دانشکده‌ی علوم ریاضی

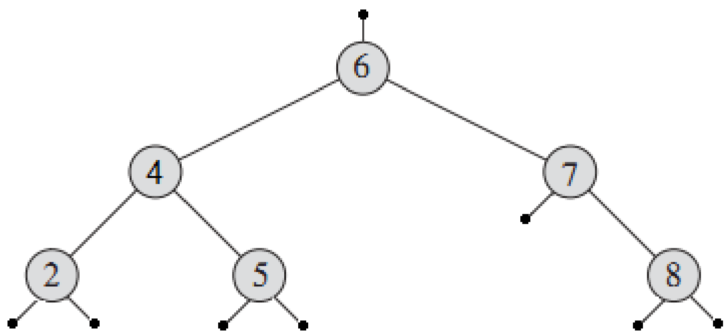
آنالیز الگوریتم‌ها | ۱۲ آذر ۹۲

جلسه‌ی ۱۹: درخت دودویی جست‌وجو

مدرّس: دکتر شهرام خزائی | نگارنده: امیر عزیزی جیرآبادی

۱ درخت جست‌وجوی دودویی چیست؟

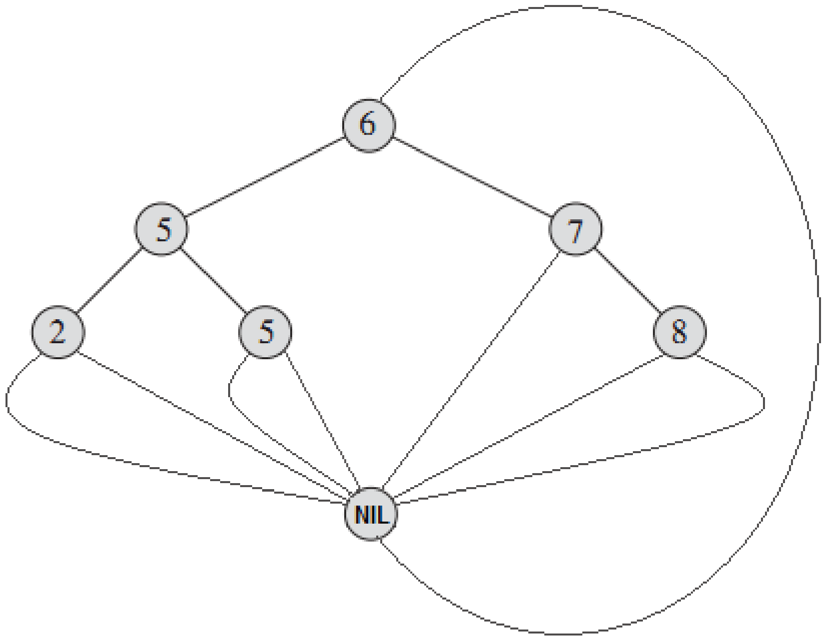
در پیاده‌سازی درخت با استفاده از اشاره‌گرها[۱] هر رأس با اشاره‌گری مانند x شناخته می‌شود که دارای چهار مؤلفه است. یک مؤلفه اشاره‌گر به نام p به پدرش، دو اشاره‌گر $left$ و $right$ به ترتیب برای اشاره به فرزندان چپ و راست می‌باشد. همچنین هر رأس دارای یک کلید به نام key نیز می‌باشد. در مواردی رأس فرزند چپ یا راست یا پدر نداشته باشد، از مقدار null استفاده خواهیم کرد. همانگونه که در شکل زیر مشاهده می‌شود، به جای پدر ریشه درخت یعنی رأس با کلید ۶ و همچنین رئوسی که فرزند چپ یا راست ندارند، مانند رأس با کلید ۸، از مقدار null استفاده شده است:



شکل ۱: استفاده از مقدار null

[۱] Pointer

همانند پیاده سازی لیست برای راحتی، از یک رأس NIL به جای مقدار null استفاده خواهیم کرد:



شکل ۲: رأس NIL

در ادامه برای سادگی رأس NIL را رسم نخواهیم کرد.

