ساختمان داده ها

پیمایش درخت دودویی توسط پیمایشگرها

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

مساله

- می خواهیم گره های درخت دودویی را توسط Iterator ها به صورت های زیر پیمایش کنیم:
  - Inorder –
  - Preorder –
  - Postorder –
- می خواهیم بتوانیم هر پیمایش را در دو جهت مستقیم و معکوس انجام دهیم.

پیمایش سطح به سطح و پیمایش برگ ها

- همچنین می خواهیم یک Iterator داشته باشیم که گره های درخت را سطح به سطح پیمایش کند (نیازی به پیمایش معکوس سطح های درخت نیست).
- همچنین می توانیم یک Iterator داشته باشیم که بر روی برگ های درخت و از چپ به راست (و یا برعکس) حرکت می کند.

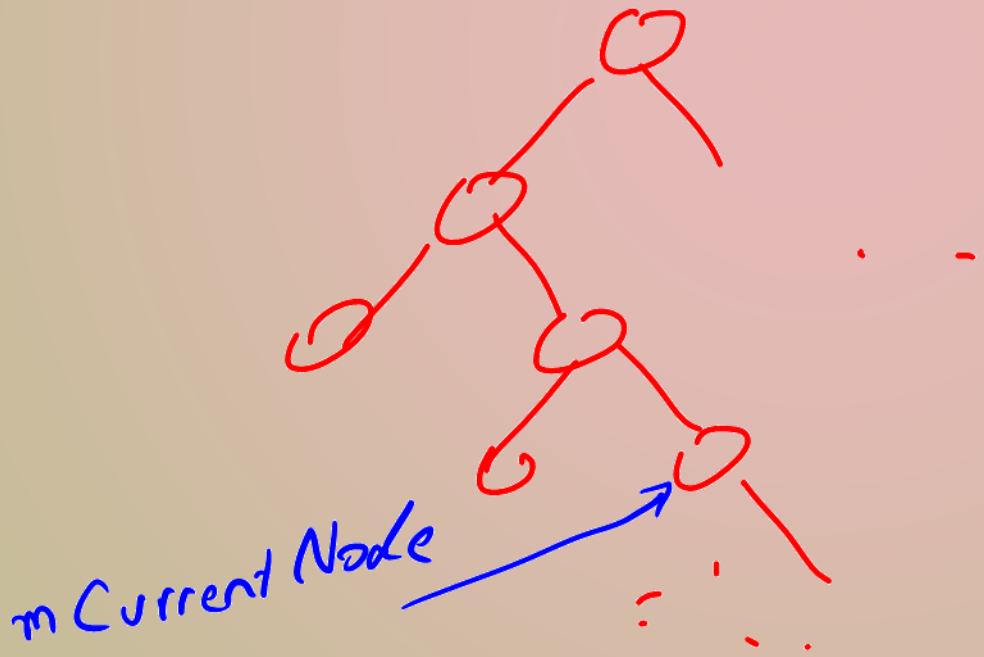
عملیات متناظر با هر پیمایش

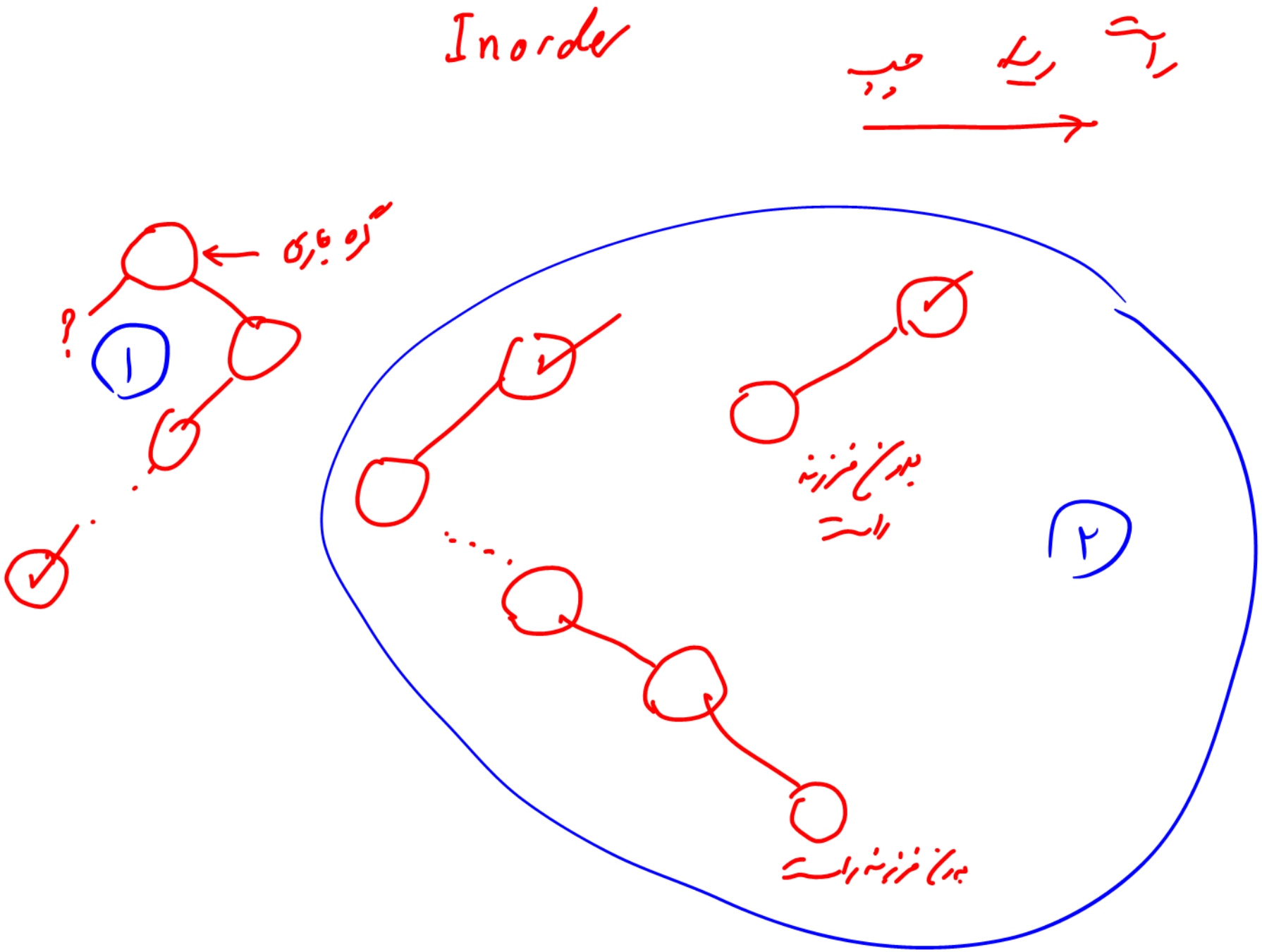
- در کلاس BinaryTree و برای هر پیمایشگر (XXX)
- XXXbegin()
- XXXend()
- XXXrbegin()
- XXXrend()
- در کلاس پیمایشگر
- operator++
- operator—
- سوال: سرعت در کدام عملیات اهمیت بیشتری دارد؟

