درخت ها

فرشته دهقانی
۹۹-۹۸

فهرست مطالب

- درخت و ذخیره سازی آن
- پیمایش درخت
- درخت دودویی جستجو
- جستجو، درج، حذف
- میانگین ارتفاع درخت جستجو
- پایین ترین جد مشترک

درخت‌ها

- داده‌ساختارهایی خطی: آرایه‌ها، لیست‌های پیوندی، صف‌ها و پشته‌ها
  - مناسب برای مشخص کردن ترتیبی از عناصر که در خود ذخیره کرده‌اند

- داده ساختار غیرخطی: درخت
  - یکی از این کاربردها ذخیره‌سازی ساختار پوشه‌ها در کامپیوتر است که از ساختاری سلسله مراتبی پیروی می‌کند.
- همچنین می‌توان درخت‌ها را در ساختارهایی کارآمدتر برای انجام عمل هایی خاص روی ساختارهای خطی نیز به کار گرفت.

مفاهیم و تعاریف

- **درخت**: گراف همبند بدون دور است
- **درخت ریشه‌دار**: درختی است که در آن یک رأس خاص به عنوان ریشه انتخاب شده است و ساختاری سلسله مراتبی به آن بخشیده است.
- **عمق (depth) یک رأس**: فاصله‌ی رأس تا ریشه را عمق آن می‌نامیم.
- **ارتفاع (height) درخت**: ماکسیمم عمق در بین همه‌ی رأس‌ها را ارتفاع درخت می‌نامیم.
- **پدر و فرزند**: وقتی دو رأس مجاور باشند، رأسی که به ریشه نزدیک‌تر است را پدر دیگری می‌نامیم و رأس دورتر را فرزند رأس نزدیک‌تر.
- **اجداد**: اجداد یک رأس همه‌ی رئوسی هستند که روی مسیر آن رأس به ریشه قرار دارند. منظور از جد i ام یک رأس جدی است که فاصله‌اش تا آن رأس i است.
- **نوادگان**: نوادگان یک رأس همه‌ی رئوسی هستند که این رأس جدشان است. زیردرخت یک رأس مجموعه‌ی همه‌ی نوادگان آن رأس است.
- **برگ**: رأسی که هیچ فرزندی نداشته باشد (این اصطلاح از درختان واقعی گرفته شده است).

ساختار یک درخت

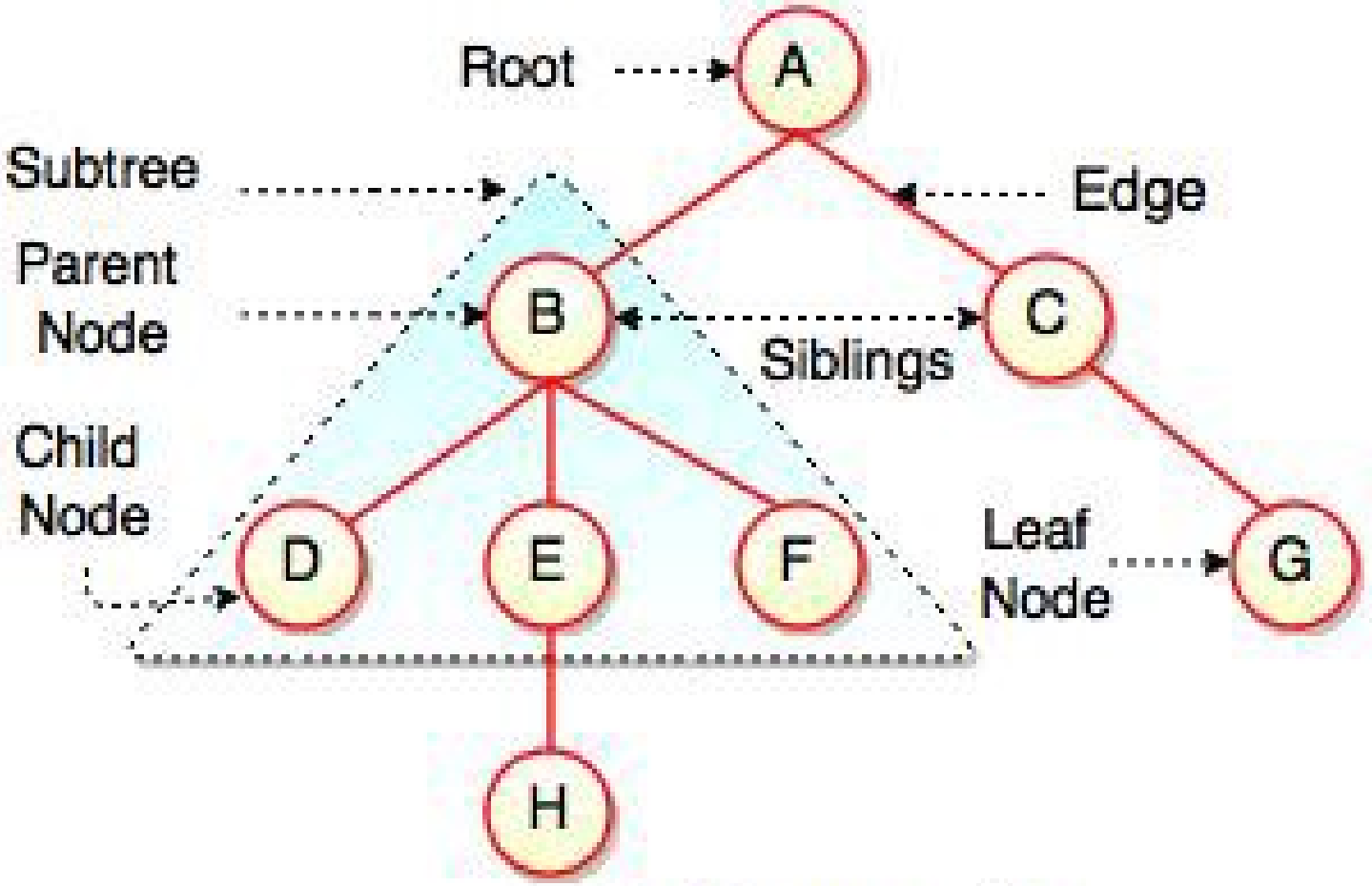
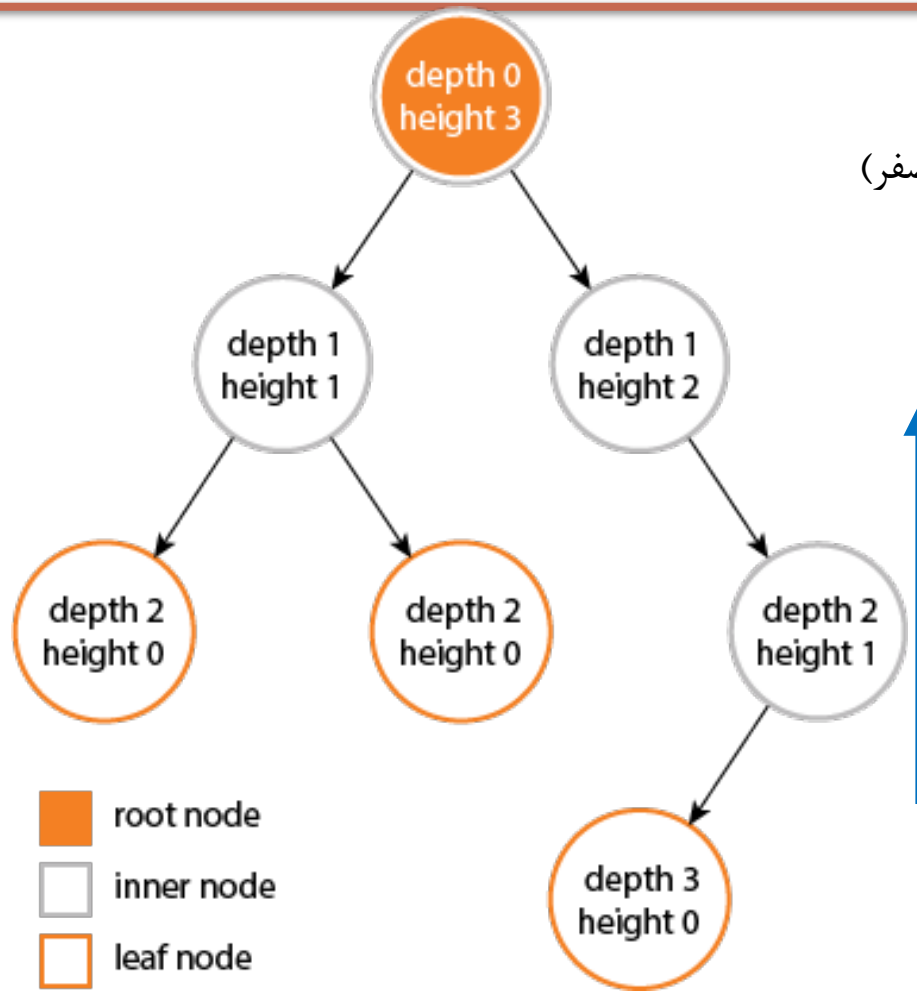
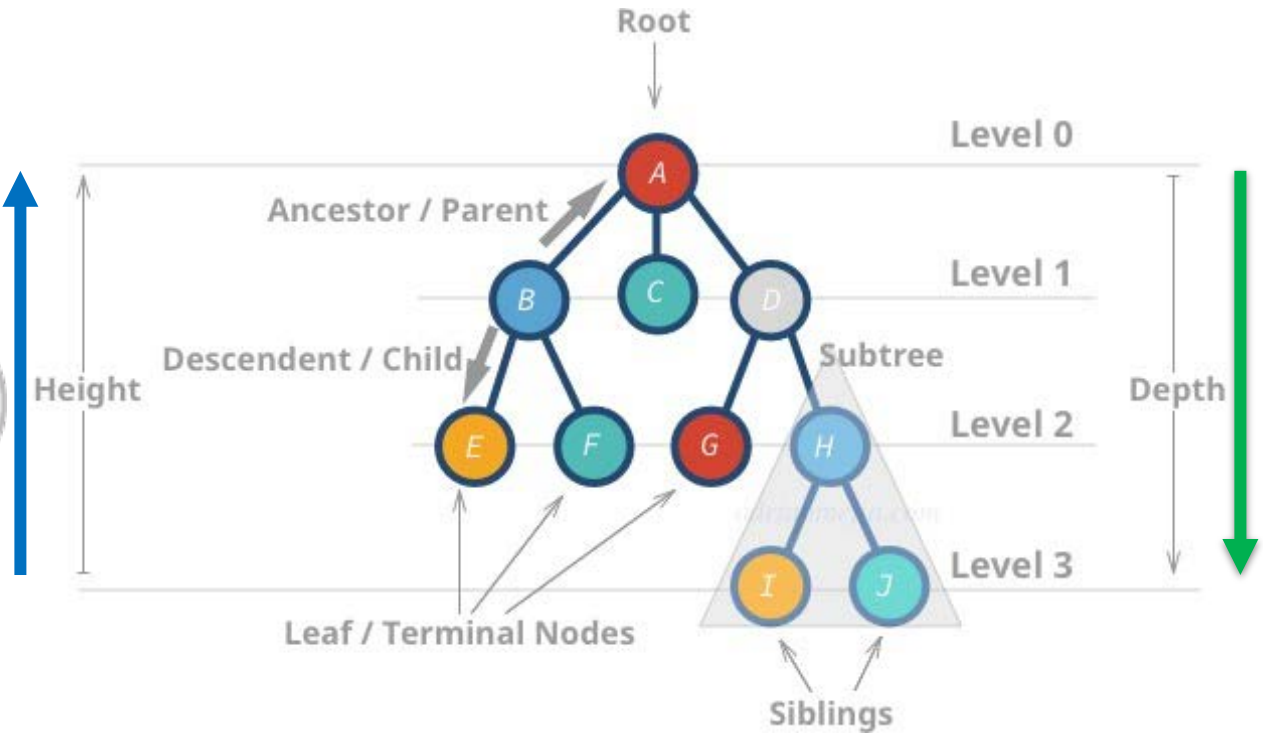