درخت جست‌وجوی دودویی[2] درختی است که ویژگی زیر را دارد:

تعریف ۱ (ویژگی درخت جست‌وجوی دودویی) *اگر x رأسی در درخت باشد، برای تمام رئوس مانند r در زیردرخت[3] راست x، رابطه $r.key > x.key$ برقرار است. همچنین برای هر رأس مانند l در زیردرخت چپ x، رابطه $l.key \leq x.key$ برقرار است.*

۲ عملیات بر روی درخت جست‌وجو

درخت‌های جست‌وجو لازم است، از عملیات‌های زیر پیشتیبانی کنند:

- **جست‌وجو**[4] کردن یک کلید در درخت
- پیدا کردن عضو **کمینه**[5] و **بیشینه**[6] درخت
- پیدا کردن **عنصر پسین**[7] و **عنصر پیشین**[8] یک عضو درخت

[2] Binary Saerch Tree(BST)
[3] Subtree
[4] Search
[5] Minimum
[6] Maximum
[7] Successor
[8] Predecessor

در ادامه رَویه[۹]ای برای هر یک از موارد فوق ارائه می‌کنیم.

۱.۲ جست‌وجو

برای پیدا کردن یک کلید کافیست رویه BST-Search را بر روی کلید مورد نظر فراخوانی کنیم:

Algorithm 1 Search an Element on a BST

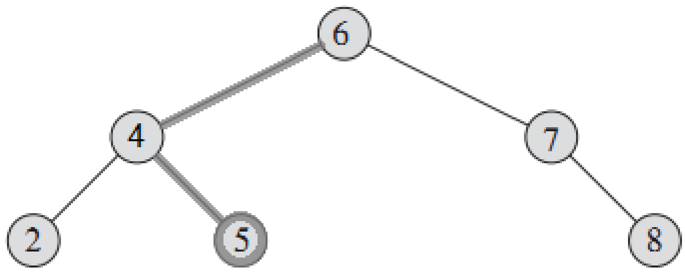
```
function BST-Search(Tree T, key k)
    return Recursive-Search(T.root, k)

function Recursive-Search(Node x, key k)
    if x = NIL or x.key = k then
        return x
    if k ≤ x.key then
        return Recursive-Search(x.left, k)
    else
        return Recursive-Search(x.right, k)
```

رویه BST-Search برای پیدا کردن کلید k در درخت T رویه Recursive-Search را برروی ریشه درخت فراخوانی کرده، و خروجی آن را به عنوان خروجی برمی‌گرداند.
الگوریتم Recursive-Search نیز بصورت بازگشتی عمل می‌کند؛ به این صورت که در هر بارفراخوانی:

- اگر رأس x برابر با NIL باشد، یعنی جست‌وجو ناموفق بوده و کلید عضو درخت نیست، در نتیجه رأس NIL را برمی‌گرداند.
- اگر کلید مورد نظر با کلید x برابر باشد، جست‌وجو موفق بوده و رأس x را بعنوان خروجی برمی‌گرداند.
- اگر k کمتر یا مساوی کلید x باشد، طبق ویژگی درخت جست‌وجو دودویی، کلید مورد نظر در صورت وجود در زیردرخت چپ رأس x قرار دارد، و رویه Recursive-Search بر روی فرزند چپ x فراخوانی خواهد شد.
- اگر k بیشتر از کلید x باشد، طبق ویژگی درخت جست‌وجو دودویی، کلید مورد نظر در صورت وجود در زیردرخت راست رأس x قرار دارد، و رویه Recursive-Search بر روی فرزند راست x فراخوانی خواهد شد.

مثال ۱ *شکل صفحه بعد نحوه اجرای الگوریتم BST-Search را برای درختی دودویی نشان می‌دهد:*



شکل ۳: جست‌وجو کلید ۵

[۹] Routine

از لحاظ زمان اجرا نیز، طبق ویژگی درخت جست‌وجو دودویی، اگر کلید مورد نظر عضو درخت باشد، یا رأسی است که روی آن قرار داریم و یا دقیقا عضو یکی از زیردرخت‌های سمت چپ و راست رأسی است که روی آن قرار داریم. بنابراین هزینه اجرای رویه فوق برابر با $O(h)$ خواهدبود، که h در آن ارتفاع درخت است.

۲.۲ عضو کمینه و بیشینه

برای پیدا کردن عضو کمینه یا بیشینه کافیست رویه‌های BST-Minimum و BST-Maximum را بر روی درخت فراخوانی کنیم:

Algorithm 2 Finding Minimum and Maximum Element of a BST

```
function BST-Minimum(Tree T)
    x ← T.root
    while x.left ≠ NIL do
        x ← x.left
    return x
```

```
function BST-Maximum(Tree T)
    x ← T.root
    while x.right ≠ NIL do
        x ← x.right
    return x
```

طبق ویژگی درخت جست‌وجو دودویی، عضو بیشینه سمت راست‌ترین فرزند ریشه خواهدبود. رویه BST-Maximum نیز ابتدا ریشه درخت را به x نسبت می‌دهد، و تا زمانی که فرزند راست x برابر با NIL نباشد، فرزند راست x را به آن نسبت می‌دهد؛ و در انتها مقدار نهایی x را بعنوان خروجی بر می‌گرداند.
از لحاظ زمان اجرا نیز همانند رویه جست‌وجو، هزینه پیدا کردن اعضای کمینه و یا بیشینه متناسب با ارتفاع درخت می‌باشد.

