درخت‌های جستجوی متوازن

سید ناصر رضوی www.snrazavi.ir
۱۳۹۵

مرور جدول نماد

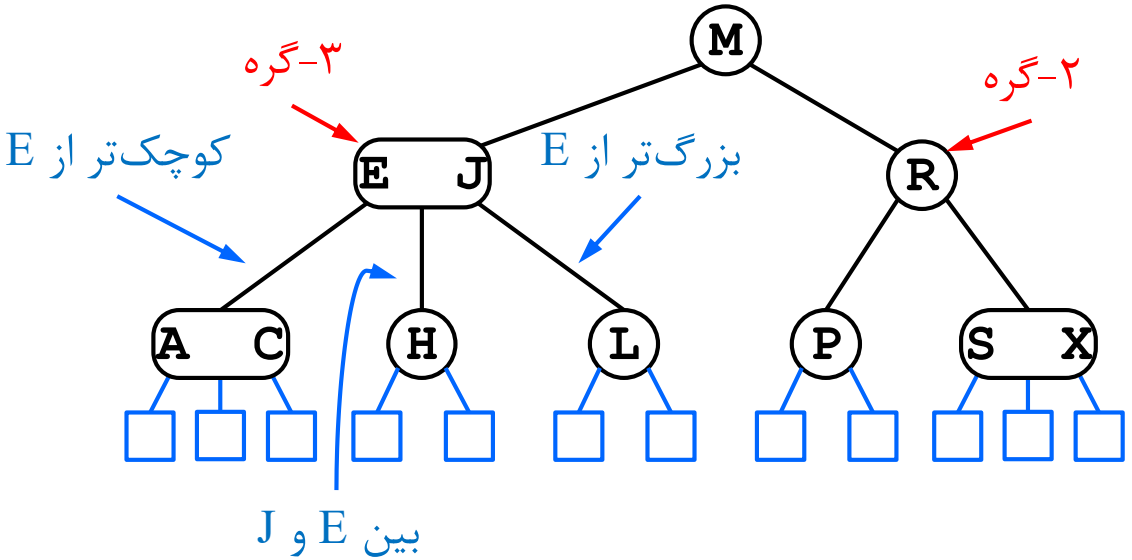
پیاده‌سازی	بدترین حالت			حالت متوسط			ترتیب؟
	جستجو	درج	حذف	جستجو	درج	حذف	
جستجوی ترتیبی (لیست نامرتب)	N	N	N	N/2	N	N/2	خیر
جستجوی دودویی (آرایه‌ی مرتب)	lg N	N	N	N/2	N/2	N/2	بله
درخت جستجوی دودویی	N	N	N	1.39 lg N	1.39 lg N	√N	بله
هدف	lg N	lg N	lg N	lg N	lg N	lg N	بله

- چالش. تضمین کارایی
- راه‌حل‌های ممکن. درخت‌های ۲-۳، درخت‌های قرمز-سیاه و درخت‌های بی!

درخت‌های ۲-۳

درخت ۲-۳

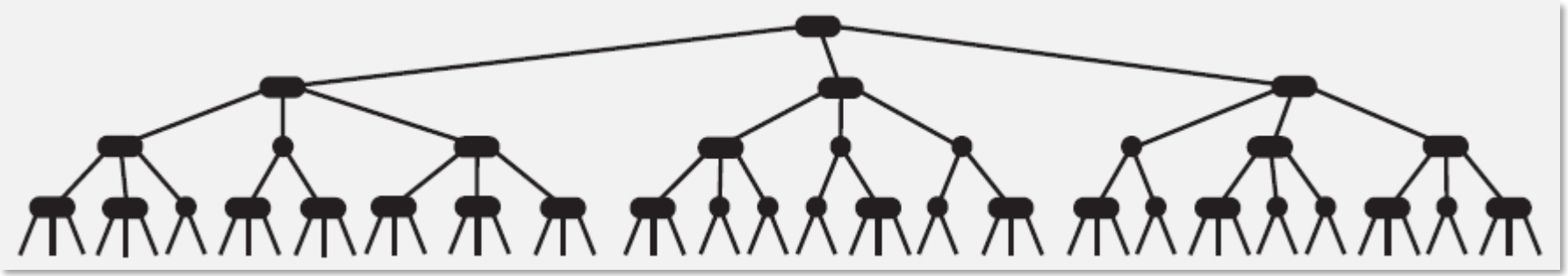
- یک درخت جستجو که امکان ذخیره‌ی یک یا دو کلید را در هر گره فراهم می‌کند:
  - ۲-گره: یک کلید و دو فرزند
  - ۳-گره: دو کلید و سه فرزند



- خصوصیات درخت‌های ۲-۳.
  - خاصیت جستجو: پیمایش میان‌ترتیب درخت باعث پیمایش گره‌ها به ترتیب صعودی می‌شود.
  - توازن کامل: طول همه‌ی مسیرها از ریشه تا پیوندهای پوچ برابر است.

توازن کامل

- توازن کامل. طول همه‌ی مسیرها از ریشه تا پیوندهای پوچ برابر است.



- ارتفاع درخت.
  - بدترین حالت: $\lfloor \log_2 N \rfloor$ [همه‌ی گره‌ها از نوع ۲-گره]
  - بهترین حالت: $\lfloor \log_3 N \rfloor$ [همه‌ی گره‌ها از نوع ۳-گره]
  - یک میلیون گره: بین ۱۲ و ۲۰
  - یک میلیارد گره: بین ۱۹ و ۳۰

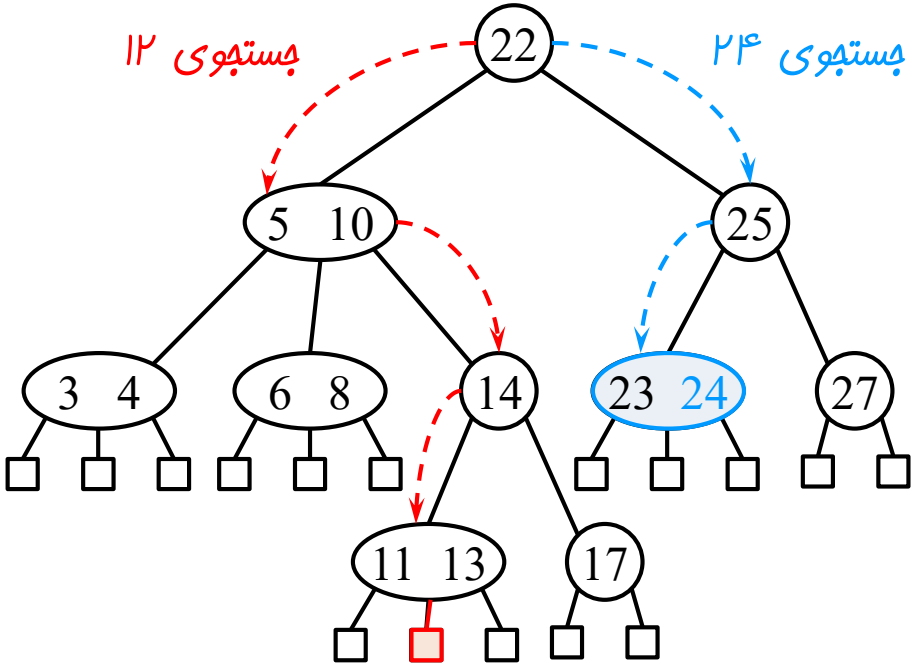
درخت ۲-۳ (تعریف بازگشتی)

تعریف. درخت جستجوی ۲-۳.

- یک درخت جستجوی ۲-۳ یا تهی است یا:
  - شامل یک ۲-گره با یک کلید و دو پیوند است، یک پیوند چپ به یک درخت ۲-۳ با کلیدهای کوچک‌تر و یک پیوند راست به یک درخت ۲-۳ با کلیدهای بزرگ‌تر.
  - شامل یک ۳-گره با دو کلید و سه پیوند است، یک پیوند چپ به یک درخت ۲-۳ با کلیدهای کوچک‌تر، یک پیوند میانی به یک درخت ۲-۳ با کلیدهایی بین دو کلید ریشه و یک پیوند راست به یک درخت ۲-۳ با کلیدهای بزرگ‌تر.

درخت جستجوی چند طرفه

- درخت جستجوی چند طرفه.
  - یک درخت جستجو که در آن هر گره داخلی حداقل دو فرزند دارد.



ارتفاع درخت ۲-۳

- ارتفاع یک درخت ۲-۳ شامل N کلید، همواره در رابطه‌ی زیر صدق می‌کند:

$$\log_3 N \leq h(N) \leq \log_2 N$$

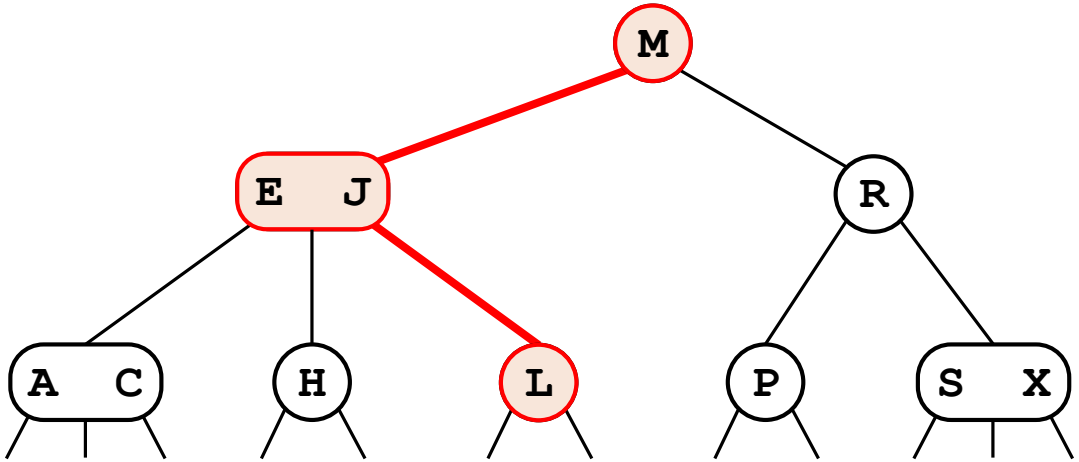
  - بدترین حالت: همه‌ی گره‌های داخلی از نوع ۲-گره هستند.
  - بهترین حالت: همه‌ی گره‌های داخلی از نوع ۳-گره هستند.

- نتیجه. هزینه‌ی درج، حذف و جستجو در یک درخت ۲-۳ با N کلید برابر است با $O(\log N)$.

درج در درخت ۲-۳

- حالت ۱) درج در یک ۲-گره در انتهای درخت.
  - مکان کلید مورد نظر را جستجو کن؛
  - ۲-گره یافته شده را با یک ۳-گره جایگزین کن.

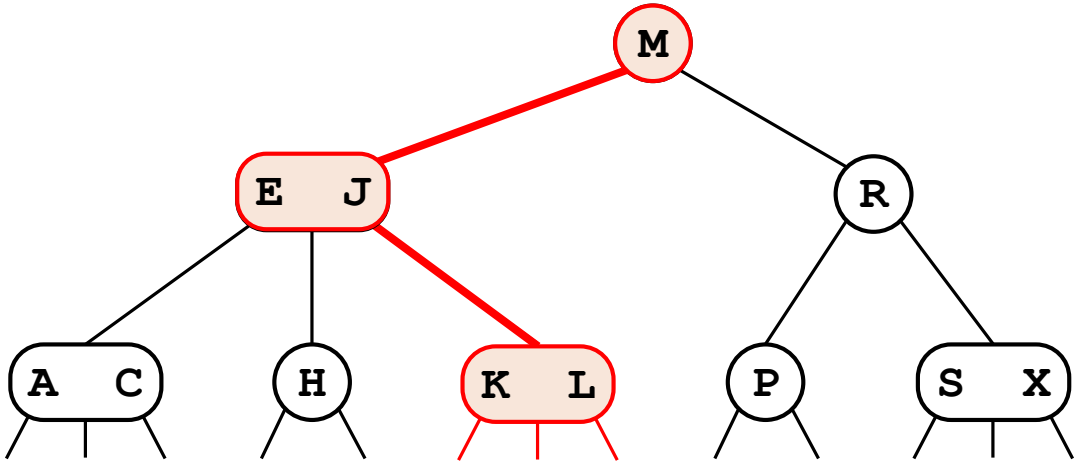
درج K



درج در درخت ۲-۳

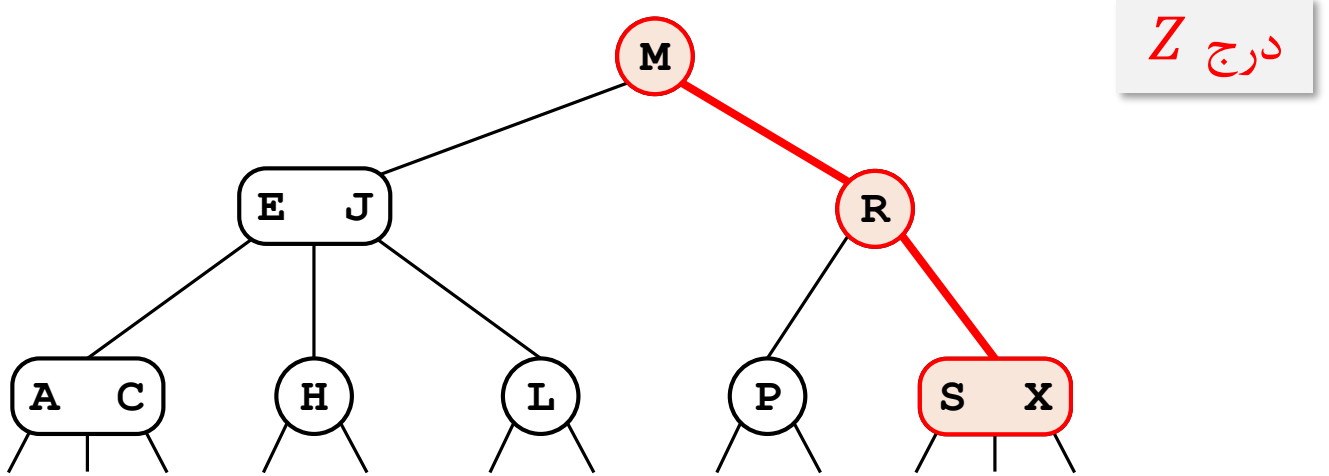
- حالت ۱) درج در یک ۲-گره در انتهای درخت.
  - مکان کلید مورد نظر را جستجو کن؛
  - ۲-گره یافته شده را با یک ۳-گره جایگزین کن.

