فرشید نوش 9831098 ۱-

مقادیری که ارتفاع درخت را زیاد می‌کنند باید یا فرزند 112 باشند یا فرزند 39.

۱) رئوسی که فرزند ۱۱۲ هستند، باید از ۹۵ بزرگتر، از ۱۱۴ کوچکتر و از ۱۱۱ بزرگتر باشند. که می‌شود:

$95 < x$ & $x < 113$ & $111 < x$ → $111 < x < 114$

(با فرض اینکه مقادیر تکراری نداریم، مقادیری هستند که می‌توانند فرزند ۱۱۲ شوند) → $x = 112$

۲) رئوسی که فرزند ۳۹ می‌شوند باید از ۹۵ کوچکتر، از ۴۰ کوچکتر، از ۳۷ بزرگتر باشند

$x < 95$ & $x < 40$ & $x > 37$ → $37 < x < 40$

با فرض تکراری نبودن مقادیر → $x = 38$, 39 → (39 آمده است) $x = 38$

۲- خیر نمی‌شود چنین درختی را پیدا کرد: چون مقادیر یکسان هستند با استقرا روی n (تعداد رئوس) اثبات می‌کنیم که هر ۲ درخت BST با مقادیر یکسان و متفاوت (یعنی value ها unique هستند) را می‌توان با دنباله‌ای از rotate ها به هم تبدیل کرد.

پایه: $n = 0$ → دو درخت با صفر رأس بدون انجام حرکتی با هم مشابه‌اند.

فرض: به ازای $K < n$ ما هر دو درخت BST با دنباله‌ی مقادیر یکسان و متفاوت (مقادیر هر درخت unique هستند) را با دنباله‌ای از rotate ها به هم تبدیل کنیم.

اثبات: ریشه‌ی درخت اول درخت دوم را در نظر بگیرید و تا زمانی که ریشه‌ی درخت اول در درخت دوم ریشه نشده بود کار زیر را می‌کنیم:

v =: ریشه‌ی درخت اول در درخت دوم

```
if ( v is left child (Par[v]))
    rightrotate for Par[v] ;
else left rotate for Par[v] ;
```

خورشیدنوش ۹۸۲۱۰۴۸

۲-

الگوریتم ۳ همینطور به بالای درخت می رود تا به ریشه برسد وقتی به ریشه رسید چون مقادیر دو درخت یکسان و unique هستند مقادیر در زیر درخت چپ ۳ با زیر درخت چپ ریشه ی درخت اول برابرند و مقادیر زیر درخت راست ۳ با مقادیر زیر درخت راست ریشه ی درخت اول یکی هستند پس فرض استقرا برای این ۲ زیر درخت قابل استفاده هست و در نتیجه این دو درخت BST به هم تبدیل پذیر گردند.

۳- برای inorder:

1) یک stack خالی دارم (S)
2) ریشه را در نقطه ای میگیریم به عنوان curr
3) curr را در S اضافه می کنم و تا زمانی که curr نال شود می کنم
curr = curr.left;
4) اگر curr نال بود و اگر S خالی نبود
الف) عنصر اول S را برمی دارم (P)
ب) P را چاپ می کنم
پ) curr = P→right;
ت) تکرار خط ۳
5) اگر curr نال و S خالی بود کار تمام است

inorder:

```
Stack s = {}
curr = root;
while (s isnot empty or curr isnot null) {
    if (curr is not null){
        s.add(curr)
        curr = curr.left
    }else{
        P = s.top
        s.pop
        Print(P)
        curr = P→right
    }
}
```

تحلیل زمانی inorder ، preorder و Postorder ←

O(n)

هر رأس حداکثر 5 بار به stack اضافه می شود
و تعداد عملیات های مربوط به هر رأس مقدار ثابتی است
و حد بالای آن به طور مثال 50 است پس هر کدام این روش ها
همگی از O(n) هستند چون هر رأس به اندازه ی مشخصی
بازدید می شوند.
(تعداد اضافه شدن و حذف شدن های رئوس از stack تعداد مشخصی است
و به طور مثال دارای حد بالای 50 است!)

