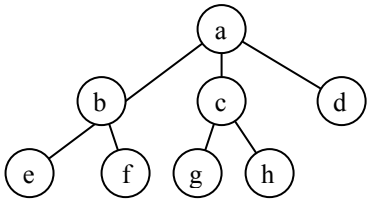
درخت (Tree)

درخت: مجموعه محدودی از n گره بطوریکه دارای گره خاصی بنام ریشه باشد و سایر گره‌ها به k مجموعه مجزای $T_1, T_2, \ldots T_k$ تقسیم شوند به گونه‌ای که هر یک از این مجموعه‌ها خود یک درخت باشد. (تعریف درخت بازگشتی است).



درجه یک گره: تعداد فرزندان یک گره (تعداد زیردرختها) درجه آن نام دارد.

برگ: گره‌های که درجه صفر دارند، برگ یا گره‌های پایانی یا گره‌های خارجی نامیده می‌شوند.

درجه درخت: حداکثر درجه گره‌های درخت را درجه درخت می‌گویند.

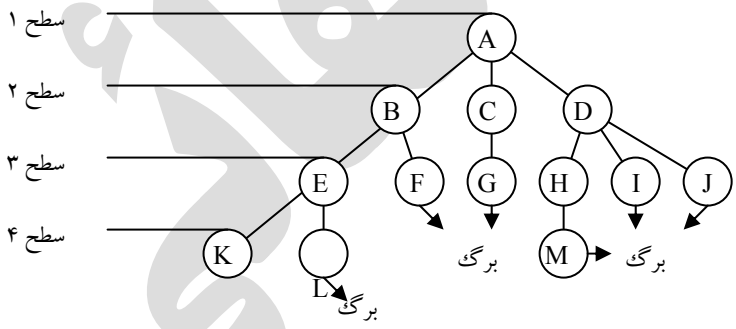
گره‌های همزاد: فرزندان یک گره، گره‌های همزاد با هم نیا نامیده می‌شوند.

اجداد یک گره: گره‌هایی هستند که در مسیر طی شده از ریشه تا آن گره وجود دارند.

سطح یک گره: فاصله یک گره تا ریشه +۱ را سطح یک گره گویند (سطح ریشه درخت ۱ می‌باشد).

ارتفاع یا عمق یا سطح درخت: به حداکثر سطح گره‌های یک درخت گویند.

ریشه: گره‌ای که خود فرزند گره دیگری نباشد.

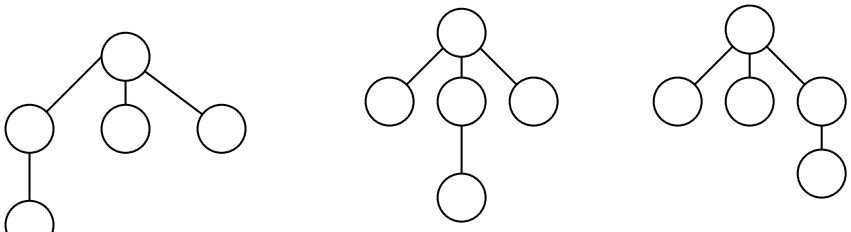


انواع درخت:

درخت عمومی- درخت k تایی

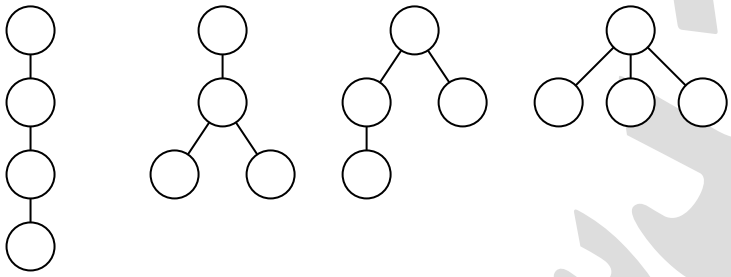
درخت عمومی:

- نمی‌تواند تهی باشد.
- مرتب نیست و ترتیب قرار گرفتن زیر درختها مهم نیست.
- در تعداد فرزندان محدودیت وجود ندارد.



این ۳ درخت با هم مساوی و مشابه می‌باشند زیرا ریشه هر سه درخت درجه ۳ بوده و فقط یک فرزند درجه یک دارد و دو فرزند دیگر برگ می‌باشند. (البته اگر آنها را به دید درخت عمومی نگاه کنیم).

مثال: چند درخت عمومی با ٤ گره می‌توان ایجاد کرد؟ ٤درخت



نکته: رابطه‌ای برای محاسبه تعداد درخت عمومی با n گره وجود ندارد.

درخت k تایی:

- می‌تواند تهی باشد.
- مرتب است و ترتیب قرار گرفتن زیر درختها مهم است.
- هر گره حداکثر k فرزند می‌تواند داشته باشد.

درخت دودویی (Binary Tree)

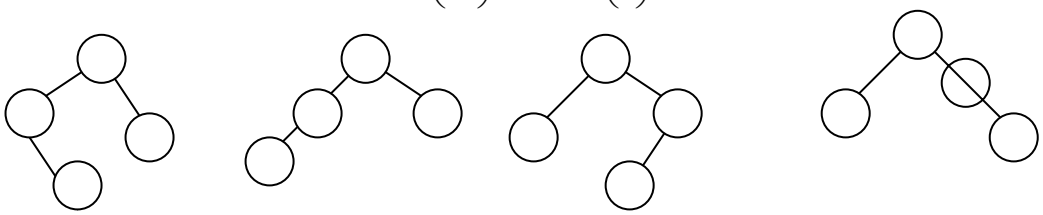
درختی که هر گره آن حداکثر دو فرزند داشته باشد. در حقیقت یک درخت k تایی می‌باشد که در آن $k = ۲$ می‌باشد. مبحث درختها معمولاً روی درختها دودویی متمرکز است.

