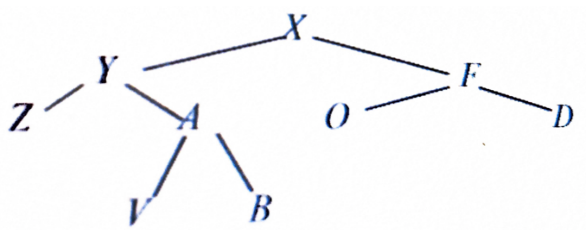
پاسخنامه تمرین شماره ۳

سوال ۱

مراحل باز سازی درخت:

- X ریشه ی درخت است.
- Y فرزند چپ X می باشد.
- Z برگ است و به عنوان فرزند چپ Y قرار می گیرد. زیردرخت بعدی(ABV) جایگاه فرزند راست X را اشغال می کند و درخت به بن بست می خورد. در حالی که هنوز سه گره باقی مانده است.
- زیردرخت ABV (A پدر دو برگ B و V است) در جایگاه فرزند راست Y قرار می گیرد.در این جا کار زیر درخت چپ X به پایان می رسد.
- F فرزند راست X است.
- برگ های O و D به ترتیب فرزندان چپ و راست F هستند.



- پس پیمایش پس ترتیب درخت ZBVAYODFX خواهد شد.

سوال ۲

اگر شما در کنار preoder یا postorder، inorder را داشته باشیم شما همیشه می توانید یک درخت باینری منحصر به فرد را ایجاد کنید، زیرا اطلاعات pre و post به شما ریشه درخت را می دهد. هنگامی که ریشه درخت را می دانید، می توانید به طور مجدد زیر درختان چپ و راست که خود درخت های دودویی هستند را ایجاد کنید و بنابراین محصول نهایی unique خواهد بود.

درخت دودویی به تنهایی از مسیرهای preorder و postorder به دست نمی آید، زیرا تنها inorder موقعیت زیردرخت ها نسبت به ریشه را مشخص می کند.

برای دو حالت اول الگوریتم اراءه می دهیم و حالت سوم را با یک مثال نقض رد می نماییم.

الف) مثال نقض می توانید به لینک زیر مراجعه کنید:

https://www.geeksforgeeks.org/?p=657

توجه شود که در حالت کلی نمی توان با داشتن این دو پیمایش به درخت اصلی رسید اما اگر بدانیم درخت کامل است دیگر ابهاماتی در ساخت و ایجاد درخت ایجاد نخواهد شد به لینک زیر مراجعه کنید:

https://www.geeksforgeeks.org/full-and-complete-binary-tree-from-given-preorder-and-postorder-traversals/

ب و ج) از لینک زیر کمک بگیرید:

https://www.geeksforgeeks.org/print-postorder-from-given-inorder-and-preorder-traversals

سوال ۳

تعداد درخت جستجو دوتایی =
(تعداد راه های انتخاب ریشه)*
(تعداد زیر درخت های جستجوی دودویی چپ) *
(تعداد زیر درخت های جستجوی باینری سمت راست)

در حال حاضر، از آنجا که "n" گره در BST وجود دارد، تعداد BST با n گره را برابر C (n) در نظر می گیریم.

ما می توانیم تعداد BST ها را به صورت بازگشتی به صورت زیر پیدا کنیم:

- ۱ را به عنوان ریشه انتخاب کنید، هیچ عنصری در زیر درخت چپ نباشد. عناصر n-1 در زیر درخت راست.
- ۲را به عنوان ریشه انتخاب کنید، ۱ عنصر در زیر درخت چپ و n-2 عناصر در زیر درخت راست.
- ۳را به عنوان ریشه انتخاب کنید، ۲ عنصر در زیر درخت چپ و n-3 عناصر در زیر درخت راست.

به طور مشابه، برای عنصر i ام به عنوان ریشه، عناصر i-1 در سمت چپ و n-i در سمت راست.

این زیردرخت ها نیز BST هستند، بنابراین ما می توانیم بنویسیم:

C (n) = C (0) C (n-1) + C (1) C (n-2) + ... + C (i-1) C (ni) ... + C (n-1) C (0)

C (0) = 1، چون دقیقا یک روش برای ایجاد BST با 0 گره وجود دارد.C (1) = 1 ، چون دقیقا یک روش برای ساخت یک BST با 1 گره وجود دارد.

$$C(n) = \sum_{i=1}^{n} C(i-1)\, C(n-i)$$

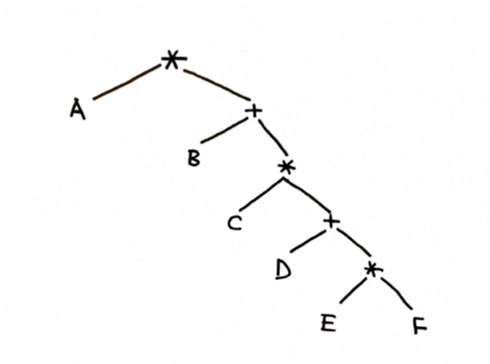
جمع بندی فوق به عدد کاتالان خواهد رسید.

$$C(n) = \frac{\binom{2n}{n}}{n+1}$$

سوال ۴

الف)

Preorder: *A + B * C + D * EF



از پیمایش پس ترتیب و میان ترتیب درخت عبارت به دو پاسخ زیر خواهیم رسید:

Postorder:ABCDEF*+*+*
Inorder: A*(B+(C*(D+(E*F))))

ب) به طور مشابه الف حل خواهد شد.

سوال ۵

از یک استک و آرایه کمکی(mark) کمک می گیریم. ابتدا ریشه را در درون استک push می کنیم. آرایه mark را برای این گرفتیم که چک کنیم آیا گره i را قبلا پیمایش کردیم یا خیر. در صورت پیمایش (یا true بودن) دیگر به پیمایش آن نمی پردازیم . در هر مرحله عنصر بالای استک را در نظر می گیریم مثلا V سپس اگر بچه سمت چپ این گره false بود آن را در استک push می کنیم در غیر این صورت خود v را بررسی می کنیم .mark[v] را true می کنیم اگر هم از قبل true بود سراغ بچه سمت راست گره v می رویم. اگر v.right نیز TRUE بود V را POP می کنیم .در غیر این صورت v.rightرا push در استک می کنیم .در این روش inorder درخت را غیر بازگشتی طی کردیم.