Post order:
1) یک stack خالی S دارم
2) تا زمانی که root نال نشده است این کار را می کنم
a) در S بچه ی راست root را اضافه می کنم و بعد خود root را
b) root = root.left
3) رأس بالای stack را به عنوان P از S برداریم (pop کنیم)
a) root = P;
b) اگر P بچه ی راست دارد و آن بچه عنصر top S هست
then → S.pop → S.push(root) → root = root.right
c) در غیر این صورت (ادامه قبل)

Scanned with CamScanner

خورشیدنوش ۹۸۳۱۰۹۸

۳ - ادامه c :

c) در غیر این صورت (نود قبل)
then → root پرینت شود → root = null;
+) مراحل ۲ و ۳ تا زمانی که که خالی نشود تکرار شود

Post order:

```
Stack S = { root }
while ( S isnot empty) {
    while( root isnot null) {
        S.add(root.right);
        S.add( root );
        root = root.left;
    }
root = S.top;
  S.pop;

  if(root.right isnot null and root.right is S.top) {
        S.pop;
        S.add(root)
      root = root.right }
  }
  else {
    print root
    root = null
  }
```

۳-

Pre order:

1) یک Stack خالی که داریم و در آن ریشه را می‌ریزیم

2) تا زمانی که خالی نشده است این کار را تکرار کن

a) عنصر بالای که را pop کن و اسمش را p بگذار

b) print (P)   p را پرینت کن

c) S.add(P.right)   بچه‌ی راست را اضافه کن

d) S.add (P.left)   بچه‌ی چپ را اضافه کن

```
Stack S = {root}
while (S isnot empty){
  P = S.top
  S.Pop
  Print (P)
  S.add (P.right)
  S.add (P.left)
}
```

۴-

الف) فرض کنید تابع dfs خواسته‌ی مسئله را به صورت bool برمی‌گرداند (true: زیرمجموعه است، false: زیرمجموعه نیست)

```
bool dfs(node v){
   if (!T_2.contains(v.value)){
       return false;
   }else{

     bool left = true;
     bool right = true;
     if (v.left ≠ null)
       left = dfs(v.left)
     if (v.right ≠ null)
       right = dfs(v.right)
   return left & right;

}
```

تابع contains همان تابع جست و جوی عادی در BST هست که در کلاس توضیح داده شده ($O(\lg(n_2))$)

پس خواسته‌های اردر این بخش رعایت شده چون هر راس در dfs حداکثر ۱ بار بازدید می‌شود و یک $\log(n_2)$ هم در هر مرحله جست و جو می‌شود که می‌شود در این تابع $O(n_1 \log(n_2))$ و حافظه‌ی اضافی هم مقدار ثابتی است.

خورشیدنوش ۹۸۲۱۰۴۹۸

ب)

```
List1 = T1.inorder();   --> همان تابع inorder عادی است که ترتیب
List2 = T2.inorder();   --> inorder را همان BST برمی‌گرداند. (در کلاس پیاده کرده‌ایم)
P = 1;
for ( i from 1 to n1) {
    while (P <= n2 && list2[P] != List1[i]) {
        P ++;
    }
    if ( P == n2+1)
        return false;
    P ++;
}
```

i و P مستقل از هم به ترتیب حداکثر n_1 و n_2 زیاد می‌شوند ← $O(n_1+n_2)$ زمانی

List1 و List2 دارای حافظه‌ی اضافی به ترتیب n_1 و n_2 هستند ← $O(n_1+n_2)$ حافظه‌ی اضافی

۵ - اگر خروجی تابع true باشد یعنی BST بوده درخت ما اگر false بود یعنی bst نیست.

↳ bool dfs (node root)

1_ root را به عنوان ریشه در نظر بگیر

۲_ اگر root بچه‌ی راست داشت

a) اگر مقدار بچه‌ی راست کمتر از مقدار root بود خروجی false را برگردان

b) bool isright valid = dfs (root.right);

خروجی تابع را برای بچه‌ی راست را در نظر بگیر به عنوان isrightvalid.

۳_ اگر root بچه‌ی چپ داشت.

a) اگر مقدار بچه‌ی چپ بیشتر از root بود خروجی false را برگردان.

b) bool isleft valid = dfs(root.left);

خروجی تابع را برای بچه‌ی سمت چپ را به عنوان isleftvalid در نظر بگیر.

۴_ مقدار isright valid && isleft valid را به عنوان خروجی برگردان

خورشید نوش ۹۸۳۱۰۴۸

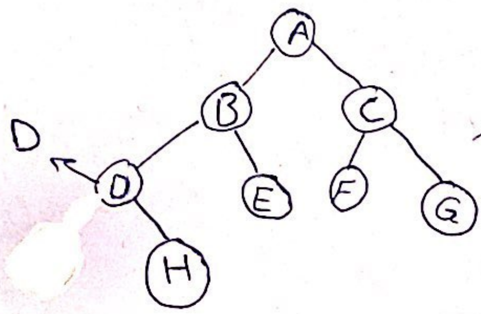
```
bool dfs(node root){
    bool isrightvalid = true;
  if (root -> right ≠ null) {
      if (root.val > root -> right.val)
           return false;
      isrightvalid = dfs(root -> right)
  }
  bool isleftvalid = true;
  if (root -> left ≠ null){
     if (root.val < root.left.val) :
         return false;
     isleftvalid = dfs(root -> left)
  }
  return isleftvalid && isrightvalid;
```

dfs(tree.root) فراخوانی اولیه.

