* با داشتن پیمایش های Preorder , Inorder می توان برای یک درخت دودویی درخت یکتا را ترسیم کرد.

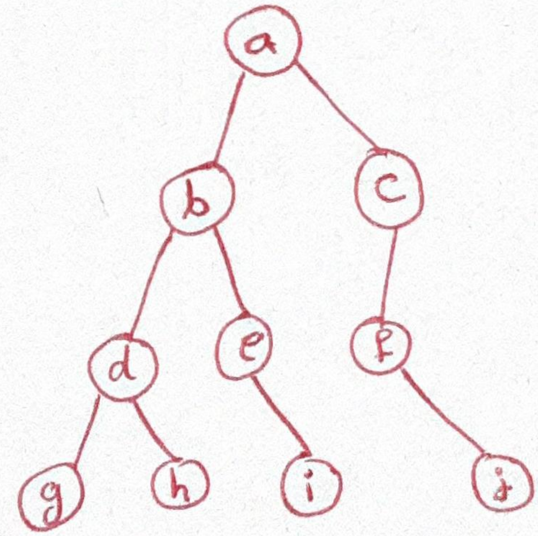
* با داشتن پیمایش های Postorder, Inorder می توان برای یک درخت دودویی درخت یکتا را ترسیم کرد.

علت داشتن درخت یکتا با دو پیمایش اخیر آن است که پیمایش Inorder باعث می شود تفکیک بین گره های فرزند نیز لحاظ شود درحالیکه دیگر پیمایش ها تنها بر مکان ملاقات ریشه نسبت به سایر فرزندان تاکید داشت.

مثال ۱: برای درخت دودویی:

Inorder(T) = g d h b e i a f j c

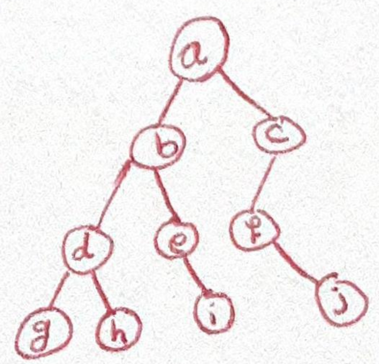
Preorder(T) = a b d g h e i c f j



مثال ۲: برای درخت دودویی:

Inorder(T) = g d h b e i a f j c

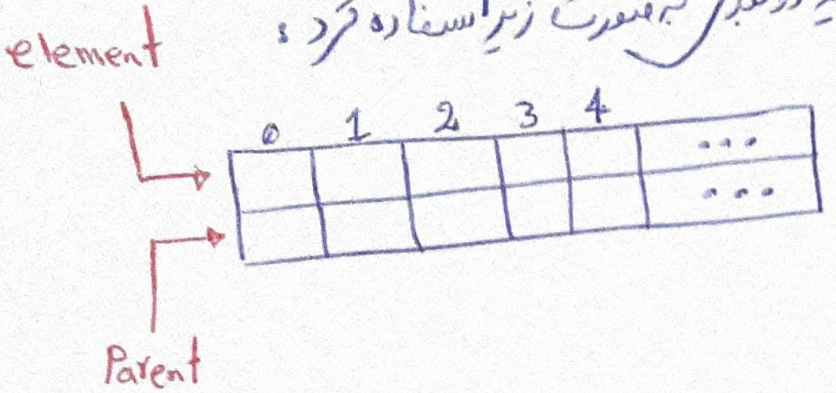
Postorder(T) = g h d i e b j f c a



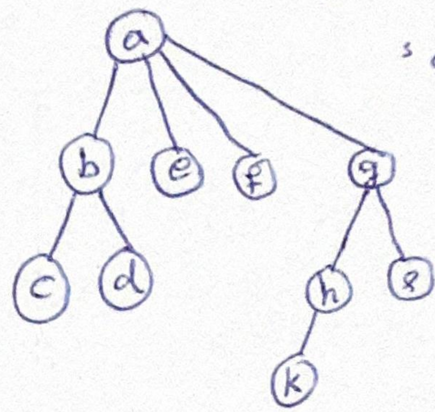
جمع بندی: درخت حاصل از مثال ۱ و ۲ به صورت یکسانی ترسیم شد.

