



دانشگاه شهید بهشتی کرمان

# هشت: درخت جستجو دودویی

ساختمان داده‌ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

# مسئله

ساختمان داده ای ارائه کنید که به نحوی کارا بتواند اعمال زیر را انجام دهد:

- درج عنصر : یک عنصر جدید را در مجموعه عناصرهایش اضافه کند.
- حذف عنصر : یک عنصر را که قبلاً درج شده، از مجموعه عناصرهای حذف کند.
- پرسش عنصر : برای یک عنصر بگوید که آیا در حال حاضر در مجموعه ما وجود دارد؟

# مسئله

ساختمان داده ای ارائه کند که به نحوی کلا اتماند اعمایا زیر انجام دهد:

			{1, 2, 3, 4, 5}	
ف کند.	Add	6	:	{6, 1, 2, 3, 4, 5} : درج عنصر
وجود دارد؟	Add	5	:	{6, 1, 2, 3, 4, 5} : حذف عنصر
	Ask	7	:	No : پرسش عنصر
	Ask	3	:	Yes
	Remove	2	:	{6, 1, 3, 4, 5}
	Ask	2	:	No

# Binary Search Tree

# درخت جستجو دودویی

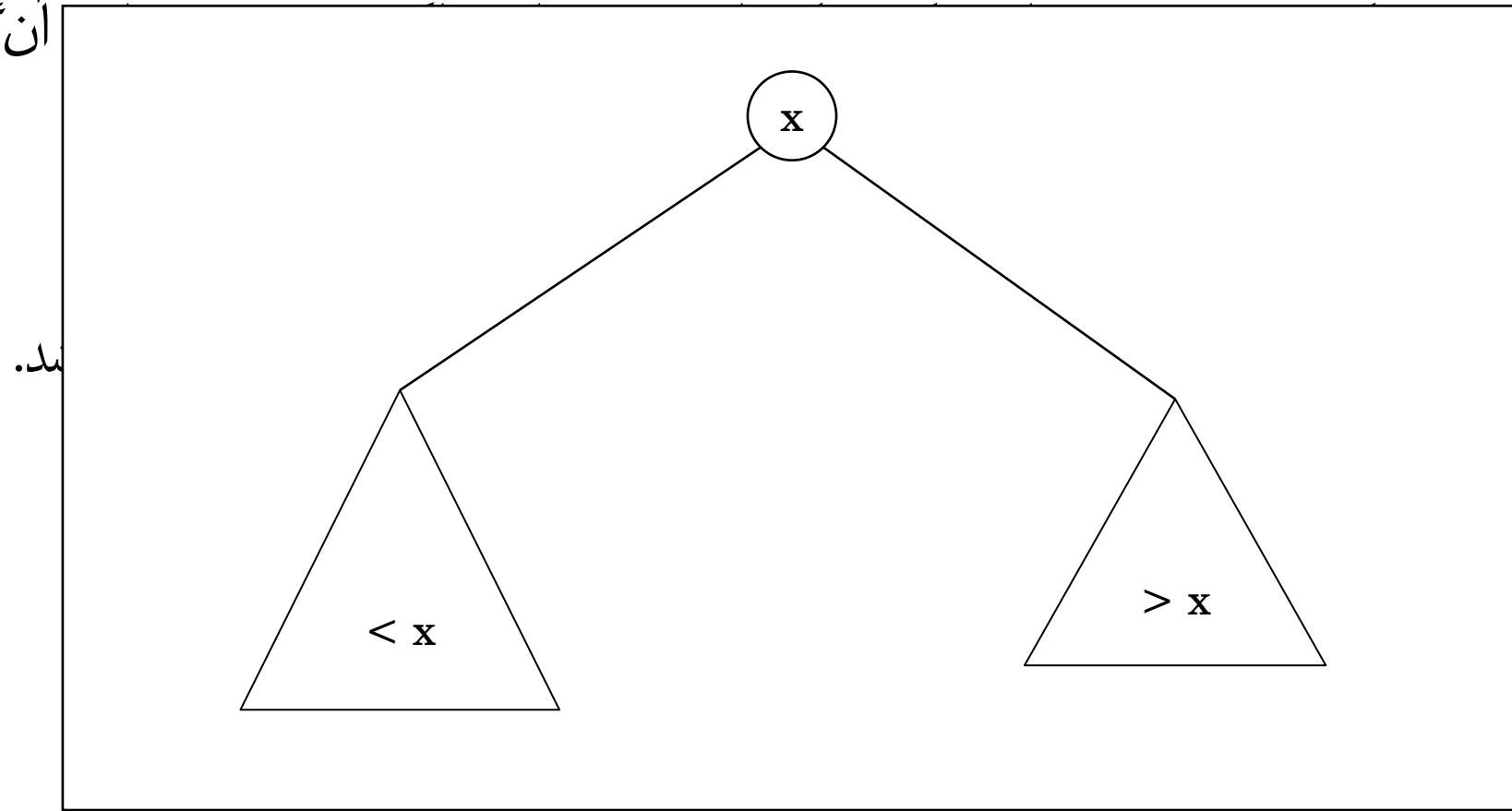
یک درخت جستجوی یک درخت دودویی است که ممکن است تهی باشد. اگر درخت تهی نباشد آن‌گاه دارای خصوصیات زیر خواهد بود:

۱. هر گره دارای کلید یکتا و منحصر به فرد بوده و دو عنصر نباید دارای کلید یکسان باشند.
۲. کلیدهای واقع در زیر درخت چپ باید کوچک‌تر از مقدار کلید واقع در ریشه زیردرخت راست باشد.
۳. کلیدهای واقع در زیردرخت راست باید بزرگ‌تر از مقدار کلید واقع در ریشه زیردرخت چپ باشد.
۴. زیردرختان چپ و راست خود نیز درختان جستجوی دودویی می‌باشند.

# Binary Search Tree

# درخت جستجو دودویی

آن گاه دارای



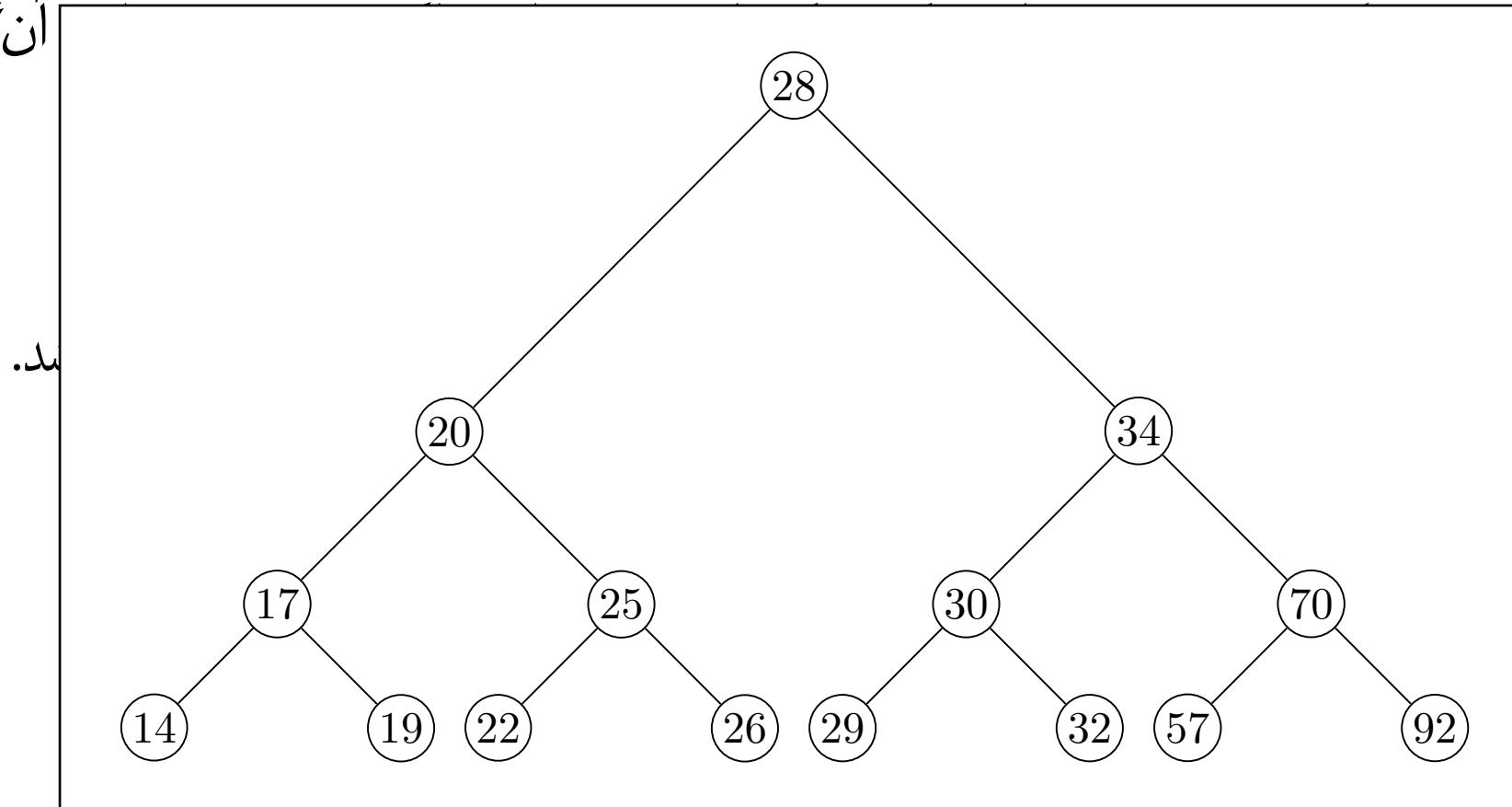
یک درخت جس  
خصوصیات زیر

۱. هر گره دارای
۲. کلیدهای واب
۳. کلیدهای واب
۴. زیردرختان ج

# Binary Search Tree

# درخت جستجو دودویی

آن گاه دارای

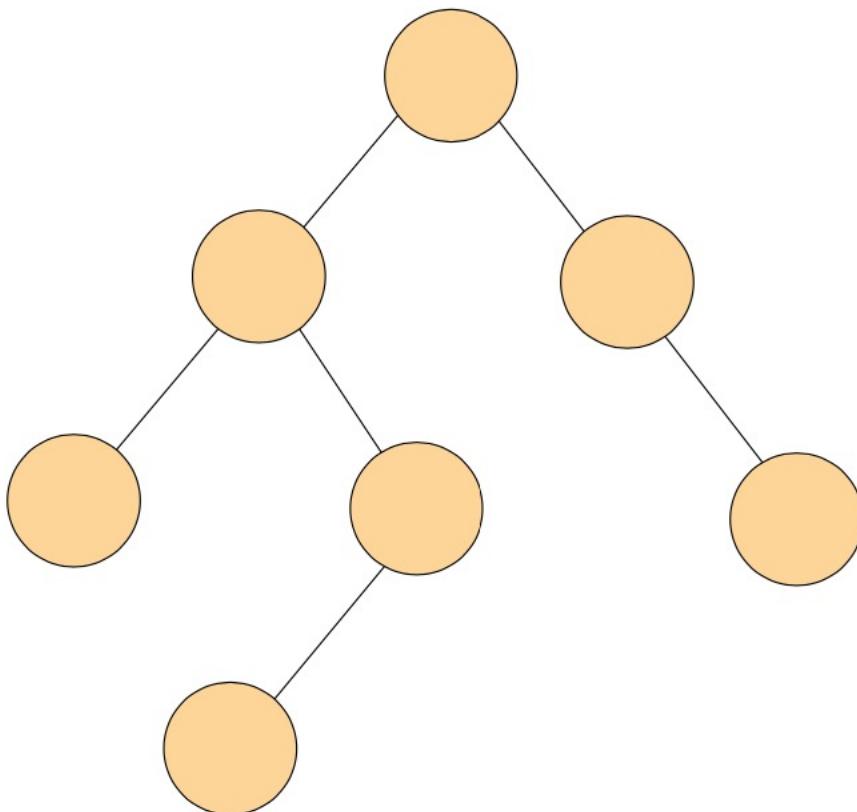


یک درخت جس  
خصوصیات زیر

۱. هر گره دارای کلیدهای وابسته
۲. کلیدهای وابسته
۳. زیردرختان جستجویی
۴. زیردرختان جستجویی

# مثال

اعداد ۱، ۸، ۲۲، ۹، ۱۴، ۱۳، ۶ را چنان به درخت زیر نسبت دهید، که درخت جستجو دودویی شود.

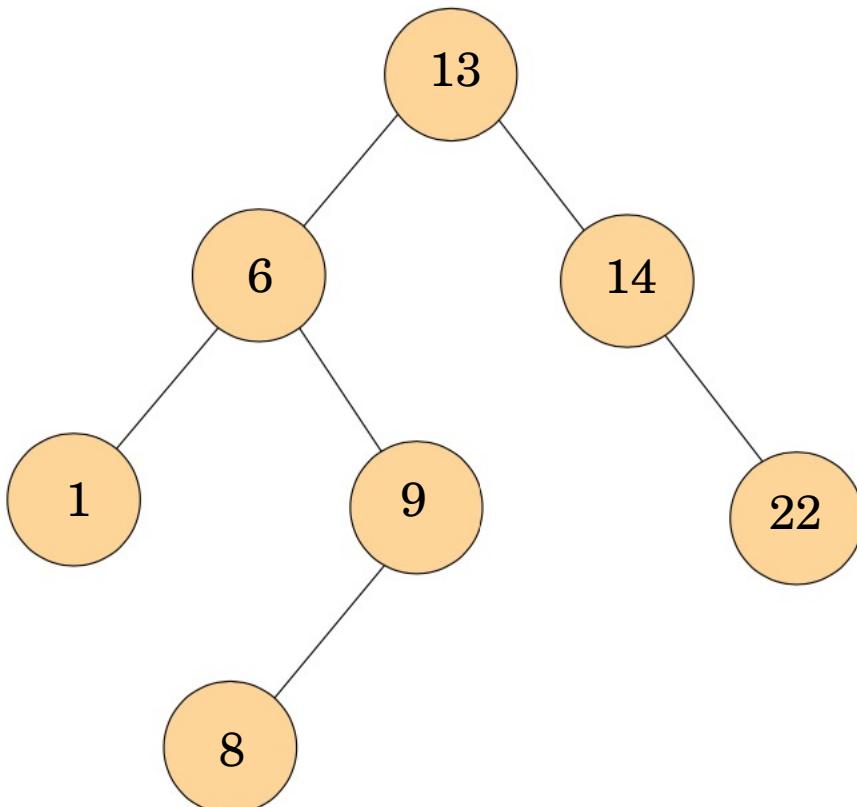


# مثال

این دنباله ها، درخت مورد نظر را می سازند:

- 13,6,9,1,14,8,22
- 13,14,6,22,1,9,8
- 13,6,1,9,8,14,22

⋮



# الگوریتم جستجو

در صورتی که  $x$  اشاره گری باشد که در ابتدا به ریشه اشاره کرده و  $key[x]$  مقدار کلید داخل آن باشد؛ همچنین  $k$  مقدار کلید مورد جستجو در درخت باشد آنگاه :

الگوریتم جستجو را از ریشه آغاز کرده و ممکن است حداکثر تا عمق ادامه پیدا کند در صورتی که مقدار کلید  $k$  در درخت وجود داشته، ( $k = key[x]$ ) شده و اشاره گر به گره مورد نظر ( $x$ ) برگردانده می‌شود؛ در غیر این صورت null بر می‌گرداند.