Fig. Structure of Tree

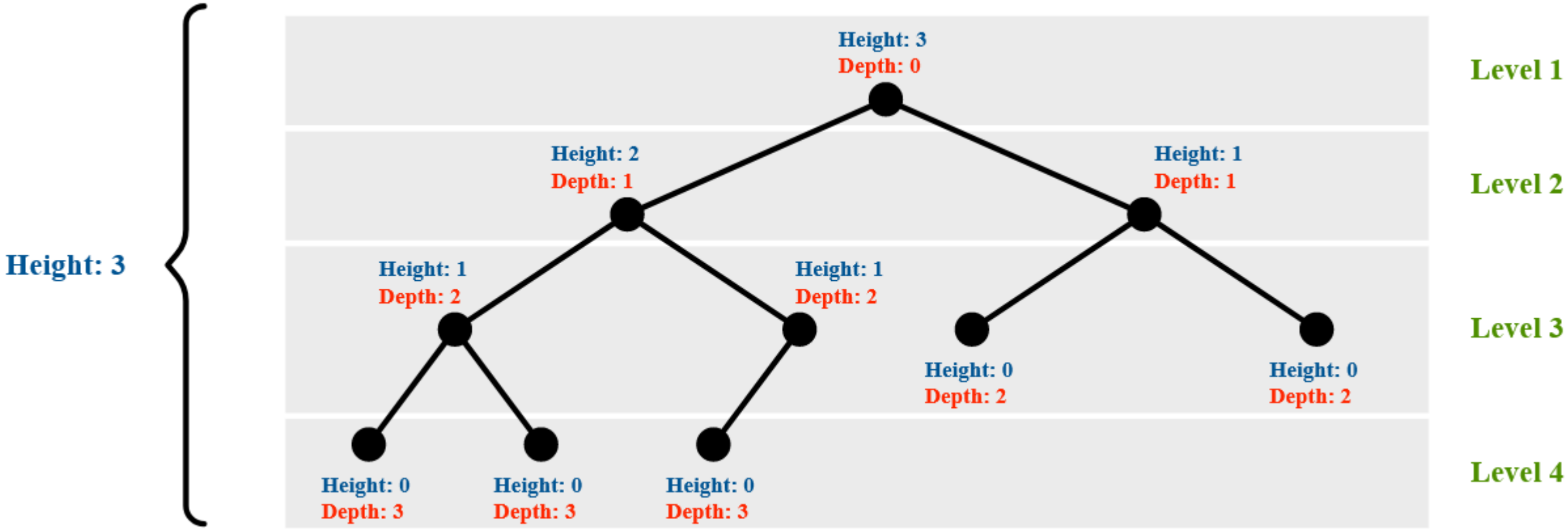
تفاوت بین عمق و ارتفاع

- **عمق (depth) یک رأس**: تعداد یال ها از رأس مورد نظر تا ریشه (عمق ریشه: صفر)
- **ارتفاع (height) یک راس**: تعداد یالها از بلندترین مسیر آن راس تا برگ (ارتفاع برگ: صفر)



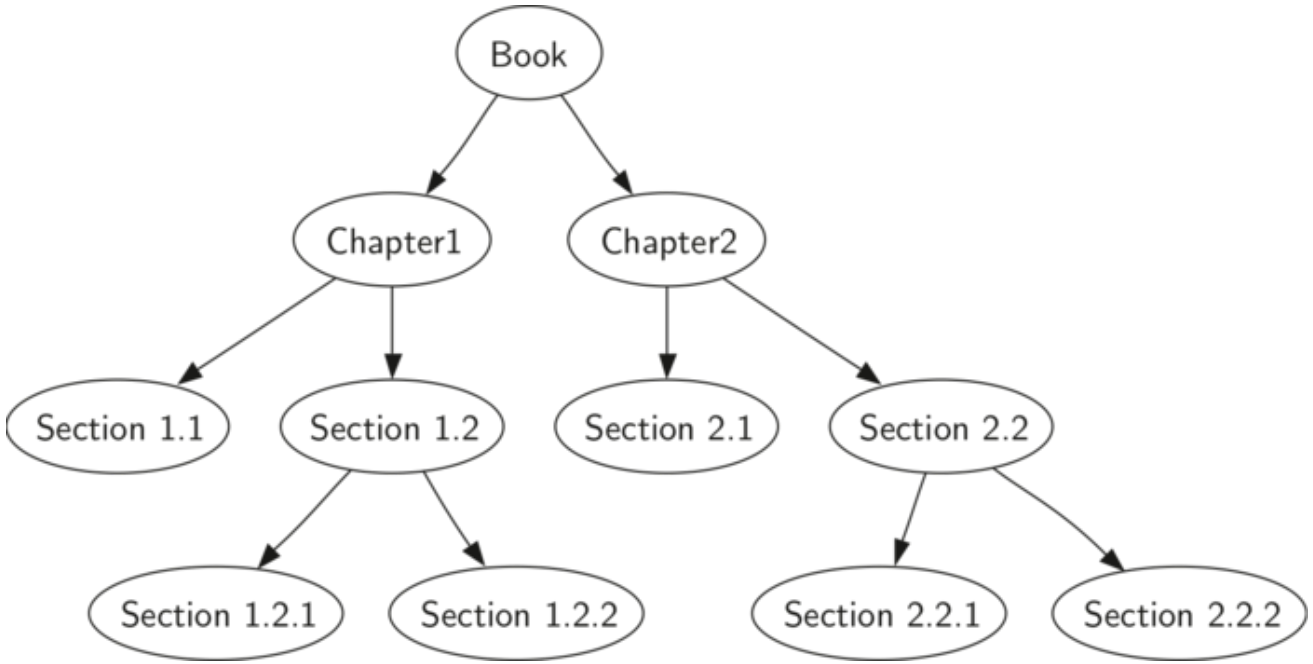


مثال دیگر..


Height: 3
Height: 3
Depth: 0
Height: 2
Depth: 1
Height: 1
Depth: 1
Height: 1
Depth: 2
Height: 1
Depth: 2
Height: 0
Depth: 2
Height: 0
Depth: 2
Height: 0
Depth: 3
Height: 0
Depth: 3
Height: 0
Depth: 3
Level 1
Level 2
Level 3
Level 4

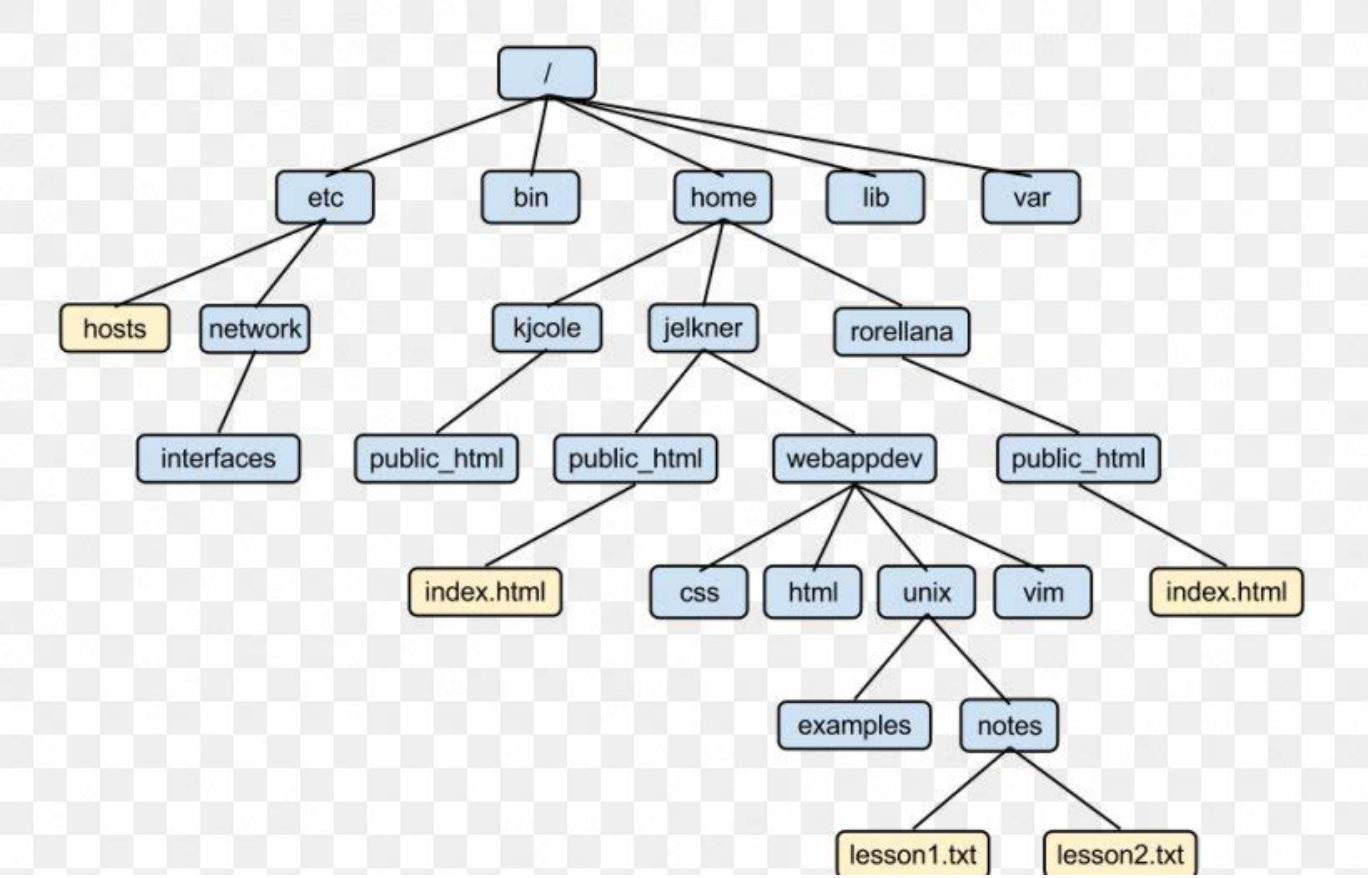
کاربرد درخت

نمایش ساختار کتاب

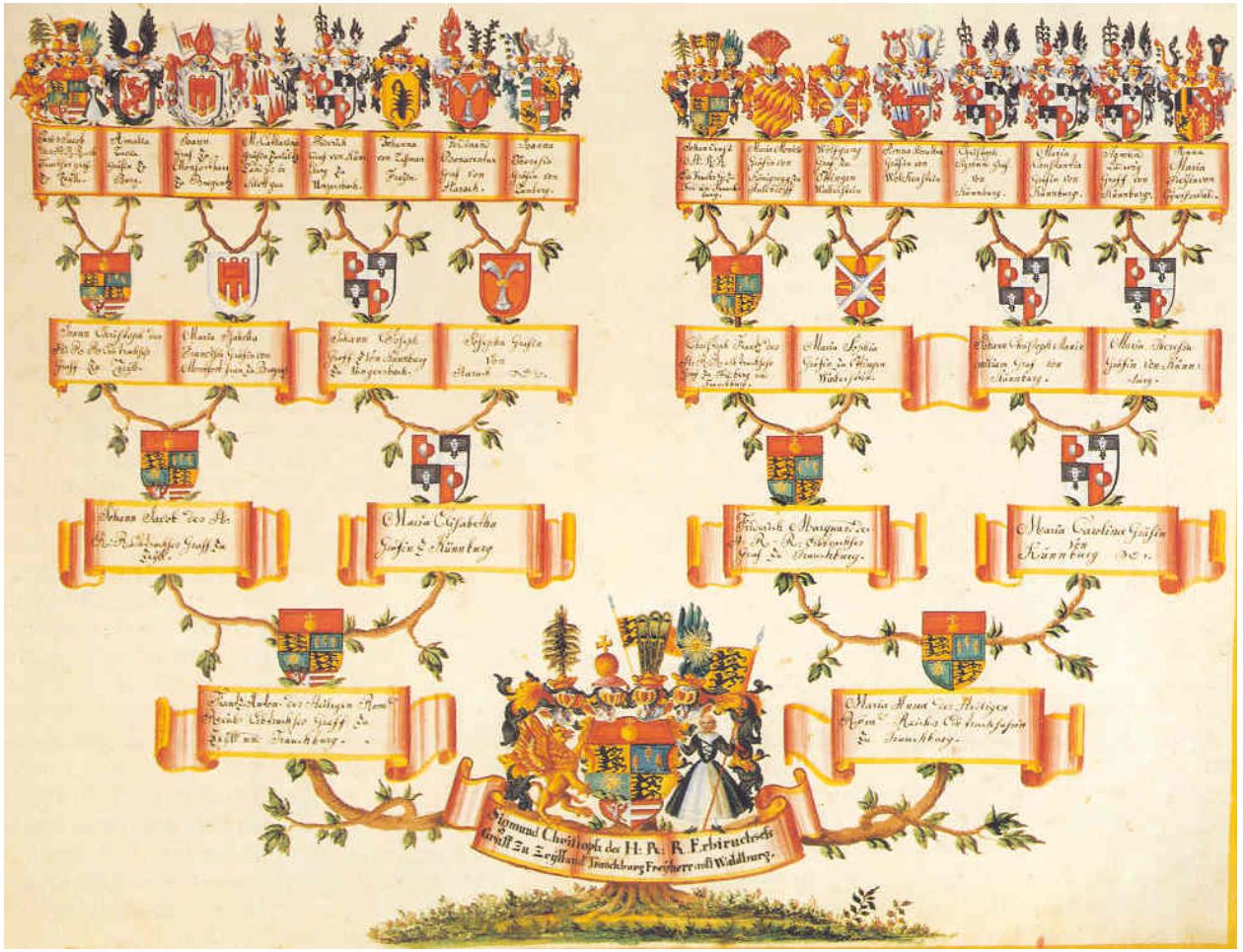


کاربرد درخت

نمایش ساختار فایل سیستم ها



کاربرد درخت – شجره نامه



پیاده سازی عمومی درخت (روش بد)