پیاده‌سازی درخت k-تایی:

راه حل ۱: می‌توان از یک آرایه دوبعدی به صورت زیر استفاده کرد:



مثال:



	0	1	2	3	4	...
element	a	b	e	f	g	...
Parent	-1	0	0	0	0	

"معادل null"

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

مشکلات راه حل ۱: ① دسترسی به فرزندان به سادگی امکان‌پذیر نیست و عملاً باید عضوهای آرایه را پیمایش کنیم.

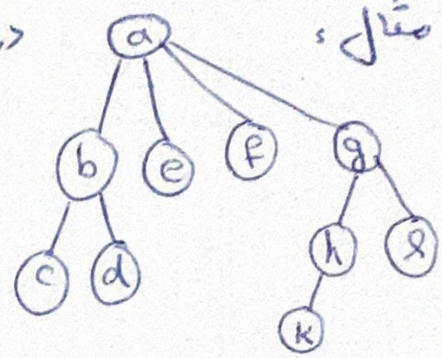
② در صورت حذف و اضافه عضوها در آرایه، ترتیب زیردرخت‌ها باید مدیریت شود که کار ساده‌ای نیست.

مزیت راه حل ۱: به ازای تعداد گره‌های درخت فضا گرفته می‌شود و هدر رفتی از حافظه نداریم.

راه حل ۲: در نظر گرفتن k+2 آرایه برای یک درخت k-تایی:

	0	1	2	…
element →				
Parent →				
$child_1$ →				
⋮				⋮
$child_k$ →				

مثال:



درخت ۴ تایی:

	0	1	2	3	4	
element →	a	b	e	f	g	…
Parent →	-1	0	0	0	0	…
$child_1$ →	1				…	…
$child_2$ →	2				…	…
$child_3$ →	3				.	…
$child_4$ →	4					

مزایای راه حل ۲: ① نگرانی در رابطه با ترتیب فرزندان نیست و در نتیجه مدیریت این ترتیب در
با راه حل ۲ به صورت خودکار انجام می‌شود.

② دسترسی به فرزندان هر گره به سادگی و در زمان $O(1)$ قابل انجام است.

مشکل راه حل ۲: پتانسیل هدر رفت حافظه در صورتی که نودهای یک درخت k-تایی کمتر از k باشد، بالا است.

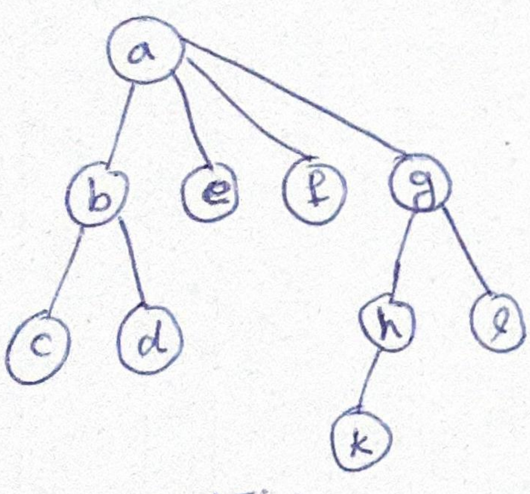
نکته تکمیلی: پیاده سازی های فوق می‌توانند با استفاده از لیست پیوندی نیز انجام شود که برتری آن نسبت به آرایه آن است که محدودیت طول آرایه را در صورت رشد درخت نداریم.

تبدیل یک درخت k-تایی به درخت دودویی معادل:

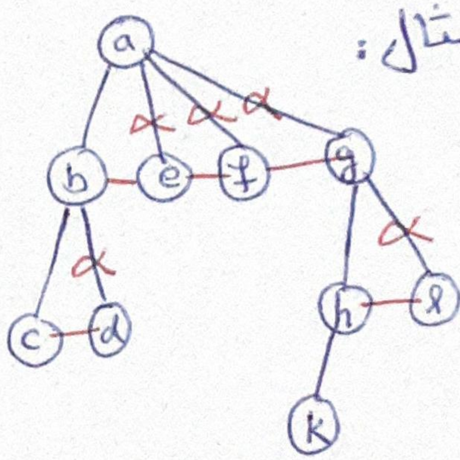
راهکار استفاده از k+2 آرایه برای یک درخت k-تایی، همانطور که دیدیم پتانسیل اتلاف حافظه دارد. با اینحال این اتلاف برای درخت دودویی به مراتب کمتر است. یک راهکار برای جلوگیری از این اتلاف حافظه در درخت k-تایی، تبدیل آن به درخت دودویی معادل است.