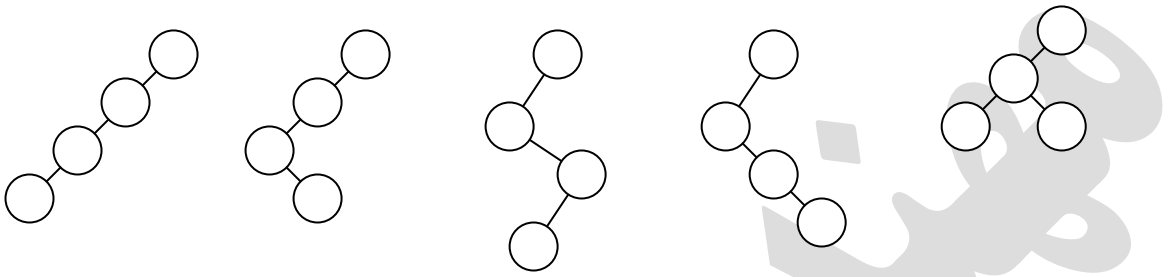
مثال: دو درخت زیر اگر عادی باشند یکسان هستند و اگر دودویی باشند، با هم تفاوت دارند.

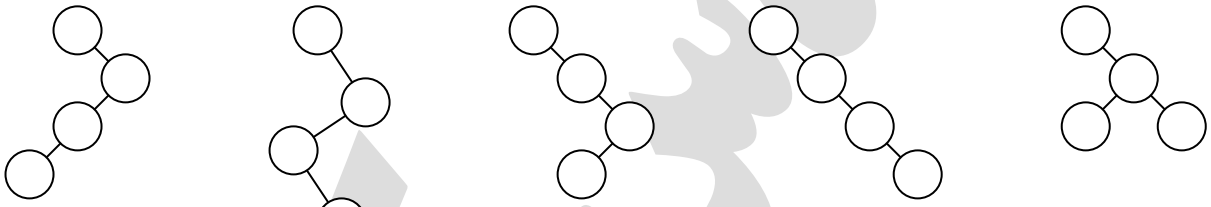
نکته: تعداد درختان دودویی که با n گره می‌توان ایجاد کرد از رابطه زیر بدست می‌آید.

$$\frac{۱}{n+۱} * \binom{۲n}{n}$$

مثال: چند درخت دودویی با ٤ گره وجود دارد؟ $\frac{۱}{n+۱}*\binom{۲n}{n}=\frac{۱}{۴+۱}*\binom{۸}{۴}=۱۴$

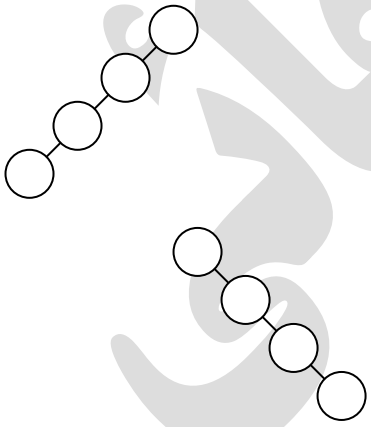






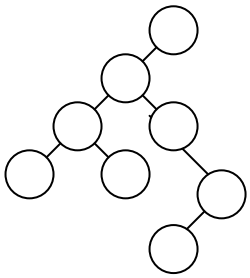
دسته‌بندی درختها:

درخت مورب چپ: هر گره فقط می‌تواند فرزند سمت چپ داشته باشد.

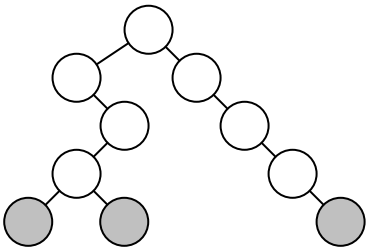


درخت مورب راست: هر گره فقط می‌تواند فرزند سمت راتس داشته باشد.

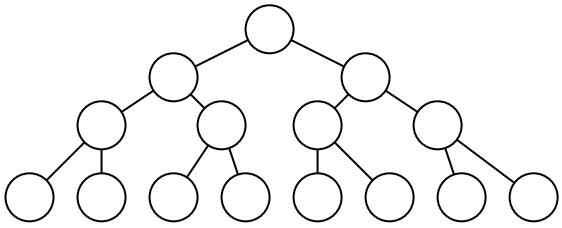
درخت متوازن: درختی که اختلاف سطح برگهای آن حداکثر ۱ باشد.



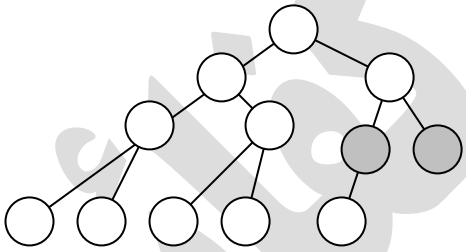
درخت کاملاً متوازن: درختی که اختلاف سطح برگهای آن صفر باشد (همه برگها در یک سطح باشند).



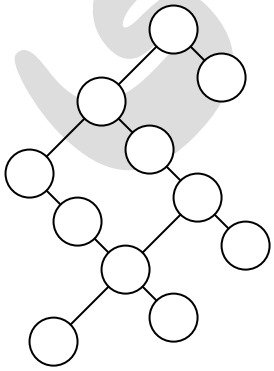
درخت پر: درختی که همه گره‌ها بغیر از گره‌های سطح آخر، حداکثر تعداد فرزندان را داشته باشد.



درخت کامل: درختی که تا سطح یکی به آخر، پر باشد و در سطح آخر نیز گره‌ها از چپ به راست قرار گرفته باشند.



درخت محض دودویی: درختی که درجه گره‌های آن یا ۲ باشد یا صفر (برگ باشد).