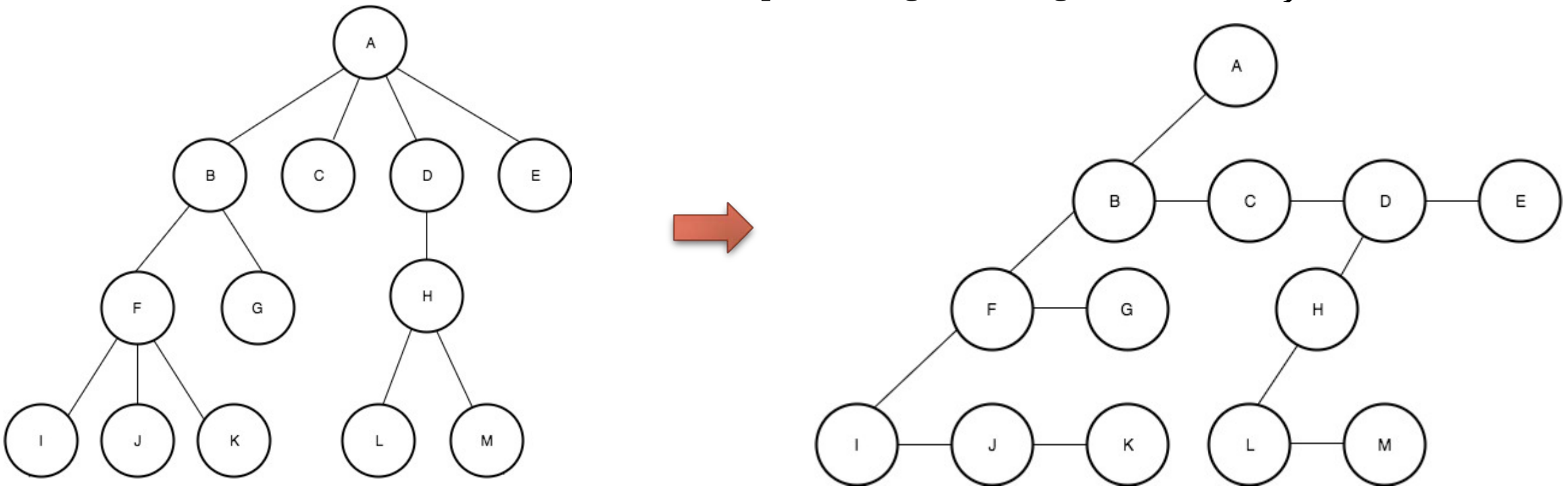
- هر گره شامل یک فیلد برای ذخیره اطلاعات (label) و k فیلد برای ذخیره آدرس فرزندان (k حد بالای تعداد فرزندان هر گره در درخت است)

item	Child 1	Child 2	Child 3	…	Child k

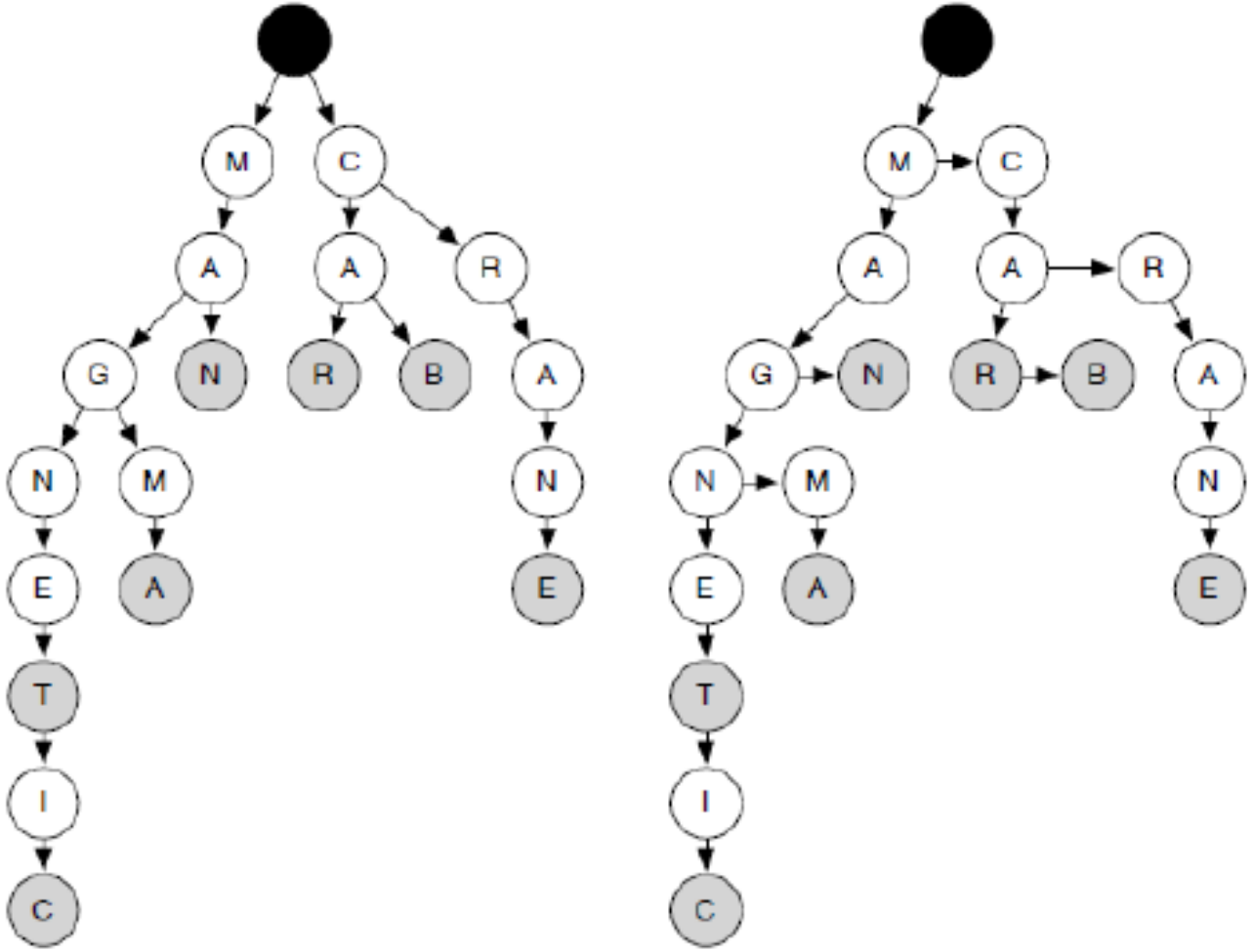
- تعداد کل فیلدهای اشاره گر به فرزندان : n*k
- تعداد فیلدهای اشاره گر غیر پوچ: n-1
- تعداد اشاره گر پوچ: nk-(n-1)=n(k-1)+1

چگونگی ذخیره‌سازی درخت
LEFTMOST CHILD-RIGHT SIBLING

1. برای هر رأس شماره‌ی پدرش و یک لیست از شماره‌های فرزندانش را ذخیره کنیم.
2. راه دیگر ذخیره‌سازی درخت استفاده از یک شیء برای نمایش هر رأس است.(برای هر رأس تنها ۳ اشاره‌گر انتخاب کرد: left_child، right sibling، parent)



LEFTMOST CHILD-RIGHT SIBLING



مزایا و معایب
LEFTMOST CHILD-RIGHT SIBLING

مزایا:

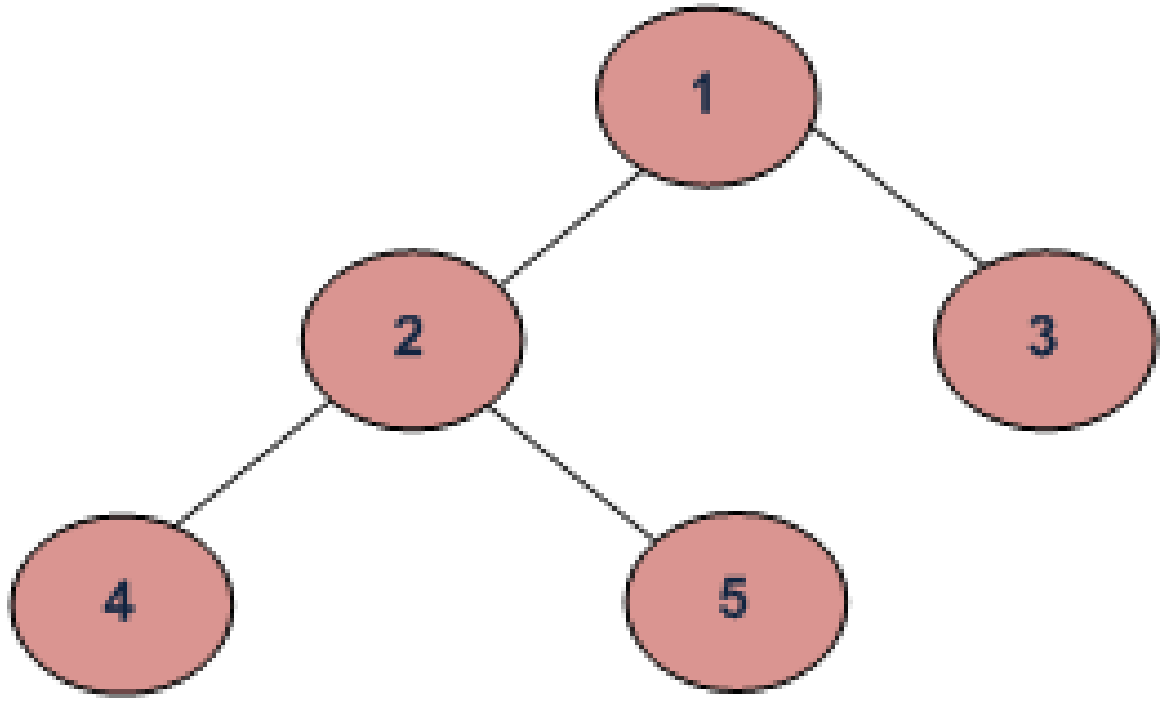
- دخیره حافظه : حداکثر هر گره دو عدد اشاره گر دارد
- پیاده سازی و تبدیل به کد آسان

معایب:

- عملیاتی مانند جستجو/درج/حذف زمان بیشتری را نسبت به نگهداری تمام فرزندان مستقیم هر درخت دارد. زیرا مثلا برای پیدا کردن اخرین فرزند یک گره باید تمام right sibling ها دیده شود تا به آن برسیم

پیمایش درخت

(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1



InOrder(root) visits nodes in the following order:

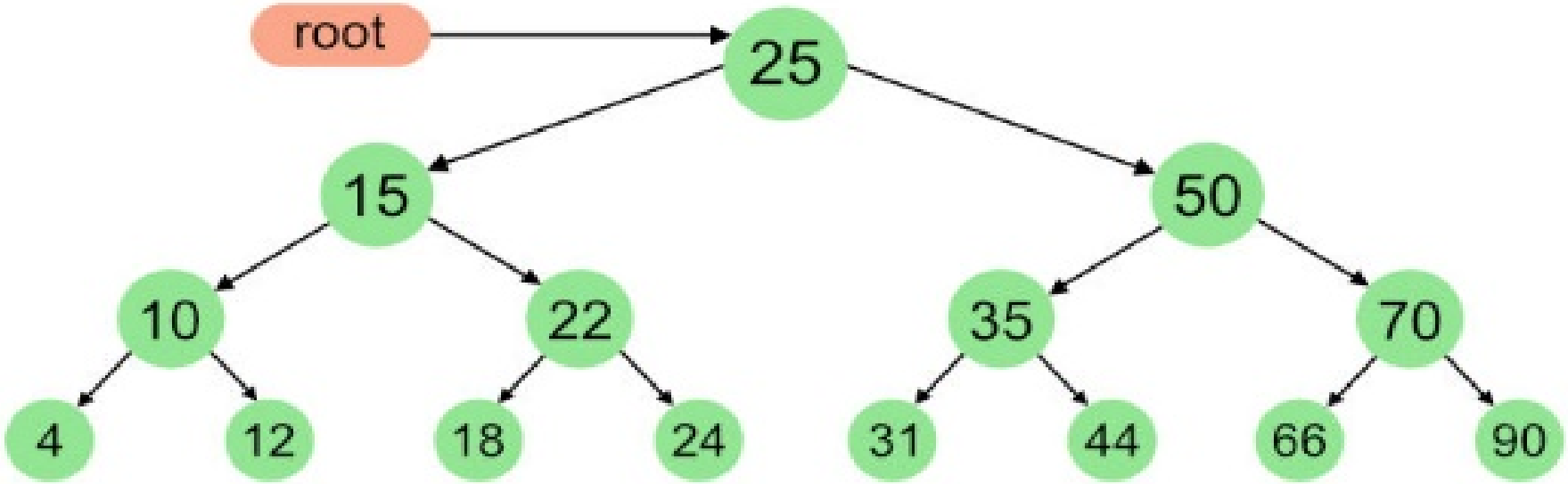
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:

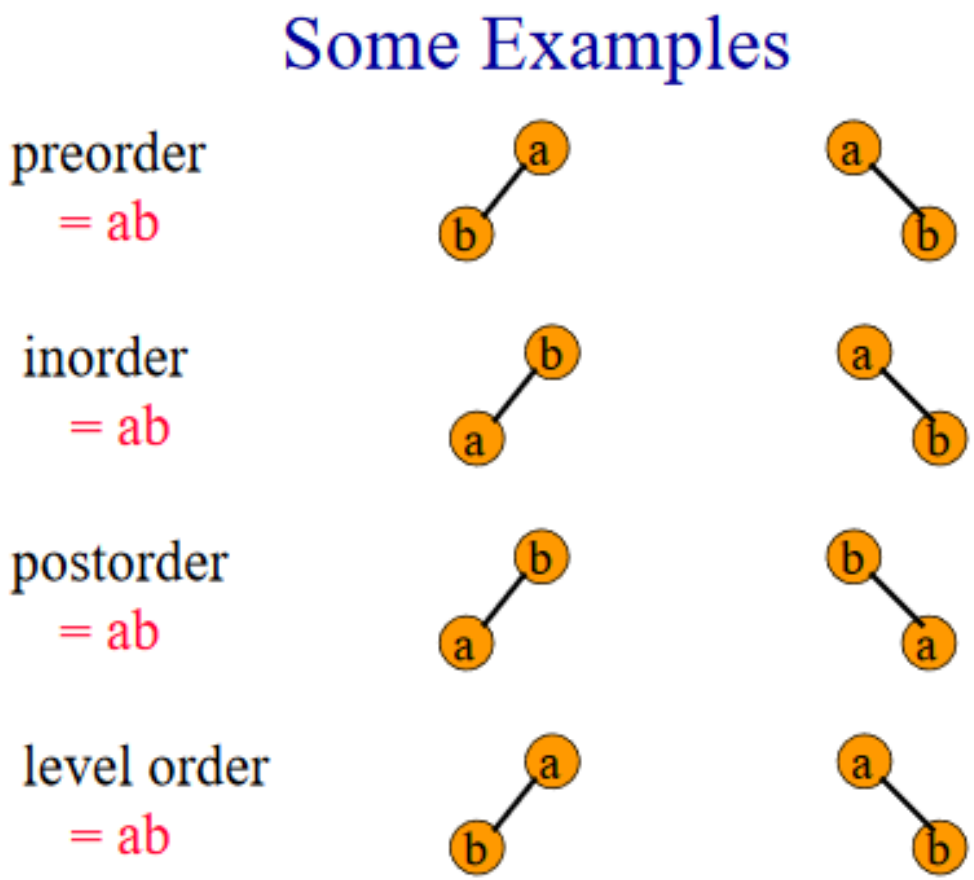
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



الگوریتم پیمایش درخت

```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
Algorithm Preorder(tree)
   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)
Algorithm Postorder(tree)
   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root.
```

آیا پیمایش ها، درخت یکسانی را تولید می کنند؟



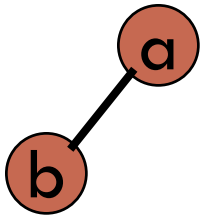
ساخت درخت با توجه به پیمایش های آن

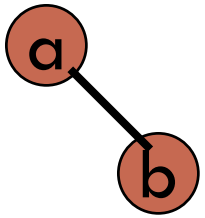
آیا با استفاده از هر دو نوع پیمایش درخت، می توان درخت یکتایی به دست آورد؟

PREORDER AND POSTORDER

preorder = ab

postorder = ba



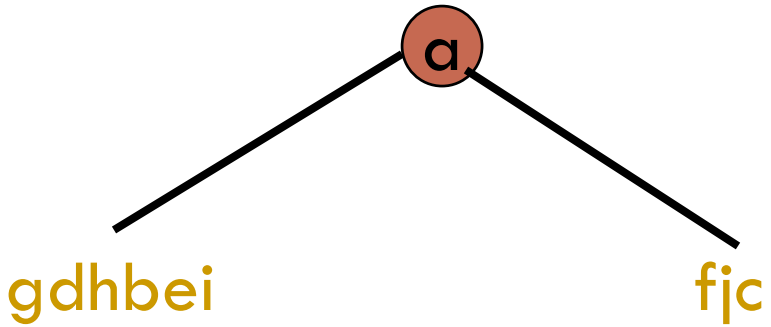


- Preorder and postorder **do not uniquely** define a binary tree.

INORDER AND PREORDER

inorder = g d h b e i **a** f j c

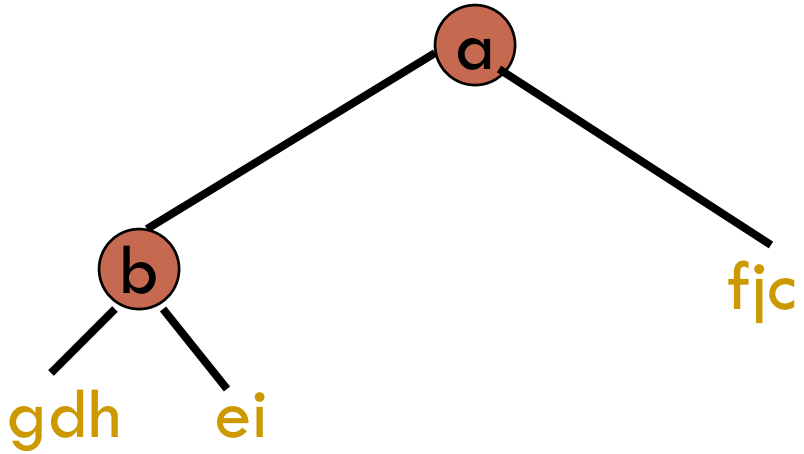
preorder = **a** b d g h e i c f j



INORDER AND PREORDER

inorder = g d h **b** e i **a** f j c

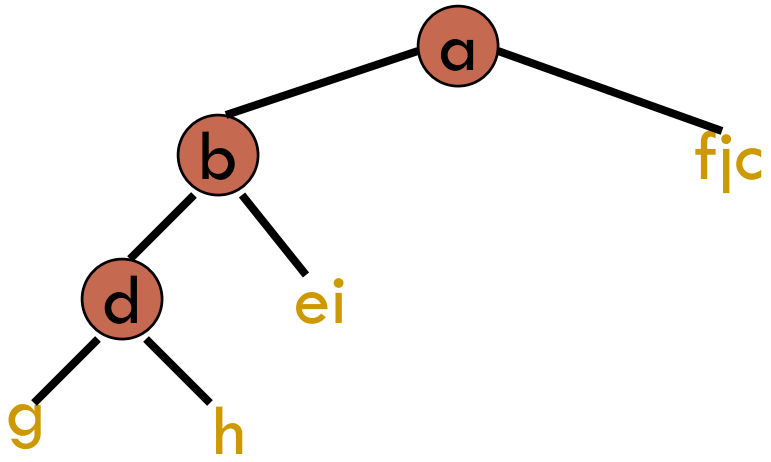
preorder = **a** **b** d g h e i c f j



INORDER AND PREORDER

inorder = g **d** h **b** e i **a** f j c

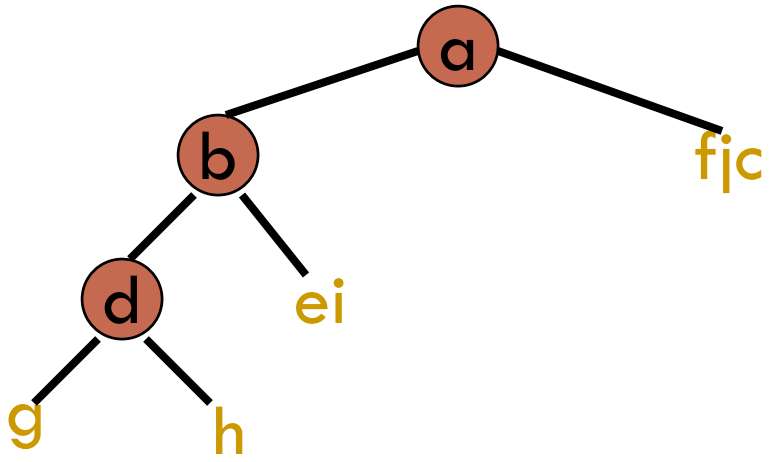
preorder = **a** **b** **d** g h e i c f j



INORDER AND PREORDER

inorder = **g** **d** h **b** e i **a** f j c

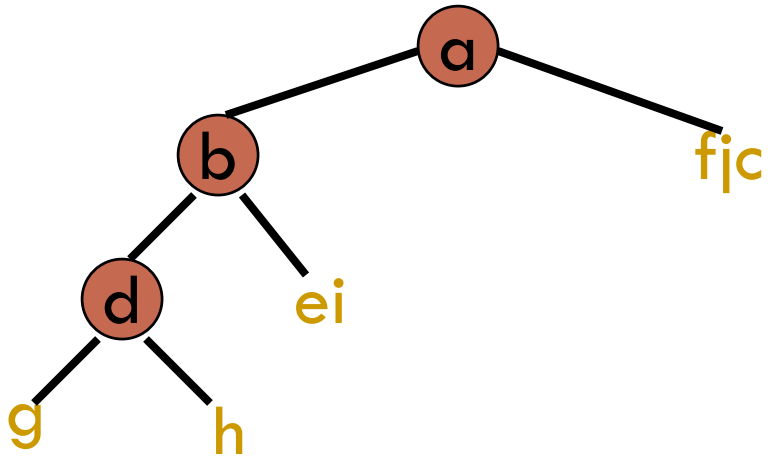
preorder = **a** **b** **d** **g** h e i c f j



