بسمه تعالی

جزوه درس ساختمان داده

درخت جست‌وجوی دودویی

استاد: دکتر فیلی

علی شیراوند

1- مفهوم دیکشنری

در علوم کامپیوتر، هر ساختمان داده‌ای که دارای هر 3 خاصیت زیر باشد، (در مورد آن عملیات‌های زیر پیاده‌سازی شده باشد) دیکشنری نام دارد:

- درج گره[1]
- جست‌وجو[2]
- حذف[3]

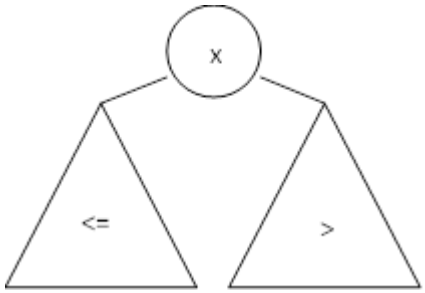
یکی از ساختمان داده‌های مهم و پرکاربرد از این دست، درخت جست‌وجوی دودویی نام دارد.

2- درخت جست‌وجوی دودویی[4]

درخت جست‌وجوی دودویی، درختی با ویژگی‌های زیر است:

1. درخت دودویی است. (هر گره حداکثر دو فرزند دارد.)

2. در این درخت، ارزش **هر** گره از تمام فرزندان سمت چپ خود بیشتر و از فرزندان سمت راست خود کمتر است. (برای حالت تساوی به دلخواه یکی از فرزندان چپ یا راست را به صورت قراردادی انتخاب می‌کنیم.) شکل 2.1



شکل 2.1

---

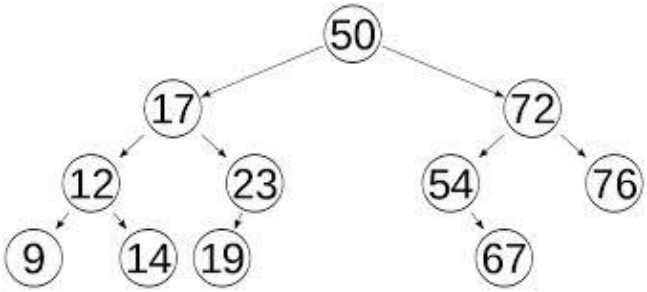
[1] Insert

[2] Search

[3] Delete

[4] Binary Search Tree (BST)

شکل‌های زیر، نمونه‌هایی از درخت جست‌وجوی دودویی را نشان می‌دهند.



شکل 2.2

شکل2.3

3. اگر روی درخت جست‌وجوی دودویی، الگوریتم میان‌ترتیب[1] را اجرا کنیم، داده های ما مرتب شده[2] هستند. برای مثال، در شکل 2.2 پیمایش میان‌ترتیب که اعداد را مرتب می‌کند برابر خواهد بود با:

$$9،12،14،17،19،23 \leq 50 < 54،67،72،76$$

[1] In-Order

[2] Sorted

2–1– عملیات‌های درخت جست‌وجوی دودویی

2-1-1-جست‌وجو:

جست‌وجو در BST به صورت بازگشتی انجام می‌شود. ابتدا داده مورد نظر را با ریشه‌ی درخت مقایسه می‌کنیم، اگر مقادیر برابری داشتند، عملیات جست‌وجو به اتمام رسیده و گره مورد نظر یافت شده‌است. اگر داده مورد جست‌وجو با ریشه برابر نبود، بررسی می‌کنیم که مقدار آن از ریشه کم‌تر است یا بیشتر. اگر کم‌تر بود، به زیردرخت سمت چپ رفته و عملیات جست‌وجو را برای فرزند سمت چپ ریشه به عنوان ریشه‌ی جدید ادامه می‌دهیم. به صورت مشابه اگر مقدار داده از ریشه بزرگ‌تر بود، عملیات را در زیر درخت سمت راست ادامه می‌دهیم. اگر گره مورد نظر موجود باشد، با این روش پیدا می‌شود، در غیر این صورت تابع مقدار NULL را بازمی‌گرداند.

