ساختمان داده ها ۲۲۸۲۲

نیم‌سال اول ۱۴۰۱-۱۴۰۰

مدرس: مدرس: حسین بومري

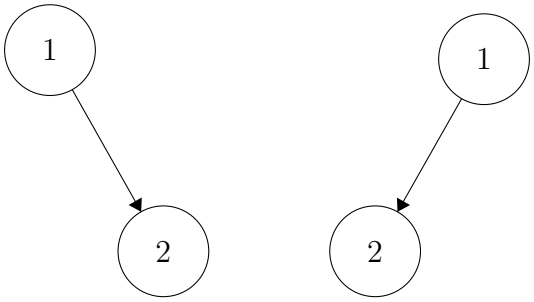
دانشکده علوم ریاضی

بحث بیشتر InFix و PreFix و PostFix

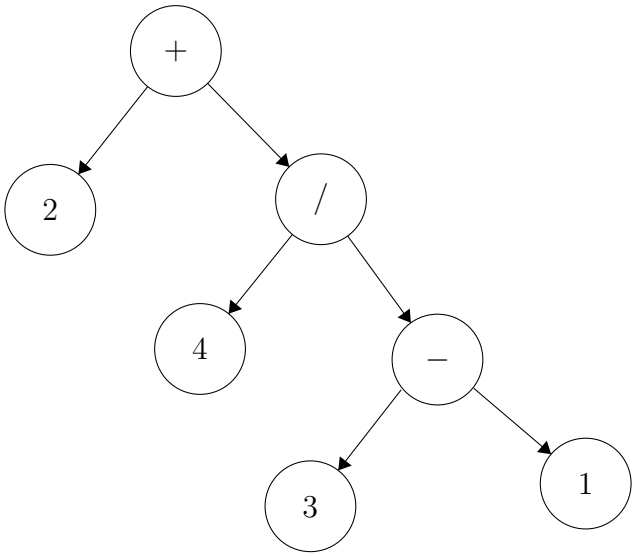
نگارنده: آرمیتا جلالیون

جلسه‌ی : بیست‌ویکم

در انتهای جلسه‌ی بیستم با پیمایش‌های مختلف مانند InFix و PreFix و PostFix آشنا شدیم.
گفتیم که در پیمایش inFix می‌توانیم به سورت شده‌ی درخت دسترسی داشته باشیم و از مزایای PreFix و PostFix این است که به ساختار درخت جست‌وجوی دسترسی داریم.
اگر درختمان درخت جست‌وجوی دودویی نباشد با داشتن تنها PreFix یا تنها PostFix نمی‌توانیم درخت را تشخیص دهیم اما با داشتن هر دوی این پیمایش‌ها می‌توانیم درخت را تشخیص بدهیم.
در مثال زیر PreFix در هر دو درخت به صورت 2 1 است



در نتیجه مشکل داشتن تنها یک پیمایش این است که تفاوت سمت چپ و راست برای ما قابل تشخیص نیست.
حالا فرض کنید در مثال بالا هم PreFix و هم PostFix را داشته باشیم. خواهیم‌دید نمایش PreFix هر دو درخت به صورت 2 1 و نمایش PostFix هر دو درخت به صورت 1 2 می‌باشد و نمی‌توانیم به کمک داشتن هر دو پیمایش به ساختار درخت پی ببریم.
در حالتی که یک راس یک بچه داشته باشد فقط نمی‌توانیم مشخص کنیم که این راس بچی چپ است یا راست و در نتیجه نمی‌توانیم تعیین کنیم ساختار درخت به چه صورت است.
درخت محاسباتی زیر را در نظر بگیرید:



PreFix : + 2 / 4 - 3 1
PostFix : 2 4 3 1 - / +

در این حالت اگر بخواهیم حاصل درخت را محاسبه بکنیم، درحالت PreFix می‌توانستیم stack داشته باشیم و از چپ شروع می‌کنیم و هر چیزی دیدیم در استک وارد می‌کنیم(چه عملگر و چه عدد)
این‌کار را تا جایی ادامه می‌دهیم که عددی ببینیم که قبل از آن هم عدد در استک push شده در این صورت دو تا عدد اخیر را پاپ می‌کنیم و به عملگر می‌رسیم (قطعا به عملگر میرسیم چرا که عددها همیشه برگ اند)
در نتیجه در مثال بالا وقتی به یک می‌رسیم، یک و سه و - را از استک پاپ می‌کنیم و حاصل آن‌ها را به صورت $(1-3)$ در انتهای استک وارد می‌کنیم (ترتیب مهم است).
و این عمل را تا جای ممکن ادامه می‌دهیم تا محاسباتمان کامل شود.

```
for a in list:
   if S.top is not value:
      S.push(a)
   else if a is operator:
      S.push(a)
   else:
      Compute(a)

\\

Compute(a):
   L <- S.pop
   O <- S.pop
   R <- a
   V <- LOR
   if S.top is value:
      Compute(V)
   else:
      S.push(V)
```

راه‌حل ما برای PostFix بسیار به راهمان برای PreFix شباهت دارد و اگر از انتها به ابتدای آن حرکت کنیم بسیار مشابه PreFix خواهد بود.
اما فرق آن در قسمت Compute است. در این حال کافی‌ست در compute بالا جای L و R را عوض کنیم و آن را به صورت

```
Compute(a):
   R <- S.pop
   O <- S.pop
   L <- a
   V <- LOR
   if S.top is value:
      Compute(V)
   else:
      S.push(V)
```

بنویسیم.
سوال:
فرض کنید که از آخر به اول نمی‌توانیم حرکت کنیم.حالا چه‌جوری محاسبه کنیم؟

سوال:
اگر عملگرهای ما اولویت داشته باشد چه‌جوری محاسبه را انجام دهیم؟

برای مثال می‌خواهیم عملیات $2 + 3 * 4 - 5\ pow\ 6$ را محاسبه کنیم. (منظور از pow همان توان است)
برای این‌کار خوب است که تابعی داشته باشیم که با گرفتن هر operand اولویت آن را به ما میدهد.
برای این که PostFix این عبارت را بنویسیم از سمت چپ شروع می‌کنیم. اگر عدد دیدیم مستقیم آنرا در خروجی می‌نویسیم و اگر operand دیدیم، آن را در استکی که داریم push می‌کنیم.
وقتی عملگر بعدی را دیدیم، اگر اولویت این عملگر بیشتر از عملگرهای قبلی استک باشد آن را در استک push می‌کنیم و اگر کمتر باشد، آن‌قدر پاپ می‌کنیم تا به اولویت کم‌تر برسیم.
به عبارتی استک ما از اولویت کم به زیاد همیشه مرتب است و هر چه به کف استک نزدیک میشویم اولویت کم می‌شود.
در مثالی که زده بودیم PostFix آن به صورت $2\ 3\ 4\ *\ +\ 5\ 6\ pow\ -$ نوشته می‌شود.
نگه‌داری درخت‌هایی با بیش از دو فرزند:
به‌ازای هر node می‌آییم فرزند چپ و هم‌نسل راستش را نگه می‌داریم $left\ child\ right\ sibling$ خوبی این درخت این است که حافظه‌ی کمی مصرف می‌کند و می‌تواند هر درخت را تبدیل به یک درخت دودویی کند.