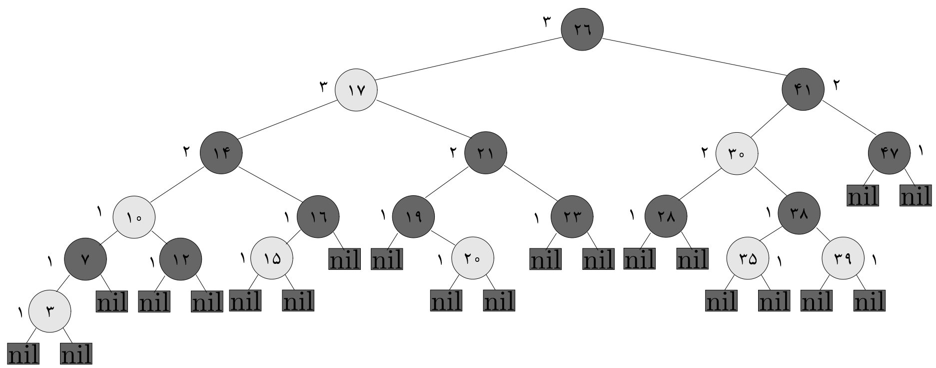
درخت قرمز- سیاه

هدف:درخت دودویی جست‌وجو که

- **ارتفاع آن $O(\lg n)$ باشد**
- **حذف و درج با $O(\lg n)$ انجام شود**
- **بسیاری از حذف و درج‌ها «عادی» باشند**

داده‌ساختارها و مبانی الگوریتم‌ها (۲۵۴—۴۰)



مثال درخت قرمز- سیاه

داده‌ساختارها و مبانی الگوریتم‌ها (۲۵۴—۴۰)

تعریف درخت قرمز- سیاه

- **درخت جست‌وحو است.**
- **هر گره یا قرمز است و یا سیاه.**
- **هر برگ nil سیاه است.**
- **دو فرزند یک گره قرمز، سیاه هستند (پدر یک گره قرمز نمی‌تواند قرمز باشد).**
- **هر مسیر ساده از یک گره به اولاد برگ شامل تعداد یکسانی گره سیاه می‌باشد.**
- **ریشه درخت سیاه است (این شرط، از شروط اساسی نیست.)**

داده‌ساختارها و مبانی الگوریتم‌ها (۲۵۴—۴۰)

تعریف‌ها و قضیه‌های اولیه

- **«سیاه‌ارتفاع»** (black-height) **گره‌ی** x ($bh(x)$):

تعداد گره‌های سیاه از x تا یک برگ فرزند (شامل برگ‌های nil است ولی خود x ر
نمی‌شماریم)

ارتفاع درخت قرمز- سیاه با n عنصر

قضیه. حداکثر ارتفاع یک درخت قرمز- سیاه که دارای n گره داخلی می‌باشد، برابر $۲\lg(n+۱)$ است.

برای اثبات نشان می‌دهیم که

لم. در درخت قرمز- سیاه، هر زیردرخت به ریشه‌ی دل‌خواه x حداقل دارای $۲^{\mathrm{bh}(x)} - ۱$ گره داخلی است.

ارتفاع درخت قرمز- سیاه با n عنصر

قضیه. حداکثر ارتفاع یک درخت قرمز- سیاه که دارای n گره داخلی می‌باشد، براب
$۲\lg(n+۱)$ است.

برای اثبات نشان می‌دهیم که

لم. در درخت قرمز- سیاه، هر زیردرخت به ریشه‌ی دل‌خواه x حداقل دارای $\mathrm{bh}(x) - ۱$
گره داخلی است.

چون

در درخت به ارتفاع h حداقل نیمی از گره‌ها (بدون درنظر گرفتن ریشه) بر روی هر مسیر ساده از ریشه به برگ، سیاه هستند.

$\Longleftarrow$ سیاه- ارتفاع ریشه‌ی درخت حداقل $\frac{h}{۲}$ خواهد بود. بنابراین:

$$n \ge ۲^{\frac{h}{۲}} - ۱ \Longrightarrow ۲^{\frac{h}{۲}} \le n + ۱ \Longrightarrow h \le ۲\lg(n+۱)$$

اثبات لم

لم. در درخت قرمز- سیاه، هر زیردرخت به ریشه‌ی دل‌خواه x حداقل دارای $\mathrm{bh}(x) - ۱$
گره داخلی است.

اثبات با استقراء بر روی سیاه–ارتفاع x

پایه‌ی استقراء : $bh(x) = ۰ \Longleftarrow x$ برگ nil است $\Longleftarrow ۲^۰ - ۱ = ۰$ گره‌ی داخلی

گام استقراء:

$bh(x) > ۰ \Longleftarrow x$ **گره داخلی است $\Longleftarrow$ دو فرزند دارد**

◁ x **قرمز است $\Longleftarrow$ سیاه- ارتفاعِ فرزندها برابر** $bh(x) - ۱$

◁ x **سیاه است $\Longleftarrow$ سیاه- ارتفاعِ فرزندها برابر** $bh(x) - ۱$ **یا** $bh(x)$

طبق فرض استقرا، زیردرخت به ریشه‌ی هرکدام از فرزندان x حداقل $۲^{\mathrm{bh}(x)-۱} - ۱$ گره داخلی دارد.

$\Longleftarrow$ **زیردرخت به ریشه‌ی x حداقل**

$$۲^{\mathrm{bh}(x)-۱} - ۱ + ۲^{\mathrm{bh}(x)-۱} - ۱ + ۱ = ۲^{\mathrm{bh}(x)} - ۱$$

گره داخلی خواهد داشت.