درج K



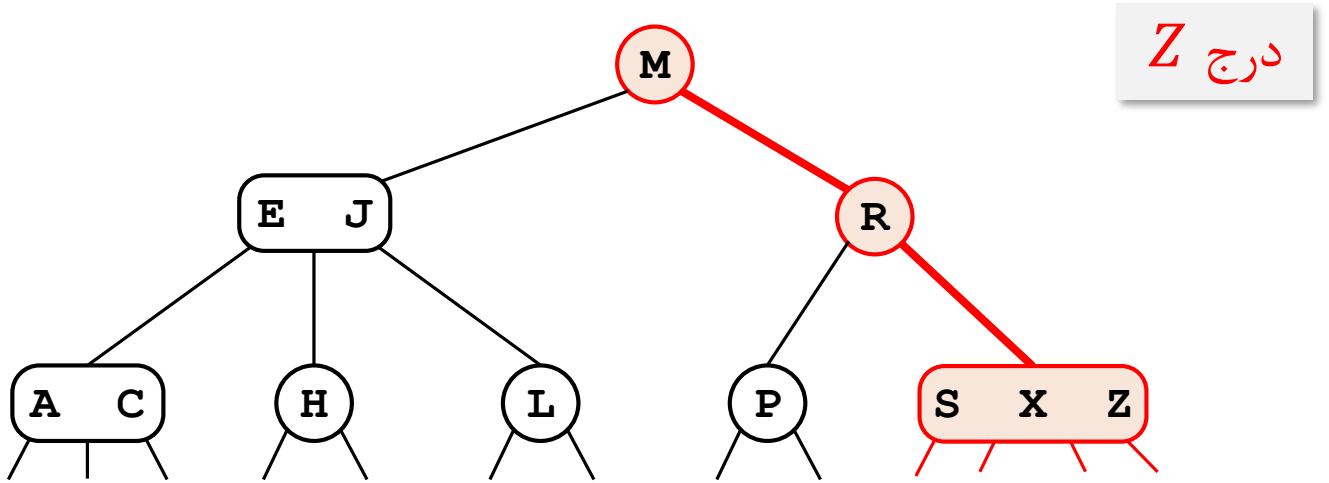
درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]



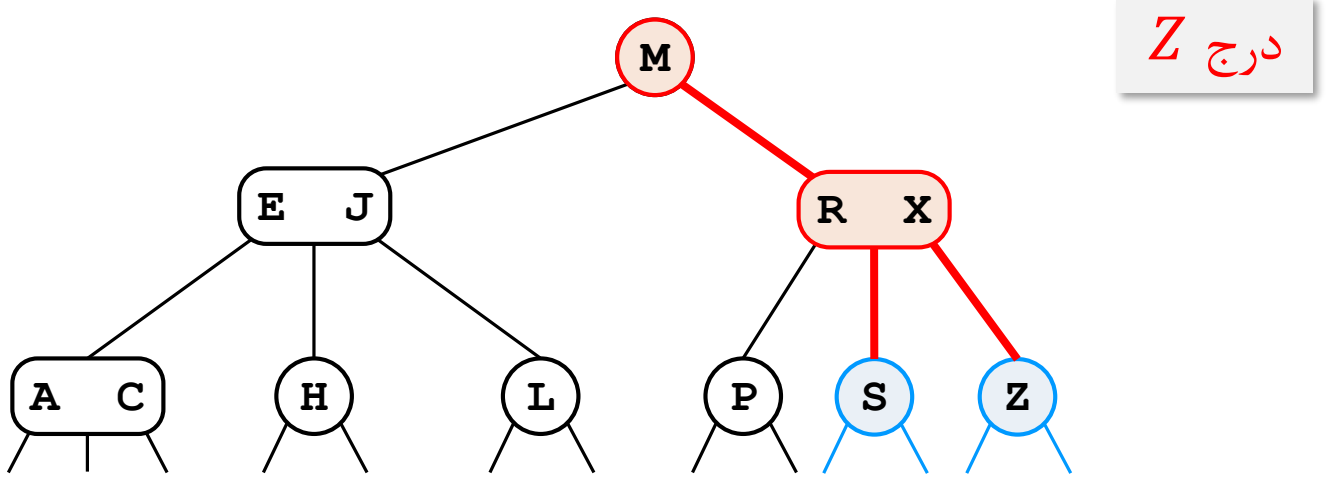
درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]



درج در درخت ۲-۳

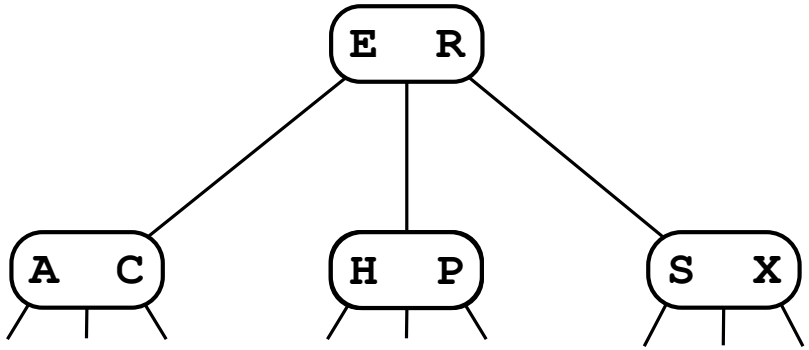
- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]



درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]

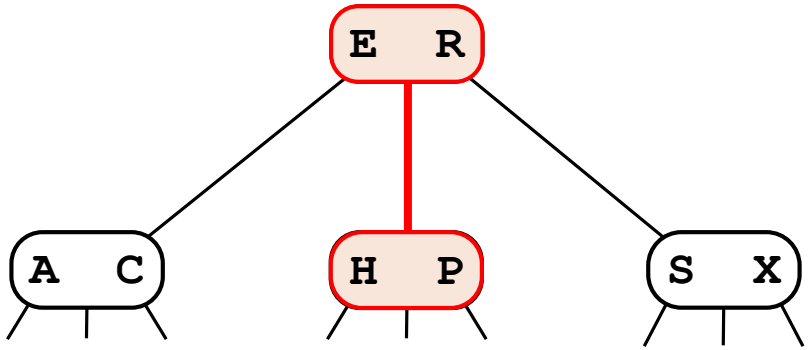
درج L



درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]

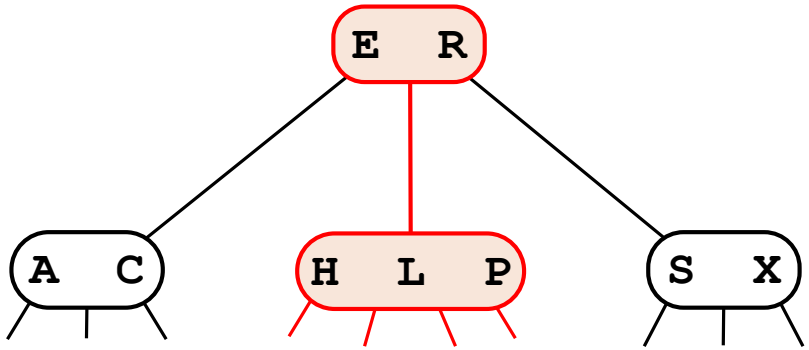
درج L



درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]

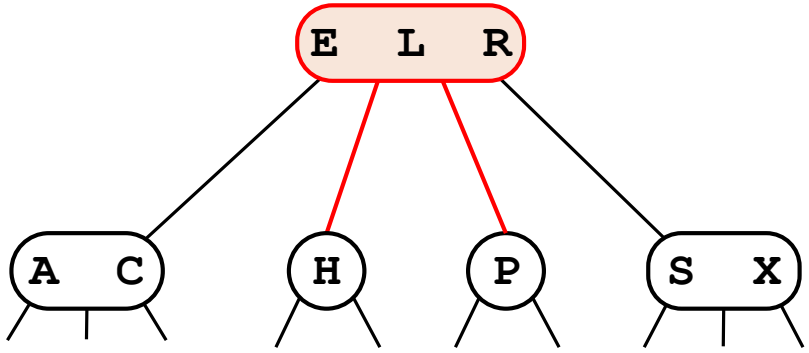
درج L



درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]

درج L

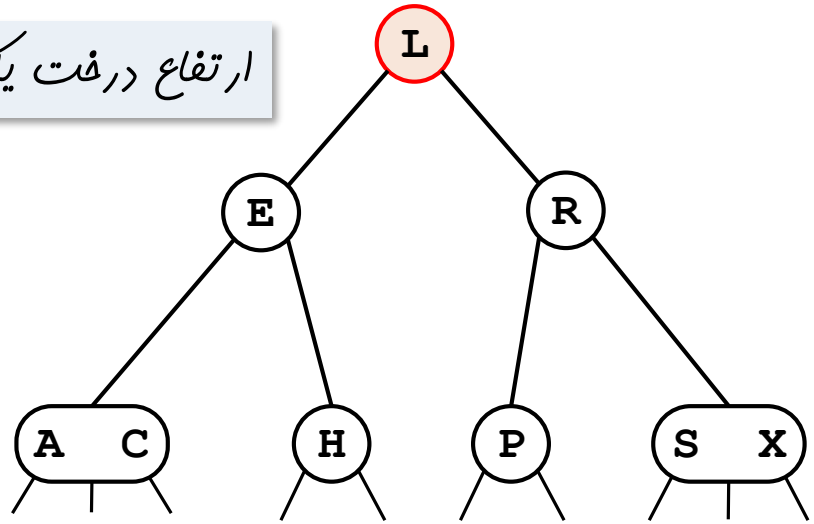


درج در درخت ۲-۳

- حالت ۲) درج در یک ۳-گره در انتهای درخت.
  - کلید مورد نظر را به ۳-گره اضافه کن و به طور موقت یک ۴-گره ایجاد کن؛
  - کلید میانی در ۴-گره ایجاد شده را به گره پدر منتقل کن. [تقسیم]

درج L

ارتفاع درخت یک واحد افزایش می‌یابد



ایجاد درخت ۲-۳

- تمرین. کلیدهای زیر را به ترتیب از چپ به راست در یک درخت ۲-۳ درج کنید.

S E A R C H X P L

پیاده‌سازی درخت ۲-۳

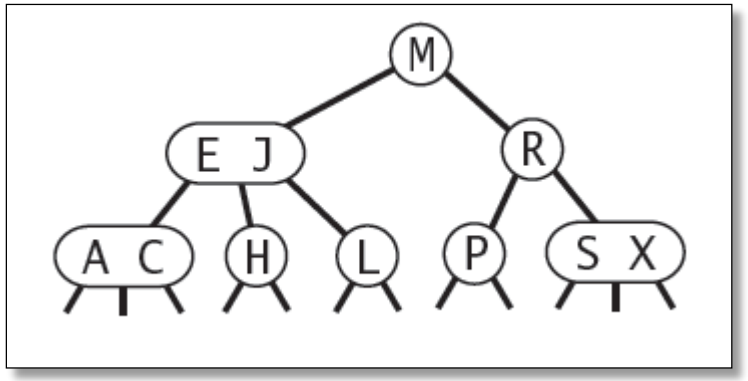
- پیاده‌سازی مستقیم درخت ۲-۳ پیچیده است، به دلیل:
  - داشتن انواع مختلفی از گره‌ها.
  - نیاز به چندین مقایسه در هر گره در حین عملیات جستجو، درج و حذف.
  - نیاز به بالا رفتن در درخت در اثر تقسیم کردن ۴-گره‌ها.
  - وجود حالت‌های متعدد برای تقسیم.

- نتیجه. پیاده‌سازی مستقیم درخت‌های ۲-۳ ممکن است، اما روش بهتری نیز وجود دارد!

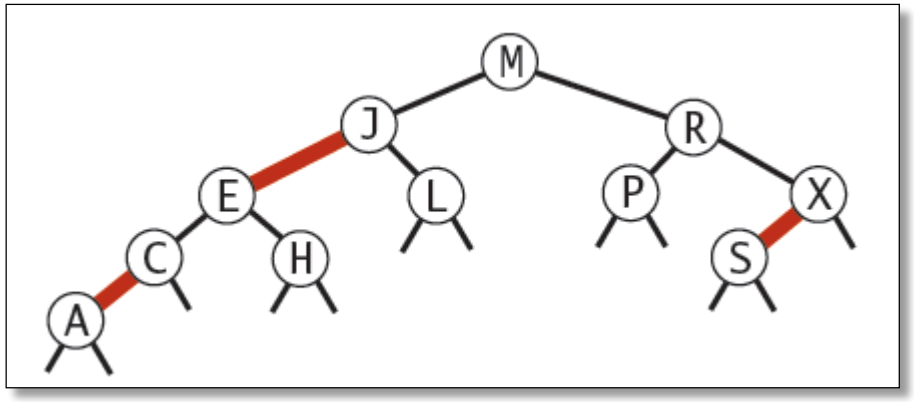
درخت‌های قرمز-سیاه

درخت‌های قرمز-سیاه (چپ مایل)

- نمایش درخت‌های ۲-۳ به صورت درخت جستجوی دودویی.
- استفاده از پیوندهای قرمز مایل به چپ برای نمایش ۳-گره‌ها.



