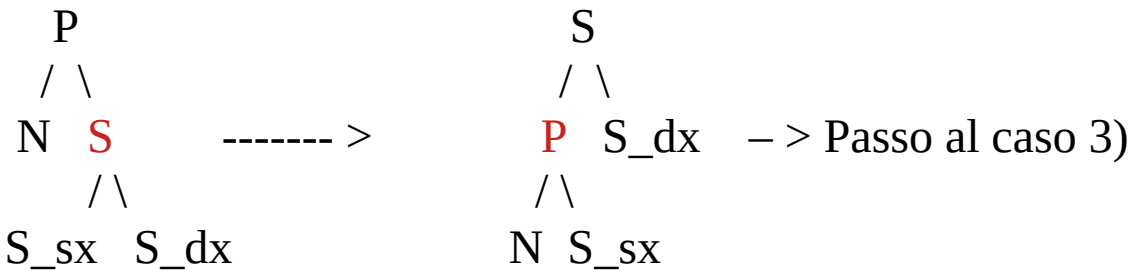
(Da notare che è possibile violare le proprietà dei Red-Black solo cancellando nodi neri.)

Se invece il nodo da eliminare non ha alcun figlio vi sono ancora 2 casi da discutere:
-Il caso in cui il nodo è rosso, è possibile eliminarlo senza violare alcuna proprietà.
-Nel caso in cui il nodo da eliminare sia nero, generiamo una violazione nell proprietà 5) ed occorre richiamare delle procedure di fixup.

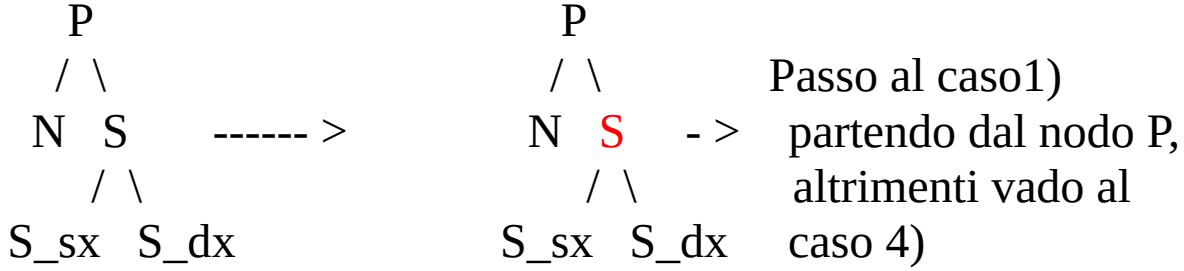
Le possibili operazioni di fixup sono 6, il codice parte sempre dal primo tipo di fixup e prosegue fino al fixup adatto al caso in questione:

1) Controllo se il nodo in questione è la radice, in questo caso termino la procedura di fixup, altrimenti passo al caso 2);

2) **SE** Il fratello del nodo è rosso, occorre scambiare il colore del fratello e del parente e ruotare intorno al padre in modo tale che il fratello diviene il nonno del nodo in questione.
IN OGNI CASO si passa al caso 3);



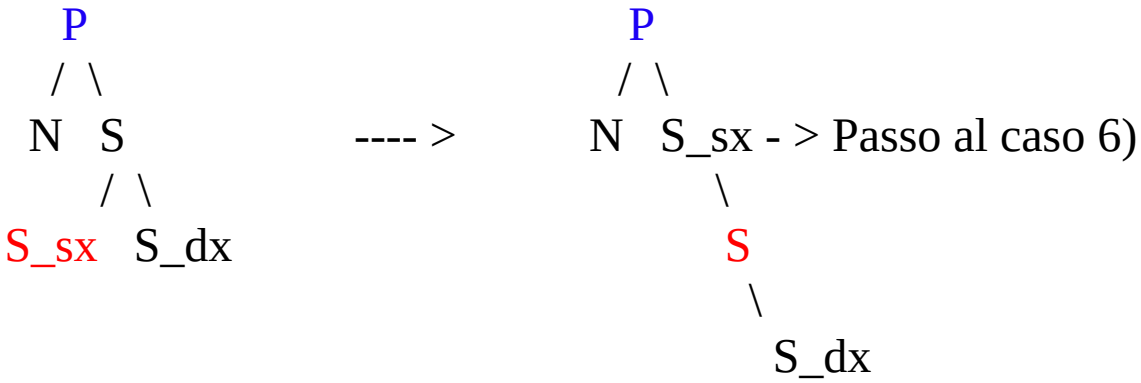
3) **SE** il parente, il fratello, ed entrambi i figli del fratello sono neri, coloro il fratello di rosso (ciò ha l'effetto di ridurre l'altezza nera) e successivamente ripartire dal tipo di correzione 1) partendo dal padre del nodo in qustione, in pratica si porta la violazione ad un livello superiore, anche se venisse eseguito sempre questo passaggio prima o poi si arriva alla radice, il che renderebbe terminale la procedura 1).
ALTRIMENTI passo al caso 4);



