بسمه تعالی

- تمرین سری سوم درس ساختمان داده ها و مبانی الگوریتم ها
- پاسخ تمرین در قالب یک فایل pdf تایپ شده یا دست نویس اسکن شده (مرتب و خوانا) و با نام StudentNumber_HW3.pdf آپلود شود.
- مهلت ارسال تمرین تا ساعت 11:59 روز سه شنبه مورخ 29 مهر 1399 می باشد.
- در صورتی که درمورد این تمرین سوال یا ابهامی داشتید با ایمیل dsfall1399@gmail.com با تدریس یاران در ارتباط باشید. لطفا برای ایمیل زدن فرمت زیر را در قسمت subject رعایت کنید:

برای سوال از مباحث مختلف:

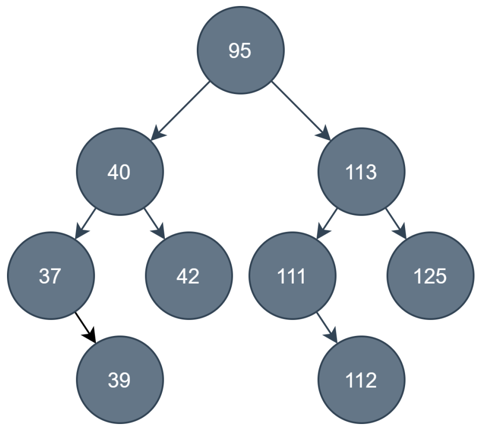
«سوال_اسم مبحث» (مثال: «سوال_رشد توابع»)

برای سوال از یک تمرین خاص:

«تمرین_شماره تمرین_شماره سوال» (مثال: «تمرین_۱_۳»)

همچنین خواهشمند است در متن ایمیل به شماره دانشجویی خود اشاره کنید.

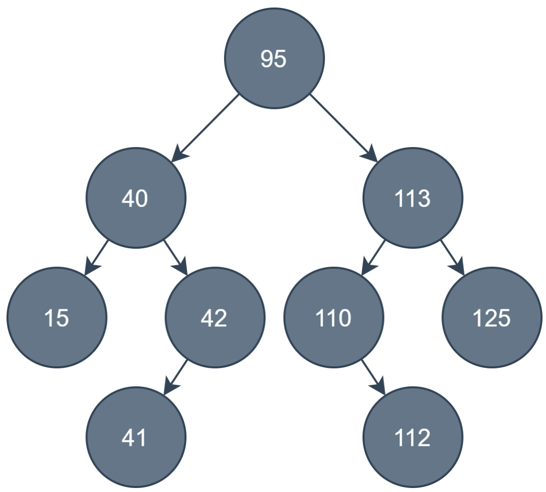
۱- درخت باینری زیر را در نظر بگیرید، بعضی از مقادیر هستند که اگر به درخت اضافه شوند، ارتفاع درخت را زیاد می‌کنند. تمام مقداری صحیحی که این موضوع برای آنها صدق می‌کند را پیدا کنید. اگر چنین مقداری وجود ندارد، علت را توضیح دهید. توجه کنید که باید هر مقدار به تنهایی ارتفاع درخت را زیاد کند و نباید چندین بار به درخت مقداری اضافه کرد.



جواب: ۳۸

۲- برای یک مجموعه از مقادیر، درخت‌های باینری زیادی وجود دارد. اگر اعمال rotation مانند آنچه که برای درخت red/black تعریف می‌شود را روی یک درخت باینری اعمال کنیم به درخت جدیدی تبدیل می‌شود. برای درخت زیر، درخت دودویی جستجوی دیگری با همین مقادیر پیشنهاد دهید که با اعمال دنباله‌ای از چرخش‌ها(left/right rotation) به آن تبدیل نشود. اگر چنین درختی وجود ندارد علت را توضیح دهید.

چنین درختی وجود ندارد زیرا:
میدانیم که یک bst را با مجموعه‌ای از چرخش‌ها می‌توانیم به یک bst زنجیره‌ای(هر عنصر فقط بچه‌ی راست داشته باشد/هر عنصر فقط بچه‌ی راست داشته باشد) با ارتفاع n تبدیل کنیم.
پس اگر مدعی هستیم که درخت a با اعمال چرخش ها به درخت b تبدیل نمی‌شود کافیست ابتدا درخت a را طی چرخش‌های R1 و درخت b را طی چرخش‌های R2 به درخت‌های راست زنجیر تبدیل کنیم(می‌دانیم که این دو درخت یکسانند). پس فقط کافیست روی درخت راست زنجیر بدست آمده برعکس اعمال R2 را اعمال کنیم تا به درخت b برسیم.



۳- یک الگوریتم غیربازگشتی برای پیمایش inorder و postorder و preorder درخت باینری ارائه دهید. (شبه کد کافیست)

پیمایش inorder با استک

پیمایش inorder بدون استک

پیمایش postorder با دو تا استک

پیمایش postorder با یک استک

پیمایش preorder با استک

۴- فرض کنید دو درخت دودویی جستجوی متوازن T_1 و T_2 به ترتیب دارای n_1 و n_2 رآس، مجموعه‌های S_1 و S_2 را نمایش می‌دهند(شبه کد الزامیست)

الف) الگوریتمی ارائه دهید که در مرتبه ی زمانی $O(n_1 log(n_2))$ و با استفاده از حافظه‌ی اضافی $O(1)$ مشخص کند که آیا $S_1 \subseteq S_2$ هست یا خیر.