INORDER AND PREORDER

inorder = **g d h b** e i **a** f j c

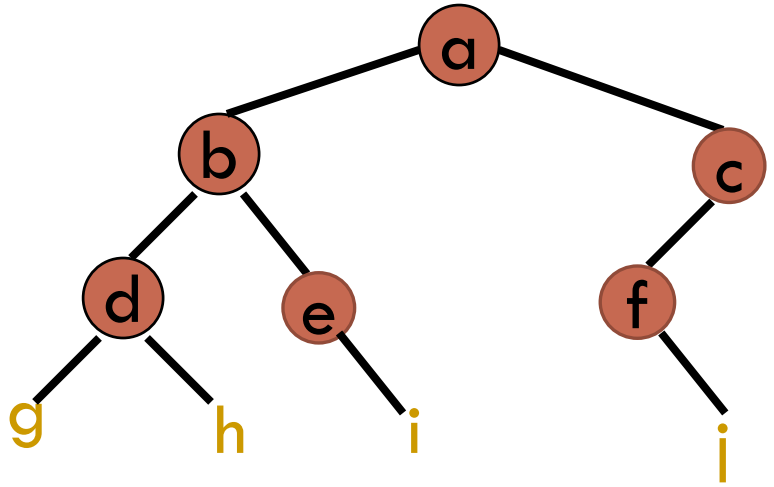
preorder = **a b d g h** e i c f j

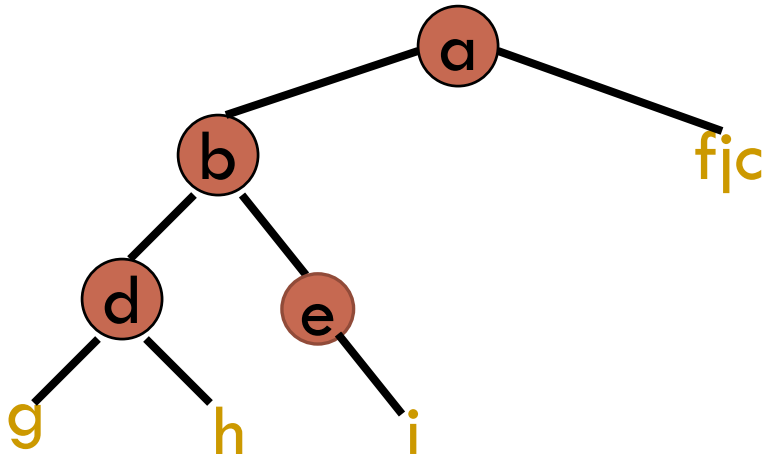


INORDER AND PREORDER

inorder = **g d h b** e i **a** f j c

preorder = **a b d g h** e i c f j



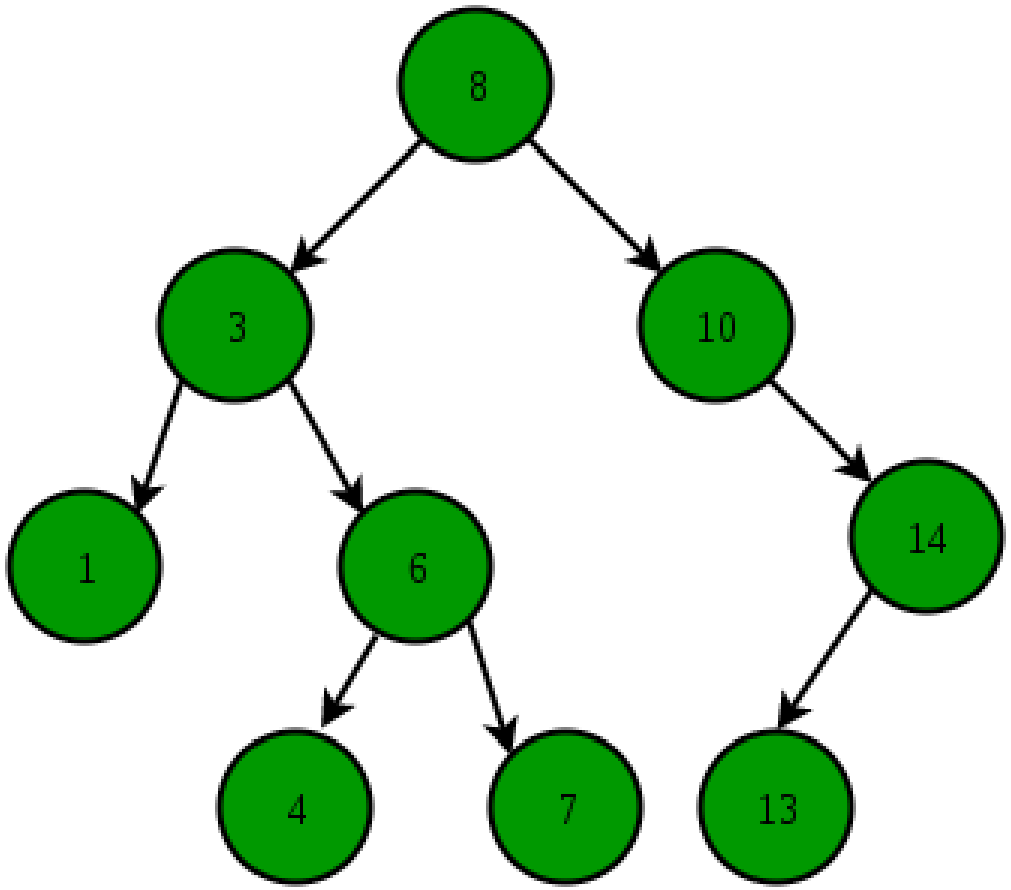


تمرین

- به زبان پایتون برنامه ای بنویسید که دو پیمایش Inorder و preorder را دریافت کند و پیمایش postorder آن را چاپ کند. (حداکثر درجه هر درخت دو باشد)
- به زبان پایتون برنامه ای بنویسید که دو پیمایش Inorder و postorder را دریافت کند و پیمایش preorder آن را چاپ کند. (حداکثر درجه هر درخت دو باشد)
- به زبان پایتون برنامه ای بنویسید که دو پیمایش Inorder و preorder را دریافت کند و اندازه زیر درخت فرزند چپ ریشه را چاپ کند

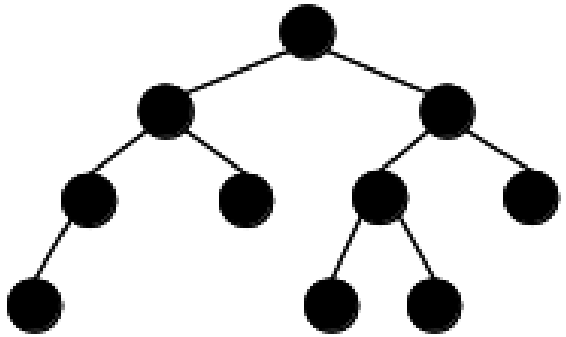
درخت دودویی

حداکثر تعداد فرزندان برابر دو است

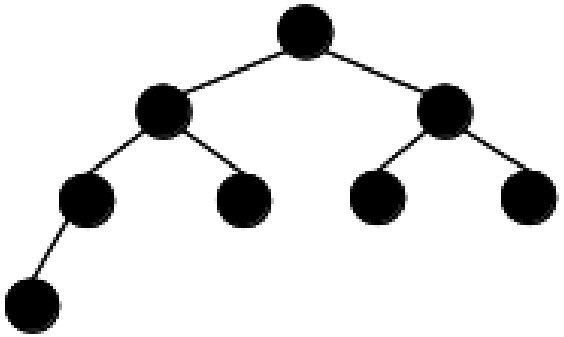


COMPLETE, FULL, PERFECT درخت

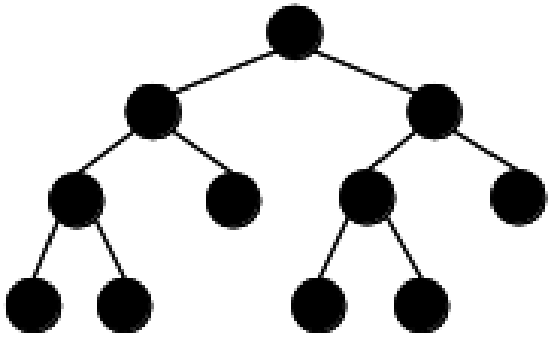
Neither complete nor full



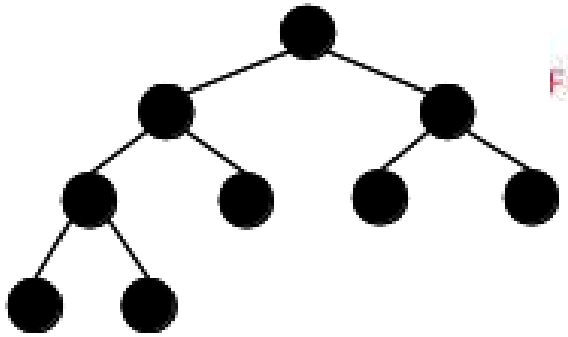
Complete but not full

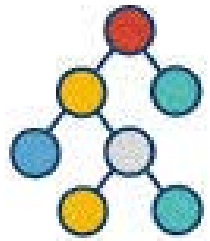


Full but not complete

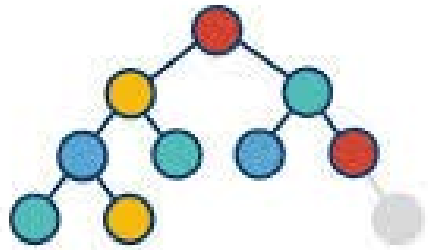


Complete and full





Full Binary Tree



Complete Binary Tree

Perfect Binary Tree

خواص درخت دودویی

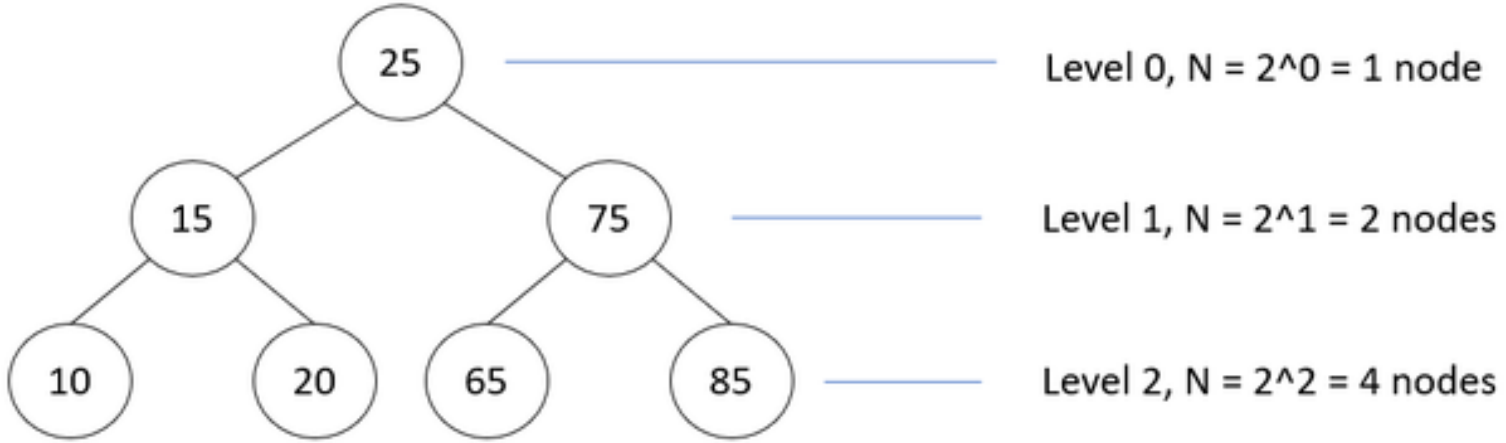
حداکثر تعداد گره ها در سطح iام: 2^i

حداکثر تعداد گره ها در درخت دودویی با ارتفاع h : 2^{h+1}-1

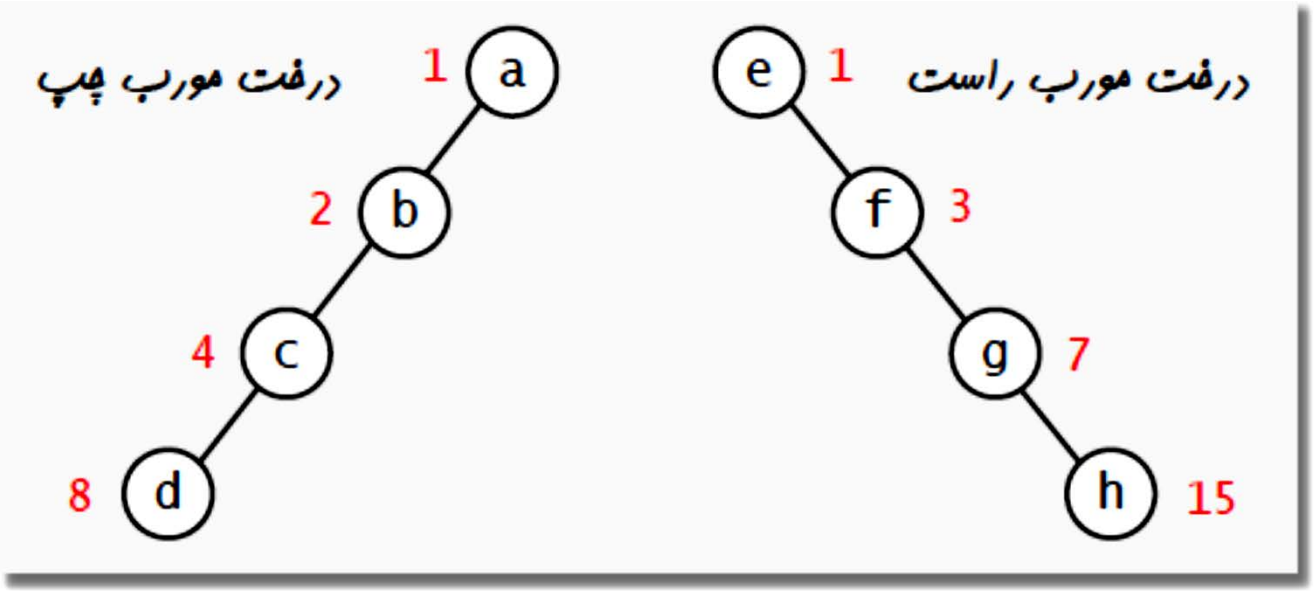
حداقل تعداد گره ها در درخت دودویی با ارتفاع h (درخت مورب): $h+1$