# الگوريتم جستجو

در صورتی که  $x$  اشاره گری باشد که در ابتدا به ریشه اشاره کرده و  $[x]$  مقدار کلید داخل آن باشد؛ همچنین  $k$  مقدار کلید مورد جستجو در درخت باشد آنگاه :

الگوريتم جستجو در صورتیکه مقدار کلید  $k$  در درخت وجود نداشته باشد، آنرا که در صورتیکه مقدار کلید  $k$  در درخت وجود نداشته باشد، آنرا که در صورت null به این صورت برمیگرداند.

```
def search(x, k):
    if(x==None or k==x.key):
        return x
    if(k<x.key):
        return search(x.left)
    else:
        return search(x.right)
```

# الگوريتم جستجو

در صورتی که  $x$  اشاره گری باشد که در ابتدا به ریشه اشاره کرده و  $[x]$  مقدار کلید داخل آن باشد؛ همچنین  $k$  مقدار کلید مورد جستجو در درخت باشد آنگاه :

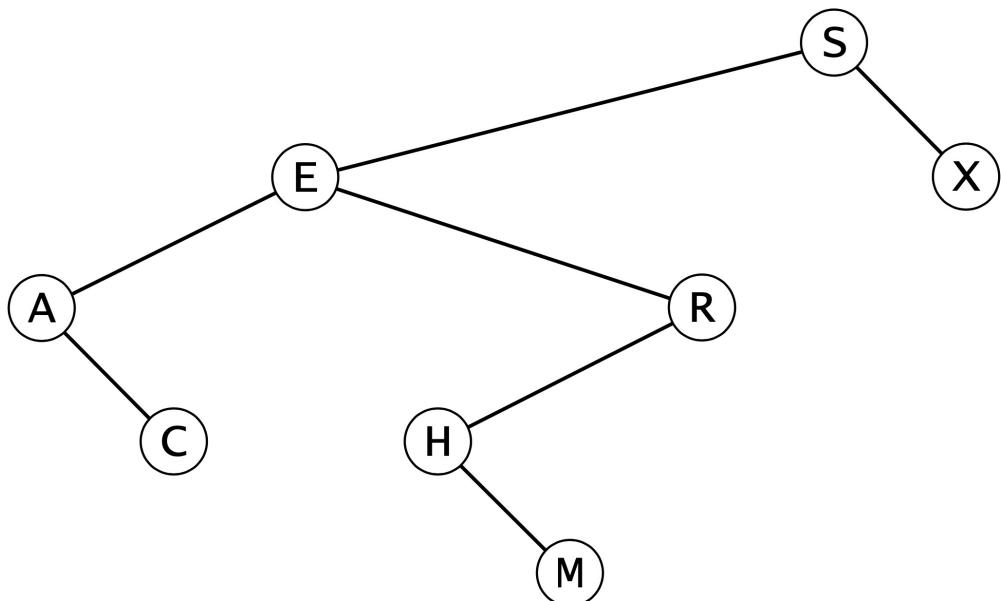
الگوريتم جستجو در صورتیکه مقدار کلید  $k$  در درخت وجود نداشته باشد آنگاه :

```
def search(x, k):
    while(x!=None and x.key!=k):
        if(k<x.key):
            x=x.left
        else:
            x=x.right
    return x
```

اين صورت null بودن در درخت وجود نداشته باشد آنگاه :

# جستجو

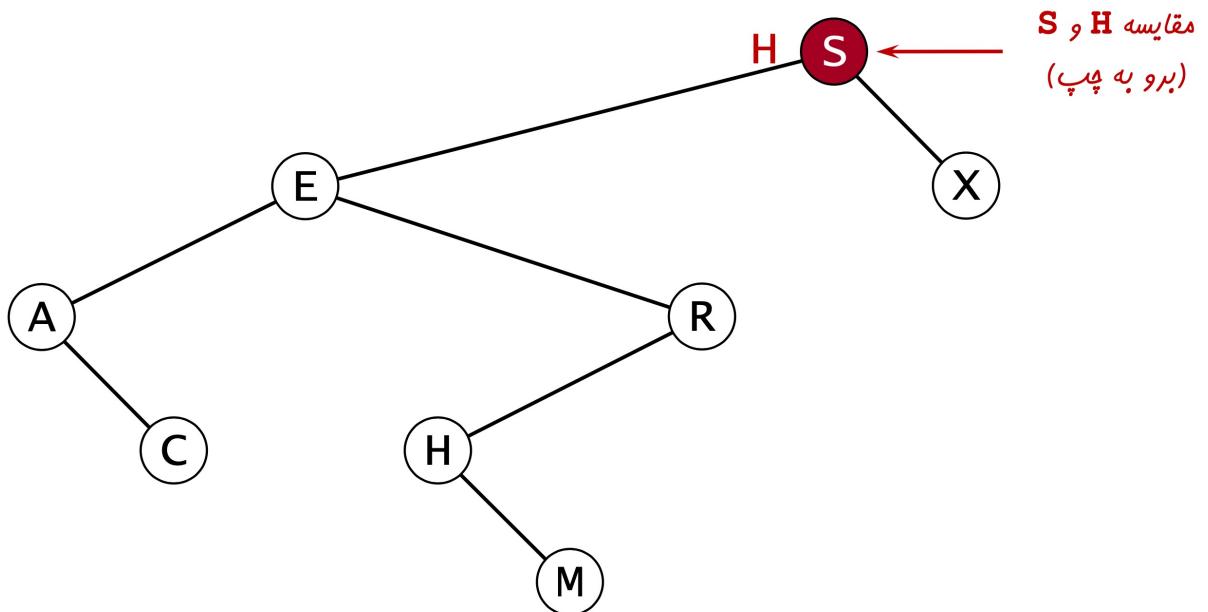
اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



جستجو موفق برای کلید H

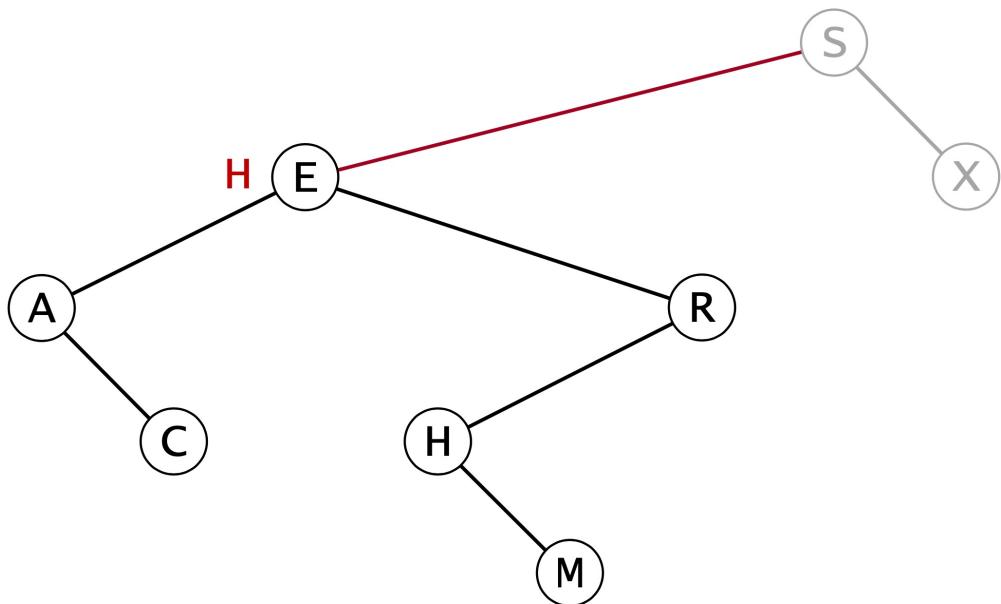
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



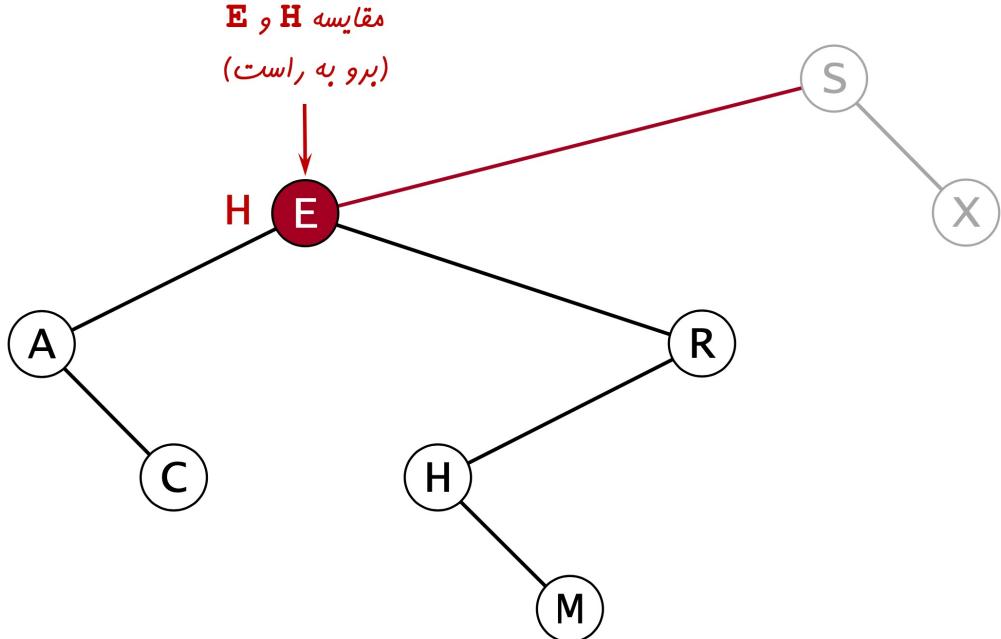
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



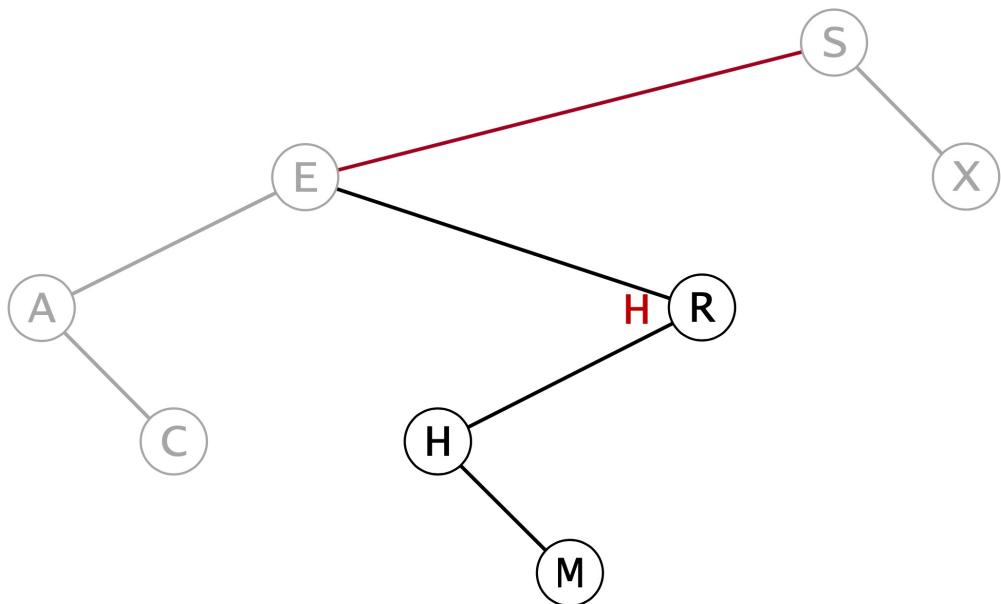
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



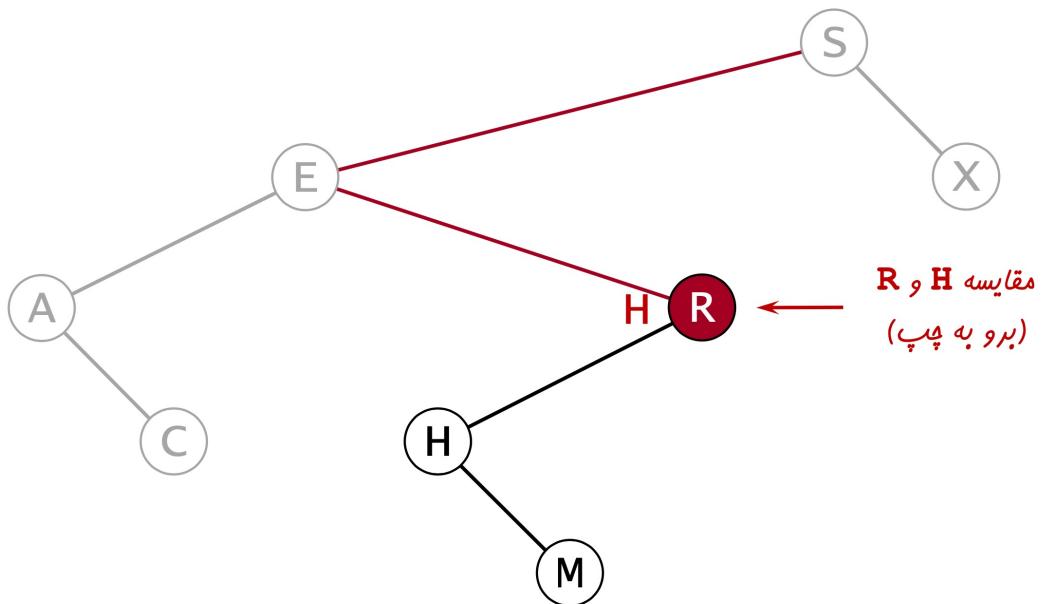
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



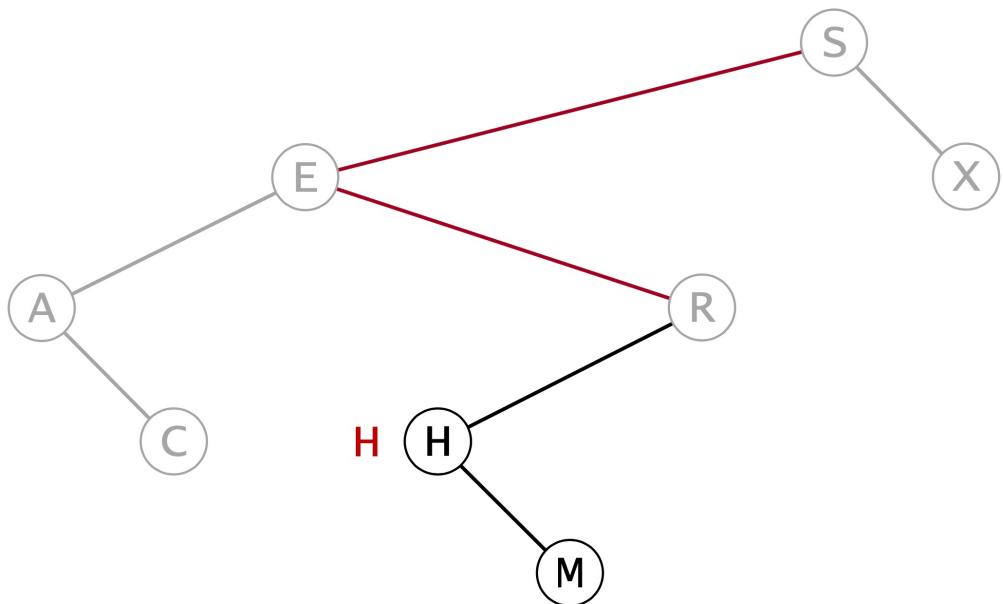
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



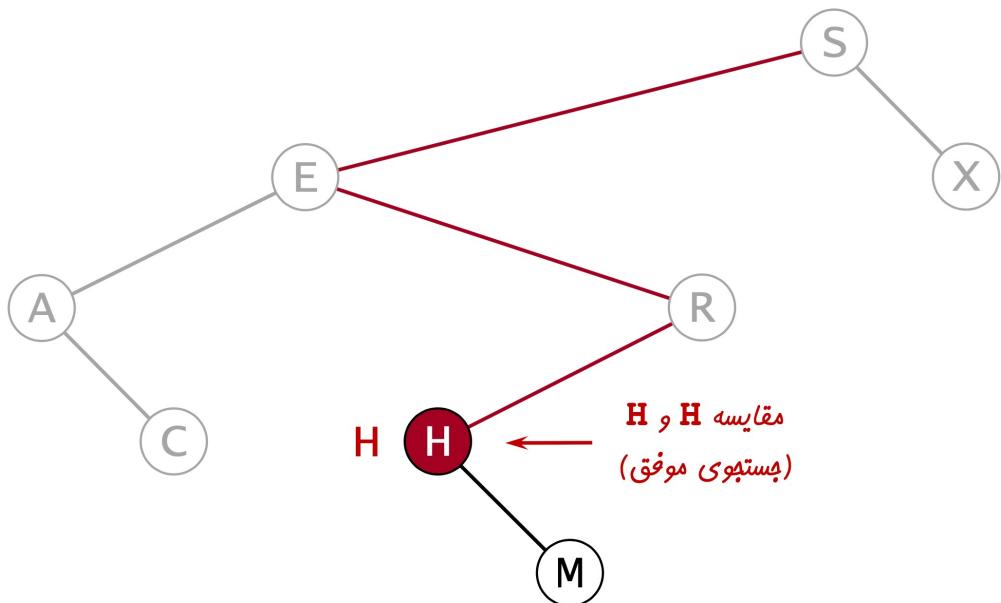
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