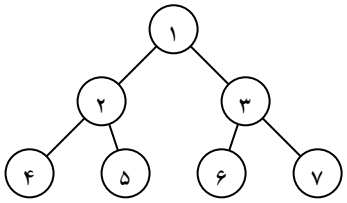


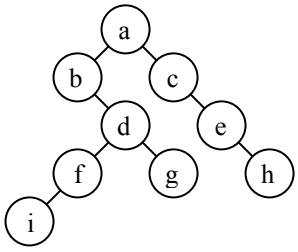
نمایش درخت دودویی:

یک درخت دودویی را می‌توان به کمک آرایه، ماتریس یا لیست پیوندی نمایش داد.

آرایه:

گره‌های درخت را بر اساس شماره‌گذاری گره‌های درخت کامل، شماره می‌زنیم و اطلاعات هر گره را در خانه‌ای از آرایه که اندیس آن با شماره آن گره مساوی است قرار می‌دهیم.





۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸	۱۹	۲۰
a	b	c		d		e			f	g				h					i

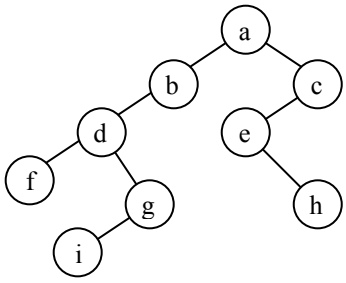
این روش برای:

- درختهای کامل و پر عالی است زیرا همه خانه‌های آرایه استفاده می‌شود و حافظ به هدر نمی‌رود.
- درختهایی که ساختارهای پر گره‌ای دارند خوب است زیرا حافظه کمی به هدر می‌رود.
- درختهای مورب بسیار بد است زیرا بیشترین خانه‌های آرایه به هدر می‌رود.

ماتریس

در این روش از یک ماتریس، ۳ ستون که ستون اول بر چسب گره، ستون دوم شماره فرزند سمت چپ و ستون سوم شماره فرزند سمت راست می‌باشد استفاده می‌شود. تعداد سطرهای ماتریس به تعداد گره‌های درخت است.

- در این روش ترتیب شماره‌گذاری بصورت سطری است.



	N	L	R
ریشه ← ۱	a	۲	۳
۲	b	٤	۰
۳	c	٥	۰
٤	d	٦	۷
٥	e	۰	۸
٦	f	۰	۰
۷	g	۹	۰
۸	h	۰	۰
۹	i	۰	۰

این روش برای درختهای مورب، خلوت یا درختهایی که در مورد ساختارشان چیزی نمی‌دانیم روش مناسبی است.
در این روش، مقدار حافظه هدر رفته به تعداد گره‌ها بستگی دارد و کاملاً از ساختار درخت مستقل است. اگر درخت n گره داشته باشد، ماتریس آن $n+1$ صفر دارد که همان فضای هدر رفته است.

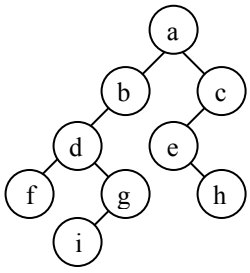
پیمایش درخت

منظور از پیمایش درخت دسترسی به تمامی گره‌های درخت است.

انواع پیمایش

الف) سطحی (اول سطح- BFS)

در این روش از سطح اول به بعد، گره‌های هر سطح از چپ به راست پیمایش می‌شود. برای پیمایش سطحی از صف استفاده می‌شود.



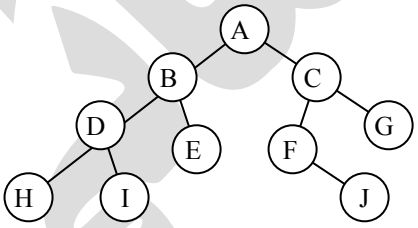
BFS : abcdefghi

ب) پیمایش عمقی (اول عمق- DFS)

٦ نوع پیمایش عمقی داریم که ۳ نوع آنها اصلی و پرکاربرد هستند. زیردرخت سمت چپ را با L، زیر درخت سمت چپ راست را با R و ریشه را با N(Node) یا V(visit) یا D(Data) یا نشان می‌دهیم.

۱- پیمایش inorder (میانوندی) LNR=LVR=LDR

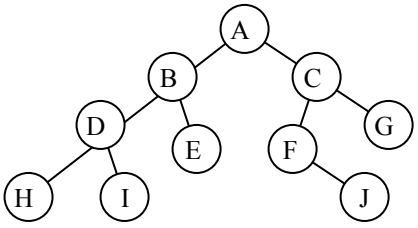
در این روش اول زیر درخت چپ، بعد ریشه و بعد زیردرخت راست پیمایش می‌شود.



IN:HDIBEAFJCG

۲- پیمایش preorder (پیشوندی) NLR=VLR=DLR

در این روش ابتدا ریشه پس زیر درخت چپ و در آخر زیر درخت راست پیمایش می‌شود.

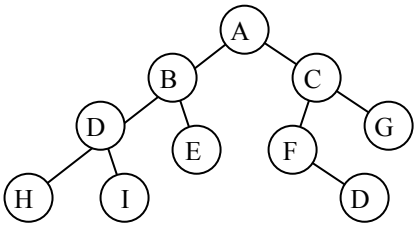


PRE:A B D H I E C F J G

نکته: در این روش ریشه درخت در ابتدای پیمایش قرار می‌گیرد.

۳- پیمایش پسوندی (Postorder) LRN=LRV=LRD

در این روش ابتدا زیر درخت چپ و پس زیر درخت راست و بعد ریشه پیمایش می‌شود.



POST: H I D E B J F G C A

نکته: در این روش ریشه در آخر پیمایش ظاهر می‌شود.

- با پیمایش پیشوندی و میانوندی می‌توان درخت یکتای آن پیمایش‌ها را رسم کرد.
- با پیمایش پسوندی و میانوندی می‌توان درخت یکتای آن پیمایش‌ها را رسم کرد.
- با پیمایش پیشوندی و پسوندی نمی‌توان درخت یکتای آن پیمایش‌ها را رسم کرد.

