دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۷

نگارنده: فرزانه کافی موسوی

۲۹ آبان ۱۴۰۰

فهرست مطالب

۱ یادآوری

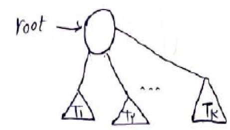
در بخش‌های قبلی دیدیم که روابط بازگشتی یک رویکرد برای حل مساله است که تسهیلاتی برای ما فراهم می‌کندمثل:

- ساده‌تر شدن الگوریتمی که می‌نویسیم
- تحلیل پیچیدگی ساده‌تر به کمک فرم‌هایی که قبلا شروع کردیم

ما می‌توانیم یک خصیه را برای یک ساختار به صورت بازگشتی هم تعریف کنیم.

*** تعریف یک درخت ریشه‌دار**

***تعریف درخت (کاملا) متعادل**



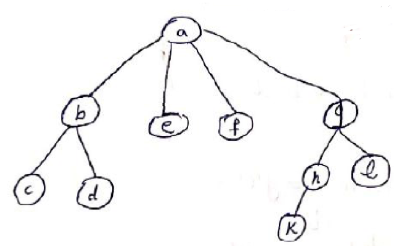
۲ پیمایش درخت‌ها (Traversal tree)

پیمایش درخت‌ها (Traversal tree) می‌توان به ازای نودهای درخت ، پیمایش‌های مختلفی در نظر گرفت. با اینحال پیمایش‌هایی که دارای خواص مطلوب برای کاربردهایی در عمل هستند در دو رده تقسیم‌بندی کرد:

- ترتیب در عمق (Defth−frist)،
- ترتیب در سطح (breadth−first)،

۱.۲ ترتیب در سطح

به این ترتیب است که نودهای یک درخت از ریشه به سمت برگ و از چپ به راست به ترتیب ملاقات می‌شوند، برای نمونه به مثال زیر توجه کنید.



پیمایش سطح $k \quad l \quad h \quad d \quad c \quad g \quad f \quad e \quad b \quad a$

$Tree - Levelorder(T)$
$1.X = t.root$
$2.while(x \neq null) \quad do$
$3. \quad visit\ mode\ X \quad \| visit(x)$
$4.Add\ clildren\ of\ X\ to\cancel{queve}\ Q \quad \| left\ to\ right$
$5.X = De\cancel{Qveve}(Q)$

deQueue(Q) to queue Q

۲.۲ ترتیب در عمق

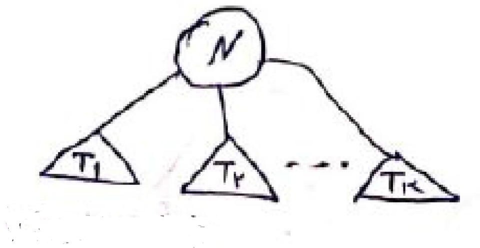
سه پیمایش مطرح در این نوع عبارتند از:

- پیش‌ترتیب(Preorder)
- میان‌ترتیب(Inorder)
- پس‌ترتیب (Postorder)

نکته: در ادامه کلیه پیمایش‌هابرای یک درخت تاییk در نظر گرفته شده است.

۳.۲ پیمایش پیش‌ترتیب

پیمایش پیش‌ترتیب: برای درخت نوعی T ، پیمایش پیش‌ترتیب بصورت زیر است:



$Preorder(T) = N\ T_1\ T_2\ ...T_k$

$Tree - Preorder(T)$

$1.if(T.root \neq null) \quad then$

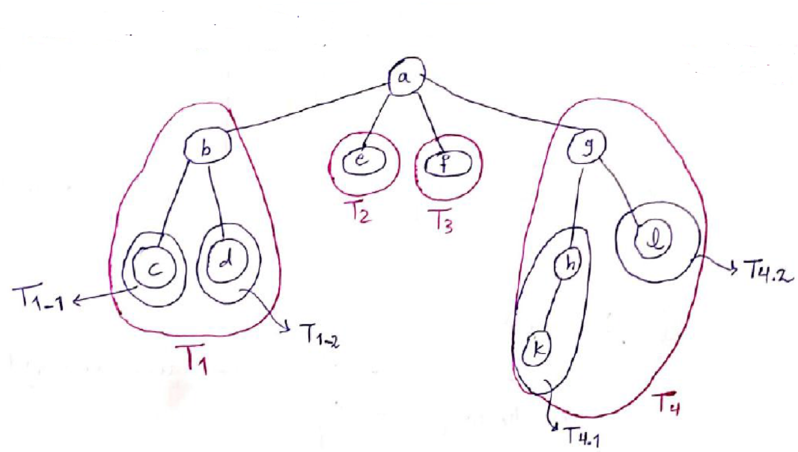
$2. \quad Visit\ node\ X = T.root \ \| Visit(X)$