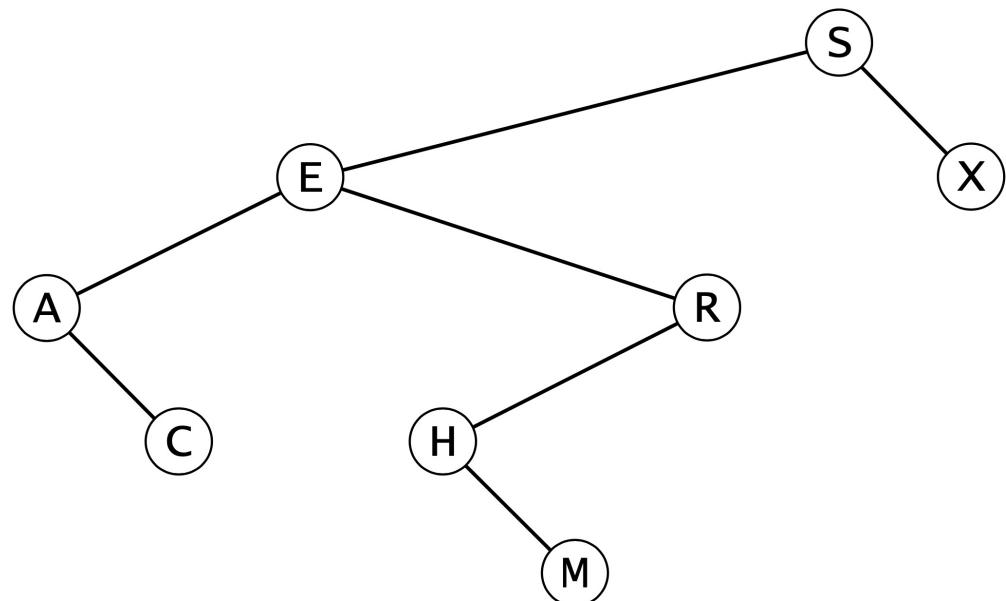
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



# جستجو

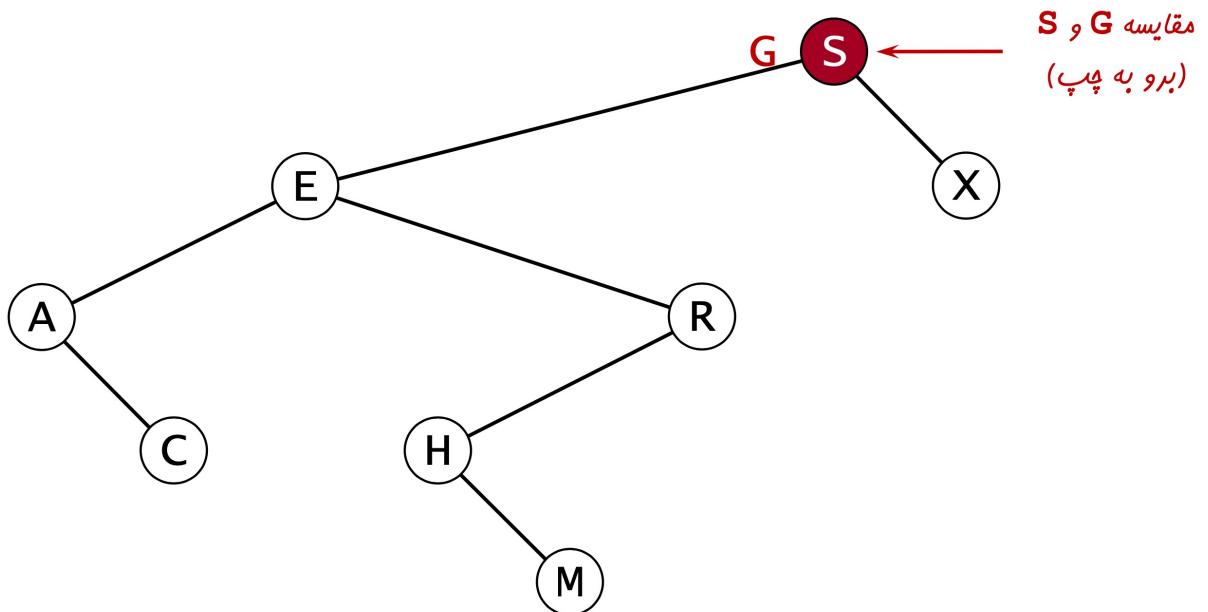
اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



جستجو ناموفق برای کلید G

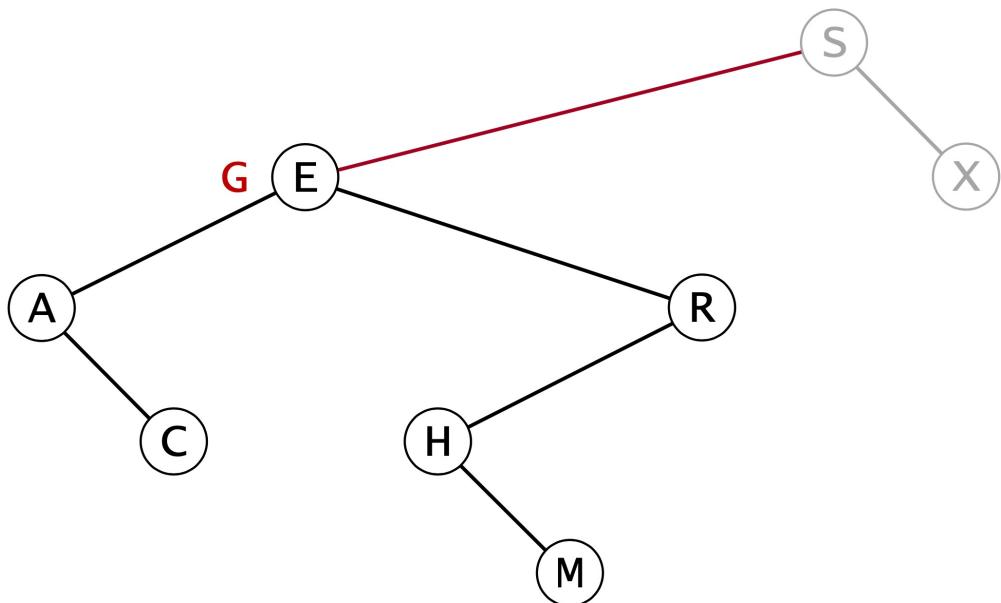
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



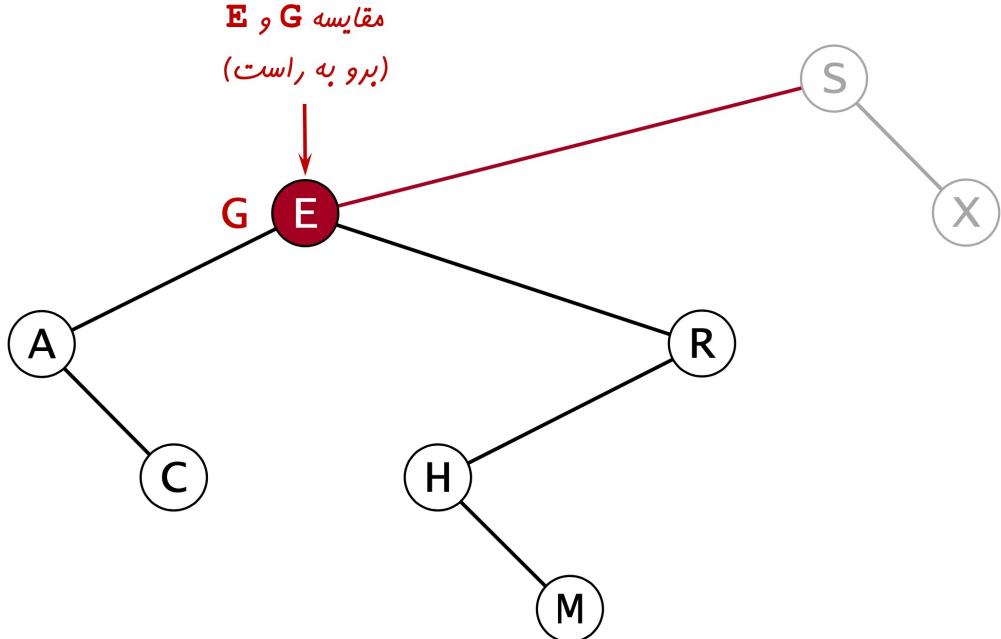
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



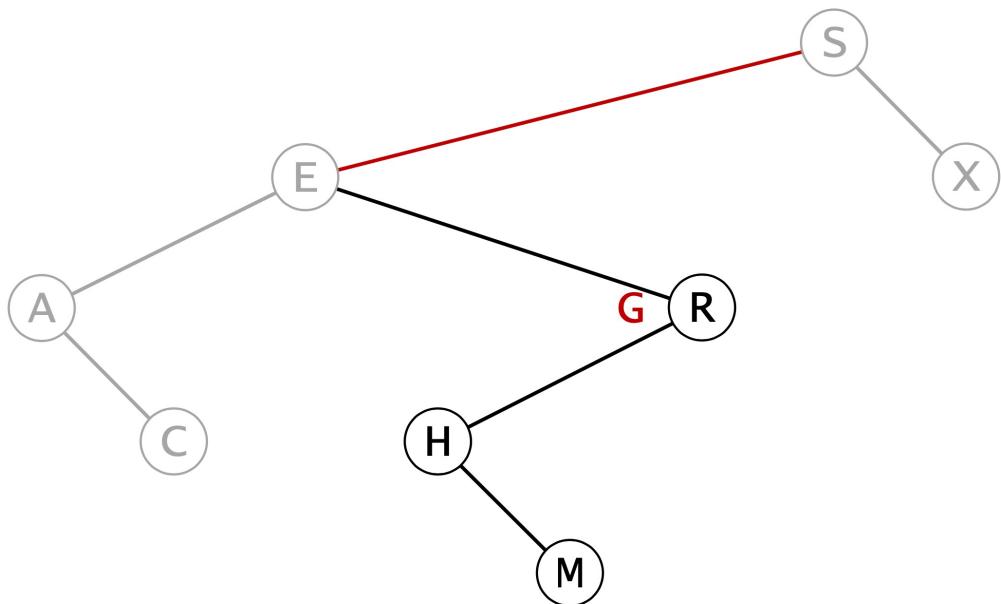
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



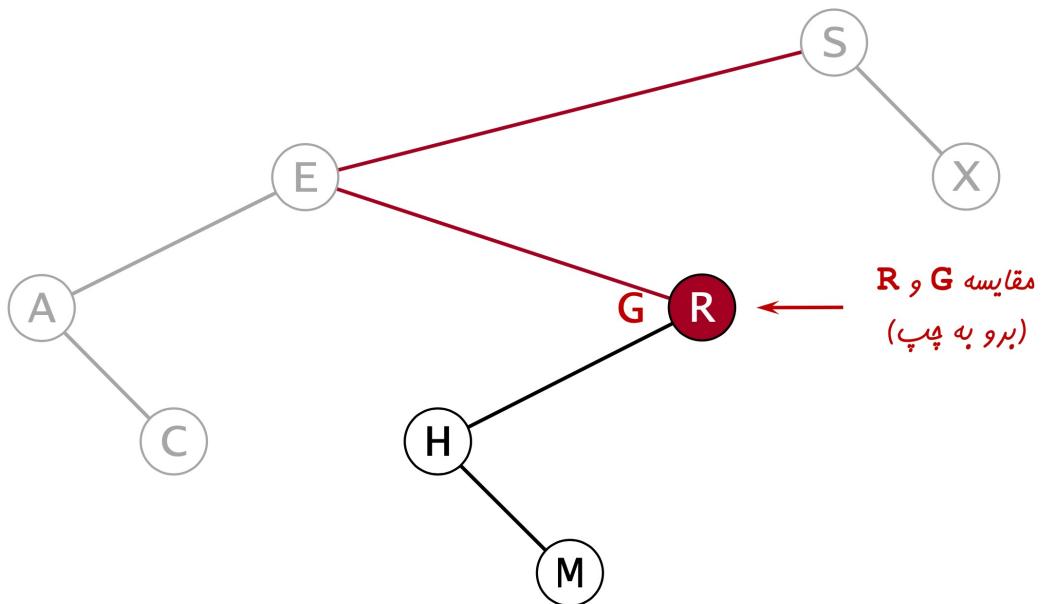
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



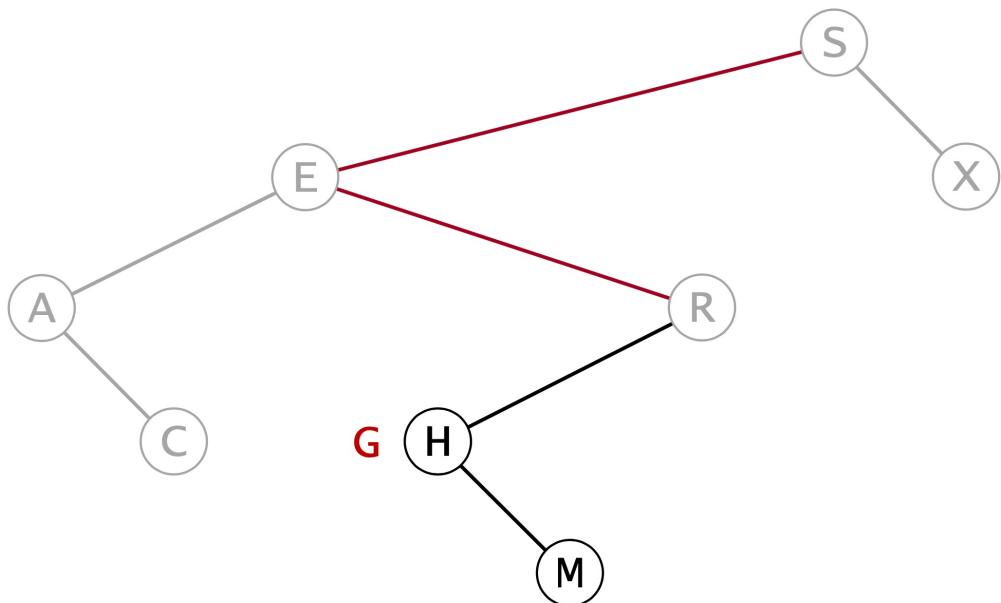
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



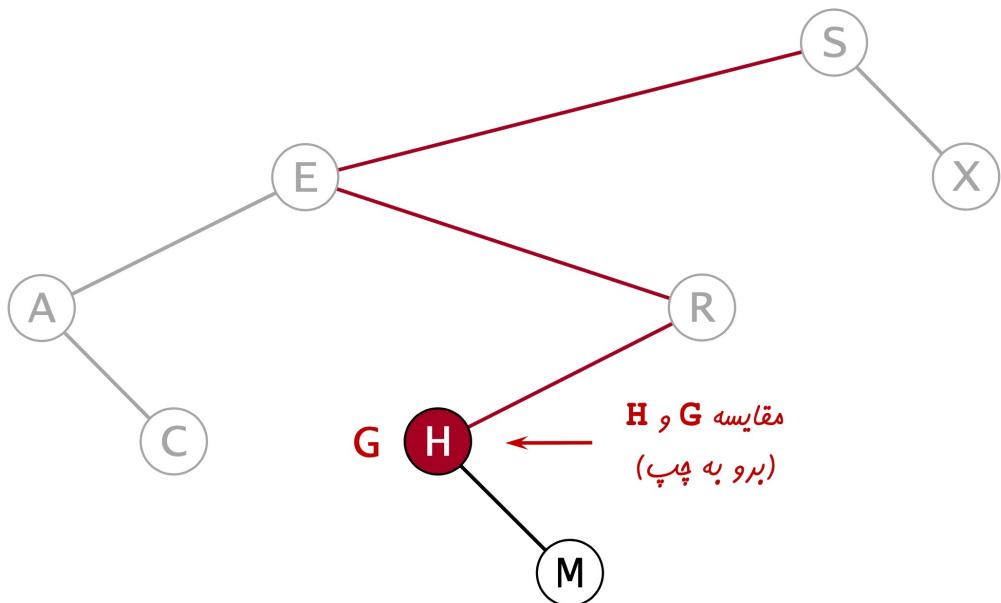
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



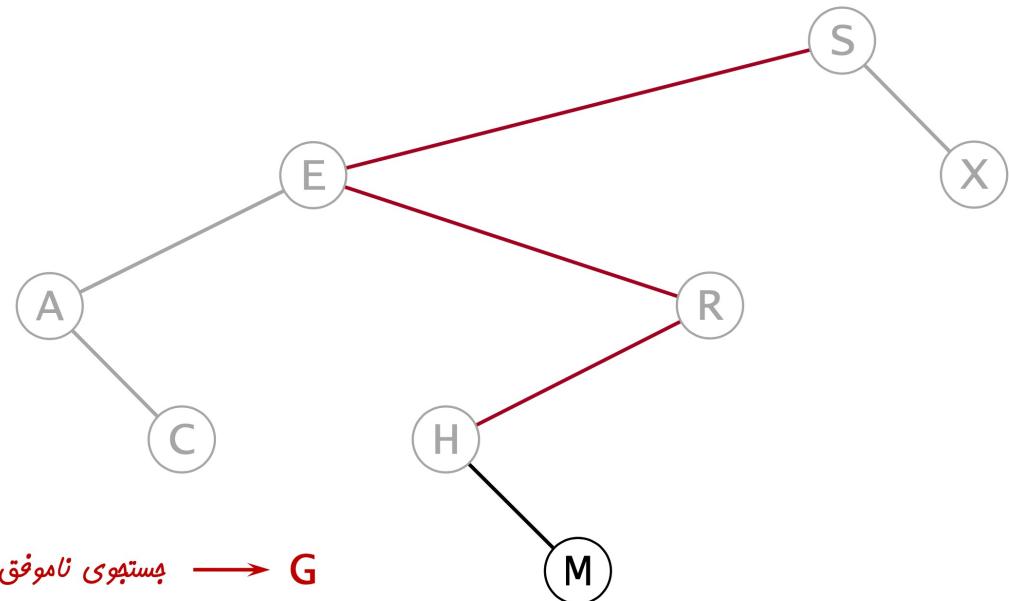
# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



# جستجو

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



# تحليل الگوریتم جستجو

از آنجا که در هر مرحله از جستجو به گره‌ای از دقیقاً یک سطح پایین‌تر می‌رویم، زمان اجرا الگوریتم جستجو برای هر کلید دلخواه حداقل برابر با  $O(h)$  است.

