دانشکده علوم ریاضی و آمار

مدرس: **دکتر مجتبی رفیعی** نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

# جلسه ۳۵

زهرا احمدی خونسارکی

۷ دی ۱۴۰۰

## فهرست مطالب

۱ **عمل درج در BST** ۱

۲ **عمل حذف از BST** ۳

۳ **برخی کاربردهای درخت BST** ۴

## ۱ عمل درج در BST

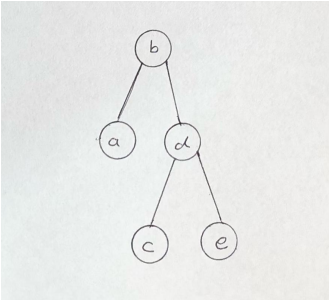
عمل درج در BST که حافظ تعادل ارتفاع در درخت دودویی باشد،پیچیده است و دراینجا تنها روی درج ساده متمرکز هستیم.
**نکته:** داده ساختار های مانند AVL و Red-Black درخت هایی هستند که درج در آنها به نحوی است که موجب ساخت و حفظ درخت متعادل می شوند.
برای درج یک عنصر در BST دو رویکرد کلی وجود دارد:

- عنصر جدید را در کنارعناصر دیگر قرار داده و سپس مبادرت به ساخت درخت BST می کنیم. در این حالت مجبوریم کل ساختار را از ابتدا بسازیم که پیچیدگی آن از مرتبه $O(nlogn)$ است.

**مثال:** توالی زیر را در نظر بگیرید:

decab

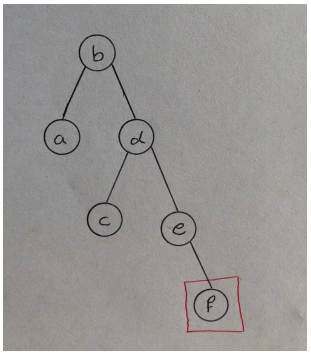
اگر ترتیب الفبایی را در نظر بگیریم درخت زیر، درخت BST حاصل از توالی بالا است.




حال فرض کنید عنصر f را بخواهیم درج کنیم ،با رویکرد ۱، توالی زیر را داریم:

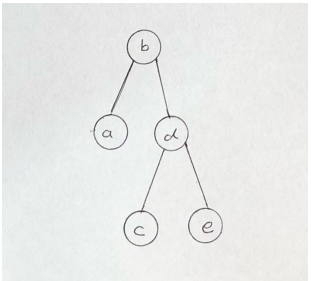
decabf

و مجددا مبادرت به ساخت درخت BST می کنیم.

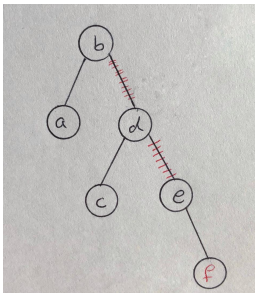



واضح است که ساخت درخت BST از روی توالی و بدون در نظر گرفتن BST ساخته شده از عناصر قبلی، طبق صحبت‌های جلسه قبل دارای پیچیدگی $O(nlogn)$ است.

- در یک رویکرد دیگر، ابتدا مکان مناسب برای عنصر مورد نظر را در درخت BST پیدا می کنیم، سپس یک نود جدید ایجاد می‌کنیم و اشاره‌گرها را به نحو مناسبی بروزرسانی می‌کنیم. پرواضح است که در این حالت، از ویژگی BST بودن درخت تا قبل از درج عنصر جدید اضافه می کنیم و پیچیدگی آن $O(n)$ است. **مثال:** درخت BST توالی مثال قبل را در نظر بگیرید،




برای درج عنصر f ابتدا مکان مناسب را پیدا می کنیم، سپس نود جدید را می‌سازیم و مقدار نود و اشاره‌گرها را به روز می‌کنیم.