hasNext() و next()

چارچوب نوشتن ++ و -- برای هر پیمایشگر

- فرض می کنیم که در گره mCurrentNode هستیم.
- گرهی را که پس از mCurrentNode باید مشاهده شود به دست می آوریم.



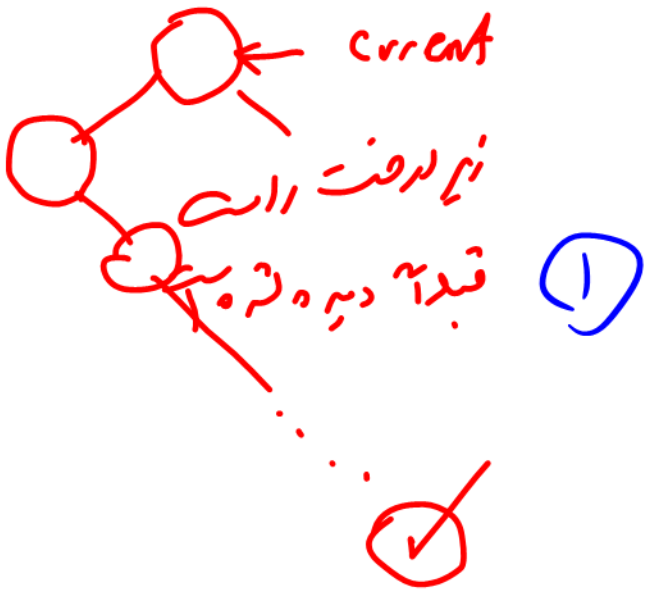

Inorder
ریشه
چپ
راست
گره جاری
?
۱
۲
بعد از فرزند راست
بعد از فرزند راست

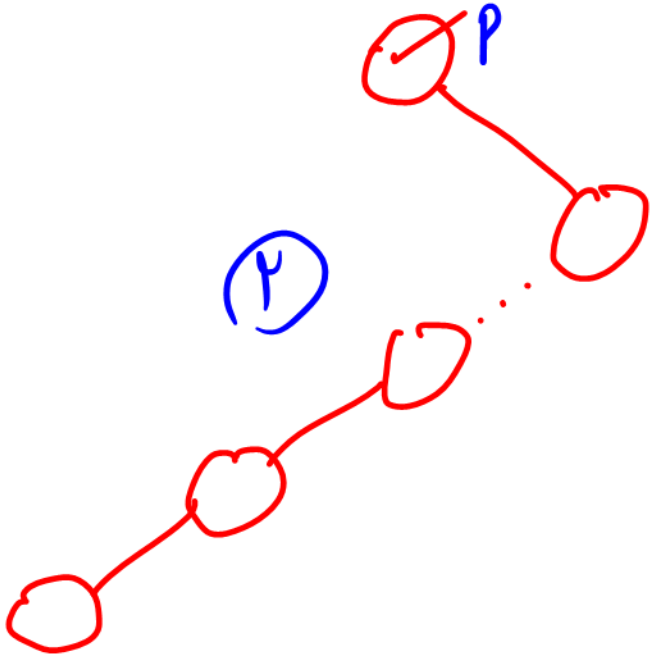
پیمایش Inorder پیش رو

- زیر درخت سمت چپ دیده شده است.
- اگر زیر درخت سمت راست خالی نباشد باید وارد آن شویم.
  - در زیر درخت سمت راست تا آنجا که می توانیم به سمت چپ می رویم. گرهی که در آن متوقف می شویم گره بعدی است.
- اگر زیردرخت سمت راست خالی باشد، باید پدر mCurrentNode را بررسی کنیم.
  - از گره جاری به سمت ریشه بالا می رویم تا به گرهی مانند p برسیم که گره جاری در زیردرخت چپ آن باشد. گره بعدی همان گره p است.