درخت ۲-۳

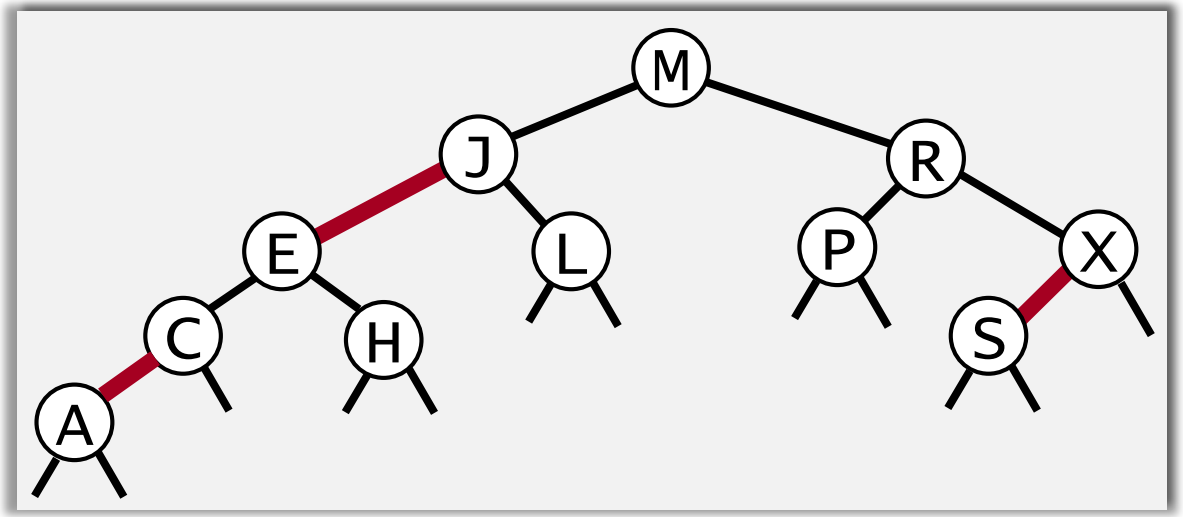


درخت قرمز-سیاه مربوطه

تعریف درخت قرمز-سیاه (چپ مایل)

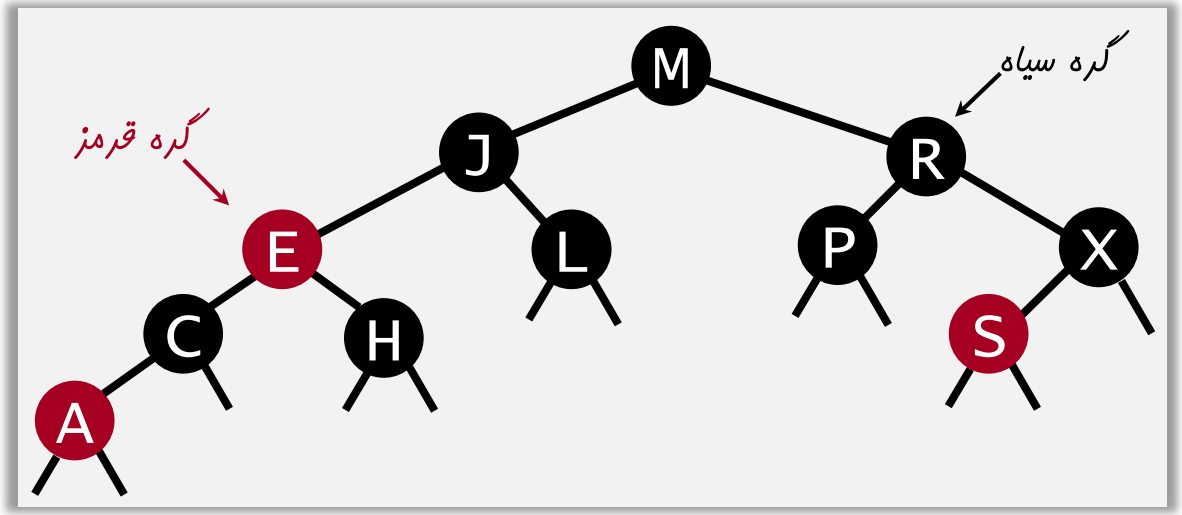
- درخت قرمز سیاه چپ مایل. یک درخت جستجوی دودویی به طوری که:
  - هیچ گره‌ای نمی‌تواند دو پیوند قرمز داشته داشته باشد.
  - تعداد پیوندهای سیاه در تمام مسیرها از ریشه به پیوندهای پوچ برابر است.
  - پیوندهای قرمز تنها می‌توانند به سمت چپ مایل باشند.

ارتفاع سیاه کامل



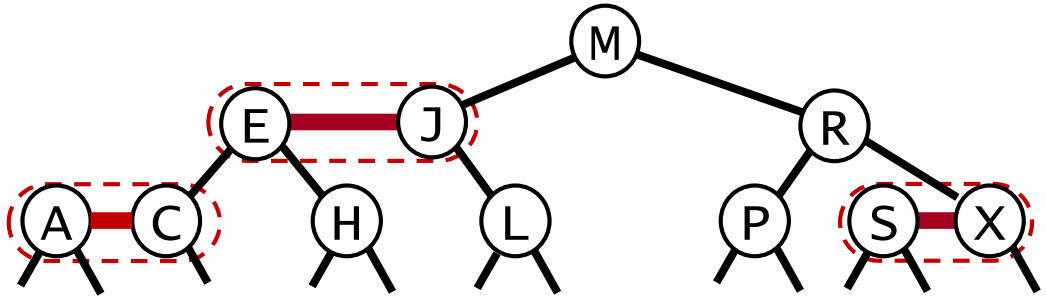
پیاده‌سازی درخت قرمز-سیاه

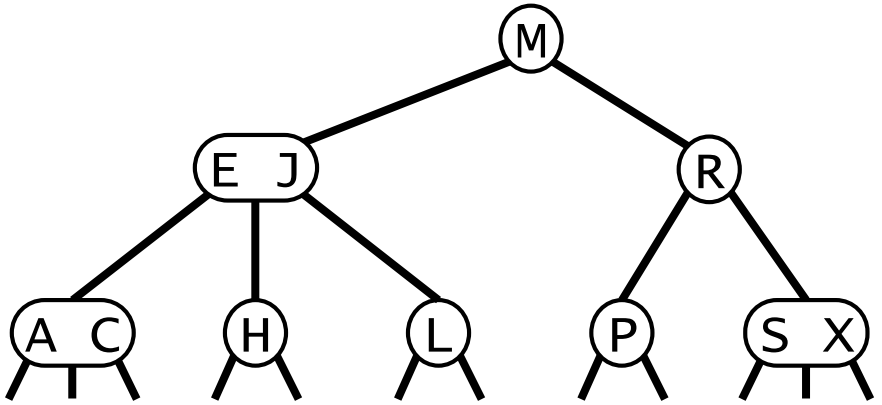
- از آنجا که هر گره تنها یک پیوند با پدرش دارد، در پیاده‌سازی می‌توان اطلاعات مربوط به رنگ پیوندها را در خود گره‌ها ذخیره نمود:
  - گره قرمز: دارای پیوند قرمز با پدر
  - گره سیاه: دارای پیوند سیاه با پدر

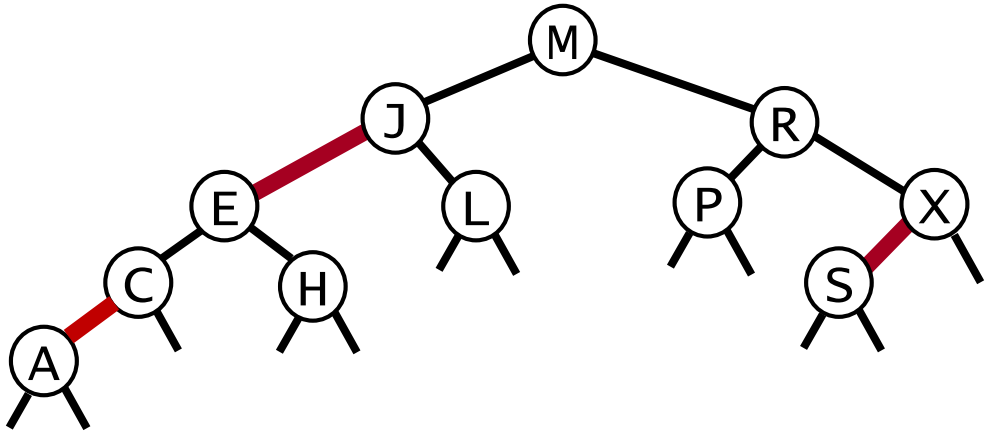


ارتباط درخت قرمز-سیاه با درخت ۲-۳

- ویژگی کلیدی. تناظر یک به یک میان درخت‌های ۲-۳ و درخت‌های قرمز-سیاه.



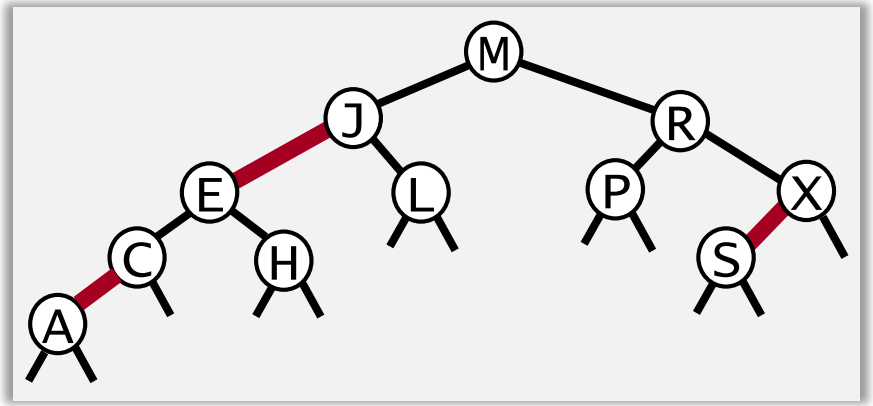




جستجو در درخت قرمز-سیاه

- مشاهده. جستجو در درخت قرمز-سیاه دقیقاً مانند جستجو در درخت جستجوی دودویی است!
  - اما به دلیل متوازن‌تر بودن درخت، سریع‌تر است.

```
public Value get(Key key)
{
   Node x = root;
   while (x != null)
   {
      int cmp = key.compareTo(x.key);
      if      (cmp <  0) x = x.left;
      else if (cmp > 0 ) x = x.right;
      else if (cmp == 0) return x.value;
   }
   return null;
}
```



- توجه. بسیاری از عملیات دیگر درخت قرمز-سیاه نیز مانند درخت جستجوی دودویی است.

[مانند یافتن کف و سقف، انتخاب و غیره]

پیاده‌سازی در جاوا

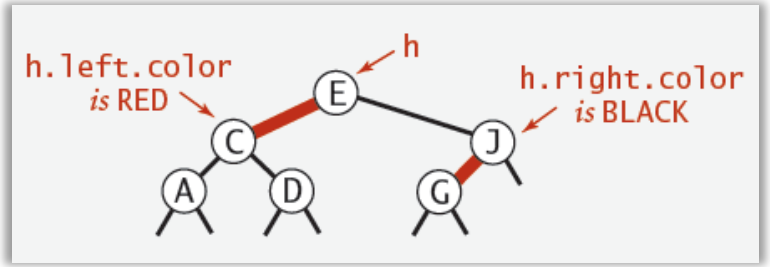
- در نظر گرفتن رنگ پیوندها در گره‌ها.

```
private static final boolean RED   = true;
private static final boolean BLACK = false;
```

```
private class Node
{
   Key key;
   Value value;
   Node left, right;
   boolean color;     // color of link to parent

}
```

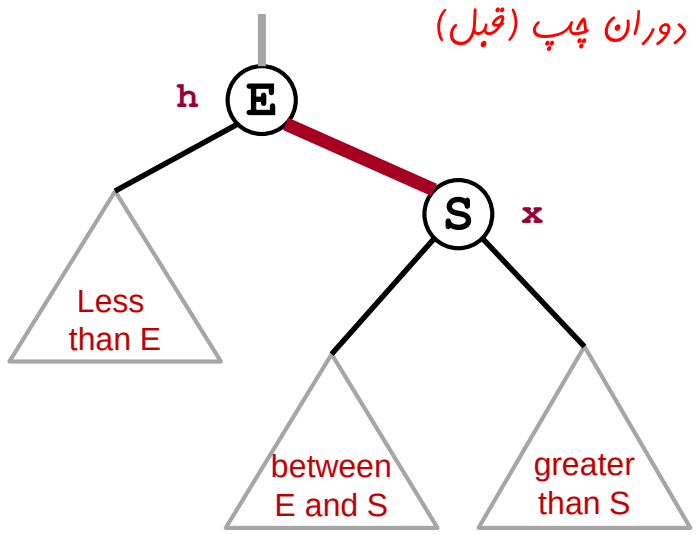
```
private boolean isRed(Node x)
{
   if (x == null) return false;
   return x.color == RED;
}
```

پیوندهای پوچ سیاه هستند



عملیات ابتدایی در درخت‌های قرمز-سیاه

- دوران چپ‌گرد. تغییر پیوند راست مایل به پیوند چپ مایل.

