عمل درج (Insertion) در BST:

عمل درج در BST که حافظ تعادل ارتفاع در درخت دودویی باشد، پیچیده است و در اینجا تنها روی درج ساده متمرکز هستیم.

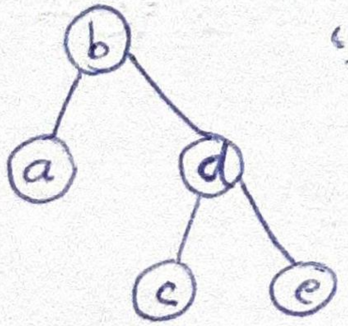
نکته: داده ساختارهای مانند AVL و Red-Black، درخت‌هایی هستند که درج در آنها به نحوی است که موجب ساخت و حفظ درخت متعادل می‌شوند.

برای درج یک عنصر جدید در BST دو رویکرد کلی وجود دارد:

رویکرد ۱: عنصر جدید را در کنار عناصر دیگر قرار داده و سپس مبادرت به ساخت درخت BST می‌کنیم. در این حالت مجبوریم کل ساختار را از ابتدا بسازیم که پیچیدگی آن $O(n\log n)$ است.

مثال: توالی روبرو را در نظر بگیرید: decab

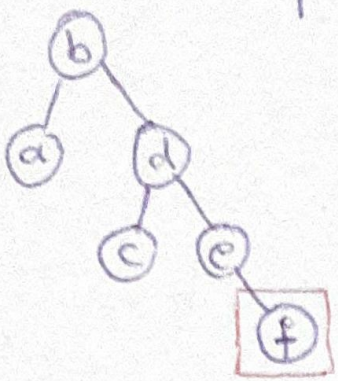
اگر ترتیب الفبایی را در نظر بگیریم درخت روبرو، درخت BST حاصل از توالی بالا است.



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

decabf

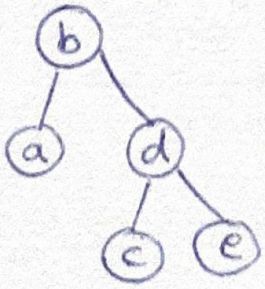
و سعی داریم به ساخت درخت BST می‌کنیم:



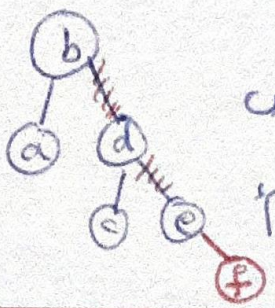
واضح است که ساخت درخت BST از روی درآلی و مثل در نظر گرفتن BST ساخته شده از عناصر قبلی، طبق صحبت‌های جلسه قبل برای ساخت BST، دارای پیچیدگی $O(n\log n)$ است.

روش کرد ۲: در این روش کرد، ابتدا امکان مناسب برای عنصر مورد نظر را در درخت BST پیدا می‌کنیم، پس یک نود جدید ایجاد می‌کنیم و اشاره‌گرها را به نحو مناسبی بروز رسانی می‌کنیم. بدیهی است که در این حالت، از دوباره BST بودن درخت تا قبل از درج عنصر جدید اطمینان داریم و پیچیدگی آن $O(\log n)$ است.

مثال: درخت BST مثال قبل را در نظر بگیرید،



برای درج عنصر f، ابتدا امکان مناسب را پیدا می‌کنیم، پس نود جدید را می‌سازیم و مقدار نود و اشاره‌گرها را به روز می‌کنیم:



جمع‌بندی: هر دو رویکرد تشریح شده برای درج یک عنصر جدید در BST، درخت BST بودن را حفظ می‌کند اما هیچ گونه ضمانتی روی حفظ متوازن بودن درخت BST ندارد.

الگوریتم درج عنصر جدید با استفاده از رویکرد ۲ به صورت زیر می‌باشد:

```
BST_Insert(T, z)
1.  y = Null        // y مراقبت والد گره z را نگه دارد
2.  x = T.root
3.  while (x ≠ Null) do
4.      y = x
5.      if (z.key < x.key) then
6.          x = x.left
7.      else
8.          x = x.right
9.  z.Parent = y
10. if (y == Null) then
11.     T.root = z
12. else if (z.key < y.key) then
13.     y.left = z
14. else
15.     y.right = z
```

پیچیدگی زمانی الگوریتم درج: $O(h)$، جایی که h بیانگر عمق درخت T است.