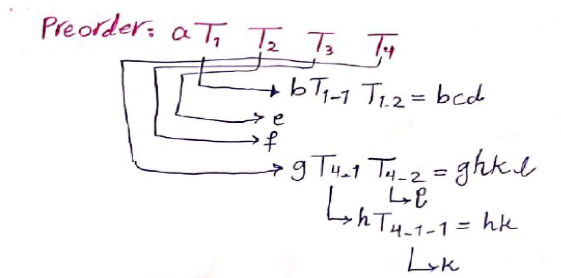
$3. \quad Tree - Preorder(T.child_1)$

$\vdots$

$4. \quad Tree - Preorder(T.child_k)$

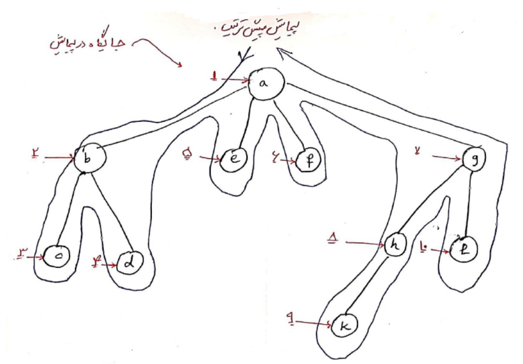
مثال پیمایش پیش‌ترتیب:





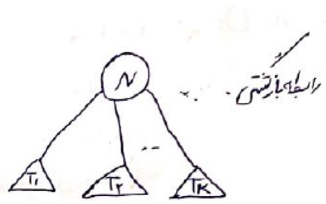
*راه‌حل ساده‌تر برای حل دستی مثال قبل پیمایش پیش‌ترتیب.
پس در نهایت داریم:

$Preorder(T) = a\ b\ c\ d\ e\ f\ g\ h\ k\ l$



۴.۲ پیمایش میان‌ترتیب

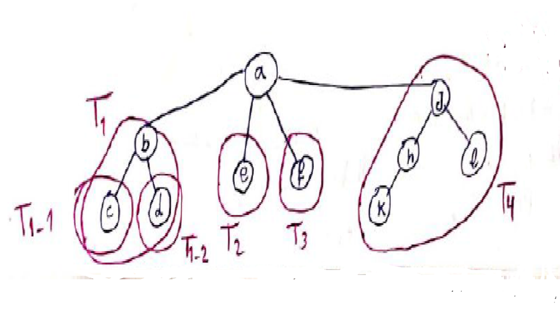
پیمایش میان‌ترتیب: برای درخت چونT ، پیمایش میان‌ترتیب بصورت زیر قابل تعریف است:

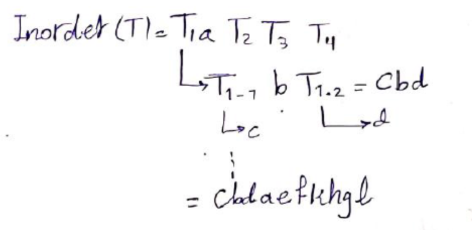


$Inorder(T) = T_1\ N\ T_2\ ...T_k$
$Tree - Inorder(T)$
$1.if(T.root \neq null) \qquad then$
$2.Tree - Inorder(T.child_1)$
$3.visit(T.root)$

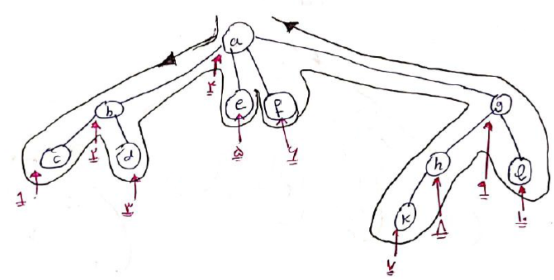
$4.Tree - Inorder(T.child_2)$
$5.Tree - Inorder(T.child_k)$

مثال پیمایش میان‌ترتیب:





راه‌حل دستی- پیمایش میان‌ترتیب:



$Inorder(T) = \cancel{d\ o\ d}\ a\ e\ f\ k\ h\ g\ l$

$\boxed{\text{Inorder(T)=cbdaefkhgl}}$

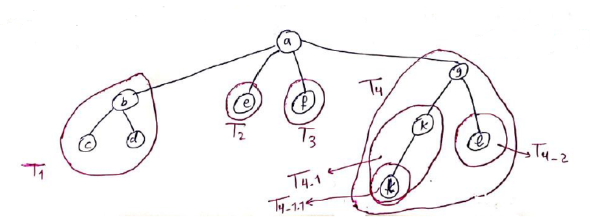
۵.۲ پیمایش پس‌ترتیب

پیمایش پس‌ترتیب: برای درخت نوعی چونT ، پیمایش پس‌ترتیب بصورت زیر قابل تعریف است:

رابطه بازگشتی ←

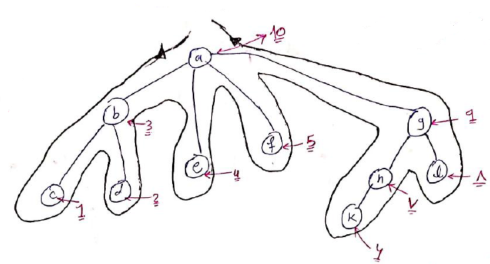
$Postorder(T) = T_1\ T_2\ ...T_k\ N$
$Tree - Postorder(T)$
$1. if (T.root \neq null) \qquad then$
$2.\ Tree - Postorder(T.child_1)$
$3.\ Tree - Postorder(T.child_2)$
$4.\ Tree - Postorder(T.child_k)$
$5.\ visit(T.root)$

مثال پیمایش پس‌ترتیب:



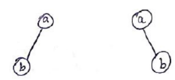
$Postorder(T) = T_1\ T_2\ T_3\ T_4\ a$
$khlg = T_{4-1}\ T_{4.2}\ g$
$kh = T_{4-1-1}\ h \qquad l$
$= cdbefkhlga$

راه‌حل دستی- پیمایش پس‌ترتیب:



برخی نکات در رابطه با پیمایش معرفی شده:
۱- از روی هر یک از پیمایش‌های معرفی شده نمی‌توان درخت یکتایی بازسازی کرد، به نمونه‌های زیر دقت کنید.

$*Preorder(T) = a\ b$



$*Inorder(T) = a\ b$

$*Postorder(T) = a\ b$

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۹: ساختمان داده و الگوریتم‌ها

نگارنده: مریم دهقان

۳ آذر ۱۴۰۰

فهرست مطالب

۱ ساخت درخت دودویی یکتا به کمک پیمایش‌های عمقی

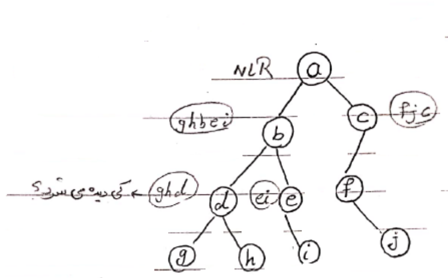
نکته تکمیلی ۱: با داشتن پیمایش‌های $Inorder$ و $Preorder$ با یک درخت دودویی می‌توان درخت یکتا را رسم کرد.
نکته تکمیلی ۲: با داشتن پیمایش‌های $Inorder$ و $Postorder$ برای یک درخت دودویی می‌توان درخت یکتا را رسم کرد.

علت داشتن درخت یکتا با پیمایش اخیر آن است که پیمایش $Inorder$ باعث می‌شود تفکیک بین گره‌های فرزند نیز لحاظ شود در حالیکه دیگر پیمایش‌ها تنها بر مکان ملاقات ریشه نسبت به سایر فرزندان تاکید داشت.

مثال ۱: برای درخت دودویی

$LNR \qquad Inorder(T) = g\ d\ h\ b\ e\ i\ a\ f\ j\ c$

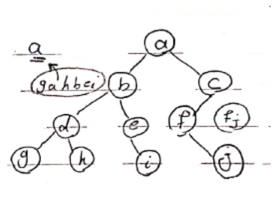
$NLR \qquad Preorder(T) = a\ b\ d\ g\ h\ e\ i\ c\ f\ j$ ریشه a است.



مثال ۲: برای درخت دودویی

$$Inorder(T) = g\ d\ h\ b\ e\ i\ a\ f\ j\ c \qquad LNR$$

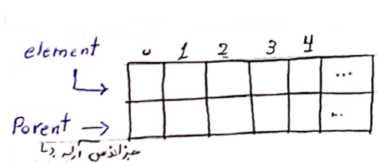
$Postorder(T) = g\ h\ d\ i\ e\ b\ j\ f\ c\ a \qquad NLR$ ریشه a است.



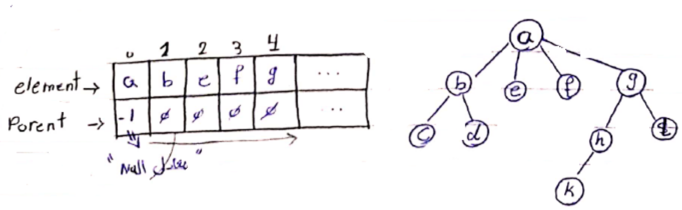
جمع‌بندی: درخت حاصل از مثال ۱ و ۲ به صورت یکسانی ترسیم شده.

۲ پیاده‌سازی درخت k تایی

راه‌حل ۱: می‌توان از یک آرایه دوبعدی به صورت زیر استفاده کرد.