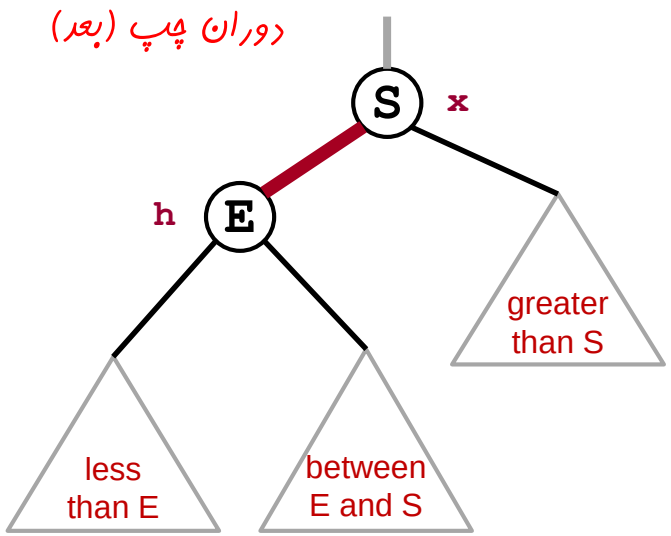



```
private Node rotateLeft (Node h)
{
   assert isRed(h.right);
   Node x  = h.right;
   h.right = x.left;
   x.left  = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

توجه. دوران چپ‌گرد خاصیت جستجو و توازن کامل را حفظ می‌کند.

عملیات ابتدایی در درخت‌های قرمز-سیاه

- دوران چپ‌گرد. تغییر پیوند راست مایل به پیوند چپ مایل.

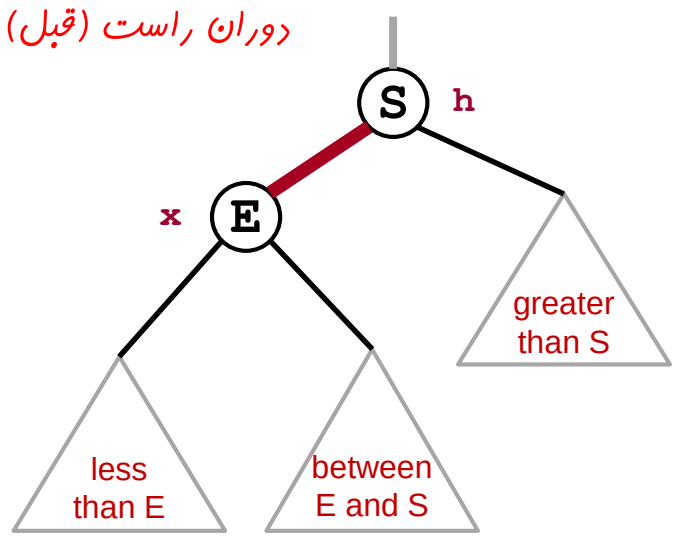



```
private Node rotateLeft (Node h)
{
   assert isRed(h.right);
   Node x  = h.right;
   h.right = x.left;
   x.left  = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

توجه. دوران چپ‌گرد خاصیت جستجو و توازن کامل را حفظ می‌کند.

عملیات ابتدایی در درخت‌های قرمز-سیاه

- دوران راست‌گرد. تغییر پیوند چپ مایل به پیوند راست مایل.

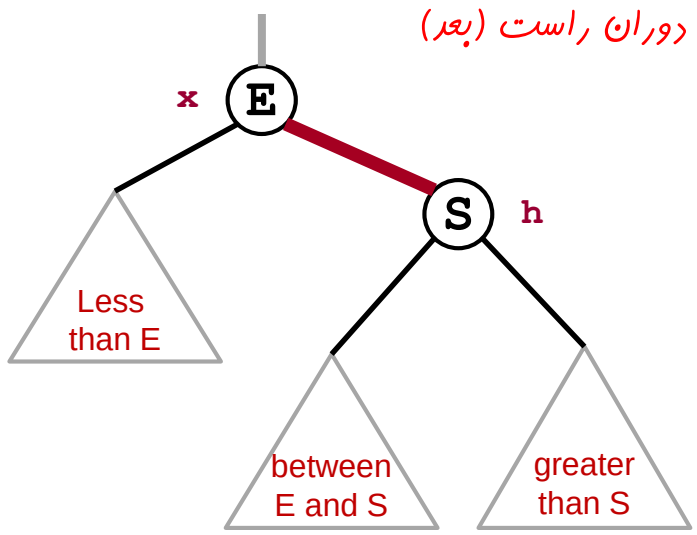



```
private Node rotateRight(Node h)
{
   assert isRed(h.left);
   Node x  = h.left;
   h.left  = x.right;
   x.right = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

توجه. دوران راست‌گرد خاصیت جستجو و توازن کامل را حفظ می‌کند.

عملیات ابتدایی در درخت‌های قرمز-سیاه

- دوران راست‌گرد. تغییر پیوند چپ مایل به پیوند راست مایل.

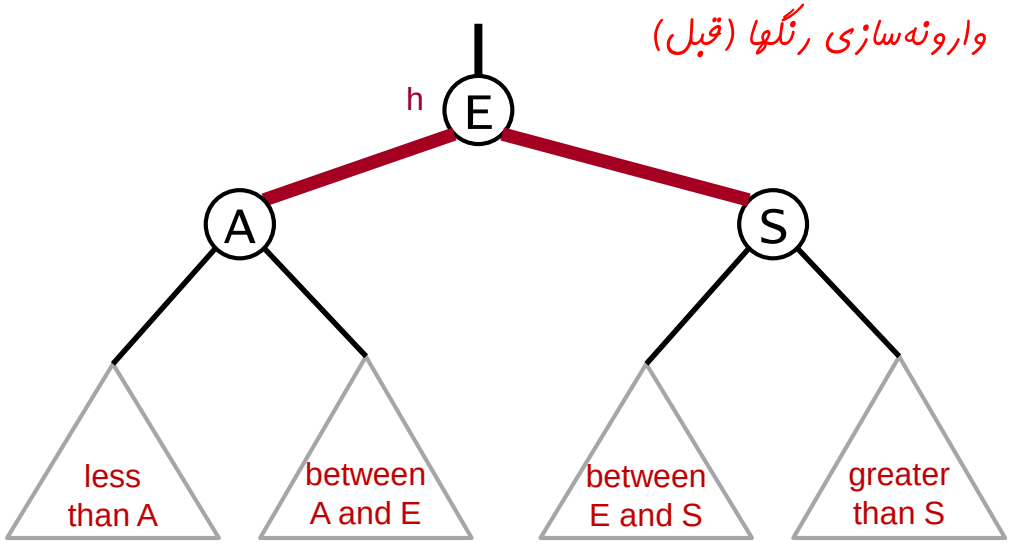



```
private Node rotateRight(Node h)
{
   assert isRed(h.left);
   Node x  = h.left;
   h.left  = x.right;
   x.right = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

توجه. دوران راست‌گرد خاصیت جستجو و توازن کامل را حفظ می‌کند.

عملیات ابتدایی در درخت‌های قرمز-سیاه

- وارونه‌سازی رنگ. به منظور تقسیم یک ۴-گره.

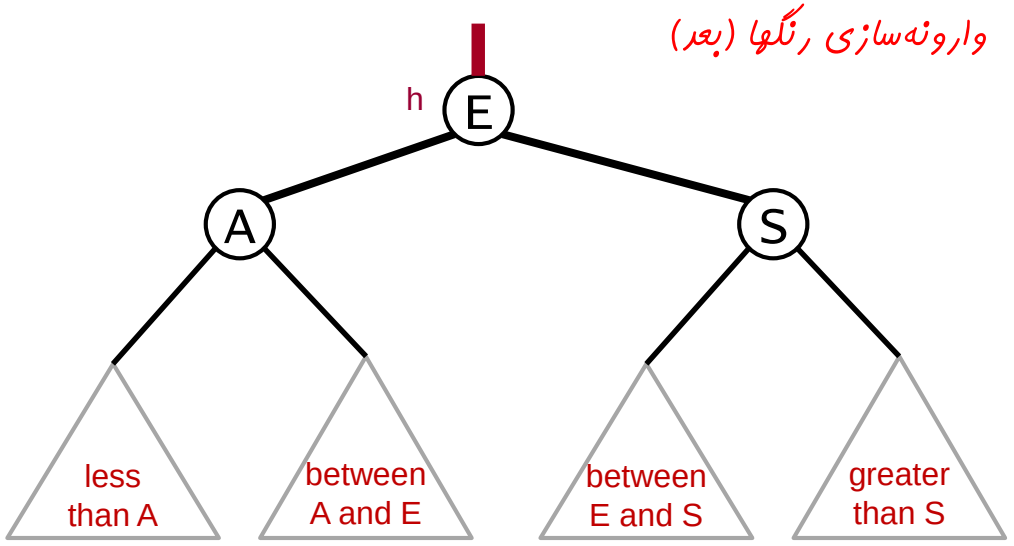



```
private void flipColors(Node h)
{
   assert !isRed(h);
   assert  isRed(h.left);
   assert  isRed(h.right);
   h.color = RED;
   h.left.color  = BLACK;
   h.right.color = BLACK;
}
```

توجه. وارونه‌سازی رنگها خاصیت جستجو و توازن کامل را حفظ می‌کند.

عملیات ابتدایی در درخت‌های قرمز-سیاه

- وارونه‌سازی رنگ. به منظور تقسیم یک ۴-گره.

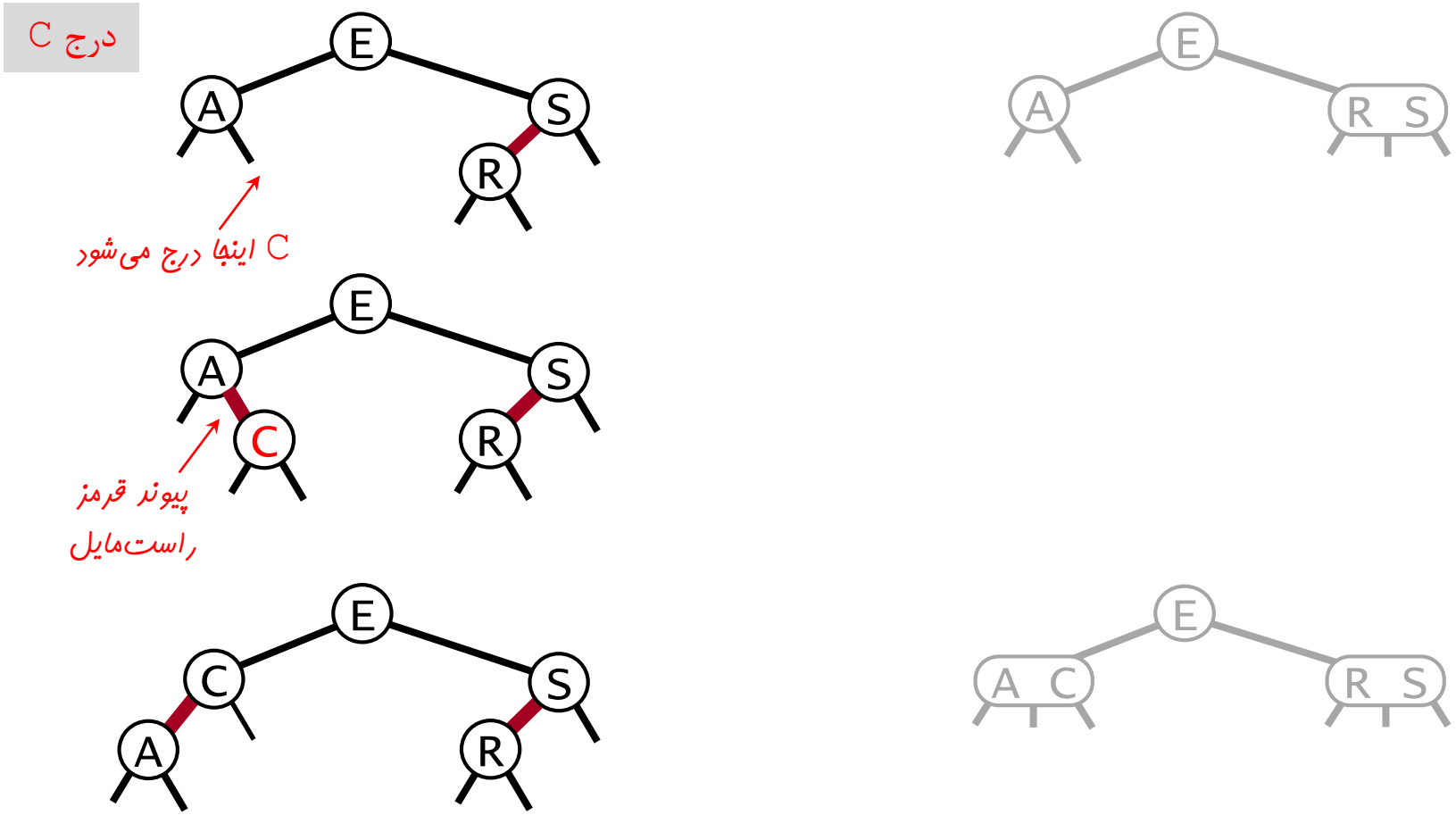



```
private void flipColors(Node h)
{
   assert !isRed(h);
   assert  isRed(h.left);
   assert  isRed(h.right);
   h.color = RED;
   h.left.color  = BLACK;
   h.right.color = BLACK;
}
```

توجه. وارونه‌سازی رنگها خاصیت جستجو و توازن کامل را حفظ می‌کند.