الف)در درخت جستجو، عمل search از اردر $O(log(n))$ است. اگر روی اعضای S1 پیمایش کنیم $O(n_1)$ و برای هر عضو بررسی کنیم که آیا در S2 هست یا نه.

```
isSubset(T1,T2){
      for (i:1→n){//O(n1)
            x = find(T1[i],T2)//O(log(n2))
            if(!x){
                  return false;
                  }
      }
      return true;
}
```

ب) الگوریتمی ارائه دهید که در مرتبه ی زمانی $O(n_1 + n_2)$ و با استفاده از حافظه‌ی اضافی $O(n_1 + n_2)$ مشخص کند که آیا $S_1 \subseteq S_2$ هست یا خیر.

ب)برای هر دو درخت پیمایش inorder انجام می دهیم.($O(n_1 + n_2)$) سپس این دو آرایه را با یکدیگر مقایسه می‌کنیم

```
isSubset(T1,T2){
      pointer=0;
      for (i:1→n2){
            if(T1[pointer]==T2[i]){
                  if(pointer==T1.length){
                  return true;
                  }
                  Pointer++;
            }
      }
      Return false;
}
```

۵- الگوریتمی ارائه دهید که در مرتبه زمانی $O(n)$ بررسی کند که آیا یک درخت دودویی، درخت دودویی جستجو هست یا خیر.

راه اول: طبق الگوریتم زیر پیش می‌رویم که در آن با شروع از ریشه، در هر مرحله مقدار آن راس با دو فرزند آن مقایسه شده و در صورتی که مطابق تعریف درخت BST باشد به سراغ راس دیگر می‌رود.
زمان اجرای برنامه O(n) خواهد بود.

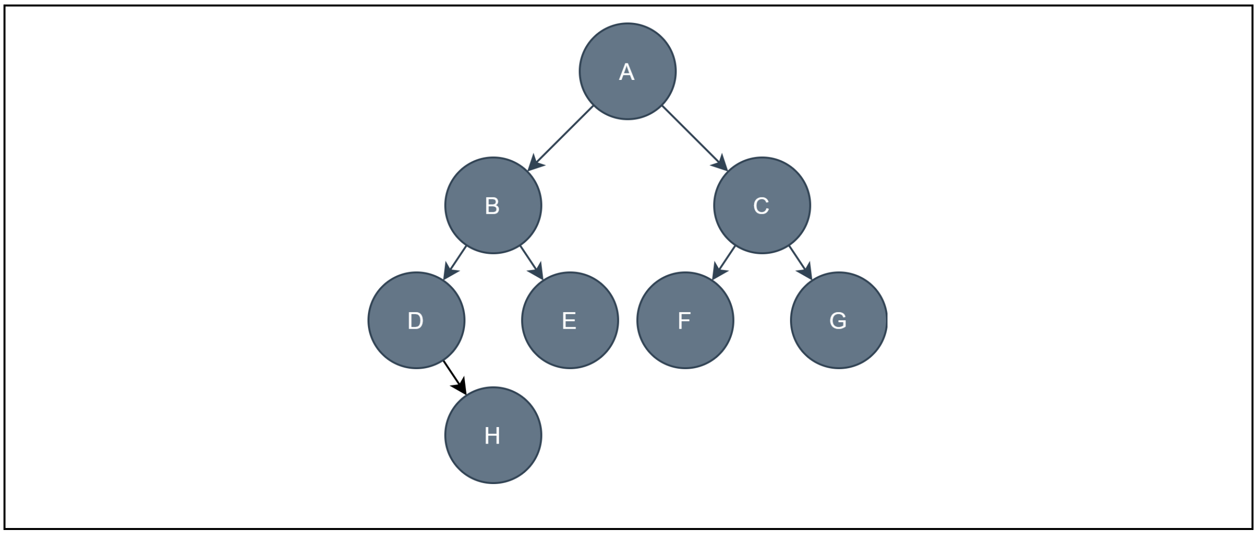
```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;


  if (node->left != NULL && node->left->data > node->data)
    return 0;


  if (node->right != NULL && node->right->data < node->data)
    return 0;

  if (!isBST(node->left) || !isBST(node->right))
      return 0;

  return 1;
}
```

راه دوم: ابتدا یک پیمایش inorder از درخت انجام می دهیم(O(n)) و مقادیر درخت را در یک آرایه می‌ریزیم.سپس بررسی می‌کنیم که آیا آرایه مرتب شده است یا خیر که این مرحله هم در O(n) انجام می‌شود.

۶- دو دنباله از پیمایش inorder و postorder یک درخت دودویی داریم. درخت اصلی را رسم کنید.

