- همان طور که قبلا در BST گفتیم، RB-INSERT در $O(h)$ که h ارتفاع درخت است رخ می دهد که اینجا $O(\lg n)$ است.

- به این زمان باید زمان Call کردن RB-INSERT-PICKUP اضافه شود.

- هر iteration زمان $O(1)$ احتیاج دارد.
- هر iteration یا z را دو level بالا می آورد (case 1)
یا آخرین (یا یکی مانده به آخرین) مرحله اجراست (case 2)
↓
(case 3)
↓
پایان
- $O(\lg n)$ levels $\Rightarrow O(\lg n)$ time
- در کل حداکثر دو تا rotation هم داریم.

$\Leftarrow$ عمل insertion، درون red-Black-tree، $O(\lg n)$ است.

- RB-DELETE در زمان $O(\lg n)$ انجام می‌شود.

- در فراخوانی RB-DELETE-FIXUP :

- case 2 بیشترین زمان را صرف می‌کند چون x را بدون حذف یک extra black با level iteration به می‌آورد، در هر iteration

- این باعث می‌شود $O(\lg n)$ تکرار در حلقه while رخ دهد.

- هرکدام از case های ۱، ۳ و ۴ یک rotation دارند.

در نتیجه در کل کمتر یا مساوی 3 تا rotation داریم.

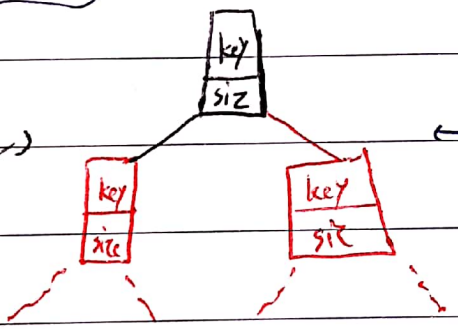
- بنابراین، زمان کل هم $O(\lg n)$ است.

Date 98/6/55 آرمین گودرزی Subject سوال ۳ تشریحی

برای تابع Find نیز می دانیم که در worst-case باید ارتفاع درخت را طی کنیم که می دانیم در RB Tree از اردر $\log n$ می باشد.

پس تا اینجا برای insert(x) ، delete(x) ، Find(x) نیازی به تغییر در RB Tree نبود. حال برای Count(x) ؛ تغییراتی در RB Tree اضافه کردن متغیری به نام size به هر node است که $size[x] = size[left[x]] + size[right[x]] + 1$

پس RB tree جدید ما به شکل زیر می آید. حال اگر



اگر تابعی به نام get_Rank(x) را تعریف کنیم که موقعیت عنصر x را در آرایه مرتب شده المنت ها به ما می دهد در واقع خروجی این تابع همان تعداد گره های کوچک تر از x می باشد.

```
get_Rank (n)
    r = siz [left[n]]+1
    y = n;
    whik (y != root)
        if (y == right[p[y]])
            r = r + size[left[p[y]]] + 1
        y = p[y]
    return r
```

حال با توجه به تابع می بینیم که worst_case زمانی است که n یکی از leaf ها باشد.

در این صورت در حلقه while ، باید ارتفاع درخت طی شود و از آنجایی که ارتفاع درخت $\log n$ است پس worst case ، $\log n$ است در نتیجه تابع get_Rank() که همان تابع count است از order $\log n$ می شود.