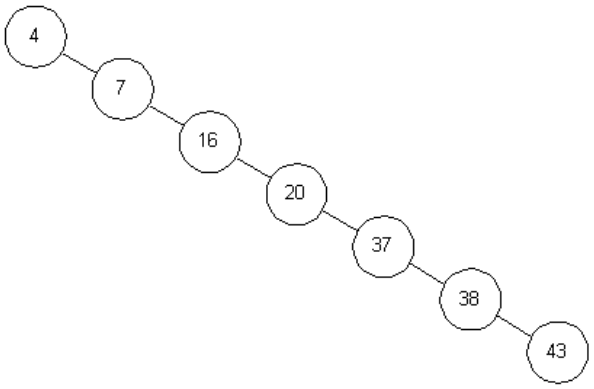
هزینه الگوریتم جست‌وجو در BST، این‌گونه است که در بدترین حالت گره‌ی مطلوب ما پایین‌ترین ارتفاع ممکن از ریشه‌ی درخت است. بنابراین هزینه آن O(h) است که در آن h ارتفاع درخت می باشد.

- نکته: اگر درخت کامل باشد، هزینه جست‌وجو به O(LOG N) تبدیل می‌شود، چون ارتفاع درخت برای درخت کامل LOG N است. در بدترین حالت ارتفاع درخت N خواهد بود. (هنگامی که ریشه اصلی درخت کوچک ترین کلید را داشته باشد و به همین ترتیب، تمام فرزندان آن به صورت مرتب قرار گیرند. شکل 2.4)

  بنابراین هزینه جست‌وجو بین O(LOG N) و O(N) تغییر می‌کند.

شبه کد زیر نحوه پیاده‌سازی جست‌وجو را نشان می‌دهد :

```
FUNCTION SEARCH (KEY, NODE):  // CALL INITIALLY WITH NODE = ROOT
  IF NODE = NULL OR NODE.KEY = KEY THEN
    RETURN NODE
  ELSE IF KEY < NODE.KEY THEN
    RETURN FIND-RECURSIVE (KEY, NODE.LEFT)
  ELSE
    RETURN FIND-RECURSIVE (KEY, NODE.RIGHT)
```



شکل2.4

2-1-2-درج گره:

برای اضافه‌کردن یک گره، جایی که عنصر باید اضافه شود را با الگوریتم جست‌وجو پیدا کرده و آن را اضافه می کنیم. نکته بسیار مهم در درخت جست‌وجوی دودویی این است که گره‌های اضافه شده، **همواره** برگ هستند.

هزینه درج $O(1)$ است. اما چون یک‌بار جست‌وجو انجام می‌شود، هزینه‌ی آن همان هزینه‌ی جست‌وجو یعنی $O(\text{Log } N)$ می‌باشد.

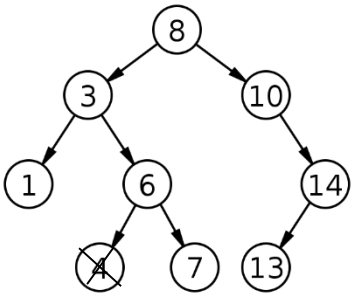
برای به دست آوردن شبه کد درج کردن، کافی است تغییرات اندکی در شبه کد جست‌وجو اعمال شود:

```
FUNCTION INSERT (KEY, NODE):  // CALL INITIALLY WITH NODE = ROOT
   IF NODE = NULL
 NODE = NEW NODE (KEY)
   ELSE IF KEY <= NODE.KEY THEN
  FIND-RECURSIVE (KEY, NODE.LEFT)
   ELSE
FIND-RECURSIVE (KEY, NODE.RIGHT)
   RETURN
```

2-1-3-حذف گره:

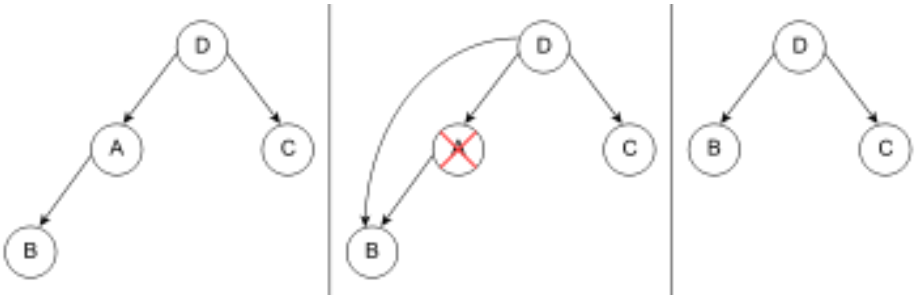
برای حذف یک گره در درخت جست‌وجوی دودویی، سه حالت مختلف ممکن است رخ دهد که در ادامه هر یک به صورت جداگانه شرح داده می‌شود:

i. **حذف یک برگ:** در این حالت به راحتی برگ مورد نظر را حذف می‌کنیم(شکل2.5) و هزینه این کار O(1) خواهد بود.



شکل2.5

ii. **حذف یک گره با یک فرزند:** در این حالت گره مورد نظر را حذف می‌کنیم و در ادامه پدر گره حذف‌شده را به فرزند آن وصل می‌کنیم.(شکل2.6) هزینه انجام این‌کار نیز همانند حالت قبل از O(1) است.



شکل2.6

iii. **حذف یک گره با دو فرزند**: این حالت پیچیده تر از دو حالت قبل است. در این حالت باید یکی از فرزندان گره مورد نظر جایگزین آن شود. اما این جایگزین می‌تواند از بین نوادگان آن نیز انتخاب گردد. دقت نمایید که در الگوریتم‌های حذف‌کردن و افزودن، ثابت نگه‌داشتن وضعیت و ویژگی‌های درخت به عنوان یک BST بعد از عملیات حذف و اضافه نکته‌ای مهم به شمار می‌رود.

(یکی از این ویژگی‌های درخت جست‌جوی دودویی، خاصیت مرتب بودن پیمایش میان ترتیب است.)

می‌دانیم گرهی که قرار است حذف شود ممکن است پدر داشته باشد. (مگر اینکه ریشه باشد.) تمام زیر درخت‌هایی که به واسطه‌ی گره مورد نظر ما (مثلا گره X) به‌وجود آمده‌اند، نسبت به پدر X همان رابطه را دارند که X داشته است. اگر X از پدرش (مثلا گره Y) بزرگ‌تر باشد، فرزندانش نیز از آن بزرگ‌ترند و بالعکس. بنابراین از نظر Y تفاوتی ندارد که کدام‌یک از فرزندان X، جای او را می گیرند. (یعنی بعد از حذف X، به فرزندان Y تبدیل می‌شوند.) بنابراین می‌توانیم X و فرزندانش را مستقلا بررسی نماییم. مشکلی که وجود دارد، رابطه‌ایست که بین فرزندان X وجود دارد. باید گره‌ای انتخاب شود که اگر جایگاه X را به ارث ببرد، تعادل زیر درخت حفظ شده و همچنان فرزندان باقی‌مانده در سمت چپ از جانشینِX کوچک‌تر یا مساوی و فرزندان سمت راست از آن بزرگ‌تر بمانند.

دو جانشین مناسب برای X پیدا می کنیم: یکی از فرزندان زیر درخت چپ و دیگری از فرزندان زیر درخت سمت راست.

به کوچک‌ترین گره زیردرخت راست گره، گره بعدی[1] گفته می‌شود. این نام‌گذاری به این خاطر است که اگر عناصر درخت با توجه به مقدار آنها مرتب شوند. (مثلا با پیمایش میان-ترتیب) این گره بلافاصله بعد از آن قرار می‌گیرد. به همین ترتیب، گره قبلی[2] یک گره نیز گرهی با بزرگ‌ترین مقدار در زیردرخت سمت چپ آن است. این گره نیز پس از مرتب‌سازی در کنار گره مفروض و قبل از آن قرار می‌گیرد. در حالت 3 برای حذف گره به جای عضو مابعد می‌توان از این گره نیز استفاده کرد.

کوچک‌ترین فرزند زیر درخت سمت راست همچنان از X و سایر راس‌ها در زیردرخت سمت چپ بزرگ‌تر و از تمام گره‌های سمت راست درخت کوچک‌تر است. از طرفی، بزرگ‌ترین فرزند زیر درخت سمت چپ از X و تمام راس های زیر درخت سمت راست کوچکتر و همچنین از تمام فرزندان زیر درخت سمت چپ بزرگتر است. این دو گره به خصوص، جانشین‌های مناسبی برای X خواهند بود.