Reverse Inorder چپ ریشه راست

با فرض وجود فرزند چپ



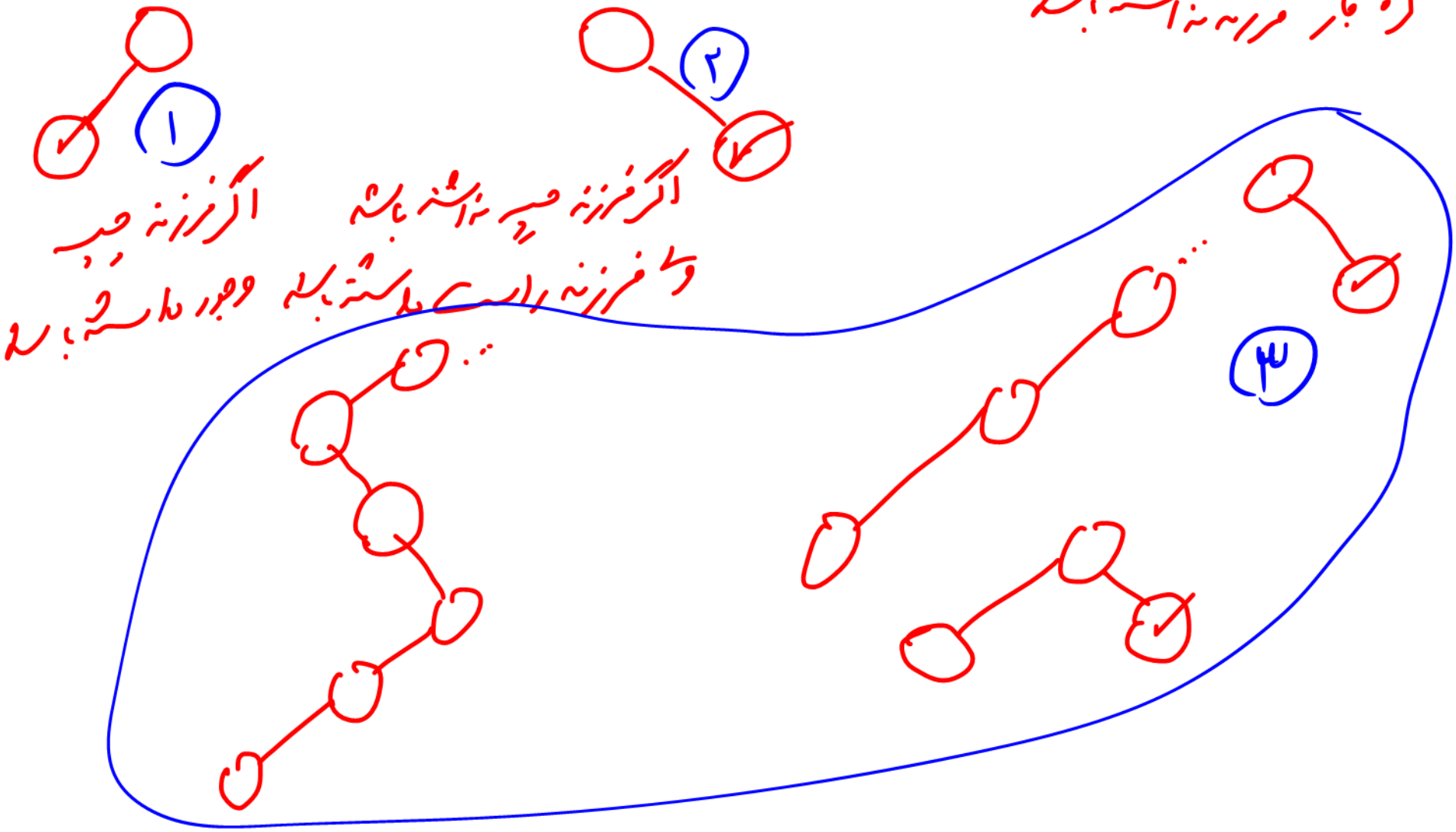


پیمایش Inorder پس رو

- زیر درخت سمت راست دیده شده است.
- اگر زیر درخت سمت چپ خالی نباشد باید وارد آن شویم.
  - در زیر درخت سمت چپ تا آنجا که می توانیم به سمت راست می رویم. گرهی که در آن متوقف می شویم گره بعدی است.
- اگر زیردرخت سمت چپ خالی باشد، باید پدر mCurrentNode را بررسی کنیم.
  - از گره جاری به سمت ریشه بالا می رویم تا به گرهی مانند p برسیم که گره جاری در زیردرخت راست آن باشد. گره بعدی همان گره p است.