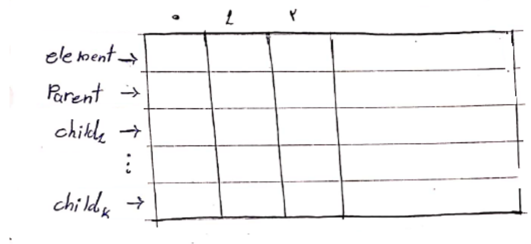


به تعداد k تا عنصر در نظر بگیریم عنصر با اندیس صفر ریشه قرارگرفته است.
درخت ما برچسب‌دار مرتب
مثال:

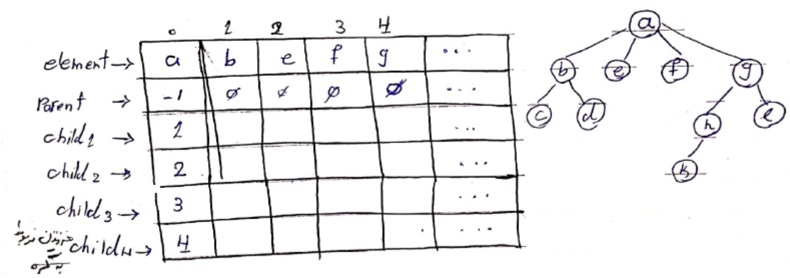


نکته: ترتیب فرزندان از اندیس کوچکتر به بزرگتر لحاظ شده است. بنابراین از روی نمایش بالا می‌توان یک درخت ترسیم کرد.

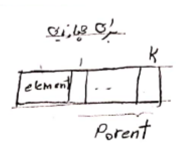
مشکلات راه‌حل ۱: ۱- دسترسی به فرزندان به سادگی امکان‌پذیر نیست و علنا باید عنصرهای آرایه را پیدایش کنیم.
۲- در صورت حذف و اضافه عنصرها در آرایه، ترتیب زیر درخت‌ها باید مدیریت شود که کار ساده‌ای نیست.
مزیت راه‌حل ۱: به ازای تعداد گره‌های درخت فضا گرفته می‌شود و هم درختی از حافظه نداریم.
راه‌حل ۲: در نظر گرفتن k=۲ آرایه برای یک درخت :تاییk



مثال:

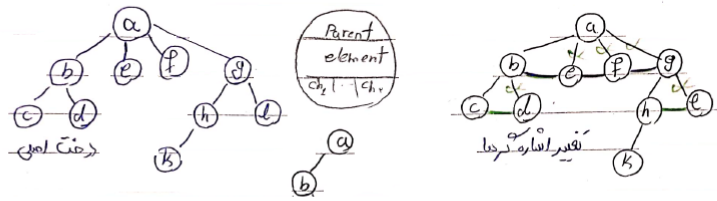


مزایای راه‌حل ۲: ۱- نگرانی در رابطه با ترتیب فرزندان نیست و در نتیجه مدیریت این ترتیب در مقایسه با راه‌حل ۱ به صورت خودکار انجام می‌شود. ۲- دسترسی به فرزندان هر گره به سادگی و در زمان O(۱) قابل انجام است.
مشکل راه‌حل ۲: پتانسیل هدر رفت حافظه در صورتی که نرده‌های یک درخت تاییk کمتر از k باشد بالا است. هدر رفت زیادی دارد.
نکته تکمیلی: پیاده‌سازی‌های فوق می‌تواند با استفاده از لیست پیوندی انجام شود که برتری آن نسبت به آرایه آن است که محدودیت هول آرایه را در صورت رشد درخت نداریم.



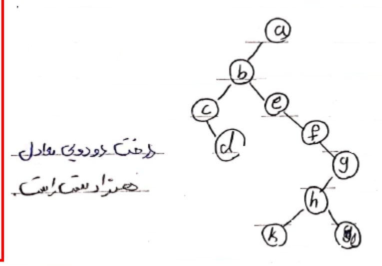
۳ تبدیل یک درخت k تایی به درخت دودویی معادل

راهکار استفاده از k=۲ آرایه برای یک درخت k تایی. همان‌طور که دیدیم پتانسیل اتلاف حافظه دارد. با این‌حال این اتلاف برای درخت دودویی به مراتب کمتر است یک راهکار برای جلوگیری ااز این اتلاف حافظه در درخت k تایی تبدیل آن به درخت دودویی معادل است.
در این تبدیل برای هر گره، اشاره‌گرهای پدر و چپ‌ترین فرزند ثابت باقی می‌ماند و اشاره‌گر سمت راست گره به نزدیکترین همزاد سمت راست (در صورت وجود) اشاره کرد.



میگه $Parent$و$element$را نگه‌دار. یکی تغییرهایی داخل $child$ ها بده $child_2$ را نگه‌دار اما بقیه k−۱ فرزند را یه جوارایی فرزند سمت راست خودشان کن. لینکه رو قطع کنرو اینایی که پدر مشترک دارند با هم وصل میشن.

به بیان ساده تر
element و parent
را نگه دارید. همچنین فرزند دوم را نیز حفظ کرده و ما بقی
k-1
فرزند را به نحوی فرزند سمت راست خودشان کنید .
در این تغیرات نودهایی که پدر مشترک دارند به یکدیگر متصل میشوند.