۵- هست O(n)
چون هر رأس تعداد دستورات مشخص دارد، حداکثر ۳ بار تابع هر رأس صدا زده می‌شود.

۶-

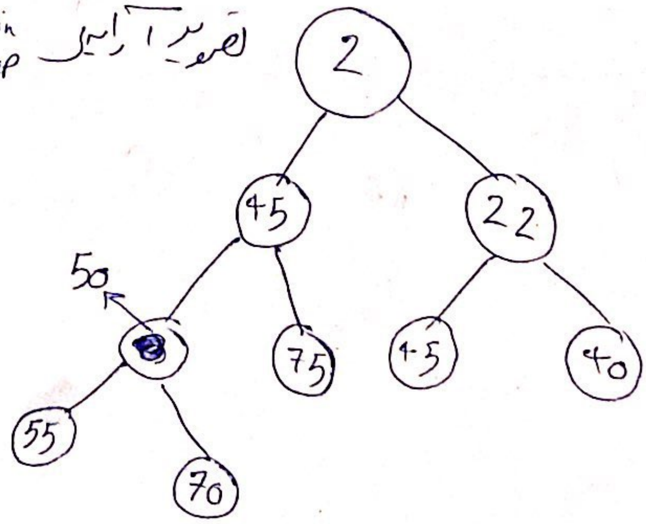
tree



راهنمایی ۱: در انتهای دنباله‌ی تولید شده.
راهنمایی ۲: زیر درخت چپ سمت چپ جایگاه ریشه در دنباله، عناصر زیر درخت راست در سمت راست جایگاه ریشه در دنباله هست.

۷-

{2, 45, 22, 50, 75, 45, 40, 55, 70} ← min Heap تصویر آرایه



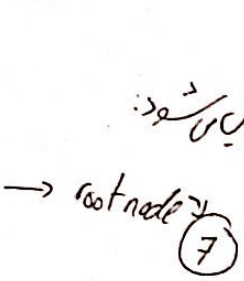
insert = { 7, 3, 18, 10, 22, 8, 11, 26, 2, 6, 13 }

Delete = { 18, 11, 3, 10, 22 }

پیاده‌سازی درخت‌ها مانند صفحه 275 کتاب CLRS
(صفحه C)

insert 7 →

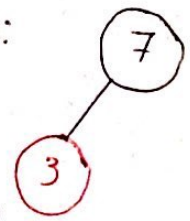
(1) اضافه کردن و دادن رنگ سیاه



رنگ ریشه به سیاه تبدیل می‌شود.

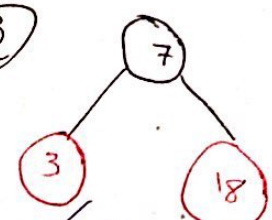
insert 3 :

(2) اضافه کردن و دادن رنگ قرمز



insert 18:

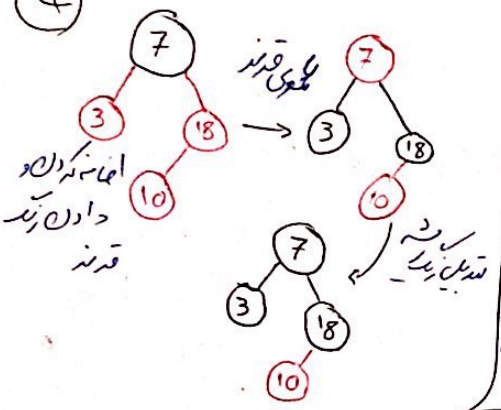
(3)



اضافه کردن و دادن رنگ قرمز

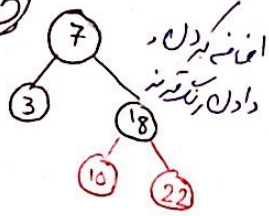
insert 10:

(4)



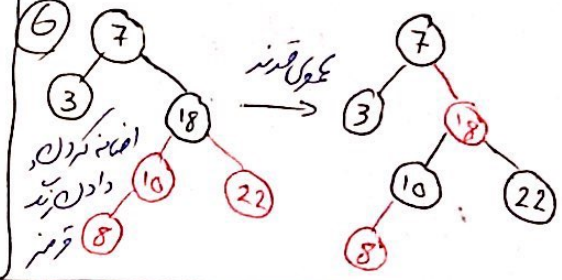
insert 22:

(5) اضافه کردن و دادن رنگ قرمز



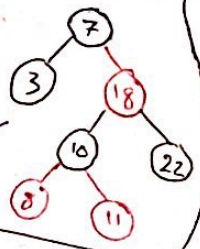
insert 8:

(6)



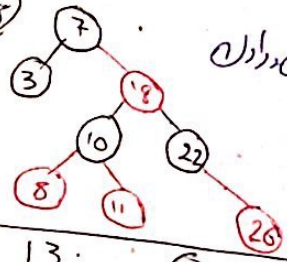
insert 11 :

(7) اضافه کردن و دادن رنگ قرمز



insert 26 :

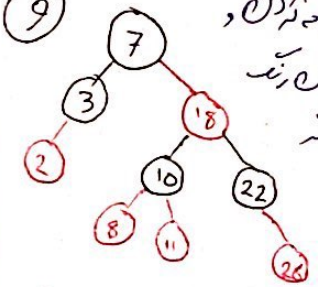
(8)



اضافه کردن و دادن رنگ قرمز

insert 2:

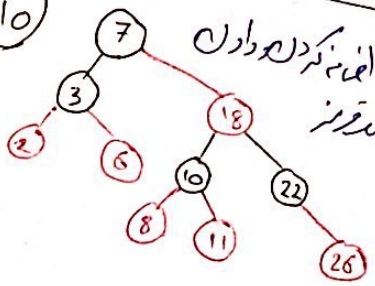
(9)



اضافه کردن و دادن رنگ قرمز

insert 6:

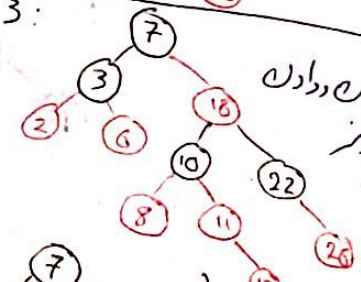
(10)



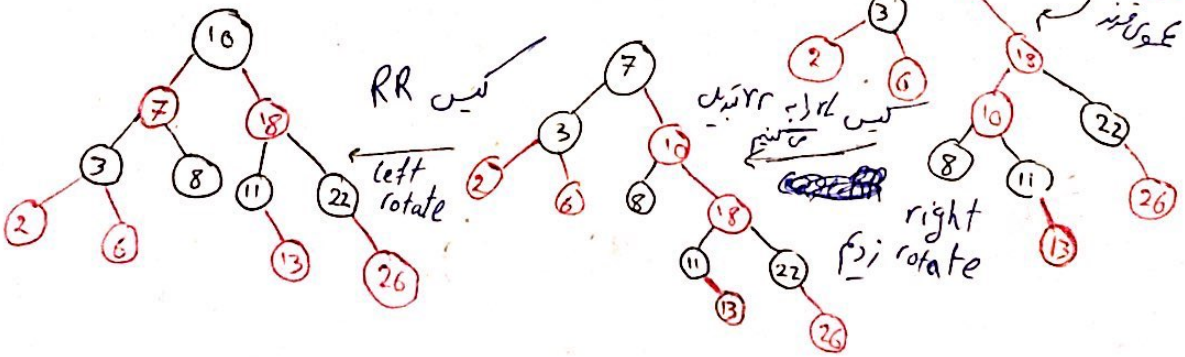
اضافه کردن و دادن رنگ قرمز

insert 13 :

(11)



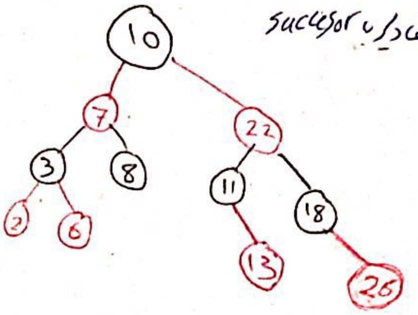
اضافه کردن و دادن رنگ قرمز



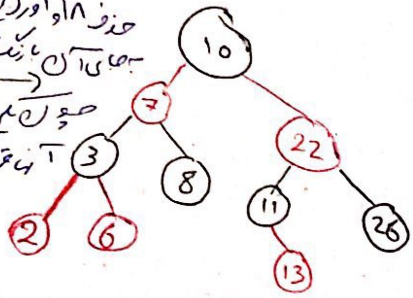
Delete :{18, 11, 3, 10, 22}

① Delete 18:

جابجایی مقدار با succesor

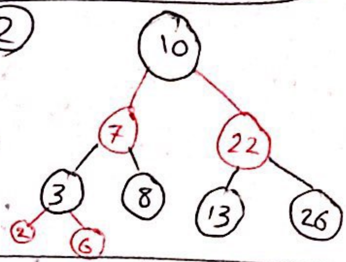


حذف ۱۸ و آوردن 26 به جای آن با رنگ مشکی چون یکی از آنها قرمز بود.