Inorder traversal =D,H,B,E,A,F,C,G

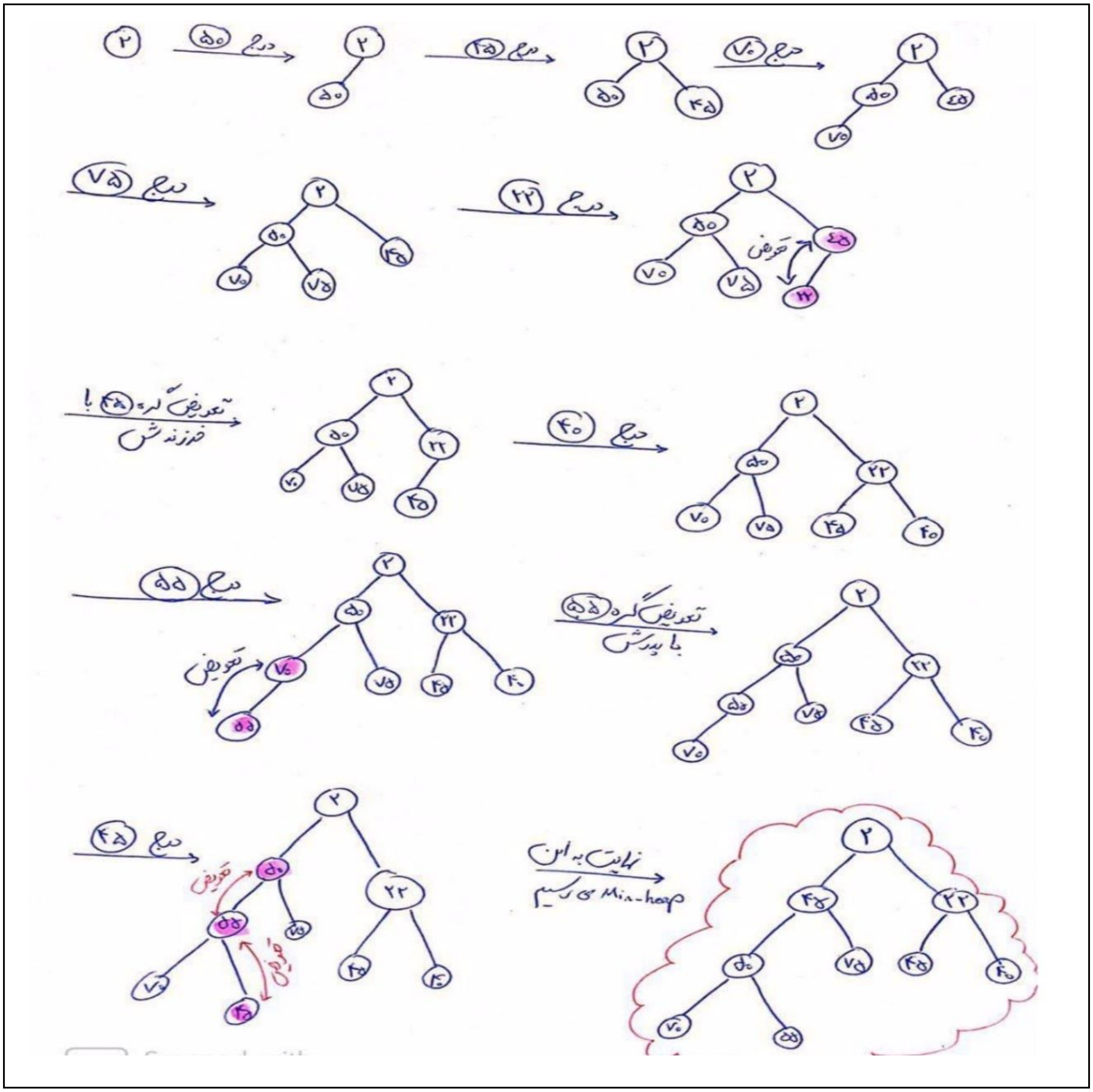
Postorder traversal = H,D,E,B,F,G,C,A



۷- با ورود مقادیر زیر به ترتیب یک MinHeap بسازید.

2,50,45,70,75,22,40,55,45

توجه کنید که اعداد به ترتیب وارد می‌شوند و فقط یک درخت MinHeap است که از این ترتیب به دست می‌آید.

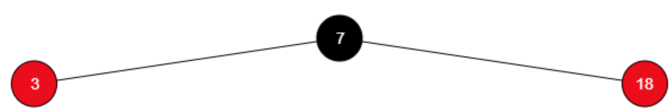
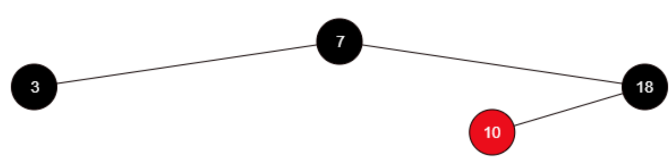