در این تبدیل، برای هر گره، اشاره‌گرهای پدر رو چپ‌ترین فرزند ثابت باقی می‌ماند و اشاره‌گر سمت راست گره به نزدیک‌ترین همزاد سمت راست (در صورت وجود) اشاره می‌کند.

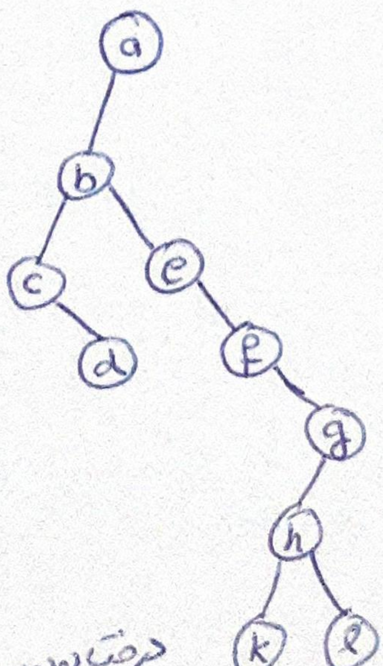
مثال:



درخت اصلی



تغییر اشاره‌گرها



درخت دودویی معادل

بررسی رابطه بین پیمایش های درخت k-تایی و درخت دودویی معادل:

تمرین: رابطه های زیر را برای درخت k-تایی (T) و درخت دودویی معادل آن (T') بررسی کنید؟

۱. آیا $Preorder(T) = Preorder(T')$ است؟

۲. آیا $Postorder(T) = Postorder(T')$ است؟

۳. آیا $Inorder(T) = Inorder(T')$ است؟

پاسخ: برای حالت کلی، اگر بخواهیم نشان دهیم که دو پیمایش یکسان نیستند فقط کافی است که یک مثال نقض پیدا کنیم، مثلاً برای حالت ۲ و ۳ بالا مثال قبل یکسان نبودن پیمایش ها را به وضوح نشان می دهد؟

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

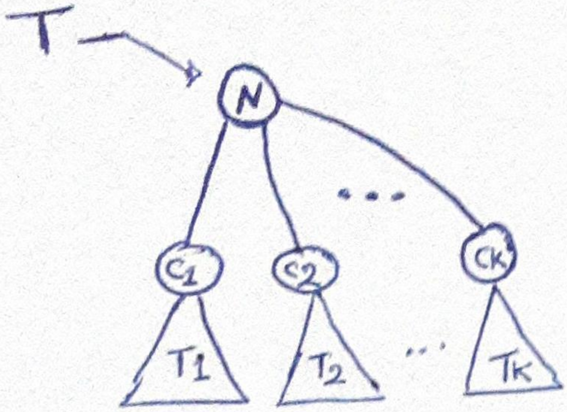
$$\begin{cases} Inorder(T') = c\,d\,b\,e\,f\,k\,h\,l\,g\,a \\ Inorder(T) = c\,b\,d\,a\,e\,f\,k\,h\,g\,l \end{cases} \rightarrow Inorder(T) \neq Inorder(T')$$

با اینحال پیمایش پیش ترتیب مثال قبل برای درخت T و درخت معادل T' آن یعنی T' برابر است:

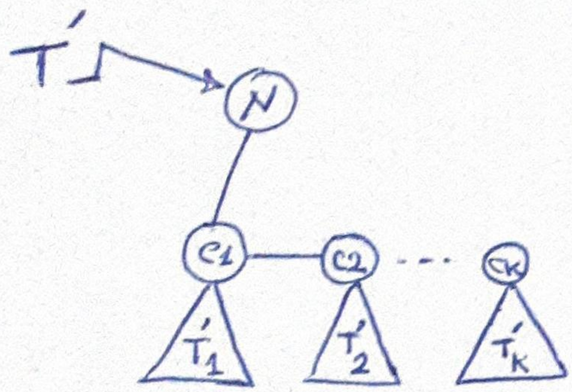
$$\begin{matrix} Preorder(T') = a\,b\,c\,d\,e\,f\,g\,h\,k\,l \\ Preorder(T) = a\,b\,c\,d\,e\,f\,g\,h\,k\,l \end{matrix} \rightarrow Preorder(T) = Preorder(T')$$

برای اثبات حالت کلی برای پیمایش پیش ترتیب می بایست از تعریف های بازگشتی به صورت زیر بهره گرفت.