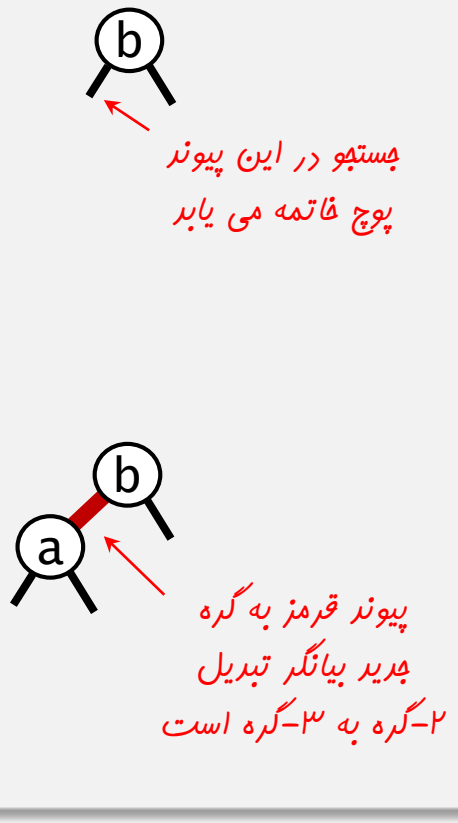
درج در درخت قرمز-سیاه: کلیات

- استراتژی اصلی. حفظ تناظر با درخت‌های ۲-۳ با به کارگیری عملیات ابتدایی.

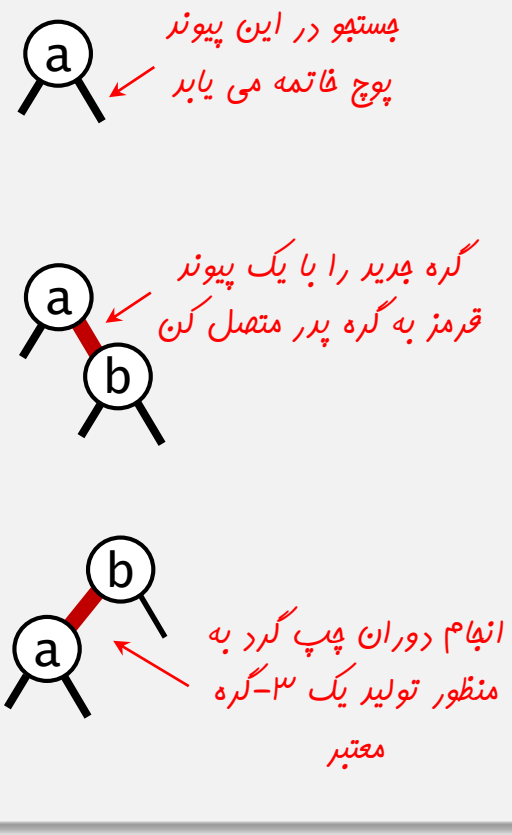


درج در یک درخت تک گره‌ای

(۱) کوچک‌تر

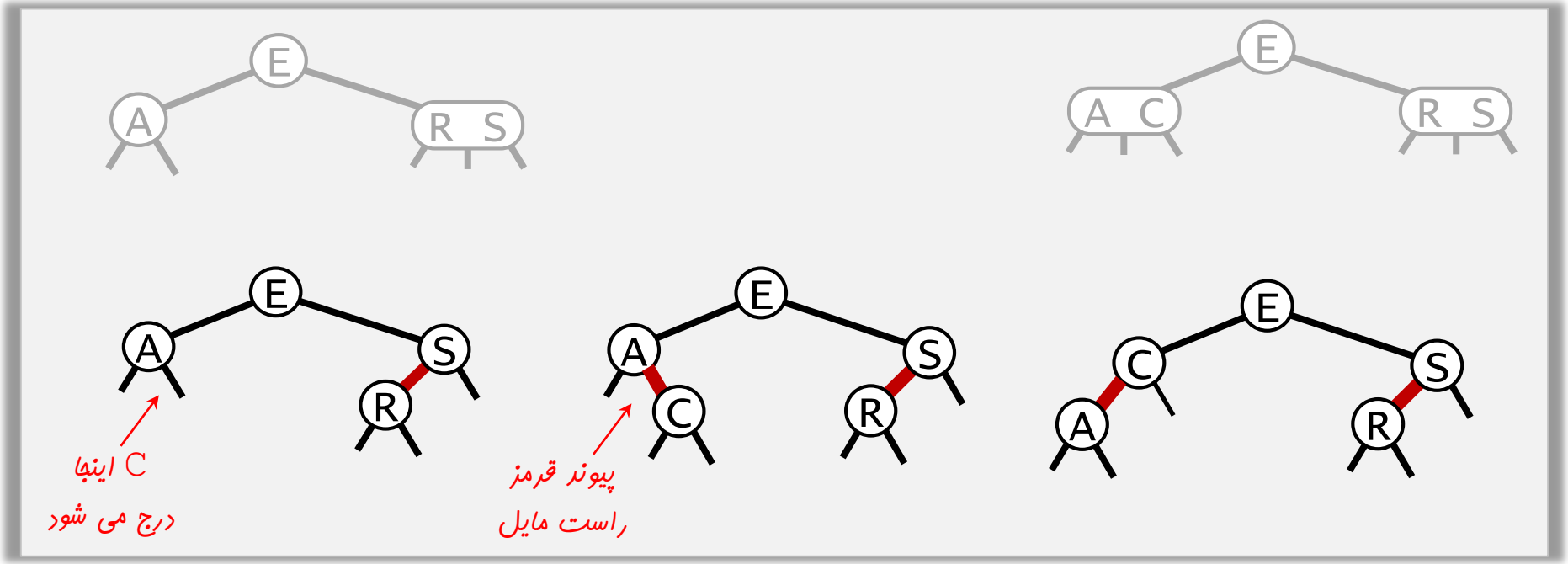


(۲) بزرگ‌تر

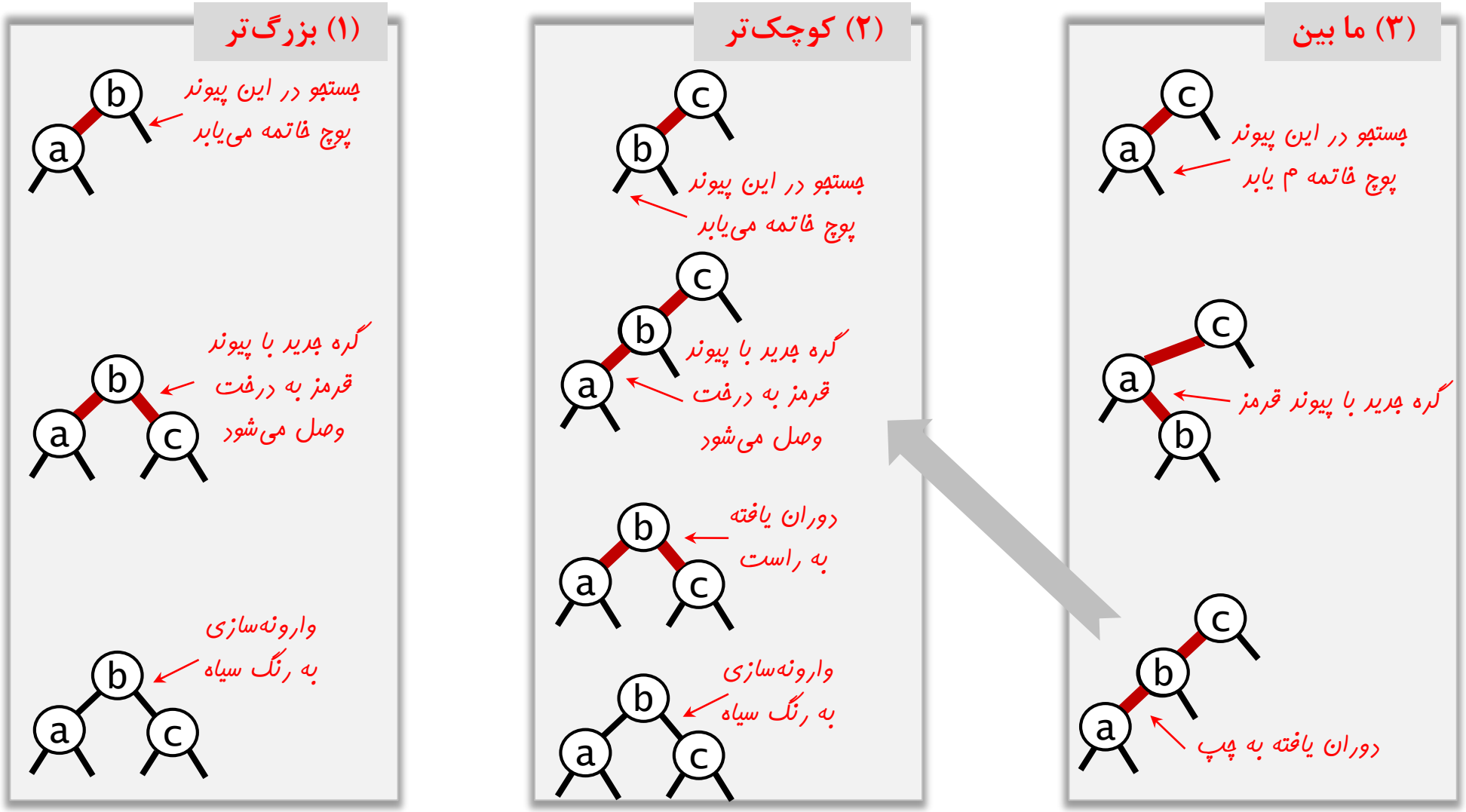


درج در درخت قرمز-سیاه

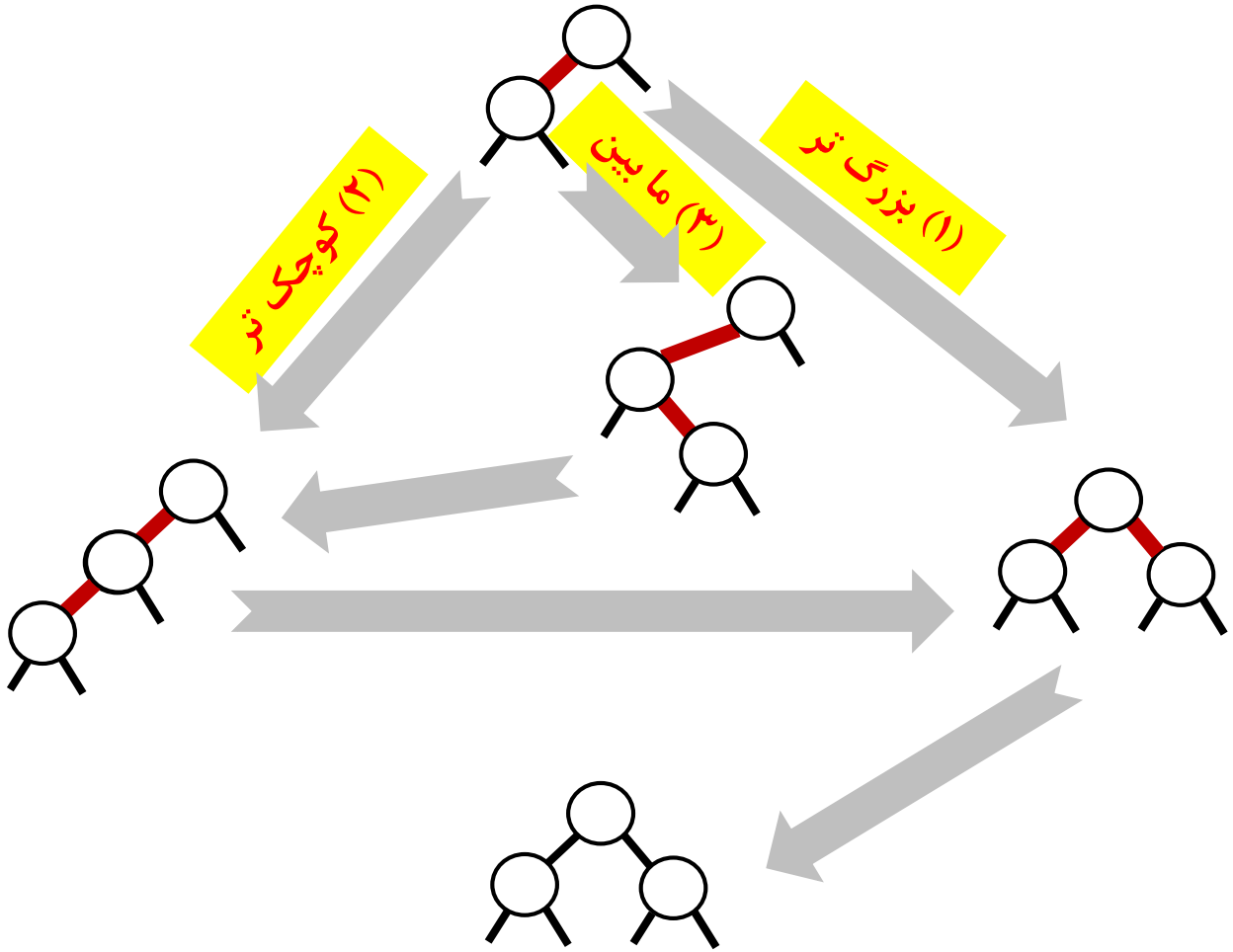
- حالت ۱. درج در یک ۲-گره در پایین درخت.
  - یک درج معمولی در درخت جستجوی دودویی انجام بده؛ پیوند جدید را قرمز کن.
  - اگر پیوند جدید راست مایل است، دوران چپ گرد انجام بده.