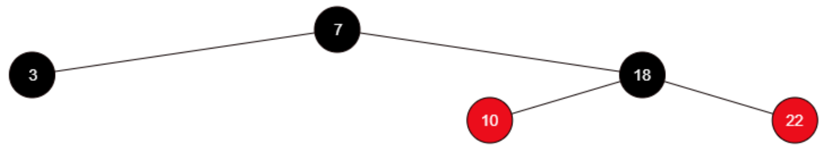
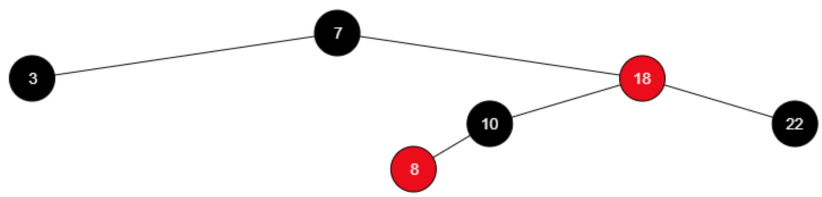
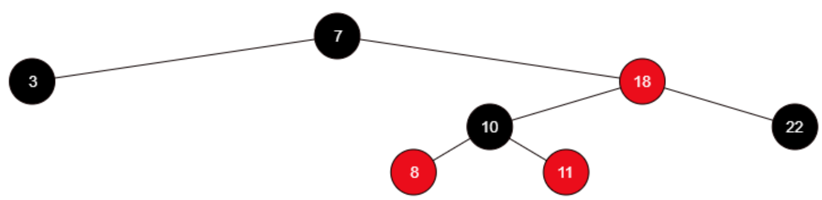
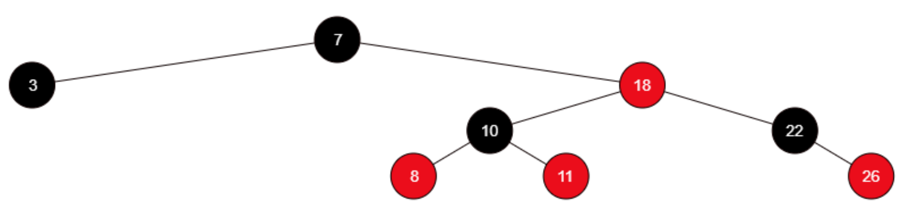
۸- مقادیر زیر را به ترتیب وارد یک درخت red/black بکنید.مراحل و case ها را در هر مرحله توضیح دهید. (اولین بار عدد ۷ وارد می‌شوند)

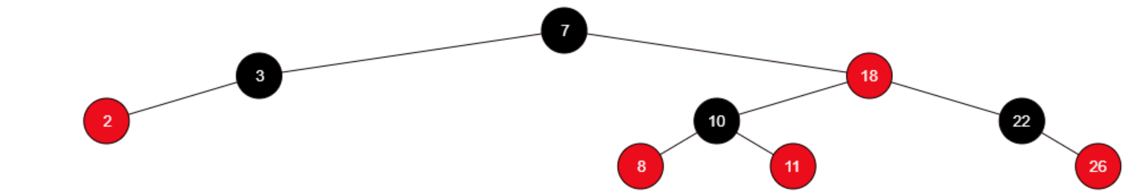
7,3,18,10,22,8,11,26,2,6,13

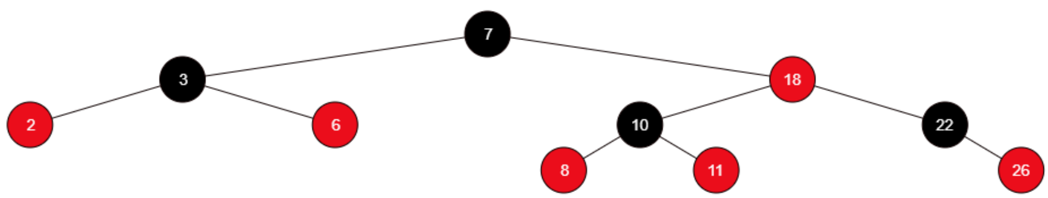
سپس مقادیر زیر را به ترتیب حذف کنید.مراحل و case ها را در هر مرحله توضیح دهید.(اولین بار عدد ۱۸ پاک می‌شود)

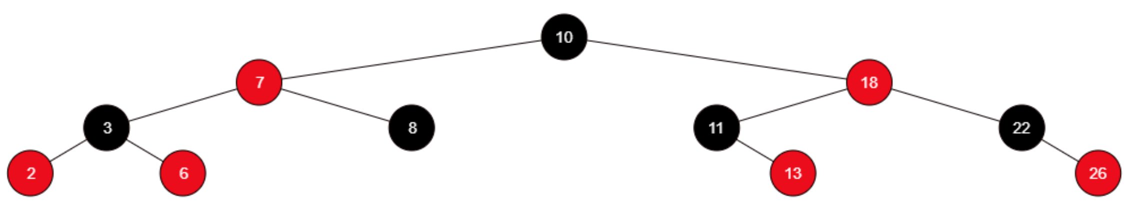
18,11,3,10,22

7, 3	 درج 7: کیس 0 (نود 7 ریشه است) درج 3: - (بابا مشکی است)
18	 درج 18: - (بابا مشکی است)
10	 درج 10: کیس 1 (بابا و عموی نود 10 قرمز است) + کیس 0 (نود 7 ریشه است)