عمل حذف (Deletion) در BST:

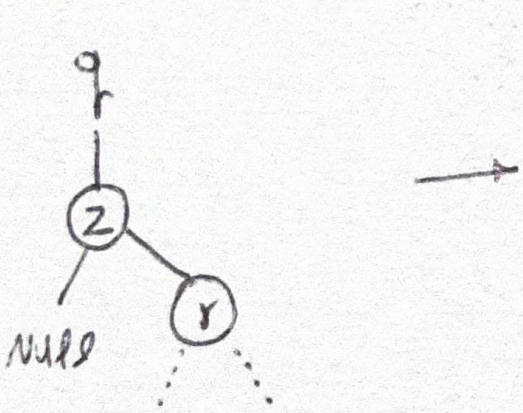
عمل حذف کمی پیچیده است و می‌بایست حالت‌های مختلف برای حذف گره درخواستی را در نظر بگیریم:

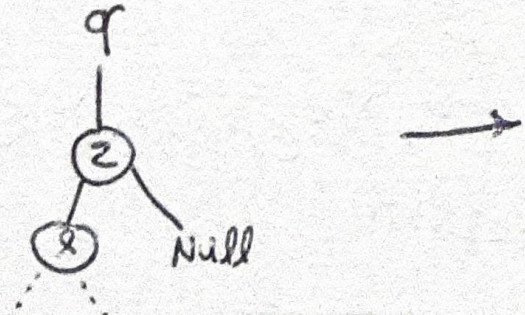
① فرض کنید گره درخواستی برای حذف (z)، یک گره برگ باشد، یعنی

شکل فوق بدین معناست که q گره پدر z بوده و z می‌تواند فرزند سمت چپ یا فرزند سمت راست باشد.

در این حالت برای حذف گره z تنها کافی است که اشاره‌گر راست چپ یا راست q را Null مقدار دهی کنیم.

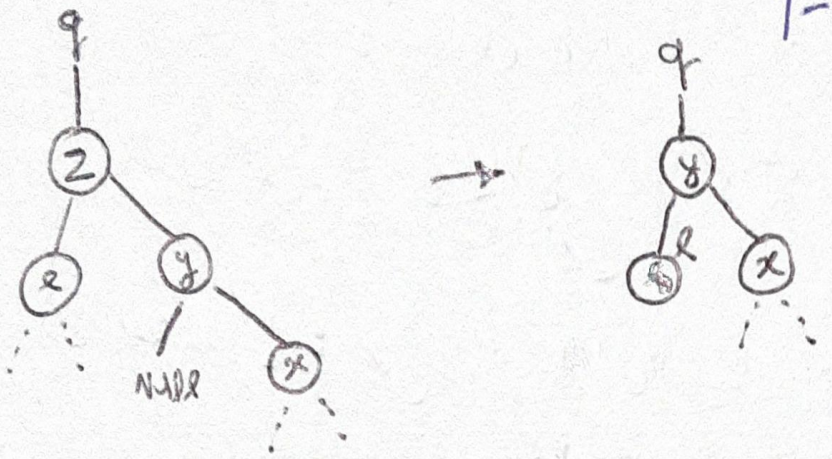
② فرض کنید گره درخواستی برای حذف (z)، تنها یک فرزند داشته باشد