- حداقل مقایسه : 1

- حداکثر مقایسه: h

- متوسط مقایسه:  $O(h) = \frac{h+1}{2}$

# مثال

فرض کنید اعداد ۱ تا ۱۰۰ در یک درخت دودویی جستجو ذخیره شده‌اند. و ما می‌خواهیم عدد ۳۶۳ را پیدا کنیم. کدام‌یک از ترتیب‌های زیر (از چپ به راست) نمی‌تواند ترتیب دسترسی به عناصر درخت در این جستجو باشد؟

$$925, 202, 911, 240, 912, 245, 363 \quad (1)$$

$$924, 220, 911, 244, 898, 258, 362, 363 \quad (2)$$

$$2, 252, 401, 398, 330, 344, 397, 363 \quad (3)$$

$$2, 399, 387, 219, 266, 382, 381, 278, 363 \quad (4)$$

# مثال

فرض کنید اعداد ۱ تا ۱۰۰ در یک درخت دودویی جستجو ذخیره شده‌اند. و ما می‌خواهیم عدد ۳۶۳ را پیدا کنیم. کدام‌یک از ترتیب‌های زیر (از چپ به راست) نمی‌تواند ترتیب دسترسی به عناصر درخت در این جستجو باشد؟

925 , 202 , 911 , 240 , 912 , 245 , 363      (۱)

گزینه ۱ درست است.      (۲)

در شرایطی که بعد از ۹۱۱ به سمت چپ ۲۴۰ برویم انتظار داریم تمام مقادیر بعدی کوچک‌تر از ۹۱۱ باشد اما در ادامه ۹۱۲ بزرگ‌تر از ۹۱۱ است و این غیر ممکن است.      (۳)

363      (۴)

363      (۵)

363      (۶)

# جستجو بیشینه و کمینه

کمینه:

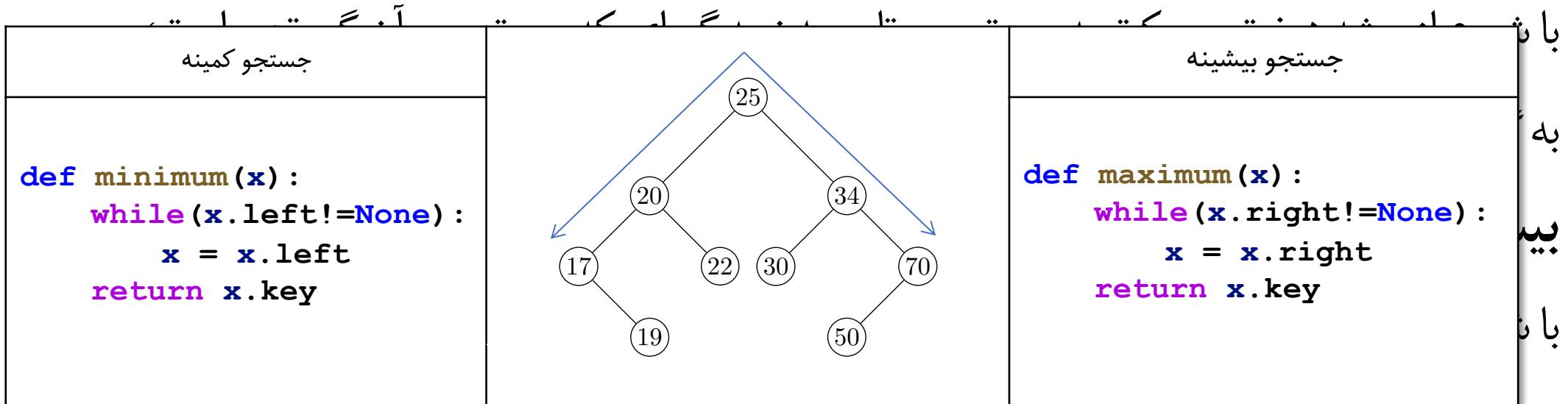
با شروع از ریشه درخت و حرکت به سمت چپ تا رسیدن به گرهای که سمت چپ آن گره تهی است؛  
به گرهای می‌رسیم که مقدار کلید آن در درخت مینیمم است.

بیشینه:

با شروع از ریشه درخت و حرکت به سمت راست تا رسیدن به گرهای که سمت راست آن گره تهی است؛  
به گرهای می‌رسیم که مقدار کلید آن در درخت مینیمم است.

# جستجو بیشینه و کمینه

کمینه:



به گره‌ای می‌رسیم که مقدار کلید آن در درخت مینیمم است.

# الگوریتم درج

برای عمل درج یک کلید دلخواه در درخت جستجو ابتدا با فراخوانی عملیات جستجو باید مطمئن شویم کلید مورد نظر در درخت وجود ندارد (درخت جستجو کلید تکراری ندارد)، بنابراین بصورت زیر عمل می‌کنیم:

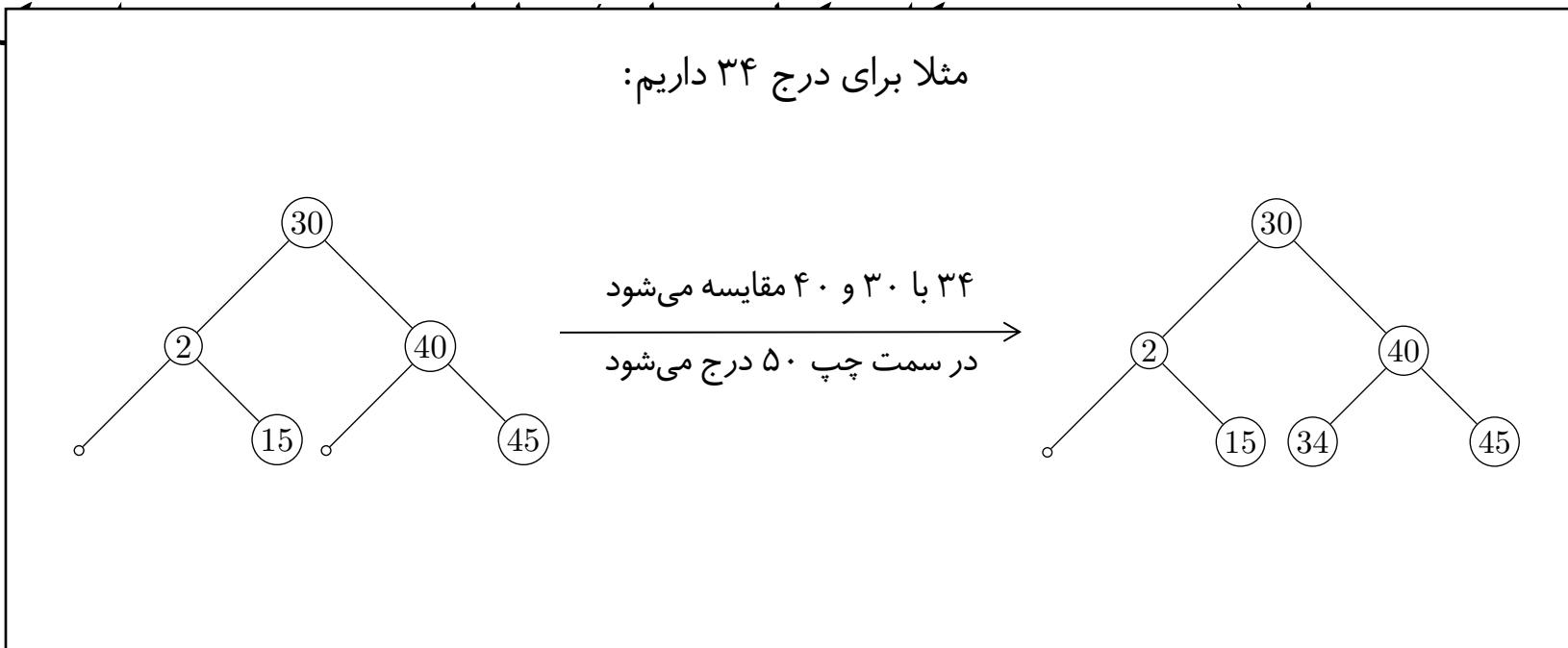
- جستجو در درخت یافتن موقعیت درج (در زمان  $(O(h))$ )
- درج گره با کلید موره نظر در موقعیتی که اشاره‌گر جستجو null شده است (در زمان  $(O(1))$ )

بنابراین در BST همواره یک گره در برگ درج می‌شود.

# الگوریتم درج

برای عمل درج یک کلید دلخواه در درخت جستجو ابتدا با فراخوانی عملیات جستجو باید مطمئن شویم کلید مورد نظر در درخت وجود دارد.

مثلا برای درج  $34$  داریم:



- جستجو در درخت
- درج گره با همواره یک گره در برگ درج می‌شود.

بنابراین در BST همواره یک گره در برگ درج می‌شود.

# الگوریتم درج

برای عمل درج یک کلید دلخواه در درخت جستجو ابتدا با فراخوانی عملیات جستجو باید مطمئن شویم کلید مورد نظر در درخت وجود ندارد (درخت جستجو کلید تکراری ندارد)، بنابراین بصورت زیر عمل می‌کنیم:

( O(1) ) در زمان

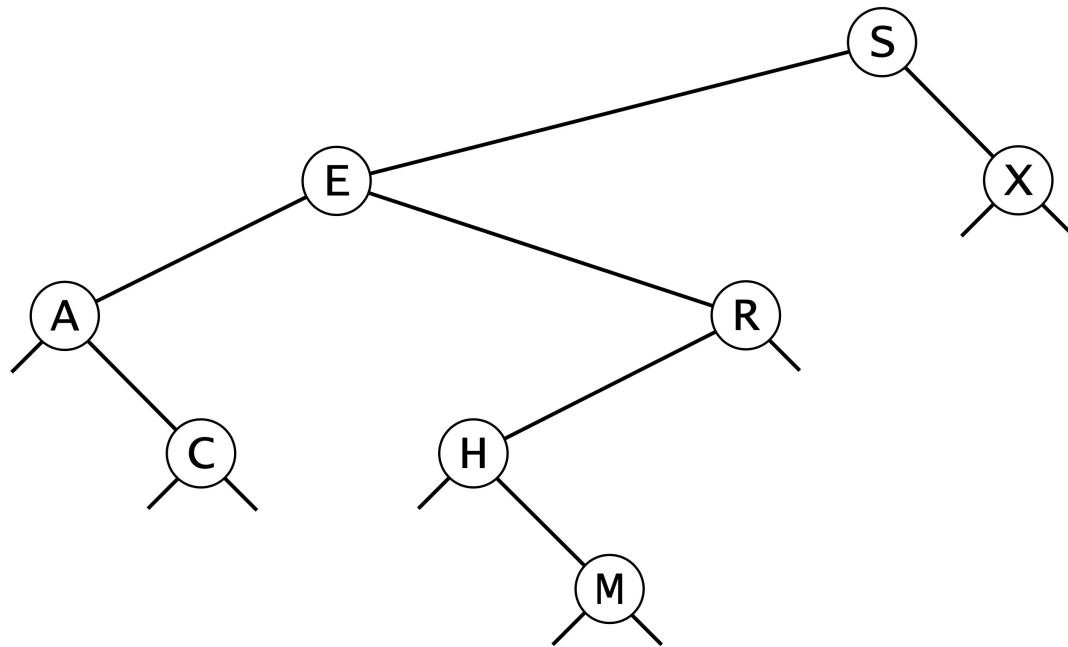
```
def insert(value):
    x = search(value)
    if(value == x):
        return
    if(value > x):
        x.right = value
    if(value < x):
        x.left = value
```

- جستجو در درخت یافتن موقعیت
- درج گره با کلید موره نظر در مو

بنابراین در BST همواره یک گره در برگ درج می‌شود.

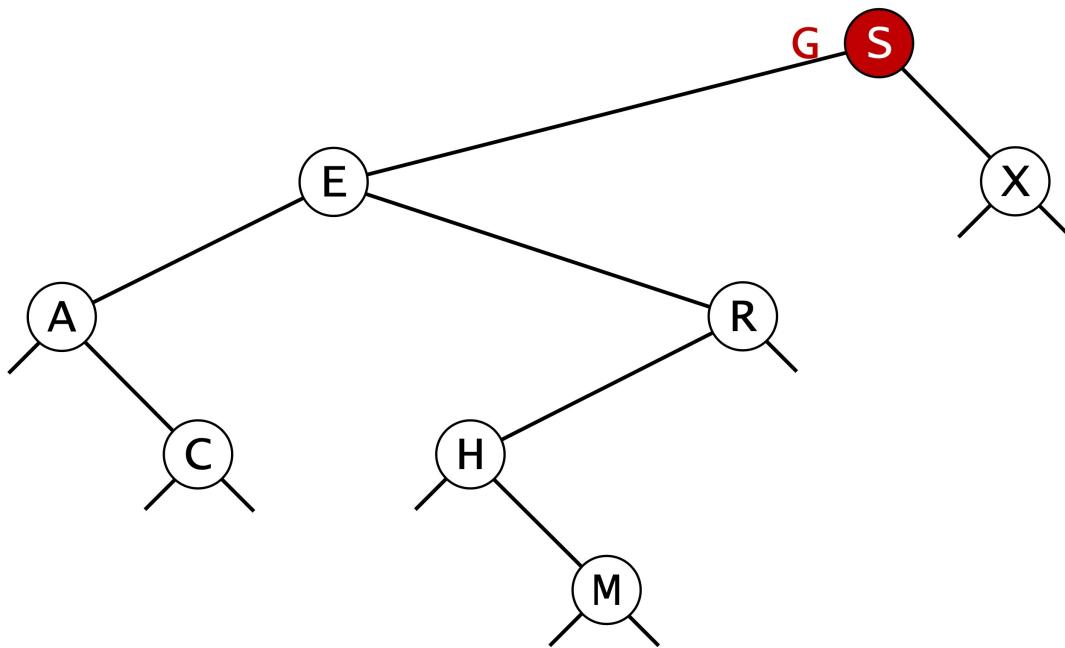
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



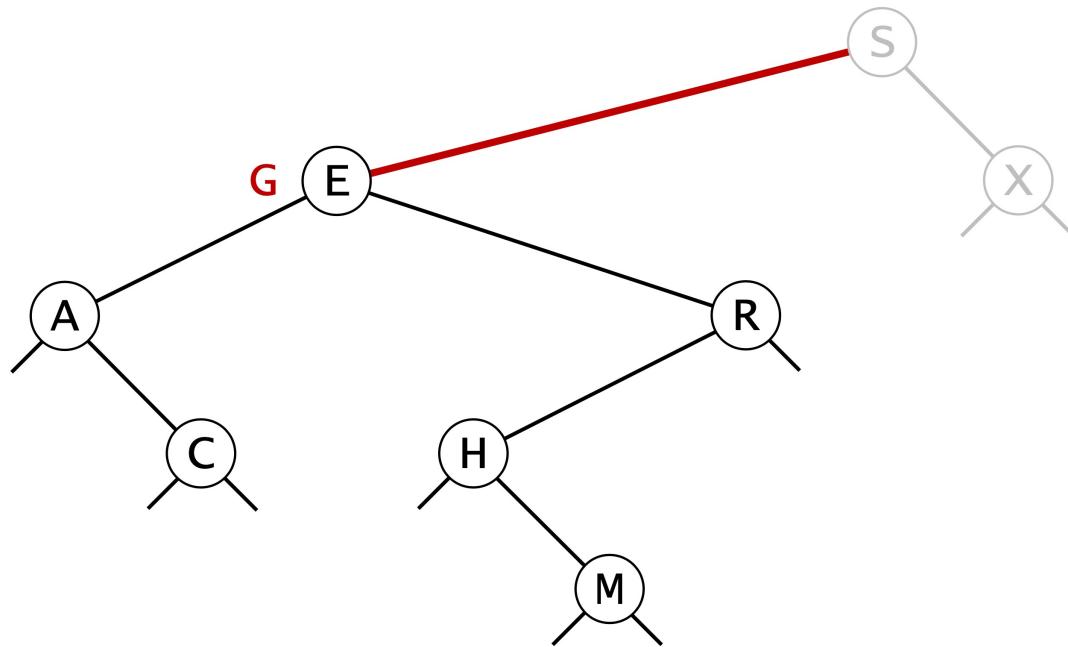
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