شهود اثبات پیش ترتیب



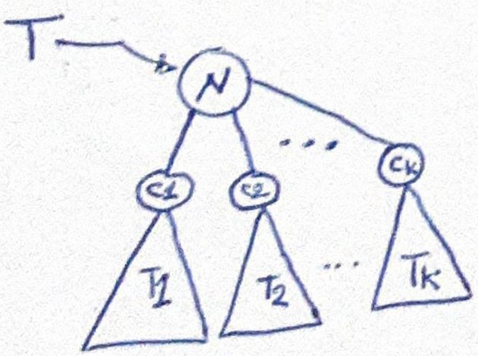
$Preorder(T) = N\, c_1 T_1\, c_2 T_2 \cdots c_k T_k$



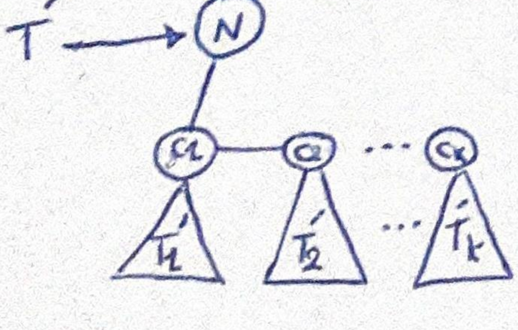
$Preorder(T') = N\, c_1 T'_1\, c_2 T'_2 \cdots c_k T'_k$

واضح است که جایگاه نودهای ملاقات شده در پیمایش پیش ترتیب T و T' یکسان است.

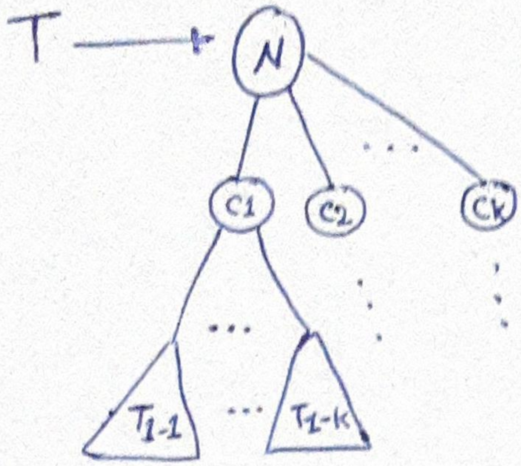
می توان از ایده بالا برای رد یکسان بودن پیمایش میان ترتیب و پس ترتیب درخت T و T' نیز استفاده کرد:



$Postorder(T) = T_1 \underline{c_1}\, T_2 \underline{c_2} \cdots T_k \underline{c_k} N$

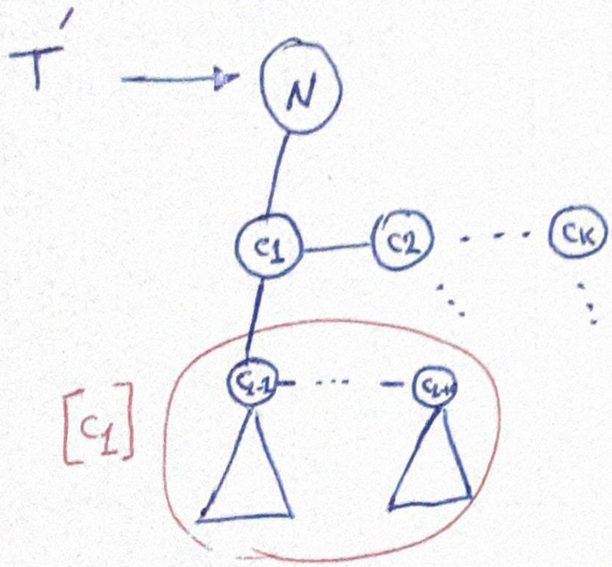


$Postorder(T') = T'_1 T'_2 \cdots T'_k \underline{c_k} \cdots \underline{c_2}\, \underline{c_1} N$



$$\text{Inorder}(T) = \underbrace{T_{1-1}\, c_1\, T_{1-2} \cdots T_{1-k}}\; N$$

$$\underbrace{T_{2-1}\, c_2\, T_{2-2} \cdots T_{2-k}} \cdots \underbrace{T_{k-1}\, c_k\, T_{k-2} \cdots T_{k-k}}$$



$$\text{Inorder}(T') = [c_1]\, c_1\, [c_2]\, c_2 \cdots$$

$$[c_k]\, c_k\, N$$