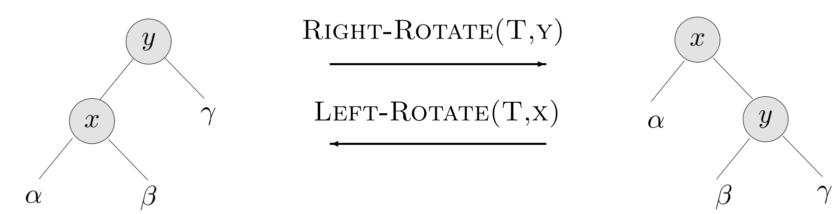
اعمال

عموی گره‌ی x

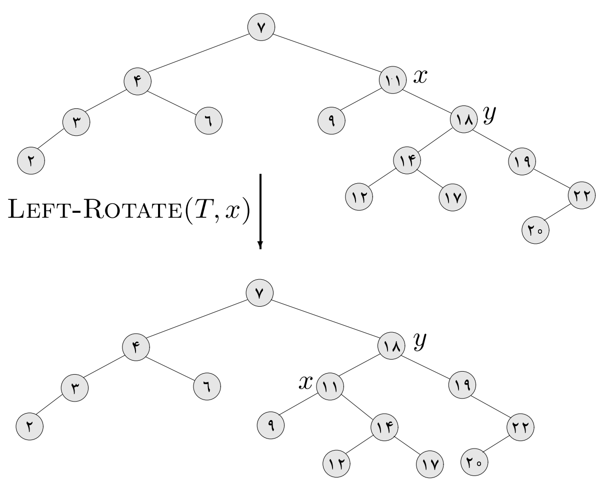
```
UNCLE(x)
if  parent[x] = right[parent[parent[x]]]
   then return left[parent[parent[x]]]
   else   return right[parent[parent[x]]]
```


دوران (Rotation)

برای بازیابی خواص درخت قرمز- سیاه بعد از عملیات درج و حذف



دوران راستگرد و چپگرد $(\alpha < x < \beta < y < \gamma)$



LEFT-ROTATE(T, x)

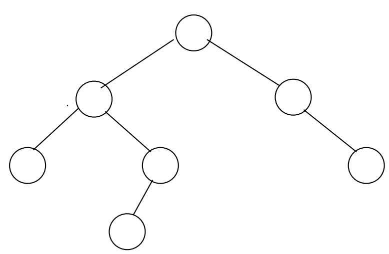
$y \leftarrow right[x]$
$right[x] \leftarrow left[y]$
if $left[y] \neq$ **null**
   then $p[left[y]] \leftarrow x$
$p[y] \leftarrow p[x]$
if $p[x] =$ **null**
   then $Root[T] \leftarrow y$
   else **if** $x = left[p[x]]$
            then $left[p[x]] \leftarrow y$
            else $right[p[x]] \leftarrow y$
$left[y] \leftarrow x$
$p[x] \leftarrow y$

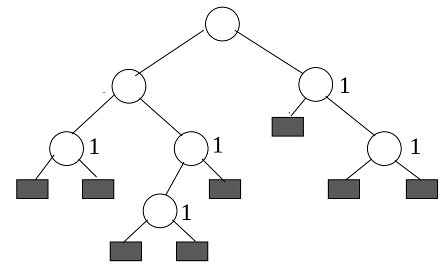
همین‌جا حل کنید!

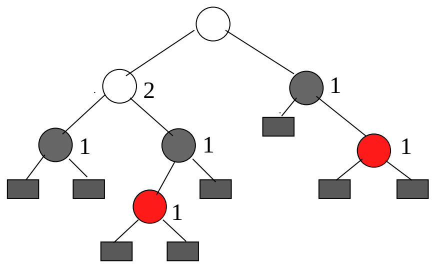
۱) آیا یک درخت قرمزسیاه با ۱۲۸ گره باید حداقل یک گره‌ی قرمز داشته باشد؟

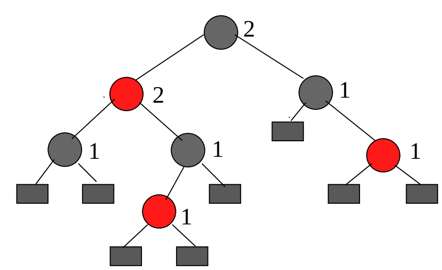
بله: چون $۲^{h+1} - ۱ = ۱۲۷$ می‌تواند تمام سیاه باشد، ولی با ۱۲۸ گره نمی‌شود.

۲) برای درخت زیر برچسب‌های «قرمز» و «سیاه» قرار دهید تا درخت قرمز–سیاه شود.

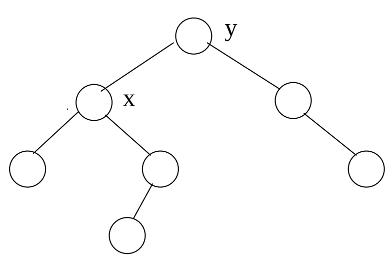




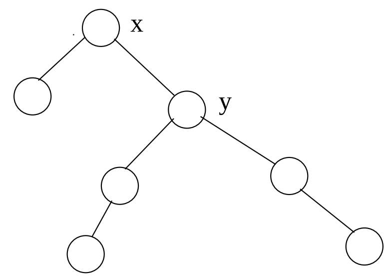


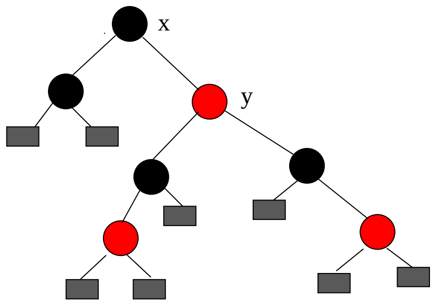


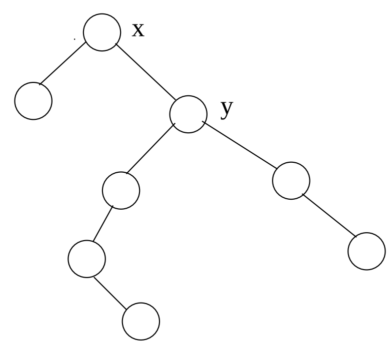
۳) با یک دوران درخت را طوری تغییر دهید تا فرزند چپ ریشه، ریشه‌ی جدید درخت شود. درخت حاصل با برچسب‌هایش را رسم کنید. آیا می‌توان با رنگ‌آمیزی درخت را قرمز-سیاه کرد؟ جواب خود را توجیه کنید.