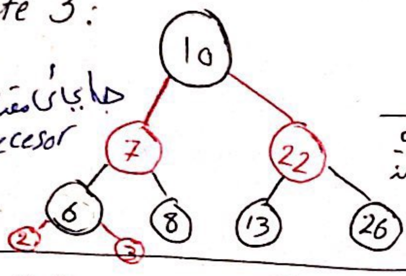
② Delete 11:

حذف حالتی که خودش فرزند قرمز داشته باشد.

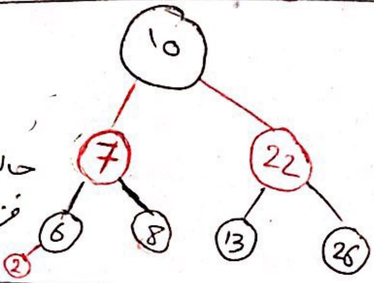


③ Delete 3:

جابجایی مقدار با succesor



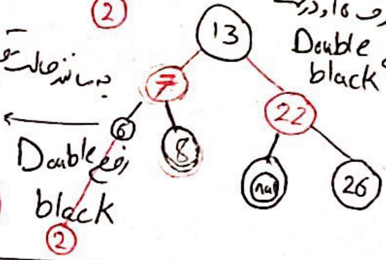
حالتی که خودش فرزند قرمز داشته باشد



④ Delete 10:

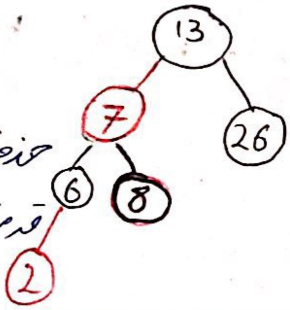
جابجایی با succesor

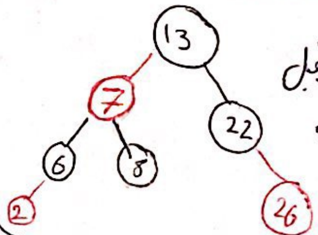
حذف 10 و در نتیجه Double black



⑤ Delete 22:

حذف حالتی که فرزند برادر قرمز یا سیاه بود.





فرشید نوش ۹۸۳۱۰۱۰۹۸

۹. تعداد رئوس مشکی بدون در نظر گرفتن خود x تا یکی از برگ ها را ارتفاع مشکی رأس x را ($bh[x]$) بگیریم. ابتدا می‌خواهیم ثابت کنیم که زیردرخت x با مقدار ارتفاع مشکی $bh[x]$ حداقل $2^{bh[x]}-1$ رأس داخلی دارد. برای اثباتش روی ارتفاع x استقرا می‌زنیم. اگر x ارتفاعش صفر باشد پس یک برگ است و $bh[x]=0$ آنگاه $2^0-1=0$ رأس داخلی در زیردرختش هست که درست است.

حال فرض کنید ارتفاع x مثبت است و یک رأس داخلی با ۲ بچه است. هر بچه یک ارتفاع مشکی به اندازه‌ی $bh[x]$ یا $bh[x]-1$ دارد. با توجه به فرض استقرا آن هم دارای تعداد رئوس داخلی حداقل $2^{bh[x]}-1$ یا $2^{bh[x]-1}-1$ دارد پس زیردرخت x حداقل

(چون ارتفاع بچه‌ها کمتر از ارتفاع x است)

$$(2^{bh[x]-1}-1)+(2^{bh[x]-1}-1)+1=2^{bh[x]}-1$$

رأس دارد که فرض استقرا را ثابت می‌کند.

حال فرض کنیم h ارتفاع درخت باشد، با توجه به ویژگی درخت rb حداقل نصف رئوس در هر مسیر از ریشه به برگ مشکی هستند (بدون در نظر گرفتن ریشه) پس $bh[root]$ باید حداقل $h/2$ باشد.

$$n \geq 2^{h/2}-1 \longleftrightarrow n+1 \geq 2^{h/2} \longleftrightarrow \lg(n+1) \geq h/2$$

$$\longleftrightarrow h \leq 2\lg(n+1)$$