ساخت درخت از روی پیمایش پیشوندی و میانوندی:

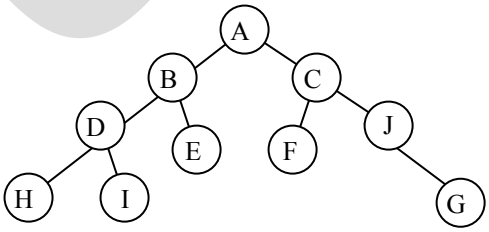
اولین حرف پیمایش پیشوندی را بعنوان ریشه انتخاب می‌کنیم سپس ریشه را در پیمایش میانوندی پیدا می‌کنیم و سمت راست آنرا بعنوان زیر درخت سمت راست و سمت چپ را بعنوان زیر درخت چپ انتخاب می‌کنیم. برای پیدا کردن ریشه در هر یک از زیردرختها باید ببینیم کدامیک از حروف در پیمایش پیشوندی زودتر آمده است.

مثال: درخت پیمایش‌های زیر را رسم کنید.

PRE: A B D H I E C F J G IN: H D I B E A F C J G

اولین حرف پیمایش پیشوندی ریشه درخت است. پس A ریشه درخت است. ریشه را در پیمایش میانوندی جستجو می‌کنیم. گره‌های قبل از A در زیر درخت سمت چپ و گره‌های بعد از A در زیر درخت سمت راست است .

برای یافتن ریشه در زیردرخت چپ باید به پیمایش پیشوندی نگاه کنیم هر کدام زودتر دیده شوند، ریشه زیر درخت چپ است (B) برای یافتن ریشه زیر درخت راست هم باید به پیمایش پیشوندی نگاه کنیم هر گره که زودتر دیده شود، ریشه زیر درخت راست است (C) اینکار را ادامه می‌دهیم تا تمام درخت رسم شود.



ساخت درخت از روش پیمایش پسوندی و میانوندی:

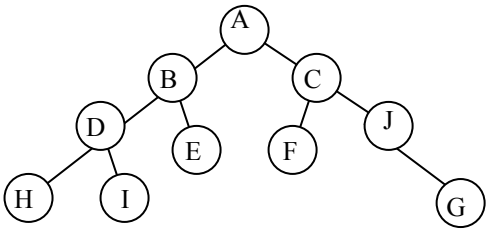
در اینحالت نیز برای ساخت درخت، مانند قبل عمل می‌کنیم با این تفاوت که باید از راست به چپ نگاه کنیم. هر گره‌ای کـه از راست به چپ زودتر بیاید، ریشه زیر درخت است.

مثال: درخت مربوط به پیمایش‌های زیر را بدست آورید.

POST: H I D E B F G J C A IN: H D I B E A F C J G

اولین حرف پیمایش پسوندی از راست، ریشه درخت است. پس A را بعنوان ریشه انتخاب می‌کنیم.

سپس ریشه را در میانوندی جستجو می‌کنیم. گره‌های سمت چپ آن، زیر درخت سمت چپ هستند که برای یـافتن ریشـه آن زیر درخت باید ببینیم کدام گره از سمت راست پیمایش پسوندی زودتر آمده‌اند. گره‌های سمت راست ریشه درخـت نیـز، زیـر درخت سمت راست هستند که برای یافتن (B) ریشه آن زیر درخت باید ببینیم کدام گـره از سـمت راسـت پیمـایش پسـوندی زودتر آمده است. (C) اینکار را ادامه می‌دهیم تا تمام درخت ایجاد شود.



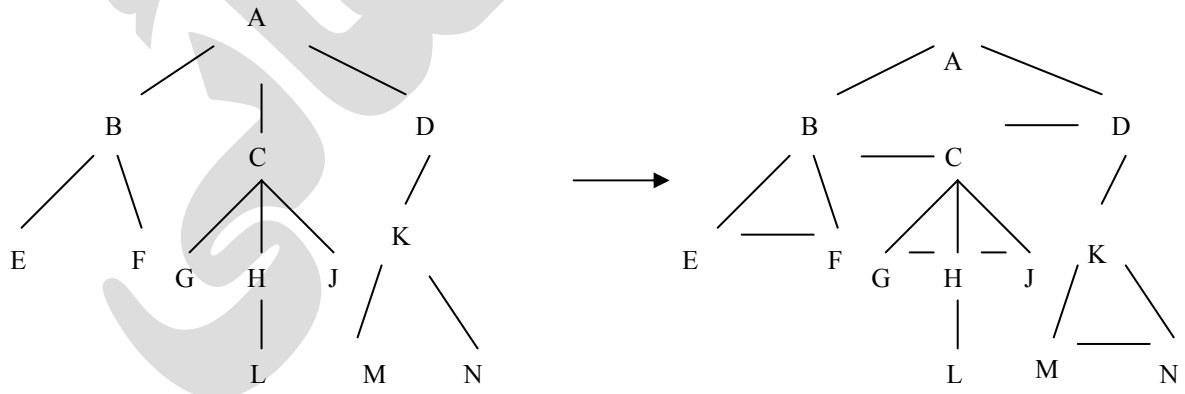
تبدیل درخت عمومی به درخت دودویی:

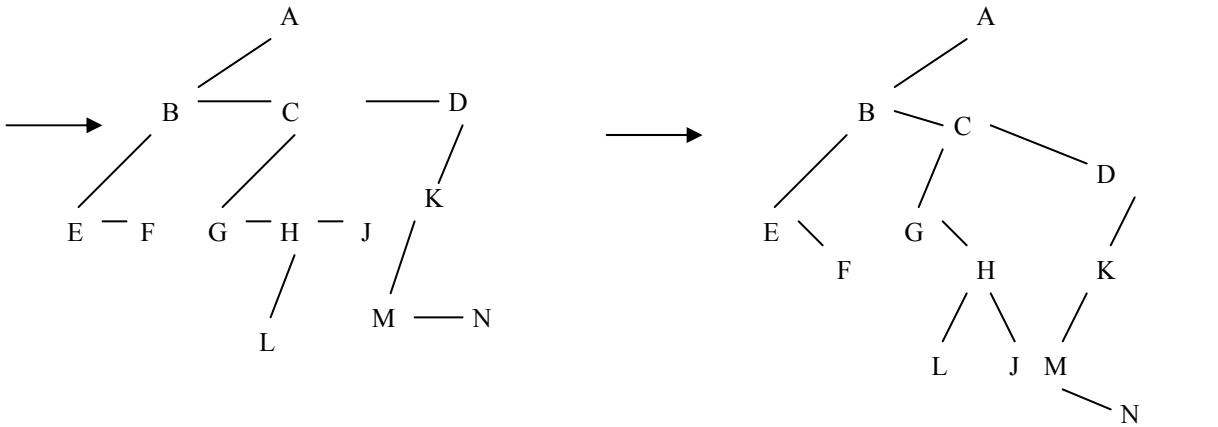
طی مراحل زیر می‌توان یک درخت عمومی را به درخت دودویی معادل آن تبدیل کرد.

۱- در هر سطح کلیه گره‌های کنار هم که فرزند یک پدر هستند را به هم وصل می‌کنیم.

۲- ارتباط همه گره‌ها با پدر را بجز اتصال سمت چپ‌ترین فرزند را قطع می‌کنیم.

۳- گره‌های متصل به هم در هر سطح افقی را ٤٥ درجه در جهت حرکت عقربه‌های ساعت می‌چرخانیم.



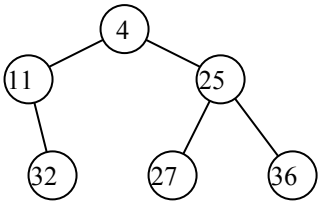


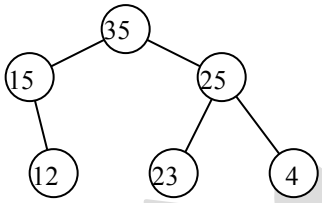
درختان خاص

درخت Heap (نیمه مرتب)

درخت maxtree: درختی که مقدار هر گره از فرزندانش بزرگتر باشد.

درخت mintree: درختی که مقدار هر گره از فرزندانش کوچکتر باشد.





درخت Max Heap: درخت دودویی کاملی است که maxtree باشد.

درخت Min Heap: درخت دودویی کاملی است که mintree باشد.

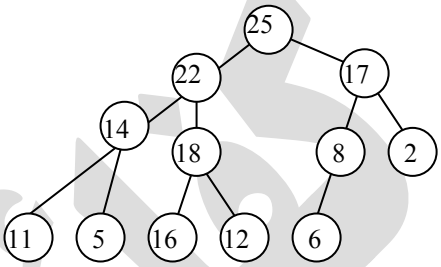
درخت Heap: درختی است که یا Max Heap باشد یا Min Heap. (پیش فرض Max Heap).

از آنجا که درخت Heap درخت کاملی است لذا:

۱- عمق آن همیشه $\lfloor \log_2^n \rfloor + ۱$ است.

۲- برای نمایش آن از روش ترتیبی (آرایه‌ای) استفاده می‌شود.

- در درخت هیپ وجود کلید تکراری منعی ندارد.

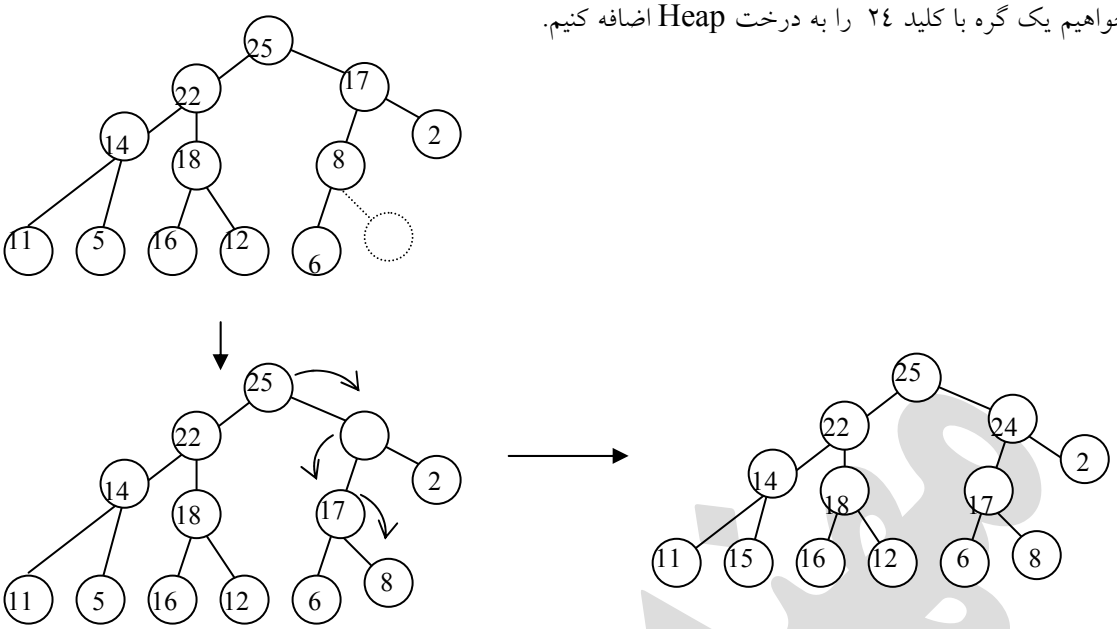


نحوه درج یک گره در درخت Max Heap:

اعمال درج، حذف در درخت نیمه مرتب باید به گونه‌ای انجام شود که درخت همچنان نیمه مرتب باقی بماند.