RIGHT-ROTATE(T, y)







۴) درخت دودویی جست‌وجوی زیر داده شده است:

```
   1
  / \
 a   2
    / \
   b   3
      / \
     c   4
        / \
       d   5
          / \
         e   f
```

با چه دوران‌هایی ارتفاع درخت را از ۵ به ۳ تقلیل می‌یابد؟

```
Left-Rotate(T,3)
    1                    1
   / \                  / \
  a   2                a   2
     / \                  / \
    b   3                b   4
       / \                 /   \
      c   4               3     5
         / \             / \   / \
        d   5           c   d e   f
           / \
          e   f
```



```
Left-Rotate(T,1)
      1                    2
     / \                 /   \
    a   2               1     4
       / \             / \   /  \
      b   4           a   b 3     5
         / \             / \     / \
        3   5           c   d   e   f
       / \ / \
      c  d e  f
```


درج x در درخت قرمز–سیاه

۱) درج در درخت دودوئی جستجو

۲) x را قرمز می‌کنیم. سیاه- ارتفاع‌ها برای کلیه گره‌ها ثابت می‌ماند.

۳) پدر x سیاه است $\Longleftarrow$ پایان

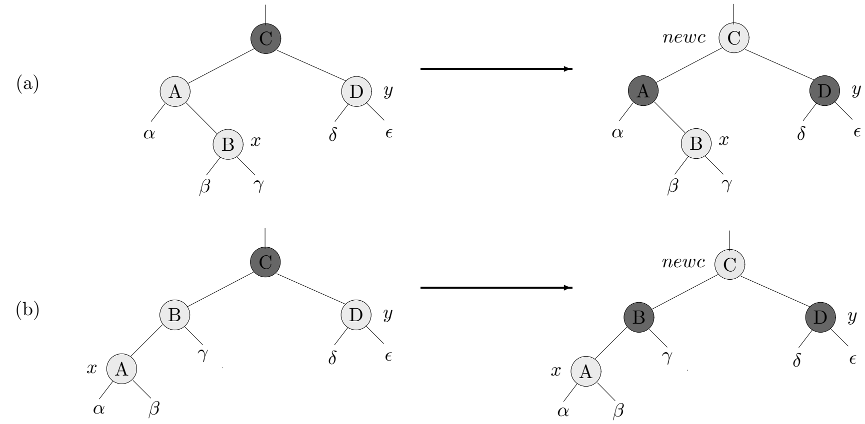
۴) پدر x قرمز است $\Longleftarrow$ به سراغ عموی x بنام y می‌رویم.

سه حالت:

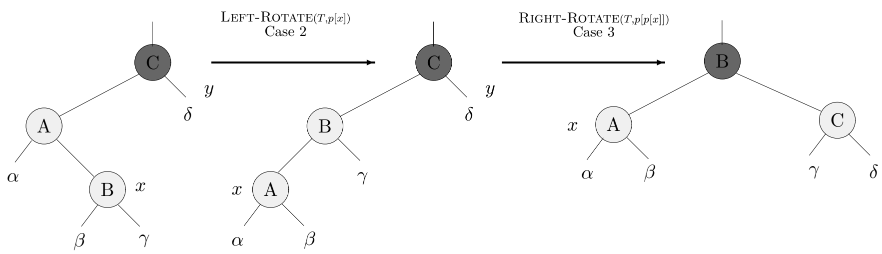
(a) y قرمز است.

(b) y سیاه است و x فرزند راست پدرش است.

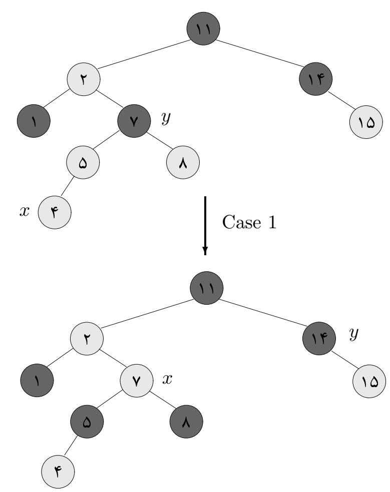
(c) y سیاه است و x فرزند چپ پدرش است.

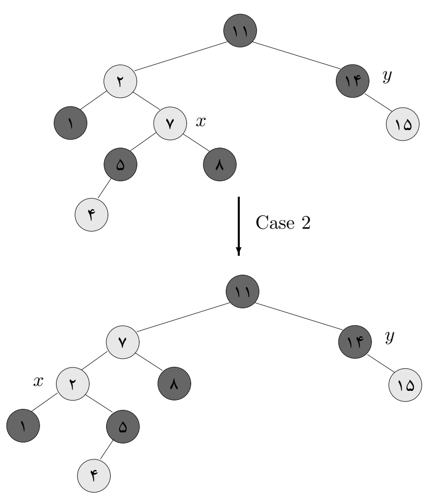


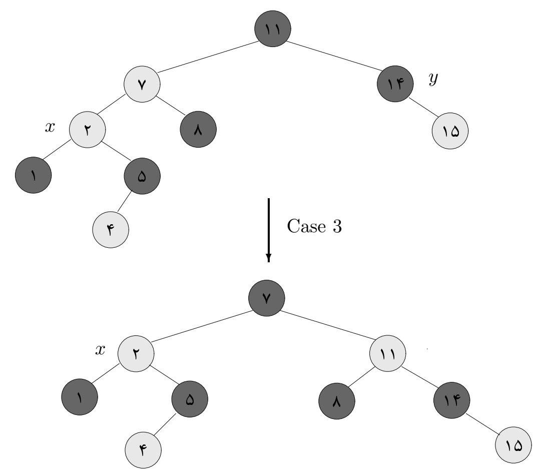
حالت اول برای RB-Insert.