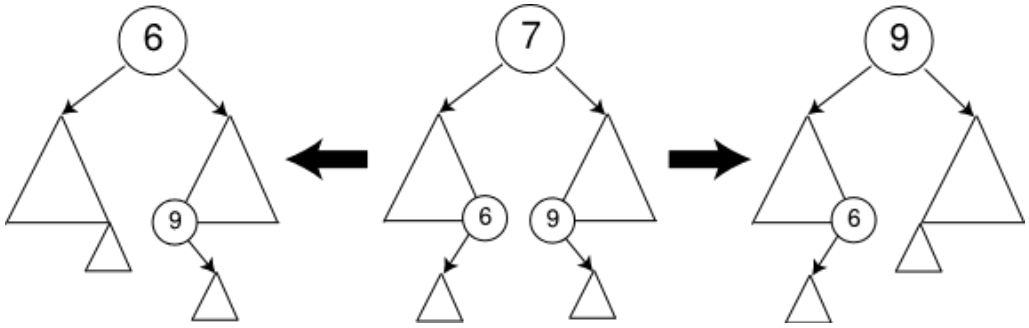
---

[1] Successor

[2] Predecessor

برای اینکه بهتر متوجه این موضوع شوید، درخت شکل2.2 و پیمایش میان‌ترتیب آن‌را در نظر بگیرید. فرض کنید می‌خواهیم ریشه درخت که مقدارش 50 است را از درخت حذف کنیم. برای جایگزین، یا بزرگ‌ترین گره زیردرخت سمت چپ (یعنی 23) و یا کوچک‌ترین گره زیردرخت سمت راست (یعنی 54) را جایگزین آن می کنیم. این دو عدد، دو مقدار مجاور 50 در پیمایش میان‌ترتیب از درخت هستند و چون پیمایش میان‌ترتیب درخت مرتب است، پس از حذف 50 و جایگزین کردن یکی از این دو عدد به عنوان ریشه، دنباله حاصل از پیمایش هم‌چنان مرتب خواهد ماند.

برای پیدا کردن بزرگ‌ترین راس زیر درخت سمت چپ، کافی است یک‌بار از X به زیردرخت وارد شویم (یک‌بار به چپ برویم) و سپس تا جای ممکن، به سمت راست حرکت کنیم. خلاف این حرکت برای زیر درخت سمت راست انجام می‌شود. گره به دست آمده حداکثر یک فرزند خواهد داشت به دلیل این‌که اگر بیش از یک فرزند داشته باشد، یکی از فرزندان آن از آن بزرگ‌تر/کوچکتر خواهد بود و می توانیم راسی بزگتر/کوچکتر از راس مورد نظر در شکل پیدا کنیم. بنابراین این راس حداکثر یک فرزند داردکه به یکی از دو روش حذف قبل، آن راس را حذف کرده و جایگزین X می‌کنیم. (شکل2.7)



شکل 2.7

برای اینکه درخت همواره مرتب بماند (پیمایش میان‌ترتیب آن مرتب بماند) در کل زمان استفاده از این ساختمان داده برای حذف یک گره با دو فرزند فقط از یکی از روش های اخیر استفاده می کنیم. برای یافتن راس جایگزین راس مورد نظر باید حداکثر به ارتفاع درخت دنبال آن بگردیم و این کار از O(h) است. هزینه حذف برگ نیز همواره O(h) است. (با در نظر گرفتن الگوریتم یافتن برگ مورد نظر)

2–1–4– پیدا کردن گره قبلی[1] و بعدی[2]:

در یک درخت جست‌وجوی دودویی، گاهی یافتن جانشین برای گره درخت یک در ترتیب منظم که با پیمایش میان‌ترتیب مشخص شده است، مسئله‌ای مهم به شمار می‌رود.

منظور ما از بعدی گره، گرهی است که در ترتیب مرتب شده کلید گره ها بلافاصله بعد از آن قرار گرفته باشد.