۳.۲ عضو پیشین و پسین

عضو پسین رأس x، رأس دارای کوچکترین کلید بزرگ‌تر از $x.key$ است. بطور مشابه پیشین عنصر x، رأس دارای بزرگترین کلید کوچک‌تر از $x.key$ می‌باشد.

Algorithm 3 Finding Successor and Predecessor of an Element on a BST

```
function BST-Successor(Node x)
    if x.right ≠ NIL then
        return BST-Minimum(x.right)
    y ← x.p
    while y ≠ NIL and x = y.right do
        x ← y
        y ← y.p
    return y
```

```
function BST-Predecessor(Node x)
    if x.left ≠ NIL then
        return BST-Maximum(x.left)
    y ← x.p
    while y ≠ NIL and x = y.left do
        x ← y
        y ← y.p
    return y
```

از آنجایی که هر دو رویه فوق مانند هم عمل می‌کنند؛ در این قسمت تنها عملکرد رویه BST-Successor را بررسی خواهیم کرد.
طبق تعریف کلید عضو پسین x از کلید x بزرگتر است، برای x ممکن است دو حالت زیر رخ دهد: :

حالت اول: اگر x فرزند راست داشته باشد، عضو پسین آن همان عضو کمینه‌ی زیردرخت سمت راستش است.

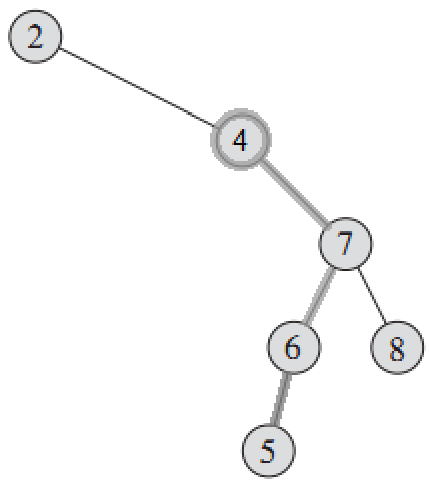
حالت دوم: اگر x فرزند راست نداشته باشد و خود فرزند چپ پدرش باشد، عضو پسین آن پدرش است.

حالت سوم: اگر x فرزند راست نداشته باشد و خود فرزند راست پدرش باشد، عضو پسین آن پدر اولین جدی از x است که فرزند چپ پدرش باشد.

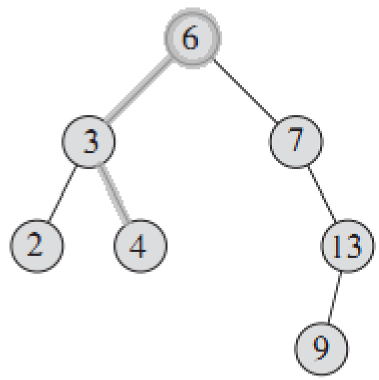
حالت چهارم: اگر x فرزند راست نداشته باشد و خود فرزند راست پدرش باشد، اما هیچ‌یک از اجداد x فرزند چپ پدرش نباشد، x عضو بیشینه درخت است و عضو پسین ندارد.

روند اجرای رویه BST-Predeccessor را می‌توان در شکل زیر مشاهده کرد:

مثال ۲ *روند اجرای رویه* BST-Successor *برای هر دو حالت فوق را می‌توان در شکل زیر مشاهده کرد:*



شکل ۴: (حالت اول) عضو پسین رأس با کلید ۴ رأس با کلید ۵ است. (حالت دوم) عضو پسین رأس با کلید ۶ رأس با کلید ۷ است.



شکل ۵: (حالت سوم) عضو پسین رأس با کلید ۴ رأس با کلید ۶ است. (حالت چهارم) رأس با کلید ۱۳ عضو پسین ندارد.

هزینه اجرای رویه‌های فوق نیز، متناسب با ارتفاع درخت می‌باشد.

۳ عملیات درج و حذف

به خاطر پیچیدگی دو رویه **حذف کردن**[۱۰] و **درج کردن**[۱۱] نسبت به علملیات‌های بخش قبلی، این دو عملیات را در بخشی جداگانه آورده‌ایم.