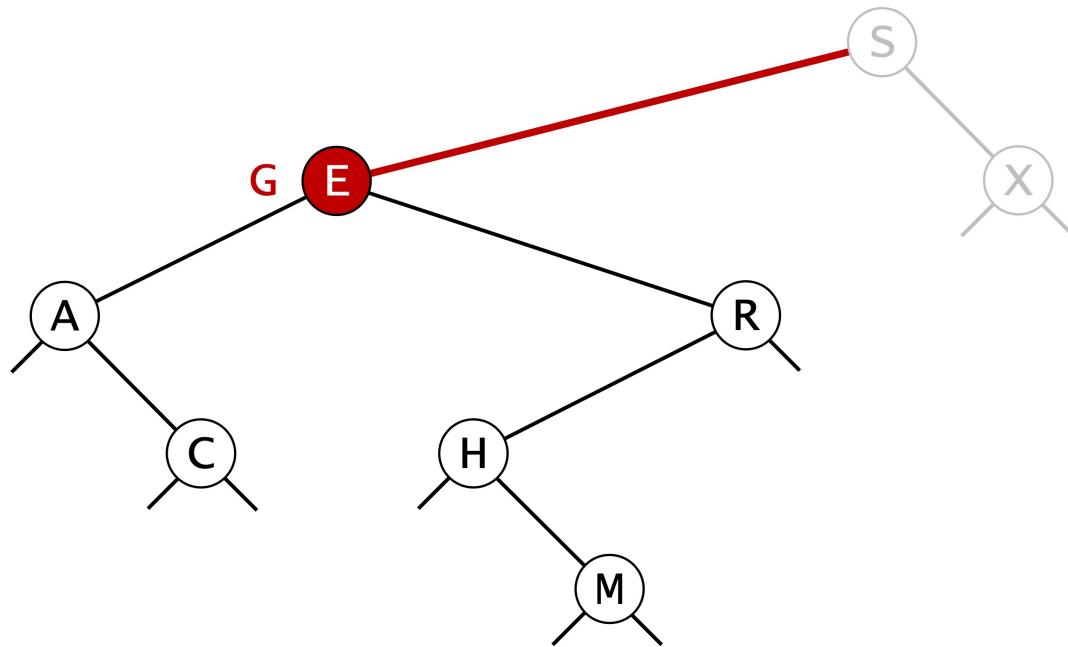
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



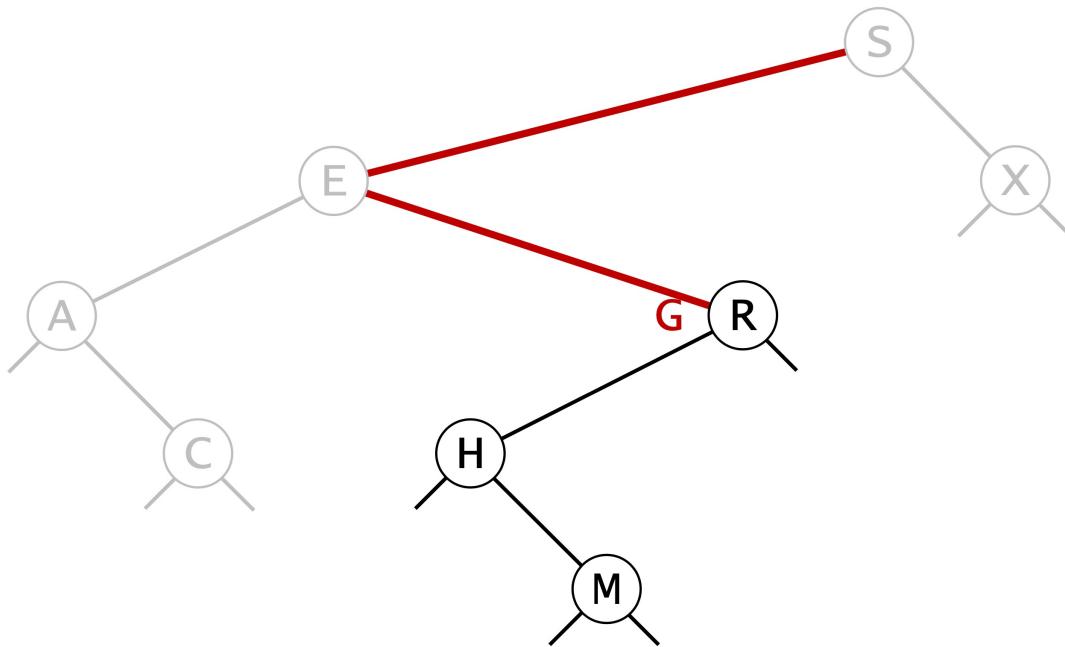
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



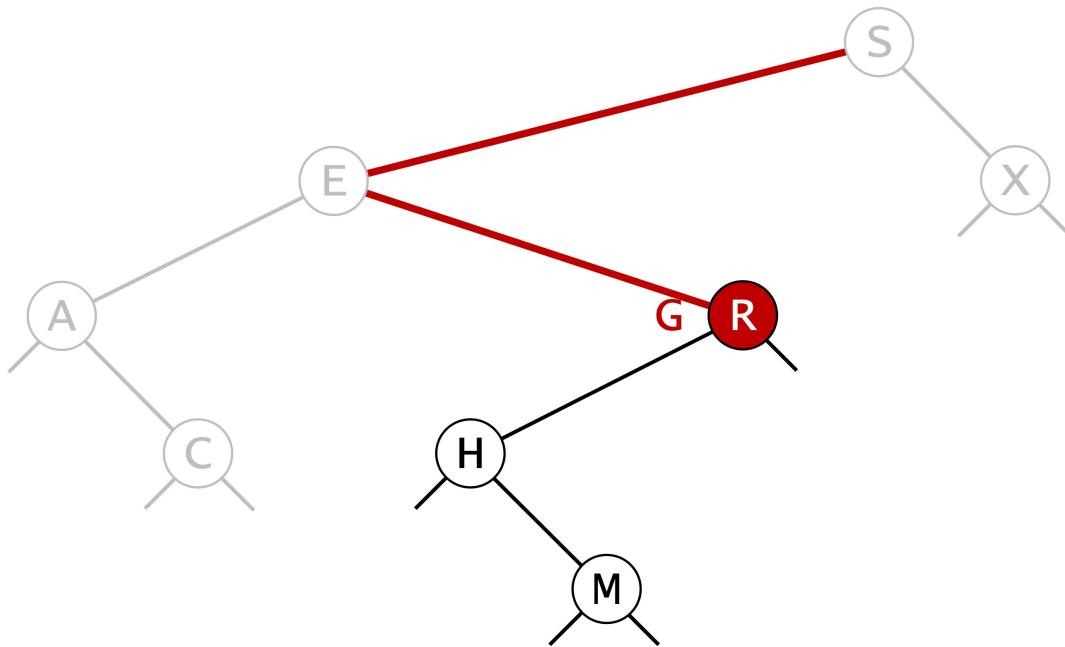
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



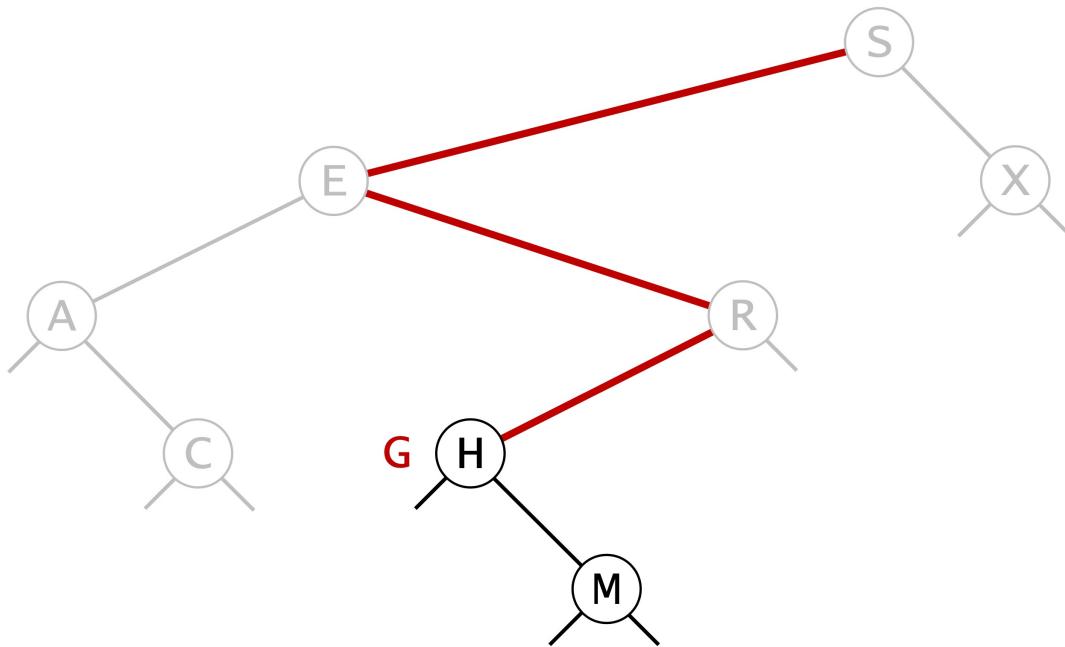
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



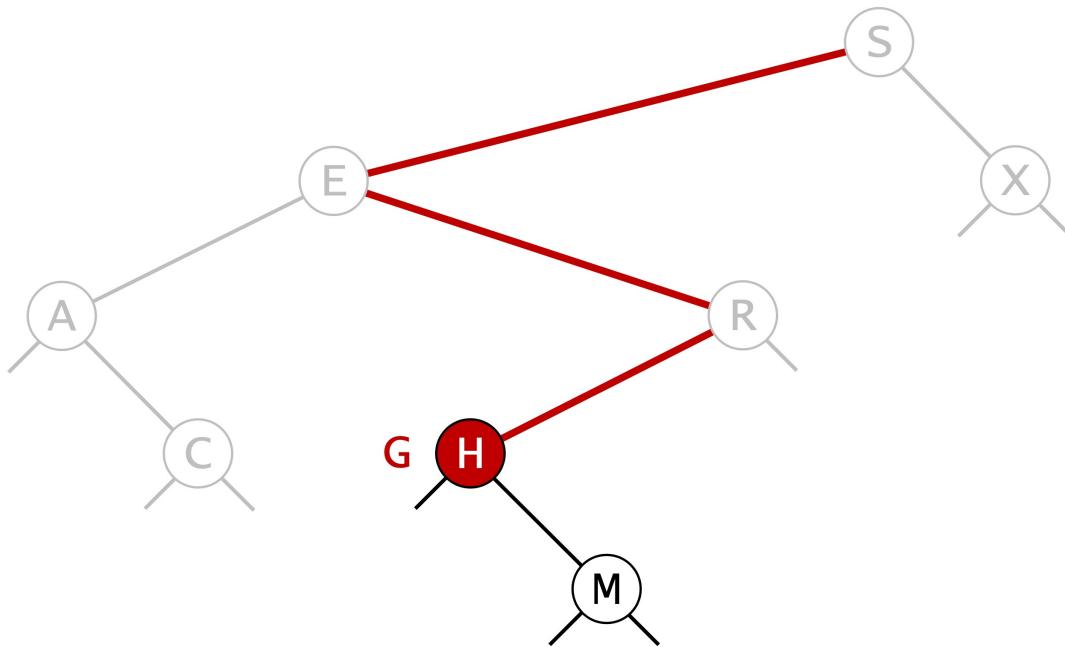
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



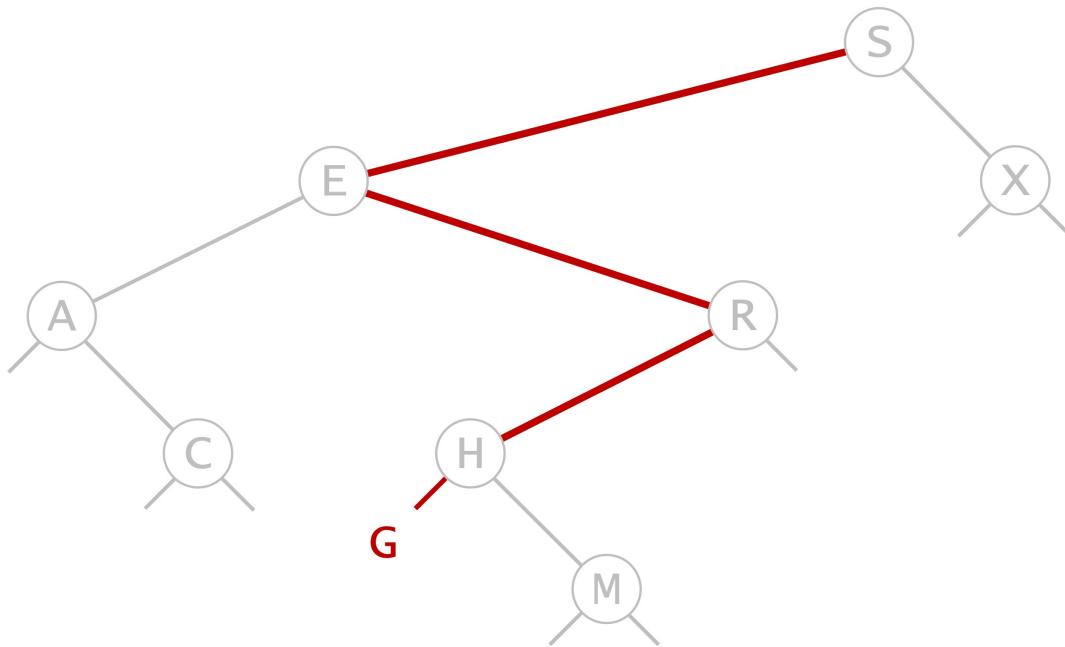
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



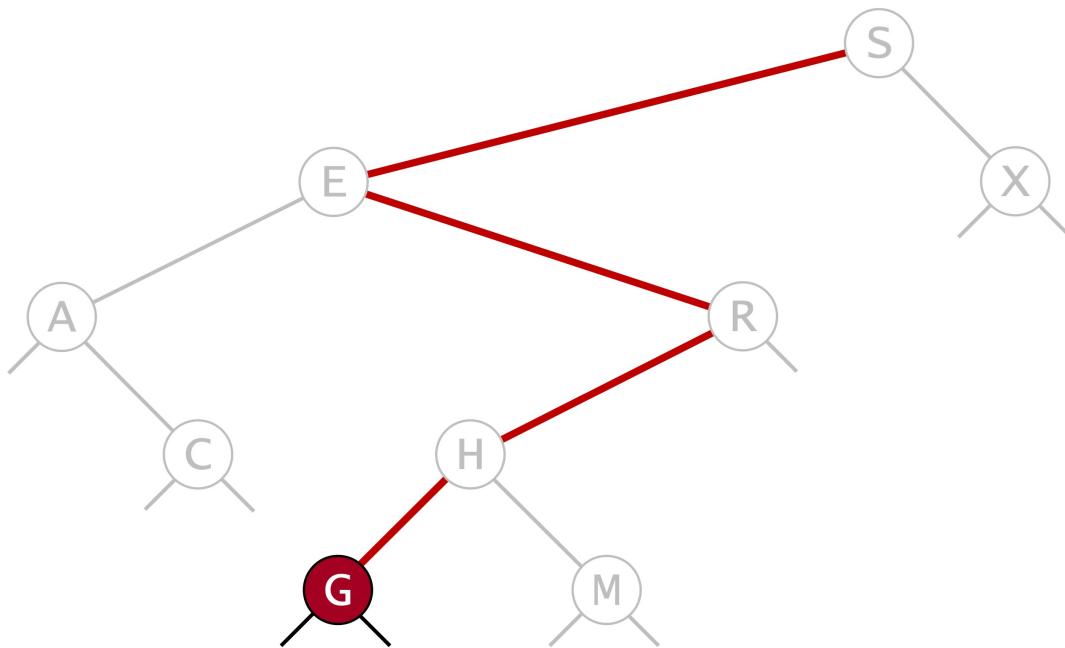
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



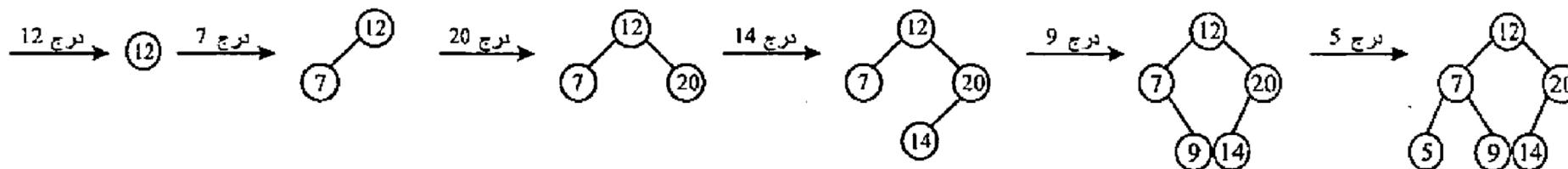
# درج

اگر کوچکتر، برو به چپ؛ اگر بزرگتر، برو به راست؛ اگر مساوی، تمام.