حالت‌های دوم و سوم برای RB-Insert.








```
RB-Insert(T, x)
  Tree-Insert(T, x)
  color[x] ← Red
  while x ≠ root[T]  and color[p[x]] = Red
        do if p[x] = left[p[p[x]]]
              then y ← right[p[p[x]]]
                   if color[y] = Red
                      then color[p[x]] ← Black          .... case 1
                           color[y] ← Black             .... case 1
                           color[p[p[x]]] ← Red         .... case 1
                           x ← p[p[x]]                  .... case 1
                      else  if x = right[p[x]]
                              then x ← p[x]             .... case 2
                                   Left-Rotate(T, x)    .... case 2
                           color[p[x]] ← Black          .... case 3
                           color[p[p[x]]] ← Red         .... case 3
                           Right-rotate(T, p[p[x]])     .... case 3
              else  (same as then clause
                            with "right" and "left" exchanged)
  color[root[T]] ← Black
```

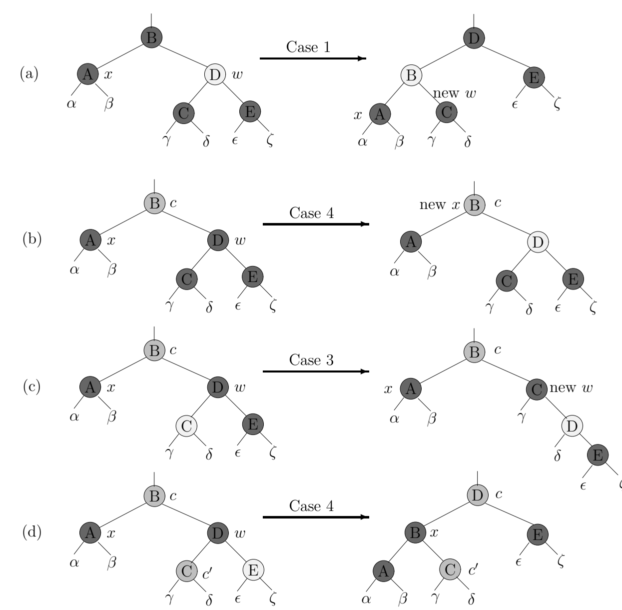

تمرین!

۱) درخت حاصل از درج عنصری با کلید ۳۶ در مثال ابتدای درس را رسم کنید.

۲) درخت قرمز-سیاه حاصل از درج به‌ترتیب ۴۱، ۳۸، ۳۱، ۱۲، ۱۹ و ۸ را در یک درخت تهی را رسم کنید.

حذف

```
RB-Delete(T, z)
if left[z] = nil[T] or right[z] = nil[T]
   then y ← z
   else y ← Tree-Successor(z)
if left[y] <> nil[T]
   then x ← left[y]
   else x ← right[y]
p[x] ← p[y]
if p[y] = nil[T]
   then root[T] ← x
   else if y = left[p[y]]
           then left[p[y]] ← x
           else right[p[y]] ← x
if y ≠ z
   then key[z] ← key[y]
        if y has other fields, copy them too
if color[y] = Black
   then RB-Delete-Fixup(T, x)
return y
```

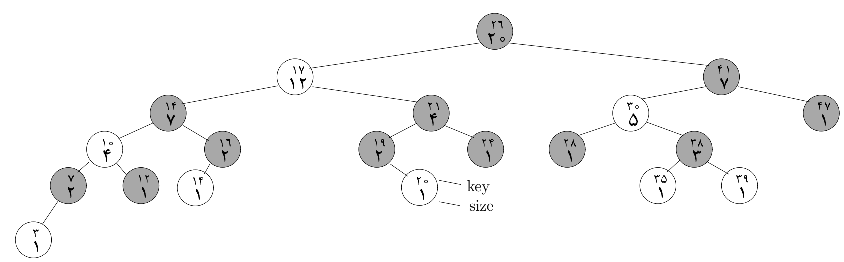

```
RB-Delete-Fixup(T, x)
while x ≠ root[T] and color[x] = Black
    do if x = left[p[x]]
          then w ← right[p[x]]
               if color[w] = Red
                  then color[w] ← Black              .... Case 1
                       color[p[x]] ← Red              .... Case 1
                       Left-Rotate (T, p[x])          .... Case 1
                       w ← right[p[x]]                .... Case 1
               if color[left[w]] = Black and color[right[w]] = Black

                  then color[w] ← Red                 .... Case 2
                       x ← p[x]                       .... Case 2
                  else if color[right[w]] = Black
                          then color[left[w]] ← Black .... Case 3
                               color[w] ← Red         .... Case 3
                               Right-Rotate(T, w)     .... Case 3
                               w ← right[p[x]]        .... Case 3
                               color[w] ← color[p[x]] .... Case 4
                               color[p[x]] ← Black    .... Case 4
```

```
        color[right[w]] ← Black        .... Case 4
        LEFT-ROTATE (T, p[x])          .... Case 4
        x ← root[T]                    .... Case 4
    else (same as then with right and left exchanges)
color[x] ← Black
```


گسترش‌های درخت قرمز–سیاه

- **درخت مرتبه‌ی آماری: اعمال درج، حذف و مرتبه‌ی آماری با** $O(\lg n)$
- **درخت بازه: درج و حذف بازه و پیداکردن هم‌پوشانی بازه‌ها در** $O(\lg n)$

درخت مرتبه‌ی آماری



تغییرات در درخت قرمز–سیاه

مولفه‌ی $size[x]$

$$size[x] = size[left[x]] + size[right[x]] + ۱$$

مقدار این مؤلفه باید در حذف و درج عناصر روزآمد شود

پیداکردن عنصر با مرتبه‌ی داده‌شده

```
OS-Select(x, i)
  r ← size[left[x]] + 1
  if i = r
    then return x
    else if i < r
           then return OS-Select(left[x], i)
           else OS-Select(right[x], i − r)
```

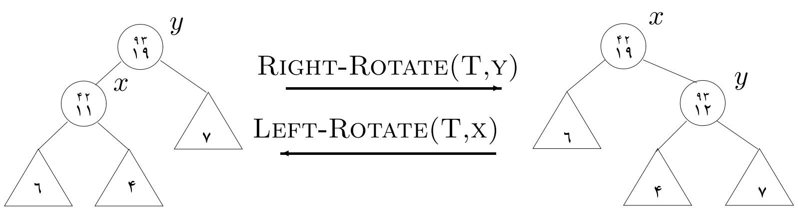

پیداکردن مرتبه‌ی یک عنصر

```
OS-Rank(T, x)
  r ← size[left[x]] + 1
  y ← x
  while y ≠ root[T]
        do if y = right[p[y]]
              then r ← r + size[left[p[y]]] + 1
           y ← p[y]
  return r
```


نگه‌داشت اندازه‌ها در درج و حذف

- در درج عادی در مسیر پیدا کردن محل درج به اندازه‌ی همه‌ی عناصر یک واحد اضافه می‌کنیم.
- در حذف از اندازه‌های عناصر این مسیر یک واحد کم می‌کنیم.
- باید تضمین کنیم که عمل دوران اندازه‌ها را به‌درستی تغییر دهد.