Preorder ریشه چپ راست

گره جاری فرزند نداشته باشد

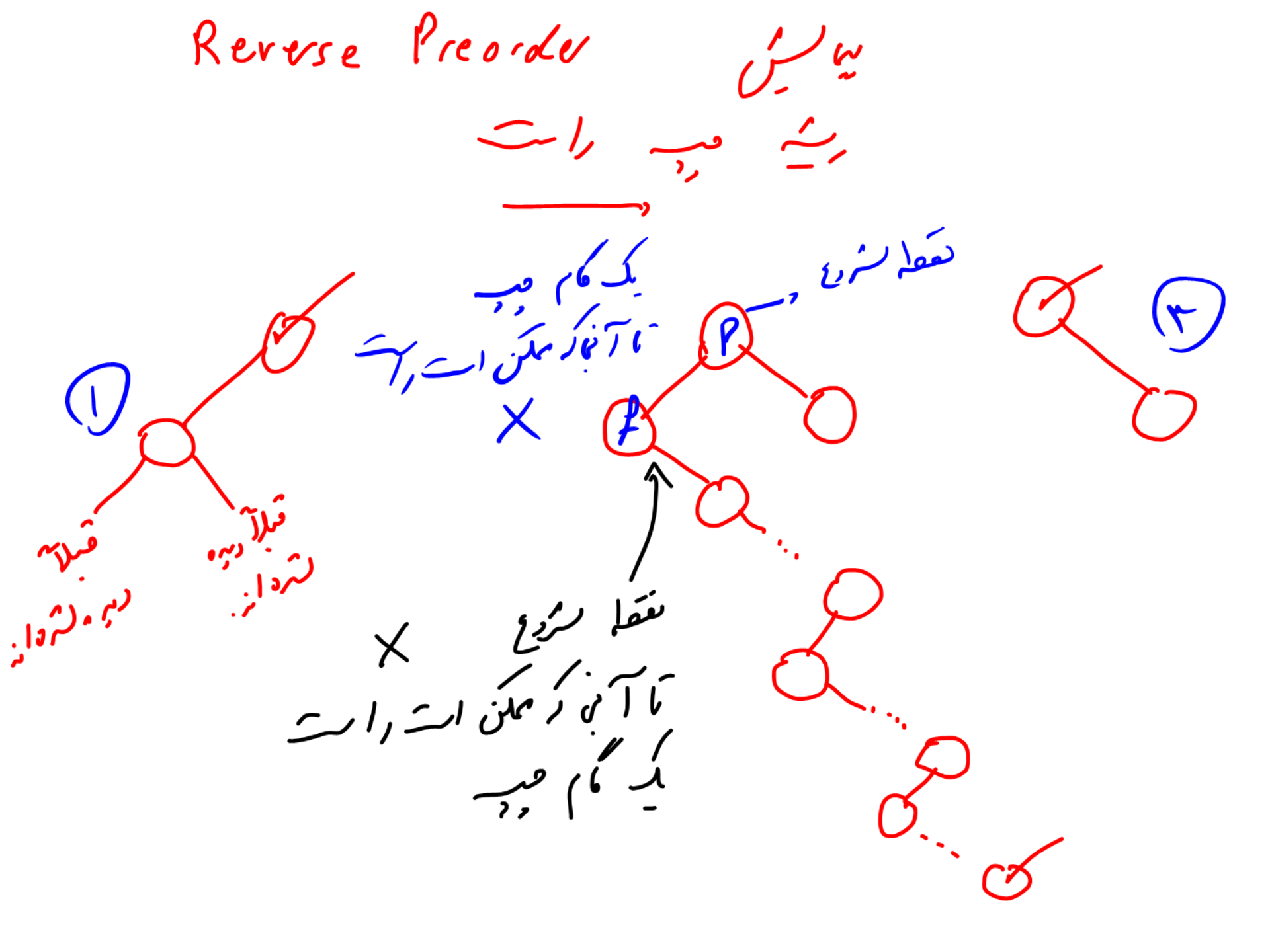


اگر فرزند چپ نداشته باشد

اگر فرزند چپ نداشته باشد و فرزند راست هم نداشته باشد

پیمایش Preorder پیش رو

- اگر گره جاری فرزند چپ دارد، گره بعدی همان فرزند چپ است.
- در غیر این صورت اگر گره جاری فرزند راست دارد، گره بعدی همان فرزند راست است.
- اگر گره جاری نه فرزند چپ و نه راست دارد، باید در درخت به سمت ریشه بالا برویم.
  - اگر به گرهی مانند p رسیدیم که در زیردرخت چپ آن هستیم و آن گره زیر درخت راست دارد، گره بعدی فرزند راست p است.
  - اگر به گرهی رسیدیم که در زیردرخت راست آن هستیم یعنی هم خود آن گره و هم زیر درخت چپ قبلا دیده شده اند. پس باید باز هم بالا رویم.
  - اگر به گرهی مانند p رسیدیم که در زیردرخت چپ آن هستیم ولی آن گره زیر درخت راست ندارد باید باز هم بالا برویم.

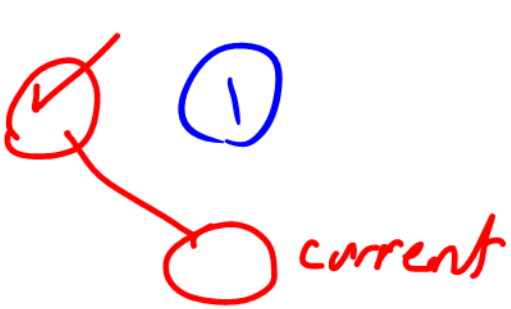

Reverse Preorder
پیمایش
ریشه چپ راست
یک گام چپ
تا آنجا که ممکن است راست
نقطه شروع
قبلاً دیده نشده است
قبلاً دیده نشده است
نقطه شروع
تا آنجا که ممکن است راست
یک گام چپ

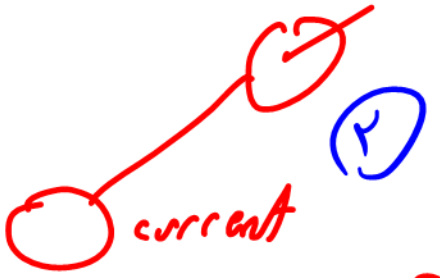
پیمایش Preorder پس رو

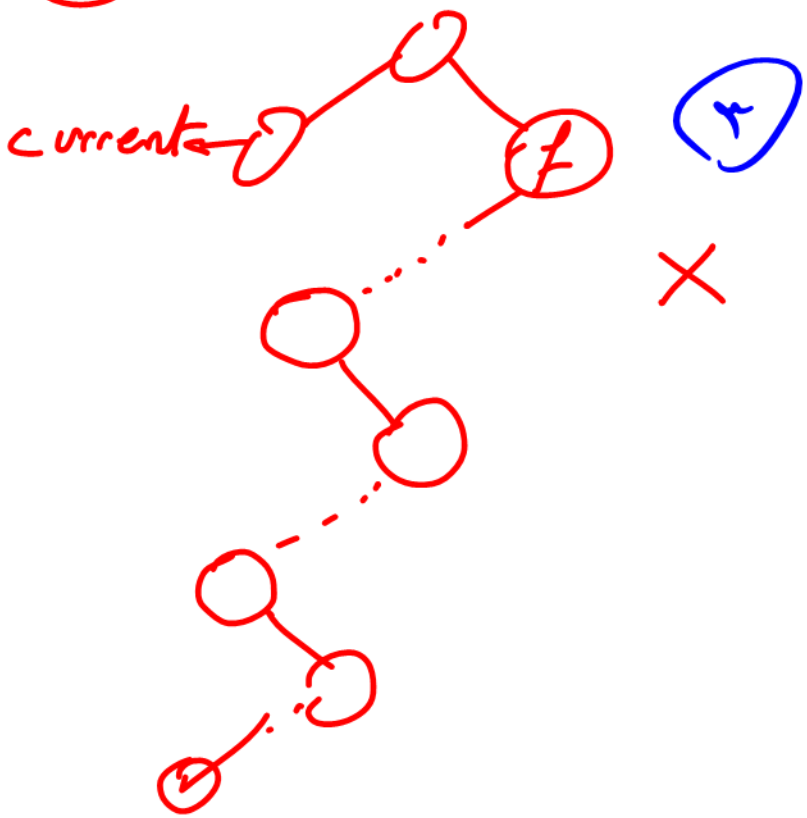
- حرکت X را به معنای به سمت راست رفتن تا حد ممکن، سپس یک حرکت به سمت چپ رفتن و مجددا تاحد ممکن به سمت راست رفتن و .... تعریف می کنیم.
- حضور ما در گره جاری نشان می دهد که زیردرخت چپ و راست دیده شده است. پس باید در درخت به سمت ریشه بالا برویم. فرض کنیم گره پدر p باشد.
- اگر فرزند چپ گره p هستیم، گره بعدی همان گره p است.
- اگر فرزند راست گره p هستیم و آن گره فرزند چپی مانند f دارد، برای رسیدن به گره بعدی باید حرکت X را بر روی f انجام دهیم.
- اگر فرزند راست گره p هستیم ولی آن گره فرزند چپ ندارد، گره بعدی همان p است.