تعداد برگ ها (l) در درخت دودویی perfect: 2^h (بنابراین تعداد گره بر حسب تعداد برگ در این درخت : $2l$-1)

حداکثر تعداد پیوند تهی در درخت perfect با n گره: $(\frac{n+1}{2})$*2 = n+1 (بنابراین تعداد گره داخلی: $\frac{n-1}{2} = \left|\frac{n}{2}\right|$)



نمایش درخت دودویی به وسیله آرایه



- ریشه در خانه اول
- برای گره‌ای در خانه iام:
  - پدر در خانه $\lfloor i/2 \rfloor$ $(i > 1)$
  - فرزند چپ در خانه 2i
  - فرزند راست در خانه 2i+1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b		c				d							

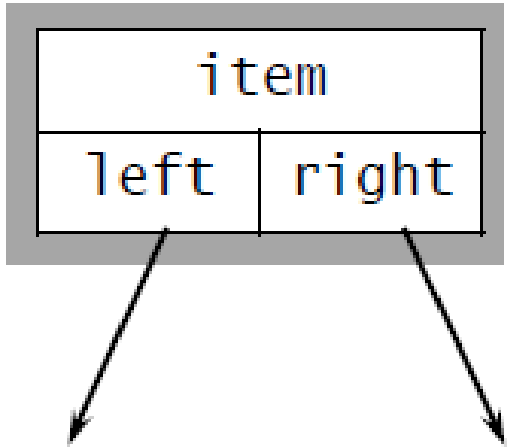
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
e		f				g								h

- عیب: اتلاف حافظه

در بدترین حالت، در درختی به ارتفاع h ، به 2^{h+1}-1خانه نیازمندیم که تنها h+1 عدد از آن حاوی داده است

پیاده سازی بهتر درخت دودویی

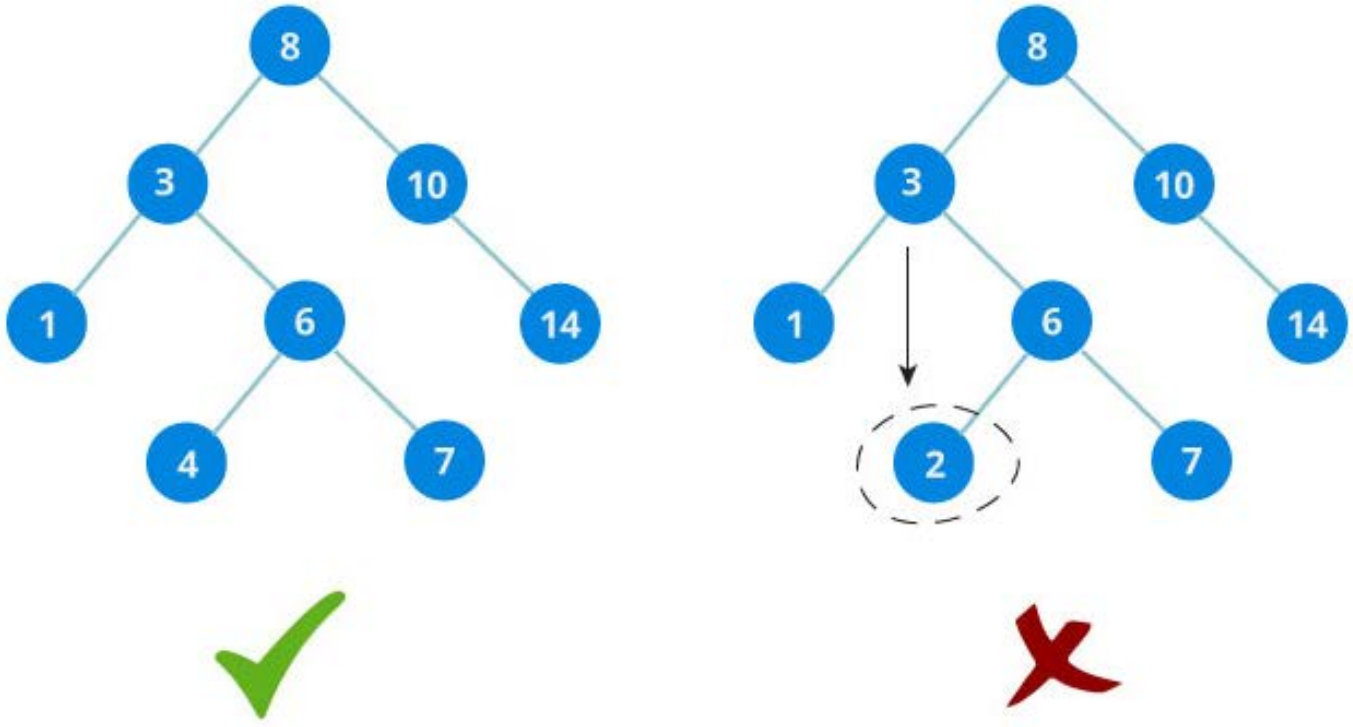
هر گره شامل داده و دو عدد اشاره گر به فرزند چپ و راست



درخت دودویی جست‌وجو
BINARY SEARCH TREE

درختی ریشه‌دار و دودویی است که به ازای هر راس مانند V، مقادیر تمامی راس های زیر درخت بچه‌ی سمت چپ آن از مقدار راسV کوچک‌تر و مقادیر تمامی راس‌های زیردرخت بچه‌ی سمت راستش از V بزرگ تر است

- تفاوت درخت دودویی جستجو با درخت دودویی ترتیب فرزندان اهمیت دارد



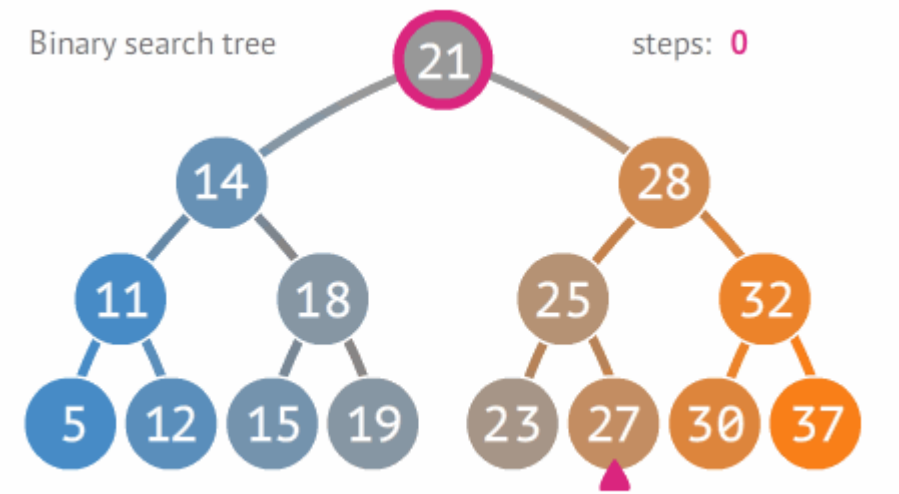
ادامه

برای کار با این ساختمان داده، ۳ عمل متصور است: (https://visualgo.net/en/bst)

- جستجو
- درج
- حذف

جستجو

- با شروع از ریشه، به ازای هر راسی که درون آن هستیم، مقداری که می‌خواهیم پیدا کنیم را با مقدار راس فعلی مقایسه می‌کنیم. در صورت مساوی بودن، عنصر مورد نظر را پیدا کرده‌ایم اما در غیر این صورت بسته به بزرگ‌تر یا کوچک‌تر بودن مقدار عنصر مورد نظر ما از برچسب راسی که داخلش هستیم، جست‌وجو را در زیردرخت سمت راست یا چپ این راس ادامه می‌دهیم.
- پیدا کردن کمترین و بیشترین عنصر : حرکت به سمت چپ و راست در درخت به ترتیب



درج

رویه بازگشتی:
با شروع از ریشه، راس vکه در حال حاضر در آن هستیم را در نظر می‌گیریم. مقداری که می‌خواهیم درج کنیم را با برچسب راس vمقایسه می‌کنیم. اگر از برچسب راس vکوچک‌تر بود، پس جایش در زیردرخت سمت چپ v است و اگر مقدارش از برچسب راس vبزرگ‌تر بود، جایش در زیردرخت سمت راست vاست. بسته به این مقایسه، به یکی از بچه‌های چپ و راست vمی‌رویم و کار را ادامه می‌دهیم. این کار وقتی پایان می‌یابد که vمساوی Noneشود. به طور مثال فرض کنید باید به زیردرخت بچه سمت راست vبرویم ولی vبچه‌ی سمت راست ندارد، در این صورت یک راس با مقدار مورد نظرمان ایجاد می‌کنیم و آن را بچه‌ی سمت راست vقرار می‌دهیم.

درج

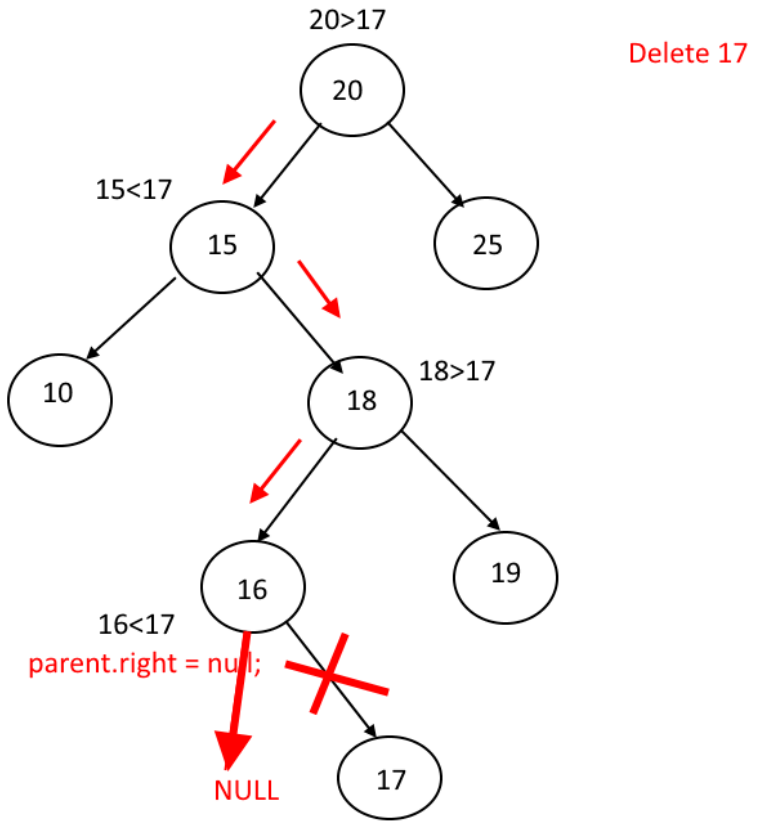
سوال

چه نوع پیمایشی، عناصر درخت را به صورت مرتب شده نمایش میدهد؟

حذف - ۱

مرحله ۳ حالت پیش می‌آید

```
       50                            50
    /     \         delete(20)      /   \
   30      70       --------->    30     70
  /  \    /  \                     \    /  \
20   40  60   80                   40  60   80
```