درج در یک درخت دو گره‌ای

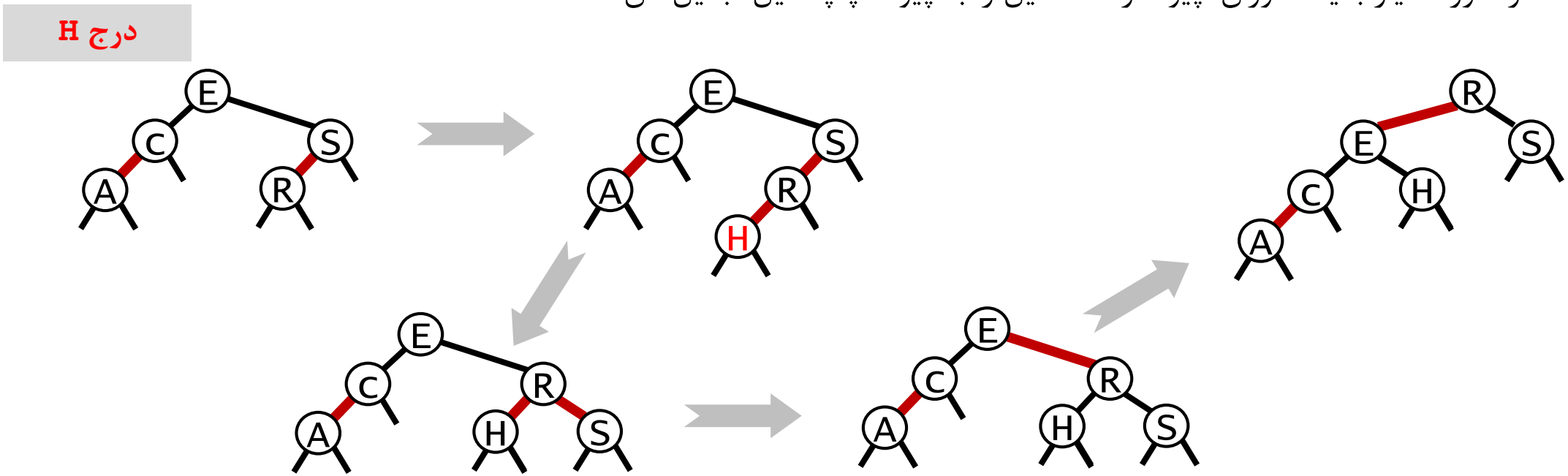


درج در یک درخت دو گره‌ای



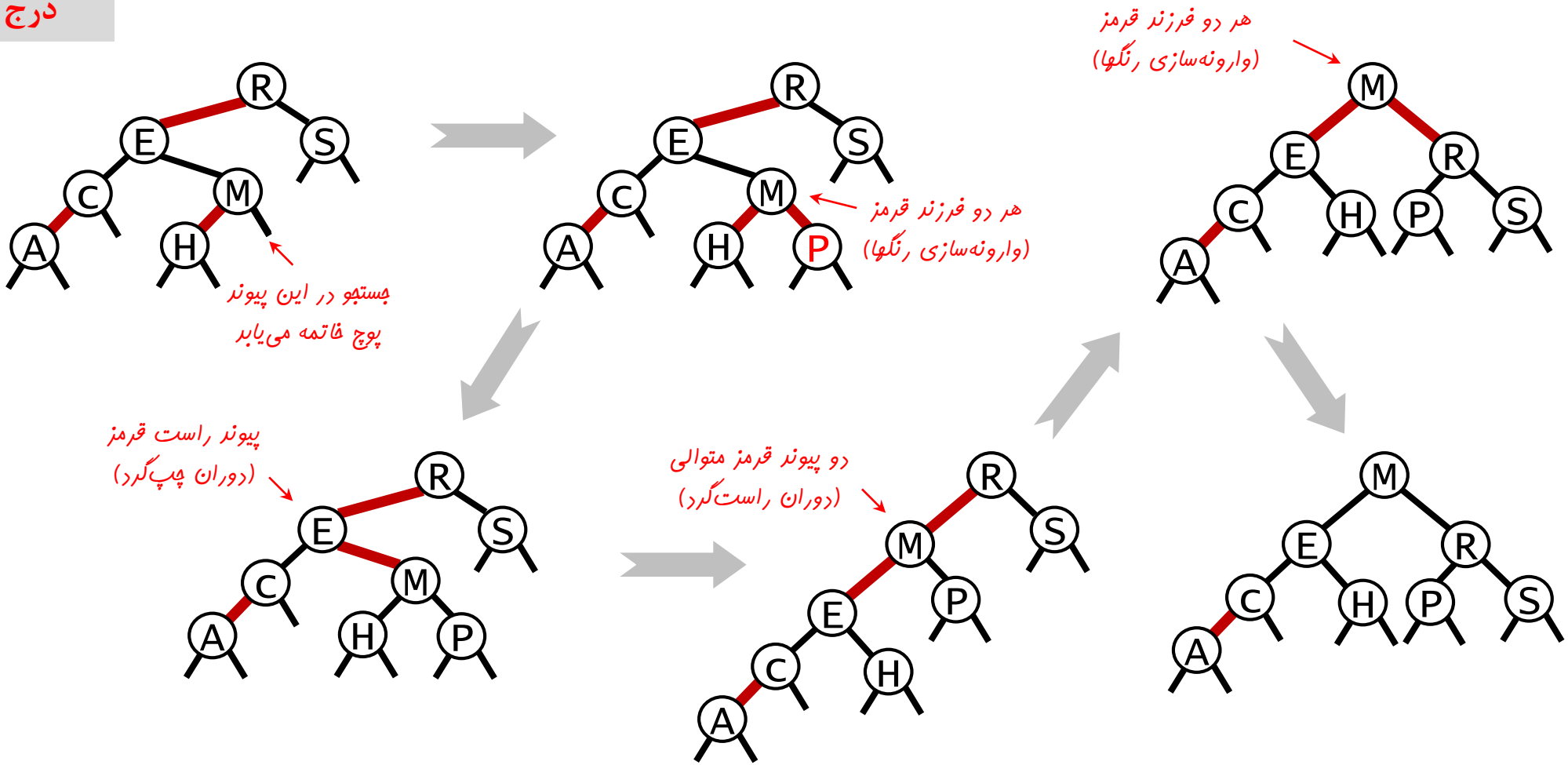
درج در درخت قرمز-سیاه

- حالت ۲. درج در یک ۳-گره در پایین درخت.
  - یک درج معمولی در درخت جستجوی دودویی انجام بده؛ پیوند جدید را قرمز کن.
  - در صورت نیاز ۴-گره ایجاد شده را برای برقراری توازن دوران بده.
  - با وارونه‌سازی رنگها، پیوند قرمز را یک سطح به بالا انتقال بده.
  - در صورت نیاز با یک دوران، پیوند راست مایل را به پیوند چپ مایل تبدیل کن.



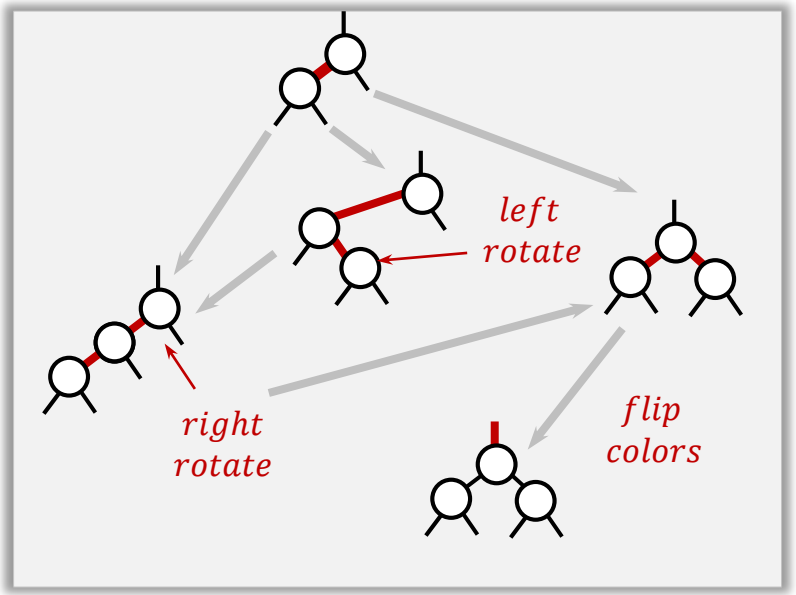
درج در درخت قرمز-سیاه

درج P



درج در درخت قرمز-سیاه (دمو)

درج در درخت قرمز-سیاه: پیاده‌سازی در جاوا

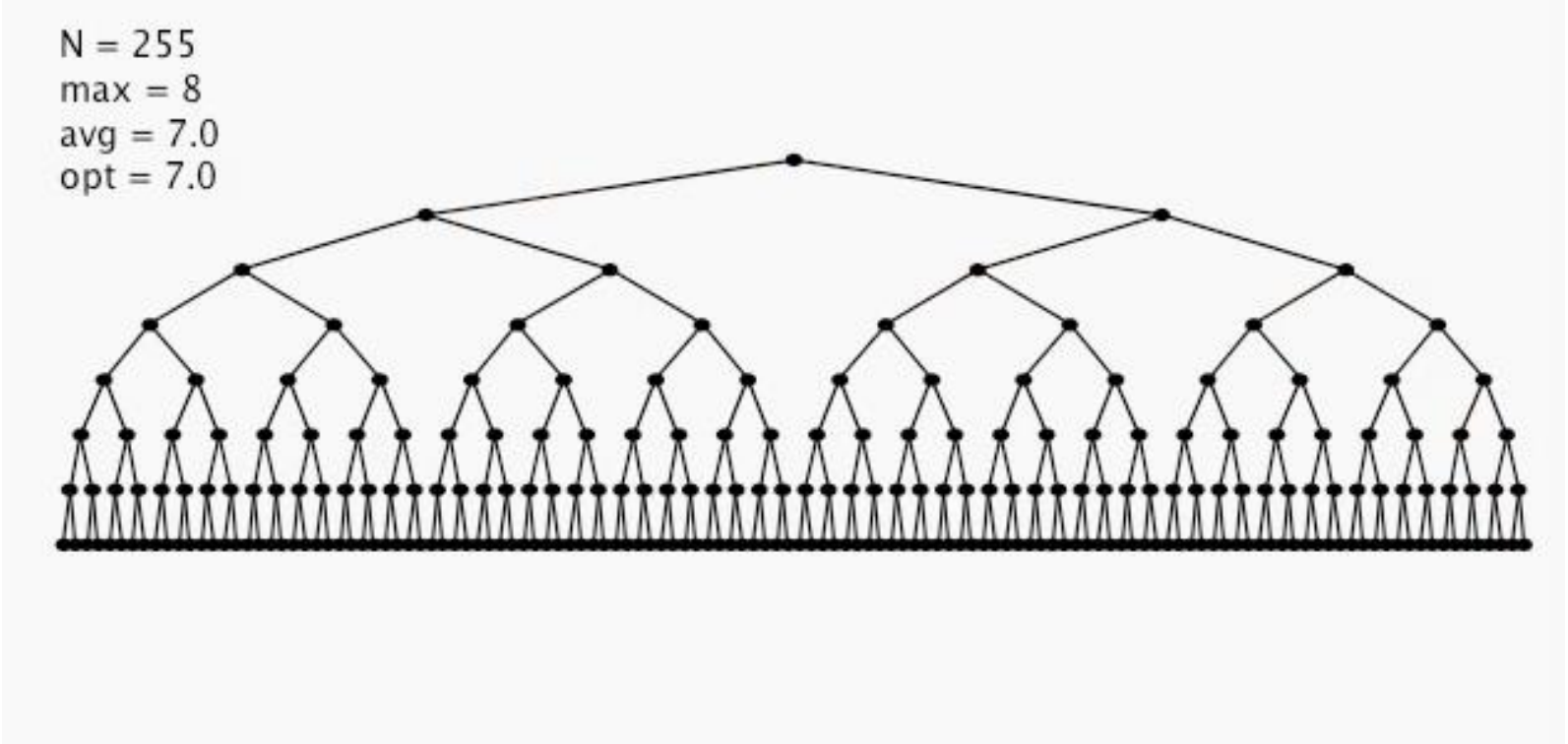



```
private Node put(Node h, Key key, Value value)
{
   if (h == null) return new Node(key, value, RED);

   int cmp = key.compareTo(h.key);
   if      (cmp < 0)  h.left  = put(h.left,  key, value);
   else if (cmp > 0)  h.right = put(h.right, key, value);
   else if (cmp == 0) h.value = value;

   if (isRed(h.right) && !isRed(h.left))      h = rotateLeft(h);
   if (isRed(h.left)  &&  isRed(h.left.left)) h = rotateRight(h);
   if (isRed(h.left)  &&  isRed(h.right))     flipColors(h);

   return h;
}
```