Postorder

ریشه راست چپ





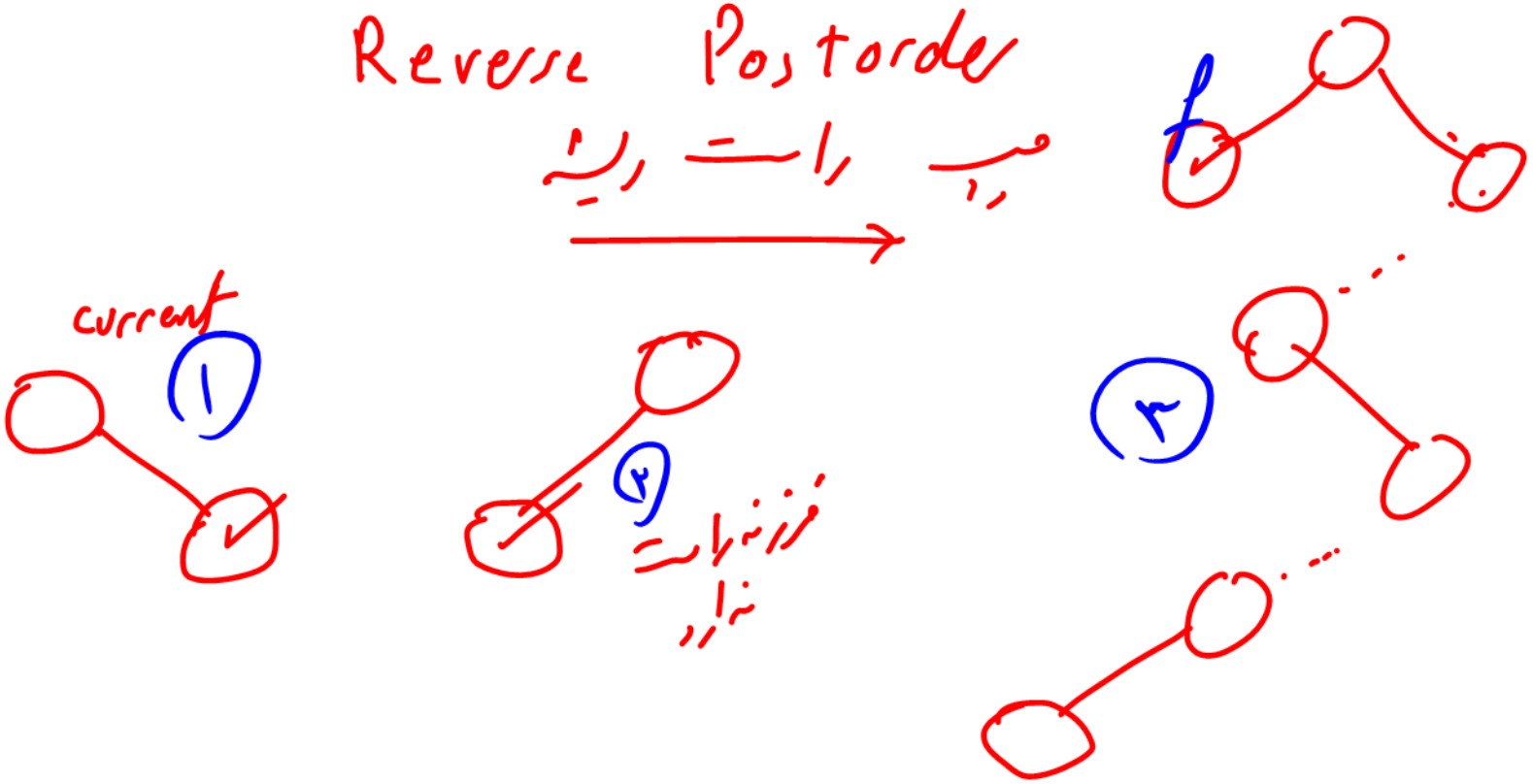


پیمایش Postorder پیش رو

- حرکت X را به معنای به سمت چپ رفتن تا حد ممکن، سپس یک حرکت به سمت راست رفتن و مجددا تاحد ممکن به سمت چپ رفتن و .... تعریف می کنیم.
- اگر فرزند چپ گره پدر هستیم و گره پدر فرزند راست ندارد یا فرزند راست گره پدر هستیم، گره بعدی همان گره پدر است.
- اگر فرزند چپ گره پدر هستیم و گره پدر دارای فرزند راست نیز هست، آنگاه گره بعدی با انجام عمل X بر روی گره راست پدر به دست می آید.