**جمع بندی:** هر دو رویکرد تشریح شده برای درج یک عنصر جدید در BST، ویژگی BST بودن را حفظ می‌کند اما هیچ گونه ضمانتی روی حفظ تعادل درخت BST ندارد.
الگوریتم درج عنصر با استفاده از رویکرد ۲ به صورت زیر می‌باشد:

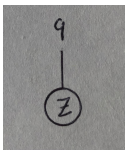
```
BST-Insert(T,Z)
  y=Null
  x=T.root
  while ( x!=Null) do
      y=x
      if (z.key < x.key) then
          x=x.left
      else
          x=x.right
  Z.parent=y
  if (y==Null) then
      T.root=Z
  else if (z.key < y.key) then
      y.left=Z
  else
      y.right=Z
```

**پیچیدگی زمانی الگوریتم درج:** O(h)، جاییکه h بیانگر عمق درخت T است.

## ۲ عمل حذف از BST

عمل حذف کمی پیچیده است و می بایست حالت های مختلف برای حذف گره درخواستی را در نظر بگیریم:

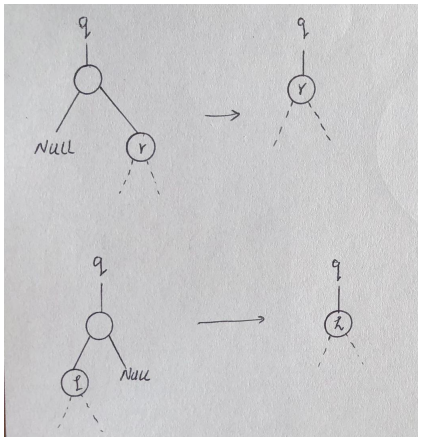
(۱) فرض کنید گره درخواستی برای حذف (Z) یک گره برگ باشد ،یعنی



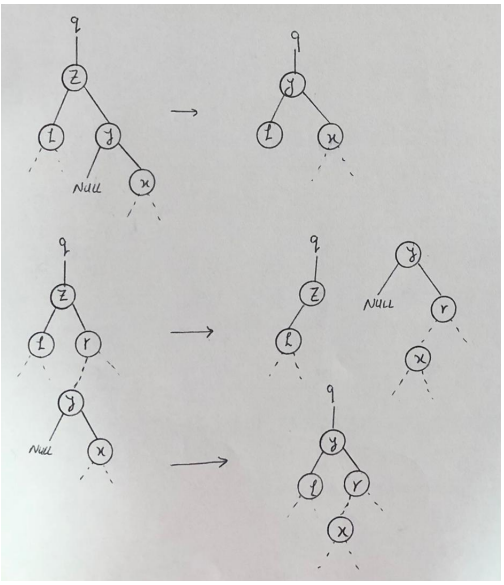

شکل فوق بدین معناست که q گره پدر Z بوده و Z می تواند فرزند سمت چپ یا فرزند سمت راست باشد.

در این حالت برای حذف گره Z تنها کافی است که اشاره گر سمت چپ یا راست را با Null مقداردهی کنیم.

(۲) فرض کنید گره درخواستی برای حذف (Z) تنها یک فرزند داشته باشد:




(۳) فرض کنید گره درخواستی برای حذف (Z) ،دارای دو فرزند باشد: در این حالت می بایست successor گره Z را پیدا کرده و به نحو مناسبی آن رابا Z جایگزین کنیم.




پیچیدگی زمانی الگوریتم حذف از مرتبه O(h) است،جاییکه h بیانگر عمق درخت BST است.

## ۳ برخی کاربردهای درخت BST

(۱) جست و جوی یک کلید

(۲) مرتب سازی:پیمایش Inorder درخت BST (مرتب سازی صعودی)

سوال:مرتب سازی نزولی را برای یک درخت BST چگونه به دست آوریم؟

(۳) صف الویت:درج در صف بر اساس الویت انجام می شود(برعکس صف ساده که همه عناصر هم الویت در نظر گرفته می شود).

بالاترین الویت:
•عدد کمتر: BST-Minimum
•عددبیشتر: BST-Maximum