# مثال

درج آرایه با کلیدهای 5, 12, 7, 20, 14, 9 به شرح زیر است:



۱۲ • : بدون مقایسه در ریشه درخت

۷ • : ۱ مقایسه و درج در سمت چپ ۱۲

۲۰ • : ۱ مقایسه و درج در سمت راست ۱۲

۱۴ • : ۲ مقایسه و درج در سمت چپ ۲۰

۹ • : ۲ مقایسه و درج در سمت راست ۷

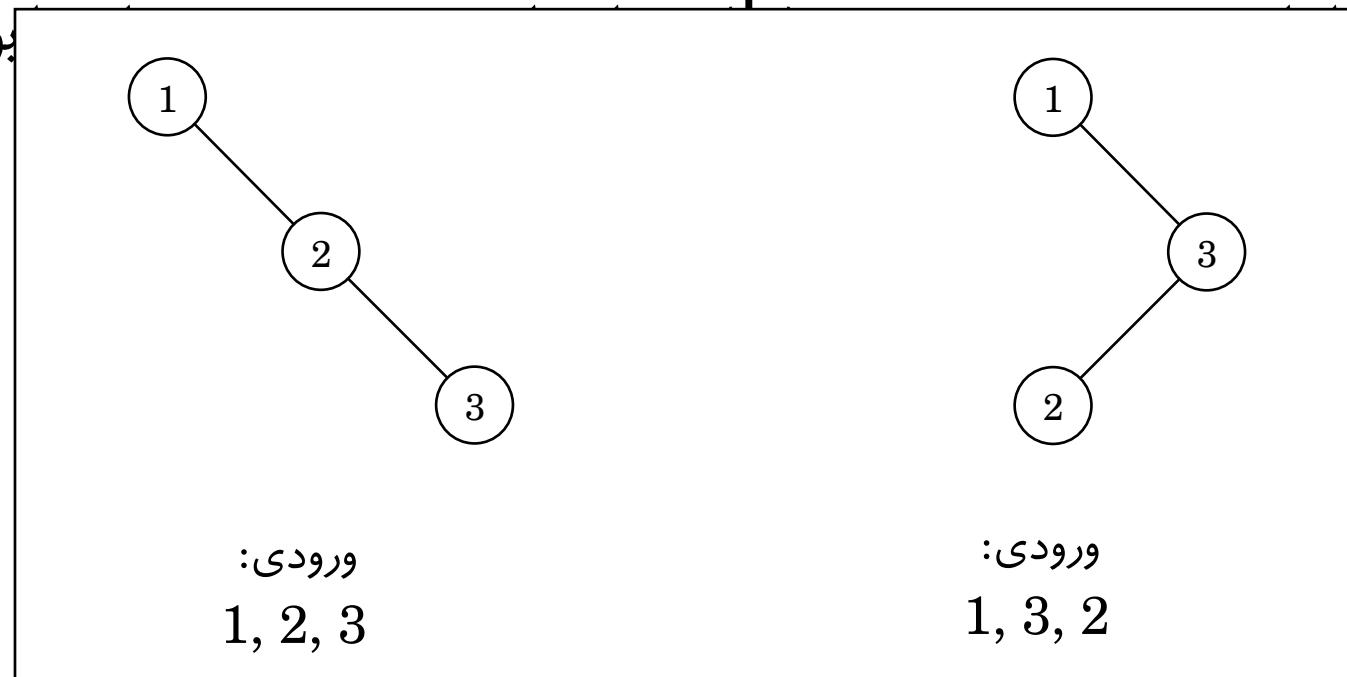
۵ • : ۲ مقایسه و درج در سمت چپ ۷

# وضعيت‌های ورودی‌ها و عمق درخت جستجو

در صورتی که ترتیب ورود کلیدها به BST متفاوت باشد، ممکن است درختان جستجو دودویی متفاوتی یا یکسانی ایجاد شود؛ بنابراین عمق درخت جستجو ( $h$ ) همواره وابسته به نحوه ورود داده‌ها بوده و تعیین کننده زمان اجرای اکثر عملیات در این درخت می‌باشد.

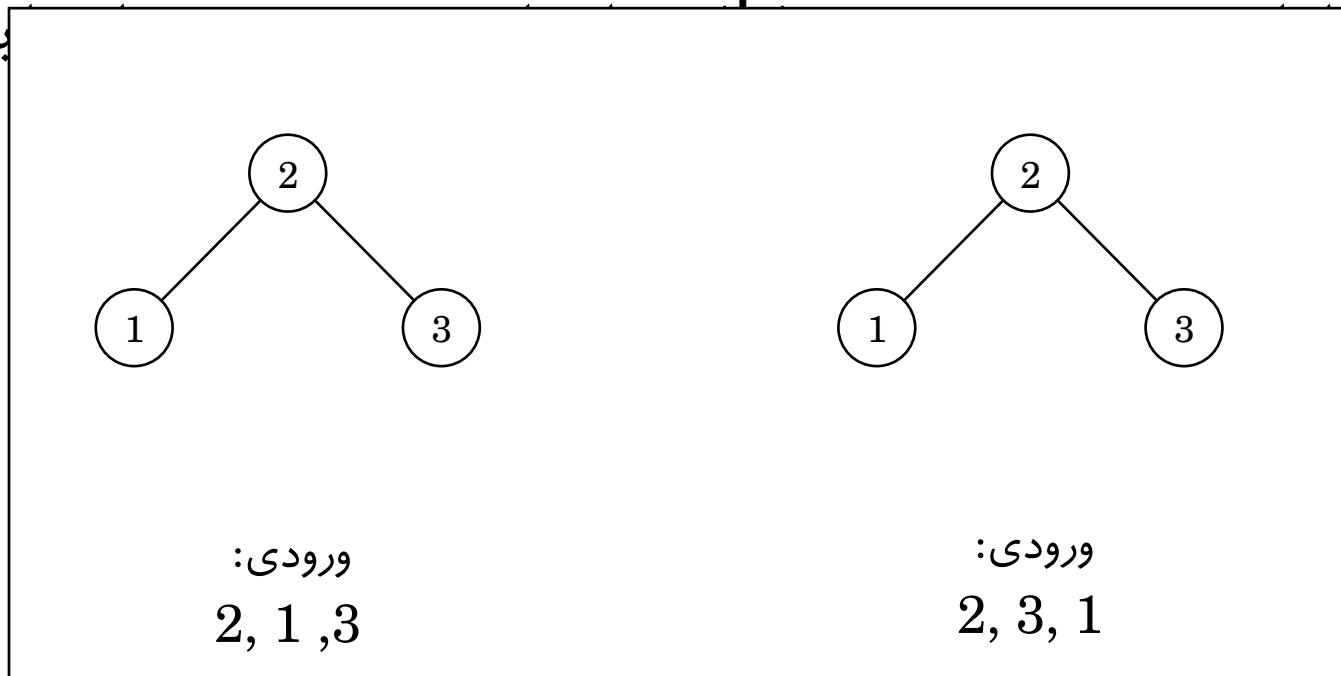
# وضایعیت‌های ورودی‌ها و عمق درخت جستجو

در صورتی که ترتیب ورود کلیدها به BST متفاوت باشد، ممکن است درختان جستجو دودویی متفاوتی یا یکسانی ایجاد شود؛ به زمان اجرای اکثر عمل



# وضعيت‌های ورودی‌ها و عمق درخت جستجو

در صورتی که ترتیب ورود کلیدها به BST متفاوت باشد، ممکن است درختان جستجو دودویی متفاوتی یا یکسانی ایجاد شود؛ با زمان اجرای اکثر عم

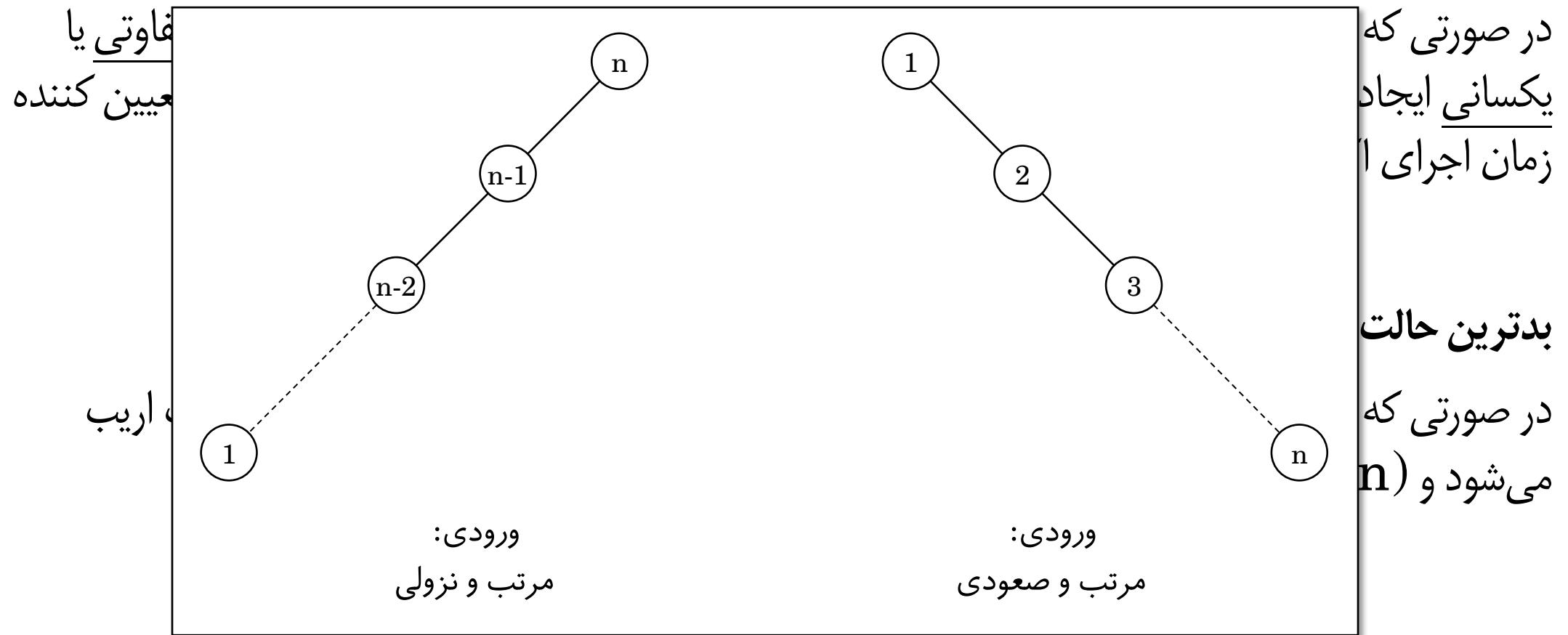


# وضعيت‌های ورودی‌ها و عمق درخت جستجو

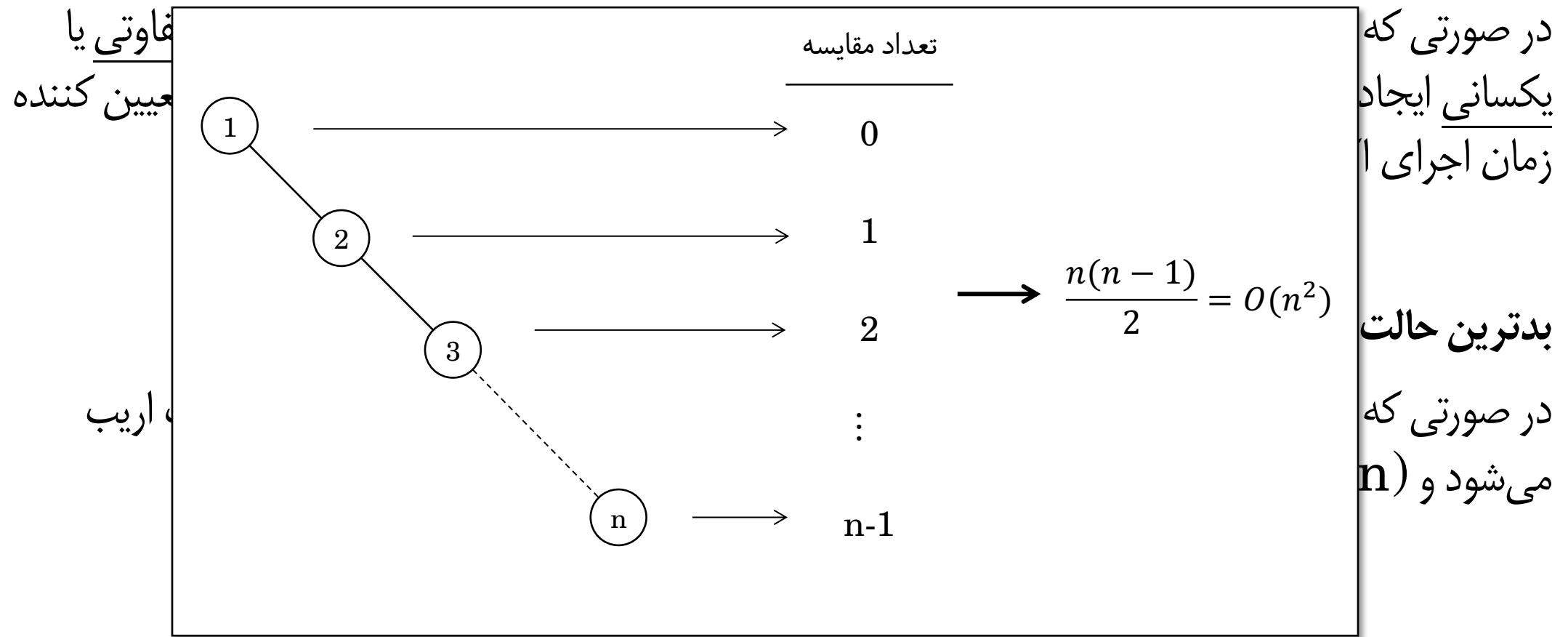
در صورتی که ترتیب ورود کلیدها به BST متفاوت باشد، ممکن است درختان جستجو دودویی متفاوتی یا یکسانی ایجاد شود؛ بنابراین عمق درخت جستجو ( $h$ ) همواره وابسته به نحوه ورود داده‌ها بوده و تعیین کننده زمان اجرای اکثر عملیات در این درخت می‌باشد.

**بدترین حالت:** ورودی مرتب یا معکوس در صورتی که کلیدها به طور مرتب (نزولی یا صعودی) در درخت قرار گیرند، درخت به چپ یا راست اریب می‌شود و  $h = O(n)$  خواهد شد.