Reverse Postorder

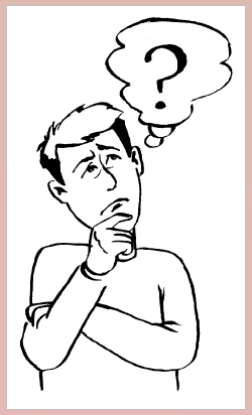
ریشه راست چپ



پیمایش Postorder پس رو

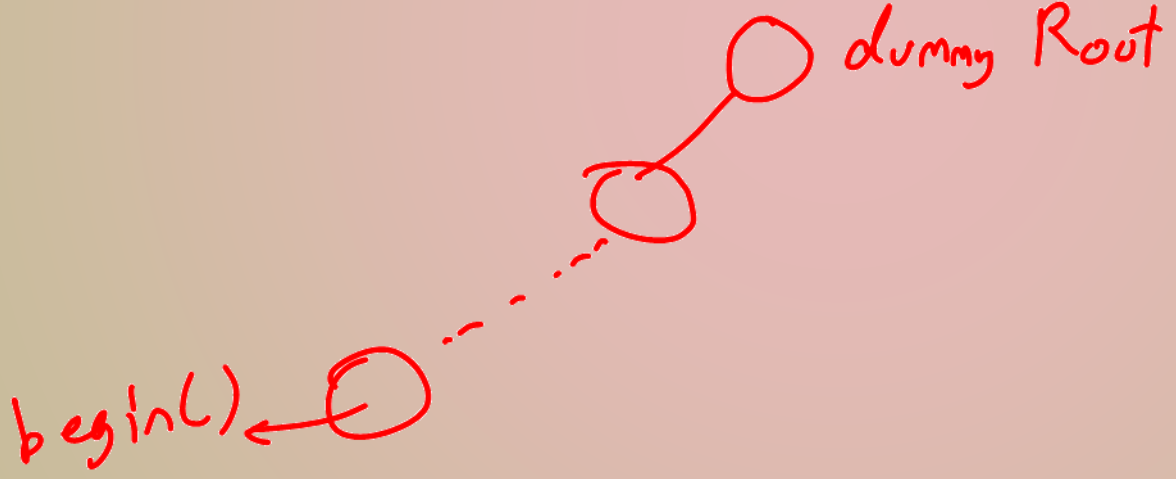
- اگر گره جاری فرزند راست دارد، گره بعدی فرزند راست است.
- اگر گره جاری فرزند راست ندارد ولی فرزند چپ دارد، گره بعدی فرزند چپ است.
- اگر گره جاری فرزند ندارد باید در درخت به سمت ریشه حرکت کنیم. تمام گره ها در مسیر ریشه قبلا دیده شده اند.
- هرگاه به گرهی رسیدیم که ما در زیردرخت راست آن بودیم و آن گره دارای فرزند چپی مانند f بود، f گره بعدی است که باید دیده شود.

چگونگی پیاده سازی begin()



مثال: InorderBegin()

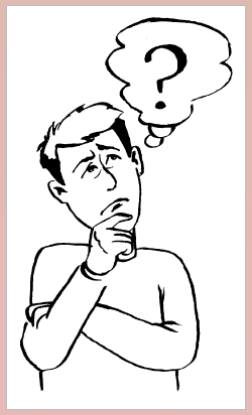
- از ریشه شروع میکنیم و تا آنجا که ممکن است به چپ میرویم. گرهی که در آن متوقف می شویم اولین گرهی است که باید در پیمایش میان ترتیب دیده شود.



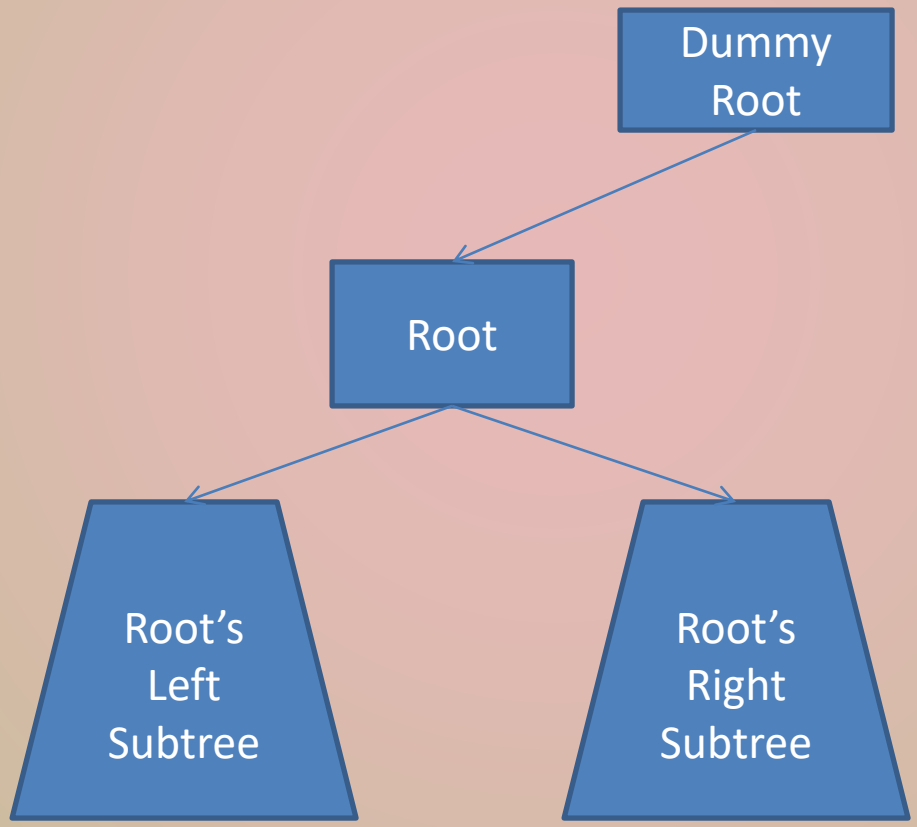
مثال: InorderReverseBegin()

- از ریشه شروع میکنیم و تا آنجا که ممکن است به راست میرویم. گرهی که در آن متوقف می شویم آخرین گرهی است که باید در پیمایش میان ترتیب دیده شود.
- بنابراین این گره اولین گرهی است که در پیمایش میان ترتیب معکوس باید دیده شود.