4) **SE** il padre del nodo è rosso e fratello con relativi figli sono neri, occorre scambiare i colori del fratello con quello del parente,

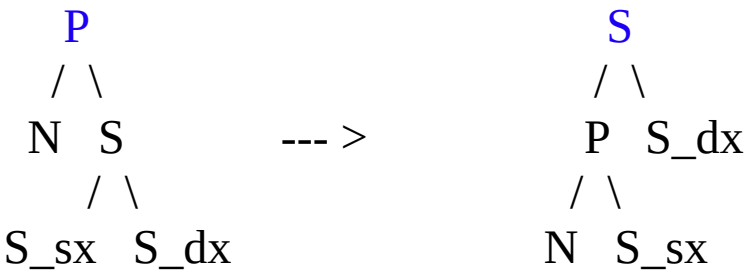
ciò ribilancia l’altezza nera lungo tutti i percosi e termina il ribilanciamento.
ALTRIMENTI occorre passare al caso 5);

5)In questo caso occorre distinguere il caso in cui il nodo analizzato sia figlio sinistro o destro.
(Esplicherò solo il caso in cui il nodo sia figlio sinistro, il caso destro è simmetrico)
SE il fratello è nero ed il suo figlio sinistro è rosso e il figlio destro è nero, scambio i colori del fratello con il suo figlio sinistro e poi effettuo una rotazione destra sul fratello del nodo, come prima non si risolvono le violazioni, ma nel momento in cui si passa al 6° e ultimo caso verranno risolte.
IN OGNI CASO si passa al caso 6);



6) Come per il caso 5 occorre distinguere il caso in cui il nodo sia figlio sinistro o destro del padre, ma sempre come il caso 5) le risoluzioni sono speculari.
SE il nodo è figlio sinistro, coloro di nero il figlio destro del fratello ed effettuo una rotazione sinistra su P.
ALTRIMENTI coloro di nero il figlio sinistro del fratello ed effettuo una rotazione destra su P.
(Osservare che questi di sopra sono gli unici casi disponibili, non può esserci una violazione che sfugge/sopravviva)

(Raffigurerò il caso sinistro)



Concludo questa sezione sul perché gli alberi Red-Black sono molto più efficienti di un BST normale.

L'altezza di un albero Red-Black è logaritmica sul numero dei nodi, più nello specifico, sia h l'altezza e n il numero di nodi si ha che:

h <= 2*lg(n + 1) , lg := logaritmo in base 2.

Inizio mostrando che per ogni nodo x, posto ad una qualunque profondità nell'albero, si ha che il numero di nodi interni sono al minimo (2^bh(x)) – 1 [con bh indica l'altezza nera del nodo].
Il modo più semplice per dimostrarlo è per induzione sull'altezza.

Caso base:
x è una foglia quindi si ha (2^0)-1=0, il che è vero.

Ipotesi di induzione:
Per un nodo di altezza x-1 vale che il numero di nodi interni è al minimo (2^bh(x-1))-1.

Caso induttivo:
Considerando il nodo x, si ha che il numero di nodi interni è la somma dei nodi nel sottoalbero sinistro + i nodi del sottoalbero più destro + 1 per il nod in questione.
Sui sottoalberi sinistro e destro è possibile usare l'ipotesi induttiva quindi, sempre considerando il nodo x, si ha che:
Il numero di nodi interni di x è almeno
((2^bh(x-1))-1) + ((2^bh(x-1))-1) + 1 = 2^bh(x)-1

che è ciò che volevo mostrare.

Per terminare questo asserto sull’altezza dei Red-Black rimane da mostrare che l’altezza è logaritmica.
Considerando un albero generico di altezza h si ha che almeno la metà dei nodi di ogni percoso semplice dalla radice ad una foglia devono essere neri, di conseguenza la bh, sempre di questo albero generico, deve essere almeno h/2.
Quindi considerando la precedente proprietà sui nodi interni e la bh si ha:

n >= 2^(h/2)-1, con un po' di algebra elementare si ottiene

h <= 2*lg(n + 1)

Ciò conclude la dimostrazione.