# وضعيت‌های ورودی‌ها و عمق درخت جستجو



# وضعيت‌های ورودی‌ها و عمق درخت جستجو



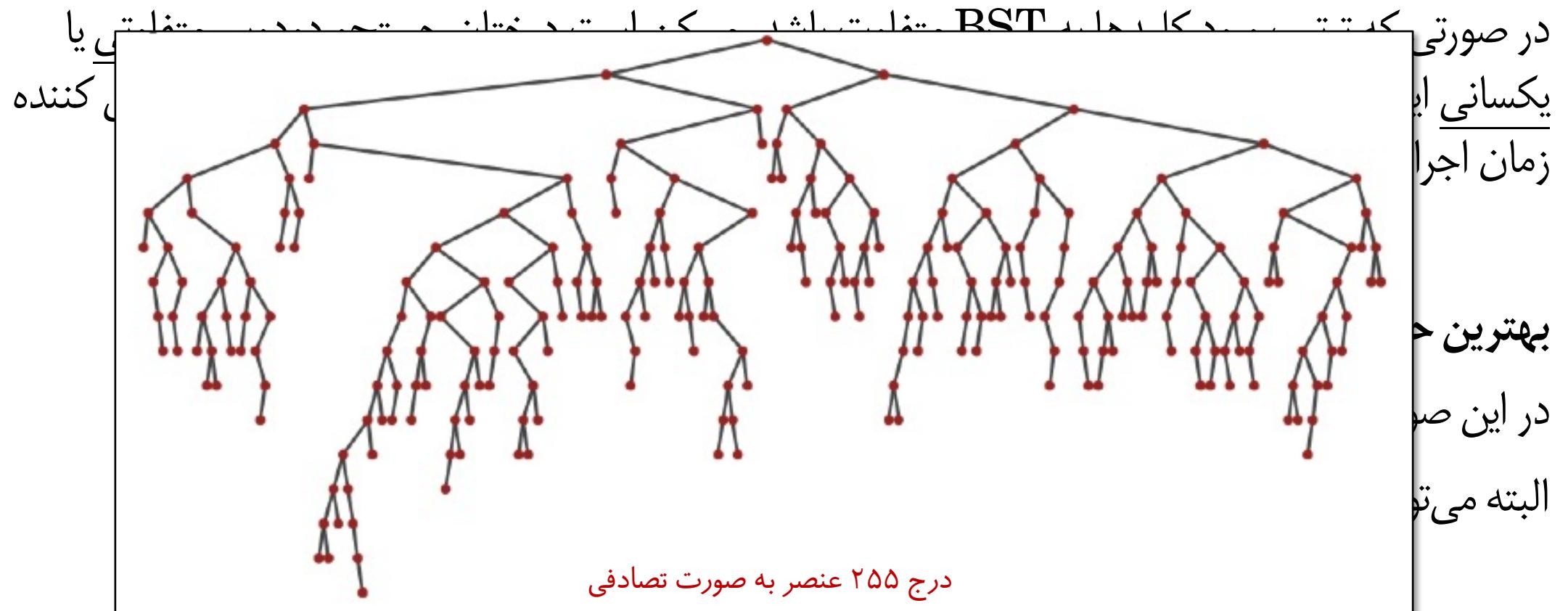
# وضعيت‌های ورودی‌ها و عمق درخت جستجو

در صورتی که ترتیب ورود کلیدها به BST متفاوت باشد، ممکن است درختان جستجو دودویی متفاوتی یا یکسانی ایجاد شود؛ بنابراین عمق درخت جستجو ( $h$ ) همواره وابسته به نحوه ورود داده‌ها بوده و تعیین کننده زمان اجرای اکثر عملیات در این درخت می‌باشد.

بهترین حالت: ورودی نامرتب

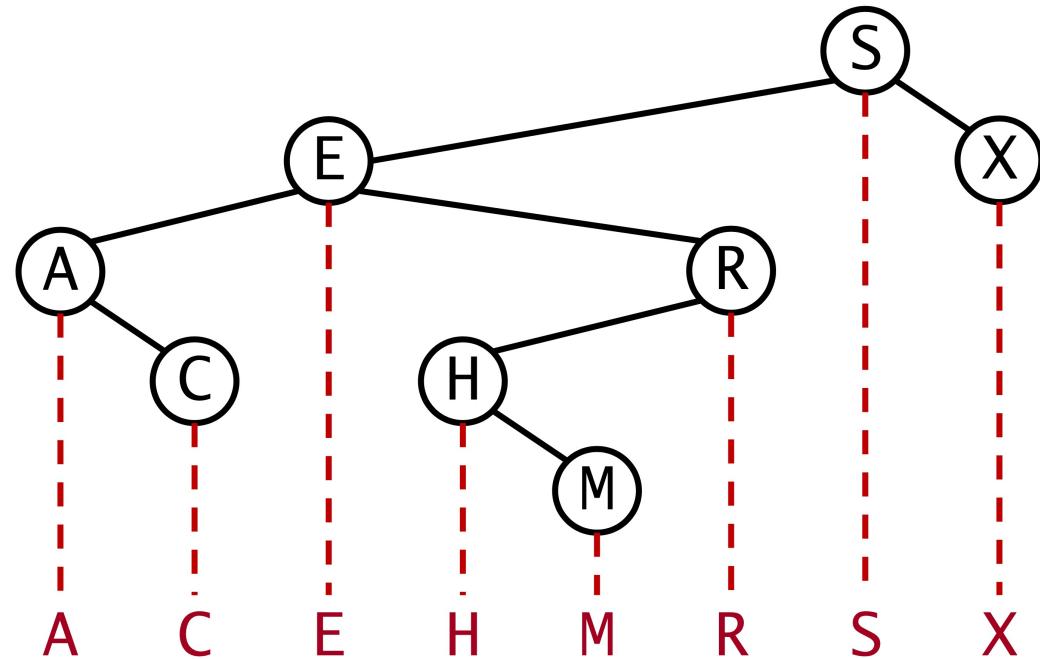
در این صورت ارتفاع درخت کمتر از تعداد گره‌ها خواهد شد و مقدار آن وابسته به ترتیب ورودی است؛  
البته می‌تواند درخت پر تشکیل دهد و آنگاه  $h = O(\log n)$  خواهد بود.

# وضعیت‌های ورودی‌ها و عمق درخت جستجو



# الگوریتم مرتب‌سازی

در پیماش میان ترتیب یک درخت جستجو دودویی، گره‌ها به ترتیب مقادیر کلیدشان از کوچک به بزرگ ملاقات می‌شوند.



# الگوریتم مرتب‌سازی

یک روش برای مرتب‌سازی  $n$  کلید:

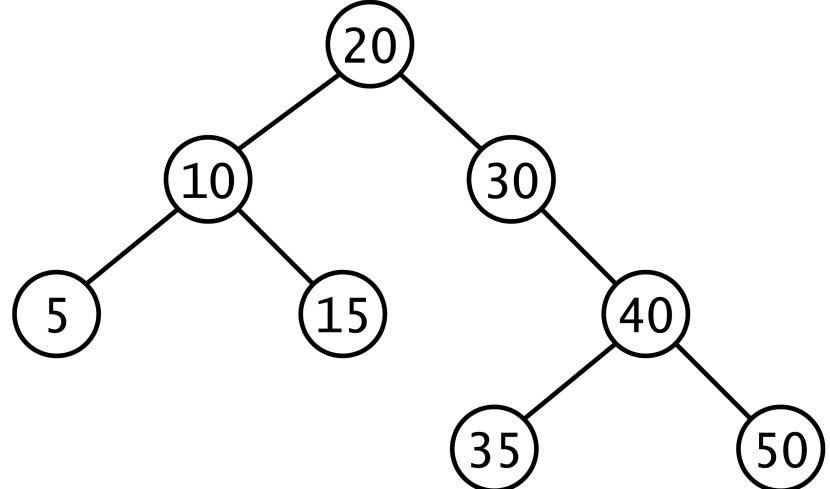
- با استفاده از عناصر داده شده یک درخت جستجوی دودویی ایجاد کن؛
- درخت حاصل را به روش میان‌ترتیب پیمایش کن.

# الگوریتم مرتب‌سازی

15, 50, 35, 5, 40, 10, 30, 20

یک روش برای مرتب‌سازی  $n$  کلید:

- با استفاده از عناصر داده شده یک درخت جستجوی دودویی ایجاد کن؛
- درخت حاصل را به روش میان‌ترتیب پیمایش کن.



5, 10, 15, 20, 30, 35, 40, 50

# الگوریتم حذف

برای حذف یک گره با کلید دلخواه از درخت BST بصورت زیر عمل می کنیم:

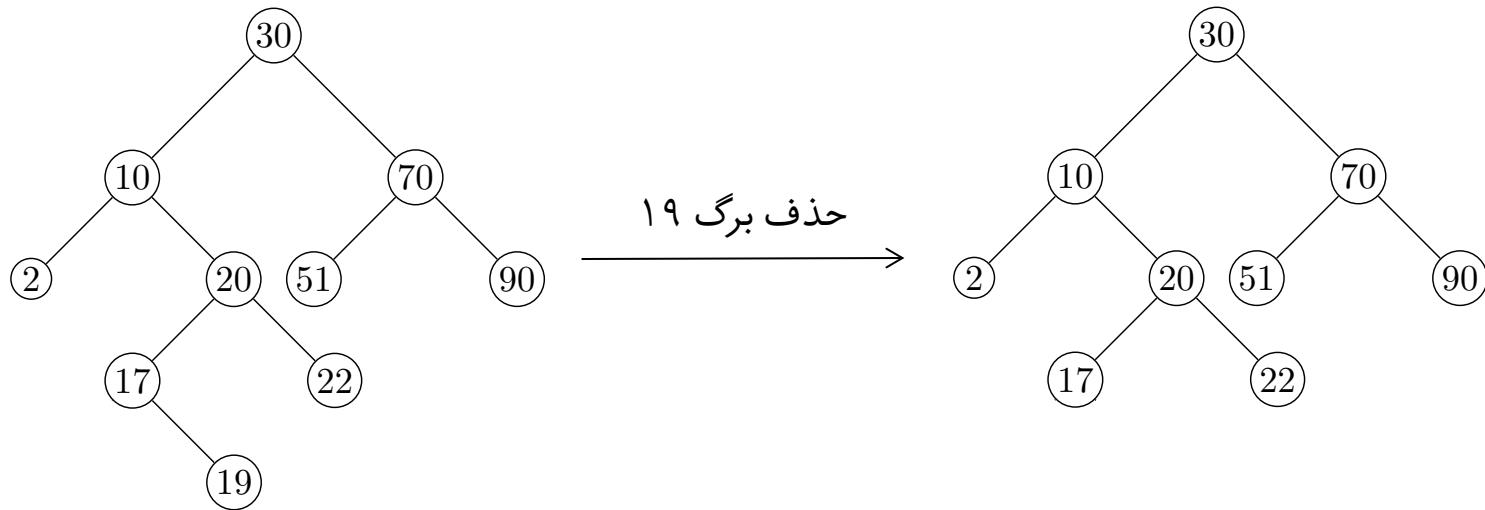
- جستجو در درخت برای یافتن موقعیت گره (در زمان  $O(h)$ )
- حذف گره مورد نظر در صورت موفقیت عملیات جستجو از موقعیت مشخص شده (در زمان  $O(1)$ )
- یک گره ممکن است بدون فرزند، تک فرزندی یا دو فرزندی باشد و روش حذف یک گره وابسته به تعداد فرزندان آن است.

بنابراین زمان اجرای الگوریتم برای حذف یک کلید دلخواه از درخت جستجو دودویی برابر با  $O(h)$  است.

# روش حذف

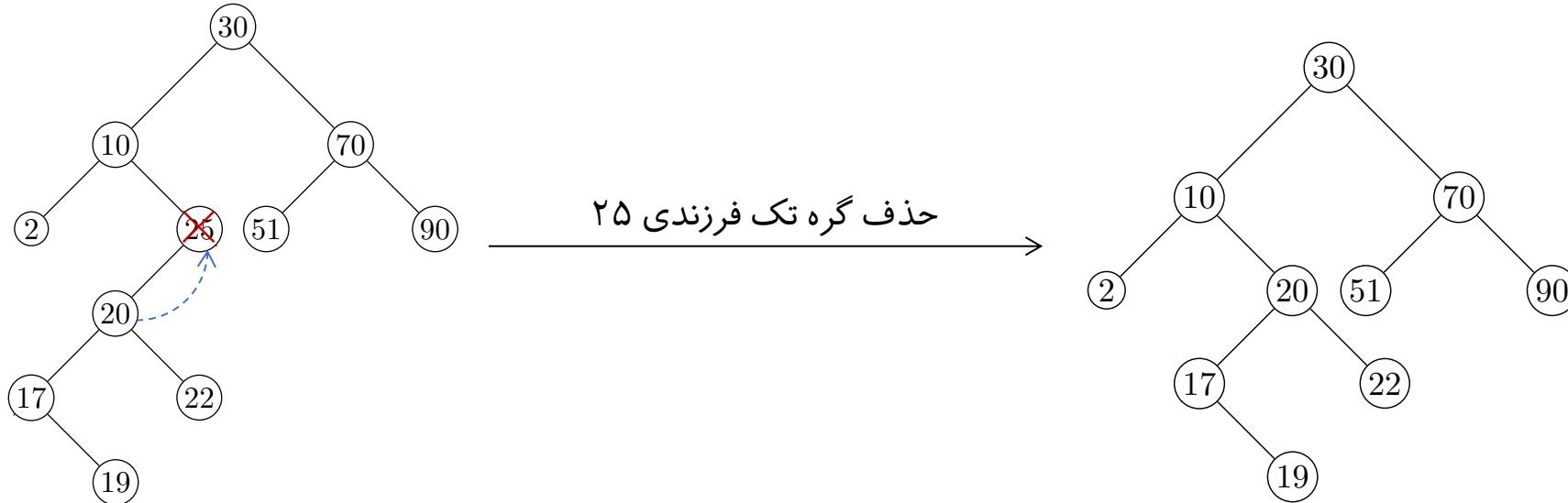
- حالت اول: **X** برگ است.

این حالت بسیار ساده است، کافیست تا اشاره‌گر پدر **X** را تهی کنیم.



# روش حذف

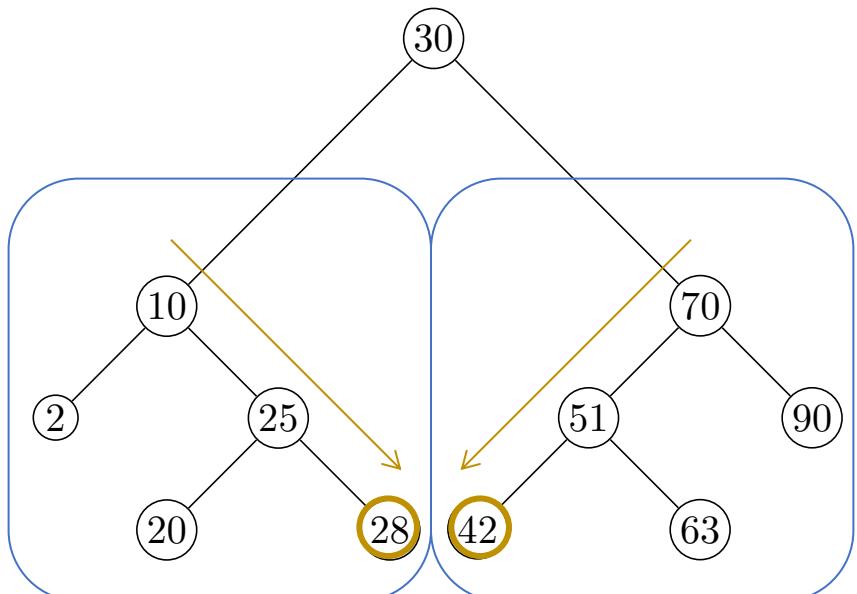
- حالت دوم:  $X$  تک فرزندی است.  
در این حالت هم اشاره‌گر پدر  $X$  را به فرزند  $X$  تغییر می‌دهیم.



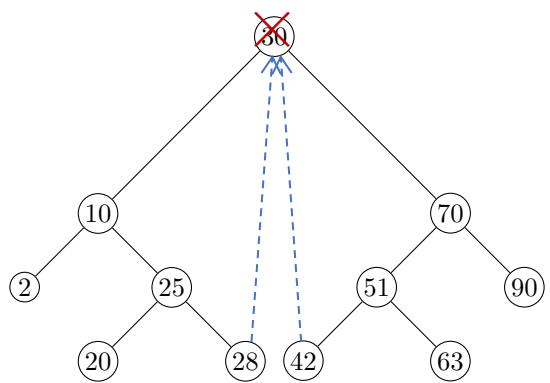
# روش حذف

- حالت سوم:  $X$  دو فرزندی است.

در حالت سوم، که در آن  $X$  دو فرزند دارد، گره‌ای را پیدا می‌کنیم که در پیماش InOrder بعد یا قبل  $X$  قرار می‌گیرد. در واقع گره کمینه زیردرخت راست و گره بیشینه زیردرخت چپ، این دو گره مورد نظر ما هستند؛ به دلخواه یکی از آنها را با  $X$  جایگزین می‌کنیم.