۱.۳ رابطه بین پیمایش درخت k تایی و درخت دودویی معادل

تمرین: رابطه‌های زیر را برای درخت k تایی (T) و درخت دودویی معادل آن (T’) بررسی کنید:
۱– آیا$Preorder(T) = Preorder(T')$ است؟
۲– آیا$Inorder(T) = Inorder(T')$ است؟
پاسخ: برای حالت کلی اگر بخواهیم نشان دهیم که در پیمایش یکسان نیستند فقط کافی است که یک مثال نقض پیدا کنیم، مثلا برای حالت ۲ و ۳ بالا مثال قبل یکسان نبودند پیدایش‌ها را به وضوح نشان می‌دهد.

$$\begin{cases} Postorder(T') = d\ c\ k\ l\ h\ g\ f\ e\ b\ a \\ Postorder(T) = c\ d\ b\ e\ f\ k\ h\ l\ g\ a \end{cases} \Longrightarrow Postorder(T) \neq Postorder(T')$$

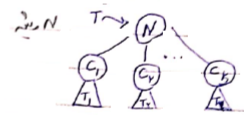
$$\begin{cases} Inorder(T') = c\ d\ b\ e\ f\ k\ h\ l\ g\ a \\ Inorder(T) = c\ d\ b\ a\ e\ f\ k\ h\ g\ l \end{cases} \Longrightarrow Inorder(T) \neq Inorder(T')$$

با این حال پیمایش پیش‌ترتیب مثال قبل برای درخت T و درخت معادل آن یعنی T’ برابر است با:

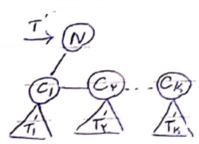
$$\begin{cases} Preorder(T') = a\ b\ c\ d\ e\ f\ g\ h\ k\ l \\ Pteorder(T) = a\ b\ c\ d\ e\ f\ y\ h\ k\ l \end{cases} \Longrightarrow Preorder(T) = Preorder(T')$$

آیا می‌توانیم ادعا کنیم برای هر درخت برقرار است؟ خیر
برای اثبات حالت کلی برای پیمایش پیش‌ترتیب می‌بایست از تعریف‌های بازگشتی به صورت زیر بهره گرفت.
پیش‌ترتیب آنگاه شهود اثبات

$$Preorder(T) = N\ C_1\ T_1\ C_2\ T_2\ ...\ C_k\ T_k \tag{۱}$$

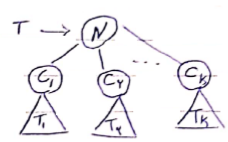


$$Preorder(T') = N\ C_1\ T'_1\ C_2\ T'_2\ ...\ C_k\ T'_k \tag{۲}$$

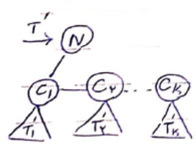


واضح است که جایگاه نودهای ملاقات شده در پیمایش پیش‌ترتیب T و T' یکسان است. می‌توان از ایده بالا برای رد یکسان بودن پیدایش میان‌ترتیب و پس‌ترتیب درخت T و T' نیز استفاده کرد.

$$Postorder(T) = T_1\ C_1\ T_2\ C_2\ ...\ T_k\ C_k\ N \tag{۳}$$

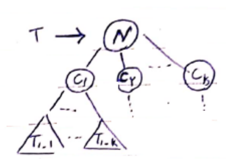


$$Postorder(T') = T'_1\ T'_2\ T'_3\ ...\ T'_k\ C_k\ ...\ C_2\ C_1\ N \tag{۴}$$

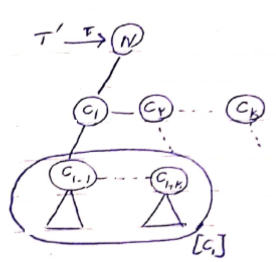


$$Inorder(T) = T_{l-1}\ C_1\ T_{l-2}\ ...\ T_{l-k}\ N \tag{۵}$$

$$T_{2-1}\ C_2\ T_{2-2}\ ...\ T_{2-k} ...\ T_{k-1}\ C_k\ T_{k-2} ...\ T_{k-k} \tag{۶}$$



$$Inorder(T') = [C_1]\ C_1\ [C_2]\ C_2\ ...\ [C_k]\ C_k\ N \tag{۷}$$



دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۳۳ : درخت دودویی عبارت

نگارنده: امین رواقی

۲۵ آذر ۱۴۰۰

فهرست مطالب

۱ درخت دودویی عبارت

درخت دودویی عبارت (Binary Expression Tree)، یک درخت دودویی است که برای نمایش عبارات ریاضی از آنها استفاده میشود.

درخت دودویی: بر نمایش عبارات ریاضی حاوی عملگرهای دو عملوندی و تک عملوندی.
عبارات ریاضی: دو نوع پر استفاده این عبارت ریاضی عبارتند از:

- عبارات جبری،
- عبارات بولی.