محاسبه‌ی اندازه‌های زیردرخت‌ها در عمل دوران



)

```
  size[y] ← size[x]
  size[x] ← size[left[x]] + size[right[x]]
```

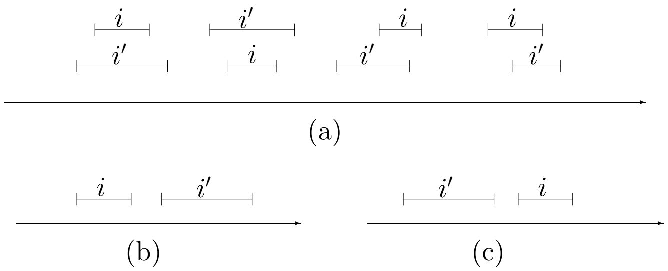
گسترش بعد: درخت بازه

بازه چیست؟

- **یک بازه‌ی بسته یک زوج مرتب از اعداد حقیقی t_1 و t_2 است که $t_1 \le t_2$ و با $[t_1, t_2]$ نمایش داده می‌شود.**
- **یک بازه‌ی $[t_1, t_2]$ شامل همه‌ی اعداد حقیقی $\{t \in R : t_1 \le t \le t_2\}$ است.**
- **یک بازه‌ی «باز» و یا «نیمه‌باز» به‌ترتیب شامل هر دو یا یکی از نقطه‌های انتهایی نیست.**

هدف داده‌ساختاری است که عناصر آن بازه باشند و بتوان اعمال درج، حذف و پیداکردن بازه‌ی هم‌پوشان یک بازه‌ی ورودی

حالت‌های هم‌پوشانی بازه‌ها



(a) چهار حالت برای هم‌پوشانی بازه‌های i و i'. (b) و (c) حالت‌های ناهم‌پوشان.

حالت‌های دو بازه نسبت به‌هم

- **برای یک بازه‌ی $i = [t_1, t_2]$، $low[i] = t_1$ را نقطه‌ی ابتدایی و $high[i] = t_2$ را نقطه‌ی انتهایی بازه می‌نامیم.**
- **دو بازه‌ی i و i' نسبت به‌هم سه حالت مختلف زیر را دارند:**

  ۱) i و i' هم‌پوشانی دارند

  ۲) $high[i] < low[i']$

  ۳) $high[i'] < low[i]$

داده‌ساختار درخت بازه

- **T : درخت بازه**
- **x : یک عنصر از آن است که $int[x]$ بازه‌ی موجود در x است.**
- **i : یک بازه‌ی ورودی**

اعمال بر روی درخت بازه

$\text{Interval-Insert}(T, x)$

درج بازه‌ی x در درخت بازه‌ی T

$\text{Interval-Delete}(T, x)$

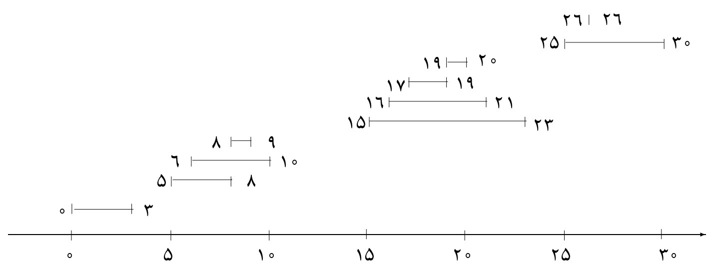
حذف بازه‌ی x از درخت بازه‌ی T

$\text{Interval-Search}(T, i)$

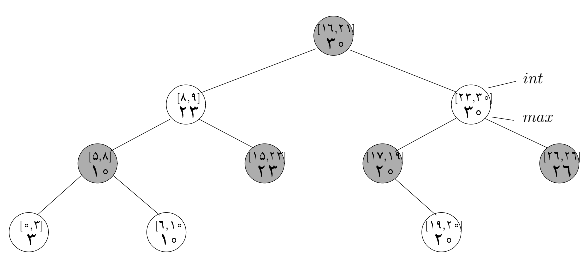
اشاره‌گر به یک عنصر x از T را باز می‌گرداند که بازه‌ی آن، $int[x]$، با بازه‌ی i هم‌پوشانی دارد. اگر این‌چنین عنصری در درخت نباشد، **null** باز می‌گرداند.

پیاده‌سازی درخت بازه

- درخت قرمز–سیاه که عناصر آن بازه هستند.
- کلید یک بازه‌ی x نقطه‌ی شروع $int[x]$ (و یا $low[int[x]]$) است.
- هر گره x در درخت مولفه‌ی $max[x]$ را دارد که نشان‌دهنده‌ی ماگزیمم حد بالای کلیه‌ی بازه‌هایی است که در زیر درخت به ریشه‌ی x قرار دارند.



(a)



(b)

درج و حذف بازه‌ها

- اعمال درج و حذف مانند درخت قرمز–سیاه انجام می‌شود.
- مانند درخت مرتبه‌ی آماری مؤلفه‌ی max عناصر را می‌توان محاسبه و ثبت کرد.
- عمل دوران هم طوری اصلاح می‌شود که این مؤلفه به‌درستی محاسبه کرد.