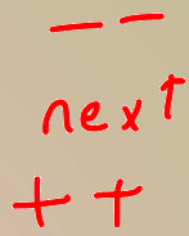
چگونگی پیاده سازی end()



چگونه پایان پیمایش را تشخیص دهیم؟



روش اول: ایده کلی



- عملیات پیمایش درخت را به نحوی بنویسیم که گویا گره DummyRoot یک گره مانند دیگر گره ها است.
- پیمایش ها به نحوی باشد که همواره آخرین گرهی که مشاهده می شود گره DummyRoot باشد.
- پس از رسیدن به گره DummyRoot الگوریتم را به نحوی بنویسیم که در حلقه نیفتد و در همان گره متوقف شود.
- پیمایشگرهای end و rend را برابر با DummyRoot قرار دهیم تا با رسیدن به این گره پیمایش تمام شود.

```
for (itr = bt.InorderBegin(); itr != bt.InorderEnd();
                                                itr++)
{

}
```

مشاهده اول: همواره پس از اتمام پیمایش گره‌های واقعی وارد گره DummyRoot می‌شویم.

- اثبات: در تمام ۶ پیمایش پیش ترتیب، میان ترتیب و پس ترتیب به صورت مستقیم و معکوس پس از مشاهده گره های درخت اصلی (فرزندان DummyRoot) از آن زیر درخت خارج می شویم (اگر چنین نباشد در حلقه تکرار می افتیم).
- با توجه به اینکه گره ها فقط از فرزندان خود و پدر خود اطلاع دارند، تنها راه خارج شده از گره های درخت مشاهده گره DummyRoot است که پدر گره ریشه است.

تضمین در حلقه نیافتادن پس از وارد شدن به DummyRoot

- هیچ یک از الگوریتم ها پس از رسیدن به DummyRoot به دنبال فرزند چپ آن نیست (زیرا الگوریتم های پیمایش درخت هیچ گاه یک گره را دو بار نمی بینند).
- فرزند راست DummyRoot را برابر Null قرار می دهیم.
- بنابراین همه الگوریتم ها به این نتیجه می رسند که:
  - یا DummyRoot باید مشاهده شود. که مطلوب ما همین است.
  - یا باید به سراغ پدر DummyRoot بروند.
- برای تضمین درحلقه نیافتادن پیمایش ها کافی است که هرگاه پدر گرهی NULL است، پیمایش متوقف شود.

استفاده از چند گره ساختگی

- مشکل راه حل قبل این است که دائما باید نگران باشیم که آیا گره پدر وجود دارد و یا اینکه NULL است.
  - هزینه این کار با چک کردن اینکه آیا به گره DummyRoot رسیده ایم یا نه قابل مقایسه است.
- راه حل: گره های ساختگی بیشتری اضافه کنیم تا هیچگاه الگوریتم های پیمایش از روال معمول خود خارج نشوند.

itr != end()