۱.۳ درج

برای درج کردن رأس z در درخت T رویه BST-Search را فرا خوانی می‌کنیم. فرض می‌کنیم فرزندان چپ و راست z برابر با NIL هستند.

Algorithm 4 Insertion of an Element to a BST and Non-Recursive Search for an Element in a BST

```
function BST-Insert(Tree T, Node z)
    [assumes z.right and z.left is NIL]
    y ← NIL
    x ← T.root
    while x ≠ NIL do
        y ← x
        if z.key ≤ x.key then
            x ← x.left
        else
            x ← x.right
    z.p ← y
    if y = NIL then
        // tree T was empty
        T.root ← z
    else
        if z.key ≤ y.key then
            y.left ← z
        else
            y.right ← z
```

```
function Search-BST(Tree T, key k)
    x ← T.root
    while x ≠ NIL and x.key ≠ k do
        if k < x.key then
            x ← x.left
        else
            x ← x.right
    return x
```

برای درج کردن رأس z در درخت T مانند رویه BST-Search با شروع از ریشه T، درخت را پیمایش می‌کنیم تا محل مناسب z و پدر آن یعنی y را بیابیم. برای این منظور، تغییرات لازم را در نسخه غیر بازگشتی الگوریتم جست‌وجو وارد می‌کنیم. پس از حلقه **while** پدر z را برابر با y قرار می‌دهیم. برای y ممکن است، دو حالت زیر رخ دهد:

- **حالت اول:** اگر y برابر با NIL باشد، یعنی درخت هیچ رأسی ندارد. پس z ریشه درخت خواهد بود.
- **حالت دوم:** در غیر این صورت، با حفظ ویژگی درخت جست‌وجوی دودویی، z را به عنوان فرزند y قرار می‌دهیم.

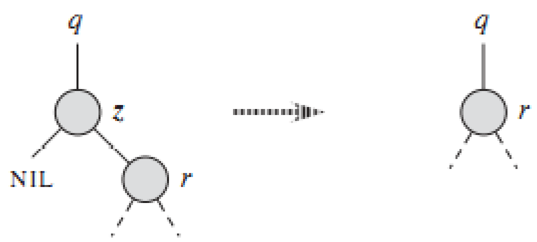
از لحاظ زمان اجرا نیز مانند رویه BST-Search هزینه اجرا متناسب با ارتفاع درخت خواهد بود.

[۱۰] deletion
[۱۱] insertion

۲.۳ حذف

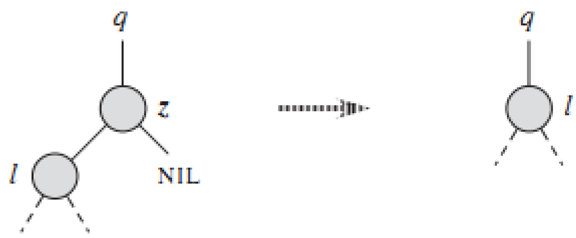
به هنگام حذف کردن رأس z از درخت T ممکن است یکی از حالات زیر رخ دهد:

- **حالت اول:** z فرزند چپ نداشه باشد؛ که در این حالت فرزند راست z را (حتی اگر NIL باشد) به جای آن قرار می‌دهیم. این کار با استفاده از رویه Transplant که بعداً معرفی خواهد شد، در زمان ثابت انجام می‌پذیرد. این رویه یک زیردرخت را به‌جای زیردرخت دیگری از درخت می‌نشاند.



شکل ۶: حالت اول ـ رأس فرزند چپ ندارد
Transplant$(T, z, z.right)$

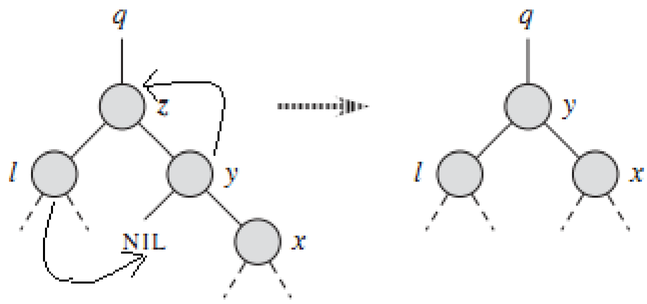
- **حالت دوم:** z فرزند راست نداشه باشد؛ که در این حالت فرزند چپ z را (حتی اگر NIL باشد) به جای آن قرار می‌دهیم:



شکل ۷: حالت دوم ـ رأس فرزند راست ندارد
Transplant$(T, z, z.left)$

- **حالت سوم:** اگر z هر دو فرزند را داشته باشد، برای عضو پسین z، یعنی y، ممکن است دو حالت رخ دهد؛ که در هر دو حالت می‌خواهیم y را جایگزین z کنیم. دقت کنید که فرزند چپ y همواره NIL است:

الف) y فرزند چپ z باشد؛ در این حالت y را به جای z قرار می‌دهیم. همچنین فرزند چپ z را به جای فرزند چپ y، قرار می‌دهیم:



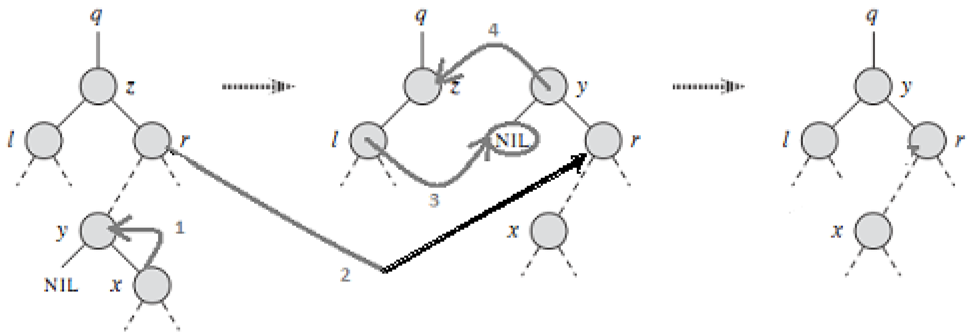
شکل ۸: حالت سوم، الف ـ حالتی که عضو پسین فرزند خود عنصر می‌باشد

$y.left \leftarrow z.left$
$y.left.p \leftarrow y$
$\textsc{Transplant}(T, z, y)$

ب) y فرزند چپ z نباشد؛ در این حالت کارهای زیر را باید انجام دهیم:

۱ قرار دادن فرزند راست y به جای خودِ y،
۲ قرار دادن فرزند راست z به جای فرزند راست y،
۳ قرار دادن فرزند چپ z به جای فرزند چپ y،
۴ قرار دادن y به جای z

در تصویر زیر چهار عمل فوق را مشاهده می‌کنید:



شکل ۹: حالت سوم، ب ـ حالتی که عضو پسین فرزند خود عنصر نمی‌باشد

$\textsc{Transplant}(T, y, y.right)$
$y.right \leftarrow z.right$
$y.right.p \leftarrow y$
$y.left \leftarrow z.left$
$y.left.p \leftarrow y$
$\textsc{Transplant}(T, z, y)$

رویه زیر هر سه حالت فوق را پوشش می‌دهد:

Algorithm 5 Deletion of an Element on a BST

```
function BST-Delete(Tree T, Node z)
    if z.left = NIL then
        Transplant(T, z, z.right)
    else
        if z.right = NIL then
            Transplant(T, z, z.left)
        else
            y ← BST-Minimum(z.right)
            if y.p ≠ z then
                Transplant(T, y, y.right)
                y.right ← z.right
                y.right.p ← y
            y.left ← z.left
            y.left.p ← y
            Transplant(T, z, y)

function Transplant(Tree T, Node u, Node v)
    if u.p = NIL then
        T.root ← v
    else
        if u = u.p.left then
            u.p.left ← v
        else
            u.p.right ← v
    if v ≠ NIL then
        v.p ← u.p
```

رویه BST-Delete رأس z را از درخت T حذف می‌کند. به اینصورت که سطور اول و دوم مربوط به حالت اول، سطور چهارم و پنجم مربوط به حالت دوم، سطور هفتم تا چهاردهم نیز مربوط به حالت سوم می‌باشد. در سطر هفتم با توجه به اینکه z هر دو فرزند خود را داراست، به‌جای پیدا کردن عضو پسین با فراخوانی رویه BST-Successor، رویه ساده‌تر BST-Minimum فراخوانی می‌شود. با توجه به رویه
عضو پسین z را می‌یابد.

رویه Transplant نیز در درخت T زیر درخت با ریشه‌ی v را با جایگزین زیردرخت با ریشه‌ی u می‌کند، که روند اجرای آن مانند شکل ۵ می‌باشد.

از نظر زمان اجرا نیز همان گونه که مشاهده می‌شود، بجز سطر پنجم، یعنی فراخوانی BST-Minimum، بقیه سطور هزینه اجرا ثابتی دارند؛ بنابراین هزینه اجرای الگوریتم همانند یافتن عضو کمینه درخت، متناسب با ارتفاع درخت خواهد بود.