پیدا کردن بازه‌ی هم‌پوشان

```
NTERVAL-SEARCH(T, i)
1  x ← root[T]
2  while x ≠ null  and i does not overlap int[x]
3        do if left[x] ≠ null  and max[left[x]] ≥ low[i]
4              then x ← left[x]
5              else  x ← right[x]
6  return x
```


چرا درست است؟

قضیه. در هر تکرار حلقه‌ی while در الگوریتم INTERVAL-SEARCH(T, i)،

۱) اگر سطر ۴ اجرا شود و جست‌وجو به فرزند چپ x برود، یا زیر درخت چپ x شامل بازه‌ای است که با i هم‌پوشان است یا هیچ بازه‌ای در زیر درخت راست x وجود ندارد که با i هم‌پوشانی داشته باشد،

۲) اگر سطر ۵ اجرا شود و جست‌وجو به فرزند راست x برود، هیچ بازه‌ای در زیر درخت چپ x وجود ندارد که با i هم‌پوشانی داشته باشد.

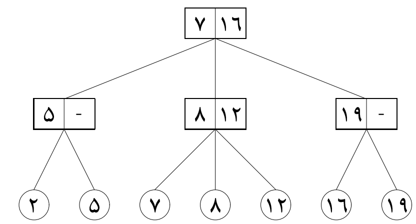
همین‌جا حل کنید!

۱) در چه صورت می‌توان یک درخت دودویی جست‌وجوی داده‌شده را به‌صورت قرمز–سیاه در آورد؟

۲) حداقل و حداکثر تعداد عناصر داخلی یک درخت «قرمز–سیاه» با سیاه‌ارتفاع h چه‌قد است؟

۳) درخت بازه‌ی دوبعدی چگونه کار می‌کند؟

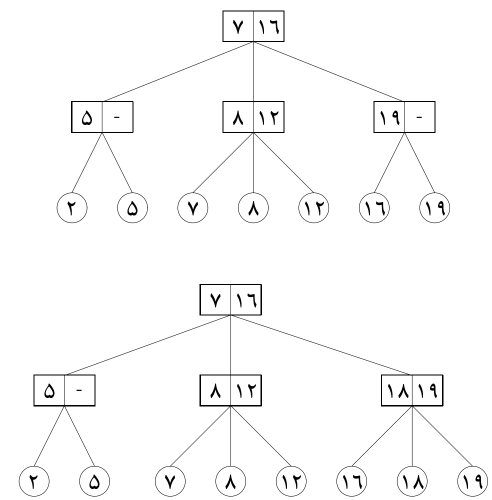
درخت ۳-۲ و بی



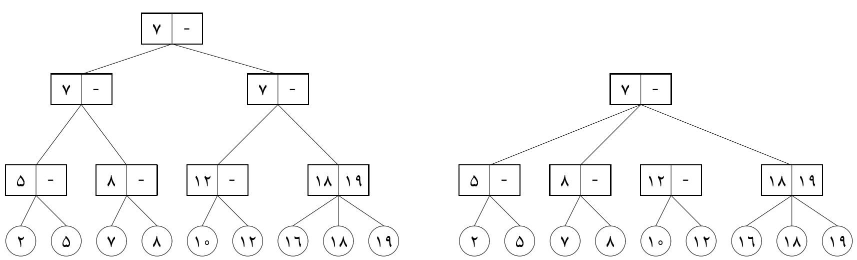
ارتفاع:

$$\lceil \log_3 n \rceil \le h \le \lceil \log_2 n \rceil$$

```
ype
  elementtype = record
     key: real;
     {other fields}
  end;
  notetypes = (leaf,interior);
  TwoThreeNode = record
    case kind: nodetype of
        leaf : (element : elementtype);
        interior : (firstchild,secondchild,thirdchild:^twotreenode;
                    lowofsecond,lowoffirst:real;)
  end;
  SET = ^TwoThreeNode;
```

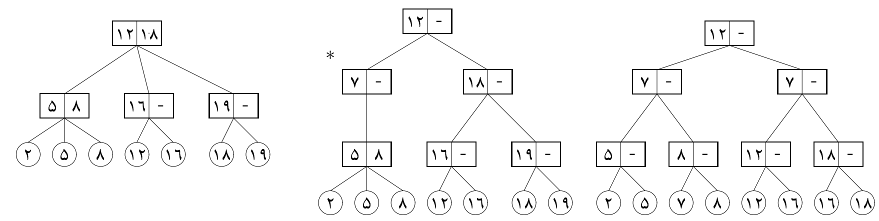



درخت حاصل از درج $x = 18$



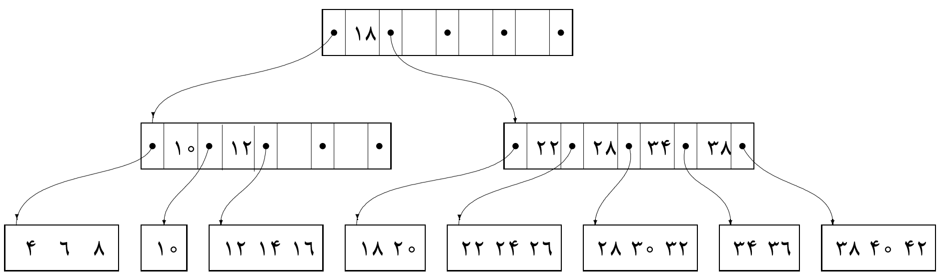
(ب) (الف)

درخت حاصل از درج $x = 10$



درخت حاصل از حذف ۱۰ و سپس ۷

درخت «بی»



یک درخت بی از مرتبه‌ی ۵ و $k = ۳$.

پیاده‌سازی مبتنی بر درخت ۳-۲

یک درخت بی از مرتبه‌ی m یک درخت جست‌وجوی m تایی با ویژگی‌ها و پارامترهای زیر است:

۱) ریشه، یا برگ است و یا حداقل دو فرزند دارد.

۲) هر گرهی بجز ریشه و برگ‌ها، حداقل $\lceil\frac{m}{۲}\rceil$ و حداکثر m فرزند دارد.