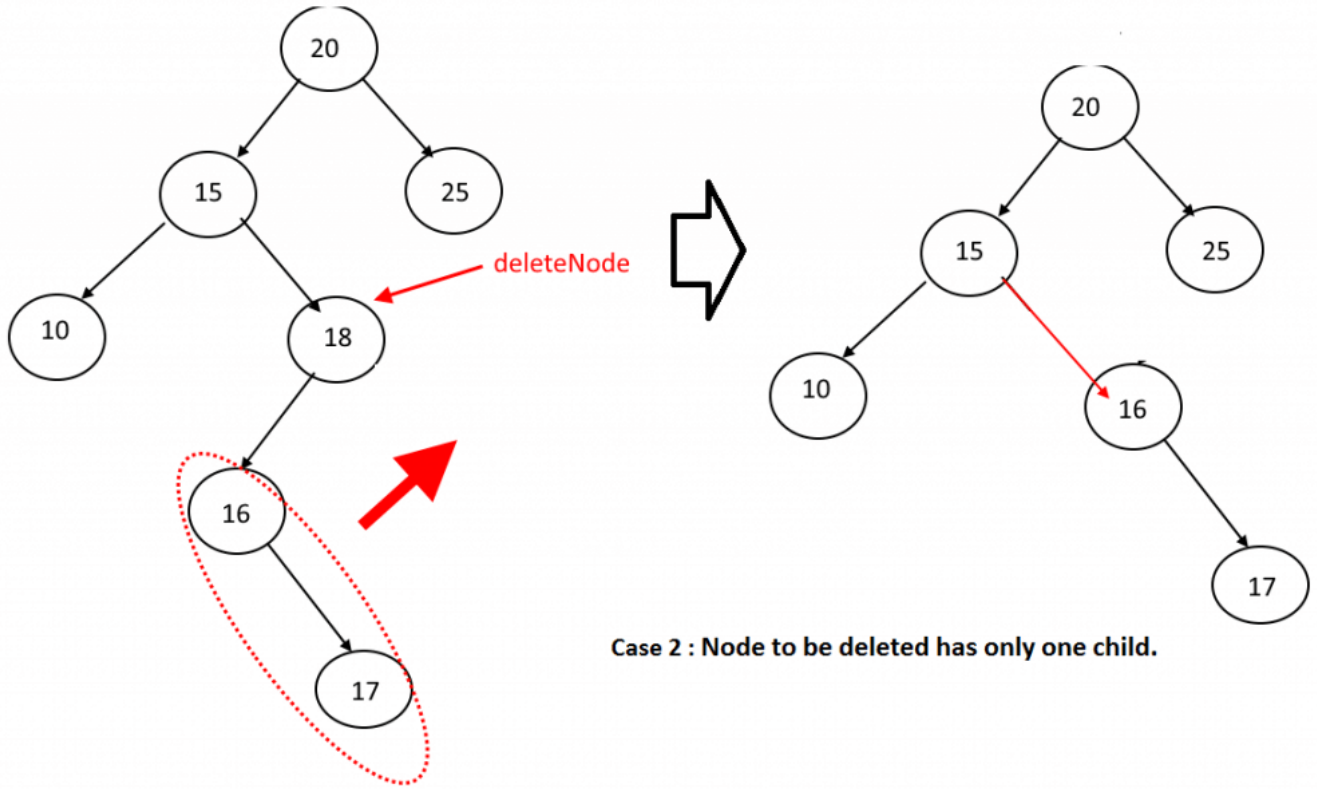


Case 1 : Node to be deleted is a leaf node (No Children).

حذف - ۲

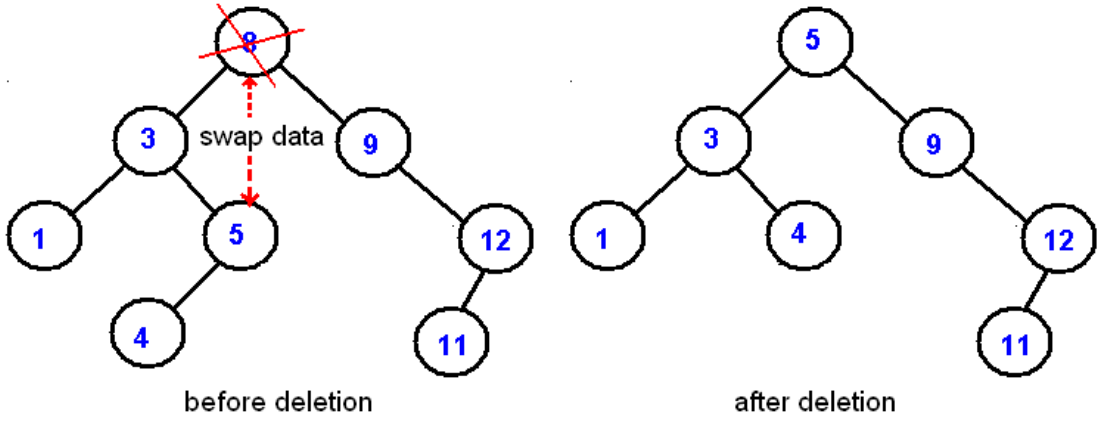
- راس v یک بچه داشته باشد: v را حذف می‌کنیم و بچه‌اش را جایش قرار می‌دهیم.

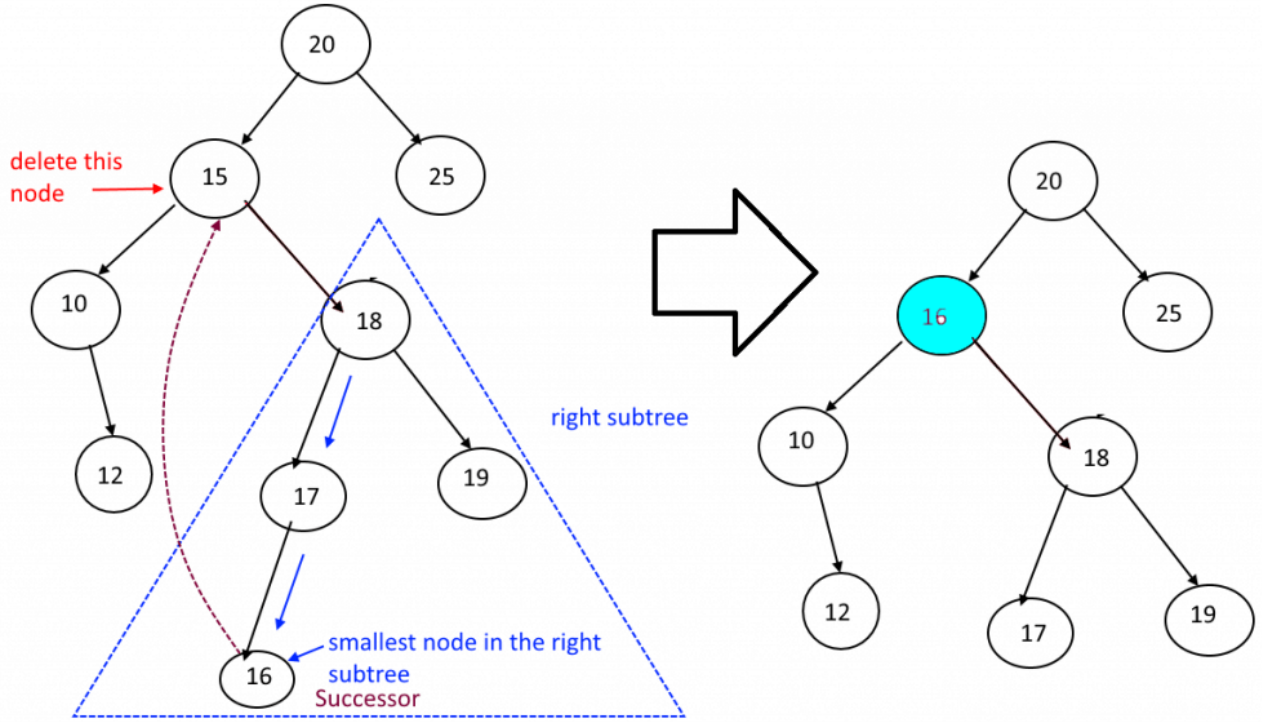
```
     50                            50
   /    \          delete(30)     /   \
  30     70        --------->    40     70
    \   /  \                          /  \
     40 60   80                      60   80
```



حذف - ۳

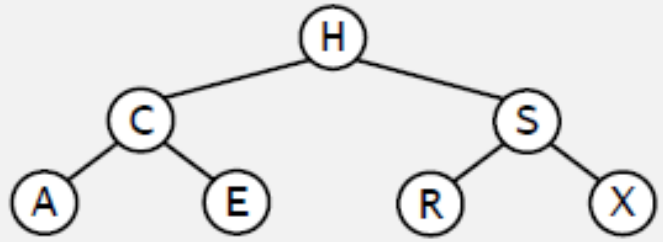
راس Vدو بچه داشته باشد: در این صورت راس مینیمم در زیر درخت سمت راست Vیا راس ماکزیمم در زیردرخت سمت چپ Vرا، به نام uرا پیدا می‌کنیم. دقت کنید هرکدام از این راس‌ها را که به جای Vقرار دهیم و
قرار است. برچسب راس uرا در Vقرار می‌دهیم و u
ز درخت است، حداکثر یک بچه خواهد داشت و



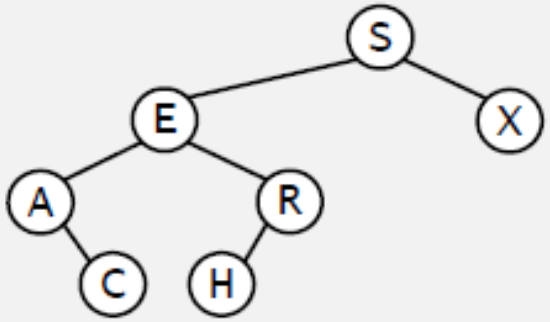


تحلیل زمان جستجوی BST

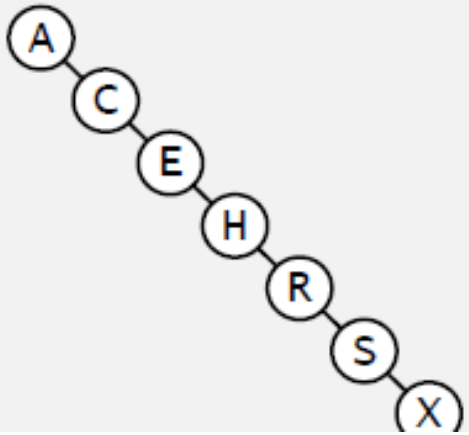
بهترین حالت



حالت متداول



بدترین حالت



حداکثر تعداد مقایسه لازم برای جستجوی کلید: h+1، پس از مرتبه O(h)

حداقل ارتفاع درخت با n گره: $\lfloor logn \rfloor$

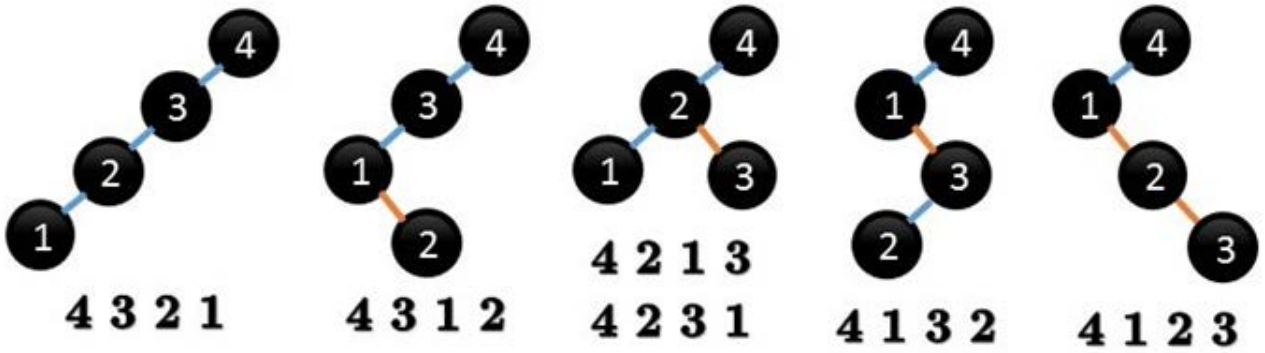
حداکثر ارتفاع درخت با n گره: n-1

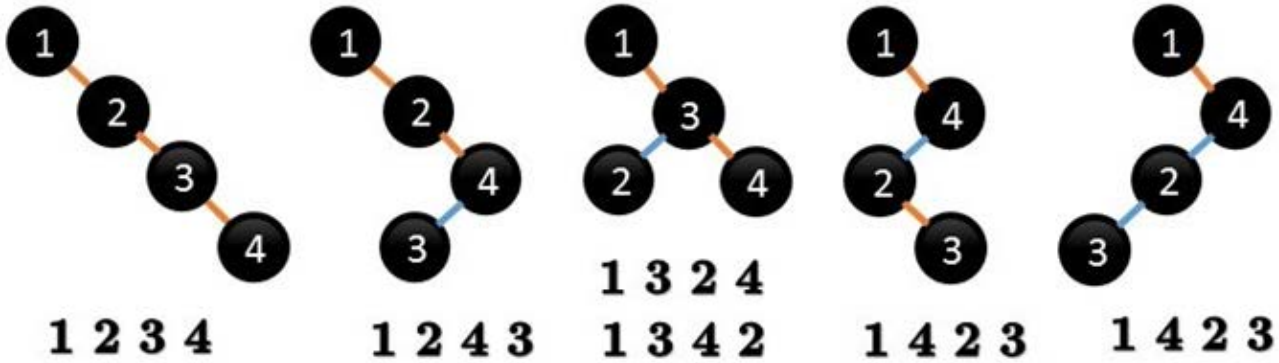
کلا عملیات درج، حذف، جستجو، یافتن کمینه و بیشینه به ارتفاع درخت وابسته است.