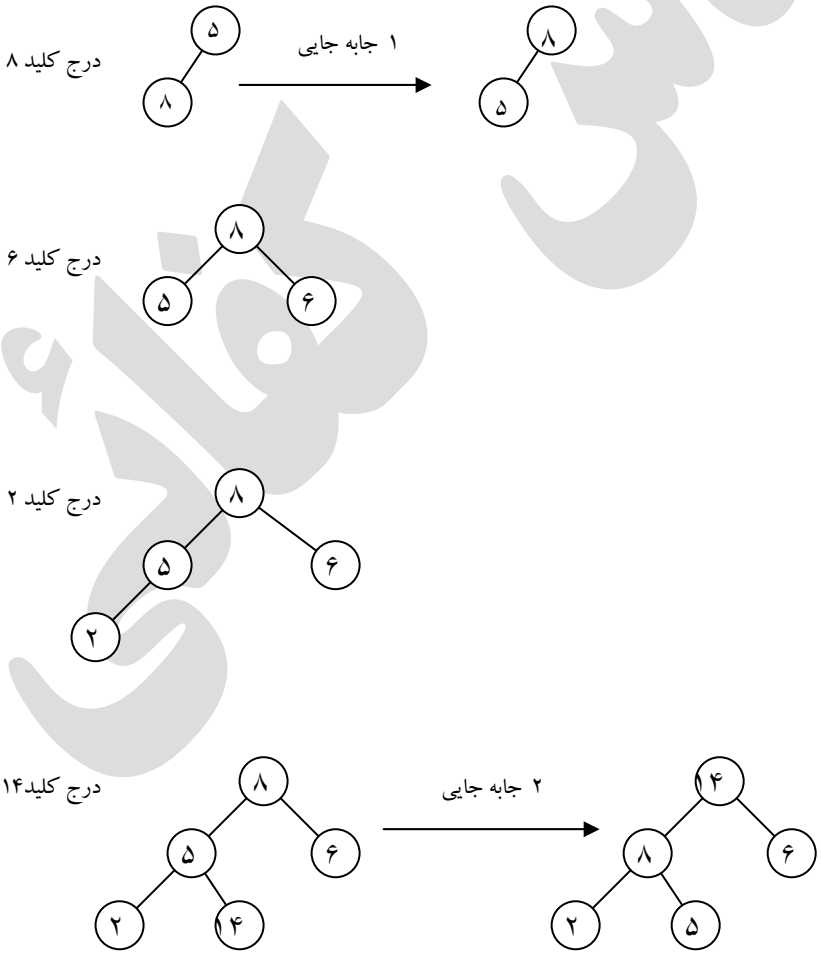
جهت درج یک گره ابتدا آن گره را در آخرین مکان به درخت اضافه می‌کنیم (زیراساختار باید همچنان کامل بماند) سپس کلید گره جدید را با کلید گره پدر مقایسه می‌کنیم اگر از آن بزرگتر بود آنرا با کلید گره پدر جابجا می‌کنیم به همین ترتیب مقایسه با کلید گره پدر ادامه می‌یابد تا یا به ریشه برسیم یا کلید گره جدید از کلید گره پدر، کوچکتر یا مساوی شود.

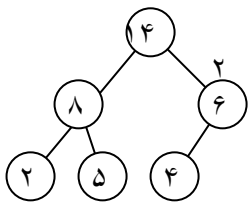
مثال: می‌خواهیم یک گره با کلید ۲٤ را به درخت Heap اضافه کنیم.



ساخت درخت Max Heap:

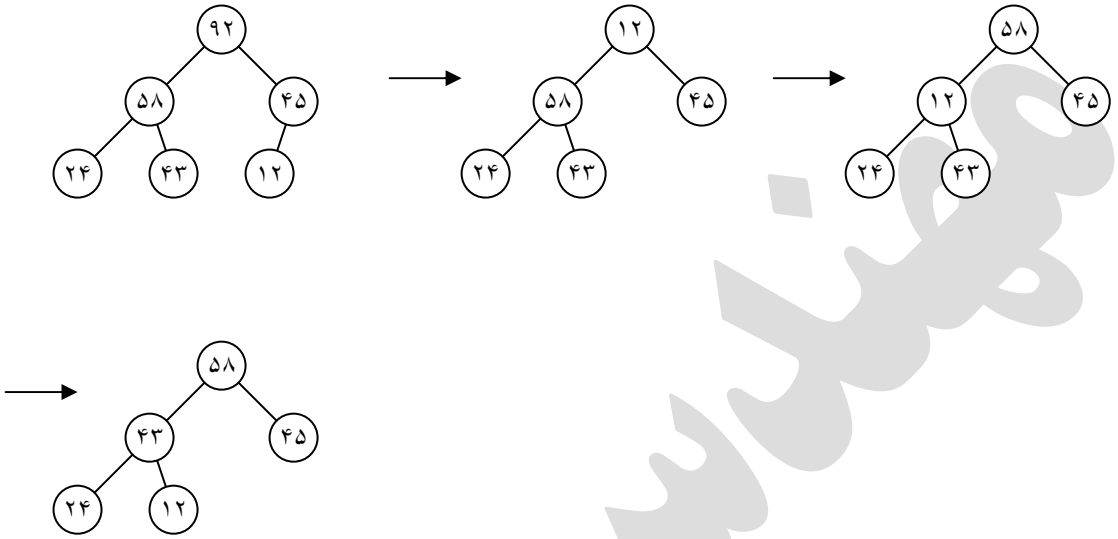
ابتدا کلید اول را بعنوان ریشه درخت در نظر گرفته و ازکلید دوم عمل درج را انجام می‌دهیم. 5,8,6,2,14,4,2,18





حذف عنصر از درخت Max Heap:

در عمل حذف، همواره مقدار ریشه حذف شده و سمت راست‌ترین عنصر موجود در پائین‌ترین سطح در ریشه قرار گرفته و درخت مجدداً تنظیم می‌شود. در تنظیم درخت عنصر ریشه تا جایی که از فرزندان خود کوچکتر است، با بزرگترین آن تعویض می‌شود.