عبارات ریاضی در حالت کلی میتوانند در چهار قالب زیر بیان شوند:

۱. **عبارات میانوندی:** به طور معمول ما با این نوع عبارات ریاضی سر و کار داریم

۲. **عبارات پسوندی:** عملگر بعد از عملوند(ها) می آید،

۳. **عبارات پیشوندی:** عملگر قبل از عملوند(ها) می آید.

۴. **درخت عبارت:** که عملوندها نودهای خارجی این درخت و عملگرها نودهای داخلی این درخت را تشکیل می‌دهند.

در ادامه ، هر یک از نوع های مذکور را شرح می‌دهیم.

۱.۱ عبارت ریاضی میانوندی

عبارت زیر را در نظر بگیرید:

$$A^* = a + (b*c)/d - e.$$

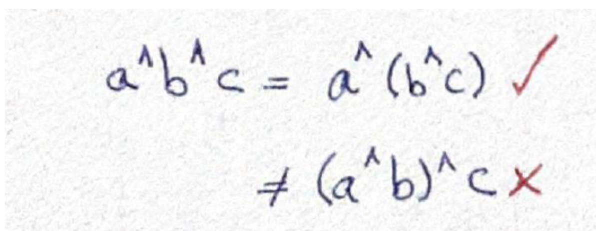
برای محاسبه بدون ابهام چنین عبارتی میانوندی دو رویکرد زیر وجود دارد :

۱. **پرانتز گذاری کامل عبارت:** در این رویکرد، با توجه به معنا (Semantic) مورد انتظار از عبارت، آن را پرانتز گذاری کامل میکنیم، مثل :

$$A^* = (((a + (b*c))/d) - e)$$

۲. **ایجاد یک قرار داد برای اولیت گذاری عملگر ها:** به عنوان مثال، می‌توان اولویت گذاری عملگرها را به ترتیب (از زیاد به کم) به صورت زیر تعیین کرد : ۱- توان از راست به چپ، ۲- ضرب و تقسیم از چپ به راست، ۳- جمع و تفریق از چپ به راست.

مثال ۱:



مثال ۲: برای عبارت

$$A^* = a + (b*c)/d - e$$

طبق اولویت عملگرها عبارت میانوندی کامل زیر را داریم:

$$A^* = ((a + ((b*c)/d)) - e)$$

نکته تکمیلی: به طور معمول در زبان‌های برنامه نویسی، وجود ابهام در محاسبه یک عبارت، احتمال وجود خطای منطقی در برنامه را بالا میبرد.

یاد آوری: انواع خطاها در برنامه: ۱-خطای نحوی، ۲-خطای منطقی، ۳- خطای زمان اجرا.
تعریف بازگشتی عبارت میانوندی کامل: یک عبارت ریاضی میانوندی کامل را می‌توان با استفاده از یک گرامر مستقل از متن (Context Free Grammer) به صورت زیر تعریف کرد:

1 E —> Operand

2 E —> (E α E)

3 α —> /|*|+|-|...

4 E —> (βE)

5 β —> Sin|cos|...

مثال: مراحل ساخت عبارت میانوندی کامل زیر را به کمک روابط بازگشتی فوق مشخص کنید.

$$A^* = ((a + ((b * c)/d)) - e)$$

EαE

$$E - - - > \boxed{(EE)} - - - > (E - E) - - - > (E - e) - - - > ((EE) - e) - - - > ((aE) - e)$$
$$- - - > ((aE) - e) - - - > ...((a + (b * c)/d)) - e)$$

نکته تکمیلی: تعریف بازگشتی فوق (گرامر مستقل از متن بالا) ضمن رعایت پرانتزگذاری کامل که سبب عدم ابهام در روال محاسبه می‌شود، این امکان را نیز فراهم میکند که اگر عبارت ریاضی نا معتبری داده شده باشد ، آن را یافته و از خطای نحوی جلوگیری کند (به عبارت دیگر همزمان خطای نحوی و خطای منطقی را پوشش می‌دهد).

۲.۱ تعریف بازگشتی عبارت پسوندی

یک عبارت ریاضی پسوندی را میتوان با استفاده از یک گرامر مستقل از متن به صورت زیر تعریف کرد:

1. E —> Operand
2. E —> EEα
3. α —> /|*|+|-|...
4. E —> Eβ
5. β —> Sin|Cos|...

مثال: عبارت میانوندی کامل زیر را به صورت عبارت پسوندی بنویسید.

$$A^* = ((a + ((b * c)/d)) - e)$$

$$bc*$$
$$bc * d/d$$
$$abc * d/+$$

در نهایت عبارت پسوندی زیر حاصل می‌شود:

$$abc * d/ + e-$$

عبارت پسوندی فوق را میتوان برای تشخیص معتبر بودن به کمک گرامر فوق پویش کنیم: EEα

$$E - - - > \boxed{EE} - - - > (EE-) - - - > (Ee-) - - - > (EEe-) - - - > ...$$

۳.۱ تعریف بازگشتی عبارت پیشوندی

یک عبارت ریاضی پیشوندی را میتوان با استفاده از یک گرامر مستقل از متن به صورت زیر تعریف کرد :