مایل به چپ کردن
متوازن کردن ۴-گره
تقسیم کردن ۴-گره

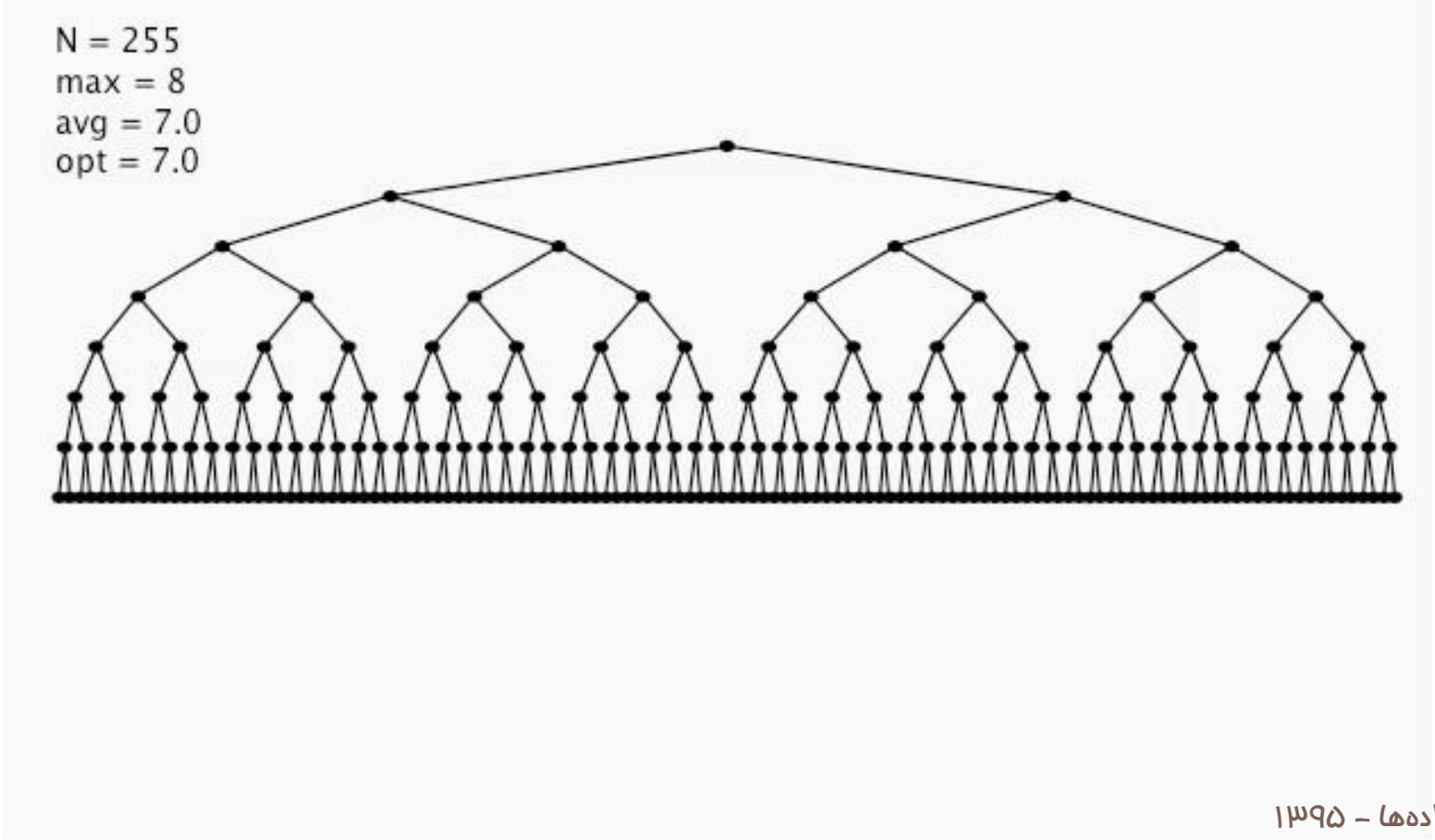
درج در درخت قرمز-سیاه

- درج ۲۵۵ کلید به ترتیب صعودی.



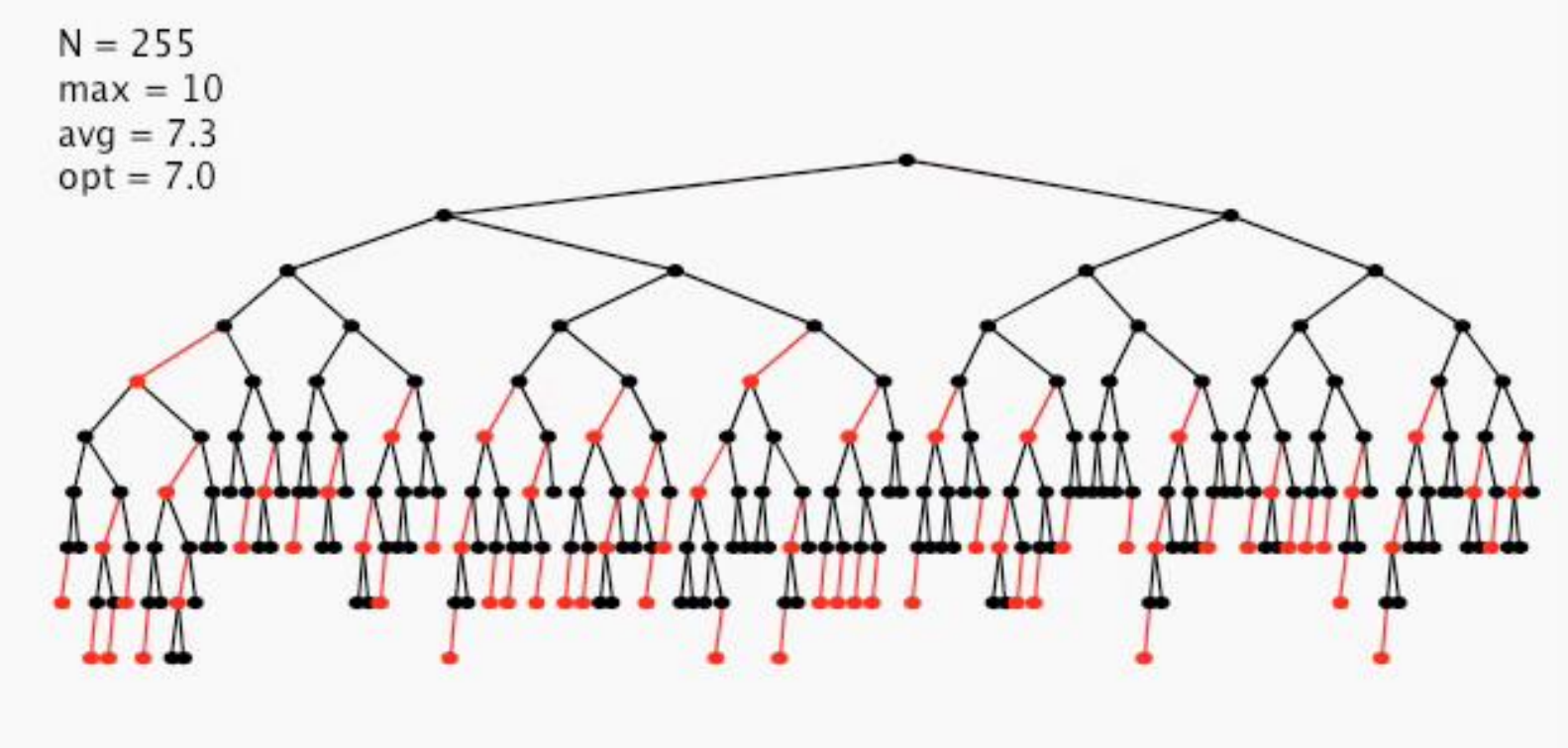
درج در درخت قرمز-سیاه

- درج ۲۵۵ کلید به ترتیب نزولی.



درج در درخت قرمز-سیاه

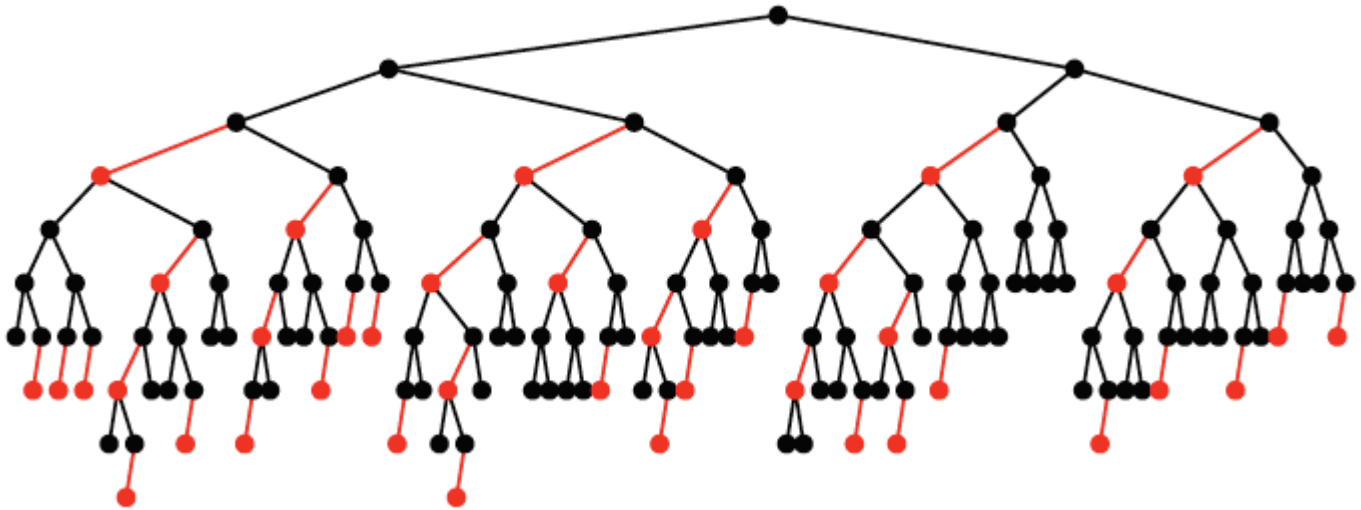
- درج ۲۵۵ کلید به ترتیب تصادفی.



توازن در درخت قرمز-سیاه

- گزاره. ارتفاع یک درخت قرمز سیاه در بدترین حالت برابر با $2\lg N$ است.

- اثبات.
  - در همه‌ی مسیرها از ریشه تا پیوندهای پوچ، تعداد پیوندهای سیاه برابر است.
  - دو پیوند قرمز پشت سرهم مجاز نیست.

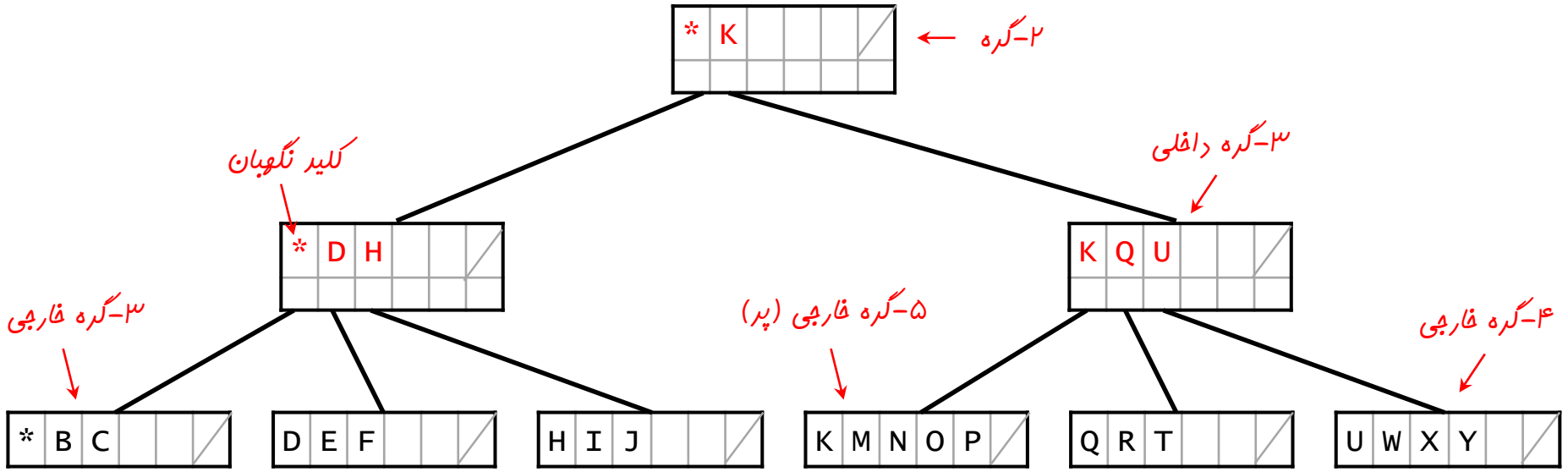


درخت‌های B

درخت‌های B

- درخت **B**: تعمیمی از درخت‌های ۲-۳ با امکان ذخیره‌سازی حداکثر `M - 1` کلید در هر گره.
  - در ریشه حداقل دو کلید وجود دارد.
  - در گره‌های دیگر حداقل `M / 2` کلید وجود دارد.
  - کلیدها در گره‌های خارجی ذخیره می‌شوند.
  - گره‌های داخلی در برگیرنده‌ی کپی‌هایی از کلیدها به منظور هدایت فرآیند جستجو هستند.