22	 درج 22: - (بابا مشکی است)
8	 درج 8: کیس 1 (بابا و عموی نود 8 قرمز است)
11	 درج 11: - (بابا مشکی است)
26	 درج 26: - (بابا مشکی است)

<table>
<tr><td>2</td><td>

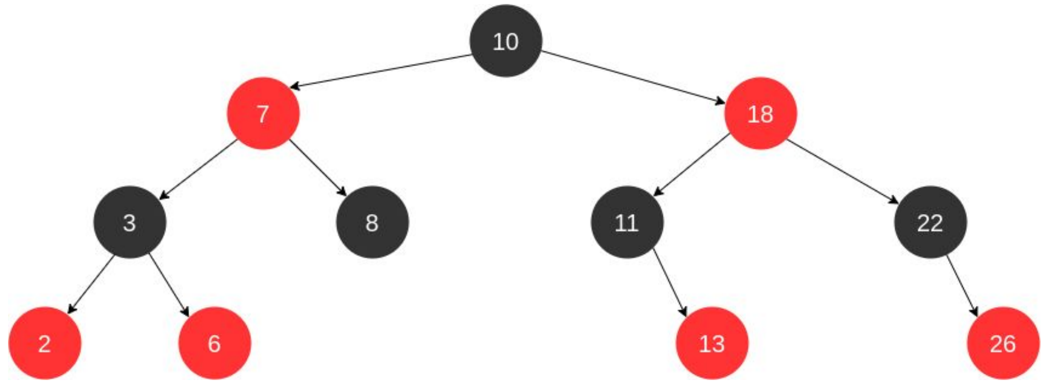
درج 2: - (بابا مشکی است)</td></tr>
<tr><td>6</td><td>

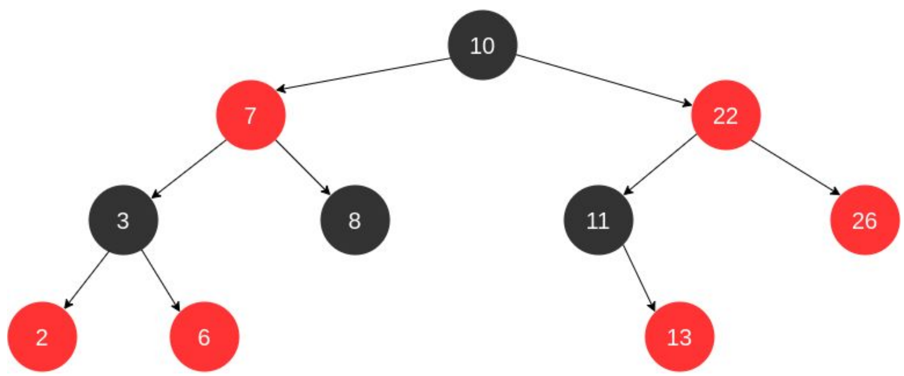
درج 6: - (بابا مشکی است)</td></tr>
<tr><td>13</td><td>

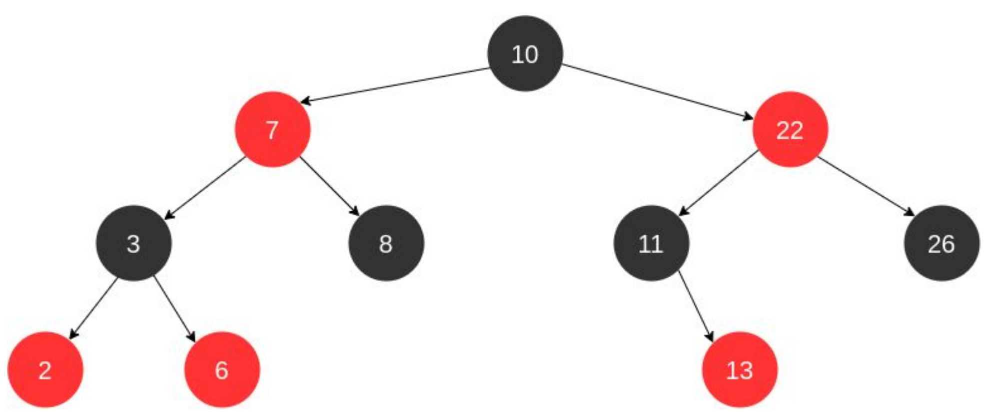
درج 13: کیس 1 (بابا و عموی نود 13 قرمز است) + کیس 2 (بابای نود 10، قرمز و عمویش مشکی است. حالت مثلث)</td></tr>
</table>

پاک کردن:

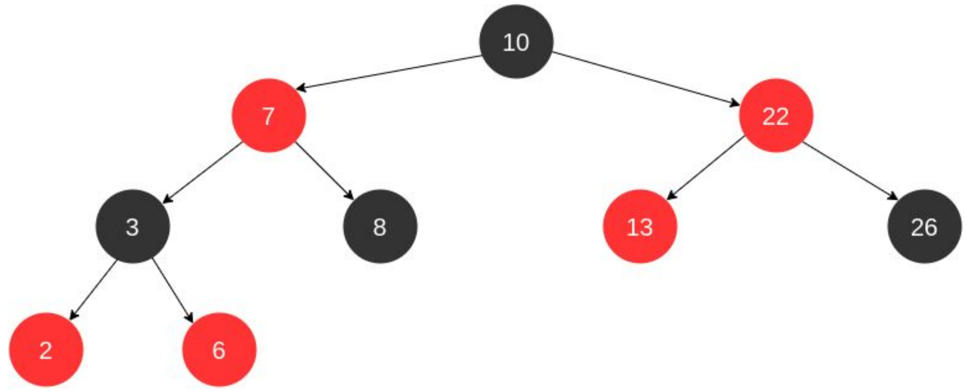
درخت اولیه:



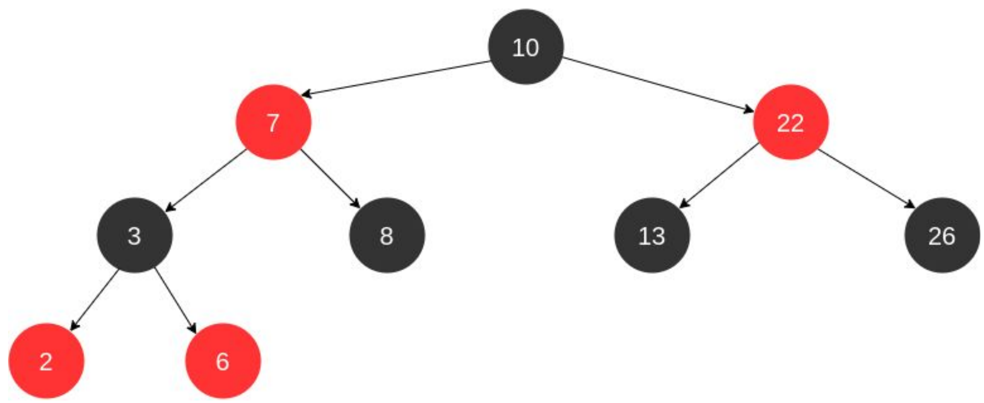
<table>
<tr>
<td>18</td>
<td>پارت ۱(حذف نود ۱۸): دوتا بچه دارد، پس کیس ۳ است. محتوای ساکسسور آن یعنی ۲۲ در آن کپی می‌شود و سپس باید آن ۲۲ اضافی را پاک کنیم که کیس ۲ است و به جایش ۲۶ می‌نشیند. حاصل:


نودی که پاک شد مشکی بود در نتیجه violation داریم. x، نود ۲۶ است و فیکس‌آپ را با آن صدا می‌زنیم.
پارت ۲ (فیکس‌آپ): کیس صفر (نقیض شرط حلقه‌ی while) حاکم است، در نتیجه کافیست x را مشکی کنیم.
حاصل:

</td>
</tr>
</table>

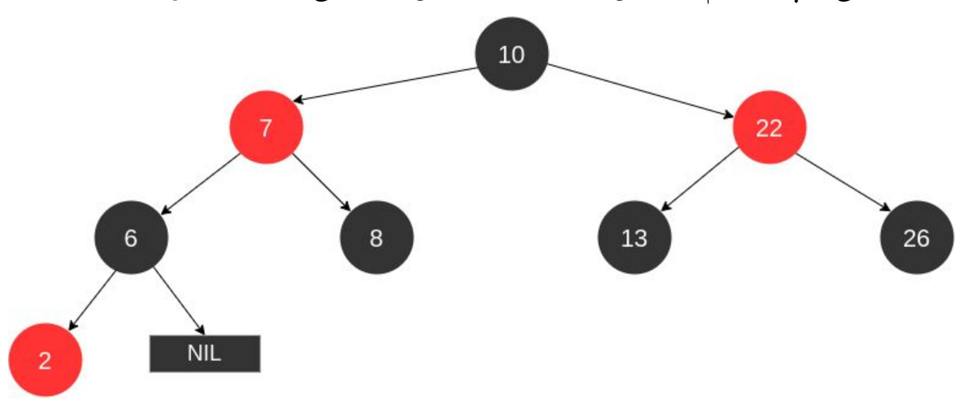
<table>
<tr><td>11</td><td>
پارت ۱ (حذف نود ۱۱): یک فرزند دارد در نتیجه کیس ۲ است. ۱۳ جایگزینش می‌شود. حاصل:



نودی که پاک شد مشکی بود، در نتیجه violation داریم. x، نود ۱۳ است و فیکس‌آپ را با آن صدا می‌زنیم.
پارت ۲ (فیکس‌آپ): کیس صفر (نقیض شرط حلقه‌ی while) حاکم است، در نتیجه کافیست x را مشکی کنیم.
حاصل:



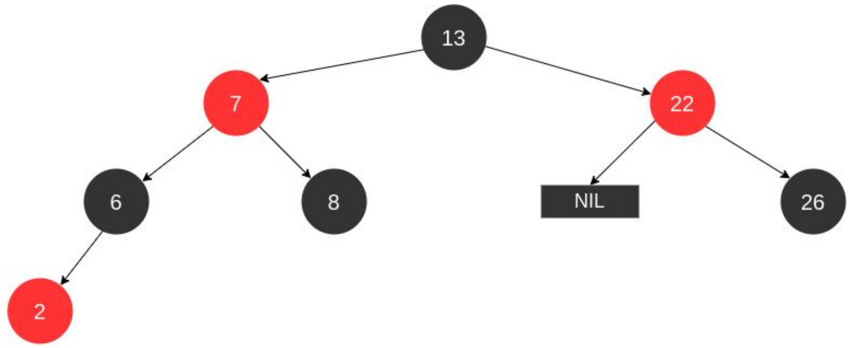
</td></tr>
<tr><td>3</td><td>
پارت ۱ (حذف نود ۳): دوتا بچه دارد، پس کیس ۳ است. محتوای ساکسسور آن یعنی ۶ در آن کپی می‌شود و سپس باید آن ۶ اضافی را پاک کنیم که کیس ۱ است و به جایش NIL می‌نشیند. حاصل:



چون نودی که پاک شد قرمز رنگ بود، در نتیجه violation ای رخ نمی‌دهد و نیاز به پارت ۲ نیست.
</td></tr>
</table>

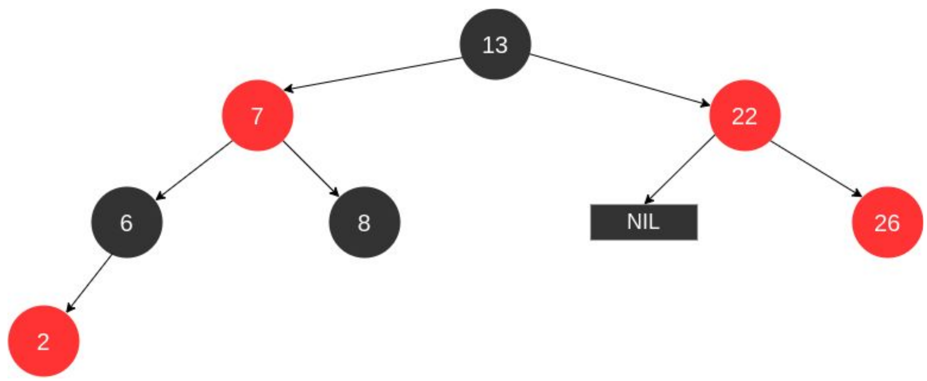
10

پارت ۱ (حذف نود ۱۰): دوتا بچه دارد، پس کیس ۳ است. محتوای ساکسسور آن یعنی ۱۳ در آن کپی می‌شود و سپس باید آن ۱۳ اضافی را پاک کنیم که کیس ۱ است و به جایش NIL می‌نشیند. حاصل:



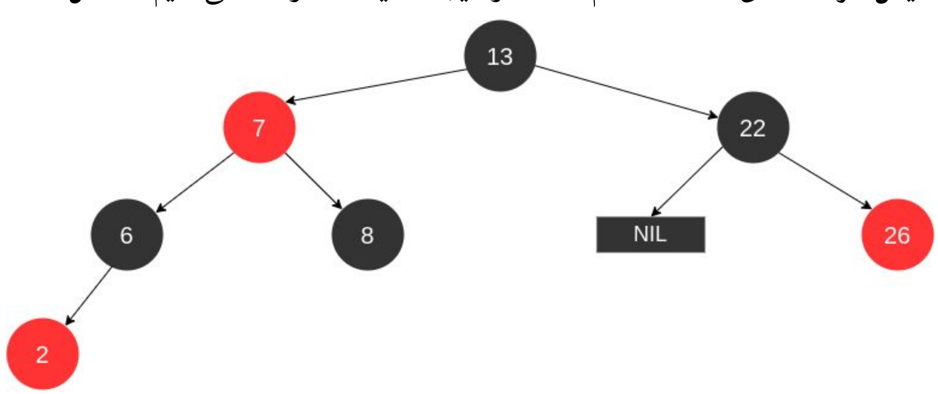
چون نود ۱۳ ای که پاک کردیم مشکی بود، پس violation رخ می‌دهد. x در اینجا NIL است و فیکس‌آپ را با آن صدا می‌زنیم.