1. E —> Oprrand
2. E —> αEE
3. α —> /|*|+|-|...
4. E —> βE
5. β —> Sin|Cos|....

مثال : عبارت میانوندی کامل زیر را به صورت عبارت پیشوندی بنویسید.

$$A* = ((a + ((b * c)/d)) - e)$$

$$*bc$$

$$*bcd$$

$$+a/*bcd$$

در نهایت عبارت پیشوندی زیر حاصل می‌شود:

$$- + a/*bcde$$

عبارت پیشوندس فوق را میتوان برای تشخیص معتبر بودن به کمک گرامر فوق پویش کنیم:

αEE

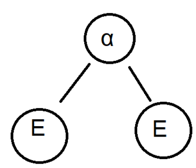
$$E ---> \boxed{EE} ---> (-EE) ---> (-EEe) ---> ... ---> -+a/*bcde$$

نکته مهم: در عبارت پسوندی و پیشوندی پرانتز گذاری نداریم و در عین حال عبارت بدون ابهامی هم برای محاسبه در اختیار داریم . با این وجود ، برای عبارت میانوندی دیدیم که اگر عبارت با پرانتز گذاری کامل نداشته باشیم ، پتانسیل ابهام در آن وجود دارد.

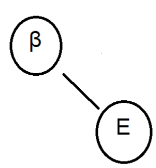
۴.۱ درخت عبارت

هر عبارت میانوندی کامل را میتوان با یک درخت دودویی نمایش داد. از آنجاییکه حالت مصور، کمک شایانی در فهم مطالب می‌کند، پرداختن به آن حایز اهمیت است.
مراحل ساخت بازگشتی یک درخت دودویی از عبارت میانوندی کامل به صورت زیر است :

۱. اگر EαE باشد داریم:

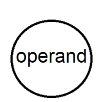


۲. اگر βE باشد ، آنگاه داریم:

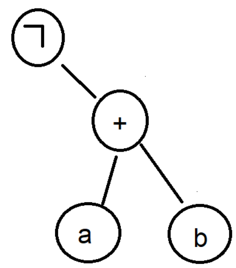


نکته: به صورت پیشفرض E را در سمت راست عملگر تک عملوندی ? قرار می‌دهیم.

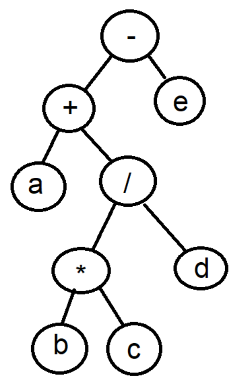
۳. اگر Operand باشد ، آنگاه داریم:



مثال ۱: درخت دودویی عبارت مربوط به عبارت ریاضی $A^* = \neg(a+b)$ را ترسیم کنید.



مثال ۲: درخت دودویی عبارت مربوط به عبارت ریاضی $A* = ((a + ((b*c)/d)) - e)$ را ترسیم کنید.



نکته مهم: پیمایش پیشوندی، پسوندی و میانوندی درخت دودویی عبارت به ترتیب معادل عبارت پیشوندی، عبارت پسوندی و عبارت میانوندی می‌باشد.

تمرین: تبدیل عبارت میانوندی کامل، پیشوندی، پسوندی و درخت دودویی عبارت به یکدیگر به چه نحو قابل انجام است؟

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰–۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۴۰

نگارنده: فرشته قادری

۱۴ دی ۱۴۰۰

فهرست مطالب

۱ داده ساختار جدول درهم ساز

یاد آوری: هدف از معرفی و بررسی داده ساختارها، ذخیره و بازیابی مجموعه‌های پویا به صورت کارا می باشد. در این راستا داده ساختارهای زیادی معرفی شده‌اند و بر روی آنها عملیات مختلفی متناسب با کاربرد مورد نیاز تعریف شده‌اند. در حالت کلی داده ساختارهایی که تا به حال فرا گرفتیم در دو رده کلی زیر قابل تقسیم بندی است:

– **داده ساختارهای مقدماتی، نظیر**: لیست، لیست پیوندی صف، صف حلقوی و پشته.

– **داده ساختار های پیشرفته نظیر**: درخت trie، درخت دودویی عبارت BET، درخت دودویی جست و جو و درخت هرم بیشینه.

سوال: پیچیدگی زمانی عملیات پایه روی داده ساختارهای فوق نظیر: عملیات درج، حذف و جست و جو را قبلا بررسی کردیم، در اینجا سوال آن است که آیا می‌توان کلیه این عملیات پایه را در $O(1)$ انجام داد، یا سعی کرد تا جایی که ممکن است، حداقل مرتبه پیچیدگی زمانی را برای

این عملیات داشت؟

در ادامه به بررسی داده ساختار جدول درهم‌ساز برای پاسخگویی به این سوال می‌پردازیم.