۳) یک گره‌ی داخلی دارای حداکثر $m-۱$ کلید و m اشاره‌گر (آدرس بکوک دیسک) از فرم $(p_۱, k_۲, p_۲, k_۳, p_۴, \cdots, k_m, p_m)$ است که p_i اشاره‌گر به i امین زیر درخت آن گره (به‌شرط وجود) و k_i کوچک‌ترین کلید زیردرخت i ام است. و نیز $k_۲ < k_۳ < \cdots < k_m$.

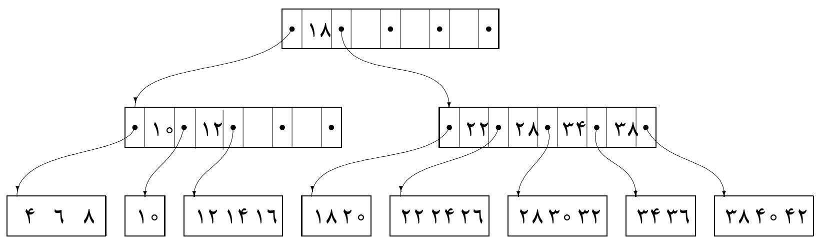
۴) برگ‌ها همه در یک سطح قرار دارند و رکوردهای ذخیره‌شده در هر برگ به‌ترتیب کلیدشان از چپ‌به‌راست در برگ‌ها قرار دارند.

۵) بسته به‌اندازه‌ی رکوردها، از یک تا حداکثر k عدد رکورد در هر برگ قرار می‌گیرد.

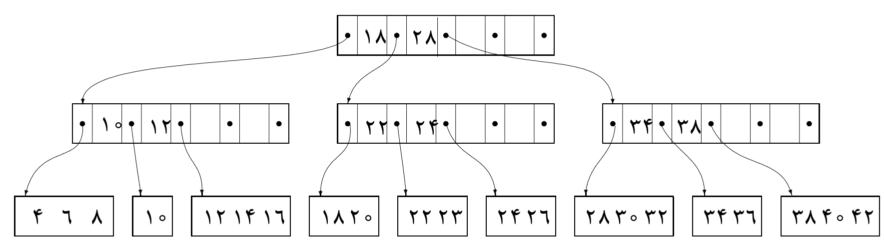
توجه: یک درخت ۳-۲ یک درخت بی از مرتبه ۳ است.

مقادیر m و k معمولاً طوری تعیین می‌شوند که هر گره‌ی داخلی و هر برگ به‌اندازه‌ی یک بلوک از دیسک باشد و بتوان با یک‌بار دست‌رسی آن را به حافظه‌ی اصلی خواند. هم‌چنین اشاره‌گرها در این درخت آدرس بلوک بر روی دیسک هستند.

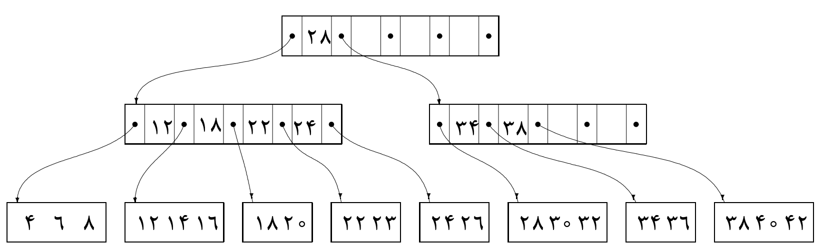
درج



پس از درج ۲۳



پس از حذف ۱۰



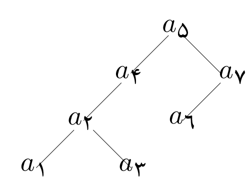
تحلیل

- درخت بی از مرتبه‌ی m شامل n عنصر
- هر برگ به‌طور متوسط b عنصر، (تخمین $b = k/۲$ برای توزیع یکنواخت درست است)، در آن‌صورت تعداد برگ‌ها برابر $\lceil n/b \rceil$
- بیش‌ترین ارتفاع درخت وقتی است که گره‌های داخلی (به‌جز ریشه) $\lceil \frac{m}{۲} \rceil$ عدد فرزند و ریشه دو فرزند داشته باشد.
- در این‌صورت، برگ‌ها حدود $۲\lceil n/b \rceil/m$ پدر و $۴\lceil n/b \rceil/m^۲$ جد و همین‌طور تا بالاتر دارد.

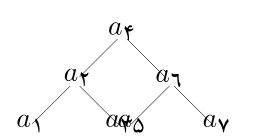
- اگر j تعداد گره‌های موجود در مسیر بین ریشه و برگ‌ها باشد، داری
$۱ \ge ۲^{j-۱}\lceil n/b \rceil/m^{j-۱}$ چرا که در غیر این‌صورت تعداد فرزندان ریشه کم‌تر از
خواهد بود.
- بنابراین $\lceil n/b \rceil \ge (m/۲)^{j-۱}$ و یا $j \le ۱ + \log_{m/۲}\lceil n/b \rceil$
- مثلاً اگر $n = ۱۰^۶$، $b = ۱۰$ و $m = ۱۰۰$، در آن‌صورت $j \le ۳.۵$.
- روشن است که اعمال جست‌وجو، و حذف حداکثر برابر j بار و درج حداکثر $+ ۱$
بار دست‌رسی به دیسک لازم دارد.

درخت دودویی جست‌وجوی بهینه (Optimal BST)

درخت شامل ۷ عنصر $a_۱ < a_۲ < \cdots < a_۷$ و احتمال جست‌وجو برای این عناصر به‌ترتیب برابر $\frac{۲}{۲۴}$، $\frac{۱}{۲۴}$، $\frac{۳}{۲۴}$، $\frac{۴}{۲۴}$، $\frac{۵}{۲۴}$، $\frac{۳}{۲۴}$ $\frac{۶}{۲۴}$ باشد

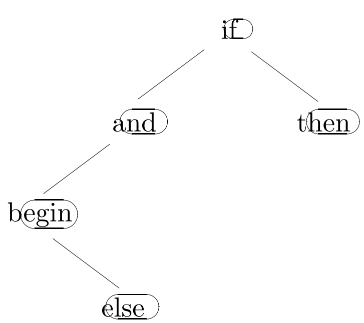


درخت نامتوازن با متوسط زمان جست‌وجوی $\frac{۳۱}{۲۴}$



درخت متوازن با متوسط زمان جست‌وجوی $\frac{۴۸}{۲۴}$

مثال‌ها: کتاب‌خانه، جدول نمادها



مسئله در حالت کلی

جست‌وجوی موفق و ناموفق (a_i و b_i): عنصر داخلی و خارجی

$$b_0 < a_1 < b_2 < \cdots < a_{i-1} < b_i < a_i < \cdots < b_{n-1} < a_n < b_n$$

لم. تعداد عناصر خارجی، برای یک درخت دودویی جستجو با n عنصر برابر $n+1$ است.