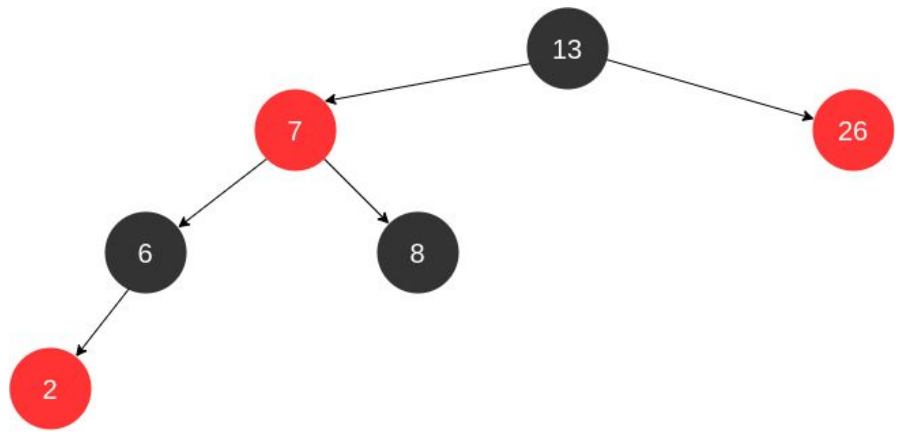
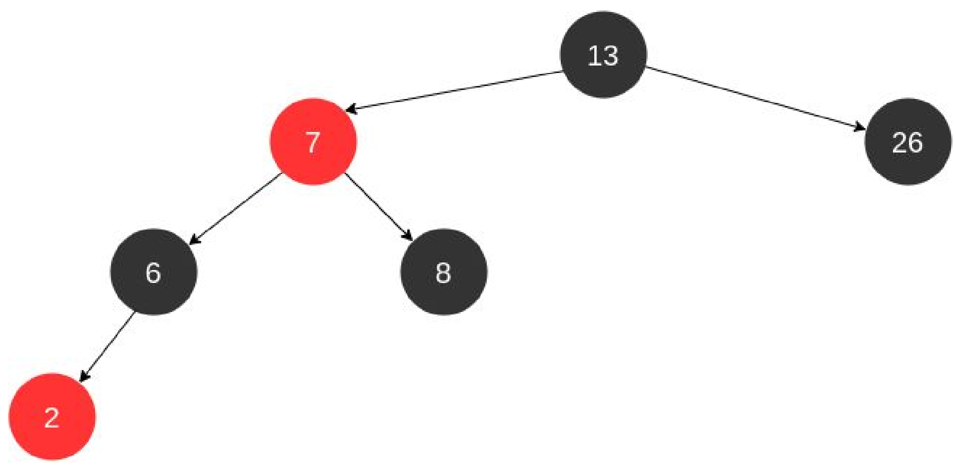
پارت ۲ (فیکس‌آپ): کیس ۲ رخ می‌دهد، زیرا sibling نود x ما (که می‌شود نود ۲۶) مشکی است و دو فرزند مشکی نیز دارد. رنگ sibling را قرمز می‌کنیم. حاصل:



حال x جدید می‌شود پدر x فعلی؛ یعنی x جدید می‌شود نود ۲۲. حال فیکس‌آپ را بر روی ۲۲ انجام می‌دهیم.

کیس صفر (نقیض شرط حلقه‌ی while) حاکم است، در نتیجه کافیست x را مشکی کنیم. حاصل:



22	پارت ۱ (حذف نود ۲۲): فقط فرزند راست دارد در نتیجه کیس ۱ است. ۲۲ پاک شده و به جایش ۲۶ می‌نشیند. حاصل:  چون نودی که پاک شد مشکی بود، پس violation داریم. x، نود ۲۶ است و فیکس‌آپ را با آن صدا می‌زنیم. پارت ۲ (فیکس‌آپ): کیس صفر (نقیض شرط حلقه‌ی while) رخ می‌دهد چون که ۲۶ قرمز است. کافیست رنگش را مشکی کنیم. حاصل: 

۹- ثابت کنید ارتفاع هر درخت red/black با n عنصر از $2 \times log_2(n+1)$ کمتر است. توضیح دهید چرا این ویژگی دلیل برتری درخت red/black بر درخت باینری است.

این اثبات با تعریف bh_x یا همان ارتفاع مشکی یا black height x شروع می‌کنیم که تعداد گره‌های مشکی از گره‌ی x تا ریشه را نشان میدهد. سپس به ترتیب عبارت های زیر را اثبات می‌کنیم:
۱- زیر درخت x را bh_x حداقل $2^{bhx}-1$ راس داخلی دارد:
اثبات:
از استقرا کمک میگیریم
حالت پایه را درختی را ارتفاع صفر در نظر می‌گیریم و $bh_x=0$ در نتیجه:
$2^0-1=0$ راس داخلی در زیر درختش وجود دارد پس حالت پایه برقرار است

حال یک گره با ارتفاع مثبت x با دو فرزند را درنظر بگیرید
اگر فرض کنیم شرط استقرا برای هریک از فرزندان برقرار باشد (بسته به مشکی یا قرمز بودن گره ها) ارتفاع مشکی هر بچه باید bh_x یا bh_x-1 باشد پس باید حداقل زیر درخت هر فرزند $2^{bhx-1}-1$ و یا $2^{bhx}-1$ گره داشته باشد و برای اثبات حکم باید ثابت کنیم فرض برای خود گره‌ی پدر نیز برقرار است یعنی گره‌ی پدر حداقل $2^{bhx}-1$ گره در زیر درختش داشته باشد که با جمع تعداد گره های زیر درخت فرزندان این حکم ثابت میشود:

$$(2^{bhx-1}-1) + (2^{bhx-1}-1) + 1 = 2^{bhx} - 1$$