++

--

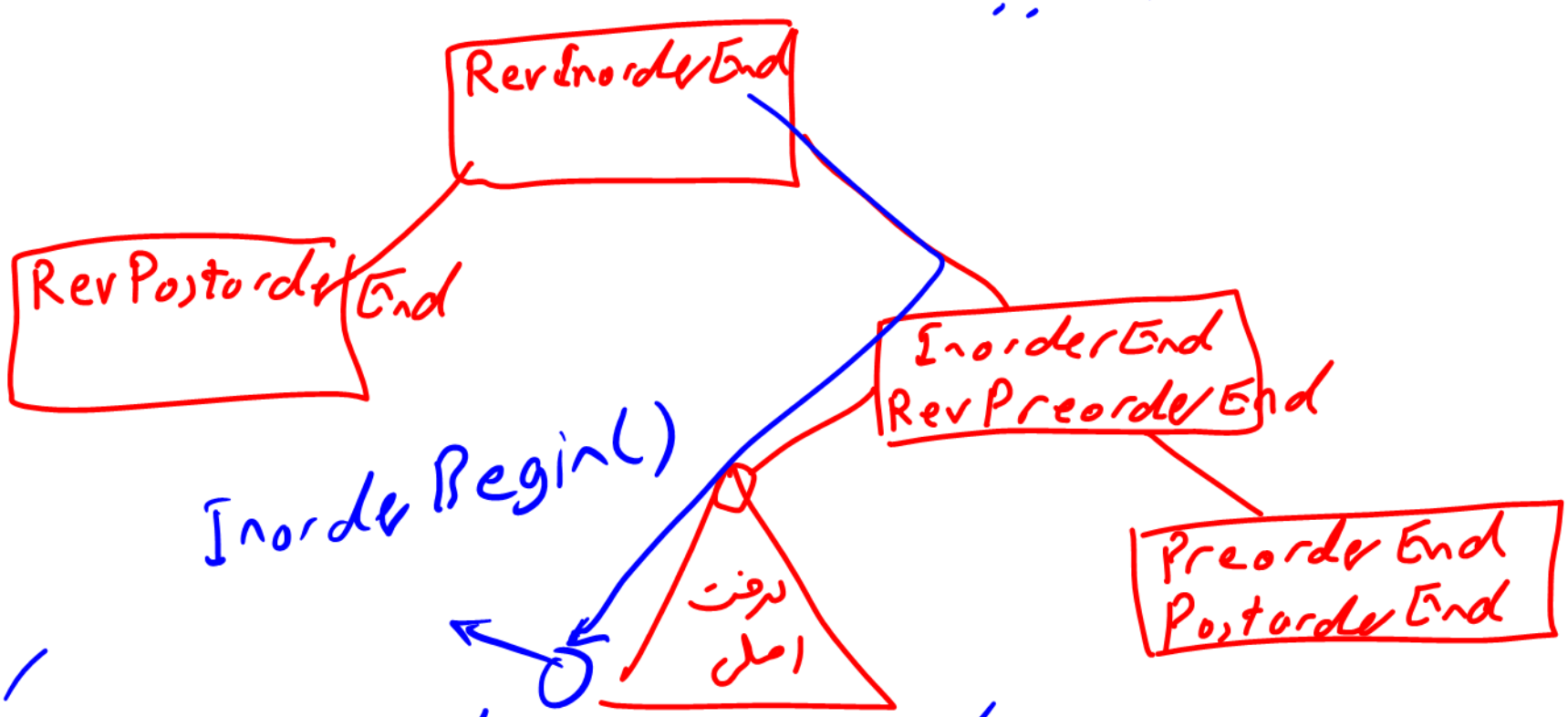
next

RevPreorder ریشه چپ راست

RevPostorder چپ راست ریشه

RevInorder چپ ریشه راست

راست ریشه چپ : ها

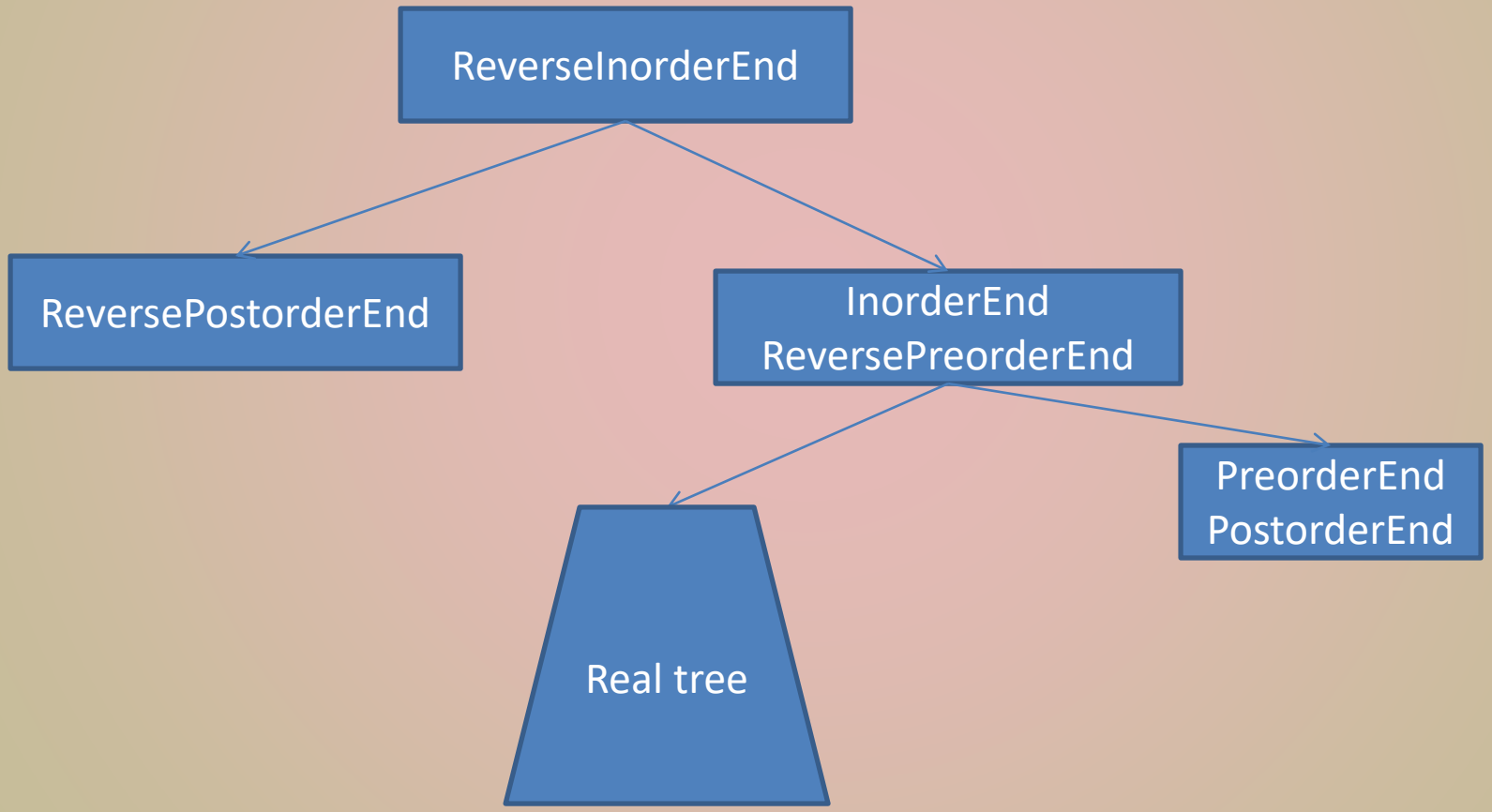


شروع کنیم Rev..End

برای پیمایش begin کار است از

و یک بار عمل ++ را انجام دهیم

ساختار کلی درخت دودویی با چند گره ساختگی



تمرین ۱

- فرض کنید ریشه واقعی به جای آنکه در سمت چپ گرهی dummy باشد، در سمت راست آن باشد.
- در این صورت گره های dummy را به نحو مناسب در بالای گره ریشه قرار دهید و همچنین محل خاتمه هر ۶ پیمایش را تعیین کنید.

تمرین ۲

- فرض کنید می خواهیم متودهای begin() و rbegin() را به نحوی پیاده سازی کنیم که به یک عنصر dummy قبل از اولین عنصر هر پیمایش اشاره کنند.
- گره های dummy را به نحو مناسب به درخت اضافه کنید و نقطه شروع هر یک از ۶ پیمایش را در بین گره های dummy تعیین کنید.