کاربرد Heap در مرتب سازی:

یکی از کاربردهای مهم Heap در مرتب‌سازی است. فرض کنیم آرایه A با n عنصر داده شده است. برای مرتب کردن این آرایه از الگوریتم Heap Sort استفاده می‌شود بدین صورت که:

۱- یک درخت Heap از عناصر آرایه A بسازیم.

۲- عنصر ریشه را بطور مکرر حذف کنیم.

برای مرتب سازی صعودی از درخت Min Heap و برای مرتب سازی نزولی از درخت Max Heap استفاده می‌شود.

درخت جستجوی دودویی یا BST (Binary Search Tree)

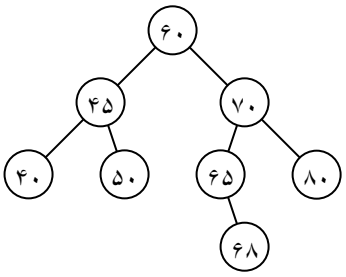
درخت Heap از نظر حذف و درج دارای پیچیدگی زمانی بسیار خوبی است ولی از نظر جستجو مطلوب نیست و مزیتی ندارد. لذا نوع درخت دیگری معرفی می‌کنیم که مخصوص جستجو طراحی شده است و اعمال جستجو را سریعتر انجام می‌دهد.

درخت BST:

درختی که دودویی است و کلید دو گره از کلید فرزندان سمت چپ بزرگتر و از کلید فرزندان سمت راست کوچکتر باشد. در این درخت گره با کلید تکراری وجود ندارد.

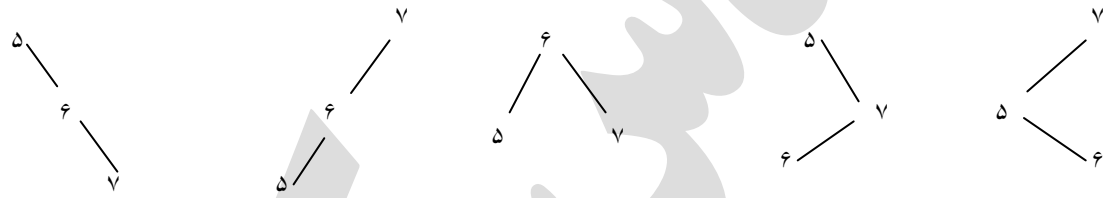
برای جستجوی یک کلید خاص از ریشه شروع می‌کنیم و اگر از ریشه بزرگتر بود به زیر درخت سمت راست می‌رویم و اگر کوچکتر بود به زیر درخت سمت چپ می رویم و این مقایسه را آنقدر تکرار می‌کنیم تا یا به گره مورد نظر برسیم یا زیردرختی که باید به آن مراجعه کنیم تهی باشد.

نکته: پیمایش inorder درخت BST باعث می‌شود لیست عناصر درخت بصورت صعودی مرتب شود.



نکته: با n کلید مختلف، به تعداد $\frac{۱}{n+۱}\binom{۲n}{n}$ درخت BST می‌توان ساخت .

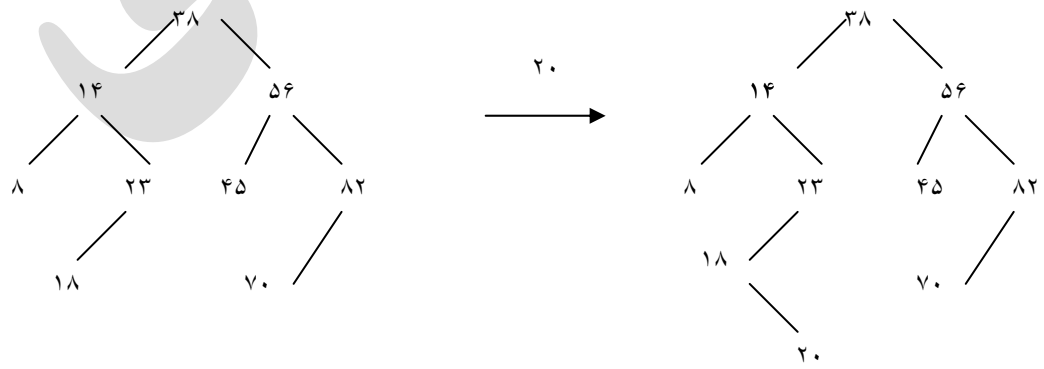
مثال: با اعداد ۵ و ٦ و ۷ چند درخت BST می‌توان ایجاد کرد؟



اضافه کردن یک عنصر به BST:

برای اضافه کردن یک عنصر، آن عنصر را در درخت جستجو می‌کنیم. اگر آن عنصر را پیدا کنیم آن عنصر را دور می‌اندازیم (در BST کلیدها منحصر بفرد است) و در صورت پیدا نکردن آن، آن عنصر را در محلی که جستجو خاتمه یافته اضافه می‌کنیم.

مثال: درخت زیر را در نظر بگیرید و عنصر ۲۰ را اضافه کنید.