تعریف: در یک درخت جستجوی دودویی، بعدی گره x ، گرهی است که دارای حداقل مقدار key باشد که بزرگتر از key[x] است.

برنامهٔ زیر بدون انجام عمل مقایسه فیلد key ، اشاره‌گری به گره بعدی گره x برمی‌گرداند. (به شرطی که وجود داشته‌باشد!)

```
TREE-SUCCESSOR (x)
1 if right [x] <> NiL
2 then return TREE-MINIMUM (right[x])
3 y ‹- p[x]
4 while y<> NiL and x= right [y]
5 do x‹- y
6 y ‹- p[y]
7 return y
```

درخط 3 ، y مساوی پدر x می شود.

در خط 4، در ریشه y=NiL می شود چون ریشه، پدر ندارد.

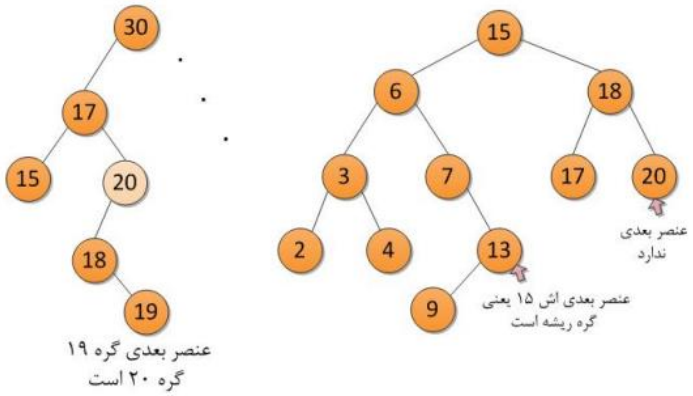
در خط 6،y مساوی پدر خودش می‌شود و p[y] اشاره‌گری به پدر y است.

الف- اگر زیر درخت راست x غیرتهی باشد، آنگاه Successor عنصر x، چپ‌ترین گره در زیردرخت راست است که در سطر2 برنامه پیدا می‌شود. به عنوان مثال:

---

[1] Successor

[2] Predecessor



Successor of 15 is 17 Successor of 6 is 7

ب- اگر زیر درخت راست x غیرتهی باشد، آنگاه بعدی عنصر x، عنصر y است بطوریکـه y کوچکترین جدّ عنصر x است که فرزند چپ آن نیز جدّی برای x باشد.

Y is the lowest ancestor of x whose left child is also an ancestor of x.

(ancestor : پدران یا اجداد)

اگر x=13 باشد، آنگاه y=15 پاسخ است چون 15 از پدران x است و 15 یک فرزند چپ 6 دارد از پدران 13 است.

پ- حالتی که زیر درخت راست x تهی است.

برای اینکه y یعنی بعدی گره x را پیدا کنیم، از گره x آنقدر به سمت بالا به سمت ریشه حرکت می‌کنیم تا به گرهی برسیم که فرزند چپ پدرش است، و در این مرحله از حلقه بیرون آمده و گره پدر این گره فرزند چپ، گره مورد نظر، یعنی y است. در شکل بالا اگر x=13 باشد، آنگاه y=15 خواهد بود. در مسیر حرکت از x=13 به ریشه به گره های 7 (فرزند چپ پدرش یعنی 6نیست) و سپس به گره 6 (فرزند چپ پدرش یعنی 15 است) می رسیم. بنابراین پدر گره 6 یعنی گره 15 گره مورد نظر ما است.

```
3y ‹- p[x]
4while y<> NiL and x= right [y]
5do x‹- y
6y ‹- p[y]
7return y
```

اگر تابع را با x=17 فراخوانی کنیم در سـطر 3،y=18 مـی‌شـود و سـپس در سـطر 4،y≠ NiL است ولی x≠ right [y] است بنابراین از حلقه خارج شد و در سطر 7 مقدار y=18 که به درستی بعدی گره x=17 است برگشت داده می‌شود.

یادآوری مهم : این تابع را نباید برای گره x که بزرگترین گـره درخـت اسـت و بعدی نـدارد صدا بزنیم.