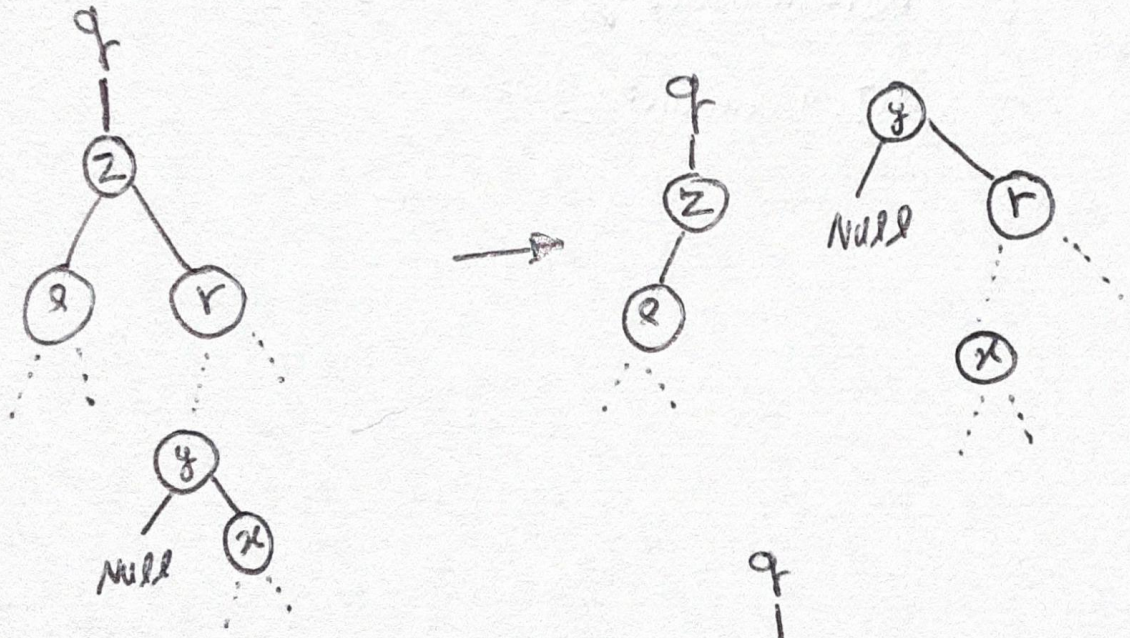


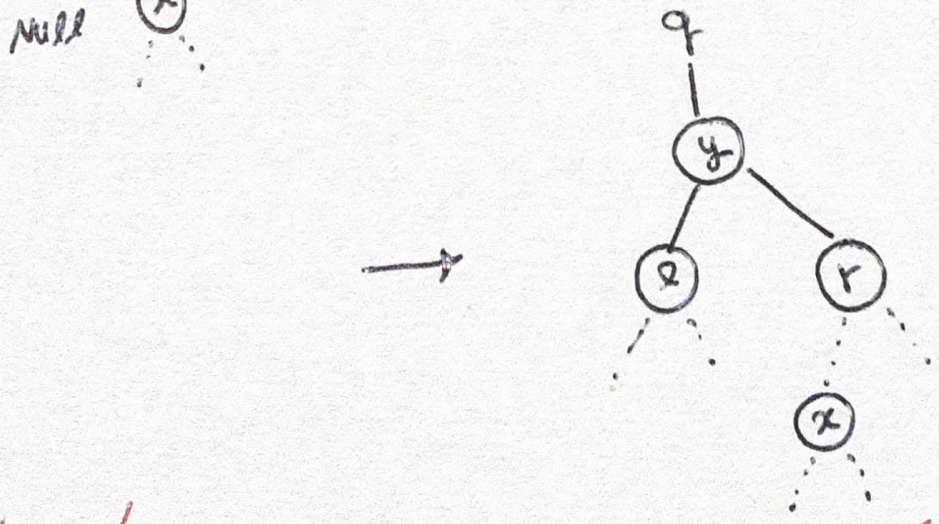


③ فرض کنید گره مورد نظر برای حذف (z)، دارای دو فرزند باشد:

در این حالت باید successor گره z را پیدا کرده و به جای مناسب آن را با z جایگزین کنیم.







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

برخی کاربردهای درخت BST:

① جستجو وجود یک کلید

② مرتب سازی: پیمایش Inorder درخت BST (مرتب سازی صعودی)

سؤال: مرتب سازی نزولی را با یک درخت BST چگونه پیاده سازی کنیم؟

③ صف الویت: درج در صف براساس الویت انجام می شود (برعکس صف ساده که همه عناصر هم الویت در نظر گرفته می شود).

BST-Minimum: عدد کمتر ← بالاترین الویت

BST-Maximum: عدد بیشتر