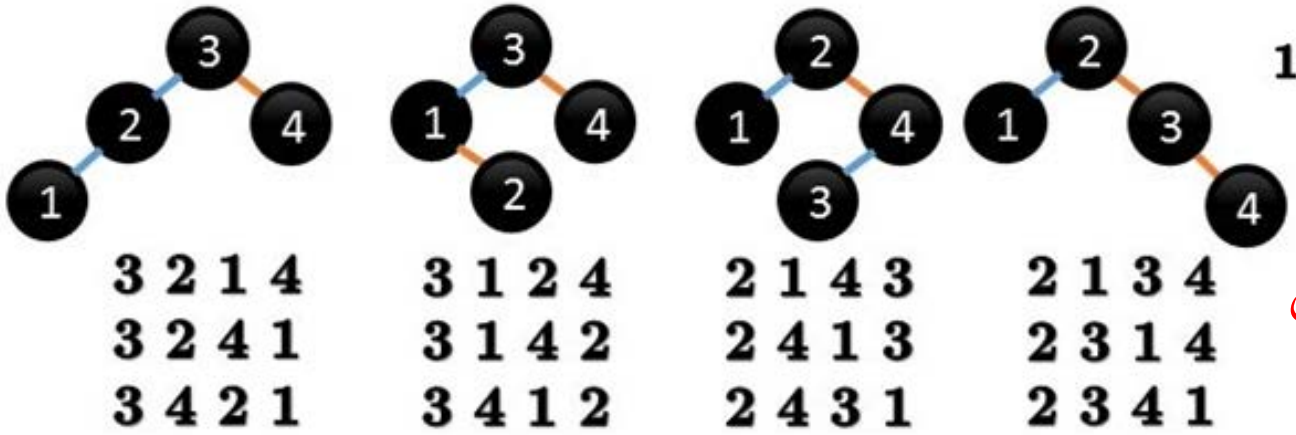
میانگین ارتفاع درخت جست و جو

- ارتفاع درخت جستجوی دودویی، به ترتیب درج عناصر بستگی دارد
- به ازای یک ترتیب درج، همیشه یک درخت جستجو به دست می آید
- در این قسمت ما به صورت شهودی میانگین ارتفاع رو تخمین میزنیم (اثبات دقیق در کتاب دکتر قدسی)

بررسی تمام جایگشت ها ی ممکن در هنگام درج اعداد ۱ تا ۴



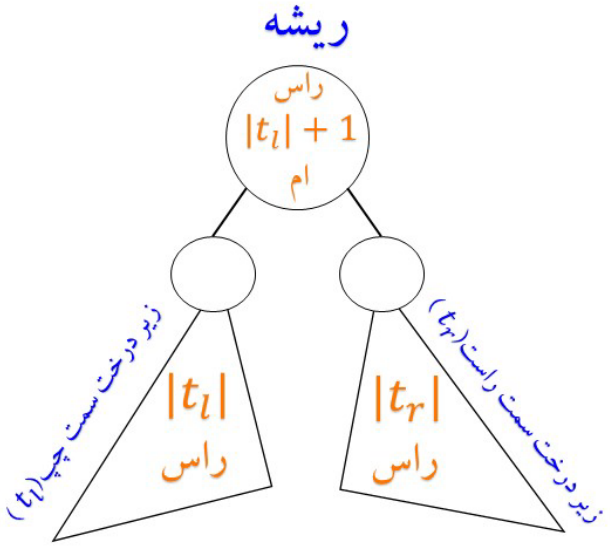




هر چه قدر د.د.ج متعادل‌تر باشد، تعداد بیشتری جایگشت با آن متناظر خواهند بود،

ادامه

رابطه زیر نشان دهنده تعداد جایگشت‌های متناظر با این درخت:
(P_t تعداد جایگشت‌های متناظر با د.د.ج t)



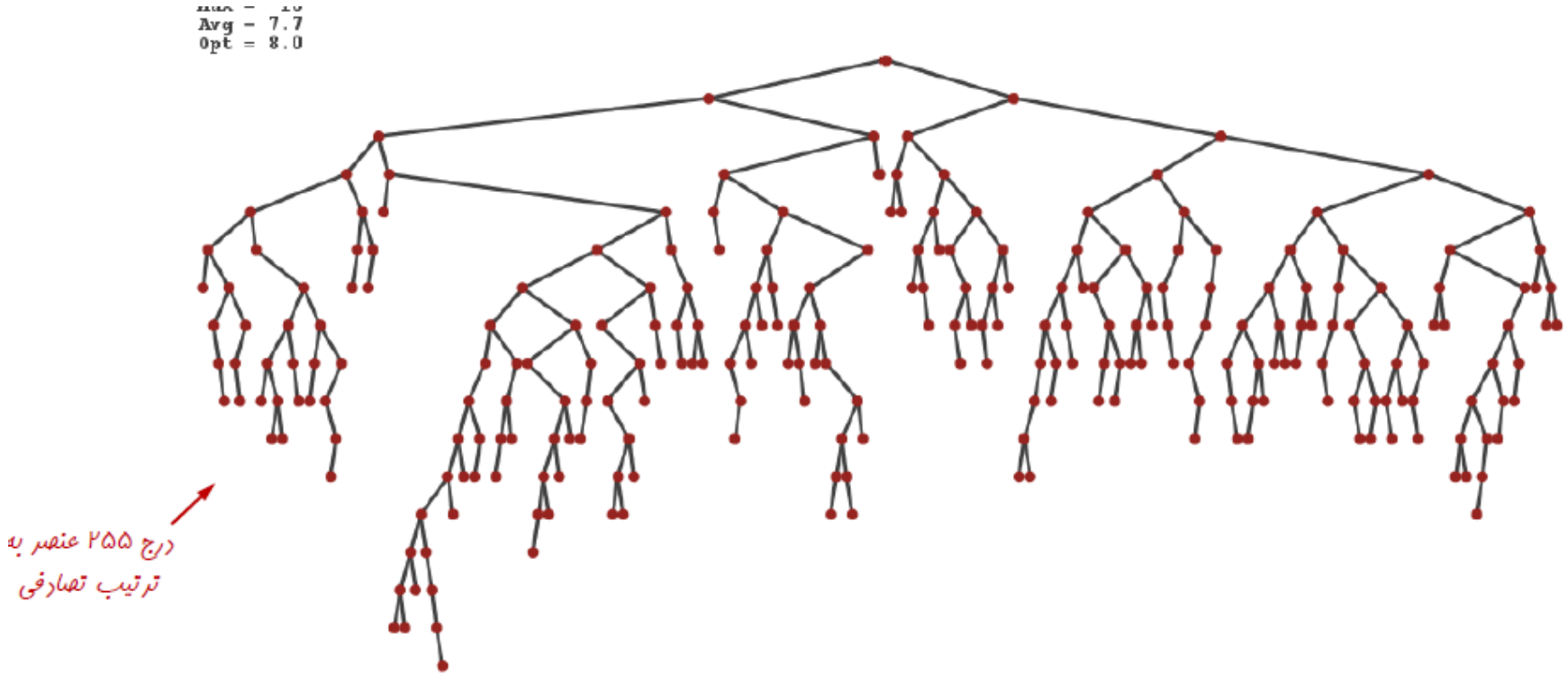
$$p_t = \underbrace{\binom{|t_l| + |t_r|}{|t_l|}} \cdot p_{t_l} \cdot p_{t_r}$$

ادامه

- حال در این فرمول،می‌توان نشان داد که هنگامی که $|t_l|$ و $|t_r|$ به هم نزدیک باشند، یعنی د.د.ج متعادل باشد، جزء انتخاب آن بسیار بسیار بزرگ تر از حالتی خواهد بود که این دو از هم فاصله داشته باشند.
- بنابراین هر چه درخت متعادل‌تر باشد، تعداد جایگشت بیشتری با آن متناظر خواهد شد.

پس به صورت شهودی در حالت میانگین، با یک درخت متعادل، که ارتفاع از $O(\log n)$ است مواجه خواهیم شد

ادامه



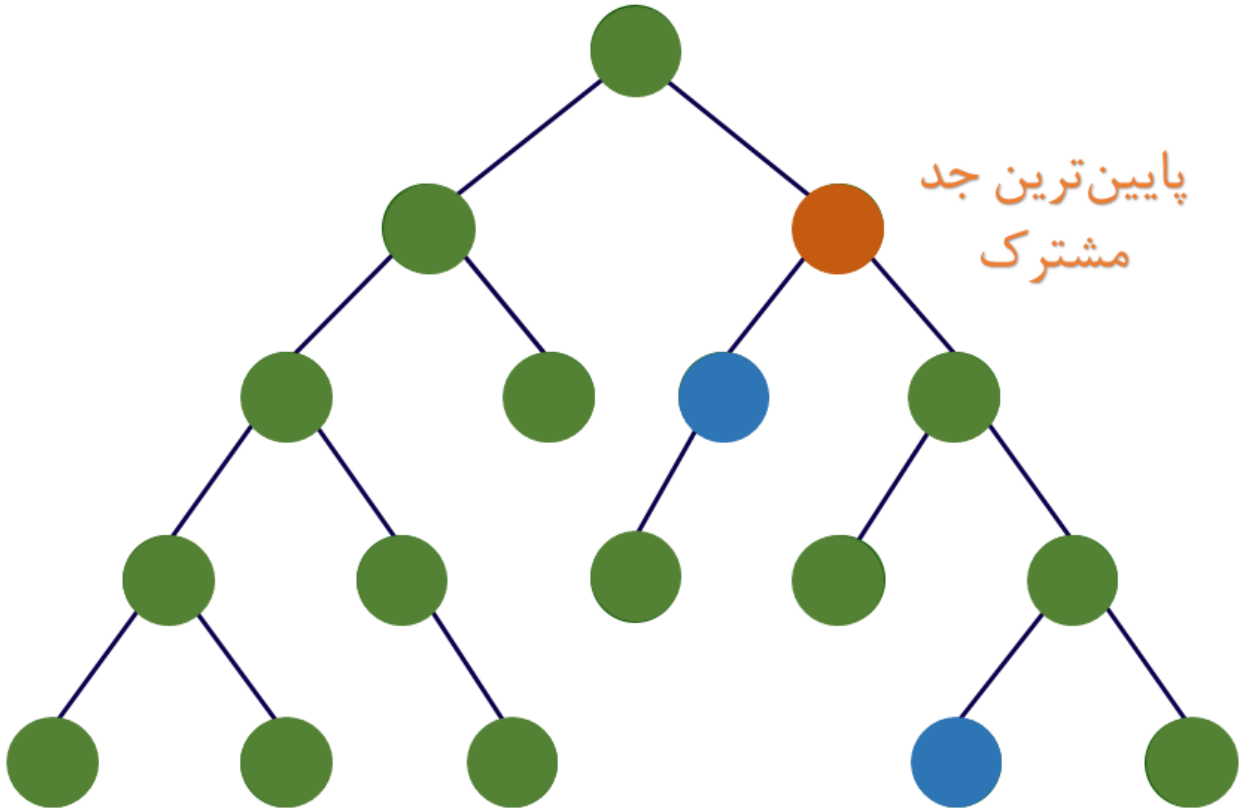
میانگین زمان جستجو در BST

- با توجه به آن چه شرح داده شد، اگر داده ها به صورت تصادفی وارد شوند، ارتفاع درخت BST ، از مرتبه logn خواهد بود

- در نتیجه ، میانگین زمان جستجو، حذف، درج در درخت BST : $\Theta(logn)$

پیدا کردن پایین ترین جد مشترک
LOWEST COMMON ANCESTOR

در یک درخت ریشه‌دار پایین ترین جد مشترک دو گره را دورترین راس از ریشه تعریف می‌کنیم که جد هر دو گره باشد.



روش حل

- از ریشه شروع کرده و مقدار آن را با دو عدد داده شده مقایسه می‌کنیم. اگر از هردو آن ها کوچکتر بود به سراغ بچه سمت راست آن می‌رویم و اگر از هردوی آن‌ها بزرگتر بود به سراغ بچه سمت چپ آن می‌رویم. این الگوریتم بازگشتی را آنقدر ادامه می‌دهیم تا به گره‌ای برسیم که مقدار آن بین دو عدد داده شده مسئله باشد.

- **تمرین: کد آن را به زبان پایتون بنویسید**