زمان اجرای الگوریتم TREE-SUCCESSOR بر روی درختی با ارتفاع h مساوی O(h) اسـت. چون یا یک مسیر به طرف ریشهٔ درخت و یا مسیری به طرف پایین (یک برگ) را طی مـی‌کنـیم و حداکثر طول مسیر همان ارتفاع درخت است.

2-1-4-پیداکردن گره حداقل:

به علت خاصیتی که یک درخت جستجوی دودویی دارد گرهی کـه دارای کلیـد حـداقل اسـت را از طریق دنبال کردن مسیری که اشاره‌گرهای فرزند چپ نشان می دهنـد تـا بـه NiL برسـیم، مـی‌توانیم پیدا کنیم.

الگوریتم زیر این کار را انجام می دهد:

```
TREE-MINIMUM (x)
1 while left [x] <>NiL
2 then x ‹- left [x]
3 return x
```

الف –گره xکه با آن فراخوانی انجام شده هیچ فرزند چپی ندارد، در این صـورت درسـطر 3 خـودx برمی گردد.

ب- گره xکه فرزند چپ دارد، در این صورت سطر 2 آن آنقدر تکرار می شود تا گره x یک برگ شود و یا اینکه گرهی بشود که فرزند چپ ندارد.

در هر دو صورت این گره x گرهی خواهد بود که فیلد key آن حـداقل در کـل درخـت جسـتجوی دودویی خواهد بود.

5-1-2-پیدا کردن گره حداکثر:

```
TREE-MAXIMUM (x)
4 while right [x] <>NiL
5 then x ‹- right [x]
6 return x
```

عملکرد این الگوریتم مانند الگوریتم TREE-MINIMUM است، با این تفاوت که مسیر فرزند راست را طی می کند تا به گرهی برسد که فرزند راست نداشته باشد.

فیلد key این گره حاوی حداکثر key این درخت خواهد بود.

زمان اجرای هر یک از دو الگوریتم فوق O(h) است که h ارتفاع درخت دودویی است.

2–2– نکات مهم :

i. بسیاری داده ها، قابل مرتب‌سازی نیستند و یا گاهی داده‌ها بسته هایی هستند از متغیرها (مانند کلاس یا Struct.) در این موارد به هر داده یک کلید نسبت می‌دهیم که قابل مرتب‌سازی باشند، سپس درختی را با استفاده از این کلیدها ایجاد کرده و عملکرها را روی آن انجام می‌دهیم.

ii. در درخت جست‌وجوی دودویی، بسته به درج‌کردن و حذف‌کردن‌هایی که روی آن انجام می‌شود، تقارن و توازن بعد از مدتی از بین رفته و ارتفاع درخت از $\log n$ به n تغییر می کند و هزینه‌ی زمانی الگوریتم‌ها افزایش می یابد. (در BST تمامی الگوریتم‌ها $O(h)$ هستند.) برای این‌که همواره تقارن و توازن برقرار بماند، درخت‌های دودویی دیگری به وجود آمده‌اند.
Red-Black Tree، AVL و B-Tree درختهایی هستند که با انجام محاسبات خاص در هنگام اعمال عملگرها، هزینه الگوریتم‌های دیکشنری در آن‌ها $O(\log n)$ می باشد.

iii. شبه کد برای پیمایش میان‌ترتیب داده‌ها:

```
Function Inorder-Tree-Walk (x)
Ifx ≠ NIL
Then Inorder-Tree-Walk (left[x])
Print key[x]
Inorder-Tree-Walk (right[x])
```

این الگوریتم، هزینه‌ای معادل $\theta(n)$ خواهد داشت. (قضیه 12.1 کتاب CLRS.)

iv. روش دیگر برای متوازن ساختن درخت جست‌وجوی دودویی، این است که ابتدا روی درخت الگوریتم میان‌ترتیب را اجرا نماییم تا دنباله مرتب داده‌ها به دست آید. سپس مجددا به صورت بازگشتی، درخت را می‌سازیم. هزینه این الگوریتم برابر هزینه الگوریتم میان‌ترتیب یعنی $\theta(n)$ خواهد بود.

v. تعداد درخت‌های BST با ارتفاع n-1 برابر است با 2^{n-1}
هم‌چنین تعداد درخت‌های BST با ارتفاع n-2 برابر است با $2^{n-2}(n-2)-1$