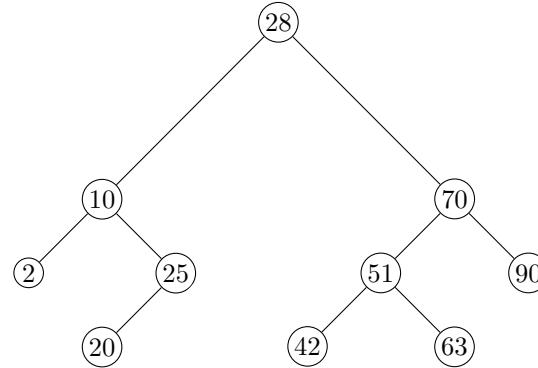


# روش حذف

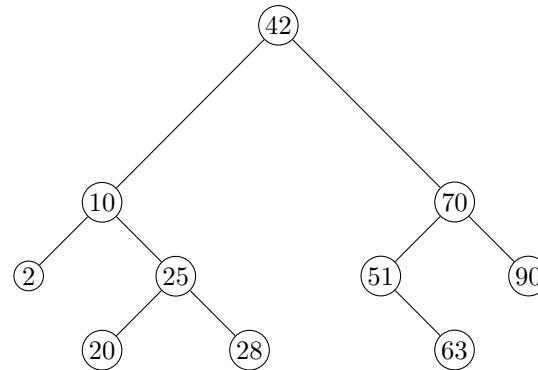


حذف گره دو فرزندی ۳۰

جایگزین شدن ۲۸

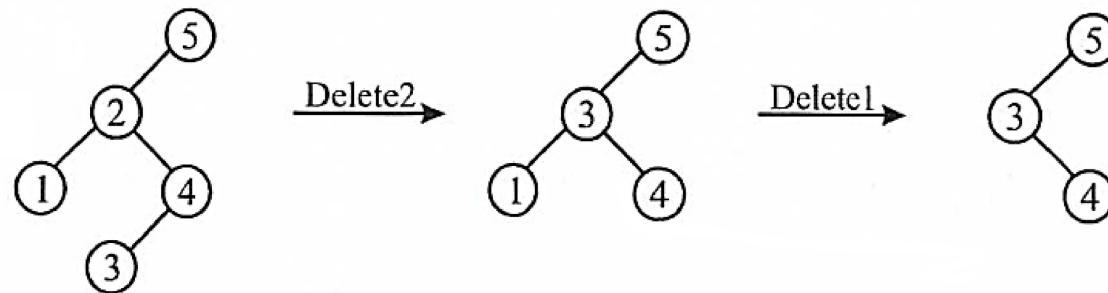
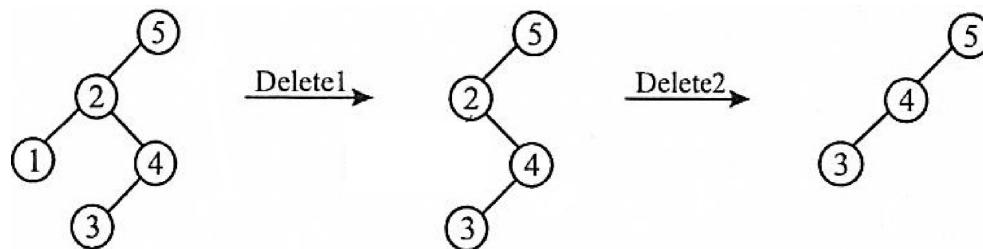


جایگزین شدن ۴۲



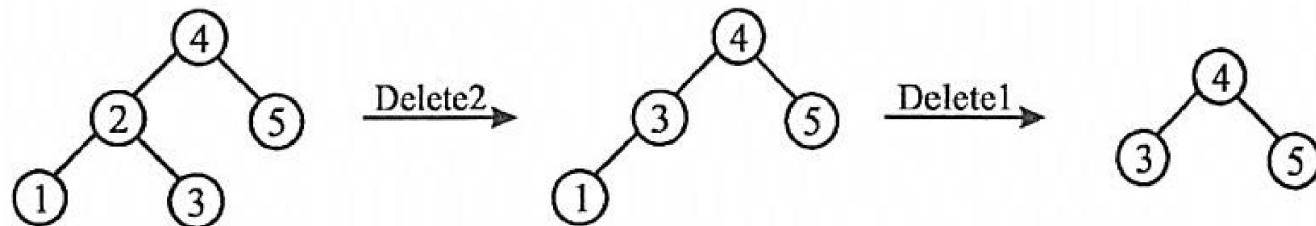
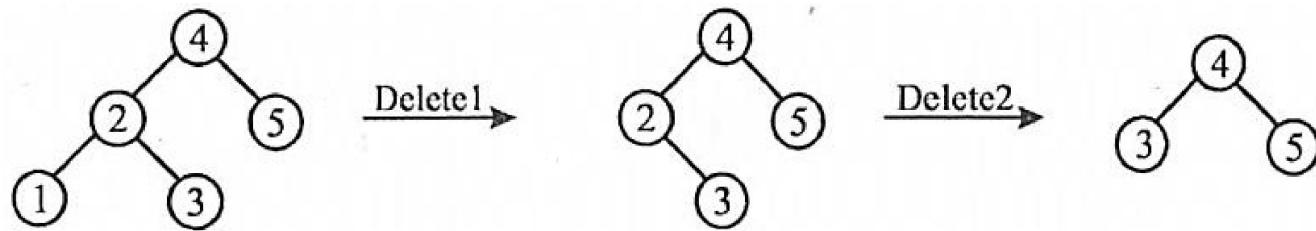
# ترتیب حذف

اگر ترتیب حذف دو کلید متفاوت از درخت تغییر کند، ممکن است که درخت های متفاوت یا یکسان ایجاد شود.



# ترتیب حذف

اگر ترتیب حذف دو کلید متفاوت از درخت تغییر کند، ممکن است که درخت های متفاوت یا یکسان ایجاد شود.



# مثال

می خواهیم با وارد کردن مقادیر ۱، ۲ و ۳ به هر ترتیب دلخواه در یک درخت جستجو دودویی تهی، یک درخت جستجو بسازیم. چند درخت متفاوت قابل ساخت است؟

۴ (۱)

۳ (۲)

۶ (۳)

۵ (۴)

# مثال

می خواهیم با وارد کردن مقادیر ۱، ۲ و ۳ به هر ترتیب دلخواه در یک درخت جستجو دودویی تهی، یک درخت جستجو بسازیم. چند درخت متفاوت قابل ساخت است؟

$$\frac{\binom{2n}{n}}{n+1} = \frac{\binom{6}{3}}{3+1} = \frac{\frac{6!}{3!(6-3)!}}{4} = 5 \quad (1)$$

۳ (۲)

۶ (۳)

۵ (۴)

# مثال

می خواهیم با وارد کردن مقادیر ۱، ۲ و ۳ به هر ترتیب دلخواه در یک درخت جستجو دودویی تهی، یک درخت جستجو بسازیم. چند درخت متفاوت قابل ساخت است؟

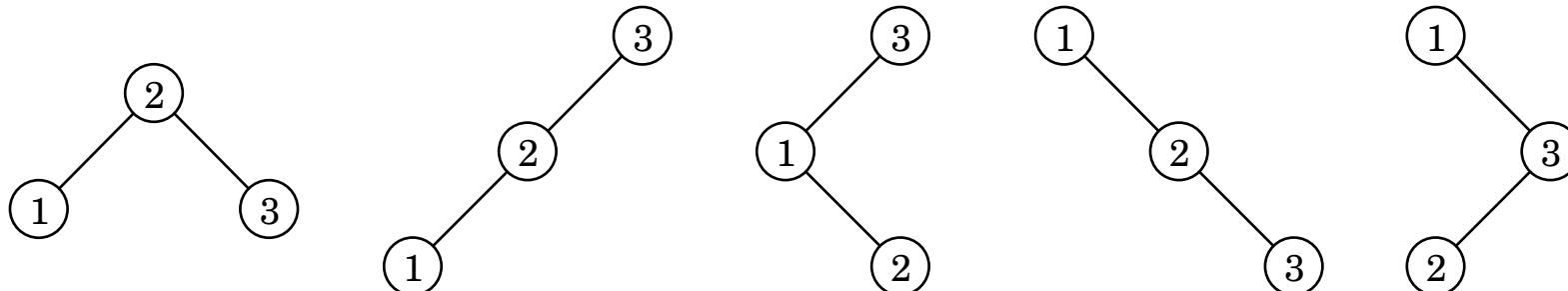
$$\frac{\binom{2n}{n}}{n+1} = \frac{\binom{6}{3}}{3+1} = \frac{\frac{6!}{3!(6-3)!}}{4} = 5$$

۴ (۱)

۳ (۲)

۶ (۳)

۵ (۴)



# مثال

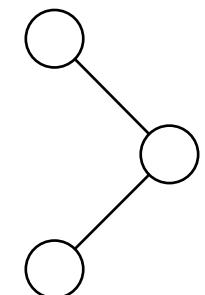
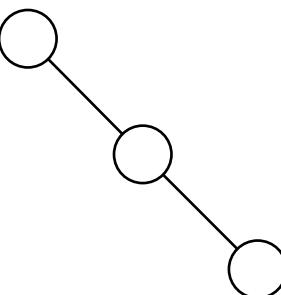
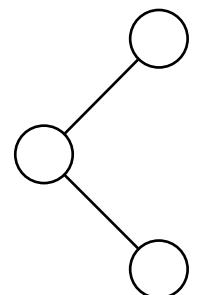
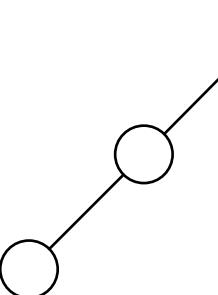
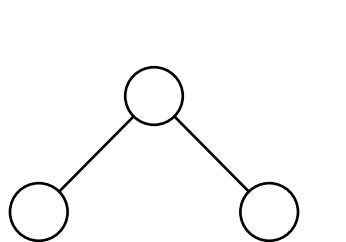
می خواهیم با وارد کردن مقادیر ۱، ۲ و ۳ به هر ترتیب دلخواه در یک درخت جستجو دودویی تهی، یک درخت جستجو بسازیم. چند درخت متفاوت قابل ساخت است؟

۴ (۱)

۳ (۲)

۶ (۳)

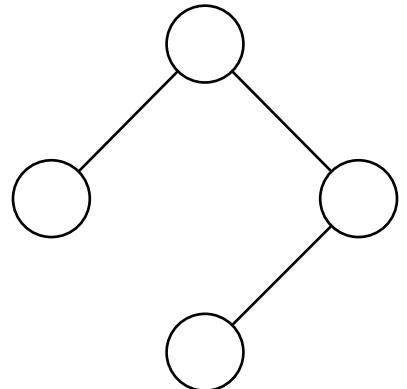
۵ (۴)



در حقیقت شکل این درختان همان درختان دودویی متفاوت است که با کاتالان محاسبه می شد.

# مثال

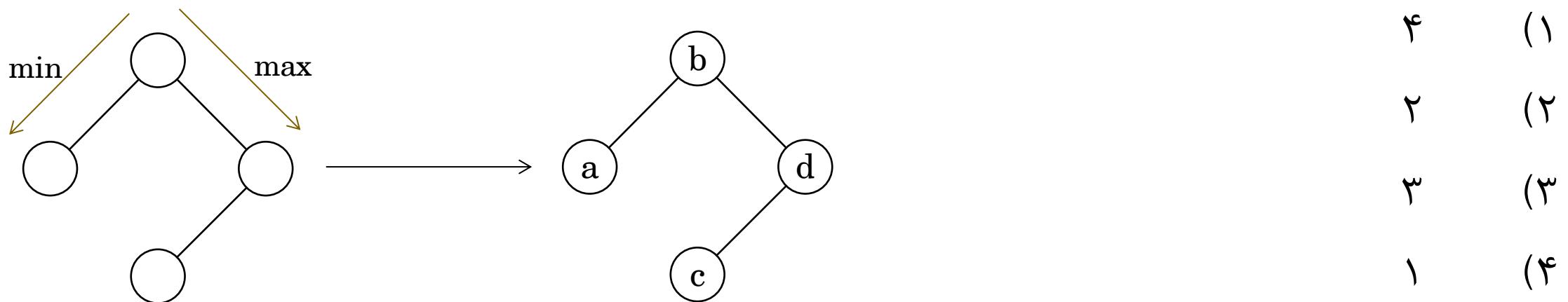
در چند حالت عناصر  $a < b < c < d$  را می‌توان وارد یک درخت جستجو دودویی تهی کرد تا درختی به شکل زیر ایجاد شود؟



- |   |     |
|---|-----|
| ۱ | (۱) |
| ۲ | (۲) |
| ۳ | (۳) |
| ۴ | (۴) |

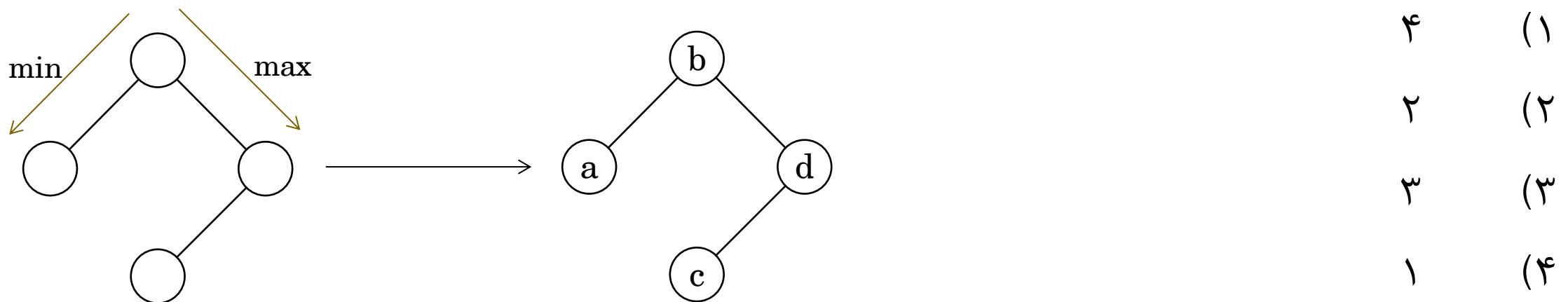
# مثال

در چند حالت عناصر  $a < b < c < d$  را می‌توان وارد یک درخت جستجو دودویی تهی کرد تا درختی به شکل زیر ایجاد شود؟



# مثال

در چند حالت عناصر  $a < b < c < d$  را می‌توان وارد یک درخت جستجو دودویی تهی کرد تا درختی به شکل زیر ایجاد شود؟

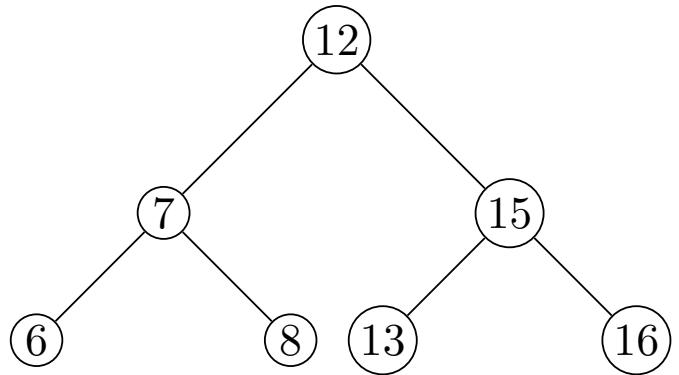


$b, d, a, c$   
 $b, d, c, a$   
 $b, a, d, c$

بنابراین  $b$  باید قبل از فرزندان  $a$  و  $d$  قبلاً از فرزند  $c$  بیاید.

# مثال

درخت جستجوی دودویی زیر نشان دهنده کدامیک از رشته‌های عدد زیر در ورودی می‌باشد؟



12, 6, 7, 8, 13, 15, 16 (۱)

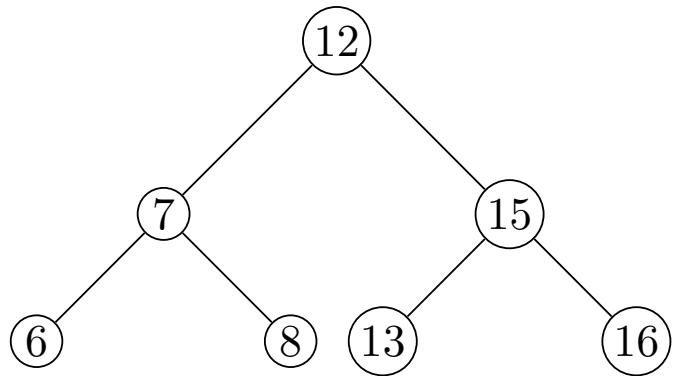
12, 7, 8, 16, 13, 15, 6 (۲)

12, 7, 6, 8, 13, 15, 16 (۳)

12, 15, 13, 7, 6, 16, 8 (۴)

# مثال

درخت جستجوی دودویی زیر نشان دهنده کدامیک از رشته های عدد زیر در ورودی می باشد؟



12, **6, 7,** 8, **13, 15,** 16 (۱)

12, 7, 8, **16,** **13, 15,** 6 (۲)

12, 7, 6, 8, **13, 15,** 16 (۳)

12, 15, 13, 7, 6, 16, 8 (۴)