تعریف دقیق ورودی مسئله

- ◁ $a_1 < a_2 < \cdots < a_n$ **عناصر درخت**
- ◁ p_i **احتمال جست‌وجو موفق برای** a_i ($i = 1..n$)
- ◁ q_i **احتمال جست‌وجوی ناموفق برای** b_i، **اگر** $a_i < b_i < a_{i+1}$ ($i = 1..n-1$)
- ◁ q_0 **احتمال جست‌وجوی ناموفق برای** $b_0 < a_1$
- ◁ q_n **احتمال جست‌وجوی ناموفق برای** $a_n < b_n$

خروجی مسئله

با داشتن p_i ها و q_i ها یک درخت دودویی جست‌وجوی بهینه بسازید که متوسط زمان جست‌وجو (اعم ازموفق یا نا موفق) در آن کمینه شود

این مقدار

$$\sum_{i=1}^{n} p_i(1 + depth(a_i)) + \sum_{i=0}^{n} q_i(depth(b_i))$$

که $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$

تعریف زیرمسئله

- **زیر مسئله‌ی** T_{ij}
  درخت OBST **برای** $a_{i+1} < \cdots < a_j$ **با ورودی** $q_i, p_{i+1}, q_{i+1}, \ldots, p_j, q_j$
  T_{0n} **مسئله‌ی اصلی است.**
- C_{ij} : **هزینه‌ی** T_{ij}

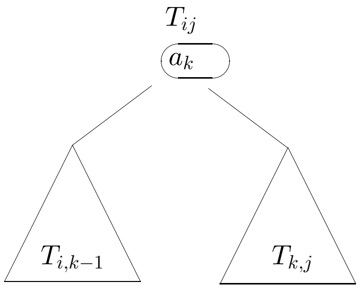
$$C_{ij} = \sum_{r=i+1}^{j} p_r[depht(a_r) + 1] + \sum_{r=0}^{j} q_r[depth(b_r)]$$

- $C_{ij}^{(k)}$: C_{ij} اگر a_k ریشه‌ی T_{ij} باشد.
- w_{ij} : وزن درخت T_{ij}، برابر مجموع احتمال‌های p_i و q_i در T_{ij}:

$$w_{ij} = q_i + \sum_{r=i+1}^{j} (p_r + q_r)$$

- r_{ij} ریشه‌ی T_{ij}

زیر درخت T_{ij} برای $a_{i+1} < \cdots < a_j$



$i < k \le j$

حل زیر مسئله‌ی T_{ij}

اگر a_k (برای $i < k \le j$) ریشه باشد، داریم:

$$\begin{aligned} C_{ij}^{(k)} &= (C_{i,k-1} + w_{i,k-1}) + (C_{kj} + w_{kj}) + p_k \\ &= C_{i,k-1} + C_{kj} + w_{ij} \end{aligned}$$

و داریم:

$$C_{ij} = \min_{i<k\le j} C_{ij}^{(k)}$$

در شروع برای T_{ii} داریم $\Longleftarrow$ $w_{ii} = q_i$ و $c_{ii} = 0$

الگوریتم

```
OBST(p_1, ..., p_n, q_0, ..., q_n)
for i ← 0 to n
    do w_ii ← q_i
       c_ii ← 0
for l ← 1 to n
    do for i ← 0 to n − l
           do j ← i + l
              w_ij ← w_{i,j−1} + p_j + q_j
              c_ij ← min_{i<k≤j} {c_{i,k−1} + c_kj + w_ij}
              r_ij ← the k for which above is mininimum
```

تحلیل

الگوریتم فوق از $\Theta(n^3)$ است.

مثال

$$n = 4$$

$$a_1 < a_2 < a_3 < a_4$$

$$p_1 = \frac{1}{4}$$
$$p_2 = \frac{1}{8}$$
$$p_3 = p_4 = \frac{1}{16}$$
$$q_0 = \frac{1}{8}$$
$$q_1 = \frac{3}{16}$$
$$q_2 = q_3 = q_4 = \frac{1}{16}$$

جدول‌ها

مراحل مختلف الگوریتم گفته‌شده در جدول زیر می‌آید:

$c_{00} = 0$ $w_{00} = 2$	$c_{01} = 9$ $w_{01} = 9$ $r_{01} = a_1$	$c_{02} = 18$ $w_{02} = 12$ $r_{02} = a_1$	$c_{03} = 25$ $w_{03} = 14$ $r_{03} = a_1$	$c_{04} = 33$ $w_{04} = 16$ $r_{04} = a_2$	۰
	$c_{11} = 0$ $w_{11} = 3$	$c_{12} = 6$ $w_{12} = 6$ $r_{12} = a_2$	$c_{13} = 11$ $w_{13} = 8$ $r_{13} = a_2$	$c_{14} = 18$ $w_{14} = 10$ $r_{14} = a_2$	۱
		$c_{22} = 0$ $w_{22} = 1$	$c_{23} = 3$ $w_{23} = 3$ $r_{23} = a_3$	$c_{24} = 8$ $w_{24} = 5$ $r_{24} = a_4$	۲
			$c_{33} = 0$ $w_{33} = 1$	$c_{34} = 3$ $w_{34} = 3$ $r_{14} = a_4$	۳
				$c_{44} = 0$ $w_{44} = 1$	۴

درخت مثال