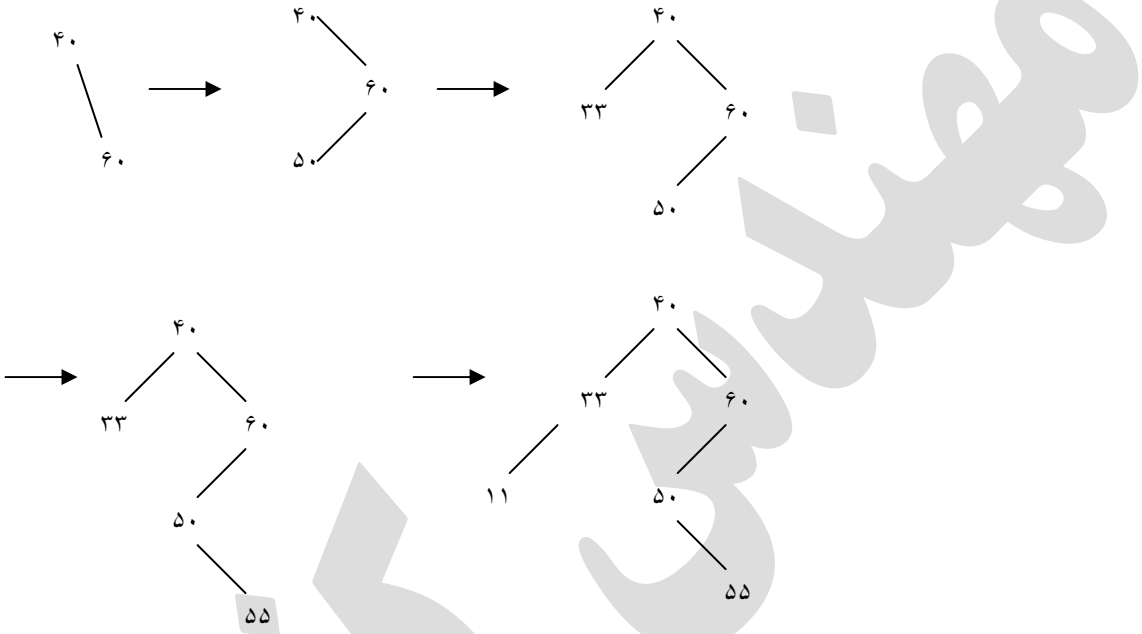


ساخت درخت BST:

برای ساخت درخت اولین گره را بعنوان ریشه انتخاب کرده سپس به تعداد گره‌های باقیمانده عمل درج را انجام می‌دهیم.

مثال: ٦ عدد زیر را در نظر گرفته و درخت BST آنها را رسم کنید.

۱۱ و ۵۵ و ۳۳ و ۵۰ و ٦۰ و ٤۰



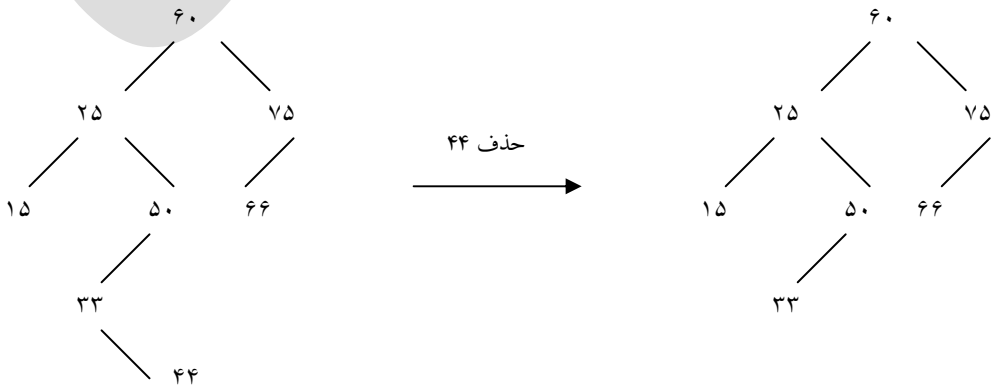
حذف یک عنصر از BST:

۱- اگر گره مورد نظر برگ باشد، آن گره را حذف کرده و نود پدر بعنوان برگ درخت در نظر گرفته می‌شود.

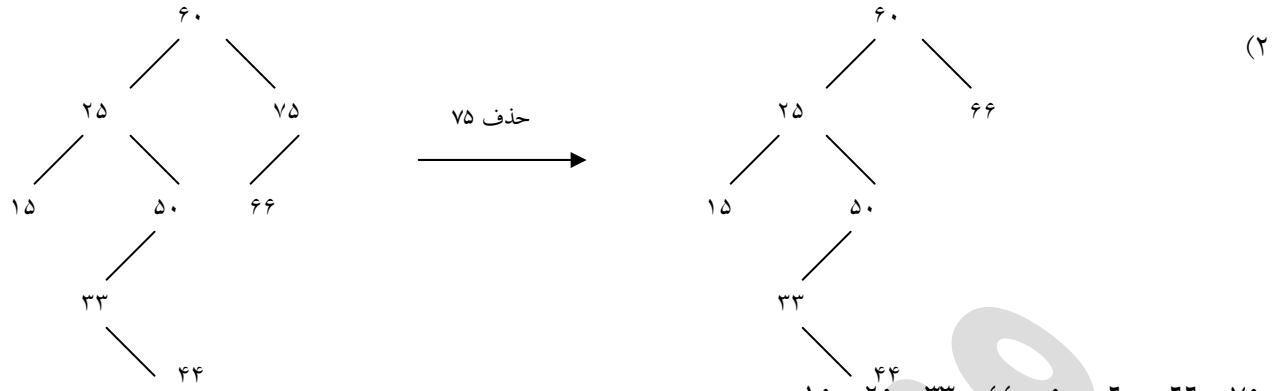
۲- اگر گره مورد نظر یک فرزند دارد، آنگاه آنرا حذف کرده و ارتباط نود پدر را با تنها فرزند برقرار می‌کنیم.

۳- اگر گره دو فرزند دارد، آنگاه آن گره را حذف کرده و سپس پیمایش inorder درخت را می‌نویسیم و یکی ا ز گره‌هایی که قبل یا بعد گره حذف شده می‌آیند را جایگزین آن می‌کنیم در حقیقت بزرگترین گره در زیر درخت چپ یا کوچکترین گـره در زیر درخت سمت راست می‌تواند جایگزین گره حذف شده شود.

۱)



۲)



۷۵ و ٦٦ و ٦٠ و ۵۰ و ٤٤ و ۳۳ و ۲۵ و ۱۵

۳)