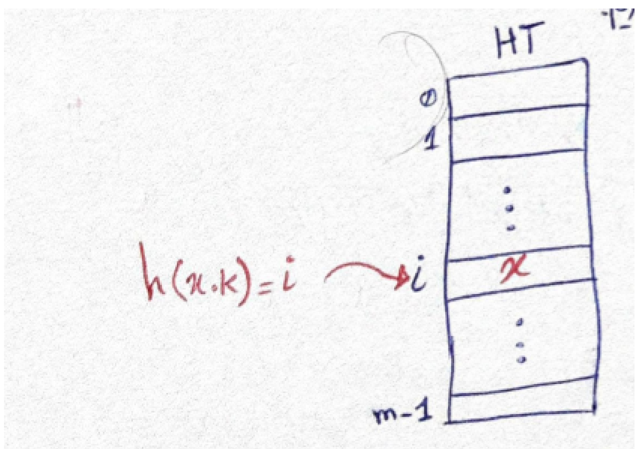
تعریف داده ساختار جدول درهم‌ساز. فرض کنید:

- m خانه از حافظه را در اختیار داریم که از صفر تا m−۱ اندیس گذاری شده است.

- n کلید که مقادیر آن از صفر تا n−۱ است نیز در اختیار داریم
- همچنین یک تابع در هم ساز h که با توصیف زیر نیز داده شده است:

$$h : [0, ..., n-1] \rightarrow [0, ..., m-1]$$

یک جدول درهم‌ساز HT با m خانه حافظه، داده ساختاری است که هر کلید k عوض مجموعه $\{0, \cdots, n-1\}$ را با استفاده از تابع درهم‌ساز h به طور مناسبی به یکی از m خانه حافظه نگاشت کرده و عنصر حاوی کلید k را در آن خانه از حافظه درج کند. شکل گرافیکی یک جدول درهم‌ساز در ادامه آورده شده است. با فرض اینکه x یک عنصر است و h(x،k)=i داریم:

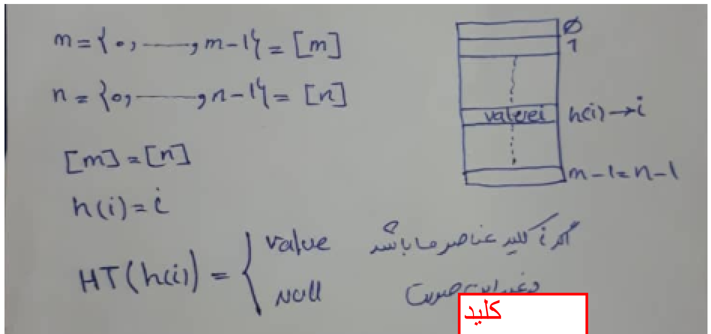


در ادامه سعی داریم به چالش های اصلی داده ساختار جدول درهم ساز، یعنی:

- تعیین خانه های حافظه،
- تعیین یک تابع درهم‌ساز خوب بپردازیم.

۲ دسترسی مستقیم یا آدرس دهی مستقیم

در این بخش به بررسی یک حالت ایده‌آل از نظر کارایی برای جداول درهم‌ساز می‌پردازیم. در این روش،با فرض عدم محدودیت روی حافظه می‌توان تنظیمات زیر را برای جدول درهم‌ساز درنظر گرفت:



کلید
k
به صورت

در این روش، عملیات درج، حذف و جستجو به صورت قابل اعمال است:

- درج مقدار value برای عنصر با کلید k به صورت $HT[h(k)] = HT[k] = value$ انجام می‌شود.
- حذف عنصر با کلید k به صورت $HT[h(k)] = HT[k] = Null$ انجام می‌شود.
- جستجو یک عنصر با کلید k به صورت زیر قابل انجام است:
  - اگر $HT[h(k)] = HT[k] = Null$ باشد، آنگاه عنصر با کلید k در مجموعه پویای خود نداریم.
  - اگر $HT[h(k)] = HT[k] \neq Null$ باشد، عنصر با کلید k موجود است و می‌توانیم آن عنصر را بازیابی کنیم.

نکته: روش آدرس دهی مستقیم،یک راه حل نظری و نه علمی است چرا که فضای مصرفی زیادی را می‌طلبد. با این حال پیچیدگی زمانی عملیات پایه: درج، حذف و جستجو در آن از مرتبه $O(1)$ است.

۳ حرکت به سوی جداول درهم‌ساز عملی (با قید محدودیت حافظه)

در ادامه به دنبال راه حلی هستیم که با اعمال محدودیت حافظه بتوانیم داده ساختار جدول درهم ساز تا حد امکان کارایی را برای عملیات پایه اریه دهیم. به عبارت دیگر میخواهیم به سراغ تنظیمات برویم که m به مراتب کوچک‌تر از n باشد.