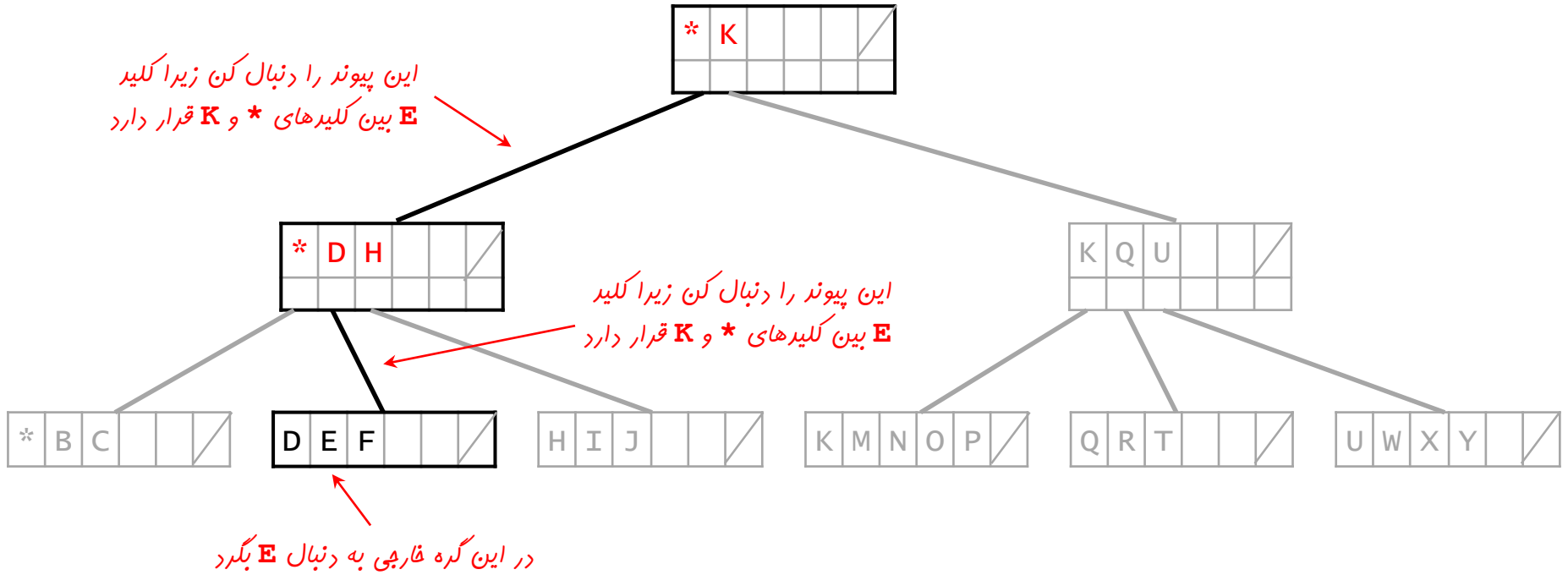
درخت B



آناتومی یک درخت بی (M = 6)

جستجو در درخت B

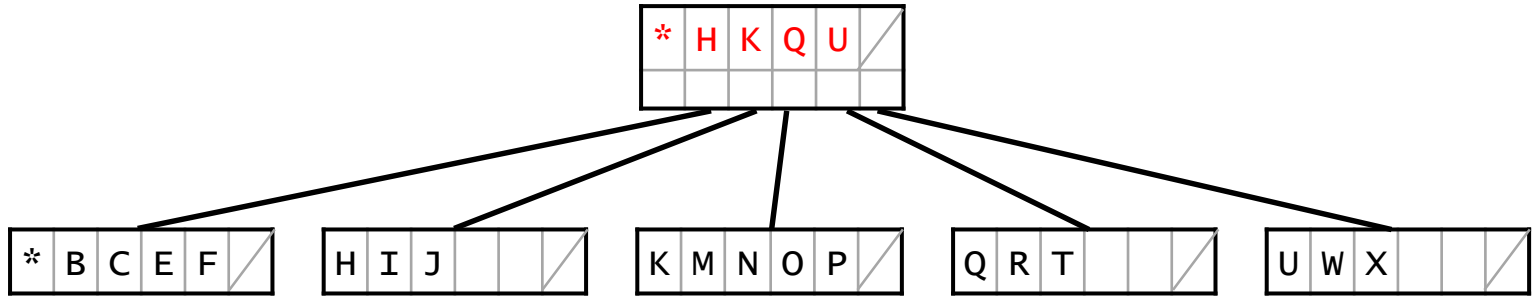
- از ریشه شروع کن.
- بازه‌ی کلید مورد جستجو را پیدا و پیوند مربوط به آن را دنبال کن.
- جستجو در کلیدهای خارجی پایان می‌پذیرد.



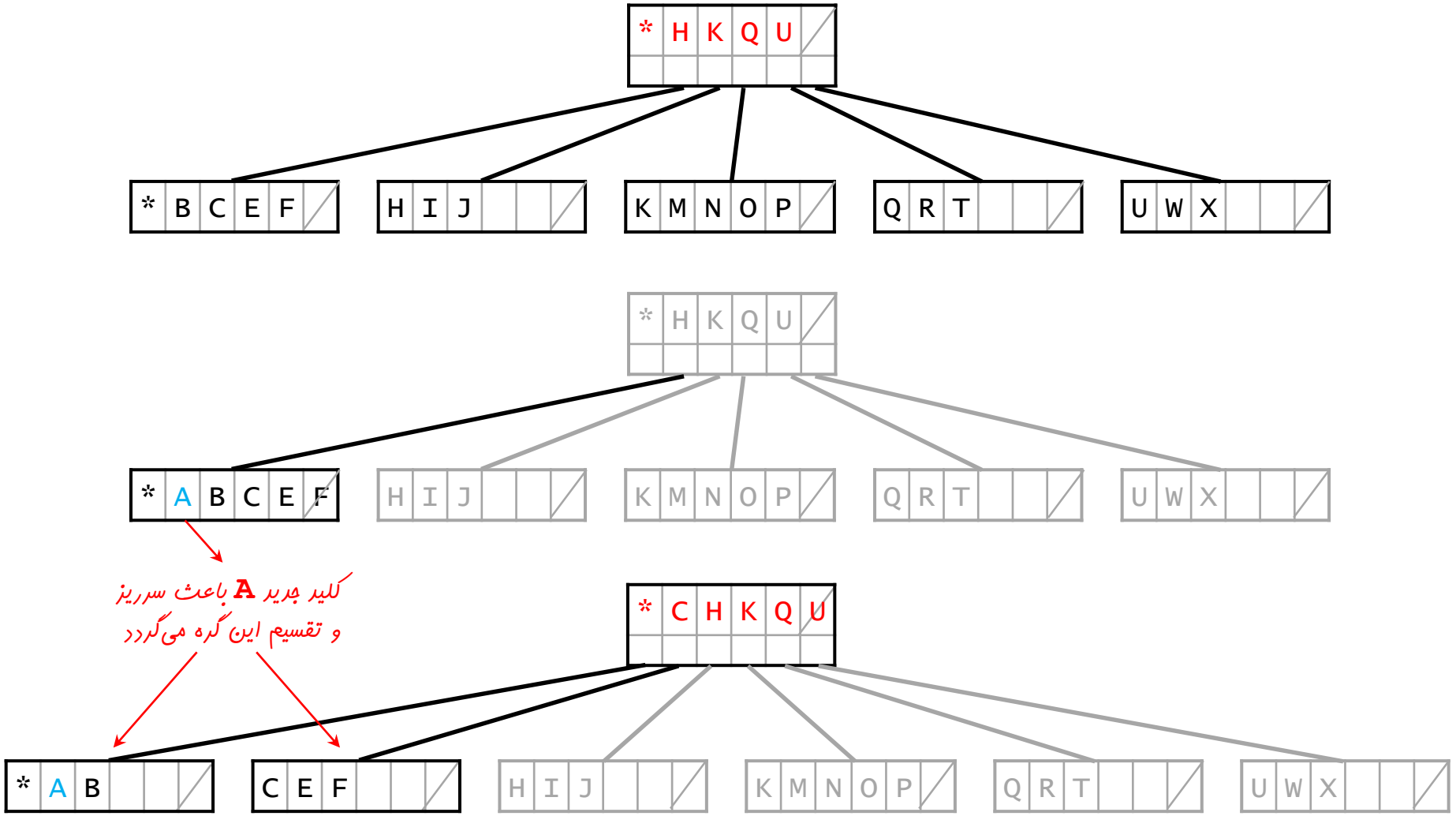
درج در درخت B

- درخت را به منظور یافتن مکان کلید مورد نظر جستجو کن.
- کلید جدید را در مکان یافته شده درج کن.
- گره‌های پر را تا رسیدن به ریشه تقسیم کن.

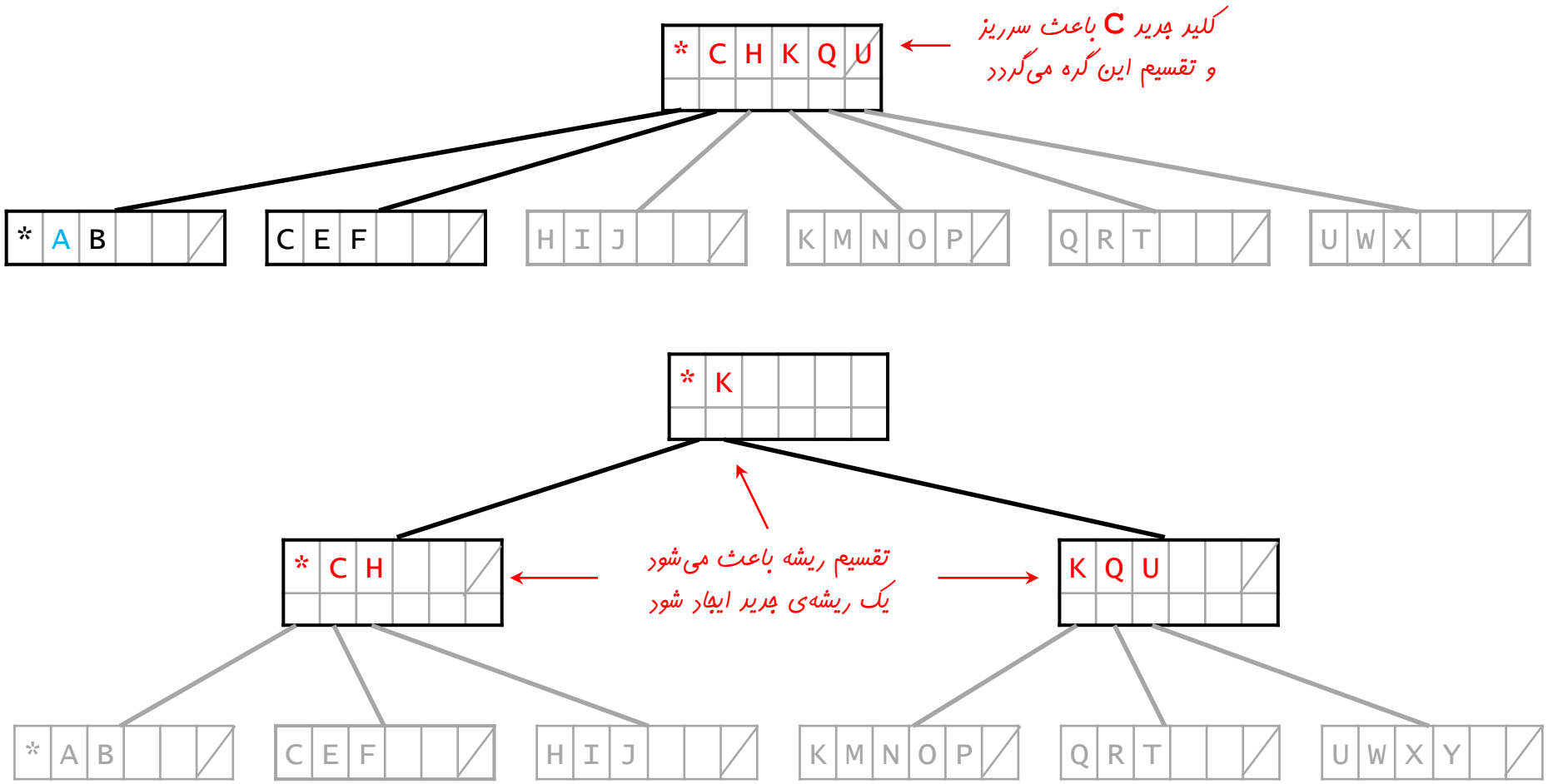
درج کلید A



درج در درخت B


* H K Q U
* B C E F
H I J
K M N O P
Q R T
U W X
* H K Q U
* A B C E F
H I J
K M N O P
Q R T
U W X
کلید جدید A باعث سرریز
و تقسیم این گره می‌گردد
* C H K Q U
* A B
C E F
H I J
K M N O P
Q R T
U W X

درج در درخت B


کلید جدید C باعث سرریز
و تقسیم این گره می‌گردد
* C H K Q U
* A B
C E F
H I J
K M N O P
Q R T
U W X
* K
تقسیم ریشه باعث می‌شود
یک ریشه‌ی جدید ایجاد شود
* C H
K Q U
* A B
C E F
H I J
K M N O P
Q R T
U W X Y

توازن در درخت B

- گزاره. تعداد گره‌های بررسی شده به منظور انجام یک عمل جستجو یا درج در یک درخت B با N کلید و از مرتبه M بین $\log_{M-1} N$ و $\log_{N/2} M$ است.

- اثبات. همه‌ی گره‌های داخلی (به جز ریشه) دارای $M/2$ تا $M-1$ پیوند هستند.

- در عمل: تعداد گره‌های بررسی شده حداکثر برابر با ۴ است. ⟵ `M = 1024, N = 62 billion`

- بهینه‌سازی: همیشه صفحه‌ی مربوط به گره ریشه را در حافظه اصلی نگهداری کن.